3. The histogram of image intensity levels

3.1. Introduction

This laboratory work presents the concept of image histogram together with an algorithm for dividing the image histogram into multiple bins and reducing the image gray levels (gray levels quantization).

3.2. The histogram of intensity levels

Being given a grayscale image with L levels of intensity (for an image having 8 bits / pixel L = 255), the intensity (grey) level histogram is defined by a function h(g) that has as value, for each intensity level $g \in [0 \dots L]$, the number of pixels in the image or in the region of interest that have intensity equal to g.

$$h(g) = N_g \tag{3.1}$$

N_g – the number of pixels in the image or in the region of interest that have the intensity equal to g.

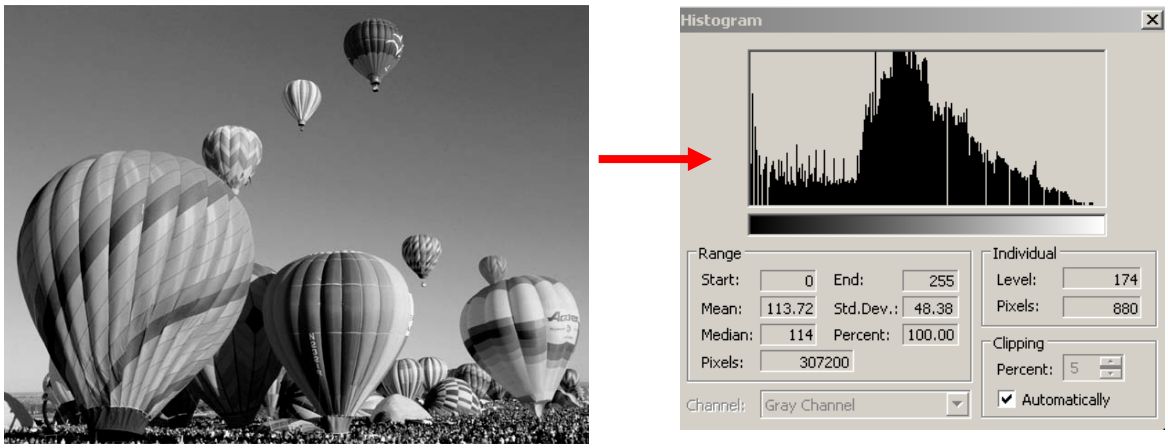


Fig. 3.1 Example: the histogram of a grayscale image

The function obtained by normalizing the histogram with the number of pixels in the image (in the ROI) is called the probability density function (PDF) of the intensity levels.

$$p(g) = \frac{h(g)}{M} \tag{3.2}$$

Where:

M = *image_height* x *image_width*

PDF has the following properties:

$$\begin{cases} p(g) >= 0 \\ \int_{-\infty}^{\infty} p(g)dg = 1, \quad \sum_{g=0}^{L} \frac{h(g)}{M} = \frac{M}{M} = 1 \end{cases} \tag{3.3}$$

3.3. Application: Multilevel thresholding

This algorithm determines multiple thresholds for reducing the number of image intensity (gray) levels. Its first step is to determine the histogram maxima. Then, each gray level is assigned to the closest maximum.

The following steps must be performed in order to determine the histogram maxima:

1. Normalize the histogram (transform it into a PDF)
2. Choose a window width 2**WH+1* (a good value for WH is 5)
3. Choose a threshold *TH* (a good value is 0.0003)
4. For each position (middle of the window) *k* from *0+WH* to *255-WH-1*
   - Compute the average *v* of normalized histogram values in the interval [*k-WH, k+WH*]. Remark: the value *v* is the average of *2*WH+1* values
   - If *PDF[k]>v+TH* and *PDF[k]* is greater or equal than all *PDF* values in the interval [*k-WH, k+WH*] then *k* corresponds to a histogram maximum. Store it and then continue from the next position.
5. Insert 0 at the beginning of the maxima position list and 255 at the end (this allows the colors black and white to be represented exactly).

The second step is thresholding. Thresholds are located at equal distances between the maxima. Therefore the algorithm for thresholding is simply: assign to each pixel the color value of the nearest histogram maximum.

