

C++ Wavelet Libraries

Documentation

08/21/2011

Contents

1	Overview	3
1.1	Changes from previous version	3
1.2	Wavelet Libraries in Win32 Environment	4
1.2.1	Microsoft Visual C++	4
1.2.2	GNU-GCC (MinGW) Based Compilation	4
1.3	Working in LINUX Environment	5
1.4	Licensing	6
2	List of Functions	7
2.1	1D DWT/IDWT Functions	7
2.2	1D SWT/ISWT Functions	10
2.3	2D DWT/IDWT Functions	10
2.4	2D SWT Functions	14
2.5	Convolution	14
2.6	Wavelet Filters	15
2.7	1D Vector Manipulation	15
2.8	2D Vector Manipulation	16
3	Sample Codes	17
3.1	1D DWT/IDWT Example Code	17
3.2	1D Linear vs Nonlinear Approximation	21
3.3	1D Stationary Wavelet Transform	26
3.4	2D DWT/IDWT Demo Code	29
3.5	Image Approximation Demo	35
3.6	2D SWT Demo	46

List of Figures

2.1	DWT stats (periodic extension) for an input signal of length 256	8
2.2	DWT stats (symmetric extension) for an input signal of length 256	10
2.3	Two Level SWT Decomposition of a 247 length signal vector . .	11
2.4	2D DWT computation using periodic extension	12
2.5	2D DWT computation using symmetric extension	13
3.1	1D DWT Console	19
3.2	Length 256 Piecewise Regular signal from WaveLab	20
3.3	2 Level DWT Decomposition Coefficients obtained using db2 wavelet	20
3.4	Reconstructed Signal After IDWT computations	21
3.5	Console showing db3 5 level computation	24
3.6	Length 256 Piecewise Regular signal from WaveLab	25
3.7	Retained Coefficients for Linear Approximation case	25
3.8	Reconstructed Signal using Linear Approximation	26
3.9	Retained Coefficients(Non-Linear Approximation case)	27
3.10	Reconstructed Signal(Non-Linear Approximation Case)	27
3.11	Console Running swt demo	29
3.12	Input Signal(Length-256)	30
3.13	SWT Coefficients for two level of Decomposition(Length 3 X 256)	30
3.14	SWT Reconstructed Signal	31
3.15	Input 250X189 Image	35
3.16	3-Level Decomposition using db3 wavelet	35
3.17	Reconstructed Image	36
3.18	Input Image	42
3.19	DWT of Input Image	43
3.20	2D Approximation using only 1/10th Coefficients	44
3.21	2D Approximation using only 1/50th Coefficients	45
3.22	Input Image	49
3.23	Approximation image at Level J = 3	50
3.24	Detail Images at levels J=3,2,1(L-R).Horizontal,Vertical,Diagonal(T- B)	51

Chapter 1

Overview

This release of C++ Wavelet Libraries is focused on speed and ease of use. Libraries are available on LINUX and Win32 Platforms. Available features include

- 1D DWT and IDWT Implementation (Two Modes)
- 2D DWT and IDWT Implementation (Two Modes)
- 1D SWT and ISWT Implementation (Stationary Wavelet Transform)
- 2D SWT Implementation
- Implemented using FFTW3 Library
- Shared(.so) and static(.a) libraries for Linux
- Shared(.dll) and static(.a) libraries for Win32 GCC (MinGW).
- Shared(.dll) libraries for Microsoft VC++

Dynamic Libraries are labeled wavelet2d while static libraries are labeled wavelet2s.

1.1 Changes from previous version

- FFTW3 Library is used to improve computation speed.
- No GNUPLOT outputs are generated.
- Periodic and Symmetric functions take same arguments. They will still be accessed by the same names, though.(dwt ,dwt_sym etc.)
- Focus on shared libraries. No static libraries for MSVC++ although MinGW and LINUX version will come with static as well as shared libraries.

1.2 Wavelet Libraries in Win32 Environment

1.2.1 Microsoft Visual C++

This package contains two DLLs for MSVC++ - one Debug and one Release version.

The list of required files in order to compile wavelet2d based programs-

- Header File- wavelet2d.h
- Import Library- wavelet2d.lib
- Wavelet DLL - wavelet2d.dll
- FFTW DLL - libfftw3-3.dll

Header file is in the source ('src') folder. Wavelet Libraries are in the respective Debug and Release folders. FFTW DLL is in the FFTW3 folder. Alternatively, you may chose to install FFT library from www.fftw.org. The FFTW source codes are also available at FFTW website under GNU-GPL license.

In order to use wavelet libraries, the easiest way is to add import library (wavelet2d.lib) to additional dependency in your project which in turn will handle the DLLs for you. The two DLLs must be in your program path. If you are new to MSVC, you may want to learn more about DLL search path at

[http://msdn.microsoft.com/en-us/library/7d83bc18\(VS.71\).aspx](http://msdn.microsoft.com/en-us/library/7d83bc18(VS.71).aspx)

You may also chose to directly call DLLs from your program. Regardless of how you handle DLLs, mixing debug and release versions is not a good idea. Both versions are named wavelet2d.dll so putting them in system path is not a good idea either. Preferred way is to use Visual Studio project settings to link to the wavelet2d.lib file and put the release and debug wavelet2d dlls along with libfftw-3.3.dll in the directory containing the respective release and debug executables of your project. Import library will open the DLLs.

1.2.2 GNU-GCC (MinGW) Based Compilation

If you use GCC compiler either through MSYS or from one of the IDEs (Codeblocks, Eclipse, Qtcreator, Netbeans etc.), I have also added both debug and release versions of wavelet2d DLLs and wavelet2s static libraries. Required Files for MinGW DLLS-

- Header Files- wavelet2d.h
- Import Library- libwavelet2d.dll.a
- Wavelet DLL - libwavelet2d.dll
- FFTW DLL - libfftw3-3.dll (Import library is also included)

It is recommended that you link to the Release and Debug folders as they are. Once you link to import library and specify the folder location, wavelet import library should automatically open the two DLLs. Mixing Debug and Release versions shouldn't pose any problems in this case.

Working with MinGW Static libraries is even more straightforward. You need to include wavelet2s.h header in your program , link to static *.a library and you are set.

1.3 Working in LINUX Environment

1. This package contains two wavelet libraries- libwavelet2d.so.1.0 (shared) and libwavelet2s.a (static) compiled essentially from the same source code. Source code is available in the 'src' folder.
2. You may need to link to header files that are included with their respective libraries. They are also available in the 'src' folder.
3. You may want to install shared library in one of your existing paths to make compilation easier. You can create sym links once inside the folder(say /usr/local/lib) by using following commands
 - `ln -sf libwavelet2d.so.1.0 libwavelet2d.so`
 - `ln -sf libwavelet2d.so.1.0 libwavelet2d.so.1`
 - `ldconfig`

You will probably need superuser privileges to perform previous steps. If you don't have su privilege then you can move libwavelet2d.so.1.0 to your work folder or where your source code is, create sym links as before and then put your folder in the path during runtime.

- `ln -sf libwavelet2d.so.1.0 libwavelet2d.so`
- `ln -sf libwavelet2d.so.1.0 libwavelet2d.so.1`
- `export LD_LIBRARY_PATH=.`

libwavelet2s.a Working with static library is pretty straightforward. You will only need to include wavelet2s.h in your program and specify the library (-lwavelet2s flag) , include path (-I<path to wavelet2s.h> flag) and library path (-L<path to libwavelet2s.a> flag).

1.4 Licensing

These libraries are licensed under GNU-GPL v2.0 (or any later version). See COPYRIGHT and COPYING files for more information. These libraries statically and dynamically link to FFTW-3.2.2 static library, FFTW3-3.3 static library and FFTw-3.2.2 dynamic libraries in different implementations. More information, fftw libraries and associated files for this version are available at www.fftw.org. I have not modified fftw source codes in any way shape or form so you may want to download copies of source code and other files from the FFTW website itself if you are so inclined. However, you will find FFTW3 licensing and copyright information in the fftw3 folder.

Chapter 2

List of Functions

2.1 1D DWT/IDWT Functions

Both periodic and symmetric extension methods for Decimated DWT take exactly same input arguments. The difference is that Output vector in periodic extension has usually the same size ($\sim N$) as the input vector (N) while output vector in symmetric extension case has redundancies with length depending on size of filter used.

Periodic Extension

1. DWT: `void* dwt(vector<double> &sig, int J, string nm, vector<double> &dwt_output, vector<double> &flag, vector<int> &length)`

where,

sig :: Input Signal vector

J :: Decomposition levels

nm :: Wavelet Name (See `filtcoef` for available wavelet families)

length :: Lengths of respective approximation and detail vectors are stored in this integer vector.

dwt_output :: Output of Discrete Wavelet Transform. It stores coefficients in following format:

`[A(J) D(J) D(J-1) D(1)]`

$A(J)$ is the approximation coefficient vector at the J th level while $D(n)$ are the detail coefficient vectors at the n th level. 'length' contains the lengths of corresponding vectors. Last entry of the length vector is the length of the original signal.

flag :: Housekeeping vector. In this implementation it contains two values-


```

user@ubuntu: ~/Downloads/wavelib/wavelet2d/test
File Edit View Terminal Help

user@ubuntu:~/Downloads/wavelib/wavelet2d/test$ g++ -Wall -g dwtptest.cpp -o dwtptest -lwavelet2d -I/usr/local/include -L/usr/local/lib
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$ ./dwtptest
*****J: LEVEL DISCRETE WAVELET TRANSFORM IMPLEMENTATION*****
This program accepts signal from the user in a file format
and performs Discrete Wavelet Transform with specified
wavelet.

The Following Wavelets are in the Database:
haar, db1, db2, db3, db4, db5, db6, db7, db8, db9, db10,
db11, db12, db13, db14, db15,
bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8,
bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4,
bior5.5, bior6.8,
coif1, coif2, coif3, coif4, coif5.
Please Enter the Wavelet Name at the Prompt( No quotes) :
db2
Enter the name of signal file at the Prompt eg., signal.txt :
./statictest/signal.txt
Please Enter the Number of DWT Stages J :
2

Length of Input Vector: 256
wavelet Used: db2
Decomposition Levels: 2

Flag Values :
flag[0] = 0
flag[1] = 2

OUTPUT STATS
Length of DWT vector: 256
Length vector:
length[0] = 64
length[1] = 64
length[2] = 128
length[3] = 256

user@ubuntu:~/Downloads/wavelib/wavelet2d/test$

```

Figure 2.1: DWT stats (periodic extension) for an input signal of length 256

- flag[0] is 0 if the signal is even and it is 1 if signal is odd and if it is made even by repeating the last value one more time
- flag[1] - contains the decomposition levels.

Housekeeping vector is a double vector as it was originally meant to store more values than it currently does.

