UNIVERSIDAD DE GRANADA

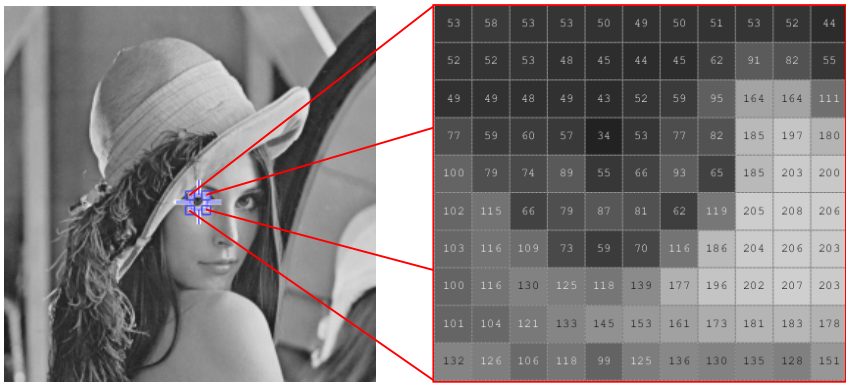
ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

DGIM

Curso 2021/2022



Guion de prácticas

Manejo del histograma y acceso a disco

Febrero de 2022

Índice

1. Objetivos

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

- Leer y grabar imágenes en disco, en un formato compatible con cualquier visualizador de fotos digitales
- Evolucionar el tipo de dato Histograma, que hasta ahora era un simple vector y transformarlo en una clase, con funciones propias, incluyendo la visualziación del histograma. Para ello habrá que restructurar algunas de las funciones de la práctica anterior.
- Realizar operaciones basadas en el histograma, como por ejemplo la segmentación de objetos de una imagen basada en sus colores.

2. Formato PGM de imágenes digitales

(Extraído de Wikipedia (Abrir →))

Netpbm (antes Pbmplus) es un paquete de programas gráficos de código abierto y una biblioteca de programación. Se utiliza principalmente en el mundo Unix, donde se puede encontrar incluido en las principales distribuciones de sistemas operativos de código abierto, pero también funciona en Microsoft Windows, macOS y otros sistemas operativos[2].

Formatos de archivo

El proyecto Netpbm utiliza y define varios formatos gráficos. El formato de mapa de píxeles portátil (PPM), el formato de mapa de grises portátil (PGM) y el formato de mapa de bits portátil (PBM) son formatos de archivo de imagen diseñados para ser intercambiados fácilmente entre plataformas. A veces también se les denomina colectivamente formato de mapa de bits portátil (PNM) que no debe confundirse con el formato de mapa arbitrario portátil (PAM). El ”número mágico”(Px) al principio de un archivo determina el tipo, no la extensión del archivo, aunque es una buena práctica utilizar la extensión correcta si es posible.

El formato PBM fue inventado por Jef Poskanzer en la década de 1980 como un formato que permitía transmitir mapas de bits monocromáticos dentro de un mensaje de correo electrónico como texto ASCII plano, lo que le permitía sobrevivir a cualquier cambio en el formato del texto Poskanzer desarrolló la primera biblioteca de herramientas para manejar el formato PBM, Pbmplus, publicada en 1988. Contenía principalmente herramientas para convertir entre PBM y otros formatos gráficos. A finales de 1988, Poskanzer había desarrollado los formatos PGM y PPM junto con sus herramientas asociadas y los añadió a Pbmplus. La versión final de Pbmplus fue el 10 de diciembre de 1991.

