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

Prácticas de Informática Gráfica

Autores:
Pedro Cano
Antonio López
Domingo Martín
Juan carlos Torres
Carlos Ureña

Curso 2013/14

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, en la cual cada práctica se basará en la realizada anteriormente, planteamos partir desde la primera práctica, que servirá para tomar un contacto inicial, y terminar generando un sistema de partículas con animación y detección de colisiones.

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.

Índice general

Práctica 1

Introducción. Visualización de modelos PLY

1.1. Objetivos

Con esta práctica se quiere que el alumno aprenda:

- A utilizar las primitivas de dibujo de OpenGL
- A distinguir la diferencia entre definir un modelo en código y definirlo mediante datos
- A crear estructuras de datos apropiadas para su uso en visualización de modelos gráficos
- A leer modelos guardados en ficheros externos y su visualización, en concreto en formato PLY

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. Para facilitar su uso, la aplicación permite abrir una ventana, mostrar unos ejes y mover una cámara básica. Así mismo, se incluye el código básico para dibujar los vértices de un cubo unidad. Se entrega también el código de un lector básicos de ficheros PLY compuestos únicamente por triángulos.

El alumno estudiará el código. En particular debe fijarse en el código que permite dibujar los 8 vértices. Se debe comprobar la rigidez que impone el definir el modelo mediante código (¿qué pasa si en vez de un cubo es un dodecaedro?). Se debe entender que la solución pasa por independizar la visualización de la definición de modelo. Reflexionar sobre esto pues es un concepto muy importante.

El alumno debe encontrar una solución que permita una mayor flexibilidad. Una vez entendido e implementado, se hará uso del código que permite leer un fichero PLY, visua-

lizándolo. Es probable que se vea la necesidad de crear estructuras de datos que faciliten el manejo de los modelos y sus datos. Por ejemplo, en vez de tener 3 flotantes para definir las coordenadas de un vértice, utilizar una estructura que los convierta en una entidad (ver el código vertex.h como ejemplo).

Finalmente se creará el código que permita visualizar el modelo con los siguiente modos:

- Alambre
- Sólido
- Ajedrez

Para poder visualizar en modo alambre (también para el modo sólido y ajedréz) lo que se hace es mandar a OpenGL como primitiva los triángulos, `GL_TRIANGLES`, y cambiar la forma en la que se visualiza el mismo mediante la instrucción `glPolygonMode`, permitiendo el dibujar los vértices, las aristas o la parte sólida.

Para el modo ajedrez basta con dibujar en modo sólido pero cambiando alternativamente el color de relleno.

1.3. Evaluación

Para evaluación de la práctica se tendrán en cuentan los siguientes items que deberán cumplirse (sobre 10):

- Creación del código que permite visualizar un vector de flotantes en modo puntos (2pt)

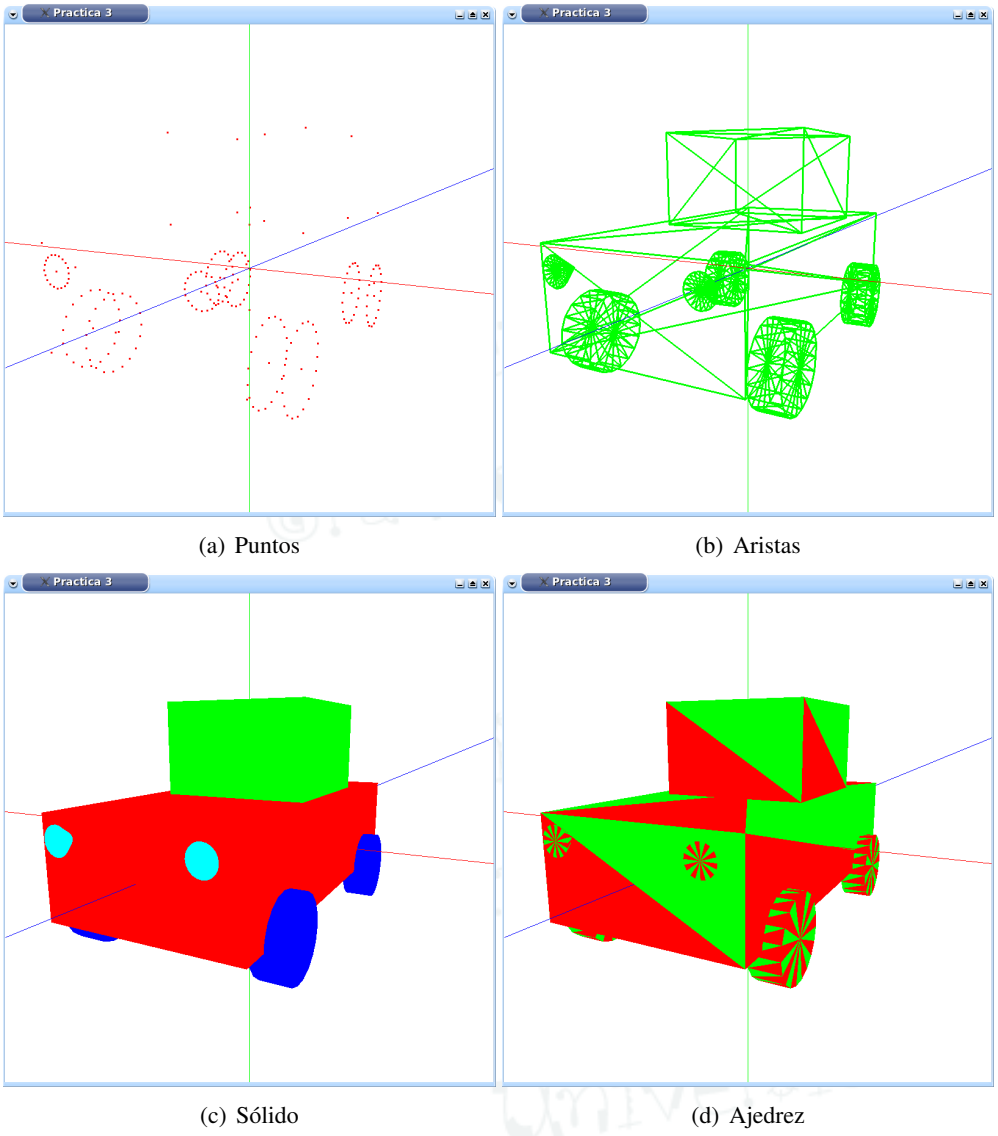
- Lectura de un fichero PLY y visualización en modo puntos (4 pt.)
  Se leerar el fichero PLY y dados los vectores de coordenadas de los vértices y el vector de los indices de vértices, se creará una estructura de datos que guarde los vértices y las caras (se recomienda usar STL y el fichero auxiliar vertex.h).

