INTRODUCCIÓN A OpenGL

INTRODUCCIÓN A OpenGL

1. Introducción

En esta tema se van a mostrar las ideas básicas para realizar dibujos simples utilizando OpenGL. Se verá la representación de algunas funciones matemáticas, como dibujar polilíneas y polígonos y la utilización de las librerías gráficas.

2. Abriendo una ventana gráfica para dibujar

El primer paso es abrir una ventana en la pantalla para dibujar. En OpenGL las funciones son "independientes del *device*", esto es provee herramientas para abrir una ventana independiente del sistema que se use.

A continuación se muestra el código para abrir una ventan, las primeras cinco funciones del programa hacen una llamada a las utilidades de OpenGL para abrir una ventana, estas funciones se encargan de abrir una ventana en la pantalla. A continuación realizamos una breve descripción de las funciones:

```
// Programa incompleto
void main (int argc, char** argv)
{
glutInit(&argc, argv); //Inicializa toolkit
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //Selecciona el modo de pantalla
glutInitWindowSize(640,480);      //Selecciona el tamaño de la ventana
glutInitWindowPosition(100, 150); //Posiciona la ventana en la pantalla
glutCreateWindow("Mi primer trabajo en OpenGL");//Abre la ventana en pantalla
glutDisplayFunc(myDisplay); //Registra la función de redibujar
myInit();
glutMainLoop(); //Bucle continuo
}
```

- `glutInit(&argc,argv);` Esta función inicializa las utilidades de OpenGl. Sus argumentos son los estándar para pasar información sobre los comandos de línea, que no los utilizaremos aquí.
- `glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);` Esta función especifica como se inicializa la ventana. Está indicando que se va a utilizar un buffer simple y que los colores se va a especificar mediante la mezcla del rojo(Red), verde(Green) y azul(Blue). El buffer doble se utiliza para animaciones.
- `glutInitWindowSize(640,480);` Esta función especifica que la ventana se debe iniciar con un tamaño de 640x480 pixel de ancho por alto. Una vez que el programa está corriendo el usuario puede modificar el tamaño si lo desea.
- `gluInitWindowPosition(100,150);` Esta función especifica que la esquina superior izquierda se situará en la pantalla 100 pixel sobre el borde izquierdo y 100 pixel sobre el superior.
- `gluCreateWindow("Mi primer dibujo");` Esta función actúa poniendo el título indicado en la barra superior de la ventana.

Cualquier otra función debe ser implementada por el programador.

3. Dibujo de primitivas gráficas

Deseamos desarrollar técnicas de programación que permitan dibujar diferentes formas geométricas. Los comandos de dibujar se realizarán mediante llamada a funciones asociadas con eventos de re-dibujar.

Lo primero que hay que hacer es establecer un sistema de coordenadas en el que podamos describir los objetos gráficos y determinar en que posición de la pantalla se van a representar. Existen varios métodos de representar el sistema de coordenadas en la ventana gráfica. Comenzaremos por el más simple e intuitivo que consiste en utilizar directamente la posición en la pantalla mediante pixels. En la figura 1 se muestra una ventana de 640x480 pixels. La coordenada x varía entre 0 y 639 y la coordenada y entre 0 y 479. El origen se encuentra en la esquina inferior izquierda.

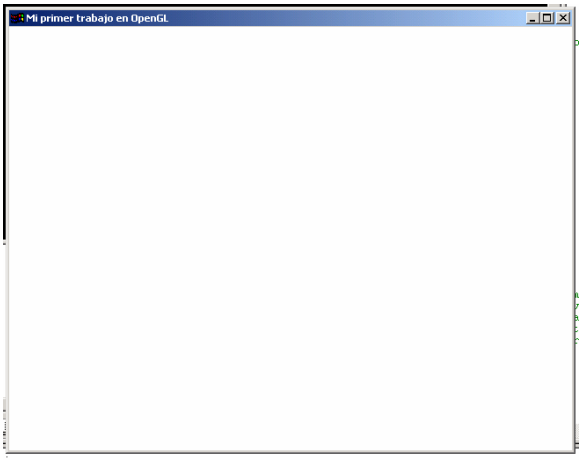

Figura 1. Sistema de coordenadas inicial. Esquina inferior izquierda 0,0. Superior derecha 640,480

OpenGL tiene herramientas que permiten dibujar primitivas, tales como líneas, polilíneas y polígonos, que están definidos por vértices. Para dibujar los objetos hay que pasar una lista de vértices. Esta lista ocurre entre dos funciones llamadas glBegin() y glEnd(). El argumento de glBegin() determina que objeto se dibuja. Por ejemplo en la figura 2 se presentan tres puntos en una ventana de 640x480 pixels, la secuencia es la siguiente.