El formato PGM ASCII

- La primera línea del fichero es diferente (ver Figura 1), siendo P2 para las de texto y P5 para las binarias. A continuación vienen el número de columnas (256) y de filas (256), y el valor máximo de

lena_texto.pgm

```
P2
256 256
255
162 162 159 162 162 159 158 157 155 162
161 165 167 172 171 171 169 167 162 148
106 108 105 107 109 107 106 109 107 109
123 122 128 125 124 132 129 132 128 128
128 135 135 131 132 133 133 132 133 133
135 133 136 135 135 135 135 132 136 133
130 133 130 129 132 135 133 133 136 132
133 128 128 132 128 135 128 128 128 131
123 120 120 116 112 109 103 108 117 125
148 151 151 151 153 153 155 155 156 155
151 155 155 156 155 156 157 156 153 155
```

lena.pgm

```
P5
256 256
255
```

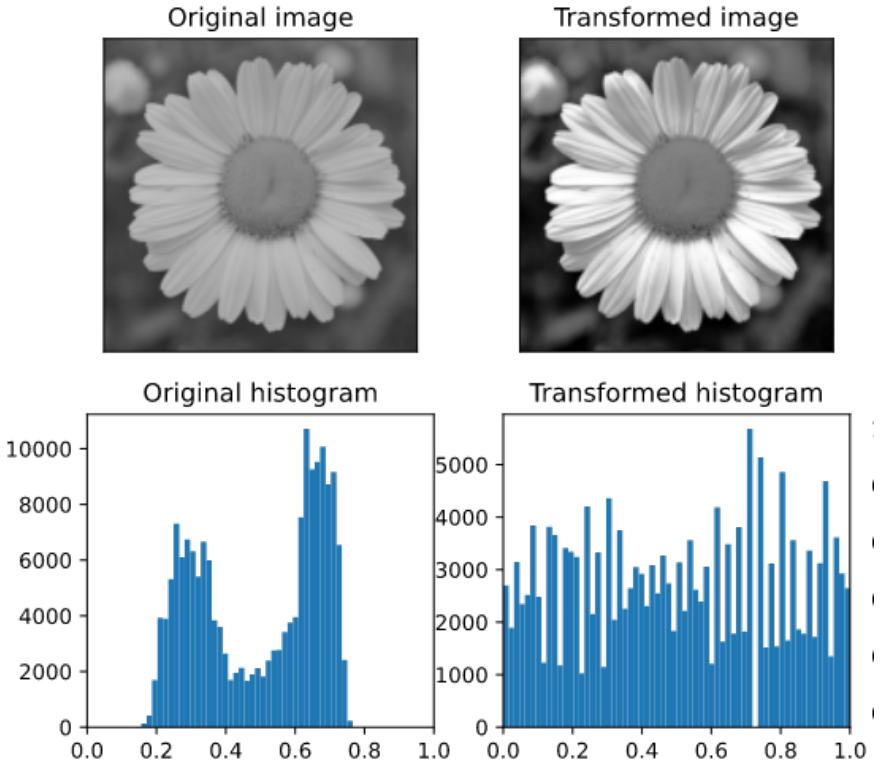
Figura 1: Contenido de la imagen 'lena.pgm' en formato de texto (izquierda) y binario (derecha).

todos de grises que contiene (siempre 255, independientemente del valor máximo real).

- El contenido de la imagen de texto es perfectamente legible con un editor de texto mientras que la imagen binaria no lo es.

3. La clase histograma

La información del histograma qe se ha visto en la práctica anterior es muy útil para realizar operaciones sobre la propia imagen, como realzar los tonos ecualizando el histograma http://(Abrir →) o la segmentación de la imagen a partir del histograma, que es la que se va a abordar en esta práctica.



