Wannabe Amazing: Una herramienta de trabajo para iluminación global

R. Montes, C. Ureña, J. Revelles, M. Lastra
Dpto. Lenguajes y Sistemas Informáticos
E.T.S. Ingeniería Informática. Universidad de Granada.
e-mail: {*rosana,almagro,jrevelle,mlastral*}*@ugr.es*

Resumen

El sistema de cálculo de iluminación global mediante trazados de rayos **GIRT** desarrollado por el grupo de Informática Gráfica del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada permite soportar el proceso completo de cálculo de iluminación realista. Nosotros presentamos una herramienta basada en Java 3D, denominada Wannabe Amazing, que surge dentro de este marco para favorecer el desarrollo rápido de escenas idóneas para la síntesis realista, ya que no es común encontrar escenas con la información necesaria en el cálculo de todos los factores que intervienen en la simulación de iluminación global. Presentaremos también un nuevo formato gráfico **GRF** (*Granada File*) para la descripción de la geometría y de la información de iluminación de la escena.

Palabras clave: formatos gráficos, reflectancia, iluminación, Java3D.

1 Introducción

Hoy en día el software de síntesis de imágenes realistas no ha entrado de lleno en el ámbito comercial, debido en gran medida, al coste computacional que conlleva el generar una imagen de calidad. Esto no significa que se esté dejando de lado, ya que son muchas las aplicaciones de estos y muchos los progresos que se están realizando. El Grupo de Investigación de Informática Gráfica del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada, lleva una línea de investigación en Visualización Realista, que comenzó con la implementación de un sistema clásico de trazado de rayos, denominado **GIRT** (*Global Illumination Ray Tracer*), [9, 10]. Posteriormente, el sistema contó con varias ampliaciones y mejoras, y se sigue trabajando en él dentro del ámbito del trazado de partículas [5]. Sin embargo, en el contexto de un proyecto más amplio de creación de sistemas usables de síntesis de imágenes

realistas se vio la necesidad de facilitar el diseño de modelos geométricos, así como la aceptación de modelos generados por programas comerciales de amplio uso entre los profesionales del sector. Dada la no disponibilidad de aplicaciones de este tipo dentro del software desarrollado por el grupo, por lo específico de sus características, se considero la necesidad de su diseño e implementación.

En concreto esta herramienta permite la conversión de datos de formatos 3D comerciales, y el diseño de modelos geométricos. Sin embargo esto puede parecer común en cualquier software gráfico. Lo que hace a *Wannabe Amazing* valioso como herramienta es que ofrece la posibilidad de enriquecer la información contenida en el modelo, añadiendo todos los datos necesarios en el proceso de simulación de la iluminación. Esto además se hace de forma que sea compatible con nuestro sistema GIRT.

En los sucesivos apartados se tratará la definición del formato utilizado para describir de forma enriquecida las escenas. Se verá que es legible por GIRT y que es totalmente extensible para cumplir las necesidades que puedan surgir en el futuro. A continuación pasaremos a describir aspectos de diseño e implementación de *Wannabe Amazing*, para terminar comentando los resultados y cómo éstos se aplican en el proceso de síntesis de imágenes.

2 Un formato de escenas adecuado

Aunque es cierto que Wannabe Amazing puede partir de una escena vacía, y generar un modelo geométrico simple, no es esta su principal función. Se diseñó para aprovechar las escenas procedentes de otras aplicaciones, que generalmente y por sus características, poseen un grado de complejidad suficiente para generar una imagen atractiva, visualmente hablando.

Se vio necesario que la herramienta aceptase escenas complejas basadas en una representación de mallas, de aquellos formatos más extendidos por su riqueza, uso, facilidad de obtención, etc. [6] y que son los estándares del modelado en 3D. Los formatos estudiados han sido:

- Kinetix 3D Studio
- Autocad Drawing eXanche Format
- Wavefront Technologies
- MultiGen OpenFlight
- Virtual Reality Modelling Language

Tras un estudio de las capacidades de cada formato [8], no se puede decir que sean lo mismo, pero sí que todas se fundamentan en la representación de la geometría mediante mallas. Partiendo de dicha información, se extiende la lista de vértices y la

lista de caras con normales a los vértices, color del vértice, irradiancia en cada vértice, así como reflectancia y otras propiedades del material.

