2D Fast Wavelet Transform Library for Image Processing

Licence	GPL3
First Posted	**14 Oct 2007**
Views	**83,803**
Downloads	**1,831**
Bookmarked	**66 times**

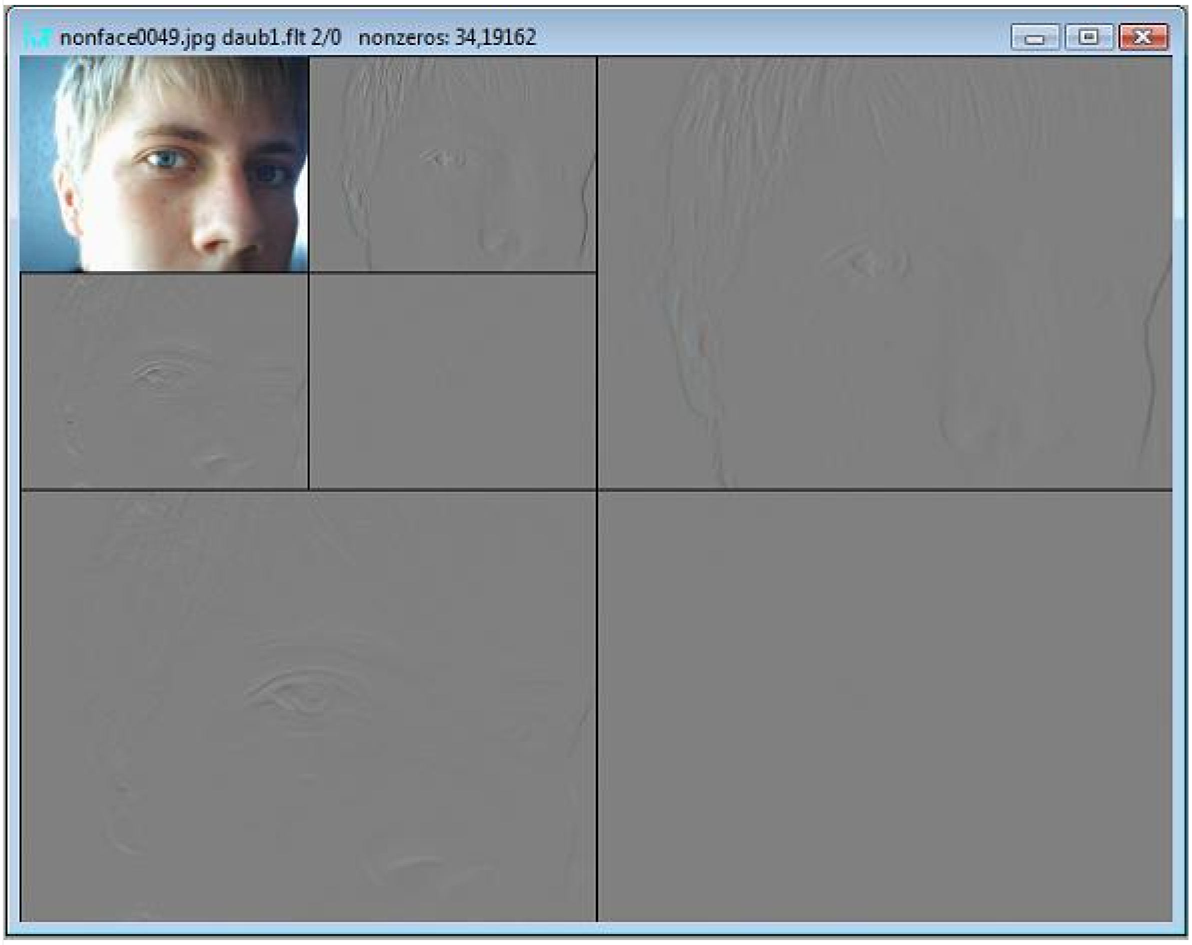
By **Chesnokov Yuriy** | 19 Oct 2007

VC8.0 .NET2.0 VS2005 C++/CLI Windows MFC Dev Intermediate

This article demonstrates the use of 2D Fast wavelet transform for image processing

★★★★★ 4.84 (24 votes)

- Download demo - 37.94 KB
- Download source - 36.57 KB



Introduction

I've been involved with wavelet-analysis since my Ph.D studies and over the years developed various wavelet-transforms C++ libraries. This one concerns 2D implementation of the Fast wavelet transform (FWT). The 2D FWT is used in image processing tasks like image compression, denoising and fast scaling. I intended to design the implementation of the 2D FWT with custom filter support and simple in usage. I've equipped it with known filter families: Daubechies, Coiflets and Biorthogonal wavelets. The wavelets used in compression tasks are Bior53 and Bior97. With Daub1 filter you can achieve fast dyadic image scaling algorithm. You may also experiment with other filters for compression and denoising and compare the results, you may also add your custom wavelets. The nice feature of this library is that reconstruction FWT filters are rearranged the way that you perform just convolution operations on the spectrum, which is very important if you'd like MMX optimizations.

