The Universe Generator System

Max Celedón C. <hydefus@inf.utfsm.cl>,
César Hernández M. <cesar@labsc.inf.utfsm.cl>
http://ugs3d.sourceforge.net

23 de diciembre de 2002

Trabajo a presentar en Encuentro Linux 2002

1. Introducción

The Universe Generator System (UGS) nace como necesidad de disponer una infraestructura para el desarrollo de aplicaciones 3d en tiempo real de superior abstracción que las herramientas que OpenGL provee. Sin dejar de lado la orientación de *máquina de estados* de OpenGL, UGS provee de un conjunto de clases independientes entre sí que permiten al desarrollador contar con elementos básicos y avanzados para la construcción de una aplicación 3d (demo). La idea de UGS no es evitar la programación sino entregar ciertas cosas ya construidas y suceptibles de mejorar. UGS se distribuye bajo la licencia GPL.

2. Historia y Motivación

A fines de los 80's era popularmente como se recuerda la época de los microcomputadores (Atari, MSX, Amiga, etc). En este contexto el desarrollo de computación gráfica se hacía con el único recurso disponible, el assembler y el acceso directo a los dispositivos. Bajo este paradigma de desarrollo nació la demoscene[**?**] en Europa, y la industria del videojuego. Con el pasar de los años y la popularización del hardware 3d, una nueva era de APIs también comenzó a emerger con fuerza, entre ellas Direct3d y OpenGL. Estas APIs permitían definir una especificación básica para mantener una cierta independencia del hardware sobre el cual se trabajaba. De estos elementos nace la motivación de desarrollar un motor gráfico, que entregase ciertos herramientas de mayor abstracción que una API no provee, elementos gráficos como sombras, generador de terrenos, el concepto abstracto de cámaras, tiempo, etc.

3. Herramientas Utilizadas

Para el desarrollo de UGS se han elegido C++ como lenguaje de desarrollo y dos APIs ampliamente conocidas, abiertas y multiplataformas:

SDL (Simple Direct Layer) [?]: Es una librería que permite interactuar sobre la mayor parte del hardware de E/S, como teclado, mouse, sonido, video, eventos y manejo del tiempo

(timers). La librería base de SDL incluye la gestión de ventanas, entre lo cual se puede destacar el control del contexto gráfico a utilizar, bit per pixel, resolución y los eventos sobre las ventanas. Adicionalmente, existe un conjunto de sublibrerías que profundizan la funcionalidad de SDL, permitiendo la carga de imágenes en distintos formatos (jpeg, png, bmp, etc.), una librería sonido (mod, xm, mp3, ogg, etc.) , un manejador de teclado, mouse, etc.

OpenGL [**?**]: Es la más ampliamente librería usada para el desarrollo de aplicaciones 2d / 3d. Fue desarrollada en 1992 por Silicon Graphics y ha sido adoptada como un estándar en computación gráfica debido a la portabilidad, disponibilidad y desempeño. Fue elegida porque es una especificación abierta, multiplataforma y bastante extendida, además de soportada por los fabricantes de hardware.

C++ : Se escogió este lenguaje de programación debido a su desempeño, portabilidad y su orientación a objetos.

4. Filosofía de UGS

La filosofía de UGS se basa en un modelo jerárquico de objetos. Estos objetos son los que componen una escena determinada, ya sean el universo mismo, luces, modelos, cámaras, etc. El orden jerárquico intuitivo es que el universo es el objeto padre del resto de los elementos. De esta forma, el universo es quien administra y conoce las cámaras, administra los recursos de luces, conoce la ubicación y controla el display de cada objeto. Existen 3 tipos de categorías básicas de objetos: luces, cámaras y modelos. Cada uno de los cuales deriva de otras clases que describen distintas características, como jerarquía, posición espacial, orientación.

El universo padre se encarga del despliegue de cada objeto ya que conoce la jerarquía actual de objetos. El recorrido del árbol es una búsqueda en profundidad y tiene como objetivo realizar una actualización de la matriz de transformación asociada a cada elemento, realizar un test de oclusión y finalmente si corresponde renderizar los objetos. El universo realiza 3 veces este recorrido debido a que es necesario mantener un orden para realizar un adecuado despliegue. Hay que considerar por ejemplo, que los objetos transparentes deben ser desplegados al final de la escena, a diferencia de las luces que son renderizadas al inicio.

