

Introducción

PARTE II-1

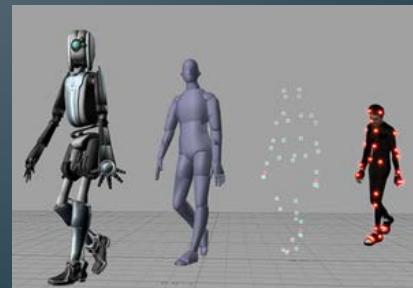
Desarrollo de aplicaciones gráficas 3D

Parte II: Índice

1. Introducción
2. Formatos gráficos
 - X3D (VRML)
 - Collada / KML
 - Aplicación: Google Earth
3. Tecnologías para gráficos 3D: WebGL
 - Gráficos 3D
 - La tubería gráfica
 - Shaders
 - WebGL
 - Frameworks y motores de videojuego

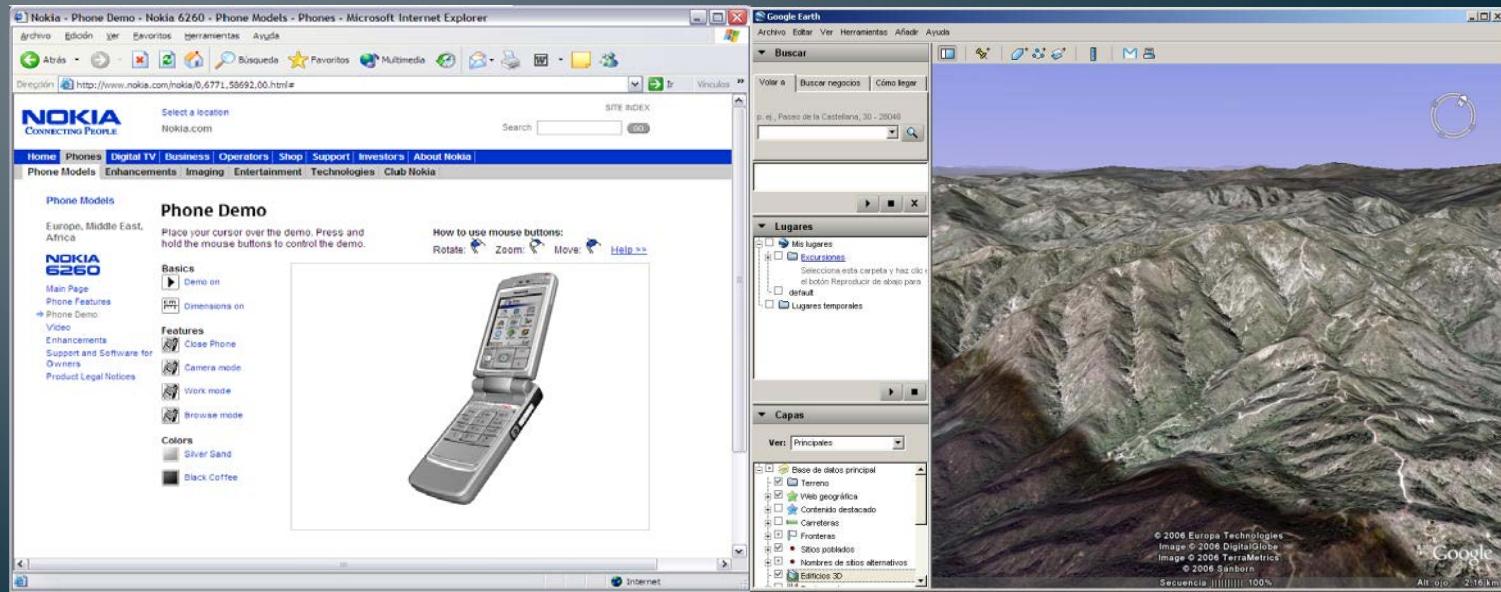
Animación 3D

- La animación 3D es digitalmente modelada y manipulada por el animador
 - Mesh (skin) es la superficie que se pinta
 - Skeleton (rig) es la estructura esquelética que controla la mesh
- Se emplean también otras técnicas
 - Funciones matemáticas (gravedad, sistemas partículas)
 - Simulación de pelo, piel, fuego o agua
 - Captura de movimiento
- A veces es difícil de distinguirla de la acción real



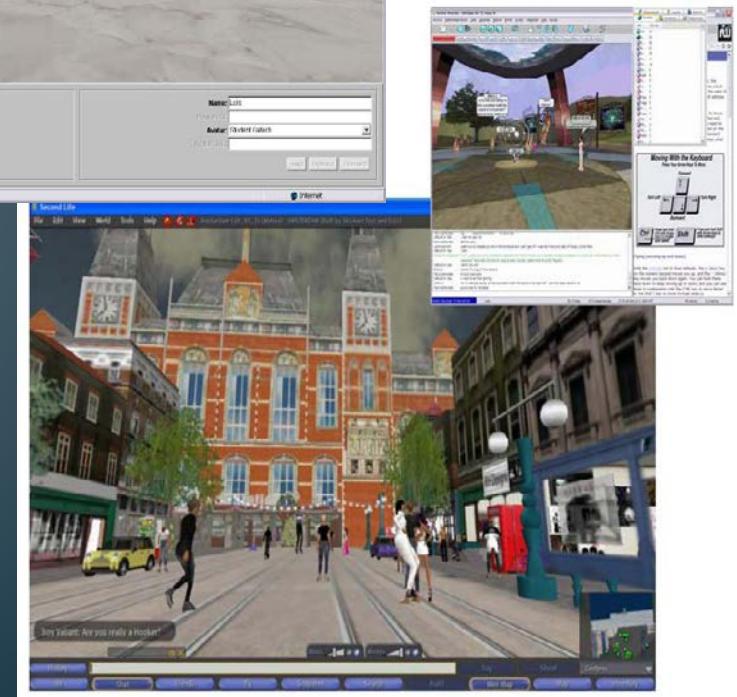
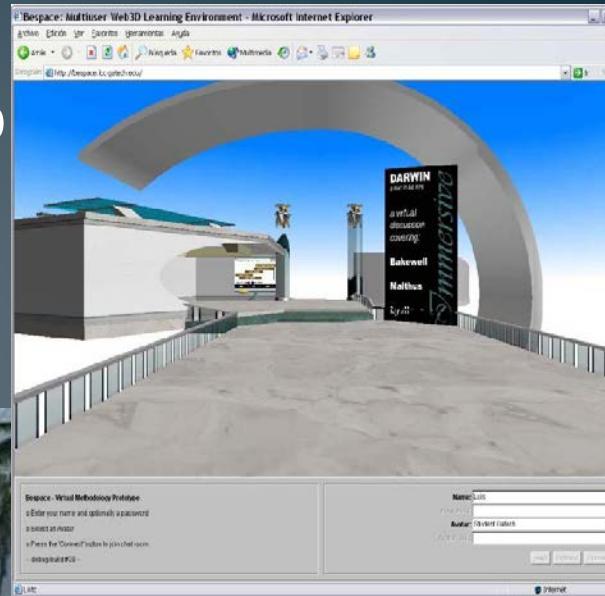
Interacción 3D

- Sistema interactivo 3D
 - El usuario experimenta libremente un contenido en 3D
 - El sistema responde a sus acciones de manera instantánea.
- Ejemplos:
 - Demostradores y visualización de terreno



Interacción 3D

- Museos virtuales
- Comunidades virtuales 3D
- Simuladores
- Videojuegos



Interacción 3D vs Animación 3D

- Diferencias conceptuales:

	Animación 3D	Interacción 3D
Narración	Narración lineal	Experimentación no lineal
Participación	Usuario espectador	Usuario participante
Inmersividad	Usuario no presente	Presencia virtual
Colaboración	Experiencia individual	Posibilidad multiusuario
Media resultante	Vídeo/audio	Modelos 3D / Video / Audio / Texto / Enlaces / Programas
Uso de la Red	Irrelevante	Importante (imprescindible en el caso multiusuario)

Interacción 3D vs Animación 3D

- Diferencias tecnológicas:

	Animación 3D	Interacción 3D
Cálculo	Render off-line	Render en tiempo real
Soporte	Vídeo/cine	Ordenador/consola/teléfono
Resolución	Fija	Variable
Realismo	Muy alto	Medio / Alto
Almacenamiento	Elevado	Reducido
Necesidades HW	CPU / GPU	GPU

Interacción 3D vs Animación 3D

- Capacidades de visualización:

	Animación 3D	Interacción 3D
Modelado	Polygonal, NURBS, modelado sólido	Polígonos
Animación/FX	Sistemas de partículas, fluidos, ropa, pelo, etc.	Con limitaciones por rendimiento del render en tiempo real
Métodos de cálculo	Gouraud, Phong, Blinn, ... Seguimiento de rayos Radiosidad, Mapas de fotones	Con limitaciones por rendimiento del render en tiempo real
Texturizado	Color, transparencia, brillo bump, autoiluminación, reflejos Múltiples texturas, máscaras, Shaders complejos a medida	Con limitaciones por rendimiento del render en tiempo real

Empleo de Shaders

Posibilidades de Interacción

- Hiperenlaces:
 - A otras zonas del mundo virtual
 - A otros mundos virtuales
 - A cualquier URL: imágenes, sonidos, páginas web, e-mail, etc.
- Control del usuario: Sensores
 - Proximidad
 - Visibilidad
 - Toque
- Interacción Multiusuario
 - Sistemas cliente/servidor para comunidades virtuales

Posibilidades de Interacción

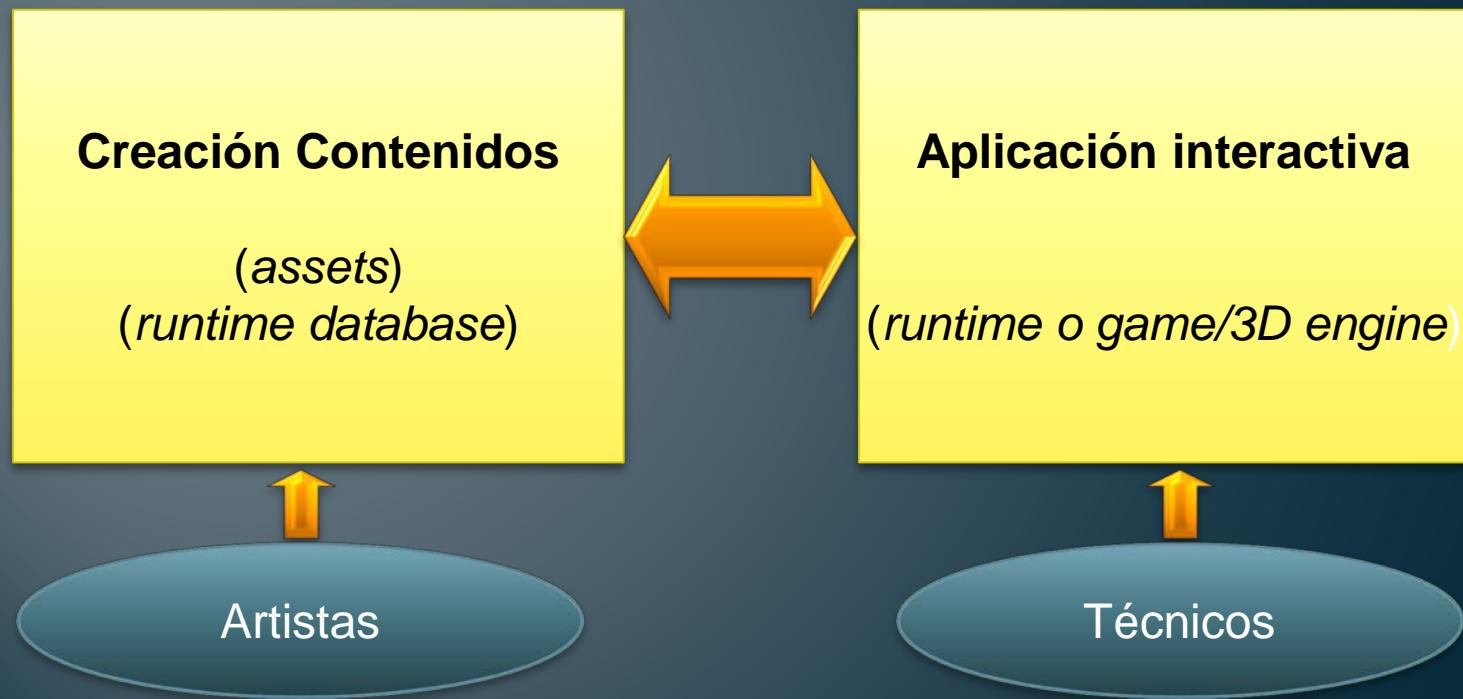
- Navegación:
 - Paseo libre por el mundo virtual
 - Vuelo libre sobre el mundo virtual
 - Examen libre de objetos (rotación, zoom, etc)
 - Paseo dirigido por eventos
- Comportamientos:
 - Animaciones condicionadas (de geometría, luz, textura, sonidos, vídeo...)
 - Permanentes
 - Activadas por los usuarios
 - Activadas por otros eventos
 - Colisiones
 - Gravedad

Necesidades de la Interacción 3D

Tiempo Real !!!

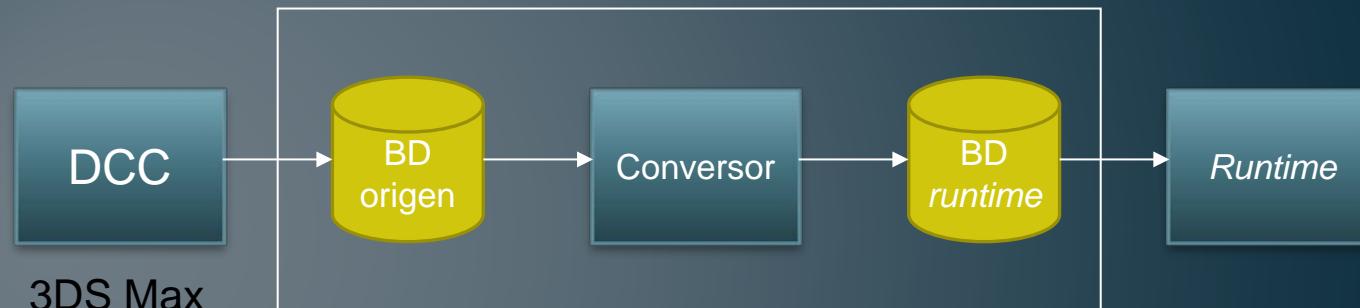
Herramientas

- Flujo de Trabajo



Herramientas

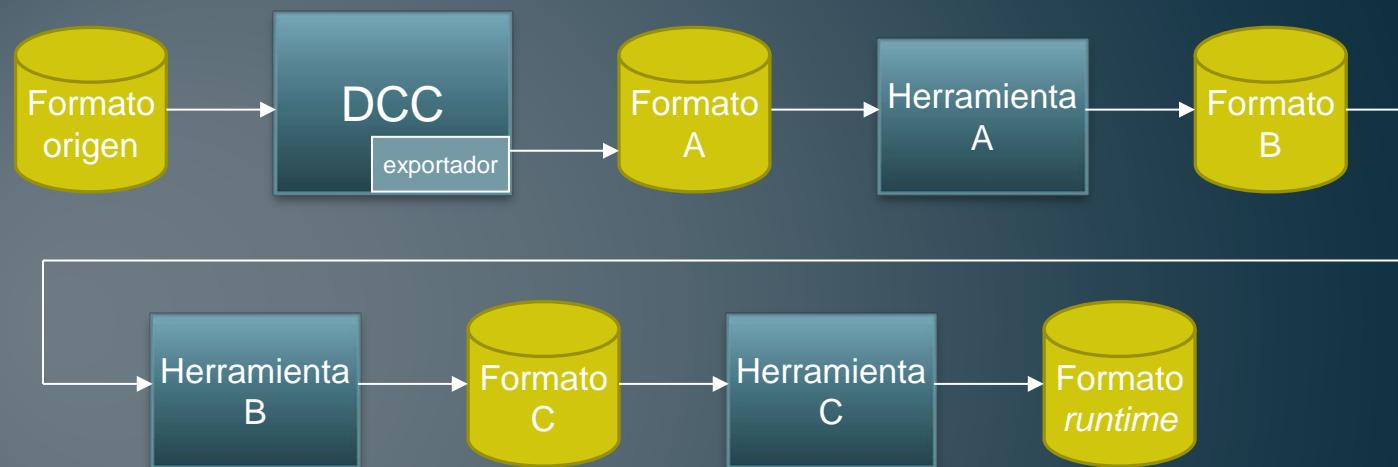
- Flujo de Trabajo



Formato DCC		Formato Runtime
Complejo		Optimizado para rendimiento en tiempo real
Cerrado		Dependiente del motor 3D utilizado
Se centra en el aspecto visual		Incluye información para comportamientos dinámicos: interacción, simulación física, IA, etc.