2. IDWT: void* idwt(vector<double> &dwttop,vector<double> &flag, string nm,vector<double> &idwt_output,vector<int> &length)

where,

dwttop :: is the DWT vector

flag :: Same Housekeeping function as obtained from the DWT function

nm :: Wavelet Used

idwt_output :: Output of IDWT

length :: Length vector obtained from the DWT computations

Symmetric Extension

3. DWT: void* dwt_sym(vector<double> &sig, int J, string nm, vector<double> &dwt_output,vector<double> &flag, vector<int> &length)

where,

sig :: Input Signal vector

J :: Decomposition levels

nm :: Wavelet Name (See filtcoef for available wavelet families)

length :: Lengths of respective approximation and detail vectors are stored in this integer vector.

dwt_output :: Output of Discrete Wavelet Transform. It stores coefficients in following format:

[A(J) D(J) D(J-1) D(1)]

A(J) is the approximation coefficient vector at the Jth level while D(n) are the detail coefficient vectors at the nth level. 'length' contains the lengths of corresponding vectors. Last entry of the length vector is the length of the original signal.

flag :: Housekeeping vector. In this implementation it contains two values-

- flag[0] is 0 if the signal is even and it is 1 if signal is odd and if it is made even by repeating the last value one more time
- flag[1] - contains the decomposition levels.

Housekeeping vector is a double vector as it was originally meant to store more values than it currently does.

idwt_output :: Output of the Inverse Discrete Wavelet Transform

length :: Length Vector Obtained from the DWT computations

4. IDWT: void* idwt_sym(vector<double> &dwttop,vector<double> &flag, string nm,vector<double> &idwt_output,vector<int> &length)

where,

dwttop :: is the DWT vector

flag :: Same Housekeeping function as obtained from the DWT function

nm :: Wavelet Used

idwt_output :: Output of IDWT

length :: Length vector obtained from the DWT computations

```

user@ubuntu: ~/Downloads/wavelib/wavelet2d/test
File Edit View Terminal Help
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$ rm dwtpertest
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$ g++ -Wall -g dwtsymtest.cpp -o dwtsymtest -I/usr/local/include -L/usr/local/lib
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$ ./dwtsymtest
*****J- LEVEL DISCRETE WAVELET TRANSFORM IMPLEMENTATION*****
This program accepts signal from the user in a file format
and performs Discrete Wavelet Transform with specified
wavelet.

The Following Wavelets are in the Database:
haar, db1, db2, db3, db4, db5, db6, db7, db8, db9, db10,
db11, db12, db13, db14, db15,
bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8,
bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4,
bior5.5, bior6.8,
coif1, coif2, coif3, coif4, coif5.
Please Enter the Wavelet Name at the Prompt( No quotes) :
db2
Enter the name of signal file at the Prompt eg., signal.txt :
./statictest/signal.txt
Please Enter the Number of DWT Stages J :
2

Length of Input Vector: 256
wavelet Used: db2
Decomposition Levels: 2

Flag Values :
flag[0] = 0
flag[1] = 2

OUTPUT STATS
Length of DWT vector: 261
Length vector:
length[0] = 66
length[1] = 66
length[2] = 129
length[3] = 256
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$

```

Figure 2.2: DWT stats (symmetric extension) for an input signal of length 256

2.2 1D SWT/ISWT Functions

5. SWT: `void* swt(vector<double> &sig, int J, string nm, vector<double> &swt_output, int &length)`

All the coefficients are of equal lengths and that value is stored in 'length'. 'swt_output' stores value in the same format as 'dwt' and 'dwt_sym' functions - Approximation coefficient vector at level J is stored at the beginning of the swt_output vector followed by detail coefficients vectors at levels J, J-1, ..., 1.

6. ISWT: `void* iswt(vector<double> &swtop, int J, string nm, vector<double> &iswt_output)`

swtop is the output of SWT stage, J - number of levels and nm is the wavelet as before. Output of ISWT is stored in iswt_output vector.

2.3 2D DWT/IDWT Functions

As in 1D case, both periodic and symmetric extension methods for Decimated DWT take exactly same input arguments. The difference is that Output vector in periodic extension has usually the same size (~NXN) as the input vector (NXN) while output vector in symmetric extension case has redundancies with length/breadth depending on size of filter used.

```

user@ubuntu: ~/Downloads/wavelib/wavelet2d/test
File Edit View Terminal Help
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$ ./swttest
*****J- LEVEL DISCRETE WAVELET TRANSFORM IMPLEMENTATION*****
This program accepts signal from the user in a file format
and performs Discrete Wavelet Transform with specified
wavelet.

The Following Wavelets are in the Database:
haar, db1, db2, db3, db4, db5, db6, db7, db8, db9, db10,
db11, db12, db13, db14, db15.
bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8,
bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4,
bior5.5, bior6.8,
coif1, coif2, coif3, coif4, coif5.
Please Enter the Wavelet Name at the Prompt( No quotes) :
db2
Enter the name of signal file at the Prompt eg., signal.txt :
./statictest/signal2.txt
Please Enter the Number of DWT Stages J :
2

Length of Input Vector: 247
wavelet Used: db2
Decomposition levels: 2

Length of SWT vector: 741
Length of each component vector: 247
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$

```

Figure 2.3: Two Level SWT Decomposition of a 247 length signal vector

Periodic Extension

7. 2D DWT: void* dwt_2d(vector<vector<double> > &origsig, int J, string nm, vector<double> &dwt_output, vector<double> &flag, vector<int> &length)

origsig :: Input Image/Matrix

J :: Number of Decomposition Levels

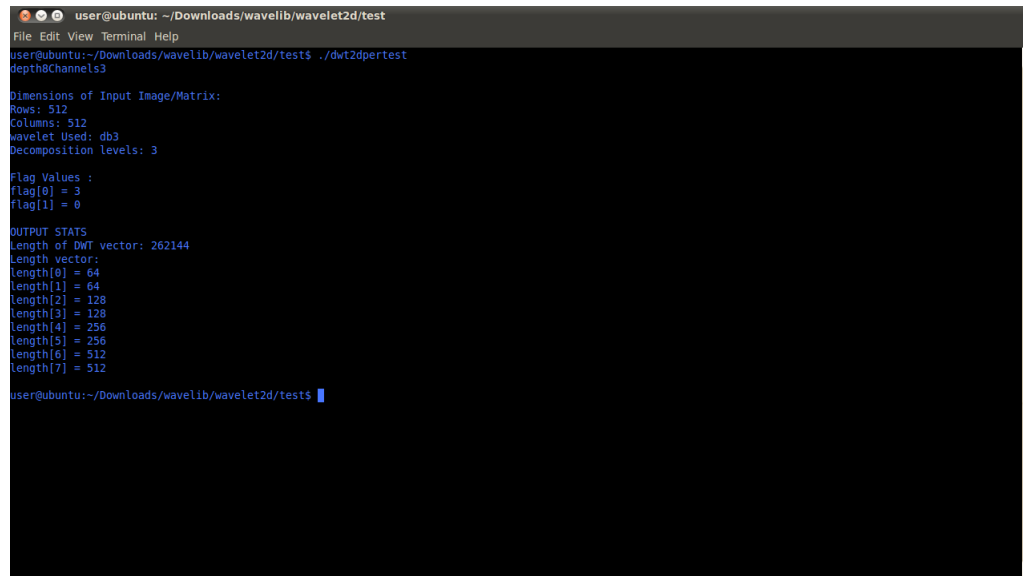
nm :: Wavelet Name

flag :: Stores values for IDWT function. Only flag0 value is important as it contains decomposition levels.

dwt_output :: 1D vector that stores the output in the following format A(J) D_h(J) D_v(J) D_d(J) D_h(1) D_v(1) D_d(1)

where A(J) is the approximation coefficient vector at the Jth level while D(n) are the three detail coefficient vectors(horizontal, vertical and detail) at the nth level. It is important to remember that approximation and detail coefficients are actually two dimensional so we need a length vector that stores rows and columns values of each coefficient element. The length vector is given by length.

For example, the first element of output vector is the approximation matrix stored as a vector and the first two elements of length vectors are row and column values of the approximation matrix. In other words, a 300 element



```

user@ubuntu: ~/Downloads/wavelib/wavelet2d/test
File Edit View Terminal Help
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$ ./dwt2dptest
depth8Channels3
Dimensions of Input Image/Matrix:
Rows: 512
Columns: 512
Wavelet Used: db3
Decomposition levels: 3
Flag Values :
Flag[0] = 3
Flag[1] = 0
OUTPUT STATS
Length of DWT vector: 262144
Length vector:
length[0] = 64
length[1] = 64
length[2] = 128
length[3] = 128
length[4] = 256
length[5] = 256
length[6] = 512
length[7] = 512
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$

```

Figure 2.4: 2D DWT computation using periodic extension

approximation matrix (15 rows X 20 columns) can be extracted from the 300 element approximation vector.

8. 2D IDWT: void* idwt_2d(vector<double> &dwttop,vector<double> &flag, string nm,vector<vector<double>> &idwt_output, vector<int> &length)

where,

dwttop :: is the DWT vector

flag :: Same Housekeeping function as obtained from the DWT function

nm :: Wavelet Used

idwt_output :: Output of IDWT which should be defined to have the same number of rows and columns as the input image/matrix

length :: Length vector obtained from the DWT computations

Symmetric Extension

9. 2D DWT: void* dwt_2d_sym(vector<vector<double>> &origsig, int J, string nm, vector<double> &dwt_output,vector<double> &flag, vector<int> &length)

origsig :: Input Image/Matrix

J :: Number of Decomposition Levels

```

user@ubuntu: ~/Downloads/wavelib/wavelet2d/test
File Edit View Terminal Help
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$ g++ -Wall -g dwtdsymtest.cpp -o dwtdsymtest -lwavelet2d -I/usr/local/include -L/usr/local/lib -lcxcore -I/usr/include/opencv -L/usr/lib
user@ubuntu:~/Downloads/wavelib/wavelet2d/test$ ./dwtdsymtest
depth8Channels3

Dimensions of Input Image/Matrix:
Rows: 512
Columns: 512
Wavelet Used: db3
Decomposition Levels: 3

Flag Values :
flag[0] = 3

OUTPUT STATS
Length of DWT vector: 269671
Length vector:
length[0] = 68
length[1] = 68
length[2] = 131
length[3] = 131
length[4] = 258
length[5] = 258
length[6] = 512
length[7] = 512

user@ubuntu:~/Downloads/wavelib/wavelet2d/test$

```

Figure 2.5: 2D DWT computation using symmetric extension

nm :: Wavelet Name

flag :: Stores values for IDWT function. Only flag0 value is important as it contains decomposition levels.

dwt_output :: 1D vector that stores the output in the following format $A(J)$ $D_h(J)$ $D_v(J)$ $D_d(J)$ $D_h(1)$ $D_v(1)$ $D_d(1)$

where $A(J)$ is the approximation coefficient vector at the J th level while $D(n)$ are the three detail coefficient vectors (horizontal, vertical and detail) at the n th level. It is important to remember that approximation and detail coefficients are actually two dimensional so we need a length vector that stores rows and columns values of each coefficient element. The length vector is given by length.

