Dpto. de Lenguajes y Sistemas Informáticos
Escuela Técnica Superior de Ingenierías
Informática y Telecomunicación

Prácticas de Informática Gráfica

Autores:
Antonio López
Domingo Martín
Francisco Javier Melero
Juan Carlos Torres
Carlos Ureña

Curso 2019/20

La Informática Gráfica

La gran ventaja de los gráficos por ordenador, la posibilidad de crear mundos virtuales sin ningún tipo de límite, excepto los propios de las capacidades humanas, es a su vez su gran inconveniente, ya que es necesario crear toda una serie de modelos o representaciones de todas las cosas que se pretenden obtener que sean tratables por el ordenador.

Así, es necesario crear modelos de los objetos, de la cámara, de la interacción de la luz (virtual) con los objetos, del movimiento, etc. A pesar de la dificultad y complejidad, los resultados obtenidos suelen compensar el esfuerzo.

Ese es el objetivo de estas prácticas: convertir la generación de gráficos mediante ordenador en una tarea satisfactoria, en el sentido de que sea algo que se hace "con ganas".

Con todo, hemos intentado que la dificultad vaya apareciendo de una forma gradual y natural. Siguiendo una estructura incremental, cada práctica irá añadiendo conceptos para la consecución del objetivo final: un proyecto completo que aglutine todos los conceptos de la asignatura.

Esperamos que las prácticas propuestas alcancen los objetivos y que sirvan para enseñar los conceptos básicos de la Informática Gráfica, y si puede ser entreteniendo, mejor.

Parte I

Introducción a las prácticas de la asignatura

Práctica	Fecha recomendada de inicio
Práctica 1. Programación con OpenGL	25/09/2019
Práctica 2. Modelos poligonales y transformaciones	2/10/2019
Práctica 3. Iluminación	16/10/2019
Práctica 4. Modelos jerárquicos y animación	30/10/2019
Práctica 5. Texturas	13/11/2019
Práctica 6. Control de cámara e interacción	27/11/2019

Tabla 1: Fechas recomendadas de inicio de cada práctica

0.1. El proyecto final

El proyecto final consiste en el desarrollo por el estudiante de un programa para visualización 3D, desarrollo que requiere conocer los distintos conceptos y elementos aprendidos en la teoría y las prácticas de la asignatura.

En esta sección se reflejarán los items a valorar en el proyecto, y los recursos (plantillas de código fuente, archivos, etc...) necesarios para su desarrollo. Antes del **10 de octubre**, el alumno deberá indicar a su profesor de prácticas un breve enunciado de la temática sobre la que versará su proyecto, y como piensa, a priori, abordar los diferentes aspectos que serán evaluados. Dicha descripción se realizará a través del espacio habilitado en PRADO.

0.2. Evaluación

El alumno podrá solicitar con antelación a su sesión de prácticas la evaluación de uno o más elementos de la rúbrica de evaluación del proyecto final. Dicha evaluación se realizará durante la sesión de prácticas, y de no ser posible por falta de tiempo, se realizará en la hora de tutorías más cercana. En estas entregas y defensas el profesor de prácticas podrá plantear a los estudiantes cuestiones, problemas o modificaciones sobre el código entregado, y verificará la originalidad de dicho código y la comprensión de los conceptos por parte de los estudiantes.

0.3. Desarrollo de las sesiones de prácticas

Siguiendo el calendario mostrado en la tabla 1 se explicarán en el aula de prácticas los conceptos a desarrollar en cada una de las prácticas. Se proporcionará al alumno un esqueleto que permita disponer de un código mantenible y escalable, siguiendo el paradigma de la programación orientada a objetos.

Para el desarrollo de esta práctica se entrega el esqueleto de una aplicación gráfica basada en eventos, mediante GLUT, y con la parte gráfica realizada por OpenGL, siendo el lenguaje de programación C++.

0.4. Elementos a evaluar del proyecto final

En cada una de las prácticas se verán unos conceptos que después deberán incluirse en el proyecto final. Dicho proyecto final tiene una valoración total de 5 puntos, desglosados como sigue:

Elementos de la Práctica 1	**Puntuación máxima**
Clase Malla3D (implementación correcta)	0.03
Clase Cubo (Creación correcta de Geometría y Topología)	0.03
Clase Tetraedro (Creación correcta de Geometría y Topología)	0.03
Dibujado en modo inmediato (glDrawElements)	0.03
Dibujado en modo diferido (VBO)	0.08
Uso de ColorArray para el color	0.03
Visualización modo puntos	0.04
Visualización modo alambre	0.04
Visualización modo sólido	0.04
Visualización modo ajedrez	0.05
CULL_FACE habilitado	0.05
Extra: Visualización simultánea puntos + lineas + sólido (cualquier combinación de ellos)	0.05