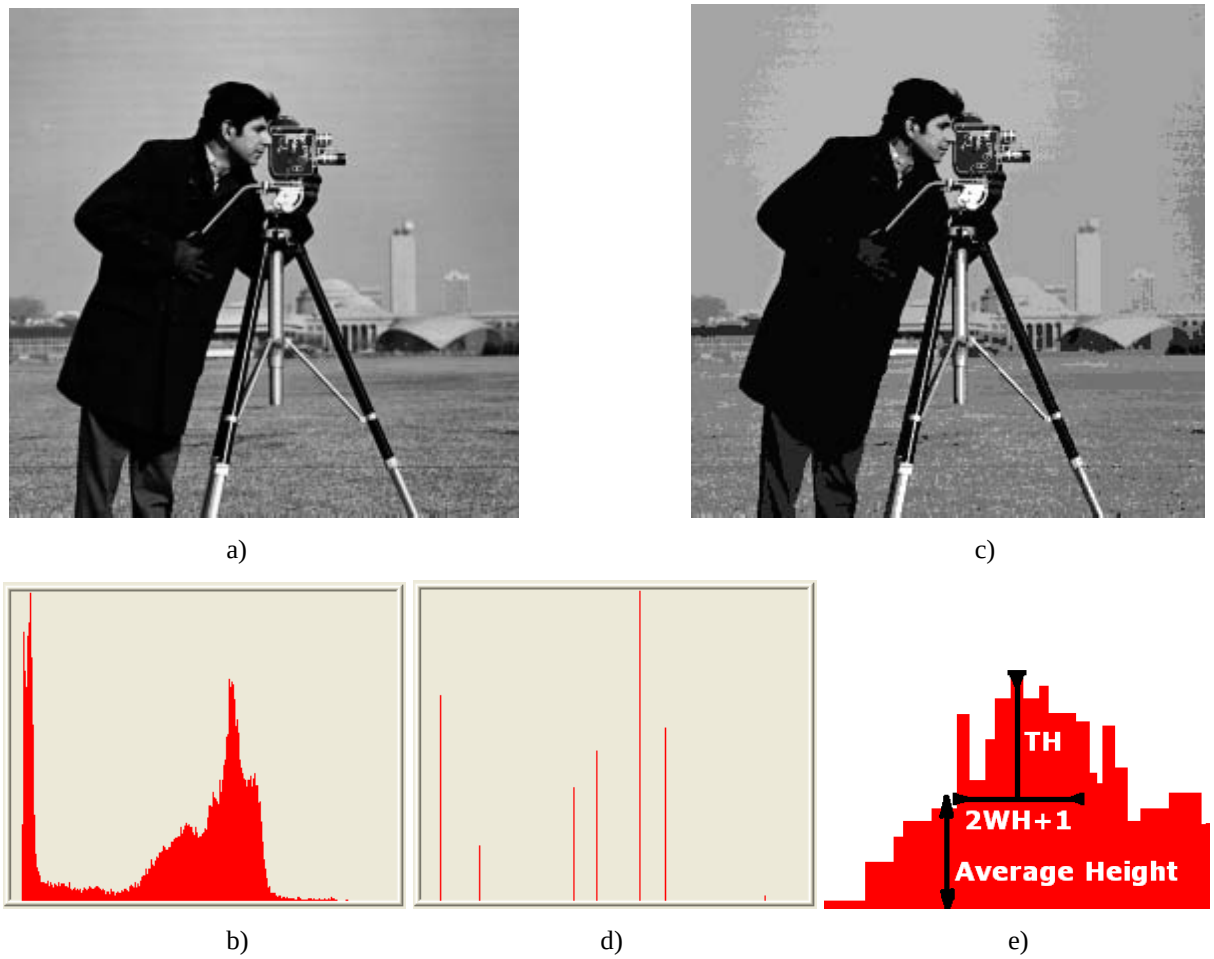


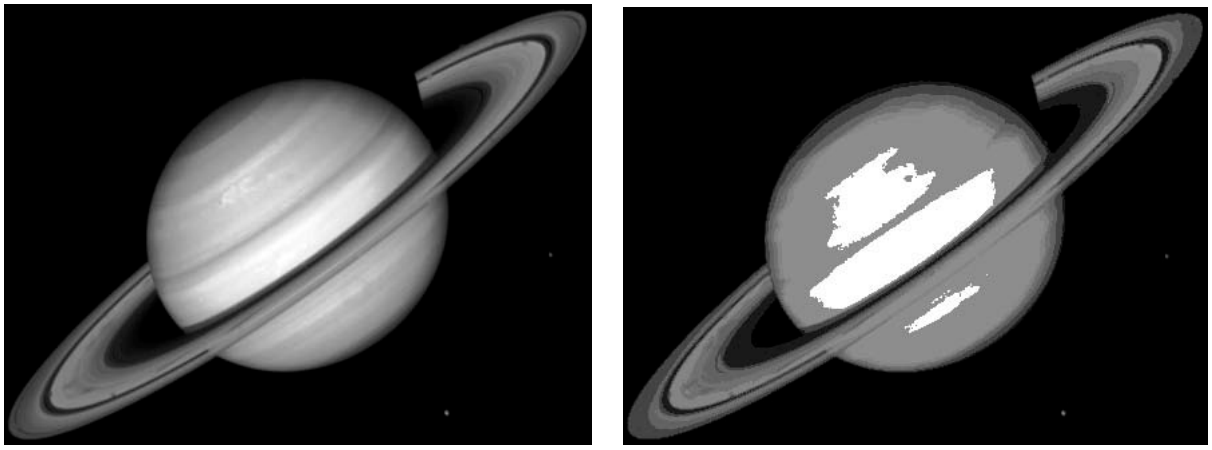
Fig. 3.2 a) The initial image; b) The histogram of the intitial image; c) The obtained multilevel thresholded image; d) The histogram of the multilevel thresholded image; e) The histogram maxima computation algorithm

As seen in the Fig. 3.3, the results are visually unacceptable when the number of gray levels is small. To obtain more visually acceptable a dithering algorithm can be applied. Such an algorithm spreads the quantization error to multiple pixels. An example of a dithering algorithm is the Floyd-Steinberg algorithm:

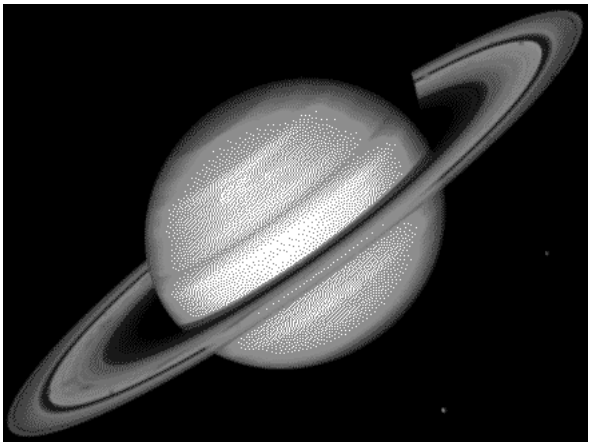
```
for each y from bottom to top
   for each x from left to right
      oldpixel := source(x,y)
      newpixel := closest histogram maximum
      destination(x,y) := newpixel
      error := oldpixel - newpixel
      source(x+1,y) := source(x+1,y) + 7*error /16
      source(x-1,y+1) := source(x-1,y+1) + 3*error/16
      source(x,y+1) := source(x,y+1) + 5*error/16
      source(x+1,y+1) := source(x+1,y+1) + error/16
```

This algorithm computes the quantization error and spreads it to the neighbouring pixels as in the following matrix:

3/16	5/16	1/16
0	0	7/16
0	0	0



a) b)



c)

Fig. 3.3 a) The initial image; b) The obtained multilevel thresholded image; c) Dithering on the initial image using the Floyd-Steinberg algorithm

3.4. Implementation details: histogram display in a dialog box

3.4.1. Option 1: Displaying the histogram in a *Picture* control

The histogram will be displayed using a dialog window (*Dialog Box*). It is used a control of type *Picture* inside the dialog box for displaying the histogram.

The control of type *Picture* will have a rectangular shape and implicitly a certain width and height. Its width must be at least L pixels, where L equals the number of intensity values in the image for which the histogram is computed (L=256 for an 8 bit/pixel grayscale image). The components of the histogram will be depicted in the form of vertical bars of height equal to the number of pixels corresponding to each intensity value. The vertical bars corresponding to the levels of intensity *0..L* will be displayed in order, from left to right.

Remark: in some cases the number of pixels having certain intensity in the histogram may be greater than the height of the *Picture* control component. For avoiding those cases, the displayed histogram will be scaled with a value (each value in the histogram array will be divided by the maximum value in the histogram and then it will be multiplied with the height of the *Picture* control component). The scaling step will be performed only if the maximum value in the histogram array is greater than the height of the display control.

