ESTUDIO Y PRAXIS DE LA IMAGEN TRIDIMENSIONAL EN LA WEB, PROPUESTA DIDÁCTICA DE NUEVAS HERRAMIENTAS PARA LA FORMACIÓN ARTÍSTICA

TESIS DOCTORAL

Presentada por: Godofredo Folgado de la Rosa

Dirigida por el Dr. Emilio Roselló Tormo

UNIVERSIDAD MIGUEL HERNÁNDEZ

Departamento de Arte

Elche, 2014

UNIVERSITAS
Miguel
Hernández

Departamento de Arte

Daniel Tejero Olivares, director del departamento de Arte de la Universidad Miguel Hernández, conforme a la normativa de esta Universidad, presto la conformidad y autorización necesaria para que el trabajo de investigación de D. Godofredo Folgado de la Rosa con el título Estudio y praxis de la imagen tridimensional en la web: propuesta didáctica de nuevas herramientas para la formación artística, pueda ser presentado, en la Comisión de Doctorado para ser defendido como Tesis Doctoral, con el fin de optar al grado de Doctor.

Elche, a de 2013

Fdo.: Daniel Tejero Olivares
Director

UNIVERSITAS
Miguel
Hernández

Emilio Roselló Tormo, Profesor Doctor del Área de Comunicación Audiovisual y Publicidad del Departamento de Ciencias Sociales y Humanas de la Universidad Miguel Hernández de Elche, conforme a la normativa de la citada Universidad, presto la conformidad y autorización para la presentación de la tesis doctoral titulada, Estudio y praxis de la imagen tridimensional en la web: propuesta didáctica de nuevas herramientas para la formación artística, realizada por D. Godofredo Folgado de la Rosa, bajo mi dirección y supervisión, para la obtención del grado de Doctor por la Universidad Miguel Hernández de Elche.

En Altea, a de 2013

El director de la tesis
Fdo.: Emilio Roselló Tormo

UNIVERSITAS
Miguel
Hernández

Resumen

La presente tesis aborda el tema de las herramientas de software para la generación de trabajos en el ámbito de la creación digital en entornos tridimensionales interactivos, pretendiendo que estas prácticas sean operativas tanto en soporte local como en red. Este trabajo se orienta hacia la metodología y tecnología utilizadas para la adquisición de los conocimientos necesarios con la finalidad de proponerlos como curso de nivel universitario.

En los primeros capítulos se establecen los objetivos de este documento junto con un estudio de situación del problema y un acercamiento a la metodología a utilizar. Lo expuesto a continuación contiene la programación de un curso dividido en tres partes: una primera visión teórica general, y dos apartados prácticos. En el apartado teórico, se explican los conceptos comunes para todos los procesos implicados en cada competencia. Para la implementación práctica primera hemos utilizado dos aplicaciones como son Autodesk 3D Studio MAX y Adobe Director. En la segunda praxis utilizaremos las herramientas propias del software Second Life que se desarrollarán en este metaverso.

Abstract

This thesis investigate the software tools for generate works in the field of digital creation in interactive three-dimensional scope, and also that these practices are operative in both local and Internet environments. This paper is oriented towards technology and methodology to acquire the necessary knowledge in order to propose them as university level course.

In the first part the characteristics of these tools are analyzed to determine its viability considering its potential. It contains programming a course divided into three parts: a first general theoretical view, and two sections practical. In the theoretical section, the common concepts for all processes involved in each competency are explained. For the first practical implementation we used two applications that we have estimated more viable for our targeted purposes such as Autodesk 3D Studio MAX and Adobe Director. In the second practice we use the tools of Second Life software to be developed in this metaverse.

UNIVERSITAS
Miguel
Hernández

A mis padres y hermanos
A mi pasado y mis raíces
A Godofredo y Carmen y a Francisco, Teresa, Jonás y David

A mi mujer
A mi presente y mi tallo
A Cristina

A mis hijos
A mi futuro y mi fruto
A Godofredo y Blanca Luna

Y a mis abuelos Lidia, Clemencia, Paco y Godofredo
Y a los demás familiares del gran árbol al que pertenezco

UNIVERSITAS
Miguel
Hernández

Agradecimientos

La tesis doctoral que presentamos en este documento es el resultado de un trabajo que ha tenido la inestimable colaboración de muchas personas que han contribuido para poder ser concluida. Quisiera dar las más sinceras gracias a personas que desinteresadamente han dedicado su tiempo y esfuerzo y a las que ni siquiera conozco en persona sino solo por Internet pero que han sido imprescindibles para que esta obra llegara a buen puerto.

Mi más sincero agradecimiento a Emilio Roselló Tormo, por su inagotable paciencia y ánimo para concluir lo comenzado. Ojala que el futuro me depare la oportunidad de resarcir mi deuda con él.

UNIVERSITAS
Miguel
Hernández

ÍNDICE

UNIVERSITAS
Miguel
Hernández

CAPÍTULO 1:

PRESENTACIÓN DEL PROBLEMA Y SOLUCIÓN PROPUESTA

Presentación del problema y solución propuesta.

En este capítulo vamos a plantear los objetivos que tiene el presente estudio y la forma prevista de obtenerlos. Por tanto, velaremos en el apartado de conclusiones para que estos objetivos se cumplan. En primer lugar, trataremos de definir el tema expuesto en este documento a partir de unas ideas básicas que están interrelacionadas, al objeto de delimitar el estudio a unos conocimientos específicos y a unos aspectos concretos. Básicamente hay involucrados tres tipos de ámbitos relacionados entre sí: el diseño, la programación y la docencia.

1. Objetivos de la investigación.

El objetivo fundamental es el estudio de algunas herramientas digitales útiles para la creación de trabajos multimedia que pueden ser ejecutados desde disco duro y/o en la red y que permiten la interacción con el usuario. El desarrollo de la investigación nos ha llevado a resolver que la mejor forma de plasmar estos conocimientos era la planificación de un curso de nivel universitario. Por tanto, la consecuencia inmediata de esta tesis será la realización de un curso base, entendiendo por ello, que no se concreta para un determinado año académico ni sus contenidos se pueden cuantificar directamente en créditos. Este curso básico se puede tomar como punto de partida para la realización de experiencias docentes de distinto tipo ya que está estructurado en módulos. Los cursos reales que se pudieran derivar de él deberán estar compuestos por módulos más elaborados y actualizados de los que presentamos ahora. Dado que el objetivo de este trabajo es aportar una visión general del tema, tenemos especial cuidado en que el resultado sea suficientemente profundo, haciendo hincapié sobre todo en unas realizaciones prácticas concretas y específicas. Además se plantean sugerencias a los “lectores” para la experimentación tomando estas prácticas como base.

El objetivo último de este trabajo estaría conseguido si un alumno adquiriera la capacidad de representar la imagen tridimensional, subirla a la red e interactuar con ella.

2. Hipótesis de trabajo y metodología.

La hipótesis de trabajo supone que los objetivos a alcanzar son realistas y que su consecución no es una quimera. Esta hipótesis se basa en tres puntos fundamentales: las herramientas software utilizadas, nuestros conocimientos y experiencia previas y el perfil del estudiante.

Por una parte, los programas a estudiar están muy difundidos y probados en el mercado actual de la creación multimedia. Como desarrollaremos más adelante, son programas avalados por años de pruebas y trabajos profesionales en distintas áreas del medio.

En segundo lugar, nuestros conocimientos previos en el área de la creación multimedia enfocadas a la educación se plasman en la realización de diez publicaciones en CD-ROM y tres publicaciones más de carácter científico sobre programación y culminadas con el “Sello Europeo 1998 otorgado por el Ministerio de Educación y Cultura y la comisión Europea para la enseñanza de lenguas extranjeras”. Por tanto, el grueso del trabajo consistirá en una orientación, estructuración y creación de contenidos basados en principios didácticos que nos llevarán a la realización de los objetivos planteados.

Damos por hecho que los alumnos a los cuales va dirigido este trabajo tienen un perfil bien definido como estudiantes universitarios en el campo del arte o de la comunicación audiovisual, siendo estas carreras vocacionales y que requieren el uso de las nuevas tecnologías en su perfil curricular.

La metodología utiliza en la medida de lo posible las pautas y parámetros de las asignaturas que se imparten en los estudios superiores tanto en nuestro país como en estudios de un nivel similar en el extranjero e incluso en academias de educación no reglada. Con esta información hemos generado un curso que puede ser el germen de uno o varios cursos de nivel universitario. Por tanto, hemos seguido los procedimientos en cuanto a la programación

didáctica que se aplican en nuestras universidades y también hemos tenido en cuenta los medios básicos habituales en estos centros educativos.

Este curso está estructurado de forma modular. Cada módulo es independiente pues tiene unos objetivos didácticos muy específicos. Esto quiere decir que se podrían combinar diversos módulos para la composición de cursos distintos, enfocados a la consecución de objetivos varios y que suelen ser dotados de su correspondiente titulación a modo de los que programan algunas academias de educación no reglada. El conjunto de cursos supondría, por tanto, la consecución de una serie de habilidades que son las necesarias para realizar infografía 3D interactiva, es decir, imágenes generadas por ordenador, imágenes de síntesis o como dicen en el mundo anglosajón CG 3D (computer graphics 3 dimensional). Esta forma modular es como trabajan la mayoría de las universidades y academias que imparten esta clase de conocimientos en el extranjero. Sobre todo en estas últimas han demostrado durante mucho tiempo que es un sistema muy productivo.

En la mayoría de los cursos revisados, el conocimiento de lo que llamamos infografía tridimensional interactiva lo podemos dividir en 8 módulos: modelado, materiales, iluminación, cámaras, animación, composición, programación y formatos de salida.

Las distintas combinaciones de estos 8 módulos, dan lugar a titulaciones como:

a) Diseño CG[1]: modelado, materiales, iluminación y formatos de salida.

b) Imagen virtual: modelado, materiales, iluminación, cámaras y formato de salida.

c) CGRT[2]: modelado, materiales, iluminación, cámaras, animación y programación.

En esta tesis desarrollaremos las teorías necesarias en cada módulo y su resolución práctica. Al separar estas dos vertientes, pretendemos desvincular la parte mas perecedera de este conocimiento que tiene que ver con la práctica (ya que depende de un software en particular y por tanto de una tecnología en constante evolución) de la parte invariante que será la parte teórica. Esto significa que aunque la “forma de hacer” cambie o se realice con programas más fáciles de utilizar, más rápidos, etc., “lo que se hace” se basa en unos conocimientos mucho más estables. Por ejemplo: la teoría de la “cinemática inversa” se basa en conocimientos matemáticos desarrollados hace décadas, y solo alrededor de 1995 están integrados en los programas 3D más difundidos.

En nuestro caso, entre todas las posibles combinaciones de módulos hemos elegido la que se corresponde con el título de “Arte en CGRT”, ya que se adapta totalmente a los objetivos de esta investigación. Además se estudian casi todos los módulos. Lo característico de este curso es su carácter integral ya que se deben controlar suficientes materias y los pasos para llegar a un resultado final satisfactorio deben estar diseñados desde el principio.

3. Fuentes.

Las fuentes utilizadas en esta investigación son fundamentalmente de tipo digital. Para el estudio de la situación general de estas tecnologías se consultaron páginas web de distintas tipologías. Exponiendo opiniones y requiriendo comentarios en blogs, foros, etc. Hemos de subrayar que teníamos la necesidad de testear en Internet los trabajos que se generan en el curso como resultado de la investigación, tanto para hacer pruebas como para realizar estudios de tiempos de interacción, etc. Por ello, construimos y publicamos distintos sitios web. Entre otros, hemos venido utilizando un blog que ha servido para estar en un contacto más directo con personas que realizan o pretender producir materiales del tipo que nos interesan, así como para dar y recibir información pertinente al tema. Estos sitios en la red han sido eliminados una vez cumplida su función. Se hicieron además, consultas vía e-mail con varios directivos de academias privadas y profesores de universidades, sobre todo de nuestro entorno. Fueron

[1] CG: Computing Graphics: Gráficos por ordenador

[2] CGRT: Computing Graphics Real Time: Gráficos por ordenador en tiempo real.

consultadas diversas bases de datos vía Internet y sobre todo se realizaron muchos requerimientos a sitios webs que contienen material didáctico, tutoriales, wikis y demás utilidades sobre los temas tratados en este estudio. Por supuesto, fueron estudiados libros sobre el tema, sin embargo, hemos de apuntar que salvo en algunas excepciones, lo publicado de forma digital está mucho más desarrollado y actualizado que lo publicado en formato papel.

4. Estructura de la tesis.

En esta tesis pretendemos realizar nuestra aportación personal de carácter metodológico y conocimiento tecnológico diferente de los típicos manuales o tutoriales al uso en este tipo de estudios. Para ello hemos dividido esta tesis en dos bloques.

La primera parte contiene una exposición de un estudio de situación del tema que nos ocupa. A partir de la introducción, este estudio se encuentra separado en cuatro apartados relacionados para clarificar las conclusiones de mayor utilidad. En primer lugar analizamos en las bases de datos de tesis doctorales lo investigado hasta el momento. En el segundo apartado estudiamos las ofertas y programas educativos de diversas empresas e instituciones docentes. En el tercero se estudian las diversas alternativas que nos ofrecen las tecnologías a través del software para las prácticas, y terminamos mostrando una serie de eventos y autores que utilizan la imagen tridimensional e interactiva como herramientas de producción multimedia.

La segunda parte se concretará en la planificación y programación de un curso modelo capaz de probar nuestra hipótesis. Este curso se planifica desde un apartado de introducción teórica y dos implementaciones prácticas con enfoques distintos y paralelos.

Para concluir este apartado vamos a tratar de sintetizar todo lo anterior en una sola cuestión que podemos formular como sigue: ¿Es posible la programación de un curso que trate sobre la creación multimedia interactiva para Internet del nivel de estudios superiores con los recursos técnicos, humanos y económicos de la universidad española actual?

CAPÍTULO 2:

ESTUDIO DE SITUACIÓN

Introducción al estudio de situación.

En este punto se trata de observar la realidad que tenemos a nuestro alrededor. Mirar y buscar lo que nos ocupa, especialmente en Internet. Esta búsqueda se concreta en tres conceptos:

1. Materia de estudio: la creación electrónica o producciones multimedia.

2. Forma de estudio: la didáctica, es la forma educativa la que nos importa.

3. Ámbito del estudio: la infografía tridimensional.

Estos tres conceptos interrelacionados permiten preguntarnos cuestiones como: ¿Existen tesis similares al presente tema? ¿Dónde se imparten este tipo de cursos? ¿En qué tecnologías están basadas?

1. Estudio de situación de este tipo de investigaciones: tesis.

Al ser esta una investigación de tesis, es decir, un estudio escrito de una investigación original sobre una determinada cuestión, necesitamos enfocar la atención del estudio a los trabajos que coincidan en el mismo tema y del mismo nivel académico.

Como todos sabemos, hay dos modos de buscar información en Internet: los directorios y los buscadores tipo arañas (spiders). Los buscadores de tipo directorio se basan en la clasificación humana de las páginas según su temática general. El caso más famoso es Yahoo (www.yahoo.com). El modo de trabajo de los directorios no es el más adecuado para llegar a encontrar los documentos que buscamos.

Las arañas tienen su principal exponente en Google (www.google.com). Buscan en sus bases de datos, utilizando como criterio una o unas palabras. Además, mediante búsquedas avanzadas podemos limitar los parámetros a utilizar en la consulta como fecha de actualización, formatos, etc.

Por último los metabuscadores, o sea, buscadores que buscan en buscadores, como metacrawler (www.metacrawler.com), suelen aportar también, mucha información útil.

Por ejemplo este es el resultado de una serie de búsquedas en Google en enero de 2012 de palabras como curso o infografía

Palabras a buscar	nº de páginas
interactivo	25.200.000
infografía	4.950.000
infografía interactiva	342.000
infografía interactiva 3D	123.000
infografía interactiva 3D tesis	17.000

Se puede apreciar que cuantas más palabras concretas requerimos de lo que queremos encontrar y menos generales, el número de páginas que concuerdan se reducen drásticamente. Pero no es bastante, porque el número todavía es muy grande y el resultado impreciso. Por eso realizamos búsquedas en bases de datos específicas.

Para poder tener datos fiables sobre estudios de tesis, hemos comenzado por consultar la base de datos Teseo. Esta es la base de datos oficial del Ministerio de Educación español que debe contener todas las tesis doctorales publicadas en España y su dirección es http://www.educacion.es/teseo.

En enero de 2012 comencé utilizando las palabras claves anteriores. Luego estuve buscando “a mano” introduciendo “arte”, “diseño”, informática”, y otras palabras relacionadas. Los resultados son bastante escasos y se puede deducir que el tema está bastante poco

desarrollado. De todas formas, existen algunos trabajos que tienen relación, bien es verdad que, tangencialmente con este trabajo de investigación. La lista que sigue son las tesis que creo tienen una mayor cercanía con el estudio que proponemos para su desarrollo:

Título:	**REALIZACION INFOGRAFICA. TECNICAS Y PROCESOS EN LA ANIMACION DE LA IMAGEN VIRTUAL 3D**
Autor:	ROSELLÓ TORMO EMILIO
Año Académico:	1994
Universidad:	POLITÉCNICA DE VALENCIA
Centro de Lectura:	BELLAS ARTES
Centro Realización:	DEPARTAMENTO: PINTURA PROGRAMA DE DOCTORADO: PLASTICA CONTEMPORANEA
Director:	JOSÉ SABORIT VIGUER

Título:	**EL DISEÑO GRAFICO Y LOS ENTORNOS INFORMATICOS**
Autor:	ESPI CERDA EMILIO
Año Académico:	1994
Universidad:	POLITECNICA DE VALENCIA
Centro de Lectura:	BELLAS ARTES
Centro Realización:	DEPARTAMENTO: DIBUJO PROGRAMA DE DOCTORADO: APROXIMACIONES AL DISEÑO INDUSTRIAL Y GRAFICO
Director:	LECUONA LOPEZ MANUEL

Título:	**LA INFLUENCIA DE LAS NUEVAS TECNOLOGIAS EN DISEÑO GRAFICO. INTERACTIVIDAD: UN NUEVO DESAFIO.**
Autor:	ARANA SAN SEBASTIAN ANTONIO
Año Académico:	1998
Universidad:	PAIS VASCO
Centro de Lectura:	BELLAS ARTES
Departamento:	DIBUJO
Programa Doctorado:	DIBUJO: EXPERIMENTACION Y PROYECTO. BIENIO 1990/92
Director:	HERRERA FERNANDEZ EDURDO

Título:	**ARTE Y ORDENADOR: INFLUENCIA DE LA COMPOSICION VISUAL EN LA PERCEPCIÓN DE LA INFORMACION**
Autor:	MILLAN PAREDES SARA ELENA
Año Académico:	2000
Universidad:	SEVILLA
Centro de Lectura:	BELLAS ARTES
Departamento:	DIBUJO
Programa Doctorado:	ARTES I
Centro Realización:	FACULTAD DE BELLAS ARTES DE SEVILLA
Director:	JIMENEZ SEGURA JESUS

Título:	**NUEVAS FORMAS DE COMUNICACIÓN MULTIMEDIA EN LA RED: ANÁLISIS Y PRODUCCIÓN DE ESPACIOS DE ENTRETENIMIENTO ONLINE**
Autor:	CANET CENTELLAS FERNANDO
Año Académico:	2001
Universidad:	POLITECNICA DE VALENCIA
Centro de Lectura:	BELLAS ARTES
Departamento:	COMUNICACIÓN AUDIOVISUAL, DOCUMENTACIÓN E HISTORIA DEL ARTE
Programa Doctorado:	COMUNICACIÓN AUDIOVISUAL
Centro Realización:	FACULTAD DE BELLAS ARTES
Director:	ADELANTADO MATEU EULALIA

Título:	**MODELADO Y DISEÑO DE EXPERIENCIAS EDUCATIVAS EN LA WORLD WIDE WEB**
Autor:	GARCÍA RUEDA JOSÉ JESÚS
Año Académico:	2001
Universidad:	POLITECNICA DE MADRID
Centro de Lectura:	INGENIEROS DE TELECOMUNICACIÓN
Departamento:	INGENIERÍA DE SISTEMAS TELEMÁTICOS
Programa Doctorado:	INGENIERÍA DE SISTEMAS TELEMÁTICOS
Centro Realización:	E.T.S. INGENIEROS DE TELECOMUNICACIÓN
Director:	SÁEZ VACAS FERNANDO

2. Estudio de la situación actual de estas enseñanzas.

Para desarrollar este punto he revisado por Internet los planes de estudio de distintas universidades y escuelas de arte con un criterio geográfico.

Para no centrarme sólo en la parte más académica he ampliado el trabajo a escuelas de gran prestigio que llenan el hueco existente en la formación oficial y que están aportando muchos autores a los proyectos más interesantes e innovadores. También he revisado las salidas profesionales que podría tener esta clase de estudios.

2.1. Situación en estudios reglados superiores y facultades de bellas artes.

Para comenzar, en España, como todos sabemos, los planes de estudios universitarios han sufrido un importante cambio en los últimos años al pasar de licenciaturas a grados todavía en fase de implantación, por ello nos centraremos en los cursos de postgrado.

Cabe destacar los cursos de especialización postgrado o master no oficiales que en distintas disciplinas han sido desarrollados. Un ejemplo es el master de creación de videojuegos de la Universidad Pompeu Fabra y que está dirigido a licenciados en Bellas Artes o Comunicación Audiovisual para especialización en diseño y a ingenieros o licenciados en Informática para la especialidad de programación.

Mas información en: (http://creaciodigital.upf.edu/web/es/master_creacion_videojuegos.php)

Otro master muy parecido lo realiza desde hace unos años la Universidad Complutense de Madrid (http://www.fdi.ucm.es/juegos3d/) pero está dirigido a la parte más técnica, la de programación, y por ello solo admiten ingenieros en software. La facultad implicada es la de informática.

Cuando se utilizan buscadores y se introducen términos como “bellas Artes” “interacción” “modelado 3D” tenemos aún pocos resultados positivos aunque cada vez son más numerosos. Por ejemplo en la Facultad de Bellas Artes de Granada Alfonso Cano, podemos estudiar “Modelado 3D, animación e interacción por ordenador”.

En Latinoamérica la cuestión no está mucho mejor, aunque encontramos cursos con algunos años en vigor pero que pasamos a detallar por que tienen similitudes con los estudios que se proponen en este trabajo aunque tampoco es imparten desde las Bellas Artes.

En la Universidad Nacional Autónoma de México (UNAM), se imparte el curso “Integración e Interacción de Ambientes 3D en Multimedia ó Web”

Profesores:	Daniel González Lorenzo Emmanuel Rajón González DGSCA; UNAM
Descripción:	Objetivo y metas: El participante trabajará con las herramientas necesarias para cumplir con el proceso de desarrollo de ambientes tridimensionales que le permitan crear escenas 3D, asignarles propiedades de interacción y publicarlas en un sistema multimedia o página electrónica. Campo: Integrantes de un equipo de desarrollo multimedia interesados en el proceso necesario para el diseño y animación de ambientes 3D, así como su integración y programación en sistemas Multimedia. Alcance: Tener una visión del proceso y las ventajas que tenemos al integrar objetos o ambientes 3D en sistemas multimedia, donde el usuario tiene posibilidades de interacción con dichos objetos.
Temario:	1. Introducción 1.1 Presentación: Una visión de las diferentes herramientas que existen para integrar Escenas 3D en sistemas Multimedia. 1.2 Presentación de las herramientas 3DMAX y Cult 3D: Justificación del uso de estas dos herramientas y la forma en que interactúan. 2. Exportación de una escena 3D desde 3DMAX: Preparar el objeto 3D para su creación y exportación a Cult 3D 3. Presentación de la interfaz gráfica del programa Cult 3D: Que el usuario se familiarice con el ambiente de trabajo de Cult 3D 4. Integración e Interacción de una escena 3D: Asignación de la interacción que tendrá esta escena con el usuario final 5. Publicación de una escena 3D en un sistema Multimedia o página Web: Integración de la escena 3D en un sistema multimedia o página electrónica.

Tabla 1: Descripción de una asignatura de la UNAM

Más información en http://www.dgsca.unam.mx/

En las universidades anglosajonas y europeas si se imparten esta clase de conocimientos y llevan a cabo cursos específicos enfocados a estudiantes de perfil artístico. De todas formas, las “Arts Schools” visitadas no suelen vincular necesariamente la realización de obras multimedia digitales y su exposición en la web aunque parezca bastante lógico.

2.2. Situación en estudios no reglados y academias particulares.

En cuanto a los estudios no universitarios el panorama es mucho más amplio. En España se pueden encontrar algunas escuelas privadas que están dotando de profesionales a empresas de todo el mundo.

Estas empresas movidas por la creciente demanda de personas con un alto grado de conocimientos tanto técnicos como artísticos, ofrecen cursos donde se pueden realizar estudios que compaginan los dos mundos.

Estas academias, normalmente, han comenzado impartiendo clases de un programa de modelado como Lightwave, Maya, etc. y todos sus cursos tienen como origen este software por lo que hay una dependencia muy grande de la tecnología del momento. Este hecho no favorece los conocimientos generales que deben de impartirse en estudios regulados por ley como son las universidades.

Algunas de estas academias pueden ser centros de estudios con calificación ATC, Authorized Training Center, es decir, con unas titulaciones que aunque se adaptan a las necesidades laborales del momento, no son oficiales. De todas formas, al tener un profundo conocimiento de un software en particular, los hace muy interesantes para los alumnos que buscan un salida profesional inmediata, ya que estas academias funcionan casi como empresas satélites de formación, poniendo en contacto a los clientes propietarios del software con los artistas que lo saben utilizar. Por todo ello, los cursos suelen ser bastante caros llegándose a pagar 6000 € por año y curso. Las titulaciones suelen extenderse de 2 a 5 años.

Algunas de las escuelas más importantes de España son:

Escuela de Artes gráficas digitales ANIMUM. Su dirección es www.animum3d.com

Imagen 2.2.1: Página inicial de Animum

Scholl of Arts Trazos. Su dirección web es www.trazos.net

Imagen 2.2.2.: Página inicial de Trazos

Escuela animación en 3D Oscillonschool. Su dirección es www.oscillonschool.com

Imagen 2.2.3.: Página inicial de Oscillon School

Para encontrar más información sobre escuelas de modelado y animación 3D podemos usar: http://www.3dwire.es/noticias/noticia/listado_de_escuelas_de_animacion_en_espa%C3%B1a

2.3. Salidas profesionales.

Como se indica en el artículo “Trabajos reales en 3D” referenciado en la bibliografía, existen muchas áreas de especialización que dan trabajo a los artistas profesionales de las 3D. En el curso que nos ocupa, creemos que hay dos salidas especialmente adecuadas en el plano profesional: el diseño de productos y el 3D para multimedia interactiva (videojuegos y anexos).

2.3.1. Diseño de productos.

Esta área es muy interesante porque actualmente está en clara expansión. Está a medio camino entre el campo del diseño industrial de productos y el modelado-animación en 3D, por lo que se requiere una combinación de conocimientos muy específica. Ya hay muchos ingenieros y arquitectos trabajando en este campo, aunque siempre con una vocación claramente artística.

El flujo del trabajo también varía de un cliente a otro, ya que a veces se debe “presentar” un trabajo en Internet que ya existe y por tanto a partir de datos ya existentes. Otras veces, se trabaja desde cero, desde el mismo departamento y con las mismas personas que van a crear el producto.

Hoy en día, en España, normalmente es un trabajo remunerado como freelance. Un ejemplo de empresa que se dedica a este tipo de trabajos es Drive Inc, Ltd. www.drivein.co.uk.

Imagen 2.2.4: Página inicial de Drivein

Un ejemplo de una universidad especializada en esta clase de estudios es la Bournemouth University con su programa de diseño industrial. Su dirección es: www.bournemouth.ac.uk.

2.3.2. 3D para multimedia interactiva.

Dentro de esta área incluiríamos los videojuegos y los productos 3D para su integración en sistemas multimedia y que utilizan muchos tipos distintos de formatos como pueden ser sistemas de simulación, de realidad virtual y otros.

En la actualidad, este tipo de trabajo es aprovechado por muchos profesionales que anteriormente realizaban cd-roms interactivos, y que ante el agotamiento del mercado, migraron hacia Internet. Hoy en día, al no terminar de despegar la web 3D interactiva, donde se podría absorber una gran cantidad de trabajo, su salida se enfoca hacia la creación de contenidos para móviles.

Normalmente se trabaja en pequeñas compañías, compuestas a su vez por freelances muy creativos y con gran capacidad técnica. La formación es esencialmente autodidacta, ya que apenas existen centros de formación que realicen estudios para un trabajo tan poco estandarizado y que requiera conocimientos de tantas disciplinas distintas.

3. Estudio de situación de estas tecnologías: programas.

3.1. Introducción a las tecnologías necesarias.

Primero vamos a aclarar qué entendemos por tecnología. Podemos definir tecnología como "conjunto de teorías y de técnicas que permiten el aprovechamiento práctico del conocimiento científico"[1].

En la tarea que nos ocupa debemos utilizar este conjunto de técnicas con el objetivo de crear una obra virtual. Evidentemente, este producto será realizado y visionado siempre de forma digital por lo que utilizará tecnología digital. La tecnología digital posee unas características

[1] Diccionario de la lengua española on line de la RAE.

propias que lo diferencian con respecto a otras formas de producción artística, pero destacaríamos sobre todo las siguientes:

Se puede distribuir fácilmente: con o sin Internet su formato hace que nuestra obra pueda ser casi universalmente vista por espectadores de todo el mundo.

Es irreal: esto permite evitar las limitaciones del mundo físico, haciendo posible que el autor pueda expresar formas, ideas y conceptos imposibles de implementar con otros medios.

Entre las desventajas, tenemos:

No son reales: nunca podremos transmitir al espectador que "sienta" realmente nuestra obra. Por muy adelantadas que estén las técnicas para "engañar" a nuestros sentidos, nuestra obra no estará realizada con átomos, sino con bits. No se olerá, ni tocará,…

Se puede copiar fácilmente: Todas las fases de creación y todos sus instrumentos podrán ser utilizados de forma general para recrear la pieza.

Es perturbable: ya que todo el sistema por ser digital es fácilmente modificable. La desventaja en este sentido con respecto al óleo, el hierro,... parece evidente.

3.2. Características que debe tener la tecnología utilizada.

Cuando propusimos realizar este trabajo, sabíamos que el principal problema radicaría en encontrar una serie de programas que permitieran realizar y exponer la "obra virtual". Es imprescindible pensar que debemos utilizar ordenadores de tipo medio y programas más o menos comunes y accesibles, entre otras cosas para dirigirnos a un público no elitista en cualquier sentido.

También es necesario pensar que el proceso va a requerir la utilización, y por tanto el dominio, de una serie de programas y no de uno solo. De todas formas, todos deben compartir unas determinadas características, que son las siguientes:

a) No depender en exceso del panorama actual del software. Debido a que el software sufre cada poco tiempo una gran cantidad de cambios y mejoras, se tiene que buscar programas muy consolidados y enfocar el aprendizaje más a los procesos y métodos que a los detalles de cómo los solucionan cada uno de los programas.

b) Estar presente en el mercado. Es decir, que sean programas que estén dentro de las exigencias que tiene el mercado global de las últimas tecnologías. Esto hace que los usuarios se puedan aprovechar de la oferta y la demanda y de la constante búsqueda de mejoras para ganar cuota de mercado de las grandes empresas de software. También supone que dispongamos en muchos casos de dos versiones: una para el aprendizaje de estudiantes y otra para profesionales y particulares.

c) Programas de empresas solventes. Al elegir una tecnología debemos tener en cuenta que vamos a invertir unos recursos en dinero y tiempo para lograr su adquisición y aprendizaje. Por eso debemos de escoger unas tecnologías que pertenezcan a empresas con un bagaje consolidado, con éxitos en su pasado profesional y sobre todo con un futuro. Para ello, podemos comparar la trayectoria de la empresa, el conjunto de productos que ofrecen, cómo son considerados por profesionales, cual es la frecuencia de las actualizaciones, etc.

Podemos optar por soluciones de varias compañías, es decir, consorcios que plantean un estándar o por empresas que funcionan por separado. En este sentido tenemos algunas opciones, que surgen cuando una tecnología no está muy extendida y por distintas razones todos ven una oportunidad de negocio pero nadie se atreve a llevarla a cabo por si mismo. Es el caso, por ejemplo, del consorcio X3D. En el lado opuesto están las empresas que crean una serie de programas para ofrecer una solución integral. En este sentido yo personalmente me

decanto por la segunda opción, ya que históricamente han sucedido varios fracasos de está índole con la opción del consorcio como fue el caso de VRML97.

d) Utilizar los menos programas posibles. Las aplicaciones que utilicemos deben ser las menos posibles para no entorpecer demasiado el flujo de trabajo de los proyectos. Pero sobre todo, debemos utilizar programas que se puedan considerar como muy conocidos en sus respectivos ámbitos de utilización. Y por supuesto que las exportaciones e importaciones funcionen adecuadamente. Los formatos que generen también deberían estar ampliamente difundidos en su ámbito de actuación.

e) Deben permitir la interactividad. Es evidente que la "producción artística virtual" tiene características propias que lo diferencian del resto. Una de estas características es que puede ser interactivo, es decir, que el producto de nuestro trabajo puede responder a las distintas acciones del usuario. Por ello, esta propiedad la tiene que soportar casi de forma natural el formato de salida que utilicemos.

f) Programación. Otra característica muy importante y que a veces no es muy evidente, es que el software debe permitir distintos grados de "control". Por lo que podría ser factible el trabajar con objetos, con acciones preprogramadas e incluso con un control total, es decir, con programación. Hay programas que implementan incluso sus propios lenguajes, aunque si es posible, recomendamos utilizar lenguajes estándar.

g) Programas escalables. El tipo de arquitectura que tengan los programas es muy importante sobre todo para la distribución del trabajo final. Hoy en día la mayoría de los programas se basan en un núcleo y una serie de módulos cuya funcionalidad puede ser aumentada instalando plugins. Hay programas que trasladan esta filosofía a los resultados finales, creando ficheros compilados que no incluyen todas las opciones para su ejecución, sino solo las que verdaderamente utilizan el trabajo final.

h) Distintos modos de visualización. Es de suma importancia que el resultado pueda visualizarse de distinta forma según donde se vaya a exponer la obra. Por ejemplo, si es en una muestra o museo, podemos prescindir de la utilización de Internet y ejecutarla en ordenadores locales. Pero también pretendemos que podemos utilizarla tal cual, sin realizar ningún cambio, de forma on-line. O sea, que tengamos el mismo trabajo pero con distintos tipos de "visualizadores".

3.3. Origen y evolución de estas tecnologías.

En los años 90 surgen una serie de lenguajes que permiten la creación de objetos tridimensionales interactivos y que además están expresamente codificados para la Red.

El más importante de todos ellos es VRML (Virtual Reality Modeling Languaje) que es un lenguaje para la modelación de realidad virtual no propietario y multiplataforma para el diseño de webs en 3 dimensiones. Este lenguaje es bastante complicado y la descripción de objetos necesita gran cantidad de texto por lo que los códigos fuente eran muy largos y casi inmanejables.

Por otro parte, se demostró claramente que era una tecnología muy avanzada para el estado tanto tecnológico como humano en el que estaba la red, por lo que su implantación dejó de crecer. Los problemas que tenía esta tecnología se pueden enumerar en:

1. Necesidad de instalación de plugins para su visualización en los navegadores.

2. Conexiones lentas a Internet, normalmente a una velocidad máxima de 56 K.

3. Falta de contenidos interesantes, ya que su ámbito se limitaba a las universidades.

4. Falta de desarrolladores, ya que debían ser programadores expertos por la falta de compiladores, sistemas de ayuda y editores de aplicaciones de alto nivel.

5. Falta de interés de las empresas tras los primeros fracasos y además por su déficit en actualización de contenidos.

6. No llegó al gran público, entre otras por las anteriores razones.

Esta tecnología, de la empresa Silicon Graphics, además no tuvo un amplio apoyo por parte de las empresas del sector y tampoco consiguió imponerlo. Hoy en día se puede apreciar que el interés por estas tecnologías no ha desaparecido. Los programadores y desarrolladores, los que tejen la telaraña mundial de conocimiento (la famosa www), no dan su brazo a torcer. Podemos decir que estamos en un tiempo de espera. Poco a poco, parece que algo se mueve. Se perciben pequeños movimientos en las grandes empresas. Algunos auguran una época de renacimiento, seguida de la implantación y el éxito, por fin, de la web 3D.

En la actualidad la difícil relación entre empresa y conocimiento, técnicos y empresarios, se concreta en la aparición de dos grandes tendencias, que por otra parte han seguido casi todas las innovaciones tecnológicas informáticas:

1) El consorcio. Este surge cuando la visión de un conjunto de personas, pertenecientes a distintas empresas y ante una necesidad evidente, se ponen de acuerdo, formando una serie de grupos de trabajo para crear una agrupación de empresas que permitan, en este caso, la creación y disposición para los desarrolladores de una tecnología, capaz de rellenar los nichos tecnológicos que existen en la red. Normalmente, estos consorcios son mantenidos por distintas empresas, con los problemas que ello acarrea. Los distintos enfoques, intereses y malas interpretaciones que pueden surgir, hacen que pocos órganos de este tipo tengan, al final, un gran éxito. En este momento, el último intento no cuajado es representado por “consorcio X3D”.

2) La empresa individual. Existen empresas individuales, que ya ejerciendo una posición dominante en el mercado, o por su gran visión de futuro, apuestan por una tecnología propia. Si estas tecnologías poseen mejoras inigualables y además, tienen la fortuna de llegar al gran público, sus trabajos se convierten en estándares de facto. El ejemplo de la empresa Macromedia y su programa para gráficos vectoriales interactivos Flash fue un claro ejemplo de esto. Partiendo de algo que, en realidad, ya existía y enfocándolo de la manera adecuada, consiguieron imponerse en Internet. Además, hicieron posible que el software necesario para su visualización, su plugin, no se tuviera que cargar on-line, sino que los propios navegadores lo incluyen en sus instalaciones. Esto hace que otras aplicaciones se desarrollen alrededor de la posición central y dominante de este. Por ejemplo, Macromedia con el software Director 8.5, introdujo la posibilidad de trabajar con web3d y a partir de un paquete de tanto éxito, Adobe lanzó Atmosphere (generador de mundos 3D para Internet) que era gratuito y podía trabajar junto con Poser mediante el software Avatar Lab. El fin de la historia es conocido: Adobe compró Macromedia en 2005 y ha desarrollado Director hasta la actualidad.

Hoy en día los juegos on-line es donde se puede observar un camino de futuro para el web 3D. Tecnologías que en su día se consideraron para juegos como Shockwave o Havok 3D pueden utilizarse para la creación de muchas aplicaciones web3d, incluida, por supuesto, la faceta artística.

Otras áreas de interés donde se pueden aplicar web 3D es por ejemplo el e-learning que permite mostrar conceptos e ideas en 3D como refuerzo en la enseñanza de ingeniería (mecanismos), medicina (cuerpo), etc.

3.4. Tecnologías estudiadas

3D Webmaster

Este programa está ya obsoleto. Permite la programación en VRML97 y como hemos escrito antes, es un lenguaje difícil de aprender y ya superado por otros. Permite la importación de

objetos VRML, pero no exporta datos. No existe facilidad alguna para la realización de las interacciones y todo se tiene que programar al mismo nivel, con programación VRML.

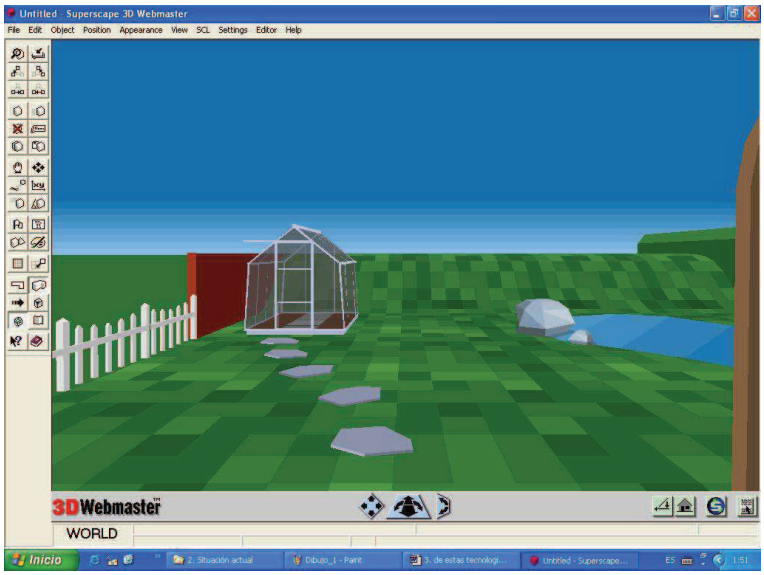


Imagen 2.3.1: Ejemplo realizado con Webmaster

No es un programa que se halla mantenido, de hecho no se puede adquirir hoy en día. Si lo utilizáramos correríamos el seguro peligro de invertir mucho tiempo y sacar poco provecho. Sí nos parecen interesantes los ejemplos realizados que trae consigo, y que podemos estudiar para implementarlos con tecnologías más modernas.

Fue una opción muy interesante para aplicaciones que se visualizan fuera de la red. Para su distribución on-line, requiere de un plugin externo que el navegador debe cargar antes de la visualización. Por ejemplo, uno que funcionaba muy bien, es el famoso Cosmoplayer. No se puede utilizar como una alternativa con futuro, ya que no cumple casi ninguna de las características que nos interesa.

Pulse

Este programa fue una gran alternativa al principio de la anterior década, pero no tenemos noticias de nuevas actualizaciones, aunque la empresa sigue en pie. Fue el gran competidor de Cult3D, otro programa enfocado exclusivamente a la web 3D.

El tipo de arquitectura interna, utilizando plugins, y totalmente abierta, ayudó a su difusión. Este hecho no debe sorprendernos si tenemos en cuenta que la empresa desarrolladora, Pulse Entertainments fue fundada por miembros de Macromedia. De hecho, esta tecnología se basa en varios programas que se instalan independientemente pero que trabajan conjuntamente, como son: pulsctut, pulsanim y pulsdmo1. El creador de contenidos para Pulse, puede utilizar un plugin para 3dstudio MAX 3.5, lo que da una idea de la calidad con la que permite trabajar en cuanto a geometría.

El modo de trabajo es mediante proyectos. Un proyecto es una colección de recursos, y estos a su vez son geometría, conductas, materiales, sonidos, escrituras y tareas. Estos recursos interaccionan en una escena para obtener finalmente una web 3D. Todos estos recursos dan una idea de la “modernidad” de su planteamiento, y han sido aceptados, y hasta cierto punto “copiados” por otros programas.

La disposición de ventanas, también es muy familiar hoy en día en muchas aplicaciones. Disponemos de ventanas como:

a) Build: para la construcción y programación de la interactividad, animación,...

b) Geometry: para controlar los objetos en escena como modelos, cámaras, luces,…

c) Window: que nos proporciona una vista de la escena, su zoom, ángulo, etc.

Las conductas o "behaviors" son muy familiares a los desarrolladores de productos Macromedia (ahora Adobe) como Flash, Director o Dreamweaver. El concepto es bastante sencillo: se pueden realizar acciones sobre los objetos en escena, respondiendo a eventos generados por el usuario. Esto nos lleva a "escribir" las acciones en cierta forma llamadas escrituras o scripts y en un lenguaje que en Pulse se llama JoeScript, el cual aporta un alto grado de funcionalidad.

Desde luego a Pulse se le debe agradecer un modo visionario para resolver los problemas que tiene la web 3D, por lo que su falta de un éxito rotundo se debe seguramente a otras razones.

Web3D Consortium

Este es un consorcio de empresas para la implantación de la web 3D. Se basa en la tecnología definida por el estándar ISO X3D que define sistemas y mecanismos para la transmisión en tiempo real de contenidos y aplicaciones 3D en la red. Soporta varios lenguajes y tipos de ficheros. Crea las condiciones necesarias para la manipulación, comunicación y visualización de escenas interactivas tridimensionales.

Imagen 2.3.2: Página inicial de web 3D Consortium

Empresas miembros de este consorcio son, entre otras, Hp, nVIDIA y Sun Microsystems, lo que indica que es la heredera de VRML. En el fondo, no es más que esto, es decir, una actualización, con una visión mucho más moderna de estándares como VRML. Las aplicaciones que siguen estas normas se pueden dividir en dos grupos: los programas de autor o edición y los programas de visualización.

Programas de autor más destacados:

Flux: es de licencia comercial. Solo disponible para Windows y es quizá la mejor opción para profesionales.

Vizx3d: permite el modelado de objetos, y la creación de ficheros x3d. Sencillo de usar y programar. Actualmente está en Beta y solo se encuentra disponible para Windows.

Wirefusion: quizá el más fácil de todos. Es de licencia comercial y para Windows y MAC. Permite crear presentaciones web3d y combinarlas con audio mp3, video mpeg y animaciones Flash. Usa java como lenguaje nativo.

Programas de visualización más destacados:

Blaxsun3d: es de uso comercial y soporta java. Es un pequeño applet de 56k que permite la visualización de cualquier escena X3D.

Flux Player: es de uso comercial y solo disponible para plataformas Windows. Es un entorno para profesionales y también lo son sus características. Permite ficheros X3D y VRML97. Provee de una serie de facilidades de navegación por defecto de altas prestaciones.

FreeWRL: es freeware, y para plataformas como Windows, Mac, Linux, etc. Sus prestaciones no son las mayores del mercado, pero es una buena forma cómoda y fácil de visualizar nuestras escenas X3D.

Esta solución no es viable para nuestro proyecto, ya que hace un tiempo que no publican nuevas actualizaciones y además no tiene un peso real en nuestro ámbito.

Shockwave 3D

Esta tecnología es una opción muy válida hoy en día. Vamos a pasar a estudiar una a una las características que debe cumplir para adaptarse a nuestros objetivos. Esta no es una tecnología “del momento”, es decir, en su creación se tuvo en cuenta el momento tecnológico adecuado. Es un software integrado en el programa líder de las presentaciones profesionales Adobe Director, con más de 20 años en el mercado.

Shockwave esta presente en el mercado. Es un programa que tiene una buena aceptación y con características muy avanzadas que se ha venido actualizando hasta la fecha. Otras compañías han sacado al mercado nuevos productos capaces de igualar a Shockwave 3D en prestaciones, aunque no deja de ser un entorno de programación 3D muy robusto, es decir, completo y probado. Es por tanto un programa para profesionales.

Programa de empresa solvente. Esta tecnología es propiedad de Adobe. Es la empresa más importante en su ámbito empresarial. Propietaria de programas como PhotoShop, Illustrator, Dreamweaver, Flash, Director,... Con esta solución, utilizamos pocos programas. Es posible realizar todo el trabajo propuesto en Shockwave 3D, ya que forma una unidad compacta dentro de Director. Esto no impide, sin embargo, que en caso necesario, utilicemos los demás componentes de este. Por eso, es una buena solución para los profesionales que ya llevan años utilizando un programa tan conocido como Director y que necesitan dar el salto a la 3D interactiva.

También cabe destacar que aunque tiene una buena librería de modelado por programación, permite importar ficheros de los programas más conocidos, como 3D Studio MAX, LightWave, Cinema, Maya, Truespace. Por lo tanto, si en necesario, se puede trabajar con programas específicos que permiten aumentar la productividad.

En cuanto a la interactividad, es uno de los puntos fuertes de este software. Permite programar la interacción con un nivel de control muy profundo. Además, se puede programar mediante acciones en respuestas a eventos (triggers). También vienen ya disponibles una serie de comportamientos (Behaviors), que hacen ahorrarnos muchas líneas de código.

Shockwave 3D permite la programación. Todo el entorno es programable. Desde el modelado a la animación y la presentación final. Por lo tanto, los ejecutables son anormalmente pequeños si no utilizamos elementos “externos” como texturas, etc. Se pueden definir y configurar por programación todo lo necesario para una animación o película interactiva: los modelos, las texturas, las luces, las cámaras, los keyframes, etc. El lenguaje nativo, es decir, el propio del programa es Lingo pero se puede utilizar indistintamente Java 3D.

Es un programa escalable. Es una filosofía que desde el principio mantiene Adobe. El propio Shockwave 3D, se puede considerar un plugin independiente dentro de Director. Además se ha colaborado con Intel y empresas de tarjetas gráficas como nVidia para aumentar la aceleración por hardware y también se han creado rutinas run-time para escalar la precisión del 3D según la velocidad de la máquina cliente.

Permite distintos modos de visualización. Las películas producidas con Director se puede orientar tanto a su visualización en un cd-rom o dvd como a la web. En Internet, es necesario un pequeño plugin que se descarga en el momento si no lo tenemos ya instalado. Sin embargo, tenemos que tener en cuenta que hoy en día Shockwave es un software contestado. Mientras unos analistas consideran que prácticamente está en vía muerta, otros lo consideran un estándar para juegos en red. De hecho, existen juegos para Sony Playstation y Microsoft Xbox realizados con Shockwave 3D.

Unity3D

En las últimas fechas, es muy comentada la aparición de este software. Es una tecnología muy válida y con características interesantes que vamos a comentar una por una. Podemos calificar a Unity3D como un “motor de juegos”. Es decir, su objetivo principal es la creación de escenas interactivas en tres dimensiones. Este tipo de software se adapta perfectamente a nuestras necesidades.

Uno de los puntos fuertes es el precio. Unity 3D siempre ha tenido una doble licencia, una gratuita y otra profesional que tiene características más avanzadas. Esto quiere decir que para nuestros trabajos, tendremos un programa base que no necesitamos comprar, solo tendremos que bajarlo de http://unity3d.com/unity/download/.

Otra inmejorable característica es que es multiplataforma. Es quizá el software más versátil en este sentido del mercado. Podemos realizar el mismo proyecto y exportarlo a sistemas Windows, Iphone, Ipad, Nintendo Wii, Mac e Internet. Permite importar archivos de programas de modelado y animación tan universales como 3D MAX y Blender con el que tiene muchos puntos en común.

También hace que podamos programar en un lenguaje tan conocido como C# y tiene un sistema novedoso de escritura de acciones con lo que algunos procedimientos muy comunes no hace falta codificarlos.

Son muchos los juegos realizados con este software, aunque debido a que los grandes estudios utilizan software propio no son los más conocidos. De todas formas destacamos, por ejemplo, “Lego Star Wars”.

Sin duda, una de las propiedades a destacar, es la posibilidad de exportar directamente a la Web mediante un pequeño plugin, con formato autoejecutable EXE y/o a Flash. Esto quiere decir que su visualización es casi universal.

Si tenemos que poner algún defecto, es en el apartado de la empresa que lo desarrolla. Esta empresa, Unity Technologies, tiene un personal de altísima cualificación y muchos de ellos trabajaban en la antigua Macromedia desarrollando Shockwave 3D. El problema es que es una empresa muy joven y con recursos económicos limitados, por lo que puede ser fácilmente absorbida por otras y hacer que desaparezca esta tecnología.

UDK, Unreal Development Kit

Comenzó siendo un “mod” del videojuego Unreal Tournament. Un mod provee a un juego de la posibilidad de ser extendido. Esto significa dos cosas: que es una tecnología potentísima y que es muy complicado en todos sus aspectos.

En la actualidad, constituye un completísimo motor de desarrollo de videojuegos 3D, de tal forma que se puede asegurar que tiene las características técnicas más avanzadas en su sector y además disponemos de una versión gratuita que se actualiza mensualmente.

Es multiplataforma, y de hecho, se han realizado juegos muy conocidos en todas las plataformas modernas, aunque no permite la ejecución directamente en la red. Las últimas versiones permiten realizar trabajos para IPhone y Android.

En la parte negativa tenemos que destacar su complejidad y falta de documentación técnica en castellano, sobre todo si lo comparamos con Unity 3D.

Por lo tanto, creemos que es un software con una línea de aprendizaje, en todas sus fases, demasiado costosa en comparación con otras alternativas como para utilizarlo en nuestro estudio.

Second Life

Es una filosofía distinta a todas las tecnologías estudiadas con antelación ya que no es una aplicación al uso, sino un metaverso. Esto implica que es una realidad virtual para Internet. Por tanto no se puede ejecutar off line. Esta característica entorpece mucho su aplicación, ya que no podemos realizar un trabajo propio de forma autónoma y por ejemplo, exponerlo en una sala sin conexión a la Red.

Permite la programación mediante un lenguaje propio llamado LSL que tiene algunas características de muy alto nivel. Es relativamente fácil de aprender y está enfocado al propio sistema, por lo que es de una aplicabilidad total.

Es interactivo en sí mismo y por supuesto, tridimensional. En realidad no existe la posibilidad de trabajar en 2D. Todo está enfocado a la inmersión en el ambiente multimedia que lo envuelve todo. Está estructurado en islas de terreno donde podemos vivir.

La empresa que lo sostiene tiene un largo recorrido y a pesar de ciertos rumores, no parece que vaya a desaparecer pronto.

Nos parece una alternativa posible, precisamente por la originalidad de su planteamiento. Los metaversos podríamos decir que son juegos sociales sin un objetivo en particular. No tenemos por qué realizar nada en concreto, podemos visitar lugares, pasear, crear una escultura, etc. Pero su objetivo está sin definir, por tanto, nosotros podríamos utilizarlo como “lugar para trabajar” como han hecho otros muchos creadores. Este es otro punto fuerte: es muy conocido dentro del ambiente más “tecnológico” del mundo creativo.

Resumiendo, podemos concluir que existen tecnologías obsoletas y otras con más futuro que presente, por lo tanto, atendiendo a criterios objetivos en beneficio del alumnado como productividad, visión de futuro y adecuación al mercado, podemos asegurar que una buena tecnología puede ser Shockwave 3D a desarrollar con Director. También merece la pena aprovechar la tecnología de Second Life para el cometido de este documento, ya que son dos puntos de vista distintos y al compararlos podemos encontrar planteamientos útiles.

Con respecto a Unity 3D, no desmerece a Shockwave 3D, pero lo mismo podemos decir de otros muchos motores de juegos. Por tanto, casi todo software de este tipo podría servir para este cometido.

4. Estudio de la actualidad de este tipo de trabajos: eventos y autores.

Para dar una visión general de este punto tan extenso vamos a dividir este estudio en dos partes. En la primera veremos algunas muestras y congresos de creación virtual y en la segunda citaremos algunos artistas o grupos de estos que pueden ser de actualidad por sus obras o por la tecnología que utilizan.

Hemos de decir que esta panorámica no pretende mostrar lo más reconocido, sino lo más cercano y útil al objetivo de esta investigación.

4.1. Eventos

Web3DArt

En primer lugar nombraremos una muestra on-line de trabajos para la web 3D llamada Web3dart por ser una de las pioneras en este ámbito y de las más importantes en su tiempo. La podemos visitar en www.web3dart.org pero no ha presentado nuevas convocatorias en los últimos años.

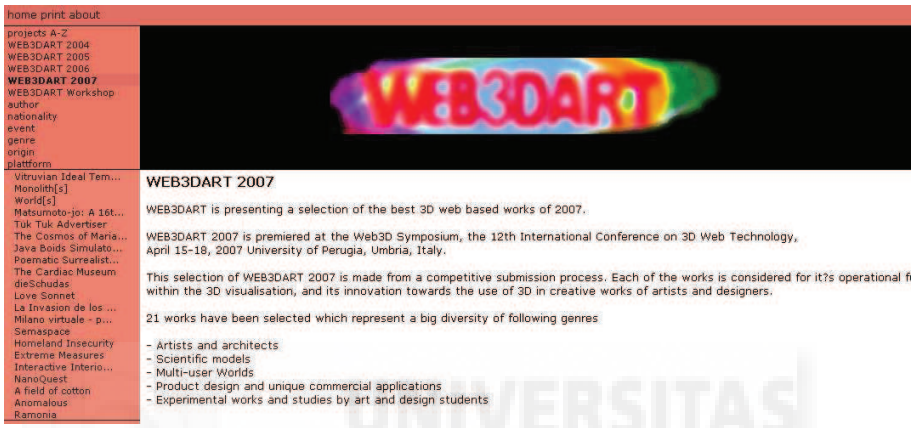


Imagen 2.4.1: Muestra Web3dart

WEB3DART presenta una selección de trabajos Shockwave 3D. Según se autoproclaman esta exhibición fue la mayor muestra internacional de su tipo, y la colección más grande de diseño virtual tridimensional.

La selección de trabajos es de gran calidad. Los trabajos se consideran según su funcionalidad operativa y su innovación. Por ejemplo, en 2004 se seleccionaron 26 proyectos, de 78 solicitantes de 20 países distintos. Desgraciadamente no hubo ningún trabajo español. Los participantes del año 2005 son de Australia, Bélgica, Suiza, Estados Unidos, Reino Unido, Letonia, Los Países Bajos, Alemania, Eslovenia, Tailandia, Canadá, Italia, Bulgaria y Francia.

Se utilizan programas y plugins para la visualización de esta muestra como Cortona, Blaxxun, Axel, Shockwave, Virtools, Cult3D,...

WEB3DART fue premiada en el Simposio de Web3D, la 9 Conferencia Internacional en 3D Tecnología de la Web, en abril de 2004 en Monterey, EE.UU. Una selección especial se presenta en la Art Gallery del SIGGRAPH 2004 en Los Ángeles

Web3D Art 2004 fué autorizada y apoyada por el Consorcio de Web3D y ACM.

Siggraph

Esta muestra se puede considerar la más grande del mundo en cuanto a gráficos por ordenador y técnicas interactivas.

Es el evento creado y realizado por el grupo SIGGRAPH de la asociación ACM que es una organización internacional científica y educativa dedicada a hacer progresar el arte, la ciencia, la ingeniería y las aplicaciones de la tecnología de la información.

Imagen 2.4.2: Muestra Siggraph 2010

Esta muestra se lleva a cabo en distintas ciudades del mundo con periodicidad anual. Trata de promover la creación e intercambio de información en las áreas de teoría, diseño, implementación y aplicación de gráficos por ordenador y de las técnicas de interacción que faciliten la comunicación. Es un espacio con un gran prestigio, por lo que muchos trabajos de investigación de estas áreas se presentan en esta feria.

Para explorar la web de la edición de 2010 que tuvo lugar en Los Ángeles tenemos la dirección http://www.siggraph.org/s2010/.

ArtFutura

Un evento muy importante, quizás el más conocido en nuestro país, es **ArtFutura** que podemos visitar en www.artfutura.org

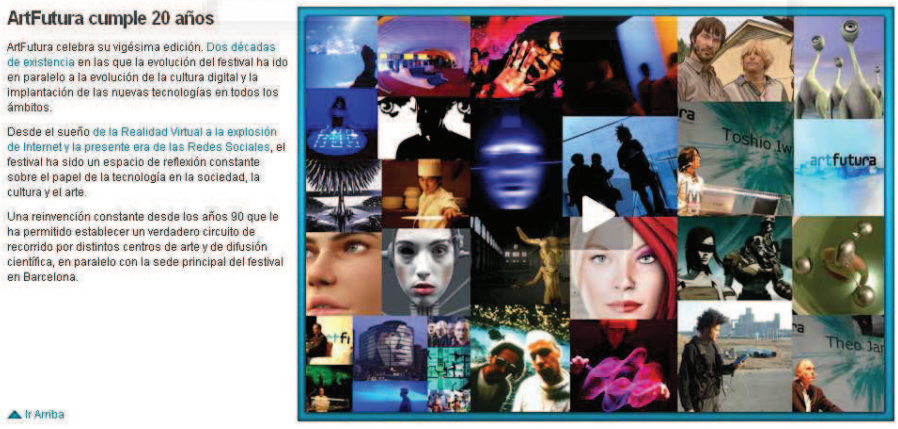


Imagen 2.4.3: Página del sitio de la muestra Artfutura

Es una muestra muy consolidada que desde enero de 1990 promueve la cultura y creatividad digital. Es una feria española pero donde exponen artistas internacionales en nuevos medios, diseños de interacción, videojuegos, animación digital, etc. Además de exposiciones, también se proyectan actividades como conferencias, talleres, instalaciones interactivas y actuaciones en directo.

Según nuestro criterio, de las últimas ediciones, fue la de 2002 que tenía el lema de “La red como lienzo” una e las más interesantes e inspiradoras ya que aparte de poder disfrutar del

trabajo de artistas como Paul Friedlander y sus “esculturas ligeras virtuales de luz”, fue uno de los acicates para emprender el presente documento.

Otros participantes de ediciones pasadas son de la talla de William Gibson, Theo Jansen, Toshio Iwai, Laurie Anderson, Hiroshi Ishii, Moebius, David Byrne, Masaya Matsuura, Howard Rheingold, Timothy Leary, Tetsuya Mizuguchi, Rebecca Allen, Ryota Kuwakubo, Orlan, Yoichiro Kawaguchi, Brian Eno, Karl Sims y otros.

Podemos observar trabajos de empresas y organizaciones como Pixar, IL& M, Plastige Image, Micros Image, Arts Electronica, Blast Theory, Sony Pictures Imageworks, Digital Domain, Amorphic Robot Works, el Media Lab de Boston o Virtual Effects del español Víctor Vinyals.

ArtFutura tiene una división llamada GaleríaFutura que promueve los aspectos más expositivos de la nueva cultura digital. Uno de sus proyectos con más éxito fue la exposición “Máquinas y Almas” presentada en el Museo Nacional Reina Sofía en el año 2008.

Esta muestra es perfecta para encuadrar el presente documento, ya que en ArtFutura se materializa que en la actualidad “arte y ciencia discurren más que nunca por caminos paralelos” y que “no es posible entender el arte desligado de los nuevos media, Internet y la tecnología digital”.

4.2. Artistas y grupos de trabajo

En cuanto a los artistas, tenemos muchísimas tendencias, aunque en particular la más provechosa para esta investigación es la que utiliza Shockwave o Second Life como principal tecnología para llevar sus creaciones a la web.

En esta introducción vamos a nombrar artistas y grupos que presentan su trabajo de forma autónoma, no en un metaverso. Esto es muy normal, ya que muchos solo utilizan soluciones como Second Life de forma secundaria. Es decir, cuando exponen en Second Life lo hacen no como fin último sino como formas de promoción o exhibiciones sin ánimo de lucro. Por tanto, veremos algunos autores de Second Life cuando le dediquemos un capítulo propio.

El problema es que la mayoría de los artistas de este grupo trabajan como directores artísticos para grandes empresas, casi todas del mundo del videojuego y por tanto no enfocan su trabajo a la exposición. Por otra parte tenemos artistas bastante reconocidos por haber sido pioneros de la creación virtual y sus trabajos se pueden considerar clásicos con menos de 10 años.

Char Davies

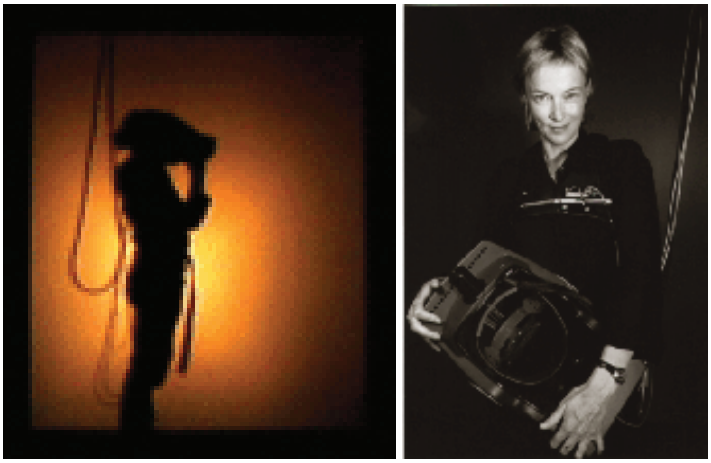


Imagen 2.4.4: Char Davies en una inmersión virtual en su trabajo

Esta canadiense tiene una larga trayectoria en la creación virtual. Con su trabajo más conocido, Osmose (1995) tuvo un rotundo éxito. Osmose es una instalación que utiliza multimedia virtual 3D y sonido en un trabajo de realidad virtual inmersivo. Implementa periféricos especiales para la visualización y la interacción es muy novedosa, ya que en parte se realiza digitalizando la respiración.

Osmose presenta un espacio virtual con presentaciones muy poéticas a medio camino entre el mundo real y el propio del sujeto que respira, ya que por medio de esta cambian paisajes y se navega. Para obtener más información: www.immersence.com. Desde el punto de vista meramente artístico, Davis cambia por primera vez el objetivo de la realidad virtual y por lo tanto su concepto en sí mismo. A diferencia de la tendencia ultra realista de la mayoría de sus camaradas, la estética de Osmose utiliza una representación abstracta, con utilización de texturas transparentes y sistemas de partículas para crear ambientes oníricos. Este trabajo continuó con la obra “Ephémerè”.

Imagen 2.4.5: Captura de pantalla del trabajo Osmose

Christophe Leske

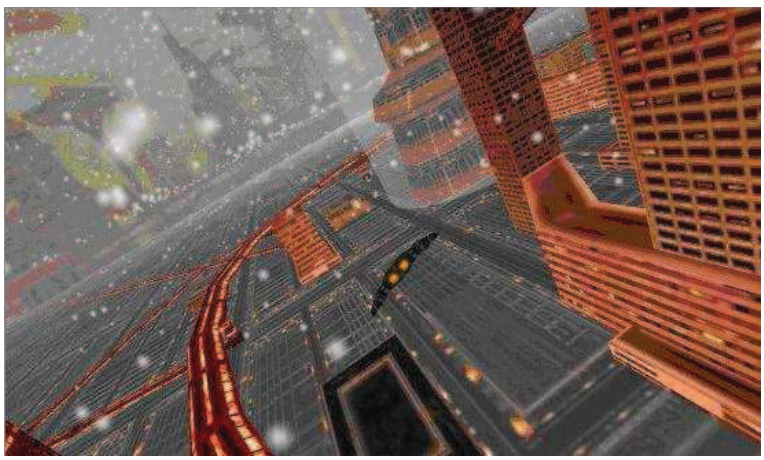


Imagen 2.4.6: Ciudad creada procedimentalmente en Shockwave 3D

Autor dedicado a la realización de experiencias artísticas en el ciberespacio. Ha trabajado como director de aplicaciones para aparatos móviles y es uno de los principales impulsores de la tecnología Shockwave 3D.

Mantiene un blog muy interesante llamado “Undocumented Lingo” (http://www.director3d.de/) sobre el desarrollo Shockwave 3D y resuelve problemas que los usuarios le comentan en www.director3d.de.

Se puede navegar por su “ciudad futura” (en la misma dirección mencionada anteriormente) y se puede experimentar cómo el paisaje, transportes y atmósfera se van creando al mismo tiempo que son visualizados, de forma procedimental, es decir, se generan por código, en tiempo real. En la imagen superior se puede ver una captura de pantalla.

Paul Catanese

Es un autor muy productivo y el creador de un libro que para los programadores en Shockwave 3D se puede considerar imprescindible que tiene por título “Director 8.5. Fundamentos de programación 3D”.

Es Master en bellas artes y tecnología por la School of the Art Institute of Chicago (SAIC) y siempre se ha dedicado al diseño industrial.

Le han sido otorgados premios por todo el mundo desde 1995 y lleva escritos 6 libros sobre arte y tecnología. Su web se puede visitar en www.paulcatanese.com

Imagen 2.4.7: Página principal de Paul Catanese

Eric Joyner

Este autor presenta una colección de trabajos muy interesante. Está muy relacionado con el web3D consortium y sus muestras se pueden considerar como un paradigma de esta tecnología y de lo que se puede conseguir con ellas. Se puede visitar su sitio en la dirección: www.thex3dxperience.com

Como se puede constatar en su web utiliza básicamente tres programas: la geometría es creada en cube3, el trabajo artístico es construido en Vizx3D y todo es expuesto en la red utilizando Flux. Estos tres programas no son gratuitos y se pueden considerar de última tecnología.

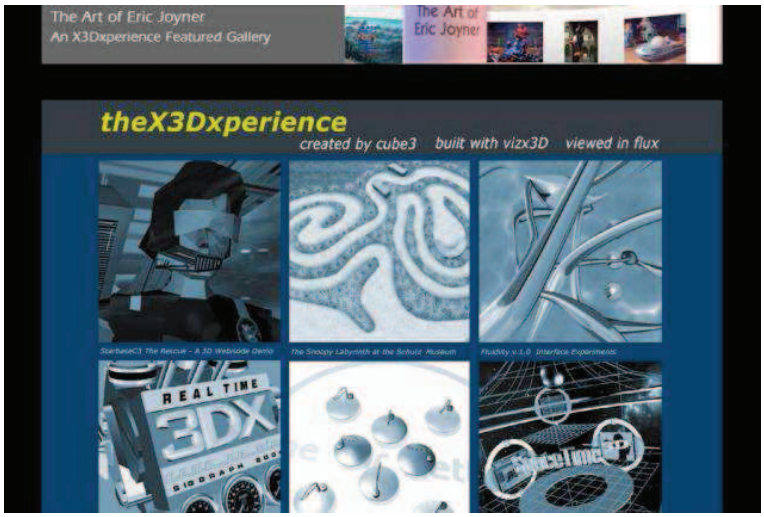


Imagen 2.4.8: Página del sitio de Eric Joyner

En la página mencionada con anterioridad se puede visitar una escena futurista que se interacciona muy fácilmente con el ratón y donde se ven calles surcadas por naves espaciales, pantallas gigantes de video y se puede saltar a otros mundos 3D. En pocas palabras, ha conseguido una ambientación que recuerda a la película Blade Runner.

En la página de Joyner se pueden encontrar trabajos sobre interacción de objetos realmente inspiradores. Para visualizar esta ciudad necesitamos un plugin que se descarga automáticamente. El sonido es una característica que aporta un alto grado de inmersión.

Imagen 2.4.9: Imagen de la ciudad 3D ideada por Joyner

Richard Smith

Este profesional de formación artística es uno de los pocos de su especie: realizador total de juegos comerciales. Cuando decimos “total” nos estamos refiriendo a un hecho excepcional. En una industria donde se mueven millones de euros e intervienen decenas de personas para realizar un juego, Richard Smith comienza con el guión y él mismo desarrolla los gráficos, los programa y produce sonido e incluso en algunos dobla las voces.

Por ello es considerado el prototipo de programador freelance ya que vende los juegos directamente a las empresas, entre las que destacan Ericsson, donde realizó un juego para móviles en 2002 que tuvo fama de ser el mejor de su segmento.

Es un entusiasta de Shockwave 3D, ya que hasta entonces realizaba su trabajo pensando en plataformas CD-ROM, pero desde la aparición de Director 8.5 trabaja fijando su mercado en la web. Sus juegos más famosos se encuentran en www.shockwave.com.

El juego Hurtle Turtle además de ser uno de los juegos on-line más solicitados tiene unos movimientos de los personajes muy conseguidos, cuestión de gran dificultad técnica.

NoiseCrime

Esta es una web de un grupo de artistas con el mismo nombre que realizan trabajos de experimentación. Lo que más destaca es la gran claridad y diseño de sus interacciones y las muestras de calidad de sus producciones.

Se puede visitar en www.noisecrime.com/develop/techdemo/d85.html

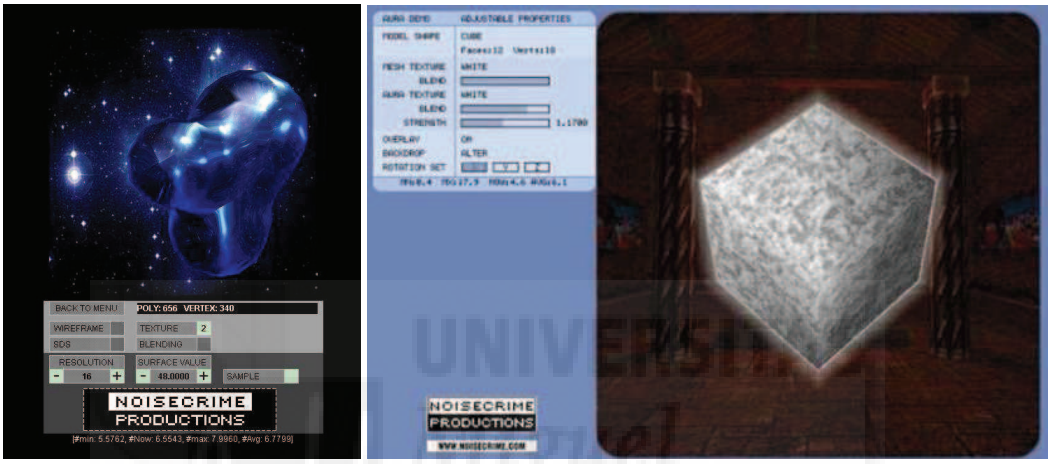


Imagen 2.4.10: Captura de pantalla de interacciones en Noisecrime

Tetsuya Mizuguchi

Mizuguchi nace en 1965 en Japón. Es un diseñador de videojuegos y fundador de la firma para el desarrollo de los videojuegos Q Entertainment. Mizuguchi trabajó para Sega durante 13 años como oficial jefe creativo.

Imagen 2.4.11: Imagen de un juego diseñado por Mizuguchi

Mantiene un blog muy dinámico en http://mizuguchi.biz/v2/html/

Ha sido director de varios videos musicales. Colabora con DJ´s e incluso a creado proyecciones holográficas como la que abrió el concierto “Live Earth” de Tokio en 2007. Fue artista invitado en ArtFutura y otras muestras de nuestro país

CAPÍTULO 3:

CURSO: PROGRAMACIÓN DIDÁCTICA

UNIVERSITAS
Miguel
Hernández

Programación didáctica

1. Introducción

Este curso va a permitir a los alumnos adquirir conocimientos y técnicas bastante novedosas para la creación de obras de última generación.

Creemos que los estudios superiores se deben adaptar a las nuevas tecnologías y a las nuevas posibilidades que de estas se derivan. No se trata de estar "a la última" sea como sea, pero sí de estar atentos a nuevas posibilidades de expresión. Sin duda alguna, la ciencia informática ha traído consigo una revolución a campos, en principio tan dispares como la medicina, la biología o los negocios. Pero todos ellos tienen algo en común, manejan grandes cantidades de información. La informática es la ciencia del tratamiento automático (es decir ayudado por máquinas) de la información. Resulta que este es el principal problema de casi todas las especialidades del saber humano. Pongamos un ejemplo ¿Cuál es sino el problema en que radica finalmente la descodificación del genoma humano? ¿Es un problema biológico o informático? La propia formulación de la pregunta responde a esta cuestión. Y de hecho, no se ha podido siquiera abordar el problema hasta que no hemos estado suficientemente ayudados por máquinas potentes y sus correspondientes aplicaciones que nos ayudasen a entender la información que desde hace tiempo sabíamos que estaba allí, que conocíamos, pero que era imposible de entender por nosotros mismos.

Creemos sinceramente, que con la multimedia pasará lo mismo. No estamos hablando de programas como Photoshop, que sin duda han sido utilizados con fines artísticos, sino de un concepto más amplio de "comunicación virtual", de formas y colores, de músicas y ambientes que solo existan en forma digital, pero que a su vez puedan ser manipuladas por el espectador según lo que disponga el autor, que sean interactivas, que modifiquen su apariencia según lo que suceda a su alrededor o a miles de kilómetros de distancia.

Conceptos como inteligencia artificial, interfaces de realidad virtual, y otros, todavía quedan muy lejos, pero hoy podemos utilizar tecnología que nos permite pensar primero en una forma y luego en un material, y no al revés, y quizá lo más interesante, es que podemos pensar en cambiar la forma y el material a voluntad del autor o del espectador.

Hoy en día, el "artista virtual" seguramente necesitará programar su obra a poco que requiera interacción. Y esto provoca muchas veces el rechazo de los no iniciados en este conocimiento. Pero, ¿no necesita cualquier autor un material para representar su obra? ¿No se mancha las manos de arcilla el escultor? ¿Es menos artista el que utiliza las mismas herramientas que un soldador naviero para crear el mensaje de su obra? Está claro que no. Hay una gran separación entre la obra y la herramienta utilizada para su realización. Por lo tanto hay que concluir que si estamos trabajando con "materiales" virtuales, debemos de conocer la base de la creación para sacarle el mayor fruto. Evidentemente, utilizaremos programas que nos ayuden en esta tarea, pero trabajar con variables, sentencias y constantes de lenguajes de programación, no nos debe asustar. En el fondo es como "hablar" con una "inteligencia extraña" en un lenguaje que queda a medio camino entre el nuestro y el suyo.

La razón principal para realizar este curso se ha enunciado anteriormente, pero queremos añadir además las que siguen:

a) Introducirnos en la creación de objetos tridimensionales que nos pueden ayudar para la realización de obras "reales" como estudios previos de volumen, etc.

b) Aprender como podemos "exponer" nuestra obra en Internet, para disponer de un espacio "abierto" y de alcance global.

c) Para la introducción en el software de ayuda a los artistas, como Poser, que nos provee de modelos virtuales que podemos ver con todo lujo de detalle, desde distintos puntos, vestimentas, luces, etc.

Con este documento pretendemos poner en manos del profesor, un documento que ayude en la realización de esta clase de cursos, que hoy en día están poco a poco introduciéndose en el ámbito universitario español.

Pensamos que este tipo de aprendizaje se debería entroncar con otros de la misma índole, pudiendo crear cursos que ahonden en cada uno de los aspectos que en este se discuten. Por lo tanto, creemos que se podrían realizar cursos donde se estudien de modo unitario el modelado, el texturizado, las luces y cámaras, etc.

La ficha que exponemos a continuación sigue un formato orientativo.

2. Ficha del curso.

Nombre del curso: Herramientas para producción virtual 3D
Obligatoriedad: No
Curso: Postgrado
Temporalidad: Cuatrimestral.

Descripción: Conocimiento de la realización y publicación de producción virtual tridimensional.
Departamento: Aquél que contenga herramientas y asignaturas afines.
Área: Nuevas tecnologías.
Titulación: Técnico en herramientas de creación virtual.
Centro: Facultad de bellas artes.

Profesorado:
Responsable: profesor titular.
Teoría: profesor con conocimientos de gráficos digitales 3D y programación.
Práctica: profesor con conocimientos de CG 3D, programación y publicación web.

Información académica:

Objetivos globales: Se pretende que el alumnado conozca los modelos y procesos para la elaboración y muestra de obras infográficas 3D y su exhibición. Desde el modelado, pasando por la texturización, animación, iluminación, cámaras, representación, programación de interactividad, manejo de los distintos tipos de formatos y la publicación en la web y/o la muestra off-line.

Teoría:

Objetivos globales teóricos: Adquirir los conceptos y procesos de construcción, programación y publicación de los elementos que intervienen en las obras 3D.

Temas teoría: Hemos programado un módulo teórico general. Además cada unidad de los casos prácticos de implementación, comienza con un desarrollo teórico específico.

Unidades del módulo teórico:

Unidad 1: Introducción a la multimedia 3D interactiva
Unidad 2: Modelado
Unidad 3: Materiales: superficies y texturas
Unidad 4: Iluminación
Unidad 5: Cámaras
Unidad 6: Animación
Unidad 7: Simulaciones físicas
Unidad 8: Sistemas de partículas
Unidad 9: Programación
Unidad 10: Interacción en mundos virtuales

Unidad 11: Integración multimedia
Unidad 12: Publicación on/off line

Realizaciones prácticas:

Objetivos globales prácticos: Praxis de casos con herramientas software específicas que se concretan en dos casos o realizaciones.

En estas implementaciones, dividimos su contenido en unidades. Estas unidades a su vez, las dividimos en dos apartados: introducción teórica y casos prácticos específicos que pueden variar en número.

La primera realización práctica mostrará cómo conseguir el objetivo del curso utilizando dos aplicaciones software: 3D Studio Max y Director.

La segunda realización práctica mostrará como generar la misma tarea utilizando el metaverso Second Life.

Las unidades funcionales de las dos implementaciones son las siguientes:

1. Implementación con aplicación de max- director

Unidad 1: Introducción a la implementación MAX-Director
Unidad 2: Flujo de trabajo
    Caso práctico 2.1: Flujo de trabajo
Unidad 3: Modelado con MAX
    Caso práctico 3.1: Modelado con primitivas y técnicas básicas
    Caso práctico 3.2: Modelado de extrusiones.
    Caso práctico 3.3: Modelado de revoluciones
    Caso práctico 3.4: Modelado de solevaciones
    Caso práctico 3.5: Modelado poligonal
Unidad 4: Materiales: superficies y texturas con MAX
    Caso práctico 4.1: El editor de materiales
    Caso práctico 4.2: Creación de superficies básicas
    Caso práctico 4.3: Mapas difusos
Unidad 5: Iluminación con MAX
    Caso práctico 5.1: Crear distintos tipos de luces
Unidad 6: Cámaras con MAX
    Caso práctico 6.1: Crear distintos tipos de cámaras
    Caso práctico 6.2: Travelling de cámara
Unidad 7: Exportación de la escena de MAX a DIRECTOR
Unidad 8: Programación en DIRECTOR
    Caso práctico 8.1: Estructura básica de la película interactiva
    Caso práctico 8.2: Comportamientos predefinidos
Unidad 9: Interacción con DIRECTOR
    Caso práctico 8.3. Transformaciones básicas de objetos
    Caso práctico 8.4: Interacción avanzada con objetos
Unidad 10: Iluminación en DIRECTOR
    Caso práctico 10.1: Iluminación en Director
Unidad 11: Cámaras en DIRECTOR
    Caso práctico 11.1: Cámaras
Unidad 12: Materiales con DIRECTOR
    Caso práctico 12.1: Asignación de texturas
    Caso práctico 12.2: Control de superficies
    Caso práctico 12.3: Texturas con transparencias: canales alpha
    Caso práctico 12.4: Texturas con mapas de reflejos
    Caso práctico 12.5: Texturas animadas por transformación
    Caso práctico 12.6: Texturas animadas por intercambio
Unidad 13: Animación con DIRECTOR

Caso práctico 13.1: Animación fotograma a fotograma
Unidad 14: Sistemas de partículas con DIRECTOR
Caso práctico 14.1: Sistemas de partículas, geometría
Caso práctico 14.2: Sistemas de partículas, emisión
Caso práctico 14.3: Sistemas de partículas, efectos atmosféricos
Unidad 15: Simulaciones físicas con DIRECTOR
Unidad 16: Integración multimedia con DIRECTOR
Unidad 17: Publicación off/on line con DIRECTOR
Caso práctico 19: Publicación del trabajo on y off line

2. Implementación con el metaverso Second Life

Unidad 1: Introducción a Second Life
Unidad 2: Herramientas y procedimientos básicos en Second Life
Caso práctico 2.1: Procedimientos básicos y flujo de trabajo
Unidad 3: Modelado de objetos
Caso práctico 3.1: Crear objetos prim
Caso práctico 3.2: Operaciones básicas con objetos
Caso práctico 3.3: Objetos enlazados
Caso práctico 3.4: Modelar con modificadores de cajas: pebetero
Caso práctico 3.5: Modelar con modificadores de esferas: escarabajo
Caso práctico 3.6: Modelar objetos con modificadores de toros: ánfora
Caso práctico 3.7: Realizar Sculps
Unidad 4: Superficies y materiales de los objetos
Caso práctico 4.1: Aplicación de superficies diversas
Caso práctico 4.2: Materiales con canales alpha
Caso práctico 4.3: Texturas animadas
Unidad 5: Iluminación en Second Life
Caso práctico 5.1: Creación de autoiluminación y resplandor
Caso práctico 5.2: Creación de objetos luminosos
Unidad 6: Cámaras y machinima en Second Life.
Caso práctico 6.1: Control de cámaras
Unidad 7: Simulaciones físicas en Second Life
Caso práctico 7.1: Crear objetos físicos
Caso práctico 7.2: Experimento de física
Caso práctico 7.3: Crear objetos flexibles
Unidad 8: Programación en Second Life
Unidad 9: Animación de objetos en Second Life
Caso práctico 9.1. Objeto rotatorio por programación
Caso práctico 9.2: Control del movimiento por programación
Unidad 10: Interacción con los objetos
Caso práctico 10.1: Interactividad con los objetos
Unidad 11: Sistemas de partículas
Caso práctico 11.1: Sistema de partículas desde el inicio: Fuego
Unidad 12: Integración multimedia en Second Life
Unidad 13: Presentaciones off line de Second Life

Objetivos específicos

Propiciar el aprendizaje de los conceptos claves y la realización de nuevas expresiones multimedia facilitadas por la informática y sus redes.

Metodología docente

Exposición por parte del profesor de teoría de los distintos temas, haciendo hincapié en los modelos a seguir para poder evitar la dependencia de un determinado programa.

En las clases prácticas se implementarán los distintos casos en programas que sean de amplia difusión en el mundo profesional. Se partirá de contenidos ya realizados que aportará el profesor con el fin de fijar los objetivos de cada práctica en concreto y a fin de evitar que se arrastren problemas de prácticas precedentes.

Los ejercicios se requerirán por escrito y se señalarán las fechas de entrega y exposición de los trabajos. Se finalizará con un proyecto completo que abarcará todas las partes imprescindibles del curso.

En horario de tutorías se atenderán las consultas individuales y se pondrán a disposición del alumno las soluciones adecuadas. También se podrán realizar tutorías por Internet para facilitar la inmediatez e interactividad de esta actividad.

La clase estándar deberá tener en cuenta que la mayor atención del alumno se obtiene si este no está utilizando el ordenador. Por eso, recomendamos que con los ordenadores apagados, el profesor haga una exposición clara de lo que se pretende. Para ello, debería utilizar su propio ordenador y un proyector y por supuesto, debe formar parte de un "proyecto", que enlace con lo visto anteriormente. Todo esto provoca que las clases deban estar muy bien preparadas.

Luego, el alumno podrá encender el ordenador asignado y seguir realizando su propio proyecto artístico pero aplicando exclusivamente las técnicas y conocimientos explicados por el profesor en esa misma clase, para desarrollar el curso de manera sincronizada.

Este tipo de clases hace imprescindible que el alumno pueda "repetir" la clase por sí mismo, por lo que sería necesario que el profesor pusiera a disposición de los alumnos un conjunto de apuntes, a ser posible en formato electrónico y ficheros donde estén realizadas las clases hasta el momento, para que no tengan que ser repetidas desde el principio. Por la naturaleza de este material, es recomendable publicarlo en páginas web. Además se podría poner otra clase de información, con enlaces a tutoriales web, etc. Esto nos conduce a utilizar una plataforma de tipo Moodle.

Esta plataforma online también podría ser un factor importante para responder a uno de los principales retos de todo curso: cómo responder a la diversidad de alumnos e intereses. En un principio, aunque las clases sean presenciales, se podrían utilizar como base para la realización de cursos blend-learning, donde se mezclan de manera adecuada la formación presencial con la realizada a distancia.

Sistema de evaluación:

Es imprescindible la asistencia a las clases con regularidad.

Se realizará un seguimiento continuado sobre todo de la parte práctica.

Se realizará un examen final por escrito que constará de dos partes: teoría y práctica.

Se realizará un proyecto individual y completo sobre la materia del curso.

Con estos tres elementos: prácticas, examen final y proyecto, se evaluará al alumno y se adjudicará una nota al alumno.

El sistema y método de evaluación se ceñirá en todo momento a la normativa de evaluación de estudiantes.

Comentarios:

La programación será flexible según los condicionantes del horario, los elementos informáticos y de red y sobre todo de los conocimientos previos sobre el tema del alumnado, para lo cual se

realizará al principio del curso una evaluación inicial en la que desarrollaremos una encuesta inicial y unos encuentros con diálogos, ideas y comentarios sobre la materia.

Medios de trabajo

Alumnos por unidad de curso: dependerá de muchos factores. Uno de los más importantes es la cantidad de ordenadores de que dispongamos, ya que es imprescindible contar con un ordenador por alumno, sobre todo para la creación de una obra personal. Aún así, por la naturaleza de la tecnología a utilizar (programación y software de última generación) creemos que los alumnos no deberían de sobrepasar el número de diez.

Tipo de ordenador y periféricos: Por las razones anteriormente expuestas, los ordenadores y los programas que utilizaremos deben ser de última generación. La eterna disquisición de la plataforma a utilizar (estaciones de trabajo, apple, pc´s) creo que se soluciona resaltando que el trabajo debe de poder ser repetido por el alumno en su lugar de trabajo, por lo que yo me decantaría por la simplicidad y precio de los ordenadores personales. Además, de este modo, también haríamos esta "creación virtual" más accesible y democrático a los alumnos.

En cuanto a los periféricos, sería necesario disponer de grabadoras de dvd´s, debido a la gran cantidad de memoria que consumen los ficheros que generaremos en este curso. También serían necesarias unas impresoras a color de gran calidad, así como escáneres de alta resolución, tabletas digitalizadoras y cámaras digitales para la creación de texturas de calidad.

Tipo de conexiones: las conexiones deben ser lo más "reales" posibles. Esto quiere decir, que como tenemos que "exponer" e interactuar con nuestras obras en la red, lo debemos hacer en igualdad de condiciones que la mayoría de nuestros posibles espectadores. Por ejemplo, no serviría de nada que nosotros utilizásemos un ADSL a 20 Mb en la facultad para ver una determinada interacción bien resuelta, si luego nosotros desde casa la vemos 10 veces más lenta.

Tipo de materiales perecederos: los normales en un aula de informática como pueden ser papel, cd´s y dvd´s, etc.

Bibliografía recomendada:

Autores: Angie Jones, Sean Bonney, Brandon Davis, Sean Miller, Shane Olsen.
Título: Edición Especial 3D Studio Max 3 Animación profesional.
Nº Edición: 1
Lugar de edición: Madrid
Editor: Prentice Hall.
Año de edición: 1998.
Nº Páginas: 724
Traducción: (3D Studio MAX 3 Professional Animation, New Riders)
ISBN: 84-205-2942-7

Autor: Paul Catanese
Título: *Director 8.5. Fundamentos de programación 3D.*
Nº Edición: 1ª
Lugar de edición: Madrid
Editor: Anaya Multimedia
Año edición: 2002
Lengua: Castellano
ISBN: 84-415-1348-1

Autor: Michael Rymaszewki
Título: *La guía oficial de Second Life.*
Nº Edición: 1ª
Lugar de edición: Madrid

Editor: Anaya Multimedia
Año edición: 2008
Lengua: CASTELLANO
ISBN: 9788441523135

Autor: Aimee Weber, Kimberly Rufer-Bach, Richard Platel
Título: *Creating Your World: The Official Guide to Advanced Content Creation for Second Life.*
Nº Edición: 1ª
Lugar de edición: Indianapolis, Indiana, USA
Editor: Linden Research, Inc.
Año edición: 2008
Lengua: Inglés
ISBN: 978-0-470-17114-1

Autor: Dana Moore, Michael Thome, Karen Haigh
Título: *Scripting Your World: The Official Guide to Second Life Scripting.*
Nº Edición: 1ª
Lugar de edición: Indianapolis, Indiana, USA
Editor: Linden Research, Inc.
Año edición: 2008
Lengua: Inglés
ISBN: 978-0-470-33983-I

Bibliografía adicional:

Autores: Ted Boardman, Jeremy Hubbell
Título: 3D Studio Max 3, Modelado, Materiales y Representación.
Nº Edición: 1
Lugar de edición: Madrid.
Editor: Prentice Hall
Año edición: 2000
Nº Páginas: 756
Traducción: (3D Studio MAX 3 Modeling, Materials, and Rendering, New Riders)
ISBN: 84-8322-188-8

Autores: Macromedia Inc.
Título: *Learning Lingo.*
Nº Edición: 1
Lugar de edición: San Francisco
Editor: Macromedia Inc
Año edición: 1997
Nº Páginas: 173
Traducción: (en inglés)
ISBN: sin isbn, manual del propio programa.

Autor: Daniel Terdiman
Título*: The Entrepreneur's Guide to Second Life: Making Money in the Metaverse.*
Nº Edición: 1ª
Lugar de edición: Indianapolis, Indiana, USA
Editor: Linden Research, Inc.
Año edición: 2008
Lengua: Inglés
ISBN: 978-0-470-17914-7

Horarios:

Las clases deberán ser de al menos 2 horas ya que tanto en las clases teóricas como en las prácticas, necesitamos un tiempo para situar la sesión en contexto, hacer resúmenes de contenidos previos,… antes de comenzar la adquisición de conocimientos nuevos.

Horario semanal: 4 horas semanales repartidas en 2 horas

Material: Sin especificar.

Evaluación: Sin especificar.

CAPÍTULO 4:

CURSO: MÓDULO TEÓRICO

Unidad 1: Introducción a la multimedia interactiva en 3D.

1. Conceptos básicos.
2. Generación de escenas en tres dimensiones.
3. Interacción con gráficos 3D.
4. Publicación de multimedia 3D interactiva.

1. Conceptos básicos.

Vamos a comenzar este curso con una introducción teórica a los conceptos básicos que trata. El objetivo del curso es conseguir la capacidad técnica para poder crear escenas en tres dimensiones con contenido multimedia e interactivo para la generación de trabajos personales.

Por supuesto, aquí se reseñan las habilidades técnicas, la posibilidad de un campo de trabajo para un artista. No pretendemos hacer arte, solamente unas herramientas que pudieran ser utilizadas o no por un artista para crear su obra.

Tenemos que enfatizar, que vamos a obtener una "escena" en tres dimensiones. Este concepto es básico para nuestro objetivo, ya que vamos a disponer de un "lugar" virtual que tiene tres dimensiones como nuestro propio mundo físico. Esta escena 3D la proporcionan muchas aplicaciones actuales.

Unas aplicaciones permiten trabajar en ellas con el único fin de generar gráficos y animaciones de estos. Normalmente, producen gráficos de mucha calidad los cuales necesitan mucho tiempo para generarse. Por ejemplo, para crear las imágenes de las películas realizadas en 3D se emplea software de este tipo, luego con otra aplicación se le añadirá el sonido para montarla.

Otro tipo de aplicación 3D son los que producen los videojuegos. Estas aplicaciones deben de responder en tiempo real a las acciones de los usuarios, por lo que utilizan tecnologías muy avanzadas para producir los gráficos requeridos al instante. Por ahora, la calidad de los gráficos aunque muy alta, no alcanza los niveles de las películas, ya que en estas se tiene todo el tiempo requerido para producir los fotogramas.

Además, necesitamos la interacción, ya que queremos aportar esta característica. Esto hace que el primer tipo de aplicaciones no pueda ser utilizado directamente para nuestros fines.

Por otra parte, queremos que el trabajo final pueda contar no solo con imágenes, sino con sonido, efectos especiales, animaciones y vídeo.

Todos estos requerimientos hacen que nuestro enfoque vaya dirigido a utilizar aplicaciones de generación de juegos pero para utilizarlos con otro objetivo. No buscaremos "jugabilidad" sino la capacitación para tener la posibilidad de generar y exponer trabajos propios.

Además, las aplicaciones enfocadas a la realización de videojuegos también tienen una característica que nos incumbe como es la facilidad para la exhibición de nuestra obra. Los juegos actuales tienen cuatro puntos de exhibición posibles: Internet, desde las memorias de los ordenadores, los móviles de última generación y en videoconsolas.

A nosotros nos interesa especialmente las tres primeras, ya que las videoconsolas no son, en principio, un medio donde hoy en día se exhiban producciones artísticas.

Por lo tanto, aunque en los dos tipos de producciones: películas y videojuegos, es indudable que se puede producir arte y que en estas siempre existe la figura del director artístico, utilizaremos la base de los videojuegos como soporte para generar las escenas que deseamos.

2. Generación de escenas en tres dimensiones.

En la actualidad, los programas con los que podemos generar gráficos en 3D, nos permiten acceder inmediatamente a una escena tridimensional que en principio estará vacía. Esta escena que tiene coordenadas en las tres dimensiones la veremos a través de una pantalla en dos dimensiones, y se manipulará normalmente con el ratón que también trabaja en dos dimensiones por lo que el sistema tendrá constantemente que "traducir" el espacio 3D a imágenes 2D para visualizaras y las órdenes 2D del ratón a operaciones 3D mediante el uso de teclas de combinación y comandos de aplicación.

Los gráficos 3d por ordenador son creados mediante la aplicación de cálculos numéricos que crean una geometría en tres dimensiones y sobre esta, una visualización de objetos y su objetivo es la creación de una imagen en dos dimensiones, ya sea para utilizarla de forma individual o como parte de una película, que necesitará una gran cantidad de estas imágenes.

Los cálculos numéricos comentados son normalmente muy complejos ya que deben trabajar con un tercer eje, el de la profundidad, y además deben de simular el sistema de visión humano que es binocular con un sistema monocular. Las técnicas y algoritmos utilizados usarán las transformaciones de tamaño, la oclusión, las perspectivas y otras para realizar estas simulaciones.

Por lo tanto, los programas de generación de gráficos 3D por ordenador, permiten crear y modificar objetos 3D en un entorno o escena también 3D con el objeto de crear imágenes 2D, bien sean estas para su impresión o muestra en dispositivos como televisiones u ordenadores.

Existen en el mercado multitud de aplicaciones para la creación de gráficos 3D. Nosotros utilizaremos tres: 3d Studio MAX, Director y Second Life.

Estas aplicaciones son muy distintas entre sí, y a lo largo del curso que sigue tendremos espacio para diferenciarlas, pero como introducción podemos decir, que mientras MAX es una aplicación para generar escenas en 3D, Director está enfocado para el tratamiento de la interacción multimedia, incluyendo las escenas 3D.

Por otra parte, Second Life, es una especie de mundo 3D, compuesto por millares de escenas 3D que al utilizar la tecnología propia de los videojuegos, incluye la interacción desde sus fundamentos. Otra propiedad que nos conviene de Second Life, es que se ejecuta y manipula desde y para Internet y que además, a diferencia de los videojuegos, su función no es lúdica, sino que no está definida, por lo que los usuarios pueden hacer en este mundo 3D lo que consideren oportuno, existiendo una especie de “anarquía” social de usuarios. Por lo que unos lo pueden usar para jugar, otros para mantener una red social, otros con vistas a la educación, etc. Además, trae consigo un compilador y un lenguaje de programación intrínseco. Es lo que se conoce como metaverso.

Con estos tres ejemplos, veremos, el abanico de aplicaciones que existen en la actualidad con los objetivos requeridos.

3. Interacción con gráficos 3D.

La interacción con gráficos en tres dimensiones conlleva la posibilidad de modificar la escena o mundo 3D. Como hemos visto anteriormente, existen aplicaciones que pueden generar escenas 3D pero no las puede modificar el usuario una vez terminadas. Sin embargo la tecnología actual permite la interacción con estas escenas 3D en tiempo real y la mejor prueba de ello son los llamados videojuegos, industria que mueve cada día más dinero en todo el mundo.

Normalmente las interacciones con el mundo 3D se realizan de dos formas básicamente. Por programación y/o por comportamientos predefinidos.

La programación supone la generación de código por parte de especialistas que producen elaborados programas que controlan y usan otros módulos del sistema 3D para realizar un trabajo colaborativo. Por ejemplo, existe un módulo en todos los videojuegos llamado máquina física encargado de calcular la física de los objetos virtuales 3D para que actúen como objetos reales. Este módulo es el encargado, por ejemplo, de que los objetos caigan cuando se les suelta a cierta altura, recreando así, la gravedad. Pues bien, si el programador del videojuego quiere que a la acción de un usuario (por ejemplo, que pulse un botón de gravedad 0), la gravedad, no tenga efecto, tendrá que escribir un código que desactive esta función, lo cual supone poseer amplios conocimientos técnicos.

Por otra parte, es posible que aquellas funciones que se repiten una y otra vez en casi todos los videojuegos, como pueden ser el movernos por el espacio tridimensional al pulsar ciertas teclas o el ratón, puedan ser preprogramadas y de esta forma solo necesitaremos decir con que tecla y a qué velocidad se llevará a cabo la acción de ir hacia delante. Es lo que se llama "comportamientos predefinidos" pero configurables.

Además de interaccionar con los objetos 3D como puede ser el entorno, los instrumentos, los personajes, etc., también podemos interaccionar con el sonido, las animaciones o el vídeo digital que aparezca en el interior de las escenas 3D.

Como hemos adelantado anteriormente, existen fundamentalmente dos tipos de aplicaciones que permitan esta "incursión" en mundos tridimensionales interactivos que son los las aplicaciones multimedia que incluyan máquinas 3D como Director, Unity 3D y otras y los metaversos que se pueden considerar como juegos sin un objetivo preestablecido. Lo importante en nuestro caso es que se pueda usar y configurar por nosotros sin necesidad de una alta cualificación como programadores.

Posiblemente, en un corto espacio de tiempo, tengamos en el mercado una multitud de estas aplicaciones, ya que las escenas 3D permiten apreciar mejor los detalles de muchos productos comerciales y pensando en Internet y la difusión de estas tecnologías será exponencial.

4. Publicación de multimedia 3D interactiva.

Relacionado con el tema de la difusión de los trabajos que pretendemos hacer, tenemos dos claras alternativas: la publicación "local" y la publicación en Internet.

Ya conocemos todos los problemas y ventajas que aporta la difusión de trabajos en la red de redes. A los aspectos de propiedad intelectual y difusión de la firma de autor, etc., se debe añadir aquí, temas tecnológicos, como la falta de plugins necesarios para la visualización de nuestros trabajos.

En los casos expuestos, el plugin para visualizar escenas 3D realizadas con Director es Shockwave que se encuentra instalado en el 49% de los ordenadores actuales, con lo que entre todos los competidores es el que encuentra en una posición más aventajada. Otro plugin con más aceptación todavía es Flash, pero no integra máquina 3D realista todavía por lo que no es una opción aceptable.

En cuanto a la difusión en modo "local", es decir, a través de los discos duros, las memorias pendrive,…, la solución puede ser más amplia, llegando incluso a la producción de ficheros autoejecutables que al ser procesados desde estas rápidas memorias no generan problemas de retardo en la interacción, con lo que la experiencia artística puede llegar a cotas bastante enriquecedoras, tanto en contenido como en interacción.

Unidad 2: Modelado.

1. Introducción.
2. Modelado de objetos básicos.
3. Modelado de objetos derivados.
    - 3.1. Modificadores.
    - 3.2. Objetos 3D a partir de formas 2D.
    - 3.3. Objetos booleanos.
    - 3.4. Otros métodos para modelar objetos derivados.
4. Técnicas de modelado.
    - 4.1. Modelado Poligonal o de superficies.
    - 4.2. Modelado NURBS.
    - 4.4. Modelado por programación.
    - 4.4. Otras técnicas de modelado.
5. Algunas aplicaciones para modelar.

1. Introducción.

A la hora de crear objetos tridimensionales tenemos dos modos básicamente:

- Definiendo su frontera: puntos, aristas, caras, superficies y sólidos.

- Definiendo su volumen: Enumeración y Octree de vóxels.

Los métodos para definir el volumen se basan en formar los modelos a partir de pequeños componentes o bloques llamados vóxel (imaginémonos los pequeños componentes del Tente o Lego) que pueden variar en tamaño, posición y orientación y que pueden ser cubos, esferas,... Mediante la composición de estos tendríamos formado el sólido.

Estos métodos permiten crear casi cualquier tipo de sólido. En contrapartida, estos métodos son complejos computacionalmente, por lo que no se emplean en los programas modeladores más conocidos actualmente.

Dependiendo del tipo de bloque y las técnicas para combinarlas, se pueden distinguir distintas técnicas para crear el sólido:

- Descomposición de celdas.

- Enumeración espacial.

- Árboles octales o Octrees.

Por lo tanto, en este trabajo nos centraremos en los métodos y aplicaciones que permiten crear objetos 3D definiendo su frontera.

La creación de una obra de este tipo en cualquier aplicación depende por supuesto, del proyecto a ejecutar. No es lo mismo realizar una obra en dos que en tres dimensiones. Esto es lo más importante, pero luego existen diferencias cómo el tipo de resultado que queremos abordar.

Por ejemplo, si queremos una obra de tipo arquitectónico un buen software puede ser Autocad, si es de tipo artístico podemos usar MAX, si queremos basarnos en métodos tradicionales de esculpir “con las manos” podemos usar Mudbox.

El modelado crea un objeto tridimensional digital. Este trabajo es hoy en día bastante especializado por muchas razones. Si se utilizan los métodos normales, es decir, programas que están diseñados para este propósito, la mayor dificultad radica en crear objetos de 3 dimensiones mediante teclado, ratón y pantalla. Moverse y trabajar en tres dimensiones con periféricos bidimensionales es intrínsecamente complicado. Incluso, si se emplean métodos más avanzados como la digitalización punto a punto o los escáneres 3D, el trabajo es para especialistas.

Nosotros trataremos de modelar con el primer método, o sea, con programas cuya funcionalidad está volcada en la creación de objetos 3D con ordenadores y periféricos para ordenadores personales.

Para crear objetos 3D, se pueden seguir muchos métodos. La clasificación de estos métodos es un tema muy delicado, ya que un objeto modelado puede haber sido construido utilizando distintas técnicas y procesos.

Para tratar de aportar la mayor claridad posible, clasificaremos el modelado según el criterio de la técnica empleada. Este criterio tiene la ventaja de que es indiscutible, es decir, la técnica empleada en un modelo hace que el resultado sea consecuencia directa de esta técnica y las propiedades de edición también. En realidad las técnicas vienen definidas por las matemáticas aplicadas al modelo, ya que estas técnicas son formas matemáticas distintas de definir los

mismos objetos 3D, aunque esto no quiera decir, que sean iguales, muy al contrario, cada una de ellas está recomendada para un tipo de modelado en especial.

Además, aquí no acaba la dificultad. Incluso si se tiene claro qué método seguir, muchas veces lo mas difícil es saber por donde empezar un trabajo de modelado. Por regla general se puede decir que cuando se acepta realizar un ejercicio de modelado, hay que estudiar bien el modelo, el o los métodos a emplear y el lugar por donde comenzar.

Clasificación de métodos y técnicas de modelado:

1. Modelado de objetos básicos.

2. Modelado de objetos derivados.

    2.1. Modificadores.

    2.2. Objetos 3D a partir de formas 2D.

    2.3. Objetos booleanos.

    2.4. Otros métodos para modelar objetos derivados.

3. Técnicas de modelado.

    3.1. Modelado Poligonal o de superficies.

    3.2. Modelado NURBS.

    3.4. Modelado por programación.

    3.4. Otras técnicas de modelado.

Hemos excluido del estudio final de este trabajo el modelado por programación (o por guiones como lo llama MAX) porque se utiliza más para la animación y es un trabajo hecho normalmente por programadores.

En esta clasificación se pueden distinguir las técnicas de mallas, nurbs y cuadrículas de corrección del resto debido a que este tipo de modelado se puede definir como contrario al paramétrico. Es decir, sería todo tipo de modelado que no se puede realizar mediante cuerpos geométricos regulares controlados por variables. Por ejemplo, pensemos en realizar el modelado de un árbol cualquiera. Con el modelado paramétrico utilizaríamos por ejemplo una esfera para la copa y un cilindro para tronco, pero podría parecer también un “chupa chups”. Si queremos algo más realista, tenemos que utilizar los distintos métodos no paramétricos y esto supone modificar a mano, no mediante variables, los puntos que forman los polígonos que a su vez forman las caras que componen la superficie de los modelos 3D.

Por todo lo anterior se puede llegar a la conclusión que para modelos idealistas son mejores los métodos paramétricos, pero si queremos modelar objetos y efectos realistas tendremos que utilizar técnicas no paramétricas en las que se introduzca la variabilidad, la no perfección geométrica.

Aparte del modo de modelar un objeto para poder realizar este tipo de tarea, se deben conocer funciones básicas para manejar los objetos. Estas funciones son comunes a todos los programas de modelado. Y son:

Imagen 4.2.1: Modelado paramétrico y no paramétrico

Manejo de vistas. Estas herramientas sirven para ver lo que deseamos en las cuatro vistas que disponemos por defecto: arriba, frontal, lateral y perspectiva. Hay herramientas para mover a mano la vista, para agrandarlas, para fijarnos en un solo objeto o para ver la escena entera. Cuando alguna de estas vistas se cambia por la de una cámara, estas herramientas cambian y aparecen otras propias de las cámaras.

Herramientas de selección de objetos. Estas herramientas son para seleccionar los objetos de la escena. Parece que sea algo sencillo y más teniendo 4 vistas de la escena pero nada más lejos de la realidad. Cuando la escena es medianamente complicada aparecen muchas dificultades.

Transformaciones básicas: mover, girar y escalar.

Simetría y alineación. Es imprescindible el poder situar el objeto no solamente en el punto indicado sino en la orientación deseada y con respecto a otros objetos de la escena, por ello, estas herramientas aunque no modifican el objeto, se pueden considerar indispensables.

2. Modelado de objetos básicos.

Cuando nos referimos a objetos básicos, queremos decir los objetos base, los objetos llamados primitivas de cada programa modelador, que como no puede ser de otra manera, son los objetos geométricos tridimensionales más usuales. A estas formas básicas se les conoce como modelos primitivas.

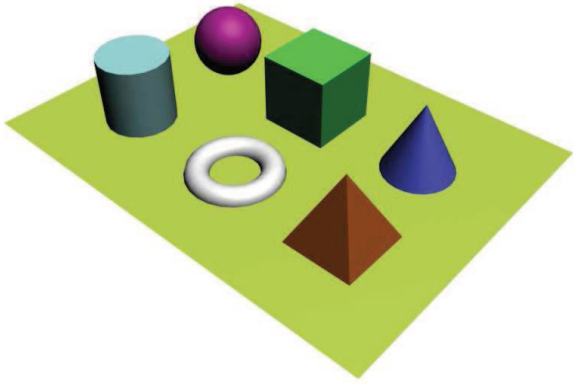


Imagen 4.2.2: Primitivas estándar: esfera, caja, cono,…

El modelado con primitivas, es aquel en el que se utilizan objetos cuya geometría viene dada por variables matemáticas. Por ejemplo, si queremos realizar una naranja “idealmente perfecta”, podríamos utilizar una esfera, cuya geometría se define por la variable radio y un punto central.

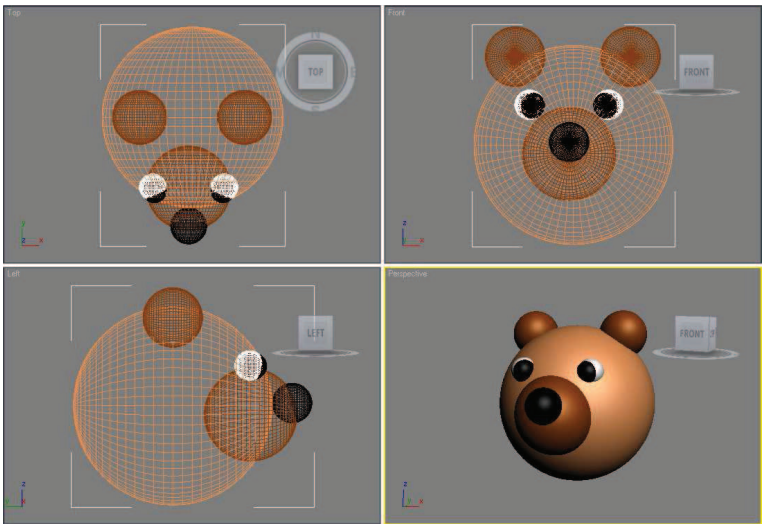


Imagen 4.2.3: Modelado a base de primitivas esfera

De todas formas es difícil que nuestro modelo quede realizado por una o pocas primitivas, por lo que normalmente se utilizan como base para después aplicar sobre ellas operaciones que las modifiquen.

Por ejemplo, 3D Studio MAX contiene 22 primitivas y en Second Life podemos crear todos los objetos que forman un mundo virtual partiendo de 15 primitivas.

En el caso de Second Life debemos destacar que entre sus primitivas se encuentran los árboles y la hierba. Esto es normal, ya que se necesitan para hacer paisajes.

3. Modelado de objetos derivados.

En este apartado vamos a estudiar formas de modelar un objeto a partir de otros ya creados con técnicas que pueden ser distintas. Por ejemplo, si quiero modelar un martillo, puedo crear una caja para la cabeza del martillo y un cilindro para el mango. Por lo tanto, no tenemos porque construir todo un objeto de una sola pieza o con una única técnica o de una sola vez.

Existen muchas formas de crear un objeto a base de otros, por lo que solo veremos las más importantes.

3.1. Modificadores.

Se trata de aplicar transformaciones geométricas a modelos tridimensionales.

Estas operaciones son bastante útiles y por nombrar algunas que comparten casi todos los programas incluidos MAX y Second Life tenemos: Taper para afilar, Bend para doblar hacia un lado, Skew para sesgar la geometría hacia un lado, Twist para retorcerla, etc.

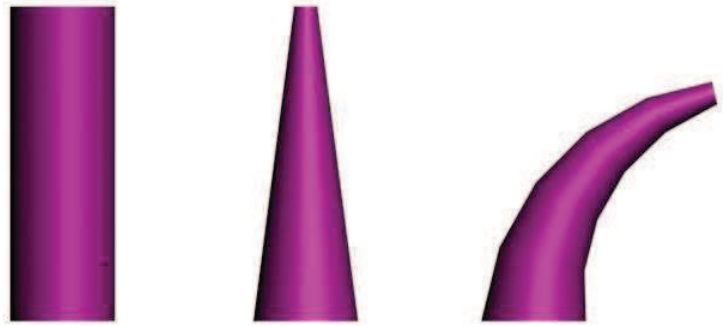


Imagen 4.2.4: Cilindro modificado con Taper y luego con Bend

3.2. Objetos 3D a partir de formas 2D.

Existen modelos 3D que se pueden crear mediante operaciones de objetos 2D. Los modelos creados con estas técnicas suelen ser modelos que representan a objetos fabricados por el hombre en la realidad, ya que responden en muchos casos a técnicas de fabricación. Por ejemplo, existen máquinas para hacer extrusiones, que es un método de modelado, también la técnica de revolución es aplicable en la vida real a los tornos de los alfareros. Las técnicas que vamos a estudiar se pueden reducir a dos aunque existen múltiples variantes de ellas, y son:

- Extrusión

Esta operación hace que una forma 2D que llamaremos perfil siga un camino 2D que llamaremos trayectoria. Por ejemplo: podríamos obtener un cilindro, a partir, de un círculo que sigue una trayectoria lineal.

En la siguiente imagen se puede observar una trayectoria con forma de S y una forma o perfil con forma de estrella y el resultado obtenido al crear una extrusión.

Imagen 4.2.5: Extrusión de una estrella sobre una trayectoria en S

Esta idea se lleva a su máxima expresión en los programas especializados en modelado 3D. Por ejemplo, la trayectoria puede ser cualquier “camino”, no tiene por que ser rectilíneo. Puede ser curvo, puede estar cerrado o no, o que todos sus puntos no estén sobre un mismo plano. Por ejemplo podríamos diseñar un perfil y hacer con un rectángulo un marco para un cuadro. También podemos hacer tuberías, cuerdas, manguitos, etc.

También cabe la posibilidad de que la forma cambie a lo largo de la trayectoria. Podemos hacer que el perfil comience siendo una estrella y se convierta en un círculo cuando lleve recorrido un 75% de la trayectoria.

Por otra parte, se puede modificar la forma en que el perfil recorre la trayectoria. Podríamos hacer que fuera girando sobre su eje y/o cambiando de escala. En la imagen siguiente se observa como un cuadrado que es la forma base va girando sobre su eje y cambiando de escala en distintos puntos de la trayectoria que en este caso es una recta.

Imagen 4.2.6: La forma puede girar y escalarse al recorrer la trayectoria

Como hemos escrito antes, la extrusión es un método de modelado que tiene su equivalente en el mundo real, es por ello, que es una forma muy sencilla de realzar figuras como pueden ser los textos al que dan cuerpo y que se utiliza mucho para crear logotipos y rotulación 3D en empresas.

- Revolución

Los objetos obtenidos por revolución son muy parecidos a los obtenidos por extrusión, pero en este caso, lo que queremos es que una forma 2D (normalmente no cerrada), gire total o parcialmente sobre un eje para obtener un elemento de sección circular. Por ejemplo, la copa de la imagen de abajo muestra estas posibilidades.

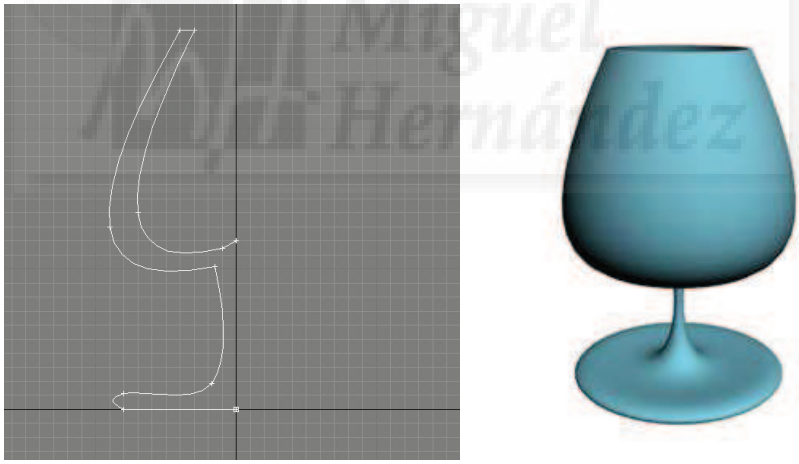


Imagen 4.2.7: Perfil y revolución obtenida sobre el eje vertical

Cómo se puede apreciar en la imagen de arriba este resultado se puede obtener en la vida real mediante la utilización de máquinas giratorias como tornos y fresadoras, por lo que debemos pensar en estas soluciones industriales para generar este tipo de modelos.

3.3. Objetos booleanos.

Los objetos booleanos se obtienen al aplicar las operaciones booleanas que son la unión, la intersección y la substracción de objetos tridimensionales.

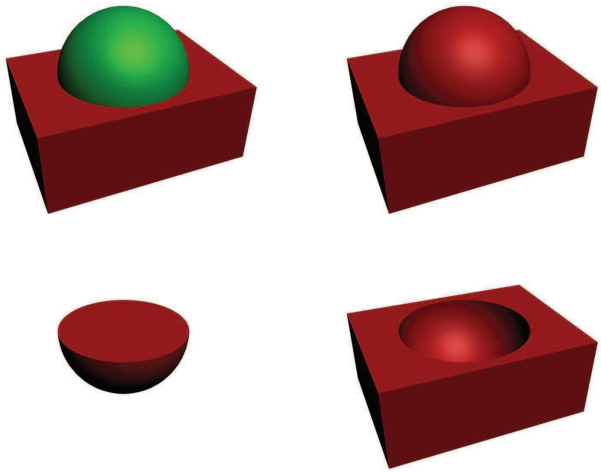


Imagen 4.2.8: Modelos con las tres operaciones booleanas aplicadas

En la imagen de arriba podemos ver de izquierda a derecha y de arriba hacia abajo, los dos objetos iniciales, la unión, la intersección y la substracción de la esfera verde sobre la caja roja.

3.4. Otros métodos para modelar objetos derivados.

Existen muchas técnicas para crear objetos derivados de otros como son la creación de grupos, la composición jerárquica de objetos que conlleva la organización estructural de estos con fines de animación, la generación de sistemas de partículas para recreación de efectos atmosféricos y otros.

El problema es que estos modos aunque prácticamente universales no se implementan en todo tipo de aplicaciones de creación de geometrías y además tienen fines que no se limitan al ámbito de los modelos 3D, como puede ser la creación de animaciones o efectos especiales por lo que no vamos a ahondar en ellos hasta ver casos concretos en las implementaciones que estudiaremos en profundidad en los siguientes capítulos.

4. Técnicas de modelado.

4.1. Modelado Poligonal o de superficies.

Es un método bastante antiguo pero que no está en absoluto obsoleto. Últimamente, además, con la aparición de los Nurms, está más de moda que nunca, ya que es un método muy útil tanto para iniciados como para expertos.

La geometría de una malla poligonal es una serie de triángulos (normalmente muy pequeños) que al interconectarse crean caras y estas forman el modelo 3D.

La forma del modelo no es manipulable por variables ya que es una colección de vértices, aristas y caras y el programa solo almacena dónde está cada vértice y como se interconectan.

Uno de sus éxitos es que este método permite un control explícito del modelo: cada vértice que se conecta con otro para componer una arista que se conecta con otras para componer cada cara del modelo se puede manipular independientemente.

El método de modelado suele empezar por una caja por ser el elemento más sencillo 3D. A partir de aquí se convierte en malla poligonal y seleccionando uno de sus subobjetos se empieza a modificar.

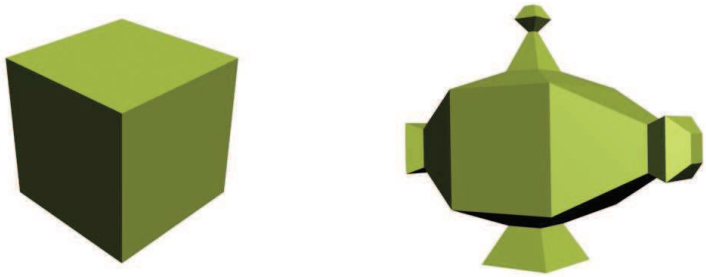


Imagen 4.2.9: Modelo poligonal creado a partir de la primitiva caja

Estas mallas poligonales se pueden animar fácilmente, ya que al saber dónde está cada vértice, el programa tiene un control exacto del modelo. Además al ser un método con muchos años, existen algoritmos muy productivos en su procesamiento, lo que la convierte en una forma de modelado para animaciones más rápidas que por ejemplo las nurbs.

El modelo poligonal se puede utilizar en principio para cualquier modelo. Siempre hablando en general, se recomienda para modelos no orgánicos, ya que es una característica intrínseca el trabajar con aristas, y las aristas marcadas casi no existen en la naturaleza. Por ejemplo para realizar cuerpos de animales, líquidos, etc. no es recomendable.

Las deficiencias de este modelo están en muchos casos compensadas por sus virtudes pero podemos citar por ejemplo, que esta tecnología no está orientada a modelos orgánicos, según el modelo, a veces cambiar un pequeño detalle lleva muchos esfuerzo por ejemplo por que buscar un determinado fallo en un vértice que estropea todo el modelo es muy difícil y también en el hecho de que si según crece el modelo, el rendimiento de MAX puede disminuir exponencialmente ya que no está pensada para modelos complejos.

Nurms

El método Nurms surge cuando se le aplica a un modelo poligonal un determinado modificador que redondea las aristas, por lo que se puede considerar un método incluido en el poligonal. Este no es un método independiente. En realidad se basa en el suavizado de un modelo poligonal mediante la aplicación de un modificador que suaviza las aristas de la malla, añadiendo muchas más caras. Es un método que a hecho que resurja con fuerza el modelado poligonal, ya que incluso se utiliza para los cuerpos orgánicos.

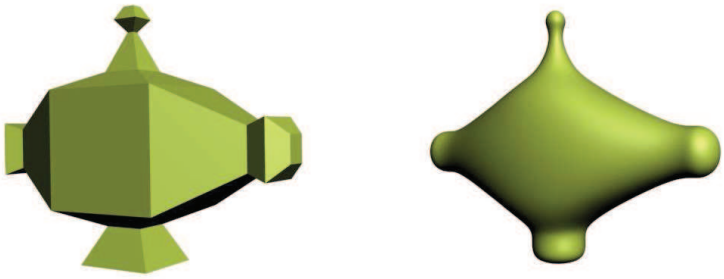


Imagen 4.2.10: Modelo poligonal y modelo Nurms

4.2. Modelado NURBS. Non-Uniform Rational B-Spline

Es un tipo de modelado con una filosofía y resultados totalmente distintos al modelado poligonal. Se puede decir que hay que cambiar de mentalidad para pasar del modelado poligonal al de nurbs.

En el modelado nurbs se parte generalmente de una primitiva que se convierte a nurbs y se pasan a utilizar unas herramientas especiales. Al igual que en el modelado poligonal, tenemos subobjetos, pero en vez de ser vértices son puntos de control que deforman la geometría. Lo

que se guarda del modelo son esos puntos de control. Los subobjetos que tenemos son los puntos o vértices, las curvas, y los distintos objetos tridimensionales que componen el objeto entero.

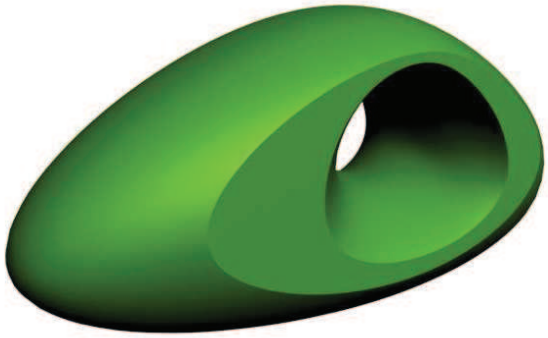


Imagen 4.2.11: Modelo Nurbs

Su uso más frecuente es para modelos objetos suaves o de aristas afiladas que se convierten o sales de curvas. Las superficies aerodinámicas son el campo preferido para aplicar Nurbs. En la imagen siguiente se puede apreciar como formas cóncavas y convexas suaves son perfectamente creadas con esta técnica de modelado. Las superficies nurbs necesitan los puntos de control. Estos puntos que pueden ser de dos formas:

Puntos: están en la propia curva y controlan directamente la forma de esta.

Vértices: forma una celosía de pesos gravitatorios independientes que modifican la curva estirando su superficie.

Se puede decir que utilizar los vértices es como modelar mercurio con imanes y modelar con puntos es como modelar arcilla con los dedos. Una cuestión muy importante es que se pueden modificar en número y “peso” los puntos, y esto es indispensable para dotar al modelo de distintos niveles de detalle. Esto hace que se pueda utilizar para modelos que requieran un nivel de detalle grande como pequeño.

Este modelado es especialmente flexible, y se puede utilizar tanto para superficies suaves como para superficies afiladas. Como hemos escrito antes, se guardan menos puntos de control que en el modelado poligonal por lo que su procesamiento requiere menos cálculo computacional, es decir, es mas rápida su visualización y modelado interactivo.

Por todo lo expuesto anteriormente sus usos son para formas orgánicas como personajes, animales, etc. También se utiliza para elementos ergonómicos como gafas, sillas, etc. por la misma razón. Un uso muy importante es en el modelado de equipos y maquinas aerodinámicas como pueden ser coches, aviones,…En cuanto a la animación podemos decir que obtiene mejores resultados que la poligonal, mas suaves, aunque se necesita mas tiempo para calcularlo ya que la mayoría de los algoritmos actuales para animación están más estudiados y son específicamente diseñados para modelos poligonales. Si tenemos que buscarle un pero a esta técnica es la simplicidad, dado que la técnica en si misma está diseñada para poder crear modelos muy complejos. Existen programas específicamente enfocados al modelado con Nubs, entre ellos debemos destacar Rhinoceros®.

4.3. Modelado por programación.

En muchos programas de modelado se pueden construir y editar modelos. Por ejemplo, en MAX existe un función llamada MAXScript, en Director el lenguaje Lingo y en Second Life existen LSL.

Con estos lenguajes podemos crear geometría en aquellos problemas en que sea más fácil describir una escena mediante sentencias que por otros métodos. Por ejemplo, si tenemos que crear un ejército de soldados, es más sencillo que podamos escribir que necesitamos 2000

soldados colocados aleatoriamente dentro de unos límites con una altura y grosor también calculados que no hacerlo por los métodos habituales.

Otro modelado típicamente resuelto mediante programación es la creación y manipulación de sistemas de partículas. Por ejemplo, si queremos crear una escena con lluvia, a nadie se le ocurre dibujar las gotas una a una. En muchos programas podemos utilizar utilidades para realizarlos pero en otros solo las podremos utilizar directamente con sentencias de programación como en el caso de Second Life. Además no debemos pensar que tenemos que tener amplios conocimientos de programación o matemáticas para adentrarnos en este mundo, ya que los lenguajes para llevar a cabo esta faena están orientados al modelado y no son de propósito general. De hecho, muchos no son verdaderamente lenguajes de programación sino de macros, o sea, funciones de más alto nivel que los lenguajes de programación generales. De todas formas es cierto que es un sistema adecuado para generar modelos muy numerosos o por ejemplo objetos fractales pero es menos intuitivo que otros métodos como el poligonal.

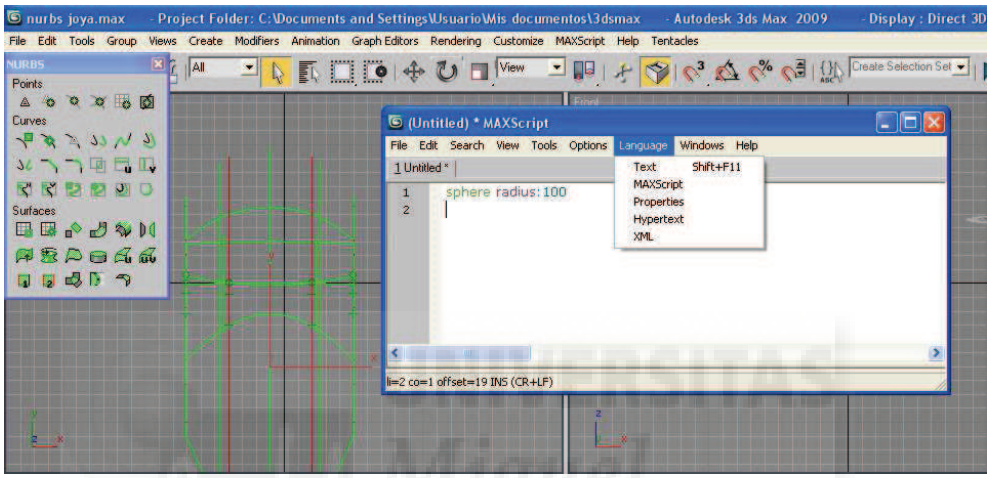


Imagen 4.2.12: Ventana para crear Scripts en MAX

4.4. Otras técnicas de modelado.

Existen muchas técnicas de modelado que varían de unas aplicaciones a otras, pero por citar alguna de las más utilizadas podemos nombrar el modelado de correctores.

El modelado de correctores es una técnica nativa de 3D Studio MAX® y al ser este programa tan ampliamente utilizado es una técnica muy conocida. Los correctores son más recientes que los polígonos y las nurbs. Los modelos realizados con correctores tienen características que la sitúan a medio camino entre el modelado poligonal y el modelado NURBS, ya que sirven sobre todo para realizar superficie curvas y con cambios suaves como las nurbs y sin embargo son menos pesados para el procesador que la poligonal. Es un método muy utilizado para modelar personajes y otros modelos orgánicos. Una superficie de tipo corrector está compuesta por superficies más pequeñas que están definidas por sus límites y curvas Beziers en su interior.

11. Algunas aplicaciones para modelar.

Existen muchas aplicaciones para modelar en Windows. En el siguiente cuadro se pueden comparar algunas de las más conocidas como cinema 4D, 3DS MAX, Softimage XSI, Maya y TrueSpace.

	trueSpace7.6	CINEMA 4D R10		3DS Max	SOFTIMAGE XSI		Maya	
		CORE	Studio		Foundation	Advanced	Complete	Unlimited
Fully Customisable Interface	•	•	•	•				
Multiple Rendering Engines	•			•			•	•
Rendering (GI, HDRI, caustics)	•		•	•	•	•	•	•
Photo-realistic workspace	•							
Modeling (Poly, SDS, NURBS)	•	•	•	•	•	•	•	•
Surfacing (UV editor)	•	•	•	•	•	•	•	•
Advanced 3D Paint		•	•				•	•
Normal and Displacement tools	•	•	•	•	•	•	•	•
Hair / Fur	•		•	•		•		•
Dynamic Hair / Fur			•	•		•		•
Non-linear Animation Tools	•	•	•	•	•	•	•	•
Skeletons with full body IK	•	•	•	•	•	•	•	•
Interactive Physics characters	•							
Fluid Dynamics and Cloth			•	•	•	•		•
Motion Data Import	•	•	•	•	•	•	•	•
Procedural Animation	•	•	•	•	•	•	•	•
Generalized Node Editor	•							
Undo With Construction History	•	•	•	•	•	•	•	•
Support for Multi-core CPU's	•	•	•	•	•	•	•	•
Real time 3D Collaboration	•							
Free Interactive Web Viewer	•							
	Free	$895	$3,495	$3,495	Discontinued	$4,995	$1,999	$6,999

Tabla 2: Comparación de algunas aplicaciones de modelado 3D

El problema de comparar estas aplicaciones se encuentra en que muchas están enfocadas a resolver problemas específicos, así por tanto, se pueden dar casos en principio sorprendentes como 3DS MAX y Maya que pertenecen a la misma empresa Autodesk® pero que tienen distintos nichos de mercado. Maya está más enfocada al modelado y animación de personajes y MAX es más generalista.

Unidad 3: Materiales: superficies y texturas.

3.1. Superficies, materiales y texturas.

3.2. Superficies estándar.

3.3. Superficies mapeadas o texturizadas.

3.4. Otras superficies.

3.1. Superficies, materiales y texturas.

Lo primero que tenemos que escribir es que tener un control de estos aspectos es una de la bases para conseguir dominar los programas de edición 3D.

Los conceptos de superficies o materiales son equivalentes. Aunque tienden a la confusión, lo que llamamos superficies en unos programas, entre ellos Second Life, en otros como 3D Studio MAX se llaman materiales.

Somos de la opinión que el nombre más adecuado es el de material, ya que el término superficie se puede llegar a confundir con el que se utiliza para describir la parte exterior de los objetos compuestos por polígonos. Por tanto a partir de este momento utilizaremos el término superficie como objeto geométrico y material como una entidad que se aplica sobre una superficie y que la dota de propiedades estéticas y por tanto define como se visualiza.

Las texturas son imágenes que junto con otros datos pueden crear un material más o menos sofisticado y que se pueden aplicar para definir el aspecto básico, niveles de opacidad, irregularidades en la superficie y otros efectos. A las texturas también se les conoce como mapas.

Pongamos un ejemplo. Si tenemos una esfera, en realidad lo que tenemos es una especie de cáscara que no tiene grosor y por tanto la esfera no es sólida sino completamente hueca, pero nosotros la visualizamos cómo un sólido ya que solo la vemos desde el exterior. Esta cáscara sería la superficie. Si aplicamos un material que venga definido por un color, podremos pintar la esfera del color que queramos, por ejemplo, azul. Pero si el material incluye una textura que sea una fotografía de una madera, la esfera se visualizará como se fuera realmente del material madera.

Todos los programas de edición 3D permiten definir superficies, materiales y aplicar texturas pero cada uno con un nivel distinto de sofisticación y realismo.

Los materiales son entidades complicadas de crear ya que pueden llegar a contener información sobre colores, brillos, lustres, mapas de distinto tipo, además pueden combinarse con otros y tener submateriales, forma que absorbe la luz, elementos reflejados, etc. esto hace que sea de vital importancia reutilizar los materiales, por ejemplo, si conseguimos un material dorado y somos orfebres, desearemos reutilizar el mismo material en muchos objetos distintos.

Por todo esto, muchos programas incluyen utilidades para editar y gestionar materiales que funcionan de forma más o menos autónoma.

Además un determinado modelo puede tener aplicados distintos materiales en las distintas caras que lo componen, por ejemplo, si tenemos un cilindro y queremos representar un cigarro, podemos aplicar un material para la boquilla en unas caras, un material para el papel en otras y un material tabaco para otras.

Los materiales son muy importantes, pero casi más importante que estos es la iluminación, ya que si no la trabajamos bien puede arruinar un gran esfuerzo de materiales. Por ejemplo una luz verde, tintará de este color todo lo que esté a su alcance o si no hay suficiente luz en una habitación no veremos los materiales tal y como se diseñaron.

Si intentamos clasificar los materiales, podemos distinguir según sus componentes tres tipos, a saber:

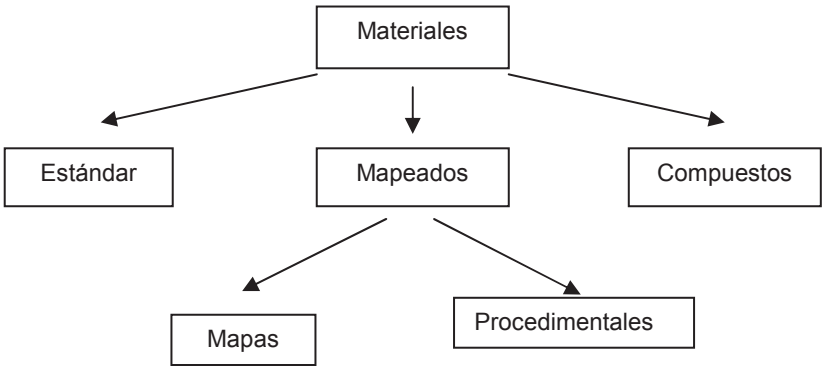
1. Estándar: vienen definidos solo por sus propiedades físicas y de luz.

2. Mapeados: incluyen imágenes para definir distintas propiedades como textura, opacidad, transparencia y otras.
2.1. Materiales procedimentales: contienen imágenes creadas y generadas por programación, por lo tanto son imágenes creadas por una aplicación.

2.2. Materiales con mapas: contienen imágenes fotográficas o modificadas por aplicaciones que parten de una fotografía para modificarlas.

3. Compuestos: los que incluyen otros materiales.

En el siguiente esquema se trata de ilustrar esta clasificación.



Gráfica 1: Tipos de materiales

3.2. Superficies estándar.

Como hemos escrito anteriormente las superficies estándar vienen dadas por sus propiedades físicas y cómo responden a la luminosidad. Por lo tanto no contendrán otros materiales ni utilizaran recursos de imágenes.

Vamos a estudiar los valores básicos que definen estas superficies.

Sombreado: es la más importante ya que define el tipo de material según su naturaleza y cómo responde a la luz. Por ejemplo, un tipo de sombreado puede ser el metálico que creará brillos alargados, o el mate que carecerá de un brillo definido. Estos sombreados limitarán los valores que puedan tomar el resto de variables. Los programas de edición 3D normalmente utilizan al menos 5 tipos básicos de sombreado.

Color difuso: es el color propio del material, el color base. Según los distintos tipos de sombreados se verá más o menos cantidad de este color. Por ejemplo, si tenemos un objeto rojo metálico, los brillos de la superficie serán muy claros, casi blancos y las sombras serán casi negras por lo que el color rojo se verá menos que en un material mate donde las zonas muy iluminadas y las sombreadas no se diferenciarán mucho del color base.

Color especular: es el color que tomarán los brillos del objeto. Normalmente son colores muy claros pero podemos utilizar cualquiera.

Color ambiental: es el color que tomarán las zonas más oscuras del objeto, normalmente son colores cercanos al negro.

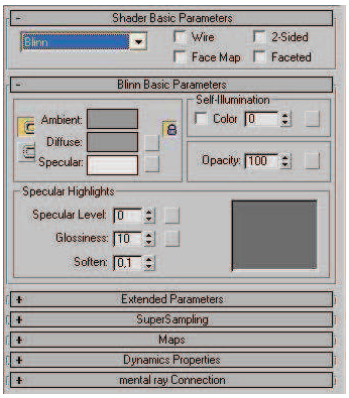


Imagen 4.3.1: Parámetros básicos de los materiales estándar en MAX

Filtro: color de la luz que pasa a través del objeto si este no es totalmente opaco.

Brillo: es muy importante para ciertos materiales y viene dado por cuatro variables:

1. Color especular: el color del propio brillo.

2. Curva de máximo brillo: define la forma del brillo.

2.1. Lustre: modifica la anchura de la curva.

2.2. Nivel especular: modifica la altura de la curva.

3. Debilitar: con valores extremos, suaviza bordes del brillo para más realismo.

Opacidad: se utiliza para controlar la transparencia de materiales como vidrio o agua. Es transparente en una mediad de 0 a 100. La total opacidad será con el valor 100. Hay que tener en cuenta una serie de detalles:

1. El color de la luz que atraviesa el material viene dada por el Filtro.

2. Como se debe "ver" a través del material, se activará la opción "2 lados".

3. Para trabajar con materiales transparentes mejor pulsar el botón "Fondo".

4. Para un mayor control de cómo la luz atraviesa el material, ir a parámetros extendidos > Opacidad. Tenemos opciones como atenuación, tipos de opacidad.

Autoiluminación: un material no puede emitir luz. Si queremos crear un objeto luminoso como una bombilla, necesitamos 3 cosas, la geometría del objeto, una luz y un material asignado con autoiluminación. Viene dado por un color y un valor de intensidad.

Alambre: sirve para mostrar el objeto en modo alámbrico, es decir, mostrando las aristas de sus caras. Como se debe "ver" a través del material, se activará la opción "2 lados". En parámetros extendidos > alambre tenemos más opciones, como la anchura de los alambres, etc.

3.3. Superficies mapeadas o texturizadas.

Mapear es un método de proyectar los materiales sobre la superficie de un objeto, que la mayoría de las veces será tridimensional. Es como envolver un regalo con papel de envolver. Para obtener imágenes realistas, se tienen que tener en cuenta los siguientes puntos:

Obtener mapas de "texturas" realistas mediante cámaras digitales, escáneres, programas de dibujo, etc.

Dentro de este apartado existen distintos aspectos que hay que abordar como el mismo concepto de mapa, cómo asignar un mapa, tipos de mapas, cómo trabajar con mapas, modificadores de mapas, etc. Algunos de estos aspectos son muy prácticos por lo que se estudiarán en el curso práctico.

Concepto de mapa: los mapas son imágenes que se asignan a un material para modificar sus características básicas de simulación física y respuesta a la luz pero que además se pueden utilizar para simular la modificación de la geometría del objeto como rugosidades, relieves, etc. Las imágenes pueden ser de los tipos más utilizados en Windows como .jpg, .bmp,...

Se puede decir que hay dos tipos de mapas: de procedimiento y de proceso.

Los mapas de procedimiento crean una textura pero no a partir de una imagen, sino de cálculos matemáticos. Por un lado no permite resultados tan detallados como una imagen pero por otro lado no se puede nunca llegar a pixelizar la imagen, ya que este material siempre se adaptará a los parámetros de la escena. Este tipo de mapas se utilizan para materiales naturales y un tanto “caóticos” como salpicaduras, maderas con vetas, etc.

Los mapas de proceso se utilizan para definir el material con imágenes, ya sea para modificar sus aspectos básicos como otras características (relieve, opacidad, etc.) Las 12 características comunes a todos los materiales se llaman Canales de mapas. Todos los mapas se asignan desde la persiana mapas del material. Existen 4 columnas que corresponden a la verificación de asignación, la característica del material, la cantidad y la imagen asignada.

Para modificar las características básicas tenemos: mapa ambiental, mapa difuso y mapa especular. Estos conceptos se explicaron anteriormente. Normalmente solo se modificará el mapa difuso para “pintar” un material con una imagen. El valor de la cantidad es normalmente porcentual, es decir, varía de 1 a 100.

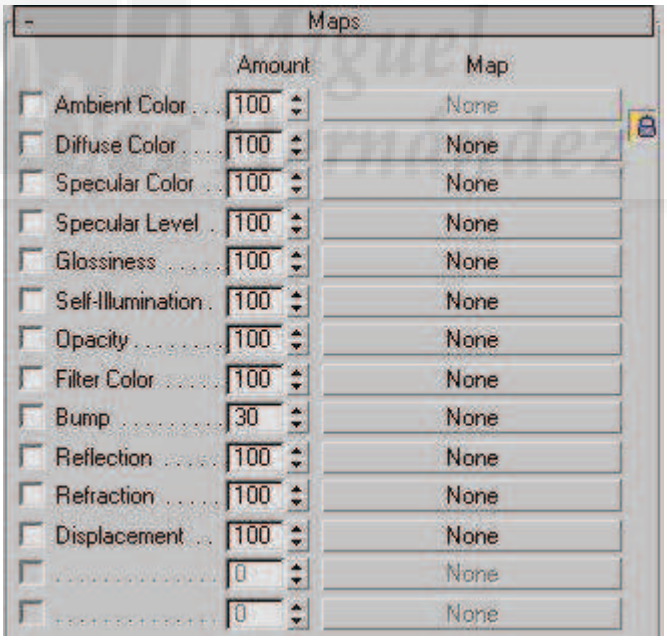


Imagen 4.3.2: Mapas para los materiales estándar

Mapas para otras características: el resto de los mapas trabajan de otra manera, no pegan la imagen, sino que extraen de ella, la cantidad de blanco y negro que contienen para crear una imagen de escala de grises, que por tanto es el tipo de imagen que se debería de asignar. Las características que se simulan son:

Nivel especular: El valor de la cantidad es entre –999 y 999. Esta característica está en relación a la opacidad y sirve para evitar los brillos, precisamente en las zonas transparentes, pero sí los mantiene en las zonas opacas.

Lustre: El valor de la cantidad es porcentual. Ya explicada anteriormente.

Autoiluminación: El valor de la cantidad es porcentual. Ya explicada anteriormente.

Opacidad: El valor de la cantidad es porcentual. Es muy útil tanto para simular materiales transparentes como para superficies planas que necesitan la transparencia en el contorno. Por ejemplo para hacer simulaciones de hojas de plantas, de personas en medios virtuales, etc.

Color de filtro: El valor de la cantidad es porcentual. Aplica una imagen al color de tinta producida por un objeto con transparencias. Tanto el objeto como las sombras que proyecta quedan coloreadas con el mapa.

Relieve: El valor de la cantidad es entre –999 y 999. Permite efectos de tridimensionalidad sin modificar la geometría de los objetos. Los colores oscuros del mapa se interpretan como más profundos y los más claros como de relieve. Esto se modifica si los valores son negativos. Un problema añadido, es que al proyectar las sombras de un objeto, no se tiene en cuenta este mapa de relieve, por lo que si es muy exagerado su valor, la “trampa” se aprecia a simple vistas.

Reflexión: El valor de la cantidad es porcentual. Para tener reflexión en un objeto se pueden utilizar dos técnicas. La primera se hace “artificialmente”, pegando en el objeto una imagen de lo que debe reflejar. Por ejemplo la misma imagen de un cielo se puede utilizar para que sea reflejado en un estanque. La segunda técnica supone más cálculo para el ordenador en el render. Se trata de que se tenga en cuenta “de verdad” el resto de objetos de la escena y utilizar un cálculo matemático de reflexión como RayTrace o simetría plana. La ventaja es su mayor realismo.

Refracción: El valor de la cantidad es porcentual. Es como la reflexión pero con el efecto de distorsión que producen los objetos transparentes. Por ejemplo lo que se ve a través de una botella. Se puede utilizar también el Raytrace o la refracción de cristal con las mismas similitudes que las vistas para la reflexión.

Desplazamiento: El valor de la cantidad es entre –999 y 999. Es muy similar al modificador Desplazar. Se puede utilizar en todo tipo de geometrías incluida la nurbs. Lo que produce es un cambio aparente en la altura por zonas de la geometría.

Cuando se pulsa un botón para asignar un mapa aparece la ventana Material/Maps Browser. Si se quiere asignar una imagen elegiremos Bitmap, las otras opciones son para crear mapas procedimentales.

Algunos de ellos son por ejemplo: Gradient, degradado. Gradient Ramp, rampa de degradado. Checker, cuadros. Swirl, remolino. Falloff, atenuación. Cellular, celular. Stucco, estuco. Noise, ruido. Smoke, humo. Marble, mármol. Perlin Marble, mármol perlín,. Planet, planeta. Mask, Máscara. Mix, mezcla. Flat mirror, simetría plana. Raytrace, traza de rayos.

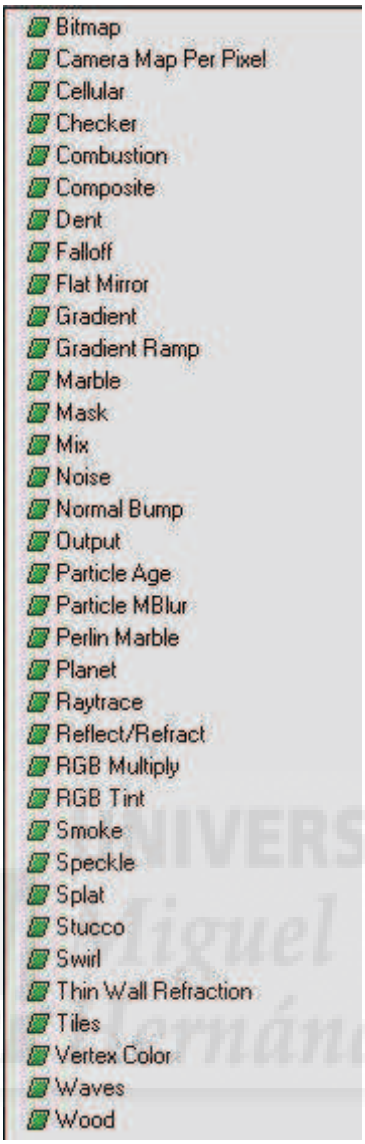


Imagen 4.3.3: Mapas procedimentales en MAX

4. Otras superficies.

4.1. Superficies con materiales compuestos.

Se utilizan materiales compuestos cuando un objeto necesita más de un material. Por ejemplo pensemos en un jarrón boca muy ancha y de barro decorado. Tenemos un único objeto, una única geometría, sin embargo, para poderlo manipularlo en animaciones, no basta con asignarle un material decorado en el exterior del jarrón, sino que se debe texturizar el interior con barro ya que algunas vistas se verá parte de ese interior y no digamos si el jarrón debe romperse a lo largo de la película.

En MAX tenemos muchas posibilidades de materiales compuestos que son los que aparecen en la Imagen 3.4.5.

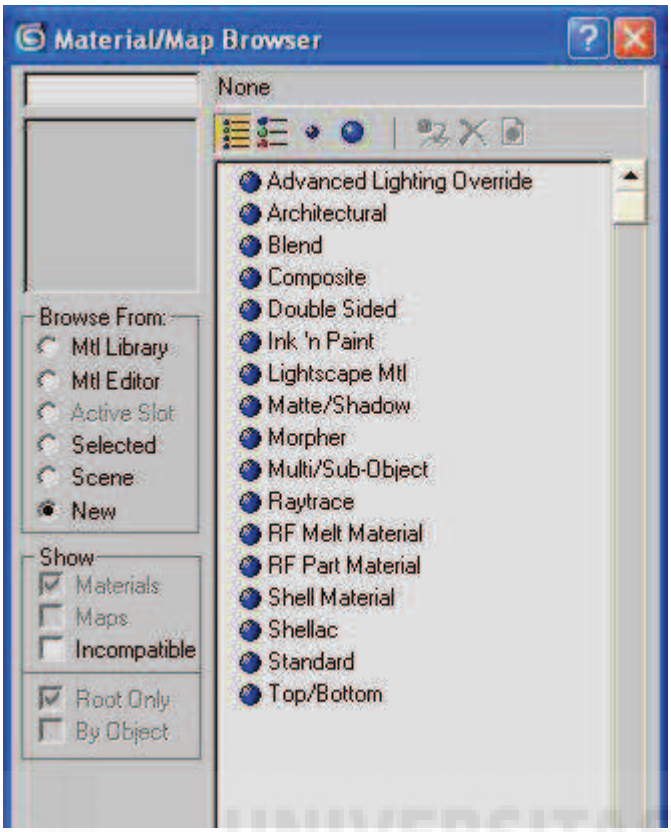


Imagen 4.3.4: Materiales compuestos

1. Composición Multi / Subobjeto: materiales diferentes en cada cara. Contiene una lista numerada de materiales que pueden asignarse independientemente a diferentes partes de un objeto, mediante la selección de ciertas caras, la asignación de un id de material para ellas y luego hacer coincidir ese nº id de material con el de la lista de este material compuesto. El nº máximo es de 1000.

2. Composición Superior / Inferior. Tiene solo 2 submateriales, según la normal de una cara apunte hacia arriba o hacia abajo se le asignará uno u otro material. Se tiene un control llamado mezcla para la obtener una fusión más o menos nítida entre los dos materiales. También se puede controlar la posición de la línea de fusión con respecto a la geometría. 50 significa que estará situada en la mitad del objeto.

3. Composición 2 Lados. Tiene también 2 submateriales, según las normal de una cara apunte hacia el exterior o hacia el interior se le asignará uno u otro material, llamados anterior y posterior. Se tiene un control llamado translucidse para que se vean simultáneamente en un porcentaje.

4. Composición Mezcla. También tiene 2 submateriales (material 1 y material 2) que se aplican al objeto a la vez pero según uno de estos criterios:

a) Una máscara. La máscara es o una imagen en escala de grises o una curva de mezcla. Si es una imagen hace que el material 1 deje paso al material 2 cuanto más blancas sean las zonas. Si se utiliza la curva de mezcla se controlará mediante un gráfico.

b) Cantidad de mezcla. Se mezclaran los dos en un tanto porcentual.

5. Composición Compuesto. Es parecido a la opción de mezcla pero permite hasta 10 materiales que se van superponiendo, partiendo siempre de un material base. Cada uno de los restantes se pueden mezclar de forma aditiva con respecto a su opacidad, de forma sustractiva o con relación a una cantidad.

6. Composición Shellac. Shellac significa barniz. Se trata de la composición de dos materiales por la superposición de los colores de ambos. Es un truco muy utilizado en programas de pintura digital para dar aspecto de barnizado a un objeto por ejemplo de madera.

7. Composición Mate / Sombra: imágenes reales + imágenes sintéticas 3D. Es un tipo de material muy utilizado para mezclar imágenes reales con imágenes de síntesis. Esto se puede hacer porque el material que se designa como mate no se representa, pero sí que se representa si recibe sombras o reflejos de otros materiales.

8. Material Raytrace. Como el mapa raytrace, se utiliza para conseguir tanto reflexión y refracción de gran calidad. Se utilizan cálculos de trazados de los rayos de luz como translucidse y florescencia. Utilizan muchos parámetros básicos que tienen los mismos nombres que el material estándar pero que no se utilizan de la misma forma.

Unidad 4: La iluminación en las escenas 3D

4.1. La luz y el color en los objetos.

4.2. Modelo de iluminación simple de Lambert.

4.3. Sombras.

4.4. Iluminación de una escena.

4.5. Tipos de fuentes de luz.

4.6. Atributos de las fuentes de luz.

4.7. Ejemplos de iluminación.

4.8. Iluminación de un exterior.

4.9. Filtros para sombras arrojadas.

4.1. La luz y el color en los objetos

Todos los objetos poseen unas propiedades que caracterizan su aspecto. Estas propiedades hacen que los objetos posean una forma definida pero no puede visualizarse sino tenemos una luz que incida sobre su superficie. Por lo tanto, es de vital importancia en una escena 3D que existan fuentes de luz que permitan mostrarla.

La generación por ordenador de las propiedades exactas de las formas y propiedades físicas de los objetos es un aspecto inabordable en la actualidad como ya estudiamos en la parte de modelado de objetos 3D. Esto se soluciona mediante aproximaciones no exactas pero con muy buenos resultados realistas. Una de las propiedades fundamentales para la visualización de un objeto es su color.

El color que muestra un objeto, es también una consecuencia de sus propiedades físicas y de las combinaciones de luz o carencia de ella que incide sobre el objeto y desde el punto de vista del espectador.

Los programas que presentan escenas 3D, permiten generar unos colores realistas en la superficie de los objetos basados en tres conceptos: el color difuso, el especular y el ambiental. Estos colores se basan en el modelo de iluminación de Lambert.

Imagen 4.4.1: Iluminación diurna y nocturna de una escena 3D

4.2. Modelo de iluminación simple de Lambert

Los modelos de iluminación, tratan de definir por aproximación los factores que explican la forma y colores que presentan los objetos ante fuentes de luz. De esta manera, tratan de resolver el sistema de la iluminación para crear escenas 3D de síntesis.

Se dice que un modelo es exacto cuando en él intervienen todas las propiedades físicas que le incumben y se calcula con exactitud. Como hemos escrito antes, en la actualidad no es posible realizar estos modelos salvo para casos de escenas tridimensionales excepcionalmente sencillas y teóricas. Por ello, todos los modelos usados por distintas aplicaciones son modelos aproximados. El resultado sin embargo, es muy realista y es cuestión de precisión y cálculo computacional, que no se pueda distinguir entre imágenes reales o de síntesis incluso entre expertos en la materia. Se supone que un modelo de iluminación es mejor que otro, cuantas más propiedades físicas intervengan en su formulación.

El modelo de Lambert calcula el color en un punto individual de la superficie de un objeto 3D considerando las siguientes propiedades físicas:

1. La cantidad de la luz ambiente que refleje.
2. La cantidad de la luz directa que refleje.
3. La orientación de la superficie hacia la fuente de luz.
4. La cantidad de luz directa que refleje en una determinada dirección.
5. La posición del observador (la cámara).

Según las propiedades físicas que imite la superficie de un objeto, tendremos que utilizar una serie de coeficientes que en relación con la iluminación definen esa superficie, y son:

1. Reflexión ambiente: comportamiento de la superficie del objeto ante la iluminación que proviene del exterior y/o de la interreflexión de la luz con el resto de los objetos de la escena.

2. Reflexión difusa: comportamiento de la superficie opaca ante la iluminación que proviene de las fuentes de luz de la escena. El color mostrado por el objeto variará de una parte a otra dependiendo de la dirección y distancia con respecto a la fuente de luz.

3. Reflexión especular: comportamiento de las superficies reflectantes ante la iluminación que proviene de las fuentes de luz existentes en la escena. La posición e intensidad de un brillo depende de la posición de la fuente de luz que incide.

4. Parámetro de reflexión especular: comportamiento de la superficie reflectante en cuanto al tamaño del brillo que se produce como consecuencia de la incidencia de una fuente de luz.

4.3. Sombras

Las sombras son zonas de la escena 3D que quedan en cierta oscuridad por que la luz queda obstaculizada de algún modo. Ocupa un espacio tras un objeto opaco con una fuente de luz frente a él. La llamada sección eficaz de una sombra es una silueta bidimensional o proyección invertida del objeto que bloquea la luz. Por lo tanto es un resultado natural que se produce al iluminar una escena. Esto quiere decir que como consecuencia normal de iluminar una escena 3D, tendremos la generación de sombras en parte de los objetos que la componen. Existen dos tipos de sombras:

1. Sombras propias: originada por la posición y orientación de la superficie del objeto respecto a las fuentes de luz existentes en la escena. Esto tiene la consecuencia de que se iluminan las partes orientadas hacia la fuente de luz y se oscurecen las que no lo están.

2. Sombras proyectadas: Es la penumbra o zona oscurecida sobre la superficie de un objeto cuando otro objeto se interpone entre él y la fuente de luz.

4.4. Objetivos de la iluminación de una escena

Al iluminar una escena podemos conseguir los siguientes objetivos:

• Hacer visible la escena, ya que debe haber al menos una fuente de luz que permita apreciar los objetos.

• Dar volumen a los objetos para poder apreciar el relieve de los objetos y la sensación de profundidad.

• Fijar la atención en partes importantes si damos protagonismo a una determinada parte iluminándola especialmente.

• Cuando iluminamos personajes o determinadas escenas, podemos expresar una emoción

• Indicar una hora del día simulando la luz solar a una determinada altura.

4.5. Tipos de fuentes de luz.

La mayoría de los programas de diseño y visualización 3D disponen de cuatro tipos de fuentes de luz:

• Luz ambiente: luz existente en la escena debido a las infinitas reflexiones que se producen entre los objetos de la misma.

• Luz puntual: fuente de luz situada en un punto del espacio y que emite rayos en todas las direcciones (omnidireccional).

• Luz focal: fuente de luz situada en un punto del espacio y que emite rayos en unas direcciones concretas y agrupadas.

• Luz direccional: representa una fuente de luz situada en el infinito cuya emisión de luz se realiza en una misma dirección. Se usa para simular la luz solar.

4.6. Atributos de las fuentes de luz

a) Posición: específico de las luces dirigidas y puntuales (omnidireccionales) que indican el lugar del espacio (coordenada 3D) donde está situada la fuente de luz.

b) Intensidad: atributo aplicable a cualquier fuente de luz que simula la potencia de emisión. Se suele ajustar con un valor entre 0 y 100.

c) Dirección: Específico de las luces dirigidas (focal y direccional) que indican el objetivo de la iluminación o motivo central a iluminar.

d) Cobertura: específico de una luz focal que indica el grado de alcance de la fuente (radio del cono o cilindro de luz).

e) Atenuación o caída: factor que mide la disminución de intensidad de la luz con la distancia.

f) Color: atributo fundamental que muestra la terna RGB de emisión de la fuente.

g) Sombras: al contrario que en el mundo real, podemos indicar que no calcule las sombras proyectadas al iluminar.

4.7. Ejemplos de iluminación

• Iluminación frontal: los objetos parecen planos, sin volumen.

• Iluminación fronto-lateral: mezcla de luz y sombra propia en los objetos dándoles volumen.

• Iluminación lateral: crea unas sombras propias muy marcadas resaltando las arrugas de la superficie.

• Iluminación trasero-lateral: deja la mayor parte del objeto oscurecido por su propia sombra, se utiliza para crear un efecto de misterio.

• Iluminación trasera: deja todo el objeto oscurecido y muestra solamente su silueta.

• Iluminación desde abajo: crea unas sombras muy marcadas en la parte alta de los objetos. Se utiliza para crear un efecto de terror.

• Iluminación completa: es una iluminación con tres luces. Una luz fronto-lateral que ilumina el objeto, otra luz para aclarar las sombras que crea la primera pero sin eliminarlas y una luz trasero-lateral que crea una línea de luz en los objetos que los remarca y los separa del fondo, el cual normalmente, dispone de otra luz que lo ilumine.

4.8. Iluminación de un exterior.

Se usa una única fuente de luz que puede ser el sol o la luna y de tipo dirección. Aunque sabemos que el sol emite luz en todas direcciones y por tanto llegan a la tierra desde distintos

ángulos, es tal la distancia que nos separa de nuestra estrella más cercana, que podemos imitarla como si se tratase de una gran foco y que en la escena los rayos de luz serán paralelos.

Imagen 4.4.2: Iluminación diurna de una escena exterior

Según el ángulo de la luz que imita al sol, podemos indicar:

La hora: el amanecer, atardecer, mediodía si lo ponemos a mucha altura, etc.

El tiempo: si las sombras son duras indicará tiempo soleado, si son difuminadas y bajamos la intensidad simularemos un día nublado.

4.9. Filtros para sombras arrojadas.

Estos filtros se emplean para iluminar una escena 3D y simular objetos que en realidad no existen en dicha escena. Por ejemplo en el delfín de la imagen siguiente, se utilizan filtros de sombras para que parezca que se ha modelado la superficie del agua.

Imagen 4.4.3: Uso de filtro para sombras arrojadas

Unidad 5: Las cámaras

5.1. Cámaras y vistas.

5.2. Parámetros de una vista.

5.3. Composición de vistas.

5.4. Efectos de cámara.

5.1. Cámaras y vistas

Debemos distinguir entre estos dos importantes conceptos. Una cámara, es el objeto que se inserta en la escena tridimensional y cuya función es crear vistas a partir de su posición y orientación u objetivo.

Una determinada vista es el resultado de una cámara. Es una imagen bidimensional obtenida a partir de una escena 3D mediante una transformación matemática que modifica los puntos 3D en 2D. Como hemos escrito, esta transformación está dirigida por los parámetros básicos de la cámara.

5.2. Parámetros de una vista

Las cámaras virtuales tienen los mismos parámetros que las cámaras reales. Estas variables las podemos clasificar en variables fundamentales y geométricas.

Parámetros fundamentales:

- Posición tridimensional de la cámara en la escena.
- Dirección del punto de mira u objetivo de cámara.
- Amplitud de cámara, es el ángulo que define la cantidad de escena que vemos.
- Tipo de proyección que queremos obtener. Puede ser de tipo paralela o perspectiva.

Parámetros geométricos:

Los parámetros geométricos que hay que definir son:

- Situación del plano de proyección o Punto de Referencia de Vista.
- Orientación del plano de proyección o Vector Normal al Plano.
- Giro del plano de proyección o Vector Vertical.
- Tamaño de la ventana.
- Punto de Referencia de Proyección o Ángulo de Vista (sólo en Perspectiva).
- Planos frontal y trasero de recorte.

En función del ángulo, una vista en perspectiva la podemos clasificar en:

- Normal: de 46º. Se corresponde con el ángulo del ojo humano. Esto supone una apertura de objetivo de una cámara réflex de 50 mm.
- Angular: cuando es mayor de 46º. Al abarcar más campo de visión, los objetos parecen más pequeños. Para sacarlos al mismo tamaño nos tenemos que acercar a ellos obteniendo grandes distorsiones. Esto supone una apertura de objetivo de una cámara réflex menor de 50 mm.
- Tele: cuando es menor de 46º. Los objetos parecen más grandes. Para sacarlos al mismo tamaño nos tenemos que alejar comprimiendo los planos. Esto supone una apertura de objetivo de una cámara réflex mayor de 50 mm.

5.3. Composición de vistas

Podemos distinguir las siguientes composiciones de planos:

- Primer plano: centra la atención en un determinado personaje que se encuadra, dejando el resto de la escena sin aparente importancia.

Imagen 4.5.1: Ejemplo de primer plano

- Plano medio: suelen incluir un personaje y algún objeto o varios personajes transmitiendo al espectador la relación que existe entre ellos.

Imagen 4.5.2: Ejemplo de plano medio

- Plano general: permite transmitir el ambiente y distribución general de la escena.

Imagen 4.5.3: Ejemplo de plano general

Composición con objetivos:

- Angular: al distanciar los objetos refuerza el alejamiento entre dos personajes o transmite la mayor importancia que tiene el personaje cercano frente al lejano.

Imagen 4.5.4: Ejemplo de imagen obtenida con objetivo angular

- Tele: al reducir la distancia entre los objetos los muestra más cercanos de lo que realmente están.

Imagen 4.5.5: Ejemplo de imagen obtenida con objetivo tele

Composición con ángulos:

- Nivel de vista: proporciona una visión neutra y objetiva de la escena.

Imagen 4.5.6: Ejemplo de plano "normal"

- Picado: hace que los personajes parezcan pequeños y débiles.

Imagen 4.5.7: Ejemplo de plano picado

- Contrapicado: hace que los personajes parezcan grandes y heroicos.

Imagen 4.5.8: Ejemplo de plano contrapicado

- Cenital: se realiza en la perpendicular desde arriba.

Imagen 4.5.9: Ejemplo de plano cenital

- Subjetivo: sitúa la cámara en el punto de vista de un personaje.

Existen muchos otras composiciones con respecto a los ángulos, aunque estas son posiblemente, las más características.

5.4. Efectos de cámara

- Seguir a un objeto: hace que la cámara siempre “mire” al objeto aunque se mueva.
- Paneo: rota la cámara sobre sus ejes.
- Travelling: desplaza la cámara sobre una línea.
- Dolly: mueve la cámara alrededor de un objeto.
- Recorrido Libre: desplaza la cámara libremente por una trayectoria cualquiera

Unidad 6: Animación

6.1. Introducción.

6.2. Animación por ordenador.

6.3. Fases de la animación.

6.4. Técnicas de la animación por ordenador.

6.4.1. Curvas de función.

6.4.2. Trayectorias de animación.

6.4.3. Fotogramas claves e interpolación.

6.4.4. Animación de personajes.

6.4.5. Animación de cámaras.

6.4.6. Animación antropomórfica.

6.5. Representación de la animación.

6.6. Posproducción.

6.1. Introducción.

Las animaciones son cambios que se realizan en las imágenes que normalmente van acompañadas de sonidos y crean la ilusión de movimiento debido a la persistencia de la visión que se produce en el caso de los humanos cuando vemos 24 o más imágenes por segundo.

Las animaciones se pueden clasificar por diversos criterios. Por ejemplo, si se hacen a mano o por ordenador, si son en dos o tres dimensiones, según las técnicas utilizadas, etc. Nosotros vamos a crear, evidentemente, las animaciones usando software que produce animaciones realizadas por ordenador en tres dimensiones y con diversas técnicas que se pueden solapar. Por lo tanto, es posible que en una misma escena coexistan varias animaciones, como por ejemplo, tener un colibrí con una animación de alas realizada con claves, un niño que se mueve mediante técnicas de movimientos de personajes, y un travelling de cámara para tomar un plano general de dicha escena.

6.2. Animación por ordenador.

Existen dos ámbitos dónde se realice animación por ordenador: en 2D y en 3D.

Aplicaciones como Flash, se utilizan para realizar animaciones en dos dimensiones, aunque la mayoría de las aplicaciones que se realizan por ordenador para 2D, están enfocadas como ayuda al método tradicional de realizarlas a mano, por tanto, no tendrían la finalidad de la producción total del proyecto, sino como animación asistida por ordenador.

La mayoría de las animaciones realizadas totalmente por ordenador están orientadas a la realización de animaciones tridimensionales. Esto hace que se tengan que elaborar escenas 3D con toda la complejidad que esto supone, ya que se le debe incorporar modelos geométricos modelados, luces, cámaras, etc.

La principal ventaja de este tipo de producciones es el manejo automático de gran cantidad de información gráfica editable. Las posibilidades son muy grandes, solo limitadas por la imaginación de los autores y los procesos de cálculo. Sin embargo, aún encontramos problemas para manejar situaciones donde necesitamos una expresividad muy detallada. Este tema es realmente difícil de conseguir, aunque también suele ser el talón de Aquiles de otros métodos de animación.

6.3. Fases de la animación.

Las animaciones profesionales normalmente son procesos muy largos. En la actualidad, la creación de animaciones por ordenador, no suponen un acortamiento en el tiempo de producción porque normalmente se aprovecha para la creación de un nivel de detalle mucho mayor que las producciones que se hacen sin el ordenador.

En la animación clásica encontramos una serie de fases que podemos enumerar:

1. Guión o Script. Como en cualquier producción cinematográfica, necesitamos tener una historia que contar en forma de guión.

2. Esquema de la historia o storyboard. Resumen gráfico de las escenas más importantes acompañadas de notas aclaratorias, tiempos, efectos de cámara, etc.

3. Banda sonora o sound track. Compuesta por la música de fondo, los diálogos y los efectos sonoros. Todo el material debe estar perfectamente producido en consonancia con los gráficos que produciremos más adelante para luego sincronizar estos dos materiales.

4. Descomposición de la banda sonora. Se analizan fonéticamente los diálogos y se documenta su posición con respecto a las imágenes. Esto será la clave para realizar imágenes

en que la posición de los labios y los rostros se correspondan con los sonidos emitidos. A este hecho se le conoce como sincronización labial.

5. Diseño. Se crean los personajes con diferentes vistas para su uso como referencia para el resto de la producción.

6. Bobina Leica. Se trata de pasar el storyboard a filmado para tener un esquema de la misma duración que la producción final y que sirve para la sincronización con la banda sonora final.

7. Prueba de línea. Sirve para crear y editar las imágenes clave. De este paso depende en gran medida la calidad de la expresión conseguida.

8. Ajustado. Sirve para crear el resto de imágenes y por tanto se trabajaran todos los detalles por partes: modelado, iluminación, etc.

9. Representación cercana. Se crean las imágenes que aparecen más cercanas al espectador. Esto hace posible utilizar unos parámetros distintos para los fondos o utilizar técnicas como el cromakey, etc.

10. Representación lejana o fondos. Se pueden representar a parte para aplicar distintas técnicas o no.

11. Doblaje. Se sincronizan las imágenes con la banda sonora y se introducen los sonidos de voces de los personajes.

12. Postproducción. Se suelen utilizar otras aplicaciones para crear efectos, comprobar sincronizaciones, superponer capas y sobre todo, para unir las dos partes: imágenes y sonidos, en una sola entidad: la película obteniendo lo que se suele llamar copia maestra o master.

En la animación por ordenador, tenemos que realizar los mismos pasos, pero se suelen llamar de otra forma y la forma de producirla es distinta, por lo que las fases reales serán una mezcla de ambos procesos.

En la animación por ordenador podemos distinguir las siguientes fases:

1. Guión o Script. Como en cualquier producción cinematográfica, necesitamos tener una historia que contar en forma de guión.

2. Esquema de la historia o storyboard. Resumen gráfico de las escenas más importantes acompañadas de notas aclaratorias, tiempos, efectos de cámara, etc.

3. Modelado. Se crean los personajes, el atrezzo y los fondos. Junto con los personajes, también debemos crear los controles que permite moverlos y añadirles expresión.

4. Animación. Se utilizan distintas técnicas, pero normalmente se crean imágenes clave y luego el software utilizado crea las intercalaciones apropiadas. Para producir las animaciones clave se utilizan técnicas de la animación clásica como la exageración, la anticipación, la puesta en escena, el aplastamiento y la extensión y otras.

5. Apariencia visual. A cada objeto de la escena se le asigna un color o textura que muestre si simula ser de madera, metal, agua, etc.

6. Iluminación. Usamos luces digitales para crear una iluminación en la escena 3D.

7. Representación. Se generan las imágenes finales calculando cada píxel a partir de los datos aportados por las fases anteriores. Esta fase suele ser crítica en cuanto al uso de la potencia de cómputo del ordenador y a veces lleva mucho tiempo. Incluso, para proyectos personales o de bajo presupuesto puede ser inviable su realización. Por otra parte, obtenemos una película que podemos elegir en cuanto a su tamaño y calidad.

8. Posproducción. Como en el caso de la animación clásica, se utilizan otras aplicaciones para crear efectos, comprobar sincronizaciones, superponer capas y sobre todo, para unir las dos partes: imágenes y sonidos, en una sola entidad que es la película.

6.4. Técnicas de la animación por ordenador.

6.4.1. Curvas de función.

Son gráficas que representan la animación de distintos parámetros. Tendremos en el eje horizontal el tiempo y en el vertical el parámetro representado. Por ejemplo, podría ser la rotación en el eje Z de un avión. Con las curvas de función conseguimos una mejor diagnosis y corrección de la animación, ya que podemos ver claramente cuando hay una falta de sincronización y además las podemos solventar directamente modificando las curvas.

6.4.2. Trayectorias de animación.

Se trata de gráficas que representan la trayectoria que sigue un objeto pero que se representan en la propia escena. Muchos programas de edición 3D permiten su modificación directamente.

6.4.3. Fotogramas claves e interpolación.

Es quizá, el tipo de técnica más utilizada debido a su simplicidad conceptual. Se trata de definir los objetos en fotogramas claves, es decir, en situaciones clave en el tiempo. Luego, el programa genera los pasos intermedios mediante el intercalado de fotogramas que son el resultado de aplicar interpolaciones entre los distintos parámetros modificados en las claves.

La interpolación normalmente es lineal pero también se pueden suavizar mediante curvas Bézier o splines. Este hecho hace que el animador pueda elegir entre cambios más inerciales o menos.

6.4.3. Animación de personajes.

La animación de personajes se basan en estructuras que podemos denominar huesos y que se disponen formando jerarquías de ellos para formas estructura mas complejas y de esta forma construir un esqueleto interno para el personaje. Esta jerarquía, hace que existan movimientos que hacen que se muevan otros objetos de jerarquía menor. Por ejemplo, en un bípedo típico como somos los humanos, el hueso con mayor jerarquía está en la cadera, y si se mueve esta, todos los demás huesos se moverán también. Cuando tenemos la jerarquía de huesos ya definida, tenemos que asignar a cada hueso una zona de influencia en la geometría que debe de arrastrar con él en los movimientos, es decir, lo que sería los músculos que implica cada hueso. Una vez definidos los huesos, su jerarquía y sus zonas de influencia, realizamos el movimiento de estos. Para ello, existen dos alternativas: cinemática directa o inversa.

Cinemática directa: se trata de mover la jerarquía desde los huesos de más jerarquía hasta los de menos. Es poco intuitiva y necesitamos realizar muchos pequeños movimientos para realizar un movimiento complejo. Por ejemplo, para mover un dedo hasta un determinado punto 3D, movemos la cadera, el hombro, el brazo, el antebrazo, la muñeca y las falanges.

Cinemática inversa: se trata de mover los huesos de menos jerarquía hasta el punto 3D deseado y que el ordenador mueva los huesos de jerarquía mayor automáticamente. Para realizar este movimiento de forma automática y creíble, el software encargado tendrá que tener definidos los límites de rotación de las articulaciones que unen los distintos huesos.

Este método es muy intuitivo para el animador ya que es así como se hace en la realidad. Por ejemplo, para mover un dedo hasta un determinado punto 3D, movemos el dedo directamente

hasta donde queramos y no calculamos conscientemente los movimientos de los huesos padres involucrados.

Segmento	Articulación	Tipo	X	Y	Z
Pie	Tobillo	Rotación	65º	30º	0º
Espinilla	Rodilla	Bisagra	135º	0º	0º
Muslo	Cadera	Rótula	120º	20º	10º
Columna	Cadera/Columna	Rotación	15º	10º	0º
Hombro	Columna	Rotación	20º	20º	0º
Bíceps	Hombro	Rótula	180º	105º	10º
Antebrazo	Codo	Bisagra	150º	0º	0º
Mano	Muñeca	Rótula	180º	30º	120º

Tabla 3: Límites de rotación de las articulaciones en el cuerpo humano

Tenemos que aclarar que en la tabla superior, el eje Z corresponde al eje longitudinal a lo largo del eje del hueso de la articulación.

6.4.4. Animación de cámaras.

Son técnicas muy utilizadas en la cinematografía y por ello, existe cierta "cultura" de cámara por lo que los espectadores están habituados a animaciones de cámaras que no se corresponden con la realidad. Y es que las cámaras, en principio, representan la visión en primera persona del propio espectador.

Disponemos de dos tipos de cámara: con y sin objetivo. Las cámaras con objetivo, pueden moverse por el espacio tridimensional mientras siguen apuntado a su objetivo. Las cámaras sin objetivo siempre mirar delante de ellas por lo que la visión cambiará con su movimiento. Además en los programas de creación de escenas 3D, tenemos total libertad para crear efectos con las cámaras como travellings, zooms, etc. Debemos tener cuidado y no abusar en exceso de efectos que puedan hacer que las vistas de la escena 3D no sean creíbles. Los tipos de movimiento más comunes son: encuadres, travellings, panorámicas, etc.

6.4.6. Animación antropomórfica.

Se trata de animar simulando los movimientos humanos, de objetos que por su naturaleza, son inanimados. Por ejemplo, en el famoso caso de píxel en que una lámpara tipo flexo, se mueve y mira observando otros objetos,…, tenemos una claro exponente de este tipo de animación.

Se utilizarán todo tipo de técnicas como doblados, giros sobre sí mismo, saltos, torsiones, para dotar de vida propia a cualquier objeto. Esta animación es bastante más sencilla si los objetos son muy maleables, es decir, si su superficie está compuesta por muchos polígonos que si es al contrario.

6.5. Representación de la animación.

La representación de la animación de la mayoría de los programas que realizan escenas 3D es un vídeo que podremos sincronizar con la banda sonora usando otras aplicaciones. Tenemos que decidir el formato de archivo que queremos utilizar: avi, mpeg,... así como el codec necesario. También si la película se reproducirá en sistemas PAL o NTSC tendremos que elegir sus parámetros.

6.6. Posproducción.

Este proceso, normalmente se lleva a cabo en aplicaciones especializadas en composición de efectos que permiten:

1. Trabajar con distintas capas donde mezclar imágenes estáticas y/o animadas con secuencias de imagen.

2. Añadir textos o subtítulos.

3. Crear transiciones entre escenas.

4. Añadir efectos especiales.

5. Sincronizar las imágenes con la banda sonora, etc.

Unidad 7: Simulaciones físicas.

7.1. Introducción.

7.2. Máquinas físicas comerciales.

7.3. Experimentación de las simulaciones físicas.

1. Introducción.

Los sistemas programados e interactivos que de alguna forma simulan la realidad o se basan en ella, como son los videojuegos, los simuladores técnicos y los metaversos necesitan separar el código que controle el tratamiento de las leyes físicas y su aplicación del resto del programa. Esto es así, debido a que estas aplicaciones son cada vez más sofisticadas y complejas y por tanto, es mejor manejar esta complejidad mediante módulos independientes. Debido a esto, surgen módulos y bibliotecas de código orientados a la simulación de conceptos físicos como el espacio, el tiempo, la masa, la velocidad, la aceleración, las fuerzas y las relaciones entre estos. Por ejemplo, es impensable que los programadores de un videojuego, codifiquen específicamente todas las situaciones que se dan en la interacción en una escena 3D para cada uno de los objetos de forma independiente. Esto haría que la programación de dichas situaciones creciera de forma exponencial y al final fuera inabarcable. Con la programación orientada a objetos, podemos abstraer muchas propiedades comunes a muchos objetos de la escena 3D y crear código común que controle características propias de cada objeto pero que les aplique las mismas leyes universales que rigen la física. Por poner un caso, en una escena podemos tener una mesa, un jarrón, una lámpara y cada objeto tendrá una posición 3D, una masa,… pero a todas les afectará la gravedad. Estas leyes físicas, se deben entender a nivel de escala humana. Estamos hablando de geometrías, texturas,… que representan objetos de nuestro entorno. No tiene sentido hablar de la naturaleza física de los objetos a nivel molecular, por ejemplo. Todo este esfuerzo de programación, normalmente se torna en una agrupación o biblioteca de utilidades y funciones que trabajan con independencia del sustrato donde se aplique, y es lo que se denomina “maquina o motor físico”.

Por tanto, la idea de una máquina física es la de un software que funciona en cooperación con otra u otras aplicaciones (que mostrará una escena 3D y tratará la interacción con el usuario, etc.) y que de manera independiente a estas, resuelve situaciones de tratamiento físico (como son la cinemática, la dinámica,…) de materiales (como la elasticidad, la masa y otros) y algunos casos de situaciones atmosféricas como el viento.

2. Máquinas físicas comerciales.

En el mercado actual encontramos a nivel de informática personal dos máquinas físicas muy conocidas: Havok y PhysX.

Estas dos máquinas físicas son muy utilizadas en ámbitos profesionales y de gran repercusión económica como los juegos para las videoconsolas más vendidas hoy en día.

En cada una de las dos soluciones que proponemos en los temas siguientes para realizar prácticas del curso en desarrollo se utiliza un motor físico distinto. Por lo tanto, está será la oportunidad para estudiarlos con un poco más de detenimiento, algo que merece la pena sobre todo cuando queremos aportar verosimilitud a la hora de hacer estudios que simulen la realidad. Por ejemplo, para trabajar con líquidos o sólidos dinámicos como pueden ser fuentes, esculturas móviles, etc.

Como pequeña introducción podemos decir que Havok puede ser la más conocida y la más empleada. Lleva más tiempo en el mercado y se utiliza entre otros en Second Life.

PhysX es un gran motor físico y también se utiliza en muchos juegos potentes, entre ellos, los más nuevos para la videoconsola PlayStation 3. PhysX es la nueva máquina física de la última versión de Director de Adobe.

En los apartados correspondientes a las máquinas físicas tanto en el capítulo dedicado a Director como a Second Life, realizaremos una introducción más profunda sobre cada una de estas soluciones. Sobre todo, cuando estudiemos Second Life, ya que al ser un metaverso, experimentaremos constantemente el funcionamiento de estos módulos.

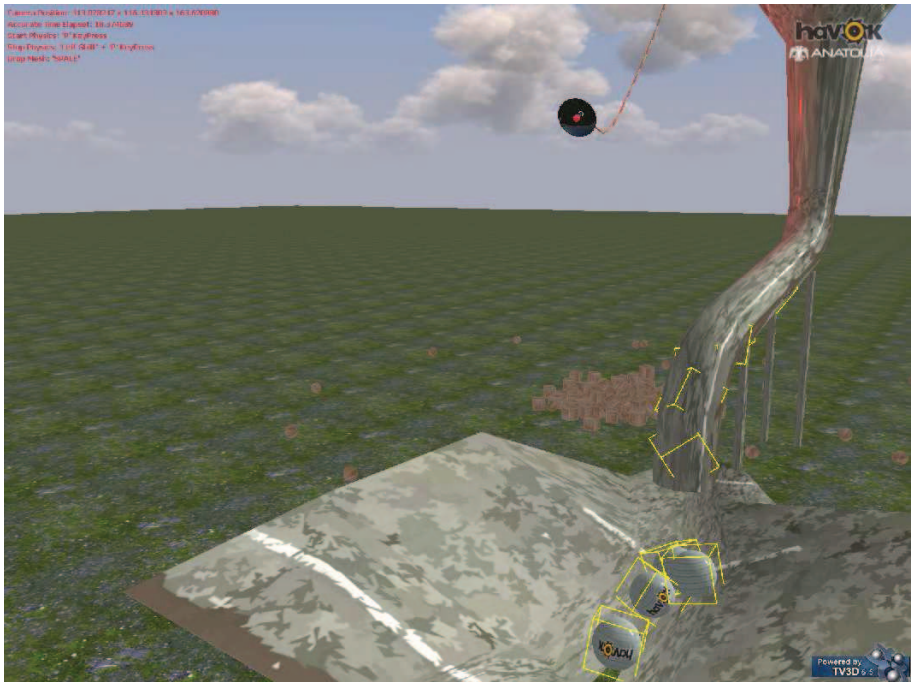


Imagen 4.7.1: Ejemplo de uso del motor físico Havok

Imagen 4.7.2: Ejemplo de uso del motor físico PhysX

3. Experimentación de las simulaciones físicas.

Los motores físicos están trabajando constantemente en su aplicación huésped. Por ejemplo, cuando estamos en Second Life y podemos caminar por un terreno, la máquina física, en este caso Havok, se encarga de que nuestro avatar no flote, sino que sea afectado por la gravedad y por otro lado, algún autor, habrá definido el terreno como “sólido” y por ello, caminaremos sobre él y no dentro de él. Es decir, no nos hundimos porque alguien lo habrá definido como no traspasable. Por tanto, en sistemas abiertos a la experimentación con Second Life y Director, podremos probar distintas opciones para definir los objetos que vamos a crear y asignarles propiedades físicas. En este sentido, Second Life es más inmediato que Director, ya que existe

un interfaz para poder manejar estas propiedades sin tener que programar, hecho ineludible en Director. Haremos experimentos de tipo físico muy sencillos, como dejar caer una pelota con cierta masa y elasticidad por planos inclinados, etc. para poder demostrar la flexibilidad que nos proporciona el dominio de la máquina física integrada en la aplicación a utilizar.

Unidad 8: Sistemas de partículas.

8.1. Introducción a los sistemas de partículas.

8.2. Características de los sistemas de partículas.

8.1. Introducción a los sistemas de partículas.

Los sistemas de partículas son entidades que generan partículas que recorren un espacio. Este movimiento se inicia en el emisor y cada partícula puede tener un recorrido independiente aunque siempre parametrizado de manera general. Las partículas se moverán por una trayectoria tras lo cual desaparecen. Es lo que se llama la "vida" de las partículas. Por tanto, un sistema de partículas en una entidad compleja compuesta por un emisor y por un número determinado de partículas que recorren trayectorias uniformes pero independientes tras lo cual desaparecen.

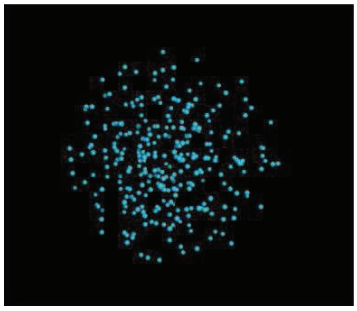


Imagen 4.8.1: Sistema de partículas

Los sistemas de partículas permiten modelar muchos fenómenos de la vida real como son los fluidos (cascadas, fuentes,…), fenómenos atmosféricos (lluvia, nieve, niebla,…) y otros más complejos como manadas de mamíferos, bandadas de aves, etc. También se pueden utilizar para modelar efectos especiales, como resplandores, halos, rayos, etc. Por esto, son de uso masivo en sistemas de videojuegos y metaversos.

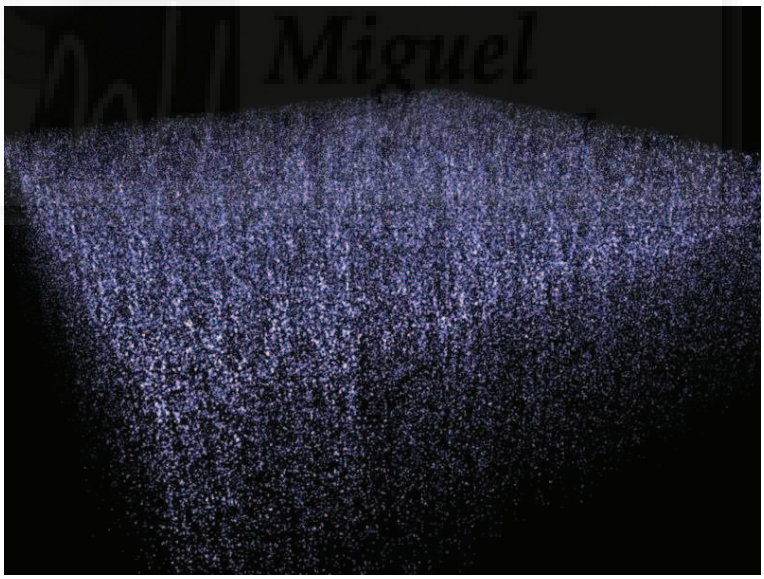


Imagen 4.8.2: Sistema de partículas 2D

Un hecho significativo es que tradicionalmente, las partículas son elementos bidimensionales, normalmente construidas con un plano mas o menos pequeño y texturizado y sin embargo, sus trayectorias son tridimensionales, por lo que todo el sistema se puede considerar un elemento 3D. Este es el caso de la lluvia. Las gotas son realizadas como pequeños planos que siempre se muestran hacia la vista del espectador y cuyo emisor es en este caso un plano que representaría la nube. Cuando el sistema de partículas se pone en marcha, las gotas "nacen" del emisor que representa la nube, siguen una trayectoria que en este caso supone caer hacia la tierra y luego desaparecen.

En la actualidad, se consideran y manejan como sistemas de partículas, aquellas entidades que utilizan como partículas a elementos geométricamente en tres dimensiones y que además no tienen una vida efímera, sino que permanecen en la escena por un tiempo preciso.

Este es el caso de los ejércitos que vemos en muchas películas y videojuegos de grandes batallas. Aquí cada partícula es en realidad una geometría 3D compleja que representa un soldado y que se mueven uniformemente pero con cierta libertad individual a las órdenes de sus comandantes, pero a diferencia de los sistemas de partículas clásicos, cada soldado no desaparece por sí mismo. Por tanto, estas evoluciones de los sistemas de partículas se pueden considerar elementos distintos y no los estudiaremos en nuestros casos prácticos.

Imagen 4.8.3: Sistema de partículas 3D

8.2. Características de los sistemas de partículas.

Como ya hemos comentado las partículas en los sistemas que vamos a realizar en los casos prácticos serán elementos muy simples: planos con un color determinado o una textura aplicada.

El emisor normalmente es un punto determinado del espacio en 3D. En algunas aplicaciones como en Second Life es obligatorio que este punto esté contenido en un modelo 3D, pero esto no es siempre así. Uno de los factores más importantes que afectan a la forma de emisión, es si esta se produce de forma continua o discreta. Al salir de forma continua, las partículas forman una masa compacta, un caudal, mientras que de forma discreta simularán como “explosiones” y las partículas saldrán “a borbotones”. La geometría del emisor también puede ser variable, por ejemplo, podemos hacer que el emisor se disponga linealmente, definiendo una curva, en un área o incluso en un volumen tridimensional. De esta manera podremos crear cascadas o paredes de fuego y también nieblas y otros efectos volumétricos.

Por otra parte, si atendemos a todo el sistema de partículas en conjunto, podemos verlo como un elemento geométrico variable. Por ejemplo, si creamos un fuego o una fuente, podemos variar parámetros, que hacen que todo el elementos varíe de forma considerable. Algunas de estas características pueden cambiar a lo largo de la “vida” de las partículas son el tamaño de las partículas, el color, la transparencia, el número y la longitud de su movimiento. Esto puede hacer que el sistema sea modificado por múltiples factores incluida la interacción con el usuario, y esto se puede utilizar para crear efectos especiales muy espectaculares.

Unidad 9: Programación.

9.1. Introducción a la programación de entornos interactivos.

9.2. Estructura de los programas orientados a objetos.

9.3. Estructura de los scripts.

9.1. Introducción a la programación de entornos interactivos.

Lo primero que tenemos que tener en cuenta, es que en la actualidad, las aplicaciones de cierta entidad, se construyen de forma modular. Esto es un hecho sea cual sea el ámbito de la aplicación, desde software para ofimática como Access hasta otros para trabajos multimedia como Flash, tienen una estructura modular. En el caso de admitir la programación como es el caso de las dos aplicaciones antes citadas, es mediante un lenguaje propio que crea pequeños subprogramas (normalmente llamados scripts) que trabajan conjuntamente con el resto de los módulos y que permite personalizar, es decir, programar a medida la aplicación para que funcione de la forma en que el programador desea.

Por ello, se puede decir que cada aplicación, tiene un funcionamiento básico intrínseco a su naturaleza, y además existe la posibilidad, de especificar un funcionamiento especial para añadir funcionalidades que lo adapten a casos particulares por medio de programación a medida. Por ejemplo, Access es un sistema gestor de bases de datos con un funcionamiento ya definido “de fábrica”, pero se pueden crear los llamados módulos de clase en lenguaje VBA (Visual Basic Application) para hacer que un Access en particular sea personalizado y trabaje correctamente para gestionar una tienda de perfumes y no un comercio de “todo a 100”.

En este sentido, nuestros casos prácticos pueden basarse en un núcleo ya programado que podríamos utilizar como base. Este núcleo podría ser el mismo que el de los videojuegos, ya que este tipo de software cumple los requisitos expuestos en la presente tesis: realizar representaciones tridimensionales e interactivas y publicables on y off line. A este tipo de núcleo o programa base para videojuegos en 3D, se les suele denominar “motores de juegos”. El problema de estos motores, es que son productos efímeros, de empresas pequeñas que hacen uso de una tecnología bastante avanzada y además dependen mucho de la arquitectura hardware donde se ejecutan, normalmente no son multiplataforma. Por otra parte, suelen tener una curva de aprendizaje muy pronunciada debiendo dedicar mucho tiempo para tener resultados aceptables. También posee grandes posibilidades modulares que sirven para que el motor del juego trabaje conjuntamente con otros módulos entre los cuales podemos destacar un motor físico, programación gráficas (para desarrollar modelos, iluminación, shaders), programación de la interacción y la navegación 3D, un motor lógico, un módulo de inteligencia artificial (para toma de decisiones, movimientos, aprendizaje, evolución,…) y otros menos importantes.

De todas formas, todos tienen una estructura común. Una especie de “base” sobre la que gira todo lo demás, o sea, el resto de módulos. Se suele denominar programa principal y está compuesto por un bucle principal y unos estados básicos definidos. Según esta idea, va a ser más fácil que entendamos este concepto al desarrollar el caso práctico con Director que con Second Life, ya que en Director empezaremos a crear nuestra aplicación desde cero, partiendo de la nada, y lo primero que necesitaremos será precisamente este “programa principal”. Por tanto, realizaremos una estructura basada en un bucle principal y definiremos los estados básicos de la aplicación, que utilizaremos para todos los casos que desarrollemos en Director. En el caso de Second Life, no podemos llegar nunca a trabajar de esta forma, ya que está pensado para trabajar modificando lo ya construido, no para comenzar desde el origen.

9.2. Estructura de los programas orientados a objetos.

En la actualidad, el modelo de programación que se utiliza universalmente, es el llamado orientado a objetos. Este método surgió como respuesta a la llamada “crisis del software” que se produjo en la década de los años 80 del siglo pasado debido al crecimiento exponencial de la complejidad con que se tenía que enfrentar un programador para realizar trabajos cada vez más exigentes.

La programación orientada a objetos construye entidades llamadas objetos definiendo sus propiedades y operaciones que los transforman. Por ejemplo, un “botón” del interfaz de una aplicación, es un objeto con muchas propiedades, entre ellas, la anchura y la altura. Alguna de las operaciones con las que trabaja este botón son por ejemplo, del tipo “responder cuando se pase el cursor por su superficie”. A estas operaciones se les suele llamar métodos del objeto.

Las propiedades de un objeto se construyen con variables, y sus métodos, se construyen con funciones. Estos dos conceptos, el de variable y función, se utilizan desde que nació la propia programación, por lo que podemos concluir, que la programación orientada a objetos, no utiliza componentes nuevos, sino que más bien se trata de una reorganización y unos nuevos métodos que producen un rendimiento incomparablemente mayor debido entre otras cosas a la reutilización de los objetos y a un uso de estos mucho más óptimo.

Además, la programación orientada a objetos, trajo consigo la programación visual y un modo de la programación de la interacción con el usuario basado en eventos y acciones. Se puede describir básicamente como sigue: las aplicaciones se mantienen en un estado inercial (no hacen nada, manteniéndose en un bucle de espera) mientras no se pida por parte del terceros (usuarios, otras aplicaciones o el hardware,…) que realice algo. Este "algo" será un evento. Los eventos, como hemos escrito se pueden producir por múltiples motivos, por ejemplo, que un usuario haga clic con el ratón sobre un botón, o por que pase un segundo desde el último evento (para medir el tiempo), etc. Este evento, será captado por el bucle de espera de la aplicación y será tratado ejecutando una acción. Por ejemplo, si un usuario hace clic sobre un botón, puede que el programa ejecute una acción que signifique por ejemplo, abrir una página web. Estas acciones normalmente hacen uso de funciones de alto nivel. Se llaman de alto nivel porque son muy comprensibles por cualquier persona aunque no sea un consumado programador. Mayor nivel significa más comprensible por el ser humano, y por tanto, bajo nivel significa más comprensible por el hardware y no por un ser humano. Estas funciones forman parte del lenguaje de programación y normalmente dependen del tipo de aplicación en la que se ejecutan. Por ejemplo, en Flash, que se encarga de realizar tratamientos multimedia, tenemos funciones de alto nivel para ejecutar un video, pararlo, rebobinar, etc.

Como hemos escrito anteriormente, el autor – programador en los casos prácticos que estudiaremos a continuación, en la mayoría de los casos, escribirá código circunscrito a un determinado objeto y se llaman scripts. Por ejemplo, podemos hacer que una luz sea creada, borrada, modificada, etc. Estos pequeños módulos implican solo a un determinado objeto y lo transforman de alguna manera.

9.3. Estructura de los scripts.

Los scripts suelen tener una forma muy definida en casi todos los lenguajes de programación modernos, basándose en una sintaxis que cuyo principal exponente es JavaScript.

Esta estructura podría sintetizarse como sigue:

Variables

On evento_1
Acciones de respuesta al evento_1
…
Fin evento_1
…
On evento_n
Acciones de respuesta al evento_n
…
Fin evento_n

Esta estructura comienza con la definición de las variables que se van a utilizar en el script, bien sean estas de ámbito local o global, es decir, valores que pueden ser modificados y ser pertinentes para todo el programa en su conjunto o solo para el código que se escribe justo después.

A continuación seguirá la captura de un evento y el tratamiento de este. Lo normal es que primero se declare el evento con el que se quiere trabajar, seguido de la o las acciones de respuesta. Como hemos escrito antes, estas acciones estarán codificadas muchas veces como

funciones de alto nivel. Sigue un ejemplo válido para el caso práctico de Director pero que es muy parecido también en Second Life y otros sistemas.

```
global mundo3D

on mouseDown
    mundo3D.model("Esfera").Scale(.5,.5,.5)
end

on mouseUp
    mundo3D.model("Esfera").Scale(2,2,2)
end
```

En el caso anterior, se declara la variable global mundo3D para poder trabajar con ella. En este caso en particular, esta variable permite acceder a los elementos que componen la escena tridimensional. Luego se tratan dos eventos. El primero de ellos, es cuando el usuario pulsa el botón izquierdo del ratón. Como no se trata el sitio, se supone que es en cualquier lugar de la escena 3D. Como respuesta, el autor escribe, que el objeto llamado “Esfera”, sufra una transformación en escala que hace que se reduzca un 50% en los tres ejes X, Y, Z. En consecuencia la esfera se reduce a la mitad de su tamaño. En el segundo de los eventos tratados, se codifica que cuando se levante el botón izquierdo del ratón, la esfera multiplique su tamaño por dos en los tres ejes.

En resumen, este ejemplo produce que si el usuario pulsa el botón izquierdo del ratón sobre la escena 3D, verá cómo la esfera se hace la mitad de grande y cuando deja de pulsar, recuperará su tamaño.

Se puede concluir que realizando esta clase de scripts y asignándolos correctamente, podemos ir modificando la escena 3D o el metaverso, en resumen, el sistema donde trabajemos para programarlo según nuestras necesidades. Esto unido a que podemos manejar igual de fácilmente cualquier elemento como texturas, geometrías de modelos, luces, movimientos,…, hace que sea abordable la tarea de generar trabajos en 3D interactivos propios.

Unidad 10: Interacción en mundos virtuales.

10.1. Introducción a la interacción.

10.2. Características de la interacción con mundos virtuales.

10.3. Cómo crear la interacción.

10.1. Introducción a la interacción.

En general entendemos la interactividad en el ámbito de la informática como un proceso en que un usuario establece una comunicación con una aplicación y por tanto es bidireccional. Esta idea significa que para diseñar una buena interactividad necesitamos implementar ideas como la de “usabilidad”. Esta idea en el ámbito de la informática, es un concepto que surgió para definir la facilidad con que las personas pueden utilizar una herramienta software. Por tanto, en el diseño y realización de la interacción de nuestros casos prácticos, tendremos que tener en cuenta tanto las posibilidades que queremos alcanzar, como la facilidad que tendrán los usuarios para alcanzar estos objetivos.

Existen métodos interactivos que debido a su universalidad, todos los usuarios de informáticas actuales los conocen, como son el manejo del ratón, el teclado, etc. En nuestro caso, tenemos una dificultad añadida y es que las escenas son tridimensionales, por lo que a un usuario no experimentado, lo puede costar un poco desplazarse en 3D, utilizando dispositivos bidimensionales como son el ratón y la pantalla.

10.2. Características de la interacción con mundos virtuales.

En el presente trabajo, la interactividad es muy importante, ya que el autor de la obra puede exponerla desde distintos ángulos si proporciona al espectador libertad de movimientos dentro de la escena 3D. También podría crear obras que fueran interactivas en sí mismas. Esta interactividad puede ser en algunos casos intrínseca al sistema de realidad virtual o no. En este último caso deberá ser resuelta por el autor.

En un mundo virtual, necesitamos desplazarnos, navegar por la escena para observar detalles, interaccionar con los objetos, etc. En este sentido, existen dos alternativas para representar el “yo”, es decir, el usuario. La primera es utilizando una cámara. Este método es el más tradicional y utilizado. No hace falta más que ver la mayoría de los videojuegos actuales. La acción se representa en primera persona. Esto quiere decir que el usuario mueve una cámara y esta muestra va mostrando la escena como si fueran los ojos del usuario. En muchos casos, para darle mayor realismo, la cámara se sitúa a una altura de más o menos 170 cms. para dar mayor realismo a la vista mostrada al ser la altura de una persona. Este será el método que utilicemos en el caso práctico resuelto con Director. Es decir, que en este software utilizaremos cámaras como modo básico de desplazamiento por la escena.

En otros sistemas como el caso práctico que veremos más adelante y en el que utilizamos Second Life, el sistema muestra un avatar. Un avatar es un modelo que representa gráficamente al usuario. Al ser un modelo virtual, puede ser cualquier cosa: un animal, un personaje, un árbol, lo que nosotros queramos, aunque por defecto es un ser humano que el usuario personaliza para elegir sexo, cuerpo, ropa, etc. Este tema, en según que sistemas cobra mucha importancia ya que es la representación ante todos los demás, de un usuario. Por lo tanto, el modo normal de desplazamiento recreará el modo de desplazamiento del avatar y tendremos, pues, modos para caminar, saltar, correr, etc. En realidad, podríamos decir que son el mismo método, pero cuando el sistema es un metaverso, por ejemplo, donde compartimos el escenario 3D con otros usuarios, podemos necesitar un método de representación del “yo” del usuario.

Otro tipo de interacción imprescindible es el que se da entre el usuario y los elementos de la escena 3D. Por ejemplo, para nuestro caso, ante la exposición de una obra como puede ser un dibujo o una fotografía, deberíamos poder acercarnos hasta un límite, o encender una luz o transportar un objeto de un sitio a otro.

Por todo ello, podemos concluir, que existen dos niveles de interacción: la interacción del usuario con su representación dentro del sistema virtual y la interacción del usuario con el resto de elementos de la escena 3D. En nuestro caso, nos interesa más la implementación del usuario con los objetos que lo rodean, ya que estos serán los objetos que queremos mostrar y que son la base del presente trabajo.

En los dos casos prácticos que estudiaremos, es posible la realización de la interacción entre el usuario y los elementos del entorno, por lo tanto, cumplen el requisito de que las soluciones propuestas han de ser interactivas. Ahora bien, mientras que en el caso de Director, como ya estudiaremos, tenemos que realizar nosotros todo el trabajo, en Second Life, la interacción está ya implementa por la propia naturaleza de un metaverso. Otra forma de expresar lo mismo es que mientras en Director tenemos que crear la interacción, en Second Life está integrada en el mismo sistema.

Otra gran diferencia se produce en la interacción "física". En concreto, en la forma en que el usuario puede "tocar" objetos, si puede o no atravesar paredes, etc. Aquí volvemos a tener la misma diferencia, en unos sistemas como Director, donde el autor debe realizar la tarea desde cero, también tiene que definir qué elementos tienen propiedades físicas y cuales no, y por tanto, que tipo de interactividad mostrará con el usuario. En otros sistemas, como los metaversos, que muestran una realidad virtual, los objetos físicos y la manera "real" de interactuar con ellos vienen ya definidos. Además, la implementación de los objetos físicos no es una tarea simple, por lo que normalmente está fuera del alcance de un autor que no tenga amplios conocimientos del sistema y de programación.

Por otra parte, podemos tomar este hecho, como algo positivo y aprovechar que en los mundos virtuales no existen verdaderas leyes físicas, para experimentar y realizar exposiciones que son imposibles en la realidad, como exponer esculturas "en el aire" o mostrar objetos que podamos atravesar con la cámara.

Podemos concluir comentando que dependiendo de que el trabajo que queremos mostrar sea bidimensional o tridimensional, la interacción para su muestra es totalmente distinta. En el caso de un objeto en 2D como una foto, bastará con que el espectador pueda situarse a la distancia que desee. El problema surge con objetos 3D. En este caso podemos implementar que esté inmóvil y el usuario se desplace a su alrededor para observarla, que gire sobre algún eje para que el usuario no se tenga que mover, etc. También podríamos hacer que el usuario, a petición propia, fuera desplazado instantáneamente por el sistema hasta un emplazamiento que el autor considerase como el óptimo para la contemplación de su obra.

Caso a parte, sería que el autor contemplase la posibilidad que los usuarios interviniesen de algún modo en la obra. Por ejemplo, si fuera una "escultura" móvil, y se permitiese que los usuarios modificasen la inercia de la obra. Es evidente que este tipo de recreación aumenta muchos la complejidad de la interacción.

10.3. Cómo crear la interacción.

Las interacciones dentro de una escena 3D, permiten que el espectador intervenga de algún modo en ella. Como hemos escrito anteriormente, un sistema como Second Life aporta una interactividad básica, que se encuentra siempre en este metaverso, mientras que en sistemas como Director, lo tienen que hacer todo los autores. Pero de todos modos, para controlar todos los aspectos de la interacción obra-espectador, sea cual sea el sistema, debemos de utilizar un método para implementarla. Para controlar esta interacción utilizaremos código escrito de programación. Cómo adelantamos en el apartado de programación, en aplicaciones de realidad virtual, el programador interviene creando pequeños módulos llamados "scritps" que trabajan en conjunto con la parte general de la aplicación y cuya función es aportar funcionalidades particulares a un determinado trabajo. Estos scripts, se pueden guardar para ser reutilizados en otros casos parecidos, Por ejemplo, supongamos que realizamos un script que permite al espectador de un objeto 3D acercarse hasta 20 cms. de esta pero sin llegar a tocarla. Este código podemos utilizarlo para varios objetos 3D en un mismo trabajo o en exposiciones distintas.

En el apartado de programación ya vimos que los scripts se basan en el paradigma de eventos y acciones, que será también el que utilizaremos para crear los scripts que controlan la interacción. Por seguir con el ejemplo anterior, ante el evento de desplazamiento del espectador, el script comprobará si está dentro del margen permitido y si es así, permitirá al

espectador moverse y si no, aparecerá un mensaje explicando que está demasiado cerca de la escultura y lo dejará en el sitio donde esté.

Existen aplicaciones como es el caso de Director, que proporcionan elementos para acelerar y facilitar la programación de sus trabajos. Estos elementos se denominan “behaviors” o comportamientos predefinidos. Estos, son en realidad el mismo concepto: scripts que realizan tareas muy generales y que ya están codificados anteriormente y que están enfocados a su reutilización. La gran diferencia es que estos comportamientos son aportados por el mismo sistema, es decir, vienen ya con la aplicación, y el o los autores no tienen más que elegirlos de un menú y aplicarlos a distintos elementos de la escena. Por ejemplo, en Director, existen comportamientos para rotar un objeto 3D en uno de los tres ejes. Por tanto, si el autor de un objeto 3D desea por ejemplo, que su obra gire lentamente sobre su eje vertical, solo tendrá que elegir el comportamiento, asignarlo a una escultura en particular y decirle mediante una caja de diálogo que gire en el eje Z y a una velocidad de 1 vuelta cada 5 segundos. Este sistema es muy productivo, ya que aunque los lenguajes que proporcionan los sistemas de realidad virtual aportan funciones de muy alto nivel, los autores, en muchos casos no van a requerir programaciones muy sofisticadas, por lo que utilizando comportamientos predefinidos no hace falta ni que entiendan el código, ya que “programarán” sus obrar mediante cajas de diálogo. Además, es un sistema muy rápido debido a que la asignación del código a los elementos es más rápido con los comportamientos. De esta forma es muy rápido trabajar en serie, y por tanto vamos a ganar mucho tiempo para hacer que por ejemplo, una serie de 10 esculturas giren todas a la vez. Lo ideal sería tener ambos sistemas: la programación de código directamente para tener un control total de la escena y poder realizar nuestros propios comportamientos, lo cual añadiría originalidad a nuestra obra y por otra parte disponer de comportamientos predefinidos para ganar en productividad. En este sentido, ya estudiaremos cómo Director aporta las dos soluciones mientras que Second Life no permite los comportamientos predefinidos directamente.

Unidad 11: Integración multimedia.

11.1. Introducción a la multimedia.

11.2. Características de la integración multimedia en mundos virtuales.

11.3. Cómo crear la integración multimedia.

11.1. Introducción a la multimedia.

El concepto multimedia es muy utilizado en la actualidad y en muy distintos ámbitos. Esto hace que muchas veces sea una idea un tanto confusa, difuminada. En el presente documento, la multimedia que queremos utilizar en nuestros trabajos se refiere a la integración de distintos medios de expresión en un mismo entorno. Estos medios de expresión son los textos, la imagen, la animación de imágenes, el sonido y el video digital. Existen autores que incluyen entre estos medios a la interactividad. Nosotros somos de la opinión que no es un medio en sí mismo y por ello, y debido también a la capital importancia que tiene en los trabajos que nos disponemos a realizar, hemos tratado la interactividad en un tema propio.

Los medios multimedia contienen los siguientes tipos de información:

- Texto: puede ser de distinto tipo, contener texto formateado o sin formatear. También se consideran los hipertextos.

- Gráficos: pueden ser un tipo especial de imagen que representan planos, dibujos vectoriales y también tendríamos en esta categoría los esquemas que representan de forma visual, los datos numéricos que se dan por ejemplo, en las hojas de cálculo.

- Imágenes: serían los documentos de tipo bitmap, es decir, formados por píxeles dispuestos en forma de una cuadrícula.

- Animación: están formados por una serie de imágenes llamadas fotogramas que al visualizarse a una velocidad determinada de fotogramas / segundo crea una sensación de movimiento.

- Sonido: pueden ser documentos audio de música, efectos especiales, diálogos,… que se pueden ejecutar de forma autónoma.

- Vídeo: este tipo de documento presenta en una sola entidad llamada película una animación junto a una banda sonora que se ejecuta al mismo tiempo.

11.2. Características de la integración multimedia en mundos virtuales.

Las posibilidades de la inclusión en nuestros trabajos de medios multimedia dependen mucho de la plataforma utilizada para implementarlos. En general podemos decir, que si una aplicación está preparada para soportar una escena tridimensional interactiva, debemos suponer que soportará también el tratamiento con distintos medios, o sea, será multimedia. Esto es debido a que, como hemos escrito con anterioridad, la propia interacción, en muchos ocasiones se supone una forma más de medio y además, el más difícil de realizar debido a su complejidad técnica, que como hemos visto en el punto precedente, requiere en la mayoría de los casos de la inclusión de modos de programación. Por tanto, al tener los medios multimedia una complejidad de realización menor que la interactividad, casi todos los sistemas de realidad virtual los soportan. De nuevo, en esta cuestión podemos hacer una comparación entre dos modos de ver la realidad virtual: las aplicaciones generalistas que incluyen también escenas 3D, y las aplicaciones no son generalistas y “solo” permiten escenas 3D al ser en su misma naturaleza una realidad virtual tal como sucede en los metaversos.

En los casos prácticos que siguen, podremos comprobar que Director pertenece a la primera categoría, es decir, trabajan con escenas 3D interactivas como un elemento más, pero en realidad su filosofía es la de “gestor multimedia”. Es evidente que todos sus procedimientos de trabajo, su interface,… están enfocados al tratamiento con distintos medios y por tanto, se supone que será fácil la incorporación y la configuración de estos. Todo el sistema trabaja desde el principio en función de la integración multimedia. También veremos, que pondremos Second Life, como paradigma de metaverso, es decir, de realización de realidad virtual en la red. En este caso, veremos multimedia por todo a nuestro alrededor, constantemente, pero si nos fijamos, parecerá que todos los medios de un determinado tipo, por ejemplo, todas las animaciones, tienen un aspecto parecido. Esto es debido, a que en los metaversos se pone el

acento en la interactividad ya que esta característica es imprescindible para dotar de realismo a la inmersión en estos mundos y los medios, aunque se pueden realizar, se deben implementar obligatoriamente con determinados tipos de archivos para mantener el refresco de todo el sistema en velocidades que permitan su interactividad inmediata.

Pongamos un ejemplo, en un metaverso, normalmente no se admitirán imágenes de tipo bmp para texturizar una casa, ya que el tratamiento de estas imágenes que ocupan muchos kilobytes de información harían peligrar el funcionamiento de todo el sistema y además tenemos que tener en cuenta que el sistema es "compartido", que nuestros actos tienen consecuencia en los usuarios con que compartimos la escena 3D. Sin embargo, en sistemas como Director, al ser estos autónomos, es decir, se ejecutan sin la intervención de otros usuarios, nos podemos permitir el usar muchos formatos distintos. Y aunque no sea recomendable, será responsabilidad del autor, la buena elección de los formatos a utilizar. Para terminar con esta cuestión, podemos escribir que Director, por poner un ejemplo, permite el trabajo con más de 40 tipos de formatos de imagen, 7 de sonido, 7 de vídeo, etc., mientras que Second Life solo permite trabajar con 1 archivo de imagen y 1 de sonido y 1 de vídeo, teniendo que implementar los autores, el resto de medios utilizando programación.

También debemos introducir el concepto de streaming. Este método de ejecución de medios multimedia es muy utilizado en la actualidad en Internet ya que suponen un modo muy eficiente de reproducción de grandes archivos. Estos archivos que ocupan una gran cantidad de espacio suele ser vídeos digitales y de sonido.

En los sistemas de realidad virtual cabe la posibilidad de oír por ejemplo una música de fondo o ver vídeos integrados en la escena 3D. Si se ejecutan por los procedimientos tradicionales, estos vídeos van a consumir muchos recursos computacionales del espectador y por tanto, pueden hacer que todo el sistema y su interactividad se venga abajo. Sin embargo, cuando se ejecuta un vídeo en streaming, este no se transfiere en su totalidad al espectador, sino que se va visualizando según se transfiere, por lo que tenemos varias ventajas. La más evidente, es que no tenemos que esperar a que el vídeo esté del todo descargado para empezar a verlo. También podemos deducir, que aprovechamos el tiempo de visualización del vídeo para seguir descargando información que aún esté siendo transferida, por tanto, mientras el espectador está ya visualizando un contenido, se están descargando los siguientes.

Otra ventaja es que tampoco son necesarias grandes cantidades de memoria secundaria en el ordenador del expositor, ya que el archivo se ejecuta "al vuelo" y es llamado por la aplicación en el momento de la ejecución. Por tanto, no tiene porque "residir" en el mismo ordenador que la aplicación que lo utiliza. Esto significa que podemos tener elementos multimedia esparcidos por la red y mediante una llamada de programación visualizarlo en nuestra escena 3D, sin que en realidad pertenezca a ella más que como referencia a una URL. La contrapartida es que este método exige, evidentemente, de una conexión a la red para hacer streaming por Internet.

Todo esto hace muy interesante que las aplicaciones de realidad virtual permitan este método de trabajo. En este sentido podemos adelantar que las dos aplicaciones que implantamos en los casos prácticos suportan esta tecnología.

11.3. Cómo crear la integración multimedia.

Como hemos adelantado anteriormente, la forma en que se integran los distintos medios dentro del mundo virtual, depende de la naturaleza y el enfoque que tengan las aplicaciones generadoras. Es evidente, que en aplicaciones donde tenemos que integrar los archivos de medios en el interior de la escena 3D, debemos de tener una forma de incorporar, de importar este archivo. Esto no supone un problema. Por ejemplo, si queremos hacer que nuestra muestra incluya trabajos de tipo fotográfico, podemos importar una imagen, supongamos un archivo de tipo jpg a la escena. El problema, es que al ser un mundo 3D, la imagen no puede estar "libre" por el mundo virtual. Lo que normalmente haremos será crear un modelo 3D muy sencillo que sirva de sustrato a la imagen. Sería como el papel del mundo real, por lo que lo podríamos implementar con una caja muy delgada o un plano y sobre él, se aplicará la imagen fotográfica, o sea, texturizaremos el plano con la imagen deseada.

Un problema parecido tenemos con el texto. Tenemos que tener en cuenta, que tal como pasa en el mundo real, el texto no se visualiza en el vacío, por lo que podemos realizarlo como una imagen y seguir el método anterior. Es evidente que si así lo hacemos, lo podemos utilizar como un periódico, pero nunca admitirá un tratamiento digital. Por ejemplo, no podrá ser un hipertexto.

Este tema se puede resolver en casi todas las soluciones estudiadas para esta tesis y desde luego para los dos casos prácticos. Además los hipertextos se pueden realizar mediante interacciones del espectador con elementos que no tienen que ser de tipo texto, sino que podemos hacer que la escena cambie y que nos transporte a una página web o a otro mundo virtual, al interaccionar con un elemento 3D. Por ejemplo, podemos presentar tres cubos que roten y que sobre su superficie tengamos imágenes o vídeos de los mundos dónde podemos ir al tocarlos. Esto abre muchas posibilidades de navegación e interactividad.

Tampoco suelen dar problemas los sistemas con los sonidos y en los dos casos prácticos tenemos varios canales de audio que se pueden superponer en el tiempo, con lo cual es posible oír una música de fondo o un hilo musical, al mismo tiempo que el efecto sonoro de la rotura de un jarrón. El problema lo podemos tener con los elementos que en realidad son derivados de los medios básicos, como es el caso de las animaciones o las películas.

Como ya hemos escrito, una animación es un conjunto de imágenes que se ejecutan una tras otra. Por ello, muchos sistemas no permiten la importación de animaciones como tales, sino que hacen posible que podamos importar las imágenes de una en una y luego aportan medios para que mediante la programación, el autor pueda montar y crear la animación de los fotogramas. Pueden suceder lo mismo con las películas: algunos sistemas dejan en manos del autor la ejecución conjunta pero desde orígenes distintos de la animación por una parte y la banda sonora por otra. Esto puede llevarnos a problemas de sincronización de ambas partes.

Estos problemas se suelen solucionar, como hemos visto, si las aplicaciones permiten la ejecución de medios por streaming.

Unidad 12: Publicación on/off line.

12.1. Introducción a la publicación de mundos virtuales.

12.2. Características de la publicación off line.

12.3. Características de la publicación on line.

12.1. Introducción a la publicación de mundos virtuales.

En esta introducción teórica, apenas vamos a entrever las cuestiones previas que tenemos que resolver antes de realizar la publicación en sí. Ya estudiaremos estos temas cuando realicemos los casos prácticos. De todas formas, es evidente, que antes de producir el producto final de nuestro trabajo, es decir, la versión definitiva de lo que queremos exponer, debemos de comprobar que todo funciona como habíamos planeado.

Esta cuestión no es superficial. Una de las fases más importantes de la creación digital según la ingeniería del software es la prueba. La prueba debe formar parte del proceso de creación de forma habitual y sistemática. Además tenemos que diseñar el método de prueba. Esto es aún mas importante en nuestro caso, ya que al crear escenas en tres dimensiones y utilizar recursos tan diversos como geometría, luces, multimedia, etc., y de origen tan variado como la importación desde otras aplicaciones o su definición por programación,…, es muy normal que el trabajo producido presente fallos y estados de ejecución inestables en el que puede llegar a hacer que el ordenador “se cuelgue”.

Otro aspecto a tener en cuenta, es que cuando ya tenemos realizado y probado nuestro trabajo, debemos intentar minimizar al máximo su tamaño. Por ejemplo, si no tiene una importancia capital, podríamos intentar utilizar imágenes con formatos que ocupen menos espacio que otros. Esto tendrá repercusiones en la rapidez de ejecución del entorno y por tanto en que el espectador tenga sensaciones más realistas en su inmersión en la realidad virtual.

También tendremos que revisar todos los elementos utilizados y compararlos con los que hemos importado, para si es el caso, borrar materiales que no visualizan en el trabajo final. Es una manera de compactar nuestra obra y evitar así problemas de ejecución.

En los casos que resolveremos en los temas siguientes, veremos cómo resolver estas cuestiones en aplicaciones distintas. Por ejemplo, en Director y en Second Life, tendremos que hacer lo antes comentado, es decir, comprobar constantemente que si por ejemplo, hemos realizado una escultura que gira sobre sí misma, pues que presenta el mismo comportamiento ante situaciones no habituales o interacciones no previstas por parte del espectador.

En resumen, debemos de intentar que la obra antes de publicarla y poderla presentar cumpla tres condiciones: que sea lo más probada posible, que ocupe el menor espacio que podamos asumir para no perder calidad y que no tengamos material inútil en nuestra escena.

12.2. Características de la publicación off line.

Las aplicaciones que producen realidad virtual pueden ser de muchos tipos. Las diferencias más importantes se derivan de que estén enfocadas a su reproducción desde Internet o no. Por ejemplo, los metaversos, son realidades virtuales desde y para la red. Por tanto, se necesita estar conectados para experimentar la inmersión en la escena 3D. Una solución no demasiado consoladora puede darse al crear un vídeo que permite filmar nuestras andanzas por el metaverso y por ejemplo, visitar una exposición. Es evidente que no es una solución que permita la misma experiencia sobre todo por la falta de interactividad. También podemos crear imágenes del interior del metaverso. La parte positiva es que dado que Internet se está universalizando, según pase el tiempo, este problema tenderá a minimizarse. Esto quiere decir, que por ejemplo, Second Life no se puede publicar off line.

Existen aplicaciones cuya publicación permite varios formatos. Este es el caso de Director. En estas aplicaciones se puede elegir fundamentalmente dos formas de publicación: proyector o mediante plugin. Es el caso de la mayoría de los videojuegos.

Un proyector es un fichero autoejecutable. En los sistemas operativos Windows son ficheros con la extensión .exe. Se basan en encapsular el trabajo y un plugin específico para cada sistema operativo en un solo archivo. La ventaja es que son archivos autocontenidos, por ello no hace falta ni instalarlos ni configurarlos. Se ejecutan con un simple doble click del ratón.

Estas características lo hacen idóneo para ser distribuido en formato CD-ROM o DVD. En este sentido podemos mejorar su facilidad de uso añadiendo un autoarranque. El autoarranque se realiza con un simple archivo de texto sin formato que contenga lo siguiente:

[autorun]
Open = fichero.exe

Estos proyectores también permiten ser colgados de una página web determinada pero su ejecución no se producirá "según se descargan", es decir, por streaming y esto conduce a que el espectador tendrá que esperar un tiempo para poder utilizar este fichero.

Si la opción es que se ejecuten previa instalación de un plugin, este puede estar preparado para ejecutar archivos on line y off line. Por ejemplo, en el caso de Director, el plugin necesario para su ejecución es Shockwave.

12.3. Características de la publicación on line.

Como hemos escrito en el punto anterior, existen aplicaciones que generan archivos que son ejecutables on line si el ordenador cliente tiene instalado un determinado plugin.

En el caso de Director, si el ordenador de espectador tiene instalado Shockwave, podrá ejecutar en modo streaming un fichero colgado por nosotros desde una página web. Además tenemos la posibilidad de que si no lo tiene instalado, se detecta y nos lleva a la página que permite descargar la última versión del plugin. Normalmente, este procedimiento de trabajo se puede generalizar a otras alternativas que existen en la actualidad para realizar este tipo de trabajos.

Por otra parte, si la escena 3D la hemos creado en el interior de un metaverso, la publicación en sí no existe, es decir, no se puede "empaquetar" un trozo del metaverso y hacer que no tengamos libertad para utilizarlo por completo, por lo que muchas veces lo que hacemos es que desde una página web, se crea un enlace a una determinada posición del interior del mundo virtual que sea nuestro espacio de trabajo. Este mismo pasa en Second Life.

Todas estas alternativas se estudiarán en los casos prácticos.

CAPÍTULO 5:

CURSO: IMPLEMENTACIÓN CON APLICACIÓN DE MAX Y DIRECTOR

Unidad 1: Introducción a la implementación MAX - DIRECTOR.

1. Introducción a las aplicaciones de creación y programación 3D.
2. La aplicación 3D Studio MAX.
3. La aplicación Director.
4. Respuesta de la implementación a los requisitos impuestos en la tesis.
5. Otras alternativas.
6. Arte en MAX y Director.
7. Problemas que presenta la implementación MAX-DIRECTOR

1. Introducción a las aplicaciones de creación y programación 3D.

La primera de las soluciones que proponemos para crear objetos en 2 y 3 dimensiones y poder exponerlo interactivamente desde disco duro y/o en Internet, supone la realización de los proyectos usando dos aplicaciones: Autodesk® 3ds MAX® y Adobe® Director®. Se trata de dos aplicaciones muy conocidas y consolidadas en sus respectivos campos del software actual.

El autor comenzará creando una escena tridimensional no interactiva en MAX, y luego exportaremos esa escena a Director que le proporcionará interactividad y distintas formas de exponerla.

Esta solución no se ha tomado a la ligera. Existen muchos programas para crear escenas 3D (aunque MAX es el más utilizado) pero no tanto que realicen la misma función que Director y además, proporciona una forma fácil y eficiente de importar archivos de MAX.

2. La aplicación 3D Studio MAX.

MAX es un programa para crear escenas en 3D y animarlas. Existen muchos otros, incluso de su misma empresa, Autodesk®, por ejemplo, los muy conocidos Autocad® y Maya®. También podemos tener en cuenta software de otras empresas como Rhinoceros, Blender, Luxology, Cinema 4D, Lightwave, Softimage y otros. Pero 3D Studio MAX tiene una ventaja sobre todos ellos: la popularidad. Esto hace que sea el programa por el que casi todos los diseñadores comienzan y el que tiene más libros en el mercado y más tutoriales en Internet. Esto también se traduce en que tiene la base de usuarios más amplia con diferencia de todos los competidores. Tiene muchas desventajas, como el precio, su antigüedad y muchas otras, pero puede considerarse un estándar de facto.

En la actualidad (septiembre de 2009), la última versión salida al mercado es la 2009 que será la que utilicemos en nuestros proyectos, aunque los hemos desarrollado durante un largo periodo y por tanto, probado con otras versiones anteriores.

Imagen 5.1.1: Iniciando Autodesk® 3ds MAX 2009

En la imagen que sigue podemos observar la interfaz que presenta MAX. Lo primero que llama la atención es que gran parte de la pantalla está ocupada por cuatro grandes visores para que el autor pueda ver la escena tridimensional desde muchos ángulos a la vez ya que lo principal del trabajo de un diseñador 3D es tener una visión del trabajo en las 3 dimensiones.

3D Studio MAX es un programa que permite crear modelos 3D siguiendo distintas técnicas. También permite crear cámaras, luces, texturas y todo ello además se puede animar, por lo que es complejo y el tiempo requerido en dominarlo totalmente es largo. Esto hace que normalmente los profesionales se especialicen en una de sus múltiples facetas.

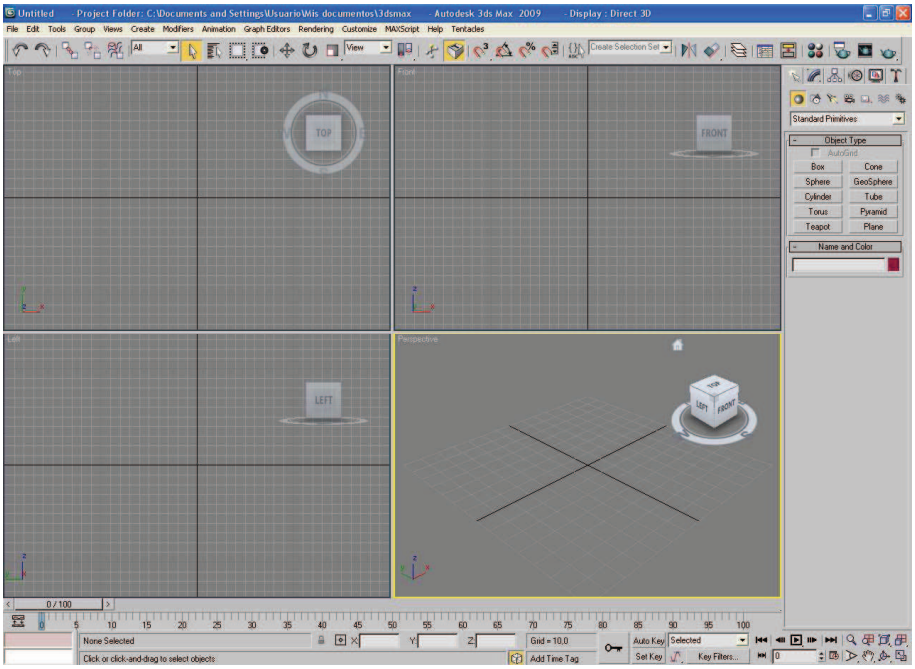


Imagen 5.1.2: Interfaz inicial de 3D Studio MAX 2009

Para aprender más sobre el programa se puede empezar por su página en Wikipedia que es: http://es.wikipedia.org/wiki/3D_Studio_Max. Siendo su web oficial la que presenta Autodesk en: http://www.autodesk.es/adsk/servlet/index?siteID=455755&id=12341473.

3. La aplicación Director.

La segunda parte del proyecto tendrá lugar con uno de las aplicaciones más útiles del mercado actual y sin embargo, relativamente poco conocida: Director. Esta aplicación que es difícil de clasificar tiene un amplio recorrido que se remonta a mediados de los años 80, siendo el "buque insignia" de la prestigiosa empresa Macromedia®. Esta empresa creo una serie de software que tuvo un éxito sin precedentes en el mundo del diseño como Flash, Dreamweaver, Freehand, Firework, Authorware y otros y en 2005 fue adquirida por Adobe. Académicamente hablando, podemos decir que Director es una aplicación de desarrollo de software multimedia.

Imagen 5.1.3: Iniciando Adobe® Director® 11

De Director se puede decir que es el "hermano mayor" de Flash®, ya que en un principio, Flash se diseño como un conjunto de Director para hacer solo una de las cosas que podía hacer Director: películas interactivas vectoriales. Pero el éxito de Flash en una época dónde Internet necesitaba de dinamismo como agua de mayo, eclipsó las posibilidades de Director, que si

bien podía funcionar on y off line, necesitaba un plugin para su utilización (igual que Flash) llamado Shockwave , pero este era de un tamaño considerablemente mayor, y no tuvo la popularidad del Flash player. Además, en Director se requiere en general de personal más especializado en programación, ya que los proyectos grandes requieren de ingeniería del software. Como colofón a lo escrito, diremos que Shockwave permite ejecutar aplicaciones Flash pero no al revés, Flash no puede ejecutar archivos Shockwave de Director.

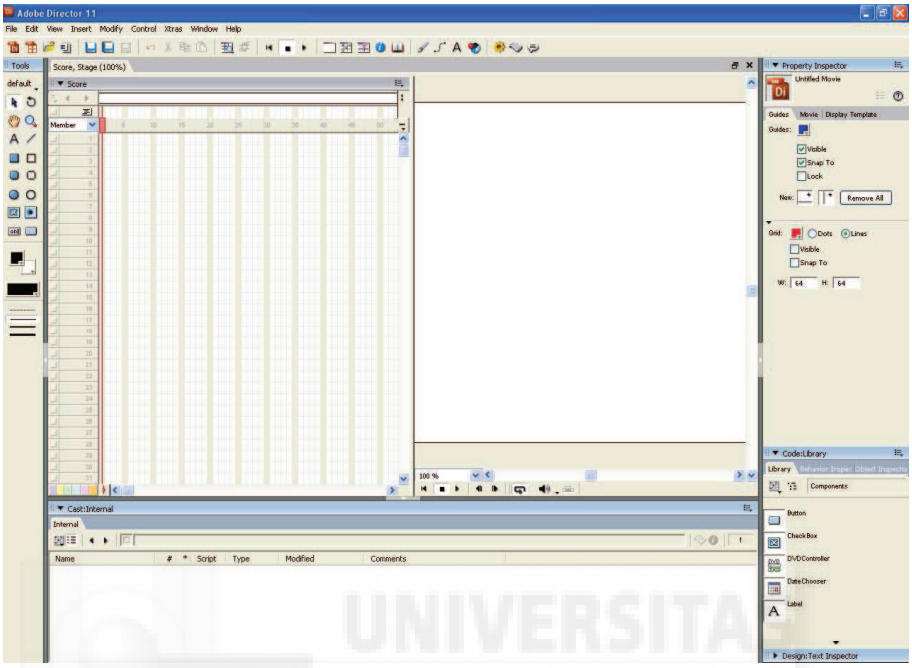


Imagen 5.1.4: Interfaz inicial de Director® 11

De todas formas, Director ha seguido creciendo y evolucionando, dando lugar a un software que hoy en día y sobre todo desde la versión 8.5 permite trabajar con 3D sobre todo con un software llamado Shockwave 3D que proporciona un player para los navegadores más conocidos, una ampliación de su lenguaje de programación llamado Lingo con objetos y métodos 3D y algún otro software de apoyo. Esta entrada de Director en las 3D produjo la creación de juegos on-line y otras aplicaciones en entornos mucho más populares de lo que se hacía hasta entonces.

La última versión salida al mercado de Director es la 11.5 que será la que utilicemos en nuestros proyectos, aunque lo hemos venido utilizando desde la versión 8.5.

Para aprender más podemos comenzar por la página dedicada a este programa en wikipedia http://es.wikipedia.org/wiki/Adobe_Director y su página oficial es la que presenta Adobe en http://www.adobe.com/es/products/director/.

La entrada de Director en el mundo 3D trajo de la mano la posibilidad de poder importar fichero de MAX y por tanto vamos a utilizar este dúo de grandes programas como una de las posibles estrategias para crear nuestras obras.

4. Respuesta de la implementación a los requisitos impuestos en la tesis.

En este documento escribimos que las soluciones propuestas que conformarían el curso debían cumplir una serie de requisitos. A continuación recordamos estas condiciones:

1. Los proyectos podrán ser en dos o más comúnmente en tres dimensiones, por lo tanto pueden ser pinturas y esculturas digitales. Esta escena no será fija, sino que el espectador podrá moverse con libertad por el espacio que rodea a las obras.

2. La segunda condición es que con el fin de que la obra llegue aun mayor número de personas el proyecto podrá visitarse por Internet o ser ejecutado desde DVD/CD-ROM y por supuesto, desde el disco duro de un ordenador cualquiera.

3. El sistema no debe obligar al autor a trabajar con un software específico. Esto quiere decir que las aplicaciones usadas deben ser en lo posible estándar, sino oficialmente, si de facto.

4. Todo el proyecto debe ser interactivo. Esto quiere decir que la ejecución puede ser totalmente controlada por el espectador.

5. El contenido del trabajo puede incluir diversos medios, esto es, puede ser de tipo multimedia, con vídeos, sonidos, etc.

6. La solución debe permitir ser programada para que el autor pueda tener el mayor control posible de su obra, de su entorno y de la ejecución de la misma.

La solución basada en MAX-Director, cumple estas 6 condiciones. Para ver que a si es, vamos a estudiarlas punto por punto. En este caso como son dos aplicaciones que vamos a utilizar de forma secuencial, primero MAX y luego Director, debemos comprobar las condiciones en este último programa ya que por ejemplo, no sirve de nada que MAX trabaje en un espacio tridimensional si luego Director no lo permite. Seguidamente vamos a ilustrar con imágenes el cumplimiento de los requisitos, y por ello, estas ilustraciones pertenecen a las obras en Director.

1. Espacio y obras tridimensionales.

Esta condición se cumple desde el inicio del proyecto, ya que trabajamos con MAX en tres dimensiones todo el rato y en Director, podemos trabajar con muchos componentes multimedia, pero uno en particular, llamado Shockwave 3D puede contener escenas 3D.

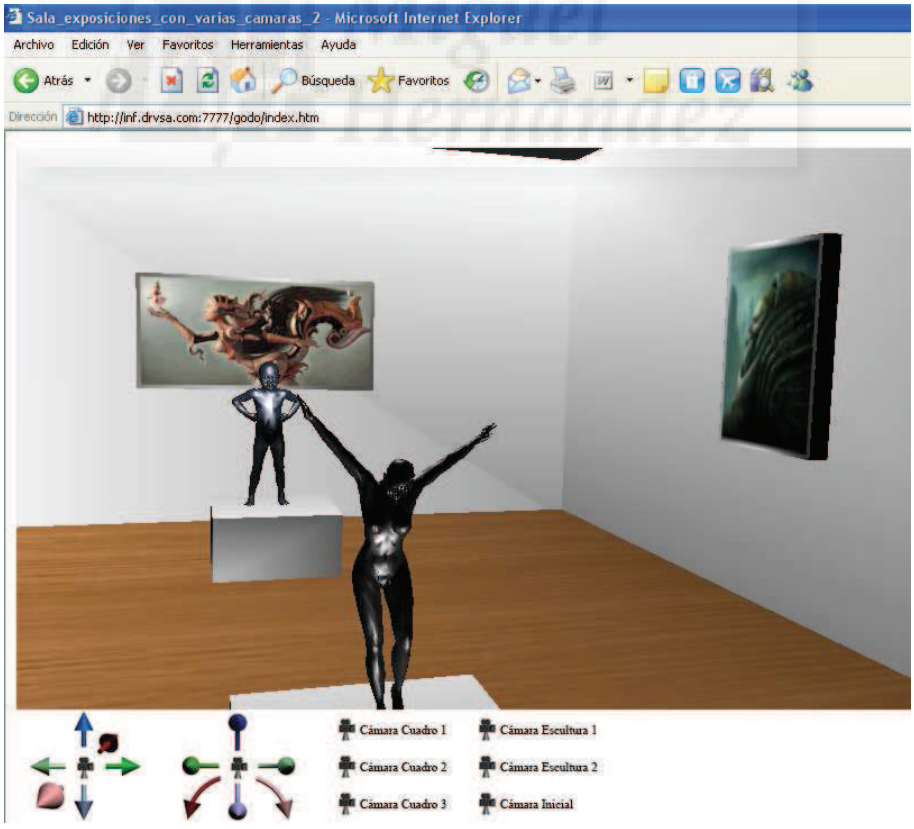


Imagen 5.1.5: Exhibición en Internet de obras 2D y 3D en un entorno 3D

En la imagen anterior podemos observar un proyecto de una pequeña sala dónde se exhiben unas obras. Tenemos tres cuadros (aunque en la imagen solo se observan 2) que por supuesto son 2D, también tenemos dos estatuas (un niño y una mujer) realizadas en tres dimensiones. Sin embargo, la muestra en sí es en 3 dimensiones y nos podemos trasladar, girar,… en los tres ejes para observar, acercarnos y navegar por la exhibición.

2. Ejecución on / off line.

Director permite que se pueda publicar su trabajo, es decir, ejecutarlos desde un ordenador ya sea desde cualquier tipo de memoria como un disco duro, una memoria USB, un CD-ROM, un DVD, incluyendo un proyector que uniéndose al trabajo realizado por el autor, forma un solo archivo y de esta manera es mucho más compacto.

Por otra parte, también permite instalar un plugin para visualizar el trabajo en Internet como muestra la imagen superior, que es una captura de pantalla de una ejecución on-line.

3. Software reconocido.

Los dos programas que utilizamos son muy conocidos en sus respectivas áreas de software. Tienen una base muy grande de usuarios y se apoyan en empresas consolidadas. Por lo tanto, podemos escribir que es difícil encontrar software más seguro al que apostar. De todas formas, en esta área nunca se sabe. Es verdad que podemos dedicar un tiempo a su aprendizaje y que al cabo de unos meses puede desaparecer. En este sentido, el mayor peligro lo tenemos en Director, que si bien es un software muy reconocido y que lleva más de 20 años en el mercado, puede ser que no sea versionado con la frecuencia que debiera. De todas formas, dejaría un hueco en un nicho del mercado del software que no llenaría ningún programa, por lo que es difícil que esto suceda en breve. Puede ser que con el tiempo, Flash vaya sustituyendo sus características más importantes y al final, parte de sus grandes capacidades sean asumidas por Flash, pero de momento y a decir de Adobe, a Director le quedan muchos años por delante.

4. Ser interactivo.

En la imagen 5.1.5 podemos ver debajo del entorno 3D unos botones para dirigir una cámara que permite sobrevolar el espacio 3D.

