Gráficos por computador

Práctica 3: CarGL

Manual del desarrollador

Rubén Martínez Vilar
48333441E
2013-2014

Contenidos

1. Introducción
2. Características implementadas
    - Estructura de clases
    - Lista de características
3. Partes opcionales
    - Más tipos de objetos
    - Texturas avanzadas. Mipmapping

1. Introducción

CarGL es una aplicación basada en la librería gráfica OpenGL y en la cual nos movemos por una ciudad virtual mediante unos vehículos seleccionables. Se proporcionan paneles desde los que se puede modificar ciertas características propias de cualquier entorno 3D.

En este documento vamos a ver que características tiene la aplicación y cómo se han implementado desde un punto de vista más técnico.

En la mayoría de apartados se muestra una cita al enunciado de la práctica original en cursiva y con fondo amarillo.

2. Características implementadas

Estructura de clases

Antes de ver la lista de características implementadas vamos a aclarar la estructura de clases y que hace cada una para poder encontrar todo fácilmente.

- **main.cpp**: mantiene básicamente la estructura original, recoge las teclas y hace las modificaciones necesarias en la clase Scene. De la misma manera con el ratón y otros eventos de la ventana. En el propio main() se agregan las luces y cámaras por defecto, se crean las instancias de Scene y GuiManager y se llama a su inicialización.
- **Global.h**: contiene una serie de estructuras de datos que se usan durante todo el programa, como puede ser el tipo Vector3, Luces, Cámaras, Materiales, etc.
- **Scene**: Aquí se realiza toda la inicialización de los estados y propiedades de OpenGL y se actualizan y renderizan los objetos de la escena. También guarda una serie de flags para activar o desactivar características, así como el objeto seleccionado y las listas de objetos y texturas.
- **GuiManager**: básicamente contiene todo lo del GLUI. Lo he separado para más claridad.
- **Object:** contiene las propiedades de cada objeto de la escena, sus display lists, sus materiales, posicion, rotación, etc. Pueden tener un objeto parent, en este caso su posicion y rotación se realiza con respecto a este, pero también se aplica una rotación local al mismo objeto. Por ejemplo las ruedas rotan y se posicionan con respecto al coche, pero también rotan sobre sí mismas.
- **Material:** encapsula las propiedades de los materiales de un objeto o de una forma parte de un objeto. Pueden tener transparencia y pueden enlazar con una textura que se guarda en Scene y que comparten entre materiales.
- **Texture**: carga desde una imagen una textura y guarda su id de opengl (dos en este caso debido a que mostramos dos técnicas de texturas), luego se encarga de hacer el bindeo de la textura cuando se le pide desde el material.
- **Clock**: se trata de un reloj funcionando sobre métodos de c++ y que uso para algunas rotaciones automáticas, de manera que se realizan más suavemente y en tiempo fijo.
- **SOIL.h**: esta es una librería para cargar imágenes en c++ un poco más completa que la que venia por defecto con la práctica, ya que permite cargar imagenes en otros formatos como .png. También permite generar la textura directamente en OpenGL pero para el aprendizaje en esta práctica simplemente utilizo la función para cargar los píxeles de la imagen y luego genero la textura manualmente.
- **tiny_obj_loader.h**: como explico más adelante he sustituido el cargador de 3ds por uno de obj/mtl ya que me permite más flexibilidad a la hora de cargar objetos divididos en varias partes con sus materiales y texturas.

Lista de características

Estos son los puntos propuestos en la práctica y que han sido implementados:

Carga de objetos en Display Lists

Debe leer una serie de objetos geométricos, (formato .3ds) modelados con Blender, almacenados en uno o varios archivos y tratar cada objeto con una lista de visualización (Display List) de OpenGL.

En este primer punto he realizado unos cambios de formato, y **en vez de utilizar el .3ds**, exporto los objetos a **.obj/.mtl** de manera que guardan las propiedades de los materiales y las normales de texturas y puedo cargar estos automáticamente o hacer modificaciones en el modelo en Blender y volver a exportar sin tener que tocar nada del código de CarGL. Además puedo mantener los objetos separado en varios modelos/formas dentro del .blend y asignar a cada uno un material y textura por separado.

Aunque existen cargadores de 3ds más avanzados he preferido el obj ya que había usado ese mismo cargador en otro proyecto y los ficheros obj/mtl son más fáciles de modificar desde cualquier editor de texto, por ejemplo para cambiar las rutas de algunas texturas.

