Course IPCis: Image Processing with C#
Chapter C2: The Histo Project

Projekt histo1 mit leerem Fenster

Anleitung für **Visual Studio 2008**:
1) Main Menu nach dem Start von VS 2008: `File → New Project... →`
`Visual Studio installed templates: Windows Forms Application`
`Name: histo1 → Location: C:\temp → Create directory for solution:` ausschalten `→ OK`
Es meldet sich `Form1.cs[Design]`.
2) Sie müssen zwei überflüssige Files löschen: `Form1.Designer.cs` und `Program.cs`.
Sie erreichen diese Files über das `Solution Explorer - histo1`-Window:
Klicken Sie das Pluszeichen vor `histo1` und dann das Pluszeichen vor `Form1.cs`.
Klicken Sie mit der **rechten** Maustaste auf den Ast `Program.cs`. Es öffnet sich ein Kontextmenu.
Sie Klicken auf `Delete`. Eine Message Box erscheint: `'Program.cs' will be deleted permanently.`
Sie quittieren mit `OK`.
Klicken Sie mit der **rechten** Maustaste auf den Ast `Form1.Designer.cs` und löschen auch dieses File.
3) Klicken Sie mit der **rechten** Maustaste auf das graue Fenster `Form1`.
Es öffnet sich ein kleines Kontextmenü. Klicken Sie auf `View Code`.
Sie sehen jetzt den vorprogrammierten Code von `Form1.cs`. Löschen Sie den gesamten Code vollständig.
4) Schreiben Sie in das vollständig leere File `Form1.cs` folgende 3 Zeilen:

```
public class Form1 : System.Windows.Forms.Form
{ static void Main() { System.Windows.Forms.Application.Run( new Form1() ); }
}
```

5) Klicken Sie `Debug → Start Without Debugging Ctrl F5`.
Beenden Sie `histo1.exe`.

Bild lesen und anzeigen

Sie löschen alles und schreiben in das leere Codefenster `Form1.cs` folgenden Code:

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

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; //for Versions 2 and 3
  Byte[,] grayarray;     //2D-Byte-Array
  Int32[] Histogram = new Int32[256];
  Rectangle histo_r = new Rectangle( 0,0,257,101 );
  Graphics g, g_histo;
```

```
  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 );
      Cursor.Current = Cursors.Arrow;
      Invalidate();
    } catch {}
  }
  void MenuFileExit( object obj, EventArgs ea )
  { Application.Exit(); }
  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 );
  }
}
```

Klicken Sie `Debug → Start Without Debugging Ctrl F5`. Erproben Sie das Programm. Lesen Sie Bilder der Formate `BMP, ICO, GIF, JPG, PNG, TIFF`.

Code für Histogramm

Beenden Sie Programm `histo1`.
Schreiben Sie in der Funktion `void MenuFileRead( object obj, EventArgs ea )` unterhalb der Zeile `bmp = (Bitmap)Image.FromFile( dlg.FileName );` aber noch vor der Zeile `Cursor.Current = Cursors.Arrow;` die neue Zeile:

```
    GenerateTheHistogram();
```

Schreiben Sie zwischen die beiden Funktionen `void MenuFileRead(...)` und `void MenuFileExit(...)` folgende neue Funktion:

```
  void GenerateTheHistogram()
  { if ( bmp == null ) return;
    bmp_binary = new Bitmap( bmp.Width, bmp.Height, PixelFormat.Format32bppRgb ); //Version 1
    //bmp_binary = new Bitmap( bmp.Width, bmp.Height, PixelFormat.Format32bppRgb ); //Version 2
    //bmp_binary = new Bitmap( bmp.Width, bmp.Height, PixelFormat.Format1bppIndexed ); //Version 3
    grayarray = new Byte[bmp.Height, bmp.Width];
    Color color;
    for ( Int32 y=0; y < bmp.Height; y++ )
      for ( Int32 x=0; x < bmp.Width; 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 < 256; i++ ) g_histo.DrawLine( bpen, i, 100, i, 100-Histogram[i] );
    g_histo.DrawRectangle( rpen, 0,0,256,100 );
  }
```

Schreiben Sie in der Funktion `protected override void OnPaint( PaintEventArgs e )` unterhalb der Zeile `e.Graphics.DrawImage( bmp, ClientRectangle );` folgende weitere Zeilen:

```
    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 );
```

Klicken Sie `Debug → Start Without Debugging Ctrl F5`. Erproben Sie das Histogramm.

Mausereignisse und Schwellwert

Beenden Sie `histo1`.
Schreiben Sie nach der Funktion `void MenuFileExit(...)` aber noch vor die Funktion `protected override void OnPaint(...)` folgende beiden Funktionen zur Behandlung der Mausereignisse:

```
  protected override void OnMouseMove( MouseEventArgs e )
  { if ( e.Button == MouseButtons.None ) return;
    if ( !histo_r.Contains( e.X, e.Y ) ) return;
    if ( bmp == null ) return;
    Byte threshold = (Byte)(e.X - histo_r.X);
    //Version 1 (no pointers but slow)************************************
    for ( Int32 y=0; y < bmp.Height; y++ )
      for ( Int32 x=0; x < bmp.Width; x++ )
      { if ( grayarray[y ,x] > threshold )
             bmp_binary.SetPixel( x, y, Color.White );
        else bmp_binary.SetPixel( x, y, Color.Black );
      }
    //End of Version 1 ***************************************************
    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 );
  }

  protected override void OnMouseUp(MouseEventArgs e)
  { Invalidate(); }
```

Klicken Sie `Debug` → `Start Without Debugging Ctrl F5`. Erproben Sie die Binarisierung.

Schneller Pixelzugriff durch Pointer

Beenden Sie `histo1`.
Die Binärbilder, die beim bewegen der Maus über dem Histogram entstehen, erscheinen ruckelig und mit Verzögerung. Man kann das deutlich beschleunigen, wenn man mit Pointern auf die Pixel zugreift, anstatt mit den Befehlen "`SetPixel( x, y, Color.White );`" und "`SetPixel( x, y, Color.Black );`"
Dazu müssen Sie zunächst dem Compiler mitteilen, dass er Pointer zulassen soll, was in C# normalerweise verboten ist. Klicken Sie im Hauptmenu von VS 2008 auf `Project → histo1 Properties → Build` und setzen Sie den 3. Flag "`Allow unsafe code`" auf `True` →
verlassen der `histo1 Property Pages` durch Click auf den Karteireiter `Form1.cs`.
Kommentieren Sie in der Funktion `protected override void OnMouseMove(...)` alles zwischen den beiden Kommenterzeilen mit zwei Kommentarklammern "`/*`" und "`*/`" aus. Etwa so:

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

zwei `for`-Schleifen bestehend aus 6 Zeilen incl. der geschweiften Klammern

```
//End of Version 1 ***************************************************
*/
```

Ersetzen Sie den auskommentierten Code durch:

```
    //Version 2 (fast pointers creating a memory wasting 32-bit binary image)*
    unsafe
    { //lock bmp_binary from being shifted in memory by the garbage collector
      binaryData = bmp_binary.LockBits( new Rectangle( 0,0,bmp.Width,bmp.Height ),
                           ImageLockMode.WriteOnly, PixelFormat.Format32bppRgb );
      //Byte* p1fix, p1run =  fixed + running pointers to grayarray  = input image
      UInt32* p2fix, p2run; //fixed + running pointers to binaryData = output image
      fixed ( Byte* p1fix = grayarray )    // lock grayarray in memory
      { Byte* p1run = p1fix;               // running pointer to grayarray
        p2fix = (UInt32*)binaryData.Scan0; // pointer to output image
        for ( int y=0; y < bmp.Height; y++ )
        { p2run = p2fix + y * bmp.Width; //p2run points to first byte in row y
          for ( int x=0; x < bmp.Width; x++ )
          { if ( *p1run++ > threshold ) *p2run++ = 0xFFFFFF; // white
            else                        *p2run++ = 0;        // black
          } // end of for x
        }   // end of for y
      }     // end of fixed, end of p1fix, unlock grayarray
      bmp_binary.UnlockBits( binaryData ); //end of p2fix, unlock bmp_binary
    } // end of unsafe
    //End of Version 2 ***************************************************
```

Klicken Sie `Debug` → `Start Without Debugging Ctrl F5`. Erproben Sie die neue Binarisierung. Sie arbeitet deutlich schneller als die alte. Bei kleinen Bildern (z.B. Madonna.bmp) folgt die Binarisierung der Mausbewegung ohne Verzögerung.
Man kann die Binarisierung weiter beschleunigen, indem man den ersten und den letzten Befehl aus dem `unsafe`-Block herausnimmt und beide zusammen ans Ende von `void GenerateTheHistogram()` verlagert und nur ein Mal (statt bei jedem `MouseMove`) ausführt. Man schreibt dort einfach:

```
      //lock bmp_binary from being shifted in memory by the garbage collector
      binaryData = bmp_binary.LockBits( new Rectangle( 0,0,bmp.Width,bmp.Height ),
                           ImageLockMode.WriteOnly, PixelFormat.Format32bppRgb );
      bmp_binary.UnlockBits( binaryData ); // end of p2fix, unlock bmp_binary
```

Das ist ganz unlogisch, aber es funktioniert, jedenfalls so lange, bis der Garbage Collector zuschlägt.

Platz sparen mit 1-Bit-Bildern

Beenden Sie `histo1`.
Es ist eine groteske Speicherplatzverschwendung, wenn man Pixel mit 32 Bit codiert, die nur schwarz oder weiß darstellen.
Die Klasse `Bitmap` stellt für solche Zwecke ein sparsames 1-Bit-Bildformat namens `Format1bppIndexed` zu Verfügung.
Das Problem ist, dass die Klasse für dieses Format weder `GetPixel`- noch `SetPixel`-Zugriffe anbietet.
Es geht hier nicht anders: Man muss mit Hilfe eines `Byte`-Pointers jeweils 8 Pixel lesen und das richtige Bit durch je ein `Byte`-Muster ausmaskieren.

Ergänzen Sie im Kopf von `Form1` unterhalb der Zeile `Graphics g, g_histo;` folgende beiden `Byte`-Arrays:

```
  Byte[]  ORmask = { 128,  64,  32,  16,   8,   4,   2,   1 };// 1 bit  each //for Version 3
  Byte[] ANDmask = { 127, 191, 223, 239, 247, 251, 253, 254 };// 7 bits each //for Version 3
```

Kommentieren Sie in `void GenerateTheHistogram()`) die beiden `bmp_binary = ...`-Befehle `//Version 1` und `//Version 2` aus und aktivieren Sie den Befehl `//Version 3`. Es muss so aussehen:

```
    //bmp_binary = new Bitmap( bmp.Width, bmp.Height, PixelFormat.Format32bppRgb  ); //Version 1
    //bmp_binary = new Bitmap( bmp.Width, bmp.Height, PixelFormat.Format32bppRgb  ); //Version 2
    bmp_binary = new Bitmap( bmp.Width, bmp.Height, PixelFormat.Format1bppIndexed ); //Version 3
```

Kommentieren Sie in der Funktion `protected override void OnMouseMove(...)` alles zwischen den beiden Kommenterzeilen mit zwei Kommentarklammern "`/*`" und "`*/`" aus. Etwa so:

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

Ersetzen Sie den auskommentierten Code durch:

```
    //Version 3 (fast pointers creating a 1-bit binary image)***************
    unsafe
    { //lock bmp_binary from being shifted in memory by the garbage collector
      binaryData = bmp_binary.LockBits( new Rectangle( 0,0,bmp.Width,bmp.Height ),
                           ImageLockMode.WriteOnly, PixelFormat.Format1bppIndexed );
      Byte* p2fix, p2row, p2run;         // pointers to binaryData = output image
      fixed ( Byte* p1fix = grayarray )  // lock grayarray = input image in memory
      { Byte* p1run = p1fix;             // running pointer to grayarray
        p2fix = (Byte*)binaryData.Scan0; // pointer to output image
        for ( int y=0; y < bmp.Height; y++ )
        { p2row = p2fix + y * binaryData.Stride; //p2row points to first byte in row y
          for ( int x=0; x < bmp.Width; x++ )
          { p2run = p2row + x / 8;
            if ( *p1run++ > threshold ) *p2run |=  ORmask[ x % 8 ];//set    1 bit
            else                        *p2run &= ANDmask[ x % 8 ];//remove 1 bit
          } // end of for x
        }   // end of for y
      }     // end of fixed, end of p1fix, unlock grayarray
      bmp_binary.UnlockBits( binaryData ); // end of p2fix, unlock bmp_binary
    } // end of unsafe
    //End of Version 3 ****************************************************
```

Klicken Sie `Debug` → `Start Without Debugging Ctrl F5`.
Wie bei Version 2 kann man auch hier die beiden `BinaryData`-Befehle ein für alle Mal außerhalb jeder Binarisierung ausführen. Siehe Ende des vorigen Absatzes.

Schneller mit DualCore CPUs und Threads

Bildverarbeitung programmieren mit Threads gibt nicht viel Sinn, wenn man nur einen Prozessor hat. Selbst wenn man die Arbeit in zwei Threads verpackt, muss die CPU beide nacheinander ausführen, was nicht viel bringen kann.
Aber für die neuen DualCore-Prozessoren von Intel **www.intel.com** und AMD **www.amd.com** macht es Sinn, die Arbeit in zwei Threads aufzuteilen. Man teilt das Bild in eine obere Bildhälfte und eine untere Bildhälfte und startet jeweils einen Thread und erhält die doppelte Geschwindigkeit bei der Binarisierung. Der Code läuft natürlich auch auf alten SingleCore-CPUs, aber er ist da nur minimal schneller.
Erproben Sie die Threading-Version (funktionell identisch mit Version 3 = Pointer und Ausgabe von 1-Bit-Bildern) unter: **CIPCisHisto_Code_Thread.htm**.

Beispielbilder

Im Prinzip sollte das Programm alle Bildformate `BMP, ICO, GIF, JPG, PNG, TIFF` lesen und anzeigen.
Falls Sie eine alte Graphikkarte mit 8 Bit benutzen und/oder falls Sie Ihren Desktop auf 256 Farben eingestellt haben, kann es sein, dass die Farben ganz schlecht aussehen.
Falls Sie keine *.bmp - Dateien auf Ihrer Harddisk finden, benutzen Sie folgende Beispielbilder:

Download: Butterfly.bmp 217 kB 24Bit-TrueColor-Bild
Download: Madonna.bmp 18 kB 8Bit-Grauwert-Bild
Download: Lena256.bmp 66 kB 8Bit-Grauwert-Bild
Download: Lena512.bmp 258 kB 8Bit-Grauwert-Bild
Download: Angiography.bmp 66 kB 8Bit-Grauwert-Bild

Weitere Aufgaben

Klicken Sie auf `Help` in der Menüleiste von Visual Studio. Klicken Sie auf das Untermenü `Index`.
Gehen Sie in das Feld `Filtered by:` und wählen Sie dort `.NET Framework`. Dann geben Sie im Feld `Look for:` folgende Schlüsselworter ein und lesen Sie die Texte:
`PixelFormat enumeration`
`Bitmap → Bitmap class → constructor`, suchen Sie den C#-Konstruktor `public Bitmap(int, int, PixelFormat);`
`arrays, multidimensional → C# Programmer's Reference`
`Color structure → all members`, suchen Sie die `Public Properties: Color.R, Color.G, Color.B`.

Verändern Sie die Position und Höhe von `histo_r`. Passen Sie die Höhennormierung des Histogramms an.
Programmieren Sie drei zusätzliche Histogrammfenster getrennt für die drei Farben rot, grün und blau.
Programmieren Sie so, um, dass man die Originalpixel unterhalb des Schwellwerts sieht und dass nur die Pixel über der Schwelle auf weiß gesetzt werden. Benutzen sie dazu die Methode `Color.FromArgb(int, int, int);`
Programmieren Sie auch den umgekehrten Fall, wo nur die Pixel unterhalb der Schwelle auf schwarz gesetzt werden.
Erfinden Sie neue Varianten des Programms in Form von neuen Projekten `histo2, histo3` usw. nach obigem Muster.