Práctica de síntesis de imagen con OpenGL. Tecnologías Multimedia 2006-2007

18 de diciembre de 2006

1. Introducción.

El objetivo de esta práctica es dominar los conceptos básicos de OpenGL relativos a los siguientes aspectos:

- Estructura básica de un programa OpenGL.
- Dibujo de primitivas.
- Transformaciones.
- Color.
- Iluminación.

Esta práctica, al igual que el resto de prácticas de la asignatura, debe realizarse de forma estríctamente individual de manera que pueda ser evaluada del mismo modo. En el guión se especifican los elementos mínimos para aprobar la práctica. El trabajo adicional realizado por el alumno será valorado para aumentar la nota.

Deberá entregarse junto con el ejecutable, el código fuente y un documento que como mínimo describa el árbol de transformaciones necesario para la realización del modelo.

2. Herramientas.

Este guión está orientado a la realización de la práctica sobre una plataforma Linux con las librerías Mesa3d instaladas.

Los ejemplos que acompañan a las librerías Mesa3d, englobados dentro del paquete denominado mesa-demo, pueden facilitar enórmemente la realización de esta práctica. Nos referimos concretamente a dos: robot.c y teapots.c. Veamos estos dos ejemplos.

2.1. El programa OpenGL robot.c

El código de este programa es el siguiente:

```
#include <GL/glut.h>
#include <stdlib.h>

static int shoulder = 0, elbow = 0;

void init(void)
{
  glClearColor (0.0, 0.0, 0.0, 0.0);
  glShadeModel (GL_FLAT);
}

void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT);
   glPushMatrix();
   glTranslatef (-1.0, 0.0, 0.0);
   glRotatef ((GLfloat) shoulder, 0.0, 0.0, 1.0);
   glTranslatef (1.0, 0.0, 0.0);
   glPushMatrix();
   glScalef (2.0, 0.4, 1.0);
   glutWireCube (1.0);
   glPopMatrix();

   glTranslatef (1.0, 0.0, 0.0);
   glRotatef ((GLfloat) elbow, 0.0, 0.0, 1.0);
   glTranslatef (1.0, 0.0, 0.0);
   glPushMatrix();
   glScalef (2.0, 0.4, 1.0);
   glutWireCube (1.0);
   glPopMatrix();

   glPopMatrix();
```

```
   glutSwapBuffers();
}

void reshape (int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluPerspective(65.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslatef (0.0, 0.0, -5.0);
}

/* ARGSUSED1 */
void keyboard (unsigned char key, int x, int y)
{
   switch (key) {
      case 's':
         shoulder = (shoulder + 5) % 360;
         glutPostRedisplay();
         break;
      case 'S':
         shoulder = (shoulder - 5) % 360;
         glutPostRedisplay();
         break;
      case 'e':
         elbow = (elbow + 5) % 360;
         glutPostRedisplay();
         break;
      case 'E':
         elbow = (elbow - 5) % 360;
         glutPostRedisplay();
         break;
      case 27:
         exit(0);
         break;
      default:
         break;
   }
}
```

```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow (argv[0]);
   init ();
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

Este programa realiza la representación 3D de un brazo robot compuesto por dos segmentos, estando el primer segmento conectado a una articulación denominada "shoulder" (hombro) y el segundo segmento conectado a otra articulación denominada "elbow" (codo). La rotación de los segmentos representados mediante prismas en torno a las articulaciones se realiza mediante la interacción con el usuario a través del teclado.

Cuando se pulsa una tecla que cambia el ángulo de alguna de las articulaciones se redibuja el brazo completo. Esto podría suponer un problema en una plataforma relativamente lenta si continuamente se está ordenando cambiar el ángulo. Para mejorar esto, se utiliza una ventana con doble buffer. Es decir, en un momento determinado los píxeles que se muestran en la ventana se obtendrían del buffer A, mientras que la renderización se produciría sobre el buffer B. Cuando finaliza la renderización se cambia el papel de los buffers A y B, con lo que se evitan los problemas asociados a cambios en el buffer de la ventana cuando no se ha finalizado la visualización de todos los píxeles mediante el dispositivo gráfico. La utilización del doble buffer se indica en la orden glutInitDisplayMode mediante la constante GLUT_DOUBLE.

Los siguientes comandos del programa main configuran las dimensiones y posición de la ventana (glutInitWindowSize, glutInitWindowPosition), y luego la crean (glutCreateWindow) asignándole el nombre especificado en el primer argumento con el que se ha llamado al programa.

Una vez realizada la inicialización de la ventana se pasa a la inicialización del dibujo 3D. Esta inicialización se establece en la función init, que en general agrupa la configuración del color, el modelo de iluminación, etcétera. En este

ejemplo, la inicialización se reduce a establecer el color de borrado del buffer de color a negro mediante la orden glClearColor, y a establecer el modelo de sombreado a GL_FLAT, mediante la orden glShadeModel.

Tras la inicialización se establecen las funciones callback para "display", "reshape" y "keyboard". Esta última es la que se encarga del control del teclado. Tras definir las funciones callback, se entre en el bucle principal de glut.

La función de control del teclado para la ventana actual llama ante una pulsación del teclado y tiene tres argumentos de entrada. El primer argumento es el código de la tecla pulsado, mientras que los otros dos son las coordenadas x,y del dispositivo apuntador en coordenadas relativas a la ventana actual. Como se deduce del código de la función keyboard, cada vez que se produce una pulsación reconocida como un comando, se modifica la variable shoulder o elbow para cambiar el ángulo de rotación y se llama a la función glutPostRedisplay(). Esta función establece una marca en la ventana, que indica que en la siguiente iteración del bucle principal (glutMainLoop), se debe llamar a la función display para redibujar el modelo.