Elementos de la Práctica 2	**Puntuación máxima**
Carga y visualiza PLY (con clase propia)	0.08
Clase ObjRevolucion	0.05
Constructor de revolución a partir de perfil .ply	0.04
Constructor de revolución a partir de vector de puntos	0.04
Generación correcta de geometría	0.05
Generación correcta de topología en el sentido del ejemplo	0.07
Generación correcta de topología en cualquier sentido	0.08
Detección de existencia de tapas	0.10
Constructores con argumento para crear o no tapas	0.08
Visualización opcional de tapas en tiempo de ejecución	0.08
Varios objetos simultáneos en la escena	0.08
Extra: rotación de perfil en cualquier eje	0.075

Elementos de la Práctica 3	**Puntuación máxima**
Cálculo correcto de las normales en los vértices	0.15
Los ejes se siguen viendo en color plano	0.10
Uso de array de normales	0.05
Implementación correcta clase Luz (y subclases)	0.10
Hay al menos una luz direccional	0.15
Hay al menos una luz puntual	0.15
Modo de visualización sombreado suave	0.15
Hay al menos tres materiales distintos en la escena simultáneamente	0.10

Elementos de la Práctica 4	**Puntuación máxima**
Diseño manual del grafo del modelo jerárquico	0.25
Estructura de clases del modelo jerárquico	0.20
Se conservan los modos de visualización	0.10
Movimiento paso a paso con teclado 3 grados de libertad (al menos una rotación y una traslación)	0.20
Animación automática de los 3 grados de libertad	0.25
Extra: Aumento/Disminución de velocidad general	0.05
Extra: Aumento/Disminución de velocidad para cada grado por separado	0.05

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

Elementos de la Práctica 6	**Puntuación máxima**
Hay una clase cámara que almacena sus parámetros intrínsecos y extrínsecos	0.20
Hay al menos tres cámaras en la escena (obligatorio: una ortográfica y una perspectiva)	0.15
La cámara activa se mueve en torno al objeto seleccionado con el ratón	0.25
Se puede hacer zoom con cada cámara	0.15
Se seleccionan objetos en la escena iluminada con materiales	0.25
La camara activa, sin objeto seleccionado, se mueve en primera persona	0.15
Las cámaras conservan su estado al pasar de una a otra	0.10
Extra: Los objetos seleccionables se visualizan de forma especial	0.125

0.5. Prerequisitos software

0.5.1. Sistema Operativo Linux

En esta sección se detallan las herramientas software necesarias para realizar las prácticas en los sistemas operativos Linux (Ubuntu u otros) y macOS (de Apple). Respecto al sistema operativo Windows, las prácticas se pueden realizar en Ubuntu ejecutándose en una máquina virtual, o bien instalando Visual Studio.

Compiladores

Para hacer las prácticas de esta asignatura en Ubuntu, en primer lugar debemos de tener instalado algún compilador de C/C++. Se puede usar el compilador de C++ open source CLANG o bien el de GNU (ambos pueden coexistir). Podemos usar `apt`para instalar el paquete `g++` (compilador de GNU) o bien `clang` (compilador del proyecto LLVM). Además, si no está disponible, es conveniente instalar el paquete `make` que proporciona la orden del mismo nombre. Todo esto lo podemos hacer con:

```
sudo apt install g++
sudo apt install make
```

OpenGL

Respecto de la librería OpenGL, es necesario tener instalado algún driver de la tarjeta gráfica disponible en el ordenador. Incluso si no hay tarjeta gráfica disponible (por ejemplo en un máquina virtual eso puede ocurrir), es posible usar OpenGL implementado en software (aunque es más lento). Lo más probable es que tu instalación ya cuente on el driver apropiado para la tarjeta gráfica.

En cualquier caso, en ubuntu, para verificar la tarjeta instalada y ver los drivers recomendados y/o posibles para dicha tarjeta, se puede usar esta orden:

```
sudo ubuntu-drivers devices
```

Lo más fácil es instalar automáticamente el driver más apropiado, se puede usar la orden:

```
sudo ubuntu-drivers autoinstall
```

Librería GLEW

La librería GLEW es necesaria en Linux para que las funciones de la versión 2.1 de OpenGL y posteriores pueden ser invocadas (inicialmente esas funciones abortan al llamarlas). GLEW se encarga, en tiempo de ejecución, de hacer que esas funciones esten correctamente enlazadas con su código.