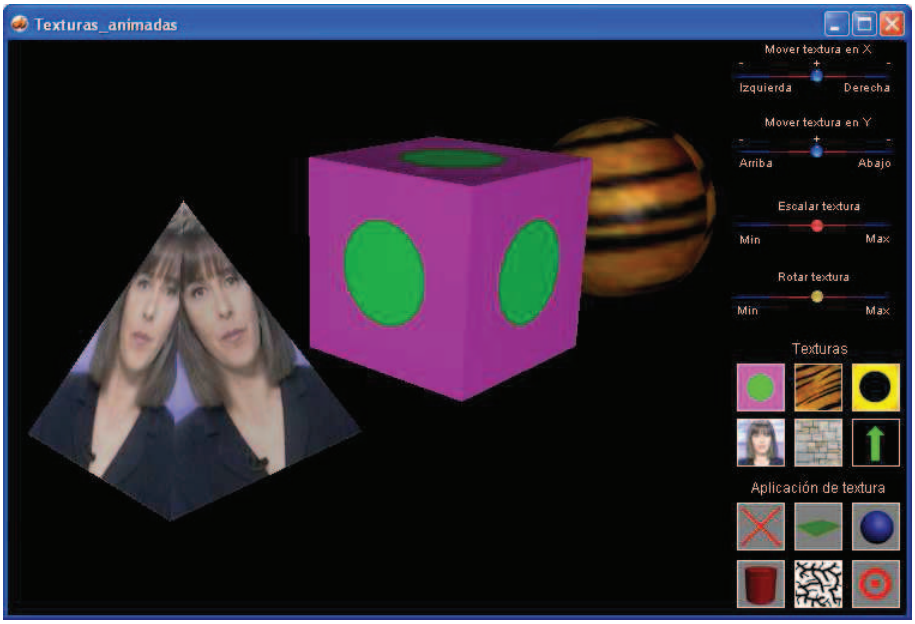


Imagen 5.1.6: Aplicación de texturas animadas a modelos 3D

En la imagen inferior podemos observar uno de los casos prácticos desarrollados. Se pueden ver que los modelos tridimensionales han sido texturizados con los elementos de la derecha y además, tenemos distintos tipos de controles como son los botones deslizantes de la parte superior derecha. Estos botones permiten dirigir la ejecución de la muestra.
Estas son solo dos pequeñas muestras de las posibilidades de interacción que nos ofrece Director y por lo tanto este requerimiento está de sobra cumplido.

5. Admitir la combinación de archivos multimedia.

En la imagen anterior, también se evidencia que podemos utilizar imágenes que se integran junto con los modelos en 3D. Como veremos en los casos prácticos pertinentes, se puede ejecutar sonido a voluntad del autor, por ejemplo, una música de fondo o como refuerzo auditivo de la interacción. En el caso concreto de la imagen de arriba se observa la aplicación de texturas animadas, lo que proporciona soporte para la exhibición de animaciones. Podemos ejecutar archivos de tipo swf, lo cual permite unir Director con los proyectos de Flash. En fin, que los medios de trabajo multimedia están soportados por esta alternativa.

6. El sistema debe poderse programar.

En la imagen inferior se puede observar como la ventana con fondo negro, muestra un sistema de partículas que trata de recrear una llama.

Esta llama ha sido completamente creada mediante programación. Creemos que no existe mayor prueba de lo útil y la potencialidad que proporciona el poder programar en Director cuando queremos obtener un control prácticamente total de nuestro trabajo.

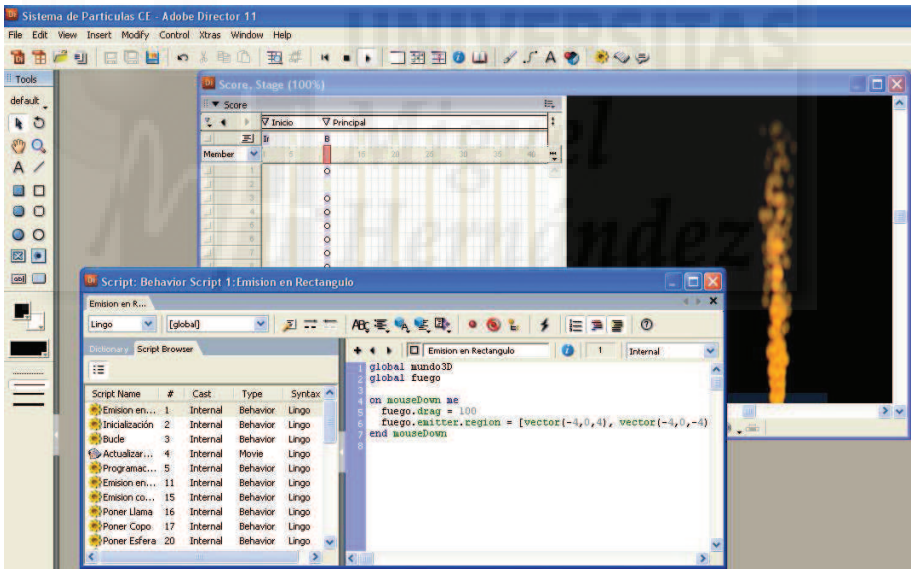


Imagen 5.1.7: Programando en Lingo un sistema de partículas de tipo Fuego

5. Otras alternativas.

En estos momentos, es difícil pensar en alternativas a esta solución. En el presente texto vamos a proponer otra alternativa totalmente distinta que cumple también todos los requisitos pero desde una perspectiva totalmente distinta: el metaverso Second Life.

La diferencia fundamental entre una y otra es que en este caso usamos programas comprados, es decir, software que tiene un valor y que está sustentado por grandes compañías de software. Sin embargo, Second Life es gratuito en cuanto a su instalación y ejecución. Tendremos que pagar solo si queremos poseer un espacio propio. Por otra parte, Second Life tiene un gran impedimento y es que siempre se ejecuta on-line, y esto a veces puede ser un gran problema, ya que no podemos exhibir nuestra obra de manera independiente. No

podemos crear una escultura virtual y venderla sin proporcionar a la vez una conexión por Internet. Esto hace que sea una solución menos autónoma y presenta algunos problemas de difícil solución, por ejemplo, la rapidez de respuesta en las interacciones dependerá del tipo de conexión utilizada por el cliente y por lo tanto no es controlable por el autor.
De todas formas, como aún no conocemos Second Life, esperaremos a su estudio para hacer una comparación entre ambas alternativas.

Otras alternativas parecidas a esta supondrían el sustituir a MAX y/o Director por otros programas. Aquí si podemos encontrar otras opciones, ya que el mercado de las aplicaciones cambia constantemente. Por ejemplo, las nuevas características de Flash, hacen que sea una elección digna de estudio para realizar algún proyecto.

También podríamos utilizar los llamados motores de videojuegos para realizar esta tarea. Al fin y al cabo, la idea principal es utilizar la tecnología de los videojuegos pero en vez de buscar la parte lúdica, buscaríamos la creación digital. Los problemas que presenta esta solución son de distinto tipo: por un lado son programas bastante caros, por otro, existe una total falta de estandarización en cuanto a lenguajes de programación a utilizar, métodos de creación de la escena 3D, etc. En pocas palabras: cada uno funciona de una manera distinta, lo que hace que exista falta de literatura de apoyo para su aprendizaje y carencia de solidez en el mercado, con lo que un esfuerzo muy grande por nuestra parte puede no ser recompensado si a los pocos años estos programas desaparecen.

6. Creación en MAX y Director.

En al primera parte de esta tesis ya incluimos un pequeño estudio sobre creaciones con Director y sus autores más reconocidos. Vamos a mostrar aquí dos trabajos para animar al lector en el estudio de estos programas como solución a sus pretensiones artísticas, adelantando que todo lo que podemos realizar con MAX no podemos exportarlo tal cual a Director tal como veremos en el siguiente punto.

Imagen 5.1.8: Recreación de “la dama de Elche” con 3ds MAX

En la imagen 5.1.8 se puede visualizar una imagen que demuestra que MAX tiene propiedades suficientes como para poder realizar modelos tridimensionales de calidad. Esta imagen la hemos obtenido buscando en Internet. Son muchos los artistas que utilizan 3ds MAX como base para realizar arte digital tridimensional.

En la imagen que sigue podemos ver algunos de los momentos de una presentación artística realizada con Director, si bien, es un poco antigua, creemos que expone muy bien las posibilidades que ofrece en cuanto al tratamiento de la multimedia, ya que incorpora música

ambiental y también como muestra artística en sí misma. Para visitar esta muestra, la dirección es http://toxi.co.uk/skyy/

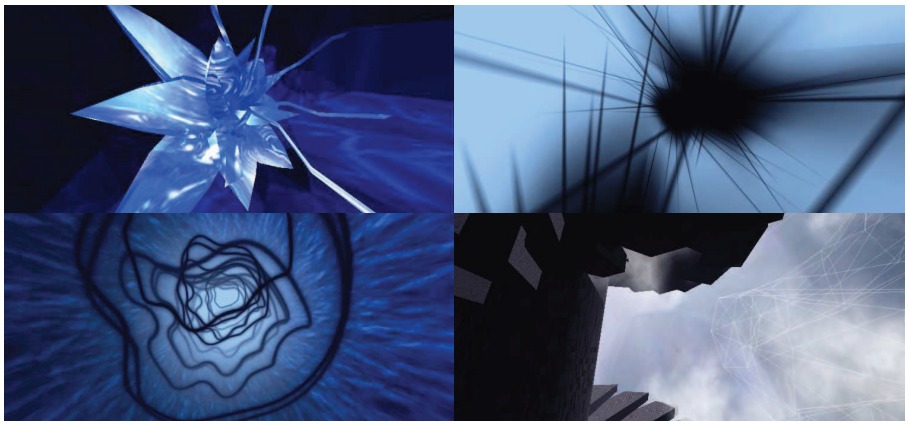


Imagen 5.1.9: Obra multimedia realizada con Director

También podemos encontrar espacios tridimensionales con formato de museo en Internet realizados Shockwave 3D dentro de Director como se visualiza en la imagen inferior y que podemos visitar en la web http://homepage.mac.com/ikiboy/ShockDemo/Museum.html

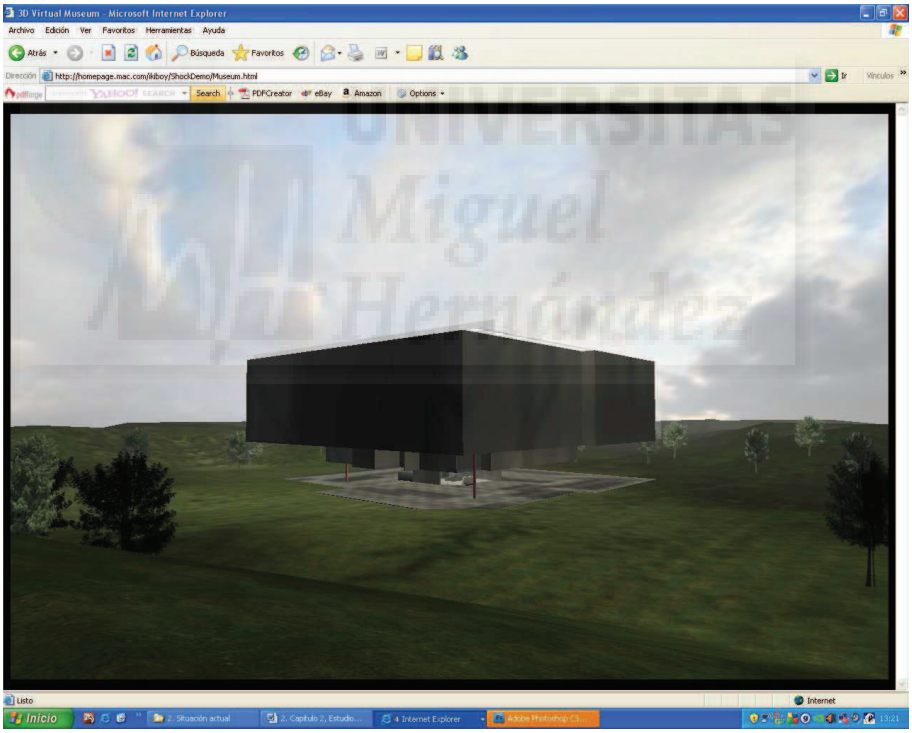


Imagen 5.1.10: Museo 3D interactivo en Internet realizado con Director

Incluso, podemos encontrar empresas que se dedican a montar espacios 3D a modo de ferias, exposición de muestras o museos como es el caso de Tradky.

Imagen 5.1.11: Museo 3D interactivo en Internet realizado por Tradky

Esta empresa crea los espacios y dota al conjunto de un sistema de navegación desarrollado por ellos mismos para agilizar los movimientos de los espectadores tal como se muestra en la imagen superior. Podemos visitar su web en http://www.tradky.com

7. Problemas y limitaciones del formato Shockwave 3D.

Desgraciadamente, a día de hoy, no podemos exportar todas las características que conseguimos con 3ds MAX a Director. Al fin y al cabo, los programas de modelado y animación que tienen características profesionales están enfocados a la creación de imágenes fijas o películas. Esto conduce a que existan claramente dos fases: producción y representación. En Director esto no es así, ya que debe producir y representar los cambios “instantáneamente”, es decir al tiempo que se van generando. Esto tiene un precio en rapidez de cálculo y se paga en limitaciones, especialmente en los mapas de texturas y la luces.

En la actualidad, ni los mejores juegos del mercado tienen capacidad para generar imágenes fotorealistas y representarlas en tiempo real. Por eso utilizan técnicas de pre-mapeado y objetos con muy pocos polígonos. Cualquier escena, en cuanto descuidamos un poco este aspecto puede tener 1000 polígonos y a partir de esta cifra tendremos problemas.

Alguno de los aspectos donde sin duda nos encontraremos con dificultades son los siguientes:

1. Mapas de relieve: como se muestra en la imagen de arriba, no se pueden representar en Director. Esto es un inconveniente muy grande, ya que en estos mapas reside gran parte del realismo de una escena. De todas formas, los mapas de relieve hasta cierto punto se pueden simular creando mapas difusos que incluyan sombras texturizadas. Bien es cierto que esta técnica requiere de mucha habilidad en el tratamiento de imagen y de todas formas, el resultado no suele ser satisfactorio al cien por cien.

Imagen 5.1.12: Escena con mapas de relieve en MAX y al exportar a Director

2. Raytrace: también hay que evitar utilizar mapas de tipo Raytrace. En las siguientes vistas vemos cómo se representan en MAX estas características y como las visualiza Director. Esta es una técnica avanzada de mapeado mediante trazado de rayos que como se puede apreciar, Director no soporta. Se utiliza para materiales con mucha refracción y con cierto grado de transparencia, buscando que sean fotorealistas.

Imagen 5.1.13: Escena con mapas Raytrace en MAX y al exportar a Director

3. Luces tipo foco y sombras: tampoco debemos utilizar luces especiales, como por ejemplo las luces tipo foco, ya sean libres o no. También las sombras nos pueden dar problemas. Veremos más adelante que en algunos casos podemos simular sombras simples en tiempo real mediante texturas.

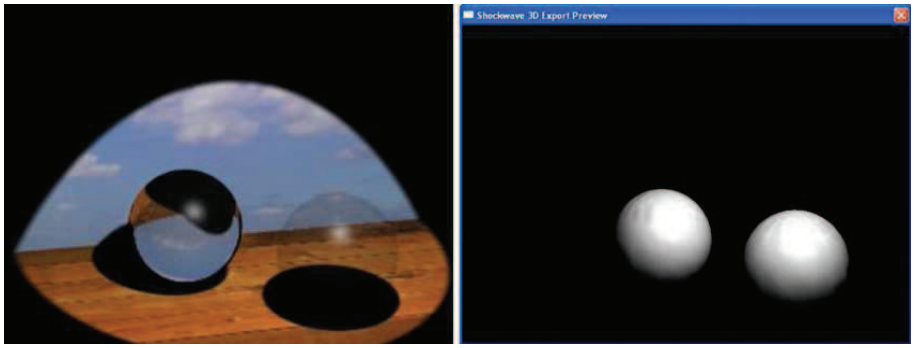


Imagen 5.1.14: Escena con focos y sombras en MAX y al exportar a Director

4. Luces tintadas: en la siguiente imagen se puede observar que las luces tintadas o con atenuación, aunque no son ignoradas por Director, no tienen el efecto que deberían. Se puede apreciar cómo los objetos que reciben la luz se tintan del color de esta, pero al no tenerse en cuenta las sombras, todo el efecto se viene abajo y deja de ser útil.

Resumiendo, tanto en los mapas especiales que no sea el de tipo difuso, como en las luces y en las sombras, casi siempre tendremos problemas al exportar nuestra escena a Director.

La cantidad de luces en Director está limitada a 7 y se recomienda crearlas directamente en este programa, ya que tiene una importancia muy grande en la visión de la escena. Sin embargo las cámaras es recomendable (al menos la principal) crearlas en 3ds MAX y exportarlas a Director.

Por todas estas circunstancias tenemos que tener presente antes de exportar a Director que podemos previsualizar cómo quedará nuestro trabajo en formato w3d y hacer los cambios pertinentes.

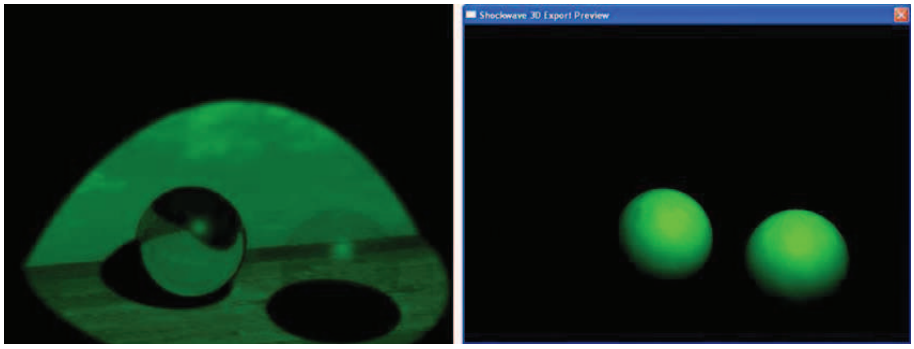


Imagen 5.1.15: Escena con luces tintadas en MAX y al exportar a Director

Unidad 2: Flujo de trabajo de la implementación MAX - DIRECTOR.

Introducción teórica:

1. Realizar una obra en MAX.
2. Exportar a Shockwave 3D.
3. Construir la presentación del trabajo Shockwave 3D en Director.
4. Publicarlo en un proyector.

Casos prácticos:

Caso práctico 2.1: Flujo de trabajo.

1. Realizar una obra en MAX.

En esta unidad asumiremos que sabemos cómo crear una obra en MAX en tres dimensiones. Por lo tanto, como el objeto es dar una visión general del flujo de trabajo, la creación de la obra en sí, no la vamos a estudiar en este punto y nos remitimos a la unidad 3 de este mismo curso.

La aplicación 3ds MAX es de suma importancia para la obra, ya que creamos su geometría, las texturas y las luces que luego utilizaremos en Director para motar la obra. La única fase que no tiene mucha importancia en el resultado final será la representación, ya que esta no será la misma en Director. Cómo ya comentamos en la unidad teórica precedente, Director no puede realizar ciertas tareas que sí permite MAX, como son la representación de sombras en las luces, los mapas de relieve, etc. A cambio, permite la interactividad y la ejecución en Internet.

Para ello ejecutaremos el programa 3ds MAX en la versión 2009. Esta versión incluye infinidad de características que no vamos a utilizar es este curso, por lo que versiones anteriores son también válidas.

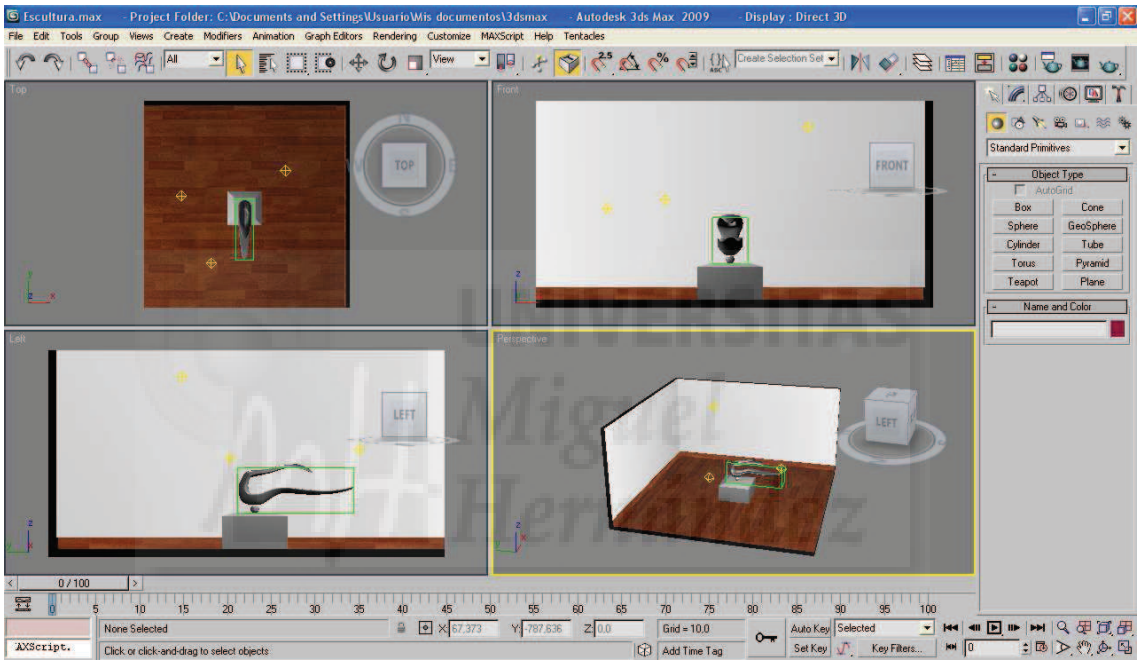


Imagen 5.2.1: Interface de 3ds MAX 2009

El flujo de trabajo de MAX puede ser variable, aunque la secuencia lógica de los pasos a dar son: primero la creación del modelado, luego texturizamos la obra y por último creamos las luces necesarias.

Cómo ya hemos señalado, tenemos características en MAX que luego no podremos utilizar en Director, por lo que es buena idea no emplearlas. Por ejemplo, no utilizaremos sombras en las luces, a menos que estemos dispuestos a recrearlas en Director por medios distintos de los empleados en MAX.

- **Modelado:**

En este punto, se hace evidente la pericia de un buen modelador, es una tarea que puede ser muy laboriosa y especializada y dónde se pueden utilizar muchas técnicas dependiendo del modelo a trabajar como son los objetos poligonales de malla, las nurbs, etc.

Una cuestión a tener en cuenta son las unidades en que se mide la escena. Estas se pueden cambiar en Customize > Units Setup. Esto es importante en trabajos profesionales donde no es lo mismo trabajar en proyectos arquitectónicos (metros) que en proyectos de joyería

(milímetros). Lo más importante no son las dimensiones “reales” sino las comparadas entre los objetos, por lo que es necesario tomar siempre una referencia válida.

Para modelar utilizaremos el panel llamado “Crear” que permite acceder a distintos modos de generar geometría 3D. Conviene ir poniendo nombres a todos los objetos que vamos creando ya que estos nombres se exportan junto con los modelos y sirven para identificarlos y en Director la programación se verá facilitada por estas referencias.

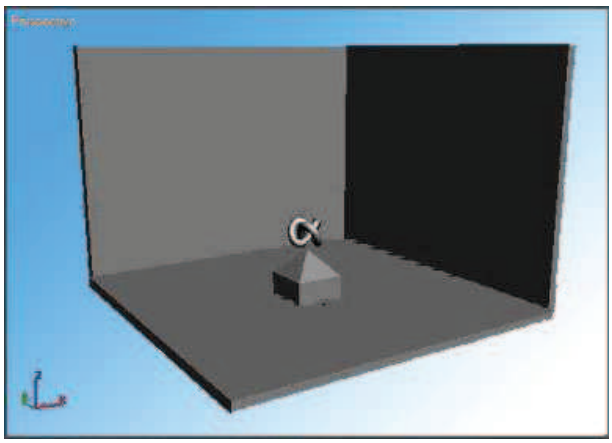


Imagen 5.2.2: Modelado de una obra

◆ **Texturizado:**

La texturización es un procedimiento que se reduce a conseguir texturas o crear materiales artificiales y luego a aplicarlas correctamente sobre la superficie de los objetos 3D.

Es imprescindible contar con buenos mapas para dar realismo a nuestros trabajos. Un mapa es una imagen bitmap. Normalmente se guardan como ficheros .jpg. Suelen ser fotografías, de hecho, para modelos recreados de la realidad, normalmente, los artistas se “fabrican” sus mapas, fotografiando el original.

Texturizar es comparable a envolver el objeto 3D con un papel como si fuera un regalo y la textura fuera el papel de regalo. Es evidente que la forma de “envolverlo” dependerá de la forma básica del objeto a texturizar. Para que 3ds MAX sepa cómo envolver el objeto, el autor crea unas “coordenadas de mapeo”. Estas coordenadas aportan información de por dónde se empezará a envolver y cómo, es decir, si se hará con planos, como una caja, etc.

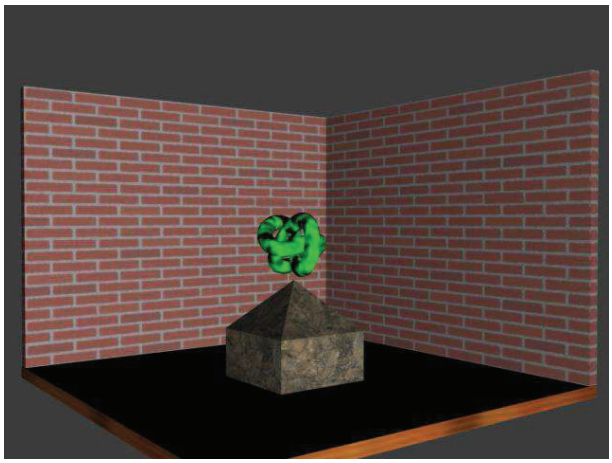


Imagen 5.2.3: Texturizado de una obra

Para texturizar, 3ds MAX aporta un módulo llamado “editor de materiales” que se abre pulsando la tecla “M”. Debemos de ser conscientes que Director no va a importar todas las texturas que sí puede tratar 3ds MAX, siendo las texturas de relieve las que más vamos a echar en falta.

- **Iluminación:**

La iluminación en MAX es muy importante, en este sentido es necesario hacer constantes representaciones para ver cómo va quedando nuestra obra. Si comparamos las imágenes del texturizado y cómo queda después de iluminar correctamente nuestra escena, llegaremos a la conclusión de que es imprescindible hacer bien este paso.

También tenemos que tener en cuenta que las iluminaciones tal como las muestra 3ds MAX no van a ser importadas a Director, sobre todo la aplicación automática de sombras será un problema que trataremos en el tema específico de iluminación.

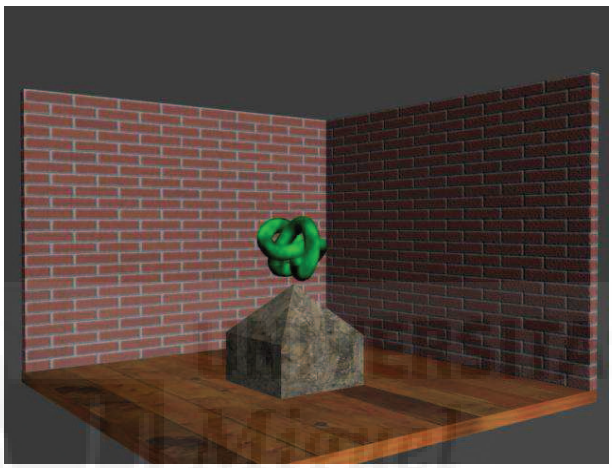


Imagen 5.2.4: Iluminación de la obra

- **Representación:**

La representación es el proceso por el que se visualiza el trabajo terminado. Se aplican los mapas, las luces, efectos de postproducción,… y todo lo que intervenga en la escena. Es el sistema de visualización final. La representación (o render como se conoce en la jerga propia de los autores 3D) supone un trabajo muy complicado desde el punto de vista de cálculo computacional que normalmente se realiza con programas externos que se complementan con MAX como sistemas plug-ins. Esta representación lleva bastante tiempo si los parámetros con que se realiza la representación son de alta calidad.

Cómo hemos escrito anteriormente, en los trabajos que nos ocupan no tiene mucha importancia ya que la representación final está a cargo de Director, aunque sí debemos utilizarla para comprobar el modelado, el texturizado y sobre todo, la iluminación de la escena. Se puede obtener una sola imagen en los formatos más utilizados hoy en día (como .jpg, .bmp, .tiff) y con los tamaños y calidades que deseemos.

También se puede representar una serie de imágenes, lo que producirá una película que podemos comprimir en formato avi, mov, etc. Las opciones de la ventana Representación no las vamos a explicar aquí, pero en general dan buen resultado los valores por defecto.

Para representar la escena pulsamos F10 o accedemos al menú Rendering > Render, apareciendo la ventana que se observa en la siguiente imagen.

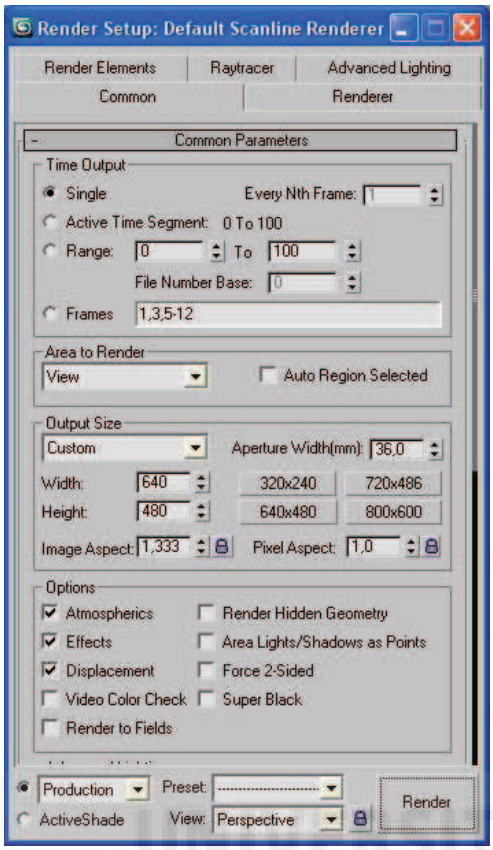


Imagen 5.2.5: Ventana "Render Setup" para controlar la representación

2. Exportar a Shockwave 3D.

A la hora de exportar e importar información entre aplicaciones debemos tener en cuenta las empresas que están detrás de esos programas. En nuestro caso, MAX es de la empresa Autodesk® y Director pertenece a Adobe®, líderes cada una en su sector. Esto es muy importante ya que a las dos empresas les conviene que sus productos sean de amplio "recorrido", que se puedan utilizar en distintos trabajos y que resulten efectivos para su principal función, como es el caso que nos ocupa. Es por esto que desde MAX no existe ningún problema para exportar a Director ya desde las primeras versiones de estos dos programas y auguramos que no tendremos ninguna traba en las posteriores versiones.

Para exportar una escena de MAX a Director lo haremos a través del formato Shockwave 3D que es un estándar de facto en este tipo de trabajos.

Exportar de MAX a Shockwave es tan sencillo como ejecutar el comando File > Export. Aparecerá la ventana "Select File to Export" que es la típica ventana para guardar un archivo. Debemos acceder a la lista desplegable de tipo de archivos y elegir la opción Shockwave 3D Scene Export (.w3d). Pondremos el nombre que deseemos para nuestro fichero y aparecerá la ventana llamada "Shockwave 3D Scene Options" que es dónde se encuentran todas las opciones de exportación como se puede ver en la siguiente imagen.

Esta ventana con todas sus opciones se explican con detalle en la unidad teórica 7. Podemos pulsar el botón "Preview" para ver como quedará una vista previa de nuestra escena. Cuando pulsemos el botón "Export" tendremos como resultado un archivo .w3d que podremos importar desde Director para trabajar con él.

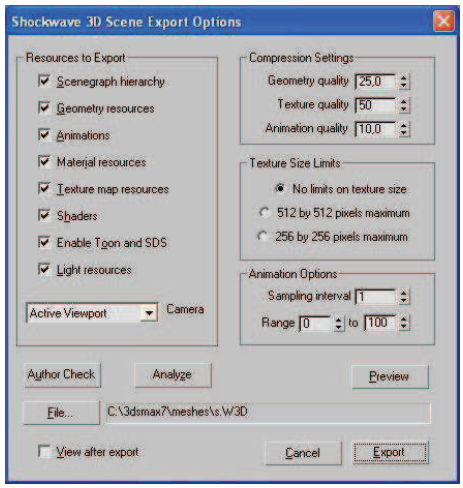


Imagen 5.2.6: La caja de diálogo Shockwave 3D Scene Options

3. Construir la presentación del trabajo Shockwave 3D en Director.

En este punto abrimos Director® 11. Este es el programa que utilizaremos para "fabricar" nuestra presentación. Director es desde sus versiones iniciales, un potente software para la creación de todo tipo de aplicaciones multimedia. Entre otros formatos, permite trabajar con escenas tridimensionales con el formato Shockwave 3D que es el que utilizaremos para nuestros fines.

En terminología de Director, al trabajo que se desarrolla se le denomina película (como en el software Flash) pero a diferencia de este, todos los elementos que la componen tienen una importarte correspondencia con los elementos de un film.

Para importar un fichero .w3d basta con ejecutar el comando File > Import y tras seleccionarlo, pulsar el botón "Import". Al principio no veremos que pase nada. Esto es porque el fichero importado pasa a ser un miembro de nuestra "película" de Director y no lo delata de ninguna manera, no presenta ningún informe o aviso, simplemente lo lleva a cabo.

Si no vemos la ventana de salida de Director, deberíamos mostrarla, para ello, utilizaremos el comando Windows > Stage. El "stage" es el escenario de nuestra presentación. Este escenario podemos modificarlo con el comando Modify > Movie > Properties.

También es necesario utilizar la ventana "Score", que es dónde podemos ver el esquema de presentación de los distintos elementos multimedia que formarán parte del trabajo. Para visualizar esta ventana usamos el comando Windows > Score. En la parte inferior veremos nuestra escena importada y seleccionándola la arrastramos hasta la ventana "Stage".

Podremos observar en la ventana "Score" que la aparición del miembro o "Sprite" que corresponde con la escena 3D ocupa una serie de fotogramas o "frames" de película.

Tenemos que redimensionarla y colocarla para que ocupe todo el "Stage". Nos podemos ayudar de las propiedades del "Sprite". Una vez hecho esto podemos pulsar al botón "Play" para ver cómo quedaría la exposición de nuestro trabajo.

Para visualizar la obra interactivamente en 3 dimensiones podemos usar Windows > Shockwave 3D. Aparecerá una pequeña ventana que tiene 3 botones principales para manejar la cámara: trasladarla en profundidad, hacia los lados y rotarla. En la imagen anterior se aprecia cómo en la ventana "Stage" aparece la escultura de forma estática, mientras que en la pequeña aparece con una perspectiva elegida por el usuario. Guardaremos el fichero de director con el comando File > Save as > y elegimos el nombre que queramos. El fichero tendrá extensión .dir

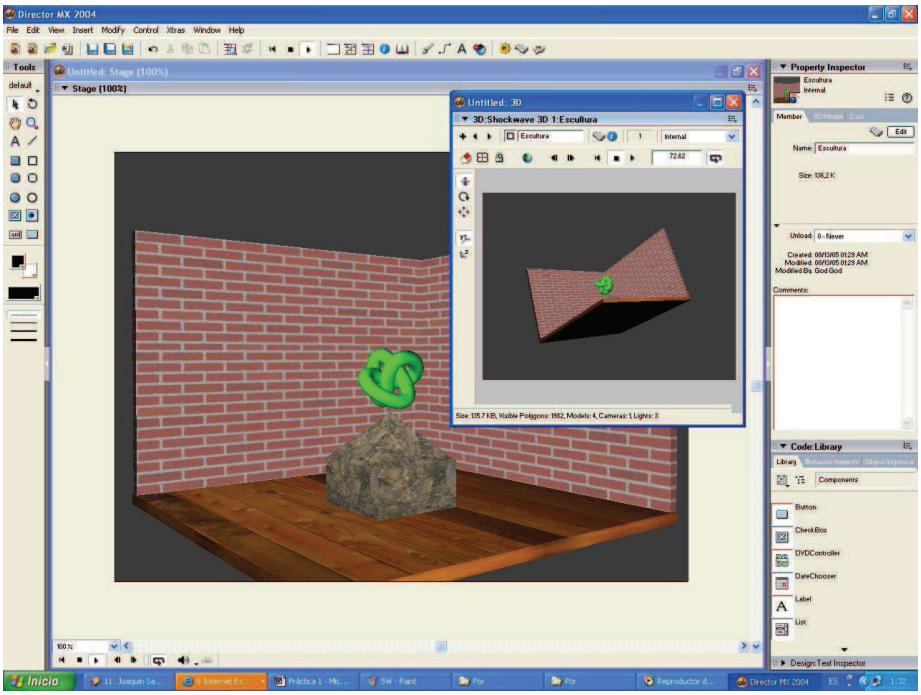


Imagen 5.2.7: Vista en 3D de nuestra escena

4. Publicarlo en un proyector.

Se trata de poder ver y manipular la película en dos posibles medios: en Internet, es decir, on-line o en un proyector "local", ya sea en el disco duro o más frecuentemente en cdrom/dvd. Por ahora vamos a utilizar esta segunda opción por eficiencia en cuanto a velocidad de ejecución y sencillez.

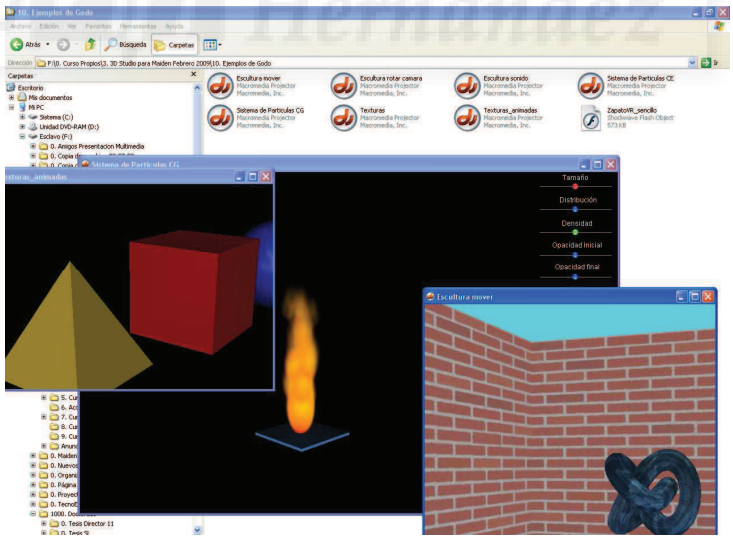


Imagen 5.2.8: Proyección autoejecutable de tres escenas interactivas 3D

Los proyectores crean archivos con la extensión .exe. Esto es una ventaja muy grande, ya que esta clase de fichero es autoejecutable, lo que quiere decir, que es compacto y siempre tendremos una visualización idéntica, sea cual sea el ordenador en el que se ejecuta, con lo que el espectador tiene el mismo resultado que el autor.

Caso práctico 2.1: Flujo de trabajo.

Objetivo: Conocer el flujo más simple posible de trabajo pero que a la vez cubra todas las etapas de creación de esta implementación: creación, construcción y publicación.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Realización de una obra 3D
2. Exportar a Shockwave 3D.
3. Construir la presentación del trabajo Shockwave 3D en Director.
4. Publicar en un proyector.

1. Realización de una obra 3D.

Como en este caso práctico queremos poner el punto de mira en el flujo de trabajo y no sobre la calidad de este, por ello, vamos a realizar un modelado muy sencillo, sin valor en sí mismo.

Para no complicar el modelado realizaremos una serie de modelos geométricos paramétricos, es decir, no utilizaremos un modelado complejo ni técnicas especiales. Vamos a utilizar las primitivas (objetos geométricos predefinidos) de MAX y lo único que modificaremos es su colocación y tamaño.

1.1. Ejecutar 3ds MAX.

1.2. Hacer clic sobre la vista Perspectiva. Esto sirve para definir la vista de trabajo.

Modelado:

1.3. Empezaremos por realizar la caja que sirve de base al pedestal, para ello ejecutamos Create > Geometry > Standard Primitives > Box. De momento, se trata de posicionarla y darle las medidas “a ojo”.

1.4. Para modificar el tamaño de la caja vamos a darle las medidas exactas. Utilizamos la entrada de datos por teclado. Para ello nos fijamos en la parte inferior de la ventana de creación. En la ficha parámetros modificamos Length, Width, Height todos a 100.

1.5. Para posicionar la caja con exactitud, pulsamos sobre el botón “Select and Move” con el botón derecho del ratón, aparecerá una caja de diálogo con la posición X, Y, Z donde podremos introducir los valores adecuados. En este caso los valores serán 0, 0, 0, lo que quiere decir que la caja se situará en el centro de la escena.

1.6. Repetir el proceso con el resto de modelos. Es decir, se trata de repetir la secuencia: crear, dimensionar y posicionar los objetos. La siguiente lista presenta la relación de los objetos de la escena:

Primitivas estándar:

Suelo	Box
Base del pedestal	Box
Pirámide del pedestal	Pyramid

Primitivas extendidas:

Esquina	L-ext.
Escultura	Torus Knot.

Todas estas primitivas se pueden generar en Crear > Geometría > 3D > Primitivas estándar o Primitivas extendidas. En este caso, no hace falta que tengamos un control total sobre las dimensiones de nuestros objetos 3D, basta que los hagamos por comparación de unos con otros.

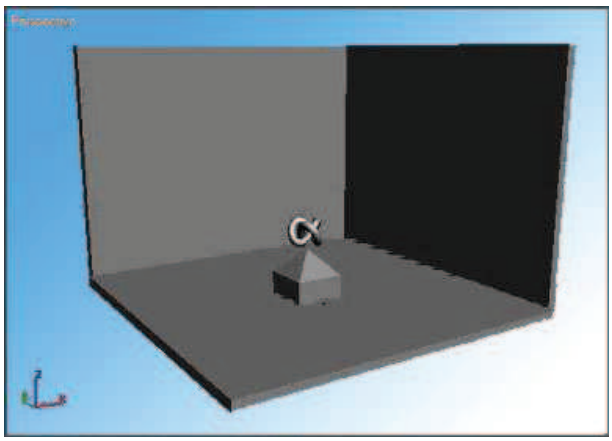


Imagen 5.2.cp.1.1: Modelado de la obra

1.7. Cambiar el nombre de los objetos en el panel Crear > Name and Color. Por ejemplo "box01" renombrarla por "pedestal" para poder identificarlo mejor en Director. Les daremos nombre a todos los elementos y para ello solo hace falta seleccionarlos y escribir el nombre deseado.

1.8. Debemos unir los dos elementos del pedestal: la caja y la pirámide. Esto lo realizaremos con una operación booleana. Para ello, hacemos Create > Compound objects > Boolean.

El resultado en la vista de perspectiva se muestra en la imagen anterior.

Materiales, Texturizado:

En este caso, vamos a texturizar los modelos utilizando los mapas que vienen con el propio programa 3ds MAX.

1.9. Pulsar la tecla "M" para abrir el "editor de materiales". Las texturas a aplicar son muy simples, por ello, no se realizan pasos previos que la mayoría de las veces son imprescindibles como son la generación de mapas.

1.10. Buscamos la ficha "Maps". Aparecerán muchas opciones, de las que elegimos "Diffuse Color" y pulsamos en el botón que la acompaña llamado "none". Aparecerá una ventana llamada "Material/Map Browser", donde elegiremos "Bitmap" haciendo doble clic. Esto nos llevará a la típica ventana de abrir fichero llamada "Select Bitmap Image File" y después de identificarlo pulsamos el botón "Abrir".

1.11. Para asignar el bitmap al objeto tenemos que asegurarnos que el objeto esté seleccionado y luego pulsar sobre un icono de los que se encuentran bajo los materiales llamado "Assign Material to Selection". No observaremos grandes cambios en la escena, pero el objeto ya estará mapeado.

1.12. Con los demás objetos haremos lo mismo salvo con la esquina y la escultura. A estos objetos además de añadirle un mapa en el canal difuso, le añadiremos uno en el canal relieve para que los ladrillos parezcan resaltar de la argamasa que los une. El proceso es similar salvo que en la ficha Maps usaremos el mapa tipo "Bump".

En la imagen 5.2.cp.1.3, se puede observar el resultado de texturizar la escena.

Iluminación:

1.13. Añadiremos una luz omni para iluminar un poco la escena. La creamos desde: Panel Crear > Luces > Omni y pulsamos sobre cualquier vista. Las luces omni aparecen como un romboide amarillo.

1.14. Posicionamos la luz cerca de la escultura y un poco por encima.

En la siguiente imagen se puede apreciar la posición donde hemos colocado la luz omni en nuestra escena.

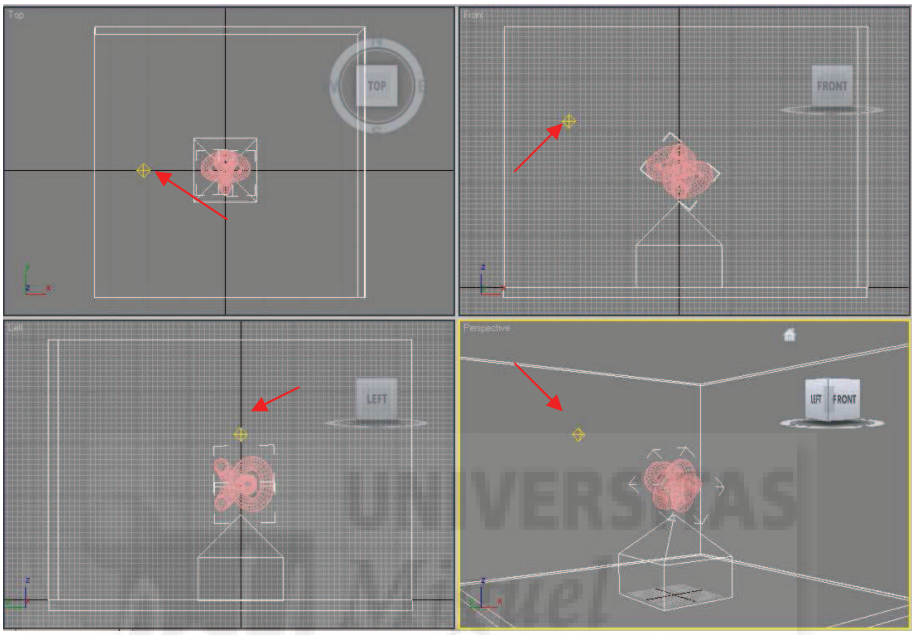


Imagen 5.2.cp.1.2: Luz en nuestra escena

Representación:

1.15. Hacemos clic sobre la vista que queremos representar. En nuestro caso hemos optado por la vista Perspectiva. Para ejecutar la representación ejecutamos F10 o Rendering > Render y en la ventana “Render Setup” pulsamos sobre el botón “Render” que se encuentra en la parte inferior.

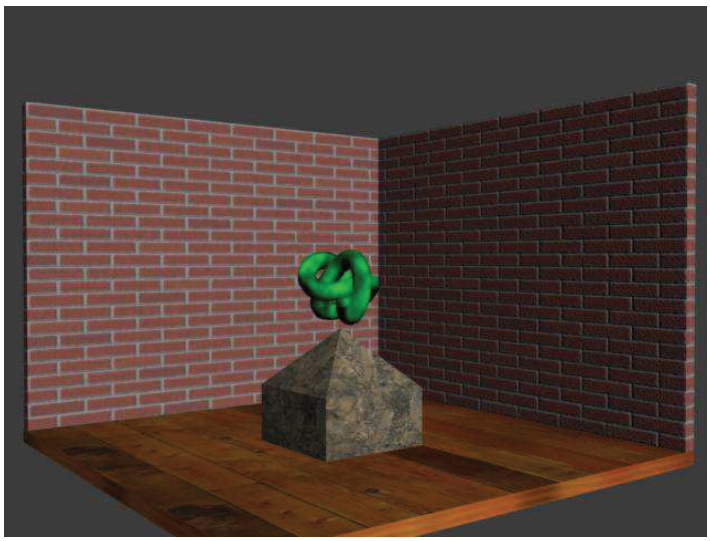


Imagen 5.2.cp.1.3: Representación final de la escena

2. Exportar a Shockwave 3D.

2.1 Ejecutar el comando File > Export.

2.2. Aparecerá la ventana “Select File to Export”. Elegimos en el tipo de fichero la opción Shockwave 3D Scene Export (.w3d). Le ponemos el nombre “escultura” y pulsamos el botón “Save”.

2.3. Aparecerá la ventana llamada “Shockwave 3D Scene Options”. Pulsamos el botón “Preview” para ver como quedará la escena.

2.4. Pulsamos el botón “Export” y tendremos como resultado el archivo llamado “escultura.w3d”. Este archivo es el abriremos desde Director.

3. Construir la presentación del trabajo Shockwave 3D en Director.

3.1. Abrimos Director® 11.

3.2. Ejecutamos File > Import y navegando hasta donde esté, seleccionamos “escultura.w3d” y pulsamos el botón “Import”. Al principio no veremos que pase nada. Esto es porque el fichero importado pasa a ser un miembro de nuestra “película” de Director.

3.3. Debemos de visualizar el “contenedor” de nuestra aplicación, esto lo hacemos ejecutando el comando Windows > Stage. El “stage” es el escenario de nuestra presentación.

3.4. Si el escenario o stage no nos satisface, lo podremos modificar con Modify > Movie > Properties o Ctrl + Shif +D. El tamaño se modifica con las propiedades size: Movie > Stage Size.

3.5. Ejecutamos Windows > Score. En la parte inferior veremos nuestra escena importada llamada escultura y seleccionándola la arrastramos hasta la ventana Stage. Podemos observar en la ventana Score la aparición (Sprite) del miembro “escultura”.

3.6. Estiramos el sprite de “escultura.w3d” para que ocupe todo el espacio del stage disponible.

3.7. Pulsamos el botón “Play” para ver cómo queda la exposición de nuestro trabajo.

3.8. Para visualizar la obra interactivamente en 3 dimensiones usamos el comando Windows > Shockwave 3D. Aparecerá una pequeña ventana integrada en Director que muestra la escena y que permite “navegarla”.

3.9. Guardamos el fichero de Director con File > Save as y elegimos un nombre, por ejemplo, escultura. El fichero tendrá extensión .dir, por lo que quedará como “escultura.dir”.

4. Publicar en un proyector.

4.1. Ejecutamos File > Publish Settings. Aparece la ventana “Publish Settings”. Aquí tenemos múltiples opciones de publicación y cada una de ellas con opciones propias que ya estudiaremos más detenidamente en los casos prácticos correspondientes.

4.2. Elegimos solamente la opción “Projector”.

4.3. Accedemos a la ficha Projector y pulsamos “Player type”. Elegimos Shockwave.

4.4. Solo nos resta pulsar sobre el botón “Publish”. Con lo que se crea un fichero .exe. Este tipo de fichero es autoejecutable desde cualquier sistema Windows, lo que hace que sea portátil y fiable. Será el archivo que podemos introducir en un CD / DVD para distribuirlo.

4.5. Para comprobar si todo el trabajo a merecido la pena, tenemos que ejecutarlo. Para ello abrimos Mi PC, navegamos hasta encontrar el archivo y hacemos doble clic sobre él.

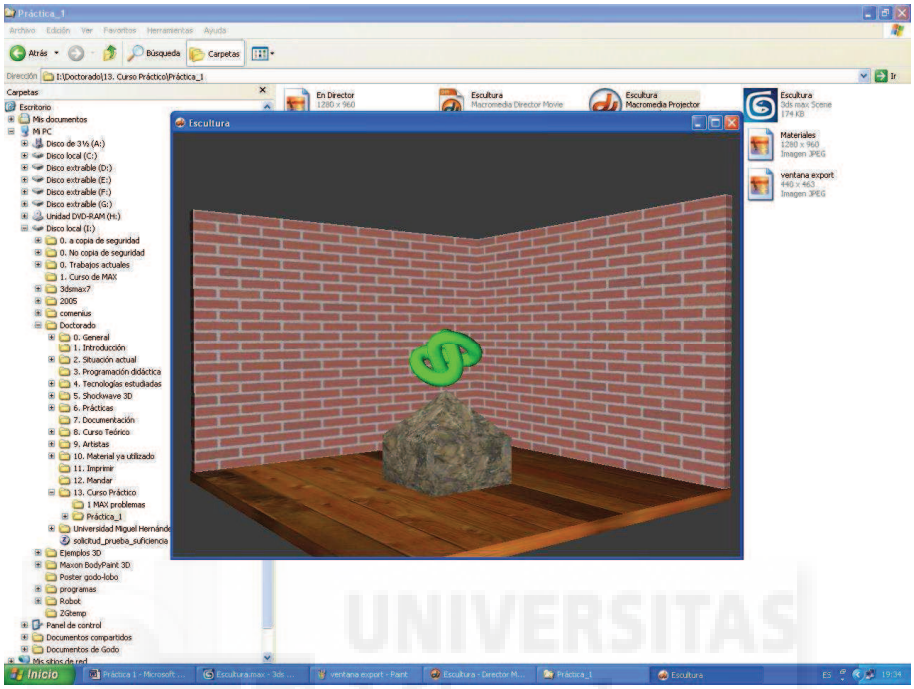


Imagen 5.2.cp.1.4: Ejecución desde el explorador de Windows

Conclusiones:

Podemos decir que el flujo de trabajo que necesitamos para esta implementación es largo pero sencillo.

Es largo porque tenemos que generar al menos 3 pasos: modelado, aplicación de materiales e iluminación, y además, en un programa complejo como es MAX. Luego debemos de exportarlo para obtener un fichero .w3d y por último, abrir director para importar el archivo y presentarlo. Sin embargo, también es un flujo de trabajo sencillo porque las “etapas” están muy bien definidas y la secuencialidad es muy clara.

Por otra parte, estas “etapas” se van a repetir siempre para los demás casos, ya que hemos realizado una práctica mínima, por lo que la cantidad de pasos puede aumentar pero no disminuir.

Unidad 3: Modelado en MAX

Introducción teórica:

1. Introducción al modelado en MAX.
2. Modelado de objetos básicos.
3. Transformaciones básicas de modelos.
4. Modelado 3D a partir de formas 2D.
5. Modificadores de geometría.
6. Operaciones booleanas.
7. Organización del modelo.
8. Modelado poligonal.
9. Modelado de curvas y superficies: Nurbs.
10. Modelado de efectos naturales.

Casos prácticos:

3.1. Modelado con primitivas y técnicas básicas.

3.2. Modelado de extrusiones.

3.3. Modelado de revoluciones.

3.4. Modelado de solevaciones.

3.5. Modelado poligonal.

1. Introducción al modelado en MAX.

3D Studio MAX es un programa de modelado 3D por frontera, es decir, los modelos no están "rellenos", el trabajo de modelado se traduce en la creación de unos polígonos que forman caras que será lo que finalmente se representará del modelo.

MAX es un programa generalista, esto quiere decir que es capaz de crear mundos virtuales de muchos tipos y siguiendo muchas técnicas distintas. Se utiliza para crear paisajes, arquitectura, personajes de películas, videojuegos, efectos especiales, etc.

Con MAX podemos usar las siguientes técnicas de modelado:

- Modelado con primitivas.
- Modelado con modificadores.
- Modelado con composición de modelos.
    - Modelado 3D a partir de formas 2D.
    - Operaciones booleanas de objetos 3D.
    - Modelado por organizaciones de modelos 3D.
- Mallas poligonales editables.
- Modelado de superficies Nurbs.
- Modelado con cuadrículas de corrección.
- Modelado de Sistemas de partículas.
- Modelado por programación.

Incluso, no nombramos alguna técnica de las menos habituales, es decir, todas las incluidas en la clasificación de los métodos estudiados en la parte teórica de este curso. Esto, que en principio es una ventaja, se puede volver en contra sino tenemos claro el método a utilizar para modelar un objeto en particular.

Para ayudarnos en la edición de los objetos tenemos múltiples herramientas. Pasamos a comentar las más utilizadas.

- **Manejo de vistas**

Entre estas herramientas tenemos zoom, zoom a todas las vistas, zoom para objeto seleccionado, zoom definido por ventan, encuadre, vista orbital y modo de una vista. Todas la posibilidades de pueden observar en la siguiente imagen.

Imagen 5.3.1: Herramientas para manejo de las vistas

- **Herramientas de selección**

Para seleccionar un objeto o parte de él podemos usar las herramientas que se observan en la siguiente imagen. Además MAX posee otros modos avanzados de selección.

Imagen 5.3.2: Herramientas para seleccionar objetos

La primera herramienta empezando por la izquierda del grupo se llama “Selección”. Haciendo clic sobre un objeto podemos seleccionarlo directamente.

La segunda es “Selección por nombre”. Aparece una caja de diálogo para poder seleccionar uno o varios objetos por su nombre que evidentemente tendremos que haber introducido anteriormente.

La tercera herramienta se llama “Región de selección rectangular” y tiene un triangulito debajo que significa que podemos tener otras opciones, como región circular, por lazo, etc. Esta herramienta está ligada a la última que es “Interior / Tocar”, que hace que la ventana que se utiliza para seleccionar tenga tan solo que tocar en parte el objeto a seleccionar o tenga que incluirlo totalmente. Por ejemplo, arriba está activa en la opción de “tocar”.

- **Transformación básica de objetos**

Las herramientas para modificar la posición, rotación y escalado de los modelos las estudiaremos en un apartado propio y las podemos observar en la siguiente imagen.

Imagen 5.3.3: Herramientas para la modificación básica de objetos

- **Simetría y alineamiento**

La herramienta simetría permite crear modelos simétricos a los que ya tenemos en escena y la herramienta de alinear permite posicionar un objeto en función de otro, es muy útil para centrar objetos, emparejarlos, etc. Quizá en este apartado podríamos incluir la utilidad llamada matriz, que permite repetir un objeto muchas veces y en los ejes que deseemos.

Imagen 5.3.4: Herramientas de simetría y alineación

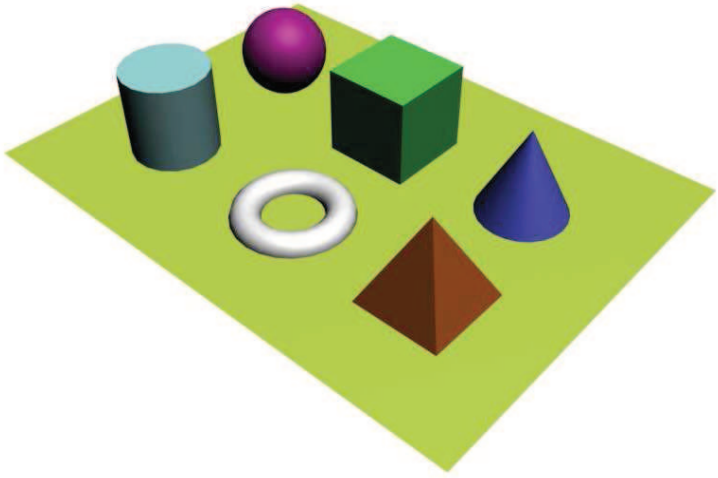


Imagen 5.3.5: Primitivas estándar de MAX

2. Modelado de objetos básicos.

Ya conocemos el concepto de primitivas, que son los objetos más básicos que se pueden crear directamente. 3D Studio MAX contiene 22 primitivas. En la imagen anterior se pueden apreciar unas primitivas realizadas con MAX.

En la imagen inferior se puede apreciar el panel "Crear" de MAX que en este caso está actualizado a objetos 3D. Este panel como su nombre indica, permite crear objetos para la escena 3D. A la derecha aparecen los paneles de los objetos 2D, las luces, las cámaras, los ayudantes, los modificadores de espacio y los sistemas. En este caso, se aprecia que las primitivas que estamos viendo son las estándar donde tenemos 10 de ellas.

Imagen 5.3.6: Primitivas estándar: Caja, Cono, Esfera, y otras

Si quisiéramos crear una esfera, solo tendríamos que elegir el botón "Sphere" y hacer clic en una vista para determinar su centro y arrastrar y soltar para determinar su radio. Luego, en el control "Name and Color" deberíamos escribir un nombre y un color base para el objeto. Para precisar mejor las dimensiones de las primitivas, tenemos un grupo de parámetros que permiten su introducción manual. Si queremos modificar estos valores después de haber creado la primitiva, basta con seleccionarla y acceder al panel "Modify".

En la imagen que sigue se puede observar los tipos de objetos que podemos crear, entre los cuales se encuentran primitivas extendidas, sistemas de partículas, nurbs, puertas y ventanas, objetos dinámicos, escaleras, etc.

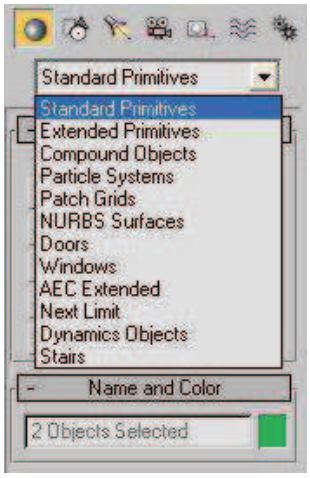


Imagen 5.3.7: Categorías del panel para crear objetos

3. Transformaciones básicas de modelos.

Estas transformaciones son tres: translación, giro y escalado. Estas son fundamentales y universales en todo software que trabaje en 3D. Todos los objetos de escena como cámaras y luces se pueden mover y girar, pero solo los modelos se pueden escalar directamente.

En la imagen 5.3.3 se puede observar los iconos que representan a las tres herramientas en la barra estándar de MAX. La forma de llevar a cabo estas transformaciones puede ser de dos maneras: por valores o interactivamente, o sea, “a ojo”. Por ejemplo, para mover un modelo a mano, basta con seleccionarlo y pulsar sobre la herramienta mover y luego arrastrar el cursor directamente sobre el visor o sobre un determinado eje para limitar su traslación a ese eje. Este método se puede realizar también en las rotaciones y los escalados.

El método más preciso se obtiene al seleccionar el modelo elegido y pulsar con el botón derecho del ratón sobre la herramienta mover. Esto produce que aparezca una caja de diálogo donde encontraremos dos grupos: universal y Offset.

El grupo universal presenta tres controles para modificar su posición universal y nos permite modificarlos numéricamente y por tanto con total exactitud.

También podemos introducir aquí tres valores para cada uno de los ejes en el grupo Offset. Estos valores moverán el objeto pero no según unas coordenadas universales, sino según las coordenadas actuales del objeto. En otras palabras, si queremos mover un objeto con respecto a su posición actual utilizaremos estos controles.

En la imagen inferior se puede apreciar la caja de diálogo que exhibe la herramienta mover cuando se pulsa botón derecho sobre ella. La herramienta de rotación y la de escalado presentan cajas de diálogo parecidas.

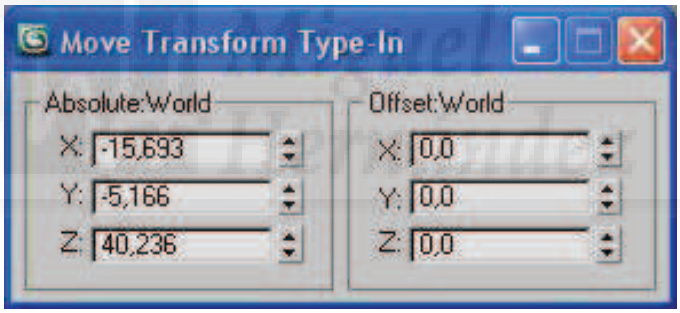


Imagen 5.3.8: Transformación de posición por valor

Un ejemplo de la utilización de cambio de posición y escalado lo podemos ver en la imagen inferior, donde podemos observar la edición de un modelo muy simple de árbol y cómo se han transformado para conseguir unidades distintas del original.

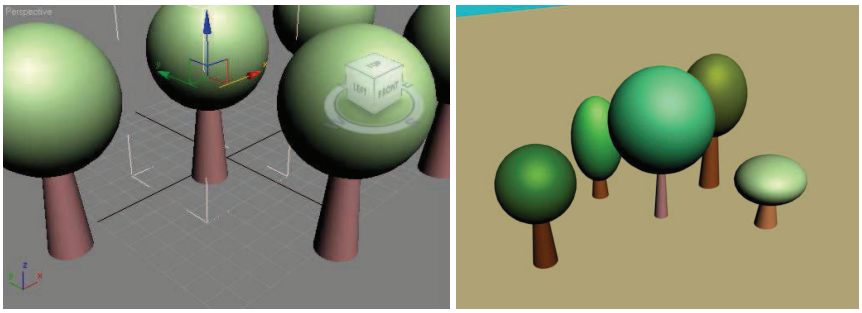


Imagen 5.3.9: Transformaciones en escalado y posición

4. Modelado 3D a partir de formas 2D.

En este punto vamos a ver tres métodos fundamentales de modelado en 3D Studio MAX: extrusión, revolución y solevación. Estos métodos permiten recrear muchos modelos reales creados por el hombre, ya que son métodos que se utilizan en máquinas de fabricación de distintos objetos en el mundo real. En la imagen inferior se pueden observar los resultados de los casos prácticos adjuntos a este tema dónde se han aplicado de izquierda a derecha, una extrusión, una revolución y una solevación.

Imagen 5.3.10: Ejemplos de modelos 3D generados a partir de formas 2D

Lo común a los tres métodos mencionados, es que parten de figuras 2D. En MAX, para realizar tales figuras debemos acceder al panel Create > Shapes que se puede observar en la imagen inferior.

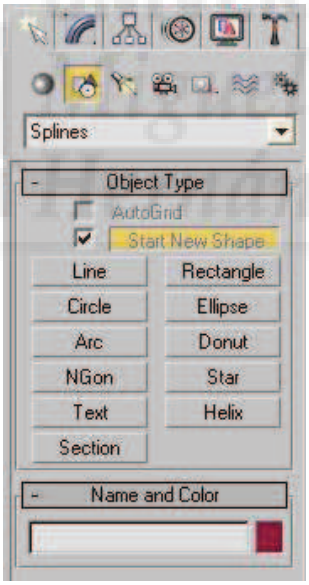


Imagen 5.3.11: Panel Shapes para formas 2D

Las formas o Shapes están clasificadas en tres grupos: splines, curvas nurbs y splines extendidas. Como podemos ver existen 11 formas de tipo “Splines”, incluyendo líneas, círculos, textos, polígonos, etc.

- **Extrusión**

Estos modelos se obtienen al dar profundidad a formas en 2 dimensiones. Por ejemplo, si tenemos un cuadrado (2D) y le añadimos cierta profundidad obtendremos una primitiva caja (3D). Una vez creada la forma base, crearemos la extrusión accediendo al panel “Modify” y a la lista de modificadores, donde elegiremos “Extrude”. Entonces aparecerán los controles para

modificar los parámetros de extrude y en “Amount” (cantidad) introduciremos la longitud de la profundidad que queramos. En la imagen inferior podemos observar el resultado del caso práctico adjunto a este punto.

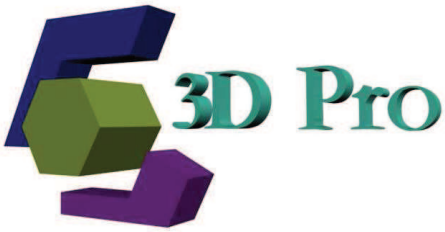


Imagen 5.3.12: Logo realizado mediante extrusiones

- **Revolución**

Los objetos modelados por el método de revolución parten de una figura 2D a la que se aplica al igual que en el caso de la extrusión, un modificador, pero en este caso se llama “Lathe”. Es muy normal que el eje de revolución sea el eje vertical y entre sus parámetros se usan muy frecuentemente “Weld Core” que produce un cierre de las áreas en contacto y “Flip normal” para definir las normales, es decir, la parte visible de los polígonos. En la imagen inferior se puede observar que el perfil (a la izquierda) es una forma abierta y por ello, en este caso debemos de utilizar “Weld Core” sino, la base de la copa que tenemos como resultado mostraría un hueco.

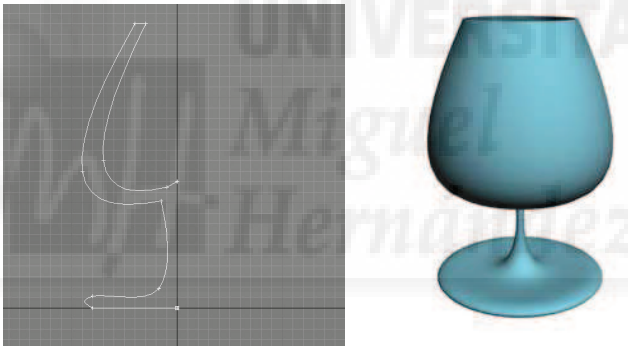


Imagen 5.3.13: Perfil y revolución obtenida sobre el eje vertical

- **Solevados**

En Max podemos crear objetos solevados. Estos objetos se crean a partir de dos formas 2D: el perfil (Shape) y la trayectoria (Path). El perfil debe ser una forma cerrada como un cuadrado, una estrella y círculo, etc. La trayectoria será una forma que puede ser abierta o cerrada como una línea recta o curva, un círculo, etc.

Imagen 5.3.14: Modelado por solevación

La solevación producirá que el perfil siga la trayectoria desde su principio hasta su fin creando un elemento 3D. Por ejemplo, si queremos hacer un manguito, podemos crear un perfil con un pequeño círculo y la trayectoria mediante una forma spline en 3D.

Para crear modelos solevados no lo hacemos mediante un modificador como en los casos anteriores. Debemos tener uno de los dos elementos 2D seleccionado, supongamos que es la trayectoria, entonces accedemos al panel Create >Geometry > Compound Objects > Loft. En este momento aparece el panel “Creation method” y pulsamos sobre “Get Shape” para al fin, seleccionar el perfil con lo que el modelo solevado queda creado.

De todas formas, el verdadero potencial del método de solevación es la posibilidad de aplicar deformaciones de forma gráfica. Para poder editar estas características seleccionamos el modelo solevado y accedemos mediante el panel Modify a “Deformations” donde tenemos las posibilidades que se observan en la imagen inferior.

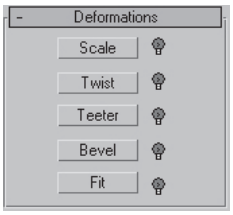


Imagen 5.3.15: Deformaciones posibles de modelos solevados

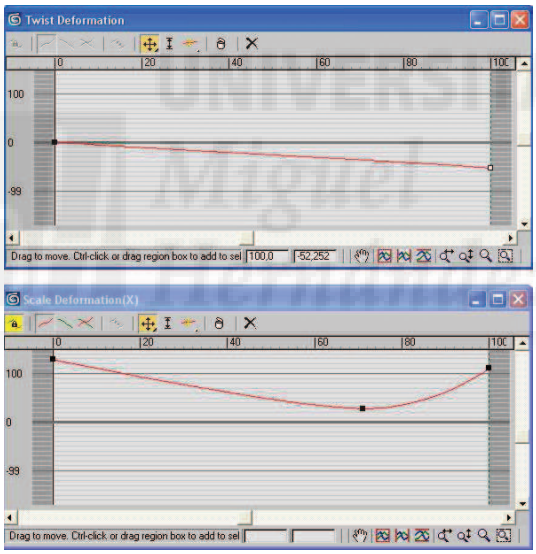


Imagen 5.3.16: Deformaciones Twist y Scale aplicadas a una caja

Estas deformaciones permiten hacer por ejemplo que el perfil vaya girando sobre sí mismo o escalándose mientras recorre la trayectoria. Para editar por ejemplo “Scale” pulsaremos el botón y aparecerá una gráfica donde podemos editar puntos que generarán las deformaciones en el modelo 3D. En la imagen que sigue se observa las gráficas de las deformaciones que producen el modelo presentado en la imagen 5.3.15.

5. Modificadores de geometría.

Podemos decir que ya hemos estudiado algunos modificadores de geometría ya que tanto Extrude como Lathe son modificadores que crean geometría, pero en este punto vamos a centrarnos en aquellos que modifican directamente una geometría 3D en otra 3D no pasando de una en 2D a otra 3D como en los dos casos expuestos.

Por ejemplo, en la imagen inferior se pueden observar tres columnas diseñadas mediante modificadores de geometría aplicados tanto a la base, como al fuste y al capitel de la primera de ellas, la situada más a la izquierda.

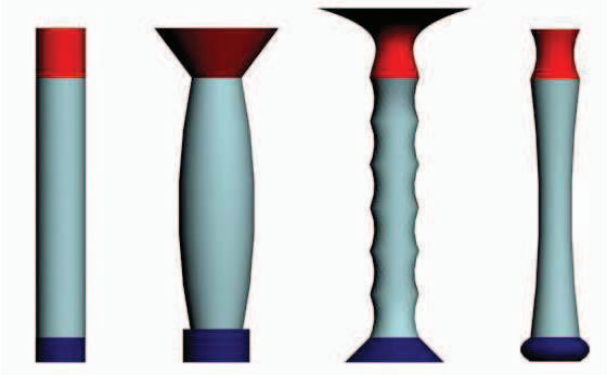


Imagen 5.3.17: Aplicación de modificadores de geometría 3D

El método para aplicar los modificadores de geometría es seleccionar la geometría inicial y acceder al panel Modify y desde aquí a la "Lista de modificadores", donde elegiremos el que queramos. Una vez elegido aparecerá abajo una serie de parámetros para poner valores a cada una de las características que definen el modificador elegido. Cada modificador tendrá distintos parámetros ya que modificarán la geometría de una forma distinta.

En la imagen anterior, la diferencia entre los capiteles de las dos primeras columnas (empezando por la izquierda) se basa en el modificador Taper, que "afila" una geometría. El fuste de la tercera columna se creó mediante el modificador Twist que retuerce un determinado modelo. Existen modificadores como el caso de FFD que presentan una malla que rodea al modelo y que se puede manipular moviendo los vértices que la definen. Esta deformación en la malla se traduce en una deformación del mismo tipo en el modelo, con lo que podemos cambiarlo a nuestro antojo y con una precisión que dependerá del número de vértices de la malla FFD.

En la siguiente imagen se puede apreciar el efecto de aplicar algunos de estos modificadores al mismo modelo, en este caso a copias del cilindro morado de la izquierda que son explicados más abajo en el mismo orden, o sea, de izquierda a derecha.

Imagen 5.3.18: Algunos modificadores de geometría de MAX

Bend: tuerce. Dobla en distintos ejes la geometría del modelo, siempre y cuando tenga segmentos suficientes en el eje afectado para doblar el modelo.

FFD: malla que circunscribe al modelo y cuyos vértices podemos reposicionar para modificar la forma del modelo. En el ejemplo, se han escalado los vértices centrales de un modificador ffd de malla 3x3x3.

Noise: ruido. Permite generar alteraciones aleatorias en los ejes y con la fuerza que deseemos. Normalmente se utiliza para crear modelos geométricos más realistas a partir de primitivas, como pueden ser troncos y ramas a partir de cilindros o piedras a partir de esferas.

Skew: sesgo. Produce una inclinación en un eje determinado.

Taper: afila un modelo en una determinada dirección.

Twist: retuerce la geometría sobre sí misma sobre un eje.

Existen muchos más modificadores de geometría, pero creemos que estos pueden ser una buena representación de cómo trabajan y las posibilidades de modelado que presentan.

No vamos a plantear un caso práctico que ilustre la forma de aplicar los modificadores de geometría debido a que sigue el mismo método que el de otros que ya hemos estudiado como las extrusiones que se encuentran explicadas en el caso práctico 3.2.

6. Operaciones booleanas

El modelado de objetos 3D mediante operaciones booleanas es muy intuitivo, ya que se basa en la teoría de conjuntos que hemos estudiado todos en matemáticas básicas con las operaciones de unión, intersección y substracción.

En la siguiente imagen se puede observar arriba a la izquierda los modelos de los que se parte: una caja roja y una esfera verde independientes entre sí. Arriba a la derecha se puede observar el resultado de la operación de unión de la caja con la esfera. No es lo mismo seleccionar primero la caja y ejecutar la operación con la esfera que al revés. Por eso, el resultado presenta un color rojo, si lo hubiéramos realizado al revés el color del modelo obtenido sería verde. En la operación de unión el orden no es muy importante pero en las demás operaciones si. En la esquina inferior izquierda de la imagen se observa media esfera roja, resultado de la operación booleanas de intersección entre la esfera y la caja, es decir, de los modelos que ocupaban un volumen conjunto. Abajo a la derecha se observa la caja y en su superficie un agujero dejado al realizar la operación de substracción o resta entre la caja y la esfera. Al ser referenciada primero la caja y luego la esfera se resta la esfera de la caja quedando el resultado que se observa. Pero esta operación habría dado un resultado distinto si se toma la esfera y luego se aplica la operación con la caja como segundo operando. Entonces el resultado sería la esfera menos la caja, es decir, tendríamos una semiesfera verde: la parte superior que no intersecta con la caja.

Por lo tanto, a efectos de modelaje las operaciones de unión e intersección son conmutativas, pero la de substracción no. A efectos de herencia de otras características como el color, las texturas, etc. ninguna de las operaciones es conmutativa ya que siempre prevalecen las características del primer operador.

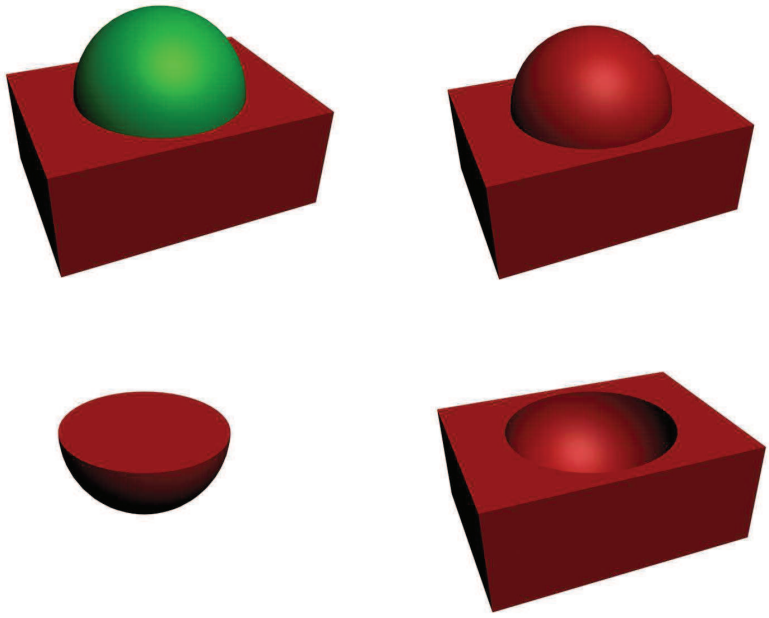


Imagen 5.3.19: Modelos con las tres operaciones booleanas aplicadas

El método para realizar las operaciones sería: primero seleccionamos el primer modelo (supongamos que es la caja), en segundo lugar accedemos al panel Create > Geometry > Compound object > Boolean, entonces veremos en MAX los parámetros para realizar las operaciones, tal como se observa en la siguiente imagen.

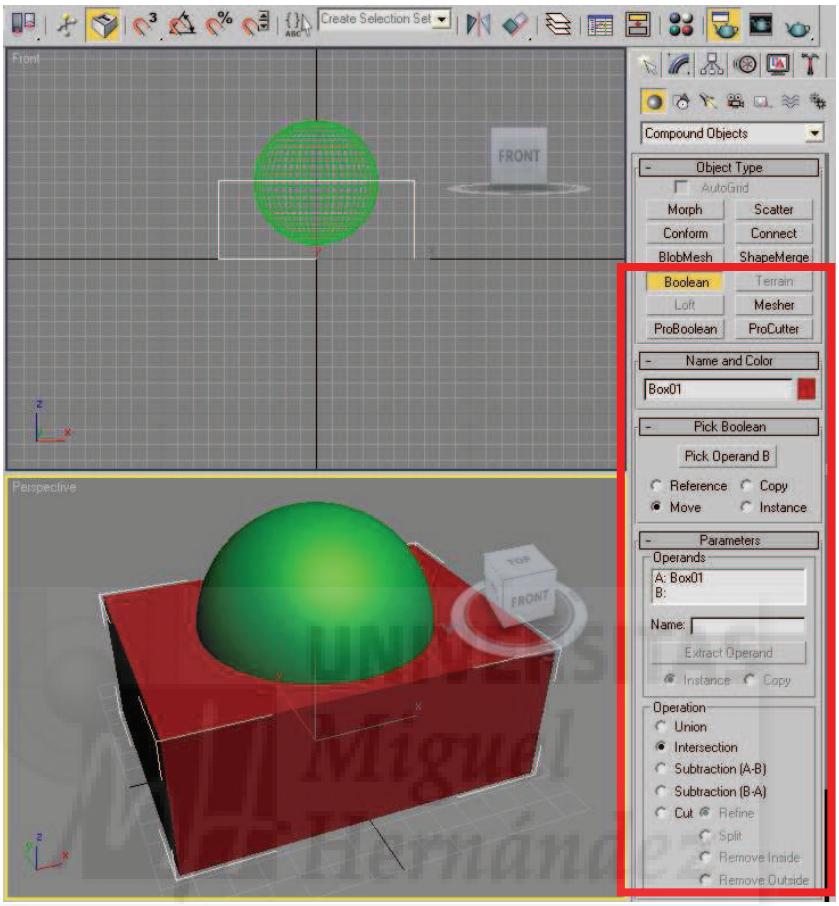


Imagen 5.3.20: Interfaz que presenta MAX para realizar modelos booleanos

En tercer lugar pulsaríamos en “Operation” (abajo del todo) la operación deseada, por ejemplo, “Union” y por último pulsaríamos sobre el botón “Pick Operand B” (más arriba) y hacemos clic sobre el segundo modelo operador (supongamos que la esfera verde).

Los modelos booleanos se utilizan masivamente ya que es el método constructivista por excelencia, adaptándose a métodos de fabricación de objetos en la vida real como puede ser la soldadura o la creación de moldes. También podemos aplicarlos a la arquitectura, por ejemplo, hacemos un muro con una caja y abrimos un hueco para una ventana mediante una operación booleana de substracción.

7. Organización del modelo.

En este punto vamos a estudiar las posibilidades que presenta MAX para que varios modelos puedan actuar conjuntamente mediante distintas técnicas.

- **Grupos**

La primera opción para editar conjuntamente una serie de modelos es agruparlos. Esta operación se puede realizar seleccionando los modelos y luego ejecutando la orden del menú Group > Group. Entonces MAX presentará una caja de diálogo para que introduzcamos el

nombre del grupo. Las agrupaciones se pueden deshacer desagrupando los objetos que lo componen para ser modificados por separado.

Cuando tenemos definido un grupo podemos trabajar con él como si de un solo objeto se tratase y por tanto le podemos aplicar transformaciones como mover, giro y escalado a todo el grupo a la vez.

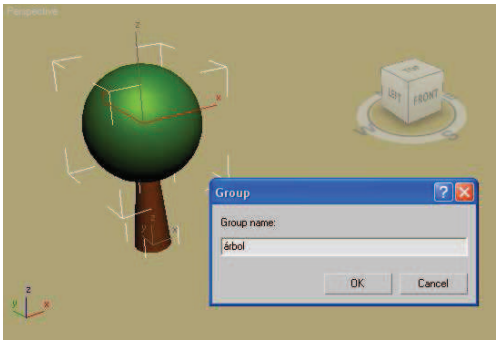


Imagen 5.3.21: Creación de un grupo

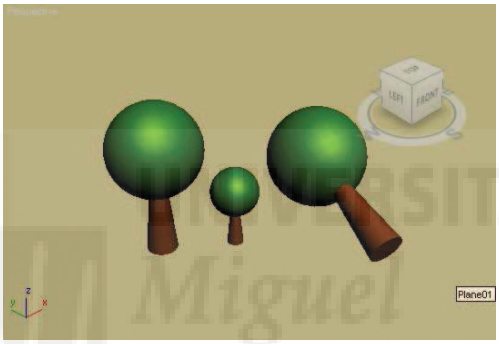


Imagen 5.3.22: Edición con grupos

En las dos imágenes de arriba podemos observar cómo hemos creado un grupo a partir de un cono y una esfera y lo nombramos como “árbol”, luego podemos modificar el grupo haciendo copias, rotándolo, escalándolo, etc. A diferencia de la operación booleana unión, esta operación no modifica las propiedades individuales de los componentes del grupo y además se pueden aplicar otras órdenes como abrir/cerrar o attach/detach para gestionar la pertenencia de distintos modelos u otros grupos a un grupo.

- **Composiciones y enlaces jerárquicos**

Otro modo de organizar un modelo es utilizar los enlaces jerárquicos, esto supone que un modelo tendrá un enlace con otro modelo y esto se puede ir repitiendo en una sucesión de enlaces direccionales. Esto quiere decir que el primer modelo enlaza con el segundo y no al revés. Se conocen como enlaces padre-hijo. Pongamos un ejemplo: si queremos construir un brazo robótico, crearemos cada uno de los modelos que la componen, pero para realizar una animación, podemos hacer que el hombro tenga un enlace con el antebrazo, este con el codo, el codo con el brazo, este con la muñeca y así sucesivamente. De tal manera que podemos decir que el hombro es el objeto padre del antebrazo y por extensión, de todos los demás componentes.

Cuando queremos construir una animación de nuestro brazo robótico, al girar el hombro, giraremos por extensión todos los demás modelos, pero no al revés. Esto se traduce en que si giramos la muñeca, giraremos también la mano y los dedos, pero si giramos un dedo, no giramos ni la muñeca, ni por supuesto, el hombro.

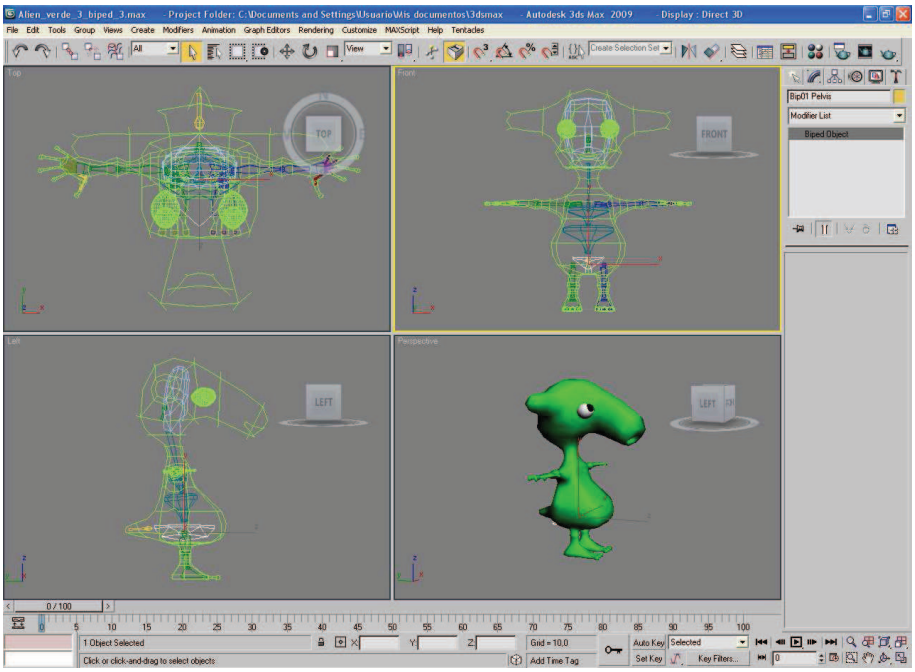


Imagen 5.3.23: Biped y su aplicación a un personaje con "Character Studio"

En MAX, existen muchas formas de crear estos enlaces cuya mayor aplicación es la animación. Se pueden establecer enlaces y dotarles de cinemática directa o inversa. La cinemática directa significa que en el caso de nuestro ejemplo, podemos mover cada uno de los componentes por separado. La cinemática inversa supone que al mover solamente el dedo hasta una posición determinada, MAX calcularía la posición de los componentes padres para ir posicionándolos de tal manera que quedasen bien colocados desde un punto de vista fisiológico. Esto supone definir los grados de libertad de cada uno de los modelos que describen articulaciones y por tanto supone un esfuerzo previo muy considerable. La ventaja es que en el trabajo de animación, el tiempo se reduce considerablemente.

MAX posee desde hace algunas versiones la posibilidad de utilizar una estructura organizativa de cinemática inversa concreta: el esqueleto bípedo. Quizá es la más utilizada de todas y por ello, MAX integra una solución para crear y animar personajes bípedos llamada "Character Studio" que está dividida en dos partes: Biped y Phisique. "Biped" crea un esqueleto bípedo que podemos personalizar con cuernos, rabo, etc., "Phisique" permite unir Biped a una malla y por tantos sería como unir un esqueleto a unos músculos, generando así un personaje. Además para nuestros proyectos, es una buena solución, ya que "Character Studio" y sus animaciones asociadas las podemos exportar y usar en Director y gestionar con Lingo. En la imagen anterior se puede visualizar la creación de un esqueleto con Biped y su aplicación al cuerpo de un personaje que es una especie de extraterrestre mediante "Character Studio".

También existen otros como "Bones" (una utilidad para crear huesos directamente) y otros que se pueden adquirir como módulos independientes. En la imagen inferior se puede observar la creación de un "esqueleto" de solo dos huesos en gris claro y a su alrededor una malla cilíndrica que representa el cuerpo. Al animar estos huesos, animan también la malla asociada y así podemos hacer que el cilindro se mueva como el codo de un brazo. Esta sería la solución para crear personajes que no fueran bípedos, como serpientes, caballos,… o estructuras como una grúa de pluma.

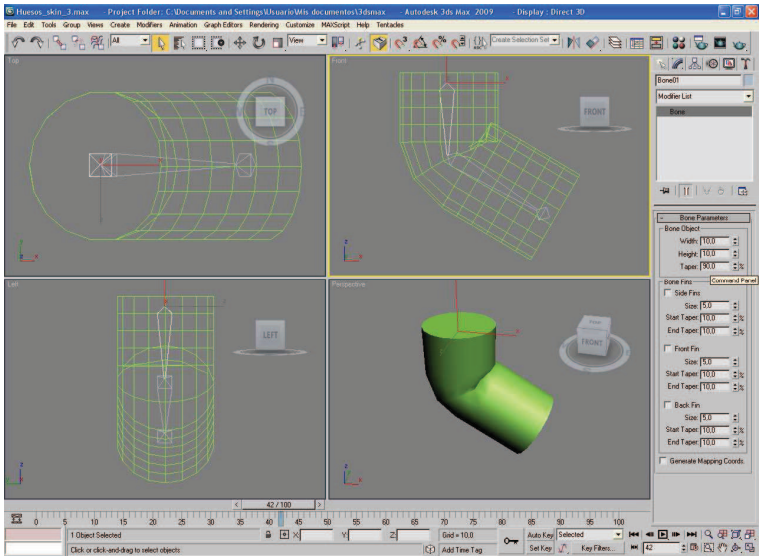


Imagen 5.3.24: Bones y su aplicación a un cuerpo

De todas formas, al ser esta una cuestión donde la aplicación es la animación y no el modelado, se encuentra fuera del objetivo de esta unidad.

◆ **Otros tipos de organizaciones de modelos**

Existen otras formas de organizar modelos, como pueden ser la creación de copias de un modelo base en matrices o realizar una o una serie de copias, referencias o instancias. Por ejemplo, si estamos creando una mesa y hemos realizado una pata, y necesitamos 3 más, podemos crear tres copias y cada una de ellas podrá ser colocada y girada dónde queramos.

Pero existen otras alternativas como hacer matrices, instancias y otras. De todas formas no los consideramos un tipo de organización de modelo complejo como para dedicarle más espacio en esta unidad.

8. Modelado poligonal.

El resultado del modelado poligonal es una malla poligonal. La malla poligonal es la 23ª primitiva de MAX, pero esta primitiva no se construye de forma paramétrica, es decir, no se puede introducir unos valores y MAX construye el modelo 3D como sí ocurre con la esfera, la caja y demás, sino que se debe crear cara a cara.

Este método se usa desde hace mucho tiempo, aunque sigue estando muy en boga, sobre todo desde que existe la tecnología para suavizar mallas y hacerla más “realista”. Esto hace que se pueda emplear tanto en modelos con aristas muy pronunciadas como un avión de combate o para modelos con aristas suavizadas como puede ser un rostro humano, un pez o cualquier modelo muy aerodinámico.

El resultado obtenido es una serie de triángulos que crean caras y estas polígonos y estos a su vez, el modelo 3D. Lo bueno del modelado poligonal es que es manipulable por niveles, es decir, si queremos modificar un vértice en particular, lo podemos hacer, o editar una cara entera o un polígono, por lo tanto, depende de la paciencia del modelador el conseguir un modelo más o menos detallado.

En la imagen inferior se puede apreciar una flecha que señala el comando que convierte cualquier primitiva en un “editable Poly”. Para que aparezca este menú contextual basta con pulsar botón derecho sobre el modelo 3D.

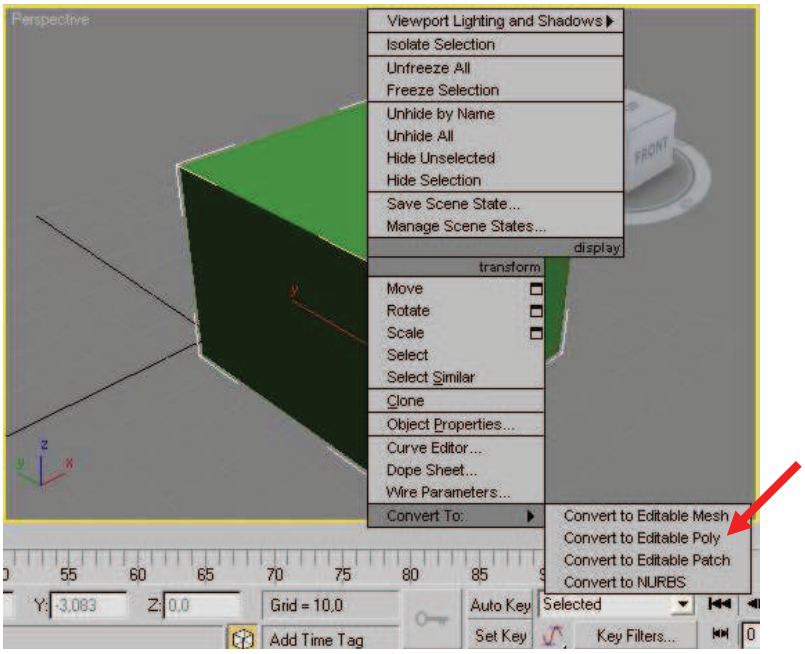


Imagen 5.3.25: Convertir una primitiva en un "editable Poly"

MAX aporta una completa gama de herramientas de modelado especialmente diseñadas para el trabajo poligonal. Antes de saber utilizar una herramienta tenemos que decidir sobre qué aplicarla, es decir, si el objeto que se verá transformado es un vértice, una arista, una triángulo, una cara o el modelo entero. A cada uno de estas partes se le llama sub-objeto, y el poder seleccionarlo y transformarlo es la base de este tipo de modelado.

En la imagen que sigue se puede observar que vamos a realizar una extrusión en un polígono determinado de un "editable Poly". A la derecha se visualiza el menú "Selection" dónde podemos elegir con qué tipo de entidad (vértice, arista, cara, polígono o todo) queremos trabajar, y en función de esta elección, MAX presentará las herramientas correspondientes.

El método utilizado es crear un modelo 3D que sirva de base, por ejemplo, una caja y luego convertirla a malla poligonal. Existen dos opciones aunque en la actualidad se usa más la opción del "polígono editable" que el de la "malla editable". En MAX todas las primitivas se pueden pasar a "polígono editable". Esto hace que dejen de ser paramétricos. Sin duda, la primitiva base más utilizada es la caja, lo que ha dado un submétodo propio de modelado denominado "Box modeling" muy utilizado para realizar personajes de videojuegos debido a algunas de sus características como rapidez, sencillez y un resultado aceptable con pocos polígonos, que es algo imprescindible para los sistemas que deben actuar en tiempo real.

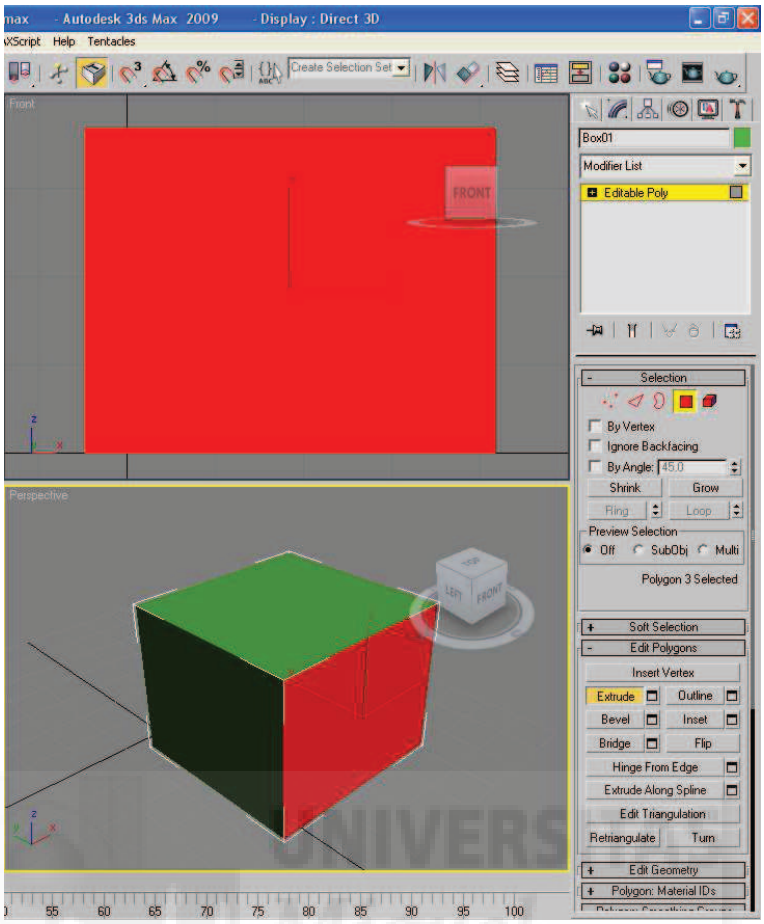


Imagen 5.3.26: Extrusión de un polígono de un "editable Poly"

En la imagen a continuación podemos ver en primer lugar el proceso que sigue la caja convertida en una malla editable hasta convertirse en "algo" parecido a un personaje bípedo.

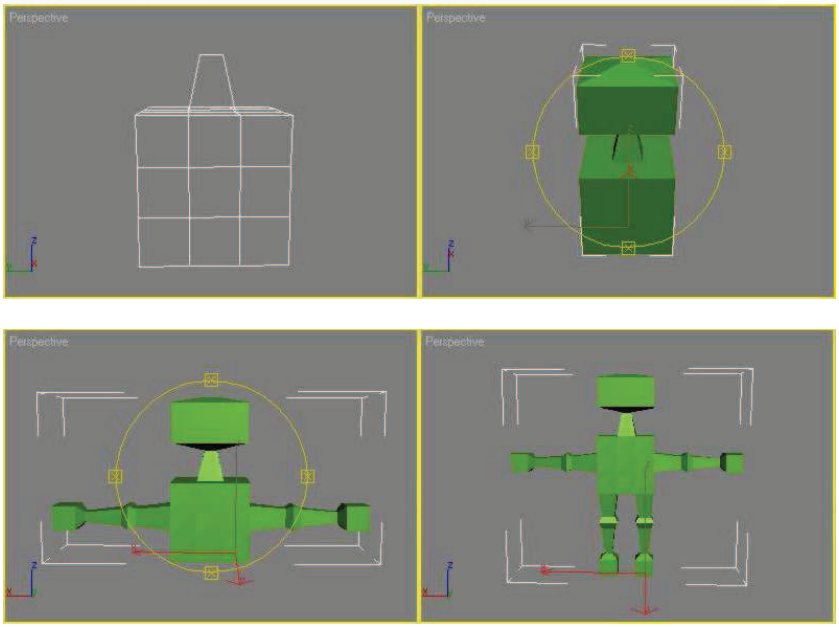


Imagen 5.3.27: Proceso típico de "Box Modeling"

Este proceso de modificación se realiza, normalmente, mediante dos operaciones que trabajan con toda una cara o polígono. Son la operación de “Extrude” y “Bevel”. Mientras “extrude” crea una extrusión de la cara en un determinado eje, “bevel” permite que además, la cara extruída pueda ser de distinto tamaño y por tanto creará una modificación en volumen de la figura. Si como en el ejemplo, la figura debe ser simétrica, tendremos que trabajar con las caras implicadas de ambos lados.

Otra operación muy común es la de ir al nivel de vértices y moverlos o escalarlos para reposicionarlos y por tanto modificar la forma de la figura para crear detalles. Como ejemplo, se puede ver el resultado del trabajo con vértices en la siguiente imagen.

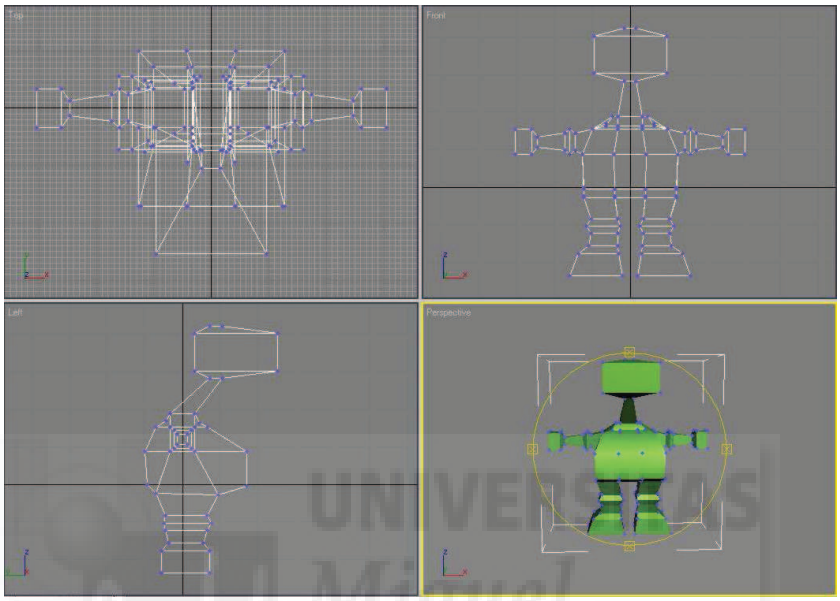


Imagen 5.3.28: Edición de los vértices

Por último, el proceso de modelado acaba normalmente con el suavizado de la malla para darle un aspecto más realista. Este paso no siempre es necesario ya que a veces buscamos modelos muy angulosos.

El suavizado de la malla se suele hacer en MAX mediante la aplicación de un modificador llamado “MeshSmooth” dónde podemos aplicar distintos métodos y valores de suavizado. En la siguiente imagen se observa el personaje terminado una vez suavizado y con los materiales aplicados. Los ojos se han modelado a parte mediante dos esferas.

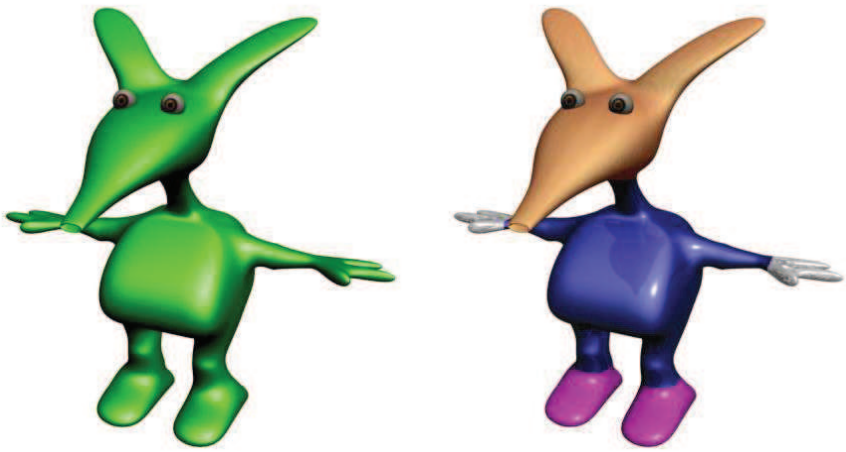


Imagen 5.3.29: Personaje realizado con “Box modeling”

Las mallas poligonales son modelos que funcionan muy bien con casi todos los métodos de animación, ya que al tener un control total con los vértices, los animadores pueden resolver muchos casos delicados que se dan al animar. Además existen programas que usan las mallas de forma muy efectiva automáticamente.

Por otra parte, MAX incluye programas como por ejemplo “Character Studio” para la creación de personajes animados bípedos que trabajan con el modelo en forma de malla.

9. Modelado de curvas y superficies: Nurbs

Aunque existen múltiples métodos para modelar curvas y algunos se utilizan directamente con mallas para suavizarlas como hemos escrito antes, en este punto nos centraremos en las llamadas Nurbs. Las nurbs crean modelos en sí mismas. Otros métodos como las nurms no son un proceso independiente ya que trabaja con aristas y por tanto se utilizan sobre otros métodos.

La palabra NURBS proviene de Non-Uniform Rational B-Spline. Es un método totalmente distinto a los que hemos visto anteriormente. MAX incluye una completa gama de herramientas para trabajar con NURBS.

Imagen 5.3.30: Herramientas Nurbs

En la anterior imagen podemos ver la caja de diálogo que presenta las herramientas que podemos emplear según el nivel de subobjeto con el que estemos trabajando. Como en el caso del “Editable Poly”, en las Nurbs también podemos trabajar a nivel de puntos, curvas y caras. El método sigue los pasos de la mallas poligonales, ya que se parte de una primitiva que convertimos en Nurbs pulsando con el botón derecho del ratón y en el menú contextual que aparece elegimos el comando Convert To > Convert to Nurbs. Entonces en el panel Modify podemos encontrar todo lo necesario para trabajar con este modelaje.

Este método es recomendado para crear modelos suavizados desde su origen o que deben presentar ciertas características para luego ser fabricados de la misma forma. Por ello, se suele emplear en la industria del automóvil, en el diseño de aviones, etc. El problema que presenta es que precisa de un alto grado de especialización y se considera más difícil de aprender que otros por lo que es menos empleado. En la imagen de abajo se puede observar el resultado de un modelado Nurbs típico: un tacón de formas curvas.

Imagen 5.3.31: Tacón modelado con Nurbs

10. Modelado de efectos naturales.

Los efectos naturales son modelos especialmente complejos para el modelador por sus especiales características. Por ejemplo, un terreno, es por definición un modelo muy grande, de una escala distinta a otros modelos como personajes, etc. Otros modelos como algunos efectos atmosféricos como la lluvia, la nieve,… son también difíciles de llevar a cabo. Por eso, vamos a comenzar con la clasificación de estos efectos naturales que vamos a dividir en tres tipos fundamentalmente: terrenos, efectos atmosféricos y sistemas de partículas.

Terrenos

Existen muchos métodos para crear terrenos en MAX, aunque uno de los más utilizados es el uso del modificador “Displace” sobre un plano.

Este método tiene la ventaja de su sencillez, ya que funciona a partir de una imagen en escala de grises y cuyo nivel de gris se traduce una vez aplicado en una determinada deformación en altura del plano. El modificador Displace se encuentra donde todos los demás, es decir, en la lista de modificadores del panel “Modify”.

En la siguiente imagen se puede apreciar la imagen en modo de escala de grises, su aplicación mediante la utilización de Displace y el resultado de texturizarlo.

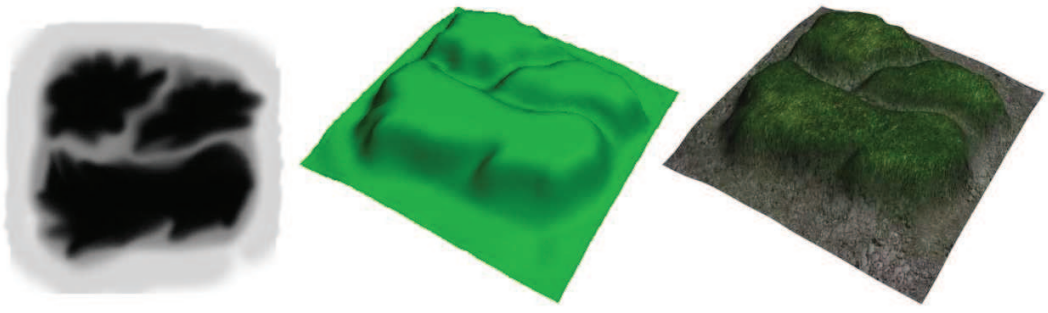


Imagen 5.3.32: Terreno realizado con el modificador “Displace”

Efectos atmosféricos

Estos efectos atmosféricos se pueden crear de muchas maneras en MAX.

El método más sencillo es crearlos al representar (renderizar) directamente sin ser modelados. Por ejemplo, la niebla se puede crear directamente desde el menú Rendering.

Otro método, quizá el más específico se encuentra en el panel Create > Ayudantes (Helpers) y en la lista desplegable elegimos Aparatos Atmosféricos. Esto permite hacer 3 tipos distintos de contenedores atmosféricos: cajas, esferas y cilindros. Una vez creados, pulsamos sobre el panel Modify y en “Atmospheres & Effects” tenemos la posibilidad de añadir al contenedor el efecto buscado, por ejemplo niebla.

Por último, para configurar el efecto, iremos al menú Rendering > Enviroment y a mitad de la caja de diálogo disponemos del efecto creado y algunos parámetros que dependen del efecto atmosférico en cuestión para que los personalicemos.

En la siguiente imagen podemos ver parte de la caja de dialogo “Enviroments and Efects” y los parámetros del efecto “Fire” junto con la representación obtenida.

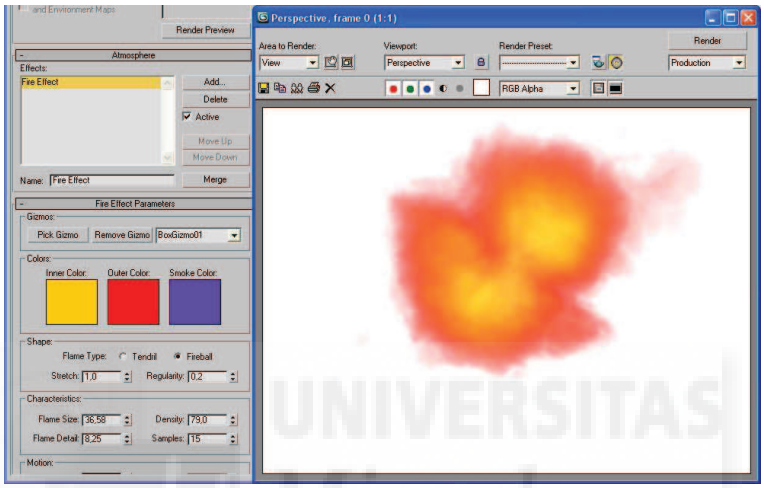


Imagen 5.3.33: Creación de efectos atmosféricos

En el siguiente punto vamos a estudiar los sistemas de partículas, con los cuales podemos crear algunos efectos atmosféricos a partir de ellas.

Sistemas de partículas

Los sistemas de partículas se utilizan frecuentemente en MAX. Con ellos podemos crear fluidos y efectos como brillos, chispas, etc.

Para crear sistemas de partículas accederemos al panel Create > Geometry y en la lista desplegable elegiremos “Particle Systems” que mostrará los controles que se observan en la siguiente imagen.

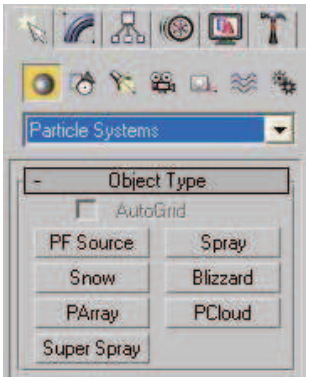


Imagen 5.3.34: Creación de sistemas de partículas

Los sistemas de partículas son el resultado de la interacción de dos elementos: el emisor y las partículas.

El emisor posee características que crean la forma en que son expulsadas las partículas, la dirección, el área de emisión y otras. El emisor no se representa en los ficheros de salida, es decir, no aparece en las imágenes ni en las animaciones.

Las partículas son los elementos representados del sistema. Normalmente son creadas en gran número y en realidad MAX las construye con elementos bidimensionales pero con características animadas, lo que les da apariencia de ser elementos en 3D.

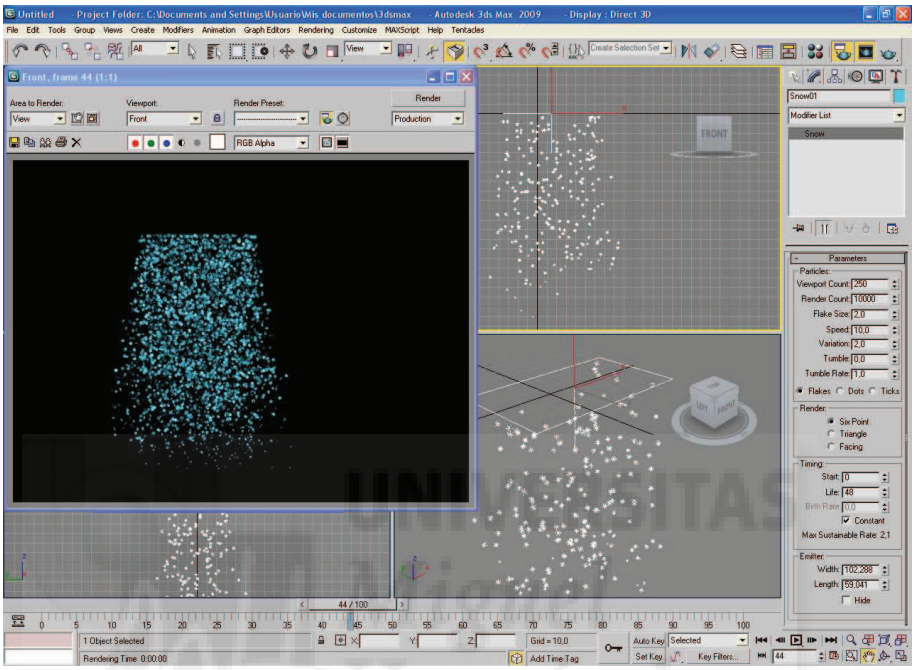


Imagen 5.3.35: Sistema de partículas

En la imagen anterior se puede ver un tipo de sistema de partícula que viene definido en MAX llamado nieve. Como se observa, es distinto lo que vemos en los visores que lo que se obtiene en la representación. Se visualiza en los visores un cuadrado blanco que es el emisor.

Los sistemas de partículas realizados en MAX son sencillos de llevar a cabo y el resultado es muy satisfactorio, pero no los vamos a exportar a Director (aunque tiene utilidad conocerlos sobre todo para tomar una idea de la forma de configurarlos) ya que los crearemos con Lingo, el lenguaje nativo de Director.

Esto mismo podemos decir de los efectos atmosféricos. No los exportaremos a Director y por tanto su modelado en MAX no es una cuestión decisiva. Con respecto al otro caso de modelado de efectos naturales, el del terreno, no hay duda que puede ser de gran utilidad para la creación de simulaciones de suelos en nuestros proyectos. Ya que podemos modelar así los terrenos en los que podemos colocar, por ejemplo, nuestras esculturas, etc.

Como ejemplo para finalizar, presentamos en la siguiente imagen unos modelos de maniquís que están realizados utilizando conjuntamente varias de las técnicas vistas en esta unidad teórica.

El trabajo de los maniquís que se visualiza a continuación está realizado a partir de dos primitivas: esferas y cilindro. Estas primitivas fueron posteriormente modificadas para realizar cabezas, torsos, brazos,... mediante modificadores. También se emplearon operaciones booleanas con objetos. Por ejemplo, se realizó un objeto ovoide y con una caja y la operación de substracción se obtuvieron los pies y las manos.

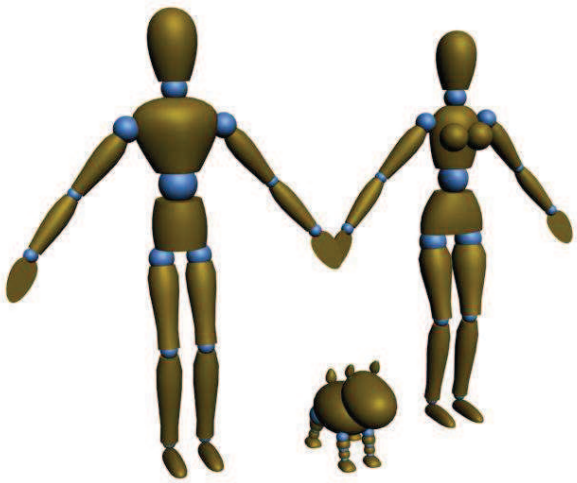


Imagen 5.3.36: Ejemplo de modelado paramétrico realizado con MAX

Caso práctico 3.1: Modelado con primitivas y técnicas básicas.

Objetivo: Crear un modelo donde utilicemos las técnicas e interfaz básica del modelado de primitivas estándar.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Crear la cabeza.
2. Crear el morro.
3. Crear las orejas.
4. Crear los ojos.
5. Crear una representación en JPG.
6. Guardar el archivo.

Imagen 5.3.cp.1.1: Distintas vistas del modelo creado

1. Crear la cabeza.

Una vez abierto 3DS MAX, seguimos los siguientes pasos:

1.1. Vamos al panel Crear > Geometry > Primitivas estándar y elegimos Esfera.

1.2. En la ventana Frontal creamos una esfera interactivamente, arrastrando el ratón.

1.3. En el panel crear ponemos en nombre “cabeza” y en color un marrón claro.

1.4. En radio le damos 60. Si por algún motivo hemos deseleccionado la esfera, para realizar todos estos cambios debemos de seleccionar la esfera y acceder al panel Modify.

1.5. Movemos la cabeza al origen de la escena 3D. Para ello seleccionamos la esfera y con el botón derecho pulsamos sobre la herramienta Mover. Aparecerá una ventana donde introducimos directamente los valores 0, 0, 0 en las coordenadas universales de posición. Debemos tener en cuenta que los ejes son según la vista por lo tanto, no confundir el eje Y con el eje Z y al revés.

1.6. Utilizamos el botón “Zoom Extended All Selected” para recolocar los visores.

2. Crear el morro y la nariz.

2.1. Creamos una esfera por el mismo método anterior pero de radio 28.

2.2. Le ponemos un color marrón oscuro y la llamamos “morro”.

2.3. Movemos el “morro” al punto 0,-47,0.

2.4. Creamos una esfera por el mismo método anterior pero de radio 12.

2.5. Le ponemos un color negro y la llamamos “nariz”.

2.6. Mover la “nariz” al punto 0,-72,7.

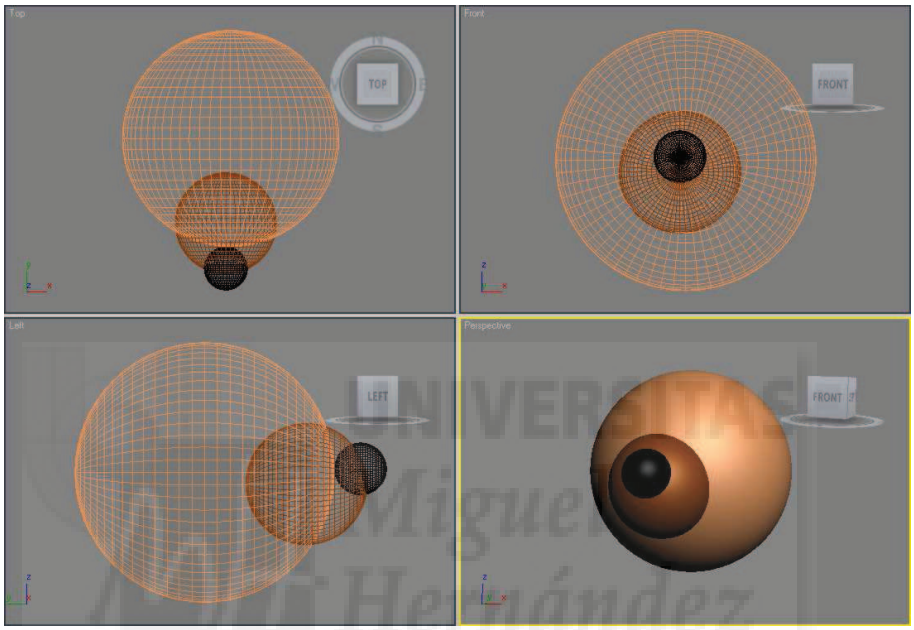


Imagen 5.3.cp.1.2: Cabeza, morro y nariz

3. Crear las orejas.

3.1. Creamos una esfera por el mismo método anterior pero de radio 18.

3.2. Le ponemos un color marrón oscuro y la llamamos “oreja derecha”.

3.3. Movemos esta oreja interactivamente hasta colocarla en su sitio.

Ahora vamos a copiar la “oreja derecha” con la herramienta Selección.

3.4. Pulsamos la tecla “Shift” a la vez que movemos la “oreja derecha”.

3.5. Aparece una caja de diálogo donde elegimos Copy y Aceptar.

3.6. La nombramos como “oreja izquierda”.

3.7. Nos movernos entre los visores para colocar “a ojo” la oreja izquierda. Para ello tenemos que pulsar sobre el nombre de los visores con el botón derecho.

4. Crear los ojos.

4.1. Creamos una esfera de tamaño 9 y color blanco.

4.2. Creamos una esfera por copia de la anterior de tamaño 8 y color negro.

4.3. Recolocamos interactivamente la bola negra sobre el globo blanco.

4.4. Seleccionamos las dos esferas y hacemos menú Group > Group. Esto nos permite agrupar objetos y que posteriores modificaciones afecten al grupo de objetos por entero, por ejemplo, si movemos todo el grupo que representa a un ojo, se moverán las dos esferas que lo forman a la vez, y también si lo hacemos más grande, lo giramos, etc.

4.5. Aparece una dialogo para introducir el nombre y lo llamamos "ojo der".

4.6. Hacemos una copia simétrica del ojo con Mirror > Copy y llamamos al grupo "ojo izq".

4.7. Movemos la nueva selección para colocarla donde corresponda.

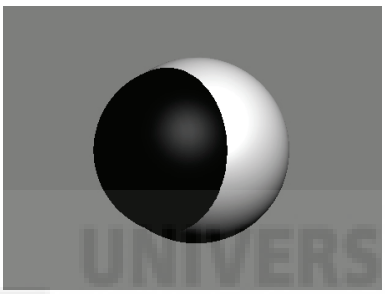


Imagen 5.3.cp.1.3: Las dos esferas que se agrupan para formar un ojo

5. Crear una representación en JPG.

5.1. Activar el visor que deseemos representar (renderizar).

Si no queremos que la representación tenga una salida a copia en disco pulsaremos F9. Esto puede suceder, por ejemplo, por que queremos ver cómo está quedando nuestro trabajo pero aún no lo hemos finalizado. Si queremos gestionar la representación para guardarla en ficheros de salida y otras operaciones ejecutamos el comando del menú Render > Render Setup o pulsamos F10.

5.2. Pulsar F10. En la diálogo que aparece elegimos en "Output size": 640 x 480.

5.3. En "Render Output", pulsamos sobre Files. Aparece una caja de diálogo para definir:

a) Carpeta de destino.

b) Nombre que queremos para el fichero de salida.

c) Formato de salida. Elegir "jpg" ya que queremos guardar una imagen.

5.4. Pulsamos el botón "Save file" y luego el botón "Render".

6. Guardar el archivo.

6.1. Ejecutamos la orden File > Save y guardamos como tipo ".max".

Conclusiones:

La realización de modelos utilizando solamente primitivas es muy sencilla, ya que solamente tenemos que elegir la primitiva y elegir unas dimensiones y una posición e ir componiendo así el objeto deseado. El problema es que los modelos naturales casi nunca son geométricamente perfectos y por tanto este método nos sirve o bien para crear objetos realizados por el hombre como mesas, balones,… o bien como base para luego aplicar cambios sobre ellos y modelar objetos más complejos.

Caso práctico 3.2: Modelado de extrusiones.

Objetivo: Crear objetos por extrusión, es decir, obtener objetos 3D a partir de la suma de una dimensión más a un objeto 2D.

Tiempo de realización: 1hora.

Pasos a realizar:

1. Crear el logo 2D.
2. Crear el texto 2D.
3. Crear los modelos 3D por extrusión.
4. Consideraciones al trabajar con splines.

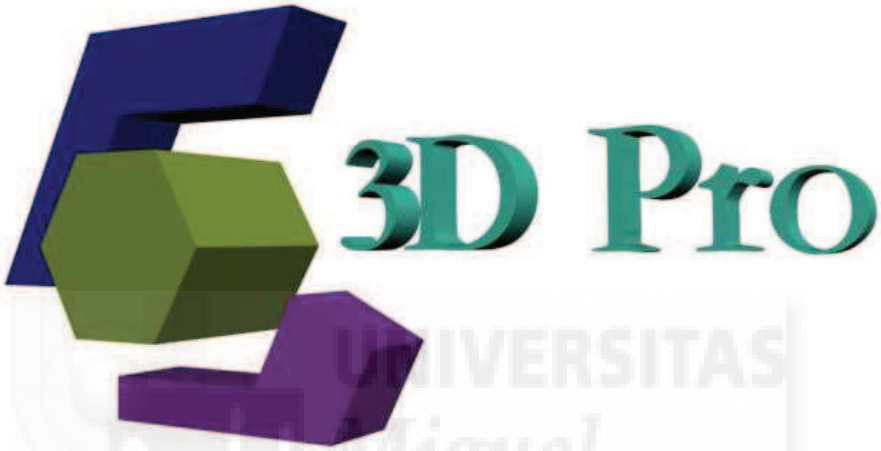


Imagen 5.3.cp.2.1: Resultado del caso práctico

1. Crear el logo 2D.

1.1. Activamos el visor frontal haciendo clic sobre él.

1.2. Activamos la ayuda “Snap 2D”. Esta ayuda a la edición hace que podamos basarnos en la rejilla de referencia para crear el logo. Al ser un diseño geométrico a base de polígonos, conviene tener esta característica activa.

1.3. Creamos un hexágono en Create > Shapes > Ngon. Sides 6. El tamaño lo vamos a establecer “a ojo” ya que son medidas relativas y será este primer polígono el que sirva de referencia a todos los demás objetos. Cambiamos su color a verde para reconocerlo mejor.

1.4. Utilizamos Create > Shapes > Lines para crear las formas azul y morada. Debemos activar “Start New Shape” para que sean formas independientes, sino, MAX interpretará que aunque sean inconexas, pertenecen a la misma entidad.

1.5. Es importante que sean figuras cerradas, es decir, que el último punto coincida con el primero. Max detectará este hecho cuando estemos cerca del punto inicial y preguntará si cierra la forma a lo que responderemos afirmativamente.

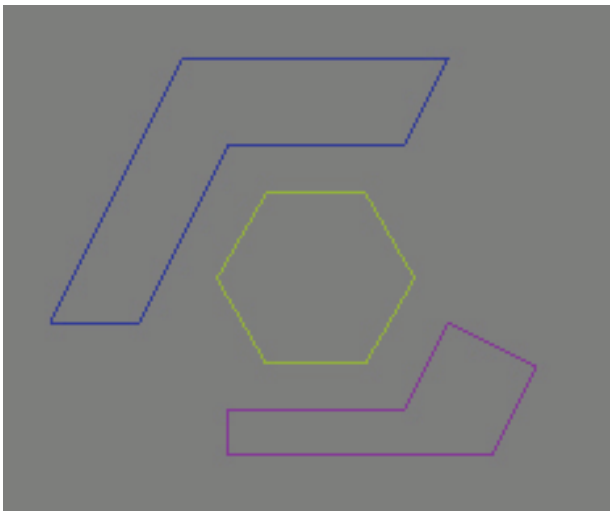


Imagen 5.3.cp.2.2: Vista desde el frontal del logo en dos dimensiones

2. Crear el texto 2D.

2.1. Creamos el texto con Create > Shapes > Text. Al elegir esta shape aparecen sus parámetros debajo. Colocar la fuente y tamaño deseados y hacemos clic en el visor frontal.

2.2. Escribimos lo que deseemos, en este caso “2D Pro”. Lo ponemos en color rojo.

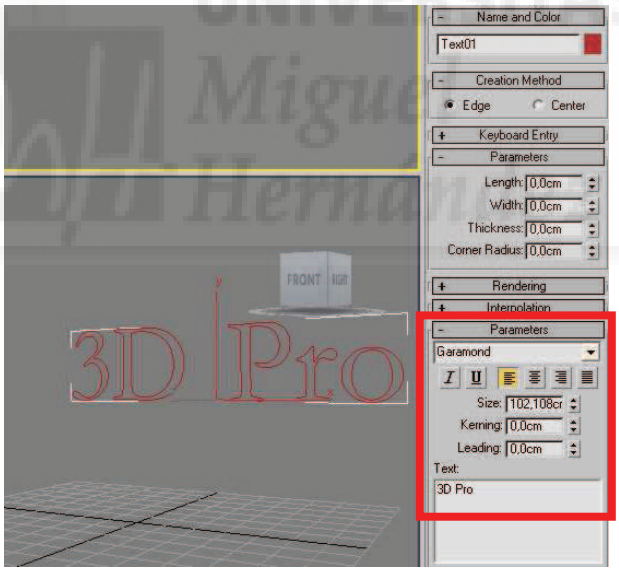


Imagen 5.3.cp.2.3: Parámetros del texto y como queda en el visor Perspectiva

3. Crear los modelos 3D por extrusión.

3.1. Seleccionamos el hexágono y accedemos al panel Modify. En la lista de modificadores buscamos “Extrude” y pulsamos sobre él.

Tenemos que decir que existen muchas clases distintas de modificadores pero todos los modificadores de modelado funcionan de una manera muy parecida: aparecen los parámetros debajo para poder asignar valores a distintas variables que los personalizan.

En la imagen de abajo se observa el modificador “Extrude” y sus parámetros.

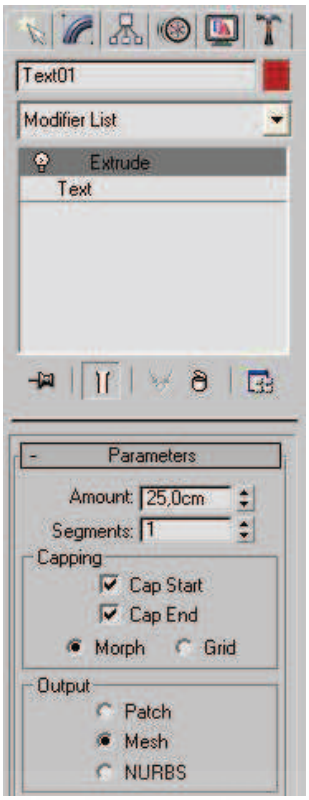


Imagen 5.3.cp.2.4: Parámetros del modificador Extrude

3.2. En cantidad (Amount) ponemos el valor 25. Esta será la profundidad del texto.

3.3. Repetir lo mismo con los demás elementos del logo y con distintas medidas.

4. Consideraciones al trabajar con Splines

Los logos con curvas, requerirán la utilización de Splines con puntos Bézier o Smooth por lo que debemos conocer algunas técnicas de imagen vectorial. Por ejemplo, para hacer punto de “pico”, hacemos clic y para hacer puntos Bézier hacemos clic y arrastramos sin levantar el botón del ratón.

Las formas Splines están formadas por puntos (como no tienen por que estar en un solo plano, pueden formar figuras 3D), por lo que es muy importante saber que debemos acceder al panel modificar y de allí al subnivel vértice para poder moverlos, etc.

Para modificar el tipo de un vértice pulsamos botón derecho sobre él y aparecen 4 paneles. En el de la izquierda y arriba tenemos los tipos distintos de puntos.

Si necesitamos crear más vértices en un segmento, utilizar la función “Refine” en el subnivel de vértices.

A veces es mejor hacer un diseño Spline partiendo de rectas y luego convertir sus puntos a curvas, sobre todo si vamos a utilizarlos para luego crear objetos booleanos.

Conclusiones:

Para realizar un modelo extruido, en realidad solo hace falta aplicar el modificador “Extrude” convenientemente. La mayor complejidad de esta práctica se debe al diseño de las formas 2D y no de la propia extrusión. Este tipo de modelado se utiliza mucho, sobre todo en la creación de arquitectura virtual, lo que se conoce como infoarquitectura.

Caso práctico 3.3: Modelado de revoluciones.

Objetivo: Crear objetos por el método de revolucionar un perfil 2D.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Crear el perfil recto y exacto con Splines.
2. Crear el perfil curvo.
3. Aplicar el torno.
4. Retoques.

Imagen 5.3.cp.3.1: Resultado del caso práctico

1. Crear el perfil recto y exacto con Splines.

Queremos crear el perfil de una copa. Cómo método habitual podemos comenzar creando el perfil con vértices rectos (corner). De esta forma podemos atender mejor a su posición, sobre todo para los vértices de los extremos, ya que deben estar exactamente en la misma coordenada para que al tornear quede un cuerpo cerrado. Para no tener que introducir las coordenadas de los puntos extremos de manera numérica y solo podamos crear vértices en las intersecciones de la rejilla, hacemos lo siguiente.

1.1. Pulsar sobre el botón Snaps Toggle 2D.

1.2. Activar el visor “Left”.

1.3. Panel Create > Shapes > Line. Ir pulsando con el botón izquierdo para ir creando puntos del perfil. Crear los vértices teniendo en cuenta que no vamos a tener una forma cerrada, por lo tanto para terminar el perfil pulsaremos botón derecho.

En la imagen que sigue se observa cómo debe quedar la figura 2D vista desde el visor izquierdo.

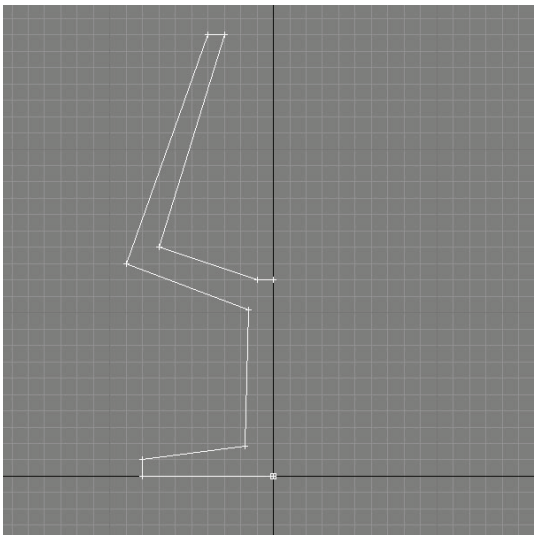


Imagen 5.3.cp.3.2: Perfil recto

2. Crear el perfil curvo.

Ahora convertimos los segmentos rectos en curvos al suavizar los vértices. No modificaremos de posición los vértices primero y último ni los suavizaremos.

2.1. Elegir un vértice y hacer botón derecho > Smooth.

2.2. Mover los vértices o convertirlos a Bézier hasta que nos guste el perfil.

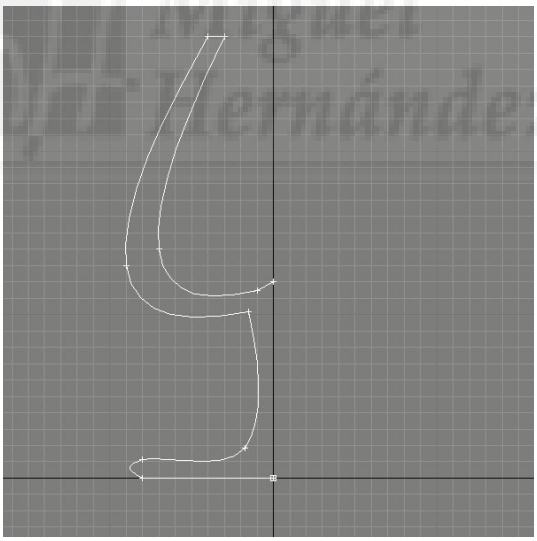


Imagen 5.3.cp.3.3: Perfil curvo

3. Aplicar el torno.

3.1. Con la spline seleccionada, accedemos al panel Modify y buscaremos “Lathe” en la lista de modificadores.

3.2. En los parámetros de Lathe, en “Direction” poner Y y en “Align” poner MAX.

Podemos experimentar con otras direcciones y alineaciones para comprobar como con un mismo perfil se pueden obtener distintos modelos.

4. Retoques.

Estos retoques solo serán necesarios si el modelo no se ha visto modificado con Lathe de la manera esperada.

4.1. En Paramenters de Lathe activar “Weld Core” y “Flip Normal”.

4.2. También podemos necesitar aumentar el número de segmentos para que el suavizado de la revolución sea de mayor calidad.

4.3. Podemos modificar el perfil en el nivel “Vertex” de la Spline y al acceder a cada uno de los vértices que definen el perfil. Al modificar estos, también estamos modificando el modelo 3D final. Por ejemplo, si queremos hacer más fina las paredes de la copa o hacer otra clase de copa distinta podemos basarnos en esta facilidad para ahorrar tiempo en la edición del perfil.

Conclusiones:

La realización de modelos obtenidos por revolución es realmente sencilla y muy intuitiva, ya que es un método de fabricación utilizado por el hombre desde el principio de nuestra historia sobre todo para la creación de objetos como vasos, botellas, etc.

Caso práctico 3.4: Modelado de solevaciones.

Objetivo: Crear modelos 3D a partir de 2 objetos 2D: una forma y una trayectoria.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Crear la forma (Shape) y la trayectoria (Path) mediante Splines.
2. Crear el modelo de tipo Loft.
3. Aplicar deformaciones.

Imagen 5.3.cp.4.1: Resultado del caso práctico

1. Crear la forma (Shape) y la trayectoria (Path) mediante Splines.

1.1. Creamos en la vista Top una Spline con forma de cuadrado mediante la forma Rectángulo y la introducción de valores numéricos de igual valor para la longitud y la anchura.

1.2. Creamos en la vista From una línea Spline recta. Esta línea es un ejemplo, podemos usar una curva (para crear esquinas), una forma 3D (para crear tubos) o cualquier otro tipo de trayectoria.

Podemos usar Snap Toggle 2D para ayudarnos en la precisión.

2. Crear el modelo de tipo Loft.

2.1. Seleccionamos la línea recta.

2.2. Accedemos al panel Create > Geometry > Compound Objects > Loft.

2.3. En el panel Creation metod > pulsamos Get Shape.

2.4. Pulsamos sobre la Spline cuadrada. Como resultado tendremos una caja.

3. Aplicar deformaciones.

3.1. Seleccionamos la caja creada y pulsamos sobre el panel Modify.

3.2. En el panel Deformations usar Twist y Scale como muestran las imágenes de abajo. Para crear puntos de control y moverlos se usará la barra superior. Los puntos se pueden cambiar de tipo pulsando botón derecho sobre ellos.

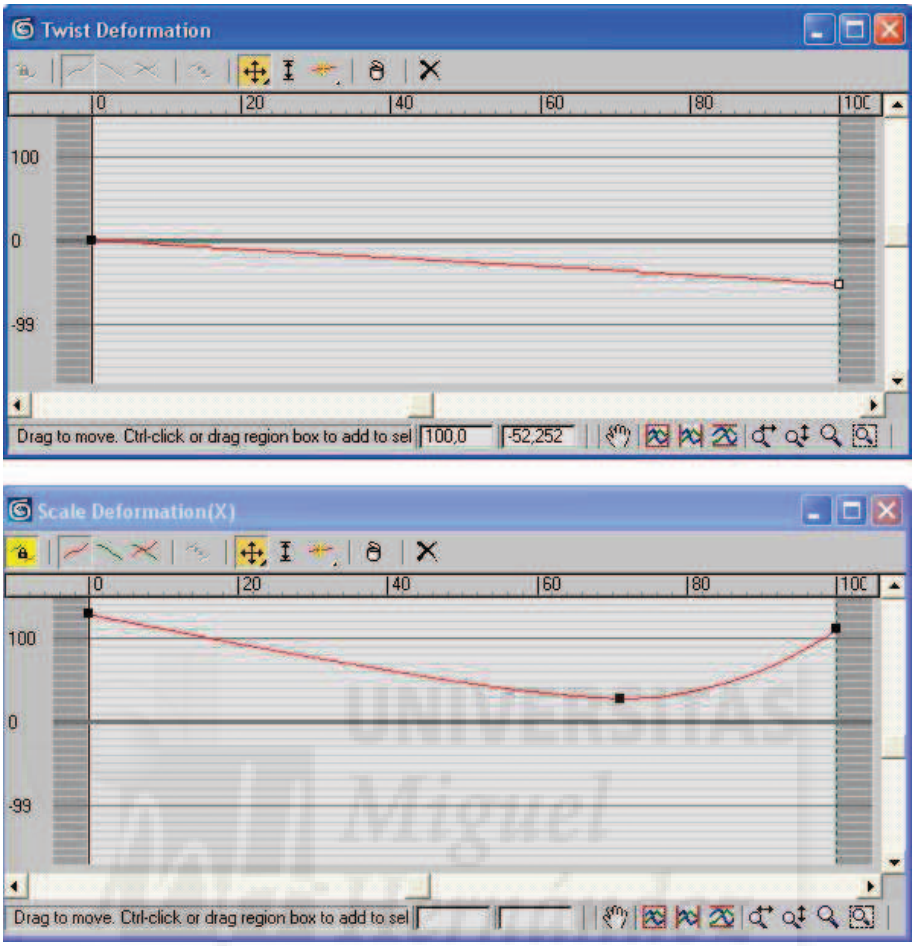


Imagen 5.3.cp.4.2: Deformaciones aplicadas

Existen muchos objetos que se pueden hacer como formas que recorren luego una trayectoria, como marcos, molduras, etc.

Es posible utilizar esté método para crear objetos Loft que empiecen con una forma y acaben con otra o incluso que a lo largo de la trayectoria soleven varias formas utilizando en el método de creación el porcentaje y una cantidad, de tal forma que cuando ya tengamos un objeto Loft podemos seguir pulsando “Get Shape” para añadir formas.

Conclusiones:

El método de solevación es muy importante ya que las deformaciones que podemos aplicar a la solevación base permite crear una variedad muy grande de objetos como cuernos, botellas de perfume, etc.

Ya hemos comentado que quedan muchas opciones por explicar sobre las posibilidades que ofrece MAX para la edición de los objetos solevados, pero son tantas que están fuera del objetivo de este texto.

Caso práctico 3.5: Modelado poligonal.

Objetivo: Vamos a crear un diseño 3D de un humanoide extraterrestre mediante el modelado poligonal y más concretamente utilizando la técnica conocida como "Box modeling".

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Crear la caja base.
2. Convertir en Editable Poly.
3. Creación de la cabeza.
4. Suavizar la malla con Meshsmooth.
5. Crear brazos y piernas.
6. Trabajo con los vértices.
7. Detalles.
8. Preparar los polígonos para aplicarles materiales.
9. Aplicación de materiales al humanoide.

1. Crear la caja base.

1.1. En el panel Create > Geometry, seleccionamos Box y hacemos una caja en Top.

1.2. En los parámetros de la caja, modificamos las dimensiones para que sean iguales en alto, ancho y profundo. Por ejemplo: 1000 y también ponemos 3 segmentos en las 3 dimensiones. La nombramos como "extraterrestre" y le ponemos un color verde.

1.3. Centramos la caja en el mundo con la herramienta de mover.

Quitamos las rejillas de los visores ya que a partir de aquí vamos a modelar "a ojo" en la vista perspectiva, por lo que no nos hace falta la rejilla de referencia. Ponemos el visor perspectiva en modo de alámbrico pulsando F3.

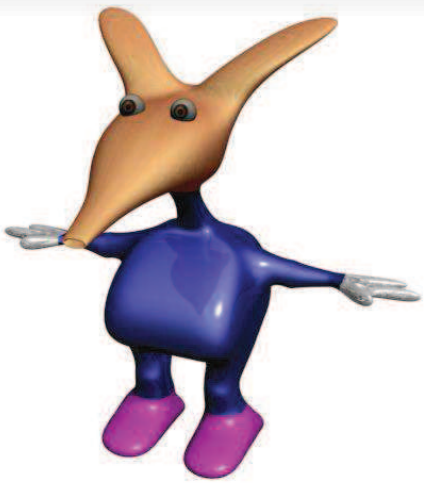


Imagen 5.3.cp.5.1: Resultado del caso práctico

2. Convertir en Editable Poly.

En la imagen inferior se puede observar el trabajo realizado: como convertir una caja en un "editable poly" que es una entidad con muchas más posibilidades de edición 3D.

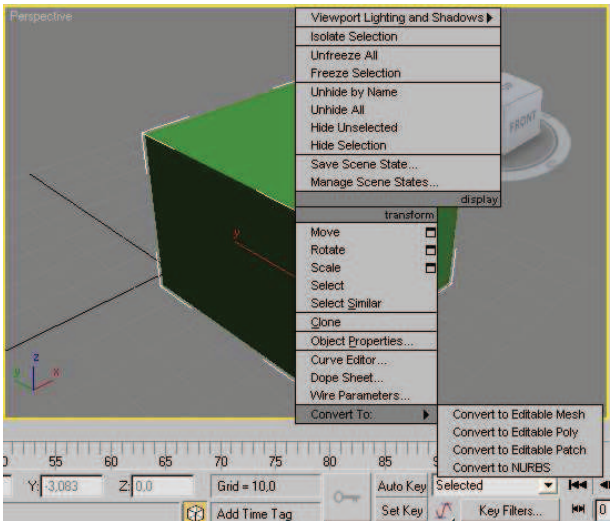


Imagen 5.3.cp.5.2: Convertir a Editable Poly

2.1. Pulsamos botón derecho sobre la caja > Convert To > Convert to Editable Poly.

Observaremos que pasamos automáticamente al panel Modify. Este panel es donde vamos a trabajar de aquí en adelante, por lo que debemos dominarlo.

3. Creación de la cabeza.

Tenemos el problema añadido de que el extraterrestre humanoide queremos que sea simétrico con respecto a la vertical. Aquí tenemos 2 opciones:

a. Trabajar todo el rato con dos polígonos seleccionados previamente.

b. Trabajar con una parte y al terminar hacer una copia simétrica para luego “coser” las dos partes.

Por simplicidad conceptual (que no instrumental) utilizaremos la opción a.

3.1. Crear el cuello:

3.1.1. Elegimos la selección “Polygon”. Seleccionar el polígono central de la parte superior.

3.1.2. Pulsamos sobre el botón al lado de Bevel, Settings. Abre una caja de diálogo:

3.1.2.1. Introducimos en Height el valor 400.

3.1.2.2. Introducimos en Outline Amount el valor -50.

3.2. Crear la cabeza. Vamos a escribir solamente el concepto, el valor y comentarios.

3.2.1. Height: 100.

3.2.2. Outline Amount: 400. Crear la parte inferior de la cabeza.

3.2.3. Extrude: 500. Crea la parte central de la cabeza.

3.2.4. Bevel: 100, -400. Esto redondea la parte superior de la cabeza.

En la imagen que sigue se puede ver el desarrollo de los polígonos que forman primero el cuello y luego, sobre este, lo que posteriormente será la cabeza.

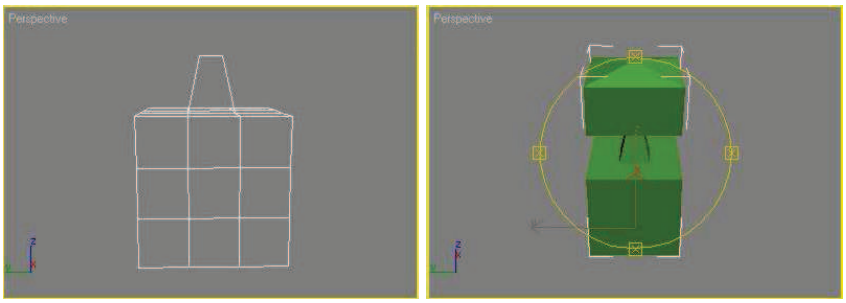


Imagen 5.3.cp.5.3: Creación de los polígonos de la cabeza

Para crear un morro podemos hacer Bevel con el polígono frontal de la cabeza varias veces y en el último ponerle un Height negativo para que se meta hacia el cuerpo formando así la boca. Lo realizamos más adelante en los detalles.

4. Suavizar la malla con Meshsmooth.

Vamos a realizar momentáneamente un suavizado de la malla para ver cómo va quedando nuestro diseño. En forma suavizada será como finalmente lo exportemos a Director.

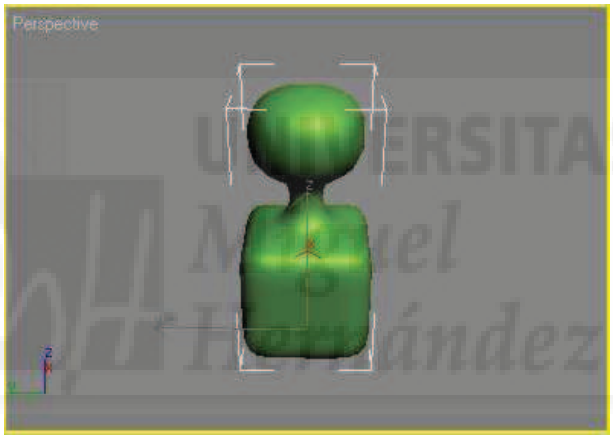


Imagen 5.3.cp.5.4: Aplicación de MeshSmooth

4.1. Acceder a la lista de modificadores y aplicar “MeshSmooth”.

4.2. En los parámetros que aparecen, ponemos en Interations el valor 3.

En el visor perspectiva pulsamos F3 para visualizarlo en modo sólido.

En la imagen anterior se observa el resultado obtenido y cómo mediante este método tenemos un modelo más realista manteniendo relativamente pocos vértices y por tanto, pocos polígonos.

5. Crear brazos y piernas.

5.1. No queremos seguir visualizando MeshSmooth para seguir editando el modelo pero no debemos borrarlo, ya que lo visualizaremos o no, según queramos ir comprobando nuestro trabajo. Para ello tenemos que pulsar en la pila de modificadores sobre el icono de la bombilla.

5.2. Seleccionamos los polígonos centrales de ambos lados de la caja. Para ello seleccionar uno de ellos, rotar la caja en la vista perspectiva para visualizar el polígono opuesto y pulsar la tecla “Control” antes de seleccionarlo. Control genera una selección múltiple.

5.3. Aplicamos los siguientes Bevel consecutivamente: 500, -50; 100, 50; 100,-50; 500,-50; 100,100; 300,0 (o extrude) para lo que serán los brazos.

5.4. Repetimos el proceso anterior pero con los polígonos desde donde salen las piernas. Bevel: 500, -50; 100, 50; 100,-50; 500,-50; 100,100; 300,0 (o extrude) para lo que serán las piernas.

Podemos modificar los vértices de codos y rodillas para darle mayor realismo.

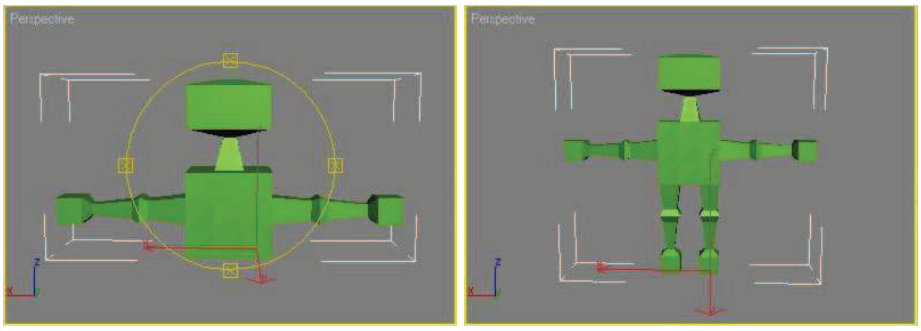


Imagen 5.3.cp.5.5: Creación de los polígonos de los brazos y las piernas

6. Trabajo con los vértices.

Esta figura es tan simétrica que más parece una pieza de lavadora que un humanoide. Pero tiene las partes fundamentales: cabeza, brazos y piernas, y por ejemplo, no es un cuadrúpedo. Ahora modificaremos los vértices para ir “modelando” el extraterrestre.

Las operaciones que realizaremos tienen un problema y es que se tienen que trabajar con todos los vértices de una alineación de aristas. Esto se resuelve fácilmente seleccionándolos mediante una ventana que los incluya. Las operaciones más habituales son mover y escalar para ensanchar o estrechar.

6.1. En el visor Left movemos los vértices como muestra la figura para hacerlo con más barriga.

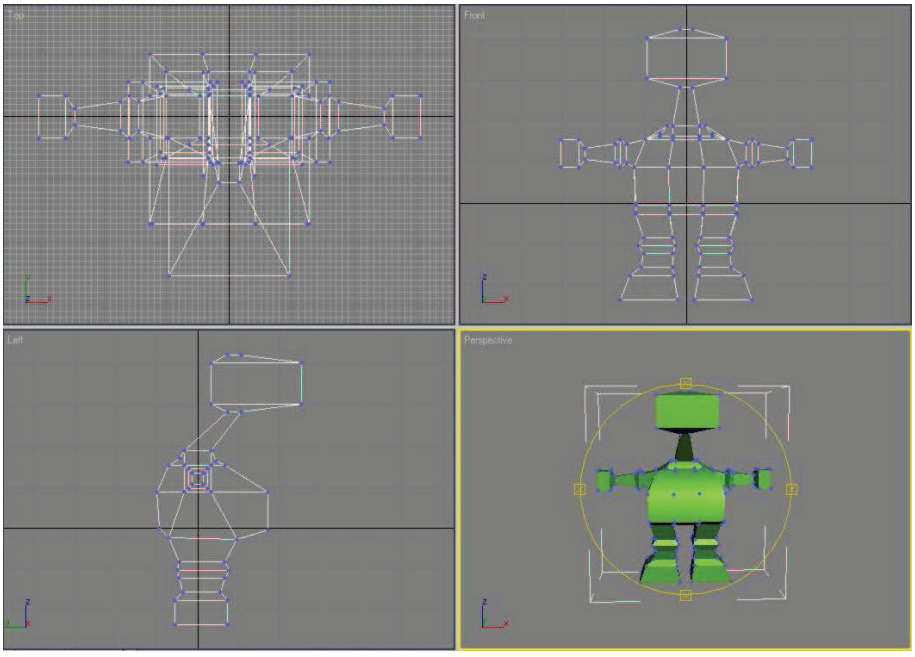


Imagen 5.3.cp.5.6: Movemos los vértices para modelar el cuerpo

6.2. Vamos a la vista Front y seleccionamos solo los vértices de sus hombros (primero unos y luego con la tecla Control pulsada los opuestos). Usamos escalar en X para estrechar los hombros como muestra la figura.

7. Detalles.

Para añadir detalles como dedos, boca, nariz y otros, tenemos que tener más polígonos. Una opción es "cortar" un polígono en 2. Vamos a realizar este trabajo.

7.1. Poner el visor en modo "Smooth + Highlights" y "Edged faces".

7.2. Hacemos las manos creando un pulgar y dos dedos.

7.2.1. Situar la perspectiva para trabajar con el polígono del extremo de la mano.

7.2.2. Seleccionar el polígono.

7.2.3. Con la herramienta "Cut", hacemos un corte vertical (marcando 2 vértices en la arista superior e inferior) del polígono.

7.2.4. Pasar al subobjeto vértice para mover los vértices recién creados y centrarlos, igualando así el área de los polígonos.

7.2.5. Pasar al subobjeto Arista, se debe seleccionar la arista intermedia. Aplicar "Chafer" en una cantidad de 10. Esto permitirá separar los dedos.

7.2.6. Seleccionar los dos polígonos extremos que generarán los dedos y aplicar un Bevel de 500, -50.

7.2.7. Seleccionar los polígonos anteriores frontales, para hacer los pulgares y aplicar un Bevel de 500, -50.

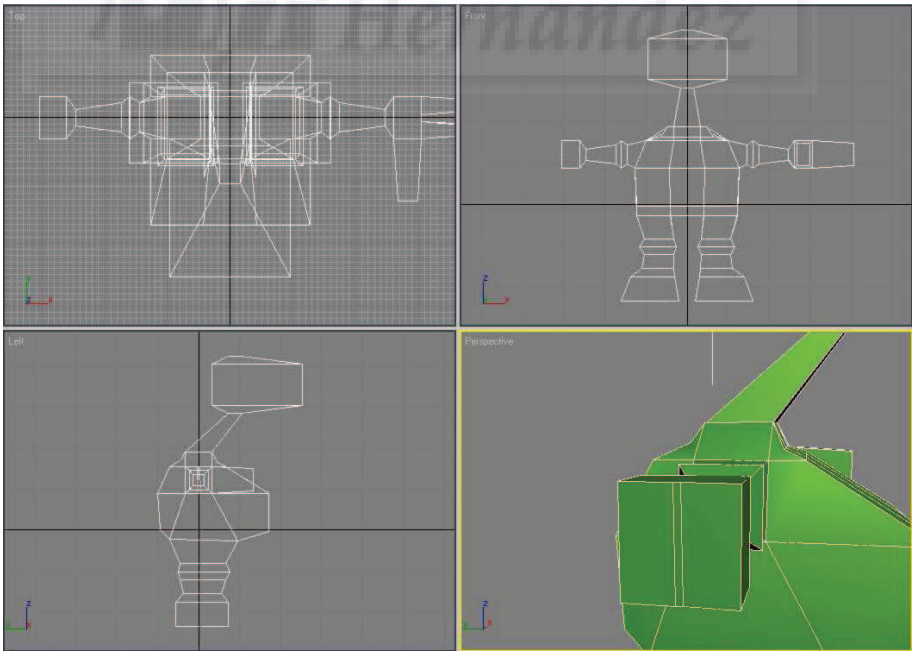


Imagen 5.3.cp.5.7: Creación de los polígonos de las manos y los pies

Repetiremos el proceso en la otra mano.

Ahora ajustamos los vértices en las dos manos a la vez para ajustar el tamaño de las manos. Normalmente, necesitaremos adelgazarlas en el eje vertical.

Repetiremos pasos parecidos para realizar los pies. En la siguiente imagen se puede observar parte de este proceso: la creación de los dedos de la mano derecha.

7.3. Hacer el morro:

7.3.1. Aplicar un Bevel al polígono frontal de la cabeza 2 veces y un último con Height negativo para realizar el morro.

Ahora queremos ver de nuevo como es nuestro modelo una vez aplicado el suavizado de malla. Para ello tenemos que pulsar en la pila de modificadores sobre el icono de la bombilla al lado de “MeshSmooth”.

La imagen siguiente muestra el resultado obtenido tras el trabajo realizado. Tenemos que tener en cuenta que se ha seguido “limando” algunos polígonos trabajando a nivel de vértices para exagerar unas características y disimular otras.

En la imagen se observan los ojos. Estos son modelos esféricos independientes que se han texturizado anteriormente mediante un mapa difuso.

Imagen 5.3.cp.5.8: El personaje ya modelado

8. Preparar los polígonos para aplicarles materiales.

Estos dos puntos que quedan no corresponden a la unidad 3 sino a la 4 que trata el tema de la texturización, por lo que es conveniente leerla antes de proseguir para entender lo que vamos a realizar.

La idea es que los polígonos que forman la malla se puedan texturizar con materiales distintos, para ello los seleccionamos y les asignamos un identificador de grupo (de polígonos).

8.1. Seleccionar todos los polígonos del modelo con una ventana que tome todo el extraterrestre para crear una asignación base, que después se traducirá por un material base.

8.2. Vamos al grupo “Polygon Properties” y en “Set ID” ponemos un 1.

8.3. Seleccionamos los polígonos que componen la cabeza y ponemos SetID a 2.

8.4. Seleccionamos los polígonos que forman las manos y ponemos SetID a 3.

8.5. Seleccionamos los polígonos que forma los pies y ponemos SetID a 4.

8.6. Si queremos comprobar que hemos realizado el trabajo bien, usaremos “Select Id”, escribiendo el número y pulsando el botón para comprobar que se selecciona los polígonos previstos.

9. Aplicación de materiales al humanoide.

9.1. Abrimos el editor de materiales pulsando la tecla “M”.

9.2. Elegimos un slot y llamamos al material “extraterrestre”.

9.3. Pulsamos sobre el botón Standard y elegimos el tipo de material Multi/Subobject.

Este material permite texturizar una malla por partes. El número de cada material se asignará a los polígonos que tengan asignado el mismo número (que establecimos antes).

9.4. Ahora solo se trata de elegir o crear los materiales deseados y arrastrarlos hasta el submaterial con el número apropiado.

En la imagen 5.3.cp.5.1 se puede observar el personaje texturizado. Tenemos que advertir que hemos utilizado materiales directamente de la librería estándar de MAX.

Como hemos escrito antes, para tener pleno conocimiento de lo realizado en el punto 8 y 9 debemos leer antes la unidad 4.

Conclusiones:

El modelado de mallas es tremendamente útil para modelos objetos complejos ya que como hemos visto podemos trabajar a distintos niveles según necesitemos. Primero podemos modelar más toscamente y luego ir accediendo a distintas partes para añadir polígonos que podemos modificar para generar más y más detalles.

Sería un buen ejercicio que el lector iniciara la tarea de crear con la misma técnica llamada “box modeling” la creación de un cuadrúpedo, por ejemplo un perro extraterrestre que acompañe a nuestro personaje.

Unidad 4: Materiales: superficies y texturas en MAX.

Introducción teórica:

1. Introducción a los materiales en MAX.
2. Materiales estándar.
3. Materiales mapeados.
4. Materiales compuestos.

Casos prácticos:

4.1. El editor de materiales.

4.2. Creación de superficies básicas.

4.3. Aplicación de mapas difusos.

1. Introducción a los materiales en MAX

Hay que empezar comentando que los profesionales de los programas 3D siempre dicen que "es más importante texturizar bien que modelar bien". Pero cómo para texturizar hay que hacerlo sobre un modelo, siempre es un "segundo" trabajo y a veces no se le da la importancia que requiere.

De todas formas, no debemos confundir las texturas con los materiales. Los materiales son como la pintura que se le aplica a un objeto y por tanto, los materiales incluyen las texturas y no al revés. Con los materiales hacemos que un ovoide parezca un huevo de codorniz y una esfera parezca una naranja. Pero no solo eso, ya que podemos hacer que un mismo objeto parezca de cristal pulido o de cristal al ácido.

Mapear es el método de proyectar los materiales sobre la superficie de un objeto, que la mayoría de las veces será tridimensional. Es como envolver un regalo 3D con un papel 2D.

Los materiales son independientes de los objetos y se pueden crear y gestionar independientemente. La mayoría de los programas incluyen un editor de materiales para crearlos y gestionar colecciones de estos en las llamadas "bibliotecas de materiales".

Estudiaremos los materiales del programa 3DS MAX, que como se dice a lo largo del trabajo, es el más utilizado en el mundo. En MAX, los materiales se trabajan desde la ventana llamada "Editor de Materiales" que vemos en la imagen que sigue.

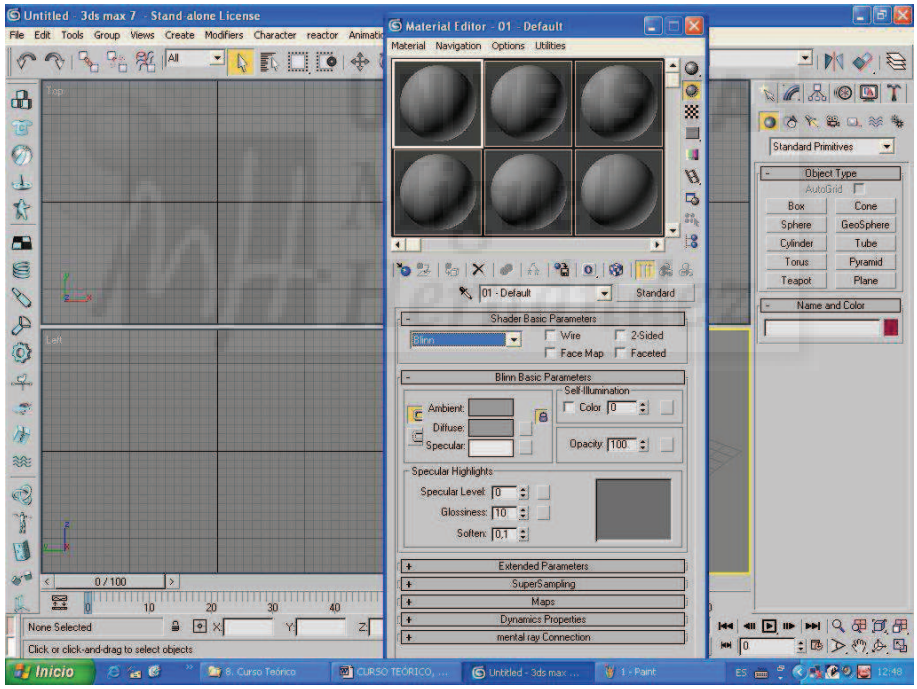


Imagen 5.4.1: Ventana del "Editor de Materiales"

El editor de materiales permite crear y gestionar todo tipo de materiales. En la actualidad se puede recrear cualquier material real. Esto nos da una idea de lo complicado que puede ser el tener un conocimiento suficiente como para controlar este editor. De todas formas, nos iniciaremos en su conocimiento en los casos prácticos adjuntos a este tema.

Un material se describe en función del tipo de superficie con la que se quiere "recubrir" el modelo y de la luz (y por tanto de las sombras) que incide sobre él. Por tanto, incluye datos sobre color, brillo, transparencia y otras muchas cualidades.

Un modelo puede tener muchos materiales. Para que un objeto adquiera un material para toda o parte de su estructura, este se le debe de asignar. Por lo tanto, en un objeto complicado puede haber muchos materiales distintos.

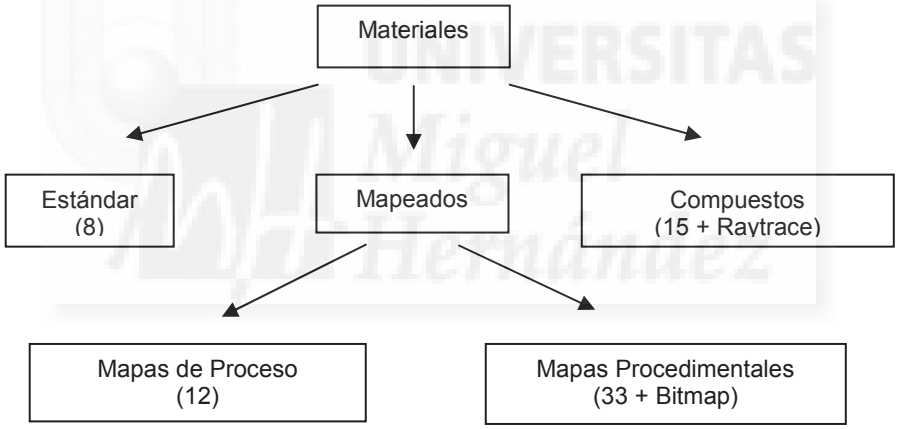
Los materiales incluyen la mayoría de la veces, los llamados mapas, es decir, imágenes que sirven para sustituir valores de la materia como el color, la opacidad o el relieve. Por lo tanto, en un mismo material pueden coexistir mapas de texturas, de opacidad, de relieve y otros. Podemos decir que los materiales son de dos tipos: los naturales y los no naturales (fabricados por el hombre). Los fabricados suelen contener mayor cantidad de mapas. Para obtener imágenes realistas, se tiene que tener en cuenta los siguientes aspectos:

1. Obtener mapas de “texturas” realistas mediante cámaras digitales, escáneres, programas de dibujo, etc.

2. La iluminación de la escena, ya que si el material que más nos interesa está en la penumbra no se visualizará.

3. La distancia a la que se coloque la cámara del material, ya que a mucha distancia no se apreciarán los detalles del trabajo y si está demasiado cerca pueden verse el pixelado de la textura.

Para hacernos una idea de la complejidad y las grandes posibilidades de la que pueden disponer los artistas, se presenta una clasificación de los materiales y entre paréntesis el nº de tipos distintos que posee MAX:



Gráfica 2: Tipos de materiales de MAX

Se pueden distinguir tres tipos de materiales según sus componentes:

1. Estándar: tienen un único color debido a sus propiedades físicas y de luz.

2. Mapeados: los que incluyen imágenes.

    2.1. Mapas de proceso: imágenes bitmaps aplicadas a cierto aspecto.

    2.2. Mapas procedimentales: son imágenes creadas por un programa.

3. Compuestos: los que incluyen otros materiales.

2. Materiales estándar:

Antes de nada, tenemos que escribir que existe un problema con los materiales estándar y es que en 3D Studio MAX vamos a poder crearlos con muchas características que después no se

van a poder exportar a Shockwave 3D. Por lo tanto, vamos a estudiarlos conociendo este hecho.

Los materiales estándar son sencillos en el sentido que aplican parámetros para definir la superficie de los objetos que son básicos, que definen la naturaleza misma del material. Por ejemplo, si queremos crear un material de oro, tendremos que conocer los valores que hacen que el color, el sombreado, la luz, la transparencia y otras cuestiones definan una superficie dorada. El problema es que estas cuestiones no son sencillas y en muchos casos deberemos acudir a conocimientos de óptica que en general pueden rebasar nuestros conocimientos. La solución puede ser el utilizar materiales de biblioteca ya creados o la experimentación con materiales parecidos.

Los parámetros que definen los materiales estándar se presentan en la imagen siguiente y los vamos a comentar todos brevemente.

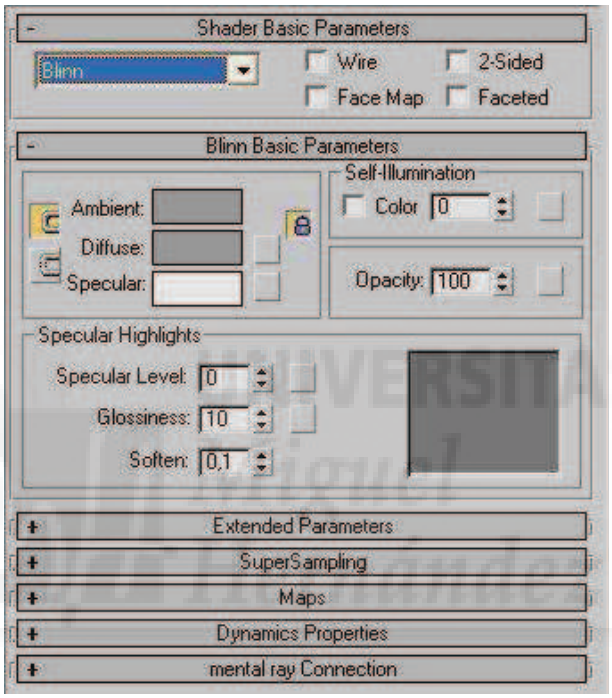


Imagen 5.4.2: Parámetros básicos de los materiales estándar

Sombreado

Es el parámetro primario del material y la función de respuesta a la incidencia de la luz. MAX incluye 8 tipos que van desde los de aspecto mate a los más brillantes. Estos son:

- Anisotropic: se utilizan para crear los brillos lineales de los metales pulidos.
- Blinn: es un derivado del Phong, pero más preciso y avanzado.
- Metal: se utiliza para metales no pulidos.
- Multi-Layer: se utiliza como dos Anisotropic en uno. Por ejemplo, para superficies con brillos metálicos cubiertos por ceras como las superficies de los coches.
- Oren-Nayar-Blinn: es una variación del Blinn pero con bordes difuminados.
- Phong: para suavizar caras. Es de los más antiguos.
- Strauss: para metales pero definido con más características.

◆ Translucent Shader: permite crear materiales translucidos.

Colores básicos

Son los colores básicos que presenta el material y vienen dados por como responden de distinta manera a la cantidad de luz que incide sobre el material, y son:

◆ Ambiental: color que tendrá una zona en sombra del objeto.

◆ Difuso: color que tendrá en la zona más expuesta a la luz. Este es el color propio del objeto, su color base.

◆ Especular: que será el color del reflejo brillante del objeto. El color del brillo.

Brillo

Es muy importante para ciertos materiales y viene dado por cuatro variables:

1. Color especular: el color del propio brillo. Descrito arriba.

2. Curva de máximo brillo: define la forma del brillo.

    2.1. Lustre: modifica la anchura de la curva, o sea, del brillo.

    2.2. Nivel especular: modifica la altura de la curva, o sea, la intensidad.

3. Debilitar: con valores extremos, suaviza los bordes del brillo para más realismo.

Opacidad

Se utiliza para controlar la transparencia de materiales como vidrio o agua. Es transparente en una medida de 0 a 100. La total opacidad será con el valor 100. Hay que tener en cuenta una serie de detalles:

1. El color de la luz que atraviesa el material viene dada por el “Filtro”.

2. Como se debe “ver” a través del material, se activará la opción “2 lados”.

3. Para trabajar con materiales transparentes mejor pulsar el botón “Fondo”.

4. Para un mayor control de cómo la luz atraviesa el material, accedemos a los parámetros extendidos > Opacidad. Tenemos opciones como atenuación, tipos de opacidad…

Filtro

Color de la luz que pasa a través del objeto si este no es totalmente opaco.

Autoiluminación

Un material no puede emitir luz pero puede parecerlo si es muy claro. Para crear un objeto luminoso como una bombilla, necesitamos 3 cosas, la geometría del objeto, una luz y un material asignado con autoiluminación. Viene dado por un color y un valor de intensidad.

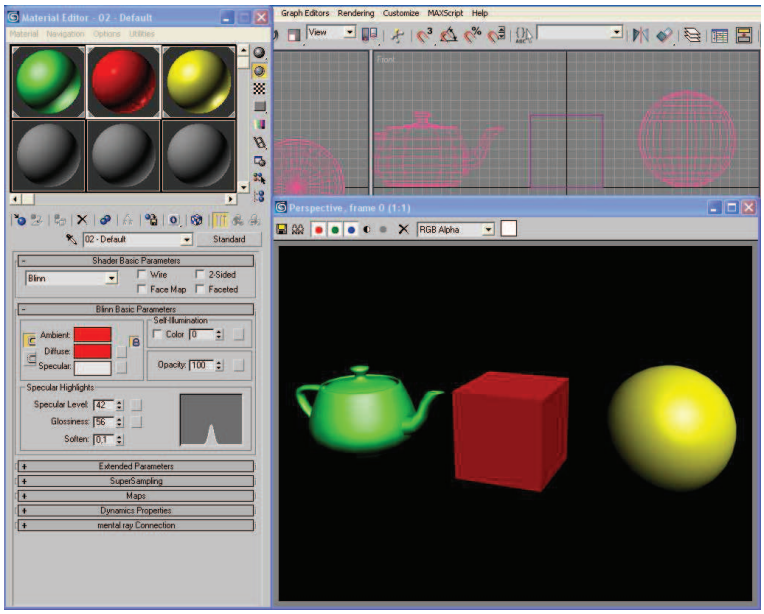


Imagen 5.4.3: Creación de materiales estándar

Alambre

Sirve para mostrar el objeto en modo alámbrico, es decir, mostrando las aristas de sus caras. Como se debe "ver" a través del material, se activará la opción "2 lados". En parámetros extendidos > alambre tenemos más opciones, como la anchura de los alambres,…

2 caras

Esta opción es muy importante activarla cuando estamos creando materiales no del todo opacos. Esa opción hace que se puedan ver las caras que se encuentran al otro lado de un objeto desde el punto de vista visible del espectador y por tanto cuando el material deja ver la parte de atrás de un objeto. Si no lo hacemos así, no conseguiremos realismo en el material.

3. Materiales mapeados:

Los materiales mapeados son los que utilizan imágenes para definir la superficie de un modelo 3D. Estos materiales si los podemos exportar a Director y utilizarlos por lo que este punto es de vital importancia para poder usar objetos realistas en nuestros proyectos.

Dentro de este apartado existen distintos aspectos que hay que abordar como el concepto de mapa, cómo asignar un mapa, tipos de mapas, cómo trabajar con mapas, modificadores de mapas, etc.

Concepto de mapa: los mapas son imágenes que se asignan a un material para modificar sus características básicas de simulación física y respuesta a la luz pero que además se pueden utilizar para simular la modificación de la geometría del objeto como rugosidades, relieves, etc. Las imágenes que admite MAX como mapas suelen ser de los tipos utilizados en Windows como .jpg, .bmp,... Existen dos tipos de mapas: de procedimiento y de proceso.

Los mapas de procedimiento crean una textura pero no a partir de una imagen, sino de cálculos matemáticos. Por un lado no permite resultados tan detallados como una imagen pero por otro lado no se puede nunca llegar a pixelizar la imagen, ya que este material siempre se readaptará a los parámetros de la escena. Este tipo de mapas se utilizan para materiales naturales y un tanto "caóticos" como salpicaduras, maderas con vetas, etc.

Los mapas de proceso se utilizan para definir el material con imágenes, ya sea para modificar sus aspectos básicos como otras características (relieve, opacidad, etc.) Las 12 características

comunes a todos los materiales se llaman Canales de mapas. Todos los mapas se asignan desde la persiana mapas del material. Existen 4 columnas que corresponden a la verificación de asignación, la característica del material, la cantidad y la imagen asignada.

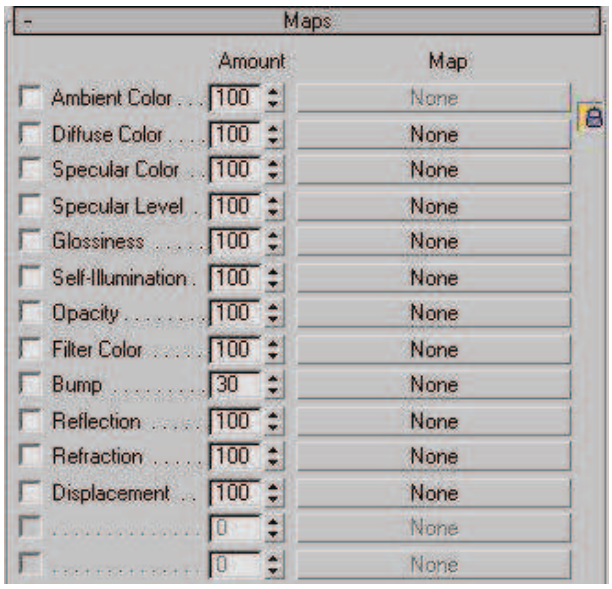


Imagen 5.4.4: Mapas para los materiales estándar

Para modificar las características básicas tenemos: mapa ambiental, mapa difuso y mapa especular. Estos conceptos se explicaron anteriormente. Normalmente solo se modificará el mapa difuso para pintar un material con una imagen. El valor de la cantidad va de 1 a 100, y por tanto, es porcentual.

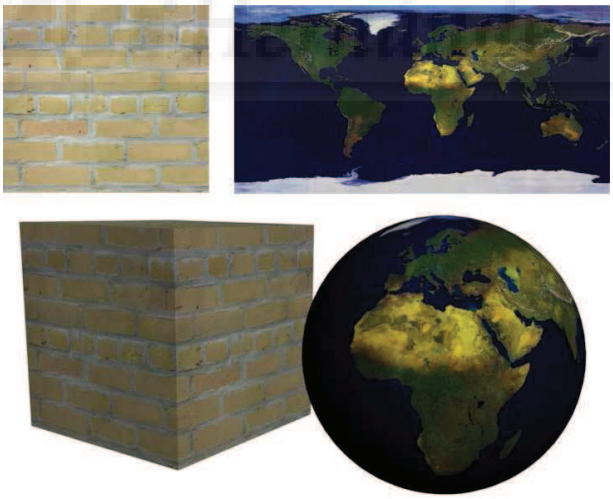


Imagen 5.4.5: Mapas difusos y de relieve y el resultado de su aplicación

Mapas para otras características: el resto de los mapas trabajan de otra manera, no pegan la imagen, sino que extraen de ella, la cantidad de blanco y negro que contienen para crear una imagen de escala de grises, que por tanto es el tipo de imagen que se debería de asignar. Las características que se simulan son:

- Nivel especular: El valor de la cantidad es entre –999 y 999. Esta característica está en relación con la opacidad y sirve para evitar los brillos precisamente en las zonas transparentes, pero sí los mantiene en las zonas opacas.

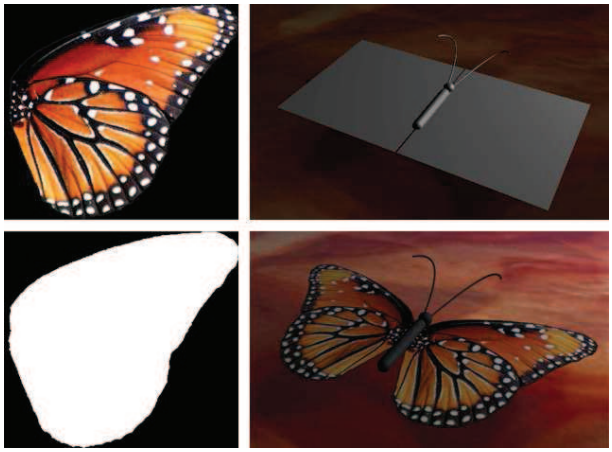


Imagen 5.4.6: Mapa de opacidad y el resultado de su aplicación

◆ Lustre: El valor de la cantidad es porcentual. Ya explicada anteriormente.

◆ Autoiluminación: El valor de la cantidad es porcentual. Ya explicada anteriormente.

◆ Opacidad: El valor de la cantidad es porcentual. Es muy útil tanto para simular materiales transparentes como para superficies planas que necesitan la transparencia en el contorno. Por ejemplo para hacer simulaciones de hojas de plantas, de personas en medios virtuales, etc. En la imagen anterior se puede observar un ejemplo de su aplicación.

◆ Color de filtro: El valor de la cantidad es porcentual. Aplica una imagen al color de tinta producida por un objeto con transparencias. Tanto el objeto como las sombras que proyecta quedan coloreadas con el mapa.

◆ Relieve: El valor de la cantidad es entre –999 y 999. Permite efectos de tridimensionalidad sin modificar la geometría de los objetos. Los colores oscuros del mapa se interpretan como más profundos y los más claros como relieve. Esto se modifica si los valores son negativos. Un problema añadido, es que al proyectar las sombras de un objeto, no se tiene en cuenta este mapa de relieve, por lo que si es muy exagerado su valor, la “trampa” se aprecia a simple vista.

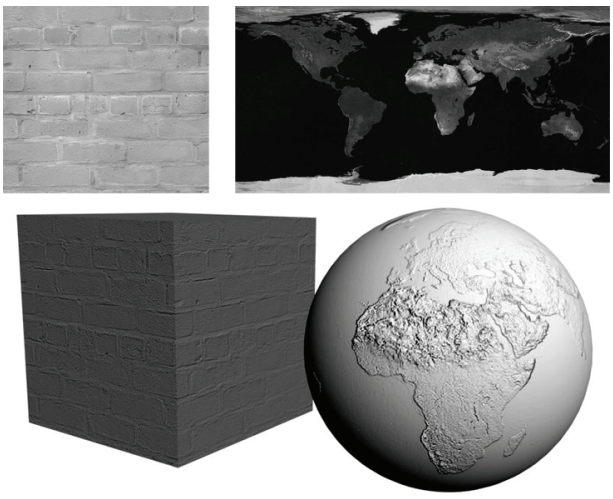


Imagen 5.4.7: Mapas de relieve y el resultado de su aplicación

◆ Reflexión: El valor de la cantidad es porcentual. Para tener reflexión en un objeto se pueden utilizar dos técnicas. La primera se hace “artificialmente”, pegando en el objeto una imagen de lo que debe reflejar. Por ejemplo la misma imagen de un cielo se puede utilizar para que sea reflejado en un estanque. La segunda técnica supone más cálculo para el ordenador en el render. Se trata de que se tenga en cuenta “de verdad” el resto de objetos de la escena y

utilizar un cálculo matemático de reflexión como RayTrace o simetría plana. La ventaja es su mayor realismo.

◆ Refracción: El valor de la cantidad es porcentual. Es como la reflexión pero con el efecto de distorsión que producen los objetos transparentes. Por ejemplo, lo que se observa a través de una botella, se visualiza como “roto”. Se puede utilizar también el Raytrace o la refracción de cristal con las mismas similitudes que las estudiadas para la reflexión.

◆ Desplazamiento: El valor de la cantidad es entre –999 y 999. Es muy similar al modificador Desplazar. Se puede utilizar en todo tipo de geometrías incluida la nurbs. Lo que produce es un cambio en la altura según el mapa aplicado.

Cuando se pulsa un botón para asignar un mapa aparece la ventana Material/Maps Browser. Si se quiere asignar una imagen elegiremos Bitmap, las otras opciones son para crear mapas procedimentales.

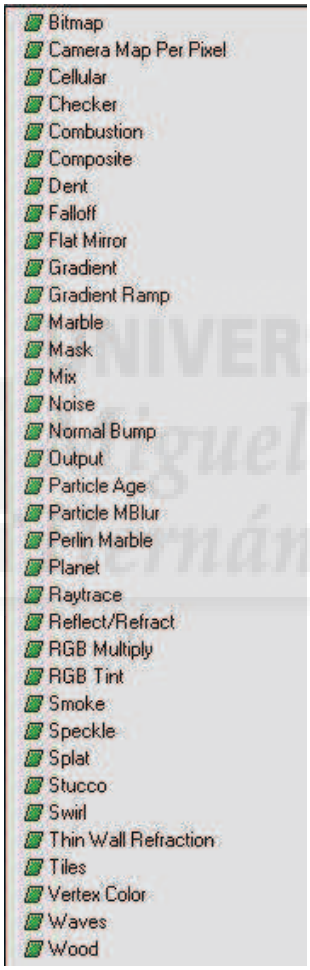


Imagen 5.4.8: Mapas procedimentales

Algunos de ellos son por ejemplo: Gradient, degradado. Gradient Ramp, rampa de degradado. Checker, cuadros. Swirl, remolino. Falloff, atenuación. Cellular, celular. Stucco, estuco. Noise, ruido. Smoke, humo. Marble, mármol. Perlin Marble, mármol perlín, Planet, planeta. Mask, Máscara. Mix, mezcla. Flat mirror, simetría plana. Raytrace, traza de rayos.

Esto quiere decir que si queremos aplicar una textura de humo, podemos utilizar la imagen procedimental Smoke en vez de una foto.

4. Materiales compuestos

Se utilizan materiales compuestos cuando un objeto necesita más de un material. Por ejemplo pensemos en un jarrón de barro, de boca muy ancha y decorado por fuera. Tenemos un único objeto, una única geometría, sin embargo, para poderlo manipularlo en animaciones, no basta con asignarle un material decorado en el exterior del jarrón, sino que se debe texturizar el interior con un material que simule el barro ya que en algunas vistas se verá parte del interior por la boca y no digamos si el jarrón debe romperse a lo largo de la película.

En MAX tenemos muchas posibilidades de materiales compuestos que son los que aparecen en la imagen que sigue y que describimos a continuación.

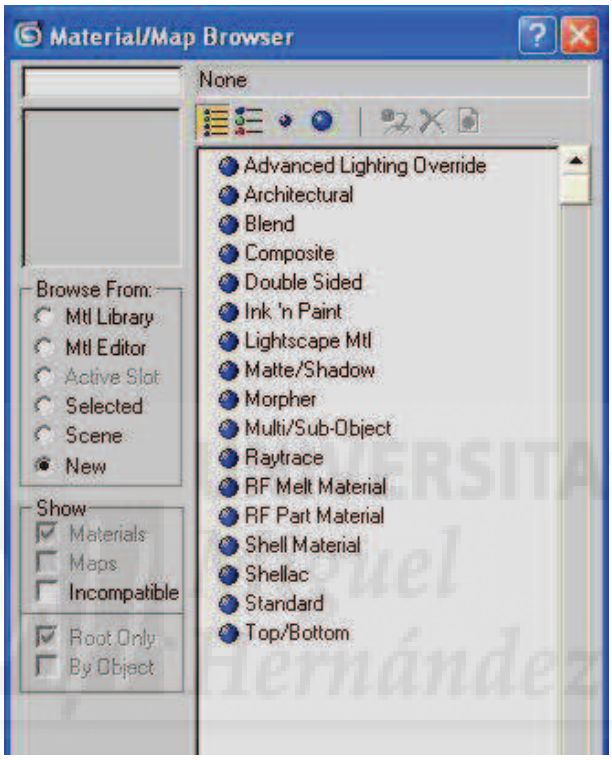


Imagen 5.4.9: Materiales compuestos

◆ Composición Multi / Subobjeto: materiales diferentes en cada cara o grupos de caras. Contiene una lista numerada de materiales que pueden asignarse independientemente a diferentes partes de un objeto mediante la selección de ciertas caras, la asignación de un id (identificador) de material para ellas y luego hacer coincidir ese nº id de material con el de la lista de este material compuesto. El nº máximo es de 1000 submateriales.

◆ Composición Superior / Inferior. Tiene solo 2 submateriales, según la normal de una cara apunte hacia arriba o hacia abajo se le asignará uno u otro material. Se tiene un control llamado “mezcla” para la obtener una fusión más o menos nítida entre los dos materiales. También se puede controlar la posición de la línea de fusión con respecto a la geometría. 50 significa que estará situada en la mitad del objeto.

◆ Composición 2 Lados. Tiene también 2 submateriales, según la normal de una cara apunte hacia el exterior o hacia el interior se le asignará uno u otro material, llamados anterior y posterior. Se tiene un control llamado “translucido” para que se vean simultáneamente en un porcentaje determinado.

◆ Composición Mezcla. También tiene 2 submateriales (material 1 y material 2) que se aplican al objeto a la vez pero según uno de estos criterios:

a) Una máscara. La máscara puede ser una imagen en escala de grises o una curva de mezcla. Si es una imagen hace que el material 1 deje paso al material 2 cuanto más blanca sea una zona. Si se utiliza la curva de mezcla se controlará mediante un gráfico.

b) Cantidad de mezcla. Se mezclaran los dos en un tanto porcentual.

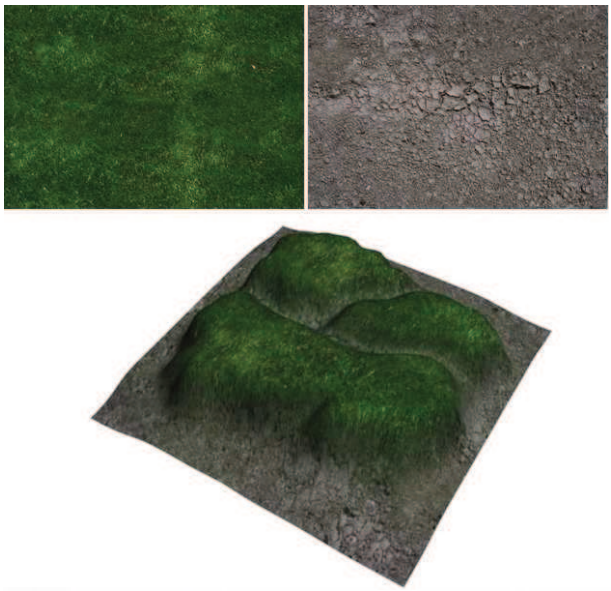


Imagen 5.4.10: Ejemplo de material de tipo mezcla

◆ Composición Compuesto. Es parecido a la opción de mezcla pero permite hasta 10 materiales que se van superponiendo, partiendo siempre de un material base. Cada uno de los restantes se puede mezclar de forma aditiva con respecto a su opacidad, de forma sustractiva o con relación a una cantidad.

◆ Composición Shellac. Shellac significa barniz. Se trata de la composición de dos materiales por la superposición de los colores de ambos. Es un truco muy utilizado en programas de pintura digital para dar aspecto de barnizado a un objeto por ejemplo de madera.

◆ Composición Mate / Sombra: Es un tipo de material muy utilizado para mezclar imágenes reales con imágenes de síntesis. Esto se puede hacer porque el material que se designa como mate no se representa, pero sí que se representa si recibe sombras o reflejos de otros materiales.

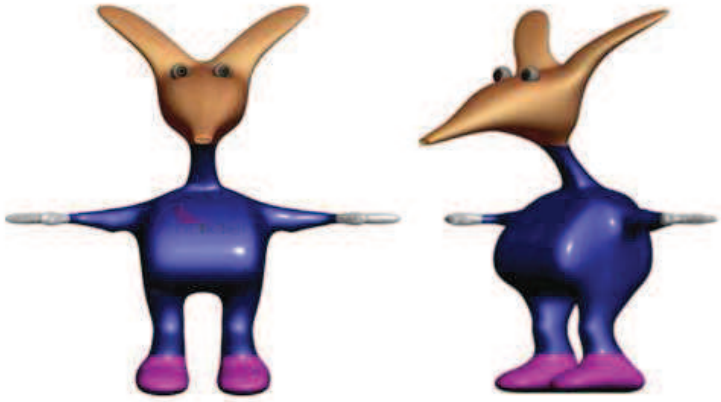


Imagen 5.4.11: Ejemplo de material multi /subobjeto

◆ Material Raytrace. Al igual que el mapa raytrace, se utiliza para conseguir tanto reflexión como refracción de gran calidad. Se utilizan cálculos de trazados de los rayos de luz. Utilizan muchos parámetros básicos que tienen los mismos nombres que el material estándar como ambient y diffuse,… pero que no se utilizan de la misma forma.

Imagen 5.4.12: Ejemplo de material Raytrace

Caso práctico 4.1: El editor de materiales.

Objetivo: Conocer el editor de materiales y el Material/Map Browser para crear materiales con las herramientas propias de MAX.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Crear objetos en escena.
2. Acceder al Editor de Materiales y al Material/Map Browser.
3. Obtener y aplicar los materiales necesarios.
4. Construir una biblioteca propia.

1. Crear objetos en escena.

1.1. Creamos objetos como cajas, esferas y cilindros como base para aplicar los materiales y observar las diferencias al aplicar el mismo material en diferentes geometrías.

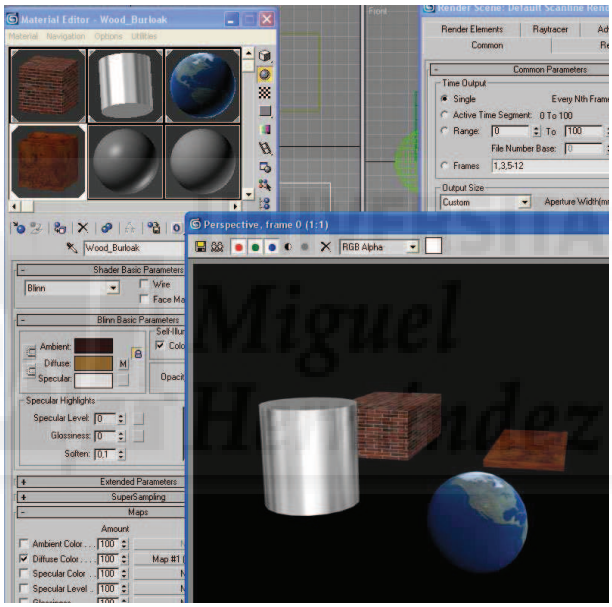


Imagen 5.4.cp.1.1: Los objetos creados para este caso práctico

2. Acceder al Editor de materiales y al Material/Map Browser.

2.1. Para acceder al editor de materiales debemos usar la tecla “M” o pulsar el icono con 4 bolitas de colores de la barra de herramientas estándar. También podemos ejecutar el comando Rendering > material editor.

Aparece la caja de diálogo llamada “Editor de materiales” que centraliza todo el diseño y la gestión de los materiales que pueden designarse a los modelos tridimensionales de la escena, desde la creación hasta la gestión de bibliotecas de materiales propias de cada diseñador 3D.

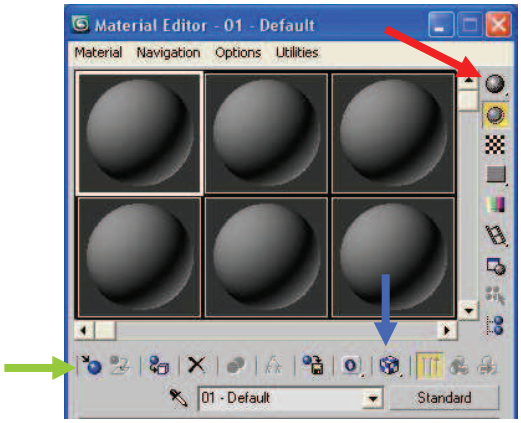


Imagen 5.4.cp.1.2: El Editor de materiales

Los slots (nichos) sirven para crear distintos materiales. Por defecto son esféricos, pero podemos cambiar el Sample Type (que se señala con una flecha roja en la imagen superior) para que se adapte a un cilindro o una caja.

2.2. Pulsamos sobre el botón “Get Material” (señalado en la imagen superior por una flecha de color verde) lo cual nos permite acceder al “Material/Map Browser”.

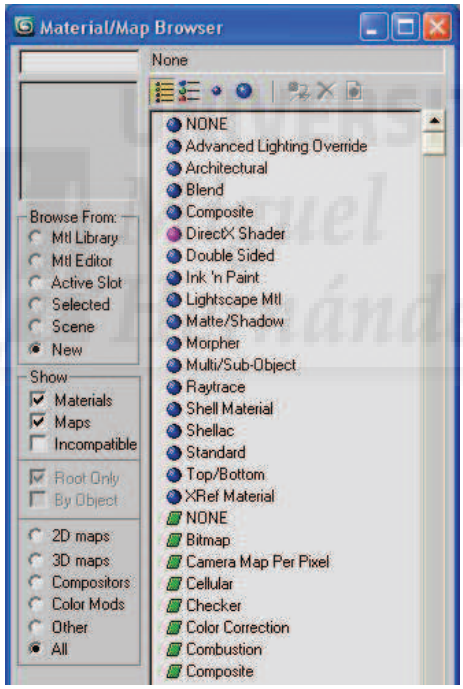


Imagen 5.4.cp.1.3: El Material / Map Browser

El Material/Map Browser permite acceder a los materiales creados y gestionar sus bibliotecas, es decir, agrupaciones de estos por diversos criterios que pueden ser creadas desde esta caja de diálogo.

3. Obtener y aplicar los materiales necesarios.

3.1. En Browser From: elegimos Mtl Library y abajo en File: Open. De esta forma abrimos la librería “3dsmax.mat”.

Esta biblioteca es estándar de MAX en la versión 2009 y desde las versiones iniciales, por lo tanto debería de existir en nuestra instalación de MAX, de todas formas si no es así, no pasa

nada, ya que el mismo procedimiento puede funcionar para cualquier biblioteca de MAX. Cuando se instala 3DS MAX, se crean dos directorios, matlibs y maps para separar los materiales de los mapas.

3.2. Pulsamos (en el grupo Show) sobre “Maps” para desactivar esta opción y por tanto veremos solamente materiales.

3.3. Identificamos el material “Brick_BrickCommon” y hacemos doble clic sobre el. Si volvemos al Material Editor observamos que ha ocupado el primer slot.

3.4. Hacemos clic sobre el 2º Slot y accediendo de nuevo al Material/Map Browser seleccionamos el material “Metal_CromeFast”. Repetir los mismos pasos para “Space_Earth” y “Wood_Burloak”.

Una vez en el editor de materiales los podremos modificar. Como este proceso aún no lo dominamos nos vamos a limitar a aplicarlos tal cual.

3.5. Arrastramos el material “Brick_BrickCommon” hasta la caja con el ratón.

Observaremos que sobre las esquinas del slot aparecen triángulos para indicar que ese material está siendo aplicado en la escena, por lo tanto puede haber varios de ellos con esta característica. El material aplicado a los objetos seleccionados se distingue del resto porque estos triangulitos están en color blanco.

Se debe observar en el visor el material aplicado. Si no es así, pulsamos el botón “Show Map in Viewport” que se puede observar en la imagen “El editor de materiales” señalado por una flecha de color azul.

4. Construir una biblioteca propia.

4.1. Volvemos al Material/Map Browser y pulsamos el botón “Clear Material Library”. Esto hace que la biblioteca actual se quede sin materiales.

¡¡¡¡ No guardar (Save) la biblioteca ahora, ya que perderíamos todos los materiales!!!

4.2. Arrastramos los 4 materiales desde el “Editor de materiales” al “Material/Map Browser”.

4.3. Pulsamos en “Material/Map Browser” sobre File en Save as y le ponemos el nombre que queramos, por ejemplo el del proyecto que estemos realizando ahora mismo.

Esto produce que se cree un archivo nuevo con la extensión “.mat”. Y ya tendríamos una biblioteca construida por nosotros mismos y con los materiales procedentes de otras bibliotecas (lo cual abre la puerta a la mezcla de bibliotecas) o diseñados por nosotros.

Conclusiones:

Debemos saber acceder y gestionar los materiales que creamos y que utilizamos en nuestros proyectos con el fin de ser reutilizados en proyectos futuros o para usar los ya creados por otros diseñadores. Las bibliotecas de materiales son archivos independientes y junto con los mapas asociados podemos reubicarlos y por lo tanto, las podemos utilizar en distintos ordenadores.

Caso práctico 4.2: Creación de superficies básicas.

Objetivo: Conocer los parámetros básicos de los materiales de MAX.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Elegir el sombreador adecuado.
2. Modificar los parámetros básicos.

1. Elegir el sombreador adecuado.

Los materiales pueden ser muy complejos. Vamos a enumerar los parámetros que podemos utilizar para crear materiales básicos, por lo que no se trata de una práctica al uso, sino que vamos a ir probando cómo quedan los valores variándolos y representándolos. Se trata de ir aprendiendo mediante el método de prueba y error.

Con los sombreadores tenemos el problema ya descrito en la unidad teórica 4 que aunque se pueden utilizar hasta 8 sombreadores distintos en MAX, al exportarlos a Shockwave 3D, esta información se pierde ya que en Director solamente tenemos un sombreador básico, por lo que este primer punto solo es útil para presentaciones y creación de imágenes fijas.

Ya enumeramos los sombreadores básicos en la unidad 4, pero a modo de ejemplo diremos que en la imagen de abajo, la tetera utiliza el sombreador Strauss, el cubo utiliza el sombreador Blinn que es el que muestra MAX por defecto, mientras que la esfera se ha realizado a partir del sombreador Multi-layer que produce superficies con dos brillos metálicos.

Imagen 5.4.cp.2.1: Algunos materiales básicos aplicados a modelos 3D

1.1. Accedemos al “Editor de materiales” pulsando la tecla “M”. Tendremos el primer slot seleccionado, por lo tanto los cambios efectuados a partir de aquí se referirán a este material.

1.2. En el control donde aparece las palabras “01. Default” podemos nombrar nuestro material, para ello escribimos directamente “metal verde”.

1.3. En el sombreador elegimos “Strauss” que permite definir metales. Esto hará que los parámetros básicos del material cambien y esto sucede para cada tipo distinto de sombreador. Por ejemplo, en Strauss tenemos los parámetros color, glossiness, metalness y opacity.

2. Modificar los parámetros básicos.

Para no hacer este caso práctico demasiado largo, vamos a estudiar los parámetros del sombreador Blinn que es el que viene por defecto.

Definición de color difuso, ambiente y especular: Se trata de definir mediante RGB un color base para el material, el difuso, otro para la zona de sombras, el ambiente y otro color para la zona de más brillo, el especular.

2.1. Pulsamos sobre el botón a la derecha de Ambient y en la caja de diálogo elegimos un color amarillo canario. Repetimos la operación para el color diffuse. El brillo lo dejamos en color blanco.

El efecto es el que se visualiza en la siguiente imagen. El amarillo de la esfera parece más bien plástico debido a que aunque el color del brillo (color especular) está definido, no tenemos definido brillo en sí mismo (Specular level = 0).

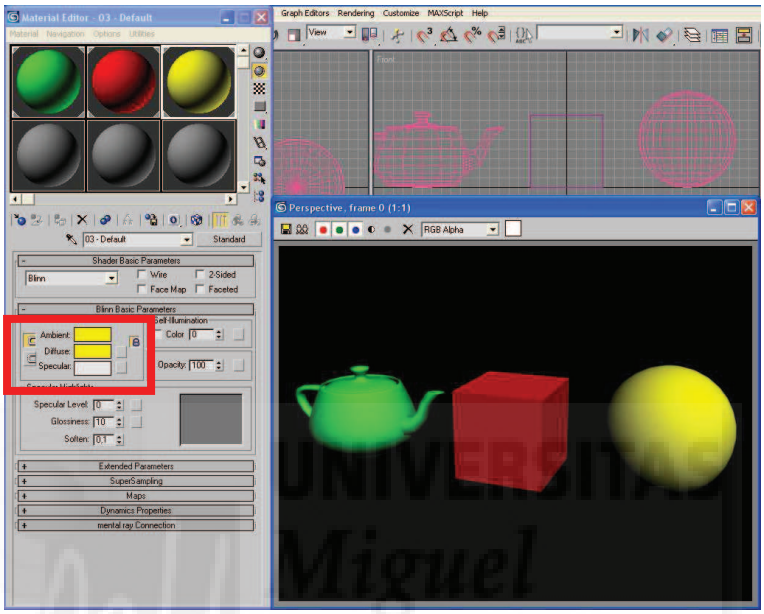


Imagen 5.4.cp.2.2: Los tres colores básicos del material

Definición del brillo resalte: el brillo se basa en dos parámetros: el nivel especular (Specular level) y en el lustre (Glossiness).

El nivel especular define la intensidad del brillo. Si se aplica con más o menos intensidad. Puede tener un valor cualquiera.

El lustre define lo grande que es el brillo, es decir, si se aplica a una superficie más o menos amplia. Su valor va de 0 (más amplio y difuminado) a 100 (más intenso y definido en los bordes).

2.2. Accedemos a los controles comentados y ponemos los valores que se aprecian en la siguiente imagen: Specular level a 42, Glosiness a 56 y Soften a 0,1. En la imagen se puede apreciar en el material rojo como aparecen pequeños brillos que podemos modificar.

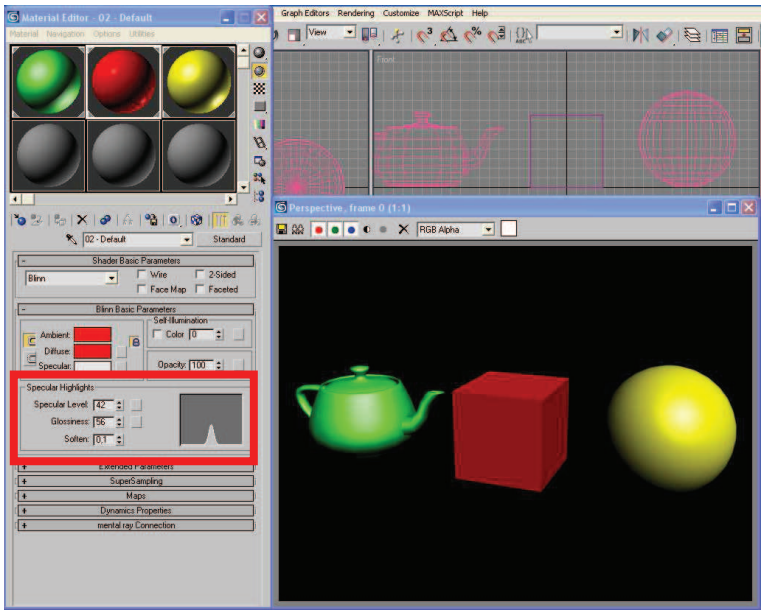


Imagen 5.4.cp.2.3: Definiendo el brillo: su intensidad y su lustre

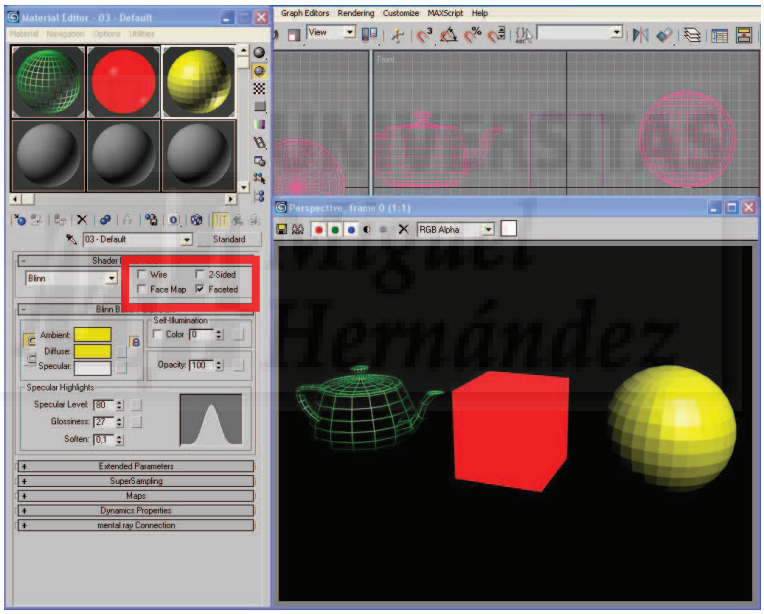


Imagen 5.4.cp.2.4: Modos de material

2.3. Accedemos a los modos del material y probamos en los distintos materiales algunos de los modos como el wire (alámbrico) que solo representa aristas o el facetado que produce un color no gradual entre las caras del modelo 3D.

Para trabajar la opacidad de los materiales tendremos que utilizar dos parámetros: “2 lados” y opacidad. El valor “2 lados” no tiene sentido si la superficie es opaca ya que hace que se vean las caras y aristas posteriores (las que normalmente no se ven) de los modelos y la Opacidad que es definida de forma porcentual.

2.4. Ponemos la opacidad del material rojo a 50% y la característica “2 lados” activa. Repetir la operación con el material amarillo. El resultado se puede apreciar en la siguiente imagen donde tanto el cubo como la esfera son semitransparentes y por eso se ve a través de ellos.

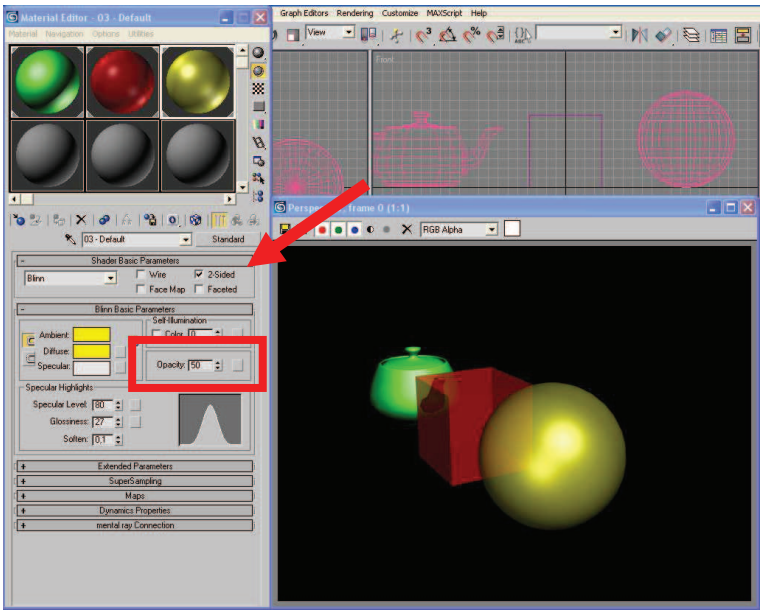


Imagen 5.4.cp.2.5: Definiendo la transparencia

La característica de “self-ilumination” permite definir un color y una cantidad para que el material parezca que emite luz. Esta característica se puede usar en una material para utilizar en bombillas, letreros luminosos, etc.

Conclusiones:

Los materiales básicos son fáciles de utilizar en MAX pero dependen totalmente del sombreador utilizado como base para definir su naturaleza. Como estos sombreadores no los podemos utilizar en Director, gran parte de nuestro trabajo lo perderemos al exportar nuestro proyecto. De todas formas, está bien estudiar estas características para así tener conocimientos sobre materiales e intentar imitarlos en Director con Lingo si es necesario como veremos en posteriores casos prácticos.

Caso práctico 4.3: Aplicación de mapas difusos.

Objetivo: Aplicar imágenes para cubrir las superficies de modelos 3D con el fin de conseguir una apariencia más realista.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Crear los modelos y los mapas adecuados.
2. Aplicar los mapas difusos.

1. Crear los modelos y los mapas adecuados.

Los mapas difusos son normalmente imágenes conseguidas directamente de la realidad haciendo fotos y luego tratándolas con PhotoShop. En la siguiente imagen se presentan dos mapas difusos conseguidos por Internet.

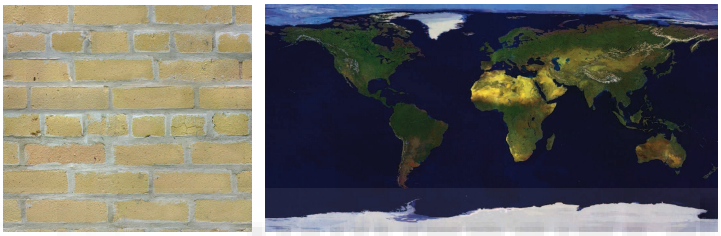


Imagen 5.4.cp.3.1: Ejemplos de mapas de color difusos

Vamos a crear dos modelos: una esfera y un cubo para aplicar directamente los mapas a modelos adecuados a ellos. No vamos a ver cómo mapear, es decir, como se aplican los mapas sobre la superficie del objeto, sino en qué consiste un mapa de este tipo. El mapeo es un tema complicado que escapa al objetivo del presente curso, por ello utilizaremos modelos que son primitivas básicas como cajas y esferas. Seguiremos los siguientes pasos.

1.1. Creamos un cubo de 75 unidades de lado y lo colocamos en el centro de la escena.

1.2. Aplicarle un color gris al cubo.

1.3. Crear una esfera de 35 unidades de radio y la colocamos al lado del cubo.

1.4. Aplicarle un color blanco a la esfera.

1.5. Representar la escena pulsando F10.

2. Añadir los mapas difusos.

Abriremos el “Editor de materiales” pulsando la tecla “M”.

2.1. Pulsar sobre la persiana Maps.

2.2. Activar “Diffuse Color”. Mantener el valor a 100.

2.3. Pulsar sobre “None”. Esto abrirá una caja de diálogo para asignar un tipo de mapa.

2.4. Elegimos la opción Bitmap. Esto crea un segundo nivel en nuestro material.

2.5. Elegimos “brick yellow.jpg” y pulsamos “Aceptar”.

2.6. Asignamos el material arrastrándolo directamente sobre el cubo de la escena.

2.7. Repetimos la operación con la esfera y el mapa “tierra1.jpg”.

2.8. Asignamos el material arrastrándolo directamente sobre la esfera de la escena.

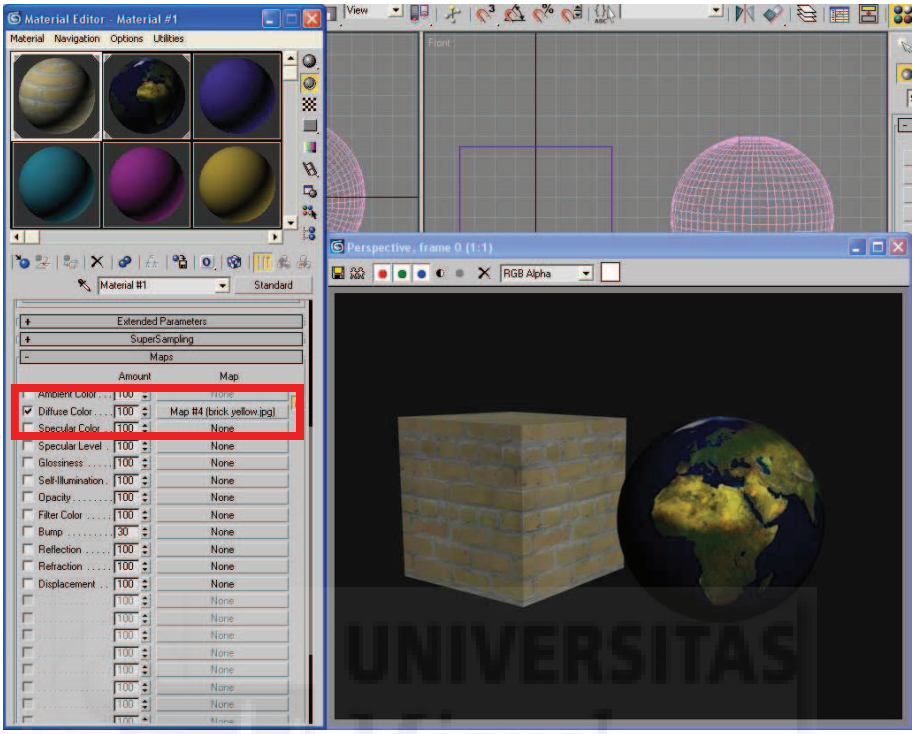


Imagen 5.4.cp.3.2: Ventana del “Editor de Materiales”

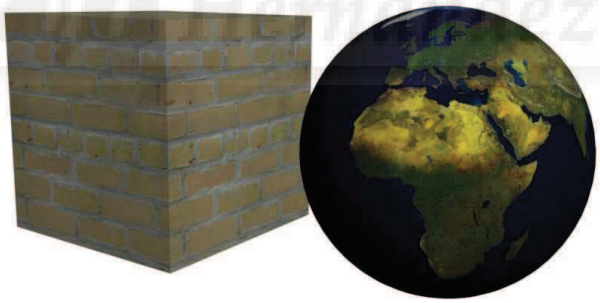


Imagen 5.4.cp.3.3: Resultado final de aplicar mapas de color difuso

Conclusiones:

Los materiales de color difuso son imágenes que son muy fáciles de conseguir mediante fotografías o en Internet. También son sencillas de aplicar sobre todo a modelos geométricos como son las primitivas. El resultado final es muy realista y además los podemos exportar y utilizar en Director sin problemas como estudiaremos más adelante.

La aplicación de mapas difusos no implica que no se puedan utilizar otros parámetros y modos como alambre, opacidad, etc.

Unidad 5: Iluminación en MAX.

Introducción teórica:

1. Introducción a la iluminación en MAX.
2. Tipos de luces.
3. Parámetros de las luces.

Casos prácticos:

5.1. Crear distintos tipos de luces.

1. Introducción a la iluminación en MAX

La iluminación es una parte muy importante en cualquier trabajo 3D ya que sin luces, no se visualizaría ningún objeto en la escena. Por otra parte es algo difícil, al fin y al cabo ¿Quien conoce conceptos como omni, lúmenes y raytrace? Podemos afirmar, que en la mayoría de los casos, la cultura general que tenemos de la naturaleza y la forma en que funciona la física de la iluminación es bastante baja por no decir nula. En conclusión, nos vamos a adentrar en un tema complicado pero al que no podemos renunciar. Lo primero que tenemos que tener claro es que la mayoría de los actuales programas no iluminan los mundos virtuales de manera "real". Aunque se conoce el método de la propagación de la luz, es demasiado costoso desde el punto de vista de cálculo como para implementarlo.

Lo que se realiza es una simulación de la iluminación real que tiene un coste computacional asumible. Esta simulación se basa en el ángulo entre la fuente de luz (en realidad, la recta que define su dirección) y cada cara de la geometría a iluminar (su normal). O sea, que las caras que "miren" más hacia la luz, quedarán más iluminadas que las caras más laterales. Esta simulación obtiene resultados sorprendentemente realistas.

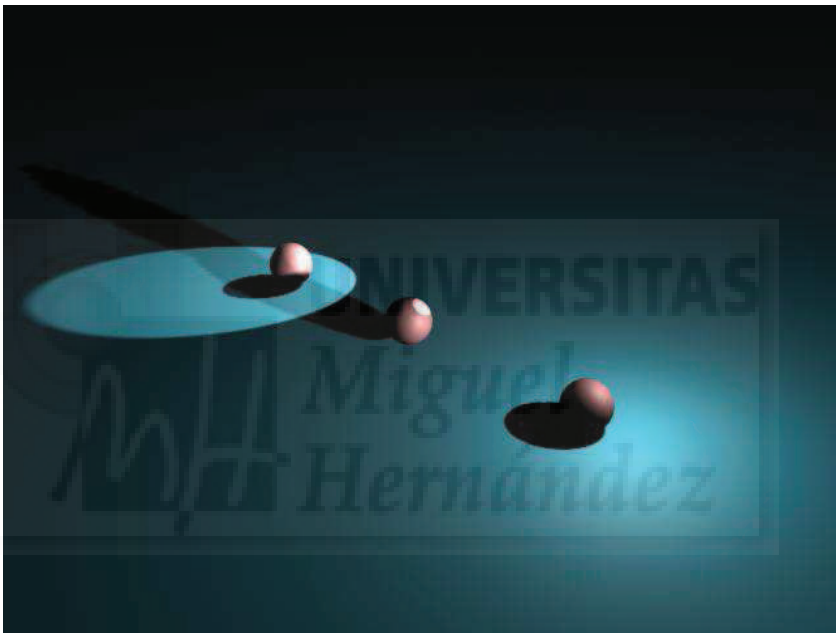


Imagen 5.5.1: Ejemplo de iluminación en MAX

Existen variables en la creación de la iluminación que podemos utilizar o no dependiendo de si las vamos a utilizar en una plataforma en tiempo real o no. Uno de estos parámetros es el de las sombras. Si vamos a realizar un trabajo para su salida en vídeo, podemos utilizar iluminación realista con sombras, raytrace y otras técnicas, pero si la salida del trabajo en MAX es una plataforma como Director debemos de inhabilitar la proyección de sombras de las luces que introduzcamos en la escena y no utilizar raytrace (trazado de rayos de luz). Para realizar una buena iluminación de la escena tenemos que tener en cuenta que hay que pensar siempre en cómo funciona el programa, no en la realidad. No hay un método infalible para iluminar bien, ya que depende de cada escena. Con una iluminación determinada un mismo personaje puede parecer siniestro o bondadoso, por lo que algunos autores que dominan las técnicas de iluminación la utilizan casi para modelar, dando a la escena matices y atmósferas difíciles de conseguir con otros medios.

2. Tipos de luces

En Max se pueden reproducir todos los efectos lumínicos que se dan en la vida real como luces volumétricas, resplandores, faros, etc.

La iluminación tiene dos posibles orígenes: los 4 tipos de luces y los efectos del módulo de postproducción. Este último módulo no es objeto de este curso, ya que sus resultados sólo son visibles en plataformas que no sean de tiempo real como vídeos e imágenes por lo que no lo podemos exportar a Director y por tanto no nos interesa su estudio. Las luces en 3D Studio MAX son de 4 tipos distintos: ambiental, omnidireccionales, direccionales y focos.

Luz Ambiental.

Por defecto es de color negro y tiene efecto en toda la escena. Se consiguen los mejores contrastes cuanto más oscura sea. Su efecto es como obtener una escena y a posteriori colorearla totalmente con una película de color. Se puede utilizar, por ejemplo, para una escena submarina con un color azulado o una escena en Marte, con un color rojizo. La luz ambiental se gestiona desde el menú Representación > Ambiente. Tiene 2 parámetros básicos:

a) Tinta: color que afectará a toda la escena.

b) Nivel: el factor de multiplicación de la intensidad de todas las luces de escena. Se utiliza para la corrección rápida de una escena con mucha o poca luz general.

Las luces que siguen se crean desde el panel Crear > Luces y se gestionan desde el panel de modificar. Son objetos luminosos que pueden ser transformados por desplazamiento, rotación, escalado y/o animación.

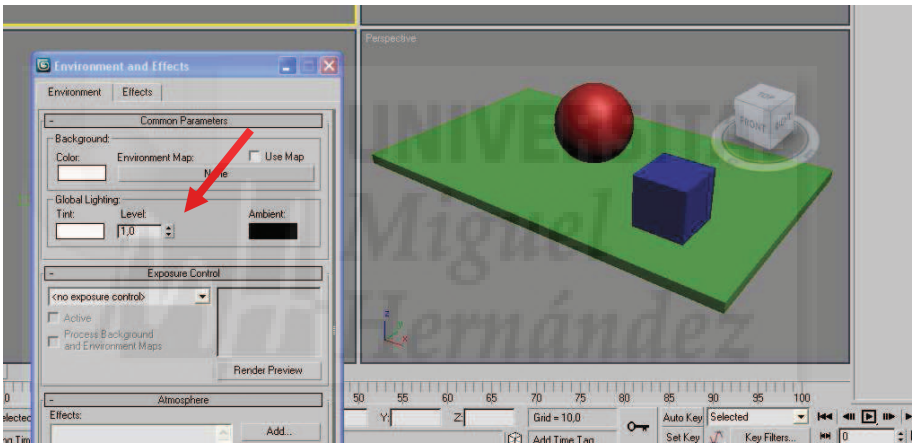


Imagen 5.5.2: Luz ambiental para la escena 3D

En la imagen anterior se puede observar dónde se encuentran los controles de la luz ambiental y en imagen siguiente se aprecia cómo cambia toda la escena si la iluminamos con luz ambiental negra (a la izquierda) o blanca (a la derecha).

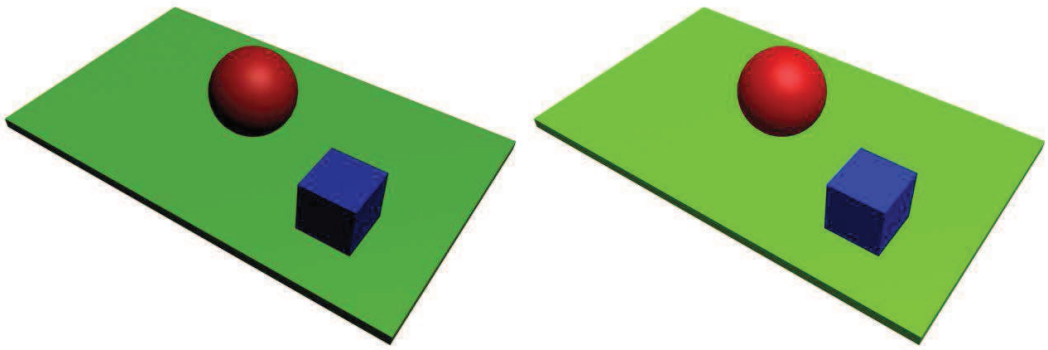


Imagen 5.5.3: La misma escena con luz ambiental negra y blanca

Luces omnidireccionales u omni.

Son puntos lumínicos que irradian luz en todas las direcciones, como por ejemplo una bombilla, una vela, etc. Se utilizan mucho en combinación con otras luces, pero hay que tener cuidado ya que producen múltiples proyecciones de sombras. Los círculos concéntricos que rodean estas sombras definen su atenuación.

En la imagen inferior se puede ver a la izquierda la escena en la vista de perspectiva y la situación de la luz de tipo omni. A la derecha se puede ver la misma vista pero una vez que se ha representado.

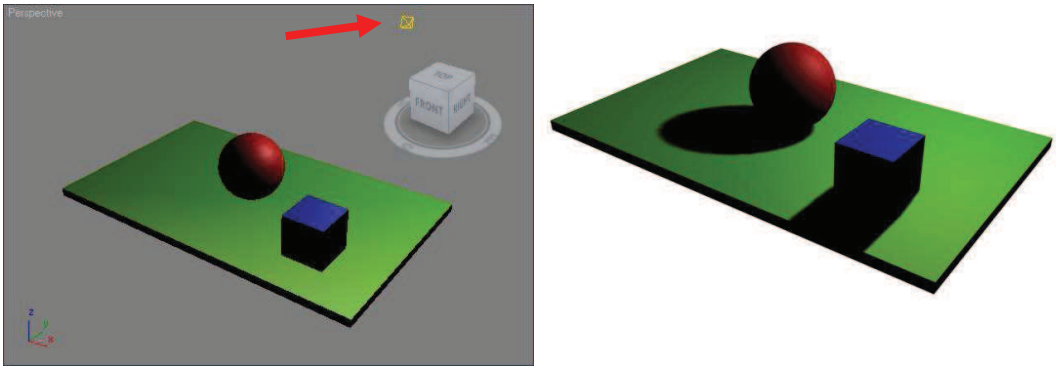


Imagen 5.5.4: Luz Omni y su resultado al renderizar la escena

Luces direccionales.

Son puntos lumínicos que irradian luz en una sola dirección, todos los rayos de luz son paralelos y circunscritos en el cilindro de proyección (aunque esto se puede modificar con la opción Rebasar). El efecto sería como la luz solar, que aunque sabemos que es omnidireccional, dada la gran distancia que nos separa la podemos considerar como direccional. También podríamos usarla para simular un rayo láser.

En la imagen siguiente se puede ver a la izquierda la escena en la vista de perspectiva y la situación de la luz de tipo direccional y su target (punto objetivo) situado sobre la superficie misma de la esfera roja. A la derecha se puede ver la misma vista pero una vez que se ha representado.

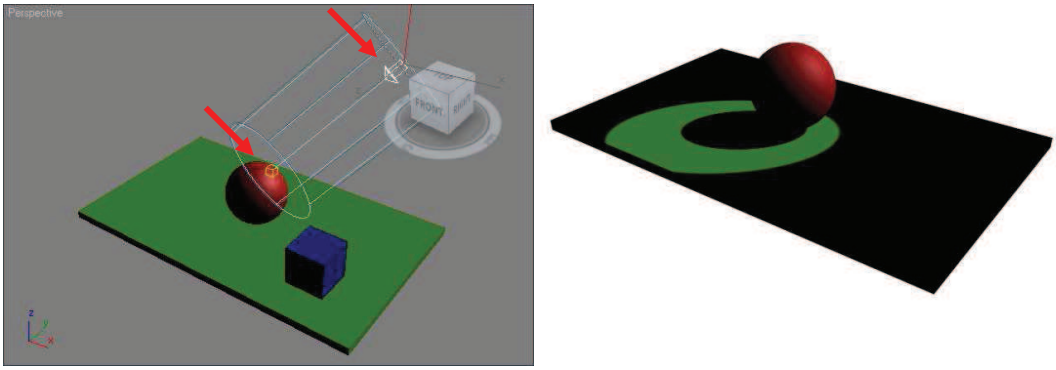


Imagen 5.5.5: Luz Direccional y su resultado al renderizar la escena

Luces focos

Son puntos lumínicos que irradian luz en forma de cono. Aparece un cono interior y otro exterior que definen las zonas de mayor luminosidad y la de atenuación. Son luces focales, por ejemplo, sirven para iluminar a modo de linterna o flexo.

En la imagen inferior se puede ver a la izquierda la escena en la vista de perspectiva y la situación de la luz de tipo focal y del target sobre la esfera roja. A la derecha se observa la misma vista pero una vez que se ha representado. La diferencia con la direccional es que el cono de luz cuando se proyecta sobre el suelo produce una superficie elíptica, no circular.

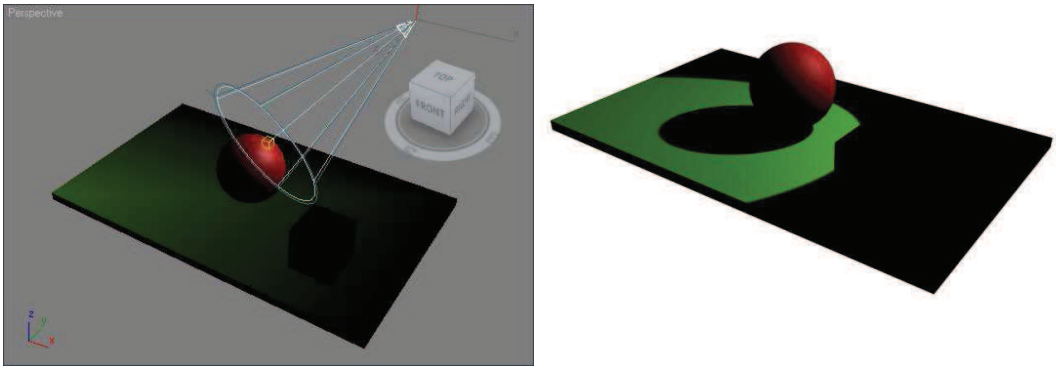


Imagen 5.5.6: Luz Foco y su resultado al renderizar la escena

Otra posible clasificación de las luces es según el objetivo o target, existen dos tipos:

Luces libres

Existen dos de estas luces en MAX, el direccional libre y el foco libre. Tiene un solo elemento: el propio punto luminoso. Se suelen utilizar para luces con una trayectoria de movimiento fija (por ejemplo los faros de un coche) y normalmente se vinculan a un objeto de escena (el coche).

Luces con objetivo

Tienen dos elementos, el punto luminoso y el punto objetivo. Son dos elementos independientes y por tanto se pueden transformar y vincularse a otros objetos por separado.

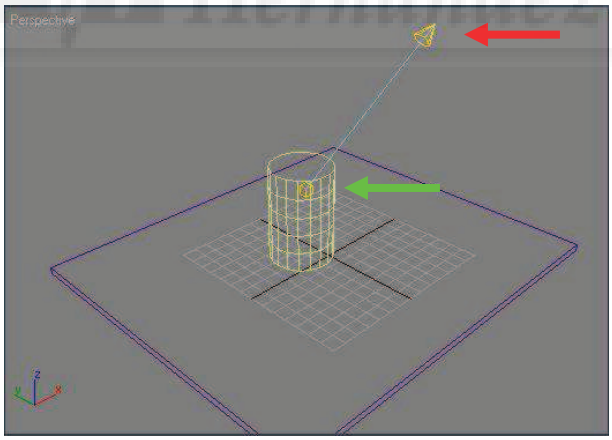


Imagen 5.5.7: Luces con objetivo

En la imagen superior se puede observar una vista de MAX con una luz con objetivo donde la fuente luz está señalada por una flecha rojo y el objetivo por una flecha verde.

Las sombras y cómo se proyectan es una cuestión fundamental para obtener buenos resultados. Hay 2 métodos para implementarlos:

1. Mapas de sombras: se utilizan para objetos totalmente opacos. Se basa en calcular las sombras de la escena e ir añadiendo estas sombras a los materiales de los objetos. Las ventajas son la rapidez de cálculo y las sombras difuminadas realistas que se consiguen.

2. Ray Tracing: se utilizan para objetos no totalmente opacos, semitransparentes. Se basa en calcular los rayos de luz sobre cada punto de la escena. Las desventajas son la lentitud del cálculo y las sombras no realistas por tener bordes totalmente definidos.

3. Parámetros de las luces.

Las luces se gestionan desde el panel Modificar y todas comparten 3 persianas básicas: parámetros generales, parámetros de atenuación y parámetros de sombra, aunque también tienen persianas específicas para cada tipo de luces.

Parámetros generales.

- Tipo: define el tipo básico de luz: omni, spot, etc.

- Sombras: se puede activar o no que una determinada luz proyecte sombras.

- Excluir: hace que podamos excluir de la iluminación de esta luz a determinados objetos. No se basa en la realidad pero puede ser útil en muchas ocasiones. Se puede excluir la iluminación de las caras de un objeto, la proyección de sombras o ambas características.

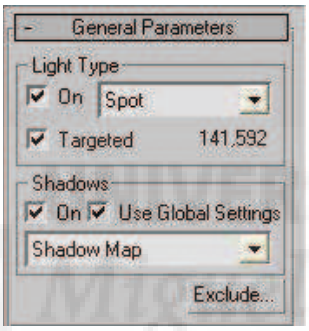


Imagen 5.5.8: Parámetros generales de las luces

Parámetros de atenuación.

- Multiplicador: factor de multiplicación de la intensidad. A veces se utiliza un multiplicador negativo para quitar luz y poder oscurecer zonas que por medio del modelado no se pueden realizar bien, por ejemplo, los agujeros de la nariz en un rostro humano.

- Color de luz: por defecto es blanca. La atenuación responde a un hecho que se da en la realidad: las luces pierden intensidad por la distancia debido a la atmósfera. MAX define la atenuación mediante dos alternativas: lejanía o caída.

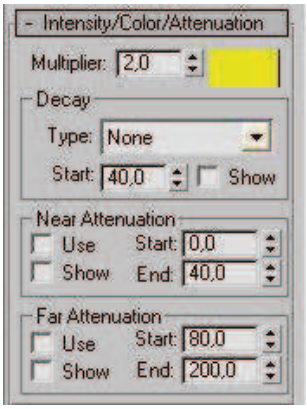


Imagen 5.5.9: Parámetros de la intensidad, color y atenuación

◆ Caída: se define mediante un tipo de caída y un comienzo de esa caída. Su opción Cuadrado Inverso crea una simulación muy real pero que a veces resulta demasiado oscura.

◆ Lejanía: utiliza dos parámetros, rango inicial, donde la luz mantiene una intensidad uniforme y rango final que define hasta dónde llega la luz, de un valor a otro la luz se atenúa.

Parámetros de sombra.

Hay que tener en cuenta que se tiene que activar la proyección de sombras correspondiente a la luz en cuestión. Si se quiere gestionar las sombras de todas las luces a la vez, se debe utilizar la opción “Usar configuración global”. Toda luz tiene los siguientes parámetros de sombra:

◆ Color: color proyectado de la sombra.

◆ Densidad: cantidad de luminosidad de la sombra.

◆ Mapa: pintura aplicada a la sombra. Hay que distinguir esta opción de la proyección de mapas, que sería como ponerle un filtro de celofán a un foco.

◆ Luz afecta al color de sombras: mezcla del color de la luz con el de la sombra.

◆ Sombras atmosféricas: definen el grado de opacidad y cantidad de color que en el caso de existir un efecto atmosférico, influye en la sombra.

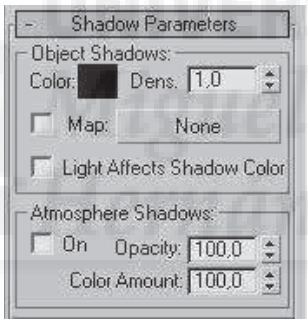


Imagen 5.5.10: Parámetros de sombras

Debemos tener en cuenta que en MAX podemos estudiar la iluminación de tipo fotorealista y atacar cuestiones tales como la incidencia de la iluminación de la escena con los materiales autoiluminados, transparentes y/o con sombras Raytrace.

El control de los mapas de sombras o de Raytrace solo afecta a la distancia a la que se genera la sombra con respecto al objeto. También podemos fijarnos en efectos con el de luces volumétricas. De todas formas, lo más importante es la generación y colocación adecuada de las luces en escena.

Pero como hemos escrito antes, estos temas no nos competen al no soportarlas Director ni ninguna plataforma de tiempo real hoy en día.

En las representaciones que siguen se puede distinguir una escena sin luces, seguida de una escena con una luz direccional. La siguiente demuestra el cambio con el color de la luz a amarillo y la última con la activación de la proyección de sombras.

Como conclusiones a este tema, podemos escribir que la iluminación no la vamos a utilizar en Director con todas sus posibilidades que presenta en MAX aunque no está mal conocerlas sobre todo para hacer presentaciones de la escena en alta definición y fotorealistas, aunque después perdamos estas características a cambio de la interactividad que ofrece Director.

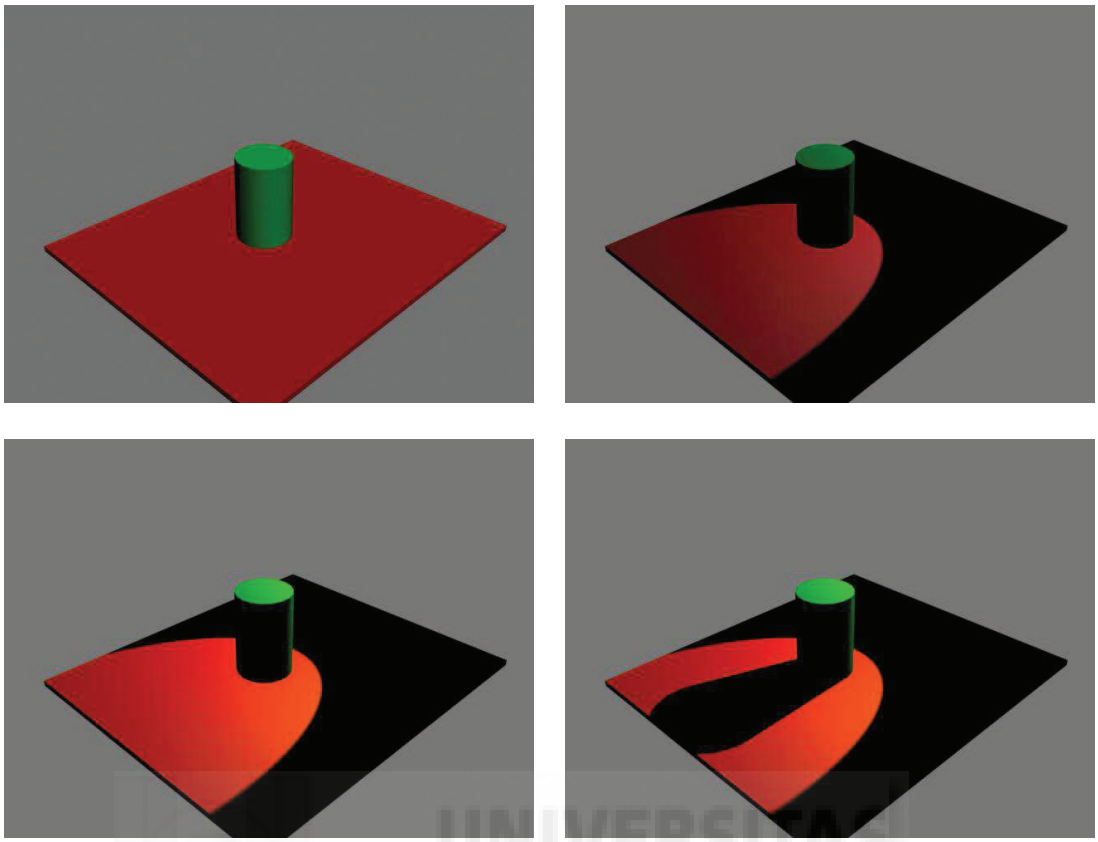


Imagen 5.5.11: Efectos de modificar las sombras

Caso práctico 5.1: Crear distintos tipos de luces

Objetivo: Conocer el método de creación de todos los tipos de luces que incluye MAX.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Crear la escena a iluminar.
2. Crear una luz que arroje sombras.
3. Cambiar el tipo de luz.

