The Fast Wavelet Transform[1]
Beyond Fourier transforms

Mac A. Cody[2]

The Fourier transform has been used as a reliable tool in signal analysis for many years. Invented in the early 1800s by its namesake, Jean Baptiste Joseph Fourier, the Fourier transform has become a cornerstone of modern signal analysis.

The Fourier transform has proven incredibly versatile in applications ranging from patten recognition to image processing. Nevertheless, it suffers from certain limitations. Recently, a new kind of transform, the wavelet transform, has been shown to be as powerful and versatile as the Fourier transform, yet without some of its limitations.

The wavelet transform is the result of work by a number of researchers. Initially, a French geophysicist, Jean Morlet, came up with an ad hoc method to model the process of sound wavelet traveling through the earth's crust. Unlike Fourier analysis, he did not use sine and cosine curves, but simpler ones which he called "wavelets." Yves Meyer, a French mathematician, recognized this work to be part of the field of harmonic analysis, and came up with a family of wavelets that he proved were most efficient for modeling complex phenomena. This work was improved upon by two researchers in America, Stephane Mallat of New York University and Ingrid Daubechies of Bell Labs. Since 1988, there has been a small explosion of activity in this area, as engineers and researchers apply the wavelet transform to applications ranging from image compression to fingerprint analysis. The wavelet transform has even been implemented in silicon, in the form of a chip from Aware Inc.

But before discussing how the wavelet transform works, let's first examine its predecessor.

How the Fourier Transform Works

A function in the time domain is translated by the Fourier transform into a function in the frequency domain, where it can be analyzed for its frequency content. This translation occurs because the Fourier transform expands the original function in terms of orthonormal basis functions of sine and cosine waves of infinite duration. The Fourier coefficients of the transformed function represent the contribution of each sine and cosine wave at each frequency.

The Fourier transform works under the assumption that the original time-domain function is periodic in nature. As a result, the Fourier transform has difficulty with functions that have transient components, that is, components which are localized in time. This is especially apparent when a signal has sharp transitions. Another problem is that the Fourier transform of a signal does not convey any information pertaining to translation of the signal in time. Applications that use the Fourier transform often work around the first problem by windowing the input data so that the sampled values converge to 0 at the endpoints. Attempts to solve the second problem, such as the development of the short-time Fourier transform, have met with marginal success.

In recent years, new families of orthonormal basis functions have been discovered that lead to transforms which overcome the problems of the Fourier transform. These basis functions are called "wavelets", and unlike the sine and cosine wave of the Fourier transform, they need not have infinite duration. They can be non-zero for only a small range of the wavelet function. This "compact support" allows the wavelet transform to translate a time-domain function into a representation that is localized not only in frequency (like the Fourier transform) but in time as well. This ability has brought forth new developments in the fields of signal analysis, image processing, and data compression.

This article provides a basic understanding of wavelets and their use in representing signals through the wavelet transform. A description and implementation of a computationally efficient fast wavelet transform is also presented. It does for the wavelet transform what the Fast Fourier Transform (FFT) does for the Discrete Fourier Transform (DFT).

---

[1] Originally appeared in Dr. Dobb's Journal, April, 1992. Copyright ©1992, M&T Publishing Inc., Miller Freeman Publishing, Inc.; preprint created by permission. Corrections made by the author.

[2] Mac is the owner of Mac A. Cody Associates, Richardson, TX. He has published Tsunami Plus, a wavelet software library written in C. He can be at contacted by phone at 214.437.5761 and on the Internet at mcody@dfw.net.

(a)

$$\phi(t) = \sum_{k \in Z} a_k \phi(2t - k)$$

(b)

$$\psi(t) = \sum_{k \in Z} (-1^k) a_{k+1} \phi(2t + k)$$

(c)

$$\phi_{(j,k)}(t) = 2^{j/2} \phi(2^j t - k); j, k \in Z$$

$$\psi_{(j,k)}(t) = 2^{j/2} \psi(2^j t - k); j, k \in Z$$

(d)

$$\sum_{k \in Z} a_{2k} = 1 \quad \sum_{k \in Z} a_{2k+1} = 1$$

$$\sum_{k \in Z} a_k a_{k+2l} = 0 \;\; for \; l \neq 0 \quad \sum_{k \in Z} \bar{a}_k a_k = 2$$

Example 1: (a) The scaling function (also called the "basic dilation equation" or fundamental recursion); (b) the basic (or primary wavelet) function, where Z is the set of integers and the a_k are the wavelet coefficients; (c) a class of orthonormal basis functions where the parameter j controls the dilation or compression of the function in time scale as well as in amplitude. The parameter k controls the translation of the function in time. The set of basis functions formed by $\Phi(t)$ and $\Psi(t)$ is a system of scaled and translated wavelets; (d) the set of wavelet conditions which must be satisfied before a set of a_k can qualify to be called wavelet coefficients.