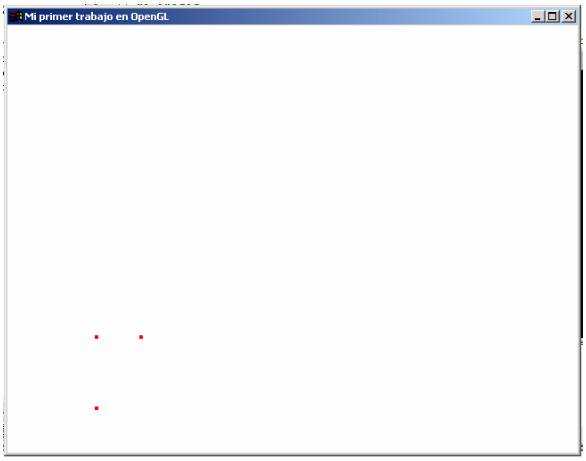

Figura 2.Dibujo de tres puntos

```
glBegin(GL_POINTS);
        glVertex2i(100, 50);
        glVertex2i(100, 130);
        glVertex2i(150, 130);
glEnd();
```

La constante GL_POINTS se construye en OpenGL. Otras primitivas son GL_LINES, GL_POLYGON, que se verán más delante.

Muchas funciones en OpenGL, tales como glVertex2i(), tienen varias variaciones, que permiten distinguir el número y el tipo de argumento pasado a la función. En la figura 3 se muestra el formato de la función.

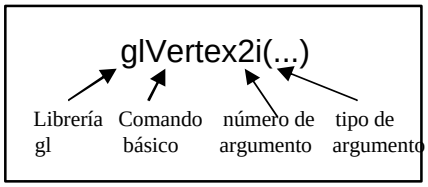

Figura 3.Formato de los comandos OpenGL

El prefijo "gl" indica que es una función de la librería OpenGL. A continuación el comando básico, seguido por el número de argumentos que se envían a la función, y finalmente el tipo de argumentos (*i* para enteros, *f* o *d* para float, etc). Es posible especificaciones sin argumento mediante un asterisco, ejemplo glVertex*(). A continuación se muestra el ejemplo anterior pasando valores reales.

```
glBegin(GL_POINTS);
        glVertex2d(100, 50);
        glVertex2d(100, 130);
        glVertex2d(150, 130);
glEnd();
```

4. Tipo de datos de OpenGL

OpenGL trabaja internamente con tipo de datos especificados. Por ejemplo, funciones tales como glVertex2i(), espera un entero de un cierto tamaño (32 bits). No hay un tamaño estándar para *float* o *double*. Para asegurar que las funciones OpenGL reciben adecuadamente el tipo de datos, se debe usar los propios de OpenGL, tales como GLint o GLfloat. En la siguiente tabla se muestran los sufijos de los comandos y tipo de argumento de los datos.

Tipo de datos OpenGL	Representación interna	Definición como tipo C	Sufijo literal en C
GLbyte	Entero de 8 bits	Char con signo	b
GLshort	Entero de 16 bits	Short	s
GLint, GLsizei	Entero de 32 bits	Long	i
GLfloat, GLclampf	Coma flotante 32 bits	Float	f
GLdouble, GLclampd	Coma flotante 64 bits	Double	d
GLubyte, GLboolean	Entero de 8 bits sin signo	Unsigned char	ub
GLushort	Entero de 16 bits sin signo	Unsigned short	us
GLuint, GLenum,GLbitfield	Entero de 32 bits sin signo	Unsigned int or long	ui

Tabla 1. Tipo de variables de OpenGL

Como ejemplo, una función que usa un sufijo *i*, espera un entero de 32-bit, pero nuestro sistema puede trasladar el int como 16-bit. Así, si deseamos encapsular el comando OpenGL para dibujar un punto en una función genérica se hará tal como muestra el código en el siguiente ejemplo:

```
void drawDOT(int x, int y)
{       //Dibuja puntos con coordenadas de valores enteros
        glBegin(GL_POINTS);
                glVertex2i(x, y);
        glEnd();
}
```

Este código pasa enteros a glVertex2i(). Trabajando en un sistema que usa enteros de 32-bit, puede causar problemas si se trabaja a 16-bit. Es por tanto más seguro re-escribir el programa utilizando GLint. En este caso el código más correcto sería.

```
void drawDOT(GLint x, GLint y)
{       //Dibuja puntos con coordenadas de valores enteros
        glBegin(GL_POINTS);
                glVertex2i(x, y);
        glEnd();
}
```

