15

Wavelet-Based Compression

15.1 Overview

In this chapter we introduce the concept of wavelets and describe how to use wavelet-based decompositions in compression schemes. We begin with an introduction to wavelets and multiresolution analysis and then describe how we can implement a wavelet decomposition using filters. We then examine the implementations of several wavelet-based compression schemes.

15.2 Introduction

In the previous two chapters we looked at a number of ways to decompose a signal. In this chapter we look at another approach to decompose a signal that has become increasingly popular in recent years: the use of wavelets. Wavelets are being used in a number of different applications. Depending on the application, different aspects of wavelets can be emphasized. As our particular application is compression, we will emphasize those aspects of wavelets that are important in the design of compression algorithms. You should be aware that there is much more to wavelets than is presented in this chapter. At the end of the chapter we suggest options if you want to delve more deeply into this subject.

The practical implementation of wavelet compression schemes is very similar to that of subband coding schemes. As in the case of subband coding, we decompose the signal (analysis) using filter banks. The outputs of the filter banks are downsampled, quantized, and encoded. The decoder decodes the coded representations, upsamples, and recomposes the signal using a synthesis filter bank.

In the next several sections we will briefly examine the construction of wavelets and describe how we can obtain a decomposition of a signal using multiresolution analysis. We will then describe some of the currently popular schemes for image compression. If you are

primarily interested at this time in implementation of wavelet-based compression schemes, you should skip the next few sections and go directly to Section 15.5.

In the last two chapters we have described several ways of decomposing signals. Why do we need another one? To answer this question, let's begin with our standard tool for analysis, the Fourier transform. Given a function $f(t)$, we can find the Fourier transform $F(\omega)$ as

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{j\omega t}dt.$$

Integration is an averaging operation; therefore, the analysis we obtain, using the Fourier transform, is in some sense an "average" analysis, where the averaging interval is all of time. Thus, by looking at a particular Fourier transform, we can say, for example, that there is a large component of frequency 10 kHz in a signal, but we cannot tell when in time this component occurred. Another way of saying this is that Fourier analysis provides excellent localization in frequency and none in time. The converse is true for the time function $f(t)$, which provides exact information about the value of the function at each instant of time but does not directly provide spectral information. It should be noted that both $f(t)$ and $F(\omega)$ represent the same function, and all the information is present in each representation. However, each representation makes different kinds of information easily accessible.

If we have a very nonstationary signal, like the one shown in Figure 15.1, we would like to know not only the frequency components but when in time the particular frequency components occurred. One way to obtain this information is via the *short-term Fourier transform* (STFT). With the STFT, we break the time signal $f(t)$ into pieces of length T and apply Fourier analysis to each piece. This way we can say, for example, that a component at 10 kHz occurred in the third piece—that is, between time $2T$ and time $3T$. Thus, we obtain an analysis that is a function of both time and frequency. If we simply chopped the function into pieces, we could get distortion in the form of boundary effects (see Problem 1). In order to reduce the boundary effects, we *window* each piece before we take the Fourier transform. If the window shape is given by $g(t)$, the STFT is formally given by

$$F(\omega, \tau) = \int_{-\infty}^{\infty} f(t)g^*(t-\tau)e^{j\omega t}dt. \tag{15.1}$$

If the window function $g(t)$ is a Gaussian, the STFT is called the *Gabor transform.*

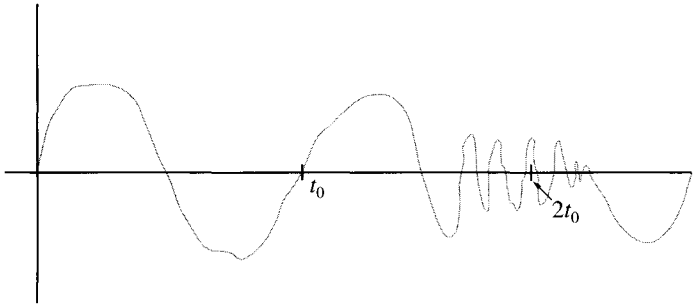

FIGURE 15. 1 A nonstationary signal.

The problem with the STFT is the fixed window size. Consider Figure 15.1. In order to obtain the low-pass component at the beginning of the function, the window size should be at least t_0 so that the window will contain at least one cycle of the low-frequency component. However, a window size of t_0 or greater means that we will not be able to accurately localize the high-frequency spurt. A large window in the time domain corresponds to a narrow filter in the frequency domain, which is what we want for the low-frequency components—and what we do not want for the high-frequency components. This dilemma is formalized in the uncertainty principle, which states that for a given window $g(t)$, the product of the time spread σ_t^2 and the frequency spread σ_ω^2 is lower bounded by $\sqrt{1/2}$, where

$$\sigma_t^2 = \frac{\int t^2 |g(t)|^2 \, dt}{\int |g(t)|^2 \, dt} \tag{15.2}$$

$$\sigma_\omega^2 = \frac{\int \omega^2 |G(\omega)|^2 \, d\omega}{\int |G(\omega)|^2 \, d\omega}. \tag{15.3}$$

Thus, if we wish to have finer resolution in time, that is, reduce σ_t^2, we end up with an increase in σ_ω^2, or a lower resolution in the frequency domain. How do we get around this problem?

Let's take a look at the discrete STFT in terms of basis expansion, and for the moment, let's look at just one interval:

$$F(m, 0) = \int_{-\infty}^{\infty} f(t) g^*(t) e^{-jm\omega_0 t} dt. \tag{15.4}$$

The basis functions are $g(t)$, $g(t)e^{j\omega_o t}$, $g(t)e^{j2\omega_o t}$, and so on. The first three basis functions are shown in Figure 15.2. We can see that we have a window with constant size, and within this window, we have sinusoids with an increasing number of cycles. Let's conjure up a different set of functions in which the number of cycles is constant, but the size of the window keeps changing, as shown in Figure 15.3. Notice that although the number of

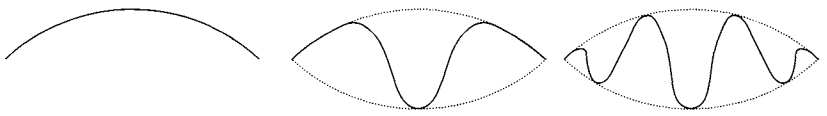

FIGURE 15. 2 The first three STFT basis functions for the first time interval.

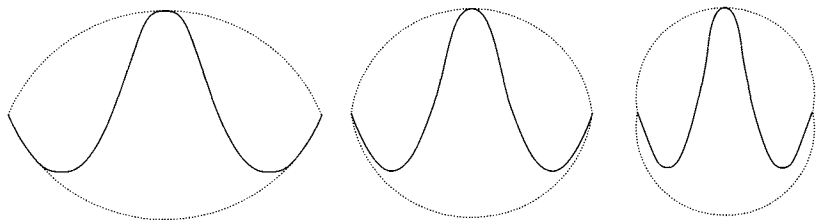

FIGURE 15. 3 Three wavelet basis functions.

cycles of the sinusoid in each window is the same, as the size of the window gets smaller, these cycles occur in a smaller time interval; that is, the frequency of the sinusoid increases. Furthermore, the lower frequency functions cover a longer time interval, while the higher frequency functions cover a shorter time interval, thus avoiding the problem that we had with the STFT. If we can write our function in terms of these functions and their translates, we have a representation that gives us time and frequency localization and can provide high frequency resolution at low frequencies (longer time window) and high time resolution at high frequencies (shorter time window). This, crudely speaking, is the basic idea behind wavelets.

In the following section we will formalize the concept of wavelets. Then we will discuss how to get from a wavelet basis set to an implementation. If you wish to move directly to implementation issues, you should skip to Section 15.5.

15.3 Wavelets

In the example at the end of the previous section, we started out with a single function. All other functions were obtained by changing the size of the function or *scaling* and translating this single function. This function is called the *mother wavelet*. Mathematically, we can scale a function $f(t)$ by replacing t with t/a, where the parameter a governs the amount of scaling. For example, consider the function

$$f(t) = \begin{cases} \cos(\pi t) & -1 \le t \le 1 \\ 0 & \text{otherwise.} \end{cases}$$

We have plotted this function in Figure 15.4. To scale this function by 0.5, we replace t by $t/0.5$:

$$f\left(\frac{t}{0.5}\right) = \begin{cases} \cos(\pi \frac{t}{0.5}) & -1 \le \frac{t}{0.5} \le 1 \\ 0 & \text{otherwise} \end{cases}$$

$$= \begin{cases} \cos(2\pi t) & -\frac{1}{2} \le t \le \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

We have plotted the scaled function in Figure 15.5. If we define the norm of a function $f(t)$ by

$$\|f(t)\|^2 = \int_{-\infty}^{\infty} f^2(t)dt$$

scaling obviously changes the norm of the function:

$$\left\|f\left(\frac{t}{a}\right)\right\|^2 = \int_{-\infty}^{\infty} f^2\left(\frac{t}{a}\right)dt$$

$$= a\int_{-\infty}^{\infty} f^2(x)dx$$

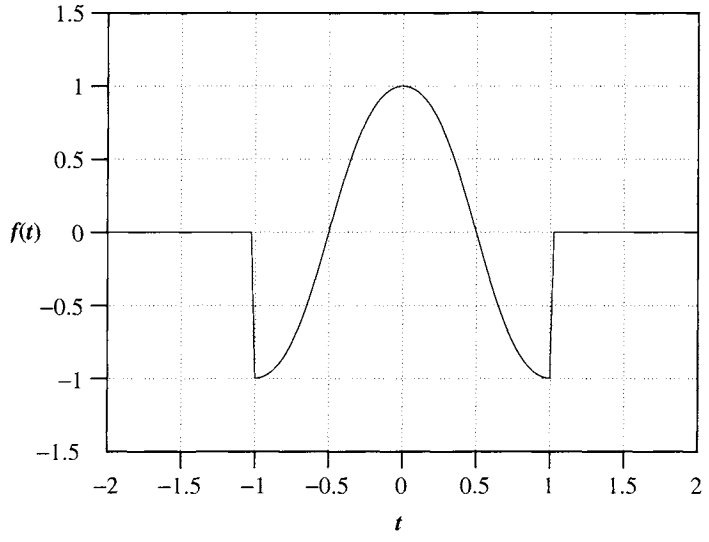

FIGURE 15. 4 **A function $f(t)$.**

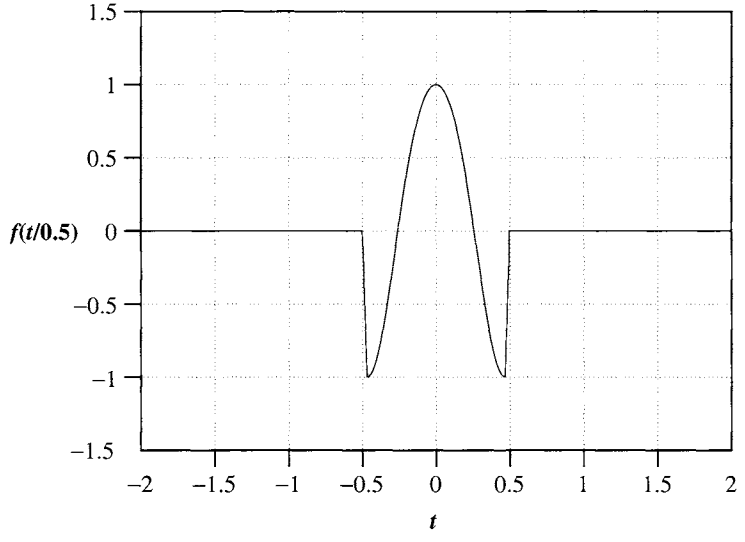

FIGURE 15. 5 **The function $f(\frac{t}{0.5})$.**

where we have used the substitution $x = t/a$. Thus,

$$\left\| f\left(\frac{t}{a}\right) \right\|^2 = a \left\| f(t) \right\|^2 .$$

If we want the scaled function to have the same norm as the original function, we need to multiply it by $1/\sqrt{a}$.