Background

You need basic understanding of wavelet transforms to be familiar with the topic discussed. You can learn it from numerous sources available on the Internet, also from Matlab Wavelet Toolbox documentation or have a look at this website. Also some C++ experience with abstract base classes is desirable if you plan to extend the code by deriving your own implementations, numerically optimized for a particular filter.

Using the Code

The GUI for the library has been designed by me using Windows Forms with .NET 2.0 Framework, so anyone can start using it right now without much ado. It provides image visualization, 2D `FWT` transformation in RGB color space and synthesis. Just unzip the binary file with the *\filters* directory and run it. Load the image using the File menu, and transform it with the Transforms menu. After clicking `FWT2D` RGB transform, you'll be presented with a translucent dialog box, where you can select the filter (present in the *\filters* dir), select the number of scales (usually 3 for image transform) and denoising threshold (in the range of 0 to 128) below which, all the pixels at the high frequency bands will be zeroed. In the `FWT` library this threshold is decreased 4 times every scale so you do not end up with too much distortion at the reconstructed image. The caption of the image frame is headed with the percent of non-zero coefficients at the `FWT` spectrum. After reconstruction you are also presented with the error in dB compared to the original, untransformed image.

The C++ library itself is composed of several classes:

- `BaseFWT2D` - The abstract base class for the transform
- `FWT2D` - The implementation of the 2D FWT, it is a derived class from `BaseFWT2D`
- `vec1D` - The 1D vector wrapper for the filters

The 2D FWT transform and synthesis functions are defined in the `BaseFWT2D` (`trans()` and `synth()`) and they use private virtual functions for transforming and synthesizing rows and columns of the image which are implemented in the `FWT2D` class. I've designed such an implementation that anyone can derive its own optimized versions of the FWT transform for a particular filter I've mentioned in the Background section. I have written the implementations of MMX optimized versions of Bior53, Bior97 for image compression and Daub1 for fast image scaling and plan to post them in the future in addition to this article.

Quick Start with the FWT Library

To start with the 2D FWT library, you just need to initialize `FWT2D` object with the filter name, providing full path to its location, check the status after the constructor has been called, initialize desired `width` and `height` of the single channel of the image (you need to arrange separate channels, from [RGB] stream copy just R channel to the image `Width` x `Height` buffer, also the same for G and B channels, you may also convert them to YUV space or the like, but then you need separate buffers for Y, U and V channels also) and continue with transformation and synthesis. This way every class object is targeted to the specific wavelet filter and image channel (you may use different filters for different channels then).

```
/*
unsigned char* pRchannel = new unsigned char[width * height];
for (unsigned int i = 0; i < width * height; i++) {
        //Taking R channels from RGB buffer
        pRchannel[i] = pRGB[3*i + 0];
}
*/

unsigned char* pRchannel;  //R channel buffer unsigned char 0...255
unsigned int width;        //R channel width
unsigned int height;       //R channel height

const wchar_t* perr;       //error text message returned after status() function
unsigned int status;       //numerical error value

unsigned int scales = 3;
unsigned int TH = 20;
```

```
FWT2D r_fwt(L"path\\bior97.flt");
r_fwt.init(width, height);
perr = r_fwt.status(status);
if (perr != 0) {
        //Output text representation of the error pointed by perr
        return;
}

//After initialization you can do multiple trans() and synth() calls
//with upcoming image data for analysis or FWT spectrum for synthesis
r_fwt.trans(pRchannel, scales, TH);
r_fwt.synth(pRchannel);
//and so on ...

//clean up before destroying the object
r_fwt.close();
```