- Creación del código que permite visualizar en los modos alambre, sólido y ajedrez. (4pt)
  Creación de los procedimientos para visualizar en los distintos modos.

1.4. Extensiones

1.5. Duración

La práctica se desarrollará en 2 sesiones



(a) Puntos (b) Aristas

(c) Sólido (d) Ajedrez

Figura 1.1: Coche mostrado con los distintos modos de visualización.

1.6. 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

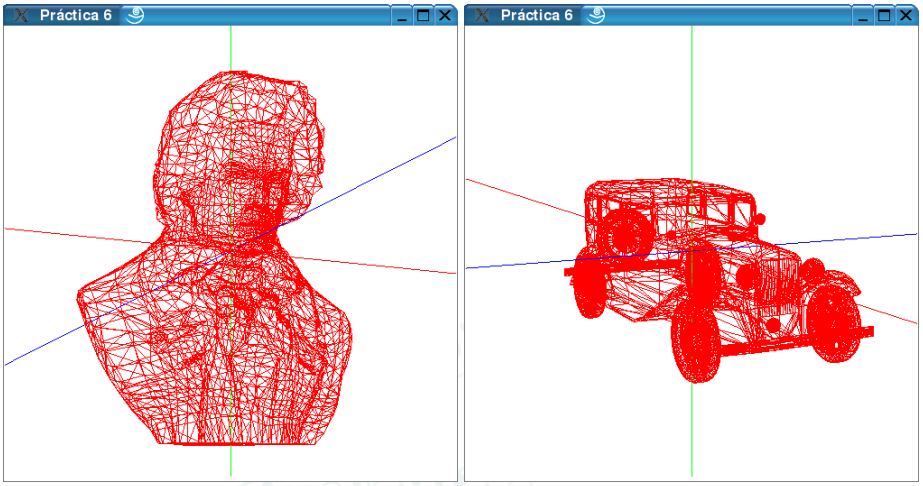


Figura 1.2: Objetos PLY.

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

Práctica 2

Modelos Poligonales

2.1. Objetivos

Aprender a:

- Modelar objetos sólidos poligonales mediante técnicas sencillas. En este caso se usará la técnica de modelado por revolución de un perfil alrededor de un eje de rotación
- Realizar cálculos geométricos sobre las caras y vértices de un modelo poligonal

2.2. Desarrollo

En primer lugar, se ha de crear un código que a partir del conjunto de puntos que representen un perfil respecto a un plano principal ($X = 0$, $Y = 0$, $Z = 0$), de un parámetro que indique el número de lados longitudinales del objeto a generar por revolución y de un eje de rotación, calcule el conjunto de vértices y el conjunto el caras que representan el sólido obtenido.

Veamos los pasos para crear un sólido por revolución.

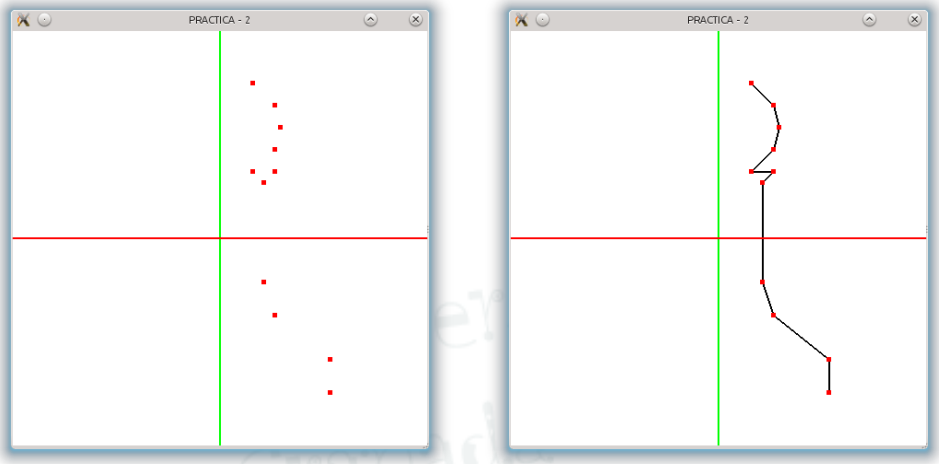
- Sea, por ejemplo, un perfil inicial Q_1 en el plano $Z = 0$ definido como:

$$Q_1\left(p_1(x_1, y_1, 0), ..., p_M(x_M, y_M, 0)\right),$$

  siendo $p_i(x_i, y_i, 0)$ con $i = 1, ...M$ los puntos que definen el perfil (ver figura 2.1).