Dentro de CarGL la clase Object lee el obj y genera las display list para este, que pueden ser varias para los objetos que se componen de varias partes, cada una con su material/textura.

Los objetos se almacenan en un vector de la clase Scene, cuando se crea un objeto, éste pregunta al Scene si existe ya un objeto igual y en caso afirmativo Scene devuelve el id de la primera Display List, como el Object sabe el la cantidad de formas que lo componen, las listas consecutivas no es necesario guardarlas, en el dibujado se realiza un bucle pasando por todas las listas consecutivas.

De esta manera un objeto puede repetirse muchas veces pero siempre usan las mismas listas y al mismo tiempo mantienen los materiales por separado, que sí son únicos por objeto (pero las texturas si se comparten, como veremos). Esto sirve, por ejemplo, en el caso de los coches que pueden cambiar el color de su carrocería.

Vehículos y demás elementos

El vehículo de la práctica habrá que sustituirlo por el propio. Añadir el resto de elementos diseñados con Blender. Las ruedas del coche girarán al avanzar y se doblarán a la izquierda y derecha al girar. El color de la carrocería del coche podrá cambiarse.

El vehículo original ha sido sustituido por un cochecito como los que se usan en los campos de golf y se han agregado otros elementos: bancos, farolas, dos tipos de edificios, aceras, carretera, papeleras, rotonda con bola giratoria, señales, etc. Se pueden mostrar u ocultar desde el panel derecho.

Las ruedas de los coches giran y doblan a izquierda y derecha. En este caso he separado en dos tipos de ruedas, de manera que las trasera no doblan a izquierda y derecha, solo las delanteras, pero todas giran en el eje

perpendicular a estas. He colocado el eje de los coches por su zona trasera de manera que giren sobre esta y no sobre su centro.

Los objetos de ruedas, al igual que el icono de selección encima de los coches, tiene como parent el mismo coche, con lo que sus posiciones y rotaciones son relativas a este, además de tener sus propias rotaciones locales al mismo centro del objeto.

En el panel de la derecha podemos encontrar las sección “Coches” donde se van agregando automaticamente entradas al agregar coches a la escena y desde el cual podemos cambiar el color de su carrocería. El hecho de cargar en obj y poder mantener las formas geométricas de los objetos por separado me permite poder modificar únicamente la carrocería, dejando el resto del coche con el color por defecto.

Selección

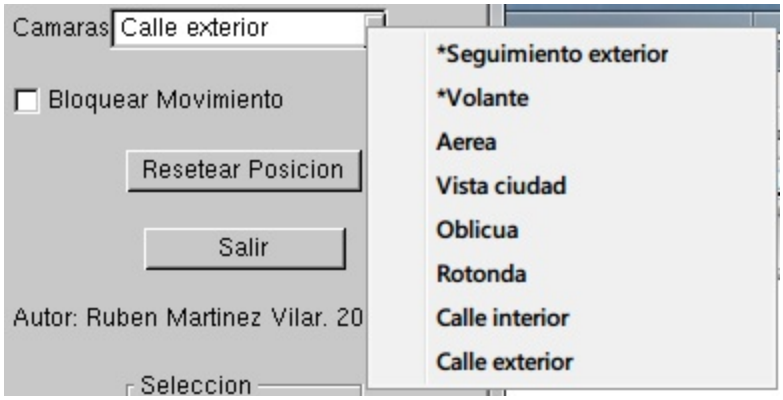
Habrá que emplear marcadores para seleccionar los coches, el marcador podrá ser desde un círculo en el suelo que rodea al coche hasta el efecto que se desee implementar.

Esto se ha implementado mediante un objeto geométrico en forma de pirámide que se puede ver encima del coche seleccionado. Este objeto tiene como parent al objeto seleccionado, cambia al cambiar de selección. También tiene un rotación para darle más animación.

Cámaras

Se añadirán al menos dos cámaras estáticas (una de ellas con cierta vista aérea), además de otra que siga al vehículo seleccionado que estamos moviendo.

En el panel de la derecha tenemos un combobox desde el cual podemos escoger entre varias cámaras. Para este entregable encontramos que las dos primeras cámaras son de **seguimiento** y el resto son **libres**.



Las cámaras estan definidas en **Global.h** tienen varias propiedades como su nombre (para el panel derecho), la posición, rotación o punto al que mirar, si están activas y si son de seguimiento, que se colocan en la posición indicada al crearlas con respecto al objeto seleccionado en ese momento.