5. Elementos Disponibles

Se han desarrollado diferentes categorías de elementos que componen el motor gráfico:

- Clases para el manejo de matrices, vectores y quaterniones.
- Clases para el manejo del tiempo, el espacio y transformaciones.
- Un conjunto de elementos para la visualización mediante el uso de cámaras abstractas.
- Modelos y derivados.
  - Generador de terrenos.
  - Sistemas de partículas (lluvia, fuego, etc.).
  - Generador de hierba.
  - Arboles procedurales fractales.

- Modelos 3d estáticos (compilados) y animados.
- Modelos con sombras volumétricas.
- Agua dinámica y reflectiva.

- Efectos especiales:
  - Motion blur (universo y modelos).
  - Depth of field (cámara).
  - Transiciones entre cámaras.
  - Lensflare y sol.
- Otras características: Pixel buffers, test de oclusión, detección de colisiones.

A continuación vamos a resumir los principales tópicos implementados en UGS.

5.1. Generador de terrenos

Para escenas exteriores es necesario contar con un generador de terrenos que permita tomar algún modelo de entrada y construir un "escenario de mundo". Para ello, en UGS se tienen 3 tipos de aproximaciones. Las mas sencillas (las primeras heurísticas) son las siguientes:

Terreno plano: Se toma un bitmap de entrada y se construye un modelo gigante a través de una display list.

Terreno esférico: Es idéntico al anterior pero consiste en una esfera gigante, dando la sensación de circularidad y de ser continuo.

Generador de terreno con algoritmo LOD[?]: Se toma un bitmap de entrada y se construye un arreglo de vértices el cual es procesado a nivel de hardware de video. El algoritmo usado en UGS es el conocido como adaptative quadtree [**?**], y define elementos independientes que representan sectores del terreno (quadsectors).

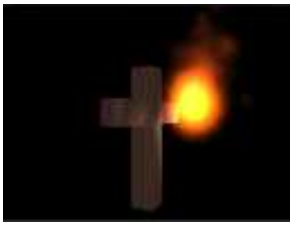
Características de un quadsector:

- Cada quadsector conoce su propio nivel de detalle, sabe dibujarse de acuerdo a ese detalle y conoce si está dentro del campo de visión de la cámara activa para ser dibujado. De esta forma se procede a realizar la tesselación correspondiente, los sectores que están más cerca de la cámara son dibujados con el máximo nivel de detalle, mientras que los más lejanos con menor nivel, a este tipo de algoritmo se le denomina algoritmo LOD (level of detail).
  - Cada quadsector conoce su propia topología y dibuja una secuencia de agua si el terreno en algún punto del sector es más bajo que un nivel dado.
  - Usando color arrays se procede a definir las transiciones de texturas usando alpha blending, por ejemplo, las zonas donde la derivada sea más grande presentaran texturas de tipo rocosa, las zonas planas de cierta altura hacia arriba, presentarán texturas de nieve, etc. Con esta misma técnica se pueden dibujar otro tipo de texturas o detalles del piso del terreno, como por ejemplo caminos, rocas, etc.

Generador de terrenos.

5.2. Sistema de Partículas

Se tiene un pequeño motor de partículas que actúa como contenedor de un grupo de ellas definiendo un comportamiento sistémico a través de fuerzas externas que actúan. A su vez cada partícula tiene un comportamiento propio. Como se puede ver en [**?**],este tipo de sistemas se realizan efectos físicos como fuego, lluvia, nieve, insectos, etc.



Fuego usando generador de partículas y luz atachada.

5.3. Generador de Hierba

Es una implementación que permite crear diferentes tipos de hierbas (pastos). La idea fundamental se basa en el cálculo de los ángulos de torsión que va adoptando cada filamento de pasto a lo largo de toda su extensión. Además se toma en cuenta el proceso de crecimiento que sufre a medida que transcurre el tiempo. Para obtener los ángulos de torsión el modelo utiliza un diagrama de fuerza de cuerpo libre, en el que se consideran las fuerza dadas por el peso y la rigidez. Es un modelo que toma elementos discreto de masa, sobre los cuales se va aplicando jerárquicamente el modelo de fuerza para así obtener la solución en ese punto.

Existen una serie de parámetros que definen el comportamiento de un tipo especial de hierba:

- Función de distribución de la masa.
- Densidad.
- Función de torsión.
- Función de longitud de crecimiento.

Generador de hierba.

- Función de engrosamiento en base al crecimiento.

Para la representación gráfica, la implementación toma como entrada un modelo 3D que utiliza como patrón para la generación de la hierba. Éste consiste de un filamento recto de pasto que contiene un conjunto de joints. Luego aplicando la información que se obtiene por cada elemento de pasto, sobre una copia del modelo 3D, se va construyendo la planta de hierba.