For example, the first element of output vector is the approximation matrix stored as a vector and the first two elements of length vectors are row and column values of the approximation matrix. In other words, a 300 element approximation matrix (15 rows X 20 columns) can be extracted from the 300 element approximation vector.

10. 2D IDWT: `void* idwt_2d_sym(vector<double> &dwt, vector<double> &flag, string nm, vector<vector<double>> &idwt_output, vector<int> &length)`

where,

dwtop :: is the DWT vector

flag :: Same Housekeeping function as obtained from the DWT function

nm :: Wavelet Used

idwt_output :: Output of IDWT which should be defined to have the same number of rows and columns as the input image/matrix

length :: Length vector obtained from the DWT computations

2.4 2D SWT Functions

11. 2D SWT: void* swt_2d(vector<vector<double> > &sig,int J, string nm, vector<double> &swt_output)

swt_output is a 1D vector which is arranged the same way as DWT output vector in the Decimated 2D cases above except that in this case all coefficients are of same size. This is a highly redundant transform as a three level decomposition of a 512X512 image results in 10 512X512 images - one approximation image and 9 detail images (three at each level).

2.5 Convolution

12. Convolution FFT_ESTIMATE (Recommended): double convfft(vector<double> &a, vector<double> &b, vector<double> &c)

Convfft function is pretty straightforward. a and b are input vectors and c is the convolution output. convfft uses FFT so it gives better results in most cases than the regular fft which is implemented by convol function.

13. Convolution Direct (Use it for only smaller vectors): double convol(vector<double> &a, vector<double> &b, vector<double> &c)

14. Convolution FFT_MEASURE (Recommended if you are going to perform convolutions of same length hundreds or thousands of time in one program): double convfftm(vector<double> &a, vector<double> &b, vector<double> &c)

convfftm is performed using MEASUREing capabilities of FFTW3 library so it is not recommended if you are going to convolve two vectors only once. This has some overhead but gives good results if multiple instances of same convolution are performed repeatedly.

2.6 Wavelet Filters

15. Filters: `int filtcoef(string , vector<double> &, vector<double> &, vector<double> &, vector<double> &)`

nm: Wavelet name.

lpd: Low Pass Decomposition Filter Coefficients.

hpd: High Pass Decomposition Filter Coefficients.

lpr: Low Pass Reconstruction Filter Coefficients.

hpr: High Pass Reconstruction Filter Coefficients.

All filters are `vector<double>` objects and can be obtained by specifying the wavelet name. Currently, following Wavelets are available:

Daubechies : `db1,db2,...,db15`

Biorthogonal: `bior1.1 ,bior1.3 ,bior1.5 ,bior2.2 ,bior2.4 ,bior2.6 ,bior2.8 ,bior3.1 ,bior3.3 ,bior3.5 ,bior3.7 ,bior3.9 ,bior4.4 ,bior5.5 ,bior6.8`

Coiflets: `coif1,coif2,coif3,coif4,coif5`

Symmlets: `sym1,sym2,....., sym10`

2.7 1D Vector Manipulation

16. Downsampling: `void downsamp(vector<double> &sig, int M, vector<double> &sig_d)`

sig :: Signal to be Downsampled

M :: Downsampling factor

sig_d :: Downsampled signal

17. Upsampling: `void upsamp(vector<double> &sig, int M, vector<double> &sig_u)`

sig :: Signal to be Upsampled

M :: Upsampling factor

sig_u :: Upsampled signal

18. Periodic Extension: `void* per_ext(vector<double> &sig, int a)`

per_ext periodically extends the signal sig by value a in either direction.

19. Symmetric Extension: `void* symm_ext(vector<double> &sig, int a)`

symm_ext symmetrically extends the signal sig by value a in either direction. This function needs refinement as it doesn't gives good results for smaller vectors.

2.8 2D Vector Manipulation

20. 2D Downsampling: `void* downsamp2(vector<vector<double> > & vec1, vector<vector<double> > & vec2, int rows_dn, int cols_dn)`

vec1 is the input, `rows_dn` and `cols_dn` are row and column downsampling factors. **vec2** is the downsampled matrix.

21. 2D Upsampling: `void* upsamp2(vector<vector<double> > & vec1, vector<vector<double> > & vec2, int rows_up, int cols_up)`

vec1 is the input, `rows_up` and `cols_up` are row and column upsampling factors. **vec2** is the upsampled matrix.

22. 2D Periodic Extension: `void* per_ext2d(vector<vector<double> > & signal, vector<vector<double> > & temp2, int a)`

signal is extended by `a` in all directions and the result is returned in 2D vector **temp2**

Chapter 3

Sample Codes

Notes

1. dwt and dwt_sym functions are interchangeable in the following code as they take the same arguments.
2. Please make sure that right header file is included. Header file for dynamic libraries is wavelet2d.h and for static libraries, wavelet2s.h is used.
3. I'm using OPENCV to handle images as i already use OPENCV for other image processing work. You may want to use some simpler image libraries as OPENCV is a full image processing suite and is very bulky or you can just use 2D matrices/build your own image classes. Regardless, DWT/IDWT operations are more important than the choice of libraries.
4. All updated example codes are available at

<http://code.google.com/p/wavelet1d/source/browse/trunk/wavelet2d/demo/>

3.1 1D DWT/IDWT Example Code

```
//=====
// Name : wavedemo1.cpp
// Author : Rafat Hussain
// Version :
// Copyright :
// Description : 1D DWT Demo
//=====
#include <iostream>
#include <fstream>
#include "wavelet2d.h"
#include <vector>
#include <string>
```

```

#include <cmath>
using namespace std;
int main() {
    cout << "*****J- LEVEL DISCRETE WAVELET TRANSFORM IMPLEMEN-
TATION*****" << endl;
    cout << "This program accepts signal from the user in a file format " <<
endl;
    cout << "and performs Discrete Wavelet Transform with specified " << endl;
    cout << "wavelet. " << endl;
    cout << " " << endl;
    cout << " The Following Wavelets are in the Database: " << endl;
    cout << " haar, db1, db2, db3, db4, db5, db6, db7, db8, db9, db10, " << endl;
    cout << " db11, db12, db13, db14, db15. " << endl;
    cout << " bior1.1, bio1.3, bior1.5, bior2.2, bior2.4,bior2.6,bior2.8, " << endl;
    cout << " bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4," << endl;
    cout << " bior5.5, bior6.8." << endl;
    cout << " coif1, coif2, coif3, coif4, coif5." << endl;
    cout << "Please Enter the Wavelet Name at the Prompt( No quotes) : " <<
endl;
    string nm; // nm will store the name of Wavelet Family
    cin >> nm;
    cout << "Enter the name of signal file at the Prompt eg., signal.txt : " << endl;
    char inp[50];
    cin >> inp;
    vector<double> sig;
    ifstream sig_inp(inp);
    if ( !sig_inp.good()){
        cout << "The File doesn't exist"<< endl;
    }
    while (sig_inp) {
        double temp;
        sig_inp >> temp;
        sig.push_back(temp);
    }
    sig.pop_back();
    vector<double> original;
    original = sig;
    cout << "Please Enter the Number of DWT Stages J : " << endl;
    int J;
    cin >> J ;
    vector<double> dwt_output, flag;
    // perform J-Level DWT
    vector<int> length;
    dwt_sym(sig, J, nm, dwt_output,flag,length);
    ofstream dwtout("dwtout.txt");
    for (unsigned int i = 0; i < dwt_output.size(); i++){

```

```

user@ubuntu: ~/Downloads/wavelib/wavelet2d/dynamictest
File Edit View Terminal Help
user@ubuntu:~/Downloads/wavelib/wavelet2d/dynamictest$ ./wavedemo
*****J- LEVEL DISCRETE WAVELET TRANSFORM IMPLEMENTATION*****
This program accepts signal from the user in a file format
and performs Discrete Wavelet Transform with specified
wavelet.

The Following Wavelets are in the Database:
haar, db1, db2, db3, db4, db5, db6, db7, db8, db9, db10,
db11, db12, db13, db14, db15.
bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8,
bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4,
bior5.5, bior6.8,
coif1, coif2, coif3, coif4, coif5.
Please Enter the Wavelet Name at the Prompt( No quotes) :
db2
Enter the name of signal file at the Prompt eg., signal.txt :
./statictest/signal.txt
Please Enter the Number of DWT Stages J :
2
Recon signal size256
user@ubuntu:~/Downloads/wavelib/wavelet2d/dynamictest$

```

Figure 3.1: 1D DWT Console

```

dwtout << dwt_output[i] << endl;
}
//Perform J-Level IDWT
vector<double> output;
idwt_sym(dwt_output, flag,nm,output,length);
ofstream sig1("recon.txt");
ofstream diff("diff.txt");
cout <<" Recon signal size" << output.size() << endl;
for (unsigned int i = 0; i < output.size(); i++){
sig1 << output[i] << endl;
diff << output[i] - original[i] << endl;
}
//gnudwtplot(J);
return 0;
}

```

This sample program asks user for wavelet name, signal file and number of DWT stages. It then computes DWT of the input signal and then the IDWT from the wavelet coefficients so obtained.

As can be seen, outputs are in .txt format but they can be modified and plotted with any software of your choice.

