Práctica 5

Texturas

5.1. Objetivos

El objetivo de esta práctica es conseguir un mayor realismo mediante el uso de las texturas. Para ello necesitamos hacer lo siguiente:

- Calcular las coordenadas de textura para un objeto sencillo
- Cargar una textura y visualizarla con/sin iluminación.

5.2. Desarrollo

El objetivo de esta práctica es conseguir un mayor realismo en la visualización de las escenas y para ello vamos a integrar el proceso de iluminación y el uso de texturas.

Aunque con la iluminación de la práctica 3 se obtiene una substancial mejora en el realismo, para mejorarlo recurrimos a las texturas, que consiste, de una manera muy general, en pegarle una foto a un objeto o parte del mismo. Para ello tendremos que ver cómo se relacionan imagen y objeto mediante las coordenadas de textura.

El siguiente paso será aumentar las estructuras de datos que representan las mallas poligonales 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 `std::vector<tupla2f>ct` 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.

5.2.1. Texturas

Imaginemos que queremos producir una imagen 2D realista: un buen pintor conseguirá un buen resultado con mucho esfuerzo, pero nada será mejor que una fotografía. Esta idea

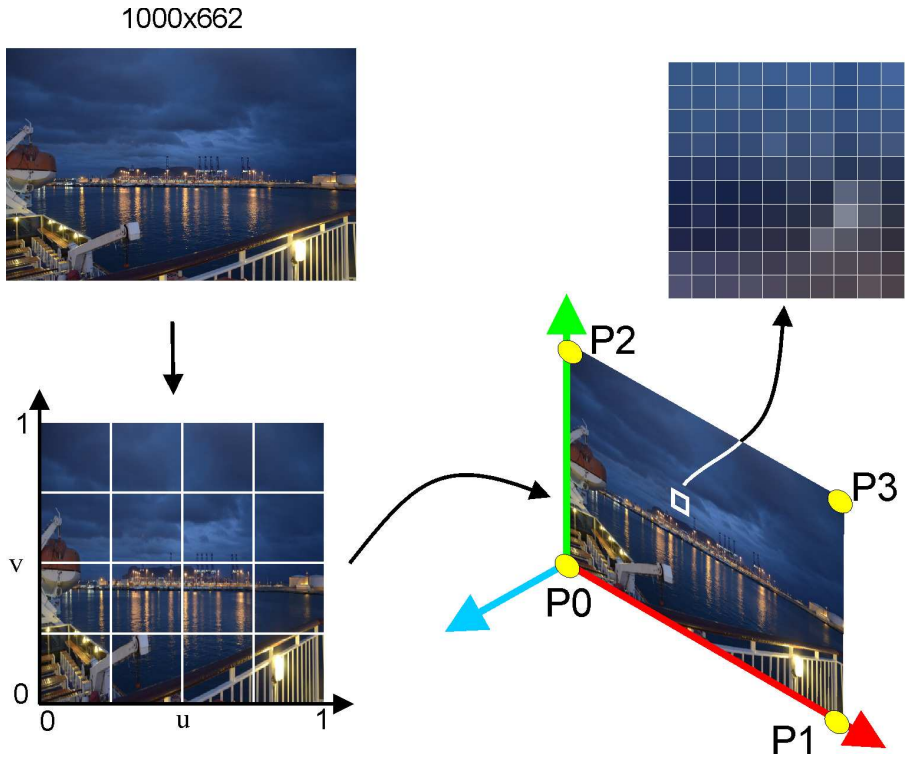

Figura 5.1: Pasos en la aplicación de una textura. DMP©

se puede extender a la visualización de modelos realistas: sólo con vértices y colores es muy difícil que consigamos un alto grado de realismo, y si se consigue sólo es posible con un gran número de vértices.

Pongamos un ejemplo. Imaginemos que queremos modelar un tablón de madera. La forma del mismo es muy fácil de modelar con un paralelepípedo, un cubo estirado. Con 8 vértices y 8 colores, poco se puede conseguir para simular la madera. La solución consistiría en incrementar el número de vértices hasta que consiguiéramos el nivel de detalle deseado.

La solución que se propone con las texturas es la siguiente: hacemos una foto de cada lado del tablón y cada foto es pegada a un lado del modelo. La geometría en sencilla pero la visualización es realista. No vamos a entrar en cómo se puede pega la imagen en el modelo, pero ayuda el pensar que la fotografía se ha imprimido en una tela que es deformable, de tal manera que la ajustamos al objeto.

