Fundamentos de Programación

Doble Grado en Ingeniería Informática y Matemáticas
Proyecto Informático con imágenes PGM de texto

1. Introducción

Una *imagen* es una representación visual de un objeto. En Informática, por imagen, se entiende un archivo codificado que, al abrirlo, muestra una representación visual de algo. El objetivo de este

Figura 1: Archivo monigote.jpg

pequeño proyecto consistirá en diseñar e implementar un programa para manipulación básica de imágenes en escala de grises. Estas imágenes se almacenarán en ficheros gráficos muy simples de extensión `.pgm`. Con esto se pretende:

- que el alumno resuelva un problema en el que es necesaria la utilización de vectores y/o matrices.
- que trate de dividir el código en módulos para trabajar en equipo.

2. Abstracción de una imagen

Para simplificar, consideraremos que las imágenes son en escala de grises, esto es, cada pixel es de un color gris cuya luminosidad variará entre blanco (la mayor luminosidad) y negro (la menor luminosidad). Matemáticamente, una imagen quedaría definida mediante una función:

$$\text{imagen} = f : \mathbb{R} \times \mathbb{R} \to [0, \infty)$$

Es decir, una imagen queda definida como una asignación de un color (una luminosidad, dada por un valor real) a cada punto del plano. Sin embargo, en un ordenador no podemos representar imágenes de tamaño infinito ni tampoco un subconjunto no discreto del plano. Además, sería recomendable que los posibles colores fueran un número finito. Por lo tanto la representación de la imagen deberá ser una función

$$\text{imagen} = f : \{0, 1, 2, \ldots, n\} \times \{0, 1, 2, \ldots, m\} \to \{0, 1, 2, \ldots, 256\}$$

Así pues, una imagen digital de niveles de gris puede verse como una matriz bidimensional de puntos (píxeles, en este contexto) en la que cada uno tiene asociado un nivel de luminosidad cuyos valores están en el conjunto $\{0, 1, ..., 255\}$ de forma que el 0 indica mínima luminosidad (negro) y el 255 la máxima luminosidad (blanco). Los restantes valores indican niveles intermedios de luminosidad (grises), siendo más oscuros cuanto menor sea su valor. Con esta gama de colores entre 0 y 255 (es decir, con esta representación) cada píxel requiere únicamente un byte, es decir, podíamos representarlo por un dato de tipo `unsigned char` (que toma valores 0 entre 255). Sin embargo, para simplificar, trataremos cada pixel como un dato de tipo `int` o `short int`. Si

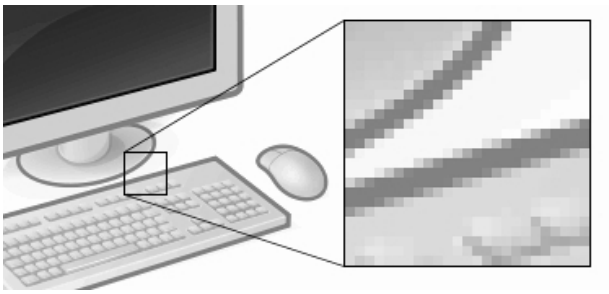

Figura 2: Imagen y ampliación para apreciar los píxeles, fuente: Wikipedia

se quiere utilizar `unsigned char`, téngase en cuenta que, al mostrarlo por pantalla con `cout`, muestra un carácter, por lo que sería necesario un casting a `int` o `short int`.

Las imágenes en color se representan de forma diferente ya que un color no puede representarse usando un único valor, sino que se deben incluir tres números. Existen múltiples propuestas sobre el rango de valores y el significado de cada una de esas componentes. De las más utilizadas es el modelo RGB, donde cada color se representa como como la suma de las tres componentes de una terna: Rojo (en inglés, Red), Verde (en inglés Green) y Azul (en inglés, Blue). De igual manera que con las imágenes en escala de grises, es habitual asignarle el rango de números enteros [0, 255], ya que permite representar cada componente con un único byte, y la variedad de posibles colores es suficientemente amplia.