(a)

$$g(t) = \sum_{l=-\infty}^{\infty} c(l)\phi_l(t) + \sum_{j=0}^{\infty} \sum_{k=-\infty}^{\infty} d(j,k)\psi_{j,k}(t)$$

(b)

$$c(l) = <\phi_l \mid g> = \int g(t)\phi_l(t)dt$$

$$d(j,k) = <\psi_{j,k} \mid g> = \int g(t)\psi_{j,k}(t)dt$$

(c)

$$g(t) = \sum_{l \in Z} c(l)\phi_l(t) + \sum_{j=0}^{J-1} \sum_{k \in Z} d(j,k)\psi_{j,k}(t)$$

Example 2: (a) When a wavelet system is created, it can be used to expand a function $g(t)$ in terms of the basis functions; (b) the coefficients of (a) calculated by inner products; (c) if the wavelet system has compact support and an upper limit is placed upon the degree of dilation j, then the expansion equation takes this form.

$$a_{-2} = [(1 + \cos\alpha + \sin\alpha) - (1 - \cos\beta - \sin\beta) + 2\sin\beta \cdot \cos\alpha]/4$$

$$a_{-1} = [(1 - \cos\alpha + \sin\alpha) - (1 + \cos\beta - \sin\beta) - 2\sin\beta \cdot \cos\alpha]/4$$

$$a_0 = [1 + \cos(\alpha - \beta) + \sin(\alpha - \beta)]/2$$

$$a_1 = [1 + \cos(\alpha - \beta) - \sin(\alpha - \beta)]/2$$

$$a_2 = 1 - a_{-2} - a_0 \quad a_3 = 1 - a_{-1} - a_1, \ where - \pi \leq \alpha, \beta < \pi$$

Table 1: This set of functions generates a parameter space for the real-valued wavelet system with two to six coefficients. A parameter space is a map pf solutions to a set of functions on one or more variables or parameters. These functions were developed by David Pollen of Aware Inc.

Meeting the Wave Head On

Before implementing a fast wavelet transform, a brief explanation of wavelets and the wavelet transform is in order. There are two fundamental equations upon which wavelet calculations are based: the scaling function (also called the "basic dilation equation" or "fundamental recursion"), see Example 1(a), and the basic (or "primary wavelet") function, see Example 1(b), where Z is the set of integers and the a_k are the wavelet coefficients. Both functions are two-scale difference equations. They are the prototypes of a class of orthonormal basis functions of the form shown in Example 1(c), where the parameter j controls the dilation or compression of the function in time scale and amplitude. The parameter k controls the translation of the function in time. The set of basis functions formed by $\phi(t)$ and $\psi(t)$ is a system of scaled and translated wavelets.

Now, not just any set of a_k can be used as wavelet coefficients. Certain wavelet conditions must be satisfied before a set of a_k to qualify as wavelet coefficients. These conditions are shown in Example 1(d) where $\bar{a}_k$ is the conjugate of a_k (a_k can be complex).

Wavelet systems can, as a result, be either real or complex- valued. Most references have used real-valued wavelet systems. Wavelet systems may or may not have compact support. (As mentioned earlier, wavelets have compact support if, and only if, they have a finite number of non-zero coefficients.) It is compact support that allows wavelets to localize in both time and frequency, so we will be dealing only with wavelets of this type only.

Several techniques have been used to create wavelet systems. These include cubic splines, complex exponentials, and parameter space constructions. The wavelet coefficient generation routine in Listing Two uses formulas to generate a parameter space for the general, real-valued wavelet system with two to six coefficients. These parameter space formulas were developed by David Pollen of Aware Inc. and are listed in Table 1. The parameter space modeled by the formulas include the wavelet systems introduced by Ingrid Daubechies of Bell Laboratories and the original wavelet basis, the Haar basis.

Once a wavelet system is created, it can be used to expand a function $g(t)$ in terms of the basis functions shown in Example 2(a), with the coefficients calculated by inner products, as seen in Example 2(b). If the wavelet system has compact support and an upper limit is placed upon the degree of dilation j, then the expansion equation becomes that shown in Example 2(c).

The expansion coefficients $c(l)$ represent the approximation of the original signal $g(t)$ with a resolution of one point per every 2^J points of the original signal. The expansion coefficients $d(j,k)$ represent the details of the original signal at different levels of resolution. These coefficients completely and uniquely describe the original signal and can be used in a way similar to the Fourier transform. The wavelet transform, then, is the process of determining the values of $c(l)$ and $d(j,k)$ for a given $g(t)$ and wavelet system.

Digital Filters, Trees, and Fast Transforms

The Fourier transform was made practical to implement on computers by the development of the Fast Fourier Transform (FFT) by J. W. Cooley and J. W. Tukey in the mid-sixties. (Actually, it appears that Cooley and Tukey rediscovered the FFT. An unpublished treatise, written by Carl Friedrich Gauss in the early 1800s, describes an algorithm similar to the one developed by Cooley and Tukey. Gauss' algorithm appears to predate Fourier's 1807 work on harmonic analysis! You can draw your own conculsions.) The FFT eliminated redundancies that exist in the Discrete Fourier Transform. While a DFT of length N requires N^2 multiplications and additions, the FFT requires only $N \log_2 N$ such operations.

The expansion equation naturally leads to a recursive algorithm for the wavelet transform given certain assumptions. First, the function $g(t)$ is taken as a sequence of discrete points, $y_{J,p}$, sampled a 2^J points per unit interval. These points can be viewed as the inner products of $\phi_{J,p}$ and $g(t)$. That is, the sample points are an approximation, or $c(l)$ coefficients, of the continuous function $g(t)$. This allows $c(l)$ and $d(j,k)$ terms to be calculated by direct convolution of $g(t)$ samples with the coefficients a_k.