Herramientas

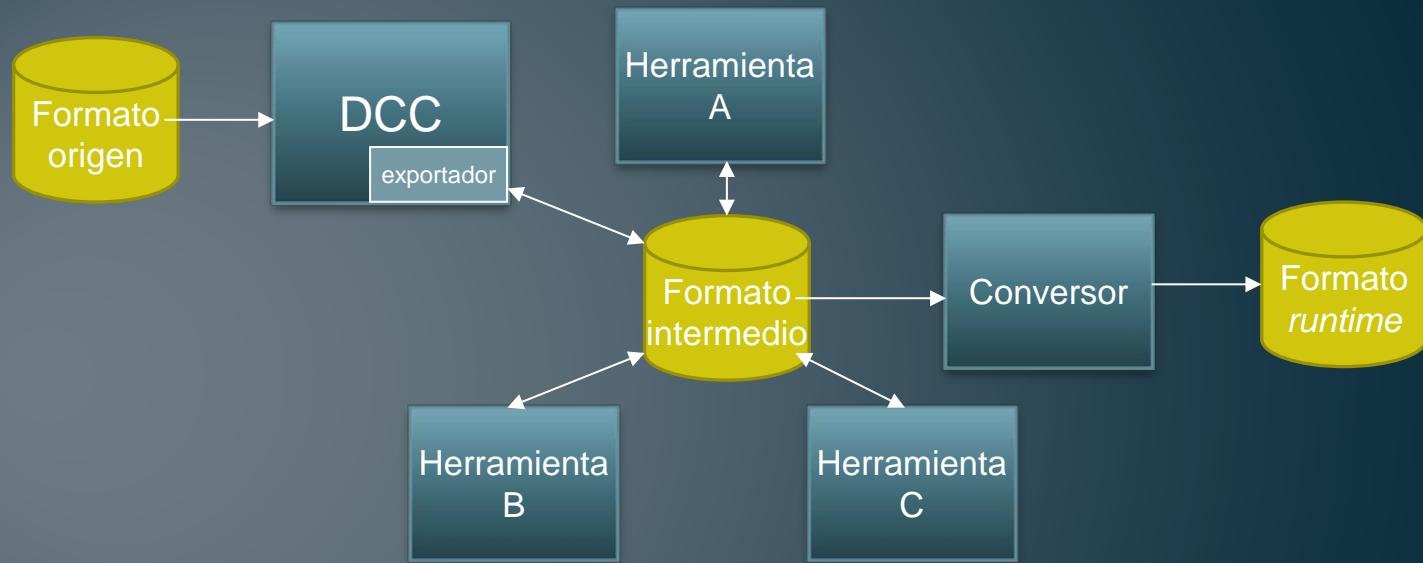
- Flujo de Trabajo



- Necesidad de estándares

- Las herramientas DCC utilizan formatos propios y cerrados.
- Cada vez que se cambia una herramienta se necesitan nuevos exportadores e importadores
- Un cambio en un punto intermedio implica pasar de nuevo por lo que queda de la cadena hasta conseguir el formato final

Solución



- Formato intermedio
 - Facilita la migración entre herramientas
 - Puede incluir información para los comportamientos dinámicos
 - Los importadores/exportadores deben conservar los contenidos del fichero

Estándares

Estándares para intercambio: **COLLADA** (formato intermedio)

Estándares para interacción: **VRML/X3D** (formato final)

Estándares para la web 3d: **WebGL**

Estándares para la integración: **x3dom**

Formatos gráficos

PARTE II-2

Desarrollo de aplicaciones gráficas 3D

Parte II: Índice

1. Introducción
2. Formatos gráficos
 - X3D (VRML)
 - Collada / KML
 - Aplicación: Google Earth
3. Tecnologías para gráficos 3D: WebGL
 - Gráficos 3D
 - La tubería gráfica
 - Shaders
 - WebGL
 - Frameworks y motores de videojuego



<http://www.web3d.org/x3d>

José Pascual Molina Massó

La nueva generación de mundos virtuales en la Web

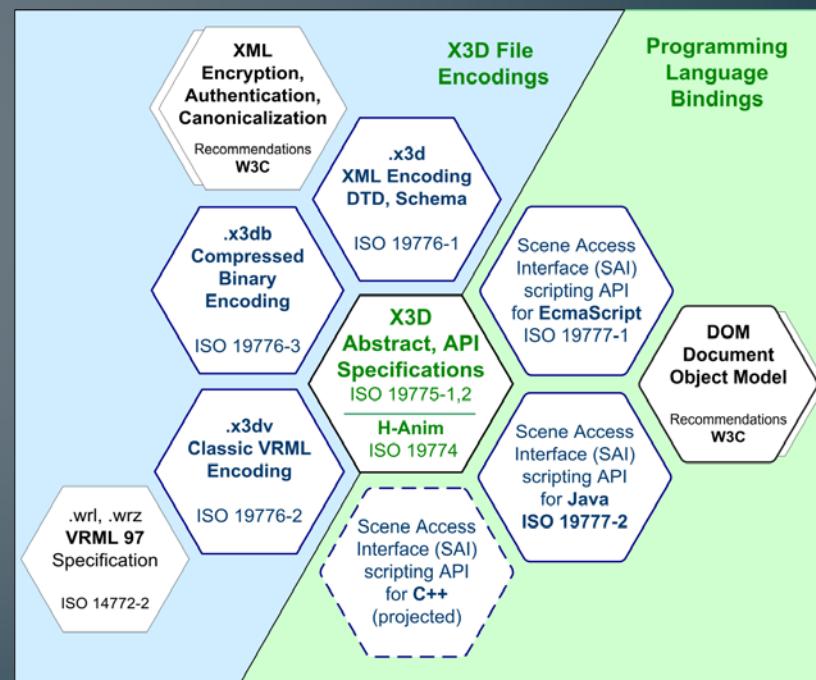
X3D

Creación de Mundos 3D

- Estándar abierto XML para comunicación de contenidos 3D
- Es un formato de fichero, pero soporta **grafos de escena**
- Soporta animación, interacción compleja, multimedia, hiperenlaces
- Diseñado para personal no experto (por encima de OpenGL/DirectX)
- Sucesor de Virtual Reality Modeling Language (VRML).
 - Nuevas capacidades en el *scene graph*
 - Humanoid animation, NURBS, GeoVRML
 - Multi-stage and multi-texture render
 - Shaders with lightmap , normalmap and CSM (Cascaded Shadow Maps)
 - SSAO (Screen Space Ambient Occlusion)
 - Optimizations for culling in the X3D scene.
 - Mayoría DCC exportan en formato VRML/X3D.
 - *Plugins* y visores *off-line* que pueden mostrarlo
 - **Poco apoyo por parte de las grandes compañías**

Introducción

- Existen cantidad de especificaciones dentro de Web3D
- Revisada y aprobada por la ISO (International Organization of Standards)

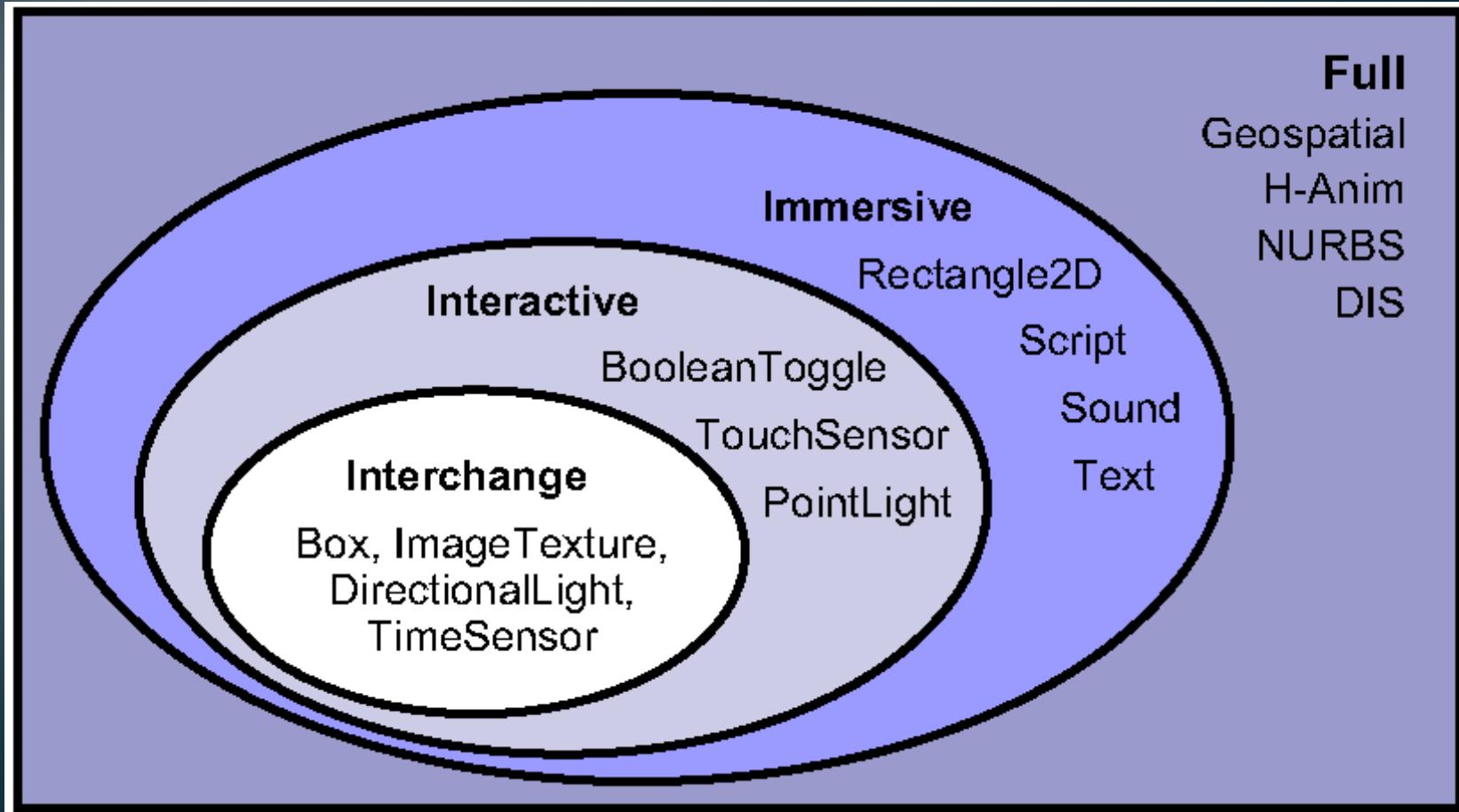


<http://www.web3d.org/x3d/specifications>

Arquitectura X3D

- X3d se estructura de forma modular permitiendo definir capas, llamadas **perfils**
 - Cada perfil agrupa a un conjunto concreto de componentes
 - Se crean en función de las necesidades de cada usuario
 - Adecuados para un grupo particular de aplicaciones
- Los **componentes** agrupan características o funcionalidades
 - geometría, apariencia, tiempo, eventos, scripts
 - Estas funcionalidades a su vez se dividen en niveles compuestos por **nodos**
- El **nodo** es el bloque básico de construcción de la escena
 - Nodos de geometría, de proximidad, de comportamiento ...

Perfiles



Perfiles

- **Core:** el conjunto más reducido.
- **Interchange:** para intercambio de geometrías, apariencia y animaciones entre diferentes aplicaciones
 - Geometrías, texturizado, luz direccional y animación
 - No incluye sonido
 - **CADInterchange:** soporte para geometrías CAD
- **Interactive:** interacción básica entre el usuario y la escena 3D
 - Añade nodos sensores para la navegación y la interacción
 - Incluye diferentes tipos de fuentes de luz
 - No incluye sonido
 - **MPEG4Interactive:** para la utilización de X3D con este otro estándar

Perfiles

- **Immersive**: mayor funcionalidad en gráficos e interacción
 - Soporte para audio
 - Colisiones y niebla
 - Permite *scripts*.
- **Full**: toda la funcionalidad de X3D
 - Incluye componentes como
 - *Geospatial*
 - *Humanoid Animation (H-Anim)*
 - *Non-uniform Rational B-Spline (NURBS)*
 - *Distributed Interactive Simulation (DIS)*

Codificación

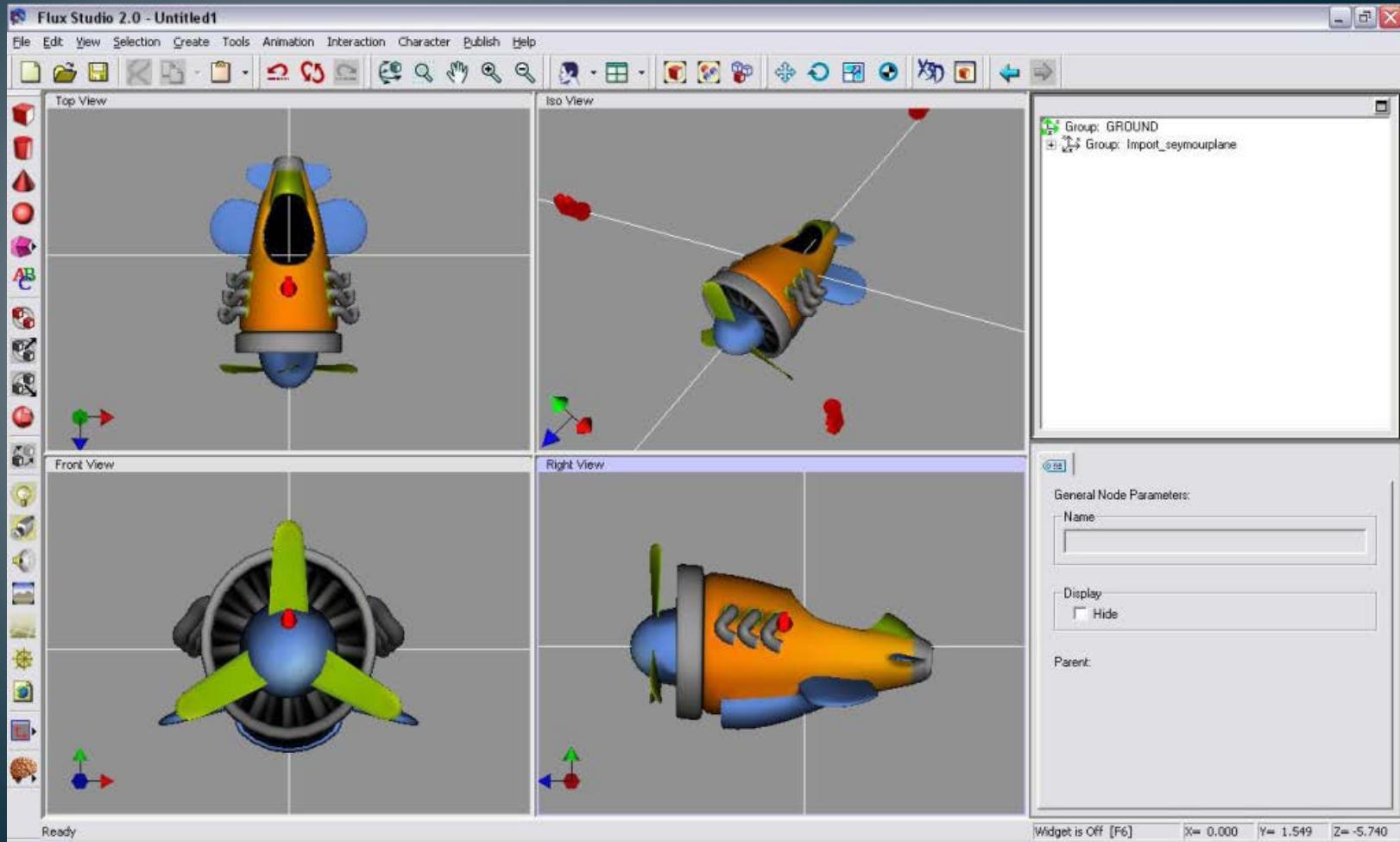
- VRML Clásica
 - Texto legible, formato con nodos anidados.
- XML
 - Texto legible, los nodos son etiquetas que contienen campos
- Binaria
 - Tamaño reducido
 - Velocidad alta de procesado
- Tipos MIME

