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
Francisco J. Melero

Curso 2018/19

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

Índice General **5**

1. Introducción. Modelado y visualización de objetos 3D sencillos **7**
- 1.1. Objetivos . . . 7
- 1.2. Desarrollo . . . 7
- 1.3. Evaluación . . . 8
- 1.4. Duración . . . 9
- 1.5. Bibliografía . . . 10

2. Modelos PLY y Poligonales **11**
- 2.1. Objetivos . . . 11
- 2.2. Desarrollo . . . 11
- 2.3. Evaluación . . . 15
- 2.4. Duración . . . 15
- 2.5. Bibliografía . . . 15

3. Modelos jerárquicos **17**
- 3.1. Objetivos . . . 17
- 3.2. Desarrollo . . . 17
  - 3.2.1. Animación . . . 18
  - 3.2.2. Resultados entregables . . . 19
- 3.3. Evaluación . . . 20
- 3.4. Duración . . . 20
- 3.5. Bibliografía . . . 20

4. Iluminación y texturas **23**
- 4.1. Objetivos . . . 23
- 4.2. Desarrollo . . . 23
  - 4.2.1. Cálculo de normales. . . . 24

4.2.2. Iluminación 25
4.2.3. Texturas 29
4.2.4. Implementación 31
4.3. Evaluación 33
4.4. Duración 33
4.5. Bibliografía 33

5. Interacción **35**
5.1. Objetivos 35
5.2. Desarrollo 35
5.2.1. Selección por color 36
5.2.2. Conversión de entero a color 36
5.3. Evaluación 37
5.4. Duración 37
5.5. Bibliografía 37

Práctica 1

Introducción. Modelado 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.

1.2. Desarrollo

Para el desarrollo de esta práctica se entrega el esqueleto de una aplicación gráfica basada en eventos, mediante GLUT y Qt 5, y con la parte gráfica realizada por OpenGL. La aplicación no sólo contiene el código de inicialización de OpenGL y la captura de los eventos principales, sino que se ha implmentado una estructura de clases que permite representar objetos 3D, incluyendo unos ejes y un tetraedro. También está implementada una cámara que se mueve con las teclas de cursor, para moverse, y página adelante y página atrás para acercarse y aljarse.

El alumno deberá estudiar y comprender el código que se entrega. Hecho esto, deberá añadir la funciones que permiten dibujar en mode relleno y modo ajedres. Además deberá crear la clase cubo y utilizar una instancia que permita visualizarla cuando se apriete la tecla 2.

Por tanto, al final se dispondrá de los siguientes modos de dibujado:

- Puntos
- Líneas
- Relleno
- Ajedrez

Para poder visualizar en modo fill, sólo hay que usar como primitiva de dibujo los triángulos, `GL_TRIANGLES`, y cambiar la forma en la que se visualiza el mismo mediante la instrucción `glPolygonMode`, para que se dibuje el interior.

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

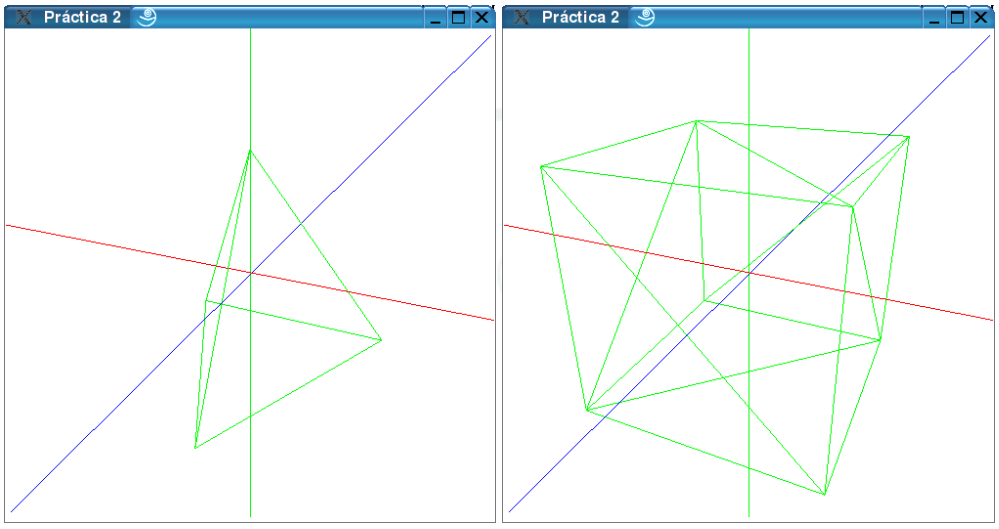
Las siguientes teclas para activar los distintos modos y objetos son las siguientes:

- Tecla p: Visualizar en modo puntos
- Tecla l: Visualizar en modo líneas/aristas
- Tecla f: Visualizar en modo relleno
- Tecla c: Visualizar en modo ajedrez
- Tecla 1: Activar tetraedo
- Tecla 2: Activar cubo

1.3. Evaluación

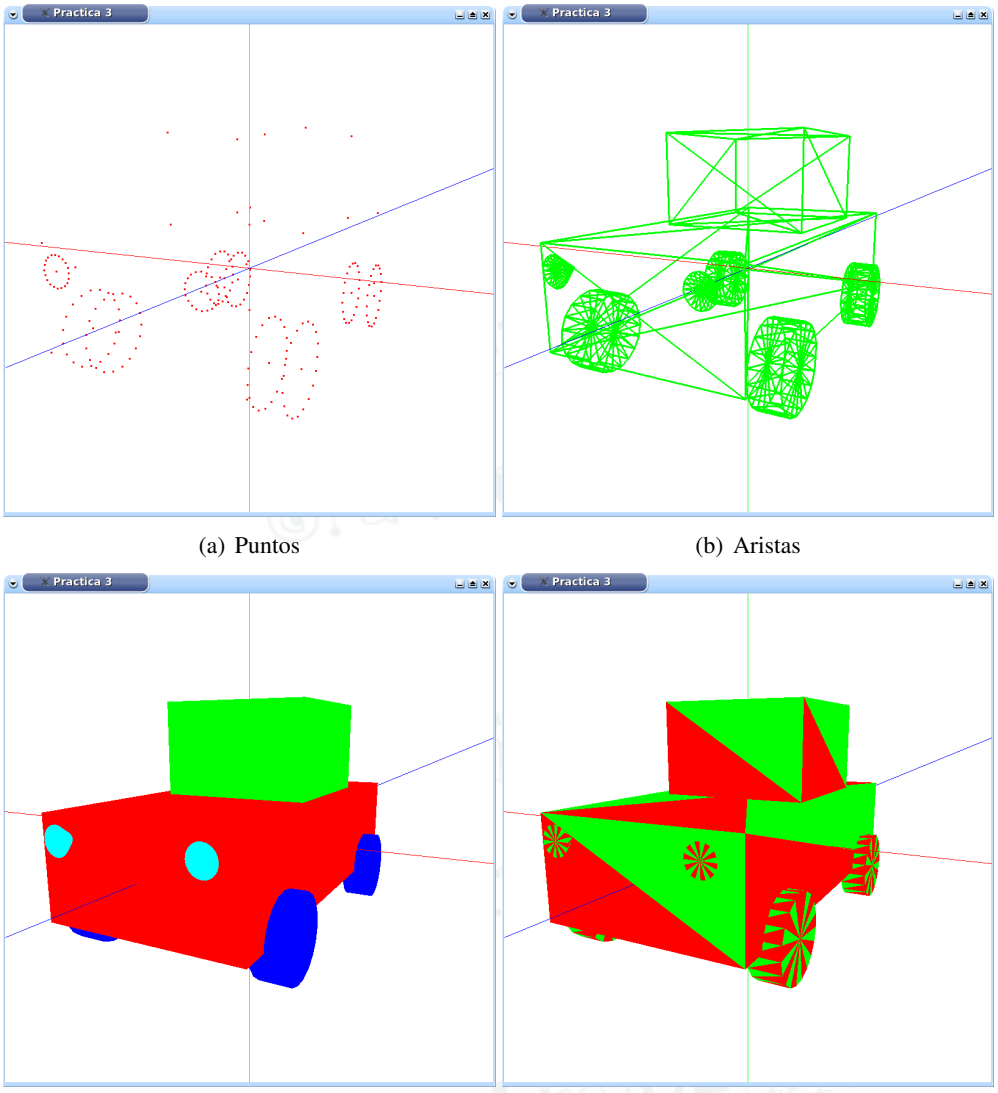
La evaluación de la práctica, sobre 10 puntos, se hará del modo siguiente:

- Creación de la clase cubo y si visualización (6 pt.)



(a) Tetraedro (b) Cubo

Figura 1.1: Tetraedro y cubo visualizados en modo alambre.



(a) Puntos (b) Aristas

(c) Sólido (d) Ajedrez

Figura 1.2: Coche mostrado con los distintos modos de visualización.

- Creación del código que permite visualizar en los modos rellenos y ajedrez. (4pt)

1.4. Duración

La práctica se desarrollará en 1 sesión

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

Práctica 2

Modelos PLY y Poligonales

2.1. Objetivos

Aprender a:

- A cargar modelos guardados en ficheros externos en formato PLY (Polygon File Format) y su visualización.
- 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.

2.2. Desarrollo

PLY es un formato para almacenar modelos gráficos mediante listas de vértices, caras poligonales y diversas propiedades (colores, normales, etc.) que fue desarrollado por Greg Turk en la universidad de Stanford durante los años 90. Para más información consultar:

`http://www.dcs.ed.ac.uk/teaching/cs4/www/graphics/Web/ply.html`

Para la realización de la práctica, en primer lugar, se visualizarán modelos de objetos guardados en formato PLY usando los modos de visualización implementados en la primera práctica. Para ello, se entregará el código de un lector básico de ficheros PLY para objetos únicamente compuestos por vértices y caras triangulares, que devuelve un vector de coordenadas de los vértices, flotantes, y un vector de los índices de vértices, enteros sin signo, que forman cada cara. A partir de los mismos se creará el objeto PLY usando las estructuras ya definidas.

En segundo lugar, dado el código que permite generar objetos por revolución de manera simple pero no óptima, habrá que modificarlo para obtener la versión que no contiene triángulos degenerados ni vértices repetidos. Para crear la versión mejorada hay que seguir el siguiente proceso:

Los pasos para crear un sólido por revolución serían:

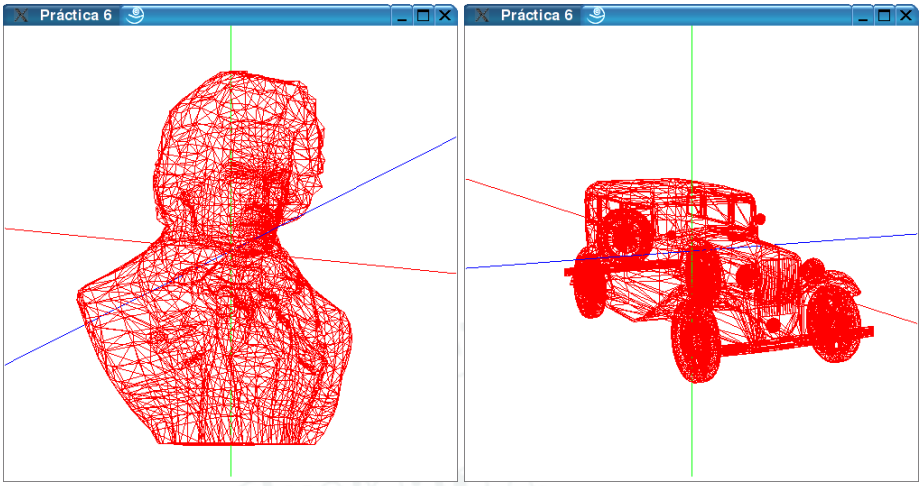
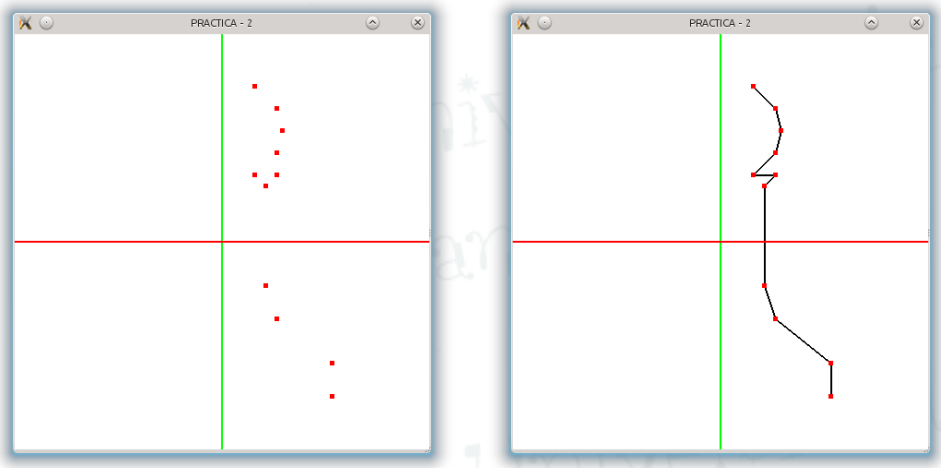


Figura 2.1: Objetos PLY.