1. Insert a new dialog box (check the laboratory work 2!).
2. In the dialog box, add a control of type *Picture* in which the histogram will be displayed. Modify the properties of the *Picture* type control (right click and *Properties*) (Fig. 3.4):
    a. Modify its ID to IDC_HISTOGRAM
    b. In the *Properties* section, for appearance, set "*True*" the values for *Client Edge* and *Modal Frame*.

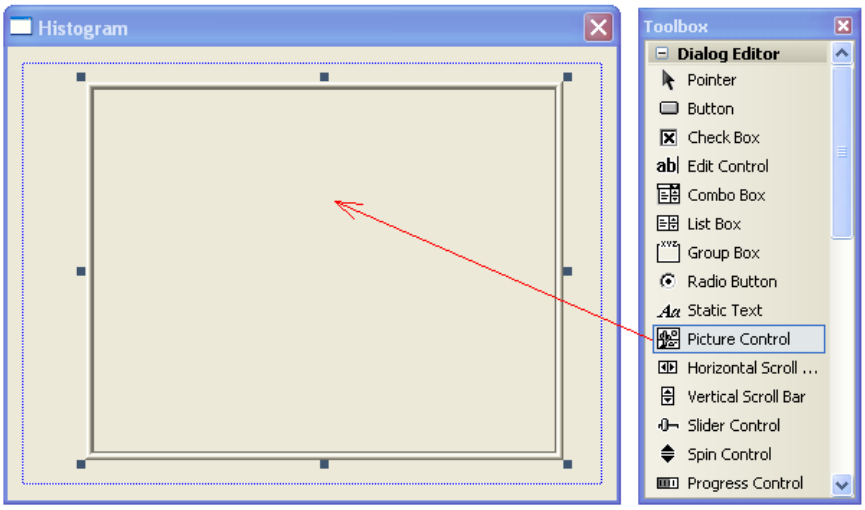


Fig. 3.4 The design of the dialog box for displaying the histogram

3. Create a new class for the previously created *Dialog Box* (right click and *Add Class…*). Name the class *CDlgHistogram* and then close the *Wizard*.
4. Create a new class for managing the control in which the histogram is displayed. This is done by accessing the main menu and following the steps bellow (Fig. 3.5):
    a. *Project -> Add class…*
    b. Choose the *Visual C++ -> MFC* category and then the *MFC Class* template
    c. Name the class *CHistogram* and choose its base class to be *CStatic*.

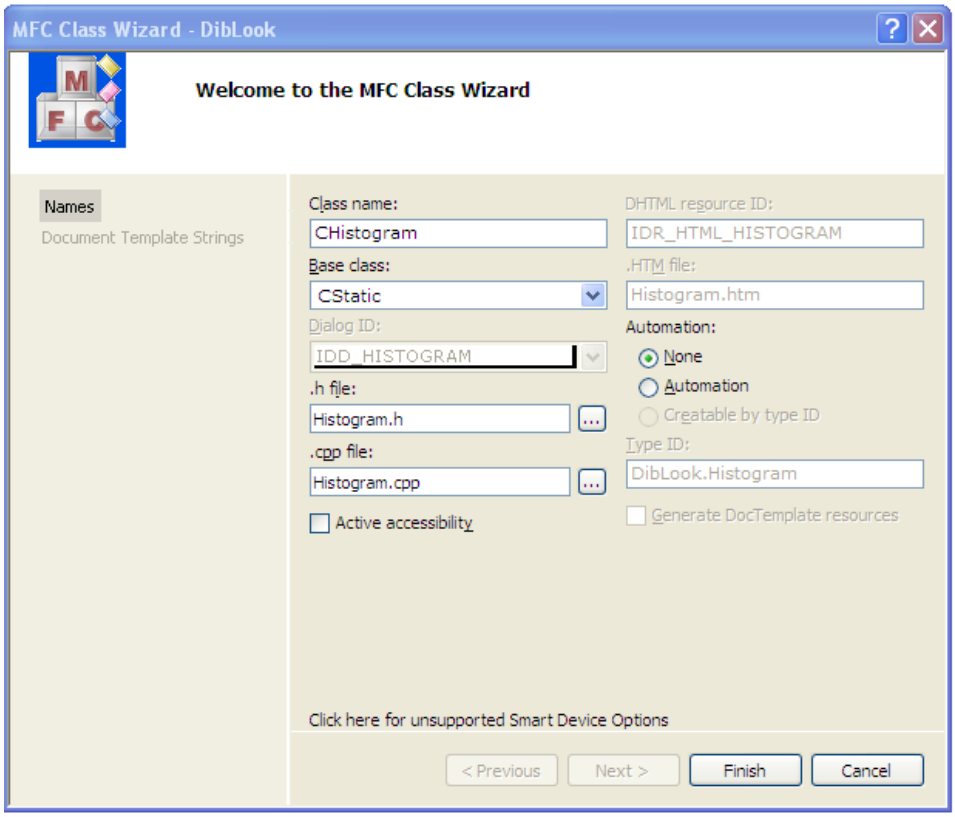


Fig. 3.5 The creation of the class *CHistogram* for the control used to display the histogram