Para instalarla, se puede usar el paquete debian `libglew-dev`. En Ubuntu, se usa la orden

```
sudo apt install libglew-dev
```

Librería GLUT

La librería GLUT se usa para gestión de ventanas y eventos de entrada. Se puede instalar con el paquete debian `freeglut3-dev`. En Ubuntu, se puede hacer con:

```
sudo apt install freeglut3-dev
```

Librería JPEG

Esta librería sirve para leer y escribir archivos de imagen en formato jpeg (extensiones `.jpg` o `.jpeg`). Se usará para leer las imágenes usadas como texturas. La librería se debe instalar usando el paquete debian `libjpeg-dev`. En Ubuntu, se puede hacer con la orden

```
sudo apt install libjpeg-dev
```

0.5.2. Sistema Operativo macOS

Compilador

En primer lugar, para poder compilar los programas fuente en C++, debemos asegurarnos de tener instalado y actualizado **XCode** (conjunto de herramientas de desarrollo de Apple, incluye compiladores de C/C++ y entorno de desarrollo). Una vez instalado XCode, usaremos el compilador de C++ incorporado. Este compilador se invoca desde la linea de órdenes con la orden `clang++` (que es equivalente y tiene los mismos parámetros que la orden `g++` en Linux).

Librerías OpenGL, GLU, GLEW y GLUT

La librería GLEW no es necesaria en este sistema operativo. Respecto a las librerías OpenGL, GLU y GLUT, tanto las cabeceras como la implementación de OpenGL (driver) ya vienen instaladas con el entorno desarrollo XCode. Particular, y usando la terminología de Apple, esos archivos están incorporados en los *frameworks* OpenGL y GLUT.

Librería JPEG

Es necesario instalar los archivos correspondientes a la versión de desarrollo de la librería de lectura de jpegs. Respecto a esta librería para jpegs, se puede compilar el código fuente de la misma. Para esto, basta con descargar el archivo con el código fuente de la versión más moderna de la librería a un carpeta nueva vacía, y después compilar e instalar los archivos. Se puede hacer con estas órdenes:

```
mkdir carpeta-nueva-vacia
cd carpeta-nueva-vacia
curl --remote-name http://www.ijg.org/files/jpegsrc.v9b.tar.gz
tar -xzvf jpegsrc.v9b.tar.gz
cd jpeg-9b
./configure
make
sudo make install
```

Estas ordenes se refieren a la versión 9b de la librería, para futuras versiones habrá que cambiar `9b` por lo que corresponda. Si todo va bien, esto dejará los archivos `.h` en `/usr/local/include` y los archivos `.a` o `.dylib` en `/usr/local/lib`.

Para poder compilar, debemos de asegurarnos de que el compilador y el enlazador tienen estas carpetas en sus paths de búsqueda, es decir, debemos de usar la opción `-I/usr/local/include` al compilar, y la opción `-L/usr/local/lib` al enlazar.

0.6. Introducción. Clases para tuplas

En el archivo de cabecera `tuplasg.h` están disponibles varios tipos de datos para tuplas de valores reales o enteros. Entre otros, se proporcionan estos tipos:

- `Tupla3f`: tuplas de 3 valores reales (tipo `float`, que es equivalente a `GLfloat`). Son adecuadas para coordenadas 3D de vértices, colores RGB y vectores normales.
- `Tupla2f`: tupla de 2 valores `float`. Es útil para coordenadas de vértices en 2D, o bien para coordenadas de textura de los vértices.
- `Tupla3i`: tupla de 3 valores enteros (tipo `int`). Es útil para la tabla de caras de las mallas indexadas.

0.7. Tipos de datos

Aquí vemos los distintos tipos de datos disponibles:

```
// adecuadas para coordenadas de puntos, vectores o normales en 3D
// también para colores (R,G,B)
Tupla3f  t1 ;   // tuplas de tres valores tipo verbfloat
Tupla3d  t2 ;   // tuplas de tres valores tipo verbdouble

// adecuadas para la tabla de caras en mallas indexadas
Tupla3i  t3 ;   // tuplas de tres valores tipo verbint
Tupla3u  t4 ;   // tuplas de tres valores tipo verbunsigned

// adecuadas para puntos o vectores en coordenadas homogéneas
// también para colores (R,G,B,A)
Tupla4f  t5 ;   // tuplas de cuatro valores tipo verbfloat
Tupla4d  t6 ;   // tuplas de cuatro valores tipo verbdouble

// adecuadas para puntos o vectores en 2D, y coordenadas de textura
Tupla2f  t7 ;   // tuplas de dos valores tipo verbfloat
Tupla2d  t8 ;   // tuplas de dos valores tipo verbdouble
```

