Práctica 3. Uso de módulos y TADs

Estructuras de Datos

Objetivo:

Vamos a continuar con el uso de archivos de código y de cabecera, incluyendo nuestra implementación de los números complejos en un nuevo programa.

Funcionalidad básica del sistema gráfico:

En esta ocasión vamos a utilizar un código que tenéis disponible junto a este guion (Bitmap.h y Bitmap.c) y permite manejar ficheros de maps de bits (.bmp). Los ficheros bmp tienen una estructura estándar, con una cabecera de tamaño fijo (54 bytes), seguida por los bytes en los que se almacenan los colores de los píxeles, sin compresión. Esto nos permite modificar fácilmente imágenes existentes, o como en este caso, crear nuevas imágenes.

En particular, vamos a generar una imagen pixel a pixel para terminar creando el fractal de Mandelbrot mediante nuestra unidad de números complejos de la práctica anterior.Como comienzo vamos a probar el código generando una imagen que muestre un gradiente de color.

```c
#include <stdio.h>
#include "Bitmap.h"
int main()
{
  int i, j;
  int ancho = 640;
  int alto = 480;
  Bitmap gradiente = createBitmap("gradiente.bmp", ancho, alto, 3);
  for(i = 0; i < ancho; i++) {
    for(j = 0; j < alto; j++) {
      putPixel(&gradiente, i, j, (i/10)% 64);
    }
  }
  saveBitmap(&gradiente);
}
```

Figura 1: Ejemplo de uso de Bitmap.h

El programa debería crear un fichero “gradiente.bmp” en la carpeta en la que se haya ejecutado el código. La función createBitmap necesita 4 argumentos, el nombre del fichero que se usará para guardar el mapa de bits, el ancho de la imagen en pixeles, el alto de la imagen en pixeles, y la profundidad de color. Este último parámetro puede valer 1 para imágenes en escala de grises o 3 para imágenes con 24 bits de color (RGB).

La función putPixel necesita un puntero a un mapa de bits, las coordenadas del pixel que vamos a modificar, y tres valores enteros en el rango 0 a 255, para las intensidades de los colores rojo, verde y azul respectivamente. El color rojo utilizará 255,0,0, el amarillo 255,255,0, el verde 0,255,0 …

Finalmente, después de modificar el bitmap, la función saveBitmap recibe un puntero a bitmap y escribe el mapa de bits generado en un fichero.

