Courses IPCx, C2: Histogram, Code Comments

In `histo1Doc.h` in front of `class CHisto1Doc : public CDocument`

`#include < vector >` //declares the dynamic arrays of the Standard Template Library STL. The blanks inside the < > clause around the word `vector` may be suppressed. They are just necessary in a HTML-Document otherwise HTML treats the word `vector` as HTML-Tag.

`BITMAPFILEHEADER FH;` //structure of overall length = 14 bytes containing 5 variables (see MSDN and C6_Bitmap_FAQs)

`BYTE IBytes[1200];` //untyped space for 1200 bytes for BitmapInfoHeader+Palette. You don't know the types of BitmapInfoHeaders you will read and you don't know if there will be palettes or not. You reserve 1200 bytes for the worst case: BITMAPV5HEADER (=136 byte) plus 256 palette entries (=1024 bytes). 1160 bytes are wasted in case of traditional 24-Bit-Bitmaps but you can probably afford that.

`BITMAPINFOHEADER* pIH;` //typed pointer allows the access to biSize, biWidth etc.

`BITMAPINFO* pI;` //This typed pointer is needed by function `StretchDIBits`. Structure BITMAPINFO (see MSDN and C6_Bitmap_FAQs) is longer than BITMAPINFOHEADER but it contains in its first part the structure BITMAPINFOHEADER completely . It allows access to both the BitmapInfoHeader and (if `biClrUsed > 0`) the palette. If `biClrUsed > 0` then `StretchDIBits` automatically accesses and uses the palette with the help of this pointer.

`std::vector< BYTE > Pixel;` //name of a dyn. array of bytes aimed to store the original pixels

`std::vector< BYTE > PixelBinary;` //name of a dyn. array of bytes for pixels after thresholding and binarisation

`int Histogram[256];` //space for 256 integers

In `histo1Doc.cpp` inside the constructor `CBitmap1Doc()` you have to initialize 3 variables:

`memset( IBytes, 0, sizeof(IBytes) );` // clear the 1200 untyped bytes in order to mark the absence of an image. Function `CHisto1View::OnDraw()` reads from here if you already loaded an image or not.

`pIH = (BITMAPINFOHEADER*) IBytes;` //The typed pointer points now to the the head of IBytes[1200]

`pI = (BITMAPINFO*) IBytes;` //The typed pointer points now to the the head of IBytes[1200] as `pIH` does.

In `histo1View.h` you have to declare the following private variables of `class CHisto1View : public CView`

`CRect histo_r;` //rectangle (256x100) to draw the histogram in the right lower corner of the image

`int threshold;` //interactive threshold separating the black background from the white foreground

`BOOL MouseFlag;` //used to remember if a threshold has been choosen via the left mouse button

In `histo1View.cpp` inside the constructor `CHisto1View()` you must intialize a variable

`MouseFlag = false;` Indicates, that there is no threshold yet.

In `histo1Doc.cpp` inside `Serialize(CArchive& ar)` inside the `else` clause: Code for reading an image and for computing its histogram

`ar.Read( & FH, sizeof(BITMAPFILEHEADER) );` //Read 14 bytes from the harddisk.

`if ( FH.bfType != 'MB') { forget_it(); return;` //The first 2 bytes form the reversed string of BM.

`if ( FH.bfSize <= 54 ) { forget_it(); return;` //A bitmap file contains at least 55 bytes.

`if ( FH.bfOffBits < 54 ) { forget_it(); return;` //The shortest possible headers need 54 bytes.

`int nBytesInfo = FH.bfOffBits - sizeof(BITMAPFILEHEADER);` //That is the space left for the BitmapInfoHeader and palette.

`int nBytesPixel = FH.bfSize - FH.bfOffBits;` //That is the space for all the pixels.

`ar.Read( IBytes, nBytesInfo );` //Read BitmapInfoHeader+palette from harddisk

`if ( !(pIH->biBitCount == 8 || pIH->biBitCount ==24) ) { forget_it(); return; }` //Ignore all 1, 4, 16 and 32 bit bitmaps. `!` = logical NOT and `||` = logical OR.