0.8. Creación, consulta y actualización

Este trozo código válido ilustra las distintas opciones, para creación, consulta y modificación de tuplas:

```
float       arr3f[3] = { 1.0, 2.0, 3.0 } ;
unsigned    arr3i[3] = { 1, 2, 3 } ;

// declaraciones e inicializaciones de tuplas
Tupla3f  a( 1.0, 2.0, 3.0 ), b, c(arr3f) ;  // indeterminado
Tupla3i  d( 1, 2, 3 ), e, f(arr3i) ;        // indeterminado
```

```
// accesos de solo lectura, usando su posición o índice en la tupla (0,1,2,...),
// o bien varias constantes predefinidias para coordenadas (X,Y,Z) o colores (R,G,B):
float x1 = a(0), y1 = a(1), z1 = a(2),    //
      x2 = a(X), y2 = a(Y), z2 = a(Z),    // apropiado para coordenadas
      re = c(R), gr = c(G), bl = c(B) ;   // apropidado para colores

// conversiones a punteros
float *       p1 = a ; // conv. a puntero de lectura/escritura
const float * p2 = b ; // conv. a puntero de solo lectura

// accesos de escritura
a(0) = x1 ;  c(G) = gr ;

// escritura en un 'ostream' (cout) se escribe como: 1.0,2.0,3.0
cout << "la tupla 'a' vale: " << a << endl ;
```

0.9. Operaciones entre tuplas

En C++ se pueden sobrecargar los operadores binarios y unarios usuales (`+`, `-`, etc...) para operar sobre las tuplas de valores reales:

```
// declaraciones de tuplas y de valores escalares
Tupla3f  a,b,c ;
float    s,l ;

// operadores binarios y unarios de asignación/suma/resta/negación
a = b ;
a = b+c ;
a = b-c ;
a = -b  ;

// multiplicación y división por un escalar
a = 3.0f*b ;      // por la izquierda
a = b*4.56f ;     // por la derecha
a = b/34.1f ;     // mult. por el inverso

// otras operaciones
s = a.dot(b)       ;  // producto escalar (usando método dot)
s = ab             ;  // producto escalar (usando operador binario barra )
a = b.cross(c)     ;  // producto vectorial (solo para tuplas de 3 valores)
l = a.lengthSq()   ;  // calcular módulo o longitud al cuadrado
a = b.normalized(); // hacer a= copia normalizada de b (a=b/modulo de b) (b no cambia)
```

Práctica 1

Representación y visualización de objetos 3D sencillos

1.1. Objetivos

Con esta práctica se quiere que el alumno aprenda:

- A crear y utilizar estructuras de datos que permitan representar objetos 3D sencillos.
- A utilizar las primitivas de dibujo de OpenGL para dibujar los objetos.
- A distinguir entre lo que es crear un modelo y a lo que es visualizarlo.

1.2. Desarrollo

Para el desarrollo de esta práctica se entrega el esqueleto de una aplicación gráfica basada en eventos, mediante GLUT, y con la parte gráfica realizada por OpenGL, siendo el lenguaje de programación C++. La aplicación no sólo contiene el código de inicialización de OpenGL y la captura de los eventos principales, sino que se ha implementado una estructura de clases que permite representar objetos 3D, incluyendo unos ejes. También está implementada la funcionalidad que simula una cámara básica que se mueve con las teclas de cursor, para moverse, y página adelante y página atrás para acercarse y alejarse. Esta funcionalidad de cámara será sustiuida en la práctica 5 por cámaras completas.

El alumno deberá estudiar y comprender el código que se entrega, cuya estructura básica puede verse en la figura 1.1:

1.2.1. Definición de geometrías

Se deberá completar los constructores de las clases `Cubo` y `Tetraedro` que permitan disponer de tantos objetos cubo o tetraedros como quisiéramos en la escena. En dichos