- Se toma como eje de rotación el eje Y y si N es número lados longitudinales, se obtienen los puntos o vértices del sólido poligonal a construir multiplicando Q_1 por N sucesivas transformaciones de rotación con respecto al eje Y, a las que notamos por $R_Y(\alpha_j)$ siendo α_j los valores de N ángulos de rotación equiespaciados. Se obtiene un conjunto de $N \times M$ vértices agrupados en N perfiles Q_j, siendo:

$$Q_j = Q_1 R_Y(\alpha_j), \quad \text{con } j = 1, \ldots, N$$



(a) Puntos (b) Polilínea

Figura 2.1: Perfil inicial.

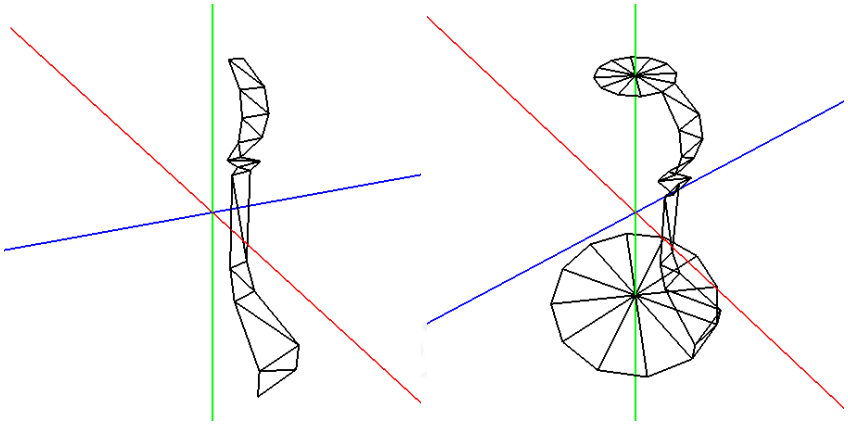
- Se guardan $N \times M$ los vértices obtenidos en un vector de vértices según la estructura de datos creada en la práctica anterior.
- Las caras longitudinales del sólido (triángulos) se crean a partir de los vértices de dos perfiles consecutivos Q_j y Q_{j+1}. Tomando dos puntos adyacentes en cada uno de los dos perfiles Q_j y Q_{j+1} y estando dichos puntos a la misma altura, se pueden crear dos triángulos. En la figura 2.2(a) se muestran los triángulos así obtenidos solamente para un lado longitudinal para una mejor visualización. Los vértices de los triángulos tienen que estar ordenados en el sentido contrario a las agujas del reloj.
- A continuación creamos las tapas del sólido tanto inferior como superior (ver figura 2.2(b)). Para ello se han de añadir dos puntos al vector de vértices que se obtienen por la proyección sobre el eje de rotación del primer y último punto del perfil inicial. Estos dos vértices serán compartidos por todas las caras de las tapas superior e inferior.
- Todas las caras, tanto las longitudinales como las tapas superior e inferior, se almacenan en la estructura de datos creada para las caras en la práctica anterior.

El modelo poligonal finalmente obtenido se podrá visualizar usando cualquiera de los distintos modos de visualización implementados para la primera práctica (ver figura 2.3).

Dos consideraciones sobre la implementación del código: primera, se puede hacer un tratamiento diferenciado cuando uno o ambos puntos extremos del perfil inicial están situados sobre el eje de rotación y segunda, el perfil inicial se puede leer de un fichero PLY cuyo contenido sólo ha de tener las coordenadas de los puntos de éste (no es difícil crear manualmente un perfil con un fichero PLY, véase el siguiente ejemplo).

