Práctica 4

Materiales, fuentes de luz y texturas

4.1. Objetivos

Con esta práctica el alumno aprenderá a:

- Incorporar a los modelos de escena información de aspecto, incluyendo las nomales de las mallas y modelos sencillos de fuentes de luz, materiales y texturas.
- Visualizar, usando la funcionalidad fija de OpenGL, escenas incluyendo varios objetos con distintos modelos de aspecto.
- Permitir cambiar de forma interactiva los parámetros de los modelos de aspecto de la escena anterior.

4.2. Desarrollo

En esta práctica incorporaremos a las mallas de triángulos información de las normales y las coordenadas de textura, así como crearemos estructuras de datos para representar modelos de fuentes de luz, materiales y las texturas, posibilitando la visualización de objetos con distintos modelos de aspecto. El código descrito aquí debe añadirse al código existente para las prácticas desde la 1 hasta la 3, sin eliminar nada de la funcionalidad ya implementada.

4.2.1. Cálculo y almacenamiento de normales.

A partir de la tabla de coordenadas de vértices y la tabla de caras de una malla, se deben calcular las dos tablas de normales (normales de caras y normales de vértices). Cada tabla de normales es un array que en cada entrada contiene una tupla de 3 reales que representa (en coordenadas maestras o de objeto) el vector perpendicular a la cara o a la superficie de la malla en el vértice. Las dos tablas de normales quedarán almacenadas en la estructura o instancia de clase que representa la malla, junto con el resto de tablas (coordenadas de vértices, de textura, etc...).

Para el cálculo de las tablas de normales se puede seguir este procedimiento:

1. En primer lugar se calcula tabla de normales de las caras. Para ello se debe de recorrer la tabla caras que hay en la malla. En cada cara se consideran las posiciones de sus tres vértices, sean estas, por ejemplo $\vec{p}$,$\vec{q}$ y $\vec{r}$. A partir de estas coordenadas se calculan los vectores $\vec{a}$ y $\vec{b}$ correspondientes a dos aristas, haciendo $\vec{a} = \vec{q} - \vec{p}$ y $\vec{b} = \vec{r} - \vec{p}$. El vector $\vec{m}_c$, perpendicular a la cara, se obtiene como el producto vectorial de las aristas, es decir, hacemos: $\vec{m}_c = \vec{a} \times \vec{b}$. Finalmente, el vector normal $\vec{n}_c$ (de longitud unidad) se obtiene normalizando $\vec{m}_c$, esto es: $\vec{n}_c = \vec{m}_c / \|\vec{m}_c\|$.

   Hay que tener en cuenta que, para objetos cerrados, es necesario que todas las normales apunten hacia el exterior del objeto, y que en los objetos abiertos, todas ellas apunten hacia el mismo lado de la superficie. Para ello la selección de los tres vértices $\vec{p}$,$\vec{q}$ y $\vec{r}$ de la cara debe hacerse de forma coherente, es decir, siempre en orden de las agujas del reloj, o siempre en el contrario (visto desde un lado de la superficie). Aunque se haga de forma coherente, las normales pueden quedar todas ellas apuntando al lado equivocado, si este fuese el caso, se puede cambiar el orden del producto vectorial, es decir,hacer $\vec{m}_v = \vec{b} \times \vec{a}$ en lugar del originalmente descrito, ya que el producto vectorial es anticonmutativo.

2. Una vez calculadas las normales de caras, se obtienen las normales de los vértices. Para cada vértice, el vector $\vec{m}_v$, es un vector aproximadamente perpendicular a la superficie de la malla en la posición del vértice. Se puede definir como la suma de los vectores normales de todas las caras adyacentes a dicho vértice, es decir:

$$\vec{m}_c = \sum_{i=0}^{k-1} \vec{u}_i$$

   donde $\vec{u}_i$ es el vector perpendicular a la i-ésima cara adyacente al vértice (suponemos que hay k de ellas). Al igual que con las caras, el vector normal al vértice $\vec{n}_v$ se define como una versión normalizada de $\vec{m}_v$, es decir: $\vec{n}_v = \vec{m}_v / \|\vec{m}_v\|$.

   Una implementación básica (derivada directamente de esta definición de $\vec{n}_v$) requeriría recorrer la lista de vértices (en un bucle externo), y en cada uno de ellos buscar sus caras adyacentes, recorriendo para ello la lista de caras completa (en un bucle interno). Esta implementación, por tanto, tiene complejidad en tiempo en el orden del producto del número de caras y de vértices (o cuadrática con el número de vértices, que es proporcional al de caras para la inmensa mayoría de las mallas). Se recomienda diseñar e implementar un método más eficiente con complejidad en tiempo en el orden del número de caras, método basado en recorrer las caras en el bucle externo, en lugar de los vértices, ya que obtener los vértices de una cara se puede hacer de forma inmediata (en $O(1)$) sin más que consultar la entrada correspondiente de la tabla de caras.