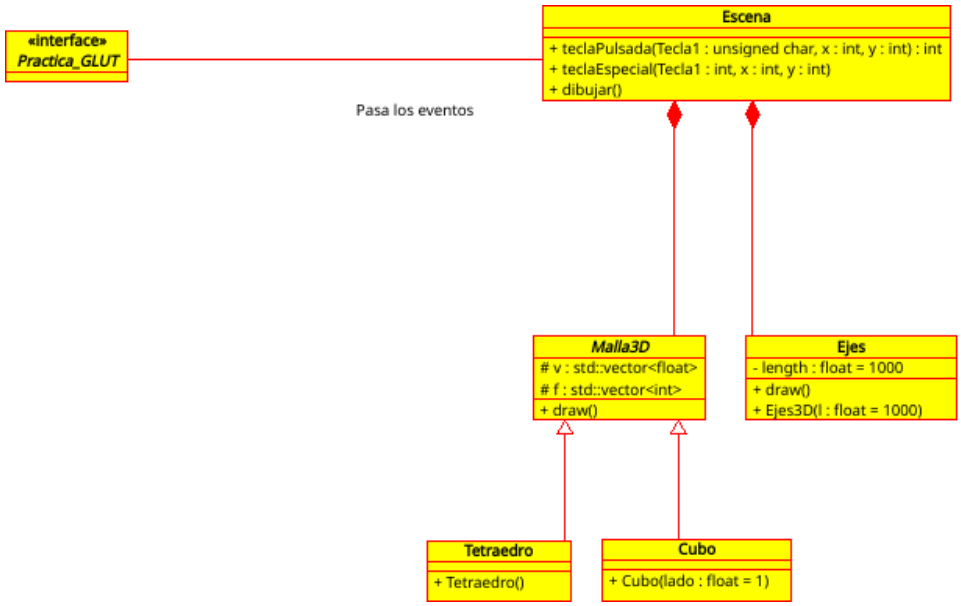


Figura 1.1: Diagrama de clases simplificado del código proporcionado

constructores tan sólo nos hemos de ocupar de completar el vector de coordenadas cartesianas de los vértices `v` y la lista de triángulos `f`. En el caso del cubo, dispondremos de un parámetro que será la longitud del lado del mismo.

En ambos casos, las figuras se crearán y visualizarán centradas en el origen de coordenadas.

Representación en memoria de Mallas Indexadas

Suponemos que las mallas indexadas se almacenan en memoria como instancias de la clase `Malla3D`. Esta clase tiene como variables de instancia privadas, como mínimo, la tabla de coordenadas de vértices y la tabla de triángulos.

Estas tablas se pueden declarar usando vectores de la librería estándar de C++. En cada entrada contienen tuplas (usando los tipos para tuplas que ya hemos visto). Es decir, se declaran así:

```
std::vector<Tupla3f> v ;      // coords de vértices (3 valores 'float' por vértice)
std::vector<Tupla3i> f;  // indices de vertices de triángulos
                                                    // (3 valores 'int' por cada
cara (triángulo)
```

1.2.2. Visualización

Objetivos

Se pide al alumno desarrollar al menos el modo de visualización con `glDrawElements` disponible en la guía didáctica de la asignatura. Además, se propone usar dibujar

los Vertex Buffer Objects (VBOs), también denominado modo diferido.

La visualización de primitivas requiere, en primer lugar, modificar el estado de OpenGL para ponerlo en el modo de visualización requerido, en función del modo actual (que puede ser, como mínimo, el modo *puntos*, el modo *líneas* y el modo *sólido*). El modo *ajedrez* consiste en dibujar la mitad de las caras de un color, y la otra mitad de otro. Es posible que haya que realizar una función de dibujado específica para este modo.

Restricciones

Tan sólo está permitido utilizar la primitiva GL_TRIANGLES, y los distintos modos de visualización de los modelos se consiguen mediante la sentencia `glPolygonMode`.

La escena debe tener activo el flag `GL_CULL_FACE`, de forma que no se pinten las caras traseras.

Además de fijar el modo actual, es necesario configurar adecuadamente diversos parámetros de OpenGL. Esto se debe a que, aunque en algunos casos los valores por defecto iniciales de esos parámetros son adecuados para la prácticas, es posible que el código de visualización previo al de la malla indexada (p.ej. la visualización de los ejes), deje OpenGL en un estado incorrecto o no esperado.

Modos de envío de primitivas

En las librerías gráficas hay dos formas o modos de envíar las primitivas a la GPU para su visualización:

- **Modo inmediato**: cada vez que se quieren visualizar se envía a la GPU las coordenadas de los vértices, los atributos, y los índices de la tabla de triángulos. Esta opción, por tanto, es lenta.
- **Modo diferido**: antes de visualizar por primera vez, se envía a la GPU todos los datos de las primitivas. Cuando se quiere visualizar, se referencian los datos ya almacenados en la GPU. Esta opción, por tanto, es mucho más rápida si la geometría no cambia, lo cual es el caso de las mallas indexadas de estas prácticas.

En esta sección veremos como enviar en modo inmediato y en modo diferido (en ambos casos con `glDrawElements`) una malla indexada. Suponemos que las tablas usan los tipos para tuplas que ya hemos visto antes.

Modo inmediato (glDrawElements)

Asumiendo un objeto tipo malla indexada con tablas de tuplas como las descritas antes, la visualización de la mallas en modo inmediato se puede hacer usando `glDrawElements` como se indica aquí abajo.

```
// habilitar uso de un array de vértices
glEnableClientState( GL_VERTEX_ARRAY );
```

```
// indicar el formato y la dirección de memoria del array de vértices
// (son tuplas de 3 valores float, sin espacio entre ellas)
glVertexPointer( 3, GL_FLOAT, 0, v.data() ) ;

// visualizar, indicando: tipo de primitiva, número de índices,
// tipo de los índices, y dirección de la tabla de índices
glDrawElements( GL_TRIANGLES, f.size()*3,
GL_UNSIGNED_INT, f.data() );

// deshabilitar array de vértices
glDisableClientState( GL_VERTEX_ARRAY );
```

Para hacer la visualización usamos los métodos `data` y `size` de los vectores de la librería estándard de C++, que sirven, respectivamente, para obtener un puntero al primer elemento del vector, y para saber cuantos elementos tiene.