5. For IDC_HISTOGRAM attach the variable *m_Histogram* of category *Control* and type *CHistogram* (Pay attention: the file *CDlgHistogram.h* must include the header *Histogram.h*) (Fig. 3.6)

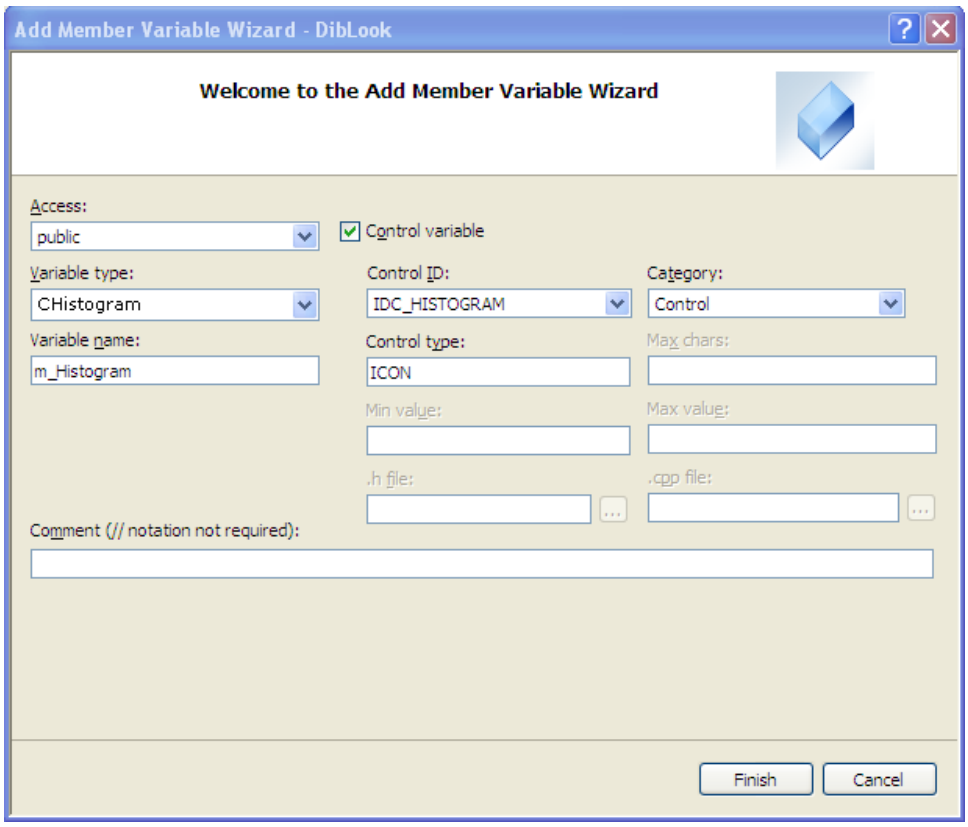


Fig. 3.6 Adding a member variable *m_Histogram* for the control in which the histogram will be displayed

6. Attach a handler for displaying the histogram:
    a. In the tabulator *Class View* perform a right click on the class *CHistogram* and then *Properties…*
    b. In the *Properties* window choose *Messages* section
    c. On the message *WM_PAINT* and the method *OnPaint* (*<Add> OnPaint*) using the combo-box