(a) Puntos (b) Polilínea

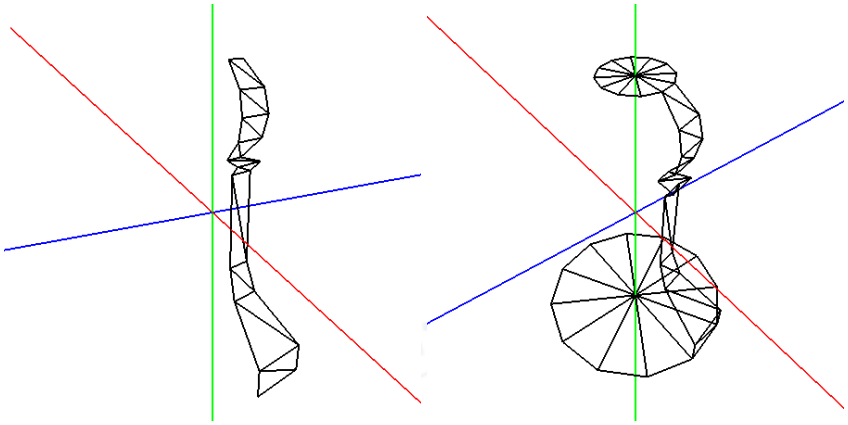
Figura 2.2: Perfil inicial.

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

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

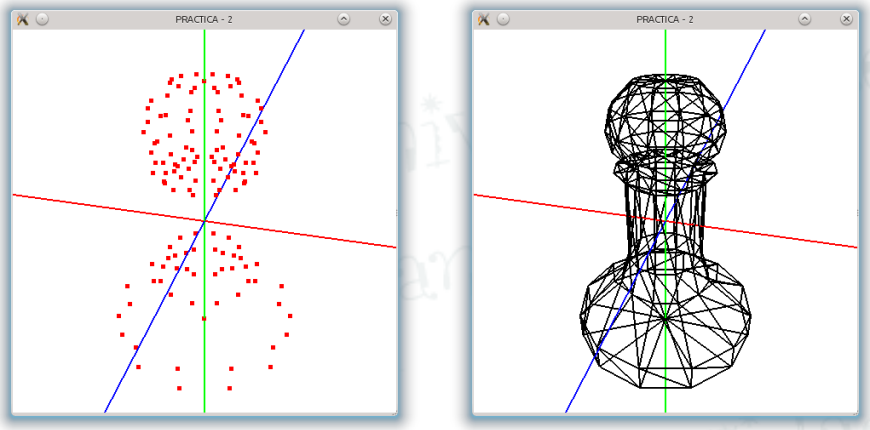
siendo $p_i(x_i, y_i, 0)$ con $i = 1, ...M$ los puntos que definen el perfil (ver figura 2.2).

- 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



(a) (b)

Figura 2.3: 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 2.4: Sólido generado por revolución con distintos modos de visualización.

conjunto de $N \times M$ vértices agrupados en N perfiles Q_j, siendo:

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

- Se guardan $N \times M$ los vértices obtenidos en un vector de vértices según la estructura de datos definida 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 2.3(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 2.3(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 también se podrá visualizar usando cualquiera de los distintos modos de visualización implementados para la primera práctica (ver figura 2.4).

Dos consideraciones sobre la implementación del código para el objeto por revolución: 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, donde hay 11 vértices y una sola cara que no se utilizará)

```
ply
format ascii 1.0
element vertex 11
property float32 x
property float32 y
property float32 z
element face 1
property list uchar uint vertex_indices
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
3 0 1 2
```

Se añadirán las siguientes teclas:

- Tecla 3: Activar cono
- Tecla 4: Activar cilindro
- Tecla 5: Activar esfera
- Tecla 6: Activar objeto PLY cargado

Es importante comprobar los cambios incluidos en el nuevo esqueleto.

2.3. Evaluación

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

- Creación de una clase para los objetos PLY (3 pts.).
- Mejora del código para el modelado de objetos por revolución de tal manera que no haya triángulos degenerados ni vértices repetidos (4 pts.).
- Creación de las clases para dibujar un cono, un cilindro y una esfera (3 pts.).

2.4. Duración

La práctica se desarrollará en 2 sesiones.

2.5. 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.
- Modificar los valores de las transformaciones automáticamente creando una animación

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 el grafo del modelo jerárquico del objeto diseñado, determinando el tamaño de las piezas y las transformaciones geométricas a aplicar. El mismo debe tener al menos 5 niveles y 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.

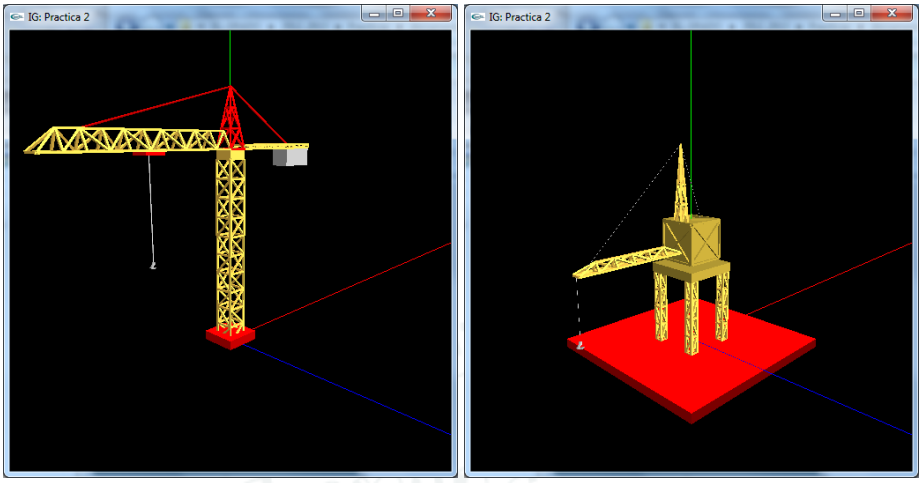


Figura 3.1: Ejemplos del resultado de la práctica 3.

2. Crear las clases necesarias para implementar el modelo jerárquico. Cada clase permitirá definir un tipo de objeto, incluyendo las transformaciones que permiten modelarlo así como las transformaciones y variables que permiten transformarlo, en su caso. La relación jeráquica, en la versión entregada, se implementa mediante la inclusión de las instancias de los objetos hijos en los objetos padres (existen otras posibilidades).

   a) Crear primero la versión estática del modelo, esto es, aquella que sólo muestra al modelo en una posición inicial pero con las dimensiones correctas.

   b) Modificar el modelo para incluir las transformaciones que permiten el movimiento. Esto implica que se deberán poder modificar los parámetros cuando se pulsen las teclas asociadas.

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

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 deben usar los elementos creados en las prácticas anteriores o nuevos que se creen para la actual.

3.2.1. Animación

Una vez se han implementado las transformaciones que aplican el movimiento, se puede modificar el modelo simplemente pulsando las teclas correspondientes. Pero también se puede conseguir que el movimiento sea automático, sin tener que estar pulsando las teclas. Para ello sólo hay que hacer que los parámetros cambien con el tiempo sin nuestra intervención. También tendremos que ajustar la velocidad a la que se producen los cambios.

Para poder automatizar el cambio de los parámetros, vamos a hacer uso del evento *idle* de GLUT (para Qt hay que usar un QTimer con un tiempo 0). Este evento, si se activa, hace

que se llame a la función que se haya indicado cuando en gestor de eventos está desocupado, idle. En general, este suele ser el estado en el que se encuentra el gestor pues los eventos, si están bien implementados, se deben resolver rápidamente. Esto significa que la función que da servicio al evento idle se estará llamando numerosas veces, y lo mejor, no interrumpirá que se produzcan otros eventos.