Classic VRML	XML	Binary
x3dv	x3d	x3db
x3dvz	x3dz	x3dbz
model/x3d+vrml	model/x3d+xml	model/x3d+binary

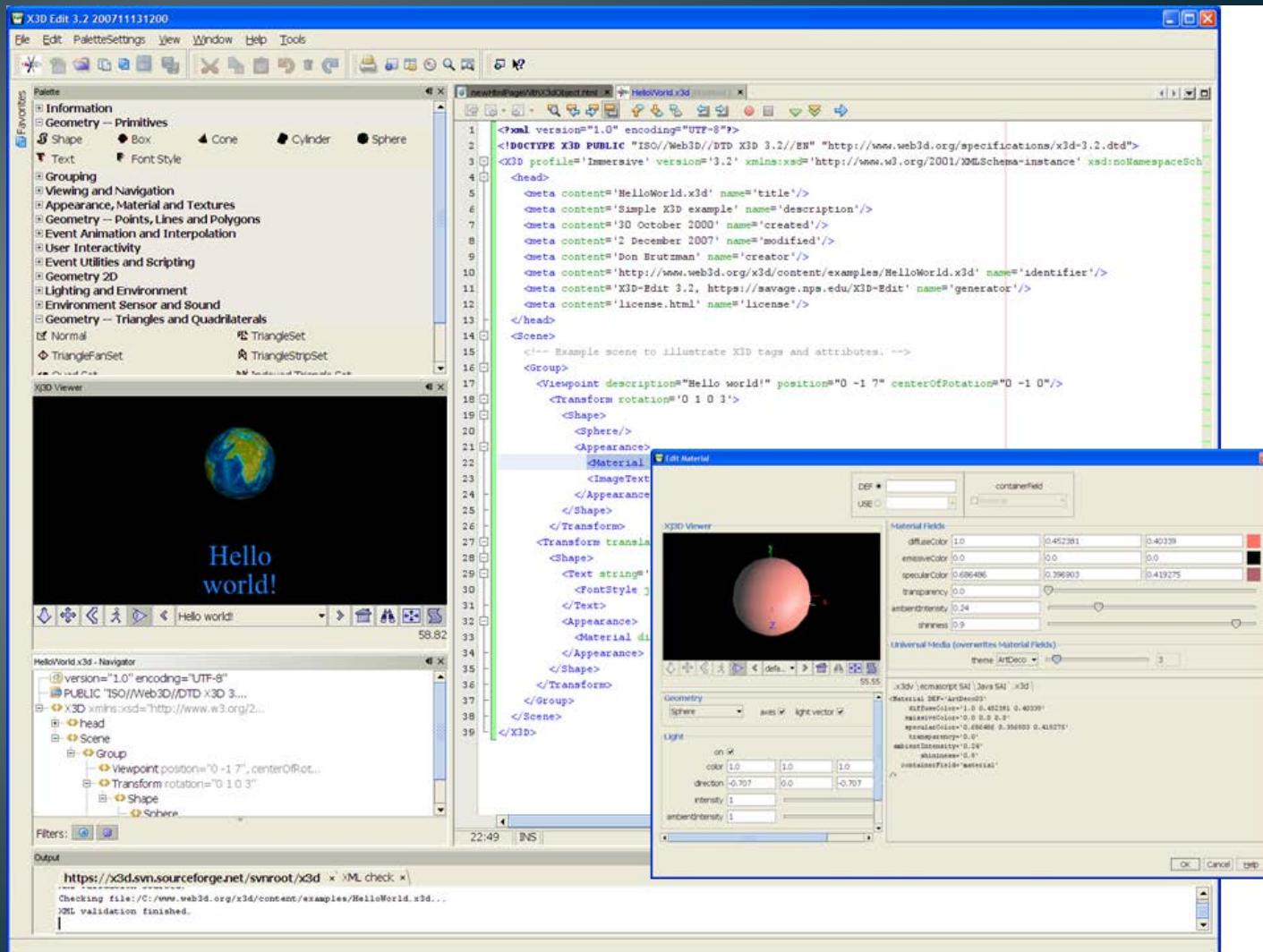
Editores, Visores y Plugins

- <http://www.web3d.org/x3d/content/examples/X3dResources.html>
- Herramientas de edición freeware:
 - [Blender \(plugin\)](#)
 - [Vivaty Studio](#)
 - [X3D-Edit](#)
- Visores
 - Castle [view3dscene](#)
 - [FreeWrl](#)
- Desarrollo
 - [Instantreality framework](#)
 - [OpenVRML](#)
- Es necesario instalar un plugin para visualizar X3D/VRML en el navegador
 - [Octaga](#)
 - [BS-Contact](#)
- **Alternativa:** El [Proyecto X3DOM](#) usa un ejecutor compatible con WebGL

Vivaty Studio



X3D-Edit



Referencias

- Especificacion
 - <http://www.web3d.org/documents/specifications/19775-1/V3.3/index.html>
- Ejemplos
 - <http://www.web3d.org/x3d/content/examples/Basic/>
 - <http://www.swirlx3d.com/tutindex.php>
 - <http://x3dgraphics.com/examples/index.php>
- Wiki
 - http://www.web3d.org/x3d/wiki/index.php/Main_Page
- Test
 - <http://www.web3d.org/x3d/content/examples/X3dResources.html>
 - <http://vrml.menteyarte.org/vbdetect.html>

Estructura de un fichero

- Cabecera del documento (XML, VRML clásico o Binario Comprimido)
- Declaración de la cabecera X3D
- Declaración del perfil (profile)
- Declaración de componentes (opcional)
- Meta-Declaraciones (opcional)
- Nodo raíz X3D
- Nodos hijos del grafo de escena X3D

Estructura de un fichero

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D
3.2//EN" "http://www.web3d.org/specifications/x3d-3.2.dtd">

<X3D profile="Interchange" version="3.2"
xmlns:xsd="http://www.w3.org/2001/XMLSchema-
instance" xsd:noNamespaceSchemaLocation="
http://www.web3d.org/specifications/x3d-3.2.xsd">
<Scene>
  <Shape>
    <Box/>
  </Shape>
</Scene>
</X3D>
```

X3D en una página Web

- Opción A: object

```
<html>
<body>
<object data="http://www.web3d.org/x3d/content/examples/HelloWorld.x3d"
        type="model/x3d+xml" height="360" width="300" ID="Object1" VIEWASTEXT>
  <param name="src" value="http://www.web3d.org/x3d/content/examples/HelloWorld.x3d"/>
  <param name="DASHBOARD" value="FALSE"/>
  <param name="SPLASHSCREEN" value="FALSE"/>
  <!-- the following anchor-link text is only shown if no X3D plugin is already installed -->
  <div class="noX3dPluginInstalled">
    <a href="http://www.web3d.org/x3d/content/examples/help.html#Applications" target="helpPage">
      Select an X3D plugin to see this example...
    </a>
  </div>
</object>
</body>
</html>
```

X3D en una página Web

- Opción B: embed

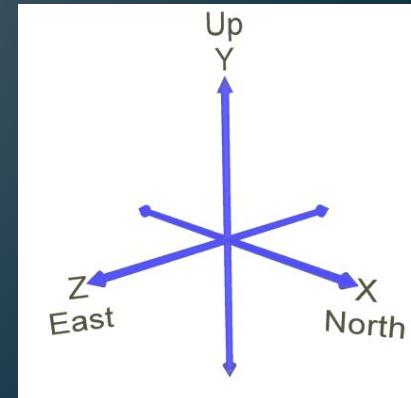
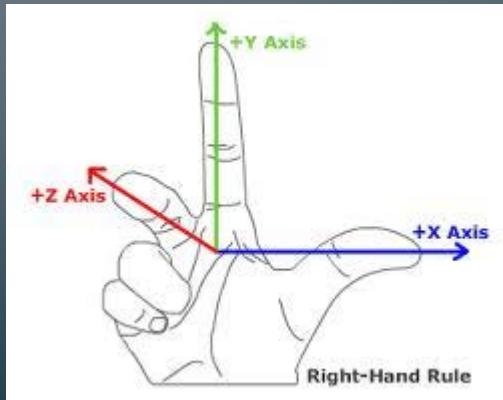
```
<html>
  <body>
    <embed src='http://www.web3d.org/x3d/content/examples/HelloWorld.x3d'
           WIDTH='300' HEIGHT='360' NAME='flux' TYPE='model/x3d+xml'
           DASHBOARD='0' LOADSCREEN='1'>
  </body>
</html>
```

Opciones del navegador para X3D

Name	Description	Type/valid range	Default
Antialiased	Render using hardware antialiasing if available	Boolean	False
Dashboard	Display browser navigation user interface	Boolean	Specified by bound NavigationInfo in content
EnableInlineViewpoints	Viewpoints from Inline nodes are included in list of viewpoints if made available by the Inline node.	Boolean	True
MotionBlur	Render animations with motion blur	Boolean	False
PrimitiveQuality	Render quality (tesselation level) for Box, Cone, Cylinder, Sphere	Low, Medium, High	Medium
QualityWhenMoving	Render quality while camera is moving	Low, Medium, High, Same (as while stationary)	Same
Shading	Specify shading mode for all objects	Wireframe, Flat, Gouraud, Phong	Gouraud
SplashScreen	Display browser splash screen on startup	Boolean	Implementation-dependent
TextureQuality	Quality of texture map display	Low, Medium, High	Medium

Primeros pasos con X3D

- Unidades de medida
 - Longitud: metro
 - Angular: radianes
 - Tiempo: segundos
 - Colores: RGB (red-green-blue) valores decimales entre (0..1)
- Sistema de coordenadas
 - Regla de la mano derecha: orden X Y Z



Figuras (Shapes)

- El componente **Shape** instancia objetos en el mundo X3D
 - Lo componen:
 - Una forma geométrica
 - Un nodo apariencia
 - Las formas estándar (*primitivas*) son representadas por los siguientes nodos de geometría:
- ```
<Shape>
 <!-- Una Geometría -->
 <Appearance> </Appearance>
</Shape>
```
- ```
<Box .../> <!-- Caja -->
<Cone .../> <!-- Cono -->
<Cylinder .../> <!-- Cilindro -->
<Sphere .../> <!-- Esfera -->
```
- A través de los valores de los atributos del nodo de geometría controlamos las dimensiones de la forma.

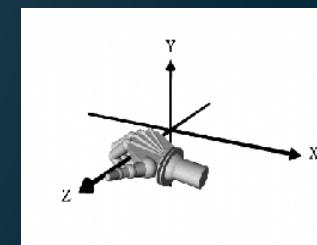
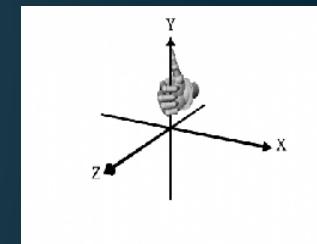
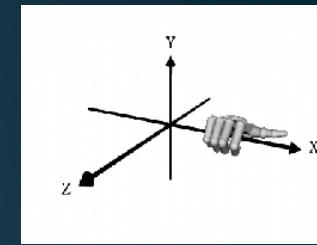
Transformaciones geométricas

- Nodo **Transform**: escalar, rotar y/o posicionar las formas con respecto al sistema de coordenadas
 - Atributo **translation**

```
<Transform translation="0 0 0">    </Transform>
```
 - Atributo **scale**

```
<Transform scale="0 0 0">          </Transform>
```
 - Atributo **rotation**

```
<Transform rotation="0 0 0 0">      </Transform>
```
- Orden de las transformaciones
 - Por defecto: Escalado + Rotación + Traslación
 - Se puede alterar anidando nodos Transform unos en otros
- **Ejercicio:** Hacer un ejemplo con las 4 figuras y aplicar alguna transformación



Apariencia: Color

- Los nodos **Appearance** modifican el color de las formas
 - Se incluye información del material o de la textura
 - Definen también las propiedades de relleno y de las líneas
 - Una forma sin Material ni Textura tiene un color difuso (0.8,0.8,0.8)
- Un nodo **Material** controla
 - El color difuso del objeto
 - La intensidad del color difuso (ambiente)
 - Los reflejos brillantes del objeto
 - El color de luz que emite
 - Su transparencia u opacidad
- **Modelo RGB (*Red, Green, Blue*)**
 - Empleado en los sistemas gráficos
 - Color = rojo + verde + azul

Color	R	G	B
Negro	1.0	1.0	1.0
Magenta	1.0	0.0	1.0
Cyan	0.0	1.0	1.0
Amarillo	1.0	1.0	0.0
Azul	0.0	0.0	1.0
Verde	0.0	1.0	0.0
Rojo	1.0	0.0	0.0
	0.0	0.0	0.0

Apariencia: **diffuseColor**

- El atributo **diffuseColor**
 - Describe el color mate de la forma.
 - Modela la **reflexión difusa**, luz que es reflejada en todas direcciones por igual, sin brillos en la superficie.
 - Afecta al color de la textura, a no ser que sea blanco

```
<Scene>
```

```
    <!-- Una esfera de color púrpura -->
```

```
    <Shape>
```

```
        <Appearance>
```

```
            <Material diffuseColor=".8 0 .8"/>
```

```
        </Appearance>
```

```
        <Sphere radius="1"/>
```

```
    </Shape>
```

```
</Scene>
```



- El atributo **ambientIntensity**

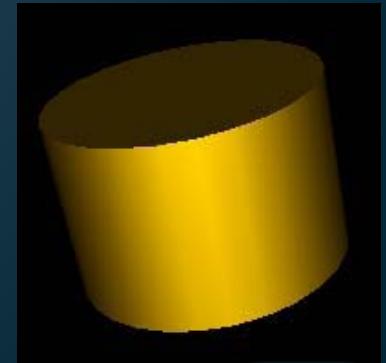
- Modela la intensidad de luz de ambiente que refleja el objeto
- En función del mismo cambia la intensidad del color difuso del objeto

Apariencia: specularColor

- Los atributos **specularColor** y **shininess**.
 - Permiten modelar los reflejos brillantes de las superficies metálicas.
 - **specularColor**: color del brillo
 - **shininess**: tamaño del brillo

Superficie	diffuseColor	specularColor	shininess
Aluminio	0.37 0.37 0.37	0.89 0.89 0.89	0.13
Plástico Azul	0.20 0.20 0.70	0.85 0.85 0.85	0.15
Cobre	0.30 0.10 0.0	0.89 0.79 0.0	0.8
Oro	0.49 0.34 0.0	0.89 0.79 0.0	0.13

- **Ejercicio:** Hacer un cilindro de color oro.



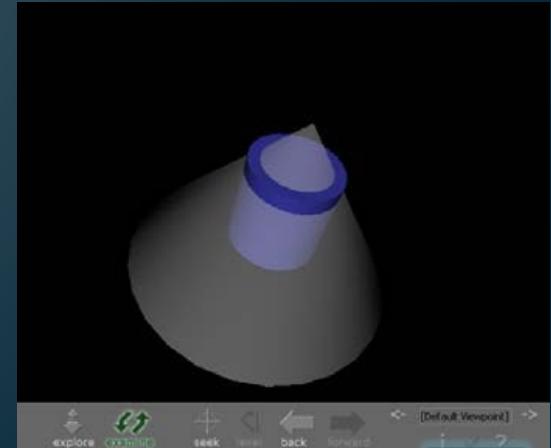
Apariencia: emissiveColor

- El atributo **emissiveColor**
 - Para crear formas que simulan fuentes de luz en su interior: bombillas, estrellas, ...