El formato definido, **GRF** *Granada File* permite a su vez la definición y uso posterior de cada una de los atributos mencionados (color, reflectancia, etc.), mediante el nombrado de estos, evitando de esta forma el tener que repetir datos ya dados. Añadir también que es posible realizar *includes* de otros ficheros de escenas con formatos GRF.

El resultado como es lógico, es un incremento en el tamaño del fichero de texto GRF de salida. En la Tabla 1, se recoge dicha proporción, que solo en algunos casos como DXF, se convierte en disminución del tamaño.

Escena	**Tamaño**	**Obj.**	**Mat.**	**Vert.**	**Caras**	**GRF**
anhk.3ds	7.8 KB	1	3	50	96	85 KB
alambrada.3ds	72 KB	1	3	2402	1266	245 KB
cage.3ds	23.2 KB	6	1	593	954	71 KB
cart_wheel.3ds	32.6 KB	14	3	720	1400	91 KB
greek_pelekys.3ds	403 KB	4	4	9736	18714	1.276 KB
syringe.3ds	169 KB	12	8	4993	5636	521 KB
tetera.3ds	68 KB	1	1	530	1024	68 KB
doric_column.obj	137 KB	1	1	2673	5326	315 KB
exclamation.obj	14 KB	1	1	280	306	31 KB
galleon.obj	289 KB	133	1	3459	2384	2.758 KB
knoxs.obj	22 KB	1	1	552	420	165 KB
tri.dxf	3 KB	1	1	12	16	2 KB
int.flt	137 KB	116	26	1216	1044	203 KB
spinnercar.flt	399 KB	358	64	4506	4070	686 KB
veritech.flt	237 KB	86	7	1422	1669	374 KB
cage.wrl	46 KB	6	1	593	954	80 KB
alien.wrl	218 KB	54	10	1632	2796	241 KB
mono.wrl	152 KB	48	9	1904	3404	296 KB
pompano.wrl	158 KB	19	7	1207	2316	191 KB
exclamation.wrl	20 KB	2	2	280	423	43 KB
coche.wrl	10 KB	61	10	816	1002	73 KB

Tabla 1: Comparación de los tamaños para originales y formato GRF de salida

En realidad el incremento en tamaño es causado por el hecho de tratarse de un formato tipo texto, que utiliza nombres descriptivos y separadores, para hacerlo legible por el usuario y totalmente editable. Por esta razón, la diferencia de tamaño se dispara entre formatos binarios (3ds por ejemplo) y el nuestro, y es más cercano a otros formatos de texto como VRML (archivos wrl). En este sentido, las similitudes

entre VRML y GRF van más allá que simplemente el tamaño:

- Están escritos en texto plano y puede utilizarse cualquier editor de textos para su edición.
- Son multiplataforma: independientes de cualquier hardware y sistema operativo.
- Indican las propiedades del material, así como texturas.
- Los objetos pueden contener enlaces a otros documentos del mismo formato.
- Definen objetos a los que se les asigna un identificador, permitiendo ser reutilizados.

En tal caso cabe preguntarse por qué no se ha considerado GRF como una extensión de VRML o de su posible sucesor X3D (*eXtensible 3D*)[11]. La respuesta es que si bien coinciden en ciertos aspectos, ambos formatos tienen finalidades diferentes. VRML esta pensado para el intercambio 3D en la web y para ello soporta hiperenlaces (URL) que identifiquen recursos remotos y permite la inclusión de elementos multimedia (imagen de texturas, vídeo, sonido y ficheros script). GRF en cambio tiene como propósito describir suficientemente los aspectos de iluminación de un modelo de mallas. Además, era necesario disponer de un formato que permitiese incluir aspectos no presentes en otros formatos, como las BRDFs o la descripción de las irradiancias.