5. Estado de OpenGL

OpenGL almacena muchas de las variables, tales como el tamaño del punto, el color del dibujo y fondo de la pantalla, etc. Los valores de estado de las variables permanecen activos hasta que se les da un nuevo valor. El tamaño del punto se especifica mediante glPointSize(); que recibe como argumento un *float*. Si el argumento es 3.0, el punto normalmente se representa como un cuadrado, de tres píxel por cada lado. El color de las entidades a dibujar se puede especificar mediante glColor3f(rojo, verde, azul); En OpenGL un color sencillo se muestra se representa como una mezcla de componentes roja, verde y azul. EL rango de cada componente puede variar de 0.0 a 1.0. Los valores para cada componente pueden ser cualquier valor de coma flotante válido entre 0 y 1, abarcando por consiguiente un número teóricamente infinito de tono posibles. En términos prácticos, OpenGL representa los colores internamente como valores de 32 bits, abarcando un máximo de 4.294.967.296 tonos (lo que suele denominar color real). En consecuencia, el rango efectivo de cada componente va de 0.0 a 1.0 en pasos de aproximadamente 0.00006. En la tabla 3 se muestra alguno de los colores normales con sus valores por componente.

Color compuesto	***Componente roja***	***Componente verde***	***Componente azul***
Negro	0.0	0.0	0.0
Rojo	1.0	0.0	0.0
Verde	0.0	1.0	0.0
Amarillo	1.0	1.0	0.0
Azul	0.0	0.0	1.0
Magenta	1.0	0.0	1.0
Ciano	0.0	1.0	1.0
Gris oscuro	0.25	0.25	0.25
Gris claro	0.75	0.75	0.75
Marrón	0.60	0.40	0.12
Naranja calabaza	0.98	0.625	0.12
Rosa Pastel	0.98	0.04	0.70
Púrpura	0.60	0.40	0.70
Blanco	1.0	1.0	1.0

Tabla 2. Algunos colores comunes por componentes

El color de fondo se determina con glClearColor(rojo, verde, azul, alpha), donde alpha especifica el grado de transparencia, es decir la cualidad de un objeto para permitir pasar la luz a través suyo (por defecto se utiliza valor 0.0). Para limpiar la ventana y usar el color de fondo se utilizar glClear(GL_COLOR_BUFFER_BIT). El argumento (GL_COLOR_BUFFER_BIT) es una constante que se construye en OpenGL.

6. Sistema de Coordenadas

El método para establecer el sistema de coordenadas, se aclarará mejor más adelante cuando se especifique *"windos, viewport y clipping"*. En el ejemplo que se muestra a continuación la función myInit() es la que establece el sistema de coordenadas. La función gluOrtho2D() es la que se encarga de realizar la transformación necesaria para representar en una ventana de 640x480 pixels.

```
void myInit(void)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, 640.0, 0, 480.0);
}
```

7. Programa completo de OpenGL

A continuación se muestra el código completo de un programa simple, que representa tres puntos. En la iniciación, myInit() se selecciona el sistema de coordenadas, el tamaño del punto, el color de fondo y color de dibujo. El dibujo se encapsula en la función myDisplay(). Este programa no es interactivo, no se utiliza otras llamadas a funciones. glFlush() se llama después para asegurarse de que todos los datos se han procesado y enviado a la ventana. Esto es importante en sistemas que trabajan en red: los datos son almacenados en el buffer del *host* y se envían a la pantalla remota solamente cuando el buffer está completamente lleno o glFlush() se ejecuta.

```
//Programa completo en OpenGL para representa tres puntos
#include<windows.h>
#include<gl/Gl.h>
#include<gl/glut.h>
// Mi_inicio
void myInit(void)
{
             glClearColor(0.0,0.0,0.0,0.0); //Selecciona el color de fondo
             glColor3f(1.0f, 0.0f, 0.0f);  //El color a dibujar
             glPointSize(4.0);                 //Tamaño de los puntos
             glMatrixMode (GL_PROJECTION);
             glLoadIdentity();
             gluOrtho2D(0.0, 640.0, 0.0, 480.0);
}
void myDisplay(void) // Mi_Ventana
{
       glClear(GL_COLOR_BUFFER_BIT); //Limpia la pantalla
       glBegin(GL_POINTS);
              glVertex2i(100, 50);//Dibuja los tres puntos
              glVertex2i(100, 130);
              glVertex2i(150, 130);
       glEnd();
       glFlush(); //Envía toda la salida a la pantalla
}
// MAIN
void main (int argc, char** argv)
{
glutInit(&argc, argv); //Inicializa toolkit
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //Selecciona el modo pantalla
  glutInitWindowSize(640,480);   //Selecciona el tamaño de la ventana
  glutInitWindowPosition(100, 150); //Posiciona la ventana en la pantalla
  glutCreateWindow("Mi primer trabajo en OpenGL");//Abre la ventana
  glutDisplayFunc(myDisplay); //Registra la función de redibujar
  myInit();
```

```
  glutMainLoop(); //Bucle continuo
}
```