5.4. Modelos 3D y Animaciones

Para este propósito se ha diseñado una clase que permite el manejo de modelos y animaciones. El proceso implica la lectura de modelos previamente diseñados por alguna aplicación de modelamiento 3D (en formato externo), los que se convierten al formato propio de UGS. Por el momento sólo es posible cargar modelos en formato Milkshape 3D [**?**].

Las animaciones se basan en una técnica conocida como animación esquelética [**?**]. Para animar un modelo se construye un esqueleto compuesto de un conjunto jerárquico de joints, de modo que a cada joint se le asocian los vértices cercanos a éste. Además, cada joint contiene información de traslación y rotación (relativa y absoluta), así como un conjunto de *keyframes*. Los keyframes son las posturas fundamentales del esqueleto con las que se define la secuencia de la animación. Para generar entre dos keyframes las sucesivas posturas del modelo se utiliza interpolación a través de quaterniones.

5.5. Sombras

Esta es una de las técnicas más ambiciadas por los motores actuales, ya que permiten construir escenas de gran realismo, el campo es extremadamente grande y algunas técnicas muy complejas. En UGS existen hasta el momento 3 tipos de sombras, y otra en desarrollo.

Planar shadows: Se tiene una matriz proyectora (llamada matriz idempotente), que no tiene inversa, y que permite llevar todos los vértices un modelo hacia un plano del espacio. En este caso se dibuja el objeto, luego se recalculan la transformación de los vértices del modelo usando este proyector, y se vuelve a dibujar el objeto pero con blending o un color adhoc de sombra.

- Ventajas: Muy rápido y fácil de implementar.
- Desventajas: Disponible sólo para superficies planas.

Volumetric stenciled shadows: Esta técnica es más avanzada, y consiste en hacer un cast y recorrer todas las aristas de un modelo, para determinar las caras visibles e invisibles

desde la luz. Una vez determinadas esas aristas que constituyen lineas de adyacencia entre estos dos tipos de polígonos, se procede a dibujar una proyección volumétrica hacia un infinito determinado. Utilizando el stencil buffer disponible en la mayoría de las tarjetas de video nuevas, se procede a determinar que pixels están sombreados y cuales no.

- Ventajas: Rápido para modelos sencillos, bastante real y se proyecta sobre cualquier cosa.
- Desventajas: Al usar stencil buffer utilizando las propiedades de culling del volumen proyectado, es necesario cambiar la orientación cuando un observador entra a un volumen de sombra, esto aumenta la complejidad. Es muy dependiente de la geometría de los objetos.

Shadow maps: Consiste en tener texturas precalculadas y mapeadas al momento de renderizar la escena. En UGS, el generador de terrenos utiliza un shadow map para crear la sombra entre cerros y montes.

- Ventajas: Excelente realismo para escenas con lighting estático, se puede generar fácilmente penumbra con cero consumo de CPU / GPU.
- Desventajas: Inimaginable para lighting dinámico.

Shadow mapping (en desarrollo actualmente): Aprovechando las capacidades de T&L del nuevo hardware disponible, se procede a renderizar la escena desde el punto de vista desde la luz. Con algún efecto de luminancia, fog o leyendo el depth buffer se construye una textura que contiene las distancias rasterizadas desde la luz hasta el pixel. Luego, se dibuja la escena normalmente y se habilita la función de proyección de texturas, con lo cual se puede comparar si el pixel está o no sombreado.

- Ventajas: Independiente de la geometría, el nuevo hardware trae soporte para esta técnica (SGIX_SHADOW) por lo que es muy rápido en hardware actual. Se proyecta en cualquier parte al igual que la sombra volumétrica.
- Desventajas: Se requiere una buena resolución de textura para que se vea bien. Algoritmo complejo de programar.

Sombras volumétricas.

5.6. Agua

Varias características definen la forma y comportamiento del agua generada en UGS.

Luz especular y normales: Se trata de emular el comportamiento especular de la luz cuando incide en una superficie de agua. Para eso es necesario que la superficie presente una gran cantidad de normales (si fuera posible por pixel) para resaltar el efecto.

Movimiento y propagación: Para el movimiento se usó una ecuación de recurrencia tomada de los autómatas celulares, que distribuye y propaga las ondas.

$$W_{t+1}(x,y) = C * e^{-t^2} * (W_t(x+1,y) + W_t(x-1,y) + W_t(x,y+1) + W_t(x,y-1))$$

Reflexión: Este efecto se realiza utilizando un pixel buffer y la pasada recursiva del display del universo. De esta forma se genera una textura desde el punto de vista de la superficie y se mapea usando generación automática de texturas.

