Simple Wavelets with Convolution

Daniel Beatty

Texas Tech University

Lubbock, Texas

April 11, 2003

Contents

Goals for this introductory document on wavelets is to define a wavelet, acknowledge alternative definitions of a wavelet, specify basic wavelet theorems, and show wavelets through practical examples. Convolution provides the key for defining the wavelet, its theorems and practical examples. The Haar Wavelet Filter is used to show these concepts.

1 Wavelet Transform Definitions:

As stated, the wavelet transform can be viewed from both a matrix representation[1] and vector convolution. Such a representation provides means to use many linear algebra and abstract algebra methods to prove theorems about wavelets. Another form is a convolution type of operation. Convolution wavelet transform methods provide a simple computational method in general. Special case convolution methods also exist, which allow even simpler algorithms to perform wavelet transforms.

In the matrix form, a wavelet matrix is defined as a generalization of a square orthogonal or unitary matrix which is a subset of a larger class of rectangular matrices . This matrix harvest information information for defining a wavelet system. Such matrices are defined in terms of their rank and genus[1].

It was Alfred Haar himself who defined an orthonormal wavelet matrix in "Zur Theorie der orthogonalen Funktionensystem Math Annual" written in 1903. The said wavelet was named in honor of Alfred Haar, the Haar Wavelet

Matrix. A Haar Wavelet Matrix has a genus of one. Such matrices have a one to one mapping to a general wavelet matrix. Also, Haar Wavelet Matrices are equivalent to the characteristic Haar Matrix. Lastly, a rank 2 Haar Matrix sufficiently shows to show all geometric considerations[1].

2 Wavelets via Convolution

Wavelets are defined in terms of average and difference components. Each component can further be transformed to isolate properties of each component. Typically, each component has the form $O \rightarrow (A|D)$ where O is the original signal, A is the average component, D is the difference component and $(A|D)$ the signal A concatenated with D. Each of these components are produced by an orthonormal basis. Also, these components are produced such that the original can be constructed easily from them.

Many mathematicians such as Walker[2], use a form that eliminates half of the values. Thus a form can be defined which has the same number of elements as the original. The rules for choosing the member elements are dependent on the wavelet filter choice.

Another useful property of wavelets via convolution is the simplicity of the operation. The general case works for all. Such an algorithm requires one nested loop as seen below:

$\forall i \in [0, M)$

```
$\forall j \in [0, N)$
    n=i-j
    if $(n \in [0, M))$
        $y_i += x_n \cdot h_j$
```

This filter simply equates to the mathematical function: $x * h = \sum_l x_{k-l} h_k$, which is the convolution operation. As we can see the operation is slightly less than $O(n^2)$. For practical use, the filter is made smaller than the actual signal being analyized. In some cases, the filter may be much smalled than the signal. Filter size matters in extracting features from the original signal.

In the case of this convolution operator, the limit is acutally M, not N. The value of M is the size of the original signal. Since the resulting vector is the same as the original, the vector is said to be fully qualified. Only half of those values are necessary to reconstruct the original (every other element).

To perform a wavelet transform via convolution, each signal is convolved twice.

$A_i = A_{i-1} * V$

$D_i = A_{i-1} * W$

where

V is the scaling wavelet vector

W is the differencing wavelet vector

A is the average vector (scaled vector)

D is the difference component vector

$\forall i \in [1, L)$ and $A_0 = f$ which is the original signal

L is the limit on the number resolutions that signal can have based on the wavelet type.

3 Class 2-D Wavelet: Complete Form

The convolution version can be used to derive a Wavelet Matrix. For a general case, it is simpler to use the convolution method. The matrix form becomes practical in repetitive special case applications.

The 2-D transform has four components: the average, vertical, horizontal, and diagonal. Two general computational means exist to generate a one-resolution transform. These can derive means for performing many resolutions.

A complete transform method returns a result matrix which is the same size as the source matrix. The result contains the four components. Each component resides on 4 corners of the matrix. Given a matrix B, the transform is to yield the following form:

$$B \Rightarrow \begin{matrix} H & D \\ A & V \end{matrix}$$

where A is the average component, H is the horizontal component, V is the vertical component, and D is the difference component. There is another form which is also used as an example:

$$B \Rightarrow \begin{matrix} A & V \\ H & D \end{matrix}$$

The first version is simple in concept, but provides a few more possibilities for error and confusion. Regardless of the case, the four components have the following definitions:

1. Average component: produced by filtering the row vectors and the column vectors with the averaging filter.
2. Vertical Component: produced by applying the average filter to the column vectors and the difference filter to the row vectors.
3. Horizontal component: produced by applying the average filter to the row vectors and the difference to the column vectors.
4. Diagonal component: produced by applying the difference filter to both the row and column vectors.

3.1 Proof of Concept

Two methods of convolving a matrix are easily conceived. First is to use 1-D wavelet. The other is to apply the convolution scheme straight to the matrix.

Included in the wavelet experiment are both. Realistically, both can and do achieve the same result. However, the direct method achieves speed advantages by the lack of overhead. The direct method leaves a temporary vector resident in memory. Also, there are 2 fewer transfers per row and columns.

3.1.1 1-D to 2-D Method

Both rows 1-D and 2-D and columns 1-D and 2-D transform are performed similarly. The obvious difference is the indexing of rows and columns.

Given 1-D wavelet transform

source matrix

Algorithm: (Row Transform)

$\forall i \in rows$

- $\forall j \in columns$
- $S[j] \leftarrow source[i][j]$
- $S \Rightarrow^{W} R$

This principle of this algorithm is simple. Only three intuitive steps are necessary per row or column. Two of these steps are array transfers (row/column transfer to an array). These arrays are fed into the 1-D transforms.

However, the 1-D wavelet transform itself includes a series of memory allocation and deallocation operations. Each memory call is at the minimum a system call.

