

Dividing the numerator by the denominator we get the following:

$$\begin{array}{r}
 1 + az^{-1} + a^2z^{-2} + \dots \\
 z - a \overline{) z} \\
 \underline{z \quad - \quad a} \\
 a \\
 \underline{a \quad - \quad a^2z^{-1}} \\
 a^2z^{-1}
 \end{array}$$

Thus, the quotient is

$$1 + az^{-1} + a^2z^{-2} + \dots = \sum_{n=0}^{\infty} a^n z^{-n}.$$

We can easily see that the sequence for which  $F(z)$  is the Z-transform is

$$f_n = a^n u[n]. \quad \blacklozenge$$

#### 12.9.4 Z-Transform Properties

Analogous to the continuous linear systems, we can define the transfer function of a discrete linear system as a function of  $z$  that relates the Z-transform of the input to the Z-transform of the output. Let  $\{f_n\}_{n=-\infty}^{\infty}$  be the input to a discrete linear time-invariant system, and  $\{g_n\}_{n=-\infty}^{\infty}$  be the output. If  $F(z)$  is the Z-transform of the input sequence, and  $G(z)$  is the Z-transform of the output sequence, then these are related to each other by

$$G(z) = H(z)F(z) \quad (12.116)$$

and  $H(z)$  is the transfer function of the discrete linear time-invariant system.

If the input sequence  $\{f_n\}_{n=-\infty}^{\infty}$  had a Z-transform of one, then  $G(z)$  would be equal to  $H(z)$ . It is an easy matter to find the requisite sequence:

$$F(z) = \sum_{n=-\infty}^{\infty} f_n z^{-n} = 1 \Rightarrow f_n = \begin{cases} 1 & n=0 \\ 0 & \text{otherwise.} \end{cases} \quad (12.117)$$

This particular sequence is called the *discrete delta function*. The response of the system to the discrete delta function is called the impulse response of the system. Obviously, the transfer function  $H(z)$  is the Z-transform of the impulse response.

#### 12.9.5 Discrete Convolution

In the continuous time case, the output of the linear time-invariant system was a convolution of the input with the impulse response. Does the analogy hold in the discrete case? We can check this out easily by explicitly writing out the Z-transforms in Equation (12.116). For

simplicity let us assume the sequences are all one-sided; that is, they are only nonzero for nonnegative values of the subscript:

$$\sum_{n=0}^{\infty} g_n z^{-n} = \sum_{n=0}^{\infty} h_n z^{-n} \sum_{m=0}^{\infty} f_m z^{-m}. \tag{12.118}$$

Equating like powers of  $z$ :

$$\begin{aligned} g_0 &= h_0 f_0 \\ g_1 &= f_0 h_1 + f_1 h_0 \\ g_2 &= f_0 h_2 + f_1 h_1 + f_2 h_0 \\ &\vdots \\ g_n &= \sum_{m=0}^n f_m h_{n-m}. \end{aligned}$$

Thus, the output sequence is a result of the discrete convolution of the input sequence with the impulse response.

Most of the discrete linear systems we will be dealing with will be made up of delay elements, and their input-output relations can be written as constant coefficient difference equations. For example, for the system shown in Figure 12.13, the input-output relationship can be written in the form of the following difference equation:

$$g_k = a_0 f_k + a_1 f_{k-1} + a_2 f_{k-2} + b_1 g_{k-1} + b_2 g_{k-2}. \tag{12.119}$$

The transfer function of this system can be easily found by using the *shifting theorem*. The shifting theorem states that if the Z-transform of a sequence  $\{f_n\}$  is  $F(z)$ , then the Z-transform of the sequence shifted by some integer number of samples  $n_0$  is  $z^{-n_0} F(z)$ .

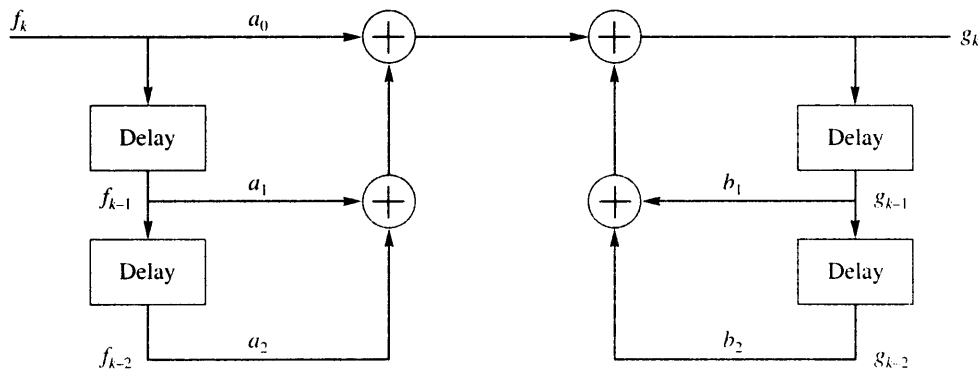


FIGURE 12. 13 A discrete system.

The theorem is easy to prove. Suppose we have a sequence  $\{f_n\}$  with Z-transform  $F(z)$ . Let us look at the Z-transform of the sequence  $\{f_{n-n_0}\}$ :

$$\mathcal{Z}\{\{f_{n-n_0}\}\} = \sum_{n=-\infty}^{\infty} f_{n-n_0} z^{-n} \quad (12.120)$$

$$= \sum_{m=-\infty}^{\infty} f_m z^{-m-n_0} \quad (12.121)$$

$$= z^{-n_0} \sum_{m=-\infty}^{\infty} f_m z^{-m} \quad (12.122)$$

$$= z^{-n_0} F(z). \quad (12.123)$$

Assuming  $G(z)$  is the Z-transform of  $\{g_n\}$  and  $F(z)$  is the Z-transform of  $\{f_n\}$ , we can take the Z-transform of both sides of the difference equation (12.119):

$$G(z) = a_0 F(z) + a_1 z^{-1} F(z) + a_2 z^{-2} F(z) + b_1 z^{-1} G(z) + b_2 z^{-2} G(z) \quad (12.124)$$

from which we get the relationship between  $G(z)$  and  $F(z)$  as

$$G(z) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - b_1 z^{-1} - b_2 z^{-2}} F(z). \quad (12.125)$$

By definition the transfer function  $H(z)$  is therefore

$$H(z) = \frac{G(z)}{F(z)} \quad (12.126)$$

$$= \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - b_1 z^{-1} - b_2 z^{-2}}. \quad (12.127)$$

## 12.10 Summary

In this chapter we have reviewed some of the mathematical tools we will be using throughout the remainder of this book. We started with a review of vector space concepts, followed by a look at a number of ways we can represent a signal, including the Fourier series, the Fourier transform, the discrete Fourier series, the discrete Fourier transform, and the Z-transform. We also looked at the operation of sampling and the conditions necessary for the recovery of the continuous representation of the signal from its samples.

### Further Reading

1. There are a large number of books that provide a much more detailed look at the concepts described in this chapter. A nice one is *Signal Processing and Linear Systems*, by B.P. Lathi [177].
2. For a thorough treatment of the fast Fourier transform (FFT), see *Numerical Recipes in C*, by W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.J. Flannery [178].

### 12.11 Projects and Problems

1. Let  $X$  be a set of  $N$  linearly independent vectors, and let  $V$  be the collection of vectors obtained using all linear combinations of the vectors in  $X$ .
  - (a) Show that given any two vectors in  $V$ , the sum of these vectors is also an element of  $V$ .
  - (b) Show that  $V$  contains an additive identity.
  - (c) Show that for every  $\mathbf{x}$  in  $V$ , there exists a  $(-\mathbf{x})$  in  $V$  such that their sum is the additive identity.
2. Prove Parseval's theorem for the Fourier transform.
3. Prove the modulation property of the Fourier transform.
4. Prove the convolution theorem for the Fourier transform.
5. Show that the Fourier transform of a train of impulses in the time domain is a train of impulses in the frequency domain:

$$\mathcal{F} \left[ \sum_{n=-\infty}^{\infty} \delta(t - nT) \right] = \sigma_0 \sum_{n=-\infty}^{\infty} \delta(w - n\sigma_0) \quad \sigma_0 = \frac{2\pi}{T}. \quad (12.128)$$

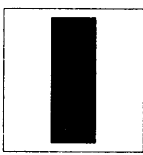
6. Find the Z-transform for the following sequences:
  - (a)  $h_n = 2^{-n}u[n]$ , where  $u[n]$  is the unit step function.
  - (b)  $h_n = (n^2 - n)3^{-n}u[n]$ .
  - (c)  $h_n = (n2^{-n} + (0.6)^n)u[n]$ .
7. Given the following input-output relationship:

$$y_n = 0.6y_{n-1} + 0.5x_n + 0.2x_{n-1}$$

- (a) Find the transfer function  $H(z)$ .
  - (b) Find the impulse response  $\{h_n\}$ .
8. Find the inverse Z-transform of the following:
    - (a)  $H(z) = \frac{5}{z-2}$ .
    - (b)  $H(z) = \frac{z}{z^2-0.25}$ .
    - (c)  $H(z) = \frac{z}{z-0.5}$ .

# Transform Coding

## 13.1 Overview

 In this chapter we will describe a technique in which the source output is decomposed, or transformed, into components that are then coded according to their individual characteristics. We will then look at a number of different transforms, including the popular discrete cosine transform, and discuss the issues of quantization and coding of the transformed coefficients. This chapter concludes with a description of the baseline sequential JPEG image-coding algorithm and some of the issues involved with transform coding of audio signals.

## 13.2 Introduction

In the previous chapter we developed a number of tools that can be used to transform a given sequence into different representations. If we take a sequence of inputs and transform them into another sequence in which most of the information is contained in only a few elements, we can then encode and transmit those elements, along with their location in the new sequence, resulting in data compression. In our discussion, we will use the terms “variance” and “information” interchangeably. The justification for this is shown in the results in Chapter 7. For example, recall that for a Gaussian source the differential entropy is given as  $\frac{1}{2} \log 2\pi e\sigma^2$ . Thus, an increase in the variance results in an increase in the entropy, which is a measure of the information contained in the source output.

To begin our discussion of transform coding, consider the following example.

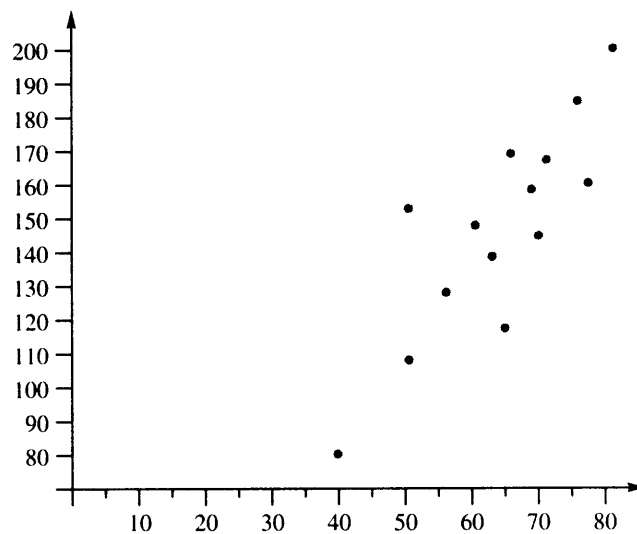
**Example 13.2.1:**

Let's revisit Example 8.5.1. In Example 8.5.1, we studied the encoding of the output of a source that consisted of a sequence of pairs of numbers. Each pair of numbers corresponds to the height and weight of an individual. In particular, let's look at the sequence of outputs shown in Table 13.1.

If we look at the height and weight as the coordinates of a point in two-dimensional space, the sequence can be shown graphically as in Figure 13.1. Notice that the output

**TABLE 13.1 Original sequence.**

Height	Weight
65	170
75	188
60	150
70	170
56	130
80	203
68	160
50	110
40	80
50	153
69	148
62	140
76	164
64	120

**FIGURE 13.1 Source output sequence.**

values tend to cluster around the line  $y = 2.5x$ . We can rotate this set of values by the transformation

$$\theta = \mathbf{A}\mathbf{x} \quad (13.1)$$

where  $\mathbf{x}$  is the two-dimensional source output vector

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \quad (13.2)$$

$x_0$  corresponds to height and  $x_1$  corresponds to weight,  $\mathbf{A}$  is the rotation matrix

$$\mathbf{A} = \begin{bmatrix} \cos \phi & \sin \phi \\ -\sin \phi & \cos \phi \end{bmatrix} \quad (13.3)$$

$\phi$  is the angle between the  $x$ -axis and the  $y = 2.5x$  line, and

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \quad (13.4)$$

is the rotated or transformed set of values. For this particular case the matrix  $\mathbf{A}$  is

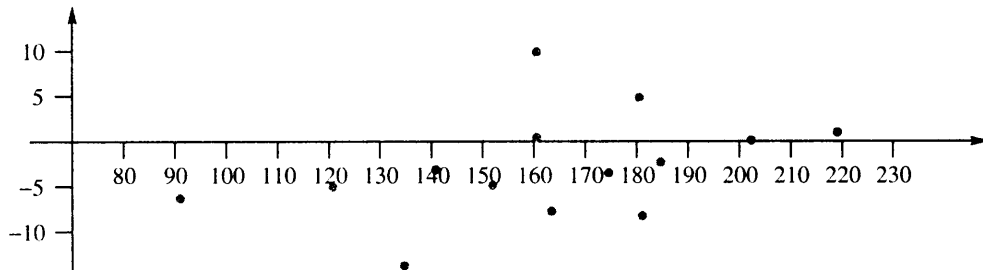
$$\mathbf{A} = \begin{bmatrix} 0.37139068 & 0.92847669 \\ -0.92847669 & 0.37139068 \end{bmatrix} \quad (13.5)$$

and the transformed sequence (rounded to the nearest integer) is shown in Table 13.2. (For a brief review of matrix concepts, see Appendix B.)

Notice that for each pair of values, almost all the energy is compacted into the first element of the pair, while the second element of the pair is significantly smaller. If we plot this sequence in pairs, we get the result shown in Figure 13.2. Note that we have rotated the original values by an angle of approximately 68 degrees ( $\arctan 2.5$ ).

**TABLE 13.2** Transformed sequence.

First Coordinate	Second Coordinate
182	3
202	0
162	0
184	-2
141	-4
218	1
174	-4
121	-6
90	-7
161	10
163	-9
153	-6
181	-9
135	-15



**FIGURE 13.2** The transformed sequence.

Suppose we set all the second elements of the transformation to zero, that is, the second coordinates of the sequence shown in Table 13.2. This reduces the number of elements that need to be encoded by half. What is the effect of throwing away half the elements of the sequence? We can find that out by taking the inverse transform of the reduced sequence. The inverse transform consists of reversing the rotation. We can do this by multiplying the blocks of two of the transformed sequences with the second element in each block set to zero with the matrix

$$\mathbf{A}^{-1} = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \quad (13.6)$$

and obtain the reconstructed sequence shown in Table 13.3. Comparing this to the original sequence in Table 13.1, we see that, even though we transmitted only half the number of elements present in the original sequence, this “reconstructed” sequence is very close to the original. The reason there is so little error introduced in the sequence  $\{x_n\}$  is that for this

**TABLE 13.3** Reconstructed sequence.

Height	Weight
68	169
75	188
60	150
68	171
53	131
81	203
65	162
45	112
34	84
60	150
61	151
57	142
67	168
50	125



particular transformation the error introduced into the  $\{x_n\}$  sequence is equal to the error introduced into the  $\{\theta_n\}$  sequence. That is,

$$\sum_{i=0}^{N-1} (x_i - \hat{x}_i)^2 = \sum_{i=0}^{N-1} (\theta_i - \hat{\theta}_i)^2 \quad (13.7)$$

where  $\{\hat{x}_n\}$  is the reconstructed sequence, and

$$\hat{\theta}_i = \begin{cases} \theta_i & i = 0, 2, 4, \dots \\ 0 & \text{otherwise} \end{cases} \quad (13.8)$$

(see Problem 1). The error introduced in the  $\{\theta_n\}$  sequence is the sum of squares of the  $\theta_n$ s that are set to zero. The magnitudes of these elements are quite small, and therefore the total error introduced into the reconstructed sequence is quite small also. ♦

We could reduce the number of samples we needed to code because most of the information contained in each pair of values was put into one element of each pair. As the other element of the pair contained very little information, we could discard it without a significant effect on the fidelity of the reconstructed sequence. The transform in this case acted on pairs of values; therefore, the maximum reduction in the number of significant samples was a factor of two. We can extend this idea to longer blocks of data. By compacting most of the information in a source output sequence into a few elements of the transformed sequence using a reversible transform, and then discarding the elements of the sequence that do not contain much information, we can get a large amount of compression. This is the basic idea behind transform coding.

