Julián Garrido Arana y Juan Valentín Guerrero Cano

PRACTICA 2 — T.D.A. IMAGEN

1. PROGRAMACIÓN:

Para realizar nuestra práctica, en primer lugar, hemos creado y completado los ficheros de la clase Imagen: imagen.h y imagen.cpp (documentados con doxygen).

Con respecto a los ejercicios propuestos, hemos escogido el 1. (Umbralizar con escala de grises), el 3. (Zoom), el 5. (Contraste) y el 6. (Morphing) .

Para realizar estas 4 funciones, hemos creado dos ficheros extra: funciones.h y funciones.cpp donde declaramos e implementamos las mismas respectivamente. También hemos creado un fichero adicional main.cpp para probar la clase y nuestras funciones generando así un programa enfocado a la transformación y modificación de imágenes. Además hemos añadido dos funciones extra a los ficheros imagenES.h e imagenES.cpp.

-imagen.h y imagen.cpp :

Aquí hemos declarado e implementado los métodos básicos para trabajar con los tda imagen tal y como pedía el guión de la práctica 2.

En particular, hemos incluido el método flat() y otro constructor por parámetros (vector 1D) ademas del de filas y columnas. (Indicados en la orientación por correo).

El método flat() permite transformar nuestra matriz de píxeles (2D) en un vector que contiene los valores de los mismos de manera lineal (1D).

El constructor mencionado Imagen(int f, int c, const byte *vector) genera una objeto imagen (matriz píxeles 2D) a partir de un vector 1D con los valores de los píxeles.

-funciones.h y funciones.cpp :

Hemos optado por que las funciones aquí contenidas devuelvan un dato de tipo entero (int) que más tarde aprovechará el main para determinar un posible error y su tipo dentro de cada una. Para ello, utilizamos una estructura con condicionales anidados y un booleano (correcto) en cada una de ellas para ir comprobando los fallos potenciales como errores de lectura, error en los parámetros, errores en la escritura (salida de imagen), error de formato de la imagen y errores particulares de cada función.

Algunas de estas funciones incluyen comentarios en la implementación para facilitar la comprensión de los pasos que siguen.

-main.cpp:

Este es nuestro fichero principal donde hemos querido generar un programa interactivo con el usuario por teclado. Recibimos por teclado los nombres de los ficheros con los que el usuario quiere trabajar, y mediante un menú (realizado por un switch) el usuario elegirá qué función aplicar a la imagen contenida en los ficheros mencionados previamente. En función de su selección, se le pedirá al usuario que introduzca nuevos parámetros específicos de las funciones.

Como explicamos en el apartado anterior, las funciones creadas para modificar y transformar las imágenes devolverán un dato de tipo entero que mediante el módulo creado en el main (errorCode) podremos concretar el tipo de error que ha ocurrido en esa función elegida. Los tipos de errores propuestos para las funciones son :

NO_ERROR (0): La función no ha tenido ningún error interno
ERROR_PARAMETROS (1): Los parámetros introducidos por el usuario no son correctos (no se ajustan a las condiciones establecidas)
ERROR_LEER (2): Error en la lectura de la imagen del fichero de entrada
ERROR_ESCRIBIR (3): Error en la escritura de la imagen en el fichero de salida
ERROR_FORMATO (4): La imagen con la que se trabaja no está en el formato correspondiente (PGM)
ERROR_CONTRASTE: La imagen introducida solo tiene un tono de gris
ERROR_MORPHING: Las dos imágenes introducidas tienen dimensiones diferentes y/o las dos imágenes introducidas son de tipo (formatos) diferentes (PPM/PGM/...)

2. MAKEFILE:

Siguiendo las instrucciones de las diapositivas, hemos elaborado un makefile (contenido en un archivo de texto) que como objetivo principal tiene la creación del ejecutable *main*. Para ello ponemos este objetivo como primera orden añadiendo las dependencias correspondientes y más adelante añadimos los objetivos necesarios para generar también todas las dependencias necesarias (los .o) incluyendo cada fichero de cabecera donde es necesario.

Por último añadimos tres últimos objetivos para poder compilar tantas veces como queramos el programa (clean y rebuild) y para generar la documentación (documentacion).

3. DOCUMENTACIÓN:

Para realizar la documentación de nuestro proyecto hemos utilizado doxygen, comentando cada uno de los archivos mencionados en el apartado “programación” de forma adecuada.

Primero hemos ejecutado la orden doxygen -g para crear un archivo de configuración para la documentación estándar, tal y como se aconseja en las transparencias. Hemos modificado una serie de opciones en dicho documento, como EXTRACT_ALL, OUTPUT_LANGUAGE, EXCLUDE (para evitar que se genere la documentación de carpetas no relacionadas con el código como son las que contienen a las imágenes y a los resultados de las pruebas realizadas con el main), EXTRACT_PRIVATE, SORT_MEMBER_DOCS, OUTPUT_DIRECTORY (/doc) entre otras…

Los tipos de documentos externos al código que hemos generado son Latex, HTML y docbooks tal y como se puede observar en la carpeta doc de nuestro proyecto.

4. ESTRUCTURA DE PROYECTO Y PRUEBAS:

Nuestro proyecto sigue la estructura de directorios vista en clase (src, inc, lib…) donde almacenamos los distintos archivos según su tipo. Además hemos incluido los directorios *results* y *figures*. El primero está subdividido en otros directorios (umbralizar, zoom, contraste y morphing) donde hemos almacenado las pruebas realizadas de cada una de nuestras funciones. Por otro lado

figures contiene varias imágenes en diferentes formatos (PGM y PPM) con las que hacer uso del programa.

El directorio lib no lo hemos usado ya que no hemos generado ninguna biblioteca, pero lo incluimos para mantener la estructura adecuada (estándar) de proyecto. También incluimos tres archivos independientes (fuera de los directorios) que corresponden al makefile (txt), al doxyfile (txt) y al pdf explicativo (pdf).

Nuestro programa se puede probar con cualquier imagen (del directorio figures o no) que esté en formato PGM. Pese a tener intención, finalmente no hemos trabajado con imágenes PPM.