PRÁCTICA 1. REPRESENTACIÓN DE PUNTOS, LÍNEAS, POLILÍNEAS Y RECTÁNGULOS

1.1 Representación de puntos a partir de sus coordenadas

Representar gráficamente usando librerías de OpenGL los puntos dados por sus coordenadas:

Los datos se leerán de un fichero externo que tendrá como nombre **puntos.dat**

El tamaño del punto en píxel se incluirá en el fichero externo.

La estructura del fichero, será la siguiente:

puntos.dat

```
3.0          //tamaño del pixel
63           //número de puntos
189  357     //coordenadas x, y
168  357
150  357
128  357
109  357
96   343
83   331
83   308
.....  ......
```

Los datos leídos deben mostrarse en pantalla. Ver apartado 10 cap2. pág. 22, 23 y 24

1.2 Representación mediante puntos de una función matemática

Represente gráficamente mediante puntos la siguiente ecuación:

$$f(x) = e^{-x}\cos(2\pi x) \quad 0 \le x \le 4$$

Los datos generados de x y f(x) se obtendrán en un fichero de salida llamado **output.dat**

NOTA:

Una forma rápida de representar la función sería dar incrementos equidistantes a *x* y calcular el valor correspondiente de la función para cada par de puntos $(x_i,\ f(x_i))$. Si tomamos por ejemplo un incremento de 0.005 entre cada valor de x, el proceso se podría obtener con el siguiente código:

```
glBegin (GLPOINTS);
  for(GL_double x =0; x < 4.0; x += 0.005)
    glVertex2d(x, f(x));
glEnd();
glFlush();
```

Aquí nos encontramos con un problema: el dibujo que se produce, será imposible verlo porque es muy pequeño, ya que los valores de x varían entre 0 y 4, y serán represetados entre los cuatro primeros pixels de la pantalla. Además valores negativos de f(x) caen fuera de pantalla. Por lo tanto necesitamos escalar y posicionar los valores para poder representarlos en la ventana apropiadamente. Este problema se resuelve mediante una transformación de coordenadas mundiales a coordenadas de sistema. Ver Cap. 3 pág 36 y 37

Los valores de *x* varían entre 0 y 4,

x	y
0	1
0.5	-0.6065
1	0.3678
4	0.0183

es decir Wl = 0 y Wr = 4. Los valores de *y* varía entre 1 para x =0 y –0.6065 para x = 0.5.

Usando las variables `An` y `Al` como constante y dandole valores de 640 (Vl = 0, Vr = 640) y 480 (Vb =0, Vt = 480) respecivamente, tenemos un factor de R=640/480 = 1.333

Si consideramos el ancho Wr – Wl = 4, para no distorsionar debemos fijar un valor de Wb – Wb = 4/1.333 = 3

Como
$$sx = Ax + C$$
$$sy = By + D$$

y

$$A = \frac{Vr - Vl}{Wr - Wl} \quad y \quad C = Vl - AWl$$

$$B = \frac{Vt - Vb}{Wt - Wb} \quad y \quad D = Vb - BWb$$

Sustituyendo tenemos

$$sx = An/4 * x$$

$$sy = Al/3 * y + Al/3 * 1.5$$

El código será

```
for (GLdouble x =0; x < 4.0 ; x += 0.0001)
{
    GLdouble y = exp(-x) * cos(2*GL_PI * x);
    Sx = (An/4.0)*x;
    Sy = (Al/3.0)*y + (Al*1.5)/3.0;
    glVertex2d(Sx, Sy);
}
```

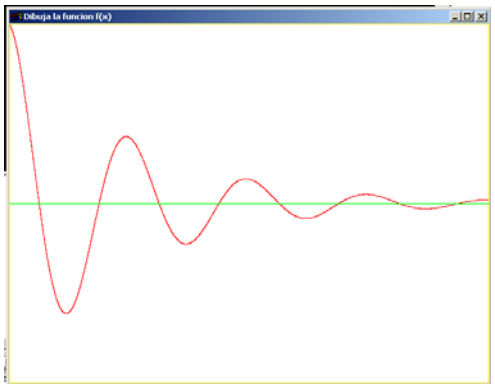

Figura 1. Salida gráfica de la practica 1.2

Representación de cualquier función

Considerando la representación de cualquier función f(.), como en el ejemplo anterior, donde la x varía entre xmin y xmax, y la función f(x) entre ymin e y max. Encontrar los valores apropiados A, B, C y D de la ecuación para generalizar el programa para cualquier ancho y alto de ventana deseado.