```
<!-- Un cono que emite luz -->
<Shape>
  <Appearance>
    <Material emissiveColor="0 .75 0"/>
  </Appearance>
  <Cone height="3" bottomRadius=".75"/>
</Shape>
```



- El atributo **transparency**
 - Para crear objetos transparentes o translúcidos: ventanas, gafas, jarras, ...
 - Formas opacas 0 (valor por defecto), transparencia total 1.

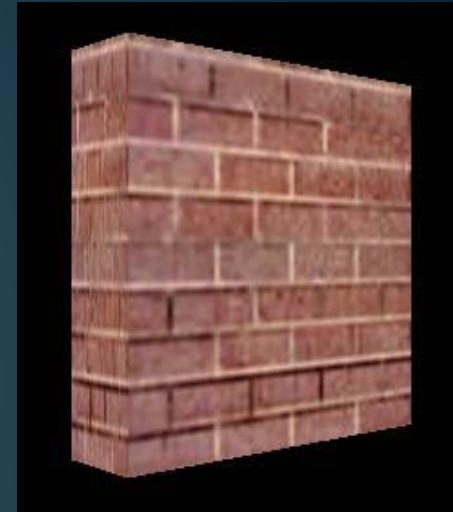


Apariencia: Texturas

- Se emplea el campo **texture** del nodo **Appearance**
- Los nodos que permiten especificar texturas tienen como base el tipo abstracto **X3DTextureNode**
 - ImageTexture
 - MultiTexture
 - MovieTexture
 - PixelTexture
- **textureTransform** controla cómo la textura se aplica sobre la forma.
 - Posición
 - Rotación
 - Escalado (factor de repetición) de la textura.
- Los colores de una textura sólo se verán afectados por las luces de la escena si se añade al nodo **Appearance** un nodo **Material**

Apariencia: Texturas

```
<Scene>
  <Shape>
    <Appearance>
      <ImageTexture url="brick_8.jpg"/>
    </Appearance>
    <Box size="4 4 1"/>
  </Shape>
</Scene>
```

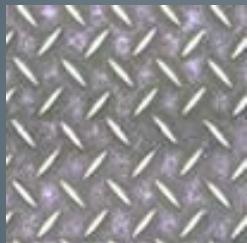


```
<Scene>
  <Shape>
    <Appearance>
      <ImageTexture url="brick_8.jpg"/>
      <TextureTransform scale=".5 .5"/>
    </Appearance>
    <Box size="4 4 1"/>
  </Shape>
</Scene>
```



Apariencia: Multitextura

- Consiste en superponer varias texturas sobre una forma.
- El nodo se llama **MultiTexture**
- **MultiTextureTransform** ajustar las texturas sobre la forma



```
<Scene>
<Shape>
  <Appearance>
    <MultiTexture mode="MODULATE">
      <ImageTexture url="metal_1.jpg"/>
      <ImageTexture url="lightmap.jpg"/>
    </MultiTexture>
  </Appearance>
  <Box size="4 4 1"/>
</Shape>
</Scene>
```

Texto

- Un nodo **Text** describe:
 - Las cadenas de texto que se desean mostrar.
 - El estilo de fuente se define con el nodo **FontStyle**.

```
<Shape>
    <Text string=...>
        <FontStyle .../>
    </Text>
</Shape>
```

- El atributo **string** es un ejemplo de **multivalue field** (admite múltiples valores)
 - separados con espacios en blanco o comas
 - encerrados entre comillas simples o dobles.
- En este caso cada cadena será mostrada en una línea diferente.

```
<Text string=' "X3D" "3D ANYWHERE" '>
```

Ejemplo de Texto

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D profile='Immersive' version='3.2'
xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance' >
<Scene>
  <!-- Scene graph nodes are added here -->
  <Shape>
    <Text string=""X3D" "3D ANYWHERE"">
      <FontStyle style="PLAIN" size="1"/>
    </Text>
  </Shape>
</Scene>
</X3D>
```



Ejercicio

- Forma tu nombre y tu e-mail:
 - Definir un material AZUL y emplearlo para crear dos cadenas de texto con fuente Sans. Una cadena irá en negrita y la otra en cursiva.
 - Nota1: Tendrás que cambiar la justificación para que ambas cadenas no se solapen.

Rafael Martinez *Rafael.Martinez@uv.es*

Fuentes de luz

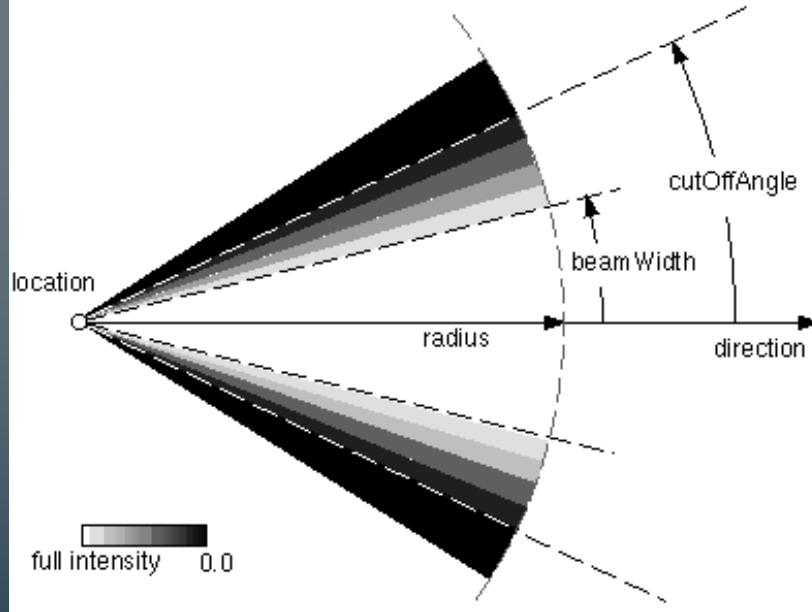
- Iluminan las formas del mundo virtual
 - El atributo **global** permite acotar el área de influencia de la luz
- No proyectan sombras
- Existe una luz por defecto añadida por el visor: el **headlight**
 - Es una luz direccional (la intensidad no varía con la distancia)
 - Apunta desde la dirección que mira el usuario
 - Se apaga mediante el atributo `headlight='false'` del campo `NavigationInfo`
- Tipos de fuentes de luz:
 - Direccional (**DirectionalLight**).
 - Puntual (**PointLight**).
 - Foco de luz (**SpotLight**).

Fuentes de luz

- **Luz ambiente**
 - Alcanza cualquier punto de la superficie de cualquier objeto
 - No depende de la posición u orientación de la fuente de luz.
 - Evita que desaparezcan los objetos en la escena por quedar en total oscuridad.
 - Es un campo que se añade a las fuentes de luz (se suma el de todas las luces)
- **Fuente de luz direccional (**DirectionalLight**)**
 - Irradia luz en una única dirección.
 - Permite simular fuentes de luz que se encuentran muy distantes (Sol)
 - La intensidad de la luz no disminuye con la distancia
 - Ejemplo: foco de luz **headlight** que el *browser X3D* proporciona.

Fuentes de luz

- Fuentes de luz puntuales
 - La intensidad de la luz disminuye con la distancia (coeficientes de atenuación) y con el ángulo de incidencia
 - El campo **radius** define el área de influencial
 - **PointLight** irradia luz desde un punto del espacio en todas direcciones
 - **SpotLight** restringe la zona de influencia a un volumen cónico



Ejemplo de fuentes de luz

- Fuente direccional



“al alba”



“al mediodía”



“al atardecer”

- Fuente puntual



Reutilización de Nodos

- Mediante **DEF/USE**

- Se asigna un nombre a un nodo o grupo utilizando **DEF**

```
<Appearance DEF="MiColor">  
    <Material diffuseColor="0 .7 0"/>  
</Appearance>
```

- Se reutiliza esa descripción de nodo con **USE** y el nombre

```
<Appearance USE="MiColor"/>
```

- **USE** no crea una segunda copia del nodo

- El nodo es insertado por segunda vez en el grafo de la escena.
 - Cualquier modificación, en el fichero o de forma dinámica, de un nodo nombrado con **DEF**, se reflejará también donde se reutilice con **USE**.

- Mediante **Inline**

- Permite componer el mundo virtual a partir de objetos descritos en diferentes ficheros.
 - En vez de shape → <Inline url="objeto.x3d"/>

Ejercicio: conjunto de árboles

- A partir de este árbol crear un conjunto de árboles (**DEF/USE**)

```
<Scene>
<!-- Un abeto-->
<Transform translation="0 .5 .0">
<Shape>
<Appearance>
    <Material diffuseColor=".5 .25 0"/>
</Appearance>
<Cylinder height="1" radius=".25"/>
<Shape>
</Transform>
<Transform translation="0 2 0">
<Shape>
<Appearance DEF="App_Copa">
    <Material diffuseColor="0 .7 0"/>
</Appearance>
<Cone height="2" bottomRadius="1.25"/>
<Shape>
</Transform>
<Transform translation="0 3 0">
<Shape>
<Appearance USE="App_Copa"/>
<Cone height="1.5" bottomRadius="1"/>
<Shape>
</Transform>
</Scene>
```



Ejercicio: bosque

- Generar un bosque a partir de abeto.x3d y cipres.x3d (Inline)

```
<!-- Un cipres -->
<Transform translation="0 .5 .0">
  <Shape>
    <Appearance>
      <Material diffuseColor=".5 .25 0"/>
    </Appearance>
    <Cylinder height="1" radius=".25"/>
  </Shape>
</Transform>
<Transform translation="0 2 0" scale="1 3 1">
  <!-- Escalar x3 en el eje vertical -->
  <Shape>
    <Appearance>
      <Material diffuseColor="0 .7 0"/>
    </Appearance>
    <Sphere radius=".5"/>
  </Shape>
</Transform>
```



Puntos de vista

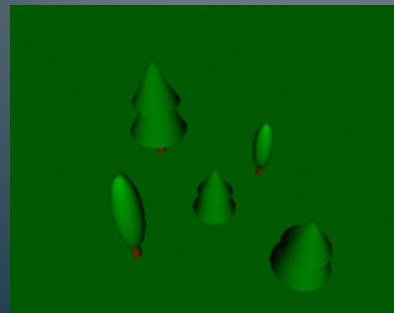
- El nodo **Viewpoint** define un punto de vista, dentro de la escena, donde se sitúa y se orienta el observador
 - **description** – cadena que se mostrará en la lista de puntos de vista.
 - **orientation** – orientación del punto de vista relativa a la orientación por defecto, que apunta en la dirección de las z negativas.
 - **position** – localización del observador.
 - **jump** – transición desde un punto de vista a otro de forma suave (False) o instantánea (True).
- El *browser X3D* presentará una lista con los puntos de vista definidos en el mundo virtual.
- El primer punto de vista encontrado será el inicial.
- También es posible definir la posición y orientación del observador anidando transformaciones.

Puntos de vista



“desde el sur”

```
<!-- Punto de vista inicial -->
<Viewpoint description="desde el sur"
    position="0 1.75 20"
    jump="FALSE"/>
<!-- Otros punto de vista -->
<Viewpoint description="desde arriba"
    orientation="1 0 0 -.785" position="0 15 12"
    jump="FALSE"/>
```



“desde arriba”

```
<!-- Posición -->
<Transform translation="0 15 12">
    <Transform rotation="1 0 0 -.785">
        <Viewpoint description="desde arriba" position="0 0 0"/>
    </Transform>
</Transform>
```

Animación

- Un modelo de eventos nos permite crear animaciones fácilmente
- Los eventos cambian la mayoría de los valores de los campos de los objetos a lo largo del tiempo:
 - posición, orientación, color, etc.
- Un evento consiste en el envío de un mensaje desde un nodo a otro
 - Animaciones a lo largo del tiempo
 - Detección y manejo de selección de objetos de la escena
 - Movimiento del usuario y detección de colisiones por la escena
- Se deben definir:
 - los instantes de comienzo y fin
 - la velocidad
- Mecanismo: Se crea una **ruta** entre nodos
 - Un sensor o disparador activa un nodo reloj (TimeSensor)
 - El TimeSensor envía valores discretos a un interpolador
 - El interpolador calcula los valores finales
 - Los valores se envían a un nodo final para modificar una propiedad del mismo

Tipos de campos

- X3D define tres tipos de campos que se pueden emplear en las animaciones a través de las rutas:
 - **inputOnly**
 - El campo es un conducto de entrada a través del cual el nodo puede recibir eventos generados por otros nodos.
 - **outputOnly**
 - El campo representa un conducto de salida a través del cual el nodo genera eventos.
 - **inputOutput**
 - El acceso al campo es total, puede ser inicializado, escrito como conducto de entrada y leído como conducto de salida.
- El campo de tipo **InitializeOnly** no se puede emplear

Conexiones con Rutas

- Las **rutas** establecen relaciones entre los campos de diferentes nodos (los que generan eventos y los que los reciben)
- Los campos deben ser del mismo tipo.
- Esta vinculación se realiza a través de la palabra clave **ROUTE**

```
<ROUTE fromNode='nodo1' fromField='output_field'  
toNode='nodo2' toField='input_field'/>
```

- Las declaraciones de las rutas se hacen después de nombrar con **DEF** los nodos origen y destino
- Con los campos **inputOutput** se toma la siguiente convención:
 - Si se usa como generador de eventos, se le añade el sufijo **_changed**
 - Si se usa como receptor de eventos, se le añade el prefijo **set_**
 - Ejemplo: **set_translation**, **translation_changed**

Construcción de la animación

1. Elegir el nodo y el campo destino a animar
2. Elegir el nombre de este nodo para la etiqueta DEF
3. Determinar el tipo de animación del campo destino
4. Determinar en función de 3. si es posible utilizar un nodo secuencia o un nodo script como fuente del evento
5. En caso de que no se emplee 4, elegir un **interpolador**
6. Elegir el nodo **sensor** disparador de la animación
7. Añadir el nodo **temporizador** con su duración y ciclicidad
8. ROUTE la salida del disparador con la entrada del timer
9. ROUTE la salida del timer con la entrada del interpolador (o secuenciador, o script)
10. ROUTE la salida del interpolador (secuenciador) con el campo de interés del nodo destino

Temporizador

- Nodo TimeSensor

- Actúa a modo de temporizador, marcando un tiempo en las animaciones o sincronizando acciones
- Permite controlar el instante de inicio y de parada, la duración del ciclo o la repetición de la animación
- Genera eventos de salida mientras está funcionando
- Puede recibir eventos de entrada para controlar su inicio
- Unidad de tiempo = segundo
 - Instantes expresados como tiempo transcurrido desde media noche del 1 de enero de 1970 (GMT).

Temporizador

- Los campos del nodo **TimeSensor** controlan:
 - **cycleInterval** – la duración del ciclo
 - **loop** – la ejecución de un único ciclo (**false**) o la repetición continua de la animación (**true**); sin no se indica nada más el valor **true** también indica que la animación comience tan pronto se haya cargado el fichero X3D.
 - **startTime** instante en el que comenzará a funcionar el temporizador
 - **stopTime** fija cuándo se debe detener el temporizador

```
<TimeSensor DEF='Timer'  
    cycleInterval='1'  
    startTime='1'  
    stopTime= '0'  
    pauseTime='0'  
    resumeTime='0'  
    loop='true' />
```

Temporizador

- El **TimeSensor** genera eventos a media que transcurre el tiempo
- Los eventos se generan a través de los campos de salida:
 - **isActive**
 - envía evento TRUE al empezar y FALSE al finalizar
 - **cycleTime**
 - envía evento tiempo al comenzar (startTime) y al comienzo de cada ciclo
 - Se emplea para enlazar con otros eventos tiempo de otros objetos
 - **fraction_changed**
 - envía continuamente floats con la fracción del ciclo actual de tiempo (valores entre 0 y 1)
 - **time**
 - envía continuamente el tiempo absoluto (segundos transcurridos desde 1 de enero de 1970)

Temporizador

- Ejemplos de uso del nodo **TimeSensor**
 - Ciclo de 3.5 segundos y repetición continua desde la carga del fichero:

```
<TimeSensor DEF="Reloj"  
    cycleInterval="3.5"  
    loop="true"/>
```

- Ciclo único de 2 segundos cuyo comienzo será fijado por algún evento:

```
<TimeSensor DEF='Reloj'  
    cycleInterval='2'  
    loop='false'/>
```

Interpoladores

- Sirven para crear los valores de la animación
- Especifican un conjunto de valores clave asociados a un conjunto de instantes dentro del ciclo de la animación
 - Los valores clave pueden ser de muy variados tipos, cada uno de los cuales tiene asociado un interpolador específico.
 - [ScalarInterpolator](#)
 - [ColorInterpolator](#)
 - [PositionInterpolator](#)
 - [OrientationInterpolator](#)
 - [NormalInterpolator](#)
 - [CoordinateInterpolator](#)
 - Durante la ejecución, el interpolador calcula los valores intermedios por interpolación lineal entre los extremos

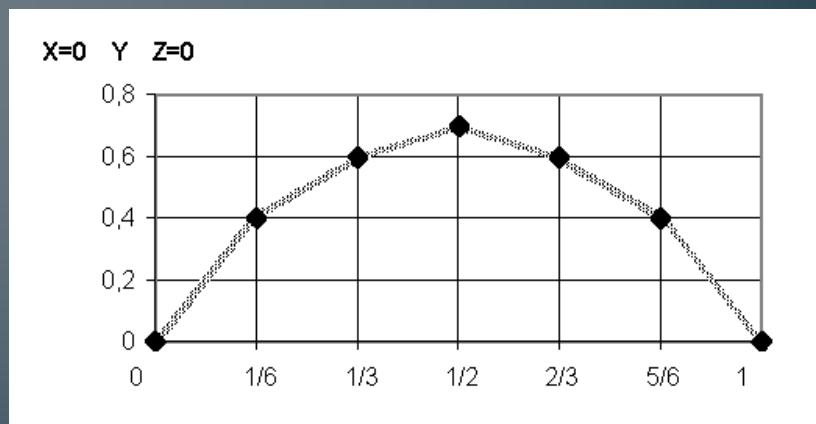
Interpoladores

- En cada interpolador se definen los siguientes campos:
 - **key** – Una lista de instantes de tiempo, con valores comprendidos entre 0 y 1, siendo 0 el inicio del ciclo de la animación y 1 el final.
 - **keyValues** – Un conjunto de valores clave asociados a los instantes de tiempo indicados en **key**.
 - **set_fraction** – El interpolador recibe eventos de entrada que representan instantes de tiempo (entre 0 y 1) dentro del ciclo de la animación.
 - **value_changed** – El interpolador genera un evento de salida con el valor calculado para el instante de tiempo recibido.