Mathematically, we can represent the translation of a function to the right or left by an amount b by replacing t by $t-b$ or $t+b$. For example, if we want to translate the scaled function shown in Figure 15.5 by one, we have

$$f\left(\frac{t-1}{0.5}\right) = \begin{cases} \cos(2\pi(t-1)) & -\frac{1}{2} \le t-1 \le \frac{1}{2} \\ 0 & \text{otherwise} \end{cases}$$

$$= \begin{cases} \cos(2\pi(t-1)) & \frac{1}{2} \le t \le \frac{3}{2} \\ 0 & \text{otherwise.} \end{cases}$$

The scaled and translated function is shown in Figure 15.6. Thus, given a *mother wavelet* $\psi(t)$, the remaining functions are obtained as

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right) \tag{15.5}$$

with Fourier transforms

$$\begin{aligned} \Psi(\omega) &= \mathcal{F}[\psi(t)] \\ \Psi_{a,b}(\omega) &= \mathcal{F}[\psi_{a,b}(t)]. \end{aligned} \tag{15.6}$$

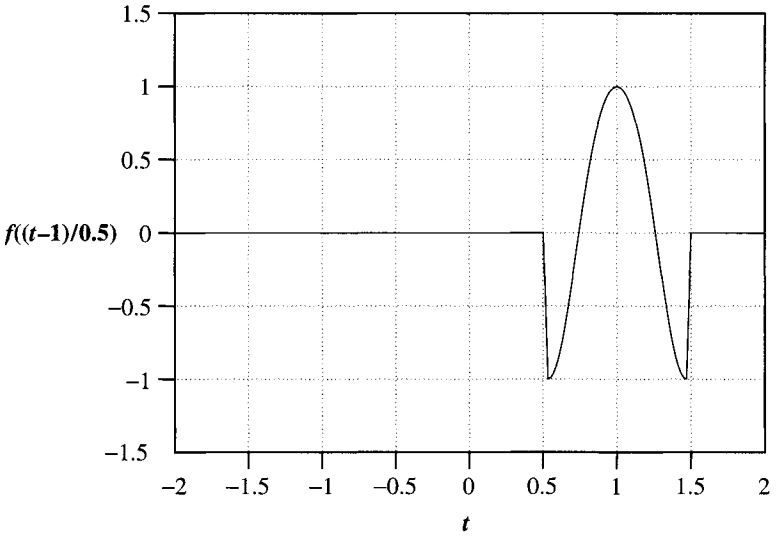

FIGURE 15.6 **A scaled and translated function.**

Our expansion using coefficients with respect to these functions is obtained from the inner product of $f(t)$ with the wavelet functions:

$$w_{a,b} = \langle \psi_{a,b}(t), f(t) \rangle = \int_{-\infty}^{\infty} \psi_{a,b}(t) f(t) dt. \tag{15.7}$$

We can recover the function $f(t)$ from the $w_{a,b}$ by

$$f(t) = \frac{1}{C_\psi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} w_{a,b} \psi_{a,b}(t) \frac{da db}{a^2} \tag{15.8}$$

where

$$C_\psi = \int_0^{\infty} \frac{|\Psi(\omega)|^2}{\omega} d\omega. \tag{15.9}$$

For integral (15.8) to exist, we need C_ψ to be finite. For C_ψ to be finite, we need $\Psi(0) = 0$. Otherwise, we have a singularity in the integrand of (15.9). Note that $\Psi(0)$ is the average value of $\psi(t)$; therefore, a requirement on the mother wavelet is that it have zero mean. The condition that C_ψ be finite is often called the *admissibility condition*. We would also like the wavelets to have finite energy; that is, we want the wavelets to belong to the vector space L_2 (see Example 12.3.1). Using Parseval's relationship, we can write this requirement as

$$\int_{-\infty}^{\infty} |\Psi(\omega)|^2 d\omega < \infty.$$

For this to happen, $|\Psi(\omega)|^2$ has to decay as ω goes to infinity. These requirements mean that the energy in $\Psi(\omega)$ is concentrated in a narrow frequency band, which gives the wavelet its frequency localization capability.

If a and b are continuous, then $w_{a,b}$ is called the *continuous wavelet transform* (CWT). Just as with other transforms, we will be more interested in the discrete version of this transform. We first obtain a series representation where the basis functions are continuous functions of time with discrete scaling and translating parameters a and b. The discrete versions of the scaling and translating parameters have to be related to each other because if the scale is such that the basis functions are narrow, the translation step should be correspondingly small and vice versa. There are a number of ways we can choose these parameters. The most popular approach is to select a and b according to

$$a = a_0^{-m}, \qquad b = n b_0 a_0^{-m} \tag{15.10}$$

where m and n are integers, a_0 is selected to be 2, and b_0 has a value of 1. This gives us the wavelet set

$$\psi_{m,n}(t) = a_0^{m/2} \psi(a_0^m t - n b_0), \qquad m, n \in Z. \tag{15.11}$$

For $a_0 = 2$ and $b_0 = 1$, we have

$$\psi_{m,n}(t) = 2^{m/2} \psi(2^m t - n). \tag{15.12}$$

(Note that these are the most commonly used choices, but they are not the only choices.) If this set is *complete*, then $\{\psi_{m,n}(t)\}$ are called *affine* wavelets. The wavelet coefficients are given by

$$w_{m,n} = \langle f(t), \psi_{m,n}(t) \rangle \tag{15.13}$$

$$= a_0^{m/2} \int f(t)\psi(a_0^m t - nb_0)dt. \tag{15.14}$$

The function $f(t)$ can be reconstructed from the wavelet coefficients by

$$f(t) = \sum_m \sum_n w_{m,n}\psi_{m,n}(t). \tag{15.15}$$

Wavelets come in many shapes. We will look at some of the more popular ones later in this chapter. One of the simplest wavelets is the Haar wavelet, which we will use to explore the various aspects of wavelets. The Haar wavelet is given by

$$\psi(t) = \begin{cases} 1 & 0 \le t < \frac{1}{2} \\ -1 & \frac{1}{2} \le t < 1. \end{cases} \tag{15.16}$$

By translating and scaling this mother wavelet, we can synthesize a variety of functions.

This version of the transform, where $f(t)$ is a continuous function while the transform consists of discrete values, is a wavelet series analogous to the Fourier series. It is also called the *discrete time wavelet transform* (DTWT). We have moved from the continuous wavelet transform, where both the time function $f(t)$ and its transform $w_{a,b}$ were continuous functions of their arguments, to the wavelet series, where the time function is continuous but the time-scale wavelet representation is discrete. Given that in data compression we are generally dealing with sampled functions that are discrete in time, we would like both the time and frequency representations to be discrete. This is called the *discrete wavelet transform* (DWT). However, before we get to that, let's look into one additional concept—multiresolution analysis.

15.4 Multiresolution Analysis and the Scaling Function

The idea behind multiresolution analysis is fairly simple. Let's define a function $\phi(t)$ that we call a *scaling* function. We will later see that the scaling function is closely related to the mother wavelet. By taking linear combinations of the scaling function and its translates we can generate a large number of functions

$$f(t) = \sum_k a_k \phi(t-k). \tag{15.17}$$

The scaling function has the property that a function that can be represented by the scaling function can also be represented by the dilated versions of the scaling function.

For example, one of the simplest scaling functions is the Haar scaling function:

$$\phi(t) = \begin{cases} 1 & 0 \le t < 1 \\ 0 & \text{otherwise.} \end{cases} \tag{15.18}$$

Then $f(t)$ can be any piecewise continuous function that is constant in the interval $[k, k+1)$ for all k.

Let's define

$$\phi_k(t) = \phi(t-k). \tag{15.19}$$

The set of all functions that can be obtained using a linear combination of the set $\{\phi_k(t)\}$

$$f(t) = \sum_k a_k \phi_k(t) \tag{15.20}$$

is called the *span* of the set $\{\phi_k(t)\}$, or $\text{Span}\{\phi_k(t)\}$. If we now add all functions that are limits of sequences of functions in $\text{Span}\{\phi_k(t)\}$, this is referred to as the closure of $\text{Span}\{\phi_k(t)\}$ and denoted by $\overline{\text{Span}\{\phi_k(t)\}}$. Let's call this set V_0.

If we want to generate functions at a higher resolution, say, functions that are required to be constant over only half a unit interval, we can use a dilated version of the "mother" scaling function. In fact, we can obtain scaling functions at different resolutions in a manner similar to the procedure used for wavelets:

$$\phi_{j,k}(t) = 2^{j/2}\phi(2^j t - k). \tag{15.21}$$

The indexing scheme is the same as that used for wavelets, with the first index referring to the resolution while the second index denotes the translation. For the Haar example,

$$\phi_{1,0}(t) = \begin{cases} \sqrt{2} & 0 \le t < \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases} \tag{15.22}$$

We can use translates of $\phi_{1,0}(t)$ to represent all functions that are constant over intervals $[k/2, (k+1)/2)$ for all k. Notice that in general any function that can be represented by the translates of $\phi(t)$ can also be represented by a linear combination of translates of $\phi_{1,0}(t)$. The converse, however, is not true. Defining

$$V_1 = \overline{\text{Span}\{\phi_{1,k}(t)\}} \tag{15.23}$$

we can see that $V_0 \subset V_1$. Similarly, we can show that $V_1 \subset V_2$, and so on.

Example 15.4.1:

Consider the function shown in Figure 15.7. We can approximate this function using translates of the Haar scaling function $\phi(t)$. The approximation is shown in Figure 15.8a. If we call this approximation $\phi_f^{(0)}(t)$, then

$$\phi_f^{(0)}(t) = \sum_k c_{0,k}\phi_k(t) \tag{15.24}$$

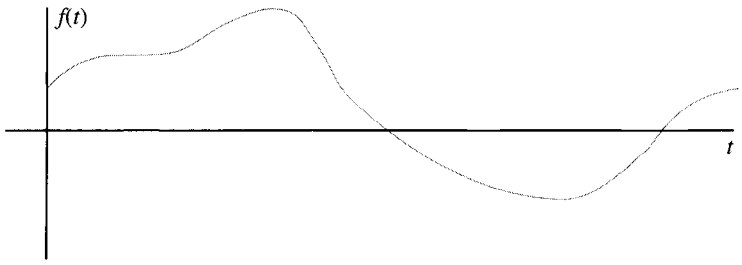

FIGURE 15.7 **A sample function.**

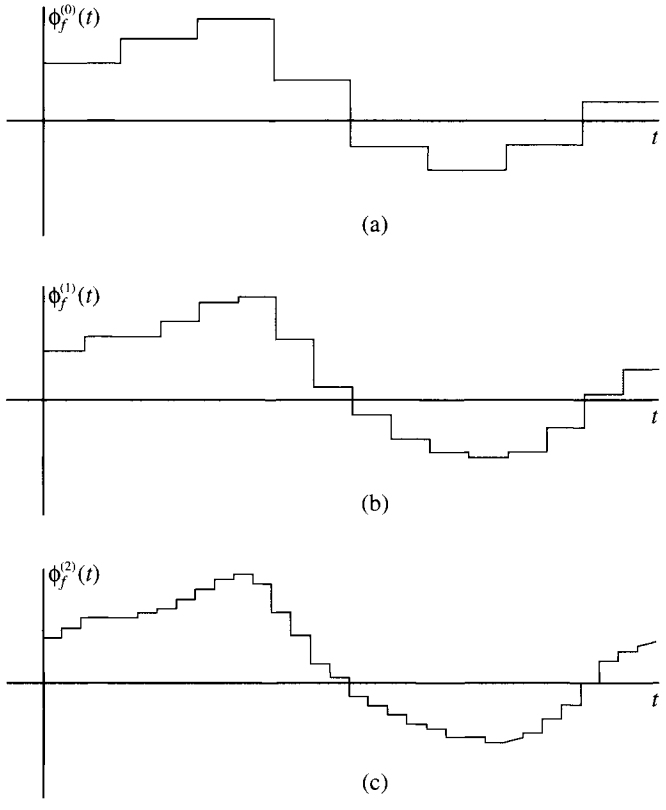

FIGURE 15.8 **Approximations of the function shown Figure 15.7.**

where

$$c_{0,k} = \int_k^{k+1} f(t)\phi_k(t)dt. \tag{15.25}$$

We can obtain a more refined approximation, or an approximation at a higher resolution, $\phi_f^{(1)}(t)$, shown in Figure 15.8b, if we use the set $\{\phi_{1,k}(t)\}$:

$$\phi_f^{(1)}(t) = \sum_k c_{1,k}\phi_{1,k}(t). \tag{15.26}$$

Notice that we need twice as many coefficients at this resolution compared to the previous resolution. The coefficients at the two resolutions are related by

$$c_{0,k} = \frac{1}{\sqrt{2}}(c_{1,2k} + c_{1,2k+1}). \tag{15.27}$$

Continuing in this manner (Figure 15.8c), we can get higher and higher resolution approximations of $f(t)$ with

$$\phi_f^{(m)}(t) = \sum_k c_{m,k}\phi_{m,k}(t). \tag{15.28}$$

Recall that, according to the Nyquist rule, if the highest frequency component of a signal is at f_0 Hz, we need $2f_0$ samples per second to accurately represent it. Therefore, we could obtain an accurate representation of $f(t)$ using the set of translates $\{\phi_{j,k}(t)\}$, where $2^{-j} < \frac{1}{2f_0}$.