Reflexión del ambiente sobre el agua.

5.7. Motion Blur

Consiste en repetir distintos frames de un objeto con la idea general de que queda una estela cuando este se mueve o produce algún tipo de irradiación. Hay actualmente 3 tipos de motion blur disponibles en UGS, brevemente:

Motion blur de un modelo: Se trata de que un modelo deja una estela cuando cambia de posición espacial, rota o está animado en un momento dado.

Radial blur de un modelo: Se dibujan con transparencias copias del mismo objeto pero escaladas dado la sensación de que el modelo produce irradiación, o tiene un aura.

Universe motion blur: La misma idea del modelo, pero llevada al universo completo. Esta técnica tiene la ventana de que es independiente de la geometría.

Motion Blur.

5.8. Depth of Field

El efecto de Depth of Field (DOF) es una parte integral de la visión humana, en la que el lente del ojo humano se acomoda a las distintas distancias de observación. Cuando un objeto está fuera de ese foco de observación se ve borroso. El grado de desenfoque depende de la potencia del lente, el diámetro y la distancia del objeto.

Se han desarrollado muchos algoritmos para crear el efecto DOF, sin embargo la mayoría de los algoritmos son demasiado costosos para utilizarlos en aplicaciones en tiempo real. En general estos algoritmos se clasifican en dos grupos: métodos de *filtrado de post–proceso* y de *rendering de múltiples pasadas* [**?**].

En nuestro motor se ha implementado un básico pero muy rápido algoritmo de DOF del primer tipo [**?**], que utiliza una técnica de mip mapping, en la que se aprovecha la aceleración disponible en la mayor parte del hardware gráfico para el manejo de texturas. El algoritmo consiste de dos etapas. En la primera etapa se generan un cierto número de texturas correspondientes a distintos niveles de mipmapping de la imagen actual. Luego, en la segunda etapa se desactiva la escritura del buffer de profundidad, y se procede generando primero el desenfoque para el volumen de visión que está más lejano al plano focal, y posteriormente el que más cercano. El efecto DOF se logra colocando las texturas de mipmapping en distintos planos de profundidad desde el foco hacia el plano lejano, y desde el foco al plano cercano.

Depth of Field.

5.9. Lensflare

Se le llama al efecto que produce la luz al entrar en el diafragma del lente de la cámara. En

UGS el lensflare se puede atachar a una luz del sistema para aumentar la sensación de luminosidad.

La complicación permanente de los algoritmos de lensflare existentes son dos:

- Realizar un test de oclusión constante que permita conocer si la luz es visible o no por la cámara, ya que algún otro objeto podría estar obstruyendo el paso.
- Cambiar la intensidad de la luz y lensflare dependiendo del ángulo y distancia desde el observador a la fuente de luz.

Actualmente está disponible la primera función, utilizando la extensión opengl introducida por HP (GL_HP_OCCLUSION_TEST) la cual realiza una comparación a nivel de depth buffer. En desarrollo se encuentra la segunda función, la heurística a utiliza será constatar el ángulo de visión con el ángulo de incidencia de la luz.

Lensflare sobre el sol.

5.10. Pixel buffers

Se definen como contextos alternativos OpenGL. Con un contexto paralelo al frame/back buffer se puede utilizar como un buffer temporal especial para la generación de texturas en tiempo real. Las tarjetas actuales implementan y soportan por hardware esta características, en windows existe WGL_PBUFFER_ARB, y el linux vía GLX (1.2 o superior) glx_pbuffer.

6. Desarrollo futuro

Existen diversos tópicos que están en los planes para ser incluidos en el motor gráfico. Entre estos elementos podemos destacar:

- Un sistema de scripts y eventos para la planificación de escenas.
- Hardware shadow mapping y proyección de texturas.
- Vertex programs y texture shaders.
- Radiosity y BRFD.
- Morphing
- Sistema de LOD e impostors para modelos.
- Extensiones y mejoras de algunas clases existentes.

7 Desarrolladores actuales

Los autores originales del proyecto comenzaron el desarrollo de UGS en marzo del 2002.

Nombre	Aporte a UGS
Franco Catrín	Convenciones de nombres, manejador de eventos, sistema de partículas, planificador.
Max Celedón	Generador de terrenos, árboles fractales, fuego, agua, lensflare y cielo, sombras, efectos de cámara, pixel buffer, motion blur, display del universo, test de oclusión.
César Hernández	Código base, Generador de hierba, soporte vectorial, DOF, modelos animados, Soporte espacial.