2.1 Descripción del formato de representación

Una vez que tenemos un modelo tridimensional cualquiera en la aplicación, podemos modificarlo, o bien exportar la escena a un formato propio, para trabajo interno. Este formato junto con la aplicación, están aun en desarrollo, por lo que posiblemente se verá ampliado en el futuro para enriquecer en la medida de lo necesario el modelo geométrico. Tanto la descripción del formato GRF como ejemplos de ficheros generados, junto a la aplicación, pueden encontrarse en la web: http://lsi.ugr.es/~rosana.

A continuación se muestra en notación BNF la descripción del formato de representación.

escena ::= *lista_bloques* .

lista_bloques ::= *bloque* | *bloque lista_bloques* | *nada* .

bloque ::= *bloque_malla* | *bloque_fuente_luz* | *bloque_include* |
bloque_transformacion | *bloque_definicion* | *nada* .

De forma general una escena va a estar dividida en secciones. Cada sección aporta una semántica extra a la dada por el conjunto de mallas que describen la geometría de los objetos de la escena. Las secciones permiten:

- Instanciación. Denominamos instanciación o uso, a la reutilización de una definición anterior. Es una forma de ahorrar espacio en el fichero y de hacerlo más legible.
- Elementos reusables. Es posible acceder a las características definidas en otro fichero GRF, mediante la inclusión de este (clásula *include*), dando la impresión de que dicha información se encontrase en el fichero actual y no en otro.
- Definiciones. Esta sección nos permite asociar el significado de un bloque a un identificador, como comodidad para su posterior uso. Se permite más de una definición elemental, aunque su ámbito es local al bloque en el que se realizó la definición.
- Asignar texturas a las caras. Es posible asignar más de una textura a la malla, mediante el uso de identificadores y de la cláusula *use*. A su vez, los vértices contendrán la información correspondiente a las coordenadas de textura para su correcta visualización.
- Realizar transformaciones a la geometría. Una operación común consistente en la rotación entorno a un eje, puede verse realizada mediante una cláusula *translation* seguida de un vector que indique el ángulo de rotación que se debe aplicar. Otras transformaciones permitidas son la escala y la traslación.

bloque_fuente_luz ::= ***light begin*** *(region)* ***end [light]***.

region ::= vector3D arista arista.

arista ::= vector3D.

bloque_include ::= ***include*** *nombreFichero*

bloque_transformacion ::= ***transform*** *transformacion* ***begin*** *lista_bloques* ***end***

transformacion ::= transformacion transformacion_elemental | transformacion_elemental

transformacion_elemental ::= ***scale*** *vector3D |* ***rotation*** *vector3D |* ***translation*** *vector3D*

bloque_definicion ::= ***def*** *definiciones* ***begin*** *lista_bloques* **end**

definiciones ::= definiciones | definicion definiciones_elementales

definiciones_elementales ::= identificador = cuerpo_definicion

cuerpo_definicion ::= definicion_textura | definicion_color | definicion_brdf

definicion_textura ::= ***texture*** *nombreFichero*

definicion_color ::= ***rgb*** *colorValue*

definicion_brdf ::= ***brdf*** *nombreFichero*

uso ::= uso_textura | uso_color | uso_brdf .

uso_textura ::= ***use texture*** *identificador .*

uso_color ::= ***use color*** *identificador .*

use_brdf ::= ***use brdf*** *identificador .*

Los elementos susceptibles a asignarles un identificador son: texturas, color y BRDFs. El identificador quedará precedido por una palabra clave que aportará semántica al identificador. Mas parecido a VRML son los casos de instanciación, que comienzan con la palabra clave *use* y le sigue la palabra empleada en la definición y el identificador. Tanto las declaraciones como en los usos, no existe ningún tipo de restricción en cuanto a orden o posición de aparición dentro del fichero de escena. El ámbito de la cláusula *use* se mantiene desde el momento de su aparición hasta el final de la malla, o bien hasta la aparición de una cláusula similar.

