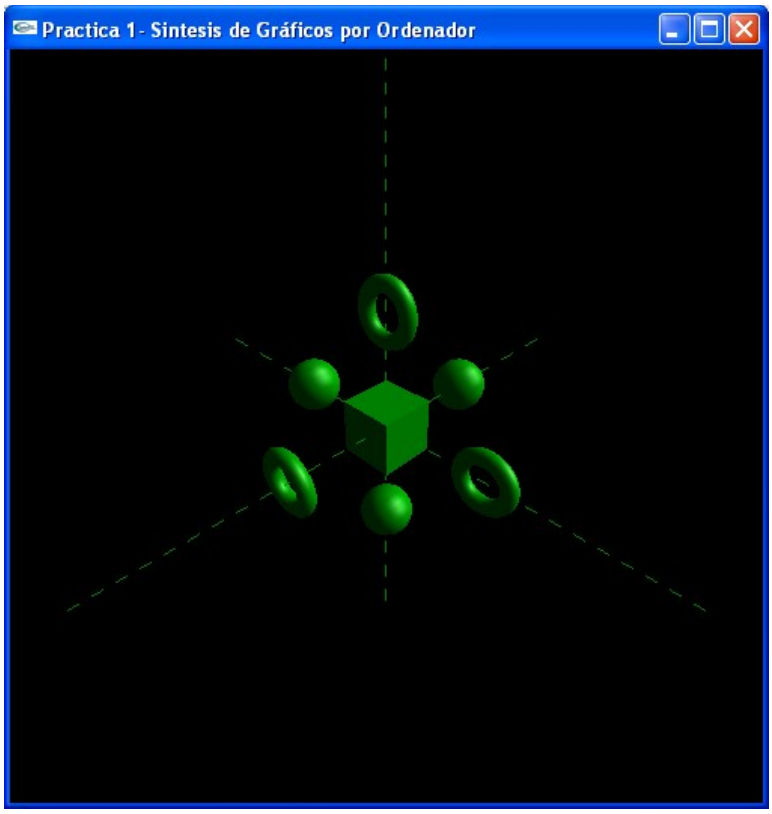
LABORATORIO DE SINTESIS DE GRÁFICOS POR ORDENADOR

Práctica 1:

INTRODUCCIÓN A OPENGL Y SUS PRIMITIVAS.



Profesor: Santiago Hermira Anchuelo
Email: santiago.hermira@uah.es

INDICE

1. INTRODUCCIÓN A OPENGL.

OpenGL es un interfaz software para hardware gráfico. Este interfaz consiste en alrededor de 150 comandos que se usan para especificar los objetos y las operaciones necesarias para producir una aplicación interactiva en 3 dimensiones.

OpenGL es una interfase independiente del hardware, lo que hace que haya sido implementado por muchas plataformas (Unix, Windows, ..). Para conseguir esta independencia HW, OpenGL no dispone de comandos para realizar ventanas o para obtener entradas del usuario, estas funciones son realizadas por aplicaciones dependientes de HW.

OpenGL, no dispone de modelos complejos para definir objetos de tres dimensiones (automóviles, cuerpos, aviones,..). Con OpenGL, si se requiere construir un objeto complejo, es necesario crearlo a partir de primitivas geométricas (puntos, líneas, polígonos).

Antes de continuar con OpenGL, es necesario conocer un termino importante "rendering", que es el proceso por el cual una computadora crea imágenes a partir de modelos. Estos modelos, u objetos, son construidos a partir de primitivas geométricas (puntos, líneas y polígonos) que son especificados mediante vértices.

2. GLUT OPENGL UTILITY TOOLKIT

Como ya hemos comentado, OpenGL esta diseñado para ser independiente del HW y del S.O, por lo que aunque es capaz de realizar "rendering", no es capaz de gestionar ventanas, o leer eventos ya sean del teclado o del ratón.

Para solucionarlo surge GLUT, que es el encargado de realizar la gestión de ventanas, eventos, … dependiente del sistema operativo. Al igual que GLUT, existen otras "toolkits" que tienen una funcionalidad parecida, por ejemplo GLAUX.

La funcionalidad de las funciones de GLUT se puede dividir en dos apartados,

- Gestión de Ventanas.
- Display Callback.

2.1. Gestión de Ventanas

Existen 5 funciones que realizan las tareas necesarias para inicializar una ventana, a continuación se presentan,

- **glutInit** (int *argc, char **argv)

  Esta función inicializa GLUT y procesa la línea de comandos (para X-Windows). Esta función debe ser invocada antes que cualquier otra función de GLUT.

- **glutInitDisplayMode** (unsigned int mode)

Especifica que características se utilizaran a la hora de representar la imagen en pantalla, el buffer de profundidad, el formato del color, doble buffer,… conceptos estos que iremos aprendiendo a lo largo de las prácticas. Un ejemplo de utilización sería

glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH).

- **glutInitWindowPosition** (int xm int y)

  Especifica la posición de la ventana desde la esquina superior izquierda de la pantalla en pixels.

- **glutInitWindowSize** (int winth, int size)

  Especifica el tamaño en pixels de la ventana

- **glutCreateWindow** (char *string)

  Esta función crea una ventana con un título. Esta función devuelve un único identificador por ventana, aunque la ventana no es presentada hasta que se ejecuta la función **glutMainLoop** ().

2.2. Display Callback.

- **glutDisplayFunc** (void (*fun.)())

  Especifica el código a ejecutar cuando se produce un evento de OpenGL. Por lo tanto, todas las rutinas que modifican la pantalla, han de colocarse en la función que es invocada por glutDisplayFunc ().

- **glutMainLoop** ()

  Es la última función a invocar en un programa OpenGL. Cuando se llama a esta función, todas las funciones que han sido creadas, se presentan en pantalla. A partir de este punto comienza el procesado de eventos y se dispara la función invocada por **glutDisplayFunc**() una vez ejecutado el bucle, este nunca termina.