1. Crear la escena a iluminar.

Como la iluminación es un tema que interviene por completo en el modo en que se visualiza la escena 3D, tenemos que construir unos modelos que se adapten bien al propósito de la explicación.

Para ello, creamos una escena con 3 modelos: una base que haremos con una caja que será donde se proyecten las sombras y los modelos a iluminar que serán una esfera roja y una caja azul. Como esto no entraña ninguna dificultad no lo detallamos. La escena propuesta queda como se puede observar en la imagen inferior en la esquina superior izquierda.

Podemos observar que aunque no exista luz, cuando se hace un render de la escena se visualizan los modelos, por lo que se puede deducir que MAX si no existe ninguna luz, asigna una iluminación estándar a la escena antes de renderizar.

Este tipo de luz no creada explícitamente por el autor, se llama luz ambiental y está suficientemente explicada en la unidad teórica correspondiente.

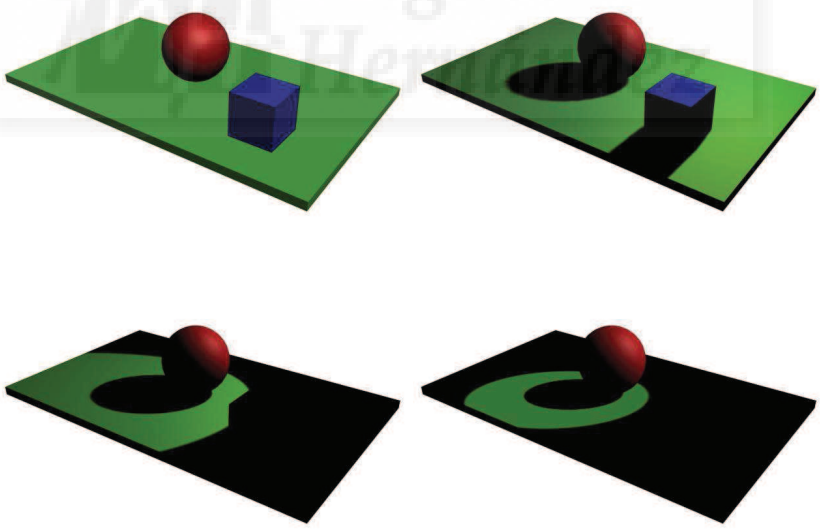


Imagen 5.5.cp.1.1: Distintos tipos de iluminaciones en MAX

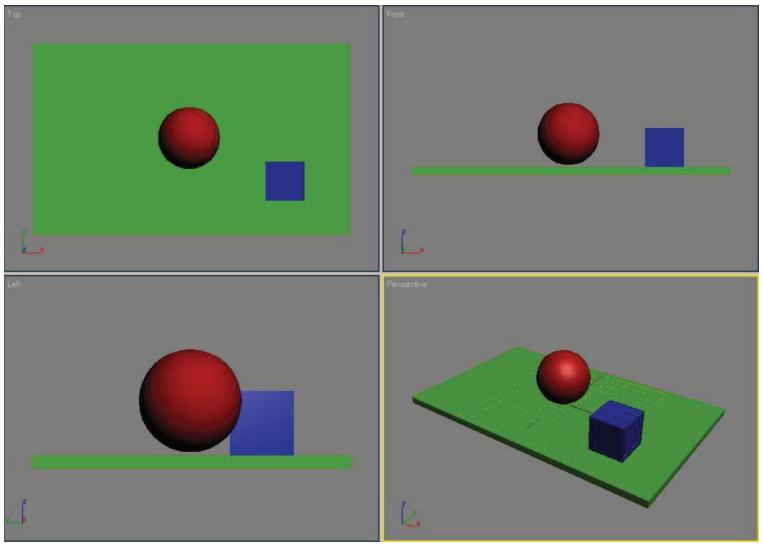


Imagen 5.5.cp.1.2: La escena 3D creada como ejemplo

2. Crear una luz tipo “Target Spot”

Vamos a crear una luz de tipo focal o como se conoce en MAX, tipo spot.

2.1. Accedemos al panel Create > Lights y pulsamos en el botón “Target Spot”.

2.2. Pulsamos sobre el visor Top y arrastramos el cursor hacia el centro del visor.

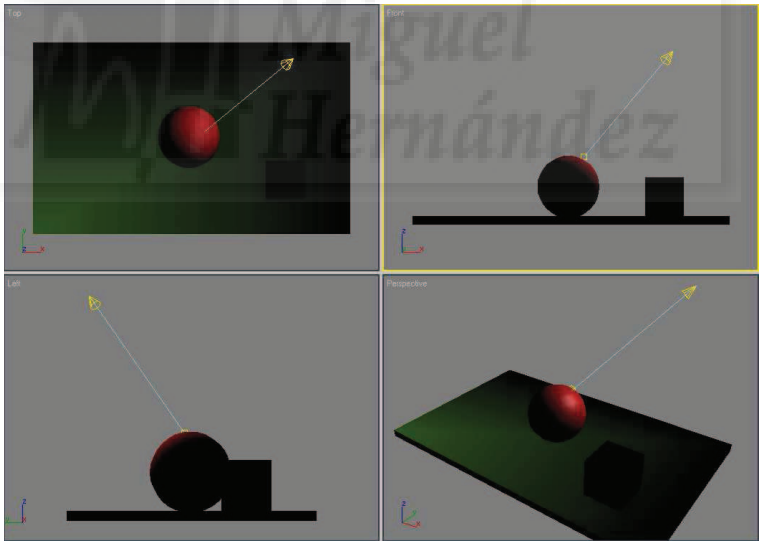


Imagen 5.5.cp.1.3: La escena 3D iluminada con una luz spot

Observaremos como hemos creado una luz de tipo “punto de luz”, que tiene una forma cónica y llega hasta un punto objetivo (target) situado sobre la esfera roja.

2.3. Con la herramienta “Select and Move” podemos mover tanto el icono que representa el punto emisor de luz como al punto target hasta situarlos tal como se observa en la imagen.

Si renderizamos (representamos) la escena observaremos que aunque se visualiza el efecto de la luz, no se ve ninguna sombra.

2.4. Con la luz seleccionada ir al panel Modify y en los parámetros activar “shadows”.

Volver a renderizar la escena para ver las diferencias. Esta representación se puede observar en la imagen inicial en la esquina inferior izquierda. Este tipo de luz es un punto y sus haces de luz se sitúan dentro de un cono. Al ser cónica se puede observar el pico de la caja, que aunque no llega a estar iluminada desde el punto de vista elegido, se supone que si lo es desde un punto de vista distinto.

3. Cambiar el tipo de luz a Direccional.

3.1. Seleccionar la luz e ir al panel Modify y en Type poner “Directional”.

Observamos que el punto de luz y el punto objetivo no cambian de lugar, pero lo que si se modifica muy evidentemente es el modo de la luz, ya que el haz de luces cambia de cónico a cilíndrico, ya que esta luz imita las luces de los focos de los escenarios.

A poca distancia no se observa mucha diferencia con la luz tipo Spot, pero cuanto aumenta la distancia entre el punto emisor de luz y el punto objetivo se puede observar que la luz direccional y la spot son luces muy distintas. Por eso se utilizan esta luz para generar una imitación de luz solar con una distancia mucho mayor, ya que los rayos del sol debido a su gran distancia a la tierra nos llegan prácticamente paralelos, como los rayos de este tipo de luz.

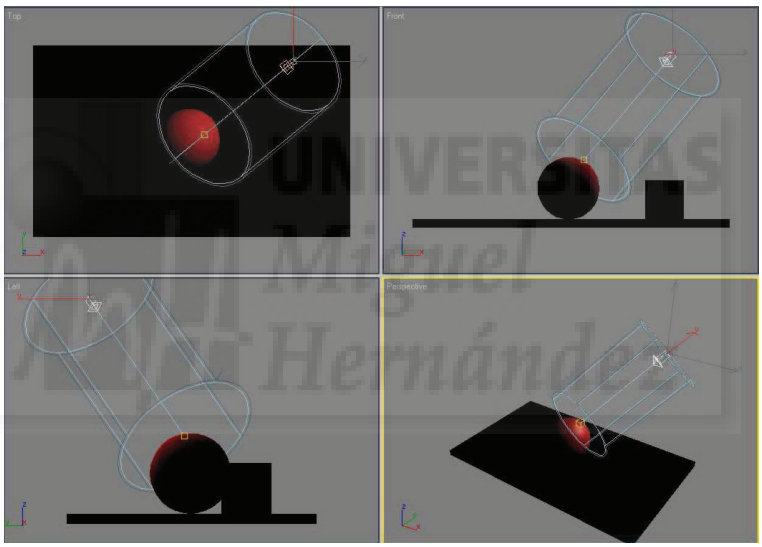


Imagen 5.5.cp.1.4: La escena 3D iluminada con una luz Direccional

La renderización se puede observar en la imagen inicial en la esquina inferior derecha. Al ser cilíndrica se puede visualizar que ya no aparece el pico de la caja.

4. Cambiar el tipo de luz a Omni.

4.1. Seleccionar la luz direccional e ir al panel Modify y en Type poner “Omni”.

Observamos que este tipo de luz se representa como una primitiva tipo “Hedra”. Si representamos la escena observaremos como la luz se emite desde un punto y en todas direcciones como si fuera la luz emitida por una bombilla.

El resultado de esta representación se puede ver en el render de la parte superior derecha donde no se aprecia ningún dibujo de atenuación, ya que los rayos luz emitidos no están circunscritos a un cilindro como en el caso de la luz direccional o a un cono como en el caso de la luz spot y por eso no se visualiza ninguna zona oscura en el suelo verde.

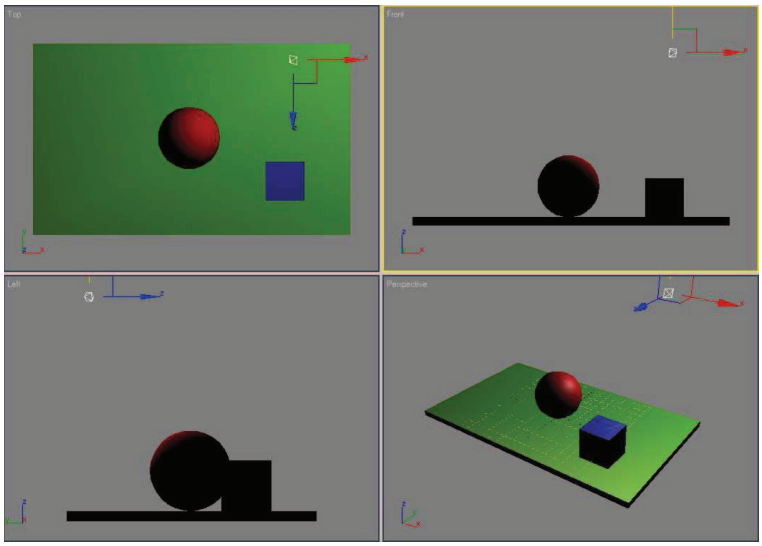


Imagen 5.5.cp.1.5: La escena 3D iluminada con una luz Omni

Conclusiones:

Estos tipos de luces bien utilizados nos proporcionan unas herramientas muy interesantes a la hora de crear escenas realistas. Bien es verdad, y como sucede con otros recursos que vemos en MAX, que no podemos exportarlas en las mismas condiciones de uso a plataformas de tiempo real como Director.

Como se escribió en la unidad teórica 5, una misma escena puede cambiar su carácter según la iluminación empleada, incluso un lugar o un personaje puede pasar de tener un carácter amable a un carácter siniestro según la posición, el color y las sombras de las luces utilizadas.

Unidad 6: Cámaras en MAX.

Introducción teórica:

1. Introducción a las cámaras.
2. Tipos de cámaras en 3D Studio MAX.
3. Cómo crear las cámaras.
4. Parámetros de las cámaras.
5. Controles del visor de cámara.

Casos prácticos:

6.1. Creación de cámaras.

6.2. Travelling de cámaras.

1. Introducción a las cámaras

Las cámaras son instrumentos que permiten definir una vista específica y original de una escena en 3D. Por eso, muchas veces se utilizan cámaras en lugar de las vistas por defecto, hecho que hace que obtengamos más riqueza visual de nuestro trabajo 3D.

Las cámaras son elementos que recrean las cámaras reales, por lo que podemos visualizar una escena virtualmente tal como lo podríamos hacer en la realidad, por lo menos en teoría. Las cámaras reales funcionan mediante lentes y su tecnología es fundamentalmente de tipo óptico mientras que las cámaras 3D son en realidad cálculos en forma de proyección de la escena 3D en un plano 2D. Estas proyecciones se hacen a tal velocidad que podemos ver animaciones o animar la cámara misma y dar la sensación de realidad virtual.

Las ventajas de utilizar cámaras en nuestros proyectos son muy grandes, ya que hoy en día son de uso muy común en la realidad, lo que hace que los usuarios estén muy acostumbrados a su utilización y por lo tanto si creamos una interfaz basado en cámaras como se puede ver en el caso práctico “Cámaras en Director” no hace falta explicación alguna para que sean bien manejadas por los usuarios.

Además las cámaras son unos objetos más dentro de la escena y por tanto tienen algunas características comunes a todos ellos, por lo que se pueden animar, mover, rotar, etc.

Podemos tener el número de cámaras que queramos y cambiar en cualquier momento de una a otra para ver la escena. La gestión de las cámaras, por otra parte, es bastante sencilla y por supuesto, no hace falta tener la experiencia de un operador de cámara en la realidad.

Por otra parte, a veces se hace un uso abusivo de las cámaras, buscando efectos visuales fáciles como el movimiento a altísimas velocidades, giros imposibles o atravesar objetos. También es necesario pensar en la escena y no en la cámara como lo más importante, haciendo que el espectador reciba la información de la mejor manera posible.

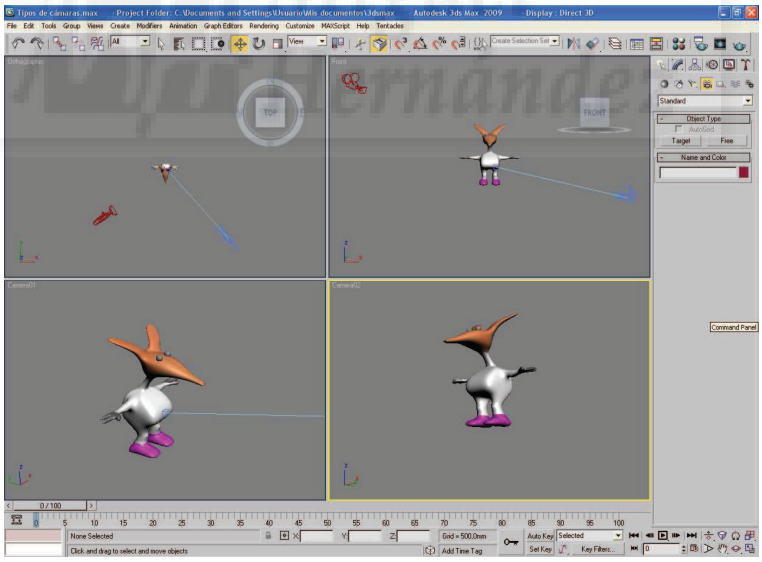


Imagen 5.6.1: Vistas de las cámaras en MAX

2. Tipos de cámaras en 3D Studio MAX

Como se puede observar en la imagen inferior, existen 2 tipos de cámaras: libres (Free) y con objetivo (Target):

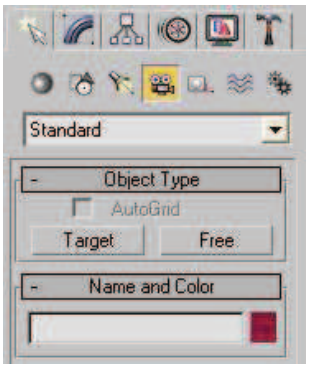


Imagen 5.6.2: Tipos de cámaras en MAX

◆ Cámara libre: es una cámara que no tiene una orientación, movimiento ni posición preestablecidas. Se puede mover o rotar libremente. En la imagen inferior se puede observar una flecha roja que señala una cámara libre.

◆ Cámara con objetivo: esta cámara cuando se mueve, lo hace apuntando siempre hacia el objetivo. Su orientación viene ya determinada lo que implica que no se puede mover o rotar libremente, sino en función del objetivo. El objetivo de la cámara se puede transformar como cualquier otro elemento y por lo tanto, de esta forma orientaremos la cámara. En la imagen inferior se puede observar una flecha que señala una cámara con objetivo que apunta al personaje. La línea azul visualiza la unión entre la cámara y su objetivo.

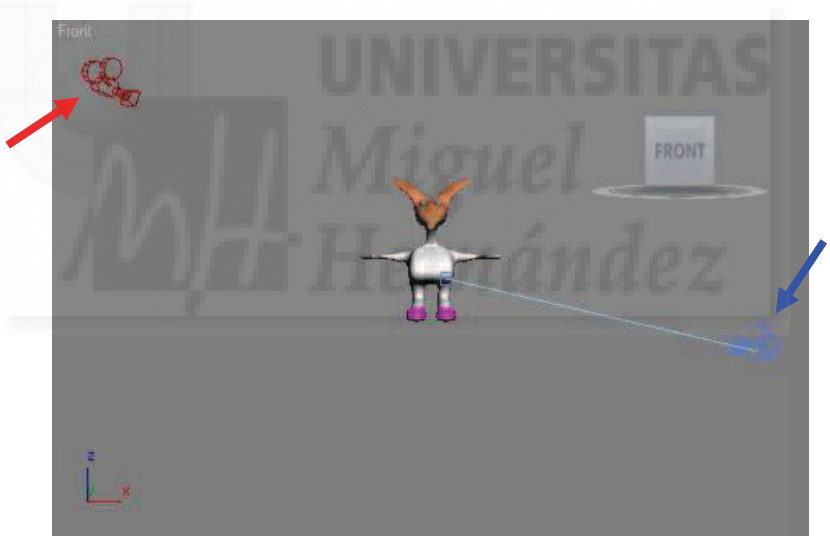


Imagen 5.6.3: Cámara libre y con objetivo

3. Cómo crear las cámaras.

Vamos a ver el método para crear la cámara sin objetivo Free. Para ello debemos acceder al botón Free que se encuentra en el panel crear > cámaras. Luego hacemos clic en el botón y volvemos a realizar un clic en el visor que deseemos aunque se recomienda la vista Top para tener una visión más clara de la escena. Al hacerlo así la cámara se encuentra a nivel del suelo, es decir, su posición en altura es 0.

Para cambiar la posición de la cámara, lo realizamos como cualquier otro objeto, o sea, manteniendo la cámara seleccionada pulsamos con el botón derecho en la herramienta “Select and Move” para que aparezcan los valores de traslación y los cambiamos por los valores que queramos o simplemente con la misma herramienta la movemos interactivamente. También el giro de cámaras es con la herramienta habitual y sirve para modificar su dirección.

Para crear una cámara con objetivo seguiremos el mismo proceso que en el caso anterior, pero cuando hacemos clic (definiendo la posición de la cámara) en el visor deseado no tenemos que soltar el botón del ratón sino dirigir el puntero y llevarlo hasta el punto donde queremos que apunte la cámara. Una vez elegido el punto soltamos el botón. De esta forma definimos el control "target" de la cámara. En este caso podemos mover y girar tanto la cámara como el target. Siempre veremos una línea que visualiza la dirección de la cámara.

Las cámaras las podemos nombrar y cambiar de color para su mejor identificación siguiendo el método estándar, es decir, utilizando los controles del grupo "Name and color".

4. Parámetros de las cámaras.

Las cámaras, como hemos escrito antes, tratan de imitar el funcionamiento de sus homónimas reales, con lo que la cantidad de parámetros que podemos variar para detallar su funcionamiento es realmente numerosa. Por esta razón vamos a nombrar solamente algunas de sus características más destacadas.

Objetivo (Lens).

Permite ajustar manualmente la distancia focal. La distancia focal es la longitud entre la lente y el punto focal, que es el punto donde convergen los rayos de luz en el objeto enfocado sin aberraciones ópticas. En la imagen se puede observar un valor de Lens de 43,456 milímetros.

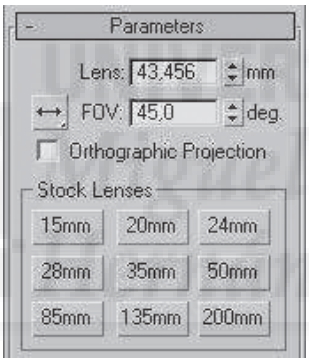


Imagen 5.6.4: Parámetros de las cámaras

Anchura del campo visual (FOV).

Campo visual que abarca la cámara medida en grados, tiene que ver con el objetivo, ya que a mayor lens, menor FOV. Por ejemplo, una lente de 15 mm. conlleva un FOV de 100º y una lente de 50 mm. no llega a 40º como se puede apreciar en las imágenes que siguen. Para interiores se suele usar una lente de 24 mm.

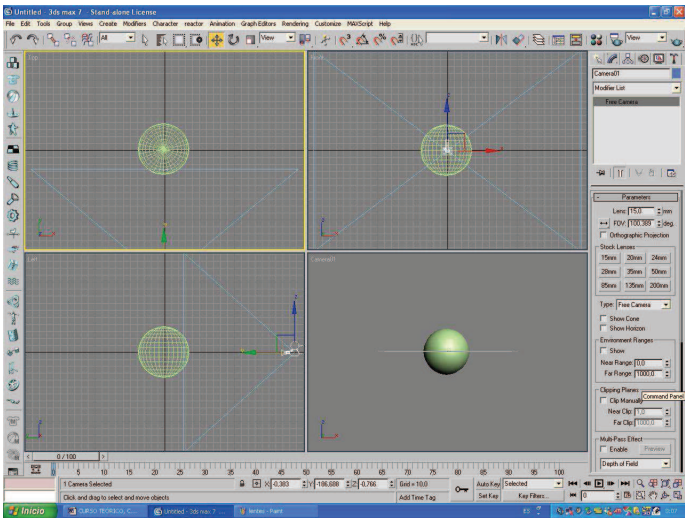


Imagen 5.6.5: Lente de 50 mm

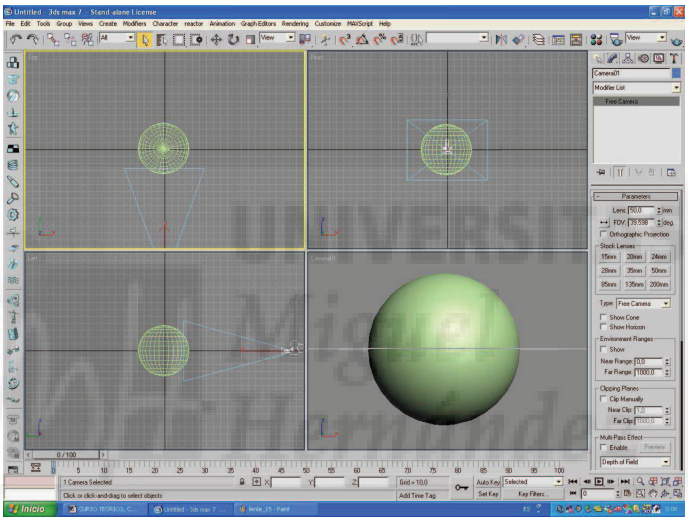


Imagen 5.6.6: Lente de 24 mm

Rangos de Entorno.

Permite definir la lejanía de los efectos atmosféricos si los hubiera.

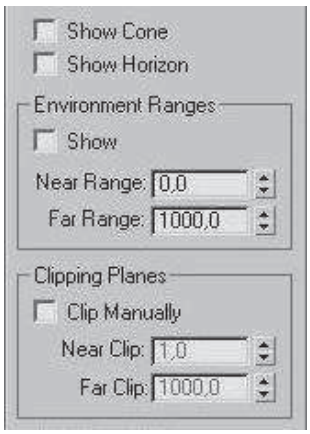


Imagen 5.6.7: Otros parámetros

Planos de recorte.

Hay dos: uno de proximidad y otro de lejanía. Estos planos permiten excluir objetos que por algún motivo la cámara debería excluir, por ejemplo, por estar demasiado cerca o al revés, por estar demasiado lejos.

Ya que la posición de estos planos se puede animar, se suele utilizar esta característica para realizar un efecto muy llamativo: cortar por capas cualquier cosa, como un edificio o un personaje como si fuera una cebolla.

5. Controles del visor de cámara

Cuando definimos una cámara, esta no aparece directamente en ningún visor. Para visualizar la vista de cámara, hacemos clic con el botón derecho del ratón en la vista que queremos cambiar, justo sobre su título. Aparece un menú dónde elegimos “Views” y en el submenú aparecen todas las vistas disponibles y las cámaras creadas encima de todas ellas con el nombre que hemos elegido al crearlas.

Una vez que tenemos la vista de cámara puesta y siendo el visor activo, en el ángulo inferior derecho de MAX, aparecerán los controles del visor de cámara que muestran unos iconos como los de la imagen que sigue.

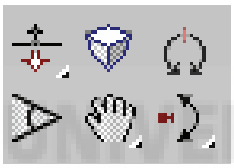


Imagen 5.6.8: Controles del visor de cámara

Vamos a explicar cada uno de estos controles por filas:

- Dolly camera: avanza la cámara en dirección al objetivo. Existen tres variantes:

    1. De cámara: acerca la cámara al objetivo a lo largo del eje de la cámara.
    2. De objetivo: desplaza solo el objetivo.
    3. De cámara y objetivo: desplaza los dos elementos.

- Perspective: añade perspectiva a la cámara sin modificar el encuadre.
- Roll camera: gira balanceando la cámara sobre su eje.
- Field of view: apertura del foco.
- Truck camera: traslada la cámara en plano perpendicular a su eje de mira.
- Orbit camera: gira la cámara usando como centro el Target. En el grupo Orbit encontraremos la Pan camera que permite a la cámara girar sobre si misma.

Como conclusión a esta unidad, podemos añadir que las cámaras son un recurso que luego exportaremos a Director y que utilizaremos ampliamente como interfaz con el usuario, por lo que conviene saber utilizarlas y definirlas en MAX.

Caso práctico 6.1: Creación de cámaras.

Objetivo: Crear los dos distintos tipos de cámaras de que dispone MAX: Free y Target.

Tiempo de realización: 30 minutos.

Pasos a realizar:

1. Crear una cámara Free.
2. Crear una cámara Target.

1. Crear una cámara Free.

Una vez abierto cualquier fichero de MAX, vamos a cambiar las vistas por defecto por vistas de cámaras. Seguir los siguientes pasos.

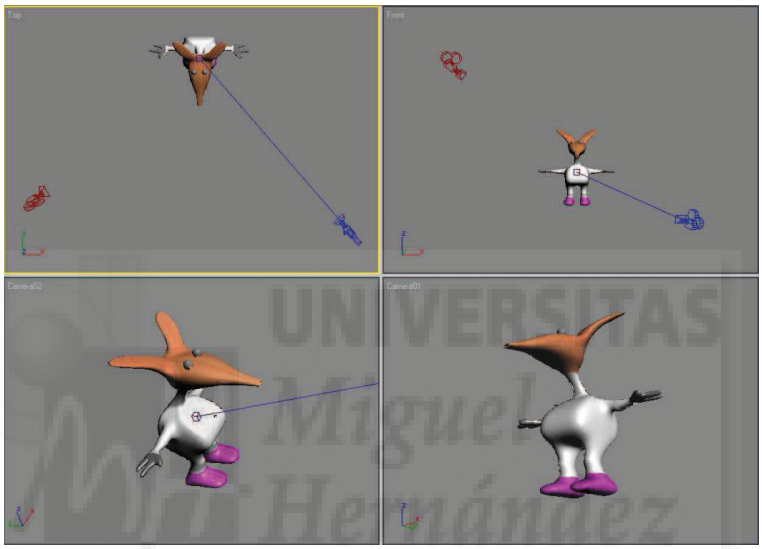


Imagen 5.6.cp.1.1: Las dos tipos de cámaras y sus vistas

1.1. Ir al panel Create > Camera y pulsamos sobre el botón “Free”.

1.2. Pulsamos sobre el visor donde queremos crear la cámara, por ejemplo, en el visor Top. No arrastrar el cursor. La cámara tipo Free no tiene objeto target y se orienta con la herramienta estándar “Girar”.

1.3. En los parámetros de la cámara, aparece el grupo “Name and Color”. El nombre por defecto es “Camera01” para la primera cámara y se irán numerando según las vayamos creando pero es muy conveniente que le pongamos un nombre propio y un color determinado para su más fácil identificación.

Ahora vamos a sustituir el visor “Perspective” por la vista de la cámara recién creada.

1.4. Pulsamos sobre el nombre del visor Perspectiva con el botón derecho del ratón.

Aparece un menú con los visores disponibles y arriba de todos ellos, tenemos las cámaras creadas con sus respectivos nombres.

1.5. En el menú que aparece elegir View > Camera01.

En la imagen de abajo podemos observar la cámara free en rojo y debajo de ella la vista obtenida desde esta cámara.

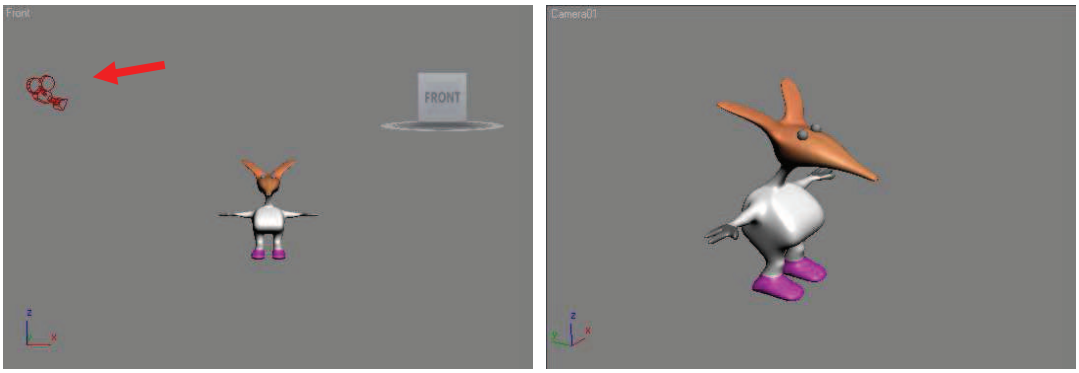


Imagen 5.6.cp.1.2: La cámara free "Camera01" y su vista

2. Crear una cámara Target.

2.1. Ir al panel Create > Camera y pulsamos en el botón "Target".

2.2. Ahora pulsamos sobre el visor Top si queremos definir la cámara aquí y arrastramos el cursor hacia el centro del visor. Observaremos como hemos creado una cámara de tipo Objetivo, ya que aparece un objeto ayudante llamado "target" que sirve para orientar la cámara.

2.3. Con la herramienta "Select and Move" movemos tanto el icono que representa tanto al punto emisor de luz como al punto Target.

En la imagen de abajo podemos observar la cámara target en azul y debajo de ella la vista obtenida desde esta cámara.

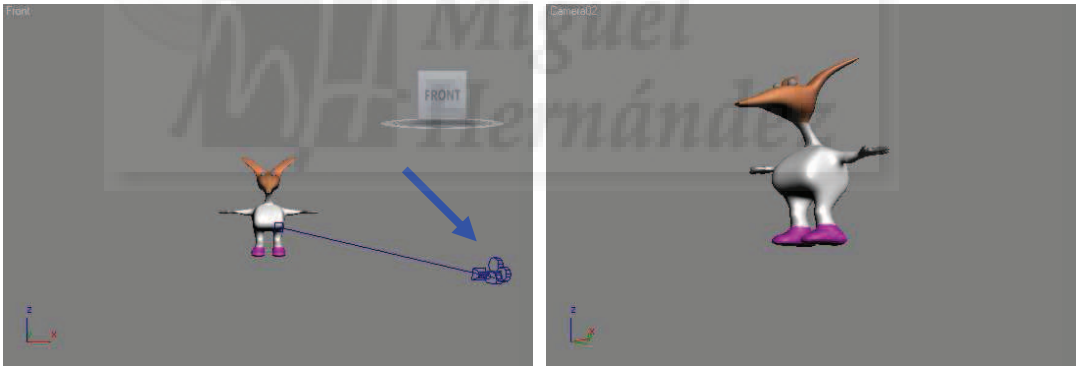


Imagen 5.6.cp.1.3: La cámara target "Camera02" y su vista

Conclusiones:

La creación de las cámaras en MAX es fácil de realizar. Sin embargo no debemos llevarnos a engaño, ya que el control de su visualización se complica debido a que necesitamos bastantes conocimientos de óptica y de cámaras reales para controlarlas del todo. Por eso recomendamos a los lectores que experimenten con los parámetros más importantes que están explicados en la unidad teórica 6.

Caso práctico 6.2: Travelling de cámaras.

Objetivo: Asignar una trayectoria a una cámara para crear una animación de su vista, efecto conocido en cinematografía como “travelling”.

Tiempo de realización: 30 minutos.

Pasos a realizar:

1. Crear la cámara.
2. Crear la trayectoria.
3. Asignar el controlador de posición del móvil a la trayectoria.
4. Producir la animación.

Consideraciones iniciales: la animación va a consistir en que una cámara girará alrededor de un objeto con la intención de mostrarlo orbitalmente. Esto es algo que se suele hacer por parte de los modeladores para mostrar sus obras. El fichero “expositor” contiene la cámara, su trayectoria y su animación y luego se puede fusionar (merge) con cualquier modelo externo para producir una visualización de este último. Esta estrategia hace que se independice el trabajo de modelador del fichero de exposición, es decir, no hay que realizar todo el trabajo de expositor para cada modelo.

Nosotros también lo podemos usar para animar una cámara en Director y tener así un interfaz distinto al habitual que consiste en animar el objeto tal como se muestra en el caso práctico 13.1. “Animación fotograma a fotograma”.

Utilizamos la asignación del controlador “Path Constrain” a la cámara, por lo que tenemos que tener cierta soltura a la hora de utilizar MAX.

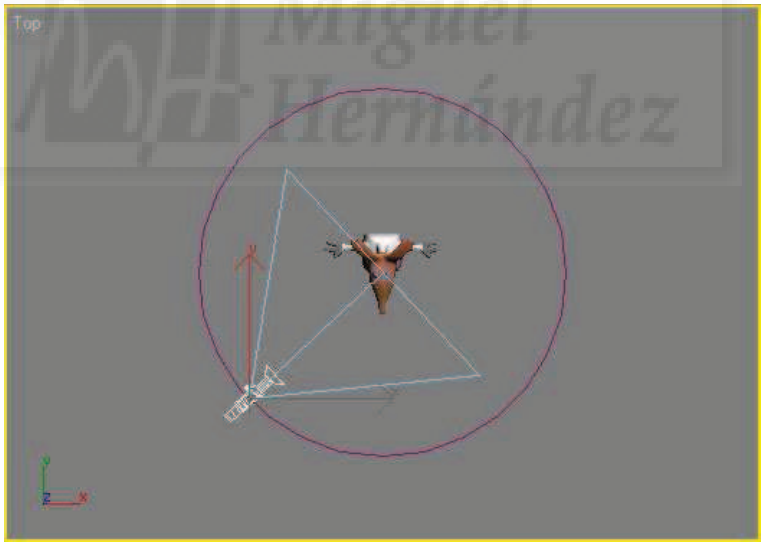


Imagen 5.6.cp.2.1: Los elementos que intervienen en el caso práctico

1. Crear la cámara.

1.1. Ir al panel Create > Camera y pulsamos en Target. Necesitamos que la cámara sea de tipo objetivo (Target) para que siempre apunte al personaje mientras se mueve siguiendo la trayectoria.

1.2. Pulsamos sobre el visor Top y arrastramos el cursor hacia el centro del visor.

1.3. Con la herramienta “Select and Move” debemos mover tanto el icono que representa a la cámara como al punto target. El target lo situaremos en el centro (0, 0, 0). Suponemos que el objeto a mostrar (el personaje), se encuentra en el centro de la escena 3D.

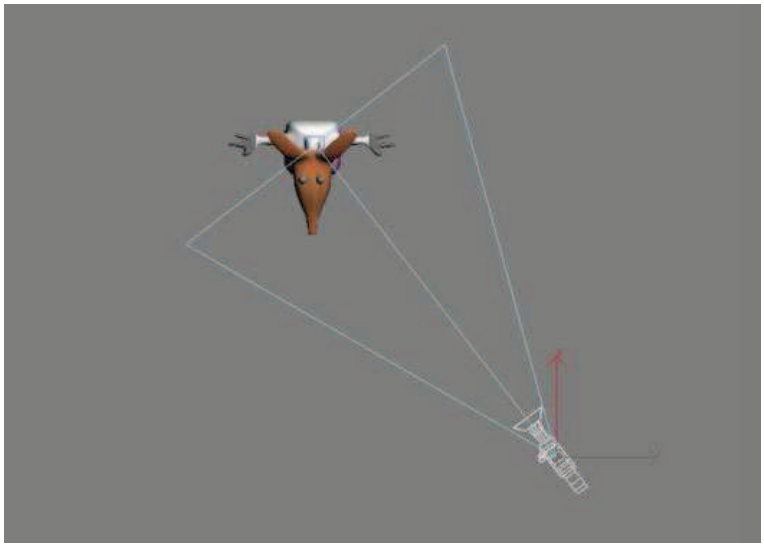


Imagen 5.6.cp.2.2: La cámara Target creada

2. Crear la trayectoria.

Vamos a crear una trayectoria circular. Esto hará que el travelling de cámara sea muy monótono pero al mismo tiempo muy típico, ya que veremos como se muestra el objeto siempre desde la misma distancia todo alrededor.

En este caso práctico podríamos utilizar cualquier otra trayectoria siempre y cuando sea una curva cerrada. Podría ser muy interesante crear una línea con splines y mover sus puntos en los tres ejes. Esto es lo que se hace en el caso práctico (antes nombrado) 13.1.

2.1. Accedemos al panel Create > Shapes > Circle. Pulsamos en el visor Top y arrastramos hasta que el círculo tenga unas dimensiones parecidas a las que deseamos.

2.2. Accedemos al panel Modify y cambiamos el radio del círculo si es necesario.

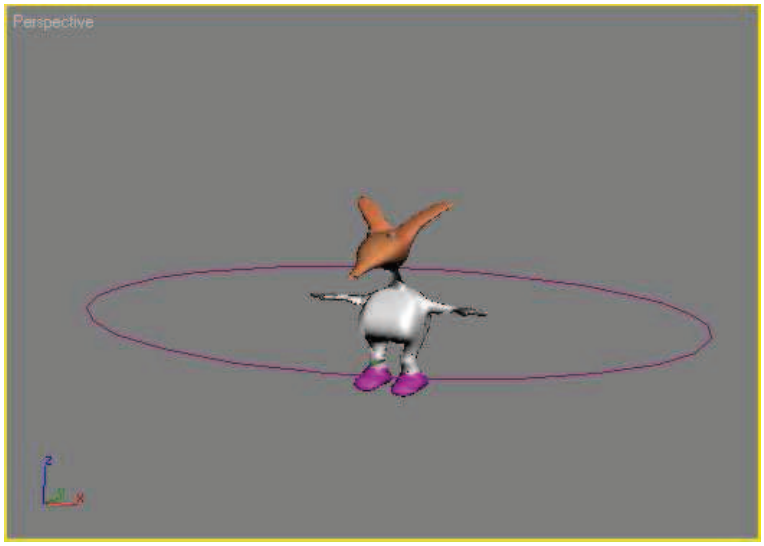


Imagen 5.6.cp.2.3: La trayectoria de cámara

2.3. Hacemos clic con el botón derecho del ratón sobre la herramienta “mover” y aparece una caja de diálogo con las coordenadas de posición del círculo. Para centrarlo en la escena 3D debemos poner los valores de X, Y, Z a 0.

En la imagen anterior se puede apreciar la trayectoria circular creada.

3. Asignar el controlador de posición del móvil a la trayectoria.

3.1. Seleccionar la cámara (no el target) e ir al “Panel Motion”.

3.2. Hacer click sobre la opción “Parameters”.

3.3. En “Assign Controller” pulsamos sobre “Position” y sobre el botón “**?**”

3.4. En las opciones pulsamos sobre “Path Constrain”. Aparecen nuevos menús.

3.5. Pulsamos sobre el botón “Add Path” y elegimos la trayectoria morada.

Existe una opción que hace que se oriente la cámara siguiendo la trayectoria. Esta opción se llama “Follow” y puede servir, por ejemplo, para hacer un travelling de una cámara libre para mostrar un vuelo a vista de pájaro. Aquí no tiene sentido ya que la dirección de la cámara la controla el target.

En la imagen inferior se puede observar dónde se sitúan los controles utilizados en este paso 3 dentro del interfaz de 3D Studio MAX.

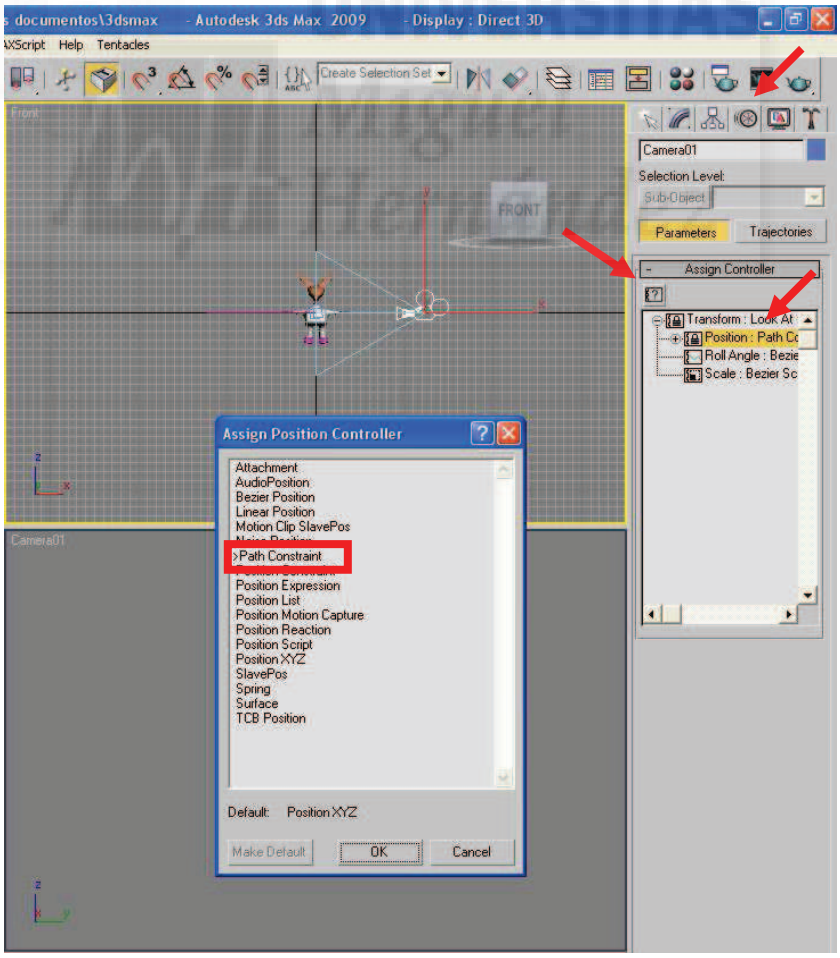


Imagen 5.6.cp.2.4: Asignación de una trayectoria a una cámara

4. Producir la animación.

Ahora la animación tendrá dos fotogramas claves asignados directamente por el controlador de movimiento que originará una vuelta completa en el tiempo dado.

Esto nos proporciona una visión de lo que la cámara visualiza de la escena 3D por si tenemos que modificar el radio de la trayectoria u otro parámetro.

4.1. Cambiar la vista del visor Perspective por la de la cámara.

4.2. Pulsar el botón “Play” para observar la animación.

Conclusiones:

Este caso práctico puede que sea un tanto complicado pero merece la pena intentar recrear este efecto en Director y poder mostrar cualquier escena tridimensional a vista de pájaro. Por ejemplo, podríamos implementar varias de estas trayectorias y que el usuario pueda elegir cual utilizar.

Unidad 7: Exportación de la escena de MAX a Director.

1. Introducción a la exportación de MAX a SHOCKWAVE 3D.
2. Adecuación de objetos de la escena antes de exportar.
3. Exportación y sus parámetros.
    - 3.1. Recursos a exportar.
    - 3.2. Configuración de la compresión.
    - 3.3. Límites de tamaño de texturas.
    - 3.4. Opciones de animación.

1. Introducción a la exportación de MAX a SHOCKWAVE 3D.

En el momento de exportar escenas realizadas en MAX a otros programas debemos tener en cuenta la aplicación destino.

En esta lección vamos a centrarnos en cómo se exportan escenas de MAX a Shockwave 3D ya que este formato es el que puede importar Director para terminar nuestro proyecto 3D interactivo. Todo el proceso lo vamos a realizar mediante un único comando de MAX que permite exportar a distintos formatos y que tendremos que parametrizar para minimizar problemas posteriores en Director.

2. Adecuación de objetos de la escena antes de exportar.

Existen algunas cuestiones que hay que tener en cuenta antes de exportar y que afectan a muchos de los objetos que componen la escena. Vamos a detallarlos por tipos.

- Modelado de objetos

Debemos exportar los menos polígonos posibles. El contador de polígonos es una utilidad que nos dará una idea de cuan grande es nuestra escena en cuanto a los polígonos que la componen. Este programa de MAX está en el panel Utilidades > Mas > Contador de polígonos.

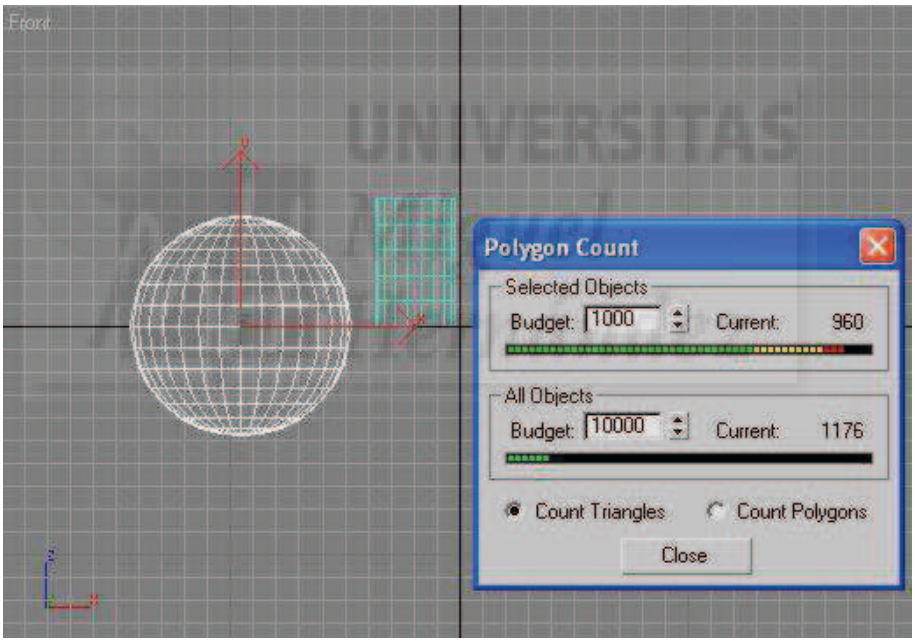


Imagen 5.7.1: Utilidad para contar polígonos de MAX

También hay que tener en cuenta que es conveniente identificar cada uno de los modelos por su nombre, ya que así será mucho más fácil de identificarlos luego en Director si queremos programarlos. También conviene nombrar las cámaras y las luces que usemos.

- Iluminación

Es un tema no del todo resuelto en Director, por lo que iluminaremos en MAX posicionando y configurando las luces que queramos para después exportar a Shockwave.

Shockwave 3D solo admite 8 luces a la vez en una escena y además “pesan” mucho por lo que hay que tener cuidado en su utilización. Además no permite la atenuación de luces, ni algunos efectos como resplandores y otros. En Lingo si existe alguna función de atenuación de luz pero hace mucha mas lenta la interacción.

◆ Texturas

Shockwave 3D soporta bien las texturas para el color difuso, pero otras no tanto, por ejemplo, las texturas de relieve o de desplazamiento no se puede exportar. La solución sería crear una sola textura con todos los efectos que podemos conseguir con todos los mapas, aunque esto es muy difícil de conseguir y el resultado final no tiene la misma calidad.

◆ Cámaras

La cámara será el punto de vista por defecto de la película Shockwave, de ahí su importancia. Es conveniente elegir bien la primera vista de nuestro trabajo. Como hemos dicho antes conviene nombrar la o las cámaras que usemos.

La longitud focal debe de ajustarse para la escena. Un FOV de 85º está bien ya que se aproxima mucho al del ojo humano.

◆ Otros

No debemos utilizar recursos de terceros como pueden ser jerarquías de huesos, materiales realistas, etc. ya que al no ser estándar, el exportador no se encuentra preparado para exportarlos de forma adecuada. Otros como Character Studio sí lo podemos utilizar pero solo a partir de la versión 3.

3. Exportación y sus parámetros.

Utilizar el exportador que proporciona MAX es la forma más segura y eficaz de realizar esta tarea y por tanto, no se recomienda utilizar software de terceros.

El método es sencillo, se ejecuta el comando File > Export y aparecerá la caja de dialogo “Select File to export” elegimos en el tipo la opción Shockwave 3D scene export (.w3d) y aparecerá la caja de diálogo mostrada en la imagen que sigue.

Como se puede apreciar tiene múltiples opciones que detallamos ya que es de vital importancia para facilitar nuestro trabajo en Director y van a determinar su calidad.

Estas opciones las vamos a enumerar en inglés ya que es como aparecen en la caja de diálogo que se presenta. Están organizadas en 4 grupos y son las siguientes:

3.1. Resources to Export, recursos a exportar:

◆ Scenegraph hierarchy: esta opción permite exportar la relación padre-hijo entre todos los objetos, incluyendo geometría, luces, grupos y cámaras. Seleccionaremos siempre esta opción cuando exportemos la escena entera, si exportamos algún elemento en particular la podemos desactivar.

◆ Geometry resources: exporta los modelos 3D con sus mallas y huesos. No tiene mucho sentido no tenerla siempre activada.

◆ Animations: la animación que se exporta es por frames y luego se comprime en un formato de stream donde para cada marco solo se escribe aquello que cambia con respecto al marco anterior. Por lo tanto hay que tener cuidado con animaciones con movimientos muy rápidos por si hay que modificar el intervalo entre frames.

◆ Material resources: la opción de los datos tipo material debe estar siempre activa.

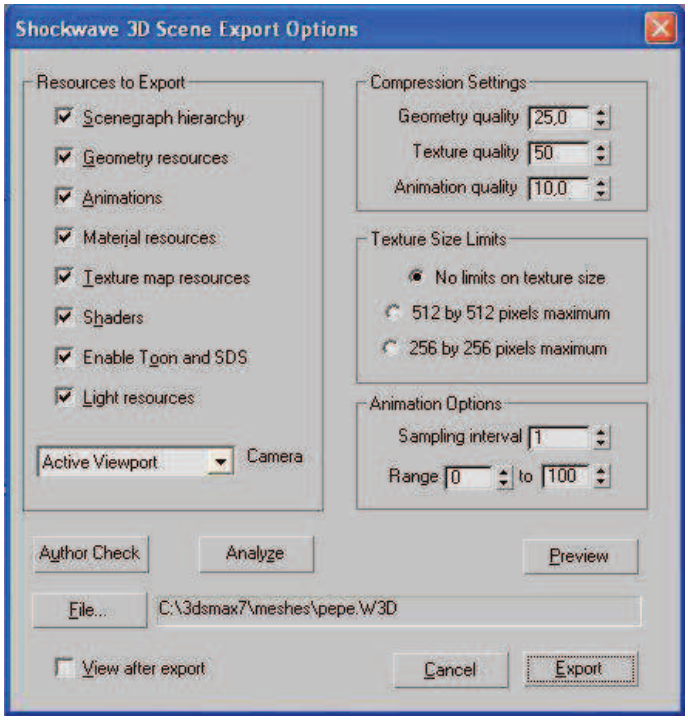


Imagen 5.7.2: Exportador de MAX a Shockwave 3D

◆ Texture map resources: esta opción hace que los materiales aparezcan en Director asignados a las mismas caras de la geometría que en MAX. Si la deseleccionamos se exportarán los modelos por un lado y las texturas por otro. Esto no es descartable, ya que puede ser que necesitemos modificar la textura de un objeto en tiempo de ejecución de la película utilizando Lingo.

◆ Shaders: esta opción determina el modo de representación del archivo exportado. Debe ser consecuente con las texturas y materiales utilizados para nuestra geometría. Shockwave 3D no diferencia entre los tipos de sombreados de MAX como Blinn, Phong, etc. solo lo utiliza para tener información de a partir de qué punto del objeto debe colocar las texturas y materiales. Si no se activa esta opción, no se representarán los objetos, aunque se puede asignar un shader al objeto con Lingo, con lo que el objeto aparecerá como por arte de magia, por lo tanto, conviene tenerla activada la mayoría de las veces que exportemos.

◆ Enable Toons and SDS: permite exportar nuestra escena con dos características. Toons es un modificador que permite representar la escena a modo de dibujos animados. SDS es un modificador que añade detalle a los modelos subdividiendo los polígonos.

◆ Light resources: normalmente está siempre activa, la única razón para no tenerla sería para exportar la escena por partes: luces, geometría, textura, etc.

3.2. Compression Settings, configuración de la compresión:

Son parámetros que van a medir la calidad con la que se exportan los elementos fundamentales. Se mide en porcentaje y no son lineales. Sus valores determinarán de forma inversamente proporcional dos características: la calidad y la memoria que ocupará el fichero.

◆ Geometry quality: determina la calidad de la exportación de la geometría. Si tiene un valor de 100 se exportará tal cual. Si se baja el valor, se exportarán menos polígonos y por lo tanto será más “ligero”.

◆ Texture quality: determina el método de compresión de las texturas. Por defecto es .jpeg. Esto implica un ahorro muy grande en memoria, por lo que se puede poner a un 50% la mayoría de las veces.

◆ Animation quality: determina la cantidad de fotogramas que se exportan y se calcula por funciones matemáticas de compresión. Su porcentaje depende de la delicadeza de las animaciones que necesitamos en Shockwave 3D. La diferencia con la opción "Sampling intervalo" que se verá más adelante, es que no se toma por ejemplo, 1 de cada 2 fotogramas automáticamente sino que se calcula cómo debe ser ese ahorro para cada caso.

3.3. Texture Size limits, límites de tamaño de texturas:

Se explica por sí misma. Tenemos 3 opciones para limitar el tamaño de las texturas. "No limits" exportará la textura tal cual. Las otras dos lo harán en el tamaño que indican.

3.4. Animation options, opciones de animación:

◆ Sampling interval: determina la fidelidad de los marcos exportados con respecto a los originales. Si ponemos 2, se creará un fichero .w3d con la mitad de marcos que el fichero de MAX, la consecuencia es que la animación es el doble de rápida y el fichero, la mitad de grande.

◆ Range: permite definir qué porción de la animación será exportada. Podemos tener una animación en MAX y desear exportar solo una parte de ella.

Además tenemos los botones:

◆ Author Check: este botón permite visualizar problemas que puedan haber en la exportación mediante un informe que presenta el módulo exportador. Es conveniente consultarlo antes de pulsar el botón "Export".

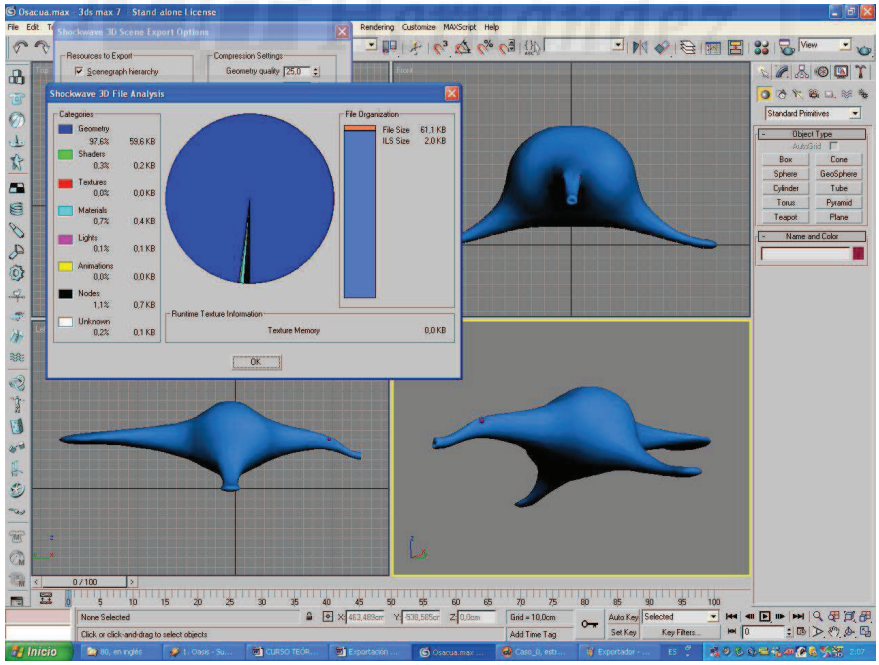


Imagen 5.7.3: Exportador de MAX mostrando la utilidad "Files Análisis"

◆ Analyze: permite visualizar una gráfica donde se puede apreciar el tanto por cien de cada parte en el "peso" del archivo a exportar. En la imagen siguiente se puede apreciar como al exportar una especie de mamífero marino, el 97% del tamaño se debe a la geometría.

◆ Preview: permite previsualizar como será el resultado que obtendremos en Shockwave. El resultado se puede observar en la imagen 5.7.4.

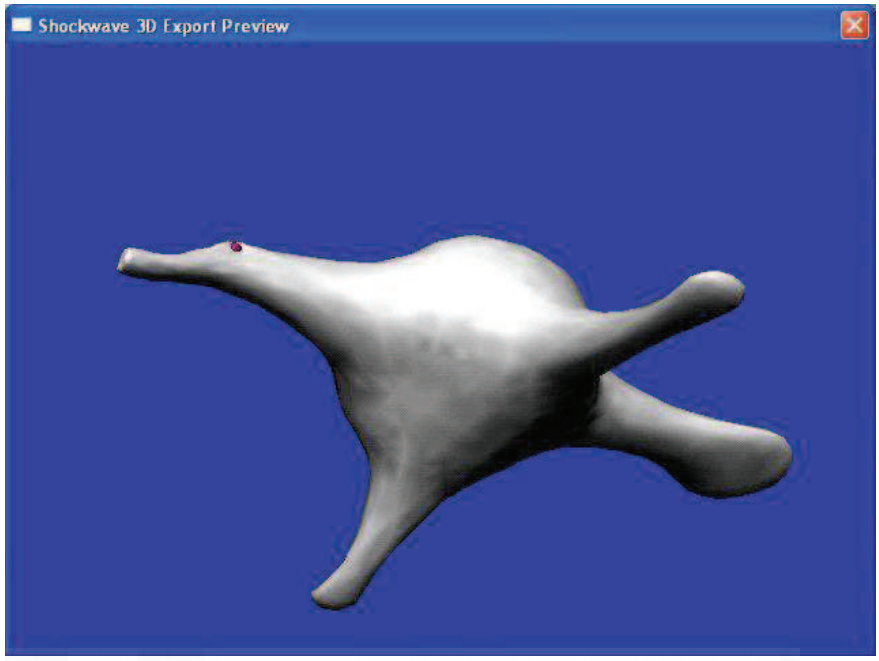


Imagen 5.7.4: Exportador de MAX, utilidad Preview, vista previa

Por lo tanto, las conclusiones a esta unidad son que nos conviene conocer todas las opciones antes detalladas para que al pasar la escena de MAX a Director perdamos la menos calidad posible y que estos recursos puedan ser utilizados de forma eficaz.

Unidad 8: Introducción al trabajo 3D en Director.

Introducción teórica:

1. Introducción a la programación en Director.
2. Creación de la película base para los casos prácticos.
3. Transformación de objetos por programación de localización, escala y rotación.

Casos prácticos:

8.1. Estructura básica de una película 3D interactiva.

8.2. Transformaciones básicas de objetos 3D.

1. Introducción a la programación en Director.

Lo primero que debemos escribir sobre la programación de Director es que no debemos tenerle miedo. Vamos a introducirnos poco a poco, de manera muy modular en su conocimiento, por lo que la curva de aprendizaje será muy suave.

Lo siguiente, es que la programación en el caso que nos ocupa es imprescindible, es decir, no tenemos elección. La programación nos aporta algo fundamental para un autor: el control total. Y a esto no podemos renunciar, ya que sería renunciar a una parte de la autoría de la obra y más cuando está al alcance de todos. No hace falta ser un programador profesional para realizar los trabajos que siguen.

◆ Programación de scripts en Director

La programación no tiene por que ser difícil, ya que se trata de escribir una serie de sentencias (frases en el lenguaje tradicional) que utilizan en todo momento la lógica. Pero al igual que en un lenguaje tradicional, las frases normalmente no se expresan sueltas ya que carecerían de sentido. Existe una estructura que podemos llamar contexto (por ejemplo una conversación) donde la frase tiene sentido. En la programación pasa lo mismo: tenemos que escribir lo que queremos pero de forma estructurada.

En Director no construimos un programa de principio a fin. A día de hoy, ningún software se crea de esta manera. Las aplicaciones son estructuras compactas pero modulares, por lo que funcionan por sí mismas y además permiten que se las pueda personalizar el funcionamiento. Esto supone que el usuario programador solo debe escribir los componentes que quieren que modifique cierta parte del funcionamiento básico del software. Este es el concepto de script, es decir, escritura. Los scripts son pequeños trozos de código que por sí mismos no realizan una tarea muy compleja, pero que en unión a un software que los integra, pueden hacer que funcione como al programador le interesa.

Por ejemplo, al escribir un script en Director, nosotros no pretendemos empezar de cero, sino que queremos que el propio Director se adapte a nuestras necesidades. En este sentido el software programable por script es muy flexible y maleable y es el paradigma de programación que se utiliza a día de hoy.

◆ El lenguaje nativo Lingo

En Director podemos programar nuestra película con un lenguaje propio llamado Lingo. Lingo es el lenguaje interno (o nativo) de Director. Lingo es una palabra inglesa que significa “jerga”. Es un lenguaje que poco a poco ha ido aumentando su capacidad para dar respuesta a nuevos formatos con los que trabaja Director. Cuando a partir de la versión 8.5 de este programa se le añadió la posibilidad de trabajar con entornos 3D, Lingo sufrió una seria transformación al añadirse toda una serie de funciones enfocadas a la manipulación y gestión de mundos tridimensionales basadas en las funciones 3D de Java.

En Director se puede utilizar cualquiera de los dos lenguajes, Lingo o JavaScript para programar en 3D.

Lingo toma una importancia muy grande en este tipo de trabajos. Para otras cuestiones, Director es muy flexible y permite que trabajadores de distinto perfil como diseñadores, programadores,…, adapten su forma y flujo de trabajo a sus propias necesidades, sin embargo, Lingo es la única manera de trabajar en 3D con Director. Lingo es un lenguaje moderno pero esto no significa que sea fácil de aprender. Por otra parte, Lingo hace que el programador tenga un control total del entorno 3D y esta es la mejor garantía de solventar los problemas que conlleva todo el trabajo en este tipo de aplicaciones.

◆ La estructura de un script de Lingo

La estructura de un script en Lingo puede resumirse en el siguiente script:

```
global mundo3D

on mouseDown
  mundo3D.model("Esfera01").Scale(.5,.5,.5)
end

on mouseUp
  mundo3D.model("Esfera01").Scale(2,2,2)
end
```

Normalmente un script usará variables globales, es decir, datos que se han definido en otro script y que pueden ser utilizados en cualquiera de ellos. Además también pueden necesitar acceder a otros datos variables. Esto es lo que se hace en la primera sentencia.

Luego escribiremos una serie de disparadores, estos se ejecutarán cuando se den los eventos que los disparan. Existe en Lingo una lista de disparadores para el ratón y el teclado. Iremos viéndolos a medida que nos hagan falta. Por ejemplo, entre los disparadores que controlan los botones del ratón tenemos a MouseUp y MouseDown y sirven para detectar cuando se pulsa el botón izquierdo del ratón y cuando se suelta respectivamente.

Podremos escribir tantos módulos disparadores como queramos. Los disparadores se escriben con la siguiente sintaxis:

```
on nombre_disparador
        acciones
end
```

◆ El material de trabajo: el entorno 3D y sus nodos.

El entorno 3D o mundo 3D es un entorno dinámico que o funciona en su totalidad o puede que no pueda ni ejecutarse. Está compuesto por una serie de elementos básicos como modelos geométricos, cámaras, luces, texturas, etc. A cada uno de estos elementos que se encuentre en el mundo 3D se le denomina nodo, y como vemos los hay de distinto tipo.

Un nodo, por tanto es una entidad u objeto que tiene propiedades y funciones propias, es decir, es una implementación al pié de la letra del paradigma de la programación orientada a objetos. Todo lo que existe en un entorno 3D, serán nodos, pero unos son más importantes que otros, podemos decir que hay elementos principales y junto a estos, tenemos unos elementos secundarios, que también son objetos pero que definen relaciones y propiedades de los nodos principales. Para aclarar un poco estos conceptos vamos a enumerar algunos de estos elementos principales y secundarios.

Elementos principales:

1. Model: son los objetos sólidos de una escena.

2. Light: fuente de iluminación, aporta la proporción de los modelos que se ven.

3. Camera: punto de vista el entorno, son como los ojos de una persona.

4. Group: agrupan varios elementos principales.

Elementos secundarios:

1. Modelresource: aporta datos geométricos de los models.

2. Shader: determina el método de la calidad de las superficies de los models.

3. Texture: determina el material de las superficies de los models.

4. Modifier: procesa propiedades básicas de los models como animación,...

5. Motion: aporta información de programas externos a Modifier.

◆ Cómo y dónde programar.

Una vez que conocemos con qué vamos a trabajar conviene saber cómo se puede “introducir” o asignar programación a estos elementos. Este trabajo lo realizaremos en la ventana Score que podemos observar en la imagen que sigue.

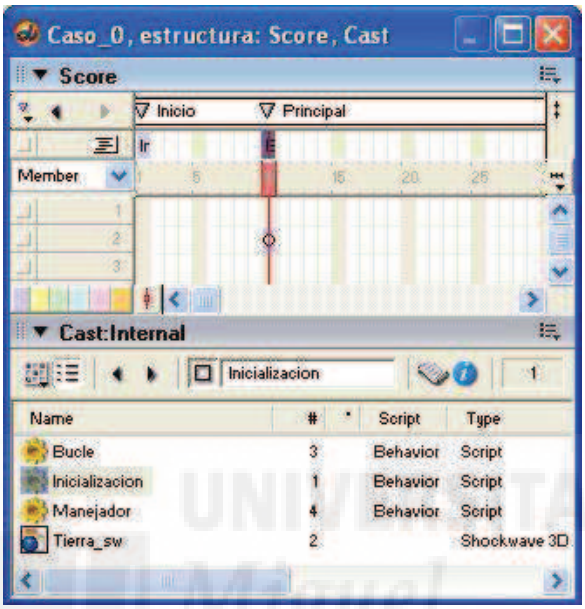


Imagen 5.8.1: Ventana Score

Es fundamental conocer qué puede realizar un script y que este puede ser asignado tanto a un sprite de un determinado member o un frame de la película. La diferencia es clara, si se le asigna al frame, queda desvinculado de un determinado miembro, es decir, se ejecuta cuando llegue ese frame y podemos manipular los members de manera “externa” a ellos mismos.

Cuando se escribe un script se utilizan normalmente variables globales, estas se escriben antes de nada. Luego se van escribiendo sentencias pero insertándolas en módulos que se ejecutan cuando se ejecuta un disparador o trigger por la intervención de un usuario. Por ejemplo, en el siguiente módulo se realiza una tarea pero sólo se ejecuta cuando se levante el dedo del botón izquierdo del ratón (mouseup)

```
on mouseUp me
   gWorld.group("phone").transform = gInitTransform
end
```

Esto hace que en un mismo script pueda haber muchos módulos de programación. Aunque de hecho vamos a programar casi toda la aplicación en un solo script para tener la mayor complejidad posible encapsulada en un solo script de programación.

◆ La ventana Script.

Cuando se crea un script, se escribe en una ventana especial que es en realidad toda una herramienta de programación.

Esta ventana es en realidad un editor de textos donde escribir la programación. Durante el curso veremos para lo que sirven sus funciones más habituales. Auque debemos destacar el

control de la parte superior izquierda donde pone Lingo. Es aquí donde decidimos si el código es escrito en Lingo o en JavaScript y por supuesto, esto es muy importante.

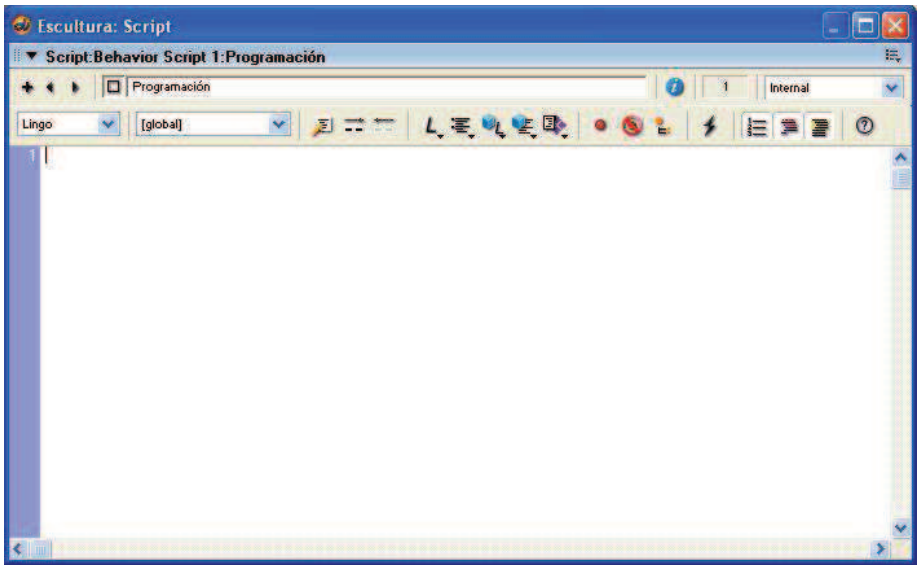


Imagen 5.8.2: Ventana de programación Script

2. Creación de la película base para los casos prácticos.

En el caso del tema 2 referente al flujo de trabajo de la solución MAX – Director estudiamos cómo incorporar una escena 3D realizada en MAX a Director. Como comprobamos en dicho caso práctico solo podemos visualizar la escena utilizando los botones propios de la ventana Shockwave 3D que aunque muy prácticos, no permiten más que una exploración de la escena 3D con funciones de la cámara por defecto de la escena como el zoom, el encuadre y la rotación. La evolución lógica de este trabajo es el aportar interactividad a la película para proporcionar libertad de acción al usuario espectador.

Por lo tanto, nuestro siguiente objetivo debe ser aportar interactividad a la película. Pero antes de esto, debemos de poder controlar los elementos importados desde MAX y poder inspeccionarlos, nombrarlos y modificarlos en Director. Y para esto es imprescindible utilizar la programación.

Es por esto que vamos a intentar, antes que estudiar las posibilidades de interacción, el crear una película con una estructura básica que nos permita un punto de inicio, una referencia, para todos los trabajos que siguen. Es decir, crear una película en Director, cuya estructura se adapte a los objetivos propuestos en la tesis: la integración de mundos 3D y la interactividad con esos mundos virtuales.

- Estructura de la película.

La mejor estructura de la película de Director que pretendemos realizar para trabajar en 3D solo utilizará dos frames y tres scripts de programación.

Los frames o fotogramas utilizados serán: el primero y otro cualquiera (en cualquier posición, por ejemplo en la 10). Esto se hace para separar claramente dos momentos distintos en la ejecución de la película que equivalen a la iniciación del programa y al desarrollo del mismo.

El primer fotograma lo llamaremos “Inicio”. La iniciación tiene por objeto esperar a que se cargue el fichero .w3d, ya que normalmente tarda un poco de tiempo. También se puede aprovechar el inicio para inicializar variables globales y otros aspectos de la programación que iremos viendo a la largo del curso.

El segundo fotograma lo llamaremos “Principal”, y va a tener dos funciones: por un lado un bucle “sin fin” que construiremos haciendo que cuando acabe de ejecutarse este fotograma se vuelva a ejecutar de nuevo, y por otra parte toda la programación que incumbe a la escena 3D.

Con respecto a los 3 scripts imprescindibles para el manejo de la película 3D los nombraremos según su función como:

Programación: lo asignaremos al sprite del miembro Shockwave 3D y contendrá la mayor parte de la programación del trabajo. De hecho, será aquí donde crearemos toda la programación que cambie de un trabajo a otro, ya que los otros dos scripts apenas tendrán que ser modificados. Incluirá la programación de la interactividad, de la presentación de la creación de entidades, la inferfaz, o sea, casi todo.

Inicialización: se asigna al frame inicial y tiene como objeto cargar el miembro 3D y retener la ejecución de la película hasta que esto suceda, ya que esta carga no es instantánea y depende de la velocidad de ejecución, es decir, del ordenador del usuario y de la complejidad del mundo 3D.

Bucle: se asigna al frame que se ejecute al mismo tiempo que el Shockwave 3D y tiene por objeto no salir de este frame, por tanto, toda la aplicación se ejecutará como suspendida en el tiempo, y toda ella se controlará internamente en el script “Programación”.

- Crear el script “Inicialización”.

Debemos crear un “Frame Script” que se corresponda con el primer fotograma. En la última fila de los canales (la ventana divide los 6 canales de los miembros para cada frame) se encuentra el canal de script, lo que permite asignar una script (pequeño trozo de código de programación) directamente a un frame (fotograma de la película interactiva). Esto se realiza haciendo clic con el botón izquierdo para el elegir el frame en cuestión, en este caso el 1, y con el botón derecho hacemos clic y aparece el menú contextual, donde elegimos la opción “Behaviors”. En “Object Inspector” aparecerá una ventana con un botón “Añadir”, donde pulsamos y esto hace que aparezca la ventana “Name Behavior” (nombre del comportamiento).

Al comportamiento le pondremos el nombre de “Inicialización”. En la ventana de Score debe aparecer un nuevo miembro que es el comportamiento recién creado. Si vemos que no está asociado al frame, lo seleccionamos y lo arrastramos hasta la casilla 1 del canal de Script. Esta casilla debe aparecer en color morado y con las dos primeras letras del nombre del comportamiento.

Imagen 5.8.3: Caja de diálogo para el nuevo comportamiento

Abriremos el comportamiento “Inicialización” seleccionándolo primero y luego pulsando el botón que se encuentra arriba y que parece una página. Aparecerá una ventana con un aspecto un tanto complicado debido a que es un verdadero centro de programación con ayuda integrada, sistema de depuración de código,… Escribimos el siguiente código:

```
global mundo3D

on enterFrame me
        mundo3D = member("escultura")
        if mundo3D.state = 4 then
                mundo3D.resetworld()
```

```
                go to frame "Principal"
            else go the frame
        end if
end
```

Lo primero que hay que señalar es que los colores los pone el editor de textos integrado de Director. El color azul denota que son palabras reservadas, es decir, pertenecientes al lenguaje Lingo. El color verde quiere decir que son nombres de variables o funciones de Lingo. Por último, las palabras en negro son las definidas por el propio usuario.

La primera línea sirve para ponerle un nombre interno a la escena importada (yo la he llamado mundo3D) para todo el programa, es decir, como variable global.

Las tabulaciones dan una idea de pertenencia de unas líneas sobre otras. Esto indica visualmente que todo el resto, está “dentro” de la función on enterFrame me y se ejecutarán cuando se ejecute esta función.

EnterFrame significa que se ejecutará cuando entremos en este fotograma, como es el primero, significa que se ejecutará nada más comenzar el programa.

Lo que hacemos es decir que “mundo3D” es la escultura. Luego que si el estado de la escultura es de cargado (cuando state vale 4), entonces podemos resetear el mundo (resetworld) e ir al fotograma de ejecución (al principal) y si no es así, debemos volver al mismo fotograma para seguir esperando a que se cargue.

◆ Crear el script “Bucle”.

Crear un “Frame Script” para el fotograma “Principal” llamado “Bucle” por el mismo proceso que el script “Inicialización” y asignarle el siguiente código:

```
on exitFrame me
  go the frame
end
```

Este código significa que cuando en la ejecución propia de la película, salgamos del fotograma volvemos inmediatamente a él, a la velocidad de X fotogramas por segundo que tenga asignada la película. Por lo tanto, estaremos creando un “bucle sin fin”.

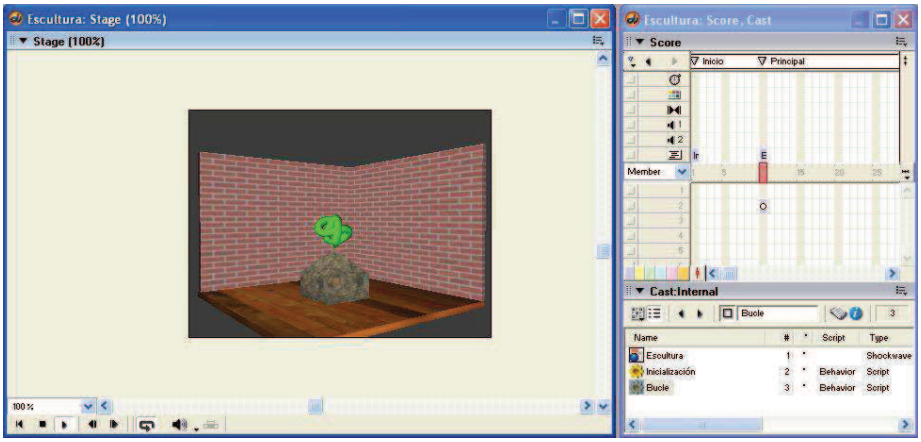


Imagen 5.8.4: Ventanas Stage, Score y Cast

Este mecanismo nos servirá para poder controlar la ejecución de la película y centralizar aquí toda la programación. Ahora las ventanas principales de Director tendrán el aspecto de la imagen 5.8.4, con el Stage (Escena), Score (Elenco) y la ventana de Cast (actores) cada uno

de lo cuales tienen distintos iconos para identificar rápidamente que son de distinto tipo: Shockwave 3d y comportamientos.

La estructura de la película es un tema muy importante para el desarrollo de nuestro sistema. Podemos decir que se conjugan dos cuestiones: por una parte el particular modo que tiene Director de trabajar, es decir, la forma de ver todas las presentaciones como si de una película se tratara. Por otra parte, trabajamos con Shockwave 3D que es un mundo virtual e interactivo "encerrado" en sí mismo e independiente. Por lo tanto tenemos que poner algo dinámico (el objeto mundo) dentro de otro también dinámico, que es la película. Todo esto se resuelve satisfactoriamente de la manera descrita en este caso práctico.

◆ Crear el script "Programación".

En el mismo fotograma "Principal" crearemos el script de programación llamado "Programación" pero no se lo asignaremos al fotograma sino al sprite del mundo 3D. Este script centrará toda la complejidad de la programación de la película y es dónde realizaremos los trabajos más complicados en cuanto a programación se refiere.

3. Transformación de posición, escala y rotación de objetos por programación.

En este punto vamos a empezar la verdadera programación, por lo que las explicaciones que siguen son de suma importancia y la base para toda la codificación futura. Comenzaremos discutiendo cómo crear un script asociado al sprite del mundo 3D y cómo podemos referenciar las distintas partes del mundo tridimensional del que disponemos en Director. Luego nos adentraremos en cómo ejecutar los comandos para controlar la ejecución del proyecto mediante eventos y por último veremos las primeras modificaciones de los modelos 3D en tiempo real.

◆ Creación del script de programación

Vamos a crear un script asociado al sprite del mundo 3D, es decir, a la aparición en la película del castmenber correspondiente. Para ello seleccionaremos el sprite de nuestra escena 3D y accederemos a la ventana Code > ficha Behavior Inspector. Pulsamos sobre el botón + y ejecutamos el comando "New Behavior". Aparece una ventana para escribir el nombre de nuestro script que será un nuevo miembro del Cast.

Nos aseguraremos de que este comportamiento esta enlazado con el Sprite 3D seleccionándolo y arrastrándolo hasta el sprite que corresponde.

◆ El acceso al mundo 3D

Para tener acceso a la escena 3D importada de MAX y todos sus componentes, lo primero que escribiremos en el script es la declaración de la variable global que la referencia. Por lo tanto, escribimos:

```
global mundo3D
```

Esta es la misma que ya definimos en el script "Inicialización" y que permitía referenciar la escena 3D. Al ser global, estamos accediendo a la misma variable.

◆ Crear el disparador

Como escribimos anteriormente, utilizaremos los disparadores cómo módulos dentro de nuestro script. Ejecutarán las sentencias escritas en su interior cuando a lo largo de la ejecución del programa suceda un evento que los active y dispare. Por ejemplo, nosotros queremos crear una interactividad muy básica, basada en hacer clic con el ratón, para ello, utilizaremos el disparador mouseDown para controlar cuando el usuario hace clic sobre la escena. Para ello, escribimos:

```
on mouseDown
        -- aquí irán las acciones
end
```

Ya explicamos el significado de los colores en los scripts, pero llama la atención las palabras en rojo. Está en rojo toda esa línea por que los primeros caracteres de esa línea son "--", esto significa que el compilador de Lingo no tiene que tener en cuenta esta fila. Pero entonces ¿para qué sirve? Pues para tener documentado nuestro programa a base de anotaciones. Por ejemplo, en este caso sirven para dejar un hueco donde escribiremos lo que hay que hacer cuando se pulse el botón izquierdo del ratón.

◆ Crear una acción por programación

Lo primero que tenemos que tener claro es lo que queremos hacer. Por ejemplo, empezaremos por hacer algo tan simple como mover la escultura.

Esto supone resolver dos problemas:

1) Cómo decirle al programa que nos referimos a la escultura y no a otro objeto ya que todos los demás objetos deben permanecer sin cambios.

2) Cómo mover un objeto.

Para resolver la primera cuestión debemos recordar que cuando se importa un mundo 3D desde otro programa, Director respeta los nombres de los objetos que se usaron en su modelado. En nuestro caso, la escultura está compuesta por un solo objeto y como no le pusimos un nombre, se lo puso MAX. Se llama "Torus Knot01".

Por otra parte, debemos saber que existe una relación de jerarquía entre todos los objetos que componen la escena. Así, resulta que el objeto escultura "pertenece" a lo que hemos llamado "mundo3D". Y por tanto podemos referirnos a nuestro objeto como: Mundo3D.model("Torus Knot01"). Podemos apreciar que existe un punto para separar los niveles jerárquicos. Model sirve para hacer referencia a los nodos de tipo modelo y no a otros elementos de la escena. Este tipo de referencia es propia de los lenguajes orientados a objetos, y Lingo lo es.

Para resolver la segunda cuestión, hay que entender que cada elemento de la escena es un objeto. Un objeto tiene una serie de propiedades como son la posición, escala,…, y una serie de procedimientos para modificar esas propiedades.

El procedimiento para trasladar un objeto tipo modelo tiene la siguiente sintaxis: traslate(desplazamientoX, desY, desZ), es decir, que se puede trasladar el objeto en 3 ejes distintos con una sola sentencia.

Por lo tanto, la sentencia para mover un objeto llamado "modelo01" 10 unidades en el eje Z y perteneciente a la escena llamada "escena01" tendrá la siguiente sintaxis:

```
Escena01.model("modelo01").traslate(0,0,10)
```

◆ Transformaciones básicas de los objetos

Para trasformar un objeto primero tenemos que saber que esta transformación necesita una referencia, es decir un origen de coordenadas. Existen dos tipos: las coordenadas locales y las universales.

Para expresar que un procedimiento debe tomar sus valores como coordenadas locales pondremos #self al final de ellos, con lo que quedará: traslate(0,0,-10, #self). Para que sean coordenadas universales pondremos traslate(0,0,-10, #world).

Las transformaciones básicas son tres: traslate (translación ,mover), rotation (rotación, girar) y scale (escala, cambio de tamaño).

El procedimiento para trasladar un objeto tiene la sintaxis siguiente:

traslate(desplazamientoX, desplazamientoY, desplazamientoZ)

El procedimiento para rotar un objeto tiene la sintaxis siguiente:

rotate(ánguloX, ánguloY, ánguloZ)

El procedimiento para escalar un objeto tiene la sintaxis siguiente:

scale(factorX, factorY, factorZ)

Hay que destacar que el procedimiento Scale no permite las coordenadas universales, ya que el factor de escala siempre se referirá al propio modelo.

Se puede ir probando los tres procedimientos para ver la diferencia. Cada vez que se cambie cualquier dato del script se debe reiniciar la película para ver los cambios. Si hay problemas para "reiniciar" la película (llamados problemas de persistencia) debemos volver a abrir el fichero a falta de soluciones más elegantes que veremos más adelante.

Para trabajar más cómodamente, podemos intentar ver el cambio y seguidamente deshacerlo. Para poder deshacer los cambios hay varias opciones entre ellas el uso de procedimientos, pero lo más rápido será deshacer las acciones con otra acción contrapuesta.

- El script final

Este script que sigue permite que un modelo llamado "Torus Knot01" perteneciente a la escena tridimentisional "mundo3d", se escale a la mitad en todos sus ejes cuando el usuario hace clic con el botón izquierdo del ratón sobre cualquier parte de la escena, y que lo escale al doble (y por tanto recupere su tamaño original) cuando suelta el botón izquierdo del ratón. Por lo tanto sirve para ilustrar una interactividad y una transformación de un objeto, sin modificarlo finalmente.

```
global mundo3D

on mouseDown
  mundo3D.model("Torus knot01").Scale(.5, .5, .5)
end

on mouseUp
  mundo3D.model("Torus knot01").Scale(2, 2, 2)
end
```

Las conclusiones que podemos sacar de este tema es que el inicio de una obra 3D en Director supones realizar una serie de "preparaciones" previas. La parte positiva es que son siempre las mismas por lo que podemos crear una "plantilla" que nos permita comenzar futuros trabajos desde un punto ya conocido y centrarnos en el desarrollo particular de la obra en concreto.

Por otra parte, la programación es una cuestión inevitable cuando se trabaja en 3D con Director. Las características del lenguaje Lingo hacen que esté muy dirigida al trabajo final, por lo que el autor se puede concentrar en la obra y no tanto en la programación, es decir, en el fin y no en el medio.

Caso práctico 8.1: Estructura básica de una película 3D interactiva.

Objetivo: Realizar la estructura de una película de Director que sea adecuada a nuestros objetivos, que son la integración de mundos 3D con Shockwave y la interactividad con ese mundo virtual por medio de la programación.

Tiempo de realización: 30 minutos.

Pasos a realizar:

1. Importar el fichero Shockwave 3D.
2. Crear el Sprite y las etiquetas básicas.
3. Crear la programación para el fotograma de iniciación.
4. Crear la programación para el fotograma de programación.
5. Publicarlo en un proyector para comprobación.

1. Importar el fichero Shockwave 3D.

1.1. Ejecutar la orden File > Import. Aparece la caja de diálogo “Import Files into internal”. Es conveniente haber abierto antes las ventanas Stage y Score. Estas operaciones las explicamos en el caso práctico del flujo de trabajo.

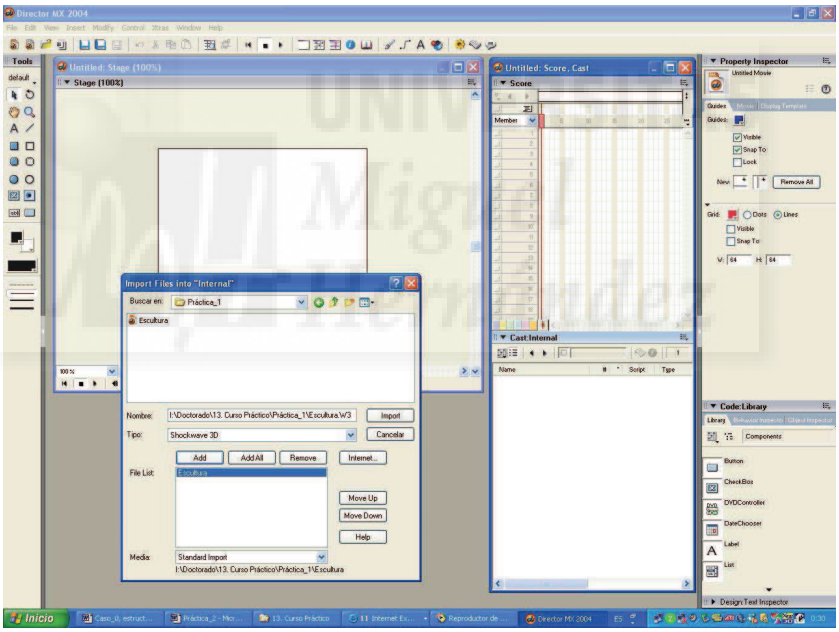


Imagen 5.8.cp.1.1: Importar archivos 3D a Director

1.2. Buscar el fichero .W3D donde tenemos la escena con la que vamos a trabajar. Supongamos que en nuestro caso el fichero se llama “escultura.w3d”, lo seleccionamos y pulsamos el botón “Add”. Pulsar el botón “Aceptar” para convertirlo en un castmember de nuestra película.

2. Crear el Sprite y las etiquetas básicas.

2.1. Abrir la ventana Score ejecutando Window > Score (o Ctrl+4). Podremos observar directamente el miembro llamado “Escultura”. Lo seleccionaremos y lo arrastramos hasta el fotograma número 10. En este momento acabamos de crear un Sprite.

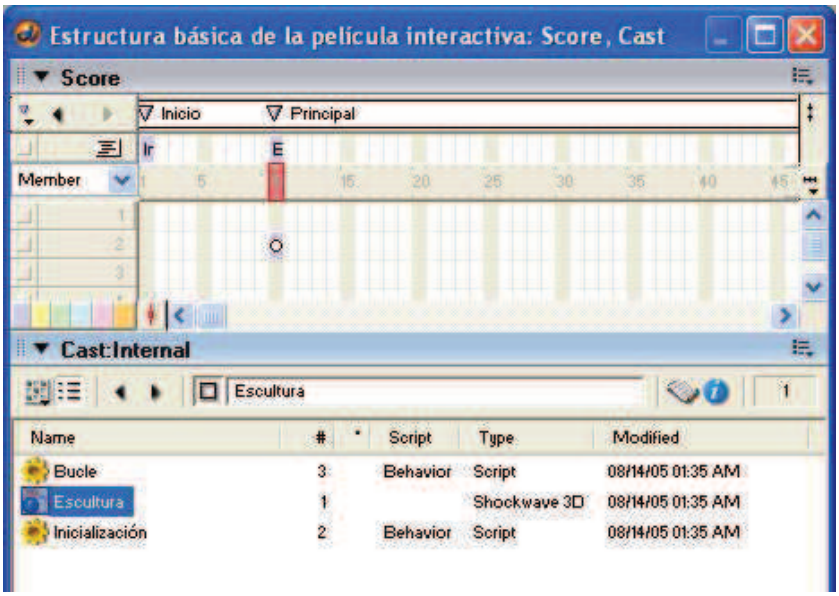


Imagen 5.8.cp.1.2: Pantalla de Score mostrando la estructura básica

2.2. Normalmente, un castmember aparecerá una serie de fotogramas, nosotros necesitamos que ocupe un solo fotograma por lo que arrastraremos desde el último fotograma ocupado por el sprite hacia el 10 hasta que su longitud sea de 1 fotograma.

2.3. Hacer clic sobre el primer fotograma en la banda en blanco situada arriba del todo. Aparecerá una ventana editable con la etiqueta "New Marker", cambiarla por el nombre "Inicio". Repetimos la operación para el frame 10 con la etiqueta "Principal". Después de estas operaciones el resultado será parecido al de la imagen 5.8.cp.1.2.

3. Crear la programación para el fotograma de iniciación.

3.1. Crear un frame Script para el primer fotograma. En la última fila de los canales, en el canal de script, hacer clic con el botón izquierdo para el elegir el frame, en este caso el 1, y con el botón derecho en el menú contextual, elegir la opción Behaviors y en el Object Inspector aparecerá una ventana con un botón de añadir, donde pulsamos para que aparezca la ventana Name Behavior, es decir, nombre del comportamiento.

3.2. Nombrar el comportamiento como "Inicialización" y pulsar OK. Si no está asociado al frame, seleccionarlo y arrastrarlo hasta la casilla 1 del canal de Script. Esta casilla aparecerá en color morado y con las dos primeras letras del nombre del comportamiento.

Imagen 5.8.cp.1.3: Caja de diálogo para nombrar el nuevo comportamiento

3.3. Abrir el comportamiento "Inicialización". Para ello seleccionarlo primero y luego pulsar sobre el botón que se encuentra arriba y que parece una página. Aparecerá una ventana de programación y escribir el siguiente código:

```
global mundo3D

on enterFrame me
        mundo3D = member("escultura")
        if mundo3D.state = 4 then
                mundo3D.resetworld()
                go to frame "Principal"
            else go the frame
        end if
end
```

4. Crear la programación para el fotograma de programación.

4.1. Crear un frame Script para el fotograma Principal llamado Bucle y escribir el siguiente código:

```
on exitFrame me
  go the frame
end
```

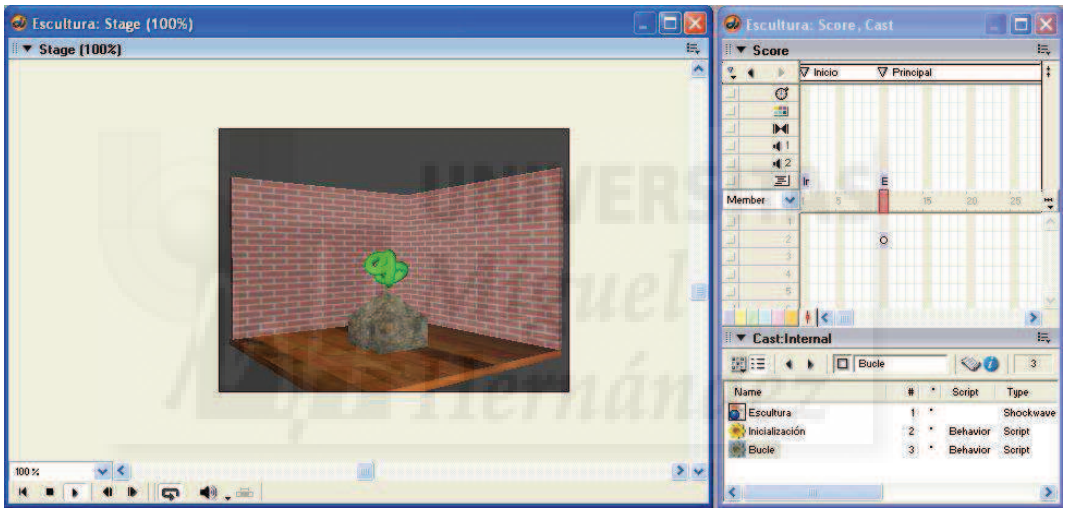


Imagen 5.8.cp.1.4: Pantalla de la inteface para este caso práctico

El código crea un “bucle sin fin”. A la altura de este mismo fotograma pero asignándolo al sprite del mundo 3D importado, añadiremos el script de programación que denominaremos “Programación” que contendrá toda la programación principal de la película, pero como queremos una película base, de momento lo dejaremos tal cual.

5. Publicarlo en un proyector para comprobación.

5.1. Hacemos File > Publish Settings > Projector > Publish. Esto producirá la creación de un fichero llamado “escultura.exe”.

5.2. Salir de Director y ejecutar el fichero “escultura.exe” para comprobar que aunque no hace nada, ya que nada hemos programado, pero podemos comprobar que al menos permanece constantemente en pantalla gracias al “bucle sin fin”.

A partir de aquí, con la estructura bien montada podemos dedicarnos a crear el resto de la programación para la escena.

Conclusiones:

Este caso práctico es de vital importancia para el resto del curso, ya que nos vamos a basar en la película creada para posteriores desarrollos. Director y su particular forma de organizar el trabajo hace que tengamos que adaptar su recursos para utilizar Shockwave 3D. Debemos apuntar que no tenemos otra alternativa para trabajar en tres dimensiones con este programa y de este mismo modo trabajan todos los autores que utilizan este software, desde los que realizar arte a los que crea videojuegos.

Caso práctico 8.2: Transformaciones básicas de objetos 3D.

Objetivo: Programar nuestro mundo 3D con Lingo o Actionscript para ir adquiriendo un control absoluto sobre todo lo que sucede en nuestro entorno. Comenzaremos aplicando transformaciones sobre algunos objetos en particular.

Tiempo de realización: 30 minutos.

Pasos a realizar:

1. Crear un comportamiento para programar nuestra escultura.
2. Dónde programar, la ventana Script.
3. Crear un disparador por programación.
4. Crear una acción por programación.
5. Transformación de un objeto.

1. Crear un comportamiento para programar nuestra "escultura".

Para poder realizar mejor este caso práctico, modificaremos nuestro último caso práctico 8.1. Crearemos un comportamiento asociado al sprite de la escena 3D. Este script centralizará toda nuestra programación.

1.1. Crear una copia de nuestro anterior fichero .DIR y llámala "escultura_transform".

1.2. Abrimos la ventana Score y abajo en Cast seleccionamos y borramos todos los scripts menos "Inicialización" y "Bucle".

1.3. Seleccionar la casilla donde esté el Sprite de nuestra escena 3D.

1.4. Ir a la ventana "Code" y en la ficha "Behavior Inspector" y pulsar sobre el botón "+". Ejecutar el comando "New Behavior". En la ventana que aparece escribir el nombre que queramos, yo he escrito "Programación". Aparecerá como un nuevo miembro del Cast.

1.5. Para asegurarnos que este comportamiento esta enlazado con el Sprite 3D podemos seleccionarlo y arrastrarlo hasta este sprite.

2. Dónde programar, la ventana Script.

2.1. En el Cast, seleccionar el comportamiento "Programación".

2.2. Pulsar sobre el botón "Cast Member Script" que  se encuentra en la parte superior de la lista de miembros (castmembers).

Aparecerá la ventana "Script" que se puede apreciar en la imagen 5.8.cp.2.1.

Esta ventana es básicamente un editor de textos donde escribir la programación. Se pueden apreciar una serie de controles que no tienen mucho sentido salvo para programadores avezados. A lo largo del curso veremos algunas de sus características. Como ya adelantamos en la unidad teórica, el control donde pone Lingo es muy importante, ya que aquí se puede elegir entre Lingo y JavaScript como lenguaje de programación y es evidente que toda la sintaxis de la programación que escribamos estará supeditada a esta elección.

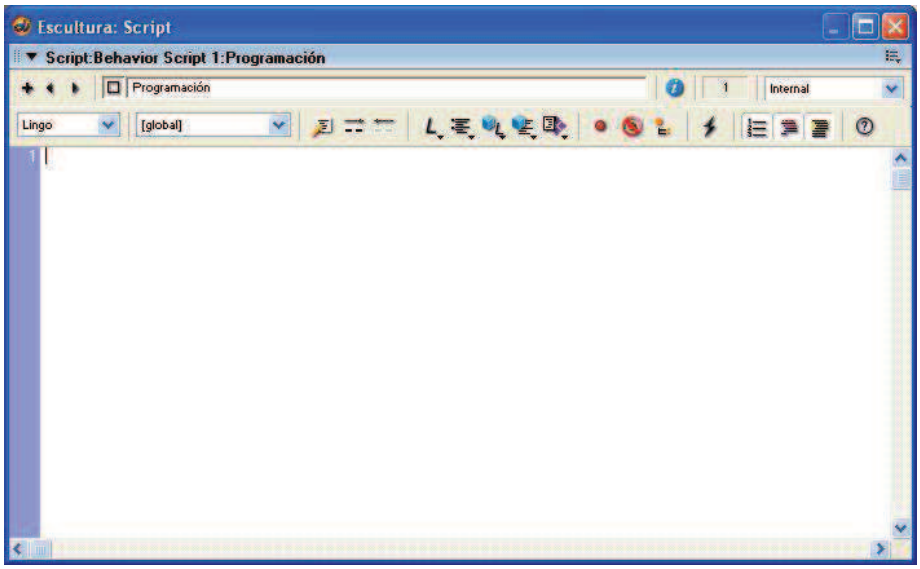


Imagen 5.8.cp.2.1: Ventana de programación Script

3. Crear un disparador por programación.

Comencemos a programar. Lo primero que vamos a escribir es la variable global que definíamos en el módulo inicialización para tener acceso a los datos del mundo 3D.

3.1. Abrimos el script “Programación” y escribimos:

```
global mundo3D
```

3.2. Utilizaremos el disparador mouseDown para que el usuario pueda interactuar muy básicamente con la película. Este disparador se activa cuando sucede el evento de que el usuario hace clic en cualquier lugar de la escena 3D. Escribir las siguientes sentencias.

```
on mouseDown

        -- aquí irán las acciones
end
```

3.3. Ejecutar la película, pulsando el botón “Play” para comprobar los cambios. Vemos que aunque pulsemos repetidamente el botón izquierdo del ratón no pasa nada, pues nada le hemos dicho que haga.

4. Crear una acción por programación.

Por ejemplo, vamos a intentar mover cualquier objeto de la escena de posición. Esto puede parecer fácil, pero primero debemos saber cómo se llama el objeto a mover y luego cómo moverlo. Ya vimos en la parte teórica como resolver estas dos cuestiones.

4.1. Escribir en el hueco para la acción:

```
mundo3D.model("Torus knot01").Traslate(0,0,-10)
```

4.2. Ejecutar la película, pulsando el botón “Play” para comprobar los cambios.

Vemos que al pulsar sobre cualquier parte del mundo 3D, y no solo sobre la escultura, esta se mueve. Si seguimos pulsando se seguirá desplazando hasta desaparecer de la escena.

5. Transformación de un objeto.

5.1. Escribir en el script el siguiente código:

```
global mundo3D

on mouseDown
  mundo3D.model("Torus knot01").Scale(.5, .5, .5)
end

on mouseUp
  mundo3D.model("Torus knot01").Scale(2, 2, 2)
end
```

En este caso utilizamos la función Scale para escalar el objeto a la mitad cuando se hace clic sobre la escena 3D y se vuelve a escalar al doble cuando se suelta el ratón. De esta manera se puede ver los efectos y al final no se modifica nada de la escena.

Debemos recordar que la función Scale de los objetos no permite coordenadas de cambio absolutas, siempre son relativas a la escala del objeto.

En este punto recomendamos que se experimente con las transformaciones básicas que son traslate (traslación), rotation (rotación) y scale (escala) y con las coordenadas relativas al objeto y las universales para poder llegar a tener un control mayor sobre los objetos de la escena.

5.2. Ejecutar la película pulsando el botón “Play” para comprobar que cuando se pulsa el botón del ratón, se produce el disparador mouseDown, y se escala la escultura en un factor para los 3 valores de .5, esto significa 0,5, es decir, la escultura se escala a la mitad.

Cuando se suelta el ratón, se produce el disparador mouseUp y lo que se hace es agrandar la escultura el doble, con lo cuál se deshace el anterior cambio. Si no programamos este disparador, veremos a la escultura empequeñecer hasta desaparecer de nuestra vista.

Conclusiones:

La primera y más importante conclusión que podemos extraer de este caso práctico es que programar no es una tarea difícil en sí misma, por lo que no debemos tener miedo a la hora de profundizar en este tema. Además es imprescindible para tener control sobre el mundo 3D que hemos creado. En este trabajo hemos aprendido a utilizar variables globales y también hemos visto las sentencias que permiten modificar objetos concretos.

Unidad 9: Interacción en Director.

Introducción teórica:

1. Introducción a la interacción mediante programación.
    - 1.1. Interactividad básica.
    - 1.2. Interfaces 2D-3D. Funcionamiento y diferencias.
    - 1.3. Interacción interna y externa a la obra. Formas de observación.
2. Creación de interactividad mediante comportamientos predefinidos.
    - 2.1. Qué es un comportamiento (behavior) y su utilidad.
    - 2.2. Cómo se utilizan los comportamientos.
3. Programación de interacción avanzada con los objetos.
    - 3.1. Interacción con la obra mediante controles de tipo botón on/off
    - 3.2. Seleccionar objetos 3D con el ratón 2D.
    - 3.3. Control de objetos mediante el teclado.
    - 3.4. Interacción interna a la obra. Formas de modificación.
    - 3.5. Añadir elementos multimedia.
    - 3.6. Manipular texturas.
    - 3.7. Ejecutar animaciones internas.

Casos prácticos:

- 9.1. Comportamientos predefinidos.
- 9.2. Interacción avanzada de objetos.

1. Introducción a la interacción mediante programación.

La interactividad es una de las condiciones que deben cumplir nuestros trabajos. Esta característica proporciona libertad al usuario para ver la obra desde distintos ángulos y al autor para definir cómo quiere que su trabajo sea percibido. Esta interactividad no existe directamente, sino que debe ser creada por el autor o autores de la obra.

Normalmente la interactividad se crea programándola directamente en un lenguaje de programación como Lingo, ActionScript, JavaScript, u otros, pero en Director tenemos la alternativa de poder utilizar los llamados comportamientos predefinidos que nos permiten aportar la interactividad más usual sin programar y en poco tiempo, aumentando la productividad.

De todas formas, no podemos olvidarnos que programar directamente, nos proporciona la herramienta más flexible y potente para individualizar nuestro trabajo y obtener un control total sobre todos los aspectos de la obra.

Por lo tanto, tenemos dos formas no excluyentes entre sí de elaborar la interactividad en Director: mediante programación o mediante los comportamientos predefinidos.

1.1. Interactividad básica.

Lo primero que tenemos que tener en cuenta es que toda interactividad en un mundo 3D se va a ver complicada por el hecho de que los periféricos que utilizamos son 2D. El periférico de visión, el monitor, es 2D y el de acción, el ratón, también es 2D. Por lo tanto, será complicado de por sí el determinar sobre todo la profundidad: el tercer eje del que carecen estos periféricos.

Normalmente hay tres grandes aspectos en el que se ejerce la interactividad, es decir, donde el usuario puede manipular el mundo 3D.

1) La cámara. Las operaciones más normales son la traslación y la rotación en los tres ejes. Además cabe la posibilidad de crear y cambiar de una cámara a otra, pudiendo ver la misma escena desde distintos puntos de vista.

2) La modificación de los modelos por traslación, rotación y/o escala en cada uno de los tres ejes. Estas operaciones se producen por manipulaciones a las que estamos acostumbrados en la informática de hoy en día como la selección por ratón, el arrastre del objeto, etc.

3) La creación de modelos geométricos básicos como pueden ser cajas, esferas, planos,… y también luces, cámaras, etc.

Todas estas interacciones se pueden llevar a cabo mediante programación en Lingo sin problemas y algunas se pueden realizar mediante comportamientos definidos.

1.2. Interfaces 2D-3D. Funcionamiento y diferencias.

El interfaz es un concepto muy amplio. Vamos a centrarnos en los interfaces más sencillos para estudiar una interactividad clásica. El interfaz de cualquier aplicación (ya sea para la web, para ofimática, etc.) funciona con múltiples controles, pero sin duda el más utilizado es el botón. Podríamos utilizar botones, por ejemplo, para ver las distintas obras de un mismo autor.

Como bien saben los diseñadores multimedia, se emplean muchos trucos para “representar” el funcionamiento de un botón real en el mundo virtual del ordenador. Normalmente se basa en al menos tres estados del botón: normal, sobre y pulsado. Cada uno de ellos requiere además de efectos que aporten información al usuario.

El estado “normal” es como está el botón presentado, casi siempre contiene información escrita de su función.

El estado de “sobre” es aquel que se produce cuando el cursor pasa sobre el botón y por tanto está a un solo clic de ser pulsado. Normalmente se denota de alguna forma que el usuario está en esta situación, por ejemplo, cambiando su forma o iluminándolo.

El estado “pulsado” es el que se produce cuando se hace clic sobre el botón. Lo normal es que se simule que baja el botón, a veces se enriquece este efecto con la ejecución de un pequeño efecto de sonido. Por último se ejecuta la acción para la que ha sido concebido todo el mecanismo.

Llevando los botones a un mundo 3D, es interesante estudiar que las posibilidades son mucho más sofisticadas. Por ejemplo, si el botón sirve para pasar de un entorno 3D a otro, (o en nuestro caso de una obra virtual a otra) se puede presentar el botón como una simplificación de ese entorno y que gire lentamente como incitando al usuario a visitarlo. El estado de “sobre”, se puede realizar texturizando ese modelo o iluminando, etc. Y por último el estado de “pulsado” puede representarse por un escalado del botón y un efecto sonoro asociado o por algo más espectacular como la visualización de un sistema de partículas.

Ni qué decir tiene que también podemos simplificar toda la interface modelando los botones con simples cajas y las texturas correspondientes.

1.3. Interacción interna y externa a la obra. Formas de observación.

Definimos la interacción externa a la obra como la interacción que se produce entre la obra y el espectador en contraposición a la interacción interna que sería la que se produce entre los elementos de la propia obra. Por ejemplo, si un espectador está visualizando una escultura móvil y cambia de punto de vista, es interacción externa, pero si la escultura emite un sonido metálico porque parte de esta escultura ha golpeado a otra parte, esto es interacción interna. Esta interacción interna forma parte del trabajo mismo del artista, ya que forma parte del mensaje inherente a la obra, de lo que el autor quiere transmitir.

Se puede pensar que la interacción externa se puede desarrollar fuera del ámbito del autor, por ejemplo, por programadores de interfaces, pero somos de la opinión que el autor debería involucrarse también en la creación de la forma de ver la obra por parte de los espectadores.

Podríamos decir que, en principio, la interacción externa se podría realizar con comportamientos predefinidos ya que es muy frecuente y la interna al ser más específica de cada obra es más adecuada para el desarrollo mediante programación en Lingo.

Para visualizar una obra de arte tenemos, en teoría, múltiples posibilidades al disponer de muchos “ojos”, es decir, cámaras disponibles en nuestro entorno. Lo ideal sería disponer cámaras en los puntos donde el artista quiera realzar la obra y luego dar libertad al usuario para usarlas como le convenga. En algunos casos, para no hacer perderse a usuarios neófitos en las 3D se podría limitar y mostrar muy controladamente los movimientos de la cámara, por ejemplo, solo permitir movimiento horizontal, vertical o circular.

Un punto importante es dotar a la presentación de un botón fuera del entorno 3D que reinicie el mundo 3D y por tanto, asegure el comienzo de la incursión en el mundo 3D en un lugar conocido. Es la misma función que el famoso botón “Home” de la web.

2. Creación de interactividad mediante comportamientos predefinidos.

En este punto vamos a tratar los siguientes aspectos:

- Qué es un comportamiento (behavior) y su utilidad.

◆ Cómo se utilizan los comportamientos.

La idea de comportamientos predefinidos entronca con el paradigma moderno de programación: la POO o programación orientada a objetos. Esta programación se basa en la reutilización de código, es decir, en "no reinventar la rueda" sino apoyarnos en el trabajo de otros programadores. La idea es que casi todo el mundo en las escenas 3D necesita, por ejemplo, que una cámara haga un zoom. Pues bien, en vez de que cada programador desarrolle esta tarea individualmente, se trata de que lo implemente alguien y que después lo ponga a disposición de otros autores. Normalmente, estas utilidades se agrupan formado las llamadas "bibliotecas".

En nuestro caso particular, esto es aún más útil, ya que podemos obviar gran parte de la complejidad de "cómo" hacer que funcionen las cosas, es decir, de cómo programarlas y centrarnos más en el "qué" queremos que hagan.

Además, este modelo de desarrollo, es colaborativo y hace que muchas personas puedan trabajar de forma eficiente en un mismo proyecto.

2.1. Qué es un comportamiento (behavior) y su utilidad.

Lo primero que debemos aclarar es que Director denomina behavior a todo script, ya sea escrito por un programador o aportado por Director. De hecho, en el fondo es lo mismo, ya que en el interior de un comportamiento se encuentra un código escrito en un lenguaje de programación llamado Lingo. La diferencia, sin embargo, es muy importante, ya que el script lo ha realizado un programador independiente, mientras que el comportamiento integrado viene escrito de "fábrica", es un programa encapsulado con un propósito específico.

Esto quiere decir que el script (si está bien escrito) hará exactamente lo que el programador quiere, lo cual le da un poder muy importante para realizar su propósito. Cuando se usa un comportamiento, este realizará funciones rutinarias o muy generales, lo bueno es que pueden ahorrar tiempo para aplicaciones muy usuales, pero no permiten su modificación para personalizarlos y adaptarlos a las necesidades que se tenga en un trabajo en particular.

Por ejemplo, una exposición de arte virtual puede orientarse de forma general. El modo y las necesidades de la visualización de un "cuadro virtual" pueden ser las mismas para distintos cuadros. Lo mismo ocurre con la visión de una escultura tridimensional. Hará falta por ejemplo, el poder acercarse o alejarse del objeto, rodearlo sin tocarlo o girarlo. En este caso pueden ser muy útiles los comportamientos.

Si el arte virtual es interactivo en sí mismo (lo que antes denominamos interacción interna a la obra), los comportamientos no nos sirven. De la misma manera que el artista real elegirá para una escultura entre el bronce y el mármol, el artista virtual tendrá que texturizarla en bronce o mármol, pero también tendrá que manipular la interacción, tendrá que fabricarla ex profeso para esa obra si forma parte de la obra misma.

2.2. Cómo se utilizan los comportamientos.

Los comportamientos están organizados en una biblioteca dentro de Director.

Hay dos tipos de comportamientos: acciones (actions) y disparadores (triggers).

◆ Acciones: las acciones son comportamientos que definen lo que sucede con el mundo 3D cuando el usuario ha ejecutado una interacción por medio del ratón o el teclado, es decir, la acción es la consecuencia de la intervención del usuario. Por ejemplo si utilizamos la acción "Dolly Camera" lo que haremos es mover hacia delante o hacia atrás la cámara con la que vemos el mundo 3D.

◆ Disparadores: estos comportamientos definen una acción por parte del usuario. Por ejemplo, el trigger “Mouse Left” se produce cuando el usuario pulsa el botón izquierdo del ratón. Un trigger siempre se enlaza con una acción.

Siguiendo el ejemplo, si enlazamos el trigger “Mouse Left” con la acción “Dolly Camera”, esto supondrá que cuando se pulse el botón izquierdo la cámara se moverá hacia delante hasta que soltemos el botón.

Por lo tanto, la idea es que las acciones del usuario disparan un trigger y a este se le enlaza una acción que será la respuesta. De esta manera se produce una acción o respuesta del software a una acción del usuario, o sea, la interacción usuario-software. Aunque estas relaciones pueden llegar a ser muy complejas, no contemplan sino una mínima cantidad de todas las posibilidades que podemos necesitar en trabajos complejos como videojuegos, etc. Por ello, no podremos prescindir nunca de utilizar Lingo.

Las acciones y los disparadores los podemos encontrar en la ventana Code: Library. Elegimos 3D mostrando el menú que se muestra en la imagen de abajo.

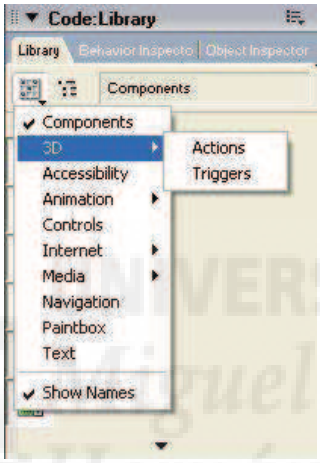


Imagen 5.9.1: Realizar acciones o disparadores

Las acciones (actions) disponibles son las siguientes:

◆ Dolly Camera: mover la cámara adelante y atrás.
◆ Drag Camera: permite el encuadre, el zoom y la rotación de la cámara.
◆ Fly Through: mueve la cámara a través de los objetos.
◆ Pan Camera Horizontal: traslada la cámara horizontalmente.
◆ Pan Camera Vertical: traslada la cámara horizontalmente.
◆ Orbit Camera: orbita la cámara alrededor del modelo.
◆ Reset Camera: coloca la cámara a su posición y orientación inicial.
◆ Rotate Camera: permite la rotación de la cámara.
◆ Automatic Model Rotation: rota automáticamente un modelo en un eje (no trigger).
◆ Clic Model Go To Market: va a un marcador cuando se hace clic a un objeto.
◆ Drag Model: arrastra un modelo con el ratón.
◆ Drag Model To Rotate: rota un modelo cuando se le arrastra con el ratón.
◆ Model Rollover Cursor: cambia el cursor al pasarlo encima de un objeto (no trigger).
◆ Play Animation: cuando se hace click comienza una animación importada.
◆ Create Box: crea una caja.
◆ Create Particle System: crea un sistema de partículas.
◆ Create Sphere: crea una esfera.
◆ Level of Detail: quita detalles a un modelo hasta un límite configurable.
◆ Sub Division Surface: añade detalles a un modelo hasta un límite configurable.
◆ Generic Do: ejecuta unas instrucciones Lingo al producirse el trigger conectado.
◆ Toggle Redraw: coloca on/off el redibujado.
◆ Toon: muestra los modelos con efecto de dibujo animado.
◆ Show Axis: muestra los ejes de los modelos.

Los disparadores (triggers) disponibles son los siguientes:

- Mouse Left: botón izquierdo del ratón.
- Mouse Right: botón derecho del ratón.
- Mouse Enter: produce un trigger cuando el ratón entra en un sprite 3D.
- Mouse Within: produce un trigger mientras el ratón esta en un sprite 3D.
- Mouse Leave: produce un trigger mientras el ratón sale de un sprite 3D.
- Keyboard Imput: entrada por teclado.

La forma de utilizarlos es primero eligiendo una acción, por ejemplo “Drag Model To Rotate”, y la arrastramos hasta el sprite del 3D castmember, en ese momento aparecerá una caja de diálogo para configurar la acción como la que se muestra en la Imagen 5.9.2.

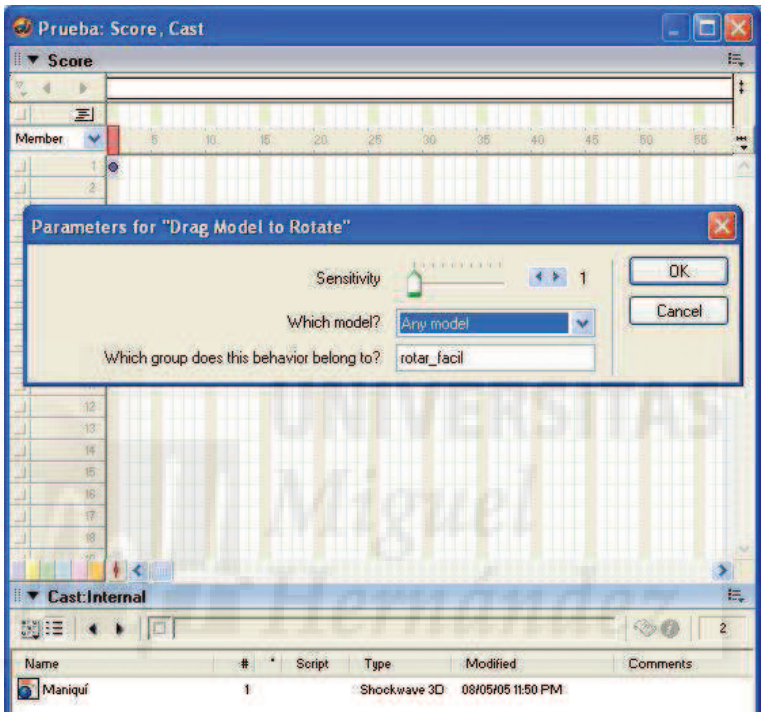


Imagen 5.9.2: Parámetros de una acción o action

En este caso en particular, tenemos tres opciones: sensibilidad, sobre qué modelo actúa el “Drag and rotate” y cómo nombramos a esta acción, en este caso “rotar_fácil”.

Seguidamente tenemos que elegir un trigger, por ejemplo “mouse left” que igual como antes arrastraremos hasta el mismo sprite del 3D castmember. Aparecerá una nueva caja de diálogo donde le diremos que está conectado con la acción anterior por su nombre, en este caso “rotar_fácil”. También podemos configurar si el trigger además se producirá por una combinación de teclas o si ejecutará una opción secundaria de la acción, por ejemplo, en “Dolly camera” podemos elegir entre mover la cámara hacia delante o hacia atrás.

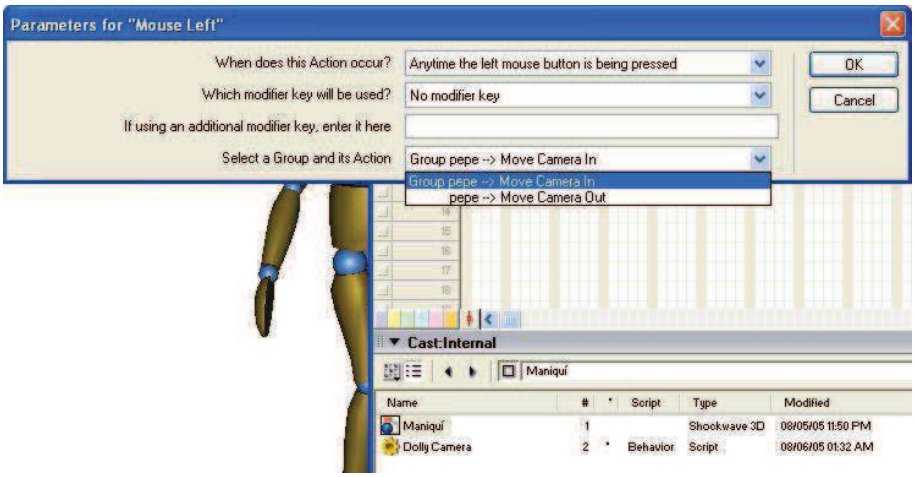


Imagen 5.9.3: Parámetros de un disparador o trigger

Y esto es todo. Al pulsar el botón “Play” de la ventana Stage podremos arrastrar el cursor del ratón para rotar cualquier elemento de la escena.

3. Programación de interacción avanzada con los objetos.

3.1. Interacción con la obra mediante controles de tipo botón on/off

Estos controles funcionan como un conmutador. Sólo tienen dos estados: encendido/on y apagado/off. Para pasar de un estado a otro requieren de la acción del usuario.

Con estos botones podemos crear buena parte de una interface, por ejemplo, un objeto 3D podría permanecer en reposo, en estado off, pero al pasar el ratón sobre él, cambiaría su estado a on y podría empezar a girar o cualquier otro efecto que se nos ocurra.

Para desarrollar estos controles, es necesario utilizar una variable que indique el estado del control. Esta variable será local, ya que solo la utilizaremos en el script asociado a la escena 3D que hemos denominado “Programación”. Por ello en vez de ser global, la declararemos local con la palabra reservada property. Por ejemplo, escribiremos:

```
property estado.
```

Seguidamente tendremos que incorporar al script dos disparadores para controlar el paso de un estado a otro. Estos disparadores definirán la interacción con el usuario. Las acciones de estos disparadores serán una simple sentencia que modifique el valor de la variable estado. Por ejemplo, escribiremos el siguiente código:

```
on mouseDown me
        estado = 1
end

on mouseUp me
        estado = 0
end
```

Esto hará que cuando se pulse el botón izquierdo del ratón, el estado tenga el valor 1 y se ejecutará el efecto, y cuando se deje de pulsar tome el valor 0, y el efecto deberá cesar.

Ahora debemos utilizar un mecanismo para que mientras el estado del control no varíe, haga que el efecto buscado no cambie. Ese mecanismo es el disparador “exitframe”, ya que la

película está constantemente entrando y saliendo del fotograma que nosotros hemos llamado "Principal" (recordemos que esto lo controlamos con el script "bucle")

En el interior de este disparador escribiremos las sentencias que producen el efecto, pero con la condición de que solo se ejecuten si el estado del control es on, o sea si estado vale 1. Esto lo codificaremos con una sentencia del tipo "if condición then sentencia". Así:

```
on exitframe
        if estado = 1 then mundo3D.model("Torus knot01").rotate(10, 10, 10)
end
```

En este caso, hacemos que mientras el estado valga 0, o sea, se esté pulsando el botón izquierdo del ratón, el objeto "Torus Knot01" no cese de girar.

3.2. Seleccionar objetos 3D con el ratón 2D.

Para aumentar el control sobre nuestra escena 3D necesitamos controlar con precisión la selección de los objetos que forman el mundo virtual. Por ejemplo, imaginemos una sala de exposiciones y una vista en perspectiva de ella y deseamos seleccionar una escultura entre muchas otras, posiblemente situadas muy cerca unas de otras, casi solapándose desde nuestro punto de vista.

El problema no es obvio, ya que tenemos un puntero que se lleva sobre la pantalla, o sea, un elemento 2D y queremos actuar sobre un mundo 3D.

Lo primero que necesitamos es una variable para que guarde el nombre del modelo seleccionado. Será una variable local, por ejemplo, si la llamamos "tocado", escribiremos:

```
property tocado.
```

Luego hay que crear un procedimiento que nos permita relacionar un clic en la pantalla con un modelo seleccionado. Pero este procedimiento existe ya y lo proporciona Director. Es un comando 3D de Lingo que se llama "modelUnderLoc". Su funcionamiento es un poco complicado de explicar pero lógico sin lugar a dudas. Vamos a explicarlo detalladamente.

En primer lugar este procedimiento pertenece a una cámara. Esta cámara es la activa por defecto (siempre hay una) del sprite de la escena 3D.

Este procedimiento necesita un punto para trabajar. Este punto debe estar incluido en el rectángulo visible del sprite que muestra la cámara.

ModelUnderLoc lo que hace es devolver la referencia al modelo más cercano a la cámara (pueden haber muchos modelos que se superpongan) que se encuentre en ese punto. Si no encuentra ningún modelo, este procedimiento devuelve el valor "void", que significa nada, vacío.

Como hemos comentado, el procedimiento necesita como entrada un punto, es decir necesita saber hacia dónde tiene que "dirigir" el cálculo. Nosotros queremos que este punto sea donde hacemos clic con el ratón.

Para obtener este punto vamos a utilizar una propiedad del objeto Mouse de Director. Este objeto tiene la propiedad "clicLoc" que identifica al último punto de la pantalla donde se hizo clic con el ratón. Hay otra propiedad muy utilizada del ratón que se llama "mouseLoc", que nos dice el punto actual del puntero.

Con todo lo dicho parece que deberíamos escribir algo como lo que sigue: camera.modelUnderLoc(clicLoc), pero nuestras dificultades aún no han acabado. Además la propiedad hay que referenciarla como "the clicLoc" ya que nos es dada por Director.

Una cuestión a resolver es que el punto que necesita el procedimiento es referenciado al origen del sprite, no de la pantalla, por lo que habrá que realizar una resta entre estos dos puntos y escribir como parámetro: the clickLoc - point(psprite.left,psprite.top)

Por último no hay que olvidar que tanto la cámara como el origen se refieren al sprite, y este no lo hemos referenciado. Para ello creamos una variable llamada por ejemplo "psprite" y luego utilizamos el disparador beginSprite para obtener el valor del sprite actual. Quedaría algo así:

```
property psprite
property tocado

on beginSprite me
        pSprite = sprite(me.spriteNum)
end
```

Luego después del disparador MouseDown deberíamos escribir:

```
tocado = psprite.camera.modelUnderLoc(the clickLoc - point(psprite.left,psprite.top))
```

Para terminar, falta saber cómo vamos a decirle al programa que si "tocado" es la escultura que queremos, ponga "estado" a 1 (on). Para esto hay que recordar que modelUnderLoc devuelve la referencia al modelo, es decir, todo el modelo. Tenemos una propiedad de los modelos que es el nombre, llamada "name" y tenemos un procedimiento para saber si una cadena de caracteres está dentro de otra llamado "contains". En el ejemplo veremos claramente como funciona. Por lo tanto escribimos:

```
if tocado.name contains "Elemento_a_seleccionar"  then  estado =1
```

Al ejecutar la película debe funcionar como sigue: si hacemos clic en un sitio que no sea el "elemento_a_seleccionar", (nuestra escultura), esta no realizará ningún efecto (por ejemplo no rotará), pero si le acertamos a hacer clic sobre ella, si lo realizará (rotará). Hay que señalar la precisión de este complicado método, ya que si hacemos clic entre huecos u orificios que pueda tener la escultura veremos que no se selecciona y por tanto, no girará

3.3. Control de objetos mediante el teclado.

Lingo tiene un procedimiento perteneciente al objeto "Key" de Director para controlar el teclado llamada keypressed. También podemos utilizar directamente "the keypressed" que es una propiedad que devuelve el carácter pulsado. Por ejemplo, si queremos saber si alguien pulsó la tecla "a" escribiremos:

```
if (the keypressed="a") then sentencia
```

Normalmente, necesitamos controlar mediante el teclado varias transformaciones a realizar en un mismo modelo, para ello haremos uso de la sentencia "case of" que permite sustituir una serie de if´s anidados. Por otra parte, el uso de case, conlleva el uso de la función "chartonum" que devuelve el código Ascii de una tecla ya que "case of" necesita como entrada datos numéricos y no teclas. Un ejemplo de utilización es el siguiente extracto del caso práctico 9.2 que ilustra estas ideas:

```
Case (chartonum (the keypressed)) of
        30: mundo3D.model[1].translate(0,0,10,#world) -- tecla flecha arriba
        31: mundo3D.model[1].translate(0,0,-10,#world)-- tecla flecha abajo
        28: mundo3D.model[1].translate(-10,0,0,#world)-- tecla flecha izquierda
        29: mundo3D.model[1].translate(10,0,0,#world) -- tecla flecha derecha
        97: mundo3D.model[1].translate(0,10,0,#world) -- tecla A
        98: mundo3D.model[1].translate(0,-10,0,#world)-- tecla B
end case
```

Donde 30, 31, 28,… se corresponden con el código Ascii de la tecla que se comenta a la derecha.

3.4. Interacción interna a la obra. Formas de modificación.

La obra en sí misma puede contener interacción. Por ejemplo, puede que sea una escultura con alguna parte móvil y que el artista pretenda que el usuario la pueda manipular.

En esta caso, hay que tener en cuenta que se podrán aplicar las transformaciones que queramos a cada uno de los elementos geométricos que componen la obra. Cuando se importa una escena de un programa externo como MAX, cada uno de los cuerpos tridimensionales son individuales. De todas formas, se pueden agrupar algunos de ellos o todo e importarlo así y Director lo respetará. Pero es mejor utilizar Lingo para agrupar los elementos geométricos que queramos y poder tener la posibilidad de referirnos a un conjunto o sus partes cuando nos convenga con las posibilidades que esto conlleva.

En Lingo podemos crear un grupo con la sentencia “newgroup”, que necesita como parámetro un nombre. Para añadir elementos al grupo se utiliza la función “addChild” que también necesita un nombre como parámetro. En el siguiente ejemplo se crea el grupo pierna y se incluyen sus partes.

```
gWorld.newGroup("pierna")
gWorld.group("pierna").addChild(“muslo”)
gWorld.group("pierna").addChild(“rodilla”)
gWorld.group("pierna").addChild(“pantorrilla”)
gWorld.group("pierna").addChild(“pie”)
```

Debemos tener en cuenta que las manipulaciones pueden ser complejas, por ejemplo, manipulando animaciones internas a la obra. Supongamos que hemos realizado una mariposa que mueve sus alas pero queremos que solo lo haga cuando le hagamos clic con el ratón. Otra posibilidad es que añadamos efectos sonoros.

Podemos tener relaciones entre los elementos geométricos muy complejas, como puede suceder al modelar el cuerpo de un humano y dotarle de huesos y la animación de esos huesos. Sabemos que en la realidad al estirar de un dedo, esto conlleva el movimiento de todo el brazo en consecuencia, esto se conoce como cinemática inversa. Por otro lado, en el ejemplo anterior, al doblar el dedo no lo podemos hacer 360º puesto que no somos de goma. Como se puede entender fácilmente, es un tema complejo ya en su origen.

En 3D Studio MAX se utilizan dos técnicas para realizar esta clase de trabajos: la creación de huesos y Character Studio. Con los huesos, el animador crea todo el sistema óseo como desee, pero es un trabajo muy arduo, ya que tiene que definir para cada unión de unos huesos con otros, el grado de libertad, longitud de estos, orientación, etc. En contraprestación, tiene total libertad y sirve para toda clase de organismos. Con Character Studio el trabajo se hace más fácil y rápido, ya que se parte de un esqueleto ya definido que solo tenemos que configurar, pero solamente sirve para animar bípedos. Tanto una como otra alternativa se pueden manejar en Director mediante Shockwave 3D.

Otra posibilidad es que aparezcan dinámicas en nuestra escultura, por ejemplo, una pelota que cae de cierta altura y rebota. O que una escultura móvil no se mueva por interacción directa del usuario, sino por el viento que entra por una ventana que el usuario puede abrir o cerrar en la escena 3D. Esto también lo podemos trabajar con Shockwave 3D.

De todas formas, estas últimas características son muy avanzadas para el presente curso, por lo que nos centraremos en las más comunes en los próximos puntos.

Por último, los sistemas de partículas como agua, humo, fuego… podemos crearlas directamente con Lingo. Estos efectos si los estudiaremos en una unidad más adelante.

3.5. Añadir elementos multimedia.

Los elementos básicos multimedia son los gráficos, los sonidos, las animaciones y los vídeos. Estos elementos se combinan para hacer presentaciones y/o interfaces. Todos estos elementos se pueden realizar fuera de Director en programas específicos para su manipulación y se importan a esta plataforma donde se combinan juntos. Para importarlos se ejecuta el comando File > Import.

En Director podemos incluir todos estos elementos para enriquecer la exhibición de nuestra obra o para dotar a la propia presentación de funcionalidad. Estos elementos tienen que verse y por tanto ocuparán parte del Stage lo que supone que este deberá tener un tamaño mayor que el del 3D castmember.

Por ejemplo, podemos añadir en la ventana Stage elementos gráficos para que hagan el papel de botones. Es evidente entonces que el Stage no puede estar totalmente ocupado por el 3D Castmember.

Otro caso es ejecutar sonidos. Esto lo podemos hacer con las sentencias:

```
sound(1).play(member("sonido_1"))      -- o con esta otra que es equivalente:

puppetsound(1,"sonido_1")              -- para quitar el sonido haremos: puppetsound(1,0)
```

Los demás elementos multimedia funcionan de una forma muy parecida: debemos importarlos e incluirlos en el score antes de utilizarlos.

3.6. Manipular texturas.

Las texturas son como la piel de los modelos geométricos. No es simplemente una cuestión estética sino que puede cambiar todo el mensaje de la obra de arte. No es lo mismo que una obra sea de metal que de agua. Hay que distinguir entre los conceptos de Shader, superficie y textura.

El shader determina las superficies de los objetos, es decir si son metálicas como el cromo, si son mate como un melocotón, si son semitransparentes como el vidrio tintado, etc. Por tanto, definen la naturaleza del material del objeto.

Las texturas definen el dibujo de la superficie, por ejemplo un objeto puede ser metálico pero puede estar oxidado o ser más pulido, puede ser de oro viejo o acero anodizado.

Las superficies son lo que se visualiza del objeto, y por tanto, la combinación del shader y las texturas aplicadas.

Las texturas de un objeto se pueden asignar en tiempo de ejecución. Esto es un buen efecto que podemos utilizar para que el usuario pueda incluso elegir el material con el que se representa una obra 3D. Si estamos mostrando cuadros, es de mucha ayuda ya que la textura es el lienzo mismo, con lo cuál sin cambiar de vista podemos ver en el mismo soporte distintas imágenes.

Para disponer de distintas texturas para un mismo objeto las tenemos que incorporar al mundo 3D desde los members. Suponiendo que tengamos miembros bitmap con las texturas y que se llamen “textura_1” y “textura_2”, y que queramos que dentro de la escena 3D se llamen “textura_a” y “textura_b”, escribiremos las siguientes instrucciones:

```
gWorld.newTexture("textura_a", #fromCastMember, member ("textura_1"))
gWorld.newTexture("textura_b", #fromCastMember, member ("screen_2"))
```

Por último la asignación de la textura al modelo se hace con la instrucción:

gWorld.shader("modelo_1").texture = gWorld.texture("texture_a")

En Lingo también se pueden utilizar efectos especiales basados en las texturas animadas, pero este es un tema que veremos más adelante.

3.7. Ejecutar animaciones internas.

Las animaciones internas son las que contiene la propia obra. Shockwave 3D llama a estas animaciones “Motion”. Los motions son animaciones incrustadas en archivos .w3d que han sido exportadas desde programas de animación externos.

Las animaciones se pueden controlar con Lingo o se pueden manipular con Comportamientos. En este caso la diferencia tiene que ver exclusivamente con la interactividad. Si se quiere no intervenir en la animación es mucho más fácil ejecutarla desde el comportamiento “Play Animation”, ya que como se puede apreciar en la imagen que sigue tiene muchas opciones para adaptarlas a nuestras necesidades. Si se quiere intervenir en la reproducción de la animación de forma no pasiva deberemos utilizar Lingo como se expone a continuación.

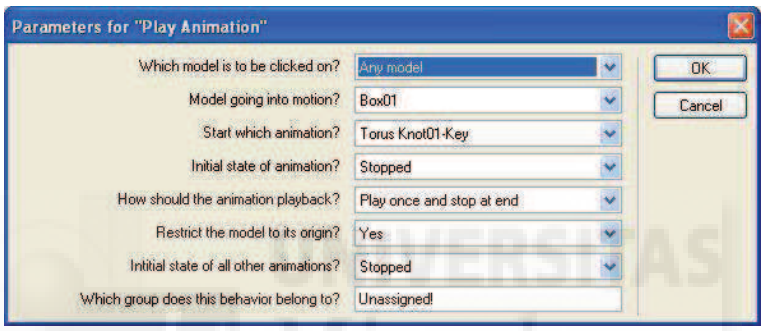


Imagen 5.9.4: Ejecución de animaciones internas

Un “motion” es un recurso especial que almacena datos sobre una animación. Hay dos clases de motions:

1. Animación de fotogramas: controlan por fotogramas la posición, rotación y escala de los modelos. No se pueden crear con código Lingo, tienen que importarse. Lingo los puede controlar por medio del modificador “Keyframeplayer” que estudiaremos un poco más adelante.

2. Animación de huesos. Controlan la animación de modelos por medio de unos modelos especiales que tiran de ellos llamados huesos o “bones”. Lingo los controla por medio del modificador “Bonesplayer”.

El modificador “Keyframeplayer” es muy interesante porque aunque se pueda haber realizado la animación para un objeto, lo que se importa es la animación en sí, no la animación pegada al modelo, por lo que la misma animación puede ser utilizada para muchos modelos dentro de un único entorno 3D.

El modificador “Keyframeplayer” debe ser añadido a la lista de modificadores de un modelo en particular. Esto se hace con una sentencia como la que sigue:

Scene.model[x].addmodifier(#Keyframeplayer)

Keyframeplayer permite un control básico de las animaciones de fotogramas como si fuera un vídeo. Para ejecutar una animación escribiremos:

Scene.model[x].Keyframeplayer.play( )

Y para pararla momentáneamente:

Scene.model[x].Keyframeplayer.pause( )

También podemos controlar la velocidad de la ejecución de la animación. Para ello utilizaremos la sentencia:

Scene.model[x].Keyframeplayer.playrate = 1

Esto hará que tenga una velocidad normal. Si se pone un número mayor que 1 se conseguirá un factor de multiplicación de la velocidad, si se quiere más lentitud de lo normal se pondrá un número entre 0 y 1 y si se quiere que vaya hacia atrás se hará con –1.

Podemos parar la animación al utilizar la variable “currentime” que va contando el tiempo de reproducción del movimiento medido en milisegundos. Si le asignamos el valor 0 la animación volverá a su inicio:

Scene.model[x].Keyframeplayer.currentime = 0

También podemos saber si la animación se está ejecutando o no con el valor booleano “playing”. En definitiva, el control de animaciones es bastante simple con Lingo pero hay que recordar que la animación en sí, se tiene que crear fuera de Director.

Caso práctico 9.1: Comportamientos predefinidos.

Objetivo: Integrar en nuestro trabajo práctico los comportamientos predefinidos con el fin de obtener resultados satisfactorios en poco tiempo, aumentando la productividad.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Preparación de la película.
2. Integración de los comportamientos predefinidos: Acciones y Disparadores.
3. Comportamientos para la interacción con un objeto.
4. Comportamientos para la interacción con una cámara.
5. Comportamiento genérico.

1. Preparación de la película.

Para poder realizar mejor este caso práctico, vamos a modificar nuestra escena del caso práctico 8.1: “Estructura básica de la película”.

1.1. Ejecutar MAX con el objeto de modificar el mapa asignado a la escultura. También vamos a mover un poco la vista de perspectiva para acercarnos más al objeto “escultura” y finalmente, exportamos la nueva escena con el nombre “escultura_nueva_vista”.

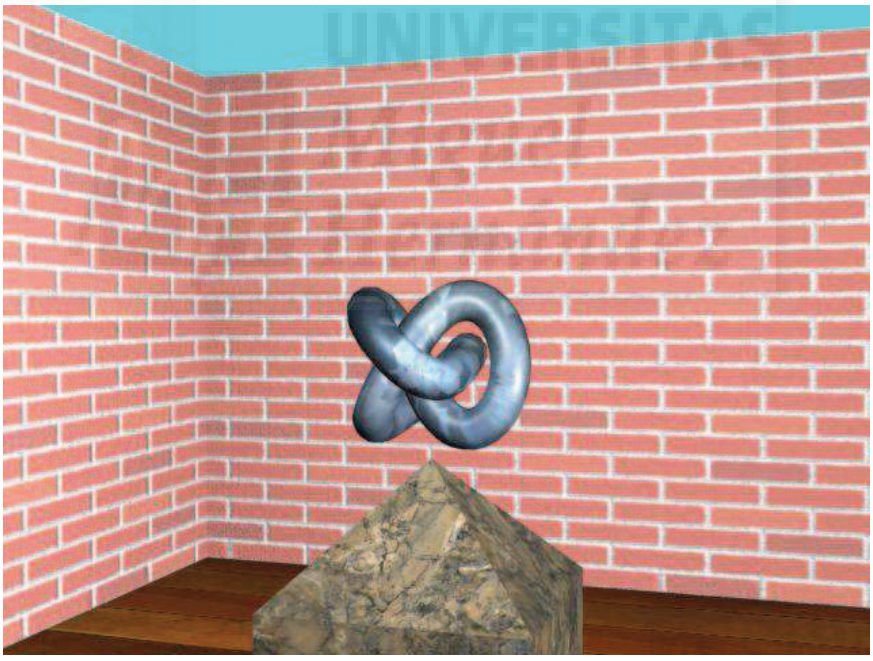


Imagen 5.9.cp.1.1: Pantalla de la obra con la que trabajamos en este caso práctico

1.2. Ejecutar Director. Importar el nuevo fichero .W3D con File > Import.

1.3. Accedemos a la ventana de “Score” y en el “Cast” borramos el miembro “escultura”.

1.4. Abrimos el script “Inicialización” y cambiamos la línea donde asignamos el miembro a la variable global mundo3D, por lo tanto quedará así:
mundo3D = member("escultura_nueva_vista")

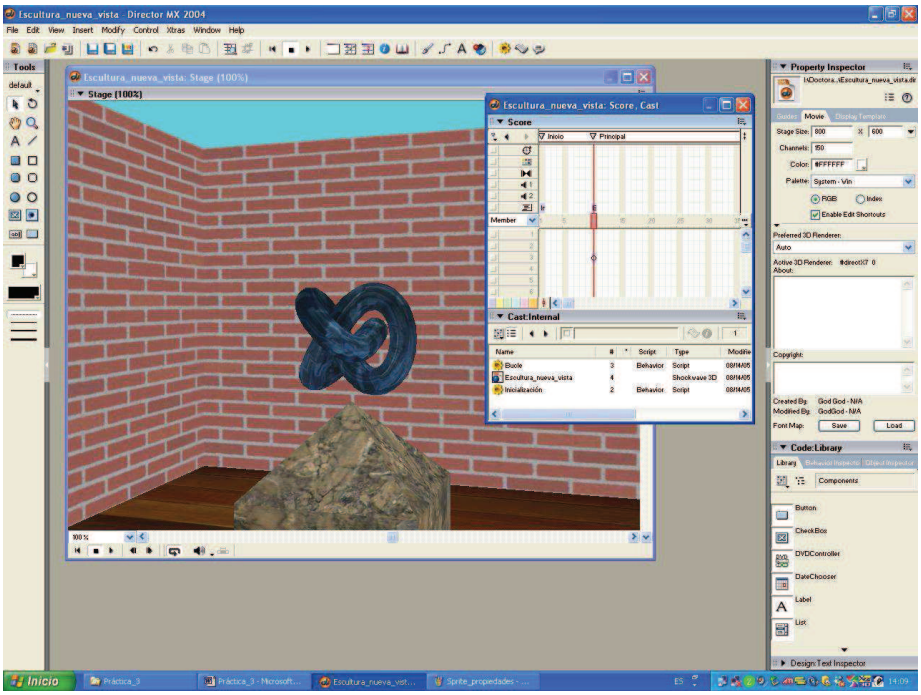


Imagen 5.9.cp.1.2: Pantalla de la obra nueva en Director

2. Integración de los comportamientos predefinidos: Acciones y Disparadores.

Los comportamientos fueron ampliamente explicados en la unidad teórica correspondiente. Ahora en este caso práctico lo que más debemos tener en cuenta es que siempre se deben utilizar por parejas: una acción siempre estará acompañada de un disparador.

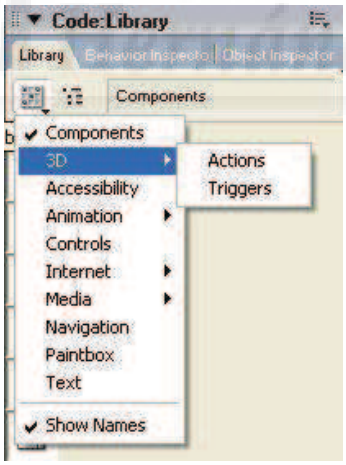


Imagen 5.9.cp.1.3: Librería de código: acciones y disparadores

La secuencia que tendremos que realizar siempre será la misma:

1. Elegir la acción que corresponda.

2. Arrastrarla sobre el Sprite de la escena. En este caso sobre el fotograma 10.

3. Aparecerá una caja donde configuramos la acción y le asignamos un nombre.

4. Elegir el disparador que hará que se ejecute la acción.

5. Arrastrarlo sobre el Sprite de la escena.

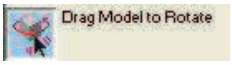
6. Aparece una caja donde lo personalizamos y elegimos la acción que se ejecutará.

Tanto las acciones (actions) como los disparadores (triggers), se encuentran en la librería de código como muestra la imagen 5.9.cp.13. Una vez elegido entre acciones o disparadores aparecerá una lista para que seleccionemos el que necesitemos.

3. Comportamientos para la interacción con un objeto.

Vamos a realizar una interacción con la propia escultura (el objeto creado a partir de la primitiva Torus Knot). La interacción consiste en poderlo girar con el botón izquierdo del ratón, de tal forma que no se traslada (es decir, no cambia de lugar y permanece siempre encima del pedestal) pero puede girar sobre sí mismo.

Esto permite visualizar desde todos los puntos de vista el objeto. No es muy real, ya que en un museo no movemos la escultura a nuestro antojo, sino que nos movemos nosotros, pero las características en el plano virtual son diferentes, además, este tipo de interacción se utiliza mucho para la muestra y el diseño de productos de diseño industrial.

3.1. Acceder a las acciones, seleccionar Drag Model 

to Rotate y arrastrarla hasta el sprite del mundo3D.

3.2. Aparecerá la caja de diálogo de la imagen que sigue.

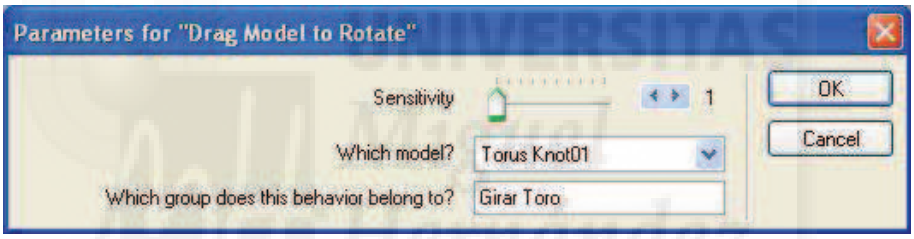
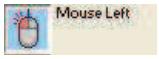


Imagen 5.9.cp.1.4: Caja de diálogo para "Arrastrar modelo a rotar"

3.3. Elegir "Torus Knot01" (la escultura) para que se mueva este objeto y no otro.

3.4. Asignarle cualquier nombre. Nosotros hemos escrito "Girar Toro".

3.5. Acceder a los disparadores, elegir "Mouse Left" y 

arrastrarlo hasta el sprite del mundo3D.

3.6. Aparecerá la caja de diálogo de la imagen siguiente.

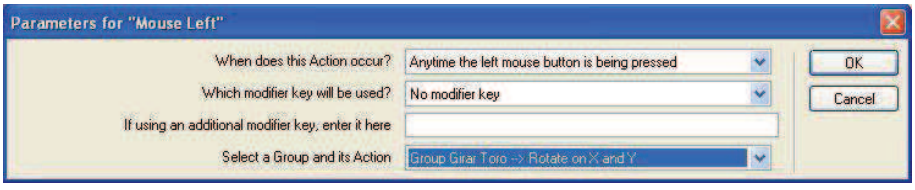


Imagen 5.9.cp.1.5: Caja de diálogo para "Botón izquierdo del ratón"

3.7. A la primera cuestión respondemos con la elección que se ve en la imagen: "Anytime the left... ". Esto significa que mientras tenga pulsado el ratón con el botón izquierdo, girará la escultura si muevo el ratón.

3.8. Por último, acceder a la última pregunta y elegir la opción "Rotate on X and Y".

3.9. Observar que los comportamientos actualizan la ventana Cast visualizándose como elementos utilizados, tal como se aprecia en la imagen que sigue.

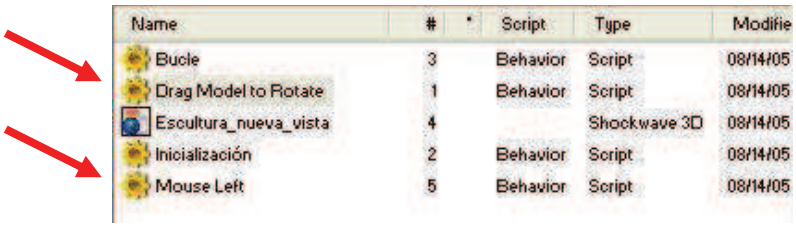


Imagen 5.9.cp.1.6: Actualización de la ventana Cast

3.10. Ejecutar la película, pulsando el botón “Play” para comprobar los cambios.

4. Comportamientos para la interacción con una cámara.

Vamos a realizar una interacción con la cámara. En realidad la cámara es como nuestros propios ojos. Lo que se pretende es que podamos cambiar el punto de vista con que se ve la escultura, sin moverla. En este sentido se parece más a lo que hacemos en un museo, es decir, cambiar nosotros de posición en vez del objeto visualizado.

4.1. Accedemos a las acciones, y seleccionar “Drag 

Camera”. Luego arrastrarla hasta el sprite del mundo3D.

4.2. Aparecerá la caja de diálogo que se puede observar en la imagen siguiente.

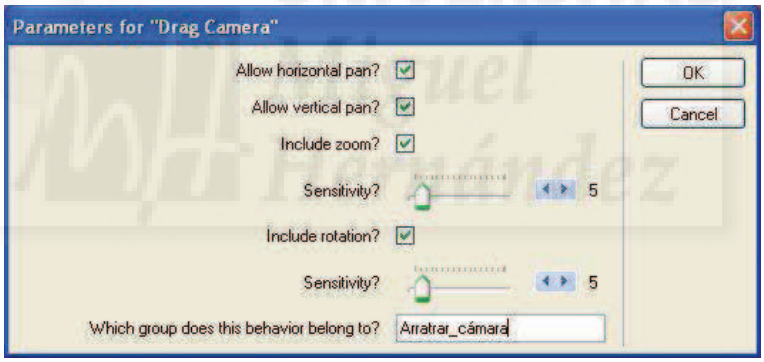
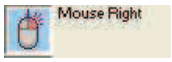


Imagen 5.9.cp.1.7: Caja de diálogo para “Arrastrar cámara”

4.3. No vamos a modificar ninguno de sus parámetros, pero si vamos a ponerle un nombre. Hemos escrito “Arrastrar_cámara” como se puede apreciar en la imagen anterior.

4.4. Accedemos a los disparadores, elegimos “Mouse 

Right” y arrastrarlo al sprite del mundo3D. Aparecerá la imagen que sigue.

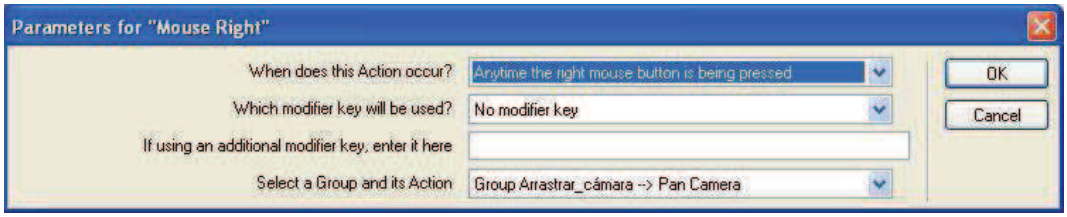


Imagen 5.9.cp.1.8: Caja de diálogo para “Botón derecho del ratón”

4.5. Aquí tampoco vamos a modificar ningún valor. Solo “conectaremos” este disparador con la acción recién creada como indica la imagen 5.9.cp.1.8.

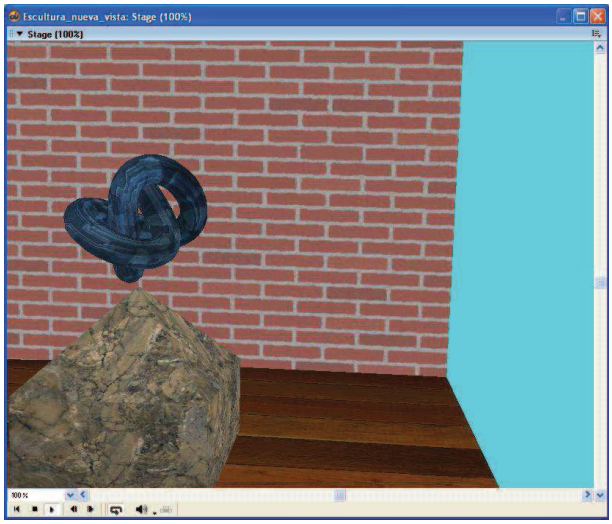


Imagen 5.9.cp.1.9: Pantalla del movimiento de la cámara

4.6. Ejecutar la película, pulsando el botón Play para comprobar los cambios. Pulsar el botón derecho para mover la cámara como se puede ver en la imagen de abajo y el botón izquierdo sobre la escultura para moverla. Esto nos va dando más sensación de libertad para observar nuestro trabajo.

Aún no hemos acabado con la cámara, ya que cuando la movemos muchas veces perdemos las referencias. Esto es muy normal, ya que, de momento, no es una cámara libre y no podemos deshacer el recorrido realizado. Además es muy difícil moverse por un espacio tridimensional a base de periféricos para 2D.

Todo esto hace que sea conveniente el utilizar otro comportamiento para resetear la cámara, es decir, dejarla en la posición inicial. En este caso, vamos a utilizar como disparador el teclado. Se puede utilizar cualquier tecla. Las más comunes para el movimiento son las teclas de flechas de dirección. Nosotros para reiniciar la cámara utilizaremos la barra espaciadora.

4.7. Accedemos a las acciones y seleccionar “Reset 

Camera”. Luego arrastrarla hasta el sprite del mundo3D.

4.8. Aparecerá la caja de diálogo de la imagen siguiente.

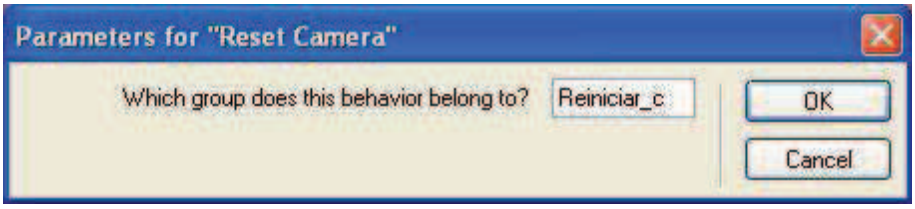
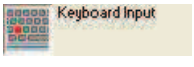


Imagen 5.9.cp.1.10: Caja de diálogo de parámetros para “Reiniciar Camera”

4.9. Como no requiere ningún parámetro, basta con ponerle un nombre.

4.10. Acceder a los disparadores, seleccionar “Keyboard 

Input” y luego arrastrarlo hasta el sprite del mundo3D.

4.11. Aparecerá la siguiente caja de diálogo:

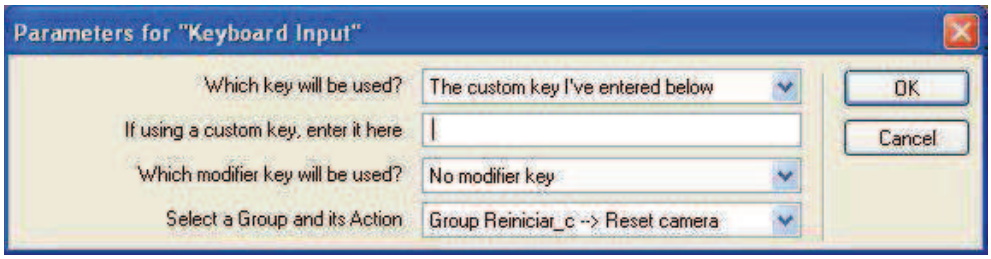


Imagen 5.9.cp.1.11: Caja de diálogo para parámetros de "Entrada de teclado"

4.12. La primera opción sirve para utilizar las teclas de dirección como disparador. Nosotros queremos que sea la barra espaciadora, para ello, solo modificamos la segunda opción simplemente pulsando esta tecla.

Como se puede observar en la imagen 5.9.cp.1.11, el cursor se encuentra por ello en el segundo carácter. El primero es el espacio, consecuencia de haber pulsado la tecla barra espaciadora.

4.13. Ejecutar la película pulsando el botón "Play" para comprobar los cambios. Observar como podemos mover la cámara y cuando pulsamos la barra espaciadora vuelve a su situación original.

5. Comportamiento genérico.

Este comportamiento lo hemos querido explicar a parte porque es especial. En primer lugar es una acción que no sirve para algo en concreto como las demás, si no que nos presenta un espacio editable para escribir una sentencia Lingo que se ejecutará como todas, es decir, cuando se dispare el trigger al que está ligada.

Para demostrar su gran utilidad vamos a realizar una tarea que consiste en acompañar de sonido el movimiento de la escultura. Es obvio que necesitamos un sonido que importaremos de la misma forma que la propia escena 3D, es decir, con el comando Import. El tipo de fichero puede ser un .wav y lo vamos a nombrar en el Cast como "sonido".

Recordemos: la escultura se mueve mientras se mantenga pulsado el botón izquierdo del ratón, pero cuando utilizamos ese trigger vimos que también están las posibilidades de dispararlo cuando se pulsa y cuando se suelta. Por lo tanto vamos a aprovechar estas posibilidades para ejecutar un sonido más o menos largo y que denote movimiento cuando se baje el botón izquierdo, luego, la escultura se moverá mientras lo tenga pulsado y por último ejecutaremos una sentencia que pare cualquier sonido cuando se suelte el botón izquierdo. El efecto será muy intuitivo y más realista.

Por lo tanto, utilizaremos dos comportamientos genéricos y dos triggers mouse left.

5.1. Accedemos a las acciones, seleccionamos "Generic Do" y la arrastramos hasta el sprite del mundo3D.

5.2. Aparecerá la caja de diálogo que se muestra en la imagen que sigue.

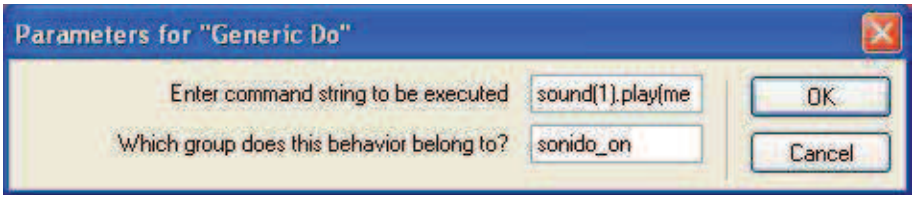


Imagen 5.9.cp.1.12: Caja de diálogo de parámetros para "Hacer genérico"

5.3. En el primer parámetro escribir: sound(1).play(member("sonido")). Esta sentencia es de Lingo y sirve para que se ejecute el sonido del Cast llamado “sonido”. Esta misma sentencia la utilizaremos en otros casos prácticos insertada como parte del código que programaremos.

5.4. Repetir todos los pasos para utilizar un trigger “Mouse Left” pero para la opción primera que dice “When does this Action occur?” (¿Cuándo ocurrirá esta acción?) Elegiremos la opción “Only when the left mouse button is first pressed” (solo cuando el botón izquierdo sea pulsado), con lo cuál no interferirá en el movimiento de la escultura ya realizado.

5.5. Acceder nuevamente a las acciones y volver a utilizar una acción “Generic Do”.

5.6. En el primer parámetro esta vez escribir: puppetsound(1,0). Esta sentencia de Lingo suspende el sonido que antes habíamos ejecutado.

5.7. Repetir nuevamente todos los pasos para utilizar otro trigger Mouse Left pero en la primera opción elegir la que dice: “Only when the left mouse button is released” (solo cuando el botón izquierdo se deje de pulsar). Observar que la ventana Cast al final tiene que tener un aspecto como muestra la imagen 5.9.cp.1.13.

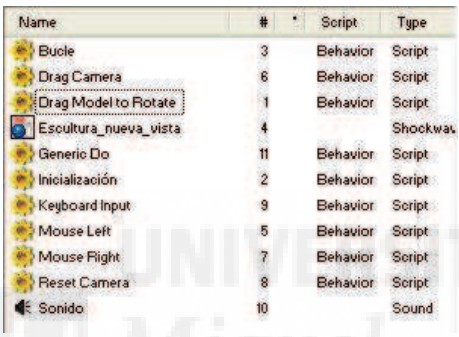


Imagen 5.9.cp.1.13: Ventana Cast

5.8. Ejecutar la película pulsando el botón “Play” para comprobar los cambios. Observar como sin modificar la funcionalidad de mover la escultura, la hemos dotado de un nuevo efecto.

5.9. Publicar la película para comprobar que se puede ejecutar fuera de Director.

Conclusiones:

En este caso práctico no hacemos sino utilizar un código ya creado por otros autores y que está pensado de forma genérica para aumentar la productividad. Aunque parezca obvio, la reutilización de código y su colección en bibliotecas es uno de los hitos de la programación moderna. El modelo actual de programación, el llamado orientado a objetos, tiene en esta característica, una de sus principales ventajas. En nuestro caso particular, esto es aún más útil, ya que podemos obviar gran parte de la complejidad de “cómo” hacer que funcionen las cosas, es decir, de cómo programarlas y centrarnos más en el “qué” queremos que hagan.

Caso práctico 9.2: Interacción avanzada de objetos.

Objetivo: Intentar desarrollar nuestras habilidades con Lingo. Lo primero que haremos es utilizar las variables globales para realizar controles on/off y aplicarlos para las transformaciones de los modelos. Luego estudiaremos un método para seleccionar un objeto del mundo 3D y poder interaccionar directamente con él, esto además con la dificultad añadida de utilizar un dispositivo 2D como es el ratón.

Tiempo de realización: 1´5 horas.

Pasos a realizar:

1. Controles on/off.
2. Seleccionar objetos 3D con el ratón 2D.
3. Control de objetos mediante el teclado.

1. Controles on/off.

Estos controles funcionan como un conmutador. Sólo tienen dos estados: encendido/on y apagado/off. Para pasar de un estado a otro requieren alguna interacción del usuario. Una aplicación de esta clase de botones podría ser una interface donde los objetos 3D harían las veces de botones.

Vamos a realizar una pequeña práctica para demostrar cómo pueden funcionar. Este trabajo práctico consiste en que cuando se pulse el botón izquierdo del ratón la escultura debe rotar sobre sí misma.

Para ello debemos utilizar una variable que indique el estado del control, la hemos llamado “rotar”. Esta variable solo tiene sentido en este script, a diferencia de “mundo3D” que hemos utilizado en varios scripts. Por ello en vez de ser global, la declararemos local con la palabra reservada property.

1.1. Vamos a realizar una copia del anterior fichero “escultura.dir” y lo llamaremos “escultura_controles” para no tener que empezar desde el principio.

1.2. Abrir la ventana Score (ctrl.+4) y abrir el script “Programación” y escribir debajo de global Mundo3D otra variable, aunque esta será local:

```
property rotar.
```

1.3. Vamos a crear dos disparadores para que cambien el estado de la variable. Escribir a reglón seguido:

```
on mouseDown me
        rotar = 1
end
```

Esto hará que cuando se pulse el botón izquierdo del ratón el control se ponga encendido, es decir, debe comenzar a rotar.

1.4. Seguidamente escribir:

```
on mouseUp me
        rotar = 0
end
```

Esto hará que cuando se deje de pulsar el botón izquierdo del ratón el control tome el valor “apagado”, es decir, debe dejar de rotar.

Para que la escultura gire mientras tanto, utilizaremos el disparador “exitframe” que hará que la película esté constantemente entrando y saliendo del fotograma llamado “Principal” (esto lo controlamos con el bucle).

1.5. Dentro de este disparador debemos escribir la sentencia que haga rotar la escultura. Pero con la salvedad de que esta rotación solo debe producirse si el estado del control es on, es decir, si la variable “rotar” vale 1. Por tanto escribimos

```
on exitframe
        if rotar = 1 then mundo3D.model("Torus knot01").rotate(10,10,10)
end
```

1.6. Ejecutar la película pulsando el botón “Play” para comprobar que efectivamente la película hace lo que pretendemos.

El script completo será como sigue:

```
global mundo3D

property rotar

on mouseDown me
    rotar = 1
end

on mouseUp me
    rotar = 0
end

on exitframe
    if rotar = 1 then mundo3D.model("Torus knot01").rotate(10,10,10)
end
```

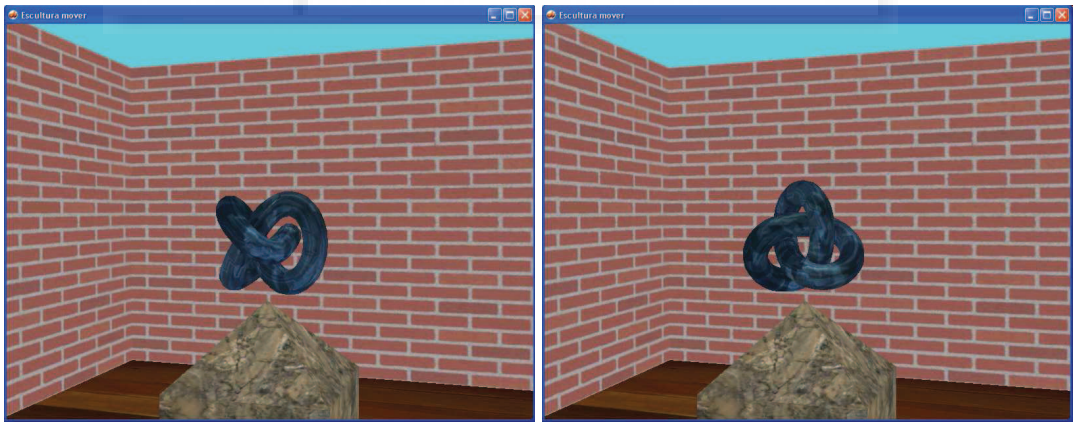


Imagen 5.9.cp.2.1: Rotación interactiva del objeto 3D “Torus Knot01”

2. Seleccionar objetos 3D con el ratón 2D.

Nuestro mundo 3D no funciona de la forma mas correcta porque aún existe el problema de que la escultura gira cuando hacemos clic con el botón izquierdo del ratón en cualquier lugar de la escena 3D y esto no es lo que queremos. Ya que para darle mayor realismo debe rotar solamente cuando se selecciona directamente el objeto 3D con el ratón. Así, si en una escena tenemos varias esculturas podremos moverlas independientemente.

2.1. Creamos una variable local para guardar el nombre del modelo seleccionado llamado "tocado" y utilizamos el procedimiento detalladamente explicado en la unidad teórica "modelUnderLoc" de la cámara por defecto, pero como este procedimiento se refiere al sprite actual, antes debemos poder nombrarlo, por ello escribiremos:

```
property psprite

on beginSprite me
        pSprite = sprite(me.spriteNum)
end
```

2.2. Ahora si podemos usar "modelUnderLoc" como sigue:

```
tocado = psprite.camera.modelUnderLoc(the clickLoc - point(psprite.left,psprite.top))
```

2.3. Ahora utilizaremos el nombre del modelo que queremos rotar para compararlo (usando la función contains) con el nombre devuelto por modelUnderLoc, y si es así, cambiaremos el valor de la variable rotar. Escribimos a continuación:

```
if tocado.name contains "Torus"  then  rotar =1
```

2.4. Ejecutar la película pulsando el botón Play para comprobar el resultado. Debe funcionar perfectamente. Si hago clic en un sitio que no sea el "Torus Knot", es decir, nuestra escultura, esta no rotará, pero si acertamos a "clickearle", rotará.

El script completo será el que sigue:

```
global mundo3D
property rotar
property psprite
property tocado

on beginSprite me
  pSprite = sprite(me.spriteNum)
end

on mouseDown me
  tocado = psprite.camera.modelUnderLoc(the clickLoc - point(psprite.left,psprite.top))
  if tocado.name contains "Torus"  then  rotar =1
end

on mouseUp me
  rotar = 0
end

on exitframe
  if rotar = 1 then mundo3D.model("Torus knot01").rotate(10,10,10)
end
```

3. Control de objetos mediante el teclado.

3.1. Cómo escribimos en la unidad teórica correspondiente, podemos utilizar la propiedad de teclado "the keypressed" que devuelve el carácter pulsado. Por ejemplo, para escalar el modelo 1 de la escena 3D si pulsamos la tecla "s" escribimos:

```
if (the keypressed="s") then mundo3D.model[1].Scale(.5,.5,.5)
```

3.2. Ahora vamos a “complicar” un poco el código para utilizar la sentencia case que nos evita utilizar una lista muy grande de if´s y queda más “elegante” (como se dice en argot de programadores). La sentencia case tiene la siguiente sintaxis:

```
case valor_lógico of
        valor_numérico_1 : sentencia
        valor_numérico_1 : sentencia
        ...
end case
```

Existe un problema que reside en que the keypressed devuelve una tecla y case necesita un número. Para solventar el problema utilizaremos la función “chartonum” que significa “carácter_a_número” y devuelve el código Ascii (o sea, un número) de la tecla. Por lo tanto podremos escribir algo así:

```
case (chartonum(the keypressed)) of
        30: sentencia    -- 30 es el valor ascii de la tecla “flecha arriba”
```

3.3. Ahora tenemos que elegir las teclas para mover nuestra escultura y saber qué código Ascii le corresponde. Hemos escogido las teclas flecha arriba y abajo para moverla por el eje Y, las teclas izquierda y derecha para moverla por el eje Z (el de profundidad) y la tecla ‘a’ y ‘A’ para moverla por el eje X.

3.4. Hay que tener cuidado con las coordenadas universales y locales. Como no determinamos las coordenadas de la escultura (aunque se puede) en MAX, es mejor utilizar coordenadas universales y probar, ya que no es fácil moverse en 3D. Por lo tanto las líneas bajo el case se parecerán a esta: 30: mundo3D.model[1].translate(0,0,10,#world). Por lo tanto todo el comportamiento debería contener el siguiente texto:

```
global mundo3D
property rotar
property psprite
property tocado

on beginSprite me
        pSprite = sprite(me.spriteNum)
end

on mouseDown me
        tocado = psprite.camera.modelUnderLoc(the clickLoc - point(psprite.left,psprite.top))
        if tocado.name contains "Torus"  then  rotar =1
end

on mouseUp me
        rotar = 0
end

on exitframe
        if rotar = 1 then mundo3D.model("Torus knot01").rotate(10,10,10)
        if the keypressed="s" then mundo3D.model[1].Scale(.5,.5,.5)
        case(chartonum(the keypressed)) of
                30: mundo3D.model[1].translate(0,0,10,#world) -- tecla flecha arriba
                31: mundo3D.model[1].translate(0,0,-10,#world)-- tecla flecha abajo
                28: mundo3D.model[1].translate(-10,0,0,#world)-- tecla flecha izquierda
                29: mundo3D.model[1].translate(10,0,0,#world) -- tecla flecha derecha
                97: mundo3D.model[1].translate(0,10,0,#world) -- tecla A
                98: mundo3D.model[1].translate(0,-10,0,#world)-- tecla B
        end case

end
```

Hay que tener cuidado al manipular el movimiento de la “escultura” ya puede ser que la hagamos desaparecer a través de la pared como demuestra la imagen que sigue.

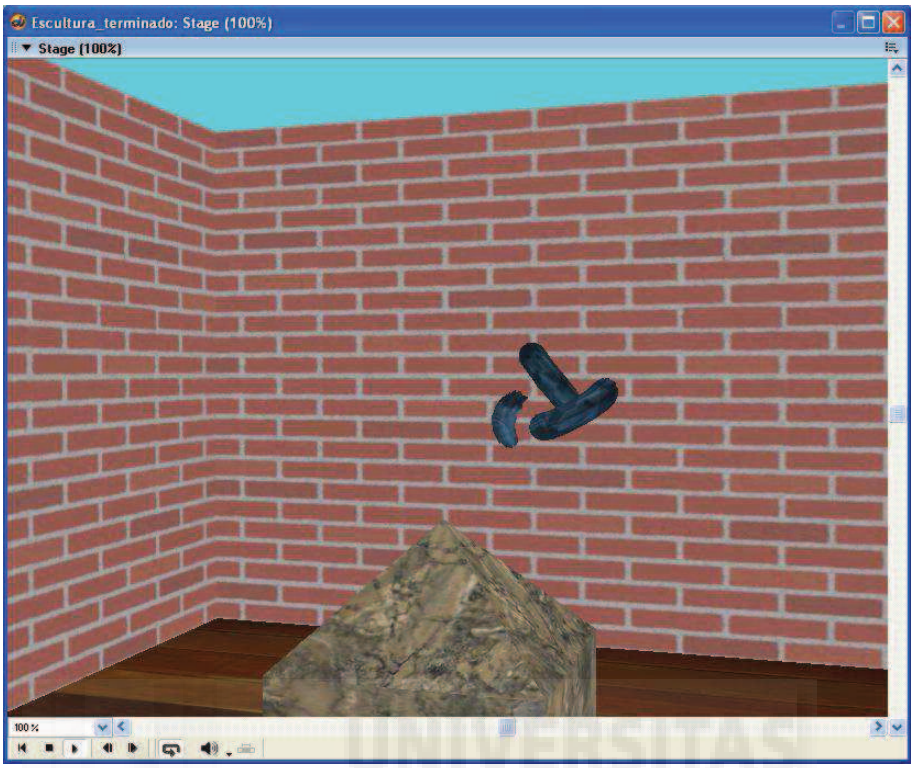


Imagen 5.9.cp.2.2: Captura de pantalla mientras se interacciona con el objeto 3D

Conclusiones:

Para ir adentrándonos en el control de los objetos que requieren nuestras obras, primero nos centramos en cómo seleccionar un objeto determinado dentro de la escena 3D. Más tarde nos empleamos en encontrar modos de programar en Lingo para manipularlos con el ratón o con las teclas de forma que podamos actuar sobre ellos.

Unidad 10: La iluminación en Director.

Introducción teórica:

1. Introducción a la iluminación en Director.
2. Tipos de luces en Director. Faltan imágenes.
3. Utilizar las luces con Lingo.
4. Problemas de la iluminación en Director.

Casos prácticos:

10.1. Iluminación en Director.

1. Introducción a la iluminación en Director.

La iluminación es uno de los temas más delicados en Director y en casi todas las aplicaciones de tipo tridimensional. Nosotros estamos trabajando en un entorno interactivo 3D en tiempo real y esto conlleva algunas limitaciones. Es en el campo de la iluminación dónde son más evidentes estas, sobre todo si lo comparamos con los resultados que podemos obtener en programas como 3d Studio MAX.

En Director, la optimización de la representación es prioritaria. Esto es debido a que para generar la iluminación en una escena tridimensional, se necesita mucha potencia de cálculo. Incluso las más modernas tarjetas gráficas, diseñadas específicamente para estos trabajos, tienen dificultades para generar escenas perfectamente iluminadas y en tiempo real.

Esta distinción es muy importante, ya que la diferencia radica en este punto: si la escena se genera en tiempo real o no. Si no se trata de una presentación en tiempo real, podemos hacer como los programas del tipo de 3D Studio MAX, la representación de la escena, también llamada renderización. En este caso, un módulo especial se encarga de los complejos cálculos para generar las iluminaciones y sus implicaciones en los materiales de los modelos como son el cálculo de refracciones, reflejos, etc. Esto puede llevar incluso horas para un solo fotograma, por lo que es imposible hoy en día tener la misma calidad para una creación en tiempo real.

Este problema de la iluminación es una cuestión que se conoce cómo implementarlo. Es solo una cuestión de velocidad de proceso, por lo que se supone que si sigue la tendencia del progreso en cálculo del hardware y en las mejoras de los algoritmos de cálculo, pronto veremos este hecho posible computacionalmente en ordenadores domésticos.

Sin embargo, hoy en día, Director no permite algunas de las más destacadas características que podemos realizar con la iluminación en MAX, como pueden ser la proyección de sombras. Director no admite que los objetos proyecten sombras. Si en un proyecto esta característica fuera muy importante, tendríamos que hacer nosotros una textura que simulase la sombra y texturizar con ella el elemento sombreado.

Por otra parte, la iluminación en Director dispone de herramientas suficientes para la mayoría de los proyectos en tiempo real y permite además, la creación de composiciones de luces de alta calidad.

Director ilumina la escena con distintos tipos de luces y cada uno de ellos se puede configurar para disponer de un amplio rango de variación de luminosidad. En general, supondremos que en la escena tendremos varias luces de distinto tipo que emitirán luz con distinto color y desde posiciones y ángulos distintos.

Además, estas luces se pueden controlar por programación. Esto es importante porque podemos disponer la luces e interactuar con el usuario. Hay luces que se utilizan para iluminar la escena en general pero hay otras que se utilizan para realzar un determinado objeto o una determinada parte de él que es especialmente importante. Además se pueden establecer distintas atmósferas o estudiar cómo quedaría una misma escultura a distintas horas del día.

Normalmente, la iluminación es un tema muy delicado y que lleva mucho tiempo comprender (y es todavía más crítico si creamos las luces con Lingo) por lo que hay toda una serie de estudios sobre cómo distribuirlas en la escena. No es objetivo de este trabajo adentrarnos muy profundamente en este engorroso tema.

Por todo ello se deben buscar métodos (muchos artistas tienen métodos muy personales para resolver la iluminación y forma parte de su propio arte) para priorizar, buscando las luces claves y las que serán complementarias.

2. Tipos de luces en Director.

Existen 4 tipos de luces en Shockwave 3D: ambient, directional, point y spot.

◆ Ambient: esta luz existe siempre, es la que aporta la luz ambiente a la escena. Esta es una característica muy importante, y más desde el punto de vista artístico, pero no aporta "profundidad" a los objetos, por lo que no debe de existir sola en la escena, ya que todos los objetos se verían muy planos.

En las siguientes imágenes se puede apreciar el problema citado cuando se configura la luz ambiente a color verde puro RGB (0, 255, 0) y se borran todas las demás luces. La luz ambient solo tiene un parámetro: el color, que se codificará en formato RGB.

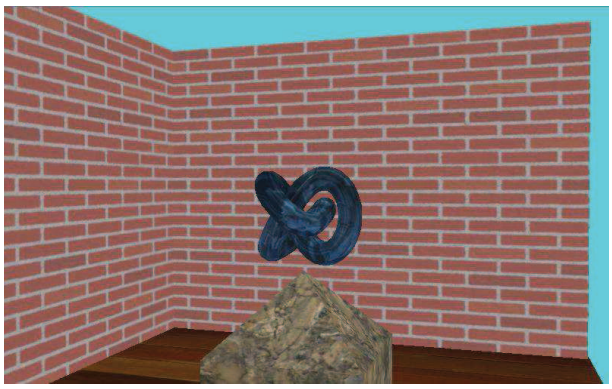


Imagen 5.10.1: Escena con luz normal

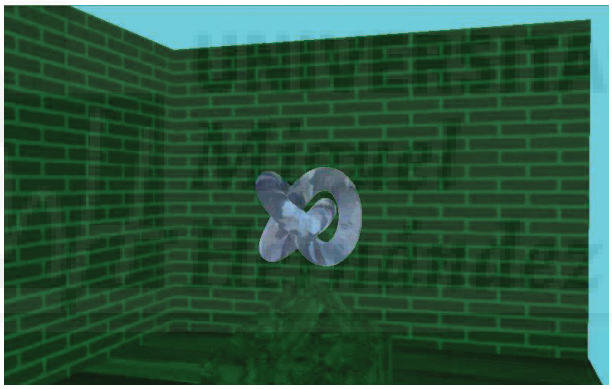


Imagen 5.10.2: Escena con luz tipo Ambient

◆ Directional: este tipo de luz se utiliza tanto para la iluminación general de la escena como para iluminar determinados modelos. El tipo de luz que aporta es muy parecida a la luz solar. Esto es debido a que los rayos de luz del sol aunque parten en todas las direcciones, al estar tan lejos, llegan hasta nosotros prácticamente paralelos. Los rayos de la luz direccional son totalmente paralelos y por eso sirven para recrear la luz solar.

Esta luz ilumina a toda la escena, todos los objetos que estén en nuestro mundo 3D se verán afectados por ella, independientemente de su situación. La luz direccional no se localiza en un punto determinado de la escena, sino en el infinito. Los parámetros que tiene este tipo de luz son el color, la propiedad especular (con valores verdadero o falso) y la rotación, que determinará la dirección de esta luz.

◆ Point: estas luces emiten rayos de luz en todas direcciones, como si fueran una bombilla. Se utilizan mucho para iluminaciones generales de una escena y también para modelar detalles de algunos modelos. Su principal característica es que se pueden crear en un punto en concreto, por lo que se suelen utilizar para iluminación de interiores.

Las características que tienen son el color, el brillo, la atenuación y la posición.

◆ Spot: este tipo de luz es el más complejo de trabajar en Shockwave 3D. Se deben utilizar con mucha precisión y cuidado, ya que son las luces que requieren más cálculo computacional y todo el rendimiento del mundo 3D se vería afectado.

Se puede comparar con la luz de una linterna o con un foco. Se usa para "modelar", es decir, resaltar zonas específicas de un modelo en especial. Para usarlas correctamente se tiene que tener un control muy grande de la escena y las relaciones geométricas de los objetos que están en ella.

Los parámetros que rigen esta luz son el color, rotación, posición, especular, ángulo del foco y la caída del foco. El ángulo del foco se mide en grados y determina la anchura del haz de luz. La caída del foco es una propiedad booleana (solo valores verdadero o falso) y determina la homogeneidad del borde del haz de luz sobre los objetos.

3. Utilizar las luces con Lingo.

En Shockwave 3D las luces es un recurso muy limitado. Según se van creando luces, se van disponiendo en una lista de luces, de tal forma, que la primera luz será la 1, la siguiente la 2 y así sucesivamente.

Para crear una luz utilizaremos la siguiente sentencia que presenta la sintaxis: member("nombre_castmember").newlight("nombre_luz",#tipo). Por ejemplo, escribir:

```
mundo3D.newlight("solar",#directional)
```

Esta luz podrá ser referenciada como "1", y por lo tanto mundo3D.light[1].color = rgb (255,0,0) pondrá la luz "solar" totalmente roja. Mundo3D.light.count es una variable que nos devolverá cuantas luces existen en escena.

Para modificar las luces tendremos que nombrarlas y modificar la propiedad que deseemos directamente, por ejemplo:

```
mundo3D.newlight("bombilla",#point)
mundo3D.light("bombilla").translate(10,0,0)
mundo3D.light("bombilla").color = rgb(0,0,255)
mundo3D.light("bombilla").specular = true
```

Esto creará una luz llamada bombilla, que situaremos en 10, 0, 0, muy cerca del origen del mundo y emitirá una luz verde con la propiedad especular.

Para borrar una luz, se utiliza la función "deletelight", por ejemplo, escribiremos:

```
mundo3D.deletelight(4)
```

4. Problemas de la iluminación en Director.

Cómo hemos escrito anteriormente, Director no puede realizar la generación de iluminación realista en tiempo real al mismo nivel de realismo que MAX.

Por ejemplo, si importamos luces de tipo foco (sean estas de tipo libres o no libre) tendremos problemas.

Tampoco la proyección de sombras se puede realizar en Director como pone de manifiesto la comparación de las imágenes de abajo.

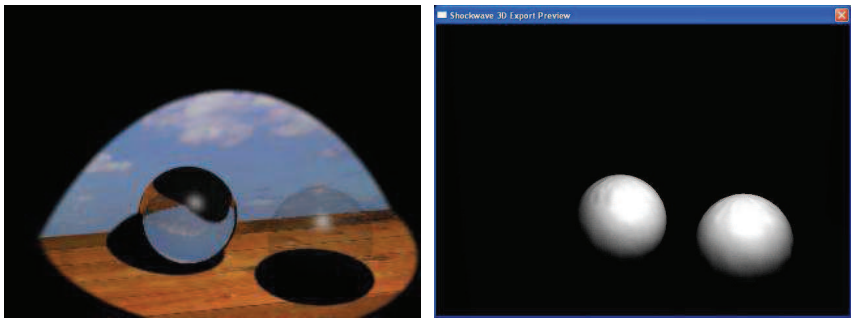


Imagen 5.10.3: Escena iluminada en MAX y exportada a Director

Comparando la imagen anterior se puede apreciar que no merece la pena utilizar este tipo de iluminación ya que Director no tiene en cuenta ningún efecto de sombreado ni directo (sobre los cuerpos) ni indirectos (de unos cuerpos sobre otros).

Problemas con las luces tintadas: en la siguiente imagen se puede observar que las luces tintadas o con atenuación aunque no son ignoradas por Director, no tienen el efecto que deberían. Se puede apreciar cómo los objetos que reciben la luz se tintan del color de esta, pero al no tenerse en cuenta las sombras, todo el efecto se viene abajo y deja de ser útil.

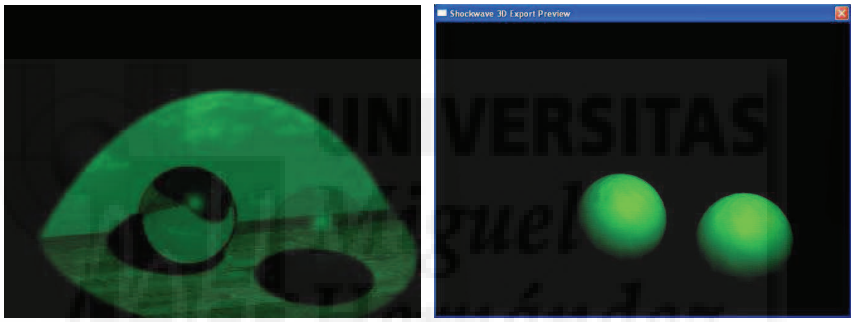


Imagen 5.10.4: Escena con luz tintada en MAX y exportada a Director

Resumiendo, con la iluminación casi siempre tendremos problemas al exportar nuestra escena a Director. Una solución puede ser el crear nosotros las luces especiales directamente con Ligo. Desde luego que es una tarea más difícil que importarlas desde MAX pero los resultados serán mejores.

Otro problema es que la cantidad de luces en Director está limitada a 7 y se recomienda crearlas directamente en este programa, ya que tiene una importancia muy grande en la visión de la escena.

Por todos estos problemas tenemos que tener presente antes de exportar a Director que podemos previsualizar cómo quedará en formato w3d y hacer los cambios pertinentes.

Caso práctico 10.1: Iluminación en Director.

Objetivo: Crear una escena donde podemos controlar interactivamente dos luces: una de tipo bombilla y otra de tipo foco. Pretendemos controlar los parámetros más importantes de ambas luces como son su posición, su color, su intensidad y si están encendidas o apagadas.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. La creación de la escena y las luces en MAX.
2. Referencia a las luces.
3. Apagar y encender luces.
4. Mover la luz bombilla.
5. Girar la luz foco.
6. Colorear las luces.
7. Cambiar la intensidad de las luces.

1. La creación de la escena y las luces en MAX.

Hemos creado las luces en MAX puesto que es más fácil crearlas y gestionarlas en este programa. Esto es así por el hecho de que las creamos y posicionamos de forma interactiva y en Director este trabajo lo tenemos que hacer mediante programación en Lingo, lo cual hace que tengamos que trabajar "a ciegas" y sea todo más complicado. Además, al ser un sistema que no trabaja en tiempo real, podemos crear imágenes y vistas del resultado de las luces sobre la escena para su modificación si hiciera falta.

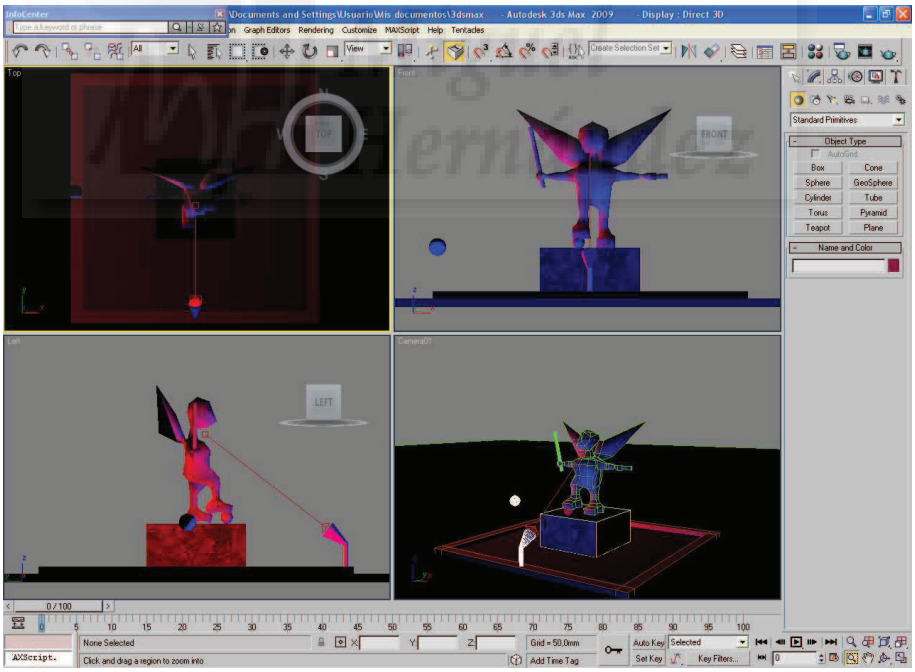


Imagen 5.10.cp.1.1: Escena en 3D Studio MAX con la iluminación realizada

La escena en sí, es una "escultura" muy tosca colocada en un pedestal. No hemos trabajado mucho en los detalles ya que lo que interesa es el estudio de las luces y su impacto sobre la propia escultura y sobre su entorno.

En esta escena incluimos el uso de cámaras. Este recurso lo explicaremos detalladamente en la unidad siguiente. Las hemos porque es un buen sistema para explorar el mundo 3D, y así visualizar la iluminación desde distintos puntos de vista.

Hemos creado 3 luces, una ambiental que no modificaremos en Director y dos que si cambiaremos interactivamente sus parámetros y que son de distinto tipo. Una es de tipo “Omni” y la otra de tipo “Target Direct”.

Las luces de tipo “Omni”, según las llama MAX, son luces que emiten luz en todas las direcciones, tal como hace una bombilla. Nosotros hemos creado una a la izquierda del pedestal, de color rojo y con una intensidad el doble de lo normal.

Las luces “Target Direct” emiten luz en una dirección y tienen un puntero para dirigir la luz, es decir, hacia dónde apuntan. Es una luz que imita la que produce un foco de los que hay, por ejemplo, en un escenario. Nosotros hemos creado una luz focal dirigida situada a los pies de la estatua justo en el reborde que la rodea y apunta directamente hacia la barbilla del personaje.

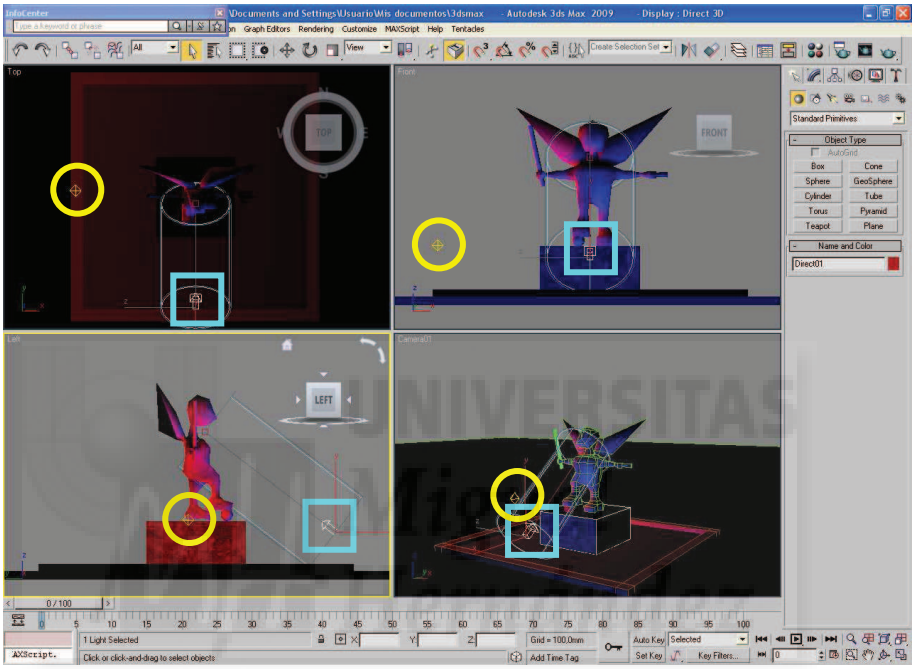


Imagen 5.10.cp.1.2: La luz Target Direct en azul y la Omni en amarillo

En la imagen superior se pueden distinguir las dos luces en la escena de MAX, la luz omni aparece como un romboide y está rodeada de un círculo amarillo. La luz focal aparece como un cilindro para representar el haz de luces y tiene dos controles, la fuente de luz, que hemos rodeado de un cuadrado azul y el objetivo que se puede distinguir como un pequeño cuadradito rojo bajo la cabeza de la estatua.

Si comparamos esta imagen con la primera de este caso práctico, veremos que existen diferencias. En la primera, donde se sitúa la luz omni, se ha creado una esfera llamada “Sphere02” y donde se sitúa la luz focal hemos creado un cono llamado “Cone01” que se apoya sobre un pequeño cilindro a modo de pie del foco. Estos modelos los hemos creado para representar las fuentes de luz en Director, ya que en este programa las fuentes de luz no se representan, solo se representa su efecto, o sea, la iluminación que consiguen, por lo que las luces en sí son invisibles. Esto lo hemos hecho para que cuando en Director cambiemos la posición de las luces, cambiaremos también la posición de los modelos asociados y así tendremos una idea más clara de dónde se encuentran.

Las luces las creamos en el menú Create > Lights > Standard. La explicación de cómo trabajar con luces en MAX la debemos buscar en la unidad correspondiente. Pero a modo de recordatorio los pasos serán los siguientes:

1.1. En el panel crear > luces, elegimos la opción estándar. Luego pulsamos el botón “omni” y hacemos clic sobre la vista que más nos convenga para situarla, por ejemplo, en la vista Top.

Con la luz seleccionada tomamos la herramienta mover y en la vista frontal la trasladamos hasta la posición deseada.

1.2. Con la luz seleccionada, accedemos al panel modificar y al rollout Intensity /Color /Attenuation y ponemos el valor Multiplier a 2 y el color en rojo puro (255, 0,0).

1.3. Repetir la operación para crear una luz focal, sólo cambia que al crear la luz, necesitamos definir dos puntos, el origen de la luz y el punto objetivo. La dirección la dará la recta entre los dos puntos. Si hace falta, se pueden seleccionar estos dos puntos por separado y posicionarlos con la herramienta mover.

1.4. Crear una tercera luz de tipo SkyLight situada en la parte derecha y arriba de la escultura para tener una luz ambiental.

Las 3 luces creadas tendrán estos parámetros:

Nombre	On	Multiplier	Color
Omni01	Si	2,0	255,0,0
Direct01	Si	2.0	0,0,255
Sky01	Si	1,0	255,255,255

Tabla 4: Luces de la escena y sus valores

Por último, con la escena (y sus luces) creada, procedemos a su exportación a W3D por el método estándar y con la única particularidad de que tenemos que tener el recurso Lights seleccionado para la exportación. No hemos tenido nunca ningún problema en este proceso, por lo que lo podemos calificar de seguro.

2. Referencia a las luces.

En Director, podemos reutilizar las películas anteriores para ahorrarnos trabajo. Si partimos de la película base y habiendo efectuado la importación del fichero W3D, solamente tendremos que cambiar la referencia a “mundo3D” en el script “Inicialización”.

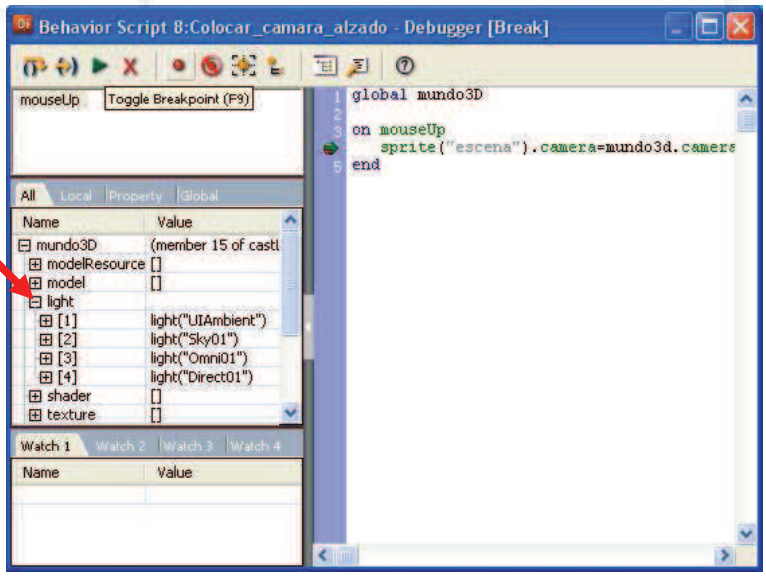


Imagen 5.10.cp.1.3: Acceso a los nodos de la escena 3D mediante un breakpoint

Para tener acceso a los nodos (componentes) de la escena 3D podemos usar varios métodos. Uno es crear un punto de ruptura de ejecución o “Toggle breakpoint” en cualquier línea de un script. Luego al pulsar el botón de “Play” para que corra el programa, hacemos que el script en cuestión se ejecute y cuando llega a la línea con el breakpoint se para.

Entonces observaremos como en la parte izquierda aparece una ficha llamada “All” y la entidad “mundo3D” del que cuelgan todos sus nodos. Accederemos al nodo llamado “Lights” y podremos ver todas las luces de la escena y los valores de sus parámetros, tal como se observa en la imagen que sigue.

Podemos destacar que aunque la escena en MAX contenía 3 luces, en Director tenemos 4, ya que aquí siempre tenemos una luz ambiental llamada “UIAmbient”.

En Director las luces se guardan en un array (vector) de luces, donde podemos referirnos a ellas mediante el lugar que ocupan en este vector. En la imagen superior se puede observar como la luz que ocupa el primer lugar es la ambiental, en segunda posición tenemos la luz importada “Sky01”, la tercera es “Omni01” y la cuarta y última es “Direct01”, también importadas.

Esto es muy cómodo, ya que cuando queramos referirnos a las luces que vamos a editar en Director nos referiremos a ellas como sigue:

```
mundo3D.Light[3]      -- esta será la forma más corta de acceder a la luz Omni01
mundo3D.Light[4]      -- esta será la forma más corta de acceder a la luz Direct01
```

Estas referencias las podremos utilizar a partir de ahora para cambiar los parámetros de ambas luces mediante Lingo. Vamos a crear scripts para modificar las luces. Estos códigos los asignamos a botones para crear funcionalidades que enriquezcan el caso práctico.

3. Apagar y encender luces.

El título de este punto indica claramente lo que queremos poner en práctica y es cómo realizar los scripts que se asociarán a dos botones por luz: uno para apagarla y otro para encenderla. Debido a que cuando se importan las luces en el fichero W3D, estas se muestran en el mismo estado que en MAX, esto es, encendidas, atacaremos primero cómo apagarlas. Para ello, debemos pensar que las luces no tienen un interruptor de on/off, simplemente poniendo una luz en negro, no se observará luz ninguna.

Para realizar este procedimiento, debemos saber que todas las luces en Director poseen varias características comunes, una de ellas es el color, por lo que para poner apagar una luz, utilizaremos un código como este:

```
mundo3D.Light[3].color = rgb(0,0,0)
```

Es muy claro de interpretar: la luz numero 3, es decir, la luz de tipo “bombilla” (u Omni), tiene un color asignado del color RGB 0,0,0 es decir, de color negro, por lo que el efecto es que se apaga.

Ahora surge un problema: cuando se apaga y se enciende una luz, además del color, interviene la intensidad de la luz. Director denomina a este parámetro como “attenuation”. Por lo tanto, al apagar tomamos la precaución de poner la luz a una intensidad normal de 1. Este parámetro ya lo estudiaremos más adelante y entenderemos porque para ponerle doble intensidad debemos poner el valor .5 en el script de encender la luz.

El script para apagar la luz de tipo bombilla quedará como sigue:

```
global mundo3D

on mouseUp
  mundo3D.Light[3].color = rgb(0,0,0)                    -- apagar la luz bombilla
  mundo3D.Light[3].attenuation = vector (1,0,0)
end
```

El script inverso, para poner la luz como al principio cuando se importo de MAX será:

```
global mundo3D

on mouseUp
  mundo3D.Light[3].color = rgb(255,0,0)                    -- poner la luz roja
  mundo3D.Light[3].attenuation = vector (.5,0,0)
end
```

4. Mover la luz bombilla.

En este punto vamos a atacar la programación para cambiar de posición las luces. Para ello, utilizaremos el método “translate” que tienen todos los componentes de la escena 3D y que nos permite cambiar el valor de la variable “position”. Para hacer esto utilizaremos un código como este:

```
mundo3D.Light[3].translate(0,5,0)                    -- mueve la luz 3 en el eje Y 5 unidades
```

Queremos hacer un botón que trabaje de forma continua, es decir que mientras lo tengamos pulsado cree el efecto deseado. Este mecanismo lo creamos mediante una variable que pueda tener dos valores 0 y 1. Si su valor es 0, el efecto no tendrá lugar, pero si vale 1, es porque se ha pulsado el botón y entonces, en un bucle y mientras valga 1, realizaremos el efecto, que en este caso es mover la luz. El script completo asignado al botón quedará así:

```
global mundo3D

property mover_luz_roja

on mouseDown
  mover_luz_roja=1
end

on mouseUp
  mover_luz_roja=0
end

on exitframe
  if mover_luz_roja = 1 then mundo3D.Light[3].translate(0,2,0)
  if mover_luz_roja = 1 then mundo3D.Model[11].translate(0,2,0)
end
```

Se puede observar que no solo movemos la luz, sino el modelo asociado a esta luz que se llama “Sphere02” y que podemos referenciar por su nombre o por su puesto en el vector de modelos, en este caso, el 11. Esto producirá un bonito efecto, y es que cuando la luz se mueva, veremos que su referencia también lo hace, con lo que todo el rato veremos el origen de la luz.

5. Girar la luz foco.

La luz focal la vamos a tratar de forma distinta a la luz de tipo bombilla ya que el foco tiene dos partes: la fuente de luz y el punto objetivo. Por lo tanto, en vez de trasladarla en los tres ejes, vemos más correcto girar la fuente de luz, para ver como modificamos su dirección.

Girar la luz tipo bombilla no tiene mucho sentido ya que emite la luz en todas direcciones y por tanto, aunque se puede realizar, no veríamos ningún efecto.

Para girar una luz cualquiera, utilizamos el método “rotate” que tienen todos los nodos de tipo luz, con lo que el código para rotar la luz será como sigue:

```
mundo3D.Light[4].rotate(2,0,0) -- mueve la luz 4 en el eje X en 2 unidades
```

Aquí aparece un nuevo problema, ya que tenemos que utilizar las coordenadas universales y la escena se visualiza desde un punto determinado y por tanto, no tienen porque coincidir las coordenadas de rotación con las coordenadas desde el punto de vista de la cámara que usamos. Por eso, a la hora de escribir las rotaciones correspondientes, es conveniente usar el método de prueba y error.

En este caso también queremos hacer un botón continuo, por lo tanto, el código asociado a uno de los botones de rotación del foco tendrá la misma forma que en el caso anterior pero cambiando la referencia a la luz y a su modelo asociado:

```
global mundo3D

property mover_luz_azul

on mouseDown
  mover_luz_azul=1
end

on mouseUp
  mover_luz_azul=0
end

on exitframe
  if mover_luz_azul = 1 then mundo3D.Light[4].rotate(2,0,0)
  if mover_luz_azul = 1 then mundo3D.Model[12].rotate(2,0,0)
end
```

Recordemos que no solo movemos la luz, sino que también giramos el cono que representa la luz focal, que en este caso es el modelo 12 en el vector de modelos de la escena.

Para girar el foco, solo nos hacen falta cuatro botones, ya que el cono del foco (y por tanto, la luz asociada) está fijada en el eje Z y por esa razón solamente se puede girar vertical u horizontalmente.

6. Colorear las luces.

Para modificar el color de las luces hemos creado una interfaz muy sencilla, ya que otras quizás más efectivas, complicarían la programación y no queremos que esto haga que se pierda la verdadera intención de este caso práctico.

Con el fin de colorear las luces hemos creado 6 botones para poner y quitar cada una de las luces de color básico, rojo, verde y azul, de tal manera que tendremos las siguientes combinaciones posibles de luces: roja, verde, azul, amarilla, turquesa, morada, blanca y negra. Como hemos escrito, con un interfaz con botones deslizables podríamos crear cualquier tipo de color para las luces.

Buscando que el efecto sea acumulativo y por ejemplo, podamos crear una luz morada al añadir color rojo a un color azul, vamos a utilizar variables globales para los colores rojo, verde y azul para ambas luces y las iniciamos con sus valores por defecto. Para disponer desde el principio de estas variables las creamos en el primer script, es decir en “Iniciación”. En el caso del foco quedarán así:

```
global focorojo
global focoverde
global focoazul

on enterFrame me
```

```
  focorojo = 0
  focoverde = 0
  focoazul = 255   -- valores para la luz foco
  …
```

En los botones donde se quite una determinada luz, bastará con poner su valor a 0 y luego actualizar el color de la luz con los nuevos valores. Para añadir la luz, el valor del color deseado será de 255, el máximo, y también tendremos que actualizar a continuación el color de la luz. Por ejemplo, para añadir al color que ya tenga, el color rojo a la luz focal, escribiremos un script que será:

```
global mundo3D
global focorojo
global focoverde
global focoazul

on mouseUp
  focorojo = 255                                          -- pone valor máximo a rojo
  mundo3D.Light[4].color = rgb(focorojo,focoverde,focoazul)     -- actualiza el color de la luz
end              -- los valores de focoverde y focoazul no se modifican aquí pero si se utilizan
```

Crearemos 6 botones por luz y los scripts serán todos tan sencillos como este.

7. Cambiar la intensidad de las luces.

En este apartado vamos a estudiar la forma de cambiar la intensidad de la luz. Este cambio de intensidad solo se puede efectuar en luces omni, no en direccionales, por lo que vamos a crear dos scripts asociados a los respectivos botones para aumentar y disminuir la intensidad de la luz de tipo bombilla.

El parámetro que contiene el valor de intensidad de la luz se llama “attenuation” y es un vector dónde solamente se emplea el primer valor, los demás se debe dejar a 0, siendo el valor por defecto de 1.

Este parámetro es parecido a “multiplier” en MAX pero aquí el valor de “attenuation” es inverso, esto quiere decir que a más valor, mayor es la atenuación de la luz, es decir, a menos distancia del foco se disipa, o sea, la luz es menos intensa.

Por lo tanto, para ponerle a la luz de tipo bombilla una atenuación menor que la actual, por ejemplo, que su intensidad sea el doble de lo normal, escribimos:

```
mundo3D.Light[3].attenuation = vector (0.5,0,0) -- su valor base es 1, así es la mitad de fuerte
```

Queremos hacer unos botones que el usuario pueda utilizar acumulativamente, es decir, que si presiona varias veces al botón de más intensidad, cada vez sea mayor la luz. Para hacer esto utilizamos una variable global que creamos en el script “Inicialización” y que por tanto estará disponible para los dos scripts, el que aumenta la intensidad y el que lo disminuye por que si no fuera así, la intensidad siempre partiría de su valor base y no funcionaría de forma acumulativa.

Además surge la cuestión de poner límites. Por ejemplo, la atenuación no puede llegar a ser nunca 0 ya que sería una luz que cegaría todo y por otro lado no puede ser infinitamente grande, por lo que despachamos este problema con una simple sentencia “if”, de tal forma que si superamos los límites, asignamos directamente el valor del límite que decidamos, o sea, explícitamente.

El script asociado al botón de incrementar la intensidad de la luz 3 quedaría:

```
global mundo3D
```

```
global bombillaintensidad
property bi

on mouseUp
  bombillaintensidad= bombillaintensidad -.1                    -- vamos disminuyendo la
atenuación
  if (bombillaintensidad < .1) then bombillaintensidad = .01    -- ponemos el límite en 0.01
  mundo3D.Light[3].attenuation = vector (bombillaintensidad,0,0)        – actualizamos
atenuación
end
```

Podemos observar algo que nos puede llamar la atención, y es llegado al límite, que pongamos el valor en 0.01. Esto lo hacemos para dar un gran salto en la intensidad de la luz directamente, porque se supone que es lo que el usuario está buscando al pulsar tantas veces el botón de disminuir la intensidad. Es una cuestión de interpretación y podría realizarse perfectamente de una forma distinta. El script asociado al botón de disminuir la intensidad de la luz 3 sería:

```
global mundo3D
global bombillaintensidad
property bi

on mouseUp

  bombillaintensidad= bombillaintensidad +.1                    -- vamos aumentando la
atenuación
  if (bombillaintensidad > 2) then bombillaintensidad = 2       -- ponemos el límite en 2
  mundo3D.Light[3].attenuation = vector (bombillaintensidad,0,0)  – actualizamos atenuación

end
```

En la siguiente imagen se puede observar el resultado del caso práctico. A destacar el complicado interface dónde hemos separado las cámaras de navegación de los controles de las dos luces, este es resultado de todas las características estudiadas en este caso.

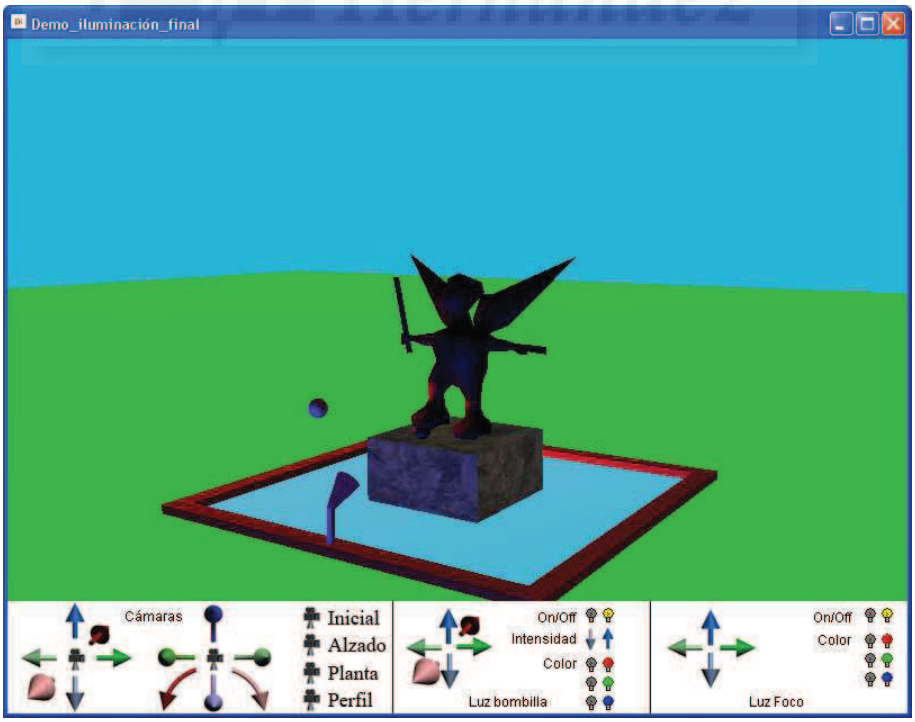


Imagen 5.10.cp.1.4: Vista del caso práctico en ejecución

Conclusiones:

La iluminación de una escena 3D es una tarea complicada. Primeramente porque es un tema muy técnico, ya que la comprensión de las luces y su interacción es un tema difícil en sí mismo. Nosotros hemos intentado realizar un caso práctico sencillo pero eficaz donde se pongan de manifiesto las similitudes y las diferencias de los dos tipos principales de luces 3D que usamos. Por lo tanto, creemos que es una cuestión difícil pero que podemos sortear no sin complicación y un poco de programación en Lingo.

Unidad 11: Cámaras en Director.

Introducción teórica:

1. Introducción a las cámaras virtuales.
2. Características de las cámaras.
3. Trabajar con las cámaras.

Casos prácticos:

11.1. Cámaras en Director.

1. Introducción a las cámaras virtuales.

En esta unidad vamos a estudiar como trabajan y cómo podemos manejar las cámaras en Director. Las cámaras permiten visualizar el mundo 3D de distintos modos y en distintas posiciones por lo que es un recurso muy útil para al creador 3D.

Lo primero que hay que tener en cuenta es que hay una gran diferencia entre las cámaras usadas en el mundo real y las cámaras de Shockwave 3D.

Las cámaras reales usan un sistema de lentes, objetivos y otros mecanismos que funcionan al pasar la luz a través de ellos. Por ejemplo: los objetivos de gran distancia representan sin perspectiva a los objetos, o si tenemos un objetivo tipo ojo de pez la visión se curvará. También utilizamos la profundidad de campo para realizar efectos de enfoque.

En las cámaras virtuales todos estos efectos no se pueden realizar por cuestiones de cálculo computacional. En realidad una cámara virtual es un método para proyectar un mundo 3D en un plano 2D, o sea, es un objeto del mundo 3D que sirve para representar proyecciones, es decir, imágenes.

2. Características de las cámaras.

Las cámaras Shockwave 3D tienen seis características con unos valores por defecto al crearse. Se presentan en la siguiente tabla:

Características	**Valor por defecto**
Posición	0,0,250
Rotación	0,0,0,
Proyección	perspectiva
Orientación	al origen del mundo 3D
Tamaño del plano de vista	34,516º
Color de fondo	negro

Tabla 5: Giro de cámara

Vamos a estudiar estas características una por una:

1) Posición. Es un lugar en el mundo 3D donde se sitúa la cámara y que se puede variar con el procedimiento “translate” como cualquier objeto del mundo 3D.

2) Rotación. Es el ángulo que varía con respecto a su posición inicial y que se puede modificar con el procedimiento “rotate” como cualquier objeto del mundo 3D.

3) Tipo de proyección. Una proyección es un método matemático para representar una escena 3D en un plano 2D. Existen dos tipos de proyecciones: perspectiva y ortográfica.

3.1. Proyección en perspectiva. Es la que tiene la cámara por defecto que existe en todo mundo Shockwave 3D.

Las cámaras con proyección en perspectiva se usan más que las cámaras ortográficas y se utilizan mucho para que el usuario actúe en primera persona (como muchos juegos) o en tercera persona (como recurso cinematográfico). También se presentan para panorámicas, exteriores, etc.

Los parámetros que intervienen en una proyección en perspectiva son 3:

1. Fieldofview, campo de vista: es muy importante, ya que determina el ángulo de vista vertical de nuestra cámara y por lo tanto, tiene efectos de acercar o alejar y del detalle con el que se visualizan los objetos.

2. Hither, plano trasero de recorte: todo lo que se encuentre entre la cámara y el plano hither no se representará.

3. Yon, plano delantero de recorte: todo lo que se encuentre más lejano que el plano yon no se representará.

Por lo tanto estos dos planos limitan lo que la cámara representa, lo que obliga a que el plano de vista esté entre uno y otro.

3.2. Proyección ortográfica. Hace que el tamaño aparente de los objetos con la distancia no se represente.

Este tipo de proyección se usa bastante cuando se necesitan vistas muy precisas de objetos o arquitecturas, así como organigramas, gráficos empresariales, simulaciones fotorealistas, etc., donde se tiene que visualizar el mundo 3D de forma muy objetiva sin intervención del punto de vista o donde se necesita ver un modelo desde varios puntos de vista a la vez.

Se utiliza el parámetro “orthoheight” en lugar de fiedofview. Mide las unidades verticales del plano de vista.

3.2.1. Proyección isométrica. Existe una vista ortográfica especial llamada isométrica que es cuando la cámara se sitúa exactamente en un ángulo de 45º respecto al modelo.

4) Orientación de la cámara. Permite direccionar la cámara hacia un punto en concreto del espacio. El procedimiento que se utiliza para orientar la cámara se llama “pointat” y admite dos parámetros que determinan la dirección: el “vector de orientación hacia delante”, con un valor por defecto (0, 0,-1) y el “vector de orientación hacia arriba” con un valor por defecto (0, 1, 0).

5) Tamaño del plano de vista. Ya hemos visto que es el parámetro que sirve para enfocar la cámara y que esta magnitud no se llama igual en los dos tipos de proyecciones. Se mide en unidades propias.

6) Color de fondo. Se establece este color con la propiedad “colorbuffer.clearvalue”. Cuando tenemos varias cámaras y se cambia la cámara activa hay que volver a establecer el color de fondo.

3. Trabajar con las cámaras.

En un mundo 3D pueden existir tantas cámaras como necesitemos. Estas se dispondrán en una lista de cámaras que podremos utilizar para referirnos a ellas individualmente.

El trabajo con las cámaras no es tan sencillo como pueda parecer, ya que el simple movimiento de una de ellas puede acarrear varios problemas. El más habitual es la pérdida de referencias y que en consecuencia “nos perdamos” en el mundo virtual.

También nos puede pasar que si estamos mostrando una escultura, al mover la cámara interactivamente con el ratón podemos hacer que nos introduzcamos dentro de los objetos que en teoría no deberíamos atravesar y toda la sensación de realidad virtual se nos venga abajo. Por supuesto, podemos hacer que los objetos no se puedan atravesar, pero es un tema bastante complejo que hay que resolver con programación.

Otra “sensación” que hay que tener en cuenta es que cuando movemos una cámara hacia la derecha, tenemos la sensación de que son los objetos los que se mueven hacia la izquierda.

Seguidamente pasamos a escribir algunas sentencias con cámaras para mostrar la sintaxis que tienen:

Crear una nueva cámara. Para ello solo se requiere un nombre:

mundo3D.newcamera(“camara_mia”)

Posicionar la cámara en un lugar del espacio 3D:

mundo3D.camera(“camara_mia”).translate(100,200,100)

Rotar la cámara, vamos a rotarla 45º en el eje X:

mundo3D.camera(“camara_mia”).rotate(45,0,0)

Enfocar la cámara. Esto depende del tipo de proyección, al no definirlo, toma el valor por defecto, por lo que la cámara será de tipo perspectiva:

mundo3D.camera(“camara_mia”).fieldofview = 20

Orientar la cámara:

mundo3D.camera(“camara_mia”).pointat(vector(0,20,0), vector(0,0,10))

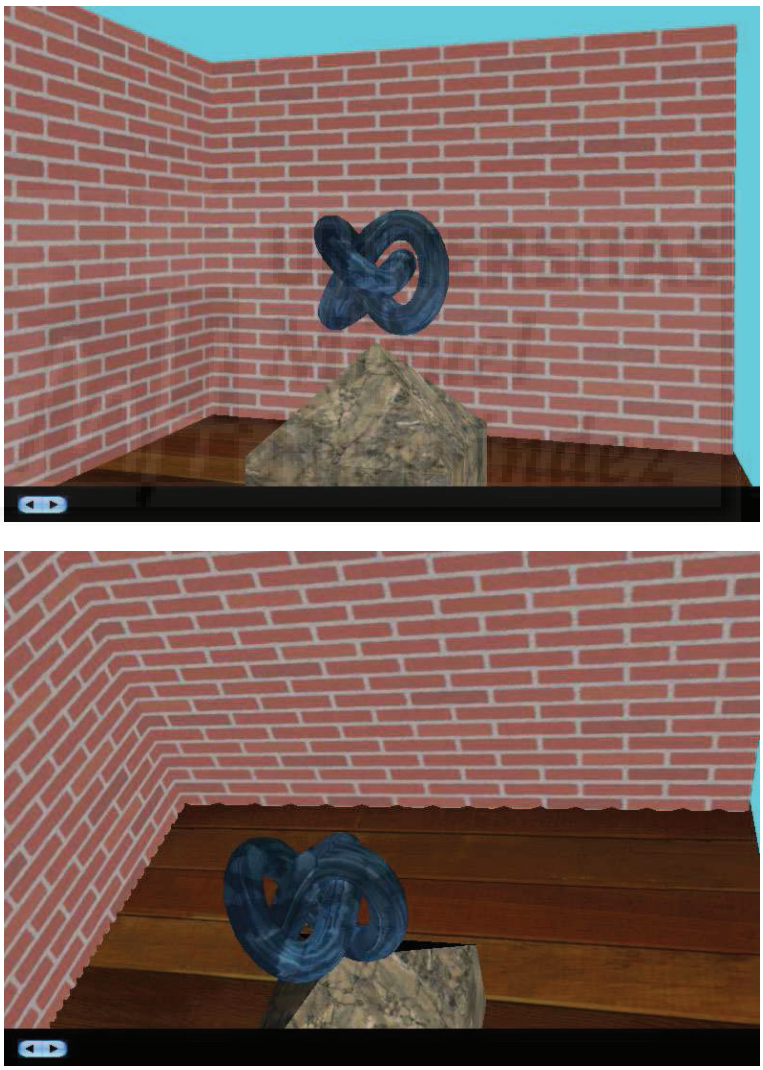


Imagen 5.11.1: Giro de cámara

Para cambiar el tipo de proyección, es decir el tipo de cámara, escribiremos:

mundo3D.camera(“camara_mia”).projection= #orthographic

Enfocar la cámara para el nuevo tipo ortográfico:

```
mundo3D.camera(“camara_mia”).orthoheight = 100
```

Color de fondo:

```
mundo3D.camera(“camara_mia”).colorbuffer.clearvalue = rgb(255,0,0)
```

Establecer la cámara activa del sprite activo. Hay que tener en cuenta que tendremos que reiniciar el color de fondo:

```
Sprite[1].camera = mundo3D.camera [2]
```

En la imagen anterior se puede observar como cambia la escena 3D al realizar un giro de la cámara en el eje Y.

Como conclusión de este tema, podemos decir que las cámaras son la vista del mundo 3D y por tanto tenemos que dominar su funcionamiento para controlar lo que mostramos. Podemos disponer de las cámaras que necesitemos para mostrar nuestra obra. Son totalmente manejables mediante Lingo y muchos de los parámetros de las cámaras virtuales se corresponden con las cámaras reales.

Caso práctico 11.1: Cámaras en Director.

Objetivo: Controlar y manejar las cámaras importadas de 3D Studio MAX con Lingo en Director. Las utilizaremos como interfaz de usuario para navegar por las escenas 3D.

Tiempo de realización: 30 minutos.

Pasos a realizar:

1. Creación de la escena 3D.
2. Las cámaras importadas.
3. Creación de los controles de cambio de cámara.
4. Creación de controles para la cámara libre.

1. Creación de la escena 3D.

Hemos creado una pequeña escena 3D con MAX con un fondo verde, un pequeño estanque y una escultura sobre un pedestal. La calidad de la obra es lo de menos, ya que pretendemos tener una escena sin muchos polígonos que nos sirva, en realidad, para trabajar con las cámaras, las luces y las texturas.

Imagen 5.11.cp.1.1: Vista perspectiva de la escena 3D en MAX

2. Las cámaras importadas.

En MAX hemos creado 4 cámaras. La explicación detallada de cómo se han creado la podemos encontrar en el caso práctico 6.1. El hecho es que tenemos 3 cámaras con nombre propio llamadas “planta”, “alzado” y “perfil” y una cámara que muestra la escena en perspectiva llamada “Camara01”.

Queremos que las 3 cámaras muestren de forma fija el objetivo en el mismo sentido en que indican sus nombres. La cámara en perspectiva la dotaremos de libertad de movimientos. Todas las cámaras son de tipo “Target” y los objetivos de las cuatro cámaras apuntan a la escultura en todo momento, por lo tanto, todos los “targets” serán fijos.

Debemos resaltar que antes de guardar el archivo y exportarlo a W3D tenemos que corresponder la vista activa con la cámara libre, es decir, la llamada “Camara01”, ya que cuando lo importemos, esta será la vista inicial que presentará la escena 3D en Director.

Por otra parte, la importación de la escena que incluye cámaras se realiza de la forma habitual, sin tener que reseñar nada destacable.

3. Creación de los controles de las cámaras.

En la siguiente imagen se aprecia el interfaz diseñado para este caso práctico. Cuenta con cuatro botones para tomar cada una de las cámaras. Todos los demás botones son para manejar la única cámara modificable: la cámara en perspectiva. Para ello tenemos seis botones para trasladar la cámara en los dos sentidos de los tres ejes y otros seis para girarla.

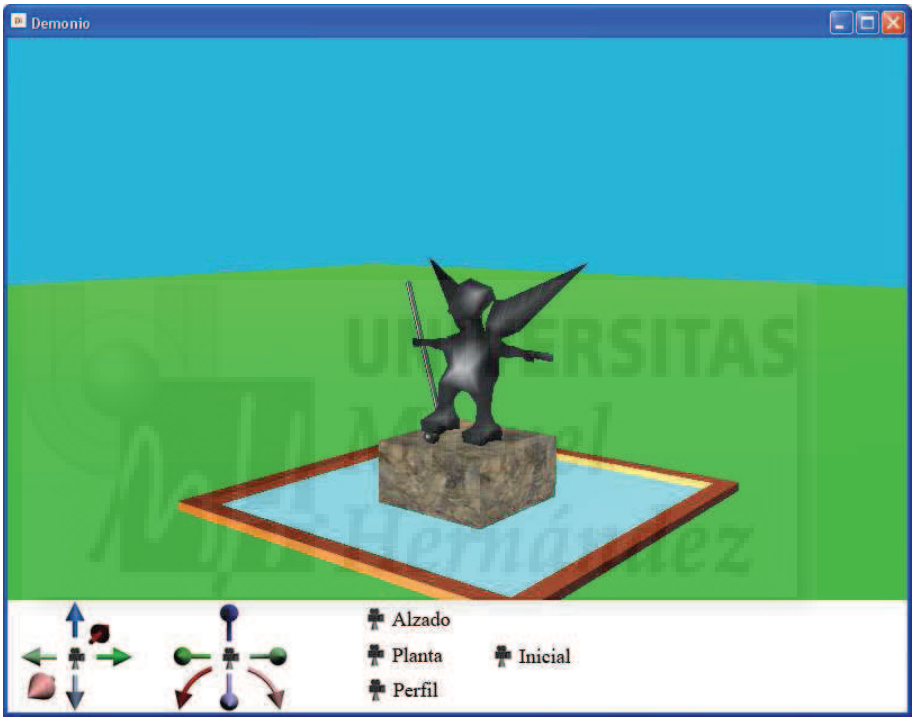


Imagen 5.11.cp.1.2: Vista con la cámara por defecto, la inicial

En la siguiente imagen se pueden observar cómo se visualiza el caso práctico cuando se pulsa sobre el botón “Perfil”. Este evento hace que la cámara que visualiza la escena pase a ser la cámara fija llamada “perfil”.

Para ello escribimos el siguiente script y lo asignamos al botón “Perfil”:

```
global mundo3D

on mouseUp
  sprite("escena").camera=mundo3d.camera("perfil")
end
```

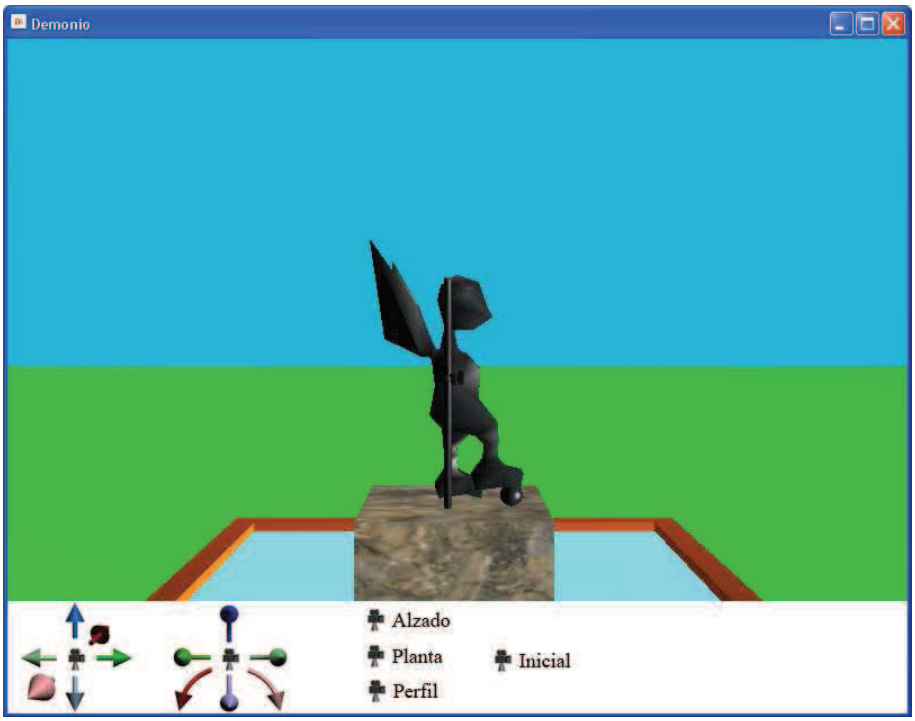


Imagen 5.11.cp.1.3: Vista con la cámara de perfil

Todos los demás controles de cambio de cámaras se programan de forma similar salvo el botón para pasar a la vista inicial. En este caso utilizaremos el nombre "DefaultView" para referirnos a ella.

4. Creación de controles para la cámara libre.

Los controles para la cámara libre (o inicial) son los que se visualizan en la siguiente imagen y que están divididos en dos grupos: los de traslación y los de giro de la cámara.

Una vez colocadas las imágenes de todos los botones, pasamos a programarlos. Queremos que funcionen como botones continuos, y esto significa que cuando se pulsan, se obtiene un efecto que se sigue produciendo hasta que se suelta el botón. Por ejemplo, si queremos trasladar la cámara hacia arriba pulsaremos el botón correspondiente e iremos viendo que la vista se va elevando hasta que dejemos de pulsarlo.

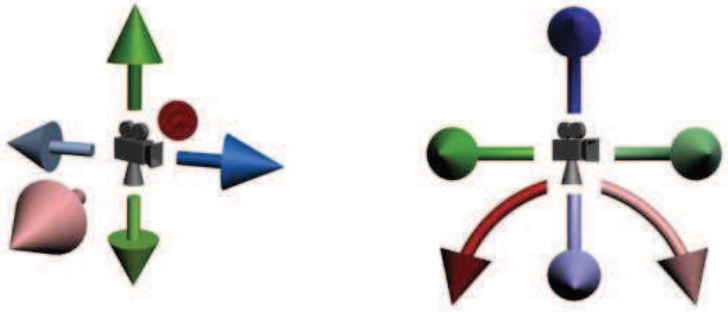


Imagen 5.11.cp.1.4: Botones de traslación y giro de la cámara libre

Este tipo de botones ya los hemos realizado con anterioridad. Necesitamos fundamentalmente dos cosas: una variable que guarde dos valores que represente si estamos o no pulsado el botón y por otra parte, un mecanismo de repetición que constantemente supervise si el botón es pulsado y en ese caso produzca el efecto.

La primera cuestión, se resuelve con una variable local que tomará el valor 1 si el botón es pulsado y 0 si no es así.

Para solventar la segunda cuestión, el mecanismo de repetición lo vamos a crear con el disparador “exitframe” que se produce cada vez que se sale del fotograma actual. Como sabemos que esto sucede 24 veces por segundo por defecto, nos sirve para realizar el script que necesitamos.

El script asociado al botón que mueve la cámara libre (llamada DefaultView) en el eje x en una unidad quedaría como sigue:

```
global mundo3D

property mover_x

on mouseDown
  mover_x = 1
end

on mouseUp
  mover_x = 0
end

on exitframe
  if mover_x = 1 then mundo3D.camera("DefaultView").translate(1,0,0)
end
```

El script asociado al botón que gira la cámara libre sería muy parecido y solamente tendríamos que cambiar el “translate” por un “rotate” de la siguiente forma:

```
on exitframe
  if mover_x = 1 then mundo3D.camera("DefaultView").rotate(1,0,0)
end
```

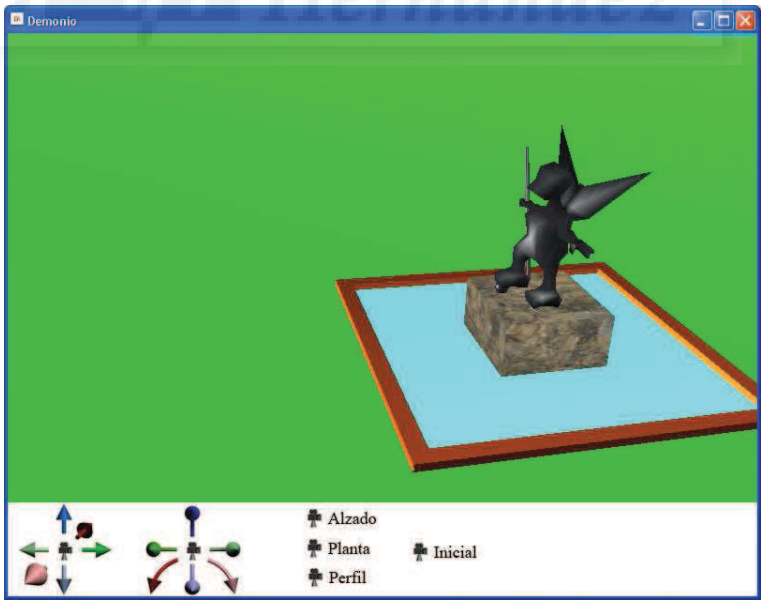


Imagen 5.11.cp.1.5: Vista mientras desplazamos la cámara libre

Debemos destacar que en ningún momento hemos utilizado el script “Programación” que asignamos al sprite de la escena 3D. Esto es debido a que toda la interacción se efectúa mediante botones independientes entre sí y la propia escena no es programada para nada, por lo que este script es prescindible para este caso práctico.

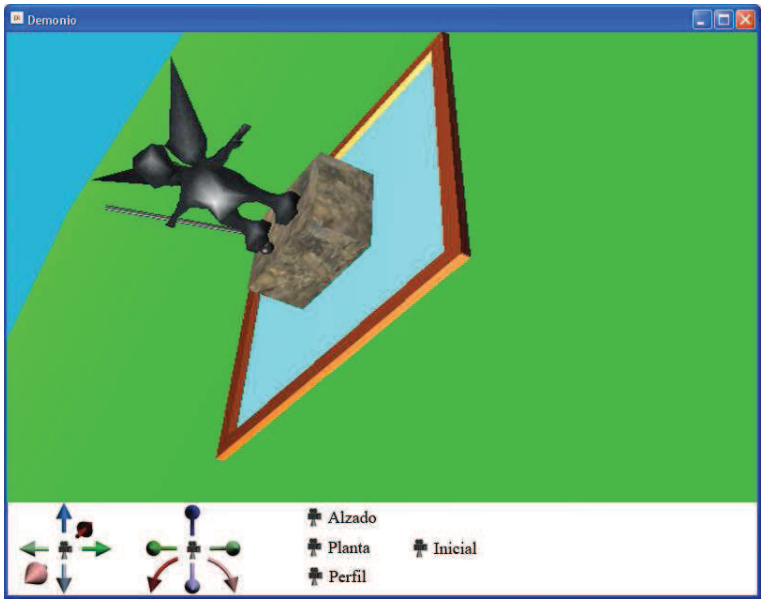


Imagen 5.11.cp.1.6: Vista mientras giramos la cámara libre

Conclusiones:

Las cámaras son objetos muy útiles y nos pueden servir para mostrar lo que el autor quiere directamente o también para que el usuario pueda navegar por la escena viéndola como le plazca por lo que tienen una especial importancia a la hora de diseñar el interface. Además son objetos fáciles de crear y manipular mediante Lingo.

Unidad 12: Materiales en Director.

Introducción teórica:

1. Introducción a los materiales en Director.

2. Asignación de mapas de textura.
    2.1. Importación de texturas.
    2.2. Utilizar imágenes de texturas externas en la escena 3D.
    2.3. Asignar una textura a un modelo 3D.

3. Control de superficies.
    3.1. Cómo ilustrar el control del Shader.
    3.2. Parámetros de las superficies de objetos 3D

4. Mapas de transparencias: canales alpha.
    4.1. Introducción a los mapas de transparencias: canales alpha.
    4.2. Creación de imágenes con canales alpha.
    4.3. Importación de imágenes con canales alpha.
    4.4. Scripts para asignar imágenes con canales alpha a los modelos 3D.

5. Texturas con mapas de reflejos.
    5.1. Introducción a los mapas de reflejos.
    5.2. Creación de los mapas de reflejos.
    5.3. Asignación de los mapas de reflejos a los modelos 3D.

6. Texturas animadas por transformación.
    6.1. Introducción a la aplicación de texturas con testuretransform.
    6.2. Transformación de escala de texturas.
    6.3. Transformación de rotación de texturas.
    6.4. Transformación de posición de texturas.
    6.5. Modos de aplicación de las texturas a los modelos.

7. Texturas animadas por intercambio.
    7.1. Introducción a las texturas animadas por intercambio.
    7.2. Creación de Casts para contener las imágenes de la animación.
    7.3. Animación de texturas por programación.

Casos prácticos:

12.1. Asignación de texturas.

12.2. Control de superficies.

12.3. Texturas con mapas de transparencia: canales alpha.

12.4. Texturas con mapas de reflejos.

12.5. Texturas animadas por transformación.

12.6. Texturas animadas por intercambio.

1. Introducción a los materiales en Director.

Vamos a tratar los materiales en Director estudiando sus características principales y resolviendo los puntos que son comunes al problema de la visualización de los modelos en 3D que componen una escena tridimensional que se ejecuta en tiempo real.

El aspecto de su ejecución en tiempo real es fundamental, ya que va a condicionar toda la representación. Es el talón de Aquiles de todo sistema de realidad virtual. La razón es sencilla, los algoritmos actuales para representar una escena realista son de sobra conocidos y se pueden implementar en programas que recrean mundos virtuales como 3D Studio MAX, pero cada imagen puede tardar de segundos a horas en generarse.

Un sistema en tiempo real como Director, tiene que representar unos 24 fotogramas por segundo para engañar al ojo humano y visualizar el mundo virtual como el real y por lo tanto no puede utilizar los mismos algoritmos y recursos que los que utiliza MAX.

La solución es utilizar variables que producen la representación a menor calidad o simplemente no tiene en cuenta algunas características de los mapas que sí calcula MAX, como son los mapas de relieve o los reflejos.

Por otra parte, Director es una solución muy buena en el tratamiento de materiales, ya que estos se representan con rapidez y eficacia. Además, se puede elegir la calidad de representación de las texturas y otros parámetros si fuera necesario.

Podemos decir que Director permite representar materiales con un sombreador o shader por defecto que tiene propiedades y métodos para resolver los aspectos más habituales y además permite los mapas de textura y los mapas de opacidad.

Veremos en los puntos que siguen cómo usando Lingo podemos crear todo tipo de texturas con opacidad y reflejos e incluso efectos especiales como texturas animadas.

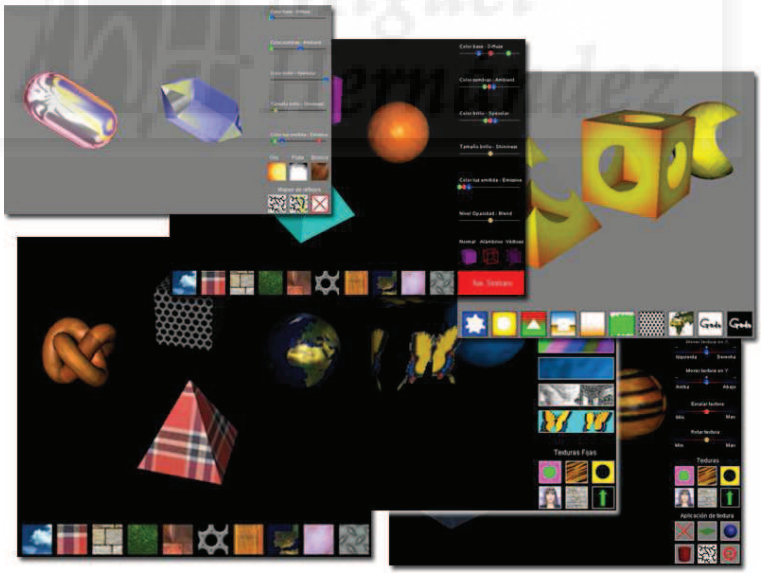


Imagen 5.12.1: Superficies en Shockwave 3D

2. Asignación de mapas de textura.

En este punto vamos a estudiar el problema de cómo importar texturas de tipo bitmap como imágenes .jpg a nuestros proyectos y luego hacer que estas texturas se apliquen a los modelos tridimensionales.

