UNIVERSIDAD DE GRANADA

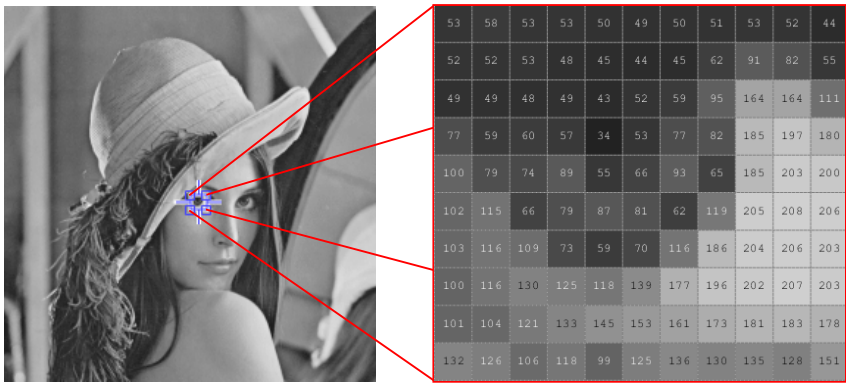
ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

DGIM

Curso 2021/2022



Guion de prácticas

Manejo de imágenes digitales

Febrero de 2022

Índice

1. Objetivos

El desarrollo de esta práctica pretende servir a los siguientes objetivos:

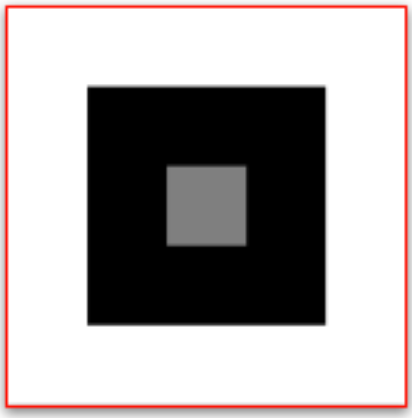
- practicar el paso de parámetros por referencia
- practicar el paso de parámetros de tipo array
- creación de bibliotecas

2. Imágenes

Para imágenes en blanco y negro, cada píxel se suele representar con un byte[1] (8 bits). El valor del píxel representa su tonalidad de gris que va desde el negro (0) hasta el blanco (255). Un píxel con valor 128 tendrá un gris intermedio entre blanco y negro. En la siguiente imagen se puede observar el valor de los píxeles para una pequeña porción de una imagen extraida de un conocido ejemplo (Lena) de procesamiento digital de imágenes..

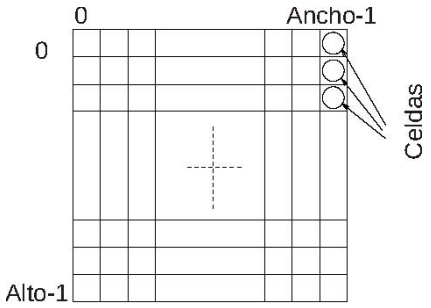
o esta otra imagen de 5x5 píxeles

255	255	255	255	255
255	0	0	0	255
255	0	128	0	255
255	0	0	0	255
255	255	255	255	255

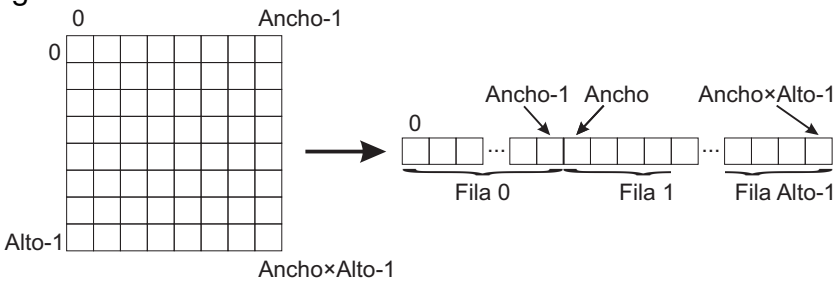


Desde un punto de vista práctico, una imagen se puede considerar como una matriz bidimensional de celdas. Cada celda de la matriz almacena la información de un píxel.

[1] Recuerde que en C++ un ”unsigned char” almacena exactamente un byte.