As

$$c_{j,k} = 2^{j/2}\int_{\frac{k}{2^j}}^{\frac{k+1}{2^j}} f(t)dt, \tag{15.29}$$

by the mean value theorem of calculus, $c_{j,k}$ is equal to a sample value of $f(t)$ in the interval $[k2^{-j}, (k+1)2^{-j})$. Therefore, the function $\phi_f^{(j)}(t)$ would represent more than $2f_0$ samples per second of $f(t)$. ♦

We said earlier that a scaling function has the property that any function that can be represented exactly by an expansion at some resolution j can also be represented by dilations of the scaling function at resolution $j+1$. In particular, this means that the scaling function itself can be represented by its dilations at a higher resolution:

$$\phi(t) = \sum_k h_k\phi_{1,k}(t). \tag{15.30}$$

Substituting $\phi_{1,k}(t) = \sqrt{2}\phi(2t-k)$, we obtain the *multiresolution analysis* (MRA) equation:

$$\phi(t) = \sum_k h_k\sqrt{2}\phi(2t-k). \tag{15.31}$$

This equation will be of great importance to us when we begin looking at ways of implementing the wavelet transform.

Example 15.4.2:

Consider the Haar scaling function. Picking

$$h_0 = h_1 = \frac{1}{\sqrt{2}}$$

and

$$h_k = 0 \qquad \text{for } k > 1$$

satisfies the recursion equation. ♦

Example 15.4.3:

Consider the triangle scaling function shown in Figure 15.9. For this function

$$h_0 = \frac{1}{2\sqrt{2}}, \quad h_1 = \frac{1}{\sqrt{2}}, \quad h_2 = \frac{1}{2\sqrt{2}}$$

satisfies the recursion equation.

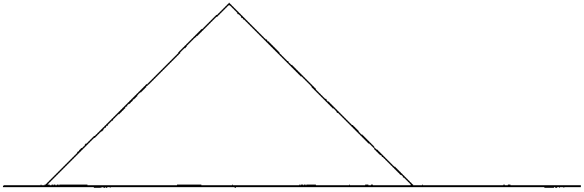

FIGURE 15. 9 Triangular scaling function. ♦

While both the Haar scaling function and the triangle scaling functions are valid scaling functions, there is an important difference between the two. The Haar function is orthogonal to its translates; that is,

$$\int \phi(t)\phi(t-m)dt = \delta_m.$$

This is obviously not true of the triangle function. In this chapter we will be principally concerned with scaling functions that are orthogonal because they give rise to orthonormal transforms that, as we have previously seen, are very useful in compression.

How about the Haar wavelet? Can it be used as a scaling function? Some reflection will show that we cannot obtain the Haar wavelet from a linear combination of its dilated versions.

So, where do wavelets come into the picture? Let's continue with our example using the Haar scaling function. Let us assume for the moment that there is a function $g(t)$ that can be exactly represented by $\phi_g^{(1)}(t)$; that is, $g(t)$ is a function in the set V_1. We can decompose

$\phi_g^{(1)}(t)$ into the sum of a lower-resolution version of itself, namely, $\phi_g^{(0)}(t)$, and the difference $\phi_g^{(1)}(t) - \phi_g^{(0)}(t)$. Let's examine this difference over an arbitrary unit interval $[k, k+1)$:

$$\phi_g^{(1)}(t) - \phi_g^{(0)}(t) = \begin{cases} c_{0,k} - \sqrt{2}c_{1,2k} & k \le t < k + \frac{1}{2} \\ c_{0,k} - \sqrt{2}c_{1,2k+1} & k + \frac{1}{2} \le t < k + 1. \end{cases} \tag{15.32}$$

Substituting for $c_{0,k}$ from (15.27), we obtain

$$\phi_g^{(1)}(t) - \phi_g^{(0)}(t) = \begin{cases} -\frac{1}{\sqrt{2}}c_{1,2k} + \frac{1}{\sqrt{2}}c_{1,2k+1} & k \le t < k + \frac{1}{2} \\ \frac{1}{\sqrt{2}}c_{1,2k} - \frac{1}{\sqrt{2}}c_{1,2k+1} & k + \frac{1}{2} \le t < k + 1. \end{cases} \tag{15.33}$$

Defining

$$d_{0,k} = -\frac{1}{\sqrt{2}}c_{1,2k} + \frac{1}{\sqrt{2}}c_{1,2k+1}$$

over the arbitrary interval $[k, k+1)$,

$$\phi_g^{(1)}(t) - \phi_g^{(0)}(t) = d_{0,k}\psi_{0,k}(t) \tag{15.34}$$

where

$$\psi_{0,k}(t) = \begin{cases} 1 & k \le t < k + \frac{1}{2} \\ -1 & k + \frac{1}{2} \le t < k + 1. \end{cases} \tag{15.35}$$

But this is simply the kth translate of the Haar wavelet. Thus, for this particular case the function can be represented as the sum of a scaling function and a wavelet at the same resolution:

$$\phi_g^{(1)}(t) = \sum_k c_{0,k}\phi_{0,k}(t) + \sum_k d_{0,k}\psi_{0,k}(t). \tag{15.36}$$

In fact, we can show that this decomposition is not limited to this particular example. A function in V_1 can be decomposed into a function in V_0—that is, a function that is a linear combination of the scaling function at resolution 0, and a function that is a linear combination of translates of a mother wavelet. Denoting the set of functions that can be obtained by a linear combination of the translates of the mother wavelet as W_0, we can write this symbolically as

$$V_1 = V_0 \oplus W_0. \tag{15.37}$$

In other words, any function in V_1 can be represented using functions in V_0 and W_0.

Obviously, once a scaling function is selected, the choice of the wavelet function cannot be arbitrary. The wavelet that generates the set W_0 and the scaling function that generates the sets V_0 and V_1 are intrinsically related. In fact, from (15.37), $W_0 \subset V_1$, and therefore any function in W_0 can be represented by a linear combination of $\{\phi_{1,k}\}$. In particular, we can write the mother wavelet $\psi(t)$ as

$$\psi(t) = \sum_k w_k\phi_{1,k}(t) \tag{15.38}$$

or

$$\psi(t) = \sum_k w_k \sqrt{2}\phi(2t-k). \tag{15.39}$$

This is the counterpart of the multiresolution analysis equation for the wavelet function and will be of primary importance in the implementation of the decomposition.

All of this development has been for a function in V_1. What if the function can only be accurately represented at resolution $j+1$? If we define W_j as the closure of the span of $\psi_{j,k}(t)$, we can show that

$$V_{j+1} = V_j \oplus W_j. \tag{15.40}$$

But, as j is arbitrary,

$$V_j = V_{j-1} \oplus W_{j-1} \tag{15.41}$$

and

$$V_{j+1} = V_{j-1} \oplus W_{j-1} \oplus W_j. \tag{15.42}$$

Continuing in this manner, we can see that for any $k \leq j$

$$V_{j+1} = V_k \oplus W_k \oplus W_{k+1} \oplus \cdots \oplus W_j. \tag{15.43}$$

In other words, if we have a function that belongs to V_{j+1} (i.e., that can be exactly represented by the scaling function at resolution $j+1$), we can decompose it into a sum of functions starting with a lower-resolution approximation followed by a sequence of functions generated by dilations of the wavelet that represent the leftover details. This is very much like what we did in subband coding. A major difference is that, while the subband decomposition is in terms of sines and cosines, the decomposition in this case can use a variety of scaling functions and wavelets. Thus, we can adapt the decomposition to the signal being decomposed by selecting the scaling function and wavelet.

15.5 Implementation Using Filters

One of the most popular approaches to implementing the decomposition discussed in the previous section is using a hierarchical filter structure similar to the one used in subband coding. In this section we will look at how to obtain the structure and the filter coefficients.

We start with the MRA equation

$$\phi(t) = \sum_k h_k \sqrt{2}\phi(2t-k). \tag{15.44}$$

Substituting $t = 2^j t - m$, we obtain the equation for an arbitrary dilation and translation:

$$\phi(2^j t - m) = \sum_k h_k \sqrt{2}\phi(2(2^j t - m) - k) \tag{15.45}$$

$$= \sum_k h_k \sqrt{2}\phi(2^{j+1} t - 2m - k) \tag{15.46}$$

$$= \sum_l h_{l-2m} \sqrt{2}\phi(2^{j+1} t - l) \tag{15.47}$$

where in the last equation we have used the substitution $l = 2m + k$. Suppose we have a function $f(t)$ that can be accurately represented at resolution $j+1$ by some scaling function $\phi(t)$. We assume that the scaling function and its dilations and translations form an orthonormal set. The coefficients c_{j+1} can be obtained by

$$c_{j+1,k} = \int f(t)\phi_{j+1,k}dt. \tag{15.48}$$

If we can represent $f(t)$ accurately at resolution $j+1$ with a linear combination of $\phi_{j+1,k}(t)$, then from the previous section we can decompose it into two functions: one in terms of $\phi_{j,k}(t)$ and one in terms of the jth dilation of the corresponding wavelet $\{\psi_{j,k}(t)\}$. The coefficients $c_{j,k}$ are given by

$$c_{j,k} = \int f(t)\phi_{j,k}(t)dt \tag{15.49}$$

$$= \int f(t)2^{\frac{j}{2}}\phi(2^j t - k)dt. \tag{15.50}$$

Substituting for $\phi(2^j t - k)$ from (15.47), we get

$$c_{j,l} = \int f(t)2^{\frac{j}{2}} \sum_l h_{l-2k}\sqrt{2}\phi(2^{j+1}t - l)dt. \tag{15.51}$$

Interchanging the order of summation and integration, we get

$$c_{j,l} = \sum_l h_{l-2k} \int f(t)2^{\frac{j}{2}}\sqrt{2}\phi(2^{j+1}t - l)dt. \tag{15.52}$$

But the integral is simply $c_{j+1,k}$. Therefore,

$$c_{j,k} = \sum_k h_{k-2m}c_{j+1,k}. \tag{15.53}$$

We have encountered this relationship before in the context of the Haar function. Equation (15.27) provides the relationship between coefficients of the Haar expansion at two resolution levels. In a more general setting, the coefficients $\{h_j\}$ provide a link between the coefficients $\{c_{j,k}\}$ at different resolutions. Thus, given the coefficients at resolution level $j+1$, we can obtain the coefficients at all other resolution levels. But how do we start the process? Recall that $f(t)$ can be accurately represented at resolution $j+1$. Therefore, we can replace $c_{j+1,k}$ by the samples of $f(t)$. Let's represent these samples by x_k. Then the coefficients of the low-resolution expansion are given by

$$c_{j,k} = \sum_k h_{k-2m}x_k. \tag{15.54}$$

In Chapter 12, we introduced the input-output relationship of a linear filter as

$$y_m = \sum_k h_k x_{m-k} = \sum_k h_{m-k}x_k. \tag{15.55}$$

Replacing m by $2m$, we get every other sample of the output

$$y_{2m} = \sum_k h_{2m-k}x_k. \tag{15.56}$$

Comparing (15.56) with (15.54), we can see that the coefficients of the low-resolution approximation are every other output of a linear filter whose impulse response is h_{-k}. Recall that $\{h_k\}$ are the coefficients that satisfy the MRA equation. Using the terminology of subband coding, the coefficients $c_{j,k}$ are the downsampled output of the linear filter with impulse response $\{h_{-k}\}$.

The detail portion of the representation is obtained in a similar manner. Again we start from the recursion relationship. This time we use the recursion relationship for the wavelet function as our starting point:

$$\psi(t) = \sum_k w_k \sqrt{2}\phi(2t-k). \tag{15.57}$$

Again substituting $t = 2^j t - m$ and using the same simplifications, we get

$$\psi(2^j t - m) = \sum_k w_{k-2m} \sqrt{2}\phi(2^{j+1}t - k). \tag{15.58}$$

Using the fact that the dilated and translated wavelets form an orthonormal basis, we can obtain the detail coefficients $d_{j,k}$ by

$$d_{j,k} = \int f(t)\psi_{j,k}(t)dt \tag{15.59}$$

$$= \int f(t)2^{\frac{j}{2}}\psi(2^j t - k)dt \tag{15.60}$$

$$= \int f(t)2^{\frac{j}{2}} \sum_l w_{l-2k}\sqrt{2}\phi(2^{j+1}t - l)dt \tag{15.61}$$

$$= \sum_l w_{l-2k} \int f(t)2^{\frac{j+1}{2}}\phi(2^{j+1}t - l)dt \tag{15.62}$$

$$= \sum_l w_{l-2k} c_{j+1,l}. \tag{15.63}$$

Thus, the detail coefficients are the decimated outputs of a filter with impulse response $\{w_{-k}\}$.