Daubechies has discovered that the wavelet transform can be implemented with a specially designed pair of Finite Impulse Response (FIR) filters called a "Quadrature Mirror Filter" (QMF) pair. FIR filters and QMFs come from the field of digital signal processing. A FIR filter performs the dot product (or sum of products) between the filter coefficients and the discrete samples in the tapped delay line of the filter. The act of passing a set of discrete samples, representing a signal, through a FIR filter is a discrete convolution of the signal with the filter's coefficients; see Figure 1.

QMF filters are used in digital speech anaylsis and spectral decomposition. Briefly, QMFs are distinctive because the frequency responses of the two FIR filters separate the high frequency and low frequency components of the input signal. The dividing point is usually half way between 0 Hertz (DC) and half the data Sampling rate (the Nyquist frequency).

The outputs of the QMF filter pair are decimated (or desampled) by a factor of two, that is, every other output sample of the filter is kept, the others are discarded. The low-frequency (low-pass) filter output is fed into another identical QMF filter pair. This operation can be repeated recursively as a tree or pyramid algorithm (Figure 2), yielding a group of signals that divides the spectrum of the original signal into octave bands with successively coarser measurements in time as the width of each spectral band narrows and decreases in frequency; see Figure 3.

Mallat has shown that the tree or pyramid algorithm can be applied to the wavelet transform by using the wavelet coefficients as the filter coefficients of the QMF filter pairs. The same wavelet coefficients are used in both the low-pass and high-pass (actually, band-pass) filters. The low-pass filter coefficients are associated with the a_k of the scaling function Φ. The output of each low-pass filter is the $c(l)$, or approximation components, of the original signal for that level of the tree. The high-pass filter is associated with the a_k of the wavelet function Ψ (Note the alternating sign change). The output of each high-pass filter is the $d(j,k)$, or detail components, of the original signal at resolution 2^j. The $c(l)$ of the previous level are used to generate the new $c(l)$ and $d(j,k)$ for the next level of the tree. Decmation by two corresponds to the multiresolutional nature (the j parameter) of the scaling and wavelet functions; see Figure 4.

The reverse fast wavelet transform essentially performs the operations associated with the forward fast wavelet transform in the opposite direction. The expansion coefficients are combined to reconstruct the original signal. The same a_k coefficients are used as in the forward transform, but in reversed order. The process works down the branches of the tree combining the approximation and detail signals into approximation signals with higher levels of detail; see Figure 5. Instead of decimation, the signals are interpolated: 0s are placed between each approximation and detail sample, and the signals are then passed through the low-pass and high-pass filters. The intermediate 0 values are replaced by "estimates" derived from the convolutions. The filters' outputs are then summed to form the approximation coefficients for the next higher level of resolution; see Figure 6. The final set of approximation coefficients at the tree's top level in the reverse transform is a reconstruction of the original signal data points.

The fast wavelet transform is actually more computationally efficient than the Fast Fourier Transform. As mentioned previously, an FFT of length N (where N is an integral power of 2) takes on the order of $N \log_2 N$ operations. A fast wavelet transform of length N requires approximately operations – the best possible. The FIR filters are linear processing elements. In the tree algorithm, each consecutive level performs only half as many operations as its predecessor.