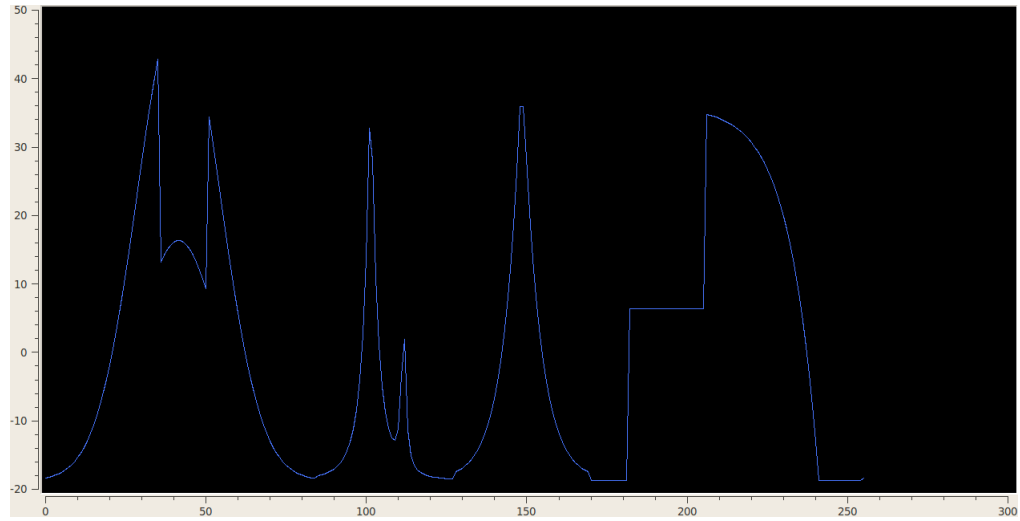


Figure 3.2: Length 256 Piecewise Regular signal from WaveLab

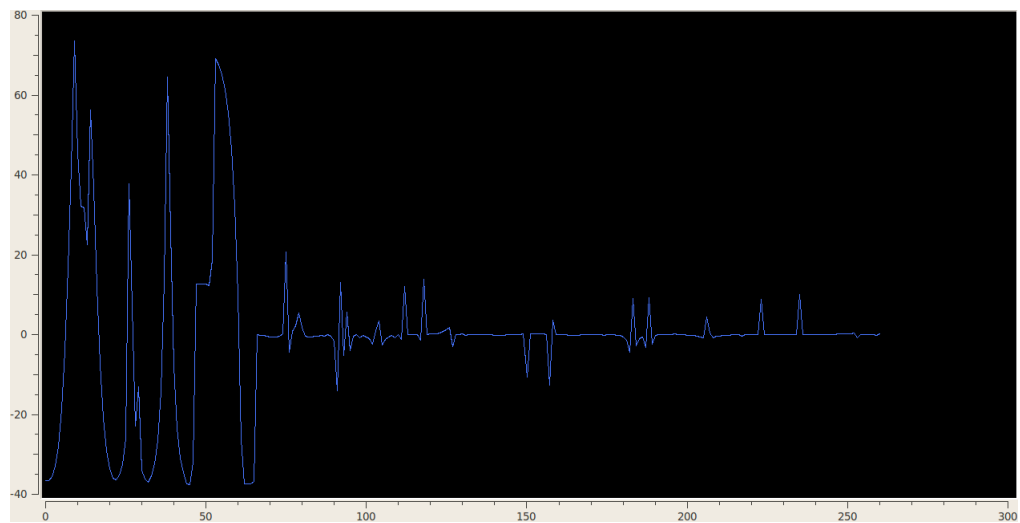


Figure 3.3: 2 Level DWT Decomposition Coefficients obtained using db2 wavelet

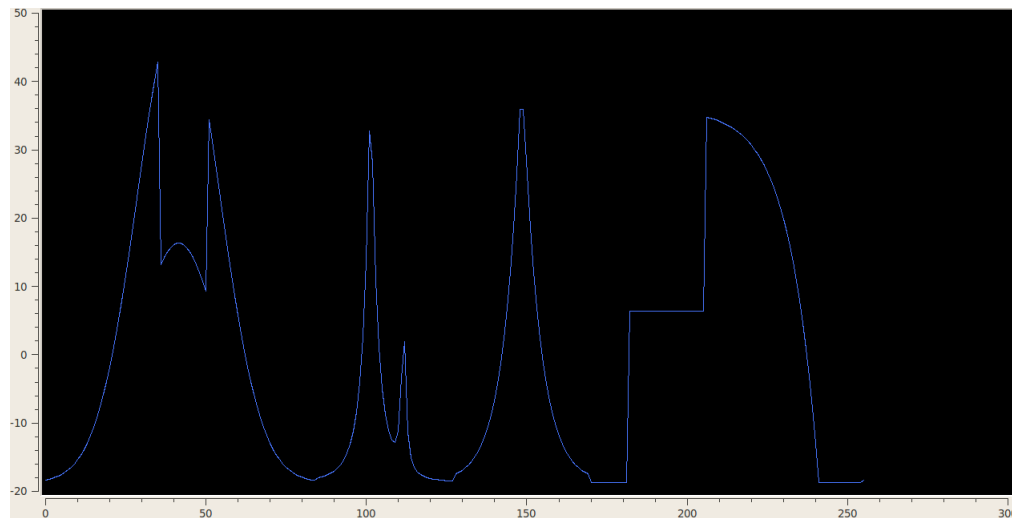


Figure 3.4: Reconstructed Signal After IDWT computations

3.2 1D Linear vs Nonlinear Approximation

```
//=====
// Name : wavedemo2.cpp
// Author : Rafat Hussain
// Version :
// Copyright :
// Description : Wavelet Demo comparing linear and non-linear approxi-
mation properties
// : of a given wavelet. Implemented using dwt_sym and idwt_sym.
//=====
#include <iostream>
#include <fstream>
#include "wavelet2s.h"
#include <vector>
#include <string>
#include <cmath>
#include <algorithm>
using namespace std;
void findthresh(vector<double> vector1, int N, double& t){
    sort(vector1.begin(), vector1.end(), greater<double>());
    t = vector1.at(N-1);
}
int main() {
    cout << "*****J- LEVEL DISCRETE WAVELET TRANSFORM IMPLEMEN-
TATION*****" << endl;
```

```

    cout << "This program accepts signal from the user in a file format " <<
endl;
    cout << "and performs Discrete Wavelet Transform with specified " << endl;
    cout << "wavelet. " << endl;
    cout << " " << endl;
    cout << " The Following Wavelets are in the Database: " << endl;
    cout << " haar, db1, db2, db3, db4, db5, db6, db7, db8, db9, db10, " << endl;
    cout << " db11, db12, db13, db14, db15. " << endl;
    cout << " bior1.1, bio1.3, bior1.5, bior2.2, bior2.4,bior2.6,bior2.8, " << endl;
    cout << " bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4," << endl;
    cout << " bior5.5, bior6.8." << endl;
    cout << " coif1, coif2, coif3, coif4, coif5." << endl;
    cout << "Please Enter the Wavelet Name at the Prompt( No quotes) : " <<
endl;
    string nm; // nm will store the name of Wavelet Family
    cin >> nm;
    cout << "Enter the name of signal file at the Prompt eg., signal.txt : " << endl;
    char inp[50];
    cin >> inp;
    vector<double> sig;
    ifstream sig_inp(inp);
    if ( !sig_inp.good()){
        cout << "The File doesn't exist" << endl;
        exit(1);
    }
    while (sig_inp) {
        double temp;
        sig_inp >> temp;
        sig.push_back(temp);
    }
    sig.pop_back();
    vector<double> original;
    original = sig;
    cout << "Please Enter the Number of DWT Stages J : " << endl;
    int J;
    cin >> J ;
    vector<double> dwt_output, flag;
    vector<int> length1;
    // perform J-Level DWT
    dwt(sig, J, nm, dwt_output, flag, length1);
    // Performing Linear Approximation by using only first 100 coefficients
    // Coefficients in dwt_output are stored as following
    // dwt_output =[ Appx(J-1) Detail(J-1) Detail(J-2) .... Detail(0)]
    int n_coef = 100; // Number of significant coefficients
    int n_non_sig= dwt_output.size() - n_coef; // Number of Coefficients that
will

```

```

// be set to zero
dwt_output.erase(dwt_output.end()- n_non_sig,dwt_output.end());
// Deleting last n_non_sig coefficients and replacing them with zeros
dwt_output.insert(dwt_output.end(),n_non_sig,0);
ofstream linearsig("linsig.txt");
for (unsigned int i = 0; i < dwt_output.size(); i++) {
linearsig << dwt_output[i] << endl;
}
// Finding IDWT with approximated coefficients
vector<double> output;
idwt(dwt_output, flag,nm,output, length1);
unsigned int count = output.size();
ofstream gnulinappx("gnulinappx.dat");
for (unsigned int i = 0; i < count; i++) {
gnulinappx << output[i] << endl;
}
gnulinappx.close();
// Performing Non Linear Approximation by using only most
// significant coefficients
vector<double> dwt_output2, flag2;
vector<int> length2;
// perform J-Level DWT
dwt(sig, J, nm, dwt_output2,flag2,length2);
double thresh = 0.0;
vector<double> temp_dwtoutput;
for (unsigned int i =0; i < dwt_output2.size();i++){
double temp = abs(dwt_output2[i]);
temp_dwtoutput.push_back(temp);
}
/*
for (unsigned int i =0; i < temp_dwtoutput.size(); i++){
cout << temp_dwtoutput[i] << endl;
}
*/
findthresh(temp_dwtoutput,n_coef, thresh);
for (unsigned int i = 0; i < dwt_output2.size();i++){
double temp = abs(dwt_output2[i]);
if (temp < thresh){
dwt_output2.at(i) = 0.0;
}
}
/*
for (unsigned int i =0; i < dwt_output2.size(); i++){
cout << dwt_output2[i] << endl;
}
*/

```



```

user@ubuntu: ~/Downloads/wavelib/wavelet2d/statictest
File Edit View Terminal Help
user@ubuntu:~/Downloads/wavelib/wavelet2d/statictest$ g++ -Wall -g wavedemo2.cpp -o wavedemo2 -lwavelet2s -I. -L.
user@ubuntu:~/Downloads/wavelib/wavelet2d/statictest$ ./wavedemo2
*****J- LEVEL DISCRETE WAVELET TRANSFORM IMPLEMENTATION*****
This program accepts signal from the user in a file format
and performs Discrete Wavelet Transform with specified
wavelet.

The Following Wavelets are in the Database:
haar, db1, db2, db3, db4, db5, db6, db7, db8, db9, db10,
db11, db12, db13, db14, db15.
bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8,
bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4,
bior5.5, bior6.8,
coif1, coif2, coif3, coif4, coif5.
Please Enter the Wavelet Name at the Prompt( No quotes) :
db3
Enter the name of signal file at the Prompt eg., signal.txt :
signal.txt
Please Enter the Number of DWT Stages J :
5
user@ubuntu:~/Downloads/wavelib/wavelet2d/statictest$

```

Figure 3.5: Console showing db3 5 level computation

```

ofstream nonlinsig("nonlinsig.txt");
for (unsigned int i = 0; i < dwt_output2.size(); i++) {
    nonlinsig << dwt_output2[i] << endl;
}
// Finding IDWT with approximated coefficients
vector<double> output2;
idwt(dwt_output2, flag2, nm, output2, length2);
unsigned int count2 = output2.size();
cout << count2 << endl;
ofstream gnunlappx("gnunlappx.dat");
for (unsigned int i = 0; i < count2; i++) {
    gnunlappx << output2[i] << endl;
}
gnunlappx.close();
return 0;
}

```

This program computes DWT of the 256-point signal and then uses only 100 coefficients to reconstruct the signal. In the first case, first 100 coefficients are used to compute linear approximation. In the second case, 100 largest coefficients are used by calculating a threshold that separates largest 100 coefficients from the others. The wavelet used is db3 and the signal is decomposed and reconstructed over 5 stages of DWT/IDWT. The results are plotted below.

In Linear Approximation case, only first 100 coefficients are retained. Others are set to zero.

