

Course IPCis: Image Processing with C#

Chapter C2: Commented Code of the Histogram Project

Copyright © by V. Miszalok, last update: 25-08-2006

```
using System; //Home of the base class of all classes "System.Object" and of all primitive data
types such as Int32, Int16, double, string.
using System.Drawing; //Home of the "Graphics" class and its drawing methods such as
DrawString, DrawLine, DrawRectangle, FillClosedCurve etc.
using System.Drawing.Imaging; //Home of the Bitmap-class.
using System.Windows.Forms; //Home of the "Form" class (base class of our main window Form1)
and its method Application.Run.
```

```
Entry public class Form1 : Form
```

```
//We derive our window Form1 from the class Form, which the compiler automatically finds in the
System.Windows.Forms namespace.
```

```
static void Main() { Application.Run( new Form1() ); //Create an instance of Form1 and
ask the operating system to start it as main window of our program.
```

```
Brush bbrush = SystemBrushes.ControlText; //Create a reference to an already existing Brush
object (just a programming shortcut).
```

```
Brush wbrush = new SolidBrush( Color.White ); //Create a new global Brush object.
```

```
Pen bpen = SystemPens.ControlText; //Create a reference to an already existing Pen object (just
a programming shortcut).
```

```
Pen rpen = new Pen( Color.Red ); //Create a new global Pen object.
```

```
Bitmap bmp, bmp_binary, bmp_histo; //Three Bitmap-objects: bmp = original image, bmp_binary
= binary image, bmp_histo = small histogram image in the right lower corner of bmp and bmp_binary.
```

```
BitmapData binaryData; //for Versions 2 and 3 //Clone of bmp_binary locked at a fixed
position in memory in order to be addressed with a pointer.
```

```
Byte[,] grayarray; //2D-Byte-Array //Raster matrix containing bmp as monochrome image. Any
gray value is the average of its three correspondent bmp-RGB-values: grayarray[x,y] = ( bmp[x,y].R +
bmp[x,y].G + bmp[x,y].B ) / 3;
```

```
Int32[] Histogram = new Int32[256]; //Histogram array of fixed length = 256.
```

```
Rectangle histo_r = new Rectangle( 0,0,257,101 ); //Size of bmp_histo.
```

```
Graphics g, g_histo;
```

```
//g = whole client area graphics object, g_histo = small graphics object just covering bmp_histo.
```

```
Byte[] ORmask = { 128, 64, 32, 16, 8, 4, 2, 1 }; // 1 bit each //for Version 3
//Array of byte masks =
10000000,01000000,00100000,00010000,00001000,00000100,00000010,00000001.
Needed for setting single bits in a byte.
```

```
Byte[] ANDmask = { 127, 191, 223, 239, 247, 251, 253, 254 }; // 7 bits each //for
Version 3 //Array of byte masks =
01111111,10111111,11011111,11101111,11110111,11111011,11111101,11111110.
Needed for removing single bits from a byte.
```

Constructor `public Form1()`

```
{ MenuItem miRead = new MenuItem( "&Read", new EventHandler( MenuFileRead ) );
//Purpose: Open an image file.
```

```
MenuItem miExit = new MenuItem( "&Exit", new EventHandler( MenuFileExit ) );
//Purpose: Leave the program.
```

```
MenuItem miFile = new MenuItem( "&File", new MenuItem[] { miRead, miExit } );
//Complete menu with two entries.
```

```
Menu = new System.Windows.Forms.MainMenu( new MenuItem[] { miFile } );
//Attach the menu to Form1.
```

```
Text = "Histo1"; //Blue title bar.
```

```
SetStyle( ControlStyles.ResizeRedraw, true );
//Redirect the OnResize-event to the OnPaint event handler.
```

```
Width = 800; //Initial window size.
```

```
Height = 600; //Initial window size. Raise the OnResize-event which calls
protected override void OnPaint(...).
```