3. Ficheros PGM

Para almacenar las imágenes utilizaremos ficheros de texto con extensión `.pgm` (existe otra version de ficheros `.pgm` que son binarios, que todavía no podemos manipular). PGM es el acrónimo de *Portable Gray Map file format*. Una imagen PGM almacena una imagen de niveles de gris, es decir, un valor numérico que indica la luminosidad para cada píxel. Un valor de luminosidad viene determinado por un número entero en el rango [0,255]. Los ficheros PGM están compuestos de una cabecera y una secuencia de enteros que representan la matriz de pixels de la imagen que contiene. Concretamente, la cabecera tiene el siguiente formato:

- Una cadena mágica (cadena de caracteres) que, para archivos `.pgm` de texto, es `P2`.
- Un separador (un salto de línea).
- El ancho o columnas de la imagen (decimal y formato ASCII), es decir, el número de píxeles de una fila.
- Un separador (espacio, tabulador o salto de linea) que, para unificar criterios, será un espacio en blanco.
- El alto o filas de la imagen (decimal y formato ASCII). Es decir, el número de píxeles de una columna.
- Un separador (espacio, tabulador o salto de linea) que, para unificar criterios, será un salto de linea.
- El valor máximo de gris (decimal y formato ASCII). Indica que el rango de valores para cada valor de luminosidad es [0,M], 0 correspondiente a negro y M a blanco. Para simplificar, $M = 255$ siempre.

- Un único separador (espacio, tabulador o salto de linea) que, para unificar criterios, será un salto de linea.

A continuación está escrito el contenido de la imagen, es decir, una matriz `filas x columnas` de valores enteros (cada fila de la matriz en una línea, cada pixel de una fila separado del siguiente pixel por un espacio en blanco). Cada uno de estos valores representa el valor de luminosidad de un pixel. La Figura 3 muestra el formato de un posible fichero `.pgm` de texto.

```
P2
10 10
255
0 0 12 50 130 12 12 15 255 255
0 0 50 12 130 12 12 15 255 255
0 0 12 50 130 12 12 15 255 255
0 0 50 12 130 12 12 15 255 255
0 0 12 50 130 12 12 15 255 255
0 0 50 12 130 12 12 15 255 255
0 0 12 50 130 12 12 15 255 255
0 0 50 12 130 12 12 15 255 255
0 0 12 50 130 12 12 15 255 255
0 0 50 12 130 12 12 15 255 255
```

Figura 3: Fichero PGM de texto

En la Figura 4 podemos observar cómo se vería la imagen a través de un visor de ficheros PGM.

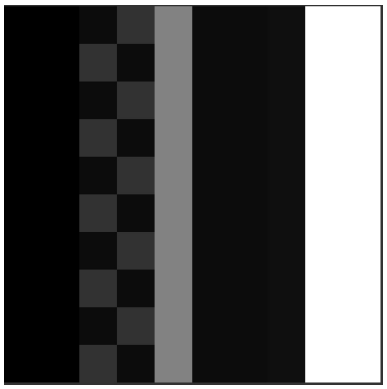

Figura 4: Imagen PGM de texto

Como visor de este tipo de archivos podemos utilizar los editores de imágenes *GIMP* (Linux) o *Photoshop* (Windows). Una aplicación gratuita para Windows es *Brennig's*. En MacOS, se reconocen los ficheros como imagen inmediatamente y pueden ser abiertos por *Vista previa*.

4. Implementaciones a realizar

4.1. Almacenamiento en memoria

En base a lo explicado en las secciones anteriores, necesitamos un tipo de dato para almacenar lo que se lea desde un fichero `.pgm`. Lo más lógico sería un dato de tipo `struct` (más adelante

veremos que lo mejor es una clase) que debería constar, al menos, con datos miembro para almacenar las dimensiones de una matriz de pixels de enteros, y la propia matriz.

Nota 4.1. *En C++ se puede dar un alias (otro nombre) a los tipos de datos. Por ejemplo, aunque la matriz sea de tipo* `int`*, ya que los enteros representan grados de color, se podría utilizar la palabra* `color` *para designar este tipo de enteros en particular. Para explicar al compilador que la palabra* `color` *es un entero camuflado con otro nombre, utilizamos el operador* `typedef`*. Por ejemplo, en el preámbulo, después del espacio de nombres, escribimos*

```
typedef short int color;
```

Y a partir de ese momento, cuando declaremos un tipo de dato `color`*, el compilador entenderá* `short int`*. Entonces una matriz de píxeles se podría declarar como*

```
color matriz_pixels[n][m];
```

Esto no significa que no se pueda utilizar `short int`*, sino que también se puede utilizar* `color` *para referirse a un* `short int`*.*

4.2. Lectura/escritura de ficheros PGM

Puesto que el tema de lectura y escritura en ficheros con C++ no se explica en esta asignatura, se trabajará con la redirección de la entrada/salida para lectura/escritura de los ficheros `pgm`.