In Example 13.2.1 we have presented a geometric view of the transform process. We can also examine the transform process in terms of the changes in statistics between the original and transformed sequences. It can be shown that we can get the maximum amount of compaction if we use a transform that decorrelates the input sequence; that is, the sample-to-sample correlation of the transformed sequence is zero. The first transform to provide decorrelation for discrete data was presented by Hotelling [179] in the *Journal of Educational Psychology* in 1933. He called his approach the *method of principal components*. The analogous transform for continuous functions was obtained by Karhunen [180] and Loève [181]. This decorrelation approach was first utilized for compression, in what we now call transform coding, by Kramer and Mathews [182], and Huang and Schultheiss [183].

Transform coding consists of three steps. First, the data sequence  $\{x_n\}$  is divided into blocks of size  $N$ . Each block is mapped into a transform sequence  $\{\theta_n\}$  using a reversible mapping in a manner similar to that described in Example 13.2.1. As shown in the example, different elements of each block of the transformed sequence generally have different statistical properties. In Example 13.2.1, most of the energy of the block of two input values was contained in the first element of the block of two transformed values, while very little of the energy was contained in the second element. This meant that the second element of each block of the transformed sequence would have a small magnitude, while the magnitude of the first element could vary considerably depending on the magnitude of the elements in the input block. The second step consists of quantizing the transformed sequence. The quantization strategy used will depend on three main factors: the desired average bit rate, the statistics

of the various elements of the transformed sequence, and the effect of distortion in the transformed coefficients on the reconstructed sequence. In Example 13.2.1, we could take all the bits available to us and use them to quantize the first coefficient. In more complex situations, the strategy used may be very different. In fact, we may use different techniques, such as differential encoding and vector quantization [118], to encode the different coefficients.

Finally, the quantized value needs to be encoded using some binary encoding technique. The binary coding may be as simple as using a fixed-length code or as complex as a combination of run-length coding and Huffman or arithmetic coding. We will see an example of the latter when we describe the JPEG algorithm.

The various quantization and binary coding techniques have been described at some length in previous chapters, so we will spend the next section describing various transforms. We will then discuss quantization and coding strategies in the context of these transforms.

### 13.3 The Transform

All the transforms we deal with will be linear transforms; that is, we can get the sequence  $\{\theta_n\}$  from the sequence  $\{x_n\}$  as

$$\theta_n = \sum_{i=0}^{N-1} x_i a_{n,i}. \quad (13.9)$$

This is referred to as the *forward transform*. For the transforms that we will be considering, a major difference between the transformed sequence  $\{\theta_n\}$  and the original sequence  $\{x_n\}$  is that the characteristics of the elements of the  $\theta$  sequence are determined by their position within the sequence. For example, in Example 13.2.1 the first element of each pair of the transformed sequence was more likely to have a large magnitude compared to the second element. In general, we cannot make such statements about the source output sequence  $\{x_n\}$ . A measure of the differing characteristics of the different elements of the transformed sequence  $\{\theta_n\}$  is the variance  $\sigma_n^2$  of each element. These variances will strongly influence how we encode the transformed sequence. The size of the block  $N$  is dictated by practical considerations. In general, the complexity of the transform grows more than linearly with  $N$ . Therefore, beyond a certain value of  $N$ , the computational costs overwhelm any marginal improvements that might be obtained by increasing  $N$ . Furthermore, in most real sources the statistical characteristics of the source output can change abruptly. For example, when we go from a silence period to a voiced period in speech, the statistics change drastically. Similarly, in images, the statistical characteristics of a smooth region of the image can be very different from the statistical characteristics of a busy region of the image. If  $N$  is large, the probability that the statistical characteristics change significantly within a block increases. This generally results in a larger number of the transform coefficients with large values, which in turn leads to a reduction in the compression ratio.

The original sequence  $\{x_n\}$  can be recovered from the transformed sequence  $\{\theta_n\}$  via the *inverse transform*:

$$x_n = \sum_{i=0}^{N-1} \theta_i b_{n,i}. \quad (13.10)$$

The transforms can be written in matrix form as

$$\boldsymbol{\theta} = \mathbf{A}\mathbf{x} \quad (13.11)$$

$$\mathbf{x} = \mathbf{B}\boldsymbol{\theta} \quad (13.12)$$

where  $\mathbf{A}$  and  $\mathbf{B}$  are  $N \times N$  matrices and the  $(i, j)$ th element of the matrices is given by

$$[\mathbf{A}]_{i,j} = a_{i,j} \quad (13.13)$$

$$[\mathbf{B}]_{i,j} = b_{i,j}. \quad (13.14)$$

The forward and inverse transform matrices  $\mathbf{A}$  and  $\mathbf{B}$  are inverses of each other; that is,  $\mathbf{A}\mathbf{B} = \mathbf{B}\mathbf{A} = \mathbf{I}$ , where  $\mathbf{I}$  is the identity matrix.

Equations (13.9) and (13.10) deal with the transform coding of one-dimensional sequences, such as sampled speech and audio sequences. However, transform coding is one of the most popular methods used for image compression. In order to take advantage of the two-dimensional nature of dependencies in images, we need to look at two-dimensional transforms.

Let  $X_{i,j}$  be the  $(i, j)$ th pixel in an image. A general linear two-dimensional transform for a block of size  $N \times N$  is given as

$$\Theta_{k,l} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} X_{i,j} a_{i,j,k,l}. \quad (13.15)$$

All two-dimensional transforms in use today are *separable* transforms; that is, we can take the transform of a two-dimensional block by first taking the transform along one dimension, then repeating the operation along the other direction. In terms of matrices, this involves first taking the (one-dimensional) transform of the rows, and then taking the column-by-column transform of the resulting matrix. We can also reverse the order of the operations, first taking the transform of the columns, and then taking the row-by-row transform of the resulting matrix. The transform operation can be represented as

$$\Theta_{k,l} = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} a_{k,i} X_{i,j} a_{i,j}. \quad (13.16)$$

which in matrix terminology would be given by

$$\boldsymbol{\Theta} = \mathbf{A}\mathbf{X}\mathbf{A}^T. \quad (13.17)$$

The inverse transform is given as

$$\mathbf{X} = \mathbf{B}\boldsymbol{\Theta}\mathbf{B}^T. \quad (13.18)$$

All the transforms we deal with will be *orthonormal transforms*. An orthonormal transform has the property that the inverse of the transform matrix is simply its transpose because the rows of the transform matrix form an orthonormal basis set:

$$\mathbf{B} = \mathbf{A}^{-1} = \mathbf{A}^T. \quad (13.19)$$

For an orthonormal transform, the inverse transform will be given as

$$\mathbf{X} = \mathbf{A}^T \Theta \mathbf{A}. \quad (13.20)$$

Orthonormal transforms are energy preserving; that is, the sum of the squares of the transformed sequence is the same as the sum of the squares of the original sequence. We can see this most easily in the case of the one-dimensional transform:

$$\sum_{i=0}^{N-1} \theta_i^2 = \theta^T \theta \quad (13.21)$$

$$= (\mathbf{A}\mathbf{x})^T \mathbf{A}\mathbf{x} \quad (13.22)$$

$$= \mathbf{x}^T \mathbf{A}^T \mathbf{A}\mathbf{x}. \quad (13.23)$$

If  $\mathbf{A}$  is an orthonormal transform,  $\mathbf{A}^T \mathbf{A} = \mathbf{A}^{-1} \mathbf{A} = \mathbf{I}$ , then

$$\mathbf{x}^T \mathbf{A}^T \mathbf{A}\mathbf{x} = \mathbf{x}^T \mathbf{x} \quad (13.24)$$

$$= \sum_{n=0}^{N-1} x_n^2 \quad (13.25)$$

and

$$\sum_{i=0}^{N-1} \theta_i^2 = \sum_{n=0}^{N-1} x_n^2. \quad (13.26)$$

The efficacy of a transform depends on how much energy compaction is provided by the transform. One way of measuring the amount of energy compaction afforded by a particular orthonormal transform is to take a ratio of the arithmetic mean of the variances of the transform coefficient to their geometric means [123]. This ratio is also referred to as the transform coding gain  $G_{TC}$ :

$$G_{TC} = \frac{\frac{1}{N} \sum_{i=0}^{N-1} \sigma_i^2}{\left( \prod_{i=0}^{N-1} \sigma_i^2 \right)^{\frac{1}{N}}} \quad (13.27)$$

where  $\sigma_i^2$  is the variance of the  $i$ th coefficient  $\theta_i$ .

Transforms can be interpreted in several ways. We have already mentioned a geometric interpretation and a statistical interpretation. We can also interpret them as a decomposition of the signal in terms of a basis set. For example, suppose we have a two-dimensional orthonormal transform  $\mathbf{A}$ . The inverse transform can be written as

$$\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{10} \\ a_{01} & a_{11} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \theta_0 \begin{bmatrix} a_{00} \\ a_{01} \end{bmatrix} + \theta_1 \begin{bmatrix} a_{10} \\ a_{11} \end{bmatrix} \quad (13.28)$$

We can see that the transformed values are actually the coefficients of an expansion of the input sequence in terms of the rows of the transform matrix. The rows of the transform matrix are often referred to as the basis vectors for the transform because they form an orthonormal basis set, and the elements of the transformed sequence are often called the transform coefficients. By characterizing the basis vectors in physical terms we can get a physical interpretation of the transform coefficients.

**Example 13.3.1:**

Consider the following transform matrix:

$$\mathbf{A} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (13.29)$$

We can verify that this is indeed an orthonormal transform.

Notice that the first row of the matrix would correspond to a “low-pass” signal (no change from one component to the next), while the second row would correspond to a “high-pass” signal. Thus, if we tried to express a sequence in which each element has the same value in terms of these two rows, the second coefficient should be zero. Suppose the original sequence is  $(\alpha, \alpha)$ . Then

$$\begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \alpha \\ \alpha \end{bmatrix} = \begin{bmatrix} \sqrt{2}\alpha \\ 0 \end{bmatrix} \quad (13.30)$$

The “low-pass” coefficient has a value of  $\sqrt{2}\alpha$ , while the “high-pass” coefficient has a value of 0. The “low-pass” and “high-pass” coefficients are generally referred to as the low-frequency and high-frequency coefficients.

Let us take two sequences in which the components are not the same and the degree of variation is different. Consider the two sequences  $(3, 1)$  and  $(3, -1)$ . In the first sequence the second element differs from the first by 2; in the second sequence, the magnitude of the difference is 4. We could say that the second sequence is more “high pass” than the first sequence. The transform coefficients for the two sequences are  $(2\sqrt{2}, \sqrt{2})$  and  $(\sqrt{2}, 2\sqrt{2})$ , respectively. Notice that the high-frequency coefficient for the sequence in which we see a larger change is twice that of the high-frequency coefficient for the sequence with less change. Thus, the two coefficients do seem to behave like the outputs of a low-pass filter and a high-pass filter.

Finally, notice that in every case the sum of the squares of the original sequence is the same as the sum of the squares of the transform coefficients; that is, the transform is energy preserving, as it must be, since  $\mathbf{A}$  is orthonormal.  $\blacklozenge$

We can interpret one-dimensional transforms as an expansion in terms of the rows of the transform matrix. Similarly, we can interpret two-dimensional transforms as expansions in terms of matrices that are formed by the outer product of the rows of the transform matrix. Recall that the outer product is given by

$$\mathbf{xx}^T = \begin{bmatrix} x_0x_0 & x_0x_1 & \cdots & x_0x_{N-1} \\ x_1x_0 & x_1x_1 & \cdots & x_1x_{N-1} \\ \vdots & \vdots & & \vdots \\ x_{N-1}x_0 & x_{N-1}x_1 & \cdots & x_{N-1}x_{N-1} \end{bmatrix} \quad (13.31)$$

To see this more clearly, let us use the transform introduced in Example 13.3.1 for a two-dimensional transform.

**Example 13.3.2:**

For an  $N \times N$  transform  $\mathbf{A}$ , let  $\alpha_{i,j}$  be the outer product of the  $i$ th and  $j$ th rows:

$$\alpha_{i,j} = \begin{bmatrix} a_{i0} \\ a_{i1} \\ \vdots \\ a_{iN-1} \end{bmatrix} [a_{j0} \ a_{j1} \ \cdots \ a_{jN-1}] \quad (13.32)$$

$$= \begin{bmatrix} a_{i0}a_{j0} & a_{i0}a_{j1} & \cdots & a_{i0}a_{jN-1} \\ a_{i1}a_{j0} & a_{i1}a_{j1} & \cdots & a_{i1}a_{jN-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{iN-1}a_{j0} & a_{iN-1}a_{j1} & \cdots & a_{iN-1}a_{jN-1} \end{bmatrix} \quad (13.33)$$

For the transform of Example 13.3.1, the outer products are

$$\alpha_{0,0} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \alpha_{0,1} = \frac{1}{2} \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \quad (13.34)$$

$$\alpha_{1,0} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix} \alpha_{1,1} = \frac{1}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \quad (13.35)$$

From (13.20), the inverse transform is given by

$$\begin{bmatrix} x_{01} & x_{01} \\ x_{10} & x_{11} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \theta_{00} & \theta_{01} \\ \theta_{10} & \theta_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (13.36)$$

$$= \frac{1}{2} \begin{bmatrix} \theta_{00} + \theta_{01} + \theta_{10} + \theta_{11} & \theta_{00} - \theta_{01} + \theta_{10} - \theta_{11} \\ \theta_{00} + \theta_{01} - \theta_{10} - \theta_{11} & \theta_{00} - \theta_{01} - \theta_{10} + \theta_{11} \end{bmatrix} \quad (13.37)$$

$$= \theta_{00}\alpha_{0,0} + \theta_{01}\alpha_{0,1} + \theta_{10}\alpha_{1,0} + \theta_{11}\alpha_{1,1}. \quad (13.38)$$

The transform values  $\theta_{ij}$  can be viewed as the coefficients of the expansion of  $\mathbf{x}$  in terms of the matrices  $\alpha_{i,j}$ . The matrices  $\alpha_{i,j}$  are known as the *basis* matrices.

For historical reasons, the coefficient  $\theta_{00}$ , corresponding to the basis matrix  $\alpha_{0,0}$ , is called the DC coefficient, while the coefficients corresponding to the other basis matrices are called AC coefficients. DC stands for direct current, which is current that does not change with time. AC stands for alternating current, which does change with time. Notice that all the elements of the basis matrix  $\alpha_{0,0}$  are the same, hence the DC designation. ♦

In the following section we will look at some of the variety of transforms available to us, then at some of the issues involved in quantization and coding. Finally, we will describe in detail two applications, one for image coding and one for audio coding.

**13.4 Transforms of Interest**

In Example 13.2.1, we constructed a transform that was specific to the data. In practice, it is generally not feasible to construct a transform for the specific situation, for several

reasons. Unless the characteristics of the source output are stationary over a long interval, the transform needs to be recomputed often, and it is generally burdensome to compute a transform for every different set of data. Furthermore, the overhead required to transmit the transform itself might negate any compression gains. Both of these problems become especially acute when the size of the transform is large. However, there are times when we want to find out the best we can do with transform coding. In these situations, we can use data-dependent transforms to obtain an idea of the best performance available. The best-known data-dependent transform is the discrete Karhunen-Loève transform (KLT). We will describe this transform in the next section.

### 13.4.1 Karhunen-Loève Transform

The rows of the discrete Karhunen-Loève transform [184], also known as the Hotelling transform, consist of the eigenvectors of the autocorrelation matrix. The autocorrelation matrix for a random process  $X$  is a matrix whose  $(i, j)$ th element  $[R]_{i,j}$  is given by

$$[R]_{i,j} = E[X_n X_{n+|i-j|}]. \quad (13.39)$$

We can show [123] that a transform constructed in this manner will minimize the geometric mean of the variance of the transform coefficients. Hence, the Karhunen-Loève transform provides the largest transform coding gain of any transform coding method.

If the source output being compressed is nonstationary, the autocorrelation function will change with time. Thus, the autocorrelation matrix will change with time, and the KLT will have to be recomputed. For a transform of any reasonable size, this is a significant amount of computation. Furthermore, as the autocorrelation is computed based on the source output, it is not available to the receiver. Therefore, either the autocorrelation or the transform itself has to be sent to the receiver. The overhead can be significant and remove any advantages to using the optimum transform. However, in applications where the statistics change slowly and the transform size can be kept small, the KLT can be of practical use [185].