3.1. Segmentación del histograma

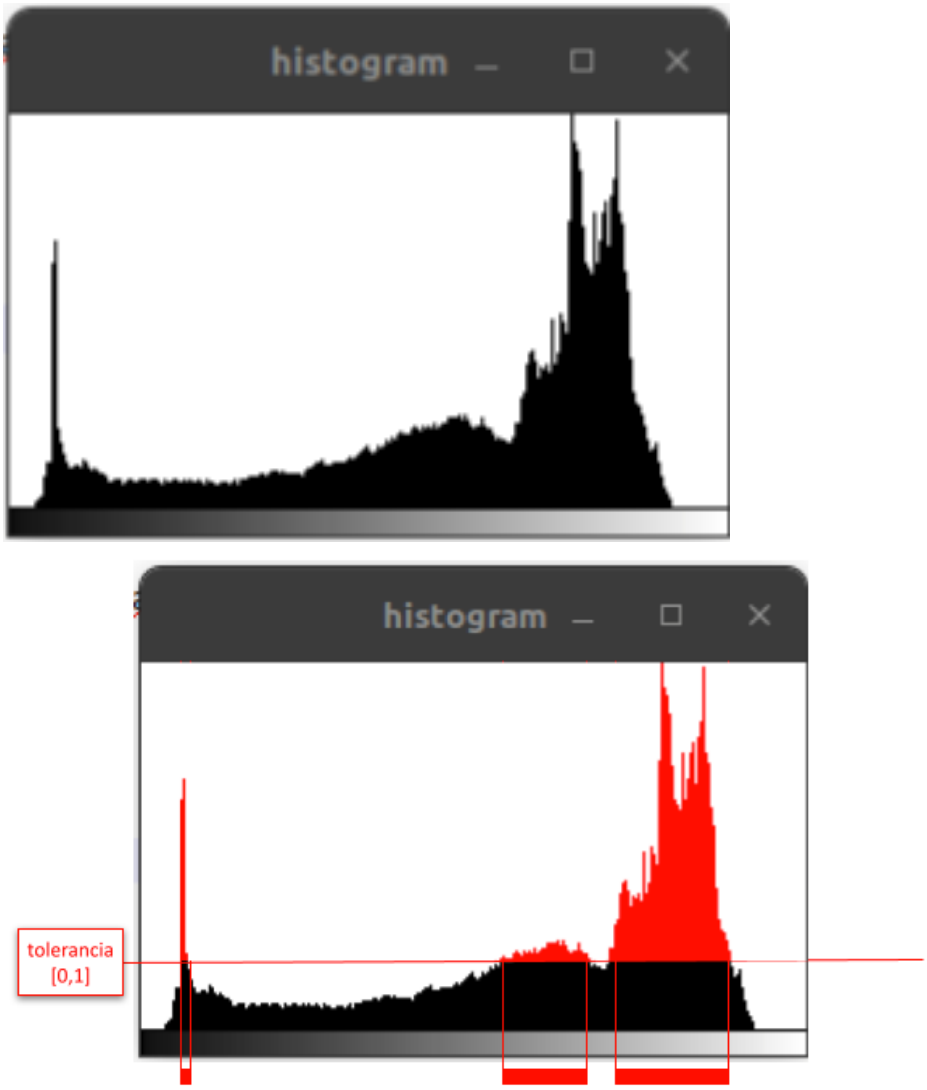
Para ello se supone que la imagen contiene objetos bien delimitados y cada uno de un color/tono diferente. De hecho si se observan estas imágenes, se puede ver en los histogramas como aparecen grupos de píxeles agrupados por tonos parecidos.

La segmentación permite identificar cada una de estas agrupaciones de píxeles de tonos similares como un objeto de la imagen. Para ello, el procedimiento será el siguiente.

1. Ir recorriendo todo el histograma y buscar qué conjunto de tonos sucesivos tiene una frecuencia igual o superior a

$$tolerancia * max_{i \in [0,255]} histograma(i)$$

$$tolerancia \in [0, 1]$$



2. Por cada grupo de tonos similares, crear una nueva imagen, de las mismas dimensiones, totalmente negra excepto aquellos píxeles que contienen el rango de tonos seleccionado. Las imágenes anteriores se han segmentado con una tolerancia del histograma de $0{,}1$.