Asignación de coordenadas de textura en un cuadro

Los pasos para aplicar una textura se muestran en la figura 5.1. El primer paso consiste en pasar una imagen matricial a un sistema de coordenadas normalizado, con coordenadas u y v, cumpliendo que $0 \leq u \leq 1$ y $0 \leq v \leq 1$. Esta normalización permite independizar el tamaño real de la imagen de su aplicación al modelo.

El siguiente paso consiste en asignarle a cada punto del modelo las coordenadas de tex-

tura correspondientes. En el ejemplo de la imagen, para representar un rectángulo necesitamos 2 triángulos. Si tenemos 4 puntos, P_0, P_1, P_2 P_3, el triángulo T_0 podría estar compuesto por los puntos $(P_2, P_0, P1)$ y el triángulo T_1 por los puntos $(P_1, P_3, P2)$. Dado que queremos mostrar toda la textura, eso implica la siguiente asignación de coordenadas de textura:

- $(0,0) \rightarrow P_0$
- $(1,0) \rightarrow P_1$
- $(0,1) \rightarrow P_2$
- $(1,1) \rightarrow P_3$

Si solo quisiéramos mostrar la parte central, las coordenadas podrían ser las siguientes:

- $(0{,}25, 0{,}25) \rightarrow P_0$
- $(0{,}75, 0{,}25) \rightarrow P_1$
- $(0{,}25, 0{,}75) \rightarrow P_2$
- $(0{,}75, 0{,}75) \rightarrow P_3$

Es importante observar con estos dos ejemplos que la diferencia en lo que se muestra se ha conseguido simplemente cambiado las coordenada de textura, no las coordenadas de los puntos.

Una vez que se ha hecho la correspondencia, OpenGL se encarga del trabajo duro, realizando la correspondencia entre lo píxeles de la imagen de entrada y los píxeles de la imagen de salida, resolviendo los distintos problemas de escala que hay, realizando la interpolación, ajustando la perspectiva, si la hay, etc.

Un detalle a tener en cuenta es que los formatos de imágenes suelen usar un sistema de coordenadas izquierdo, con el origen en la esquina superior izquierda mientras que OpenGL usa un sistema de coordenadas derecho con el origen en la esquina inferior izquierda. Por ello aplicamos una operación de reflejo horizontal.

Asignación de coordenadas de textura en objetos obtenidos por revolución.

Hay que modificadar la clase `ObjRevolución` de la práctica 2, añadiendo un método que calcule las coordenadas de textura para cada uno de los vértices y lo almacene en su tabla `ct`.

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 instancia del perfil.

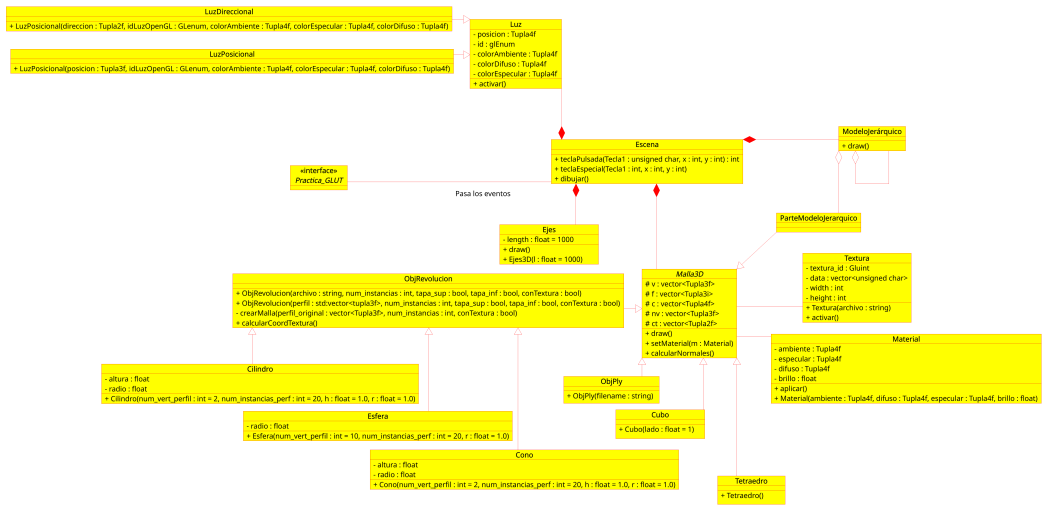

Figura 5.2: Diagrama de clases actualizado

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 $t_j = d_j/d_{M-1}$

Para permitir el uso se texturas, es necesario modificar también el método de generación de la malla del objeto de revolución. Según se hacía en la práctica 2, la última instancia del perfil (la instancia $N-1$) **no** es la que corresponde a una rotación de 2π radianes o 360^o, sino que se quedaba $2\pi/N$ antes.