bloque_malla ::= ***begin mesh*** *listaVertices listaCaras* ***end mesh*** *.*

listaVertices ::= ***begin vertexs*** *vertices* ***end vertexs*** *.*

vertices ::= vertice | vertice vertices | nada .

vertice ::= [***normal*** *vector3D] [* ***irad*** *vector3D] [* ***uv*** *vector2D]*
vector3D

vector3D ::= (valor, valor, valor) .

vector2D ::= (valor, valor) .

listaCaras ::= ***begin faces*** *caras* ***end faces****.*

caras ::= [definicion_color] [definicion_brdf] [definicion_textura] cara
| cara caras | nada .

cara ::= nvertices posicion | (posicion , posicion*) .*

nvertices ::= literal_entero .

posicion::= literal_entero .

Nuestro formato expresa cada malla como una lista de vértices y una lista de caras. Se describen los vértices que van a aparecer, incluyendo normales a los vértices, color de irradiancia, así como reflectancia (directamente o bien usando el nombrado de elementos y la posibilidad de realizar *includes* de otro fichero). Posteriormente se describen las caras como una lista de vértices dados mediante un índice o posición ocupada dentro de la lista de vértices de dicha malla. Se permiten dos formas diferentes de expresar las caras, semejantes a la empleada por otros formatos. Otros elementos que aparecen en la descripción BNF son:

nombreEscena := nombreFichero .

nombreFichero ::= cadena .

colorValue ::= (valor, valor, valor) .

cadena ::= "conjunto de caracteres Unicode" .

valor ::=literal_real | literal_entero .

literal_real ::= (decimales "." [decimales] [exponente] [sufijo_tipo]) | ("." decimales [exponente] [sufijo_tipo]) | (decimales [exponente] [sufijo_tipo]) .

decimales ::= 0..9 .

exponente ::= " e" ["+" | "-"] decimales . .

sufijo_tipo ::= "f" | "d".

literal_entero ::= 0..9 .

identificador ::= a..z, A..Z, ,0..9, $, _ .

comentario ::= // cadena .

nada ::= ;

Estamos seguros de que el lector deseará ver el típico ejemplo de un cubo tridimensional expresado en el formato GRF. Su simplicidad se comprueba a su vez en el tamaño del fichero (831 bytes).

```
   // Wannabe Amazing sep 01, 2001, 12:58
// Canvas:  Width 585 Height 415 Scale 0.2
// Background Color :  0
// Image :  0
// Components:  2
// Box :  width=2.0 height=1.0 length=0.5
// nvertexs=4 nfaces=6
// Bounding Box :  Lower=(-2.797451055, -3.276146411, -2.716787639)
// Upper=(2.5823486120, 2.1036532564, 2.6630120283)
begin mesh
     begin vertexs
        normal(0.0, 0.0, 1.0)
       (2.0, -1.0, 0.5)
        normal(0.0, 0.0, 1.0)
        (2.0, 1.0, 0.5)
        normal(0.0, 0.0, 1.0)
        (-2.0, 1.0, 0.5)
        normal(0.0, 0.0, 1.0)
```

```
      (-2.0, -1.0, 0.5)
      normal(0.0, 0.0, -1.0)
      (-2.0, -1.0, -0.5)
      normal(0.0, 0.0, -1.0)
      (-2.0, 1.0, -0.5)
      normal(0.0, 0.0, -1.0)
      (2.0, 1.0, -0.5)
      normal(0.0, 0.0, -1.0)
      (2.0, -1.0, -0.5)
   end vertexs
   begin faces
      (0,1,2,3)
      (4,5,6,7)
      (7,6,1,0)
      (3,2,5,4)
      (1,6,5,2)
      (3,4,7,0)
   end faces
end mesh
```

3 Aspectos generales de Wannabe Amazing

La herramienta desarrollada para la generación rápida de escenas adecuadas para el cálculo de iluminación, está basada en **Java** y su librería gráfica **Java3D**. De todos es conocido las facilidades de programación que ofrece Java, pero no es tan sabido qué puede ofrecer Java 3D [4, 7].