#### Example 13.4.1:

Let us see how to obtain the KLT transform of size two for an arbitrary input sequence. The autocorrelation matrix of size two for a stationary process is

$$\mathbf{R} = \begin{bmatrix} R_{xx}(0) & R_{xx}(1) \\ R_{xx}(1) & R_{xx}(0) \end{bmatrix} \quad (13.40)$$

Solving the equation  $|\lambda \mathbf{I} - \mathbf{R}| = 0$ , we get the two eigenvalues  $\lambda_1 = R_{xx}(0) + R_{xx}(1)$ , and  $\lambda_2 = R_{xx}(0) - R_{xx}(1)$ . The corresponding eigenvectors are

$$V_1 = \begin{bmatrix} \alpha \\ \alpha \end{bmatrix} \quad V_2 = \begin{bmatrix} \beta \\ -\beta \end{bmatrix} \quad (13.41)$$

where  $\alpha$  and  $\beta$  are arbitrary constants. If we now impose the orthonormality condition, which requires the vectors to have a magnitude of 1, we get

$$\alpha = \beta = \frac{1}{\sqrt{2}}$$

and the transform matrix  $\mathbf{K}$  is

$$\mathbf{K} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (13.42)$$

Notice that this matrix is not dependent on the values of  $R_{xx}(0)$  and  $R_{xx}(1)$ . This is only true of the  $2 \times 2$  KLT. The transform matrices of higher order are functions of the autocorrelation values.  $\blacklozenge$

Although the Karhunen-Loève transform maximizes the transform coding gain as defined by (13.27), it is not practical in most circumstances. Therefore, we need transforms that do not depend on the data being transformed. We describe some of the more popular transforms in the following sections.

### 13.4.2 Discrete Cosine Transform

The discrete cosine transform (DCT) gets its name from the fact that the rows of the  $N \times N$  transform matrix  $\mathbf{C}$  are obtained as a function of cosines.

$$[\mathbf{C}]_{i,j} = \begin{cases} \sqrt{\frac{1}{N}} \cos \frac{(2j+1)i\pi}{2N} & i = 0, j = 0, 1, \dots, N-1 \\ \sqrt{\frac{2}{N}} \cos \frac{(2j+1)i\pi}{2N} & i = 1, 2, \dots, N-1, j = 0, 1, \dots, N-1. \end{cases} \quad (13.43)$$

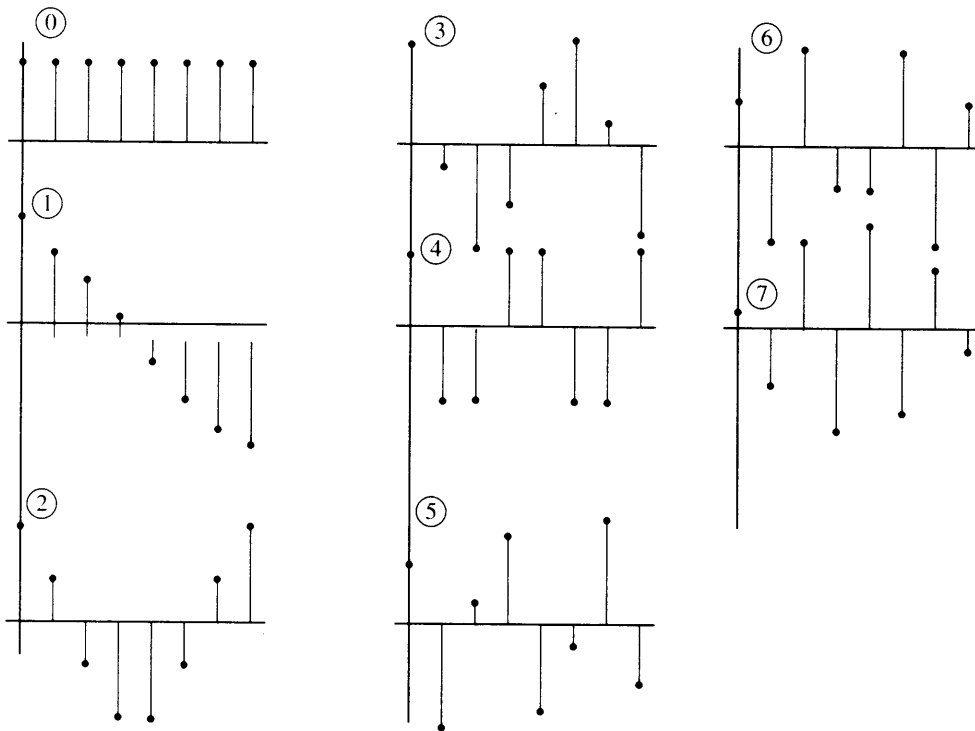
The rows of the transform matrix are shown in graphical form in Figure 13.3. Notice how the amount of variation increases as we progress down the rows; that is, the frequency of the rows increases as we go from top to bottom.

The outer products of the rows are shown in Figure 13.4. Notice that the basis matrices show increased variation as we go from the top-left matrix, corresponding to the  $\theta_{00}$  coefficient, to the bottom-right matrix, corresponding to the  $\theta_{(N-1)(N-1)}$  coefficient.

The DCT is closely related to the discrete Fourier transform (DFT) mentioned in Chapter 11, and in fact can be obtained from the DFT. However, in terms of compression, the DCT performs better than the DFT.

To see why, recall that when we find the Fourier coefficients for a sequence of length  $N$ , we assume that the sequence is periodic with period  $N$ . If the original sequence is as shown in Figure 13.5a, the DFT assumes that the sequence outside the interval of interest behaves in the manner shown in Figure 13.5b. This introduces sharp discontinuities, at the beginning and the end of the sequence. In order to represent these sharp discontinuities, the DFT needs nonzero coefficients for the high-frequency components. Because these components are needed only at the two endpoints of the sequence, their effect needs to be canceled out at other points in the sequence. Thus, the DFT adjusts other coefficients accordingly. When we discard the high-frequency coefficients (which should not have been there anyway) during





**FIGURE 13.3** Basis set for the discrete cosine transform. The numbers in the circles correspond to the row of the transform matrix.

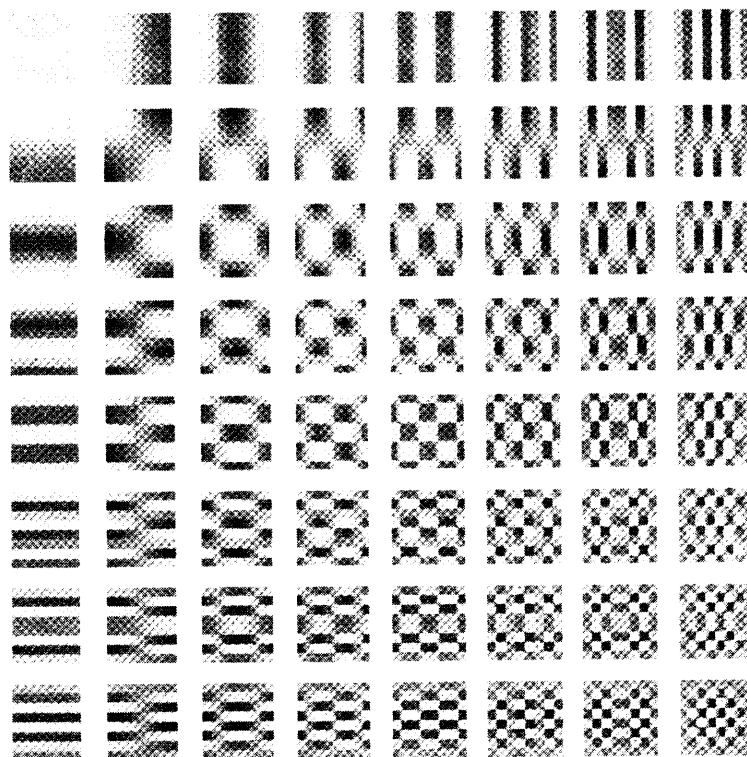
the compression process, the coefficients that were canceling out the high-frequency effect in other parts of the sequence result in the introduction of additional distortion.

The DCT can be obtained using the DFT by mirroring the original  $N$ -point sequence to obtain a  $2N$ -point sequence, as shown in Figure 13.6b. The DCT is simply the first  $N$  points of the resulting  $2N$ -point DFT. When we take the DFT of the  $2N$ -point mirrored sequence, we again have to assume periodicity. However, as we can see from Figure 13.6c, this does not introduce any sharp discontinuities at the edges.

The DCT is substantially better at energy compaction for most correlated sources when compared to the DFT [123]. In fact, for Markov sources with high correlation coefficient  $\rho$ ,

$$\rho = \frac{E[x_n x_{n+1}]}{E[x_n^2]}, \quad (13.44)$$

the compaction ability of the DCT is very close to that of the KLT. As many sources can be modeled as Markov sources with high values for  $\rho$ , this superior compaction ability has made the DCT the most popular transform. It is a part of many international standards, including JPEG, MPEG, and CCITT H.261, among others.



**FIGURE 13.4** The basis matrices for the DCT.

### 13.4.3 Discrete Sine Transform

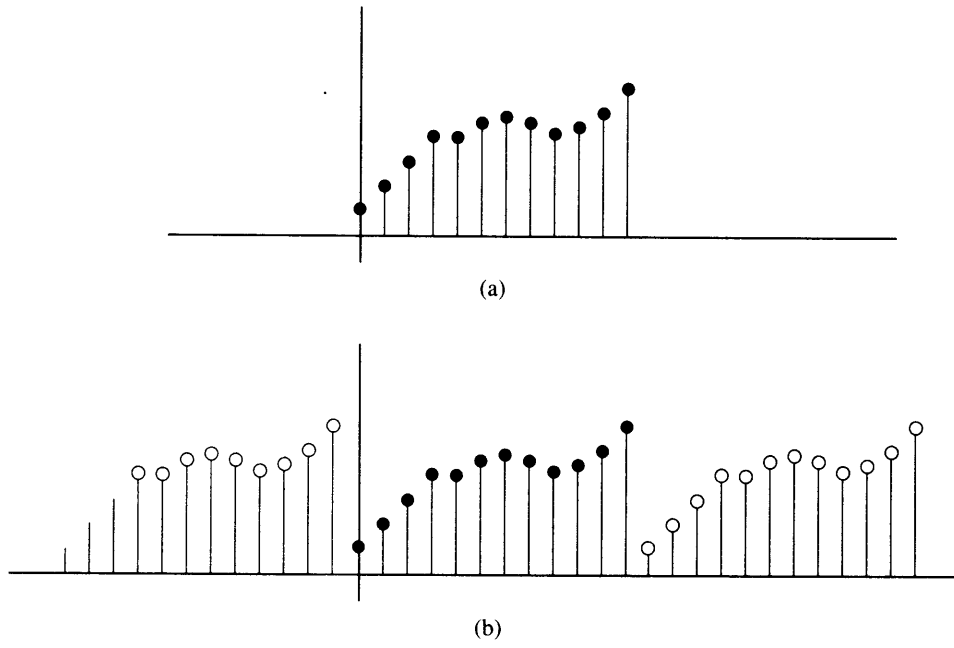
The discrete sine transform (DST) is a complementary transform to the DCT. Where the DCT provides performance close to the optimum KLT when the correlation coefficient  $\rho$  is large, the DST performs close to the optimum KLT in terms of compaction when the magnitude of  $\rho$  is small. Because of this property, it is often used as the complementary transform to DCT in image [186] and audio [187] coding applications.

The elements of the transform matrix for an  $N \times N$  DST are

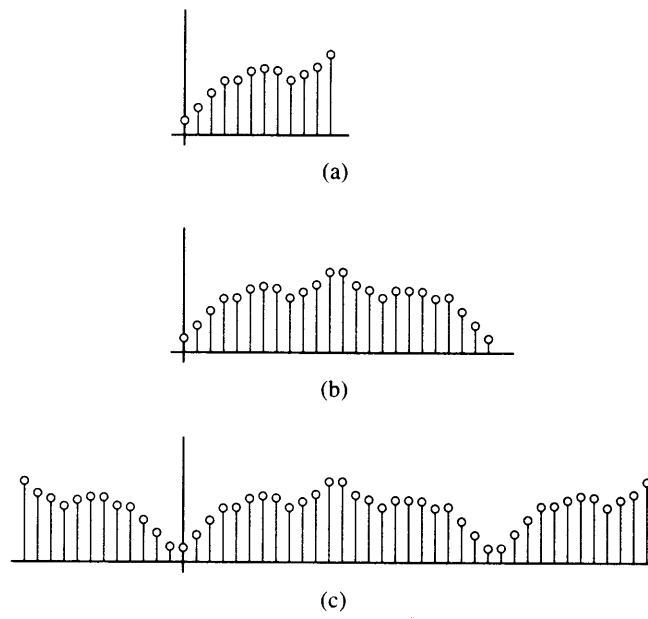
$$[S]_{ij} = \sqrt{\frac{2}{N+1}} \sin \frac{\pi(i+1)(j+1)}{N+1} \quad i, j = 0, 1, \dots, N-1. \quad (13.45)$$

### 13.4.4 Discrete Walsh-Hadamard Transform

A transform that is especially simple to implement is the discrete Walsh-Hadamard transform (DWHT). The DWHT transform matrices are rearrangements of discrete Hadamard matrices, which are of particular importance in coding theory [188]. A Hadamard matrix of order  $N$  is defined as an  $N \times N$  matrix  $H$ , with the property that  $HH^T = NI$ , where  $I$  is the  $N \times N$



**FIGURE 13.5** Taking the discrete Fourier transform of a sequence.



**FIGURE 13.6** Taking the discrete cosine transform of a sequence.

identity matrix. Hadamard matrices whose dimensions are a power of two can be constructed in the following manner:

$$H_{2N} = \begin{bmatrix} H_N & H_N \\ H_N & -H_N \end{bmatrix} \quad (13.46)$$

with  $H_1 = [1]$ . Therefore,

$$H_2 = \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (13.47)$$

$$H_4 = \begin{bmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \quad (13.48)$$

$$H_8 = \begin{bmatrix} H_4 & H_4 \\ H_4 & -H_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \quad (13.49)$$

The DWHT transform matrix  $H$  can be obtained from the Hadamard matrix by multiplying it by a normalizing factor so that  $HH^T = I$  instead of  $NI$ , and by reordering the rows in increasing *sequency* order. The sequency of a row is half the number of sign changes in that row. In  $H_8$  the first row has sequency 0, the second row has sequency 7/2, the third row has sequency 3/2, and so on. Normalization involves multiplying the matrix by  $\frac{1}{\sqrt{N}}$ . Reordering the  $H_8$  matrix in increasing sequency order, we get

$$H = \frac{1}{\sqrt{8}} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix} \quad (13.50)$$

Because the matrix without the scaling factor consists of  $\pm 1$ , the transform operation consists simply of addition and subtraction. For this reason, this transform is useful in situations where minimizing the amount of computations is very important. However, the amount of energy compaction obtained with this transform is substantially less than the compaction obtained by the use of the DCT. Therefore, where sufficient computational power is available, DCT is the transform of choice.

## 13.5 Quantization and Coding of Transform Coefficients

If the amount of information conveyed by each coefficient is different, it makes sense to assign differing numbers of bits to the different coefficients. There are two approaches to assigning bits. One approach relies on the average properties of the transform coefficients, while the other approach assigns bits as needed by individual transform coefficients.

In the first approach, we first obtain an estimate of the variances of the transform coefficients. These estimates can be used by one of two algorithms to assign the number of bits used to quantize each of the coefficients. We assume that the relative variance of the coefficients corresponds to the amount of information contained in each coefficient. Thus, coefficients with higher variance are assigned more bits than coefficients with smaller variance.

Let us find an expression for the distortion, then find the bit allocation that minimizes the distortion. To perform the minimization we will use the method of Lagrange [189]. If the average number of bits per sample to be used by the transform coding system is  $R$ , and the average number of bits per sample used by the  $k$ th coefficient is  $R_k$ , then

$$R = \frac{1}{M} \sum_{k=1}^M R_k \quad (13.51)$$

where  $M$  is the number of transform coefficients. The reconstruction error variance for the  $k$ th quantizer  $\sigma_{r_k}^2$  is related to the  $k$ th quantizer input variance  $\sigma_{\theta_k}^2$  by the following:

$$\sigma_{r_k}^2 = \alpha_k 2^{-2R_k} \sigma_{\theta_k}^2 \quad (13.52)$$

where  $\alpha_k$  is a factor that depends on the input distribution and the quantizer.