Además veremos cómo asignar estas texturas interactivamente, es decir que se puedan intercambiar en tiempo de ejecución. Esto permitirá cambiar las texturas a voluntad del espectador y por ejemplo, ver cómo quedan unos zapatos con distintas pieles o distintos colores. Vamos a estudiar cómo se aplican estas texturas sobre las distintas geometrías de los modelos, ya que una textura no "envuelve" de la misma manera a un cubo que a una esfera o una escultura.

Por supuesto que para realizar un trabajo de este tipo utilizaremos Lingo. En la imagen inferior se puede observar el caso práctico "Asignación de texturas" que ilustra cómo texturizar modelos 3D de forma interactiva.

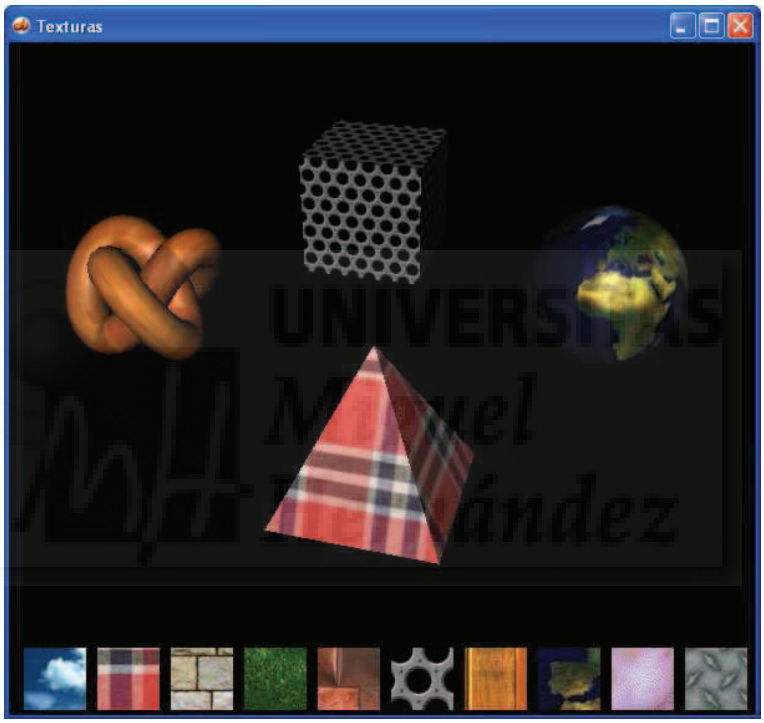


Imagen 5.12.2: Caso práctico de asignación de textura a superficies

2.1. Importación de texturas

En este texto vamos a considerar que las imágenes que utilizaremos como texturas son bitmaps con formato .JPG. La razón es que para crear texturas realistas, este formato permite una gran calidad de imagen con un tamaño relativamente pequeño con respecto a otros formatos de mapas de bits (bitmaps).

Las imágenes de textura no hace falta que sean excesivamente grandes, por supuesto, depende de la calidad que se quiera obtener, pero para la mayoría de los trabajos basta con un tamaño de 200 x 200 píxeles ya que, como estudiaremos posteriormente, existen otros muchos aspectos que inciden en la visualización de los objetos.

Las imágenes se importan a Director por el método convencional, es decir, con el comando File > Import. Pasarán a ser un castmember más.

2.2. Utilizar imágenes de texturas externas en la escena 3D

El primer problema que tenemos que resolver es cómo “introducir” imágenes externas en la escena 3D. Este problema lo resuelve la función newtexture( ) que pertenece al objeto de la propia escena de Shockwave 3D.

La función newtexture tiene la siguiente sintaxis:

```
Escena3D.newtexture("nombre_imag",#fromcastmember,member("imag_miembro"))
```

Donde “nombre_imag” es la referencia que tendrá la imagen en la escena 3D e “imag_miembro” el nombre del miembro (member) que es de tipo imagen. Por lo tanto, un ejemplo de esta función será:

```
Mundo3D.newtexture("marmol",#fromcastmember,member("imagen_marmol"))
```

Tenemos que tener cuidado al intercambiar las texturas de un modelo, ya que newtexture dará un error si intentamos “cargar” dos veces la misma textura. Es decir, si ya hemos usado newtexture para una imagen, debemos utilizar el nombre interno de la imagen en la escena 3D y no volver a llamar a newtexture con los mismos parámetros.

Para resolver esta cuestión, debemos pensar en la interfaz que queremos crear para el espectador de la obra. Si queremos que el usuario pueda cambiar la textura a un modelo en particular de entre muchos, podemos incluir en el script “Programación” todas las texturas que vayamos a utilizar a lo largo del proyecto y luego crear scripts asociados a botones que serán los que finalmente utilicen esta textura y la asignen al modelo, que deberá ser referenciado de alguna manera, por ejemplo, haciendo previamente clic sobre él.

Por lo tanto, el script “Programación” no será sencillo, ya que tendrá que resolver dos temas fundamentales: incorporar las imágenes de textura a la escena 3D y controlar el modelo 3D seleccionado mediante un clic para ser el sujeto del cambio de textura.

Esto conlleva otro tema de interfaz a discutir: si una vez seleccionado el objeto, queremos denotar esta selección de alguna manera. Como esto es un tema de interfaz menor, lo resolveremos más adelante en un caso práctico especial de interfaces. Además, en nuestro caso, si cambiamos de alguna manera el aspecto del elemento seleccionado intervendrá en la visualización de la nueva textura, cosa que no deseamos, por lo que no haremos notar el elemento seleccionado de forma alguna.

El script “Programación” será algo parecido a lo que sigue, suponiendo que tenemos dos modelos 3D y tres imágenes de texturas a elegir para cada uno de ellos:

```
global mundo3D                          -- variables globales
global modeloseleccionado               -- para pasarla a los scripts que asignan la textura

property psprite                        -- variables locales
property tocado


on beginSprite me                       -- inicialización de la presente aparición (sprite)
  pSprite = sprite(me.spriteNum)
                                -- se añaden las texturas al mundo 3D
  mundo3D.newtexture("textura01",#fromcastmember,member("imagen_textura_01"))
  mundo3D.newtexture("textura02",#fromcastmember,member("imagen_textura_02"))
  mundo3D.newtexture("textura03",#fromcastmember,member("imagen_textura_03"))
end

on mouseDown                            -- cuando se hace clic con el ratón
  tocado = psprite.camera.modelUnderLoc(the clickLoc - point(psprite.left,psprite.top))
                                        -- se detecta el objeto
```

```
  if (tocado = Void) then                          -- si no se toca ningún objeto, no hacer nada
    mundo3D.model("modelo01").rotate(0,0,10)       -- esto es un truco para “no hacer nada”
    mundo3D.model("modelo01").rotate(0,0,-10)

  else          -- si el usuario elige un modelo, capta el objeto clikeado para usar después
    if (tocado.name= "modelo01")      then modeloseleccionado = 1
    if (tocado.name= "modelo02")      then modeloseleccionado = 2
  end if

end
```

2.3. Asignar una textura a un modelo 3D

Tendremos que escribir un script que asociado al sprite de un botón permita asignar una de las texturas a un determinado modelo. El modelo en cuestión lo sabemos, pues en el script de “Programación”, gestionamos que la variable global llamada “modeloseleccionado” contendría un dato numérico que lo identifica.

Todos los objetos 3D de la escena 3D contienen datos sobre su superficie. Esta superficie esta definida por la propiedad “Shader” y esta propiedad tiene una subpropiedad llamada “Texture”. Esta propiedad del objeto, será la que finalmente deberemos modificar con Lingo en este script.

El tema de los shader es de suma importancia, ya que aquí radica toda la información de los materiales de los modelos que Shockwave 3D utiliza para su representación, por lo que lo estudiaremos en profundidad en el siguiente punto.

Como hemos escrito, la asignación de una textura a un objeto, supone la modificación de la propiedad texture, que pertenece a la propiedad shader de cada objeto de la escena 3D. Para saber el nombre interno de los shader de cada modelo podemos utilizar la característica de inspección de la escena 3D que aporta la ventana de script.

El script para asignar la textura llamada “texture01” a un modelo seleccionado previamente y con los nombres de shader convencionales sería así:

```
global mundo3D                          -- variables globales
global modeloseleccionado

on mouseDown

   if (modeloseleccionado= 1) then      -- si el modelo seleccionado es el 1 cambia su textura
      mundo3D.shader("DefaultMaterial2").texture = mundo3d.texture("texture01")
   if (modeloseleccionado= 2) then      -- repetir lo mismo con los demás
      mundo3D.shader("Default Material0").texture = mundo3d.texture("texture01")

end
```

3. Control de superficies.

Este tema es de vital importancia ya que las superficies contienen los datos para que el sistema represente los modelos de la escena 3D. Como es de suponer, en las superficies van a intervenir una gran cantidad de datos, por lo que los estudiaremos uno a uno, para ir asimilando su significado, los valores que pueden tener y como se corresponden estos valores con su representación y además, cómo modificar estas variables en Lingo.

La posibilidad de realizar estos cambios en la superficie de los objetos mediante la programación va a proporcionarnos grandes posibilidades, ya que podremos modificar interactivamente el color, el brillo, la opacidad,... de un modelo al gusto del usuario.

3.1. Cómo ilustrar el control del Shader: el caso práctico control de superficies

Como hemos escrito antes, todos los objetos 3D contienen datos sobre su superficie en unas estructuras de datos llamados “Shader” (o sombreadores en español). Estos shaders contienen múltiples datos a los que tenemos que acceder para modificarlos si queremos experimentar con los cambios en la representación de los modelos.

Para entender mejor este tema hemos creado el caso práctico 12.2 llamado “Control de superficies” donde se ilustra el funcionamiento de los shaders y que pretende ser una especie de “laboratorio de superficies”. Esto implica que tendremos que hacer algunos cambios en la estructura de la película básica que utilizamos en todos nuestros casos prácticos debido a la complejidad estructural de este ya que incluyen controles de inferface como botones deslizadores y scripts “especiales”.

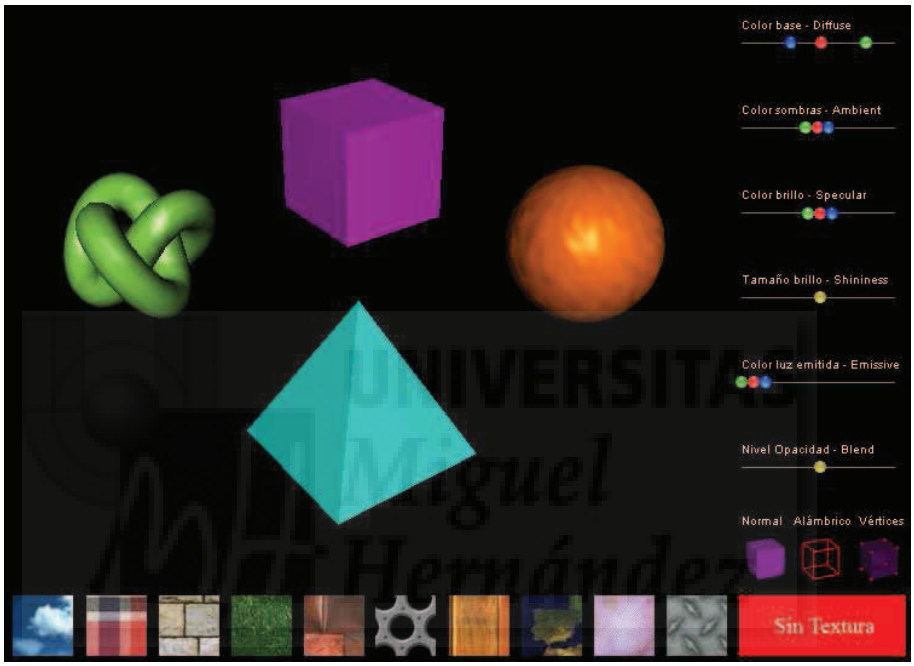


Imagen 5.12.3: Caso práctico de control de superficies

Sobre todo, los cambios en la estructura de scripts serán dos: la creación de un script no asignado a ningún sprite, que hará la función de programa independiente para ser llamado desde distintos puntos de otros scripts y por otra parte la creación de un script que nos devuelve el shader de un determinado modelo con el fin de simplificar el código.

Una vez realizados estos cambios (que están detalladamente explicados en el caso práctico) los cambios en las variables del shader serán tan sencillos como estos códigos:

```
mundo3D.model("Modelo01").visibility = #both
mundo3D.shader(shadertocado).diffuse = rgb(255,0,0)
```

La forma con que finalmente se visualiza un modelo 3D depende de muchos parámetros. Vamos a distinguir dos grupos: texturizados y no texturizados.

Debido a esta división, en el caso práctico de control de superficies se diseñó un interfaz para que se visualizase claramente esta separación. Como se puede observar en la imagen superior, las texturas se asignan con controles dispuestos en la parte inferior y los demás parámetros que no tienen que ver con la textura, se encuentran en la parte derecha.

Aunque la textura es un parámetro más del Shader, es un tema con el que tenemos que tener cuidado porque cambian totalmente el aspecto de cualquier modelo. El tema de cómo se

asignan las texturas a los modelos lo tratamos en el tema anterior, por lo que ahora vamos a dedicar nuestra atención a todos los demás parámetros que también son muy importantes sobre todo en proyectos artísticos de calidad.

3.2. Parámetros de las superficies de objetos 3D

Como hemos escrito, vamos a centrarnos en el estudio de los parámetros independientes de la textura y por tanto, exista esta o no, se aplicarán igualmente. Otra cosa es que se visualicen claramente ya que la textura puede ocultar sus efectos.

Tenemos que tener en cuenta que cuando algún parámetro se codifique mediante un color, utilizaremos el código RGB (Rojo, Verde, Azul cada uno de los cuales puede tomar valores de 0 a 255) y la función RGB( ) para definir ese color. Esta codificación hace que si los tres colores tienen el mismo valor o cercano, tomen tonos grisáceos.

Los modelos 3D tiene un sombreador básico compuesto por 3 colores: el color base o diffuse, el de las partes más sombreadas o ambient y el de las partes más iluminadas o specular. Este sombreador es del que parten todos los programas que trabajan con materiales y objetos 3D como el caso de MAX, aunque estos programas tienen la posibilidad de utilizar sombreadores más complejos y especializados como sombreadores para metales, para superficies mate, etc.

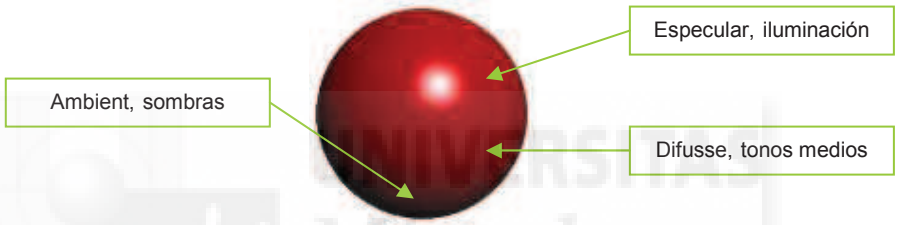


Imagen 5.12.4: Color difusse, especular y ambient

Los parámetros que podemos modificar del shader estándar son:

- Color base o diffuse.

Este es el color base del objeto. A este color se le denomina técnicamente Diffuse y tiene el mismo nombre la propiedad del shader que se puede modificar mediante Lingo.

Con el siguiente código, tintaremos del modelo seleccionado de rojo. Obsérvese el valor de rgb, dónde se le asigna el máximo valor (255) al rojo y valor nulo a verde y azul.

```
mundo3D.shader(shaderseleccionado).diffuse = rgb(255,0,0)
```

- Color de sombras o ambient.

Este color sombrea el objeto. Es como si el objeto tomase un color del entorno y por eso también se le denomina ambiente o “ambient”.

Con el siguiente código, daremos un color verdoso a las sombras de la superficie del modelo seleccionado, por ejemplo, para simular que está bajo la típica luz de neón verde de un cartel luminoso.

```
mundo3D.shader(shaderseleccionado).ambient = rgb(0,200,0)
```

- Color de brillo o specular.

Este color es el del brillo. En algunos softwares en español se le denomina lustre. Este parámetro es importantísimo, ya que el color, el tamaño y la forma specular definen muy bien el

tipo de material que queremos recrear. Por ejemplo, el metal tiene un lustre característico. El plástico, el cristal, etc., cada uno tiene el suyo característico.

Con el siguiente código, pondremos un color blanco al brillo del modelo elegido:

```
mundo3D.shader(shaderseleccionado).ambient = rgb(255,255,255)
```

- Tamaño de brillo o shininess.

Como hemos escrito, el tamaño del brillo es muy importante para dotar a la superficie de realismo. En la imagen que sigue se puede observar el mismo modelo con los mismos valores de superficie salvo el tamaño del brillo

Imagen 5.12.5: Tamaño de brillos distintos en el mismo modelo

El tamaño del brillo tiene unos valores inversos a lo normal, porque cuanto más alto es el valor más pequeño es el brillo. Los valores van de 0 a 100 y el valor máximo hace que el brillo desaparezca.

Para poner un brillo medio utilizaremos el siguiente código Lingo:

```
mundo3D.shader(shaderseleccionado).shininess = 50
```

- Color de autoiluminación o emissive.

Este parámetro define la luz emitida desde el objeto mismo, o sea, la autoiluminación, que es importante para definir algunos materiales como los tubos de neón, bombillas, etc.

Si tenemos una bombilla y queremos que tenga un aspecto distinto si está encendida o si está apagada, además de gestionar una luz, podremos utilizar la propiedad emissive para dar más realismo a la superficie de esta bombilla.

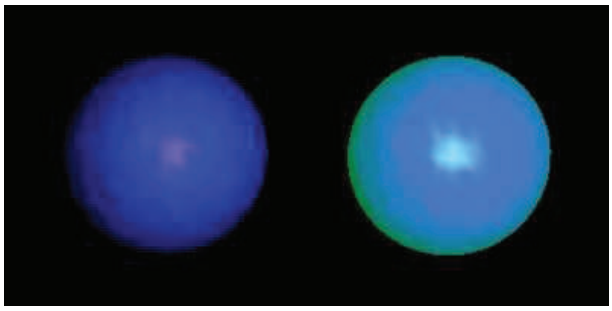


Imagen 5.12.6: Autoiluminación distinta en el mismo modelo

Este parámetro toma como entrada un color RGB, por lo que si queremos realizar la bombilla antes descrita deberemos asignarle un color muy cercano al blanco, pero si ser totalmente blanco, ya que la veríamos totalmente plana, sin ninguna curva, no parecería estar en tres dimensiones. El código en Lingo podría ser así:

```
mundo3D.shader(shaderseleccionado).emissive = rgb(200,200,200)
```

◆ Nivel de opacidad o blend.

La opacidad define la transparencia de una superficie. Muchos de los materiales más habituales tienen cierto grado de transparencia.

El shader tiene una propiedad llamada Blend para definir la opacidad. Sus valores van desde 0 que significa la transparencia total a 100 que significa la opacidad total.

En la imagen siguiente, se puede observar el mismo modelo pero con distintos valores para esta propiedad. A la izquierda la caja tiene un blend de 15, en el centro tiene un valor de 50 y a la derecha de 100.

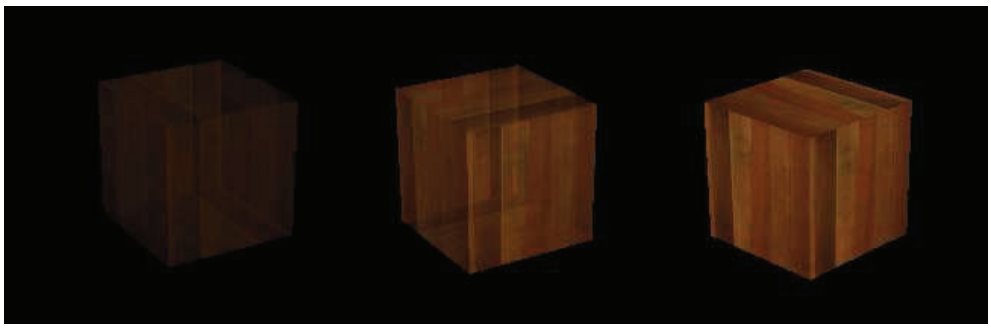


Imagen 5.12.7: Opacidades distintas en el mismo modelo

Para modificar este valor, en el script correspondiente escribiremos lo siguiente:

```
mundo3D.shader(shaderseleccionado).blend = 75
```

En este punto tenemos otro aspecto importante que tratar, y es que cuando un objeto es semitransparente, debe dejar visibles las caras posteriores, cosa que no hace falta en superficies que son opacas totalmente. Por ejemplo, si estamos modelando un jarrón de cristal transparente, debemos ver la parte de atrás del jarrón tal como ocurre en la realidad.

Sin embargo, Director no funciona de esta manera. Y la razón es sencilla, si las caras de atrás de un modelo no se van a visualizar desde el punto de vista del espectador, ¿para que las va a calcular el software?, eso que se ahorra en cálculo de la escena, y con más razón cuando es en tiempo real.

Para resolver esta cuestión basta con escribir en Lingo una sentencia que haga que el sistema represente todas las caras del modelo, sean visibles o no. Esto se hace así:

```
mundo3D.model("Modelo01").visibility=#both
```

◆ Modo de renderizado o renderstyle

El shader puede representar caras, aristas o puntos y lo hace mediante la propiedad llamada renderstyle.

La diferencia entre cada uno de estos modos se visualiza en la siguiente imagen:

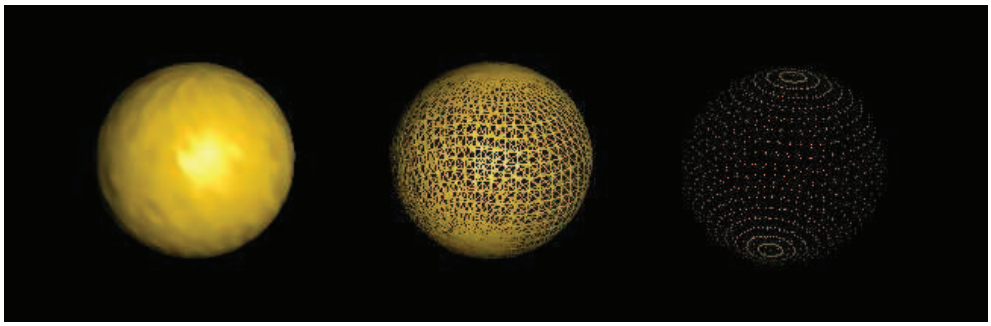


Imagen 5.12.8: Distintos modos de representación de un mismo modelo

La propiedad renderstyle puede tomar uno de estos tres valores: fill, wire o point, es decir, lleno, alambre o punto, aunque en español estas representaciones se suelen llamar normal, alámbrica y de vértices. El valor por defecto, es por supuesto, fill. Para modificar este valor por una representación alámbrica escribiremos:

```
mundo3D.model("Sphere01").renderstyle = #wire
```

4. Mapas de transparencias: canales alpha.

4.1. Introducción a los mapas de transparencias: canales alpha.

Los mapas de transparencias se usan mucho por parte de los diseñadores 3D por su facilidad conceptual y sus grandes posibilidades a la hora de modificar la geometría visible de un modelo 3D.

La idea fundamental es que de la misma manera que se aplica un mapa de textura, es decir, una imagen sobre la superficie de un objeto, podamos también aplicar otro mapa, o sea, otra imagen pero que defina la transparencia de la superficie. De esta manera, tendremos dos mapas: el de textura que define la vista y el de transparencia que define la opacidad. La opacidad y transparencia definida puede ser total o parcial. Los mapas de transparencia se definen por escalas de grises. El negro equivale a la transparencia total, los grises a la semitransparencia y el blanco puro a la opacidad total.

Pero aparece inmediatamente un problema: los shaders de los modelos 3D en la escena 3D, solo admiten una textura.

La solución es bien sencilla: se trata de crear una sola imagen que contenga tanto la información de color (mapa de textura) como la información de opacidad (mapa de opacidad). Estas imágenes o mapas se conocen como imágenes con canales alpha.

Sabemos que las imágenes de textura son bitmaps (mapas de bits), o sea, una matriz de puntos llamados píxeles que están definidos por un determinado color. Estos colores se pueden definir de muchas maneras, la más utilizada es mediante el código llamado RGB. Esto hace que cualquier imagen se pueda definir con 3 canales. Un canal donde se tendría la información para todos los píxeles que componen la imagen del color rojo, es decir, que cantidad de color rojo (codificada de 0 a 255) tiene cada píxel. Esto se repetirá para el color verde y para el azul.

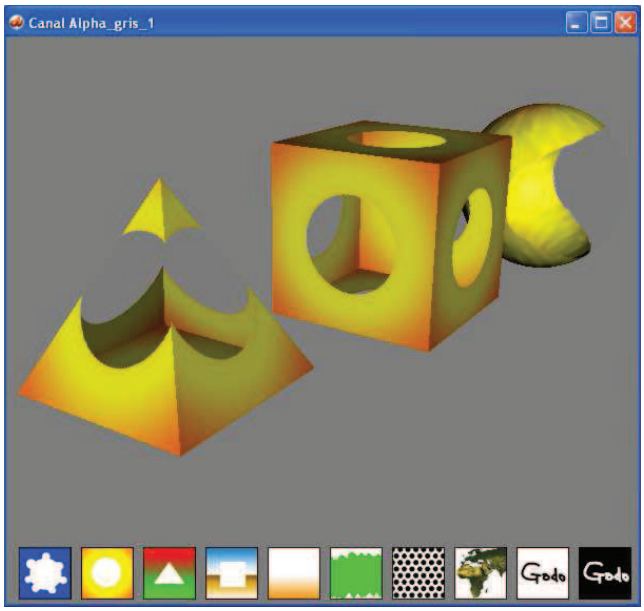


Imagen 5.12.9: Vista del caso práctico de mapas de transparencia

Las imágenes con canal alpha, añaden un canal más con la información de opacidad codificada como una imagen en una escala de grises y que el programa de diseño 3D representará como opacidad y así resolvemos el problema de integrar dos mapas e uno.

Para crear las imágenes con canales alpha utilizaremos un software externo como es Photoshop y luego las importaremos de una manera especial y las asignaremos al modelo con Lingo pero también modificando algunos parámetros del shader. Por lo tanto, es un tema que tenemos que tratar con suma atención.

Esta propiedad podría entrar en conflicto con la propiedad blend del shader, pero no es así porque blend modifica el valor de opacidad para todo el modelo a un valor constante. Con una imagen de canal alpha, se pueden elegir qué partes de un modelo 3D serán transparentes y a qué valor. Además, el hecho de que se pueda "pintar" la transparencia, hace que sea muy fácil de realizar incluso para los no iniciados y de esta forma poder "modelar" ya que podemos crear orificios, roturas, superficies translúcidas, etc.

Por ejemplo, en la imagen de arriba se muestra una vista del caso práctico "mapas de transparencia" donde hemos aplicado la textura 2 empezando por la izquierda a los tres modelos y se puede apreciar dos conclusiones: que permiten modificar la geometría visible y por otro lado que cada modelos se envuelve (en la jerga de los autores 3D, se "mapea") de forma distinta.

4.2. Creación de imágenes con canales alpha.

Para crear las imágenes con canales alpha, utilizaremos un programa externo como ®Adobe ®Photoshop ya que es un trabajo especializado y por eso tenemos que utilizar un software especial. Supondremos que tenemos unos conocimientos básicos de cómo funciona PhotoShop y los conceptos de capas y canales.

El resultado de nuestro trabajo debe ser fichero con extensión .bmp que pueden incluir 4 canales (RGB y alpha) y son los que Director puede importar con esta característica.

Como hemos escrito anteriormente, tendremos que diseñar tanto la textura como la opacidad en una sola imagen. Para diseñar la textura podremos hacer uso de todos los recursos que queramos, como degradados, capas, etc. Si queremos un buen resultado deberíamos utilizar una profundidad de 32 bits para tener mapas de textura de gran calidad. Aunque luego a la hora de utilizarlas en Director, tenemos que especificarlo con Lingo.

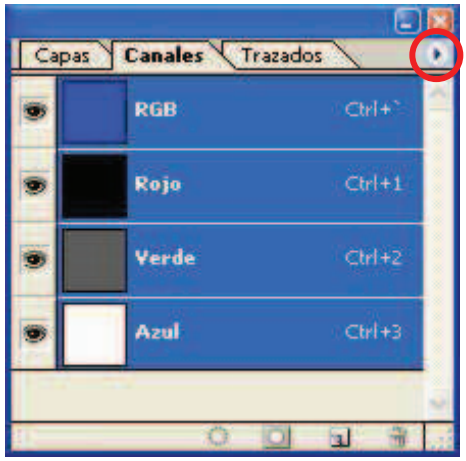


Imagen 5.12.10: Los canales normales antes de crear el canal alpha

Una vez definido el tamaño de la imagen que no debe ser muy grande (200 x 200 píxeles puede ser adecuado) crearemos el diseño que queramos. Por ejemplo, podemos hacer un diseño liso con el bote de pintura, o hacer un degradado o si queremos una piel de tigre podremos importarla de Internet. El caso es que trabajamos como siempre para la primera parte del trabajo: el mapa de textura.

Para realizar un canal alpha, accedemos al menú Ventana > Canales y aparecerá la ficha canales como se muestra en la imagen anterior. Para crear un canal nuevo pulsar sobre el botón de la esquina superior derecha que parece un botón de “Play” y que se señala en la misma imagen y esto mostrará el menú de canales.

En el menú de canales debemos pulsar la opción “nuevo canal…”, lo que produce que aparezca una nueva ventana llamada “nuevo canal” y donde pulsaremos el botón “OK”.

Tendremos un canal más llamado Alfa 1 en la lista de canales y en este canal debemos pintar con grises para definir el mapa de opacidad. Si queremos una zona totalmente transparente, pintaremos en este canal con una pintura totalmente negra, si queremos cierto nivel de opacidad pintaremos con distintos valores de grises.

En cierto sentido es como generar el mapa de textura, ya que podemos utilizar cualquier herramienta como el bote de pintura, el degradado, etc., salvo que el color debe ser siempre una escala de grises. Por ejemplo, en la imagen inferior se puede ver un canal alpha que producirá una especie de agujero irregular en el modelo al que se aplique y que se utiliza en el caso práctico de mapas de transparencias.

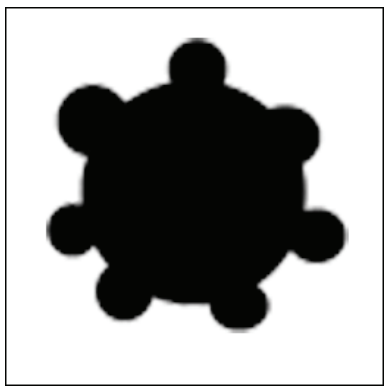


Imagen 5.12.11: Vista del canal alpha pintado

A la hora de guardar el archivo elegiremos el formato bmp, y nos aseguraremos de tener pulsada la opción “canales alfa” para que se guarde correctamente.

4.3. Importación de imágenes con canales alpha.

Para importar la imagen con canales alpha procederemos como con cualquier otra imagen, es decir, ejecutamos el comando File > Import y aparecerá la ventana “Import Files into Internal” para que se convierta en un member mas. Elegiremos el fichero de que se trate y al pulsar el botón “Import”, Director detecta que es una imagen y aparecerá la ventana “Image Options for XXX.bmp” tal como se muestra en la imagen siguiente.

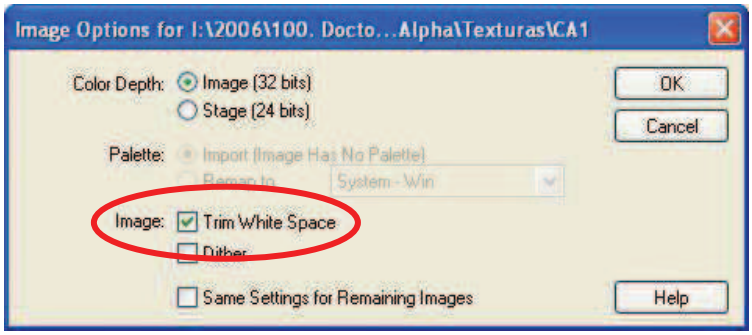


Imagen 5.12.12: Tipo de imagen en la importación a Director

Atenderemos especialmente al parámetro que está resaltado en rojo en la imagen y que debe estar deshabilitado ya que si no, la imagen se importará sin canales alpha.

También es muy importante el parámetro de profundidad de color (Color Depth) que debe corresponderse con la calidad de la textura que queremos utilizar y con el parámetro “renderformat” que será utilizado por Director para efectuar la representación de la escena con una calidad determinada.

Debemos acceder a la ventana del inspector de propiedades del castmember que hemos importado y en la ficha Bitmap debe estar habilitada la opción “Use Alpha” como muestra la siguiente imagen.

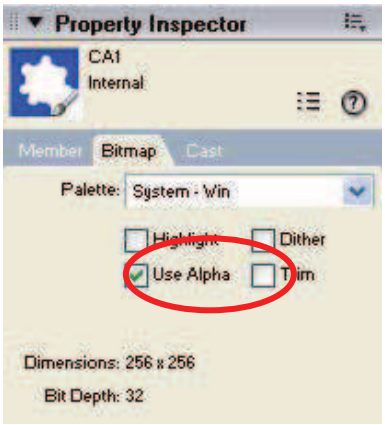


Imagen 5.12.13: Declarar el uso de canales alpha en el bitmap importado

4. Scripts para asignar imágenes de canales alpha a los modelos 3D.

En puntos anteriores ya hemos estudiado cómo integrar texturas en Shockwave 3D y asignar texturas a los modelos tridimensionales y ahora lo vamos a realizar de la misma forma.

En los casos prácticos relacionados utilizaremos botones para que sea el usuario de forma interactiva el que elija las distintas texturas para los modelos 3D. Por ello los scripts se parecen mucho a los comentados ya anteriormente.

Por ejemplo, si en el punto precedente decíamos que al intervenir el parámetro “blend” teníamos que poner el valor de “visibility” a “both”, aquí con más razón.

También tenemos que poner el parámetro “transparent” del shader al valor “true”. Así mismo también tenemos que poner el parámetro “blendsource” al valor #alpha por la misma razón.

Como escribíamos antes tenemos que cambiar el valor de la propiedad renderformat para hacer que la calidad de la representación sea alta, ya que sino los 32 bits de profundidad con que hemos definido nuestro bitmap no servirían de nada. Por lo tanto, lo pondremos al valor #rgba8888 que permite representar unas texturas con una profundidad de 24 bits para el color y 8 bits para canales alpha, esto nos permite utilizar sin problemas texturas con degradados y con detalles muy pequeños. En contra tiene que el cálculo computacional de la representación es mayor.

Puede ser que necesitemos modificar el parámetro “blendfunction” del shader que está establecido a #multiply por defecto. Se puede modificar para buscar efectos especiales.

El script que sigue asignará la textura “ICCA” (Imagen Con Canal Alpha) al modelo “Box01” que se supone tiene un shader llamado “Default Material1” y que se asignará al botón correspondiente para que el usuario lo ejecute. Se supone que la textura ha sido introducida en el script de “Programación” mediante el método “newtexture”.

```
global mundo3D
global modelotocado

on mouseDown

  mundo3D.model("Box01").visibility=#both
  mundo3D.shader("Default Material1").texture=mundo3d.texture("ICCA")
  mundo3D.shader("Default Material1").transparent=true
  mundo3D.shader("Default Material1").texture.renderformat=#rgba8888
  mundo3D.shader("Default Material1").blendfunction=#multiply
  mundo3D.shader("Default Material1").blendsource=#alpha

end
```

Podemos añadir que una vez controlados las pequeñas dificultades que tiene el uso de las imágenes con canales alpha, son muy útiles y se utilizan mucho para ahorra trabajo de modelado y de representación.

Por ejemplo, si tenemos que realizar una valla de una casa de campo, podremos utilizar esta técnica para dibujar los alambres o maderas y dejar el resto con transparencia para que se pueda ver a través de ella como pasa en una valla real y no realizar cada alambre con pequeños cilindros que harían del modelado de la escena una tarea engorrosa.

5. Texturas con mapas de reflejos.

5.1. Introducción a los mapas de reflejos.

Los mapas de reflejos son imágenes que se utilizan para emplear sobre modelos con superficies que deben reflejar su entorno, como los metales, el cristal, etc. Se utilizan en sistemas 3D en tiempo real como videojuegos para minimizar los cálculos computaciones de la representación a diferencia de entornos 3D que no son en tiempo real como MAX que normalmente hacen los cálculos necesarios estudiando los reflejos de unos objetos sobre otros

para mayor realismo. Por lo tanto es una manera falsa pero efectiva de crear superficies reflejantes como espejos y otras.

Los mapas de reflejos se crean con anterioridad a la propia escena, por lo que no se calculan en tiempo de ejecución y por lo tanto no reflejan nada en realidad, pero por otra parte, no se tiene que realizar ningún cálculo añadido.

Todo esto es posible porque Shockwave 3D lo soporta, ya que el shader estándar tiene un método llamado “reflectionmap” al que se le puede asignar una textura y que se añadirá a todos los demás parámetros del shader que ya hemos estudiado.

En la imagen siguiente se puede observar el caso práctico de mapas de reflejos donde hemos tomado unos modelos que se adaptan muy bien a estos mapas porque son muy pulidos y con aristas. Los materiales que se recrean sin mapas de reflejos y que podemos elegir directamente son metálicos como oro, plata y bronce y además se pueden modificar mediante los parámetros color diffuse, shininess, etc.

Toda esta representación se puede ver modificada por el uso de mapas de reflejos cuyos botones se encuentran en la parte inferior para probar cómo se mezclan con el resto de parámetros del shader.

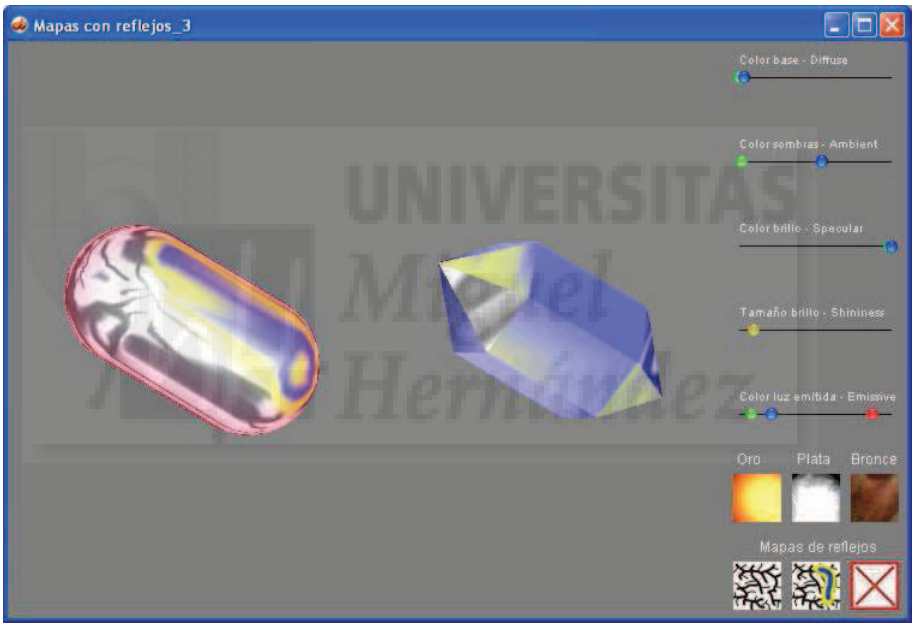


Imagen 5.12.14: Caso práctico de mapas de reflejos

En la imagen de abajo se puede apreciar la vista que ofrece el caso práctico mencionado antes de iniciar ninguna interacción con el usuario para que se observe la diferencia en la representación de los modelos y por lo tanto, el relativo éxito de la aplicación de estos mapas de reflejos.

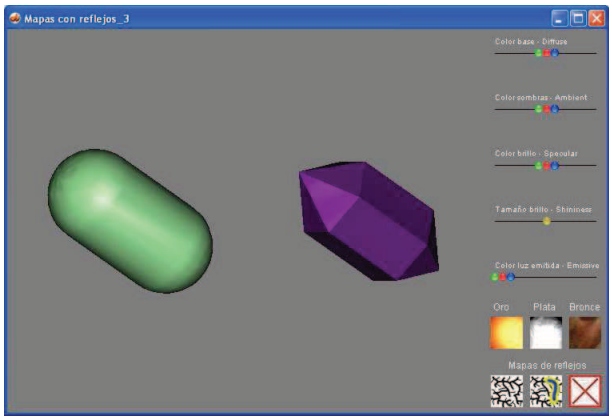


Imagen 5.12.15: Vista del caso práctico al iniciar su ejecución

5.2. Creación de los mapas de reflejos.

Hemos escrito anteriormente que los diseñadores de entornos virtuales en tiempo real como videojuegos, etc., crean los mapas de reflejos con anterioridad a la representación del mundo tridimensional.

Estos diseñadores utilizan dos estrategias distintas según sus necesidades para dos tipos distintos de mapas de reflejos: los reflejos realistas y los reflejos generales.

Los mapas de reflejos realistas se usan cuando tratan de reflejar un entorno en particular. Por ejemplo, si se trata de reflejar un paisaje montañoso determinado como el valle de Ordesa, entonces se utilizará una foto de ese paisaje en particular directamente, con lo que el efecto será muy conseguido.

Los mapas de reflejos generales se utilizan cuando no se puede saber con antelación el entorno que rodeará al modelo. Por ejemplo, si estamos creando una espada de acero que lleva el héroe de nuestro videojuego que pasa por distintos entornos como un bosque o un castillo, tendremos que crear un reflejo que funcione bien con el acero y que se aprecie como reflejante, pero no de un entorno en particular. En el caso práctico utilizamos este tipo de mapas de reflejo generales.

Hemos utilizado ®Adobe ®Photoshop para crear unos reflejos como los que se muestran en la imagen que sigue y que demuestra que no hace falta mucha pericia para crear mapas convincentes. Se puede ver el resultado experimentando en el caso práctico.

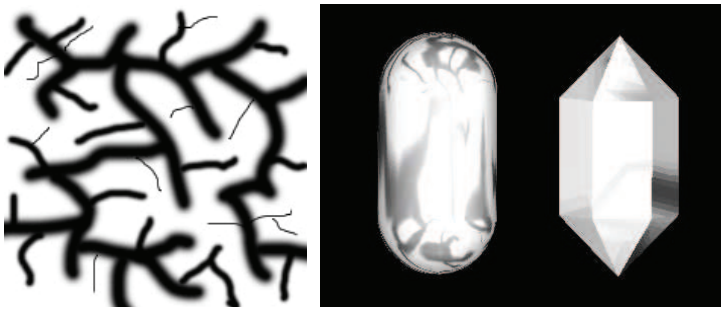


Imagen 5.12.16: Mapa de reflejo general y su resultado

Debemos tener cuidado con el uso de efectos difuminados que podemos conseguir en PhotoShop mediante filtros como el desenfoque gaussiano o por el uso de herramientas como “desenfocar” ya que podemos suponer que conseguiremos un efecto de pulido siendo el efecto conseguido el contrario, ya que al tener las fronteras entre dibujos del mapa bien marcadas, al texturizar el modelo se aprecian más definidas.

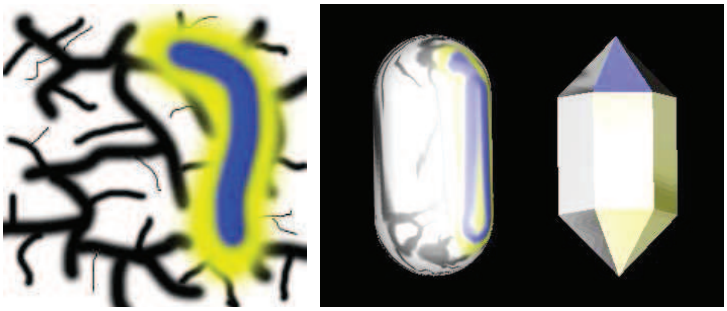


Imagen 5.12.17: Mapa de reflejo general adaptado y su resultado

Deberíamos incluir algunos colores de los que resalten alrededor de nuestro modelo para dar la sensación de que recrean el entorno. Por ejemplo, en el caso de la espada citado anteriormente podríamos usar un mapa de reflejo general pero con tonos en verde para cuando pasa por un bosque y en tonos marrones para cuando está en un castillo y luego cambiarlos cuando sea el caso. Por ejemplo, en la imagen de abajo se puede ver la modificación realizada en el mapa de reflejo anterior.

5.3. Asignar los mapas de reflejos a los modelos 3D.

Por una parte tenemos que incorporar a la escena 3D los mapas de reflejos como cualquier otra textura mediante una sentencia como esta:

```
mundo3D.newtexture("mr_1",#fromcastmember,member("mapa_reflejo_1"))
```

Entonces esta textura la tenemos que asignar a la propiedad reflectionmap del shader con una sentencia como la que sigue:

```
mundo3D.shader(shaderseleccionado).reflectionmap=mundo3d.texture("mr_1")
```

Para concluir podemos escribir que los mapas de reflejos son un gran efecto para conseguir realismo y por otra parte son fáciles de crear y utilizar, ya que el código asociado a su uso es muy sencillo.

6. Texturas animadas por transformación.

En este punto vamos a estudiar cómo crear efectos especiales sobre los modelos creando texturas animadas. Para hacer este trabajo tenemos dos posibilidades:

- Usar una textura y transformarla en posición, escala y/o rotación además de cambiar el modo en que envuelve al modelo 3D. Este método será el que estudiemos en el punto actual. Con estas técnicas podremos crear efectos muy realistas en superficies con mapas de reflejos.

- Usar muchas texturas e intercambiar unas por otras creando la sensación de que se visualiza una animación o un vídeo sobre la superficie del modelo 3D. Este método lo estudiaremos en el punto siguiente. Con esta técnica podemos por ejemplo hacer un reloj que funcione cambiando la textura que muestra los números que correspondan en cada momento.

6.1. Introducción a la aplicación de texturas con texturetransform.

En la imagen inferior se puede observar la vista que presenta el caso práctico que ilustra este punto. En su inferface se puede apreciar las diversas posibilidades que tiene una textura de ser modificada en tiempo de ejecución en cuanto a su aplicación al modelo 3D.

Como hemos comentando, la posibilidad de animación por transformación supone realizar cambios en su posición, en su tamaño y en su rotación. Estas posibilidades se implementan en el caso práctico con botones deslizantes de la parte superior derecha. Otro cambio que podemos realizar es cambiar el modo en que la textura se aplica al modelo 3D. En el caso práctico, estas posibilidades se ilustran en los botones de la parte inferior derecha.

Antes de comenzar a estudiar la manera de transformar las texturas, debemos saber que el shader puede aplicar estas texturas de dos modos básicamente: en modo mosaico o en modo patrón.

El modo patrón supone que la textura nunca se repite, por lo que si no es tan grande como para "cubrir" todo el modelo 3D, simplemente no lo hará. Este modo no nos conviene para hacer ninguna de las transformaciones que vamos a estudiar por lo que no lo tendremos en cuenta.

En modo mosaico el shader se ocupa de que la textura siempre cubra todo el modelo 3D y por tanto si no es tan grande como para cubrirlo, simplemente lo repite. Este método es el que necesitamos para nuestros trabajos y como es el modo por defecto con el que trabaja Shockwave 3D, no lo cambiaremos.

Todos los cambios que vamos a efectuar, parten de un método del shader llamado "texturetransform" que permite definir cambios sobre la forma en que la textura se aplica sobre el modelo 3D mediante sentencias Lingo. Existen una coordenadas llamadas UV para cada modelo 3D que definen cómo se aplican las texturas sobre él. Es como definir un punto de origen y unos ejes para la textura. "Texturetransform" lo que hace es variar esas coordenadas.

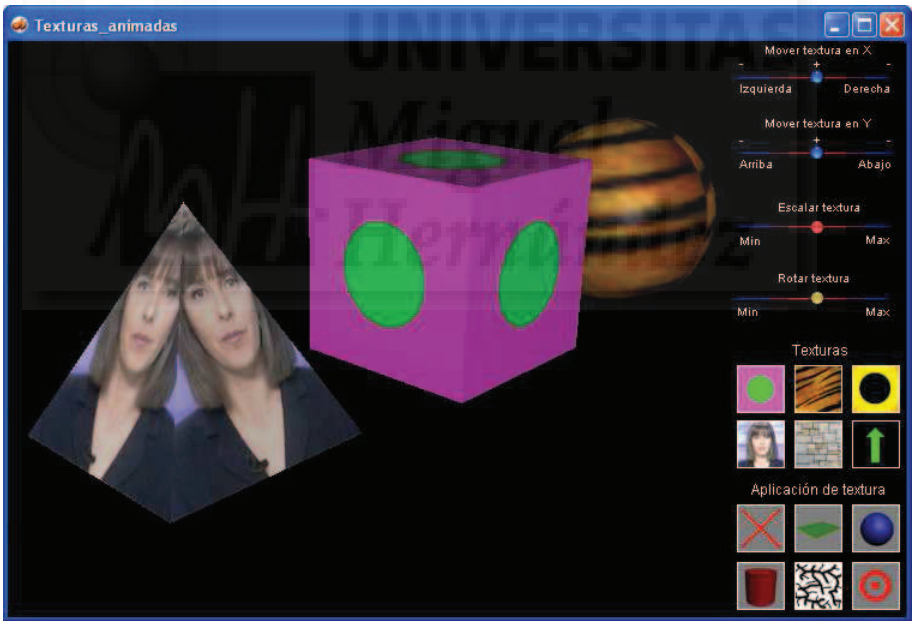


Imagen 5.12.18: Vista del caso práctico "Texturas animadas por transformación"

Esta manera de realizar la transformación es muy eficiente, ya que la textura se carga en memoria principal y por tanto solo tiene que disponerse de otra manera sobre el objeto, por lo que los cambios pueden llegar a ser rapidísimos y los efectos muy espectaculares sin gravar el rendimiento del proyecto en tiempo real.

Existe una función llamada Identity() de texturetransform y que permite volver a sus parámetros iniciales y por tanto a reiniciar la textura. Identity( ) no necesita parámetros, por lo que si queremos volver a ver un modelo con la textura sin transformar escribiremos:

```
mundo3D.shader("Default Material0").texturetransform.identity()
```

6.2. Transformación de escala de texturas.

La transformación en escala de las texturas presupone que ya no se corresponde con el modelo: se quedará corta o demasiado grande.

Tenemos que tener en cuenta que si la aplicación de estos cambios es interactiva, es decir, dependen de la elección del usuario, no podemos permitir que elija una escala de 0, ya que sería no visible, por lo que tenemos que filtrar la entrada y si el usuario elije el valor nulo, lo interpretaremos como una escala muy pequeña pero no nula. En el caso práctico se soluciona este tema simplemente usando una sentencia condicional como:

```
if (nueva_escala=0) then nueva_escala = .001
```

La función que permite escalar la textura se llama “scale” y requiere tres parámetros aunque solo utiliza dos, ya que una textura es un elemento en 2D y por tanto el último valor siempre valdrá 1. Un ejemplo de su utilización es la siguiente sentencia:

```
mundo3D.shader("Default Material0").texturetransform.scale(nueva_escala,nueva_escala,1)
```

Para que la transformación de la escala no cambie sobre otra transformación en escala anterior, se debe utilizar la función identity() explicada anteriormente para que el usuario perciba que la escala elegida es la que se aplica.

Esto hace que la aplicación del cambio de escala tenga que seguir un algoritmo que se programará finalmente así:

```
mundo3D.shader("Default Material0").texturetransform.identity()

if (nueva_escala=0) then nueva_escala = .001

mundo3D.shader("Default Material0").texturetransform.scale(nueva_escala,nueva_escala,1)
```

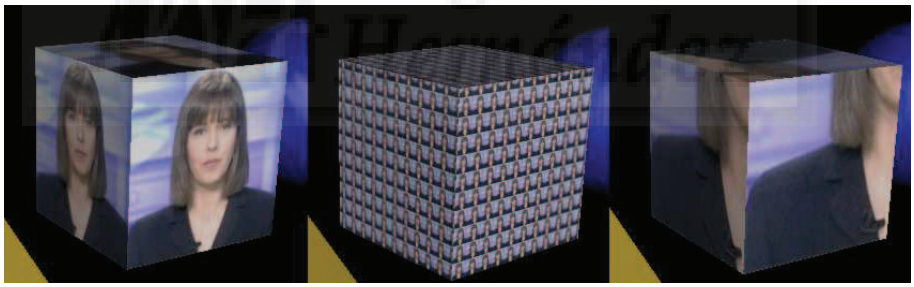


Imagen 5.12.19: El mismo modelo con una textura a distintas escalas

6.3. Transformación de giro de texturas.

La transformación de rotación de texturas supone que estas se apliquen giradas con respecto a su posición original. Esta transformación se puede usar añadida a las demás de escala y posición y por lo tanto sumar los efectos.

La transformación de rotación tiene algunas diferencias con respecto a la de escala. Por ejemplo, la rotación no hace falta inicializarla ya que puede ser en sentido negativo o positivo, pero no es acumulativa sino cíclica, esto hace que no tengamos que usar la función identity() cada vez que la transformemos en rotación.

La función de texturetransform que permite la rotación se llama “rotate” y solo requiere un parámetro, el ángulo. Como pasaba anteriormente, por motivos de isomorfismo de las funciones (un aspecto de los lenguajes de programación) se emplea un vector como parámetro

cuando en realidad solo necesita un vector, por lo que solo utiliza el último valor, teniendo que estar los otros dos a 0.

La sentencia que permite rotar una textura cierto ángulo se escribirá como:

```
mundo3D.shader("Default Material0").texturetransform.rotate(0, 0, angulo)
```

Imagen 5.12.20: El mismo modelo con una textura sin rotar y rotada

6.4. Transformación de posición de texturas.

La transformación de posición de texturas se lleva a cabo con la función “translate” de texturetransform. La función translate utiliza dos valores para cuantificar la nueva posición en el eje X y en el eje Y a la que se tiene que mover la textura. Los valores pueden ser positivos o negativos, interpretándose el signo como traslación en sentido positivo o negativo.

La sentencia que permite trasladar una textura se codificará así:

```
mundo3D.shader("Default Material1").texturetransform.translate(traslatey,traslatex,0)
```

En el caso práctico asociado a este punto, hemos implementado esta función dentro de un bucle que se ejecuta x veces por segundo, por lo que el usuario puede interactivamente elegir cuanto y cuan rápido se traslada la textura por la superficie del modelo 3D.

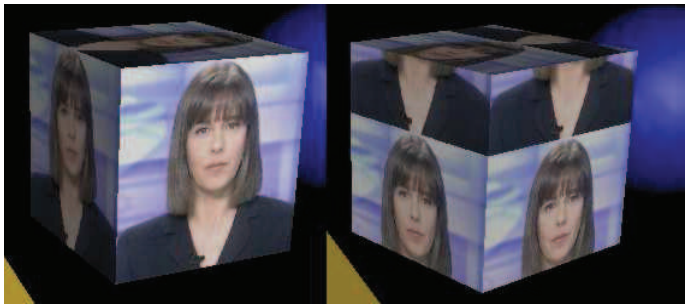


Imagen 5.12.21: El mismo modelo con una textura sin trasladar y animada

6.5. Modos de aplicación de las texturas a los modelos.

En este punto vamos a ver cómo cambiar en tiempo de ejecución la forma en que la textura es aplicada al modelo 3D en cuanto a como lo “envuelve”. Como hemos escrito antes, la textura rodea la superficie del objeto. Esto se hace aplicando una parte de la textura a la geometría del modelo. Pero esta aplicación se puede realizar de muchas maneras, que normalmente dependen de la geometría del modelo.

Por ejemplo, si un modelo es esférico podemos aplicar una sola textura y dividir en trocitos la textura para aplicárselo a cada uno de las caras que componen su superficie de forma

equitativa. Si el modelo es cúbico, se puede aplicar la textura entera sobre cada una de las seis caras que forman el cubo.

El modo en que la textura “envuelve” al modelo 3D se puede controlar en tiempo de ejecución mediante la propiedad “texturemode” del shader. Esta propiedad toma un valor de entre un conjunto dónde las opciones más usuales son cinco: ninguno, plano, esférico, cilíndrico y reflejos. Vamos a estudiar cada uno de estos posibles valores y su incidencia en la texturización de los objetos.

Tenemos que añadir, que todo modelo tiene el modo por defecto al valor ninguno y esto quiere decir que la textura se aplicará según su geometría y que viene definida desde el mismo momento en que se crea el modelo, en nuestro caso en 3D Studio MAX. Es decir, por defecto, la textura se aplica de la manera más “envolvente” para el objeto.

◆ Modo ninguno, none: este modo es el valor por defecto para “texturemode” y para todos los modelos 3D. La sentencia para aplicar este modo será como sigue:

```
mundo3D.shader(“Default Material 1”).texturemode = #none
```

Imagen 5.12.22: El modelo cubo con el modo de texturemode a none

◆ Modo plano, wrapplanar: funciona bien para objetos tipo caja pero muy planas.

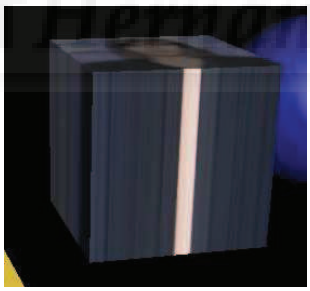


Imagen 5.12.23: El modelo cubo con el modo de texturemode a planar

Por ejemplo, si queremos modelizar una pantalla de plasma podemos usar una caja muy plana y utiliza este valor para texturemode. En la imagen inferior no se aprecia un buen efecto aunque creemos que si su utilidad. El valor para este modo se escribirá #wrapplanar.

◆ Modo esférico, wrapspherical: el modo esférico envuelve todo el modelo 3D con la textura como si solo tuviera una gran cara, por lo que no funciona bien en el caso del cubo pero queda perfecto para planetas, pelotas, ... El valor para este modo se escribirá #wrapspherical.

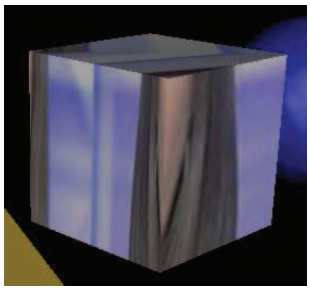


Imagen 5.12.24: El modelo cubo con el modo de texturemode a esférico

◆ Modo cilíndrico, wrapcylindrical: en este modo, la textura envuelve las caras trasversales totalmente y se repite en la cara superior y en inferior. El valor para este modo se escribirá #wrapcylindrical.

Imagen 5.12.25: El modelo cubo con el modo de texturemode a cilíndrico

◆ Modo reflejo, reflection: en este modo, la textura se aplica de la misma forma que en el modo esférico pero si el objeto se mueve la textura no lo acompaña en el movimiento por lo que produce un efecto de reflejo del entorno. El valor para este modo se escribe #reflection.

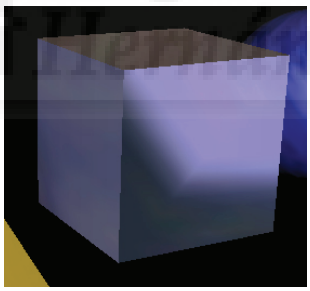


Imagen 5.12.26: El modelo cubo con el modo de textura reflection

Podemos escribir que la animación mediante cambios en las texturas aplicadas a los modelos es espectacular, aunque requiere de bastante código para obtener buenos resultados, sobre todo si queremos que las animaciones sean interactivas.

7. Texturas animadas por intercambio.

Las texturas animadas por intercambio suponen cambiar la textura del modelo 3D en tiempo de ejecución. Si el intercambio se hace a cierta velocidad, dará la sensación de que se visualiza una animación del tipo fotograma a fotograma o lo que es lo mismo, un vídeo, sobre la superficie tridimensional de un objeto. En este punto estudiaremos métodos para realizar este efecto especial así como para que lo gobierne el usuario interactivamente.

7.1. Introducción a las texturas animadas por intercambio.

En la imagen inferior se puede observar una vista del caso práctico que hemos realizado para ilustrar este punto. Se pueden observar que las texturas animadas que vamos a aplicar pueden contener canales alfa como en el caso de las mariposas.

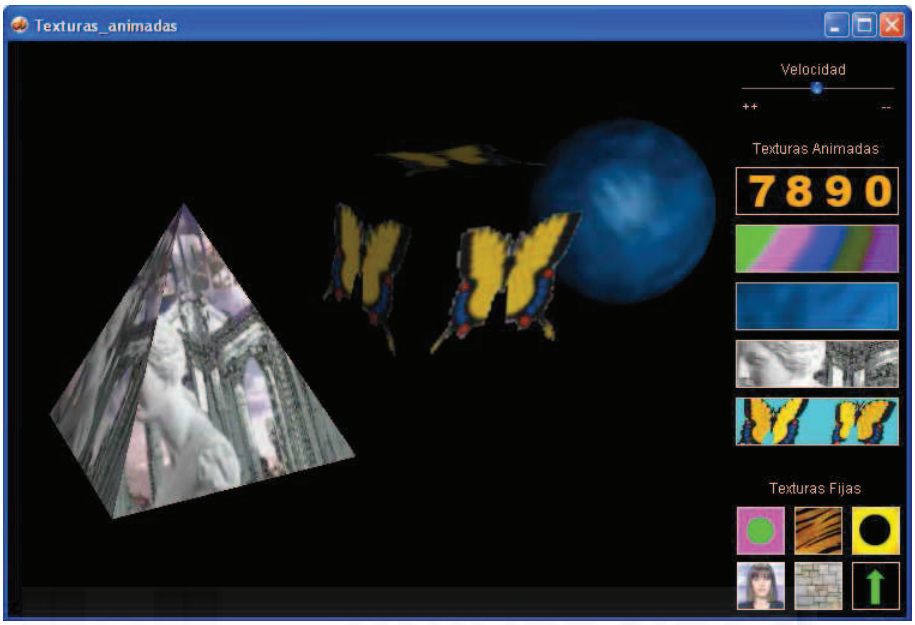


Imagen 5.12.27: Vista del caso práctico "Texturas animadas por intercambio"

En el caso práctico, a la hora de implementar esta idea de animar texturas intercambiándolas, es donde se pone de manifiesto el problema que se nos presenta. El problema es dónde escribir el código para cambiar las texturas, o sea, en qué script y cómo crear la interactividad para que el usuario lo ejecute.

El problema de la interacción con el usuario es peliagudo ya que este solo deberá elegir el modelo 3D y un botón que representa la textura animada (en realidad un conjunto de texturas) y el programa se tiene que encargar del resto.

Para resolver este problema tenemos que tener claro que la textura animada será un conjunto de imágenes incorporadas como castmembers a Director y que se agrupan creando una secuencia como los fotogramas en una película.

Para gestionar convenientemente estas series de imágenes castmembers debemos de crear nuestros propios "Casts".

7.2. Creación de Casts para contener las imágenes de la animación.

Los Casts permiten agrupar elementos que luego se utilizarán en una película de Director. Esto nos viene muy bien para crear un cast para cada secuencia de imágenes que serán los fotogramas de las texturas animadas.

Además tienen la ventaja de poder referirnos a los elementos que incluyen de forma muy eficiente y también permiten que los nombres de distintos castmembers se puedan repetir siempre y cuando se encuentren en casts distintos. Por ejemplo, podemos tener el miembro "fotograma_1" en el cast "horas" y otro que se llame igual en el cast "video".

Por lo tanto, tenemos el cast por defecto llamado "Internal" que contiene todos los elementos propios del interfaz de la película y otros cast creados por nosotros cada uno de los cuales se corresponderá con una secuencia de textura animada.

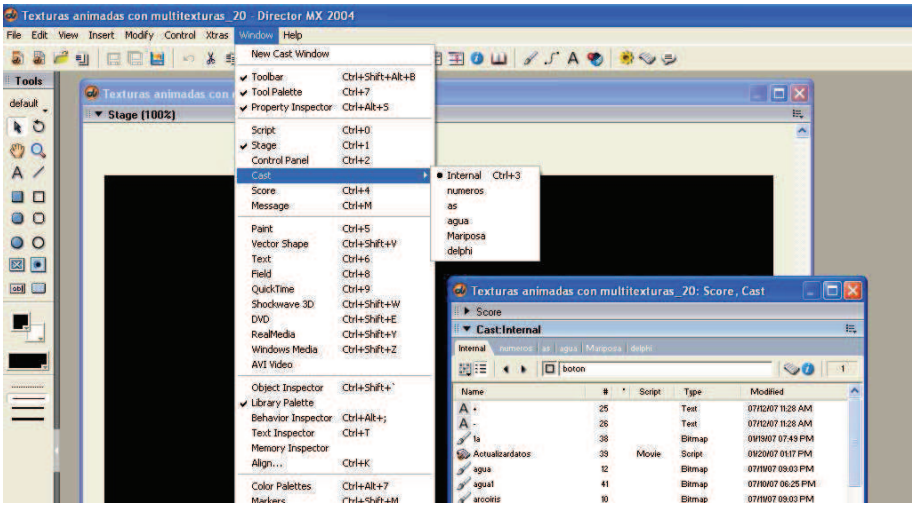


Imagen 5.12.28: Menú para crear nuevos Casts

Como se puede observar en la imagen precedente para crear un Cast nuevo en Director ejecutamos: menú Windows > grupo Cast (o pulsar “Ctrl+3”). Aparecerán los cast creados y elegiremos “Internal”. Aparecerá la ventana del cast “Internal”.

Hacemos clic sobre el botón mas a la izquierda llamado “Choose Cast” y aparece un menú donde pulsamos sobre “New Cast”. Aparecerá la ventana de creación de un nuevo cast.

En esta ventana debemos de poner un nombre a nuestro nuevo Cast y elegir el tipo de almacenamiento “Internal” para que los castmembers se almacenen en el interior de la película de Director y evitar que después tengamos problemas cuando publiquemos nuestros trabajos. Cuando pulsemos sobre “Create” tendremos un nuevo Cast vacío.

7.3. Animación de texturas por programación.

La descripción de cómo animar texturas de intercambio se puede realizar asÍ: la textura aplicada al modelo 3D debe de se intercambiada por la siguiente cuando pase una cantidad determinada de tiempo. Si la textura en cuestión es la última, debe de cambiarse por la primera. De esta forma crearíamos un bucle sin fin. La textura animada se visualizará mientras el espectador no elija lo contrario. Además, el usuario puede elegir la velocidad de la animación.

Expresado de manera más esquemática, debemos resolver los siguientes puntos:

- Crear un bucle para visualizar una imagen tras otras consecutivamente.
- Hacer que el bucle sea infinito hasta que el usuario elija lo contrario.
- Controlar la velocidad de ejecución del intercambio de texturas interactivamente.
- Poder referirnos los miembros de un cast por su posición secuencial en el cast.

Para resolver el primer punto, es decir, para realizar un bucle sin fin, la mejor opción es utilizar el disparador exitframe del script llamado “Bucle” ya que es un mecanismo como el que buscamos a falta de calibrar la velocidad de intercambio.

Para resolver el segundo punto, antes vamos a estudiar el cuarto y último, ya que es aquí dónde podemos encontrar la solución. Director aporta una forma de poder referirnos a los miembros de un Cast por su posición en él. Esta forma es el método “member of castlib” que podemos estudiar en este código:

```
mundo3d.texture("textura").member=member num of castlib "textura_animada_1"
```

Siendo “textura_animada_1” el nombre del cast y num el número de lugar que ocupa un member dentro de ese cast en particular.

Por lo tanto, para hacer que el bucle sea infinito y no termine cuando se acaban las imágenes en un cast, podemos usar un viejo truco de programación que asegura que si estamos en el último fotograma, empiece con el primero y es como sigue:

```
num = (contador mod numero_de_miembros_del_cast) + 1
```

Este código hace que el número siempre esté dentro de los límites del número de los componentes de un cast. El operador mod devuelve el resto de la división entera. Si suponemos que una determinada textura tiene 13 componentes cuando estemos en el componente 13, este código determinará 13 mod 13 = 0 +1 dará 1, con lo que volverá a comenzar. Este código lo situaremos como filtro de tal manera que, si no da 0, el contador simplemente se incrementará en una unidad y por ejemplo, del fotograma 7, pasará al 8.

Para resolver el punto tres dónde tenemos que controlar la velocidad de ejecución de la animación, podemos utilizar un botón deslizante. El problema es que si el usuario elije un intervalo de transición que signifique 0 para verlo muy rápidamente, es evidente que el software no lo puede admitir y de alguna manera lo tiene que filtrar. Recuérdese que pasaba lo mismo en el punto anterior con el intercambio de texturas por transformación.

Por lo tanto, antes de efectuar el intercambio en sí mismo, utilizaremos una condición parecida a la siguiente para este límite:

```
if (velocidad mod veloc =0) then …
```

Donde veloc sera el valor devuelto por el botón deslizante. Por lo tanto el script “Bucle” quedará algo así:

```
global mundo3d
global contador
global velocidad
global veloc
global num
global shadertocado
global boton

On exitFrame me

velocidad= velocidad +1

  if (boton=1) and (velocidad mod veloc =0) then          // para cada botón de textura animada
    contador=contador+1
    num=(contador mod 9)+1
    queshader()
    mundo3d.texture("nueva").member=member num of castlib "numeros"
    mundo3d.shader(shadertocado).texture= mundo3d.texture("nueva")
  end if

  if (boton=2) and (velocidad mod veloc =0)then
…

end
```

Suponiendo que esta textura animada tiene 9 fotogramas y se llama “números” y que se tiene que ejecutar al pulsar un botón que pone la variable global “boton” al valor 1.

Como conclusión a este tema diremos que los shader son unos recursos que debemos conocer bien para lograr realismo en nuestros trabajos 3D y hace falta mucha pericia para lograr resultado realistas de calidad dado que es un sistema en tiempo real y esto condiciona mucho las posibilidades. Las superficies de los objetos representados serán finalmente la combinación de las propiedades definidas en el shader y de las propiedades de las luces de la escena 3D.

En lo que respecta a los cambios en la textura, según nuestra opinión cuanto más vistosos queremos que sean los efectos, más complicada se convierta la programación necesaria para llevarlos a cabo. De todas formas, creemos que es un tema abordable. Tenemos que decir que toda la teoría aportada en este tema es puesta en práctica en los casos adjuntos. De hecho, todas las imágenes que se han integrado en este documento son capturas de pantalla de la ejecución de estos casos prácticos.

Teniendo en cuenta que queremos que los casos prácticos puedan ser entendidos como una unidad autónoma, puede darse el caso de duplicar conceptos y textos aportados en este tema, por lo que a veces resulta repetitivo.

Caso práctico 12.1: Asignación de texturas.

Objetivo: Comprender a trabajar con las texturas y asignarlas a objetos 3D utilizando Lingo 3D.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Crear una escena de cuatro modelos con movimiento de rotación.
2. Exportar el fichero a formato w3d incluidos los datos de la animación.
3. Incorporar el Shockwave 3D y las texturas al mundo 3D.
4. Crear los botones para elegir las texturas y programar su aplicación a los modelos.

1. Crear una escena de cuatro modelos con movimiento de rotación.

Para este caso práctico queremos crear una escena donde tendremos varios objetos y texturas de manera que pulsando sobre uno de ellos en concreto, le asignaremos la textura deseada de forma interactiva. Con el fin de apreciar mejor cómo se disponen las texturas sobre el modelo, haremos que estos tengan formas distintas, en concreto, una caja, una pirámide, una esfera y un “toro nudo”. Con este mismo propósito, haremos que se sitúen a distinta distancia de la cámara y rotando continuamente sobre su eje X.

Como siempre utilizaremos 3d Studio MAX para realizar esta parte del trabajo.

1.1. Construcción de los modelos. Este punto es bastante sencillo, ya que los cuatro modelos, aunque de geometría diferente, son formas primitivas. Por lo tanto se trata de crearlos, modificar sus tamaños para que sean comparables y por último, posicionarlos.

1.1.1. Crear la pirámide: panel Crear > Geometría > Primitivas estándar > Pirámide.

1.1.2. En la vista superior, hacemos clic y arrastramos para definir la base haciendo clic, luego al movernos estamos definiendo la altura que fijamos haciendo clic de nuevo.

1.1.3. Ir al panel Modificar > Parámetros. Modificar los valores para que quede una pirámide proporcionada.

1.1.4. Modificar el nombre y el color.

1.1.5. Repetir los pasos con los otros tres modelos. El toro nudo, se encuentra en la categoría de “primitivas extendidas”. Conviene recordar los nombres que les damos a los modelos, y asignar un color distinto a cada uno de ellos.

El resultado se debe parecer a la figura 5.12.cp.1.1.

1.2. Animación de la pirámide. Este trabajo lo haremos con fotogramas claves. Para ello crearemos 4 fotogramas claves en los frames 25, 50, 75 y 100, ya que la animación es de 100 fotogramas. En cada una de estos fotogramas giraremos 90º en el eje X para completar una vuelta entera al terminar la animación, que se repetirá ininterrumpidamente. El proceso detallado seguirá los siguientes pasos:

1.2.1. Pulsar el botón de “AutoKey” para que los cambios que efectuemos se tomen como una animación y se interpolen automáticamente los movimientos necesarios para pasar de un fotograma clave a otro.

1.2.2. Posicionar el botón de fotogramas en la posición 25.

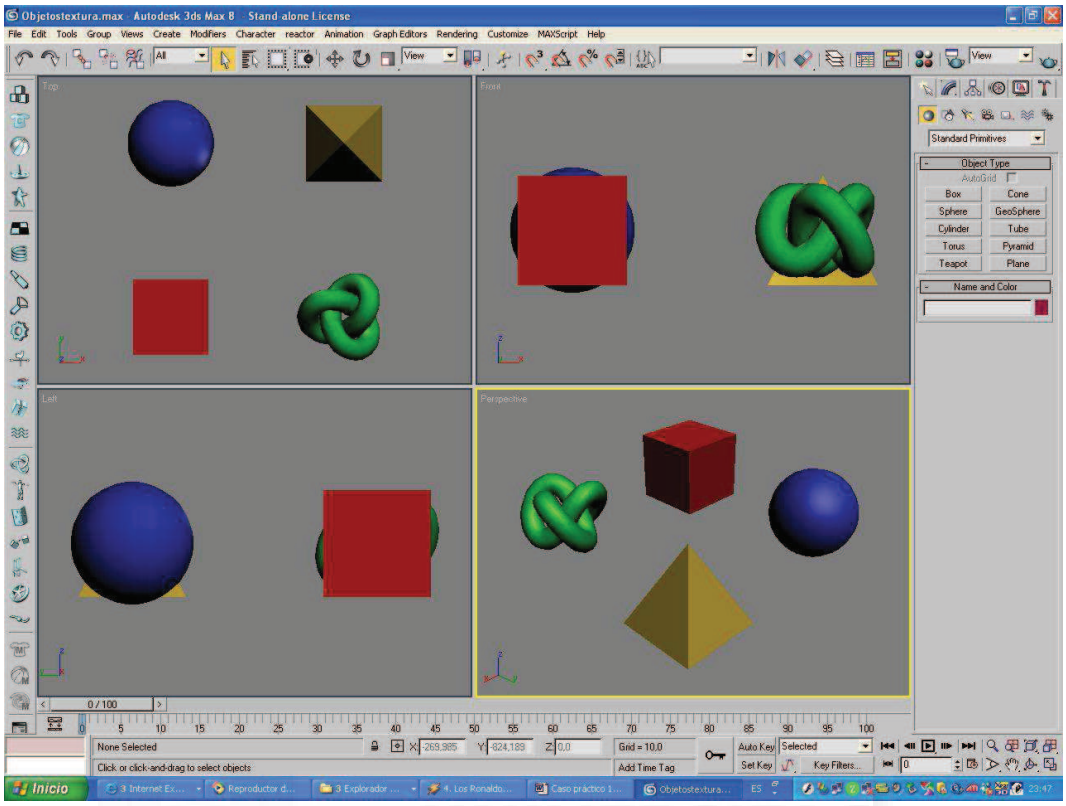


Imagen 5.12.cp.1.1: Pantalla de MAX para el modelado del mundo 3D

1.2.3. En la vista superior, seleccionar la herramienta girar. Sobre el círculo amarillo arrastrar con el ratón hacia abajo levemente hasta que marque 90º. Si no se consigue realizar de forma interactiva por alguna razón, se puede establecer numéricamente en el control pertinente. Se observará que aparece un rectángulo verde sobre el fotograma 25 que indica que se ha convertido en fotograma clave.

1.2.4. Repetir el proceso con los fotogramas que faltan. En la imagen que sigue se puede observar cómo quedaría la animación de la pirámide terminada.

En realidad, esta animación no estaría perfecta. Para comprobar esto basta con pulsar sobre el botón de “play” para ver que existe un pequeño salto cuando se pasa del fotograma 100 al 1. Esto se debe a que habría que haber calculado exactamente el porcentaje de giro que representa cada uno de los fotogramas, es decir, 360º / 100. Para resolver este pequeño problema haremos lo que sigue.

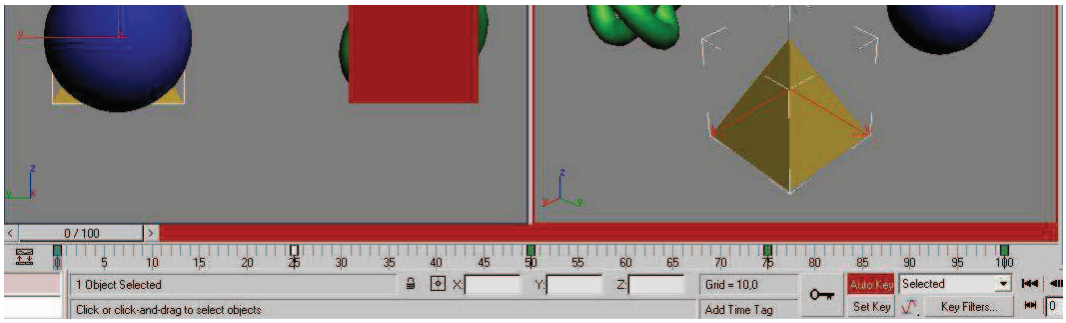


Imagen 5.12.cp.1.2: Captura de pantalla de la animación con Auto Key

1.3. Control en los límites de la animación. Para solucionar este problema que tenemos en los límites de la animación, donde se detecta un pequeño salto, usaremos el “Track View – Curve Editor”.

1.3.1. Pulsar en la barra de herramientas sobre el icono que se muestra abajo.

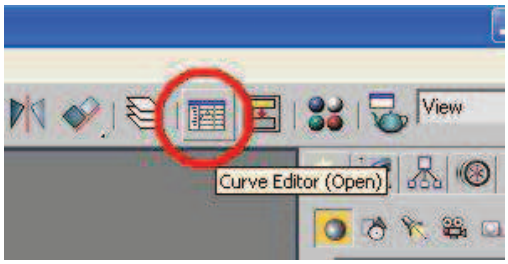


Imagen 5.12.cp.1.3: Icono del editor de curvas

1.3.2. Aparecerá una pantalla parecida a la que muestra la imagen 5.12.cp.1.4. Desactivar las pistas X e Y. Comprobaremos que la líneas correspondientes a los ejes X e Y desaparecen, sólo quedará la línea de color azul, que contendrá unos cuadrados pequeños y grises que se corresponden con las claves de animación.

Estas líneas denotan la animación en forma de curvas. Estas curvas tienen puntos de control por cada fotograma clave definido. Los puntos de la curva son de tipo Bézier y los podemos modificar.

1.3.3. Modificar los puntos para que la línea quede lo más recta posible. Esto hay que hacerlo con especial cuidado en los puntos inicial y final

1.3.4. Cerrar esta ventana y volver a ejecutar la animación para comprobar que ahora ya no distinguimos ningún salto entre el fotograma 100 y el 1 cuando vuelve a empezar.

1.4. Copiar la animación a los demás modelos. Vamos a repetir las animaciones muy rápidamente.

1.4.1. Volvemos a abrir la ventana Track View – Curve Edit.

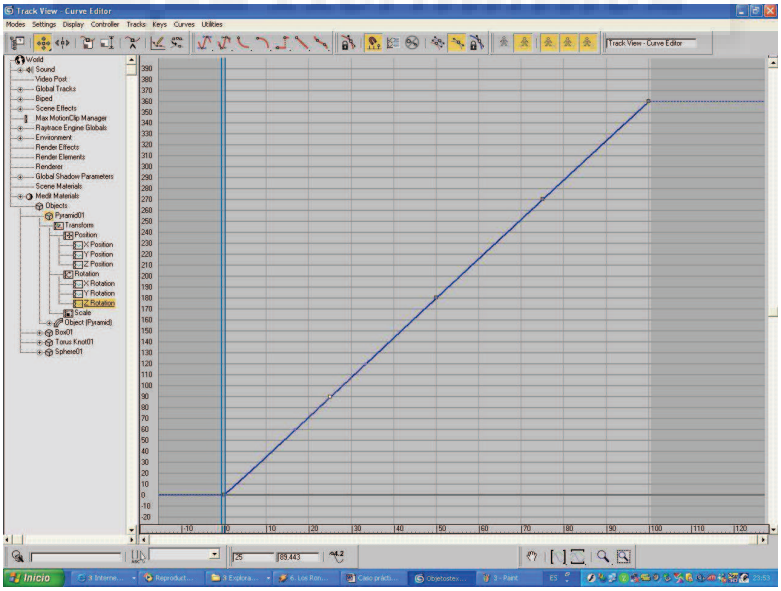


Imagen 5.12.cp.1.4: Curva asignada a la rotación en el eje Z

1.4.2. Seleccionamos la curva azul que muestra la rotación que nos interesa.

1.4.3. La copiamos con Ctrl + C.

1.4.4. Cerramos la ventana Track View.

1.4.5. Seleccionamos uno de los otros objetos, como por ejemplo la caja y abrimos directamente el Track view – Curve edit. Y seleccionamos el Z rotation.

1.4.6. Pegamos la curva con Ctrl + V.

1.4.7. Comprobar que la animación se ha copiado de un objeto a otro. Repetir la operación con los otros dos objetos; el toro y la esfera.

Este proceso es importante para tener la seguridad de que todos los objetos comparten exactamente la misma animación.

2. Exportar el fichero a formato w3d incluidos los datos de la animación.

2.1. En Max, ejecutamos el comando Archivo > Exportar.

2.2. En el tipo de archivo elegimos Shockwave 3D scene.

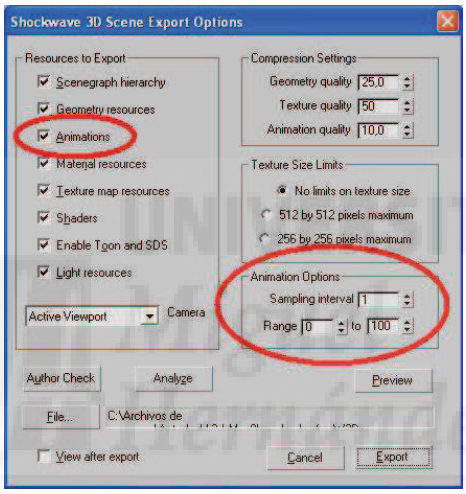


Imagen 5.12.cp.1.5: Elementos de animación en la ventana de exportación

2.3. En la imagen de arriba se puede observar los dos aspectos a tener en cuenta. El más importante es tener las animaciones marcadas como recurso a exportar.

2.4. También tenemos que tener cuidado en las “Animation Options” de tener un fotograma de la animación en Shockwave 3D por cada uno de los que tenemos en MAX, y de que exportamos el rango de la animación con la opción de “Range”.

3. Incorporar el Shockwave 3D y las texturas al mundo 3D.

3.1. Este paso supone importar dentro del fichero de director tanto la escena 3D generada anteriormente como una serie de imágenes. Estas imágenes serán del tipo .jpg. Por cada textura que queremos trabajar, importaremos dos ficheros: la propia textura y una imagen que hará las veces de botón para seleccionarla.

Las texturas las he nombrado según su contenido (cielo, madera, etc.) y para que ocupe menos memoria tienen una extensión de 200 x 200 píxeles. Los botones se llaman “botónX”, siendo X el nombre de la textura. Tienen un tamaño de 50 x 50 píxeles. Por ejemplo, la imagen para poner la textura “cielo” se llamará “botónCielo”.

3.2. Ejecutamos Archivo > Importar. Luego seleccionamos el fichero Shockwave y pulsamos el botón “Añadir”. También podemos seguir los mismos pasos para añadir las imágenes de las texturas y de los botones. Todos estos elementos aparecerán en la ventana de Cast Member.

En la imagen de abajo se puede apreciar la escena 3D y los botones de las texturas en su parte inferior.

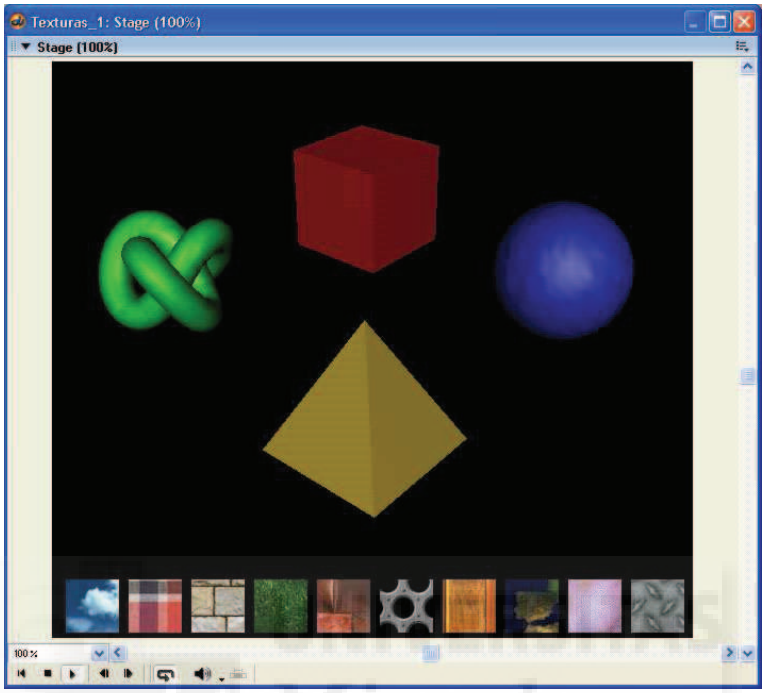


Imagen 5.12.cp.1.6: Ventana de la aplicación ejemplo

3.3. En la imagen 5.12.cp.1.7, se puede observar que seguimos utilizando la estructura de 2 fotogramas de la película en Director para realizar todo el trabajo. Por lo tanto, tendremos un solo miembro que será el mundo 3D, en este caso, llamado “objetos” y tres comportamientos: “Inicialización” asignado al fotograma 1, “Bucle” asignado al fotograma 10 y “Programación” asignado al castmember “objetos”.

Arrastrando los miembros botones, incorporamos estos elementos a la película. Debemos hacer que tengan una duración de un sólo fotograma al igual que el mundo 3D. Este fotograma es al que le hemos puesto el nombre de “Principal”.

3.4. Posteriormente lo que haremos es colocar estos botones fuera del área que ocupa el “mundo3d”. Para tener hueco suficiente, tenemos que modificar el tamaño del “stage” y hacerlo un poco más alto. Ir a Property Inspector > Movie > Stage Size para esta tarea.

3.5. Para incorporar las texturas al interior del mundo 3D y poderlas utilizar, debemos de conocer “newtexture”, que es una propiedad de los objetos Shockwave 3D. Por ejemplo podemos escribir:

```
mundo3D.newtexture("aa",#fromcastmember,member("marmol"))
```

Esta sentencia hace que un castmenber de tipo imagen llamado “mármol” se incorpore como una textura dentro del mundo 3D y se conocerá como “aa”. Es evidente que tendremos que incluir una sentencia de este tipo por cada una de las texturas que queramos incorporar. El problema es dónde colocar estas sentencias ya que solamente las podemos escribir una vez. Si las escribimos más de una vez a lo largo de la ejecución del programa, aparecerá un error diciendo que la textura que pretendemos incluir ya pertenece a la escena 3D.

Una posibilidad es incluirlas en el behavior “Programación”.

Este behavior es una buena solución, ya que además este comportamiento incluirá un modo para seleccionar uno de los cuatro objetos al que podemos asignarle una de las diez texturas y por lo tanto será el lugar donde se concentre toda la programación novedosa de esta práctica. La asignación de la textura se realizará, finalmente, en un comportamiento escrito para cada botón textura.

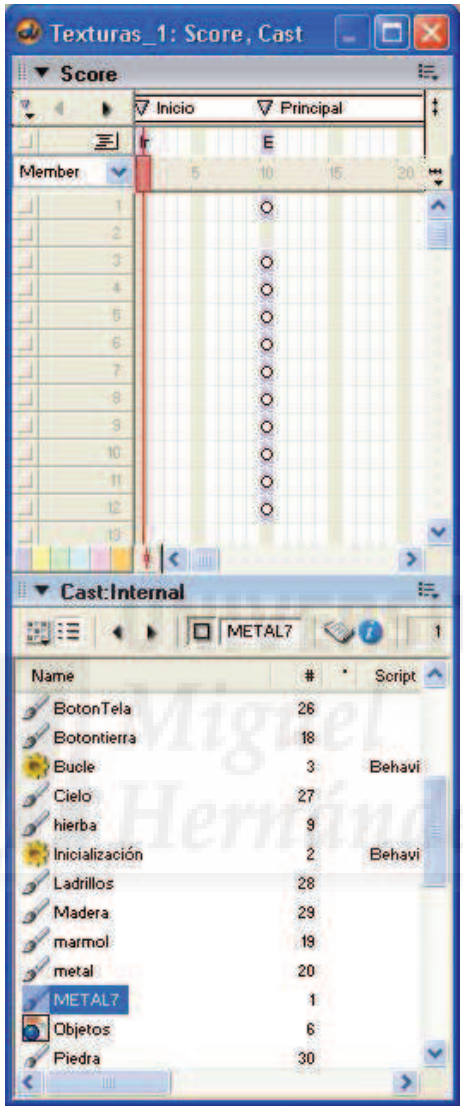


Imagen 5.12.cp.1.7: Ventanas Cast y Score de Director

El diseño del interface, esto es, la manera en que el usuario elige una textura para un modelo, podría realizarse de múltiples formas. La que hemos elegido, hará que esta interacción sea muy intuitiva y dinámica ya que separaremos la elección del objeto a trabajar de la elección de la textura a asignar. Por lo tanto, no tendremos cada vez, que pulsar un objeto y una textura, ya que el objeto quedará como el elegido para asignar una textura tras otra. Se podría discutir si deberíamos hacer algo que denotase qué modelo 3D es el "elegido" en un momento dado, es decir, podríamos "marcarlo" de alguna forma, pero ello modificaría la forma en que lo observamos, y por lo tanto incidiría en la textura asignada, por lo que hemos llegado a la conclusión de no evidenciar su elección de forma alguna. El comportamiento "Programación" quedará como sigue:

```
global mundo3D                                    -- variables globales
global modelotocado
```

```
property psprite                                        -- variables locales
property tocado

on beginSprite me                                       -- inicialización de la presente aparición (sprite)
  pSprite = sprite(me.spriteNum)                        -- lo que sigue añade las texturas al mundo 3D
  mundo3D.newtexture("aa",#fromcastmember,member("marmol"))
  mundo3D.newtexture("bb",#fromcastmember,member("metal"))
  mundo3D.newtexture("cc",#fromcastmember,member("tierra"))
  mundo3D.newtexture("m7",#fromcastmember,member("metal7"))
  mundo3D.newtexture("h1",#fromcastmember,member("hierba"))
  mundo3D.newtexture("cie",#fromcastmember,member("cielo"))
  mundo3D.newtexture("lad",#fromcastmember,member("ladrillos"))
  mundo3D.newtexture("mad",#fromcastmember,member("madera"))
  mundo3D.newtexture("pie",#fromcastmember,member("piedra"))
  mundo3D.newtexture("rej",#fromcastmember,member("rejilla"))
  mundo3D.newtexture("tel",#fromcastmember,member("tela"))
end

on mouseDown                                    -- cuando se hace clic con el ratón
  tocado = psprite.camera.modelUnderLoc(the clickLoc - point(psprite.left,psprite.top))
                                                -- se detecta el objeto
  if (tocado = Void) then                               -- si no se toca ningún objeto, no hacer nada
    mundo3D.model("Box01").rotate(0,0,10)       -- esto es un truco para “no hacer nada”
    mundo3D.model("Box01").rotate(0,0,-10)
  else              -- si el usuario elige un modelo, capta el objeto clikeado para usar después
    if (tocado.name= "Torus Knot01")    then modelotocado = 1
    if (tocado.name= "Pyramid01")       then modelotocado = 2
    if (tocado.name= "Box01")           then modelotocado = 3
    if (tocado.name= "Sphere01")        then modelotocado = 4
  end if

end
```

4. Crear los botones para elegir las texturas y programar su aplicación a los modelos.

Una vez que tenemos las imágenes de nuestros botones para la elección de la textura en nuestra película de Director, tenemos que hacer que funcione efectivamente como un botón.

Para ello, crearemos un comportamiento con el nombre “Poner X”, donde X será el nombre de la textura. Por ejemplo, el comportamiento para asignar la textura “Cielo” a un objeto se llamará “Poner Cielo”.

En la programación de “Poner Cielo” debemos hacer referencia al modelo seleccionado y por tanto, tendremos que utilizar la variable “modelotocado” que tendrá que ser definida como variable de tipo global.

La asignación de una textura a un objeto, supone la modificación de la propiedad texture, que pertenece a la propiedad shader de cada objeto de la escena 3D.

En posteriores casos prácticos utilizaremos con más profundidad esta propiedad. En realidad se trata de un simple intercambio de una textura por otra, pero la forma en que se escribe es un poco complicada ya que tenemos que hacer referencia al nombre del shader para cada objeto y posiblemente no sabremos cómo se llama. Esto se puede resolver de distintas formas. Una es la utilización de subprogramas o “Xtras” específicos que permiten la fácil exploración de las escenas 3D importadas, uno muy utilizado se llama “3DPI”. Otra forma de resolver la cuestión y sin tanta complicación es aprovechar que la ventana donde escribimos los scripts, cuando aparece un error de sintaxis, proporciona unos controles para profundizar en los elementos individuales del mundo tridimensional.

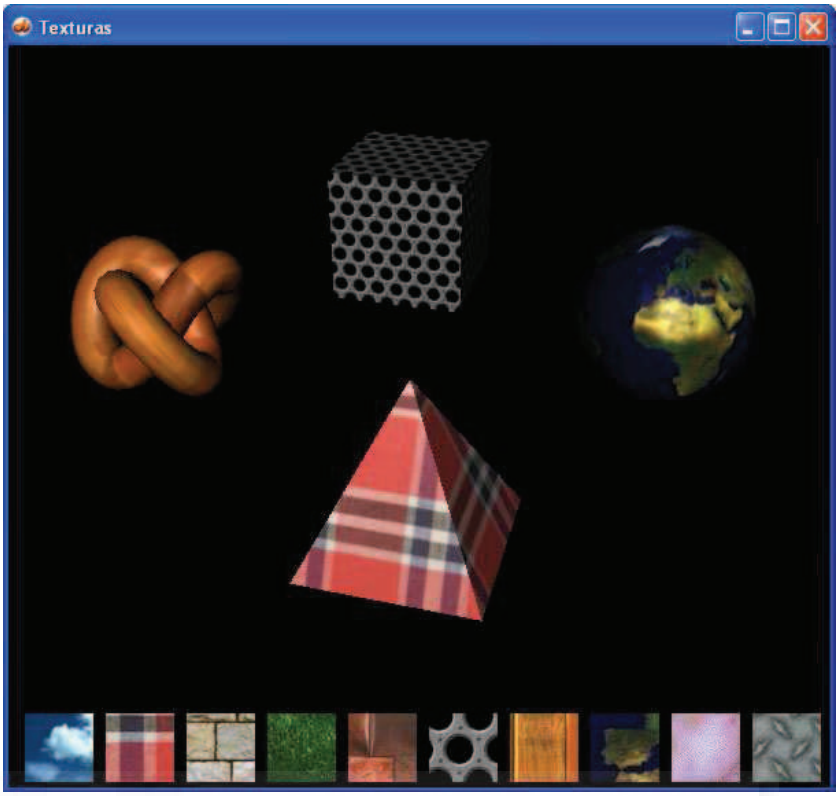


Imagen 5.12.cp.1.8: Ventana de la aplicación ejemplo después de asignar texturas

El comportamiento “Poner Cielo” quedará como sigue:

```
global mundo3D                                    -- variables globales
global modelotocado

on mouseDown

  if (modelotocado= 1) then                -- si el modelo seleccionado es el 1 cambia su textura
    mundo3D.shader("DefaultMaterial2").texture = mundo3d.texture("cie")   -- toro knot

  if (modelotocado= 2) then                -- repetir lo mismo con los demás
    mundo3D.shader("Default Material0").texture = mundo3d.texture("cie")   -- pirámide

  if (modelotocado= 3) then
    mundo3D.shader("Default Material1").texture = mundo3d.texture("cie")   -- caja

  if (modelotocado= 4) then
    mundo3D.shader("Default Material3").texture = mundo3d.texture("cie")   -- esfera

end
```

Por último, podemos copiar y pegar para realizar los demás comportamientos necesarios para los nueve botones de texturas restantes y solo tendremos que modificar el nombre de la textura aplicada.

Conclusiones:

Esta práctica pone de manifiesto que el trabajo con texturas es bastante sencillo en Shockwave 3D y además, que si queremos tener un control mayor sobre características de representación de los modelos 3D como pueden ser los brillos, tipos de superficies, etc., tenemos que estudiar más profundidad el shader de los objetos.

Caso práctico 12.2: Control de superficies.

Objetivo: Esta práctica trata de mostrar cómo podemos controlar los parámetros que van a intervenir en la representación de los modelos 3D que intervienen en nuestras escenas.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Modificar el fichero Director de la práctica anterior para esta práctica.
2. Crear un componente de tipo “deslizador” para reutilizarlo posteriormente.
3. Crear un handler personalizado para modularizar la programación.
4. Crear los scripts para modificar los distintos parámetros de las superficies.

1. Modificar el fichero Director de la práctica anterior para esta práctica.

Vamos a modificar la superficie que muestran los objetos. Por lo tanto, tendremos que entender las variables que entran en juego, los valores que pueden tomar y las implicaciones de ello sobre la visión final de la superficie. Para ello, nos basamos en la práctica precedente. Usaremos los mismos modelos y texturas, pero añadiendo unos botones (muchos de ellos del tipo “deslizador”) para modificar interactivamente las superficies.

Para realizar esta operación debemos de hacer una copia del archivo de director .dir y ensanchar la ventana de Stage. Esto va a suponer que tendremos más espacio para mostrar nuestro trabajo. En la imagen de abajo se puede apreciar la práctica terminada para tener una idea más exacta de lo que queremos desarrollar. Se puede destacar sobre todo, que hemos añadido un espacio en la parte derecha de un tamaño adecuado para que quepan los elementos deslizadores.

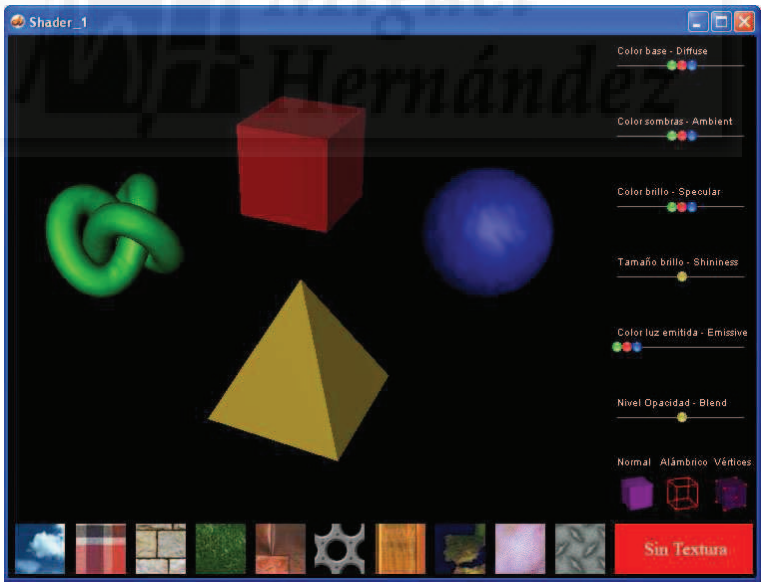


Imagen 5.12.cp.2.1: Vista de la práctica terminada

1.1. Copiar el fichero .DIR. Para ello podemos seguir cualquiera de los métodos conocidos. No vamos a tener que copiar el fichero .w3d, ya que se encuentra embebido en el fichero original. Conviene crear una carpeta nueva para aislar las prácticas.

1.2. Accedemos a la ventana “Property Inspector” y en la ficha “Movie”, cambiamos el Stage Size a 750 X 540. El tamaño original es de 600 x 540 píxeles. La diferencia en anchura es de 150 píxeles. Esta cifra no se elige arbitrariamente, ya que debemos tener en cuenta las dimensiones de los elementos interactivos (los botones deslizadores) que vamos a utilizar.

Estos elementos, como veremos más adelante, van a interpretar gráficamente unos valores que van de 0 a 255 o de 0 a 100. Por cuestiones estéticas, haremos todos los deslizadores de la misma anchura. Nosotros hemos elegido una anchura de 127 píxeles, ya que es la mitad de 255 y por otra parte la cifra 100 nos parecía demasiado pequeña para ser manipulada cómodamente. Por lo tanto, para que quepan holgadamente hemos aumentado la anchura original en 150 píxeles.

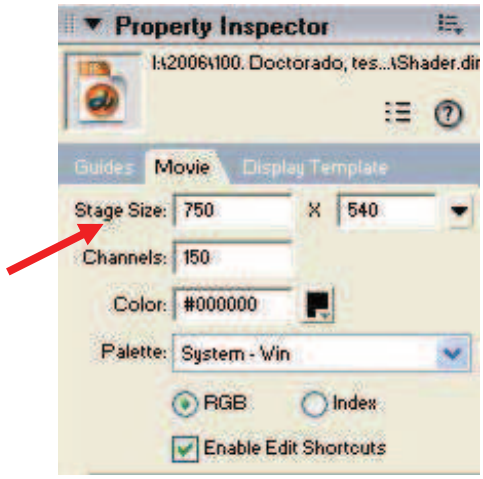


Imagen 5.12.cp.2.2: Ventana Property Inspector

2. Crear un componente de tipo “deslizador” para reutilizarlo posteriormente.

2.1. Cada tipo de parámetro de superficie necesita un tipo de control interactivo. En nuestro caso, vamos a utilizar un control llamado “deslizador” que sirve para elegir un valor entre un rango de valores.

Por ejemplo, para que el usuario elija un determinado color, vamos a utilizar la codificación de color tipo RGB. Esto significa que el color final será el resultado de la mezcla de los tres colores básicos (Red, Green, Blue). Cada uno de estos tres colores puede tomar un valor de 0 a 255.

En estos casos el control más intuitivo de manejar será un deslizador que el usuario puede mover a lo largo de una recta, y que está limitado por los valores 0 y 255.

En nuestro caso, como son muchos los parámetros que se pueden representar por un color y para ahorrar espacio, he utilizado la misma línea para los tres colores, que son representados por tres botones del mismo color que controlan. Por lo tanto, por cada línea tendremos tres botones, uno rojo, uno verde y otro azul.

Los elementos gráficos que intervienen son cuatro. La línea que muestra el rango (en esta práctica es horizontal) y también tenemos 3 botones que serán los elementos que podremos mover, pero nunca más allá de los límites de la línea.

En la imagen 5.12.cp.2.1, se puede apreciar en la esquina superior derecha que para interaccionar con el color base, o sea, el valor diffuse de un modelo, disponemos de un elemento como el descrito anteriormente.

2.2. La programación que tenemos que utilizar se le aplicará al elemento botón mediante un script. El siguiente script será el que se aplique al botón que determinará el parámetro rojo del color base que tendrá el modelo 3D seleccionado.

```
global mundo3D                    -- Variables globales
global diffuserojo                -- *** Variable global que toma su valor del deslizador, del
usuario
```

```
property valor                  -- Variables locales
property delcentro
property slider
property oldFacerValue
property newLocH
property limiteizquierda
property limitederecha

on beginSprite me               -- Cuando del botón aparece en escena lo hace asi:

  slider = sprite(me.spriteNum)
  limiteizquierda=610           -- Limita el movimiento del botón en su extremo izquierdo
  limitederecha=737             -- Limita el movimiento del botón en su extremo derecho
  sprite(slider).locH= 675      -- *** Al comenzar aparece en el centro de la recta

end

on mouseDown me                 -- Cuando se hace clic y arrastra el botón se controla así:

  delcentro = the mouseH - sprite(slider).locH
  oldFaderValue = 0
  repeat while the mouseDown
    newLocH = the mouseH - delcentro
    if newLocH < limiteizquierda  then newLocH = limiteizquierda        -- no sobrepasar límite
    if newLocH > limitederecha then newLocH = limitederecha             -- izquierdo y derecho
    faderValue = newLocH - limiteizquierda - ((limitederecha - limiteizquierda) / 2)
    oldFaderValue = faderValue
    sprite(slider).locH = newLocH
    updateStage
  end repeat

end mouseDown

on mouseUp me                   -- Cuando se deja de arrastrar el botón se hace esto:

  valor=sprite(slider).locH     -- Se toma el valor en la horizontal donde se ha dejado
  diffuserojo=(valor-610)*2     -- *** Se traduce la coordenada al valor que deseamos controlar
  Actualizardatos()             -- Se llama a un subprograma que actualiza el mundo 3d

end mouseup
```

Es evidente que el anterior script, aunque está comentando casi línea por línea, entra de lleno en la programación en Lingo, por lo que para los no programadores puede resultar un poco arduo enfrentarse a esta tarea. Una estrategia que se recomienda es no intentar comprender todo el funcionamiento a la vez sino por partes.

Este script formará una unidad que se puede reutilizar copiando y pegando y luego modificando solamente algunas líneas. Las líneas que habrían de ser modificadas para, por ejemplo, utilizarlo con el parámetro verde del color diffuse, serían las que tienen un triple asterisco.

Estas líneas, como se puede observar solamente son tres: la primera es donde se define la variable global. La segunda lo que hace es posicionar el botón del que se trate en una posición inicial. Al comienzo, la posición de los botones debería ser para los tres colores la misma, pero esto haría que se taparan unos a otros y solo se visualizara el que estuviera más arriba. Con el objeto de mostrar los tres, se ponen adjuntos.

La última y más importante línea que hay que cambiar para cada botón es la que modifica el valor de la variable global, en este caso diffuserojo, que toma le valor siguiente: la variable

valor se obtiene de la línea de arriba y vale la posición horizontal del botón. En la línea que nos incumbe, a ese valor se le restan 610 que es el límite mínimo (por la izquierda) del botón, pero además, como estamos siempre deslizándonos por un recorrido que es de máximo 127 píxeles, debemos multiplicarlo por dos para obtener el valor real (de 0 a 255) que deseamos para el color concreto.

3. Crear un handler personalizado para modularizar la programación.

Hasta ahora, hemos realizado una serie de script, o escrituras, que son en sí mismas pequeñas unidades de programación que en Director se denominan "Behavior". Estas unidades de programación se pueden llamar de muchas formas en los distintos lenguajes de programación. Se suelen llamar módulos, subprogramas, etc., pero obedecen siempre al mismo principio: es mejor separar un programa largo en pequeños programas que independicen el código y lo hagan más manejable.

Los Behavior o comportamientos se asignan a sprites, es decir, apariciones en escena (stage) de un elemento (castmember) multimedia de nuestro programa. Esto se hace simplemente arrastrando el behavior sobre el sprite en cuestión cuando se visualiza este en la ventana Score.

Ahora lo que vamos a realizar es un módulo independiente de los sprites. La idea es que sea general, que se pueda invocar (ejecutar) desde cualquier otro módulo. Esto hace que no tengamos que repetir código innecesariamente. Es como si sacamos factor común de una serie de scripts y lo independizamos para escribirlo una sola vez y reutilizarlo muchas veces.

La programación en sí, no sufre ninguna modificación. Es la forma en que se crea el script lo que es distinto. Para realizarlo, debemos acceder a la ficha Behavior Inspector y pulsar sobre el botón +. Luego hay que acceder a la ventana Property Inspector y en la ficha Script, en el control "Type" elegiremos Movie en lugar de Behavior que es la opción por defecto.

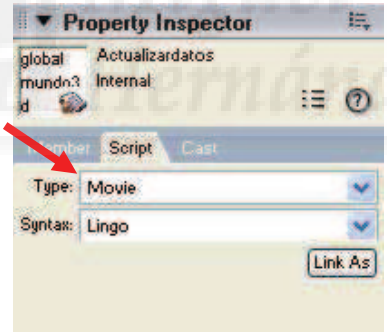


Imagen 5.12.cp.2.3: Script tipo Movie

Esto hará que el nuevo script tenga un icono distinto de los demás como pone de manifiesto la figura de abajo, donde se puede observar que "Actualizardatos" es un script independiente, mientras que "Ambientazul" es un behavior enlazado a un determinado sprite.

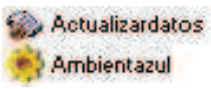


Imagen 5.12.cp.2.4: Scripts de tipo movie y behavior

Para llamar (ejecutar) un handler personalizado debemos utilizar la sintaxis nombre_del_hadler(), por ejemplo en este caso, desde cualquier lugar de un script podemos escribir Actualizardatos().

Dentro del script, podemos utilizar variables globales o locales y las mismas sentencias que en cualquier otro script. El handler se ejecutará desde la línea on nombre_del_handler hasta que acabe con un “end”.

Por ejemplo, el siguiente script es una versión resumida de “Actualizardatos”, que como se puede observar, llama a su vez a un módulo independiente llamado “queshader”. Este modificará el valor de “shadertocado”, que es una variable global, para saber con que modelo y shader estamos trabajando. Lo que hace “Actualizardatos”, en realidad, es poner la visibilidad total para las caras de los modelos 3D, llamados “box01” y “sphere01”, y después actualizar el color diffuse. Este color es el color base del modelo 3D seleccionado con los parámetros diffuserojo, verde y azul, que a su vez habrán sido elegidos por el usuario. Por eso, se llama “Actualizardatos”, porque modifica, si es el caso, el color de un modelo.

```
global mundo3d

global diffuserojo
global diffuseverde
global diffuseazul

global shadertocado

on actualizardatos

  queshader()
  mundo3D.model("Box01").visibility=#both       -- pone los modelos visibles
  mundo3D.model("Sphere01").visibility=#both  -- cambia el color base según variables
globales
  mundo3D.shader(shadertocado).diffuse=rgb(diffuserojo,diffuseverde,diffuseazul)

end
```

4. Crear los Scripts para modificar los distintos parámetros de las superficies.

4.1. Disposición de los controles.

Como se puede observar en la figura 5.12.cp.2.5, los botones deslizadores del parámetro color base han sido modificados y esto también se puede ver en el color de los modelos 3D. La aplicación para mostrar las posibilidades del shader tiene múltiples controles, Estos controles se disponen en una columna a la derecha de los cuatro modelos 3D para poder apreciar las diferencias. En la fila inferior se pueden elegir unas texturas para observar como interfieren los shaders y las texturas en la creación de superficies.

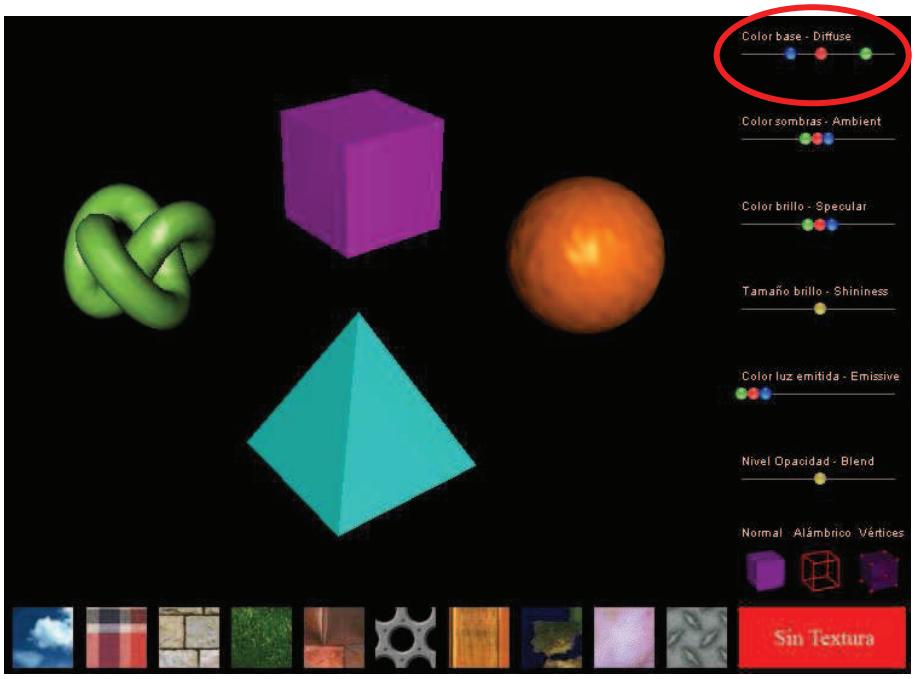


Imagen 5.12.cp.2.5: Modificación del color diffuse

La forma con que finalmente se percibe al objeto 3D depende de muchos parámetros. Fundamentalmente hay dos grandes grupos: texturizados y no texturizados. Las texturas son unos elementos que cambian radicalmente el aspecto de cualquier modelo. Por eso la texturización la tratamos en una práctica aparte.

De todas formas, controlar los demás parámetros de los shader (las texturas son uno más de ellos) nos puede convenir mucho, sobre todo para proyectos artísticos.

El primer elemento a considerar en los elementos no texturizados es el color en general del modelo 3D. El "color" en realidad son tres colores: el color base o diffuse, el de las partes más sombreadas o ambient y el de las partes más iluminadas o specular

4.2. Color base o diffuse.

En primer lugar veremos cómo modificar el color base del objeto. A este color se le denomina técnicamente Diffuse. Al ser este color tan importante, se ha colocado su controlador el más arriba de todos. Cómo se puede observar en la imagen 5.12.cp.2.5, tenemos tres pequeños botones que controlan el color base. Si se sitúan a la izquierda del todo su valor será 0, si se colocan a la derecha del todo tendrá un valor de 255. Si los tres colores tienen el mismo valor o cercano, tomarán tonos grisáceos.

En la imagen anterior se puede apreciar que se han modificado los colores base de cada uno de los modelos utilizando este control. Para ello se hace clic sobre cualquier modelo y luego se modifica los colores arrastrando cada uno de los botones deseados horizontalmente.

De la misma forma se han creado controles y se manipulan para otros parámetros como el color ambient y specular, el tamaño del brillo, el color de luz emitida y el nivel de opacidad. Todos son controlados por botones deslizantes que funcionan de forma similar.

También hemos creado tres botones para controlar el modo de representación o "renderstyle" que como ya vimos en la unidad teórica puede tomar tres valores: fill, wire y point. Por ejemplo, para poner el modelo esfera como aparece en el centro de la imagen 5.12.cp.2.7 escribiríamos:

```
mundo3D.model("Sphere01").renderstyle=#wire
```

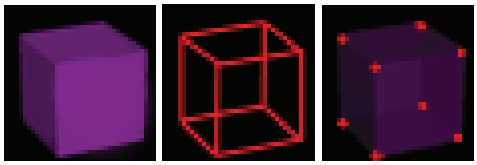


Imagen 5.12.cp.2.6: Botones para variar el modo de representación

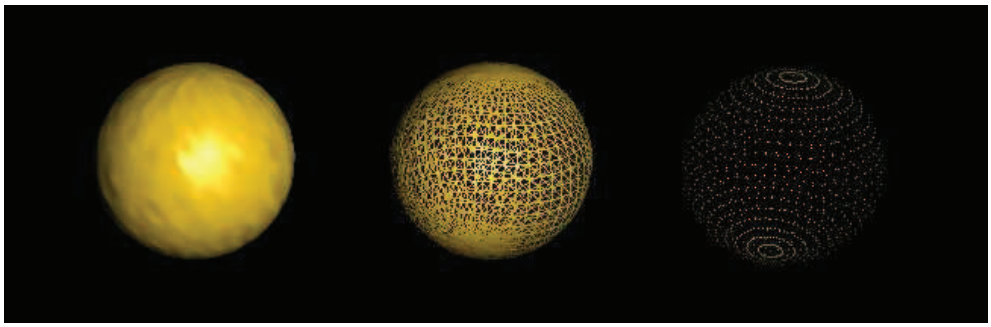


Imagen 5.12.cp.2.7: Botones y modos de representación de un mismo modelo

Conclusiones:

De esta práctica se pueden extraer varias conclusiones. La primera y más obvia es la gran cantidad de variables que pueden llegar a ser importantes en la superficie de los modelos. Otra conclusión es que es un tema muy importante a tratar, y más cuanta más calidad tengamos necesidad de utilizar en nuestros trabajos. Por último, destacar que es un tema difícil de controlar y debemos abordarlo con paciencia. Advertir que se quedan unas cuestiones por cubrir que resolveremos en las próximas prácticas.

Caso práctico 12.3: Texturas con mapas de transparencias: canales alpha

Objetivo: Vamos a realizar una práctica que tiene el objetivo de entender cómo podemos aplicar texturas con canales alfa. Estas texturas especiales son muy útiles y utilizadas por los creadores 3D para crear mapas de opacidad.

Tiempo de realización: 3 horas.

Pasos a realizar:

1. Los mapas de transparencia.
2. Crear los modelos 3D e interfaz.
3. Creación de texturas con canales alpha.
4. Importación de texturas con canales alpha.
5. Crear los scripts para asignar las texturas a los modelos 3D.

1. Los mapas de transparencia.

Como escribimos en la unidad teórica, los mapas de transparencia son imágenes que contienen unos canales especiales llamados alpha. Todas las imágenes bitmaps contienen información del color que tiene cada píxel que lo compone. Si estos colores se codifican en RGB, tendrán un canal que defina los tonos rojos para cada uno de los píxeles, y también tendrá un canal para el verde y otro para el azul. Pero a estas imágenes el autor les puede añadir un canal más: el canal alpha. Este canal define mediante un código en gris, el nivel de transparencia para cada uno de los píxeles de la imagen. De esta manera, además de definir una imagen o mapa de textura, definimos un mapa de transparencia. Esto es útil porque además de colorear un modelo, podemos definir una máscara para crear orificios o materiales semitransparentes, etc.

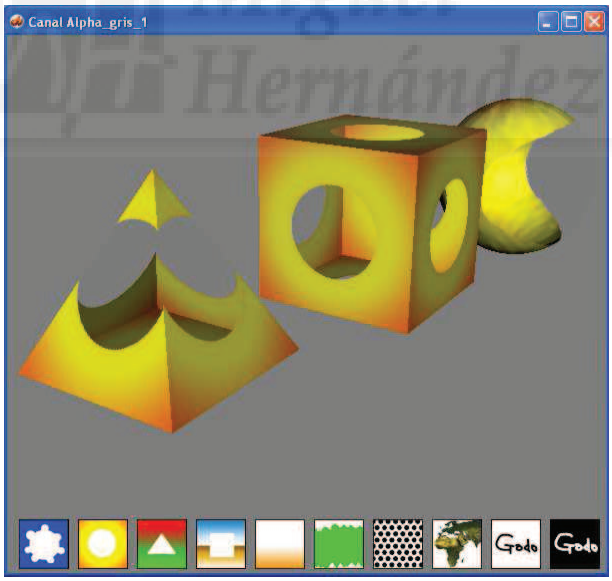


Imagen 5.12.cp.3.1: Vista de la práctica terminada

2. Crear los modelos 3D e interfaz.

Cómo se puede apreciar en la imagen 5.12.cp.3.1, donde se puede visualizar el caso práctico en ejecución, los resultados de una misma texturización conteniendo canales alpha, depende mucho de los modelos 3D a los que se aplique.

Por esta razón, la textura elegida no se aplica individualmente a los objetos sino que se aplica a los tres a la vez. Para seguir con los ejemplos ya realizados de superficies, se ha seguido el mismo método de interfaz pero con la salvedad mencionada con el objeto de que el usuario pueda prestar atención a lo verdaderamente importante de este caso práctico.

Los modelos se han dispuesto en perspectiva y en ocasiones, se pueden ver a unos modelos a través de otros, ya que tienen huecos en sus caras debido precisamente a la aplicación de los mismos canales alpha a los tres elementos.

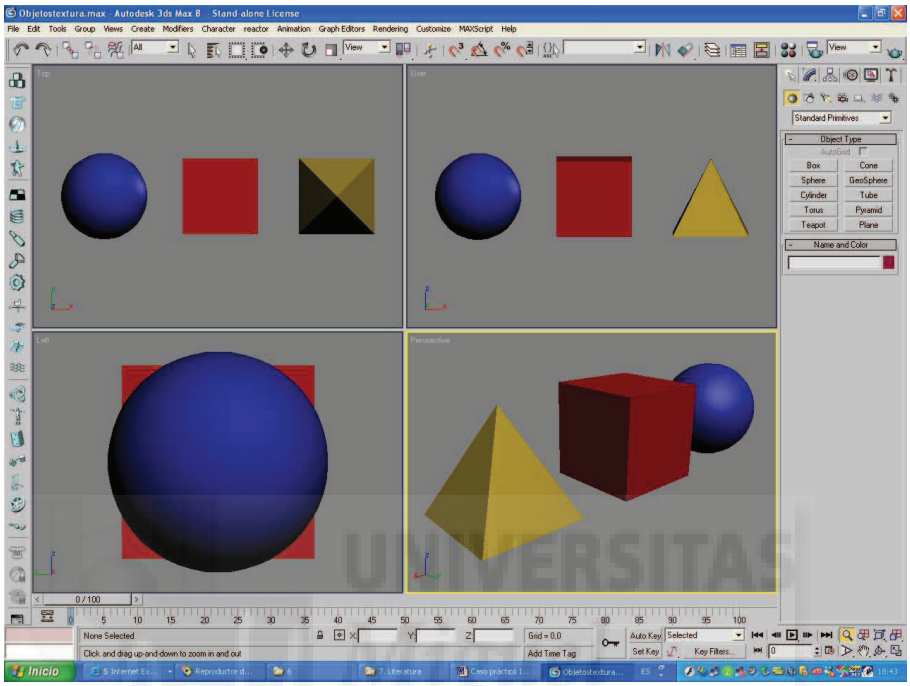


Imagen 5.12.cp.3.2: Vista del fichero de 3D Studio MAX

Para realizar los modelos utilizamos como siempre el programa 3D Studio MAX, donde se crearon los objetos de otras prácticas, pero desechando el “Torus Knot” y disponiéndolos de otra manera. Hemos buscado que aparecieran en la vista de perspectiva uno tras otro ocupando gran parte de la pantalla. El resultado antes de importar a fichero .w3d es el que se muestra en la imagen 5.12.cp.3.2, donde podemos apreciar que se mantienen los colores originales de los objetos.

Cómo hemos escrito, se pueden aprovechar las anteriores prácticas para no empezar desde el principio, por lo que crearemos una carpeta propia para esta práctica y luego haremos una copia del fichero .dir que deseemos para estar en disposición de empezar a modificarlo y obtener un interfaz como el que se representa en la imagen 5.12.cp.3.2.

3. Creación de texturas con canales alpha.

Para la creación de las texturas que incluirán canales alpha hemos utilizado ®Adobe ®Photoshop. Este programa es reconocido como uno de los mejores en su clase. Los ficheros que debemos producir son imágenes de tipo bitmap. Los guardaremos con la extensión .bmp, ya que pueden incluir colores “normales” además de información de canales alpha. Podemos utilizar tantas capas como queramos. Siempre teniendo en cuenta que al exportarlo como fichero .bmp, solo se verá una capa de color, que además, se verá afectada por la información de opacidad del canal alpha.

Vamos a suponer que el alumno tiene un conocimiento previo mínimo de Photoshop. Nuestro fichero final contendrá información de capas y de canales.

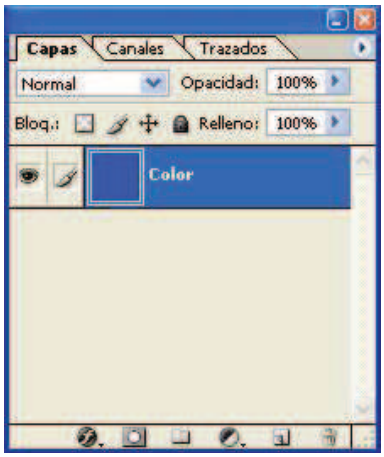


Imagen 5.12.cp.3.3: Vista de la ficha "Capas" de la textura de muestra

Para este primer ejemplo vamos a realizar el fichero correspondiente a la primera textura que se puede ver en la práctica terminada. Se trata de una figura que generará un agujero no geométrico en las caras de los modelos. La parte visible será de un azul uniforme.

En la imagen 5.12.cp.3.3, se puede observar que tenemos una sola capa que hemos rellenado de color azul y en la imagen siguiente como queda esta textura con un tamaño de 200 x 200 píxeles. Para realizar este trabajo haremos los siguientes pasos:

3.1. Pulsamos sobre el menú Archivo > Nuevo (o Ctrl.+N). Aparece una caja de diálogo llamada "Nuevo".

3.2. Lo más importante es fijarnos en los parámetros de anchura y altura y en las unidades en que están medidos. Escribiremos 200 píxeles tanto para la anchura como para la altura. Los demás parámetros los dejaremos como están, ya que en principio no debe haber ningún factor que impida seguir con la práctica de forma correcta, ni siquiera el valor de resolución, ya que para esta tarea no tiene importancia.

3.3. Elegir el color frontal. Para ello basta con hacer doble clic en el selector del color y en la caja de diálogo "Selector de Color" poner los parámetros R: 0, G: 50, B: 250.

3.4. Elegir la herramienta "Bote de pintura" (o pulsar G). Y hacer clic sobre nuestra imagen. El resultado producido debe parecerse a la imagen 5.12.cp.3.4. O sea, toda la imagen está llena de un color azul intenso.

Podemos realizar el diseño que queramos. Por ejemplo, podemos utilizar los degradados, los pinceles, importar imágenes fotográficas, etc.

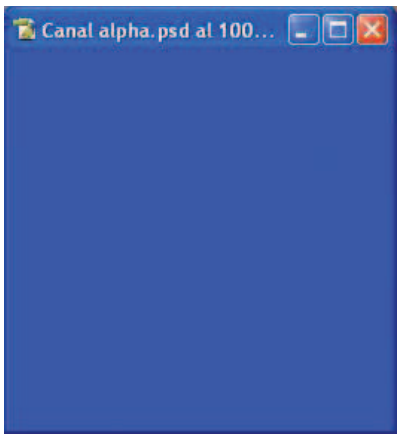


Imagen 5.12.cp.3.4: Vista del color base de la textura

3.5. Ahora lo que tenemos que hacer es abrir un canal nuevo. Para ello accedemos a la ficha canales. Si no la tenemos visible hacemos: menú Ventana > Canales. Debe de aparecer una ventana parecida a la imagen siguiente. Esto es, tendremos ya 3 canales, uno por cada color básico y otro con los tres mezclados llamado RGB.

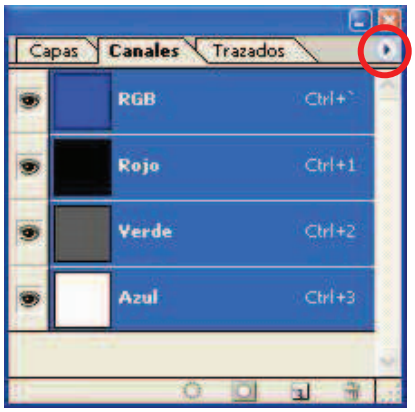


Imagen 5.12.cp.3.5: Los canales normales antes de crear el canal alpha

3.6. Pulsar sobre el botón redondo con un triángulo hacia la derecha que está a la altura de las fichas que aparece señalizado en la imagen anterior y aparecerá un menú como se muestra en la imagen que sigue.

En este menú debemos pulsar sobre la opción “Nuevo canal”.

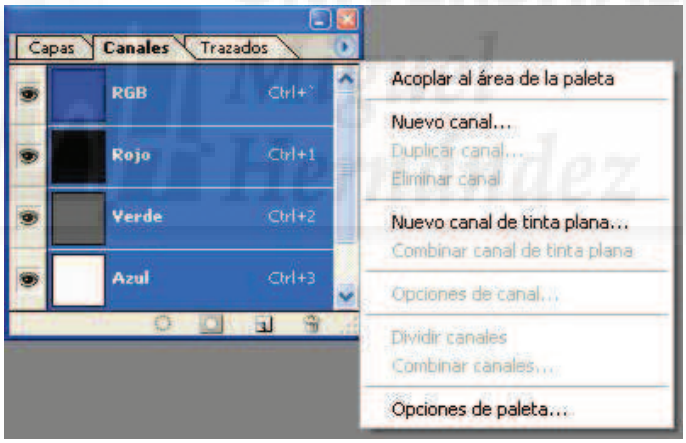


Imagen 5.12.cp.3.6: Menú de canales

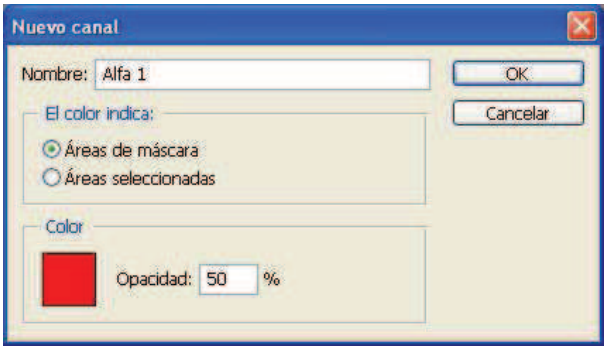


Imagen 5.12.cp.3.7: Ventana para crear el canal alpha

3.7. Aparecerá la caja de diálogo llamada “Nuevo canal” como la que podemos ver en la imagen 5.12.cp.3.7. En esta caja tenemos distintos parámetros pero no nos hace falta modificar ninguno. Podemos cambiar el nombre si lo deseamos. Luego pulsamos el botón “Ok”.

3.8. Observar en la lista de canales que tenemos un canal más. Lo seleccionamos haciendo clic sobre él y entonces su fondo se pondrá de color azul.

3.9. Ahora ya podemos pintar el canal alpha deseado. En nuestro caso hemos hecho la forma que se aprecia en la imagen 5.12.cp.3.8. En realidad, puede ser cualquier dibujo. Lo interesante es que si pintamos en negro puro una zona, esta se volverá totalmente transparente en los modelos a los que se la aplique. Por lo tanto, hay que ser conscientes que estamos “fabricando” un orificio para nuestros modelos un poco inusual, sobre todo por su irregularidad. Esto lo hemos realizado así para poner en evidencia que se puede “pintar” la opacidad tanto con la forma como con la intensidad que se desee.

3.10. Vamos a guardar el resultado. Para ello accedemos al menú y ejecutamos Archivo > Guardar como. Aparecerá la caja de diálogo “Guardar como”. Lo más importante es que el parámetro Tipo lo elijamos .bmp y nos tenemos que asegurar que la opción “Canales alfa” esta elegida. Por último, le ponemos el nombre que deseemos y pulsamos el botón “Guardar”. En nuestro caso lo hemos nombrado como “CA1”.

Imagen 5.12.cp.3.8: Vista del canal alpha pintado

4. Importación de texturas con canales alpha.

4.1. Abrir el fichero .dir. En esta práctica lo hemos llamado “canales_alpha.dir”.

4.2. El siguiente paso es incorporar al fichero “canales_alpha.dir” la textura creada anteriormente como un miembro más del castmember. Ejecutamos el comando File > Import y aparecerá la caja de diálogo “Import Files into Internal”.

4.3. Elegimos el fichero que hemos creado anteriormente llamado “CA1.bmp” y pulsamos el botón “Add”. Entonces nos muestra la lista de archivos por si queremos importar más archivos. Como solo vamos a trabajar con una textura, pulsamos el botón “Import”.

4.4. Debe aparecer la caja de diálogo llamada “Image Options for CA1.bmp”. Aquí vamos a tener especial cuidado con dos parámetros.

La profundidad de color o “Color Depth”. Por defecto está a la máxima calidad de la imagen. La elección de modificar este parámetro depende del trabajo que estemos realizando. Si necesitamos mucha calidad en las texturas, tenemos que saber que serán más costosas de calcular y por lo tanto penalizaremos la tarea del sistema de representación. Además, existen otros parámetros que deben estar en concordancia con la elección de una calidad máxima de las texturas como es “renderformat”. Este valor controla por programación la calidad de representación, y sería una pena, hacer que nuestro fichero final, fuera más grande y no se

aprovechase luego en la representación de nuestro trabajo 3D. La explicación de cómo se utiliza y para qué Renderformat lo veremos en la sección que sigue.

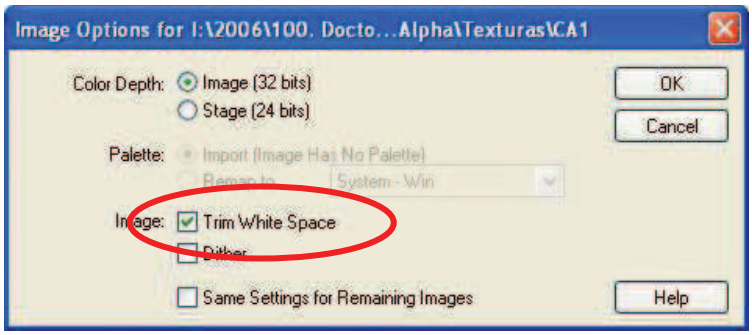


Imagen 5.12.cp.3.9: Tipo de imagen en la importación a Director

También nos tenemos que fijar en quitar la opción Image: Trim White Space, ya que sin este detalle el fichero de imagen se importará correctamente pero sin canales alpha, por lo que todo nuestro trabajo no habrá servido de nada. En la imagen 4.11.9 se puede observar dónde se sitúa dicha opción. Solamente tenemos que deshabilitarla.

4.5. Una vez el fichero en nuestro archivo .dir, debemos asegurarnos que en la ficha Bitmap de la ventana “Property Inspector” tenemos la opción “Use alpha” seleccionada tal y como muestra la imagen 4.11.10 y ya tendríamos preparada la textura para ser utilizada en nuestro mundo 3D.

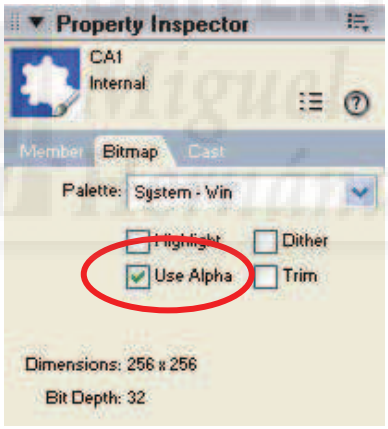


Imagen 5.12.cp.3.10: Declarar como alpha el bitmap importado

5. Crear los scripts para asignar las texturas a los modelos 3D.

Como se puede apreciar en la imagen 5.12.cp.3.1, al principio de esta práctica, tenemos diez texturas con canales alpha para poder mostrar algunos de los efectos que podemos conseguir con ellas.

Para aplicar cada una de las texturas es necesario crear un botón. En nuestro caso no es más que una imagen que representa la textura con que se representará a los tres modelos 3D que forman nuestro trabajo. Al ser imágenes, las importaremos como bitmaps normales pero aprovechándonos de los ficheros generados en Photoshop para crear las texturas a las que hacen referencia.

Una vez que tenemos los “botones” incorporados a nuestra práctica, tenemos que escribir un script que será de tipo “behavior”, es decir, un trozo de programación que no será general, sino que solo se ejecutará cuando se pulse sobre el botón en cuestión. El objetivo de la

programación asignada a este botón es bastante sencilla en su estructura: solo tendremos un evento al que responder que es el de “mousedown”, es decir, el que se produce como consecuencia de hacer clic sobre el botón.

Aunque se puede hacer una programación más eficiente, hemos optado por la claridad, y por tanto, para cada botón se utiliza un script independiente con las siguientes características:

5.1. Primeramente nos asegurarnos de que se van a ver las caras posteriores y anteriores de los objetos, esto lo hacemos con la propiedad “visibility” puesta a #both. Por ejemplo con la línea de código:

```
mundo3D.model("Box01").visibility=#both
```

5.2. También nos aseguramos de que los shaders de cada uno de los modelos 3D permiten la transparencia. Esto lo hacemos con la propiedad “transparent” puesta a true. Por ejemplo:

```
mundo3D.shader("Default Material6").transparent=trae
```

Luego para cada shader se realizan las siguientes tareas:

5.3. Introducimos la textura que corresponda en el shader de cada modelo 3D. Es evidente que habrá que introducir la textura en nuestro mundo 3D en otro script. Nosotros lo hemos escrito en el script “Programación” que está asignado al sprite del mundo 3D.

5.4. Posteriormente asignamos a la propiedad renderformat un valor de #rgba8888. Este valor es el de más alta calidad posible y permite representar unas texturas con una profundidad de 24 bits para el color y 8 bits para canales alpha. Hemos tomado esta decisión al tratarse de una práctica que muestra las posibilidades de estas técnicas, pero este hecho se tiene que estudiar para cada caso en concreto. Por ejemplo, para la primera textura se podría utilizar un valor más bajo como #rgba4444, que permite representar unas texturas con una profundidad de 12 bits para el color y 4 para canales alpha, pero en las siguientes texturas donde se representa tanto degradaciones de color como degradaciones de opacidad, estos valores resultarían escasos, ya que se apreciarían las fronteras de color y por tanto los degradados dejarían mucho que desear en su visualización.

5.5. El parámetro blendfunction de cada shader está puesto a #multiply. Es un valor normal, aunque se puede modificar cuando se utilizan más de una textura y se buscan efectos un poco más complicados. En nuestro caso este valor funcionará correctamente.

5.6. También ponemos el parámetro blendsource al valor #alpha por que si no es así, simplemente no se representarían los canales que hemos incorporado y que están embebidos en nuestras texturas. Este es uno de los detalles que debemos de cuidar especialmente.

El script que sigue puede ser un ejemplo de cómo quedaría el behavior asignado a un sprite de un botón que asignará la textura “ca11” a los tres modelos:

```
global mundo3D
global modelotocado

on mouseDown

  mundo3D.model("Box01").visibility=#both
  mundo3D.model("Sphere01").visibility=#both
  mundo3D.model("Pyramid01").visibility=#both

  mundo3D.shader("Default Material6").transparent=true
  mundo3D.shader("Default Material7").transparent=true
  mundo3D.shader("Default Material8").transparent=true

  mundo3D.shader("Default Material6").texture=mundo3d.texture("ca11")
```

```
  mundo3D.shader("Default Material6").texture.renderformat=#rgba8888
  mundo3D.shader("Default Material6").blendfunction=#multiply
  mundo3D.shader("Default Material6").blendsource=#alpha

  mundo3D.shader("Default Material7").texture=mundo3d.texture("ca11")
  mundo3D.shader("Default Material7").texture.renderformat=#rgba8888
  mundo3D.shader("Default Material7").blendfunction=#multiply
  mundo3D.shader("Default Material7").blendsource=#alpha

  mundo3D.shader("Default Material8").texture=mundo3d.texture("ca11")
  mundo3D.shader("Default Material8").texture.renderformat=#rgba8888
  mundo3D.shader("Default Material8").blendfunction=#multiply
  mundo3D.shader("Default Material8").blendsource=#alpha

end
```

El script que sigue es un ejemplo de cómo quedaría el behavior “Programación” asignado al único sprite de mundo 3D:

```
global mundo3D
global modelotocado

property psprite
property tocado

on beginSprite me
  pSprite = sprite(me.spriteNum)

-- añade una nueva textura al mundo 3D desde un castmember que contiene un bitmap
  mundo3D.newtexture("ca11",#fromcastmember,member("CA1"))
  mundo3D.newtexture("ca22",#fromcastmember,member("CA2"))
  mundo3D.newtexture("ca33",#fromcastmember,member("CA3"))
  mundo3D.newtexture("ca44",#fromcastmember,member("CA4"))
  mundo3D.newtexture("ca55",#fromcastmember,member("CA5"))
  mundo3D.newtexture("ca66",#fromcastmember,member("CA6"))
  mundo3D.newtexture("ca77",#fromcastmember,member("CA7"))
  mundo3D.newtexture("ca88",#fromcastmember,member("CA8"))
  mundo3D.newtexture("ca99",#fromcastmember,member("CA9"))
  mundo3D.newtexture("ca10",#fromcastmember,member("CA10"))

end
```

En este último script, lo único a destacar es la incorporación mediante el método “newtexture” de las texturas al interior del mundo 3D.

Conclusiones:

En este trabajo se puede apreciar que los canales alpha tienen muchas aplicaciones. Una de ellas es que permiten “ahorrar” mucho trabajo a un modelador de objetos tridimensionales sobre todo si es una pieza con orificios, huecos, etc. Y de esto también se deduce que los cálculos para su representación serán menores lo que redunda en una mejora en la calidad de la visualización. Por todo esto es necesario conocer estas técnicas y aplicarlas convenientemente.

Caso práctico 12.4: Texturas con mapas de reflejos

Objetivo: La práctica que sigue pretende mostrar qué son y cómo se utilizan los llamados mapas de reflejos. Estos mapas son utilizados para crear la ilusión de que algunos objetos tienen superficies que reflejan el entorno.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Introducción a los mapas de reflejos.
2. Crear los modelos 3D y el interfaz.
3. Creación e importación de los mapas de reflejos.
4. Crear los Scripts para asignar los mapas de reflejos a los modelos 3D.

1. Introducción a los mapas de reflejos.

Los mapas de reflejos son imágenes que se van a utilizar para crear unos reflejos que en realidad son falsos. Estas técnicas se utilizan en todos los programas de 3D del mercado, pero sobre todo en los que trabajan en tiempo real, como videojuegos, etc.

En los programas de modelado como 3D Studio MAX estas texturas pueden formar parte de otras texturas más complejas, y su interacción con los demás componentes hacen que la ilusión funcione muy bien si se emplean correctamente.

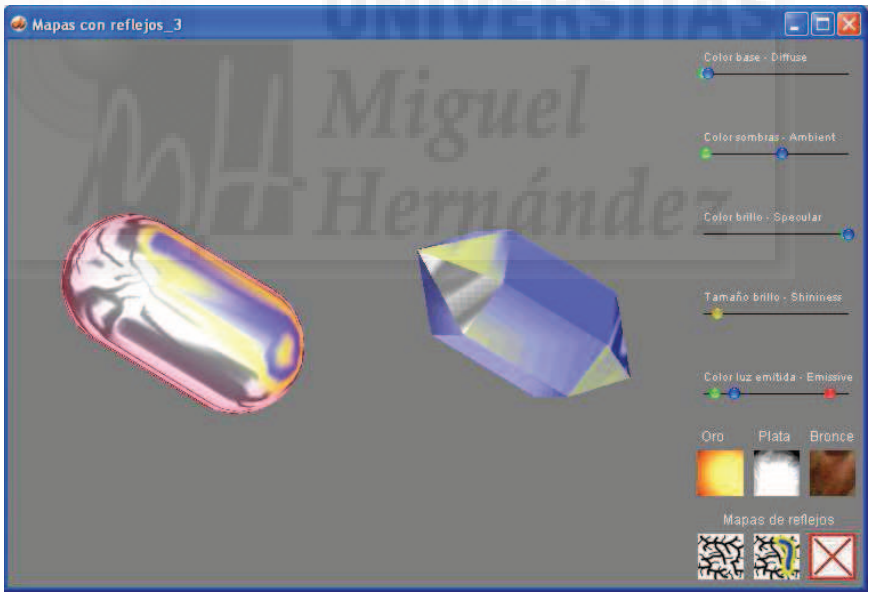


Imagen 5.12.cp.4.1: Vista de la práctica "Mapas de reflejos" terminada

Es evidente que los mapas de reflejos están creados con anterioridad a la ejecución del programa y por tanto no reflejan el entorno que le rodea en tiempo real. Esto quiere decir que al estar ya precalculada la textura, no cargamos de más trabajo el sistema de representación de tiempo real, esta es la parte positiva, pero la parte negativa es que toda la visualización es falsa. Dicho con otras palabras, en Shockwave, no se pueden crear objetos como por ejemplo espejos en una escena y que funcionen "realmente" reflejando su entorno en tiempo real. En programas como MAX si se puede hacer con métodos avanzados de visualización que tardan mucho tiempo en calcularse y por tanto no son en tiempo real, ya que esto supone una velocidad de representación de la escena de 24 fotogramas por segundo.

En un sistema como el nuestro los mapas de reflejos se pueden utilizar con buenos resultados para representar objetos metálicos como oro, plata, cromo, etc.

El shader estándar contiene un método predefinido para utilizar los mapas de reflejos que se llama Shader.feflectionmap. Este método no impide que pueda ser usado conjuntamente con otros parámetros y métodos del shader estándar que ya hemos estudiado anteriormente, como los tipos de colores y parámetros y que bien utilizados pueden tener efectos en la superficie muy llamativos como los mostrados en la imagen 5.12.cp.4.1, donde se han mezclado los mapas de reflejos con distintos colores.

2. Crear los modelos 3D y el interfaz.

Para realizar esta práctica hemos vuelto a utilizar 3D Studio MAX, donde se crearon dos modelos tridimensionales pero con características distintas. Esto se hizo pensando en los tipos de mapas de reflejos que queremos utilizar. Un modelo es muy curvo y sin aristas y el otro todo lo contrario. Por eso hemos elegido una “capsule” y un “spindle”. Luego, para que a su vez fueran comparables las creamos del mismo tamaño y les asignamos la misma animación.

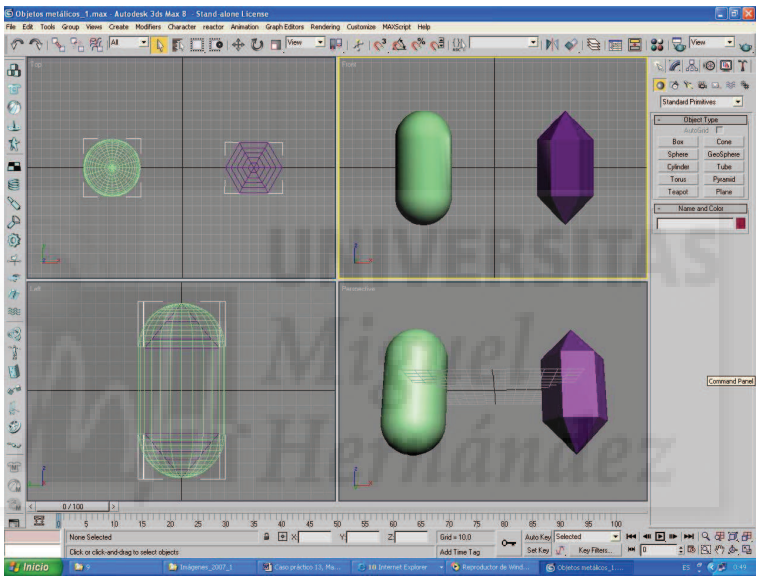


Imagen 5.12.cp.4.2: Vista del fichero de 3D Studio MAX

Para crear los modelos realizar los siguientes pasos:

2.1. Ir al panel Create > Geometry > Extended primitives > y elegir Capsule.

2.2. En la vista Top hacer clic y arrastrar para crear la sección horizontal y luego volver a arrastrar para definir la longitud de nuestro modelo. Si es necesario, acceder al panel Parameters para modificar los valores de Radius y Height.

2.3. Repetir la operación para el modelo Spindle.

El resultado antes de importar al fichero .w3d se muestra en la imagen 5.12.cp.4.2.

La interfaz de la práctica se puede ver en la imagen 5.12.cp.4.3. Podemos observar que utilizamos un fondo gris, porque como los cambios que efectuaremos en la superficie de los modelos 3D son a veces muy sutiles pueden no apreciarse convenientemente en fondos muy oscuros o demasiado claros.

Este interfaz utiliza solamente dos modelos 3D. Esto tiene una explicación y es la forma en que actúan los mapas de reflejos. La diferencia se puede poner de manifiesto si aplicamos los mismos parámetros a modelos curvos o con caras afiladas. De todas formas se permite aplicar

y probar distintos parámetros del shader y distintos mapas de reflejos (o ninguno) a los dos modelos de forma independiente para poder comparar y experimentar más libremente.

Para elegir entre uno y otro modelo basta con hacer clic con el botón izquierdo del ratón sobre él y empezar a modificar sus parámetros con los controles situados en la parte derecha del programa.

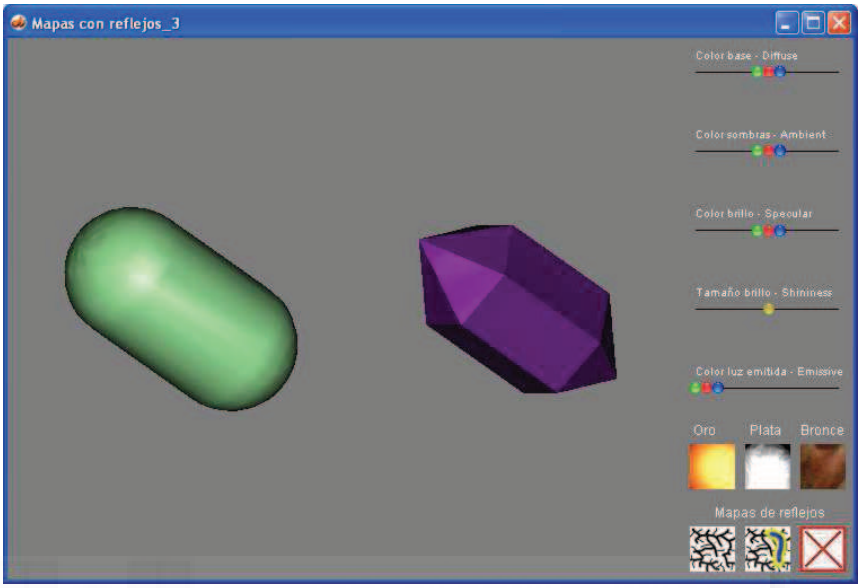


Imagen 5.12.cp.4.3: Vista de la práctica al iniciar su ejecución

Cómo siempre, hemos aprovechado las anteriores prácticas para no empezar desde el principio. Para seguir con la misma filosofía de trabajo, crearemos una carpeta propia para esta práctica y la llamamos “mapas de reflejo”. En este punto, es necesario estudiar que trabajos anteriores se asemejan más a lo que queremos hacer. Creemos que la más cercana es la llamada “control del shader” y por lo tanto hacemos una copia del fichero .dir de esta práctica y la copiamos en la carpeta recién creada.

Los controles de que va a disponer nuestra práctica deben responder a la funcionalidad que esperamos de ella, y estos son de 3 tipos diferentes:

1. Controles de parámetros de color del shader: en este caso reproduciremos los controles tipo deslizadores de orientación horizontal que nos van a permitir elegir el valor de tres variables (la cantidad de rojo, verde y azul) para definir un determinado color. Mediante estos controles controlaremos los parámetros diffuse, ambient, specular, emissive y shininess.

2. Botones de metales: hay que destacar que no se trata de texturas aplicadas a los objetos, sino de colocar los parámetros de color a determinados valores. Esto facilita la modificación de dichos valores, y por lo tanto, la experimentación. Se utilizan valores de “metales” porque como hemos visto anteriormente funcionan bien con los mapas de reflexión. Los modelos originalmente tienen unos colores poco recomendados para la aplicación de estos mapas. Esto se debe a que queremos poner de manifiesto que merece la pena el estudio de los parámetros básicos del shader para obtener buenos resultados.

3. Botones de mapas de reflejos: estos botones son tres. El primero aplica una textura en blanco y negro muy sencilla de pintar, ya que son simplemente rayas de distintos grosores. El segundo se basa en el primero y simplemente se añade una pequeña mancha de color. La última es un botón para quitar los mapas de reflejos.

Podríamos crear un entorno más sofisticado, sobre todo con más objetos y utilizar fotos como mapas de reflejos, pero en nuestra práctica esto no tiene mucho sentido. Todo lo contrario sucedería en un entorno en que se quisiera imitar la realidad.

3. Creación e importación de los mapas de reflejos.

En este apartado vamos a estudiar cómo se obtienen los valores para los parámetros del shader con el objeto de crear metales como oro, plata y bronce y además, los mapas de reflejos utilizados. Vamos a comenzar por los metales. Ya hemos escrito que utilizamos la "imitación" de metales por que son los que mejor funcionan al añadirle los mapas de reflejos. Estos se obtendrán modificando los valores de los parámetros básicos del shader estándar.

Los valores para obtener una superficie parecida al oro serán los siguientes:

Concepto	Parámetro	Valor
Color base	Shader.Diffuse:	rgb(180,20,0)
Color de las sombras	Shader.Ambient:	rgb(220,100,5)
Color de los brillos	Shader.Specular:	rgb(255,255,150)
Tamaño del brillo	Shader.Shininess:	2
Color de la luz emitida	Shader.Emissive:	rgb(115,60,15)

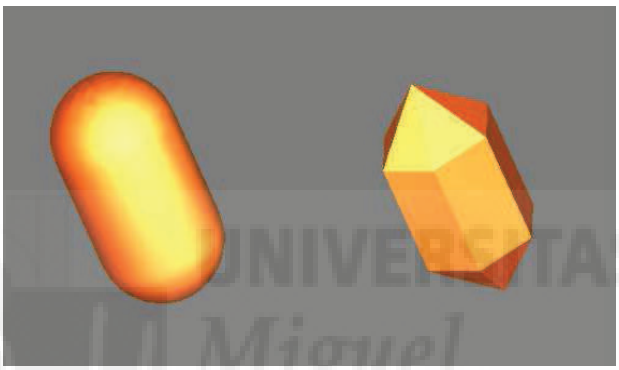


Imagen 5.12.cp.4.4: Los modelos con superficie de "oro"

Los valores para obtener una superficie plateada los hemos colocado como sigue:

Concepto	Parámetro	Valor
Color base	Shader.Diffuse:	rgb(0,0,0)
Color de las sombras	Shader.Ambient:	rgb(0,0,0)
Color de los brillos	Shader.Specular:	rgb(255,255,255)
Tamaño del brillo	Shader.Shininess:	10
Color de la luz emitida	Shader.Emissive:	rgb(0,0,0)

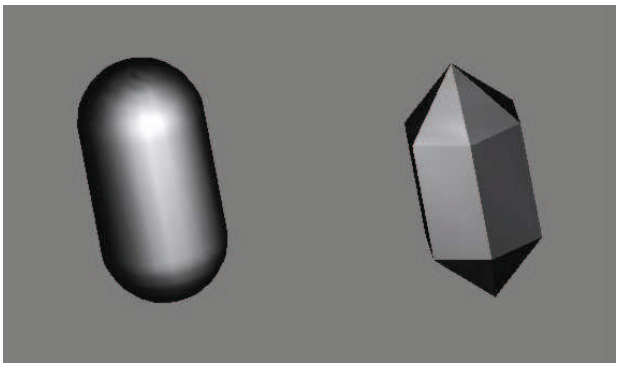


Imagen 5.12.cp.4.5: Los modelos con superficie de "plata"

Los valores para obtener una superficie que imite al bronce los hemos colocado así:

Concepto	Parámetro	Valor
Color base	Shader.Diffuse:	rgb(50,0,0)
Color de las sombras	Shader.Ambient:	rgb(0,0,0)
Color de los brillos	Shader.Specular:	rgb(255,255,150)
Tamaño del brillo	Shader.Shininess:	5
Color de la luz emitida	Shader.Emissive:	rgb(15,15,15)

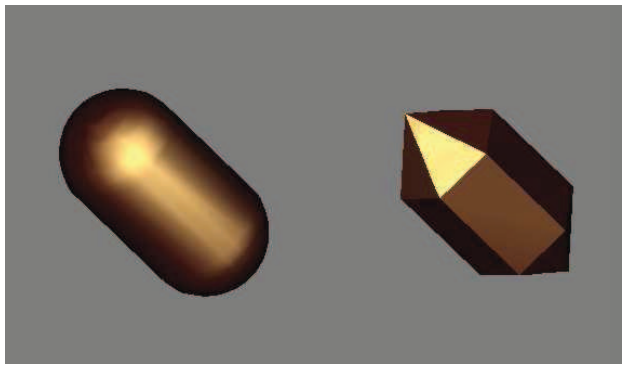


Imagen 5.12.cp.4.6: Los modelos con superficie de "bronce"

Ahora vamos a crear los mapas de reflejos. Pueden darse dos estrategias si buscamos mapas con distintas características. Por una parte, podemos utilizar fotos si un elemento queremos que refleje un paisaje o imágenes de la propia escena renderizada. Por otro lado, podemos crear las imágenes que queramos en programas como Photoshop.

En la imagen 4.12.7, se pueden observar los dos mapas de reflejo que hemos pintado en Photoshop y que están incluidos en la práctica. Es muy recomendable experimentar sobre todo con las dos variantes citadas: nítido / difuminado y fotográfico / pintado, además de estudiar los colores de la escena.

Imagen 5.12.cp.4.7: Mapa de reflejos generales

4. Crear los Scripts para asignar los mapas de reflejos a los modelos 3D.

En el interfaz que se puede apreciar en la imagen 5.12.cp.4.1 se distinguen tres botones para elegir un tipo determinado de metal base y tres botones para aplicar los mapas de reflejo.

Escribiremos un "behavior" para cada tipo de botón según su funcionalidad. Lo que tienen en común es que los seis botones sólo responden al evento "mousedown", o sea, que lo que tienen que hacer solo se produce si se hace clic con el botón izquierdo del ratón.

4.1. Seguidamente se reproduce el script que sirve para asignar una superficie que imita al oro al objeto pulsado:

```
global diffuserojo          //se crean tantas variables globales como parámetros hemos de
modificar
global diffuseverde
global diffuseazul

global ambientrojo
global ambientverde
global ambientazul

global specularrojo
global specularverde
global specularazul

global emissiverojo
global emissiveverde
global emissiveazul
global tamanobrillo

on mouseDown

  diffuserojo= 180          //se colocan los parámetros a los valores antes estudiados
  diffuseverde= 20
  diffuseazul= 0

  ambientrojo=220
  ambientverde=100
  ambientazul=5
  specularrojo=255
  specularverde=255
  specularazul=150

  emissiverojo=115
  emissiveverde=60
  emissiveazul=15

  tamanobrillo=2

 Actualizardatos()

end
```

4.2. Repetir este mismo behavior pero con los valores que correspondan para los botones de plata y bronce. Para realizar esta tarea más eficientemente se puede usar el comando “Copy Cast Member” que se encuentra en el menú contextual del script.

4.3. Crear el behaviour que debe ir asignado al botón para incluir uno de los mapas de reflejos que se representa en la imagen 5.12.cp.4.7, como se escribe bajo estas líneas. En realidad lo único que hace es poner a 1 una variable global llamada reflejo y llamar al script global “Actualizardatos”.

```
global reflejo

on mouseDown
  reflejo=1                                  // se pone la variable globar a 1
  Actualizardatos()                          // se pide actualizar el mundo 3D
end
```

4.4. El script “Actualizardatos” tiene como objetivo modificar el mundo 3D actualizando los parámetros necesarios si estos han sido modificados por distintos script de botón. Es de tipo “movie” para que pueda ser llamado desde cualquiera de estos script tipo behavior.

```
on actualizardatos

if (reflejo =1) then mundo3D.shader(shadertocado).reflectionmap=mundo3d.texture("r1")
        //asignamos R_1
  else if (reflejo =2) then mundo3D.shader(shadertocado).reflectionmap=mundo3d.texture("r2")
        //al  shader del
     else if (reflejo =3) then mundo3D.shader(shadertocado).reflectionmap=void
        //modelo “tocado”
end
```

4.5. Por último, esta aplicación no funcionaría si los mapas de reflejos no formaran parte de nuestro mundo 3d como una textura más. Por lo tanto, tenemos que incluir al menos las siguientes líneas en nuestro behavior “Programación” que es asignado al sprite del mundo 3D. Esto supone que los mapas de reflejos han sido importados como imágenes tipo bitmaps, por ejemplo, como .jpg´s y han pasado a formar parte del Cast, con los nombres “R_1” y “R_2”.

```
global mundo3D
global modelotocado
global reflejo

property psprite
property tocado

on beginSprite me

  pSprite = sprite(me.spriteNum)
  mundo3D.newtexture("r1",#fromcastmember,member("R_1"))
        // incorporamos el mapa de reflejo sin color
  mundo3D.newtexture("r2",#fromcastmember,member("R_2"))
        // incorporamos el mapa de reflejo con color
  modelotocado=3

end
```

Conclusiones:

Esta práctica es bastante sencilla en el aspecto de la programación, ya que el único método nuevo empleado es “reflectiomap”. Este hecho, sin embargo, no le resta importancia a este trabajo ya que el efecto final merece la pena y tiene una aplicabilidad muy alta. La representación puede tener a partir de ahora un toque mucho más realista con la inclusión de los mapas de reflejos y además esto no influirá mucho ni en lo que ocupa el programa ni en la velocidad de cálculo necesaria para visualizarla.

Caso práctico 12.5: Texturas animadas por transformación.

Objetivo: En este trabajo práctico el objetivo es demostrar cómo se pueden realizar efectos de animación de texturas. El método que vamos a utilizar es el de modificar el modo en que la textura se aplica sobre el modelo 3D. Esto quiere decir que la textura aplicada no cambia, lo que se modifica es el modo de aplicación, es decir, la forma en que "envuelve" al objeto.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Introducción a la aplicación de texturas por transformación.
2. Transformación de escala de texturas.
3. Transformación de giro de texturas.
4. Transformación de la posición de texturas.
5. Modos de aplicación de las texturas a los modelos.

1. Introducción a la aplicación de texturas.

Existe otro método para realizar texturas animadas que supone el cambio de unas texturas por otras, por eso, para especificar este método, hemos añadido la coletilla "por transformación". Para que se aprecien mejor los cambios, esta práctica aporta tres funciones. Primero, que se pueda elegir la textura aplicada. En segundo lugar, que se pueda modificar la transformación en velocidad, en escala y en rotación. Y por último, también podemos modificar el modo de "envolver" al objeto 3D, o sea, si se aplica la textura de forma plana, cilíndrica, etc.

En prácticas anteriores ya hemos estudiado cómo se pueden aplicar las texturas a un modelo 3D. Además, hemos estudiado que se puede cambiar la textura de un modelo mientras se visualiza, es decir, sin tener que parar el programa. En esta práctica se trata de aprender más sobre el tema, de profundizar en la forma en que se puede modificar una misma textura en su modo de aplicación.

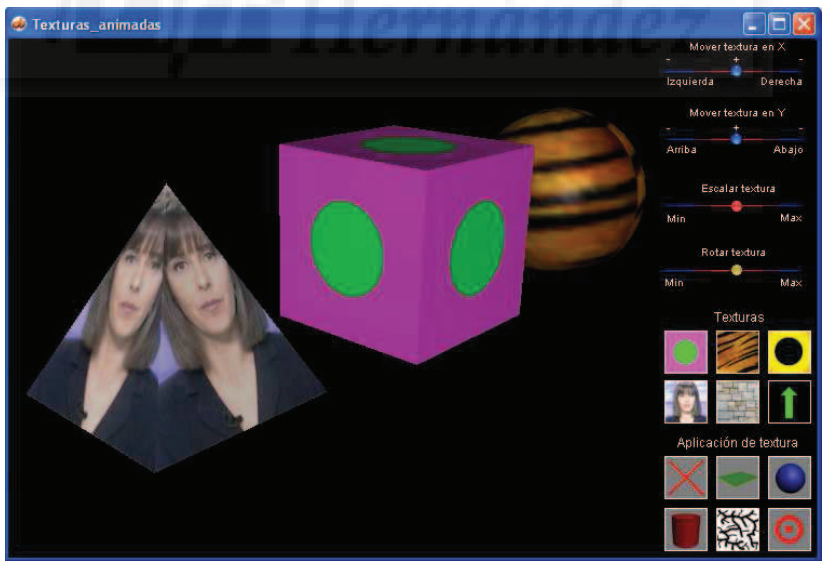


Imagen 5.12.cp.5.1: Vista de la práctica "Texturas animadas por transformación"

En la figura 4.13.1 se puede ver una captura de pantalla de la práctica terminada y en ejecución. Hemos creado tres clases distintas de controles: los de modificación de las texturas, que están en la parte superior, los de elección de la textura, que se encuentran en medio y los de elección del modo de aplicación, que están bajo del todo.

Para comenzar, vamos a recordar cómo se aplica una textura durante la ejecución. El método general de aplicación de una textura a un modelo 3D, independientemente de en qué script convenga más escribirlo, es el siguiente:

1.1. Creación de una nueva textura en nuestro espacio tridimensional a partir de un bitmap incorporado a nuestro castmember. Suponiendo que el miembro se llama "textura_1 y dentro de mi espacio (llamado mundo3D) lo queremos llamar "t1", deberemos escribir:

```
mundo3D.newtexture("t1",#fromcastmember,member("textura_1"))
```

1.2. Para la aplicación de la textura recién creada a un modelo 3D cuyo superficie (o shader) se llama "Default Material 1", escribimos la siguiente sentencia:

```
mundo3D.shader("Default Material 1").texture=mundo3d.texture("t1")
```

En esta práctica hemos utilizado tres modelos y fondo negro, para apreciar mejor las diferencias en los cambios en las texturas como se ve en la imagen 5.12.cp.5.2.

La interacción puede ser de muchas formas, pero hemos decidido que se puedan aplicar texturas distintas a cada uno de los tres modelos, previamente seleccionados al hacer clic sobre ellos, sin embargo, los cambios de transformación, afectarán a todas las texturas por igual para observar mejor su funcionamiento.

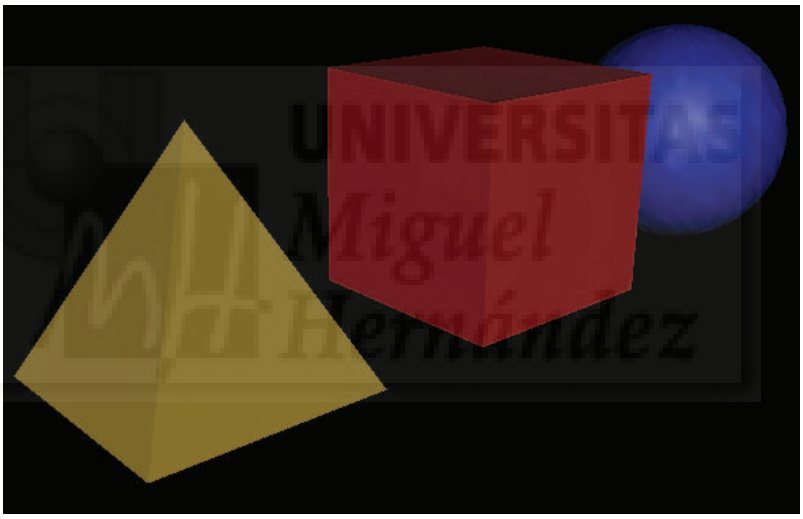


Imagen 5.12.cp.5.2: Modelos utilizados en la práctica sin textura aplicada

2. Transformación de escala de texturas.

Como escribimos en el tema teórico, el escalado de la textura de un modelo va a tener una consecuencia, y es que no se va a corresponder con el tamaño de las caras del modelo. Esto quiere decir que o bien se va a quedar "corta", o sea, no tapará todo el modelo y por tanto se repetirá formando un mosaico, o va a quedar demasiado grande y en consecuencia, no se verá completa, pues habrá parte de la textura que no será representada.

Para controlar interactivamente la escala de la textura, hemos dotado al programa de un control tipo deslizador. Los valores deben ir desde un valor mínimo a un máximo. Es evidente que el valor mínimo no puede ser 0, puesto que esto provocaría un error en tiempo de ejecución, ya que no se puede escalar un bitmap cada vez más pequeño hasta el infinito. El valor máximo también ha de controlarse. Hemos asignado un valor máximo de 2 (el doble de su tamaño original), ya que en la visualización será bastante evidente que se ha "perdido" la mitad de la textura.

La programación del script que controla el deslizador y que hemos llamado “Modificar escala” es el siguiente:

```
global mundo3D
global scalemio
...
//toda la programación ya conocida del controlador y explicado en otras prácticas.
...
on mouseUp me
  valor=sprite(slider).locH
  scalemio= (valor-610) / 63.0
 // como tiene un recorrido hasta X píxeles se divide por el cociente que da máximo 2, o sea, 63
 // tener cuidado y poner en las divisiones los valores con decimales

  if (scalemio=0) then scalemio=.001

 // en el caso de situarse en el extremo mínimo se le aplica una escala mínima pero no 0

  mundo3D.shader("Default Material0").texturetransform.identity()
  mundo3D.shader("Default Material1").texturetransform.identity()
  mundo3D.shader("Default Material2").texturetransform.identity()

 // para cada shader de cada modelo 3D se inicia la transformación desde el principio para
partir     siempre de 0

  mundo3D.shader("Default Material0").texturetransform.scale(scalemio,scalemio,1)
  mundo3D.shader("Default Material1").texturetransform.scale(scalemio,scalemio,1)
  mundo3D.shader("Default Material2").texturetransform.scale(scalemio,scalemio,1)

 // se aplica la escala elegida a las tres texturas de los shaders correspondientes

  Actualizardatos()  // se llama a actualizardatos por si hace falta controlar algún otro parámetro
end mouseup
```

Creemos que los comentarios en el script aclaran su funcionamiento bastante bien. Una vez que se suelta el control deslizador, es decir, en el “mouseup”, se obtiene la posición relativa del control, y por tanto, la cantidad de escala elegida por el usuario, que guardamos en la variable “scalemio”. Con esta información comprobamos si la escala elegida es la mínima, o sea, 0. Si es así la ponemos muy próxima pero no a 0 por las razones antes expuestas.

Antes de escalar, ponemos todos las texturas tal como estaban en un principio, es decir sin cambios. Al ser esta una práctica de demostración, tenemos que poder apreciar bien los cambios que sufren, y esto se dificultaría si no partimos de un punto inicial y reconocible, ya que arrastraríamos los cambios consecutivamente. Por ejemplo, si al principio le aplicamos una textura de 0.2, y luego la hacemos el doble, en realidad debe quedar reducida en un 0.4. Pero esta forma de actuar no es adecuada para el objetivo de esta práctica. Por ello se utiliza la función identity() que hace volver a la textura a sus parámetros iniciales.

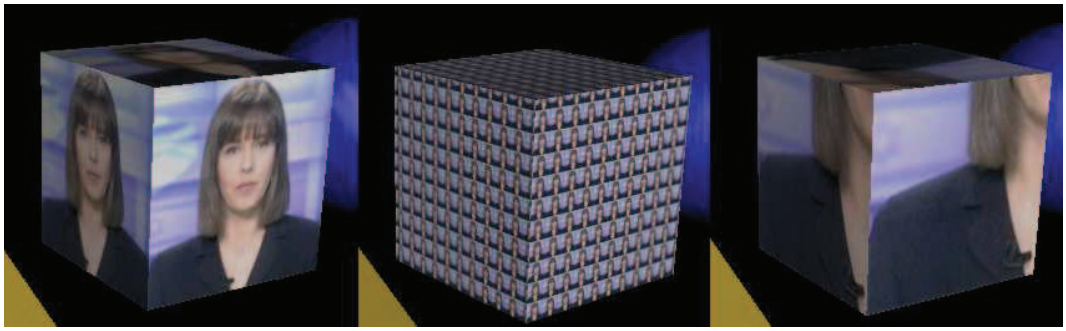


Imagen 5.12.cp.5.3: El modelo con la textura en distintas escalas

Por último, con la función “scale” de “texturetransform” se modifica el tamaño de la textura. Cómo se puede apreciar en el script, escribiremos tres parámetros aunque sólo utilizaremos dos, ya que una escala en Z, o sea, en profundidad no tiene sentido en un objeto 2D como es una textura.

En la imagen anterior se observa cómo cambia la escala de la textura aplicada.

3. Transformación de giro de texturas.

También podemos hacer que las texturas giren. Lo que se realizará es una reposición de la textura pero con un giro que puede ser positivo o negativo (en un sentido o en otro). Esta operación se podrá llevar a cabo de forma independiente con respecto a las demás y por tanto también se podrá superponer su efecto a otros como lo escala o reposición de la textura.

Hemos diseñado un control de tipo deslizador para controlar el giro aplicado a la textura. Como sucedía con el control para realizar el escalado, se han limitado los valores de la rotación aplicada a la textura. Estos valores los hemos denominado mínimo y máximo y aquí, a diferencia de lo que pasaba con la escala, si se puede acumular la rotación, ya que por su propia naturaleza, es de tipo cíclica.

El script que contiene la programación para controlar el giro lo hemos llamado “Modificar giro” y es como sigue:

```
global mundo3D
global angulo
...
//toda la programación ya conocida del controlador y explicado en otras prácticas.
...

on mouseUp me

  valor=sprite(slider).locH
  angulo=(valor-610)

// en este caso se deja el ángulo en su valor original, lo que da un rango de 0 hasta 127 grados
// se aplica el giro  a las tres texturas de los shaders correspondientes

  mundo3D.shader("Default Material0").texturetransform.rotate(0,0,angulo)
  mundo3D.shader("Default Material1").texturetransform.rotate(0,0,angulo)
  mundo3D.shader("Default Material2").texturetransform.rotate(0,0,angulo)
  Actualizardatos()

// se llama a actualizardatos por si hace falta controlar algún otro parámetro

end mouseup
```

Hemos de recordar que este script está vinculado a la aparición en escena (sprite) de un control tipo deslizador determinado. Cuando el usuario lo maneja de un lado a otro (ya que no le está permitido moverlo en vertical) llega un momento en que lo suelta, definiendo así una cantidad deseada de giro que debemos aplicar. Cuando esto sucede, se produce el evento mouseup, al que responde el manejador que se describe arriba. El manejador mouseUp lo que hace es tomar el valor del control en cuestión, y esto lo hace sabiendo cuál es su posición, luego deduce su ángulo al restarle la posición mínima que puede elegir, o sea, 610.

Por último, lo que hacemos es aplicar el giro a la textura. Para ello realizamos la operación texturetransform.rotate sobre el shader que deseemos. Cómo se puede observar en el script, se aplican a los tres shaders a la vez, es decir, sobre los tres modelos. Esta manera de proceder es un poco discutible ya que quizá podríamos aplicarlo al modelo seleccionado previamente, lo cuál, por otra parte, carece de cualquier dificultad de realización. De hecho hay prácticas donde se incluye un script de tipo movie que cuando un modelo es “tocado”, devuelve

el shader que le corresponde, y por tanto, sobre el que habría que aplicar el giro. En esta versión el script quedaría así:

```
mundo3D.shader(shadertocado).texturetransform.rotate(0, 0, angulo)
```

La función “rotate” de “texturetransform” solo utiliza un parámetro que es el ángulo, y en este caso se aplica como un vector, donde el ángulo se incluye en tercera posición. También se podría aplicar previamente a la rotación una función del tipo “identity” para que la rotación no se acumulase y esto lo haríamos escribiendo lo que sigue:

```
mundo3D.shader("Default Material 1").texturetransform.identity()
```

En la imagen siguiente se aprecia cómo cambia la rotación de la textura aplicada.

Imagen 5.12.cp.5.4: El mismo modelo con la textura sin rotar y rotada

4. Transformación de posición de texturas.

Vamos a estudiar un aspecto que puede realizar un efecto realmente llamativo. Basándonos en un método que tiene la propiedad “texturetransform” para posicionar la textura en las coordenadas que deseemos, vamos a realizar una animación de la textura. El método es ejecutar esta función llamada “translate”, pero en lugar de una vez como los anteriores casos, lo ejecutaremos dentro de un bucle que se repite cada vez que la cabeza lectora salga del fotograma y vuelva a entrar.

Es evidente que la velocidad de cambio de posición de la textura dependerá de la velocidad de fotogramas por segundo que muestre Director. Pero en este punto también va a incidir la cantidad de espacio que se va a mover la textura con respecto a su anterior posición, o sea, la cantidad de espacio que se traslade, lo que también incidirá en la velocidad de la animación visualizada.

Por lo tanto, aunque esta operación se va a controlar independientemente, al no ser estática sino dinámica, va a incidir en la forma de percibir las operaciones tanto de giro como de escala. Esto quiere decir que este efecto va a ser más visible que el que provocan las operaciones de escala y giro.

La función “translate” tiene dos parámetros que controlan la posición en el eje X y en el eje Y. En esta práctica hemos realizado dos controles deslizadores independientes para controlar el movimiento en cada uno de estos ejes.

La programación del script para controlar el parámetro de trasladar la textura en el eje X es así:

```
global mundo3D
global traslatex
...
on mouseUp me
  valor=sprite(slider).locH
```

```
  traslatex=(valor-610)/127.0
 // el valor traslatex varía en función de lo que el usuario mueva el control deslizador.
 // se divide por 127 que es la anchura del deslizador para tener un control muy fino de la
traslación de la textura
  Actualizardatos()
end mouseup
```

La programación del script del bucle, que en realidad efectuará la animación es así:

```
global mundo3d
global traslatex
global traslatey
global traslatez

on exitFrame me
  go the frame
  mundo3D.shader("Default Material1").texturetransform.translate(traslatey,traslatex,traslatez)
  mundo3D.shader("Default Material0").texturetransform.translate(traslatey,traslatex,traslatez)
  mundo3D.shader("Default Material2").texturetransform.translate(traslatey,traslatex,traslatez)
end
```

En el script de iniciación del programa se han dado los valores 0 a traslatex, traslatey y traslatez.

En la imagen siguiente se puede observar cómo cambia la rotación de la textura aplicada al cubo. No hace falta explicar que en esta imagen no se puede observar la animación. Aunque se puede apreciar la diferencia en cuanto a cómo cambia la textura con respecto a las anteriores transformaciones. Aquí observamos cómo la textura no cambia haciendo un giro o cambiando en tamaño, sino que se traslada hacia arriba. Si se ejecuta la práctica veremos como se produce una animación al estar escrita esta transformación asociada al mecanismo de bucle.

Debemos tener en cuenta otra cuestión. Aunque la traslación sea en el eje Y, esto controla la dirección, pero no el sentido. Es decir, que la traslación puede ser positiva (en el eje Y, será hacia arriba) o negativa (hacia abajo). Por este motivo es por lo que el controlador asociado tiene en sus extremos dos menos, que indican mínima, o sea, nula traslación, y sin embargo en el centro muestra un + como signo de máxima velocidad. Hacia la izquierda la textura se traslada en sentido ascendente y a la derecha en sentido descendente.

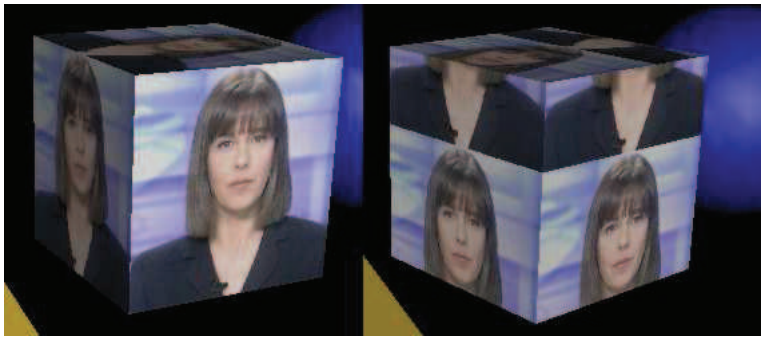


Imagen 5.12.cp.5.5: El mismo modelo con la textura sin trasladar y animada

5. Modos de aplicación de las texturas a los modelos.

En esta práctica hemos aumentado la funcionalidad de la misma para estudiar otra faceta de la aplicación de las texturas sobre los modelos 3D: el mapeado. En la unidad teórica ya estudiamos que el mapeado controla la forma en que la textura es aplicada sobre un objeto.

Hay muchas formas de afrontar un problema como este, o sea, de envolver un objeto. Por ejemplo, si el objeto es en gran medida esférico (como una pelota de fútbol), se puede considerar que la forma más correcta es envolverlo en un papel de regalo enorme que abarque todo el objeto. Si por el contrario el objeto se parece más a un cilindro (como un cigarrillo) se puede intentar usar tres trozos de papel, una para cada una de las caras circulares y luego un gran trozo rectangular para la cara central. Pues bien, tal como ocurre en la vida real, en Director también podemos optar por aplicar las texturas de distintos modos.

Existe una propiedad del shader para los modelos 3D que se llama “texturemode” y que realiza esta función. Esta propiedad toma una serie de valores posibles que hacen que la textura se aplique sobre el modelo de forma diferente. Siempre vamos a tomar el modo por defecto como referencia. Esto tiene una ventaja y es que no hace falta especificar un modo concreto para aplicarla, ya que es el valor por defecto de “texturemode”. La desventaja es que no es muy buen ejemplo didácticamente hablando, ya que el modo depende del modelo en cuestión, es decir, que aparte de la forma en que se visualice, también interviene la geometría del modelo 3D.

Existen siete formas distintas de modos de aplicación, aunque en la práctica y como algunas de ellas no varían para los modelos que nos sirven de referencia, no las hemos usado. Y son las siguientes:

Modo ninguno, none.

mundo3D.shader(“Default Material 1”).texturemode=#none

Imagen 5.12.cp.5.6: El modelo cubo con el modo de textura none

Modo plano, wrapplanar.

mundo3D.shader(“Default Material 1”).texturemode=#wrapplanar

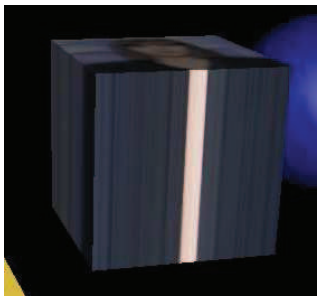


Imagen 5.12.cp.5.7: El modelo cubo con el modo de textura planar

Modo esférico, wrapspherical.

mundo3D.shader(“Default Material 1”).texturemode=#wrapspherical

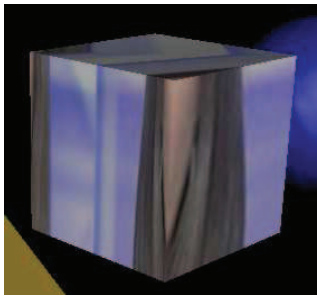


Imagen 5.12.cp.5.8: El modelo cubo con el modo de textura esférico

Modo cilíndrico, wrapcylindrical.

mundo3D.shader(“Default Material 1”).texturemode=#wrapcylindrical

Imagen 5.12.cp.5.9: El modelo cubo con el modo de textura cilíndrico

Modo reflejos, reflection.

mundo3D.shader(“Default Material 1”).texturemode=#reflection

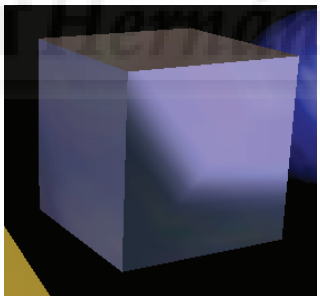


Imagen 5.12.cp.5.10: El modelo cubo con el modo de textura reflection

En este caso también podemos utilizar la función identity() para reiniciar la aplicación de la textura a sus valores iniciales, o sea, los que tiene por defecto, que en lo que incumbe a la forma de envolver la textura, es el valor none. Esta función se escribe como sigue:

mundo3D.shader(shadertocado).texturetransform.identity()

Conclusiones:

Podemos ver que con pequeños scripts se pueden crear efectos especiales. Los cambios que se le aplican a la textura son en esencia sencillos, pero afectan mucho a la visualización de los objetos. Los efectos especiales derivados pueden ser más llamativos si partimos de texturas más sofisticadas y en combinación con canales alfa. Más adelante veremos que existen otros métodos para animar texturas, cómo aplicar más de una textura a la vez al mismo modelo, etc.

Caso práctico 12.6: Texturas animadas por intercambio

Objetivo: En este caso práctico, haremos un ejercicio donde crearemos texturas animadas que aplicaremos sobre los objetos 3D y que podrán ser cambiadas en tiempo de ejecución. Esto significa que el cambio de unas texturas por otras, se hará "al vuelo", esto quiere decir, que tendrá lugar mientras el programa se está ejecutando.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Introducción a las texturas por animación.
2. Creación de Casts.
3. Animación de texturas por programación.
4. Aplicación de los scripts a los botones de interface.

1. Introducción a las texturas por animación.

El interfaz de la práctica que realizaremos se puede observar en la imagen que sigue. Vamos a basarnos en el caso práctico precedente para minimizar esfuerzos. Por ello, la construcción de la escena 3D y la estructura de película son idénticas.

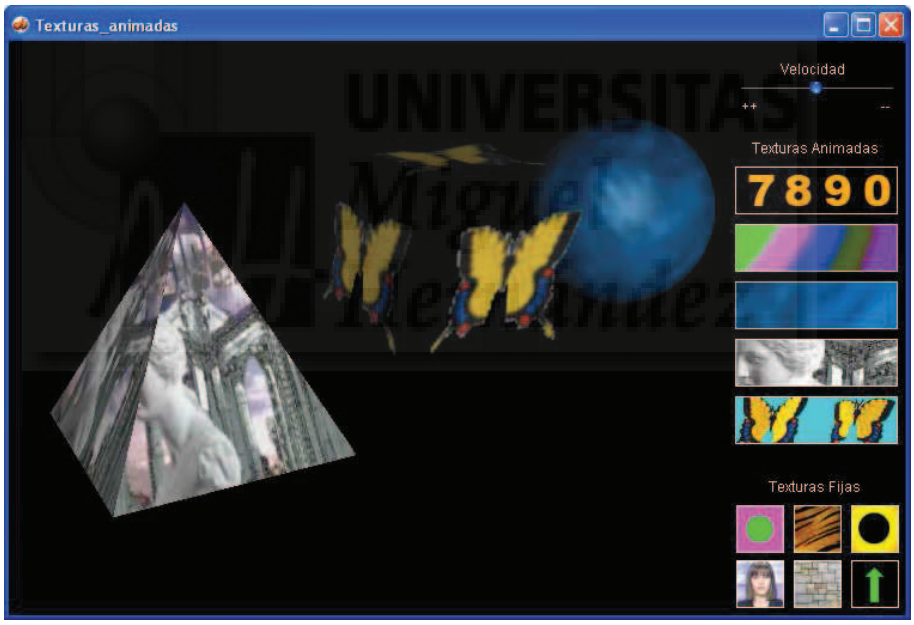


Imagen 5.12.cp.6.1: Vista de la práctica finalizada

Por lo tanto, desde el principio, podemos centrarnos en la creación de un sistema de intercambio entre distintas texturas para el mismo objeto. El mecanismo que "dispara" este cambio, o sea, el que lo produce, será normalmente, una fracción de tiempo. Cuando pase una cantidad dada de tiempo la textura deberá cambiar por otra. También queremos que el usuario tome el mando de la práctica y pueda cambiar tanto la velocidad de las texturas animadas como la propia textura. Además, para que tenga relación con los casos anteriores de texturización, se podrá elegir entre texturas animadas y no animadas. Poder cambiar las texturas de un objeto no debería plantearnos grandes problemas a priori, puesto que esta cuestión ya la hemos resuelto en prácticas anteriores. De hecho, en el caso práctico 9, llamado "Asignación de texturas", realizamos una práctica para poder elegir una textura de entre 10 de ellas para asignarlas a 4 objetos diferentes.

Ya estudiamos en la parte teórica que se nos plantean unas cuestiones y cómo solventarlas, por lo tanto vamos a pasar ya a la realización de las soluciones.

2. Creación de Casts.

En la unidad teórica 12, ya vimos que debemos organizar los bitmaps de las distintas texturas de forma eficiente separándolos en Casts distintos. Estos se pueden nombrar con el nombre que deseemos y en la programación podemos referirnos a un elemento en concreto de un determinado cast para gestionarlo sin problemas.

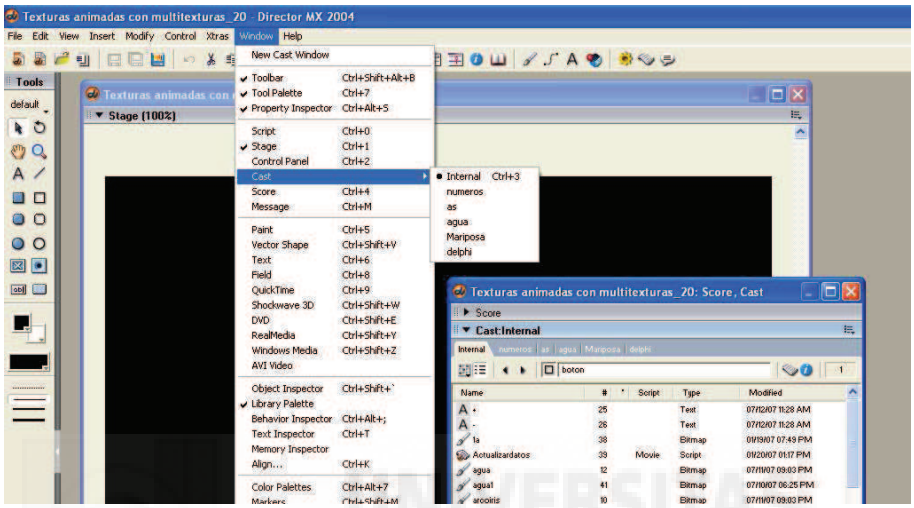


Imagen 5.12.cp.6.2: El menú para crear nuevos Casts

Por tanto lo que vamos a hacer es crear un cast distinto para cada textura animada distinta y así separar unas de otras. Por otra parte, todos los demás bitmaps, esto es, los botones y las texturas fijas, estarán en el cast por defecto que hemos venido usando en todas las prácticas precedentes y que es el llamado “Internal”.

Para crear un cast nuevo debemos fijarnos en la imagen 5.12.cp.6.2, donde se puede apreciar dónde está el menú de cast. Los pasos a seguir son los siguientes:

1.1. Ir al menú Windows de Director y abrir el grupo Cast, o pulsar “ctrl.+3”.

1.2. En el submenú aparecen los cast disponibles, por lo que pulsamos al cast por defecto, llamado “Internal”. Con esto hacemos que aparezca la ventana de Cast como la que se puede apreciar en la imagen 5.12.cp.6.3.

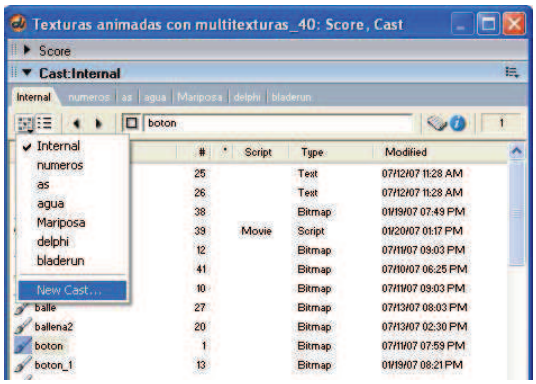


Imagen 5.12.cp.6.3: Ventana de Cast

1.3. Ahora debemos hacer clic sobre el primer botón por la izquierda llamado “Choose Cast” y en el menú que aparece (y que se reproduce en la imagen 5.12.cp.6.3.) elegir la opción “New Cast”. Esto hará que aparezca la ventana de creación de un nuevo cast.

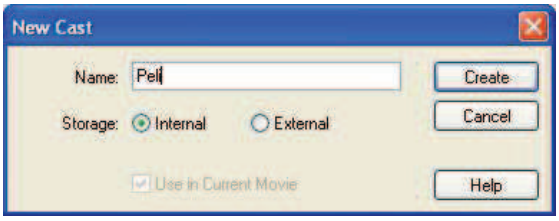


Imagen 5.12.cp.6.4: Ventana para crear nuevos Casts

1.4. En la ventana “New Cast” que se puede observar en la imagen 5.12.cp.6.4. Introducimos el nombre que deseemos, por ejemplo “Peli” y en “Storage”, es decir, almacenamiento, elegimos “Internal”, para evitar problemas posteriores con la publicación del trabajo.

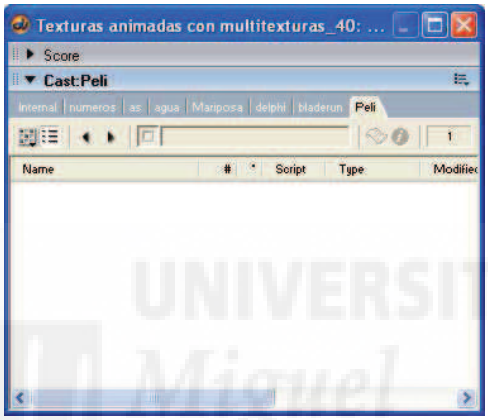


Imagen 5.12.cp.6.5: El nuevo Cast creado

1.5. Finalmente pulsamos el botón “Create”. El resultado será un nuevo cast llamado “Peli” que estará inicialmente vacío, pero que podemos utilizar para importar nuevos bitmaps que representen una textura animada. En la ventana Cast, aparecerá una nueva ficha llamada Peli como la que se puede observar en la imagen 5.12.cp.6.5.

También podríamos utilizar este método para separar los distintos elementos de un programa multimedia como sonidos, videos, etc., pero este no es el caso que nos ocupa.

3. Animación de texturas por programación.

Quizá la primera cuestión que debamos resolver es cómo podemos hacer que la textura de un objeto cambie cuando pase un intervalo de tiempo. Además está el problema de automatizar este cambio. Se debe cambiar de textura a la siguiente y cuando ya no queden más texturas, volver a la primera.

Para cambiar de textura podemos usar una combinación de la variable member que tienen las librerías de contenido (o cast) y que devuelve un determinado componente. El código quedaría así:

```
mundo3d.texture("textura").member=member num of castlib "texturanimada1"
```

Para controlar que no nos salgamos de componentes podemos utilizar el mecanismo estudiado en la unidad teórica 12 y es que el número de componente a solicitar al cast esté dentro de los límites de estos. Como sabemos el número de bitmaps que conforman una determinada textura

animada, podemos hacer que el número sea el resto de la división (el operador mod) del contador por el número de bitmaps de la textura. El código asociado a esta idea es:

```
num=(contador mod 9)+1                    // esto debe escribirse antes del código anterior
```

Una cuestión que debemos resolver es como controlar la velocidad en el intercambio de unos bitmaps por otros dentro de una textura animada. El elemento que sirve de interface con el usuario puede ser un botón deslizante que hemos venido utilizando constantemente en casos prácticos anteriores. Debemos interpretar la posición del botón como una cantidad de velocidad relativa, siendo la más alta aquella que hace que se produzca un intercambio cada vez que se produce un evento “exitFrame”. El código sería algo así:

```
On exitFrame me
  velocidad = velocidad +1
  if (boton=1) and (velocidad mod veloc = 0) then
…
end
```

Donde veloc sería el valor devuelto por el botón deslizante.

Por último debemos resolver la cuestión de cómo mantener la funcionalidad de las texturas fijas. Para ello nos fijaremos en el manejador de pulsación de sus propios botones y no modificaremos el script “Bucle”. Esto se resuelve fácilmente al tener acceso al shader del objeto elegido por el usuario mediante el script “queshader”. Lo único a destacar es que debemos poner la textura fija aplicada con transparencia. Por tanto el código del botón al elegir una textura fija contendrá estas dos líneas:

```
mundo3D.shader(shadertocado).texture=mundo3d.texture("t2")
mundo3D.shader(shadertocado).transparent=true
```

4. Aplicación de los scripts a los botones de interface.

Para que todo lo expuesto quede más claro, seguidamente se escribirán los códigos con comentarios de los scripts más importantes de este caso práctico:

Script “Bucle” asociado al frame donde se sitúa el mundo 3D:

```
global mundo3d

global contador
global velocidad
global veloc
global num
global shadertocado
global boton

On exitFrame me

velocidad= velocidad +1

  if (boton=1) and (velocidad mod veloc =0) then          // para cada botón de textura animada
    contador=contador+1
    num=(contador mod 9)+1
    queshader()
    mundo3d.texture("nueva").member=member num of castlib "numeros"
    mundo3D.shader(shadertocado).texture= mundo3d.texture("nueva")
  end if

  if (boton=2) and (velocidad mod veloc =0) then
```

…

end

Script de botones de texturas animadas:

```
global mundo3d
global boton

on mouseDown
  boton=1
end
```

Script de botones de texturas fijas:

```
global mundo3d
global shadertocado
global boton

on mouseDown

  boton= 10
  queshader()
  mundo3D.shader(shadertocado).texture=mundo3d.texture("t2")
  mundo3D.shader(shadertocado).transparent=true

end
```

Script “Programación” asociado a la instancia del mundo 3D:

```
on beginSprite me
  pSprite = sprite(me.spriteNum)
  mundo3D.newtexture("t1",#fromcastmember,member("y1"))
```

Script asociado con el botón deslizante “Poner Velocidad”:

```
on mouseUp me
  valor=sprite(slider).locH
  veloc=(valor-612) / 6.35                    //esto hace que veloc tome un valor entre 20
  if (veloc=0) then veloc = 1
end mouseup
```

Conclusiones:

Como conclusiones podemos decir que las texturas animadas por intercambio tiene una solución relativamente sencilla. Los códigos asociados a esta solución son bastante fáciles de entender incluso para programadores no experimentados. Todo esto cambiaría mucho se pretendemos controlar todos los aspectos susceptibles de ello en el intercambio, aparte de la velocidad, como podría ser parar, ir hacia adelante o hacia atrás, etc.

Lo más interesante de este caso práctico es que abre las puertas a la experimentación con pequeños vídeos y otras texturas prerepresentadas en programas como MAX para la creación de efectos especiales. Si a esto unimos la capacidad de ejecutar sonido, podríamos integrar vídeos sonoros en la superficie de objetos tridimensionales y esto podría dar mucho juego para una creación artística.

Unidad 13: Animación en Director.

Introducción teórica:

1. Introducción a las animaciones en Director: fotogramas y huesos.
2. La importación de animaciones desde MAX.
3. Control de la animación de fotogramas con Lingo: el modificador Keyframer.
4. Control de la animación de huesos con Lingo: el modificador Bones.

Casos prácticos:

13.1. Animación fotograma a fotograma.

1. Introducción a las animaciones en Director: fotogramas y huesos.

En este tema queremos introducirnos en las técnicas de animación de Director. Existen muchas técnicas de animación hoy en día. La más sencilla y antigua es la animación fotograma a fotograma. Estas animaciones se caracterizan porque los modelos varían sus características propias (posición, rotación y escala) a lo largo de los fotogramas que componen la animación. Por ejemplo, si un modelo se traslada de posición de un fotograma a otro, se produce una animación en el tiempo. Este tipo de animación es la que se utilizaba en las primeras películas de dibujos animados. Para ver detalladamente cómo crear la o las animaciones podemos estudiar el caso práctico asociado a este tema: 13.1. Animación fotograma a fotograma.

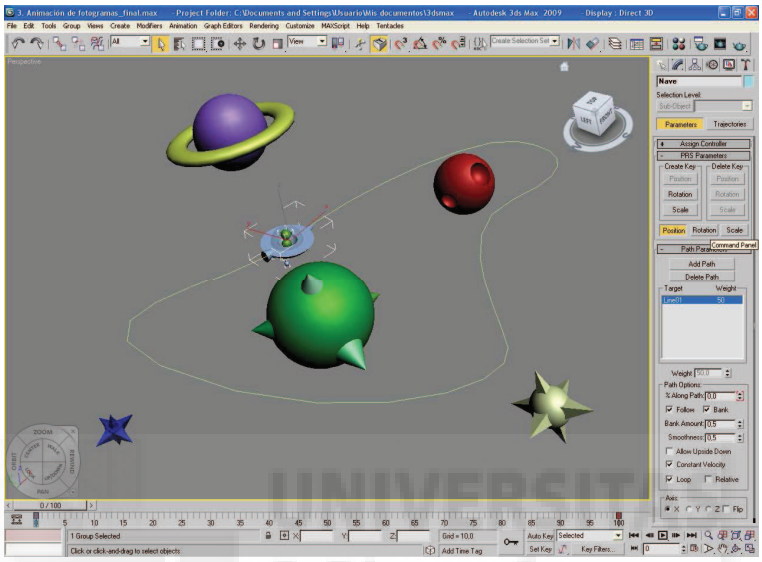


Imagen 5.13.1: Creación de animación de fotogramas en MAX

Otro tipo de animaciones muy utilizadas hoy en día son las animaciones de huesos. Este nombre, no implica que sólo se pueda utilizar para modelar esqueletos, sino cualquier estructura jerárquica de elementos móviles. Las animaciones de huesos son las que se utilizan actualmente en el cine para crear animales, robots y personas con movimientos realistas.

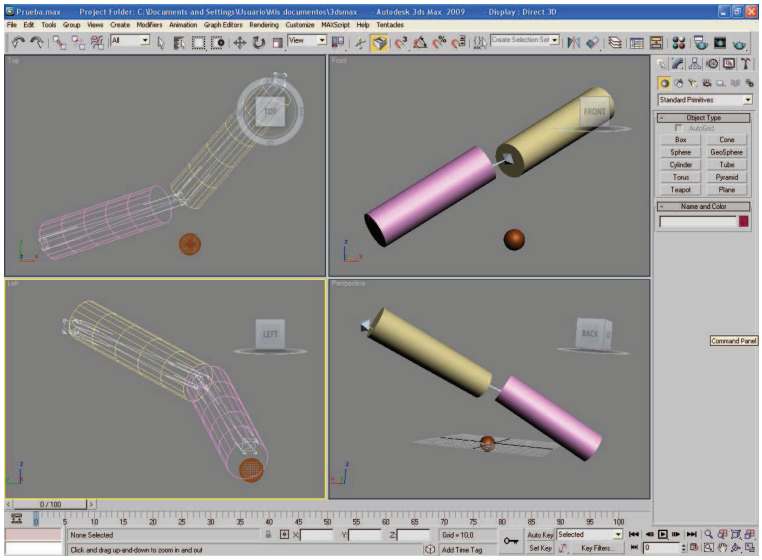


Imagen 5.13.2: Creación de animación de huesos en MAX

Estos dos tipos de animaciones serán más que suficientes para crear los proyectos artísticos que nos incumben.

Las animaciones en sí mismas, tanto la de fotogramas como la de huesos, las crearemos en otros programas de modelado (en nuestro caso en 3D MAX), y se incrustan al exportar el fichero en formato W3D. Nuestro objetivo es controlar una animación importada, para posteriormente, ser capaces de simultanear más de una animación.

2. La importación de animaciones desde MAX.

La importación de animaciones desde MAX tiene dificultad, ya que en la mayoría de los recursos a exportar basta con marcar la casilla correspondiente y en el caso de las animaciones no, por lo que vamos a estudiar los pasos en los que tenemos que tener cuidado.

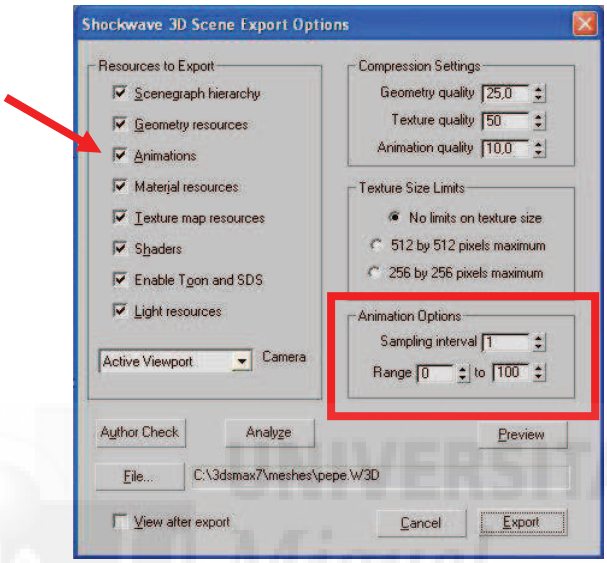


Imagen 5.13.3: Exportador de MAX a Shockwave 3D

Cuando ejecutamos el comando Exportar y elegimos Shockwave 3D, aparece la caja de diálogo de exportación. Lo primero que debemos hacer es marcar las animaciones como recurso a exportar tal como se visualiza en la imagen anterior señalado por una flecha.

Debemos comprobar en las “Animation Options” que cada fotograma de la animación de MAX se va a corresponder con un fotograma de la animación en Director mediante el control “Sampling interval".

También podemos seleccionar el rango de la animación a exportar mediante la opción “Range”. Por lo tanto, en la mayoría de los casos debemos exportar el rango total de la animación para que cuando abramos el archivo en Director nos encontremos la animación completa.

3. Control de la animación de fotogramas con Lingo: el modificador Keyframer.

Lo primero que tenemos que aclarar es que las animaciones o movimientos de fotogramas no se pueden crear con Lingo aunque si se puede modificar su ejecución con este lenguaje. Por lo tanto, para crear animaciones de fotogramas requerimos siempre de otro software.

Para poder crear un objeto animado, los programas de animación le asignan una propiedad más a ese objeto: la animación. Por ejemplo, todos los modelos tienen una serie de polígonos que definen su geometría. También tienen unos materiales que definen su aspecto. Pero no todos los modelos están animados. Si un modelo contiene una animación se le debe añadir esa información que la definirá. 3D Studio MAX es un buen programa para crear estas animaciones y por lo tanto, para añadir esa información a los objetos.
Cuando un objeto con animación es importado desde MAX a Director, los datos de la animación son convertidos a un modificador llamado “Keyframer” que es añadido como un

modificador más al objeto en cuestión. “Keyframer” se utilizará para controlar el movimiento de fotogramas del modelo.

Una vez importada la animación a Shockwave 3D de Director utilizaremos Lingo junto con el modificador Keyframe para controlar la animación. Por lo que tenemos que tener clara la interface que le vamos a presentar al espectador.

Supongamos que estamos presentando un trabajo dónde el espectador puede iniciar la animación de un objeto 3D. Es de suponer que le demos la mayor libertad de acción posible. Si esto es así, podemos pensar en utilizar en nuestra animación los mismos controles que existen en un aparato reproductor de vídeo normal. Por ejemplo, podríamos implementar botones de play, pause, etc. Por último, también podríamos darle libertad al espectador para que manipule la velocidad de la ejecución de la animación. Esto hará que tengamos los siguientes botones y su programación correspondiente:

Botón “Iniciar”: botón que ejecuta la animación desde su origen. Su funcionalidad se puede programar como sigue:

```
global mundo3D  -- la referencia interna de nuestra escena 3D

on mouseDown
  mundo3D.model("Nombre_modelo").keyframeplayer.currenttime=0
end
```

Este código es fácil de entender: primero se nombra la variable de referencia de la escena tridimensional, en este caso “mundo3D” que es de tipo global, o sea, accesible desde este pequeño script.

Utilizamos el disparador mouseDown, para que cuando se produzca un clic del botón izquierdo del ratón se ejecuten las sentencias que escribamos en su interior.

La propiedad “currenttime” del keyframe controla la reproducción del movimiento en el tiempo. “Currenttime” se mide en milésimas de segundo, por lo tanto al ponerlo a 0, le estamos diciendo que se vuelva a iniciar.

La propiedad “currenttime” es muy flexible ya que nos puede permitir crear un deslizador para que el espectador pueda llevar la ejecución al fotograma que desee, como sucede con los visualizadores de vídeo por Internet o con Windows Media Player.

Botón “Play”: hace que la animación se ejecute desde donde esté situada en un momento dado. En la ventana “Script” del behavior asociado a esto botón introduciremos el siguiente código:

```
global mundo3D  -- la referencia interna de nuestra escena 3D

on mouseDown
  mundo3D.model("Nombre_modelo ").keyframeplayer.play()
end
```

La explicación es muy obvia: existe una función del keframeplayer que directamente hace lo que pretendemos, es decir, que la animación se ejecute.

Botón “Pause”: botón que sirve para detener la animación. Seguirá el mismo proceso que el anterior pero su programación usará la función pause y será la siguiente:

```
global mundo3D  -- la referencia interna de nuestra escena 3D

on mouseDown
  mundo3D.model("Nombre_modelo ").keyframeplayer.pause()
end
```

Botón de “cambio de velocidad”: sirve para asignar una velocidad dentro de un rango y por lo tanto, para modificarla. Si queremos que cambie la velocidad de la animación a una velocidad de, supongamos, el doble de la normal quedaría como sigue:

```
global mundo3D  -- la referencia interna de nuestra escena 3D

on mouseDown
  mundo3D.model("Nombre_modelo ").keyframeplayer.playrate=2
end
```

Esta sentencia, no ejecuta una función, como se puede observar, ya que no utiliza paréntesis, en realidad es una propiedad: “playrate”. En este caso vale 2, porque queremos que la nueva velocidad de ejecución sea el doble de lo normal, pero puede tener los siguientes valores:

- Velocidad mayor de la normal hacia delante: tendrá valores mayores que 1.
- Velocidad menor de la normal hacia delante: valores entre 0 y 1.
- Velocidad normal hacia atrás: valor –1.
- Velocidad menor de lo normal hacia atrás: valores entre 0 y –1.
- Velocidad mayor de lo normal hacia atrás: valores menores que –1.

En la imagen de abajo se visualiza cómo queda el trabajo realizado en el caso práctico que ilustra esta unidad. El interfaz se sitúa abajo y cumple exhaustivamente los requerimientos expuestos salvo que el botón de control de velocidad lo hemos sustituido por un botón deslizador con el fin de que puede ejecutarse de forma interactiva.

Imagen 5.13.4: Interfaz típico de una escena 3D animada

4. Control de la animación de huesos con Lingo: el modificador Bones.

Como hemos escrito anteriormente, las animaciones de huesos se utilizan mucho en la actualidad para crear el movimiento tanto de seres vivos como de estructuras en los que

subyace un esqueleto o una estructura jerárquica de elementos móviles como puede ser el movimiento de una persona o de una grúa.

Al igual que con las animaciones de fotogramas, la animación de huesos no se pueden crear con Lingo. Debemos crearlas con un software especializado como MAX y luego exportarlas. La estructura esquelética se añadirá al modelo en cuestión y tendrá una vinculación directa con su geometría, puesto que los cambios en el "esqueleto" modificarán la geometría u otras variables como su posición, giro y escala, de la misma forma que hace un esqueleto con la masa muscular que lo rodea.

Existen dos formas de controlar una animación de huesos: mediante cinemática inversa o directa.

La cinemática directa supone que el movimiento del esqueleto (y por tanto de la geometría que lo rodea) se realiza directamente. Por ejemplo, si queremos mover un dedo, el animador moverá los huesos del hombro, luego el del antebrazo, luego el del brazo, luego el de la muñeca y por último el del dedo. Su configuración previa es rápida ya que solamente supone definir los huesos y la geometría correspondiente, pero por otra parte, la animación se hace tediosa y difícil.

La cinemática inversa supone que el movimiento se realiza automáticamente moviendo solamente la parte final, situándose el resto de elementos jerárquicos por sí mismos. Por ejemplo, para mover un dedo, moveríamos solamente el dedo al sitio deseado y el software con ik (inverse kinematic) realizaría el trabajo. Su configuración es muy lenta, ya que el animador previamente tiene que definir los ángulos de giro posibles de cada articulación para que se adapten a la anatomía real de un brazo humano (por ejemplo, no pueden girar 180º sin romperse), pero la animación en sí, es mucho más rápida. Afortunadamente, para esqueletos muy conocidos como el humano y en general cualquier bípedo, MAX incluye animaciones de huesos con cinemática inversa incluida como la herramienta "huesos" o el conocido "Character Studio"

Cuando creamos una jerarquía de huesos en MAX sea con huesos o con "Character Studio" debemos tener algunas precauciones antes de exportar esta jerarquía a Director. Las reglas que debemos seguir son:

1) No agrupar la jerarquía.

2) No poner nombres no convencionales a los huesos o no se reconocerán.

3) Si utilizamos el modificador "Físico" para unir la geometría a los huesos usando "Character Studio", este modificador debe ser el último que se utilice, sino no se reconocerá.

Una vez exportado, el fichero .W3D contendrá la jerarquía de huesos y sus movimientos asociados, todo lo cual se podrá controlar desde Director con Lingo mediante el modificador "Bonesplayer".

Bonesplayer a diferencia de Keyframer, controlará la ejecución del conjunto de las animaciones asignadas a una jerarquía de huesos. Sus funciones más representativas son para controlar esta lista de animaciones como Play() para ejecutar una animación, Playnext() para realizar la animación siguiente en la lista o Removelast() para borrar la última animación realizada.

Para entender esto mejor, podemos imaginar que hemos realizado un personaje con "Character Studio" en MAX y le hemos dotado de una serie de animaciones como caminar, correr, saltar, etc. Una vez exportado a Director, podremos utilizar Bonesplayer mediante Lingo para dar el control de estos movimientos al usuario.

Como apunte final, conviene resaltar que en el caso de un personaje con muchos movimientos deberíamos exportar el personaje en un archivo W3D y luego un archivo W3D para cada uno de los movimientos independientes. Luego, en Director utilizaríamos el comando "Loadfile" para ponerlos todos juntos y Bonesplayer para configurar la lista de animaciones y ejecutarlas.

Caso práctico 13.1: Animación fotograma a fotograma.

Objetivo: realizar una animación fotograma a fotograma. Esta técnica es muy útil para los proyectos artísticos que nos ocupan. La animación en sí misma la crearemos en MAX y veremos como exportarla al fichero de Director, que nos permitirá tener un control total sobre esta animación mediante Lingo. También crearemos un interface para que el usuario controle la animación.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Crear un modelo con animación fotograma a fotograma.
2. Exportar el fichero a w3d incluidos los datos de la animación.
3. Crear los botones de acción.
4. Cómo controlar el modificador Keyframeplayer mediante la programación.

1. Crear un modelo con animación fotograma a fotograma.

La animación que vamos a tomar como ejemplo para adentrarnos en este tipo de trabajos es de las más sencillas que podemos construir. Se trata de un objeto 3D al que asignamos un recorrido que tendrá lugar en un determinado tiempo, tras lo cual comienza de nuevo.

1.1. Construcción del móvil: una nave extraterrestre.

Para ello vamos a construir un pequeño modelo de una nave espacial con extraterrestre incluido. Aunque parezca una frivolidad por la cantidad de memoria que gastamos, si mantenemos la norma de usar primitivas, los modelos no generarán archivos muy grandes. También le añadimos un “morro” y un “motor” en cola con el fin de ver la orientación que sigue el móvil. Para modelarlo, hemos utilizado sobre todo esferas o hemisferios y luego como en el caso de la cabeza del extraterrestre o sus ojos, los hemos achatado, haciendo una escala en un solo eje, es decir, no uniforme.

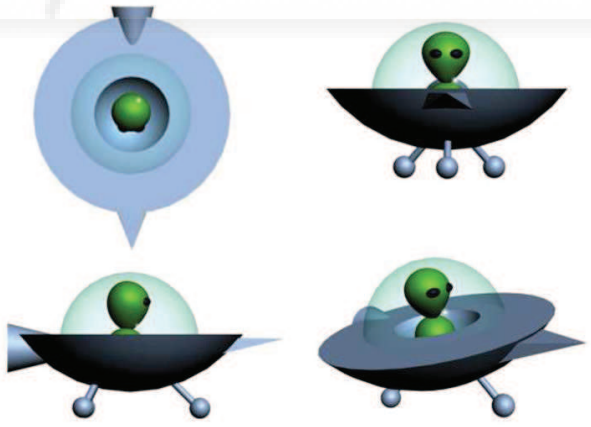


Imagen 5.13.cp.1.1: Vistas de la nave espacial

1.2. Construcción del recorrido.

Para modelar el recorrido hemos realizado los siguientes pasos:

1.2.1. Ir en la ficha Create > Shapes > Splines > Line.

1.2.2. En la vista superior he realizado una forma arriñonada. Lo más importante es que sea una curva cerrada para que el móvil pueda realizar un circuito continuo.

1.2.3. Con la línea seleccionada, vamos al panel Modificar > Selección > Vértices.

1.2.4. En la vista frontal hemos ido seleccionando puntos y modificándolos en altura con la herramienta mover para proporcionar al recorrido una verdadera tridimensionalidad.

El recorrido queda como se muestra en la imagen 5.13.cp.1.2.

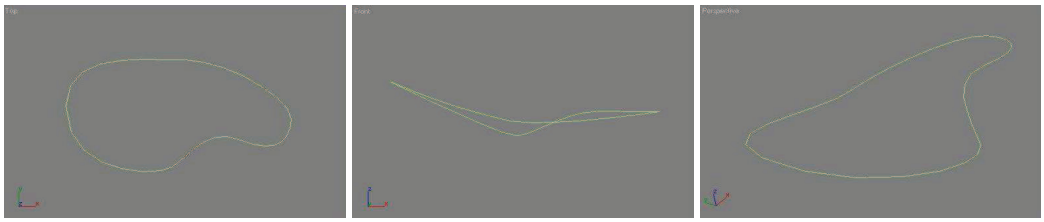


Imagen 5.13.cp.1.2: Vista superior, frontal, y en perspectiva del recorrido

1.3. Creación del “cosmos”.

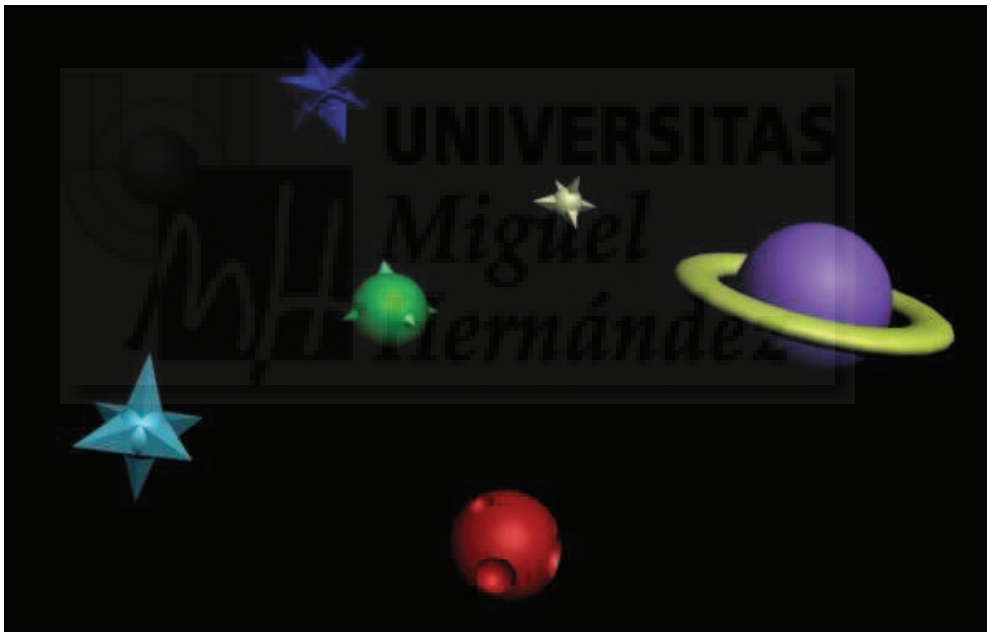


Imagen 5.13.cp.1.3: Vista en perspectiva del cosmos que recorre nuestro móvil

1.3.1. Para que el espectador pueda apreciar que el móvil se mueve en tres dimensiones hemos creado objetos que se interponen entre la vista de este y el móvil. Los objetos son unas estrellas y planetas con apariencia de dibujos animados. Estos modelos los hemos realizado con primitivas de tipo esferas y pirámides y operaciones booleanas.

1.4. Hacer que el móvil siga el recorrido.

1.4.1. Primero tenemos que seleccionar el grupo “Nave”, o sea, el móvil. Luego accedemos al panel Motion y hacemos clic sobre “Position” dentro de “Assign Controller como se muestra en la imagen 5.13.cp.1.4. Es evidente que tenemos la posibilidad de modificar otros parámetros como la rotación y la escala.

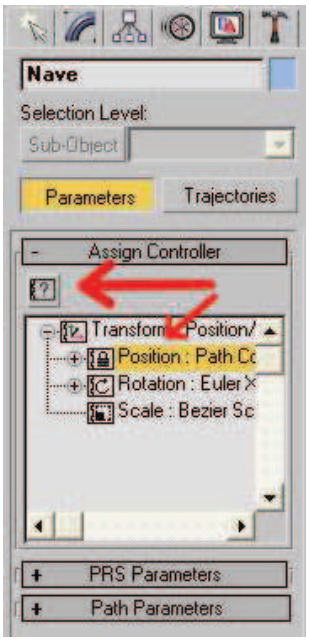


Imagen 5.13.cp.1.4: Asignación de un controlador de posición al móvil

1.4.2. Posteriormente hacemos clic en el botón con un signo de interrogación que se encuentra en la parte superior tal y como muestra la imagen anterior. Esta acción abrirá la caja de diálogo "Assign Position Controller" que se puede observar en la imagen que sigue.

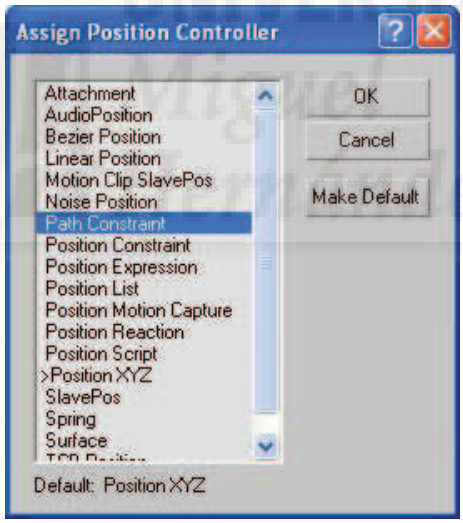


Imagen 5.13.cp.1.5: "Path constraint" como controlador de posición

1.4.3. En esta caja de diálogo, elegimos la opción "Path constraint", es decir, "restringir a ruta". Esto hace que el móvil siga una determinada curva durante el tiempo que dure su animación. Como en nuestro caso no hemos variado los valores por defecto que tienen los archivos en MAX, la animación estará compuesta de 100 frames, que con un valor de 30 frames (fotogramas) por segundo, hace que nuestra animación tenga una duración de 3 segundos y 33 décimas.

1.4.4. Si hacemos una prueba de la animación (pulsando el botón de play a la derecha de la línea de tiempo) comprobaremos que efectivamente el móvil sigue la trayectoria fijada. Pero lo hace de una forma muy poco natural ya que los tramos más curvos los realiza sin girar, o sea, no "toma la curva". Para solucionar esto ponemos el valor "Follow" activo. Este parámetro se encuentra en las "Path Options" como muestra la imagen siguiente.

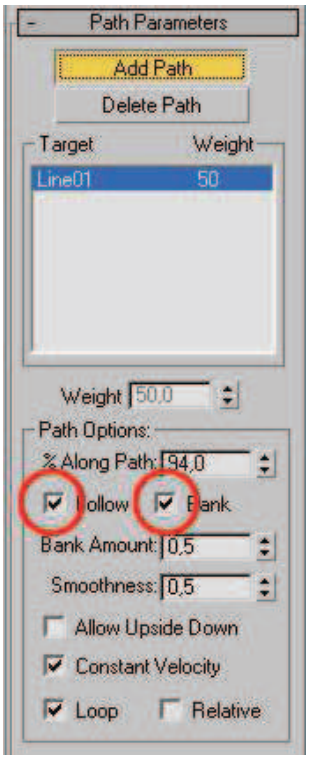


Imagen 5.13.cp.1.6: Asignación de un recorrido y variables relacionadas

1.4.5. Volvemos a probar nuestra animación y podemos observar que sobre todo en las curvas, es muy rígido. Para solucionar el problema utilizamos el parámetro “Bank”. Esta variable produce un “balanceo” horizontal cuando sale de una curva. En nuestro caso lo hemos puesto a valor activo para exagerar más el movimiento y darle un aspecto más parecido al de los dibujos animados. Debajo de este valor se puede introducir un valor para hacerlo mayor o menor. Nosotros hemos utilizado el valor por defecto.

2. Exportar el fichero a w3d incluidos los datos de la animación.

2.1. Ejecutamos Archivo > Exportar > y en tipo de archivo elegimos Shockwave 3D scene.

2.2. En la caja de diálogo de exportación que aparece debemos tener en cuenta sobre todo tres aspectos. El más importante es marcar las animaciones como recurso a exportar.

2.3. También tenemos que comprobar en las “Animation Options” el tener un fotograma de la animación en Shockwave 3D por cada uno de los que tenemos en MAX activando la opción “Sampling intervalo”.

2.4. Por último, tenemos que elegir exportar el rango de la animación que deseemos con la opción de “Range”. Es evidente que si deseamos exportar solo una parte de los frames de MAX, se puede utilizar esta opción para indicarlo.

3. Crear los botones de acción.

Lo primero que tenemos que considerar es que el control de la animación en Director lo vamos a realizar pulsando una serie de botones. Las funciones que queremos crear para controlar la animación son: parar, continuar, comenzar desde el principio y cambiar la velocidad.

Estos botones van a interactuar con la animación importada del móvil. Para facilitar el trabajo el móvil es una serie de objetos agrupados y nombrados a tal efecto. En nuestro caso hemos

llamado al móvil como “Nave”. La operación de agrupar y nombrar al móvil lo hemos realizado en MAX, ya que también se importa sin problemas a Director.

En la figura siguiente se puede ver como queda el trabajo final. Lo primero que puede llamarnos la atención es que existen tres botones “normales” y otro que se desliza solo en el eje horizontal y entre unos límites determinados. Los botones de la izquierda controlan el comienzo de la animación, el continuar y el parar. El botón deslizador controlará la velocidad del móvil. Este botón es muy útil para que el usuario interaccione con el programa e introduzca un valor entre un rango limitado. En nuestro caso la velocidad mínima la ponemos casi a 0 (no parado del todo, ya que para esto tenemos un botón propio) y una velocidad máxima 5 veces superior a la normal. La velocidad normal es la inicial, es decir, la misma que la importada desde MAX, o sea, que es de una vuelta cada 3,33 segundos.

Imagen 5.13.cp.1.7: El ejemplo de animación en ejecución

Como se puede observar los botones utilizados tiene la misma estética que los de Windows Media Player. En este caso, la interface no es la más adecuada, ya que, normalmente, cuando el botón de “play” es pulsado pasa a ser el de “pause”, y viceversa, con lo que la interfaz es más práctica, ya que no aparece un botón hasta que no se puede utilizar y por otra parte no ocupa espacio innecesario. De todas formas, hemos pensado que como no es un caso práctico de interface sino de animación, conviene no centrar la programación en los botones, sino en el control de la animación. Por tanto los botones “normales” los realizaremos como otros ya utilizados.

3.1. Hacemos clic en Archivo > Importar y seleccionamos las imágenes de los cuatro botones. Abrimos la ventana de Cast y veremos que disponemos de 4 castmembers con los mismos nombres que los ficheros de la imagen de la que proceden.

3.2. Los incorporamos a la película de Director. Para ello, hace falta modificarla para hacerla más alta. Para esto vamos a la ventana Property Inspector > Movie > y en el cuadro “Stage Size” escribimos la anchura y la altura que deseemos en píxeles.

3.3. Para llevarlos a la película debemos arrastrar el castmember involucrado hasta una de las pistas de la ventana Score. Creamos así una sprite, es decir, una aparición de un castmember en la película durante n fotogramas. El problema es que debemos hacer que este sprite dure solo un fotograma, que es en el que está el mundo 3D. Este fotograma es el que llamamos

"Principal". Para modificar la duración de los sprites basta con arrastrar sus extremos hasta el fotograma que queramos.

3.5. Repetiremos la operación con los tres botones que faltan.

4. Cómo controlar el modificador Keyframeplayer mediante la programación.

Vamos a programar cada uno de los botones de acción que ya debemos tener disponibles en nuestra película. Estos botones se han diseñado para tener un comportamiento independiente del resto de botones, lo cual simplifica mucho la escritura en Lingo.

4.1. El botón "Iniciar" tendrá una programación como sigue:

```
global mundo3D

on mouseDown
  mundo3D.model("Nave").keyframeplayer.currenttime=0
end
```

La propiedad currenttime controla la reproducción del movimiento en el tiempo. La unidad de medida son las milésimas de segundo, por lo tanto al ponerlo a 0, le decimos que debe volverse a iniciar. También nos abre la posibilidad de crear un deslizador para controlar la ejecución de la animación al estilo de Windows Media Player, donde se pueda visualizar la ejecución en el tiempo y si es interactivo, recomenzar desde donde queramos.

4.2. Seleccionar el botón Play en la ventana Castmember. Para programar su función, es necesario escribir un script y asignárselo al sprite del botón. Por eso una vez seleccionado vamos a la ventana Code: Behavior Inspector y pulsamos sobre el signo "+", luego sobre "New Behavior" y en la ventana que sigue le ponemos un nombre, por ejemplo: play. Aparecerá como un castmember más, pero con el icono de una rueda de engranaje amarilla.

Pulsar sobre el botón "Cast Member Script"  que se encuentra en la parte superior de la lista de miembros.

En la ventana "Script" introduciremos el siguiente código:

```
global mundo3D

on mouseDown
  mundo3D.model("Nave").keyframeplayer.play()
end
```

Este código es bastante sencillo: primero se nombra la variable mundo3D que es de tipo global, es decir, accesible desde cualquier script del proyecto. Esta variable contiene el acceso a la escena tridimensional.

Luego utilizamos el handler mouseDown, o sea, que cuando se produzca un clic del botón izquierdo del ratón se ejecutarán las sentencias que escribamos entre el "on" y el "end".

En este caso escribimos mundo3D.model("Nave").keygrameplayer.play(), esto significa que accediendo a la escena 3D y luego al modelo llamado "nave", utilizamos su propiedad keyframplayer, que es un modificador que contiene datos de movimiento. Por último, le estamos diciendo que debe ejecutar el subprograma play. Este programa hace justo lo que queremos, es decir, hace que el movimiento se ejecute.

4.3. El botón de "Parar" seguirá el mismo proceso que el anterior pero su programación será la siguiente:

```
global mundo3D

on mouseDown
```

```
  mundo3D.model("Nave").keyframeplayer.pause()
end
```

4.4. El botón deslizador de “Cambio de velocidad” ya hemos comentado que es más complicado. Por lo tanto, y con fines didácticos, vamos a explicar cómo lo programaríamos si fuera un botón como los otros, es decir, que cambiase la animación a una velocidad de, supongamos, el doble de la normal. Es evidente que en el caso del botón deslizador, el valor elegido está entre un conjunto de valores, y por tanto no es fijo. Esto quedaría como sigue:

```
global mundo3D

on mouseDown
  mundo3D.model("Nave").keyframeplayer.playrate=2
end
```

Conclusiones:

Las animaciones sencillas son bastante fáciles de programar, ya que el modificador “keyframeplayer” tiene una serie de propiedades y procedimientos muy sencillos. Lo que no es tan evidente son cuestiones como la manipulación de una serie de animaciones que pueden estar anidadas, o enlazadas secuencialmente. Esto puede ser el objetivo de un caso práctico más avanzado. También nos hemos quedado con la sensación de que el interfaz para controlar nuestra animación puede ser claramente mejorado con dos botones deslizadores, uno para controlar la velocidad de ejecución y otro para controlar la ejecución misma en el tiempo.

Unidad 14: Sistemas de partículas en Director.

Introducción teórica:

1. Introducción a los sistemas de partículas.
2. Creación de un sistema de partículas con una textura y Lingo.
3. Características de las partículas.
4. Características del emisor.
5. Características físicas del sistema de partículas.

Casos prácticos:

14.1. Sistemas de partículas, características geométricas.

14.2. Sistemas de partículas, características de emisión.

14.3. Sistemas de partículas, creación de efectos atmosféricos.

1. Introducción a los sistemas de partículas.

Los sistemas de partículas son entidades complejas. Se suelen utilizar como efectos especiales, aunque en la actualidad también se están empleando para crear algunos modelos muy característicos.

Los sistemas de partículas se denominan sistemas porque son estructuras con varios componentes, todos interrelacionados entre sí. Los podemos imaginar como "fuentes" de partículas. Por lo tanto, tendrán un origen, un emisor de partículas. Las partículas son pequeños elementos que tendrán un recorrido y luego desaparecerán.

Los sistemas de partículas se pueden utilizar para crear efectos visuales como el que se observa en la imagen inferior. En los sistemas en tiempo real como los videojuegos, se utilizan para generar fuego, rayos, etc.

Imagen 5.14.1: Sistema de partículas en ejecución

También se utilizan para crear efectos atmosféricos como niebla, lluvia y otros. Entre los casos prácticos se ilustra cómo utilizar los sistemas de partículas con este fin.

Los sistemas de partículas son en 3 dimensiones porque el recorrido de las partículas es tridimensional, pero en la mayoría de los sistemas en tiempo real, las propias partículas son elementos en 2 dimensiones, también en Director. Los resultados suelen ser muy efectivos, ya que normalmente las partículas son tan pequeñas que no hace falta que sean tridimensionales.

En la actualidad, los sistemas de partículas también se utilizan para objetos que antes no se podían realizar como pelo, hojas, césped, etc., lo que supone crear muchas partículas pero con "cuerpo" 3D (aunque muy pequeño) y que no siguen un recorrido, no se mueven y con la particularidad que el emisor no es un punto sino una superficie. Este tipo de partículas no las podemos crear hoy en día con Director.

Hoy en día, se están empleando sistemas de partículas dónde estas son modelos tridimensionales, sobre todo en el cine. En esta industria y/o en programas como 3d Studio MAX, no tenemos la condicionante del tiempo y por tanto, se pueden crear sistemas complejos donde las partículas forman manadas de mamíferos, cardúmenes de peces, batallones de soldados, etc.

En Director podemos crear sistemas de partículas donde las partículas son móviles y tengan dos dimensiones. O sea, las partículas serán imágenes. De todas formas, podemos utilizar imágenes tipo .gif que estén compuestas por varios fotogramas lo que nos proporciona grandes posibilidades para crear efectos especiales como veremos en los casos prácticos.

En la siguiente imagen se pueden apreciar dos sistemas de partículas. En el sistema de la izquierda, las partículas son 2D y en la derecha forman cuerpos 3D. Las diversas posibilidades de esta opción son grandísimas pero su gasto en cálculo computacional también.

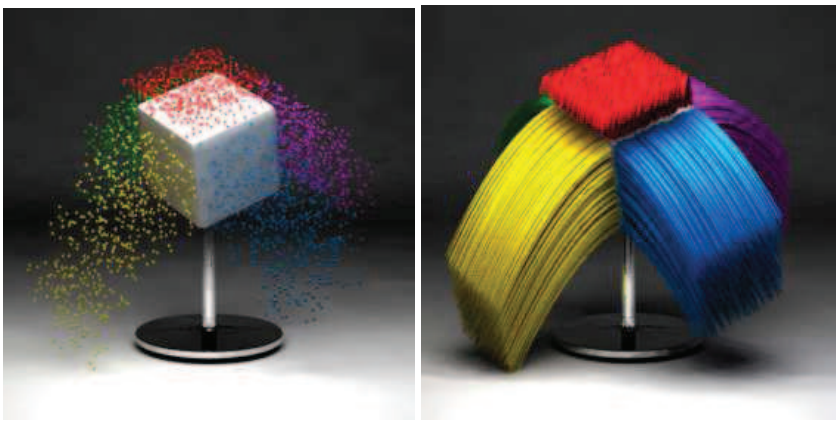


Imagen 5.14.2: Sistema con partículas 2D y 3D

En esta unidad teórica vamos a estudiar cómo crear un sistema de partículas desde el inicio utilizando solamente Lingo y una textura. Tomaremos como objetivo la creación de un sistema que recrea un fuego.

Luego nos dedicaremos a ahondar en las variantes que nos ofrece Lingo para crear sistemas de partículas muy versátiles dividiendo el trabajo en dos partes: primero veremos las características en cuanto a la forma del sistema, y en la segunda parte veremos las variables que inciden en la manera de emisión de las partículas.

Esto hará que tengamos dos casos prácticos para ilustrar cómo implementar esta teoría y acabaremos con un caso práctico dónde veremos cómo utilizar estas herramientas para modelar algunos fenómenos atmosféricos como lluvia y nieve.

Imagen 5.14.3: Sistema de partículas para generar lluvia

1.1. Creación de un sistema de partículas con una textura y Lingo.

Como hemos escrito anteriormente, en Director, para crear las partículas, debemos utilizar elementos 2D, es decir, imágenes. Podemos utilizar imágenes con o sin canales alpha, con animaciones como los de formato gif, etc.

Vamos a modelar un fuego y lo primero que necesitamos es crear una partícula base, una pequeña llama. Para ello emplearemos ® Adobe ® PhotoShop como en otras ocasiones.

En este caso utilizaremos una imagen con canal alpha para desdibujar los bordes tal como pasa con una llama real, dónde los bordes no son muy nítidos.

En las siguientes imágenes se puede observar el trabajo realizado en PhotoShop, en la primera de ellas se puede observar la textura creada. Los colores son muy típicos, porque el fuego que

queremos crear también lo es. En la segunda imagen vemos los canales y con una flecha roja se señala el canal “Alfa 1” que crea un recorte elíptico.

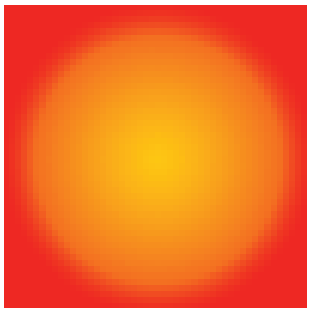


Imagen 5.14.4: Textura utilizada en la partícula

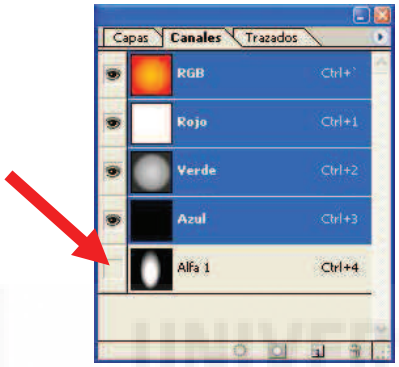


Imagen 5.14.5: Canal alpha utilizado en la partícula

Los sistemas de partículas varían considerablemente cuando cambiamos la partícula, por lo que en los casos prácticos hemos implementado métodos para intercambiarlas en tiempo de ejecución. También sería interesante experimentar con la creación de imágenes para poder obtener distintos efectos. La importación de las imágenes con canales alpha tiene alguna complicación como estudiamos en la unidad dedicada a las superficies de los modelos 3D por lo que remitimos a los lectores a esta unidad para solventar cualquier duda.

En el script “Programación” podemos escribir las siguientes sentencias que vamos a comentar para crear un completo sistema de partículas:

```
global mundo3D
property psprite
global fuego

property txt
property contador

on beginSprite me
        pSprite = sprite(me.spriteNum)

-- Preparación de la textura
        txt = mundo3d.newtexture("llamita", #fromcastmember, member("llama"))
        txt.renderformat = #rgba8888
-- Sistema de partículas FUEGO

        fuego = mundo3d.newmodelresource("particulas2", #particle) -- Creación del modelo
                                                                   sistema de partículas

        fuego.emitter.maxspeed = 1              -- Control de la velocidad de las partículas
        fuego.emitter.numparticles=100 -- Control del número de partículas
```

```
fuego.sizerange.start = 2                -- Control del tamaño inicial de las partículas
fuego.sizerange.end = 1                  -- Control del tamaño final de las partículas
fuego.blendrange.end = 10                -- Control del tamaño inicial de las partículas
fuego.lifetime = 750                     -- Control de la duración de las partículas en
                                         el sistema
fuego.texture = txt                      -- Asignación de la textura a cada una de las
                                         partículas

llamaradas = mundo3d.newmodel("fue1", fuego)          -- Llevamos fuego a
representarlo
                                                      en el mundo 3D
end
```

Se puede observar que la creación del sistema de partículas supone apenas nueve sentencias que comprenderemos mejor al seguir avanzando en su estudio. El resultado se puede apreciar en la imagen siguiente:

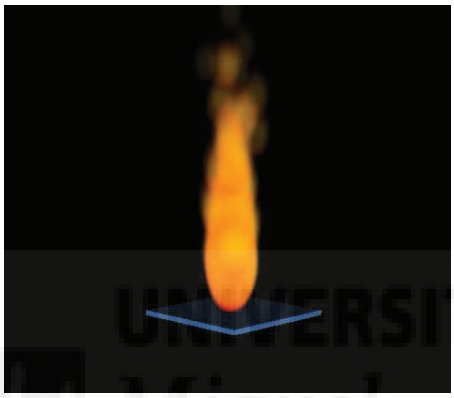


Imagen 5.14.6: Canal alpha utilizado en la partícula

2. Características de las partículas.

Estas características de las partículas se refieren a la forma en que estas realizan su recorrido e inciden mucho en la forma en que vemos el sistema de partículas, en su "geometría". Por ejemplo, si partimos del sistema de partículas del fuego creado anteriormente, podemos estudiar como hacerlo más alto, más ancho, más opaco, con otros colores, etc.