4.2.2. Almacenamiento y visualización de coordenadas de textura

El siguiente paso será aumentar las estructuras de datos que representan las mallas en memoria y extender el código de visualización para calcular y visualizar las coordenas de textura.

Para ello, se añadirá a las mallas la tabla de coordenadas de textura, que será un array con n_v pares de valores valores de tipo `GLfloat`, donde n_v es el número de vértices. Esta tablas se usarán para asociar coordenadas de textura a cada vértice en algunas mallas. En las mallas que no tengan o no necesiten coordenadas de textura, dicha tabla estará vacía (0 elementos) o será un puntero nulo.

Se crearán el código para dos nuevos modos de visualización (*modos con iluminación*), adicionales a los que se indican en el guión de la práctica 1. En ambos modos (y para las mallas que dispongan de ellas) se enviarán junto con cada vértice sus coordenadas de textura, pero no se enviarán los colores de los vértices ni de los triángulos. Respecto a las normales, se deben usar las tablas calculadas con el mismo código de las prácticas anteriores. Se procede según el modo:

- *modo con iluminación y sombreado plano*: se activa el modo de sombreado plano de OpenGL, y se envían con *begin/end* los triángulos, usando la tabla de normales de triángulos.
- *modo con iluminación y sombreado de suave (Gouroud)*: se activa el modo de sombreado suave, y se envían los vértices usando la tabla de normales de vértices.

4.2.3. Asignación de coordenadas de textura en objetos obtenidos por revolución.

Una vez definidas las tablas de coordenadas de textura, se creará una nueva versión modificada del código o función que crea **objetos por revolución** de la práctica 3. En los casos que así se especifique, dicho código nuevo incluirá la creación de la tabla de coordenadas de textura.

Supongamos que el el perfil de partida tiene M vértices, numerados comenzando en cero. Las posiciones de dichos vértices serán: $\{p_0, p_1, \ldots, p_{M-1}\}$. Si suponemos que hay N copias del perfil, y que en cada copia hay M vértices, entonces el j-ésimo vértice en la i-ésima copia del perfil será $q_{i,j}$, con $i \in [0 \ldots N-1]$ y $j \in [0 \ldots M-1]$ y tendrá unas coordenadas de textura (s_i, t_j) (dos valores reales entre 0 y 1. La coordenada S (coordenada X en el espacio de la textura) es común a todos los vértices en una copia del perfil.

El valor de s_i es la coordenada X en el espacio de la textura, y está entre 0 y 1. Se obtiene como un valor proporcional a i, haciendo $s_i = i/(N-1)$ (la división es real), de forma que s_i va desde 0 en el primer perfil hasta 1 en el último de ellos.

El valor de t_j es la coordenada Y en el espacio de la textura, y también está entre 0 y 1. Su valor es proporcional a la distancia d_j (medida a lo largo del perfil), entre el primer vértice del mismo (vértice 0), y dicho vértice j -ésimo. Las distancia se definen como sigue: $d_0 = 0$ y $d_{j+1} = d_j + \|p_{j+1} - p_j\|$, y se pueden calcular y almacenar en un vector temporal durante la creación de la malla. Conocidas las distancias, la coordenada Y de textura (t_j) se obtiene como $t_j = d_j/d_{M-1}$

Hay que tener en cuenta que en este caso, la última copia del perfil (la copia $N-1$) es la que corresponde a una rotación de 2π radianes o 360^o. Esta copia debe ser distinta de la primera copia, ya que, si bien sus vértices coinciden con aquella, las coordenadas de textura no lo hacen (en concreto la coordenada S o X), debido a que en la primera copia

necesariamente dicha coordenada es $s_0 = 0$ y en la última es $s_{N-1} = 1$. Este es un ejemplo de duplicación de vértices debido a las coordenadas de textura.

4.2.4. Fuentes de luz

El siguiente paso será diseñar e implementar las **fuentes de luz** de la escena (al menos dos). Al menos una de las dos fuentes será direccional y tendrá su vector de dirección definido en coordenadas esféricas por dos ángulos α y β , donde β es el ángulo de rotación en torno al eje X (latitud), y α es la rotación en torno al eje Y (longitud). Cuando α y β son ambas 0, la dirección coincide con la rama positiva del eje Z. el vector de dirección de la luz se considerará fijado al sistema de referencia de la camara (la luz se "mueveçon el observador).