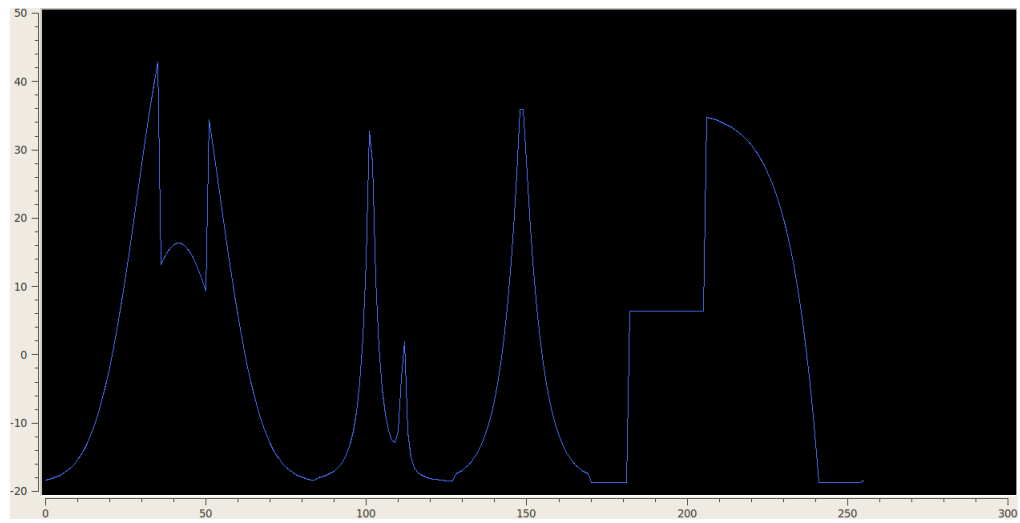


Figure 3.6: Length 256 Piecewise Regular signal from WaveLab

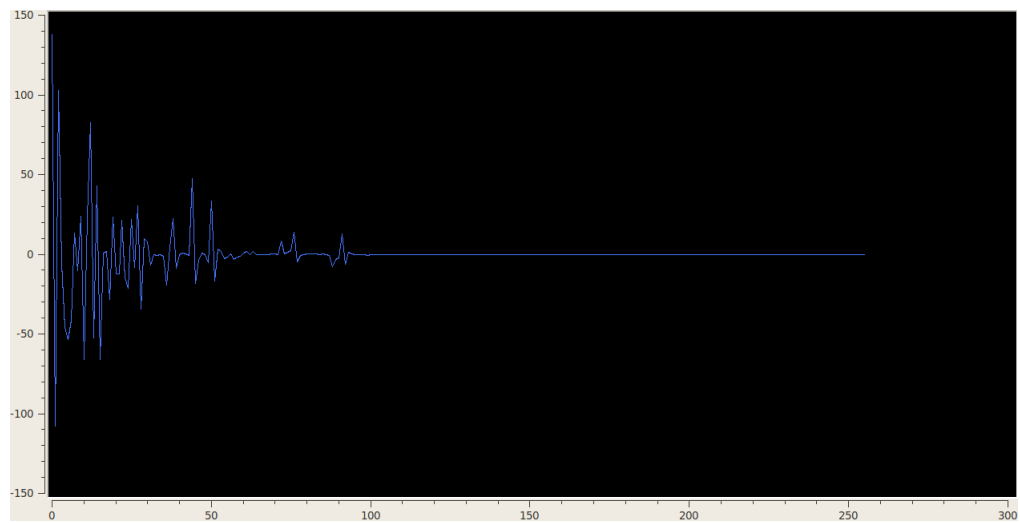


Figure 3.7: Retained Coefficients for Linear Approximation case

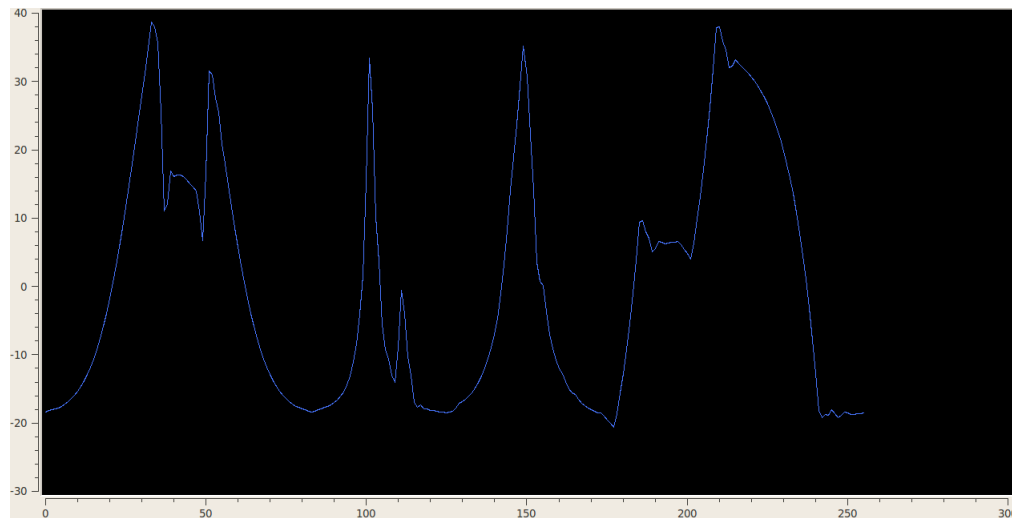


Figure 3.8: Reconstructed Signal using Linear Approximation

In Non-Linear case, 100 largest coefficients are retained.

3.3 1D Stationary Wavelet Transform

```
//=====
// Name : swtdemo.cpp
// Author : Rafat Hussain
// Version :
// Copyright :
// Description : 1D Stationary Wavelet Transform Demo
//=====
#include <iostream>
#include <fstream>
#include "wavelet2d.h"
#include <vector>
#include <string>
#include <cmath>
using namespace std;
int main() {
    cout << "*****J- LEVEL DISCRETE WAVELET TRANSFORM IMPLEMEN-
TATION*****" << endl;
    cout << "This program accepts signal from the user in a file format " <<
endl;
    cout << "and performs Discrete Wavelet Transform with specified " << endl;
    cout << "wavelet. " << endl;
```

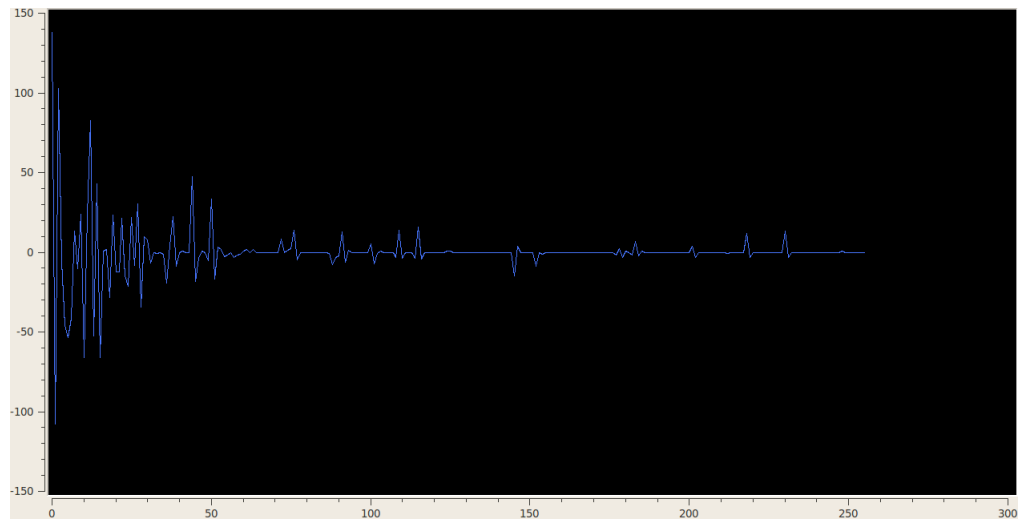


Figure 3.9: Retained Coefficients(Non-Linear Approximation case)

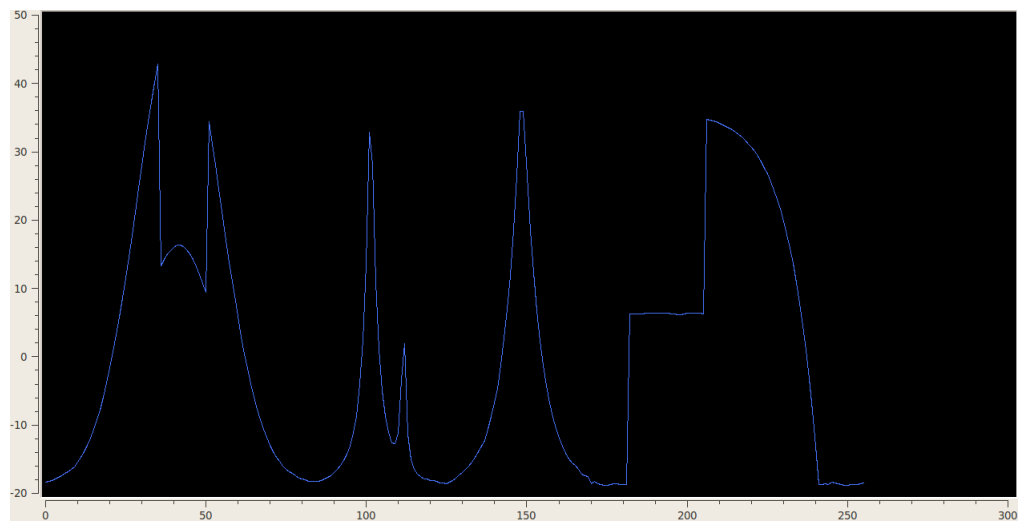
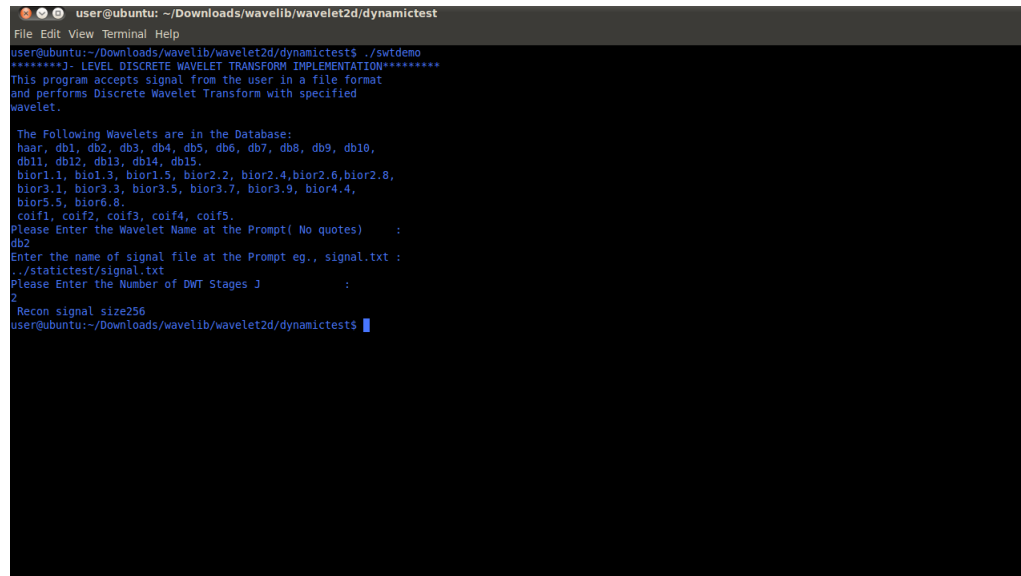