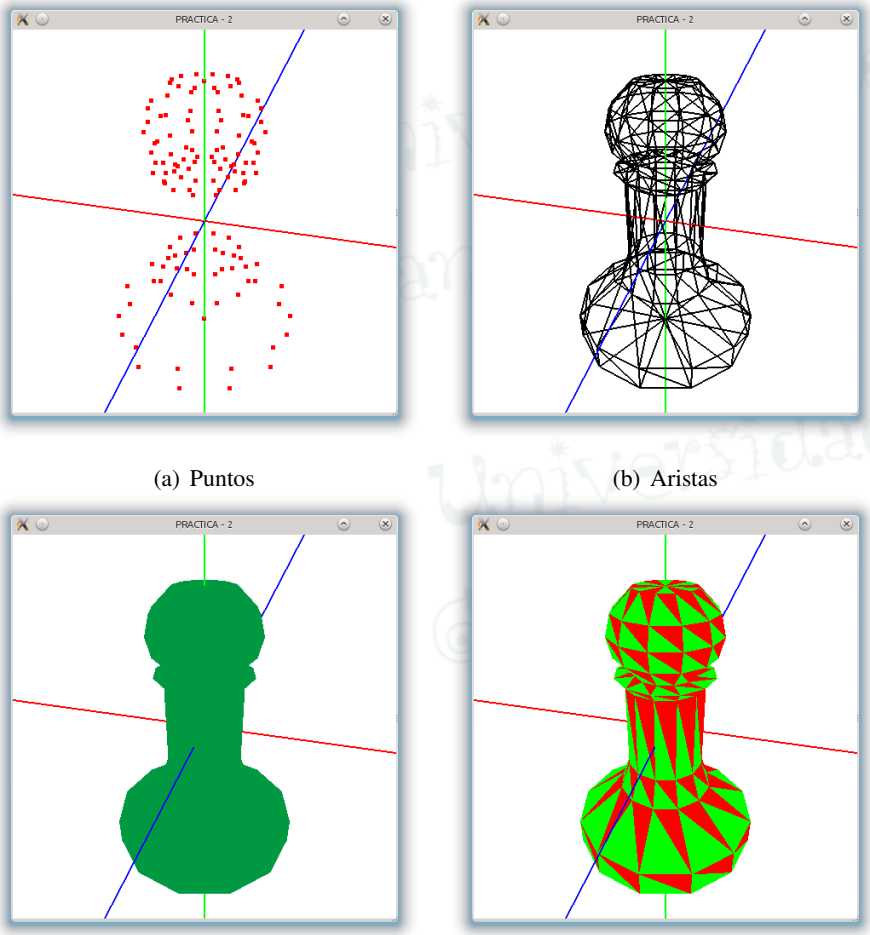
```
ply
format ascii 1.0

©
```



(a) (b)

Figura 2.2: Caras del sólido a construir: (a) logitudinales (solo un lado es mostrado) y (b) incluyendo las tapas superior e inferior.



(a) Puntos (b) Aristas

(c) Sólido (d) Ajedrez

Figura 2.3: Sólido generado por revolución con distintos modos de visualización.

```
element vertex 11
property float32 x
property float32 y
property float32 z
end_header
1.0 -1.4 0.0
1.0 -1.1 0.0
0.5 -0.7 0.0
0.4 -0.4 0.0
0.4 0.5 0.0
0.5 0.6 0.0
0.3 0.6 0.0
0.5 0.8 0.0
0.55 1.0 0.0
0.5 1.2 0.0
0.3 1.4 0.0
```

En segundo lugar, como cálculo geométrico sobre el modelo, se ha de realizar código para obtener los vectores normales a las caras y los vectores normales a los vértices. Estos vectores normales se utilizarán en la práctica dedicada a iluminación.

- A partir de la lista de vértices y de caras del objeto sólido, se calculan los vectores normales de todas las caras. Dada una cara triangular constituida por vértices A, B y C ordenados en el sentido contrario a las agujas del reloj, si se definen dos vectores $\vec{AB}$ y $\vec{BC}$ un vector normal al plano que contiene a la cara $\vec{N}$ se obtiene mediante la expresión $\vec{N} = \vec{AB} \otimes \vec{BC}$, donde $\otimes$ representa el producto vectorial de dos vectores. Un vector normal a una cara no tiene porque ser un vector normalizado o unitario. Así, los vectores normales $\vec{N}$, que han sido calculados previamente, tienen que ser normalizados.

  Se ha de crear una estructura de datos para almacenar los vectores normales a las caras.

- Los vectores normales a los vértices se calculan a partir de la lista de vectores normales a las caras que se obtiene en el paso anterior. El vector normal a un vértice v, que denotamos por $\vec{N}_v$ se obtiene como el promedio normalizado de los vectores normales de las caras que comparten dicho vértice:

$$\vec{N}_v = \frac{\sum_{i=1}^{n} \vec{N}_i}{|\sum_{i=1}^{n} \vec{N}_i|} \tag{2.1}$$

  donde n es el número de caras que se reúnen en el vértice v y $|\cdot|$ representa el módulo de un vector.

  Se ha de crear también, una estructura de datos para almacenar los vectores normales a los vértices.

2.3. Evaluación

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

- Creación del código para el modelado de objetos por revolución (6 pts.).
- Creación del código para el cálculo de los vectores normales (4 pts.).

2.4. Extensiones

Se propone como extensión modelar sólidos por barrido a partir de un contorno cerrado.

2.5. Duración

La práctica se desarrollará en 3 sesiones

2.6. Bibliografía

- Mark Segal y Kurt Akeley; *The OpenGL Graphics System: A Specification (version 4.1)*; http://www.opengl.org/
- P. Shirley y S. Marschner; *Fundamentals of Computer Graphics, 3rd Edition*; A K Peters Ltd. 2009.
- J. Vince; *Mathematics for Computer Graphics*; Springer 2006.

Práctica 3

Modelos jerárquicos

3.1. Objetivos

Con esta práctica el alumno aprenderá a:

- Diseñar modelos jerárquicos de objetos articulados.
- Controlar los parámetros de animación de los grados de libertad de modelos jerárquicos usando OpenGL.
- Gestionar y usar la pila de transformaciones de OpenGL.