3.1.2 Vector - Matrix Method

The principle of this algorithm is more complicated. All functionality, such as convolution, is built into the method. There are fewer calls and passing of structures to external functions to compute the transform.

This method has a few givens. The source matrix, the Haar average filter, and Haar difference filter are given arguments. The result argument is the return argument. The transform signals sub-functions row transform and column transform to perform the work.

The algorithm is as follows for the row tranform (and is similar for the column transform):

```
$\forall i \in rows$

1. Initialize temporary array/vector to all zeros (x).

2. $\forall k \in columns$

    (a) $\forall l \in ha.Size$ x += $\begin{array}{ll} S_{i,k-l} * hA_l & k-l \in columns \\ 0 & otherwise \end{array}$

3. $\forall k \in columns/2$

    $result_{i,k} = x_{2k+1}$ (In other words, odd split)

4. Initialize x to all zeros.

5. $\forall k \in columns$

    (a) $\forall l \in hd.Size$ x += $\begin{array}{ll} S_{i,k-l} * hD_l & k-l \in columns \\ 0 & otherwise \end{array}$

6. $\forall k \in columns/2$

    $result_{i,k+columns/2} = x_{2k+1}$ (In other words, odd split)
```

3.2 Multi-Resolution

The multi-resolution wavelet transform and the inverse multi-resolution transform resemble the vector-matrix version. All functionality is built into this method. However, there are structural changes.

The wavelet transform (multiresolution) uses private members of the class (hA, hD, xD/yD, xA/yA). Both Haar filters are maintained this way. Also both row and column transforms have average and difference myVector classes for temporary storage. All of these members are allocated and destroyed by the wavelet transform method itself. The simplified algorithm of the row transform is the following:

```
1. Initialize xA and xD to zero
2. $\forall k \in columns \forall l \in filter$
    • n = k - l
    • if ( $n \in columns$ )
      $xA_k = W_{i,n} * hA_l$
      $xD_k = W_{i,n} * hD_l$
3. Transfer back to W
   $W_i = xA|xD$
```

```
1. initialize yA and yD to zero
2. $\forall k \in rows \forall l \in filter$
```

- n = k - 1
- if ($n \in columns$)

  $yA_k = W_{i,n} * hA_l$

  $yD_k = W_{i,n} * hD_l$

3. Transfer back to W

   $W_j = yA|yD$

Note: W_i names the row vectors and W_j names the column vectors, and $W_{i,j}$ is the element from the ith row and jth column.

3.3 Computational Cost:

The cost of this algorithm is computed first for each row and each column. This value is used to compute the cost of the matrix. The cost of computing the matrix is used to compute the cost of the multi-resolution steps. Per row the cost is 3k, where k is the number of columns. Per column the cost is 3l, where l is the number of rows. For the whole matrix, one resolution costs 6kl operations to compute the wavelet transform. Per resolution, the rows and columns shrink by 2^i for each resolution, i, performed. The limit of this cost equals $12kl$ operations. Thus, the cost is linear.

4 Results - 1-D Wavelet Transform

Testing of the 1-D wavelet was performed on a sinusoidal wave form of 128 elements. The given function has the equation shown in Figure 1.

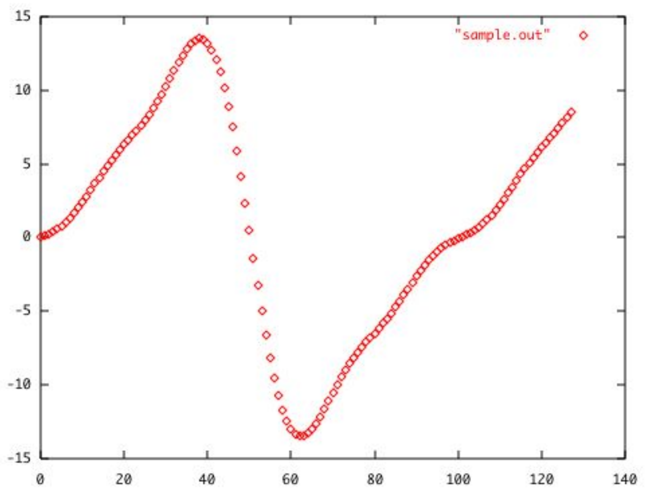

Figure 1: Sample function. The x-axis is the array index (index n). The y value is simple – the value y[n].

$$y[n] = 10\sin(\tfrac{n}{128}) - 5\sin(\tfrac{n}{64}) + 2\sin(\tfrac{3n}{128}) - \sin(\tfrac{n}{32})$$

The first version of the 1-D transform used the even elements of both convolutions to generate the wavelet transform. These even elements came from the over-complete form and naturally allow the potential to have complete information. However, in doing so, a fundamental flaw appears.

In order to evaluate the effectiveness of the wavelet transform three tests have been devised. First, energy equivalence is used to determine how much energy is retained in the transform from the original. The general shape is used on a the first resolution to test if the average signal has the same general shape as the original. Lastly, the inverse transform is used to recover the original signal. A comparison is made between the original and the recovered signal.

After one resolution, the transformed signal has the same energy as the original. This is good since it allows the original to be recovered from the transform. Also, the average component of the transform has the same shape as the original. This is good. However, the recovered signal is missing the last element. Refer to Figure 2

The secret is in which elements are used from the over-complete to make the complete. The over-complete in this project comes from the average component and difference which are simply the result of convolution.

The convolution means is at the heart of the issue. The convolution operator in this case starts with the first element of the filter against the first element of the signal. In the simple Haar Wavelet case, there is a transformation pairing. $(S_i, S_{i-1}) \rightarrow A_i$ and $(S_i, S_{i-1}) \rightarrow D_i$

In this pairing with zero indexed signals, the odd indexed elements from the over-complete must be used to have all elements of the original accounted for.

Also this produces a functional difference between wavelet inverse transform for odd and even versions. The difference is slight; however, the last element is lost in the even indexed form.