Figure 3.10: Reconstructed Signal(Non-Linear Approximation Case)

```

cout << " " << endl;
cout << " The Following Wavelets are in the Database: " << endl;
cout << " haar, db1, db2, db3, db4, db5, db6, db7, db8, db9, db10, " << endl;
cout << " db11, db12, db13, db14, db15. " << endl;
cout << " bior1.1, bio1.3, bior1.5, bior2.2, bior2.4,bior2.6,bior2.8, " << endl;
cout << " bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4," << endl;
cout << " bior5.5, bior6.8." << endl;
cout << " coif1, coif2, coif3, coif4, coif5." << endl;
cout << "Please Enter the Wavelet Name at the Prompt( No quotes) : " <<
endl;
string nm; // nm will store the name of Wavelet Family
cin >> nm;
cout << "Enter the name of signal file at the Prompt eg., signal.txt : " << endl;
char inp[50];
cin >> inp;
vector<double> sig;
ifstream sig_inp(inp);
if ( !sig_inp.good()){
cout << "The File doesn't exist"<< endl;
}
while (sig_inp) {
double temp;
sig_inp >> temp;
sig.push_back(temp);
}
sig.pop_back();
vector<double> original;
original = sig; // Make a copy of the signal if you want to use original
signal
// later on. The other option is to use IDWT output as the SWt/ISWT
system is
// Perfect Reconstruction system.
cout << "Please Enter the Number of DWT Stages J : " << endl;
int J;
cin >> J;
vector<double> swt_output;
// perform J-Level DWT
int length; // All coefficients are of same length. Variable "length" returns
length
// of coefficients. It is not required for ISWT computations.
// If signal is not of dyadic length it is zeropadded as the algorithm is
designed to
// work with signals of 2^M length. These extra zeros can be removed
after reconstruction
swt(sig, J, nm, swt_output, length);
ofstream swtcoeff("swtcoeff.txt");

```



```

user@ubuntu: ~/Downloads/wavelib/wavelet2d/dynamictest
File Edit View Terminal Help
user@ubuntu:~/Downloads/wavelib/wavelet2d/dynamictest$ ./swtdemo
*****J- LEVEL DISCRETE WAVELET TRANSFORM IMPLEMENTATION*****
This program accepts signal from the user in a file format
and performs Discrete Wavelet Transform with specified
wavelet.

The Following Wavelets are in the Database:
haar, db1, db2, db3, db4, db5, db6, db7, db8, db9, db10,
db11, db12, db13, db14, db15,
bior1.1, bior1.3, bior1.5, bior2.2, bior2.4, bior2.6, bior2.8,
bior3.1, bior3.3, bior3.5, bior3.7, bior3.9, bior4.4,
bior5.5, bior6.8,
coif1, coif2, coif3, coif4, coif5.
Please Enter the Wavelet Name at the Prompt( No quotes) :
db2
Enter the name of signal file at the Prompt eg., signal.txt :
./statictest/signal.txt
Please Enter the Number of DWT Stages J :
2
Recon signal size256
user@ubuntu:~/Downloads/wavelib/wavelet2d/dynamictest$

```

Figure 3.11: Console Running swt demo

```

for (unsigned int i=0; i < swt_output.size(); i++) {
    swtcoeff << swt_output[i] << endl;
}
vector<double> iswt_output;
iswt(swt_output,J, nm,iswt_output);
ofstream sig1("recon.txt");
ofstream diff("diff.txt");
cout << " Recon signal size" << iswt_output.size() << endl;
for (unsigned int i = 0; i < iswt_output.size(); i++){
    sig1 << iswt_output[i] << endl;
    diff << iswt_output[i] - original[i] << endl;
}
return 0;
}

```

3.4 2D DWT/IDWT Demo Code

```

//=====
// Name : imagedemo_sym.cpp
// Author : Rafat Hussain
// Version :
// Copyright :

```

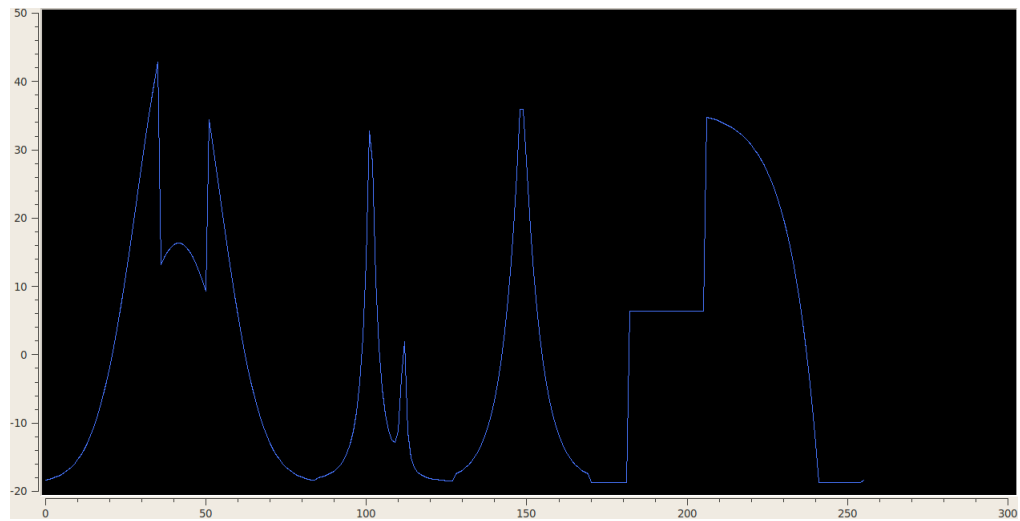


Figure 3.12: Input Signal(Length-256)

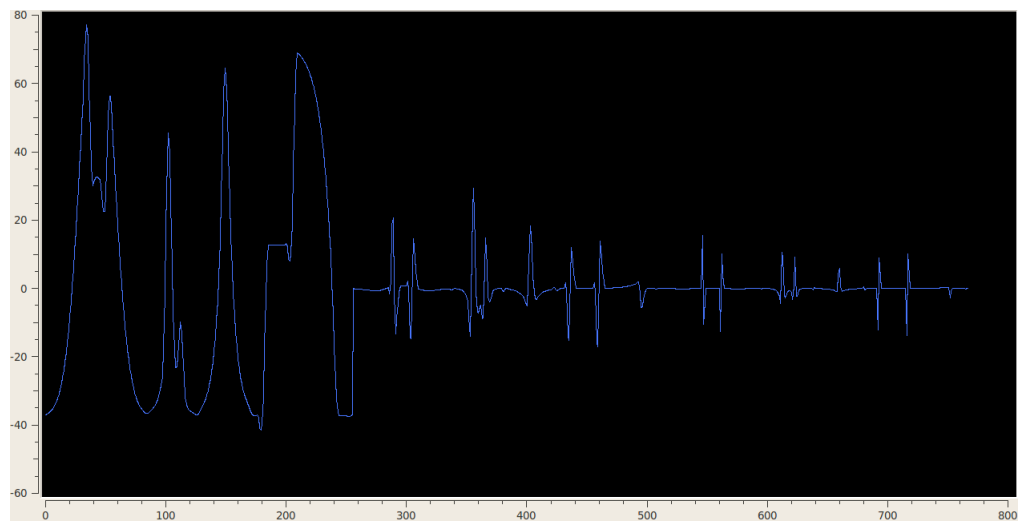


Figure 3.13: SWT Coefficients for two level of Decomposition(Length 3 X 256)

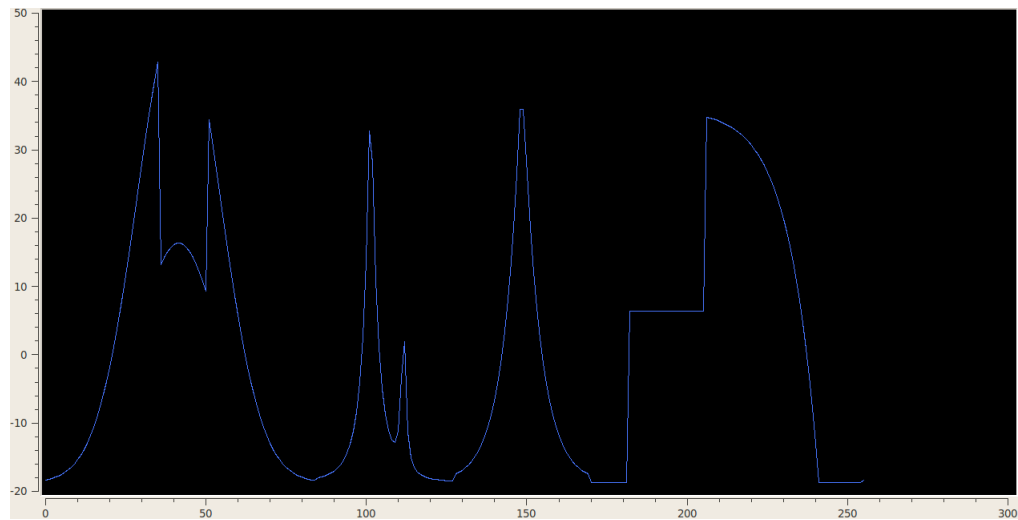


Figure 3.14: SWT Reconstructed Signal

```
// Description : DWT of arbitrary size image using symmetric or periodic
extension
//=====
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <complex>
#include <cmath>
#include <algorithm>
#include "wavelet2d.h"
#include "cv.h"
#include "highgui.h"
#include "cxcore.h"
using namespace std;
using namespace cv;
void* maxval(vector<vector<double> > &arr, double &max){
    max = 0;
    for (unsigned int i=0; i < arr.size(); i++) {
        for (unsigned int j=0; j < arr[0].size(); j++) {
            if (max <= arr[i][j]){
                max = arr[i][j];
            }
        }
    }
    return 0;
}
```



```

    }
    void* maxval1(vector<double> &arr, double &max){
        max = 0;
        for (unsigned int i =0; i < arr.size(); i++) {
            if (max <= arr[i]){
                max = arr[i];
            }
        }
        return 0;
    }
    int main() {
        IplImage* img = cvLoadImage("snow.jpg");
        if (!img){
            cout << "Can't read Image. Try Different Format." << endl;
            exit(1);
        }
        int height, width;
        height = img->height;
        width = img->width;
        int nc = img->nChannels;
        // uchar* ptr2 =(uchar*) img->imageData;
        int pix_depth = img->depth;
        CvSize size;
        size.width =width;
        size.height=height;
        cout << "depth" << pix_depth << "Channels" << nc << endl;
        cvNamedWindow("Original Image", CV_WINDOW_AUTOSIZE);
        cvShowImage("Original Image", img);
        cvWaitKey();
        cvDestroyWindow("Original Image");
        cvSaveImage("orig.bmp",img);
        int rows =(int) height;
        int cols =(int) width;
        Mat matimg(img);
        vector<vector<double> > vec1(rows, vector<double>(cols));
        int k =1;
        for (int i=0; i < rows; i++) {
            for (int j =0; j < cols; j++){
                unsigned char temp;
                temp = ((uchar*) matimg.data + i * matimg.step)[j * matimg.elemSize() + k
];
                vec1[i][j] = (double) temp;
            }
        }
        string nm = "db3";
        vector<double> l1,h1,l2,h2;