Para activar el evento idle hay que hacer lo siguiente:

```
glutIdleFunc(funcion_idle);
```

siendo `void función_idle()` el nombre de la función que se llama. Lo que se va hace en esta función es actualizar el valor de los parámetros que controlan las transformaciones asociadas al movimiento. Por ejemplo, el ángulo para un rotación.

Hay que tener en cuenta que lo que estamos haciendo es lo siguiente $P' = f(P)$ o $P' = f(t)$. Esto es, calculamos un nuevo valor para el parámetro P usando el valor anterior, o calculamos el nuevo valor como función del tiempo.

Por ejemplo, podemos calcular el siguiente valor del ángulo de una rotación de la siguiente manera: $\alpha' = \alpha + \delta$. δ es el incremento de ángulo que se aplica en cada paso. Por ejemplo podría ser 1° o 5°. En el primer caso el cambio se produce más lentamente que en el segundo. Por tanto, al cambiar el valor de δ estaríamos cambiando la velocidad de giro. Para las traslaciones sería similar.

Hay recordar que una vez se han actualizado las variables hay que indicar que se redibuje la escena con `glutPostRedisplay`.

Es importante comprobar los cambios incluidos en el nuevo esqueleto.

3.2.2. Resultados entregables

El alumno entregará un programa que represente y dibuje un modelo jerárquico con al menos una jerarquía de 5 niveles y con 3 grados de libertad, cuyos parámetros se podrán modificar por teclado.

Para esta práctica se deberán incorporarlas siguientes teclas:

- Tecla 7: Activar objeto jerárquico
- Tecla A: Activar/desactivar la animación
- Teclas Q/W: modifica primer grado de libertad del modelo jerárquico (aumenta/disminuye)
- Teclas S/D: modifica segundo grado de libertad del modelo jerárquico (aumenta/disminuye)
- Teclas Z/X: modifica tercer grado de libertad del modelo jerárquico (aumenta/disminuye)
- Tecla E/R: incrementar/decrementar la velocidad de modificación del primer grado de libertad del modelo jerárquico

- Tecla T/Y: incrementar/decrementar la velocidad de modificación del segundo grado de libertad del modelo jerárquico
- Tecla U/I: incrementar/decrementar la velocidad de modificación del tercer grado de libertad del modelo jerárquico

3.3. Evaluación

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

- Creación del modelo jerárquico y las clases correspondientes con al menos una jerarquía de 5 niveles (5 puntos)
- Inclusión de las transformaciones para que el modelo tenga 3 grados de libertad y se pueda mover con las teclas indicadas (3 puntos)
- Animación (2 puntos)

3.4. Duración

La práctica se desarrollará en 3 sesiones.

3.5. 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.

Apéndice

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

Iluminación y texturas

4.1. Objetivos

Los objetivos de esta práctica son conseguir un mayor realismo mediante el uso de la iluminación y las texturas. Para ello necesitamos hacer lo siguiente:

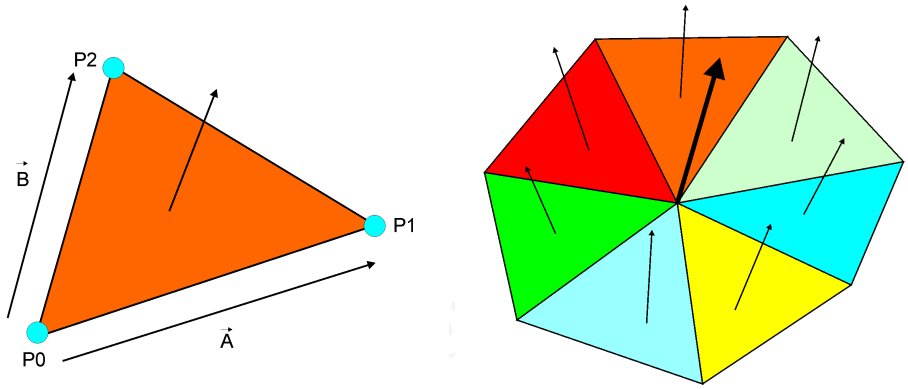
- Iluminación
  - Aprender a generar las normales de las caras y de los vértices para un modelo poligonal hecho con triángulos.
  - Añadir y usar fuentes de luz
  - Añadir y usar materiales
  - Iluminar un objeto
- Texturas
  - Calcular las coordenadas de textura para un objeto sencillo
  - Cargar una textura y visualizarla sin iluminación.

4.2. 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.

Para poder aplicar la iluminación es necesario disponer de al menos un objeto, una fuente de luz, y un modelo que nos indique cómo se refleja la luz en dicho objeto. Para calcular la reflexión no sólo es importante la posición de la luz (otros parámetros como el color son menos importantes) sino que también debemos saber la orientación de la superficie. Para ello es necesario calcular las normales.

Aunque con la iluminación 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.



(a) Normal de un triángulo (b) Normal de un vértice

Figura 4.1: Cálculo de las normales

4.2.1. Cálculo de normales.

Lo primero que vamos a hacer es el cálculo de las normales. En una superficie de la que disponemos de su descripción exacta, el problema consistiría en calcular la normal en un punto de la superficie. Nuestros modelos de vértices y triángulos en general[1] son aproximaciones de modelos curvos y continuos por lo que obtenemos también, en general, son aproximaciones.

Dado que tenemos triángulos y vértices, podremos calcular las normales de dichos elementos. Empezamos por el cálculo de las normales de los triángulos.

El cálculo de la normal de un triángulo, es muy sencillo. La normal del plano que incluye al triángulo se obtiene de la siguiente manera. Dados los puntos P_0, P_1 y P_2, podemos calcular los vectores $\vec{A} = P_1 - P_0$ y $\vec{B} = P_2 - P_0$. Si aplicamos el producto vectorial $\vec{A} \times \vec{B}$ obtenemos el vector normal, $\vec{N}$, cuyo sentido viene dado por la regla de la mano derecha (en general, se entiende que la normal apunta hacia el lado exterior del polígono) (ver figura 4.1(a)).