If you want to access the FWT spectrum after `trans()` call for visualization or entropy coding, you can do it with `BaseFWT2D` functions `getspec()` or `getspec2d()`. The first one provides 1D `char` pointer to the spectrum and the later 2D `char` pointer so you can access the spectrum as a two dimensional array. Note, that the original color channels are converted to `char` by subtracting 128 and the FWT spectrum is in `char` interval too -128 ... 127.

```
char** pspec = r_fwt.getspec2d();
for (unsigned int j = 0; j < height; j++ ) {
        for (unsigned int i = 0; i < width; i++ ) {
                char val = pspec[j][i];
        }
}
```

Detailed Reference to the FWT Library

The nice 1D vector wrapper is implemented in the `vec1D` class. It allows defining the starting index of the first element in the array. By default, it is 0, as in usual C array, but you can also define positive one, for example 1, to end up with Matlab like array, or negative one, as it is with wavelet filters. If you define -3 as the starting index and 6 as the length of the array (total number of items) you will access the elements of the array with the indices: -3, -2, -1, 0, 1, 2. This way your array is centered around 0, which coincides with particular wavelet filter center for example. The array data itself is 16 bit aligned for performing SSE optimized floating point operations.

```
//plain C array of 6 elements with indices from 0 to 5 initialized to 0.0f
unsigned int size = 6;
vec1D vec1(size);
vec1(0) = 1.0f; //Set the first element to 1.0

//Initialize the array data with external buffer
unsigned int offset = -3;
float data[] = {-3.0f, -2.0f, -1.0f, 0.0f, 1.0f, 2.0f};
vec1D vec2(size, offset, data);

//Print out the array contents
for (int i = vec2.first(); i <= vec2.last(); i++ ) {
        wprintf(L"%i %f\n", i, vec2(i));
}
```

The abstract base class `BaseFWT2D` provides all the necessary functions for transforming and reconstructing the image, getting the spectrum, dumping loaded wavelet filters, retrieving the number of scales in the spectrum. Start by constructing the object of `FWT2D` class which publicly derives from `BaseFWT2D`.

-
  ```
  FWT2D::FWT2D(const w_char* filter_name);
  ```

-
  ```
  FWT2D::FWT2D(const wchar_t* fname, const float* tH, unsigned int thL,
      int thZ, const float* tG, unsigned int tgL, int tgZ, const float* H,
      unsigned int hL, int hZ, const float* G, unsigned int gL, int gZ);
  ```

With the first constructor you provide the filters from an external file specifying full path to it. With the second one, you can provide the filters from memory buffers. `tH` and `tG` are the low-pass and high-pass filters for image transformation, `thL` and `tgL` their length and `thZ` and `tgZ` are the first indices values (-3 for example). The same applies to `H` and `G` filters but they are used for image synthesis from the spectrum.

-
  ```
  const wchar_t* BaseFWT2D::status(int& status);
  ```

Check the status after constructors. `status` = 0 indicates success and the function returns a `NULL` pointer, otherwise inspect the returned error message.

Then you need to initialize the class object with either of the functions:

-
  ```
  void BaseFWT2D::init(unsigned int width, unsigned int height);
  ```

-
  ```
  void BaseFWT2D::init(char* data, char* tdata, unsigned int width,
      unsigned int height);
  ```

The last one provides an opportunity to supply external buffers to the class object `data` and `tdata` both of the `width` by `height` size. This way you can fill the `data` buffer from external source and proceed with:

-
  ```
  int BaseFWT2D::trans(unsigned int scales, unsigned int th = 0);
  ```

The image will be replaced with FWT spectrum in the `data` buffer in case the function returns `NULL` on success or `-1` if you have not initialized the library.

Or use the `init(unsigned int width, unsigned int height)` function for initialization and perform the image transformation supplied from external `data` buffer with two functions:

-
  ```
  int BaseFWT2D::trans(const char* data, unsigned int scales, unsigned int th = 0);
  ```

-
  ```
  int BaseFWT2D::trans(const unsigned char* data, unsigned int scales,
      unsigned int th = 0);
  ```

The first one transforms the image supplied in the data buffer of type `char`, that is your data already DC shifted and in the range -128 ? 127. The last one accepts unsigned `char` buffer and subtracts 128 from it using MMX optimized `private` function. The functions return `NULL` if they succeed, otherwise `-1` if you have not initialized the library. Note that the image data is kept intact after the transformation. `scales` defines the number of levels of FWT transform and `th` is the denoising threshold (0 ... 128).

You can access the FWT spectrum using the following functions:

-
  ```
  char* BaseFWT2D::getspec();
  ```

-
  ```
  char** BaseFWT2D::getspec2d();
  ```

With `char*` pointer you've got straight `width` by `height` array and with `char**` pointer you can access individual coefficient at (column, row) location as with 2D array pointers.