4. Histogram

```
1  /**
2   * @file Histogram.h
3   * @author MP
4   */
5  #include <istream>
6  #include <fstream>
7  #include "Byte.h"
8  #include "Image.h"
9
10 #ifndef _HISTOGRAM_H_
11 #define _HISTOGRAM_H_
12
13
14
15 #define HISTOGRAM_LEVELS MAX_BYTE ///< Max number of bytes allowd for
16 #define HISTOGRAM_TOLERANCE 0.01
17
18 /**
19 @brief A black and white histogram
20  */
21 class Histogram {
22 private:
23     int _data[HISTOGRAM_LEVELS]; ///< datos de la imagen
24
25 public:
26
27     /**
28      * @brief It builds an empty
29      */
30     Histogram();
31     /**
32      * @brief It returns the number of levels in the histogram
33      * @return The number of levels
34      */
35     int size() const;
36     /**
37      * @brief Sets the whole histogram to 0
38      */
39     void clear();
40     /**
41      * @brief It returns the value associated to the level indicated
42      * @param level The level indicated
43      * @return The value associated to the level
44      */
45     int getLevel(Byte level) const;
46     /**
47      * @brief It sets the value associated to the level
48      * @param level The level
49      * @param npixeles The new value
50      */
51     void setLevel(Byte level, int npixeles);
52     /**
53      * @brief It returns the maximum value stored
54      * @return The max of the levels
55      */
56     int getMaxLevel() const;
57     /**
58      * @brief it returns the average value stored
59      * @return The average level
60      */
61     int getAverageLevel() const;
62     /**
63      * @brief It returns a unique hash code for every object so that they might be compared
64      * @return The hash code as an string
65      */
66     std::string inspect() const;
67 };
68 #endif
```