Las cámaras de seguimiento no pueden ser rotadas ni modificadas mediante el interfaz del panel inferior, pero el resto de cámaras si. Esto se ha hecho así para evitar algunos problemas de GLUI al cambiar entre distintos controles.

Podemos resetear la posicion/rotación de la **cámara actual** seleccionada con el botón “Resetear Posicion”.

Utilizar el ratón para situar la cámara, es decir, rotación global de la escena, zoom y desplazamiento. Para ello se empleará la siguiente combinación de teclas:

- *Rotación: el movimiento del ratón y botón izquierdo (+CTRL para el eje z).*
- *Traslación: (SHIFT) + el movimiento del ratón y botón izquierdo (+CTRL para el eje z)*
- *Escalado: el movimiento del ratón y botón derecho (~~+CTRL para el eje z~~).*

Esto también se ha implementado tal y como se muestra pero con **un cambio**, el **escalado** se hace uniformemente sobre los ejes x,y,z, de esta manera escalamos sin deformar y se consigue una mejor navegación por la escena.

Luces

Luz ambiente activa o inactiva
Esto puede ser modificado simplemente desde el apartado “Propiedades” del panel derecho.

Al menos tres focos de luz predefinidos posibles (se podrán cambiar todos su parámetros)

Para este entregable se han agregado 5 luces por defecto, aunque por código se pueden crear más luces fácilmente y se agregarán automáticamente al panel, al igual que con las cámaras.

La estructura de las luces la podemos encontrar en Global.h y tienen propiedades que pueden ser modificadas desde el panel derecho, como el color, intensidad y si estan activas o no.

Menús

Desde el primer apartado del panel derecho, en “Propiedades” podemos activar o desactivar algunas características de la escena. Junto a cada propiedad se incluye la tecla de acceso directo para modificarla mientras el foco lo tiene la escena.

- **Dibujar reflejos (R)**: Hace que la carretera refleje los objetos de la escena. Al tratarse de una superficie muy grande el rendimiento se ve afectado en gran medida por lo que se mantiene desactivada por defecto.
- **Luz ambiente (L)**: Iluminación de la escena.
- **Modo alámbrico (W):** Propiedad de relleno de polígonos.
- **Z-Buffer (Z):** Buffer de profundidad.
- **Culling (C):** No dibuja las caras con normal contraria a la cámara. En objetos transparentes siempre se desactiva como se explica en el apartado de blending.
- **Texturas (T)**: Activa o desactiva las texturas.
- **Mipmapping (M)**: Activa o desactiva las texturas generadas con mipmapping.
- **Modelo de sombreado:** Por **defecto** en **smooth**, se suaviza el sombreado entre caras. En flat podemos ver claramente la diferencia de sombreado entre las caras de los polígonos, se aconseja probarlo con texturas desactivadas.
- **Proyección:** Vista en proyección **Perspectiva** por **defecto**.
- **Sentido caras exteriores:** Configura como se recorren las caras para dibujar los polígonos de los objetos, por **defecto** en **Antihorario**.

Materiales

Al menos tres materiales predefinidos posibles

Como ya se ha explicado antes, los materiales se recogen del fichero .mtl de cada objeto y un objeto puede estar compuesto por varios materiales que se pueden encontrar en este mismo fichero.

Para realizar cambios sobre el material se puede hacer fácilmente sobre el modelo en Blender, exportarlo de nuevo a obj/mtl o simplemente copiar los cambios del material en cuestión y editarlo sobre el fichero mtl directamente, otro punto donde este formato nos proporciona más flexibilidad que el 3ds.

La clase que encapsula las propiedades de los materiales es la clase **Material**, el formato del material es de tipo **material_t**, que podemos encontrar en **Global.h**. Los materiales tienen propiedades de ambiente, difuso, especular, transmitancia, emisión, brillo (shininess) e indice de refracción. Además de enlace a texturas difusas, ambiente, especular y normal. En esta práctica solo hago uso del campo diffusse para la carga de texturas.

Por otro lado también se pueden definir materiales semitransparentes, pero esto se explica en el apartado de blending.

Texturas

Se implementarán texturas a un nivel básico

La clase Texture encapsula las texturas implementadas mediante dos técnicas, la normal y mediante mipmapping (ver 3. Partes Opcionales).