8. Dibujo de líneas con OpenGL

La línea es la entidad básica en los gráficos por ordenador, todos los sistemas gráficos tiene "driver" con "rounitas" para dibujar líneas. Con OpenGL es muy fácil dibujar líneas usando GL_LINES como argumento en la función glBegin(), y pasando los dos puntos correspondientes a las vértices del segmento a representar. Ejemplo para dibujar una línea entre los puntos de coordenadas (40, 100) y (202, 96) el código sería el siguiente:

```
glBegin(GL_LINES); //Se pasa como argumento la constante GL_LINES
      glVertex2i(40, 100);
      glVertex2i(202, 96);
glEnd();
```

Este código puede ser encapsulado por conveniencia en una función drawLineInt();

```
Void draLineInt(GLint x1, GLint y1, GLint x2 GLinty2)
{
      glBegin(GL_LINES);
            glVertex2i(x1, y1);
            glVertex2i(x2, y2);
      glEnd();
}
```

Si se especifican más de dos vértices entre glBegin(GL_LINES) y glEnd(), estos se taman de dos en dos, se dibujan líneas separadas cada dos puntos. El siguiente ejemplo represeta una línea horizontal y otra vertical:

```
      glBegin(GL_LINES);
            glVertex2i(10, 40);
            glVertex2i(40, 20);
            glVertex2i(20, 10);
            glVertex2i(20, 40);
            //aqui se pueden añadir más puntos si se desea
      glEnd();
      glFlush();
```

OpenGL tiene herramientas para modificar las atributos de las líneas. El color de la línea se designa de la misma manera que los puntos, usando glColor3f(). En la figura 3 se muestra un ejemplo de diferentes grosores utilizando glLineWidth(GLfloat ancho). Por defecto el valor de grosor es 1.0.

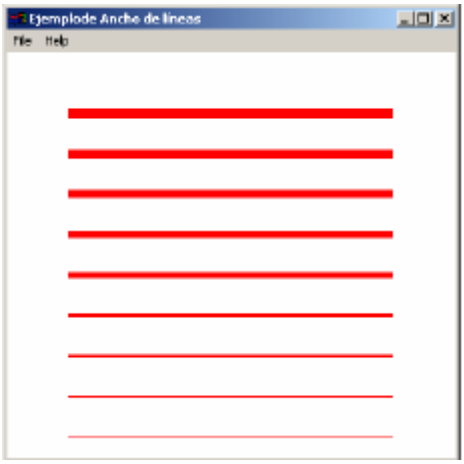

Figura 3.Ejemplo de distintos anchos de líneas

La función glLineWidth toma un solo parámetro que especifica el ancho aproximado, en pixels, de la línea dibujada. Al igual que el ancho de puntos no están soportados todos los anchos de línea, y hay que verificar que el ancho de línea que se quiere está disponible para asegurarse. A continuación se muestra el código que hay que utilizar para conocer el rango del ancho de línea y el intervalo más pequeño entre ello.

```
GLfloat dimensn[2];     //Almacena el rango de tamaños de líneas soportados
GLfloat paso;           //Almacena los incrementos en el tamaño de línea soportado
// Obtiene el rango y paso de los tamaños de punto soportados
glGetFloatv(GL_LINE_WIDTH_RANGE, dimensn);
glGetFloatv(GL_LINE_WIDTH_GRANULARITY, &paso);
```

Aquí la matriz *dimensn* contendrá dos elementos con los valores más pequeño y más grande disponible entre ancho de línea. La implementación de Microsoft permite tamaños de línea desde 0.5 a 10.0, con el paso mínimo de tamaño de 0.125.

9. Patrones de línea

Además de cambiar el ancho de líneas, se pueden crear líneas con un patrón punteado o rayado, denominado *patron*. Para usar patrones de línea lo primero que hay que hace es habilitar el patón con la función glEnable(GL_LINE_STIPPLE). Entonces la función glLineStipple establece el patrón que usarán las líneas para dibujar. Cualquier función que se activa con glEnable() puede desactivarse con glDisable().

```
Void glLineStipple ( Glint factor, Glushort patron);
```

El parámetro *patron* es un valor de 16 bits que especifica el patrón que habrá que usar cuando se dibuje la línea. Cada bit representa una sección del segmento de línea que puede o no estar activo. Por ejemplo, cada bit corresponde a un solo píxel, pero el parámetro *factor* sirve como multiplicador para incrementar el ancho el patrón.