3.1 Implementación

La API de Java 3D es un conjunto de clases para crear aplicaciones y applets con elementos 3D. Ofrece a los desarrolladores la posibilidad de manipular escenas complejas en tres dimensiones. La principal ventaja que presenta este API 3D frente a otros entornos de programación 3D es que permite crear aplicaciones gráficas 3D independientes del tipo de sistema. Las aplicaciones están constituidas por elementos gráficos individuales, como objetos separados que se ven conectados dentro de una estructura en árbol llamada **grafo de escena** que contiene una descripción completa de la escena, o como suele denominarse *universo virtual*.

Una vez lanzado Wannabe Amazing, este crea un universo, vacío de contenido geométrico pero con otros elementos como son el observador, las características del medio (niebla, luz ambiental), las fuentes de luz y un gestor de eventos de teclado y ratón (Behavior en Java3D).

Cuando se procede a abrir un fichero de escena de cualquiera de los formatos soportados, se realiza una identificación del formato presente para llamar al cargador adecuado. Este leerá la descripción geométrica y algunos detalles propios de dicho formato, saltándose la información que Wannabe Amazing aún no usa, pero que en el futuro podrá implementar (como por ejemplo LODs).

Para cada escena leída se añade una rama en el grafo de escena, de forma que podemos unir dos o más escenas procedentes de formatos distintos en una sola, con la opción *Añadir*. Si en su lugar se emplease *Nuevo*, se eliminarían ramas anteriores, dejando nuevamente al universo vacío de contenido geométrico antes de llamar al cargador adecuado.

Al mismo tiempo que se añade los nodos a la escena, se calculan los nuevos parámetros de visualización: caja englobante, posición del observador y planos de corte delantero y trasero. El cargador es el encargado de ir generando la descripción de la malla al tiempo que realiza la lectura del fichero de entrada, que tomara forma física cuando el usuario seleccione la opción de *Guardar*.

3.2 Opciones

La aplicación como puede observarse en la Tabla 2, está dividida en dos ventanas, la primera contiene todos los elementos de interacción como menús y botones. La segunda se utiliza para visualizar el modelo, con el que podremos interaccionar mediante las teclas el teclado y el ratón.

Con Wannabe Amazing podemos realizar cambios en las propiedades de los nodos que forman parte del grafo:

- El nodo Background puede cambiar de color (entre 16 posibles) o puede presentar una imagen escalada, centrada o en mosaico.
- El nodo ExponentialFog genera un efecto interesante en la visualización, al permitirnos añadir niebla a la escena.
- Mediante los nodos PointLight, AmbientLight y SpotLight, y sus propiedades (posición, dirección, atenuación, color, etc.), podemos editar la iluminación del modelo.
- Por su parte los nodos ColoringAttributes y PolygonAttributes permiten modificar en tiempo de ejecución las propiedades de renderizado de todos los objetos a los que se les haya asignado. Desde el sombreado de las caras, modo puntos, línea o sólido, hasta el material.
- Las primitivas geométricas de Java3D, son soportadas mediante clases equivalentes en nuestro sistema (Cubo, Cono, Cilindro y Esfera). Podemos utilizarlas en la edición de la escena, y modificar propiedades antes de exportarlas.