Pese a que una imagen se trata habitualmente como una matriz bidimensional de bytes, es usual representarla internamente como un vector en el que las filas se van guardando una tras otra, almacenando consecutivos todos los bytes de la imagen. Así, la posición 0 del vector tendrá el píxel de la esquina superior izquierda, la posición 1 el de su derecha, y así hasta el píxel de la esquina inferior derecha, como se muestra en la figura siguiente:



Así se puede acceder fácilmente a las posiciones de la imagen de forma consecutiva pero, para acceder a cada píxel (x, y) de la imagen es necesario convertir las coordenadas de imagen (x, y) en la coordenada de vector (i). Para ello se aplicará la siguiente fórmula:

$$i = y * Ancho + x.$$

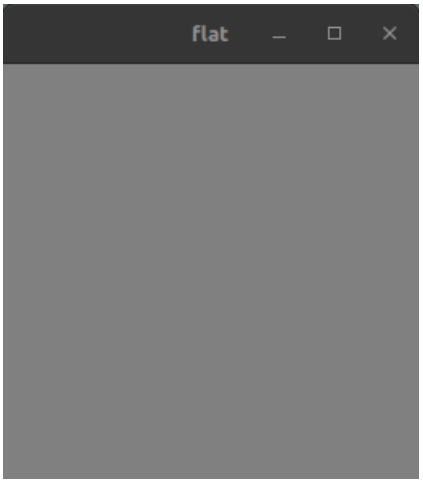
2.1. Image

```
1  /**
2  @file Image.h
3  @brief Manejo de imÃ¡genes digitales en formato PGM blanco y negro
4  @author MP-DGIM - Grupo A
5   **/
6
7  #ifndef _IMAGE_H_
8  #define _IMAGE_H_
9
10 #include <istream>
11 #include <fstream>
12 #include "Byte.h"
13
14 #define IMAGE_MAX_SIZE 200000 ///< Max number of bytes allowd for
15 #define IMAGE_DISK_OK 0 ///< Image read/write successful
16 #define IMAGE_ERROR_OPEN 1 ///< Error opening the file
17 #define IMAGE_ERROR_DATA 2 ///< Missing data in the file
18 #define IMAGE_ERROR_FORMAT 3 ///< Unknown image format
19 #define IMAGE_TOO_LARGE 4 ///< The image is too large and does not fit into memory
20 /**
21 @brief A black and white image
22  */
23 class Image {
24 private:
25     Byte _data[IMAGE_MAX_SIZE]; ///< Bytes of the image
26     int _height; ///< number of rows
27     int _width; ///< number of columms
28 public:
29     /**
30      * @brief It builds an empty, image
31      */
32     Image();
33     /**
34      * @brief It builds a fully black image with @a width columns and @a height rows
35      * @param height number of rows
36      * @param width number of columns
37      */
38     Image(int width, int height);
39     /**
40     @brief It gives the number of rows of the image
41     @return number of rows
42      */
43     int height() const;
44     /**
45     @brief It gives the number of columns of the image
46     @return The number of rows
47      */
48     int width() const;
49     /**
50      * @brief It assigns the value @a v to the position(x,y) of the image.It must check that
51      * the values x and y are valid, otherwise, it does not do anythig.
52      * @param x The column
53      * @param y the row
54      * @param v The new value
55      */
56     void setPixel(int x, int y, Byte v);
57     /**
58      * @brief It returns the value of the requested (x,y) position. It must check that
59      * the values x and y are valid, otherwise, it returns a negative value. Please note that
60      * the value returned is a int
61      * @param x The column
62      * @param y the row
63      * @return The value of the pixel in [0-256] or -1 if there is an access error
64      */
65     int getPixel(int x, int y) const;
66     /**
67      * @brief It assigns the value @a v to the linear position i of the image. It must check that
68      * the values i is valid, otherwise, it does not do anythig.
69      * @param i The linear position
70      * @param v The new value
71      */
72     void setPos(int i, Byte v);
73     /**
74      * @brief It returns the value of the requested linear position. It must check that
75      * the value i is valid, otherwise, it returns a negative value. Please note that
76      * the value returned is a int
77      * @param i The linear position
78      * @return The value of the pixel in [0-256] or -1 if there is an access error
79      */
80     int getPos(int i) const;
81     /**
82      * @brief  It sets all pixels of the image to the value given
83      * @param b The value
84      */
85     void flatten(Byte b);
86     /**
87      * @brief It produces a mesh of vertical and horizontal stripes all along the
88      * image. Every prim pixels it is set to 255 anad every sec pixels
89      * it is set to 127
90      * @param prim Gap between primary mesh
91      * @param sec Gap between secondary mesh, Default value is 0
92      */
93     void mesh(int prim, int sec=0);
94     /**
95      * @brief It calculates the histogram of the image, and returns it into the array values such
96      * that values[i] = number of pixels whose tone is i
97      * @param values
```