3.2. Desarrollo

Para realizar un modelo jerárquico es importante seguir un proceso sistemático, tal y como se ha estudiado en teoría, poniendo especial interés en la definición correcta de los grados de libertad que presente el modelo.

Para modificar los parámetros asociados a los grados de libertad del modelo utilizaremos el teclado. Para ello tendremos que escribir código para modificar los parámetros como respuesta a la pulsación de teclas.

Las acciones a realizar en esta práctica son:

1. Diseñar un modelo jerárquico con al menos 3 grados de libertad distintos (al menos deben aparecer giros y desplazamientos). Puedes tomar como ejemplo el diseño de una grúa semejante a las del ejemplo (ver figura 3.1). En el ejemplo, estas gruas tienen al menos tres grados de libertad: ángulo de giro de la torre, giro del brazo y altura del gancho.

2. Diseñar el grafo del modelo jerárquico del objeto diseñado, determinando el tamaño de las piezas y las transformaciones geométricas a aplicar (tendrás que entregar el grafo del modelo en papel, o en formato electrónico: pdf, jpeg, etc, cuando entregues la práctica).

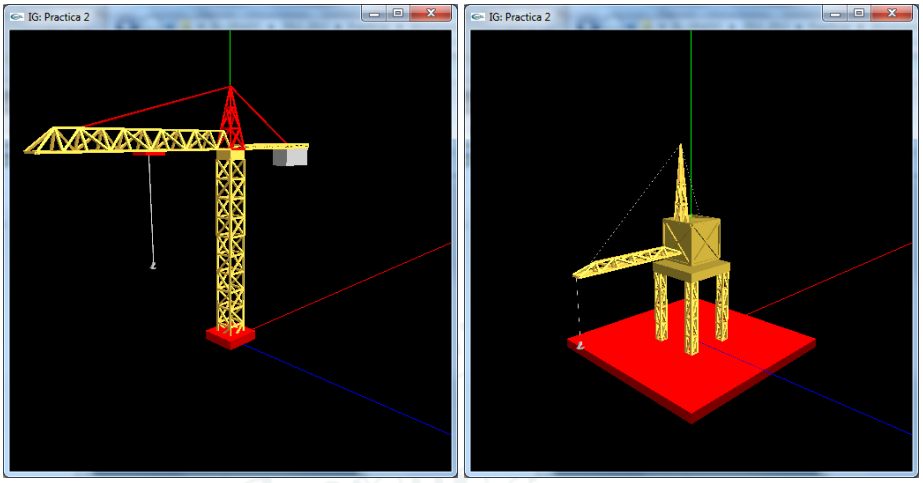


Figura 3.1: Ejemplos del resultado de la práctica 3.

3. Crear las estructuras de datos necesarias para almacenar el modelo jerárquico. El modelo debe contener la información necesaria para definir los parámetros asociados a la construcción del modelo (medidas de elementos, posicionamiento, etc.) y los parámetros que se vayan a modificar en tiempo de ejecución (grados de libertad, etc.). Hay que tener en cuenta que un modelo jerárquico esta formado por otros objetos, normalmente más sencillos, entre los que existen relaciones de dependencia hijo-padre, por lo que se deberá poder almacenar en la estructura de datos los distintos componentes que lo forman , asi como las transformaciones que le afectan a cada uno con su tipo de transformación y sus parámetros.

   Si el modelo no almacena explícitamente el grafo jerárquico, sino que dicha jerarquía se contruye en base objetos ya creados y a la gestión de la pila de transformaciones de OpenGL, se deberá crear el código que permite representar el modelo jerárquico en base a los parámetros de construcción y grados de libertad que se definan en el modelo.

4. Inicializar el modelo jerárquico diseñado para almacenar en la estructura de datos los componentes y parámetros necesarios para su construcción.

5. Crear el codigo necesario para visualizar el modelo jerárquico utilizando los valores de los parámetros del modelo almacenado en la estructura de datos. El alumno podra utilizar las funciones de visualización implementadas en las anteriores practicas que permiten visualizar los modelos con las distintas técnicas implementadas.

6. Incorporar código para cambiar los parámetros modificables del modelo y para controlar su movimiento. Añadir la ejecución de dicho código en las funciones de control de pulsación de teclas para modificar el modelo de forma interactiva. Hay que tener presente los límites de cada movimiento.

7. Ejecutar el programa y comprobar que los movimientos son correctos.

3.2.1. Reutilización de elementos

En esta práctica para construir los modelos jerárquicos se deben utilizar otros elementos más sencillos que al combinarse mediante instanciación utilizando las transformaciones geométricas necesarias, nos permitirán construir modelos mucho más complejos. Se puede partir de cualquier primitiva que nos ofrezcan las propias librerías de OpenGL, o reutilizar los elementos implementados en las prácticas anteriores.