Si lo dejamos así, lo que ocurrirá es que en una tira vertical del objeto de revolución estaremos uniendo de golpe las coordenadas de textura $s_n = 1 - 1/N$çon la coordenada de textura $s_0 = 0$ (la primera instancia), y genera una costura a todas luces extraña. Debemos crear una instancia extra del perfil, que coincidirá en coordenadas geométricas con la instancia 0 pero tendrá diferentes coordenadas de textura, de forma que en la primera instancia necesariamente dicha coordenada es $s_0 = 0$ y en la última será $s_N = 1$. Este es un ejemplo de duplicación de vértices debido al uso de las coordenadas de textura.

5.2.2. Implementación

A continuación se diseñarán e implementarán las **texturas** de la escena, como una clase nueva `Textura`. Todo objeto `Malla3D` tendrá un atributo `textura` que, en caso de no ser nulo, será utilizado en su visualización.

```
class Textura {
```

```
        GLuint textura_id = 0;
        std::vector<unsigned char> data;
        int width,height;


 public.
        Textura(string archivo);
        activar();
}
```

```
class Malla3D {
        // ...
        Textura *textura=nullptr;
        std::vector<Tupla2f> ct
        // ...
}
```

Para cada textura, se debe guardar un puntero a los pixels en memoría dinámica, y el identificador de textura de OpenGL. Después se definirá en el programa el método para *activar* una textura. *Activar* aquí 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. Este método no se puede ejecutar sin que la máquina de OpenGL esté inicializada.

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 cada vez que se dibuja, sino hacerlo exclusivamente la primera vez que se intenta activar.

Carga de texturas y envío a la memoria de vídeo o la GPU

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();          // núm. columnas (unsigned)
tamy = pimg->tamY();          // núm. 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:

- `jpg_imagen.hpp` declaración de métodos públicos.
- `jpg_jinclude.hpp` declaración de algunas funciones auxiliares
- 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).

Cálculo y almacenamiento de coordenadas de textura.

Respecto de las coordenadas de textura, la clase `Malla3D` debe de incorporar un vector stl o un puntero a un array de C/C++ con las coordenadas de textura. Tendrá tantas entradas como vértices, y en cada una de ellas se guardan una tupla de dos valores flotantes (`Tupla2f`). Si una malla no tiene coordenadas de textura (porque no lleva coordenadas de textura o bien porque se usa generación automática de coordenadas), el vector stl estará vacío o el puntero al array es nulo.

De esta forma, la función `draw` incluirá también, en caso de existir las coordenadas de textura, las siguientes líneas:

```
glEnableClientState( GL_TEXTURE_COORD_ARRAY );
glTexCoordPointer( 2, GL_FLOAT, 0, ct);
```

Cuando se crea una malla por revolución de un perfil, se puede especificar si se desean generar las coordenadas de textura o no se desean. Esto se hace mediante un parámetro booleano que se añade al constructor de la clase `ObjRevolucion`. En dicho constructor, si se desean coordenadas de textura, dichas coordenadas se calcularán como se describe en el guión. Reiteramos que hay que tener en cuenta que, cuando se requieren coordenadas de textura, la primera y última copia del perfil no coinciden.

Activación de las texturas

Las ordenes glEnable y glDisable se pueden usar para activar o desactivar toda la funcionalidad de OpenGL relacionada con las texturas, que se encuentra inicialmente desactivada:

```
glEnable( GL_TEXTURE_2D ) ; / / habilita texturas
glDisable( GL_TEXTURE_2D ) ; / / deshabilita texturas
```

Cuando se habilitan las texturas hay una textura activa en cada momento, que se consulta cada vez que un polígono se proyecta en un pixel, antes de calcular el color de dicho pixel:

- con iluminación **desactivada**, el color de la textura sustituye al especificado con glColor
- con iluminación **activada**, el color de la textura sustituye a las reflectividades del material (usualmente a la difusa y la ambiental).

OpenGL puede gestionar más de una textura a la vez. Para diferenciarlas usa un valor entero único para cada una de ellas, que se denomina identificador de textura (texture name) (de tipo `GLuint`). Para crear o generar un nuevo identificador de textura único (distinto de cualquiera ya existente) usamos:

```
GLuint idTex ;
glGenTextures( 1, &idTex ); // idTex almacena el nuevo identificador
```

Para crear `n` nuevos identificadores de textura (en un array de GLuint), hacemos:

```
GLuint arrIdTex[n] ;
glGenTextures( n, arrIdTex);
```

En el estado interno de OpenGL hay en cada momento un identificador de textura activa:

- Cualquier operación de visualización de primitivas usará la textura asociada a dicho identificador.
- Cualquier operación de configuración de la funcionalidad de texturas se referirá a dicha textura activa.