Para implementar las fuentes se definen en el programa las variables globales, estructuras o instancias de clase que almacenan los parámetros de cada una de las fuentes de luz que se planean usar. Para cada fuente de luz, se debe guardar: su color S_i (una terna RGB), su tipo (un valor lógico que indique si es direccional o posicional), su posición (para las fuentes posicionales), y su dirección en coordenadas esféricas (para las direccionales). Al menos para la fuente dada en coordenadas esfñericas, se guardarán asimismo los valores α y β

Para cada fuente de luz, se definirá una función (o habrá método de clase común) que se encargue de activarla. Aquí *activar* una fuente significa que en OpenGL se habilite su uso y que se configuran sus parámetros en función del valor actual de las variables globales que describen dicha fuente.

4.2.5. Carga, almacenamiento y visualización de texturas.

A continuación se diseñarán e implementarán las **texturas** de la escena. Se incluirán en el programa las variables o instancias que almacenan los parámetros de cada una de las texturas que se planean usar. Para cada textura, se debe guardar un puntero a los pixels en memoría dinámica, el identificador de textura de OpenGL, un valor lógico que indique si hay generación automática de coordenadas de textura o se usa la tabla de coordenadas de textura, y finalmente los 8 parámetros (dos arrays de 4 flotantes cada uno) para la generación automática de texturas.

Después se definirán en el programa las funciones o métodos para *activar* cada una de las texturas que se planean usar. *Activar* significa habilitar las texturas en OpenGL, habilitar el identificador de textura, y si la textura tiene asociada generación automática de coordenadas, fijar los parámetros OpenGL para dicha generación.

Hay que tener en cuenta que, la primera vez que se intente activar una textura, se debe *crear* la textura, esto significa que se deben leer los texels de un archivo y enviarlos a la GPU o la memoria de vídeo, inicializando el identificador de textura de OpenGL. Es importante no repetir la creación de la textura una vez en cada cuadro, sino hacerlo exclusivamente la primera vez que se intenta activar.

Para la lectura de los texels de un archivo de imagen, se puede usar, entre otras, la funcionalidad que se proporciona en la clase `jpg::Imagen`, que sirve para cargar JPGs y

se usa con el siguiente esquema:

```
#include "jpg_imagen.hpp"
   ....
// declara puntero a imagen (pimg)
jpg::Imagen * pimg = NULL ;
   ....
// cargar la imagen (una sola vez!)
pimg = new jpg::Imagen("nombre.jpg");
   .....
// usar con:
tamx = pimg->tamX(); // num. columnas (unsigned)
tamy = pimg->tamY(); // num. filas (unsigned)
texels = pimg->leerPixels(); // puntero texels (unsigned char *)
```

En memoria, cada texel es una terna rgb (`GL_RGB`), y cada componente de dicha terna es de tipo `GL_UNSIGNED_BYTE`.

Para poder usar estas funciones, es necesario tener estos archivos fuente:

- Cabeceras:
  - `jpg_imagen.hpp` (métodos públicos)
  - `jpg_jinclude.hpp`
- Unidades de compilación (se deben compilar y enlazar con el resto de unidades):
  - `jpg_imagen.cpp`
  - `jpg_memsrc.cpp`
  - `jpg_readwrite.cpp`

Este código usa la librería `libjpeg`, que debe enlazarse tambien (con el *switch* `-ljpeg` en el enlazador/compilador de GNU). Esta librería puede instalarse en cualquier distribución de linux usando paquetes tipo rpm o o debian. Al hacer la instalación se debe usar la versión de desarrollo (incluye las cabeceras).

4.2.6. Materiales.

El siguiente paso será diseñar e implementar los **materiales** de la escena, entendiendo como un *material* a un conjunto de valores de los parámetros del modelo de iluminación de OpenGL relativos a la reflectividad y brillo de la superficie de los objetos.

Será necesario definir en el programa las variables o instancias que almacenan los parámetros de cada uno de los materiales que se planean usar. Para cada material, se almacenará la reflectividad ambiental (M_A) la difusa (M_D), la especular (M_S) y el exponente de brillo (e). Cada reflectividad es un array con 4 valores `float`: las tres componentes RGB más la cuarta componente (opacidad) puesta a 1 (opaco). Asimismo, un material puede llevar asociada una textura o no llevarla. Si la lleva, junto con el material se almacenan los parámetros de dicha textura (un puntero a ellos), descritos más arriba en el documento. Si el material no lleva textura, el citado puntero será nulo.