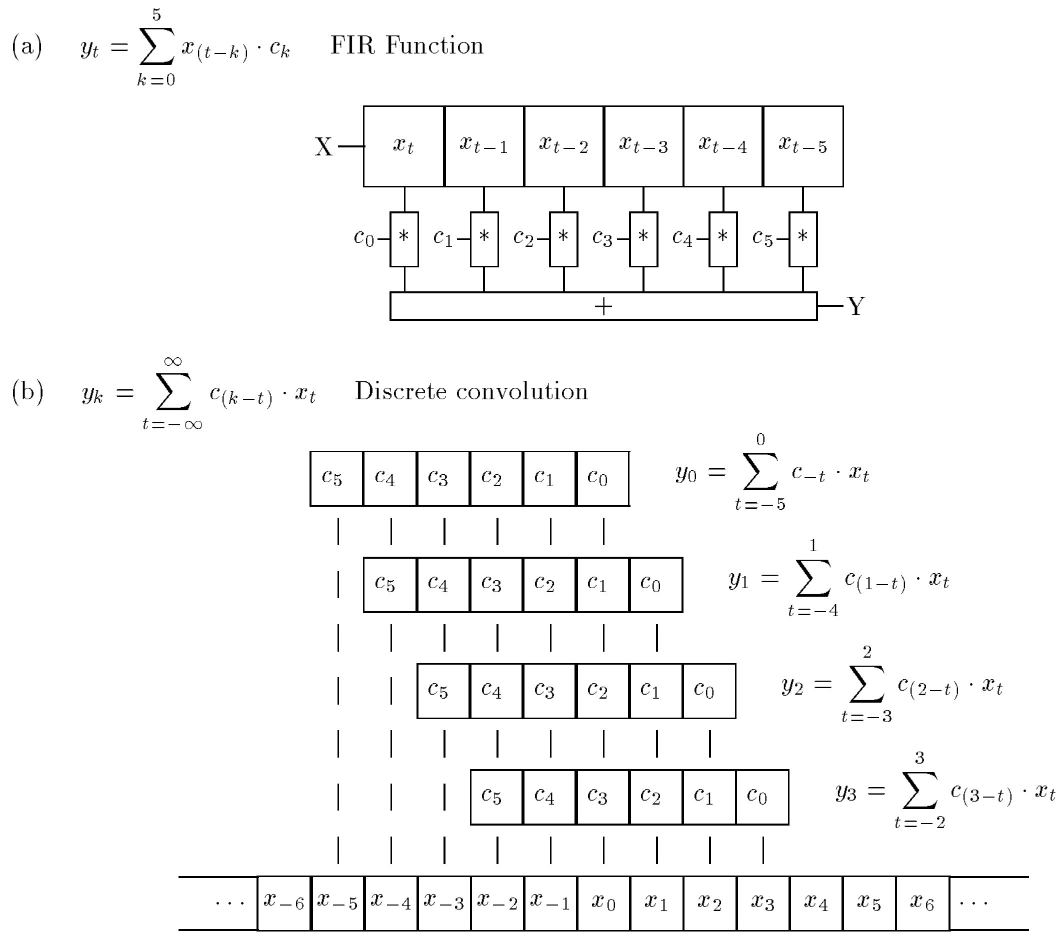

Figure 1: The Finite Impulse Response (FIR) filter (a) creates a new output y_t, each time the input sequence is shifted and a dot product is calculated. The accumulated dot products can be viewed as a piece-wise convolution of the FIR filter coefficients with the input data stream (b).

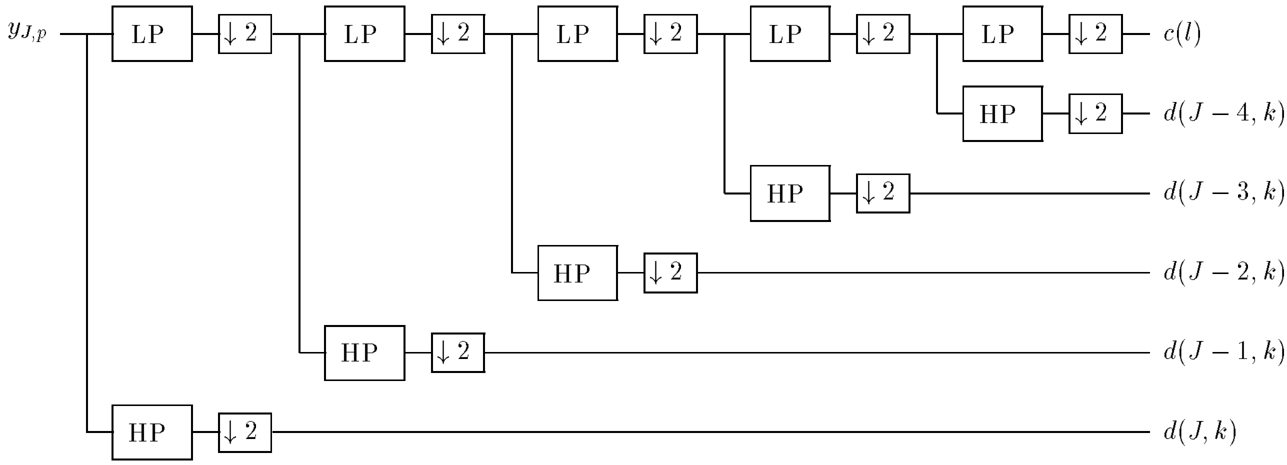

Figure 2: The tree or pyramid algorithm of the forward fast wavelet transform contains repeated convolutions of the input data $y_{J,p}$, with the wavelet coefficients Low-Pass (LP) and High-Pass (HP) filters. Each level of "filtering" is followed by a down-sampling or decimation by a factor of two. The end result is the wavelet transform coefficients, $c(l)$ amd $d(j,k)$.

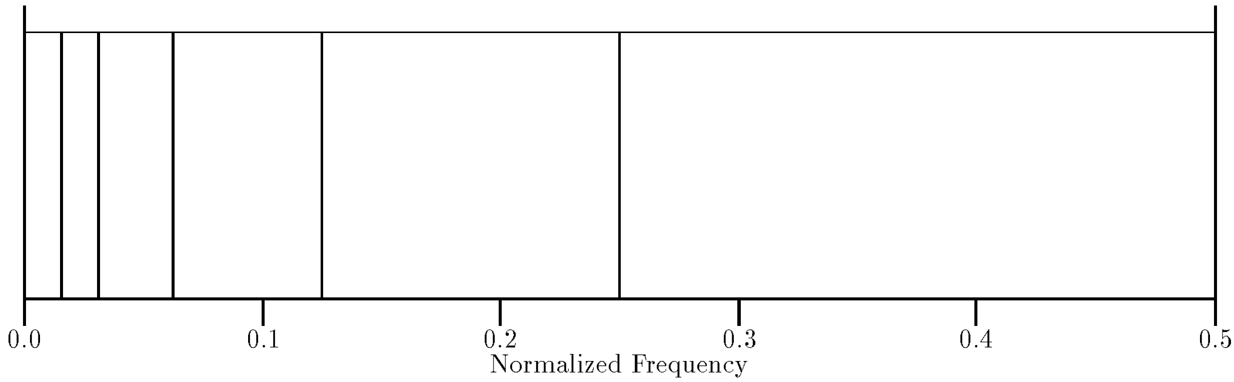

Figure 3: The forward fast wavelet pyramid algorithm (Figure 2) divides the spectrum into octave bands (shown as the normalized frequency between 0 Hz and half the data sampling rate, or Nyquist frequency). The low-pass band at the far left represents the frequency range covered by the approximation coefficients, $c(l)$, of the transform. The other five frequency bands represent the detail coefficients, $d(j,k)$, of the transform. Each band covers half the frequency range and half the time resolution of the next-higher band.