3. MI PRIMER PROGRAMA.

Este primer programa muestra un ejemplo sencillo de la aplicación y funcionalidad de OpenGL. El código de "ejemplo1.c" abre una ventana y dibuja dentro un triángulo en blanco.

/* ejemplo1.c Primer programa de OpenGL */

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

void init ( void )
{
  glEnable     ( GL_DEPTH_TEST );
  glClearColor ( 0.0, 0.0, 0.0, 0.0 );
}

void display ( void )
```

```
{
  glClear ( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
  glPushMatrix ( );
  glColor4f ( 1.0, 1.0, 1.0, 1.0 );
 glTranslatef(0.0, 0.0, -4.0);
            glBegin(GL_TRIANGLES);
                        glVertex3f( 0.0f, 1.0f, 0.0f);
                        glVertex3f(-1.0f,-1.0f, 0.0f);
                        glVertex3f( 1.0f,-1.0f, 0.0f);
              glEnd();
 glPopMatrix  ( );
  glutSwapBuffers ( );
}

void reshape(int w, int h)
{
  glViewport     ( 0, 0, w, h );
  glMatrixMode   ( GL_PROJECTION );
  glLoadIdentity ( );
  if ( h==0 )
    gluPerspective ( 80, ( float ) w, 1.0, 5000.0 );
  else
    gluPerspective ( 80, ( float ) w / ( float ) h, 1.0, 5000.0 );
  glMatrixMode   ( GL_MODELVIEW );
  glLoadIdentity ( );

}

void keyboard ( unsigned char key, int x, int y )
{
  switch ( key ) {
    case 27:  /*  tecla de Escape  */
      exit ( 0 );
      break;
    case 'f':
      glutFullScreen ( );
      break;
    case 'w':
      glutReshapeWindow ( 250,250 );
      break;
    default:
      break;
  }
}

void mouse ( int button, int state, int x, int y )
{
if ( button == GLUT_LEFT_BUTTON && state == GLUT_DOWN )
  {
    printf(" \nSe pulso el botón izquierdo del ratón");
}
}

int main ( int argc, char** argv )
{
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowSize  (250,250);
  glutInitWindowPosition  (100,100);
```

```
  glutCreateWindow    ("Programa ej1.c");
  init ( );
  glutReshapeFunc      ( reshape  );
  glutKeyboardFunc   ( keyboard );
  glutMouseFunc       (mouse );
  glutDisplayFunc     ( display  );
  glutMainLoop        ( );
  return 0;
}
```

Para compilar este código, el comando a ejecutar sería:

gcc ejemplo1.c –o ejemplo1 –lglut –lGLU –lGL

3.1. ANALISIS DE LA FUNCIÓN MAIN.

La estructura básica de un programa de OpenGL se muestra en ejemplo1.c, estructura esta que pasamos a describir.

Analicemos en primer lugar la función **main()**

```
glutInit (&argc, argv);
```

Esta función es la encargada de inicializar GLUT, además de realizar un análisis de los parámetros suministrados en la línea de comandos.

```
glutInitDisplayMode (GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
```

Define el modo en el que se tiene que dibujar la ventana, como en otras funciones los parámetros a utilizar se definen con bit o máscaras de bits. Para este programa en concreto:

- **GLUT_RGB** indica que los colores se definirán con tres componentes (Red, Green, Blue)
- **GLUT_DOUBLE** indica que se utilizarán dos buffer para componer la imagen.
- **GLUT_DEPTH** indica que se va a utilizar buffer de profundidad.

```
glutInitWindowSize  (250,250);
```

Esta función define el tamaño de la ventana expresada en pixels.

```
glutInitWindowPosition  (100,100);
```

Esta función indica la posición de la esquina superior izquierda de la ventana desde la esquina superior izquierda de la pantalla.

```
glutCreateWindow    ("Programa ejemplo1.c");
```

Esta función es la que propiamente crea la ventana, el parámetro es el nombre asignado a la misma.

```
init ( );
```

En esta función definida por el usuario, se inicializan parámetros propios de OpenGL antes de pasar el control del programa a GLUT. Para el ejemplo,

```
glEnable     ( GL_DEPTH_TEST );
```

Se habilita la utilización de cálculos de profundidad.

```
glClearColor ( 0.0, 0.0, 0.0, 0.0 );
```

Se limpia el buffer de color y se define por defecto el color negro.

```
glutDisplayFunc     ( display  );
```

Aquí se define el primer callback. La función pasada como parámetro, será llamada cada vez que GLUT determine que la ventana debe ser redibujada (por ejemplo, al maximizar, poner otras ventanas encima, ...)

```
glutMainLoop        ( );
```

Esta función cede el control de flujo del programa a GLUT, que a partir de estos "eventos", irá llamando a las funciones que han sido pasadas como callback.

3.2 Análisis de la función display ().

Como ya hemos comentado, la función **display()** es la función que se ejecuta cuando se produce algún evento de GLUT. Esta función ha de contener el código de los objetos que queremos dibujar.

```
glClear ( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

Esta función resetea los buffer de color y profundidad, eliminando posibles contenidos de ventanas anteriores.

```
glPushMatrix ( );
```

En OpenGL, los cálculos de rotaciones, texturas, translaciones, se realizan mediante el cálculo de matrices. Esta función crea una copia de la matriz de modelado y la empuja una posición la pila de matrices.

```
glColor4f (1.0, 1.0, 1.0, 1.0);
```

Esta función especifica el color en formato RGB, el color es blanco.

```
glTranslatef ( 0.0, 0.0, -4.0 );
```

Esta función realiza una translación del origen de coordenadas al punto (0,0,-4)

.

```
glBegin(GL_TRIANGLES);
                    glVertex3f( 0.0f, 1.0f, 0.0f);
                    glVertex3f(-1.0f,-1.0f, 0.0f);
```

```
                    glVertex3f( 1.0f,-1.0f, 0.0f);
glEnd();
```

Esta estructura es la encargada de dibujar el triángulo. **glBegin()** comienza una secuencia de vértices con los que se construirá la primitiva. El tipo de primitivas viene dado por el parámetro de glBegin(), en este caso GL_TRIANGLES que se encarga de definir un triángulo. **glEnd()** se encarga simplemente de cerrar la estructura. Existen más posibles parámetros de glBegin () para dibujar otras primitivas, que se verán mas adelante.

```
glPopMatrix  ( );
```

Una vez realizadas los cálculos sobre la matriz de segundo plano, se coloca esta en la cima de la pila de matrices.

```
glutSwapBuffers ( );
```

Esta función se encarga de intercambiar el buffer posterior con el buffer anterior, siendo necesaria porque se ha definido que se trabaje con un doble buffer. La utilización de este doble buffer evita parpadeos en la generación de la imagen.