Después se definirán funciones o métodos para activar cada uno de los materiales. Cuando se activa un material, se habilita la iluminación en OpenGL, y se configuran los parámetros de material de OpenGL usando las variables o instancias descritas en el párrafo anterior. Para los materiales que lleven asociada una textura, se llamará a la función o método para activar dicha textura, descrita anteriormente. Si el material no lleva textura, se deben deshabilitar las texturas en OpenGL.

4.2.7. Escena completa

Figura 4.1: Vista de la escena completa.

El último paso será definir la **función principal de visualización** de la escena para esta **práctica 4**. La primera vez que se invoque, se crearán en memoria las variables o instancia de clase que representan los objetos geométricos, las fuentes de luz, y los materiales y texturas de la escena. La escena estará compuesta de varios objetos (ver figura 4.1), en concreto los siguientes:

- Objeto **lata** (ver figura 4.2). Este objeto está compuesto de tres sub-objetos. Cada uno de ellos es una malla distinta, obtenida por revolución de un perfil almacenado en un archivo ply. En concreto:
  - `lata-pcue.ply` : perfil de la parte central, la que incorpora la textura de la lata (archivo `text-lata-1.jpg`). Es un material difuso-especular.
  - `lata-psup.ply` : tapa superior metálica. No lleva textura, es un material difuso-especular de aspecto metálico (ver figura 4.3, derecha).

- `lata-pinf.ply` : base inferior metálica, sin textura y del mismo tipo de material (ver figura 4.3, izquierda).

- Objetos **peón**: son tres objetos obtenidos por revolución, usando el mismo perfil de la práctica 2. Los tres objetos comparten la misma malla, solo que instanciada tres veces con distinta transformación y material en cada caso:

  - Peón **de madera**: con la textura de madera difuso-especular, usando generación automática de coordenadas de textura, de forma que la coordenada *s* de textura es proporcional a la coordenada X de la posición, y la coordenada *t* a Y (ver figura 4.4, izquierda). La textura está en el archivo `text-madera.jpg` (ver figura 4.4, derecha).
  - Peón **blanco**: sin textura, con un material puramente difuso (sin brillos especulares), de color blanco (ver figura 4.5, izquierda).
  - Peón **negro**: sin textura, con un material especular sin apenas reflectividad difusa (ver figura 4.5, izquierda).

Figura 4.2: Objeto lata obtenido con tres mallas creadas por revolución partir de tres perfiles en archivos ply (izquierda). La malla correspondiente al cuerpo incorpora la textura de la imagen de la derecha.

4.2.8. Implementación

Para esta práctica se pueden seguir básicamente dos estrategias de diseño alternativas:

- Asociar cada textura, material o fuente de luz con una función C/C++ específica que se encarga de activarla (la primera vez que se intente activar una textura, se cargará en memoria). En este caso, el código de la práctica sirve únicamente para visualizar

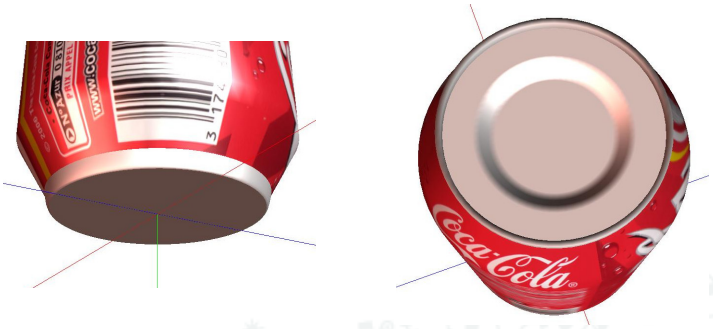

Figura 4.3: Vista ampliada de las tapas metálicas inferior (izquiera) y superior (derecha) de la lata (ambas sin textura)

Figura 4.4: Vista del objeto tipo peón (izquierda) y la textura de madera que se le aplica (derecha). La textura se aplica usando generación automática de coordenadas de textura (la textura se proyecta en el plano XY).

los modelos de aspecto que están implicitamente definidos en dicho código, al estar unidos modelos y código.

- Asociar cada textura, material o fuente de luz con una instancia de una clase o con una variable `struct` que contiene los parámetros que la definen. Se deben definir clases o tipos `struct` para cada tipo de elemento. Para la activación, se usará un método de clase o bien una función que acepta como parámetro un puntero a la struct. Por tanto, este código será más flexible que con la opción anterior, ya que se separa la definición de los modelos de aspecto de su visualización, al igual que ya hicimos con el modelo geométrico, en las cual se separa el almacenamiento de la malla en tablas del código que las visualiza.