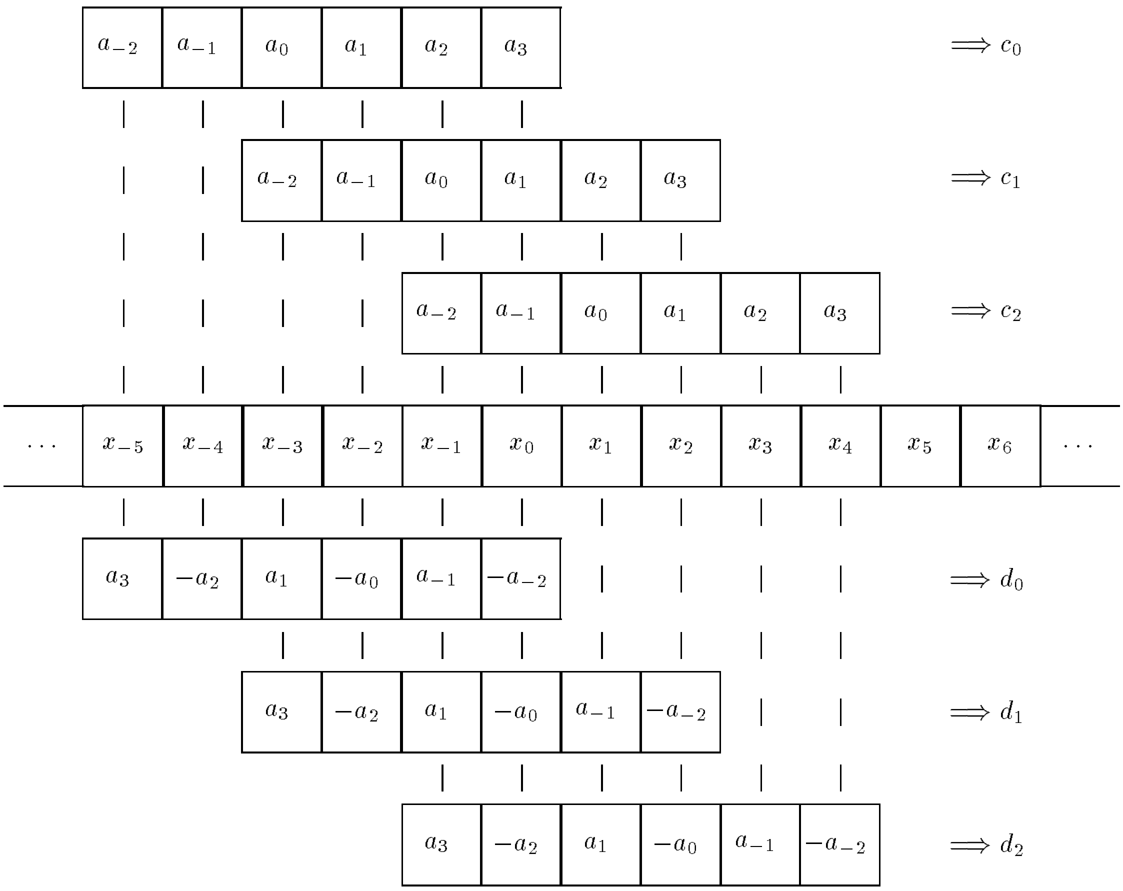

Figure 4: The convolution and decimation operations of the forward fast wavelet transform are done by performing the dot product between a group of samples x and the high-pass and low-pass wavelet filter coefficients. The filter coefficients are then shifted by two samples. The data rate of the approximation (c) and the detail (d) transform coefficients is half that of the x input samples.