At this point we can use exactly the same arguments to further decompose the coefficients $\{c_j\}$.

In order to retrieve $\{c_{j+1,k}\}$ from $\{c_{j,k}\}$ and $\{d_{j,k}\}$, we upsample the lower resolution coefficients and filter, using filters with impulse response $\{h_k\}$ and $\{w_k\}$

$$c_{j+1,k} = \sum_l c_{j,l} b_{k-2l} \sum_l d_{j,l} w_{k-2l}.$$

15.5.1 Scaling and Wavelet Coefficients

In order to implement the wavelet decomposition, the coefficients $\{h_k\}$ and $\{w_k\}$ are of primary importance. In this section we look at some of the properties of these coefficients that will help us in finding different decompositions.

We start with the MRA equation. Integrating both sides of the equation over all t, we obtain

$$\int_{-\infty}^{\infty} \phi(t)dt = \int_{-\infty}^{\infty} \sum_k h_k \sqrt{2}\phi(2t-k)dt. \tag{15.64}$$

Interchanging the summation and integration on the right-hand side of the equation, we get

$$\int_{-\infty}^{\infty} \phi(t)dt = \sum_k h_k \sqrt{2} \int_{-\infty}^{\infty} \phi(2t-k)dt. \tag{15.65}$$

Substituting $x = 2t-k$ with $dx = 2dt$ in the right-hand side of the equation, we get

$$\int_{-\infty}^{\infty} \phi(t)dt = \sum_k h_k \sqrt{2} \int_{-\infty}^{\infty} \phi(x)\frac{1}{2}dx \tag{15.66}$$

$$= \sum_k h_k \frac{1}{\sqrt{2}} \int_{-\infty}^{\infty} \phi(x)dx. \tag{15.67}$$

Assuming that the average value of the scaling function is not zero, we can divide both sides by the integral and we get

$$\sum_k h_k = \sqrt{2}. \tag{15.68}$$

If we normalize the scaling function to have a magnitude of one, we can use the orthogonality condition on the scaling function to get another condition on $\{h_k\}$:

$$\int |\phi(t)|^2 dt = \int \sum_k h_k \sqrt{2}\phi(2t-k) \sum_m h_m \sqrt{2}\phi(2t-m)dt \tag{15.69}$$

$$= \sum_k \sum_m h_k h_m 2 \int \phi(2t-k)\phi(2t-m)dt \tag{15.70}$$

$$= \sum_k \sum_m h_k h_m \int \phi(x-k)\phi(x-m)dx \tag{15.71}$$

where in the last equation we have used the substitution $x = 2t$. The integral on the right-hand side is zero except when $k = m$. When $k = m$, the integral is unity and we obtain

$$\sum_k h_k^2 = 1. \tag{15.72}$$

We can actually get a more general property by using the orthogonality of the translates of the scaling function

$$\int \phi(t)\phi(t-m)dt = \delta_m. \tag{15.73}$$

Rewriting this using the MRA equation to substitute for $\phi(t)$ and $\phi(t-m)$, we obtain

$$\int \left[\sum_k h_k \sqrt{2}\phi(2t-k)\right]\left[\sum_l h_l \sqrt{2}\phi(2t-2m-l)\right]dt$$

$$= \sum_k \sum_l h_k h_l 2 \int \phi(2t-k)\phi(2t-2m-l)dt. \tag{15.74}$$

Substituting $x = 2t$, we get

$$\int \phi(t)\phi(t-m)dt = \sum_k \sum_l h_k h_l \int \phi(x-k)\phi(x-2m-l)dx \tag{15.75}$$

$$= \sum_k \sum_l h_k h_l \delta_{k-(2m+l)} \tag{15.76}$$

$$= \sum_k h_k h_{k-2m}. \tag{15.77}$$

Therefore, we have

$$\sum_k h_k h_{k-2m} = \delta_m \tag{15.78}$$

Notice that this is the same relationship we had to satisfy for perfect reconstruction in the previous chapter.

Using these relationships, we can generate scaling coefficients for filters of various lengths.

Example 15.5.1:

For $k = 2$, we have from (15.68) and (15.72)

$$h_0 + h_1 = \sqrt{2} \tag{15.79}$$

$$h_0^2 + h_1^2 = 1. \tag{15.80}$$

These equations are uniquely satisfied by

$$h_0 = h_1 = \frac{1}{\sqrt{2}},$$

which is the Haar scaling function. ♦

An orthogonal expansion does not exist for all lengths. In the following example, we consider the case of $k = 3$.

Example 15.5.2:

For $k = 3$, from the three conditions (15.68), (15.72), and (15.78), we have

$$h_0 + h_1 + h_2 = \sqrt{2} \tag{15.81}$$

$$h_0^2 + h_1^2 + h_2^2 = 1 \tag{15.82}$$

$$h_0 h_2 = 0 \tag{15.83}$$

The last condition can only be satisfied if $h_0 = 0$ or $h_2 = 0$. In either case we will be left with the two-coefficient filter for the Haar scaling function. ♦

In fact, we can see that for k odd, we will always end up with a condition that will force one of the coefficients to zero, thus leaving an even number of coefficients. When the

number of coefficients gets larger than the number of conditions, we end up with an infinite number of solutions.

Example 15.5.3:

Consider the case when $k = 4$. The three conditions give us the following three equations:

$$h_0 + h_1 + h_2 + h_3 = \sqrt{2} \tag{15.84}$$

$$h_0^2 + h_1^2 + h_2^2 + h_3^2 = 1 \tag{15.85}$$

$$h_0 h_2 + h_1 h_3 = 0 \tag{15.86}$$

We have three equations and four unknowns; that is, we have one degree of freedom. We can use this degree of freedom to impose further conditions on the solution. The solutions to these equations include the Daubechies four-tap solution:

$$h_0 = \frac{1+\sqrt{3}}{4\sqrt{2}}, \quad h_1 = \frac{3+\sqrt{3}}{4\sqrt{2}}, \quad h_2 = \frac{3-\sqrt{3}}{4\sqrt{2}}, \quad h_3 = \frac{1-\sqrt{3}}{4\sqrt{2}}.$$ ♦

Given the close relationship between the scaling function and the wavelet, it seems reasonable that we should be able to obtain the coefficients for the wavelet filter from the coefficients of the scaling filter. In fact, if the wavelet function is orthogonal to the scaling function at the same scale

$$\int \phi(t-k)\psi(t-m)dt = 0, \tag{15.87}$$

then

$$w_k = \pm(-1)^k h_{N-k} \tag{15.88}$$

and

$$\sum_k h_k w_{n-2k} = 0. \tag{15.89}$$

Furthermore,

$$\sum_k w_k = 0. \tag{15.90}$$

The proof of these relationships is somewhat involved [207].

15.5.2 Families of Wavelets

Let's move to the more practical aspects of compression using wavelets. We have said that there is an infinite number of possible wavelets. Which one is best depends on the application. In this section we list different wavelets and their corresponding filters. You are encouraged to experiment with these to find those best suited to your application.

The 4-tap, 12-tap, and 20-tap Daubechies filters are shown in Tables 15.1–15.3. The 6-tap, 12-tap, and 18-tap Coiflet filters are shown in Tables 15.4–15.6.

TABLE 15.1 Coefficients for the 4-tap Daubechies low-pass filter.

h_0	0.4829629131445341
h_1	0.8365163037378079
h_2	0.2241438680420134
h_3	−0.1294095225512604

TABLE 15.2 Coefficients for the 12-tap Daubechies low-pass filter.

h_0	0.111540743350
h_1	0.494623890398
h_2	0.751133908021
h_3	0.315250351709
h_4	−0.226264693965
h_5	−0.129766867567
h_6	0.097501605587
h_7	0.027522865530
h_8	−0.031582039318
h_9	0.000553842201
h_{10}	0.004777257511
h_{11}	−0.001077301085

TABLE 15.3 Coefficients for the 20-tap Daubechies low-pass filter.

h_0	0.026670057901
h_1	0.188176800078
h_2	0.527201188932
h_3	0.688459039454
h_4	0.281172343661
h_5	−0.249846424327
h_6	−0.195946274377
h_7	0.127369340336
h_8	0.093057364604
h_9	−0.071394147166
h_{10}	−0.029457536822
h_{11}	0.033212674059
h_{12}	0.003606553567
h_{13}	−0.010733175483
h_{14}	0.001395351747
h_{15}	0.001992405295
h_{16}	−0.000685856695
h_{17}	−0.000116466855
h_{18}	0.000093588670
h_{19}	−0.000013264203

TABLE 15.4 Coefficients for the 6-tap Coiflet low-pass filter.

h_0	−0.051429728471
h_1	0.238929728471
h_2	0.602859456942
h_3	0.272140543058
h_4	−0.051429972847
h_5	−0.011070271529

TABLE 15.5 Coefficients for the 12-tap Coiflet low-pass filter.

h_0	0.011587596739
h_1	−0.029320137980
h_2	−0.047639590310
h_3	0.273021046535
h_4	0.574682393857
h_5	0.294867193696
h_6	−0.054085607092
h_7	−0.042026480461
h_8	0.016744410163
h_9	0.003967883613
h_{10}	−0.001289203356
h_{11}	−0.000509505539

TABLE 15.6 Coefficients for the 18-tap Coiflet low-pass filter.

h_0	−0.002682418671
h_1	0.005503126709
h_2	0.016583560479
h_3	−0.046507764479
h_4	−0.043220763560
h_5	0.286503335274
h_6	0.561285256870
h_7	0.302983571773
h_8	−0.050770140755
h_9	−0.058196250762
h_{10}	0.024434094321
h_{11}	0.011229240962
h_{12}	−0.006369601011
h_{13}	−0.001820458916
h_{14}	0.000790205101
h_{15}	0.000329665174
h_{16}	−0.000050192775
h_{17}	−0.000024465734

15.6 Image Compression

One of the most popular applications of wavelets has been to image compression. The JPEG 2000 standard, which is designed to update and replace the current JPEG standard, will use wavelets instead of the DCT to perform decomposition of the image. During our discussion we have always referred to the signal to be decomposed as a one-dimensional signal; however, images are two-dimensional signals. There are two approaches to the subband decomposition of two-dimensional signals: using two-dimensional filters, or using separable transforms that can be implemented using one-dimensional filters on the rows first and then on the columns (or vice versa). Most approaches, including the JPEG 2000 verification model, use the second approach.

In Figure 15.10 we show how an image can be decomposed using subband decomposition. We begin with an $N \times M$ image. We filter each row and then downsample to obtain two $N \times \frac{M}{2}$ images. We then filter each column and subsample the filter output to obtain four $\frac{N}{2} \times \frac{M}{2}$ images. Of the four subimages, the one obtained by low-pass filtering the rows and columns is referred to as the LL image; the one obtained by low-pass filtering the rows and high-pass filtering the columns is referred to as the LH image; the one obtained by high-pass filtering the rows and low-pass filtering the columns is called the HL image; and the subimage obtained by high-pass filtering the rows and columns is referred to as the HH image. This decomposition is sometimes represented as shown in Figure 15.11. Each of the subimages obtained in this fashion can then be filtered and subsampled to obtain four more subimages. This process can be continued until the desired subband structure is obtained. Three popular structures are shown in Figure 15.12. In the structure in Figure 15.12a, the LL subimage has been decomposed after each decomposition into four more subimages, resulting in a total of 10 subimages. This is one of the more popular decompositions.