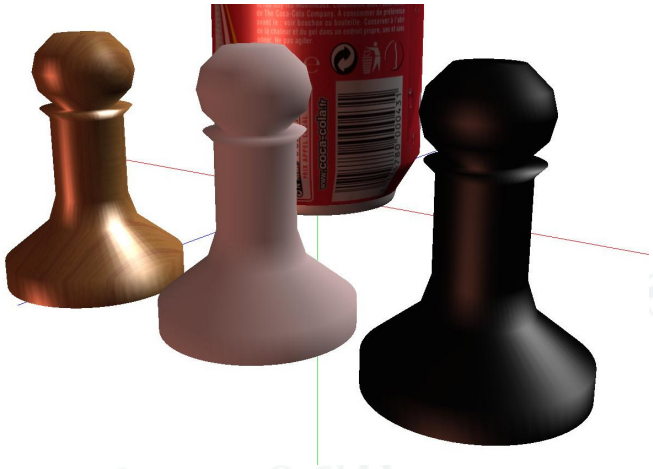

Figura 4.5: Vista ampliada de los tres objetos tipo peón.

4.2.9. Resultados entregables

El alumno entregará un programa que represente y dibuje la escena compuesta por la lata y los tres peones con los materiales y texturas especificados en la sección 4.2.7.

El programa permitirá, pulsando la tecla 4, entrar y salir del *modo práctica 4*. En modo práctica 4, la función gestora (*callback*) de redibujado llamará a la función principal de visualización descrita en la sección 4.2.7. Cuando no se está en el modo de la práctica 4, el programa se comporta igual que en la práctica anterior, la práctica 3, visualizando el objeto jerárquico o los objetos de las prácticas 1 y 2.

En el modo de la práctica 4, se pueden usar las teclas para mover la cámara (igual que en las anteriores prácticas), y además se procesarán teclas adicionales para mover la dirección de una de las fuentes de luz (la fuente direccional expresada en coordenadas polares con dos ángulos α y β, según se describe en la sección 4.2.4). Las teclas son:

- Tecla `A` : aumentar el valor de β
- Tecla `Z` : disminuir el valor de β
- Tecla `X` : aumentar el valor de α
- Tecla `C` : disminuir el valor de α

(da igual pulsarlas en mayúsculas o minúsculas)

4.3. Evaluación

La evaluación de la práctica, sobre 10 puntos, se hará del modo siguiente:

- Cálculo de las tablas de normales de caras y vértices (1,5 puntos).

- Incorporación de la tabla de coordenadas de textura a las mallas y su visualización (1,5 puntos)
- Código de asignación de coordenadas de textura al objeto de revolución (1,5 puntos)
- Código de carga y visualización de texturas (1,5 puntos)
- Definición correcta de materiales y su código de activación (2 puntos)
- Definición de fuentes de luz, del código de activación, y de la modificación interactiva de la dirección de una de ellas (2 puntos)

4.4. Extensiones

Como extensión, se propone incorporar texturas y fuentes de luz al objeto jerárquico creado para la práctica 3. A dicho objeto se le pueden aplicar distintas texturas y/o materiales a las distintas partes de forma que se incremente su grado de realismo, así como definir fuentes de luz.

Para llevar a cabo esta tarea, hay que tener en cuenta que el código que visualiza las mallas debe enviar las normales, y en los casos que corresponda las coordenadas de textura. Si se usa alguna librería para visualizar las mallas (como `glu`, `glut` u otras), debemos de asegurarnos que la librería envía normales y cc.tt., si esto no es así habrá que considerar otra librería o escribir el código de visualización de esas mallas.

4.5. Duración

Esta tercera práctica se desarrollará en 3 sesiones.

4.6. Bibliografía

- Mark Segal y Kurt Akeley; *The OpenGL Graphics System: A Specification (version 4.1)*; http://www.opengl.org/
- Edward Angel;*Interactive Computer Graphics. A top-down approach with OpenGl*; Addison-Wesley, 2000
- J. Foley, A. van Dam, S. Feiner y J. F. Hughes; *Computer Graphics: Principles And Practice, 2 Edition*; Addison-Wesley,1992
- P. Shirley y S. Marschner; *Fundamentals of Computer Graphics, 3rd Edition*; A K Peters Ltd. 2009.