La función rehape es otra función callback definida por glutReshapeFunc. Recordemos que esta función callbak se llama cada vez que la ventana cambia de tamaño y/o forma. También se llama justo antes de que se produzca la primera llamada a la función de visualización (en este caso display) después de que una ventana ha sido creado. En la función reshape se establecen las transformaciones de proyección y viewport. En este ejemplo se utiliza la función gluPerspective que permite una definición más sencilla del volumen de visualización a partir de los parámetros ángulo de apertura, relación de aspecto, distancia al plano near y distancia al plano far. En esta función reshape se introduce también una transformación inicial para la matriz "modelview", retirando el objeto hacia el lado negativo del eje z en 5 unidades.

La función display es la que realmente realizará el dibujo. Será llamada cuando explícitamente se marque la ventana actual para ser redibujada, o implícitamente cuando el sistema gestor de ventanas de cuenta de que es necesario redibujar el contenido de la ventana. Esta función comienza con una llamada a glClear para limpiar el buffer de color (previamente en init se ha definido que el color a aplicar en el limpiado de este buffer deba ser el color negro). A continuación se pasa a realizar el dibujo. Para ello se realizan un conjunto de transformaciones y se llama a la función glutWireCube. Esta función dibuja el esqueleto de un cubo. Veamos en detalle el dibujo del primer segmento del brazo.

Se está utilizando un esquema de transformaciones donde juegan un papel importante las llamadas glPushMatrix y glPopMatrix. Estas funciones permiten controlar la pila de matrices de transformación (en este caso la pila

de ModelView). La instrucción glPushMatrix toma la matriz de transformación actual y la coloca en la pila. La instrucción glPopMatrix retira la última matriz de la pila y la coloca como matriz de transformación actual. Este mecanismo sirve para realizar transformaciones de una manera jerárquica. Como se observa en el código al entrar en display se guarda la matriz de transformación actual. Esta matriz había sido creada en la función reshape y consistía en una transformación de desplazamiento. Podemos pensar en las transformaciones como un árbol. La primera transformación que se guarda en la pila sería la raíz de este árbol. Las siguientes tres transformaciones, descritas en el orden que serían aplicadas consisten en una traslación en una unidad en el sentido positivo del eje x, una rotación en torno al eje z y una traslación contraria a la primera. Si pensamos que el primer segmento será dibujado como un prisma centrado en el origen de longitud 2 en el eje x, vemos como para rotar respecto a un extremo, primero desplazamos el prisma hacia la derecha, para que su extremo quede situado en el punto origen de los ejes absolutos, luego realizamos la rotación y finalmente lo volvemos a situar en la posición original, deshaciendo la primera transformación.

Tras estas tres operaciones volvemos a situar llamar a glPushMatrix. De esta manera el conjunto de tres transformaciones descritas será un nuevo nodo del árbol que se conecta con el nodo raíz. Las siguientes dos operaciones son las correspondientes al escalado y al dibujo de un esqueleto de un cubo. La operación de escalado sirve para aplastar y alargar el cubo de forma que tenga las características geométricas deseadas. A continuación aparece una instrucción glPopMatrix(), que restablece la matriz de transformación anterior al escalado. Cada vez que aparezca una llamada a glPopMatrix() podemos entender que las operaciones realizadas desde el último glPushMatrix, configuran un nodo terminal del árbol. En un nodo terminal tendremos operaciones de dibujo. Para comprender las transformaciones realizadas sobre este dibujo, debemos desandar los nodos hasta llegar al nodo raíz. En este caso, se comienza por la operación de escalado, luego una traslación para colocar el extremo del prisma en el origen, a continuación la rotación, luego deshacemos la traslación y finalmente alejamos el modelo cinco unidades hacia el lado negativo del eje z. De esta manera queda dibujado el primer segmento.

La gran utilidad de esta técnica radica en el ensamblado de partes. Veamos ahora las transformaciones relativas al segundo de los segmentos del brazo. Las siguientes operaciones configurarían dos nuevos nodos que colgarían del nodo encargado de la primera rotación. El nodo terminal dibujaría también un prisma centrado en el origen, que constituiría el segundo segmento. El siguiente nodo trasladaría el prisma para colocar su extremo en el origen y luego pasar a hacer una rotación respecto al eje z. Finalmente la última

traslación desplazaría el prisma rotado para situar su extremo en el punto $(1, 0, 0)$, que coincide con la terminación del primer segmento.

3. El programa ejemplo teapots.c

La utilización del modelo de iluminación de opengl constituye otra parte importante de esta práctica. El siguiente ejemplo permite visualizar diferentes combinaciones de parámetros, en los que puedes basarte para construir la iluminación del hexápodo.