1.3 Representación mediante segmentos rectos de una función matemática

En el ejemplo anterior, se representraba una función f(x) frente x mediante una secuencia de puntos dados por la posición (x_i, $f(x_i)$). Una extensión en esta práctica consiste en unir los puntos mediante segmentos rectos que forman una polilínea. En este caso la función viene dada por:

$$f(x) = 300 - 100\cos(2\pi x/100) + 30\cos(4\pi x/100) + 6\cos(6\pi x/100)$$

donde x varía de 0 a 300 (incremento de x= 3). El proceso de dibujar la función con segmentos lineales es similar al que se ha utilizado anteriormente. Hay que escalar y desplazar apropiadamente la figura en la ventana. Esto requiere calcular las constantes A, B, C y D de la misma manera que se hizo antes.

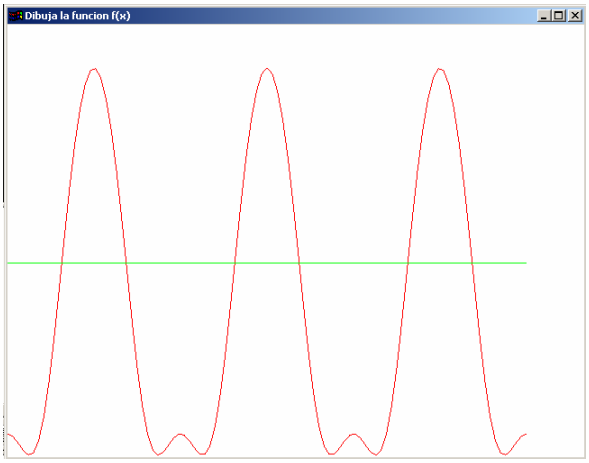

Figura 2. Salida gráfica de la practica 1.3

1.4 Representación mediante puntos el gráfico de Sierpinski

El gráfico de Sierpinski se muestra en la figura 3. Es una colección de puntos que se genera "*proceduralmente*", que significa que cada punto sucesivo que genera mediante una regla procedural. Aunque la regla es muy simple, el resultado final es un fractal.

Para generar el gráfico de Sierpinski utilizar una función llamada *drawDot()*, que será llamada varias veces al pasarle las posiciones de los puntos (x_0, y_0), $(x_1\ y_1$, $(x_2\ y_2)$, determinados por un algoritmo muy simple. Si llamamo $p_k = (x_k, y_k)$ a la posición del punto *k*. Cada punto se basa en el punto previo p_{k-1}. El proceso es el siguiente:

1. Se elijen tres punto iniciales T_0, T_1, T_2 que forman un triángulo.
2. Se elije un punto inicila p_0 que será uno de los T_0, T_1 o T_2 aleatoriamente.
   Ahora comienza el proceso iterativo hasta obtener el patrón deseado
3. Se elije un de los tres puntos T_0, T_1 o T_2 aleatoriamente; y se llama *T*.
4. Se calcula el siguiente punto p_k com el punto medio entre *T* y el previo encontrado p_{k-1}. Esto es p_k = *puntomed de* p_{k-1} y *T.*
5. Dibujar p_k usando *drawDot()*.

Es conveniente definir una clase simple, *GLintPoint*, para definir un punto con coordenadas enteras:

```
class GLintPoint{
public:
        GLint x, y;
};
```

Se inicializa la matriz con los tres punto iniciales. T[0], T[1], y T[2], que definin el triángulo inicial, ejemplo: *GLintPoint T[3]={ {50,60}, {340,80}, {240,350} };* (Usar una ventana de **400x400**). Es necesario almacenar cada p_k en la secuencia que se genera, puesto que lo necesitamos para generar el siguiente, lo almacenamos en *point*. En cada iteración, *point* se actualiza por un valor nuevo.

Utilizar *i = random(3)* para elegir uno de los puntos *T[i]* aleatoriamente. *random(3)* devuelve 0, 1, o 2. Esta función se define como

```
int random(int m)
{
       return rand()%m;
}
```

Nota: la función rand() devuelve un valor pseudoaleatorio en el rango 0 a 32767. El módulo se reduce a un valor entre 0 y 2.

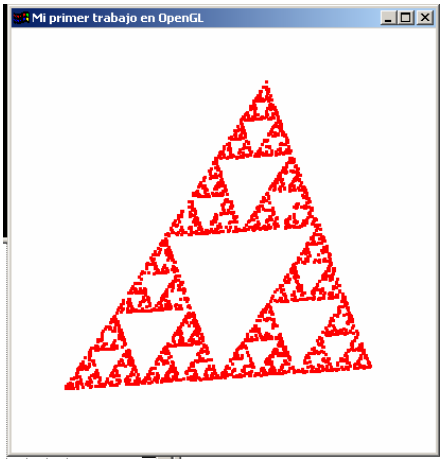

Figura 3. Salida gráfica de la practica 1.4

1.5 Dibujo de polilíneas almacenadas en un fichero externo

Realizar un programa de tal forma que represente en una ventana un archivo de polilíneas almacenadas en un fichero dado ***"dino.dat"***. Ver TEMA 2, apartado 10 pág. 21 a 23, la función drawPolyLineFile debe modificarse según se indica en la NOTA.