Por ejemplo, seleccionando el *factor* 5 se provoca que cada bit en el patrón represente cinco pixels en una fila que puede o no estar activa. Además el bit 0 del patrón se emplea primero para especificar la línea. La figura 4 muestra un ejemplo aplicado a un segmento de línea, usando el patrón 0x555, pero para cada línea el multiplicador de patrón es incrementado en 1.

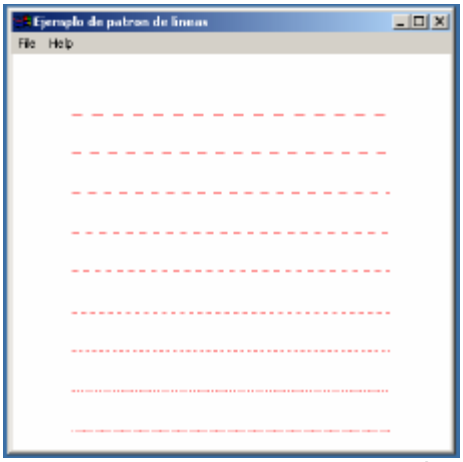

Figura 4.Ejemplo de distintos patrones de líneas

En la siguiente tabla se muestran alguno ejemplo de patrones. El patrón se da en formato hexadecimal. Ejemplo para 0xEECC el patrón especificado es el siguiente 1110111011001100.

Patrón	Factor	resultado
0xFF00	1	…….. …….. ……. …….. …….
0xFF00	2	…… …….. ……
0x5555	1	. . . . .
0x3333	2	.. ..
0x7733	1	… … .. .. … … .. .. … … .. ..

Tabla 3.Ejemplos de patrones simples

10. Dibujo de polilíneas y polígonos

Se entiende por polilínea a una colección de segmentos de tal forma que el último de uno es el primero del siguiente. Se suelen escribir como lista ordenada de puntos como la siguiente ecuación

$$p_0 = (x_0, y_o), p_1 = (x_1, y_1), \ldots p_n = (x_n, y_n),$$

En OpenGL, una polilínea se denomina “”line strip” y se dibuja especificando sus vértices, que se encuentra entre glBegin(GL_LINE_STRIP) y glEnd(), como se muestra en el código del siguiente ejemplo:

```
glBegin(GL_LINE_SRIP); // Dibuja una polilínea abierta
        glVertex2i(20,10);
        glVertex2i(50,10);
        glVertex2i(20,80);
        glVertex2i(50,80);
glEnd();
glFlush();
```

El resultado se ve en la figura 5. Los atributos tales como color, espesor y patrón se realiza de igual manera que con las líneas simples, como se vio anteriormente. Si se desea conectar el último punto con el primero, para obtener una polilínea cerrada, simplemente se remplaza (GL_LINE_STRIP) por (GL_LINE_LOOP). El resultado se muestra también en la figura5.

Los polígonos dibujados usando (GL_LINE_LOOP) no se pueden rellenar con color o patrón. Para rellenar un polígono se debe usar glBegin(GL_ POLYGON).

Figura 5 Polilínea y Polígono

11. Dibujo de polilíneas almacenadas en un fichero

Dibujos más complejos se realizan dibujando una serie de polilíneas almacenadas en un fichero que contiene la información de las coordenadas convenientemente almacenados de los distintos segmentos a representar. A continuación se muestra una routina que dibuja polilíneas almacenadas en un fichero de datos. La estructura del fichero de datos tiene el siguiente formato:

21	Número de polilíneas del archivo
4	Número de puntos de la primera polilínea
169 118	Primer punto de la primera polilínea
174 120	Segundo punto de la segunda polilínea
179 124	
178 126	
5	Número de puntos de la segunda polilínea
298 86	Primer punto de la segunda polilínea
304 92	
310 104	
314 114	
314 119	
29	
32 435	
10 439	
- - -	etc.

