

Course IPCis: Image Processing with C#

Chapter C2a: The Complete Code of the Histogram with Threads

Copyright © by V. Miszalok, last update: 09-01-2008

Copy all this code into an empty `Form1.cs` of a new Windows Application C#-project `HistoWithThreads` and clear `Form1.Designer.cs` and `Program.cs`.

This code is 95% identical with [CIPCisHisto_Code.htm](#) which just has one main thread. The differences are highlighted in red.

The trick is to slice the image horizontally into an upper part processed by `thread1` and a lower part processed by `thread2`. With DualCore-CPU's this code runs 200% faster than the code without threading. Try it out with big images!

Problem+Solution 1: A thread-subroutine just accepts one single `object` as parameter. Simple solution: Pass just one integer number 1 or 2 to determine what thread to run and set the appropriate parameters inside the subroutine.

Problem+Solution 2: Both threads need the properties `bmp.Width` and `bmp.Height` and the value `threshold`. In order to avoid access-collisions to `bmp` and to avoid parameters, we define `int w=bmp.Width;`, `int h=bmp.Height;` and `Byte threshold;` as global variables.

Caution: Solutions 1 and 2 are far from being professional, but this is an introduction for newcomers who may be grateful for the slim code.

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;
using System.Threading;

public class Form1 : Form
{
    static void Main() { Application.Run( new Form1() ); }
    Brush bbrush = SystemBrushes.ControlText;
    Brush wbrush = new SolidBrush( Color.White );
    Pen bpen = SystemPens.ControlText;
    Pen rpen = new Pen( Color.Red );
    Bitmap bmp, bmp_binary, bmp_histo;
    BitmapData binaryData; //Descriptor of the Binary Output Image
    Byte[,] grayarray; //2D-Byte-Array
    Int32[] Histogram = new Int32[256];
    Rectangle histo_r = new Rectangle( 0,0,257,101 );
    Graphics g, g_histo;
    Byte[] ORmask = { 128, 64, 32, 16, 8, 4, 2, 1 };// 1 bit each
    Byte[] ANDmask = { 127, 191, 223, 239, 247, 251, 253, 254 };// 7 bits each
    Byte threshold;
    Int32 w, h; //bmp.Width and bmp.Height

    public Form1()
    {
        MenuItem miRead = new MenuItem( "&Read", new EventHandler( MenuFileRead ) );
        MenuItem miExit = new MenuItem( "&Exit", new EventHandler( MenuFileExit ) );
        MenuItem miFile = new MenuItem( "&File", new MenuItem[] { miRead, miExit } );
        Menu = new System.Windows.Forms.MainMenu( new MenuItem[] { miFile } );
        Text = "Histo1";
        SetStyle( ControlStyles.ResizeRedraw, true );
        Width = 800;
        Height = 600;
    }

    void MenuFileRead( object obj, EventArgs ea )
    {
        OpenFileDialog dlg = new OpenFileDialog();
        if ( dlg.ShowDialog() != DialogResult.OK ) return;
        try
        {
            Cursor.Current = Cursors.WaitCursor;
            bmp = (Bitmap)Image.FromFile( dlg.FileName );
            w = bmp.Width; h = bmp.Height;
            GenerateTheHistogram();
            Cursor.Current = Cursors.Arrow;
            Invalidate();
        } catch {}
    }

    void GenerateTheHistogram()
```

```

{ if ( bmp == null ) return;
  bmp_binary = new Bitmap( w, h, PixelFormat.Format1bppIndexed );
  grayarray = new Byte[h, w];
  Color color;
  for ( Int32 y=0; y < h; y++ )
    for ( Int32 x=0; x < w; x++ )
      { color = bmp.GetPixel( x, y );
        Int32 gray = ( color.R + color.G + color.B ) / 3;
        grayarray[y, x] = (Byte)gray;
        Histogram[gray]++;
      }
  Int32 hmax = 0;
  for ( Int32 i=0; i < 256; i++ )
    if ( Histogram[i] > hmax ) hmax = Histogram[i];
  for ( Int32 i=0; i < 256; i++ )
    Histogram[i] = (100*Histogram[i]) / hmax;
  bmp_histo = new Bitmap( histo_r.Width, histo_r.Height, PixelFormat.Format32bppRgb );
  g_histo = Graphics.FromImage( bmp_histo );
  g_histo.FillRectangle( wbrush, 0,0,256,100 );
  g_histo.DrawString( "click here and move!", Font, bbrush, 1, 1 );
  for ( Int32 i=0; i < binaryData.Length; i++ )
    ImageLockMode.WriteOnly, PixelFormat.Format1bppIndexed );
  Thread th1 = new Thread( myThreads ); th1.Start( 1 );
  Thread th2 = new Thread( myThreads ); th2.Start( 2 );
  th1.Join(); th2.Join();
  bmp_binary.UnlockBits( binaryData ); // unlock bmp_binary
  g = CreateGraphics();
  g.DrawImage( bmp_binary, ClientRectangle );
  g.DrawImage( bmp_histo, histo_r );
  g.DrawLine( rpen, histo_r.X+threshold, histo_r.Y,
              histo_r.X+threshold, histo_r.Y+histo_r.Height-1 );
}

public void myThreads( object number )
{ // Default values for th1: process upper half of the image
  Int32 grayarrayOffset = 0, firstLine = 0, lastLine = h/2;
  if ( (Int32)number == 2 ) // values for th2: process lower half of the image
    { grayarrayOffset = (h/2) * w; firstLine = h/2; lastLine = h; }
  unsafe
  { Byte* p2fix, p2row, p2run;
    fixed ( Byte* plfix = grayarray )
      { Byte* plrun = plfix + grayarrayOffset;
        p2fix = (byte*)binaryData.Scan0;
        for ( int y = firstLine; y < lastLine; y++ )
          { p2row = p2fix + y * binaryData.Stride;
            for ( int x=0; x < w; x++ )
              { p2run = p2row + x / 8;
                if ( *plrun++ > threshold ) *p2run |= ORmask[ x % 8 ];
                else *p2run &= ANDmask[ x % 8 ];
              } // end of for x
            } // end of for y
          } // end of fixed, end of plfix
      } // end of unsafe
  }

protected override void OnMouseDown(MouseEventArgs e)
{ Invalidate(); }

protected override void OnPaint( PaintEventArgs e )
{ if ( bmp == null )
  { e.Graphics.DrawString( "Open an Image File!", Font, bbrush, 0, 0 ); return; }
  e.Graphics.DrawImage( bmp, ClientRectangle );
  histo_r.X = ClientRectangle.Width - histo_r.Width - 10;
  histo_r.Y = ClientRectangle.Height - histo_r.Height - 10;
  e.Graphics.DrawImage( bmp_histo, histo_r );
}
}

```