```

```

    filtcoef(nm,l1,h1,l2,h2);
    // unsigned int lf=l1.size();
    // int rows_n =(int) (rows+ J*(lf-1));
    // int cols_n =(int) (cols + J * ( lf -1));
    // Finding 2D DWT Transform of the image using symetric extension al-
    gorithm
    // Extension is set to 3 (eg., int e = 3)
    vector<int> length;
    vector<double> output,flag;
    int J =3;
    dwtdisp_sym(vec1,J,nm,output,flag,length);
    double max;
    vector<int> length2;
    // This algorithm computes DWT of image of any given size. Together
    with convolution and
    // subsampling operations it is clear that subsampled images are of differ-
    ent length than
    // dyadic length images. In order to compute the "effective" size of DWT
    we do additional
    // calculations.
    dwtdisp_dim_sym(length,length2,J);
    // length2 is gives the integer vector that contains the size of subimages
    that will
    // combine to form the displayed output image. The last two entries of
    length2 gives the
    // size of DWT ( rows_n by cols_n)
    int siz = length2.size();
    int rows_n=length2[siz-2];
    int cols_n = length2[siz-1];
    vector<vector< double> > dwtdisp(rows_n, vector<double>(cols_n));
    dispDWT(output,dwtdisp, length ,length2, J);
    // dispDWT returns the 2D object dwtdisp which will be displayed using
    OPENCV's image
    // handling functions
    vector<vector<double> > dwtdisp= dwtdisp;
    maxval(dwtdisp,max); // max value is needed to take care of overflow
    which happens because
    // of convolution operations performed on unsigned 8 bit images
    //Displaying Scaled Image
    // Creating Image in OPENCV
    IplImage *cvImg; // image used for output
    CvSize imgSize; // size of output image
    imgSize.width = cols_n;
    imgSize.height = rows_n;
    cvImg = cvCreateImage( imgSize, 8, 1 );

```

```

// dwt_hold is created to hold the dwt output as further operations need
to be
// carried out on dwt_output in order to display scaled images.
vector<vector<double>> dwt_hold(rows_n, vector<double>( cols_n));
dwt_hold = dwt_output;
// Setting coefficients of created image to the scaled DWT output values
for (int i = 0; i < imgSize.height; i++ ) {
for (int j = 0; j < imgSize.width; j++ ){
if ( dwt_output[i][j] <= 0.0){
dwt_output[i][j] = 0.0;
}
if ( i <= (length2[0]) && j <= (length2[1]) ) {
((uchar*)(cvImg->imageData + cvImg->widthStep*i))[j] =
(char) ( (dwt_output[i][j] / max) * 255.0);
} else {
((uchar*)(cvImg->imageData + cvImg->widthStep*i))[j] =
(char) (dwt_output[i][j]) ;
}
}
}
cvNamedWindow( "DWT Image", 1 ); // creation of a visualisation win-
dow
cvShowImage( "DWT Image", cvImg ); // image visualisation
cvWaitKey();
cvDestroyWindow("DWT Image");
cvSaveImage("dwt.bmp",cvImg);
// Finding IDWT
vector<vector<double>> idwt_output(rows, vector<double>(cols));
idwt_2d_sym(output,flag, nm, idwt_output,length);
//Displaying Reconstructed Image
IplImage *dvImg;
CvSize dvSize; // size of output image
dvSize.width = idwt_output[0].size();
dvSize.height = idwt_output.size();
cout << idwt_output.size() << idwt_output[0].size() << endl;
dvImg = cvCreateImage( dvSize, 8, 1 );
for (int i = 0; i < dvSize.height; i++ )
for (int j = 0; j < dvSize.width; j++ )
((uchar*)(dvImg->imageData + dvImg->widthStep*i))[j] =
(char) (idwt_output[i][j]) ;
cvNamedWindow( "Reconstructed Image", 1 ); // creation of a visualisa-
tion window
cvShowImage( "Reconstructed Image", dvImg ); // image visualisation
cvWaitKey();
cvDestroyWindow("Reconstructed Image");
cvSaveImage("recon.bmp",dvImg);

```



Figure 3.15: Input 250X189 Image



Figure 3.16: 3-Level Decomposition using db3 wavelet

```
return 0;
}
```

Input Image is a low resolution 250X189 grayscale snow.jpg image.

3.5 Image Approximation Demo

```
//=====
// Name : imagedemo2.cpp
// Author : Rafat Hussain
// Version :
// Copyright :
// Description : Image Approximation using symmetric extension 2D DWT
//=====
// IMPORTANT - Algorithm used to display Image is imprecise because
of int 8 overflow issues
// and it shouldn't be used to judge the performance of the DWT. The DWT
and IDWT outputs
// should be used for performance measurements. I have used maximum
value rescaling to
```



Figure 3.17: Reconstructed Image

// solve overflow issues and , obviously, it is going to result in suboptimal performance but

// it is good enough for demonstration purposes.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <complex>
#include <cmath>
#include <algorithm>
#include "wavelet2s.h"
#include "cv.h"
#include "highgui.h"
#include "cxcore.h"
using namespace std;
using namespace cv;
void findthresh(vector<double> &vector1, int N, double& t){
    sort(vector1.begin(), vector1.end(), greater<double>());
    t = vector1.at(N-1);
}
void* maxval(vector<vector<double> > &arr, double &max){
    max = 0;
    for (unsigned int i=0; i < arr.size(); i++) {
        for (unsigned int j=0; j < arr[0].size(); j++) {
            if (max <= arr[i][j]){
                max = arr[i][j];
            }
        }
    }
    return 0;
}
void* maxval1(vector<double> &arr, double &max){
    max = 0;
```

```

    for (unsigned int i=0; i < arr.size(); i++) {
        if (max <= arr[i]){
            max = arr[i];
        }
    }
    return 0;
}

int main() {
    IplImage* img = cvLoadImage("lena512.bmp");
    if (!img){
        cout << " Can't read Image. Try Different Format." << endl;
        exit(1);
    }
    int height, width;
    height = img->height;
    width = img->width;
    int nc = img->nChannels;
    // uchar* ptr2 =(uchar*) img->imageData;
    int pix_depth = img->depth;
    CvSize size;
    size.width =width;
    size.height=height;
    cout << "depth" << pix_depth << "Channels" << nc << endl;
    cvNamedWindow("Original Image", CV_WINDOW_AUTOSIZE);
    cvShowImage("Original Image", img);
    cvWaitKey();
    cvDestroyWindow("Original Image");
    cvSaveImage("orig.bmp",img);
    int rows =(int) height;
    int cols =(int) width;
    Mat matimg(img);
    vector<vector<double> > vec1(rows, vector<double>(cols));
    int k=1;
    for (int i=0; i < rows; i++) {
        for (int j=0; j < cols; j++){
            unsigned char temp;
            temp = ((uchar*) matimg.data + i * matimg.step)[j * matimg.elemSize() + k
];
            vec1[i][j] = (double) temp;
        }
    }
    string nm = "db2";
    vector<double> l1,h1,l2,h2;
    filtcoef(nm,l1,h1,l2,h2);
    // unsigned int lf=l1.size();
    // int rows_n =(int) (rows+ J*(lf-1));

```

```

// int cols_n=(int) (cols + J * ( lf -1));
// Finding 2D DWT Transform of the image using symetric extension al-
gorithm
// Extension is set to 0 (eg., int e = 0)
vector<int> length;
vector<double> output,flag;
int J =6;
dwt_2d(vec1,J,nm,output,flag,length);
double max;
vector<int> length2;
// This algorithm computes DWT of image of any given size. Together
with convolution and
// subsampling operations it is clear that subsampled images are of differ-
ent length than
// dyadic length images. In order to compute the "effective" size of DWT
we do additional
// calculations.
dwt_output_dim_sym(length,length2,J);
// length2 is gives the integer vector that contains the size of subimages
that will
// combine to form the displayed output image. The last two entries of
length2 gives the
// size of DWT ( rows_n by cols_n)
int siz = length2.size();
int rows_n=length2[siz-2];
int cols_n = length2[siz-1];
vector<vector< double> > dwtdisp(rows_n, vector<double>(cols_n));
dispDWT(output,dwtdisp, length ,length2, J);
// dispDWT returns the 2D object dwtdisp which will be displayed using
OPENCV's image
// handling functions
vector<vector<double> > dwt_output= dwtdisp;
// Stroing the DWT coefficients in two different vectors that will be used
to approximate
// Image with two different sets of chosen coefficients.
vector<double> dwt_coef1;
vector<double> dwt_coef2;
dwt_coef1 = output;
dwt_coef2 = output;
maxval(dwt_output,max); // max value is needed to take care of overflow
which happens because
// of convolution operations performed on unsigned 8 bit images
//Displaying Scaled Image
// Creating Image in OPENCV
IplImage *cvImg; // image used for output
CvSize imgSize; // size of output image

```

```

imgSize.width = cols_n;
imgSize.height = rows_n;
cvImg = cvCreateImage( imgSize, 8, 1 );
// dwt_hold is created to hold the dwt output as further operations need
to be
// carried out on dwt_output in order to display scaled images.
vector<vector<double>> dwt_hold(rows_n, vector<double>( cols_n));
dwt_hold = dwt_output;
// Setting coefficients of created image to the scaled DWT output values
for (int i = 0; i < imgSize.height; i++ ) {
for (int j = 0; j < imgSize.width; j++ ){
if ( dwt_output[i][j] <= 0.0){
dwt_output[i][j] = 0.0;
}
if ( i <= (length2[0]) && j <= (length2[1]) ) {
((uchar*)(cvImg->imageData + cvImg->widthStep*i))[j] =
(char) ( (dwt_output[i][j] / max) * 255.0);
} else {
((uchar*)(cvImg->imageData + cvImg->widthStep*i))[j] =
(char) (dwt_output[i][j]) ;
}
}
}
cvNamedWindow( "DWT Image", 1 ); // creation of a visualisation win-
dow
cvShowImage( "DWT Image", cvImg ); // image visualisation
cvWaitKey();
cvDestroyWindow("DWT Image");
cvSaveImage("dwt.bmp",cvImg);
// Case 1 : Only 10% of the largest coefficients are considered
// Output is the 1D DWT vector
int n_coef1= int (output.size()/ 10);
cout << n_coef1 << endl;
// Finding Threshold Value corresponding to n_coef1
vector<double> temp1;
cout << "size: " << (int) temp1.size() << "\n";
cout << "capacity: " << (int) temp1.capacity() << "\n";
cout << "max_size: " << (int) temp1.max_size() << "\n";
for (unsigned int i=0; i < dwt_coef1.size(); i++) {
double tempval = abs(dwt_coef1[i]);
temp1.push_back(tempval);
}
double thresh1= 0.0;
findthresh(temp1,n_coef1,thresh1);
cout << "thresh" << thresh1 << endl;
ofstream temp("temp.txt");

```



