1. INTRODUCCIÓN A OPENGL: DIBUJANDO UNA TORTUGA CON OPENGL

1.1 ¿QUÉ ES OPENGL?

Qué es OpenGL? OpenGL es una interfaz software de harware gráfico, es decir define las funciones que se pueden utilizar en una aplicación para acceder a las prestaciones de un dispositivo gráfico. Es un motor 3D cuyas rutinas están integradas en tarjetas gráficas 3D. Fue desarrollado por Sillicon Graphics, Inc. (SGI) con el afán de hacer un estándar de representación en 3D. Es compatible con prácticamente cualquier plataforma harware asi como con muchos lenguajes de programación (C, C++, Visual Basic, Visual Fortran, Java).

1.2 ABRIENDO UNA VENTANA CON OPENGL

En esta práctica se va a trabajar en un único archivo con extensión llamado *inicio_tortuga.cpp* cuyo código es el siguiente:

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

int main(int argc, char** argv) {

  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH |
  GLUT_DOUBLE); glutInitWindowSize(512, 512);
  glutInitWindowPosition(20, 20);
  glutCreateWindow("tecnunLogo");

  glutDisplayFunc(display);

  glutMainLoop()
  ; return 0;
}
```

Para poder utilizar las librerías OpenGL y GL Utility Toolkit (GLUT) es necesario incluir el archivo glut.h como se muestra en la primera línea de código.

En la función **main()**, **glutInit()** es la función que inicializa la librería GLUT y negocia con el sistema de ventanas la apertura de una nueva ventana. Sus parámetros deben ser los mismos que los de la función **main()** sin modificar. A continuación, **glutInitDisplayMode()** define el modo en el que se debe dibujar la ventana. Sus parámetros, como en muchas de las funciones OpenGL, se definen con flags o máscaras de bits. En este caso, GLUT_RGB indica el tipo de modelo de color con el que se dibujará (Red-Green-Blue), GLUT_DEPTH indica que se debe incluir un buffer de profundidad y GLUT_DOUBLE que se debe utilizar un doble buffer.

Antes de crear una ventana, es necesario definir sus propiedades. Con la función **glutInitWindowSize()** se define el tamaño de la ventana en píxeles (anchura y altura) y con la función **glutInitWindowPosition()**, la distancia horizontal y vertical con respecto de la esquina superior izquierda del monitor donde la ventana deberá aparecer. Finalmente, con la función **glutCreateWindow()** se crea propiamente la ventana, y el string que se pasa como argumento, es utilizado como nombre de la nueva ventana.

Ahora que la ventana ha sido creada, es necesario mostrarla. Para ello la función main llama a la función **glutDisplayFunc()**. Esta función es la más importante de las funciones callback. Gracias a la definición de las funciones callback, GLUT hace posible una dinámica de programación de aplicaciones OpenGL. Una función callback será llamada por GLUT para hacer alguna operación especifica cada vez que se produzca un evento. En este caso, **glutDisplayFunc**(display), define que la función **display** que es pasada como argumento sea ejecutada cada vez que GLUT determine que la ventana debe ser dibujada (la primera vez que se muestra la ventana) o redibujada (cuando se maximiza, cuando se superponen varias ventanas, etc).

La ultima función que es llamada en el main es **glutMainLoop()**. Esta función se encarga de pasar el control del flujo del programa a la GLUT, de manera que cada vez que ocurra un "evento" sean llamadas las funciones definidas como callbacks hasta que el la ventana se cierre.

La función **display()**, definida como función callback para dibujar o redubujar la ventana cada vez que sea necesario, esta también contenida en el archivo *tecnunlogo.c*. Como todas las funciones callback que serán utilizadas, **display()** es del tipo void. Como este ejercicio es bastante simple y no se va a dibujar ninguna figura en la ventana, el contenido de la función es bastante sencillo. En ella solo se van a definir las funciones que siempre deben aparecer en cualquier función display callback.