3.2.2. Resultados entregables

El alumno entregará un programa que represente y dibuje un modelo jerárquico con al menos tres grados de libertad, cuyos parámetros se podrán modificar por teclado. Para esta práctica se deberá incorporar el control con las siguientes teclas para activar/desactivar cada acción posible de las realizadas en las 3 primeras prácticas:

- Tecla p: Visualizar en modo puntos
- Tecla l: Visualizar en modo líneas/aristas
- Tecla s: Visualizar en modo sólido
- Tecla a: Visualizar en modo ajedrez
- Tecla 1: Activar objeto PLY cargado
- Tecla 2: Activar objeto por revolución
- Tecla 3: Activar objeto jerárquico
- Tecla Z/z: modifica primer grado de libertad del modelo jerárquico (aumenta/disminuye)
- Tecla X/x: modifica segundo grado de libertad del modelo jerárquico (aumenta/disminuye)
- Tecla C/c: modifica tercer grado de libertad del modelo jerárquico (aumenta/disminuye)

3.3. Evaluación

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

- Diseño del modelo jerárquico y del grafo correspondiente que muestre las relaciones entre los elementos (2 puntos)
- Creación de las estructuras necesarias para almacenar el modelado jerárquico (4 puntos)
- Creación del código para poder visualizar de forma correcta el modelo (1 punto)

- Control interactivo del cambio de los valores que definen los grados de libertad del modelo (3 puntos)

3.4. Extensiones

Como extensión optativa se propone la incorporación de animación automática de los componentes del modelo. Para ello:

- Añade al modelo lo que estimes necesario para almacenar una velocidad de movimiento para cada uno de los parámetros.
- Añade opciones en el control de teclas para fijar la velocidad de cada parámetro a un valor positivo, negativo o cero.
- Ahora puedes animar el modelo haciendo que el valor del parámetro que modifica cada grado de libertad se modifique segun su velocidad.

Para animar el modelo utiliza una función de fondo que se ejecute de forma indefinida en el ciclo de ejecución de la aplicación OpenGL, en la que puedes realizar la actualización de cada parámetro en función de su velocidad. La función de fondo estará creada en el fichero principal y se activa desde el programa principal con:

```
glutIdleFunc( idle );
```

siendo ”void idle()” el nombre de la función que se llama para modificar los parámetros. Esta función de animación debe cambiar los parámetros del modelo en función de la velocidad y para que se redibuje, llamar a:

```
glutPostRedisplay();
```

Cambia finalmente el procedimiento de entrada para que desde el teclado se pueda cambiar también las velocidades de modificación de los grados de libertad:

- Tecla B/b: incrementa/decrementa la velocidad de modificación del primer grado de libertad del modelo jerárquico
- Tecla N/n: incrementa/decrementa la velocidad de modificación del segundo grado de libertad del modelo jerárquico
- Tecla M/m: incrementa/decrementa la velocidad de modificación del tercer grado de libertad del modelo jerárquico

3.5. Duración

Esta tercera práctica se desarrollará en 3 sesiones.

3.6. 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.

3.7. Algunos ejemplos de modelos jerárquicos

En las figuras 3.2 y 3.3 podéis ver algunos ejemplos de modelos jerárquicos que se pueden construir para la práctica (simplificando todo lo que se quiera los distintos elementos que los componen).

Estudiar con detalle cada uno y seleccionar el que os interese, o diseñar otro que tenga al menos 3 grados de libertad similares a los de las gruas que tenéis en el ejemplo.

Figura 3.2: Ejemplos de posibles modelos jerárquicos.

Figura 3.3: Ejemplos de posibles modelos jerárquicos.

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 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 coordenadas de textura, así como crearemos estructuras de datos o variables globales con los parámetros de las fuentes de luz, los materiales y las texturas, para proceder a 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. Almacenamiento y visualización con coordenadas de textura

El primer paso será aumentar las estructuras de datos que representan las mallas en memoria y extender el código de visualización.

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.

4.2.2. 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.

4.2.3. 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, 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 encarge 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.

4.2.4. 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).

4.2.5. 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.

4.2.6. 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.

4.2.7. 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).
- 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

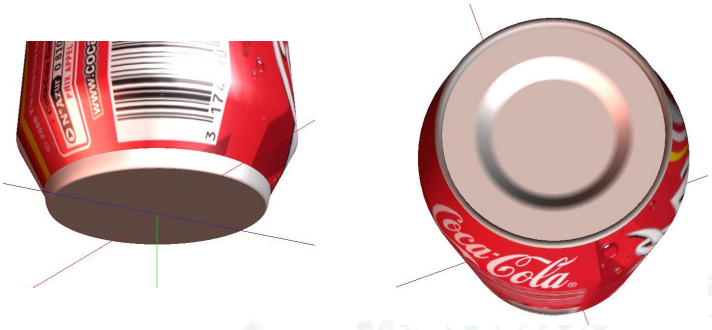


Figura 4.3: Vista ampliada de las tapas metálicas inferior (izquiera) y superior (derecha) de la lata (ambas sin textura)

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).

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.

4.2.8. 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 4.2.6.

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 4.2.6. 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

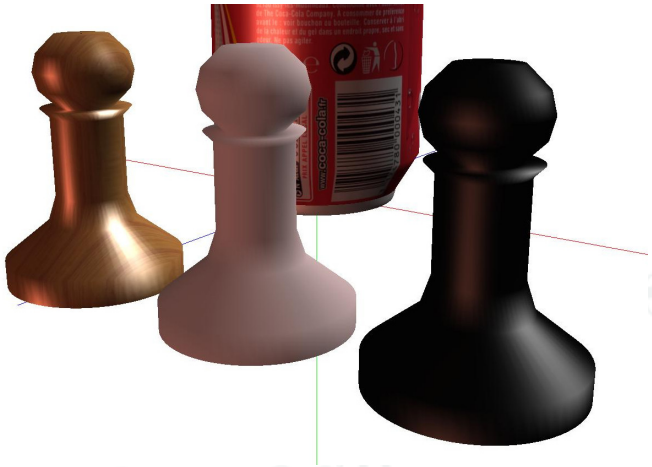


Figura 4.5: Vista ampliada de los tres objetos tipo peón.