1.2.3. Modo diferido (glDrawElements)

En el modo diferido (mucho más eficiente), es necesario, una única vez, enviar las tablas de vértices y triángulos a la memoria de la tarjeta gráfica (la GPU), y después, cada vez que se quiera visualizar, se debe de referenciar esas tablas, de forma que no son transferidas de nuevo (la GPU las lee directamente de su memoria). Este es el modo usado normalemente en OpenGL, ya que evita transferencias innecesarias de datos.

El término VBO (*Vertex Buffer Object*) referencia a un bloque de memoria contigua en la GPU, y que se usa para almacenar una o varias tablas. Un VBO puede contener o bien una o varias tablas de atributos de vértices (coordenadas de posición, colores, coordenadas de textura, normales, etc....), o bien una tabla de índices de vértices. No se pueden mezclar en un VBO tablas de índices y de atributos.

Cada VBO tiene asociado un valor entero único (que llamamos *identificador del VBO*), mayor estricto que cero, y que se usa para referenciar un VBO. Cada tabla dentro de un VBO se identifica por dos valores: el identificador del VBO, y el *offset* o desplazamiento en bytes del primer byte de la tabla, respecto del primer byte del VBO. Los identificadores VBOs son de tipo `GLuint` (equivalente a `unsigned`).

En estas prácticas vamos a usar un VBO distinto para cada tabla de datos (vértices y triángulos). Por tanto, será necesario almacenar en cada objeto de tipo malla indexada los dos identificadores de VBOs, como variables de instancia. Como cada VBO tiene una sola tabla, el *offset* de las tablas será 0. Los identificadores de VBO se deben incializar a 0, de esta forma, la primera vez que se invoque al método para visualizar en modo diferido, se detectará que los VBOs no están creados (ya que sus identificadores son nulos) y será necesario crearlos, antes de la primera visualización. A partir de esa creación, los identificadores tendrán un valor distinto de 0. Si una malla no se llega a visualizar en modo diferido por cualquier motivo, los VBOs no se crean.

Para facilitar la creación de los VBOs, podemos añadir una función (`CrearVBO`) en el archivo de código fuente de las mallas3D. Esta función recibe como parámetros: el tipo de VBOs (un valor que indica si el VBO contiene tablas de atributos de vértice o una tabla de

índices), el tamaño en bytes del VBO, y el puntero a la memoria RAM desde donde leer los datos para transferirlos. Como resultado, produce un identificador de VBO.

El tipo de VBO se identifica por un valor `GLuint` que solo puede valer `GL_ARRAY_BUFFER` (para VBOs con tablas de atributos de vértices) o bien `GL_ELEMENT_ARRAY_BUFFER` (para VBOs con una tabla de triángulos, es decir, de índices). El tamaño en bytes se puede calcular usando el método `size` de las tablas y la función `sizeof` aplicada a `float` o `int`. Para el puntero usamos el método `data` de las tablas.

```
GLuint CrearVBO( GLuint tipo_vbo, GLuint tamanio_bytes,
GLvoid * puntero_ram )
{
GLuint id_vbo ;                          // resultado: identificador de VBO
glGenBuffers( 1, & id_vbo );             // crear nuevo VBO, obtener identificador (nunca 0)
glBindBuffer( tipo_vbo, id_vbo );  // activar el VBO usando su identificador

// esta instrucción hace la transferencia de datos desde RAM hacia GPU
glBufferData( tipo_vbo, tamanio_bytes, puntero_ram, GL_STATIC_DRAW );

glBindBuffer( tipo_vbo, 0 );       // desactivación del VBO (activar 0)
return id_vbo ;                          // devolver el identificador resultado
}
```

En el método `draw_modoDiferido` será necesario verificar si los identificadores de VBOs son 0, y, en ese caso, invocar a |CrearVBO| una vez por cada tabla.

A continuación se puede visualizar la malla, usando para ello `glDrawElements`, de una forma parecida a como se hace en el modo inmediato, excepto que en lugar de punteros a memoria RAM, usamos offsets dentro los VBOs. Estos offsets son siempre 0 pues únicamente alojamos una tabla por VBO, la cual, por tanto, siempre comienza al inicio de dicho VBO. Además, es necesario activar los VBOs antes de la llamada a `glVertexPointer` (para el VBO de vértices) o a `glDrawElements` (para el VBO de índices).