The total reconstruction error is given by

$$\sigma_r^2 = \sum_{k=1}^M \alpha_k 2^{-2R_k} \sigma_{\theta_k}^2. \quad (13.53)$$

The objective of the bit allocation procedure is to find  $R_k$  to minimize (13.53) subject to the constraint of (13.51). If we assume that  $\alpha_k$  is a constant  $\alpha$  for all  $k$ , we can set up the minimization problem in terms of Lagrange multipliers as

$$J = \alpha \sum_{k=1}^M 2^{-2R_k} \sigma_{\theta_k}^2 - \lambda \left( R - \frac{1}{M} \sum_{k=1}^M R_k \right). \quad (13.54)$$

Taking the derivative of  $J$  with respect to  $R_k$  and setting it equal to zero, we can obtain this expression for  $R_k$ :

$$R_k = \frac{1}{2} \log_2 (2\alpha \ln 2 \sigma_{\theta_k}^2) - \frac{1}{2} \log_2 \lambda. \quad (13.55)$$

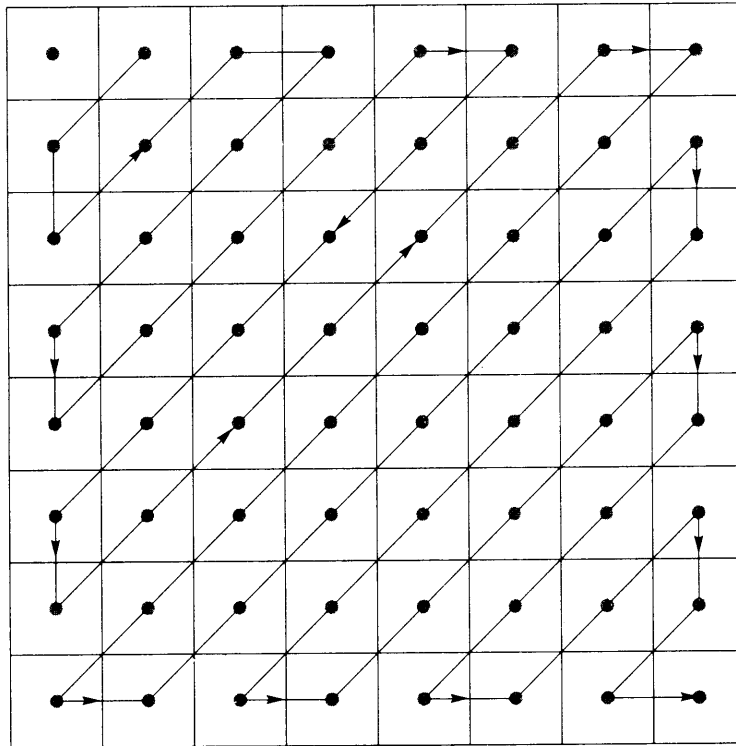
Substituting this expression for  $R_k$  in (13.51), we get a value for  $\lambda$ :

$$\lambda = \prod_{k=1}^M (2\alpha \ln 2 \sigma_{\theta_k}^2)^{\frac{1}{M}} 2^{-2R}. \quad (13.56)$$



no bits assigned to them. This means that the reconstructed image will not contain a very good representation of the edges.

This problem can be avoided by using a different approach to bit allocation known as *threshold coding* [190, 93, 191]. In this approach, which coefficient to keep and which to discard is not decided a priori. In the simplest form of threshold coding, we specify a threshold value. Coefficients with magnitude below this threshold are discarded, while the other coefficients are quantized and transmitted. The information about which coefficients have been retained is sent to the receiver as side information. A simple approach described by Pratt [93] is to code the first coefficient on each line regardless of the magnitude. After this, when we encounter a coefficient with a magnitude above the threshold value, we send two codewords: one for the quantized value of the coefficient, and one for the count of the number of coefficients since the last coefficient with magnitude greater than the threshold. For the two-dimensional case, the block size is usually small, and each "line" of the transform is very short. Thus, this approach would be quite expensive. Chen and Pratt [191] suggest scanning the block of transformed coefficients in a zigzag fashion, as shown in Figure 13.7. If we scan an  $8 \times 8$  block of quantized transform coefficients in this manner, we will find that in general a large section of the tail end of the scan will consist of zeros. This is because



**FIGURE 13.7** The zigzag scanning pattern for an  $8 \times 8$  transform.

generally the higher-order coefficients have smaller amplitude. This is reflected in the bit allocation table shown in Table 13.4. As we shall see later, if we use midread quantizers (quantizers with a zero output level), combined with the fact that the step sizes for the higher-order coefficients are generally chosen to be quite large, this means that many of these coefficients will be quantized to zero. Therefore, there is a high probability that after a few coefficients along the zigzag scan, all coefficients will be zero. In this situation, Chen and Pratt suggest the transmission of a special *end-of-block* (EOB) symbol. Upon reception of the EOB signal, the receiver would automatically set all remaining coefficients along the zigzag scan to zero.

The algorithm developed by the Joint Photographic Experts Group (JPEG), described in the next section, uses a rather clever variation of this approach.

### 13.6 Application to Image Compression—JPEG

The JPEG standard is one of the most widely known standards for lossy image compression. It is a result of the collaboration of the International Standards Organization (ISO), which is a private organization, and what was the CCITT (now ITU-T), a part of the United Nations. The approach recommended by JPEG is a transform coding approach using the DCT. The approach is a modification of the scheme proposed by Chen and Pratt [191]. In this section we will briefly describe the baseline JPEG algorithm. In order to illustrate the various components of the algorithm, we will use an  $8 \times 8$  block of the Sena image, shown in Table 13.5. For more details, see [10].

#### 13.6.1 The Transform

The transform used in the JPEG scheme is the DCT transform described earlier. The input image is first “level shifted” by  $2^{P-1}$ ; that is, we subtract  $2^{P-1}$  from each pixel value, where  $P$  is the number of bits used to represent each pixel. Thus, if we are dealing with 8-bit images whose pixels take on values between 0 and 255, we would subtract 128 from each pixel so that the value of the pixel varies between  $-128$  and 127. The image is divided into blocks of size  $8 \times 8$ , which are then transformed using an  $8 \times 8$  forward DCT. If any dimension of the image is not a multiple of eight, the encoder replicates the last column or row until the

**TABLE 13.5** An  $8 \times 8$  block from the Sena image.

124	125	122	120	122	119	117	118
121	121	120	119	119	120	120	118
126	124	123	122	121	121	120	120
124	124	125	125	126	125	124	124
127	127	128	129	130	128	127	125
143	142	143	142	140	139	139	139
150	148	152	152	152	152	150	151
156	159	158	155	158	158	157	156



**TABLE 13.6** The DCT coefficients corresponding to the block of data from the Sena image after level shift.

39.88	6.56	-2.24	1.22	-0.37	-1.08	0.79	1.13
-102.43	4.56	2.26	1.12	0.35	-0.63	-1.05	-0.48
37.77	1.31	1.77	0.25	-1.50	-2.21	-0.10	0.23
-5.67	2.24	-1.32	-0.81	1.41	0.22	-0.13	0.17
-3.37	-0.74	-1.75	0.77	-0.62	-2.65	-1.30	0.76
5.98	-0.13	-0.45	-0.77	1.99	-0.26	1.46	0.00
3.97	5.52	2.39	-0.55	-0.051	-0.84	-0.52	-0.13
-3.43	0.51	-1.07	0.87	0.96	0.09	0.33	0.01

final size is a multiple of eight. These additional rows or columns are removed during the decoding process. If we take the  $8 \times 8$  block of pixels shown in Table 13.5, subtract 128 from it, and take the DCT of this level-shifted block, we obtain the DCT coefficients shown in Table 13.6. Notice that the lower-frequency coefficients in the top-left corner of the table have larger values than the higher-frequency coefficients. This is generally the case, except for situations in which there is substantial activity in the image block.

### 13.6.2 Quantization

The JPEG algorithm uses uniform midtread quantization to quantize the various coefficients. The quantizer step sizes are organized in a table called the *quantization table* and can be viewed as the fixed part of the quantization. An example of a quantization table from the JPEG recommendation [10] is shown in Table 13.7. Each quantized value is represented by a label. The label corresponding to the quantized value of the transform coefficient  $\theta_{ij}$  is obtained as

$$l_{ij} = \left\lfloor \frac{\theta_{ij}}{Q_{ij}} + 0.5 \right\rfloor \quad (13.58)$$

**TABLE 13.7** Sample quantization table.

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

**TABLE 13.8** The quantizer labels obtained by using the quantization table on the coefficients.

2	1	0	0	0	0	0	0
-9	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

where  $Q_{ij}$  is the  $(i, j)$ th element of the quantization table, and  $\lfloor x \rfloor$  is the largest integer smaller than  $x$ . Consider the  $\theta_{00}$  coefficient from Table 13.6. The value of  $\theta_{00}$  is 39.88. From Table 13.7,  $Q_{00}$  is 16. Therefore,

$$l_{00} = \left\lfloor \frac{39.88}{16} + 0.5 \right\rfloor = \lfloor 2.9925 \rfloor = 2. \quad (13.59)$$

The reconstructed value is obtained from the label by multiplying the label with the corresponding entry in the quantization table. Therefore, the reconstructed value of  $\theta_{00}$  would be  $l_{00} \times Q_{00}$ , which is  $2 \times 16 = 32$ . The quantization error in this case is  $39.88 - 32 = -7.88$ . Similarly, from Tables 13.6 and 13.7,  $\theta_{01}$  is 6.56 and  $Q_{01}$  is 11. Therefore,

$$l_{01} = \left\lfloor \frac{6.56}{11} + 0.5 \right\rfloor = \lfloor 1.096 \rfloor = 1. \quad (13.60)$$

The reconstructed value is 11, and the quantization error is  $11 - 6.56 = 4.44$ . Continuing in this fashion, we obtain the labels shown in Table 13.8.

From the sample quantization table shown in Table 13.7, we can see that the step size generally increases as we move from the DC coefficient to the higher-order coefficients. Because the quantization error is an increasing function of the step size, more quantization error will be introduced in the higher-frequency coefficients than in the lower-frequency coefficients. The decision on the relative size of the step sizes is based on how errors in these coefficients will be perceived by the human visual system. Different coefficients in the transform have widely different perceptual importance. Quantization errors in the DC and lower AC coefficients are more easily detectable than the quantization error in the higher AC coefficients. Therefore, we use larger step sizes for perceptually less important coefficients.

Because the quantizers are all midtread quantizers (that is, they all have a zero output level), the quantization process also functions as the thresholding operation. All coefficients with magnitudes less than half the corresponding step size will be set to zero. Because the step sizes at the tail end of the zigzag scan are larger, the probability of finding a long run of zeros increases at the end of the scan. This is the case for the  $8 \times 8$  block of labels shown in Table 13.8. The entire run of zeros at the tail end of the scan can be coded with an EOB code after the last nonzero label, resulting in substantial compression.

Furthermore, this effect also provides us with a method to vary the rate. By making the step sizes larger, we can reduce the number of nonzero values that need to be transmitted, which translates to a reduction in the number of bits that need to be transmitted.

### 13.6.3 Coding

Chen and Pratt [191] used separate Huffman codes for encoding the label for each coefficient and the number of coefficients since the last nonzero label. The JPEG approach is somewhat more complex but results in higher compression. In the JPEG approach, the labels for the DC and AC coefficients are coded differently.

From Figure 13.4 we can see that the basis matrix corresponding to the DC coefficient is a constant matrix. Thus, the DC coefficient is some multiple of the average value in the  $8 \times 8$  block. The average pixel value in any  $8 \times 8$  block will not differ substantially from the average value in the neighboring  $8 \times 8$  block; therefore, the DC coefficient values will be quite close. Given that the labels are obtained by dividing the coefficients with the corresponding entry in the quantization table, the labels corresponding to these coefficients will be closer still. Therefore, it makes sense to encode the differences between neighboring labels rather than to encode the labels themselves.

Depending on the number of bits used to encode the pixel values, the number of values that the labels, and hence the differences, can take on may become quite large. A Huffman code for such a large alphabet would be quite unmanageable. The JPEG recommendation resolves this problem by partitioning the possible values that the differences can take on into categories. The size of these categories grows as a power of two. Thus, category 0 has only one member (0), category 1 has two members ( $-1$  and  $1$ ), category 2 has four members ( $-3$ ,  $-2$ ,  $2$ ,  $3$ ), and so on. The category numbers are then Huffman coded. The number of codewords in the Huffman code is equal to the base two logarithm of the number of possible values that the label differences can take on. If the differences can take on 4096 possible values, the size of the Huffman code is  $\log_2 4096 = 12$ . The elements within each category are specified by tacking on extra bits to the end of the Huffman code for that category. As the categories are different sizes, we need a differing number of bits to identify the value in each category. For example, because category 0 contains only one element, we need no additional bits to specify the value. Category 1 contains two elements, so we need 1 bit tacked on to the end of the Huffman code for category 1 to specify the particular element in that category. Similarly, we need 2 bits to specify the element in category 2, 3 bits for category 3, and  $n$  bits for category  $n$ .

The categories and the corresponding difference values are shown in Table 13.9. For example, if the difference between two labels was 6, we would send the Huffman code for category 3. As category 3 contains the eight values  $\{-7, -6, -5, -4, 4, 5, 6, 7\}$ , the Huffman code for category 3 would be followed by 3 bits that would specify which of the eight values in category 3 was being transmitted.

The binary code for the AC coefficients is generated in a slightly different manner. The category  $C$  that a nonzero label falls in and the number of zero-valued labels  $Z$  since the last nonzero label form a pointer to a specific Huffman code as shown in Table 13.10. Thus, if the label being encoded falls in category 3, and there have been 15 zero-valued labels prior to this nonzero label in the zigzag scan, then we form the pointer  $F/3$ , which points

**TABLE 13.9 Coding of the differences of the DC labels.**

0			0			
1			-1	1		
2		-3	-2	2	3	
3	-7	...	-4	4	...	7
4	-15	...	-8	8	...	15
5	-31	...	-16	16	...	31
6	-63	...	-32	32	...	63
7	-127	...	-64	64	...	127
8	-255	...	-128	128	...	255
9	-511	...	-256	256	...	511
10	-1,023	...	-512	512	...	1,023
11	-2,047	...	-1,024	1,024	...	2,047
12	-4,095	...	-2,048	2,048	...	4,095
13	-8,191	...	-4,096	4,096	...	8,191
14	-16,383	...	-8,192	8,192	...	16,383
15	-32,767	...	-16,384	16,384	...	32,767
16			32,768			

**TABLE 13.10 Sample table for obtaining the Huffman code for a given label value and run length. The values of Z are represented in hexadecimal.**

Z/C	Codeword	Z/C	Codeword	...	Z/C	Codeword
0/0 (EOB)	1010			...	F/0 (ZRL)	1111111001
0/1	00	1/1	1100	...	F/1	11111111110101
0/2	01	1/2	11011	...	F/2	111111111110110
0/3	100	1/3	1111001	...	F/3	111111111110111
0/4	1011	1/4	111110110	...	F/4	111111111111000
0/5	11010	1/5	11111110110	...	F/5	111111111111001
⋮	⋮	⋮	⋮		⋮	

to the codeword 111111111110111. Because the label falls in category 3, we follow this codeword with 3 bits that indicate which of the eight possible values in category 3 is the value that the label takes on.

There are two special codes shown in Table 13.10. The first is for the end-of-block (EOB). This is used in the same way as in the Chen and Pratt [191] algorithm; that is, if a particular label value is the last nonzero value along the zigzag scan, the code for it is immediately followed by the EOB code. The other code is the ZRL code, which is used when the number of consecutive zero values along the zigzag scan exceeds 15.

To see how all of this fits together, let's encode the labels in Table 13.8. The label corresponding to the DC coefficient is coded by first taking the difference between the value of the quantized label in this block and the quantized label in the previous block. If we assume that the corresponding label in the previous block was -1, then the difference would be 3. From Table 13.9 we can see that this value falls in category 2. Therefore, we

would send the Huffman code for category 2 followed by the 2-bit sequence 11 to indicate that the value in category 2 being encoded was 3, and not  $-3$ ,  $-2$ , or 2. To encode the AC coefficients, we first order them using the zigzag scan. We obtain the sequence

$$1 -9 3 0 0 0 \dots 0$$

The first value, 1, belongs to category 1. Because there are no zeros preceding it, we transmit the Huffman code corresponding to 0/1, which from Table 13.10 is 00. We then follow this by a single bit 1 to indicate that the value being transmitted is 1 and not  $-1$ . Similarly,  $-9$  is the seventh element in category 4. Therefore, we send the binary string 1011, which is the Huffman code for 0/4, followed by 0110 to indicate that  $-9$  is the seventh element in category 4. The next label is 3, which belongs to category 2, so we send the Huffman code 01 corresponding to 0/2, followed by the 2 bits 11. All the labels after this point are 0, so we send the EOB Huffman code, which in this case is 1010. If we assume that the Huffman code for the DC coefficient was 2 bits long, we have sent a grand total of 21 bits to represent this  $8 \times 8$  block. This translates to an average  $\frac{21}{64}$  bits per pixel.