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 4.2.3). 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:

- Incorporación de la tabla de coordenadas de textura a las mallas y su visualización (2 puntos)
- Código de asignación de coordenadas de textura al objeto de revolución (2 puntos)
- Código de carga y visualización de texturas (2 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.

Práctica 5

Interacción

5.1. Objetivos

El objetivo de esta práctica es aprender a desarrollar aplicaciones gráficas interactivas, gestionando los eventos de entrada y realizando operaciones de selección.

5.1.1. Requisitos

La práctica debe cubrirlos siguientes requisitos:

- Mover la cámara usando el ratón.
- Seleccionar componentes del modelo geométrico usando el ratón.

Los requisitos se pueden cubrir de distintas formas:

- Añadiendo código a la práctica 3 (modelos jerárquicos) para seleccionar y mover componentes del modelo.
- Anadiendo código a la práctica 2 (modelos de revolución) para introducir y editar el perfil de la poligonal.

En cualquiera de las dos opciones se pueden realizar cualesquiera de los tres requisitos.

5.1.2. Estructura de este documento

El resto del guión de esta práctica se estructura de la siguiente forma:

- La sección 5.2 describe las funciones a realizar en la práctica (control de cámara y selección).

- La sección 5.3 enumera los elementos que se evaluarán en la práctica, con la puntuación correspondiente a cada uno de ellos. Puedes observar que suman más de 10 puntos. Decide que requisitos quieres cubrir y de que práctica partirás para saber que elementos se te van a valorar.
- La sección 5.4 describe posibles extensiones de la práctica.

5.2. Funciones a implementar

Esta sección describe las funciones que se deben realizar en la práctica a nivel conceptual. Debes leerla y entenderla independientemente del código de partida que utilices para realizar la práctica. Esta documentación puedes encontrarla también la teoría de la asignatura o en la bibliografía.

5.2.1. Movimiento de cámara

Para controlar la cámara con el ratón es necesario hacer que los cambios de posición del ratón afecten a la posición de la cámara.

En primer lugar debemos de controlar los eventos de ratón. Para ello hay que activar el procesamiento de los eventos de ratón. Esto se hace pasándole a *glut* las funciones que queremos que procesen estos eventos (en el programa principal antes de la llamada a *glutMainLoop*();):

```
glutMouseFunc( clickRaton );
glutMotionFunc( RatonMovido );
```

y declarar estas funciones en el código.

La función *clickRaton* será llamada cuando se actué sobre algún botón del ratón. La función *RatonMovido* cuando se mueva el ratón manteniendo pulsado algún botón. El cambio de posición de la cámara se hará en *RatonMovido*, que solo recibe la posición del cursor, por tanto debemos almacenar el estado de los botones del ratón cada vez que se llama a *clickRaton*. Dentro de esta función debemos determinar cual es el estado de ratón, debemos comenzar a mover la cámara solo cuando se ha pulsado el botón derecho. La información de los botones se recibe de *glut* cuando se llama al callback:

```
void clickRaton( int boton, int estado, int x, int y );
```

por tanto bastará con analizar los valores de *boton* y *estado*, y almacenar información que nos permita saber si el botón derecho está pulsado y la posición en la que se encontraba el cursor cuando se pulso

```
if ( boton == GLUT_RIGHT_BUTTON )
{  if ( estado == GLUT_DOWN )
   {  estadoRaton[2] = 1 ;
      xc = x ; yc = y ;
   }
   else
      estadoRaton[2] = 1 ;
```

```
}
```

En la función *RatonMovido* comprobaremos si el botón derecho está pulsado, en su caso actualizaremos la posición de la cámara a partir del desplazamiento del cursor

```
void RatonMovido( int x, int y )
{  ......
   ......
   if ( estadoRaton[2]==1 )
   {
      getCamara( &x0, &y0 );
      yn = y0+(y-yc) ; xn = x0-(x-xc) ;
      ....
      setCamara( xn, yn );
      xc = x ; yc = y ;
      glutPostRedisplay();
  }
```

Estamos asignando directamente a la posición de la cámara, dada en coordenadas esféricas, el desplazamiento del cursor.

La modificación de la posición de la cámara se hace en el ejemplo anterior con *setCamara* y *getCamara*. La implementación de estas funciones dependerá del código de partida, en cualquier caso deben modificar u obtener los paramétros de la cámara (si se utiliza una cámara orbital las coordenadas esféricas de la misma).

Si en la práctica la transformación de visualización se hace, por ejmeplo, en

```
void change_observer()
{
   // posicion del observador
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslatef(0,0,-Observer_distance);
   glRotatef(Observer_angle_x,1,0,0);
   glRotatef(Observer_angle_y,0,1,0);
}
```

En este caso, los parámetros a modificar son *Observer_angle_x* y *Observer_angle_y*.

Las funciones *getCamara* y *setCamara* se crearán en el módulo en el que estén definidos los parámetros de la cámara y se utilizan para acceder desde el módulo de interacción a los parámetros de la cámara.