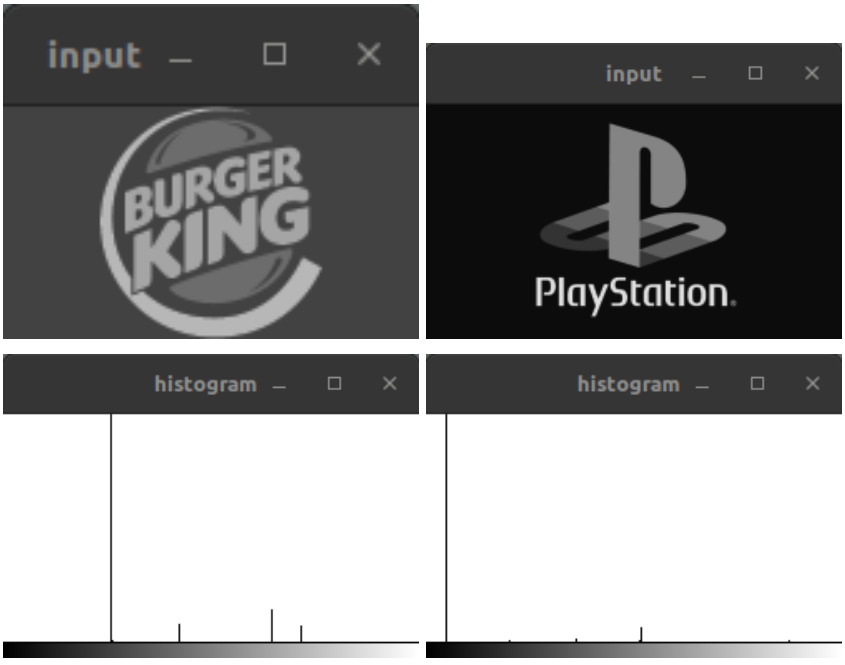
5. Image

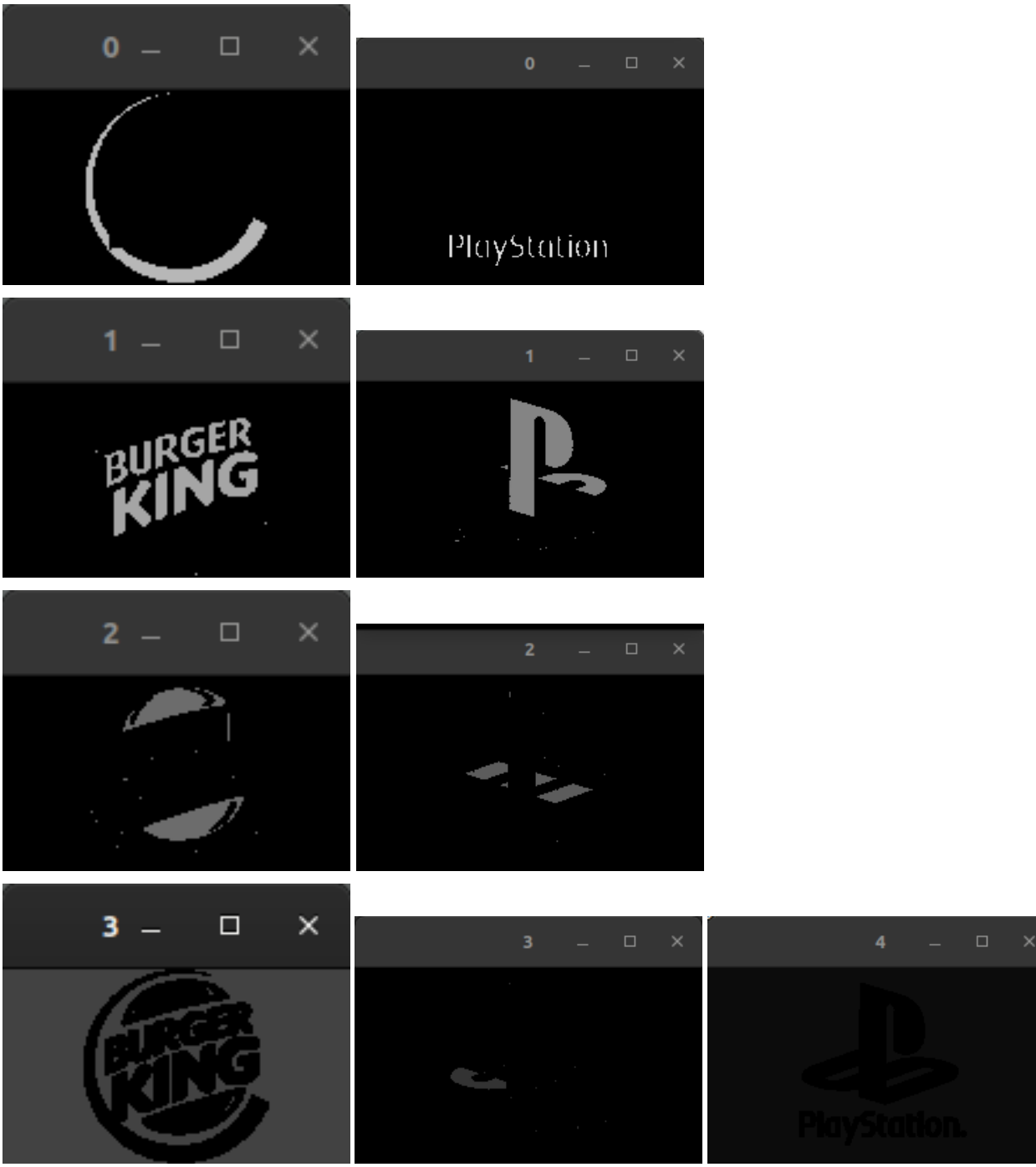
```
/**
@file Image.h
@brief Manejo de imÃ¡genes digitales en formato PGM blanco y negro
@author MP-DGIM - Grupo A
 **/

#ifndef _IMAGE_H_
#define _IMAGE_H_

#include <istream>
#include <fstream>
#include "Byte.h"
#include "Histogram.h"

#define IMAGE_MAX_SIZE 200000 ///< Max number of bytes allowd for
#define IMAGE_DISK_OK 0 ///< Image read/write successful
#define IMAGE_ERROR_OPEN 1 ///< Error opening the file
#define IMAGE_ERROR_DATA 2 ///< Missing data in the file
#define IMAGE_ERROR_FORMAT 3 ///< Unknown image format
#define IMAGE_TOO_LARGE 4 ///< The image is too large and does not fit into memory
/**
@brief A black and white image
 */
class Image {
private:
    Byte _data[IMAGE_MAX_SIZE]; ///< Bytes of the image
    int _height; ///< number of rows
    int _width; ///< number of columms
public:
    /**
     * @brief It builds an empty, image
     */
    Image();
    /**
     * @brief It builds a fully black image with @a width columns and @a height rows
     * @param height number of rows
     * @param width number of columns
     */
    Image(int width, int height);
    /**
    @brief It gives the number of rows of the image
    @return number of rows
     */
    int height() const;
    /**
    @brief It gives the number of columns of the image
    @return The number of rows
     */
    int width() const;
    /**
     * @brief It assigns the value @a v to the position(x,y) of the image.It must check that
     * the values x and y are valid, otherwise, it does not do anythig.
     * @param x The column
     * @param y the row
     * @param v The new value
     */
    void setPixel(int x, int y, Byte v);
    /**
     * @brief It returns the value of the requested (x,y) position. It must check that
     * the values x and y are valid, otherwise, it returns a negative value. Please note that
     * the value returned is a int
     * @param x The column
     * @param y the row
     * @return The value of the pixel in [0-256] or -1 if there is an access error
     */
    int getPixel(int x, int y) const;
    /**
     * @brief It assigns the value @a v to the linear position i of the image. It must check that
     * the values i is valid, otherwise, it does not do anythig.
     * @param i The linear position
     * @param v The new value
     */
    void setPos(int i, Byte v);
    /**
     * @brief It returns the value of the requested linear position. It must check that
     * the value i is valid, otherwise, it returns a negative value. Please note that
     * the value returned is a int
     * @param i The linear position
     * @return The value of the pixel in [0-256] or -1 if there is an access error
     */
    int getPos(int i) const;
    /**
     * @brief  It sets all pixels of the image to the value given
     * @param b The value
     */
    void flatten(Byte b);
    /**
     * @brief It produces a mesh of vertical and horizontal stripes all along the
     * image. Every prim pixels it is set to 255 anad every sec pixels
     * it is set to 127
     * @param prim Gap between primary mesh
     * @param sec Gap between secondary mesh, Default value is 0
     */
    void mesh(int prim, int sec=0);

    /**
```

```
97        * @brief It shows an image in an external window, ready for inspection. It uses
98        * the program display (ImageMagick) to display every image. For an easier identification