- Odd $R_{2i} = (A_i - D_i)\sqrt{1/2}$ $R_{2i+1} = (A_i + D_i)\sqrt{1/2}$
- Even $R_{2i} = (A_i + D_i)\sqrt{1/2}$ $R_{2i-1} = (A_i - D_i)\sqrt{1/2}$

An odd indexed wavelet transform yields the same energy. However, all of the values are accounted for. Refer to Figure 3.

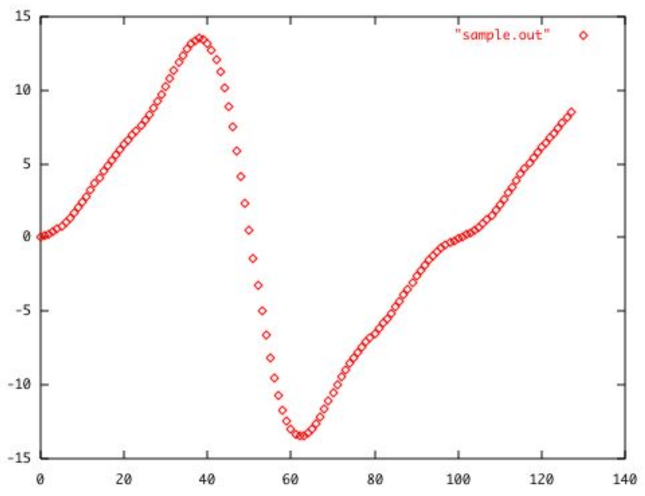

Figure 2: Recovered function. The x-axis is the array index (index n). The y value is simply the value y[n]. The function was recovered from an even indexed wavelet transform.

5 Results: 2-D Wavelet Transform

A simple room picture shows the difference correct indexing produces in the wavelet transform and its inverse. The 1-D to 2-D method shows the incorrectly indexed case. A correctly indexed version is shown in the vector-matrix method.

The 1-D to 2-D method has a serious issue with memory leak errors (Macintosh OSX, using gcc 3.1). Memory is allocated and deallocated quickly, and on some platforms shows up as an error. On other platforms, the result is degraded performance (IRIX, SGI Octane2 using gcc 2.9). An example image of 720 x 486 requires nearly 10 minutes to compute the wavelet transform by this method on an SGI Octane2. However, this method does eventually return a correct result.

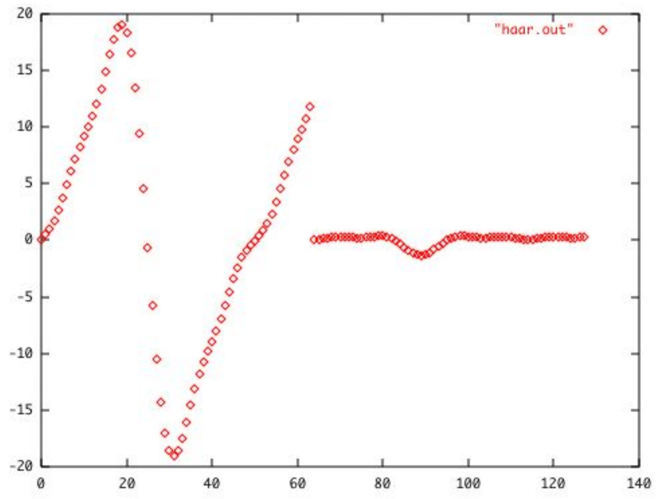

Figure 3: Recovered function. The x-axis is the array index (index n). The y value is simply the value y[n]. The function was recovered from an odd indexed wavelet transform.

The matrix-vector method also yields the correct result. However, there is less memory overhead in this method as compared to the 1-D to 2-D method. As a result, both the row wavelet transform and column wavelet transforms are performed more quickly, with fewer memory transfers and allocations. Obviously, this also allows for the operation to be conducted almost entirely in cache memory on both the SGI Octane2 and Macintosh G4 based machines. A Macintosh G3 based machine still requires main memory at a minimum to execute the same operation which yields a slower performance.

A correct result must also be matched to a correct inverse method. The indexing order matters. The inverse transform method is a forward inverse transform method. In the case of 1-D to 2-D transform, the ordering was reverse

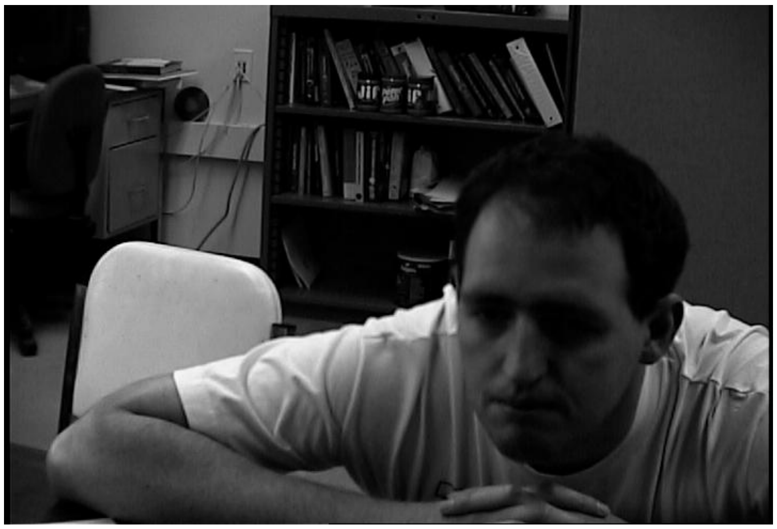

Figure 4: Original Image. This image is the original image.

indexed (Figure 6). As a result of an error in indexing, ringing is seen on edges in this method(Figure 8) for a case in point. Caution is incredibly important when matching both forward and reverse indexing, since matching the mathematics to the actual ordering can be obscure and tricky.