El resultado del código anteriormente descrito fue

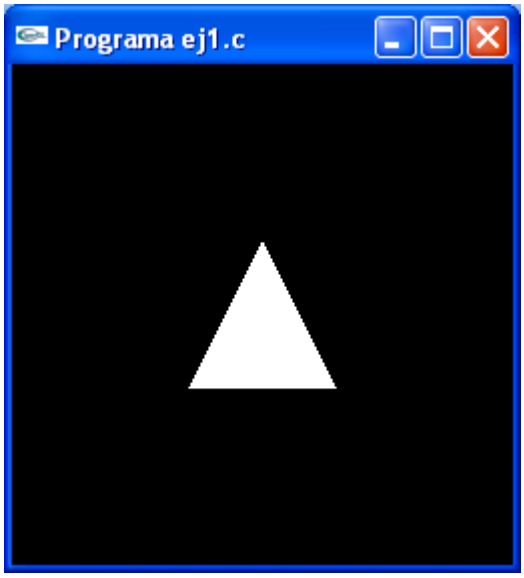


Ilustración 1 Triangulo.

3.2. Análisis de la función reshape(). Definición del área de proyección inicial.

OpenGL, es una interfaz de diseño gráfico en 3D, por lo tanto, todas las primitivas han de ser situadas en el espacio 3D. En la siguiente figura, se presenta como se modela el espacio en 3D.

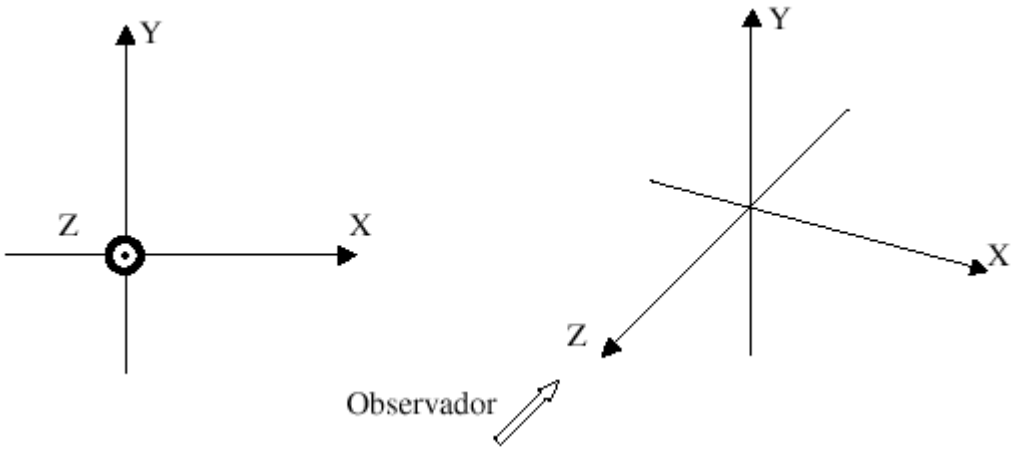


Ilustración 2 Modelado del espacio 3D.

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 que 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.

Conviene 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.

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

```
glutReshapeFunc(reshape);
```

Esta función indica la callback a ejecutar cuando se realiza una redimensión de la ventana. También es utilizada para definir el área de proyección de la figura en la ventana.

```
void reshape(int w, int h) {
```

Como toda callback, la función reshape() es de tipo void. Se pasan como argumentos los valores de ancho y alto después de la redimensión de la ventana.

```
glViewport(0, 0, w, h);
```

Esta función indica la porción de ventana donde OpenGL podrá dibujar.

```
glMatrixMode(GL_PROJECTION);
```

Esta función especifica la matriz de proyección como matriz sobre la que se van a realizar las transformaciones. OpenGL dispone de tres tipos de matrices, la matriz de proyección (GL_PROJECTION), la matriz de modelo (GL_MODELVIEW) y la matriz de textura (GL_TEXTURE).

```
glLoadIdentity();
```

Esta función carga como matriz de proyección la matriz identidad.

```
gluPerspective(80.0, (GLfloat)h / (GLfloat)w, 1.0, 5000.0);
```

Esta función 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.

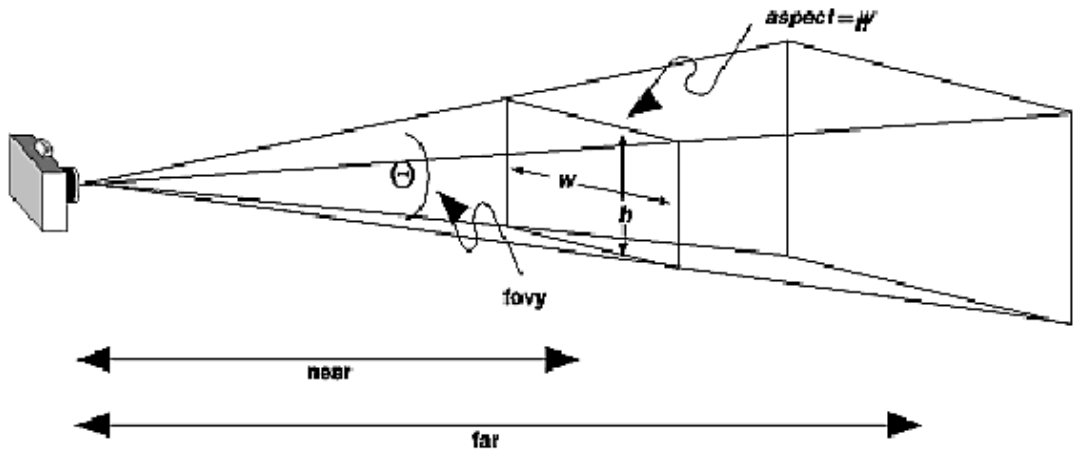


Ilustración 3 Definición de perspectiva en OpenGL.

```
glMatrixMode(GL_MODELVIEW);
```