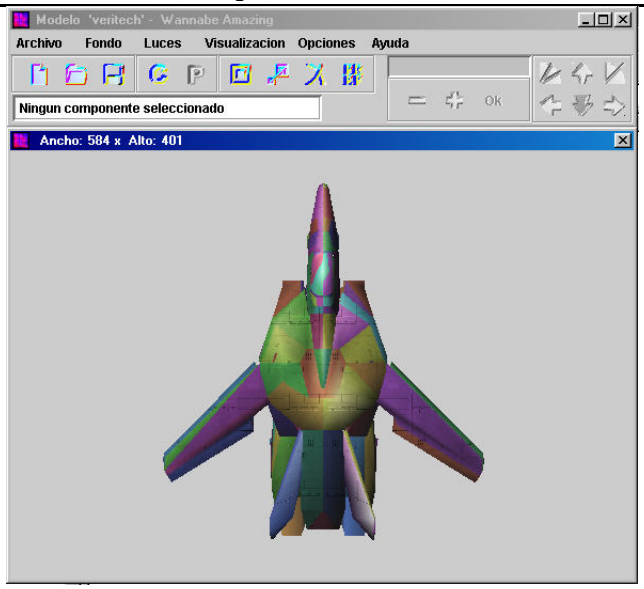
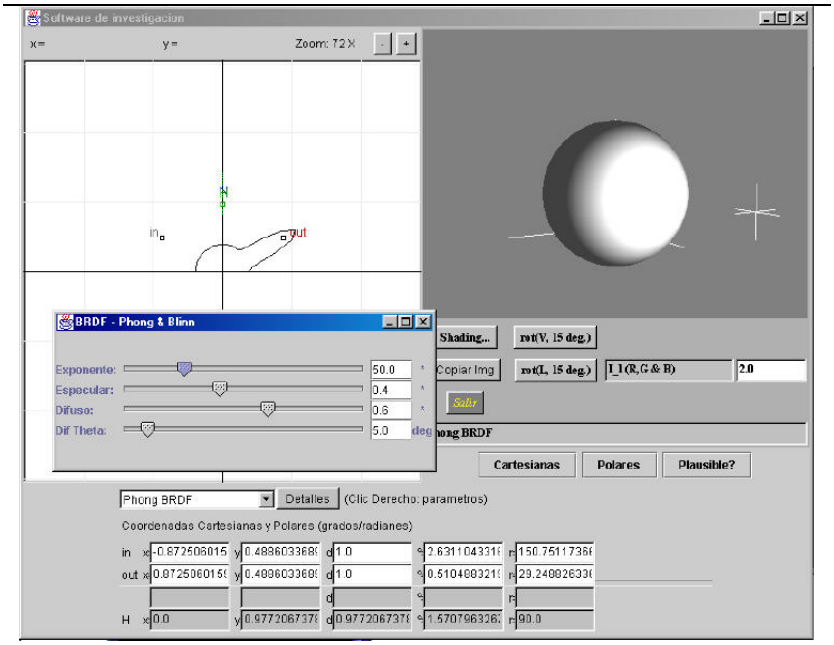
Cargador de Escenas

Editor de BRDFs


Tabla 2: Dos instantáneas de Wannabe Amazing

3.3 Atributos

El contenido del fichero GRF consiste fundamentalmente en una descripción geométrica de los elementos de una escena, pero cabe preguntarse qué información adicional nos interesa añadir para describir la iluminación de esta. En principio es fundamental la descripción de las fuentes extendidas de luz como se comentó en la sección 2.1, pero además en la descripción de la misma malla podrán aparecer para cada vértice, la reflectancia, el color, así como la posición, normal y coordenada de textura de este. Finalmente, la información de la irradiancia será añadida a los ficheros GRF por GIRT, lo que distingue a GRF del resto de formatos estudiados.