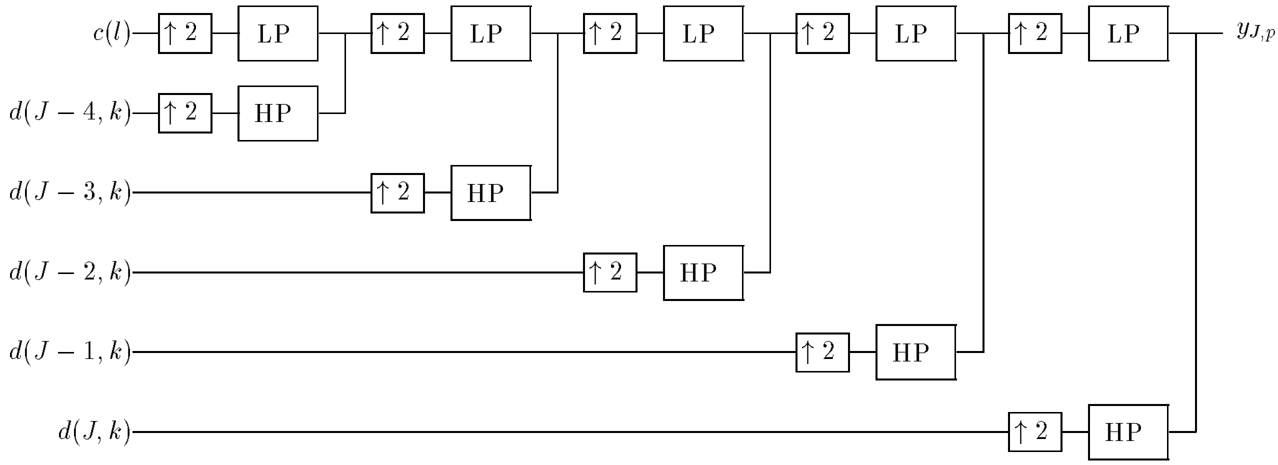

Figure 5: The tree or pyramid algorithm of the inverse fast wavelet transform reverses the operations of the forward fast wavelet transform. During the convolution, the input coefficients are up-sampled or interpolated by a factor of two. The wavelet coefficients' order of listing is reversed during the convolutions. Thw outputs of the high-pass or low-pass filters are then summed. The wavelet coefficients, $c(l)$ amd $d(j,k)$, are reconstructed to form the original sampled input $y_{J,p}$.

Catch a Wave ...

Wavelet filter coefficients are stored in arrays of type double, six elements in length. Unused array elements are set to 0. Storage for the sampled data signal is also an array of type double; see Figure 7(a). The only constraint on the length of the array is that the number of elements must be an integer multiple of the number of sample points per unit interval, 2^J. (J, once again, is the number of levels in the wavelet transform to be calculated.) An additional five storage locations are allocated to allow for convolution overrun. Transform coefficient storage consists of a dynamically created array of arrays of type double; see Figure 7(b). The pairs of arrays store the approximation and detail coefficients for each given level. The length of each pair of arrays is set according to the length of the array containing the sampled data signal and the number of levels in the transform. If the length of the input data array is $N+5$, then the topmost pair of arrays has a length of $N/2+5$ data elements, the next pair has a length of $N/4+5$ data elements, and so on.

Listings One and Two, page 100, provide a set of C functions for implementing of the forward and inverse fast wavelet transforms, the generation of wavelet filter coefficients, and data structure management. Headers for the functions are provided in Listing Three; see page 101. The functions are written in Turbo C 2.0 and conform to Standard C conventions. The code should be portable to most compilers and host systems with few problems.

The functions in Listing One provide utilities for the management of the storage structure for wavelet transform coefficients. *BuildTreeStorage* uses *calloc* to create a dynamic storage structure (array of arrays) for wavelet coefficients and intermediate results. *DestroyTreeStorage* uses *free* to delete an existing wavelet coefficient storage structure. *TreeCopy* copies wavelet coefficients from one storage structure to another. Only the $c(l)$ (approximation) coefficients on the last level are copied along with all the $d(j,k)$ (detail) coefficients. *TreeZero* sets all elements of the transform coefficient storage structure to zero. *ZeroTreeDetail* sets a given number of levels of detail coefficients to 0. The routine starts with the top level of detail coefficients and works down the required number of levels.

WaveletCoeffs, see Listing Two, implements the formulas to generate a parameter space for the general, real-valued wavelet system with two to six coefficients. The formulas are listed in Table 1. Varying the values of α and β yields an infinite variety of wavelet coefficient sets (a_k). *MakeWaveletFilters* takes the a_k coefficients created by *WaveletCoeffs* and creates the "low-pass" and "high-pass" filters for either the forward or the inverse fast wavelet transform.

To make the implementation of the forward and inverse fast wavelet transforms easier to comprehend,