Esta función especifica la matriz de modelado como matriz sobre la que se van a realizar las transformaciones.

```
glLoadIdentity();
```

Esta función carga como matriz de modelado la matriz identidad.

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

Esta función 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, el vector de dirección.

3.3. Análisis de la función keyboard(). Eventos de teclado.

El objetivo de este punto es el de indicar los mecanismos que nos ofrece GLUT para interactuar con el entorno definido cuando se detecta un evento de teclado.

Al igual que ya hemos comentado en los puntos anteriores, esta interacción se realiza mediante llamadas a callback. La función que indica a GLUT la callback a ejecutar ante un evento de teclado es **glutKeyboardFunc** (keyboard) que ha de ser colocada en la función **main**()

```
glutKeyboardFunc (keyboard);
```

Esta función indica la callback utilizada por GLUT ante un evento de teclado.

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

Esta función es llamada cada vez que se produce un evento de teclado, recibe como parámetros la tecla pulsada y la posición del ratón en ese instante.

```
switch (key) {
    case 27:  /* tecla de Escape */
      exit (0);
      break;
        case 'w':
      glutReshapeWindow (250,250);
      break;
    default:
      break;
  }
```

En esta sección de código se indica las tareas a realizar en caso de que se haya pulsado una tecla. En el ejemplo en caso de pulsar la tecla 27 (ESC) se termina la ejecución del programa. En caso de que la tecla pulsada fuera “w”, se redimensiona la pantalla a un tamaño de 250 por 250 pixels.

3.4. Análisis de la función mouse (). Eventos de ratón.

El objetivo de este punto es el de indicar los mecanismos que nos ofrece GLUT para interactuar con el entorno cuando se detecta un evento de ratón.

Al igual que en puntos anteriores, esta interacción se realiza mediante llamadas a callback. La función que indica a GLUT la callback a ejecutar ante un evento de teclado es **glutMouseFunc** (mouse) que ha de ser colocada en la función **main**()

```
glutMouseFunc (mouse);
```

Esta función indica la callback utilizada por GLUT ante un evento de teclado.

```
void mouse ( int button, int state, int x, int y )
```

Esta función es llamada cada vez que se produce un evento de ratón, recibe como parámetros el botón del ratón, si se ha pulsado o se ha soltado, y la posición del ratón ante el evento.

```
if ( button == GLUT_LEFT_BUTTON && state == GLUT_DOWN )
  {
```

```
    printf(" \nSe pulso el botón izquierdo del ratón");
}
```

En esta sección de código se indica las tareas a realizar en caso de que se haya producido algún evento de ratón (pulsar un botón, soltar un botón,...). En el ejemplo, en caso de pulsar el botón izquierdo del ratón, se imprime por pantalla el mensaje **"Se pulsó el botón izquierdo del ratón".**

4. PLANTILLA.

A continuación se presenta una plantilla básica para la realización de programas con OpenGL.

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

void init ( void )
{
/*  En esta sección se indicarán los parámetros de inicialización de OpenGL (estado de buffers, .) */
}

void display ( void )
{
/*  En esta sección, se indicarán las características de OpenGL .incluyendo  el código necesario para representar los objetos en pantalla */

}

void reshape ( int w, int h)
{
/*  En esta sección, se indicará el código a ejecutar cuando se produce una redimensión de la ventana. También es utilizada para definir el área de proyección de la figura en la ventana.
*/
}

void keyboard (unsigned char key, int x, int y)
{
/*  En esta sección, se indicarán las tareas a realizar ante un evento de teclado. Se ha de tener presente que todos los eventos de teclado no producen un redibujado de la pantalla, por lo que es conveniente forzar este redibujado con funciones como glutPostRedisplay () */
}

void mouse ( int button, int state, int x, int y )
{
/*  En esta sección, se indicarán las tareas a realizar ante un evento de ratón. Se ha de tener presente que todos los eventos de ratón no producen un redibujado de la pantalla, por lo que es conveniente forzar este redibujado con funciones como glutPostRedisplay () */
}


int main ( int argc, char** argv )
{
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  glutInitWindowSize  (250,250);
  glutInitWindowPosition  (100,100);
  glutCreateWindow    ("Título de la Ventana
  init ( );
  glutReshapeFunc     ( reshape  );
```

```
  glutKeyboardFunc    ( keyboard  );
  glutMouseFunc    ( mouse  );
  glutDisplayFunc    ( display  );
  glutMainLoop       ( );
  return 0;
}
```

5. PRIMITIVAS DE DIBUJO.

Una primitiva, es la unidad mínima que interpreta OpenGL. Básicamente, es un conjunto de vértices que OpenGL interpreta de una manera específica al representarlos en pantalla.

Para crear primitivas en OpenGL se utilizan las funciones **glBegin ()** y **glEnd().** La sintaxis para la creación de la primitiva es:

```
glBegin() ("Nombre de la primitiva");
        glVertex(…);
        …
        glVertex(…);
glEnd();
```

Las primitivas son:

- **GL_POINTS** Crea puntos individuales
- **GL_LINES** Crea líneas.
- **GL_LINE_STRIP** Crea líneas discontinuas.
- **GL_LINE_LOOP** Crea líneas cerradas.
- **GL_TRIANGLES** Crea triángulos
- **GL_TRIANGLE_STRIP** Crea triángulos discontinuos
- **GL_TRIANGLE_FAN** Crea triángulos en abanico.
- **GL_QUADS** Crea cuadrados.
- **GL_QUAD_STRIP** Crea cuadrados discontinuos
- **GL_POLYGON** Crea polígonos.

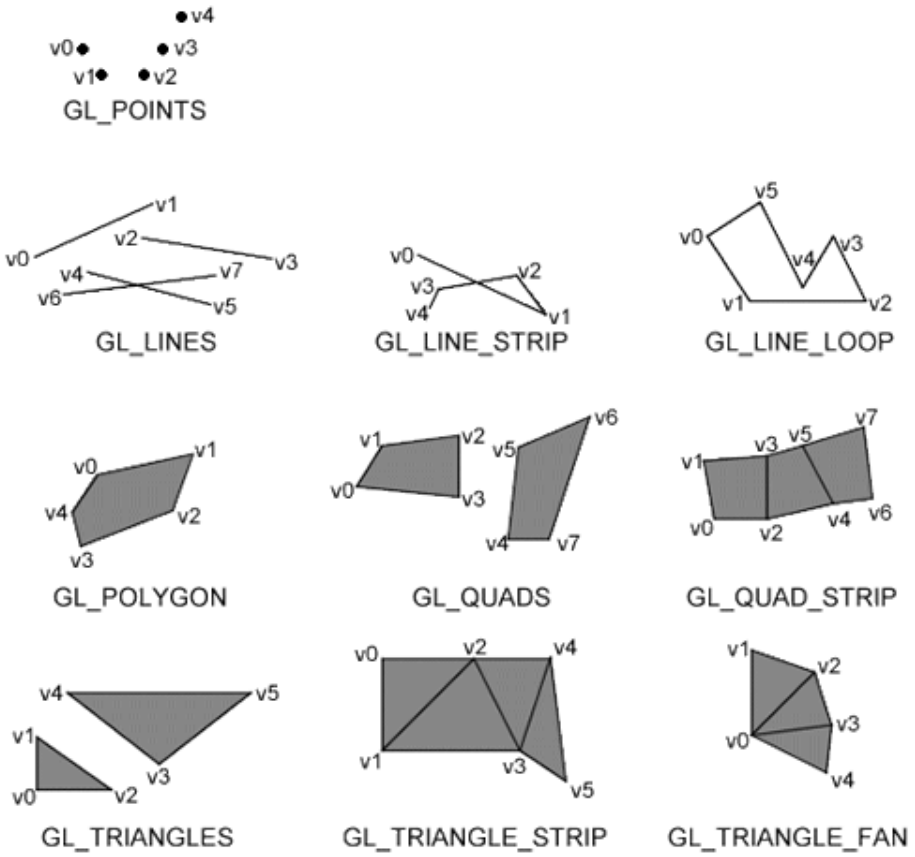


Ilustración 4 Poligonos básicos en OpenGL.

5.1. Definición de vértices

Para la definición de los vértices, existen muchas funciones, siendo todas ellas una combinación de formato de los puntos (entero, flotante,…) y el número de dimensiones asignadas al punto (2D, 3D,…)

glVertex{1234}{sifd}[v](TYPE coords)

Donde las coordenadas posibles son 1,2,3 ó 4, y los tipos de datos son sufijos de tipo:

Sufijo	Tipo de datos	Tipo de Datos en C	Definición de tipo OpenGL
B	Entero 8 bits	signed char	GLbyte
S	Entero 16 bits	short	GLshort
I	Entero 32 bits	long	GLint, GLsizei
F	Real 32 bits	float	GLfloat, GLclampf
D	Real 64 bits	double	GLdouble, GLclampd
Ub	Entero sin signo 8 bits	unsigned char	GLubyte,GLboolean
Us	Entero sin signo 16 bits	unsigned short	GLushort
Ui	Entero sin signo 32 bits	unsigned long	GLuint,GLenum,GLbitfield

Ilustración 5 Definición de Vértices en OpenGL

Ejemplos:

- glVertex2s (2, 3);

- glVertex3d (0.0, 0.0, 3.1415926536898)

5.2. Dibujo de puntos

Es la primitiva de OpenGL más simple. El parámetro pasado a la función glBegin() es GL_POINT, lo que hace que OpenGL lo interprete como puntos todos los vértices contenidos entre la función glBegin() y glEnd().

A continuación se muestra un código donde se dibujan dos puntos (0.0.0) y (2.2.2).

```
glBegin(GL_POINT);
{
        glVertex3f (0.0f, 0.0f, 0.0f);
        glVertex3f (2.0f, 2.0f, 2.0f);
}
glEnd();
```

5.3. Dibujo de líneas.

En OpenGL, cada línea queda definida por dos puntos, un punto inicial donde comienza la línea y un punto final donde acaba. Para dibujar líneas, se ha de pasar a la función glBegin () el parámetro GL_LINES, siendo evaluados los vértices por parejas. A continuación se muestra el código donde se dibujan dos líneas en el plano Z=0, que se unen en el punto (0.0.0).

```
glBegin(GL_LINES);
{
        glVertex3f ( -1.0f, -1.0f, 0.0f);
        glVertex3f ( 1.0f, 1.0f, 0.0f);
        glVertex3f ( -1.0f, 1.0f, 0.0f);
        glVertex3f ( 1.0f, -1.0f, 0.0f);
}
glEnd();
```

Obteniendo como resultado

Ilustración 6 Programa Ejemplo 1

5.4. Dibujo de triángulos.

En OpenGL, un triángulo queda definido por tres puntos. Para dibujar triángulos, se ha de pasar como parámetro a la función glBegin () el parámetro GL_TRIANGLES, evaluándose los vértices de tres en tres. En la definición de un triángulo es importante el orden que se sigue en la definición de los vértices, horario o anti horario. Cuando un polígono se define en sentido horario, OpenGL interpreta que tiene que mostrar la cara posterior, mientras que si el sentido es anti horario, OpenGL interpreta que tiene que mostrar la capa anterior. Esta particularización es importante debido a que dependiendo de los parámetros configurados en OpenGL, solo una de las caras anterior o posterior es visible.

A continuación se muestra el código que dibuja un triángulo en el plano Z=0, que ha sido definido en sentido horario, consecuentemente muestra su cara anterior.