Cuando hablamos de irradiancia nos referimos a la densidad flujo incidente en la superficie por unidad de área [2]. Es posible modelar la reflectancia sobre un punto $\mathbf{x}$ sobre la superficie, calculando la integral doble sobre el hemisferio de todas las direcciones incidentes, relativas a la medida del ángulo solido proyectado. La ecuación de radiancia [1] es la siguiente:

$$L_o(\mathbf{x}; \overrightarrow{w_o}) = \int_\Omega f_r(\mathbf{x}; \overrightarrow{w_o}; \overrightarrow{w_i}) L_i(\mathbf{x}; \overrightarrow{w_i}) d\sigma_x^\perp(\overrightarrow{w_i}).$$

Aquí $L_o(\mathbf{x};\overrightarrow{w_o})$ es la radiancia que deja la superficie desde $\mathbf{x}$ en dirección $\overrightarrow{w_o}$, y $L_i(\mathbf{x};\overrightarrow{w_i})$ es la radiancia procedente de la dirección $\overrightarrow{w_i}$ que llega hasta el punto x en la superficie. El termino f_r representa a la función de bidireccional de distribución de la reflectancia, denominada aquí *BRDF*.

Estas funciones son representaciones parametrizadas de la reflectancia, que intentan expresar la interacción de la luz con la superficie, relacionando la radiancia que llega con la que sale de un punto dado. Actualmente trabajamos en la parte de editor de BRDF (Tabla 2), habiéndose implementado los modelos de reflexión más conocidos: los empíricos de Phong, Blinn y Lewis, el modelo anisotrópico de Ward, y los modelos físicos de Torrance-Sparrow, Poulin-Fournier y He-Torrance [3].

4 Discusión

En resumidas cuentas, *Wannabe Amazing* consigue unificar todas las particularidades de cada formato en uno solo. La visualización tridimensional es llevada a cabo empleando la librería Java3D, y gracias a la incorporación de objetos tipo *Scene, Appearance, Material*, es posible aplicar propiedades a un grupo de elementos independientemente del formato de fichero de procedencia.

Tras esto, el sistema de síntesis de imágenes GIRT es capaz de leer los modelos GRF y aplicarlos dentro de un proceso que emplea dos pasadas. La primera para la aproximación de la emisión difusa de las superficies, y la segunda, para considerar las reflexiones especulares perfectas usando los resultados de la primera pasada. Se encuentra disponible en la web (http://lsi.ugr.es/~curena/soft/grf/) la implementación de un parser y un visor de GRF para Linux con el que hemos hecho las pruebas de integración entre Wannabe Amazing y GIRT.

4.1 Agradecimientos

Este trabajo ha sido soportado por el proyecto TIC2001-2392-C03-03 concedido por la Comisión Interministerial de Ciencia y la Tecnología.

Referencias

[1] M. Cohen, J. Wallance. *Radiosity and Realistic Image Synthesis* Ed. Academic Press, 1993.

[2] J. Foley, A. van Dam. *Computer Graphics - Principles and Practice.* Addison-Wesley, Massachusetts, 2ª Ed., 1990.

[3] A. Glassner, *Principles of Digital Image Synthesis.* Morgan Kaufmann, San Francisco, CA, 1995.

[4] Java3D Comunity Site. *http://www.j3d.org*

[5] M. Lastra, *Síntesis Eficiente de Imágenes Fotorrealistas por Simulación del Modelo de Partículas de la Luz*, Documento de trabajo LSI-2000-3. Granada, 2000.

[6] Repositorio gratuito de modelos. *http://www.3dcafe.com*

[7] Sun Microsystems. *http://java.sun.com/products/javamedia/3D*

[8] The programmerś File Format Collections. *http://www.wotsit.org*

[9] C. Ureña, J. Parets, J.C. Torres, V. del Sol. *An Object-Oriented aproach to Ray Tracing Image Synthesis implementations.* Computers & Graphics Vol 16, No. 4, pp 363-368, Ed. Pergamon Press, 1992.

[10] C. Ureña, J.C. Torres, J. Revelles, P. Cano, V. del Sol, M. Cabrera. *GIRT: Un sistema orientado a objetos para simulación precisa de la iluminación* VII Congreso Español de Informática Gráfica (CEIG 97). pp. 191-207. Barcelona, 1997.

[11] VRML200x Specification, ISO/IEC 14772. *http://www.openworlds.com/x3d/* y *http://www.web3d.org/vrml/xd3/*