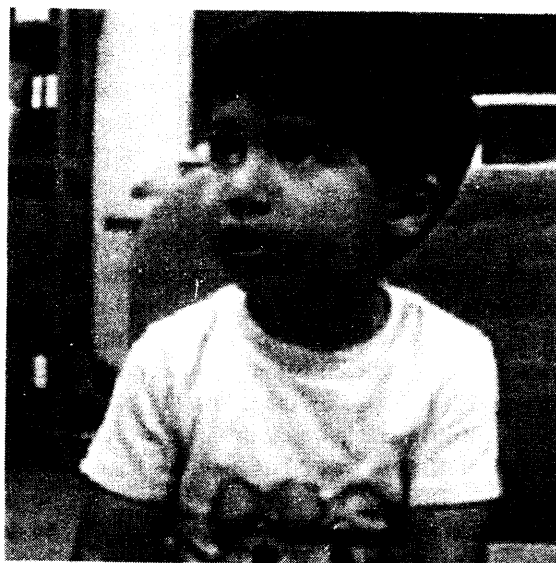
To obtain a reconstruction of the original block, we perform the dequantization, which simply consists of multiplying the labels in Table 13.8 with the corresponding values in Table 13.7. Taking the inverse transform of the quantized coefficients shown in Table 13.11 and adding 128, we get the reconstructed block shown in Table 13.12. We can see that in spite of going from 8 bits per pixel to  $\frac{9}{32}$  bits per pixel, the reproduction is remarkably close to the original.

**TABLE 13.11** The quantized values of the coefficients.

32	11	0	0	0	0	0	0
-108	0	0	0	0	0	0	0
42	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

**TABLE 13.12** The reconstructed block.

123	122	122	121	120	120	119	119
121	121	121	120	119	118	118	118
121	121	120	119	119	118	117	117
124	124	123	122	122	121	120	120
130	130	129	129	128	128	128	127
141	141	140	140	139	138	138	137
152	152	151	151	150	149	149	148
159	159	158	157	157	156	155	155



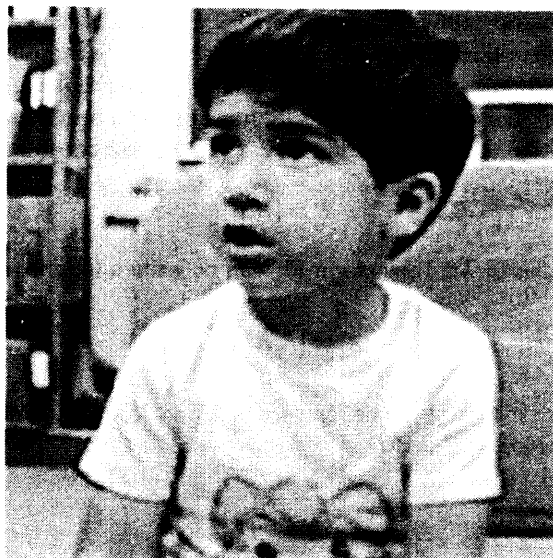
**FIGURE 13.8** Sinan image coded at 0.5 bits per pixel using the JPEG algorithm.

If we wanted an even more accurate reproduction, we could do so at the cost of increased bit rate by multiplying the step sizes in the quantization table by one-half and using these values as the new step sizes. Using the same assumptions as before, we can show that this will result in an increase in the number of bits transmitted. We can go in the other direction by multiplying the step sizes with a number greater than one. This will result in a reduction in bit rate at the cost of increased distortion.

Finally, we present some examples of JPEG-coded images in Figures 13.8 and 13.9. These were coded using shareware generated by the Independent JPEG Group (organizer, Dr. Thomas G. Lane). Notice the high degree of “blockiness” in the lower-rate image (Figure 13.8). This is a standard problem of most block-based techniques, and specifically of the transform coding approach. A number of solutions have been suggested for removing this blockiness, including postfiltering at the block edges as well as transforms that overlap the block boundaries. Each approach has its own drawbacks. The filtering approaches tend to reduce the resolution of the reconstructions, while the overlapped approaches increase the complexity. One particular overlapped approach that is widely used in audio compression is the modified DCT (MDCT), which is described in the next section.

### **13.7 Application to Audio Compression—The MDCT**

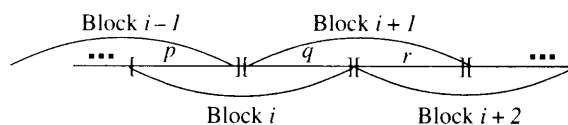
As mentioned in the previous section, the use of the block based transform has the unfortunate effect of causing distortion at the block boundaries at low rates. A number of techniques that use overlapping blocks have been developed over the years [192]. One that has gained



**FIGURE 13.9** Sinan image coded at 0.25 bits per pixel using the JPEG algorithm.

wide acceptance in audio compression is a transform based on the discrete cosine transform called the modified discrete cosine transform (MDCT). It is used in almost all popular audio coding standards from *mp3* and AAC to Ogg Vorbis.

The MDCT used in these algorithms uses 50% overlap. That is, each block overlaps half of the previous block and half of the next block of data. Consequently, each audio sample is part of two blocks. If we were to keep all the frequency coefficients we would end up with twice as many coefficients as samples. Reducing the number of frequency coefficients results in the introduction of distortion in the inverse transform. The distortion is referred to as time domain aliasing [193]. The reason for the name is evident if we consider that the distortion is being introduced by subsampling in the frequency domain. Recall that sampling at less than the Nyquist frequency in the time domain leads to an overlap of replicas of the frequency spectrum, or frequency aliasing. The lapped transforms are successful because they are constructed in such a way that while the inverse transform of each block results in time-domain aliasing, the aliasing in consecutive blocks cancel each other out.



**FIGURE 13.10** Source output sequence.

Consider the scenario shown in Figure 13.10. Let's look at the coding for block  $i$  and block  $i + 1$ . The inverse transform of the coefficients resulting from both these blocks will result in the audio samples in the subblock  $q$ . We assume that the blocksize is  $N$  and therefore the subblock size is  $N/2$ . The forward transform can be represented by an  $N/2 \times N$  matrix  $P$ . Let us partition the matrix into two  $N/2 \times N/2$  blocks,  $A$  and  $B$ . Thus

$$P = [A|B]$$

Let  $x_i = [p|q]$ , then the forward transform  $Px_i$  can be written in terms of the subblocks as

$$X_i = [A|B] \begin{bmatrix} p \\ q \end{bmatrix}$$

The inverse transform matrix  $Q$  can be represented by an  $N \times N/2$ , which can be partitioned into two  $N/2 \times N/2$  blocks,  $C$  and  $D$ .

$$Q = \begin{bmatrix} C \\ D \end{bmatrix}$$

Applying the inverse transform, we get the reconstruction values  $\hat{x}$

$$\hat{x}_i = QX_i = QPx_i = \begin{bmatrix} C \\ D \end{bmatrix} [A|B] \begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} CAp + CBq \\ DAp + DBq \end{bmatrix}$$

Repeating the process for block  $i + 1$  we get

$$\hat{x}_{i+1} = QX_{i+1} = QPx_{i+1} = \begin{bmatrix} C \\ D \end{bmatrix} [A|B] \begin{bmatrix} q \\ r \end{bmatrix} = \begin{bmatrix} CAq + CBr \\ DAq + DBr \end{bmatrix}$$

To cancel out the aliasing in the second half of the block we need

$$CAq + CBr + DAp + DBq = q$$

From this we can get the requirements on the transform

$$CB = 0 \quad (13.61)$$

$$DA = 0 \quad (13.62)$$

$$CA + DB = I \quad (13.63)$$

Note that the same requirements will help cancel the aliasing in the first half of block  $i$  by using the second half of the inverse transform of block  $i - 1$ . One selection that satisfies the last condition is

$$CA = \frac{1}{2}(I - J) \quad (13.64)$$

$$DB = \frac{1}{2}(I + J) \quad (13.65)$$



The forward modified discrete transform is given by the following equation:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left(\frac{2\pi}{N}\left(k + \frac{1}{2}\right)\left(n + \frac{1}{2} + \frac{N}{4}\right)\right) \quad (13.66)$$

where  $x_n$  are the audio samples and  $X_k$  are the frequency coefficients. The inverse MDCT is given by

$$y_n = \frac{2}{N} \sum_{k=0}^{\frac{N}{2}-1} X_k \cos\left(\frac{2\pi}{N}\left(k + \frac{1}{2}\right)\left(n + \frac{1}{2} + \frac{N}{4}\right)\right) \quad (13.67)$$

or in terms of our matrix notation,

$$[P]_{i,j} = \cos\left(\frac{2\pi}{N}\left(i + \frac{1}{2}\right)\left(j + \frac{1}{2} + \frac{N}{4}\right)\right) \quad (13.68)$$

$$[Q]_{i,j} = \frac{2}{N} \cos\left(\frac{2\pi}{N}\left(i + \frac{1}{2}\right)\left(j + \frac{1}{2} + \frac{N}{4}\right)\right) \quad (13.69)$$

It is easy to verify that, given a value of  $N$ , these matrices satisfy the conditions for alias cancellation.

Thus, while the inverse transform for any one block will contain aliasing, by using the inverse transform of neighboring blocks the aliasing can be canceled. What about blocks that do not have neighbors—that is, the first and last blocks? One way to resolve this problem is to pad the sampled audio sequence with  $N/2$  zeros at the beginning and end of the sequence. In practice, this is not necessary, because the data to be transformed is windowed prior to the transform. For the first and last blocks we use a special window that has the same effect as introducing zeros. For information on the design of windows for the MDCT, see [194]. For more on how the MDCT is used in audio compression techniques, see Chapter 16.

## 13.8 Summary

In this chapter we have described the concept of transform coding and provided some of the details needed for the investigation of this compression scheme. The basic encoding scheme works as follows:

- Divide the source output into blocks. In the case of speech or audio data, they will be one-dimensional blocks. In the case of images, they will be two-dimensional blocks. In image coding, a typical block size is  $8 \times 8$ . In audio coding the blocks are generally overlapped by 50%.
- Take the transform of this block. In the case of one-dimensional data, this involves pre-multiplying the  $N$  vector of source output samples by the transform matrix. In the case of image data, for the transforms we have looked at, this involves pre-multiplying the  $N \times N$  block by the transform matrix and post-multiplying the result with the

transpose of the transform matrix. Fast algorithms exist for performing the transforms described in this chapter (see [195]).

- Quantize the coefficients. Various techniques exist for the quantization of these coefficients. We have described the approach used by JPEG. In Chapter 16 we describe the quantization techniques used in various audio coding algorithms.
- Encode the quantized value. The quantized value can be encoded using a fixed-length code or any of the different variable-length codes described in earlier chapters. We have described the approach taken by JPEG.

The decoding scheme is the inverse of the encoding scheme for image compression. For the overlapped transform used in audio coding the decoder adds the overlapped portions of the inverse transform to cancel aliasing.

The basic approach can be modified depending on the particular characteristics of the data. We have described some of the modifications used by various commercial algorithms for transform coding of audio signals.

### Further Reading

1. For detailed information about the JPEG standard, *JPEG Still Image Data Compression Standard*, by W.B. Pennebaker and J.L. Mitchell [10], is an invaluable reference. This book also contains the entire text of the official draft JPEG recommendation, ISO DIS 10918-1 and ISO DIS 10918-2.
2. For a detailed discussion of the MDCT and how it is used in audio coding, an excellent source is *Introduction to Digital Audio Coding Standards*, by M. Bosi and R.E. Goldberg [194]
3. Chapter 12 in *Digital Coding of Waveforms*, by N.S. Jayant and P. Noll [123], provides a more mathematical treatment of the subject of transform coding.
4. A good source for information about transforms is *Fundamentals of Digital Image Processing*, by A.K. Jain [196]. Another one is *Digital Image Processing*, by R.C. Gonzales and R.E. Wood [96]. This book has an especially nice discussion of the Hotelling transform.
5. The bit allocation problem and its solutions are described in *Vector Quantization and Signal Compression*, by A. Gersho and R.M. Gray [5].
6. A very readable description of transform coding of images is presented in *Digital Image Compression Techniques*, by M. Rabbani and P.W. Jones [80].
7. *The Data Compression Book*, by M. Nelson and J.-L. Gailly [60], provides a very readable discussion of the JPEG algorithm.

### 13.9 Projects and Problems

1. A square matrix  $\mathbf{A}$  has the property that  $\mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T = \mathbf{I}$ , where  $\mathbf{I}$  is the identity matrix. If  $X_1$  and  $X_2$  are two  $N$ -dimensional vectors and

$$\Theta_1 = \mathbf{A} X_1$$

$$\Theta_2 = \mathbf{A} X_2$$

then show that

$$|X_1 - X_2|^2 = |\Theta_1 - \Theta_2|^2 \quad (13.70)$$

2. Consider the following sequence of values:

$$\begin{array}{cccccccc} 10 & 11 & 12 & 11 & 12 & 13 & 12 & 11 \\ 10 & -10 & 8 & -7 & 8 & -8 & 7 & -7 \end{array}$$

- (a) Transform each row separately using an eight-point DCT. Plot the resulting 16 transform coefficients.
- (b) Combine all 16 numbers into a single vector and transform it using a 16-point DCT. Plot the 16 transform coefficients.
- (c) Compare the results of (a) and (b). For this particular case would you suggest a block size of 8 or 16 for greater compression? Justify your answer.
3. Consider the following “image”:

$$\begin{array}{cccc} 4 & 3 & 2 & 1 \\ 3 & 2 & 1 & 1 \\ 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array}$$

- (a) Obtain the two-dimensional DWHT transform by first taking the one-dimensional transform of the rows, then taking the column-by-column transform of the resulting matrix.
- (b) Obtain the two-dimensional DWHT transform by first taking the one-dimensional transform of the columns, then taking the row-by-row transform of the resulting matrix.
- (c) Compare and comment on the results of (a) and (b).
4. (This problem was suggested by P.F. Swazek.) Let us compare the energy compaction properties of the DCT and the DWHT transforms.
- (a) For the Sena image, compute the mean squared value of each of the 64 coefficients using the DCT. Plot these values.
- (b) For the Sena image, compute the mean squared value of each of the 64 coefficients using the DWHT. Plot these values.
- (c) Compare the results of (a) and (b). Which transform provides more energy compaction? Justify your answer.

5. Implement the transform and quantization portions of the JPEG standard. For coding the labels use an arithmetic coder instead of the modified Huffman code described in this chapter.
  - (a) Encode the Sena image using this transform coder at rates of (approximately) 0.25, 0.5, and 0.75 bits per pixel. Compute the mean squared error at each rate and plot the rate versus the mse.
  - (b) Repeat part (a) using one of the public domain implementations of JPEG.
  - (c) Compare the plots obtained using the two coders and comment on the relative performance of the coders.
6. One of the extensions to the JPEG standard allows for the use of multiple quantization matrices. Investigate the issues involved in designing a set of quantization matrices. Should the quantization matrices be similar or dissimilar? How would you measure their similarity? Given a particular block, do you need to quantize it with each quantization matrix to select the best? Or is there a computationally more efficient approach? Describe your findings in a report.

# Subband Coding

## 14.1 Overview

**I**n this chapter we present the second of three approaches to compression in which the source output is decomposed into constituent parts. Each constituent part is encoded using one or more of the methods that have been described previously. The approach described in this chapter, known as subband coding, relies on separating the source output into different bands of frequencies using digital filters. We provide a general description of the subband coding system and, for those readers with some knowledge of Z-transforms, a more mathematical analysis of the system. The sections containing the mathematical analysis are not essential to understanding the rest of the chapter and are marked with a  $\star$ . If you are not interested in the mathematical analysis, you should skip these sections. This is followed by a description of a popular approach to bit allocation. We conclude the chapter with applications to audio and image compression.