La normal de la cara se usa no solo para cálculos de iluminación sino que también sirve para determinar la orientación de la cara, si la misma mira hacia adentro del objeto o hacia afuera. Es importante que mire hacia afuera para que la cara pueda ser visualizada, en el caso de que se indique, mediante la instrucción glPolygonMode, que sólo se muestren las caras orientadas hacia adelante, mediante el valor GL_FRONT. Una solución consiste en visualizar las caras que miran hacia adelante y hacia atrás, mediante el valor GL_FRONT_AND_BACK, pero salvo que se quiera hacer porque se planee el introducirse en el interior del objeto, dicha opción evitará que se pueda mejorar el rendimiento, pues si así se indica, OpenGL puede eliminar de los cálculos todas aquellas caras que no son visibles en relación al observador. Para determinar si una cara es visible o no desde la posición del observador, basta con hacer el producto escalar entre la normal y el vector que se forma

[1] Si el objeto es poligonal, como por ejemplo un cubo, la representación es exacta

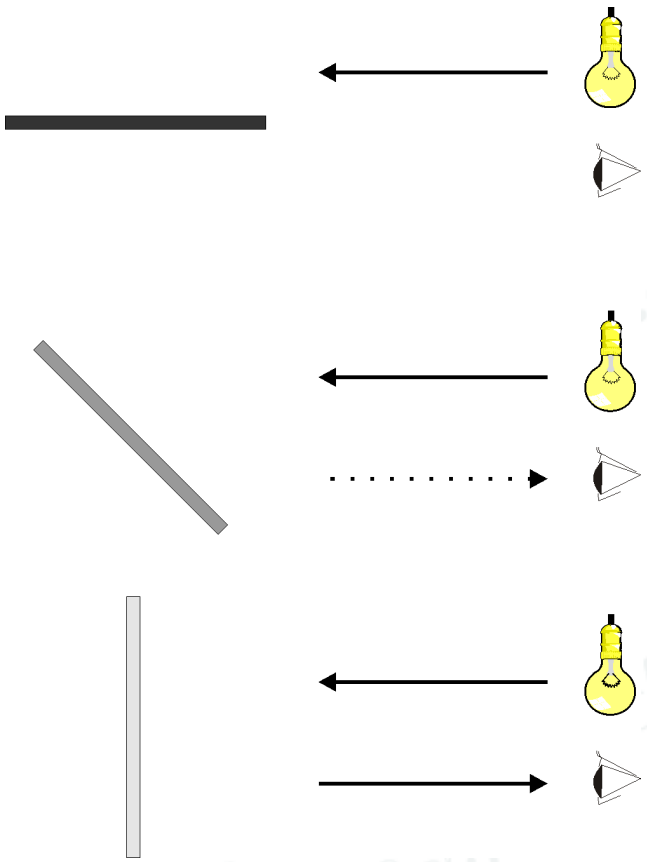


Figura 4.2: Reflexión difusa: cómo afecta la orientación

entre la posición del observador y una posición de la cara.

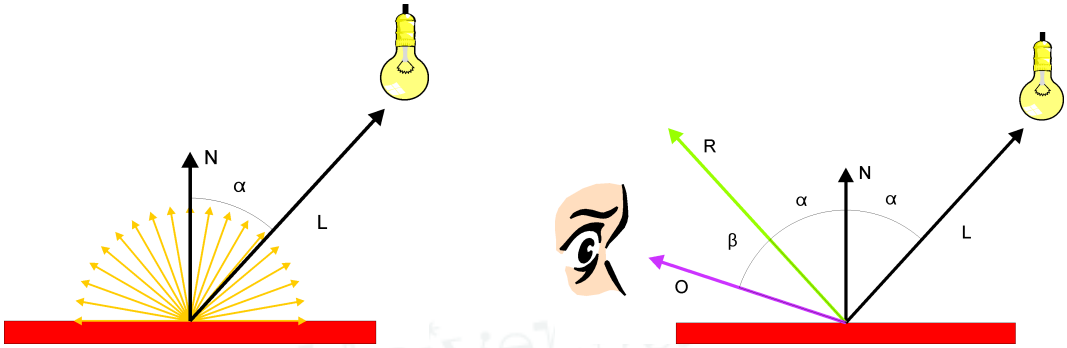
Para el cálculo de la normal de un vértice en un modelos de triángulos se parte de las normales de los triángulos que confluyen en dicho vértice y se calcula el valor medio de las normales (ver figura 4.1(b)). Esto es:

$$\vec{N} = \frac{\sum_{i=1}^{n} \vec{N}_i}{n}$$

Es muy importante que una vez que hayamos calculado las normales las normalicemos.

4.2.2. Iluminación

Una vez que disponemos de las normales vamos a ver cómo se incluyen los distinto elementos para producir la iluminación. Para alcanzar un mayor realismo mediante la iluminación vamos a implementar el suavizado plano, FLAT_SHADING, el suavizado de Gouraud, GOURAUD_SHADING.



(a) Reflexión difusa (b) Reflexión especular

Figura 4.3: Tipos de reflexión

Para poder simular la iluminación nos hace falta un modelo de reflexión. El que vamos a usar es un modelo sencillo que tiene tres componentes: ambiental, difusa y especular.

Empecemos por la componente más sencilla de entender, y en muchos casos la que mayor importancia tiene, la componente difusa. La idea básica para entender su comportamiento consiste en entender que la orientación del objeto respecto a la fuente de luz modifica la cantidad de luz reflejada: cuanto más perpendicular esté a los rayos de luz más reflejará. La figura 4.2 muestra la idea.

Dado que las superficies con las que estamos trabajando son triángulos, implica que cuanto mas perpendicular sea el plano en el que se encuentra inscrito el triángulo mayor será la cantidad de luz reflejada. ¿De qué manera queda definido el plano? ¡Mediante la normal! Si usamos la normal para definir la orientación del triángulo, la reflexión será 0 cuando sea perpendicular al vector que indica la dirección de la luz, y será máxima cuando los dos vectores sean paralelos. O sea cuando los vectores son perpendiculares el valor debe ser 0 y cuando son paralelos debe ser máximo. Podemos ver que el producto escalar tiene esa propiedad. O si tenemos en cuenta el ángulo que se forma entre ambos vectores, cuando el ángulo es 0 el valor debe ser máximo y cuando el ángulo es 90 grados, el valor debe máximo. La función coseno cumple con la propiedad, si entendemos que el valor que devuelve servirá de modulador. Así, $\cos(0) = 1$ y $\cos(90) = 0$. Para calcular el coseno vamos a usar el producto escalar: $\vec{N} \cdot \vec{L} = |\vec{N}| * |\vec{L}| * \cos(\alpha)$. Si tenemos los vectores normalizados, entonces $|\vec{N}| = 1$ y $|\vec{L}| = 1$ y por tanto, la ecuación queda así: $\vec{N} \cdot \vec{L} = \cos(\alpha)$. La figura 4.3(a) muestra el efecto de la reflexión difusa y la disposición de los vectores. Es importante hacer notar que para esta componente la posición del observador no afecta el resultado. Esto es, cuando llega un rayo de luz el objeto refleja en todas las direcciones. La mayor parte de los objetos que vemos suelen tener una componente mayoritariamente difusa.

También es importante tener en cuenta que el producto escalar puede dar valores negativos, en cuyo caso implica que la parte delantera de la cara está orientada hacia atrás. Este método también se puede usar para evitar cálculos de caras que no son visibles, es el llamado *culling* que puede ser habilitado en OpenGL.