99        * process of all images shown are labeled with a title
100       * @param title The title on top of the window
101       */
102      void showInWindow(std::string title);
103      /**
104       * @brief It calculates the hash value of the image and returns it inside a string,
105       * together with its dimension.
106       * @return a string that contains the dimension and the hash value of the image
107       */
108      std::string inspect();
109      /**
110       * @brief It opens a file that contains a PGM Image and reads the data into
111       * a iname in memmory
112       * @param filename Name of the file
113       * @return a code that means the following: 0 - Successful operation.
114       * 1 - Error opening the file
115       * 2 - Error reading the data
116       * 3 - The detected data does not follow the PGM techincal description
117       */
118      int readFromFile(const char filename[]);
119
120      /**
121       * @brief It writes the Image on disk, in PGM ascii format
122       * @param filename The name of the disk file which will contain the image
123       * @return The same code that readFromFile()
124       */
125      int saveToFile(const char filename[]) const;
126      /**
127       * @brief It calculates the histogram of the image, and returns it into an
128       * instance of the class Histogram
129       * @param values
130       */
131      Histogram getHistogram() const;
132      /**
133       * @brief It takes the histogram of the image and depicts a new image with the
134       * visualization of the histogram according to these rules
135       * ~~~~
136       *    +-     +--------------------------------o
137       *    |      |                                | ---- 1 pix white line
138       *    |      |    *                           | -+
139       *    |      |    **                          |  |
140       *    |      |    **                          |
141       *    |      |    ***                    *    |   Normalized histogram
142       *           |    ***                   **    |     150 pix
143       *    h=160  |    ****     *            **    |  |
144       *           |    *****   ***          ****   |  |
145       *    |      |   ************************** | -+
146       *    |      |--------------------------------| 1 pix  white line
147       *    |      |--------------------------------| -+
148       *    |      |                                |  8 pix Scale of gray levels
149       *    +-     o--------------------------------+ -+
150       *
151       *           |                                |
152       *           +---------- w  = 256 ------------+
153       * ~~~~
154       * @return
155       */
156      Image depictsHistogram() const;
157
158      /**
159       * @brief It segements the histogram by groups whose value is higher than the admitted tolerance and
          returns
160       * an array of images, each of which corresponds to one of these objects
161       * @param set The array of images
162       * @param nimages The number of images found
163       * @param maximages The max number of images to be found
164       * @param tolerance The tolerance addmitted
165       */
166      void extractObjects(Image set [], int &nimages, int maximages, double tolerance=HISTOGRAM_TOLERANCE)
          const;
167
168  };
169  #endif
```