`Pixel .resize( nBytesPixel );` //Keep free the necessary space for all the original pixels in main memory.

`PixelBinary.resize( nBytesPixel );` //Double the space for storing the black and white pixels.

`ar.Read( &Pixel.front(), nBytesPixel );` //Read all original pixels from the harddisk into the main memory starting at the first adress of the dynamic byte array named `Pixel`.

`memset( Histogram, 0, sizeof(Histogram) );` //Clear the 256 histogram integers to zero values.

`int sum, i, hmax = 0;` //some local variables, one initialized by zero

`std::vector< BYTE >::iterator pointer;` //declaration of a pointer pointing arbitrarily into a dyn. byte array. Such special pointers are called iterators.

`switch ( pIH->biBitCount )` //jump depending on biBitCount, which can be 8 or 24.

`case 8: for ( pointer=Pixel.begin(); pointer < Pixel.end(); pointer++ )` //Loop through all pixels of a 8-bit-bitmap.

`Histogram[ *pointer ]++;` //Take the gray value (or the color index) and increment one of the counters of the array `Histogram`.

`break;` //This is a jump out of and beyond the end of the `switch` clause. If you forget this statement, the program continues with case 24 and computes a strange `Histogram`.

`case 24: for ( pointer=Pixel.begin(); pointer < Pixel.end(); pointer+=3 )` //Loop through all pixels of a 24-bit-bitmap.

`sum = *pointer + *(pointer+1) + *(pointer+2);` //Add the 3 values for blue, green and red.

`Histogram[ sum / 3 ]++;` //Divide the sum by 3 and increment one of the counters of the array `Histogram`.

`for ( i = 0; i < 256; i++ ) if ( Histogram[i] > hmax ) hmax = Histogram[i];` //Find out hmax = the most common gray value of the image = highest column of `Histogram`.

`for ( i = 0; i < 256; i++ ) Histogram[i] = (100*Histogram[i])/hmax;` //Scale hmax to 100 and all other columns of `Histogram` linearily.

In `histo1Doc.cpp` inside `void CHisto1Doc::forget_it()()` //Small private member function of `CHisto1Doc`

`memset( IBytes, 0, sizeof(IBytes) );` //If this was not a reasonable bitmap, then clear all 1200 bytes of IBytes to zero.

`for ( int i=0; i < 10; i++ ) MessageBeep(-1);` //This is a loud protest against misuse.

In `histo1View.cpp` inside `void CHisto1View::OnDraw(CDC* pDC)` //

`CHisto1Doc* pDoc = GetDocument();` //This line was prepared by Visual Studio. Keep it. It give us a pointer to reach the variables which have been declared in class `CHisto1Doc`

`if ( !pDoc->pIH->biSize ) { pDC->TextOut(0,0,"Open a *.BMP file !"); return; }` //Advice to users who do not know what they should do after they started the program.

`BYTE * pointer;` //Declares a local pointer.

`if ( !MouseFlag ) pointer = &(pDoc->Pixel.front());` //If nobody presses the left mouse button upon the rectangle showing the histogram in the right lower orner of the image , set the pointer to the first pixel of the original image.

`else pointer = &(pDoc->PixelBinary.front());` //If we have a threshold, set the pointer to first pixel of the binary image.

`CRect R; GetClientRect( R );` //Find out the current dimensions of the client area that could be covered with the image.

`StretchDIBits( pDC->GetSafeHdc(),` //Draw the image into the graphics board and on the screen. The first parameter has to be a handle to the Device Context.

`0, 0, R.Width(), R.Height(),` //Parameters 2,3,4,5 of `StretchDIBits` indicate the rectangular destination dimensions.

`0, 0, pDoc->pIH->biWidth, pDoc->pIH->biHeight,` //Parameters 6,7,8,9 of `StretchDIBits` indicate the rectangular original source dimensions.

`pointer, pDoc->pI,` //Parameter 10 has to point to the first pixel and parameter 11 must be a typed pointer to a BitmapInfo-structure (BitmapInfoheader plus palette).