Existen objetos que reflejan la luz de otra manera, que vamos a llamara reflexión espe-

cular. Un ejemplo son los espejos, casi un reflector especular ideal. En este tipo de objetos, un rayo de entrada es reflejado en un rayo de salida con el mismo ángulo de entrada que en la salida. La idea se muestra en la figura 4.3(b). En este caso sí que es importante la posición del observador pues dependiendo de la misma es posible que vea el rayo reflejado o no. En un reflecto especular ideal, el ángulo entre el rayo reflejado y la dirección del observador tendría que ser 0. Lo que ocurre normalmente es que los reflectores especulares no son ideales y se produce una cierta dispersión alrededor del rayo reflejado. Para modelar este comportamiento de nuevo recurrimos al coseno, pero esta vez, para el ángulo β que se forma entre $\vec{R}$ y $\vec{0}$: cuando β es 0 se obtiene la máxima reflexión y cuando es 90^o se obtiene 0. Dado que podemos encontrar objetos más o menos especulares, es necesario añadir un modificador de la función coseno: elevar el coseno a una potencia, con valores entre 0 e infinito. En las implementaciones, se usan valores finitos. Por tanto, la reflexión especular nos quedaría de la siguiente manera $\vec{R} \cdot \vec{O} = \cos^n(\beta)$, estando los vectores $\vec{R}$ y $\vec{O}$ normalizados y siendo n el exponente.

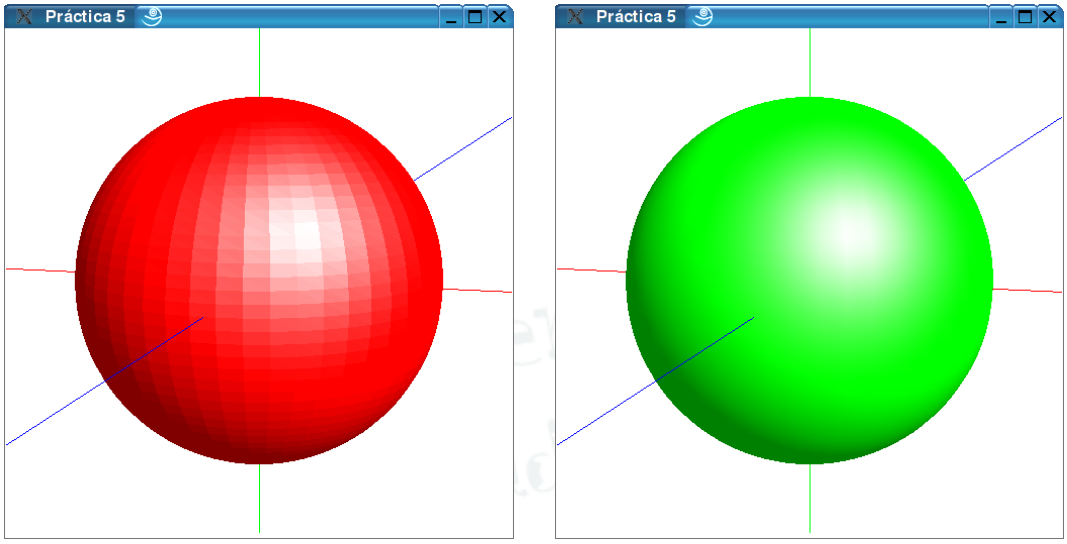
La última componente que nos falta es la ambiental. Con esta componente se pretende modelar una especie de flujo de luz constante que viene de todas direcciones. Este flujo no is imaginado sino que tiene un fundamento físico. Imaginemos una habitación pequeña sin ninguna iluminación salvo la luz que entra por un pequeño agujero. Un observador que esté en el interior podrá ver las distintas partes de la misma. El motivo es que la luz se empieza a reflejar en las paredes una y otra vez haciendo visible lo que en principio no estaba iluminado directamente. Por tanto, es una componente que forma a partir de las numerosas reflexiones de los rayos de luz en los distintos objetos. Es fácil ver que conforme los rayos se reflejan irán adquiriendo la tonalidad del material del objeto que produce la reflexión. En un modelo de iluminación global, todos estos reflejos se tienen en cuenta. En el modelo sencillo de iluminación que estamos explicando, un modelo local, esta componente se simplifica mediante el uso de un valor constante de baja intensidad. Además, evita el efecto de que las caras que no están iluminadas directamente se vean de color negro, lo cual resulta poco natural.

Recapitulemos. Nuestro modelo de reflexión de las luz es el siguiente: $I_r = A + D + E$. Esto es, la intensidad de luz reflejada por un objeto es la suma de las componentes ambiental, difusa y especular. Hemos visto que el valor de las reflexiones difusa y especular depende de la orientación del objeto con respecto a la luz en el caso de la reflexión difusa, y del observador con respecto al rayo reflejado en la reflexión especular. Ahora tenemos que incluir la intensidad de la fuente de luz y el material: si no hay luz no podemos iluminar nada, el material del objeto modula la reflexión, por ejemplo un color oscuro refleja menos luz que un color claro.

Si la intensidad de la fuente de luz es I_l, si la capacidad del material para reflejar la componente difusa es K_d, la capacidad del material para reflejar la componente especular es K_e, y la reflexión ambiental se modela con K_a, la fórmula nos queda: $I_r = I_l * K_a + I_l * K_d * cos(\alpha) + I_l * K_e * \cos^n(\beta)$

Dado que tanto la luz como los materiales se definen como colores RGB, la formula anterior se debe extender a las tres componentes.

Un detalle importante a tener en cuenta en la ecuación que hemos explicado es la posibilidad de obtener valores de intensidad reflejada mayor que 1, cuando en OpenGL, la



(a) Sombreado plano (b) Sombreado de Gouraud

Figura 4.4: Tipos de sombreado/suavizado

máxima intensidad es 1. Veamos un ejemplo. Si suponemos que el valor de la luz es $I_l = 1$, luz blanca, la componente ambiental es $K_a = 0{,}25$, la componente difusa es $K_d = 0{,}8$, y la componente especular es $K_d = 0{,}5$, un gris medio, podemos ver que como mínimo la intensidad reflejada será $I_r = 1 * 0{,}25 \rightarrow 0{,}25$, pero el valor máximo se dará cuando α y β sean 0. En tal caso el valor será:$I_r = 1 * 0{,}25 + 1 * 0{,}8 * 1 + 1 * 0{,}5 * 1 \rightarrow 1{,}55$. Este valor tiene que ser recortado a 1. Lo que veremos es que muchas partes del objeto se ven con una intensidad máxima. Por tanto, es importante modular correctamente los valores de los materiales, ya que será bastante normal usar fuentes de luz de color blanco $I_l(1,1,1)$.

Por último veamos las dos posibilidades que ofrece OpenGL para aplicar la iluminacion: sombreado plano (GL_FLAT) y sombreado de Gouraud (GL_SMOOTH). La diferencia está en que en el sombreado plano sólo se asigna una normal a los tres vértices de cada triángulo, la normal del triángulo y por tanto se obtiene un color uniforme para toda la cara (Figura 4.4(a)). En el sombreado de Gouraud se asigana a cada vértice su normal correspondiente. Por tanto tendremos tres colores en los vértices y para las posiciones intermedias OpenGL aplica una interpolación produciendo el efecto de que parece que la superficie es curva (Figura 4.4(b)).