Interpoladores

- Ejemplo de trayectoria

```
<PositionInterpolator DEF="Interp_Posicion"
    key=" 0, .166, .333, .5, .666, .833, 1"
    keyValue="0 0 0, 0 .4 0, 0 .6 0, 0 .7 0, 0 .6 0, 0 .4 0, 0 0 0"
/>
```

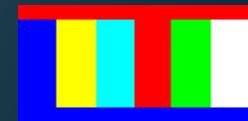
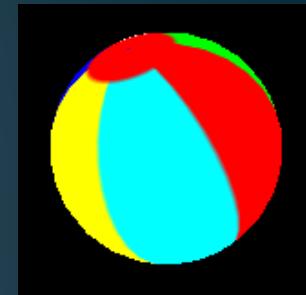


Interpoladores

- Una pelota que bota

```
<!-- Punto de vista inicial -->  
<Viewpoint position="0 .3 2"/>
```

```
<!-- Pelota -->  
<Transform DEF="Pelota" rotation="1 1 1 .785">  
  <Shape>  
    <Appearance>  
      <ImageTexture url="pelota.png"/>  
    </Appearance>  
    <Sphere radius=".25"/>  
  </Shape>  
</Transform>
```



Interpoladores

- Una pelota que bota

```
<!-- Temporizador -->
<TimeSensor DEF="Reloj" cycleInterval="3.5" loop="TRUE"/>

<!-- Interpolador de posición -->
<PositionInterpolator DEF="Interp_Posicion"
    key=" 0, .166, .333, .5, .666, .833, 1"
    keyValue="0 0 0, 0 .4 0, 0 .6 0, 0 .7 0, 0 .6 0, 0 .4 0, 0 0 0"/>

<!-- Rutas -->
<ROUTE fromNode="Reloj"   fromField="fraction_changed"
      toNode="Interp_Posicion" toField="set_fraction"/>

<ROUTE fromNode="Interp_Posicion" fromField="value_changed"
      toNode="Pelota" toField="set_translation"/>
```



Interpoladores

- Bombilla que se enciende y apaga

```
<!-- Punto de vista inicial -->
<Viewpoint position="0 .025 .15"/>

<!-- Bombilla -->
<Transform translation="0 .027 0">
  <Shape>
    <Appearance>
      <Material DEF="Mat_Bombilla"
        diffuseColor="1 1 1"
        emissiveColor="1 1 .2"/>
    </Appearance>
    <Sphere radius=".02"/>
  </Shape>
</Transform>

<Shape>
  <Appearance>
    <Material diffuseColor=".4 .4 .4"
      specularColor=".7 .7 .7"
      shininess=".6"/>
  </Appearance>
  <Cylinder height=".02" radius=".01"/>
</Shape>

<Transform translation="0 -.01 0">
  <Shape>
    <Appearance>
      <Material diffuseColor="1 1 1"/>
    </Appearance>
    <Sphere radius=".005"/>
  </Shape>
</Transform>
```

Interpoladores

- Bombilla que se enciende y apaga

```
<!-- Temporizador -->
<TimeSensor DEF="Reloj" cycleInterval="5" loop="TRUE"/>

<!-- Interpolador de color -->
<ColorInterpolator DEF="Interp_Color"
    key="0, .5, 1"
    keyValue="0 0 0, 1 1 .2, 0 0 0"/>

<!-- Rutas -->
<ROUTE fromNode="Reloj"      fromField="fraction_changed"
      toNode="Interp_Color"      toField="set_fraction"/>
<ROUTE fromNode="Interp_Color" fromField="value_changed"
      toNode="Mat_Bombilla"
      toField="set_emissiveColor"/>
```

Nodos secuencia

- Son nodos similares a los nodos interpoladores
- Se diferencian en que producen valores discretos
 - IntegerSequencer
 - BooleanSequencer
- En cada interpolador se definen los siguientes campos:
 - **set_fraction** – El secuenciador recibe eventos de entrada que representan instantes de tiempo (entre 0 y 1) dentro del ciclo de la animación. A continuación se produce un evento de salida *value_changed*
 - **value_changed** – El interpolador genera un único evento de salida discreta con el valor *keyValue[i]*. Para calcular la i, se busca el *key[i]* que es igual o mayor que el valor recibido en el evento *set_fraction*
Sucesivos eventos *set_fraction* no producen nuevos eventos a no ser que el valor suponga entrar en el siguiente intervalo (el valor debe ser mayor o igual que *key[i+1]*)
 - **next (previous)** permite avanzar el estado del nodo al siguiente (previo) estado (*key, keyvalue*) de forma cíclica

Nodos secuencia

```
<Switch DEF='ShapeSwitcher' whichChoice='-1'>
  <Shape DEF='Child0'>
    <Box/>
  </Shape>
  <Shape DEF='Child1'>
    <Cone/>
  </Shape>
  <Shape DEF='Child2'>
    <Cylinder/>
  </Shape>
  <Shape DEF='Child3'>
    <Sphere/>
  </Shape>
</Switch>

<TimeSensor DEF='Clock' cycleInterval='2' loop='true'/>

<IntegerSequencer DEF='ChildSequencer' key='0 0.2 0.4 0.6 0.8 1' keyValue='0 1 2 3 -1 0' />

<ROUTE fromField='fraction_changed' fromNode='Clock'
      toField='set_fraction' toNode='ChildSequencer' />

<ROUTE fromField='value_changed' fromNode='ChildSequencer'
      toField='whichChoice' toNode='ShapeSwitcher' />
```

Sensores

- Las escenas soportan una gran variedad de dispositivos
 - Ratón, touchpad, teclado, joysticks ...
- Los sensores detectan el movimiento de estos dispositivos
- Se utilizan para implementar escenas interactivas
 - Selección de figuras geométricas
 - Captura de pulsación de teclas o la introducción de una frase completa
 - Generación de eventos de las teclas de modificación
- Las salidas de los sensores se conectan a otros nodos a través de ROUTE
- Se emplean descripciones que alertan al usuario de la disponibilidad del sensor

Sensores de contacto

- Nodo **TouchSensor**: permite detectar la interacción del ratón sobre uno o varios objetos
- Acciones
 - **isOver** se modifica cuando se está encima de una geometría
 - **isActive** se modifica cuando se selecciona o deselecciona una geometría
 - Adicionalmente cuando se mueve el ratón, varían
 - La posición del cursor sobre la geometría
 - El vector normal
 - La coordenada de textura de la intersección
- Hace efecto sobre los grupos que están por debajo del sensor
- Para activar el sensor es necesario un cambio de posición del puntero (no se activa si lo que se mueve es la geometría)
- Se genera un evento **touchTime** cuando se pulsa el botón primario del ratón dentro de una geometría

Sensores de Arrastre

- Responden a la interacción humana
- Son invisibles (no tienen una representación gráfica en la escenas)
- Los cambios en su posición y/o rotación se pueden trasladar a otros nodos de la escena
 - Nodo **PlaneSensor**: cambia la posición en X e Y, pero no se mueve a lo largo del eje Z y no gira.
 - Nodo **CylinderSensor**: sólo gira alrededor del eje Y y modifica el ángulo, no la translación. Funciona como una articulación de bisagra (como el codo o la rodilla).
 - Nodo **SphereSensor**: cambia tanto el eje como el ángulo de rotación pero no la translación. Funciona como una articulación redonda (como el hombro o la cadera).

Sensores de Texto

- El nodo **KeySensor** proporciona un interface carácter a carácter
 - Devuelve un evento por cada pulsación o suelta de tecla
 - isActive se pone a true al pulsar y a false al soltar
 - Detecta caracteres especiales:
 - shiftKey, controlKey y altKey
- El nodo **StringSensor** adicionalmente también genera un evento cuando se detecta el final de la cadena
- No depende de la posición de ninguna geometría
- La tecla o frase introducida es un evento de tipo SFString
 - Se puede utilizar javascript para cambiar los tipos de los datos
 - Por ejemplo de SFString a MFString

X3dom (<http://www.x3dom.org>)

- Es un entorno de trabajo que permite la integración de HTML5 y contenido declarativo 3D en páginas web
- Permite incluir elementos X3D como parte de un árbol HTML5 DOM
 - Permite tener una escena interactiva
 - Permite manipular contenido 3D añadiendo, eliminando o cambiando elementos DOM
- No se necesita instalar ningún complemento adicional en el navegador, a diferencia del X3D
- Soporta algunos de los eventos HTML (como “onclick”) sobre objetos 3D
- **Cuidado, hay que cerrar las etiquetas !!!!**

X3dom example

```
<!DOCTYPE html >
<html >
<head>
    <title>Hello World</title>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
    <link rel="stylesheet" type="text/css" href="x3dom.css" />
    <script type="text/javascript" src="x3dom.js"></script>
</head>
<body>
    <h1>HTML5 X3dom Hello World</h1>
    <x3d showStat="false" showLog="false" x="0px" y="0px" width="400px" height="400px">
        <scene>
            <Viewpoint position='0 0 10'></Viewpoint>
            <shape>
                <appearance>
                    <Material diffuseColor='1 0 0'></Material>
                </appearance>
                <box></box>
            </shape>
            <Inline DEF="tambor" url="TamborGirando.x3d" />
        </scene>
    </x3d>
</body>
</html>
```



Collada - COLLABorative Design Activity

<http://collada.org>

Especificación

- Es un formato de intercambio de información digital
- Basado en XML
 - Se emplea la extensión .dae (digital asset exchange)
 - Se emplea un esquema propio para el espacio de nombres
 - xmlns="http://www.collada.org/2005/11/COLLADASchema"
- Extensible
 - Facilidad para evolucionar e incorporar los nuevos avances
 - Contenido dinámico
 - Shaders
 - Física
 - Desarrollo colaborativo
 - Los importadores/exportadores no modifican los contenidos no soportados
- Validable
 - Se puede validar la sintaxis sin considerar el contenido del XML
 - Se puede verificar la calidad de los exportadores e importadores

Historia

- Proyecto formado por diferentes compañías:
 - SONY ComputerEntertainment(SCE), VicariousVision, Emdigo, Novodex
 - Discreet, Alias (Autodesk), Softimage, ATI, NVIDIA, 3Dlabs, Nokia
- Propuesto en SIGGRAPH'03. Versión 1.0 en SIGGRAPH'04
- Administrado por el Grupo Khronos (versión 1.4.0) desde 2005
 - Es un consorcio tecnológico sin ánimo de lucro
- Se adoptó como estándar en Enero de 2006
- Última actualización: Collada 1.5 (octubre 2008)
- Evoluciona hacia el glTF (más orientado al runtime)
 - Formato de transmisión compatible con WebGL, OpenGL y OpenGL ES
- Usado por:
 - OGRE, Unreal, Nvidia, Google Earth, Autodesk, Blender, OSG, SketchUP
 - Adobe Photoshop software since version CS5.
 - Unreal, ShiVa, Torque 3D, and Unity game engines
 - Second Life and OpenSimulator since October, 2010.

Características

- Geometrías con mallas (Mesh)
- Transformaciones (rotación, traslación, escalado, matrices)
- Materiales
- Luces
- Cámaras
- Instanciación
- Texturas
- Animaciones (Skinning / bones)
- Física (cuerpos rígidos, restricciones, avatares, volúmenes de colisión)
- Programas de Shaders (Cg, GLSL, GLES)
- Efectos de Shaders (FX)
- Datos externos del usuario

Formato de Archivo

- Extension “.dae” (Digital Asset Exchange)
- Extensión “.zae” (Zipped Asset Exchange)
- Puede incorporar archivos incrustados (.zip, .rar, .kmz, .zae)
- Puede incorporar nodos externos (node proxies)
 - Escenas jerárquicas
 - Niveles de Detalle (LoD)
 - Bounding boxes
 - Carga progresiva o en segundo plano
 - Streaming

Ejemplo

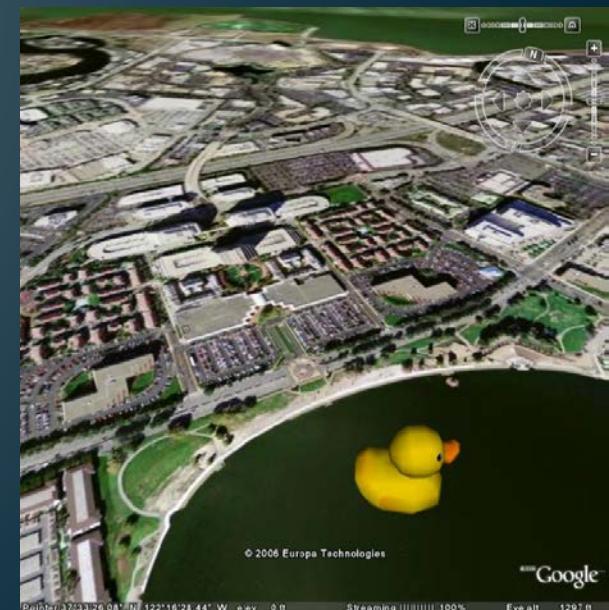


PS3 ducky - Imagen cortesía de Sony Computer Entertainmen

GoogleEarth

- Google Earth ha adoptado COLLADA para sus modelos 3D
 - Es posible arrastrar un fichero .dae sobre GE
 - Las animaciones, shaders, etc aún no se han incorporado
 - Las aplicaciones que soportan GE ahora también soportan COLLADA (**SketchUp**, RealViz, Autocad, Photomodeler, ...)
 - Todos los modelos 3D de GE están ahora en COLLADA

<http://earth.google.com/intl/es/gallery/index.html>
<http://sketchup.google.com/3dwarehouse/>

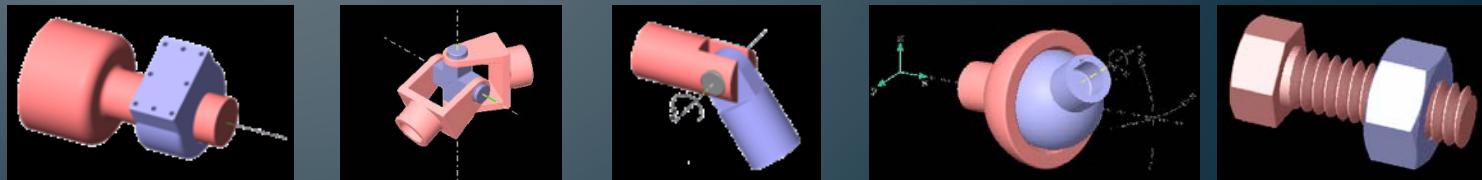


Collada Core

- B-rep (boundary representation) describe objetos CAD
 - Se describe exactamente el modelo original
 - Más simple que una mesh → Apropiado en objetos muy complejos



- Kinematics
 - Simulación cinemática de objetos mediante articulaciones y enlaces