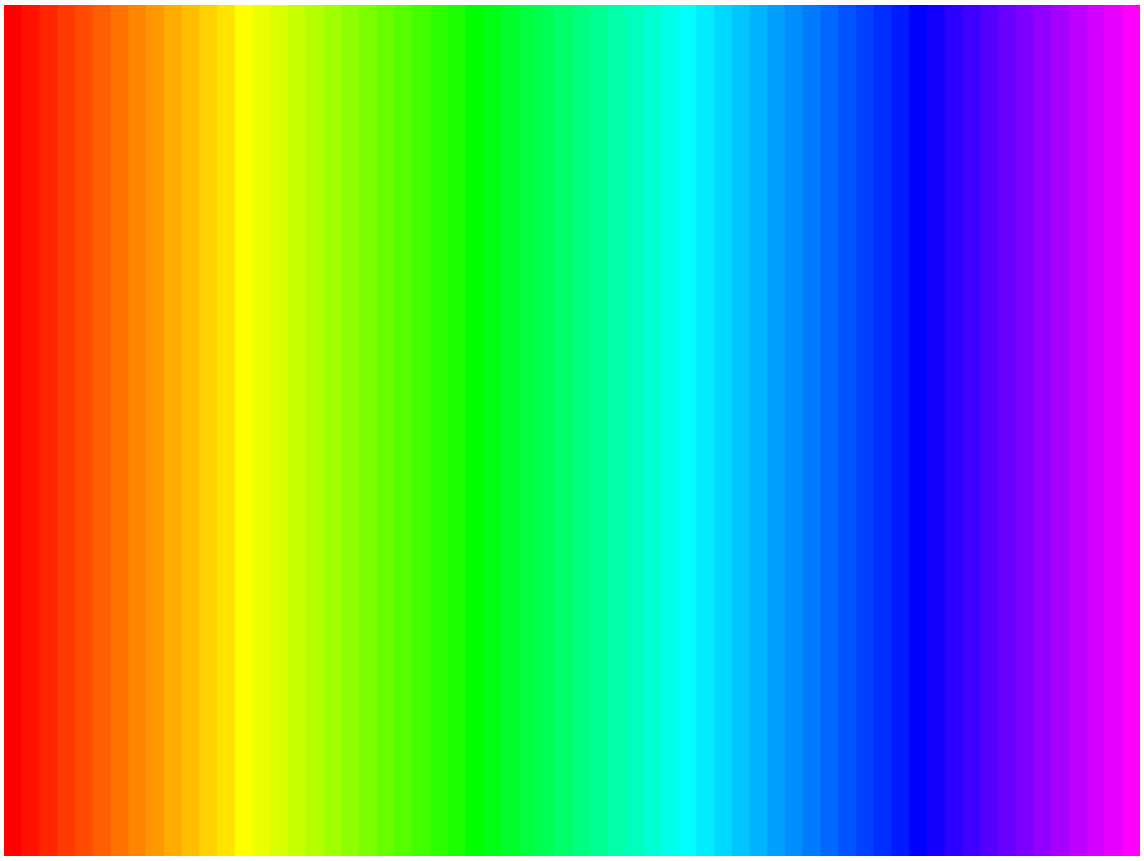

Figura 2: Imagen producida por la ejecución del código de la Figura 1.

Conjunto de Mandelbrot:

El conjunto de Mandelbrot está considerado como el objeto geométrico más complicado creado hasta el momento por el hombre. La frontera que delimita este objeto en el plano complejo es fractal (patrón que se repite en cada escala). Su nombre se debe al matemático Benoît Mandelbrot (1924-2010), que investigó sobre él en los años setenta.

El conjunto de Mandelbrot se puede definir en el plano complejo de la siguiente manera. Llamemos c a un número complejo cualquiera. Con ese valor de c, se construye la siguiente sucesión:

$$\begin{cases} z_0 = 0 & n = 0 \\ z_{n+1} = z_n^2 + c & n > 0 \end{cases}$$

donde n hace referencia al **número de iteración**. Veamos algunos ejemplos:

- Supongamos que c = 1 + 0i = 1. Para ese caso, los primeros términos de la sucesión de Mandelbrot son:

$$z_0 = 0$$
$$z_1 = z_0^2 + c = 0 + 1 = 1$$
$$z_2 = z_1^2 + c = 1^2 + 1 = 2$$
$$z_3 = z_2^2 + c = 2^2 + 1 = 5$$
$$z_4 = z_3^2 + c = 5^2 + 1 = 26$$
…

- Para $c = -1 + 0i = -1$, la sucesión de Mandelbrot es:

$$z_0 = 0$$
$$z_1 = z_0^2 + c = 0 - 1 = -1$$
$$z_2 = z_1^2 + c = (-1)^2 - 1 = 0$$
$$z_3 = z_2^2 + c = 0^2 - 1 = -1$$
$$z_4 = z_3^2 + c = (-1)^2 - 1 = 0$$
…

- La sucesión de Mandelbrot para $c = -0.1 + 0.1i$ es:

$$z_0 = 0$$
$$z_1 = z_0^2 + c = 0^2 - 0.1 + 0.1i = -0.1 + 0.1i$$
$$z_2 = z_1^2 + c = -0.1 + 0.1i^2 - 0.1 + 0.1i = -0.1 + 0.08i$$
$$z_3 = z_2^2 + c = -0.1 + 0.08i^2 - 0.1 + 0.1i = -0.0964 + 0.084i$$
$$z_4 = z_3^2 + c = -0.0964 + 0.084i^2 - 0.1 + 0.1i = -0.097763 + 0.083805i$$
…

Si esta sucesión queda acotada, entonces se dice que c pertenece al conjunto de Mandelbrot, y si no, c no pertenece. Como se puede intuir en los ejemplos anteriores, $c = 1 + 0i$ no es un elemento del conjunto de Mandelbrot, puesto que su sucesión no parece estar acotada, mientras que $c = -1 + 0i$ y $c = -0.1 + 0.1i$ sí pertenecen al conjunto de Mandelbrot, ya que sus sucesiones sí parecen estar acotadas.

A menudo se visualiza el conjunto mediante el algoritmo de tiempo de escape, que asigna un color a cada punto complejo en el espacio 2D en función de la iteración en la que se detecta su divergencia.

En la imagen mostrada en la Figura 3 se representa dicho tiempo de escape con un color diferente en cada punto del plano complejo 2D. Como se puede observar, aparecen regiones de colores diferentes, que indican que los tiempos de escape son diferentes en cada caso. Por ejemplo, la región roja exterior se corresponde con valores que han divergido en la primera iteración, mientras que los puntos de la región central coloreados de negro no presentan una sucesión divergente tras calcular 64 términos de la sucesión.