Event Handler `void MenuFileRead(object obj, EventArgs ea)`

```
{ OpenFileDialog dlg = new OpenFileDialog();
//Call the standard modal OpenFileDialog-box.
```

```
if ( dlg.ShowDialog() != DialogResult.OK ) return;
//Forget it if there was no reasonable user reaction.
```

```
Cursor.Current = Cursors.WaitCursor; //Change the mouse cursor to hour glass to inform the
user that a time consuming process is underway.
```

```
bmp = (Bitmap)Image.FromFile( dlg.FileName );
//Read the image using the powerful Bitmap-class.
```

```
GenerateTheHistogram(); //Call a subroutine. (See some lines below.)
```

```
Cursor.Current = Cursors.Arrow; //Change back the mouse cursor to its normal form.
```

```
Invalidate(); //
```

```
Call protected override void OnPaint(...) and show the image together with its histogram.
```

```
Function void GenerateTheHistogram()
```

```
{ if ( bmp == null ) return; //Get out if there is no image.

//bmp_binary = new Bitmap( bmp.Width, bmp.Height, PixelFormat.Format32bppRgb );
//Version 1 //Black and white coded by ARGB = 32 bits.

//bmp_binary = new Bitmap( bmp.Width, bmp.Height, PixelFormat.Format32bppRgb );
//Version 2 //Black and white coded by ARGB = 32 bits.

bmp_binary = new Bitmap( bmp.Width, bmp.Height, PixelFormat.Format1bppIndexed );
//Version 3 //Black and white coded by 1 bit.

grayarray = new Byte[bmp.Height, bmp.Width]; //Raster of gray values.

Color color; //ARGB = 32 bit variable.

for ( Int32 y=0; y < bmp.Height; y++ ) //for any row.

for ( Int32 x=0; x < bmp.Width; x++ ) //for any column.

{ color = bmp.GetPixel( x, y ); //Read a pixel.

Int32 gray = ( color.R + color.G + color.B ) / 3; //Average (R+G+B)/3.

grayarray[y, x] = (Byte)gray; //Store the gray value.

Histogram[gray]++; //Increment the corresponding gray column of the histogram

Int32 hmax = 0; //Begin with zero.

for ( Int32 i=0; i < 256; i++ ) //For all columns of the histogram.

if ( Histogram[i] > hmax ) hmax = Histogram[i]; //Find the highest column.

for ( Int32 i=0; i < 256; i++ ) //For all columns of the histogram.

Histogram[i] = (100*Histogram[i]) / hmax; //Reduce the maximal height to 100.

bmp_histo = new Bitmap(histo_r.Width, histo_r.Height, PixelFormat.Format32bppRgb);
//Create an artificial color image of 257x101.

g_histo = Graphics.FromImage( bmp_histo );
//Derive a graphics object from that 257x101 image.

g_histo.FillRectangle( wbrush, 0,0,256,100 );
//Fill the 257x101 image with a white background.

g_histo.DrawString( "click here and move !", Font, bbrush, 1, 1 );
//Write some text into the 257x101 image.

for ( Int32 i=0; i < 256; i++ ) g_histo.DrawLine( bpen, i, 100, i, 100 -
Histogram[i] ); //Write the columns of the histogram into the 257x101 image.

g_histo.DrawRectangle( rpen, 0,0,256,100 ); //Draw a red frame around the 257x101 image.
```

```
Event Handler void MenuFileExit( object obj, EventArgs ea )
```

```
{ Application.Exit(); } //Kill the current instance of program histo1.
```

```
Event Handler protected override void OnMouseMove( MouseEventArgs e )
```

```
{ if ( e.Button == MouseButton.None ) return;
//Do nothing when no mouse button is pressed.
```

```
if ( !histo_r.Contains( e.X, e.Y ) ) return;
//Do nothing when the mouse is outside the 257x101 image.
```

```
if ( bmp == null ) return; //Do nothing when there is no image at all.
```

```
Byte threshold = (Byte)(e.X - histo_r.X); //Threshold = distance between the mouse X-
position and the left border of the 257x101 image.
```

```
//Version 1 (no pointers but slow)*****
```