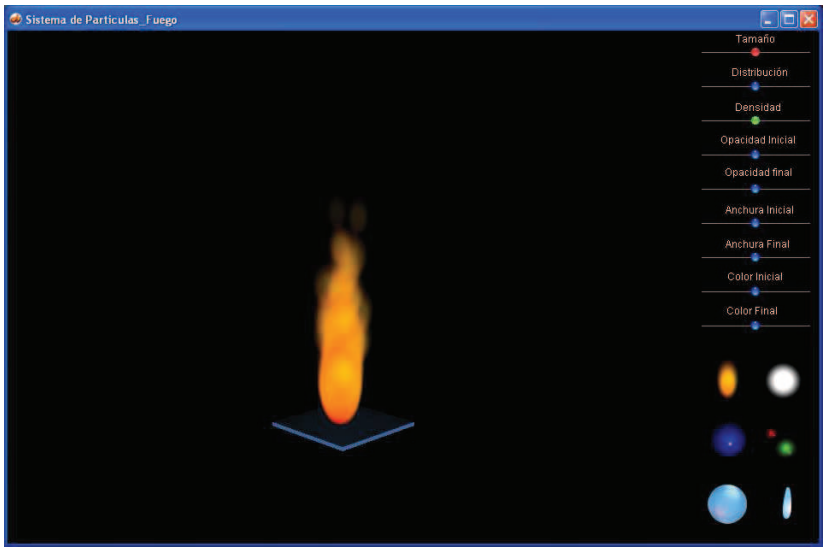


Imagen 5.14.7: Vista del caso práctico de geometría de los s. de partículas

En la imagen que sigue se puede apreciar una captura de pantalla del caso práctico 14.1 que ilustra las características que a continuación vamos a tratar.

Vamos a estudiar cada característica y como implementarla, es decir, los códigos en Lingo que permiten modificar los valores de estas propiedades para que cambien en tiempo de ejecución y por tanto, puedan ser experimentadas por los usuarios.

En la siguiente tabla se puede observar un resumen de cada magnitud que podemos modificar y del parámetro asociado a él, así como del rango de valores válidos para cada uno:

Magnitud	**Parámetro en Lingo**	**Rango válido de valores**
Tamaño	LilfeTime	De 0 a lo que queramos
Distribución	Emitter.maxspeed	De 1 a lo que queramos
Densidad	Emitter.numparticles	De 1 a lo que queramos
Opacidad inicial	Blendrange.start	De 0 a 100
Opacidad final	Blendrange.end	De 0 a 100
Tamaño inicial	Sizerange.start	De 1 a lo que queramos
Tamaño final	Sizerange.end	De 1 a lo que queramos
Color inicial	Colorrange.start	RGB
Color final	Colorrange.end	RGB
Textura	Texture	

Tabla 6: Parámetros geométricos de los sistemas de partículas

◆ Tamaño:

El tamaño del sistema de partículas, en realidad, lo determina el tiempo en el que las partículas siguen su recorrido. O sea, cuanto “viven”, cuanto tiempo son visibles antes de desaparecer. En Lingo este parámetro se llama “Lifetime” y se mide en milésimas de segundo.

Por ejemplo, en la imagen siguiente, el fuego de la izquierda tiene un valor de Lifetime de 10 milisegundos, en el fuego central las partículas duran 700 mseg. y el fuego de la derecha tiene un Lifetime de 1300 mseg. La diferencia es evidente. La sentencia para hacer que las llamas del fuego tarden 2 segundos en “morir” es: fuego.lifetime = 2000.

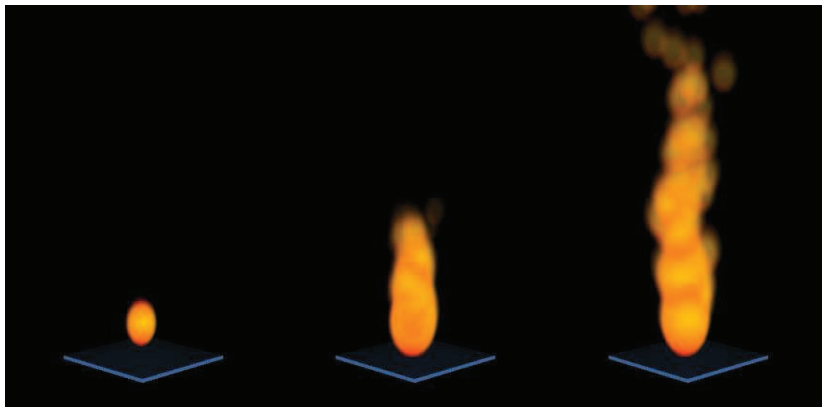


Imagen 5.14.8: Variación en el tamaño del sistema de partículas

◆ Distribución:

Este parámetro define la forma en que son emitidas las partículas, si el foco emisor es muy pequeño o más grande. Se le denomina “maxspeed” y nosotros hemos trabajado con los

valores de 1, 6 y 12 para los fuegos representadas en la imagen que sigue. Por ejemplo, para poner un fuego concentrado, escribiremos el código: fuego.emitter.maxspeed = 3.

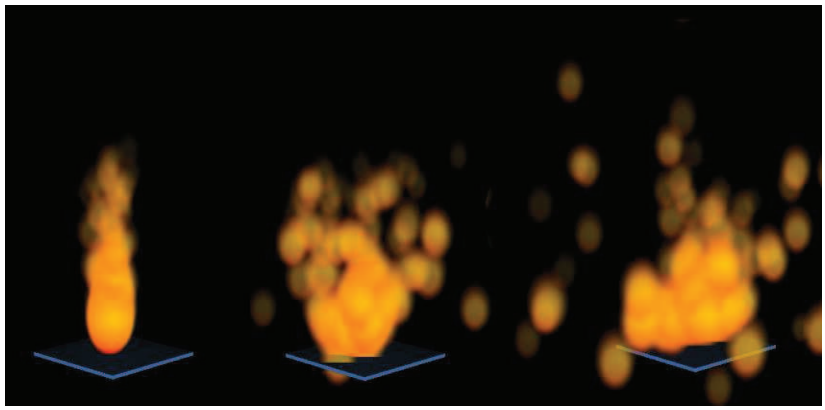


Imagen 5.14.9: Variación en la distribución del sistema de partículas

◆ Densidad:

Por densidad entendemos la cantidad de partículas que son emitidas por unidad de tiempo. Esta variable tiene una importancia capital en la visualización del sistema de partículas, ya que a mayor densidad, más "lleno" se visualizará el sistema.

En el caso del fuego se visualizará más compacto, las llamas serán más en el mismo espacio. Por ejemplo, para crear los fuegos que se puede ver en la imagen siguiente se han utilizado valores de 6, 70 y 120 para la variable "numparticles". Se ha asignado este valor de la siguiente forma: fuego.emitter.numparticles = 70.

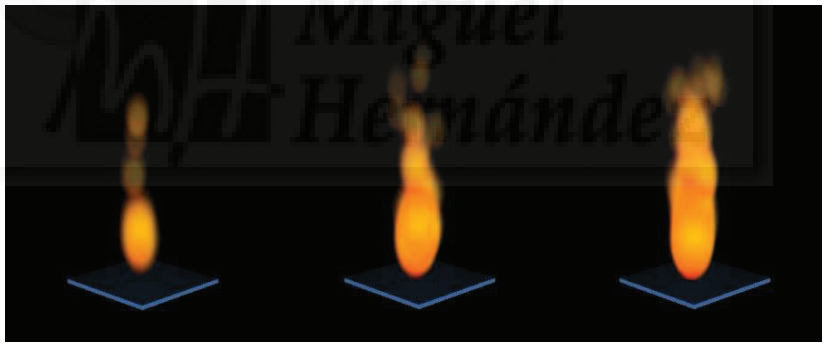


Imagen 5.14.10: Variación en la densidad del sistema de partículas

◆ Opacidad inicial:

La opacidad inicial, va ha controlar la opacidad / transparencia de las partículas en el momento en que salen del emisor. Dependiendo del sistema que queramos imitar, podemos necesitar que comiencen siendo emitidas más o menos invisibles. Este parámetro se controla utilizando la variable "blendrange.start".

En la imagen que sigue se pueden apreciar valores para esta magnitud de 0, 50 y 100 unidades. Esta variable solamente admite valores de 0 a 100. Para modificar esta variable escribimos: fuego.blendrange.start = 50.

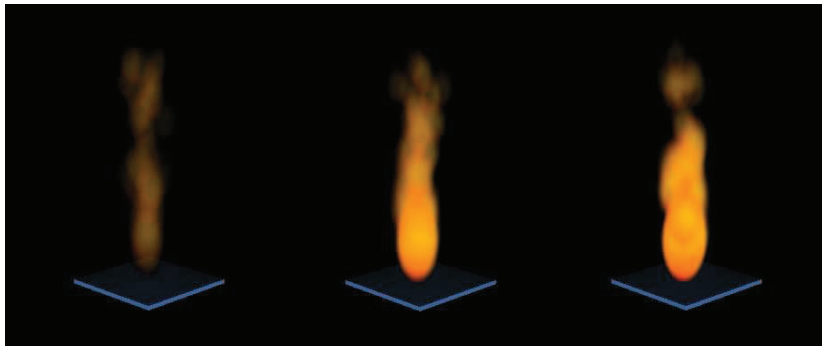


Imagen 5.14.11: Variación de la opacidad inicial

◆ Opacidad final:

La opacidad final, va ha controlar la opacidad / transparencia de las partículas antes de desaparecer mediante la variable "blendrange.end". Esta variable solo admite valores de 0 a 100. En la imagen siguiente se pueden apreciar valores para esta magnitud de 0, 50 y 100 unidades. Para modificarla usamos la sentencia Lingo: fuego.blendrange.end = 50.

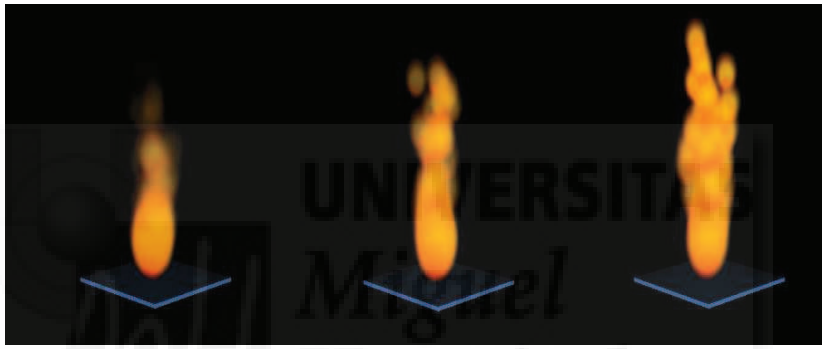


Imagen 5.14.12: Variación de la opacidad final

◆ Tamaño inicial:

El tamaño inicial del sistema de partículas va a determinar cuan grande son las partículas en origen. De esta forma genera emisiones totalmente distintas si variamos esta magnitud. La unidad de medida es el tamaño de la partículas base, es decir, el tamaño de la imagen importada como textura 2D, por lo que el mejor método para modificar este tamaño inicial es por comparación.

Por ejemplo, en la imagen que sigue, podemos visualizar tamaños iniciales de 1, 3 y 6 veces la textura de la partícula base. Para modificar este parámetro escribimos el siguiente código: fuego.sizerange.start = 3.

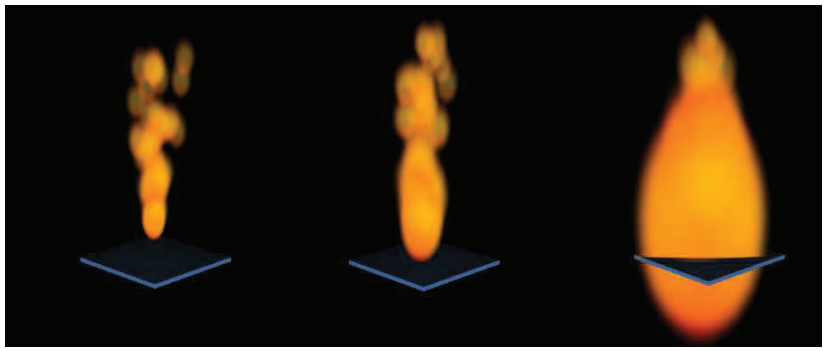


Imagen 5.14.13: Variación del tamaño inicial

◆ Tamaño final:

El tamaño final, de forma análoga a la variable antes descrita, va a determinar lo grande que son las partículas del sistema justo antes de desaparecer. Esto hace que se visualicen emisiones muy diferentes al modificar esta magnitud. La unidad de medida, igual que antes, es el tamaño de la partícula base.

En la imagen que sigue, se pueden ver tamaños finales de 1, 3 y 6 con respecto al tamaño de la partícula base. Para modificar el parámetro y tener una llama como la del centro de la imagen, escribimos el siguiente código: fuego.sizerange.end = 3.

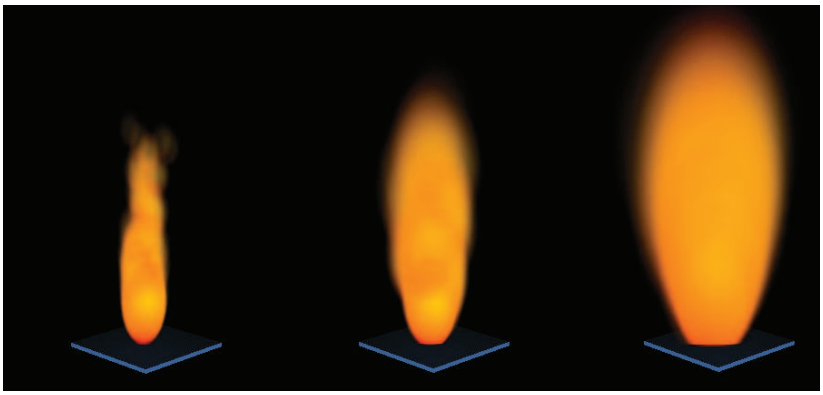


Imagen 5.14.14: Variación del tamaño final

◆ Color inicial:

El color inicial es una variable que tomando valores RGB (Rojo, Verde, Azul) coloreará las partículas al inicio de su emisión.

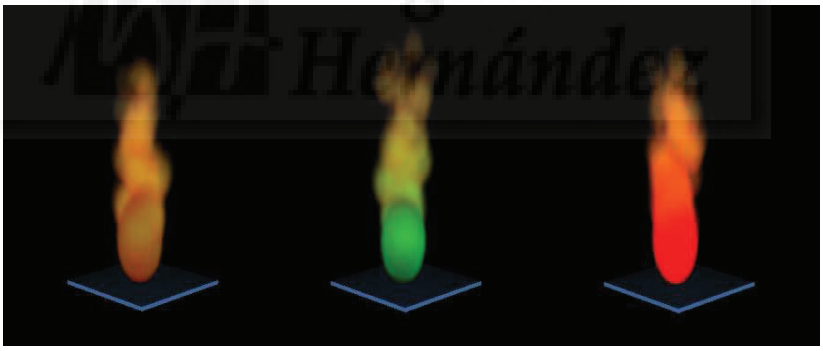


Imagen 5.14.15: Variación del color RGB inicial

Como se puede observar en la imagen anterior, tenemos tres sistemas de partículas idénticos salvo que esta variable se ha modificado para que en el caso del centro, su valor equivalga al color verde y en el caso de la derecha a rojo.

Debemos señalar que el color se mantiene en las partículas iniciales y se va modificando para acabar con su color original.

Como se aprecia en la imagen antes citada, el color del final del fuego es idéntico para los tres casos, aunque vamos a ver que también este dato lo podemos variar a lo largo de la ejecución de nuestros trabajos mediante la programación.

Para modificar este color utilizaremos un código como este: fuego.Colorrange.start = rgb(R, G, B), donde “R”, “G” y “B” será un valor numérico de 0 a 255 que represente la cantidad de color rojo, verde y azul respectivamente.

◆ Color final:

Este parámetro funciona de forma muy parecida al anterior, pero como su nombre indica, permite modificar el color de las partículas unos instantes antes de desaparecer, es decir, al final de la emisión del sistema de partículas.

En el caso del color final, el código para modificarlo será: fuego.Colorrange.end = rgb(R, G, B), donde “R”, “G” y “B” será un valor numérico de 0 a 255 que represente la cantidad de color rojo, verde y azul.

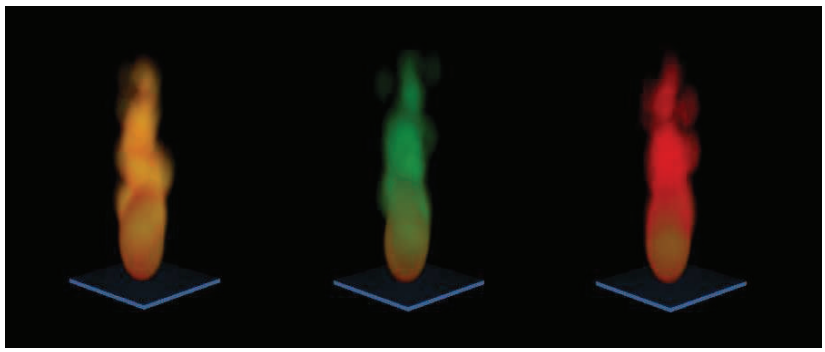


Imagen 5.14.16: Variación del color RGB final

◆ Textura:

La textura es un gráfico tipo mapa de bits que permite aplicar y visualizar cada partícula emitida por el sistema. Cómo se puede observar en la imagen que sigue, en el caso práctico hemos dispuesto 6 texturas distintas para poder compararlas y estudiar la forma en la que queda el sistema.

Es evidente que las texturas que empleemos deben estar integradas dentro de nuestro fichero de director. Esto ya lo estudiamos en anteriores prácticas. Estas texturas, normalmente, tendrán canales alpha para poder crear partículas más realistas.

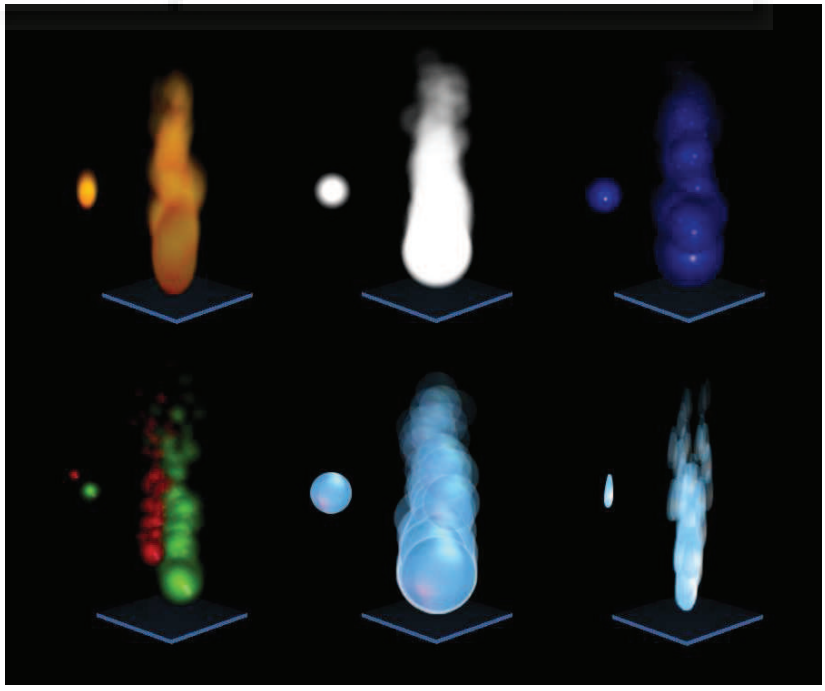


Imagen 5.14.17: El mismo sistema de partículas con distintas texturas

El código para asignar una textura al sistema de partículas será: fuego.texture = textura, pero teniendo en cuenta que la textura debe estar en el mundo tridimensional y que la calidad de representación de la textura debe permitir los canales alpha, previamente harán falta los códigos que siguen:

```
textura= mundo3d.texture("cop")
textura.renderformat = #rgba8888
```

3. Características del emisor.

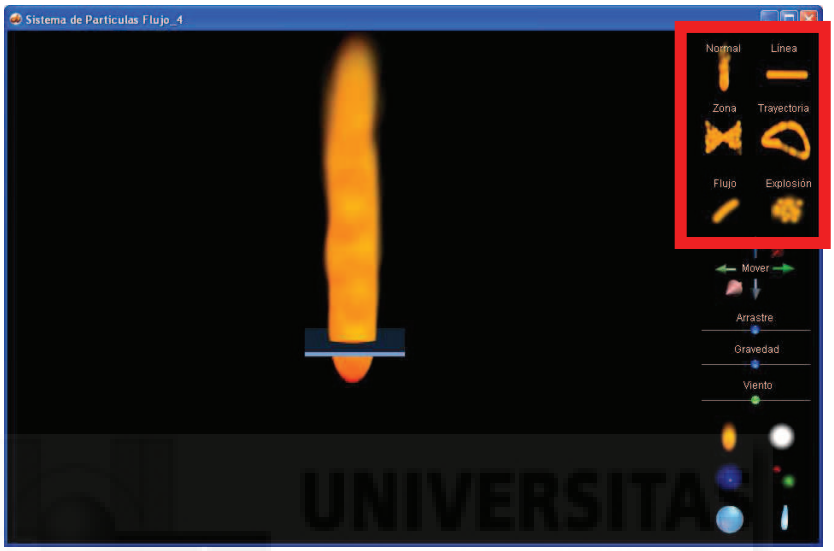


Imagen 5.14.18: Vista del caso práctico "Características de emisión"

En este punto, vamos a estudiar en profundidad las características de emisión de los sistemas de partículas. Veremos cómo puede ser el emisor, la cadencia de la emisión, y cómo el emisor puede ser un punto, una región o una trayectoria.

En la imagen anterior se puede observar una captura de pantalla del caso práctico 14.2, que ilustra las características que a continuación vamos a tratar y su interface. Se puede observar en rojo los botones que afectan a las magnitudes que estudiaremos en este punto.

Vamos a ahondar en los parámetros que inciden en el emisor de partículas y cómo se codifican. En la siguiente tabla se puede observar un resumen de cada magnitud que podemos modificar y del parámetro asociado a él, así como del rango de valores válidos para cada uno:

Magnitud	**Parámetro en Lingo**	**Rango válido de valores**
Modo de emisión	Emitter.mode	#stream o #burst
Región de emisión	Emitter.region	Grupo de vectores
Trayectoria de emisión	Emitter.path	Grupo de vectores

Tabla 7: Parámetros del emisor de los sistemas de partículas

- Modo de emisión

Este parámetro va a determinar la manera en que las partículas salen del emisor: si esta salida va a ser de forma continua o discreta. Por lo tanto, puede tomar dos valores: el modo continuo se llama modo de emisión de flujo o "afluencia" y el modo discreto se denomina modo "explosión".

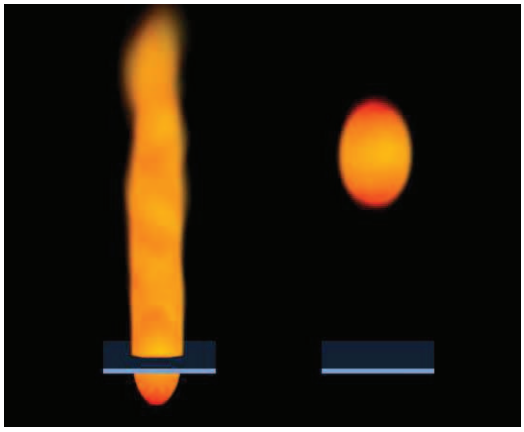


Imagen 5.14.19: Vista del sistema de partículas con los dos modos de emisión

• Flujo o afluencia:

En este modo la emisión de partículas es continúa. Para que este modo funcione correctamente, es muy importante que otros parámetros estén bien configurados. Quizá el más influyente sea “numparticles” que representa el caudal del emisor.

Para utilizar este método, debemos utiliza el valor #stream para la variable modo del emisor. Por ejemplo, la sentencia para poner el sistema de partículas fuego en forma continuo sería: fuego.emitter.mode= # stream.

• Modo Explosión:

En este modo, el emisor del sistema de partículas hace fluir a estas cada cierto tiempo, como si fueran pequeñas explosiones. Se emiten un número (numparticles) todas a la vez y luego se detiene para después repetir el proceso. Es el modo inverso al modo flujo.

Para utilizar este método, debemos utilizar el valor #burst para la variable modo del emisor. Por lo tanto, para realizar un botón que ponga el sistema de partículas en modo de explosión lo único que deberíamos hacer es poner la sentencia fuego.emitter.mode= #burst.

En la imagen anterior tenemos el mismo sistema de partículas. Todas sus características son idénticas salvo que el modo de emisión en el sistema de la izquierda está en modo flujo y en el de la derecha en modo explosión. Como se observa la diferencia es muy evidente en cuanto a resultado.

◆ Región de emisión

• Modo Normal:

Este modo se denomina normal porque los sistemas de partículas lo tienen por defecto. Se puede describir como una emisión donde las partículas salen desde un solo foco emisor.

Este modo se ve alterado por dos posibles parámetros del emisor: “region” y “path”, que intervienen directamente en la forma del emisor, que como hemos escrito antes, por defecto es un punto en el espacio 3D.

Si tenemos definida una región o trayectoria de emisión y queremos poner este modo activo, basta con poner valor nulo en los dos parámetros mencionados tal y como muestra este script:

```
global mundo3D
global fuego

on mouseDown me
```

```
  fuego.drag = 0
  fuego.emitter.region = [vector(0,0,0)]  -- para reiniciar región si la hubiera asignada
  fuego.emitter.path = [ ]                -- para reiniciar trayectoria si la hubiera asignada

  fuego.emitter.maxspeed = .5             -- características del sistema de partículas base, fuego
  fuego.emitter.numparticles=300
  fuego.sizerange.start = 2
  fuego.sizerange.end = 2
  fuego.blendrange.end = 10
  fuego.lifetime = 1000

  fuego.gravity=vector(0,0,0.5)
end mouseDown
```

Se puede observar en este script que hemos utilizado parámetros que aún faltan por explicar, pero sirve como base para su estudio.

• Modo Línea:

En realidad, el modo línea aquí definido no existe como tal, sino que lo hemos creado como un modo particular del modo región. Lo hemos definido porque su uso es muy habitual para crear paredes de fuego, cascadas de agua, etc.

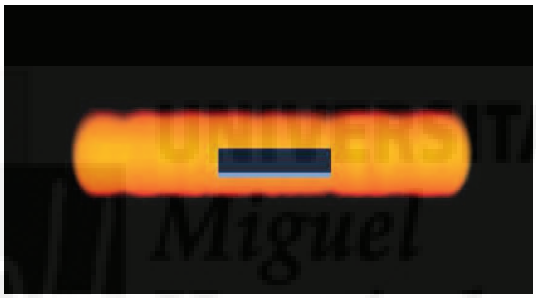


Imagen 5.14.20: Vista del sistema de partículas en el modo Línea

Para definir una región como una línea, tenemos que poner el parámetro “region” definido por dos vectores en línea sobre la que queremos que se cree la emisión de las partículas tal como muestra la siguiente sentencia:

```
fuego.emitter.region = [vector(-4,0,0), vector(4,0,0)]      -- crea una región lineal
```

En la imagen de arriba se puede observar cómo queda. Lo único destacable es que hemos utilizado un arrastre máximo (fuego.drag = 100) para que se observe bien la línea de fuego conseguida.

• Modo Zona o Región:

Este es el modo general del parámetro región. Viene definido normalmente por cuatro vectores en 3D que definen esta región. Con este parámetro, tenemos abierta la posibilidad de texturizar superficies con sistemas de partículas y poder hacer esculturas de fuego, agua, etc.

En la imagen siguiente se puede observar como se produce la emulsión de las partículas en toda una región a la vez. En el caso práctico 14.2, hemos realizado una región 2D por motivos didácticos ya que no hemos implementado una cámara para poder apreciar la profundidad del sistema de partículas. Para obtener una región de emisión como la que se muestra en la imagen anterior, hemos la hemos definido como sigue:

```
fuego.emitter.region = [vector(-4,0,4), vector(-4,0,-4), vector(4,0,-4), vector(4,0,4)]
```

Es necesario volvemos a utilizar un arrastre máximo para que se visualice fácilmente.

Con los sistemas de partículas con emisión en regiones podemos hacer sobre todo la recreación de fenómenos atmosféricos. Por ejemplo, podemos recrear una lluvia, ya que las gotas pueden ser emitidas desde una región que puede ser un plano y hacer que la emisión sea hacia abajo. También podemos crear niebla con las partículas adecuadas y partiendo de un plano o como hemos escrito antes elementos tridimensionales como puede ser la ladera de una montaña, una esfera que represente a un planeta, etc.

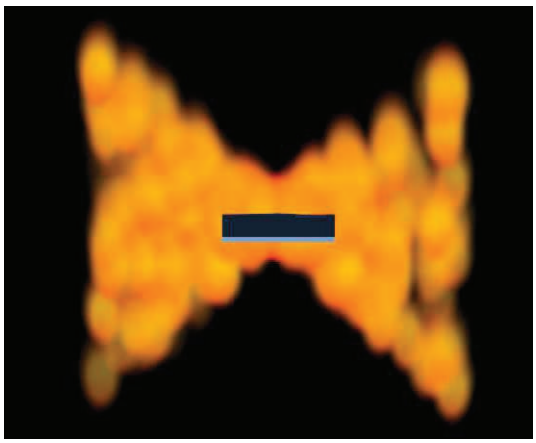


Imagen 5.14.21: Vista del sistema de partículas en el modo Región

- Trayectoria de emisión

Este punto es muy importante para crear efectos especiales. Normalmente los sistemas de partículas tiene un punto de emisión que expulsa partículas y estas salen despedidas siguiendo cada una de ellas una trayectoria propia gobernada por una serie de parámetros generales pero que generan cierta variabilidad en las trayectorias individuales.

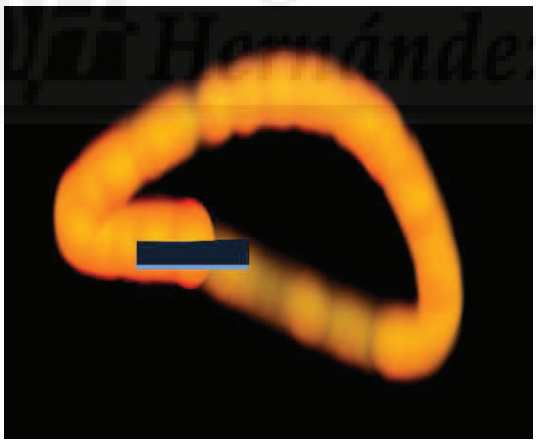


Imagen 5.14.22: Vista del sistema de partículas en el modo Trayectoria

En este modo, las partículas van a seguir una trayectoria diseñada por el autor, creando una especie de camino de partículas. En el caso que nos ocupa, un camino de fuego. Este camino tendremos que definirlo punto por punto.

La definición del camino a seguir por las partículas no servirá de nada sino alteramos el valor de otros parámetros como “drag” y “pathstrengh” que definen la fuerza con la que el camino tira de las partículas para que no se salgan de su trayectoria.

Para controlar todos los parámetros que tenemos que abordar se necesita cierta experiencia, por lo que seguidamente se muestra el script que hace que un botón determinado al ser

pulsado genere la trayectoria y haga que el sistema se visualice como muestra la imagen superior:

```
global mundo3D
global fuego

on mouseDown me
  fuego.emitter.region = [vector(0,0,0), vector(0,0,0)]      -- para reiniciar región
  fuego.drag = 5

  fuego.emitter.pathstrength = 3                              -- con 1 hace una espiral
  fuego.emitter.path = [vector(0,0,0),vector(-2,0,0),vector(0,0,2),vector(4,0,0),vector(0,0,-1)]
end mouseDown
```

Si en el código anterior ponemos la variable “pathstrength” al valor 1, produce una especie de espiral en vez del triángulo, por eso debemos probar para dominar este parámetro. Las aplicaciones de trayectorias son muchas como pueden un rayo o un halo que no sean lineales o que unas mariposas sigan cierta trayectoria, etc.

5. Características físicas del sistema de partículas.

Estudiaremos aquí los parámetros que podemos modificar en el sistema de partículas para que este adquiera características físicas. Vamos a utilizar tres variables: arrastre, viento y gravedad, además de la traslación del emisor. En la imagen que sigue se puede apreciar una captura de pantalla del caso práctico que ilustra las características que a continuación vamos a tratar y su interface. Se puede observar en rojo los botones que afectan a las magnitudes que estudiaremos en este punto.

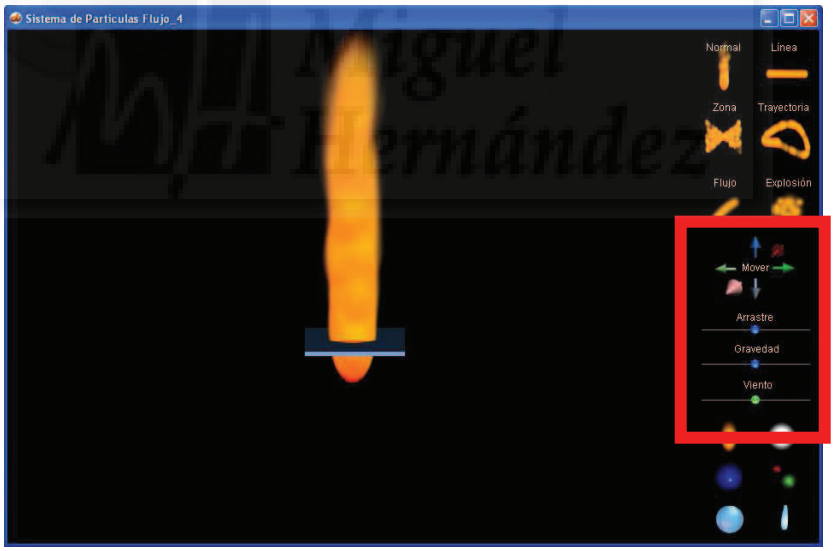


Imagen 5.14.23: Vista del caso práctico “Características de emisión”

En la siguiente tabla se puede observar un resumen de cada magnitud que podemos modificar y del parámetro asociado a él, así como del rango de valores válidos para cada uno:

Magnitud	Parámetro en Lingo	Rango válido de valores
Traslación	ninguno	
Arrastre	Drag	De 0 a 100
Gravedad	Emitter.path	Grupo de vectores
Viento		

Tabla 8: Parámetros físicos del emisor de los sistemas de partículas

Si tenemos un control suficiente de estas características podremos crear aplicaciones con los sistemas de partículas pare recrear efectos naturales y físicos. Por ejemplo, podremos crear cascadas donde las partículas se vean condicionas por la gravedad y el viento.

- Traslación

Con esta funcionalidad, tenemos la posibilidad de trasladar el sistema de partículas por el mundo tridimensional interactivamente. En el caso práctico adjunto hemos diseñado unos botones para que cuando son pulsados, todo el sistema de partículas se traslada por los tres ejes y en las dos direcciones para que pueda ser trasladado donde el usuario desee.

En la imagen inferior se puede observar como el fuego original se ha trasladado hacia abajo. Nos tenemos que fijar en la caja azul que sirve como referencia y que en este caso queda a mitad del sistema de partículas en vez de abajo del todo.

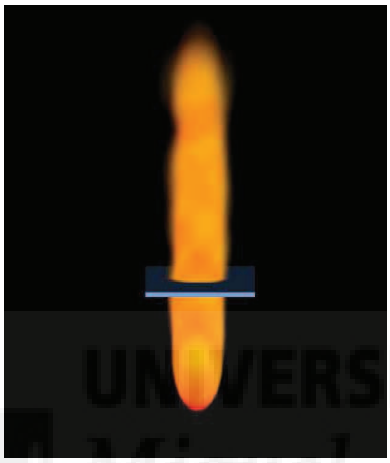


Imagen 5.14.24: Muestra del sistema de partículas trasladado hacia abajo

Para realizar el script asociado a un botón que por ejemplo, traslade el sistema de partículas hacia abajo, debemos utilizar una variable global que contenga el identificador del sistema definido en otro script y simplemente lo usamos para aplicarle la función “translate” que es accesible a cualquier objeto del mundo virtual. Otro aspecto a tener en cuenta es que al pulsar el botón implicado, el cambio de posición será discreto, es decir, en cierta cantidad, por lo tanto tendremos que pulsar varias veces los botones para llevar el sistema de un sitio a otro. Podríamos hacer un botón que funcionase de forma continua pero los scripts involucrados se complicarían.

```
global mundo3D
global refspmio

on mouseDown me
  refpmio.translate(-1,0,0)
end mouseDown
```

El objeto fuego, es el modelo que hace de huésped en el mundo 3D para el sistema de partículas cuyas características se habrán definido previamente. Esto se hace en el script “Programación” con el código que se muestra abajo.

```
refspmio = mundo3d.newmodel("loquesea", sistema_de_particulas)
```

- Arrastre

El arrastre es una variable que tiene mucha influencia en otras variables como la gravedad, el viento, etc. Mide el tanto por ciento de partículas que se verán afectadas por ellas. Su valor puede ir de 0 a 100. Para su interacción se ha creado un botón deslizante.

Por lo tanto, para observar como funciona, debemos crear un sistema con viento o gravedad y modificar el arrastre para ver como los afecta. En la imagen inferior se puede visualizar como para una cantidad determinada de viento, todo el sistema cambia en función del porcentaje de partículas que se ven afectadas por el viento, es decir, el arrastre asignado.

En el caso práctico 14.2, se utiliza un botón deslizador (ya estudiamos en otros temas como se puede general este tipo de control) que devuelve un valor entre 0 y 100 y cuyo código asociado se presenta debajo:

```
on mouseUp me
  valor=sprite(slider).locH
  arrastre=(valor-812)/12.7
  if (arrastre=0) then arrastre = 1
  fuego.drag=arrastre
end mouseup
```

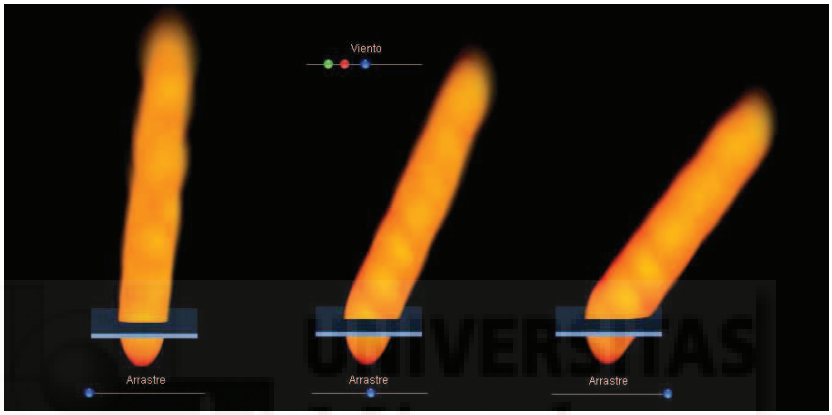


Imagen 5.14.25: Cambios en el sistema por distintos arrastres para un mismo viento

- Gravedad

La gravedad es una característica que afecta a todo el sistema de partículas. Sirve para hacer más realista el flujo creado al verse afectado por una fuerza física universal como es la gravedad, por que en la mayoría de los casos se utilizará para generar esta fuerza dinámica y que puede tener cualquier dirección tridimensional. La gravedad en Lingo tenemos que codificarla mediante un vector, por ejemplo, podemos poner los valores fuego.gravity = vector(-xg, -yg, -zg), para que la gravedad, haga como en la tierra y origine que las partículas se desplacen hacia abajo, o sea, cayendo. En la siguiente imagen se puede observar como el fuego está cayendo debido al efecto producido por este parámetro.

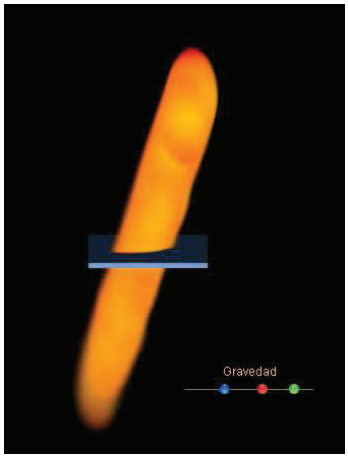


Imagen 5.14.26: Sistema de partículas afectado por Gravedad

- Viento

Es una fuerza parecida a la de gravedad pero que se utiliza para que las partículas se vean arrastradas como si de un viento se tratara. Al igual que la gravedad, el valor del viento lo introduciremos como un vector de 3 dimensiones ya que esta fuerza es tridimensional.

Así entonces, el siguiente código define el viento para el sistema fuego:

```
fuego.wind=vector(xv,yv,zv)          //definimos el viento en una determinada dirección
```

Como conclusión a este tema podemos decir que los sistemas de partículas, aunque es un tema un poco complicado de partida, merece la pena controlarlo por las grandes posibilidades que nos proporciona para dotar de realismo y/o efectos especiales a nuestra película interactiva.

Como hemos estudiado, partiendo de un sistema básico, podemos ir modificando valores de parámetros para hacer creaciones muy originales. Si bien es cierto, que la única herramienta con la que trabajar es directamente en Lingo, también podemos apuntar que todas sus variables son claramente comprensibles y nos pueden dar un control muy importante para mejorar nuestros proyectos con características innovadoras.

Caso práctico 14.1. Sistemas de Partículas, características geométricas.

Objetivo: Con esta práctica pretendemos introducirnos en los sistemas de partículas. Vamos a realizar un trabajo para estudiar las características "geométricas" de los sistemas de partículas como son el tamaño, la distribución, los colores, la densidad de partículas y otras. Además también podremos cambiar la forma de las partículas.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Preparar un mundo 3D para introducir el sistema de partículas.
2. Creación del sistema de partículas que simulan un fuego.
3. Características geométricas del sistema de partículas.

1. Preparar un mundo 3D para introducir el sistema de partículas.

Vamos a comenzar creando un sistema de partículas que imite un fuego. Aunque parezca una pretensión difícil, resulta que se adapta muy bien a la forma en que funcionan los sistemas de partículas. Luego lo podremos utilizar como prototipo para experimentar y poder realizar por ejemplo, una nevada, lluvia y otros efectos. Estos "efectos atmosféricos" se pueden utilizar no solo como ambientación en nuestras obras artísticas, sino también formando parte de ellas. El objetivo final es poder integrar nuevos tipos de sistemas de partículas que nos permitan experimentar con ellos para plasmar nuestra creatividad, es decir, crear una especie de laboratorio de sistemas de partículas.

El sistema de partículas que queremos crear se puede ver en la imagen 5.14.cp.1.1. Se puede observar que es como una llama de cierta altura que parece surgir de una caja muy delgada, casi un plano.

1.1. Construcción de la escena 3D.

La escena tridimensional no tiene ninguna dificultad, ya que se trata simplemente de crear una caja de unas dimensiones determinadas que nos sirvan de referencia para colocar el fuego en un espacio 3D. Para hacernos una idea, la caja creada es de 6 x 6 x 0,2 centímetros y está situada en las coordenadas (0, 0, 0).

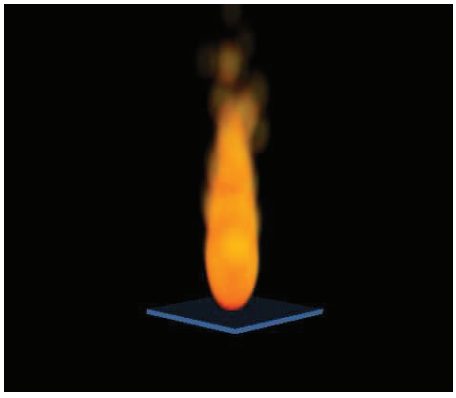


Imagen 5.14.cp.1.1: Vista del sistema de partículas utilizado en la práctica

1.2. Exportar la escena 3D a Shockwave. Para ello hacemos como siempre, File > Export y en la caja de diálogo "Select File to Export", elegimos una carpeta de destino, ponemos un nombre (yo he utilizado "Paisaje en negro") y el tipo ".w3d".

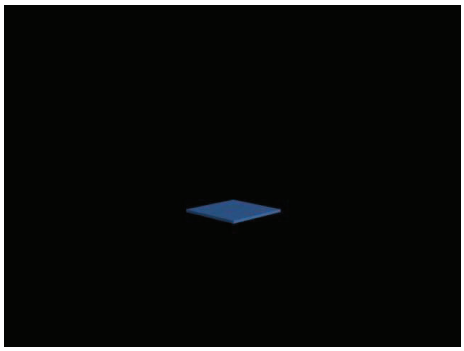


Imagen 5.14.cp.1.2: Vista de la escena 3D en Director

Con respecto a la carpeta, merece la pena, como hemos hecho en anteriores ocasiones, crear una carpeta propia para esta práctica, yo la he llamado “Sistema de partículas Fuego”.

1.3. Estructura del archivo de Director. En nuestro caso he preferido reutilizar una práctica ya creada para no tener que generar behaviors, etc. Pero comenzar desde el principio tampoco lleva mucho tiempo. Sea como fuere, al final, debemos tener un fichero de Director con la estructura típica de dos fotogramas: inicio y principal y al menos, 3 Behaviors: Inicialización, Bucle y Programación.

1.4. Importar la escena a Director. Realizaremos File > Import y en la caja de diálogo “Import Files into Internal”, elegiremos “Paisaje en negro.w3d” y luego pulsaremos “Add” y finalmente Import. Entonces podremos comprobar que lo tenemos disponible como un castmember más en la ventana Cast. Luego lo arrastramos hasta la línea de tiempo de tal manera que tengamos un sprite que ocupe un solo fotograma, el del “principal”. Por último solamente nos falta arrastrar el behavior llamado Programación sobre el sprite del mundo 3D para asignárselo.

2. Creación del sistema de partículas que simulan un fuego.

2.1. Crear las texturas. Los sistemas de partículas son elementos que funcionan de una manera muy particular, ya que hay que pensar que las partículas son elementos en 2D, no son 3D. En realidad, lo que se emite, son planos texturizados. Por tanto, antes de crear un sistema de partículas, tenemos que crear la textura básica que formará el fuego. Este trabajo lo realizamos con PhotoShop.

De todas formas, para poder demostrar que el mismo sistema de partículas cambia sustancialmente cuando cambia la partícula en sí, hemos creado 6 partículas, todas ellas con elementos bien diferenciados para poder experimentar con distintas formas base.

En la imagen siguiente se pueden observar las 6 texturas creadas para este caso práctico. La primera empezando por la izquierda será la que utilicemos para recrear un fuego. La siguiente la hemos importado de un trabajo práctico y servía como copo de nieve. La última también se empleó anteriormente como gota de lluvia. Las otras tres se han introducido para experimentar con texturas no frecuentes, por ejemplo, hay una partícula con dos núcleos, otra de aspecto muy artificial como si fueran de plástico y la última imita a una burbuja.

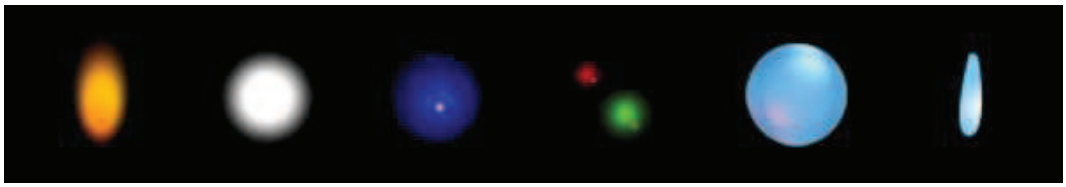


Imagen 5.14.cp.1.3: Las seis texturas creadas

La creación de las texturas con canales alfa ya la estudiamos anteriormente. En la imagen que sigue, se puede apreciar que el canal Alfa 1 tiene una degradado que formará una especie de elipse vertical difuminado para imitar una pequeña llama al recortar el canal RGB.

Imagen 5.14.cp.1.4: Vista de los canales alfa utilizados en "llamita"

2.2. Importar las texturas. La importación de estos bitmaps con canales alpha es como cualquier otro caso por lo que remitimos al caso práctico 12.3.

2.3. Crear el sistema de partículas inicial. Esta tarea se hace en Lingo y lo haremos en el behavior "Programación" que esta asignado al sprite del mundo 3D. Para ello escribimos:

```
global mundo3D
property psprite
global fuego

property txt
property contador

on beginSprite me

        pSprite = sprite(me.spriteNum)

-- Preparación de la textura
        txt = mundo3d.newtexture("llamita", #fromcastmember, member("llama"))
        txt.renderformat = #rgba8888

-- Sistema de partículas FUEGO

        fuego = mundo3d.newmodelresource("particulas2", #particle) -- Creación del modelo
                                                                   sistema de partículas
        fuego.emitter.maxspeed = 1              -- Control de la velocidad de las partículas
        fuego.emitter.numparticles=100 -- Control del número de partículas
        fuego.sizerange.start = 2               -- Control del tamaño inicial de las partículas
        fuego.sizerange.end = 1                 -- Control del tamaño final de las partículas
        fuego.blendrange.end = 10               -- Control del tamaño inicial de las partículas
        fuego.lifetime = 750                    -- Control de la duración de las partículas en
                                                el sistema
        fuego.texture = txt                     -- Asignación de la textura a cada una de las
                                                partículas

        llamaradas = mundo3d.newmodel("fue1", fuego)  -- Llevamos fuego a representarlo
                                                        en el mundo 3D

end
```

En esta práctica no hemos querido complicar más el código y dejaremos propiedades como la interacción del sistema de partículas con la gravedad y el viento para otra práctica donde se estudien sus propiedades que tienen relación con otros aspectos no “geométricos”.

3. Características geométricas del sistema de partículas.

En la imagen siguiente se puede apreciar los botones que se pueden utilizar para cambiar la forma del sistema de partículas.

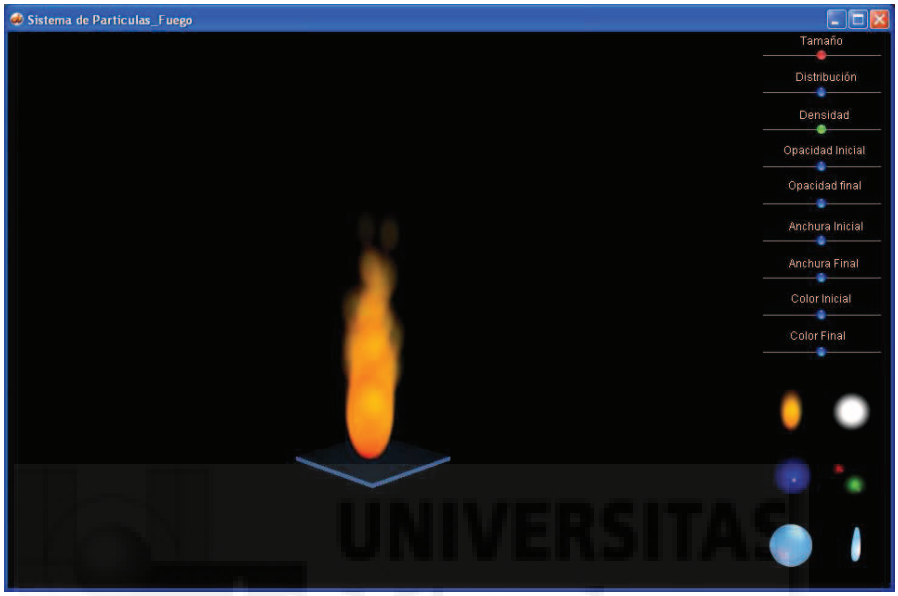


Imagen 5.14.cp.1.5: Vista de la práctica al inicio de su ejecución

También se pueden observar 6 botones que representan otras tantas partículas básicas. Es evidente que la mayoría de estas partículas tienen forma redondeada. Esto se debe a que es la forma que más se asemeja a los sistemas reales y por tanto los que mejor funcionan. Para modificar estas partículas básicas nos limitaremos a cambiar la textura que presentan. Las características que vamos a poder modificar partiendo del sistema de partículas “fuego” son muchas pero solo vamos a estudiar las que se refieren a la forma básica, la forma “geométrica” del sistema de partículas. Cada característica tendrá un código que nos permitirá modificar su valor en tiempo de ejecución, es decir, interactivamente, mientras se ejecuta nuestra aplicación. Así mismo, cada propiedad admitirá unos valores dentro de un rango.

Para una mayor comodidad y comprensión por parte del usuario, hemos utilizado controles tipo botones deslizantes, por lo que tenemos que tener en cuenta que habrá que “traducir” lo que el usuario ha manipulado a un número dentro del rango posible de valores de la variable en cuestión, ya que de otra forma, nos dará un error al querer asignar un valor imposible a una variable que no puede admitirlo.

3.1. Escribir la programación para controlar el tamaño del sistema de partículas. El siguiente script se le asigna a un botón deslizador que en nuestro caso, servirá para cambiar interactivamente la altura del fuego.

```
global mundo3D
global fuegotamanno
global fuego

property valor
property delcentro
property slider
property oldFacerValue
```

```
property newLocH
property limiteizquierda
property limitederecha

on beginSprite me
  slider = sprite(me.spriteNum)
  limiteizquierda=812
  limitederecha=940
  sprite(slider).locH= 875
end

on mouseDown me
  delcentro = the mouseH - sprite(slider).locH
  oldFaderValue = 0
  repeat while the mouseDown
    newLocH = the mouseH - delcentro
    if newLocH < limiteizquierda  then newLocH = limiteizquierda
    if newLocH > limitederecha then newLocH = limitederecha
    faderValue = newLocH - limiteizquierda - ((limitederecha - limiteizquierda) / 2)
    oldFaderValue = faderValue
    sprite(slider).locH = newLocH
    updateStage
  end repeat
end mouseDown

on mouseUp me
   valor=sprite(slider).locH
   fuegotamanno=(valor-812)*10                -- cálculo del valor introducido interactivo
   if (fuegotamanno=0) then fuegotamanno = 1  -- filtro por si fuera un tamaño imposible
   fuego.lifetime = fuegotamanno              -- asignación del valor a lifetime
end mouseup
```

Se puede observar que las tres líneas comentadas son las más importantes ya que es dónde verdaderamente se produce la asignación de un valore a la variable que permite establecer el tamaño del sistema de partículas, que no es otro que Lifetime.

En el caso del tamaño determinado por la variable “Lifetime” no tenemos problemas con el valor mayor, pero si con el mínimo ya que si vale 0, las partículas desaparecerían nada más salir del emisor y por tanto no se vería nada.

El cambio de tamaño se puede observar en la siguiente imagen. De izquierda a derecha os valores son de 10, 700 y 1300 milisegundos. Esto hace que las partículas permanezcan un tiempo mayor de recorrido y por tanto que todo el sistema sea más grande.

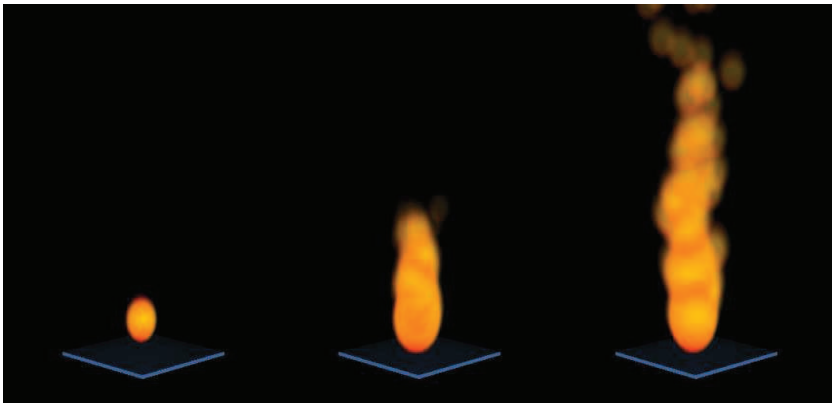


Imagen 5.14.cp.1.6: Variación en el tamaño del sistema de partículas

De la misma forma programaríamos los demás botones deslizables y solo tendríamos que modificar las tres sentencias comentadas para adaptarlas a cada magnitud. En las siguientes líneas se relacionan las variables y magnitudes relacionadas.

Magnitud	**Parámetro en Lingo**	**Rango válido de valores**
Tamaño	LilfeTime	De 0 a lo que queramos
Distribución	Emitter.maxspeed	De 1 a lo que queramos
Densidad	Emitter.numparticles	De 1 a lo que queramos
Opacidad inicial	Blendrange.start	De 0 a 100
Opacidad final	Blendrange.end	De 0 a 100
Tamaño inicial	Sizerange.start	De 1 a lo que queramos
Tamaño final	Sizerange.end	De 1 a lo que queramos
Color inicial	Colorrange.start	RGB
Color final	Colorrange.end	RGB
Textura	Texture	

Tabla 9: Parámetros geométricos de los sistemas de partículas

3.2. Crear los botones y los script para controlar el resto de magnitudes. Como se puede apreciar en la imagen 5.14.cp.1.5, utilizaremos botones de desplazamiento para variar el valor de todos los parámetros excepto de la textura.

3.3. Programar los botones para elegir la textura de las partículas. La textura será elegida con un botón “normal”. Esto quiere decir que pulsaremos directamente sobre la imagen de la textura deseada que por tanto es un botón. Estos botones los hemos realizado ya muchas veces, por lo que solamente cabe recordar que la textura debe estar integrada dentro de Shockave 3D. También tenemos que tener en cuenta que la calidad de representación de la textura debe permitir los canales alpha.

En la siguiente imagen se puede observar una nueva textura y cómo queda el sistema de partículas al aplicarla. No se han modificado los demás parámetros para poder comparar los sistemas de partículas resultantes con mayor facilidad.

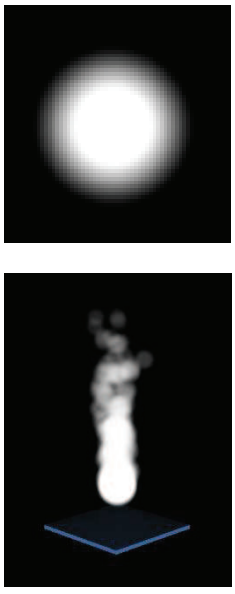


Imagen 5.14.cp.1.7: Aplicación de otra textura en el mismo S. de partículas

El siguiente script se le asigna al botón que aplica la nueva textura en el sistema de partículas, suponiendo que la textura ya está incorporada en Shockwave 3D y se llama “copo”:

```
global mundo3d
global fuego

property textur

on mouseDown
  textur= mundo3d.texture("copo")
  textur.renderformat = #rgba8888
  fuego.texture = textur
end
```

Conclusiones:

Nuestro sistema 3D nos permite la creación de sistemas de partículas, que sin duda, aportan unas posibilidades muy creativas para el artista 3D. Por una parte, permite la simulación de fenómenos físicos realistas como fuegos, lluvia, motores a reacción, y por otra, nos deja las manos libres para poder crear todo tipo de sistemas que se nos ocurran y que podemos utilizar para crear efectos especiales. Además, la forma en que se crean los sistemas de partículas es bastante comprensible, a pesar de que conllevan la utilización de muchas características dinámicas.

Esta herramienta será del gusto de muchos artistas 3D, ya que sus aplicaciones dependen mucho de la observación y la imaginación por lo que es una ventana abierta a la experimentación. En la siguiente práctica nos adentraremos en el control del flujo del sistema de partículas que hemos empezado a crear en esta práctica.

Caso práctico 14.2. Sistema de Partículas, características de emisión

Objetivo: En esta segunda práctica sobre los sistemas de partículas vamos a realizar un ejercicio para poner de manifiesto las características de "emisión" de las partículas, tales como regiones, trayectorias o puntos y los modos continúo y explosión. También podremos experimentar cómo se puede manipular el lugar de emisión en tiempo de ejecución y aplicarle tanto gravedad como viento.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Creación del mundo 3D y el sistema de partículas base.
2. Creación de los botones para control del modo de emisión.
3. Creación de los botones para control de la forma de emisión
4. Creación de los botones para control de características físicas.

1. Creación del mundo 3D y el sistema de partículas base.

1.1. Construcción de la escena 3D.

El caso práctico que vamos a generar es la que se puede ver en la imagen siguiente. Debemos destacar que nos vamos a basar en el caso práctico precedente para minimizar esfuerzos. Por ello, la construcción de la escena 3D y la puesta en marcha será idéntica al anterior trabajo. Nos limitaremos, pues, a cambiar unos botones por otros y a dotarles de la funcionalidad requerida.

Imagen 5.14.cp.2.1: Vista de la práctica que se propone terminada

Lo que sí debemos hacer notar, es que tendremos 4 tipos de botones: los que modifican el tipo de emisión del sistema de partículas, los que modifican la forma del sistema, los que modifican sus características físicas y por último, los botones para poder cambiar de partícula y que sirven para poder interpretar mejor las pruebas que hagamos.

2. Creación de los botones para control del modo de emisión.

En este apartado vamos a estudiar las características que influyen directamente en el modo en que las partículas salen del foco de emisión, que puede ser de dos tipos: continuo o explosión, tal como se observa en la siguiente imagen.

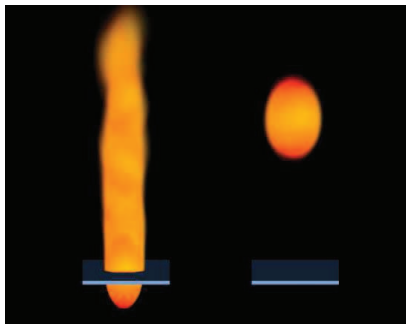


Imagen 5.14.cp.2.2: Modo continuo y modo explosión del mismo sistema

2.1. Crear el botón para poner el modo flujo.

Cuando este modo esta puesto, la emisión de partículas es continúo. Muchas características influyen en el forma en que se emitan las partículas, pero la más influyente es la cantidad de partículas emitidas por unidad de tiempo. Esta magnitud la controla la variable “numparticles”. El botón será una imagen como la que se muestra a la que se le asignará el script que sigue:

Imagen 5.14.cp.2.3: Botón para el modo flujo

```
global modo
global fuego

on mouseDown
   fuego.emitter.mode= #stream
end
```

2.2. Crear el botón para poner el modo explosión.

En este modo las partículas son emitidas a explosiones. Se emiten las partículas todas juntas, se detiene un instante y luego se vuelven a emitir. Es el modo inverso al continuo. El botón será una imagen simple a la que se le asigna un script. La imagen del botón que llevará a cabo esta función es la que sigue y a continuación tenemos el script que se le asigna.

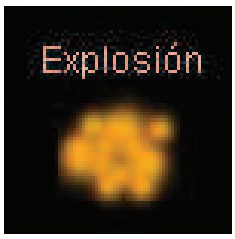


Imagen 5.14.cp.2.4: Botón para el modo explosión

```
global modo
global fuego

on mouseDown
   fuego.emitter.mode= #burst
end
```

3. Creación de los botones para control de la forma de emisión

3.1. Crear el botón para poner la forma de emisión normal.

En este modo de emisión, las partículas van a ser emitidas desde un punto 3D. Este es el modo por defecto en que funcionan los sistemas de partículas. El problema es volver a este modo si estamos en uno de las otras formas de emisión, ya que esto supone utilizar valores nulos para ellos. En el siguiente script se comentan dos líneas que realizan esta función.

```
global mundo3D
global fuego

on mouseDown me
  fuego.drag = 0
  fuego.emitter.region = [vector(0,0,0)]  -- para reiniciar región si la hubiera asignada antes
  fuego.emitter.path = [ ]                -- para reiniciar trayectoria si la hubiera asignada antes

  fuego.emitter.maxspeed = .5             -- características del sistema de partículas base, fuego
  fuego.emitter.numparticles=300
  fuego.sizerange.start = 2
  fuego.sizerange.end = 2
  fuego.blendrange.end = 10
  fuego.lifetime = 1000

  fuego.gravity=vector(0,0,0.5)
end mouseDown
```

3.2. Crear el botón para poner la forma de emisión en línea.

En realidad, la forma de emisión en línea no existe directamente sino que es un modo recortado del modo región. Para ello, definimos la región con solamente dos vectores, lo que evidentemente crea una región lineal como la que se aprecia en la imagen siguiente.

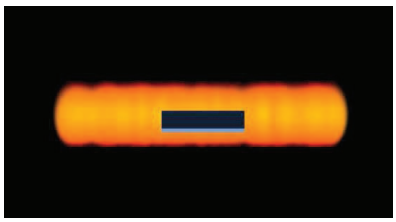


Imagen 5.14.cp.2.5: Forma de emisión en Línea

Para que este modo sea claramente visible, la variable “drag” debe tener un valor máximo y así hacer que las partículas no se separen mucho del emisor. El script necesario para crear un sistema de partículas como el que se visualiza en la anterior imagen será:

```
global mundo3D
global fuego

on mouseDown me
  fuego.drag = 100
```

```
  fuego.emitter.region = [vector(-4,0,0), vector(4,0,0)]
end mouseDown
```

3.3. Crear el botón para poner la forma de emisión en región.

En esta forma de emisión se permiten crear zonas tridimensionales de las que parten las partículas. En la imagen inferior se puede apreciar cómo se ha definido una zona en 2D con una forma muy particular.

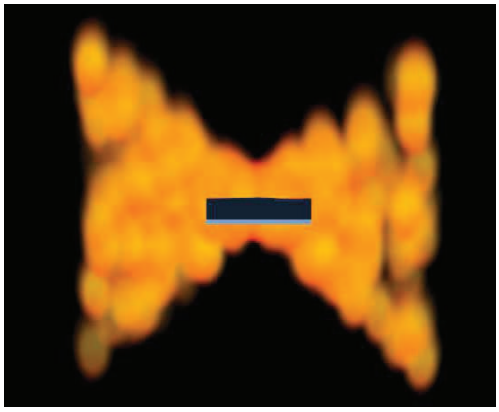


Imagen 5.14.cp.2.6: Forma de emisión en Región

En el código que sigue se puede observar como, al igual que en el caso anterior se pone la variable drag a 100 y seguidamente se define la región, compuesta en esta ocasión por cuatro vectores.

```
global mundo3D
global fuego

on mouseDown me
  fuego.drag = 100
  fuego.emitter.region = [vector(-4,0,4), vector(-4,0,-4), vector(4,0,-4), vector(4,0,4)]
end mouseDown
```

3.4. Crear el botón para poner la forma de emisión en trayectoria.

En esta forma, las partículas recorren un camino o trayectoria que es especificado punto por punto en el script.

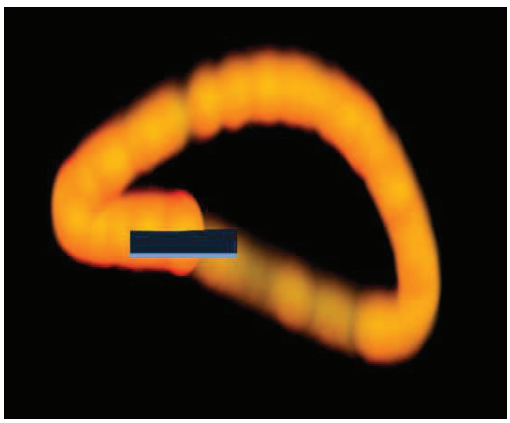


Imagen 5.14.cp.2.7: Forma de emisión en Trayectoria

En este caso, a la variable drag, se le suma pathstrengh para controlar que las partículas no se aparten demasiado del camino definido. Es una variable de difícil comprensión, por lo que podemos utilizar el método de prueba y error para ir entendiendo cómo funciona.

Por ejemplo, en el código que se muestra abajo, el cambiar la variable pathstrength del valor 1 al 3 hace que tengamos una especie de espiral, en vez del triángulo que se busca.

```
global mundo3D
global fuego

on mouseDown me
  fuego.emitter.region = [vector(0,0,0), vector(0,0,0)]      -- para reiniciar región
  fuego.drag = 5
  fuego.emitter.pathstrength = 3                              -- con 1 hace una espiral
  fuego.emitter.path = [vector(0,0,0),vector(-2,0,0),vector(0,0,2),vector(4,0,0),vector(0,0,-1)]
end mouseDown
```

4. Creación de los botones para control de características físicas.

Las características que vamos a trabajar son 4: el movimiento del punto de emisión de las partículas, el arrastre, la gravedad y el viento.

4.1. Crear los botones para trasladar el sistema de partículas.

En total tenemos 6 botones, ya que tenemos dos sentidos en cada uno de los tres ejes para mover el sistema de partículas como se puede ver en la imagen de abajo. Los botones se han creado para que cada vez que se pulsen, el sistema se traslade una cantidad discreta, en este caso una unidad.

Imagen 5.14.cp.2.8: Botones diseñados para mover el sistema de partículas

En el script que sigue se traslada el objeto “spmio” en una unidad hacia la izquierda mediante la función “translate”.

```
global mundo3D
global spmio

on mouseDown me
  spmio.translate(-1,0,0)
end mouseDown
```

“Spmio” es un modelo del mundo 3D que tiene como objeto alojar el sistema de partículas, por lo tanto, el propio sistema de partículas debe haber sido creado con anterioridad. En el script “Programación”, que como sabemos es asignado al sprite que contiene el mundo tridimensional, se tendrá que introducir una sentencia del tipo:

```
spmio = mundo3d.newmodel("loquesea", fuego)
```

Esta sentencia solo hace falta una vez por botón, por tanto, el script deberá repetirse para cada uno de los seis botones de traslación modificando únicamente las coordenadas de “translate”.

4.2. Crear el botón deslizante para controlar el arrastre del sistema de partículas.

El arrastre mide el tanto por ciento de partículas que se verán afectadas por otras magnitudes como el viento o la gravedad, por ello, su rango de valores va de 0 a 100. El código asignado al botón deslizante es el siguiente:

```
on mouseUp me
  valor=sprite(slider).locH
  arrastre=(valor-812)/12.7
  if (arrastre=0) then arrastre = 1
  fuego.drag=arrastre
end mouseup
```

En la imagen que sigue se puede apreciar cómo afecta al viento constante aplicado al sistema de partículas con arrastres de distinto valor.

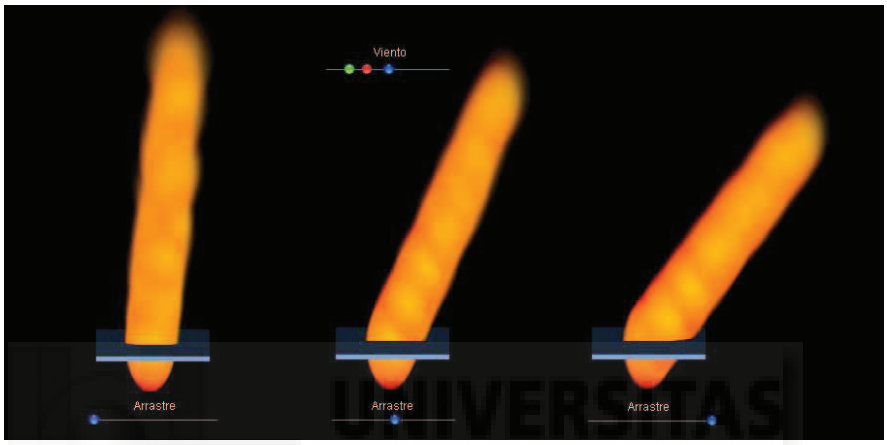


Imagen 5.14.cp.2.9: Distintos arrastres para un mismo viento

Recordamos que se trata de un botón deslizador y que toda la funcionalidad de estos botones, un tanto compleja está explicada ya en otras prácticas, por lo que solo hacemos mención a la instrucción de asignación que se hace directamente al objeto fuego y a la propiedad “Drag”.

4.3. Crear el botón deslizante para controlar la gravedad del sistema de partículas.

Vamos a implementar la interacción para aplicar gravedad mediante tres botones deslizantes pero sobre la misma línea para no saturar el interfaz del caso práctico. Cada botón deslizante controlará uno de los tres ejes en que podemos definir la gravedad. El botón azul es el que controla el eje X, el rojo el eje Y, y el de color verde el eje Z. A destacar que se cambian los signos de las magnitudes ya que se asocia la gravedad a “caída” de partículas y por tanto, el signo en el eje Y es normalmente negativo.

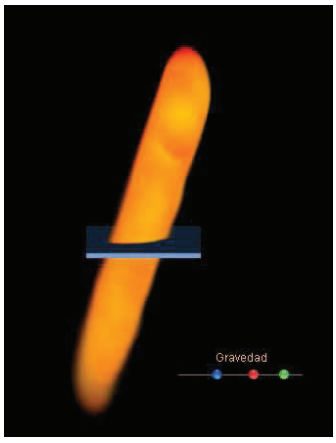


Imagen 5.14.cp.2.10: Sistema de partículas afectado por Gravedad

Por ejemplo, el código para el botón azul es:

```
on mouseUp me
  valor=sprite(slider).locH
  xg=(valor-812)/127.0
  fuego.gravity=vector(-xg,-yg,-zg)                    //definimos la gravedad
end mouseup
```

4.4. Crear el botón deslizante para controlar el viento del sistema de partículas.

Cómo se puede apreciar en la imagen 5.14.cp.2.9, el viento se puede generar con unos botones deslizantes superpuestos como los empleados para la gravedad, ya que en el fondo definen lo mismo, esto es: un vector en tres dimensiones. En este caso las cantidades no las hemos cambiado de signo, ya que el viento se asimila a algo por encima del foco de emisión. El código de colores es el mismo que en la gravedad. Por tanto, el código para el botón color azul quedará así:

```
on mouseUp me
  valor=sprite(slider).locH
  xv=(valor-812)/12.7
  fuego.drag=100
  fuego.wind=vector(xv,yv,zv)                          //definimos el viento
end mouseup
```

Antes de definir el viento con la propiedad wind, definimos una cantidad de arrastre, en este caso 100 para que se muestre bien el efecto del viento.

Conclusiones:

Las conclusiones que podemos obtener es que los sistemas de partículas debemos considerarlos como sistemas dinámicos. Es decir, si no entendemos que la dinámica de las partículas modifica totalmente el aspecto de todo el sistema, estaremos en un error. La dinámica la debemos entender como los cambios que van a afectar a una determinada partícula en el camino recorrido, y además como forma parte de un conjunto mayor que es el propio sistema.

En esta práctica en particular tenemos que tener en cuenta que hay características que nos permiten modificar el comportamiento de manera sencilla. Por un lado definiendo vectores que afectan a las partículas y por otro definiendo dónde se van a encontrar esas partículas en un momento determinado. Por lo tanto, es una buena herramienta para crear desde realismo hasta los efectos visuales más novedosos.

Caso práctico 14.3. Sistema de Partículas, creación de efectos atmosféricos

Objetivo: En esta tercera práctica sobre los sistemas de partículas vamos a realizar dos casos de efectos atmosféricos: lluvia y nieve. No se trata de conseguir un efecto fotorealista, sino de aprender cómo se han realizado y por tanto, la didáctica prevalece con respecto al resultado.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Creación del mundo 3D de la llanura helada.
2. Programación del sistema de partículas para la lluvia.
3. Programación del sistema de partículas para la nieve.

1. Creación del mundo 3D de la llanura helada.

Los dos sistemas de partículas que vamos a realizar comparten una misma escena 3D que hemos creado en 3D Studio MAX. En este caso se trata de un simple plano de color blanco y un fondo azul como se puede ver en la imagen siguiente.

Imagen 5.14.cp.3.1: Vista de la práctica que se propone terminada

Como la realización de este fichero en MAX carece de dificultad, no lo comentamos en detalle. Su exportación a Director tampoco tiene nada que reseñar.

2. Programación del sistema de partículas para la lluvia.

2.1. Realización de la textura de lluvia.

Para crear la textura de lluvia, simplemente se ha realizado en PhotoShop una textura degradada y luego se ha “recortado” mediante el diseño de un canal alpha con la forma de una “lágrima” o gota cayendo. De esta forma se pueden crear fácilmente muchos diseños si cambiar los colores básicos. El resultado es el que se aprecia en la siguiente imagen.

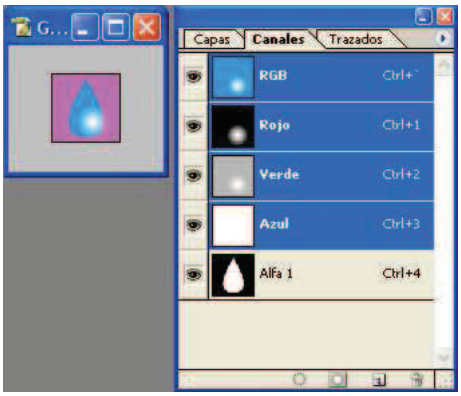


Imagen 5.14.cp.3.2: Cómo se diseño la partícula de lluvia

En la siguiente imagen se puede apreciar el sistema de partículas de lluvia. Se pueden apreciar las gotas cayendo.

Tenemos que añadir que aprovechamos el fuego original que creamos en los casos prácticos precedentes modificando dos parámetros: la textura y la gravedad. Para la textura se aplicó también la gota de agua y la gravedad se modificó para que fuera hacia abajo. La fuente se aprecia en el centro de la llanura.

Imagen 5.14.cp.3.3: Vista del sistema de partículas lluvia

2.2. El script que crea los sistemas de partículas.

El siguiente script crea dos sistemas de partículas: el de la lluvia y el de la fuente. Las principales sentencias están comentadas.

De todas formas, no está demás señalar que la principal diferencia entre un sistema y otro (y en realidad la razón para aplicar la misma textura en los dos), es la forma de emisión. La fuente tiene un único punto de emisión, por tanto, utiliza la emisión normal, la emisión por defecto. Sin embargo, la nieve utiliza como forma de emisión una zona rectangular que simula las nubes, y por ello, las partículas de gota caen desde un plano superior hacia abajo.

```
global mundo3D
property psprite
property txt

property otratxt
property contador
```

```
on beginSprite me

  pSprite = sprite(me.spriteNum)

  txt = mundo3d.newtexture("perla", #fromcastmember, member("gotaagua_2"))
  txt.renderformat = #rgba8888

  otratxt = mundo3d.newtexture("llamita", #fromcastmember, member("gotaagua_2"))
  otratxt.renderformat = #rgba8888

  -- Sistema de particulas LLUVIA
  parti = mundo3d.newmodelresource("particulas1", #particle)

  parti.emitter.maxspeed = 100
  parti.emitter.numparticles=20000
  parti.sizerange.start = 1
  parti.sizerange.end = .3
  parti.lifetime = 10000

  parti.emitter.region = [vector(-30,50,0), vector(-30,-50,0), vector(30,-50,0), vector(30,50,0)]
  parti.gravity=vector(0,0,-.1)
  parti.wind=vector(1,0,0)

  parti.texture = txt

  particul = mundo3d.newmodel("explode", parti)
  particul.translate(0,0,20)

  -- Sistema de particulas FUENTE  basado en el de FUEGO
  fuego = mundo3d.newmodelresource("particulas2", #particle)

  fuego.emitter.maxspeed = 1
  fuego.emitter.numparticles=100
  fuego.sizerange.start = 2
  fuego.sizerange.end = 1
  fuego.blendrange.end = 10
  fuego.lifetime = 750

  fuego.gravity=vector(0,0,-.5)

  fuego.texture = otratxt

  llamaradas = mundo3d.newmodel("lala", fuego)
  particul.translate(0,0,70)

end
```

3. Programación del sistema de partículas para la nieve.

3.1. Realización de la textura de nieve.

Para crear la textura de nieve, solamente tuvimos que “recortar” una zona blanca con un degradado circular para realizar un copo muy simple. El resultado es el que se aprecia en la siguiente imagen.

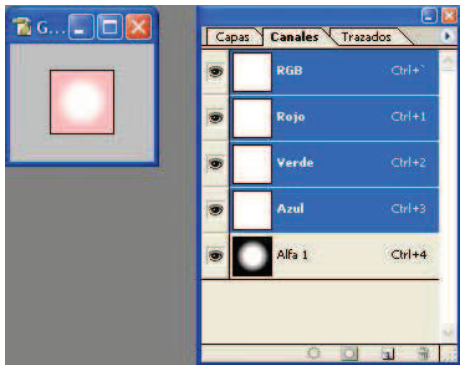


Imagen 5.14.cp.3.4: Cómo se diseño la partícula de nieve

En la imagen inferior se puede apreciar el resultado. Como la captura de pantalla se tomó al poco de iniciarse el sistema, se han generado pocos copos.

Además se observa que el sistema de partículas de fuego creado anteriormente no se ha modificado y así, nos sirve de referencia y podemos apreciar que los copos caen en profundidad, unos más cerca y otro más alejados de la vista de cámara.

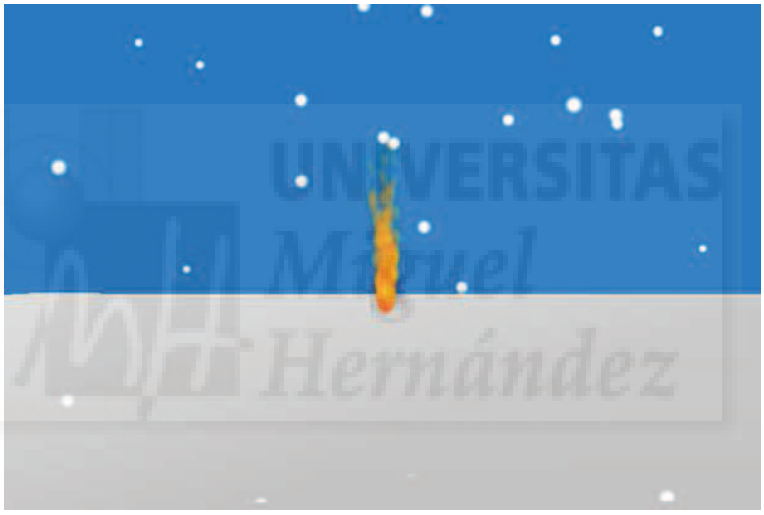


Imagen 5.14.cp.3.5: Vista del sistema de partículas nieve

3.2. El script que crea los sistemas de partículas es el siguiente:

```
global mundo3D
global fuego
property psprite
property txt

property otratxt

property contador

on beginSprite me

  pSprite = sprite(me.spriteNum)

  txt = mundo3d.newtexture("perla", #fromcastmember, member("copo_nieve"))
  txt.renderformat = #rgba8888

  otratxt = mundo3d.newtexture("llamita", #fromcastmember, member("llama"))
```

```
  otratxt.renderformat = #rgba8888

  -- Sistema de particulas NIEVE
  parti = mundo3d.newmodelresource("particulas1", #particle)

  parti.emitter.maxspeed = 1
  parti.emitter.numparticles=100
  parti.sizerange.start = 1
  parti.sizerange.end = 1
  parti.lifetime = 10000

  parti.emitter.region = [vector(-30,50,0), vector(-30,-50,0), vector(30,-50,0), vector(30,50,0)]
  parti.gravity=vector(0,0,-.1)
  parti.wind=vector(1,0,0)

  parti.texture = txt

  particul = mundo3d.newmodel("explode", parti)
  particul.translate(0,0,20)

  -- Sistema de particulas FUEGO
  fuego = mundo3d.newmodelresource("particulas2", #particle)

  fuego.emitter.maxspeed = 1
  fuego.emitter.numparticles=100
  fuego.sizerange.start = 2
  fuego.sizerange.end = 1
  fuego.blendrange.end = 10
  fuego.lifetime = 750
  fuego.gravity=vector(0,0,0.5)

  fuego.texture = otratxt

  llamaradas = mundo3d.newmodel("lala", fuego)

end
```

Conclusiones:

En este caso práctico hemos querido poner en evidencia, que todo el sistema cambia mucho a la mínima modificación. Por ejemplo, en el caso de la lluvia, los sistemas son prácticamente los mismos, pero pasamos de tener una fuente a un efecto atmosférico con solamente dos sentencias modificadas.

También podemos añadir que realizar otros efectos atmosféricos más realistas es cuestión de aumentar el número de partículas, texturizar con más fotorealismo y utilizar un paisaje más elaborado. Si quisiéramos abarcar más espacio, bastaría con aumentar la magnitud de los vectores que definen la zona de emisión.

Unidad 15: Simulaciones físicas en Director.

Introducción teórica:

1. Introducción a las simulaciones físicas.
2. Métodos para realizar la detección de colisiones en Director.
3. El modificador Collision.
4. Programación de límites.
5. La máquina física PhysX engine.

Casos prácticos:

15.1. Simulación física con PhysX.

1. Introducción a las simulaciones físicas.

Las máquinas físicas o motores físicos son módulos de software encargados de una función en concreto: simular el mundo físico en las escenas tridimensionales. Esto quiere decir que tenemos dos niveles de relación entre los modelos en una escena 3D. Por una parte, tenemos el mundo 3D con sus componentes (o nodos como los denomina Director) y estos se interrelacionan entre sí. Por ejemplo, cada modelo se relaciona con su propio sombreador (shader) y este con las luces, etc. Por otra parte, podemos tener un software "especial" que es capaz de simular fuerzas del mundo real como son la gravedad, los choques, la inercia, el rozamiento, etc. que se encargan de calcular los movimientos, las trayectorias, las aceleraciones que luego aplican a los modelos 3D y por tanto simulan el comportamiento dinámico que se da en la realidad. Estos motores físicos, son por tanto, imprescindibles en sistemas en 3D que traten de recrear la realidad, como videojuegos, metaversos y otros. En el mercado existen diversos motores físicos. Director hasta la versión 8.5 utilizó el conocido motor Havok de Intel. Este motor físico es quizá el más utilizado hasta la fecha y por ejemplo, lo soportan 3D Studio MAX, Second Life y multitud de videojuegos. Tenemos que añadir que el hecho de que Havok esté integrado en MAX no es de mucha ayuda, ya que si queremos aportar realismo a nuestras obras tendremos que tratar las simulaciones físicas en Director para que estas se produzcan de forma interactiva y no desde MAX. Por ejemplo, supongamos que queremos realizar que si alguien tira un jarrón desde una altura, este caiga por efecto de la gravedad y luego se rompa. Para hacer esto, tenemos que implementarlo en Director.

Imagen 5.15.1: Máquina física Havok

Director 11 utiliza PhysX engine de la empresa Nvidia. Este motor también es utilizado por empresas de la competencia de Director como "Unity 3D" y muchas mas. PhysX engine es un motor más moderno que Havok y tiene características novedosas sobre todo en cuanto a efectos especiales. Es muy potente debido a que está especialmente diseñado para su ejecución en las tarjetas gráficas Nvidia. Se supone que el rendimiento es mayor que el motor Havok debido a que descarga de cálculos a la CPU y los realiza en el GPU o Unidad de Procesamiento de Gráficos de estas tarjetas.

Imagen 5.15.2: Máquina física PhysX

Quizá la simulación física más utilizada y que es normalmente lo que primero busca un autor de mundos virtuales es la detección de colisión, es decir, que el sistema detecte y resuelva cuando dos modelos 3D, o dos componentes cualquiera, colisionan entre sí. Este tema es el que vamos a estudiar a continuación.

2. Métodos para realizar la detección de colisiones en Director.

Como hemos escrito anteriormente, la detección de colisiones es la simulación física más utilizada. Vamos a ver como podemos realizar esta detección en Director. Existen tres formas de atacar este problema.

La primera forma es utilizar un modificador que aporta Director llamado “Collision” y que podemos aplicar a cualquier modelo 3D. Este método está muy en desuso ya que tiene muchos problemas sobre todo porque ralentiza mucho la ejecución del programa.

La segunda opción es programar directamente nosotros la detección de colisiones. Esta solución tiene una ventaja principal y es que tendremos un control total sobre lo que hacen los modelos 3D y también resuelve el problema que presentaba la primera opción, ya que su ejecución es rápida. Pero también tiene múltiples desventajas entre las que se encuentran que requiere de amplios conocimientos en programación y de métodos específicos de búsqueda de límites, etc. Además de no ser un método reutilizable directamente ya que depende de las formas de los modelos 3D.

El tercer método y el más utilizado es el uso de un motor físico como el que aporta Director que es PhysX. Esta opción resuelve los problemas anteriores porque es rápido y relativamente fácil de usar aunque depende un poco del hardware, ya que al ser un producto de una determinada marca comercial, su funcionamiento se puede ver mejorado para algunas tarjetas gráficos, lo cual no le quita universalidad.

3. El modificador Collision.

Director, entre otros modificadores que ya hemos estudiado, proporciona “Collision”, que tiene como objetivo, la detección de colisiones de modelos individuales.

Collision deberá aplicarse a cada modelo individual que pueda verse involucrado en una colisión que queramos detectar y gestionar. Para añadir el modificador Collision a la lista de modificadores de un modelo, escribiremos en Lingo:

```
Scene.model[x].addmodifier(#collision)
```

Es evidente que para detectar una colisión entre dos objetos, los dos tendrán que haber sido modificados con Collision.

Collision tiene cuatro propiedades para funcionar que son:

1. Enabled. Esta propiedad solo puede tener dos valores: verdadero o falso. Siendo verdadero su valor por defecto. Sirve para activar o no Collision en ciertos momentos.

2. Resolve. Esta propiedad solo puede tener dos valores: verdadero o falso. Siendo verdadero su valor por defecto. Sirve para enviar o no al manejador de Collision la información sobre la colisión, es decir, los modelos colisionados, el ángulo de colisión, etc.

3. Inmovable. Esta propiedad solo puede tener dos valores: verdadero o falso. Siendo falso su valor por defecto. Si vale verdadero indica que el modelo no es animado. Por ejemplo, si el modelo es un muro que define el límite de una carretera en un juego de carreras, deberá tener el valor verdadero, ya que el muro no se mueve aunque los coches si.

4. Mode. Esta propiedad sirve para definir la geometría que se comprobará por parte de Collision con relación a las colisiones y que difiere de la geometría real del objeto. Su función es minimizar el tiempo de cálculo y por tanto la respuesta del modificador. Puede tener tres valores que son:

4.1. #sphere: hace que collision suponga que el cuerpo es esférico. Por lo tanto, funciona bien con balones, balas, cabezas, planetas, etc.

4.2. #box: hace que collision suponga que el cuerpo es de tipo caja. Por lo tanto, funciona bien con planos, mesas, raquetas, etc.

4.3. #mesh: hace que collision calcule el cuerpo tal cual es. Por lo tanto, debemos aplicar este valor cuando no tengamos más opción y para cuerpos especialmente complicados como puede

ser un personaje bípedo. Esta opción ralentiza tanto la ejecución, que a veces, aunque sea más impreciso, es conveniente utilizar cualquiera de las otras dos alternativas.

Programar Collision es más sencillo que hacerlo directamente, aunque su lentitud lo hace especialmente desaconsejable para los requerimientos de un videojuego. En nuestro caso, se podría utilizar para ciertas obras artísticas, pero, en principio, no lo vamos a implementar.

4. Programación de límites.

La programación de límites y por tanto, de colisiones es abordable debido al alto nivel de Lingo, que proporciona sentencias, variables, objetos y métodos tan buenos que podemos aventurarnos a programar nosotros mismos.

La detección de colisión se puede calcular de muchas maneras, pero la más sencilla es la gestión de límites. Este método está basado en el concepto de plano de seguridad. El plano de seguridad será un plano que esté delante del objeto que se mueve y que se encuentra a cierta distancia delante de él. El objeto o partes de él lanzarán “rayos” hacia delante hasta el plano de seguridad. Estos rayos son del mismo tipo de sentencias que vimos para programar la detección de objetos cuando estudiamos la interactividad.

Si los rayos lanzados colisionan con el plano de seguridad, es que no existe ningún objeto en el espacio que separa al objeto móvil y al plano, y por tanto no va a existir una colisión a corto plazo. Este método obliga a aumentar o disminuir el espacio entre el objeto y su plano de seguridad según su velocidad, ya que cuando la velocidad sea mayor, también mayor tiene que ser la distancia con el plano de seguridad.

Otro tema será detectar el ángulo de la colisión. Para realizar este objetivo, se utilizan varios rayos, normalmente dos para saber desde dónde puede venir la colisión.

5. La máquina física PhysX engine.

Como hemos escrito anteriormente PhysX en una máquina física que está incorporada en Director 11.

Esta máquina física es muy potente y fácil de utilizar en la plataforma de Director pero esta tarea solamente la podemos llevar a cabo con programación.

Lo primero que debemos hacer es incorporar PhysX a nuestro proyecto. Para ello, basta con insertar desde el menú de Director un “Media Element” de tipo physX. Luego debemos de inicializar el simulador. Las sentencias que realizan esta tarea las estudiaremos con detalle en el caso práctico adjunto.

Para iniciar el simulador físico utilizaremos el método Init() que requiere 4 variables, tres de ellas sirven para controlar el modo en que el simulador controla el tiempo a intervalos. La otra variable sirve para definir el factor de escala con el que trabaja el simulador y por tanto es una medida del tamaño físico de este. Un ejemplo inicio del simulador físico puede ser así:

```
fisica.Init(member(mundo3D), vector(1,1,1),#equal, intervalo, subintervalo )
```

Otros métodos del mundo físico son los siguientes:

- angularDamping
- contactTolerance
- friction
- gravity
- IsInitialized
- linearDamping
- restitution

- scalingFactor
- sleepMode
- sleepThreshold
- subSteps
- timeStep
- timeStepMode

Debemos destacar el método gravity, o sea, gravedad, ya que, por supuesto, tienen implicaciones muy importantes en el comportamiento de todo el sistema. La gravedad se define mediante un vector. Un ejemplo de definición de la gravedad terrestre sería así:

```
fisica.gravity = vector(0, -9.8, 0)
```

En PhysX, los cuerpos rígidos se generan a partir de los modelos 3D añadiéndoles el modificador #meshdeform y luego se utiliza el método createRigidBody() para su creación. En esta función se definen dos parámetros muy importantes. El primero es la forma básica del cuerpo rígido que puede tomar 4 valores: caja, esférico, convexo o cóncavo. El segundo define si el cuerpo será estático o dinámico. Un ejemplo de creación de un cuerpo rígido y dinámico puedes ser el siguiente:

```
cuerpo.addmodifier(#meshdeform)
sphereRigidBody = fisica.createRigidBody(pelota.name,cuerpo.name,#sphere,#dynamic)
```

Luego debemos especificar algunas propiedades para cada uno de los cuerpos que gobernará el simulador. Al menos definiremos “restitution” y “mass”.

“Restitution”, representa el coeficiente de restitución del cuerpo rígido. El coeficiente de restitución es una medida del grado de conservación de la energía cinética en un choque entre dos cuerpos. Su rango de valores va de 0 a 1, aunque un valor de 1 es imposible desde el punto de vista físico, ya que en un choque entre cuerpos siempre se pierde energía.

“Mass” representa la masa del cuerpo medida en kgr.

Por tanto, podemos escribir las siguientes características para un cuerpo rígido:

```
sphereRigidBody.restitution = 0.5
sphereRigidBody.mass = 1
```

Otras propiedades de los cuerpos rígidos que podemos emplear para programar nuestro simulador pueden ser:

- angularDamping
- angularMomemtum
- angularVelocity
- friction
- gravity
- mass
- restitution
- linearDamping
- orientation
- position
- centerOfMass

El sistema debe llamar a la función simulate() cada vez que salga de cada frame dónde se aplique el simulador.

También debemos utilizar la función destroy() si deseamos detener el simulador y liberar así todos los recursos que consume.

La siguiente imagen muestra tres momentos de una simulación física dónde una pelota choca contra el suelo y rebota hacia arriba, pero al perder energía no sube a la misma altura de la que partió. Al repetirse este proceso, la pelota termina por detenerse. Es el resultado del caso práctico 15.1.

Imagen 5.15.2: Simulación física con PhysX en el caso práctico

Como conclusión a este tema podemos añadir que si necesitamos una simulación potente y sólida, debemos de utilizar physX. Esta máquina física es sencilla de manipular y sus resultados son profesionales, por lo que merece la pena el aprendizaje sobre todo si buscamos generar en nuestras obras una simulación realista.

Caso práctico 15.1: Simulación física con PhysX.

Objetivo: Realizar una demostración de las posibilidades de la máquina física PhysX utilizando cuerpos dinámicos y estáticos. En este caso, simularemos el bote de una pelota sobre el suelo.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Crear los miembros Shockwave 3D y PhysX.
2. Incorporación de la máquina física al mundo tridimensional.
3. Crear la pelota o cuerpo dinámico.
4. Crear el suelo o cuerpo estático.
5. Comprobar la simulación física.

1. Crear los miembros Shockwave 3D y PhysX.

El caso práctico que nos ocupa es bastante especial ya que el trabajo más importante lo tenemos que realizar obligatoriamente por programación. Por ello, hemos decidido que lo realizaremos totalmente de esta forma, o sea, programando.

Imagen 5.15.cp.1.1: El caso práctico en ejecución

1.1. Crear la película básica.

Para mantener el formato de nuestra película que hemos empleado en todos los demás casos prácticos, utilizaremos, dos fotogramas llamados “Inicio” y “Principal” y los comportamientos asociados ya conocidos de “Inicialización”, “Bucle” y “Programación”.

Si tenemos alguna duda, o no queremos comenzar desde el principio, podemos utilizar el resultado del caso práctico 8.1, “Estructura básica de la película”.

1.2. Crear Shockwave 3D.

1.2.1. Menú Insert > Media elements > Shockwave 3D.

Con este comando añadimos un miembro a la película de Director que contiene un mundo tridimensional ya que no lo importamos desde MAX como en anteriores ocasiones. Lo creamos directamente y estará totalmente vacío.

1.2.2. En la ventana del Cast. Nombrar el elemento Shockwave 3D como “mundo”.

1.2.3. Acceder al behavior “Iniciación” y modificar si es necesario, la sentencia que referencia al mundo 3D. Debe de quedar así:

```
mundo3D = member("mundo")
```

1.3. Crear PhysX.

1.3.1. Menú Insert > Media elements > PhysX.

1.3.2. En la ventana del Cast. Nombrar el elemento PhysX como “maqfisica”.

2. Incorporación de la máquina física al mundo tridimensional.

Esta es la parte más delicada de este caso práctico. Tenemos que hacer que el mundo 3D de Shockwave trabaje conjuntamente con la máquina física PhysX. Para realizar esto, debemos de definir una serie de parámetros imprescindibles para PhysX y luego iniciar la máquina.

Los parámetros que necesita PhysX están relacionados con el intervalo de tiempo en que trabaja, sería algo así como saber cada cuanto tiempo debe actualizar todo el simulador. Por otra parte, también necesita conocer cual será su unidad de medida. Esto lo solucionamos definiendo una unidad de escala.

2.1. Definir las variables locales. Además de una referencia al mundo tridimensional, necesitamos las tres variables que escribimos a continuación para definir los parámetros del simulador:

```
global mundo3D
property intervalo
property subintervalo
property fisica
```

Toda la programación que sigue hasta la definición de la gravedad para el simulador físico la escribiremos en el manejador “beginSprite”. Para una mayor claridad de la estructura final, leer directamente el todo el behavior “Programación” escrito en el último punto de este caso práctico.

2.2. Definir los intervalos del simulador físico. Esto lo vamos a realizar definiendo tres variables que se comentan en rojo:

```
pTimeStepMode = #equal          -- El intervalo base se especifica por el programador con #equal
intervalo = 0.33                -- En modo #equal, define el valor del intervalo base.
subintervalo = 5                -- Es el valor de subintervalos para funciones con precisión
```

2.3. Iniciar la máquina física. Esto requiere de dos sentencias, añadir el miembro a la película interactiva e inicializarlo dentro de Shockwave 3D como sigue:

```
fisica = member("maqfisica")
fisica.Init(member(mundo3D), vector(1,1,1),#equal, intervalo, subintervalo )
```

2.4. Dotar al simulador de propiedades físicas. Solo programaremos que tenga una gravedad parecida a la de la tierra.

```
fisica.gravity = vector(0,-10,0)            -- Definimos la gravedad
```

2.5. Cuando salimos del sprite del mundo 3D, debemos destruir el simulador. Para ello, después de beginSprite escribiremos:

```
on endSprite me
  if fisica.isInitialized then
    fisica.destroy()
  end if
end
```

2.6. Con el código que sigue hacemos que el simulador funcione de manera continua.

```
on exitFrame me
    fisica.simulate(intervalo, subintervalo)
end
```

3. Crear la pelota o cuerpo dinámico.

En este caso práctico, los dos modelos 3D que utilizamos son simples. Por ello, no los creamos en MAX como hemos venido haciendo, sino que los creamos directamente con Lingo en Shockwave 3D.

3.1. Crear una esfera de radio 30 y nombre “pelota”. Para ello, escribimos el código:

```
pelotamod = mundo3D.newModelResource("Esfera", #sphere)
pelotamod.radius = 30
pelota = mundo3D.newModel("Esfera", pelotamod)
```

Seguidamente, vamos a aplicar una imagen o textura sobra la superficie de la esfera recién creada. Por lo tanto, necesitamos crear e importar esta textura, proceso que ya hemos realizado en numerosas ocasiones anteriores. El paso siguiente presupone que ya tenemos la textura en el cast de nuestra película y su nombre es “bp”.

3.2. Texturizar la pelota con el miembro imagen “bp”. Para ello, escribimos el código:

```
mundo3D.newTexture("Texture01",#fromImageObject,member("bp").image)
newShader01 = mundo3D.newShader("newPainter01",#standard)
newShader01.texture = mundo3D.texture("Texture01")
pelota.shader = newShader01
```

3.3. Para que la esfera se vea influenciado por las fuerzas aplicadas a la máquina física escribimos el siguiente código:

```
pelota.addmodifier(#meshdeform)
sphereRigidBody = fisica.createRigidBody(pelota.name,pelota.name,#sphere,#dynamic)
sphereRigidBody.restitution = 1
sphereRigidBody.mass = 1
```

En realidad, la única fuerza que existe en el mundo físico definido anteriormente es la gravedad. En la primera línea de código se le añade el modificador “meshdeform” a la pelota, lo que la define cómo componente del mundo físico. La siguiente línea define el modo en que será evaluado por el sistema, es decir, como un cuerpo rígido y dinámico.

La tercera línea, aplica un coeficiente de restitución de 1 a la pelota. El coeficiente de restitución es una medida del grado de conservación de la energía cinética en un choque entre dos cuerpos. Su rango de valores va de 0 a 1. Esto quiere decir que la pelota después de botar en el suelo debido a la gravedad, no pierde energía, y por lo tanto, volverá a la altura de la que cayó. Esto, evidentemente, no es cierto, por lo tanto, para tener un sistema más real, podemos bajar este valor a 0.5.

La última línea de código, define la masa de la pelota de 1 Kgr.

4. Crear el suelo o cuerpo estático.

La creación de suelo es prácticamente igual que la creación de la esfera, salvo que se trata de un modelo tipo caja y que no será dinámico sino estático.

4.1. Creación del modelo suelo.

```
suelomod = mundo3D.newModelResource("suelomio", #box)
suelomod.height = 10
suelomod.width = 180
suelomod.length = 5
suelo = mundo3D.newModel("suelomio", paredmod)
```

4.2. Trasladar el modelo “suelo” a la parte inferior de la vista del mundo 3D.

```
suelo.transform.identity()
suelo.transform.translate(0,-75,0)
```

4.3. Texturizar el suelo con el miembro de tipo imagen llamado “madera”.

```
mundo3D.newTexture("Texture02",#fromImageObject,member("madera").image)
mundo3D.model("suelomio ").shader.texture= mundo3D.texture("Texture02")
```

4.4. Definir “suelo” como un cuerpo rígido y estático. Añadimos que su coeficiente de restitución es de 1 y su masa de 100 Kgr.

```
suelo.addmodifier(#meshdeform)
floorRigidBody = fisica.createRigidBody(suelo.name, suelo.name,#box,#static)
floorRigidBody.restitution = 1
floorRigidBody.mass = 100
```

5. Comprobar la simulación física.

Para realizar la comprobación, tenemos que ejecutar la película y ver el resultado. Luego, debemos de cambiar la masa y restitución de la pelota para ver cómo trabaja la máquina física physX.

Por ejemplo, se pueden utilizar coeficientes de restitución de entre 0.5 y 1 para ver cómo la pelota va rebotando de distinta forma, perdiendo a cada choque con el suelo, más energía hasta que se detiene. El hecho de pararse no sucederá si dejamos el valor de restitución a 1, cosa que en la vida real no existe.

También podríamos jugar con el valor de la gravedad para ver el efecto que tendríamos por ejemplo, en la luna.

Para que quede todo más claro, en las siguientes líneas de código se exhibe todo el behavior “Programación” del caso práctico:

```
global mundo3D

property intervalo
property subintervalo
property fisica

on beginSprite me

  mundo3D.resetWorld()
  -- Los intervalos del mundo físico pueden ser automático
  -- o especificado por el programador con la   opción #equal
  pTimeStepMode = #equal

  -- Si usamos el modo #equal para el tiempo de intervalo,
  -- especifica el intervalo de tiempo para el mundo físico.
  intervalo = 0.33

  -- El número de pasos secundarios para cada llamada al simulador físico
```

```
  subintervalo = 5

  -- Iniciar al maquina física
  fisica = member("maqfisica")
 fisica.Init(member(mundo3D), vector(1,1,1),#equal, intervalo, subintervalo )

 -- Definimos la gravedad
 fisica.gravity = vector(0,-1,0)

 -- crear la pelota
 pelotamod  = mundo3D.newModelResource("Esfera", #sphere)
 pelotamod.radius = 30
 pelota = mundo3D.newModel("Esfera", pelotamod)

 -- Añadimos un sombreador con textura al modelo de pelota
 mundo3D.newTexture("Texture01",#fromImageObject,member("bp").image)
 newShader01 = mundo3D.newShader("newPainter01",#standard)
 newShader01.texture = mundo3D.texture("Texture01")
 pelota.shader = newShader01

 -- Creamos un cuerpo rígido dinamico para la pelota
 pelota.addmodifier(#meshdeform)
 sphereRigidBody = fisica.createRigidBody(pelota.name,pelota.name,#sphere,#dynamic)
 sphereRigidBody.restitution = 0.5
 sphereRigidBody.mass = .1

 -- Crear el suelo
 suelomod        = mundo3D.newModelResource("suelomio", #box)
 suelomod.height   = 10
 suelomod.width    = 180
 suelomod.length   = 5
 suelo = mundo3D.newModel("suelomio", suelomod)

 -- Colocar el suelo en su sitio
 suelo.transform.identity()
 suelo.transform.translate(0,-75,0)

 -- Añadir un sombreador con textura para el modelo del suelo
 mundo3D.newTexture("Texture02",#fromImageObject,member("madera").image)
 mundo3D.model("suelomio ").shader.texture= mundo3D.texture("Texture02")

 -- Creamos un cuerpo rígido estático para el suelo
 suelo.addmodifier(#meshdeform)
 floorRigidBody = fisica.createRigidBody(suelo.name, suelo.name,#box,#static)
 floorRigidBody.restitution = 1
 floorRigidBody.mass = 100

end

on endSprite me
  if fisica.isInitialized then
    fisica.destroy()
  end if
end

on exitFrame me
  fisica.simulate(intervalo, subintervalo)
end
```

Conclusiones:

Las simulaciones físicas son muy interesantes y además pueden tener implicaciones muy interesantes para aplicar en obras artísticas. Es una herramienta, que como otras muchas de Director, solo es abordable si lo hacemos desde el lenguaje de programación Lingo. De todas formas, cómo se puede deducir de este caso práctico, las sentencias empleadas son fácilmente entendibles y abordables aunque no seamos unos programadores avanzados. Lo más apetecible de physX sea quizá las nuevas posibilidades de experimentación que supone.

Unidad 16: Integración multimedia en Director.

Introducción teórica:

1. Introducción a la multimedia en Director.
2. Medios de imagen en Director.
3. Medios de sonido en Director.
4. Medios de animación en Director.
5. Medios de vídeo en Director.
6. Streaming de audio y vídeo en Director.
7. Otros medios multimedia en Director.

Casos prácticos:

16.1. Medios multimedia.

1. Introducción a la multimedia en Director.

Lo primero que debemos tener en cuenta es que no es lo mismo la multimedia en Director que en Shockwave 3D. Como sabemos, Director es una plataforma precisamente para trabajar con objetos de muy distinto formato, esto es, multimedia. Si algo tiene Director es su gran capacidad para integrar todo tipo de medios de distinto tipo en un solo proyecto. Shockwave 3D es una máquina 3D, o sea, es una estructura de software que incluye protocolos con el hardware de tarjetas gráficas y otras utilidades para que podemos utilizar 3D dentro de Director. Por lo tanto, la multimedia en Director es algo totalmente factible y relativamente fácil de llevar a cabo. Pretendemos poder ver imágenes, oír sonidos, visualizar vídeos junto con otros medios, dentro del mundo 3D de Shockwave que a su vez está integrado en Director.

Director soporta muchos formatos de archivos, estos son algunos:

Formatos de imagen: más de 40 como GIF, BMP, JPEG, PSD, PNG, y TIFF.

Formatos de audio: MP3, SWA, AU, AIFF, WAV, WMA, y RA.

Formatos de video: SWF, FLV, DVD-Video, WMV, RM, MOV, AVI y otros.

Para ejecutar los distintos elementos multimedia tenemos dos opciones: podemos ejecutar archivos internos, es decir, members o archivos externos haciendo streaming. Los archivos externos más susceptibles de ser tratados con streaming son los de audio y vídeo ya que son los que ocupan más espacio, y por tanto, los más idóneos para gestionar fuera del propio archivo de Director, como archivos independientes. Estos archivos pueden estar en cualquier carpeta del disco duro o en Internet. En este último caso tendríamos que referenciarlos por su URL.

Si queremos ejecutar los archivos miembros, primero tendremos que importarlos. Esto no tiene ninguna dificultad. Hacemos Archivo > Importar y elegimos los archivos de imagen, sonido, etc. que queramos y formarán parte como castmembers del cast. Pero como hemos escrito antes, estos formatos los podemos usar en Director pero no en la escena 3D, no con Shockwave 3D, por lo menos directamente. Con sonidos que nos sirvan para crear una interface o un efecto, no tendremos excesivos problemas, ya que existen comportamientos que nos permiten usarlos directamente como un castmember más. El problema surge a la hora de exhibir una imagen o una animación que al fin y al cabo es una sucesión de imágenes dentro de una escena 3D. ¿Dónde la visualizaremos? ¿En qué soporte se verá? La solución más sencilla es utilizar la superficie de los objetos como lienzo para ver los formatos que se basan en imágenes. Es decir, utilizar un objeto 3D para texturizarlo con la o las imágenes que formen la multimedia a utilizar.

El problema que se plantea es que no todos los modelos 3D son aptos para tal fin, ya que por ejemplo, una esfera produce una deformación de las imágenes. Además, es como estar buscando una solución para una nueva versión del problema tratado en las superficies de los modelos cuando estudiamos las “texturas animadas por intercambio” en el caso práctico 12.6, cuestión que no es sencilla. Sin embargo, podemos adelantar, que casi todos los problemas de imágenes, animaciones y vídeos se solucionan de la misma manera: utilizamos el formato de flash SWF como base para aplicarlo como textura en objetos 3D. Debido a que controlar imágenes y animaciones es relativamente sencillo con ®Adobe Flash, tendremos un método uniforme para visualizarlos en nuestra escena 3D.

2. Medios de imagen en Director.

En el caso práctico “12.1. Asignación de texturas”, ya estudiamos cómo podemos incluir una imagen con formato JPG y texturizar con ella la superficie de un modelo 3D.

Basado en este caso práctico, podemos probar a utilizar otros formatos como gif, tiff, etc. No tendremos problemas. Se trata simplemente de importar las imágenes en el formato deseado y proceder de la misma manera.

El método se basa en importar una imagen de cualquier formato y entonces formará parte del cast o librería de componentes. Supongamos que se llama “imagen”. Lo primero que haremos es incorporarlo como una textura más a nuestra escena 3D. Usaremos el nombre de “teximagen” para la textura y el de “mundo3D” para la escena en tres dimensiones. En Lingo escribiríamos lo siguiente:

```
mundo3D.newtexture("teximagen",#fromcastmember,member("imagen"))
```

Y luego para asignarla a un determinado modelo cuyo sombreador supongamos que se llama “shader1”, utilizaríamos una sentencia así:

```
mundo3D.shader(“shader1”).texture=mundo3d.texture(“teximagen”)
```

Es así de simple: con la primera sentencia añadimos un recurso más a nuestra película, en este caso, una imagen en forma de textura. Podemos tener todas las texturas que queremos en nuestra escena 3D, las utilicemos o no. Con la segunda sentencia aplicamos esta textura en particular a un determinado modelo 3D, independientemente del formato utilizado.

Debemos añadir que si el formato es SWF, es decir, el formato publicable de las películas Flash, el método a seguir es el mismo, pero no texturizamos una animación, sino el primer fotograma de esta. Si no hemos definido un color de fondo en Flash, se tomará como una imagen con transparencias, o sea, lo que conocemos por canales alpha.

3. Medios de sonido en Director.

En las últimas versiones de Director, se han mejorado mucho las funciones de audio. Se soporta sonido envolvente de 5 canales. También se pueden crear efectos de audio en tiempo real así como el streaming de vídeo y por tanto, de audio en alta definición. Estas capacidades son muy interesantes para proyectos de alta calidad, aunque en un documento como este, nos vamos a referir al control básico del audio con el fin de poderlo integrar en nuestros trabajos. Esto no quiere decir que el sonido tenga que ser de baja calidad, vamos a trabajar con sonido estéreo de 44,1 Khz. En este punto estudiaremos la forma de integrar sonido “interno” que forme parte del archivo de trabajo como un member (miembro) mas. El streaming de audio lo veremos en el punto 6 de esta misma unidad. Para ejecutar sonidos dentro de nuestra escena 3D, procedemos como con las imágenes, esto se puede resumir en tres partes.

Primero importar el fichero correspondiente para tenerlo disponible como un member. Supongamos que pasa a formar parte del cast con el nombre de “sonido”.

En segundo lugar, queremos oír el sonido cuando se produzca alguna interacción del usuario que provoque esta ejecución. Para realizar esta acción, escribiremos en Lingo la siguiente sentencia:

```
sound(1).play(member("sonido"))
```

En este caso, el sonido no lo incorporamos dentro de la escena 3D. De hecho no se puede hacer, ya que no existe ningún nodo de tipo sonido en la máquina Shockwave 3D. Es decir, no existe este formato como tipo nativo en Shockwave 3D.

Esta sentencia utiliza el canal 1 para ejecutar un sonido. Director 11 tiene disponibles 8 canales para sonidos, por lo que podemos oír simultáneamente hasta 8 sonidos.

Por último, si queremos dejar de oír el sonido, podemos ejecutar la sentencia:

```
puppetsound(1,0).
```

Esta sentencia de Lingo suspende el sonido en el canal 1.

4. Medios de animación en Director.

Sabemos que las animaciones son una sucesión de imágenes. Para introducir animaciones en Director y poderlas reproducir en el entorno 3D, podemos utilizar como hemos escrito antes, el intercambio de imágenes controlado por Lingo que se encuentra documentado en el caso práctico 12.6. El problema de esta solución es que es un poco engorrosa, pues debemos de importar las imágenes que componen la animación una a una y además en una librería de componentes especialmente construida para ello. Si la animación es de tipo Flash, tenemos una parte del trabajo adelantado. El hecho de que una animación sea realizada con Flash no nos debe extrañar, ya que hoy en día es el software favorito de los diseñadores para realizar sus animaciones, aunque no hay que olvidar que aún existen muchas de estas animaciones en formato GIF. Por lo tanto, si la animación es GIF o de cualquier otro formato, podemos importarlas a Flash u optar por la solución ya estudiada. Si la animación es un SWF, es decir, el formato de publicación de Flash, seguiremos el método que se explica a continuación.

Flash y Director son programas muy unidos. Antes de 2005 eran los programas más conocidos de Macromedia, y en la actualidad ambos pertenecen a Adobe. Esto ha provocado que siempre se haya podido trabajar en Director con archivos Flash, pero no al revés, ya que el "player" de Director es mucho más potente que el player de Flash. Y hoy en día, esto se sigue produciendo. Si importamos un fichero SWF a Director lo podremos utilizar como una animación en la ventana de salida en 2D, lo cual nos puede venir muy bien para hacer llamativos botones. Si lo incorporamos a la escena 3D como una textura lo veremos como una imagen directamente, exactamente en el frame 1 de la película Flash. Pero nosotros desearíamos ver todos los frames de la película Flash texturizando un modelo 3D y para ello utilizaremos la propiedad posterFrame que selecciona un frame de una película Flash.

Un tema a abordar sería encontrar un script dónde poder controlar el paso de fotograma a fotograma de la película Flash. La solución, vuelve a ser, como en el caso práctico 12.6, el script "Bucle".

En las siguientes líneas se muestra cómo resulta el behavior "Bucle" en el caso práctico que ilustra este tema. "Contador" es una variable local que lleva la cuenta del fotograma que se debe visualizar en un momento determinado. "Anima" es la animación en formato SWF y "textura" es una textura ya creada e introducida en la escena 3D. Esta textura será la que tengamos asignada al modelo 3D en cuya superficie queremos representar la animación.

```
global mundo3d
property contador

on exitFrame me
  if (contador<10) then contador=contador+1
  else contador =1
  member("anima").posterFrame = contador
  mundo3d.texture("textura").image= member("anima").image.duplicate()
  go the frame
end
```

El script funcionaría como sigue: si "contador" no ha llegado al último fotograma, por ejemplo, en este caso 10 (es decir, que "anima" está compuesto por 10 fotogramas), el contador toma su valor + 1 y si no, vale 1 y por lo tanto, vuelve a comenzar a contar.

Luego hacemos que contador sea el fotograma dónde se posiciona "anima" y en la última sentencia duplicamos la imagen interna de "anima" y se la asignamos al valor "image" de la textura que se está representando en ese momento en particular.

La última sentencia "go the frame" es en realidad el bucle en sí, que está entre el manejador on exitFrame me. Esto quiere decir que este código se ejecuta según la velocidad asignada a Director, que es usualmente de 24 fotogramas por segundo, lo cual coincide perfectamente con una velocidad adecuada tanto para animaciones como para vídeos.

5. Medios de vídeo en Director.

Los vídeos son una sucesión de imágenes con una determinada velocidad de reproducción (normalmente de 24 imágenes por segundo), y normalmente, acompañadas de sonido. También podemos añadir que en la actualidad, un vídeo debe tener un tamaño y una resolución de calidad mínima.

Para ejecutar un vídeo con Lingo integrado en la escena 3D, no tenemos más remedio que acudir a las soluciones vistas en el punto 3 y 4 de este mismo tema, lo cual quiere decir que no tendremos problemas técnicos pero sí complicaciones derivadas de ejecutar las imágenes por un lado y el sonido por otra. Primero tendremos que separar el sonido de las imágenes, cuestión que podremos resolver de forma sencilla con terceros programas como ®Adobe ®Premiere u otros. Luego ejecutaremos el vídeo en cuestión sobre la superficie de un modelo 3D a la vez que el sonido.

6. Streaming de audio y video en Director.

La técnica del streaming permite la ejecución de archivos muy grandes desde el exterior del archivo que los muestra. De esta manera, el programa se aligera y además permite que el vídeo o audio se almacene en cualquier dirección local o de Internet mediante un direccionamiento URL.

Existe otra ventaja muy apreciada y es que la ejecución no requiere que el archivo haya sido descargado en su totalidad, sino que se puede ir ejecutando mientra va siendo transmitido. Esto hace que el espectador no tenga que esperar largos periodos de tiempo para empezar a disfrutar de un determinado contenido.

Director permite la ejecución de streaming de audio y vídeo pero Shockwave 3D no permite incorporarlo en las escenas tridimensionales. Por lo tanto, vamos a estudiar solamente como incorporar audio en streaming. Para incorporar streaming de audio a nuestros proyectos en 3D, primero debemos crear un miembro con esta característica. En el menú Insert > Media element tenemos la posibilidad de crear un “Shockwave Audio”. Esto hace que aparezca la caja de diálogo que aparece en la siguiente imagen.

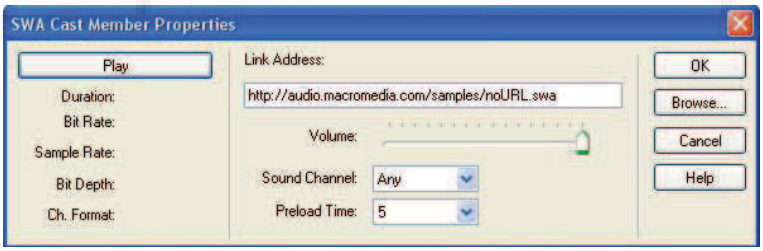


Imagen 5.16.1: Propiedades de cast member tipo SWA

En la caja de diálogo “SWA Cast Member Properties” tenemos que seleccionar la dirección del fichero a ejecutar. También podemos controlar el volumen, el canal de sonido utilizado y la memoria para el flujo (stream buffer) que viene con el nombre de “Preload Time”.

Una vez establecidos estos parámetros habremos creado un miembro nuestro en el cast Internal como se puede apreciar en la imagen inferior con el nombre “SBEP”.

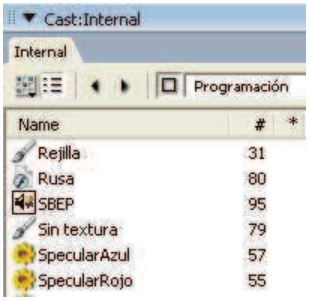


Imagen 5.16.2: Cast Internal con un componente streaming de audio

Para controlar este nuevo member no podemos utilizar los métodos empleados con un sonido "normal" ya que este no se encuentra directamente en nuestro archivo sino en un archivo externo. Por lo tanto utilizaremos los métodos play(), pause() y stop() para medios externos. Por ejemplo:

```
member("SBEP").play() -- ejecuta el sonido

member("SBEP").pause()        -- hace pausa en la ejecución del audio streaming

member("SBEP").stop()-- para detener el audio streaming
```

No tenemos problemas para ejecutar muchos sonidos en forma de streaming ya que como miembro se guarda solamente su referencia. Esto supone que el archivo que lo utiliza puede permanecer relativamente pequeño aunque tenemos que tener cuidado en que la referencias a los archivos externos permanezca idénticas cuando se realizan copias de seguridad, etc.

7. Otros medios multimedia en Director.

Los medios multimedia deben incluir por supuesto al texto. El problema es que el texto tal cual, con las propiedades normales, no tiene cabida en una escena 3D. Puede ser que quisiéramos aportar la visualización de texto para aplicaciones educativas,…, pero la solución más normal sería incorporarlo en la visualización pero en la parte 2D, no en la escena en tres dimensiones. De hecho, no existe ningún objeto o nodo "texto" definido en Shockwave 3D.

En relación a los textos, animaciones,…, en multimedia se suele manejar el concepto de "Billboard". Esta palabra la podemos traducir como "cartelera". Esto quiere decir que este modelo que si es habitual en escenas 3D, lo podemos semejar a un cartel o una pantalla que muestra un contenido, normalmente una imagen conteniendo texto o no. Los Billboard se construyen con planos, ya que lo vamos a texturizar y es un modelo más efectivo que una caja, ya que al texturizar este modelo tenemos que representar las imágenes en las seis caras que la componen, mientras que en un plano solo lo haríamos en dos.

Como comentario final a este tema podemos decir que aunque Director es un gran contenedor multimedia, en las escenas 3D no podemos utilizar tantos medios de forma nativa como quisiéramos. Tenemos que tener en cuenta que lo único que visualizamos son objetos 3D y por tanto, tienen geometría. Esto hace que medios típicamente planos como el texto, tengamos que adecuarlos a un entorno distinto.

Caso práctico 16.1: Medios Multimedia.

Objetivo: Crear una escena tridimensional dónde podamos interactuar con distintos medios multimedia que incluyen textos, imágenes, sonido, animaciones y videos.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Crear la escena 3D.
2. Interacción del caso práctico.
3. Texto.
4. Imagen.
5. Sonido interno.
6. Sonido en streaming.
7. Animaciones.
8. Videos.
9. Botón para quitar el sonido.

1. Crear la escena 3D.

En este caso práctico, hemos empleado la misma estrategia que en la mayoría de las prácticas para realizar la escena, o sea, utilizar 3D MAX para crear una serie de modelos 3D con movimiento con el fin de que sirvan de lienzo para texturizar sobre su superficie los elementos multimedia adecuados.

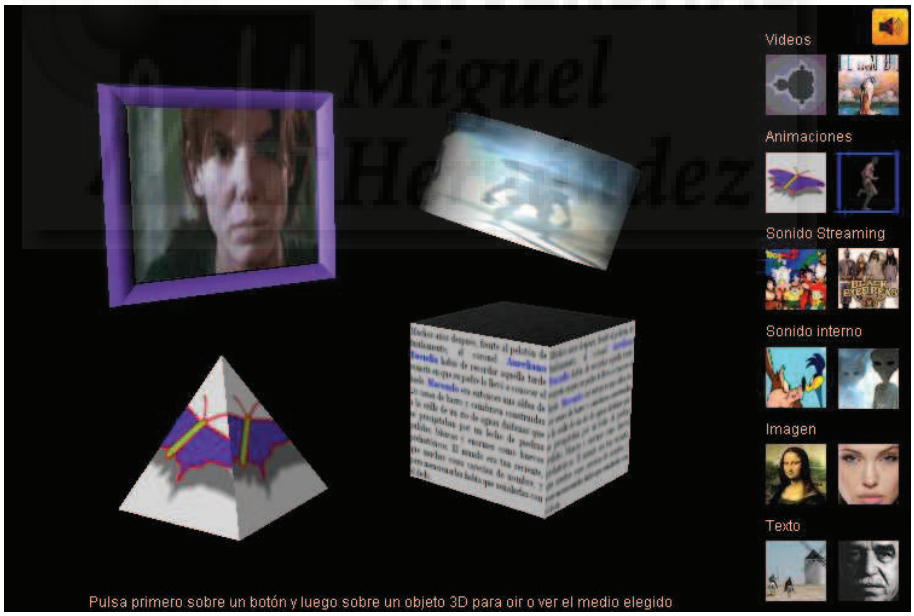


Imagen 5.16.cp.1.1: El caso práctico en ejecución

En este sentido, podemos destacar que hemos creado una caja con un fondo muy estrecho para que haga las veces de “pantalla”, y para potenciar este efecto, lo hemos rodeado de un marco. Los demás elementos son objetos primitivos que se utilizan para observar la deformación que sufre la textura al aplicarlas en las caras de distintos objetos 3D.

Los objetos cilindro, pirámide y cubo giran sobre su eje para observarlos desde distintos puntos de vista mientras que la pantalla permanece siempre en la misma posición.

La creación de los objetos y su animación no tienen complicación alguna. Si acaso, comentar que el marco de la pantalla se realizó con el modificador “Profile Bevel” de MAX.

En cuanto a la exportación de la escena a Director, no podemos escribir nada destacable ya que hemos seguido el método estándar utilizado en los casos anteriores.

2. Interacción del caso práctico.

La interacción ha sido diseñada de forma muy sencilla. Para ejecutar un medio multimedia, el usuario debe pulsar sobre un botón de los que se muestra en la parte derecha y seguidamente pulsar sobre un objeto 3D. De esta manera estamos "uniendo" ambos mundos, el 2D y el 3D, que puede ser muy útil a la hora de crear obras más complicadas.

Por ello, el stage, es decir, la pantalla de la aplicación, está dividida en dos zonas (aunque al tener ambas el fondo de color negro no lo aparenta). En la parte izquierda hemos posicionado la pantalla para Shockwave 3D y en la parte derecha tenemos la zona 2D para contener la botonera. También existe una pequeña franja en la parte inferior para informar al usuario de cómo debe de interactuar con la aplicación.

El mecanismo se efectúa con la siguiente secuencia:

2.1. Creamos un botón como una imagen utilizando, por ejemplo, PhotoShop.

2.2. Se importa la imagen como un miembro del cast internal.

2.3. Se arrastra el miembro que contiene la imagen del botón a la zona del Stage 2D. Tenemos que visualizar la ventana de Score para recolocar, si es necesario, la aparición del botón, de tal manera que se sitúe en un solo fotograma: el que hemos llamado "Principal".

2.4. Se crea un behavior y se arrastra sobre el sprite del botón creado, es decir, se arrastra hasta la posición que ocupa en la ventana Score. Casi todos behaviors creados con este fin tendrán el siguiente código:

```
global variable

on mouseDown
  variable="nuevo valor"  -- la variable indica al resto de behaviors que este botón se ha pulsado
  Actualizadatos()              -- para que Director consulte y actualice los parámetros activos
end
```

No todos los botones realizados se "comportan" de la misma manera, por eso vamos a ir estudiando caso por caso.

3. Texto.

Vamos a suponer que ya tenemos el botón correspondiente integrado en la botonera. Para introducir texto a partir de la interacción del usuario seguimos los siguientes pasos:

3.1. Crear y asignar un behavior donde cambiaremos el valor de la textura activa por un elemento imagen que visualizará un texto. El código quedará como sigue:

```
global textura

on mouseDown
  textura="texto_1"              -- en este ejemplo cambiamos la textura activa por esta
  Actualizardatos()              -- para que Director consulte y actualice los parámetros activos
end
```

3.2. En el behavior "Actualizardatos" que es de tipo general, tendremos entre otras líneas de código, una que asigna la textura activa (en este caso la que muestra el texto) al objeto 3D seleccionado, tal como muestra el siguiente código:

```
mundo3D.shader(shadertocado).texture=mundo3d.texture(textura)
```

Para que esta interacción funcione, es evidente, que utilizaremos una variable (que en este caso es global y se llama “shadertocado”) que contiene el identificador del sombreador del objeto pulsado por el usuario como receptor del texto a visualizar. Este proceso se realiza dentro de los behaviors “Programación” y “Queshader”. Como este mecanismo ya ha sido explicado en casos prácticos anteriores no lo detallamos aquí.

4. Imagen.

El medio “imagen” se realiza de la misma forma que el medio texto. Como hemos comentado anteriormente, el texto, no es más que una imagen. Por lo tanto, lo escrito para el punto anterior se puede aplicar directamente en este caso.

5. Sonido interno.

Estos archivos, normalmente están en formato .MP3 por la gran difusión y rendimiento que tienen en la actualidad. Estos sonidos que se guardan internamente suelen ser efectos sonoros cortos y que se repiten muchas veces, como pueden ser la respuesta a pulsaciones de botones, etc.

Para ejecutar un sonido guardado en la propia aplicación bajo demanda del usuario, realizaremos los siguientes pasos:

5.1. Importar un sonido como member. Hacemos File > Import y examinamos nuestro disco duro para referenciar el archivo a importar.

5.2. Crearemos un botón y le asignaremos un behavior con un código como este:

```
global botonpulsado

on mouseDown
  botonpulsado=3          -- suponiendo que 3 identifica al sonido que queremos oír.
end
```

5.3. En el script “Programación” que se ejecuta en cuanto el usuario interacciona con la escena 3D (por ejemplo al pulsar sobre un objeto 3D), introduciremos un código como este:

```
if (botonpulsado=3)    then
   sound(1).play(member("Sonido_1"))   -- ejecuta el sonido_1 por el canal 1 y lo oímos.
end if
```

En el caso práctico, además de la ejecución del sonido, se texturiza el objeto pulsado con una imagen y se llama al procedimiento “Actualizadatos” para recalcar la interacción.

6. Sonido en streaming.

Para hacer streaming de audio debemos proceder con los siguientes pasos:

6.1. Ejecutar en el menú Insert > Media element > Shockwave Audio. Esto hace que aparezca la caja de diálogo que se muestra en la imagen siguiente.

6.2. En la caja de diálogo “SWA Cast Member Properties” tenemos que seleccionar la dirección del fichero a ejecutar. Pulsar el botón “Browse” si es necesario para referenciarlo.

6.3. En esta misma caja de dialogo podemos controlar el volumen, el canal de sonido utilizado y la memoria para el flujo (stream buffer) que viene como “Preload Time”.

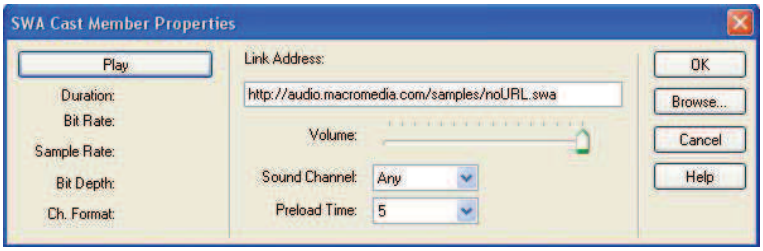


Imagen 5.16.cp.1.2: Propiedades de cast member tipo SWA

6.4. Pulsamos el botón "OK" y entrará a formar parte como una nueva referencia en el cast Internal, tal como se puede apreciar en la imagen inferior con el nombre "SBEP".

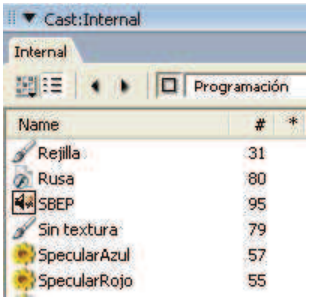


Imagen 5.16.cp.1.3: Cast Internal con un componente streaming

Para controlar este nuevo "member" no podemos utilizar los métodos empleados con un sonido "normal" ya que este no se encuentra directamente en nuestro archivo sino en un archivo externo. Por lo tanto, utilizaremos los métodos play(), pause() y stop() para medios externos como mostramos a continuación.

```
if (tocado.name= "Pyramid") then member("SBEP").play()
```

En este caso se ejecutará el sonido llamado "SBEP" por streaming cuando se produzca una condición, que en este caso es que se pulse sobre un determinado modelo 3D llamado "Pyramid". También podemos utilizar las siguientes sentencias.

```
member("SBEP").pause()          -- hace pausa en la ejecución del audio streaming

member("SBEP").stop()-- detiene la ejecución del audio streaming
```

7. Animaciones.

Para ejecutar una animación bajo demanda del usuario seguiremos la misma estrategia que con otros medios. Es decir, creamos un botón que el usuario pulsará para que se ejecute la visualización de la animación elegida en la superficie del último objeto 3D que habrá sido pulsado con anterioridad.

De todas formas, existen algunas diferencias con respecto a la integración de otros medios, de las cuales destacamos que la ejecución propiamente dicha del medio, no se realiza en el behavior "Programación" sino en el behavior "Bucle".

Tenemos que recordar que la animación será una película de formato .SWF. Las razones para ello están ampliamente explicadas en la unidad teórica, aunque podemos resumirlas en un hecho en concreto: este formato permite acceder a cada una de los fotogramas que componen la película de forma independiente y mediante código Lingo.

Los pasos que tenemos que seguir para incluir animaciones serán:

7.1. Suponiendo que tenemos ya importada la animación en un archivo swf, el código asignado al botón que ejecutará la animación será como el siguiente:

```
global animacion1

on mouseDown
  animacion1=1    -- 1 significa que el botón para la ejecución de la animación 1 ha sido pulsado
end
```

7.2. Como hemos escrito anteriormente, la ejecución de la animación se llevará a cabo en "Bucle", que es un behavior que se ejecuta 24 veces en un segundo, al mismo ritmo que la ejecución de la propia película. Por tanto, en este behavior tendremos un código como el que mostramos a continuación:

```
global animacion1
property contador1

if (animacion1 = 1) then
  if (contador1<10) then contador1 = contador1+1
  else contador4 =1
  member("Animacion_1").posterFrame = contador1
  mundo3d.texture("textura").image= member("Animacion_1").image.duplicate()
  go the frame
end if
```

La explicación exhaustiva del funcionamiento de este código se encuentra en la unidad teórica 16. De todas formas, recordaremos que ejecutamos un truco para ir extrayendo del fichero swf una imagen tras otra ayudándonos de la variable "contandor1" y todo ello, "supervisado" por las sentencias condicionales del principio, que controlan que no nos salgamos del rango de posibles valores del fotograma a mostrar.

7.3. Para terminar, apuntar que es necesario inicializar la variable "animacion1" al valor 0, al principio de la aplicación. Por tanto, tendremos el código que sigue en el behavior "Inicialización" asignado al primer fotograma de la aplicación:

```
global animacion1

animacion1= 0
```

8. Videos.

Para realizar el medio de video en Director de una forma sencilla, podemos ejecutar a la vez un sonido (que normalmente será ejecutado en forma de streaming) y una animación. Como estos dos medios ya han sido explicados con anterioridad, no debemos tener grandes problemas para llevar a cabo esta estrategia.

Donde sí podemos tener dificultades es en la creación de los archivos fuente: obtener el audio y el vídeo en archivos separados. Esto se debe a que normalmente, el video lo adquirimos en un formato como mpeg, avi,… donde los dos medios se ejecutan y almacenan de forma conjunta.

Este problema lo podemos resolver con Flash. Este software permite la importación de videos. Debemos de elegir la opción que permite importar un video y crear una línea de tiempo en formato Flash y además, separar el audio en otro elemento.

Una vez realizado esta separación, tendremos en la llamada "Biblioteca" de Flash, dos elementos: el audio y el video por separado. Con lo cual podemos copiar y pegar el audio a un archivo nuevo vacío y por tanto, separarlo del video.

Una vez realizado este paso, solo queda exportar cada uno de los dos archivos al formato que deseamos, esto es, el video a .swf y el audio a un archivo que referenciamos como un Shockwave audio para ejecutarlo en streaming.

8.1. En este caso, la ejecución del audio, la podemos realizar directamente en el behavior "Actualizardatos" que se ejecuta directamente desde el comportamiento que a su vez se ejecuta cuando el usuario hace click sobre el botón correspondiente. El código sería como el que sigue:

```
if (video1= 1)  then
  member("Video_1_sonido").play()
  member("Video_2_sonido").stop()
  member("Cancion_1").stop()
  member("Cancion_2").stop()
end if
```

Donde suponemos que "Video_1" es una variable que sirve para comunicar a los distintos behaviors que el botón para la ejecución del primer video, ha sido pulsado por el usuario. Además destacamos que cuando se pone en funcionamiento el sonido que corresponde, se silencian todos los posibles sonidos de tipo streaming. Esto se hace para "limpiar" de sonido la aplicación, ya que se pueden superponer los sonidos de larga duración y esto crearía una ejecución confusa.

9. Botón para quitar el sonido.

En este caso, el botón debe realizar lo que hemos visto anteriormente, que es parar la ejecución de los sonidos streaming, pero también de cualquier sonido interno.

9.1. Por tanto el código asociado al botón para silenciar todos los sonidos será como sigue:

```
member("video_1_sonido").stop()
member("video_2_sonido ").stop()
member("Cancion_1").stop()
member("Cancion_2").stop()
puppetsound(1,0)     -- suponemos que los sonidos internos siempre se ejecutan por el canal 1
```

En resumen, lo que hacemos es parar la ejecución de todos los sonidos de tipo streaming y cerramos el canal 1, que es el que utilizamos para los sonidos que se ejecutan internamente.

Conclusiones:

La realización de este caso práctico, pone en evidencia, que la adaptación a Shockwave 3D, supone realizar cierta "trampa", ya que existen medios como es el caso del texto, que no tienen correspondencia en 3D, y que tenemos que simular con una imagen. Esto no siempre es satisfactorio, ya que, en realidad no lo podemos tratar como texto, ni imprimir, ni hacer scroll, etc. Aunque sí que podemos afirmar, que con códigos no muy complejos, podemos integrar varios medios de forma efectiva y efectista, ya que de este modo podemos acumular varios refuerzos multimedia en la acción del usuario, y de esta manera, podemos enriquecer tanto la propia interacción como el contenido de la escena 3D.

Unidad 17: Publicación on/off line en Director.

Introducción teórica:

1. Publicación del trabajo realizado, cuestiones previas.
2. Diferencias y similitudes de la publicación en línea y fuera de línea.
3. Publicación del trabajo en un CD-ROM /DVD.
4. Publicación del trabajo en una web.
5. El plugin Shockwave 3D.

Casos prácticos:

17.1. Publicación on/off line del proyecto.

1. Publicación del trabajo realizado, cuestiones previas.

Una de las cuestiones que hay que tener en cuenta antes de empezar a pensar en la publicación del trabajo es la prueba. En ingeniería del software se le da una importancia capital a la prueba del software antes de ponerlo a disposición de los clientes, en nuestro caso, del público espectador de nuestras obras.

Normalmente, un trabajo serio y consistente es largo, a veces lleva meses. Un proyecto donde tenemos modelos tridimensionales e interactivos, supone un trabajo de diseño pero también casi inevitablemente de programación. Se dice que cuando un individuo inicia la programación "se sabe cuando empieza pero no cuando acaba". Esto es más cierto que nunca en el tipo de programación y modelado que estamos llevando a cabo, dado que la inspiración tanto artística como al programar puede venir según se va creando la obra.

Es importante que mientras se va creando, se vaya probando. Hay que realizar pruebas siempre que se pueda e incluso administrar nuestro tiempo para su realización sistemática y profundamente a menudo. La diferencia entre un trabajo serio y "jugar" es la prueba y la consiguiente solidez del trabajo terminado.

Cuando trabajamos en proyectos multimedia utilizamos las posibilidades del ordenador al máximo. Los juegos y presentaciones multimedia son los que utilizan las últimas tecnologías para conseguir trabajos espectaculares y por tanto, las pruebas se tienen que diseñar y realizar muchas veces, en situaciones extremas de cálculo computacional.

2. Diferencias y similitudes de la publicación en línea y fuera de línea.

La publicación del trabajo sea cual sea la plataforma de visualización supone que el archivo esté totalmente probado. Una vez superado este autoexamen no tendríamos que tocar el código Lingo y nos podríamos dedicar a minimizar el tamaño del archivo. Esto es relativamente importante para su publicación en CD o DVD pero es crítico e irrenunciable para su exhibición en Internet.

Para intentar bajar un poco el tamaño del archivo hay varias opciones. Una posibilidad es bajar el tamaño de todos los gráficos que hayamos utilizado. A esta técnica se la conoce como "dithering gráfico". La mayoría de estos gráficos los habremos realizado con 24 o 32 bits de profundidad con la mejor intención pensando en garantizar una calidad buena, pero normalmente es un desperdicio de memoria, ya que en muchos casos una bajada en la profundidad de bits, es decir en la calidad, no tiene consecuencias "visibles" y sin embargo supone un ahorro grande de memoria.

En Director esto se puede realizar seleccionando el miembro gráfico y ejecutando el comando Modify > Transfor Bitmap, apareciendo la caja de diálogo de la imagen que sigue.

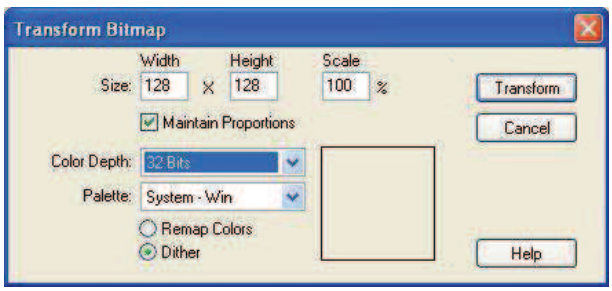


Imagen 5.17.1: Caja de diálogo para modificar los parámetros de una imagen

Modificando el parámetro "Color Depth" a 16 bits de profundidad, en casi todos los casos tendremos una calidad muy buena y nos ahorramos la mitad de memoria.

Si este trabajo ya está hecho, debemos guardar el archivo pero con la opción “Save and Compact”, ya que este comando va a comprimir los datos si es posible y además, si un elemento es redundante o no se emplea en la película no lo guardará, esto es más necesario cuanto más grande es el proyecto.

3. Publicación del trabajo en un CD-ROM /DVD.

Para publicar el trabajo en un CD-ROM o DVD, tenemos que crear lo que Director llama un Proyector. Un proyector es un fichero .exe. Este tipo de ficheros son autoejecutables en todos los sistemas Windows y esto quiere decir que no hace falta instalarlos ni configurarlos.

Las opciones para la creación de este proyector y de todos los demás tipos de publicaciones se realiza ejecutando el comando File > Publish Settings, entonces aparecerá una caja de diálogo como la que se muestra en la imagen siguiente.

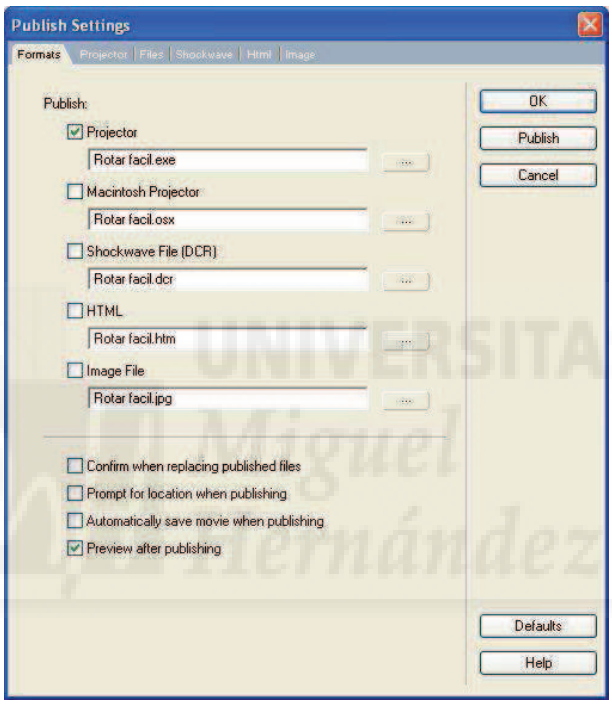


Imagen 5.17.2: Configuración de las opciones de publicación

En esta Imagen se puede apreciar cómo se permite publicar el trabajo en varios formatos. Debemos publicar en dos formatos: proyector y shockwave file. Este último lo ensamblaremos con una página web usando otra aplicación como por ejemplo, Dreamweaver también de Adobe, ya que la opción HTML que permite Director es muy rudimentaria.

Una vez que hemos elegido la primera y la tercera opción, pasaremos a configurar el proyector. Al elegir esta opción en las pestañas que tenemos arriba, aparece la imagen 5.17.3.

Las opciones que aparecen son muchas y no vamos a detallarlas todas. En realidad, las vamos a dar todas por buenas. Pulsaremos al botón “Publish” para que se realice la operación de crear el fichero .exe. Este fichero se llamará como el archivo de trabajo de Director, pero en vez de tener la extensión .dir tendrá la extensión .exe y se creará, por defecto, en el mismo directorio. Si accedemos a dicho directorio veremos dos tipos de iconos, el naranja es el .dir y el blanco el .exe.

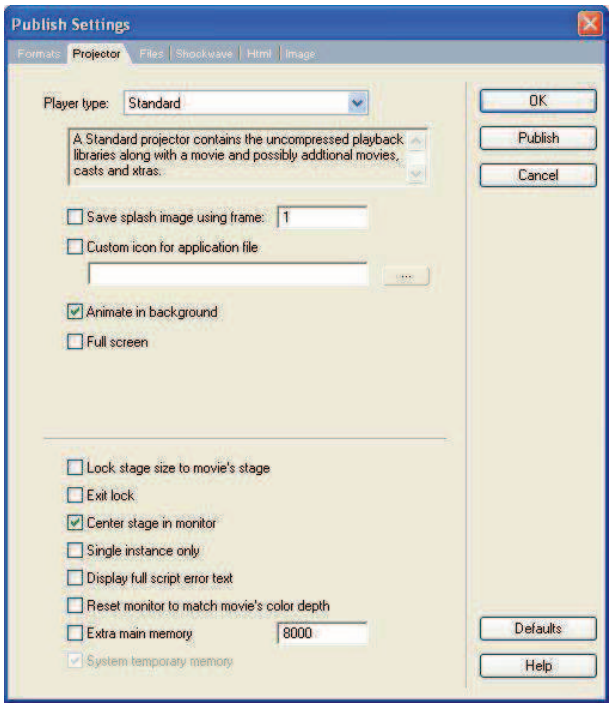


Imagen 5.17.3: Configuración de las opciones de publicación: Proyector

Probaremos el ejecutable para comprobar que todo funciona correctamente. Para ello, basta con acceder hasta el fichero y hacer doble clic con el botón izquierdo del ratón.

Este fichero lo copiaremos en un CD-ROM o DVD. Por otra parte, es conveniente crear un fichero de autoarranque para que nuestro programa se ejecute nada más introducirlo en un lector compatible, ya que es lo que esperan los usuarios que suceda. La mayoría de las personas, cuando introducen un CD-ROM o DVD esperan que “empiece” solo. Para ello basta con crear un fichero de texto sin formato y llamarlo “autorun.ini” y en su interior escribir:

```
[autorun]
open=nombre_del_fichero.exe
```

Esta tarea la podemos hacer directamente con la aplicación “Bloc de notas” de Windows, ya que este procesador de texto no permite el formato del texto y este fichero tampoco debe tener su contenido con ningún formato.

4. Publicación del trabajo en una web.

La publicación para la web es la más delicada, ya que nuestro proyecto tiene ante sí dos escollos que salvar.

Primero el ordenador del usuario. No sabemos en qué condiciones se encuentra, la memoria RAM dispone, ni el procesador, ni la versión de sistema operativo que utiliza.

Segundo la transmisión por Internet. Esta depende fundamentalmente de la calidad de la conexión del usuario y además necesita tener instalado un plug-in: Shockwave.

Nosotros tenemos que ejecutar la publicación del .DCR, luego crear la página web, es decir, el documento HTML y luego probarlo una vez publicado en un servidor web. Todo este flujo de trabajo hace que las pruebas sean muy costosas en tiempo y recursos.

Para crear el .DCR ejecutaremos el comando File > Publish Settings y elegimos la opción Shockwave. La imagen muestra las opciones de esta elección:

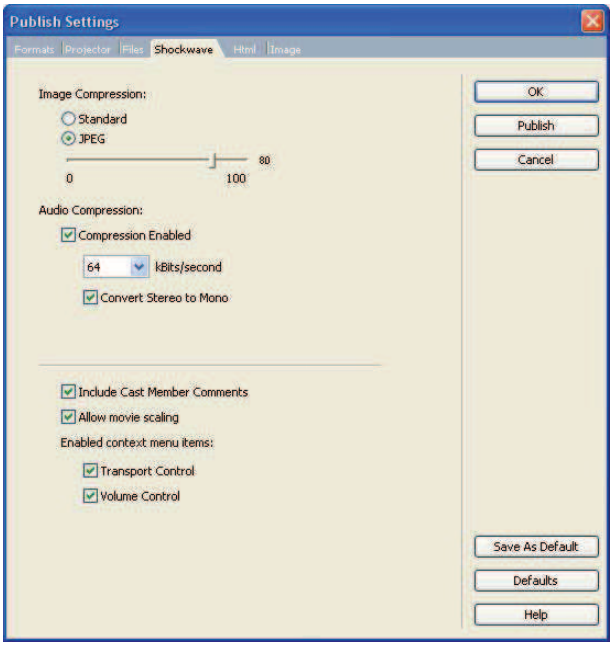


Imagen 5.17.4: Configuración de las opciones de publicación: Shockwave

En anteriores versiones se podía elegir entre versiones de Shockwave Player como la 10 o 8.5, pero esto conlleva complicaciones de compatibilidad. Ahora no se ofrece esta elección y las demás opciones se suelen poner en los valores que muestra la imagen, que son así por defecto. Una vez que pulsemos "Publish" obtendremos el fichero con extensión DCR.

5. El plugin Shockwave 3D.

Por último, resaltar que tenemos dos cuestiones por resolver. La obra necesita de Shockwave para visualizarse. Según cómo lo hayamos publicado tendremos una serie de opciones.

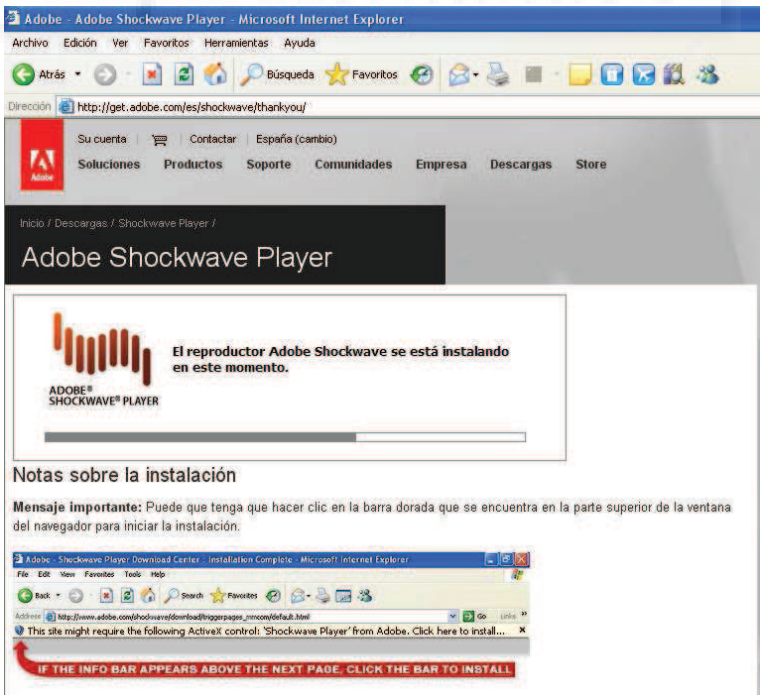


Imagen 5.17.5: Página de instalación del plug-in Shockwave

Para poder probar el .DCR en nuestro ordenador tenemos que tener, como todos los usuarios que lo vayan a visualizar, el plugin para navegadores de Shockwave instalado. Para conseguirlo hay que bajarlo de la dirección http://get.adobe.com/es/shockwave/

En la imagen anterior se observa la página de descarga del plug-in en el momento de la instalación en el ordenador del usuario.

Por último, hay que añadir que Shockwave 3D es un Xtra de Director. Y este Xtra es necesario para la ejecución de la obra en formato proyector. Tenemos dos alternativas, podemos embeberla dentro del archivo o pedir que se transmita en el momento en que se necesite. Nosotros recomendamos embeberla, ya que así hacemos un fichero totalmente autónomo, aunque el archivo final se hace más grande. Esto no es muy importante cuando se va a publicar en CD o DVD.

Para hacer esto tenemos que ejecutar Modify > Movie > Xtras > Add > y elegir “Shockwave 3D Asset .X32”.

Si no la embebemos podemos usar la opción “Download if hended” que está en la ventana Xtra, lo que obliga a bajarla de Internet en el mismo momento para seguir con la ejecución, cosa que puede irritar mucho al usuario y por supuesto, exige que tenga conexión a Internet.

En la siguiente imagen se muestra la caja de diálogo que muestra cómo añadir la Xtra de 3D embebida a nuestro archivo.

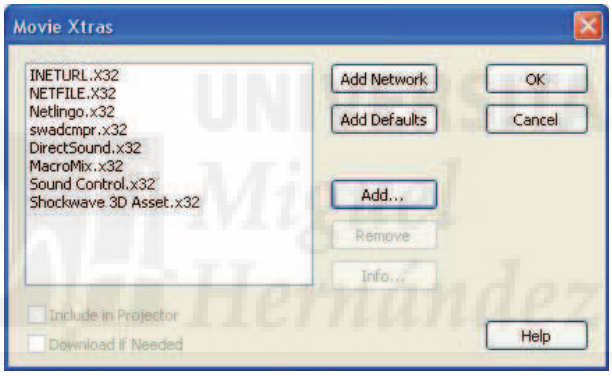


Imagen 5.17.6: Añadir la Xtra 3D embebida

Como comentario final a esta unidad, escribir que si se siguen los pasos estándar para la publicación de nuestros trabajos, no encontraremos grandes dificultades para cumplir el objetivo, y este no es otro, que el espectador de nuestra obra la visualice como nosotros hemos decidido que lo haga.

En este tema, buscar atajos para una exhibición más rápida o productiva, normalmente es sinónimo de fracaso. Si tenemos la posibilidad, deberíamos ofrecer la alternativa al usuario: o la universalidad de Internet o la productividad del CD o DVD.

Caso práctico 17.1: Publicación on/off line del proyecto

Objetivo: Publicar en CD / DVD e Internet nuestro trabajo en Director.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Publicación del trabajo en un CD-ROM /DVD.
2. Publicación del trabajo en una web.

1. Publicación del trabajo en un CD-ROM /DVD.

Cuando después de realizar un trabajo deseamos exponerlo, una vez realizadas las pertinentes pruebas, pasaremos a publicar el resultado en dos medios distintos. El primero es en CD-ROM/DVD donde disponemos de una gran cantidad de memoria, además es barato y prácticamente universal. El segundo es Internet, en la web tenemos una oportunidad muy grande de llegar al máximo número de espectadores de nuestra obra, pero debemos de publicar trabajos no muy grandes para agilizar su transmisión. En este caso práctico publicaremos una sala de exposiciones como ejemplo de trabajo que ocupa bastante memoria.

Para publicar el trabajo en un CD-ROM o DVD, vamos a crear un Proyector. Un proyector es un fichero .exe que se puede ejecutar en la mayoría de los ordenadores personales y es autocontenido y por lo tanto autónomo.

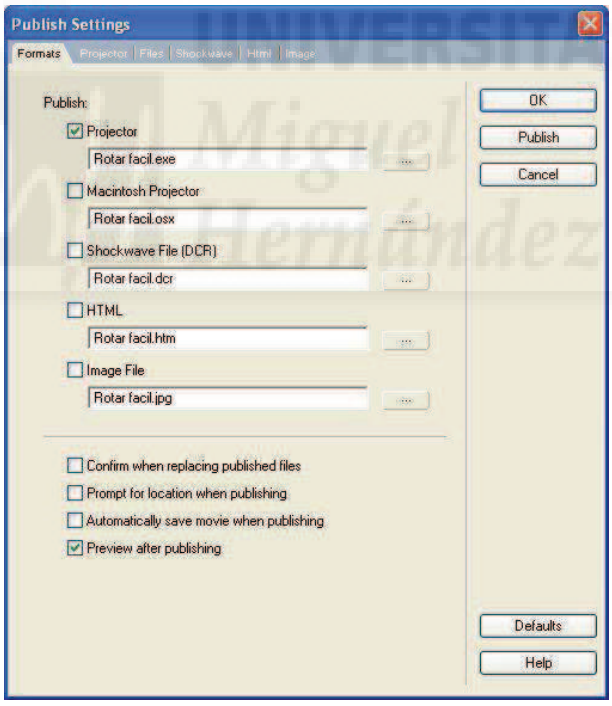


Imagen 5.17.cp.1.1: Caja de diálogo para configurar la publicación de la obra

1.1. Ejecutar el comando File > Publish Settings. Ante la caja de diálogo como la que se muestra en la imagen anterior, hay que elegir la opción Projector.

1.2. Pulsar sobre la ficha “Projector”. Entonces aparecerá la caja de diálogo siguiente.

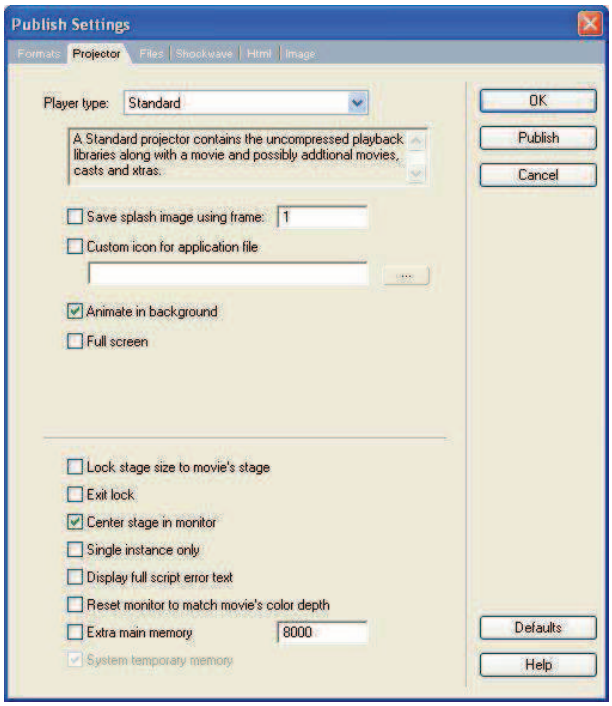


Imagen 5.17.cp.1.2: Caja de diálogo para configurar la publicación del Proyector

1.3. Nos fijaremos especialmente en la opción “Player Type” y comprobaremos que está en el valor Standard. Pulsar el botón “Publish”.

1.4. Nos pedirá el directorio donde se creará el fichero deseado. Conviene crear un directorio nuevo para este trabajo.

1.5. Ahora podemos salir de Director, ir al directorio elegido y hacer doble clic sobre el fichero para que se ejecute y comprobemos que todo funciona correctamente.

1.6. Vamos a crear un fichero de autoarranque para la publicación. Ejecutar cualquier procesador de textos que permita guardar en formato solo texto como por ejemplo el bloc de notas de Windows.

1.7. Escribir el texto:

```
[autorun]
open=nombre_del_fichero.exe
```

Por ejemplo, suponiendo que el fichero se llama “sala_de_exposiciones”, escribimos:

```
[autorun]
open=sala_de_exposiciones.exe
```

1.8. Guardar el fichero recién creado con el nombre de “autorun.ini” en el directorio junto con el programa, de forma que cuando lo grabemos solo tendremos dos ficheros para grabar y esto hará que tengamos menos posibilidades de equivocarnos.

Por último, señalar que conviene probar el CD-ROM/DVD en otros ordenadores más lentos y donde no tengan instalado Director para comprobar que se puede empezar a distribuir.

2. Publicación del trabajo en una web.

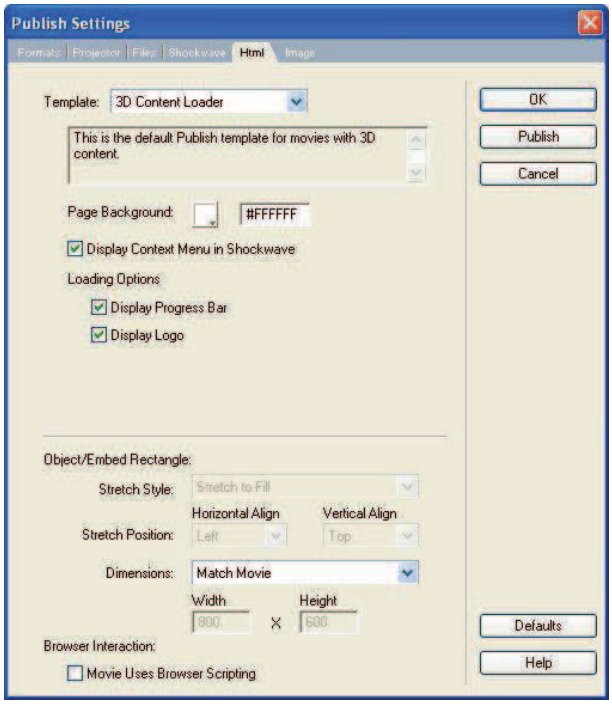


Imagen 5.17.cp.1.3: Caja de diálogo para configurar la publicación. Ficha Html

La publicación para la web es más crítica con el tamaño de los ficheros. No es recomendable publicar ficheros mayores de 1Mb.

2.1. Ejecutar el comando File > Publish Settings. Elegir la ficha “HTML”.

2.2. Comprobar que tenemos la opción “Display Progress Bar” elegida. Esta opción creará un fichero más pero ejecutará una barra de progreso de la carga del trabajo que será muy útil y más aún para trabajos tan grandes como este.

2.3. Hacer clic sobre el botón “Publish”.

2.4. Nos pedirá el directorio donde se creará el fichero deseado. Conviene crear un directorio nuevo para este trabajo.

2.5. Ahora podemos salir de Director, ir al directorio elegido y hacer doble clic sobre la página web recién creada.

Podremos observar que en el directorio tenemos 3 ficheros en total que se llamarán:

Nombre_trabajo.html: documento que contiene la página web.

Nombre_trabajo.dcr: nombre del fichero que se ejecuta dentro de la página web.

3D_progbar.dcr: fichero para la barra de progresión de la transmisión del trabajo.

Si el usuario no tiene instalado el plug-in para que su navegador visualice contenidos Shockwave 3D, la página web lo avisará y nos pedirá permiso para instalarlo en nuestro ordenador con lo que la mayoría de problemas se resuelven automáticamente.

2.6. Una vez comprobado que podemos ejecutarlo en modo local lo publicaremos en Internet mediante cualquier software FTP como cuteftp, etc. y en un servidor que no necesite ninguna característica especial. La ejecución desde este medio será la verdadera prueba de que hemos realizado el trabajo correctamente.

Conclusiones:

La publicación de nuestro trabajo es un tema delicado. Director está pensado para publicación en sistemas locales: cd´s o dvd´s. Las últimas versiones permiten publicar para Internet, teniendo en cuenta las necesidades de hoy en día. Shockwave está instalado en la actualidad en cerca del 50% de ordenadores de todo el mundo, ya que es un plugin muy utilizado para juegos on-line por lo que muchas veces podremos ejecutarlo sin tener que cargarlo en ese momento y solo tendremos que preocuparnos del tamaño final de nuestra obra.

CAPÍTULO 6:

CURSO: IMPLEMENTACIÓN CON EL METAVERSO SECOND LIFE

Unidad 1: Introducción a Second Life.

1. Introducción a los metaversos y origen (historia) de Second Life.
2. Definición de Second Life.
3. Respuesta de Second Life a los requisitos impuestos por la tesis.
4. Comparación con la alternativa Max-Director. Ventajas e inconvenientes.
5. Comparación con otros metaversos.
6. Arte en Second Life
7. Qué experimentar en Second Life.

1. Introducción a los metaversos y origen (historia) de Second Life.

Second Life es un tipo de software que podemos calificar de metaverso. El metaverso es una entidad que surge por primera vez dentro de una novela de 1993 llamada “Snow Crash” de Neal Stephenson.

En esta novela, Stephenson propone un futuro de los Estados Unidos donde las diferencias entre clases son abismales y la mayoría de la población, para escapar de sus miserables vidas acceden al “metaverso”. Este es una realidad virtual en 3 dimensiones de acceso público donde las personas aparecen como avatares. No es por lo tanto un juego, sino una alternativa a su vida real, un sitio donde vivir mejor.

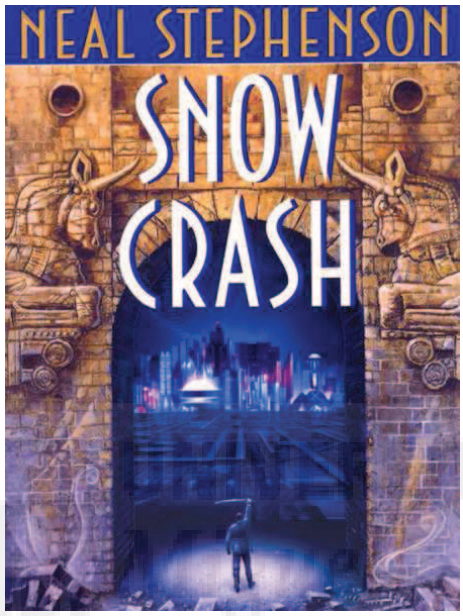


Imagen 6.1.1: Portada de la novela que inspiró la creación de Second Life

Este concepto pronto inspiró a desarrolladores de los años 90 sobre todo en EEUU y empezaron a aparecer metaversos que en su implementación debemos considerar como una combinación de los juegos 3D y software para la interacción social como chats.

Los expertos consideran a Meridian 59 como el primer metaverso. Es publicado por la compañía Interactive Archetype el 15 de diciembre de 1995 y pasó a ser comercial en septiembre de 1996. Sus creadores son Andrew y Chris Kirmse. Le siguió Ultime Online (UO) de Electronics Art que aparece el 25 de septiembre de 1997 y es considerado el primer MMORPG (Massively multiplayer online role-playing game), es decir, juego de rol para Internet multijugador en masa. El primer metaverso de éxito social se puede considerar a Norrath.

En diciembre de 2001 Edward Castronova, profesor de Economía y Telecomunicadiones en la Universidad de Indiana publicó “Virtual Worlds: A First-Hand Account of Market and Society on the Cyberian Frontier”. Este trabajó fue muy polémico y su autor tachado de loco. Hoy en día es una referencia obligada para conocer la evolución de los metaversos. En esta publicación, Castronova enumera las características de los metaversos. Que son los siguientes:

Interactividad: los usuarios pueden acceder de manera remota y simultánea a las máquinas que contienen el mundo virtual y sus acciones son percibidas e influyen en las acciones del resto de usuarios.

Corporeidad: los usuarios acceden al programa mediante un interfaz que simula un entorno en primera persona sometido a las leyes de la física.

Persistencia: el sistema existe independientemente de los usuarios y aunque estos no estén conectados mantiene la localización de objetos. Cuando el usuario vuelve a conectarse lo hace

en las posiciones y con las propiedades con las que se desconectó. La responsabilidad del mantenimiento del sistema es del publicador del metaverso.

Se ha dicho que los metaversos están aquí para quedarse, bien sea como complementos o evolución de la web. Según algunos, aunque hoy solo sea una nueva forma de utilizar Internet, en el futuro, toda la Red será en diferentes mundos virtuales en 3 dimensiones, sobre todo, porque representa el mundo real de forma inmejorable.

2. Definición de Second Life.

Philip Rosedale, lanzó la versión beta de Second Life en 2002. Es, hasta la fecha, el metaverso más influyente. Philip Rosedale (Philip Linden dentro de Second Life) es licenciado en física y aplicó aspectos de la teoría del caos al crear este mundo virtual. Según su creador “no es un juego sino una nueva tierra”.

Philip Rosedale, aspira a que su empresa Linden Labs, lidere con sus protocolos cliente – servidor la web 3.0.

Second Life está creado por los propios usuarios y es esta una característica que lo distingue de los demás metaversos, ya que no tiene un objetivo propuesto como el jugar, el intercambiar información, o establecer redes sociales, sino que cada usuario lo utiliza para lo que desea. Se puede decir que en cierta medida es anárquico, ya que no existe una jerarquía establecida entre los usuarios. Además cuenta con muchos usuarios y una repercusión en la prensa bastante notable. Tiene mecanismos que han producido una emergente economía propia y además aumenta su resolución y funcionalidad siguiendo la Ley de Moore (ley empírica que formula que cada 18 meses se duplica el número de transistores que se colocan en un circuito integrado pero que actualmente se puede extender a otros aspectos de la industria informática), es decir, que aumenta de forma exponencial.

Su principal característica es la libertad que se otorga a los usuarios y de las posibilidades que esto genera y que se ven apoyadas por las herramientas de creación 3D, de integración multimedia y de personalización de avatares y objetos que aporta el propio sistema.

Permite a cualquier participante la creación de cualquier tipo de contenido digital como edificios, ropa, accesorios, vehículos, arte, etc., garantizando los derechos de propiedad intelectual, y es esto lo que permite la venta de estos objetos y por tanto la creación de negocio. Por tanto, casi siempre es necesario tener algo de dinero virtual en la moneda propia de Second life, los Linden Dollars, que nos costará algo de dinero real (según mercado).

Second Life está sustentado por 8.000 servidores que funcionan con Linux, donde se mantiene y ejecutan en tiempo real todos los objetos, texturas, sonidos, imágenes y avatares de todos los usuarios del mundo de Second Life.

El negocio de Second Life está en la venta de terreno y la compañía Linden Labs (San Francisco, EEUU), también gana dinero como soporte físico de estos objetos: la creación de espacio en sus discos duros y su puesta en Internet. Los usuarios pueden ser gratuitos o de pago, la diferencia es que los de pago pueden comprar terrenos donde construir sus propios edificios y terrenos. Esto genera que Second Life vaya creciendo y se creen nuevas islas, regiones y paisajes. En realidad es como en la web común, pero en vez de cobrar por mantener información de páginas 2D, cobran por mantener información de objetos en 3D.

Los usuarios de pago pueden comprar con 7,5 euros una parcela de 15.000 metros cuadrados (Julio 2009) y construir sobre ella hasta 900 objetos para lo que quieran. La extensión actual de SL es de 579,42 kilómetros cuadrados. En esta nueva tierra se movieron en abril algo menos de seis millones de euros. Siendo su PIB de unos 700 millones y es jugado unos 25 millones de horas al mes en todo el mundo.

Se supone que llegan a jugar hasta 60.000 personas a la vez de pico y la media ronda los 40.000, esto, de todas formas, no quiere decir nada, ya que la enorme extensión de Second

Life puede hacer que estemos viajando minutos sin ver a nadie. Están dados de alta 6 millones de personas de las cuales 180.000 son españoles, aunque se contabilizan todos, incluso los que hace años que no entran en el metaverso. Un dato importante es que el perfil de los usuarios suele tener un alto grado de creatividad y valía profesional.

El perfil del español usuario de Second Life es un varón (65%) de mediana edad (33 años), con estudios universitarios (54%) y con empleo estable (70%) y este perfil si que diferencia Second Life de otros MMORPG.

Fuente: http://www.baquia.com/noticias.php?id=12373 (17/05/2007)

Se ha liberado el código de Second Life, por lo que la creación de programas complemento y la expansión hacia nuevas entidades del metaverso se hace inminente. Un ejemplo es OpenSim que comentaremos más adelante.

3. Cómo Second Life responde a los requisitos impuestos por la tesis.

Establecimos en la presente tesis que implementaríamos un curso como muestra de las herramientas que se pueden emplear para responder a ciertas demandas educativas.

Las herramientas que debíamos proponer y estudiar habían de cumplir una serie de requisitos que se pueden enumerar como sigue:

1. Los trabajos serán bi o tridimensionales, es decir, pueden ocupar un volumen y por tanto debe de mostrar un espacio en tres dimensiones por el que los autores y los espectadores puedan trasladarse.

2. El escenario y las obras que lo incluyen y sus propiedades podrán ser accesibles por la red de Internet.

3. Debe permitir que el usuario – artista pueda crear sus obras en el software que desee y por tanto debe ofrecer herramientas para importar obras de otras aplicaciones.

4. La herramienta debe permitir la interactividad. Esto quiere decir que el artista puede crear obras que respondan al espectador, es decir, interactivas.

5. Las obras podrán ser de tipo multimedia, es decir, estar compuestas por imágenes, vídeos, sonidos, animaciones,…

6. El sistema debe poder ser programable para que el artista tenga la libertad de controlar todo el entorno y la forma en que la obra se exhibe y se percibe.

Estas 6 condiciones que debe cumplir Second Life, las lleva a cabo de manera absoluta y eficiente. Cumple punto por punto todas las expectativas y además al ser un sistema alimentado por los propios usuarios y de código abierto, es de suponer que evolucione constantemente y podamos adaptarnos cada vez mejor a los condicionantes.

Vamos a estudiar cómo Second Life cumple uno por uno todos los requerimientos:

1. Espacio y obras tridimensionales.

En la imagen que sigue se puede observar un elemento en tres dimensiones que ocupa un volumen dentro del espacio de Second Life. Cómo ya hemos escrito anteriormente, Second Life es un metaverso, y en su propia definición viene establecido que es un espacio en tres dimensiones.

Un artista puede crear objetos en tres dimensiones directamente en Second Life. Bien es verdad que no puede hacerlo en cualquier sitio (ya que la mayoría de los terrenos son de un

determinado propietario) pero sí en lugares llamados areneros (sandbox) que propietarios de terrenos dedican a este fin y por supuesto, en los terrenos propios.

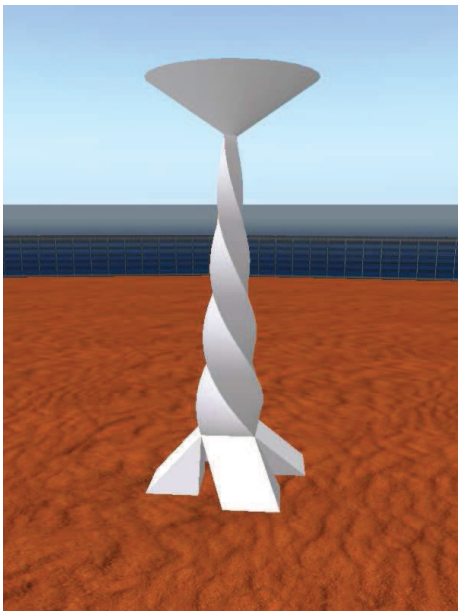


Imagen 6.1.2: Obra arquitectónica en 3D

2. Ejecutable en Internet.

Los metaversos son, por definición, aplicaciones que funcionan en la red y sus características están diseñadas para ser manejadas on-line. Esto no quiere decir que todo el sistema resida en la web. De hecho, Second Life, se ejecuta en parte desde el ordenador personal de cada usuario, pero siempre tenemos que tener acceso a Internet, estar “on-line”. Esta característica, en principio, puede ser una ventaja, pero también debemos tener en cuenta que no podemos mostrar nuestros trabajos si no estamos conectados. Por ejemplo, no podemos hacer un CD o DVD y ponerlo a disposición de los espectadores para que los visualicen de forma autónoma. Podemos hacer capturas de pantalla y vídeos, pero evidentemente perderemos toda interactividad.

Imagen 6.1.3: Pantalla de inicio de una sesión de Second Life

3. Importar obras desde otras aplicaciones.

Este no es uno de los puntos fuertes de Second Life, pero las causas son muy razonables. Lo primero es aclarar que si se pueden utilizar elementos editados con otros programas pero de momento, debemos de realizar esta tarea importando por separado la geometría y por otro las texturas.

Con respecto a las texturas, no hay problemas, ya que Second Life las trata muy eficientemente. En lo que atañe a la geometría, debemos importarla en forma de imagen que modifica geométricamente un objeto nativo de Second Life llamado Sculp. Esto quiere decir, que la forma de hacerlo es eficiente pero inusual.

Entre las razones para que esta característica sea no muy utilizada son varias:

1. Second Life incorpora herramientas para construir cualquier objeto que se nos ocurra. De hecho, casi todo lo que contiene Second Life está realizado con herramientas propias y es evidente que con muy buenos resultados.

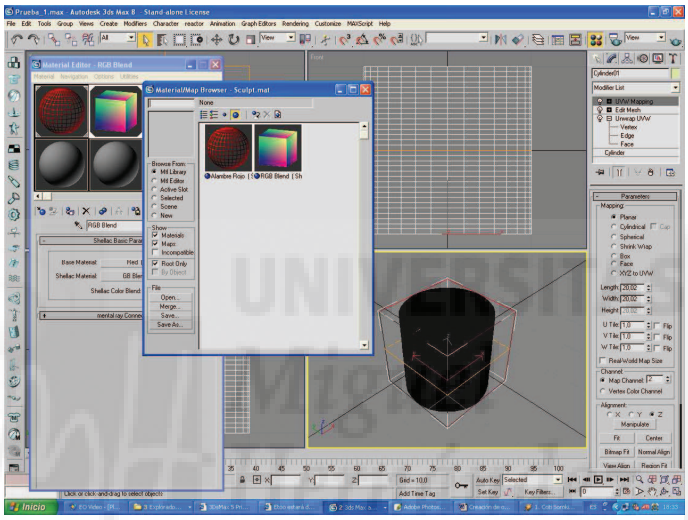


Imagen 6.1.4: Creación de geometría en 3D Studio MAX

Imagen 6.1.5: Geometría importada a Second Life

2. Estas herramientas son muy potentes y además muy fáciles de aprender y utilizar. La curva de aprendizaje es tan suave que usuarios no modeladores y texturizadores se atreven a utilizarla, obteniendo en muchos casos resultados sorprendentemente realistas.

3. La manera en que Second Life guarda la geometría y las texturas de los objetos es muy eficiente a efectos de explotación de los recursos del ordenador y la red.

De todas formas, arriba se pueden apreciar dos imágenes y en la primera de ellas se puede ver que estamos usando el programa más conocido para creación 3D como es 3D Studio MAX, y en la segunda imagen se puede ver el resultado ya incorporado a Second Life.

4. Debe ser interactivo.

Second Life es totalmente interactivo, tanto como un juego cualquiera. Aunque no se puede considerar un juego, podemos decir, que utiliza todos los recursos de la tecnología de los videojuegos para llevar a cabo una vida paralela, es decir, la interactividad es algo intrínseco en una realidad virtual.

Podemos caminar, volar, dar patadas, montar en un globo, Si nos cae un objeto físico nos afectará a nuestro equilibrio,… La diferencia es que en un juego todo está enfocado a recibir estímulos con el fin de seguir jugando y con fines lúdicos, sin embargo en Second Life esta tecnología está ahí, sin ningún fin específico. Podemos charlar o no, podemos caminar u observar sin movernos lo que pasa a nuestro alrededor.

Por ejemplo, en la imagen de abajo se puede observar como un objeto es rojo y cuando un avatar lo toca (o se lo pide por el chat, o…) cambia para mostrar una textura y por tanto una apariencia distinta.

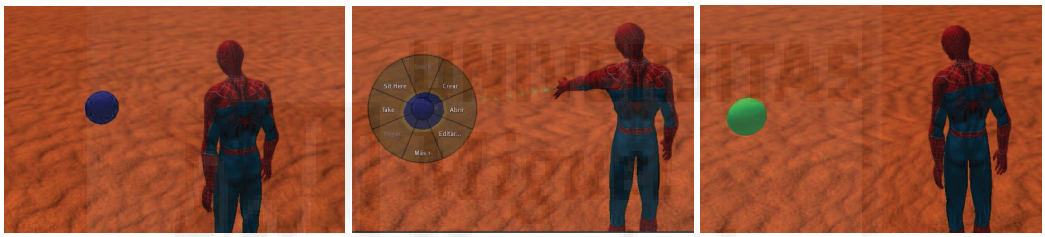


Imagen 6.1.6: El objeto cambia de apariencia al ser tocado

5. Integración de medios multimedia.

En este metaverso podemos utilizar distintos formato digitales multimedia e integrarlos juntos si así lo decidimos.

El sonido está presente desde el mismo momento en que entramos. Tenemos sonidos ambientales, además de sonidos emitidos por aparatos, etc. Por lo tanto, nosotros podemos incorporar sonidos a nuestras obras y que se ejecuten a nuestra voluntad. Por ejemplo, en la imagen de abajo se puede ver una imagen de un concierto realizado en este entorno.

Imagen 6.1.7: Sonido en Second Life

El vídeo también es un medio que podemos integrar en nuestros trabajos como se puede ver en la imagen de abajo. Podemos usar vídeos que se difunde en modo streaming, y esto significa que la ejecución se produce de forma gradual, se va reproduciendo mientras el vídeo o el sonido va llegando al ordenador local. Además se difunde a muchos usuarios de forma simultánea y muy efectiva, no recargando el trabajo de nuestro ordenador.

Imagen 6.1.8: Vídeo en Second Life

Las animaciones y el movimiento también lo podemos ejecutar en Second Life. Podemos distinguir dos clases de movimiento, lo que tienen los avatares y los que tienen el resto de objetos que componen Second Life.

El movimiento de los avatares es muy importante para los usuarios, ya que proporciona capacidades como saltar, bailar, hacer deporte, etc., pero sin embargo, para nosotros carece de interés, porque lo que pretendemos es aportar movimiento a nuestras obras. En este sentido, Second Life, deja toda la funcionalidad en manos del programador, ya que solo permite definir el movimiento autónomo de los objetos mediante programación.

De todas formas, el ver vehículos, autómatas, etc., se puede apreciar hasta dónde podemos llegar a la hora de aportar animación a nuestras creaciones.

Imagen 6.1.9: El mundo animado de Second Life

6. Debe ser programable.

Es una característica muy importante en una plataforma donde buscamos la mayor libertad posible para representar creaciones artísticas. Y la programación permite un control casi total de lo que queremos que vea el espectador.

Puede ser que en principio pensemos que este no es una herramienta propia de un artista. Tema por otro lado, ya discutido en el planteamiento de esta tesis. Pero ya concluimos que es mejor que el sistema tenga la posibilidad de ser manipulado mediante programación por muchas razones, aunque la mejor de ellas es que el software queda en manos del autor.

Si por cualquier razón, el artista no quiere programar, puede colaborar con otros autores aprovechando las utilidades de comunidades de trabajo en grupo. Además Second Life dispone de herramientas para programar las obras y el entorno de estas. Por tanto, la programación se hace fundamental para realizar obras originales y personales, ya que es lo que le proporciona cierta individualidad.

Por otra parte, Second Life, proporciona un compilador (traductor y ejecutador de código) siempre disponible, siempre "a mano". Es un sistema muy sencillo y se puede acceder directamente a través de los objetos que queremos programar. El lenguaje se llama Linden Scripting Language y es sencillo y muy parecido a otros lenguajes modernos como JavaScript y ActionScript pero enfocado a la realidad virtual que suponen los metaversos. Por eso las funciones que nos encontramos son para mover objetos, para cambiarles el color, para colocarlos en un determinado sitio, etc.

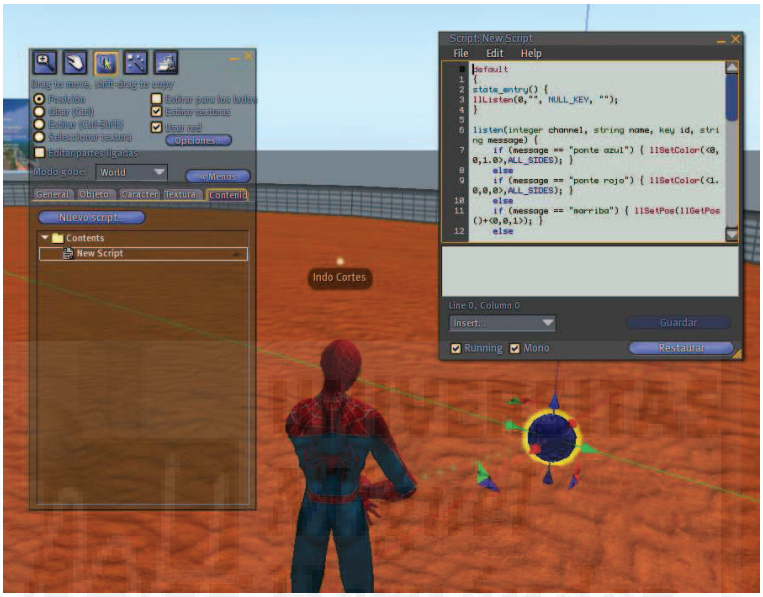


Imagen 6.1.10: Programando en Second Life

4. Comparación con la 1ª propuesta de la tesis. Ventajas e inconvenientes.

Evidentemente, las dos soluciones propuestas cumplen las condiciones impuestas por el problema que plantea la tesis. Sin embargo, una vez estudiada la primera opción (la utilización de Director) vamos a compararla desde el punto de vista que plantea Second Life y así tener una visión más amplia de las dos. La comparación la vamos a resumir en la siguiente tabla:

Característica	Director	Second Life
Tridimensionalidad	Si	Si
Importar objetos	Si	Si
Interactividad	Si	Si
Multimedia	Si	Si
Programable	Si	Si
Publicación en Internet	Si	Si
Publicación en CD-DVD	Si	No

Tabla 10: Comparando Second Life y Director

Entonces, ¿dónde están las diferencias? ¿Porqué utilizar una y no otra solución? Para responder a estas preguntas, debemos comenzar diciendo que no hay una sola respuesta. Las diferencias no son pequeñas, pero la respuesta depende de muchos factores.

Por ejemplo, el perfil del autor y el perfil del espectador de la obra artística deben de estudiarse para tomar una opción u otra. También el tipo de publicación requerida ya que Second Life tiene que ejecutarse siempre on-line.

La primera opción, supone crear la geometría y las texturas desde MAX y luego exportarlas a Director para "montarlas", crear la interactividad con la obra, los manejadores y luego publicarla en Internet o en un DVD. Todo este intercambio de información hace que se tenga la sensación de que todo es un poco más complicado, menos directo que en Second Life. Por lo tanto, podemos pensar que esta opción es para autores con conocimientos suficientes de informática en general, sin miedo a exportar datos entre aplicaciones y procedimientos parecidos. Además, el espectador, debe cargar el plugin Shockwave (si no lo tiene instalado ya) que hace que se retrase más la entrada en el sitio propuesto por el autor, con los problemas de tiempo, actualizaciones, etc., que esto conlleva. Deben ser por tanto espectadores con cierta experiencia, al fin y al cabo, la publicación on-line es entrar en un sitio web especial.

Sin embargo, en la publicación fuera de línea, por ejemplo en DVD, son todo ventajas, ya que se pueden realizar ejecutables que comiencen a visualizarse instantáneamente en cuanto se hace doble clic sobre ellos y además, se ejecutan sin problemas en cualquier ordenador PC y MAC, lo que es una gran ventaja con respecto a Second Life, que requiere no solo un acceso a Internet, sino un programa instalado en el ordenador: el programa cliente de Second Life.

Second Life, se supone que es una solución más compacta, ya que solamente utilizamos un software. Instalamos un programa y ya está, sin problemas típicamente informáticos de actualizaciones, etc. Y esta sensación es la que se transmite al espectador. Si es un usuario de Second Life, basta con que quedemos con él o le demos la dirección de nuestra exposición. En este sentido se debe considerar a Second Life no cómo un sitio web sino como una evolución 3D de la propia web.

De todas formas, en ambas soluciones tenemos un mismo problema común: las dos dependen de empresas y por tanto de su arbitrariedad a la hora de sacar nuevas versiones, incorporar nuevas tecnologías, etc.

En el caso de MAX, la empresa es ®Autodesk y Director depende de ®Adobe, mientras que Second Life es propiedad de ®Linden Labs. En este sentido, vemos una ligera diferencia de Second Life, ya que la evolución de la red hace que la nueva programación para la red tenga más posibilidades y que las nuevas tecnologías sean más fácilmente incorporadas. De todas formas, no olvidemos que el "hermano pequeño" de Director, Flash es omnipresente en Internet y que las nuevas posibilidades 3D, puedan pensar en una convergencia de los programa Director y Flash en un futuro no muy lejano. Por lo tanto, son dos tecnologías muy modernas y potentes pero distintas. Lo idóneo puede ser que podemos mostrar nuestra obra en los dos casos.

5. Comparación de Second Life con otros metaversos.

Debemos distinguir los competidores de Second life en dos categorías: los otros metaversos y las herramientas para crear metaversos.

Esta distinción se debe a que Second Life es creado y publicado por Linden Labs. Nosotros creamos nuestro "mundo virtual" en un metaverso, lo utilizamos, pero no lo creamos. Sin embargo, hoy en día se pueden crear otros metaversos y diseñarlos a medida. Evidentemente esta tarea es mucho más complicada y requiere conocimiento informáticos, si bien no muy profundos, por lo menos fuera del alcance de un estudiante no introducido en estos temas.

De todas formas, en este punto expondremos algunas de las aplicaciones que existen para tener finalmente nuestra obra artística publicada en Internet.

En la red se publican trabajos que comparan los metaversos. Hemos encontrado, por ejemplo, un estudiante que como parte de una tesis ha realizado un trabajo de comparación de herramientas para la creación de metaversos. Se puede encontrar en:

http://sigma39.sigma.cl/dotproject/files/temp/metaversos/formulario_evaluacion.html

Algunas herramientas para la creación de metaversos son: OpenSim, ActiveWorld, Croquet, DarkStart y Ogoglio. Pero dada la alta volatilidad de este tipo de software debemos centrarnos en los más sólidos y utilizados. A saber:

OpenSim: es hoy en día la herramienta para crear nuestro propio metaverso de más éxito y popularidad. Está orientada a las relaciones entre los usuarios. Esta herramienta surge de la liberación por parte de Second Life de su código de la aplicación cliente. Luego unos cuantos entendidos dedujeron el protocolo de comunicación entre el cliente y el servidor Second Life y montan sus propios servidores utilizando ese mismo protocolo. Esta programado en .net y utiliza el protocolo http 1.1.

Imagen 6.1.11: Isla en OpenSim

RealXtend: es una bifurcación de OpenSim de código libre pero solo precompilado para Windows. No tiene ni se augura en un futuro, tanta difusión como su predecesor.

WonderLand: es un metaverso orientado a las relaciones laborales, no personales. Los responsable de Sun Microsystems están de acuerdo en la visión de que los metaversos son la evolución lógica de los navegadores web y compran el protocolo y los servidores de Second Life para la intranet de la empresa, como estudio previo de lo que hoy día es WonderLand. Permite compartir conocimientos y realizar reuniones donde los usuarios se pueden comunicar por voz o chat. Además pueden ejecutar aplicaciones de PC y la ventana de las aplicaciones se visualiza dentro del metaverso como una proyección en una pantalla en la pared de la sala de reuniones.

Imagen 6.1.12: WonderLand

La conclusión es que Second Life está en cabeza de esta tecnología, ya que es el origen del resto de tecnologías. Su amplia difusión y éxito, hace que pueda ser considerado como la

herramienta a utilizar. Sus competidores son muchos y su futuro está en especializarse, por ejemplo, There se ha especializado en captar usuarios jóvenes, OpenSim en usuarios que prefieren no entrar en el negocio de la compra de terrenos de Second Life, WonderLand se usa para reuniones corporativas, etc. Por supuesto que Second Life no es eterna, puede verse superado en cualquier momento y más en un momento tecnológico como el que vivimos, pero hoy en día es el rival a batir, y para un autor artístico, sin duda la opción más segura y con una más amplia gama de espectadores de todo tipo de nivel cultural, poder adquisitivo y dispersión geográfica.

6. Arte en Second Life

Analizar el arte en Second Life es muy difícil. Primero porque casi todo el mundo que realiza algo en este metaverso es en cierta medida un autor artístico. Por eso, en este punto introductorio vamos simplemente a mostrar los elementos artísticos más comunes como son los museos, los galeristas y los artistas. En Second Life existen muchos museos, uno de los más reconocidos es “The Tech Virtual Museum” que acoge múltiples exposiciones fijas e itinerantes y que colabora con otros museos de la vida real, además se ha convertido en plataforma para el intercambio de obras virtuales y por ejemplo, participa en el día mundial de los museos que se celebra el 18 de mayo.

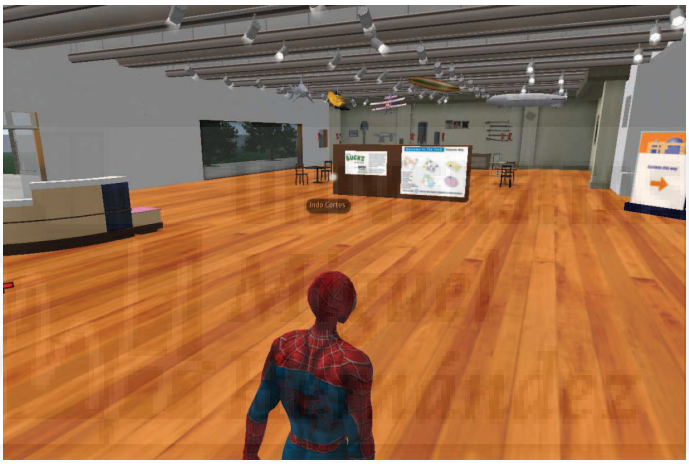


Imagen 6.1.13: The Tech Virtual Museum

Imagen 6.1.14: Galerías de arte en Second Life

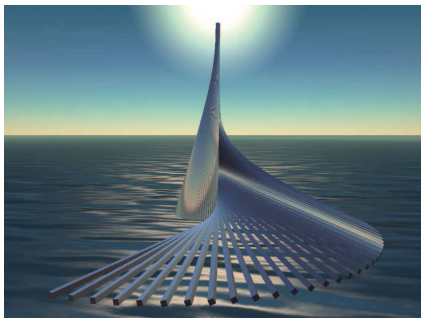
Los galeristas y los artistas son un grupo muy dinámico en Second Life. Por ejemplo, existe un grupo que lidera Sasun Steinbeck y que mantienen un listado de todos los artistas que se integran en su grupo y un blog muy activo llamado “Art Galleries of Second Life”.

Con respecto a los artistas, podemos decir que la variedad de tendencias y tipología de obras es tan extensa, si no más, que en el mundo real. Se da muy a menudo que un artista comparte su arte en los dos mundos al mismo tiempo, este es el caso de Jeffrey Lipsky que en Second Life es Filthy Fluno. Artista reconocido y con referencias en “The New York Times” en sus dos vertientes.

Imagen 6.1.15: Jeffrey Lipsky - Filthy Fluno y sus obras reales y virtuales

Por otra parte tenemos a los autores “clásicos”, entendiendo por estos a los que presentan obrar escultóricas utilizando las herramientas propias de Second Life y que tienden a tener un carácter matemático, geométrico.

Un buen ejemplo es Henry Segerman, un artista representativo de las obras que se pueden crear y exponer en Second Life (http://www.segerman.org/2ndlife.html) de las que se exponen algunos trabajos que mezclan la geometría de objetos y la incorporación de scripts con fines artísticos. Su nombre en Second Life es “Seifert Surface” Henry Segerman lleva muchos años utilizando Second Life como un laboratorio. Entre sus obras destaca una amplia colección de arte fractal. Algunas de sus obras comenzaron a fraguarse en 2004, por lo que su experiencia es muy grande. En la realidad es doctor en matemáticas y trabaja como tal en la universidad de Autin, Texas.


"Archimedean Spire", 12-2006


"Negative curvature Fractal 4", 09-2005


"Noobility", 08-2008

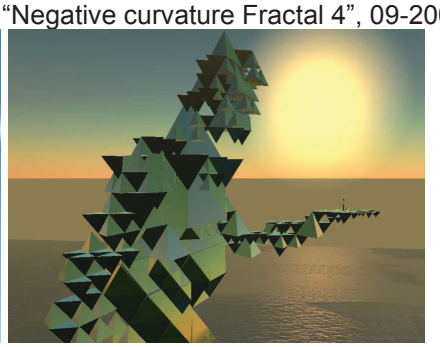

"Tetrahedrom man", 08-2006

Imagen 6.1.16: Algunas obras de Segerman

Como se puede comprobar en sus obras, son esculturas de clara inspiración matemáticas donde la geometría en tres dimensiones y los fractales se aprecian constantemente. Para ver las obras expuestas debemos visitar el "Sculpture garden".

Imagen 6.1.17: Jardín de las esculturas de Segerman

Algunos de los sitios para visitar y saber más sobre el arte en Second Life son:

The visual artist in Second Life, blog sobre artistas visuales que se encuentra en http://slartist.wordpress.com/

Art and artist network!, donde podemos encontrar noticias de eventos SL de arte. Lo podemos ver en http://slartistnetwork.proboards.com/index.cgi

Open this end, grupo para la investigación del arte en Second Life.

7. Qué experimentar en Second Life.

En Second Life se puede hacer de otra manera casi todo lo que podemos hacer en la vida real como bailar, trabajar, ir de tiendas, buscar pareja, jugar, educarse, etc. Pero además también podemos experimentar lo que nos proporcionan los videojuegos como explorar tierras de fantasía, ir a lugares fuera del tiempo y conocer a gente que puede ser y vestirse cómo quiera, podemos hablar con dragones, piratas y robots. Siempre sabiendo que detrás de un avatar hay una persona real. Podemos experimentar un salto de paracaídas, y aunque no en las mismas condiciones que en la vida real, cada día el realismo y la inmersión son mayores.

- Aprender

En Second Life tienen sede muchas de las principales instituciones educativas del mundo como universidades y centros de cultura.

Existen universidades como Harvard que ofrecen cursos a través de Second Life, con todas las garantías y con la comunicación directamente con los profesores titulares. Esto ayuda a la democratización de la cultura además de no requerir asistencia real.

Por ejemplo**, "**La isla de la Casa Encendida" es hasta la fecha, el proyecto español más ambicioso. En la vida real la Casa Encendida es un centro cultural situado en Madrid, concretamente en Lavapiés. En Second Life han reproducido la forma del edificio y se pueden realizar muchas actividades sociales, exhibiciones culturales y conciertos. También se puede pasear por sus jardines. Además han creado unos cursos para incorporarse a Second Life, construyendo avatares y mostrando sus posibilidades.

Imagen 6.1.18: Proyecto cultural español, la casa encendida

Otro ejemplo es el Instituto Cervantes, que podemos encontrar en Second Life en http://secondlife.cervantes.es/es/default.htm

Otro ejemplo es Sloodle. Este es un proyecto de integración entre Second Life y Moodle que permite la rápida adaptación de cursos de esta exitosa plataforma educativa en el metaverso.

- Hacer negocios

Se ha escrito mucho sobre esta vertiente de Second Life: la de ser una economía paralela, virtual en los modos pero real en los resultados.

Debemos comenzar escribiendo que Second Life no es gratis para los negocios. Cuando se construye un negocio en Second Life necesitas terreno y objetos y esto ocupa un espacio real en los discos duros de los servidores que son propiedad de Linden Labs y por ello cobran de igual manera que cuando un negocio publica su sitio web paga a una empresa de hosting para que esté accesible en la red. Lógicamente se paga más si el terreno es mayor. Esto es así de evidente, pero aún existen ciertas reticencias simplemente por la lógica resistencia al cambio de modelo.

El marketing que funciona hoy en día es el marketing viral, el boca a boca.

Somos de la opinión que hoy en día, no se puede usar Second Life como una alternativa mas para crear negocios que nos permitan ganarnos la vida. Es cierto que se producen negocios en Second Life pero debemos matizar estos hechos.

Debemos distinguir entre los negocios pequeños, de una o pocas personas y los macronegocios de grandes empresas.

Actualmente creemos que los micronegocios se pueden y se deben instalar en Second Life como reclamo y complemento de uno existente en la vida real. Por ejemplo, una academia real puede complementar sus clases en Second Life, pudiendo atender a sus alumnos de vacaciones o hospitalizados temporalmente. O como ayuda para el aprendizaje de idiomas o la experimentación en geometría, etc. Otro ejemplo puede ser una librería, y desde allí vender libros electrónicos, tomar pedidos, etc.

Los negocios personales no son de momento lucrativos pues el esfuerzo no es recompensado, no nos podemos fiar de los que dicen que se han hecho millonarios. Además deben ser modelos de negocios dinámicos, debe cambiar constantemente para ser atractivos y esto solo se lo pueden permitir los grandes inversores.

Los macronegocios no creemos que puedan recuperar la inversión realizada en términos de ganancias netas pero sí en otros importantísimos factores como son la presencia de marca, publicidad, muestras en tres dimensiones y además en un entorno dónde el perfil del público es muy alto en cuanto a nivel profesional, cultural y adquisitivo. Por lo tanto, no creemos que las grandes compañías deban de hacer las inversiones pensando en un retorno rápido del dinero, sino más bien como una experiencia nueva, una nueva forma de comunicación en su relación con potenciales clientes

Por ejemplo, la presentación de hoteles, las pruebas virtuales de edificios o interiorismo es una buena forma de utilización ya que en Second Life se pueden mostrar fielmente las dimensiones de las construcciones con respecto al tamaño de las personas y tiene un alto grado de inmersión por lo que los clientes se pueden hacer una idea bastante aproximada de cómo son los edificios, los cruceros, los coches, etc. Además pueden atender a los comentarios de los clientes, etc.

De todas formas, Second Life requiere de un alto compromiso de las empresas, sean grandes o pequeñas, ya que no basta con realizar un hotel virtual y ponerlo en una isla para que los clientes lo busquen y lo visiten. Por una parte, necesita publicidad en sí mismo para ser encontrado en Second Life, y por otra se le debe de acompañar cuando llega, es decir, debe haber un avatar que le muestre el producto y se lo venda. Aquí pasa como en la vida real, aunque tengas toda la información en la web de una entidad bancaria, antes de hacer una operación, aún queremos ir en persona y mirar a la cara a nuestro gestor. En este sentido existen mecanismos como llamadas a móviles o e-mails para avisar de la llegada de clientes-avatares a una determinada zona.

Como conclusión podemos decir que Second Life es una ventana al futuro, casi seguro que el futuro de las relaciones entre los seres humanos por medio del ordenador no se hará mediante Second Life, pero sí con una forma muy parecida. Posiblemente los metaversos son la forma que tendrá el Internet del futuro próximo.

Unidad 2: Herramientas y procedimientos básicos en Second Life.

1. Cómo ser miembro de Second Life.
2. Cambiar la apariencia de nuestro avatar.
3. Cómo moverse.
4. Relaciones humanas: chatear, hablar, grupos de amigos.
5. Cómo crear objetos.
6. Cómo guardar cosas. El inventario.
7. Cómo exponer nuestro trabajo off line: Vídeo y Foto.
8. Permisos en Second Life: intelectuales, de suelo y otros.
9. Comprar objetos.
10. Comprar terrenos.
11. Resumen de atajos de teclado para distintas funciones.

1. Cómo ser miembro de Second Life.

En este punto vamos a ver los pasos lógicos para comenzar en Second Life. Estos se pueden enumerar como sigue:

1. Comprobar los requisitos mínimos de Second Life.
2. Crear una cuenta.
3. Descargar e instalar el programa.
4. Ejecutar el programa.
5. Gestión de la cuenta.

Desarrollaremos cada uno de estos puntos:

1. Requisitos mínimos de nuestro sistema para instalar Second Life.

Como es normal, estos requisitos cambian con cada nueva versión del programa y ya sabemos que existen requisitos mínimos oficiales y reales, por lo que recomendamos visitar alguna página actualizada como http://second-life.softonic.com/ y la web oficial de Linden Lab. http://secondlife.com/support/sysreqs.php.

Los requisitos para Windows, a día de hoy, son los que se muestran seguidamente:

Windows	Minimum Requirements	Recommended
Internet Connection*:	Cable or DSL	Cable or DSL
Operating System:	XP, or Vista	XP or Vista
Computer Processor:	800 MHz Pentium III or Athlon, or better	1.5 GHz (XP), 2-GHz (Vista) 32-bit (x86) or better
Computer Memory:	512 MB or more	1 GB or more
Screen Resolution:	1024x768 pixels	1024x768 pixels or higher
Graphics Card for XP**:	• NVIDIA GeForce 6600 or better • **OR** ATI Radeon 8500, 9250 or better • **OR** Intel 945 chipset	**NVIDIA Graphics cards** 6000 Series: • 6600, 6700, 6800 7000 Series: • 7600, 7800, 7900 8000 Series: • 8500, 8600, 8800 GeForce Go Series: • 7600, 7800, 7900 **ATI Graphics Cards** • X800, X900, X1600, X1700, X1800, X1900 • x2600, x2900 • x3650, x3850
Graphics Card for Vista (requires latest drivers)**:	• NVIDIA GeForce 6600 or better • **OR** ATI Radeon 9500 or better • **OR** Intel 945 chipset	**NVIDIA Graphics cards** 7000 Series: • 7600, 7800, 7900 8000 Series: • 8500, 8600, 8800 GeForce Go Series: • 7600, 7800, 7900 **ATI Graphics Cards** • X1600, X1700, X1800, X1900 • x2600, x2900 • x3650, x3850

Imagen 6.2.1: Requisitos de Second Life para un sistema Windows

2. Crear una cuenta.

Ir a nuestro navegador web y escribir la dirección http://secondlife.com. Aparecerá la web como la que se puede ver en la imagen siguiente. Pulsamos sobre el botón “Get Started” de la página de Second Life o directamente vamos a la página https://join.secondlife.com/.

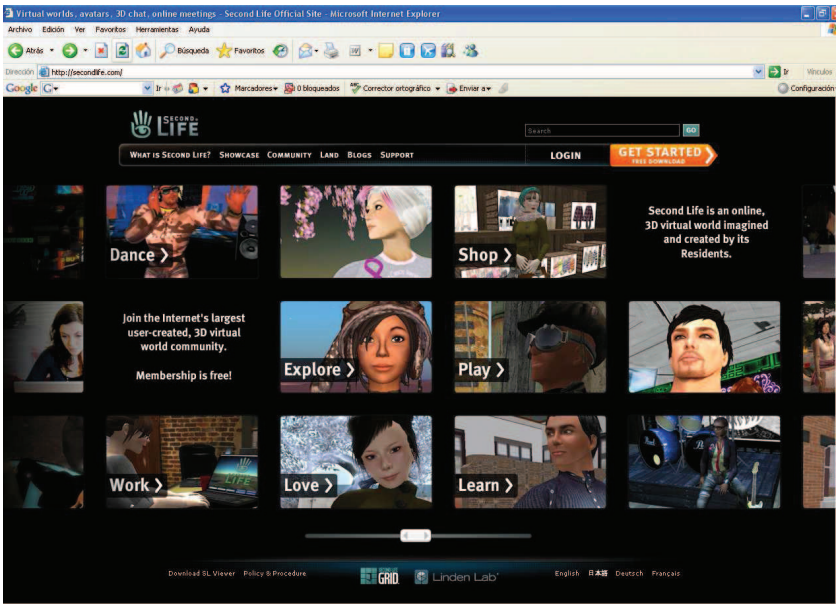


Imagen 6.2.2: Sitio web de Second Life

Imagen 6.2.3: Página en español para crear una cuenta Second Life

Pulsa sobre el menú de arriba y elige “español”. Después de rellenar todos los datos, pulsamos sobre el botón “Crear la cuenta” y luego debemos confirmar los datos del correo electrónico que has dado en la cuenta.

3. Descargar e instalar el programa cliente de Second Life.

Igual que para navegar en 2D necesitamos un navegador (Explorer, Opera, Firefox, etc.), en 3D necesitamos instalar una aplicación. Al pasar a la página de “Descargar e instalar” aparece la típica ventana para bajar archivos de Internet y nos preguntará el sistema operativo que utilizamos, una vez resueltas estas cuestiones, pulsamos el botón Ejecutar.

4. Ejecutar el programa.

Luego de instalar el cliente de Second Life, nos presentará una pantalla de inicio. En la primera pantalla aparecen unos datos: hora (de California), las entradas de usuarios en el último mes y los usuarios presentes en este mismo momento.

Imagen 6.2.4: Página de inicio de Second Life

5. Gestión de la cuenta.

Si en cualquier momento tenemos que cambiar algunos datos de nuestra cuenta como cambiar el correo electrónico, nuestra contraseña, comprar o vender L$ o tierra, darnos de baja, recordar una contraseña olvidada, etc., tenemos que acceder al portal oficial de Second Life en la siguiente dirección:
https://secure-web2.secondlife.com/account/login.php?type=second-life-member&nextpage=/account/index.php/

2. Cambiar la apariencia de nuestro avatar.

En la página dónde creamos nuestra cuenta tenemos una serie de aspectos iniciales que van cambiando según las versiones de Second Life.

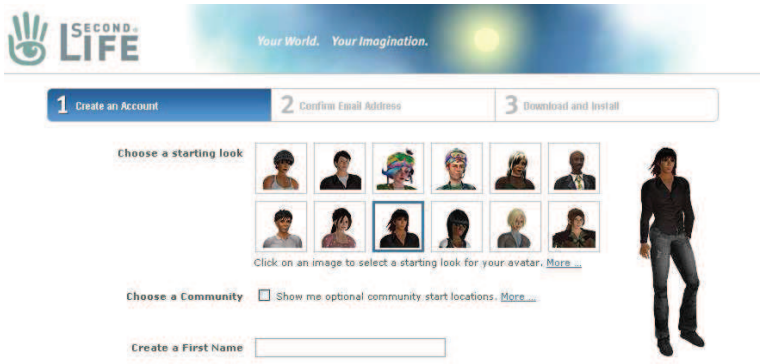


Imagen 6.2.5: Elección de la apariencia inicial de nuestro avatar

Poder cambiar nuestro aspecto es fundamental en Second Life ya que va a ser la imagen que perciben todos los demás y por tanto un rasgo fundamental de nuestras relaciones sociales. Al poco tiempo de comenzar en Second Life, nos daremos cuenta que los usuarios gastan mucho tiempo y dinero en conseguir aspectos muy sofisticados.

Lo primero que debemos hacer es salvaguardar nuestra apariencia actual por si no nos gustan los cambios y queremos recuperarla. Esto se hace pulsando con el botón derecho sobre nuestro avatar, luego elegimos “Apariencia” y “Guardar como” y le damos un nombre. Nuestra apariencia actual se guarda directamente en el inventario.

Para comenzar a cambiar nuestra apariencia debemos tener en cuenta que podemos modificar todo a la vez, sólo algunas partes o podemos añadir o dejar complementos.

Cambiar nuestra apariencia por entero supone, por ejemplo, cambiar todo el cuerpo, cambiar de sexo o cambiar toda la ropa a la vez. Para esto último, iremos al Inventario > clothing > y arrastramos una determinada carpeta sobre el avatar.

Imagen 6.2.6: Cambio total de apariencia al arrastrar una carpeta

Es muy cómodo que toda una apariencia, es decir, anatomía, ropa y complementos esté en una única carpeta ya que si queremos donarla (y no hay problemas de permisos como es el caso de la ropa de Spiderman que llevo puesta en las imágenes) a otro avatar basta con acceder a nuestro inventario y arrastrar esta carpeta al avatar que queramos.

Para modificar alguna parte específica como altura, color, pelo, etc. Pulsamos con el botón derecho sobre nuestro avatar > Apariencia (o menú Edit > Apariencia) y luego elegimos lo que queramos cambiar, por ejemplo: Shape > Body > Height (o por ejemplo Nose > Nose size). Si no nos gustan los cambios pulsamos el botón “Revert”. Si queremos aceptar los cambios elegiremos la opción “Guardar como…”

Imagen 6.2.7: Cambio anatómico al acceder a los datos de Apariencia

En la imagen superior se puede ver cómo hemos cambiado de sexo y ahora somos una chica, y más bajito y rechoncho modificando los parámetros de altura, delgadez y sexo.

Para poner y quitar accesorios como gafas, sombreros, espadas, etc. es evidente que los tenemos que haber conseguido comprándolos o creándolos y por lo tanto estarán en nuestro inventario. Entonces lo que haremos será arrastrarlos del inventario directamente hasta el lugar del cuerpo dónde queremos que se “peguen”. Por ejemplo, la espada en la mano o el sombrero sobre la cabeza.

Para quitar un complemento (por ejemplo unas gafas situadas sobre la nariz) pulsamos con el botón derecho sobre nuestro avatar, entonces aparecerá el menú contextual del avatar y separar > cabeza > nariz y luego botón derecho sobre las gafas > separar.

Imagen 6.2.8: Poner y quitar complementos de todo tipo

En la imagen anterior hemos utilizado unas gafas que traen los avatares nada más crearlos y que se encuentra en el inventario en Library > Clothing > Musician Guy by Nilon Pinkney > Musician Guy sunGlasses (también podemos utilizar la herramienta de buscar y escribir "glasses" para encontrarlas).

3. Cómo moverse.

Poder desplazarnos con facilidad por Second Life, es algo fundamental, ya que es un lugar muy grande que crece constantemente.

Tenemos muchas herramientas para movernos, pero lo primero que debemos de tener en cuenta es que podemos desplazar dos entidades: nuestro avatar y nuestra cámara, es decir, nuestra vista. Esto quiere decir que podemos ir andando por una calle y observar nuestro alrededor mediante la vista de la cámara normal (situada ligeramente encima y atrás de nuestro avatar por eso lo vemos a él mismo y no desde el punto de vista de los ojos del avatar), pero si por algún motivo no queremos o no podemos movernos, podemos dirigir la cámara para separarla de nuestro avatar y por tanto ver los alrededores sin mover el avatar de sitio.

Comenzaremos describiendo los métodos para desplazar nuestro avatar. Podemos desplazarnos por cuatro métodos: andando, corriendo, volando y mediante teletransporte. No vamos a considerar los pequeños desplazamientos originados por animaciones o bailes.

Para andar, basta con utilizar las teclas de flechas. En la siguiente imagen se puede observar la ventana que nos permite controlar los movimientos básicos del avatar al pulsar sobre los botones, para que aparezca esta ventana pulsamos sobre el comando Ver > Controles de movimiento. Estos son:

Función	Atajo de teclas
Movernos hacia delante	Flecha ↑
Movernos hacia atrás	Flecha ↓
Movernos hacia la izquierda	Shift + Flecha ←
Movernos hacia la derecha	Shift + Flecha →
Girarnos hacia la izquierda	Flecha ←
Girarnos hacia la derecha	Flecha →
Saltar	Re pág
Agacharse	Av pág

Imagen 6.2.9: Ventana de controles de movimiento del avatar

También podemos correr. Para ello, debemos poner el modo correr con Ctrl + R y para quitarlo volveremos a pulsar Ctrl + R otra vez.

Cuando estamos en este modo todo el movimiento se hará corriendo, por lo que las funciones de andar también sirven para correr. También parece que los saltos y el agacharse se realizan con más dinamismo.

Otra opción para desplazarnos, es volar. Volar es una forma muy rápida y relajante de ver nuestro entorno. Para volar pulsaremos la tecla "Inicio", para dejar de volar volveremos a pulsar la misma tecla. También podemos usar el botón "Volar" que aparece en la parte inferior de la pantalla de Second Life.

Las funciones de dirección son las mismas que para andar, por lo que para orientar el vuelo utilizamos las teclas de flechas. La teclas de Re pág y Av pág estando en el modo vuelo sirven para subir más alto o bajar, es decir, controlan la altura del vuelo.

Imagen 6.2.10: Visión de toda una isla mediante un vuelo de altura

El último modo para desplazarnos es el teletransporte. El teletransporte nos permite cubrir grandes distancias rápidamente. Por ejemplo, si estamos en una isla bailando y queremos ir a otra isla con un monasterio para meditar, utilizaremos el teletransporte. Para usar el teletransporte se tiene que saber dónde ir. Para ello existen dos métodos, desde dentro de Second Life con las landmark o desde la web con las SLurl.

Si estamos dentro de Second Life y por cualquier motivo estamos en un sitio, podemos registrar una marca de tierra o “Landmark”, que almacenará en el Inventario las coordenadas de ese sitio junto con su nombre. Esto sirve para volver a ese sitio posteriormente, o para pasárselas a un amigo, o lo que queramos, es como una lista de sitios favoritos de Second Life.

Para crear un landmark tenemos que ir al lugar deseado, lo más lógico es pulsar sobre el botón Mapa. En el apartado “Search” ponemos cualquier palabra que nos interese, por ejemplo “Alicante” y si encontramos uno o varios sitios, seleccionamos el que queramos y pulsamos “Teleport”. Cuando lleguemos, si nos gusta podemos grabar sus coordenadas como una landmark, para ello usaremos el comando Mundo > Crear punto de referencia aquí.

Por ejemplo, en la imagen siguiente hemos seguido este método para buscar un lugar llamado “Alicante” y una vez visitado, hemos creado un landmark.

Otro método para ir a un determinado sitio son las SLurl. Las SLurl son enlaces desde la web a direcciones dentro de Second Life. Por ejemplo, si estamos navegando y visitamos una página web de una empresa que tenga sede en Second Life, seguramente nos presente la posibilidad de visitarla mediante un enlace SLurl. Este enlace abrirá el cliente de Second Life y nos teletransportará automáticamente.

Imagen 6.2.11: Crear una landmark o punto de referencia

Las cámaras y sus implicaciones en Second Life las estudiaremos con más detenimiento en un tema propio, por ello, ahora nos vamos a limitar a enumerar las funciones de control de la cámara por defecto. Estas son:

Modo	Teclado y ratón	Función
Focus	Alt + botón izquierdo del ratón	Zoom sobre un objeto
Orbital	Alt + Control + botón izquierdo del ratón	Orbitar sobre un objeto
Encuadre	Alt + Control + Shift + botón izquierdo del ratón	Encuadre de cámara
Giro	Teclas de flechas izquierda y derecha	Giro a izquierda y derecha
Zoom	Botón central rotatorio del ratón	Zoom de la vista de cámara

En la imagen 6.2.12 se puede observar la caja de diálogo llamada “controles de cámara”. Esta diálogo la podemos visualizar desde el comando Menú > Controles de cámara y aparecerá centrada en la parte superior.

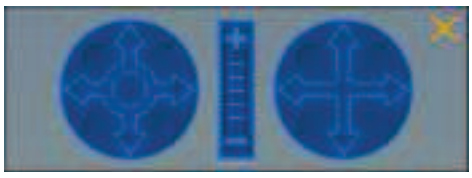


Imagen 6.2.12: Ventana de control de cámara

Se puede observar que tiene 3 controles. El de la izquierda es el “orbital”, el del centro controla el “focus” y el de la derecha controla el “encuadre”. Podemos utilizar esta caja de diálogo para acostumbrarnos a controlar la cámara antes de utilizar el teclado y el ratón.

En este punto también deberíamos nombrar dos herramientas fundamentales que aporta Second Life para poder visitar el metaverso: el mapa y el “minimapa”.

El mapa es una ventana que se activa y desactiva con las teclas Ctrl + M o con el botón “Mapa” de la barra inferior. Nos muestra el mapa de Second Life más cercano al punto donde nos encontremos. En el menú de la derecha encontramos distintas entidades para buscar lo que deseemos, personas, terrenos en venta, etc. También tenemos un buscador de sitios por palabras que es muy útil y que permite el teletransporte. A veces el mapa es demasiado grande para buscar cualquier cosa directamente en nuestro alrededor y por ello, es más usado el minimapa sobre todo si ya estamos en el lugar deseado.

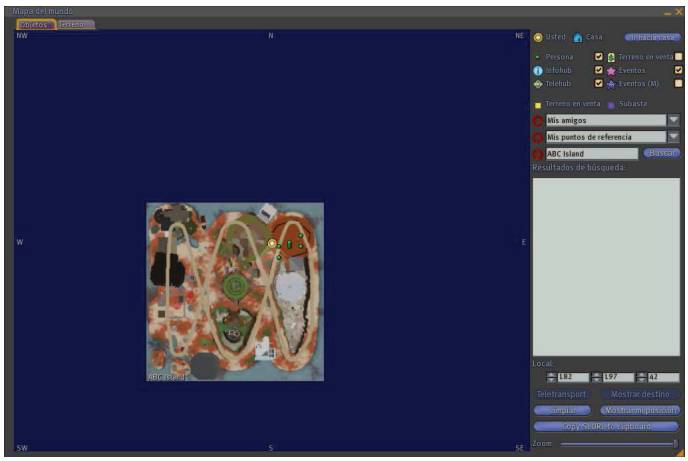


Imagen 6.2.13: Mapa

El minimapa es una ventana independiente que se activa y desactiva con las teclas Ctrl + Shift + M. Nos muestra un área alrededor nuestro donde el avatar propio aparece como un punto amarillo y los demás avatares como puntos verdes. También se visualiza un triángulo para saber nuestra dirección, es decir, hacia dónde estamos mirando.

Imagen 6.2.14: Minimapa

4. Relaciones humanas: chatear, hablar, grupos de amigos.

Second Life surge como una nueva forma de establecer relaciones entre las personas y esto se traduce en una serie de herramientas que junto con un entorno más natural que el bidimensional que tanto ha triunfado en Internet, hace augurar un gran desarrollo en el campo de la comunicación interpersonal para este metaverso. Podemos comunicarnos directamente con los avatares que están a nuestro alrededor. También podemos mantener una lista de amigos o de grupos a los que estemos inscritos y otras alternativas que pasamos a estudiar brevemente.

Chat: esta herramienta se utiliza para charlas con los avatares que tenemos cerca. Su manejo es tan sencillo como acceder a la línea de comandos y escribir lo que queramos. Es la manera más utilizada de comunicación. Tenemos que tener en cuenta que casi todo el mundo es de habla inglesa si no estamos en una isla determinada, por lo que es una buena herramienta para practicar el inglés.

Imagen 6.2.15: Utilizando el chat

En la imagen superior se puede observar lo fácil que es establecer relaciones con los avatares que tenemos cerca mediante el chat. Solo tenemos que escribir lo que queramos en el hueco marcado en la imagen con una flecha roja y las líneas de la conversación se visualizarán en una pequeña lista en la parte de arriba. La flecha verde señala un acceso a amigos y grupos y la flecha azul al chat de voz.

Lista de amigos: a esta lista accedemos desde la barra de herramientas, con el botón Amigos. Con ella podemos darlos de alta y de baja, hablar directamente con ellos u ofrecerle que se teletransporte a nuestro lado. Para añadir a alguien como amigo, hacemos clic con el botón izquierdo sobre su avatar y seleccionas la opción “Add a friend”. Si la otra persona acepta esa invitación de amistad, aparecerá en la lista.

Para acceder a una caja de diálogo que reúne a los amigos y grupos, hacemos clic con el botón derecho sobre nuestro propio avatar y elegimos la opción “Grupos” o directamente sobre el botón señalado por una flecha verde en la imagen anterior, entonces aparecerá la ventana que se observa en la siguiente imagen.

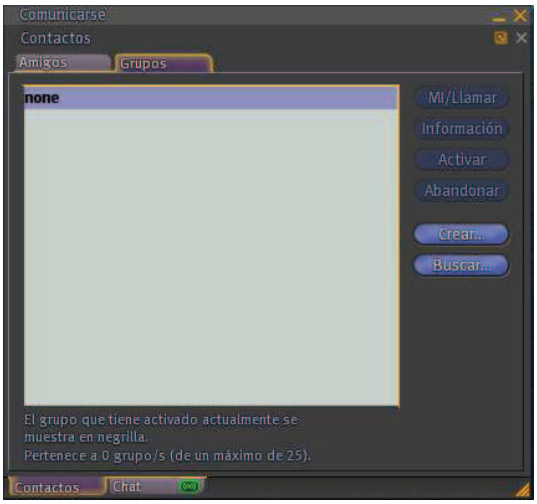


Imagen 6.2.16: Comunicarnos con grupos de amigos

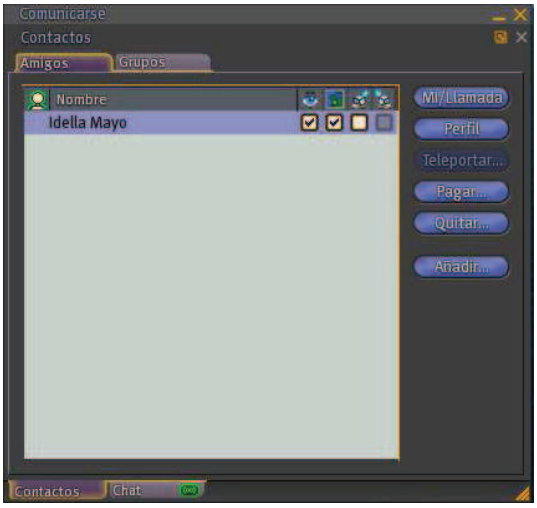


Imagen 6.2.17: Comunicarnos con amigos

Chat de voz: es un chat donde se pueden utilizar directamente un micrófono y unos altavoces para mantener una conversación con los avatares que estén cerca. Se trata de una interacción síncrona entre todos los avatares a nuestro alrededor. Podemos selecciona el volumen y saber quien está hablando.

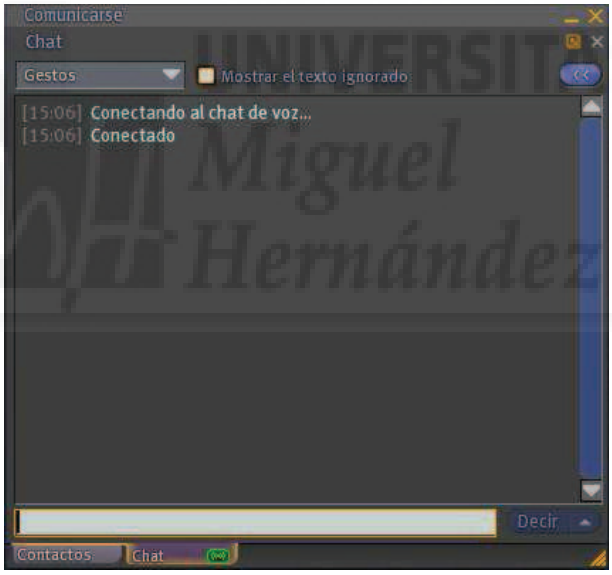


Imagen 6.2.18: Comunicarnos mediante chat de voz

I/M: mensajería instantánea, esta es una herramienta para realizar mensajería escrita e individual. Por ejemplo, un profesor podría dar por escrito instrucciones a los alumnos, etc.

Notas de texto: dentro del inventario podemos crear notas escritas o notecards que podemos compartir con otros avatares y por tanto podrían ser manuales, exámenes, etc.

Nuestro avatar también puede dar tarjetas a otros usuarios, las tarjetas de llamada o “calling cards” se guardan en el inventario en un directorio homónimo y únicamente se usan para saber si alguien está conectado o no, y si fuera así, para hablar con él mediante mensajes instantáneos.

Second Life permite una comunicación que no es muy habitual y a veces, siquiera posible en otros sistemas: la comunicación no verbal, o sea, la comunicación gestual. Esto se lleva a cabo mediante un botón que está por el centro de la barra de herramientas llamado “gestos”.

También podemos acceder a una caja de diálogo para gestionar estos gestos haciendo clic con el botón derecho sobre nuestro avatar y elegir en el menú contextual la opción “Gestos”.

Estos gestos son muy habituales y crearán una animación acompañada de un sonido que todos a nuestro alrededor podrán ver y escuchar. La lista aparece en ingles.

Algunas de estos gestos son hacer aplausos, estar aburrido y bostezar, hacer músculos, negar de manera vehemente, etc., por lo que tenemos que tener cuidado si los probamos para que estemos a solas ya que los demás avatares pueden deducir una comunicación errónea por nuestra parte.

Estos gestos se pueden adquirir y por tanto modificar la lista de gestos disponibles y también se pueden construir por el propio usuario aunque es un tema un tanto difícil que además requiere de terceras aplicaciones.

En la imagen siguiente se puede observar los gestos de que disponemos en la lista inicial de gestos. Normalmente son más que suficientes para expresiones habituales.

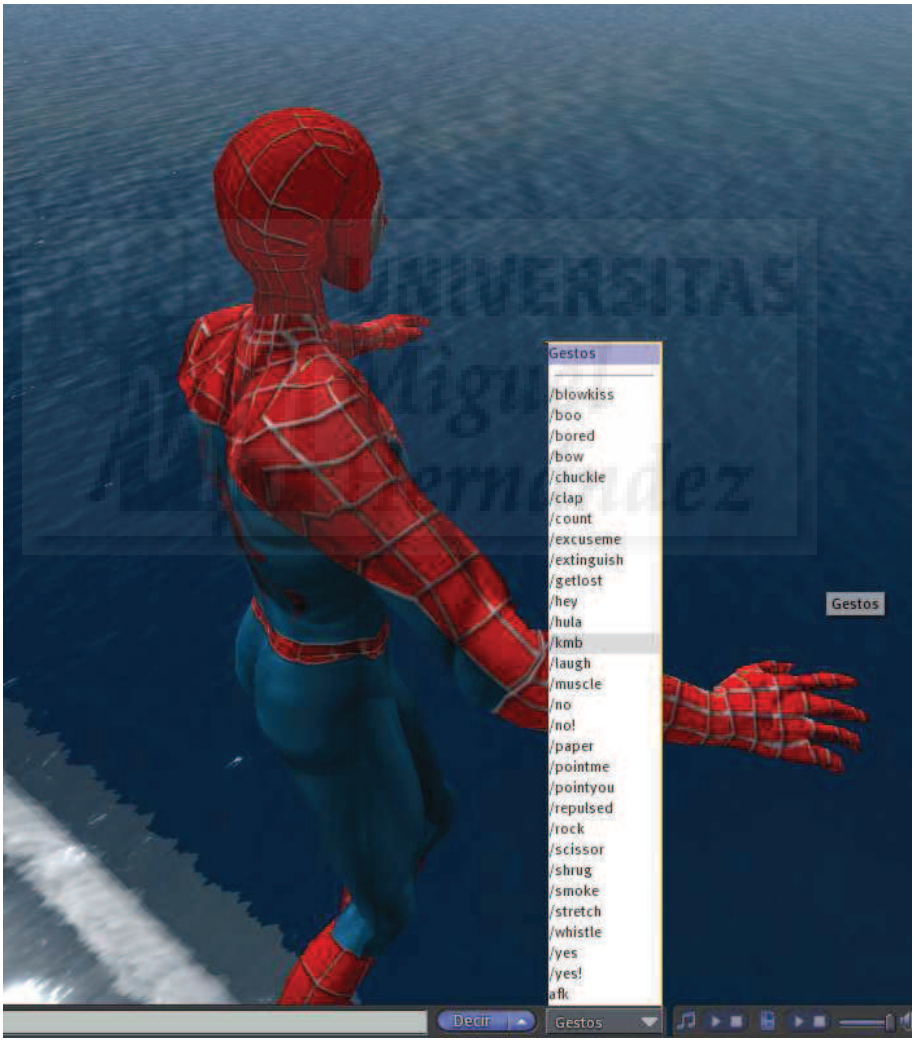


Imagen 6.2.19: Comunicación por gestos

5. Cómo crear objetos.

En la siguiente imagen se puede apreciar en la parte superior una vista de una determinada isla tal como se puede visualizar por cualquier usuario y en la parte de abajo las cajas que

circunscriben la geometría de todos y cada uno de los objetos que existen y que se pueden apreciar desde esta vista.

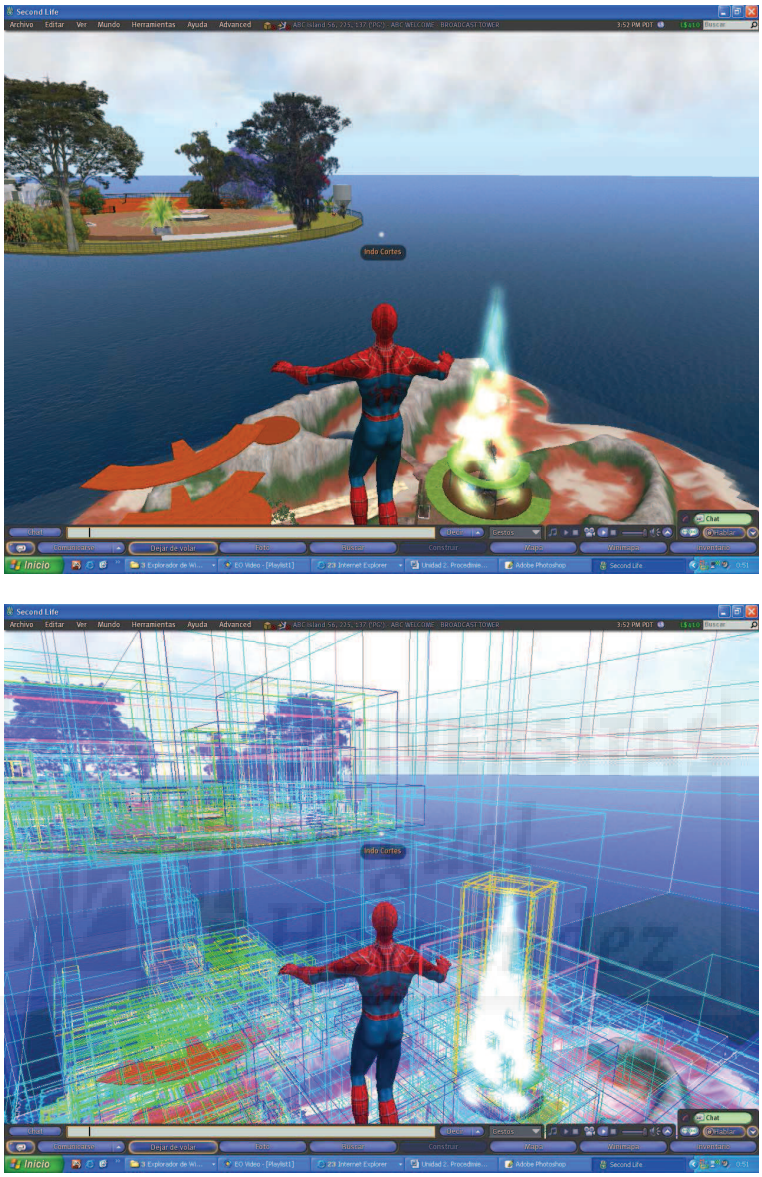


Imagen 6.2.20: La complejidad de las creaciones en Second Life

Aunque este tema lo trataremos en profundidad en el siguiente tema. Es importante señalar que Second Life está creado por los propios usuarios. Todo lo que vemos, salvo el terreno de las islas y el mar, lo han creado usuarios de Second Life con sus conocimientos y su técnica. En la imagen anterior se puede observar árboles, pájaros, automóviles, fuentes, valles, lagos,… y todo esto lo hemos creado los usuarios mediante una mezcla de geometrías y scripts para darles “vida”, es decir, interacción, movimiento, etc.

Para crear objetos nuevos en Second Life tenemos dos opciones: tener terreno o utilizar areneros o como se les conoce en Second Life, sandboxs. Nosotros en este documento utilizaremos sandboxs ya que son gratuitos y tenemos que partir de la premisa que supone la realización de obras 3D interactivas y si puede ser gratis.

En los areneros o sandbox podemos crear lo que queramos, por ejemplo, en la imagen se puede observar que alguien ha creado una casa muy grande, tanto que he tenido que volar y alejarme para tomar esta instantánea, es tan grande que casi no cabe en el arenero que yo mismo utilizo normalmente para mis creaciones.

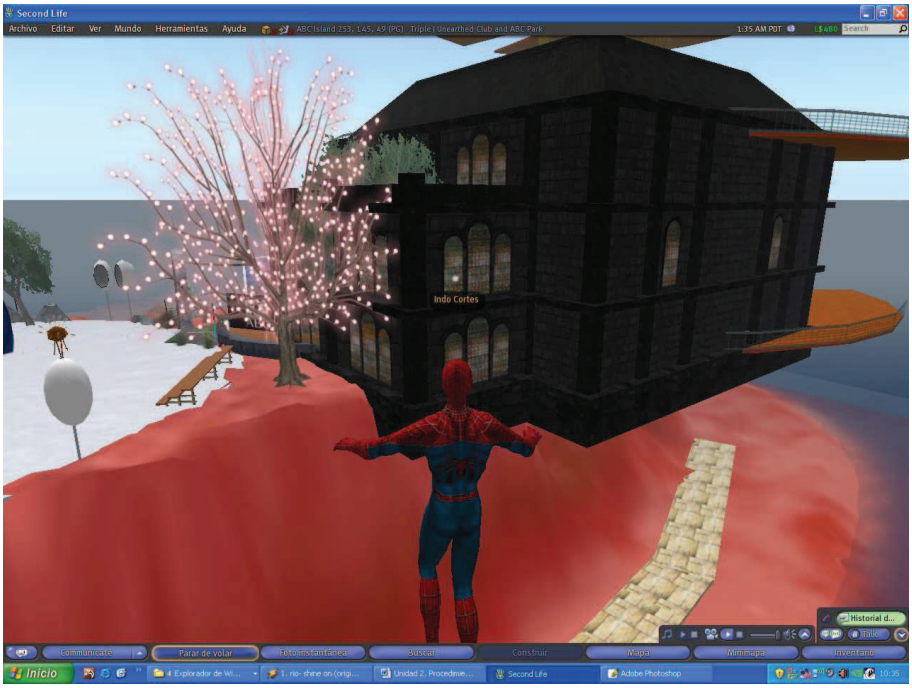


Imagen 6.2.21: Magnitud de las creaciones

Cuando hacemos un objeto, lo podemos guardar en nuestro inventario con lo cual no lo perdemos y aunque no esté en permanente exposición, lo tenemos siempre a mano para mostrarlo siempre que lo deseemos.

Podemos guardar obras que incluyan geometría, texturas, sonidos, programación, utilidades, etc. Podemos guardar desde una hormiga hasta un rascacielos, por lo que es un buen sistema para poder construir verdaderas obras digitales, obras que tengan unas medidas relativas o absolutas según nos convenga.

6. Cómo guardar cosas. El inventario.

El inventario nos permite almacenar todo lo que tenemos como usuarios de Second Life, lo que vamos poseyendo durante nuestra vida virtual, desde objetos que hacemos o compramos hasta texturas, notas, etc. Es un almacén propio de cada usuario de Second Life y él es responsable de mantenerlo.

Normalmente son muchos los artículos que podemos tener en el inventario y por ello, tiene capacidades de organización, búsqueda y aplicación de filtros.

Para abrir el Inventario basta con pulsar el atajo de teclado Ctrl + I o el botón más a la derecha de todos como se muestra en la siguiente imagen.

Imagen 6.2.22: Exposición de un objeto del Inventario

En la imagen superior se puede observar un objeto que ha sido extraído del Inventario arrastrándolo hasta el suelo de un Sandbox, con lo cual puede ser visto por el resto de usuarios e interactuar con él.

En principio, el inventario no tiene límites de tamaño ni de número de artículos admitidos, podemos introducir todo lo que queramos, pero esto no siempre ha sido así, antes estaba limitado el número de objetos que podíamos tener. En la actualidad también tenemos que tener cuidado con todo lo que guardamos y gestionarlo, ya que cuanto mayor es el inventario, mayor puede ser el "lag", aparte de ser menos eficiente el tener un almacén lleno de objetos que no utilizamos.

Podemos tener muchos tipos de artículos en nuestro inventario y para cada tipo existe una carpeta y si es necesario podemos crear otras. Estos son:

- Animaciones: movimientos para enriquecer las aptitudes de nuestro avatar.
- Body parts: formas, partes y objetos que pueden formar el cuerpo del avatar.
- Calling cards: tarjetas para comunicarnos con nuestros amigos.
- Clothings: ropa y toda clase de complementos para vestir al avatar.
- Gestures: animaciones para aumentar la comunicación gestual.
- Landmarks: referencias a puntos del mundo virtual que deseamos guardar.
- Notecards: notas que realizamos como utilidad sobre avatares, sitios, objetos,…
- Objects: compuestos por geometría, textura, contenidos como scripts y otros.
- Photo Album: colecciones de fotografías.
- Scripts: pequeños trocos de código ejecutables aplicables a los objetos y avatares.
- Sounds: sonidos que aumentan la realidad virtual y son usados en interacciones.

◆ Textures: mapas de imágenes para crear superficies realistas sobre los objetos.

El inventario es algo imprescindible para poder permanecer en Second Life. Es lo que le proporciona un soporte para una de sus principales características: la posesión de entidades. Sin el inventario, no podríamos tener objetos propios, con lo que la economía en Second Life no podría existir. Además proporciona la forma de guardar objetos para aquellos habitantes que no poseen tierra propia pero sí posesiones de otro tipo como fotos, objetos, relaciones,…

El inventario se usa generalmente mediante un menú muy simple que presenta los siguientes submenús: archivo, crear y ordenar, tal como muestra la siguiente imagen.