5.2.2. Selección

Para seleccionar se debe crear una función de selección (*pick*). A modo de ejemplo, si se está editando la poligonal, se puede crear la función con la siguiente interfaz:

```
int pick( int x, int y, int * elemeto, int * i )
```

siendo, x, y la posición del cursor que se va a usar para realizar la selección. Elemento un parámetro que indica si se ha seleccionado un vértice o una arista e i el índice del elemento seleccionado. La función devuelve un valor entero que indica si se ha seleccionado algo.

De esta forma la función de selección puede analizar el buffer de selección, sin tener que devolverlo completo.

Esta función se debe de llamar desde la función *clickRaton* cuando se pulse el botón izquierdo.

Para poder seleccionar los distintos componentes de la escena se deben añadir identificadores (*names*) al dibujarlos.

El procedimiento de selección (*pick*), debe realizar los siguientes pasos:

```
// 1. Declarar buffer de selección
      glSelectBuffer(...
// 2. Obtener los parámetros del viewport
      glGetIntegerv (GL_VIEWPORT, viewport)
// 3. Pasar OpenGL a modo selección
      glRenderMode (GL_SELECT)
// 4. Fijar la transformación de proyección para la seleccion
      glMatrixMode (GL_PROJECTION);
      glLoadIdentity ();
      gluPickMatrix (x,(viewport[3] - y),5.0, 5.0, viewport);
      MatrizProyeccion(); // SIN REALIZAR LoadIdentity !
// 5. Dibujar la escena
      DibujarVentana();
// 6. Pasar OpenGL a modo render
      hits = glRenderMode (GL_RENDER);
// 7. Restablecer la transformación de proyección (sin gluPickMatrix)
// 8. Analizar el contenido del buffer de selección
// 9. Devolver los valores encontrados en is,js y el resultado
```

Interpretación del buffer de selección

Cuando la selección funcione puedes extraer del buffer de selección la información necesaria en cada momento.

En la función pick añade código para buscar el primer objeto seleccionado en el buffer.

Nombres

Antes de poder realizar selecciones debes añadir identificadores a los objetos. Decide como colocar los identificadores y añade identificadores. Ten en cuenta que es lo que necesitas seleccionar.

Ahora compila y ejecuta. Procura gestionar bien la pila de nombres, asgurandote de que está vacia al comienzo del ciclo de dibujo y que no se retiran nombre cuando esá vacia.

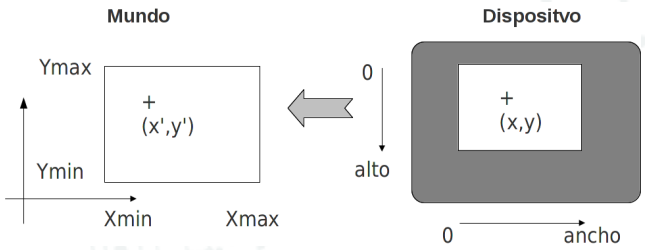
Posiciones

Cuando se desplacen vértices o se modifiquen parámetros de modelo jerárquico puedes utilizar el desplazamiento del ratón para modificar la posición o el parámetro.

Puedes simplemente utilizar el desplazamiento en coordenadas de pantalla escalado para modificar el valor de la posición o el parámetro seleccionado. En el caso de que estés

moviendo vértices del perfil, también puedes obtener la posición en coordenadas del modelo que corresponde a la posición del cursor. Para ello es necesario invertir la transformación de proyección. La forma mas simple de hacerlo es usar una proyección paralela de forma que el plano de proyección coincida con uno de los planos del sistema de coordenadas.

A modo de ejemplo, si la proyección es obre el plano XY, la inversión puede realizarse usando las siguientes ecuaciones



$$x' = X_{min} + x * (X_{max} - X_{min})/ancho \tag{5.1}$$

$$y' = Y_{max} - y * (Y_{max} - Y_{min})/alto \tag{5.2}$$

Los parámetros son los indicados a las funciones glOrtho y glViewport.

```
glOrtho( Xmin, Xmax, Ymin, Ymax, Zmin, Zmax );

glViewport( x0, y0, ancho, alto );
```

5.3. Evaluación

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

- Mover la cámara usando el ratón (4 puntos)
- Seleccionar (de componentes de un modelo jerárquico o de puntos de la poligonal) usando el ratón (4 puntos)
- Modificación interactiva de los parámnetros de la articulación o de la posición del vértice seleccionado (2 puntos)

5.4. Extensiones: posicionamiento

Puedes añadir funciones para leer posiciones 2D.

Si quieres que el posicionamiento se realice pulsando y liberando el botón del ratón, de forma que mientras mueves el ratón con el botón pulsado veas donde quedaría el punto, y este se introduzca al soltar el botón, debes usar la transformación anterior en el callback de movimiento de ratón.

Si se está modificando el modelo jerárquico esta posición se puede utilizar para añadir nuevos modelos, así si la práctica modela un robot, con esta operación se añadiría un nuevo robot en la posición indicada. Si estás añadiendo puntos al perfil del objeto de revolución esta posición será la del nuevo punto. En cualquier caso obtendrás solo dos coordenadas, a la tercera le puedes asignar un valor fijo: la que corresponde al plano en el que estas dando el punto. Por ejemplo, para el perfil de revolución la z puede ser cero.