La routina en C++ abrirá un archivo contenido en el “string” fileName y representará las polilíneas.

```
#include <fstream.h>
void drawPolyLineFile(char *  fileName)
{
  fstream inStream;
  inStream.open(fileName, ios ::in); // abre el fichero
  if(inStream.fail() )
       return;
  glClear(GL_COLOR_BUFFER_BIT); // limpia la pantalla
  GLint numpolys, numLines, x, y;
  inStream>> numpolys;            //lee el número de polilíneas
   for(int j= 0; j < numpolys; j++) //lee cada polilínea
  {
       inStream >> numLines;
             glBegin(GL_LINE_STRIP);     // dibuja la siguiente polilínea
  for (int i =0 ; i < numLines; i++)
    {
inStream >> x >> y;  //lee el siguiente par de x, y
glVertex2i(x, y);
      }
   glEnd();
   }
glFlush();
inStream.close();
}
```

NOTA: La versión anterior, no tiene ningún control de errores. Si el archivo no se encuentra, devuelve el control a la función principal. Si el archivo contiene datos erróneos, tales como valores reales en vez de enteros, el resultado puede ser imprevisible.

12 Parametrización de figuras

En la figura 6 se muestra el dibujo de la silueta de una casa mediante polilíneas. La cual se puede dibujar mediante el código mostrado a continuación.

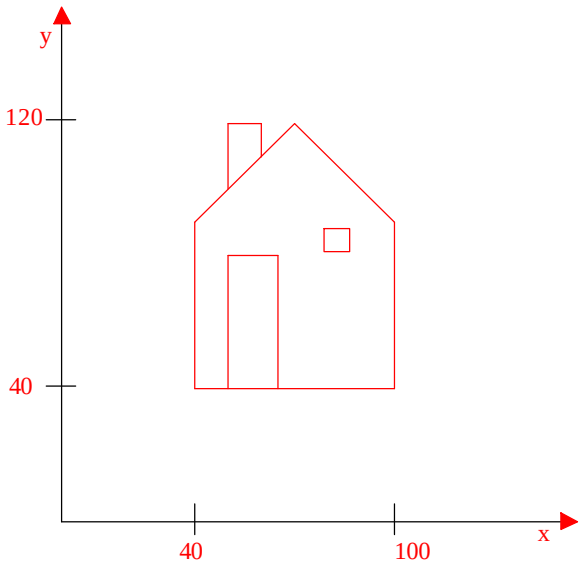

Figura 6 Dimensiones del gráfico de una casa

No es una aproximación muy flexible, ya que cada coordenada de la posición de un vértice se fija en el código, lo que implica que solamente se puede dibujar en la misma posición y con el mismo tamaño. Más flexible sería poder parametrizarla.

```
Void Casa(void)
{
glBegin(GL_LINE_LOOP);
glVertex2i(40, 40); //Dibuja la casa
glVertex2i(40, 90);
glVertex2i(70, 120);
glVertex2i(100, 90);
glVertex2i(100, 40);
glEnd();
glBegin(GL_LINE_STIP);
glVertex2i(50, 100); //Dibuja la chimenea
glVertex2i(50, 120);
glVertex2i(60, 120);
glVertex2i(60, 110);
glEnd();
. . . // Dibujar la puerta
. . .// Dibujar la ventana
}
```

De esta forma, sería posible dibujar una familia de objetos distintos al poder cambiar los parámetros que la definen. La parametrización se especifica mediante la localización del vértice superior del tejado, ancho y alto de la casa. De la misma manera se podría parametrizar la ventana y puerta.

```
Void Casa_parametrizada(GLintPoint pico, GLint ancho, Glint alto)
//El tope de la casa es pico, el tamaño se da por ancho y alto
{
glBegin(GL_LINE_LOOP);
glVertex2i(pico.x, pico.y)
glVertex2i(pico.x+ancho/2, pico.y –3 *alto/8);
glVertex2i(pico.x+ancho/2, pico.y – alto/8);
glVertex2i(pico.x-ancho/2, pico.y – alto);
glVertex2i(pico.x-ancho/2, pico.y –3 *alto/8);
glEnd();
....//Dibujar la chimenea de forma similar
....//Dibujar la puerta de forma similar
....//Dibujar la ventanade forma similar
}
```

13 Dibujo de polilíneas guardadas en una lista

Como veremos, algunas aplicaciones guardan los vértices de una polilínea en una lista. De esta forma si se añade a nuestra propia herramienta de rutinas una función que acepte la lista como parámetros se pueden dibujar las correspondientes polilíneas. La lista puede ser de forma matricial. Vamos a definir una matriz mediante la siguiente clase.

```
Class GlintPointArray{
       Const int MAX_NUM = 100;
       Public:
              Int num;
              GLintPoint pt[MAX_NUM];
};
```