To reconstruct the image, you use the following functions:

-
  ```
  int BaseFWT2D::synth();
  ```

-
  ```
  int BaseFWT2D::synth(char* data);
  ```

-
  ```
  int BaseFWT2D::synth(unsigned char* data);
  ```

The first one just reconstruct the image from the spectrum to the class's own internal buffers (that can be supplied externally with corresponding `init()` function). The second and third reconstruct the image and copy it to `data` buffer in char -128 ... 127 or unsigned `char` 0 ... 255 format.

You can keep transforming and synthesizing the images any number of times once you created the class object and initialized it with image `width` and `height`. After you finished with the class object, call `close()` method and destroy the object.

-
  ```
  void BaseFWT2D::close();
  ```

You can get and set the number of `FWT` scales with the functions:

-
  ```
  unsigned int BaseFWT2D::getJ();
  ```

-
  ```
  void BaseFWT2D::setJ(unsigned int j);
  ```

The last one provides the opportunity to change the number of scales before synthesis, so if you've got `FWT` spectrum with 3 scales you can change it to, say, 1 and perform only one level of `FWT` synthesis.

If you are planning to extend the code yourself, you need to derive your own class from `BaseFWT2D` class and provide overrides for `transrows()`, `transcols()`, `synthrows()` and `synthcols()`. You need to be well versed with 2D FWT analysis also. It is useful if you'd like to implement MMX, SSE optimized versions for a filter of a particular length. As I mentioned before I'm planning to post SSE optimizations for Bior53 and Bior97 wavelet-filters and MMX one for Daub1 filter in the future.

Points of Interest

The tedious point to program was writing `BaseFWT2D::makeHGsynth()` function, which rearranges the synthesis filters coefficients to odd and even ones. Having these, you can substitute 2*m and 2*m+1 operations of selecting even and odd coefficients from the synthesis filters during reconstruction process and proceed with just straight convolution.

License

About the Author

Chesnokov Yuriy

Software Developer (Junior)
Ayonix (face recognition)
Russian Federation

Member

Former Cambridge University post-doc (http://www-ucc-old.ch.cam.ac.uk/research/yc274-research.html) with Research intrests in digital signal processing in medicine, image and video processing, pattern recognition, AI methods, computer vision. You may approach me for the code/research development in the above areas at http://cambridge.academia.edu/YuriyChesnokov

Publications:

Complexity and spectral analysis of the heart rate variability dynamics for distant prediction of paroxysmal atrial fibrillation with artificial intelligence methods. Artificial Intelligence in Medicine. 2008. V43/2. PP. 151-165 (http://dx.doi.org/10.1016/j.artmed.2008.03.009)

Face Detection C++ Library with Skin and Motion Analysis. Biometrics AIA 2007 TTS. 22 November 2007, Moscow, Russia. (http://www.dancom.ru/rus/AIA/2007TTS/ProgramAIA2007TTS.html)

Screening Patients with Paroxysmal Atrial Fibrillation (PAF) from Non-PAF Heart Rhythm Using HRV Data Analysis. Computers in Cardiology 2007. V.

34. PP. 459–463 (http://www.cinc.org/archives/2007/pdf/0459.pdf)

Distant Prediction of Paroxysmal Atrial Fibrillation Using HRV Data Analysis. Computers in Cardiology 2007. V. 34. PP. 455-459 (http://www.cinc.org/archives/2007/pdf/0455.pdf)

Individually Adaptable Automatic QT Detector. Computers in Cardiology 2006. V. 33. PP. 337-341 http://www.cinc.org/archives/2006/pdf/0337.pdf)

Past/recent outsourcing code/research:

www.bitcycle.com - AI consulting
www.cit.ie - augmented reality C++
www.apneicare.com - ECG processing C++/C#
www.ayonix.com - face recognition C/C++/C#/ASP.NET/MATLAB
confidential company in Australia - video capture C#/DirectShow
www.system7.co.uk - CBIR C#/ASP.NET (cbir.system7.com)
confidential enterprise in UK - pedestrian detection C++/CLI
www.trulyintelligent.com - AI consulting
www.devline.ru - video codecs C++
ecotec.kuban.ru - embedded programming in C/C++

Comments and Discussions

37 messages have been posted for this article Visit **http://www.codeproject.com/KB/graphics/2D_FWT_lib.aspx** to post and view comments on this article, or click **here** to get a print view with messages.