Course IPC7, C2: Histogram, Bauanleitung mit C++/MFC7.0

Copyright © by V. Miszalok, last update: 03-10-2002

Projekt histo1 mit leerem Fenster

`Microsoft Visual Studio.NET starten`
`File – New – Project – Project Types: Visual C++ Projects, Templates: MFC Application`
Name: `histo1`
Location: `C:\temp`
Button `OK` unten Mitte klicken
Es meldet sich der `MFC Application Wizard - histo1` mit der Seite `Overview`, die uninteressant ist.
Links unter `Overview` auf `Application Type` klicken.
Schritt1: **`Single document`** einschalten
Schritt 2: **`Document/View architecture support`** Checkbox **einschalten**
Schritt 3: `Finish`

Präprozessorbefehle und Deklarationen in histo1Doc.h und histo1View.h

Klicken Sie im Hauptmenu von VS.NET auf den Menupunkt `View` und dann auf den Unterpunkt `Class View Ctrl+Shift+C`.
Sie sehen im rechten Bereich des VS.NET-Hauptfensters das Unterfenster `Class View - histo1`.
An dessen unteren Rand die Registerkarte Klassen klicken.
Darin sehen Sie die Zeile `+ histo1` , klicken Sie das Pluszeichen.
Sie sehen u.a. `+ Chisto1Doc`, doppelklicken Sie diesen Klassennamen.
Sie editieren jetzt das File `histo1Doc.h`, Schreiben Sie dort vor die Klasse `class Chisto1Doc : public CDocument` am besten unter die Zeile `#pragma once` die Präprozessoranweisungen:

```
#include < vector > //für die dynamischen Arrays der STL
```

Schreiben Sie in die Klasse direkt unter die sich öffnende geschweifte Klammer die Deklarationen:

```
public:
  BITMAPFILEHEADER    FH;
  BYTE                IBytes[1200];//bytes for BitmapInfoHeader+Palette
  BITMAPINFOHEADER*   pIH;         //pointer on BitmapInfoHeader
  BITMAPINFO*         pI;          //pointer on BitmapInfo
  std::vector< BYTE > Pixel;       //dyn. array for pixel
  std::vector< BYTE > PixelBinary; //dyn. array for binary pixel
  int                 Histogram[256];
```

Doppelklicken Sie auf den Konstruktor der Klasse: `Chisto1Doc(void)`. Sie editieren jetzt das File `histo1Doc.cpp.` Sie müssen dort vier Variable initialisieren:

```
Chisto1Doc::Chisto1Doc()
{  memset( IBytes, 0, sizeof(IBytes) );
   pIH = (BITMAPINFOHEADER*) IBytes;
   pI  = (BITMAPINFO*)       IBytes;
}
```

Im `Class View - histo1`- Fenster doppelklicken Sie den Klassennamen `Chisto1View`.
Sie editieren jetzt das File `histo1View.h`, Schreiben Sie dort vor die Klasse `class Chisto1View : public CView` am besten unter die Zeile `#pragma once` die Präprozessoranweisungen:

```
#include < vector > //für die dynamischen Arrays der STL
```

Schreiben Sie in die Klasse direkt unter die sich öffnende geschweifte Klammer die Deklarationen:

```
private:
  CRect histo_r;
  int   threshold;
  BOOL  MouseFlag;
```

Im `Class View - histo1`- Fenster klicken Sie auf das Pluszeichen vor `Chisto1View`. Klicken Sie den Konstruktor der Klasse: `Chisto1View(void)`. Sie editieren jetzt das File `histo1View.cpp.` Sie müssen dort eine Variable initialisieren:

```
Chisto1View::Chisto1View()
{   MouseFlag = false;
}
```

Prüfen, ob alles soweit in Ordnung ist: `Debug -> Start Without Debugging`. Beenden.