1.6 Rectángulos de tamaño y color aleatorios

Se trata de realizar un función encargada de representar rectángulos alineados, utilizando la función de OpenGL glRecti(), de tal forma que tanto las coordenadas de los extremos y el color de relleno de generen de una forma aleatoria. La función debe pasar los argumentos siguiente:

dibujaRectangulos (int num, float numColor, int Ancho, int Alto);

Siendo *num* el número de rectángulos que se quieren generar.
*numColo*r el color de fondo de la pantalla (1 blanco, 0 negro).
Ancho y Alto los datos de partida para generar los valores aleatorios de x1, x2, y1, y2.
Los valores de x1, x2 se obtiene mediante la función random(Ancho) y los valores de y1, y2 de la función random(Alto). La función random se describe en la práctica 1.4.
De igualmanera el color a utilizar se obtendrá a partir de la función random como sigue:

```
. . . . .
GLfloat lev = random(10)/10.0; //valor aleatorio entre 0 y 1
glColor3f(lev, lev, lev);
. . . . .
```

1.7 Dibujo el tablero de ajedrez

Escribir una función ajedrez(int size) que dibuje el tablero de ajedrez como se ve en la figura 4. Situar el tablero con la esquina inferior izquierde en el (0, 0). Siendo el ancho de l a ventana abierta de 400x400 pixels. Cada cuadrado debe tener el tamaño del argumento *size*. Elegir dos colores adecuados.

Solución: Las coordenadas de los ij cuadrados serán (i*size, j*size, i*size+size, j*size+size). El color debe ir alternado entre (r_1, g_1, b_1) y (r_2, g_2, b_2) usando el siguiente código:

```
if ((i +j)%2 ==0) // si i+j es par
       glColor3f(r1, g1, b1);
else
       glColor3f(r2, g2, b2);
```

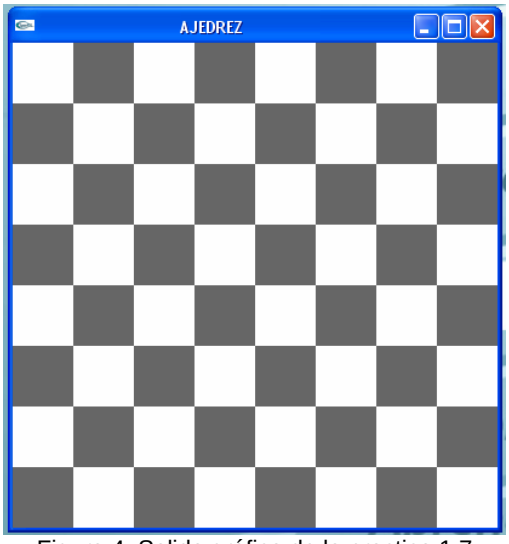

Figura 4. Salida gráfica de la practica 1.7

1.8 Dibujo de líneas con patrones

Dibujar líneas horiznotales con diferentes patrones de líneas y factor de escala. Como ejemplo usar los patrones de línea mostrador en el apartado 9 del tema 2. Un total de 9 líneas separadas 20 unidades en *y*, los valores de *x* comprendidos entre –80 y 80 .Utilizar una ventana de 400x400.

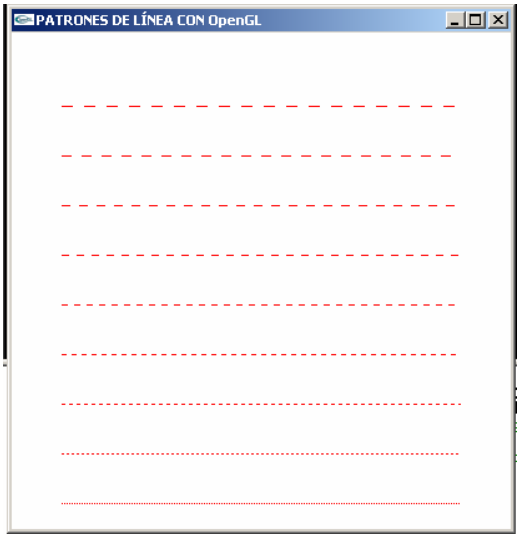

Figura 5. Ejemplo de salida gráfica de la practica 1.8

1.9 Dibujo de líneas con diferentes grosores

Dibujar líneas horizontales con diferentes grosores. Siguiendo el mismo criterio para el tamaño de ventana y la variación de las coordenadas *x* e *y*. Utilizar una ventana de 400x400.