6. Práctica a entregar

- Se deben implementar las funciones incluidas en el fichero `Image.h` y en `Histogram.h`
- Lectura de datos en disco. Las funciones deben abrir un fichero PGM ASCII con el formato comentado anteriormente y deben cargar los datos leídos en una imagen en memoria. La función de lectura debe comprobar que todo ha funcionado correctamente e infomar de ello (ver Imagen.h). En caso contrario, deberá indicar un código de error en las siguientes circunstancias
  1. El fichero no ha podido abrirse correctamente.
  2. El fichero no sigue el formato de encabezado de un PGM ASCII.
  3. La imagen des disco es demasiado grande y no cabría en la memoria asignada (ver constantes definidas en Imagen.h)
  4. El fichero contiene menos datos de los esperados.
- Escritura de datos a disco. Las funciones deben de poder escribir los datos contenidos en una instancia de la clase Imagen en un fichero PGM ASCII con el formato comentado anteriormente. Igualmente, se deben comprobar los posibles errores en esta operación.
- Se debe implementar la segmentación de objetos por histograma tal y como se ha comentado anteriormente, comenzando a recorrer el histograma desde los puntos más brillantes (255) hasta los más oscuros, y devolverlos en un array de imágenes, de forma que la posición [0] sea siempre la agrupación de tonos más brillantes.




0
0
PlayStation
1
BURGER KING
1
2
2
3
BURGER KING
3
4
PlayStation.

6.1. Ejemplo de ejecución

Este ejemplo se realiza sobre la imagen `burgerking.pgm` que se encuentra en la carpeta `data` con $tolerancia = 0{,}1$ y todas las imágenes resultantes se guardan en la carpeta `data`.

```
...Reading image from ./data/burgerking.pgm 150x84

[im_input] 150x84 18121322677578906296

[im_histogram] 256x160 13018380072908941681

Found object 0 in [182,183]
Found object 1 in [164,165]
Found object 2 in [107,108]
Found object 3 in [65,67]
Found 4objects

[im_collection[i]] 150x84 8097018573058301236

...Saving image into ./data/0.pgm

[im_collection[i]] 150x84 10495473064609598397

...Saving image into ./data/1.pgm

[im_collection[i]] 150x84 8805045236942730063

...Saving image into ./data/2.pgm

[im_collection[i]] 150x84 7680631663591832521

...Saving image into ./data/3.pgm
```

6.2. Tests run

```
---
```