A correct result is shown in Figure **??**. In this case, the indexing was matched up and ringing is not present. It is clear that the recovered image and the original (Figure 4) are nearly indistinguishable.

5.1 Multi -resolution

The expected result is a picture within a picture. Each average component has a further transform on it. The three resolution transform has the form:

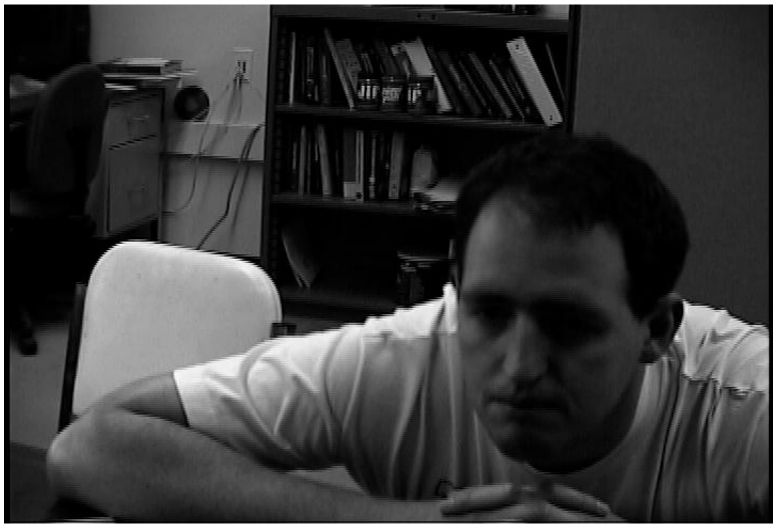

Figure 5: Recovered Image. This image is the recovered image. Depending on whether the image was saved as a picture first can affect the white spots in the picture. Ringing is also an issue.

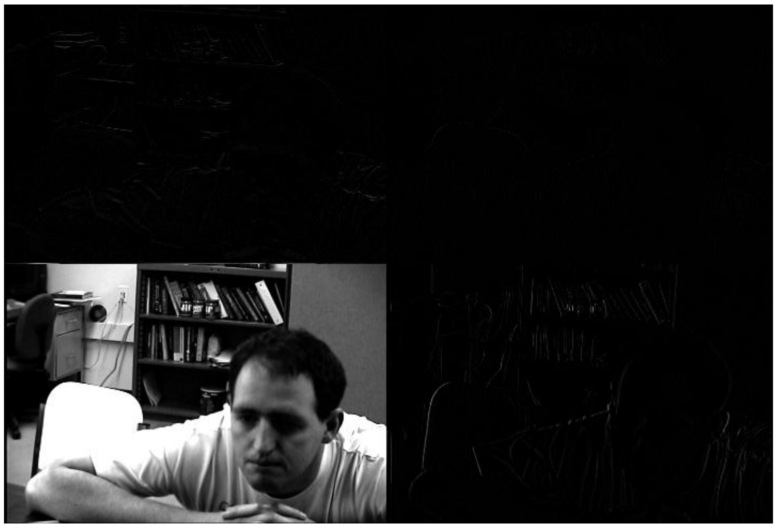

Figure 6: Wavelet Transform Image. This image is divided in to average, horizontal, vertical and diagonal components.

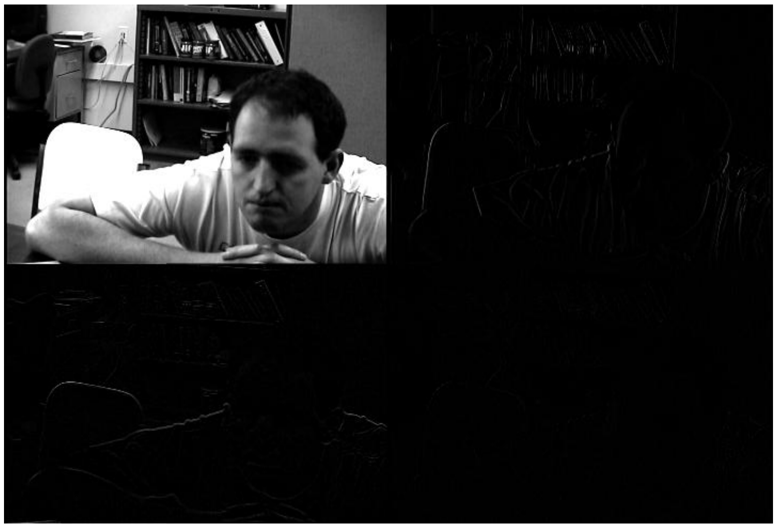

Figure 7: Wavelet Transform Image. This image is divided in to average, horizontal, vertical and diagonal components, using the vector-matrix version.

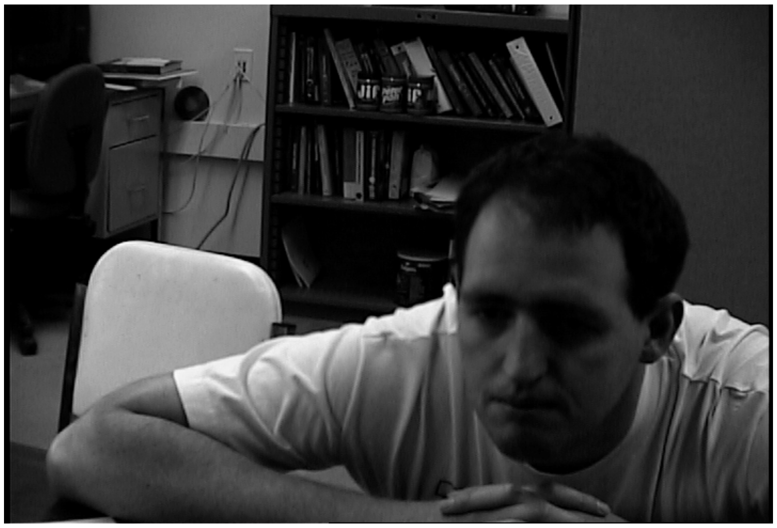

Figure 8: Recovered Image (Vector-Matrix Method). This image is the recovered image. This version avoids the ringing by using the vector-matrix version which is more aligned for the inverse wavelet transform.