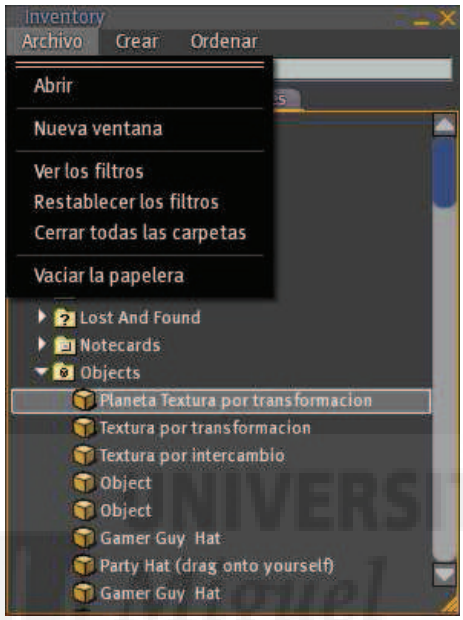


Imagen 6.2.23: Gestión del Inventario

Se pueden crear almacenes e introducir dentro de ellos todo tipo de artículos en una “prim”. Es decir, en un objeto. Esto tiene la ventaja de que podemos crear almacenes con todas las propiedades que tiene un objeto como posición, color, etc. Pero tiene un inconveniente y es que no podemos ver lo que hay dentro de él hasta que no se abre y además no permite crear subcarpetas. Para realizarlo solo hace falta crear un prim y arrastrar desde el Inventario los artículos deseados. Además lo podemos declarar como almacén público o privado.

Para mantener la gestión del Inventario debemos vaciar frecuentemente la papelera, además de tener una organización eficiente mediante el uso de subcarpetas y por supuesto, realizar periódicamente copias del Inventario.

El Inventario, por lo tanto, es una base de datos importantísima para nuestra vida en Second Life y se da la circunstancia que se mantiene en servidores de Linden Labs. A veces, pueden desaparecer objetos o que cambien de nombre y otros problemas por lo que conviene hacer una copia de seguridad de nuestro Inventario.

Realizar una copia de nuestro Inventario de Second Life es bastante sencillo en un sistema Windows. En otros sistemas esto se complica. El inventario se guarda en la carpeta c:\Documents and Settings\[usuario]\Datos de programa\SecondLife\cache y tiene la extensión .INV. Tendremos un archivo de este tipo por cada avatar que tengamos en Second Life. Este archivo es un .zip. Por lo tanto, para tener una copia de seguridad de nuestro Inventario basta con hacer una copia de este archivo en otra carpeta o mejor en una unidad de almacenamiento externo.

Si de todas formas perdemos objetos que hayamos comprado, podemos pedirlos nuevamente al vendedor si probamos que hemos pagado por ellos. Por eso es bueno acudir a una historia de transacciones realizadas en Second Life.

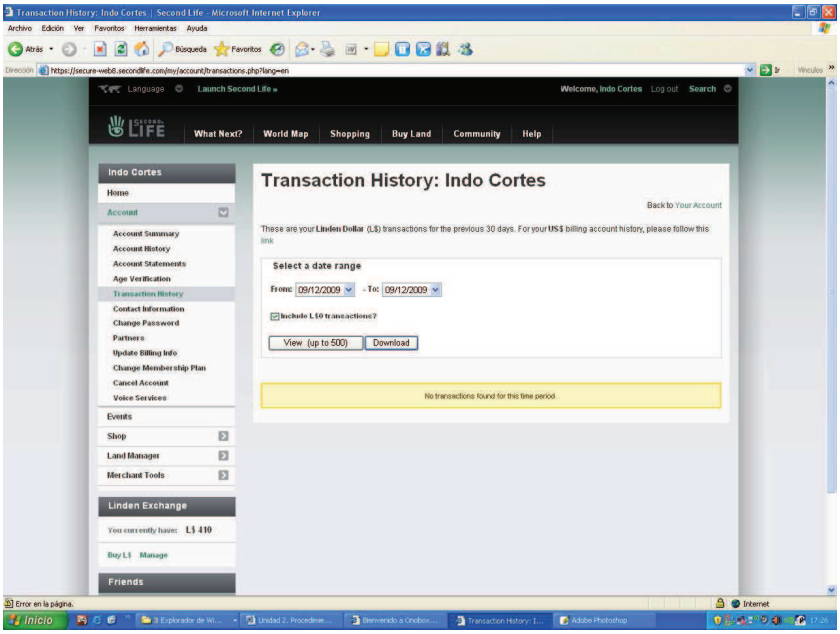


Imagen 6.2.24: Exposición de un objeto del Inventario

Este archivo se puede bajar directamente en formato Excel desde la dirección http://www.secondlife.com/account/transactions.php dónde nos pedirán el nombre, apellidos y la clave como muestra la anterior imagen.

También existen programas de terceros que permiten gestionar el Inventario de forma visual y algún cliente de Second Life distinto del oficial que mejora la gestión y transferencia de los objetos como AjaxLife.

Existe una utilidad dónde podemos guardar entidades que se llama biblioteca pero a diferencia del inventario, es pública, por lo tanto el contenido de la biblioteca es accesible por cualquier avatar. Además, la biblioteca no puede ser modificada para borrar o modificar artículos, mientras que en el inventario si.

7. Cómo exponer nuestro trabajo off line: Vídeo y Foto.

En este punto lo que queremos estudiar es la forma de mostrar nuestro trabajo a espectadores que no tienen cuenta en Second Life. Es evidente que los medios no serán interactivos ya que esta característica es propia de Second Life. Pero podemos ver la manera de mostrar cómo hemos trabajado para realizar una obra o la obra en sí misma y en el caso de que sea interactiva, la manera en cómo lo hace. Para ello podemos crear una serie de instantáneas o realizar una captura en vídeo.

Para hacer una instantánea tenemos un método general de Windows que es pulsar la tecla PrtScr (o ImpPnt) lo que se conoce como “pantallazo” y luego hacer un copiar con Ctrl+C en algún programa de edición de imágenes como puede ser PhotoShop. Por ejemplo, esta es la forma en que se ha tomado todas las ilustraciones de este documento.

Second Life proporciona su propia toma de imágenes mediante el comando Archivo > Hacer una foto (o pulsando el atajo de teclado Ctrl + Shift + S) que presenta la caja de diálogo que se muestra en la siguiente imagen.

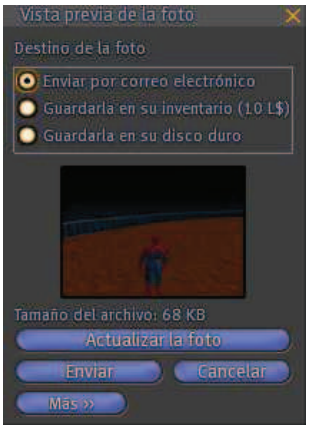


Imagen 6.2.25: Hacer una foto

Para capturar nuestros movimientos en Second Life y poder exponerlos en forma de vídeo como en www.Youtube.com utilizaremos software de terceros. La propia compañía Linden Labs. recomienda el uso de alguno de estos programas entre los que destaca Fraps (http://www.fraps.com/) y Camtasia (http://www.techsmith.com/camtasia.asp). En versiones anteriores de Second Life suministraba una utilidad para capturar vídeo, pero superada por otros programas como los nombrados, optaron por no incluirla en las nuevas versiones.

8. Permisos en Second Life: intelectuales, de suelo y otros.

Este es un tema fundamental en Second Life, y en cierta manera lo que le distingue de otros metaversos. En Second Life se prima el trabajo de los autores proporcionando una serie de utilidades para mantener la propiedad intelectual de los usuarios que creen el metaverso desde distintas visiones creativas.

El mecanismo es sencillo. Cada objeto contiene información de su creador y una serie de permisos para distintas acciones. Cabe recordar que en Second Life, los objetos pueden contener otros objetos como scripts, texturas, y otros.

Antes de ver los permisos tenemos que decir que se pueden regalar copias de los objetos propios a otras personas y para ello solo tenemos que arrastrarlos desde el inventario hasta el avatar al que se lo quieras regalar (y por supuesto, que él lo acepte).

Existen cuatro permisos básicos: "mover", "modificar", "copiar" y "transferir". Todos ellos pueden estar aplicados a cuatro categorías: "dueño", "grupo", "todos los demás" y "siguiente dueño".

Vamos a describir las características de los permisos:

- Modificar.

Si está habilitado, podemos editar propiedades como el nombre, la escala, la textura, etc. También podemos borrar el objeto. Si está compartido con el grupo, cualquier miembro del grupo lo puede modificar. Cuando un objeto es "no modificable para el siguiente dueño", no podrá hacer objetos derivados de este. Este permiso no es jerárquico hacia arriba pero sí hacia abajo, es decir, si un objeto modificable contiene otros objetos no modificables, el objeto contenedor podrá ser modificable y por tanto se podrá borrar, pero si no es modificable sus objetos contenidos tampoco lo podrán ser.

- Copiar.

Si está habilitado para un objeto, lo podemos copiar tantas veces como queramos. Los objetos copiados mantienen la información del creador y nunca pueden tener mayor permisos que el objeto padre. Si un objeto tiene objetos contenidos y uno de ellos no se puede copiar el objeto contenedor tampoco se podrá copiar. Si un objeto no se puede copiar, no quiere decir que sus contenidos no se puedan copiar.

- Transferir.

Si transferir está habilitado, el dueño puede donar el objeto a quien le plazca, pero si no está habilitado no puede venderlo, regalarlo o introducirlo en ningún objeto para regalarlo después. Cuando un objeto se transfiere es porque el dueño inicia este proceso y los permisos al "siguiente dueño" se copian al campo "dueño" del objeto copiado.

- Mover.

Si está habilitado, el objeto podrá cambiar de posición, es decir, podrá ser movido.

En cuanto a las categorías, cada una tiene sus propios permisos. Los permisos para "grupo" y "siguiente dueño" nunca pueden ser más permisivos que los permisos del dueño actual. Los permisos para "todos los demás" nunca pueden incluir "modificable" y nunca pueden ser más permisivos que los permisos para "grupo". Los objetos que pertenecen a un grupo, igualan los permisos para las categorías de "dueño" y "grupo".

Por ejemplo, vamos a estudiar distintas situaciones y cómo serían los permisos:

Si queremos vender un arma y que el cliente lo pueda modificar y hacer copias pero no lo pueda vender, entonces los permisos tendremos que ponerlos en:

Si	Siguiente dueño	Modificar
Si	Siguiente dueño	Copiar
No	Siguiente dueño	Transferir

Si queremos vender un objeto con un script contenido, para que pueda modificar el color del objeto pero no ver ni modificar el script, podremos los permisos como sigue:

OBJETO:

SÍ	Siguiente dueño	Modificar
No	Siguiente dueño	Copiar
Sí	Siguiente dueño	Transferir.

SCRIPT:

No	Siguiente dueño	Modificar
No	Siguiente dueño	Copiar
Sí	Siguiente dueño	Transferir.

9. Comprar objetos.

Si queremos ir de tiendas lo más rápido es usar el control de búsqueda que está situado arriba a la izquierda y escribir por ejemplo "Shopping clothes", entonces aparecerá una diálogo con direcciones a las que ir, tal como muestra la imagen inferior. Aquí podemos elegir la que queramos y pulsar a "Teleport", con lo cual nos teleportaremos y podremos recorrer las tiendas.

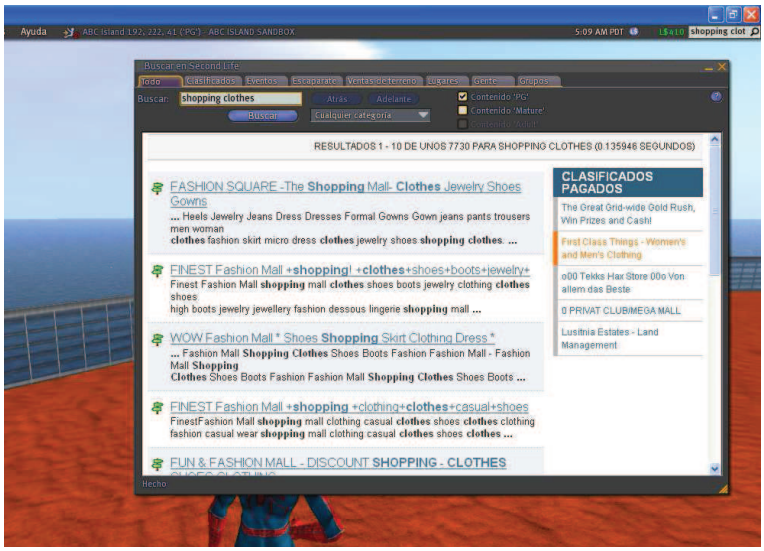


Imagen 6.2.26: Buscar tiendas

Cuando compramos algo, aparece en el inventario, normalmente en “recent ítems”. Cada objeto puede comprarse de una forma distinta, ya que este procedimiento depende del dueño aunque normalmente los objetos en venta vienen en una caja. Entonces hacemos clic con el botón derecho y elegiremos “abrir”. Esta acción abrirá una lista dónde elegimos copiar al inventario.

Imagen 6.2.27: Tienda de calzado en un centro comercial

10. Comprar tierras.

En lo que se refiere a la compra de terrenos, tenemos que empezar diciendo que es una cuestión importante, ya que tendremos que realizar una inversión relativamente importante en dinero “de verdad”. Al comprar terreno, estamos, en realidad comprando una porción de espacio en los servidores de Linden Labs. Y en este espacio podemos construir objetos y escribir scripts de la misma manera que otros publican sitios webs.

En este documento, pretendemos crear obras virtuales en tres dimensiones y que esto lo puedan realizar estudiantes y autores de bellas artes. Esta premisa hace que la inversión de una cantidad considerable de dinero no se contemple, aunque de todas formas vamos a describir algunos hechos sobre cómo adquirir terrenos.

Para comprar terreno tenemos que comprar parcelas de unos determinados metros cuadrados, no podemos elegir exactamente los metros a comprar. La parcela mínima es de 512 m^2, aunque esto puede variar en cualquier momento.

La política divide el territorio de Second Life en dos tipos: Mainland y State. En los terrenos Mainland o continentales, tenemos que tener una cuenta Premium y cada propietario hace lo que quiere con su terreno, por ejemplo, puedes construir tu casa y tener al lado un trozo de selva o un prostíbulo. En State, es decir, en islas privadas, se deben seguir unas reglas que para cada lugar serán distintas y que hace que sean más homogéneas, este es el modo que utilizan los lugares comerciales en general, que además, como en la vida real, tienden a concentrarse.

En cuanto al modo de adquirir un terreno también existen dos modalidades: la venta directa y la subasta. Existe el comentario más o menos extendido que en subasta, se pueden conseguir mejores terrenos a menor precio, pero también en la vida real depende de muchas circunstancias y los chollos no abundan. Las subastas de terreno se realizan vía web en la dirección http://secondlife.com/auctions.

La tierra tiene un coste que se abona cuando pasa a ser del propietario que para el continente es Linden Labs. y a esto hay que sumar el coste del mantenimiento.

El mantenimiento difiere si la tierra es continental o no y otros factores. En caso de ser en una isla privada, el propietario puede pedir lo que quiera de mantenimiento. En el continente, al tener una cuenta Premium el mantenimiento para una parcela de 512 m^2 es 0, pero a partir de aquí crece según la cantidad de metros poseída y varía con el mercado.

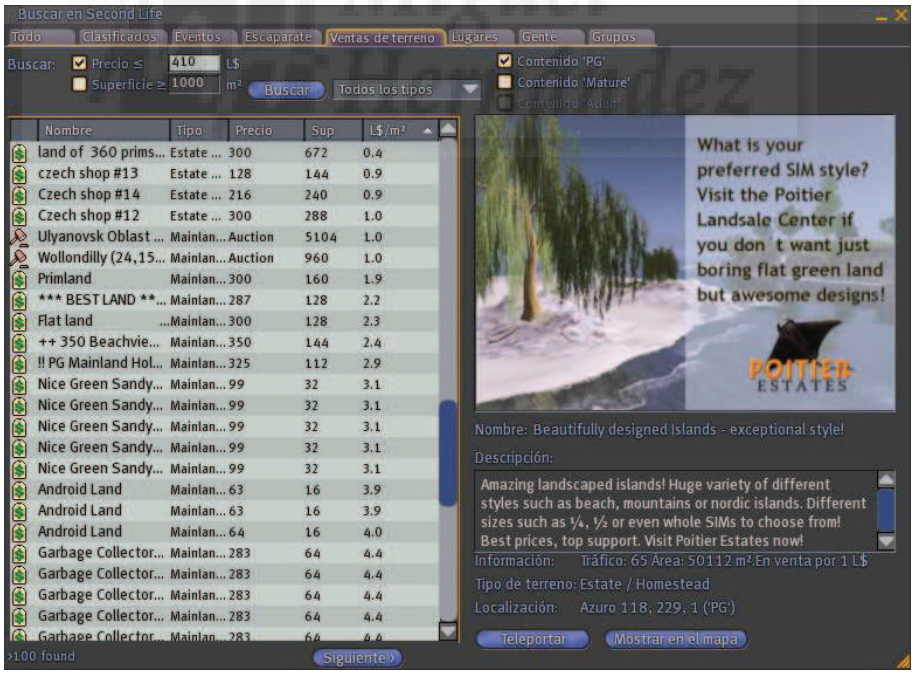


Imagen 6.2.28: Búsqueda de terreno en venta

También podemos comprar una isla entera cuyo tamaño típico es de 65.536 m^2, es decir, todo un sim (simulador completo). Estas islas tienen un mantenimiento directamente con Linden Labs de 295 $ USA al mes y 195 $ USA al mes para organizaciones educativas o sin fines de lucro.

Para buscar tierra en venta, vamos a la herramienta de búsqueda y pulsamos la pestaña “Ventas de terrenos”. En la imagen anterior podemos visualizar el resultado de buscar terrenos a un precio inferior a 410 L$ que es lo que me queda en mi cuenta y que en el momento de la redacción de este texto (septiembre de 2009) equivalen a unos 2 $ USA.

Para comprar tierra tenemos que teletransportarnos a ella y con el botón derecho pulsar sobre el terreno, luego pulsamos sobre “Acerca del terreno” y finalmente sobre “Comprar terreno”.

En la siguiente imagen se puede observar una subasta de terrenos de Second Life por Internet.

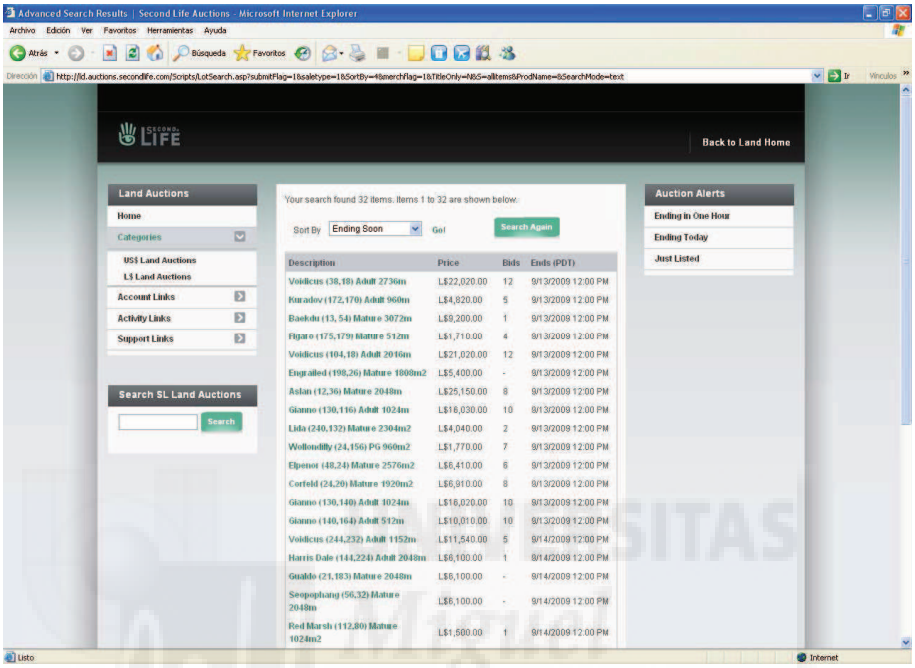


Imagen 6.2.29: Subasta de terreno en venta

11. Resumen de atajos de teclado para distintas funciones.

Estos que siguen son solo algunos de los atajos de teclado de los muchos que vienen definidos en Second Life:

Atajo del teclado	Función
General	
Ctrl + P	Abre Preferencias
Ctrl + Shift + S	Tomar una foto
Ctrl + Shift + D	Mostrar el menú avanzado
Ctrl + I	Abrir el Inventario
Ctrl+3	Abre la herramienta de construcción en modo edición.
B o Ctrl+4	Abre la herramienta de construcción para crear un prim.
Ctrl + Q	Salir de Second Life
Desplazamientos	
Flechas	Dirección
Ctrl + Shift + H	Te teleporta a casa
Inicio	Comenzar/terminar de volar
Repag	Volar más alto
Avpag	Volar más bajo
Ctrl + R	Correr
E	Saltar
Ctrl + M	Abrir el Mapa
Ctrl + Shift + M	Abrir el Minimapa
Vistas	
Ctrl+Shift+Y	Pones el sol al mediodía
Alt+Enter	Poner a pantalla completa
Ctrl+Alt+T	Resalta lo transparente
Ctrl+Alt+R	Recarga las texturas sobre nuestro avatar
Escape	Pone la cámara (o zoom) a su posición original
M	Vista de ratón, miramos con los ojos del avatar
Alt + ratón	Zoom a objetos concretos
Comunicación	
Ctrl+T	Abre los IMs/MIs
Ctrl+Shift+F	Abre la ventana de amigos y grupos.
Ctrl+F	Abre el buscador.
Ctrl+H	Abre el historial del chat de texto

Unidad 3: Modelado de objetos.

Introducción teórica:

1. El modelador de Second Life.
2. Limitaciones de los objetos en Second Life.
3. Qué son y cómo crear prims.
4. Operaciones básicas con objetos.
5. Sistemas de coordenadas.
6. Qué son y cómo crear objetos enlazados.
7. Qué son los modificadores y clasificación.
8. Modificadores.
    - 8.1. Modificadores de la primitiva caja.
    - 8.2. Modificadores de la primitiva esfera.
    - 8.3. Modificadores de la primitiva toro.
9. Qué son y cómo crear los sculpted prims.
10. Otras características no modelables de los objetos.
11. Sitios para aprender más.

Casos prácticos:

Caso práctico 3.1: Crear objetos prim.

Caso práctico 3.2: Operaciones básicas con objetos.

Caso práctico 3.3: Objetos enlazados.

Caso práctico 3.4: Modelar objetos con modificadores de cajas: pebetero

Caso práctico 3.5: Modelar objetos con modificadores de esferas: escarabajo.

Caso práctico 3.6: Modelar objetos con modificadores de toros: ánfora

Caso práctico 3.7: Realizar sculps

1. El modelador de Second Life.

Second Life es un simulador de un mundo virtual. En este complejo mundo tenemos entidades visuales y no visuales. Entre las no visuales están los scripts y toda la programación que subyace y que por supuesto, puede tener unas consecuencias visuales.

En SL también tenemos tres entidades visuales fundamentales: tierra, avatares y todo lo demás. Ese "todo lo demás" lo han construido los propios usuarios. Por lo tanto, La construcción del mundo virtual Second Life recae en su mayor parte en los propios usuarios. Estamos incluyendo toda clase de edificios, artilugios, vehículos, animales, partículas, etc.

La manera en que podemos modelar objetos tridimensionales en Second Life es ciertamente original. Sobretodo si lo comparamos con otros programas de creación 3D.

Existen en el mercado programas como MAX, Maya, Softimage, Cinema 4D, etc. que también permiten modelar. Evidentemente todos tratan de encontrar un nicho en un mercado muy competitivo, pero al fin y a la postre se obtienen los mismos resultados. Por ejemplo, dos de los programas más conocidos como son MAX y Maya son de la misma empresa: Autodesk, y aunque MAX es más generalista y Maya se utiliza mucho en la industria cinematográfica y en la creación de personajes, en el fondo, lo que se obtiene es lo mismo: una serie de vértices que componen polígonos, que a su vez se estructuran en mallas de puntos 3D. También pueden tener la posibilidad de crear modelos 3D de muchas otras formas. Esto en cuanto al modelado 3D. A parte, también utilizan módulos de iluminación, animación, representación, etc.

En Second Life todo es diferente. Aquí aunque hay vértices y polígonos, no podemos manipularlos independientemente. Por lo tanto la mayoría de las formas de creación 3D clásicas como los objetos loft, edición de mallas, splines, etc. no se pueden realizar en SL.

Estas diferencias tienen sus ventajas y desventajas. Por una parte, un modelador 3D avanzado necesita "reaprender" para realizar el mismo tipo de trabajo. Esto no siempre es fácil. Por otra parte, para usuarios no avanzados puede ser algo más accesible, más visual y más inmediato.

De todas formas, con los primitivos de tipo scultped, se abre un camino para importar objetos creados en aplicaciones como MAX y otras aplicaciones del mismo tipo a Second Life.

2. Limitaciones de los objetos en Second Life.

Los objetos complejos se forman mediante objetos geométricos básicos llamados prims. Los prims que podemos crear en Second Life tienen límites. El límite inferior es de 0.010 metros. Es decir, en principio no podemos realizar una caja (un cubo) menor de 1 centímetro de lado, ni una esfera menor de 1 cm de diámetro, y así con el resto de primitivos. El límite superior se encuentra en los 10 metros. Por lo tanto "solo" podemos hacer directamente cajas de 10 x 10 x 10 metros.

Estos límites pueden ser superados con distintas técnicas como pueden ser el uso de texturas especialmente preparadas, el uso de megaprims con algunas limitaciones de uso, etc.

También existen límites en cuanto al número de prims que un usuario puede crear. Por supuesto que si estamos en un sandbox, el propietario de este terreno puede poner cualquier tipo de restricción, en tiempo, en número y tamaño. Pero aunque seamos propietarios de tierra, existe un número máximo de prims que podemos crear. Esto es lógico si lo comparamos con la web actual. Cuando "compramos" cierta parte de un servidor para alojar nuestra web lo hacemos con restricciones de tamaño (x gigas, o lo que sea) por lo tanto, cuando tenemos tierra en realidad también estamos haciendo uso masivo de memoria, estructuras de datos, y otros recursos que están siendo almacenados por una empresa privada, que pone límites.

Por ejemplo, para un terreno de 4096 m^2, tenemos un límite construcción de 1256 prims. Por supuesto, que esto depende de Linden Labs. y se puede modificar en un futuro.

3. Qué son y cómo crear Prims.

Todo lo que se visualiza en Second Life que no sean islas y humanos, todo lo demás, ha sido construido. Todas estas "construcciones" se han realizado a partir de formas en tres dimensiones llamadas prims.

La palabra prims deriva del término primitivo (primitive en inglés). Indica formas en 3D fundamentales para la construcción en Second Life. Por ejemplo un prim es un cubo o una esfera. Son como las piezas de un "tente". En SL tenemos 8 piezas fundamentales o primitivas. Dicho de otro modo: todo lo que vemos en SL está construido a partir de esas 8 piezas. Estas piezas básicas son: caja, cilindro, prisma, esfera, toro, tubo, anillo y sculpted. Todos salvo el sculpted son piezas bien conocidas en la geometría básica. En la figura que sigue se puede ver un ejemplo de los prims:

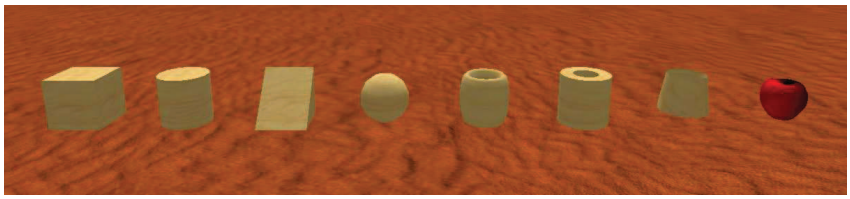


Imagen 6.3.1: Las ocho piezas que componen las construcciones en Second Life

Para crear un objeto debemos ser propietarios de tierra o buscar un sandbox. Una vez resuelto este tema, solo tenemos que hacer clic con el botón izquierdo sobre el terreno y aparecerá un menú contextual de forma circular.

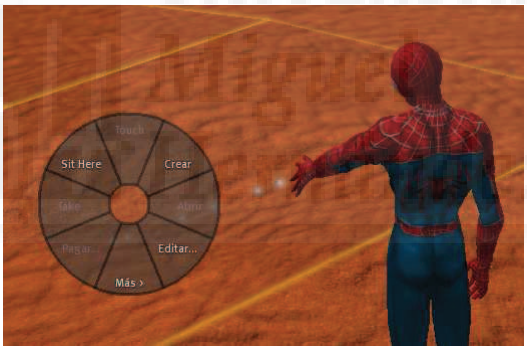


Imagen 6.3.2: Crear objetos

Entre las opciones que presenta debemos pulsar en Crear (o en el botón "Build" en el centro de la pantalla) y esto nos llevará a una ventana de edición de modelos 3D. Esta caja permite manipular al modelador 3D que Second Life tiene embebido en su código y que es accesible en todo el simulador. En esta primera ventana tenemos 5 botones para acceder a: Focus, Mover, Editar, Crear y Tierra. Para crear objetos básicos utilizaremos fundamentalmente los paneles de Editar y Crear, pero los explicaremos todos para tener una idea mayor de las posibilidades que tenemos para trabajar.

Focus (Enfocar): tenemos la posibilidad de utilizar zoom, órbita y enfocar. Esta última opción es muy útil cuando creamos objetos por lo que debemos dominar esta herramienta. Se realiza con las teclas Shift + Alt y con el ratón podemos hacer clic y arrastrar para acercarnos o alejarnos del objeto que queramos. También la podemos utilizar para visualizar la situación del avatar como si fuera una cámara independiente.

Imagen 6.3.3: Panel Focus

Move (Mover): podemos mover, trasladar, e incluso girar de forma interactiva sin la referencia de las coordenadas. No se utiliza mucho ya que para ello tenemos el panel de editar que es mucho más preciso.

Imagen 6.3.4: Panel Mover

Edit (Editar): en este panel podemos cambiar de posición, girar y escalar los objetos. Además podemos elegir las coordenadas a utilizar para estas operaciones y otras funciones que aparecen cuando pulsamos el botón de “Más”.

Imagen 6.3.5: Panel Editar

Create (Crear): aquí podemos elegir el objeto base a crear. Tenemos la opción de elegir entre 15 objetos que se utilizan muy a menudo. También tenemos “Mantener herramienta seleccionada”, que puede ser útil para crear una serie de objetos del mismo tipo.

Imagen 6.3.6: Panel Crear

Land (tierra): este panel solo es útil si poseemos nuestra propia tierra. Cómo se puede observar en la imagen, sirve para crear la forma que tendrá el terreno, sus elevaciones y pequeñas depresiones, etc.

Imagen 6.3.7: Panel Tierra

Cómo hemos visto en el panel Crear, se presentan 15 figuras de las que 7 son primitivos (falta el scultped) y las 8 restantes son prims modificados. Los modelos 3D que podemos crear directamente son:


Caja. Es un primitivo. Representa un cubo.


Prisma triangular. Es un primitivo.


Pirámide cuadrangular. No primitivo. Se puede crear con una caja y el mod. Taper.


Pirámide triangular. No primitivo. Se puede crear con un prisma y el mod. Taper.


Cilindro. Es un primitivo.


Medio cilindro. No primitivo. Se puede crear con un cilindro y el modificador Path Cut.


Cono. No primitivo. Se puede crear a partir de un cilindro con el modificador Taper.


Medio cono. Se puede crear a partir de un cono con el modificador Path Cut.


Esfera. Es un primitivo.


Hemisferio. Se puede crear a partir de un cono con el modificador Path Cut.


Toro. Es un primitivo.


Tubo. Es un primitivo.


Anillo. Es un primitivo.


Árboles. No se pueden crear salvo en tierra propia.


Hierba. No se pueden crear salvo en tierra propia.

Debemos pulsar sobre el botón “Más” para ver las pestañas que permiten acceder a todas las funcionalidades del modelador. Esta extensión de la ventana y las pestañas que presenta es idéntica para todos los paneles salvo para el de tierra y son las siguientes:

General (General): esta pestaña permite acceder a datos como el nombre, descripción, propietario, permisos, y otros parámetros que afecten al objeto de manera propia.

Object (Objeto): en esta pestaña encontraremos los parámetros básicos para la edición del objeto como son: posición, dimensiones, rotación, escala y todos los modificadores que podamos aplicar. Es por ello que la mayoría del trabajo se realiza en esta pestaña.

Features (Características): aquí encontraremos parámetros que afectan a su flexibilidad y luminosidad.

Texture (Textura): esta pestaña contiene datos que afectan a la visualización del objeto. Alguno de estos son el color, la textura 2D aplicada, efectos de relieve, autoiluminación, opacidad, etc.

Content (Contenido): permite introducir tres entidades fundamentales: scripts y texturas y sonidos. Esto significa que podemos programar los objetos independientemente si los dotamos de pequeños programas. Por otra parte, también podemos guardar en el propio objeto las texturas y sonidos que utilicemos para dotar de multimedia al objeto.

- **Pestaña General**

En esta pestaña tenemos controles que permiten editar propiedades que afectan al objeto como una entidad única e independiente. Estos son los siguientes:

Nombre: permite identificar fácilmente el objeto. Esto es muy importante cuando tenemos muchos y sirve también para buscarlos y gestionarlos de manera óptima en el inventario.

Descripción: este campo lo utilizaremos para introducir una pequeña frase que permita saber su función (sobre todo si forma parte de una estructura) y otros datos aclaratorios.

Creador, propietario y grupo: aquí podemos introducir estos datos que serán fundamentales si pretendemos integrarnos en la economía de SL, ya que se basa en la posesión de objetos. Es evidente que el creador del objeto y el propietario no tienen por qué ser el mismo usuario. Un objeto, además, puede ser compartido por un grupo.

Permisos: para compartir con un grupo, para que sea movido, copiado, buscado y además que pueda ser comprado. Para ello le ponemos un precio.

Venta: una vez que el objeto tiene precio, podemos editar los derechos que tendrá el comprador, si es un original o no y si puede modificarlo o hacer más copias futuras.

Interacción al hacer clic: este control, interpreta cómo debe responder el objeto cuando un usuario hace clic con el botón izquierdo del ratón sobre él. Por defecto, debe ser interpretado como un evento de “Tocar”. Este evento puede ser detectado por un programa y obrar en consecuencia, por ejemplo, saludando. Pero existen más opciones que aparecen en la lista desplegable, por ejemplo, puede ser que al hacer clic sobre el objeto no sentemos encima de él, o si es un panel, que nos dé información, etc.

- **Pestaña Objeto**

Esta pestaña permite editar el objeto, cambiar su geometría y otras propiedades “físicas”. Para ello dispone de 5 grupos de parámetros que se puede ver en la imagen de abajo: propiedades de edición (en rojo), bloque de construcción (en verde), propiedades básicas (en amarillo), modificadores (en azul) que dependen de cada tipo de bloque de construcción elegido y material (en rosa).

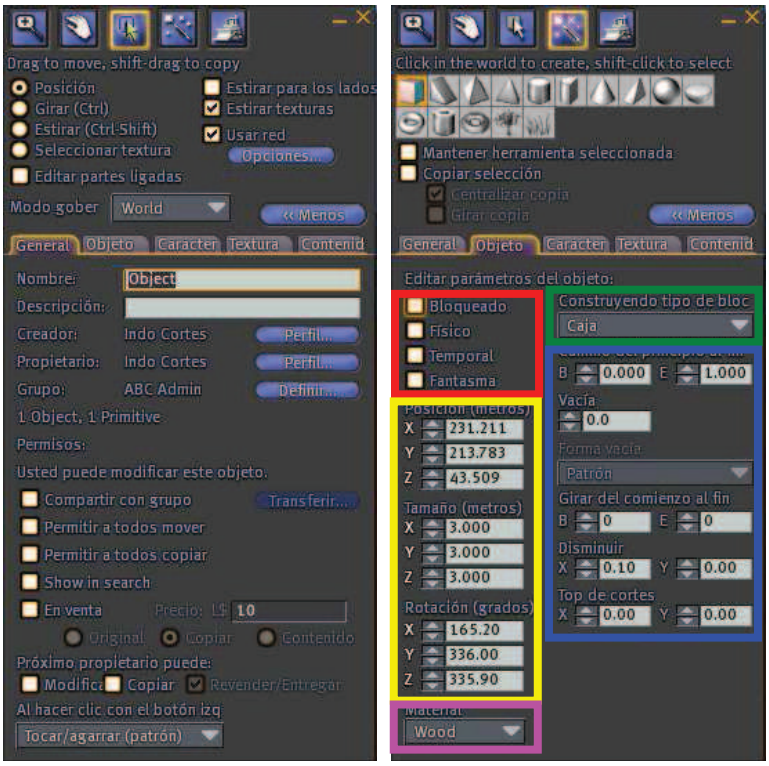


Imagen 6.3.8: Panel General y panel Objeto

Propiedades de edición: son una serie de características importantes para su edición.

Bloque de construcción: permite modificar el prim del que parte el objeto.

Propiedades básicas: su posición, tamaño y rotación.

Modificadores: las operaciones geométricas que convierten el prim en otros objetos.

Material: si es físico, el tipo de material define como rebotará, caerá, etc.

Vamos a estudiar uno por uno estos importantes conceptos, ya que la mayor parte del trabajo de modelado 3D se realizará en esta pestaña. Es como el taller de herramientas de Second Life y tenemos que conocer todos sus entresijos.

Propiedades de edición:

Bloqueado: si está on, el objeto no se podrá editar para evitar modificaciones accidentales. Si hemos terminado de realizar una parte de un objeto complejo y no debemos apartarlo porque quizá nos sirva de referencia para seguir trabajando, por lo menos lo podemos poner como bloqueado.

Físico: esto significa que el objeto se verá sujeto a las leyes físicas como la gravedad, podrá ser empujado, arrastrado, etc. o también que no lo traspasemos, sino que chocaremos con él, lo podremos dejar caer, por lo tanto será como un objeto sólido con cierto peso que dependerá de sus dimensiones y material.

Temporal: esta propiedad se utiliza para hacer pruebas, ya que será automáticamente borrado tras 1 minuto. Se utiliza para ahorrarnos el trabajo de borrarlos.

Fantasma: el objeto se podrá traspasar por otros objetos y avatares. Es lo contrario a físico en este sentido. Se puede utilizar para objeto que queramos que sean visibles pero que permitan la movilidad. Por ejemplo, podemos poner en lugar de una puerta, una cortina, o una cascada, que se verá pero que podremos traspasar.

Bloque de construcción:

Desde esta lista desplegable podremos cambiar en cualquier momento el primitivo o prim que será el objeto 3D básico. Es fundamental tener claro el prim del que debemos partir para modelar el objeto que queremos, ya que el resultado de los modificadores aplicados a distintos prims tienen, por supuesto, diferentes resultados. Pero como hemos dicho antes, si queremos ver cómo quedaría un modificador en otro prim lo podemos elegir desde aquí. Tenemos disponibles los 8 prims de que dispone SL: Caja, Cilindro, Prisma, Esfera, Toro, Tubo, Anillo y Sculpted. Al elegir prims distintos podremos observar cómo se modifican los modificadores disponibles, ya que son distintos para cada prim.

Propiedades básicas:

En este grupo tenemos los valores para la posición, tamaño y giro. Vamos a estudiar estos parámetros junto con las operaciones que permiten su modificación en el apartado siguiente.

Material:

Este concepto es importante porque en principio lo podemos confundir con la textura, ya que por defecto, al crear un objeto nuevo se le asigna una textura de madera y el material también es madera (wood). Pero no tiene nada que ver con esto. Como se ha escrito anteriormente, este material será el que aporte datos al simulador físico a la hora de visualizar efectos físicos en los objetos. Por ejemplo, si dejamos caer un objeto de plástico, no caerá, ni rebotará igual que si es de piedra. Los materiales disponibles son estos 7: piedra, metal, cristal, madera, carne, plástico y caucho.

Modificadores:

Hemos dejado este aspecto para el siguiente tema por que es el bastante extenso y requiere un enfoque didáctico eminentemente práctico.

4. Operaciones básicas con objetos.

Todos los objetos se pueden modificar en tres aspectos básicos: posición, escala y rotación. Estas modificaciones son tan básicas como universales, todos los programas 3D permiten estas operaciones en cada una de las tres dimensiones y son las operaciones de mover, escalar y rotar. Estas operaciones se pueden llevar a cabo tanto introduciendo lo valores en las casillas correspondientes como arrastrando con el ratón de forma interactiva.

Todas estas modificaciones necesitan un punto de origen de la operación. Por ejemplo, si rotamos una caja, no es lo mismo rotarla con respecto a su centro (o sea, que gire sobre sí misma) que con respecto a un punto situado fuera del objeto. En Second Life tenemos tres clases de coordenadas: mundo, local y referencia (o en inglés: world, local y reference). Para modificar el sistema de coordenadas de un objeto, debemos editar el objeto y pulsar sobre la pestaña General. Aparece una lista desplegable con las tres opciones. En las unidades prácticas realizaremos cambios de unos sistemas de coordenadas a otros.

Con la intención de facilitar la manipulación de prims, se utiliza un código de color para los ejes. De esta forma, los controles rojos modifican el prim en el eje X de Este a Oeste, si tienen color verde lo harán en el eje Y de Norte a Sur y si son azules en el eje Z, en altitud.

Operación de mover, cambio en las propiedades de Position (Posición): permite especificar la posición del centro del objeto mediante las coordenadas X, Y y Z con respecto a un origen de

coordenadas, que en principio son tipo mundo. Las unidades son los metros, y podemos ajustarla hasta las milésimas.

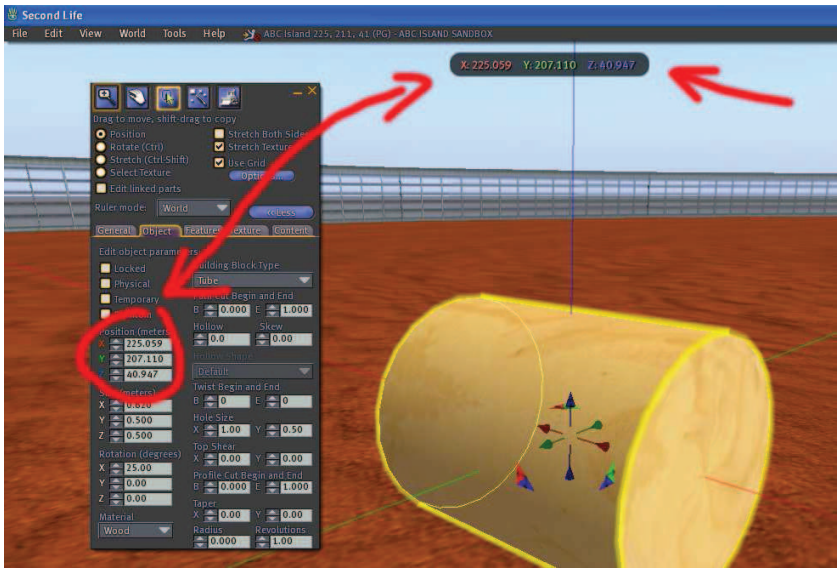


Imagen 6.3.9: Información sobre la posición de un objeto

La posición de un objeto es tan importante que en cuanto nos ponemos a editarlo, aparece su posición en la parte de arriba de la aplicación como se muestra en la siguiente figura. También podemos apreciar las casillas dónde podemos introducir directamente la posición a la que queremos trasladar el objeto.

La siguiente figura, muestra los controles de tipo flecha que podemos utilizar para mover interactivamente los objetos por el mundo 3D. Para ello basta con seleccionar una flecha (se mostrará resaltada) y arrastrar hasta donde deseemos dejarla.

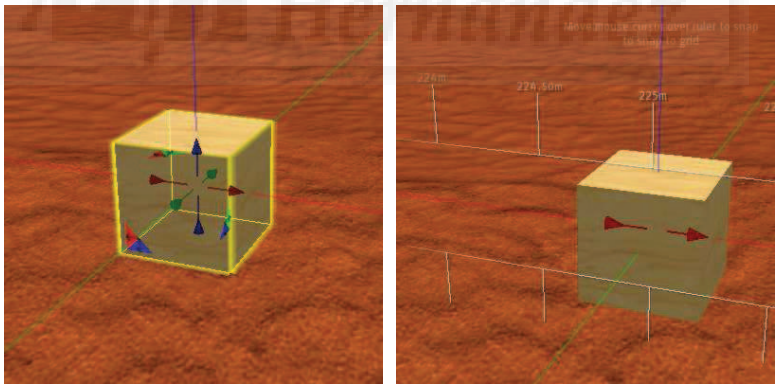


Imagen 6.3.10: Controles para mover el prim caja en los tres ejes

Operación de escalar, cambio en las propiedades de Scale (Tamaño): se mide en metros y su precisión llega hasta las milésimas. Modificar el tamaño desde aquí es mucho más preciso que realizarlo “a ojo”, y como suele ser una cuestión crítica, la mayoría de los modeladores utilizan estos controles a mano mucho menos que los de posición o giro.

Para realizar una operación de escalado interactivamente debemos de pulsar las teclas Shift + Control con lo que aparecen unos puntos en los colores de los ejes. Si queremos escalar de forma proporcional en las tres dimensiones lo haremos arrastrando los controles de color gris. En la imagen siguiente se puede observar una escala no proporcional en el eje Z.

También podemos escalar en un solo eje pero a ambos lados si tenemos pulsado el control “Strech both sides”, con lo que la caja de la figura se achatará por los dos lados.

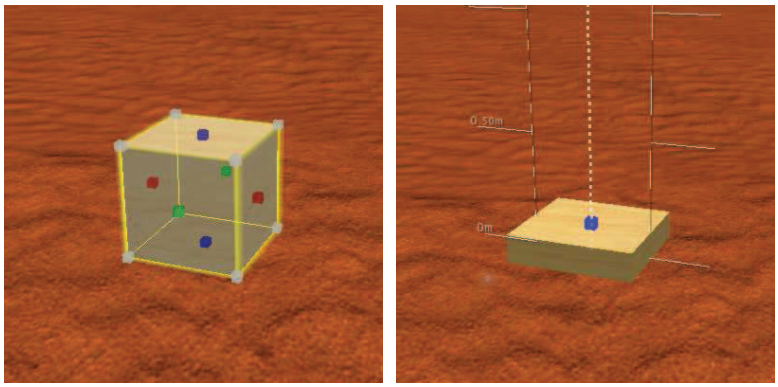


Imagen 6.3.11: Controles para escalar el prim caja en los tres ejes

La operación de escalar un objeto es fundamental para poder trabajar en un entorno virtual, ya que de esta manera se puede trabajar a un tamaño elegido por el usuario y luego se puede agrandar o empequeñecer tanto como se desee. Por ejemplo, supongamos que queremos realizar una hormiga. Podemos hacerla a tamaño “humano” (avatar) y una vez construida con el nivel de detalle que queramos, empequeñecerla hasta el tamaño real que tiene. Por supuesto, también podemos hacer lo contrario para realizar una escultura.

Operación de girar, cambio en las propiedades de Rotation (Rotación): se mide en grados y cómo vemos en la figura siguiente tiene una precisión de dos decimales que es suficiente para la mayoría de los trabajos.

La operación de girar objetos se puede realizar de forma interactiva arrastrando unos controles en forma de anillos que aparecen cuando pulsamos la tecla Control. En la imagen siguiente se puede apreciar un giro en el eje Y.

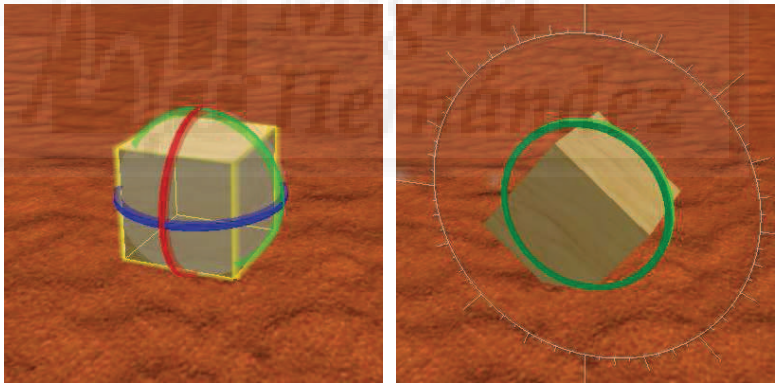


Imagen 6.3.12: Controles para girar el prim caja en los tres ejes

5. Sistemas de coordenadas.

Conocer este concepto es fundamental para el modelador en Second Life. El sistema de coordenadas será el origen de cualquier tipo de manipulación de un objeto 3D. Como escribíamos anteriormente, al girar un objeto, podemos hacerlo sobre sí mismo o con respecto a otro objeto, por lo que el resultado es totalmente distinto.

En Second Life tenemos tres clases de coordenadas: mundo, local y referencia (en inglés: world, local y reference). Para modificar el sistema de coordenadas de un objeto, debemos editar el objeto y pulsar sobre la pestaña General. Aparece una lista desplegable con las tres opciones.

El sistema de coordenadas siempre está compuesto por tres ejes, la diferencia es dónde está el origen de esos tres ejes. En el sistema de coordenadas mundo, el origen se encuentra en el

origen del sim. Un sim es normalmente una isla. El punto origen de la isla será el punto Sur-Oeste, a una altura de 20 metros. Las dimensiones típicas de una isla son de 256 x 256 metros, por lo que las posiciones de los objetos podrán tomar valores de 0 a 255 metros.

En realidad, las islas son espacios tridimensionales como se observa en la imagen siguiente. De hecho, se puede construir y "vivir en altura". El suelo base se encuentra a 20 metros del origen, por lo que tenemos un subsuelo, (que de momento no se utiliza) de 20 metros. Por encima del suelo podemos llegar a los 768 metros de altura. En este punto se plantea un problema y es que no podemos "volar" por nosotros mismos más allá de los 200 metros de altura, por lo que tendremos que usar vehículos y otros artefactos.

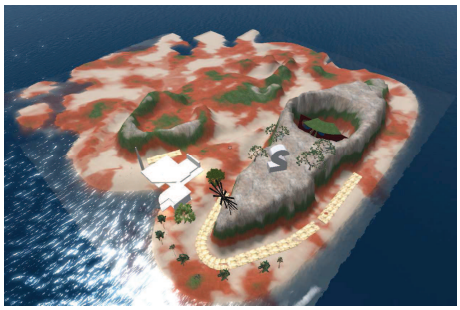


Imagen 6.3.13: Isla en Second Life

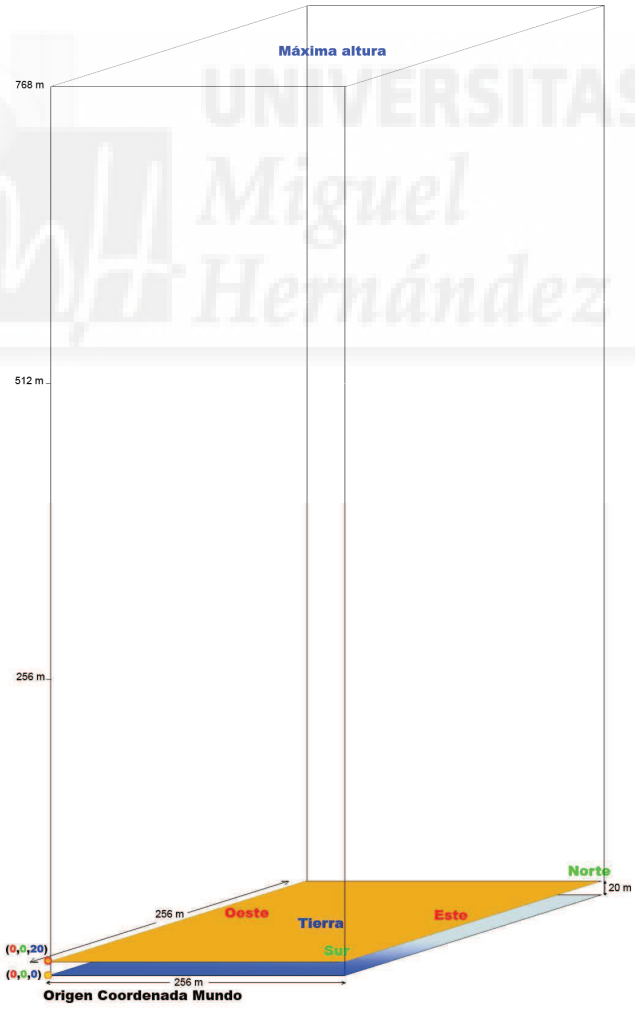


Imagen 6.3.14: Origen y dimensiones de las coordenadas mundo

6. Qué son y cómo crear objetos enlazados.

Los objetos complejos se realizan por dos vías: una es enlazando varios objetos para crear uno nuevo, por supuesto, más complejo que sus componentes, la otra es modificando los primitivos para obtener objetos nuevos derivados (modificados). Debido a la complejidad que algunas veces tiene el uso de modificadores, lo estudiaremos en un tema independiente.

Existen objetos muy complejos que estarán formados por una serie de prims que se enlazarán o agruparán para formar objetos más sofisticados. Por ejemplo una habitación estará formada por cuatro paredes, un techo y un suelo. Cada una de estos 6 elementos se pueden construir con un prim tan básico como una caja, pero una vez enlazados, deberían constituir el objeto "habitación" y este, ser tratado como un objeto nuevo pero con las características básicas de un objeto prim como son tener un nombre propio, un creador, una serie de permisos, poder rotarlo, escalarlo y posicionarlo, etc. Podemos pensar de esta forma estructurada para realizar estructuras complejísimas como por ejemplo un vehículo con sus subsistemas motor, habitáculo, etc.

En la imagen de abajo se puede observar una silla realizada con dos cajas y 6 cilindros que constituye un nuevo objeto en sí mismo. Se puede gestionar (guardar, mover, etc.) como una entidad u objeto individual.

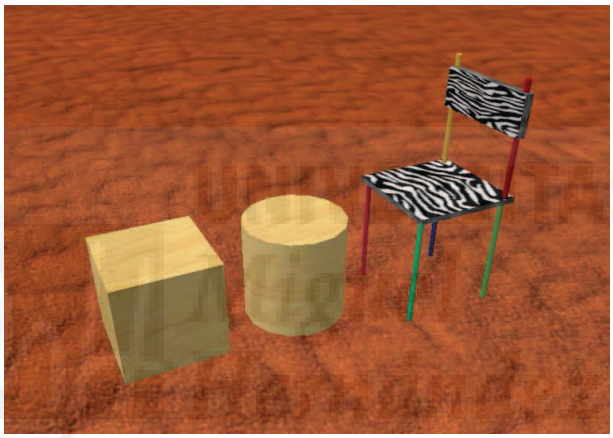


Imagen 6.3.15: Objeto compuesto a partir de dos primitivos

Enlazar y desenlazar objetos. En realidad la forma de crear objetos enlazados es muy sencilla. Se trata de seleccionar dos o más objetos y ejecutar el comando enlazar. Al enlazar varios objetos, habrá uno especial, llamado objeto raíz que será el último que hallamos seleccionado antes de enlazar. El objeto raíz se visualiza con un halo en amarillo, todos los demás en azul. En la imagen siguiente se puede observar que la pata trasera izquierda es el objeto raíz. Esta operación, es reversible, ya que ningún objeto pierde entidad individual, y por lo tanto se puede desenlazar.

Estos comandos están en el menú Tools (Herramientas) **>** Link (enlazar) o Unlink (desenlazar). Cuando se enlazan objetos, las operaciones de edición quedan restringidas para los objetos individuales, por ejemplo, no se pueden mover, cambiar de tamaño, rotar ni borrar de forma individual.

No se anidan los enlaces. Si tenemos dos grupos de objetos enlazados (por ejemplo si en la silla de arriba tenemos "asiento" con 5 objetos y "respaldo" con tres objetos), podemos enlazarlos en uno solo (silla), el objeto raíz será el objeto raíz del grupo seleccionado en último lugar. Si desenlazamos este grupo no obtendremos los grupos anteriores, sino los objetos iniciales, en el caso de la silla tendríamos 8 objetos. Por lo tanto, no se anidan las agrupaciones de objetos.

Desenlazar objetos individuales de un grupo. En la ventana Build (Crear)**,** activamos la casilla "edit linked parts y haciendo clic sobre el objeto lo desenlazamos del grupo.

Imagen 6.3.16: Objeto compuesto mediante enlaces

7. Qué son los modificadores y clasificación.

Anteriormente hemos establecido el concepto de objeto primitivo. Ahora estudiaremos cómo podemos modificarlos para tener elementos derivados de ellos.

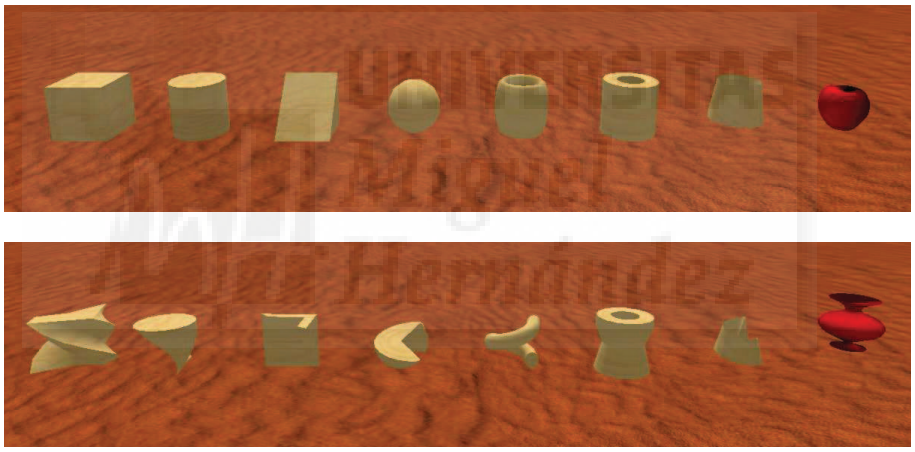


Imagen 6.3.17: Los ocho objetos primitivos y ocho objetos modificados

Los prims pueden ser modificados hasta el extremo en que no se reconoce su forma original. Esto implica que si queremos realizar ciertos diseños y alcanzar madurez como modeladores, debemos conocer las técnicas que permiten obtener estos objetos complicados a partir de los prims más sencillos. En la imagen anterior se puede observar los objetos primitivos y los mismos pero con un modificador aplicado. Imaginemos cómo podemos llegar a modificar un objeto si aplicamos una serie de modificadores en lugar de uno solo. Es evidente que son herramientas muy potentes que perece la pena conocer.

Los modificadores se encuentran en la pestaña Objeto que ya estudiamos en el tema anterior. Esta pestaña permite editar el objeto, cambiar su geometría y otras propiedades “físicas”. En la siguiente imagen se ven los modificadores disponibles para un prim de tipo caja. En realidad, el concepto de modificador puede ser difícil de entender de manera teórica. Se aprende mucho mejor practicando y además es lo que recomiendan los mejores modeladores de Second Life.

Los modificadores que se pueden aplicar a un objeto dependen del objeto primitivo del que parten o dicho de otra manera, del “bloque de construcción”. Tenemos que tener claro que los modificadores disponibles para los distintos primitivos no son los mismos, y además, que el mismo modificador con los mismos valores, afectan de distinta forma a los distintos prims, salvo

los objetos sculpted que no se ven afectados por modificador alguno. Por lo tanto, vamos a tratar de clasificar los modificadores según estos bloques.

Sigue una tabla que trata de aclarar un poco estas relaciones prim – modificadores. Aunque el resultado es un poco extraño, hemos puesto las primitivas en español y los modificadores en inglés por que para alguno de ellos, no existe una buena traducción a nuestro idioma y además casi toda la literatura que podemos encontrar sigue esta pauta.

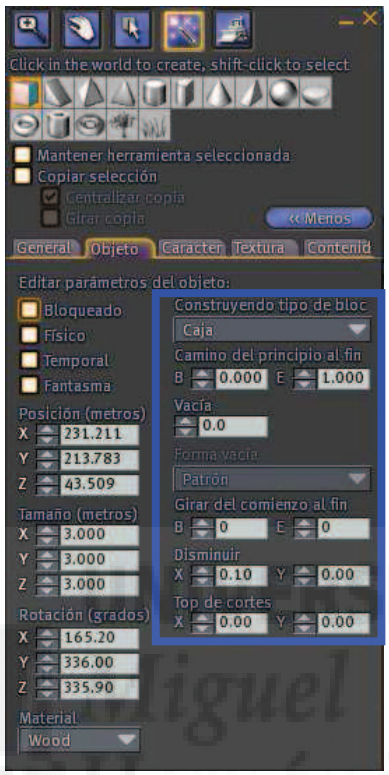


Imagen 6.3.18: Ficha Objeto

	Caja	Cilindro	Prisma	Esfera	Toro	Tubo	Anillo	Sculpted
Path Cut begin and end	X	X	X	X	X	X	X	
Hollow	X	X	X	X	X	X	X	
Hollow shape	X	X	X	X	X	X	X	
Twist begin and end	X	X	X	X	X	X	X	
Taper	X	X	X		X	X	X	
Top Shear	X	X	X		X	X	X	
Dimple begin and end				X				
Skew					X	X	X	
Hole Size					X	X	X	
Profile cut begin and end					X	X	X	
Radius					X	X	X	
Revolutions					X	X	X	

Tabla 11: Tabla de prims y sus modificadores

Si observamos la tabla podemos llegar a una conclusión: existen tres grupos de primitivas que se ven afectados por modificadores.

En el primer grupo tenemos la caja, el cilindro y el prisma, y los modificadores disponibles son path cut, hollow, hollow shape, twist, taper y top shear.

El segundo grupo está constituido exclusivamente por la esfera, que tiene disponible solamente algunos de los anteriores modificadores pero posee uno en exclusividad: dimple.

Por último el tercer grupo está compuesto por los prims toro, tubo y anillo, que junto a los modificadores del primer grupo, añaden los siguientes: skew, hole size, profile cut, radius y revolutions.

Por lo tanto, para ver un ejemplo de cada modificador tendremos que realizar ejemplos con al menos un primitivo de estos tres grupos. Utilizaremos la caja, la esfera y el toro para estudiar los efectos de los modificadores.

7. Modificadores.

7.1. Modificadores para el prim Caja:

En la imagen inferior se puede apreciar un primitivo tipo Caja que utilizaremos como base para aplicar modificadores. Es un cubo y por tanto tiene 6 caras del mismo tamaño.

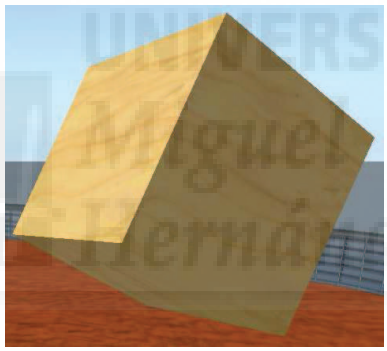


Imagen 6.3.19: Objeto caja

- **Modificador Path Cut Begin and End:**

Se puede traducir como Trayectoria de corte comienzo y fin. Este modificador se basa en dos números: B (principio) y E (end). B vale inicialmente 0.000 y E vale 1.000. Esto significa que la primitiva está completa. Si B aumenta se producirá un corte transversal en la primitiva en cierto porcentaje desde el principio al fin. Es como si le faltara un trozo a una tarta. El valor de E también puede variar hacia abajo y recortar la figura, pero empezando por el final y realizándolo hacia el comienzo.

Por ejemplo, en la imagen 5.2.6 se puede observar como quedaría nuestro cubo aplicando los valores B= 0.100 y E= 0.800. Lo hemos girado para ver mejor el efecto.

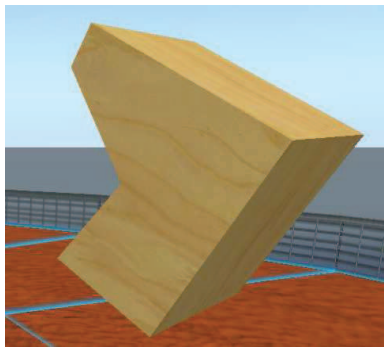


Imagen 6.3.20: Path cut begin and end

- **Hollow, orificio**

Traducido en el programa por “Vacía”, aunque creo que sería mejor utilizar la palabra agujero: este valor puede ir de 0 al 95% del tamaño de la primitiva y generará un orificio en dirección trasversal, por lo que generará un vacío interior.

- **Hollow shape, forma vacía:**

Si existe orificio puede ser de sección circular (Circle), cuadrada (Square) o triangular (Triangle). Existe para cada primitiva una hollow shape por defecto que también podemos elegir. En algunos objetos como los Toro, la forma triangular se convierte en hexagonal.

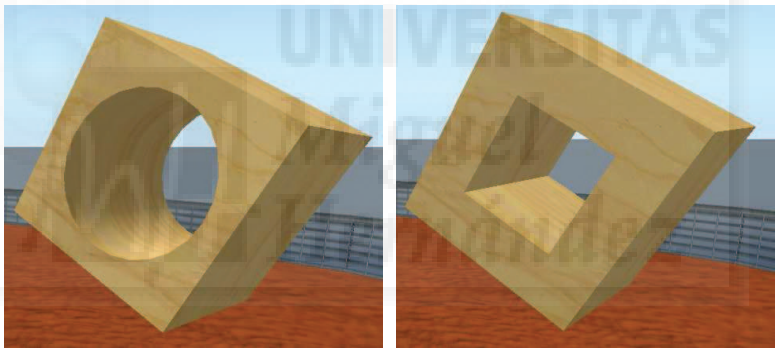


Imagen 6.3.21: Forma del orificio trasversal

- **Twist begin and end.**

Lo podríamos traducir por retorcido del comienzo al fin: este valor se basa en dos números: B (principio) y E (end). Normalmente B y E valen 0.000. Esto significa que no está retorcido. En la imagen se puede observar al cubo de muestra retorcido 180 unidades en B y 0 en E. Los valore admitidos van desde –180 hasta 180 tanto para B como para E.

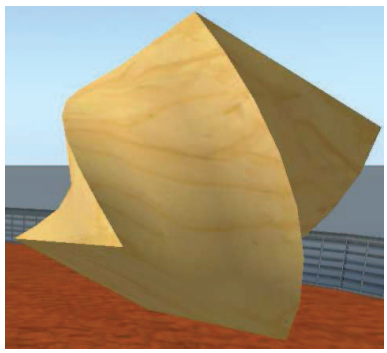


Imagen 6.3.22: Modificador Twist

- **Taper.**

Este modificador también se conoce como “Afilar”. Lo que hace es disminuir las caras del objeto en las dimensiones X e Y. Los valores permitidos van de –1 (para la cara inferior) hasta 1 (para la cara superior). 0 significará que no está disminuida y 1 que está totalmente reducida. En la imagen se puede observar cómo queda el cubo para los valores X=0.80 Y 0.40. Si pusiéramos los valores X e Y a 1, en realidad, obtendríamos una pirámide cuadrada.

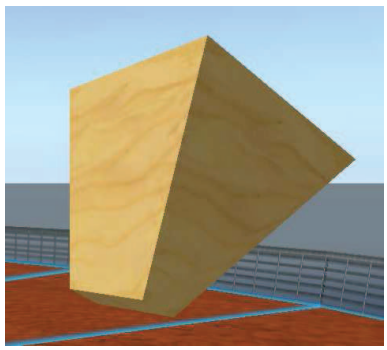


Imagen 6.3.23: Modificador Taper

- **Shear.**

Esta modificación produce un “Sesgo trasversal”. Lo que hace es modificar el paralelismo entre las caras superior e inferior. Los valores pueden ser positivos o negativos para un sentido u otro. El rango de valores va desde –50 a 50. En la imagen siguiente se puede observar el cubo con esta operación para los valores X= 0.50 Y = 0.00

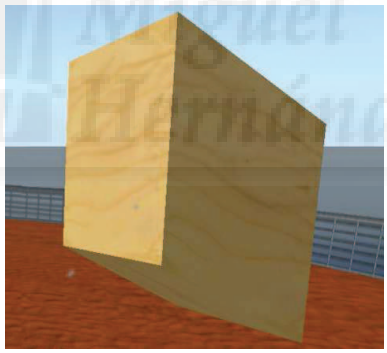


Imagen 6.3.24: Modificador Shear

7.2. Modificadores para el prim Esfera:

- **Dimple.**

En la figura que sigue podemos ver una esfera y como queda cuando le aplicamos el modificador que es exclusivo de esta primitiva: dimple. Es difícil explicar como funciona, ya que produce efectos distintos si sobrepasa o no el ecuador de la esfera. Funciona mediante dos parámetros Begin (de 0.000 a 0.950) y End (de 1.000 a 0.050). En la figura los valores son begin: 0.200 y end: 0.300.

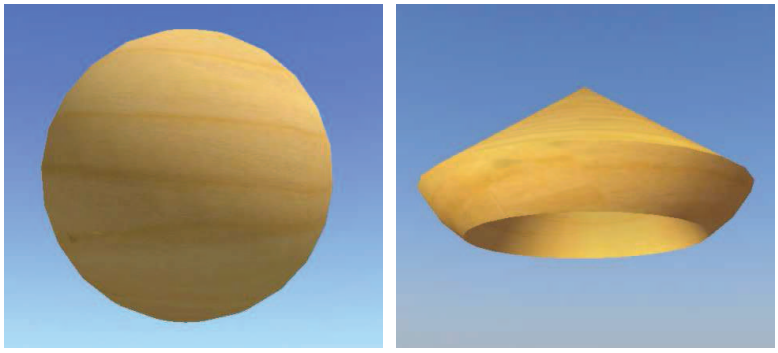


Imagen 6.3.25: Modificador Dimple Begin and End aplicado a una esfera

7.3. Modificadores para el prim Toro:

Como escribimos con anterioridad, se pueden aplicar modificadores que solo existen para el toro, el anillo y el tubo y no para las demás primitivas.

En la imagen que sigue se puede observar un objeto toro recién creado, los valores de los modificadores están todos por defecto. Por ejemplo, los que indican el tamaño del orificio interno llamado Hole size valen X: 1.00 e Y: 0.25.

Imagen 6.3.26: Primitiva toro

- **Hole size.**

Este modificador cambia el tamaño del orificio interno y por tanto también de todo el toro. El de la imagen de abajo tiene unos valores para hole size de X: 0.35 e Y: 0.25

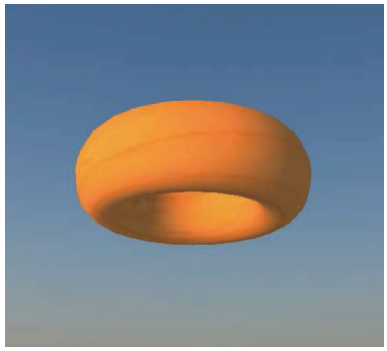


Imagen 6.3.27: Modificador Hole size

- **Profile Cut Begin and End.**

Imagen 6.3.28: Modificador Profile Cut begin and end

Este modificador cambia el perfil curvo por otro recto mediante dos valores comienzo y fin que definen la forma externa del toro. En la imagen anterior se observa como desde arriba (desde el fin) se hace el perfil recto hasta el 70 %. Los valores son B: 0.000 y E: 0.700.

- **Revolutions.**

Este modificador hará que el toro defina una serie de curvas con respecto a su eje trasversal. Los valores permitidos van de 0 (queda normal) hasta 4, que es el valor que se muestra en la imagen que sigue.

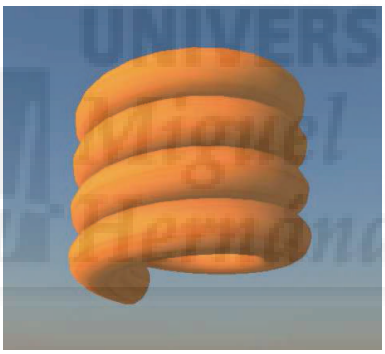


Imagen 6.3.29: Modificador Revolutions

- **Skew:**

Este modificador produce un cambio radical en la forma de la primitiva. En un principio, corta transversalmente el toro por la mitad y luego lo va retorciendo, haciendo que su parte central disminuya de sección. En la imagen de abajo se observa este modificador en su valor mayor, 0.95, por lo que el cambio es el más radical. Los valores que puede tomar van de –0.95 a 0.95.

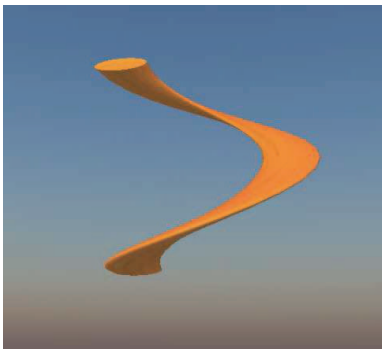


Imagen 6.3.30: Modificador Skew

- **Twist Begin and End:**

Como hemos visto antes, twist produce que se retuerzan sobre sí mismos los objetos. En la imagen de abajo se observa este efecto en un toro para los valores B: 72 y E: 0. Los valores que pueden tomar van desde –360 a +360 tanto para B como para E.

Imagen 6.3.31: Modificador Twist begin and end

- **Radius:**

Este modificador corta transversalmente el toro y luego crea una espiral que en sus valores máximos (-0.667 a 0.007) tomará la forma que se observa en la siguiente imagen.

Imagen 6.3.32: Modificador Twist begin and end

Por último, queremos hacer ver lo que podemos conseguir con una sola primitiva al usar sus modificadores de forma conjunta. En la imagen que sigue tenemos un solo toro con los valores: Hollow: 75; Hollow shape: circle; Profile Cut Begin and end: B: 0.000 E: 0.750

Imagen 6.3.33: Modificadores combinados

9. Qué son y cómo crear los Sculpted prims.

Un sculpt, sculptie o sculpted prims es un objeto primario de SL cuya forma viene determinada por una serie de coordenadas x, y, z almacenadas como valores RGB en un fichero de imagen llamado Sculpt Texture o Sculpt Map. Son objetos esculpidos por texturas, es decir, su geometría se obtiene mediante la aplicación de una textura a la superficie de un objeto base.

Se usan para crear formas complejas, normalmente con formas orgánicas que sean muy difíciles de conseguir con los objetos primitivos de SL. Normalmente, se realizará un sculptie para modelos que en otros programas haríamos con mallas de vértices o splines.

Imagen 6.3.34: Sculpted prims

En la imagen anterior se pueden ver 7 prims de tipo sculpted: dos plátanos, dos manzanas, dos naranjas y un plato. Posiblemente tengamos que haber producido solo 4 sculpt map, ya que los plátanos, las manzanas y las naranjas tienen la misma forma y por tanto comparten el mismo sculpt map.

Para aclarar este concepto nos fijaremos en los plátanos (lo mismo parece suceder con las manzanas pero no con las naranjas), que parecen distintos en su superficie pero idénticos en su forma. La forma la proporciona la “Textura sculpt” y la superficie la imagen “Textura”. Por lo tanto, para poder realizar los objetos que se observan en la imagen, el creador habrá “subido” a Second Life un total de10 texturas: 4 para las formas, 1 para la superficie del plato, otra para las naranjas, 2 para las manzanas y 2 más para los plátanos.

Second life apenas proporciona herramientas para trabajar con Sculpties. Esto hace necesario que los creadores usen software 3D como 3D Studio MAX, Blender, Maya, etc., para producir las mayas que luego serán exportadas como ficheros de imágenes.

Estas imágenes serán subidas a SL (cada vez que subimos una textura, nos cuesta 10 L$) y aplicadas a un objeto definido como Sculpted prim mediante las herramientas habituales de construcción de objetos. Second Life interpretará esta imagen para dar forma al objeto tal como se muestra en la imagen inferior.

La forma en que se crean los modelos y luego se exportan como ficheros de imágenes varía mucho de un software a otro. Nosotros utilizaremos 3D Studio MAX por ser uno de los programas más conocidos por los diseñadores 3D y además para mantener cierta coherencia con los casos prácticos de Director.

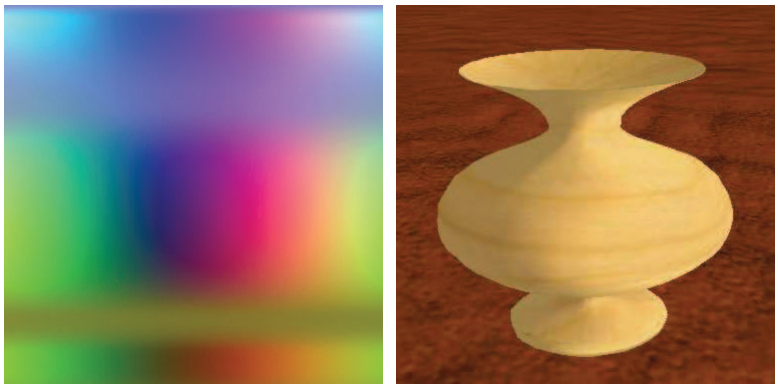


Imagen 6.3.35: Sculpted map y su aplicación a un sculpt prim

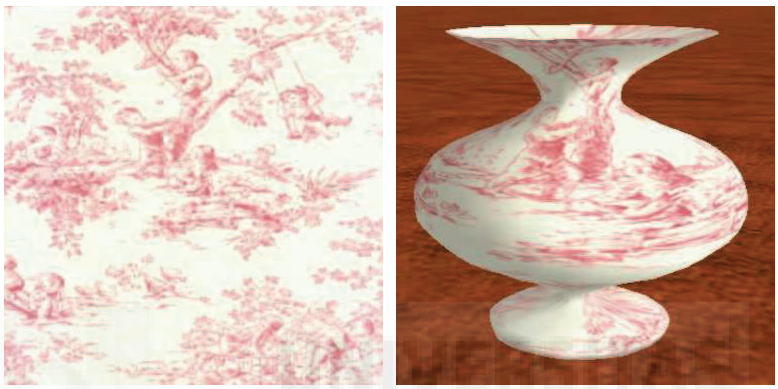


Imagen 6.3.36: Textura y su aplicación a un sculpt prim

La textura que vemos arriba está incorporada en Second life y la podemos encontrar en el Inventario > Library > Textures > Fabric con el nombre “PinkToile”.

10. Otras características no modelables de los objetos.

- **Pestaña carácter o Features:**

Permite caracterizar los objetos con flexibilidad y luz. En esta pestaña tenemos la posibilidad de hacer que los objetos que creemos posean dos propiedades muy interesantes: la flexibilidad y la luz.

El tema de la luz de los objetos lo estudiaremos en la unidad 5 que está dedicada a la iluminación en Second Life, por lo que los parámetros que nos encontramos en esta pestaña se estudiarán con detalle en dicho tema.

Así mismo, los objetos flexibles se estudian en el punto 5 de la unidad 7 que aborda el tema de las simulaciones físicas, por lo que remitimos al lector a dicho punto para ahondar en estos parámetros.

Imagen 6.3.37: Pestaña Caracter

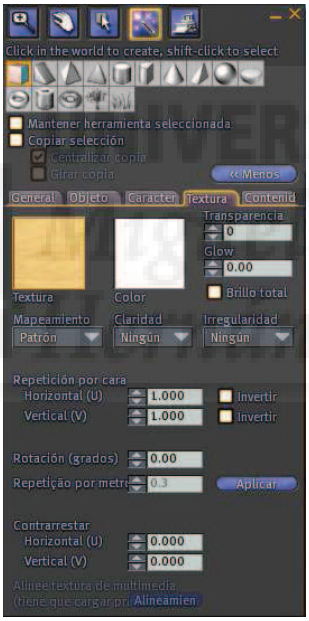


Imagen 6.3.38: Pestaña Textura

- **Pestaña textura o Texture:**

Permite definir materiales, superficies y texturas y aplicarlos a los objeto. Esta pestaña la estudiaremos a fondo en el siguiente tema que trata de la texturización de los objetos, por lo que nos remitimos al mismo.

- **Pestaña contenido o content:**

Permiten hacer de un objeto, un contenedor. Por ejemplo, podemos introducir scripts, texturas, sonidos, otros objetos, etc. En esta pestaña encontraremos normalmente los materiales que el objeto utilizará para su funcionamiento, apariencia y otras características.

Para colocar los elementos dentro de un objeto, basta con abrir la pestaña Contents y arrastrarlos desde el Inventario. Tenemos que tener en cuenta los permisos que poseemos

sobre los objetos, ya que algunos afectan a su inclusión dentro de otros objetos. En la actualidad no hay un límite máximo de objetos que podamos introducir. Además los objetos multiprim cuentan como una unidad.

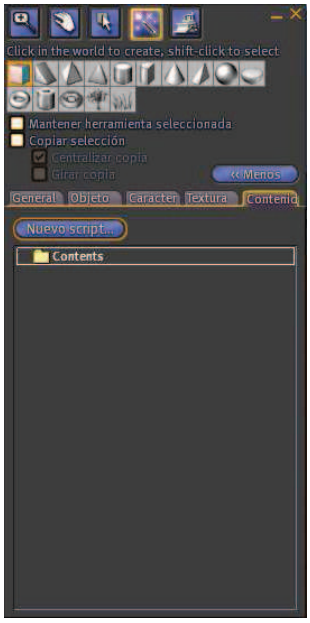


Imagen 6.3.39: Pestaña Contenido

11. Sitios para aprender más.

Existen islas pensadas especialmente para ayudar a los usuarios en la construcción de objetos. Posiblemente la más famosa sea la isla “Ivory Tower”, donde encontraremos muchos recursos sobre construcción de objetos.

Está totalmente estructurada para que la adquisición de conocimientos y recursos sea gradual y por tanto, el aprendizaje del modelado de objetos sea más cómodo, ya que no es un tema fácil.

En las siguientes imágenes podemos apreciar algunas vistas de la isla Ivory Tower.

Imagen 6.3.40: Ivory Tower y una sala de aprendizaje

Caso práctico 3.1: Crear objetos prim.

Objetivo: Crear objetos primitivos, conocer sus principales características y almacenarlos.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Buscar un sandbox.
2. Crear un objeto primitivo.
3. Modificar sus propiedades básicas.
4. Comprobar sus propiedades.
5. Guardar en el inventario.

1. Buscar un sandbox.

Para crear un objeto necesitamos un sitio que permita esta función. En Second Life solo se pueden crear objetos en zonas propias de las que seamos los propietarios o en zonas donde el propietario crea lugares especiales conocidas como areneros o “sandbox”.

Los sandbox no son un lugar para acumular objetos sino para que los usuario puedan empezar a aprender a construir, o como un lugar de encuentro de “constructores”. Normalmente no tienen las mismas posibilidades que si tuviéramos nuestra propia parcela. Además debemos de guardar una serie de normas que varían de unos sandboxes a otros. Por ejemplo, la mayoría de ellos tienen un límite de tiempo (normalmente unas horas) para los objetos, pasado el cual, se borran. Esto no es problema en la mayoría de los casos, ya que es responsabilidad del usuario guardarlo en su propio inventario. Además, es inevitable que los sandbox pongan este tipo de límites ya que hay usuarios que prueban cosas o las dejan a medias y luego se trasladan a otro lado dejando tras de sí su “basura” como desgraciadamente pasa en el mundo real. Por ello, lo primero que vamos a hacer es buscar un sandbox mediante la orden buscar.

1.1. Menú Edit > Search (o Control + F), esta es la herramienta de búsqueda de Second Life. Permite buscar sitios, usuarios, etc.

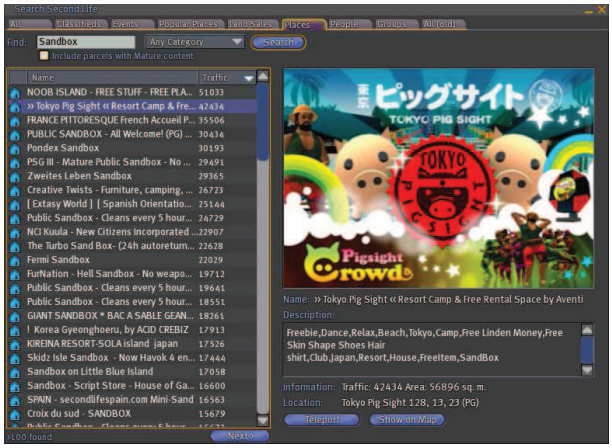


Imagen 6.3.cp.1.1: Herramienta de búsqueda Search de Second Life

1.2. Hacemos clic sobre la ficha Places. En Find escribimos la palabra “Sandbox”. Podemos ordenarlo por tráfico de menor a mayor para poder elegir un sitio más discreto o con más densidad de usuarios.

1.3. Nosotros nos decidimos por “HackIndex Sandbox”, así que lo señalamos y pulsamos el botón “Teleport”. Esto hará que lleguemos al lugar elegido directamente.

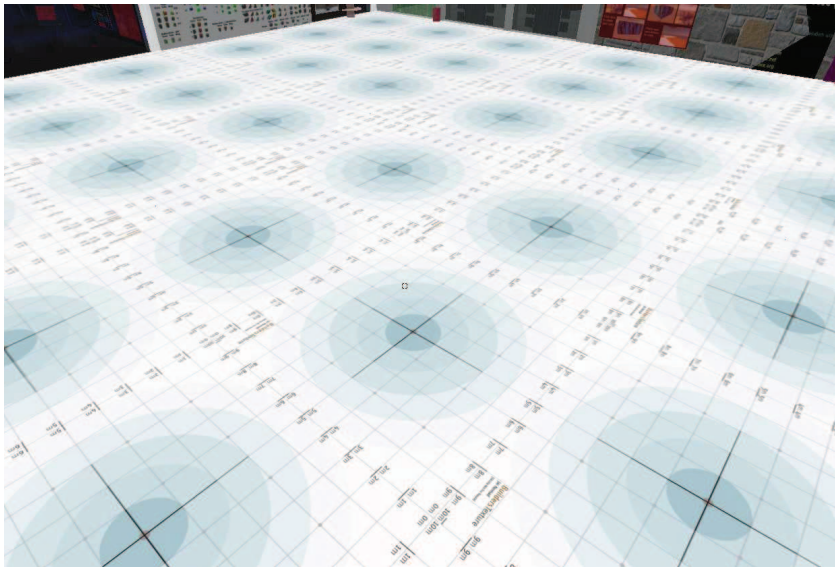


Imagen 6.3.cp.1.2: Sandbox elegido para nuestro caso práctico

2. Crear un objeto primitivo.

2.1. Hacemos clic con el botón derecho para que aparezca el siguiente menú.

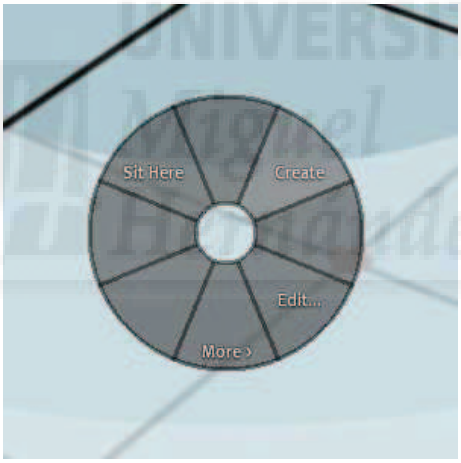


Imagen 6.3.cp.1.3: Menú para crear objetos

2.2. Elegimos Create (Crear) y a continuación, hacemos clic sobre el primitivo que queramos, en este caso, nosotros hemos elegido la esfera.

2.3. Haciendo clic en el suelo aparece una esfera como la que se muestra.

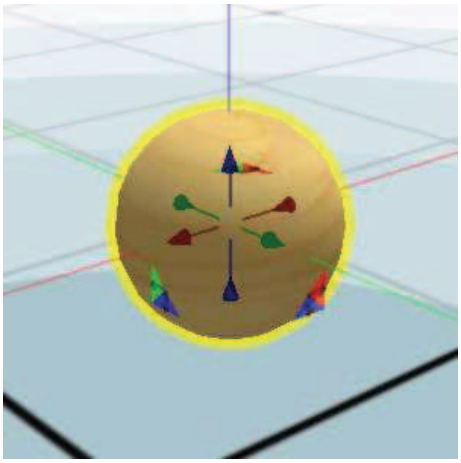


Imagen 6.3.cp.1.4: Modo de edición por defecto

Podemos observar el modo de edición por defecto. Las flechas que aparecen permiten mover los objetos interactivamente. Veremos en la siguiente práctica cómo se puede mover mediante la introducción de valores precisos.

Si queremos borrarla basta con que pulsamos la tecla Suprimir o si ya hemos terminado de editarlos, hacer clic con botón derecho y en el menú contextual elegiremos More > Delete.

3. Modificar sus propiedades básicas.

Lo primero que vamos a realizar es nombrar nuestro nuevo objeto.

3.1. Si no vemos los paneles inferiores aparecerá en su lugar el botón More, por lo que tenemos que usar una u otra alternativa. A continuación, pulsamos la ficha “General”.

3.2. En el control nombre introducimos “Cabeza” y en Descripción escribimos “Parte de un maniquí”.

En la figura que sigue podemos comprobar que automáticamente aparece nuestro nombre tanto para creador como para propietario (owner). Esta es la opción por defecto, pero se puede cambiar y por ejemplo que el creador esté trabajando para construir objetos que serán propiedad de otras personas.

3.3. Hemos puesto activa la opción “Share with group” para poder de esta manera, compartirla con nuestro grupo (suponiendo que pertenecemos a alguno).

Las demás propiedades no las hemos modificado. Algunas son muy importantes, por ejemplo para las opciones comerciales de Second Life, sobre todo las que tienen que ver con los permisos y el precio pero como no tienen relación con la construcción del objeto no las veremos en esta práctica.

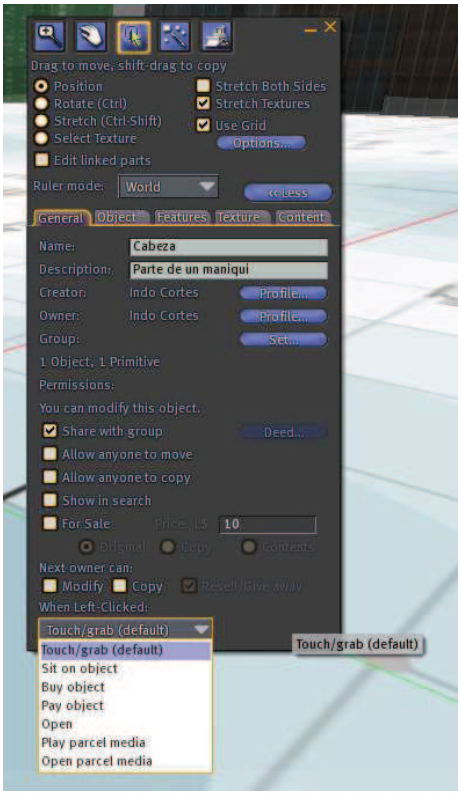


Imagen 6.3.cp.1.5: Introduce el nombre y la descripción del objeto

A destacar la propiedad “When left-clicked”, que aporta la interactividad básica con el objeto. No la modificamos y queremos que al hacer click sobre él, nuestro objeto lo interprete como un “tocar”.

3.4. Abrir la ficha de “Object”. En esta ficha tendremos las funciones más importantes para editar el objeto.

Supongamos que después de trabajar con nuestro nuevo objeto, decidimos que la “cabeza” del maniquí la queremos empezar a modificar a partir de un cubo y no de una esfera.

3.5. En “Building Block Type” elegimos Box en lugar de Sphere, con lo que nuestro objeto pasará a ser una caja al instante. Este control lo destacamos en rojo en la imagen siguiente.

Además (con fines que de momento se nos escapan al objetivo de esta práctica) queremos que su materia sea metálica (y no de otra cualquiera como madera o plástico). Para ello hacemos lo siguiente:

3.6. Abajo, tal como muestra la imagen que sigue, en la lista desplegable “Material”, que se visualiza con un rectángulo amarillo, elegimos la opción “Metal”.

Esta elección, no tiene mucho sentido si el objeto no es físico, es decir, no se va a ver involucrado en caídas, interacciones, etc. Por lo que lo definimos como físico en la opción que se destaca en color fucsia en la imagen siguiente.

3.7. En la opción Physical, pulsamos para activarla.

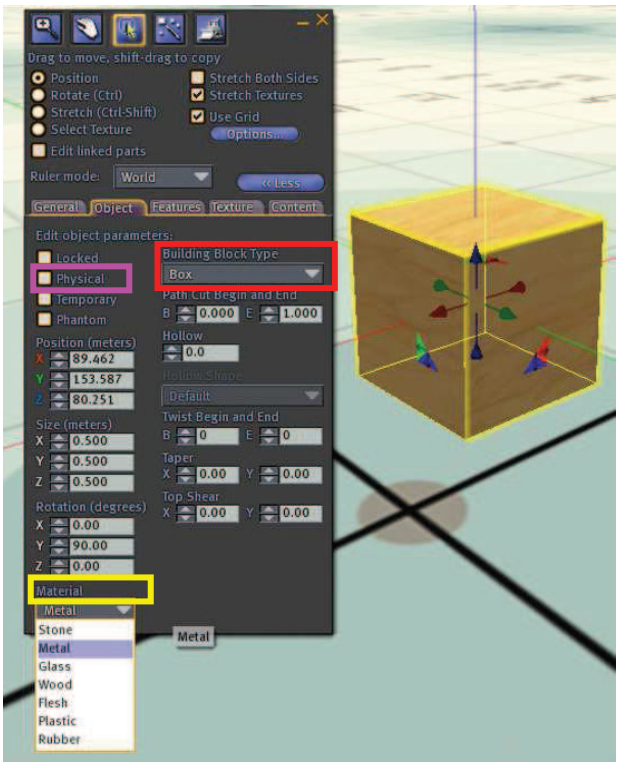


Imagen 6.3.cp.1.6: Opciones tipo de bloque y material

4. Comprobar sus propiedades físicas.

Para comprobar que es un objeto físico, basta con que lo movamos hacia arriba y lo dejemos caer. Para ello podemos hacer lo siguiente:

4.1. Pulsamos el botón derecho del ratón sobre el objeto. Y elegimos la opción Edit.

4.2. Aparecerán las flechas para cambiar su posición, lo que haremos es seleccionar la azul y arrastrar hacia arriba, con lo que el objeto sube y se queda en el aire esperando más modificaciones.

4.3. Cerrar la ventana de construcción. En cuanto la cerremos el simulador detecta que es un objeto con peso y al estar en el aire caerá. Repetir la operación con diferentes alturas y sobre todo con diferentes materiales, ya que si por ejemplo, es de material plástico rebotará contra el suelo más que si es de metal.

Por lo tanto el material tiene que ver con la física del objeto y la masa de un objeto será directamente proporcional a su tamaño.

5. Guardar en el inventario.

Cuando queramos guardar un objeto que nos puede ser útil, lo podemos dejar en nuestra tierra si la tenemos o en nuestro inventario. Para ello seguir los siguientes pasos.

5.1. Hacer clic con el botón derecho el ratón sobre el objeto. Aparece el menú contextual.

5.2. Elegir la opción Take.

Esto supone que el objeto desaparece de la escena para pasar al inventario. El objeto pasará a nuestro inventario con el nombre que le hemos asignado.

Si queremos borrarlo basta con elegir la opción “Delete” del menú contextual en el inventario. También podemos crear carpetas en el inventario para gestionar nuestros objetos y poder agruparlos a voluntad.

Para volver a tener el objeto en pantalla lo arrastraremos desde el inventario a la escena, podemos arrastrar tantos objetos como queramos, estos serán instancias del objeto del inventario. Los cambios producidos en estos nuevos objetos no afectarán al objeto guardado a menos que utilicemos de nuevo la opción Take con el mismo nombre del objeto. Por lo tanto, este método puede ser muy útil para hacer diversas versiones a partir de un objeto base y solo guardar aquellas que realmente queramos mantener.

Imagen 6.3.cp.1.7: Guardar un objeto en el Inventario

Conclusiones:

En este primer caso práctico hemos aprendido cómo realizar una objeto, modificar algunas de sus propiedades y almacenarlo para utilizarlo posteriormente. Creemos sinceramente que es una práctica sencilla que demuestra que Second Life es un sistema orientado a la creación de un mundo virtual por parte de personas a las que no se les exigen conocimientos específicos previos. Por lo tanto, es un sistema asequible.

Caso práctico 3.2: Operaciones básicas con objetos.

Objetivo: Controlar la posición, rotación y dimensiones de los objetos.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Extraer un objeto del inventario.
2. Modificar la posición de los objetos.
3. Modificar las dimensiones del objeto.
4. Rotar un objeto.
5. Trabajar a la vez con varios objetos.

1. Extraer un objeto del inventario.

El Inventario actúa como un almacén de objetos. Es un almacén privado de cada usuario y por tanto el recurso básico para guardar nuestros objetos.

En el caso práctico anterior vimos como se pueden almacenar los objetos mediante el comando “Take”. La operación inversa es bien sencilla pero otra cuestión puede ser cómo buscar un objeto en un inventario que a veces puede acumular muchos objetos.

1.1. Abrir el inventario pulsando sobre el botón homónimo que se encuentra en la esquina inferior derecha de nuestro interfaz.

1.2. En el espacio en blanco de la parte superior escribir el nombre del objeto a buscar y aparecerán todas las coincidencias como muestra la siguiente imagen.

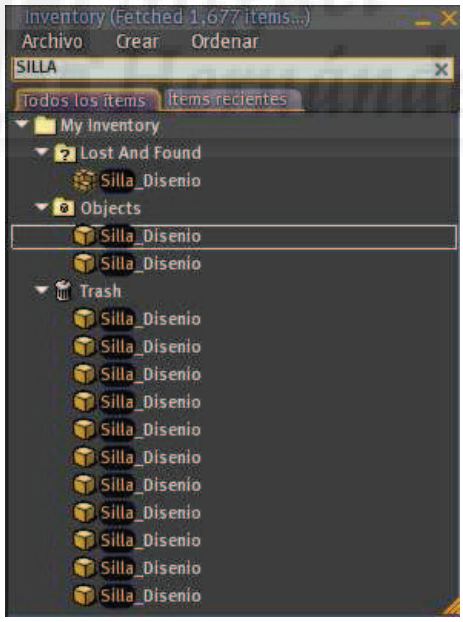


Imagen 6.3.cp.2.1: Buscar un objeto en el Inventario

1.3. Una vez encontrado el objeto hacemos clic sobre él y lo arrastramos hasta el sanbox. Esto hará que aparezca (rez en jerga de Second Life), es decir, que se represente como muestra la imagen inferior.

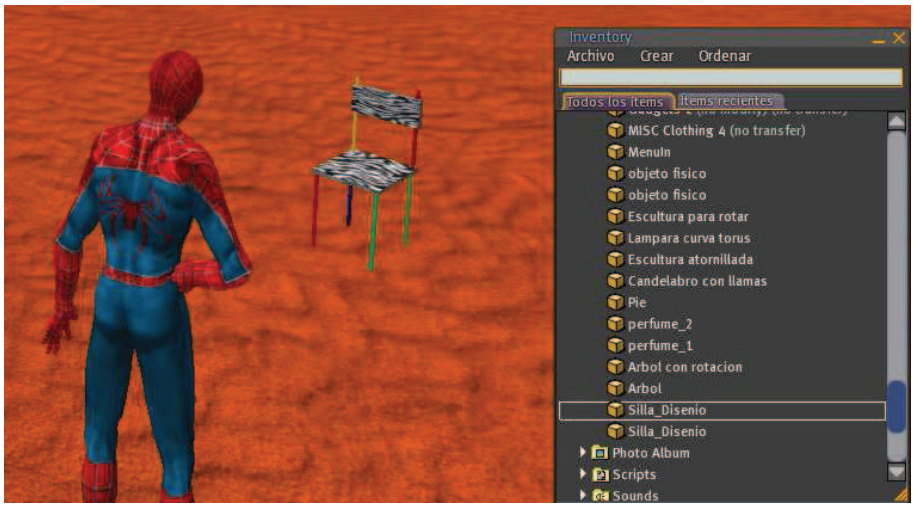


Imagen 6.3.cp.2.2: Objeto extraído de nuestro Inventario

2. Cambio de posición de los objetos.

La posición de un objeto se puede modificar mediante dos procedimientos: interactivamente con los controles en forma de flecha y de modo más preciso introduciendo directamente las coordenadas en la ficha Objeto.

Para cambiar la posición interactivamente hacemos:

2.1. Seleccionar el objeto y editarlo mediante la pulsación de botón derecho sobre el propio objeto y elegimos la opción editar.

2.2. Seleccionar uno de los 3 controles en forma de flecha para cada uno de los ejes, por ejemplo, hacer clic sobre la flecha verde y arrastrar para mover hacia delante o atrás el objeto.

Para cambiar la posición mediante coordenadas haremos:

2.3. Abrimos la ficha “Objeto” y escribimos directamente en los controles X, Y, Z el valor que queremos que tome. Por ejemplo, si queremos mover 2 metros en el eje X la silla cambiaremos su valor en X que vale 187 metros por 189 metros, como muestran la siguiente imagen.

Cabe recordar que si el objeto es, como en este caso, un objeto compuesto por varios prims enlazados, la posición que muestra los valores se refiere al centro geométrico del prim raíz del objeto compuesto.

En la siguiente imagen se ven dos momentos del cambio de posición de la silla. En la parte de arriba se aprecia que está en la posición X con un valor de 187 metros y en la parte de abajo el valor es de 189 metros y desde la posición de la cámara se visualiza más alejada.

De esta imagen podemos deducir varias conclusiones: que la posición se mide en metros y puede tener una precisión de milímetros, que en principio, los 189 metros es con respecto al origen del simulador donde estemos, es decir, de la isla donde nos hallamos en este momento, y por último que la visión de la posición no es objetiva, ya que depende de dónde se sitúe la cámara con que observamos la escena.

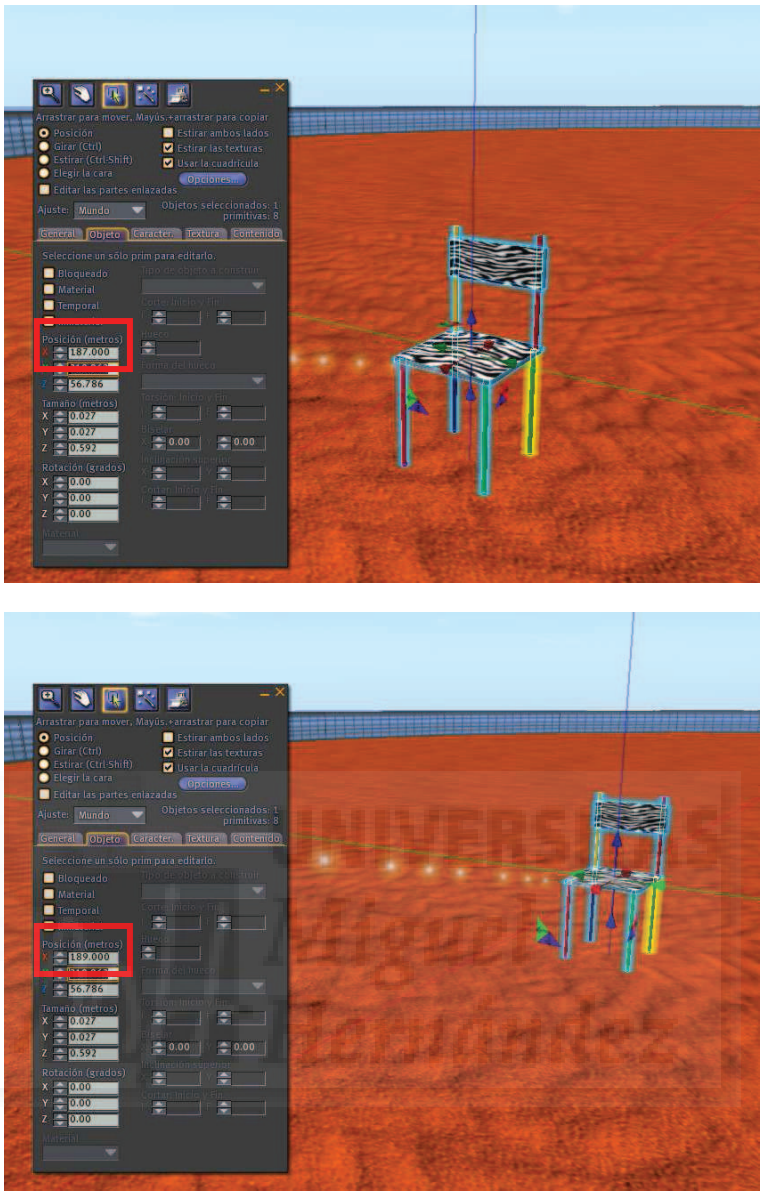


Imagen 6.3.cp.2.3: Cambio de posición del objeto

3. Modificar las dimensiones de los objetos.

Si el objeto es un prim simple, podemos cambiar el tamaño de este de dos formas: interactivamente con los controles de redimensionamiento o de forma numérica con los controles de tamaño que aparecen en la ficha “Objeto”.

Por supuesto que la primera opción es más precisa, ya que al introducir una medida mediante un número estamos describiendo la magnitud totalmente. Second Life permite una precisión de milímetros. Debemos recordar que los objetos básicos, o sea, los prims, no pueden ser mayores de 10 metros para cada una de sus magnitudes. Estas medidas se pueden introducir desde los controles X, Y, Z de la pestaña “Edit” de la caja de diálogo de edición de objetos.

Al ser la silla un elemento enlazado solo podemos aplicar la escala interactiva, es decir, “a ojo”, ya que de esta forma se redimensiona toda la silla y no solo el prim raíz.

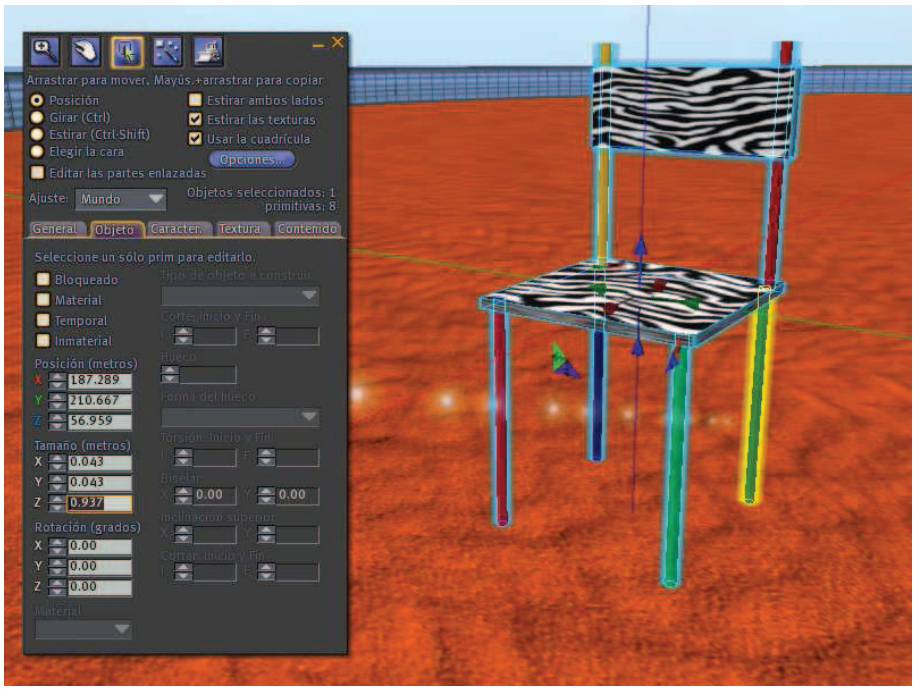


Imagen 6.3.cp.2.4: Cambio de tamaño del objeto

3.1. Una vez seleccionado el objeto y en situación de editarlo, pulsar las teclas Shift y Ctrl a la vez para que aparezcan sobre las esquinas de su rectángulo delimitador unos cubos de color gris claro (si el objeto no es compuesto aparecen más cubos de distintos colores según el eje) que representan los controles de redimensionamiento.

3.2. Manteniendo las teclas pulsadas, seleccionar un control cualquiera y arrastrar hacia fuera para hacer el modelo más grande o hacia el interior para hacerlo más pequeño.

En la imagen superior se observa la silla en el misma posición y con la misma cámara que anteriormente pero mucho más grande. El apunte de la cámara es importante ya un objeto puede parecer más grande si lo observamos más de cerca.

4. Rotar un objeto.

También la operación de rotación de un objeto se puede hacer de forma interactiva o numérica, pero en este caso, aunque el modelo sea compuesto podemos utilizar los dos métodos. La operación de girar objetos se puede realizar de forma interactiva arrastrando unos controles en forma de aros que aparecen cuando pulsamos la tecla Control.

4.1. Seleccionar y editar el objeto a rotar.

4.2. Mantener la tecla Ctrl y arrastrar uno de los controles según el eje deseado.

4.3. En el control X del grupo Rotación, introducir el valor 45 para que el modelo gire 45 grados en el eje X.

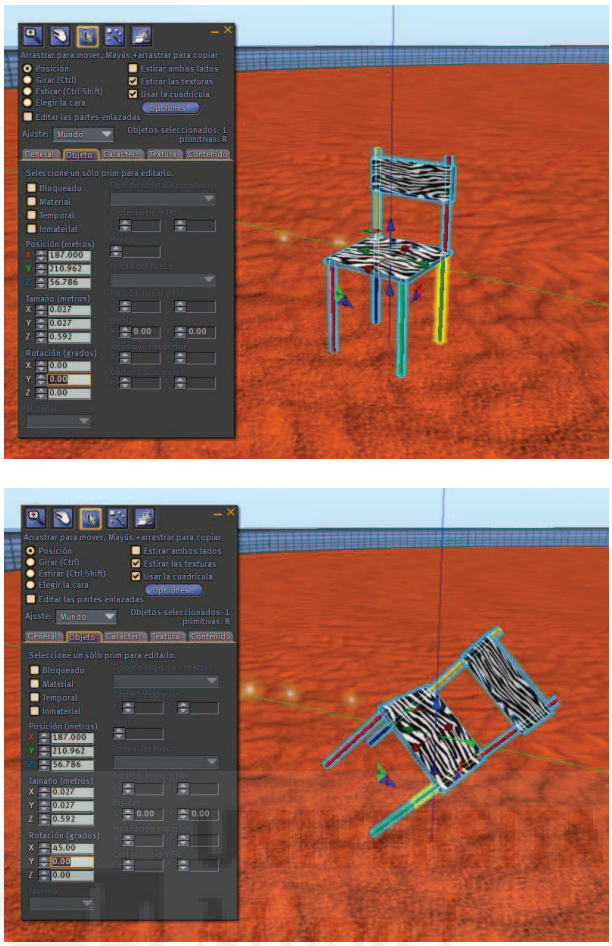


Imagen 6.3.cp.2.5: Rotación de un modelo compuesto

En la imagen anterior se observa en la parte superior el modelo compuesto sin rotación y en la parte inferior, el mismo modelo pero con un giro en el eje Y introducido de manera numérica.

5. Trabajar a la vez con varios objetos.

Como es evidente, debemos tener varios objetos para poder ver cómo se pueden transformar a la vez. Para copiar objetos, sean estos simples o compuestos por varios prims, podemos utilizar el siguiente método.

5.1. Seleccionar el objeto a copiar, en este caso la silla, y editarlo.

5.2. Pulsamos la tecla Shift y al mismo tiempo arrastramos una de las flechas de control de movimiento que aparecen sobre el objeto editado según el eje en el que queramos crear la copia. Hacer varias copias, en la imagen inferior se observan 6 sillas en total.

Para trabajar con varios objetos, es necesario seleccionarlos todos a la vez. Luego le aplicamos cualquier transformación, por ejemplo, una rotación.

5.3. Hacer clic en una silla cualquiera y mantener la tecla Shift pulsada mientras hacemos clic sobre todas las demás sillas. Observar como el punto del que parten las flechas cambia al centro de la selección múltiple que hemos realizado.

5.4. Pulsamos Ctrl + Shift para que aparezcan los controles de rotación y arrastramos uno cualquiera de ellos para observar cómo la operación se aplica a todo el grupo en su conjunto como se puede ver en la imagen de abajo.

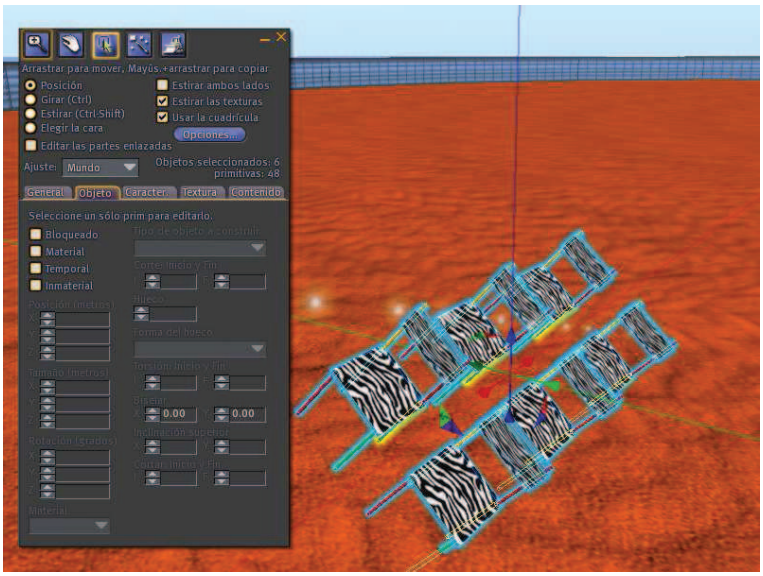


Imagen 6.3.cp.2.6: Rotación de un modelo compuesto múltiple

Conclusiones:

Las operaciones que hemos estudiado en esta práctica son fundamentales para tener un control adecuado del objeto. A partir de aquí podemos entrar en otras cuestiones más particulares pero la operaciones de posición, giro y escala son tan universales y usuales que se dan en todos los programas de edición ya sean en dos o en tres dimensiones.

Caso práctico 3.3: Objetos enlazados.

Objetivo: Aprender cómo crear y editar objetos compuestos mediante enlace de otros.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Crear los objetos primitivos.
2. Enlazar los objetos.
3. Utilizar el objeto enlazado.
4. Desenlazar uno de los objetos.

1. Crear los objetos primitivos.

Crearemos un objeto diseñado de forma modular, a base de 3 objetos primitivos. Estará compuesto por una caja, un cilindro y un prisma triángulo. En la siguiente imagen se puede apreciar el objeto terminado.

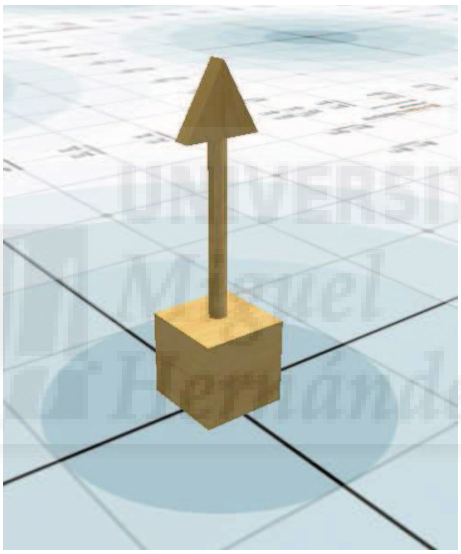


Imagen 6.3.cp.3.1: Objeto realizado por enlace de otros objetos

1.1. Hacemos clic en el suelo del sandbox con el botón derecho y elegir la opción Create. Elegimos la opción Caja y hacemos clic en el suelo. Aparecerá un cubo que seguidamente vamos a modificar.

1.2. En la ventana Create, elegimos la ficha Object y en Size dejamos los tres valores a 0.5 metros.

1.3. Ahora vamos a realizar dos copias de la caja que luego convertiremos en el cilindro y el triángulo. Para ello, con la tecla Shift pulsada arrastrar la flecha azul hacia arriba y soltar. Crearemos una copia de la caja 1. Repetir la operación para una tercera caja como se observa en la imagen siguiente.

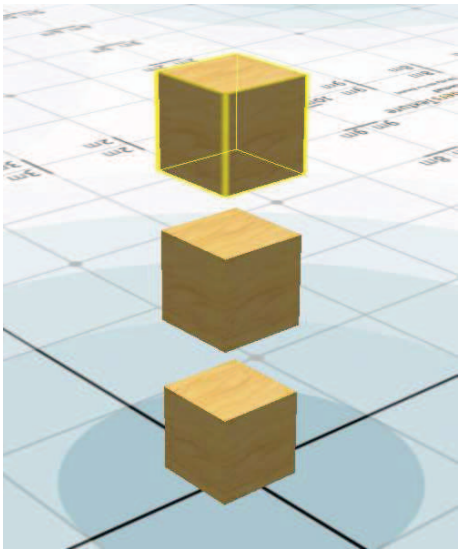


Imagen 6.3.cp.3.2: Copia de cajas para su posterior modificación

Esta forma de realizar una “serie” de objetos es muy típica de Second Life. Esta estrategia de modelado tiene la ventaja de que los tres objetos sea cual sea su forma están ya alineados verticalmente de manera automática.

1.4. Seleccionamos el segundo objeto y en la ficha Object > Building Block Type elegimos Cylinder. Repetimos la operación anterior con la tercera caja pero la convertimos en un Prism.

1.5. Modificamos el prisma en la ficha Object poniendo los valores Size: 0,5; 0,5; 0,1 y Rotation: 90; 0; 90;. Modificamos el cilindro en la ficha Object poniendo los valores Size: 0,1; 0,1; 1,0 y Rotation: 0; 0; 0;

El objeto debe ir tomando forma tal y como muestra la imagen siguiente.

1.6. Ahora vamos a poner juntos los tres elementos de tal manera que se toquen. Para ello basta con seleccionarlos haciendo clic. Aparecerán unas flechas de colores para moverlos en los tres ejes. En este caso solo debemos arrastrar las flechas azules que moverán los objetos en el eje vertical.

Si no lo queremos hacer “a ojo” podemos utilizar los valores Position de la ficha Object para saber cual es su posición exacta y teniendo en cuenta sus dimensiones, pegarlos con gran exactitud. Este método no se suele utilizar porque es bastante complicado.

Para esta tarea puede ser de gran ayuda el control de la cámara de visualización que se obtiene pulsando simultáneamente la teclas Control + Alt y arrastrar para acercar o alejar.

Hay que tener en cuenta que los objetos se “tocan”, pero no están pegados, es decir, que aún se trasladan, rotan, etc. independientemente. Para enlazar dos objetos no es necesario que se toquen.

Imagen 6.3.cp.3.3: Los tres componentes de nuestro objeto final

2. Enlazar los objetos.

2.1. Ahora vamos a enlazar (linkar) los tres elementos en uno solo para moverlos, rotarlos y escalarlos todos a la vez como si de un solo objeto se tratara. Para ello, seleccionamos el prisma, luego el cilindro y por último, (este orden es importante) la caja. Accedemos al menú Tools y ejecutamos Link (o Control + L). Se coloreará el cilindro y el prisma de color azul y la caja en color amarillo.

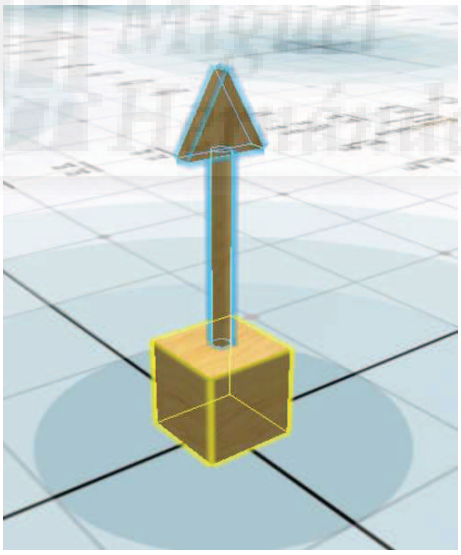


Imagen 6.3.cp.3.4: El objeto raíz en amarillo y todos los demás en azul

Los colores que toman los objetos denotan el objeto raíz y todos los objetos que componen el objeto enlazado y que no son raíz. El objeto raíz será la referencia para el objeto enlazado, de tal forma que cuando se quiera hacer una operación de cambio de posición, los datos introducidos tomarán como referencia el punto central del objeto raíz. También, cuando se quiera hacer una rotación se tomará como pivote ese mismo punto.

3. Utilizar el objeto enlazado.

Vamos a comprobar que los tres objetos que antes eran independientes, ahora están unidos. Para hacer esto vamos a mover y girar el conjunto, el nuevo objeto.

3.1. Seleccionar el objeto. Aparecerán unas flechas de tres colores. Acercar el cursor hasta que se resalte la flecha verde y hacer clic arrastrando. Observaremos como podemos trasladar todo el conjunto sobre el eje Z, acercando o alejando el objeto enlazado.

3.2. Ahora pulsar la tecla Control para pasar a modo rotación. Observaremos que las flechas se cambian por unos círculos de colores. Hacer clic y arrastrar para girar en cualquier eje el objeto. Esto se puede observar en la imagen que sigue.

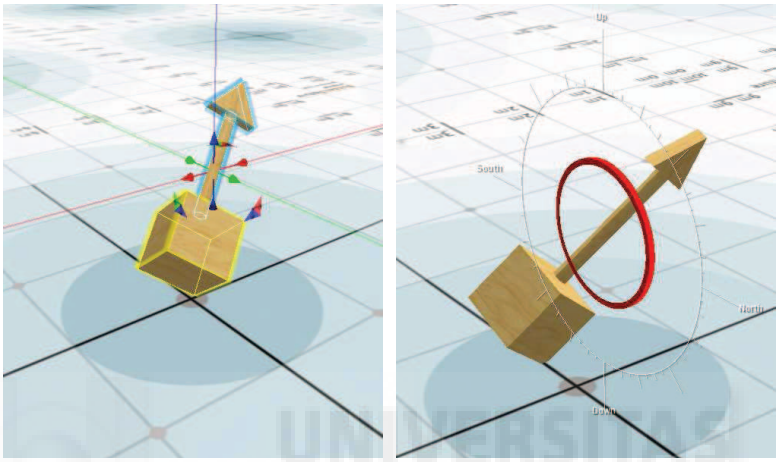


Imagen 6.3.cp.3.5: Operaciones de traslación y rotación con el objeto enlazado

4. Desenlazar uno de los objetos.

El objeto enlazado, como hemos visto, se puede manipular como una unidad. Pero esto no quiere decir que los objetos componentes dejen de tener entidad propia. Para demostrar esto, vamos a desenlazar un objeto, en este caso el objeto raíz y comprobar el resultado.

4.1. Seleccionar el objeto enlazado.

4.2. Hacemos botón derecho y en el menú contextual, elegimos Edit. Aparece el panel de la imagen siguiente.

4.3. Aquí debemos pulsar la opción “Edit linked parts” o “Editar partes ligadas” que señala el rectángulo en rojo.

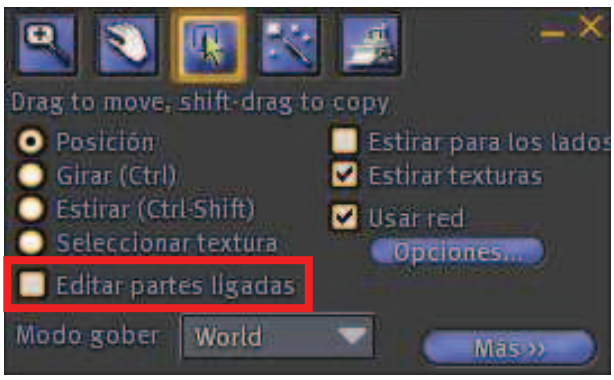


Imagen 6.3.cp.3.6: Editar partes ligadas

Esto sirve para que Second Life interprete que las operaciones que vamos a realizar seguidamente se refieren a elementos individuales del objeto enlazado y no al objeto entero.

4.4. Ahora podemos seleccionar cualquier objeto, por ejemplo, la caja.

4.5. Vamos al menú Tools > Unlink (o Control + Shift + L) y la caja ya no estará enlaza con el resto del grupo.

Esto lo podemos comprobar moviendo la caja y el objeto enlazado independientemente como muestra la figura que sigue.

Imagen 6.3.cp.3.7: Desenlazado de un objeto del conjunto

Debemos tener en cuenta que el nuevo objeto enlazado necesita un objeto de referencia o raíz. Si nos fijamos en la figura observaremos que es el cilindro, ya que era el objeto que seleccionamos en segundo lugar cuando linkamos los tres objetos en uno solo. Por lo tanto, tenemos que tener en cuenta que es muy importante el orden de selección de los objetos al enlazar, ya que de algún modo, estamos estableciendo una especie de jerarquía, de dependencia de unos objetos con respecto a otros.

Conclusiones:

El enlace entre prims es muy importante en el modelado de objetos ya que introduce la posibilidad del diseño modular. Esto es, crear un modelo por partes o módulos de tal manera que se pueda repartir el trabajo que supone un modelo complejo en partes distintas. Pongamos por ejemplo el modelado de un vehículo. En este caso, podemos dedicarnos a realizar el chasis, el motor, la carrocería,… en momentos distintos e incluso pueden intervenir otros usuarios y realizar cada uno una tarea. Además al llevar implícita una jerarquía de prims, siempre tenemos definido un punto de referencia único para un modelo por muy complejo que este sea.

Caso práctico 3.4: Modelar objetos con modificadores del prim caja: un pebetero.

Objetivo: Crear un elemento arquitectónico utilizando solamente cajas y modificadores.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Preparar el trabajo.
2. Creación del pie.
3. Creación de la columna.
4. Creación del pebetero.
5. Terminar el trabajo.

1. Preparar el trabajo.

El trabajo que queremos hacer es, como muestra la figura, un pebetero que utilizaremos para contener fuego que estudiemos en el tema de sistemas de partículas. Hemos realizado todo el pebetero a base de cajas modificadas y luego realizaremos cambios para tener algunas versiones. En la imagen inferior podemos observar un pebetero realizado con cajas y un cilindro.

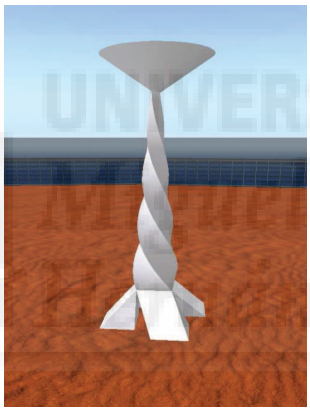


Imagen 6.3.cp.4.1: Imagen de referencia

Comenzaremos creando una caja que coloreamos de blanco y que posteriormente se convertirá en la parte central del pebetero, la columna. Nos servirá también como elemento para orientarnos, como elemento referente para los demás, ya que todos se crearán a partir de este.

1.1. Crear la caja. En un sandbox, hacemos clic en el suelo y con el botón derecho elegimos “Crear”. Tomada la opción caja, hacemos clic en el suelo con el botón izquierdo. Aparece una caja de madera.

1.2. Seleccionamos la caja y si no se observan pestañas, pulsamos el botón “más”. Elegimos la pestaña “Textura”. Lo que pretendemos es quitarle la textura de madera con fines didácticos, ya que en blanco se pueden observar mejor algunas modificaciones.

1.3. Hacemos clic sobre la miniatura de “Textura”. Aparecerá la ventana de textura y elegimos el botón “En blanco”. En la versión en inglés pondrá “Blank”. Esto significa que no utilizamos ninguna textura. Como el color blanco es el que se utiliza por defecto, nuestra caja será ya de este color. Si queremos que sea de cualquier otro color, haremos clic en la miniatura “Color” y podemos elegirlo directamente de unas muestras o mediante código RGB. El resultado se muestra en la siguiente imagen.

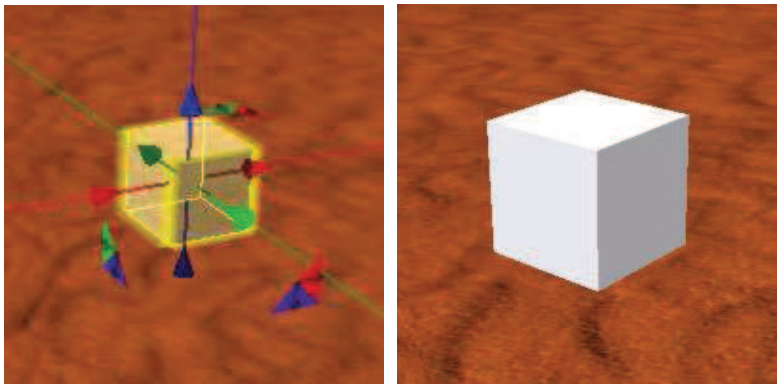


Imagen 6.3.cp.4.2: La caja que nos servirá de referente

2. Creación del pie.

El pie lo realizaremos como una base compuesta por cuatro cajas modificadas.

2.1. Realizar una cruz de cajas. Vamos a crear cuatro cajas alrededor de la caja inicial de tal forma que desde arriba parecerá una cruz. Esto lo vamos a realizar utilizando los parámetros “Copy Selection” y “Center Copy” que están disponibles donde muestra la imagen siguiente.

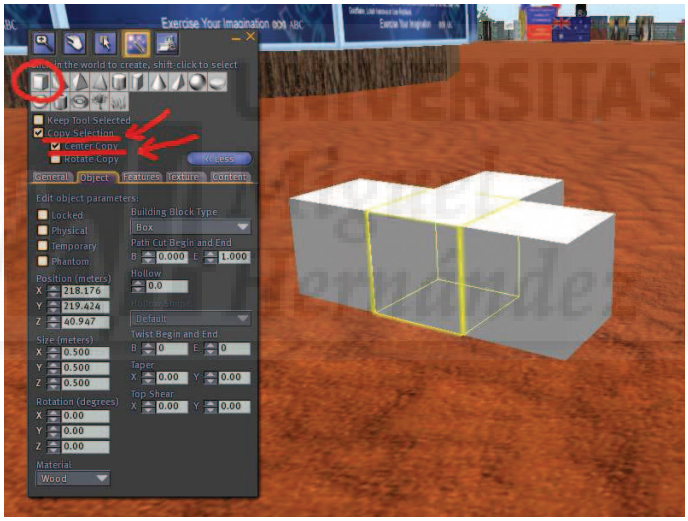


Imagen 6.3.cp.4.3: Crear cajas anexas a la caja referente

Para ello, seleccionamos la caja de referencia y con la ventana de creación de objetos tal como muestra la imagen, hacemos clic sobre la caja que queremos quede pegada a la nueva caja. Luego movemos la cámara, para lo cual es conveniente hacer uso de las teclas Shift+Alt. Repetiremos esta operación para las cuatro caras laterales de la caja.

2.2. Aplicamos modificadores a las cajas.

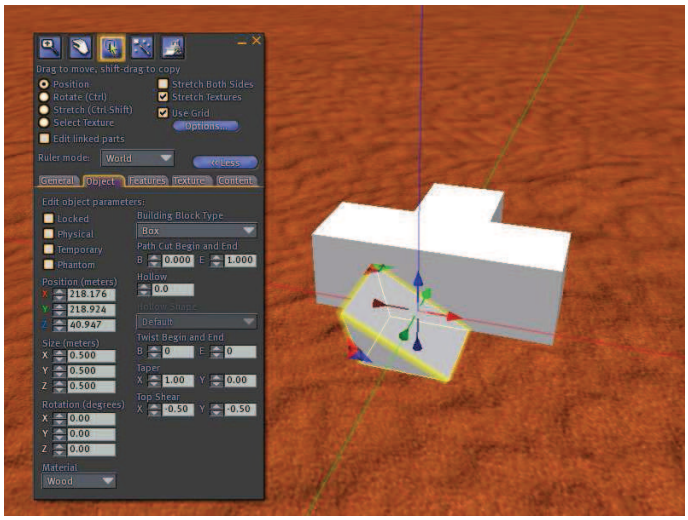


Imagen 6.3.cp.4.4: Aplicar los modificadores Taper y Top Shear

Aplicamos los siguientes modificadores que siguen a la caja más cercana: Taper: X: 1.00; Y: 0.0. Top shear: X: -0.5; Y:-0.5.

A continuación repetimos los mismos modificadores a las tres cajas restantes.

2.3. También vamos a colorear las cajas (aunque este trabajo lo podíamos haber realizado antes) para que se vean mejor y así minimizar los posibles errores. Para ello seguimos el paso 1.3 de esta misma práctica.

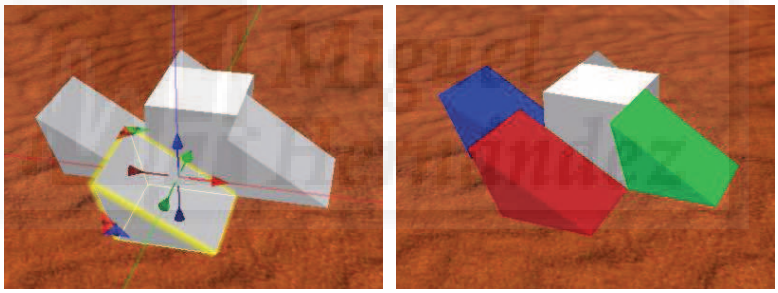


Imagen 6.3.cp.4.5: Aplicar los modificadores a las cuatros cajas y colorearlas

2.4. Aplicamos rotaciones a las cajas tal como se muestra en la imagen que sigue.

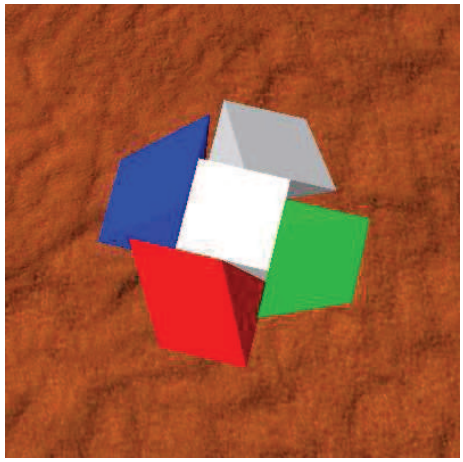


Imagen 6.3.cp.4.6: Vista desde arriba de la rotación de las cajas laterales

Caja Roja: Rotación: 0; 0; 270;

Caja Azul: Rotación: 0; 0; 180;

Caja Gris: Rotación: 0; 0; 90;

Caja Verde: Rotación: 0; 0; 0; es la única que se deja como está desde el principio.

2.5. Mover las cajas para hacer coincidir las aristas superiores de las cajas laterales con la arista que corresponda de la caja de referencia. De esta forma terminaremos la parte inferior de nuestro trabajo. Debe quedar como muestra la imagen.

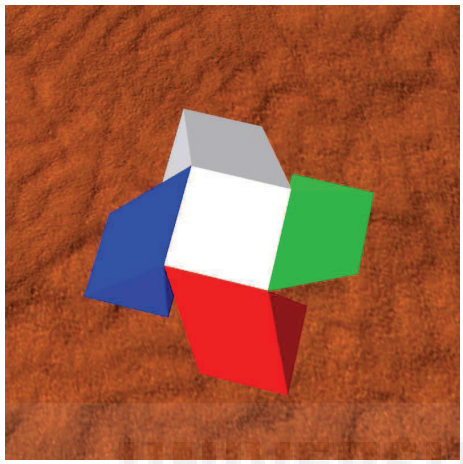


Imagen 6.3.cp.4.7: Pie terminado

3. Hacer la columna

3.1. Crear la columna central. Para ello, vamos a cambiar las dimensiones de la caja de referencia, de tal forma que mida 2,5 metros de alto. Por tanto, la seleccionamos y en su tamaños, ponemos: X: 0.5; Y: 0.5; Z: 2.5;

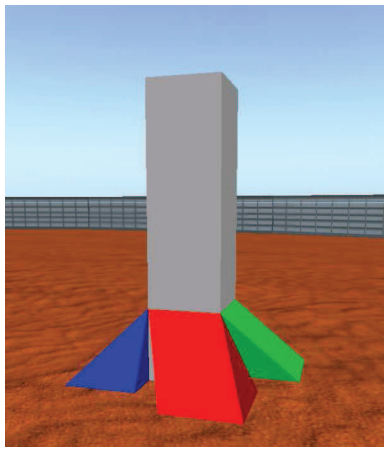


Imagen 6.3.cp.4.8: Cambio de altura de la caja de referencia

3.2. Movemos la caja de referencia hacia arriba. Cuando cambiamos las dimensiones de la columna, parte se queda dentro de la “arena”. Seleccionamos la flecha azul y arrastramos hacia arriba para sacarlo de la “arena” de tal manera que coincidan las aristas como antes, pero en este caso, las aristas inferiores. Esto lo haremos a simple vista aunque podríamos calcular su posición con total precisión, no creemos que sea necesario.

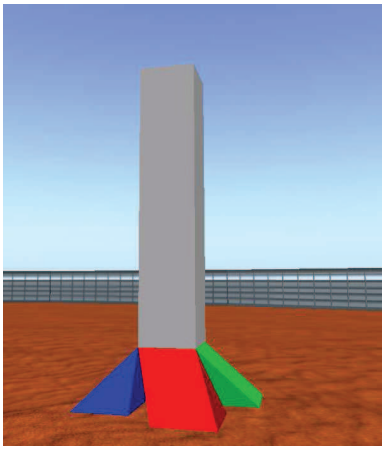


Imagen 6.3.cp.4.9: Movimiento de la columna a su posición correcta

3.3. Ahora lo que haremos es aplicar modificadores con dos objetivos. Afilar la columna de tal manera que la parte de arriba sea más estrecha y luego retorcerla sobre sí misma para crear un modelo un poco más trabajado como se observa en la imagen siguiente. En la pestaña objeto cambiar los valores de los siguientes modificadores:

Twist begin and end: B: 180; E:-180; Esto retuerce 6 veces la caja.

Taper: X: 0.8; Y: 0.8; Esto afila la parte superior.

Imagen 6.3.cp.4.10: La columna central terminada

4. Hacer el pebetero.

4.1. Realizamos una copia de la columna que servirá de pebetero. Para que quede en la misma vertical lo copiaremos por el método de arrastrar. Seleccionamos la columna y con la tecla “Shift” pulsada, arrastramos la flecha azul hacia arriba. Esto hará que tengamos una copia de la columna pero situada más arriba.

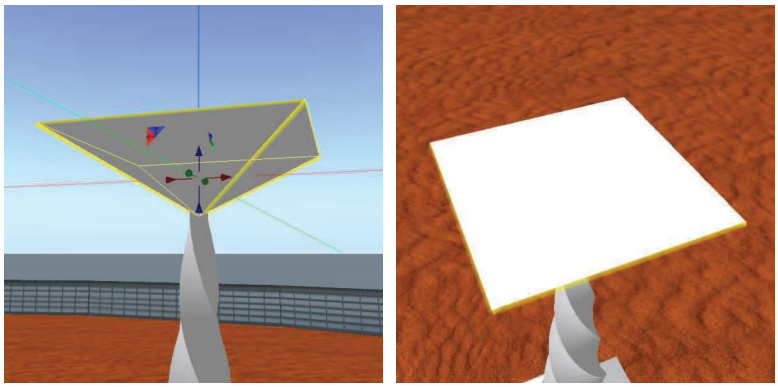


Imagen 6.3.cp.4.11: la caja que sirve de pebetero desde dos puntos de vista

4.2. Modificamos el valor de sus parámetros de la siguiente forma:

Size: 1.5; 1.5; 0.5;

Twist begin and end: B: 0; E: 0;

Taper: X: -1; Y: -1;

4.3. Ahora lo que vamos a realizar es un agujero de tal manera que el pebetero tenga gran parte de su cuerpo hueco. Para ello ponemos el valor de su modificador “Hollow” a 90, tal como muestra la imagen de abajo.

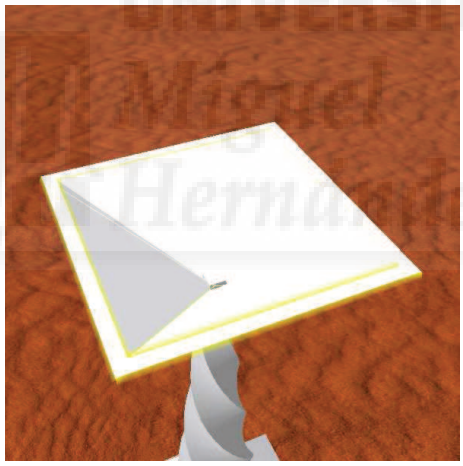


Imagen 6.3.cp.4.12: Aplicación del modificador Hollow

4.4. Recolocar verticalmente. Debido a los cambios en la geometría del pebetero, probablemente necesitemos moverlo verticalmente para que se sitúe justo sobre la columna.

5. Variaciones

Abajo podemos ver una serie de variantes del pebetero terminado.

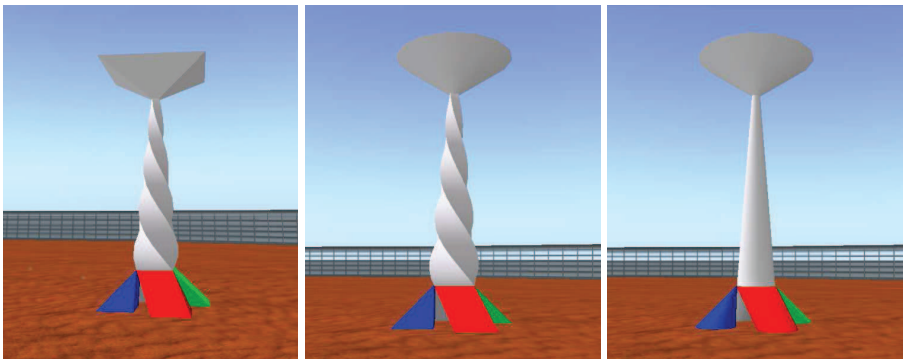


Imagen 6.3.cp.4.13: Variante del pebetero

Estas variaciones las obtenemos al modificar el parámetro “Building Block Type”, que como sabemos varía el prim utilizado como base para el modelo. En el de la izquierda se ha dejado el bloque tipo caja, en el centro hemos variado a un cilindro y a la derecha, hemos cambiado todas las cajas por cilindros.

6. Terminar el trabajo.

6.1. Ponerlo todo de blanco. Mirar el proceso descrito en el paso 1.3.

6.2. Unirlo todo. Seleccionamos todos los elementos que componen nuestro pebetero haciendo clic sobre cada uno de ellos mientras pulsamos la tecla Shift. Es muy importante el último elemento seleccionado ya que será este el que se convierta en referente de todo el grupo. Debemos seleccionar las 4 cajas que componen el pie, la columna central y la parte superior en último lugar.

6.3. En el menú Tools pulsamos “Link” (o Ctrl + L).

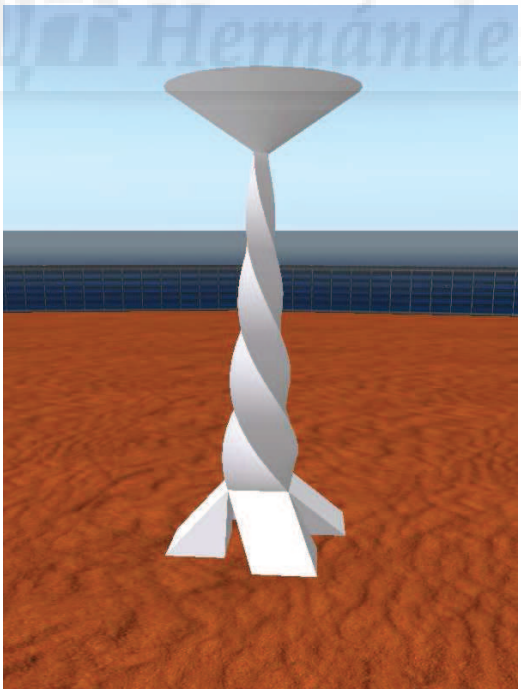


Imagen 6.3.cp.4.14: El pebetero terminado con todos los elementos

6.3. Guardarlo. Para ello pulsaremos la opción Take del menú que aparece cuando hacemos clic sobre él con el botón derecho. Esto hará que pase a nuestro Inventario.

Conclusiones:

Este caso práctico es importante ya que muchos modelos son variaciones del prim caja, sobre todo elementos artificiales como muebles, edificios, etc. De todas formas, solo pretendemos ver algunos modificadores de los muchos que se pueden aplicar estos prims.

Caso práctico 3.5: Modelar objetos con modificadores de esfera: el escarabajo sagrado.

Objetivo: Realizar una escultura utilizando solamente esferas para estudiar sus modificadores.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Estudio del trabajo.
2. Crear el cuerpo.
3. Crear las patas.
4. Crear las alas.
5. Terminar el objeto.

1. Estudio del trabajo.

El trabajo que vamos a realizar está basado en la idea que representa la foto de abajo: un escarabajo sagrado del antiguo Egipto.

Vamos a utilizar solamente esferas modificadas y por lo tanto, nos vamos a tomar muchas "licencias", por lo que el resultado final variará mucho con respecto al "original".

Imagen 6.3.cp.5.1: Foto de referencia

Lo que vamos a realizar tendrá tres partes separadas: cuerpo, patas y alas. Utilizaremos un código de colores para poder distinguirlos bien y posteriormente lo utilizaremos como base para las prácticas de aplicación de texturas, es lo que se conoce como premateriales. Tenemos que tener en cuenta que el modelo texturizado ganará mucho en apariencia y vistosidad, y sobre todo es una forma de crear realismo, por lo que no tenemos por que pensar que nuestro modelo es excesivamente simple. Si es necesario crearemos o modificaremos alguno de sus componentes para que tenga una presencia más espectacular.

En la imagen que sigue se puede observar el trabajo terminado que es el resultado de seguir paso a paso este caso práctico. Se puede visualizar claramente que es una copia simple y esquemática pero que cumple con todos los elementos. Las alas es una de las comparaciones más evidentes pero esto lo podríamos solucionar haciendo copias de las alas creadas y modificándolas en escala podríamos conseguir un modelo más sofisticado.

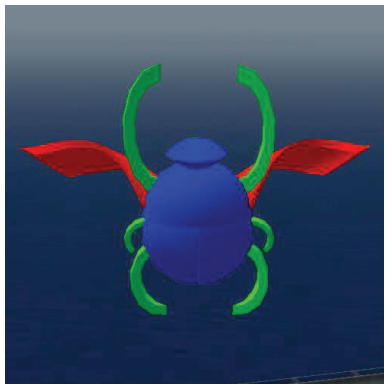


Imagen 6.3.cp.5.2: El trabajo terminado

Las herramientas y procedimientos que utilizaremos son las imprescindibles del modelado. Usaremos la copia de objetos con “Shift” y arrastrando las flechas. También tendremos en cuenta que si nos equivocamos, podemos pulsar CTRL + Z para deshacer cambios. Un buen uso adecuado de la cámara (Ctrl + Alt + botón izquierdo del ratón) y el zoom (con la rueda central del ratón) es imprescindible para ver los detalles y trabajar cómodamente.

2. Crear el cuerpo y la cabeza.

2.1. Creamos una esfera y le ponemos un color azul. Sus valores básicos son:

Tamaño: 0,5; 0,5; 0,5, que son los valores por defecto.

Rotación: 0; 90; 0, importante para seguir el resto de la práctica con precisión.

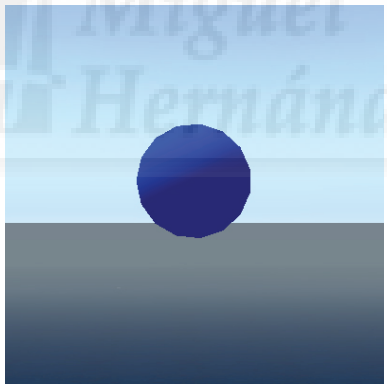


Imagen 6.3.cp.5.3: La esfera primera

2.2. Copiamos la bola azul 2 veces hacia la derecha y las coloreamos de rojo y verde. Esto lo podemos hacer arrastrando con la flecha verde y manteniendo la tecla “Shift” pulsada al mismo tiempo.

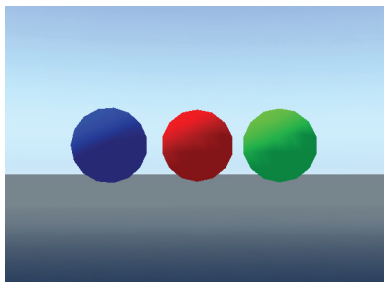


Imagen 6.3.cp.5.4: Las esferas coloreadas

2.3. Copiamos la bola azul 3 veces hacia abajo. Para ello pulsamos sobre la esfera y arrastrando la flecha azul al mismo tiempo que mantenemos la tecla “Shift” pulsada como en el paso precedente. Por lo tanto, debemos tener 4 esferas, 2 alineadas verticalmente y las dos restantes alineadas horizontalmente. Observar la figura siguiente:

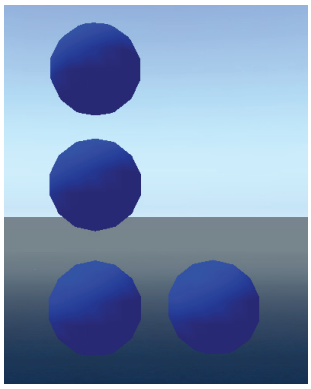


Imagen 6.3.cp.5.5: Las esferas para el cuerpo

2.4. Creación de la cabeza. Seleccionamos la esfera superior. En la pestaña “Object”, modificamos el valor “Dimple” que modificará la esfera como muestra la siguiente imagen: Dimple begin and end: B: 0.700; E: 1.000;

Imagen 6.3.cp.5.6: La cabeza

2.5. Cuerpo, parte superior. Seleccionar la segunda esfera. Modificar los valores:

Tamaño: 0,5; 0,75; 0,75;

Rotación: 180, 0, 90;

Pat Cut begin and end: B: 0.500; E: 1.000;

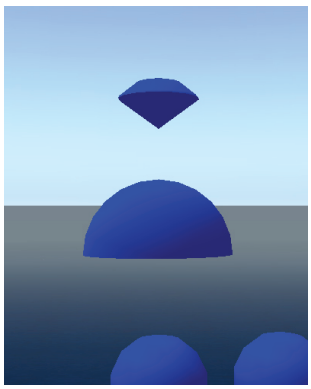


Imagen 6.3.cp.5.7: La partes superior del cuerpo

2.6. Cuerpo, parte izquierda. Seleccionar la tercera esfera. Modificar los valores:

Tamaño: 0,75; 0,50; 0,75;

Rotación: 0; 90; 0;

Pat Cut begin and end: B: 0.500; E: 1.000;

2.7. Cuerpo, parte derecha. Seleccionar la esfera segunda. Modificar los valores:

Tamaño: 0,75; 0,50; 0,750;

Rotación: 0; 90; 0;

Pat Cut begin and end: B: 0.000; E: 0.500;

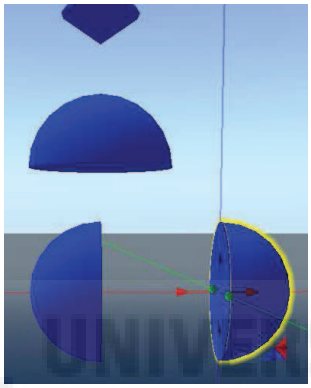


Imagen 6.3.cp.5.8: Las partes inferiores del cuerpo

Utilizar las flechas para modificar la posición interactivamente. Usar solo las flechas azul y roja, nunca la verde, ya que la profundidad debe estar bien controlada. Si queremos ser precisos podemos consultar los valores x, y, z.

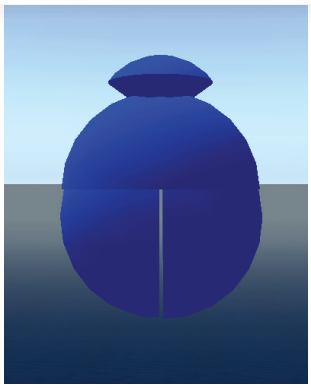


Imagen 6.3.cp.5.9: El cuerpo bien posicionado

3. Crear las patas.

3.1. Patas inferiores. Para ello seleccionamos la esfera verde. Modificar los valores:

Tamaño: 0,6; 0,50; 0,50;

Rotación: 0; 90; 0;

Pat Cut begin and end: B: 0.700; E: 0.800;

Hollow: 75.0

3.2. Hacer una copia de la pata manteniendo la tecla “Shift” y arrastrando las flechas. Rotarla para que queden los valores: 0; 270; 180. Debe quedar simétrica a la anterior. Posicionarlas correctamente con respecto al cuerpo en los tres ejes.

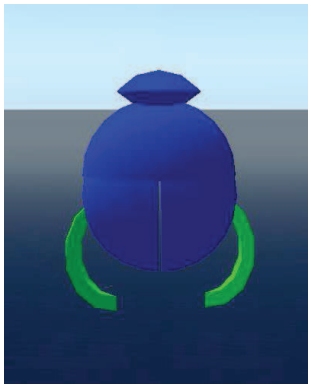


Imagen 6.3.cp.5.10: Las patas traseras

3.3. Patas superiores. Seleccionar la dos patas con la tecla “Shift” y hacer una copia al arrastrar hacia arriba mientras seguimos pulsando la tecla “Shift”. Posicionarlas como las patas delanteras y cambiar únicamente su tamaño a: 0.85; 0,5; 0.5. Este cambio hará que se escalen hacia arriba.

Imagen 6.3.cp.5.11: Las patas delanteras

3.4. Patas intermedias. Repetir la operación anterior y posicionarlas adecuadamente pero modificando los valores siguientes:

Tamaño: 0.3; 0.3; 0.3;

Rotación: 180; 45; 180. Esta última operación, hace que se inclinen 45º en el eje Y.

La otra pata será simétrica, por lo que copiamos la anterior y la posicionamos

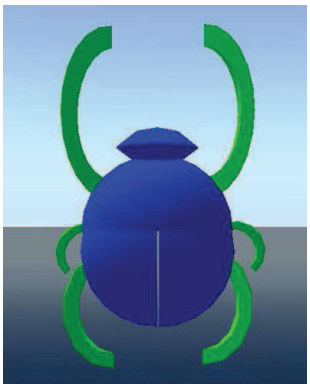


Imagen 6.3.cp.5.12: Todas las patas en su posición correcta

4. Crear las alas.

4.1. Seleccionar la esfera roja y hacer una copia arrastrando la flecha y mientras mantenemos la tecla “Shift” pulsada.

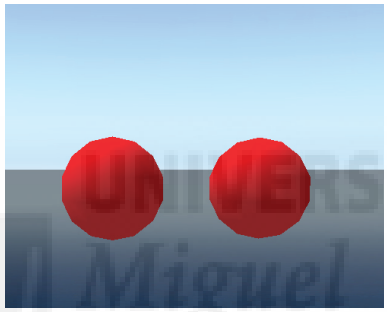


Imagen 6.3.cp.5.13: Las esferas para las alas

Modificaremos las dos esferas a la vez para estudiar su evolución. Al final del proceso, deben quedar los siguientes valores:

Modificadores	Primera esfera	Segunda esfera
Size	1.5, 0,5, 0,5	1.5, 0,5, 0,5
Rotation	180, 330, 0	0, 330, 180
Path Cut Begin and End	B: 0.400 E: 1.000	B: 0.000 E: 0.600
Hollow	85.0	85.0
Twist Begin and End	B: 150 E: 0	B: 0 E:150
Dimple Begin and End	B: 0.150 E: 0.350	B: 0.150 E: 0.350

4.2. Las siguientes imágenes ilustran como evolucionan las “alas” según vamos modificando los valores:

Cambio de tamaño y rotación:

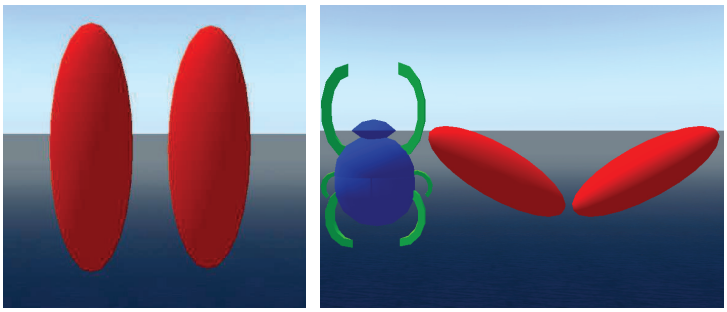


Imagen 6.3.cp.5.14: Cambio de tamaño y rotación

Cambio de Path Cut y Hollow:

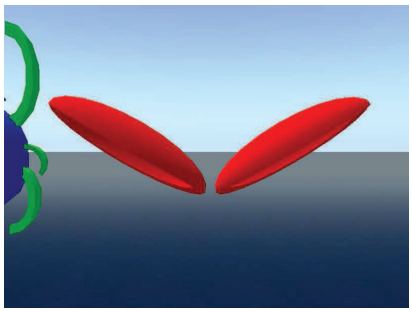


Imagen 6.3.cp.5.15: Modificador Hollow y Path Cut

Cambio del Twist y Dimple:

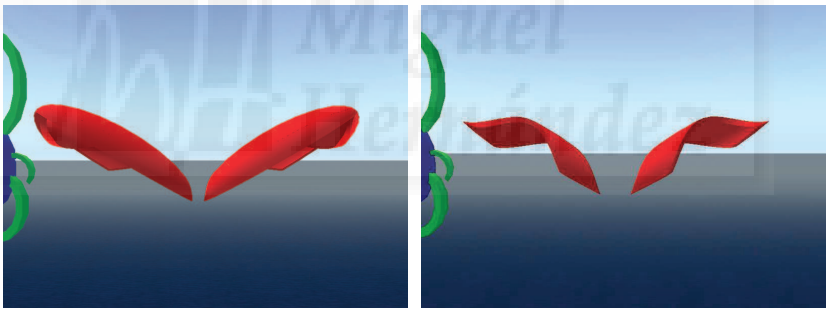


Imagen 6.3.cp.5.16: Modificador Twist y Dimple

4.3. Posicionarlas correctamente para que queden como se muestra la imagen.

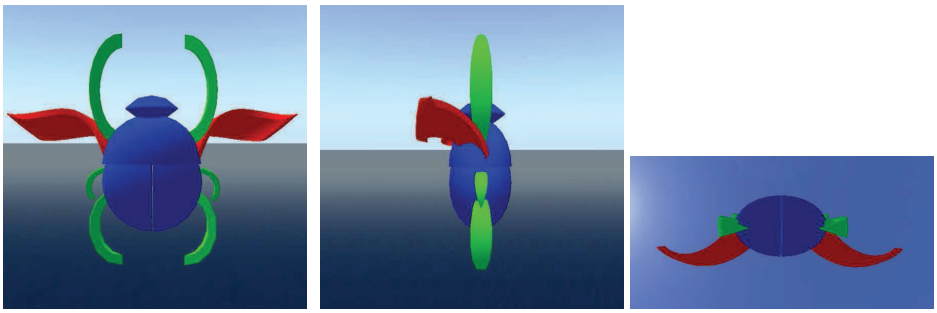


Imagen 6.3.cp.5.17: Posicionamiento de las alas

5. Terminar el objeto.

5.1. Unirlo todo. Seleccionamos todos los elementos que componen nuestro escarabajo haciendo clic sobre cada uno de ellos mientras pulsamos la tecla “Shift”. El elemento que será el último en ser seleccionado y por lo tanto será el llamado objeto raíz será la partes superior del cuerpo.

5.2. Luego en el menú Tools pulsamos Link (o Ctrl + L). Quedará como muestra la imagen que sigue.

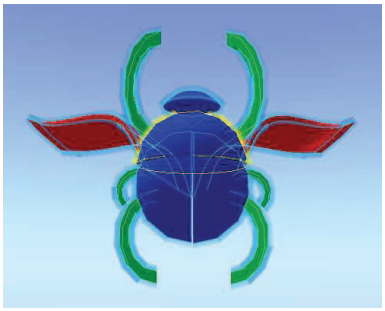


Imagen 6.3.cp.5.18: Unión de todos los elementos que componen el escarabajo

6.3. Guardarlo. Para ello pulsaremos la opción “Take” del menú contextual del objeto que aparece cuando hacemos clic sobre él con el botón derecho. Esto hará que pase a nuestro Inventario.

Conclusión:

Los modificadores que podemos aplicar al prim esfera son propios. Esto quiere decir que no los comparte con ningún otro prim. Quizá el más original sea el “Dimple” que realiza una especie de abolladura o filo según en la semiesfera en que se produzca.

Por otra parte, resulta evidente después del trabajo realizado, que el prim esfera es muy útil y versátil.

Caso práctico 3.6: Modelar objetos con modificadores de toros: un ánfora

Objetivo: Aplicar los modificadores típicos de los prims toro, tubo y anillo.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Estudio del trabajo.
2. Crear el macetero.
3. Crear el ánfora.
4. Terminar el objeto.

1. Estudio del trabajo.

El trabajo que vamos a realizar es un ánfora y un macetero para colocarla. La originalidad de la práctica radica en que la vamos a llevar a cabo utilizando un solo tipo de objeto primitivo: el toro, o como lo llaman en SL, el relieve. A partir de este prim modificaremos su geometría para observar cómo se pueden obtener objetos que en un principio no identificaríamos con el objeto primitivo inicial.

En la imagen que sigue se puede observar el prim toro del que partimos.

Imagen 6.3.cp.6.1: Modelo obtenido con el prim toro

2. Crear el macetero.

Primero crearemos los toros iniciales. En total vamos a utilizar 5. Uno para el “cuenco” y 4 patas. Cómo las patas son todas iguales y lo único que cambia es su colocación y rotación, haremos copias de la primera pata que realicemos.

2.1. Crear los toros iniciales. En un sandbox > botón derecho > Crear > Toro > y pulsar en el suelo. El resultado es un modelo como el que se puede ver en la imagen de arriba.

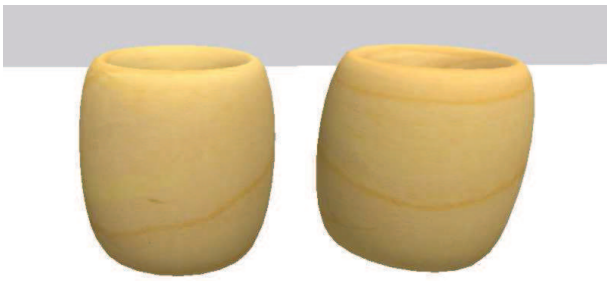


Imagen 6.3.cp.6.2: Los dos prim toros iniciales

2.2. Hacemos clic sobre el primer toro y con la tecla “Shift” pulsada arrastramos la flecha verde hacia la derecha con lo que creamos una copia del toro. El resultado se puede observar en la imagen de arriba.

2.3. Crear el receptáculo. Para ello seleccionamos uno de los dos toros y hacemos botón derecho > Editar.

2.4. Modificamos sus parámetros básicos como sigue:

Tamaño, size: X: 0.35; Y: 0.5; Z: 0.5;

Rotación, rotation: 270, 0, 270

2.5. Aplicamos los siguientes modificadores:

Inclinación, skew: 0.8;

Inicio y fin del corte de perfil, profile cut begin and end: B: 0.6 E: 1.0

Disminuir, taper: X: -1; Y: -1;

Imagen 6.3.cp.6.3: El toro después de las tres primeras modificaciones

Rayo delta, radius: -1;

Revoluciones, revolutions: 4

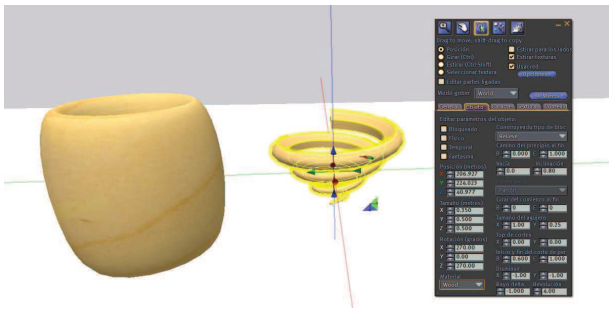


Imagen 6.3.cp.6.4: El cuenco terminado después de aplicar radius y revolutions

2.6. Crear las patas. Seleccionamos el otro toro y pulsamos botón derecho y en el menú la opción “Editar”. Aplicamos los siguientes modificadores:

Camino del principio al fin, path cut begin and end: B: 0.2 E: 0.9

Tamaño del agujero, hollow size: X: 1; Y: 0.1

Inicio y fin del corte de perfil, profile cut begin and end: B: 0.6 E: 1.0

Disminuir, taper: X: -1; Y: 0;

Imagen 6.3.cp.6.5: Creación inicial de las patas

2.7. Modificar los parámetros básicos:

Tamaño, size: X: 0.1; Y: 0.6; Z: 0.6;

Rotación, rotation: X: 0.0; Y: 0.0; Z: 0.0;

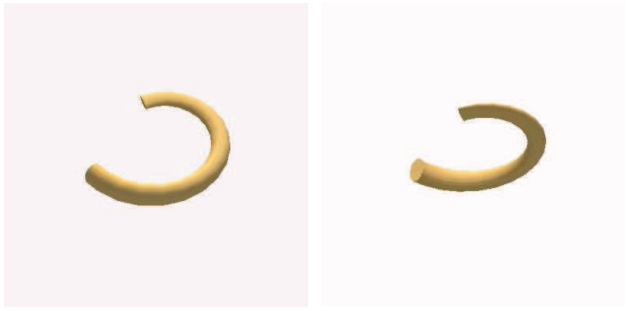


Imagen 6.3.cp.6.6: Cambios en los parámetros básicos: tamaño y rotación

2.8. Hacer tres copias más. Luego las situaremos adecuadamente y colocaremos el cuenco encima como muestran las imágenes siguientes:

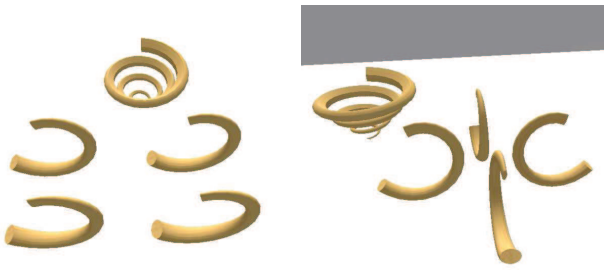


Imagen 6.3.cp.6.7: Copia de las patas y su colocación

Imagen 6.3.cp.6.8: Macetero terminado

3. Crear el ánfora.

3.1. Crearemos un primitivo tipo toro nuevo como muestra la imagen.

3.2. Aplicamos los siguientes modificadores:

Tamaño, size: X: 0.900; Y: 0.500; Z: 0.500;

Tamaño del agujero, hollow size: X: 1.00; Y: 0.50;

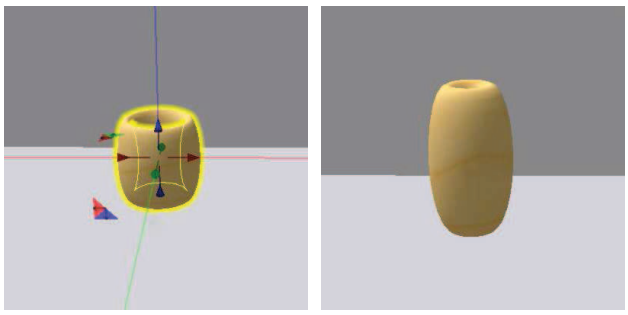


Imagen 6.3.cp.6.9: el objeto primitivo y su modificación como base del ánfora

3.3. Creamos una copia de la base del ánfora para realizar el cuello con la tecla “Shift” pulsada y la flecha azul para que queden alineadas verticalmente.

3.4. Aplicamos los siguientes modificadores:

Tamaño, size: X: 0.65; Y: 0.500; Z: 0.500;

Agujero, hollow: 95.0;

Tamaño del agujero, hollow size: X: 0.95; Y: 0.45;

Inicio y fin de cortes del perfil, profile cut begin and end: B: 0.00; E: 0.5;

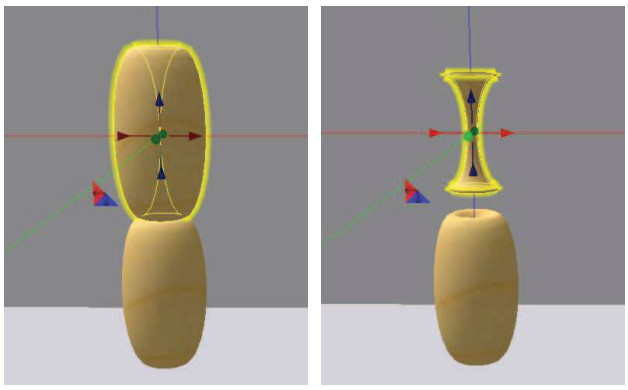


Imagen 6.3.cp.6.10: cuello del ánfora

3.5. Vamos a realizar las asas también con primitivos toros. Para ello con la tecla “Shift” pulsada y la flecha roja creamos una copia de la base para que quede centrada horizontalmente.

3.6. Aplicamos los siguientes modificadores:

Tamaño, size: X: 0.05; Y: 0.500; Z: 0.500;

Camino de inicio y fin, path cut bejín and end: B: 0.45; E: 0.900;

Tamaño del agujero, hollow size: X: 1.00; Y: 0.10;

3.7. Ahora lo que hacemos es rotarla con la tecla “Control” pulsada.

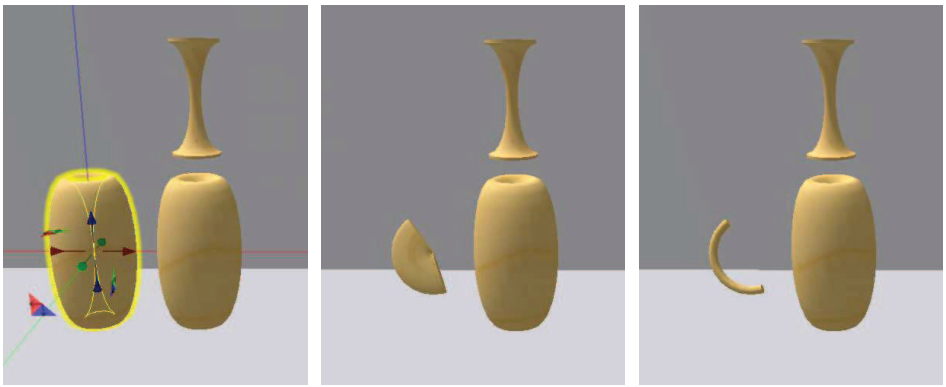


Imagen 6.3.cp.6.11: creación del asa del ánfora

3.8. Vamos a realizar una copia del asa. Para ello con la tecla “Shift” pulsada y la flecha roja arrastramos para que quede centrada horizontalmente.

3.9. La rotamos para que quede en la posición correcta.

3.10. Colocamos el cuello y las asas en la posición correcta y unimos todos los elementos mediante menú herramientas > Unir (o control + L).

En la imagen que sigue podemos observar una ilustración de estos pasos.

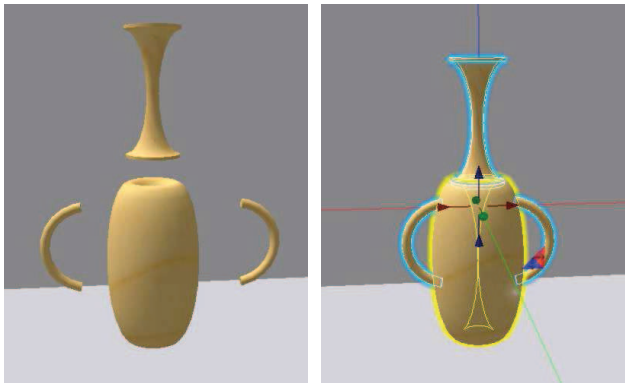


Imagen 6.3.cp.6.12: creación del asa simétrica y unión de todos los elementos

4. Terminar el objeto.

En las imágenes siguientes se puede observar el ánfora sobre su macetero con la textura por defecto y por último, con una textura aplicada.

Imagen 6.3.cp.6.13: presentación del modelo y su versión texturizada

Conclusiones:

El prim toro es una muy buena base para modelos que requieran orificios como pueden ser los vasos, tubos, cañones y toda clase de modelos huecos. Además soportan una serie de modificadores que lo hace uno de los prims más versátiles y utilizados.

Caso práctico 3.7: Realizar Sculpts.

Objetivo: Poder utilizar objetos en SL realizados con objetos editables en 3d Studio MAX.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Crear la caja de referencia.
2. Crear la materia para la escultura.
3. Crear la escultura.
4. Representación de la escultura.
5. Importar la escultura en SL.

1. Crear la caja de referencia.

1.1. Abrimos 3D Studio MAX 8 y creamos una caja de 20 x 20 x 20 unidades.

1.2. La centramos en el origen universal (0, 0,-10) con la herramienta mover.

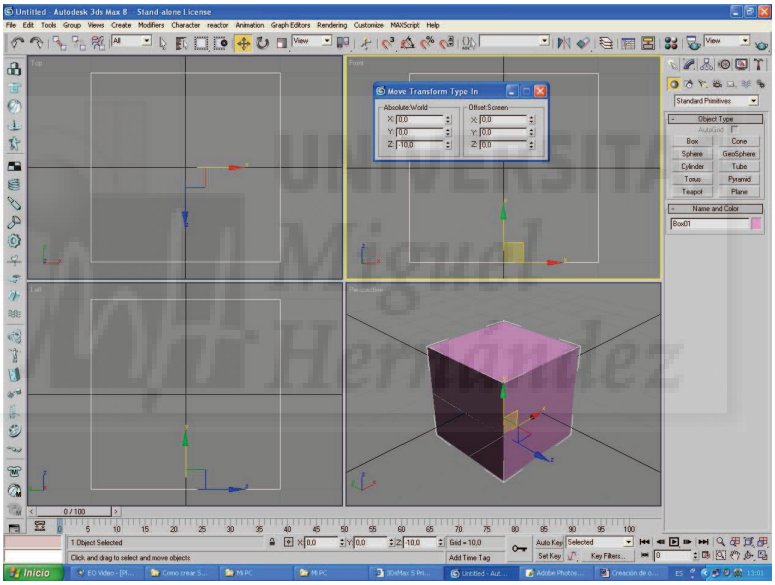


Imagen 6.3.cp.7.1: Caja de referencia

1.3. Abrimos el editor de materiales (M) y creamos un nuevo material llamado “alambre rojo” eligiendo un “slot” cualquiera.

1.4. Lo ponemos en modo “wired”.

1.5. Para los valores ambient y diffuse aplicamos un color rojo puro (255, 0, 0).

1.6. Aplicamos este material a la caja. En la Imagen 6.3.cp.7.2, se puede observar cómo queda la caja con el material creado. Esta caja será una especie de “jaula” que limitará las dimensiones de nuestra “escultura”.

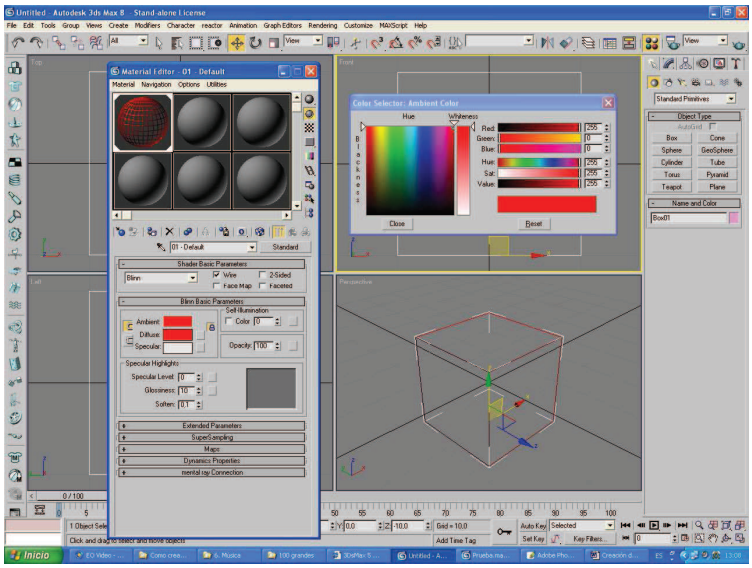


Imagen 6.3.cp.7.2: Material alámbrico aplicado a la caja de referencia

1.7. Ir a las propiedades de la caja haciendo botón derecho > properties.

1.8. Elegimos la opción “Freeze” y quitar “Show Frozen in Gray”.

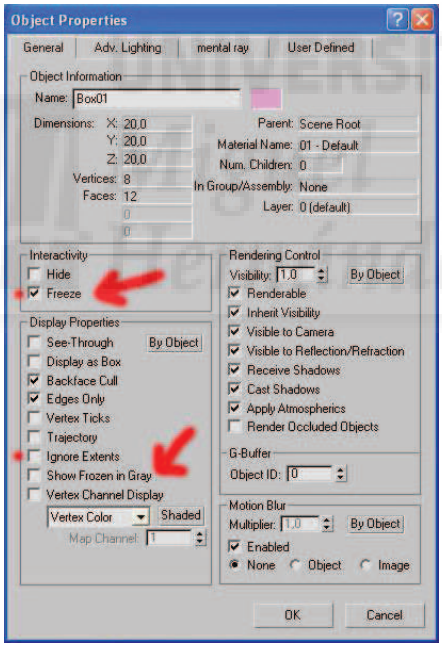


Imagen 6.3.cp.7.3: Caja de diálogo “Object Properties” de la caja

2. Crear la materia para la escultura.

2.1. Creamos un cilindro con las siguientes características:

Radius: 10;
Height: 20;
Segments: 32;
Cap segments: 1;
Sides: 32.

2.2. Ponemos activa la opción “Generate Mapping Cord”. Para que se general coordenadas de mapeo de la superficie a texturizar.

2.3. Centramos el cilindro en el origen universal (0, 0,-10). El valor para el eje Z, viene dado porque el punto de referencia del cilindro es el centro de su tapa inferior, y por tanto para que quede circunscrito en la caja creada anteriormente tenemos que descender este punto a la mitad de su altura total, es decir, -10. Otra alternativa sería centrar su punto de referencia, pero como el proceso es más engorroso preferimos la opción antes explicada.

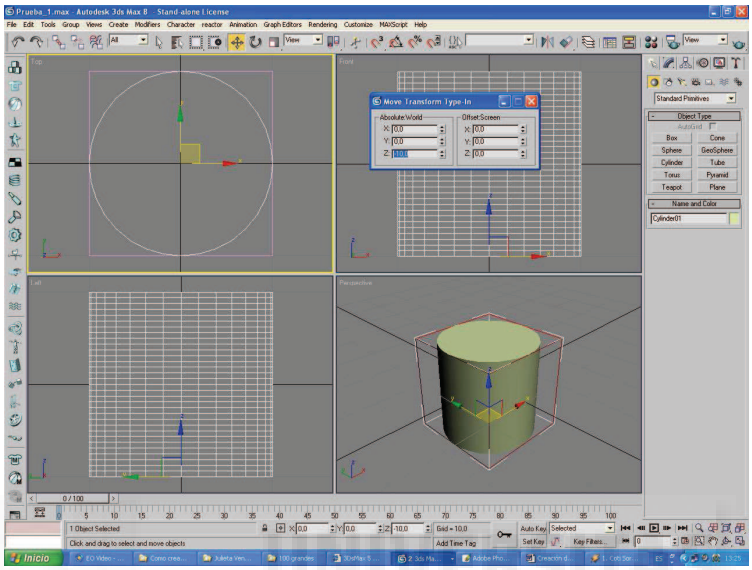


Imagen 6.3.cp.7.4: Material alámbrico aplicado a la caja de referencia

2.4. Seleccionamos el cilindro y le aplicamos el modificador “Unwrap UVW”. Este modificador nos permitirá diseñar la forma en que se “envuelve” la textura en el objeto cilindro.

2.5. Pulsamos el botón “Edit” en las opciones y se abrirá la ventana “Edit uvw”.

2.6. Quitamos la textura de fondo, pulsando sobre el icono que se presenta como un cubo ajedrezado que se encuentra en la barra de la parte superior.

2.7. Seleccionamos todos los vértices con una ventana que los incluyan y estos se pondrán en rojo.

2.8. Seleccionando la herramienta “Mover” (la cruz de flechas) moveremos todos los vértices para que coincidan en la cuadrícula color azul oscuro.

2.9. Hemos concluido esta fase. Cerramos la ventana de edición del “unwrap uvw “.

2.10. Seleccionamos “Unwrap uvw” asegurándonos de no seleccionar “Select Face”.

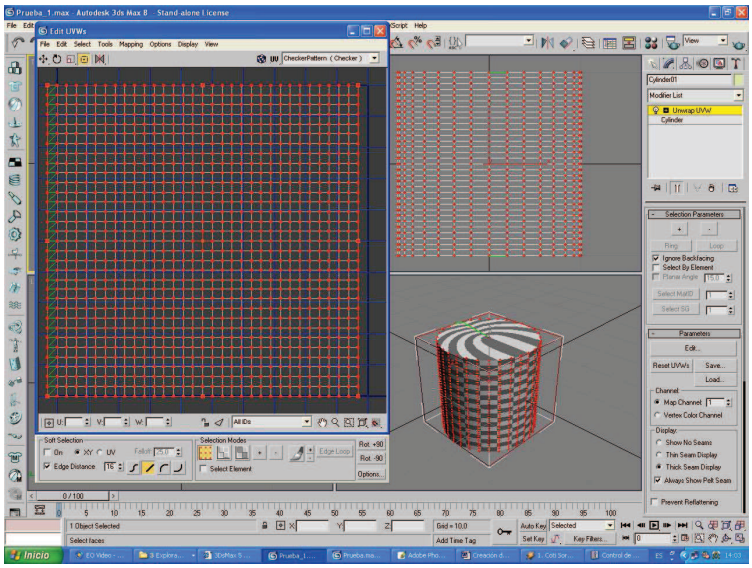


Imagen 6.3.cp.7.5: Edición de Unwrap UVW

2.11. Añadimos el modificador “Edit mesh” y seleccionamos la opción “vertex”.

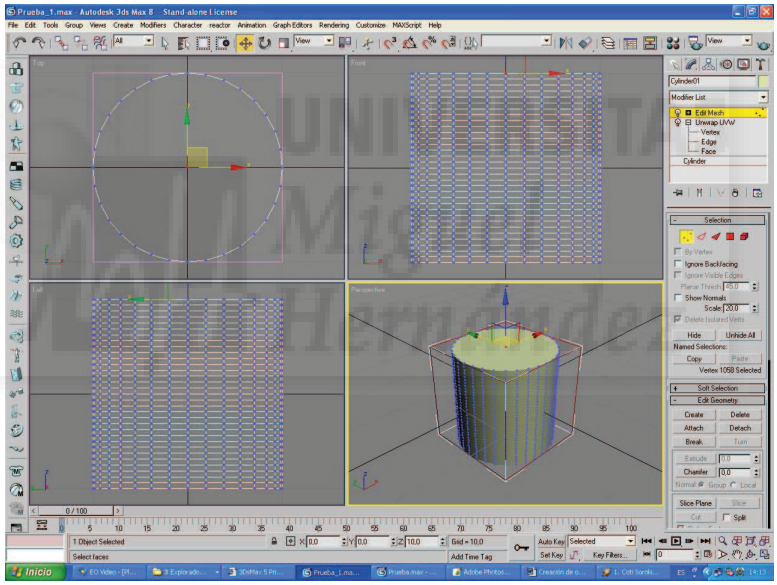


Imagen 6.3.cp.7.6: Edición de vértices en Edit Mesh

2.12. Ahora seleccionamos los dos vértices de la tapa superior e inferior del cilindro y los borramos. Esto lo hacemos para tener solamente vértices en la cara vertical del cilindro y no en sus “tapas”.

2.13. Deseleccionamos la opción “vertex”.

2.14. Añadimos un modificador “UVW Map”.

2.15. Lo vamos a dejar con lo valores que tiene por defecto que deben ser entre otros, que el tipo sea “planar” y tener una longitud y anchura de 20.02.

2.16. Ponemos (en la parte inferior) el “Map Channel” en 2.

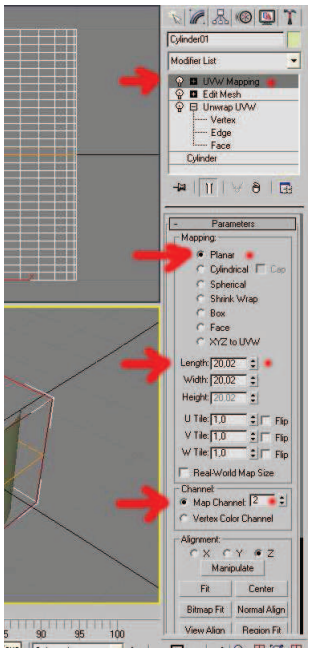


Imagen 6.3.cp.7.7: Edición del modificador UVW Map

2.17. Ahora nos vamos al editor de materiales pulsando la tecla “M” y abrimos la librería de materiales llamada “sculpt.mat”. No debe haber problemas con esta librería ya que es estándar de 3D Studio MAX.

2.18. Finalmente, asignamos el material “RGB Blend” al cilindro.

Observar la imagen siguiente para comprobar que el material asignado no se representa directamente en las cuatro ventanas de edición del modelado, ya que esto haría que la aplicación fuera muy lenta. Ahora el modelo está preparado para empezar la edición de la primitiva “sculpt” para SL.

2.19. Salvamos el fichero como modelo para hacer otras “sculpt” a partir de aquí.

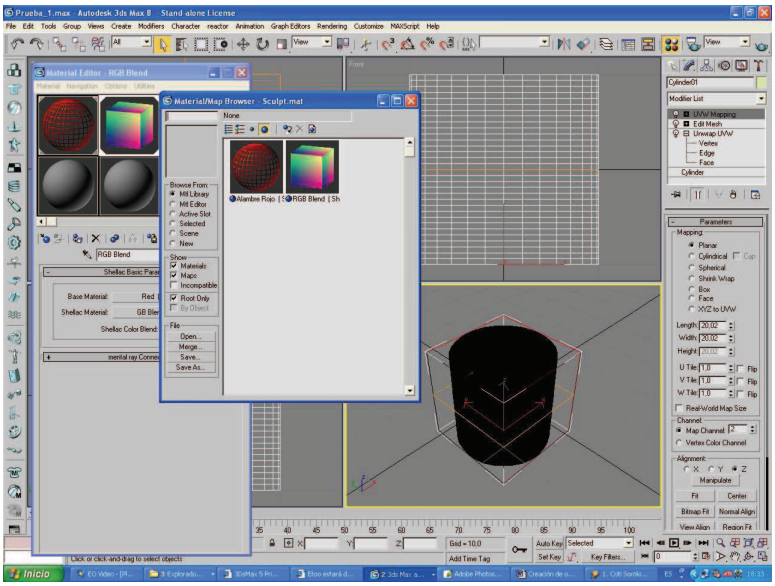


Imagen 6.3.cp.7.8: Material RGB Blend asignado al cilindro

3. Crear la escultura

Nota: no debemos modelar fuera de los límites del cubo alámbrico rojo.

3.1. Acceder al modificador “Edit mesh” y pulsamos el botón “Vertex”. Podemos modelar las mallas modificando las entidades que deseemos: vértices, caras, aristas, o polígonos. Nosotros hemos empleado los vértices porque su edición se adapta mejor al objeto que vamos a modelar.

3.2. Editamos la escultura. En este caso hemos utilizado sobre todo las herramientas de selección para escoger unos vértices y otros no y seguidamente hemos aplicado la herramienta escala no uniforme para mover los vértices en el eje X, creando así el jarrón que se puede apreciar en la imagen 6.3.cp.7.9.

Podemos utilizar otros modificadores y técnicas. Por ejemplo, normalmente, al modelar la malla, se suele utilizar el modificador “MeshSmooth” para suavizarla.

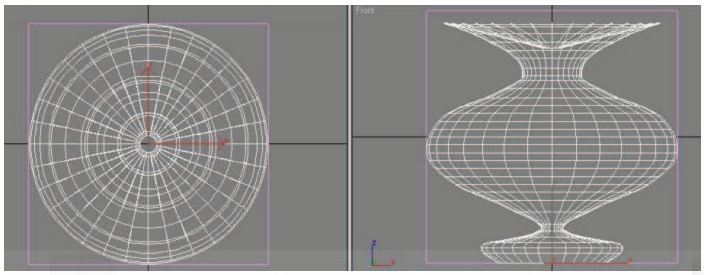


Imagen 6.3.cp.7.9: Vista superior y frontal en 3D Studio MAX del modelo creado

4. Representación de la escultura.

Vamos a utilizar un método para representar el objeto creado muy poco frecuente. Ya que la única manera de introducir un objeto tipo malla en Second Life es mediante una imagen, la representación del objeto lo haremos a una imagen bidimensional, que en realidad contiene un código de colores. Second Life puede interpretar este código y traducirlo a puntos tridimensionales, y por ello, rehacer el objeto. Es decir, que el archivo producido por este método es representado por Second Life (y otras aplicaciones) como un objeto 3D donde la profundidad de los vértices viene dada por su color. Es lo que se denomina un “Sculp”.

4.1. Con la escultura seleccionada, hacemos: menú Rendering > render to texture.

4.2. Aparece una nueva ventana, subimos hacia arriba y pulsamos el botón “Add”.

4.3. En la ventana que aparece y que se llama “Add Texture Elements...”, elegimos “CompleteMap” y “Add elements”. Esto hará que la ventana se cierre.

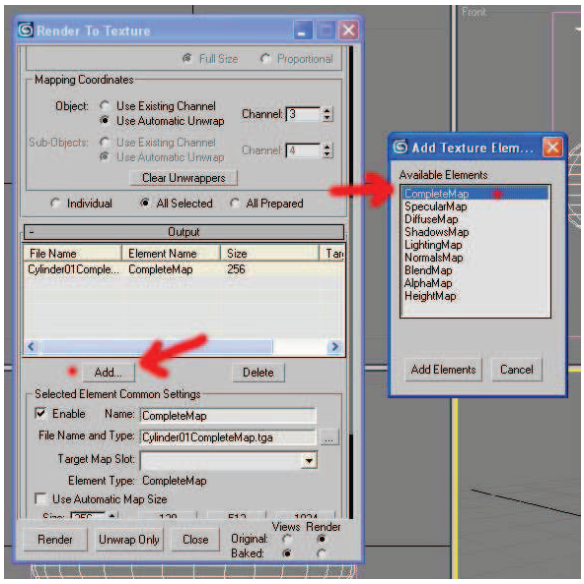


Imagen 6.3.cp.7.10: Ventana de "Render to Texture"

4.4. De vuelta a la ventana anterior, ponemos "Channel" a 1, "padding" a 0 y el tamaño a 64 x 64 (este tamaño es más que suficiente).

4.5. Pulsamos "File name and type" para elegir el formato del fichero de salida. Este formato debe ser TGA para que podamos luego exportarlo a Second Life.

4.6. Pulsamos el botón "Render" de la esquina inferior izquierda para que la representación tenga lugar con los parámetros indicados.

4.7. Aparecerá una ventana. Respondemos eligiendo el objeto y pulsando "Continue".

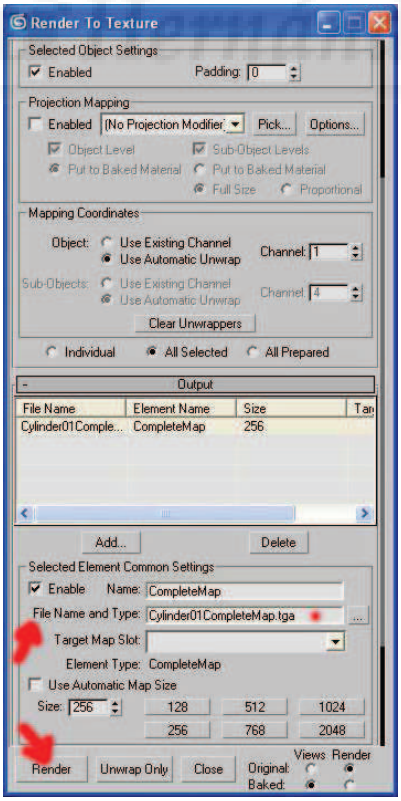


Imagen 6.3.cp.7.11: Formato de salida y render

Obtendremos el archivo que se aprecia en la imagen 6.3.cp.7.12 y ahora ya podemos cerrar 3D Studio MAX y disponernos para exportarlo a SL.

Imagen 6.3.cp.7.12: Representación de la escultura

5. Importar la escultura en SL.

5.1. Abrimos Second Life y buscamos un sitio donde nos permitan crear objetos.

5.2. Pulsamos con el botón derecho sobre la tierra y en el menú que aparece elegimos la opción “Create”.

Imagen 6.3.cp.7.13: Creación de una caja

Imagen 6.3.cp.7.14: Conversión a Sculpt

5.3. Luego elegimos el objeto caja (en realidad lo mismo no da, ya que después lo modificaremos) y hacemos clic con la varita mágica sobre el suelo.

5.4. Pulsamos sobre el botón “More” y seguidamente en la ficha “Object”.

Imagen 6.3.cp.7.15: Textura Sculpt y el objeto 3D resultado

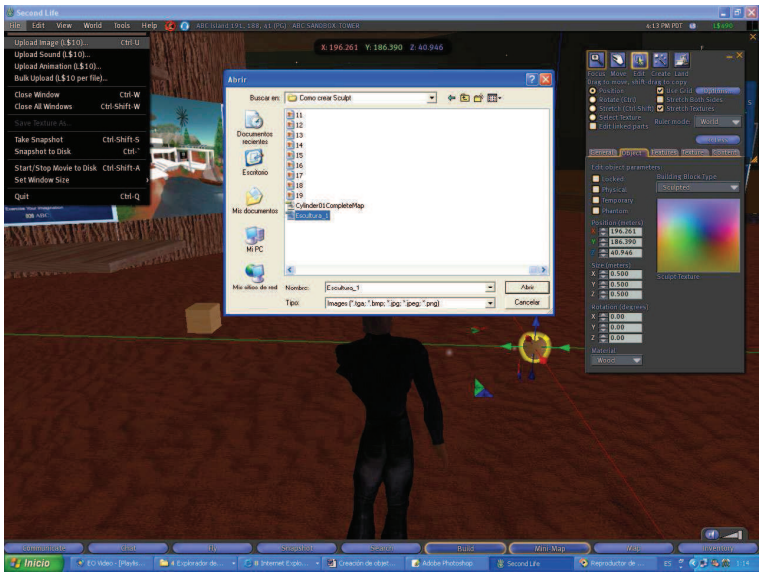


Imagen 6.3.cp.7.16: Importar imagen

5.5. En la opción “Building Block type” elegimos “Sculpted”. La caja se convertirá en una manzana. Esto no es un error, es debido a que es la forma inicial de todo Sculpt.

5.6. Ahora necesitamos importar la imagen generada en 3D Studio MAX 8. Cada imagen que introducimos en SL cuesta 10 Linden Dollars, por lo que tenemos que tener una cuenta con algunos L$ para terminar el caso práctico.

5.7. En el menú archivo > Unload image (Ctrl + u). Aparece la ventana “Abrir” donde podemos buscar la imagen en el disco duro. La imagen importada se almacena en el inventario, por lo tanto tenemos que buscarla allí.

5.8. En la ventana que aparece, le damos el nombre y descripción que queramos.

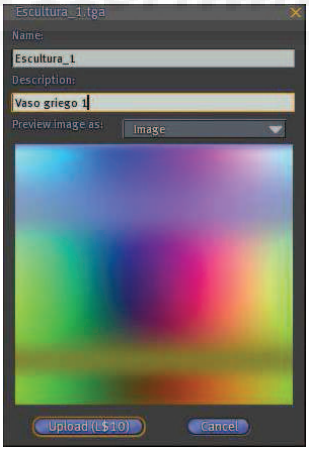


Imagen 6.3.cp.7.17: Importar imagen

5.9. Hacemos clic sobre la textura aplicada al objeto y seleccionamos del Inventario.

Imagen 6.3.cp.7.18: Cambiar la imagen del Sculpt

5.10. Entonces el objeto sculpt se modelará como aparece en la imagen de abajo.

Imagen 6.3.cp.7.19: Nuestra escultura en Second Life

5.11. En este caso la hemos texturizado con una textura de las que aporta Second Life llamada “pinktoile” del grupo “fabric”.

Imagen 6.3.cp.7.20: Nuestra escultura en Second Life con textura

Conclusiones:

Como hemos podido comprobar, realizar “sculpts” es un proceso bastante difícil y engorroso sobre todo por que es un proceso muy largo, con muchos pasos. La parte positiva es que algunos de los puntos que hemos llevado a cabo no hace falta repetirlos, ya que nos sirven para otras esculturas. Como conclusión podemos añadir, que es una buena solución cuando queremos objetos que no podamos realizar con las herramientas propias de Second Life y sobre todo si queremos utilizar modelos de mallas.

Unidad 4: Superficies de objetos.

Introducción teórica:

1. Concepto de superficie de los objetos.
2. Características de las superficies.
3. Canales alpha.
4. Reflejos.
5. Texturas animadas por transformación.
6. Texturas animadas por intercambio.

Casos prácticos:

Caso práctico 4.1: Aplicación de superficies con diversas características.

Caso práctico 4.2: Materiales con canales alpha

Caso práctico 4.3: Texturas animadas

1. Concepto de superficie de los objetos.

La superficie de los objetos es un elemento importantísimo, ya que es lo que realmente vemos de ellos. Cuando modelamos un objeto, en realidad, estamos construyendo algo vacío, hueco y sin masa. En realidad es una superficie, de modo que solo vemos un lado.

Por ejemplo, supongamos que creamos una esfera. Debemos ser conscientes que esa esfera no tiene masa internamente, lo que el software guarda, es una serie de puntos, que forman una figura geométrica. Es como si en vez de tener una naranja, tuviéramos la cáscara 3D, creada a base de polígonos en 2D, que debido a la posición relativa de esos polígonos unos con otros, envolvieran el objeto, en este caso la naranja.

El problema, es que aunque tengamos la forma correcta (una esfera para la naranja), eso no es suficiente para que parezca una naranja, hace falta una superficie que identifiquemos con una naranja. Por ejemplo, la misma forma que la naranja tenemos para el planeta tierra. Sabremos que una esfera es una naranja porque tendrá un color, una rugosidad, un brillo, etc., que serán distintos a la superficie de la tierra o a la de la luna.

Imagen 6.4.1: El mismo modelo con distinta textura

En la figura precedente, el modelo es el mismo, pero con superficies distintas. Es evidente que lo más visible de una superficie es la textura, pero también cambian otras variables que estudiaremos más adelante, por ejemplo, en la naranja se aplican unas rugosidades para que su imagen sea más real. En la figura de abajo, se aplica la misma textura a distintos objetos con el fin de visualizar que las texturas y los modelos están íntimamente relacionadas, ya que esta textura solo tiene sentido para el balón de baloncesto.

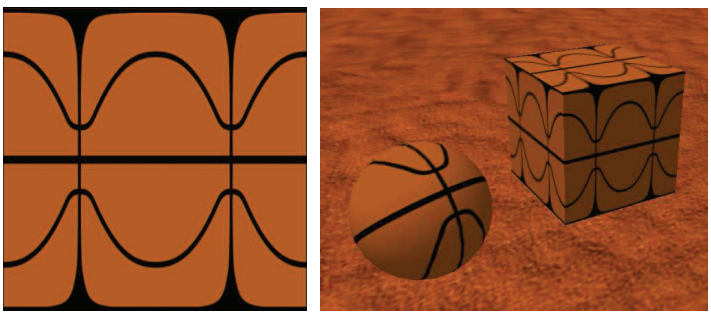


Imagen 6.4.2: La misma textura aplicada a distintos modelos

2. Características de las superficies.

Las superficies de los objetos pueden tener muchas características. Algunas son tan obvias como un color o una textura, pero otras son realmente complejas como pueden ser características de movimiento o de reflejo. Por ejemplo, si la superficie de un objeto debe reflejar nítidamente lo que hay a su alrededor como en el caso de un espejo o no tan nítidamente como en el caso de una superficie cromada de un coche, es evidente que es un tema complejo es sí mismo.

Las características de superficies con que podemos trabajar en Second Life hoy en día son: el color, la textura, la transparencia, el resplandor (glow), el brillo, el mapeamiento, la claridad, la irregularidad, y con respecto a la textura, la cantidad de repeticiones por cara, la rotación, y el origen de la aplicación. Para enumerarlas hemos utilizado el nombre que aparece en la interface en español del programa (aunque existen muchos fallos en la traducción). Como son muchas características las estudiaremos detenidamente una a una.

Para gestionar las características enumeradas anteriormente, existe un panel que se llama ”Textura” en la ventana de edición de los objetos. Este panel se puede observar en la figura siguiente tanto en español como en inglés. Tenemos que tener en cuenta que existen otras posibilidades que no se encuentran en este panel, como la anteriormente citada de material de reflejo. Esto lo estudiaremos mas adelante.

Imagen 6.4.3: La pestaña Textura en español e inglés

◆ **Texture, Textura:**

El concepto de textura es tan importante que a veces se confunde con el concepto de superficie, es decir, se confunde la parte con el todo. Al proceso de crear una superficie, a veces se le denomina texturización, ya que es la característica más espectacular de una superficie y la que aporta a los objetos su apariencia real.

Una textura es una imagen de tipo mapa de bits. Existen varios tipos, las más normales son las texturas fotográficas. Por ejemplo, si quiero un objeto 3D que parezca una piedra, aplicaremos a un modelo 3D la foto de una piedra real. Otras texturas pueden ser generadas por ordenador, por ejemplo, las texturas procedimentales que crean imágenes “sintéticas” y por lo tanto no recogidas de la naturaleza. Estas imágenes se utilizan luego como las fotográficas para su aplicación en el modelo.

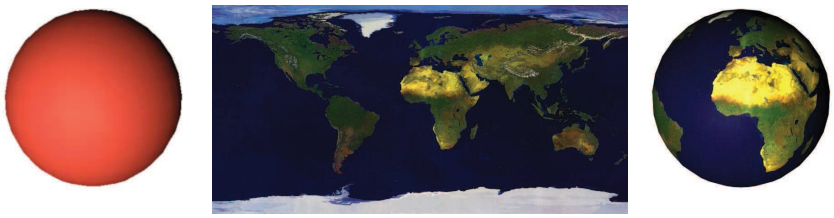


Imagen 6.4.4: Modelo 3D, textura y el resultado

La textura por defecto de los modelos es de madera contrachapada como se muestra en la imagen 6.4.3. Por lo tanto, cada vez que creamos un objeto lo veremos inicialmente con apariencia de madera.

- **Color, Color:**

Todo modelo construido en SL tiene un color que por defecto es blanco como se puede ver en la imagen 6.4.3. Este color se representa en código RGB. Cuando tenemos un textura aplicada, el color se aprecia muy poco y dependiendo de la textura, sirve para tintarla, de modo que podemos aplicar una textura, pongamos por ejemplo, de madera, y utilizar el color para verla en distintos colores.

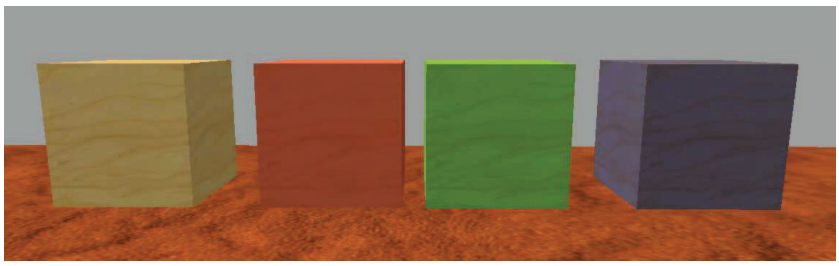


Imagen 6.4.5: Caja texturizada por defecto y coloreada usando el parámetro Color

- **Transparency, Transparencia:**

Como se puede apreciar en la imagen que sigue, la transparencia modifica la opacidad de los objetos. Su valor varía de 0, que hará que se visualice absolutamente opaco, a 100 en el que se considerará absolutamente transparente. En la imagen siguiente podemos observar el mismo objeto con transparencias de 0, 25, 50, 75 y 100. La transparencia total no se visualiza tal cual para que podemos saber que el objeto existe.

Imagen 6.4.6: Distintos grados de transparencia para el mismo objeto

- **Glow, Glow:**

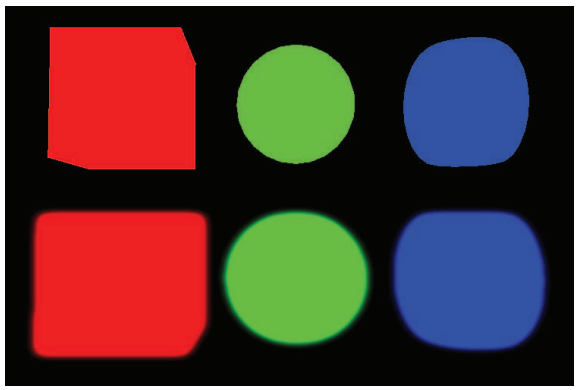


Imagen 6.4.7: Un objeto con resplandor y sin él

En la imagen 6.4.7 podemos observar el efecto del parámetro glow, que se podría traducir al español como resplandor. Los valores que admite van de 0 a 1. En la parte de arriba tenemos tres objetos de distinto color y abajo tenemos los mismos objetos pero con glow puesto a su máximo valor. Lo que produce es una difuminación y aumento del borde de los objetos, con los

que estos parecen mayores. Se utiliza mucho para crear elementos luminosos y efectos especiales como por ejemplo luces de neón y en combinación con el parámetro brillo y los sistemas de partículas se pueden construir simuladores de motores a reacción, etc.

◆ **Full Bright, Brillo Total:**

En la imagen que sigue podemos observar una caja con brillo total en primer plano y sin brillo la de más atrás. Normalmente esta característica se aplica en combinación con el glow, ya que se potencian mutuamente. El brillo total solo admite dos valores: verdadero o falso. Cuando está activado no se aplican sombras de ninguna clase, de modo que aunque sea de noche, el objeto estará iluminado de la misma forma que si fuera medio día.

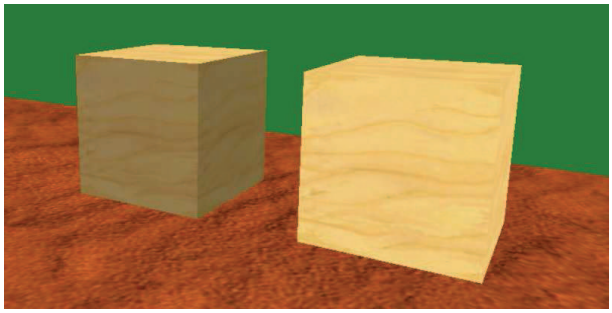


Imagen 6.4.8: Un objeto con brillo total y sin él

Esta característica se utiliza mucho para crear objetos que tenga luz propia y por lo tanto la estudiaremos también en el tema de iluminación y en especial para la creación de objetos con autoiluminación como pueden ser las bombillas.

◆ **Mapping, Mapeamiento:**

El mapeado se refiere a la acción de colocar una textura sobre un objeto tridimensional. Es una tarea que puede llegar a ser compleja, ya que colocar una textura 2D sobre un objeto 3D supone un trabajo parecido al de “envolver” una escultura con un papel de regalo. Si la escultura tiene forma geométrica, la tarea es sencilla, pero si tiene una forma irregular, debemos de elegir entre distintos modos.

En Second Life solo tenemos dos modos de mapeado: default y planar. El mapeado default, como su propio nombre indica, es el mapeado que SL aplica por defecto a cualquier objeto cuando se texturiza. En este caso el programa SL se encarga de elegir la mejor manera de “envolver”, de aplicar la textura sobre el objeto dependiendo de la geometría de este. En el caso de la opción “Planar”, SL le aplica toda la textura a cada una de las caras que componen el objeto, de forma que si el objeto es por ejemplo un cubo, aparecerá la textura repetida 6 veces, una en cada cara.

Imagen 6.4.9: Textura por defecto y su mapeado default y planar

En la imagen de arriba se puede ver dos pares de objetos. A los de arriba se le ha aplicado un mapeado planar y a los de abajo uno default. A las cajas no les afecta el cambio de mapeado, ya que el default y el planar coinciden en su forma de aplicación para las cajas, pero no sucede así para las esferas. La esfera de arriba no queda tan natural como la de abajo ya que el programa envuelve la textura de forma adecuada solo en el caso default.

Por otro lado, debemos comprender la manera en que SL aplica las texturas según la geometría del objeto y obrar en consecuencia. Por ejemplo, en la imagen siguiente podemos ver como una textura para un balón de playa o de baloncesto, se debe de crear ex profeso ya que sobre la superficie de una esfera funciona bien pero no sobre la superficie de una caja.

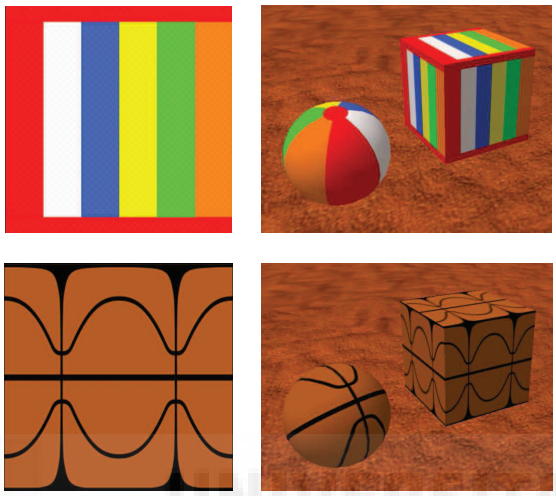


Imagen 6.4.10: Texturas preparadas para su aplicación en esferas

- **Shininess, Claridad:**

Este parámetro muestra unas superficies más o menos pulidas. Puede tomar cuatro valores: ninguno, bajo, medio y alto. En la figura de abajo se puede apreciar la diferencia entre los cuatro valores para distintos objetos. También podemos comprobar que este parámetro funciona mejor para superficies curvas que para superficies planas.

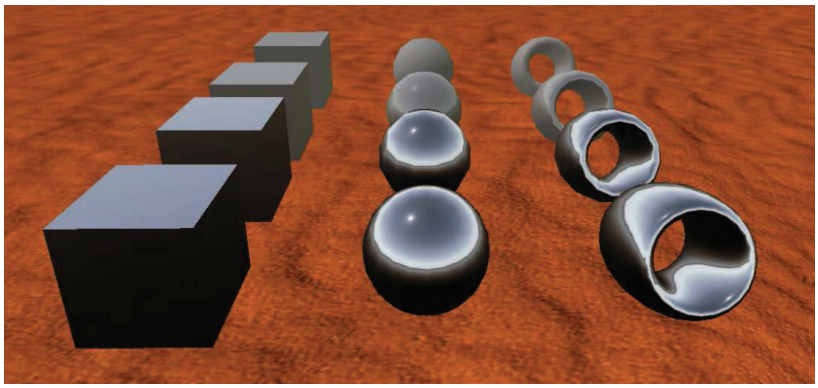


Imagen 6.4.11: Shininess en sus cuatro valores

- **Bumpiness, Irregularidad:**

Este concepto es muy importante para aportar realismo a nuestros objetos. Se trata de las pequeñas irregularidades de las superficies que son muy difíciles de crear mediante modelado. Por ejemplo, debemos imaginar la dificultad de crear la ingente cantidad de polígonos necesarios para modelar las pequeñas elevaciones de una pared estucada. Además esto haría que el software encargado de representar los polígonos en la pantalla se saturase de cálculos y al final fallase.

Para resolver esta cuestión, el software crea unas texturas especiales llamadas texturas “Bump” o de relieve, que aplicadas junto con otras texturas, proporcionan realismo sin una gran cantidad de cálculo. Son texturas en tonos de grises que el software interpreta como pequeñas hendiduras o resaltes dependiendo si son más cercanos al blanco o al negro.

En Second Life, no se puede proporcionar una textura bump a nuestros objetos como en otros programas, sino que debemos elegir de una lista desplegable entre 18 valores. En las dos imágenes de abajo solo representamos 16, ya que existen dos valores: Brightness y Darkness que solo modifican la luminosidad.

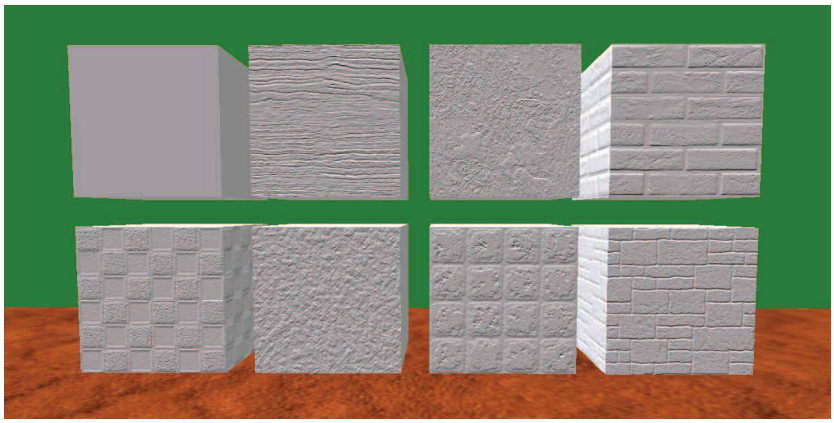


Imagen 6.4.12: Distintos valores para el parámetro Bumpiness

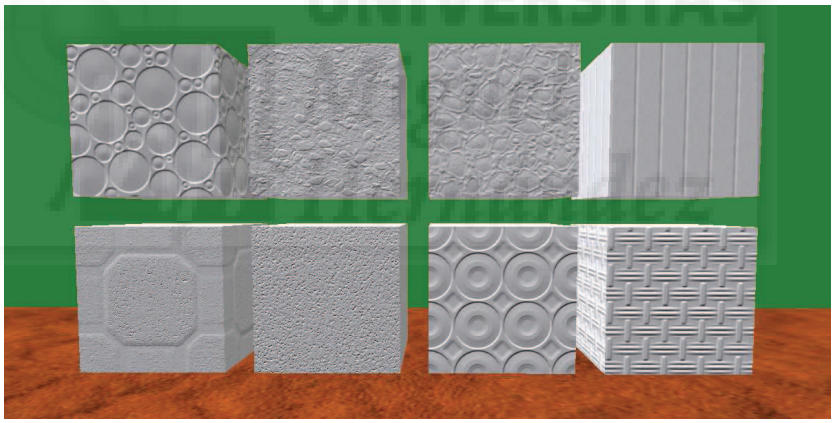


Imagen 6.4.13: Textura por defecto y su mapeado default y planar

En la imagen 6.4.12 se pueden observar las siguientes irregularidades de superficie: None, Woodgrain, Bark, Bricks, Checker, Concrete, Crustytile y Cutstone.

En la imagen 6.4.13 se pueden observar las siguientes irregularidades de superficie: Discs, Gravel, Petridish, Siding, Stonetile, Stucco, Suction y Weave.

- **Repeats Per Face, Repetición por cara:**

Esta característica hace posible que podemos repetir una misma textura más de una vez en una cara. Por ejemplo, si tenemos una textura de un azulejo y tenemos un objeto que es una pared, podemos repetir la textura 50 veces en horizontal por 20 en vertical para que parezca una pared realista con 1000 azulejos. Esta solución es muy práctica ya que la pared será una caja con 6 caras, en lugar de crear 1000 objetos (cajas) con 6.000 caras en total y con el exceso de cálculos que eso supone.

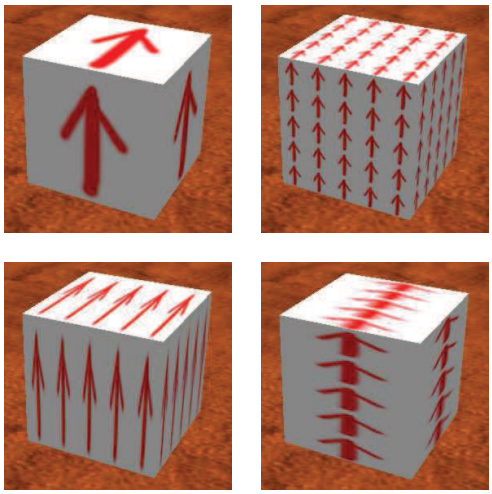


Imagen 6.4.14: La textura original y Repeticiones por cara de: 5 x 5, 5 x 1 y 1 x 5

Existe la posibilidad de invertir una textura si no nos gusta el sentido en que queda, para ello pulsar la opción “Flip”, Invertir, y quedará como muestra la imagen que sigue (debemos fijarnos en el sentido de las flechas).

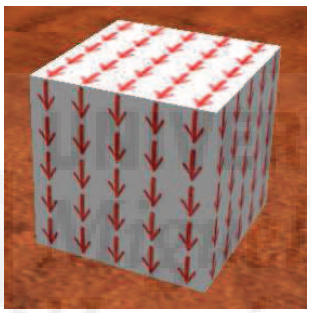


Imagen 6.4.15: Repeticiones por cara: invertir

- **Rotation, Rotación:**

La rotación hace que la textura gire y por lo tanto cambie su orientación. La rotación se mide en grados. En la imagen inferior se puede observar que la textura ha girado 45 grados. Podemos interpretar que es un error, ya que parece que sean –45º, pero debemos tener en cuenta que el giro se hace con respecto a una cara de referencia que en este caso se encuentra debajo.

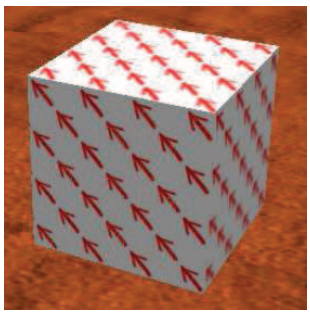


Imagen 6.4.16: Rotación: 45º

- **Repeats per meter, Repetición por metro:**

Esta opción solo aparece si usamos un mapeado Default. En la imagen se observan dos cubos de 1 metro de lado. La textura está repetida por metros, en la de la izquierda tiene un valor de 1 y a la derecha tiene 3 repeticiones por metro.

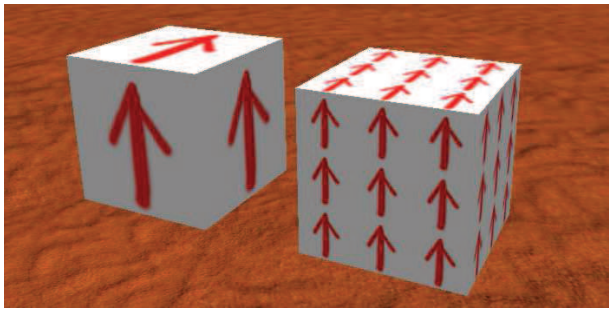


Imagen 6.4.17: Repeticiones por metro

◆ **Offset, Contrarrestrar:**

Esta variable lo que hace es comenzar a texturizar las caras no desde el origen de la textura sino desde el valor que le introduzcamos. Puede tomar valores desde –1 a +1, de tal forma que si vale 0, el origen de la textura y la cara a texturizar coinciden. Si va aumentando, observaremos como la textura se desplaza hacia arriba como muestra la imagen siguiente (el sentido dependerá de la orientación del objeto).

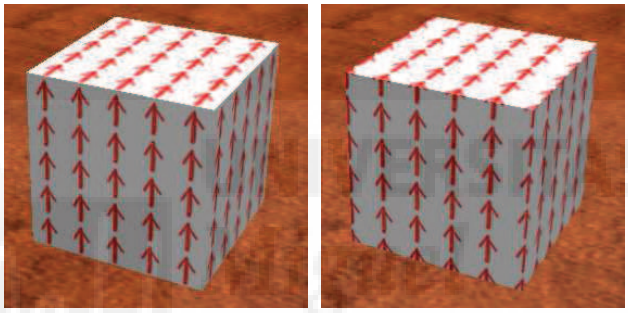


Imagen 6.4.18: Offset de 0,5

Por ejemplo, arriba observamos la textura normal a la izquierda y a la derecha la textura con un offset de 0,5 tanto vertical como horizontalmente, por eso se desplaza hacia arriba y hacia la izquierda, lo que hace que las flechas queden partidas justo por las aristas de las caras.

◆ **Align media texture, Alinee textura de multimedia:**

Esta función se supone que es para un uso avanzado de las texturas. Se usa para cuando queremos poner como textura de un objeto, una sucesión de imágenes, es decir, un vídeo. Esto se puede realizar y posiblemente cuando “cargamos” el vídeo, se visualice desplazado. Con este botón podemos ajustar el vídeo o textura cargada con el objeto 3D. Veremos un ejemplo de su uso en el tema dedicado a la utilización de material multimedia.

3. Canales alpha.

Los canales alpha, también llamados máscaras alpha se aplican mucho en Second Life. Estos canales permiten definir zonas opacas, con algún grado de transparencias o totalmente transparentes mediante las aplicaciones de imágenes. Esto supone que es una forma relativamente sencilla de crear distintas zonas de transparencia sobre los objetos. Se utilizan en ropas para crear prendas semitransparentes o para “cortarlas” ya que se pueden usar texturas con zonas no visibles donde no tenga que verse ropa. Así también podemos crear tatuajes que se aplicarán sobre la piel. En arquitectura se utilizan mucho ya que permiten crear orificios, cristales, celosías, etc. Por ejemplo, si queremos realizar un enrejado para una ventana, es mejor aplicar una máscara alpha que modelar cada una de las varillas que componen el enrejado. Además de ser más productivo desde el punto de vista del tiempo y el

esfuerzo empleado en la creación, también supone una mejora en el rendimiento de Second Life puesto que al fin y al cabo, tiene que mostrar menos polígonos en la escena.

Si construimos un modelo y le aplicaremos una imagen con canal alpha ilustraremos mejor este concepto. Para crear el canal alpha utilizaremos el conocido programa de retoque de imágenes ®Adobe ®Photoshop.

En PhotoShop, tendremos que abrir la ventana canales donde se encuentran separados todos los colores que componen una imagen. El código más utilizado es RGB, por lo tanto si nuestra imagen codifica los colores en RGB tendremos tres canales: rojo, verde y azul. Lo que haremos será añadir un canal más. Este canal será de tipo alpha y por tanto será en escalas de grises. Estos grises serán interpretados por Second Life como grados de transparencia. Si el color es negro, Second Life lo interpretará como transparente total y si es blanco como totalmente opaco y en función de esto, los tonos grises.

En la imagen inferior se puede ver a la izquierda del todo, la foto de una mariposa bajada de Internet. En medio podemos visualizar cómo es el canal alpha creado en PhotoShop y a la derecha, la textura con canal alpha que se aplicará a un objeto en Second Life. En los casos prácticos detallaremos el proceso completo de la realización de estas texturas.

Imagen 6.4.19: Imagen original, máscara alpha y a la derecha la imagen a aplicar

Una vez creada la textura, guardamos el archivo como un “targa” con la extensión TGA y con 32 bits/píxel de resolución. Luego la subimos a Second Life y la aplicamos a un objeto mediante el método normal, o sea, editamos el objeto y vamos a la pestaña Textura y hacemos clic en el botón Textura. Esto nos permitirá buscar en nuestro inventario la textura recién subida y aplicarla al objeto.

En la imagen inferior se puede observar a la izquierda la textura con canal alpha que hemos mostrado y cómo queda en una caja de aproximadamente tres metros de ancha por dos de alta y apenas 5 cms. de profundidad.

Imagen 6.4.20: La misma textura en Second Life sin y con semitransparencia

En la parte derecha de la imagen de arriba podemos observar la misma mariposa pero con las alas inferiores semitransparentes. Esto es debido a que hemos modificado la textura con canal alpha y ahora el canal no es solo negro y blanco, sino que incluye un tono de gris como se puede observar en la imagen siguiente.

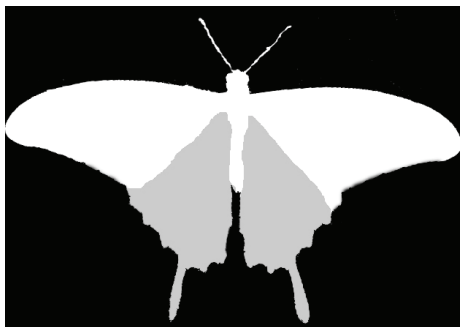


Imagen 6.4.21: Canal alpha con semitransparencia

4. Reflejos.

Los reflejos en Second Life son un recurso que tiene un coste muy alto en cuanto a gasto computacional. Por este motivo y para no aumentar el "lag", es decir, el retardo producido en el tiempo real de ejecución del metaverso, se toman varias medidas.

Por una parte, se limita su uso al agua. Si observamos cualquier tipo de agua, la del mar que rodea nuestras islas o al agua de lagos, etc., podemos observar nuestro reflejo, siempre y cuando hayamos habilitado su uso. Otra medida es que el cálculo de los reflejos esté en la parte del cliente, es decir, siempre dependerá del tipo de ordenador y sobre todo de que el hardware de la tarjeta gráfica del ordenador cliente, los pueda generar y visualizar.

Esto quiere decir que no vamos a poder crear nuestro propios espejos en tiempo real, por lo menos de momento (aunque como es un problema de cálculo computacional, solo es cuestión de tiempo), pero podemos crear objetos que simulen la reflexión y por lo tanto que parezcan de cromo, acero, etc. y también hemos visto trucos para crear suelos muy brillantes para que parezcan muy pulidos aunque a cambio de cierta complejidad y por supuesto, no interaccionan en tiempo real, lo que le resta realismo.

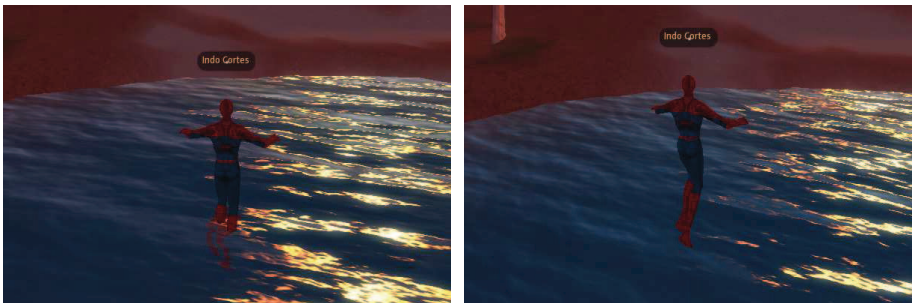


Imagen 6.4.22: Se visualiza el reflejo si nos introducimos parcialmente

En la imagen superior se puede observar cómo cuando estamos introducidos parcialmente en el agua como en la imagen de la izquierda, nuestro avatar se deforma y se observa también el reflejo de la luz solar en el agua. Pero en cuanto subimos un poco más aunque solo sean unos centímetros, como en la imagen de la derecha, Second Life ya no produce ningún reflejo.

Imagen 6.4.23: En el mar y aguas interiores si se visualiza el reflejo

Si comparamos las imágenes superior e inferior observaremos una gran diferencia. Y es que en el mar o en aguas que sean calculadas como tal, es decir, como agua, la refracción se visualiza. En la imagen de abajo, el agua es simulada mediante textura animada y por tanto no se calcula ninguna distorsión ni refleja ningún tipo de material.

Imagen 6.4.24: En el agua realizada con texturas animadas no se visualiza el reflejo

En la imagen siguiente se observa el fondo de una laguna cuando estamos sumergidos y cómo son reflejados los objetos emergidos que están alrededor de forma suficientemente realista.

Imagen 6.4.25: Reflejo de los objetos emergidos cuando nos sumergimos en agua

Imagen 6.4.26: Objeto reflectivo falso

En la imagen anterior se puede observar la creación de un objeto que simula los reflejos mediante un truco. Se puede comprobar que no es un reflejo en tiempo real, ya que nos deberíamos de ver reflejados nosotros mismos debido a que una superficie tan pulida como esta debería servir como espejo, y está claro que esto no se produce.

El truco se trata simplemente de no utilizar ninguna textura y dejarla en blanco y además ponemos el parámetro shininess al valor más alto. Esto hace que la superficie del objeto parezca más pulido y junto con un color determinado podamos dar una apariencia de que refleja su alrededor.

Este truco funciona mejor para superficies redondeadas o con aristas y en aquellos que en la vida real son por defecto pulidos y metálicos como llantas de coches, barandillas, etc.

5. Texturas animadas por transformación.

Las texturas animadas son propiamente las texturas que cambian en tiempo real en cuanto a su aplicación sobre la superficie del objeto 3D al que "envuelven". Estas texturas permiten realizar una gran cantidad de efectos, ya que podemos rotar, escalar, cambiar de posición o cambiar el sentido del movimiento de una imagen sobre un objeto 3D.

Las texturas animadas por transformación las podemos utilizar para crear efectos especiales de muchos tipos. Por ejemplo, supongamos que tenemos una imagen de una galaxia en espiral sobre una caja que simula un plano de fondo, entonces podríamos darle un movimiento giratorio para que simulase una galaxia real. Además en este caso, como en muchos otros, sería la única forma de hacerlo.

Para modificar el mapeado o aplicación de una textura sobre un objeto 3D en tiempo real, Second Life nos proporciona la función de LSL llamada llSetTextureAnim que vamos a detallar a continuación. La sintaxis es como sigue:

llSetTextureAnim (integer mode, integer face, integer sizex, interger sizey, float start, float length, float rate);

Un ejemplo típico puede ser esta sentencia:

llSetTextureAnim (ANIM_ON | SMOOTH | LOOP, ALL_SIDES, 1, 1, 0.0, 0.0, 0.5);

Dónde los primeros parámetros están separados por barras y como indica la sintaxis definen el modo de ejecución. Pueden tener los siguientes significados:

- ANIM_ON: activa que la textura se pueda animar.
- SMOOTH: suaviza la transformación de la textura.
- LOOP: crea un bucle infinito para que la transformación se haga continuamente.
- REVERSE: cambia el sentido del movimiento de la transformación.
- PING_PONG: cambia el sentido de la transformación alternativamente.
- ROTATE: gira la textura en vez de moverla.
- SCALE: escala la textura en vez de moverla o rotarla.

El parámetro ALL_SIDES se corresponde con faces e indica las caras que se verán involucradas en el cambio de transformación de texturas.
Los parámetros sizex y sizey indican la repetición de la textura por cada cara en el eje X y en el eje Y.

Start normalmente vale 0, ya que indica el Offset de aplicación de la textura.

Lenght indica el número de veces que se visualizará la textura y tiene que estar en concordancia con sizex y sizey. Por ejemplo si ponemos 16, sizex y sizey deberían valer 4 cada una para ver la textura repetida 16 veces en 4 filas de 4 columnas por cada cara.

Rate es la velocidad medida en frames por segundo. Si ponemos 0 no se moverá. Si esta velocidad tiene un valor negativo, tendrá el mismo efecto que si ponemos en el modo el valor REVERSE.

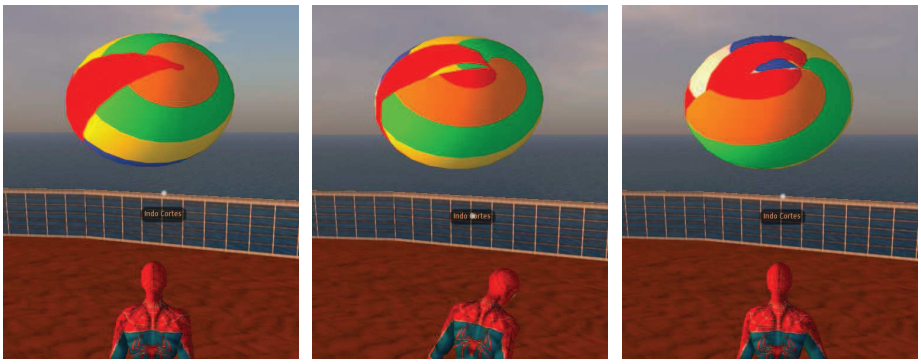


Imagen 6.4.27: Textura animada por transformación

En la imagen de arriba podemos visualizar tres momentos de la aplicación de una textura animada por transformación. Se puede ver cómo la textura cambia sobre la esfera que permanece estática.

Como hemos escrito antes, las aplicaciones son muchísimas y podríamos realizar una esfera y texturizarla con un mapa de la tierra y sobre esta con un mapa de canal alpha que representase las nubes y mediante la transformación de esta última podríamos hacer una representación muy realista ya que la esfera podría girar en un sentido y las nubes de forma independiente. Para poder utilizar las texturas animadas, antes debemos tener habilitada esta función. En general no hay problema ya que está habilitada por defecto. Pero por si acaso, tenemos que utilizar un menú que no aparece normalmente y que se llama Advanced, es decir, avanzado. Para mostrar este menú debemos pulsar la combinación de teclas Ctrl + Alt + D. Dentro de este menú accedemos a rendering y veremos Animate texturas como se observa en la imagen inferior.

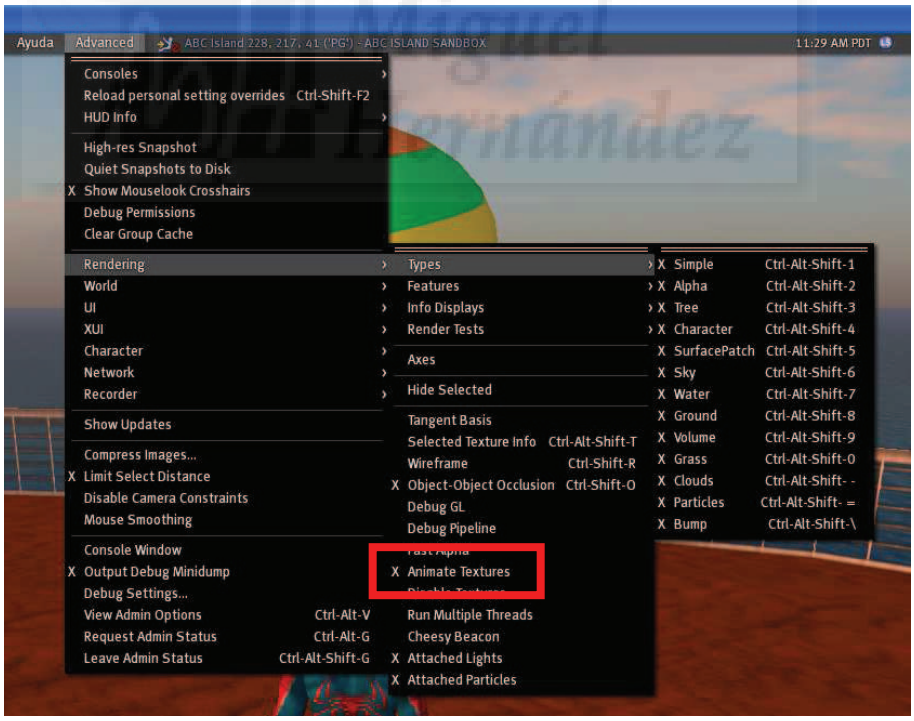


Imagen 6.4.28: Habilitar las texturas animadas

Este menú avanzado tiene gran importancia en todo lo referente a la visualización al igual que el comando preferencias > gráficos que se encuentra en el menú Edición. Todas estas opciones deben ser controladas por el usuario y en consonancia con la tarjeta gráfica de que dispongamos en cada ordenador podremos aumentar o no la calidad gráfica con que se visualiza Second Life.

6. Textura animadas por intercambio.

Este tipo de texturas animadas por intercambio, supone que cada cierto tiempo la textura que tiene un objeto, se cambia automáticamente por otra. Por lo tanto no es una textura animada en sí misma, sino una serie de texturas que se van alternando y aplicando al objeto.

Sus aplicaciones son muchas, ya que podemos "envolver" un objeto 3D con una animación, lo que abre las puertas para crear proyectores de cine, o realizar efectos especiales como cascadas de agua, puertas de ciencia ficción, etc.

El problema es que no se pueden aplicar animaciones tipo gifs animados sobre los objetos directamente, sino que tenemos que crear nosotros un script para hacerlo. Este script no es muy difícil, pero evidentemente nos estamos adelantando a la unidad dónde veremos las herramientas de programación que posee Second Life. Por lo tanto, vamos a explicar muy por encima cómo funciona este programa y a comprobar su resultado.

Antes de enfocar nuestra atención al programa en sí, tenemos que tener en cuenta que los objetos tridimensionales pueden tener muchas caras y por tanto, puede ser que solo queramos que tenga texturas animadas una de ellas. Por ejemplo, supongamos que queremos realizar una especie de anuncio y esto supone una animación de 10 frames o imágenes. Si queremos "proyectar" este anuncio sobre una caja, no tiene sentido que lo hagamos sobre las 6 caras que la componen, sino sobre una de ellas, o como mucho sobre la que ve el espectador y la de detrás, por lo que el script debe recoger esta especificación.

Por otra parte, debemos considerar la mejor manera de acceder a las texturas y para ello, sin duda, debemos usar la pestaña Contenido del propio objeto a texturizar. Por lo tanto, antes de ejecutar el script, pondremos las texturas involucradas en el "content" del objeto. El script quedaría como sigue:

```
float tiempo=1.0;          // tiempo de intercambio de una textura por otra
integer numtext=3;         // nº de texturas que componen la textura animada y que están en content
integer cara=0; // 0 será la cara a texturizar, si queremos todas mirar la línea llSetTexture
integer i;                 // contador para saber por que textura vamos dentro del bucle

default
{
   state_entry()
   {
      for (i=0;i<numtext;i++)
      {
         llSetTexture(llGetInventoryName(INVENTORY_TEXTURE, i),cara);
                  // pone la textura que toque dentro del content del objeto
                  // si queremos todas poner ALL_SIDES en lugar de "cara"
         llSleep(tiempo);
      }
      llResetScript();
   }

}
```

Este script es la versión más reducida y simple que cumple con los objetivos que hemos estudiado. Creemos que es una buena solución para muchas de las necesidades con que podamos toparnos dentro de Second Life y además no da problemas. Por ejemplo, podemos incluir entre nuestras texturas, aquellas que queramos con canales alpha y hacer así un estanque donde en primer plano veamos la superficie con ondulaciones y debajo unos peces moviéndose.

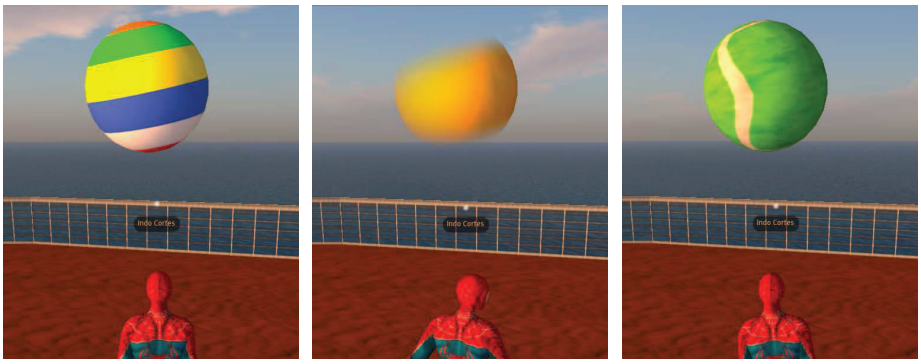


Imagen 6.4.29: Textura animada por intercambio

En la imagen anterior podemos observar las tres texturas que se intercambian en tiempo real mapeando una esfera. Debemos destacar que en la toma segunda, se puede ver cómo la textura contiene un canal alpha, lo que abre un gran abanico de posibilidades para esta técnica.

Como conclusiones a este tema podemos decir que el control de superficies es de una utilidad muy grande, que podríamos clasificar de imprescindible para un creador que se mueva en el ámbito de Second Life. Por decirlo de otra manera, no se puede crear algo realmente bueno en este metaverso sin controlar los recursos que hemos estudiados en este tema. Es tanto o más importante tener el control de la superficie de los objetos como el modelado de los mismos.

Caso práctico 4.1: Aplicar distintas superficies a los objetos.

Objetivo: Controlar los parámetros básicos para crear distintos tipos de superficies.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Modelado.
2. Asignar texturas.
3. Crear superficies metálicas.
4. Crear superficies con irregularidades.
5. Crear superficies con resplandor.
6. Crear superficies transparentes.

1. Modelado.

La diferencia entre las dos imágenes que se muestran a continuación es muy evidente. En la imagen superior utilizamos la textura por defecto, que es de tipo madera. En la imagen inferior trabajamos las superficies de cada una de las partes distintas para que se representen como objetos distintos.

Imagen 6.4.cp.1.1: Objetos con la textura por defecto y texturizados ex profeso

Para realizar esta práctica, hemos creado un rincón de una casa. Esto hace que el entorno no sea extraño, sino al contrario, muy familiar y así podemos concentrarnos en las superficies y no en los modelos que a veces son verdaderamente complejos y no habituales.

Hemos creado dos paredes y un suelo con cajas para delimitar el entorno. Luego hemos creado una sencilla lámpara con una base compuesta por una esfera, un cono y un toro. Internamente existe una bombilla creada con un toro y una esfera y luego está la pantalla de la lámpara creada con un cilindro de paredes muy delgadas.

2. Asignar texturas.

En la imagen de abajo se aprecia la textura aplicada a la base de la lámpara, la esfera. Pretendemos cambiar una textura por otra. Para ello seguiremos los siguientes pasos.

Imagen 6.4.cp.1.2: La esfera y el toro son mapeados con una textura nueva

2.1. Seleccionamos la esfera que compone la parte inferior de la lámpara. Hacemos clic con el botón derecho y elegimos Editar. Pulsamos sobre la pestaña “Textura”.

2.2. Hacemos clic sobre el control “Textura” y esto hará que aparezca la ventana de selección de textura como se puede apreciar en la imagen de abajo. La ventana se llama “Pick: Texture” y sirve para poder acceder a nuestro inventario y elegir la textura que deseemos. En principio, suponemos que no hemos subido ninguna textura y por lo tanto solo disponemos de las texturas que nos aporta el propio Second Life.

2.3. Elegimos la textura Library > Textures > Wood > Bubinga.

2.4. Repetimos la operación para la parte superior, como muestra la imagen 6.4.cp.1.6. Para ello, utilizar el botón de cuentagotas seleccionándolo y haciendo clic sobre la textura de la base esférica. Esto hace que podamos copiar y pegar texturas ya utilizadas en otros objetos.

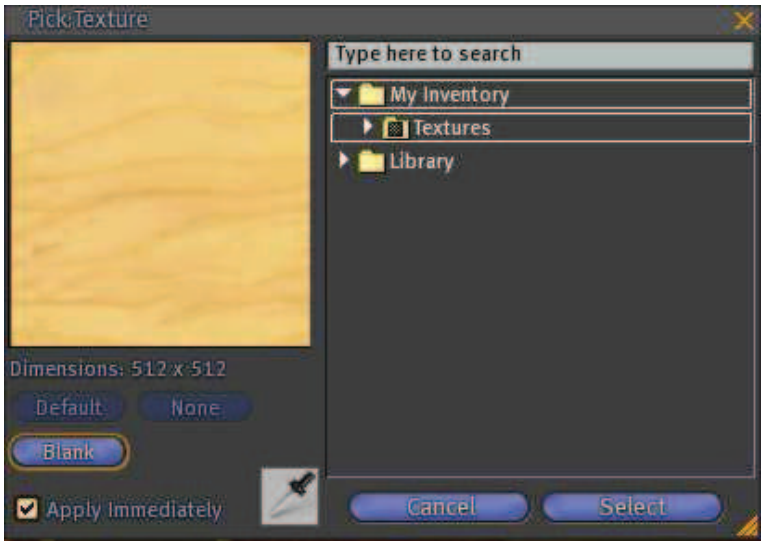


Imagen 6.4.cp.1.3: Ventana para seleccionar la textura

Por lo tanto a la hora de asignar una textura a un objeto nos encontramos con que podemos seleccionarla de una lista de nuestro inventario o tomarla de otro objeto de Second Life. Pero aún hay otra alternativa: podemos arrastrar la textura hasta el objeto para texturizar cara por cara.