Para poder iluminar la escena hace falta que definamos al menos una fuente de luz. Para ello usaremos la función de OpenGL glLight en sus distintas versiones. Con esta función podemos controlar cosas como la posición, el color, si es de tipo foco, etc. En nuestro caso, vamos a usarla para definir la posición y el color. Para ello se usa de la siguiente forma glLightXY(LUZ,PARAMETRO,VALOR). LUZ es la luz que queremos cambiar. Por defecto hay ocho y se nombram con GL_LIGHT0 hasta GL_LIGHT7. Como vamos a cambiar la posición usamos como PARAMETRO el valor GL_POSITION. Finalmente en valor pondríamos las dirección de un vector con las coordenadas de la posición de la luz. Es importante notar que

hay que mandar 4 coordenadas, teniendo en cuenta que la w se interpreta de la siguiente manera: si $w = 0$ entonces la luz está en el infinito; si $w \neq 0$ entonces la luz no está en el infinito.

Un detalle muy importante a tener en cuenta cuando se indica la posición de la fuente de luz es que a la misma se le aplica la transformación que haya definida en la GL_MODELVIEW. Esto permite conseguir que las luces puedan cambiar de posición mediante transformaciones.

Mientras que para la luz 0 el color esta definido como blanco, para el resto de luces es negro. Por tanto, si añadimos una segunda luz habrá que indicar su posición y también su color. Para ello se usa la misma función pero cambiando el PARAMETRO a GL_DIFFUSE, GL_SPECULAR, etc.

Recordar que hay que habilitar la iluminación y también las luces que se vayan a usar para que se pueda visualizar correctamente.

Los materiales son muy fáciles de definir mediante la función glMaterial. El formato es el siguiente: glMaterialXY(LADO,PARAMETRO,VALOR). Lado es el lado de la cara sobre el que queremos hacer la modificación. Puede ser el lado delantero, el trasero o los dos. Esto ya lo hemos visto con glPolygonMode. Los parámetros que nos van a interesar son GL_DIFFUSE, GL_SPECULAR, GL_AMBIENT y GL_SHININESS, que tienen una correspondencia directa con el modelo de iluminación que se ha explicado anteriormente. Sólo tener en cuenta que el brillo es un solo flotante mientras que los otros parámetros necesitan un color.

4.2.3. Texturas

En este ejemplo vamos a hacer uso de una de las funcionalidades que permite obtener mayor realismo: las 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 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.

Los pasos para aplicar una textura se muestran en la figura 4.5. 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.

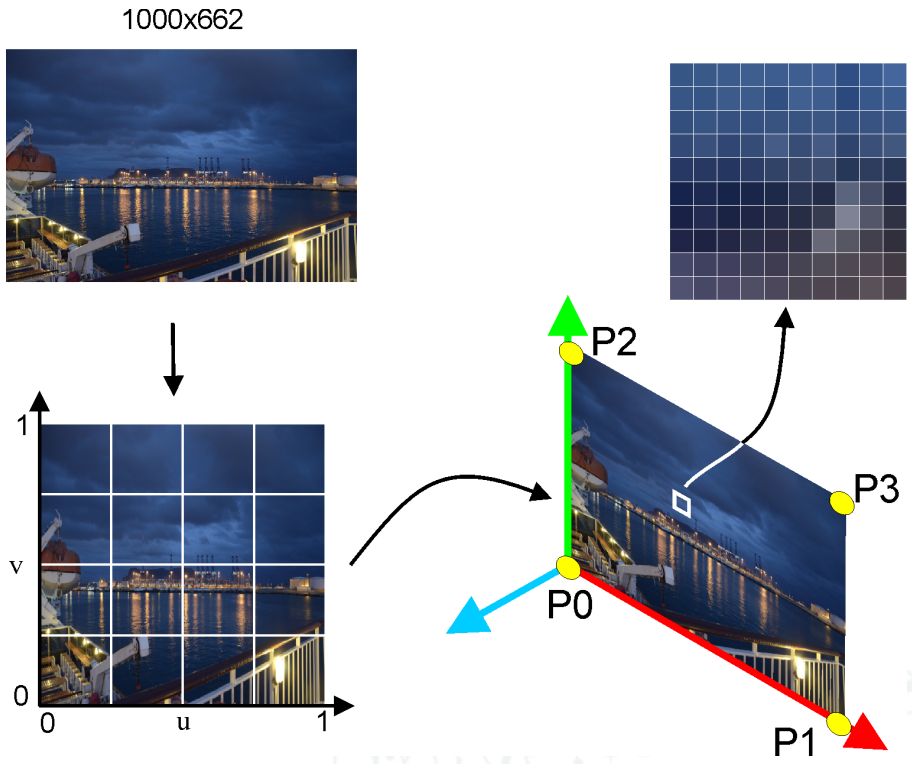


Figura 4.5: Pasos en la aplicación de una textura

El siguiente paso consiste en asignarle a cada punto del modelo las coodenadas de textura 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.

4.2.4. Implementación

Dado el código que se entrega para la práctica, el alumno realizará las siguientes tareas:

- Iluminación
  - Crear el código que permite calcular las normales de los triángulos y los vértices. Se añadirá el código en las funciones ya definidas en _object3D. Esto permitirá ver el tetraedro con la textura.
  - Crear el código que permite dibujar con sombreado plano y de Gouraud. Se añadirá el código en las funciones ya definidas en _object3D
  - Dado el código que define una luz en el infinito, se añadirá una segunda luz de color magenta para las componentes difusa y especular. Se añadirá el código en las funciones ya definidas en la función initializeGL, en donde se indica.
  - Esta segunda luz se colocará en una posición finita en la función. Se añadirá el código en set_lights
  - La segunda luz girará alrededor del objeto cuando se active la animación.
  - Se añadirán dos materiales más en la función set_materials
  - Para poder visualizar la iluminación con la esfera y dado su modo de construcción, se sobrecargarán las funciones para calcular las normales de los triángulos y los vértices.
- Texturas
  - Se añadirá el código para la clase chess_board que permite crear un cuadrado centrado con respecto al origen y situado en el plano $z = 0$. Se calcularán las coordenadas de textura para que la imagen ocupe todo el tablero.
  - Crear el código que permite dibujar la textura sin iluminación y con iluminación con sombreado plano y de Gouraud. Se añadirá el código en las funciones ya definidas en _object3D

Se hace un pequeño cambio en el control con las teclas que quedan de la siguiente manera:

- Tecla p: Visualizar en modo puntos
- Tecla l: Visualizar en modo líneas/aristas
- Tecla f: Visualizar en modo relleno

- Tecla 1: Activar tetraedo
- Tecla 2: Activar cubo
- Tecla 3: Activar cono
- Tecla 4: Activar cilindro
- Tecla 5: Activar esfera
- Tecla 6: Activar objeto PLY cargado
- Tecla 7: Activar objeto jerárquico
- Tecla 8: Activar tablero