Las texturas se guardan en Scene de manera que solo se crean una vez y se comparten entre los distintos materiales de los objetos. Por defecto las texturas estan en modo REPEAT y para superficies como la carretera o la acera se han establecido desde Blender el tamaño de las texturas sobre el mapa uv para que se repitan más o menos.

En este caso también es de utilidad el formato obj/mtl para guardar las coordenadas de textura.

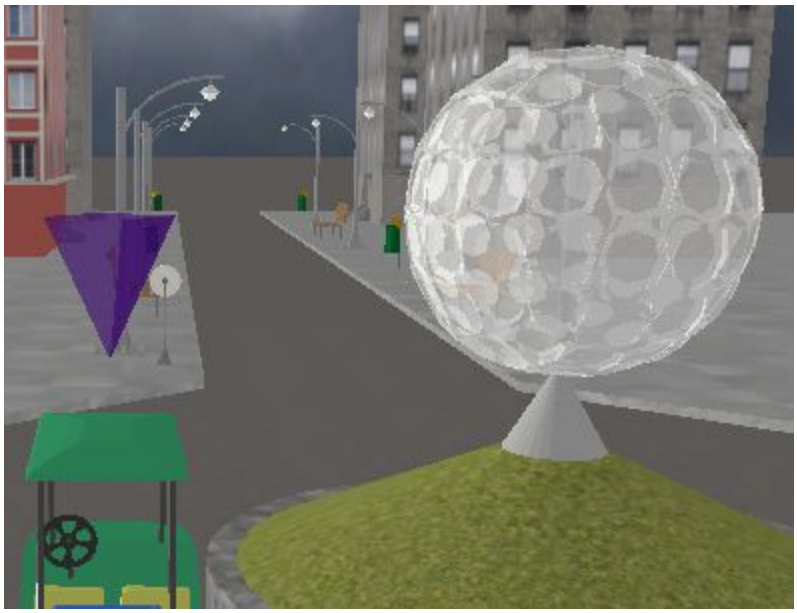
Objetos semitransparente (blending)

Se implementarán objetos semitransparentes como cristal (blending)

En la clase Material se ha agregado una propiedad que indica si este es semitransparente o no. En caso de ser transparente se activa el blending y se establece un alpha por defecto de 0.6. También desactivamos el culling, ya que en los objetos semitransparente se deberían ver las caras interiores.

Los objetos con transparencia se crean los últimos para que se dibuje todo lo que se ve a través de ellos.

```
glDisable(GL_CULL_FACE); // Queremos que se vean las caras interiores
glEnable (GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
color[3] = 0.6; // Alpha para transparencia
```



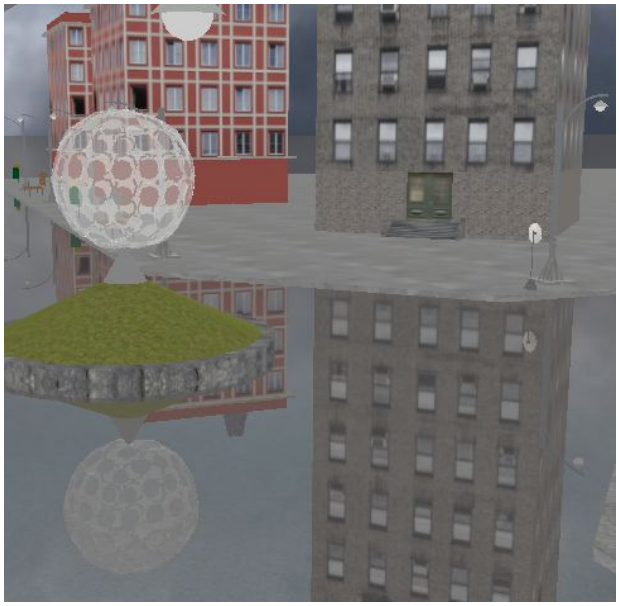
Podemos ver este efecto tanto en la bola de la rotonda como en el icono de selección encima del coche.

Reflejos

Se implementarán reflejos. Los coches (y el resto de los objetos se deberán sobre el asfalto, sólo sobre él, y no sobre el resto de objetos, aceras, parques, etc.)

En la función **renderReflection** de **Scene** se puede encontrar el método para renderizar la escena invertida y que se muestre solamente sobre la carretera.

Se dibuja la carretera desactivando el test de profundidad y activando el stencil test para marcar la carretera, a continuación se dibuja el resto de objetos solo donde el stencil tenga valor 1. Se desactiva el stencil test y volvemos a dibujar la carretera de modo normal ya desde la función render(). El resultado es el siguiente.