```
glBegin(GL_TRIANGLES);
{
        glVertex3f ( -1.0f, 0.0f, 0.0f);
        glVertex3f ( 0.0f, 1.0f, 0.0f);
        glVertex3f ( 1.0f, 0.0f, 0.0f);
}
glEnd();
```

Obteniendo como resultado

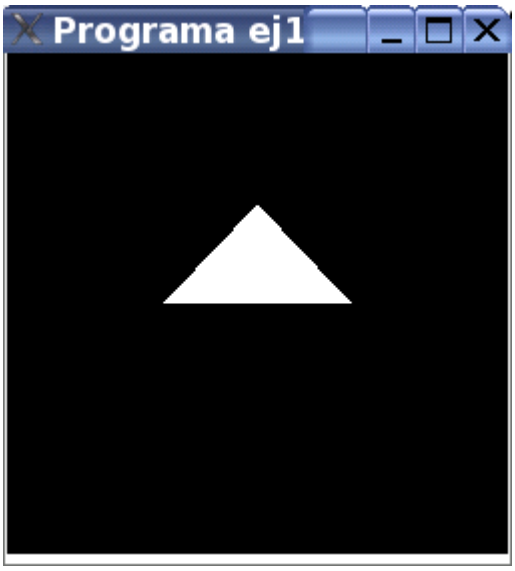


Ilustración 7 Programa ejemplo 2.

5.5. Dibujo de cuadrados.

Esta primitiva funciona exactamente igual que la de triángulos, con la salvedad de que los vértices son evaluados de 4 en 4.

6. COLOR DE RELLENO.

Para elegir el color de los polígonos, basta con hacer una llamada a glColor entre la definición de cada polígono (glBegin () – glEnd()).

La función glColor define el color de rellenado actual y lleva como parámetros los valores de las componentes RGB del color deseado, y, opcionalmente, un cuarto parámetro con el valor alpha. Al igual que con la función glVertexXX(), existen muchas funciones para definir el color, en función del número de componentes de color utilizadas y el formato de estas componentes (entero, flotante, …).

A continuación se muestra el código para dibujar un triangulo en el plano Z=0 de color azul.

```
glBegin(GL_TRIANGLES);
{
        glColor3f ( 0.0f, 0.0f, 1.0f);
        glVertex3f ( -1.0f, 0.0f, 0.0f);
        glVertex3f ( 0.0f, 1.0f, 0.0f);
        glVertex3f ( 1.0f, 0.0f, 0.0f);
}
glEnd();
```

Obteniendo como resultado

Ilustración 8 Programa ejemplo 3

7. MODELO DE SOMBREADO.

Es el método que utiliza OpenGL para rellenar de color los polígonos. Se especifica con la función glShadeModel. Si el parámetro es GL_FLAT, OpenGL rellenará los polígonos con el color activo en el momento que se definió el último parámetro; si es GL_SMOOTH, OpenGL rellenará el polígono interpolando los colores activos en la definición de cada vértice.

Este código es un ejemplo de GL_FLAT:

```
glShadeModel (GL_FLAT);
glBegin(GL_TRIANGLES);
{
        glColor3f ( 0.0f, 0.0f, 1.0f);  // Activamos el color Azul.
        glVertex3f ( -1.0f, 0.0f, 0.0f);
        glColor3f ( 1.0f, 0.0f, 0.0f);   // Activamos el color Rojo.
        glVertex3f ( 0.0f, 1.0f, 0.0f);
        glColor3f ( 0.0f, 1.0f, 0.0f);   // Activamos el color Verde.
        glVertex3f ( 1.0f, 0.0f, 0.0f);
}
```

Obteniendo como resultado un triángulo de color Verde

Ilustración 9Utilización de GL_FLAT.

Sin embargo, el mismo código cambiando la primera línea,

```
glShadeModel (GL_SMOOTH);
glBegin(GL_TRIANGLES);
{
        glColor3f ( 0.0f, 0.0f, 1.0f);  // Activamos el color Azul.
        glVertex3f ( -1.0f, 0.0f, 0.0f);
        glColor3f ( 1.0f, 0.0f, 0.0f);   // Activamos el color Rojo.
        glVertex3f ( 0.0f, 1.0f, 0.0f);
        glColor3f ( 0.0f, 1.0f, 0.0f);   // Activamos el color Verde.
        glVertex3f ( 1.0f, 0.0f, 0.0f);
}
```

se obtiene un triángulo donde se han interpolado los colores asignados a los vértice, como se puede ver en la figura.

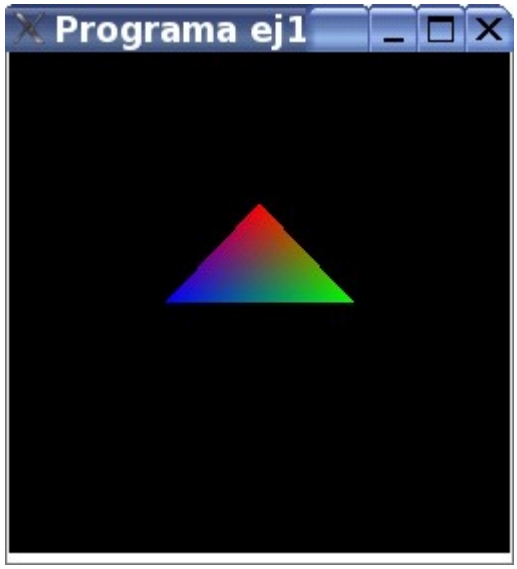


Ilustración 10 Utilización de GL_SMOOTH

8. OBJETOS GEOMÉTRICOS 3D.

GLUT incluye un conjunto de rutinas que permiten la generación de forma fácil de objetos geometricos en 3D. A continuación se presentan las más importantes, así como un ejemplo de su uso.

- **glutSolidSphere**
- **glutSolidCube**
- **glutSolidCone**
- **glutSolidTorus**
- **glutSolidDodecahedron**
- **glutSolidOctahedron**
- **glutSolidTetrahedron**
- **glutSolidIcosahedron**
- **glutSolidTeapot**

A continuación se presenta el código necesario para dibujar un dodecaedro solido