```
void display(void) {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT |
  GL_DEPTH_BUFFER_BIT); glutSwapBuffers();
}
```

La función **display**() se debe incluir antes de la función **main**() para que no se produzca un error al compilar cuando se utiliza en la función **main**().

La función **glClearColor**() establece el color de fono de la ventana, que es con el que se "borra" la ventana. A continuación se llama, antes de dibujar cualquier cosa, a la función **glClear()**. Esta función se encarga de borrar el fondo de la ventana.

Acepta como argumento el buffer específico que se desea borrar, en este caso el GL_COLOR_BUFFER_BIT y el GL_DEPTH_BUFFER_BIT.

La función **glSwapBuffers()** se encarga de intercambiar el buffer posterior con el buffer anterior y es necesaria porque se ha definido que se trabaja con doble buffer. Cuando se dibuja cualquier figura, esta es dibujada en el buffer posterior (el que está atrás) y cuando el dibujo está terminado los dos buffers se intercambian.

El resultado de ejecutar este proyecto es el que se muestra en la figura 1:

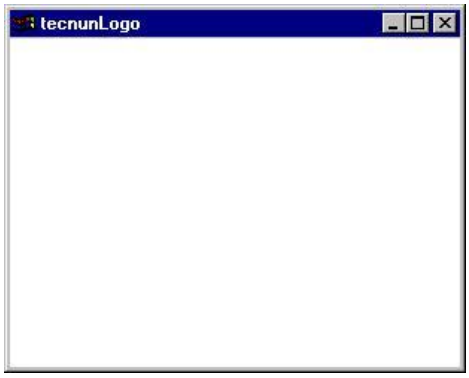

Figura 1 Ventana inicial

1.3 DIBUJANDO UN TOROIDE EN LA VENTANA

El objetivo de este ejercicio es dibujar, en la ventana del ejercicio anterior, un toroide con un cubo inscrito en su interior. Para ello será necesario incluir algunas operaciones más en la función **display()**:

```
void display(void) {
  glClearColor(1.0, 1.0, 1.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT |
  GL_DEPTH_BUFFER_BIT); glColor3f(1.0,0.0,0.0);
  glutWireTorus(0.25,0.75, 28,
  28); glColor3f(0.0,0.0,1.0) ;
  glutWireCube(.60) ;
  glutSwapBuffers();
}
```

En este ejemplo serán introducidas tres nuevas funciones GL (una función OpenGL y dos funciones GLUT). La función OpenGL **glColor3f()** establece el color actual con el que se va a dibujar una figura. El color será el mismo hasta que se cambie el 'estado' de esta variable con la función **glColor3f** nuevamente. Esto es lo que se quiere decir cuando se habla de OpenGL como una maquina de estados. Todas las funciones de OpenGL comienzan con el prefijo ª glº y en muchas (como es el caso de **glColor3f**) aparece un sufijo compuesto por un número y una letra. El número simboliza el numero de parámetros que se debe pasar a la función y la letra, el tipo de estos parámetros. En este caso, se deben pasar 3 parámetros de tipo float. Al estar trabajando en un modelo de color de tipo RGB (Red-Green-Blue), cada uno de estos parámetros representa el valor de cada color respectivamente.

La función GLUT **glutWireTorus(0.25, 0.75, 28, 28)** dibuja un toroide de frame de hilos cuyo radio interno es el double 0,25; radio externo el double 0,75; el primer entero 28 representa el numero de lados que se puede observar en cada sección radial y el segundo entero 28 el numero de divisiones radiales del toroide.

La función GLUT **glutWireCube(0.60)** dibuja un cubo cuyo tamaño queda determinado por su único parámetro de valor float.

El resultado es el que se muestra en la figura:

Dibujar los ejes del sistema de coordenadas de la ventana utlilizando los colores rojo, verde y azul (RGB) para los ejes x, y, z correspondientemente.

Dibujar en la ventana las diferentes primitivas de GLUT (se pueden encontrar en el tutorial de OpenGL en el web de la asignatura).

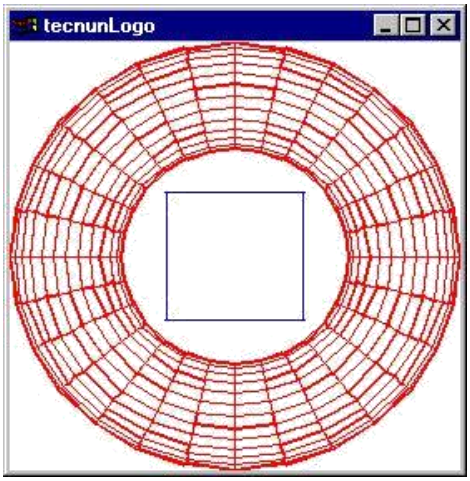

Figura 2 Ventana con toroide y cubo

1.4 DEFINIENDO EL ÁREA DE PROYECCION INICIAL

Una vez que se ha dibujado un objeto en la ventana es necesario definir el área de proyección inicial que se desea de la figura en la ventana. Para ello se debe manipular el área de proyección por medio de la función callback **glutReshapeFunc()**. Esta función callback especifica cuál función será llamada cada vez que la ventana sea redimensionada o movida, pero también es utilizada para definir inicialmente el área de proyección de la figura en la ventana.

Muchas de las funciones que OpenGL pone a disposición para definir la proyección están basadas en matrices de transformación que, aplicadas sobre el sistema de coordenadas de la ventana, definen el punto desde donde será observada la figura. Estas matrices y sus transformaciones se explicarán con más detenimiento en el siguiente capitulo.

Antes de explicar el código de este ejercicio es conveniente recordar la disposición del sistema de coordenadas de OpenGL, en el que el eje vertical es el Y y el eje de visión por defecto es el Z.

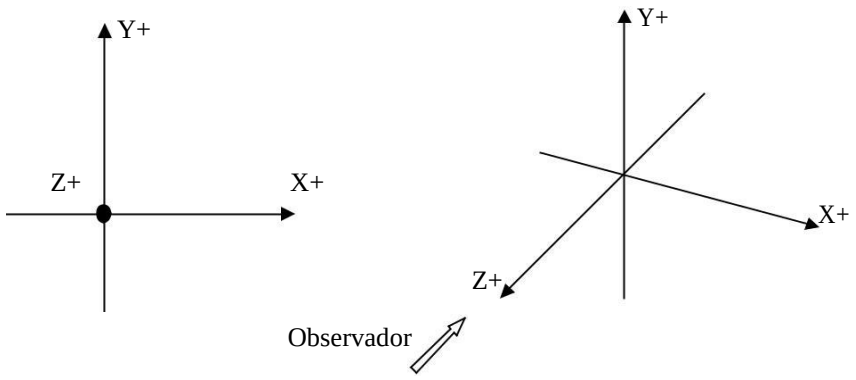

Figura 3 Sistema de Coordenadas de OpenGL

La función **glutReshapeFunc(reshape)** debe ser incluida en el código de la función **main**():