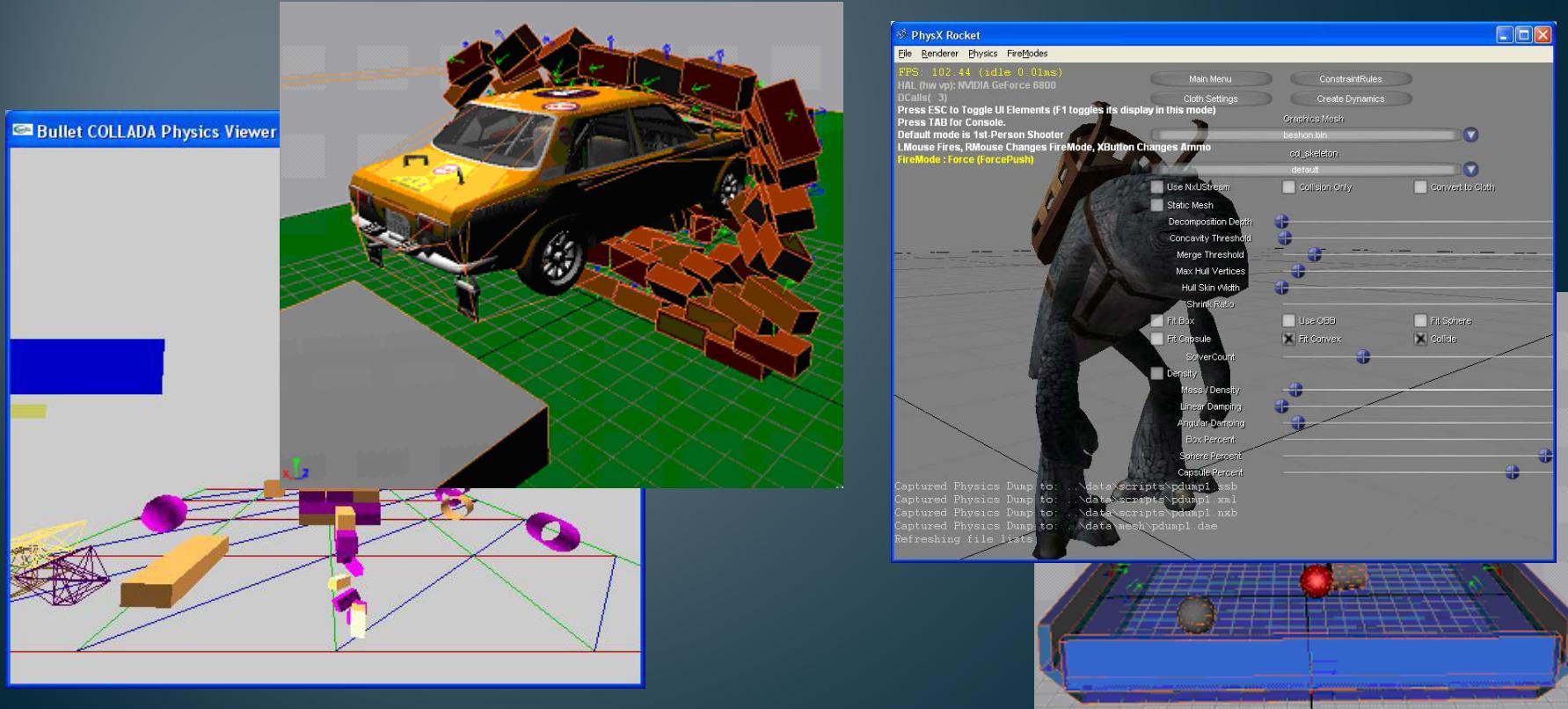
- Geolocalización de Contenidos en aplicaciones GIS
 - Se referencian mediante latitud, longitud y altura

COLLADA Visual FX

- Permite introducir efectos de visualización avanzada a las escenas
- Nuevo perfil para interactuar con otras tecnologías
 - OpenGL Shading Language, Cg and CgFX, DirectX FX
- Características:
 - Permite combinar varios shaders en un único fichero
 - Permite compartir variables entre varios shaders
 - Renderizado multipasada
 - Manejo avanzado de imágenes y texturas
- Soporte para OpenGL ES (móviles)

COLLADA Physics

- Detección de colisiones
- Comportamientos físicos en tiempo real
 - Cuerpos rígidos, uniones/restricciones, volúmenes de colisión
 - Habilita el intercambio de datos entre: [IBullet](#), [ODE](#), [PAL](#) and [PhysX](#)

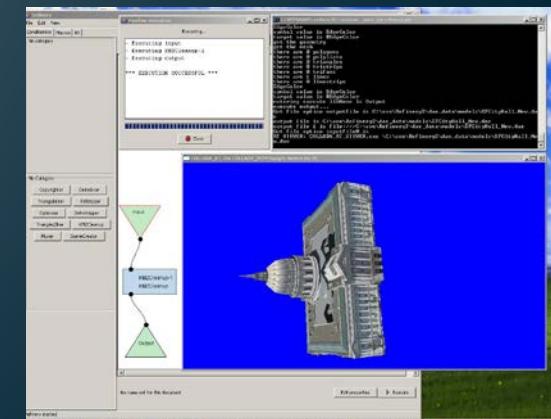


Ejemplo de Documento .dae

```
<?xml version="1.0" encoding="utf-8"?>
<COLLADA xmlns="http://www.collada.org/2005/11/COLLADASchema" version="1.4.0">
    <asset>
        <contributor>
            <authoring_tool>SketchUp 5.0 Collada exporter v1.0</authoring_tool>
        </contributor>
        <unit name="inches" meter="0.0254"/>
        <up_axis>Z_UP</up_axis>
        <Location>
            <longitude>-0.114086084916</longitude>
            <latitude>51.515803220526</latitude>
            <altitude>20.039941014546</altitude>
        </Location>
        <Orientation>
            [extension of COLLADA <up_axis>]
            <heading>0</heading>
            <tilt>0</tilt>
            <roll>0</roll>
        </Orientation>
        <Scale>
            [extension of COLLADA <unit>]
            <x>1</x>
            <y>1</y>
            <z>1</z>
        </Scale>
    </asset>
</COLLADA>
```

Herramientas: Collada Refinery

- **Herramienta para editar contenido COLLADA**
 - Permite crear y comprobar un pipeline de contenido 3D
 - Utiliza modulos llamados acondicionadores (conditioners)
 - Cada acondicionador es una función en C++
 - Funciona en lotes o de forma interactiva con una interfaz de usuario basada en Java
 - Permite la combinación de los conditioners o su uso como macros
 - Un acondicionador es el “test de coherencia” (Coherency test)
 - Sirve para validar el contenido XML
- **Ejemplos de acondicionadores**
 - Triangularización
 - Optimización (T-mesh)
 - Eliminación de ficheros (kmz files)
 - Conversión (1.4.0 -> 1.4.1)
 - Cambio de ejes (X_UP -> Y_UP)



Herramientas Open Source

- **COLLADA dom**
 - API C++ que permite la carga del documento
 - Incluye
 - COLLADA RT – ejemplo de RunTime / visor (PC (OpenGL), PS3 (PSGL)...
 - COLLADA FX – Cargador de efectos con shaders
 - <http://collada-dom.sourceforge.net/>
- **OpenCOLLADA**
 - (<http://opencollada.org>)
 - Permite usar COLLADA con otras herramientas de edición de contenido
 - 3ds Max y Maya

Test de Conformidad

- Es el *Collada Conformance Test Suite*
- Está compuesto por un conjunto de tests (613)
 - Permiten comprobar el correcto funcionamiento de los exportadores e importadores de las aplicaciones
- Funcionalidades que se comprueban:
 - Lo completo que es el soporte para cualquier característica
 - Lo robusta que es la aplicación cuando existen datos inválidos
 - Si las imágenes y los vídeos se ven correctamente
 - Si todas las características se mantienen durante un ciclo de carga y almacenamiento
- Se necesita firmar un contrato para emplearlo
- Existe un [Tutorial](#)

<http://www.khronos.org/conformance/implementers/collada/>

COLLADA vs X3D

- ***COLLADA es un formato intermedio***
 - Define el formato del contenido 3D pero no la semántica de ejecución
 - Permite la transformación de los assets
 - Desde herramientas de edición que usan descripciones de alto nivel
 - Hasta aplicaciones que requieren descripciones optimizadas específicas
 - Facilita el intercambio de datos → Se pueden usar más herramientas específicas de una forma más sencilla
- ***X3D es un formato de transmisión***
 - Se focaliza en la visualización de los contenidos
 - Está dirigido principalmente a aplicaciones Web
 - Especifica comportamientos e interacciones. Incluye:
 - Un *run-time* que permite seleccionar, ver, navegar y scripts
 - Una API que manipula el grafo de escena en tiempo real.

X3D empieza donde COLLADA termina

Referencias

- COLLADA <http://www.collada.org>
- Khronos Group <http://www.khronos.org/collada>
- Wiki <http://www.collada.org/mediawiki>
- Herramientas DCC que soportan Collada
 - [SketchUp](#) – Free DCC tool Import/export
 - [Vivaty Studio](#) – Free DCC tool, Import/export
 - [3dsMax](#) – Free, Open Source Import/export
 - [Maya](#) – Free, Open Source Import/export
 - [XSI](#) – Included, Source included in SDK, import/export
 - [Blender](#) – OSS DCC tool, Free, Open Source, Import/export
 - [Houdini](#) – Import only for now
 - ...

Referencias

- Motores de Juegos
 - [Unreal 3D](#) (Epic games engine): usada para PS3, XBOX 360, Midway, EA, Ubisoft ...
 - [Ogre](#) (Open source)
 - [C4 Engine](#)
 - [Irrlicht Engine](#) (Open source)
 - PS3 SDK (incluye COLLADA DOM & RT)
 - ... la mayoría de juegos actuales permiten cargar modelos COLLADA
- Modelos Collada en la web
 - [3D Warehouse](#)
 - [3dVia](#)
 - [Daz](#)

Google Earth

<http://earth.google.com/>

Crear un mapa

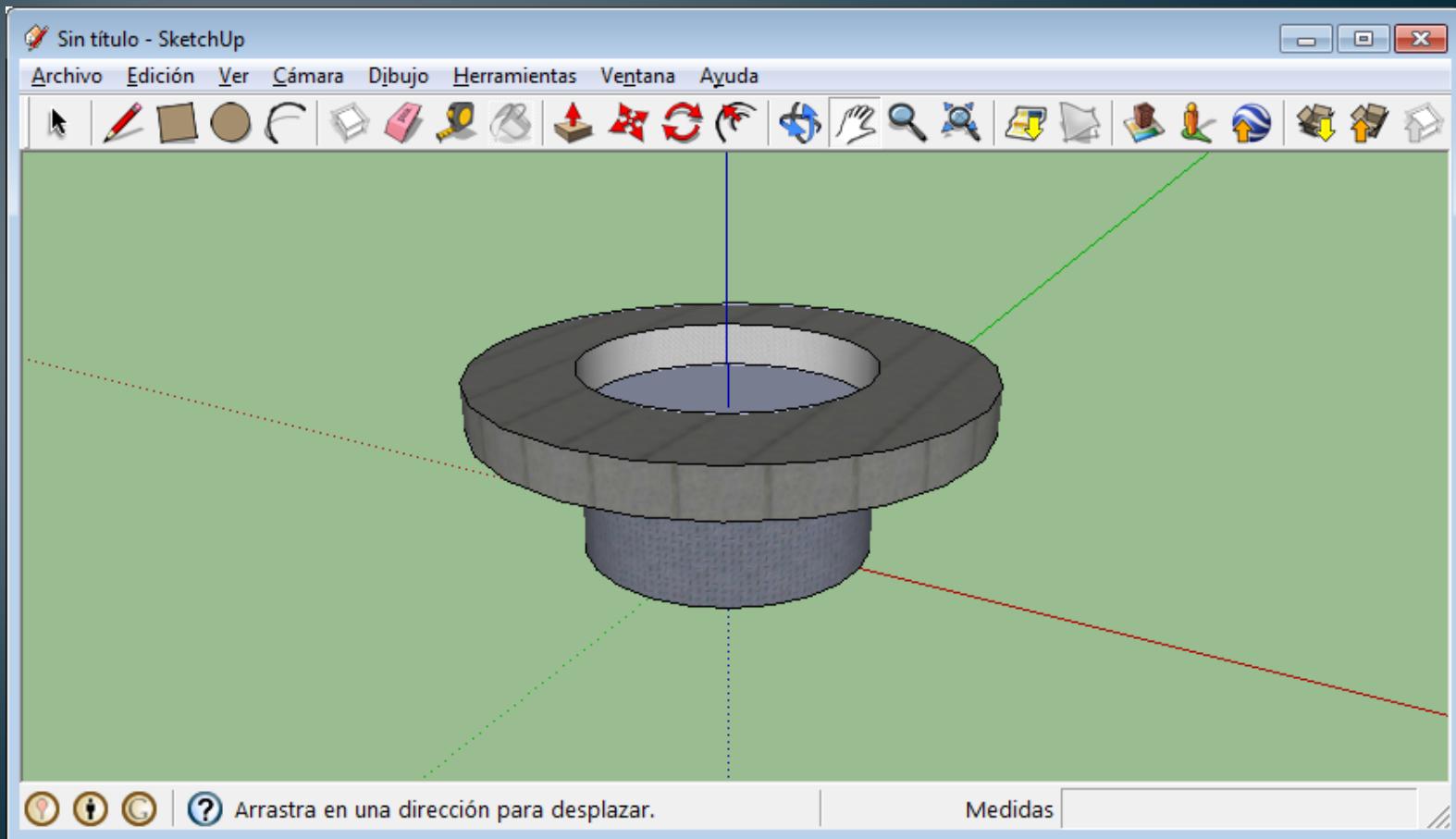
- Abrir Google Earth y viajar a algún lugar
- Añadir una **marca de posición**:
 - Pulsando en el menú “Añadir” y luego en “Marca de posición”
 - Pulsando en la chincheta amarilla de la barra de herramientas
 - Tecleando Ctrl+Shift+P
 - Se puede arrastrar con el ratón para situarla en la posición deseada
 - Se abrirá una ventana que nos permitirá definir los atributos:
 - **Nombre**: el nombre del lugar. (Ej: Torre inclinada de Pisa, “mi casa”)
 - **Latitud y Longitud**: lugar en el que se encuentre la chincheta
 - **Descripción**:
 - comentario que aparece al pulsar sobre la chincheta.
 - texto sencillo, texto formateado, imágenes, enlaces, tablas

Crear un mapa

- **Estilo, color:**
 - **Color de la etiqueta:** color del texto que acompaña a la chincheta
 - **Escala:** para aumentar o disminuir el tamaño del nombre de la chincheta.
 - **Opacidad:** opacidad de la etiqueta (a menos opacidad, más “transparente” es el texto) .
 - **Color del Icono:** color de la chincheta.
 - **Escala y opacidad:** referente a la chincheta.
- **Ver:** (sobre la ubicación de nuestra chincheta)
 - **Latitud y Longitud:** cogerá los valores que hemos comentado antes.
 - **Alcance:** altura a la que situaremos la “vista”.
 - **Encabezado:** permite rotar la imagen. Está relacionado con la rotación que podemos controlar con la brújula que tenemos en la esquina derecha superior del mapa.
 - **Inclinación:** inclinación de la vista.
- **Altitud:** altura queremos que aparezca la chincheta. Por defecto es 0 metros (pegada al suelo).
- Click con el botón derecho: Guardar como .kml
- También se puede añadir un modelo COLLADA (.dae)
 - Menú Añadir modelo (CTRL-M)

Editores KML

- Google SketchUp <http://sketchup.google.com>



KML de un modelo

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
  <Folder>
    <open>1</open>
    <name>seymourplane</name>
    <description>Ejemplo seymourplane</description>
    <visibility>1</visibility>
    <LookAt>
      <longitude>-0.475759073153997</longitude>
      <latitude>39.4901117818682</latitude>
      <range>365.815475023847</range>
      <tilt>52.4366489124468</tilt>
      <heading>52.8900360405898</heading>
      <altitudeMode>relativeToGround</altitudeMode>
      <altitude>4</altitude>
    </LookAt>
    <Placemark>
      <visibility>1</visibility>
      <name>Model</name>
      <Style id="default"></Style>
      <Model id="plane">
        <altitudeMode>relativeToGround</altitudeMode>
        <Location>
          <longitude>-0.475759073153997</longitude>
          <latitude>39.4901117818682</latitude>
```

```
          <altitude>5.000000000000001</altitude>
        </Location>
        <Orientation>
          <heading>115</heading>
          <tilt>-15</tilt>
          <roll>0</roll>
        </Orientation>
        <Scale>
          <x>200.0</x>
          <y>200.0</y>
          <z>200.0</z>
        </Scale>
        <Link>
          <href>seymourplane_triangulate.dae</href>
        </Link>
        <ResourceMap>
          <Alias>
            <targetHref>./planeDiffuse.tga</targetHref>
            <sourceHref>./planeDiffuse.tga</sourceHref>
          </Alias>
        </ResourceMap>
      </Model>
    </Placemark>
  </Folder>
</kml>
```



KML desde una página web

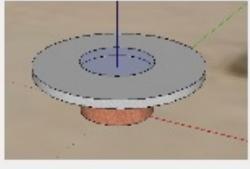
- Los ficheros kml se pueden ver con Google Earth
- En la página web hay que añadir un enlace al documento