Si utilizamos este método tenemos que tener en cuenta que si el objeto tiene varias caras como en el caso de una caja, solo texturizaremos la cara a la que arrastremos la textura. Esto se puede evitar si pulsamos la tecla Shift mientras arrastramos la textura. Si el objeto no tiene más de una cara, por ejemplo en el caso de una esfera o un toro, esta distinción no tiene sentido. La siguiente imagen trata de ilustrar estos procedimientos.

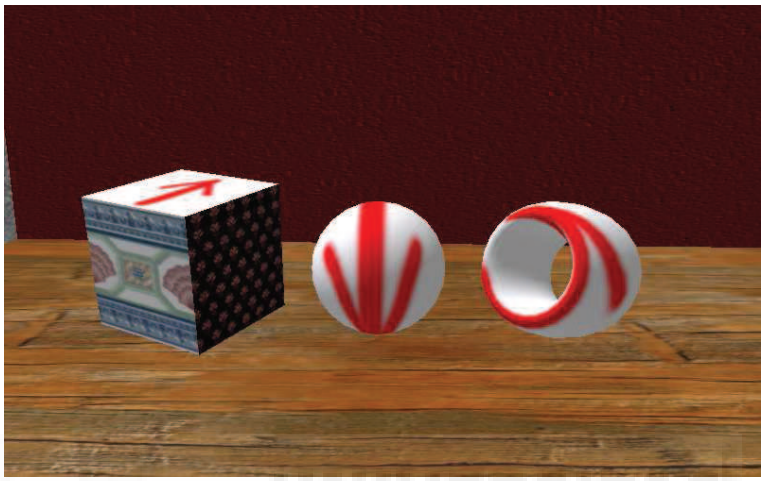


Imagen 6.4.cp.1.4: Texturización por caras

Otro hecho que tenemos que conocer es que cuando el objeto tenga un “hollow” o agujero, este formará caras internas y estas caras no se pueden texturizar por separado solo se admite una textura para todas las caras internas.

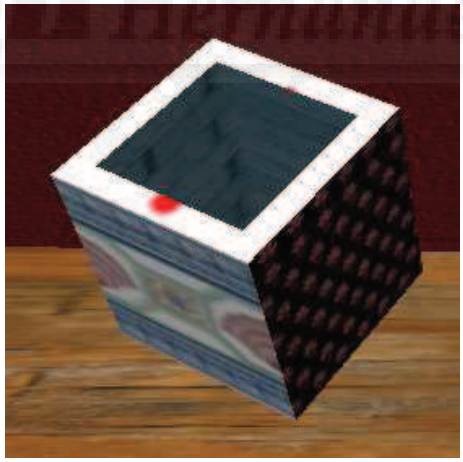


Imagen 6.4.cp.1.5: Texturización del orificio

3. Crear superficies metálicas.

3.1. Para la parte central de la base de la lámpara, realizamos una superficie metálica. Para ello seleccionamos el cono que constituye el modelo y le quitamos su textura por defecto. Para ello basta con pulsar sobre la textura y luego sobre el botón “Blank” (En blanco).

3.2. Luego pulsamos sobre Color y elegimos un blanco o un gris muy claro (por ejemplo, un RGB de 200, 200, 200).

3.3. Por último cambiamos el valor de Shininess (Claridad) y elegimos el valor “alto”. Esto creará unos brillos que le darán aspecto de metal brillante.

Imagen 6.4.cp.1.6: Superficie metálica

4. Crear superficies con irregularidades.

4.1. Seleccionar la pared de fondo y editar la textura. Ponemos la textura “En blanco”, es decir, descartamos utilizar textura alguna. Como color elegimos un granate. Esto hace que la pared se vea como muestra la imagen superior.

Imagen 6.4.cp.1.7: Superficie con irregularidades

4.2. Para darle mayor realismo y un aspecto de pintura estucada, utilizamos el concepto de Bumpiness (Irregularidad) y elegimos el valor “estuco”. Esto hará que la superficie de la pared se vea como muestra la imagen inferior.

4.3. Seleccionar la pared de la izquierda y editar la textura. Ponemos la textura en blanco y como color elegimos un gris claro. En el control de Irregularidad elegimos el valor “hormigón”.

4.4. Realizamos la misma operación de texturización pero con el suelo. En este caso no aplicamos irregularidad, solo la textura que encontramos dentro del inventario en Library > Textures > Wood > Old wood. El resultado se puede observar en la imagen que sigue.

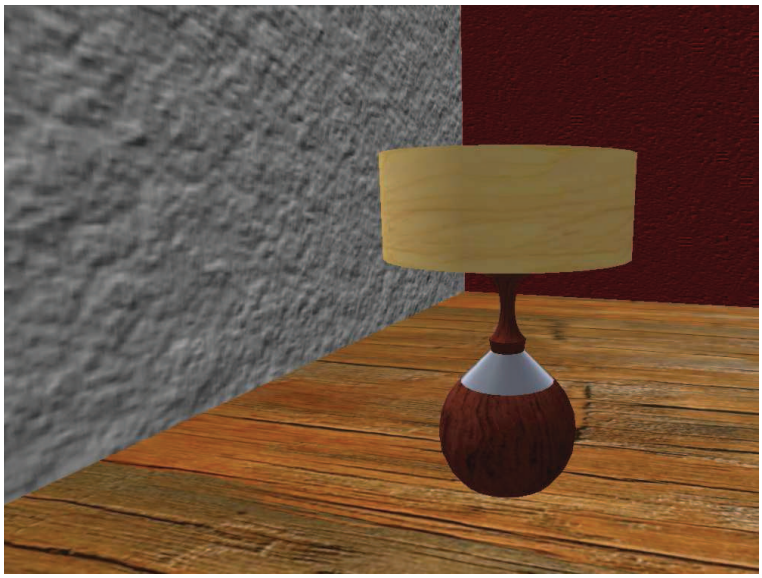


Imagen 6.4.cp.1.8: Las paredes y suelo texturizados

4.5. Seleccionar la pantalla de la lámpara. Le asignamos la textura Library > Textures > Fabric > Plaid 1.

4.6. También debemos texturizar la rosca de la bombilla como una parte metálica, por lo que seguiremos el procedimiento descrito en el punto 3 de este mismo caso práctico. Tanto la bombilla como su rosca se encuentran cubiertas por la pantalla de la lámpara, por lo que tenemos que desplazar esta hacia arriba para poder seleccionar los objetos.

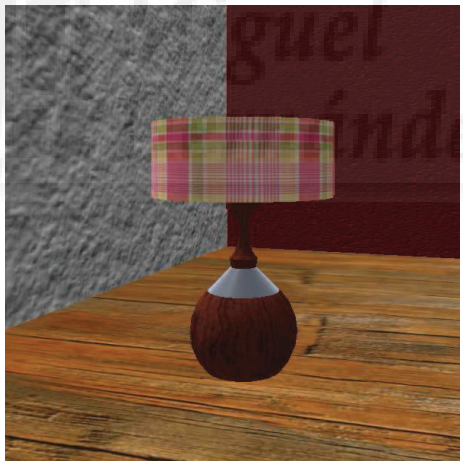


Imagen 6.4.cp.1.9: La textura de tela aplicada a la pantalla

5. Crear superficies con resplandor.

5.1. Seleccionamos la bombilla y ponemos su Textura en blanco. Luego le aplicamos un color amarillo. Y ponemos el valor de Glow a 1.00 y el brillo total activado. El efecto se puede observar en la siguiente imagen.

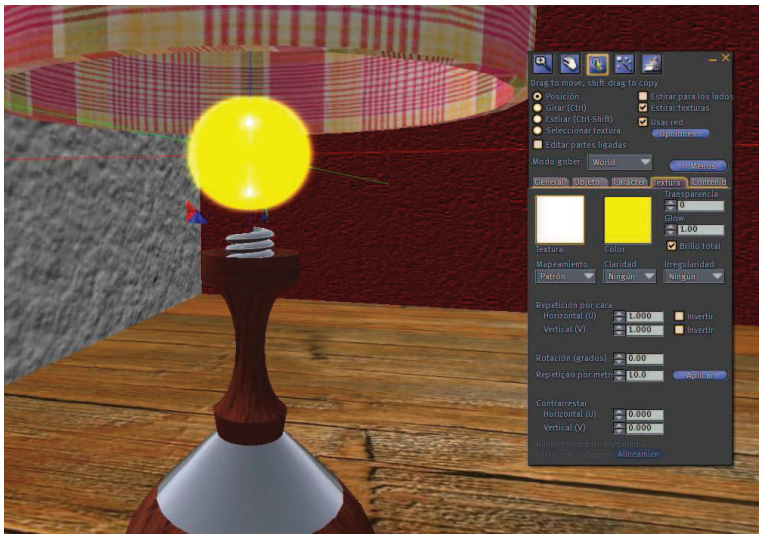


Imagen 6.4.cp.1.10: La bombilla con Glow y brillo total

6. Crear superficies transparentes.

El problema de los objetos transparentes es que permiten visualizar la geometría del objeto, y por lo tanto, la forma de construcción. Es un valor que puede ir de 0 a 100.

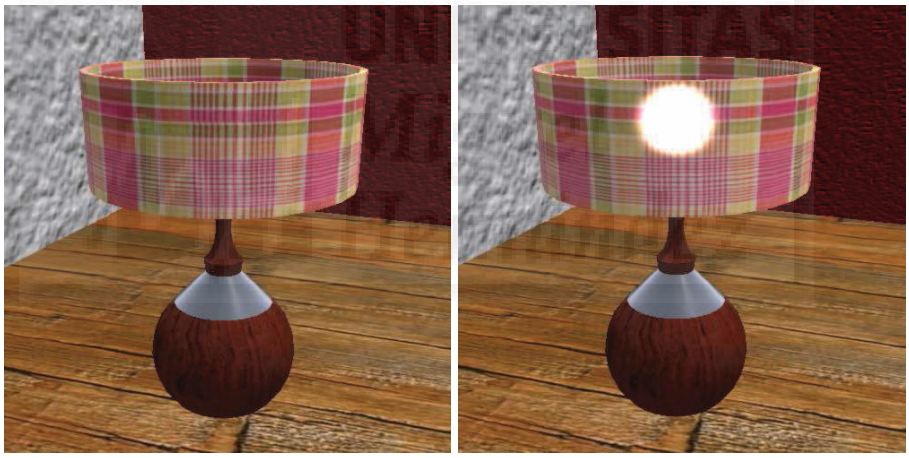


Imagen 6.4.cp.1.11: Pantalla con valor de transparencia de 0% y 1%

Además, cuando tenemos detrás objetos con glow, este resplandor se visualiza aunque la transparencia valga un 1%, como pone de manifiesto la imagen anterior.

6.1. Para poner la pantalla transparente, seleccionamos el cilindro que la forma y en Transparencia ponemos 35.

Observar cómo prácticamente no cambia la percepción de la bombilla comparándola con la imagen de transparencia 1% y sin embargo, el resto de la tela si es bastante menos opaca.

Imagen 6.4.cp.1.12: Pantalla con valor de transparencia de 35%

6.2. Si queremos crear una superficie imitando al cristal, lo mejor será que quitemos cualquier textura, luego elegimos un color suave y le aplicamos una transparencia moderada.

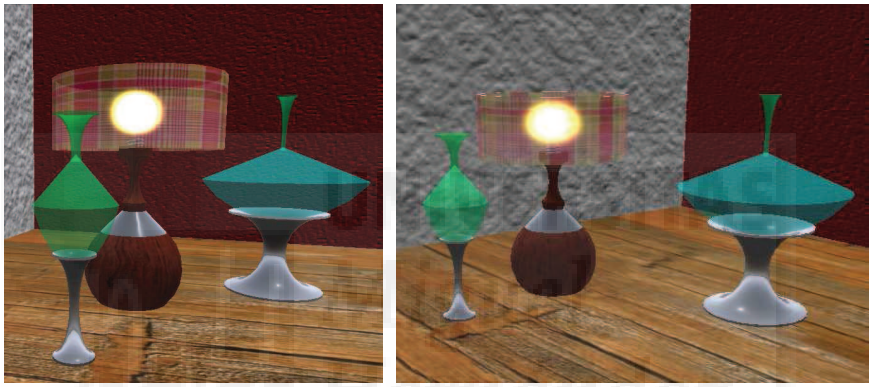


Imagen 6.4.cp.1.13: Dos objetos de "cristal", transparentes y sin textura

En la imagen precedente, se pueden observar dos jarrones con una textura de cristal, uno tiene un color verde y el otro azulado.

Conclusiones:

La adecuada texturización de los modelos es un proceso bastante fácil de realizar y además es muy metódico: se repiten muchas veces los mismos pasos. Además, conocer cómo se texturizan los objetos es algo indispensable por lo que esta tarea es ineludible para un creador en Second Life.

Caso práctico 4.2: Materiales con canales alpha.

Objetivo: Crear texturas con la opacidad definida en distinto grados y por áreas.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Creación de las texturas con canal alpha.
2. Aplicación sobre las geometrías.

1. Creación de las texturas con canal alpha.

En esta práctica vamos a crear modelos con las primitivas básicas y les aplicaremos unas sencillas texturas con canales alpha por que lo importante es el proceso, no el resultado final y no queremos que la complejidad del modelo o la textura influya en la percepción del método de creación.

Para crear las texturas con canal alpha llevaremos a cabo los siguientes pasos:

1.1. Abrimos Adobe® Photoshop®. Podemos realizar esta tarea con otro programa de tratamiento de imágenes pero nosotros utilizaremos este.

1.2. Creamos la imagen que queramos con las herramientas que necesitemos. Por ejemplo, nosotros hemos usado sobre todo la herramienta degradado.

1.3. Accedemos a la ficha Canales y creamos un nuevo canal pulsando sobre el icono nuevo canal en la parte inferior de la ficha. Aparecerá un nuevo canal con el nombre “alpha 1”. En principio aparece todo en negro.

1.4. Dibujamos en negro lo que queramos que quede transparente del todo y en blanco lo que queramos que quede opaco y en tonos grises lo que deseemos que quede en una transparencia según una escala de grises.

Imagen 6.4.cp.2.1: Textura con canal alpha y los canales creados

En la imagen superior se puede ver a la izquierda la textura creada y a la derecha la ficha de canales en el que destaca el último, el canal alfa con un cuadradito negro que será lo que quede transparente cuando se aplique sobre un cuerpo 3D en Second Life. Para hacer este cuadrado, hemos seleccionado un cuadrado, luego hemos ejecutado en el menú Selección > Invertir y hemos puesto el color frontal en blanco y hemos ejecutado en el menú Edición > Rellenar con color frontal.

1.5. Hacemos en el menú capa > acoplar imagen.

1.6. Guardamos el archivo con Archivo > Guardar como y tenemos que poner en tipo “targa” (extensión TGA) luego de ponerle un nombre y pulsar el botón “Aceptar” aparecerá una caja de diálogo dónde elegiremos la última opción: una resolución de 32 bits por píxel.

2. Aplicación sobre las geometrías.

2.1. Ahora abrimos Second Life y creamos tres objetos, en este caso hemos creado tres cajas y las hemos desplazado para que no se representen demasiado planas. En la imagen inferior se puede observar las tres texturas creadas con canales alpha con este método. Las zonas rojizas señalan las partes que luego se interpretarán como transparentes en Second Life

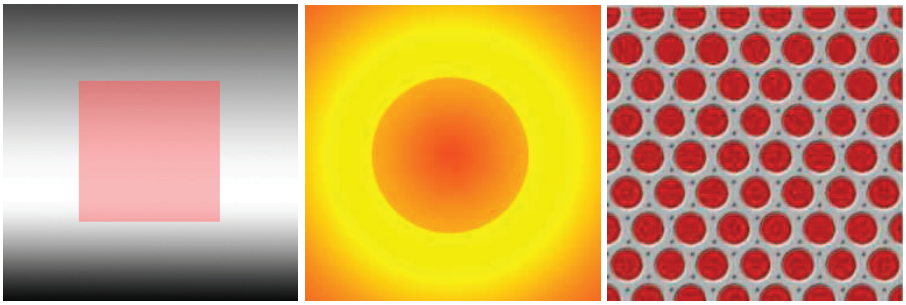


Imagen 6.4.cp.2.2: Texturas con canal alpha

2.2. Subimos las texturas con canal alpha haciendo Archivo > Upload imagen. Esta acción abre una diálogo para poder referenciar un archivo que como hemos escrito antes debe ser un TGA. Una vez aceptado, aparece un mensaje que nos advierte que el subir esta imagen nos ha costado 10 L$. Esta imagen pasará a formar parte del inventario en la carpeta de texturas. En las imágenes siguientes se puede apreciar cómo quedan los objetos.

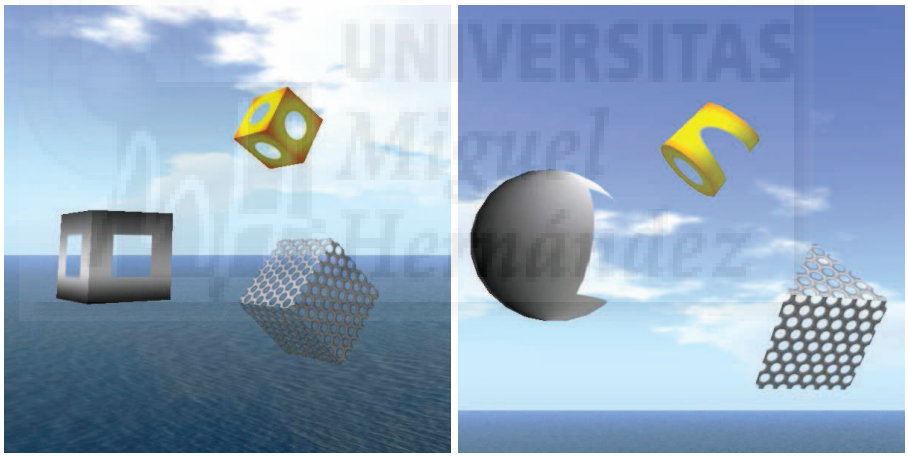


Imagen 6.4.cp.2.3: Texturas con canal alpha aplicadas a distintas geometrías

En la siguiente imagen se puede observar el canal alpha para una textura de mariposa dónde tenemos blanco, negro y gris y más abajo podemos observar cómo queda al aplicarla en Second Life.

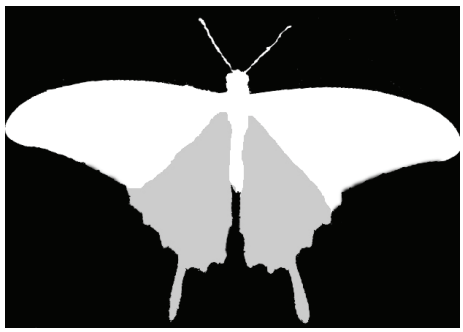


Imagen 6.4.cp.2.4: Canal alpha con semitransparencia

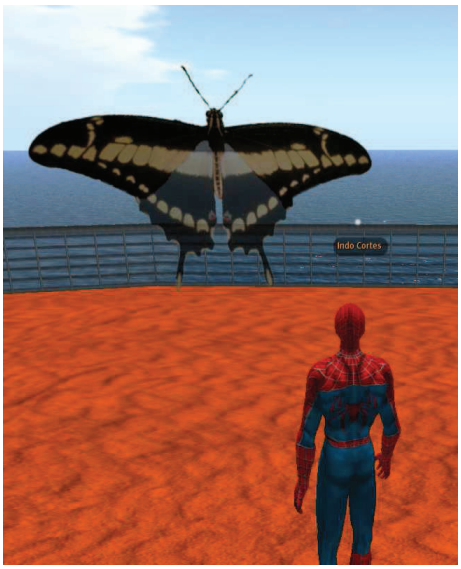


Imagen 6.4.cp.2.5: Aplicar semitransparencia en Second Life

La propia valla del sanbox está realizada con texturas que utilizan los canales alpha, lo que permite que los modelos sean simples cajas y que la propia textura defina finalmente lo que aparece del modelo.

Conclusiones:

En este apartado debemos de destacar la simplicidad con que nos permite trabajar este tipo de texturas en Second Life. Además este proceso nos va a permitir que modelar objetos complejos con mayor comodidad como pone de manifiesto las imagen que hemos producido con rejillas y semitransparencias, ya que utilizando las texturas de canales alpha nos ahorramos mucho tiempo, esfuerzo y polígonos sobre todo en modelar objetos.

Caso práctico 4.3: Texturas animadas por transformación e intercambio.

Objetivo: Realizar texturas que cambien en tiempo real para crear efectos y animaciones.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Crear una textura animada por transformación.
2. Crear una textura animada por intercambio.

1. Crear una textura animada por transformación.

Como ya vimos en la unidad teórica, las texturas asignadas a un modelo se pueden modificar en tiempo de ejecución, en tiempo real. Para ello utilizaremos una función que proporciona LSL que ya estudiamos en detalle y cuya sintaxis recordamos a continuación.

llSetTextureAnim (integer mode, integer face, integer sizex, interger sizey, float start, float length, float rate);

Vamos a crear una especie de universo esférico para lo cual utilizaremos una textura que representa una galaxia de tipo espiral y queremos que se mueva girando, pero que no gire el cuerpo esférico sino la textura aplicada. Para ello ejecutaremos los siguientes pasos:

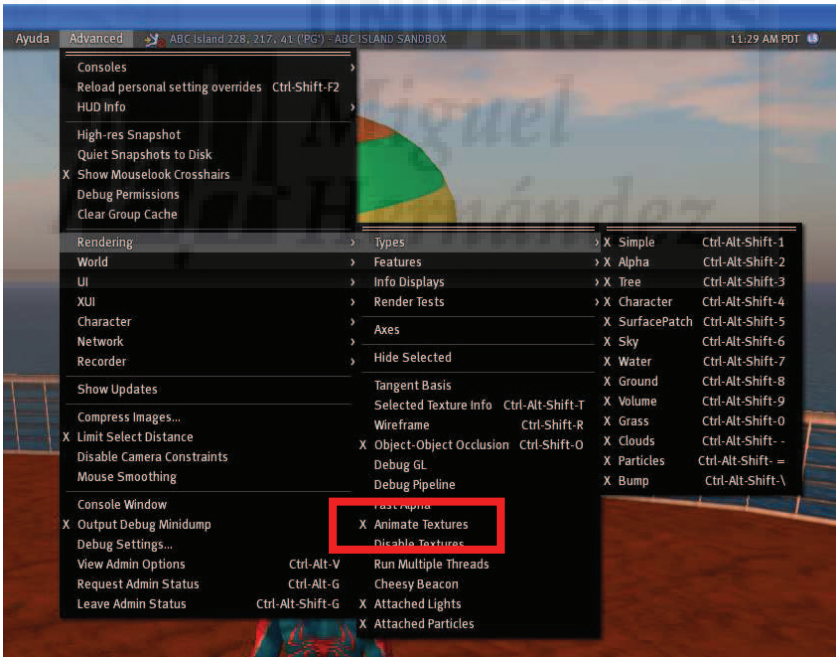


Imagen 6.4.cp.3.1: Habilitar las texturas animadas

1.1. Comprobaremos que Second Life permite las texturas animadas. Para ello pulsar Ctrl + Alt +D para que aparezca un menú que normalmente está oculto llamado “Advanced” y que no está traducido al español.

1.2. En este menú accederemos a “Rendering” y debemos observar un aspa a la izquierda de “Animate Textures” tal como muestra la anterior imagen.

1.3. Subiremos la imagen para texturizar nuestro objeto si no la tenemos en nuestro inventario todavía. Para ello haremos Archivo > Subir > Imagen.

En la siguiente imagen se puede observar la textura utilizada:

Imagen 6.4.cp.3.2: Textura para animar

1.4. Una vez en Second Life y en un Sandbox que nos permita crear y editar objetos, pulsamos con el botón derecho del ratón y en el menú contextual que aparece elegimos Crear.

1.5. Elegimos una esfera y pulsamos sobre el suelo con el botón izquierdo del ratón.

1.6. Accedemos a la pestaña “Objeto” y modificamos su tamaño para que valga 3 metros en los tres ejes: x, y, z.

1.7. Vamos a la pestaña “Contenido” y pulsamos sobre el botón “Script nuevo”. Esto hará que aparezca el icono que representa un nuevo script. Hacemos doble clic sobre él para abrirlo.

1.8. Escribimos el siguiente script.

```
default
{
   state_entry()
   {
      llSetTextureAnim(ANIM_ON|SMOOTH|ROTATE|LOOP, ALL_SIDES, 2,2,1, 0, 0.5);
   }

}
```

1.9. Pulsamos sobre los botones de “Guardar” y “Reiniciar”.

Ya debe funcionar, lo que veremos es algo parecido a lo que se visualiza en la siguiente imagen donde podemos ver tres momentos de la transformación de la textura sin que veamos moverse ni girar la esfera ni nosotros ni nuestro punto de vista.

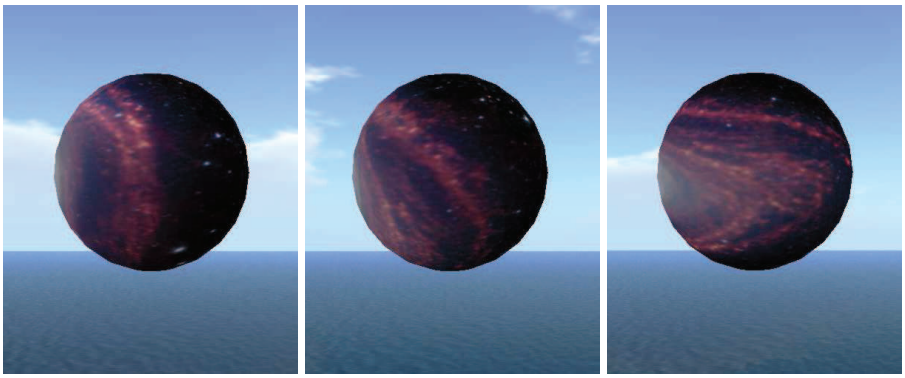


Imagen 6.4.cp.3.3: Textura animada por transformación

2. Crear una textura animada por intercambio.

Hacer que una textura se intercambie por otra no está resuelto directamente por una función de LSL, sino que tenemos que implementar un script que lo realice. Los fundamentos teóricos de este script se encuentran bastante detallados en la unidad teórica correspondiente, pero lo más importante es tener en cuenta que las texturas que se intercambiarán se encuentran en la pestaña "Contenido" del objeto a texturizar, que utilizaremos la función llSetTexture para aplicarla y llSleep para controlar el tiempo que cada textura permanecerá en el objeto 3D.

Los pasos a seguir para crear un intercambio de texturas son los siguientes:

1. Localizar y renombrar en serie las texturas que queremos utilizar. Este trabajo lo llevaremos a cabo en el Inventario aunque para comprobar como quedan sobre el objeto en cuestión debemos editarlo y aplicarla en la pestaña "Textura".

2. Vamos a la pestaña "Contenido" y pulsamos sobre el botón "Script nuevo". Esto hará que aparezca el icono que representa un nuevo script. Pulsamos doble clic sobre él para abrirlo.

3. Escribimos el siguiente script.

```
float tiempo=1.0;          // tiempo de intercambio de una textura por otra
integer numtext=3;         // nº de texturas que componen la textura animada y que están en content
integer cara=0; // 0 será la cara a texturizar, si queremos todas mirar la línea llSetTexture
integer i;                 // contador para saber por que textura vamos dentro del bucle

default
{
   state_entry()
   {
      for(i=0;i<numtext;i++)
      {
         llSetTexture(llGetInventoryName(INVENTORY_TEXTURE, i),cara);
                  // pone la textura que toque dentro del content del objeto
                  // si queremos todas poner ALL_SIDES en lugar de "cara"
         llSleep(tiempo);
      }
      llResetScript();
   }
}
```

9. Pulsamos sobre los botones de "Guardar" y "Reiniciar". Una vez realizados estos pasos debemos ver cómo cada segundo la textura del objeto en cuestión cambia como muestra las imagen inferior.

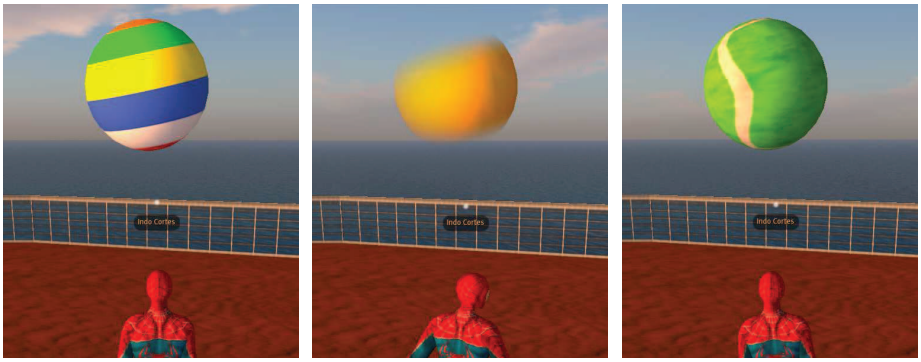


Imagen 6.4.cp.3.4: Textura animada por intercambio

Conclusiones:

Como podemos comprobar, crear texturas animadas con uno u otro método, es realimente fácil y puede resolvernos muchos problemas y además nos permite crear algunos efectos especiales que no se podrían realizar de otra forma.

Por otra parte, como las dos soluciones suponen la creación de un script hace que sea una solución muy práctica, muy manejable, la podemos utilizar cuantas veces queramos y es bastante productiva ya que no aumenta en exceso la carga de computación del sistema al estar a cargo del software en la parte del cliente.

Unidad 5: Iluminación en Second Life.

Introducción teórica:

1. Introducción a la iluminación en Second Life.
2. Preparar el cliente SL y posibilidades.
3. Iluminación dinámica.
4. Objetos autoiluminados.
5. Real glow.
6. Objetos que emiten luz.

Casos prácticos:

Caso práctico 5.1: Creación de objetos con autoiluminación y resplandor.

Caso práctico 5.2: Creación de objetos luminosos.

1. Introducción a la iluminación en Second Life.

La iluminación en Second Life es un tema que debemos estudiar en profundidad debido a la estructura de este metaverso. Como ya sabemos, lo que vemos de Second Life se produce desde dos fuentes: nuestro cliente de Second Life instalado localmente en nuestro ordenador y por otra parte, lo que producen los servidores de Linden Lab.

En Second Life no podemos crear luces como en otros programas 3D, ya que están muy limitadas las opciones por el hecho de tener que ser representadas en tiempo real.

Aún así, podemos crear objetos que tengan una superficie clara o brillante que imite a la autoiluminación independientemente del color o textura que tenga. También podemos hacer objetos que emitan una iluminación con resplandor hacia el exterior como la de los luminosos de neón de las tiendas.

Además podemos crear objetos que iluminen a los objetos que los rodean. O sea, se va a simular un objeto que emite fotones y por lo tanto que irradia luz. Por lo tanto podemos crear bombillas y otros objetos parecidos.

Además de todo esto, podemos modificar la iluminación dinámica que ofrece Second Life y por ejemplo, hacer que nunca se haga de noche, o si queremos una cita romántica, podemos hacer que sean las 12 de la noche, el atardecer o el amanecer automáticamente.

De todas formas, debemos decir que la iluminación en Second Life es muy deficiente con respecto a sistemas que no requieren tiempo real y en especial si la comparamos con Director. No tenemos luces realistas ni de distinto tipo como focos, cónicas, etc. Tampoco tenemos sombras, lo que resulta imprescindible para dotar de realismo a una escena. Esto provoca que los usuarios modeladores de Second Life utilicen trucos como rayos de luz realizados con objetos, texturas de sombras, etc.

2. Preparar el cliente SL y posibilidades

Antes de estudiar las posibilidades de iluminación de Second Life, debemos controlar las opciones de configuración que tenemos activadas en nuestro programa cliente.

En el menú Edit, tenemos el comando “Preferences” que muestra las opciones de configuración local de Second Life. Pulsar en la ficha “Graphics” y luego pulsar sobre la opción “Custom” para ver la ventana como en la imagen 6.5.1.

Fijémonos en el grupo “Lighting Detail”. Debemos elegir la 2ª opción: “Nearby local lights”, ya que si elegimos la opción por defecto, esto es, la primera, la llamada “Sun and moon only”, los objetos luminosos no actuaran como tal, y por tanto solo tendremos luz proveniente del sol y de la luna. Esto quiere decir que cuando se haga de noche, solo la luz de la luna iluminará nuestra escena y ningún objeto podrá emitir luz.

Otro tema importante es el que trata la representación del resplandor o glow en inglés. Este es un recurso muy importante para dotar de realismo a nuestro metaverso pero consume muchos recursos de cálculo de nuestro procesador, por ello se llevan a cabo en las unidades de proceso de gráficos de las tarjetas gráficas y solo algunas de las más modernas pueden realizar esta tarea.

Tenemos que poner “Basic Shaders” activo. Esto no solo hará que los resplandores se visualicen (si nuestra tarjeta gráfica lo permite) sino que podemos ver reflejos en el agua y efectos atmosféricos si los activamos. Esto hará que la calidad de representación de la escena aumente mucho como se observa en las siguientes imágenes. Para generarlas hemos puesto la iluminación en “puesta de sol” (sunset).

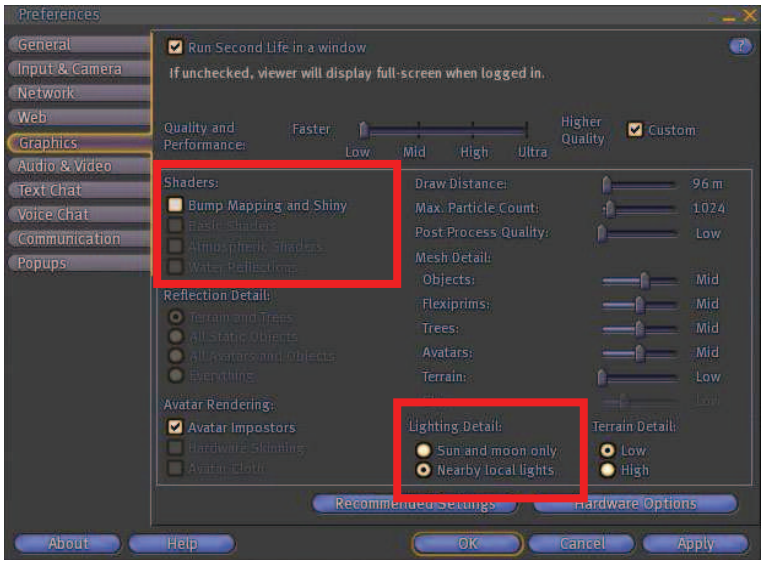


Imagen 6.5.1: Edición de preferencias de luz

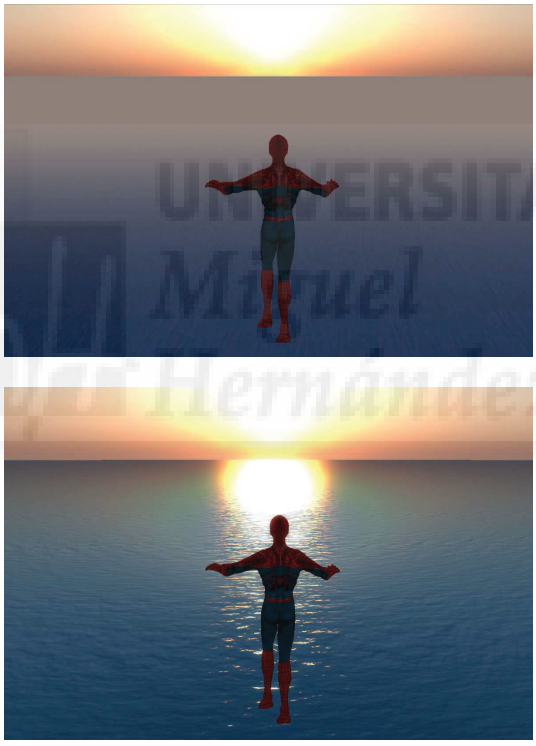


Imagen 6.5.2: Reflejos en el agua al activar Basic Shaders

3. Iluminación dinámica.

La iluminación dinámica es una característica que tiene Second Life para aumentar la sensación de inmersión en este mundo sintético. Consiste en que la luz ambiental va cambiando según la hora del día, por lo tanto, en el tiempo en que estamos en Second Life puede ser de día o de noche y veremos cambios de luz constantes sobre todo a la hora del amanecer y el anochecer.

Este hecho se puede cambiar a voluntad del usuario. Podemos controlar todos los parámetros que nos imaginemos si somos propietarios de terreno. Si no es así, no tendremos todas las posibilidades al completo.

Los cambios los podemos efectuar desde el menú World (Mundo) en el grupo de comandos “Enviroment Settings”. Los posibles comandos para cambiar a una hora determinada son: Sunrise, Midday, Sunset, y Midnight, es decir, amanecer, mediodía, puesta de sol y medianoche. También tenemos la opción de “Revert to Region Default” que hará que la iluminación ambiental dinámica vuelva a la hora que le corresponde.

Por último, el comando “Enviroment Editor” que nos proporciona unas posibilidades muy grandes para editar el cielo, el sol, las nubes, etc. En las siguientes imágenes se puede observar una misma escena a distintas horas del día, lo que da una idea de la sensación de inmersión conseguida con esta característica.

Imagen 6.5.3: Iluminación ambiental dinámica

4. Objetos autoiluminados o “Full Bright”.

Los objetos pueden presentar un efecto de autoiluminación que se visualizará como el color o textura base pero con más cantidad de blanco.

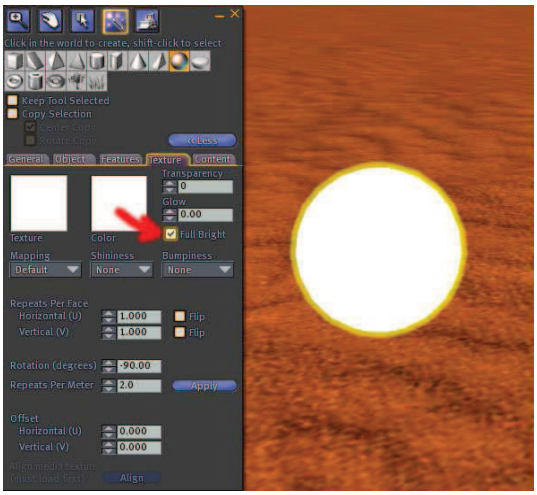


Imagen 6.5.4: Activación del “Brillo total”

No podemos elegir un porcentaje de brillo ya que es un control booleano, por tanto si queremos un brillo más tenue deberemos buscar otras soluciones como modificar las texturas.

El brillo total lo podemos activar al editar el objeto en la ficha Texture. La anterior imagen muestra una flecha roja sobre el control de activación.

En la siguiente imagen podemos apreciar tres esferas. La primera es una esfera con la textura por defecto. A la segunda se le activó el brillo total y a la tercera también, pero en este último caso, se quitó la textura de madera, con lo que el cambio es más radical.

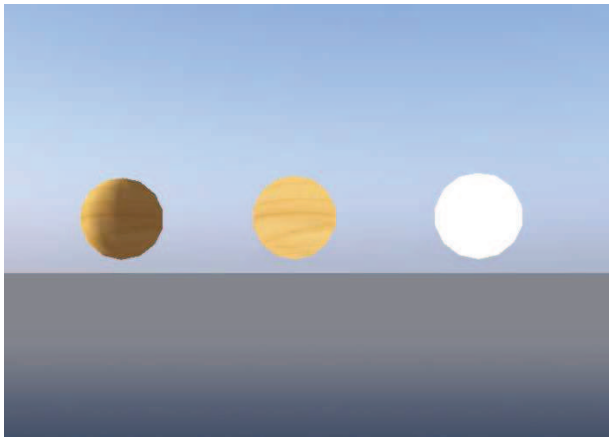


Imagen 6.5.5: "Brillo total" con y sin textura

5. Real Glow

Lo que se conoce como "Real Glow" o resplandor real es un efecto que trata de imitar a las luces de neón y efectos parecidos.

Como hemos estudiado antes, es necesario configurar nuestro cliente de Second Life para que podamos visualizarlo en Edición > Preferencias > Gráficos > Basic Shaders.

En la imagen que sigue se puede apreciar tres objetos compuestos a su vez por dos objetos unidos. En el objeto de la izquierda no hemos aplicado resplandor. En el objeto de en medio hemos aplicado un resplandor intermedio. El objeto de la derecha muestra el máximo efecto de resplandor que admite Second Life.

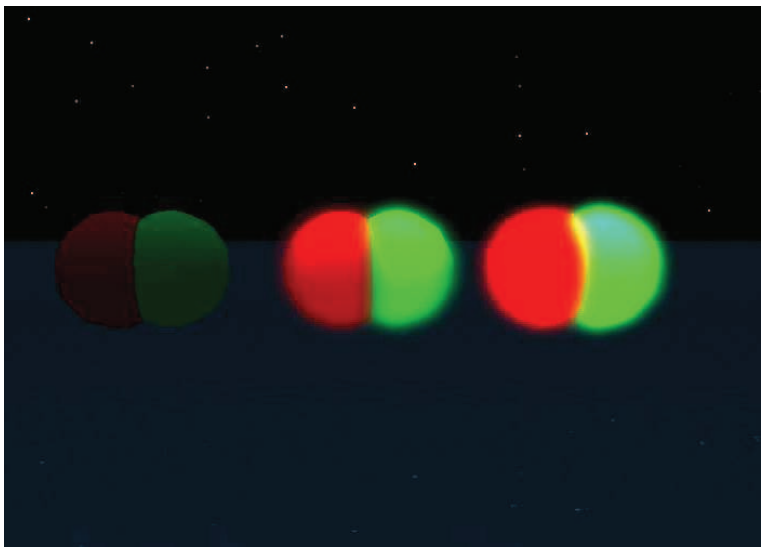


Imagen 6.5.6: Tres niveles de Real Glow

Queremos señalar la interacción de colores que se aprecia en el resplandor entre los dos objetos, creando parte del resplandor en amarillo, es decir, la mezcla de los resplandores verdes y rojos, lo que aumenta la verosimilitud del efecto.

Si tenemos dudas de la capacidad de nuestra tarjeta gráfica para visualizar los glows, lo podemos comprobar por nosotros mismos de la siguiente forma.

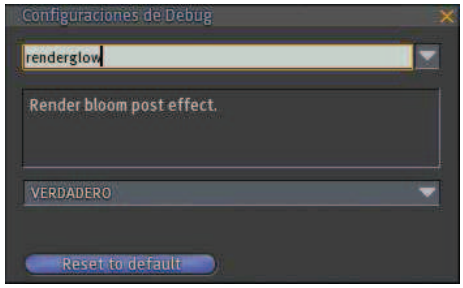


Imagen 6.5.7: Comprobación de la tarjeta gráfica para realizar glows

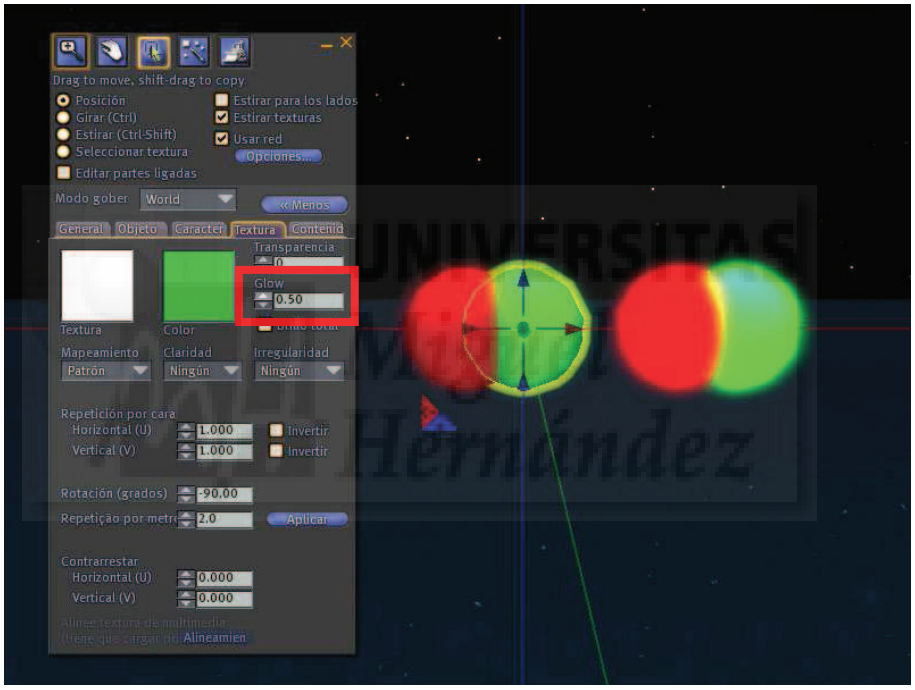


Imagen 6.5.8: Asignar un glow a un objeto

Primero debemos de hacer aparecer el menú “Advanced”. Para ello pulsar Ctrl + Shift + Alt + D. Aquí debemos ejecutar el comando “Debug Settings…” y aparece una caja de diálogo llamada “configuraciones de Debug”. Escribir “renderglow” y si tenemos una respuesta de verdadero quiere decir que podemos ver los resplandores.

Para asignar un resplandor a un objeto debemos de acceder a su edición y en la ficha Textura tenemos el control llamado “Glow” que puede tomar valores desde 0 a 1 que sería el máximo resplandor producido.

6. Objetos que emiten luz.

En Second Life podemos crear objetos que emitan luz. Para ello, lo editaremos y en la ficha Carácter (Features) en el grupo de controles que se encuentran debajo de Leve (para traducción de Light que es como aparece en la versión en inglés). Lo primero que haremos será activar la luz pulsando sobre “Leve”. Los valores que muestran la siguiente imagen son los valores por defecto para los cuatro parámetros que explicaremos a continuación.

Para estudiar los parámetros que se encuentran involucrados en la definición de la luz emitida por un objeto, hemos realizado una serie de elementos para que debido a su geometría y posición relativa, nos permitan visualizar claramente las diferencias de iluminaciones al cambiar el valor de las variables.

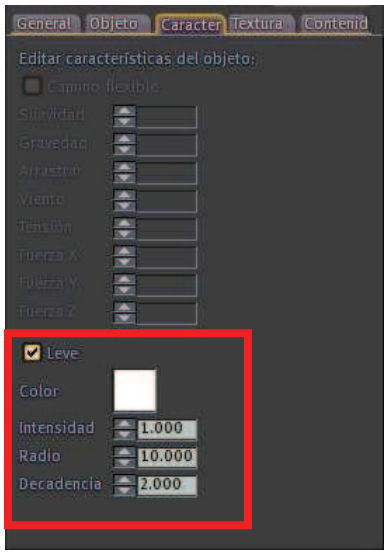


Imagen 6.5.9: Variables para la creación de objetos luminosos

La composición realizada está formada por tres esferas blancas situadas más o menos a la misma altura con respecto al suelo y por dos cubos grises situadas entre las esferas y como a un metro por debajo de la altura de estas. La situación de los cinco cuerpos se puede apreciar en la siguiente imagen que se aprecia oscura por que aún no hemos implementado ningún tipo de luz.

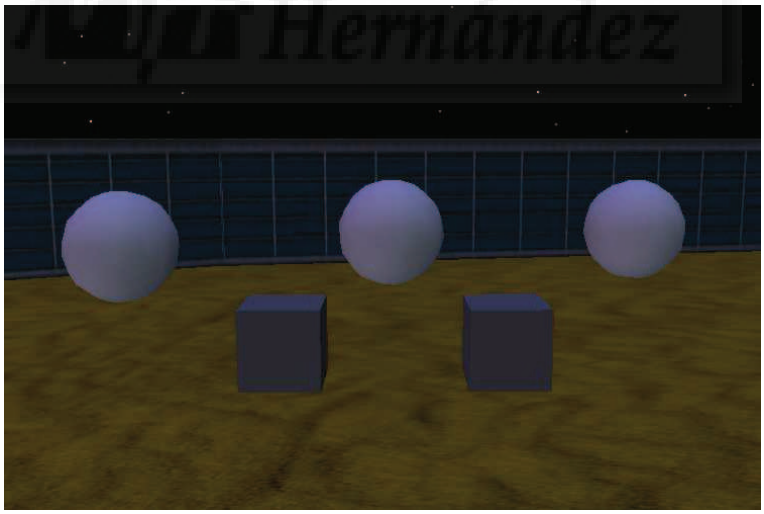


Imagen 6.5.10: Composición para estudiar las variables de iluminación

Color: define el color de emisión de la luz. El color por defecto es el blanco. No hay limitaciones del color y por lo tanto podemos elegir cualquiera aunque el color de luz más normal tanto en la vida real como en Second Life es el blanco.

En la imagen de abajo se puede observar como queda la escena al aplicar a la esfera de la izquierda un color de luz roja, la del centro emite luz blanca y la de la derecha arroja una luz verde. Todas a la máxima intensidad. El efecto es que las esferas emisoras de luz no se tiñen

con su propia luz pero la esfera que está en el centro muestra el color de la luz que le llega desde cada uno de los lados.

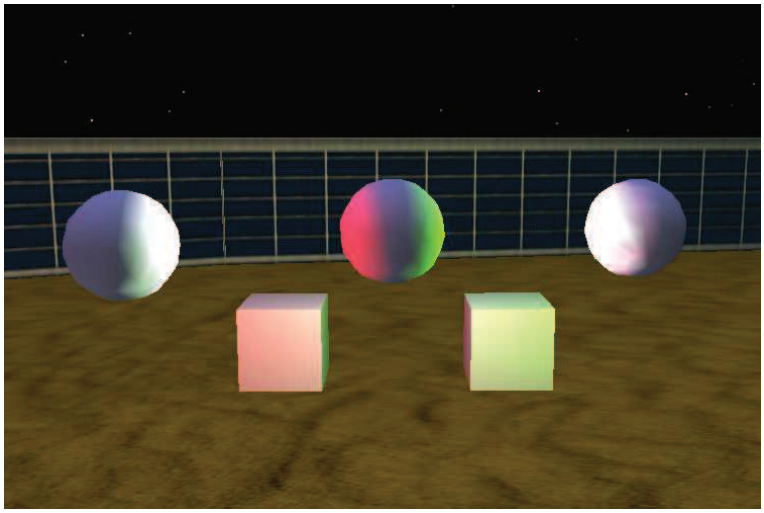


Imagen 6.5.11: Color de la luz

Intensidad: esta opción indica la fuerza de la luz, la cantidad de “fotones” que va a irradiar la fuente de luz. Es evidente que como esto es un simulador, la intensidad se visualizará por tonalidades más blancas en los objetos iluminados. Su valor va desde 0 que indicará que no hay luz, hasta 1 como valor máximo.

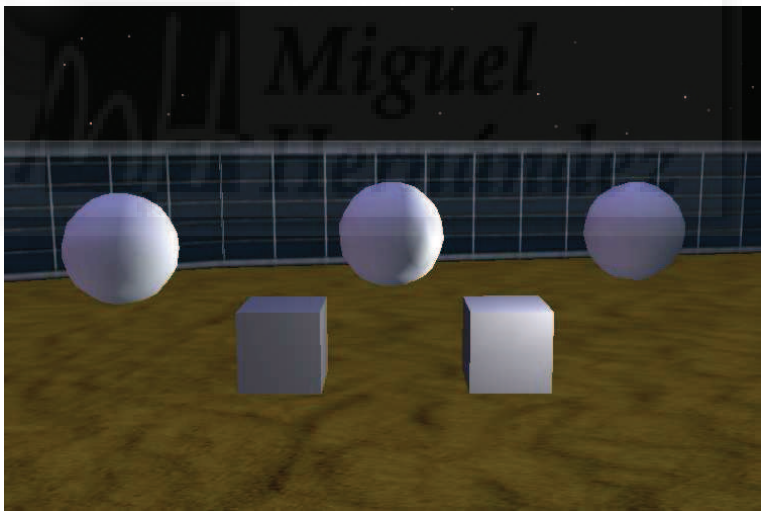


Imagen 6.5.12: Intensidad de la luz

En la escena de la imagen precedente, la esfera de la izquierda tiene una intensidad de 0.1, la esfera del centro no tiene luz y la de la derecha a la máxima intensidad, a 1, por eso, no es iluminada y sí ilumina a los demás cuerpos en su proximidad, Obsérvese como el cubo más próximo está más iluminado.

Radio: sirve para indicar la distancia máxima a la que llegará la luz, es decir, el radio de una esfera que será rellena de luz y cuyo centro será el objeto luminoso. Su valor por defecto es 10 metros, aunque puede variar de 0 a 20 metros.

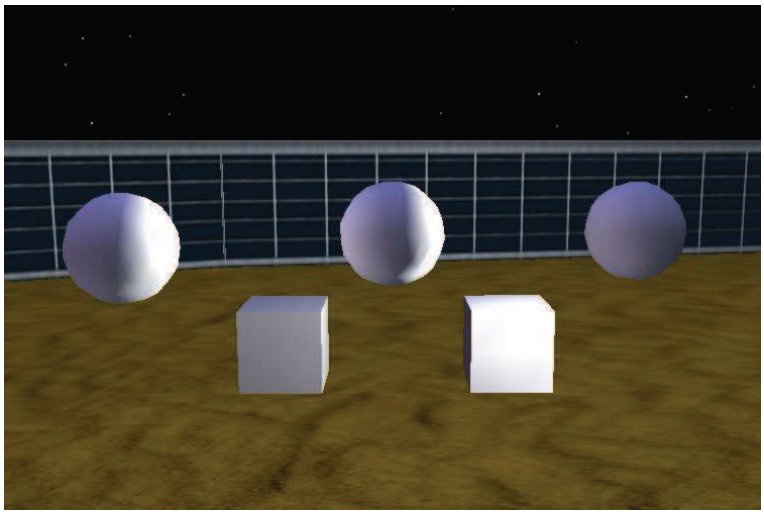


Imagen 6.5.13: Radio de la luz

En la imagen de arriba se visualiza una escena donde la esfera de la izquierda emite luz a la máxima intensidad pero el radio está a 1 metro, la esfera del centro no emite luz y la de la derecha esta a la máxima intensidad pero con el máximo de radio, o sea, 20 metros. Por ello ilumina con la misma cantidad de fotones pero “llegan” más lejos. El efecto es evidente: ilumina más.

Decadencia o Falloff: esta variable indica la difuminación de la iluminación en los bordes de la esfera iluminada. O sea, lo que define son los metros que hay para que la fuerza de la iluminación se diluya y al final sea nula. Esto proporciona un método sencillo para obtener cierto grado de realismo en una iluminación ambiental. Su valor puede variar de 0 a 2 metros.

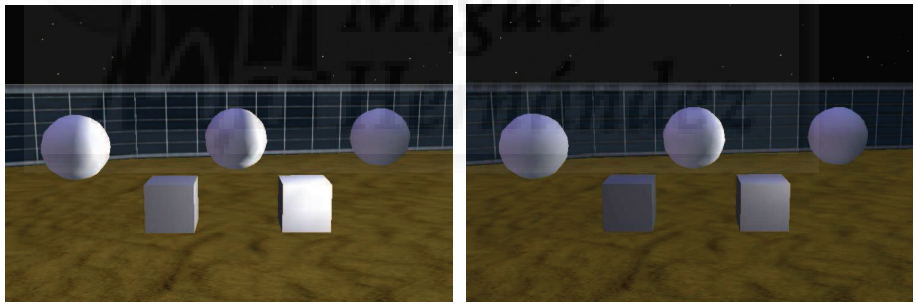


Imagen 6.5.14: Decadencia o radio de la luz

La esfera de la izquierda y la del centro no emiten luz y la de la derecha se encuentra a la máxima intensidad y a 10 metros de radio pero en la parte izquierda se observa sin decadencia (valor 0) y en la derecha con la decadencia máxima (2 metros). Esto explica que aunque la luz llegue hasta los 10 metros en ambos casos, empieza a difuminarse a los 2 metros y por tanto no ilumina tan claramente a los objetos que se encuentran a su alrededor.

Caso práctico 5.1: Creación de objetos con autoiluminación y resplandor.

Objetivo: Crear objetos cuya superficie muestra luz interior y un resplandor de tipo neón.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Crear Autoiluminación.
2. Crear resplandor

1. Autoiluminación.

Para iluminar la superficie de un objeto realizaremos los siguientes pasos:

1.1. En un sandbox, hacemos clic con el botón derecho sobre el terreno y pulsamos en Create. Pulsamos sobre el prim Esfera y la llamamos “Esfera_Brillo”.

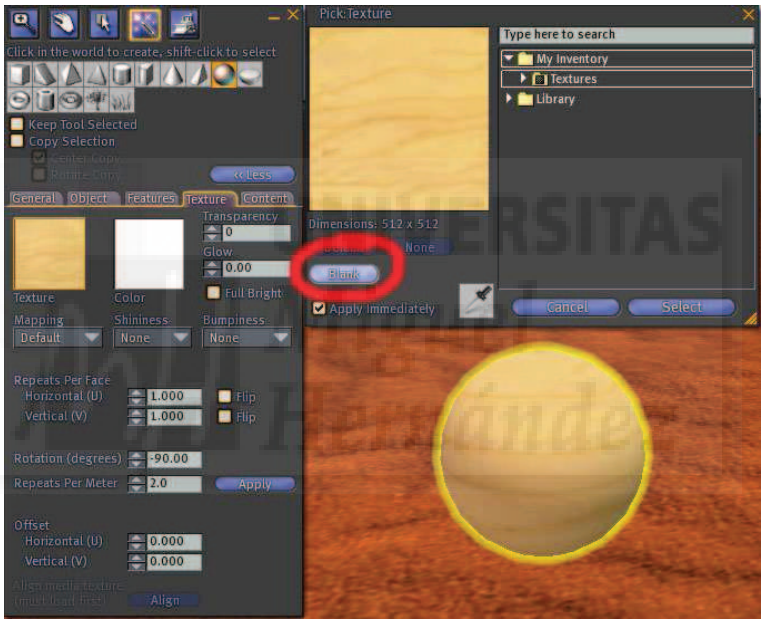


Imagen 6.5.cp.1.1: Quitar la textura por defecto

1.2. Pulsamos sobre la ficha “Texture” y luego sobre el botón “Texture”. En la ventana que aparece pulsamos sobre el botón Blank, para no utilizar ninguna textura y apreciar mejor el efecto tal como muestra la imagen precedente.

1.3. En la ficha “Textura” existe una opción llamada “Full Bright” que hemos señalado con una flecha roja en la imagen anterior. Una vez pulsada, se puede observar el cambio entre antes y después de pulsarlo en la imagen que sigue. Este efecto de día se observa como un cambio de la cantidad de “blanco” que se visualiza. Con otros colores, el efecto es que se vuelven más claros. Por la noche se aprecia como si tuviera luz interna. El color elegido puede ser cualquiera, en este caso, es totalmente blanco.

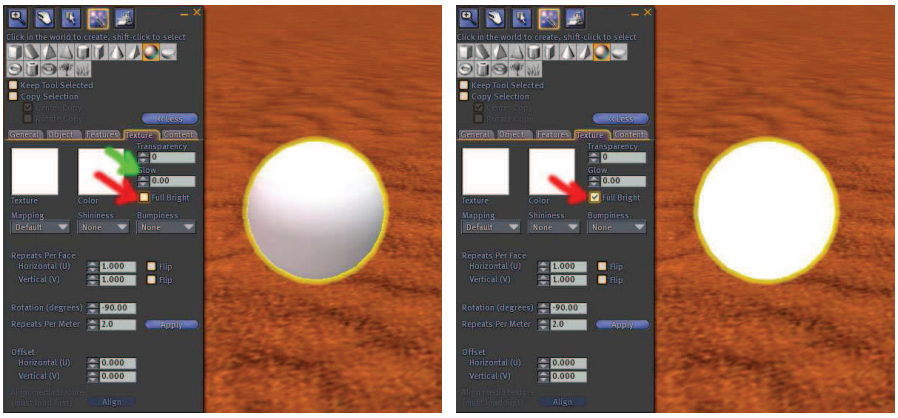


Imagen 6.5.cp.1.2: Esfera antes y después de aplicar Full Bright

2. Crear resplandor

Debemos modificar las opciones por defecto en la configuración gráfica de en nuestro programa cliente de Second Life.

2.1. En el menú Edit, pulsamos en "Preferences". Pulsamos en la ficha "Graphics" y luego pulsar sobre la opción "Custom". Esto hace que aparezcan muchos más controles.

2.2. Lo primero es cambiar en el grupo "Lighting Detail" la opción por defecto al valor "Nearby Local Light".

2.3. También cambiamos en el grupo Shader la activación del valor "Basic Shaders".

2.4. Ahora editamos el objeto que queramos que emita un resplandor y accedemos a la pestaña Textura y en el control "Glow" elegimos una cantidad entre 0 y 1.

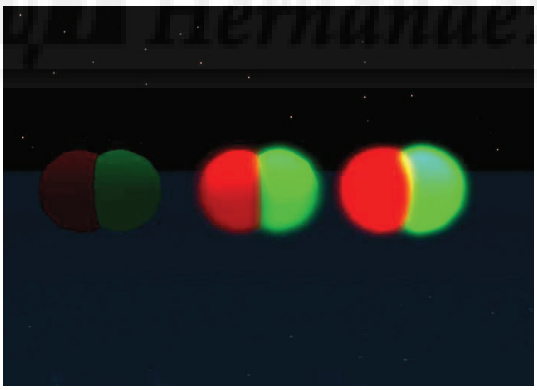


Imagen 6.5.cp.1.3: El mismo objeto con distintos valores de glow

En la imagen precedente se aprecia el mismo objeto pero con valores 0, 0,5 y 1. Es decir, sin resplandor, con uno de intensidad intermedia y más a la derecha con un resplandor de la máxima intensidad que produce Second Life.

Conclusiones:

La creación de objetos con efectos luminosos podemos realizarla fácilmente ya que se trata de controles que se encuentran dentro de la ficha Textura. El problema lo podemos encontrar en la configuración estándar de Second Life a nivel local de nuestro ordenador y en las posibilidades de nuestra propia tarjeta gráfica.

Caso práctico 5.2: Creación de objetos luminosos.

Objetivo: Crear objetos que imitan la producción de luz.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Crear la emisión de luz.

1. Crear la emisión de luz.

Los objetos que irradian luz los vamos a crear desde la edición de objeto. Pero antes de esto tenemos cambiar unos valores en Second Life, ya que por defecto solo se tienen en cuenta la luz del sol y el brillo de la luna.

Imagen 6.5.cp.2.1: Edición de preferencias de luz

1.1. Abrir el menú “Edit” y en “Preferences” pulsar la ficha “Graphics” y luego pulsar sobre la opción “Custom” para ver la ventana como en la imagen 6.5.cp.2.1.

1.2. En el grupo de “Lighting Detail” debemos elegir la 2ª opción: “Nearby local lights”.

Una vez realizado este cambio volvemos al sandbox para realizar un objeto luminoso. En nuestro caso hemos realizado el trabajo cuando ya faltaba luz ambiental para que se observase mejor el proceso. De todas formas, podemos elegir la luz ambiental que deseemos y si nos conviene más, por ejemplo, trabajar a las doce de la noche. Para ello accedemos al menú Word o Mundo y en el comando “Enviroment Settings” elegimos la opción deseada.

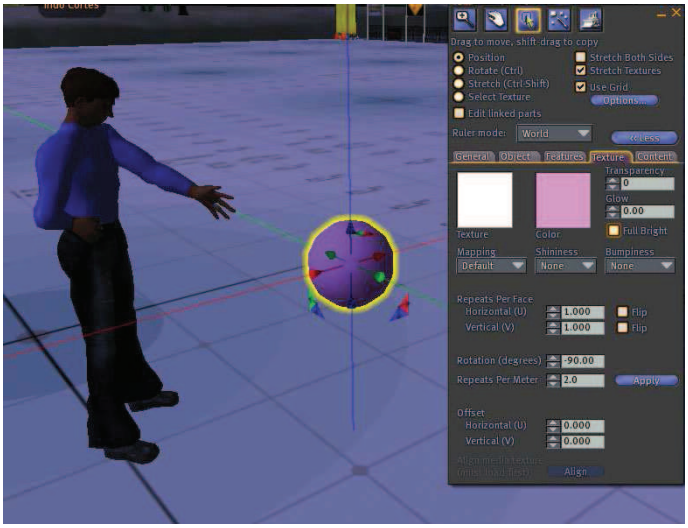


Imagen 6.5.cp.2.2: Creamos el objeto esfera

1.3. Creamos un objeto cualquiera haciendo clic sobre el terreno y eligiendo Create en el menú contextual. Cómo se observa en la imagen siguiente hemos creado una esfera que hemos vuelto brillante siguiendo los pasos del caso práctico anterior.

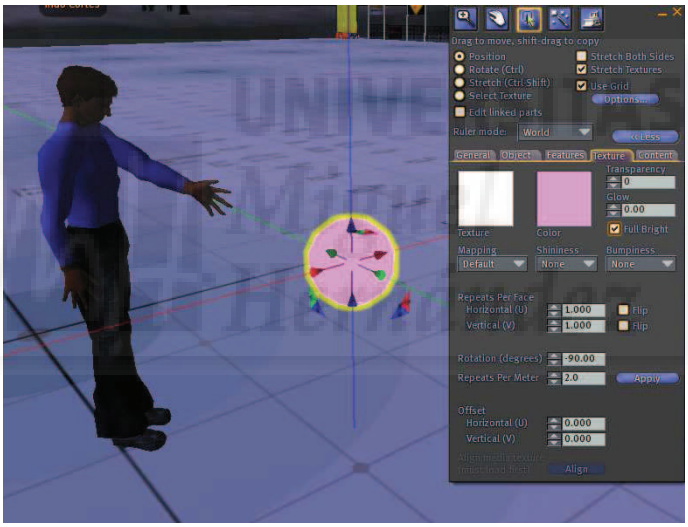


Imagen 6.5.cp.2.3: Hacemos el objeto brillante

1.4. Cómo muestra la imagen siguiente para definir el objeto como luminoso debemos ir a la ficha “Features” y pulsar sobre la opción “Light. Esto hace accesibles las opciones de luz que son el color, la intensidad, el radio y la decadencia, ya explicadas en la unidad teórica.

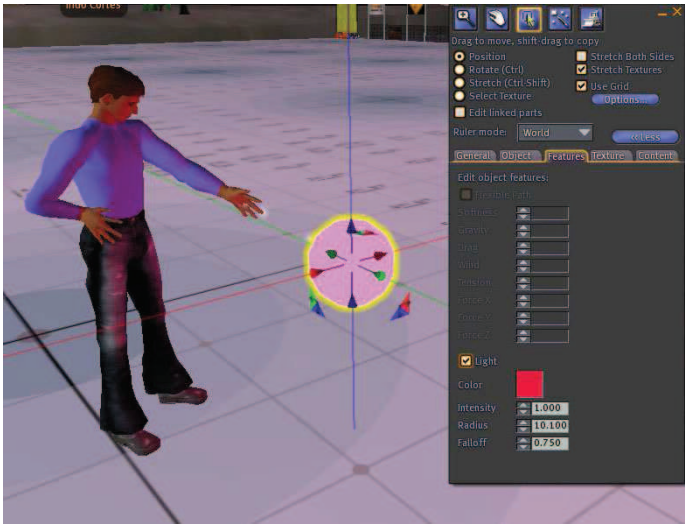


Imagen 6.5.cp.2.4: Hacer el objeto luminoso

Los valores que se pueden apreciar en la imagen de arriba son: color: cereza, intensidad: 1, la máxima, radio: 10, el valor intermedio y decadencia: 0,75.

Cómo podemos observar en la imagen siguiente el objeto sigue poseyendo los valores de iluminación aún al editarlo en cualquier forma. Por ejemplo, nosotros hemos cambiado su bloque básico de construcción para que se vuelva un cilindro a modo de tubo fluorescente, pero esto no le impide seguir iluminando la escena como se puede observar en las distintas caras de la pirámide que se sitúa a su lado. Para poder observar mejor todos estos detalles deberíamos probarlos en un habitación cerrada o con poca luz y con distintos objetos y esquinas de paredes donde la esfera de luz se visualizaría con claridad.

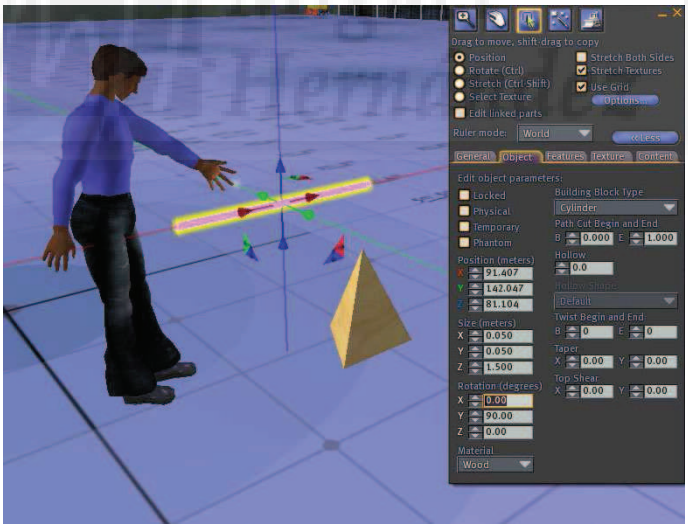


Imagen 6.5.cp.2.5: Modificar el aspecto del objeto luminoso

Conclusiones:

Crear un objeto luminoso no es muy complicado, ya que los valores que podemos modificar son solamente cuatro pero esto no es una ventaja sino un inconveniente ya que pone de manifiesto que Second Life debe mejorar mucho el tratamiento de la luz en su camino de buscar una realidad sintética más parecida a la real. En su descargo podemos decir que este es un tema muy complicado y que consume muchos recursos gráficos del ordenador por lo que no se tratan de momentos las sombras proyectadas ni otros muchos efectos luminosos.

Unidad 6: Cámaras y Machinima.

Introducción teórica:

1. Las cámaras en Second Life.
2. Control de la cámara por defecto.
3. Control de la cámara de ratón.
4. Machinima.
5. Crear una cámara con programación.

Casos prácticos:

Caso práctico 6.1: Control de cámaras.

1. Las cámaras en Second Life.

En Second Life tenemos siempre una cámara que permite la visualización de nuestro avatar y del entorno desde un punto determinado de vista.

Una cámara es precisamente esto, una vista del entorno desde un punto determinado. Imaginemos una película donde se está produciendo una escena. El director puede relatarla desde muchos puntos de vista o cámaras. Nosotros en Second Life también podemos ver lo que nos rodea desde distintas perspectivas.

Cuando nos introducimos en Second Life por primera vez, observaremos el metaverso desde una cámara situada justo detrás de nosotros y un poco más alta, tal como se observa en la imagen siguiente. Esta cámara es la que viene activada por defecto. Es imprescindible que controlemos los movimientos de esta cámara en todas sus opciones para poder manejar lo que nos muestra el programa y poder construir objetos, relacionarnos y en una palabra estar cómodos en este entorno virtual.

Imagen 6.6.1: Vista de la cámara por defecto

En Second Life tenemos dos cámaras o vistas básicas: la cámara por defecto y la llamada "visión del ratón" o "Mouse look". Esta cámara se sitúa en los ojos de nuestro avatar y se manipula de forma distinta a la cámara por defecto. En la siguiente imagen podemos ver cómo se visualiza el mismo paisaje en modo ratón.

Para pasar de la cámara por defecto a la cámara de ratón ejecutamos el comando del menú Ver > visión del ratón. Para volver a la cámara por defecto pulsamos la tecla "Esc".

En la imagen 6.6.2 también se puede apreciar que desaparecen las barras de herramientas y la barra del menú y en la parte izquierda aparece un letrero que dice "press ESC to leave Mouselook". También es destacable que el cursor cambia de forma a un cuadradito pequeño y blanco.

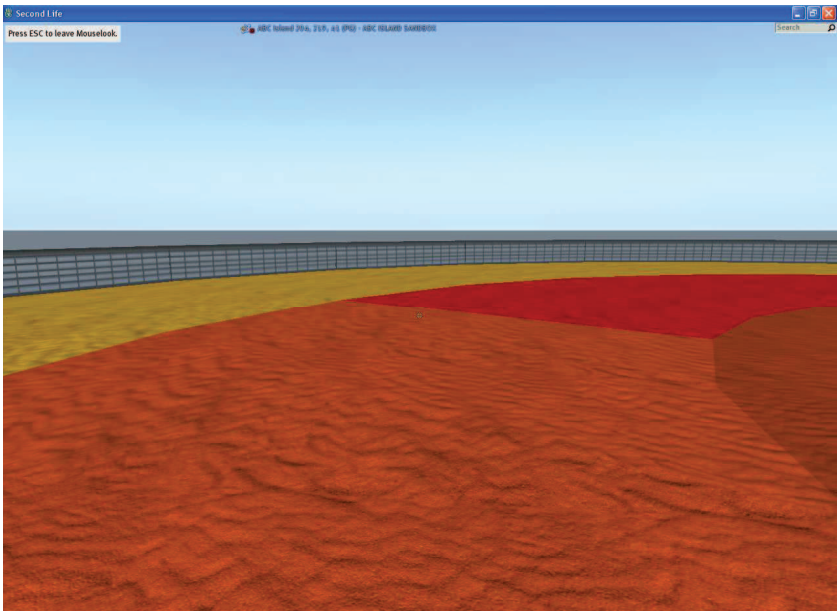


Imagen 6.6.2: Vista de la cámara con el modo "visión del ratón"

2. Control de cámaras.

Con la cámara por defecto tenemos dos opciones de control de la vista: con una combinación de teclado y ratón o bien mediante botones de una caja de diálogo. Antes de ver cómo se maneja vamos a estudiar qué operaciones se pueden llevar a cabo.

Modo Focus: esta operación permite acercarnos a cualquier objeto pulsando sobre él y tiene la propiedad que al mover el ratón hace zoom y al movernos lateralmente giramos la vista pero siempre manteniendo el foco sobre el objeto seleccionado. Cuando pasamos a este modo el cursor cambia a una lupa cuadrada con una cruz en su interior.

Modo Orbital: con esta operación debemos pulsar sobre un objeto de la escena y la cámara girará en cualquier posición sobre ese objeto manteniendo la distancia fija. Podría suponerse que la cámara se puede mover sobre la superficie de una esfera invisible que tiene de radio la distancia de la cámara al objeto. Sirve por tanto para visualizar un objeto a una distancia fija pero desde distintos ángulos. Cuando pasamos a este modo el cursor cambia a una lupa cuadrada con un pequeño dibujo de rotación a su lado.

Modo Encuadre: este modo permite trasladar la cámara horizontalmente y verticalmente pero no en profundidad. Podría decirse que es como si la cámara se trasladase sobre la superficie de una pared invisible. Podemos utilizar esta operación para centrar perfectamente una vista. Cuando pasamos a este modo el cursor cambia a una lupa cuadrada con una cruz de flechas.

Control con teclado. Si se pulsan las teclas de flechas laterales, es decir, la izquierda y derecha, es como si girásemos la cabeza y por tanto la vista de la cámara. Si pulsamos las teclas de las flechas arriba y abajo, el avatar se mueve y por tanto también cambiamos la posición de la cámara que nos sigue por defecto.

En la siguiente tabla se resumen las combinaciones de teclas y ratón necesarias para el control de la cámara por defecto.

Modo	Teclado y ratón	Función
Focus	Alt + botón izquierdo del ratón	Zoom sobre un objeto
Orbital	Alt + Control + botón izquierdo ratón	Orbitar sobre un objeto
Encuadre	Alt + Control + Shift + botón izq. ratón	Encuadre de cámara
Giro	Teclas de flechas izquierda y derecha	Giro a izquierda y derecha

Tabla 12: Control de cámara por defecto con teclado y ratón

En la imagen que sigue se puede observar la caja de diálogo llamada “controles de cámara”. Esta diálogo la podemos visualizar desde el comando Menú > Controles de cámara y aparecerá en la parte superior de manera centrada.

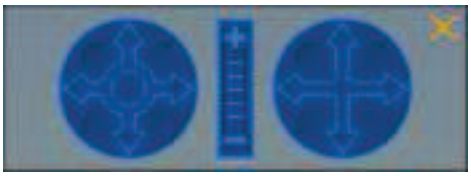


Imagen 6.6.3: Ventana de control de cámara

Se puede observar que tiene 3 controles. El de la izquierda es el “orbital”, el del centro controla el “focus” y el de la derecha controla el “encuadre”. Podemos utilizar esta caja de diálogo para acostumbrarnos a controlar la cámara antes de utilizar el teclado y el ratón.

3. Control de la cámara de ratón.

Hemos comentado anteriormente que la cámara del ratón se sitúa en los mismos ojos de nuestro avatar y por lo tanto, la cámara muestra lo que el propio avatar visualiza desde su punto de vista. El control de esta cámara es muy sencillo e intuitivo ya que al mover el ratón, es como si girásemos la cabeza y por tanto el icono de modo cámara de ratón (el cuadradito blanco) siempre se va a situar sobre el centro de la pantalla, cuando desplazamos el cursor a la izquierda, la vista girará también mostrando otra parte de Second Life y el icono seguirá justo en el centro.

4. Machinima.

El concepto de machinima es relativamente nuevo y por el momento no ha salido mucho fuera del círculo del arte digital. Machinima es una implementación para grabar películas en mundos virtuales tales como videojuegos y metaversos como si se tratara de un documental en el mundo real. Normalmente, los resultados son cortos 3D creados con una estética de videojuegos y por tanto muy actual y de imágenes de síntesis.

Un ejemplo de película machinima se puede observar en la siguiente página web: http://www.bellsandspurs.com/_video/. Esta es una película dirigida e implementada por Eric Call llamada “Silver bells and golden spurs” ambientada en el oeste. En las siguientes imágenes podemos apreciar unas secuencias.

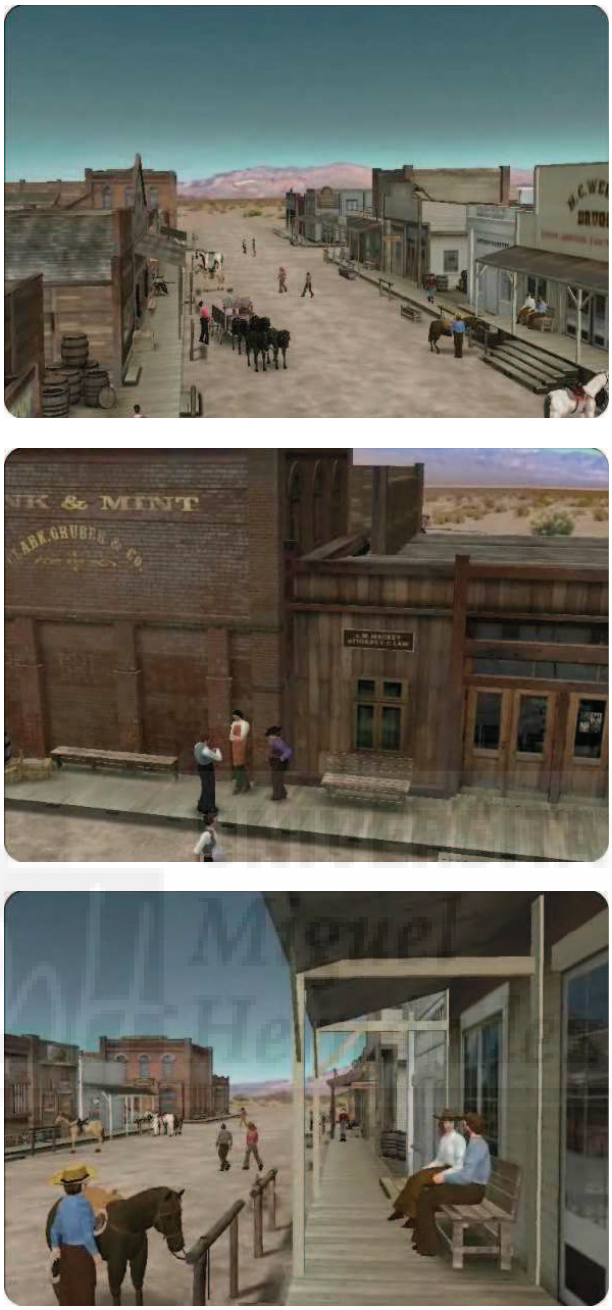


Imagen 6.6.4: Fotogramas de una película machinima creada en Second Life

En Second Life tenemos la ventaja, con respecto a otras soluciones, que el propio sistema nos provee desde el principio de un personaje y de un escenario. Por lo tanto para ambientar una película en el lejano oeste, solo hace falta encontrar una isla con esta ambientación o crearla nosotros mismos como si nuestro metaverso fuera un gigantesco estudio cinematográfico. Además dada la intrínseca capacidad para modificar el aspecto y vestimenta de los avatares, podemos fácilmente quedar con otros usuarios para interpretar escenas de “cine” machinima. Mi propio avatar suele ir vestido como “Spiderman” pero con un par de clicks de ratón se puede convertir en un “sheriff”. Tenemos pues, los actores y el escenario fácilmente, por lo tanto, lo único que faltaría sería un sistema de cámaras y una forma de guardar en vídeo todo lo que suceda en nuestra “toma”.

La segunda cuestión, la de guardar en vídeo lo que sucede en un momento dado de Second Life, ya no se puede hacer directamente desde nuestro programa. En versiones anteriores existía un comando de grabar y guardar en disco local un archivo de vídeo, pero en las últimas

versiones ya no está implementado ya que programas externos como Camtasia ® o Fraps ® tenían mucha mayor aceptación por parte de los usuarios.

Con respecto a tener un equipo de cámaras, tenemos que decir que es posible, aunque las cámaras con mejores características no son gratuitas, sino que las tenemos que comprar. Como ejemplo incluimos los datos (según sus creadores) de una cámara de venta en Second Life llamada “MachinimaCam Pro” con un precio de 2000 dólares Linden.

Control de la cámara:

Puede ser operada por entero con interfaces ocultos.
Controlada por atajos del teclado.
Control de voz.
Control del menú.
Posición bloqueada de la cámara
Atajo del teclado para volver a la posición por defecto de la cámara.
Control de la velocidad para efectos especiales.
Dispositivo visible en pantalla auto-oculto.
Herramientas de efecto integradas.

Efectos de seguimiento de cámara:

1. Cámara fija.
2. Sobre la cabeza.
3. Cámara a ras de suelo.
4. Cámara de lado.

Efectos especiales:

1. Cámara giratoria.
2. Cámara en espiral.
3. Cámara en tornado.
4. Efecto de terremoto.
5. Traveling de cámara.

5. Crear una cámara con programación.

Linden Scripting Language o LSL como se le conoce, proporciona recursos software para dar la posibilidad a los desarrolladores de crear sus propias cámaras.

Lo que sigue es un pequeño script para convertir cualquier objeto en una cámara. El objeto se debe unir al cuerpo del avatar mediante un enlace y la vista del avatar estará ahora a la altura del objeto enlazado. Es un ejemplo de cómo cambiar el modo de cámara en tiempo real.

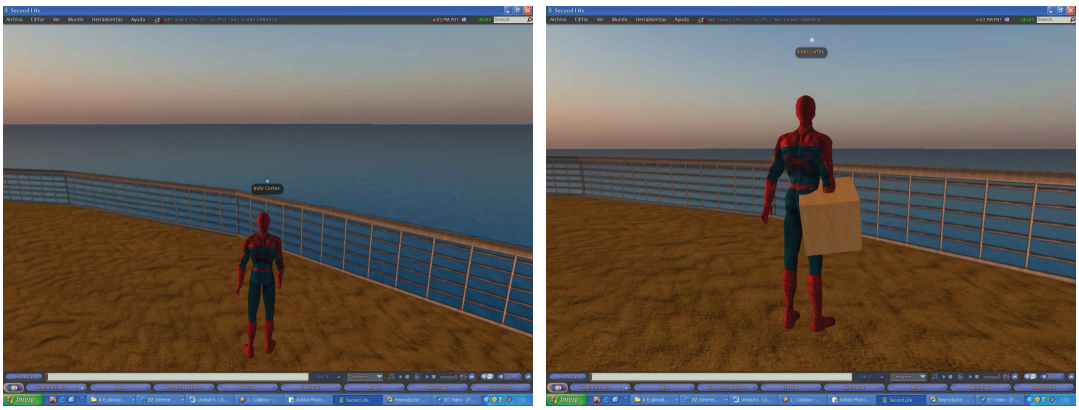


Imagen 6.6.5: Cámara creada por programación

```
// Camera Overide Script by Ben Linden
//
// Put this script on an object, and attach the object.
// Your camera will be changed from default, feel free to tweak as you like.
// This script is an example of changing camera modes in real time.

integer on = FALSE;
integer flying;
integer falling;

list fly_cam_fwd =
[
	CAMERA_ACTIVE, TRUE,
	CAMERA_BEHINDNESS_ANGLE, 0.0,
	CAMERA_BEHINDNESS_LAG, 0.5,
	CAMERA_DISTANCE, 3.0,
	CAMERA_PITCH, 10.0,

	// CAMERA_FOCUS,
	CAMERA_FOCUS_LAG, 0.05,
	CAMERA_FOCUS_LOCKED, FALSE,
	CAMERA_FOCUS_THRESHOLD, 0.0,

	// CAMERA_POSITION,
	CAMERA_POSITION_LAG, 0.5,
	CAMERA_POSITION_LOCKED, FALSE,
	CAMERA_POSITION_THRESHOLD, 0.0
];

list falling_camera =
[
	CAMERA_ACTIVE, TRUE,
	CAMERA_BEHINDNESS_ANGLE, 0.0,
	CAMERA_BEHINDNESS_LAG, 0.5,
	CAMERA_DISTANCE, 3.0,
	CAMERA_PITCH, 0.0,

	// CAMERA_FOCUS,
	CAMERA_FOCUS_LAG, 0.05,
	CAMERA_FOCUS_LOCKED, FALSE,
	CAMERA_FOCUS_THRESHOLD, 0.0,

	// CAMERA_POSITION,
	CAMERA_POSITION_LAG, 0.05,
	CAMERA_POSITION_LOCKED, TRUE,
	CAMERA_POSITION_THRESHOLD, 0.0
];

list walking_camera_fwd =
[
	CAMERA_ACTIVE, TRUE,
	CAMERA_BEHINDNESS_ANGLE, 0.05,
	CAMERA_BEHINDNESS_LAG, 1.0,
	CAMERA_DISTANCE, 3.0,
	CAMERA_PITCH, 10.0,

	// CAMERA_FOCUS,
	CAMERA_FOCUS_LAG, 0.05,
	CAMERA_FOCUS_LOCKED, FALSE,
	CAMERA_FOCUS_THRESHOLD, 0.0,
```

```
        // CAMERA_POSITION,
        CAMERA_POSITION_LAG, 0.05,
        CAMERA_POSITION_LOCKED, FALSE,
        CAMERA_POSITION_THRESHOLD, 0.0
];

default
{
    attach(key id)
    {
      if(id)
      {
            llRequestPermissions(llGetOwner(),PERMISSION_TAKE_CONTROLS |
            PERMISSION_CONTROL_CAMERA);
      }
      else
      {
            llSetCameraParams([CAMERA_ACTIVE, FALSE]);
            llSetTimerEvent(0.0);
            llReleaseControls();
      }
    }

    run_time_permissions(integer perms)
    {
       if(perms)
       {
            llTakeControls(CONTROL_BACK, TRUE, TRUE);
            llSetCameraParams([CAMERA_ACTIVE, TRUE]);
            llSetTimerEvent(0.1);
      }
    }

 control(key id, integer level, integer edge)
    {
        if(!falling)
        {
          if(edge & level)
          {
            if(flying)
            {
              llSetCameraParams([CAMERA_ACTIVE, FALSE]);
            }
          }
          else if(edge)
          {
            if(flying)
            {
              llSetCameraParams(fly_cam_fwd);
            }
            else
            {
              llSetCameraParams(walking_camera_fwd);
            }

           }
      }
    }
```

```
    timer()
    {
        integer agent_info = llGetAgentInfo(llGetOwner());
        if( agent_info & AGENT_IN_AIR)
        {
            if( agent_info & AGENT_FLYING)
            {
                //fly_camera
                falling = FALSE;
                flying = TRUE;
                llSetCameraParams(fly_cam_fwd);
            }
            else
            {
                //falling or jumping
                falling = TRUE;
                flying = FALSE;
                llSetCameraParams(falling_camera);
            }
        }
        else
        {
            falling = FALSE;
            flying = FALSE;
            llSetCameraParams(walking_camera_fwd);
            //on ground, use walk camera
        }
    }
}
```

Como conclusión a esta unidad podemos reincidir en la importancia que tiene el control de las cámaras de las que nos provee Second Life ya que van a ser nuestros ojos dentro del metaverso. Las cámaras serán el método más importante de recibir información de él, la forma de entrada de datos en nuestra dirección y por tanto una herramienta fundamental en un entorno interactivo.

Caso práctico 6.1: Control de la cámara por defecto.

Objetivo: Aprender a controlar los modos de la cámara por defecto en Second Life.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Crear el objeto de referencia.
2. Practicar el modo “focus” de cámara.
3. Practicar el modo “orbital” de cámara.
4. Practicar el modo encuadre de cámara.

1. Crear el objeto de referencia.

1.1. En un Sandbox hacer botón derecho sobre el suelo.

1.2. En la caja de diálogo que aparece elegir caja.

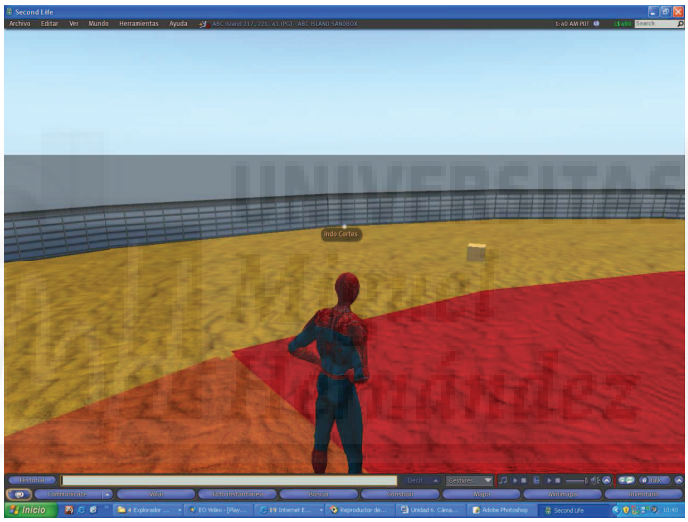


Imagen 6.6.cp.1.1: Creación de un objeto como referencia de cámara

2. Practicar el modo “focus” de cámara.

2.1. Pulsa la tecla “Alt” y con el puntero del ratón haz clic sobre la caja y arrastra el ratón hacia delante o los lados hasta que la vista de la cámara quede como deseamos.

Para poder trabajar mejor dentro de Second Life es imprescindible que podamos acercarnos cuanto queramos a los objetos que nos rodean. Por ejemplo, si tenemos que leer un cartel o crear nuestros propios muebles, necesitaremos acercarnos al objeto de estudio hasta la distancia deseada.

2.2. Para volver a la situación de inicio de la cámara pulsamos la tecla derecha o izquierda y la cámara volverá a situarse detrás de nuestro avatar.

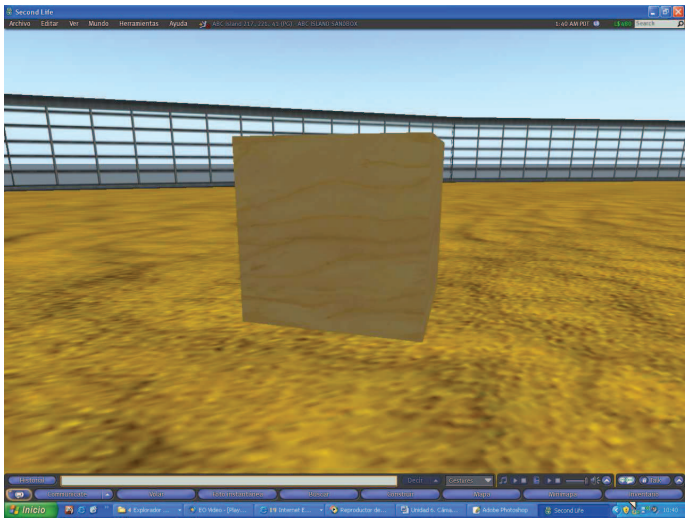


Imagen 6.6.cp.1.2: Focus de cámara sobre el objeto deseado

3. Practicar el modo “orbital” de cámara.

3.1. Desde la situación inicial de la cámara pulsar las teclas Control y Alt al mismo tiempo y hacer clic con el cursor sobre la caja o el objeto que queramos.

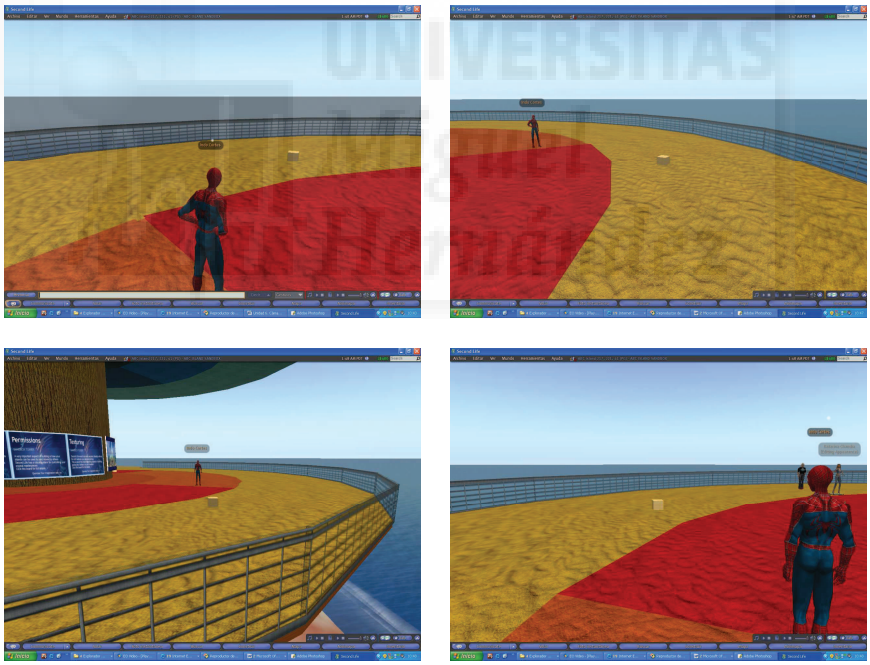


Imagen 6.6.cp.1.3: Modo “orbital” de cámara sobre el objeto deseado

En la sucesión de imágenes de arriba, se puede apreciar como el objeto enfocado, la caja, permanece a la misma distancia de la cámara siempre, pero es la cámara la que se desplaza de manera esférica a una distancia constante, es decir, de forma orbital.

4. Practicar el modo encuadre de cámara.

4.1. Desde la situación inicial de la cámara pulsar las teclas Control, Alt y Shift y hacer clic con el cursor sobre cualquier punto de la pantalla.

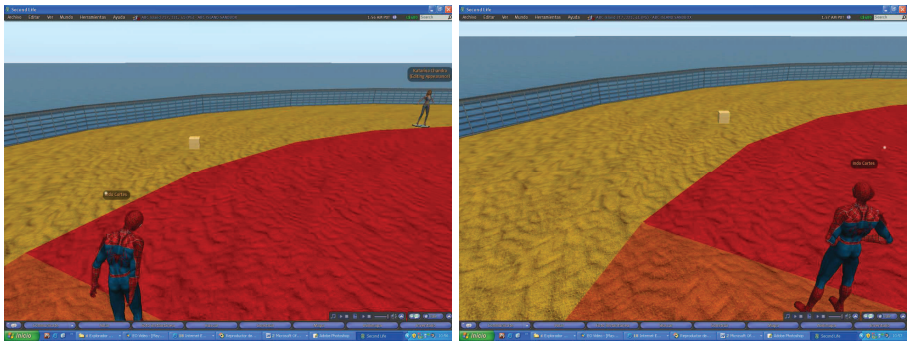


Imagen 6.6.cp.1.4: Modo encuadre de cámara

Observaremos como modificamos la vista arriba y abajo y hacia los lados pero sin cambiar la distancia de la cámara a ningún objeto, sino en un plano con respecto a su posición original. Esto se puede apreciar muy bien en las imágenes de arriba, ya que el avatar tiene el mismo tamaño y sin embargo, la vista ha cambiado.

Conclusiones:

En Second Life estamos utilizando constantemente una cámara. Debemos ser conscientes de los modos de utilización tanto de la cámara por defecto como de cámara de ratón y utilizar todas las posibilidades si queremos poder realizar todas las actividades que nos permite este metaverso.

Unidad 7: Simulaciones físicas.

Introducción teórica:

1. Qué son las simulaciones físicas: máquinas físicas.
2. Cómo realiza SL simulaciones físicas: Havok.
3. Cómo experimentar las simulaciones físicas.
4. Objetos físicos.
5. Objetos flexibles.

Casos prácticos:

Caso práctico 7.1: Crear objetos físicos.

Caso práctico 7.2: Experimento de física.

Caso práctico 7.3: Crear objetos flexibles.

1. Qué son las simulaciones físicas: máquinas físicas.

En todo sistema programado que trate de simular la realidad como son los videojuegos, los metaversos y otros, es necesario tener un sistema para simulación de fuerzas, impulsos, velocidades iníciales y finales, gravedad, etc. que simule la dinámica y cinemática reales. Por supuesto, cuando decimos “reales” nos referimos a todo lo que existe en la realidad a escala humana. Por ejemplo, hechos como el típico problema de física que estudia el choque entre dos bolas de billar, no necesitamos simularlos a nivel atómico o molecular, sino que partimos de dos modelos esféricos del tamaño de una bola de billar real. Aun así, todo lo que nos rodea genera una cantidad de información muy grande, ya que los objetos al interaccionar entre sí, producen encadenamientos de efectos físicos, por ejemplo, una bola de billar en movimiento choca con otra que se encuentra parada y esta a su vez choca con los bordes de la mesa y vuelve a chocar con la primera. Toda esta información hace que se complique su tratamiento y la cantidad de cálculo necesaria para llevarla a cabo crezca exponencialmente. Además, en un metaverso necesitamos controlar fenómenos atmosféricos como el viento que también incidirán en la física de los objetos expuestos a sus efectos.

Por todos estos hechos, los desarrollos que necesitan simular la realidad, integran módulos de simulación que se integran en su estructura pero que normalmente son propiedad de terceras compañías, son lo que se conoce como máquinas físicas.

Una máquina física es un programa que simula interacciones y colisiones newtonianas de objetos en un programa de simulación virtual calculado por ordenador. Son capaces de simular la gravedad, la elasticidad y la conservación del impulso (la inercia entre objetos que colisionan).

Existen dos máquinas físicas muy conocidas en el mercado actual: Havok y PhysX desarrollado por Ageia y actualmente presentes en las tarjetas de NVidia. Havok es, quizás la más conocida y de más amplia difusión en metaversos y videojuegos, mientras que PhysX es la nueva máquina física de la nueva versión de Director de Adobe.

2. Cómo realiza SL simulaciones físicas: Havok.

El mundo virtual de Second Life está compuesto de diferentes partes o módulos que funcionan conjuntamente. Uno de esos módulos es la máquina física.

Second Life necesita ejecutar constantemente la simulación física. Por ejemplo, utiliza la máquina física a su nivel más bajo para distinguir el espacio vacío del espacio lleno. Sin la máquina física, un avatar en el espacio 3D se podría mover a través de las cosas y caería por el suelo (y estaría siempre cayendo si se le aplica gravedad al avatar).

Una máquina física es un componente especial de código que hace simulaciones físicas cuando suceden cosas como:

- Alguien o algo son empujados.
- Alguien o algo colisionan con un otro alguien o algo.
- Alguien o algo que está en movimiento y con impulso.
- La fricción y/o amortiguación frena el movimiento de algo o alguien.

La máquina física nos asegura que el avatar puede caminar por el terreno y además, que al repetir la misma operación lo hará de la misma forma.

Second Life utiliza HAVOK para las simulaciones físicas. Havok es una máquina física producida comercialmente y SL fue construida usando Havok1.

Havok Game Dynamics SDK, es un motor o máquina física utilizada en videojuegos que recrea las interacciones entre objetos y personajes del juego. Detecta colisiones, gravedad, masa y velocidad en tiempo real de los elementos en juego, llegando a recrear ambientes muy realistas y naturales. Havok, en sus últimas versiones, se ejecuta por entero por hardware mediante el

uso de la GPU (liberando así de dichos cálculos a la CPU). Se apoya en las librerías de Direct3D y OpenGL compatibles con Shader Model 3.0.

Algunos juegos que utilizan Havok son: Age of Empires III, Assassin´s Credd, Halo, Half life, Spore, Resident Evil, Star Wars, The Matrix …

La unidad de procesamiento gráfico o GPU (acrónimo del inglés Graphics Processing Unit) es un procesador dedicado exclusivamente al procesamiento de gráficos, para aligerar la carga de trabajo del procesador central en aplicaciones como los videojuegos y/o aplicaciones 3D interactivas.

Direct 3D es parte de DirectX, una API propiedad de Microsoft disponible tanto en los sistemas Windows de 32 y 64 bits, como para sus consolas Xbox y Xbox 360 para la programación de gráficos 3D. El objetivo de esta API es facilitar el manejo y trazado de entidades gráficas elementales como líneas, polígonos y texturas, en cualquier aplicación que genere gráficos en 3D, así como efectuar de forma transparente transformaciones geométricas sobre dichas entidades. Direct3D provee también una interfaz independiente del hardware de aceleración gráfica. Direct 3D se usa principalmente en aplicaciones donde el rendimiento es fundamental, como los videojuegos, aprovechando el hardware de aceleración gráfica disponible en la tarjeta gráfica.

El principal y prácticamente único competidor de Direct3D es OpenGL, desarrollado por Silicon Graphics Inc. OpenGL (Open Graphics Library) es una especificación estándar que define una API multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Fue desarrollada originalmente por Silicon Graphics Inc. (SGI) en 1992 y se usa sobretodo en CAD, realidad virtual, representación científica, visualización de información y simulación de vuelo, etc. También se usa en desarrollo de videojuegos, donde compite con Direct3D en plataformas Microsoft Windows.

PhysX: La PPU (Unidad de Procesamientos de Física) PhysX es un chip y un kit de desarrollo diseñados para llevar a cabo cálculos físicos muy complejos. Conocido anteriormente como la SDK de NovodeX, fue originalmente diseñada por AGEIA y tras la adquisición de AGEIA, es actualmente desarrollado por Nvidia e integrado en sus chips gráficos más recientes. El 20 de Julio de 2005, Sony firmó un acuerdo con AGEIA para usar la SDK de NovodeX en la consola PlayStation 3. Esto provocó que muchos desarrolladores empezaran a crear juegos muy complejos, antes impensables, gracias a esta tecnología. AGEIA afirmó que el PhysX era capaz de realizar estos procesos de cálculos físicos cien veces mejor que cualquier CPU creada anteriormente.

3. Cómo experimentar las simulaciones físicas.

En Second Life podemos experimentar las simulaciones físicas de dos modos: cambiando las características de los objetos y/o programando su comportamiento.

En cuento a la primera opción, podemos decir que es la más sencilla y rápida de llevar a cabo y la que utilizaremos para hacer los casos prácticos. Como ya sabemos, en los paneles de características de los objetos podemos definir su comportamiento básico.

Por ejemplo, en la pestaña objeto (object) tenemos la posibilidad de declarar un objeto como físico. Esto supone que tendrá una masa y que por tanto se verá involucrado en la interacción con la gravedad.

Otro ejemplo se encuentra en la pestaña carácter (Features) donde podemos declarar un objeto como flexible y esto hará que pueda ser mecido por el viento y se pueda doblar sobre sí mismo según unos parámetros fácilmente modificables.

En lo que incumbe a la programación, es justo decir, que son métodos y funciones que no son muy fáciles de controlar, una de las más usadas es “llPassCollisions” que se ejecuta controlada por los eventos “collision start” y “collision end”.

Mención a parte se merecen las funciones que controlan los objetos dinámicos por excelencia: los móviles (coches, motos, cohetes y toda clase de artilugios), que como sabemos son muy utilizados en Second Life. La programación de un móvil de este tipo es una de las tareas más difíciles que puede llevar a cabo un programador en Second Life.

4. Objetos físicos.

Para que un objeto se convierta en físico solamente hace falta que lo declaremos como tal en la pestaña objeto. Otra característica que tenemos que tener en cuenta es la de material, ya que esto informa a Second Life de parámetros físicos como la elasticidad y masa relativa del objeto.

Desde el momento en que se declara una masa para un objeto, si está en el aire, caerá hacia el centro de la “tierra” con una aceleración de 9,8 metros/segundos^2, chocará y tendrá impulso por lo que puede ser que traslade ese impulso a los objetos con los que interaccione en su camino. En la imagen 6.7.1 se señalan los parámetros comentados.

Realizaremos una práctica para experimentar estos hechos con planos inclinados y objetos cilíndricos para ver cómo toman impulso y cómo su energía potencial se convierte en energía cinética y por tanto llegan más lejos cuanto más alto se deja caer el objeto por el plano inclinado.

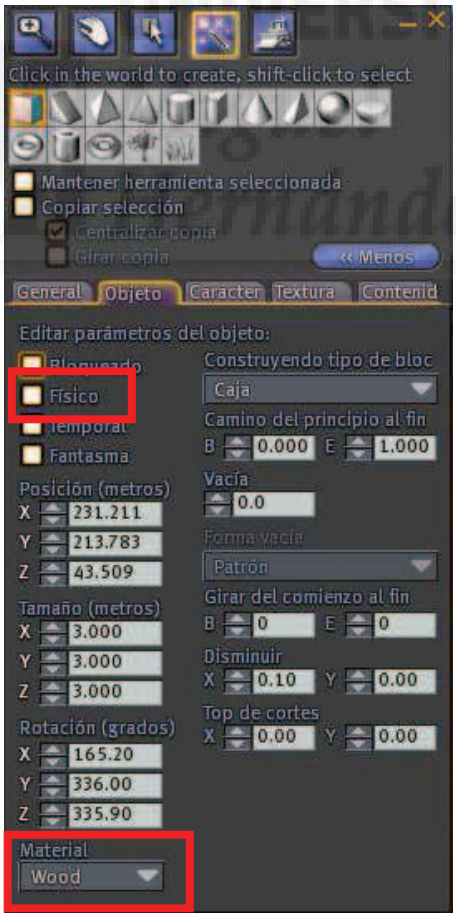


Imagen 6.7.1: Pestaña Objeto

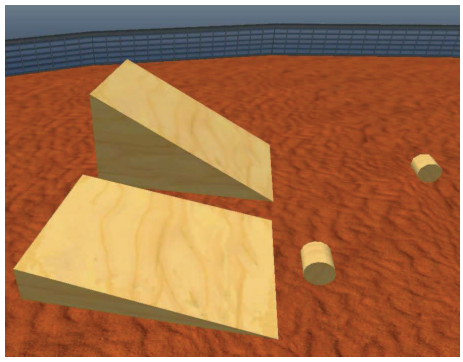


Imagen 6.7.2: Experimento de dinámica con objetos físicos

5. Objetos flexibles.

Hasta ahora hemos trabajado con objetos rígidos, pero con la pestaña “Carácter” podemos hacer pelo, tela, banderas, flecos, ramas, y en general cualquier objeto que tenga cierta flexibilidad. En la jerga de Second Life, a los objetos flexibles se les denomina “flexi”. Cuando un objeto es flexible se convierte automáticamente tanto en físico como en fantasma (que no puede responder a las colisiones) y por tanto podrá ser traspasado por otros objetos o avatares. A continuación describimos los parámetros de flexibilidad de los objetos.

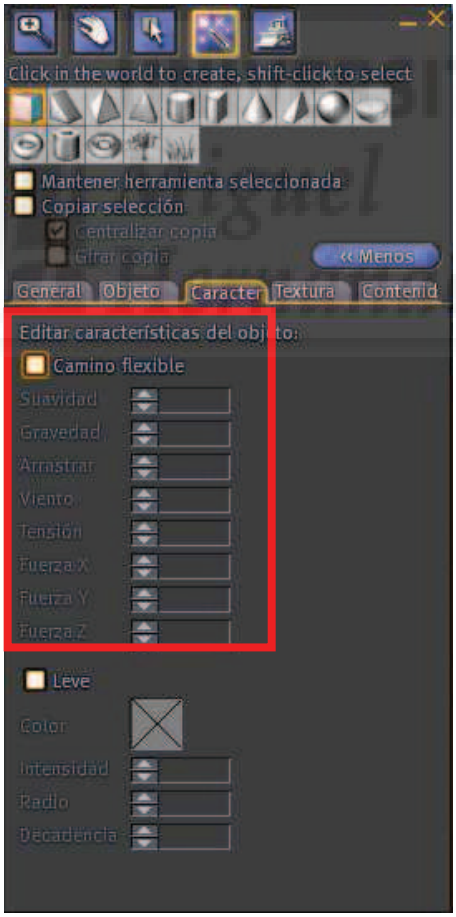


Imagen 6.7.3: Pestaña Carácter

Suavidad (Softness): admite valores de 0 a 3. Sirve para definir cuanto de blando es el objeto. Esto lo hace creando segmentos trasversales, de tal manera que si ponemos una suavidad de X, el objeto tendrá 2^X segmentos que se verán afectados por otros parámetros. Por ejemplo, si

ponemos una suavidad de 3 tendremos 8 segmentos, lo que le hará parecer más “blando” si les afecta la gravedad o el viento.

Gravedad (Gravity): admite valores de –10 a 10. Simula la masa de un cuerpo, de tal forma que cuanto mayor es la cantidad, más rápido caerá. Si el valor es negativo la dirección no será hacia la tierra si no hacia el cielo.

Arrastrar (Drag): admite valores de 0 a 10. Cuanto mayor es su valor, menos cambia la forma del objeto. En realidad mide la rigidez del objeto, si queremos simular tela de tipo seda pondremos el drag a 0, si queremos tela de tipo esparto pondremos el drag a 10.

Viento (Wind): admite valores de 0 a 10. Cuanto mayor es la cifra, más sensible es el objeto al viento del simulador.

Tensión (Tension): admite valores de 0 a 10. Mide lo tenso del cuerpo. Cuanto mayor es, más “flexible” será, más se moverá y más dinámico será el objeto. De alguna manera, se contrapone al parámetro Arrastrar.

Fuerza X, Y, Z: admite valores de -10 a 10 y sirve para hacer que el objeto sufra fuerzas orientadas desde los tres ejes.

La mejor manera de aprender cómo funcionan estos parámetros es construir un objeto muy alto para que tenga la posibilidad de verse afectado por distintos parámetros, por ejemplo, una columna realizada a partir de un simple cilindro. Luego modificaríamos los parámetros y estirando de las flechas de posición, haríamos que el movimiento fuera muy descriptivo de las fuerzas que lo afectan. Realizaremos un caso práctico que ilustre estas características.

En la siguiente imagen se visualiza un ejemplo de la práctica que realizaremos con objetos flexibles.

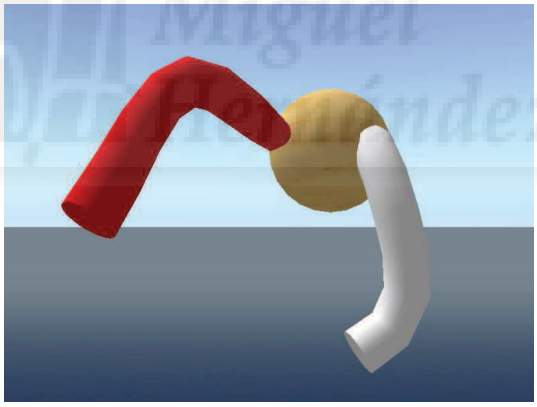


Imagen 6.7.4: Objetos flexibles y rígidos

Caso práctico 7.1: Crear objetos físicos.

Objetivo: Crear objetos que se vean afectados por simulaciones físicas como la gravedad, colisiones, vientos, etc.

Tiempo de realización: 1/4 hora.

Pasos a realizar:

1. Crear el objeto.
2. Hacer que sea de tipo Físico.
3. Comprobación.

1. Crear el objeto.

Esta práctica es muy fácil de realizar debido a que SL permite definir un objeto como físico simplemente activando una casilla.

1.1. Creamos en un sandbox un objeto cualquiera, en este caso una caja.

1.2. Ahora pulsamos Shift + Ctrl. Para que aparezcan los controles de escala y alargamos el objeto para que sea muy alto.

Este objeto podría ser un pilar de un edificio o el tronco de un árbol, cualquier cosa que se nos ocurra y que tenga el centro de masas bastante elevado.

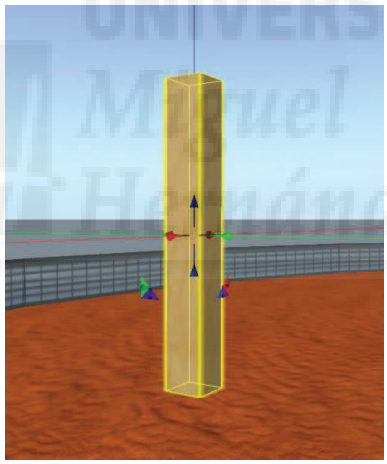


Imagen 6.7.cp.1.1: Creación de un objeto con el centro de masas elevado

2. Hacer que sea de tipo Físico.

2.1. Para editar el objeto, hacer botón derecho > Editar > ir a la pestaña Objeto.

2.2. Entre los parámetros del objeto activar la casilla “Físico”, tal como muestra la imagen que sigue a continuación.

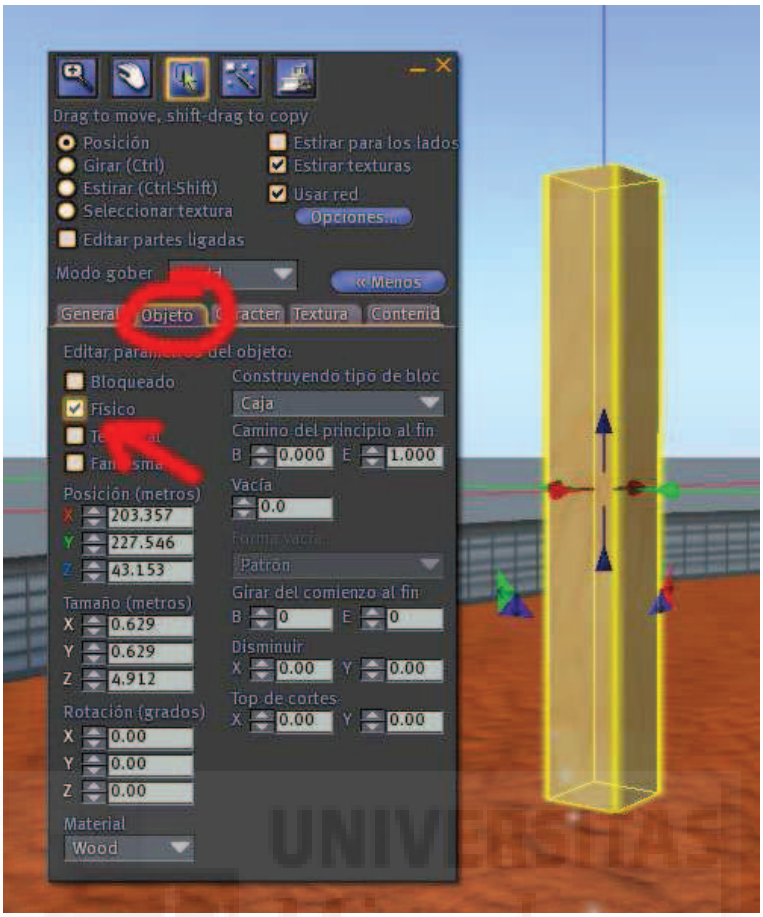


Imagen 6.7.cp.1.2: Definición del objeto como Físico

3. Comprobación.

Para comprobar que es un objeto físico podemos intentar traspasarlo con el avatar y comprobaremos que no podemos, que chocamos contra él. Otra forma de verificar que es físico es hacer que caiga “por su propio peso”. Para ello, situaremos el objeto en una posición inestable y entonces debería caer por efecto de la gravedad.

Es muy importante que comprobemos estos hechos cambiando el material del objeto, ya que no debería tener el mismo efecto de “rebote” un cuerpo de metal que un cuerpo de plástico.

3.1. En el último control de la ficha “Objeto” cambiar el valor “wood”, por “metal”.

3.2. Mientras lo estamos editando, pulsamos Ctrl para rotar el objeto.

3.3. Sobre el control rojo arrastramos el ratón y lo giramos.

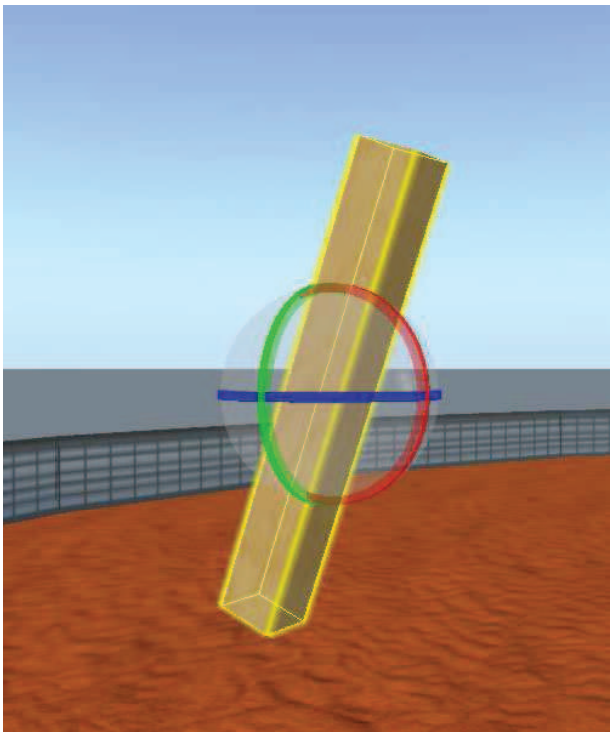


Imagen 6.7.cp.1.3: Giro del objeto para dejarlo en una posición inestable

3.4. En cuanto nos salimos del modo edición, el objeto se ve afectado por la fuerza de la gravedad y cae el suelo como muestra la siguiente imagen.

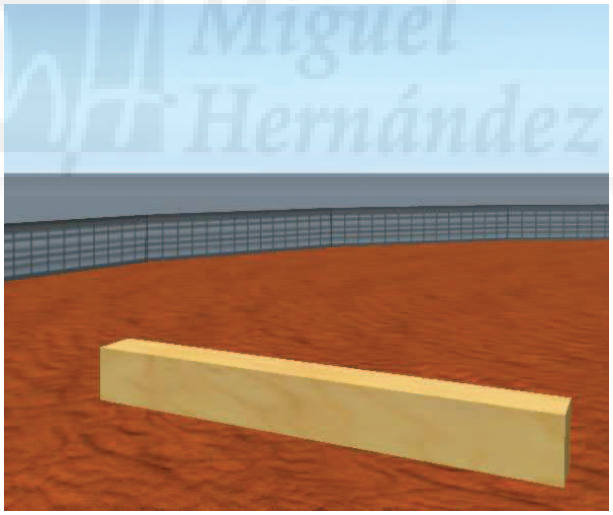


Imagen 6.7.cp.1.4: El objeto en el suelo después de caer

Conclusiones:

Para realizar un objeto que se vea involucrado por fuerzas como el viento o la gravedad, solo debemos activar la casilla “Físico”. Esto es realmente sencillo y automáticamente hace que nuestro objeto sea tomado en cuenta por el sistema de simulación. Por lo tanto, elegir si algo es “terrestre” o no es muy fácil, y por lo tanto, la base para construir dentro de Second Life sistemas de simulación tiene una base muy consistente.

Caso práctico 7.2: Experimento de física.

Objetivo: Poner a prueba la fiabilidad del sistema mediante la simulación de un sencillo experimento físico con planos inclinados.

Tiempo de realización: 1 hora.

Pasos a realizar:

1. Crear los planos inclinados.
2. Crear los cilindros físicos.
3. Realización del experimento.

1. Crear los planos inclinados.

Comenzamos creando un objeto tipo prisma que duplicaremos y posteriormente, cambiaremos de tamaño para que tengan distinta inclinación.

1.1. En un Sandbox, hacer botón derecho sobre el terreno y elegir la opción Crear.

1.2. Elegir el tipo de objeto prisma y pulsar nuevamente sobre el terreno.

1.3. Aparece un prisma que podemos cambiar de tamaño matemáticamente o interactivamente, para ello pulsar las teclas Shift y control a la vez, aparecen unos controles de escala y lo que debemos hacer es aumentarlo en todas las dimensiones.

1.4. Una vez satisfechos con su tamaño, pulsamos la tecla Shift a la vez que arrastramos desde los controles para mover el prisma, esto hará que se arrastre una copia del prisma original. De esta manera tenemos dos prismas idénticos.

1.5. Ahora utilizando de nuevo los controles de escalado, reducimos o aumentamos la altura de un prisma mientras que el otro permanece inalterado. Por lo tanto, tendremos dos prismas de alturas diferentes, tal como muestra la imagen siguiente.

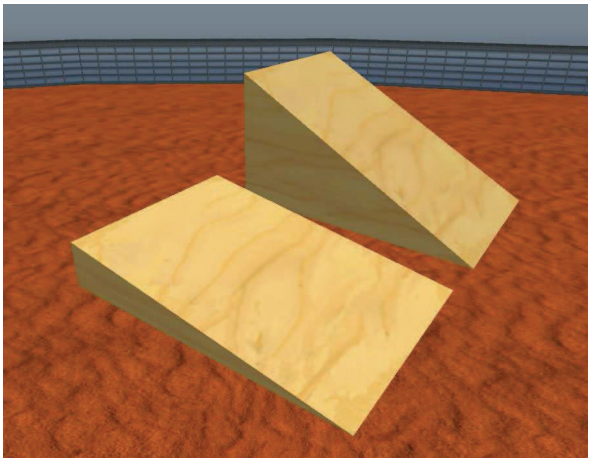


Imagen 6.7.cp.2.1: Creación de dos planos con distinta inclinación

2. Crear los cilindros físicos.

En este punto vamos a crear dos objetos cilíndricos iguales de tipo físico.

2.1. Por el mismo método que en el punto 1, creamos un objeto tipo cilindro.

2.2. Ahora vamos a la pestaña Objeto y en Rotación X, ponemos 90 grados. Esto hará que el cilindro se ponga en línea con el suelo como muestra la siguiente imagen.

2.3. Entre los parámetros del objeto activar la casilla “Físico”.

2.4. Ahora, hacer una copia del cilindro mediante el procedimiento de arrastrar el control de mover mientras mantenemos la tecla Shift pulsada. De esta forma tenemos dos cilindros físicos.

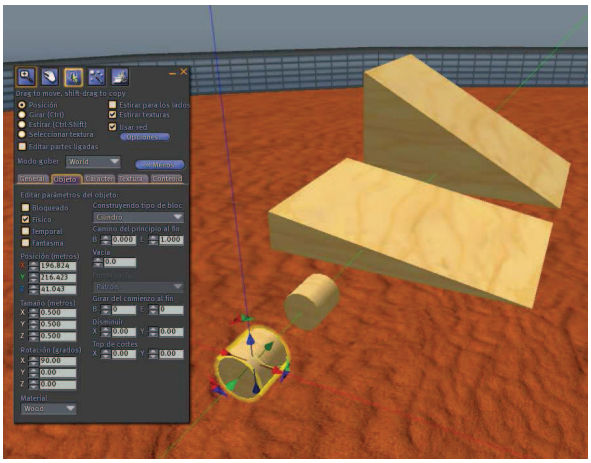


Imagen 6.7.cp.2.2: Creación de dos cilindros físicos

3. Realización del experimento.

3.1. Mover uno de los cilindros a la parte más alta del prisma de ángulo menor.

3.2. Hacer clic sobre el terreno para dejar de editar el objeto.

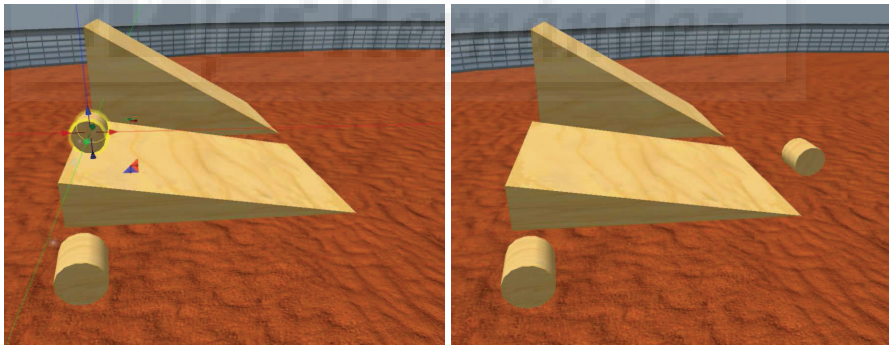


Imagen 6.7.cp.2.3: Movimiento de un cilindro por un plano inclinado

3.3. Observar que el objeto rueda por el plano inclinado y recorre cierta distancia antes de detenerse.

3.4. Ahora repetir los pasos anteriores pero con el otro cilindro y con el plano inclinado de ángulo mayor. Una vez que el cilindro se detiene después de rodar, podremos observar como ha recorrido mayor distancia que en el caso anterior debido a la inclinación mayor de este segundo prisma, lo cual demuestra de forma empírica que la simulación física en Second life funciona.

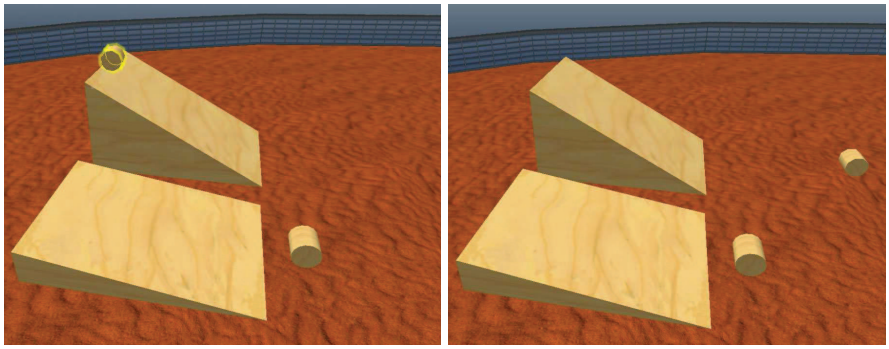


Imagen 6.7.cp.2.4: Repetición del experimento en el plano inclinado en mayor grado

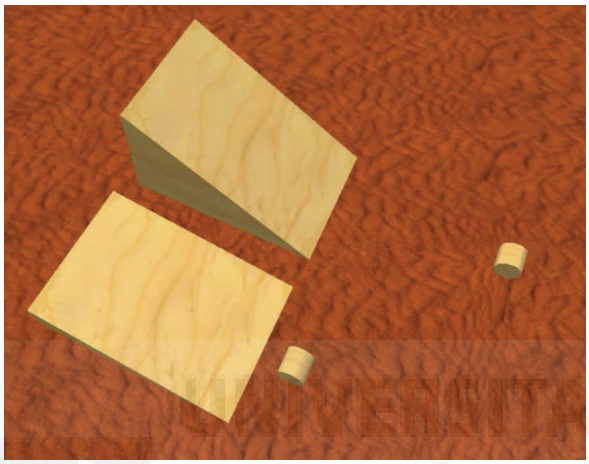


Imagen 6.7.cp.2.5: Repetición del experimento en el plano inclinado en mayor grado

Cómo se puede observar en esta última imagen desde arriba los planos inclinados son de la misma distancia, así pues, la diferencia de la distancia recorrida por los cilindros es proporcional a la diferencia de ángulos entre los planos que recorren.

Conclusiones:

Esta práctica trata de demostrar que nos podemos “fiar” del software que controla la física dentro de Second Life. El experimento tiene una consecuencia y es la previsibilidad y consistencia del software de simulación física y por tanto nos puede servir como base para realizar otros experimentos de física básica, como por ejemplo, de dinámica y cinemática.

Caso práctico 7.3: Crear objetos flexibles.

Objetivo: Crear objetos que sean flexibles y por tanto su estructura se vea modificada en el tiempo debido a distintas fuerzas externas como la gravedad o el viento y a fuerzas internas como la elasticidad y masa.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Planteamiento del objeto final.
2. Creación de los objetos componentes.
3. Establecer la flexibilidad de los objetos componentes.
4. Unir los objetos componentes en el objeto final.

1. Planteamiento del objeto final.

Para visualizar claramente las propiedades de los objetos flexibles, crearemos una especie de pulpo con un núcleo central sin flexibilidad y 2 patas flexibles.

El núcleo lo modelaremos con una simple esfera. Haremos las patas con cilindros, pero mientras a una le asignaremos valores de flexibilidad muy altos, a la otra le asignaremos valores muy bajos. Luego uniremos las 2 patas al núcleo para que al arrastrarlo, las patas se muevan con él y se aprecie con claridad la diferencia de comportamiento debido a la diferente flexibilidad.

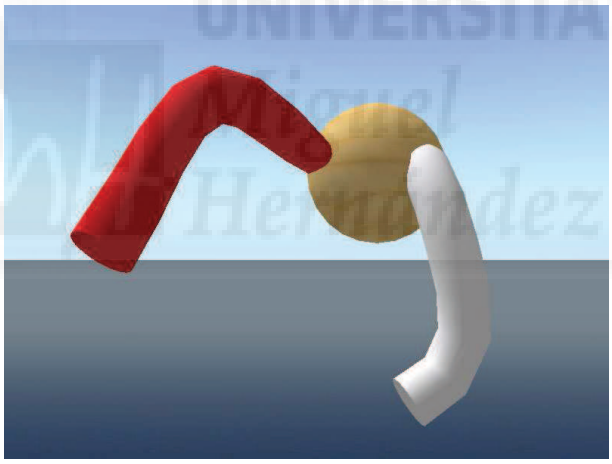


Imagen 6.7.cp.3.1: El objeto flexible terminado

2. Creación de los objetos componentes.

2.1. Pulsar sobre el terreno con botón derecho > Crear > Esfera.

2.2. Cambiar su tamaño pulsando la teclas Shift + Control y al mismo tiempo arrastramos desde alguno de los botones blancos de esquina hasta el tamaño deseado.

2.3. Accedemos a la pestaña “General” y en nombre escribimos “cabeza”.

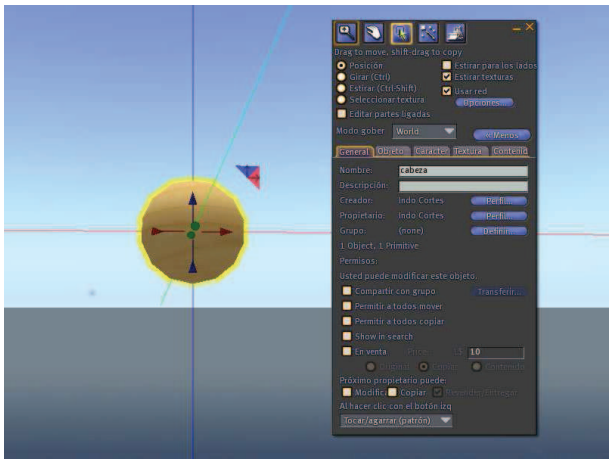


Imagen 6.7.cp.3.2: El objeto rígido "cabeza"

2.4. Pulsamos sobre el terreno con botón derecho > Crear > Cilindro.

2.5. Cambiar su altura pulsando a la vez la teclas Shift + Control y al mismo tiempo arrastramos desde el botón azul inferior hasta alargarlo hasta el tamaño deseado.

2.6. Accedemos a la pestaña General y en nombre escribimos "muy_flexible".

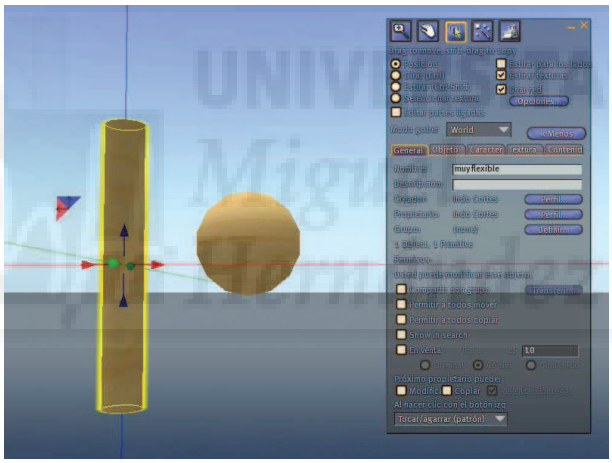


Imagen 6.7.cp.3.3: El objeto "muy_flexible"

Nota: como queremos que las patas sean de forma completamente iguales, crearemos la segunda pata a partir de la primera mediante una copia de esta.

2.7. Mientras mantenemos la tecla Shift pulsada, seleccionar la flecha roja y arrastrar hacia la derecha para crear una copia de "muy_flexible".

2.8. Accedemos a la pestaña "General" y en nombre escribimos "poco_flexible".

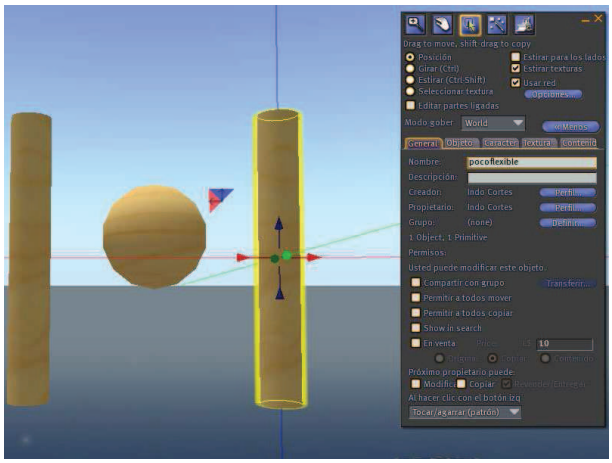


Imagen 6.7.cp.3.4: El objeto “poco_flexible”

3. Establecer la flexibilidad de los objetos componentes.

La esfera queremos que sea un objeto rígido, por ello no hace falta que la modifiquemos, ya que en Second Life los objetos son rígidos por defecto.

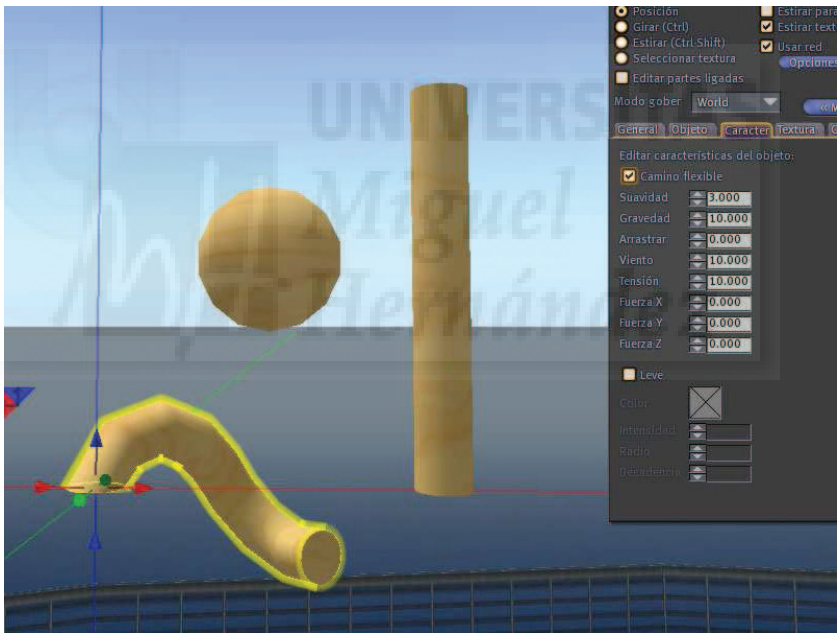


Imagen 6.7.cp.3.5: Parámetros de flexibilidad del objeto “muy_flexible”

3.1. Seleccionar el objeto llamado “muy_flexible”.

3.2. Acceder a la pestaña “Carácter” (Features) y pulsar sobre “camino flexible”. Esto hace que los parámetros de flexibilidad estén disponibles.

3.3. Establecer los siguientes valores:

Suavidad:	3.0
Gravedad:	10.0
Arrastrar:	0.0
Viento:	10.0
Tensión:	10.0

3.4. Repetir la misma operación con el objeto “poco_flexible” y establecer los valores:

Suavidad:	1.0
Gravedad:	1.0
Arrastrar:	10.0
Viento:	0.0
Tensión:	0.0

4. Unir los objetos componentes en el objeto final.

4.1. Seleccionar los dos objetos flexibles y moverlos de tal forma que se incrusten ligeramente en el objeto “cabeza”.

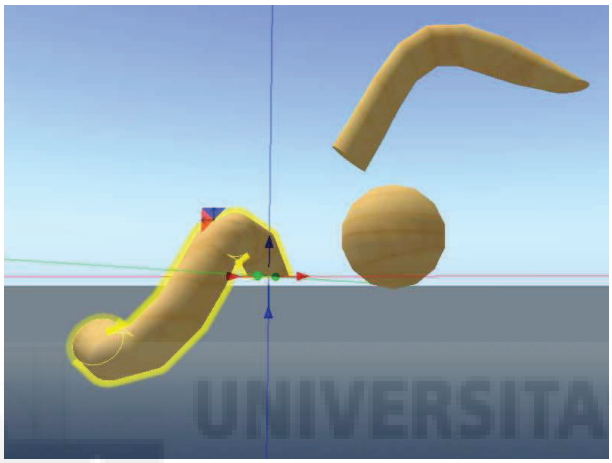


Imagen 6.7.cp.3.6: Posicionar los tres objetos para que se toquen

Comprobaremos que el objeto “muy_flexible” se mueve tanto (debido al propio movimiento de posicionarlo y/o al viento, etc.) que es muy difícil maniobrar con él.

Nota: hemos quitado la textura y al mismo tiempo coloreado las 2 patas para que se aprecien mejor sus diferencias, de tal forma que la pata roja es “poco_flexible” y la blanca es “muy_flexible”. Aparecen en posiciones distintas a como se crearon debido a que ahora el viento las mueve y para posicionarlas con respecto a “cabeza” las hemos movido.

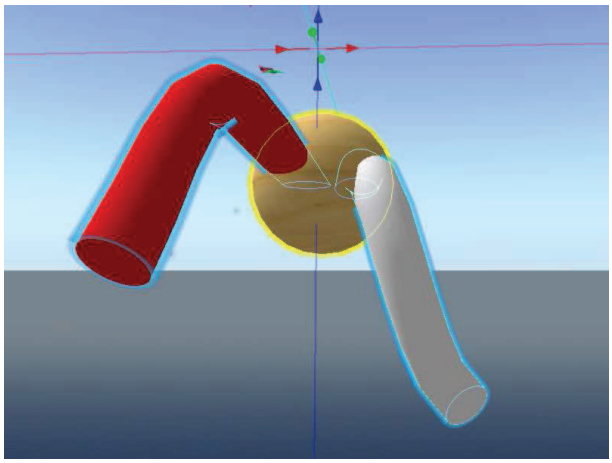


Imagen 6.7.cp.3.7: Unión de los tres objetos con la “cabeza” como objeto raíz

4.2. Seleccionar el objeto “muy_flexible” y editarlo pulsando botón derecho > editar.

4.3. Mantenido la tecla Shift pulsamos hacer clic sobre el objeto “poco_flexible” y por último sobre el objeto “cabeza”.

4.4. Ir al menú Herramientas y ejecutar el comando “Unir”.

4.5. Editar el objeto unido y moverlo para comprobar como las patas siguen en su movimiento a la cabeza pero cada una de ellas con sus propias características de flexibilidad.

Conclusiones:

Cuando necesitemos objetos flexibles, como pelo, banderas y en general, cualquier cosa “blanda”, podemos utilizar estas características que permiten generar elementos dinámicos e interactivos tanto con el entorno como con los avatares. La mejor forma de entender todas las variables que definen la flexibilidad de un objeto es probando sus distintos valores, ya que unos parámetros interaccionan con la función de otros.

Unidad 8: Programación en Second Life.

Introducción teórica:

1. Introducción a la programación en Second Life.
2. Dónde crear Scripts: en el inventario y en los objetos.
3. El primer programa en LSL.
4. Utilizar variables.
5. Estados.
6. Eventos.
7. Control de flujo.
8. Funciones de usuario.
9. Funciones de biblioteca.

1. Introducción a la programación en Second Life.

Second Life contiene un compilador embebido en su código. Esto es, un software que permite escribir código a los usuarios y asignarlo a distintas entidades como pueden ser avatares y objetos. Que este metaverso pueda ser programado nos permite interactuar y cambiar el mundo virtual de Second Life. El código se escribir en un lenguaje llamado Linden Scripting Language (LSL). Para ver una documentación completa del lenguaje podemos usar el menú Ayuda de Second Life o usar la ayuda que se encuentra en el directorio donde tenemos instalado Second Life, por ejemplo en C:\archivos de programa\second life\lsl_guide. LSL es un lenguaje moderno, del tipo de java o C++. Estos lenguajes son orientados a objetos y a eventos.

La programación orientada a objetos (POO) permite utilizar objetos software ya creados o crear nosotros objetos nuevos. Por ejemplo los botones, ventanas o avatares son objetos. La programación orientada a eventos supone que el programador utiliza un objeto y escribe a qué evento le hará caso, y si tiene lugar (es decir, si sucede), con qué acciones responderá. Por ejemplo, un botón se puede programar para que cuando suceda el evento que el usuario haga clic sobre él, este, genere la emisión de un sonido y cambie de aspecto. También tenemos el concepto de manejadores o handlers que son los elementos que detectan un evento y ejecutan el código correspondiente.

El código que escribimos lo hacemos en forma de scripts. Un script es un pequeño trozo de programación que realiza una tarea específica y que por sí solo no tiene sentido. Se debe ejecutar dentro de Second Life y trabajando en unión con el resto de códigos que crean el mundo virtual.

LSL tiene una característica muy destacable llamada persistencia. La persistencia hace que cuando un script es asignado a un objeto, este recuerda los valores de la última vez que se ejecutó ese script. Por ejemplo, si un script hace que una puerta se abra o cierre y en un momento dado salimos de Second Life mientras la puerta se encuentra abierta, la próxima vez que entremos la encontraremos abierta. Es como si en el tiempo donde no hemos estado en el metaverso, la puerta ha permanecido abierta. Esta característica es de suma importancia para dotar de “realismo” a este mundo.

2. Dónde crear Scripts: en el inventario y en los objetos.

El proceso para crear scripts tanto en el inventario como en los objetos es muy parecido por lo que detallaremos las diferencias. Lo más importante es que mientras que el script creado en un objeto pertenece solo a ese objeto, si lo creamos en el inventario, lo podremos tratar como general, es decir, que un script en el inventario no pertenece a un objeto en especial y por tanto, lo podemos utilizar en cualquier objeto. Pongamos un ejemplo. Supongamos que tenemos una sala de esculturas digitales y queremos que algunas giren sobre su eje vertical para que un visitante las pueda observar. Si escribimos un script en el inventario que se llame girar, el mismo script lo utilizaremos para varias esculturas. En el fondo es lo mismo, ya que si escribimos el mismo script para una escultura lo podemos exportar a otras con copiar y pegar, ya que solo se trata de texto. Pero si queremos modificar el script y está en el Inventario, solo tendremos que cambiarlo una vez siendo por tanto su mantenimiento más sencillo. Es una decisión más de organización que de programación.

En el inventario, creamos el script siguiendo el proceso:

a) Elegimos una carpeta dónde ubicarlo (generalmente la carpeta Scripts).
b) Botón derecho sobre la carpeta y elegimos “New Script”.
c) Hacemos doble clic sobre el nuevo script.

En el objeto, creamos el script realizando los siguientes pasos:

a) Seleccionamos el objeto y hacemos botón derecho y “Edit”.
b) Seleccionamos la pestaña “Contents”.

c) Pulsamos sobre el botón “New Script”.
d) Hacemos doble clic sobre el nuevo script.

El interfaz que presenta es el que se muestra en la imagen 6.8.1.

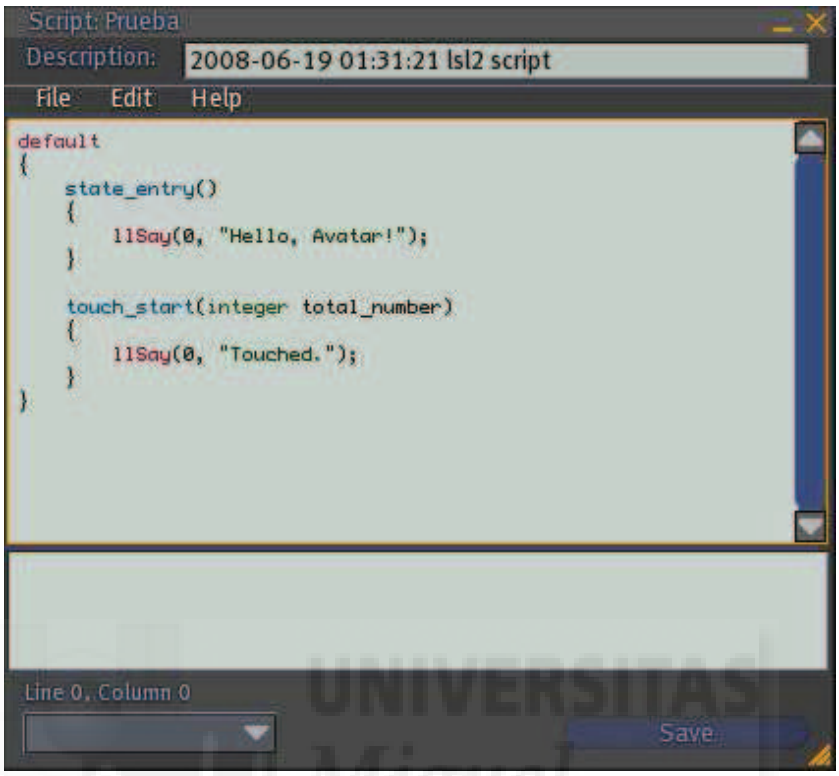


Imagen 6.8.1: Ventana para editar Script

En ambos casos cuando acabemos de escribir el script pulsaremos al botón “Save”. Si el código es correcto sintácticamente, se guardará, si no lo es, el cursor se posicionará en la primera palabra del código donde se supone que existen un problema.

3. El primer programa en LSL.

LSL es un lenguaje en el que todos los programas tienen la misma estructura. Cuando queremos programar, y requerimos hacer un nuevo script, accedemos a la ventana correspondiente y nos encontramos con un código ya escrito que es el siguiente:

```
default                    //definimos un estado, siempre default es el primero
{
        state_entry()     // cuando empieze este estado, es decir, al comenzar, saludamos
        {
                llSay(0, "Hello, Avatar!");
                // dice hello avatar en el canal 0, el único que un avatar puede usar
         }

        touch_start(integer total_number)        // cuando nos tocan, constestamos “tocado”
        {
                llSay(0, "Touched.");
        }
}
```

Lo primero que debemos hacer notar es que usamos "//" para poner comentarios que aclaren lo que escribimos en el script. Por ejemplo, se supone que si leemos los comentarios dentro del script anterior podremos entenderlo mejor. Esto lo hacen los programadores para documentar sus scripts para ellos mismos o para otros desarrolladores. Esto es más útil cuanto más grande y complejo sea el script. Por tanto todo lo que escribimos detrás de "//" no es programación, y el compilador lo ignora, es decir, en ningún momento pasa a ser manipulado, es solo texto.

Los programas en LSL están organizados en estados. Un estado es una condición particular en que se encuentra un objeto, por ejemplo un motor puede estar apagado, encendido, a x revoluciones, etc. A estos estados podemos llegar por eventos, que como hemos apuntado antes, son sucesos que se producen. Por ejemplo, un botón que es pulsado y que queremos que signifique llegar al estado de encendido de un motor.

default es la primera palabra que escribimos en todos los scripts. Este es el estado por defecto, es lo primero que ejecuta el script y además es obligatorio en todos los scripts. Es como decir "el script comienza aquí".

Observamos que después de default sigue un "{". Este símbolo significa que comienza un bloque que terminará con un "}". Un bloque puede contener otros subbloques.

state_entry() no es un estado, sino una función. Esta función es de tipo evento. Esto quiere decir que ejecuta su código cuando se produzca un suceso que lo active. En este caso el suceso es que comienza el programa. Por lo tanto el código escrito arriba significa que cuando comience el script va a saludar.

llSay() es también una función, pero en este caso es de tipo interactiva. Lo que hacen este tipo de funciones es modificar o interferir de alguna manera en el mundo de Second Life. En este caso, muestra un mensaje por pantalla.

Observamos cómo al terminar de escribir la función llSay sigue un ";". Este símbolo sirve para separar unas órdenes de otras, por lo tanto siempre que escribamos una, la finalizaremos con un punto y coma.

touch_start() también es una función de tipo evento. Esta función ejecutará su código cuando el objeto sea "tocado", es decir, cuando alguien haga clic con el ratón sobre él.

Por lo tanto, lo que hace este script es por una parte saludar y además cuando es tocado decir "touched".

Para modificar este script podríamos empezar por algo sencillo como sería traducirlo al español. No podemos escribir el código en el idioma que queramos, tenemos que utilizar LSL, pero las respuesta que proporciona el objeto mediante el script si que las podemos modificar, ya que al fin y al cabo es tan solo texto, también conocidos como "literales".

Modificar "Hello, Avatar!" por la frase "Hola, ¿qué tal?". Luego modificar la palabra "Touched" por "Me has tocado". Lo que hemos modificado no cambia la estructura del script, solo su "salida". Seguidamente pulsaremos el botón "Save" y probaremos el script haciendo clic sobre el objeto.

Es evidente que para programar scripts interesantes necesitamos conocer muchas más funciones de eventos y funciones de interacción. También necesitamos utilizar variables y constantes. Es decir, recursos que nos permitan crear y utilizar datos para manipularlos en el script.

4. Utilizar variables.

Una variable es un elemento de programación que sirve para que el desarrollador guarde valores. Por ejemplo, si queremos girar un objeto una cantidad determinada de grados y esta cantidad queremos que la pueda elegir un avatar, necesitamos utilizar una variable que sirva

para tomar el dato del avatar y luego para pasar su valor a una función que girará el objeto. En este sentido, una variable debemos verla como una zona de memoria donde se almacenan valores y que podemos identificar por una referencia, o sea, un nombre.

Antes de utilizar una variable debemos declararla. Cuando la declaramos, el compilador (el traductor del texto de programación a acciones de Second Life. Es el software que convierte el código que escribe un ser humano en instrucciones ejecutables por un ordenador) reserva una cantidad de memoria a nombre de la variable. Esta cantidad de memoria depende del tipo de variable. No es lo mismo guardar un nombre que un número, ya que gastan distinta cantidad de memoria. Para declarar una variable pondremos primero el tipo, seguido del nombre con que queramos referirnos a la variable. Por ejemplo:

integer gradosgiro; // la variable se llama gradosgiro y permite guardar un nº entero.

string saludo; // se llama saludo y permite guardar un grupo de caracteres.

Normalmente se aprovecha la declaración de la variable para inicializarla, o sea, darle un valor inicial antes de ser utilizada. Por ejemplo las variables anteriores se pueden declarar e inicializar en una sola sentencia:

integer gradosgiro = 0; // gradosgiro comienza teniendo el valor 0.

string saludo = "hola"; // saludo comienza teniendo el valor hola.

Si queremos utilizar una variable booleana, es decir, que solo admita dos valores: verdadero y falso, la debemos declarar como "integer" y luego utilizar las constantes TRUE Y FALSE para sus valores. Por ejemplo:

integer tocado = FALSE; // es un entero, FALSE es un entero que significa falso.

Pasemos ahora a utilizar las variables. La sentencia más básica es la asignación de un valor a una variable, que como hemos visto se realiza por medio del "=".

También utilizaremos operadores y funciones que permitan cambiar su valor o leerlo. Los operadores que se pueden aplicar a las variables dependen del tipo de estas. Por ejemplo el operador "+" aplicado a un string produce una concatenación y aplicado a un integer produce una suma. Por ejemplo

gradosgiro = gradosgiro + 2; // esto hace que gradosgiro aumente en 2 su valor.
// por ejemplo, si valía 0, ahora vale 2.
saludo = saludo + " mundo" // si saludo valía "hola", ahora vale "hola mundo".

También podemos usar las siguientes notaciones:

gradosgiro += 2; // esto hace que gradosgiro aumente en 2 su valor.

gradosgiro -= 2; // esto hace que gradosgiro disminuya en 2 su valor.

En el siguiente ejemplo se puede observar como cambiamos el valor de una variable al valor contrario con el operador !:

if (text == "on") color_on = !color_on;

Las variables pueden ser de dos tipos en cuanto a dónde utilizarlas: globales y locales. Una variable global se declara y utiliza como una local, pero la diferencia está en el lugar del script donde se puede nombrar, es decir, dónde se le puede hacer referencia. Las variables globales pueden ser accesibles y están activas en cualquier parte del script. Para declarar una función global basta con declararla antes del default. Por tanto, una variable no declarada global estará declarada dentro de un bloque (definido por { }) y esto quiere decir que solo estará activa y será accesible dentro de ese determinado bloque.

Los tipos de variables de los que disponemos en LSL son:

- integer: para contener valores numéricos sin decimales.
- float: para contener valores numéricos con decimales.
- string: para contener una cadena de caracteres.
- key: identificador único para avatares y objetos en SL.
- vector: son tres floats: x, y, z que se pueden usar para velocidades, colores, etc.
- rotation: son como vectores pero con un 4º componente: la velocidad de rotación.
- list: lista heterogénea compuesta por los otros tipos de datos.

Si en nuestros scripts necesitamos una constante, es decir, hacer referencia a un valor que no varía nunca, por ejemplo, PI, debemos usar las que define el propio lenguaje. No podemos crearlas nosotros. Una característica que las distingue es que siempre las escribiremos en mayúsculas.

Un literal es un texto que no varía. Se utiliza entre comillas y no hace falta declararlo. Lo podemos usar como valor para una variable tipo string.

También debemos comentar que aunque tengamos una variable de un tipo se puede "convertir" a otro tipo. Esto se llama "casting". Por ejemplo si tenemos una variable de tipo integer y queremos ver su valor por la pantalla, lo debemos convertir en string para usarlo dentro de la función llSay que es la que hace esta función y que solo admite strings. En el siguiente código se puede ver cómo se hace un casting:

```
altura = 180;
llSay (0, "mido "+ (string)altura + " centímetros");
```

5. Estados.

La idea de lo que es un estado en LSL, es un tanto indefinida y esto hace que sea un poco incomprensible al usuario no experimentado. Un estado de un objeto es como en la vida real, por ejemplo, un jarrón puede estar cayendo, girando, quieto,... En Second Life podemos escribir scripts para definir estos estados y qué hará el objeto cuando un suceso produzca que entremos dentro de uno de ellos.

Todos los scripts tienen un estado obligatorio que es default y es el primero que se ejecuta en un script, aunque se hayan escrito líneas de código antes, el programa empieza por la palabra default.

Para definir un estado nuevo utilizamos la palabra de LSL state. Por ejemplo:

```
default {
        state_entry(){
                state aparece;
        }
}

aparece {
        state_entry(){
                llSay(0, "Estoy en el mundo")
        }
        touch_start (integer total_number) {
                state responde;
```

```
        }
}

responde{
        state_entry(){
                llSay(0, "¿Por qué me tocas?")
        }       // de aquí no sale
}
```

6. Eventos.

Un evento es algo que sucede. Podemos atrapar los eventos con unas funciones que detectan un determinado suceso asociado a un objeto y se llaman "manejadores de eventos". Sirven para definir como interactuamos con los objetos.

Por ejemplo, un jarrón puede detectar que ha sido tocado, que aparece en el mundo, que recibe un mensaje, que un avatar se acerca, etc.

En los scripts anteriores ya hemos utilizado eventos y sus manejadores. En la siguiente tabla aparecen los manejadores de eventos de Second Life:

at_rot_target	Cuando el objetivo de tipo rotacional es alcanzado tras la ejecución de la función llRotTarget.
at_target	Cuando el target set es alcanzado tras la ejecución de llTarget.
attach	Cuando un objeto es attached or dettached a un agente o avatar.
changed	Cuando cambian estos valores de un objeto: inventory, color, shape, scale, texture, link, ownership.
collision	Durante la colisión o contacto de un prim (objeto… primitiva…).
collision_end	Cuando dos tareas (o prims) dejan de colisionar/contactar entre si.
collision_start	Cuando dos tareas (o prims comienzan a colisionar/contactar entre si.
control	Cuando uno de los controles tomados por llTakeControls es presionado, sostenido o liberado.
dataserver	Cuando un script recibe datos asíncronos.
email	Cuando un prim recibe un email.
http_response	Cuando un script recibe respuesta a la petición hecha con llHTTPRequest.
land_collision	Mientras una tarea (objeto, prim con script) colisiona con tierra.
land_collision_end	Cuando una tarea /prim deja de colisionar con tierra.
land_collision_start	Cuando una tarea/prim con script comienza a colisionar con tierra.
link_message	Cuando una tarea recibe un mensaje enviado mediante llMessageLinked.
listen	Cuando un script esta a la escucha en un canal indicado en llListen.
money	Cuando una tarea recibe dinero detecta un pago al prim donde está el script.
moving_end	Cuando una tarea deja de moverse.
moving_start	Cuando una tarea comienza a moverse.
no_sensor	Cuando la llamada a las funciones llSensor / llSensorRepeat da nulo.
not_at_rot_target	Cuando un objetivo de tipo rotacional en la ejecución llRotTarget no ha sido alcanzado todavía.

not_at_target	Cuando un destino en la ejecución de la funcion llTarget no ha sido alcanzado todavía.
object_rez	Cuando una tarea crea otra tarea mediante llRezObject.
on_rez	Cuando una tarea es creada, rezzed, desde el inventario o desde otra tarea.
remote_data	Cualquier comunicacion del tipo XML-RPC.
run_time_permissions	Cuando un agente concede permisos run-time a una tarea que fue requerida mediante la funcion llRequestPermissions.
sensor	Resultado de las funciones llSensor o llSensorRepeat.
state_entry	En cualquier transición en el estado y en el arranque (startup).
state_exit	En cualquier transición de salida del estado.
timer	En los intervalos determinados por la función llSetTimerEvent.
touch	Cuando el agente hace click en la tarea.
touch_start	Cuando un agente comienza a hacer click en una tarea.
touch_end	Cuando un agente deja de hacer click en la tarea.

Tabla 13: Manejadores de eventos de Second Life

7. Control de flujo

Como en todo lenguaje de programación, el programador necesita controlar el orden en que se ejecutan los elementos que conforman el programa. Para ello tendremos una serie de sentencias de control del flujo que son las siguientes:

if (condición) código_si_cierto

Sintaxis:

condición: si la condición es TRUE (verdadera) se ejecuta el código_si_cierto.
código_si_cierto: una sentencia simple, un bloque de código o una sentencia nula.

Ejemplo:

if (llDetectedKey(0) == llGetOwner()) state abrir_puerta;

if (condición) código_si_cierto else código_si_falso

Sintaxis:

condición: si la condición es verdadera se ejecuta el código_si_cierto.
código_si_cierto: una sentencia simple, un bloque de código o una sentencia nula.
código_si_falso: una sentencia simple, un bloque de código o una sentencia nula.

Ejemplo:

if (llDetectedKey(0) == llGetOwner()) state abrir_puerta else state no_abrir_puerta;

for(inicializador ; condición; incremento) código

Sintaxis:

inicializador: ejecutado una vez antes de comprobar condición.
condicion: si la condición es verdadera se ejecuta el código.
incremento: ejecutado después del código, y luego la condición se evalúa de nuevo.
codigo: puede ser una sentencia simple, un bloque de código o una sentencia nula.

Ejemplo:

```
For (i = 0; i <= pasos; ++i) {
        llSleep(0.1);
}
```

while(condición) código

Sintaxis:

condición: si la condición es verdadera se ejecuta el código.
código: puede ser una sentencia simple, un bloque de código o una sentencia nula.

Ejemplo:

```
integer i=3;
while (i--) {
        llSay(0, “Hola”);
}
```

do código while (condición);

Sintaxis:

código: ejecutado una vez, a continuación se evalúa la condición.
condición: si la condición es verdadera, vuelve atrás y se ejecuta el código otra vez.

Ejemplo:

```
integer i=0;
do {
        llSay(0, “Hola”);
                i++;
} while (i<3
```

jump etiqueta

Sintaxis:

Etiqueta: nombre de una etiqueta dentro del ámbito. Define un sitio para un posible salto.

Ejemplo:

```
jump salto;
```

@etiqueta

Sintaxis:

Etiqueta: nombre de etiqueta para un salto en el mismo ámbito o en un ámbito hijo.

Ejemplo:

```
@salto;
```

return valor

Sintaxis:

valor: valor o variable que devuelve la función, el tipo debe ser correcto.

Ejemplo:

return calculo;

return

Sintaxis:

Se utiliza para volver de forma inmediata o prematura al ámbito superior antes de llegar al final de la funcion.

Ejemplo:

return;

8. Funciones de usuario.

Estas funciones se llaman de usuario porque son creadas y utilizadas por los propios programadores en los scripts que escriben.

Las funciones se utilizan en todos los lenguajes de programación. Sirven para estructurar un programa en módulos. Es decir, dividir un problema complejo en partes más sencillas de entender y resolver.

Una función es un bloque independiente de código que tiene un nombre para poder referenciarlo y así ejecutarlo desde otra parte del programa. Tiene la propiedad de poder devolver un resultado o no y también de recibir o no una serie de datos de entrada, a los que se denomina “parámetros de entrada”.

Por ejemplo una función puede calcular la media de tres números que por supuesto deben ser introducidos en la función. En este caso funcionaria como una función matemática.

También puede ser que ejecute líneas de código sin devolver ningún resultado al finalizar y que por tanto sirva solamente como modulo independiente del resto. Este tipo de funciones se utiliza mucho cuando se ejecuta una tarea muy repetitivamente en el tiempo.

Una función puede ser llamada, es decir ejecutada desde cualquier punto del programa si es definida antes de “default”, si no es así, solo se podrá llamar dentro del bloque donde es definida. Normalmente esta opción no es muy efectiva. Sin embargo, desde dentro de una función se puede llamar a cualquiera otra función.

Por ejemplo, supongamos que queremos que nuestro avatar o un objeto nos hable en pesetas en vez de en euros. Escribiremos una función básica para pasar de euros a pesetas que seria algo así:

```
integer enpesetas (integer eneuros)
{
        integer resultado = a*166;
        return resultado;
}
```

Para utilizarla, escribiremos un script donde llamaremos a la función como sigue:

```
default
{
        state_entry()
        {
                llSay(0, "Hello, Avatar!");
```

```
        }
        touch_start(integer total_number)
        {
                llSay(0, "3 euros son "+ (string)enpesetas(3) + " pesetas");        // traduce a
        pesetas
                // (string) es para convertir un entero a cadena de caracteres.
        }
}
```

9. Funciones de biblioteca.

Estas funciones se llaman de biblioteca porque son creadas por Linden Labs., especialmente para ser utilizadas en Second Life y las ponen a disposición de los programadores de este metaverso, por lo tanto, la diferencia con las funciones de usuario es su procedencia.

En LSL existen unas 200 funciones de biblioteca. Todas comienzan con un "ll" y son seguidas de una mayúscula. A modo de ejemplo describimos algunas de las más utilizadas:

llGetObjectName();

Devuelve el nombre del objeto (un string) al que está unido el script. Por ejemplo:

```
llSetText (llGetObjectName(), <0.0, 1.0,0.0>, 1.0);
// obtiene el nombre  y lo muestra en verde mediante la función llSetText
```

llGetOwner();

Devuelve un código de tipo key que es el del propietario del script.

llGetPos ();

Devuelve un vector con la posición del objeto que ejecuta la función. Por ejemplo:

```
default {
        touch_start(integer total_number) {
                llSay ( 0, "mi posicion es " + (string) llGetPos());
        }
}
```

llGetScale ();

Devuelve un vector con la escala del objeto que ejecuta la función. Por ejemplo:

```
default {
        touch_start(integer total_number) {
                llSay ( 0, "mis dimensiones son " + (string) llGetPos());
        }
}
```

llGetTime();

Devuelve un número en formato float que son los segundos desde la última inicialización del simulador. Para que sea útil y contemos un periodo de tiempo, se utiliza en combinación con llResetTime() que reinicia el conteo de segundos cuando la ejecutamos.

```
default {
        state_entry() {
                llResetTime();
        }
```

```
        touch_start(integer num_touch) {
                float tiempo = llGetTime();
                llResetTime();
                llSay(0,(string)tiempo + " segundos han pasado desde la última
vez que me tocaste." );
        }
}
```

llListen (integer canal, string nombre, key identificador, string mensaje);

Escuchará un determinado mensaje en un determinado canal. Podemos hacer un filtro para oír solo a un nombre o con un determinado código (key). Devuelve un entero que se puede utilizar para borrar o desactivar la escucha. En el siguiente ejemplo nos escuchamos a nosotros mismos diciendo "abre la puerta".

```
llListen (0, "", llGetOwner(), "abre la puerta");
```

llLoadURL(key avatar, string mensaje, string url);

Muestra una caja de diálogo con el mensaje al avatar. Tendremos dos botones, si el avatar pulsa "Aceptar", se cargará la web con la URL.

```
default {
        touch_start(integer total_number) {
                key idavatar llDetectedKey(0);
                llLoadURL( idavatar, "ver mi página web ", "www.godofredo.com);
        }
}
```

llPassTouches (integer detector);

Permite la detección de cuando se toca a un avatar u objeto. El valor por defecto es "false" si no hay eventos en el script que lo detecten. Si es "trae" el permiso pasa también a los objetos hijos si los hubiera.

llResetTime();

Inicializa el contador de tiempo del script a 0. Ver el ejemplo de la función llGetTime().

llSetColor(vector color, integer cara);

Coloca el color en la cara del prim (avatar u objeto). La cara viene identificada por un entero. Si ponemos el valor "ALL_SIDES" a este parámetro, se aplicará a todas las caras del prim. En el siguiente ejemplo, pintamos totalmente de azul el prim que ejecute el script que contenga la siguiente línea:

```
llSetColor(<0.0, 0.0,1.0>, ALL_SIDES);
```

llSetPos(vector posicion);

Mueve el objeto a la posición referenciada sin física. En el siguiente ejemplo, el objeto que ejecute el script, se moverá 1 metro hacia arriba cada vez que sea tocado.

```
default {

        touch_start(integer total_number) {
                llSetPos( llGetPos()+<0,0,1>);
        }
}
```

llSetScale(vector escala);

Escala un prim sin física. En este ejemplo hacemos el objeto un 20% más pequeño:

llSetScale(<0.1,0.1,0.1>);

llSetTexture (key textura, integer cara);

Coloca una textura en una cara determinada del prim.

llSetTexture(llGetInventoryName(INVENTORY_TEXTURE, i),cara);

llSetText(string texto, vector color, flota alpha);

Muestra un texto sobre un prim en un color y transparencia determinado. Ejemplo:

llSetText ("Pepe", <1.0, 0.0,0.0>, 1.0); // mostrará el texto Pepe en rojo y opaco.

llSetTimerEvent(float segundos);

Hace que el evento Timer() sea llamado cada x segundos. Por ejemplo si ponemos llSetTimerEvent (3) hará que se llame al evento Timer cada 3 segundos. Lo que queremos que se haga cada 3 segundos tendremos que escribirlo dentro del Timer(); En el ejemplo que sigue, hacemos que se diga el número de segundos que han pasado del script cada 5 segundos.

```
float contador;
default {
        state_entry(){
                llSetTimerEvent(5.0);
        }
        timer() {
                contador = contador + 5.0;
                llSay( (0, (string)contador+" segundos han pasado");
        }
}
```

llSleep (float segundos);

Detiene el script x segundos en los que no efectuará ninguna acción.

```
default {
        state_entry(){
                llSay( (0,"me voy a dormir 5 segundos");
                llSleep(5.0);
                llSay( (0,"que buena es una siestecita de vez en cuando");
        }
}
```

llTriggerSound (string sonido, float volumen);

Emite un sonido del inventario del propietario del script a un volumen. El volumen con valor 0.0 será en silencio, 1.0 será a su máximo volumen. Por ejemplo la siguiente línea emite un sonido llamado hola.

llTriggerSound("hola", 1.0);

llTargetOmega(vector ejes, float velocidadgiro,float fuerza);

Esta función gira sobre los ejes definidos como un vector a una determinada velocidad de giro. El parámetro fuerza sirve para crear efectos de inercia y vale normalmente 1. La velocidad de

giro se mide en radianes por segundo. En el siguiente ejemplo rotamos el objeto 2 veces por segundo en el eje X, 1 vez por segundo en el eje Y y media vuelta por segundo en el eje Z.

llTargetOmega(<2,1,0.5>,TWO_PI,1);

Unidad 9: Animación de objetos.

Introducción teórica:

1. La animación y el movimiento en Second Life.
2. Interacción de los avatares con los objetos animados.
3. Las funciones más usuales en animación.
4. Ejemplos de animación en Second Life.

Casos prácticos:

Caso práctico 9.1. Hacer que un objeto rote por programación.

Caso práctico 9.2: Control del movimiento de los objetos por programación.

1. La animación y el movimiento en Second Life.

La animación y Second Life van indisolublemente unidos. Casi todo en Second Life se mueve. Solo hace falta dar un pequeño paseo por cualquier sitio para darnos cuenta que todo a nuestro alrededor está animado. Como en la imagen de abajo donde vemos a nuestro avatar ante un tiovivo donde lo caballitos se mueve arriba y abajo mientras dan vueltas. Los molinos de viento del fondo, las nubes, los árboles mecidos por el viento, todo se mueve.

Imagen 6.9.1: En Second Life casi todo se mueve

Como ya hemos visto, debemos distinguir entre los avatares y todos los demás objetos, ya que detrás de cada avatar existe un usuario y por lo tanto, las animaciones que se pueden realizar con ellos son directamente ejecutadas por órdenes de los usuarios. El resto de objetos que presenta Second Life pueden o no estar animados, pero si vemos una puerta rotar sobre sus bisagras, es porque alguien ha escrito un pequeño código de programación para que esto suceda.

Por lo tanto podemos ver Second Life como un mundo animado que dispone de varios métodos para producir estos movimientos. Además a todo esto debemos sumar la interacción constante que supone la existencia de una máquina física, un software que permite representar a los objetos como entidades con propiedades físicas como la masa y por tanto sentir fuerzas naturales como la gravedad. Esto quiere decir que algunos movimientos no son ejecutados por una programación directa de los usuarios sino por el simulador de realidad virtual que es Second Life. Por ejemplo, si empujamos a un objeto declarado como “físico” este puede colisionar con otro objeto físico y moverlo a su vez.

Podemos crear movimiento con muchos objetivos. Por ejemplo, podemos crear hormigas y dotarlas de movimiento para simular su comportamiento y su búsqueda de comida. También podemos crear utensilios y herramientas que simulen la vida real como el movimiento inherente a puertas y ventanas. Mención aparte merecen los automóviles, ya que son objetos con una programación muy complicada para que el realismo sea mayor.

Dicho esto, el movimiento que nos interesa controlar es el que proporciona a las obras artísticas la cualidad de ser móviles o de ser exhibidas de una manera más efectiva.

Por ejemplo, si queremos crear una escultura que se mueva o que tenga partes móviles, tenemos que proporcionarle este movimiento nosotros mismos y lo tenemos que hacer mediante la aplicación de programación en forma de scripts.

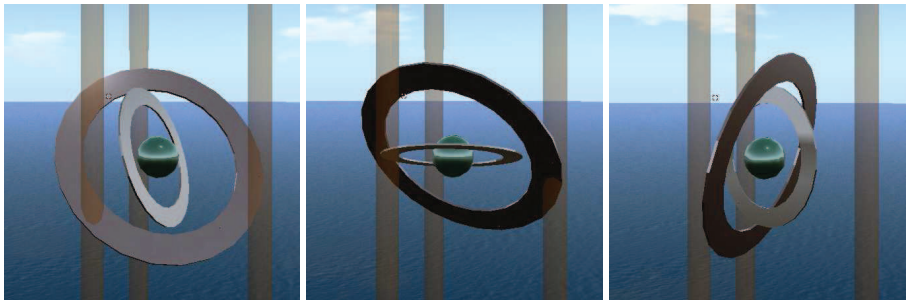


Imagen 6.9.2: Escultura móvil con movimientos rotatorios

También podemos hacer que la escultura, a requerimiento de un usuario o constantemente, gire lentamente para que pudiera verse desde todos los ángulos incluso por los avatares que permanecieran parados.

La mayoría de los movimientos que realizaremos serán por tanto de rotación y traslación, es decir, sin mucha complejidad técnica, y sin embargo, nos pueden proporcionar unas propiedades en nuestras creaciones que pueden ser muy útiles.

2. Interacción de los avatares con los objetos animados.

Ya hemos introducido este tema antes al dar a entender que muchas veces nos puede convenir que el movimiento que tenga nuestra obra artística no se produzca constantemente, sino en respuesta a una petición de un avatar.

Esta interacción puede hacer que el movimiento sea una aportación más útil, ya que solo se producirá cuando sea necesario, pero también puede introducir elementos de complejidad. Por ejemplo, supongamos que exponemos una escultura y que esta es capaz de rotar sobre su eje X a petición de los avatares que la observan. ¿Cómo gestionamos la respuesta si un avatar pide que rote y otro que se detenga prácticamente a la vez? Una posible solución es utilizar distintos canales de comunicación para distinguir entre usuarios. Estas cuestiones, por supuesto, están en manos del "exhibidor" pero no se tiene que pasar por alto para no convertir lo que a priori es una ventaja de nuestra obra, en un problema.

En el tema en que se trata la interacción en Second Life se profundiza en la materia que permite que el avatar se relacione con la obra desde distintos puntos de vista.

De todas formas, lo más habitual suelen ser que el avatar escriba en el chat público un comentario que esté dirigido a la obra, también puede ser que la toque o que toque a otro objeto que a su vez intervenga en la obra con la que se quiere "comunicar". Como ya hemos dicho, estas formas de interacción no se discutirán en este tema pero sí en el tema 10.

3. Las funciones más usuales en animación.

llGetPos(): Devuelve un vector que es el de la posición de región del objeto. La posición de región quiere decir que se toma como origen el punto 0,0,0 de la región donde se encuentra el objeto.

llMoveToTarget(vector target, float tau): Mueve inercialmente un objeto físico a una posición (target) en tau segundos.

llRotTarget(rotation rot, float error): Esta función se utiliza para saber cuando un objeto ha llegado a una rotación (rot) dentro de unos límites dados por "error". Devuelve el resultado desencadenando los triggers at_rot_target y not_at_rot_target continuamente hasta que la rotación objetivo sea eliminada por la función llRotTargetRemove()

llRotTargetRemove(): Borra la rotación objetivo para un movimiento de rotación asignada con la función llRotTarget().

llSetPos(vector pos): Mueve un objeto hasta la posición "pos" sin inercia. En este sentido se diferencia de la función llMoveToTarget, ya que esta hace moverse un objeto pero con efectos inerciales, lo que hace que esta última función sea más adecuada para movimientos realistas.

llSetTimerEvent(interval): Esta función hace que x veces por segundo se ejecute la función asociada timer() que se puede usar para muchas cosas, como por ejemplo, mover un objeto de posición a posición. Por tanto llSetTimerEvent() se puede usar para discretizar el movimiento en el tiempo. O sea, para ejecutar un código que mueva el objeto un determinado espacio cada X tiempo.

llSitTarget (vector offset, rotation rot): Establece una localización para que un avatar se pueda sentar en un objeto. Offset y rot son la posición y el giro con respecto a la posición y giro del prim que sirve de asiento.

llStopMoveToTarget(): Para automáticamente un movimiento inercial de un objeto físico. Se utiliza conjuntamente con llMoveToTarget().

llTarget (target, range): Esta función se utiliza para saber cuando un objeto ha llegado a una posición (target) dentro de unos límites dados por "range". Devuelve el resultado desencadenando los triggers at_target y not_at_target continuamente hasta que la posición objetivo usada por llTarget sea borrada por la función llTargetRemove()

llTargetOmega(eje,velocidad,inercia): Rota un objeto en un determinado eje, y a una velocidad dada. Inercia es un valor que sirve para crear efectos realistas al terminar la rotación, pero si le ponemos un valor de 0 no tiene lugar la rotación, la desactiva.

llTargetRemove(target): Borra la posición objetivo para un movimiento asignada con la función llTarget().

timer(): Este es un evento que se repite a intervalos que marca la función llSetTimerEvent() y por tanto se asocia a ella. De esta forma, la función marca la frecuencia y el evento timer lo que se debe repetir con esa frecuencia.

llVecDist (pos1, pos2): Devuelve la distancia en valor absoluto entre las posiciones "pos1" y "pos2".

```
Default {
        State_entry () {
                vector posicion_1= <1.0,2.0,3.0>;
                vector posicion_2= <3.0,2.0,1.0>;
                llOwnerSay ("La distancia entre "+(string) posicion_1 + " y " + (string)
posicion_2 + "es: " + (string) llVecDist (posicion_1,posicion_2));
        }
}
```

4. Ejemplos de animación en Second Life.

En este apartado queremos presentar algunos scripts que producen animación en los objetos donde están contenidos. Estos scripts o parecidos los explicaremos con más detalle en los casos prácticos.

- Script 1: para que un objeto rote continuamente escribimos el siguiente código:

```
default
{
```

```
    state_entry()
    {
      llTargetOmega(<0,0,1>,PI,1);
    }

}
```

◆ Script 2: para que un objeto se desplace 1 m. en distintos ejes e interactivamente:

```
default
{
state_entry() {
llListen(0,"", NULL_KEY, "");
}

listen(integer channel, string name, key id, string message) {
    if (message == "arriba") { llSetPos(llGetPos()+<0,0,1>); }
    else
    if (message == "abajo") { llSetPos(llGetPos()+<0,0,-1>); }
    else
    if (message == "izquierda"){ llSetPos(llGetPos()+<1,0,0>);}
    else
    if (message == "derecha") { llSetPos(llGetPos()+<-1,0,0>); }
    else
    if (message == "alejate") { llSetPos(llGetPos()+<0,1,0>); }
    else
    if (message == "acercate") { llSetPos(llGetPos()+<0,-1,0>); }
    else
    if (message == "diagonal") { llSetPos(llGetPos()+<1,1,1>); }
    else llSay(0,"No te entiendo");
}
}
```

◆ Script 3: para que un objeto siga una trayectoria recta en un determinado eje:

```
integer cont;

default
{

state_entry() {
llListen(0,"", NULL_KEY, "");
}

listen(integer channel, string name, key id, string message) {

    if (message == "ponte azul") { llSetColor(<0,0,1.0>,ALL_SIDES); }
    else
    if (message == "ponte rojo") { llSetColor(<1.0,0,0>,ALL_SIDES); }
    else
    if (message == "marriba") {

       for (cont=0; cont<= 5;++cont) {
          llSetPos(llGetPos()+<0,0,0.5>); }
       }
    else llSay(0,"No te entiendo");
}
}
```

Caso práctico 9.1. Hacer que un objeto rote por programación.

Objetivo: Escribir un script que haga que un objeto rote sobre si mismo en el eje vertical.

Tiempo de realización: 1/2 hora.

Pasos a realizar:

1. Crear el objeto.
2. Crear un script para rotar el objeto sobre el eje z.

1. Crear el objeto.

Crearemos un objeto diseñado expresamente para poder observar bien las rotaciones. Estará compuesto por una caja, un cilindro y un triángulo. En la siguiente imagen se puede apreciar el objeto terminado.

Imagen 6.9.cp.1.1: Objeto realizado para programarlo posteriormente

En primer lugar tendremos que estar en una parcela editable o un sandbox. Entonces realizamos lo siguientes pasos.

1.1. Hacer clic en el suelo con botón derecho y elegir la opción Create. Elegimos la opción “Caja” y hacemos clic en el suelo. Aparecerá un cubo que seguidamente vamos a modificar.

1.2. En la ventana del objeto, elegimos la ficha “Object” y en “Size” ponemos los tres valores a 0.5 metros.

1.3. Ahora vamos a realizar dos copias de la caja que luego convertiremos en el cilindro y el triángulo. Para ello, con la tecla “Shift” pulsada arrastrar la flecha azul hacia arriba y soltar. Con esta acción creamos una copia de la caja 1.

1.4. Repetir la operación para crear una tercera caja como se observa en la imagen 6.9.cp.1.2. Estas cajas tienen una característica y es que al ser creadas con este método nos aseguramos de que están alineadas verticalmente.

1.5. Seleccionamos el segundo objeto y en la ficha “Object”, en el control “Building Block Type” elegimos el valor “Cylinder”. Repetimos lo anterior con la tercera caja y la convertimos en un prisma.

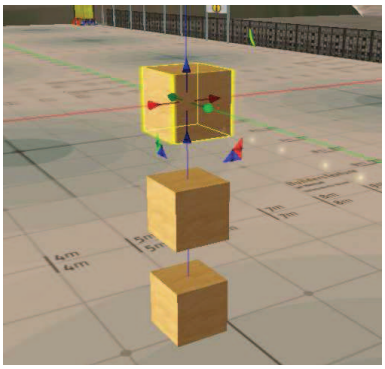


Imagen 6.9.cp.1.2: Copia de cajas para su posterior modificación

1.6. Modificamos el prisma en la ficha “Object” poniendo los valores Size: 0,5; 0,5; 0,1 y Rotation: 90; 0; 90;

1.7. Modificamos el cilindro en la ficha “Object” poniendo los valores Size: 0,1; 0,1; 1,0 y Rotation: 0; 0; 0;

El objeto debe ir tomando forma tal y como muestra la imagen que sigue.

Imagen 6.9.cp.1.3: Los tres componentes de nuestro objeto final

1.8. Ahora vamos a poner juntos los tres elementos de tal manera que se toquen. Para ello basta con seleccionarlos haciendo clic. Aparecerán unas flechas de colores para moverlos en los tres ejes. En este caso solo debemos arrastrar las flechas azules que moverán los objetos en el eje vertical.

Para esta tarea puede ser de gran ayuda el control de la cámara de visualización que se obtiene pulsando simultáneamente la teclas Control + Alt y arrastrar para acercar o alejar.

Nota: hay que tener en cuenta que los objetos se “tocan”, pero no están pegados, es decir, que aún se trasladan, rotan, etc. independientemente.

1.9. Ahora vamos a unir (linkar) los tres elementos en uno solo (se puede deslinkar) para moverlos o rotarlos todos a la vez como si de un solo objeto se tratara. Para ello, seleccionamos el prisma, luego el cilindro y por último (esto es importante) la caja. Accedemos al menú Tools y ejecutamos Link (o Control + L). Se coloreará el cilindro y el prisma de color azul y la caja en color amarillo.

1.10. Comprobar que están unidos moviendo y rotando el conjunto.

2. Crear un script para rotar el objeto sobre el eje z.

El script se puede crear en el Inventario o en el propio objeto. Si lo creamos en el Inventario, se puede reutilizar para más de un objeto y por ello será la opción elegida.

2.1. Abrimos el inventario y en la carpeta “Script” elegimos “New script”. Esto hace que se abra un editor de código, donde escribimos el siguiente texto:

```
default
{
   state_entry()
   {
      llTargetOmega(<0,0,1>,PI,1);
   }

}
```

2.2. Llamamos al script “Rotar en Z” y pulsamos “save”.

2.3. Arrastramos el script desde el inventario hasta la escultura para asignárselo.

Comprobamos que en cuanto le asignamos el script, este se ejecuta y se realiza la rotación. Como la caja es el origen (por el link), toma su centro como punto de pivote de toda la “escultura” para llevar a cabo la rotación.

Debemos tener en cuenta que la función que realiza la rotación es llTargetOmega(), por lo que debemos estudiarla a fondo.

Los valores utilizados debemos medirlos en radianes y no en grados.

<0, 0, 1> es el sentido de la rotación. Determina los ejes x, y, z en que rotará.

PI es la velocidad de rotación que se multiplica por el valor de los ejes. En este caso 0 PI por segundo en X y en Y, y 1 PI por segundo en Z.

El último parámetro, que en el ejemplo toma un valor de 1, es la fuerza, no debe valer 0 o no rotará, el valor máximo es 1 y esto provoca que la rotación sea sin física, es decir, sin inercia.

Por último cabe probar algunas modificaciones para experimentar cómo trabaja esta función, por ejemplo cambiar el valor de PI por el valor TWO_PI para comprobar que rota al doble de velocidad.

También podemos modificar el vector de rotación, con lo obtendremos resultados como los que muestra la Imagen 6.9.cp.1.4

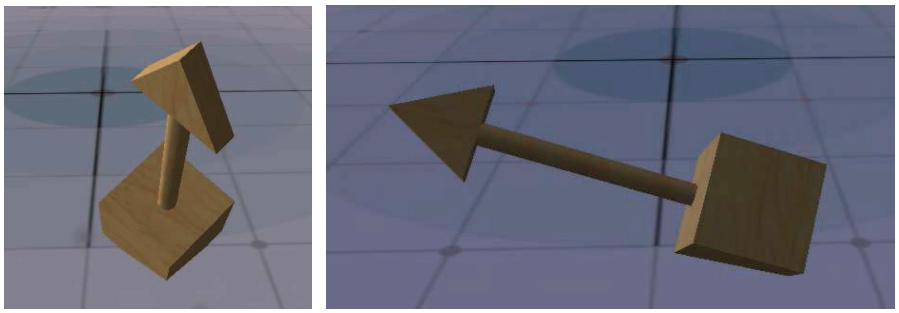


Imagen 6.9.cp.1.4: Rotaciones obtenidas al modificar los valores de llTargetOmega()

Conclusiones:

Hacer una animación mediante programación no tiene gran dificultad siempre que podemos utilizar una función que realice el trabajo, ya que en entonces nuestro esfuerzo se basa en estudiar y utilizar esta función y los parámetros con los que trabaja. Sin embargo, si no existe una función específica para el movimiento deseado y la tenemos que programar completamente nosotros, el trabajo requerirá amplios conocimientos de programación y física.

Caso práctico 9.2: Control del movimiento de los objetos por programación

Objetivo: Programar scripts para dotar de movimiento y animación propia a los objetos.

Tiempo de realización: 2 horas.

Contenidos:

1. Introducción al movimiento por programación.
2. Movimiento continúo.
3. Posicionamiento de un objeto a petición de un avatar.
4. Movimiento rectilíneo.

1. Introducción al movimiento por programación.

En la unidad teórica correspondiente ya vimos que el movimiento de los objetos lo tenemos que desarrollar mediante programación. Tenemos que tener en cuenta que estamos hablando de movimiento propio de los objetos y este es diferente al que se produce por el hecho de que un objeto sea físico. Esto quiere decir, que si un objeto cae de una altura por ser "físico" no se tiene que programar nada por parte del usuario, ya que de esto se encarga la máquina física. Por lo tanto, nos referimos a un movimiento "no natural", por ejemplo, si queremos que una escultura tenga un movimiento rotatorio sobre su eje vertical o algo así.

En la práctica que sigue se visualiza muy claramente que el movimiento es una combinación de dos magnitudes: el espacio y el tiempo. Por lo tanto debemos de conocer suficientemente el lenguaje LSL para tener distintas opciones que nos permitan trabajar con estos dos conceptos.

Sigue una tabla donde se resumen algunos de los elementos de LSL que utilizaremos en estos ejemplos de movimiento:

Magnitud espacio	**Uso**
llGetPos()	Nos da la posición de un objeto.
llSetPos()	Mueve un objeto hasta una posición sin inercia.
llTargetOmega()	Hace girar el objeto.

Magnitud tiempo	**Uso**
llSetTimerEvent()	Función para controlar la frecuencia de ejecución de timer().
timer()	Evento que ejecuta sentencias. Se utiliza junto a llSetTimerEvent().
for ()	Sentencia para repetir sentencias una cantidad de veces.

Tabla 14: Elementos para programar el movimiento

2. Movimiento continúo.

El movimiento continuo es aquel que vamos a proporcionarle a un elemento durante toda su "vida". Si por ejemplo es una escultura que tenemos en nuestra exposición, desde que lo creamos o sacamos del inventario, va a verse afectado por ese movimiento.

En el caso práctico precedente que realizamos para ilustrar cómo se lleva a cabo la animación por programación ya realizamos este ejercicio y se explica detalladamente.

El código que sigue hace que cualquier objeto gire con respecto a su eje Z a una velocidad de PI radianes por segundo y con una fuerza de 1. Esto significa que no se experimentará ningún efecto de fuerzas añadidas al movimiento.

```
default
{
   state_entry()
   {
     llTargetOmega(<0,0,1>,PI,1);
   }

}
```

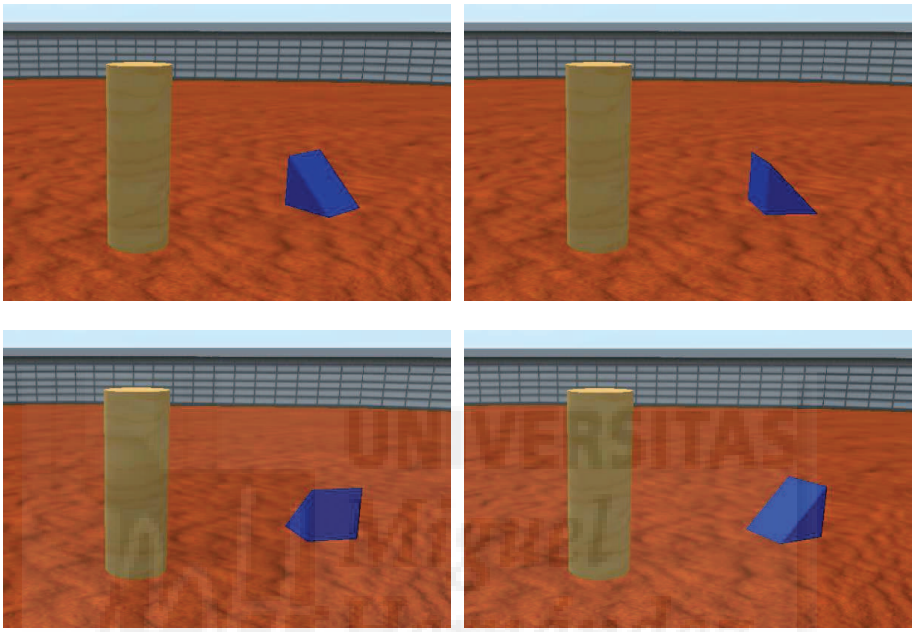


Imagen 6.9.cp.2.1: Movimiento de giro sobre el eje Z

Para crear el ejemplo que se puede ver en las siguientes imágenes seguir los pasos:

2.1. Crear en un sandbox un objeto de referencia. Para ello pulsar con el botón derecho sobre el suelo y en el menú contextual elegimos “Crear” y elegimos el cilindro. Luego lo modificamos para que sea un poco más alto que el cilindro por defecto.

2.2. Crear el objeto que rotará. Para ello pulsar con el botón derecho sobre el suelo y en el menú contextual elegimos “Crear” y elegimos el prisma. No lo modificamos en tamaño pero si que le cambiamos el color a azul y le quitamos la textura para que se visualice mejor.

2.3. Cuando estamos editando el objeto prisma, pulsamos sobre la pestaña “Contenidos” y luego sobre el botón “New Script” y en la ventana que se abre escribimos el código expuesto arriba.

En cuanto dejamos de editar el objeto observaremos como se pone a girar sobre sí mismo. Podemos experimentar modificando el vector que define el eje de giro y la velocidad para ver la manera en que afecta a su movimiento.

Con este caso, hemos visto cómo podemos crear un objeto y añadirle movimiento, en los casos que siguen modificaremos el script para tener otros resultados, pero basta con cambiar un script por otro. Por lo tanto, al utilizar los mismos dos objetos, el de referencia y el móvil, podremos ver más fácilmente las diferencias que se producen al modificar los scripts.

3. Posicionamiento de un objeto a petición de un avatar.

En este caso pretendemos desplazar el objeto 1 metro directamente desde la posición que ocupe en cualquiera de los ejes y en los dos posibles sentidos.

Para que este script se pueda realizar, es necesario que utilice algún modo de interacción con el objeto. En este caso hemos optado porque el avatar se comunique con el objeto por medio del chat. Para aclarar mejor la parte interactiva del script deberemos estudiar antes la unidad 10.

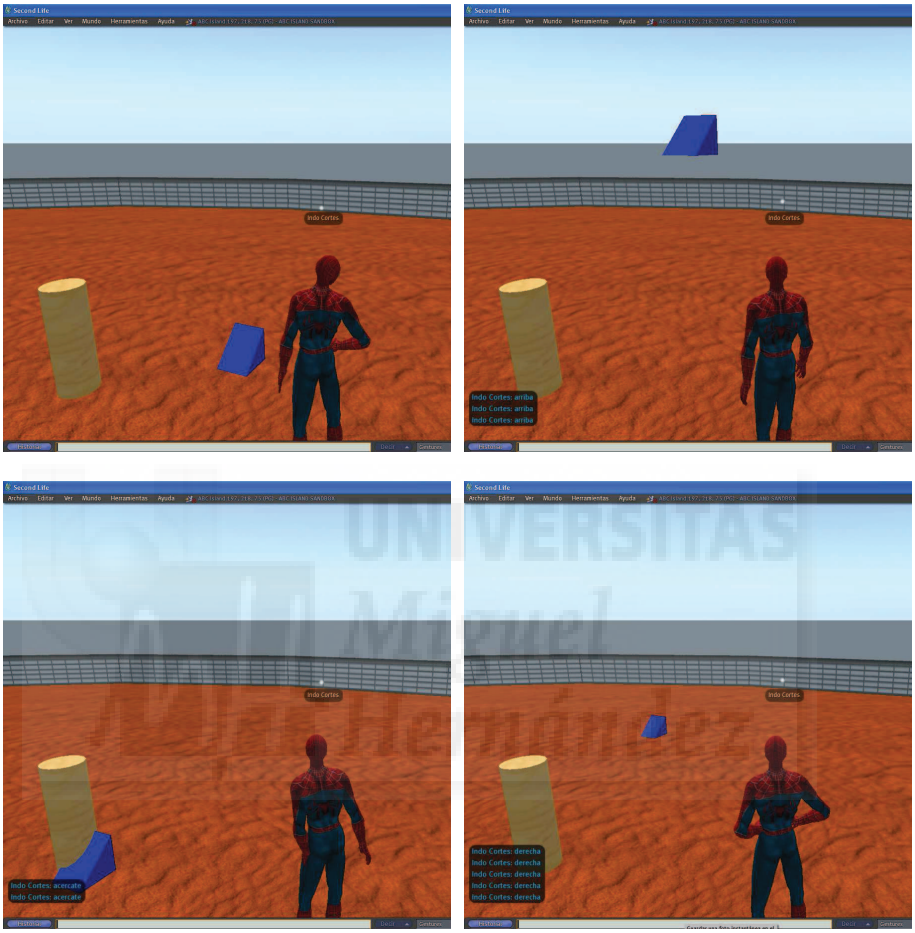


Imagen 6.9.cp.2.2: Posición de un objeto gobernada por comandos

En las imágenes de arriba se puede observar como un elemento prisma azul es posicionado dónde el avatar desea escribiendo en el chat las órdenes pertinentes. En la imagen de arriba a la izquierda se puede observar la escena recién creada, con un cilindro fijo y que sirve de referencia visual.

En la imagen de arriba a la derecha se puede observar la posición del prisma azul después de escribir 3 veces consecutivas la orden “arriba” cada una de las cuales hace que se desplace 1 metro en el eje Z en sentido positivo.

En la imagen de abajo a la izquierda se puede observar la posición del prisma azul después de escribir 2 veces consecutivas la orden “acercate” cada una de las cuales hace que se desplace 1 metro en el eje Y en sentido negativo. Podemos observar dos cosas más, una que el prisma se ha introducido dentro del cilindro. Esto es posible ya que si el cilindro no está definido como “físico” y por tanto, cualquier objeto lo puede traspasar. La segunda cuestión, pero la más importante para el tema que nos concierne, es observar cómo los ejes (en este caso el Y) no son referenciados con respecto al objeto ni a la cámara, sino con respecto a la región, por lo

que aunque digamos “acercate”, el objeto se acerca hacia el origen de esta isla de Second Life y no hacia el avatar.

Por último, en la imagen de abajo a la derecha se puede observar la posición del prisma azul después de escribir 5 veces consecutivas la orden “derecha” cada una de las cuales hace que se desplace 1 metro en el eje X en sentido negativo. En este caso pasa lo mismo que en el caso anterior, el efecto es que se desplaza en profundidad con respecto a la cámara, cuando en realidad se desplaza hacia la derecha con respecto al origen de la región.

El siguiente script lleva a cabo lo que pretendemos:

```
default
{
state_entry() {
llListen(0,"", NULL_KEY, "");
}

listen(integer channel, string name, key id, string message) {
   if (message == "arriba") { llSetPos(llGetPos()+<0,0,1>); }
   else
   if (message == "abajo") { llSetPos(llGetPos()+<0,0,-1>); }
   else
   if (message == "izquierda"){ llSetPos(llGetPos()+<1,0,0>);}
   else
   if (message == "derecha") { llSetPos(llGetPos()+<-1,0,0>); }
   else
   if (message == "alejate") { llSetPos(llGetPos()+<0,1,0>); }
   else
   if (message == "acercate") { llSetPos(llGetPos()+<0,-1,0>); }
   else
   if (message == "diagonal") { llSetPos(llGetPos()+<1,1,1>); }
   else llSay(0,"No te entiendo");
}
}
```

Podemos ver cómo se controla una 7ª entrada que es “diagonal” y es la que hace que el objeto se desplace 1 metro en los tres ejes y con sentidos positivos en los tres casos. Este caso lo hemos puesto para que quede claro que el objeto se puede desplazar a la vez en cualquier eje y sentido.

Deberíamos experimentar con este script y modificarlo para poder conseguir varios objetivos, como que se pudiera introducir el sentido con respecto a la cámara o poder tomar también del chat la cantidad de desplazamiento que queramos y no hacerlo fijo de 1 metro. También podríamos buscar un sistema que nos permita especificar los ejes y sentidos en que queremos que se produzca el movimiento.

El problema es que para solucionar estos temas debemos de controlar la creación de funciones y el paso de parámetros a estas funciones y por lo tanto perderíamos el objetivo que tiene este caso práctico.

4. Movimiento rectilíneo.

Este es el primer caso práctico en el que realmente vamos a crear un movimiento para un objeto, ya que introducimos los mecanismos para que la posición del objeto cambie a lo largo del tiempo.

El objetivo se puede alcanzar de muchas formas posibles pero vamos a utilizar la más sencilla posible. Esta forma hace que el objeto cambie su posición con respecto a su posición anterior (algo ya resuelto en el caso precedente) pero un número de veces consecutivas.

Para controlar el número de veces que se ejecuta el cambio de posición vamos a utilizar la sentencia de iteración “for”.

El código que mueve el objeto 5 metros a intervalos de 1 metro en el eje X y en sentido positivo es el siguiente:

```
for (cont=0; cont<= 5;++cont) {
        llSetPos(llGetPos()+<1,0,0>); }
```

Por supuesto, que este código no se puede introducir tal cual en el script del prisma azul. Por ejemplo, debemos definir la variable “cont”. Además para que el movimiento se produzca de forma controlada debemos aplicar estas sentencias solamente si el avatar lo pide, lo que requiere nuevamente un script interactivo.

Imagen 6.9.cp.2.3: Movimiento de un objeto gobernado por comandos

En la imagen de arriba en la izquierda se aprecian los objetos en su estado inicial.

En la imagen de la derecha se puede apreciar como el objeto ha subido 3 metros al “recibir” la orden de “marriba” (de movimiento hacia arriba) que ha tecleado el usuario en el canal 0 del chat, es decir, en el canal público.

Para que el prisma azul pueda responder a más órdenes y por lo tanto, ser mejor gobernado, hemos introducido el siguiente script:

```
integer cont;

default
{
state_entry() {
llListen(0,"", NULL_KEY, "");
}

listen(integer channel, string name, key id, string message) {

   if (message == "mizquierda") {

      for (cont=0; cont<= 3;++cont) {
         llSetPos(llGetPos()+<1,0,0>); }
      }
   else
   if (message == "malejate") {

      for (cont=0; cont<= 3;++cont) {
```

```
            llSetPos(llGetPos()+<0,1,0>); }
        }
    else
    if (message == "marriba") {

        for (cont=0; cont<= 3;++cont) {
            llSetPos(llGetPos()+<0,0,1>); }
        }
    if (message == "mabajo") {

        for (cont=0; cont<= 3;++cont) {
            llSetPos(llGetPos()+<0,0,-1>); }
        }
    else llSay(0,"No te entiendo");
}
}
```

Conclusiones:

Los movimientos no físicos deben ser creados por los usuarios (por supuesto, también se pueden comprar y vender) y esto no siempre es sencillo. Si los movimientos deben ser realistas, con inercia, fricción con el medio, etc. se convierten en algo muy complicado, por ejemplo, para vehículos, maquinaría, etc. Pero por otra parte, la mayoría de los trabajos artísticos que podemos exhibir, normalmente no requerirán más que pequeñas rotaciones a velocidades constantes, por ejemplo, para mostrar esculturas, lo que sí es bastante fácil de llevar a cabo.

Unidad 10: Interacción con los objetos.

Introducción teórica:

1. La interacción en Second Life.
2. Interacción de los avatares con los objetos.
3. Interacción por programación en LSL.
4. Los eventos más usuales en interacción.
5. Las funciones más usuales en interacción.
6. Ejemplos de uso de eventos - funciones.

Casos prácticos:

Caso práctico 10.1: Interactividad con los objetos.

1. La interacción en Second Life.

Second Life es una herramienta que proporciona una “realidad virtual”. Esto lleva implícito que estemos inmersos en un mundo, con el que interactuamos. Por lo tanto, la interacción en Second Life es algo “natural”, continuo e irrenunciable una vez que nos introducimos en él.

Podríamos distinguir dos tipos de interacciones: la interacción de nuestro “yo”, es decir de nuestro avatar con otros avatares y la interacción de nuestro avatar con los objetos.

Para empezar, creo que deberíamos tener la certidumbre de que un avatar no es un objeto más del mundo virtual. Un avatar es una representación de una persona. Por lo tanto, tenemos una entidad cuyas acciones se deben a que detrás existe una persona que las decide y ejecuta. Es como una marioneta, no es el actor en sí mismo, pero el actor actúa a través de ella. De alguna manera, es una extensión de una persona.

En nuestro caso, este tipo de interacciones no las vamos a tratar en este tema aunque sean interesantes, ya que se no se adapta al objetivo de la tesis y además ya se enumeraron estas posibilidades de desarrollo e interacción social en el unidad 2.

Por supuesto que las obras que realizamos son creadas para exhibirlas, para que las vean otras personas, experimenten con ellas, etc. Pero no vamos a estudiar aquí cómo promocionar nuestra obra o cómo realizar nuevas exposiciones. Nos ceñiremos a obras expuestas y no estudiaremos métodos para que otros avatares intervengan de forma activa en la realización o experimentación con los objetos – obras.

Por otra parte, la interacción de los avatares con el mundo que los rodea, se hace a dos niveles “de experiencias”: con los objetos y con la máquina física. Esta clase de interacción es la que más nos importa a nosotros.

En el primer lugar, tenemos objetos en Second Life y además podemos crear otros nuevos nosotros mismos. Estos objetos están compuestos por polígonos y por materiales a su vez creados por colores y texturas. La mayoría de los usuarios de Second Life suelen utilizar estos objetos para crear casas, muebles, automóviles, etc. Nosotros utilizaremos estas posibilidades para crear nuestra obra artística que puede estar compuesta por objetos representaciones de cerámica, cuadros, fotografías, esculturas, etc.

En segundo lugar, los objetos se representan con unas cualidades físicas porque existe un software funcionando, la llamada máquina física, que produce que los objetos se queden quietos o se muevan por una superficie o no los traspasemos (o si lo hagamos) cuando chocamos con ellos. Bien, pues estos “objetos virtuales” pueden ser manipulados para su exhibición, puede que en sí mismos sean esculturas móviles, etc. Pero debemos tener en cuenta que además pueden verse inmersos en simulaciones físicas como caídas, choques,...

2. Interacción de los avatares con los objetos.

El tipo de interacción que puede tener un avatar con una obra artística expuesta en Second Life es muy parecido al que puede tener en la vida real. Evidentemente si es una fuente con agua, por mucho que nos acerquemos no nos mojaremos realmente pero sí virtualmente. Pero esto lo debemos tomar como una ventaja, ya que la ausencia de leyes físicas reales nos propone casi de inmediato el hacer esculturas imposibles en la vida real como por ejemplo bajo el agua o en el interior de una casa en llamas. También podremos realizar el castillo sobre el aire de Magritte.

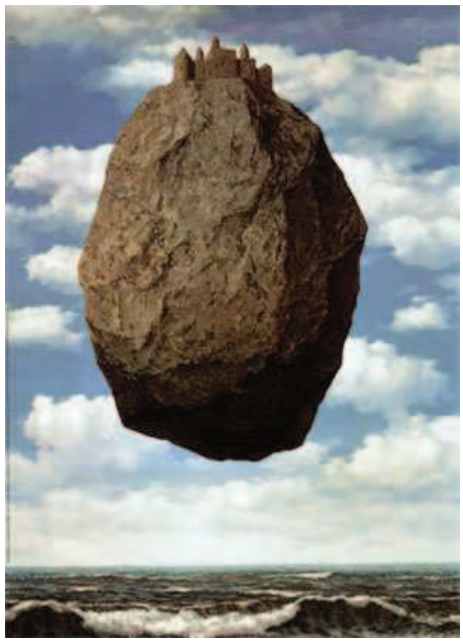


Imagen 6.10.1: Magritte es posible en Second Life

Si el tipo de obra que queremos mostrar es plana, como puede ser una fotografía, un dibujo o una pintura, la interacción del avatar puede ser simplemente el acercarse hasta la distancia idónea como para percibir los detalles. En este punto tenemos que tener en cuenta la resolución de la imagen expuesta. Para este tipo de interacción es muy importante controlar bien las cámaras. Especialmente es adecuado el comando de Focus que se activa con Alt + botón izquierdo del ratón y hacer clic sobre el objeto de estudio. El focus permite acercar o alejar la cámara sin mover el avatar y por ejemplo en una exposición de fotos, podemos ver todas las que se encuentren en una habitación en la que estemos, sin apenas movernos. También podemos utilizar la opción del teletransporte, que permite a todo avatar acceder a un punto exacto de Second Life.

Si la obra es tridimensional, pongamos una escultura, podemos tener varias alternativas para visualizarla por completo.

Una de ellas sería como en la vida real, es decir estaría expuesta de forma fija y el avatar se podría mover alrededor de ella y usar las cámaras para ver los detalles.

Otra posibilidad es que la escultura pudiera estar en constante movimiento para que mientras los avatares permanecen estáticos puedan ver la escultura por todos sus lados.

Una tercera opción es que podemos crear un objeto que actúe como una especie de mando a distancia, es decir, que el avatar pueda manipular uno o varios objetos y que estos a su vez interactúen con la escultura. Por ejemplo, podríamos crear un objeto que fuera un botón y que al ser pulsado por un avatar, una escultura rotase o se parase.

Por supuesto que también podemos hacer que la interacción del avatar sea más compleja y permita modificar partes de la escultura, rotando o cambiando de sitio distintas partes. Todas las posibilidades se pueden llevar a cabo.

3. Interacción por programación en LSL.

Hemos visto que en Director existen dos niveles para crear interacción: los behaviors o comportamientos y la programación directa. En el fondo esto es falso, ya que los behaviors no son más que códigos de programación encapsulados para que su complejidad quede interna y que así sean más fáciles de manipular. Esto quiere decir, que en el fondo, utilizamos siempre programación para crear la interacción con los objetos.

En Second Life, solo tenemos la opción de la programación en LSL. Esto no es un gran problema ya que LSL es un lenguaje muy compacto y esto significa que no es de propósito general (por ejemplo, no podemos crear un programa de contabilidad con él). Está concebido desde su diseño inicial para que el usuario de Second Life pueda crear interacciones entre los avatares y los objetos de manera rápida y sencilla aún sin tener profundos conocimientos previos de programación.

En cuanto a documentación, cuando se instala SL, se crea una página web en la misma carpeta que la aplicación que se llama "lsl_guide.htm", que nos lleva a la referencia oficial de este lenguaje. Además existen en Internet innumerables trabajos, tutoriales y recopilación bien documentada de toda clase de scripts para SL.

En Second Life, la programación está siempre "a mano", cualquier objeto de nuestra propiedad lo podemos programar y probar nuevos códigos y es reutilizable, es decir que si nos gusta lo que hace un código, lo podemos copiar fácilmente en otros objetos.

LSL es el lenguaje universal en Second Life. Esto es positivo en la medida en que esto quiere decir que no pasa como en la vida real, que existen multitud de lenguajes de programación y cada uno con unos objetivos específicos. Por lo tanto la elección no tiene lugar: LSL es el lenguaje que utilizaremos.

LSL es un lenguaje de programación moderno y funcional, que además es revisado y corregido constantemente. Cumple con todos los requisitos para considerarlo un lenguaje de última generación y por lo tanto es orientado a objetos y se ejecuta por eventos. Esto supone que el programador solo programa las funciones que responden a los eventos que quiere tener en cuenta y no a todos. O sea, que solo tiene que programar lo que verdaderamente le importa.

Podríamos decir que funciona por acciones que responden a eventos siguiendo la primera ley de newton: "Todo cuerpo persevera en su estado de reposo o movimiento uniforme y rectilíneo a no ser en tanto que sea obligado por fuerzas impresas a cambiar su estado". Por lo tanto estructuralmente es sencillo. Debemos tener en cuenta que programaremos teniendo siempre en cuenta la dupla: evento – acción. Se puede decir que el propio lenguaje de programación es interactivo intrínsecamente.

Por ejemplo, si hacemos una escultura que cuando se toca gira 10 grados, no programaremos lo que sucederá si un avatar no la toca. Y si más adelante queremos programar otra interacción nos centraremos en ella sin tener que modificar la que funciona correctamente. Por ejemplo si queremos que además de girar, responda con un saludo, su título y su precio cuando algún avatar se dirija a nuestra obra y la salude nos centraremos en resolver este problema independientemente del giro. Esto proporciona modularidad y la posibilidad que de que varios programadores se dividan la tarea.

LSL dispone de todos lo elementos para realizar la programación que necesitamos como son la utilización de variables, sentencias de asignación, control de flujo, etc., y sobre todo los elementos que estamos definiendo como más importantes en la interacción, los eventos y las funciones de biblioteca y de usuarios como acciones de respuesta.

4. Los eventos más usuales en interacción.

Los eventos que se producen en Second Life pueden ser de distinta naturaleza. Cuando escribimos un script (código en LSL) nos interesa que el prim (el objeto receptor del script) responda a unos eventos y no a otros, por lo que necesitamos "capturar" los eventos que se produzcan de un determinado tipo y poder así responder mediante una acción. Lo que escribimos en un script de programación no son eventos, sino manejadores de eventos o "event-handlers". Por ejemplo, si tenemos un timbre y queremos que el avatar obtenga una respuesta cuando toque el timbre, tendremos que capturar el evento "tocar el objeto" con el manejador de eventos "touch_stat()".

Los eventos pueden cambiar en LSL según evoluciona el propio lenguaje, hoy en día existen unos 33 manejadores de eventos, aunque debemos consultar la guía oficial en "lsl_guide.htm" para estar siempre actualizados.

Hemos determinado que los eventos más frecuentes en interacción son cinco, que en la siguiente tabla relacionamos con sus manejadores:

Evento	**Manejador**
Cuando el script comienza a ejecutarse.	state_entry()
Cuando un avatar toca un objeto en particular.	touch_start(integer total_number)
Cuando un avatar se acerca a un objeto.	sensor()
Cada cierto tiempo.	timer()
Cuando alguien dice algo específico.	listen()

Tabla 15: Los eventos más usuales en interacción

5. Las acciones más usuales en interacción.

Las acciones que se programan como respuestas a eventos son en forma de funciones. La mayoría de las funciones tendrán parámetros, es decir, variables que hacen que funcionen de una manera específica. Por ejemplo, si utilizamos la función llSetColor, que sirve para ponerle un color específico a un objeto, la función necesitará saber el color deseado así como a qué partes o caras del objeto hay que aplicarle esa modificación. Por ello, estas variables irán incluidas en el código de la función. Por lo tanto, esta podría tener una forma como: llSetColor(<0,0,1.0>,ALL_SIDES). Esto significa que si se produce el evento todo el color del objeto se cambiará a azul.

En la siguiente tabla se presentan solamente 10 de las más de 200 funciones de que dispone Second Life para interaccionar en el simulador. Son las siguientes:

Acción	**Función**
Que aparezca un objeto del inventario.	llRezObject()
Dar algo a un avatar.	llGiveInventory()
Decir algo en el chat.	llSay()
Cambiar el color de un objeto.	llSetColor()
Cambiar la textura de un objeto.	llSetTexture()
Abrir una diálogo para ver página web en SL.	llLoadURL()
Crear un texto anexo.	llSetText()
Emitir un sonido.	llPlaySound()
Mover un objeto.	llSetPos()
Rotar un objeto.	llSetRot()

Tabla 16: Las acciones más usuales en interacción

Es evidente que estas no son más que algunas funciones y por tanto la elección realizada depende de nuestras necesidades y posiblemente variará mucho. Desde el punto de vista de un artista que pretende exhibir su obra en Second Life pueden ser suficientes o no y esto nos lo dirá las experiencia que vayamos acumulando.

6. Ejemplos de uso de eventos-funciones.

Seguidamente se exponen dos pequeños scripts que tienen el objetivo de mostrar la combinación del mismo manejador de eventos (en este caso el que maneja el evento de tocar a un objeto) junto con dos respuestas (funciones de biblioteca) distintas. Se pretende que el lector aprecie la estructura evento – función que subyace en los códigos que siguen:

- Ejemplo 1: Decir algo por el chat público cuando un objeto es tocado:

```
default
{
        touch_start(integer total_number) {

                llSay(0,"Hola, me llamo “escultura 5” y mi precio son 50 L$");
        }
}
```

En la imagen siguiente se puede apreciar la secuencia que ilustra la ejecución del script escrito:

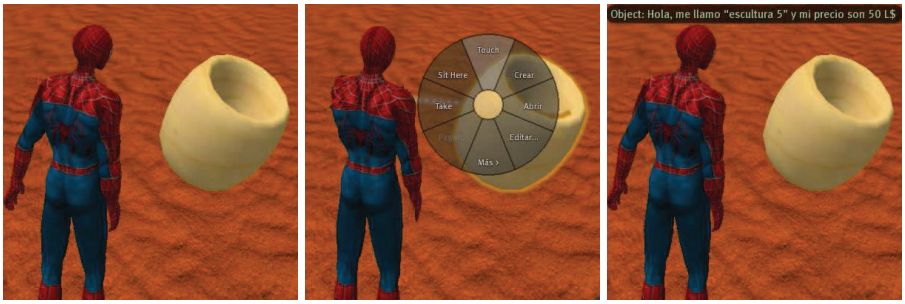


Imagen 6.10.2: El objeto contesta al ser tocado

◆ Ejemplo 2: Poner de color rojo todas las caras de un objeto al ser tocado:

```
default
{
        touch_start(integer total_number) {

                llSetColor(<1.0,0,0>,ALL_SIDES);
        }
}
```

En la imagen siguiente se puede apreciar la secuencia que ilustra la ejecución del script escrito:

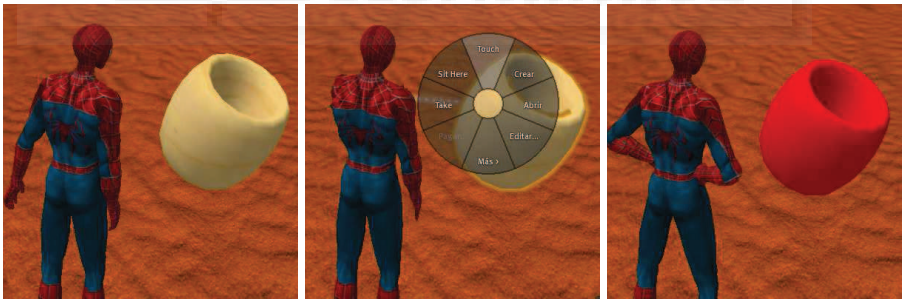


Imagen 6.10.3: El objeto cambia de color al ser tocado

Como conclusión a este tema, podemos añadir que las interacciones en Second Life son constantes e intrínsecas del metaverso y por lo tanto, si queremos exponer nuestros trabajos en este sistema, es imprescindible que sepamos crear y controlar las formas más habituales en que los habitantes están acostumbrados a manipular objetos.

Caso práctico 10.1: Interactividad con los objetos

Objetivo: Crear pequeños scripts que nos permitan programar reacciones de los objetos.

Tiempo de realización: 2 horas.

Contenidos:

1. Cómo escribir códigos interactivos en LSL.
2. Respuestas únicas al tocar un objeto.
3. Respuestas específicas al hablarle a un objeto.
4. Hablar entre objetos.

1. Cómo escribir códigos interactivos en LSL.

Los scripts serán combinaciones de manejadores de eventos y la o las funciones de respuesta a estos. Ya vimos algunos de los eventos más frecuentes en interacción:

Evento	Manejador
Cuando un avatar toca un objeto en particular.	touch_start(integer total_number)
Cuando un avatar se acerca a un objeto.	sensor()
Cada cierto tiempo.	timer()
Cuando alguien dice algo específico.	listen()

Tabla 17: Eventos y sus manejadores frecuentes en interacción

Estos eventos tienen una importancia muy distinta en el caso de la interactividad. Por ejemplo cuando un avatar toca a un objeto o se acerca a él, este puede responder de muchas formas, pero lo lógico es que sea respondido con la misma respuesta siempre, ya que el evento "tocar" solo tiene dos valores: si o no. Si el evento es que alguien dice algo específico, el objeto puede responder de muchas maneras distintas según lo que se diga. El evento del paso del tiempo puede ser importante para otras tareas pero no lo es mucho para el caso de la interacción. Por lo tanto, es más interesante desde el punto de vista de lo interactivo, el evento listen "escuchar" que el evento touch "tocar".

También adelantamos que algunas de las funciones de librería más utilizadas son:

Acción	Función
Que aparezca un objeto del inventario.	llRezObject()
Dar algo a un avatar.	llGiveInventory()
Decir algo en el chat.	llSay()
Cambiar el color de un objeto.	llSetColor()
Cambiar la textura de un objeto.	llSetTexture()
Abrir una diálogo para ver página web en SL.	llLoadURL()
Crear un texto anexo.	llSetText()
Emitir un sonido.	llPlaySound()
Mover un objeto.	llSetPos()
Rotar un objeto.	llSetRot()

Tabla 18: Funciones de librería frecuentes en interacción

Para comenzar, deberíamos intentar realizar pequeños programas que tengan efectos limitados. Normalmente, es suficiente para hacer scripts útiles, el pensar en dos pasos: el evento que queremos que sea atendido y las acciones que queremos que sean llevadas a cabo como respuesta a ese evento en particular.
Para escribir el script seguiremos los pasos típicos para programar un objeto:

1.1. Crear o arrastrar desde el inventario un objeto al sandbox.

1.2. Botón derecho sobre el objeto > Editar.

1.3. Elegir la pestaña “Contenido” > pulsar sobre el botón “Nuevo script”.

1.4. Aparece una ventana con un script por defecto que sustituimos por el nuestro.

2. Respuestas únicas al tocar un objeto.

La respuesta se generará cuando se toque a un objeto en particular. Esto evento se produce aproximándonos al objeto en cuestión y con el botón derecho pulsamos sobre él y en el menú contextual, elegimos la opción “Touch” (tocar) como muestra la siguiente imagen.

Imagen 6.10.cp.1.1: Acción que genera el evento touch_start

Ejemplo 1: Responder con una presentación cuando un objeto es tocado. Script:

```
default
{
        touch_start(integer total_number) {
                llSay(0,"Hola, me llamo “escultura 5” y mi precio son 50 L$");}
}
```

Imagen 6.10.cp.1.2: El objeto contesta al ser tocado

Ejemplo 2: Cambiar el color de un objeto al ser tocado:

```
default
{
        touch_start(integer total_number) {
                llSetColor(<1.0,0,0>,ALL_SIDES); // pone de color rojo todas la caras
        }}
```

Imagen 6.10.cp.1.3: El objeto cambia de color al ser tocado

Ejemplo 3: Cambiar la textura de un objeto al ser tocado:

```
default
{
        touch_start(integer total_number) {
                llSetTexture("beachball",ALL_SIDES);
        }
}
```

Imagen 6.10.cp.1.4: El objeto cambia de textura al ser tocado

Para que el script funcione, la textura a la que se hace referencia en el script tiene que estar "visible" desde el propio objeto. Para ello, basta con arrastrarla desde el inventario del usuario hasta la ficha "Content" (contenido) del objeto como muestra la siguiente imagen.

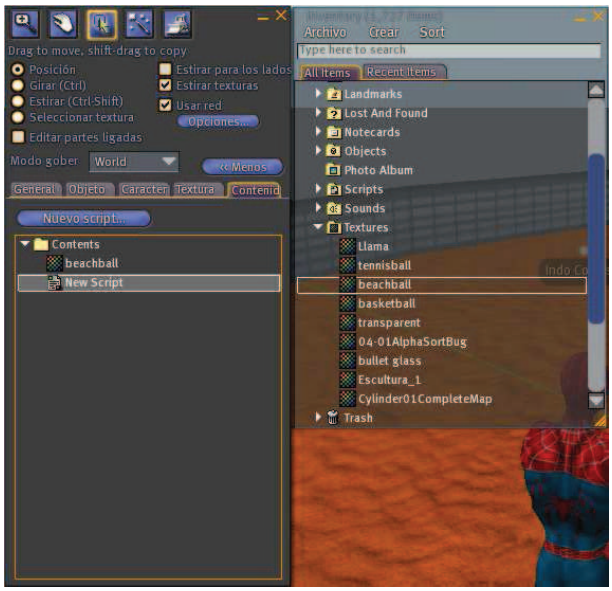


Imagen 6.10.cp.1.5: El objeto debe tener en su inventario la textura a utilizar

3. Respuestas específicas al hablarle a un objeto.

Cuando le "hablamos" a un objeto, en realidad hablamos a todo el mundo a nuestro alrededor, pero si en el objeto que nos interesa tenemos escrito un script que nos atiende podremos interactuar con él para que nos responda de muchas maneras distintas según lo que digamos.

Para ello debemos escribir en el chat público o en cualquier otro canal lo que queremos que haga el objeto y estar a unos pocos metros del objeto que debe responder.

Imagen 6.10.cp.1.6: Debemos "hablar" por el chat público

Nuestro script debe tener una determinada estructura. La manera más sencilla es capturar el mensaje escrito y responder con una serie de "ifs" anidados, de tal manera que si la palabra es aceptada se responda y si no se pase a otro "if" y se compruebe si es aceptada y así la vamos filtrando por sucesivos "ifs". Si al final de todos no a sido cierta para ningún "if" podemos hacer que responda simplemente "no te entiendo".

El siguiente script realiza varias respuestas que se comentan:

```
default
{
state_entry() {
llListen(0,"", NULL_KEY, "");
}

listen(integer channel, string name, key id, string message) {
    if (message == "ponte azul") { llSetColor(<0,0,1.0>,ALL_SIDES); } // pone de color azul
    else
    if (message == "ponte rojo") { llSetColor(<1.0,0,0>,ALL_SIDES); } // pone de color rojo
    else
    if (message == "ponte verde") { llSetColor(<0,1.0,0>,ALL_SIDES); } // pone de color verde
    else
    if (message == "rota"){ llTargetOmega(<0,0,1>,PI,1);} // gira el objeto en el eje X.
    else
    if (message == "para") { llTargetOmega(<0,0,0>,PI,1);} // deja de girar el objeto
    else
    if (message == "+peque"){ llSetScale (llGetScale()*.5);} // escala a la midad de tamaño
    else
    if (message == "+grande"){ llSetScale (llGetScale()*2);} // escala al doble de tamaño
    else llSay(0,"No te entiendo");                    // si al final si no hace nada responde
}
}
```

Imagen 6.10.cp.1.7: Objeto después de pedirle dos veces que se haga más grande

Imagen 6.10.cp.1.8: Objeto después de pedirle que se ponga azul

Imagen 6.10.cp.1.9: El objeto no nos entiende al pedirle que se ponga "morao"

4. Hablar entre objetos.

En este punto queremos resolver un tipo de interacción que podríamos llamar indirecta, en el sentido en que el usuario no interacciona con un objeto directamente sino a través de otro.

Este tipo de interacción, abre la puerta a la realización de objetos que van a actuar como controles para manipular otros objetos. Por ejemplo, podemos diseñar un objeto que cuando se le toca, pueda hacer que una escultura gire sobre su eje, podría ser algo así como un "mando a distancia".

Este tipo de objetos servirían como interfaz entre el usuario y el objeto final y podría evidenciar su función en su propio diseño. Por ejemplo, si realizamos una escultura y ponemos a disposición del usuario que se pueda manipular, se podría presentar una especie de cabina de mandos con todas las posibilidades de cambio de la escultura, y si por ejemplo, se puede girar, que existiera un objeto botón o un objeto flecha que evidenciará para lo que sirve mediante una textura o por la propia forma de la flecha.

En el siguiente ejemplo, modificamos el script anterior para que al hablarle a un objeto y pedir "escultura 1", este saque del Inventario la escultura pedida y la muestre públicamente.

```
default
{
state_entry() {
llListen(0,"", NULL_KEY, "");
}

listen(integer channel, string name, key id, string message) {
if (message == "ponte azul") { llSetColor(<0,0,1.0>,ALL_SIDES); } // pone de color azul
else
if (message == "ponte rojo") { llSetColor(<1.0,0,0>,ALL_SIDES); } // pone de color rojo
else
if (message == "ponte verde") { llSetColor(<0,1.0,0>,ALL_SIDES); } // pone de color verde
else
if (message == "rota"){ llTargetOmega(<0,0,1>,PI,1);} // gira el objeto en el eje X.
else
if (message == "para") { llTargetOmega(<0,0,0>,PI,1);} // deja de girar el objeto
else
if (message == "+peque"){ llSetScale (llGetScale()*.5);} // escala a la midad de tamaño
else
if (message == "+grande"){ llSetScale (llGetScale()*2);} // escala al doble de tamaño
else
if (message == "escultura 1") {
```

```
llRezObject("Macetero", llGetPos() + <0, 0, 2>, ZERO_VECTOR, ZERO_ROTATION, 42);
}                                   // hace que aparezca otro objeto
else llSay(0,"No te entiendo");             // si al final si no hace nada responde
}
}
```

Conclusiones:

La interactividad con los objetos de Second Lilfe tiene a favor que todo el sistema está pensado para ser manipulado y además se pone en manos del usuario una serie de herramienta que tienen mucha potencia. Por otra parte, la principal herramienta para crear interacción es hacer uso del sistema de programación inherente a Second Life, lo que conlleva la necesidad de escribir código.

Unidad 11: Sistemas de partículas en Second Life.

Introducción teórica:

1. Los sistemas de partículas en Second Life.
2. Implementación en Second Life.
3. Función para controlar los sistemas de partículas.
    3.1. Script mínimo para crear sistemas de partículas.
    3.2. Script para desactivar un sistema de partículas.
    3.3. Sintaxis de la función llParticleSystem()
    3.4. Parámetros de la función llParticleSystem()
4. Apariencia de los sistemas de partículas.
5. Flujo de los sistemas de partículas.
6. Movimiento de los sistemas de partículas.
7. Posicionamiento de los sistemas de partículas.

Caso práctico:

Caso práctico 11.1: Crear un sistema de partículas desde el inicio: Fuego

1. Los sistemas de partículas en Second Life

Los sistemas de partículas son elementos ampliamente utilizados para efectos visuales en juegos, exposiciones artísticas, mundos virtuales, etc.

Con los sistemas de partículas en Second Life podemos crear elementos como fuegos artificiales, brillos de mentales, efectos meteorológicos como lluvia, nieve, niebla,... y también grupos de animales como cardúmenes de peces, grupos de mariposas, bandadas de pájaros, etc.

Para trabajar con estos elementos es necesario que tengamos claro dos conceptos: partícula y sistemas de partículas.

Una partícula en Second Life es un elemento bidimensional con una textura que forma parte de un grupo creado con un fin propio. No tiene sentido sino es dentro de ese grupo. No se manejan individualmente, por lo tanto es un concepto distinto al de prim.

Las partículas tienen las siguientes características:

1. Es una imagen bidimensional, no un prim.

2. Siempre se ven sin ángulos, aunque los avatares roten siempre los ven de frente.

3. Emanan siempre del centro de un prim llamado emisor.

Un sistema de partículas es un conjunto de elementos bidimensionales, llamados partículas que forman una entidad propia tanto estética como funcional.

Los elementos estructurales para crear un sistema de partículas son 3:

1. Un prim emisor.

2. El prim emisor contendrá un script que incluye la función llParticleSystem.

3. Una textura.

2. Implementación de sistemas de partículas en Second Life.

En Second Life, los sistemas de partículas no se crean como todos los demás objetos tridimensionales, ya que no lo son. Por lo tanto, no los podemos crear y modificar con el modelador incorporado en SL. En su lugar, para crearlas, utilizaremos una función de LSL llamada llParticleSystem.

LlParticleSystem es una función muy compleja que posee 28 parámetros con los cuales se pueden definir sistemas realmente complejos.

El software encargado de crear y mostrar las partículas es el cliente SL, es decir, el programa que instalamos en nuestro ordenador para entrar en SL. Esto significa que su visualización no depende de los servidores de Internet propiedad de Linden Labs y por tanto no producen lag, es decir, retardo en la visión e interacción del mundo virtual. La parte negativa es que no todos los avatares visualizan los sistemas de partículas exactamente de la misma forma.

En el cliente de Second Life podemos configurar la forma de visualizar las partículas. Con la orden del menú View > Beacons > Hide Particles podemos activar o no la opción de ver las partículas. En el mismo sitio tenemos también la opción “Particle Sources” que podrá visualizar o no el prim emisor del sistema de partículas.

También podemos controlar el número máximo de partículas que se visualizarán. Este aspecto lo configuramos en el menú Edit > Preferences > Graphics. Pulsamos el botón “Custom” y

podemos ver “Max. Particle Count” a un valor de 1024 pero que podemos modificar desde 0 hasta 8192.

El sistema de partículas no será exactamente igual en todas las circunstancias, ya que existen parámetros externos al script que lo genera, que lo pueden alterar. Estas circunstancias son tres: el viento, el movimiento del prim emisor y el movimiento del objetivo. Ya veremos que el objetivo o target será un avatar u objeto al que llegan las partículas y si se está moviendo, es evidente que todo el sistema variará en función de esto, al igual que si el viento es muy fuerte o si el prim emisor está cambiando de localización.

Los sistemas de partículas también tienen límites, por ejemplo:

◆ Las partículas tienen una “vida” máxima de 30 segundos y no pueden medir más de 4 x 4 metros ni menos de 4 x 4 centímetros.

◆ El emisor de partículas puede ser cualquier prim menos árboles o hierba.

En Second life, el lugar más conocido para aprender sobre los sistemas de partículas se llama Particle Lab, de cuya entrada tenemos una captura de pantalla en el siguiente imagen. Este sitio está creado especialmente para adquirir conocimientos sobre un tema tan complejo por lo que el enfoque es totalmente didáctico y contiene múltiples ejemplos y explicaciones organizadas convenientemente.

Imagen 6.11.1: Entrada a “Particle Lab”

3. Función para controlar los sistemas de partículas.

3.1. Script mínimo para crear sistemas de partículas.

En primer lugar vamos a implementar el mínimo script para generar un sistema de partículas. Como hemos escrito antes, los sistemas de partículas solo se pueden crear por un método: la llamada a la función llParticleSystem() dentro de un script en código LSL.

Esta función es especialmente compleja en cuanto a la cantidad de parámetros que admite debido a que la mayoría de esos parámetros no son de obligada escritura. Ya que el sistema

les da valores por defecto, podemos escribir un script mínimo para obtener un sistema de partículas de manera relativamente sencilla.

Por lo tanto, si tenemos cualquier prim creado, podemos asignarle un script como el que sigue para obtener nuestro primer sistema de partículas.

```
default
{
   state_entry()
   {
         llParticleSystem ([
                  PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_EXPLODE
         ]);
   }
}
```

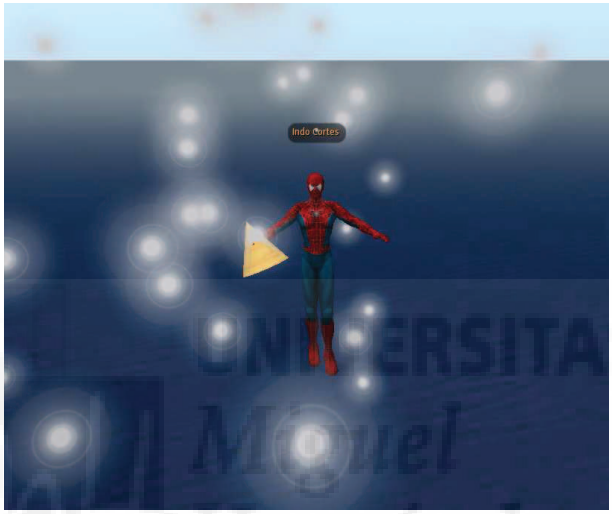


Imagen 6.11.2: El sistema de partículas con el script por defecto

En la imagen anterior se puede observar el prim emisor, que en este caso es un cono, en la mano de mi avatar. El sistema de partículas que aparece utiliza todos los parámetros con valores por defecto menos PSYS_SRC_PATTERN y este hace que precisamente se visualice el sistema de partículas. El valor que se asigna a “pattern” es “explode” aunque podría ser cualquiera de los cuatro posibles que admite y que ya estudiaremos.

3.2. Script para desactivar un sistema de partículas.

El script más pequeño que podemos escribir para desactivar un sistema de partículas ya existente es muy sencillo, ya que si ponemos todos los parámetros por defecto, desactivamos el sistema, pues el valor por defecto es off para el parámetro de visualización, y de esta forma tendríamos un método para quitar un sistema de partículas de la escena. Por lo tanto, si modificamos el anterior script (simplemente borrando la línea central) desactivamos el sistema de partículas, aunque no lo hacemos desaparecer, quedando de la siguiente forma.

```
default
{
   state_entry()
   {
         llParticleSystem ([]);
   }
}
```

3.3. Sintaxis de la función llParticleSystem().

La sintaxis de esta función, nos indica que para escribir varios parámetros los separaremos con una coma del valor que toma y con otra coma del siguiente parámetro. El último parámetro no tendrá coma al final, tal como muestra el siguiente ejemplo.

```
default
{
   state_entry()
   {
      llParticleSystem ([
          PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_EXPLODE,
          PSYS_PART_MAX_AGE, 1.0
      ]);
   }
}
```

3.4. Parámetros de la función llParticleSystem().

Como hemos visto antes, los sistemas se crean y manipulan mediante la función llParticleSystem que tiene 28 parámetros para realizar toda clase de partículas.

La traducción al inglés de cada parámetro nos puede ir muy bien a la hora de escribirlos en la función, por lo que junto al concepto en español, añadiremos su traducción al inglés y su forma de escribirlo en la función.

Partiendo del script mínimo podemos modificar muchos parámetros para ir estudiándolos. Por ejemplo, parámetros como scale, color o alpha modifican aspectos básicos de la apariencia de las partículas, pero son demasiados como para “ir probando”.

Cómo podemos deducir, es de suma importancia el estudio metodológico de los parámetros más importantes. Una solución es clasificarlos o agruparlos por algún criterio como puede ser su función. Todo ello, hace que los parámetros se puedan dividir en cuatro tipos:

1. Apariencia: modificando estos parámetros podemos variar el aspecto de las partículas en tamaño, color, opacidad y textura.

2. Flujo: se configuran datos que tienen que ver con la descripción de la “corriente” de las partículas como la cantidad y el tiempo de vida de las partículas y otros que afecta al emisor en cuanto al tiempo entre explosiones, el tiempo en que el emisor está activo, etc.

3. Movimiento: se modifican variables que intervienen en la dirección que toman las partículas, la velocidad y aceleración que poseen, resistencia al viento, etc.

4. Posicionamiento: en este apartado incluiremos los parámetros que afectan al modo y la forma de emisión de las partículas. Por ejemplo, el modo continuo o en explosión.

4. Apariencia de los sistemas de partículas.

Los parámetros de apariencia nos permiten modificar el aspecto y cómo cambian a la largo de su vida las partículas. Por ejemplo, si queremos tener un sistema dónde las partículas sean rojas y casi transparentes al principio y terminen siendo verdes y totalmente opacas modificaremos 4 parámetros. Si un parámetro queremos que no cambie, basta con ponerlo al principio como queremos y no modificarlo al final, es decir, no se tienen porque utilizar estos datos por parejas. Los parámetros involucrados modifican los valores de: tamaño (scale), color (color), opacidad (alpha) y textura (texture). Vamos a estudiarlos uno por uno.

TAMAÑO, SCALE: afecta al tamaño de las partículas al principio y al final de su vida. Sus valores pueden ir desde 4 metros a 4 centímetros. Se utilizan valores 3D, pero al ser un elemento 2D, el último siempre será 0, no tiene profundidad. El siguiente código producirá partículas del tamaño máximo que se irán reduciendo hasta el mínimo:

PSYS_PART_START_SCALE, <0.04, 0.04, 0.0>
PSYS_PART_END_SCALE, <2.0, 2.0, 0.0>

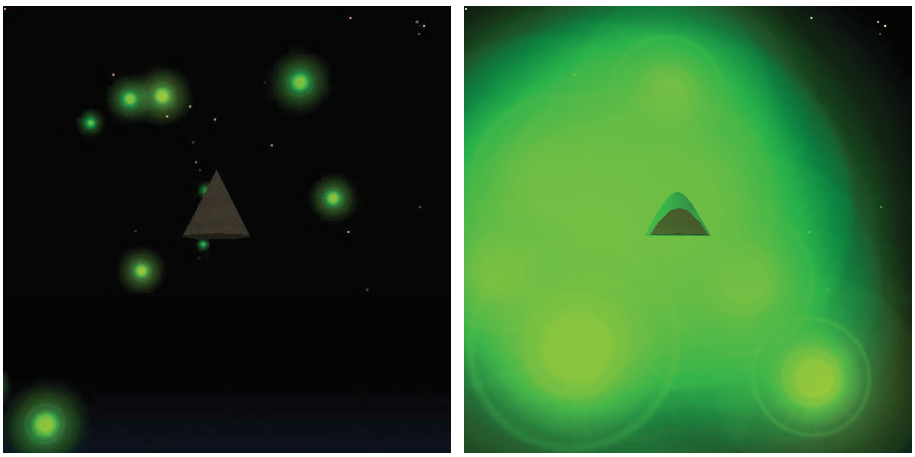


Imagen 6.11.3: El mismo sistema de partículas pero con tamaños distintos

En las imágenes de arriba se observa cómo cambia un mismo sistema de partículas si se modifica el tamaño de las mismas. En la izquierda tenemos partículas de 4 x 4 centímetros y en la derecha tenemos las mismas partículas pero de 4 x 4 metros.

COLOR, COLOR: el color de las partículas al principio y al final de su vida pueden ser controlados mediante código como el que se muestra abajo que harán que las partículas sean al comienzo rojas y se conviertan en verdes al final. Los valores se introducen en código RGB pero con un rango de 0 a 1 en vez de 0 a 255.

PSYS_PART_START_COLOR, <1.0, 0.0, 0.0>
PSYS_PART_END_COLOR, <0.0, 1.0, 0.0>

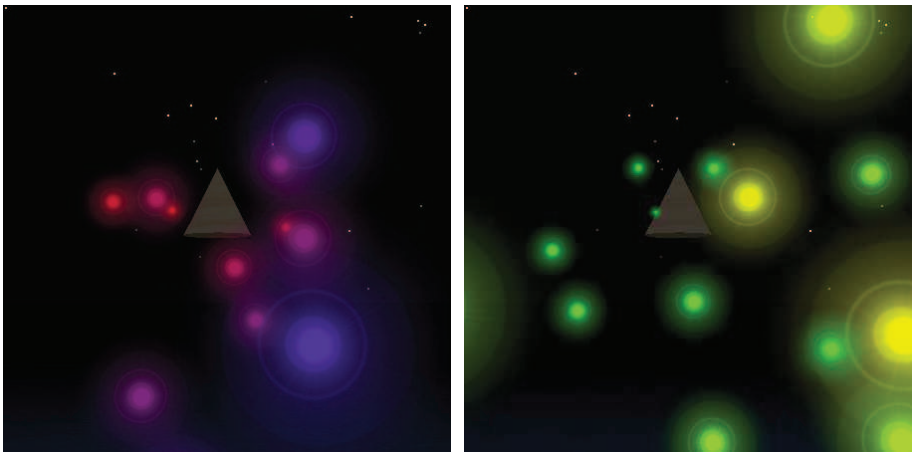


Imagen 6.11.4: El mismo sistema de partículas pero con colores distintos

En las imágenes de arriba se observa el mismo sistema de partículas pero con colores distintos. En la imagen de la derecha el color inicial de las partículas es rojo y acaban en azul y en la izquierda empiezan siendo verdes y finalizan su vida de color amarillo.