Esto quiere decir que, para leer el fichero, el código deberá hacerse como si se le pidiera al usuario los datos de un fichero PGM por teclado (primero la cadena mágica, segundo las dimensiones de la imagen, etc.). De igual manera, para escribir en el fichero, se implementará para mostrar por pantalla los datos de un fichero PGM en el orden y forma correctos. Y a la hora de ejecutar cualquier programa, debe hacerse desde el terminal con la orden

```
programa.exe < PGM_entrada.pgm > PGM_salida.pgm
```

Por otro lado, recordemos que no sabemos reservar memoria en tiempo de ejecución (cuando nos dan las dimensiones de la imagen). Así que tenemos que reservar memoria suficiente a priori y esperar que las dimensiones de la imagen no superen ese máximo reservado. Para especificar las dimensiones máximas de una imagen utilizaremos dos constantes globales (declaradas después del espacio de nombres

```
const int MAX_FILAS = 650;
const int MAX_COLUMNAS = 650;
```

Con estas dimensiones, las imágenes de ejemplo de PRADO se pueden almacenar en memoria.

Nota 4.2. *Es posible que al tratar ciertas imágenes "grandes", cuando se ejecute alguno de los programas (depende de la implementación), se obtenga un mensaje de* `segmentation fault: 11`*. Esto es debido a que la reserva de memoria en la pila es algo limitada y no ha podido reservar tanta memoria para almacenar la imagen. Para aumentar la memoria de la pila en DevC++, ir a* `Herramientas -> Opciones del Compilador`*. En la pestaña* `Compilador`*, marcamos* `Añadir los siguientes comandos al llamar al compilador` *y escribimos en el recuadro* `-Wl,--stack,21000000` *donde el número 21000000 es el tamaño de la pila.*

Esto se podrá mejorar en el segundo semestre, al utilizar memoria dinámica, ya que esta no se reserva en la pila (stack). La memoria dinámica se reserva en el montón (heap), y allí se puede reservar una cantidad mayor de memoria. La solución "definitiva" es utilizar ficheros, lo que también aprenderéis en Metodología de la Programación*.*

4.3. Acciones con ficheros PGM

Hay que implementar cuatro programas que realicen las siguientes funciones:

- **Cambiar una imagen a blanco** (`blanquear.exe`). Este programa debe dar a cada pixel de la imagen el color blanco (si no os gusta el blanco, podéis elegir cualquier otro color en la escala de grises de 0 a 255).
- **Rotar una imagen** (`rotar.exe`). Este programa rotará la imagen 90º grados a la derecha.
- **Maximizar contraste** (`contraste.exe`). Este programa cambiará los colores menores de 125 a 0, y los mayores o iguales, a 255.
- **Calcular negativo** (`negativo.exe`). Este programa cambiara un pixel con valor x a $255-x$ (en el caso extremo, $x = 0$ o $x = 255$, cambia blanco por negro y negro por blanco).

5. En resumen, pasos a seguir

1. Formar un equipo de 2 programadores (o uno, si se quiere hacer en solitario)
2. Con un lápiz y un papel, diseñar el proyecto. ¿En qué partes se divide el proyecto? ¿cómo enlazan unas partes con otras? ¿qué datos necesita cada parte y qué datos devuelve? ¿qué funciones necesita cada parte?
3. Dividir el trabajo entre los programadores, y que cada uno implemente su parte.
4. Juntar el código en archivos .cpp que contengan los programas de la sección anterior. Obtener un prototipo de los programas.
5. Verificar su funcionamiento (depurar errores si no funciona todo lo bien que uno quisiera).
6. Escribir una memoria contando el diseño general realizado (una hoja como mucho) y, cada uno de los programadores, qué ha hecho en su parte (una hoja como mucho por programador)
7. Subirlo a PRADO, antes de que termine 2019.