Si suponemos que el identificador del VBO de vértices es `id_vbo_ver`, y el de índices es `id_vbo_tri`, el código quedaría así:

```
// especificar localización y formato de la tabla de vértices, habilitar tabla

glBindBuffer( GL_ARRAY_BUFFER, id_vbo_ver );  // activar VBO de vértices
glVertexPointer( 3, GL_FLOAT, 0, 0 );         // especifica formato y offset (=0)
glBindBuffer( GL_ARRAY_BUFFER, 0 );           // desactivar VBO de vértices.
glEnableClientState( GL_VERTEX_ARRAY );       // habilitar tabla de vértices

// visualizar triángulos con glDrawElements (puntero a tabla == 0)

glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, id_vbo_tri ); // activar VBO de triángulos
glDrawElements( GL_TRIANGLES, 3*f.size(), GL_UNSIGNED_INT, 0 ) ;
glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, 0 );          // desactivar VBO de triángulos

// desactivar uso de array de vértices
```

```
glDisableClientState( GL_VERTEX_ARRAY );
```

1.2.4. Interacción con el sistema

El alumno deberá programar un sistema de interacción con el programa basado inicialmente en la interacción con el teclado. Para ello se seguirá la siguiente sistemática en la práctica (en el proyecto se puede seguir algo similar si se desea):

- Tecla 'O': Activa el modo de *selección de objeto*, tras lo cual, si se pulsa:
  - Tecla 'C': Se visualiza/oculta el Cubo
  - Tecla 'T': Se visualiza/oculta el Tetraedro
  - Tecla 'Q': Se sale del modo *selección de objeto*.
- 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
  - Tecla 'L': Se activa/desactiva la visualización en modo líneas
  - Tecla 'S': Se activa/desactiva la visualización en modo sólido (por defecto)
  - Tecla 'A': Se activa/desactiva la visualización en modo ajedrez
  - Tecla 'Q': Se sale del modo *selección de modo de visualización*.
- Tecla 'D': activa el modo de *selección de modo de dibujado*, en cuyo modo, si se pulsa:
  - Tecla '1': Se activa la visualización usando glDrawElements
  - Tecla '2': Se activa la visualización usando Vertex Buffer Objects (VBOs)
  - Tecla 'Q': Se sale del modo *selección de modo de dibujado*.

Toda esta interacción forma parte de `Escena`, no del callback de glut, por lo que no hay que tocar para ello el código de `practica.cc`. Debe salir por terminal mensajes que informen de la tecla pulsada y el estado en que se encuentra cada uno de los modos, o el mensaje de error (p.ej. opción no válida) en caso de ser necesario.

1.3. Temporización recomendada

Se recomienda no dedicar más de una semana a estas tareas.

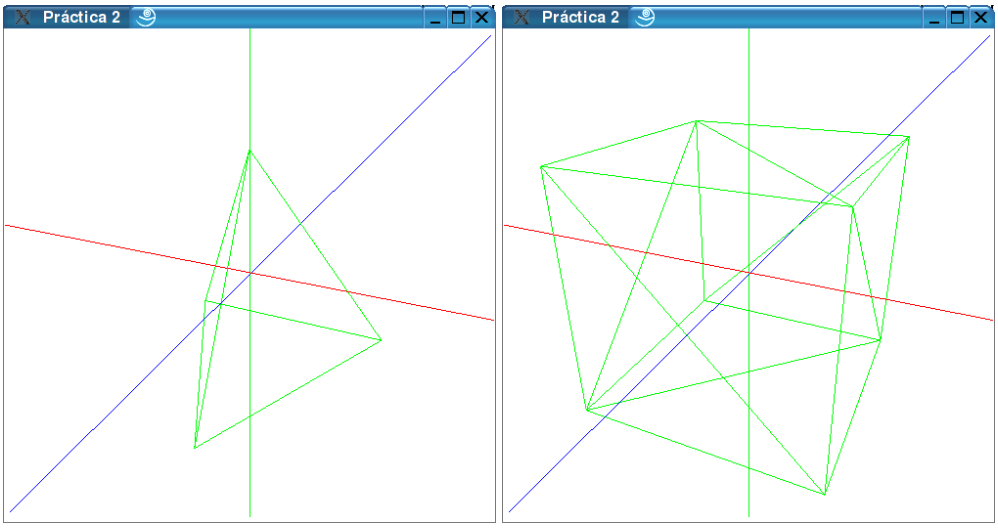
Criterios	Puntuación máxima
Clase Malla3D (implementación correcta)	0.03
Clase Cubo (Creación correcta de Geometría y Topología)	0.03
Clase Tetraedro (Creación correcta de Geometría y Topología)	0.03
Uso de VBO para dibujar	0.08
Uso de drawElements para dibujar	0.03
Uso de ColorArray para el color	0.03
Visualización modo puntos	0.04
Visualización modo alambre	0.04
Visualización modo sólido	0.04
Visualización modo ajedrez	0.05
CULL_FACE habilitado	0.05
Extra: Visualización simultánea puntos + lineas + sólido (cualquier combinación de ellos)	0.05