Code für Serialize in Chisto1Doc

Doppelklicken Sie auf `Serialize(CArchive& ar)`. Ersetzen Sie die fast leere Funktionshülse durch folgenden Code:

```
void Chisto1Doc::Serialize(CArchive& ar)
{ if (ar.IsStoring()) {}
  else
  {  ar.Read( & FH, sizeof(BITMAPFILEHEADER) );
     if ( FH.bfType   != 'MB') { forget_it(); return; }
     if ( FH.bfSize    <= 54 ) { forget_it(); return; }
     if ( FH.bfOffBits <  54 ) { forget_it(); return; }

     int nBytesInfo  = FH.bfOffBits - sizeof(BITMAPFILEHEADER);
     int nBytesPixel = FH.bfSize    - FH.bfOffBits;

     ar.Read( IBytes, nBytesInfo ); //BitmapInfoHeader+Palette

     if ( !(pIH->biBitCount == 8 ||
            pIH->biBitCount ==24) ) { forget_it(); return; }

     Pixel      .resize( nBytesPixel );
     PixelBinary.resize( nBytesPixel );

     ar.Read( &Pixel.front(), nBytesPixel );

     memset( Histogram, 0, sizeof(Histogram) ); //Null setzen
     int sum, i, hmax = 0;
     std::vector< BYTE >::iterator pointer;

     switch ( pIH->biBitCount )
     {  case 8:  for ( pointer=Pixel.begin(); pointer < Pixel.end(); pointer++ )
                    Histogram[ *pointer ]++;
                 break;
        case 24: for ( pointer=Pixel.begin(); pointer < Pixel.end(); pointer+=3 )
                 {  sum = *pointer + *(pointer+1) + *(pointer+2);
                    Histogram[ sum / 3 ]++;
                 }
      }
      for ( i = 0; i < 256; i++ ) if ( Histogram[i] > hmax ) hmax = Histogram[i];
      for ( i = 0; i < 256; i++ ) Histogram[i] = (100*Histogram[i])/hmax;
    }
}
```

Klicken Sie im `Class View - bimap1` - Fenster mit der rechten Maustaste auf die Klasse `Chisto1Doc`. Es öffnet sich ein Kontextmenü. Stellen Sie die Maus auf den 4 Menüpunkt `Add` und klicken Sie dann auf `Add Function...` . Es erscheint der "`Add Member Function Wizard - histo1`". Tragen Sie in seine Dialogfelder ein:
`Return type : void`
`Function name: forget_it`
`Access : private`
`.cpp file : histo1doc.cpp`
Alle weiteren Felder und Checkboxen bleiben leer. Sie verlassen den Wizard mit dem Button `Finish`.

Der Wizard erzeugt Ihnen einen Funktionshülle, die Sie folgendermaßen füllen:

```
void Chisto1Doc::forget_it()
{ memset( IBytes, 0, sizeof(IBytes) );
  for ( int i=0; i < 10; i++ ) MessageBeep(-1);
}
```

Ausführen, lesen muss funktionieren, aber man sieht noch nichts. Falls man eine Datei öffnet, die kein oder ein falsches Bitmap enthält, muss es piepsen.

Code für OnDraw in Chisto1View

Nun programmieren Sie in `histo1View.cpp` die bereits vorhandene Funktion: `void Chisto1View::OnDraw(CDC* /*pDC*/)` bis sie so aussieht:

```
void Chisto1View::OnDraw(CDC* pDC)
{  Chisto1Doc* pDoc = GetDocument();
   if ( !pDoc->pIH->biSize ) { pDC->TextOut(0,0,"Open a *.BMP file !"); return; }

   BYTE * pointer;
   if ( !MouseFlag ) pointer = &(pDoc->Pixel      .front());
   else              pointer = &(pDoc->PixelBinary.front());

   CRect R; GetClientRect( R ); //welche Größe hat die ClientArea?
   StretchDIBits( pDC->GetSafeHdc(),
                  0, 0, R.Width(), R.Height(),
                  0, 0, pDoc->pIH->biWidth, pDoc->pIH->biHeight,
                  pointer, pDoc->pI,
                  DIB_RGB_COLORS, SRCCOPY );
   histo_r.right  = R.Width() - 10;
   histo_r.left   = histo_r.right - 256;
   histo_r.bottom = R.Height() - 10;
   histo_r.top    = histo_r.bottom - 100;
   pDC->Rectangle( histo_r );
   pDC->TextOut( histo_r.left+1, histo_r.top+1, "click and move here!" );
   for ( int i = 0; i < 256; i++ )
   {  pDC->MoveTo( histo_r.left + i, histo_r.bottom );
      pDC->LineTo( histo_r.left + i, histo_r.bottom - pDoc->Histogram[i] );
   }
   if ( MouseFlag )
   {  pDC->MoveTo( histo_r.left + threshold, histo_r.top );
      pDC->LineTo( histo_r.left + threshold, histo_r.bottom );
   }
}
```

`histo1` ist vorläufig fertig, ausfuehren, `*.bmp`-File öffnen, Sie sehen das Histogramm, aber Sie koennen noch kein Binärbild erzeugen, beenden.