```
glutSolidDodecahedron();
```

Obteniendo como resultado

Ilustración 11 Dodecaedro sólido.

9. DESARROLLO DE LA PRÁCTICA.

9.1. Definición del eje de coordenadas.

En la ventana del área de trabajo, dibujar los ejes de coordenadas X de color rojo, Y de color verde y Z de color azul con líneas discontinuas, como se indica en la figura.

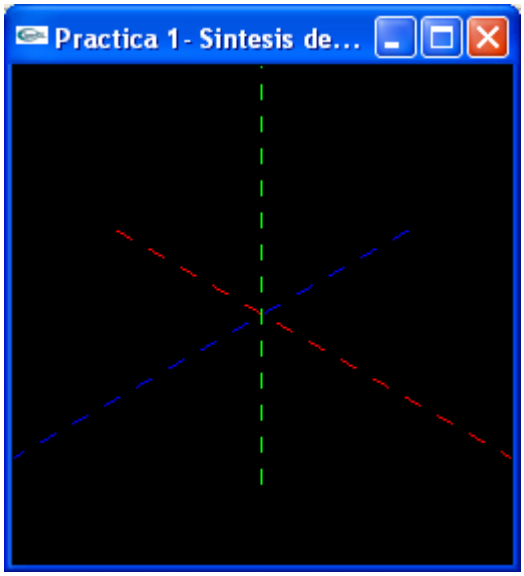


Ilustración 12 Ejes de Coordenadas

9.2. Dibujo de un cubo sólido

En el espacio tridimensional creado, con centro en el origen de coordendas, dibujar un cubo sólido de color amarillo, como se indica en la figura

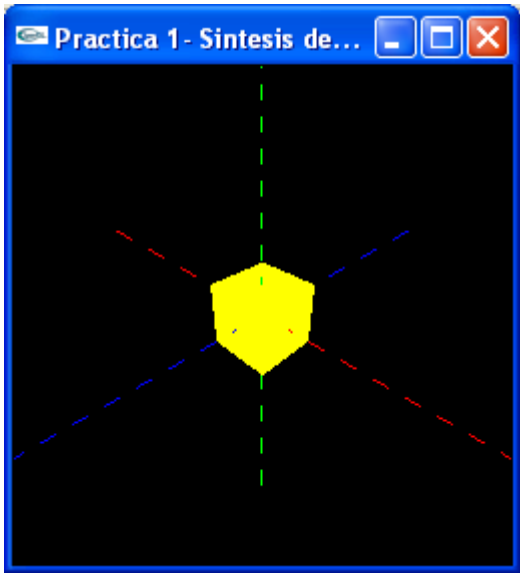


Ilustración 13 Cubo sólido

9.3. Dibujo de toro y esfera.

En el espacio tridimensional establecido, dibujar sobre cada uno de los ejes coordenados y del color de estos un toro y una esfera como se muestran en la figura adjunto.

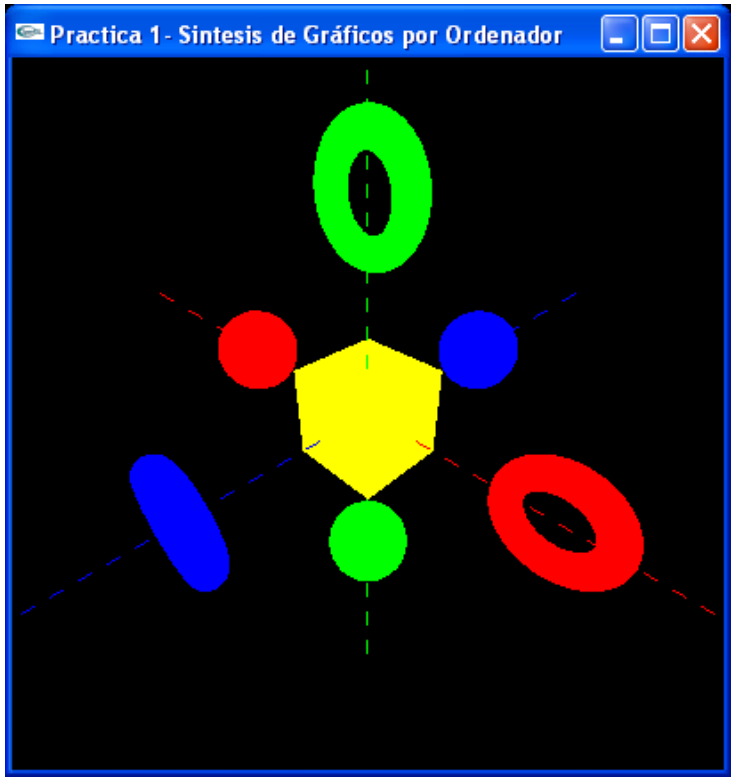


Ilustración 14 Toros y esferas

9.4. Asignación de colores e interceptación de eventos de teclado.

Modificar los colores asignados a cada uno de los ejes coordenados y sus figuras con el siguiente patrón:

1 Modificar el color asignado al eje X.

2 Modificar el color asignado al eje Y.

3 Modificar el color asignado al eje Z.

9.5. Interceptación de eventos de ratón.

Modificar el tamaño de las figuras geométricas creadas de forma que aumenten su tamaño al pulsar el botón derecho y disminuyan su tamaño al pulsar el botón izquierdo, como se indican en las figuras adjuntas.