## 14.2 Introduction

In previous chapters we looked at a number of different compression schemes. Each of these schemes is most efficient when the data have certain characteristics. A vector quantization scheme is most effective if blocks of the source output show a high degree of clustering. A differential encoding scheme is most effective when the sample-to-sample difference is small. If the source output is truly random, it is best to use scalar quantization or lattice vector quantization. Thus, if a source exhibited certain well-defined characteristics, we could choose a compression scheme most suited to that characteristic. Unfortunately, most source outputs exhibit a combination of characteristics, which makes it difficult to select a compression scheme exactly suited to the source output.

In the last chapter we looked at techniques for decomposing the source output into different frequency bands using block transforms. The transform coefficients had differing statistics and differing perceptual importance. We made use of these differences in allocating bits for encoding the different coefficients. This variable bit allocation resulted in a decrease in the average number of bits required to encode the source output. One of the drawbacks of transform coding is the artificial division of the source output into blocks, which results in the generation of coding artifacts at the block edges, or blocking. One approach to avoiding this blocking is the lapped orthogonal transform (LOT) [192]. In this chapter we look at a popular approach to decomposing the image into different frequency bands without the imposition of an arbitrary block structure. After the input has been decomposed into its constituents, we can use the coding technique best suited to each constituent to improve compression performance. Furthermore, each component of the source output may have different perceptual characteristics. For example, quantization error that is perceptually objectionable in one component may be acceptable in a different component of the source output. Therefore, a coarser quantizer that uses fewer bits can be used to encode the component that is perceptually less important.

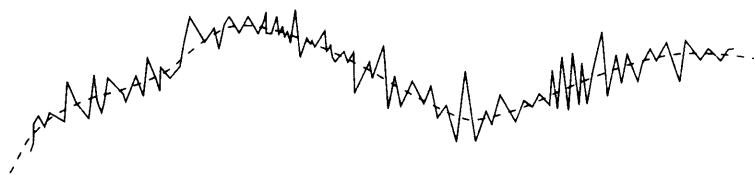
Consider the sequence  $\{x_n\}$  plotted in Figure 14.1. We can see that, while there is a significant amount of sample-to-sample variation, there is also an underlying long-term trend shown by the dotted line that varies slowly.

One way to extract this trend is to average the sample values in a moving window. The averaging operation smooths out the rapid variations, making the slow variations more evident. Let's pick a window of size two and generate a new sequence  $\{y_n\}$  by averaging neighboring values of  $x_n$ :

$$y_n = \frac{x_n + x_{n-1}}{2}. \quad (14.1)$$

The consecutive values of  $y_n$  will be closer to each other than the consecutive values of  $x_n$ . Therefore, the sequence  $\{y_n\}$  can be coded more efficiently using differential encoding than we could encode the sequence  $\{x_n\}$ . However, we want to encode the sequence  $\{x_n\}$ , not the sequence  $\{y_n\}$ . Therefore, we follow the encoding of the averaged sequence  $\{y_n\}$  by the difference sequence  $\{z_n\}$ :

$$z_n = x_n - y_n = x_n - \frac{x_n + x_{n-1}}{2} = \frac{x_n - x_{n-1}}{2}. \quad (14.2)$$



**FIGURE 14.1** A rapidly changing source output that contains a long-term component with slow variations.

The sequences  $\{y_n\}$  and  $\{z_n\}$  can be coded independently of each other. This way we can use the compression schemes that are best suited for each sequence.

**Example 14.2.1:**

Suppose we want to encode the following sequence of values  $\{x_n\}$ :

10 14 10 12 14 8 14 12 10 8 10 12

There is a significant amount of sample-to-sample correlation, so we might consider using a DPCM scheme to compress this sequence. In order to get an idea of the requirements on the quantizer in a DPCM scheme, let us take a look at the sample-to-sample differences  $x_n - x_{n-1}$ :

10 4 -4 2 2 -6 6 -2 -2 -2 2 2

Ignoring the first value, the dynamic range of the differences is from  $-6$  to  $6$ . Suppose we want to quantize these values using  $m$  bits per sample. This means we could use a quantizer with  $M = 2^m$  levels or reconstruction values. If we choose a uniform quantizer, the size of each quantization interval,  $\Delta$ , is the range of possible input values divided by the total number of reconstruction values. Therefore,

$$\Delta = \frac{12}{M}$$

which would give us a maximum quantization error of  $\frac{\Delta}{2}$  or  $\frac{6}{M}$ .

Now let's generate two new sequences  $\{y_n\}$  and  $\{z_n\}$  according to (14.1) and (14.2). All three sequences are plotted in Figure 14.2. Notice that given  $y_n$  and  $z_n$ , we can always recover  $x_n$ :

$$x_n = y_n + z_n. \quad (14.3)$$

Let's try to encode each of these sequences. The sequence  $\{y_n\}$  is

10 12 12 11 13 11 11 13 11 10 9 11

Notice that the  $\{y_n\}$  sequence is "smoother" than the  $\{x_n\}$  sequence—the sample-to-sample variation is much smaller. This becomes evident when we look at the sample-to-sample differences:

10 2 0 -1 2 -2 0 2 -2 -1 -1 2

The difference sequences  $\{x_n - x_{n-1}\}$  and  $\{y_n - y_{n-1}\}$  are plotted in Figure 14.3. Again, ignoring the first difference, the dynamic range of the differences  $y_n - y_{n-1}$  is  $4$ . If we take the dynamic range of these differences as a measure of the range of the quantizer, then for an  $M$ -level quantizer, the step size of the quantizer is  $\frac{4}{M}$  and the maximum quantization

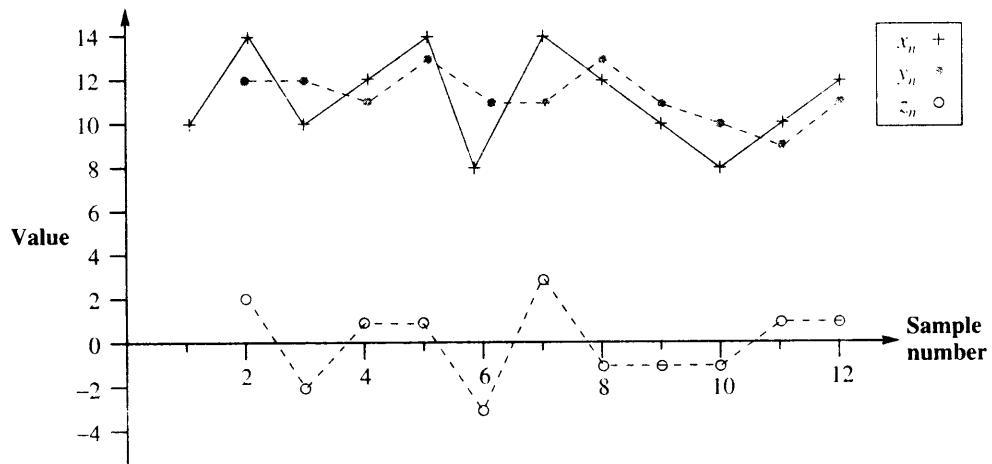


FIGURE 14.2 Original set of samples and the two components.

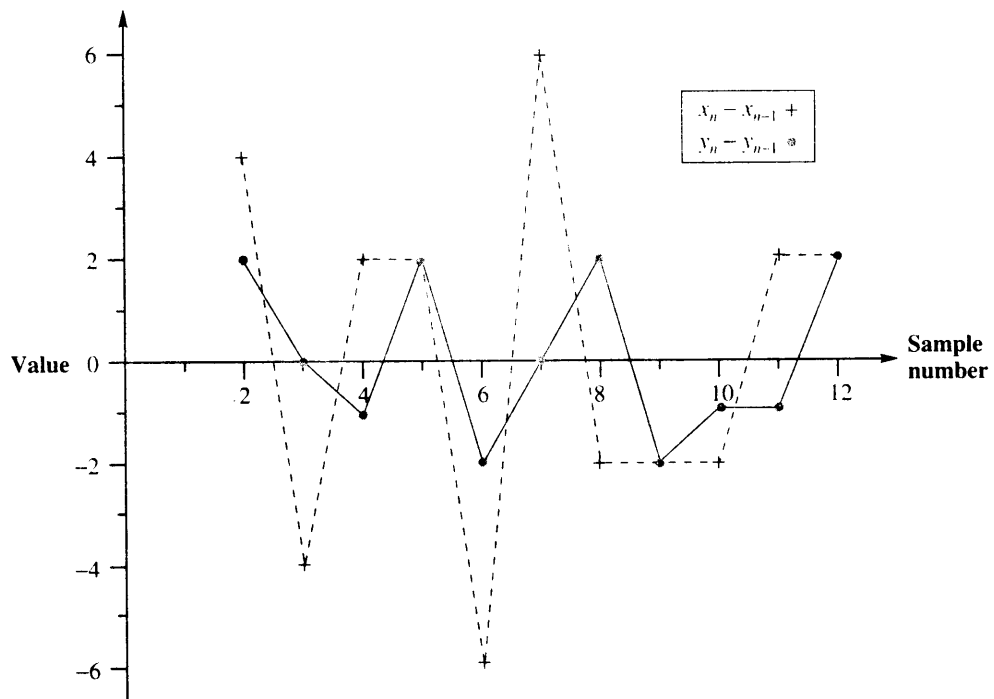


FIGURE 14.3 Difference sequences generated from the original and averaged sequences.



error is  $\frac{2}{M}$ . This maximum quantization error is one-third the maximum quantization error incurred when the  $\{x_n\}$  sequence is quantized using an  $M$ -level quantizer. However, in order to reconstruct  $\{x_n\}$ , we also need to transmit  $\{z_n\}$ . The  $\{z_n\}$  sequence is

$$0 \quad 2 \quad -2 \quad 1 \quad 1 \quad -3 \quad 3 \quad -1 \quad -1 \quad -1 \quad 1 \quad 1$$

The dynamic range for  $z_n$  is 6, half the dynamic range of the difference sequence for  $\{x_n\}$ . (We could have inferred this directly from the definition of  $z_n$ .) The sample-to-sample difference varies more than the actual values. Therefore, instead of differentially encoding this sequence, we quantize each individual sample. For an  $M$ -level quantizer, the required step size would be  $\frac{6}{M}$ , giving a maximum quantization error of  $\frac{3}{M}$ .

For the same number of bits per sample, we can code both  $y_n$  and  $z_n$  and incur less distortion. At the receiver, we add  $y_n$  and  $z_n$  to get the original sequence  $x_n$  back. The maximum possible quantization error in the reconstructed sequence would be  $\frac{5}{M}$ , which is less than the maximum error we would incur if we encoded the  $\{x_n\}$  sequence directly.

Although we use the same number of bits for each value of  $y_n$  and  $z_n$ , the number of elements in each of the  $\{y_n\}$  and  $\{z_n\}$  sequences is the same as the number of elements in the original  $\{x_n\}$  sequence. Although we are using the same number of bits per sample, we are transmitting twice as many samples and, in effect, doubling the bit rate.

We can avoid this by sending every other value of  $y_n$  and  $z_n$ . Let's divide the sequence  $\{y_n\}$  into subsequences  $\{y_{2n}\}$  and  $\{y_{2n-1}\}$ —that is, a subsequence containing only the odd-numbered elements  $\{y_1, y_3, \dots\}$ , and a subsequence containing only the even-numbered elements  $\{y_2, y_4, \dots\}$ . Similarly, we divide the  $\{z_n\}$  sequence into subsequences  $\{z_{2n}\}$  and  $\{z_{2n-1}\}$ . If we transmit either the even-numbered subsequences or the odd-numbered subsequences, we would transmit only as many elements as in the original sequence. To see how we recover the sequence  $\{x_n\}$  from these subsequences, suppose we only transmitted the subsequences  $\{y_{2n}\}$  and  $\{z_{2n}\}$ :

$$y_{2n} = \frac{x_{2n} + x_{2n-1}}{2}$$

$$z_{2n} = \frac{x_{2n} - x_{2n-1}}{2}$$

To recover the even-numbered elements of the  $\{x_n\}$  sequence, we add the two subsequences. In order to obtain the odd-numbered members of the  $\{x_n\}$  sequence, we take the difference:

$$y_{2n} + z_{2n} = x_{2n} \tag{14.4}$$

$$y_{2n} - z_{2n} = x_{2n-1} \tag{14.5}$$

Thus, we can recover the entire original sequence  $\{x_n\}$ , sending only as many bits as required to transmit the original sequence while incurring less distortion.

Is the last part of the previous statement still true? In our original scheme we proposed to transmit the sequence  $\{y_n\}$  by transmitting the differences  $y_n - y_{n-1}$ . As we now need to transmit the subsequence  $\{y_{2n}\}$ , we will be transmitting the differences  $y_{2n} - y_{2n-2}$  instead. In order for our original statement about reduction in distortion to hold, the dynamic range

of this new sequence of differences should be less than or equal to the dynamic range of the original difference. A quick check of the  $\{y_n\}$  shows us that the dynamic range of the new differences is still 4, and our claim of incurring less distortion still holds. ♦

There are several things we can see from this example. First, the number of different values that we transmit is the same, whether we send the original sequence  $\{x_n\}$  or the two subsequences  $\{y_n\}$  and  $\{z_n\}$ . Decomposing the  $\{x_n\}$  sequence into subsequences did not result in any increase in the number of values that we need to transmit. Second, the two subsequences had distinctly different characteristics, which led to our use of different techniques to encode the different sequences. If we had not split the  $\{x_n\}$  sequence, we would have been using essentially the same approach to compress both subsequences. Finally, we could have used the same decomposition approach to decompose the two constituent sequences, which then could be decomposed further still.

While this example was specific to a particular set of values, we can see that decomposing a signal can lead to different ways of looking at the problem of compression. This added flexibility can lead to improved compression performance.

Before we leave this example let us formalize the process of decomposing or *analysis*, and recomposing or *synthesis*. In our example, we decomposed the input sequence  $\{x_n\}$  into two subsequences  $\{y_n\}$  and  $\{z_n\}$  by the operations

$$y_n = \frac{x_n + x_{n-1}}{2} \quad (14.6)$$

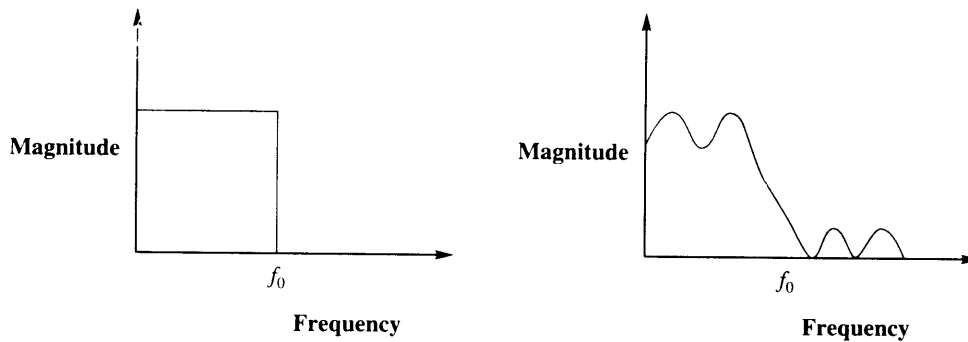
$$z_n = \frac{x_n - x_{n-1}}{2}. \quad (14.7)$$

We can implement these operations using discrete time filters. We briefly considered discrete time filters in Chapter 12. We take a slightly more detailed look at filters in the next section.

### 14.3 Filters

A system that isolates certain frequency components is called a *filter*. The analogy here with mechanical filters such as coffee filters is obvious. A coffee filter or a filter in a water purification system blocks coarse particles and allows only the finer-grained components of the input to pass through. The analogy is not complete, however, because mechanical filters always block the coarser components of the input, while the filters we are discussing can selectively let through or block any range of frequencies. Filters that only let through components below a certain frequency  $f_0$  are called low-pass filters; filters that block all frequency components below a certain value  $f_0$  are called high-pass filters. The frequency  $f_0$  is called the *cutoff frequency*. Filters that let through components that have frequency content above some frequency  $f_1$  but below frequency  $f_2$  are called band-pass filters.

One way to characterize filters is by their *magnitude transfer function*—the ratio of the magnitude of the input and output of the filter as a function of frequency. In Figure 14.4 we show the magnitude transfer function for an ideal low-pass filter and a more realistic low-pass filter, both with a cutoff frequency of  $f_0$ . In the ideal case, all components of the input signal with frequencies below  $f_0$  are unaffected except for a constant amount of



**FIGURE 14.4** Ideal and realistic low-pass filter characteristics.

amplification. All frequencies above  $f_0$  are blocked. In other words, the cutoff is sharp. In the case of the more realistic filter, the cutoff is more gradual. Also, the amplification for the components with frequency less than  $f_0$  is not constant, and components with frequencies above  $f_0$  are not totally blocked. This phenomenon is referred to as *ripple* in the passband and stopband.