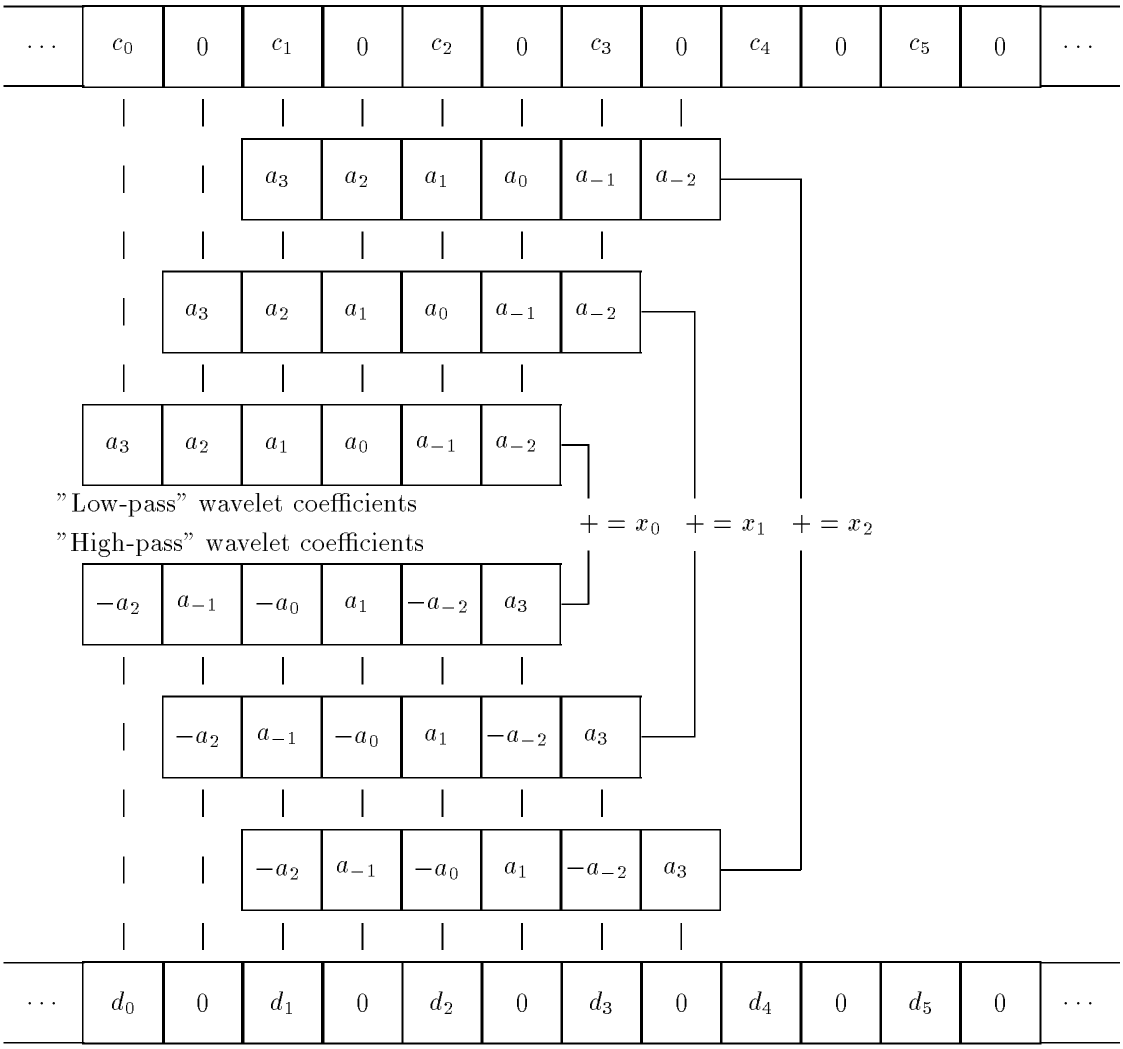

Figure 6: The interpolation and convolution operations of the inverse fast wavelet transform are accomplished by first inserting $0s$ between the transform coefficients. The dot product between a group of detail transform coefficients (d) and the high-pass wavelet coefficients are then summed with the dot products between a group of approximation transform coefficients (c) and the low-pass wavelet coefficients for that level. This yields the next, more-detailed level of approximation (or the original $y_{J,p}$ samples at the end of the tree) at double the data rate of the previous level of transform coefficients.

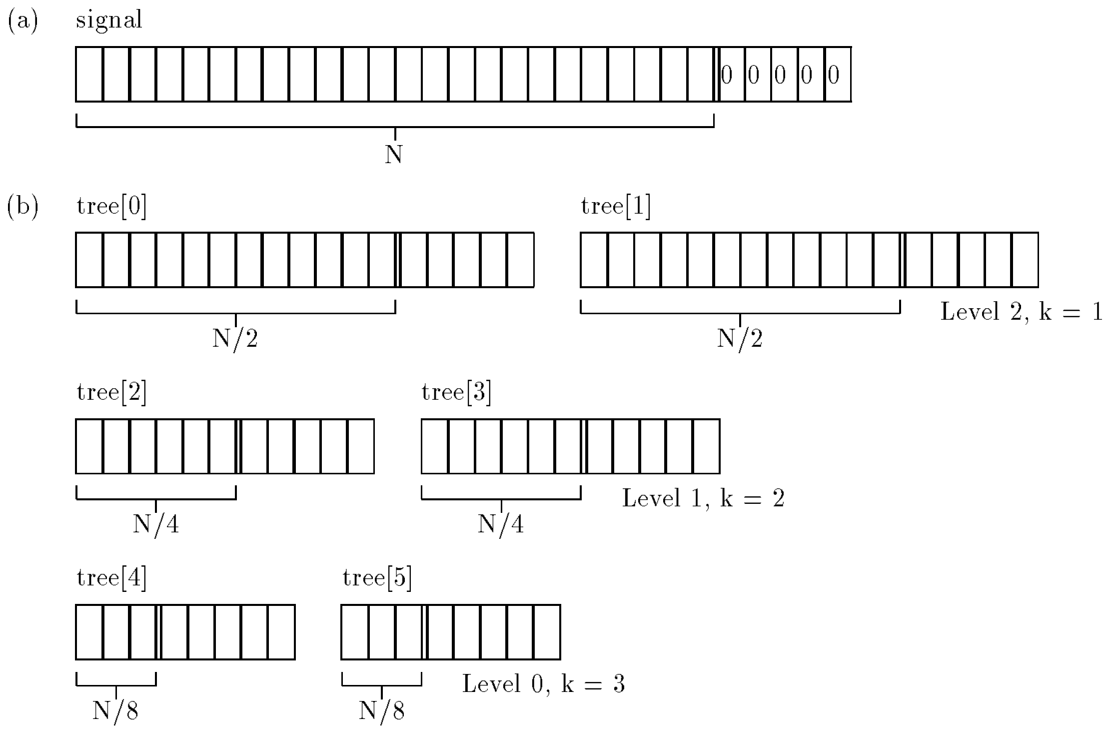

Figure 7: The signal storage structure (a) is an array of type double. The length of the array is $N+5$, where N is the length of the signal. (The additional storage is for convolution overlap.) The wavelet coefficient storage structure (b) is an array of double arrays, as created by *BuildTreeStorage*. Each pair of arrays stores the coefficients for the level of approximation and detail for each level of the transform algorithm. The length of each pair of arrays is $5+N/(2^k)$, where k is the depth of the array pair in tree. For this example, the intermediate signal approximations are stored in array tree[0] and tree[2]. The final approximation wavelet transform coefficients $[c(l)]$ are stored in array tree[4]. The detail wavelet transform coefficients $[d(j,k)]$ are stored in arrays tree[1], tree[3], amd tree[5].

the algorithms have been coded into several simple routines. These routines represent different levels of the algorithms' functionality.