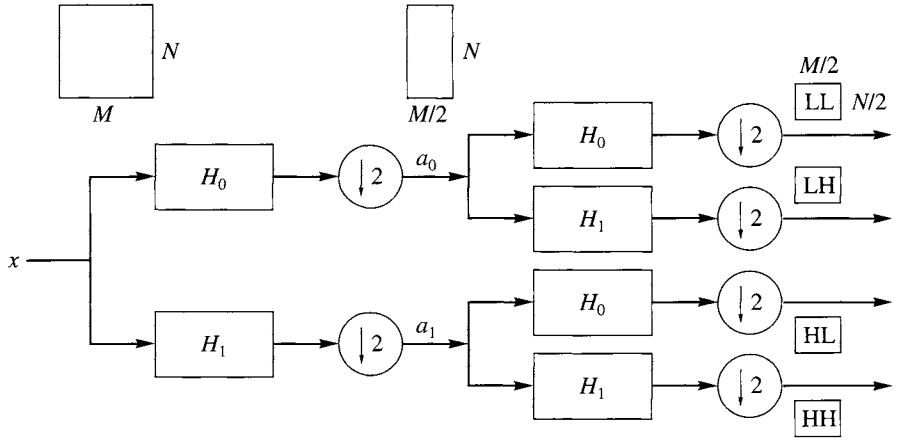

FIGURE 15.10 **Subband decomposition of an $N \times M$ image.**

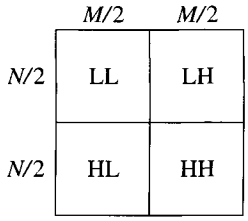

FIGURE 15. 11 First-level decomposition.

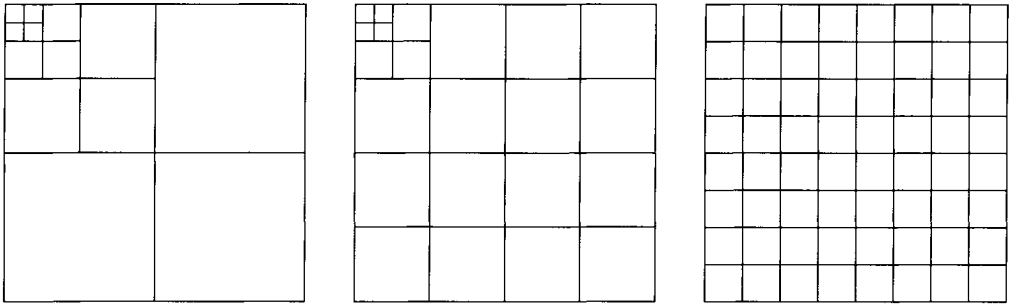

FIGURE 15. 12 Three popular subband structures.

Example 15.6.1:

Let's use the Daubechies wavelet filter to repeat what we did in Examples 14.12.2 and 14.12.3 using the Johnston and the Smith-Barnwell filters. If we use the 4-tap Daubechies filter, we obtain the decomposition shown in Figure 15.13. Notice that even though we are only using a 4-tap filter, we get results comparable to the 16-tap Johnston filter and the 8-tap Smith-Barnwell filters.

If we now encode this image at the rate of 0.5 bits per pixel, we get the reconstructed image shown in Figure 15.14. Notice that the quality is comparable to that obtained using filters requiring two or four times as much computation. ♦

In this example we used a simple scalar quantizer for quantization of the coefficients. However, if we use strategies that are motivated by the properties of the coefficients themselves, we can obtain significant performance improvements. In the next sections we examine two popular quantization strategies developed specifically for wavelets.

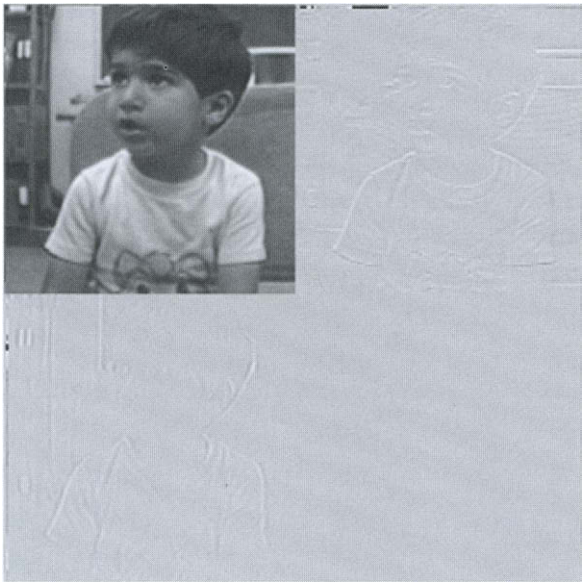

FIGURE 15. 13 **Decomposition of Sinan image using the four-tap Daubechies filter.**

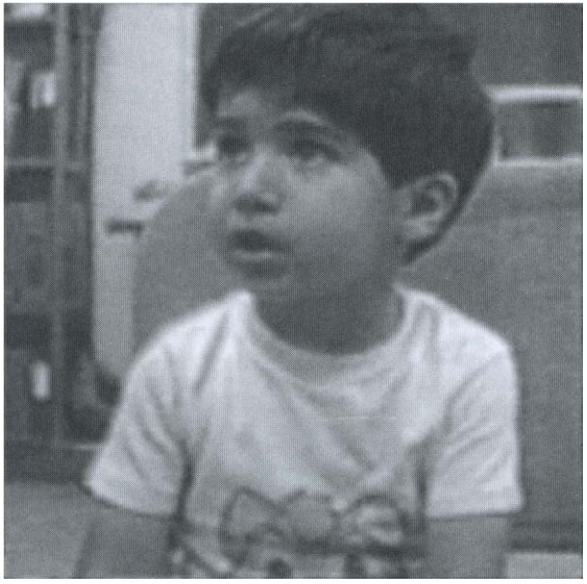

FIGURE 15. 14 **Reconstruction of Sinan image encoded using 0.5 bits per pixel and the four-tap Daubechies filter.**

15.7 Embedded Zerotree Coder

The embedded zerotree wavelet (EZW) coder was introduced by Shapiro [208]. It is a quantization and coding strategy that incorporates some characteristics of the wavelet decomposition. Just as the quantization and coding approach used in the JPEG standard, which were motivated by the characteristics of the coefficients, were superior to the generic zonal coding algorithms, the EZW approach and its descendants significantly outperform some of the generic approaches. The particular characteristic used by the EZW algorithm is that there are wavelet coefficients in different subbands that represent the same spatial location in the image. If the decomposition is such that the size of the different subbands is different (the first two decompositions in Figure 15.12), then a single coefficient in the smaller subband may represent the same spatial location as multiple coefficients in the other subbands.

In order to put our discussion on more solid ground, consider the 10-band decomposition shown in Figure 15.15. The coefficient a in the upper-left corner of band I represents the same spatial location as coefficients a_1 in band II, a_2 in band III, and a_3 in band IV. In turn, the coefficient a_1 represents the same spatial location as coefficients a_{11}, a_{12}, a_{13}, and a_{14} in band V. Each of these pixels represents the same spatial location as four pixels in band VIII, and so on. In fact, we can visualize the relationships of these coefficients in the form of a tree: The coefficient a forms the root of the tree with three descendants a_1, a_2, and a_3. The coefficient a_1 has descendants a_{11}, a_{12}, a_{13}, and a_{14}. The coefficient a_2 has descendants a_{21}, a_{22}, a_{23}, and a_{24}, and the coefficient a_3 has descendants a_{31}, a_{32}, a_{33}, and a_{34}. Each of these coefficients in turn has four descendants, making a total of 64 coefficients in this tree. A pictorial representation of the tree is shown in Figure 15.16.

Recall that when natural images are decomposed in this manner most of the energy is compacted into the lower bands. Thus, in many cases the coefficients closer to the root of the tree have higher magnitudes than coefficients further away from the root. This means that often if a coefficient has a magnitude less than a given threshold, all its descendants will have magnitudes less than that threshold. In a scalar quantizer, the outer levels of the quantizer correspond to larger magnitudes. Consider the 3-bit quantizer shown in Figure 15.17. If we determine that all coefficients arising from a particular root have magnitudes smaller than T_0 and we inform the decoder of this situation, then for all coefficients in that tree we need only use 2 bits per sample, while getting the same performance as we would have obtained using the 3-bit quantizer. If the binary coding scheme used in Figure 15.17 is used, in which the first bit is the sign bit and the next bit is the most significant bit of the magnitude, then the information that a set of coefficients has value less than T_0 is the same as saying that the most significant bit of the magnitude is 0. If there are N coefficients in the tree, this is a savings of N bits minus however many bits are needed to inform the decoder of this situation.

Before we describe the EZW algorithm, we need to introduce some terminology. Given a threshold T, if a given coefficient has a magnitude greater than T, it is called a *significant* coefficient at level T. If the magnitude of the coefficient is less than T (it is insignificant), and all its descendants have magnitudes less than T, then the coefficient is called a *zerotree root*. Finally, it might happen that the coefficient itself is less than T but some of its descendants have a value greater than T. Such a coefficient is called an *isolated zero*.

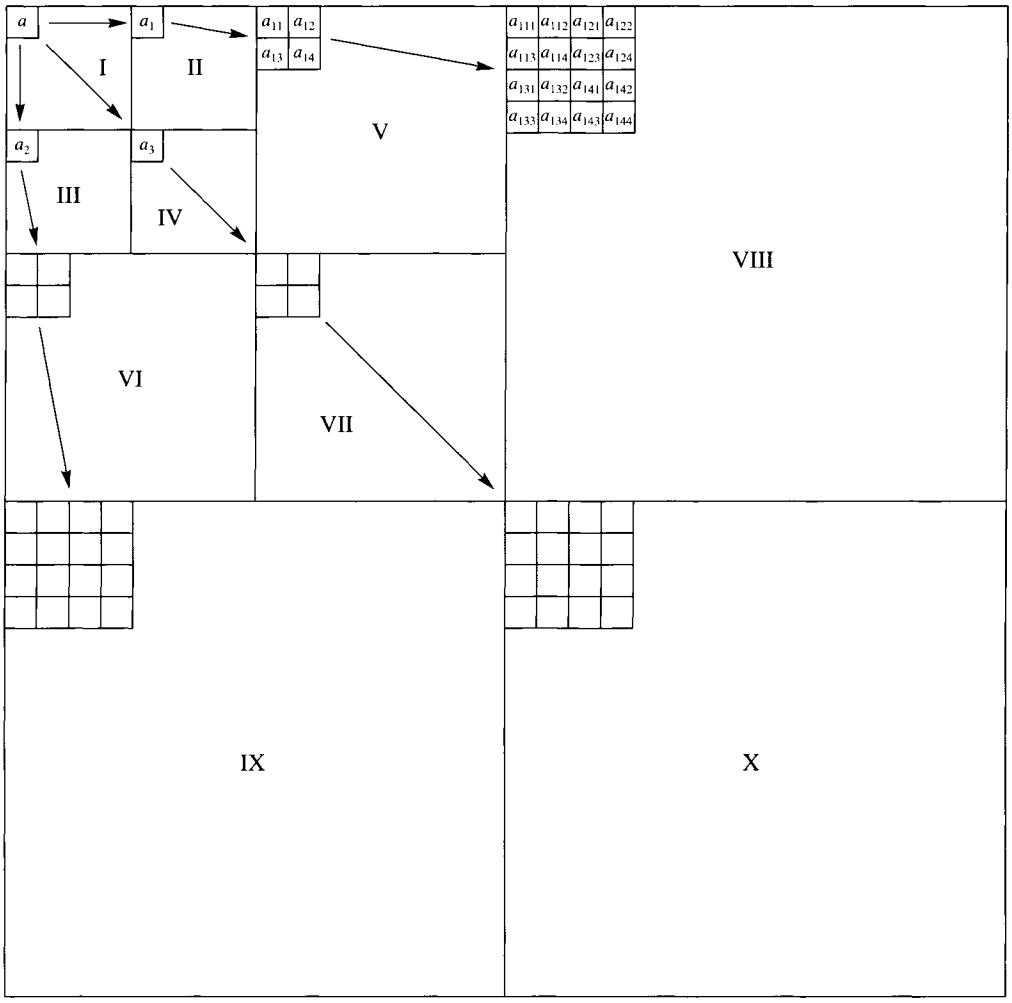

FIGURE 15.15 A 10-band wavelet decomposition.