Para cambiar el identificador de textura activa podemos hacer:

```
glBindTexture( GL_TEXTURE_2D, idTex ); // activa textura con id 'idTex'
```

En cualquier momento podemos especificar cuál será la imagen de textura asociada al identificador de textura activa, con glTexImage2D:

```
glTexImage2D( GL_TEXTURE_2D,
                          0, // nivel de mipmap (para imágenes multiresolución)
                          GL_RGB, // formato interno
                          ancho, // núm. de columnas (potencia de dos: 2n)
                          alto, // núm de filas (potencia de dos: 2m)
                          0, // tamaño del borde, usualmente es 0
                          GL_RGB, // formato y orden de los texels en RAM
                          GL_UNSIGNED_BYTE, // tipo de cada texel
                          texels // puntero a los bytes con texels (void *)
);
```

Al llamar a esta función, OpenGL leerá los bytes de la RAM y los copiará en otra memoria (típicamente la memoria de vídeo o de la GPU, en un formato interno). Si es posible usar GLU, hay una alternativa preferible a glTexImage2D que no requiere imágenes de tamaño potencia de dos, y que además genera automáticamente versiones a múltiples resoluciones (mipmaps)

```
gluBuild2DMipmaps( GL_TEXTURE_2D,
                                    GL_RGB,  // formato interno
                                    ancho,  // núm. de columnas (arbitrario)
                                    alto,  // núm de filas (arbitrario)
                                    GL_RGB,  // formato y orden de los texels en RAM
                                    GL_UNSIGNED_BYTE,  // tipo de cada texel
                                    texels  // puntero a los bytes con texels (void *)
);
```

Esta función hace varias versiones de la imagen, para usar una u otra a distintas resoluciones, en función de los píxeles que ocupa en pantalla el modelo.

5.2.3. Interacción

Tras esta práctica tendremos las siguientes teclas para la interacción con el sistema:

- Tecla 'V': activa el modo de *selección de modo de visualización*, en cuyo modo, si se pulsa:
  - Tecla 'P': Se activa/desactiva la visualización en modo puntos, y desactiva iluminación
  - Tecla 'L': Se activa/desactiva la visualización en modo líneas, y desactiva iluminación
  - Tecla 'S': Se activa/desactiva la visualización en modo sólido (por defecto), y desactiva iluminación
  - Tecla 'A': Se activa/desactiva la visualización en modo ajedrez, y desactiva iluminación
  - Tecla 'I': Se activa la visualización con iluminación
    - Se usarán las teclas '0' a '7' para activar las diferentes luces que haya en la escena.
    - Tecla 'A': activará el modo variación del ángulo alfa
    - Tecla 'B': activará el modo variación del ángulo beta
    - Tecla '>': incrementa el ángulo (según lo último pulsado, A o B)
    - Tecla '<': decrementa el ángulo (según lo último pulsado, A o B)
    - Tecla 'P': se anima automáticamente la luz puntual
  - Tecla 'Q': Se sale del modo *selección de modo de visualización*.
- Tecla 'A': Se activa/desactiva la animación automática de los objetos de la escena (jerárquicos o no)

  - Tecla '+' aumentará la velocidad de la animación (opcional)
  - Tecla '-' disminuirá la velocidad de la animación (opcional)
- Tecla 'M': Se mueve manualmente el grado de libertad del objeto jerárquico (desactivando animación automática)
  - Se usarán las teclas '0' a `numGradosLibertad` para seleccionar cada uno de los grados de libertad
  - Tecla '+' aumentará el valor aplicado al grado de libertad seleccionado
  - Tecla '-' disminuirá el valor aplicado al grado de libertad seleccionado
  - Tecla 'Q': Se sale del modo de animación manual del modelo jerárquico
- Tecla 'Q': Se sale del programa

Es importante comprobar los cambios incluidos en el nuevo esqueleto.

5.3. Evaluación

Elementos de la Práctica 5	**Puntuación máxima**
Asignación correcta de coordenadas de texturas en cuadro	0.20
Visualización correcta al menos un objeto con la textura	0.15
Animación de la luz puntual	0.20
Extra: Hay en la escena un cilindro texturizado (p.ej. lata de refresco) o una esfera texturizada (p.ej. tierra)	0.15

5.4. Duración

La práctica se desarrollará en 2 sesiones.

5.5. 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.