```
#include <stdlib.h>
#include <GL/glut.h>

void myinit(void)
{
  GLfloat ambient[] =
  {0.0, 0.0, 0.0, 1.0};
  GLfloat diffuse[] =
  {1.0, 1.0, 1.0, 1.0};
  GLfloat position[] =
  {0.0, 3.0, 3.0, 0.0};

  GLfloat lmodel_ambient[] =
  {0.2, 0.2, 0.2, 1.0};
  GLfloat local_view[] =
  {0.0};

  glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
  glLightfv(GL_LIGHT0, GL_POSITION, position);
  glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
  glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);

  glFrontFace(GL_CW);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_AUTO_NORMAL);
  glEnable(GL_NORMALIZE);
  glEnable(GL_DEPTH_TEST);
  glDepthFunc(GL_LESS);
```

```
}

void renderTeapot(GLfloat x, GLfloat y,
  GLfloat ambr, GLfloat ambg, GLfloat ambb,
  GLfloat difr, GLfloat difg, GLfloat difb,
  GLfloat specr, GLfloat specg, GLfloat specb, GLfloat shine)
{
  float mat[4];

  glPushMatrix();
  glTranslatef(x, y, 0.0);
  mat[0] = ambr;
  mat[1] = ambg;
  mat[2] = ambb;
  mat[3] = 1.0;
  glMaterialfv(GL_FRONT, GL_AMBIENT, mat);
  mat[0] = difr;
  mat[1] = difg;
  mat[2] = difb;
  glMaterialfv(GL_FRONT, GL_DIFFUSE, mat);
  mat[0] = specr;
  mat[1] = specg;
  mat[2] = specb;
  glMaterialfv(GL_FRONT, GL_SPECULAR, mat);
  glMaterialf(GL_FRONT, GL_SHININESS, shine * 128.0);
  glutSolidTeapot(1.0);
  glPopMatrix();
}