OPACIDAD, ALPHA: este parámetro puede tomar valores entre 0 (totalmente transparentes) y 1 (totalmente opacas). Existe un problema y es que la variación puede ir de 1 a 0 y no de 0 a 1 por un problema con el actual cliente de SL. El siguiente código hará que las partículas vayan desapareciendo a lo largo de su vida.

PSYS_PART_START_ALPHA, 1.0
PSYS_PART_END_ALPHA, 0.0

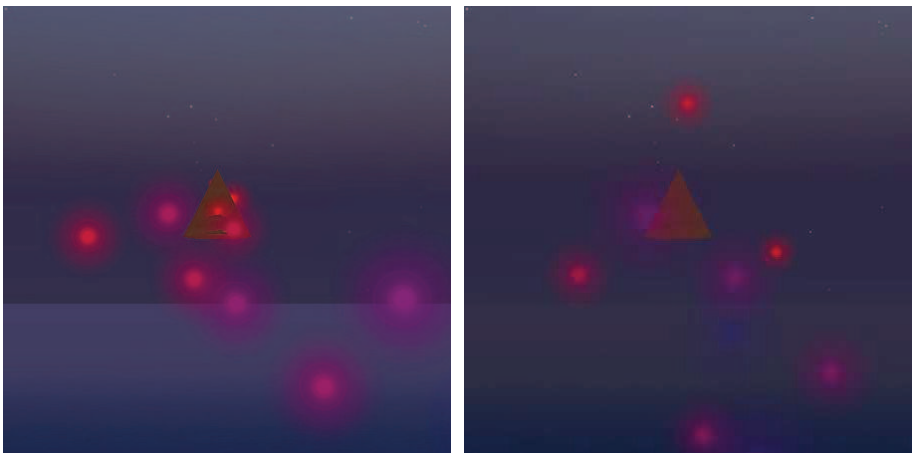


Imagen 6.11.5: Partículas con y sin alpha al final

En la imagen anterior a la izquierda se observa a las partículas que aunque tienen una opacidad en parte de su textura, mantiene la misma a lo largo de su vida, mientras que a la derecha se visualiza como van desapareciendo al ir cambiando alpha a lo largo de su vida.

BRILLO: antes de seguir viendo otras características debemos tener en cuenta que tanto el color como la transparencia, se ven afectados por otras variables como pueden ser las luces del entorno, la luz atmosférica, etc. En ambos casos, la solución es la misma: hacer que las partículas se muestren con la máxima intensidad de brillo posible. Para ello utilizaremos el parámetro PSYS_PART_FLAGS con el valor PSYS_PART_EMISIVE_MASK. Por lo tanto escribiremos el siguiente código:

PSYS_PART_FLAGS, (PSYS_PART_EMISIVE_MASK)

Imagen 6.11.6: Partículas con y con brillo mediante PSYS_PART_EMISIVE_MASK

En la imagen anterior a la izquierda se observa el sistema de partículas perfectamente mientras en la derecha apenas se distingue por que es de noche. Este parámetro es de tipo flag. Las flags son variables booleanas que pueden tomar solamente dos valores on u off. Las mask o máscaras son activadores de flags, de tal manera que puede poner en juego las flags.

TEXTURA, TEXTURE: este parámetro es el que modificará espectacularmente el aspecto del sistema de partículas. Pongamos un ejemplo: no es lo mismo una cascada que una fina lluvia, pero en ambos casos necesitaríamos una textura que simulará una gota de agua. El modo de ser emitida hará que sea una cosa u otra, pero la textura será la misma. Para poner una textura utilizaremos el siguiente código:

PSYS_SRC_TEXTURE, “uuid”

Siendo uuid el identificador de textura de nuestro inventario. Para obtenerlo basta con ir al inventario y con el botón derecho pulsar sobre la textura deseada, aparecerá la opción Copy Asset HUID, luego hacemos un “pegar” entre las comillas y ya está.

El siguiente script es el que produce el cambio de la textura que se observa en las imágenes que siguen.

```
default
{
   state_entry()
   {
        llParticleSystem ([
      PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_EXPLODE,
      PSYS_SRC_TEXTURE, "bca29a42-c2f3-bf92-c614-ed91654c7bf7"
]);
   }
}
```

Otra opción es utilizar el nombre de la textura directamente, pero para ello la debemos de incluir en la sección Content del prim emisor.

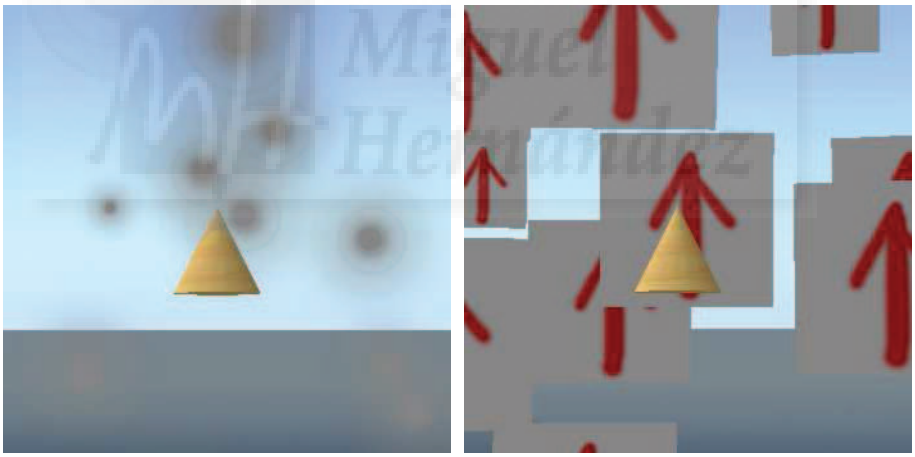


Imagen 6.11.7: Partículas con texturas por defecto y modificada

5. Flujo de los sistemas de partículas.

Las variables que vamos a estudiar en este apartado controlaran la manera en que las partículas estarán visibles.

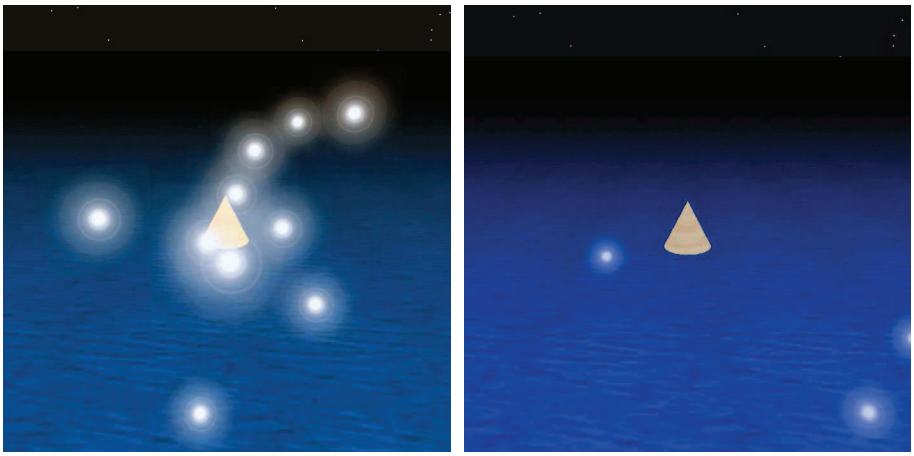


Imagen 6.11.8: Sistemas de partículas con una edad de emisor baja

Por ejemplo, si queremos un sistema que simule una fuente, podemos hacer que cuando expulse el agua hacia arriba, llegue más o menos alto. Esto lo haremos al ponerle valor a los siguientes conceptos: tiempo de vida del sistema, tiempo de vida de las partículas, cantidad de "explosiones" por segundo y número de partículas emitidas en cada "explosión". Estudiaremos los parámetros asociados con cada uno de estos conceptos:

EDAD MÁXIMA DEL EMISOR, SCR_MAX_AGE: admite un float y especifica la cantidad de tiempo, en segundos, que el emisor funcionará. Cuando acabe, no se emitirá ninguna partícula más, a excepción de que se especifique mediante otros valores. Un valor de 0 dará una duración infinita al sistema. Por ejemplo, el parámetro PSYS_SRC_MAX_AGE, 5.0 hará que el sistema dure 5 segundos emitiendo partículas. En las imágenes de abajo se observa el sistema a los 2 segundos y a los 10 segundos después de iniciarse.

EDAD MÁXIMA DE LAS PARTÍCULAS, PART_MAX_AGE: admite un float, y especifica el tiempo de vida de cada partícula emitida medida en segundos. El valor máximo será de 30.0 segundos. Durante este tiempo, la partícula aparece, cambia de apariencia y se mueve de acuerdo a los parámetros especificados en otras secciones y al final, desaparece.

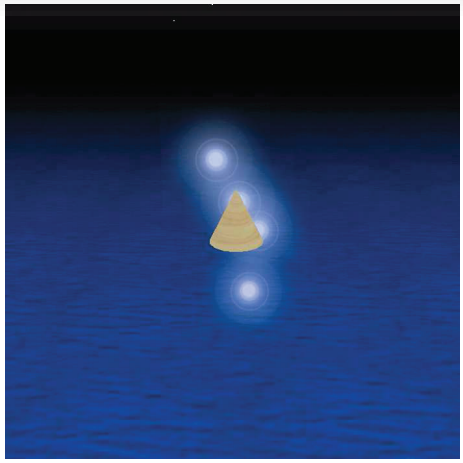


Imagen 6.11.9: Sistemas de partículas con una edad de partículas baja

Si es un tiempo corto, no llegarán muy lejos, como se aprecia en la imagen 5.2.8. Las partículas del sistema con la línea PSYS_PART_MAX_AGE, 1.0 solo durarán un segundo, aunque la vida del emisor sea de 5 segundos.

TIEMPO ENTRE EXPLOSIONES, SRC_BURST_RATE: admite un float. Especifica el tiempo, en segundos, entre explosiones de partículas emitidas. Si le damos el valor 0.0, la velocidad de explosiones será tan rápida como el visualizador sea capaz de generar y generará un sistema

de partículas muy compacto, que no cesará de emitir partículas continuamente. Esto se puede apreciar en la imagen de abajo a la izquierda, sin embargo solo con cambiar este parámetro al valor de 1 segundo, hace que el sistema cambie totalmente como se comprueba en la imagen de la derecha.

PSYS_SRC_BURST_RATE, 0.0

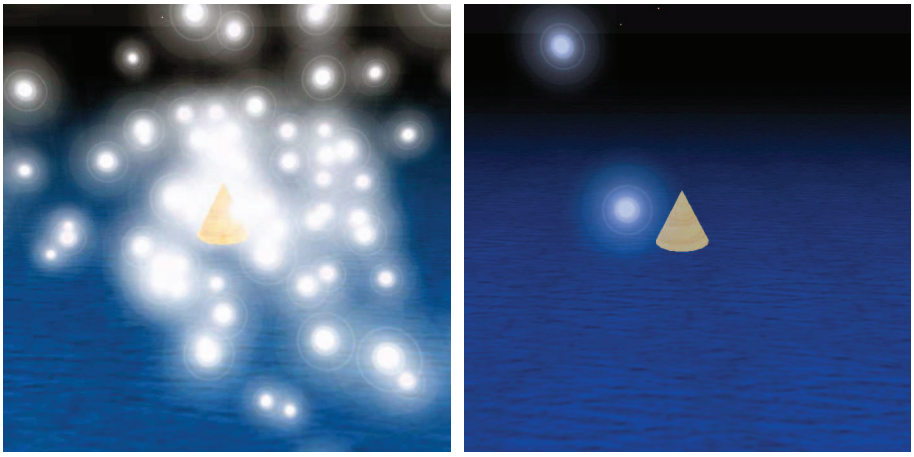


Imagen 6.11.10: Tiempo entre explosiones de 0 y 1 segundo

Nº DE PARTÍCULAS POR EXPLOSIÓN, SRC_BURST_PART_COUNT: admite un integer. Especifica el número de partículas emitidas en cada explosión. En la imagen 6.11.11 se puede ver la gran diferencia entre ponerle a este parámetro el valor 2 (a la izquierda) o 200 (a la derecha). Abajo se muestra cómo quedaría la línea de script:

PSYS_SRC_BURST_PART_COUNT, 200

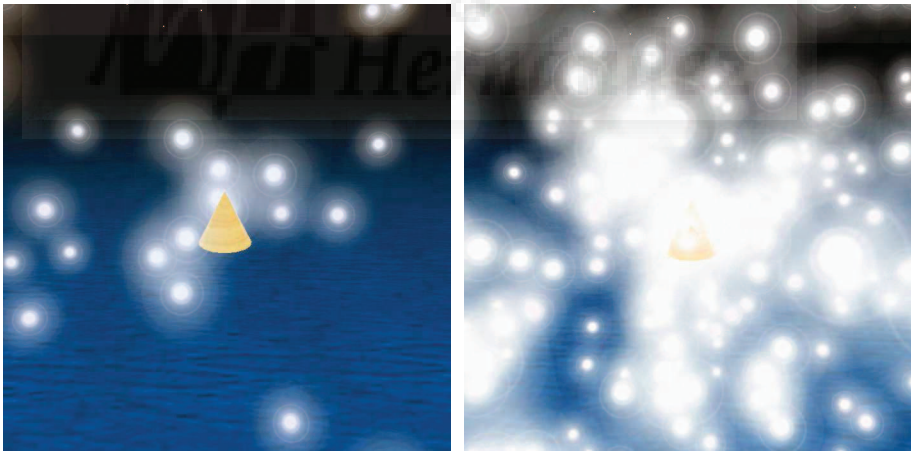


Imagen 6.11.11: Número de partículas con valores 2 y 200

6. Movimiento de los sistemas de partículas.

En el apartado de parámetros que controlan el movimiento de las partículas, vamos a ver por ejemplo, cómo modificar la velocidad y aceleración inicial con que son emitidas las partículas. Esto provocará que el resto de su movimiento sea más lento o más rápido. Los parámetros que vamos a estudiar son:

ACELERACIÓN DEL EMISOR, SRC_ACCEL: admite un vector. Especifica un vector de aceleración direccional aplicada a cada partícula cuando es emitida, en metros por segundo. Por lo tanto se puede utilizar para dirigir las partículas hacia donde queramos. Los valores

válidos son de 0.0 a 100.0 para cada dirección según las coordenadas de la región. Por ejemplo, si queremos que las partículas salgan rápidamente "disparadas" hacia abajo, escribiremos:

PSYS_SRC_ACCEL, <0.0, 0.0, -50.0>

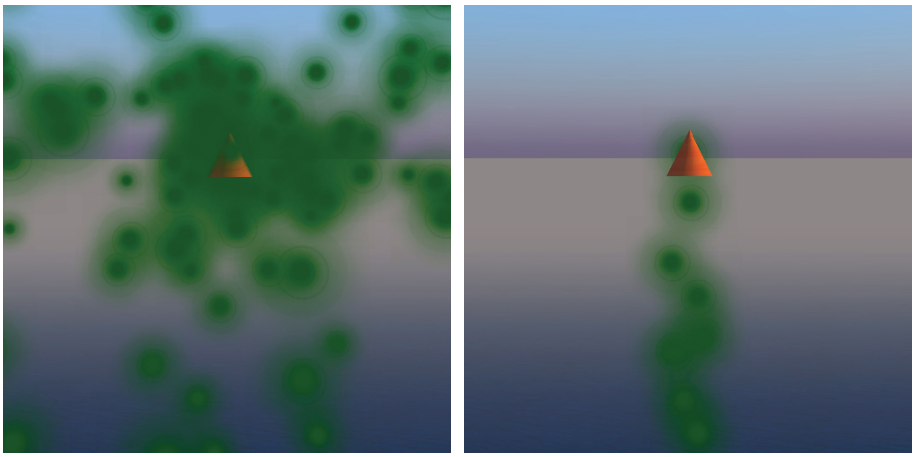


Imagen 6.11.12: Control de la dirección del sistema de partículas

GIRO DEL EMISOR, SRC_OMEGA: admite un vector. Especifica el giro rotacional del emisor en radianes por segundo a lo largo de cada eje. El emisor encara este desfase desde el eje positivo Z del prim y se visualiza mejor en las presentaciones direccionales. El giro del prim (vía llTargetOmega) no tiene efecto sobre el giro del emisor, es decir, es independiente. La siguiente línea hará que gire un radian por segundo en el eje Y.

PSYS_SRC_OMEGA, <0.0, 1.0, 0.0>

MÍNIMA VELOCIDAD DE EXPLOSIÓN, SRC_BURST_SPEED_MIN: admite un float. Especifica el mínimo valor de un rango aleatorio de valores que es seleccionado para la velocidad inicial (desde la emisión) de cada partícula de una explosión, en metros por segundo. Es decir, la velocidad mínima de salida de la explosión. Debemos tener en cuenta que el valor de este parámetro y el de PSYS_SRC_BURST_SPEED_MAX son examinados por el cliente de SL de tal manera que el parámetro tomará el más pequeño de los dos valores. Por ejemplo:

PSYS_SRC_BURST_SPEED_MIN, 50.0

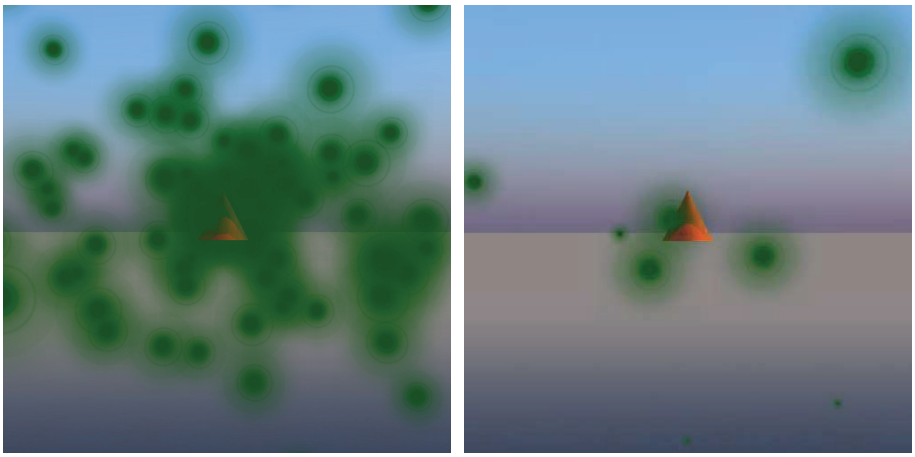


Imagen 6.11.13: Velocidad de salida mínima de 0 y 50 metros por segundo

MÁXIMA VELOCIDAD DE EXPLOSIÓN, SRC_BURST_SPEED_MAX: admite un float. Especifica el máximo valor de un rango aleatorio de valores que es seleccionado para la

velocidad inicial (desde la emisión) de cada partícula de una explosión, en metros por segundo. Es decir, la velocidad máxima de salida de la explosión. Debemos tener en cuenta que el valor de este parámetro y el de PSYS_SRC_BURST_SPEED_MIN son examinados por el cliente de SL de tal manera que el parámetro tomará el más grande de los dos valores. Por ejemplo:

PSYS_SRC_BURST_SPEED_MAX, 10.0

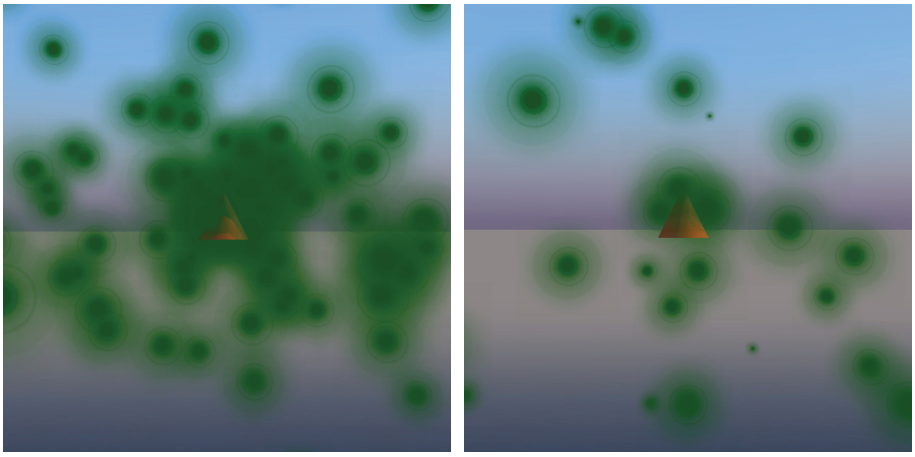


Imagen 6.11.14: Velocidad de salida máxima de 0 y 10 metros por segundo

Se puede apreciar en las imágenes 5.2.12 y 5.2.13 que se puede conseguir el mismo efecto con cada una de las variables. En el caso de la velocidad de salida mínima de 50 metros por segundo, al salir a tan gran velocidad, las partículas se alejan muy rápidamente del emisor, por lo que a pesar de expulsar la misma cantidad de partículas en todos los casos, el emisor se visualiza mucho más libre de partículas alrededor.

7. Posicionamiento de los sistemas de partículas.

En este apartado estudiamos los parámetros que están relacionados con la forma de la emisión de las partículas. Esto puede hacer que las partículas salgan de forma continua o de forma discreta, es decir, como explosiones. Los parámetros que vamos a estudiar son:

MODELO DE FUENTE, PSYS_SRC_PATTERN: admite un entero. Especifica el modelo de emisión. El valor de este parámetro solo puede tomar 5 posibles valores:

PSYS_SRC_PATTERN_DROP	*0x01*
PSYS_SRC_PATTERN_EXPLODE	*0x02*
PSYS_SRC_PATTERN_ANGLE	*0x04*
PSYS_SRC_PATTERN_ANGLE_CONE	*0x08*
PSYS_SRC_PATTERN_ANGLE_CONE_EMPTY	*0x10*

Vamos a utilizar un mismo sistema de partículas y modificaremos este parámetro para ver claramente sus diferencias. El sistema de partículas se creará con el siguiente script:

```
default
{
   state_entry()
   {
         llParticleSystem( [
               PSYS_SRC_PATTERN, PSYS_SRC_EXPLODE, // o el que sea
               PSYS_SRC_ANGLE_BEGIN, 0.0,
               PSYS_SRC_ANGLE_END, PI,
               PSYS_PART_FLAGS, PSYS_PART_INTERP_SCALE_MASK |
               PSYS_PART_INTERP_COLOR_MASK,
               PSYS_PART_START_SCALE,< .1, .1, 0 >,
```

```
                PSYS_PART_END_SCALE,< 5.0, 5.0, 0 >,
                PSYS_PART_START_COLOR,<1,0,1>,
                PSYS_PART_END_COLOR,<0,1,0>,
                PSYS_PART_START_ALPHA,1.0,
                PSYS_PART_END_ALPHA,0.1,
                PSYS_SRC_BURST_PART_COUNT,50,
                PSYS_SRC_BURST_RATE,1.0,
                PSYS_PART_MAX_AGE,5.0
        ] );
    }
}
```

Los parámetros que siguen a continuación intervienen en los modos angulares por lo que adelantamos su explicación para comprender los modos mejor.

COMIENZO DEL ÁNGULO, PSYS_SRC_ANGLE_BEGIN: admite un float. Especifica una sección circular en radianes (teniendo por referencia la rotación del emisor) donde las partículas no pueden ser emitidas.

FINAL DEL ÁNGULO, PSYS_SRC_ANGLE_END: admite un float. Especifica una sección circular en radianes (teniendo por referencia la rotación del emisor) donde las partículas no pueden ser emitidas. Si el valor es 0.0 el resultado será que las partículas se podrán emitir en una estrecha línea en la dirección de la primera cara del emisor. Si toma el valor de PI, las partículas serán emitidas en círculo o en un arco esférico alrededor del emisor.

Este parámetro junto con el PSYS_SRC_ANGLE_BEGIN son internamente revisados de tal manera que se toma el mayor de los dos valores.

MODELO EXPLOSIÓN, PSYS_SRC_PATTERN_EXPLODE: presenta las partículas siendo disparadas desde el emisor en todas las direcciones de acuerdo con los valores del movimiento de la explosión. Este modo se puede utilizar para crear muchos efectos especiales, desde explosiones a fuegos, destellos, etc. En la figura 6.11.15 se puede apreciar este tipo de modelo.

Imagen 6.11.15: Modelo de fuente tipo explosión

MODELO CAIDA, PSYS_SRC_PATTERN_DROP: presenta las partículas cayendo desde el emisor sin fuerza. Esto hace que si cambiamos la escala inicial y final produzcan pulsiones de las partículas o combinada con la función PSYS_SRC_ACCEL que el sistema se vuelva lineal (sino rotamos el emisor). Normalmente se utilizan para producir resplandores o brillos fuertes o en automóviles cuando queramos que dejen una estela. Ignora los valores para PSYS_SRC_BURST_RADIUS, SPEED_MIN y SPEED_MAX.

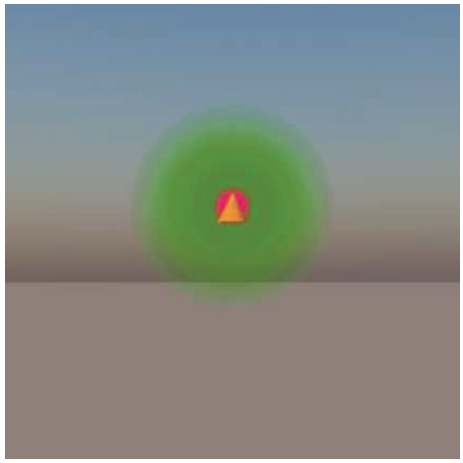


Imagen 6.11.16: Modelo de fuente tipo drop

MODELO ANGULAR, PSYS_SRC_PATTERN_ANGLE: presenta las partículas en una sección circular de dos dimensiones definidas por PSYS_SRC_ANGLE_BEGIN y PSYS_SRC_ANGLE_END. La orientación del plano de la partícula es X=0 con respecto a la rotación de la región a la que pertenece el emisor. Si no se le da valores a estos parámetros el sistema se presenta de forma lineal tal como se muestra en la imagen siguiente.

Imagen 6.11.17: Modelo de fuente tipo angular sin definir el comienzo y fin del ángulo

Los parámetros PSYS_SRC_ANGLE_BEGIN y PSYS_SRC_ANGLE_END se han modificado a 0.0 y PI respectivamente en la imagen 6.11.18 y a PI/2 y PI en la imagen 6.11.19.

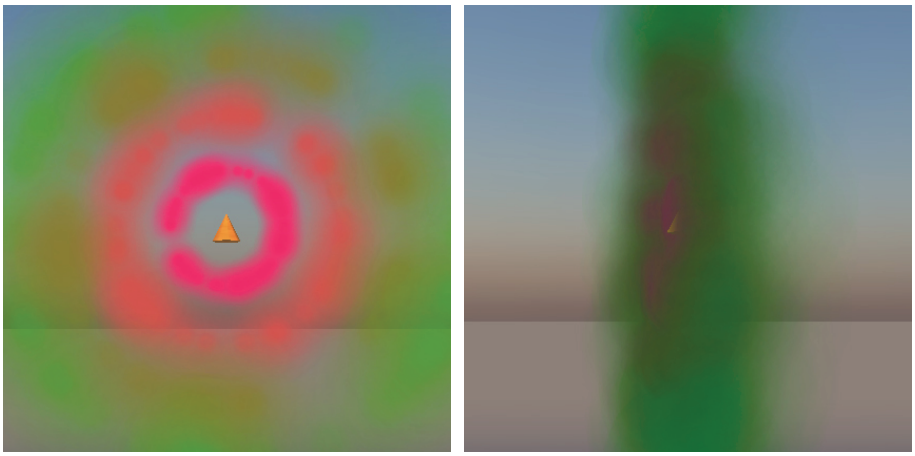


Imagen 6.11.18: Vista frontal y lateral del modelo de fuente tipo angular

Imagen 6.11.19: Vista frontal y lateral de fuente tipo angular modificada

En las dos imágenes precedentes se puede observar cómo el modo “angle” se visualiza en dos dimensiones. Muy parecido al anterior modo pero con la posibilidad de definir un espacio en tres dimensiones es el modo que sigue.

MODELO CONO ANGULAR, PSYS_SRC_PATTERN_ANGLE_CONE: Presenta las partículas en una sección esférica tridimensional definidas por PSYS_SRC_ANGLE_BEGIN y PSYS_SRC_ANGLE_END. Si no se le da valores a estos parámetros el sistema se presenta de forma lineal tal como se muestra en la imagen 6.11.20.

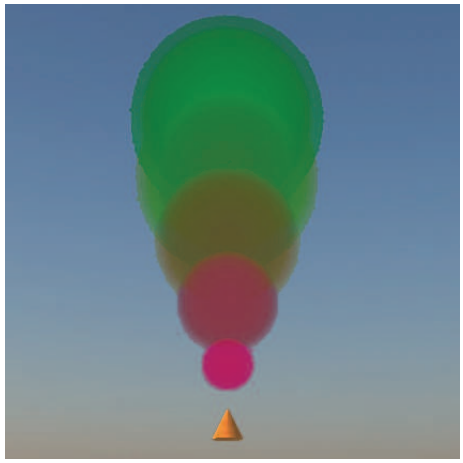


Imagen 6.11.20: Modelo de fuente cono angular sin definir el comienzo y fin del ángulo

Debemos introducir valores a los parámetros PSYS_SRC_ANGLE_BEGIN y PSYS_SRC_ANGLE_END para que este modo de fuente tenga sentido. Por ejemplo, en la imagen 6.11.21 se han utilizado los siguientes valores:

PSYS_SRC_ANGLE_BEGIN, 0.0,
PSYS_SRC_ANGLE_END, PI,

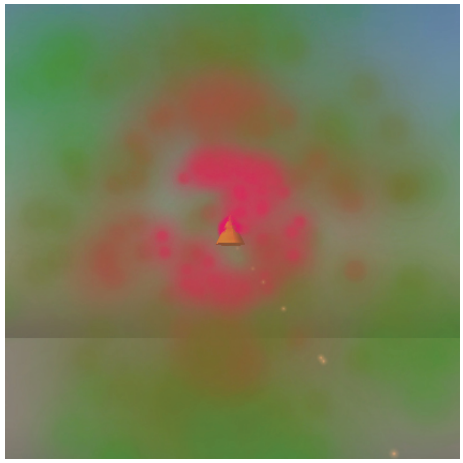


Imagen 6.11.21: Modelo de fuente cono angular modificando sus límites

En la imagen 6.11.22. se han cambiado los valores por PSYS_SRC_ANGLE_BEGIN, PI/2 y PSYS_SRC_ANGLE_END, PI.

Imagen 6.11.22: Vista frontal y lateral de fuente tipo cono angular modificada

Se puede observar cómo al ser una distribución tridimensional, no podemos apreciar grandes cambios entre la vista frontal y lateral del tipo cono angular, a diferencia del modelo angular cuya distribución es bidimensional y por lo tanto el ángulo en que se visualiza el sistema es muy importante.

OBJETIVO IDENTIFICADOR, PSYS_SRC_TARGET_KEY: admite un key. Especifica la clave (key) de un objeto objetivo, prim o avatar hacia las que se dirigen las partículas. Llegan hasta el centro del objetivo al final de su tiempo de vida. Requiere que el flag PSYS_PART_TARGET_POS_MASK este puesto. En el ejemplo que sigue las partículas salen despedidas y vuelve al emisor, haciendo un viaje de ida y vuelta.

```
…
llParticleSystem([
        PSYS_PART_FLAGS, PSYS_PART_TARGET_POS_MASK
        PSYS_SRC_TARGET_KEY, llGetKey(),
        PSYS_PART_MAX_AGE, 10.0,
```

```
        PSYS_SRC_BURST_PART_COUNT, 10
        PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_EXPLODE
        ]);
    }
}
```

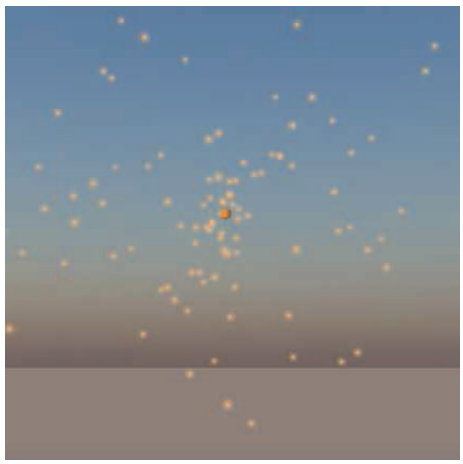


Imagen 6.11.23: Modo Target Key

RADIO DE EXPLOSIÓN, PSYS_SRC_BURST_RADIUS: admite un float. Especifica la distancia desde el emisor cuando las partículas se crean. Este valor se ignora cuando tenemos el flag PSYS_PART_FOLLOW_SRC_MASK puesto. Su valor máximo es 50.0. Por ejemplo la siguiente línea hace que las partículas se creen a 2 metros del centro del emisor, la diferencia se puede apreciar en la imagen siguiente.

PSYS_SRC_BURST_RADIUS, 2.0

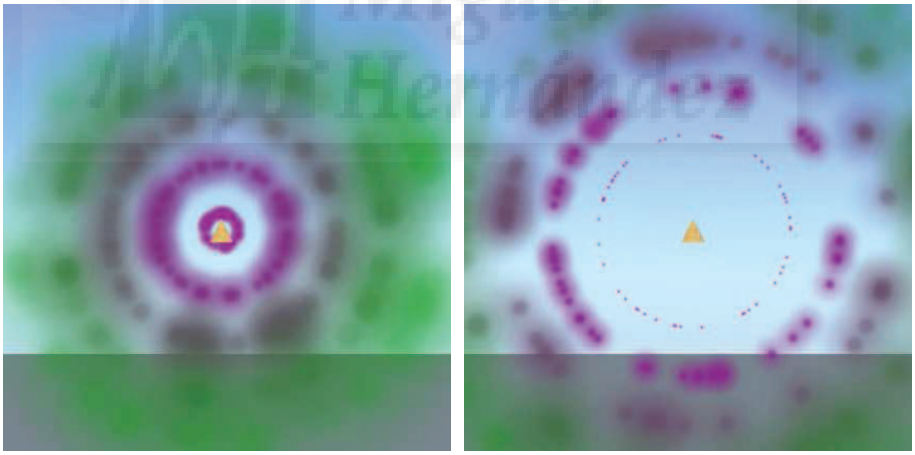


Imagen 6.11.24: Valores de 0 y 2 metros para PSYS_SRC_BURST_RADIUS

Caso práctico 11.1: Crear un sistema de partículas desde el inicio: Fuego

Objetivo: Crear un sistema de partículas desde el inicio y por completo que simule un fuego como ejemplo de los procedimientos a seguir en otras implementaciones de este tipo.

Tiempo de realización: 2 horas.

Pasos a realizar:

1. Crear el ambiente y objeto emisor.
2. Programación del sistema de partículas básico.
3. Programación de la física de las partículas y los flags.
4. Programación de la física de la fuente de partículas.
5. Cambios en el sistema.

1. Crear el ambiente y objeto emisor.

Vamos a crear un sistema de partículas y puede ser que no lo observemos de manera correcta por que el ambiente cambia dinámicamente. Por ello, trabajaremos con una luz controlada y un fondo de color neutro.

1.1. Para que la luz no cambie durante la realización del ejercicio pulsamos en el menú: Word > Environment settings > Midday. Esto hará que veamos todo con la luz del mediodía.

1.2. Por otra parte crearemos un rincón con 3 cajas cuyas dimensiones son 10x10x0.01 metros y con color puesto a un gris con RGB= 125, 125, 125. Este color hace que podamos ver mejor los colores propios del fuego que serán anaranjados.

1.3. Por último, creamos un objeto para que haga de emisor. Hemos elegido un cono que ha sido girado180º en el eje Y, aunque bien podía haber sido cualquier otro modelo.

El resultado de estas operaciones se observa en la siguiente imagen.

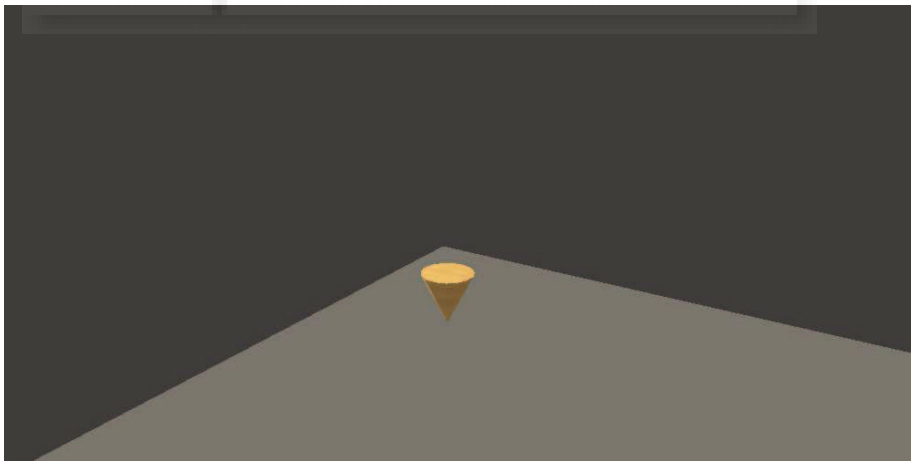


Imagen 6.11.cp.1.1: Fondo neutro y emisor de partículas para la práctica

2. Programación del sistema de partículas básico.

Vamos a escribir un script que le asignaremos al emisor, o sea, al cono. Para ello realizamos los siguientes pasos:

2.1. Botón derecho sobre el cono.

2.2. En el menú que aparece elegimos la opción Edit > Content.

2.3. Pulsamos sobre el botón “New Script”. Esto hace que el sistema cree un script por defecto que aparece debajo con el nombre “new script”.

2.4. Hacemos doble clic sobre el icono del script. Aparece un script ya programado que no nos vale, por lo tanto, lo borramos.

2.5. Escribimos el script que tenemos debajo. Pulsamos sobre la casilla “running” y sobre el botón “OK”. El resultado se debe parecer al que muestra la imagen que sigue.

```
default
{
   state_entry()
   {
         llParticleSystem ([
                  PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_EXPLODE
         ]);
   }
}
```

Nota: Es importante que al escribir el script tengamos activada la opción “running” para ir viendo los cambios interactivamente según escribimos el código de programación.

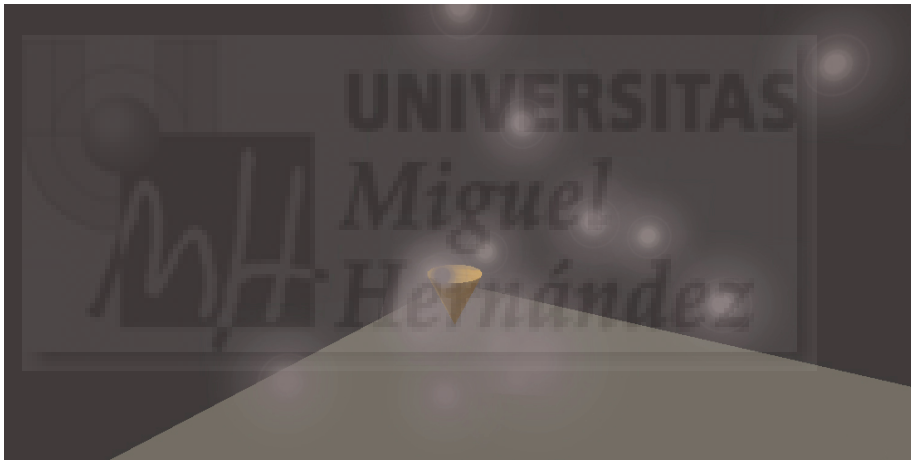


Imagen 6.11.cp.1.2: Sistema de partículas mínimo

3. Programación de la física de las partículas y los flags.

Lo primero que vamos a modificar es el tipo de fuente de explosión a una forma angular para tener más control sobre el camino que recorren las partículas con la sentencia:

PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_ANGLE

Luego, cambiaremos el color inicial y final de las partículas con las sentencias:

PSYS_PART_START_COLOR, <1.0, 0.5, 0.0>, // color inicial: naranja

PSYS_PART_END_COLOR, <1.0, 1.0, 0.0> // color inicial: amarillo

El script quedará como sigue y el sistema como muestra la siguiente imagen:

```
default
{
   state_entry()
```

```
    {
        llParticleSystem ([
        PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_ANGLE,  // modo angular
        PSYS_PART_START_COLOR, <1.0, 0.5, 0.0>,// color inicial: naranja
        PSYS_PART_END_COLOR, <1.0, 1.0, 0.0>      // color inicial: amarillo
        ]);
    }
}
```

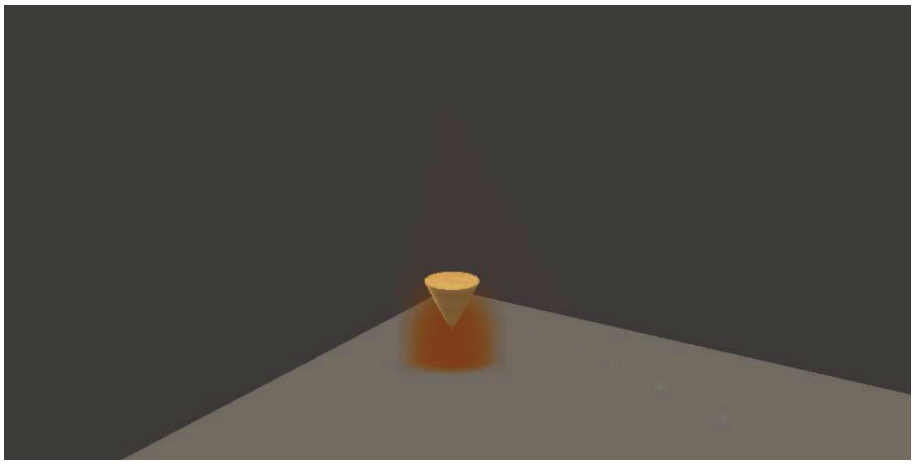


Imagen 6.11.cp.1.3: Sistema en modo angular y con color de partículas adecuado

Otra evolución de nuestro script que sin embargo no se reflejará de momento en la visualización del sistema es la incorporación de los flags emissive_mask y interp_color_mask. Estos flags funcionan como casillas de verificación, es decir, si están escritos, se ponen en funcionamiento. Emissive_mask sirve para que las partículas sean más brillantes. Interp_color_mask hace que las partículas pasen del color inicial al final de forma suave, lo que hace es interpolar de un color a otro suavemente. Estas flags se incluyen al principio de la función llParticleSystem como se observa en el script.

```
default
{
    state_entry()
    {
        llParticleSystem ([
        PSYS_PART_FLAGS, // vamos a controlar el comportamiento de las partículas
        | PSYS_PART_EMISSIVE_MASK // para que emitan un aro de luz
        | PSYS_PART_INTERP_COLOR_MASK,  //  interpolación de color de start a end
        PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_ANGLE,  // modo angular
        PSYS_PART_START_COLOR, <1.0, 0.5, 0.0>,// color inicial: naranja
        PSYS_PART_END_COLOR, <1.0, 1.0, 0.0>      // color inicial: amarillo
        ]);
    }
}
```

El sistema parece igual, aunque no lo sea, como se observa en la siguiente imagen.

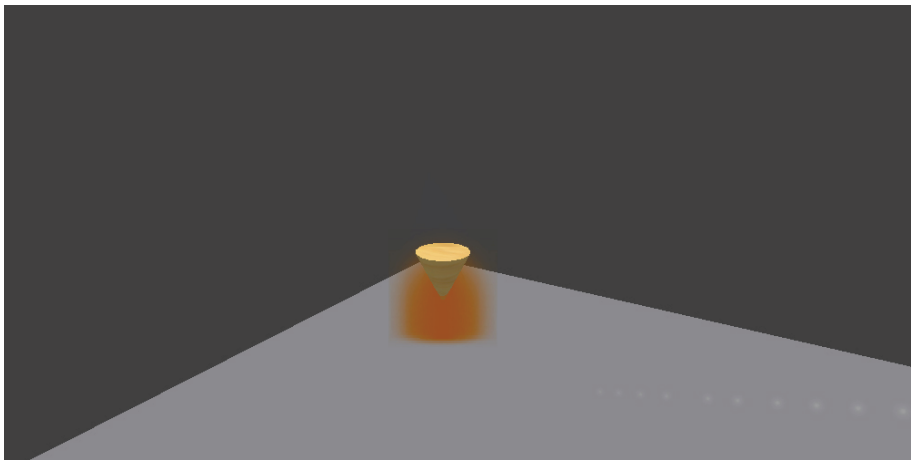


Imagen 6.11.cp.1.4: Introducción de los primeros flags

Ahora lo que hacemos es introducir sentencias que modifican la física de las partículas tales como “start_scale” y “end_scale” que cambian el tamaño de estas.

También introducimos la sentencia max_age que controla hasta dónde llegará el sistema, en nuestro caso, hasta dónde llegará el fuego, o sea la altura que tendrá.

El script quedará como sigue:

```
default
{
   state_entry()
   {
         llParticleSystem ([
         PSYS_PART_FLAGS, // vamos a controlar el comportamiento de las partículas
         | PSYS_PART_EMISSIVE_MASK // para que emitan un aro de luz
         | PSYS_PART_INTERP_COLOR_MASK,  //  interpolación de color de start a end

         PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_ANGLE,  // modo angular

         PSYS_PART_START_COLOR, <1.0, 0.5, 0.0>,// color inicial: naranja
         PSYS_PART_END_COLOR, <1.0, 1.0, 0.0>,       // color inicial: Amarillo
         PSYS_PART_START_SCALE,<0.6000, 0.8000, 0.6000>, // tamaño inicial de las part.
         PSYS_PART_END_SCALE,<0.2000, 0.4000, 0.3000>,     // tamaño final de las part.
         PSYS_PART_MAX_AGE, 3.0,  // tamaño de la llama

         ]);
   }
}
```

El sistema se visualizará como muestra la imagen que sigue.

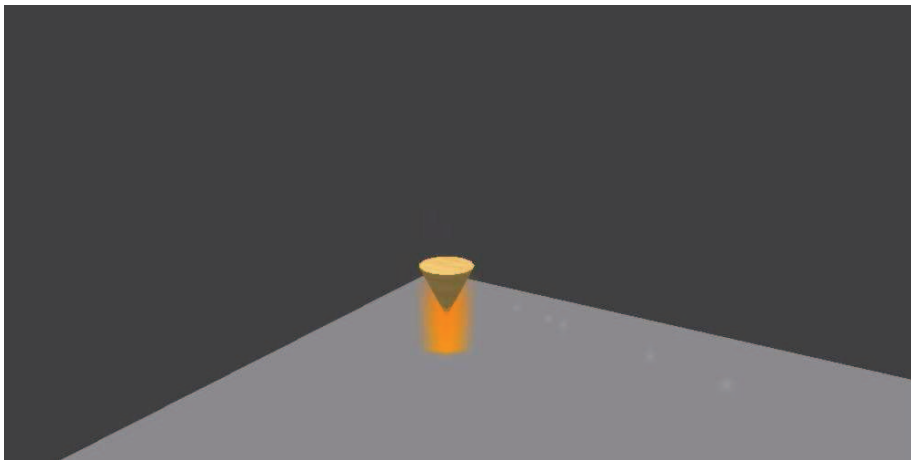


Imagen 6.11.cp.1.5: Física básica de las partículas

Lo último que haremos en este paso es terminar de programar los flags y las partículas. Para ello introducimos los flags interp_scale_mask que interpola la escala de las partículas, de tal forma que van empequeñeciendo poco a poco. También hacemos que se dirijan hacia arriba con follow_velocity_mask y con Bounce_mask. Incluimos wind_mask que hace que la llama fluctúe en función del tiempo. Esto hace que tengo mayor realismo.

Aparte de estos cambios, modificamos la transparencia de las partículas de nada (totalmente opacas) al principio hasta casi transparentes al final, para generar el efecto de que las llamas se "deshacen".

```
default
{
   state_entry()
   {
         llParticleSystem ([
         PSYS_PART_FLAGS, // vamos a controlar el comportamiento de las partículas
         | PSYS_PART_EMISSIVE_MASK // para que emitan un aro de luz
         | PSYS_PART_INTERP_COLOR_MASK,  //  interpolación de color de start a end
         | PSYS_PART_INTERP_SCALE_MASK   //  interpolación de tamaño de start a end
         | PSYS_PART_FOLLOW_VELOCITY_MASK // velocidad de rotación del eje vertical
         | PSYS_PART_WIND_MASK// velocidad de las partículas debido al viento
         | PSYS_PART_BOUNCE_MASK, // las partículas solo para el eje positivo

         PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_ANGLE,  // modo angular

         PSYS_PART_START_COLOR, <1.0, 0.5, 0.0>,// color inicial: naranja
         PSYS_PART_END_COLOR, <1.0, 1.0, 0.0>,       // color inicial: Amarillo
         PSYS_PART_START_SCALE,<0.6000, 0.8000, 0.6000>, // tamaño inicial de las part.
         PSYS_PART_END_SCALE,<0.2000, 0.4000, 0.3000>,     // tamaño final de las part.
         PSYS_PART_START_ALPHA,1.000000,  // no hace falta, es su valor por defecto
         PSYS_PART_END_ALPHA,0.100000       // si no está no es realista, no se deshace
         PSYS_PART_MAX_AGE, 3.0,  // tamaño de la llama
         ]);
   }
}
```

En la siguiente imagen se puede observar como queda el sistema con las máscaras de viento y dirección adecuados.

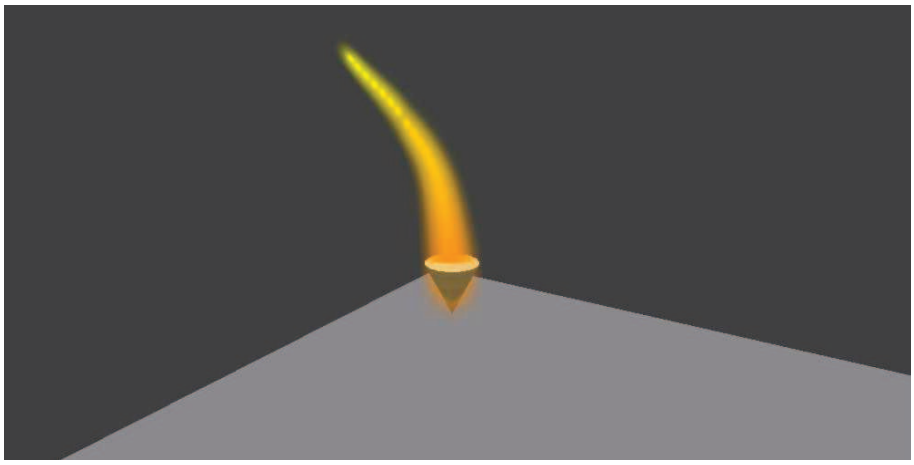


Imagen 6.11.cp.1.6: Máscaras wind y bounce

En la siguiente imagen se aprecia el cambio cuando se programan la máscara y física de la transparencia de las partículas.

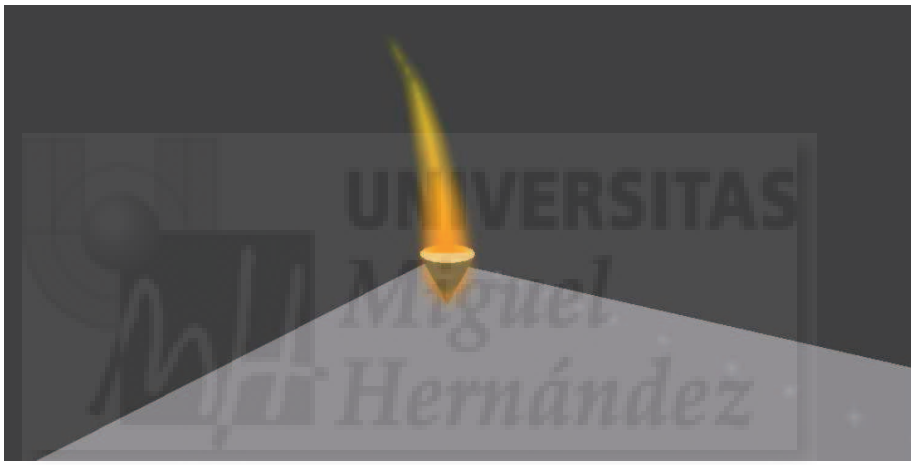


Imagen 6.11.cp.1.7: Máscara y física de partículas alpha

4. Programación de la física de la fuente de partículas.

En este punto vamos a tratar la fuente de partículas con sentencias como burst_rate que controla el intervalo de expulsión de partículas y burst_part_count, donde le decimos cuantas partículas salen en cada expulsión. Por lo tanto, la "virulencia" del fuego vendrá data por estos dos factores: cuantas "llamas" aparecen y en cuanto tiempo. Una característica sin mucha importancia en este caso es burst_radius, que permite especificar la distancia desde el centro del emisor desde el que se emiten las partículas. En este caso no se modifica apenas.

El script queda como sigue:

```
default
{
   state_entry()
   {
      llParticleSystem([
          // modo de trabajo mediante flags
          PSYS_PART_FLAGS,
          | PSYS_PART_EMISSIVE_MASK // para que emitan un aro de luz
          | PSYS_PART_INTERP_COLOR_MASK  //  interpolación de color de start a end
          | PSYS_PART_INTERP_SCALE_MASK   //  interpolación de tamaño de start a end
```

```
        | PSYS_PART_FOLLOW_VELOCITY_MASK // velocidad de rotación del eje vertical
        | PSYS_PART_WIND_MASK// velocidad de las partículas debido al viento
        | PSYS_PART_BOUNCE_MASK, // las partículas solo para el eje positivo

        // modo de fuente
        PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_ANGLE,  // modo angular

        // física de las partículas
        PSYS_PART_START_COLOR, <1.0, 0.5, 0.0>,// color inicial: naranja
        PSYS_PART_END_COLOR, <1.0, 1.0, 0.0>,       // color inicial: amarillo
        PSYS_PART_START_SCALE,<0.6000, 0.8000, 0.6000>, // tamaño inicial de las part.
        PSYS_PART_END_SCALE,<0.2000, 0.4000, 0.3000>,     // tamaño final de las part.
        PSYS_PART_START_ALPHA,1.000000,  // no hace falta, es su valor por defecto
        PSYS_PART_END_ALPHA,0.100000       // si no está no es realista, no se deshace
        PSYS_PART_MAX_AGE, 3.0,  // tamaño de la llama, alarga la llama

        // física de la fuente
        PSYS_SRC_BURST_RATE,0.020000,       // intervalo entre apariciones de partículas
        PSYS_SRC_BURST_PART_COUNT,10,   // cantidad de partículas por aparición
        PSYS_SRC_BURST_RADIUS,0.001000,   // distancia de aparición al centro emisor
        ]);
    }
}
```

El sistema resultante es el siguiente:

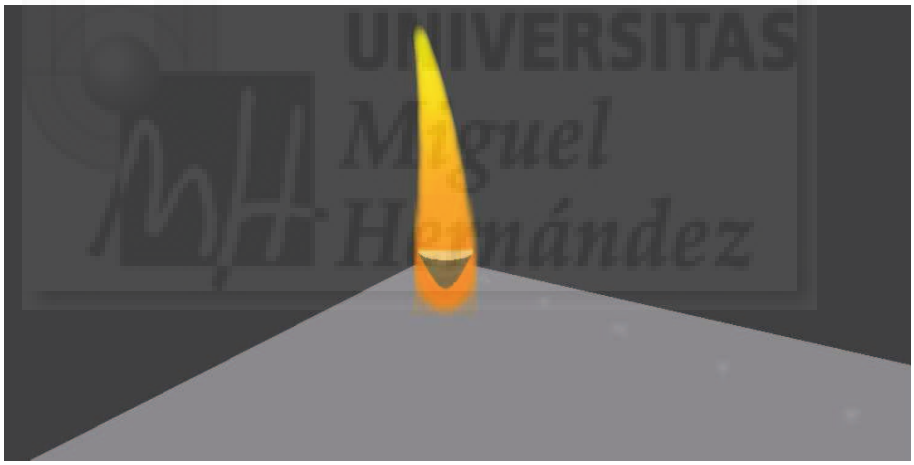


Imagen 6.11.cp.1.8: Introducción de la física para el emisor

Por último, modificamos la velocidad mínima y máxima del sistema. Estos dos parámetros modifican drásticamente el sistema tal como se aprecia en la imagen 6.11.cp.1.9.

```
default
{
    state_entry()
    {
        llParticleSystem([

        // modo de trabajo mediante flags

        PSYS_PART_FLAGS,
        | PSYS_PART_EMISSIVE_MASK // para que emitan un aro de luz
        | PSYS_PART_INTERP_COLOR_MASK  //  interpolación de color de start a end
        | PSYS_PART_INTERP_SCALE_MASK   //  interpolación de tamaño de start a end
        | PSYS_PART_FOLLOW_VELOCITY_MASK // velocidad de rotación del eje vertical
        | PSYS_PART_WIND_MASK// velocidad de las partículas debido al viento
```

```
        | PSYS_PART_BOUNCE_MASK, // las partículas solo para el eje positivo

        // modo de fuente

        PSYS_SRC_PATTERN, PSYS_SRC_PATTERN_ANGLE,  // modo angular

        // física de las partículas

        PSYS_PART_START_COLOR, <1.0, 0.5, 0.0>,// color inicial: naranja
        PSYS_PART_END_COLOR, <1.0, 1.0, 0.0>,       // color inicial: amarillo
        PSYS_PART_START_SCALE,<0.6000, 0.8000, 0.6000>, // tamaño inicial de las part.
        PSYS_PART_END_SCALE,<0.2000, 0.4000, 0.3000>,     // tamaño final de las part.
        PSYS_PART_START_ALPHA,1.000000,  // no hace falta, es su valor por defecto
        PSYS_PART_END_ALPHA,0.100000       // si no está no es realista, no se deshace
        PSYS_PART_MAX_AGE, 3.0,  // tamaño de la llama, alarga la llama

        // física de la fuente

        PSYS_SRC_BURST_RATE,0.020000,      // intervalo entre apariciones de partículas
        PSYS_SRC_BURST_PART_COUNT,10,   // cantidad de partículas por aparición
        PSYS_SRC_BURST_RADIUS,0.001000,   // distancia de aparición al centro emisor
        PSYS_SRC_BURST_SPEED_MIN,0.100000,   // velocidad mínima
        PSYS_SRC_BURST_SPEED_MAX,0.500000,  // velocidad máxima
        // si le ponemos el doble, la llama se hace más alta y el fuego más potente
        ]);
    }
}
```

Imagen 6.11.cp.1.9: Terminación del fuego

5. Cambios en el sistema.

En la imagen de abajo se puede apreciar cómo varía el fuego al cambiar el valor de la sentencia max_age de 3.0 a 6.0. Es decir, si cambiamos la edad que deben permanecer las partículas en Second Life, esto hace que se extiendan más, y por lo tanto que aumente el tamaño de todo el sistema.

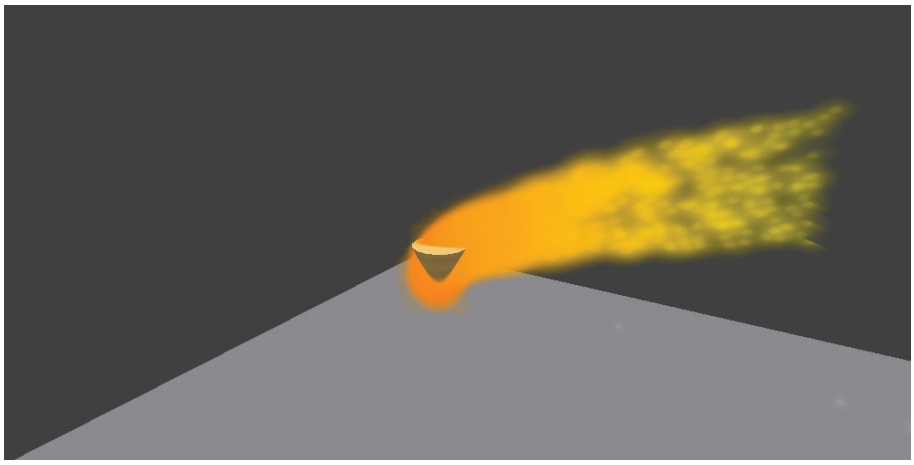


Imagen 6.11.cp.1.10: Aumento de la edad de las partículas

Ahora lo que hacemos es cambiar el modo angular por el de explosión y el resultado es el que se aprecia en la imagen siguiente.

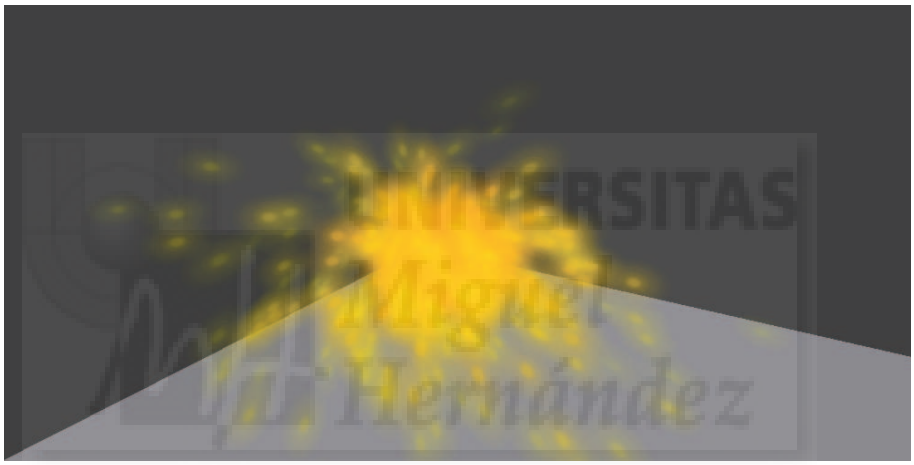


Imagen 6.11.cp.1.11: Cambio del modo angle a modo explode

Conclusiones:

Para entender bien el funcionamiento del sistema de partículas conviene realizar cambios y ver cómo estos repercuten en su funcionamiento, por lo que es conveniente ir modificando los parámetros para llegar poco a poco a entenderlos bien.

El planteamiento del script de programación se debe realizar teniendo en cuenta que hay tres grupos de sentencias: los flags que controlan las interpolaciones y los modos predeterminados, los que influyen en la dinámica y la forma de las partículas y los que lo hacen en el modo de expulsión de las partículas desde el emisor.

Unidad 12: Integración multimedia.

Introducción teórica:

1. La multimedia en Second Life.
2. Textos.
3. Imágenes.
4. Animaciones.
5. Sonidos.
6. Vídeos. Reproducción por streaming.
7. Otros medios multimedia en Second Life.

1. La multimedia en Second Life.

En Second life se puede observar multimedia todo el tiempo a nuestro alrededor. La propia definición de multimedia, es decir, uso de muchos medios, es una de las características de todo metaverso ya que si se quiere recrear la vida real, no hay más remedio que ofrecer funciones para convertir la experiencia en multimedia.

Por ejemplo, si estamos en un determinado terreno o en nuestra propia casa, puede que queramos ver un vídeo o escuchar un hilo musical u oír la radio.

Otro ejemplo puede darse al querer impartir una clase de biología. Supongamos que tenemos que usar sonidos de animales, junto con unas animaciones de sus movimientos, fotos y vídeos de su vida en libertad y todo perfectamente documentado por texto. Todo esto formaría un conjunto típicamente multimedia.

En el caso que nos afecta a nosotros, podemos crear una escultura en movimiento que use vídeos y sonidos integrados en la misma escultura y que esta fuera interactiva.

Estas tres experiencias pueden ser realizadas en Second Life con mayor facilidad que en otros sistemas debido a sus características inherentes por su condición de metaverso.

2. Textos.

Imagen 6.12.1: Texturas con texto

El uso de textos en Second Life tiene dos vertientes. Al ser un sistema 3D en sí mismo, cuando visualizamos textos (por ejemplo de ayuda), en realidad son imágenes que texturizan un panel de ayuda, un letrero o pizarra creados con objetos 3D y por lo tanto no los podemos clasificar como textos. Digamos que tienen forma de texto pero no lo son. Este es el caso del ejemplo que se puede observar en la imagen precedente.

Por otro lado, Second Life hace uso de texto real constantemente, ya que el propio chat o el editor de scripts se utilizan escribiendo directamente y podemos utilizar las funciones de copiar y pegar desde y a otros programas de texto como Word, etc.

Para nuestro caso, si queremos incorporar texto en obras artísticas, la solución más sencilla es crear el texto, convertirlo en imagen y luego ponerla como superficie de un objeto. Además

como podemos “envolver” con texturas cualquier tipo de objeto, podríamos aplicar imágenes con texto sobre superficies no planas para crear un efecto muy plástico.

Quizá el mejor uso de texto propiamente dicho sea el uso de las Note Cards que podemos encontrar al tocar un objeto en algunos Sandbox y otros sitios divulgativos. Las Note Cards tienen por objeto el que los habitantes de Second Life puedan compartir textos, que suelen ser de ayuda, aclaraciones, etc. Las Note Cards, son ofrecidas por otros avatares o por objetos y si son aceptadas, se almacenarán en el Inventario. En la siguiente imagen se puede apreciar cómo estamos estudiando un texto sobre programación.

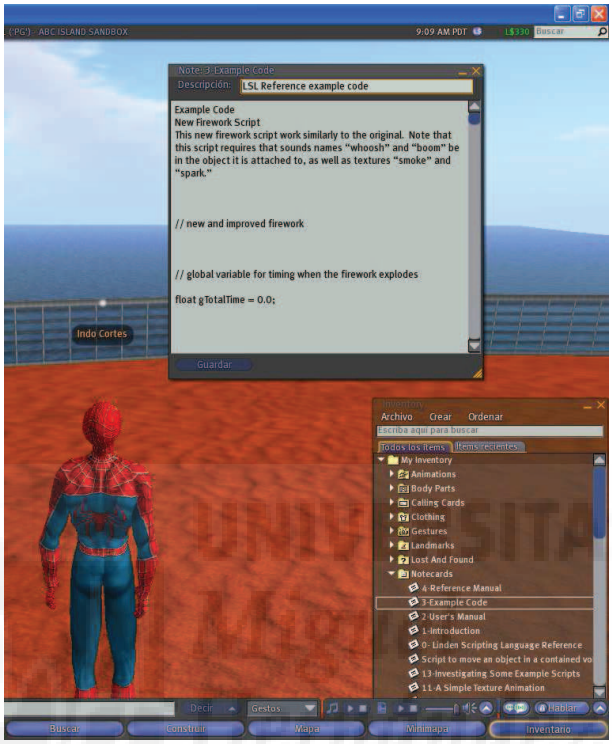


Imagen 6.12.2: Textos mediante Note Cards

Otra alternativa para mostrar textos es que estén en formato HTML. Sabemos que el texto puede tener distintos formatos, pero si tiene esta extensión sí lo podemos ver como tal mediante la creación de un reproductor de páginas web. Este tema lo veremos en el punto 7.

3. Imágenes.

Las imágenes en Second Life se visualizan por todos lados. Ya hemos visto cómo se pueden utilizar en general y a nosotros nos pueden venir bien para texturizar nuestras obras artísticas con distintos materiales como estudiamos en la unidad 4: superficies y materiales de los objetos.

Imagen 6.12.3: Comparación entre una captura de pantalla y una instantánea

Cómo ya vimos en la introducción a Second Life, las imágenes que nos rodean pueden ser tan plásticas que existe un comando para tomar una instantánea de la vista, es decir, una especie de fotografía que mostrará la vista pero sin ningún elemento de interface. Por ejemplo en la anterior imagen se puede observar la diferencia entre una captura de pantalla típica realizada pulsando la tecla ImpPnt y una instantánea realizada con Second Life.

Para realizar una instantánea ejecutaremos el comando del menú Archivo > Hacer una foto. Esta orden abrirá una caja de diálogo que nos permite guardar la imagen en nuestro disco duro local. El formato de la imagen es .PNG.

4. Animaciones.

Las animaciones que se utilizan son de tres tipos. La primera es a nivel del avatar y se crean con programas como ®Poser. El segundo tipo serían las animaciones construidas con una serie de imágenes y por último las animaciones de objetos. Para este documento, el primer tipo de animaciones carece de interés ya que crean gestos y otros efectos que no tienen, en principio, nada que ver con una obra artística, si exceptuamos claro está, que queramos representar un baile o una obra teatral.

Pero para crear obras tridimensionales de tipo escultura, nos interesan el segundo y el tercer tipo. Las animaciones realizadas mediante imágenes las tratamos especialmente en el tema de las texturas animadas que se suelen utilizar frecuentemente para crear efectos.

El movimiento de objetos lo podemos utilizar para crear esculturas total o parcialmente móviles. En la imagen que sigue podemos ver una de muchas esculturas cinéticas que se encuentran en “Gallery of musical sculptures” de Alarbus, uno de tantos sitios de arte interesantes en Second Life.

Imagen 6.12.4: Escultura cinética

5. Sonidos.

Los sonidos pueden tener numerosas fuentes. Quizá el más “natural” es el que producimos nosotros mismos al hablar por el chat de voz. También podemos grabar sonidos digitalmente, luego modificarlos y subirlos a Second Life para utilizarlos en nuestros trabajos.

En el chat de voz tenemos varias utilidades para aumentar el realismo de nuestro avatar. Podemos usar el “Customize speech gestures” que son una serie de sonidos anexos a los gestos habituales y también podemos realizar “sincronización de labios”.

El chat de voz se inicia pulsando sobre el botón “Hablar” en la parte derecha de la pantalla, entonces se desplegará un pequeño menú para gestionarlo. De todas formas tan solo hace falta tener un micrófono y hablar para comprobar si nos oyen los avatares cercanos.

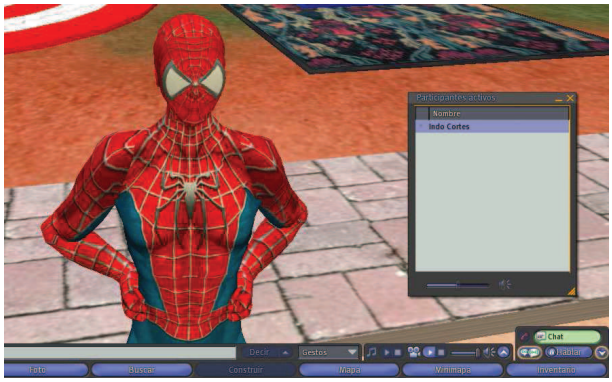


Imagen 6.12.5: Chat de voz

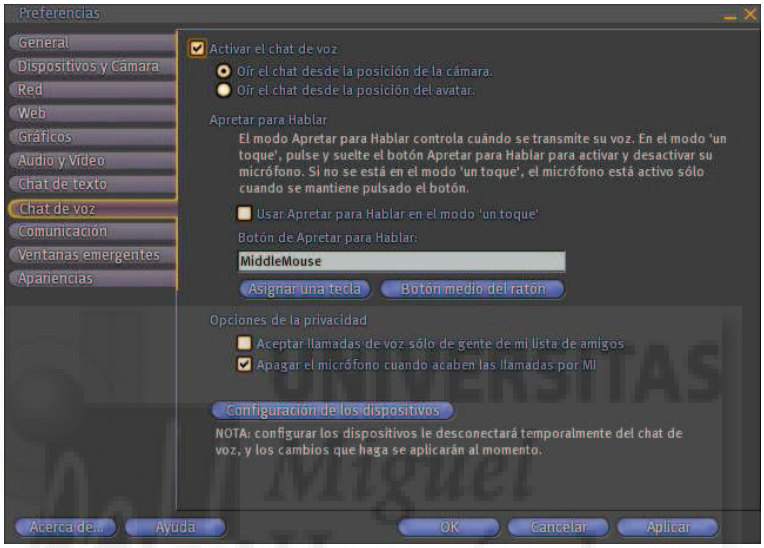


Imagen 6.12.6: Preferencias del Chat de voz

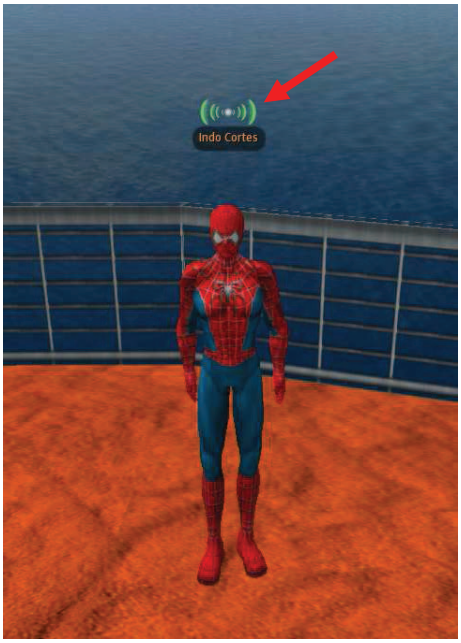


Imagen 6.12.7: Utilizando el Chat de voz

La “sincronización de labios” lo que pretende es que cuando hablemos, nuestros labios se muevan en función de las palabras pronunciadas. Esta es una característica muy avanzada pero que solo es útil en determinadas ocasiones y no es muy normal en otros metaversos. Para utilizar la sincronización de labios tenemos que ir al menú Advanced (Ctrl + Shift + D) > Character > Enable Lips Sync.

Imagen 6.12.8: Sincronización de labios

Para utilizar sonidos digitales en Second Life, basta con subirlos, para ello, hacemos Archivo > Subir > Sonido. Esta operación cuesta 10 L$ por sonido. El sonido pasará a nuestro Inventario a la carpeta de sonidos. El sonido lo podemos convertir en un gesto.

6. Vídeos. Reproducción de vídeo por streaming.

La música ambiental y los vídeos tienen en Second Life una característica común: no se pueden “subir”. De hecho, podemos subir sonidos (solo recomendable para cortos sonidos de efectos especiales o gestuales) pero no podemos subir vídeos ya que no existe ninguna opción en los menús.

Los vídeos se reproducen en Second Life mediante streaming, para ello debemos conocer su dirección en la red, es decir, dónde están almacenados en un servidor externo.

Se reproducen con Quicktime y por tanto se pueden reproducir archivos QuickTime (.mov), QuickTime streams en tiempo real (rtsp://), DVD (.mp4), QuickTime VR escena y objetos (.mov) y películas flash no interactivas (.swf)

Para ver lugares con vídeos basta con ira a la herramienta de búsqueda y en la pestaña lugares escribir algo como “video media”. Nosotros hemos usado este método para visualizar el vídeo que se observa en la siguiente imagen.

Imagen 6.12.9: Visualizando un vídeo

El mecanismo de reproducción funciona sustituyendo una textura que se aplica sobre un objeto por una película. Por lo tanto la pantalla máxima será el tamaño máximo de un Prim, es decir, 10 x 10 metros.

Para poder hacer este intercambio y emitir vídeo, el terreno lo debe permitir y si es así aparecerá un control para reproducir el vídeo en la parte inferior derecha. Por lo tanto para reproducir vídeo en un terreno necesitamos ser el propietario, ya que vamos a acceder a los opciones “media” de ese terreno.

Imagen 6.12.10: Controles para la reproducción de vídeo

El proceso empezaría por pulsar sobre el nombre de la parcela y sus coordenadas que indican dónde nos encontramos. Esta información se sitúa en la barra de menús tal como muestra la siguiente imagen.

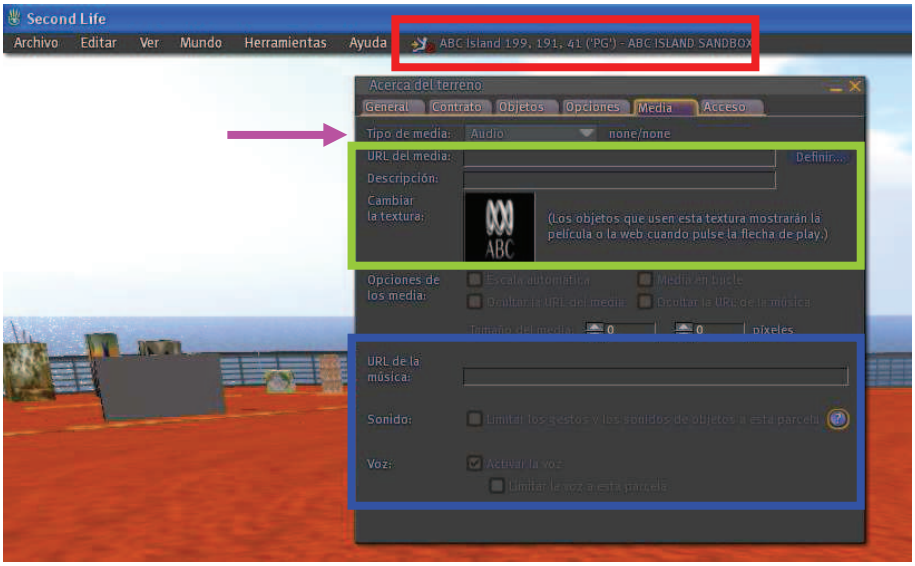


Imagen 6.12.11: Diálogo "Acerca del terreno", pestaña "Media"

Seguidamente pulsamos sobre la pestaña Media y en "Tipo de media" (en la imagen precedente se visualiza con una flecha fucsia) elegiremos vídeo. Vamos a "Cambiar la textura", y normalmente tendremos una X si no se ha usado ninguna anteriormente. Hacemos clic y aparecerá una caja de diálogo para escoger una textura que haya sido aplicada al prim que hará de soporte del vídeo. En el control dónde pone "URL del media" introducimos la dirección URL dónde se encuentra el vídeo y ya hemos terminado.

Al cerrar la caja de diálogo "About Land", nos debe aparecer el control de vídeo que hemos señalado en la imagen anterior. Y pulsando al botón Play comenzaremos a ver el vídeo. Puede ser que aparezca desplazado debido en realidad a un mal ajuste de la textura de intercambio, por lo que tendremos que editar el objeto y usar el botón "Align media texture".

Tenemos que advertir que cuando ejecutemos audio y vídeo en nuestra parcela puede ser a petición de un usuario o automáticamente. El ejecutar sonido o vídeo automáticamente tiene el problema que puede aumentar el lag muchísimo y por otra parte puede ser molesto para un invitado estar oyendo algo que no ha solicitado explícitamente. Esta característica se gestiona desde el menú Edición > Preferencias > Audio y vídeo. Esta caja de diálogo se puede observar en la siguiente imagen.

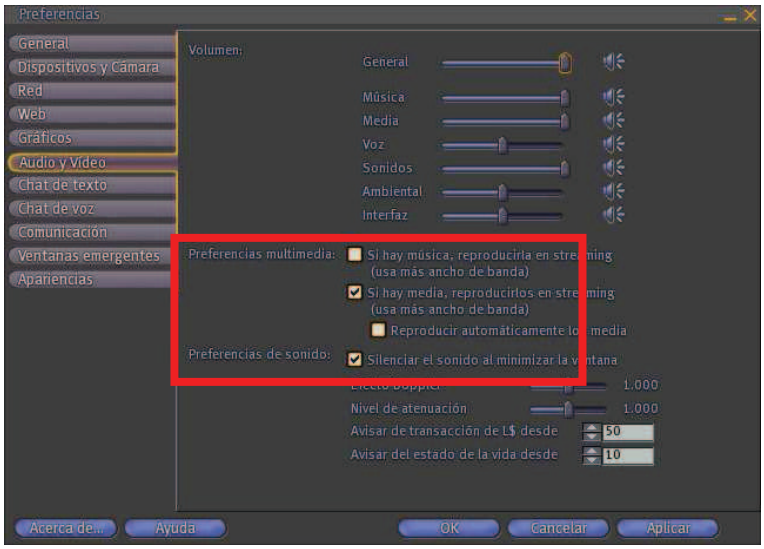


Imagen 6.12.12: Preferencias de audio y vídeo

8. Otros medios en Second Life.

- Páginas Html con un reproductor o con el navegador integrado.

Antes de nada, debemos comentar que Second Life contiene un navegador web integrado que se abre al pulsar F1 y cuya página inicial es la ayuda html de Second Life. Pero, no obstante, podemos visualizar páginas HTML mediante el reproductor creado para visualizar vídeos cómo hemos visto en el punto anterior. Solamente debemos poner el valor “Contenido web” en el control “Tipo de media” y la dirección completa de la página en el control “URL del media”. Esto hará que tengamos un control parecido al utilizado para reproducir vídeo pero con iconos distintos.

Al ser este un reproductor y no un navegador, podemos visualizar la página HTML pero no podemos navegar con ella, es decir, los enlaces no se pueden utilizar.

Si editamos el prim que contiene el reproductor de HTML, en la parte inferior de la pestaña General veremos el control “Al hacer clic” (asegurarme) y el último valor “Open parcel media” hace que al pulsar sobre la página, Second Life abra el navegador interno de Second Life con la página que nosotros tenemos puesta en el reproductor y en este caso ya será navegable por entero. También tenemos la inversa, es decir, el navegador tiene un botón para que aparezca la página que estamos viendo como la página que se muestra en nuestra parcela.

Imagen 6.12.13: Abrir los media de la parcela nos indica los objetos multimedia

Imagen 6.12.14: Navegador Web interno de Second Life

- Presentaciones de PowerPoint.

Las presentaciones tipo PowerPoint no se pueden realizar directamente y con todas sus funciones y materiales multimedia, sonidos y vídeos. Pero si tenemos una presentación de diapositivas típica en este sistema, podemos exportarlas desde el propio programa a imágenes .jpg. Tendremos entonces una imagen por diapositiva. La idea es exportarlas a Second Life y texturizar con ellas un prim que normalmente será una caja que haga las veces de pantalla.

Luego podemos crear otros prims que hagan las veces de botones para poder ir al menos hacia delante o hacia atrás en la presentación. Esto, por supuesto, requiere de la creación de scripts específicos para cada presentación, ya que deberán controlar, entre otras cosas, el número de diapositivas que componen la presentación, por ello, no los estudiaremos en este tema.

En el link http://www.youtube.com/watch?v=kZvRk1Fi0Lk se presenta una solución bastante adecuada en formato vídeo.

- Televisión vía satélite en directo.

La investigación en torno a Second Life es también una forma de crear integración multimedia. En este sentido, el EUVE (European Virtual Engineering Technological Center) ha creado un programa para introducir imágenes reales en directo en Second Life. Esto abre un campo que es el de la creación de televisiones a bajo precio ya que pueden tener escenarios virtuales a un coste mínimo.

Este programa utiliza una cámara de vídeo, un fondo de color verde para le cromakey, un ordenador conectado a Internet y un hardware llamado Ultimate que digitaliza automáticamente la televisión vía satélite. Este sistema también se puede usar para hacer exposiciones corporativas en directo.

Unidad 13: Publicaciones off line de trabajos realizados en Second Life.

Introducción teórica:

1. Presentación de trabajos en Second Life no conectados.
2. Toma de instantáneas.
3. Creación de vídeos.
4. SRURL.

1. Publicación de trabajos en Second Life no conectados.

Second Life es un metaverso y por tanto su propia naturaleza hace que solo se pueda experimentar cuando estamos conectados. El problema surge cuando usamos este software como plataforma para producir trabajos que queremos mostrar también de forma que estemos fuera de línea. La solución es evidente: tendremos que prescindir de características intrínsecas a un metaverso como son la inmersión, la interactividad y otras. Por supuesto, esto restará "realismo" a las presentaciones que podamos hacer de nuestras obras.

Es lógico pensar que si nuestros trabajos están enfocados a la interactividad, al movimiento, a la conjunción con el sonido, etc. no vamos a poder nunca suplir la visita directa a Second Life, pero si queremos exponer imágenes en un determinado ambiente o por ejemplo, esculturas que no sean cinéticas, podemos utilizar tomas de vídeos o capturas de imagen para al menos poder mostrar la estética de nuestro trabajo.

Por otra parte, a veces el objetivo no es experimentar directamente una obra sino por ejemplo, editar un catálogo y en este caso, si podemos hacer un buen trabajo. Si lo que queremos es crear una página web o un CD/DVD las posibilidades son mayores ya que podemos capturar vídeos o emplear enlaces directos a una localización determinada interna de Second Life.

Por tanto, no podemos hablar de publicación de trabajos realizados en Second Life sino de presentación de estos en otras formas.

2. Toma de instantáneas.

Este tema ya lo hemos abordado en otras unidades teóricas, sin ir más lejos, en la anterior, vimos cómo se hacen y las diferencias con las simples capturas de pantalla. Por lo tanto solo cabe recordar que las instantáneas se guardan en el disco local del usuario en formato .png y que no mostrarán ningún elemento de interfaz, solo el paisaje.

Imagen 6.13.1: Instantánea de una escultura cinética y musical

2. Creación de vídeos.

Los vídeos los debemos crear con un programa externo ya que Second Life, en las últimas versiones no nos ofrece esta utilidad.

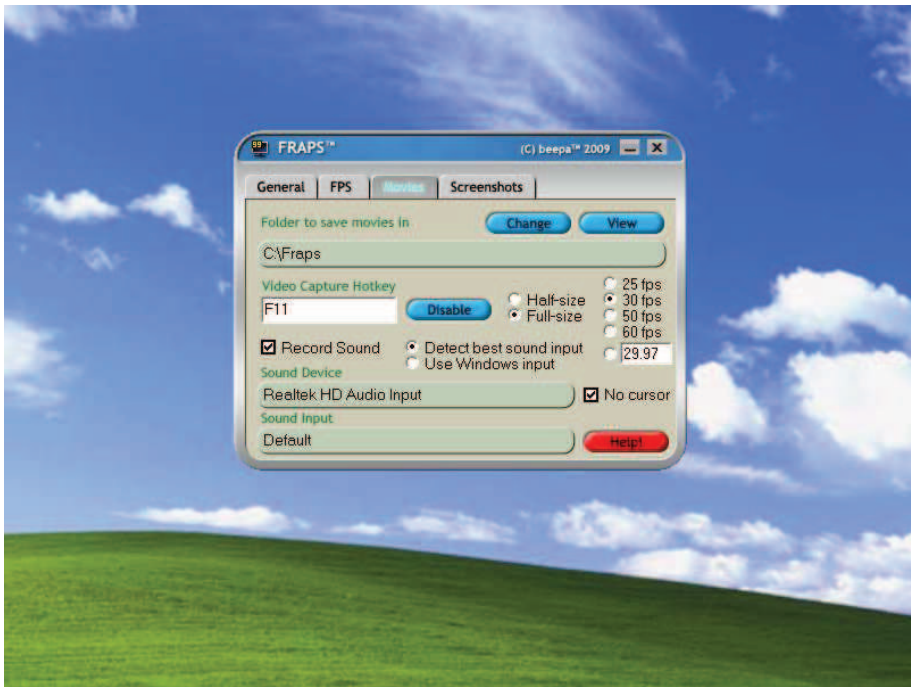


Imagen 6.13.2: Interface de Fraps

Uno de los softwares más utilizados para captura de vídeos, juegos, y programas 3D se llama Fraps® y es de la empresa Beepa®.

Fraps tiene muy buenas características para lo que buscamos, ya que permite la captura de pantallas y vídeo de alta calidad con o sin sonido y se ejecuta en segundo plano, por lo que no aparece en el resultado final. A esto debemos sumar su facilidad de manejo. Nos lo podemos bajar de multitud de páginas web y el instalable apenas ocupa 2,27 Mb. En la imagen anterior se puede observar la interface del programa.

Es un programa gratuito si no necesitamos las prestaciones más altas. Por ejemplo, las capturas de pantalla las realiza solo en formato .bmp. Si queremos poder guardar archivos en .jpg y otros nos debemos registrar y desembolsar 37 dólares USA.

En la siguiente imagen podemos observar el vídeo capturado con Fraps que muestra una visita a un museo de arte dentro de Second Life. El funcionamiento es tan sencillo como arrancar el programa y mientras estamos en Second Life pulsar una tecla determinada para comenzar a grabar el vídeo y volverla a pulsar para terminar de grabar.

Además esta tecla se puede elegir por parte del usuario en la pestaña “Movies”, en el control llamado “Video capture hockey” para que no interaccione con otras que podemos tener asignadas en el software a grabar. Para cambiar la tecla pulsaremos al botón azul llamado “Disable” y aparecerá el valor “None” para el control y seguidamente pulsamos la tecla deseada. Como se puede ver en la imagen nosotros elegimos la tecla F11.

Si no cambiamos más parámetros el vídeo se guardará en formato .avi en la carpeta c:\fraps, aunque esto también lo podemos elegir.

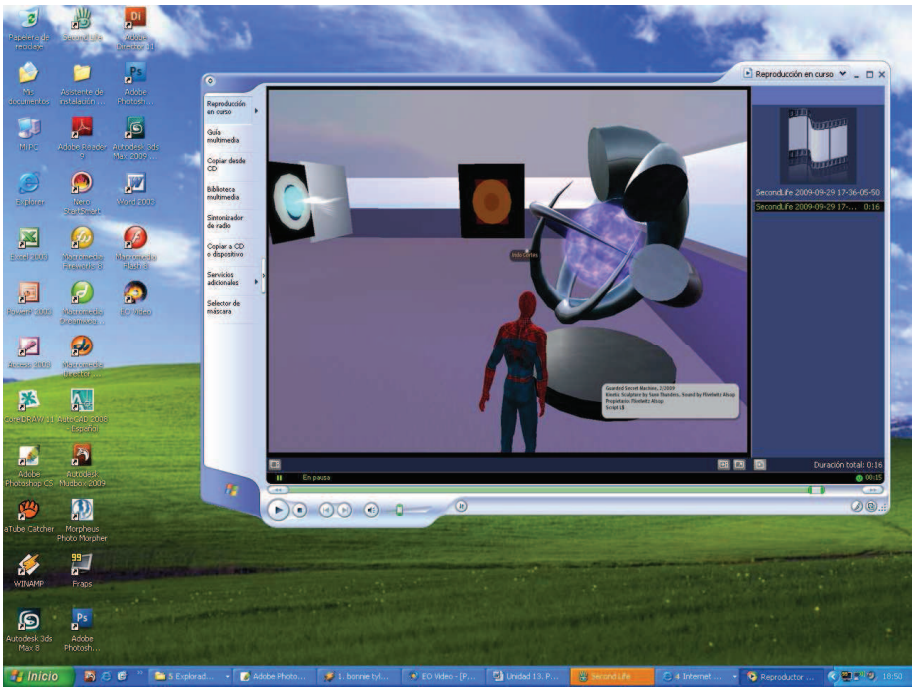


Imagen 6.13.3: Captura de vídeo que muestra experiencias en Second Life

3. SRURL.

Las SLurl son enlaces que permiten teletransporte directamente desde un navegador web a lugares en Second Life. Por supuesto, tenemos que tener instalado Second Life, pero permite unas referencias más ágiles entre los dos sistemas.

Los lugares en Second Life tienen una dirección que empieza por “secondlife://” y esto hace que cuando estamos en la web y ponemos un enlace “normal”, el navegador nos da un mensaje de error, de esta forma aunque no tengamos Second Life instalado nos llevará a la página de registro en este metaverso.

Para crear una SLurl tenemos que ir con nuestro habitual navegador web a la dirección: http://slurl.com/secondlife/<region>/<x-coordinate>/<y-coordinate>/<z-coordinate>/

En la siguiente imagen se puede observar la página web dónde podemos construir nuestra SLurl. La SLurl tomará forma como una página web independiente a la que podemos llamar mediante un enlace normal.

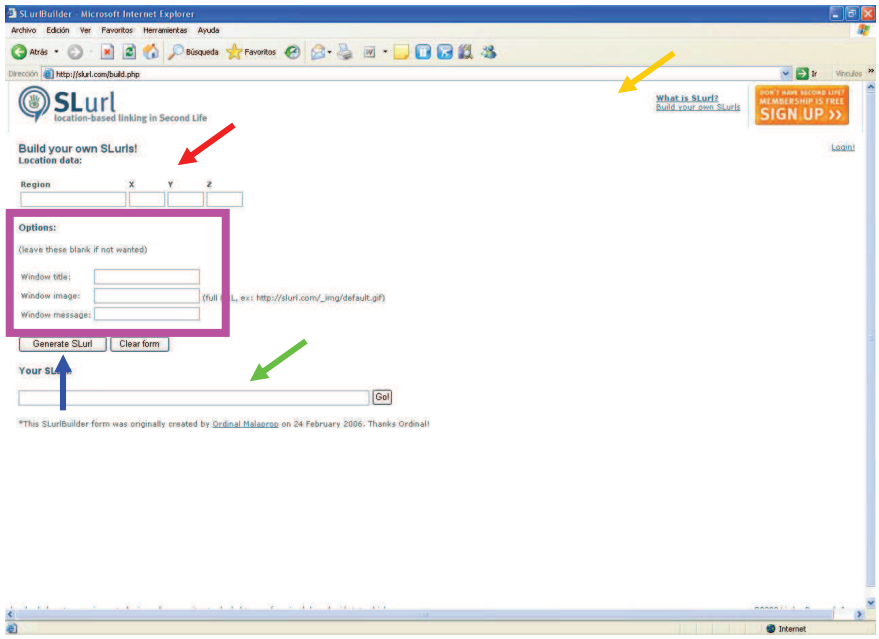


Imagen 6.13.4: Construcción de la SLurl

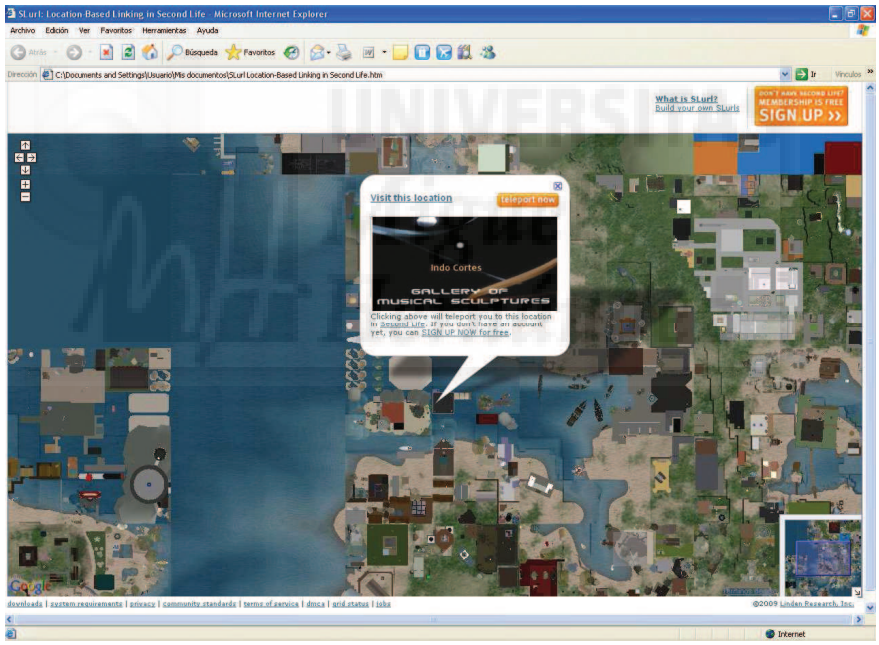


Imagen 6.13.5: SLurl

Con una flecha amarilla se señala el enlace pulsado que se llama “Build your own SLurl”. Con la flecha roja se señala las coordenadas del lugar que queremos direccionar que es una región y tres coordenadas X, Y, Z que son las que aparecen cuando estamos en Second Life arriba centradas en el interface. Con la flecha azul se señala el botón que tenemos que pulsar y con la flecha verde aparecerá una página web que contendrá el mapa y la SLurl.

Las opciones sirven para poder personalizar nuestro SLurl con un título de ventana, una imagen y un mensaje. Estas opciones se visualizan recuadradas en color fucsia.

El resultado de esta operación es una página web que muestra un mapa al estilo de Google maps tal como muestra la imagen anterior. En esta página encontraremos un botón para teletransportarnos al lugar indicado si lo deseamos. Si en ese momento no tenemos abierto el cliente de Second Life, esta página se encargará de abrirlo.

Esta página la podemos guardar en nuestro disco duro o publicarla en Internet. También la podemos publicar en otros medios no online como CD´s y/o DVD´s.

La conclusión que podemos sacar de este tema es que no podemos publicar en otros medios los trabajos realizados en Second Life tal cual y esto le resta versatilidad. Dicho de otro modo, para que un espectador o cliente visualice una obra artística necesita hacerlo con Second Life y por tanto tener instalado el cliente de Second Life, además de estar conectado. Estas dos restricciones son a nuestro modo de ver, la mayor desventaja de esta alternativa en comparación con la primera, donde usábamos Director para montar la obra.

Si se ve desde el otro punto de vista puede ser una ventaja. Es decir, si realizamos una obra sabiendo que solo será visualizada en Second Life, sabemos que las personas que existen tras los avatares están en general bastante preparadas en el uso de las nuevas tecnologías y que posiblemente tengan un alto nivel cultural. Por lo tanto, quizá no llegue al gran público pero sí a un público más preparado.

Conclusiones

En este apartado vamos a establecer de forma justificada una serie de conclusiones que nos permitan valorar el presente trabajo con el objetivo de evaluarlo. En nuestro extenso documento hemos tratado un gran número de conceptos y procedimientos que están relacionados entre sí y que se desarrollan buscando la consecución de unos objetivos muy concretos que pretendemos exponer de la manera más clara posible.

Esta tesis parte de un objetivo principal: la creación de un curso enfocado a unos alumnos con un perfil muy determinado para que adquieran una capacidad creativa nueva. Los alumnos a los que se dirige este trabajo son de nivel universitario, con amplios conocimientos de las artes y el lenguaje publicitario y con tendencias a expresar su creatividad en un marco tecnológico. La capacidad creativa que vamos a tratar es el desarrollo de aplicaciones multimedia interactivas para su ejecución directamente desde la red y desde memorias secundarias de ordenador.

La primera y principal conclusión es que la tesis se ha probado en la práctica: el objetivo inicial que se expone en el capítulo primero, se ha conseguido, es decir, hemos generado un curso que cumple las especificaciones que pretendíamos.

En nuestra opinión, el tema planteado tiene suficiente interés por sus implicaciones tanto en el ámbito académico como en el plano profesional, ya que las aplicaciones del tipo que desarrollamos en este documento son muy solicitadas en el contexto laboral actual.

Queremos poner de manifiesto que abordamos un tema novedoso. En este sentido, no hemos encontrado tesis en español que respondan a las cuestiones planteadas por este trabajo, tal como se puede deducir del capítulo 2.

Aunque lo desarrollamos más adelante, queremos adelantar que al ser el objetivo práctico de esta tesis la realización de un curso, se ha hecho hincapié en la utilización de una metodología sencilla y eficaz para estudiar de forma sistemática los objetivos planteados, es decir, que se adecua al nivel de los alumnos y a los recursos, que en general, disponemos actualmente en nuestras universidades.

Así mismo, queremos destacar que en las prácticas realizas, en todo momento se ha tenido en cuenta las características que más pueden aprovechar los estudiantes de arte, aunque en muchas ocasiones no se ha podido evitar cierta dificultad técnica.

A continuación vamos a exponer las conclusiones agrupándolas en los distintos aspectos tratados, para intentar exponerlas adecuadamente.

Conclusiones referentes al tipo de alumno al que va dirigido este trabajo.

En el curso desarrollado, hemos buscado siempre hacer hincapié en las características que puedan aumentar las posibilidades creativas de los trabajos realizados en detrimento de las cuestiones más tecnológicas. Esto hace que constantemente hayamos estado revisando la dificultad técnica, y aportando, siempre que se pueda, soluciones lo más sencillas posibles.

No siempre es posible sortear estos escollos, ya que sobre todo en la creación de la interacción, los conocimientos técnicos son directamente proporcionales al control que obtenemos de nuestras creaciones, y esto se refleja en la calidad final del trabajo. Por ello, hemos tratado de buscar un equilibrio entre los objetivos a alcanzar y la dificultad técnica que requiere. Esto se plasma en que hemos buscado el diseño de prácticas factibles y realizables en tiempo y utilidad práctica.

Por tanto, una conclusión que podemos extraer es que el curso creado se adapta en lo posible al tipo de alumno al que va dirigido teniendo en cuenta que los conocimiento previos de esta materia pueden llegar a ser prácticamente inexistentes.

Conclusiones referentes a las tecnologías utilizadas.

Como hemos comentado antes, los alumnos plasmarán sus ideas por medio de herramientas software. En este trabajo presentamos un estudio de algunas de las aplicaciones digitales que podemos utilizar y que deben cumplir una serie de condiciones. Estos requisitos vienen determinados por el tipo de alumno al que va enfocado esta tesis y por el tipo de trabajo creativo que queremos realizar. En estos trabajos utilizaremos elementos de distinto tipo multimedia y deben integrarse completamente. Podemos tener elementos de imagen en dos dimensiones, otros objetos concebidos desde su inicio en tres dimensiones, así como con archivos de sonido y vídeo. Y todo ello, además podrá seguir una sincronización y por tanto permitir la animación. Todos estos elementos, deben poder ser manipulados en tiempo real por el espectador del trabajo, es decir, deben ser interactivos.

El software utilizado debe posibilitar una o varias soluciones de presentación o "exhibición" de los trabajos realizados, siendo conveniente que se pueda ejecutar directamente desde Internet mediante un navegador como Explorer, Firefox… y además, también se pueda alojar en la memoria secundaria de un ordenador personal como puede ser un disco duro, una memoria flash, o un DVD y en un sistema operativo tan común como Windows XP y posteriores. Finalmente, nos decidimos por utilizar como primera solución a la combinación de 3D Studio Max y Adobe Director y como segunda opción a Second Life.

Comentar también, que las herramientas propuestas, cumplen las condiciones requeridas. Bien es verdad, que hay una diferencia importante en la última de las cuestiones que se refiere a su modo de ejecución, ya que, si bien, la primera de las realizaciones prácticas, que utiliza el software Director permite su publicación en Internet y en ordenador personal, la segunda solución propuesta, es decir, Second Life, requiere la entrada en sus servidores, lo que significa que su visualización solo es posible mediante navegación por Internet, por tanto, no permite su ejecución si no estamos en la red.

Debemos añadir, que no son estas las únicas soluciones posibles al problema planteado. De hecho, el autor de este documento, utiliza otras tecnologías afines para realizar obras similares, pero requieren de conocimientos más técnicos sobre todo en el área de la programación.

Por tanto, como conclusión, podemos decir que las aplicaciones que cumplen con las expectativas que buscábamos existen actualmente, y además, tienen una implantación en el mercado informático muy consolidada avalada por sucesivas versiones.

Conclusiones referentes al curso y su estructura.

Una vez que teníamos claro el objeto y ámbito de estudio, nos centramos en la consecución del curso que pretendíamos realizar. El primer reto era encontrar una estructura interna que haga de este curso un soporte sólido, eficaz y factible para que los alumnos puedan conseguir la capacidad de crear contenidos multimedia interactivos. Estudiamos algunos de los cursos que desarrolla la universidad y también algunas asignaturas para utilizar su programación didáctica como base de nuestro desarrollo.

Basándonos en que existen unos conocimientos comunes de tipo teórico en todos los programas evaluados, llegamos a la conclusión que era conveniente crear un capítulo en el que se abordase exclusivamente la parte teórica. Este capítulo sería impartido en primer lugar. Por otra parte, el estudio de las alternativas para llevar a la práctica esta tesis, nos hizo deducir que sería mejor realizar este trabajo mediante dos desarrollos prácticos y en cierta forma, complementarios. Este planteamiento tiene la ventaja de que una determinada cuestión se estudia primero teóricamente y luego se implementa desde dos puntos de vista distintos. Esto hace que el tema de estudio sea más fácil de entender y aporta una panorámica más global.

Estos hechos nos llevó a plantear el curso en tres secciones: una primera parte teórica y dos realizaciones prácticas con una misma estructura interna. Este diseño puede parecer repetitivo, pero creemos que aporta a todo el conjunto una visión muy sólida. Hace del curso una entidad muy compacta y didácticamente eficiente, ya que se repiten los conceptos en el mismo orden,

lo que facilita su comprensión. Por ejemplo, si estamos tratando el tema de la iluminación, primero se aborda desde el plano teórico, luego en su solución práctica Max-Director y por último en su versión Second Life. Lo que nos lleva a concluir que la estructura del curso podemos considerarla bastante adecuada desde el punto de vista de la enseñanza y el aprendizaje.

Otro tema a tratar es la extensión y profundidad del curso que hemos realizado. Para ello, como ya explicamos, estudiamos la oferta que se encuentra hoy en día en academias privadas y universidades. Estos estudios siempre se dividen en módulos hasta cierto punto independientes, entre los que podemos citar: modelado, texturizado, iluminación, animación... El criterio final a la hora de incluir o no ciertas partes, lo tomamos nosotros mismos, siendo conscientes que existen algunos módulos que no hemos abordado. Al mismo tiempo, estamos satisfechos de los temas elegidos, sobre todo por su practicidad.

Por otra parte, la propia extensión de este documento, ilustra que debíamos limitar la profundidad con la que se trabajan cada uno los módulos. Concluimos pues, que pretendiendo hacer un curso práctico, es bastante equilibrado en extensión y profundidad, aportando conocimientos generales válidos y útiles.

Al hablar de la estructura temporal del curso quisiéramos comentar que los módulos siguen una secuencia “natural” en el desarrollo de este tipo de trabajos creativos. Por ejemplo, en primer lugar se modela, luego se texturiza, más tarde se ilumina, etc. Este orden secuencial se utiliza universalmente. El propio autor de esta tesis, ha impartido con esta metodología algunos cursos de este tipo y realizado trabajos comerciales y concluye que es acertada y lógica para la consecución de los objetivos docentes.

Queremos recalcar que esta tesis aunque indisolublemente ligada a las tecnologías empleadas y enfocada a un tipo de alumno en particular, se plasma en un curso donde utilizamos una metodología didáctica que no depende de estos dos factores, ni del tipo de alumno ni de la tecnología utilizada. Esto queda claramente expuesto en el mismo índice, ya que se abordan los mismos temas pero utilizando aplicaciones distintas; lo cual prueba que existen siempre unos ítems en común que tienen todas las aplicaciones multimedia interactivas, tenga este el objetivo de hacer videojuegos, exposiciones multimedia o la que fuere.

Al comienzo de este documento comentamos que existe una parte invariable del tema que nos afecta y otra que evoluciona día a día, ya que depende directamente del software. Por tanto, debemos resaltar el hecho de que podemos extraer un conocimiento permanente, que no varíe en el tiempo significativamente y utilizarlo como método en el aprendizaje de nuevas tecnologías. La conclusión evidente es que el software de este tipo, evoluciona en cuanto a velocidad de ejecución, número de polígonos que puede visualizar, la metodología de trabajo,... pero suelen mantener las características básicas.

Conclusiones referentes a las realizaciones prácticas.

Una conclusión que nos parece de especial relevancia, es que los objetivos propuestos al principio que resaltaban la necesidad de alcanzar realizaciones realistas, se han cumplido. En efecto, las prácticas realizadas demuestran en sí mismas que se han podido llevar a cabo. Además, el autor de este documento, al ser quien ha ido desarrollando estas prácticas, ha comprobado su grado de dificultad, su componente técnica, e incluso una estimación sobre el tiempo que puede llevar su realización en un ordenador de uso común. Profundizando en esta cuestión, el procedimiento de realización de todas y cada una de las prácticas se ha llevado a cabo acompañado de una amplia exposición de ilustraciones para guiar al lector de la mejor manera que el autor ha entendido.

También queremos destacar que en todo momento, hemos atendido a la máxima del hecho práctico. En las dos implementaciones prácticas se han diseñado ejercicios que tienen una propiedad que debemos valorar. Y es que las prácticas descritas están llevadas al papel, directamente después de su realización y comprobación por parte del autor. Siempre se ha

buscado resolver de la manera más sencilla los problemas técnicos, por tanto, no se plantean sus consecuencias artísticas, dado el ámbito de conocimiento del autor de este escrito.

Creemos que son realizaciones prácticas concretas y específicas que intentan dar una solución a una cuestión determinada y que deben valorarse en su conjunto. Una conclusión que podíamos extraer es que los alumnos pueden encontrar útil este curso para controlar las tecnologías expuestas. También podemos concluir que tal como está diseñado el curso que proponemos, se puede llevar a cabo como uno o varios cursos reales, es decir, que se puede llegar a dividir por módulos y por realizaciones prácticas.

Conclusiones referentes a la elección de las aplicaciones empleadas.

Una vez aclarado el tipo de aplicaciones que necesitábamos para realizar nuestro cometido, surgieron muchas alternativas interesantes. Para estudiar de una manera ecuánime estas opciones, planteamos una serie de condiciones que debían cumplir para que pudiesen ser tomados como posibles realizaciones de la parte práctica. La primera de estas condiciones era el precio del software, ya que la filosofía de trabajo es utilizar recursos comunes. Las demás circunstancias se estudian con detenimiento en el texto.

Dada la formación como Ingeniero en Informática de este autor y sus experiencias previas en este campo y en empresas de investigación como INESCOP (Instituto Español del Calzado y Conexas) y otras instituciones privadas, se tenía una idea aproximada de la situación general del mercado, y se ha intentado plasmar en la tesis. Finalmente, se eligieron Director y Second Life como tecnologías que pueden servir satisfactoriamente a nuestros fines. Por tanto, podemos sacar la conclusión que existiendo varias alternativas que cumplen todos los requisitos, las utilizadas en esta tesis han sido elegidas de forma objetiva.

Como autor de este texto, tengo que comentar que en ocasiones utilizo otras aplicaciones como Unity 3D y UDK. Sin embargo, el método para introducirme y empezar a realizar pruebas en estos programas fue el mismo que el utilizado en esta tesis. Podemos plantearnos entonces la pregunta: ¿por qué no utilizar estas otras alternativas? La respuesta es que en la actualidad existen aplicaciones como las dos citadas y otras, que incorporan más utilidades que Director y Second Life. Por ejemplo, las dos tienen un sistema de creación de paisajes superiores al que incorpora Second Life.

Este es un claro ejemplo de lo que escribíamos anteriormente: en lo fundamental todas las aplicaciones son parecidas pero existen diferencias según el tipo de trabajos al que están enfocados. Si UDK tiene como objetivo crear videojuegos, tendrá en común con Director el ser un software que permita la multimedia interactiva, pero además será muy útil que cuente con una herramienta de creación de "terrenos", algo totalmente prescindible en otras aplicaciones mas generalistas. Podemos decir, pues, como conclusión, que hemos optado por programas que cumplan con las condiciones necesarias pero sin algunas complicaciones accesorias que podemos evitar, siempre pensando en el tipo de curso que queríamos realizar.

Conclusiones referentes a los estudios de situación de estos ámbitos del conocimiento.

Otra serie de conclusiones que podemos extraer son las que se derivan de la realización de los estudios de situación de las aplicaciones y obras multimedia, y que se exponen en el capítulo 2. Por resumir la cuestión, podemos afirmar que al examinar los estudios del nivel de una tesis y centrándonos en el Estado español, hemos encontrado que sería interesante la realización del presente trabajo, ya que podría complementar de alguna manera las ya existentes. No hemos encontrado ninguna tesis que coincida de manera evidente con el mismo asunto aquí estudiado por lo que creemos que es de interés el planteamiento y las soluciones aportadas en este documento.

Cuando estudiamos la oferta de centros de nivel universitario, tanto públicos como privados, que proponen el estudio de este tipo de conocimiento, confirmamos que no eran muy

numerosos. Buscamos la oferta de estudios de un tipo de software, no de un programa en particular, pero aún así fue difícil encontrar cursos del nivel y los contenidos que proponemos.

En España, solamente en las academias privadas de grandes ciudades como Madrid, Barcelona, Valencia, etc. se abordan estos estudios. Además, casi nunca se plantean para un perfil de estudiante de Bellas Artes, sino que se enfocan a estudiantes técnicos, por lo que ponen un énfasis muy grande en determinados apartados como el de programación.

Tras ampliar la búsqueda fuera de nuestras fronteras, vimos un cambio bastante grande, ya que en otros países de nuestro entorno como por ejemplo, Francia y Reino Unido son habituales los cursos de medios tecnológicos en facultades y escuelas de Arte, sobre todo como cursos postgrado. Al trasladar nuestra búsqueda a Estados Unidos y Canadá, confirmamos que estos estudios están integrados en algunas escuelas de Arte, incluso desde el primer curso. Como conclusión podemos decir, que el curso propuesto en este documento podría ocupar un cierto espacio débilmente utilizado hasta ahora.

Cuando una persona comienza a estudiar una tecnología nueva, tiene que estar dispuesto a dedicar cierto tiempo y esfuerzo para llegar a dominarla. En muchas ocasiones, se estudia lo que otros usuarios más avanzados han conseguido al utilizar con maestría estos programas como referencia de aprendizaje. En este documento, hicimos esto mismo, estudiando a algunos creadores avanzados en los programas que nos incumben y también lo extendimos al área de exposiciones. Llegamos a concluir que existen muestras muy conocidas, así como autores que utilizan con éxito los programas que proponemos desde hace años y que sus producciones muestran en sí mismas el potencial creativo que buscamos.

Este trabajo busca que el alumno que siga el curso adquiera una nueva capacidad creativa. El objetivo fundamental de la tesis es que adquiera los conocimientos necesarios para crear entornos tridimensionales multimedia interactivos. Estos medios que incluyen imágenes en dos dimensiones, modelos tridimensionales, sonidos, animaciones y video, no serán estáticos, sino que el autor tendrá la posibilidad de dotarles de interacción, esto supone dar respuesta ante acciones de diversos orígenes como el ratón, el teclado, el paso del tiempo,... El creativo también podrá decidir si la presentación de su obra se realiza desde disco duro o en la red, pero siempre podrá optar por Internet.

Hemos demostrado con este curso, que esta capacidad creativa es posible adquirirla siguiendo nuestro documento y realizando las prácticas propuestas. De hecho, el autor de esta tesis, empleó las técnicas estudiadas para generar un espacio virtual que recrea un museo donde se muestran obras pictóricas y otras en tres dimensiones. También incluye sonido ambiental e iluminación interactiva que permite encender, dirigir y apagar luces. Otra propiedad implementada es que podemos movernos con libertad dentro del espacio creado. Todo este conjunto, se puede experimentar desde un navegador web actual. Por eso, podemos concluir que el alumno al terminar el curso tendrá unas posibilidades creativas útiles y tecnológicamente avanzadas.

Conclusiones referentes al nivel del curso propuesto.

Al comenzar a diseñar el curso nos fijamos el objetivo de seguir los procedimientos didácticos que se utilizan hoy en día en las universidades españolas, en el sentido de seguir una programación que establezca un marco procedimental didáctico. Sinceramente, al principio tuvimos ciertas reservas de si sería posible ajustar el esquema modular variable que habíamos elegido (que estructura el contenido) con las programaciones didácticas, que son más rígidas y que siguen las asignaturas impartidas en nuestras facultades. La razón es que para diseñar el contenido y el orden de los módulos hemos tenido como referente principal a instituciones educativas del ámbito privado que normalmente son menos estrictos (en el peso relativo de tiempos y otros factores) que los estudios realizados en las universidades donde todo está más establecido y medido con precisión. Debemos confesar que al final no fue un problema demasiado grave ya que llegamos al final del primer caso práctico de forma muy fluida y esto mismo se confirmó al concluir el segundo desarrollo práctico.

De lo comentado en el párrafo anterior podemos establecer dos conclusiones. La primera de ellas nos reafirma en la idea de que el curso realizado se puede impartir en nuestras facultades sin grandes problemas ni inversión en nuevos recursos. Se podría programar como curso postgrado, ya que permite ser asimilada como otra asignatura o curso más. La segunda conclusión es que existe cierta naturalidad o universalidad en el desarrollo de los módulos, ya que en todas las entidades educativas estudiadas se llevan a cabo en el mismo orden. Esto hace que el uso de materiales como tutoriales o literatura al respecto, sea fácil de encontrar y adquirir.

Otro tema a estudiar y que también se ha de incluir en la programación didáctica del curso es el de los recursos tanto materiales como humanos necesarios para desarrollarlo. En este sentido no tuvimos problemas porque se establecieron inicialmente unas condiciones muy precisas al respecto. Por una parte se utilizarán herramientas software que se puedan ejecutar en los ordenadores más utilizados en la actualidad como son los ordenadores personales con sistema operativo Microsoft Windows.

Por otra parte, el curso debe desarrollarse a partir de unos programas que puedan ser utilizados de forma escalar, es decir, que se pueda decidir con qué complejidad técnica se quiere trabajar. De esta forma, los alumnos con conocimientos dispares podrán encontrar en el mismo programa utilidades en consonancia, es decir, que se adapten a su nivel inicial. Además, las herramientas utilizadas incluyen métodos que permiten realizar trabajos complejos sin casi escribir código. Al elegir el software utilizado en el curso, se mantuvo la atención en este punto y podemos afirmar que se cumple satisfactoriamente. En este punto concluimos que el curso se puede impartir en ordenadores personales de precio asequible y por profesorado que no requiere una alta cualificación técnica.

Conclusiones referentes a la evolución futura de estas tecnologías.

El tipo de software que hemos venido utilizando en esta tesis, a dado lugar a aplicaciones especializadas, como por ejemplo, los llamados “motores de videojuegos”. Por otra parte, se supone que la implantación de estándares que permiten la creación y ejecución de entornos 3D multimedia interactiva directamente en los navegadores como son HTML 5, WebGL, Unity Web Player y otros, auguran un rápido crecimiento y exposición de este tipo de aplicaciones en un futuro cercano.

Creemos que este hecho proporcionará las condiciones para que aparezcan editores que faciliten la realización de obras digitales del mismo tipo que las propuestas en el presente escrito. Esta facilidad hará que estas herramientas multimedia sean utilizadas cada vez más por estudiantes de un perfil creativo. Este campo creativo ya se han visto beneficiado y en el futuro aún lo será más, por la aparición de dispositivos móviles con acceso a Internet como smartphones y tablets.

Nos gustaría destacar que debido a la evolución continua que sufre el mundo de la tecnología informática en general, y en particular del tipo de aplicaciones que nos incumben, hemos tenido la tentación de cambiar de tecnología. Por supuesto, hemos estudiando alternativas al software utilizado y en la actualidad existen muy serias opciones aparte de las expuestas en esta tesis. Sin embargo, si lo miramos desde un punto de vista distinto, podemos también afirmar, que el tiempo empleado en adquirir los conocimientos necesarios para poder extraer buenos resultados del software empleado, no es en balde, ya que aunque con bastante tiempo de vida, siguen siendo plenamente vigentes. Por tanto, como conclusión podemos escribir que el software de este tipo no es tan efímero como a veces se pueda llegar a pensar.

Conclusiones referentes a la completitud del curso

En cuanto al contenido de los módulos que componen el curso, podemos afirmar que son bastante completos. Ya comentamos con anterioridad que hemos buscado que el curso tenga un carácter general. Tampoco debemos olvidar que existen materias que a veces se pasan por alto pero que en este curso en particular, cobran especial importancia. Quisiera resaltar dos de ellas: los motores físicos y los sistemas de partículas.

Se entiende por física o motor físico, aquella parte del software que se encarga de calcular efectos físicos que imiten la realidad al aplicarlo sobre objetos modelados en tres dimensiones. Este apartado normalmente se implementa como un módulo accesorio al software principal y es crucial para dar verosimilitud a una simulación de la realidad física. Aporta propiedades a las escenas y objetos como gravedad, peso, densidad y efectos atmosféricos como viento, lluvia,... Por todo ello no podíamos pasar por alto este tema tan importante en esta tesis. Por otro lado, es un apartado intrínsecamente complicado. Además, cualquier realización práctica que contenga “físicas” eleva su complejidad, por lo que requiere ordenadores más potentes para ejecutarlas en tiempo real.

Con respecto a los sistemas de partículas, son necesarias para realizar efectos especiales o modelos complejos en sí mismos como una bandada de pájaros, el agua de una fuente y otros. También en este caso hemos abordado el tema sin complejos, realizando un ejercicio completo y muy típico: el fuego de una llama, que se explica en ambos casos prácticos desde el principio y con un desarrollo paso a paso por su propia dificultad.

Por esto, podemos extraer como conclusión que el curso generado es bastante completo pues aborda todos los temas que hemos considerado imprescindibles. Esto no quiere decir, que no existan otras materias que para algunos trabajos creativos pueden ser muy útiles como los ya citados módulos de creación de terrenos, vegetales, etc.

Queremos aclarar que estamos en todo momento utilizando programas que facilitan la labor del autor de obras multimedia interactivas. Son editores creados por empresas con mucha experiencia que pretenden llegar a un público muy extenso, y por ello, intentan suavizar la curva de aprendizaje de las mismas. Esto quiere decir que aunque alguna materia en particular puede ser complicada, lo es por sí misma y por el estado tecnológico en que se encuentra su desarrollo.

Los programas utilizados como 3D Studio MAX, Director y Second Life, facilitan esta ardua tarea y para ello disponen de sofisticados mecanismos como los “comportamientos” o modelos de programación entre lo que cabe citar la programación orientada a objetos, con la atención puesta siempre en la labor de edición del creativo. Nosotros hemos aprovechado estas características para intentar reducir en lo posible el esfuerzo que tiene que hacer el alumno para dominar estas herramientas tan potentes. En conclusión, podemos decir, que hemos simplificado en cierta medida el aprendizaje que esta tesis propone.

Conclusiones finales.

Para acabar, queremos enfatizar la existencia de objetivos concretos en todo el planteamiento y resolución de esta tesis. Los objetivos siempre han sido de carácter práctico. Además hemos tratado de ilustrar con ejemplos, el interés general que el tema tratado tiene actualmente dado el momento tecnológico que estamos viviendo. Las ventajas profesionales que este tipo de conocimientos pueden aportar a nuestro alumnado parecen evidentes.

También queremos recalcar que hemos seguido una metodología de estudio y realización del curso avalada por instituciones de ámbito público y privado de dentro y fuera de nuestras fronteras. Además esta metodología ha sido experimentada en primera persona por el autor.

El interés general y dificultad de los conocimientos propuestos son de cierta magnitud lo que ha supuesto un reto para el autor, siempre teniendo presente la finalidad didáctica de esta tesis.

Bibliografía.

Libros:

Jones, A; et al. *Edición Especial 3D Studio Max 3 Animación profesional.* Prentice Hall, Madrid, 1998.

Chismar J. P. *Edición Especial 3D Studio Max 3, Animación*. Prentice Hall, Madrid, 2000.

Boardman, T; Hubbell, J. *3D Studio Max 3, Modelado, Materiales y Representación*. Prentice Hall, Madrid, 2000.

Thomas, Tejedor; Bootello, Ignacio; Burgos, Javier. *3D Studio Max R3.* Anaya Multimedia, Fuenlabrada (Madrid), 2000.

Guerrero, Carlos. *Guía práctica 3D Studio Max*. Prensa Técnica, Madrid, 1997.

Macromedia Inc. *Lingo Dictionary*. Macromedia Inc., San Francisco, 1997.

Macromedia Inc. *Learning Lingo*. Macromedia Inc., San Francisco, 1997.

Macromedia Inc. Using Director. Macromedia Inc., San Francisco, 1997.

Rymaszewki, Michael. *La guía oficial de Second Life*. Anaya Multimedia, Madrid, 2008.

Weber, A; Rufer-Bach, K; Platel, R. *Creating Your World: The Official Guide to Advanced Content Creation for Second Life.* Linden Research, Inc., Indianapolis, Indiana, USA, 2008.

Moore, D; Thome, M; Haigh K. *Scripting Your World: The Official Guide to Second Life Scripting*. Linden Research, Inc., Indianapolis, Indiana, USA, 2008.

Terdiman, D. *The Entrepreneur's Guide to Second Life: Making Money in the Metaverse*. Linden Research, Inc., Indianapolis, Indiana, USA, 2008.

Martínez López, Ruth; García Serrano, Alberto. *Exprime Second Life*. Anaya Multimedia, Fuenlabrada (Madrid), 2007.

Goldstone, W. *"Unity Game Development Essentials"*, Packt Publishing, Birmingham, UK, 2009.

Artículos de revista:

Ramshaw, Mark. *Trabajos reales en 3D*. Todo 3D, número 50 (marzo, 2005), página 25.

García, Moisés. *vWWW: Internet en tres dimensiones*. iWorld, número 52 (septiembre, 2001), página 51

Arte y Diseño por ordenador. *Software de animación 3D*. Arte y Diseño por ordenador, número 66 (marzo, 2005), página 88.

PC Today. *Programas de 3D, mundos virtuales*. PC Today, número 119 (junio, 2005), página 24.

Arte y diseño por ordenador. *Usa Unity 3D para crear un entorno 3D virtual para la web*. Arte y diseño por ordenador, número 118 (julio, 2009), página 82.

Artículos en la red:

Plumbo, Marty. "*Creating Interactive Virtual Objects with Shockwave3D"* [en línea], Macromedia Inc., 2005, <http://www.adobe.com/devnet/director/articles/virtual_objects.html> [20 de marzo de 2009]

Martín, Domingo. "*Modelado 3D y animación por ordenador"* [en línea], Departamento de lenguajes y sistemas informáticos. Universidad de Granada, 2008 <http://lsi.ugr.es/docencia/lbbaa/m3dao/> [22 de marzo de 2010]

Blanes, Joaquín. "*Ángulos de los planos"* [en línea], blog personal, 2007 <http://laimagendescubierta.wordpress.com/2007/10/06/angulo-de-los-planos/> [06 de septiembre de 2009]

Brautigan, Rack. "*Diseño de ropa con sombreado"* [en línea], blog personal, 2008 <http://imagenperdida.blogspot.com/2008/04/tutorial-crear-ropa-para-sl.html> [10 de junio de 2009]

Guerrero, Atlante. *"Sketchlife convierte modelos de Google Sketchup a Second Life"*, CreaSL, 2009, <http://www.creasl.org/> [06 de octubre de 2009]

Linden, Samuel. *"Shared media, Security, and Privancy"*, Linden Research, Inc., 2010 <https://blogs.secondlife.com/community/technology/blog/2010/03/15/shared-media-security-and-privacy> [18 de marzo de 2010]

Aldrich, Luis. *"Avatares y RL mezclados en Avatar Studios"*, Virtual Spain, 2010 <http://www.virtual-spain.es/home/blogs/avatares-y-rl-mezclados-en-avatar-studios.html> [25 de marzo de 2010]

Tradky software. "*Museos y entornos monumentales virtuales 3D"*, Tradky software, 2008 <http://www.tradky.com/m-galerias-de-arte-virtuales-3d.html> [15 de mayo de 2009]

Reventós, Laia. *"El software de Tradky construye espacios virtuales 3D en Internet"*, Elpais.com, 2008, <http://www.elpais.com/articulo/red/software/Tradky/construye/espacios/virtuales/3D/Internet/elpeputeccib/20080724elpcibenr_3/Tes> [10 de agosto de 2008]

Albatros, Llilliam. *"Nanoprims"*, Virtualisland.es, 2008, <http://www.virtualisland.es/tutoriales/tinyprims/index.html> [14 de marzo de 2009]

Benazzi, Arkadi. "*Script para control de puertas"*, Secondspain, 2007, <http://foros.secondspain.es/index.php?option=com_joomlaboard&id=23582&catid=14&func=sb_pdf> [23 de noviembre de 2008]

Wordpress. *"A guide for teachers"*, Wordpress**,** 2007, <http://vorticism.wordpress.com/help/a-guide-for-teachers/> [07 de febrero de 2008]

Manson, Hilary. *"Make my script"*, Seven on seven, 2007, <http://www.3greeneggs.com/autoscript/> [02 de octubre de 2007]

Rosenbaum, Eric. *"Scratch for Second Life"*, Eric Rosenbaum blog, 2009, <http://web.mit.edu/~eric_r/Public/S4SL/> [30 de julio de 2009]

Outkast. *"Curso para emprendedores en Second Life"*, Outkast, 2008 <http://revver.com/video/820710/10-curso-para-emprendedores-en-second-life/> [01 de agosto de 2009]

D´Angelica, Alicia. *"¿Qué es el arte en 3D?"*. Página personal, 2010, <http://www.aliciadangelica.com.ar/arte_digital_art.html> [01 de abril de 2010]

Enlaces de interés:

http://homepage.mac.com/ikiboy/ShockDemo/Museum.html

Ejemplo de museo virtual realizado en Director con Shockwave 3D.

http://homepage.mac.com/ikiboy/ShockDemo/

Conjunto de demostraciones de Shockwave 3D.

http://www.aliciadangelica.com.ar/

Diseñadora argentina que trata el tema sobre todo de fractales.

http://director-online.com/

Portal y foro sobre el software Director y Shockwave 3D.

http://www.demoscene.info/

Portal de muestras con trabajos que mezclan imagen por ordenador y sonido.

http://demo.monostep.org/

Portal de trabajos artísticos por ordenador para la web.

http://www.interaccess.org/exhibitions/index.php?id=32

The appearance machine. Trabajo experimental visionado por la web.

http://www.e-77.net/

Página personal del diseñador español de personajes 3D Ernesto Rodríguez.

http://www.benayoun.com/

Página personal de Maurice Benayoun, artista con trabajos web 3D

ANEXO I:

FIGURAS

Imágenes

Tablas

Gráficas

ANEXO II:

GLOSARIO DE TÉRMINOS DE 3D STUDIO MAX Y DIRECTOR

Glosario de términos utilizados en las aplicaciones 3D Studio MAX y Director.

No vamos a incluir aquí las palabras que son directamente la traducción exacta del inglés ya que no se trata de un diccionario español-inglés. Por lo tanto las palabras que tienen las mismas acepciones que en español no las introduciremos en este glosario temático.

Categorías empleadas para clasificar los términos:

Software

Director > General
Director > Animación
Director > Importación
Director > Interacción
Director > Modelado
Director > Texturizado
Director > Iluminación
Director > Cámaras
Director > Programación
Director > Publicación
Director > Sistemas de partículas

MAX > General
MAX > Animación
MAX > Cámaras
MAX > Exportación
MAX > Modelado
MAX > Iluminación
MAX > Renderizado
MAX > Superficies

Acciones
Director > Interacción
Este es un tipo de comportamiento que ejecutará una respuesta a la aparición de un disparador que se habrá ejecutado, a su vez, como consecuencia de una interacción del usuario. Las acciones siempre son consecuencia de un disparador. Existe en Director una biblioteca de acciones muy comunes en 3D que se pueden configurar.

AddChild
Director > Interacción
Es un método que permite añadir elementos 3D a un grupo y que por tanto, necesita el nombre del objeto como parámetro.

Adobe® Director®
Software
Es una aplicación para desarrollar software de autor. Enfocado a la creación de programas ejecutables ricos en contenidos multimedia. Se considera una de las herramientas más potentes para la integración y programación de material digital de distinto tipo como es la imagen, el texto, el audio, el vídeo, las animaciones flash y los mundos 3D. Contiene herramientas para la programación en Lingo o JavaScript incluyendo editores de texto, un compilador y un depurador.

Ambient
Director > Iluminación
Es una luz siempre existente en una escena de Director con Shockwave 3D ya que es la luz por defecto. No aporta profundidad a los objetos, aunque se puede colorear para crear distintos ambientes de base.

Analyze
MAX > Exportación
Hace que el programa exportador presente un gráfico de tipo tarta dónde se puede comparar los distintos pesos que tienen los distintos elementos exportados como puede ser la geometría, las texturas, las animaciones, etc.

Anchura del campo visual
MAX > Cámaras
Ver FOV

Animación de fotogramas
Director > Animación
Este tipo de animación es muy empleado por su sencillez. Se trata de crear una serie de fotogramas donde los modelos pueden cambiar sus propiedades básicas como posición, rotación y escala.

Animación de huesos
Director > Animación
Este tipo de animación es bastante compleja pero muy espectacular, ya que suponen la creación de una jerarquía de estructuras que normalmente se llaman huesos. Cada uno de los huesos se asocia con un elemento visible de tal manera que al mover el hueso, este mueve el objeto asociado. Esto crea una animaciones muy realistas de animales, personas, robots, etc.

Aparatos atmosféricos
MAX > Modelado
Son unas herramientas que permiten crear y configurar efectos atmosféricos. MAX proporciona este tipo de herramienta en la ventana “Enviroment and effects” que aparece al ejecutar el comando del menú Rendering > Enviroment. Aquí pulsaremos sobre el botón “Add” dentro del grupo Atmosphere y podremos ver una ventana para elegir el aparato atmosférico deseado que puede ser niebla, fuego, luces volumétricas, etc.

Atenuación
MAX > Iluminación
Simulación mediante unos parámetros de las luces, del hecho físico de la pérdida de intensidad de la iluminación proporcionada por una fuente de luz con la distancia.

Atenuacion, Attenuation
Director > Iluminación
Es una propiedad de las luces de tipo punto y foco que determina la disminución de la cantidad de luz con la distancia. Su valor se introduce con un vector que no es una dirección, ni posición 3D sino tres datos de tipo decimal para definirla.

Author check
MAX > Exportación
Hace que el programa exportador presente un informe de posibles problemas al exportar.

Autodesk® 3ds MAX®
Software
Es un programa de creación de gráficos y animación 3D desarrollado por Autodesk Media & Entertainment. Es uno de los programas de animación 3D más utilizados. Dispone de una sólida capacidad de edición, una omnipresente arquitectura de plugins y una larga tradición en plataforma Microsoft Windows. Es utilizado en mayor medida por los desarrolladores de videojuegos, proyectos de animación de películas y anuncios de televisión y arquitectura.

Autoiluminación
MAX > Superficies
Parámetro de los materiales estándar de MAX que controla un efecto de iluminación del propio material. No produce ningún tipo de iluminación en la escena como resplandores, sombras, etc salvo que la geometría a la que se asigne este material se verá con más tonos blancos.

Autorun.ini
Director > Publicación
Fichero de autoarranque para sistemas operativos Windows que se incluye normalmente en el directorio raíz de un CD o un DVD. Consiste en un fichero de texto con las sintaxis:
[autorun]
Open= nombre_del_fichero_inicial.exe

AVI
Software
Audio Video Interleave o intercalado de audio y video. Es un formato que permite guardar audio y vídeo, fue lanzado en 1992 por Microsoft como soporte para su tecnología "Video for Windows". Permite almacenar un flujo de video y varios flujos de audio simultáneamente. AVI es conocido como un formato contenedor porque no concreta el formato concreto ni de audio ni del vídeo, que tendrá que ser ejecutado por un programa ajeno a AVI al que generalmente se le denomina códec. Algunos de estos códec son DivX, Xvid y otros.

Behavior
Director > Interacción
Ver comportamiento

Bend
MAX > Modelado
Modificador de geometría utilizado en MAX que curva el modelo en un determinado eje. Lo dobla en una cantidad porcentual que podemos elegir junto con el eje de torsión. Por ejemplo, de esta manera podemos hacer que un cilindro recto se curve ligeramente para crear el mástil de una farola y que la luz se oriente hacia el suelo.

Bevel
MAX > Modelado
Operación utilizada en las mallas poligonales a nivel del subobjeto poligono. Produce una extrusión de la cara de un modelo seguido de un cambio de escala en el polígono exterior. De esta manera se pueden crear más polígonos hacia fuera o hacia dentro de la malla poligonal e incrementando su complejidad. Es una operación fundamentalmente utilizada en el método llamado "Box modeling".

Bézier
MAX > Modelado
Son curvas descritas por Pierre Bézier en la década de 1960 para el trazado de dibujos técnicos que aplicó al diseño de automóviles Renault donde trabajaba en programas CAD. En la actualidad, las aplicaciones de diseño vectorial como Adobe Illustrator, Corel Draw y otros como PhotoShop, incluso lenguajes de descripción gráfica como PostScript, denominan algunas herramientas de dibujo como trazados bézier, pluma o lápiz bézier, etc. En MAX podemos crear shapes que serán curvas creadas con este método.

Biblioteca
Director > Programación
Se conoce con este nombre a agrupaciones de behaviors. Es decir, cuando tenemos varios comportamientos que tienen funciones relacionadas, normalmente se agrupan en una biblioteca. Por ejemplo, Director proporciona bibliotecas de acciones y disparadores para poder trabajar con comportamientos predefinidos.

Biblioteca de materiales
MAX > Superficies
Es una archivo de extensión .mat de MAX que permite guardar en memoria secundaria una colección de materiales.

Biped
MAX > Animación
Modificador de Character Studio utilizado en MAX para crear un esqueleto de un personaje bípedo. Junto con physique, permiten la creación de personajes con esqueleto con el fin de crear animaciones.

Bitmap
Software
También conocidos como raster o mapa de bits. Es una imagen con una estructura que almacena una cuadrícula rectangular de píxeles (punto de color). Por tanto, su calidad vendrá dada por una anchura y altura (la llamada resolución) y una profundidad de color (la cantidad de bits por píxel que determina el número de colores distintos que se pueden codificar por píxel). Se les distingue del otro gran tipo, las imágenes vectoriales, en que estas están se generan mediante objetos geométricos definidos matemáticamente como curvas de Bézier, polígonos y otros.

Blend
Director > Texturizado
Ver nivel de opacidad.

Blendrange
Director > Sistemas de partículas
Controla la opacidad del sistema de partículas mediante dos propiedades: start y end, que se pueden configurar para definir la opacidad al principio y al final de la vida de una partícula. Entre uno y otro momento, el sistema interpolará la opacidad de las partículas.

Bones
MAX > Modelado
Sistema de huesos de 3d Studio MAX. Permite la creación de una jerarquía de huesos al construirlos de uno en uno. Cada uno tiene una parte rotatoria y una longitud, por lo que podemos construir bípedos, cuadrúpedos y otros como serpientes, etc. Es más complejo que el sistema Character Studio pero tan bien más flexible.

Bonesplayer
Director > Animación
Modificador asociado a un objeto 3D que permite definir una animación de huesos. Contiene propiedades y procedimientos para ejecutar y controlar este tipo de animación.

Boton on/off
Director > Interacción
Estos controles funcionan como un conmutador, ya que solo pueden tener dos valores: verdadero o falso, activo o no activo. Se utilizan para el control de funciones que solo puedan estar en dos estados y para cambiar de uno a otro requieren de la interacción con el usuario.

Box Modeling
MAX > Modelado
Método de construcción de una malla poligonal que toma como base una caja para ir construyendo un modelo complejo a base de mover vértices y operaciones de extrusión y bevel de polígonos. Es un método muy intuitivo y genera modelos con baja cantidad de polígonos por lo que se suele utilizar para crear personajes para videojuegos. El problema es que suelen ser muy toscos por lo que se normalmente se suaviza su superficie con modificadores como meshsmooth (suavizado de malla) que por otra parte, aumenta mucho la cantidad de polígonos del modelo, por lo que se debe utilizar con cautela.

Cámara con objetivo
MAX > Cámaras
Es una cámara con dos componentes: el punto de vista y el target u objetivo. Este objetivo se fija en escena mediante un punto tridimensional, por tanto, es una cámara simple pero que permite una más fácil definición de la dirección final de la cámara. Este tipo de cámaras

permiten la creación de efectos cinematográficos como el travelling, ya que podemos mover la cámara a través de una trayectoria mientras el punto al que se dirige permanece fijo.

Cámara Libre
MAX > Cámaras

Se trata de una entidad de la escena 3D que permite la visualización de esta desde una determinada posición. Este objetivo lo realiza mediante una proyección del 3D al 2D, por lo tanto, no es más que una transformación matemática de polígonos. Existen varios tipos de proyección como la isométrica, la perspectiva, etc.

Campo de vista, Field of view
Director > Cámaras

Es la apertura de foco. Determina el ángulo de vista vertical de la cámara y por tanto afecta a lo que se visualiza de los objetos en cuanto al detalle.

Canal alpha
Director > Texturizado

Es un canal que admite texturas que incluyen además de unos bits de color, unos bits que definen la transparencia del modelo sobre el que se apliquen. De esta manera dentro de estas texturas se puede definir la vista y la opacidad. Esta opacidad viene definida en escala de grises, donde el color negro equivale a transparencia y blanco a opacidad. Se usan para crear orificios en la superficie de los modelos dónde se aplican así como para efectos especiales.

Canal de desplazamiento, displace
MAX > Superficies

El canal displace en los materiales estándar de MAX permite asignar una textura al parámetro desplazamiento de un material y por tanto, esta textura hará que la superficie sea modificada en su geometría y por tanto es un método más de modelado de geometría.

Canal de opacidad
MAX > Superficies

El canal de opoacidad en los materiales estándar de MAX permite asignar una a la propiedad de opacidad de un material. Esta textura vendrá dada en escala de grises. Si el color es negro puro, el programa simulará la opacidad nula (transparencia total, será invisible) y al revés con el color blanco puro. Los grises se codificarán en consecuencia y por tanto el color RGB, 125,125,125 se interpretará por el renderizador como de opacidad al 50%.

Canal de relieve
MAX > Superficies

El canal de relieve o bump en los materiales estándar de MAX permite asignar una textura al parámetro bump de un material y por tanto, esta textura hará que la superficie, sin ser modificada, aparezca con una rugosidad determinada por este mapa. Ver bump.

Canal difuso
MAX > Superficies

El canal difuso en los materiales estándar de MAX permite asignar una textura al color difuso (el propio) de un material y por tanto será el método más utilizado para asignar una textura a un objeto.

Character Studio
MAX > Modelado

Sistema integrado en MAX compuesto por un sistema de esquelo para bípedos llamado biped y un sistema para conectar el esqueleto a una geometría de malla de polígonos que represente el cuerpo del personaje. Permite la animación mediante la cinemática inversa y otras técnicas como la de huellas.

Chartonum
Director > Programación

Devuelve el código Ascii de una tecla.

ClicLoc
Director > Interacción
Es una propiedad del objeto Mouse de Director que contiene las coordenadas X e Y del último punto de la pantalla donde se hizo clic con el ratón.

Color ambiental
MAX > Superficies
Es el color que toma una superficie estándar en las zonas más sombreadas y por tanto más oscuras de su superficie. Se supone (ya que el sistema no de ja de ser una simulación) que este color es debido al color de la luz ambiental. Si la luz es blanca, normalmente estas zonas presentan un color oscuro.

Color difuso
MAX > Superficies
Es el color que toma una superficie estándar en las zonas de tonos medios, es decir, ni muy iluminados ni muy poco. Este color es debido al color propio del material. Por ejemplo, el oro, mostrará este color en tono dorado.

Color de autoiluminación, emissive
Director > Texturizado
Propiedad del shader standard. Imita el efecto de superficies que emiten luz. Este efecto lo hace eliminando sobras y haciendo que el resto tome colores más brillantes. Se introduce como un color. Su valor por defecto es rgb(0,0,0) y si lo ponemos con rgb(255,255,255) el modelo se rellenará totalmente de blanco, con lo que parecerá que no tiene 3D.

Color especular
MAX > Superficies
Es el color que toma una superficie estándar en las zonas más iluminadas y brillantes de su superficie. Se supone (ya que el sistema no de ja de ser una simulación) que este color es debido a la naturaleza propia del material que simula. Los brillos suelen ser de color blanco, pero lo que realmente simula el tipo de material es la forma y el ángulo de los brillos. Por ejemplo, los metales tienden a mostrar brillos alargados y en filos y los materiales aterciopelados no muestran ningún brillo definido

Colorrange
Director > Sistemas de partículas
Controla el color de cada partícula mediante dos propiedades: start y end, que se pueden configurar para definir el color al principio y al final de la vida de una partícula. Entre uno y otro momento, el sistema interpolará el color de las partículas.

Comportamiento
Director > Programación
Los comportamientos son módulos escritos en Lingo que pueden ser generales o específicos. Los comportamientos específicos son los más usuales y se asignan a sprites para controlarlos. Pueden utilizar variables locales y globales y están estructurados en manejadores que definen cómo se responde a los distintos eventos que capturan.

Comportamiento predefinido
Director > Interacción
Estos comportamientos están ya escritos y vienen integrados en Director. Por ello, el autor solo debe elegirlos y configurarlos. Por ejemplo, existen comportamientos para manejar la cámara por defecto, y el autor debe configurar con que teclas se gobierna dicha cámara.

Currentime
Director > Interacción
Es una propiedad del modificador Keyframeplayer que permite controlar la reproducción del movimiento fotograma a fotograma. El valor de currentime será el valor del fotograma actual y por tanto de esta forma se puede posicionar la animación en el tiempo.

DefaultView
Director > Cámaras
Nombre q ue tiene la cámara creada en 3D Max y que es exportada a Director. Por tanto, esta cámara es la que se usa por defecto.

Deletelight
Director > Iluminación
Es un procedimiento de la escena 3D que permite eliminar una luz mediante código Lingo en una escena Shockwave 3D. Requiere un parámetro: el nombre de la luz.

Diffuse color
MAX > Superficies
Es el color base de una superficie. Cuando un objeto muestra un color sólido, este se crea utilizando tres colores según la cantidad de luz que lo ilumine: color en las sombras también llamado color ambiental (normalmente tiende a negro), color del brillo también llamado color especular (normalmente tiende a blanco) y color en medios tonos o difuso. El diffuse color define el color en los medios tonos de iluminación.

DIR
Director > General
Es la extensión de los archivos editables de Adobe Director. Por lo tanto, se puede considerar como un archivo o proyecto multimedia programable

Disparadores, triggers
Director > Interacción
Son comportamientos que se ejecutan al intervenir de algún modo el usuario. Por ejemplo, si este toca una determinada tecla o pulsa un botón del ratón, el autor puede haber decidido un disparador de respuesta y que este genere, a su vez, unas acciones que por tanto, serán finalmente la respuesta por parte de la aplicación al usuario. Existen una completa biblioteca de triggers 3D en Director que el autor puede configurar.

Displace
MAX > Modelado
Modificador de geometría utilizado en MAX que aplica cambios en la superficie de los objetos tridimensionales en cierta cantidad y según una imagen llamada mapa de desplazamiento. A veces se le compara con los mapas de bump, pero a diferencia de estos, el displace si modifica la geometría del objeto y por lo tanto aumenta la cantidad de polígonos que componen su superficie.

Distancia focal
MAX > Cámaras
Es la longitud entre la lente y el punto focal, es decir, el punto donde convergen los rayos luminosos en el objeto enfocado.

Dithering gráfico
Director > Publicación
Técnica que se utiliza para hacer que los archivos ocupen menos espacio y que consiste en reducir el tamaño de todos los ficheros de imágenes que hallamos utilizado bajando la profundidad de color de los bitmaps. Esto supone una reducción en la calidad que en muchos casos no es apreciable y sin embargo el ahorro en tamaño y sobre todo para una publicación en la web es muy importante.

Drag
Director > Sistemas de partículas
Los sistemas de partículas pueden simular características de tipo físico. Drag es una variable del sistema que permite simular el arrastre. El arrastre mucha influencia sobre otras variables físicas como el viento o la gravedad ya que mide el tanto por ciento de partículas que se verán afectadas por ellas.

Editor de materiales
MAX > Superficies
Módulo de 3d MAX que permite la creación y gestión de superficies para su aplicación en geometrías. Incluye la posibilidad de crear texturas y superficies que las incluyan totalmente parametrizadas. También permite gestionar librerías de materiales y superficies para guardarlas, mezclaras, etc.

Emisor de partículas:
Director > Sistemas de partículas
Origen de un sistema de partículas. Puede ser un punto tridimensional o varios y es configurado por algunos parámetros como mode, region y path.

Emissive
Director > Texturizado
Ver color de autoiluminación.

Emitter:
Director > Sistemas de partículas
Es una propiedad de un sistema de partículas que permite definir cómo es el emisor de las partículas y por tanto, cómo es todo el sistema.

Emitter.maxspeed
Director > Sistemas de partículas
Define la forma en que son emitidas las partículas. En realidad define el tamaño del foco emisor, por tanto, el efecto es si salen de una en una o muchas a la vez.

Emitter.mode
Director > Sistemas de partículas
Variable del emisor de partículas que define el modo de emisión de estas y que puede tomar dos valores: stream o burst.

Emitter.region
Director > Sistemas de partículas
Modo de emisión de partículas en una región del espacio tridimensional y definida por una serie de vectores 3D que describirán la zona desde donde parten las partículas.

Emitter.path
Director > Sistemas de partículas
Modo de emisión de partículas lineal y definido por una serie de vectores 3D que describirán la curva desde donde parten las partículas.

Emitter.pathstrengh
Director > Sistemas de partículas
Esta variable del emisor, define la fuerza con que las partículas siguen la trayectoria marcada por el modo de emisión path. Esta variable trabaja junto con drag para definir lo fiel que es el recorrido que hacen las partículas en su camino definido por el valor de emitter.path.

EXE
Software
Viene de executable, ejecutable. Es la extensión de los archivos ejecutables de código reubicable, lo cual significa que pueden procesarse en direcciones de memoria relativas. Cada vez que se ejecutan pueden utilizar posiciones físicas de memoria distintas. Son por tanto, los archivos que generan procesos de todo tipo: del sistema operativo y de las aplicaciones de usuario serán archivos .exe. Este formato es usado por los sistemas operativos de Microsoft como MSDOS y la familia Windows.

Exitframe
Director > Programación
Este es un disparador que sucede cuando Director sale del fotograma actual, cosa que sucede según la velocidad de reproducción de la película, normalmente 24 veces por segundo. Como

los trabajos que utilizan Shockwave 3D se ejecutan en un solo fotograma, normalmente se utiliza este disparador para controlar la ejecución de código en el tiempo, como pueden ser bucles.

Extrude
MAX > Modelado

Modificador de geometría utilizado en MAX para crear extrusiones. Toma un modelo 2D realizado generalmente con las herramientas "shapes" y los extruye, es decir, le aplica una profundidad en cualquiera de los ejes para que posea las tres dimensiones.

Extrusión
MAX > Modelado

Método de creación de modelado tridimensional que se basa en añadir el eje de profundidad a una superficie 2D. Por ejemplo, si tenemos un texto 2D, podemos convertirlo en 3D si toda la superficie la repetimos en profundidad y añadimos polígonos que cierren su superficie.

FFD
MAX > Modelado

Modificador de geometría utilizado en MAX para crear modificaciones en un objeto 3D. Presenta una estructura en forma de caja que rodea al objeto y que puede ser de 2 x2, 3x3 o 4x4 vértices por cara. Cuando se mueve un vértice de esta exoestructura, la superficie más próxima del objeto se trasladad en el mismo sentido y así se produce la modificación de esta.

FOV
MAX > Cámaras

También conocido como Anchura del campo visual. Mide el campo visual que abarca la cámara medido en grados. Normalmente cuanto mayor es el objetivo de la cámara, menor es el FOV.

Frames
Director > General

Es un concepto fundamental en Director ya que es cada uno de los fotogramas que componen la película. Por tanto, se puede considerar la unidad fundamental de este software, tanto en tiempo como en contenido. Cada frame pueden contener todo tipo de contenido multimedia independientemente así como código de programación.

Geometry quality
MAX > Exportación

Permite definir la calidad de la exportación de la geometría de los objetos. 100 equivale a la calidad que tienen en MAX, un valor menor supone bajar en ese tanto por ciento la cantidad de polígonos y por lo tanto su detalle, aunque ocupará menos memoria.

Gravity
Director > Sistemas de partículas

Es una característica física de los sistemas de partículas que permite simular la gravedad. Sin esta variable, sería muy difícil simular por ejemplo, una fuente de agua o una cascada en un río ya que necesitamos que las partículas tengan un movimiento propio y que además este movimiento se vea afectado por la fuerza gravitatoria. Gravity se define por un vector 3D, por lo que también podemos definir gravedad cero, negativa, etc.

Grid
MAX > Modelado

Rejilla de ayuda a la edición situada en altura 0 en cada uno de los visores. Puede servir de referencia o junto con snap toggle para la edición con ajuste a rejilla, favorecer la edición con precisión de objetos tanto 2D como 3D. Se puede visualizar o no pulsando la tecla G y se configura desde la pestaña Home Grid que aparece con el comando Grid and Snap Settings del menú Tools.

Grupo Maps
MAX > Superficies
Es un grupo de controles que se incluyen en la definición de materiales en el editor de materiales. Permite la inclusión de texturas como parte de la definición de distintos aspectos del material como puede ser el color difuso, la refracción o la reflexión.

Hither
Director > Cámaras
Mirar Plano trasero de recorte

Interacción externa
Director > Interacción
Consideramos interacción externa la que se produce entre la aplicación y el usuario como consecuencia de un diálogo establecido entre estas dos entidades a través del interfaz.

Interacción interna
Director > Interacción
La interacción interna es la que se produce entre los elementos que componen la aplicación. Dado que estamos trabajando en un mundo tridimensional y en algunos casos con un simulador físico, puede darse el caso que un elemento interno interacciones con otro sin la intervención directa del usuario.

Interactividad
Director > Interacción
Concepto que supone la comunicación entre una aplicación y un usuario lo cual supone un intercambio de comunicación entre hombre y máquina.

Interfaz
Director > Interacción
Es el medio para realizar una interactividad entre un usuario y una aplicación. Normalmente nos referimos como interfaz de usuario a los elementos que presenta el software con objeto de que el usuario pueda aportar y recibir información del software.

Keyframeplayer
Director > Animación
Modificador asociado a un objeto 3D que permite definir su animación fotograma a fotograma. Contiene propiedades y procedimientos para ejecutar y controlar este tipo de animación.

Keypressed
Director > Interacción
Es un procedimiento del objeto Key de Director que devuelva la última tecla pulsada por el usuario. Una propiedad directamente relaciona es “the keypressed” que nos da directamente este mismo valor.

Lathe
MAX > Modelado
Modificador de geometría utilizado en MAX para crear geometría por revolución. Ver revolución.

Lens
MAX > Cámaras
Ver Objetivo
Lifetime
Director > Sistemas de partículas
Determina el tiempo en el que las partículas realizan su recorrido. Por lo tanto, si le damos un valor alto a este parámetro, el efecto es que el sistema se alarga. Esto se traduce en que se puede modificar todo el tamaño del sistema de partículas cambiando esta variable.

Lingo
Director > Programación
Lingo es el lenguaje de programación de Director. Integra funciones para manipular contenido multimedia (como texto, imagen, sonido y video digital) y su interacción. Se considera un lenguaje de alto nivel de autor y para aplicaciones de este tipo puede ser una alternativa a otros lenguajes con C++ o Java.

Lista de modificadores
MAX > Modelado
Es un sistema que incorpora MAX para la aplicación de modificadores de distinto tipo y que se acumulan en apilándose. Esto visualiza una secuencia de aplicación temporal de los modificadores y además permite la edición de estas aplicaciones, con lo que podemos borrar un modificador de la lista y el objeto receptor volverá a su estado anterior.

Loft
MAX > Modelado
Método de construcción 3D. Mirar solevado.

Lustre, glosiness
MAX > Superficies
Parámetro de los materiales estándar de MAX que controla el difuminado del brillo, simulará por tanto, superficies más pulidas o más mates. Existe una utilidad que visualiza gráficamente este valor.

Luz ambiental
MAX > Iluminación
En realidad no se trata de un tipo de luz, sino de un color que se le aplica a la escena final en tiempo de renderizado y que la colorea. Se utiliza para crear distintos ambientes como por ejemplo, en un fondo marino pero ha de utilizarse con mucha cautela, ya que los efectos incumben a toda la escena 3D.

Luz ambiental, Ambient Light
Director > Iluminación
Es un tipo de luz de Director. Es la única luz que existe siempre en toda escena de Shockwave 3D. No tiene una fuente de luz determinada, por lo que es plana. No aporta profundidad a los objetos 3D por lo que normalmente su puede usar para tintar la escena y crear distintas atmósferas.

Luz con objetivo
MAX > Iluminación
Luz con Target. Mirar Target.

Luz direccional
MAX > Iluminación
Es un tipo de luz de MAX que permite simular una luz tipo focal pero de sección cilíndrica. Pueden ser libres o con target. Este tipo de luz permite simular focos en un escenario, etc.

Luz direccional, Directional Light
Director > Iluminación
Este tipo de luz de Director se utiliza para iluminar la totalidad de la escena y trata de imitar la luz solar mediante un gran foco colocado a cierta distancia. Añade profundidad general a los objetos. Se puede modificar en color y rotación.

Luz libre
MAX > Iluminación
En MAX se denominan así a las luces que carecen de target.

Luz focal, Spot Light
Director > Iluminación
Estas luces de Director actúan como un foco, es decir emiten luz desde un determinado punto y los rayos de luz se encuentran circunscritos a un cilindro luminoso. Son luces complejas que se deben utilizar con mesura ya que consumen mucho cálculo del procesador.

Luces omnidireccionales
MAX > Iluminación
Ver Omni.

Luz puntual, Point Light
Director > Iluminación
Este tipo de luz de Director, la emite en todas las direcciones a partir de un punto en concreto como una bombilla. Se suele utilizar para ambientaciones de interiores o resaltar ciertos objetos.

Luz Spot
MAX > Iluminación
Es un tipo de luz de MAX que permite simular una luz tipo focal pero de sección cónica. Pueden ser libres o con target. Este tipo de luz permite simular las luces de lámparas, farolas, etc.

Malla
MAX > Modelado
Superficie tridimensional creado a partir de los vértices que forman polígonos. Existen diversos sistemas para crearlos pero todos tienen como resultado la estructura basada en vértices, aristas, caras y polígonos. Permite el modelado de superficies complejas como por ejemplo cuerpos de personajes y animales, etc.

Mapa bump
MAX > Superficies
Mirar mapa de relieve

Mapa de reflejos
Director > Texturizado
Son imágenes que se utilizan en texturas complejas. Estas imágenes son a su vez texturas que crean la ilusión de reflejar el entorno. Esta técnica ahorra mucho tiempo de cálculo al sistema de representación, pero evidentemente es irreal. No tiene la capacidad de calcular los reflejos del entorno en tiempo real. Por lo tanto sirve para crear la ilusión de tener en nuestro mundo 3D elementos de metal como oro, plata,… pero por ejemplo, no podemos crear espejos en tiempo real con esta técnica.

Mapas de relieve, bump map
MAX > Superficies
Son imágenes que utiliza el representador de MAX para añadir protuberancias a las superficies de los objetos en los que se aplica. En realidad, no modifican la geometría de los objetos, solo lo aparentan. Por ello, los cambios observables no pueden ser de mucha entidad y se emplean por ejemplo, para que una esfera parezca con rugosidades como la piel de una naranja, o una pared aparezca estucada.

Mapas de sombras
MAX > Iluminación
Texturas de tipo fotográfico que se utilizan para que una fuente de luz al atravesarlas produzcan distintos efectos sobre una geometría. Estas tipo de sombras se suelen llamar sombras arrojadas. Sería el mismo efecto que las sombras chinescas.

Mapas de transparencia
Director > Texturizado
Ver canal alpha

Mapeado
MAX > Superficies
Proceso por el que se controla la disposición de una superficie texturizada en un objeto controlando la forma de "envolverlo", su escala, rotación y otras variables que influyen de manera crítica en la forma en que se representa.

Material/Map Browser
MAX > Superficies
Es una ventana a la que podemos acceder a partir del Editor de materiales y cuya función es la de gestionar los materiales y las librerías en los que se colecciona. A partir de esta utilidad podemos acceder a las memorias secundarias para abrir, guardar, cambiar el nombre, … ya sea para un materiales en particular o una librería de materiales.

Material Mezcla, Blend
MAX > Superficies
Tipo de material de MAX no estándar. Presenta dos submateriales y una máscara, ya que blend se utiliza para crear materiales que sean mezcla de otros dos pero con la mezcla dirigida por la máscara que suele ser un bitmap. Por ejemplo, si queremos crear un materiales que presente césped entre unas piedras planas para realizar una senda rústica, podemos crear estos dos submateriales independientemente y luego utilizar una máscara que describa la posición de las piedras para entre los huecos, visualizar el césped.

Material Shellac
MAX > Superficies
Tipo de material de MAX no estándar. Presenta dos submateriales, ya que shellac se utiliza para crear materiales complejos de dos o más capas. Por ejemplo, la madera barnizada o las superficies metálicas de los automóviles.

Materiales procedimentales
MAX > Superficies
Son un tipo de materiales que presentan una textura de naturaleza no bitmap. Por tanto, no se aplican imágenes fotográficas. Estas texturas se calculan por módulos de programación y por lo tanto son intrínsecamente escalables. Su desventaja es que no son fotorealistas. Se suelen utilizar para texturas veteadas como la madera o el mármol.

MAX
MAX > General
Formato de archivo nativo de 3d MAX. Permite guardar escenas tridimensionales que incluyen geometría, luces, cámaras, texturas y animaciones.

MeshSmooth
MAX > Modelado
Modificador de geometría que permite suavizar la superficie de un objeto 3D aplicando uno de estos tres métodos: nurms, classic o quad. La cantidad de suavizado aplicada será variable. No debemos utilizar valores muy altos, ya que retardará mucho la representación del objeto debido al alto consumo de cálculo computacional.

ModelUnderLoc
Director > Interacción
Es un procedimiento, o sea, una función creada ya en Lingo que podemos utilizar para relacionar un clic en la pantalla por parte de un usuario con un modelo 3D. Su funcionamiento es bastante complejo pero lo utilizaremos para controlar el objeto seleccionado por el usuario con un puntero 2D como es el ratón con un objeto que se encuentra en un espacio 3D.

Modo de renderizado, renderstyle
Director > Texturizado
Propiedad del shader standard. Permite elegir entre modos distintos de representar un modelo 3D. Los valores que admite son fill, wire y point. Fill es el valor por defecto y significa que el modelo tendrá una superficie normal. Wire hace que se vea en modo alámbrico. Point tiene el efecto de que solo se pueden ver los puntos que representan los vértices de la malla.

Motion
Director > Animación
Son animaciones internas de la propia escena que se han realizado en MAX e importando con el fichero w3d.

MouseLoc
Director > Interacción
Es una propiedad del objeto Mouse de Director que contiene las coordenadas X e Y del punto de la pantalla donde se encuentra el ratón en ese momento.

MOV
Software
Es un formato propiedad de Apple. Contiene películas QuickTime y es multiplataforma, esto quiere decir que se pueden visualizar en ordenadores con distintos sistemas operativos. En la actualidad se pueden realizar películas interactivas 3D y aplicaciones de realidad virtual.

Newgroup
Director > Programación
Sentencia de Lingo que permite crear un grupo, es decir agrupaciones de nodos 3D para su posterior utilización conjunta. Necesita un nombre como parámetro.

Newlight
Director > Iluminación
Método que permite crear una luz mediante programación en código Lingo o JavaScript. Utiliza dos parámetros: el nombre de la nueva luz y el tipo de esta, que puede ser direccional (direccional), punto (point) o foco (spot).

Newtexture()
Director > Texturizado
Es una función de la propia escena Shockwave 3D. Esta función permite introducir imágenes externas dentro de la escena. Se supone que después utilizaremos estas imágenes para texturizar con ellas los modelos 3D que existan en dicha escena.

Ngon
MAX > Modelado
Primitiva de tipo Shape, es decir, 2D, que permite la creación de polígonos de distintas cantidades de lados. Por ejemplo, podemos crear un triángulo, un pentágono, etc.

Nivel de opacidad, blend
Director > Texturizado
Propiedad del shader standard. Mide la cantidad de transparencia de un modelo 3D. No tiene efecto si la propiedad transparent no es verdadera. Puede tener un valor entre 0 (totalmente transparente, invisible) y 100 (totalmente opaco).

Noise
MAX > Modelado
Modificador de geometría utilizado en MAX para crear superficie rugosas aleatorias mediante la generación de cambios en los polígonos. La cantidad de ruido aplicada puede variar de 0 a 100, además de elegir distintos modos de generación. Este modificador, puede hacer por ejemplo que una esfera de superficie perfecta se convierta en una piedra a modo de canto rodado más o menos pulido.

Multi/Subobject material
MAX > Texturizado
Tipo de material que está compuesto por varios materiales a su vez. Por ejemplo, podemos asignar este tipo de materiales a un objeto poligonal y a ciertos polígonos le asignaremos un submaterial y a otros otro submaterial. Es una forma eficiente de agrupar materiales. Por ejemplo, si tenemos que asignar materiales a un personaje creado con un edit Poly, podemos asignar un material a la cabeza y otro distinto a los pies.

Nivel especular
MAX > Superficies
Parámetro de los materiales estándar de MAX que controla la intensidad del brillo. Existe una utilidad que visualiza gráficamente este valor.

Nurms
MAX > Modelado
Es una técnica de subdivisión de superficie usada para malla con pocos polígonos para controla la forma del suavizado de superficies. Es el acrónimo de Non-Uniform Rational Mesh Smooth.

Objetivo, Lens
MAX > Cámaras
Es un parámetro de las cámaras que permite ajustar el valor de la distancia focal.

Objeto booléano
MAX > Modelado
Es un objeto creado mediante la combinación de dos o más objetos. Esta combinación se crea realizando operaciones booleanas entre los objetos. Estas operaciones booleanas son tres: la unión, la sustracción y la intersección. La unión y la intersección son conmutativas, es decir, tienen el mismo resultado sea cual sea el orden de elección de los objetos operandos, pero la sustracción no es conmutativa por lo que el resultado depende del orden en que intervienen los objetos operandos.

Objetos compuestos, Compound Object
MAX > Modelado
En 3d Studio MAX, se conoce como objetos compuestos a aquellos que han sido creados siguiendo distintos métodos de construcción que utilizan otros objetos como operandos. Por ejemplo, los objetos boolenanos, los solevados, los de revolución, etc son objetos compuestos. Unas operaciones de construcción son reversibles (permiten recuperar los objetos base y deshacer la operación) y otros no.

Omni
MAX > Iluminación
Tipo de iluminación de MAX que emite luz en todas las direcciones con lo que se suele utilizar para crear “bombillas”.

Opacidad
MAX > Superficies
La opacidad es una propiedad óptica de la material que se presenta cuando no deja pasar la luz en proporción apreciable. Un material puede ser más o menos opaco y esta propiedad es simulada en MAX mediante un parámetro de los materiales estándar que viene dado en modo porcentual.

Orthoheight
Director > Cámaras
Es un parámetro que controla las unidades verticales del plano de vista en una proyección de tipo ortográfica. Se puede comparar con el parámetro Fieldofview de una proyección de tipo perspectiva.

Partícula, particle:
Director > Sistemas de partículas
Son los elementos que componen el sistema. Son elemento 2D, compuesto por un plano texturizado o coloreado que normalmente es de un tamaño relativo muy pequeño. El sistema de partículas es un elementos 3D por el gran número de partículas 2D que genera y que si que suelen tener un movimiento en 3D.

Phisique
MAX > Animación
Modificador de Character Studio utilizado en MAX para crear una conexión entre parte de la geometría de un modelo y una parte de biped, es decir, de un esqueleto bípedo. Esta modificador se configura, por tanto, después de realizada la configuración de biped, y supone que cuando se mueva un hueso de bidep, la malla poligonal que representa el cuerpo, lo sigua y por tanto, que se pueda simular la animación de personajes bípedos.

Plano delantero de recorte
Director > Cámaras
Plano vertical con respecto a la cámara. Sirve para limitar lo que representa la cámara. Todo lo que se encuentre más alejado del plano delantero de recorte no se representará.

Plano trasero de recorte
Director > Cámaras
Plano vertical con respecto a la cámara. Sirve para limitar lo que representa la cámara. Todo lo que se encuentre más cercano del plano trasero de recorte no se representará.

Planos de recorte
MAX > Cámaras
Son dos planos. Uno de proximidad y otro de lejanía a la cámara. Estos planos permite excluir objetos que la cámara debería excluir por ejemplo por estar demasiado cerca o lejos de la cámara como para que su visión sea significativa.

Play Animation
Director > Interacción
Es un comportamiento que permite animar un objeto 3D configurando varios parámetros como el estado inicial, el disparador de comienzo, etc.

Pointat
Director > Cámaras
Función que permite orientar una cámara hacia un punto en concreto del espacio 3D. Esta función admite dos parámetros: vector de orientación hacia delante y el de orientación hacia arriba.

Polígono editable, Editable Poly
MAX > Modelado
Es una malla que forma una geometría tridimensional y permite editarlo por niveles de forma escalar. Por ejemplo, si elegimos el subobjeto polígono, aparecen los comandos necesarios para editar solo polígonos. También podemos manipular independientemente los vértices, las aristas, las caras, los polígonos y el elementos en su conjunto.

Previsualización, preview
MAX > Exportación
Función del módulo exportador desde Max a Director que permite visualizar cómo va a observarse la escena tridimensional antes de haberla exportado, por tanto, permite ganar tiempo a la hora de editar el fichero a exportar si tenemos algo que modificar.

Primitiva
MAX > Modelado
Modelos geométricos en tres dimensiones que se pueden crear en MAX simplemente definiendo un origen y unos parámetros imprescindibles, por ello también se les conoce como objetos paramétricos. Por ejemplo, una esfera solo necesita un centro y un radio. Las primitivas son como los componentes iniciales de una escena 3D y en MAX son objetos primitivos la esfera, la “caja”, la pirámide, el cilindro, el plano, etc.

Programación orientada a objetos, POO
Director > Programación
Este es el modelo de programación en la actualidad. Se basa en la creación de objetos que no son más que agrupaciones de variables y sus valores y unos procedimientos que sirven para

modificar estos valores. Las consecuencias son la fácil reutilización del código y por tanto, el elevado rendimiento del trabajo en grupo entre otras ventajas.

Projector
Director > Publicación
Es uno de las posibles opciones para publicar un trabajo desde Director y consiste en la creación de un archivo autoejecutable de extensión .exe. Esto quiere decir, que a la escena tridimensional e interactiva se le añade un proyector, es decir, un software run-time que se ejecuta de manera autónoma al abrir el fichero y por lo tanto no necesita de la instalación de otro software para su uso. Esta propiedad tiene una contraprestación, y es el aumento del tamaño del archivo en más de 3 Mbytes.

Property
Director > Programación
Esta palabra reservada es utilizada en Lingo para crear una variable local en un behaviour. En contraposición con global que establece variables accesibles desde toda la aplicación, property crea una variable que solo tiene sentido en el comportamiento donde se ha creado. No hace falta que el autor destruya esta variable explicitamente.

Proyección
Director > Cámaras
Método matemático para representar una escena 3D en un plano 2D. Existen dos tipos de proyecciones: perspectiva y ortográfica.

Proyección isométrica
Director > Cámaras
Se trata de una proyección ortográfica donde la cámara se sitúa en un ángulo de 45º con respecto al modelo.

Proyección ortográfica
Director > Cámaras
Produce que el tamaño aparente de los objetos con la distancia no se represente y por tanto, no muestra la perspectiva real.

Proyección perspectiva
Director > Cámaras
Tipo de proyección por defecto que representa la perspectiva real de la vista.

Publish Settings
Director > Publicación
Es un comando de Director que encontramos en el menú Archivo y que nos permite configurar los llamados parámetros de publicación de Director. Normalmente se conocen como opciones de publicación y nos sirven para que Director exporte nuestro trabajo a distintos formatos como pueden ser un ejecutable .exe, un archivo shockwave .dcr y otros.

Rangos de entorno
MAX > Cámaras
Es un parámetro que permite definir la distancia de los efectos atmosféricos si los hubiera en la escena 3D.

Raytrace
MAX > Texturizado
Es la aplicación en MAX del raytracing o trazado de rayos. Este es un algoritmo para la creación de imágenes 3D que permite mediante el análisis de rayos que se trazan desde la cámara de visualización hacia la escena y el cáculo de las intersecciones de estos con los objetos, la generación de escenas de gran realismo y simulación de efectos de reflexión y refracción complejos, así como la generación de sombras generadas por los propios objetos de la escena 3D. En la actualidad este algoritmo ha sido superado aunque sigue siendo la base de muchos sistemas de representación realista en 3D.

Reflectionmap
Director > Texturizado
Ver mapas de reflejos

Reflexión
MAX > Superficies
Es un fenómeno por el cual un rayo de luz, al incidir sobre una superficie, es reflejado. En MAX, este fenómeno se utiliza para simular materiales como espejos, cromados, etc.

Refracción
MAX > Superficies
Es el cambio de dirección que experimenta una onda al pasar de un medio material a otro. Por ejemplo, si se sumerge un lápiz en un vaso con agua parece quebrado En MAX, este fenómeno se utiliza para visualizar un cambio de material simulando la refracción natural, por ejemplo, utiliza este hecho para simular cristales, agua, etc. Se pueden simular utilizando incluso algunas texturas o mapas de refracción.

Render Setup
MAX > Renderizado
Orden de 3d MAX para la configuración del render o representación de la escena. Aquí, el usuario puede elegir el nombre, tamaño y tipo de archivo de salida, así como el módulo que será el encargado de realizar esta representación y que pueden ser adquiridos independientemente del programa MAX.

Renderstyle
Director > Texturizado
Ver modo de renderizado

Representación, render
MAX > Renderizado
Proceso por el cual, los programas de creación de escena tridimensionales calculan y muestran el resultado de una escena en formatos de imagen o animación como videos, etc. Este proceso requiere normalmente grandes cantidades de cálculos de procesadores por lo que pueden ser críticos dado que debe de calcular mediante algoritmos muy complejos aspectos como la proyección de sombras, el suavizado de los polígonos de la geometría de objetos, la iluminación fotorealista, etc.

Revolución
MAX > Modelado
Es un método de creación de objetos 3D utilizando un perfil o línea 2D. Este perfil se gira normalmente sobre el eje Y y se van creado copias del perfil cada cierto nº de grados y estas constituyen el soporte sobre el que se crean los polígonos que forman la malla del objeto resultante. Es una realización infográfica del método utilizado por los ceramistas, que hacen girar un torno para crear objetos como cuencos, jarrones, etc.

Sampling intervalo
MAX > Exportación
Permite definir la correspondencia entre los fotogramas exportados y los originales en el archivo de MAX en una animación.

Score
Director > General
Ventana que sirve para gestionar de forma visual los sprites, o sea, las apariciones en la película de los distintos componentes que la forman. Esto lo realizan mediante la muestra de una línea de tiempo donde se representan los fotogramas de la película. También permite la asignación de módulos de programación o behaviors tanto a fotogramas como a sprites.

Shader
Director > Texturizado
Es una propiedad de un modelo 3D que define los efectos simulados de la luz que se refleja en la superficie de ese modelo. Por ejemplo si es un metal, si es agua o cristal, etc. Es una forma de dibujar la superficie de un modelo. En realidad Shader es otra forma de referirse a el primer elemento de la lista de shaders de un modelo 3D, es decir, que corresponde a shaderlist[1].

Shader.Flat
Director > Texturizado
Propiedad del shader standard. Tiene valor verdadero o falso, lo que significa que las caras que componen el modelo 3D se sobrearan suavemente o no. Si toma un valor falso se podrá apreciar la geometría del modelo de forma más basta.

ShaderList
Director > Texturizado
Lista de shaders de un modelo 3D. El número de shaders disponibles para un modelo 3D depende del número de mallas que tenga el modelo. Por ejemplo un plano, como tiene dos lados tendrá dos shaders, un cubo tendrá 6, etc.

Shader.Transparent
Director > Texturizado
Propiedad del shader standard. Tiene un valor booleano, es decir, verdadero o falso. El valor verdadero significa que el modelo es semitransparente. Debemos de utilizar esta propiedad junto con otras como blend y model.visibility.

Shininess
Director > Texturizado
Ver Tamaño del brillo.

Sistemas de partículas
Director > Sistemas de partículas
Conjunto de partículas que partiendo de un emisor que puede tomar varias formas, permanecen visibles un tiempo hasta desaparecer. El movimiento de las partículas suele ser 3D, por eso el sistema entero de partículas se considera un elemento tridimensional. Es un recurso infográfico utilizado por los programas de generación de escenas 3D para imitar distintos efectos especiales como resplandores, halos,… fenómenos atmosféricos como lluvia, niebla,… y otros.

Sizerange
Director > Sistemas de partículas
Controla el tamaño de cada partícula mediante dos propiedades: start y end, que se pueden configurar para definir la opacidad al principio y al final de la vida de una partícula. Entre uno y otro momento, el sistema interpolará el tamaño de las partículas.

SDS
MAX > Exportación
Este modificador permite añadir mayor detalle a la geometría de los objetos subdividiendo los polígonos, lo que aumenta su calidad y lo que ocupa en memoria. Esta opción se puede elegir para ser exportada a Director.

Shapes
MAX > Modelado
Categoría de edición de objetos 2D en MAX. Dentro de esta pestaña encontramos objetos como splines, elipse, rectángulo, texto, etc.

Shockwave 3D
Director > General
Es un pluging para navegadores web que permite la reproducción de gran número de contenidos multimedia además de un motor 3D, lo que hace de él el más potente de su segmento, aunque no ha tenido tanto éxito como su hermano menor, el pluging de Flash.

Skew
MAX > Modelado
Modificador de geometría utilizado en MAX para crear un sesgo, es decir, una alteración del ángulo sobre uno de los ejes de un objeto 3D. El resultado es que el objeto parece "inclinado" hacia ese eje. Es muy utilizado en sistema 2D para por ejemplo, hacer un rombo tumbado a partir de un rectángulo. En MAX se utiliza con los mismo efectos pero con modelos 3D.

Smooth
MAX > Modelado
Significa suave y se usa como un modificador que permite suavizar la superficie de un objeto 3D subdividiendo los polígonos que componen su superficie en polígonos más pequeños. El problema es que este suavizado supone un mayor cálculo a la hora de la representación de estas superficies.

Snaps Toggle 2D, 3D
MAX > Modelado
Función de MAX que permite la edición en 2D o 3D ajustada a la cuadrícula o a otros elementos como vértices creados con anterioridad. Es pues, una utilidad que se utiliza sobre todo para editar modelos con precisión.

Solevación
MAX > Modelado
Método de construcción de objetos tridemsionales a partir de dos curvas: la trayectoria y el perfil. El perfil se repetirá con una frecuencia dada siguiendo la curva definida como trayectoria y a partir de esta estructura se crea la malla que compone el objeto. Por ejemplo, un cilindro se puede considerar un elementos solevado con perfil circular y trayectoria recta. De esta manera es relativamente sencillo crear cables, marcos de cuadro, etc.

Sombras, Shadows
MAX > Iluminación
Las sombras son zonas de la escena 3D que quedan en cierta oscuridad por que la luz queda obstaculizada de algún modo. Se producen siempre para la participación en la escena 3D de una fuente de luz. En MAX, podemos elegir que la fuente de luz produzca o no sombras, el color de estas y su calidad al representarlas.

Sombreadores
MAX > Superficies
El sombreador es el parámetro del material que lo define intrínsecamente, ya que a partir del tipo de sombreado se aplican las demás variables. Por ejemplo, si elegimos el modo de sombreado metal, las variables que aparecen en el editor de materiales cambian para mostrarnos solo aquellas que le incumben, como el color, el tipo de brillo, etc. Los sombreadores básicos de MAX son: Anisotropic (para crear los brillos lineales de los metales pulidos), Blinn (derivado del Phong pero más preciso), Metal (para metales no pulidos), Multi-Layer (como dos Anisotripic en uno para superficies los metales cubiertos por ceras como la superficies de los cochees), Oren-Nayar-Blinn (variación del Blinn con bordes difuminados), Phong (para suavizar caras), Strauss (para metales complejos) y Translucent Shader (para materiales translucidos).

Specular
Director > Iluminación
Es una propiedad de las luces point y spot que define el brillo de la luz. Solo puede tener dos valores: verdadero o falso. Produce que la luz muestre un brillo en la zona especular que le añade profundidad.

Spline
MAX > Modelado
Es una curva definida matemáticamente que se utiliza en los programas de modelado para crear perfiles de objetos 3D, caminos a recorrer por una cámara, etc. A partir de ella se crean otros objetos curvos con implicaciones en el modelado como las B-spline y las Nurbs.

Sprite
Director > General
El sprite es la aparición de un actor o "member" de la película. Esta aparición es el stage, dado que si no es así no se visualizará. Los distintos sprites o apariciones se gestionan en la ventana llamada Score.

Stage
Director > General
Ventana de Director que permite la visualización de la película. Por tanto, en el stage o escenario, se colocan todos los elementos que en un momento dado aparecen en escena. Puede ser 2D o 3D mediante el uso de Shockwave 3D.

Standard material
MAX > Texturizado
Tipo de material utiliza en MAX que tiene ciertas variables que se consideran standard. Esto no quiere decir que sea un material fácil de configurar, aunque si es verdad que es el material por defecto con que nos encontramos al abrir la utilidad del editor de materiales.

SWF
Director > General
Formato y extensión de fichero ejecutable Flash tanto en memoria secundaria como a través de Internet. Por tanto, es una película de Flash normalmente interactiva y ejecutable directamente a través del plugin de Flash que en la actualidad se encuentra instalado en un 98% de los ordenadores conectados a Internet, lo que proporciona a este tipo de archivo una gran universalidad.

Tamaño del brillo, shininess
Director > Texturizado
Propiedad del shader standard. Define el tamaño de la luz specular, es decir, el tamaño de la zona más iluminada del modelo 3D. En Director puede tener valores entre 0 y 100. Con un valor muy alto la zona más iluminada será muy pequeña pero muy nítida, lo que es perfecto para materiales metálicos. Si tiene un valor muy cercano al cero, será una zona grande pero sin brillo, adecuado para superficies mates.

Taper
MAX > Modelado
Modificador de geometría utilizado en MAX para crear un estrechamiento en un extremos cualquiera de un objeto 3D. El efecto es que el objeto se "afila", por lo que si por ejemplo, se le aplicamos a un cilindro, podemos llegar a tener un cono truncado o completo.

Target
MAX > Iluminación
Es el objetivo de una fuente de luz. MAX permite que cierto tipo de luces tengan o no target, es decir un punto tridimensional en la escena que sirve para orientar la fuente de luz.

Texture
Director > Sistemas de partículas
Es una variable del sistema de partículas que define la textura aplicada a cada partícula emitida y por tanto, se asocia a la visibilidad que presenta el sistema.

Texture
Director > Texturizado
Propiedad del shader standard. Esta característica permite asignar un bitmap a la superficie de un modelo 3D. Puede tomar su valor mediante Lingo como un objeto del archivo de Director o haberlo importado con el archivo shockwave 3D desde un programa de modelado externo.

Texturemode
Director > Texturizado

Es una función o método del shader. Permite definir la forma en que la textura "envuelve" el modelo 3D sobre el que se aplica el shader. Es decir, permite modificar en tiempo real y mediante programación en Lingo, el mapeo del modelo.

Texturetransform
Director > Texturizado
Es una función o método del shader. Permite definir transformaciones en la forma en que la textura se aplica sobre el modelo 3D sobre el que se aplica el shader. Por tanto, esto supone que por programación, podemos variar en tiempo real la forma en que se visualiza el modelo, lo cual abre la puerta a su aplicación en procesos de animación de texturas por el método de transformación.

Texturizado
MAX > Superficies
Proceso por el que se colocan texturas, es decir, superficies en las caras de los polígonos que forman la geometría de los objetos tridimensionales. El texturizado puede incluir definiciones parámetricas, o sea, mediante variables y/o utilizar imágenes importadas para crear estas superficies.

Toggle breakpoint
Director > Programación
También conocido como punto de ruptura. Se puede establecer un "breakpoint" en cualquier línea de un script y sirve para que el programa en ejecución se pare en este punto y poder estudiar así el valor de las variables en un determinado momento. Es pues, una herramienta para el análisis de la programación.

Toons
MAX > Exportación
Este es un modificador que permite representar la escena con una estética de dibujos animados, es decir, con los bordes de los objetos muy definidos y de color negro y el relleno con colores planos y sombreado apenas visible. En la exportación podemos elegir exportar este modificador.

Translate
Director > Sistemas de partículas
Es una característica física de los sistemas de partículas que permite trasladar el emisor en tiempo real y por medio de programación. Esto abre las puertas a la interacción y por ejemplo, se puede usar esta variable para crear efectos especiales interactivos en su posición. Translate se define por un vector 3D.

Travelling
MAX > Cámaras
Es un efecto cinematográfico normalmente realizado con una cámara con objetivo. Consiste en que mientras la cámara se modifica en posición, el punto al que se dirige permanece fijo. Por ejemplo, podemos mover la cámara alrededor de un objeto mientras su objetivo es siempre el propio objeto para conseguir una toma que lo muestra desde distintos ángulos.

Triggers
Director > Interacción
Ver Disparadores.
Twist
MAX > Modelado
Modificador de geometría utilizado en MAX para crear una torsión sobre sí mismo de un objeto 3D en cualquiera de los ejes x, y o z. El efecto es que el objeto se "retuerce" y se puede emplear para realizar objetos que tengan esta característica como cierto tipo de columnas, etc.

Ventana Code: Library
Director > Interacción

Es una ventana que sirve para acceder a las bibliotecas de Director. Tenemos en la pestaña Library y menú que presenta la opción 3D y esta las opciones de elegir entre las bibliotecas de Actions y Triggers.

Visibility
Director > Texturizado
Propiedad del modelo 3D. Tiene 4 posibles valores front, back, both y none. Front solo mostrará las caras más cercanas al espectador ocultando las que se encuentran detrás. Both, que mostrará todas las caras y que es un valor perfecto para representar objetos transparentes. Back solo presenta las caras posteriores del modelo 3D. None hace que el modelo sea invisible.

W3D
MAX > Exportación
Formato y extensión de fichero exportable de 3d MAX a Director para su inclusión en una escena Shockwave 3D. Permite incluir geometría, texturas, animación, iluminación, cámaras así como información general del archivo.

Weld Core
MAX > Modelado
Es una característica que se puede requerir en el proceso de realización de un objeto 3D mediante el método de revolución (ver revolución) de una línea perfil 2D. Weld Core lo que proporciona es que el centro del cuerpo resultante quede cerrado, de tal manera que no quede ningún hueco interno.

Wind
Director > Sistemas de partículas
Es una característica física de los sistemas de partículas que permite simular el viento. Con esta variable que podemos definir mediante un vector tridimensional cómo las partículas son arrastradas por una fuerza externa a su propio movimiento.

Yon
Director > Cámaras
Mirar “Plano delantero de recorte”

#burst
Director > Sistemas de partículas
Valor que puede tomar la variable emitter.mode (junto a stream) de un sistema de partículas y que hace que las partículas emitidas lo hagan todas a la vez y cada cierto tiempo de forma periódica, como si fueran explosiones.

#none
Director > Texturizado
Valor que puede tomar la variable texturemode (junto a wrapplar, wrapspherical, wrapcylindrical y reflection) del shader de un modelo en shockwave 3D. Este valor hace que la textura se aplique intacta a cada cara del modelo, por ejemplo en un cubo, la textura estaría repetida en cada una de sus 6 caras.

#orthographic
Director > Cámaras
Valor que puede tomar la variable projection (junto a perspective que es el valor por defecto) de una cámara en shockwave 3D. En este modo, la cámara hace que el tamaño aparente de los objetos no se modifique con la distancia. Se usa cuando se necesitan vistas muy precisas, así como en arquitectura, gráficos empresariales y simulaciones.

#particle
Director > Sistemas de partículas
Valor que puede tomar la función “newmodelresource” y que produce la creación de un nuevo sistema de partículas mediante programación.

#perspective
Director > Cámaras
Valor por defecto de la variable projection (junto a orthographic) de una cámara en shockwave 3D. En este modo, los objetos se ven como en la realidad, es dedir que su tamaño aparente se modifique con la distancia y por tanto se vean en perspectiva.

#reflection
Director > Texturizado
Valor que puede tomar la variable texturemode (junto a wrapplar, wrapspherical, wrapcylindrical y none) del shader de un modelo en shockwave 3D. Este valor hace que la textura se aplique a todas las caras del modelo a la vez, pero aunque el modelo se mueva, la textura no lo seguirá, lo que crea un efecto de reflejo.

#rgba4444
Director > Texturizado
Valor que puede tomar la variable renderformat (junto a rgba8888 y otros) de una textura de un shader. Este valor hace que la textura se represente con una profundidad de 12 bits para el color y 4 para los canales alpha.

#rgba8888
Director > Texturizado
Valor que puede tomar la variable renderformat (junto a rgba4444 y otros) de una textura de un shader. Este valor hace que la textura se represente con una profundidad de 24 bits para el color y 8 para los canales alpha. Es la calidad máxima que permite la representación 3D.

#stream
Director > Sistemas de partículas
Valor que puede tomar la variable emitter.mode (junto a burst) de un sistema de partículas y que hace que las partículas emitidas lo hagan en modo afluencia, es decir, como si fuera un caudal continuo.

#wire
Director > Texturizado
Valor que puede tomar la variable renderstyle (junto a fill y point) de un modelo en shockwave 3D y que hace que un modelo tridimensional se visualice en modelo alámbrico, es decir, que se ven solamente las aristas que lo componen.

#wrapcylindrical
Director > Texturizado
Valor que puede tomar la variable texturemode (junto a wrapplarnar, wrapspherical, reflection y none) del shader de un modelo en shockwave 3D. Este valor hace que la textura se aplique como a un cilindro, o sea, una vez envolviendo las caras centrales y otras dos para las caras superior e inferior.

#wrapplanar
Director > Texturizado
Valor que puede tomar la variable texturemode (junto a reflection, wrapspherical, wrapcylindrical y none) del shader de un modelo en shockwave 3D. Este valor hace que la textura se aplique como si el modelo fuera plano.

#wrapsherical
Director > Texturizado
Valor que puede tomar la variable texturemode (junto a wrapplar, refletion, wrapcylindrical y none) del shader de un modelo en shockwave 3D. Este valor hace que la textura se aplique a todas las caras del modelo a la vez, como si solo tuviera una cara y la rodease completamente.

ANEXO III:

GLOSARIO DE TÉRMINOS DE SECOND LIFE

Glosario de términos utilizados en Second Life.

No vamos a incluir aquí las palabras que son directamente la traducción exacta del inglés ya que no se trata de un diccionario español-inglés. Por lo tanto las palabras que tienen las mismas acepciones que en español no las introduciremos en este glosario temático.

Categorías empleadas para clasificar los términos:

Arte
Animación
Cámaras
Comunicación
Construcción
Empresa-Negocios
Educación
General
Iluminación
Integración Multimedia
Interacción
Lugares
Personas-Avatares
Programación
Publicación offline
Simulación física
Sistema de partículas
Superficies y texturas

AjaxLife
General
Cliente de Second Life alternativo al distribuido por Linden Labs y que tiene una mejora bastante significativa en la gestión del Inventario.

Align media texture, alinear textura multimedia
Superficies y texturas
Se utiliza en texturas especiales que se transforman en vídeos y que al realizar el cambio pueden tener problemas de alineación con respecto a la geometría que texturizan. Esta función alineará el vídeo (o la textura que lo contiene) con la geometría del modelo en cuestión.

ALL_SIDES
Superficies y texturas
Es un valor del lenguaje de programación LSL que se utiliza en distintas funciones y que equivale a “todas la caras del modelo”. Por supuesto que el sentido final del valor depende de lo que haga la función donde se integra.

Animated Textures
Superficies y texturas
Opción que permite visualizar la animación de texturas y por lo tanto que la aplicación de la función llSetTextureAnim() tenga efecto. Esta opción es booleanas, solo puede valer o verdadero o falso y se encuentra en el menú Advanced > Rendering.

Apariencia
Personas-Avatares
Esta característica de los avatares engloba todo lo que se refiere a la imagen de este. Modificar nuestro aspecto es fundamental de nuestras relaciones sociales dentro de Second Life por lo que existen muchas facilidades para cambiar y crear la apariencia de nuestro avatar.

Applications programming Interface, API
Simulación física
Es un interfaz de comunicación entre componentes de tipo software. Consiste en que un proceso (aplicación en ejecución) llama a funciones de biblioteca. Este mecanismo permite la abstracción en programación, es decir, que software de alto nivel utilice software de bajo de nivel.

Arrastrar, drag
Simulación física
Simula la flexibilidad de un objeto con respecto a su propio movimiento. Sus valores pueden ir de 0 (cambia constantemente si se mueve) hasta 10 (no cambia cuando se le mueve).

Arte fractal
Arte
Arte que utiliza los fractales, es decir estética geométrica que presenta una imagen que se repite a diferentes escalas. El término fue propuesto por Mandelbrot en 1975 para describir objetos con esta característica que curiosamente poseen también muchas estructuras naturales como el crecimiento de muchas plantas, los copos de nieves, etc.

Avatar
General
Es la imagen que los habitantes de SL muestran. Según la RAE es una representación gráfica de nuestra reencarnación o transformación y sirve para la identificación en las nuevas tecnologías de individuos reales.

AVI
Publicación offline
Es un formato de vídeo y audio propiedad de Microsoft® lanzado en 1992 por lo que está muy extendido. AVI son las siglas de Audio Video Interleave.

Basic Shaders
Iluminación
Opción que permite si nuestra tarjeta también lo permite, la visualización de resplandores y halos. Se encuentra en Edit > Preferences > Graphics > Custom.

Blank
Superficies y texturas
Botón de la ficha Texturas en la edición de objetos que permite la eliminación de cualquier textura y por tanto el modelo en cuestión mostrará el color asignado. Todos los modelos pueden o no tener asignada una textura, pero todos tienen un color asignado.

Bloque de construcción
Construcción
Cuando estamos creando un objeto, siempre (salvo en el caso de los sculp) nos estamos basando en un objeto primitivo. El bloque de construcción es la primitiva de partida que modificamos. Este bloque de construcción lo podemos variar en cualquier momento.

Blue Ocean Island
Programación
Isla para aprender a programar. Existe un grupo el Blue Ocean Scripters Group que forman una comunidad de programadores.

Body parts
Personas-Avatares
Cada una de las partes que componen la fisonomía del avatar y que podemos modificar a nuestro gusto como pueden ser las piernas, la nariz, etc.

Build
Construcción
Botón de la interfaz que aparece si estamos en una parcela que nos permite crear y editar objetos y que una vez pulsado nos presentará una ventana con varias pestañas para realizar esta tarea.

Building Block Type
Construcción
El tipo de bloque de construcción define el primitivo básico de un modelo tridimensional. Este bloque de construcción es muy importante porque cada uno de ellos permite aplicar unos modificadores u otros al modelo. Por ejemplo, si queremos realizar una cabeza es conviene utilizar el bloque de construcción esfera en vez del cono. Este parámetro se puede modificar en cualquier momento de la construcción del objeto.

Bumpiness, irregularidad
Superficies y texturas
Característica de las superficies en Second Life que permite visualizar pequeñas irregularidades y por tanto representarlas con mayor realismo. No se pueden crear sino que se tiene que elegir de entre 16 opciones.

Cámara
General
Una cámara presenta una determinada vista del mundo que lo rodea. La cámara por defecto se encuentra situada ligeramente encima y atrás de la cabeza de nuestro avatar por eso lo vemos a él mismo y no desde el punto de vista de los ojos del avatar, aunque existen funciones para poderla gobernar de otros modos y por tanto, tener otras vistas.

Cámara por defecto
Cámaras
Es la cámara más básica y la que tenemos si no la cambiamos por otra desde el principio. Se sitúa detrás y un poco por encima de la cabeza de nuestro avatar, por lo que parte de nuestra representación es visible en pantalla.

Camping
Empresa-Negocios
Actividad para ganar unos L$ por el mero hecho de estar en un determinado lugar. Sería como cobrar por ser público, ya que esto genera tráfico en este lugar y aumenta las estadísticas con fines publicitarios.

Camtasia
Arte
Software para la captura de imágenes de pantalla y sonido que produce un vídeo. Muy utilizado en Second Life y otros interfaces para la creación de videotutoriales. Fraps es otra alternativa.

Canal alpha, máscara alpha
Superficies y texturas
Es un canal que se añade a una imagen como si fuera un color más, pero en este caso define grados de transparencia. Es decir, mediante un canal alpha podemos pintar cuanto de opaco es una superficie con colores en escala de grises. Estos canales se integran en una imagen que después exportamos a Second Life y que este utiliza para texturizar objetos y que tendrán distintas intensidades de opacidad correspondiéndose con el canal alpha.

Casting
Programación
Práctica de programación por el que una variable de un determinado tipo se convierte en otro tipo. Por ejemplo, si tenemos una variable de tipo numérico y su valor lo queremos mostrar por pantalla y la función para hacer solo admite caracteres, tendremos que hacer casting para convertir el número en caracteres.

Castronova, Edward
Personas-Avatares
profesor de Economía y Telecomunicaciones en la Universidad de Indiana. Se hizo famoso al publicar “Virtual Worlds: A First-Hand Account of Market and Society on the Cyberian Frontier” en diciembre de 2001, que resultó muy contestado por la comunidad de economistas más reaccionaria a las economías virtuales y donde describe las posibilidades reales de las economías derivadas de los metaversos.

Chat
Comunicación
Sistema de comunicación escrito y público que requiere cercanía. Si escribimos algo en el chat cualquiera que este cerca (hasta 20 mts.) lo leerá. Además de “hablar”, en el chat podemos gritar y en este modo nos leerán hasta a 100 mts. a la redonda.

Chat de voz
Comunicación
Es un chat donde se utiliza un micrófono y unos altavoces para mantener una conversación hablada con los avatares que estén cerca. Se trata de una interacción síncrona entre todos los avatares a nuestro alrededor.

Collision start, collision end
Simulación física
Son disparadores de los eventos de comienzo y finalización de una colisión entre objetos en Second Life. Los podemos utilizar en LSL para programar lo que tiene que suceder cuando dos objetos colisionan. Por ejemplo, podríamos programar un juego si programamos lo que sucede cuando un avatar hace colisionar su espada contra la espada de otro avatar.

Configuración del entorno, enviroments settings
Iluminación
Submenú de Second Life en el menú Mundo que permite controlar los cambios atmosféricos y luminosos de nuestro entorno. Permite elegir el momento del día deseado entre los valores amanecer, mediodía, atardecer, medianoche o volver a los valores por defecto. También permite editar el entorno.

Compilador
Programación
Un compilador es un programa que traduce un programa escrito en un lenguaje de programación (normalmente de alto nivel) a otro lenguaje que será directamente interpretable por el procesador.

Covenant, contrato
General
Se refiere a todos los términos en los cuales uno compra un terreno, ya sea en el continente de Linden Labs o en un sim privado. En este contrato aparecen las condiciones de utilización y comportamiento e incluso pueden aparecer condiciones referidas a las arquitecturas así como las posibles razones para la confiscación de los terrenos.

Creador, creator
Construcción
Usuario que crea un objeto en Second Life. Este control permite “firmar” nuestros modelos y por tanto tener derechos de propiedad intelectual. Se utiliza desde el control “Creator” de la ficha General en la edición del objeto. También presenta un botón para acceder al perfil del creador que no tiene por que coincidir con el propietario.

Cuenta Gratuita
Empresa-Negocios
Cuenta por defecto que se adquiere al entrar en SL. No supone desembolso alguno.

Cuenta Premium
Empresa-Negocios
Cuenta que se obtiene por 9 dólares (aunque esto varía con el tiempo) y que nos proporciona una serie de ventajas como la de comprar terrenos y otras.

Customize speech gestures
Integración Multimedia
Relación de sonidos anexos a gestos habituales que normalmente se utilizan en el chat de voz para aumentar el realismo o enfatizar nuestro discurso. También se suele usar junto con la función de sincronización de labios.

Decadencia, falloff
Iluminación
Una de las variables que definen la luz que se puede implementar en la pestaña “Carácter” de la edición de objetos. Decadencia o falloff indica la cantidad de difuminación de iluminación en los bordes. Proporciona realismo ya que el falloff de las luces reales es considerable.

Default
Programación
Es el estado por defecto de todos los scripts. Es la “puerta de entrada” de los scripts y por tanto, es obligatorio en todos ellos.

Design Studio
Empresa-Negocios
Empresa especializada en crear espacios en SL.

Dimple
Construcción
Modificador exclusivo para los prim de tipo Esfera. Viene dado por dos parámetros y produce una concavidad o convexidad dependiendo de estos valores.

Direct3D
Simulación física
Es la parte para tres dimensiones de DirectX, que es un API para la programación de gráficos y en especial de gráficos 3D. Propiedad de Microsoft, se utiliza en Windows y la videoconsola Xbox. Competidor de OpenGL.

Electric Sheep
Empresa-Negocios
Empresa especializada en crear espacios en SL.

Emisor
Sistema de partículas
Todo sistema de partículas necesita un prim emisor. El punto exacto desde el que nacen las partículas será el centro del emisor, por lo que al mover el emisor también reposicionamos el origen del sistema de partículas. El emisor puede ser visible o no.

Enlazar
Construcción
Es una operación por la que dos o más objetos se enlazan y con ello permiten que se pueda ejecutar operaciones de movimiento, escalado y rotación de forma conjunta. Esta operación permite realizar objetos muy complejos a partir de algunos más simples. El objeto enlazado tiene que tener un prim llamado raíz cuyo centro definirá el punto de transformación de todo el objeto enlazado. La operación de enlazar objetos se realiza desde el menú Tools > Link.

Enviroment Editor
Iluminación
Comando que se encuentra en el menú Mundo > configuración del entorno. Permite definir la hora exacta del día en que queremos situar nuestra escena, además de elegir el nivel de

nubosidad y el color y la claridad del agua. También presenta dos botones para definir todos los parámetros del agua y el cielo llamados agua avanzada y cielo avanzado.

Esculturas cinéticas
Integración Multimedia
Esculturas que se distinguen por el uso del movimiento en parte o la totalidad de la obra. El origen del movimiento puede ser mecánico mediante motores o por otros métodos como el viento, el agua, etc.

Estado
Programación
Un estado define una forma de responder a Second Life por parte de un objeto. Por ejemplo, Default es un estado por defecto de todos los scripts, pero podemos crear otros para definir que un vehículo está en marcha o parado, etc. Por lo tanto es un concepto muy vago que depende directamente del objeto al que esté asociado el script al que pertenece.

Evento
Programación
Es un suceso. Se utiliza para poder interactuar con los objetos. El mecanismo es que sucede algo y si el objeto ha sido programado para responder a este suceso lo hará mediante acciones. Por ejemplo, un objeto puede haber sido programado para que responda diciendo "buenos días" si alguien lo toca. El evento en este caso sería el "tocar" al objeto.

Explode
Sistema de partículas
Uno de los 5 posibles valores de PSYS_SRC_PATTERN que define el modo de emisión de las partículas en un sistema de estas. El modo explode es el más habitual y permite una flujo continuo de partículas.

Fantasma
Simulación física
En Second Life se denomina objeto fantasma aquel que puede ser atravesado por otros objetos o por los avatars sin problemas de colisión. Son objetos que tienen apariencia pero no son "físicos".

Flag
Sistema de partículas
Es un tipo de parámetro muy utilizado en llParticleSystem() debido al gran número de valores posibles que puede tener un determinado parámetro. Son variables tipo booleanas que pueden tomar solo dos valores: verdadero o falso. Se utilizan junto con las máscaras de flags que permiten la combinación de estas.

Flexi, objetos flexibles
Simulación física
Son objetos flexibles, conocidos en la jerga de Second Life como "flexis". Al definir un objeto como flexible se convierte automáticamente en un objeto fantasma.

Flip
Superficies y texturas
Cuando estamos texturizando un objeto, podemos pulsar este control para invertir de sentido la aplicación de la textura, es decir, voltea la textura aplicada a un objeto.

Focus, enfocar
Construcción
Una de las opciones que tenemos para controlar la vista de los objetos mientras los editamos, junto con órbita y zoom.

Fraps
Publicación offline
Software para la captura en forma de imágenes y vídeos de juegos y programas 3D. Es propiedad de la empresa Beepa®. Muy utilizado en Second Life y otros interfaces para la creación de videotutoriales. Camtansia es otra alternativa.

Fuerza X, Y, Z
Simulación física
En realidad son tres parámetros que sirven para definir fuerzas orientadas que le afectan solo a el objeto involucrado desde cada uno de los 3 posibles ejes. Sus valores pueden ir desde -10 a +10 ya que tienen magnitud y sentido.

Full Bright, brillo total
Superficies y texturas
Característica de las superficies en Second Life que permite crear una superficie más clara de tal manera que se visualice como una autoiluminación. Se utiliza de forma combinada con la propiedad Glow para crear objetos luminosos.

Funciones de biblioteca
Programación
Son funciones creadas por Linden Labs. y puestas a disposición de los programadores de Second Life para ayudarles en su tarea. Normalmente son funciones de bajo nivel, es decir, que tocan directamente los recursos que usa Second Life para crear el metaverso. Todas empiezan por ll. Por ejemplo, la función llSay() es una función de biblioteca y permite "decir" cosas a un objeto y por tanto, un programador no la tiene que crear, solo utilizarla de modo correcto.

Funciones de usuario
Programación
Son funciones creadas por los propios programadores posiblemente porque su tarea sea muy específica y no existan funciones de biblioteca que realicen la función que pretenden programar. A veces, también pueden tener un sentido meramente organizativo.

Gallery of musical sculptures
Integración Multimedia
Lugar de Second Life que presentan una colección de esculturas multimedia. La mayoría son esculturas cinéticas que presentan cambios de color dinámicos y emiten música, halos y otros efectos multimedia.

Gestos, gestures
Comunicación
Son animaciones acompañadas de sonidos que sirven para enriquecer la comunicación con los avatares que tenemos a nuestro alrededor ya que suponen una comunicación gestual, no verbal. Su ejecución debe ser muy rápida por lo que el acceso es directo desde el menú contextual del avatar.

Glow
Superficies y texturas
Característica de las superficies en Second Life que crean un halo o resplandor y que admite valores porcentuales. Se emplea para crear materiales como neones, bombillas, etc.

Graef, Ailin
Personas
Propietaria de la inmobiliaria virtual XXX, que es la primera millonaria de SL en la vida real, ya que su empresa vale más de un millón de dólares.

Graphics Processor Unit, GPU
Simulación física
Es un microprocesador dedicado al procesamiento de gráficos y que se utiliza para sistemas 3D interactivos como videojuegos. De esta forma aligera el trabajo de la CPU que puede

dedicar su potencia de cálculo a tareas como el cálculo de la inteligencia artificial presente en los metaversos y videojuegos.

Gravedad, gravity
Simulación física
Indica la masa de un cuerpo con valores desde -10 a 10. En realidad lo que hace es crear una animación con distinto sentido si es – o + (si es hacia la tierra o hacia el espacio) y cambiar la velocidad según el valor para simular gravedad.

Griefer
Personas-Avatares
Avatar molesto e insultante que tiene un comportamiento indeseado para al resto de los habitantes de SL, ya sea por palabras ofensivas, actitudes y actividades que interfieren con la vida normal.

Havok
Simulación física
Es una máquina física comercial. Havok se ejecuta en hardware mediante el GPU. Actualmente lo integran muchos juegos como Age of Empires, Halo 3, Residente Evil y Second Life.

Help Island
Lugares
Isla de SL donde se citan ayudantes, es decir, personas que voluntariamente se ofrecen para resolver dudas sobre todo de programación.

Hole Size, tamaño del agujero
Construcción
Modificador aplicable a los prims basados en el Toro. Cambia el tamaño del orificio central típico de estas geometrías.

Hollow, orificio
Construcción
Modificador aplicable a los prims basados en la Caja. Crea un orificio central que puede ser desde inexistente a ocupar el 95% del tamaño total del prim.

Hollow Shape, forma del orificio
Construcción
Modificador que indica la forma del orificio Hollow si existiera, puede tomar tres formas: circular, cuadrada o triangular.

HUD
General
“Heads up display“. Objetos que se agregan a la pantalla de la aplicación, permitiendo realizar funciones adicionales. No son visibles por otros usuarios. Son como plugins para mejorar o crear nuevas funcionalidades a la experiencia Second Life

Iluminación dinámica
Iluminación
Característica que pretende aumentar el realismo de la inmersión en el metaverso. Lo que produce son cambios en la iluminación según la hora del día que sea y por tanto para ver cambios grandes es mejor la hora de amanecer y el anochecer. La iluminación dejará de ser dinámica si elegimos nosotros directamente el momento del día dónde nos apetece estar con el comando mundo > configuración del entorno.

Interacción
Interacción
Nos referimos con interacción al proceso de acción y reacción que se produce en Second Life entre los avatares que representan a los usuarios y los objetos que se encuentran en el metaverso.

In-world
General

Así denominan las personas a los encuentros, eventos o cualquier cosa que sucede en SL. Por ejemplo, si quieren señalar que una lección se dará en SL dicen que serán clases in-world.

Instant Message, IM o Mensajería instantánea
Comunicación

Comunicación privada de personas a persona o dentro de un grupo de personas, que pueden estar en cualquier lugar de SL, por tanto no requiere cercanía. La diferencia con el Chat, es que este es público.

Inventario
General

Ventana que nos permite gestionar todo lo que poseemos como usuarios y que está organizado por carpetas como objetos, sonidos, texturas, landmarks, etc.

Islas privadas, state
Lugares

Territorios no administrados directamente por Linden Labs pero mantenidos en sus servidores que tienen un dueño que a su vez vende parcelas. Casi siempre crean un contrato con condiciones que deben cumplir los compradores y que provoca homogeneidad en construcciones, avatares y objetivos de estos. Por ejemplo, puede ser una isla dedicada a los robots o a recrear el principado de Montecarlo. Una isla privada ocupa todo un servidor (un sim) y cuesta 295 $USA para empresas y 195 $USA para organizaciones educativas o sin fines de lucro.

La casa encendida
Lugares

El proyecto español más ambicioso en el ámbito cultural en Second Life donde se ha reproducido la forma del edificio real y que realiza actividades sociales, exhibiciones culturales como conciertos en la vida real y en SL.

Landmark, LM o punto de referencia
Lugares

Marca de un lugar. Se puede guardar las coordenadas de un lugar y reverenciarlo con un nombre y guardarlo mediante el comando del menú Mundo > Crear punto de referencia aquí. Y entonces entrará a formar parte de nuestro inventario en la carpeta Landmarks.

Lag
General

El retardo producido por el necesario cálculo del simulador.
El lag, o lentitud en los procesos de SL, es uno de los primeros reclamos de cualquier usuario al intentar moverse por SL. Puede deberse a muchas cosas, desde la velocidad del procesador, la velocidad de la conexión a Internet, hasta los procesos propiamente de SL en el sitio donde uno se encuentra. Es decir, existe lag client side (del lado del cliente) y server side (del servidor). Uno originado por la transferencia de los datos y procesamiento en tu PC (o Mac) y el otro por el trabajo que se realiza en el servidor de SL.
Hay muchas cosas que pueden ayuda a disminuirlo:

1. Cuando subas una textura, que no sea mayor a 512×512 px. Más que eso no sirve en SL (máxima resolución es 512×512), y sólo ayuda que el tiempo de transferencia sea mayor.
2. Evita utiliza objetos que emitan partículas, incluyendo brillos (bling).
3. Evita los scripts, principalmente aquellos que están permanentemente “abiertos”, escuchando el canal del chat (0). Incluso scripts dormidos consumen recursos del Sim.
4. Desviste (quítate) cualquier script que tengas puesto pero no estes utilizando: armas, defensas, HUDs, animaciones (AO).
5. Configura adecuadamente las preferencias de SL (muy importante):
    1. Ancho de banda (Bandwidth) 500 o menos
    2. Distancia de visión (Draw Distance) 90 o meno. A más distancia visual, más se demora en cargar todo.

3. Disminuye el número de partículas visibles a cada momento, máximo 1024.
4. Disminuye el detalle del terreno a Bajo (low)
5. Cache a 200
6. Desactiva “Filtro Anisotrópico”

Además, algo básico es contar con una conexión a Internet lo más rápida posible, un procesador relativamente poderoso y RAM en cantidad astronómica. SL no es para cualquier computador, consumo muchos recursos.

Ley de Moore
General
Ley empírica que formula que cada 18 meses se duplica el número de transistores que se colocan en un circuito integrado pero que actualmente se puede extender a otros aspectos de la industria informática que también aumentan de forma exponencial.

Libsecondlife
General
El proyecto libsecondlife es un esfuerzo para la comprensión de sl desde una perspectiva técnica y extender e integrar el metaverso con el resto de la web. Esto incluye entender como los clientes oficiales de sl operan y como se comunican con los servidores de simulación, así como el desarrollo de herramientas independientes de terceros.

Lighting Detail
Iluminación
Es un control que se sitúa dentro de las opciones de configuración que aparecen en la ficha graphics dentro de las preferencias del cliente de Second Life. Estas preferencias se muestran en el menú Edit. En principio no se muestran todas las posibilidades de edición y debemos pulsar la opción Custom para que aparezcan. Lighting Detail presenta dos alternativas: “sun and moon only” que es la opción por defecto y supone que las luces vistas serán solamente la del sol y la luna o “nearby local lights” que permite ver luces locales cercanas y que será la opción que tendremos que elegir cuando creamos y editamos luces propias o queremos ver las luces que nos rodean.

Linden Dollars (L$)
Empresa-Negocios
Moneda oficial de SL. Sufre variaciones en su cotización diarias en dólares estadounidenses. Por ello, se pueden canjear por cualquier moneda del mundo real. Existen también bancos on-line que permiten este cambio.

Linden Labs
Empresa-Negocios
Empresa californiana de San Francisco que publica y mantiene Second Life.

Lipsky, Jeffrey
Arte
Reconocido artista que también crea versiones virtuales de sus obras reales y cuyo nombre en Second Life es Filthy Fluno. Es un influyente gurú del arte digital.

Lista de amigos
Comunicación
Se trata de un listado con la identificación de los avatares que deseemos y que podemos gestionar para darlos de alta y baja, hablar directamente con ellos u ofrecerles que se teletransporten a nuestro lado.

llGetPos()
Animación
Esta función nos devuelve un vector que es el da la posición del objeto que la referencia, o sea, el X, Y y Z con respecto al punto 0,0,0 de la región dónde se encuentra el objeto.

llGiveInventory()
Interacción
Función muy utilizada en procesos interactivos entre avatares y que hace que un objeto que puede ser de tipo textura, sonido, imagen, prim, etc. sea pasado de el inventario de un avatar al inventario de otro.

llMoveToTarget()
Animación
Mueve inercialmente un objeto físico a una posición determinada en x segundos. Tanto la posición objetivo como los segundos que debe emplear en el movimiento se pasan por parámetro a la función.

llParticleSystem()
Sistema de partículas
Función que permite la definición de un sistema de partículas en Second Lilfe. Es la única forma de crear y definir un sistema de partículas y es por ellos que es una función donde se pueden utilizar hasta 28 parámetros.

llPlaySound()
Interacción
Función muy utilizada en procesos interactivos como respuesta a un evento y que produce que se emita un sonido. Normalmente este sonido será corto y representará un efecto sonoro, por ejemplo, cuando se abre una puerta o se rompe un jarrón, se puede ejecutar esta función para aumentar el realismo.

llSetColor()
Interacción
Función muy utilizada en procesos interactivos como respuesta a un evento y que produce que se cambie el color de las caras de un objeto. Se puede cambiar algunas o todas las caras.

llSetPos()
Animación
Sirve para poner en una posición determinada el objeto que referencia esta función. Las coordenadas de la posición objetivo se pasa por parámetro mediante un vector.

llSetTexture()
Interacción
Función muy utilizada en procesos interactivos como respuesta a un evento y que produce que se cambie la imagen o textura con que se recubren las caras de un objeto. Se pueden texturizar algunas o todas las caras.

llSetTextrueAnim
Superficies y texturas
Función del lenguaje de programación LSL que permite modificar la forma en que una textura es aplicada sobre un objeto y además que estos cambios se puedan controlar en el tiempo, lo que podemos utilizar para crear texturas animadas por transformación de la propia textura.

llTargetOmega()
Animación
Sirve para girar objetos en un determinado eje a una velocidad constante.

LSL, Linden Scripting Language
Programación
Es el lenguaje que proporciona Second Life para que los usuarios puedan programar scripts y por tanto crear funciones, animaciones, interfaces y por tanto crear más Second Life. Tiene una sintaxis muy parecida a C pero con funciones enfocadas a la realidad virtual.

Mapa
General

Se muestra tecleando Ctrl + M o pulsando el botón “Mapa”.Nos muestra el mapa de Second Life más cercano al punto donde nos encontremos. Permite mostrar distintas entidades para buscar lo que deseemos, personas, terrenos en venta, etc. También tenemos un buscador de sitios por palabras y que permite el teletransporte. Normalmente muestra una extensión demasiado grande de terreno.

Machinima
Cámaras

Se trata de la realización de películas utilizando videojuegos y metaversos como escenarios y los avatares como personajes. Normalmente son cortometrajes. Surgen como la evolución cultural de las películas de promoción de los videojuegos.

MachinimaCam Pro
Cámaras

Cámara a la venta en Second Life por un precio aproximado de unos 2000 L$ y que tiene características que permiten un fácil control además de efectos incorporados como giros en espiral, travelling, etc.

Manejadores de eventos
Programación

Estos manejadores lo que hacen es atrapar e identificar los sucesos y ejecutar las acciones de respuesta si es el caso.

Mapeado default
Superficies y texturas

Este mapeado, es decir, esta forma de aplicar un mapa sobre el objeto es el que se aplica por defecto y en principio es el más adecuado, ya que Second Life se encarga de ver la forma básica del objeto y en función de ello, la forma de aplicar la textura.

Mapeado planar
Superficies y texturas

Este mapeado aplica la textura de forma plana sobre el objeto aunque en principio no se adapte bien por no ser plano el objeto, por ejemplo, sobre un objeto esférico va a resultar no muy natural.

Mapping, mapeamiento
Superficies y texturas

Forma de aplicar una imagen sobre la superficie del objeto. Al ser los modelos objetos 3D y la imagen un objeto 2D, el problema se hace evidente. Se puede pensar que es el mismo problema que tenemos cuando queremos envolver una balón de futbol (3D) con un papel de regalo (2D).

Máquina física
Simulación física

Es un sistema del tipo conocido como máquina física. Se integra en simuladores 3D como videojuegos y metaversos para dotarlos de simulación física como colisiones, inercias, etc.

Material
Simulación física

Es un control que aparece en la parte inferior de la ficha Objeto en la edición de prims y que sirve para definir el material con que está hecho. Este dato lo utiliza el simulador físico en caso de que este objeto se defina como físico. De esta manera aplicará distintos parámetros de elasticidad, peso específico,… según el material. Esto provocará por ejemplo, que el simulador físico, haga que un objeto de caucho rebote más que otro de las mismas características pero de madera.

Max Particle Count
Sistema de partículas
Número máximo de partículas que se visualizarán. Es evidente que tendrá un efecto grande en la forma del sistema de partículas. Se puede asimilar al caudal de cualquier flujo.

Menú Advanced
Superficies y texturas
Menú que no aparece por defecto pues contiene algunas funciones que afecta a la forma en que se visualiza Second Life y se supone que es para usuarios expertos. Aparece el pulsar simultáneamente las teclas control + shift + D.

Metafutiring
Empresa-Negocios
Empresa especializada en crear espacios en SL.

Metaverse
Empresa-Negocios
Empresa especializada en crear espacios en SL.

Metaverso
General
Concepto que aparece por primera vez den la novela “Snow Crash”, publicada en 1992 por Neal Stephenson. Se trata de una visión de mundos en 3D donde los humanos interactúan como en el mundo real, trabajando, relacionándose, pasando el rato, etc.

Minimapa
General
Se muestra tecleando las teclas Ctrl + Shift + M. Nos muestra un área alrededor más reducida que el Mapa, por lo que muchas veces es más útil. Muestra el avatar propio como un punto amarillo y los demás como puntos verdes. También se visualiza un triángulo visualizar la dirección hacia dónde estamos mirando.

MMORPG, Massively Multiplayer Online Role-Playing Game
General
Juego de rol para jugar conectado a Internet multijuador en masa. Alguno de los más conocidos son World of Warcraft, Dofus, Runscape y otros.

Modificador
Construcción
Operaciones geométricas aplicadas sobre objetos en construcción que varían la forma de un objeto 3D en distintos grados. Los modificadores aplicables a un objeto dependen de su bloque de construcción o prim base.

Modo de emission
Sistema de partículas
Es el parámetro mínimo necesario para crear un sistema de partículas y define la forma de emisión, es decir, el modo de la fuente de partículas. Tiene 5 posibles valores que son drop, explode, angle, angle_cone y angle_cone_empty. El modo más utilizado es el modo explode.

Modo Encuadre
Cámaras
Se inicial al pulsar las teclas Alt+Control+Shift+botón izquierdo del ratón. Permite trasladar la cámara horizontal y verticalmente pero no en profundidad.

Modo Focus
Cámaras
Se inicial al pulsar las teclas Alt+botón izquierdo del ratón. Permite tener un control en profundidad de la cámara por defecto sobre el objeto al que apuntemos con el ratón. El objeto además pasará a ser una especie de objetivo de cámara por lo que los giros y movimientos con el ratón tendrán el objeto como punto de transformación.

Modo Órbital
Cámaras
Se inicial al pulsar las teclas Alt+Control+botón izquierdo del ratón. Permite girar la cámara en cualquier dirección sobre un objeto al que se apunte con el ratón pero no en profundidad, es decir, se mantendrá a una distancia fija.

Moodle
Educación
Implementación de código abierto para la gestión de cursos muy extendido en la actualidad como soporte para la educación online. La primera versión fue creada por Martin Dougiamas en agosto de 2002

Multimedia
Integración Multimedia
Se utiliza este término para describir sistemas digitales que integran distintos formatos de contenidos como el texto, imagen, animación, vídeo, audio y la interactividad.

Nearby local lights
Iluminación
Opción que permite que los objetos luminosos se visualicen como tales en contraposición a la opción “Sun and moon” que solo permite ver la luz que proviene del sol y la luna, por lo tanto la primera opción permite ver la luz artificial. Esta opción se encuentra en el menú Edit > Preferences > Graphics > Custom en el grupo “Lighting Detail”.

Newbie
Personas-Avatares
Nuevo miembro de SL. Persona que carece de experiencia y por tanto necesita cierta ayuda.

Notas de texto, notecards
Comunicación
Son documentos que se guardan en el Inventario y que podemos utilizar como queremos aunque normalmente sirven para compartir textos con otros avatares.

Objeto autoiluminados, full bright
Iluminación
Se utiliza para crear un efecto de autoiluminación en los objetos. Estos objetos presentan una textura que será mucho más luminosa de lo habitual, es decir, tendrá más cantidad de blanco. Esta característica se elige en la edición del objeto en la pestaña Features y solo admite dos valores: o se activa o no.

Objeto enlazado
Construcción
Los objetos enlazados son aquellos a los que se han aplicado una operación de enlace y por tanto forman un grupo con otros para crear un objeto estructuralmente más complejo. Se pueden manipular de forma conjunta y con un único punto de transformación situado en el centro del prim raíz.

Objeto físico
Objetos
Objeto creado en Second Life y que tiene propiedades físicas, lo que hace que pueda ser reconocido y gestionado por la máquina física y por tanto presentar inercia, fricciones, peso,…

Objeto raíz
Construcción
Uno de los prims que forman parte de un objeto enlazado será definido como objeto raíz y será por haber sido enlazado al resto en último lugar. El objeto raíz tiene la función de servir de referencia al objeto compuesto y su centro, será el punto de transformación para las operaciones que se le apliquen al objeto enlazado. Además se denota el objeto raíz porque los objetos compuestos cuando son seleccionados presentan un halo en dos colores: amarillo para el objeto raíz y azul para el resto de objetos.

Offset
Superficies y texturas
Característica de las superficies en Second Life que produce un cambio en el origen de la aplicación de una textura y por tanto podemos utilizar para desplazarla tanto vertical como horizontalmente.

Oldbie
Personas-Avatares
Usuarios con mucha experiencia en Second Life.

OpenGL
Simulación física
Es un API multilenguaje y multiplataforma para crear aplicaciones gráficas 2D y 3D. Creado por Silicon Graphics Inc. Se usa en software para juegos, simuladores y CAD. Competidor de DirectX.

OpenSim, OpenSimulator
General
Plataforma de código abierto para la publicación de metaversos cuyo objetivo es mantener la compatibilidad con Second Life, por lo tanto, se puede considerar el Second Life en software libre y permite implementar simuladores en servidores particulares.

Orientation Island
Lugares
Isla de SL donde aterrizamos cuando nacemos en SL. Podemos empezar a entrenarnos aquí con el interfaz del programa antes de teletransportarnos a otras islas.

Parámetros de entrada
Programación
Son las variables que necesita una función para realizar su tarea. Por ejemplo, la función llSay() necesita saber lo que tiene que decir, le tenemos que “pasar” pues, una cadena de caracteres para que realice su función.

Particle Lab.
Sistema de partículas
Lugar de Second Life dedicado al aprendizaje de los sistemas de partículas. Muy bien organizado y con amplia documentación y ejemplos.

Particle Sources
Sistema de partículas
Esta es una opción para visualizar o no el prim emisor del sistema de partículas y se encuentra en el menú Views > beacons.

Partícula
Sistema de partículas
Es un elemento bidimensional con una textura que forma parte de un grupo llamado sistema de partículas con sentido propio. No se pueden gestionar individualmente sino como integrantes del sistema. Se le aplican parámetros como el tiempo de visualización, su posición, su color, etc y casi todos ellos permiten un grado de aleatoriedad.

Path Cut
Construcción
Modificador aplicable a los objetos basados en la primitiva Caja. Viene dado por dos valores que limitan una trayectoria de corte trasversal por la cual se reduce el perímetro del prim. Es como si el cuerpo fuera una tarta al que le falta un trozo.

Permisos
General
Característica esencial de Second Life para preservar la propiedad de los objetos en el metaverso. La propiedad puede ser intelectual o comercial y es lo que hace posible la creación

de una economía propia. Existen 4 tipos de permisos (mover, modificar, copiar y transferir) que pueden estar aplicados a cuatro categorías: dueño, grupo, todos los demás y siguiente dueño.

Persistencia
Programación
Es una característica del propio metaverso que sirve para dotarlo de más realismo ya que permite que los objetos mantengan su estado a lo largo del tiempo si no ocurren eventos que los modifiquen. Por ejemplo, la persistencia hace que si tenemos un vehículo en marcha y dejamos el metaverso, cuando volvamos lo encontraremos en marcha. El mantenimiento de la persistencia reside en el publicador del metaverso.

Physics processing unit, PPU
Simulación física
Es un microprocesador dedicado al cálculo de problemas físicos y que normalmente forma parte de una máquina (o simulador) física. Se utilizan para sistemas 3D como videojuegos.

PhysX
Simulación física
Es una PPU o unidad de procesamiento de física. Compuesto por hardware, en un chip y un kit de desarrollo para su funcionamiento en aplicaciones 3D interactivas. En la actualidad viene integrado en los chips gráficos de la marca NVidia.
Polígono
Superficies y texturas
Unidad de superficie. Normalmente compuesto por vértices que los limitan y aristas que unen los vértices y le proporcional dirección. Se unen para formar superficies tridimensionales y por tanto modelos 3D.

Prim
Construcción
Objeto primitivo básico que puede ser modificado y enlazado con otros para crear cualquier cosa que existan en Second Life. Existen 15 objetos distintos de los que partir para crear todos los demás entre los que se incluyen árboles y hierba, siendo todos los demás de carácter geométrico.

Profile Cut
Construcción
Modificador aplicable a los objetos basados en la primitiva Toro. Cambia el perfil curvo típico de los toros por otro rectilíneo que por tanto modifica la forma externa y que viene dado por dos valores que limitarán su aplicación.

Programación orientada a objetos, POO
Programación
Modelo o paradigma de la programación moderna que usa entidades a nivel de objetos y sus interacciones para crear programas. Usa técnicas como el encapsulamiento, la herencia, la modularidad y el polimorfismo. Surge en los años 80 como respuesta a la última “crisis del software” y en la actualidad casi todos los lenguajes de programación de alto nivel usados en el mundo son de este tipo.

Propietario, owner
Construcción
Usuario propietario de un objeto en Second Life y por tanto poseedor de los derechos comerciales de dicho objeto. Se utiliza desde el control Creator de la ficha General de la caja de diálogo de edición del objeto. También presenta un botón para acceder al perfil del creador.

PSYS_SRC_PATTERN
Sistema de partículas
Ver Modo de emisión de sistemas de partículas.

QuickTime
Integración Multimedia
Arquitectura multimedia compuesta por una serie de bibliotecas de software y un visualizador de contenidos multimedia. Se utiliza más para contenidos de vídeo que de audio. Fue desarrollado y es propiedad de Apple®. Permite la visualización y la edición de formatos como MOV, QT y otros.

Radius
Construcción
Modificador aplicable a los objetos basados en la primitiva Toro. Corta el toro por lo mitad y aplica una espiral sobre el eje trasversal del cuerpo.

Real Glow
Iluminación
Efecto luminoso que trata de imitar el halo o resplandor de algunas luces, como por ejemplo las de neón. Este efecto se controla desde la edición del objeto en la pestaña Features mediante el control llamado “Glow” y puede tomar valores desde 0 a 1.

RealXtend
General
Plataforma de código abierto para la publicación de metaversos que surge como alternativa a OpenSim pero de momento solo para Windows. Esta puede ser que sea la razón de que de momento tenga menos aceptación que OpenSim.

Reflexión
Superficies y texturas
Puede tener distintos significados pero en el ámbito de las superficies se refiere a las propiedades que tienen ciertos materiales de reflejar los objetos que lo rodean como por ejemplo los espejos.

Renderglow
Iluminación
Comando que se debe introducir escribiéndolo en la caja de diálogo “Debug Settings” que aparece cuando se ejecuta el comando homónimo del menu “Advanced”. Si nos devuelve el valor verdadero, quiere decir que nuestra tarjeta gráfica es capaz de representar resplandores de tipo “glow”. El menú “Advanced” se hace visible al pulsar simultáneamente Ctrl+Shift+ Alt´D.

Repeats per face, repetición por cara
Superficies y texturas
Repetición de la textura al aplicarla por cada una de las caras que compongan el objeto 3D. Esta característica es muy importante para ahorrarnos texturas, ya que si queremos crear por ejemplo una pared de azulejos, repetiremos la textura de un solo azulejo las veces necesarias.

Repeats per meter, repetición por metro
Superficies y texturas
Repite una textura una cantidad de veces por metro. Esto hace que podamos ahorrar en el número de texturas aplicadas por objeto.

RGB
Superficies y texturas
Código digital basado en tres colores: rojo, verde y azul cada uno de los cuales puede tomar valores de 0 a 255 para describir todos los colores posibles.

Revolutions
Construcción
Modificador aplicable a los objetos basados en la primitiva Toro. Hace que el cuerpo toroide se retuerza en espiral a modo de muelle.

Rezzear
Objetos
Crear un objeto en SL y hacerlo aparecer. Su origen puede ser el propio modelado o que lo tengamos en nuestro Inventario y lo saquemos al espacio SL.

Rosedale, Philip
Personas-Avatares
Creador de Second Life. Su nombre dentro del metaverso es Philip es licenciado en física y lanzó la primera versión en 2002 con la intención original expuesta en Snow Crash, es decir, dar una alternativa virtual a la vida real.

Sandbox
Programación
Es una isla o parte de una isla donde se permite a los avatares no propietarios crear objetos y aprender a modelar y programar.

Script
Programación
Unidad de programación que implementa un algoritmo y por tanto tiene un fin determinado. En Second Life el código está escrito en LSL. Los scripts son contenidos en objetos y por tanto serán estos los que se vean afectados por el resultado del código escrito.

Sculpt, sculptie o sculpted
Construcción
Uno de los prims de Second Life cuya forma viene dada por una imagen. Por lo tanto es una geometría esculpida por una textura de tal modo que el código de color RGB de la imagen se convierte en código geométrico de vértices. Estas imágenes se generan en terceros programas y se utilizan para modelos objetos de tipo malla como orgánicos y realistas.

Sculpt map
Construcción
Imagen utilizada para crear sculpts. En principio es una imagen normal con un código RGB que Second Life se encarga de interpretar y visualizar como una geometría.

Sculpture garden
Arte
Espacio en Second Life, dedicado a la escultura fractal y con base matemática donde podemos ver obras de incluso 2004. Es propiedad e Henry Segerman.

Second Life
General
Es el principal exponente de los metaversos.

Segerman, Henry
Arte
Doctor en matemáticas y profesor en la universidad de Austin, Texas, es uno de los pioneros en arte en Second Life, sobre todo es esculturas de carácter matemático como fractales. Es el propietario del conocido “Sculpture garden”.

Sensor()
Interacción
Manejador de evento muy utilizada en procesos interactivos que se activa cuando un avatar se aproxima al objeto que posee el script donde se utiliza sensor(). Se suele utilizar para identificar lo que pasa alrededor de un objeto o para que este mande una alerta al propietario del objeto con el fin de que pueda atender al visitante.

Share with Group
Construcción
Si un objeto tiene esta opción seleccionada, cualquier miembro del grupo del creador puede usar libremente el objeto. El grupo en cuestión se visualiza en la parte de arriba de este control que se encuentra en la ficha General de la caja de diálogo de edición de objeto.

Shear, sesgo
Construcción
Modificador aplicable a los objetos basados en la primitiva Caja. Produce un cambio en el paralelismo de las caras dos a dos que conforman la caja creando un efecto de inclinación.

Sincronización de labios
Integración Multimedia
Función de Second Life que permite simular el movimiento de la boca cuando hablamos por el chat de voz. Permite aumentar el realismo de la conversación. Esta función se puede habilitar desde el menú avanzado (que aparece al pulsar Ctrl + Shift + D) y en el submenú Character encontramos el comando “Enable Lips Sync”.

Shininess
Superficies y texturas
Característica de las superficies en Second Life que definen la cantidad de pulida que está una superficie. Esto se visualiza como una superficie con muchos brillos superficiales.

Sim, Simulador
General
Cada una de las islas que componen Second Life. Cada isla o sim es en la vida real un servidor al que nos conectamos cuando entramos en SL. Cuando nos teleportamos, pasamos de un servidor a otro.

Sistema de partículas
Sistema de partículas
Es un conjunto de elementos bidimensionales llamadas partículas que forman una entidad propia tanto estética como funcional y que se emplean en sistemas 3D interactivos para crear efectos especiales como explosiones, resplandores,… y fenómenos atmosféricos como niebla, lluvia y otros.

Skew
Construcción
Modificador aplicable a los objetos basados en la primitiva Toro. Produce un cambio total en la geometría que es difícil de describir. Por un lado, corta el toro por la mitad creando una especie de cilindro retorcido y de grosor variable.

Sloodle, Simulation Linked Object Oriented Dynamic Learning Enviroment
Educación
Implementación de código abierto libre para la gestión de cursos integrados en Second Life.

SRurl
Publicación offline
Son enlaces que permiten teletransporte directamente desde un navegador web a lugares en el metaverso mediante la ejecución del programa cliente de Second Life.

Snow Crash
General
Novela de 1993 llamada “Snow Crash” del escritor Neal Stephenson que describe por primera vez un metaverso y en la que se inspiran los autores que después los implementaron.

SRC_BURST_RATE
Sistema de partículas
Ver Tiempo entre explosiones de sistemas de partículas.

State_entry()
Programación
Es una función de tipo evento. El evento al que responde es el propio inicio del script, por lo tanto podemos decir que representa la iniciación del script, o sea, que si escribimos algo a continuación, será lo primero que ejecute el script.

Steinbeck, Sasun
Arte
Editor del blog “Art Galleries of Second Life”, posiblemente el más reconocido sobre arte en Second Life.

Stephenson, Neal
Personas-Avatares
Autor de la novela Snow Crash y que describe por primera vez los metaversos.

Streaming
Integración Multimedia
Se trata de una distribución de contenido multimedia por Internet bajo demanda que no requiere de la descarga de ficheros en el ordenador del cliente. Para realizar streaming se requiere un buffer o memoria temporal. Se utilizar para crear radio por Internet, cursos a distancia, etc.

String
Programación
Tipo de variable que puede contener una cadena de caracteres alfanuméricos.

Suavidad, softness
Simulación física
Indica si un objeto es muy rígido o muy blando con un valor de 0 a 3. Para ello, crea secciones trasversales en altura del objeto. Si el valor es 0 no tendrá secciones, si vale 1 tendrá 2 secciones, si vale 2 tendrá 4 secciones y si vale 3 tendrá 8 secciones. Estas secciones se verán supeditadas a cambios de posición independientes y por tanto simularán flexibilidad.

Superficie
Superficies y texturas
Los modelos geométricos no son sólidos sino que se crea solo la superficie exterior por tanto es lo que vemos de ellos. Es cómo una cáscara que puede tener diversas características para cambiar su apariencia. Esta compuesta normalmente por polígonos que a su vez se encuentran definidos por vértices tridimensionales.

Taper, afilar
Construcción
Modificador aplicable a los objetos basados en la primitiva Caja. Permite modificar la escala de una determinada cara de la caja y esto tiene la consecuencia de “afilar” el objeto.

Targa, TGA
Superficies y texturas
Tipo de fichero que puede importar Second Life y que se utiliza para subir texturas, fotos, etc. al metaverso. Este tipo de archivo se puede trabajar y exportar desde PhotoShop.

Target
Sistema de partículas
Es un prim al que pueden dirigirse las partículas una vez que salen del emisor. Por lo tanto, podemos hacer que el flujo de partículas tenga un objetivo, el target.

Tarjetas de llamada, calling cards
Comunicación
Son como invitaciones que un avatar puede dar a otros usuarios para si está conectado comunicarse con él mediante mensajes instantáneos.

Teleport, Teletransporte
Lugares
Método para teletransportarse o teleportarse de un sitio a otro de Second Life. En realidad, permite pasar de un servidor a otro o incluso de una isla a otra inmediatamente.

Tensión, tension
Simulación física
Es un parámetro que contrarrestar o compensa a "arrastrar". Por tanto, simula la fricción o resistencia al movimiento de un cuerpo. Puede tomar valores de 0 (no se mueve) hasta 10 (se mueve al mínimo moviendo).

Textura
Superficies y texturas
Característica de las superficies en Second Life que permite incluir imágenes reales como parte de una superficie y por tanto obtener objetos fotorealistas.

Textura animada por intercambio
Superficies y texturas
Produce un efecto especial en un objeto 3D como consecuencia de intercambiar unas texturas por otras cada cierto tiempo.

Textura animada por transformación
Superficies y texturas
Produce un efecto especial en un objeto 3D como consecuencia de transformar la forma en que se aplicar la textura propia de un objeto creando efectos de movimiento, escalado, rotación y otros mediante la función llSetTextrueAnim

Textura Bump
Superficies y texturas
Estas texturas se utilizan para definir irregularidades en las superficies y son creadas en tonos de grises lo cual indicará el grado y sentido de la irregularidad.

Textura por defecto
Superficies y texturas
Es la textura que tienen asignada todos los prims en el momento de crearlos. Es una textura de un tipo de madera muy clara y un poco veteada.

Textura procedimental
Superficies y texturas
Texturas que se utilizan en las superficies pero a diferencias de las texturas fotográficas, estas no han sido tomadas a través de una lente, sino generadas por ordenador. Son típicas las superficies que imitan la madera o el agua. Son menos realistas pero ocupan menos espacio en memoria que es una característica muy importante para aplicaciones en tiempo real.

There
Empresa-Negocios
Metaverso solo para Windows que tiene un aspecto mucho menos realista que Second Life y que está enfocado a adolescentes. Toda la funcionalidad es de pago.

The Tech Virtual Museum
Arte
Museo virtual es Second Life que colabora con museos reales para exponer obras virtuales. Posiblemente, el museo de arte más conocido en SL.

Tiempo entre explosiones
Sistema de partículas
Especifica el tiempo en segundos entre las explosiones de partículas que se emiten. Su valor debe ser un flota. Si ponemos un tiempo entre explosiones de 0.0 el flujo será continuo.

Tierras continentales, mainland
Lugares
Territorio administrado por Linden Labs y cuyos propietarios tienen una cuenta Premium. No tienen reglas estrictas y esto provoca heterogeneidad de construcciones y avatares.

Timer()
Interacción
Manejador de evento muy utilizada en procesos interactivos que se activa cuando pasa cierto tiempo. Lo que produce es una frecuencia en el tiempo que podemos utilizar para hacer algo repetitivo como por ejemplo apagar y encender una luz intermitente.
Twist , retorcer
Construcción
Modificador aplicable a los objetos basados en la primitiva Toro. Viene definido por dos valores que producen un retorcimiento de la figura y que puede llegar a ser de dos vueltas sobre sí mismo.

Ultimate
Integración Multimedia
Hardware preparado para implementar junto con otros dispositivos y software un sistema que permite digitalizar vídeo directamente y transmitirlo vía satélite con el fin de ser visualizado en Second Life. Este sistema ha sido realizado por el EUVE, European Virtual Engineering Techonological Center.

Variable
Programación
Es un elemento de la programación que sirve para guardar valores de distinto tipo que pueden cambiar a lo largo del tiempo. Una variable guardará una zona de memoria para poder almacenar el valor y esta podrá ser referenciada por un nombre a instancias del programador.

Vector
Animación
Variable utilizada en LSL que compuesta por tres floats, o sea, números con decimales y que se utiliza para contener coordenadas tridimensionales.

Viento, wind
Simulación física
Simula la sensibilidad al viento del terreno dónde este un objeto. Sus valores van de 0 (no le afecta el viento) hasta 10 (se mueve a la menor brisa)

Visión del ratón, Mouse look
Cámaras
Cámara que al activarla sustituye a la cámara por defecto y se sitúa sobre los ojos de nuestro avatar por lo que ya no aparece en pantalla. Se dirige con el movimiento del ratón y no del propio avatar. Para activarla ejecutamos el comando del menú Ver > visión del ratón.

Welcome Island
Lugares
Isla de SL donde se tienen muchos lugares y muy diversos y es un buen sitio para observar las oportunidades que nos ofrece este metaverso.

When left –clicked
Construcción
Permite modificar la acción por defecto con la que responde el objeto cuando es clickeado con el botón izquierdo del ratón. La acción por defecto es disparar el evento de “tocado”, pero podemos cambiarla por otras como abrir el objeto, ejecutar medios multimedia del terreno, comprar el objeto, sentarnos en él, etc.

WonderLand
Empresa-Negocios
Metaverso basado en Second Life de Sun Microsystems orientado a las relaciones laborales y no a las personales. Según Sun Microsystems los metaversos son la evolución lógica de los navegadores webs por lo que compraron el protocolo y los servidores de Second Life para adaptarlos a la intranet de su propia empresa de lo que derivó Wonderland. Su principal característica que lo distingue de otros es que permite la ejecución de aplicaciones de PC directamente desde el metaverso.