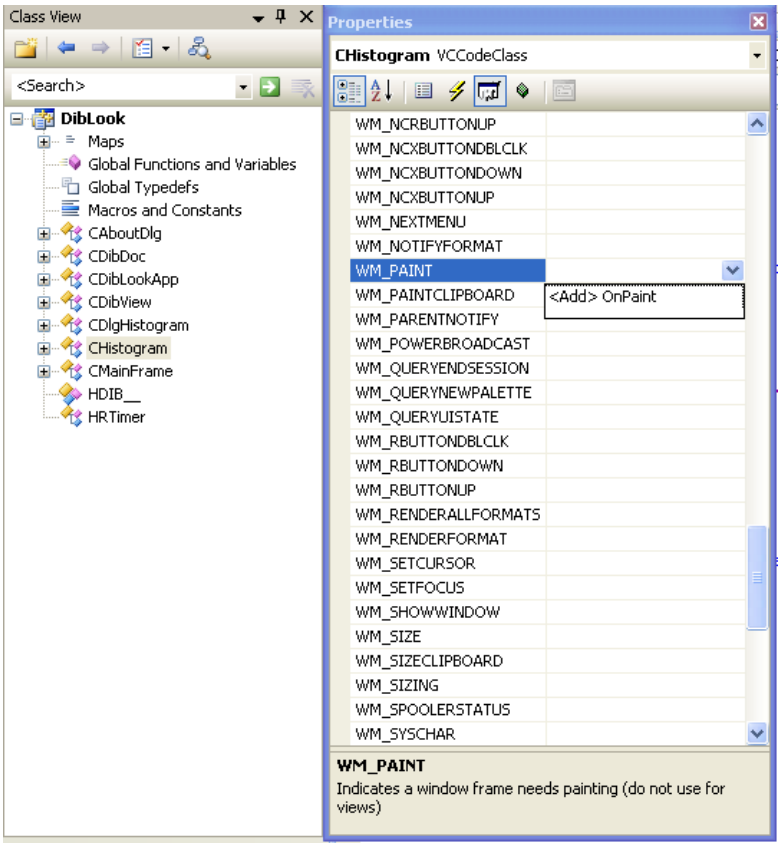


Fig. 3.7 Adding the method *OnPaint* attached to the message *WM_PAINT* for the class *CHistogram*

7. In the header file *Histogram.h* at the *public* section define the integer array *values[256]* that represents the histogram to be displayed (possibly scaled before being displayed)

```
class CHistogram : public CStatic
{
// Construction
public:
    CHistogram();

// Attributes
public:
    int values[256];

// There are no changes in the following lines
.
.
.
}
```

8. In the source file *Histogram.cpp*, rewrite the *OnPaint()* method for displaying (and possibly scaling) the histogram defined by the input array *values[256]*:

```
void CHistogram::OnPaint()
   {
       CPaintDC dc(this); // device context for display
       CPen pen(PS_SOLID, 1, RGB(255,0,0)); // define the display pen-
                                                    for red color
       CPen *pTempPen=dc.SelectObject(&pen); // select the display pen
       CRect rect;
       GetClientRect(rect); // get the available display rectangular area
       int height=rect.Height(); // height of the display area
       int width=rect.Width(); // width of the display area

       // find the maximum in the array values[256]
       int i;
       int maxValue=0;
       for (i=0;i<256;i++)
            if (values[i]>maxValue)
                 maxValue=values[i];

       // check if scaling is necessary
       double scaleFactor=1.0;
       if (maxValue>=height)
       {
            // scaling is necessary
            scaleFactor=(double)height/maxValue;
       }

       // display the histogram in the form of vertical bars
       for (i=0;i<256;i++)
       {
            // find the length of the line
            int lengthLine=(int)(scaleFactor*values[i]);
            //display the line
            dc.MoveTo(i,height);
            dc.LineTo(i,height-lengthLine);
       }

       dc.SelectObject(pTempPen); // restore the display pen
   }
```

9. Display the computed histogram in the created *Dialog-Box*:
    a. Add a new processing menu for computing the histogram and add a method associated to it : *OnDisplayHistogram()*
    b. Include the file *DlgHistogram.h* in the file *DibView.cpp*
    c. The method *OnDisplayHistogram()* attached to the click event on the processing menu will be defined as follows:

```
void CDibView::OnDisplayHistogram()
{
     BEGIN_SOURCE_PROCESSING;

     int histValues[256];
     float FDPValues[256];

     // write the code for computing the histogram and store it in the array
     of int, histValues[256]
     // write the code for computing the PDF and store it in the array of
     double FDPValues[256]

     // instantiate a dialog box for display and associate the histogram
     CDlgHistogram dlg;
     memcpy(dlg.m_Histogram.values,histValues,sizeof(histValues));
     // display the dialog box
     dlg.DoModal();

     END_SOURCE_PROCESSING;
}
```