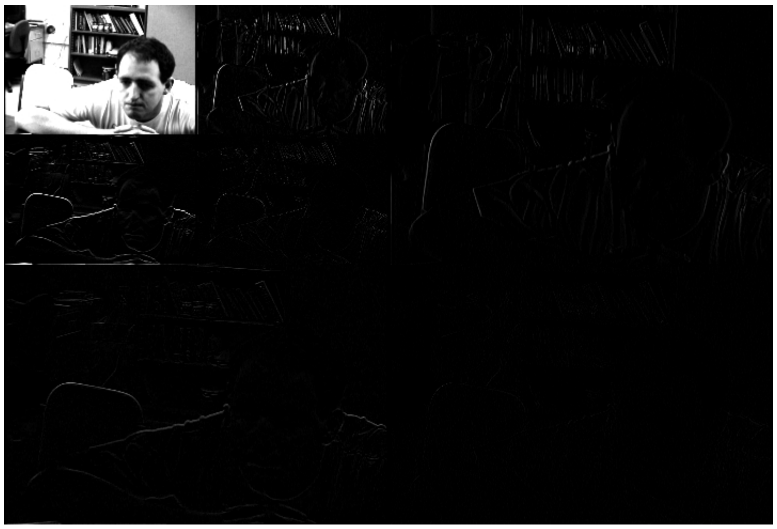

Figure 9: Wavelet Transform Image. This image is divided in to average, horizontal, vertical and diagonal components using multi-resolution wavelet transform. Note the the average component was transformed one step further.

$$W_3 = \begin{array}{cc} \begin{array}{cc} \begin{array}{cc} A_3 & V_3 \\ H_3 & D_3 \end{array} & V_2 \\ H_2 & D_2 \end{array} & V_1 \\ H_1 & D_1 \end{array}$$

Refer to Figure 9 for the image transform results.

To obtain the inverse, an exact reverse procedure is necessary, otherwise the distortion is hideous. The first attempt of the wavelet inverse transform was out of order, refer to Figure 10 . A correct picture was obtained during the second attempt. Correct order yielded correct results, refer to Figure 8 .

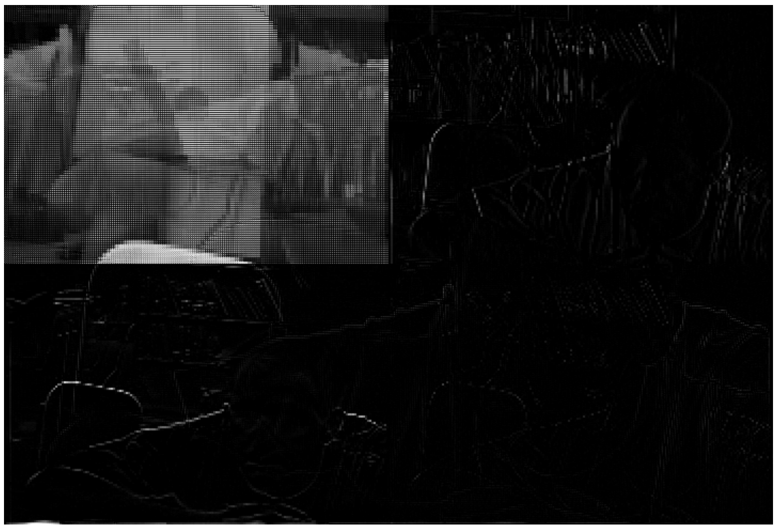

Figure 10: Recovered Image - Wrong Order (Multi-Resolution). This image shows a 2-D wavelet transform after it was recovered out of order. Obviously, the distortion is hideous.

5.2 Threshold FIltering

After a triple resolution, a 0.02 threshold will eliminate 81.1706 percent of elements in the original sample picture. Also at this point, the effects of removing these elements becomes visually evident (Figure 11). At a 0.01 threshold, 66.0205 percent of the elements are removed. Visually, the recovered sample and the original appear to be the same (Figure 14). At a threshold of 0.1, 92.9987 percent of the elements are reduced to zero. However, the distortions are clearly visible at this level of thresholding (Figure 12). Even at a threshold of 0.001 which is below the numerical precision of the original, 16.0814 elements are reduced to zero. At a threshold of 0.002, 28.9683 percent is removed.

Consequently after a triple resolution, nearly 29 % of the data was irrelevant for the image's brightness resolution (which also applies to color). 60 % to 85

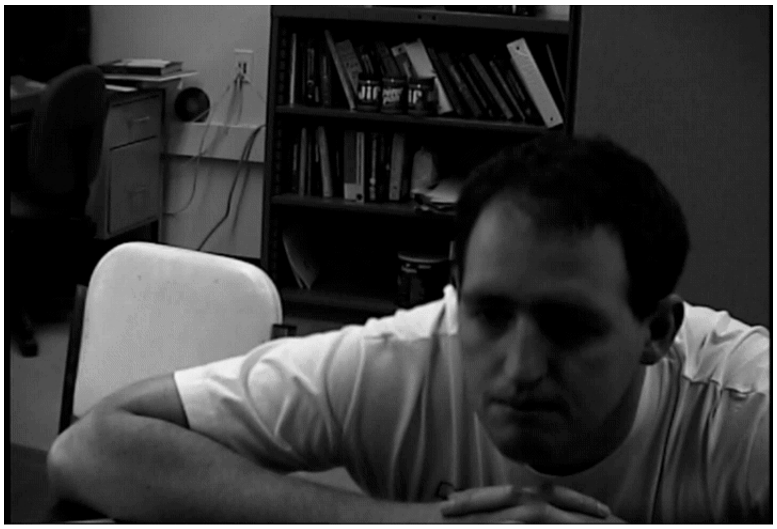

Figure 11: Recovered Image - 2% threshold (Multi-Resolution). This image had nearly 83% of its elements removed in the triple resolution wavelet transform.