Both `GetPixel`- and `SetPixel` methods are slow, because they lock (and unlock) the `Bitmap`-object from being moved by the garbage collector which sometime tries to optimize memory usage. This locking and unlocking procedure at any call of `GetPixel`- or `SetPixel` is quite time consuming.

```
for ( Int32 y=0; y < bmp.Height; y++ ) //For any row.
```

```
for ( Int32 x=0; x < bmp.Width; x++ ) //For any column.
```

```
{ if ( grayarray[y ,x] > threshold ) //If brightness higher than threshold.
```

```
bmp_binary.SetPixel( x, y, Color.White ); //Set bmp_binary[x,y] to white.
```

```
else bmp_binary.SetPixel( x, y, Color.Black ); //Set bmp_binary[x,y] to black.
```

```
//Version 2 (fast pointers creating a memory wasting 32-bit binary image)*
```

`unsafe` //In order to use fast pointers, we must declare the block to be unsafe and we have to explicitly set the compiler option `/unsafe`. The user of the program will be asked by his operating system if he agrees to run this unsafe code.

```
binaryData = bmp_binary.LockBits( new Rectangle( 0,0,bmp.Width,bmp.Height ),
ImageLockMode.WriteOnly, PixelFormat.Format32bppRgb );
```

//Lock `bmp_binary` from being accidentally shifted in memory by the garbage collector. This is necessary in order to obtain a pointer to the first pixel of `bmp_binary`.

```
UInt32* p2fix, p2run; // pointers to binaryData = output image
//Declaration of a fixed and a running pointer to binaryData = output image.
```

```
fixed ( Byte* plfix = grayarray ) // lock grayarray in memory
```

//The keyword `fixed` locks `grayarray` from being voluntarily shifted in memory by the garbage collector as `LockBits(...)` does with an image.

At the same time it furnishes a pointer `Byte* plfix = grayarray`.

```
{ Byte* plrun = plfix; // running pointer to grayarray //plfix cannot be incremented,
therefore we must copy its content to another pointer Byte* plrun.
```

```
p2fix = (UInt32*)binaryData.Scan0; // pointer to output image
//Scan0 furnishes a pointer to the first pixel of bmp_binary.
```

```
for ( int y=0; y < bmp.Height; y++ ) //For any column.
```

```
{ p2run = p2fix + y * bmp.Width; //p2run points to first byte in row y
//Pointer to the leftmost pixel in the row/line.
```

```
for ( int x=0; x < bmp.Width; x++ ) //For any row.
```

```
{ if ( *p1run++ > threshold ) *p2run++ = 0xFFFFFFFF; // white//
```

- 1) Compare the source pixel with the threshold.
 - 2) Shift the p1run-pointer to the next source pixel.
 - 3) Set color white into the destination pixel.
 - 4) Shift the p2run-pointer to the next destination pixel.
-

```
else *p2run++ = 0; // black//
```

- 1) Set color black into the destination pixel.
 - 2) Shift the p2run-pointer to the next destination pixel.
-

```
bmp_binary.UnlockBits( binaryData ); //end of p2fix, unlock bmp_binary //The garbage
collector can do whatever it wants with binaryData.
```

```
//Version 3 (fast pointers creating a 1-bit binary image)*****
```

```
unsafe
```

//In order to use fast pointers, we must declare the block to be unsafe and we have to explicitly set the compiler option /unsafe. The user of the program will be asked by his operating system if he agrees to run this unsafe code.

```
binaryData = bmp_binary.LockBits( new Rectangle( 0,0,bmp.Width,bmp.Height ),
ImageLockMode.WriteOnly, PixelFormat.Format1bppIndexed );
```

//Lock bmp_binary from being voluntarily shifted in memory by the garbage collector. This is necessary in order to obtain a pointer to the first pixel of bmp_binary.

```
Byte* p2fix, p2row, p2run; // pointers to binaryData = output image
```

//Declaration of a fixed and a running pointer to binaryData = output image.

```
fixed ( Byte* plfix = grayarray ) // lock grayarray = input image in memory
```

//The keyword fixed locks grayarray from being voluntarily shifted in memory by the garbage collector as LockBits(...) does with an image.