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  renderTeapot(2.0, 17.0, 0.0215, 0.1745, 0.0215,
    0.07568, 0.61424, 0.07568, 0.633, 0.727811, 0.633, 0.6);
  renderTeapot(2.0, 14.0, 0.135, 0.2225, 0.1575,
    0.54, 0.89, 0.63, 0.316228, 0.316228, 0.316228, 0.1);
  renderTeapot(2.0, 11.0, 0.05375, 0.05, 0.06625,
    0.18275, 0.17, 0.22525, 0.332741, 0.328634, 0.346435, 0.3);
  renderTeapot(2.0, 8.0, 0.25, 0.20725, 0.20725,
    1, 0.829, 0.829, 0.296648, 0.296648, 0.296648, 0.088);
  renderTeapot(2.0, 5.0, 0.1745, 0.01175, 0.01175,
```

```
  0.61424, 0.04136, 0.04136, 0.727811, 0.626959, 0.626959, 0.6);
renderTeapot(2.0, 2.0, 0.1, 0.18725, 0.1745,
  0.396, 0.74151, 0.69102, 0.297254, 0.30829, 0.306678, 0.1);
renderTeapot(6.0, 17.0, 0.329412, 0.223529, 0.027451,
  0.780392, 0.568627, 0.113725, 0.992157, 0.941176, 0.807843,
  0.21794872);
renderTeapot(6.0, 14.0, 0.2125, 0.1275, 0.054,
  0.714, 0.4284, 0.18144, 0.393548, 0.271906, 0.166721, 0.2);
renderTeapot(6.0, 11.0, 0.25, 0.25, 0.25,
  0.4, 0.4, 0.4, 0.774597, 0.774597, 0.774597, 0.6);
renderTeapot(6.0, 8.0, 0.19125, 0.0735, 0.0225,
  0.7038, 0.27048, 0.0828, 0.256777, 0.137622, 0.086014, 0.1);
renderTeapot(6.0, 5.0, 0.24725, 0.1995, 0.0745,
  0.75164, 0.60648, 0.22648, 0.628281, 0.555802, 0.366065, 0.4);
renderTeapot(6.0, 2.0, 0.19225, 0.19225, 0.19225,
  0.50754, 0.50754, 0.50754, 0.508273, 0.508273, 0.508273, 0.4);
renderTeapot(10.0, 17.0, 0.0, 0.0, 0.0, 0.01, 0.01, 0.01,
  0.50, 0.50, 0.50, .25);
renderTeapot(10.0, 14.0, 0.0, 0.1, 0.06, 0.0, 0.50980392, 0.50980392,
  0.50196078, 0.50196078, 0.50196078, .25);
renderTeapot(10.0, 11.0, 0.0, 0.0, 0.0,
  0.1, 0.35, 0.1, 0.45, 0.55, 0.45, .25);
renderTeapot(10.0, 8.0, 0.0, 0.0, 0.0, 0.5, 0.0, 0.0,
  0.7, 0.6, 0.6, .25);
renderTeapot(10.0, 5.0, 0.0, 0.0, 0.0, 0.55, 0.55, 0.55,
  0.70, 0.70, 0.70, .25);
renderTeapot(10.0, 2.0, 0.0, 0.0, 0.0, 0.5, 0.5, 0.0,
  0.60, 0.60, 0.50, .25);
renderTeapot(14.0, 17.0, 0.02, 0.02, 0.02, 0.01, 0.01, 0.01,
  0.4, 0.4, 0.4, .078125);
renderTeapot(14.0, 14.0, 0.0, 0.05, 0.05, 0.4, 0.5, 0.5,
  0.04, 0.7, 0.7, .078125);
renderTeapot(14.0, 11.0, 0.0, 0.05, 0.0, 0.4, 0.5, 0.4,
  0.04, 0.7, 0.04, .078125);
renderTeapot(14.0, 8.0, 0.05, 0.0, 0.0, 0.5, 0.4, 0.4,
  0.7, 0.04, 0.04, .078125);
renderTeapot(14.0, 5.0, 0.05, 0.05, 0.05, 0.5, 0.5, 0.5,
  0.7, 0.7, 0.7, .078125);
renderTeapot(14.0, 2.0, 0.05, 0.05, 0.0, 0.5, 0.5, 0.4,
  0.7, 0.7, 0.04, .078125);
glFlush();
```