```
glutReshapeFunc(reshape);
```

A continuación se define la funcion **reshape()**:

```
void reshape(int width, int height)
  { glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0, (GLfloat)height / (GLfloat)width, 1.0,
  128.0); glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
```

```
  gluLookAt(0.0, 1.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}
```

De nuevo, como toda función callback, la función **reshape()** es del tipo void. Se le pasan como argumentos el ancho y el alto de la ventana después del reescalado. La función **glViewport** define la porción de ventana donde OpenGL podrá dibujar. Sus parámetros son: primero la distancia horizontal y vertical de la esquina superior izquierda del ª cuadroº donde OpenGL puede dibujar con respecto a la ventana; segundo, el ancho y alto de la ventana.

A continuación, **glMatrixMode()** especifica la matriz de transformación sobre la que se van a realizar las operaciones siguientes (de nuevo, recordar que OpenGL es una maquina de estados). Los tres tipos de matrices que existen son: matriz de proyección (GL_PROJECTION), matriz de modelado (GL_MODELVIEW) y matriz de textura (GL_TEXTURE).

En este caso, **glMatrixMode(GL_PROJECTION)** afecta la perspectiva de la proyección. La función **glLoadIdentity()** carga como matriz de proyección la matriz identidad. Esto es como inicializar a uno los valores de dicha matriz. **gluPerspective()** opera sobre la matriz de proyección y define el ángulo del campo de visión en sentido vertical (en grados), la relación entre la altura y la anchura de la figura (aspecto), el plano más cercano a la cámara y el plano más lejano de la cámara, respectivamente. Estos dos últimos son los planos de corte, que son los que se encargan de acotar el volumen de visualización por delante y por detrás de la figura. Todo lo que esté por delante del plano más cercano y todo lo que esté detrás del plano más lejano no será representado en la ventana.

Ahora, **glMatrixMode(**GL_MODELVIEW**)** define que las operaciones que se realicen a continuación afectarán a la matriz de modelado. Nuevamente se carga la matriz identidad por medio de la función **glLoadIdentity**. A continuación, **gluLookAt()** define la transformación sobre la vista inicial. Esta función junto con **gluPerspective**() se explican con detalle en el capítulo 3, pero aquí se hacer una rápida descripción.

La función **gluLookAt()** tiene 9 parámetros: los primeros tres representan la distancia en x, y, z de los ojos del observador; los siguientes tres, las coordenadas x, y, z del punto de referencia a observar y los últimos tres, la dirección del **upVector**.

*Modificar la función **glView port** de manera que al alargar la ventana la figura no se deforme. Se logra haciendo que el viewport sea siempre cuadrada, de dimensión el menor de los valores de la altura y la anchura. El valor de la relación entre la altura y la anchura para la función gluPerspective() es ahora siempre 1.*

*Probar diferentes vistas iniciales con la función **gluLookAt** .*

1.5 INTERACTUANDOCON EL TECLADO

El objetivo de este ejercicio es añadir la posibilidad de interactuar desde el teclado del ordenador con la figura representada en la ventana. De nuevo utilizaremos una función callback para este propósito, ya que es la GLUT, por medio de este tipo de funciones, quien gestiona cualquier tipo de ª eventoº .

Es necesario incluir en el main del programa la función callback **glutKeyboardFunc(keyboard)**:

```
  glutKeyboardFunc(keyboard)
  ;
```

La función **keyboard()** que es pasada como parámetro será llamada cada vez que ocurra un evento en el teclado. Se define a continuación la función **keyboard**():

```
void keyboard(char key, int x, int y) {

  switch (key) {
  case 'h':
    printf("help\n\n");
    printf("c            -  Toggle culling\n");
    printf("q/escape     -  Quit\n\n");
    break;
  case 'c':
    if (glIsEnabled(GL_CULL_FACE))
      glDisable(GL_CULL_FACE);
    else
      glEnable(GL_CULL_FACE)
      ;
    break;
  case '1':
    glRotatef(1.0,1.,0.,0.);
    break;
  case '2':
    glRotatef(1.0,0.,1.,0.);
    break;
  case 'q':
  case 27:
    exit(0);
    break;
  }
  glutPostRedisplay();
}
```

Un bucle switch-case se encarga de identificar que tecla ha sido pulsada. Si la tecla presionada es la 'h', se escribirán en la pantalla de la consola las indicaciones del funcionamiento del programa. Con las funciones **glEnable(**GL_CULL_FACE**)** y **glDisable(**GL_CULL_FACE**)** se muestran o se ocultan las líneas de las caras traseras de la figura. Si la tecla es la 'c' y las líneas traseras están activadas, entonces se desactivan. En caso contrario, se activan. La tecla '1' hará que la figura rote por medio de la función **glRotatef()** cuyos parámetros corresponden al ángulo a rotar y a los componentes x, y, z del eje sobre el que va a rotar la figura. Se habla en todo momento que es la figura quien va a rotar porque es la matriz de modelado la ultima que fue cargada. La tecla '2' hará que la figura rote en el eje y. Esta función y su efecto en la matriz de modelado se trata en el siguietne capítulo. Finalmente, la tecla 'q' produce la finalización del programa.

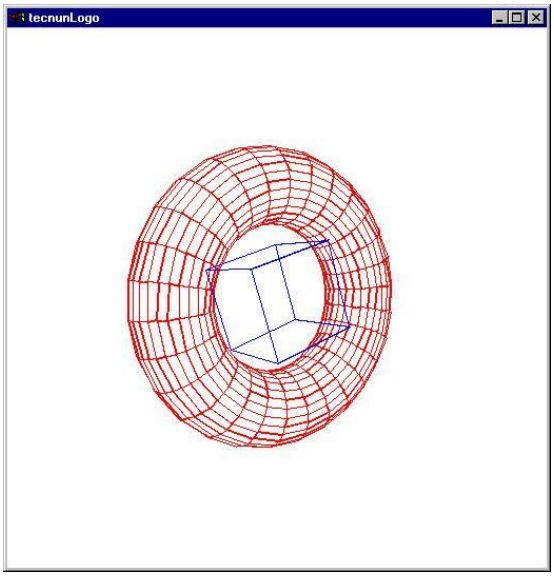

Figura 4 Ventana con respuesta al teclado

En la figura 4 se muestra el efecto de rotar la figura con las teclas 1 y 2.

Es importante observar que al final de la función **keyboard()** esta

la llamada a la función GLUT **glutPostRedisplay()**. Esta función da la indicación a la GLUT que es necesario redibujar la ventana. Si no se incluyera esta rutina, OpenGL no sabría que tiene que redibujar la ventana cuando se ha presionado una tecla.

Introducir un comando nuevo de manera que al apretar la tecla 'a' (axis), se muestren los ejes de la figura si están desactivados, o se desactiven si están activados.

Introducir otro comando de manera que con las teclas 'u', 'd', 'r' y 'l' (up, down, right, left) se tralade la cámara manipulando la función ***gluLookAt*** *.*

1.6 REPRESENTANDO UNA TORTUGA

La función **glBegin** comienza una secuencia de vértices con los que es posible dibujar polígonos. De esta manera, definiendo puntos, OpenGL da la oportunidad de construir nuevos elementos a base de líneas y polígonos. El parámetro de **glBegin** es el tipo de primitiva que se va a definir (triangle, poligon, etc.) Los vértices (puntos en un espacio 3D) se definen en OpenGL con la función **glVertex3f**() o **glVertex3d**().

Realizar la función ***draw Turtle****() que dibujar por medio de la función* ***glBegin( )*** *una tortuga. Los puntos de la tortuga se almacenan en dos vectores x[] y z[], y despues se llama a la función glVertex3d() dentro de un bucle que recorre las coordenadas de estos vectores. Al ser la tortuga simétrica sólo es necesario definir la mitad y volver a recorrer los puntos en orden inverso. La función* ***draw Turtle****() será llamada desde la función* ***display****().*

Vector de coordenadas x:

```
double x[] = {0.0, 0.1, ...};
```

Bucle for:

```
for (i=0; i < npoints; i++){
```

Comparar la utilización de

```
glBegin(GL LINE LOOP);
```

y

```
glBegin(GL POLYGON);
```

En la figura 5 se muestra el resultado con GL_LINE_LOOP

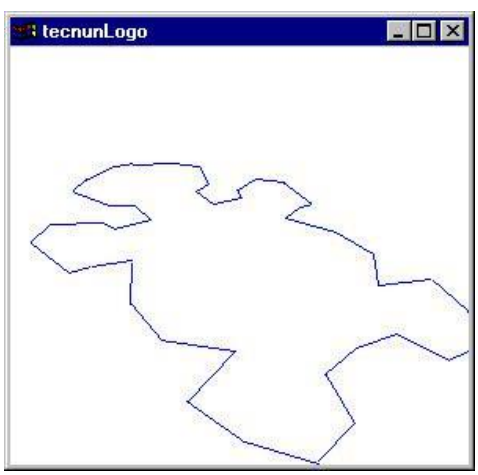

Figura 5Tortuga con GL_LINE_LOOP

Realizar la función ***draw SphereTurtle****() que dibuja una tortuga mediante la primitiva de la función glutWireSphere(). Esta función tiene como argumentos el radio y la precisión en latitud y longitud.*

En la figura 6 se muestra el resultado utilizando 6 esferas. La precisión utilizada en ambos ejes es de 10.

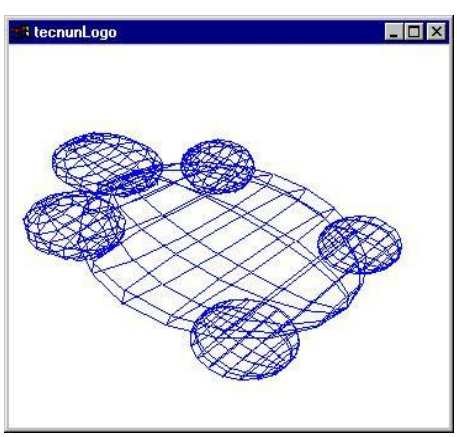

Figura 6 Tortuga con WireSphere' s