At the same time it furnishes a pointer Byte* plfix = grayarray.

```
{ Byte* plrun = plfix; // running pointer to grayarray //plfixcannot be incremented,
therefore we must copy its content to another pointer Byte* plrun.
```

```
p2fix = (byte*)binaryData.Scan0; // pointer to output image
```

//Scan0 furnishes a pointer to the first pixel of bmp_binary.

```
for ( int y=0; y < bmp.Height; y++ ) //For any column.
```

```
{ p2row = p2fix + y * binaryData.Stride; //p2row points to first byte in row y
//Pointer to the leftmost pixel in the row/line. binaryData.Stride furnishes the real length of a line
which is at least bmp_binary.Width/8. Furthermore binaryData.Stride automatically fills up
missing bits because bit series do seldom end at byte borders.
```

```
for ( int x=0; x < bmp.Width; x++ ) //For any row.
```

```
{ p2run = p2row + x / 8;
```

//Running byte pointer = pointer to leftmost byte in the row + (Int32)(bit_number/8). The remainder of this division (= bit_number modulo 8) will be used in the next two lines as index of both bit mask arrays.

```
if ( *p1run++ > threshold ) *p2run |= ORmask[ x % 8 ]; //set 1 bit //
```

- 1) Compare the source pixel with the threshold.
- 2) Shift the p1run-pointer to the next source pixel.
- 3) Select an ORmask depending on the remainder of (Int32)(bit_number/8).
- 4) Bitwise OR sets the 1-bit that the ORmask carries into the output image.

```
else *p2run &= ANDmask[ x % 8 ]; //remove 1 bit //
```

- 1) Select an ANDmask depending on the remainder of (Int32)(bit_number/8).
- 2) Bitwise AND sets the 0-bit that the ANDmask carries into the output image.

```
bmp_binary.UnlockBits( binaryData ); // end of p2fix, unlock bmp_binary
//The garbage collector can do whatever it wants with binaryData.
```

```
//End of Version 3 *****
```

```
g = CreateGraphics(); //We need a current graphics object of the whole client area of Form1.
```

```
g.DrawImage( bmp_binary, ClientRectangle );
//Draw the black and white bmp_binary image stretched to fill the whole client area of Form1.
```

```
g.DrawImage( bmp_histo, histo_r );
//Draw the small fixed size bmp_histo image into the lower right corner of bmp_binary.
```

```
g.DrawLine( rpen, histo_r.X+threshold, histo_r.Y, histo_r.X+threshold,
histo_r.Y+histo_r.Height-1 );
//Draw a vertical red line indicating the current threshold onto bmp_histo.
```

```
Event Handler protected override void OnMouseUp( MouseEventArgs e )
```

```
{ Invalidate(); } //Erase bmp_binary and show the original bmp again.
```

```
Event Handler protected override void OnPaint( PaintEventArgs e )
```

```
if ( bmp == null ) { e.Graphics.DrawString( "Open an Image File !", Font, bbrush, 0, 0 ); return; }
//Inform the user what to do, if there is no image at all.
```

```
e.Graphics.DrawImage( bmp, ClientRectangle );
//Draw the original bmp image stretched to fill the whole client area of Form1.
```

```
histo_r.X = ClientRectangle.Width - histo_r.Width - 10; //Horizontal position of the small
fixed size bmp_histo image in the lower right corner of the client area.
```

```
histo_r.Y = ClientRectangle.Height - histo_r.Height - 10; //Vertical position of the small
fixed size bmp_histo image in the lower right corner of the client area.
```

```
e.Graphics.DrawImage( bmp_histo, histo_r );
//Draw the small fixed size bmp_histo image into the lower right corner of the original bmp.
```