`DIB_RGB_COLORS, SRCCOPY );` //Parameters 12 and 13 are constants which influence the use of colors and transparency (see MSDN).

`histo_r.right = R.Width() - 10;` //Right border of the histogram box near the right side of the image.

`histo_r.left = histo_r.right - 256;` //Left border is 256 pixels left of the right border.

`histo_r.bottom = R.Height() - 10;` //Bottom of the box near the bottom of the image.

`histo_r.top = histo_r.bottom - 100;` //Top of the box is 100 pixel above the bottom.

`pDC->Rectangle( histo_r );` //Draw a white rectangle surrounded by a black border.

`pDC->TextOut( histo_r.left+1, histo_r.top+1, "click and move here!" );` //Display this text inside the rectangle.

`for ( int i = 0; i < 256; i++ )` //Draw one perpendicular black line for each of the 256 entries of `Histogram`

`pDC->MoveTo( histo_r.left + i, histo_r.bottom );` //from bottom

`pDC->LineTo( histo_r.left + i, histo_r.bottom - pDoc->Histogram[i] );` //in upward direction.

`if ( MouseFlag )` //If somebody had pressed the left mouse button over the histogram box, you have to visualize the position of the interactive threshold.

`pDC->MoveTo( histo_r.left + threshold, histo_r.top );` //Show the current threshold as black line from bottom

`pDC->LineTo( histo_r.left + threshold, histo_r.bottom );` //to top

In `void CHisto1View::OnLButtonDown(UINT nFlags, CPoint point)` //The user presses the left mouse button.

`if ( !histo_r.PtInRect( point ) ) return;` //There is nothing to do, if the mouse is not over the histogram.

`MouseFlag = true;` //Inform `OnDraw` to display the binary image instead of the original one.

In `void CHisto1View::OnMouseMove(UINT nFlags, CPoint point)` //This function computes the current threshold and the current binary image and calls `OnDraw` via `Invalidate()`

`if ( !nFlags ) return;` //Do nothing if the user moves the mouse without pressing a button.

`if ( !histo_r.PtInRect( point ) ) return;` //Do nothing if the mouse is not over the Histogram.

`CHisto1Doc* pDoc = GetDocument();` //You need a pointer to class `CHisto1Doc` in order to access the BitmapInfoHeader and the pixels.

`if ( !pDoc->pIH->biSize ) return;` //There is no image at all.

`threshold = point.x - histo_r.left;` //The threshold depends on the horizontal position relative to the left border of the histogram box.

`std::vector< BYTE >::iterator pointer1 = pDoc->Pixel .begin();` //Pointer to the first pixel of the original image

`std::vector< BYTE >::iterator pointer2 = pDoc->PixelBinary.begin();` //Pointer to the first pixel of the binary image

`switch ( pDoc->pIH->biBitCount )` //Jump depending on biBitCount which may be 8 or 24.

`case 8: for ( ; pointer1 < pDoc->Pixel.end(); pointer1++, pointer2++ )` //Go through all pixels of the 8-bit-image

`if ( *pointer1 > threshold ) *pointer2 = 255;` //If the grey value is above the threshold, make it completly white.

`else *pointer2 = 0;` //Otherwise set it black.

`break;` //Jump out of and beyond the end of the `switch`-clause.

`case 24: for ( ; pointer1 < pDoc->Pixel.end(); pointer1+=3, pointer2+=3 )` //Go through all RGBTriples of the original image

`int sum = *pointer1 + *(pointer1+1) + *(pointer1+2);` //Add the red, green and blue values together.

`if (sum > 3*threshold) *pointer2=*(pointer2+1)=*(pointer2+2)=255;` //If the sum is > 3 thresholds, set the RGBTRIPLE to white,

`else *pointer2=*(pointer2+1)=*(pointer2+2)= 0;` //otherwise set it to black.

`Invalidate( false );` //Ask the operating system to redraw the client area. Erasing the former content is not necessary.

In `void CHisto1View::OnLButtonUp(UINT nFlags, CPoint point)` //The user releases the mouse button.

`MouseFlag = false;` //No more thresholding and binary images.

`Invalidate( false );` //Redraw the original image.