3.4.2. Option 2: Displaying the histogram directly in the user area of a dialog box

Create a new dialog box and attach it the class *CDlgGrayStatistics*. Displaying the histogram directly in the user area of a dialog box as in Fig. 3.8 can be done by defining an *OnPaint* function corresponding to the WM_PAINT message associated to the dialog box/window (Fig. 3.9) as in the example bellow:

```
#define LEFT 10
#define HEIGHT 100
#define BOTTOM 200

void CDlgGrayStatistics::OnPaint()
{
	CPaintDC dc(this); // device context for painting
	POINT pct;
	for (int g=0; g<256; g++) {
		pct.x=LEFT + g;
		int N=(float)(m_hist[g]*HEIGHT)/(float)m_maxhist;
		for (int n=0; n <N ; n++) {
			pct.y=BOTTOM-n;
			dc.SetPixel(pct,RGB(0,0,0));
		}
	}
}
```

Where: `int *m_hist; int m_maxhist;` are public members of the dialog class and are initialized in your CDibView:: processing function …

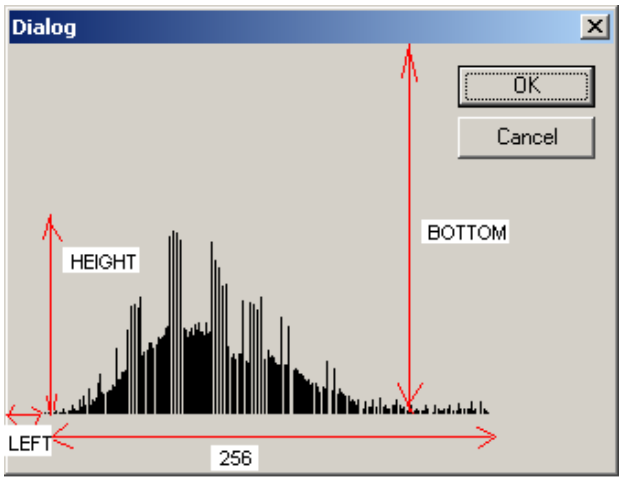


Fig. 3.8 Example of displaying the histogram directly in the user area of a dialog box

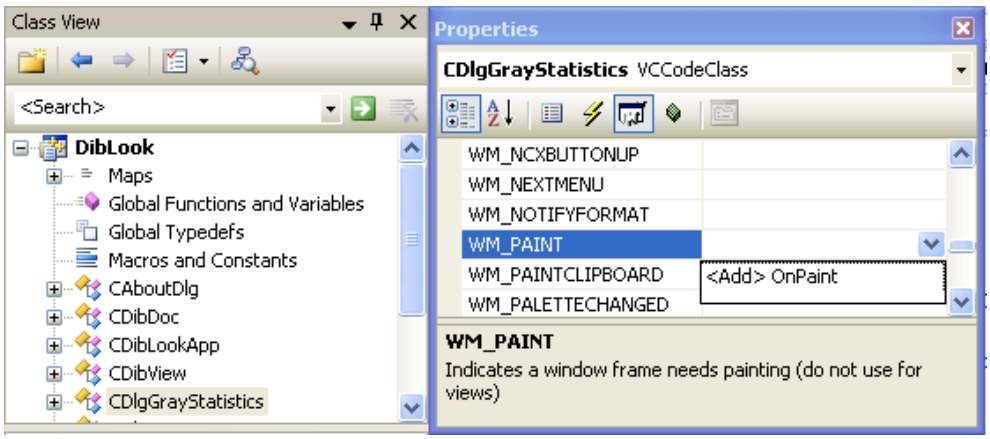


Fig. 3.9 Associating the *OnPaint* function to the WM_PAINT message of the dialog box

3.5. Practical work

1. Compute the histogram for a given grayscale image (in an array of integers having dimension 256) and the PDF (in a vector of float of dimension 256). Display the computed histogram by choosing a method presented in the section 3.4.
2. Implement the multilevel thresholding algorithm.
3. Enhance the multilevel thresholding algorithm using the Floyd-Steinberg dithering.
4. Save your work. Use the same application in the next laboratories. At the end of the image processing laboratory you should present your own application with the implemented algorithms.

Bibliography

[1]. R.C.Gonzales, R.E.Woods, *Digital Image Processing. 2-nd Edition*, Prentice Hall, 2002.
[2]. Floyd-Steinberg algorithm, http://en.wikipedia.org/wiki/Floyd-Steinberg_dithering