```
<a href="mesa.kmz">  
    &nbsp;Abrir en Google Earth  
</a>
```

The image shows a screenshot of a web browser. On the left, there is a logo for 'octaga VISUAL SOLUTIONS' featuring a stylized play button icon. To the right of the logo is a section titled 'Ejemplo EGI' which contains a 3D rendering of a table model in Google Earth. Below the image is a blue link labeled 'Abrir en Google Earth'.

Ejemplo EGI

Ejemplo de visualización en Google Earth de un documento km1, que contiene un modelo collada, desde una página web

 [Abrir en Google Earth](#)

API Google Earth

- Incrustación de Google Earth en tu página web
- Instalar el complemento de Google Earth:
 - Descarga y ejecuta el [programa de instalación del complemento de Google Earth.](#)
- Documentación y ejemplos
<https://developers.google.com/earth/documentation/>



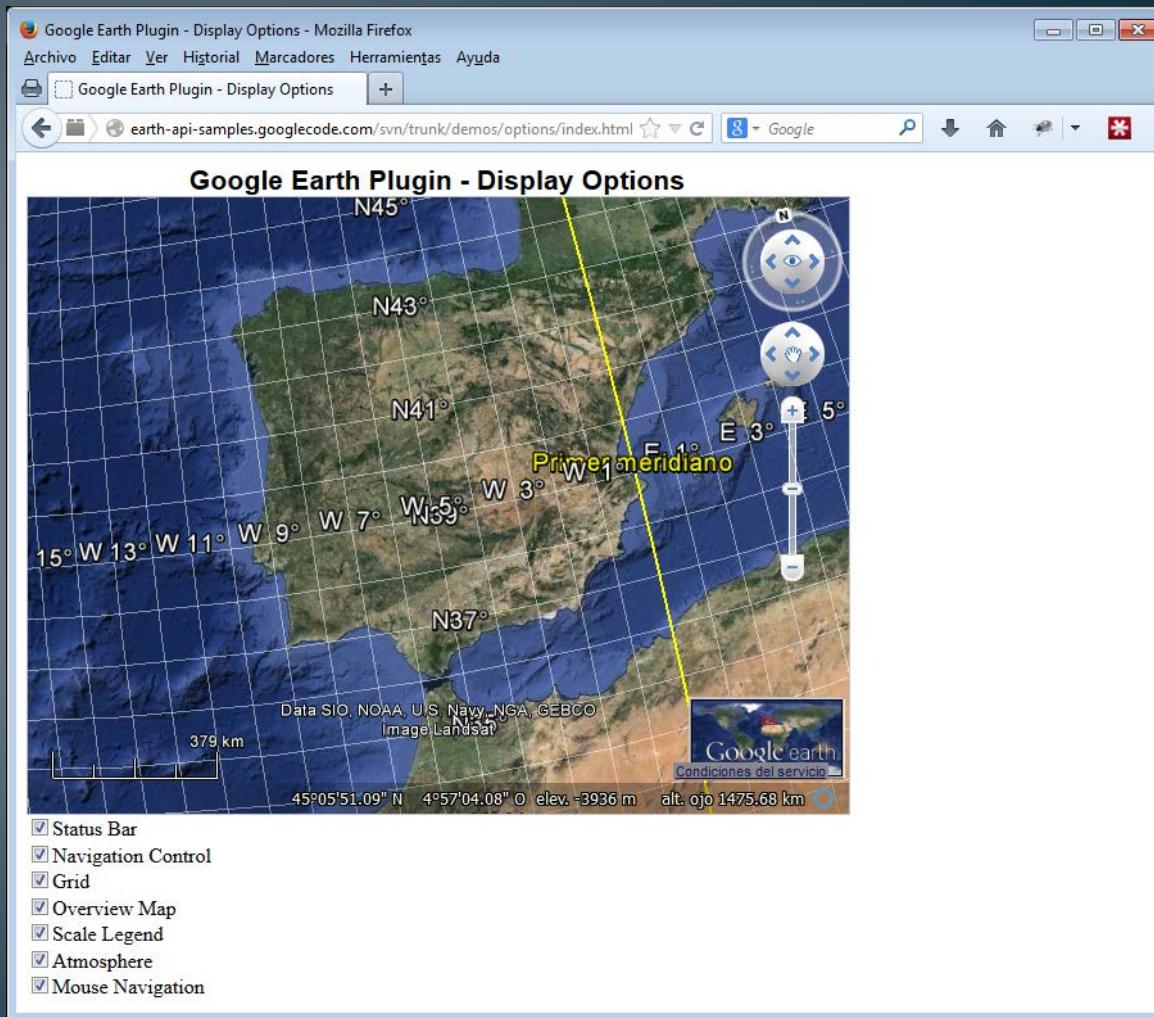
API Google Earth

```
<html>
<head>
  <title>Sample</title>
  <script type="text/javascript" src="https://www.google.com/jsapi"> </script>
  <script type="text/javascript">
    var ge;
    google.load("earth", "1");
    function init() {
      google.earth.createInstance('map3d', initCB, failureCB);
    }
    function initCB(instance) {
      ge = instance;
      ge.getWindow().setVisibility(true);
    }
    function failureCB(errorCode) {
    }
    google.setOnLoadCallback(init);
  </script>
</head>
<body>
  <center>
    <div> ¡Hola, mundo! </div>
    <div id="map3d" style="height: 400px; width: 600px;"></div>
  </body>
</html>
```

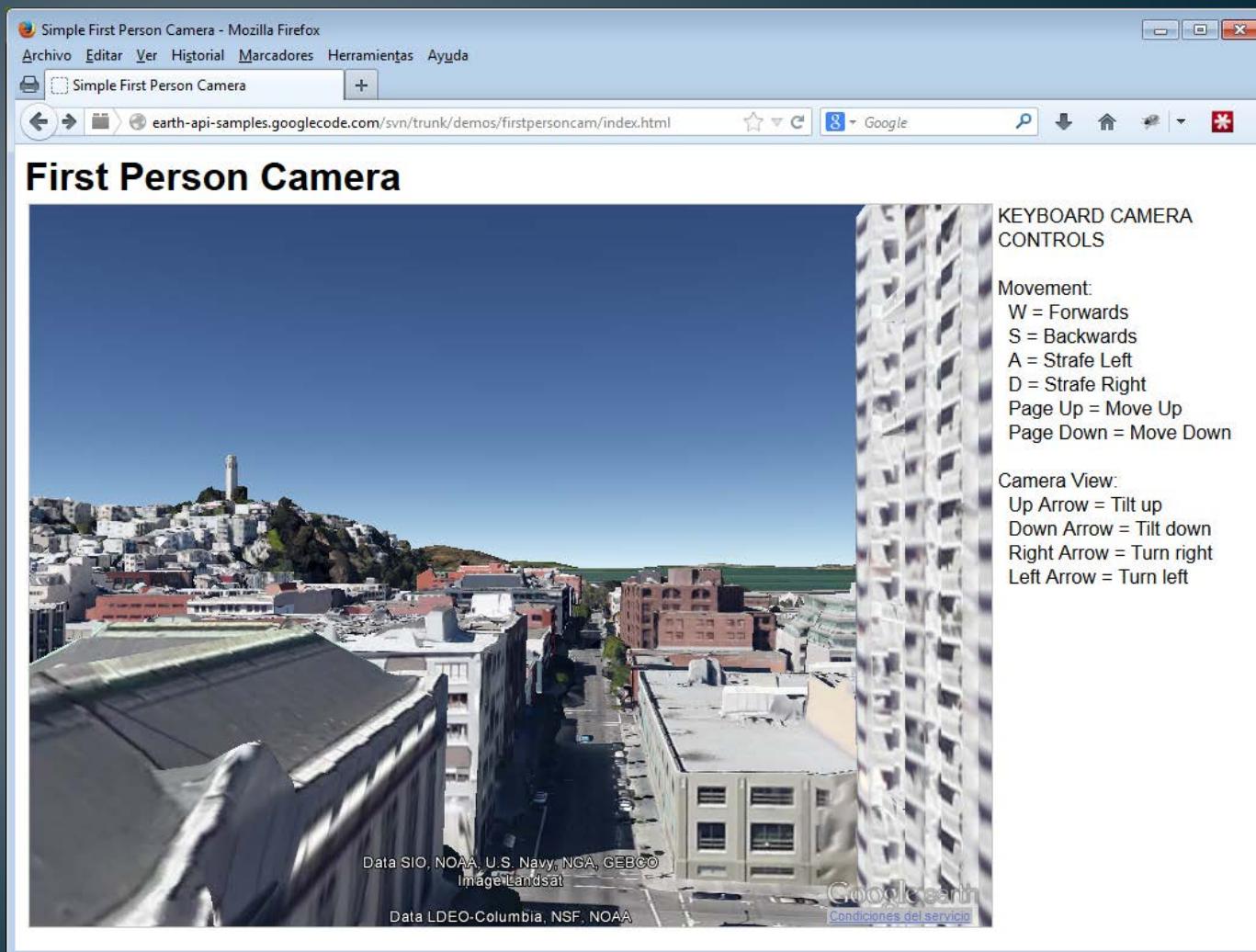
Kml desde la API de Google Earth

```
function addKmlFromUrl(kmlUrl) {  
  
    var link = ge.createLink("");  
    link.setHref(kmlUrl);  
  
    var networkLink = ge.createNetworkLink("");  
    networkLink.setLink(link);  
    networkLink.setFlyToView(true);  
  