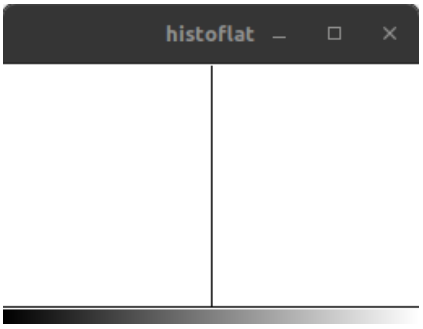
```
98          */
99         void getHistogram(int values[]) const;
100        /**
101         * @brief It shows an image in an external window, ready for inspection. It uses
102         * the program display (ImageMagick) to display every image. For an easier identification
103         * process of all images shown are labeled with a title
104         * @param title The title on top of the window
105         */
106        void showInWindow(std::string title);
107        /**
108         * @brief It calculates the hash value of the image and returns it inside a string,
109         * together with its dimension.
110         * @return a string that contains the dimension and the hash value of the image
111         */
112        std::string inspect();
113    };
114    #endif
```

3. Práctica a entregar

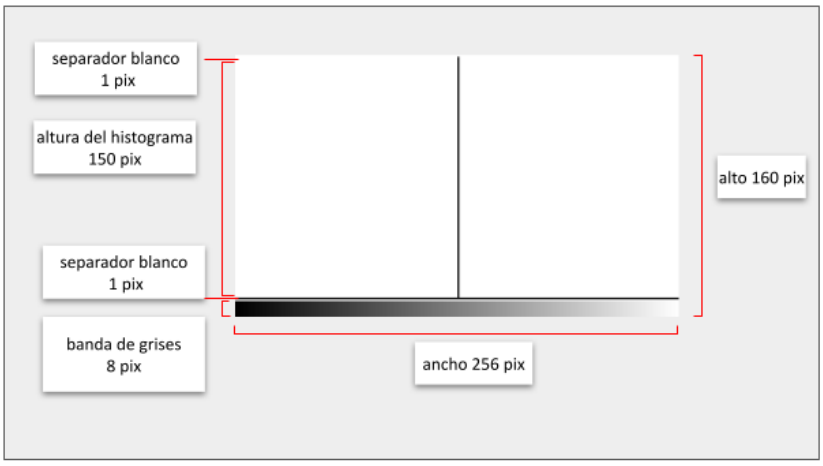
- Se deben implementar las funciones incluidas en el fichero `Image.h`
- se debe implementar un `main.cpp` que construya una imagen de 256x256 totalmente plana al valor 128



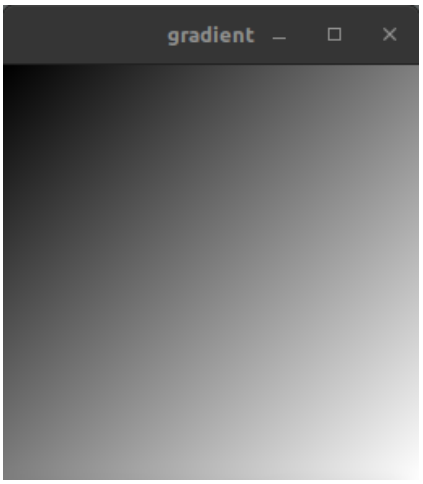
y calcular y mostrar su histograma. El histograma de una imagen es un vector de 256 posiciones de números enteros de forma que la posición i contiene el número de píxeles de la imagen que tienen el valor i.



Para visualizar el histograma, se hará uso de una imágen auxiliar cuyas proporciones se muestran a continuación y que se deberá construir a partir del histograma. Dado que la altura disponible para el histograma es de 140 píxeles, se debe ponderar mediante una sencilla proporción lineal.



- A continuación, debe crear una imagen 256x256 con un relleno degradado en diagonal tal y como muestra la figura siguiente, de forma que la posición (0,0) tenga el valor 0 y la posición (255,255) tenga el valor 255. El resto se distribuyen como una proporción lineal.



El histograma de esta imagen degradada es muy característico

Para comprobar que las imágenes se han generado correctamente, se debe generar su hash, un código biunívoco que las identifica en base a su contenido, haciendo uso de la función miembro `inspect`

3.1. Ejemplo de ejecución

Se muesta la salida, la cual contiene únicamente los código hash de las imágenes generadas y sus correspondientes histogramas

```
[flat] 256x256 5949961167589535660

[histogram] 256x160 2973619044510661841

[gradient] 256x256 15601680897343161879

[histogram] 256x160 14936481948836317653
```

3.2. Tests run

```
---
```