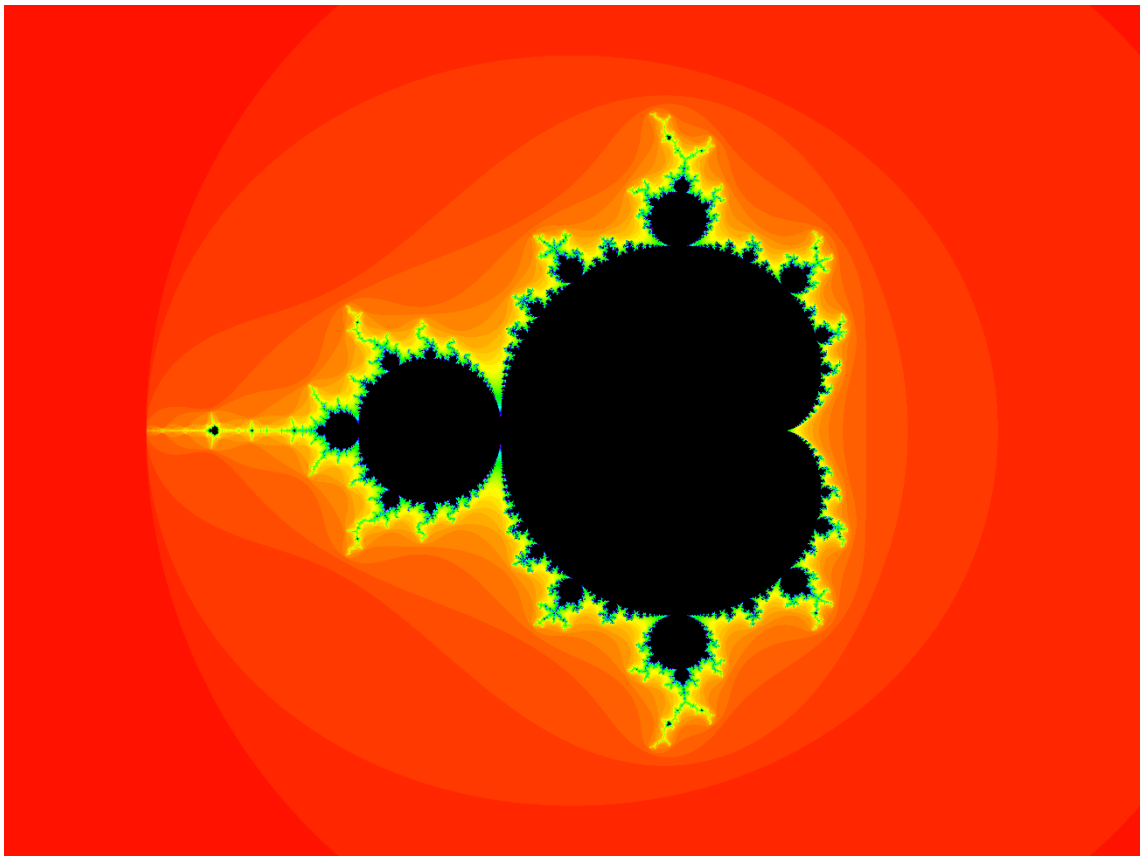

Figura 3: Conjunto de Mandelbrot para los números complejos en el rango [-2.5-1.5i, 1.5+1.5i].

¿Cómo saber entonces si una sucesión es divergente? Se sabe que los puntos del plano complejo cuyo módulo es superior a 2 no pertenecen al conjunto de Mandelbrot. Por lo tanto, basta encontrar un solo término de la sucesión que verifique que $|zn|$ > 2 para estar seguro de que c no está en el conjunto. Por tanto, si el valor del módulo pasada alguna iteración de cálculo es mayor que 2 se le asigna el número de iteración al color del punto.

Y ese va a ser el objetivo de esta práctica. Utilizando la unidad de los números complejos ya realizada en clase y las indicaciones dadas en este enunciado, construir un programa que represente en una imagen de 640x480 píxeles la velocidad de divergencia para cada punto del plano complejo entre los valores [−2.5 − 1.5i, 1.5 + 1.5i]. Como queremos dibujar en el rango de valores [−2.5,1.5] en el eje real (4 unidades de anchura), y [−1.5i, 1.5i] en el eje imaginario (3 unidades de altura), tenemos que asociar a cada píxel el número complejo que va a representar. Sabiendo que el origen de coordenadas de la imagen (0,0) es la esquina superior izquierda, tenemos que a cada píxel de la imagen en posición [xs, ys] le corresponde el número complejo (xs/160 − 2.5) + (ys/160 − 1.5) i (considerando la proporción 640/4=160 y 480/3=160). A continuación, hay que calcular la velocidad de escape para cada uno de esos números complejos. Esto se hace calculando su sucesión hasta llegar al número máximo de iteraciones predefinido (nosotros calcularemos un máximo de 64 iteraciones) o hasta comprobar que diverge (si alguno de los términos calculados tiene un módulo mayor que 2, esa sucesión es divergente). Finalmente, se colorea el píxel correspondiente al valor complejo con el valor de la máxima iteración alcanzada.