The filters we will discuss are digital filters, which operate on a sequence of numbers that are usually samples of a continuously varying signal. We have discussed sampling in Chapter 12. For those of you who skipped that chapter, let us take a brief look at the sampling operation.

How often does a signal have to be sampled in order to reconstruct the signal from the samples? If one signal changes more rapidly than another, it is reasonable to assume that we would need to sample the more rapidly varying signal more often than the slowly varying signal in order to achieve an accurate representation. In fact, it can be shown mathematically that if the highest frequency component of a signal is  $f_0$ , then we need to sample the signal at more than  $2f_0$  times per second. This result is known as the *Nyquist theorem* or *Nyquist rule* after Harry Nyquist, a famous mathematician from Bell Laboratories. His pioneering work laid the groundwork for much of digital communication. The Nyquist rule can also be extended to signals that only have frequency components between two frequencies  $f_1$  and  $f_2$ . If  $f_1$  and  $f_2$  satisfy certain criteria, then we can show that in order to recover the signal exactly, we need to sample the signal at a rate of at least  $2(f_2 - f_1)$  samples per second [123].

What would happen if we violated the Nyquist rule and sampled at less than twice the highest frequency? In Chapter 12 we showed that it would be impossible to recover the original signal from the sample. Components with frequencies higher than half the sampling rate show up at lower frequencies. This process is called *aliasing*. In order to prevent aliasing, most systems that require sampling will contain an “anti-aliasing filter” that restricts the input to the sampler to be less than half the sampling frequency. If the signal contains components at more than half the sampling frequency, we will introduce distortion by filtering out these components. However, the distortion due to aliasing is generally more severe than the distortion we introduce due to filtering.

Digital filtering involves taking a weighted sum of current and past inputs to the filter and, in some cases, the past outputs of the filter. The general form of the input-output relationships of the filter is given by

$$y_n = \sum_{i=0}^N a_i x_{n-i} + \sum_{i=1}^M b_i y_{n-i} \quad (14.8)$$

where the sequence  $\{x_n\}$  is the input to the filter, the sequence  $\{y_n\}$  is the output from the filter, and the values  $\{a_i\}$  and  $\{b_i\}$  are called the *filter coefficients*.

If the input sequence is a single 1 followed by all 0s, the output sequence is called the impulse response of the filter. Notice that if the  $b_i$  are all 0, then the impulse response will die out after  $N$  samples. These filters are called *finite impulse response (FIR)* filters. The number  $N$  is sometimes called the number of *taps* in the filter. If any of the  $b_i$  have nonzero values, the impulse response can, in theory, continue forever. Filters with nonzero values for some of the  $b_i$  are called *infinite impulse response (IIR)* filters.

### Example 14.3.1:

Suppose we have a filter with  $a_0 = 1.25$  and  $a_1 = 0.5$ . If the input sequence  $\{x_n\}$  is given by

$$x_n = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0, \end{cases} \quad (14.9)$$

then the output is given by

$$\begin{aligned} y_0 &= a_0 x_0 + a_1 x_{-1} = 1.25 \\ y_1 &= a_0 x_1 + a_1 x_0 = 0.5 \\ y_n &= 0 \quad n < 0 \text{ or } n > 1. \end{aligned}$$

This output is called the impulse response of the filter. The impulse response sequence is usually represented by  $\{h_n\}$ . Therefore, for this filter we would say that

$$h_n = \begin{cases} 1.25 & n = 0 \\ 0.5 & n = 1 \\ 0 & \text{otherwise.} \end{cases} \quad (14.10)$$

Notice that if we know the impulse response we also know the values of  $a_i$ . Knowledge of the impulse response completely specifies the filter. Furthermore, because the impulse response goes to zero after a finite number of samples (two in this case), the filter is an FIR filter.

The filters we used in Example 14.2.1 are both two-tap FIR filters with impulse responses

$$h_n = \begin{cases} \frac{1}{2} & n = 0 \\ \frac{1}{2} & n = 1 \\ 0 & \text{otherwise} \end{cases} \quad (14.11)$$

for the “averaging” or low-pass filter, and

$$h_n = \begin{cases} \frac{1}{2} & n = 0 \\ -\frac{1}{2} & n = 1 \\ 0 & \text{otherwise} \end{cases} \quad (14.12)$$

for the “difference” or high-pass filter.

Now let’s consider a different filter with  $a_0 = 1$  and  $b_1 = 0.9$ . For the same input as above, the output is given by

$$y_0 = a_0 x_0 + b_1 y_{-1} = 1(1) + 0.9(0) = 1 \quad (14.13)$$

$$y_1 = a_0 x_1 + b_1 y_0 = 1(0) + 0.9(1) = 0.9 \quad (14.14)$$

$$y_2 = a_0 x_2 + b_1 y_1 = 1(0) + 0.9(0.9) = 0.81 \quad (14.15)$$

$\vdots$   $\vdots$

$$y_n = (0.9)^n. \quad (14.16)$$

The impulse response can be written more compactly as

$$h_n = \begin{cases} 0 & n < 0 \\ (0.9)^n & n \geq 0. \end{cases} \quad (14.17)$$

Notice that the impulse response is nonzero for all  $n \geq 0$ , which makes this an IIR filter.  $\blacklozenge$

Although it is not as clear in the IIR case as it was in the FIR case, the impulse response completely specifies the filter. Once we know the impulse response of the filter, we know the relationship between the input and output of the filter. If  $\{x_n\}$  and  $\{y_n\}$  are the input and output, respectively, of a filter with impulse response  $\{h_n\}_{n=0}^M$ , then  $\{y_n\}$  can be obtained from  $\{x_n\}$  and  $\{h_n\}$  via the following relationship:

$$y_n = \sum_{k=0}^M h_k x_{n-k}, \quad (14.18)$$

where  $M$  is finite for an FIR filter and infinite for an IIR filter. The relationship, shown in (14.18), is known as *convolution* and can be easily obtained through the use of the properties of linearity and shift invariance (see Problem 1).

Because FIR filters are simply weighted averages, they are always stable. When we say a filter is stable we mean that as long as the input is bounded, the output will also be bounded. This is not true of IIR filters. Certain IIR filters can give an unbounded output even when the input is bounded.

**Example 14.3.2:**

Consider a filter with  $a_0 = 1$  and  $b_1 = 2$ . Suppose the input sequence is a single 1 followed by 0s. Then the output is

$$y_0 = a_0x_0 + b_1y_{-1} = 1(1) + 2(0) = 1 \quad (14.19)$$

$$y_1 = a_0x_1 + b_1y_0 = 1(0) + 2(1) = 2 \quad (14.20)$$

$$y_2 = a_0x_2 + b_1y_1 = 1(0) + 2(2) = 4 \quad (14.21)$$

$$\vdots \quad \vdots$$

$$y_n = 2^n. \quad (14.22)$$

Even though the input contained a single 1, the output at time  $n = 30$  is  $2^{30}$ , or more than a billion!  $\blacklozenge$

Although IIR filters can become unstable, they can also provide better performance, in terms of sharper cutoffs and less ripple in the passband and stopband for a fewer number of coefficients.

The study of design and analysis of digital filters is a fascinating and important subject. We provide some of the details in Sections 14.5–14.8. If you are not interested in these topics, you can take a more utilitarian approach and make use of the literature to select the necessary filters rather than design them. In the following section we briefly describe some of the families of filters used to generate the examples in this chapter. We also provide filter coefficients that you can use for experiment.

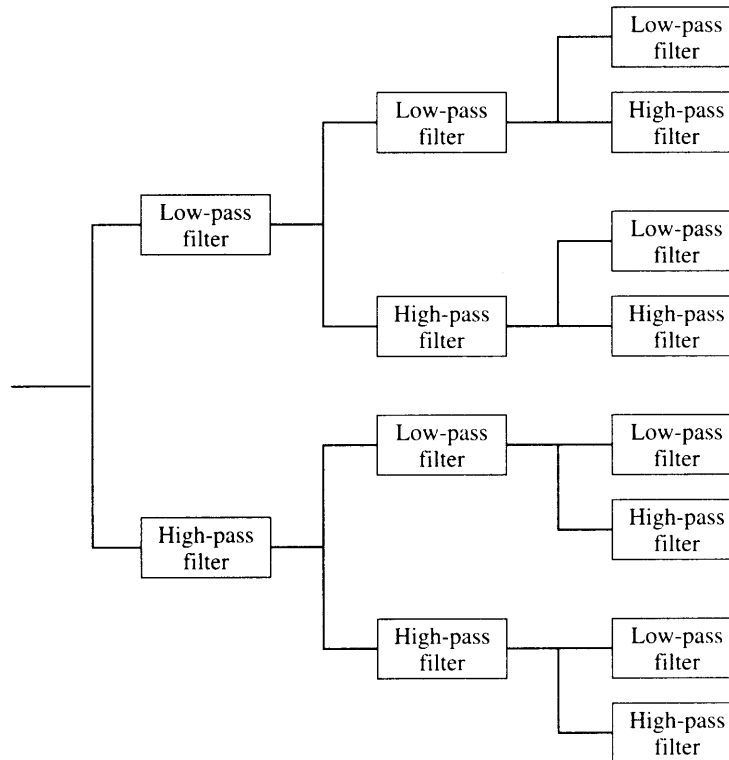
**14.3.1 Some Filters Used in Subband Coding**

The most frequently used filter banks in subband coding consist of a cascade of stages, where each stage consists of a low-pass filter and a high-pass filter, as shown in Figure 14.5. The most popular among these filters are the *quadrature mirror filters* (QMF), which were first proposed by Crosier, Esteban, and Galand [197]. These filters have the property that if the impulse response of the low-pass filter is given by  $\{h_n\}$ , then the high-pass impulse response is given by  $\{(-1)^n h_{N-1-n}\}$ . The QMF filters designed by Johnston [198] are widely used in a number of applications. The filter coefficients for 8-, 16-, and 32-tap filters are given in Tables 14.1–14.3. Notice that the filters are symmetric; that is,

$$h_{N-1-n} = h_n \quad n = 0, 1, \dots, \frac{N}{2} - 1. \quad (14.23)$$

As we shall see later, the filters with fewer taps are less efficient in their decomposition than the filters with more taps. However, from Equation (14.18) we can see that the number of taps dictates the number of multiply-add operations necessary to generate the filter outputs. Thus, if we want to obtain more efficient decompositions, we do so by increasing the amount of computation.

Another popular set of filters are the Smith-Barnwell filters [199], some of which are shown in Tables 14.4 and 14.5.



**FIGURE 14.5** An eight-band filter bank.

**TABLE 14.1** Coefficients for the 8-tap Johnston low-pass filter.

$h_0, h_7$	0.00938715
$h_1, h_6$	0.06942827
$h_2, h_5$	-0.07065183
$h_3, h_4$	0.48998080

**TABLE 14.2** Coefficients for the 16-tap Johnston low-pass filter.

$h_0, h_{15}$	0.002898163
$h_1, h_{14}$	-0.009972252
$h_2, h_{13}$	-0.001920936
$h_3, h_{12}$	0.03596853
$h_4, h_{11}$	-0.01611869
$h_5, h_{10}$	-0.09530234
$h_6, h_9$	0.1067987
$h_7, h_8$	0.4773469

**TABLE 14.3** Coefficients for the 32-tap Johnston low-pass filter.

$h_0, h_{31}$	0.0022551390
$h_1, h_{30}$	-0.0039715520
$h_2, h_{29}$	-0.0019696720
$h_3, h_{28}$	0.0081819410
$h_4, h_{27}$	0.00084268330
$h_5, h_{26}$	-0.014228990
$h_6, h_{25}$	0.0020694700
$h_7, h_{24}$	0.022704150
$h_8, h_{23}$	-0.0079617310
$h_9, h_{22}$	-0.034964400
$h_{10}, h_{21}$	0.019472180
$h_{11}, h_{20}$	0.054812130
$h_{12}, h_{19}$	-0.044524230
$h_{13}, h_{18}$	-0.099338590
$h_{14}, h_{17}$	0.13297250
$h_{15}, h_{16}$	0.46367410

**TABLE 14.4** Coefficients for the eight-tap Smith-Barnwell low-pass filter.

$h_0$	0.0348975582178515
$h_1$	-0.01098301946252854
$h_2$	-0.06286453934951963
$h_3$	0.223907720892568
$h_4$	0.556856993531445
$h_5$	0.357976304997285
$h_6$	-0.02390027056113145
$h_7$	-0.07594096379188282

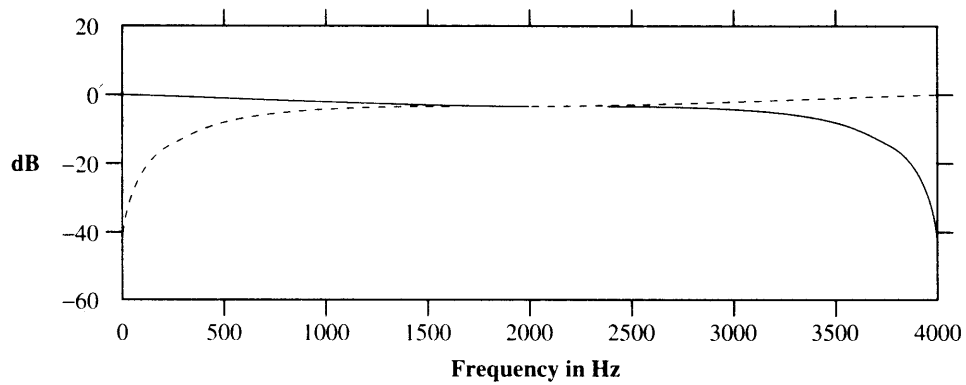
**TABLE 14.5** Coefficients for the 16-tap Smith-Barnwell low-pass filter.

$h_0$	0.02193598203004352
$h_1$	0.001578616497663704
$h_2$	-0.06025449102875281
$h_3$	-0.0118906596205391
$h_4$	0.137537915636625
$h_5$	0.05745450056390939
$h_6$	-0.321670296165893
$h_7$	-0.528720271545339
$h_8$	-0.295779674500919
$h_9$	0.0002043110845170894
$h_{10}$	0.02906699789446796
$h_{11}$	-0.03533486088708146
$h_{12}$	-0.006821045322743358
$h_{13}$	0.02606678468264118
$h_{14}$	0.001033363491944126
$h_{15}$	-0.01435930957477529

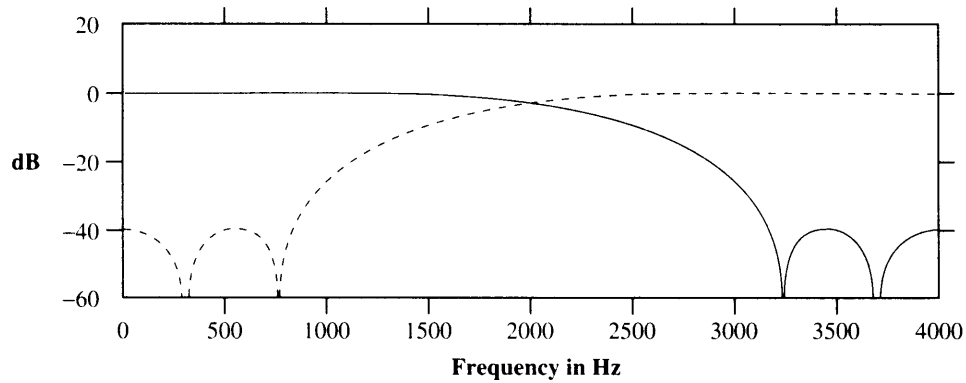


These families of filters differ in a number of ways. For example, consider the Johnston eight-tap filter and the Smith-Barnwell eight-tap filter. The magnitude transfer functions for these two filters are plotted in Figure 14.6. Notice that the cutoff for the Smith-Barnwell filter is much sharper than the cutoff for the Johnston filter. This means that the separation provided by the eight-tap Johnston filter is not as good as that provided by the eight-tap Smith-Barnwell filter. We will see the effect of this when we look at image compression later in this chapter.

These filters are examples of some of the more popular filters. Many more filters exist in the literature, and more are being discovered.