The EZW algorithm is a multiple-pass algorithm, with each pass consisting of two steps: *significance map encoding* or the *dominant pass*, and *refinement* or the *subordinate pass*. If $c_{\max}$ is the value of the largest coefficient, the initial value of the threshold T_0 is given by

$$T_0 = 2^{\lfloor \log_2 c_{\max} \rfloor}. \tag{15.91}$$

This selection guarantees that the largest coefficient will lie in the interval $[T_0, 2T_0)$. In each pass, the threshold T_i is reduced to half the value it had in the previous pass:

$$T_i = \frac{1}{2} T_{i-1}. \tag{15.92}$$

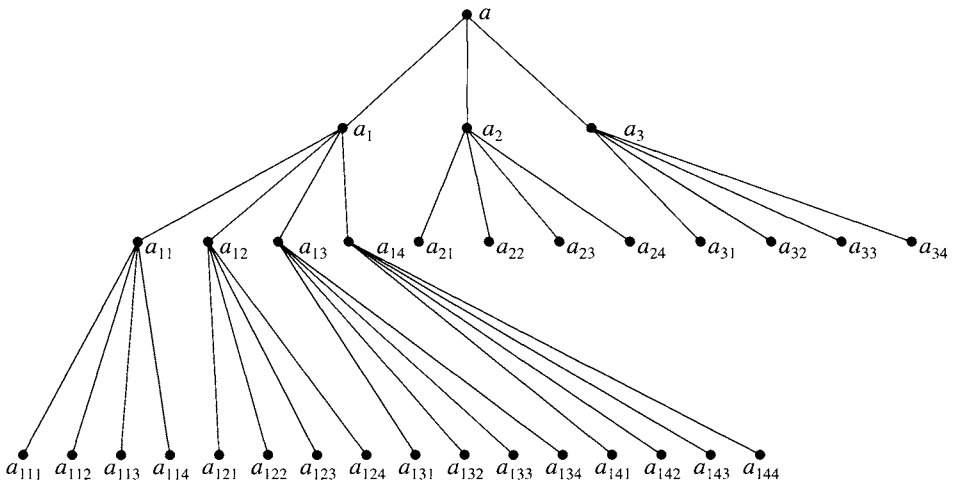

FIGURE 15. 16 Data structure used in the EZW coder.

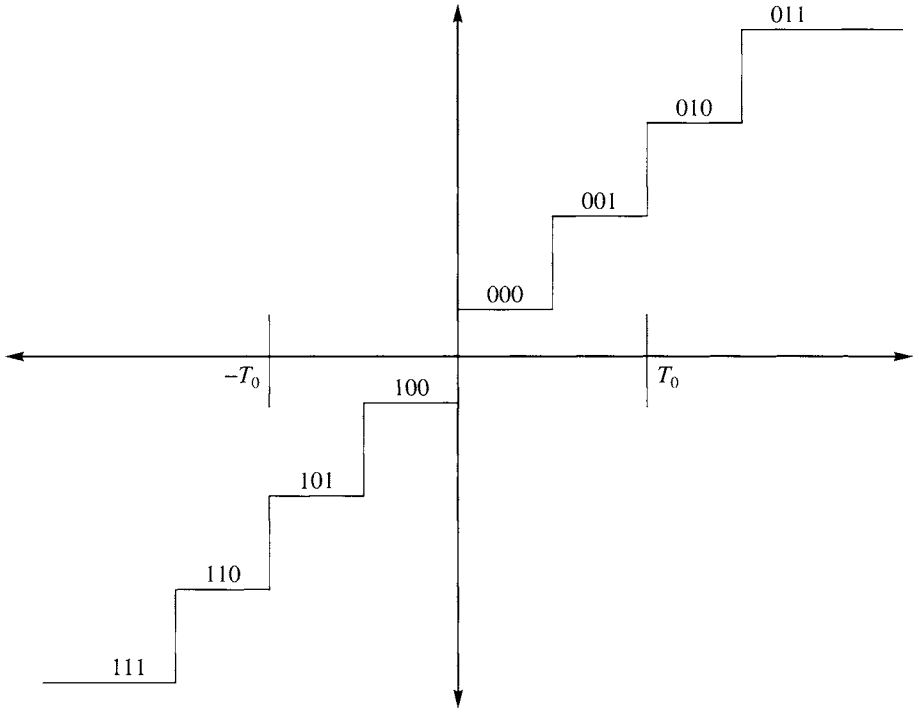

FIGURE 15. 17 A 3-bit midrise quantizer.

For a given value of T_i, we assign one of four possible labels to the coefficients: *significant positive (sp), significant negative (sn), zerotree root (zr),* and *isolated zero (iz).* If we used a fixed-length code, we would need 2 bits to represent each of the labels. Note that when a coefficient has been labeled a zerotree root, we do not need to label its descendants. This assignment is referred to as *significance map coding.*

We can view the significance map coding in part as quantization using a three-level midtread quantizer. This situation is shown in Figure 15.18. The coefficients labeled *significant* are simply those that fall in the outer levels of the quantizer and are assigned an initial reconstructed value of $1.5T_i$ or $-1.5T_i$, depending on whether the coefficient is positive or negative. Note that selecting T_i according to (15.91) and (15.92) guarantees the significant coefficients will lie in the interval $[T, 2T)$. Once a determination of significance has been made, the significant coefficients are included in a list for further refinement in the refinement or subordinate passes. In the refinement pass, we determine whether the coefficient lies in the upper or lower half of the interval $[T, 2T)$. In successive refinement passes, as the value of T is reduced, the interval containing the significant coefficient is narrowed still further and the reconstruction is updated accordingly. An easy way to perform the refinement is to take the difference between the coefficient value and its reconstruction and quantize it using a two-level quantizer with reconstruction values $\pm T/4$. This quantized value is then added on to the current reconstruction value as a correction term.

The wavelet coefficients that have not been previously determined significant are scanned in the manner depicted in Figure 15.19, with each parent node in a tree scanned before its offspring. This makes sense because if the parent is determined to be a zerotree root, we would not need to encode the offspring.

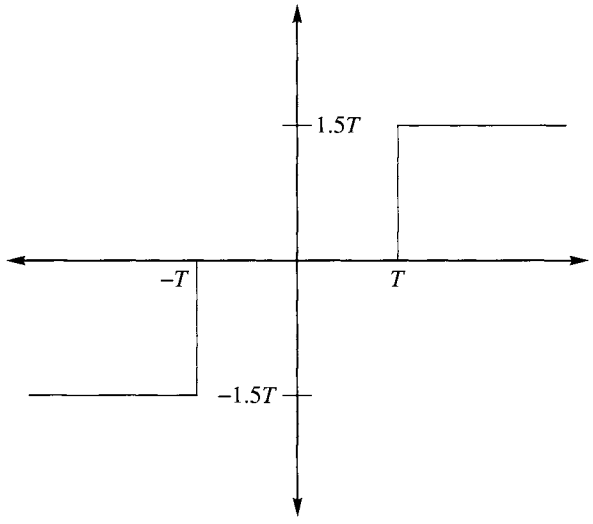

FIGURE 15. 18 A three-level midtread quantizer.

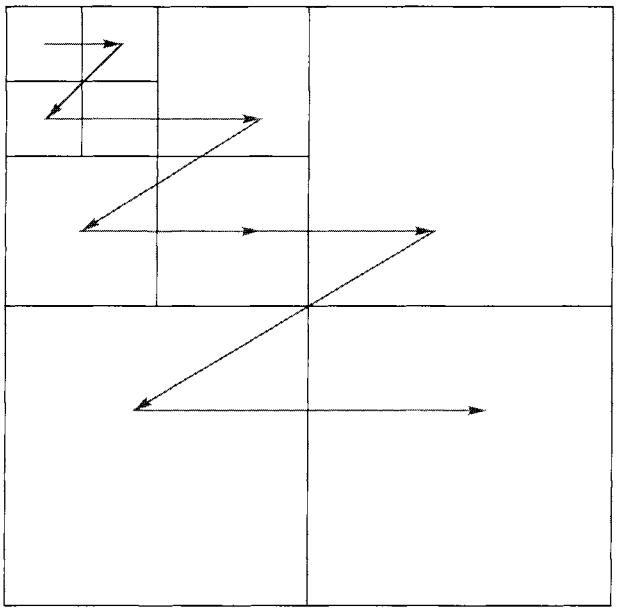

FIGURE 15. 19 **Scanning of wavelet coefficients for encoding using the EZW algorithm.**

Although this may sound confusing, in order to see how simple the encoding procedure actually is, let's use an example.

Example 15.7.1:

Let's use the seven-level decomposition shown below to demonstrate the various steps of EZW:

26	6	13	10
−7	7	6	4
4	−4	4	−3
2	−2	−2	0

To obtain the initial threshold value T_0, we find the maximum magnitude coefficient, which in this case is 26. Then

$$T_0 = 2^{\lfloor \log_2 26 \rfloor} = 16.$$

Comparing the coefficients against 16, we find 26 is greater than 16 so we send *sp*. The next coefficient in the scan is 6, which is less than 16. Furthermore, its descendants (13, 10, 6, and 4) are all less than 16. Therefore, 6 is a zerotree root, and we encode this entire set with the label *zr*. The next coefficient in the scan is −7, which is also a zerotree root, as is 7, the final element in the scan. We do not need to encode the rest of the coefficients separately because they have already been encoded as part of the various zerotrees. The sequence of labels to be transmitted at this point is

$$sp\ zr\ zr\ zr$$

Since each label requires 2 bits (for fixed-length encoding), we have used up 8 bits from our bit budget. The only significant coefficient in this pass is the coefficient with a value of 26. We include this coefficient in our list to be refined in the subordinate pass. Calling the subordinate list L_S, we have

$$L_S = \{26\}.$$

The reconstructed value of this coefficient is $1.5T_0 = 24$, and the reconstructed bands look like this:

24	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

The next step is the subordinate pass, in which we obtain a correction term for the reconstruction value of the significant coefficients. In this case, the list L_S contains only one element. The difference between this element and its reconstructed value is $26 - 24 = 2$. Quantizing this with a two-level quantizer with reconstruction levels $\pm T_0/4$, we obtain a correction term of 4. Thus, the reconstruction becomes $24 + 4 = 28$. Transmitting the correction term costs a single bit, therefore at the end of the first pass we have used up 9 bits. Using only these 9 bits, we would obtain the following reconstruction:

28	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

We now reduce the value of the threshold by a factor of two and repeat the process. The value of T_1 is 8. We rescan the coefficients that have not yet been deemed significant. To

emphasize the fact that we do not consider the coefficients that have been deemed significant in the previous pass, we replace them with $\star$:

$\star$	6	13	10
−7	7	6	4
4	−4	4	−3
2	−2	−2	0

The first coefficient we encounter has a value of 6. This is less than the threshold value of 8; however, the descendants of this coefficient include coefficients with values of 13 and 10. Therefore, this coefficient cannot be classified as a zerotree root. This is an example of what we defined as an isolated zero. The next two coefficients in the scan are −7 and 7. Both of these coefficients have magnitudes less than the threshold value of 8. Furthermore, all their descendants also have magnitudes less than 8. Therefore, these two coefficients are coded as *zr*. The next two elements in the scan are 13 and 10, which are both coded as *sp*. The final two elements in the scan are 6 and 4. These are both less than the threshold, but they do not have any descendants. We code these coefficients as *iz*. Thus, this dominant pass is coded as

$$iz\ zr\ zr\ sp\ sp\ iz\ iz$$

which requires 14 bits, bringing the total number of bits used to 23. The significant coefficients are reconstructed with values $1.5T_1 = 12$. Thus, the reconstruction at this point is

28	0	12	12
0	0	0	0
0	0	0	0
0	0	0	0

We add the new significant coefficients to the subordinate list:

$$L_S = \{26, 13, 10\}.$$

In the subordinate pass, we take the difference between the coefficients and their reconstructions and quantize these to obtain the correction or refinement values for these coefficients. The possible values for the correction terms are $\pm T_1/4 = \pm 2$:

$$\begin{aligned} 26 - 28 = -2 &\Rightarrow \text{Correction term} = -2 \\ 13 - 12 = 1 &\Rightarrow \text{Correction term} = 2 \\ 10 - 12 = -2 &\Rightarrow \text{Correction term} = -2 \end{aligned} \tag{15.93}$$

Each correction requires a single bit, bringing the total number of bits used to 26. With these corrections, the reconstruction at this stage is

26	0	14	10
0	0	0	0
0	0	0	0
0	0	0	0

If we go through one more pass, we reduce the threshold value to 4. The coefficients to be scanned are

⋆	6	⋆	⋆
−7	7	6	4
4	−4	4	−3
2	−2	−2	0

The dominant pass results in the following coded sequence:

sp sn sp sp sp sp sn iz iz sp iz iz iz

This pass cost 26 bits, equal to the total number of bits used previous to this pass. The reconstruction upon decoding of the dominant pass is

26	6	14	10
−6	6	6	6
6	−6	6	0
0	0	0	0

The subordinate list is

$$L_S = \{26, 13, 10, 6-7, 7, 6, 4, 4, -4, 4\}$$

By now it should be reasonably clear how the algorithm works. We continue encoding until we have exhausted our bit budget or until some other criterion is satisfied. ♦

There are several observations we can make from this example. Notice that the encoding process is geared to provide the most bang for the bit at each step. At each step the bits

are used to provide the maximum reduction in the reconstruction error. If at any time the encoding is interrupted, the reconstruction using this (interrupted) encoding is the best that the algorithm could have provided using this many bits. The encoding improves as more bits are transmitted. This form of coding is called *embedded coding*. In order to enhance this aspect of the algorithm, we can also sort the subordinate list at the end of each pass using information available to both encoder and decoder. This would increase the likelihood of larger coefficients being encoded first, thus providing for a greater reduction in the reconstruction error.