- Tecla A: Activar/desactivar la animación

- Teclas Q/W: modifica primer grado de libertad del modelo jerárquico (aumenta/disminuye)
- Teclas S/D: modifica segundo grado de libertad del modelo jerárquico (aumenta/disminuye)
- Teclas Z/X: modifica tercer grado de libertad del modelo jerárquico (aumenta/disminuye)

- Tecla E/R: incrementar/decrementar la velocidad de modificación del primer grado de libertad del modelo jerárquico
- Tecla T/Y: incrementar/decrementar la velocidad de modificación del segundo grado de libertad del modelo jerárquico
- Tecla U/I: incrementar/decrementar la velocidad de modificación del tercer grado de libertad del modelo jerárquico

- Tecla F1: Visualización en modo sólido
- Tecla F2: Visualización en modo ajedrez
- Tecla F3: Visualización en modo iluminación sombreado plano

- Tecla F4: Visualización en modo iluminación sombreado Gouraud
- Tecla F5: Visualización en modo textura sin iluminar
- Tecla F6: Visualización en modo textura con iluminación sombreado plano
- Tecla F7: Visualización en modo textura iluminación sombreado Gouraud

- Tecla J: Activa/desactiva la primera luz
- Tecla K: Activa/desactiva la segunda luz

- Tecla M: Rota entre los tres materiales

Es importante comprobar los cambios incluidos en el nuevo esqueleto.

4.3. Evaluación

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

- Creación y funcionamiento correcto de las normales para objetos normales (2 puntos)
- Creación y funcionamiento correcto de las normales para la esfera (1 punto)
- Creación y funcionamiento correcto de las luces (1 puntos)
- Creación y funcionamiento correcto de los materiales (1 punto)
- Implementación correcta de los modos de dibujado con sombreado (1 punto)
- Implementación correcta de los modo dibujado con textura (1 punto)
- Creación del tablero (1 punto)
- Definición correcta de las coordenadas de textura para el tablero (2 puntos)

4.4. Duración

La práctica se desarrollará en 3 sesiones.

4.5. 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.

Práctica 5

Interacción

5.1. Objetivos

Con esta práctica se quiere que el alumno aprenda:

- a utilizar los eventos del ratón para mover la cámara.
- a crear una cámara con proyección paralela y realizar un zoom con la misma
- a realizar la operación de selección, *pick*, a un objeto.

5.2. Desarrollo

Dado el esqueleto, el alumno tendrá que añadir el código para que los dos grados de libertad en el movimiento del ratón se conviertan en los dos giros de la cámara que hay por defecto. El movimiento de la rueda se mapeará con el acercamiento y alejamiento de la cámara.

Mediante las teclas c y v se podrá cambiar entre una cámara con proyección de perspectiva y otra paralela. Se guardarán los valores correspondientes para que el cambio no produzca discontinuidades.

Se va a implementar un *pick* tal que seleccione uno de los triángulos del objeto, el cual se mostrará en color amarillo. Para ello se tendrá que hacer lo siguiente:

- Añadir a cada objeto una variable que indique si un triángulo está seleccionado y si es así, el número del triángulo.
- Modificar el evento de pulsación del botón derecho para que cuando se suelte se haga la selección y se muestre el triángulo seleccionado si se está en el modo draw_fill.
- Modificar el método de dibujo draw_fill para que pueda pintar el triángulo seleccionado en color amarillo.

- Definir la función draw_selection la cual se encarga de dibujar los triángulos del modelo pero haciendo que el color de cada uno dependa de su posición. Esto implica que la posición debe ser convertida a un color.
- Modificar la función *pick* para que:
  - Dependiendo del objeto que se este visualizando se llame a la función draw_selection correspondiente
  - Convierta el color leído a una posición
  - Dependiendo del objeto que se este visualizando se actualice el número de triángulo seleccionado

Se añadirán las siguientes teclas:

- Tecla C: Proyección perspectiva
- Tecla V: Proyección paralela

5.2.1. Selección por color

La idea de la selección por color es muy sencilla: cada objeto que queremos poder seleccionar tendrá asociada un identificador único. Al dibujar cada objeto, el identificador es convertido a un color, que, por tanto, también será único. El algoritmo del z-buffer se encarga de que sólo se dibujen los píxeles visibles más cercanos. Lo único que queda es, dada la posición de la selección, las coordenadas enteras x e y, leer el color del correspondiente píxel y convertirlo de nuevo a un identificador.

Normalmente se usa como identificador único la posición. Por ejemplo, si estamos identificando los triángulos de un objeto, cada uno ocupará una posición en el vector de triángulos.

5.2.2. Conversión de entero a color

Para poder realizar la conversión de un entero a un color RGB hay que tener en cuenta como se almacenan ambos. Un entero se almacena en 4 bytes, XYZW, y vamos a asumir que un color también se almacena en 4 bytes, XRGB. Los primero que podemos entender es que dado que tengo 3 bytes para definir el color, sólo los primeros 24 bits del entero pueden ser convertidos. Esto implica que 16777216 objetos pueden ser seleccionados. Realmente, hay que restar uno, siendo 16777215, ya que necesitamos un color, el blanco, para definir que no haya objeto. Esto implica que el color del fondo será blanco.

Así, por tanto, podemos establecer la siguiente relación entre posiciones del valor entero y las posiciones de los colores: Y $\rightarrow$ R, Z $\rightarrow$ G, W $\rightarrow$ B. Esto significa que los primeros valores llenarán la componente azul primero, después la componente verde y finalmente la componente roja.

Para la asignación utilizaremos nuestros conocimiento sobre el uso de mascaras a nivel de bits y las operaciones de desplazamiento. Así, para obtener la parte correspondiente a la

componente roja tendríamos que primero extraerla con una máscara y después aplicar un desplazamiento: Red= (Posicion & 0x00FF0000) >> 16. Recuérdese que & es el operador AND a nivel de bits.

Con esto tenemos la componente roja, pero con un valor entero, mientras OpenGL usa valores flotantes normalizados. Para conseguirlo, tenemos que convertir nuestra componente a flotante y dividirla por 255.0.

Este proceso se debe aplicar a las otras dos componentes pero teniendo en cuenta su posición.

Para pasar de RGB a un valor entero se debe aplicar el proceso inverso. Sólo hay que tener un detalle: si la máquina es *litte indian*, los valores menos significativos están en las posiciones más pequeñas y los valores más significativos en las más altas. Recordando, si tengo el siguiente número 0x12345678, en una máquina con arquitectural *litte indian*, todos los Intel, se guardaría en las siguientes posiciones, de menor a mayor:0x78563412. Realice los ajuste necesarios para poder hacer el cambio correctamente.

5.3. Evaluación

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

- Mover la cámara con el ratón (2 puntos)
- Implementar la cámara ortogonal y el zoom de la misma (2 puntos)
- Selección (6 puntos)

5.4. Duración

La práctica se realizará durante 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
- M. E. Mortenson; *Geometric Modeling*; John Wiley & Sons,1985
- http://www.lighthouse3d.com/opengl/picking/index.php