```
}

void myReshape(int w, int h)
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if (w <= h)
    glOrtho(0.0, 16.0, 0.0, 18.0 * (GLfloat) h / (GLfloat) w,
      -10.0, 10.0);
  else
    glOrtho(0.0, 16.0 * (GLfloat) w / (GLfloat) h, 0.0, 18.0,
      -10.0, 10.0);
  glMatrixMode(GL_MODELVIEW);
}

static void key(unsigned char k, int x, int y)
{
  switch (k) {
  case 27:  /* Escape */
    exit(0);
    break;
  default:
    return;
  }
  glutPostRedisplay();
}

int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutCreateWindow(argv[0]);
  myinit();
  glutReshapeFunc(myReshape);
  glutDisplayFunc(display);
  glutKeyboardFunc(key);
  glutMainLoop();
  return 0;             /* ANSI C requires main to return int. */
}
```

En la rutina my_init se crea el modelo de iluminación. En este caso tenemos una fuente de luz (GL_LIGHT0), en la que se cambian los valores por defecto de luz ambiente, luz difusa y posición, mediante llamadas a la función glLightfv. Por otra parte, la función glLightModelfv cambia la luz ambiente general (GL_LIGHT_MODEL_AMBIENTE) y se asume que a la hora de calcular los ángulos de reflexión en las superficies el observador se sitúa en el infinito (GL_LIGHT_MODEL_LOCAL_VIEWER y valor 0.0).

La orden glFrontFace con el argumento GL_CW establece que los polígonos trazados en sentido de las agujas del reloj muestran su cara frontal. A la hora de realizar la ocultación de superficies se tiene en cuenta este hecho. Se habilita a continuación la iluminación y se enciende la fuente de luz GL_LIGHT0. A continuación se establece que al reescalar un objeto se recalculen automáticamente los vectores normales y que se realice la normalización de los vectores normales. Se habilita el test de profundidad para eliminar superficies ocultas y se establece la operación de comparación para la eliminación de superficies ocultas.

La función de renderizado obtiene por parámetros la definición del material, para la luz ambiente, la difusa y la especular. Además define el exponente especular (GL_SHININESS) (mientras más alto el valor, más pequeño y concentrado parecerá el brillo en la superficie). La construcción del objeto se realiza con glutSolidTeapot.

4. Resultados mínimos.

A continuación se describen los resultados que como mínimo se deben cubrir en esta práctica para considerarla superada.

- Se deberá representar en 3D un "hexápodo" sólido, realizado a partir de esferas y prismas. En concreto serán necesarias tres esferas, cada una de ellas representando un segmento del cuerpo. Cada segmento tendrá una pata a cada lado, que estará formada por dos prismas conectados. El prisma unido directamente a la esfera (primer segmento) se conectará a la misma por un punto articulado con dos grados de libertad que vienen dados por la rotación respecto a ejes perpendiculares. El otro prisma se conecta al anterior por un punto articulado donde sólo se permitirá un grado de libertad, dado por la rotación respecto al eje perpendicular al plano que contiene a los dos primas.

- Además de los elementos mencionados el hexápodo deberá tener alguna características adicional a elegir por el alumno (cabeza, alas, ...).

- Los ángulos asociados a los grados de libertad mencionados en el punto uno, se podrán variar mediante pulsación de teclas. Para cada grado de libertad, una tecla diferente permitirá aumentar el ángulo y otra permitirá disminuirlo.
- Deberá utilizarse al menos una fuente de iluminación. Las características del modelo de iluminación son libres, con el único requerimiento de que la superficie de las esferas que conforman el cuerpo del hexápodo deben tener un aspecto ligeramente metálico.

5. Consejos y ayudas.

- Puedes utilizar las funciones glutSolidSphere, glutWireSphere, glutSolidCube y glutWireCube para crear las esferas y los segmentos del hexápodo.
- Puedes crear una primera versión sin iluminación con elementos obtenidos a partir de glutWireSpher y glutWireSphere. Una vez funciones correctamente puedes emplear objetos sólidos.
- Planea las transformaciones en papel utilizando la técnica del árbol.
- La compilación puede ser realizada mediante la orden gcc -o file -lGL -lglut file.c.