% of was not noticeable to human perception. Which leaves only 15 % to 40 % of the data actually contributing or being necessary to reconstruct the image.

A Haar Wavelet Transform Class:

The Haar Wavelet Transform class has the purpose of taking in a signal and outputing the signal processed by the Haar Wavelet Transform. It is dependent on the convolution function for computation. It is also dependent on the haar function generator to establish a Haar Scaling and Wavelet vector. For practical I/O, a result plotter is also necessary.

All of the above can be accommodated within one class. One other class that is necessary is the myVector class. This class provides a simple, yet practical

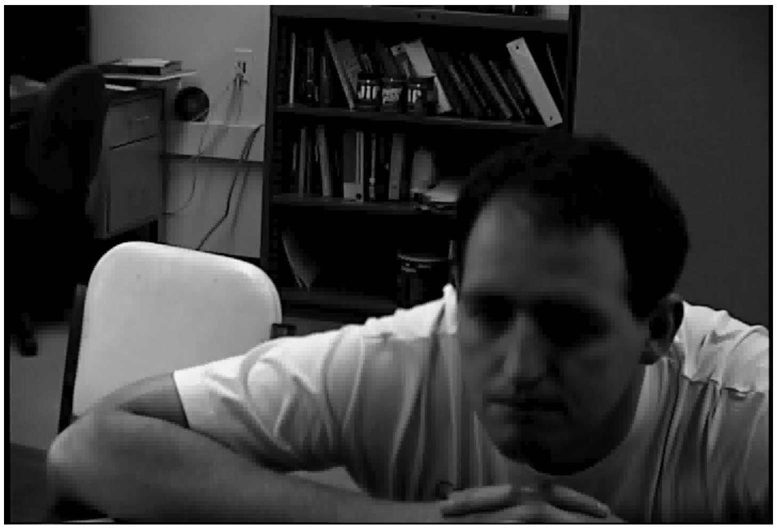

Figure 12: Recovered Image - 10% threshold (Multi-Resolution). This image had nearly 93% of its elements removed in the triple resolution wavelet transform.

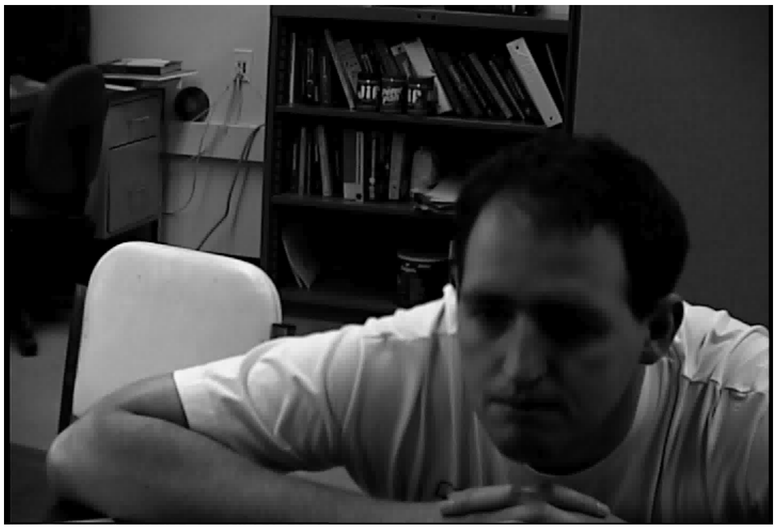

Figure 13: Recovered Image - 5% threshold (Multi-Resolution). This image had nearly 85% of its elements removed in the triple resolution wavelet transform.

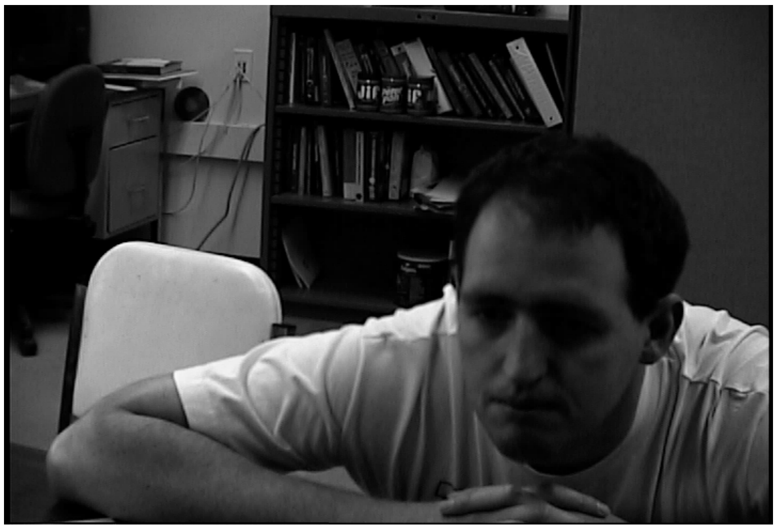

Figure 14: Recovered Image - 1% threshold (Multi-Resolution). This image had nearly 60% of its elements removed in the triple resolution wavelet transform.

data type for the computation. It can be allocated and deallocated such that its size is appropriate for the task.

It is projected that the two-dimensional version will simply add a matrix manipulator. What such a manipulator extracts rows and columns into myVector structures such that the computation to procede on each row and column..