3. Partes opcionales

Más tipos de objetos

Se podrá añadir más tipos de objetos (coches distintos con otro movimiento de ruedas, etc.)

En este punto podría nombrar el volante de los coches, que gira sobre sí mismo cuando se pulsa la tecla de girar a izquierda o derecha y vuelve a su posición inicial cuando se suelta la tecla. Podemos verlo mejor desde la cámara interior del coche “Cámara volante”.

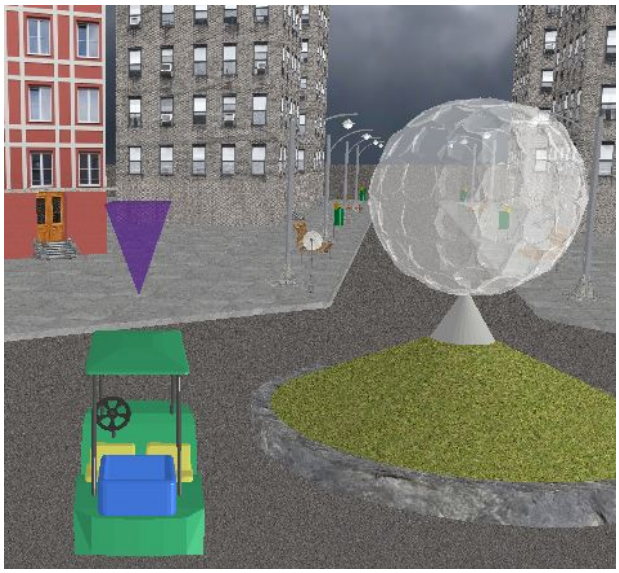
También se ha agregado una pelota giratoria a la rotonda del centro del escenario.

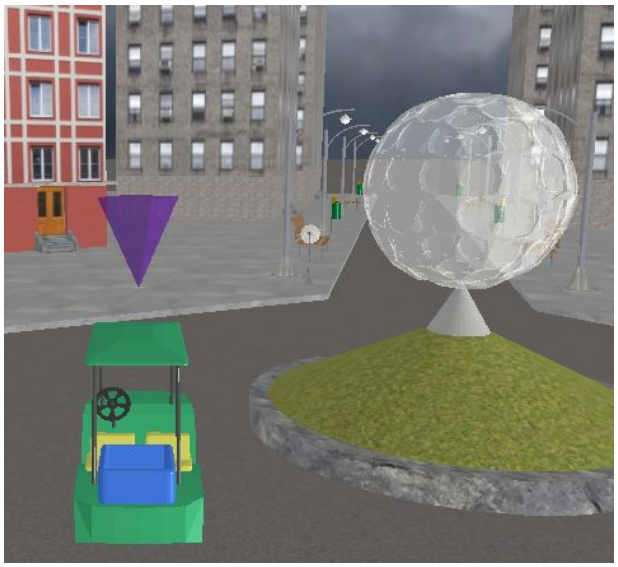
Texturas avanzadas. Mipmapping

Implementación avanzada de las texturas

Además del método normal de cargar texturas mediante **glTexImage2D**, también se generan texturas en modo mipmapping mediante **gluBuild2DMipmaps**. Esta no es la mejor manera de utilizar mipmapping y ya no se usa nunca, pero como la mayoría de funciones utilizadas en esta práctica ya son antiguas he dejado este método, que además no requiere de extensiones, cosa que el resto de los indicados en la wiki oficial de OpenGL si (http://www.opengl.org/wiki/Common_Mistakes#Automatic_mipmap_generation).

Aún así la mejora es sustancial con respecto al método normal, podemos ver la diferencia descarmando la casilla mipmapping del panel de “Propiedades” o pulsando la tecla “M”. En superficies con texturas que se repiten y con mucho detalle como la carretera, aceras y edificios se nota mucho el suavizado del ruido.





Skybox

También podemos ver que he agregado un Skybox a la escena para cubrir el fondo del escenario. Se dibujan 6 quads y se les pega a cada uno su textura correspondiente, invirtiendo las normales de textura para que miren hacia adentro. Otro método sería desactivando el culling para poder ver las caras exteriores.

Usando extensiones se podría aplicar texturas de tipo CUBEMAP con modo SEAMLESS con lo que no veríamos los bordes del cubo, las líneas que unen cada quad.