A continuación se muestra una posible implementación de una routina de una polilínea. La routina toma el parámetro *cerrado*: Si *cerrado* es no nulo, el último vértice se conecta con el primero. El valor de *cerrado* se pasa como argumento a glBegin().

```
Void drawPolyline(GlintPointArray poly, int cerrado)
{
       glBegin(cerrado ? GL_LINE_LOOP : GL_LINE_STRIP);
              for (int i = 0; i < poly.num; i++)
              glVertex2i(poly.pt[i].x, poly.pt[i].y);
       glEnd();
glFlush();
}
```

14 Dibujo de rectángulos alineados

Un caso especial de un polígono es un rectángulo alineado, llamado así porque sus lados son paralelos a los ejes coordenados. Es posible construir una función para dibujarlo, pero OpenGL ya dispone de una función.

```
glRecti (GLint x1, GLint y1, GLint x2, GLint y2);
// Dibuja un rectángulo siendo (x1, y1) (x2, y2) los vértices de las esquinas
opuestas
// Rellena el rectángulo con el color actual
```

Ejemplo de código, en este caso particular el segundo rectángulo se pinta sobre el primero

```
glClearColor(1.0, 1.0, 1.0, 0.0); //Fondo de color blanco
glClear(GL_COLOR_BUFFER_BIT); //Limpia la ventana
glColor3f(0.6,0.6,0.6) //color gris claro
glRecti(20,20,100,70);
glColor3f(0.2,0.2,0.2) //color gris oscuro
glRecti(70,50,150,130);
```

```
glFlush();
```

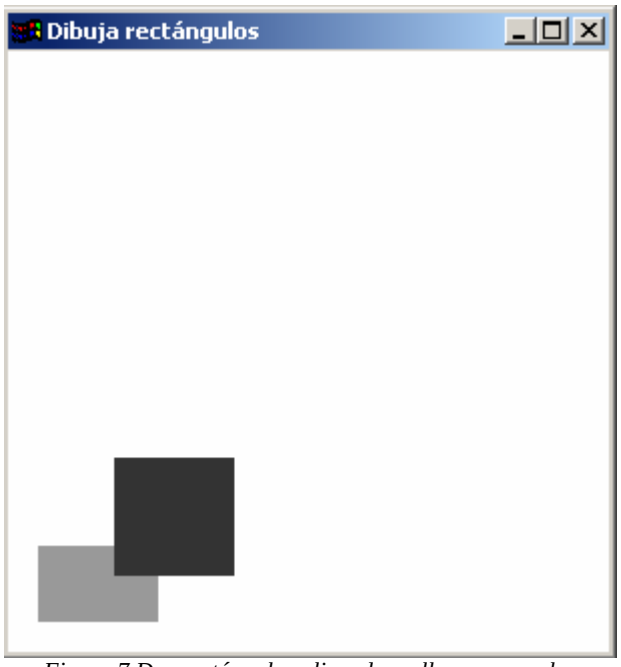

Figura 7 Dos rectángulos alineados rellenos con color

15 Relación de aspecto de rectángulos alineados

Las características principales de un rectángulo son su tamaño, posición, color y forma. Su forma viene reflejada por su **relación de aspecto**. La relación de aspecto de un rectángulo es el cociente entre el ancho y alto.

Relación de aspecto = ancho/alto

La relación de aspecto de una televisión es de 4/3 (640/480). Una relación muy famosa es la conocida como **rectángulo dorado**, en este caso su valor es de 1.618034 (El Partenón, La mona Lisa de da Vinci, El último sacramento de Dalí, son ejemplos de este caso).

Formas alternativas de especificar un rectángulo

Un rectángulo también se puede definir mediante otra forma distinta a la descrita anteriormente. Dos de estas posibilidades pueden ser:

- punto central, anchura y altura
- Esquina inferior izquierda y su relación de aspecto.

Para esto habría que definir las funciones convenientemente a partir de glRecti().

16 Rellenado de polígonos

Con OpenGL es posible rellenar polígonos con un patrón de color. La restricción es que los polígonos deben de ser convexos. Para dibujar un polígono convexo dado por los vértices *(x_0, y_0), (x_1, y_1),....., (x_n, yn),* se utiliza un lista de vértices entre glBegin(GL_POLYGON) y glEnd()

```
glBegin(GL_POLYGON);
       glVertex2f(x0, y0);
       glVertex2f(x2, y1);
       . . . . .
       glVertex2f(xn, yn);
glEnd();
```

Los polígonos se rellenan con el color actual.