DotP performs the dot product for the forward fast wavelet transform. Each coefficient of the wavelet filter is used in calculating a dot product. *ConvolveDec2* implements the convolution and decimation for the transform. The dot products of the wavelet filter coefficients and the sample data in the input array are calculated via calls to *DotP*. As shown in Figure 4, there is a shift of two samples between each dot product, thus accomplishing the decimation. The dot products become the transform coefficients of the branch of the next level. Special test conditions are taken for data samples at the beginning and end of the input array.

DecomposeBranches decomposes a level of the forward fast wavelet transform. The tree's current approximation level is transformed into two branches containing the detail and approximation decimations for the next level. This is accomplished with two calls to *ConvolveDec2*. One call passes the low-pass wavelet filter coefficients to generate the approximation coefficients for the next lower transform level. The second call passes the high-pass wavelet filter coefficients to generate the detail coefficients for the next lower transform level.

WaveletDecomposition is the driver routine for the forward fast wavelet transform algorithm. The code performs the necessary number of decompostions for the desired number of levels via calls to *DecomposeBranches*. For the first pass of the loop, *InData* is the sampled input signal. For subsequent loops, the approximation coefficients of the previously calculated level are used.

DotpEven and *DotpOdd* use the even- and odd-numbered filter coefficients to calculate dot products for the inverse fast wavelet transform. *ConvolveInt2* implements the convolution, interpolation, and summation operation for the inverse fast wavelet transform. The dot products of the wavelet filter coefficients and groups of samples in the input array are calculated via calls to *DotpEven* and *DotpOdd*. Alternating application of *DotpEven* and *DotpOdd* on the same subset of data points emulates the interpolation of the data with 0s inserted between each data point, as shown in Figure 6. Special test conditions are taken for data samples at the beginning and end of the array.

ReconstructBranches performs reconstruction of an inverse wavelet transform level. *ConvolveInt2* is called twice to convolve and sum the approximation and detail branches to form the next, more-detailed level of approximation coefficients. The first call of *ConvolveInt2* convolves the low-pass wavelet filter with the level's approximation branch. The second call of convolves the high-pass wavelet filter with the level's detail branch of the level and sums the results of both convolutions. *WaveletReconstruction* is the driver routine for inverse fast wavelet transform. It performs the necessary number of reconstructions for the number of levels of wavelet coefficients using *ReconstructBranches*.

Making Waves ...

To experiment with the wavelet transform, use the WAVEDEMO program, which employs the functions in Listings One and Two. WAVEDEMO is written in Turbo C 2.0 and uses Borland's BGI graphics to provide support for CGA, EGA, VGA, and Hercules graphics adapters. The display shows the original signal and the expansion coefficients at decreasing resolutions. An alternate display shows the expansion coefficients and the reconstructed signal. The program can perform up to six levels of coefficient expansion. The screen acts as a window that can be scrolled over the displayed coefficients. Another display shows the signal data and expansion coefficients in numerical format. A menu interface allows the user to generate wavelet coefficients; decompose and reconstruct signals; view and modify the expansion coefficients; and store and retrieve signal data and expansion coefficients from disk files. Because of its size, the program is available electronically; see page 3[3]. The electronic version includes the WAVEDEMO executable and source, sample input data and wavelet coefficient files, and documentation.

Wave Goodbye!

The wavelet transform allows for the simultaneous analysis of time and frequency in signals. An infinite variety of wavelets enables the user to tailor that analysis to extract this information of interest. An efficient

[3] The listings, source code, and programs described in this article are available via anonymous ftp from ftp.mv.com in the /pub/ddj/1992.04 directory. The file to download is 1992-apr.arc.

implemetation for computers has been made possible through the fast wavelet transform algorithm. The software in this article now makes this tool available to the masses.

Special thanks is given to Mr. Karl Jagler of Aware Inc. (Cambridge, Mass.), for giving me the insights and understanding into implementing the fast wavelet transform. His time and patience were at least as important in helping me understand wavelets as the reference articles and algorithm examples he sent me.

Bibliography

Elliot, Douglas F. ed. *Handbook of Digital Signal Processing: Engineering Applications.* San Deigo, Calif.: Academic Press, 1987.

Mallat, Stephane G. *A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. IEEE Transactions on Pattern Analysis and Machine Intelligence* (July 1989).

Resnikoff, H. L. "Foundations of Arithmeticum Analysis: Compactly Supported Wavelets and the Wavelet Group." *Aware Report No. AD890507.1.*

Resnikoff, H. L. and C. S. Burrus. "Relationships Between the Fourier Transform and the Wavelet Transform." *Aware Report No. AD900609.*

Strang, Gilbert. "Wavelets and Dilation Equations: A Brief Introduction." *SIAM Review (December 1989).*

The Ultra Wave Explorer User's Manual. Cambridge, Mass.: Aware, Inc., 1989.