A.1 One Dimensional Wavelet Transform

The primary function in the Wavelet transform class is the one dimensional wavelet transform (hWaveXform). More than one form of this function could possible exist; however, for simplisty only one was produced. It takes two myVector classes (input and output). It produces a haar difference and scaling vector, as well as a R_A, R_D, T, F and S myVector classes for use within the function. The algorithm is as follows

```
$S = input$
$l =$ is the size of S (or the input)
$\forall i \in [0, l)$
    $R_A = S * H_A$
    $R_D = S * H_D$
    T = join ($R_A$,$R_D$)
    F = evenSplit (T)
    ForceInsert (F, output)
    end = floor ( $\frac{l}{2^{i+1}}$)
    Extract (output, S, 0, end)
```

The Extract procedure is to take values from output up to the index end, and make S a copy of that vector. Thus S is called the resolution vector. T is always twice the size of the working S vector. F is always half. The resolution vector decreases by half each iteration, and starts at the same size as the input.

A.2 Join Procedure

The join procedure is rather simple. Produce a result whose size is the size of the two sources combined, and whose elements are those of the two input vectors.

```
Input:
    myVector left, right
```

Output:

    myVector Result

Algorithm

```
result.deallocate
$s_l = size(left)$
$s_r = size(right)$
$s = s_l + s_r$
result.allocate(s)
$\forall i \in [0, s_l)$
    result[i]=left[i]
$\forall i \in [s_l, s)$
    result[i]= right[$i - s_l$]
```

A.3 Procedure: Even Split

This procedure is simply a special type for condensing procedure. Simply put, this procedure makes the result half the size of the original input. Both the input and result are myVector classes. The characteristic of the elements of the result is:

    result[i] = input[2*i]

A.3.1 Input:

myVector s

A.3.2 Output:

myVector r

A.3.3 Algorithm

```
l = ceil (s.size/2)
   r.allocate(l)
   $\forall i \in [0, l)$
       $r_i = s_{i*2}$
```

A.4 Procedure: Force Insert

Purpose: To insert one myVector ,s, into a larger myVector, r. The constraint is that r be larger than s. As long as this is the case, then forced insertion can occur. Special cases can include start and end points. In which case, the start and end points must be within the specified range.

A.4.1 Input:

myVector s, r

A.4.2 Output:

myVector r

A.4.3 Algorithm:

```
$l_1 = s.size$
    $l_2 = r.size$
    if ($l_1 < l_2$) and (start ¡ end)
        $\forall i \in [start, end]$
            $r_i = s_i$
```

A.5 Procedure: Extract

Purpose: To produce a new or replace a myVector whose length is that of the section to copied and extracted. The point behind this procedure is provide allow a wavelet transform to be performed on a segment of data, and keep the procedure the same each resolution.

A.5.1 Input:

The inputs for the wavelet Xform extraction procedure are as follows:

myVector S,

integer a

integer b

These names are arbitray. The variables a and b specify the start and end points respectively. Obviously, there is an inequality relationship here.

$0 \leq a \leq b \leq S.size$

A.5.2 Output:

The output of the wavelet Xform ¿ extraction procedure is simply:

myVector R

The elements of R are a copy of the elements of S from index a to index b.

A.5.3 Algorithm

```
if ( 0 ¡a ¡ b ¡ S.size)
        R.deallocate
        l = b - a +1
        R.allocate(l)
        $\forall i \in [a, b]$
            $R_{i-a} = S_i$
```

A.6 Procedure: Haar Wavelet Inverse (Left Side)

Purpose: These procedures are specific case toward the 2-element Haar Wavelet. They take in an array of even length and return an array of equal size which is the Inverse Haar Wavelet Transform of the original. The format of the original is assumed to be $(A|D_1|D_2|...|D_n)$.

A.6.1 Input

A "MyVector" class which is an array with simple operations associated with it. The object name is source. The source is of the form $(A|D_1|D_2|...|D_n)$.

A.6.2 Output

A "MyVector" class with an object name of result.

A.6.3 Algorithm

There is a difference between the current algorithm and the ideal algorithm which may be the source of error. This algorithm uses the following symbols to aide in its description:

- S is the source myVector
- A is the average term for which $A_i = S_i$
- D is the difference term for which $D_i = S_{i+l/2}$
- l is the length of S

The first version has the following algorithm:

1. Initialize the result, R.
2. $\forall i \in [0, \frac{l}{2}]$ $R_{2i} = (A_i + D_i)\sqrt{\frac{1}{2}}$
3. $\forall i \in [0, \frac{l}{2}]$ $R_{2i-1} = (A_i + D_i)\sqrt{\frac{1}{2}}$

There is a problem with this method. Let us start with the even values: $R_0 = (A_0 + D_0)\sqrt{1/2}$. Now the odd, not that $R_{-1} = (A_0 + D_0)\sqrt{1/2}$ which of course does not exist. Next, $R_1 = (A_1 - D_1)\sqrt{1/2}$ is valid. Lastly, $R_{l-1} =$

$R_{2*(l/2)-1} = (A_{l/2} - D_{l/2})\sqrt{1/2}$. The problem is that $A_{l/2}$ and $D_{l/2}$ do not exist. Thus, R_{l-1} does not exist, either.

B 2-D Wavelet Transform Class

This class provides three functions. The column wavelet provides a transform solely on the columns. The row wavelet likewise, provides a wavelet transform on each of the rows. The 2-D wavelet transform is simple a row wavelet followed by a column wavelet transform.

It should be noted that these wavelet transforms are Haar Wavelet transforms.. Multiple resolutions functions can be made, but again it is still the Haar Wavelet at the core. Other class clones may be produced to use other wavelet basis such as Daubachie or Coeflet.

Another support class required for support of the two dimensional wavelet is the image reader/ writer. Such a class of functions provide means of acquiring data for a matrix to be computed. Granted this matrix could have been populated by other means. Generating such a class consumed some time on this project. The reason is that there a few mechanisms for doing this. One is to use raw image types. These types are known as by their extensions: pgm and ppm. Another popular mechanism is Image Magick. The reason for Image Magick is that it is available on most UNIX platforms (Linux, IRIX, Solaris, and Mac OSX). Otherwise, schemes such as Apple's Quicktime would be used. The other reason for the use of other software to acquire the image is that the objective of this class is not produce image translators for each image type imaginable.