17 Otras primitivas gráficas

Es posible dibujar otros cinco objetos más utilizando OpenGL como se muestra en la figura 8. Para dibujar alguno en particular simplemente hay que usarlo como argumento en gBegin(), a continuación se describen.
. GL_TRINAGLES: toma la lista de vértices de tres en tres, dibujando triángulos independientes.
. GL_QUADS: toma cuarto vértices y dibuja cuadrilateros independientes.
. GL_TRINGLE_STRIP: dibuja una serie de triángulos basado en tripleta de vértices: v_0, v_1, v_2, después v_2, v_1, v_3 y v_2, v_3, v_4, etc. (en sentido de las agujas del reloj).
. GL_TRINAGLE_FAN: dibuja una serie de triángulos conectados basados en tres vértices de la forma v_0, v_1, v_2, después v_0, v_2, v_3 y v_0, v_3, v_4
. GL_QUAD_SIP: dibuja una serie de cuadrláteros basados en cuatro vértices: primero v_0, v_1, v_3, v_2 después v_2, v_3, v_5, v_4 y v_4, v_5, v_7,v_6, etc.(en sentido de las agujas del reloj).

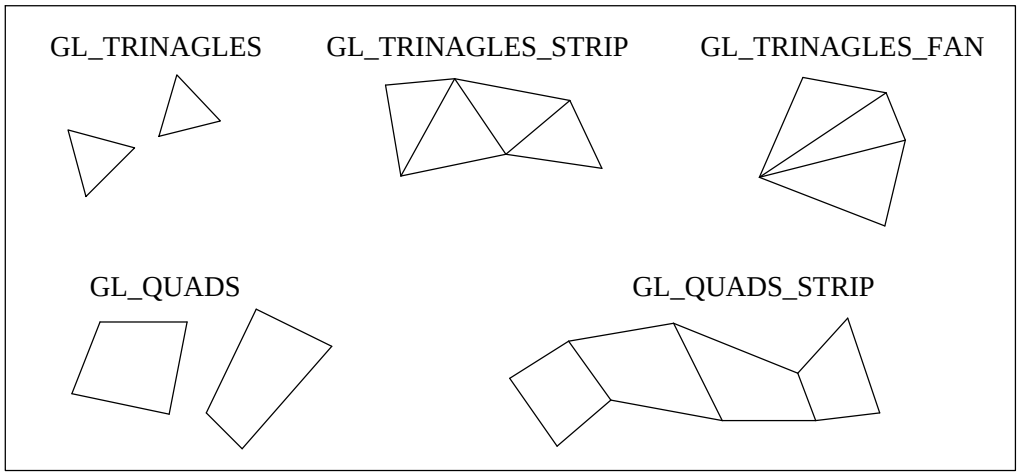

Figura 8 Otras primitivas geométricas

18 Interacción simple del ratón y el teclado

Las aplicaciones interactivas permiten al usuario controlar el flujo del programa de una forma natural, bien posicionando y pulsando el ratón o mediante el uso del teclado. La posición del ratón al pulsarlo y la identificación del teclado es posible realizarlo dentro de un programa y procesarlo adecuadamente. Usando las utilidades de OpenGL GLUT (GL Utility Toolkit) el programador puede registrar la llamada a funciones de cada uno de estos eventos mediante los siguiente comandos:

- `glutMouseFunc(myMouse)`, el cual registra en `myMouse()` el evento que ocurre cuando el se presiona el botón del ratón.
- `glutMotionFunc(myMovedMouse)`, que registra en `myMovedMouse()` el evento que ocurre cuando se desplaza el ratón con el botón pulsado.
- `glutKeyboardFunc(myKeyboard)` que registra en `myKeyBoard()` el evento que ocurre cuando se presiona una tecla.

Interacción con el ratón

Para interactuar con el ratón hay que definir la función `myMouse()` que toma cuatro parámetros, con el siguiente prototipo

```
Void myMouse(int button, int state, int x, int y)
```

Entonces cuando un evento de ratón ocurre, el sistema llama a la función registrada, simplemente con los valores de estos parámetros. El valor de `button` podrá ser

GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, y GLUT_RIGHT_BUTTON

Siendo el botón izquierdo, central o derecho respectivamente. El valor de `state` será GLUT_UP y GL_DOWN. El valor de x e y indica la posición del ratón en el momento del evento. El valor numérico de x corresponde al pixel a partir de la a izquierda de la ventana, pero el valor de y es el valor del pixel a partir de la parte superior de la ventana.

Interacción con el teclado

El presionar una tecla puede ser un evento interpretado por el programa. La función `myKeyboard()` es la que recoge el evento a través de `glutKeyboardFunc(myKeyboard)`. El prototipo de la función es el siguiente

```
Void myKeyboard(unsigned int key, int x, int y)
```

El valor de la tecla es el valor del código ASCII. El valor x e y devuelve el valor de la posición del ratón en el momento de presionar la tecla (ya se indicó anteriormente la posición del pixel en x e y).