Mausereignisse in CHisto1View

Klicken Sie im Klassenbaum mit der rechten Maustaste auf `Chisto1View`.
Es öffnet sich ein Kontextmenü. Klicken Sie ganz unten im Kontextmenü auf `Properties`.
Es öffnet sich unter dem `Class View - histo1` - Fenster ein `Properties` - Fenster mit der Überschrift `Chisto1View VCCodesClass`.
Im Toolbar dieses Fensters klicken Sie rechts neben dem gelben Blitz auf das `Messages` - Symbol, das aussieht wie ein weißer Topf mit blauem Deckel.
Sie sehen nun die Liste der Windows-Messages von `Chisto1View`.
Klicken Sie auf `WM_LBUTTONDOWN` und dann auf das schwarze Abwärtsdreieck rechts in dieser Zeile und dann auf `Add OnLButtonDown`.
Klicken Sie auf `WM_MOUSEMOVE` und dann auf das schwarze Abwärtsdreieck rechts in dieser Zeile und dann auf `Add OnMouseMove`.
Klicken Sie auf `WM_LBUTTONUP` und dann auf das schwarze Abwärtsdreieck rechts in dieser Zeile und dann auf `Add OnLButtonUp`.

Programmieren Sie die eben generierten Funktionen bis sie so aussehen:

```
void Chisto1View::OnLButtonDown(UINT nFlags, CPoint point)
{  if ( !histo_r.PtInRect( point ) ) return;
   MouseFlag = true;
}

void Chisto1View::OnMouseMove(UINT nFlags, CPoint point)
{  if ( !nFlags ) return;
   if ( !histo_r.PtInRect( point ) ) return;
   Chisto1Doc* pDoc = GetDocument();
   if ( !pDoc->pIH->biSize ) return;
   threshold = point.x - histo_r.left;

   std::vector< BYTE >::iterator pointer1 = pDoc->Pixel      .begin();
   std::vector< BYTE >::iterator pointer2 = pDoc->PixelBinary.begin();
   switch ( pDoc->pIH->biBitCount )
   {  case  8: for ( ; pointer1 < pDoc->Pixel.end(); pointer1++, pointer2++ )
               {  if ( *pointer1 > threshold ) *pointer2 = 255;
                  else                         *pointer2 =   0;
               }
               break;
      case 24: for ( ; pointer1 < pDoc->Pixel.end(); pointer1+=3, pointer2+=3 )
               {  int sum = *pointer1 + *(pointer1+1) + *(pointer1+2);
                  if ( sum > 3*threshold )
*pointer2=*(pointer2+1)=*(pointer2+2)=255;
                  else                    *pointer2=*(pointer2+1)=*(pointer2+2)=
0;
               }
   }
   Invalidate( false );
}

void Chisto1View::OnLButtonUp(UINT nFlags, CPoint point)
{  MouseFlag = false;
   Invalidate( false );
}
```

`histo1` ist fertig, ausfuehren, Bitmaps von Harddisk laden, ein Bild öffnen. Kllicken Sie mit der linken Maustaste in das Histogrammrechteck und verschieben Sie die Schwelle. Sie sehen dabei Binärbilder. Nach dem Loslassen der linken Maustaste erscheint wieder das Originalbild.
Die Binärbilder erscheinen nur ruckig langsam. Das Berechnen der Binärbilder geht viel schneller und synchron zur Mausbewegung, wenn Sie das Projekt ohne Debugger im `release` - mode übersetzen. Gehen Sie dazu im Hauptmenü von VS auf `Build -> Configuration Manager`. Es öffnet sich ein Fenster `Configuration Manager`. Sie setzen das Feld `Active Solution Configuration` auf "`Release`" und die Spalte `Configuration` auch auf "`Release`". Verlassen mit `Close`. Dann alles neu übersetzen, linken und ausführen mit `Debug -> Start Without Debugging`.
Erpoben, beenden, löschen, neu programmieren.

Beispielbilder

Im Prinzip sollte das Programm alle Arten von Bitmaps 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 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