    ge.getFeatures().appendChild(networkLink);  
}  
  
function initCB(instance) {  
    ge = instance;  
    ge.getWindow().setVisibility(true);  
    addKmlFromUrl('http://www.uv.es/rmtnez/campus.kml');  
}
```

Ejemplos de aplicación



Ejemplos de aplicación





Referencias

- [Página oficial de GE](#)
- [API de Google Earth](#)
- [Google Earth API Demo Gallery](#)
- [Foro de GE](#)
- [Galería 3D de Google](#)
- [Galería Picasso 3D](#)
- [Earth Contest](#)
- [Google SketchUp](#)
-  [GoogleEarthHacks.com](#)
-  [Google-Earth.es](#)
- [Tutorial de KML](#)
- [Tutorial de KML en español](#)
- [Ejemplos KML](#)

Ejercicio

- Crear una página web que incluya:
 - Una instancia de google earth que importe un fichero .kml con:
 - Un modelo COLLADA
 - 2 Tipos de marcas KML distintas
 - El modelo anterior en x3d (usando el X3dom)

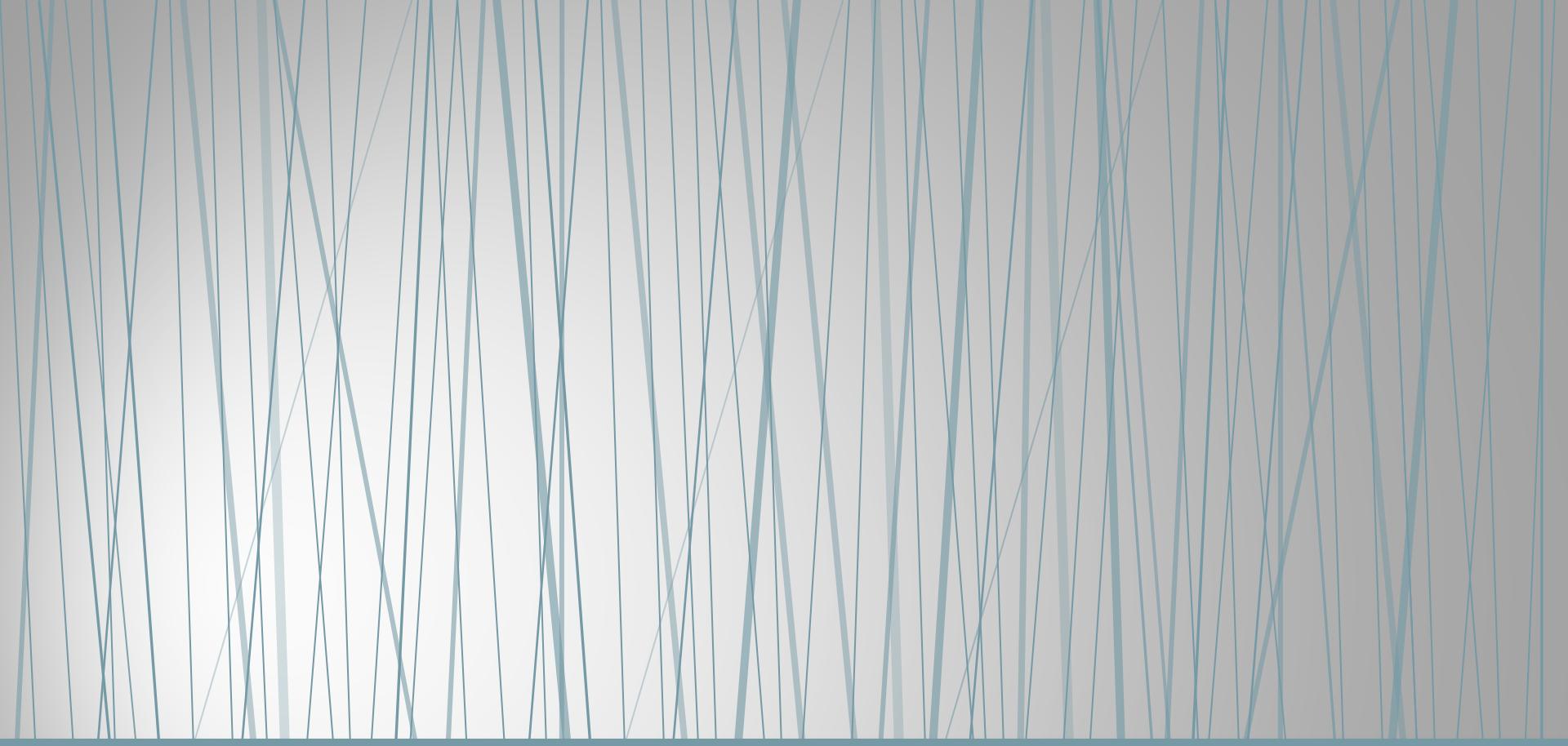
Tecnologías para Gráficos 3D

PARTE II-3

Desarrollo de aplicaciones gráficas 3D

Parte II: Índice

1. Introducción
2. Formatos gráficos
 - X3D (VRML)
 - Collada / KML
 - Aplicación: Google Earth
3. Tecnologías para gráficos 3D: WebGL
 - Gráficos 3D
 - La tubería gráfica
 - Shaders
 - WebGL
 - Frameworks y motores de videojuego



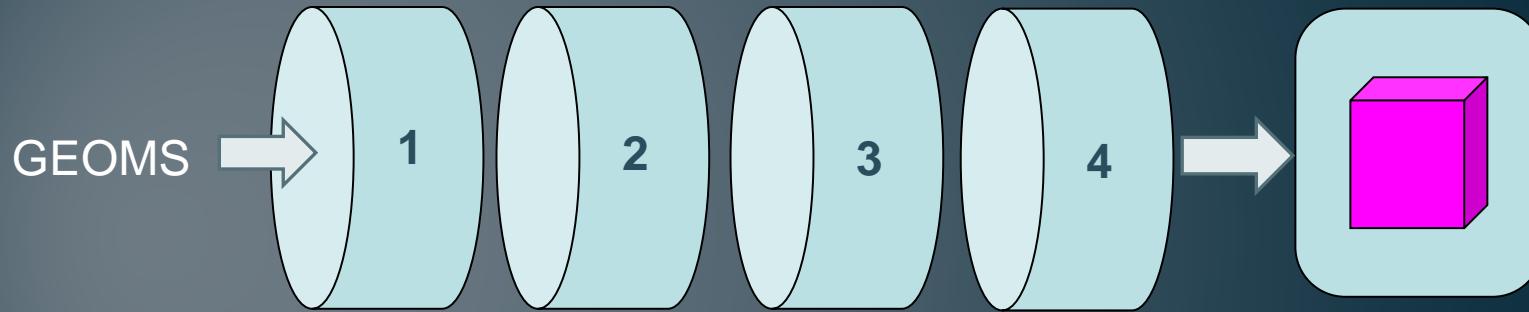
WebGL

Gráficos 3D

- Se dispone de geometrías 3D que conforman un mundo virtual
- Al mundo lo miramos desde una posición (a través de una cámara)
- Las cosas que vemos las proyectamos sobre una pantalla en 2D
- Independencia
 - Posición de los objetos en el mundo
 - Posición del observador (cámara)
 - Posición de la pantalla
- Todo ello se realiza a través de la tubería gráfica

La tubería gráfica

- Representa los procesos encadenados para visualizar “renderizar” una escena



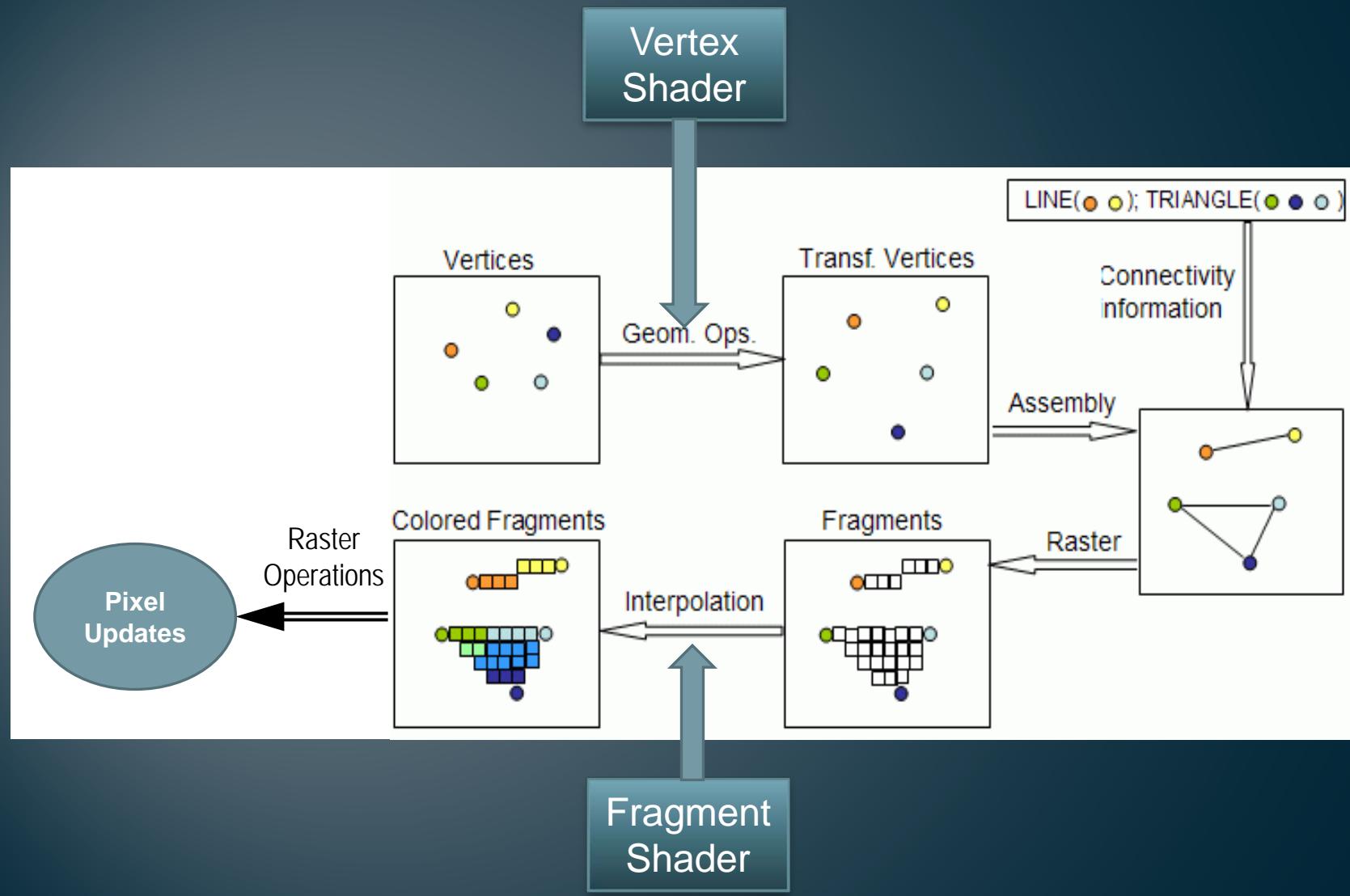
- 1 – Transformaciones afines
- 2 – Transformación de vista
- 3 – Recorte y Transformaciones de proyección
- 4 – Dibujado de primitivas (Rasterización)

(http://www.songho.ca/opengl/gl_transform.html)

(<http://db-in.com/blog/2011/04/cameras-on-opengl-es-2-x>)

(<http://www.geeks3d.com/20110704/3d-graphics-pipeline-explained>)

Estados de la tubería gráfica



Shaders: Tipos de datos y variables

- Tipos de datos:
 - Existen 4 tipos principales: float, int, bool y sampler
 - Para los 3 primeros existen tipos vectorizados de 2,3 y 4 elementos
 - Para los floats también existen matrices de 2x2, 3x3 y 4x4
- Entradas y salidas
 - **Attributes:** Diferentes para cada vértice
 - Sólo disponibles en el shader de vértices
 - Son valores de entrada de sólo lectura
 - Ejemplos: posiciones de vértices o normales
 - **Uniforms:** Valores constantes que no cambian durante la ejecución
 - Valores de entrada de solo lectura
 - Disponibles en ambos shaders
 - Ejemplos: posiciones o colores de las luces
 - **Varyings:** Se emplean para pasar datos del shader de vértices al de fragmentos.
 - Los valores para los fragmentos se calculan en función de la perspectiva
 - Son de L/E en el shader de vértices, pero sólo de lectura en el de fragmentos
 - Debe ser declarada con el mismo nombre en ambos shaders

Variables predefinidas en shaders

- *Attributes* en un shader de Vértices
 - `gl_Vertex` Vector 4D que representa la posición del vértice
 - `gl_Normal` Vector 3D que representa la normal del vértice
 - `gl_Color` Vector 4D que representa el color del vértice
 - `gl_MultiTexCoord[X]` Vector 4D que representa la coordenada de una textura
- Uniforms:
 - `gl_ModelViewMatrix` Matriz 4x4 con la matriz modelo-vista
 - `gl_ModelViewProjectionMatrix` Matriz 4x4 con la matriz modelo-vista-proyección.
 - `gl_NormalMatrix` Matriz 3x3 con la inversa de la matriz transpuesta modelo-vista
- Varyings:
 - `gl_FrontColor` vector 4D que define el color frontal de las primitivas
 - `gl_BackColor` vector 4D que define el color de fondo de las primitivas
 - `gl_TexCoord[X]` vector 4D que define la coordenada de textura X
- Salidas:
 - `gl_Position` Vector 4D con la posición final del vértice (Sh. de vértices).
 - `gl_FragColor` Vector 4D con el color final escrito en el frame buffer
 - `gl_FragDepth` float con la profundidad del fragmento escrita en el depth buffer

Shader de vértices

- Se ejecuta para cada vértice de los triángulos que se renderizan
- La entrada del shader son los datos de los vértices
 - Posición
 - Color
 - Normales
- La salida es como mínimo la variable: *gl_Position*
 - Almacena la nueva posición del vértice
- Las operaciones que se realizan en el shader de vértices son:
 - Transformación de la posición de los vértices mediante las matrices afines, de vista y de proyección (matrices *modelview* y *projection*)
 - Transformaciones de las normales y normalización
 - Generación y transformación de las coordenadas de texturas
 - Iluminación por vértice o cálculos para la iluminación por punto
 - Cálculo de los colores
- El procesador de vértices no tiene información sobre la conectividad de los mismos
 - Opera sobre los vértices y no sobre las caras
 - No es posible realizar ocultamiento de las caras de detrás (back face culling)

Ejemplo de un Shader de Vértices

```
<script id="vshader" type="x-shader/x-vertex">

uniform mat4 u_modelViewProjMatrix;
uniform mat4 u_normalMatrix;
uniform vec3 lightDir;

attribute vec3 vNormal;
attribute vec4 vTexCoord;
attribute vec4 vPosition;

varying float v_Dot;
varying vec2 v_texCoord;

void main() {
    gl_Position = u_modelViewProjMatrix * vPosition;
    v_texCoord = vTexCoord.st;
    vec4 transNormal = u_normalMatrix * vec4(vNormal, 1);
    v_Dot = max(dot(transNormal.xyz, lightDir), 0.0);
}

</script>
```

El Shader de Fragmentos

- Se ejecuta para cada pixel de los triángulos que ha transformado el shader de vértices.
- Se encarga de:
 - Calcular los colores y las coordenadas de texturas para cada pixel
 - Aplicar las texturas
 - Calcular la sombra
 - Calcular las normales de los puntos para la iluminación por pixel
- Las entradas que recibe consisten en los puntos **interpolados** calculados previamente
 - Posiciones de los vértices, colores y normales
- El resultado del shader puede ser de dos formas diferentes:
 1. Eliminar el fragmento, no produciendo ninguna salida
 2. Calcular:
 - *gl_FragColor* (el color final para el fragmento)
 - *gl_FragData* cuando se renderiza en diferentes destinos
- El shader reemplaza todo lo que se hacía antes en la tubería fija
 - El programador debe codificar todos los efectos que requiere la aplicación (texturas, niebla)
- No accede al frame buffer, por lo que aún quedan efectos por aplicar (blending)

Ejemplo de Shader de Fragmentos

```
<script id="fshader" type="x-shader/x-fragment">

uniform sampler2D sampler2d;

varying float v_Dot;
varying vec2 v_texCoord;

void main() {
    vec2 texCoord = vec2(v_texCoord.s, 1.0 - v_texCoord.t);
    vec4 color = texture2D(sampler2d, texCoord);
    color += vec4(0.1, 0.1, 0.1, 1);
    gl_FragColor = vec4(color.xyz * v_Dot, color.a);
}

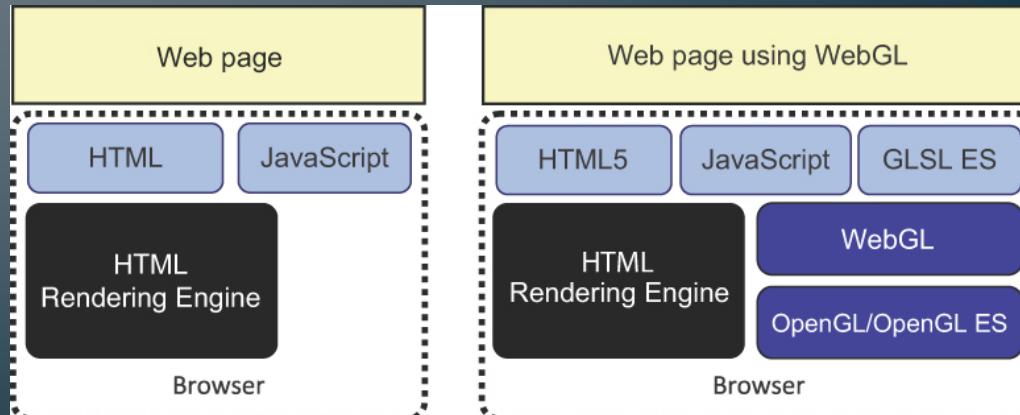
</script>
```

Depurando shaders OpenGL

- Es bastante difícil
- Algunos depuradores a considerar
 - GeeXLab (<http://www.geeks3d.com/geexlab>)
 - AMD gDEBugger (<http://developer.amd.com/tools/gDEBugger>)
 - glslDevil (<http://cumbia.informatik.uni-stuttgart.de/glsldevil>)
 - gDEBugger (<http://www.gremedy.com>)
 - NVIDIA Shader Debugger plug-in para FX Composer 2.5 (<http://developer.nvidia.com/fx-composer>)

WebGL

- WebGL es una librería para gráficos con el navegador
 - Emplea JavaScript para dotar a los gráficos de interactividad
 - Utiliza el canvas HTML5 → No se necesita plug-in
 - Está basado en OpenGL ES 2.0 → Funciona sólo con shaders !!!
- La especificación se publicó como versión 1.0 en el 2011
- WebGL está soportado por los principales navegadores
 - Google Chrome, Mozilla Firefox 4, Safari, Opera
- Existen librerías de más alto nivel que facilitan el trabajo



Ejemplos

- Para comprobar la compatibilidad del navegador
<http://webglreport.sourceforge.net>
- Ejemplos:
 - [Bouncing Mandelbrot Cubes.](#)
 - El [Google Body Browser](#).
 - [ChemDoodle 3D](#) visualiza moléculas
 - [Khronos demo repository](#)
 - [Contribuciones de usuarios en la WebGL Wiki](#)
 - [WebGL section](#) en la web de Chrome
 - <https://demos.mozilla.org>
 - <http://planet-webgl.org/>

Ejemplo: Dibuja un punto

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Ejemplo simple de punto</title>
  </head>

  <body onload="main()">
    <canvas id="webgl" width="400" height="400">
      Please use a browser that supports "canvas"
    </canvas>

    <script src="webgl-utils.js"></script>
    <script src="webgl-debug.js"></script>
    <script src="cuon-utils.js"></script>
    <script src="Point.js"></script>
  </body>
</html>
```

Ejemplo: Point.js

```
// Vertex shader program
var VSHADER_SOURCE =
  'attribute vec4 a_Position;\n' + // // attribute variable
  'void main() {\n' +
  '  gl_Position = a_Position;\n' +
  '  gl_PointSize = 10.0;\n' +
  '}';

// Fragment shader program
var FSHADER_SOURCE =
  'void main() {\n' +
  '  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);\n' +
  '}';
```

Ejemplo: Point.js

```
function main() {
    // Retrieve <canvas> element
    var canvas = document.getElementById('webgl');

    // Get the rendering context for WebGL
    var gl = getWebGLContext(canvas);
    if (!gl) {
        console.log('Failed to get the rendering context for WebGL');
        return;
    }

    // Initialize shaders
    if (!initShaders(gl, VSHADER_SOURCE, FSHADER_SOURCE)) {
        console.log('Failed to initialize shaders.');
        return;
    }
}
```

Ejemplo: Point.js

```
// Get the storage location of a_Position
var a_Position = gl.getAttribLocation(gl.program, 'a_Position');
if (a_Position < 0) {
    console.log('Failed to get the storage location of a_Position');
    return;
}
// Pass vertex position to attribute variable
gl.vertexAttrib3f(a_Position, 0.0, 0.0, 0.0);
// Specify the color for clearing <canvas>
gl.clearColor(0.0, 0.0, 0.0, 1.0);
// Clear <canvas>
gl.clear(gl.COLOR_BUFFER_BIT);

// Draw
gl.drawArrays(gl.POINTS, 0, 1);
}
```

Aprendiendo WebGL

Libro

- [WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL](#)

Tutoriales

- <http://www.jlabstudio.com/webgl/tutoriales-webgl>
- http://learningwebgl.com/cookbook/index.php/Main_Page
- <http://learningwebgl.com>
- <http://www.rozengain.com/blog/2010/02/22/beginning-webgl-step-by-step-tutorial/>
- <https://developer.mozilla.org/en/WebGL>
- <http://www.lighthouse3d.com/tutorials/glsl-tutorial>
- <http://khronos.org/webgl/wiki/Tutorial>

Depurando la aplicación

- Para depurar, en cada función WebGL se debe llamar a `getError`
- Es más sencillo incluir un script que emplea una librería
 - `<script src="webgl-debug.js"></script>`
 - Los errores aparecen en la consola JavaScript automáticamente
- También se pueden programar excepciones y manejadores

```
function throwOnGLError(err, funcName, args) {  
    throw WebGLDebugUtils.glEnumToString(err) + " was caused by call to" +  
    funcName;  
};  
ctx = WebGLDebugUtils.makeDebugContext(canvas.getContext("webgl"),  
    throwOnGLError);
```

- Existe una función para convertir códigos de error a string

```
WebGLDebugUtils.init(ctx);  
alert(WebGLDebugUtils.glEnumToString(ctx.getError()));
```
- Depuración interactiva con el [WebGL Inspector](#)

Frameworks y motores de videojuego

- Existen multitud de frameworks y motores de juego
- Es difícil saber cual elegir
 - Framework: [Three.js](#)
 - Motor de videojuego: [Unity](#)
- Se puede consultar para tener una lista
 - <http://webglframeworks.org/framework-documentation/framework-list/>
 - <http://techslides.com/html5-game-engines-and-frameworks/>
 - http://www.khronos.org/webgl/wiki/User_Contributions#Frameworks