B.1 Method: Column Wavelet Transform

The column wavelet transform is provides a source matrix, and returns a result. In order to yield this result, each column is extracted into a vector and that vector is fed into a one-dimensional transform. The result of the one dimensional transform is placed the corresponding row of the result matrix. Mathematically, this algorithm is as follows:

```
$\forall j \in col$
    n = 0
    $\forall i \in row$ in reverse order
        $S_{n++} = \text{source}_{i,j}$
    $S \overset{W}{\Rightarrow} R$
    n = 0
    $\forall i \in row$ in reverse order
        $\text{result}_{i,j} = R_{n++}$
```

Note that the need for the n index is keep the order straight for this wavelets convention.

B.2 Method: Row Wavelet Transform

Like the column wavelet transform, the row wavelet transform takes a source matrix and returns a resulting matrix. The only two significant differences are first the items being operated on (rows not columns). Second, the lack of need for the n index.

```
$\forall i \in row$
    $\forall j \in col$
        $S_j$ =source$_{i,j}$
    $S \overset{W}{\Rightarrow} R$
    n = 0
    $\forall j \in col$ in reverse order
        result$_{i,j}$ $= R_j$
```

B.3 Method: Self Row/Column Wavelet Transform

There is a significant difference between the proof of concept version and self row/column wavelet transforms. The self versions use a matrix convolution.

B.4 Method:Wavelet Transform

As stated above, the two dimensional wavelet transform is simply row wavelet transform followed by a column wavelet transform. It could have been done in reverse order. However, the result would be the same.

Just of note, this class lacks for the moment an inverse transform function. Not that such a thing does not exist mathematically. It simply was not implemented at the time of this document.

B.5 Method: Row Wavelet Inverse Transform

The row wavelet inverse transform uses the one-dimensional form to transform each row of the matrix. The algorithm is as follows:

1. $\forall i \in [0, k)$

   (a) Assign S the values of row i in such that $S_i = M_{i,j}$

   (b) Perform inverse wavelet transform on S: $S \rightarrow R$

   (c) reintegrate R into result such that $N_{i,j} = Rj$

The above algorithm is defined with the following notation

- S is the source vector for use in the inverse wavelet transform.
- R is the result vector used in the inverse wavelet transform.
- M is the source matrix
- N is the result matrix

B.6 Method: Column Wavelet Inverse Transform

The column wavelet inverse transform is nearly identical to the row wavelet inverse transform. The exception is of course that the source vector S is assigned to equal the columns. Another significant issue is that the column indices and source vector indices are in reverse order.

$S_j = M_{i,l-j}$ and $N_{i,l-j} = R_j$

B.7 Method: Wavelet Inverse Transform v1

The wavelet inverse transform simply calls the row wavelet inverse transform first, and uses the results to in column wavelet inverse transform. The result of the column wavelet inverse transform is used as the result for the wavelet inverse transform.

B.8 Method: Self Wavelet Inverse Transform

This method is used to save on memory leaks by only allocating the temporary matrix and result only once. This method uses two other methods, the Self Column Inverse Wavelet Transform and the Self Row Inverse Wavelet Transform.

B.8.1 Given

The two items provided are references to the source matrix and the result matrix.

B.8.2 Algorithm

- $\forall j \in W.columnsSelfColumnInverseXform(W, R, j)$
- $\forall j \in W.rowsSelfRowInverseXform(W, R, i)$

B.9 Method: Self Column Inverse Wavelet Transform

The code name for this method is selfColumnInverseXform, and it takes three arguments. This particular method performs a column inverse wavelet transform on a particular column, designated j. The return value is placed in R, in the correct column.

B.9.1 Given

Two references are given. One to the source matrix, and the other to the result matrix. The third item is an integer, j. This integer identifies the column to be transformed.

B.9.2 Notation

Two symbols are used to simplify the writing of the algorithm.

- k is number of rows in source matrix minus 1.
- k2 is the half the number of rows in the source matrix.
- W is the source matrix
- R is the result matrix.

B.9.3 Algorithm

$\forall i \in [0, k2)$

- $R_{2i,j} = (W_{i,j} + W_{i+k2,j})\sqrt{\frac{1}{2}}$
- $R_{2i+1,j} = (W_{i+k2,j} - W_{i,j})\sqrt{\frac{1}{2}}$

B.10 Method: Self Row Inverse Wavelet Transform

The code name for this method is selfRowInverseXform, and it takes three arguments. This particular method performs a column inverse wavelet transform on a particular column, designated i. The return value is placed in R, in the correct column.

B.10.1 Given

Two references are given. One to the source matrix, and the other to the result matrix. The third item is an integer, j. This integer identifies the column to be transformed.

B.10.2 Notation

Two symbols are used to simplify the writing of the algorithm.

- l is number of columns in source matrix minus 1.
- l2 is the half the number of columns in the source matrix.
- W is the source matrix
- R is the result matrix.

B.10.3 Algorithm

$\forall i \in [0, l2)$

- $R_{i,2j} = (W_{i,j} - W_{i,j+l2})\sqrt{\frac{1}{2}}$
- $R_{i,2j+1} = (W_{i,j+l2} + W_{i,j})\sqrt{\frac{1}{2}}$

References

[1] Howard L. Resnikoff. and Raymond O. Wells, Jr. *Wavelet Analysis: The Scalable Structure of Information* copyright Springer-Verlag New York, Inc. New York, NY 10010, USA, 1998

[2] James Walker *A Primer on Wavelets and Their Scientific Applications* copyright Chapman & Hall/CRC : Boca Raton, FL, USA, 1999

[3] Gavin Tabor *Wavelets - The Idiots Guide* http://monet.me.ic.ac.uk/people/gavin/java/waveletDemos.html

[4] Amara Graps *Introduction to Wavelets* http://www.amara.com/current/wavelet.html