Finally, in the example we determined the number of bits used by assuming fixed-length encoding. In practice, arithmetic coding is used, providing a further reduction in rate.

15.8 Set Partitioning in Hierarchical Trees

The SPIHT (Set Partitioning in Hierarchical Trees) algorithm is a generalization of the EZW algorithm and was proposed by Amir Said and William Pearlman [209]. Recall that in EZW we transmit a lot of information for little cost when we declare an entire subtree to be insignificant and represent all the coefficients in it with a zerotree root label zr. The SPIHT algorithm uses a partitioning of the trees (which in SPIHT are called *spatial orientation trees*) in a manner that tends to keep insignificant coefficients together in larger subsets. The partitioning decisions are binary decisions that are transmitted to the decoder, providing a significance map encoding that is more efficient than EZW. In fact, the efficiency of the significance map encoding in SPIHT is such that arithmetic coding of the binary decisions provides very little gain. The thresholds used for checking significance are powers of two, so in essence the SPIHT algorithm sends the binary representation of the integer value of the wavelet coefficients. As in EZW, the significance map encoding, or set partitioning and ordering step, is followed by a refinement step in which the representations of the significant coefficients are refined.

Let's briefly describe the algorithm and then look at some examples of its operation. However, before we do that we need to get familiar with some notation. The data structure used by the SPIHT algorithm is similar to that used by the EZW algorithm—although not the same. The wavelet coefficients are again divided into trees originating from the lowest resolution band (band I in our case). The coefficients are grouped into 2×2 arrays that, except for the coefficients in band I, are offsprings of a coefficient of a lower resolution band. The coefficients in the lowest resolution band are also divided into 2×2 arrays. However, unlike the EZW case, all but one of them are root nodes. The coefficient in the top-left corner of the array does not have any offsprings. The data structure is shown pictorially in Figure 15.20 for a seven-band decomposition.

The trees are further partitioned into four types of sets, which are sets of coordinates of the coefficients:

- $\mathcal{O}(i, j)$ This is the set of coordinates of the offsprings of the wavelet coefficient at location (i, j). As each node can either have four offsprings or none, the size of $\mathcal{O}(i, j)$ is either zero or four. For example, in Figure 15.20 the set $\mathcal{O}(0, 1)$ consists of the coordinates of the coefficients b_1, b_2, b_3, and b_4.

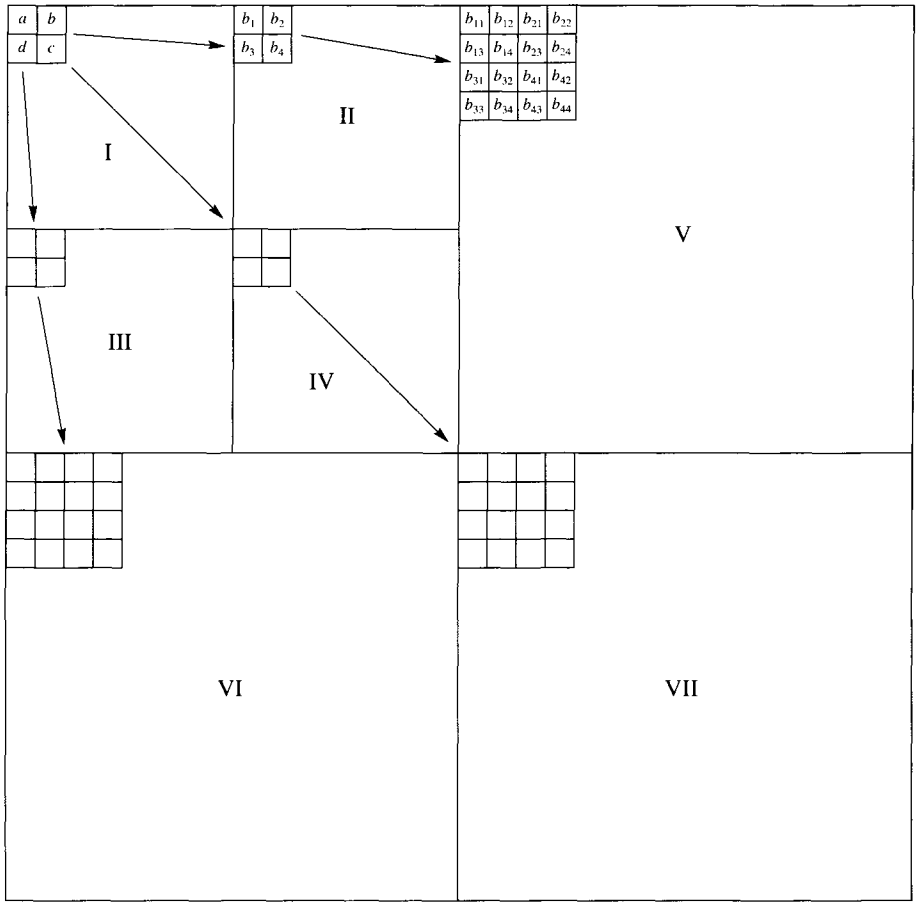

FIGURE 15. 20 Data structure used in the SPIHT algorithm.

- $\mathcal{D}(i, j)$ This is the set of all descendants of the coefficient at location (i, j). Descendants include the offsprings, the offsprings of the offsprings, and so on. For example, in Figure 15.20 the set $\mathcal{D}(0, 1)$ consists of the coordinates of the coefficients $b_1, \ldots, b_4, b_{11}, \ldots, b_{14}, \ldots, b_{44}$. Because the number of offsprings can either be zero or four, the size of $\mathcal{D}(i, j)$ is either zero or a sum of powers of four.
- $\mathcal{H}$ This is the set of all root nodes—essentially band I in the case of Figure 15.20.
- $\mathcal{L}(i, j)$ This is the set of coordinates of all the descendants of the coefficient at location (i, j) except for the immediate offsprings of the coefficient at location (i, j). In other words,

$$\mathcal{L}(i, j) = \mathcal{D}(i, j) - \mathcal{O}(i, j).$$

In Figure 15.20 the set $\mathcal{L}(0, 1)$ consists of the coordinates of the coefficients $b_{11}, \ldots, b_{14}, \ldots, b_{44}$.

A set $\mathcal{D}(i, j)$ or $\mathcal{L}(i, j)$ is said to be significant if any coefficient in the set has a magnitude greater than the threshold. Finally, thresholds used for checking significance are powers of two, so in essence the SPIHT algorithm sends the binary representation of the integer value of the wavelet coefficients. The bits are numbered with the least significant bit being the zeroth bit, the next bit being the first significant bit, and the kth bit being referred to as the $k-1$ most significant bit.

With these definitions under our belt, let us now describe the algorithm. The algorithm makes use of three lists: the *list of insignificant pixels* (LIP), the *list of significant pixels* (LSP), and the *list of insignificant sets* (LIS). The LSP and LIS lists will contain the coordinates of coefficients, while the LIS will contain the coordinates of the roots of sets of type $\mathcal{D}$ or $\mathcal{L}$. We start by determining the initial value of the threshold. We do this by calculating

$$n = \lfloor \log_2 c_{\max} \rfloor$$

where $c_{\max}$ is the maximum magnitude of the coefficients to be encoded. The LIP list is initialized with the set $\mathcal{H}$. Those elements of $\mathcal{H}$ that have descendants are also placed in LIS as type $\mathcal{D}$ entries. The LSP list is initially empty.

In each pass, we will first process the members of LIP, then the members of LIS. This is essentially the significance map encoding step. We then process the elements of LSP in the refinement step.

We begin by examining each coordinate contained in LIP. If the coefficient at that coordinate is significant (that is, it is greater than 2^n), we transmit a 1 followed by a bit representing the sign of the coefficient (we will assume 1 for positive, 0 for negative). We then move that coefficient to the LSP list. If the coefficient at that coordinate is not significant, we transmit a 0.

After examining each coordinate in LIP, we begin examining the sets in LIS. If the set at coordinate (i, j) is not significant, we transmit a 0. If the set is significant, we transmit a 1. What we do after that depends on whether the set is of type $\mathcal{D}$ or $\mathcal{L}$.

If the set is of type $\mathcal{D}$, we check each of the offsprings of the coefficient at that coordinate. In other words, we check the four coefficients whose coordinates are in $\mathcal{O}(i, j)$. For each coefficient that is significant, we transmit a 1, the sign of the coefficient, and then move the coefficient to the LSP. For the rest we transmit a 0 and add their coordinates to the LIP. Now that we have removed the coordinates of $\mathcal{O}(i, j)$ from the set, what is left is simply the set $\mathcal{L}(i, j)$. If this set is not empty, we move it to the end of the LIS and mark it to be of type $\mathcal{L}$. Note that this new entry into the LIS has to be examined during *this* pass. If the set is empty, we remove the coordinate (i, j) from the list.

If the set is of type $\mathcal{L}$, we add each coordinate in $\mathcal{O}(i, j)$ to the end of the LIS as the root of a set of type $\mathcal{D}$. Again, note that these new entries in the LIS have to be examined during this pass. We then remove (i, j) from the LIS.

Once we have processed each of the sets in the LIS (including the newly formed ones), we proceed to the refinement step. In the refinement step we examine each coefficient that was in the LSP *prior to the current pass* and output the nth most significant bit of $|c_{i,j}|$.

We ignore the coefficients that have been added to the list in this pass because, by declaring them significant at this particular level, we have already informed the decoder of the value of the nth most significant bit.

This completes one pass. Depending on the availability of more bits or external factors, if we decide to continue with the coding process, we decrement n by one and continue. Let's see the functioning of this algorithm on an example.

Example 15.8.1:

Let's use the same example we used for demonstrating the EZW algorithm:

26	6	13	10
−7	7	6	4
4	−4	4	−3
2	−2	−2	0

We will go through three passes at the encoder and generate the transmitted bitstream, then decode this bitstream.

First Pass The value for n in this case is 4. The three lists at the encoder are

$$\text{LIP}: \{(0,0) \rightarrow 26, (0,1) \rightarrow 6, (1,0) \rightarrow -7, (1,1) \rightarrow 7\}$$
$$\text{LIS}: \{(0,1)\mathcal{D}, (1,0)\mathcal{D}, (1,1)\mathcal{D}\}$$
$$\text{LSP}: \{\}$$

In the listing for LIP, we have included the $\rightarrow$ # to make it easier to follow the example. Beginning our algorithm, we examine the contents of LIP. The coefficient at location (0, 0) is greater than 16. In other words, it is significant; therefore, we transmit a 1, then a 0 to indicate the coefficient is positive and move the coordinate to LSP. The next three coefficients are all insignificant at this value of the threshold; therefore, we transmit a 0 for each coefficient and leave them in LIP. The next step is to examine the contents of LIS. Looking at the descendants of the coefficient at location (0, 1) (13, 10, 6, and 4), we see that none of them are significant at this value of the threshold so we transmit a 0. Looking at the descendants of c_{10} and c_{11}, we can see that none of these are significant at this value of the threshold. Therefore, we transmit a 0 for each set. As this is the first pass, there are no elements from the previous pass in LSP; therefore, we do not do anything in the refinement pass. We have transmitted a total of 8 bits at the end of this pass (10000000), and the situation of the three lists is as follows:

$$\text{LIP}: \{(0,1) \rightarrow 6, (1,0) \rightarrow -7, (1,1) \rightarrow 7\}$$
$$\text{LIS}: \{(0,1)\mathcal{D}, (1,0)\mathcal{D}, (1,1)\mathcal{D}\}$$
$$\text{LSP}: \{(0,0) \rightarrow 26\}$$