Situación inicial

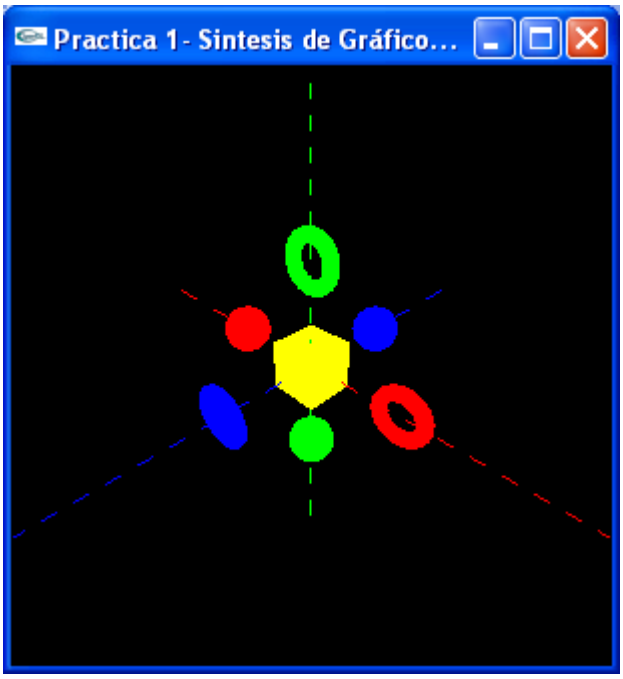


Ilustración 15

Pulsando botón derecho:

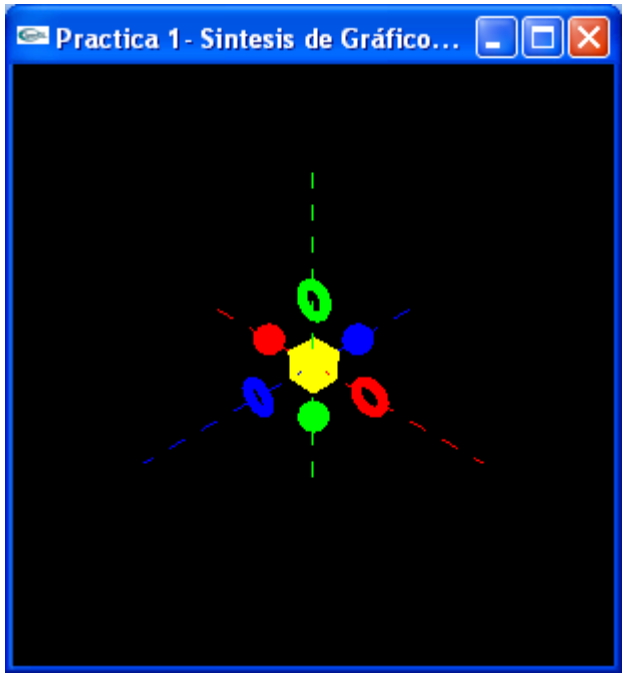


Ilustración 16 Pulsando botón derecho.

Pulsando botón izquierdo

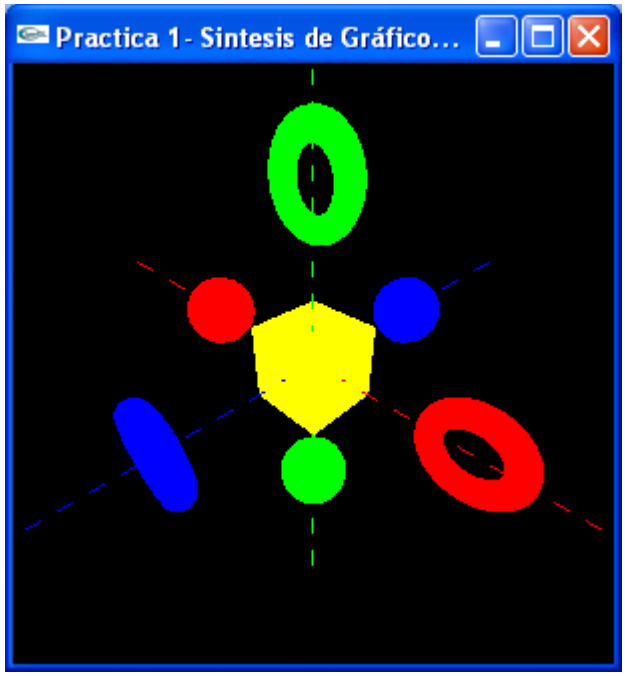


Ilustración 17 Pulsando botón izquierdo

9.6. Planificación.

Para la realización de la práctica 1 se disponen de 3 sesiones de laboratorio.

9.7. Evaluación de la práctica.

Tras la realización de la práctica, esta se mostrará al profesor de la asignatura. Así mismo, se entregará una memoria sobre la práctica, donde se comenten las principales funciones utilizadas en la práctica y su funcionamiento. Así mismo, se entregará el código fuente C de la práctica y los makefiles utilizados para la compilación en su caso.

10. BIBLIOGRAFÍA.

10.1. Libros:

- **Opengl programming Guide (Red book).**

  Libro que da una visión completa de OpenGL, con ejemplos prácticos.

  http://www.opengl.org/documentation/red_book/

- **Opengl reference manual (Blue book).**

  Libro que proporciona una guía rápida a cada una de las funciones de OpenGL

  http://www.opengl.org/documentation/blue_book/

- **Programación en OpenGL. Una guía de referencia completa de OpenGL**
  - Autores: Richard S. Wright JR / Michael Sweet.
  - Editorial Anaya
  - ISBN 84-415-0176-9

10.2. URL:

- Web Home de OpenGL
  www.opengl.org

- Una de las implementaciones mas utilizadas de OpenGL. Se pueden descargar manuales, código fuente, ejemplos, ...

  www.mesa3d.org