```

for (unsigned int i=0; i < temp1.size(); i++){
    temp << temp1[i] << " ";
}
// Reset coefficients value depending on threshold value
for (unsigned int i=0; i < dwt_coef1.size(); i++) {
    double temp = abs(dwt_coef1[i]);
    if (temp < thresh1){
        dwt_coef1.at(i)= 0.0;
    }
}
// Finding IDWT
vector<vector<double>> > idwt_output(rows, vector<double>(cols));
idwt_2d( dwt_coef1,flag, nm, idwt_output,length);
double max1;
maxval(idwt_output,max1);
//Displaying Reconstructed Image
IplImage *dvImg;
CvSize dvSize; // size of output image
dvSize.width = idwt_output[0].size();
dvSize.height = idwt_output.size();
cout << idwt_output.size() << idwt_output[0].size() << endl;
dvImg = cvCreateImage( dvSize, 8, 1 );
for (int i = 0; i < dvSize.height; i++) {
    for (int j = 0; j < dvSize.width; j++) {
        if ( idwt_output[i][j] <= 0.0){
            idwt_output[i][j] = 0.0;
        }
        ((uchar*)(dvImg->imageData + dvImg->widthStep*i))[j] =
        (char) ((idwt_output[i][j] / max1) * 255 );
    }
}
cvNamedWindow( "10% Coeff Reconstructed Image", 1 ); // creation of a
visualisation window
cvShowImage( "10% Coeff Reconstructed Image", dvImg ); // image visu-
alisation
cvWaitKey();
cvDestroyWindow("10% Coeff Reconstructed Image");
cvSaveImage("recon.bmp",dvImg);
// Case 2 : Only 2% of the largest coefficients are considered
// Output is the 1D DWT vector
int n_coef2= int (output.size()/ 50);
cout << n_coef2 << endl;
// Finding Threshold Value corresponding to n_coef1
vector<double> temp2;
for (unsigned int i =0; i < dwt_coef2.size(); i++) {
    double tempval = abs(dwt_coef2[i]);

```

```

temp2.push_back(tempval);
}
double thresh2= 0.0;
findthresh(temp2,n_coef2,thresh2);
cout << "thresh" << thresh2 << endl;
// Reset coefficients value depending on threshold value
for (unsigned int i=0; i < dwt_coef2.size(); i++) {
double temp = abs(dwt_coef2[i]);
if (temp < thresh2){
dwt_coef2.at(i)= 0.0;
}
}
// Finding IDWT
vector<vector<double>> idwt_output2(rows, vector<double>(cols));
idwt_2d( dwt_coef2,flag, nm, idwt_output2,length);
double max2;
maxval(idwt_output2,max2);
//Displaying Reconstructed Image
IplImage *dvImg2;
CvSize dvSize2; // size of output image
dvSize2.width = idwt_output2[0].size();
dvSize2.height = idwt_output2.size();
cout << idwt_output2.size() << idwt_output2[0].size() << endl;
dvImg2 = cvCreateImage( dvSize2, 8, 1 );
for (int i = 0; i < dvSize2.height; i++ ) {
for (int j = 0; j < dvSize2.width; j++ ) {
if ( idwt_output2[i][j] <= 0.0){
idwt_output2[i][j] = 0.0;
}
((uchar*)(dvImg2->imageData + dvImg2->widthStep*i))[j] =
(char) ((idwt_output2[i][j]/ max2) * 255 ) ;
}
}
cvNamedWindow( "2% Coeff Reconstructed Image", 1 ); // creation of a
visualisation window
cvShowImage( "2% Coeff Reconstructed Image", dvImg2 ); // image visu-
alisation
cvWaitKey();
cvDestroyWindow("2% Coeff Reconstructed Image");
cvSaveImage("recon2.bmp",dvImg2);
return 0;
}

```

This code decomposes a 512X512 grayscale image to 6 levels using db2 wavelets and uses a) only 10% coefficients and b) only 2% coefficients to re-construct the image.



Figure 3.18: Input Image

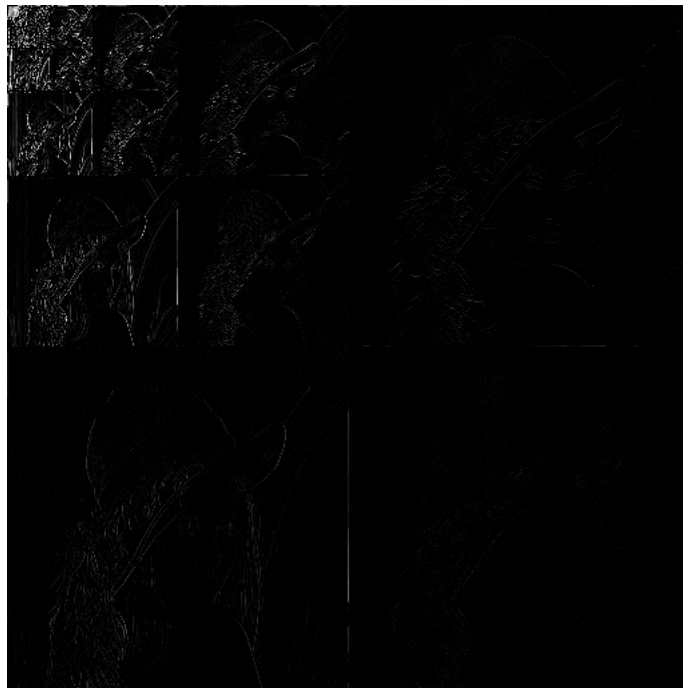


Figure 3.19: DWT of Input Image



Figure 3.20: 2D Approximation using only 1/10th Coefficients



Figure 3.21: 2D Approximation using only 1/50th Coefficients

3.6 2D SWT Demo

```
//=====
// Name : swt2Ddemo.cpp
// Author : Rafat Hussain
// Version :
// Copyright :
// Description : 2D SWT Demo using OPENCV
//=====
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <complex>
#include <cmath>
#include <algorithm>
#include "wavelet2d.h"
#include "cv.h"
#include "highgui.h"
#include "cxcore.h"
using namespace std;
using namespace cv;
void* maxval(vector<vector<double> > &arr, double &max){
    max = 0;
    for (unsigned int i=0; i < arr.size(); i++) {
        for (unsigned int j=0; j < arr[0].size(); j++) {
            if (max <= arr[i][j]){
                max = arr[i][j];
            }
        }
    }
    return 0;
}
int main() {
    IplImage* img = cvLoadImage("lena512.bmp");
    if (!img){
        cout << " Can't read Image. Try Different Format." << endl;
        exit(1);
    }
    int height, width;
    height = img->height;
    width = img->width;
    int nc = img->nChannels;
    // uchar* ptr2 =(uchar*) img->imageData;
    int pix_depth = img->depth;
    CvSize size;
```

```

size.width =width;
size.height=height;
cout << "depth" << pix_depth << "Channels" << nc << endl;
cvNamedWindow("Original Image", CV_WINDOW_AUTOSIZE);
cvShowImage("Original Image", img);
cvWaitKey();
cvDestroyWindow("Original Image");
cvSaveImage("orig.bmp",img);
int rows =(int) height;
int cols =(int) width;
Mat matimg(img);
vector<vector<double> > vec1(rows, vector<double>(cols));
int k =1;
for (int i=0; i < rows; i++) {
for (int j =0; j < cols; j++){
unsigned char temp;
temp = ((uchar*) matimg.data + i * matimg.step)[j * matimg.elemSize() + k
];
vec1[i][j] = (double) temp;
}
}
string nm = "db2";
// vector<double> l1,h1,l2,h2;
// filtcoef(nm,l1,h1,l2,h2);
vector<double> output;
int J =3;
swt_2d(vec1,J,nm,output);
cout << "OUTPUT size" << output.size() << endl;
cout << "LOOP OK" << endl;
int row,col;
dwt_output_dim(vec1, row, col );
// Extract and Display Low Pass Image at the Jth stage
vector<vector<double> > blur(row,vector<double>(col));
for (int i=0;i < row; i++){
for (int j=0; j < col;j++){
double temp = output[i*col + j];
blur[i][j]= temp;
}
}
double max;
maxval(blur,max);
// Creating Image in OPENCV
IplImage *cvImg; // image used for output
CvSize imgSize; // size of output image
imgSize.width = col;
imgSize.height = row;

```



```

cvImg = cvCreateImage( imgSize, 8, 1 );
for (int i = 0; i < imgSize.height; i++) {
for (int j = 0; j < imgSize.width; j++) {
if ( blur[i][j] <= 0.0){
blur[i][j] = 0.0;
}
((uchar*)(cvImg->imageData + cvImg->widthStep*i))[j] =
(char) ( (blur[i][j] / max) * 255.0);
}
}
cvNamedWindow( "Low Pass Image", 1 ); // creation of a visualisation
window
cvShowImage( "Low Pass Image", cvImg ); // image visualisation
cvWaitKey();
cvDestroyWindow("Low Pass Image");
cvSaveImage("blur.bmp",cvImg);
// Displaying BandPass Images
vector<vector<double>> detail(3*row,vector<double>(J * col));
for (int k=0; k < J; k++) {
for (int i=0; i < 3*row; i++) {
for(int j=0+ k*col; j < (k+1)*col; j++) {
double temp = output[(3*k+1)*row*col+ i * col +j - k*col];
detail[i][j]= temp;
}
}
}
IplImage *dvImg; // image used for output
CvSize imgSz; // size of output image
imgSz.width = J*col;
imgSz.height = 3*row;
dvImg = cvCreateImage( imgSz, 8, 1 );
for (int i = 0; i < imgSz.height; i++) {
for (int j = 0; j < imgSz.width; j++) {
if ( detail[i][j] <= 0.0){
detail[i][j] = 0.0;
}
((uchar*)(dvImg->imageData + dvImg->widthStep*i))[j] =
(char) detail[i][j];
}
}
cvNamedWindow( "Band Pass Image", 1 ); // creation of a visualisation
window
cvShowImage( "Band Pass Image", dvImg ); // image visualisation
cvWaitKey();
cvDestroyWindow("Band Pass Image");
cvSaveImage("detail.bmp",dvImg);

```



Figure 3.22: Input Image

```
return 0;  
}
```

Three level Stationary Wavelet Transform is computed using db2 wavelet. Approximation coefficients are stored only for the final ($J=3$) stage while the three detail coefficients(Horizontal, Vertical and Diagonal) are stored for each value. All 10 sets of coefficients are 512×512 .



Figure 3.23: Approximation image at Level $J = 3$



Figure 3.24: Detail Images at levels $J=3,2,1$ (L-R).Horizontal,Vertical,Diagonal(T-B)