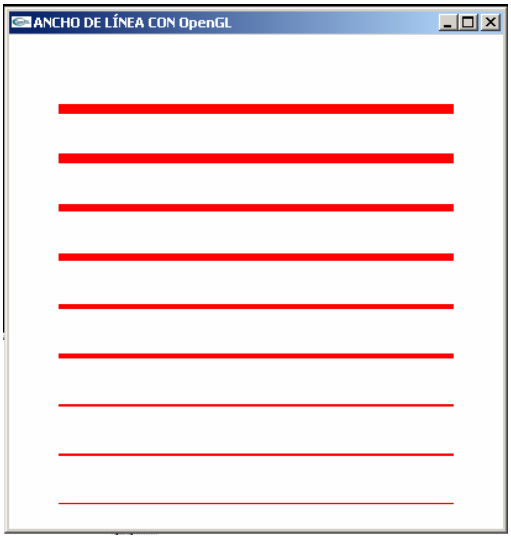

Figura 6. Ejemplo de salida gráfica de la practica 1.9.

1.10 Dibujo de polilíneas, marcadores y caracteres de mapas

Se trata de dibujar un conjunto de polilíneas, un conjunto de polimarcadores y de etiquetas basadas en caracteres de mapa de bits para generar un gráfico lineal que representa una serie de datos mensuales a lo largo de un periodo de un año.
Los valores de los datos son: 420, 342, 324, 310, 262, 185, 190, 196, 217, 240, 312, 438.
La ventana gráfica será de 600x500 pixel y la salida como la que se muestra en la figura 7.

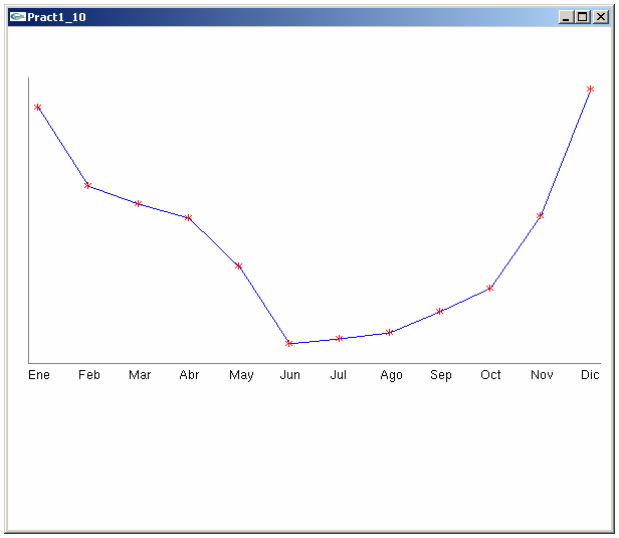

Figura 7. Ejemplo de salida gráfica de la practica 1.10.

1.11 Dibujo de polilíneas, marcadores y caracteres de mapas

Con el mismo conjunto de datos dibujar un diagrama de barras como se muestra en la figura 8. Para dibujar las barras utilizar la función glRecti.

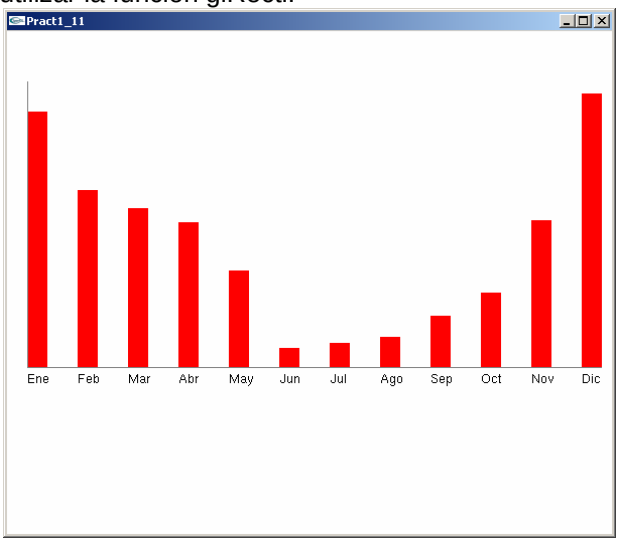

Figura 8. Ejemplo de salida gráfica de la practica 1.11.

1.12 Dibujo de polilíneas, marcadores y caracteres de mapas

Utilizando las funciones para añadir menús y submenús asociados emergentes simples que disponen las librerías GLUT, para los dispositivos de entrada. Realizar un programa que muestre la salida de la práctica 1.10 o 1.11 al seleccionarlo con el menú emergente.