1.4. Evaluación

En la evaluación del proyecto final, los contenidos a evaluar serán:

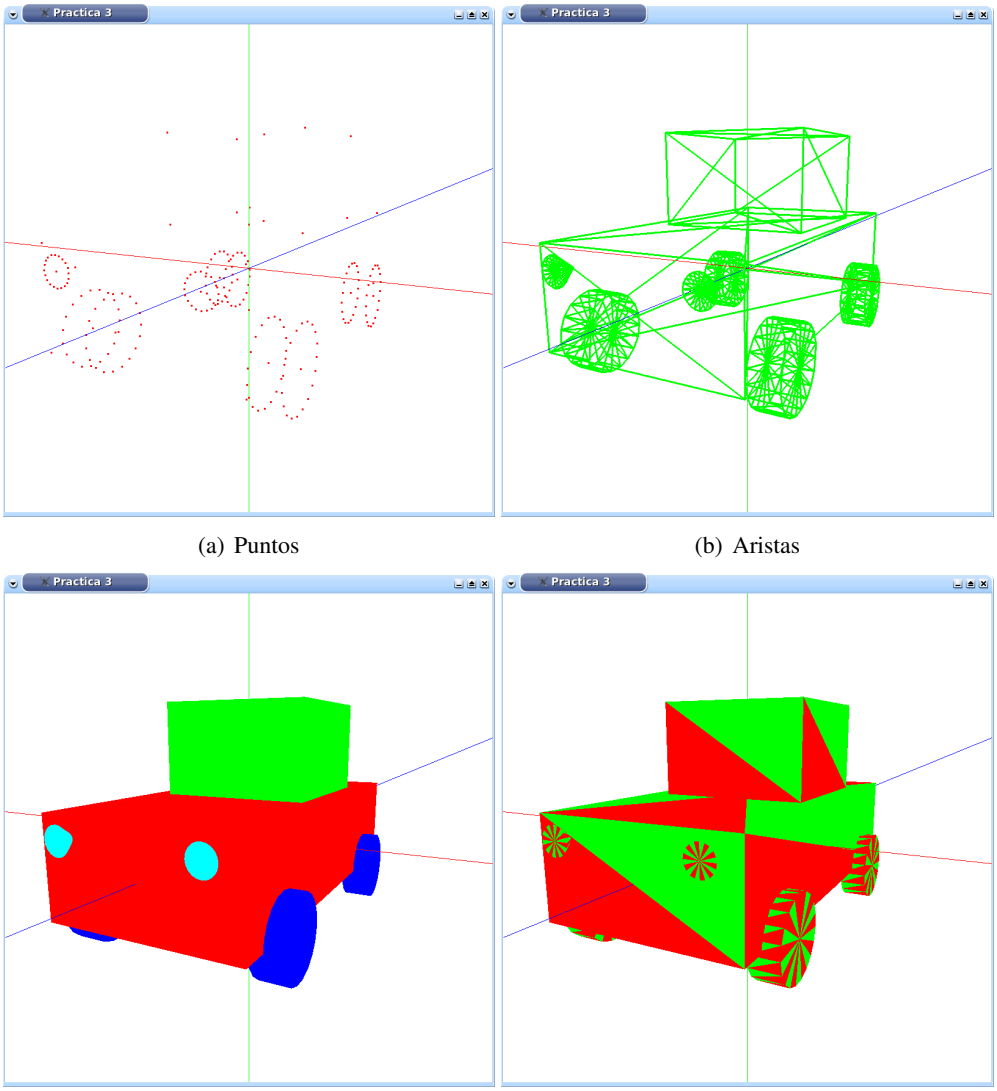
El apartado de evaluación extra (10 % del total) consiste en que se pueda ver simultáneamente el objeto en diferentes modos de visualización.

Cada semana (a partir de octubre) se habilitará una entrega en Prado para que el alumno que desee ser evaluado suba a dicha plataforma el código de su proyecto que considere que está listo para ser valorado. Si en algún momento posterior del cuatrimestre, algo evaluado



(a) Tetraedro (b) Cubo

Figura 1.2: Tetraedro y cubo visualizados en modo alambre.DMP©



(a) Puntos (b) Aristas

(c) Sólido (d) Ajedrez

Figura 1.3: Coche mostrado con los distintos modos de visualización.DMP©

positivamente deja de funcionar, se eliminará esa nota.

1.5. 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
- M. E. Mortenson; *Geometric Modeling*; John Wiley & Sons,1985