Second Pass For the second pass we decrement n by 1 to 3, which corresponds to a threshold value of 8. Again, we begin our pass by examining the contents of LIP. There are three elements in LIP. Each is insignificant at this threshold so we transmit three 0s. The next step is to examine the contents of LIS. The first element of LIS is the set containing the descendants of the coefficient at location (0, 1). Of this set, both 13 and 10 are significant at this value of the threshold; in other words, the set $\mathcal{D}(0, 1)$ is significant. We signal this by sending a 1 and examine the offsprings of c_{01}. The first offspring has a value of 13, which is significant and positive, so we send a 1 followed by a 0. The same is true for the second offspring, which has a value of 10. So we send another 1 followed by a 0. We move the coordinates of these two to the LSP. The next two offsprings are both insignificant at this level; therefore, we move these to LIP and transmit a 0 for each. As $\mathcal{L}(0, 1) = \{\}$, we remove $(0, 1)\mathcal{D}$ from LIS. Looking at the other elements of LIS, we can clearly see that both of these are insignificant at this level; therefore, we send a 0 for each. In the refinement pass we examine the contents of LSP from the previous pass. There is only one element in there that is not from the current sorting pass, and it has a value of 26. The third MSB of 26 is 1; therefore, we transmit a 1 and complete this pass. In the second pass we have transmitted 13 bits: 0001101000001. The condition of the lists at the end of the second pass is as follows:

$$\begin{aligned}
\text{LIP}: &\ \{(0, 1) \rightarrow 6, (1, 0) \rightarrow -7, (1, 1) \rightarrow 7(1, 2) \rightarrow 6, (1, 3) \rightarrow 4\} \\
\text{LIS}: &\ \{(1, 0)\mathcal{D}, (1, 1)\mathcal{D}\} \\
\text{LSP}: &\ \{(0, 0) \rightarrow 26, (0, 2) \rightarrow 13, (0, 3) \rightarrow 10\}
\end{aligned}$$

Third Pass The third pass proceeds with $n = 2$. As the threshold is now smaller, there are significantly more coefficients that are deemed significant, and we end up sending 26 bits. You can easily verify for yourself that the transmitted bitstream for the third pass is 10111010101101100110000010. The condition of the lists at the end of the third pass is as follows:

$$\begin{aligned}
\text{LIP}: &\ \{(3, 0) \rightarrow 2, (3, 1) \rightarrow -2, (2, 3) \rightarrow -3, (3, 2) \rightarrow -2, (3, 3) \rightarrow 0\} \\
\text{LIS}: &\ \{\} \\
\text{LSP}: &\ \{(0, 0) \rightarrow 26, (0, 2) \rightarrow 13, (0, 3) \rightarrow 10, (0, 1) \rightarrow 6, (1, 0) \rightarrow -7, (1, 1) \rightarrow 7, \\
&\ (1, 2) \rightarrow 6, (1, 3) \rightarrow 4(2, 0) \rightarrow 4, (2, 1) \rightarrow -4, (2, 2) \rightarrow 4\}
\end{aligned}$$

Now for decoding this sequence. At the decoder we also start out with the same lists as the encoder:

$$\begin{aligned}
\text{LIP}: &\ \{(0, 0), (0, 1), (1, 0), (1, 1)\} \\
\text{LIS}: &\ \{(0, 1)\mathcal{D}, (1, 0)\mathcal{D}, (1, 1)\mathcal{D}\} \\
\text{LSP}: &\ \{\}
\end{aligned}$$

We assume that the initial value of n is transmitted to the decoder. This allows us to set the threshold value at 16. Upon receiving the results of the first pass (10000000), we can see

that the first element of LIP is significant and positive and no other coefficient is significant at this level. Using the same reconstruction procedure as in EZW, we can reconstruct the coefficients at this stage as

24	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

and, following the same procedure as at the encoder, the lists can be updated as

$$\text{LIP}: \{(0,1),(1,0),(1,1)\}$$
$$\text{LIS}: \{(0,1)\mathcal{D},(1,0)\mathcal{D},(1,1)\mathcal{D}\}$$
$$\text{LSP}: \{(0,0)\}$$

For the second pass we decrement n by one and examine the transmitted bitstream: 0001101000001. Since the first 3 bits are 0 and there are only three entries in LIP, all the entries in LIP are still insignificant. The next 9 bits give us information about the sets in LIS. The fourth bit of the received bitstream is 1. This means that the set with root at coordinate (0,1) is significant. Since this set is of type $\mathcal{D}$, the next bits relate to its offsprings. The 101000 sequence indicates that the first two offsprings are significant at this level and positive and the last two are insignificant. Therefore, we move the first two offsprings to LSP and the last two to LIP. We can also approximate these two significant coefficients in our reconstruction by $1.5 \times 2^3 = 12$. We also remove $(0,1)\mathcal{D}$ from LIS. The next two bits are both 0, indicating that the two remaining sets are still insignificant. The final bit corresponds to the refinement pass. It is a 1, so we update the reconstruction of the (0, 0) coefficient to $24 + 8/2 = 28$. The reconstruction at this stage is

28	0	12	12
0	0	0	0
0	0	0	0
0	0	0	0

and the lists are as follows:

$$\text{LIP}: \{(0,1),(1,0),(1,1),(1,2),(1,3)\}$$
$$\text{LIS}: \{(1,0)\mathcal{D},(1,1)\mathcal{D}\}$$
$$\text{LSP}: \{(0,0),(0,2),(0,3)\}$$

For the third pass we again decrement n, which is now 2, giving a threshold value of 4. Decoding the bitstream generated during the third pass (101110101011011001100000010), we update our reconstruction to

26	6	14	10
−6	6	6	6
6	−6	6	0
0	0	0	0

and our lists become

LIP: $\{(3,0),(3,1)\}$

LIS: $\{\}$

LSP: $\{(0,0),(0,2),(0,3),(0,1),(1,0),(1,1),(1,2),(2,0),(2,1),(3,2)\}$

At this stage we do not have any sets left in LIS and we simply update the values of the coefficients. ♦

Finally, let's look at an example of an image coded using SPIHT. The image shown in Figure 15.21 is the reconstruction obtained from a compressed representation that used 0.5

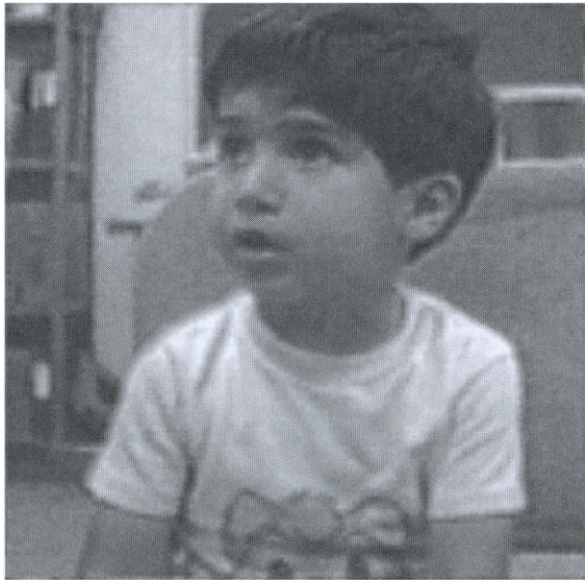

FIGURE 15.21 **Reconstruction of Sinan image encoded using SPIHT at 0.5 bits per pixel.**

bits per pixel. (The programs used to generate this image were obtained from the authors) Comparing this with Figure 15.14, we can see a definite improvement in the quality.

Wavelet decomposition has been finding its way into various standards. The earliest example was the FBI fingerprint image compression standard. The latest is the new image compression being developed by the JPEG committee, commonly referred to as JPEG 2000. We take a brief look at the current status of JPEG 2000.

15.9 JPEG 2000

The current JPEG standard provides excellent performance at rates above 0.25 bits per pixel. However, at lower rates there is a sharp degradation in the quality of the reconstructed image. To correct this and other shortcomings, the JPEG committee initiated work on another standard, commonly known as JPEG 2000. The JPEG 2000 is the standard will be based on wavelet decomposition.

There are actually two types of wavelet filters that are included in the standard. One type is the wavelet filters we have been discussing in this chapter. Another type consists of filters that generate integer coefficients; this type is particularly useful when the wavelet decomposition is part of a lossless compression scheme.

The coding scheme is based on a scheme, originally proposed by Taubman [210] and Taubman and Zakhor [211], known as EBCOT. The acronym EBCOT stands for "Embedded Block Coding with Optimized Truncation," which nicely summarizes the technique. It is a block coding scheme that generates an embedded bitstream. The block coding is independently performed on nonoverlapping blocks within individual subbands. Within a subband all blocks that do not lie on the right or lower boundaries are required to have the same dimensions. A dimension cannot exceed 256.

Embedding and independent block coding seem inherently contradictory. The way EBCOT resolves this contradiction is to organize the bitstream in a succession of layers. Each layer corresponds to a certain distortion level. Within each layer each block is coded with a variable number of bits (which could be zero). The partitioning of bits between blocks is obtained using a Lagrangian optimization that dictates the partitioning or truncation points. The quality of the reproduction is proportional to the numbers of layers received.

The embedded coding scheme is similar in philosophy to the EZW and SPIHT algorithms; however, the data structures used are different. The EZW and SPIHT algorithms used trees of coefficients from the same spatial location across different bands. In the case of the EBCOT algorithm, each block resides entirely within a subband, and each block is coded independently of other blocks, which precludes the use of trees of the type used by EZW and SPIHT. Instead, the EBCOT algorithm uses a quadtree data structure. At the lowest level, we have a 2×2 set of blocks of coefficients. These are, in turn, organized into sets of 2×2 *quads*, and so on. A node in this tree is said to be significant at level n if any of its descendants are significant at that level. A coefficient c_{ij} is said to be significant at level n if $|c_{ij}| \geq 2^n$. As in the case of EZW and SPIHT, the algorithm makes multiple passes, including significance map encoding passes and a magnitude refinement pass. The bits generated during these procedures are encoded using arithmetic coding.

15.10 Summary

In this chapter we have introduced the concepts of wavelets and multiresolution analysis, and we have seen how we can use wavelets to provide an efficient decomposition of signals prior to compression. We have also described several compression techniques based on wavelet decomposition. Wavelets and their applications are currently areas of intensive research.

Further Reading

1. There are a number of excellent introductory books on wavelets. The one I found most accessible was *Introduction to Wavelets and Wavelet Transforms—A Primer*, by C.S. Burrus, R.A. Gopinath, and H. Guo [207].
2. Probably the best mathematical source on wavelets is the book *Ten Lectures on Wavelets*, by I. Daubechies [58].
3. There are a number of tutorials on wavelets available on the Internet. The best source for all matters related to wavelets (and more) on the Internet is "The Wavelet Digest" (*http://www.wavelet.org*). This site includes pointers to many other interesting and useful sites dealing with different applications of wavelets.
4. The JPEG 2000 standard is covered in detail in *JPEG 2000: Image Compression Fundamentals, Standards and Practice*, by D. Taubman and M. Marcellin [212].

15.11 Projects and Problems

1. In this problem we consider the boundary effects encountered when using the short-term Fourier transform. Given the signal

$$f(t) = \sin(2t)$$

   (a) Find the Fourier transform $F(\omega)$ of $f(t)$.

   (b) Find the STFT $F_1(\omega)$ of $f(t)$ using a rectangular window

$$g(t) = \begin{cases} 1 & -2 \le t \le 2 \\ 0 & \text{otherwise} \end{cases}$$

   for the interval $[-2, 2]$.

   (c) Find the STFT $F_2(\omega)$ of $f(t)$ using a window

$$g(t) = \begin{cases} 1+\cos(\frac{\pi}{2}t) & -2 \le t \le 2 \\ 0 & \text{otherwise.} \end{cases}$$

   (d) Plot $|F(\omega)|$, $|F_1(\omega)|$, and $|F_2(\omega)|$. Comment on the effect of using different window functions.

2. For the function

$$f(t) = \begin{cases} 1+\sin(2t) & 0 \le t \le 1 \\ \sin(2t) & \text{otherwise} \end{cases}$$

using the Haar wavelet find and plot the coefficients $\{c_{j,k}\}$, $j = 0, 1, 2$; $k = 0, \ldots, 10$.

3. For the seven-level decomposition shown below:

21	6	15	12
−6	3	6	3
3	−3	0	−3
3	0	0	0

(a) Find the bitstream generated by the EZW coder.

(b) Decode the bitstream generated in the previous step. Verify that you get the original coefficient values.

4. Using the coefficients from the seven-level decomposition in the previous problem:

(a) Find the bitstream generated by the SPIHT coder.

(b) Decode the bitstream generated in the previous step. Verify that you get the original coefficient values.