(a)



(b)

**FIGURE 14. 6** Magnitude transfer functions of the (a) eight-tap Johnston and (b) eight-tap Smith-Barnwell filters.

## 14.4 The Basic Subband Coding Algorithm

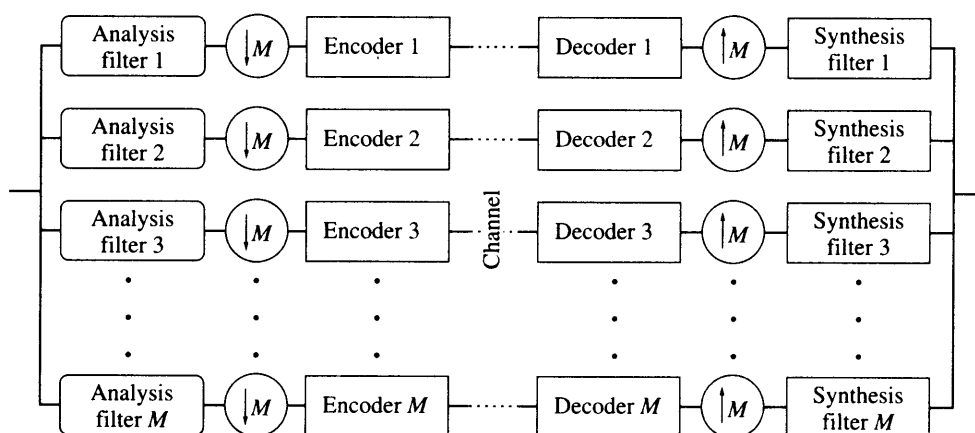
The basic subband coding system is shown in Figure 14.7.

### 14.4.1 Analysis

The source output is passed through a bank of filters, called the analysis filter bank, which covers the range of frequencies that make up the source output. The passbands of the filters can be nonoverlapping or overlapping. Nonoverlapping and overlapping filter banks are shown in Figure 14.8. The outputs of the filters are then subsampled.

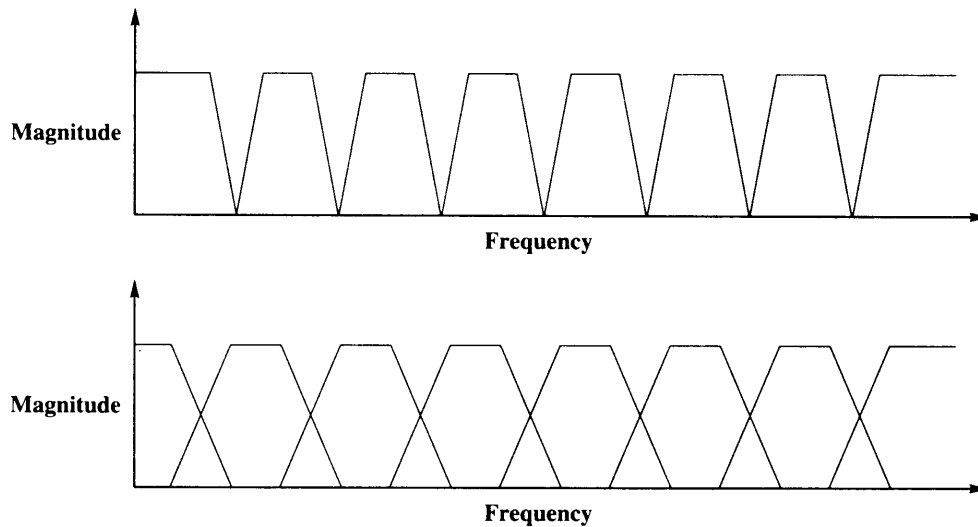
The justification for the subsampling is the Nyquist rule and its generalization, which tells us that we only need twice as many samples per second as the range of frequencies. This means that we can reduce the number of samples at the output of the filter because the range of frequencies at the output of the filter is less than the range of frequencies at the input to the filter. This process of reducing the number of samples is called *decimation*,<sup>1</sup> or *downsampling*. The amount of decimation depends on the ratio of the bandwidth of the filter output to the filter input. If the bandwidth at the output of the filter is  $1/M$  of the bandwidth at the input to the filter, we would decimate the output by a factor of  $M$  by keeping every  $M$ th sample. The symbol  $M \downarrow$  is used to denote this decimation.

Once the output of the filters has been decimated, the output is encoded using one of several encoding schemes, including ADPCM, PCM, and vector quantization.



**FIGURE 14.7** Block diagram of the subband coding system.

<sup>1</sup> The word *decimation* has a rather bloody origin. During the time of the Roman empire, if a legion broke ranks and ran during battle, its members were lined up and every tenth person was killed. This process was called decimation.



**FIGURE 14.8** Nonoverlapping and overlapping filter banks.

### 14.4.2 Quantization and Coding

Along with the selection of the compression scheme, the allocation of bits between the subbands is an important design parameter. Different subbands contain differing amounts of information. Therefore, we need to allocate the available bits among the subbands according to some measure of the information content. There are a number of different ways we could distribute the available bits. For example, suppose we were decomposing the source output into four bands and we wanted a coding rate of 1 bit per sample. We could accomplish this by using 1 bit per sample for each of the four bands. On the other hand, we could simply discard the output of two of the bands and use 2 bits per sample for the two remaining bands. Or, we could discard the output of three of the four filters and use 4 bits per sample to encode the output of the remaining filter.

This *bit allocation* procedure can have a significant impact on the quality of the final reconstruction, especially when the information content of different bands is very different.

If we use the variance of the output of each filter as a measure of information, and assume that the compression scheme is scalar quantization, we can arrive at several simple bit allocation schemes (see Section 13.5). If we use a slightly more sophisticated model for the outputs of the filters, we can arrive at significantly better bit allocation procedures (see Section 14.9).

### 14.4.3 Synthesis

The quantized and coded coefficients are used to reconstruct a representation of the original signal at the decoder. First, the encoded samples from each subband are decoded at the receiver. These decoded values are then upsampled by inserting an appropriate number of

0s between samples. Once the number of samples per second has been brought back to the original rate, the upsampled signals are passed through a bank of reconstruction filters. The outputs of the reconstruction filters are added to give the final reconstructed outputs.

We can see that the basic subband system is simple. The three major components of this system are the *analysis and synthesis filters*, the *bit allocation* scheme, and the *encoding* scheme. A substantial amount of research has focused on each of these components. Various filter bank structures have been studied in order to find filters that are simple to implement and provide good separation between the frequency bands. In the next section we briefly look at some of the techniques used in the design of filter banks, but our descriptions are necessarily limited. For a (much) more detailed look, see the excellent book by P.P. Vaidyanathan [200].

The bit allocation procedures have also been extensively studied in the contexts of subband coding, wavelet-based coding, and transform coding. We have already described some bit allocation schemes in Section 13.5, and we describe a different approach in Section 14.9. There are also some bit allocation procedures that have been developed in the context of wavelets, which we describe in the next chapter.

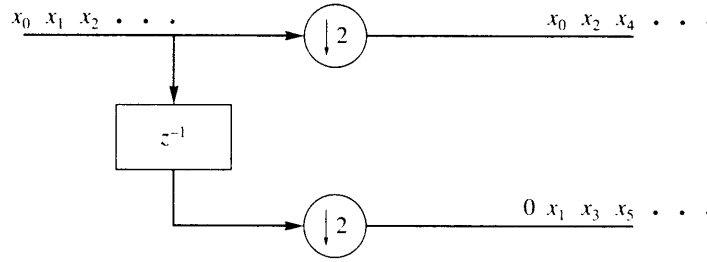
The separation of the source output according to frequency also opens up the possibility for innovative ways to use compression algorithms. The decomposition of the source output in this manner provides inputs for the compression algorithms, each of which has more clearly defined characteristics than the original source output. We can use these characteristics to select separate compression schemes appropriate to each of the different inputs.

Human perception of audio and video inputs is frequency dependent. We can use this fact to design our compression schemes so that the frequency bands that are most important to perception are reconstructed most accurately. Whatever distortion there has to be is introduced in the frequency bands to which humans are least sensitive. We describe some applications to the coding of speech, audio, and images later in this chapter.

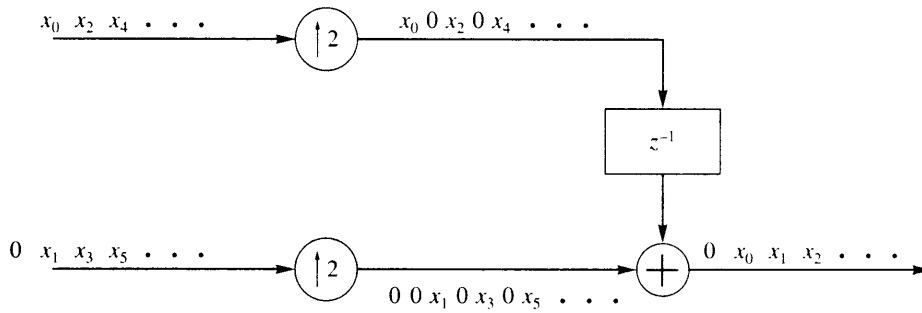
Before we proceed to bit allocation procedures and implementations, we provide a more mathematical analysis of the subband coding system. We also look at some approaches to the design of filter banks for subband coding. The analysis relies heavily on the Z-transform concepts introduced in Chapter 12 and will primarily be of interest to readers with an electrical engineering background. The material is not essential to understanding the rest of the chapter; if you are not interested in these details, you should skip these sections and go directly to Section 14.9.

### 14.5 Design of Filter Banks ★

In this and the following starred section we will take a closer look at the analysis, down-sampling, up-sampling, and synthesis operations. Our approach follows that of [201]. We assume familiarity with the Z-transform concepts of Chapter 12. We begin with some notation. Suppose we have a sequence  $x_0, x_1, x_2, \dots$ . We can divide this sequence into two subsequences:  $x_0, x_2, x_4, \dots$  and  $x_1, x_3, x_5, \dots$  using the scheme shown in Figure 14.9, where  $z^{-1}$  corresponds to a delay of one sample and  $\downarrow M$  denotes a subsampling by a factor of  $M$ . This subsampling process is called *downsampling* or *decimation*.



**FIGURE 14. 9** Decomposition of an input sequence into its odd and even components.

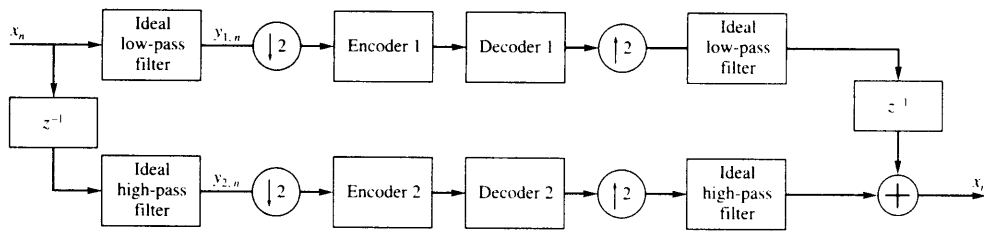


**FIGURE 14. 10** Reconstructing the input sequence from its odd and even components.

The original sequence can be recovered from the two downsampled sequences by inserting 0s between consecutive samples of the subsequences, delaying the top branch by one sample and adding the two together. Adding 0s between consecutive samples is called *upsampling* and is denoted by  $\uparrow M$ . The reconstruction process is shown in Figure 14.10.

While we have decomposed the source output sequence into two subsequences, there is no reason for the statistical and spectral properties of these subsequences to be different. As our objective is to decompose the source output sequences into subsequences with differing characteristics, there is much more yet to be done.

Generalizing this, we obtain the system shown in Figure 14.11. The source output sequence is fed to an ideal low-pass filter and an ideal high-pass filter, each with a bandwidth of  $\pi/2$ . We assume that the source output sequence had a bandwidth of  $\pi$ . If the original source signal was sampled at the Nyquist rate, as the output of the two filters have bandwidths half that of the original sequence, the filter outputs are actually oversampled by a factor of two. We can, therefore, subsample these signals by a factor of two without any loss of information. The two bands now have different characteristics and can be encoded differently. For the moment let's assume that the encoding is performed in a lossless manner so that the reconstructed sequence exactly matches the source output sequence.



**FIGURE 14. 11** Decomposition into two bands using ideal filters.

Let us look at how this system operates in the frequency domain. We begin by looking at the downsampling operation.

**14.5.1 Downsampling ★**

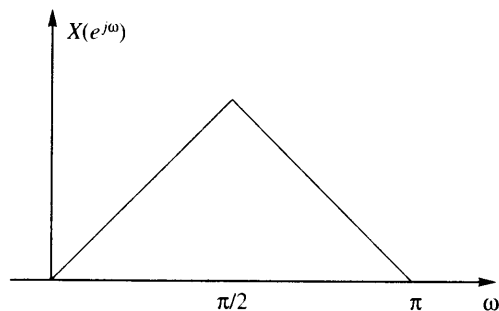
To see the effects of downsampling, we will obtain the Z-transform of the downsampled sequence in terms of the original source sequence. Because it is easier to understand what is going on if we can visualize the process, we will use the example of a source sequence that has the frequency profile shown in Figure 14.12. For this sequence the output of the ideal filters will have the shape shown in Figure 14.13.

Let's represent the downsampled sequence as  $\{w_{i,n}\}$ . The Z-transform  $W_1(z)$  of the downsampled sequence  $w_{i,n}$  is

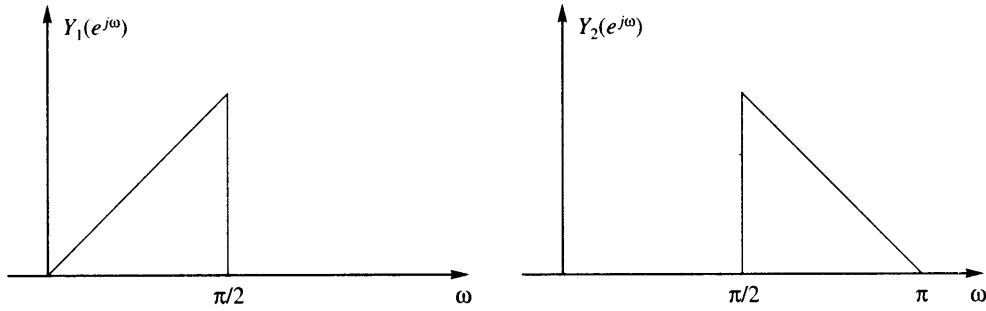
$$W_1(z) = \sum w_{1,n} z^{-n} \tag{14.24}$$

The downsampling operation means that

$$w_{1,n} = y_{1,2n} \tag{14.25}$$



**FIGURE 14. 12** Spectrum of the source output.



**FIGURE 14.13** Spectrum of the outputs of the ideal filters.

In order to find the Z-transform of this sequence, we go through a two-step process. Define the sequence

$$y'_{1,n} = \frac{1}{2}(1 + e^{jn\pi})y_{1,n} \quad (14.26)$$

$$= \begin{cases} y_{1,n} & n \text{ even} \\ 0 & \text{otherwise.} \end{cases} \quad (14.27)$$

We could also have written Equation (14.26) as

$$y'_{1,n} = \frac{1}{2}(1 + (-1)^n)y_{1,n}$$

however, writing the relationship as in Equation (14.26) makes it easier to extend this development to the case where we divide the source output into more than two bands.

The Z-transform of  $y'_{1,n}$  is given as

$$Y'_1(z) = \sum_{n=-\infty}^{\infty} \frac{1}{2}(1 + e^{jn\pi})y_{1,n}z^{-n}. \quad (14.28)$$

Assuming all summations converge,

$$Y'_1(z) = \frac{1}{2} \sum_{n=-\infty}^{\infty} y_{1,n}z^{-n} + \frac{1}{2} \sum_{n=-\infty}^{\infty} y_{1,n}(ze^{-j\pi})^{-n} \quad (14.29)$$

$$= \frac{1}{2}Y_1(z) + \frac{1}{2}Y_1(-z) \quad (14.30)$$

where we have used the fact that

$$e^{-j\pi} = \cos(\pi) - j \sin \pi = -1.$$

Noting that

$$w_{1,n} = y'_{1,2n} \quad (14.31)$$

$$W_1(z) = \sum_{n=-\infty}^{\infty} w_{1,n} z^{-n} = \sum_{n=-\infty}^{\infty} y'_{1,2n} z^{-n}. \quad (14.32)$$

Substituting  $m = 2n$ ,

$$W_1(z) = \sum_{n=-\infty}^{\infty} y'_{1,m} z^{-\frac{m}{2}} \quad (14.33)$$

$$= Y'_1(z^{\frac{1}{2}}) \quad (14.34)$$

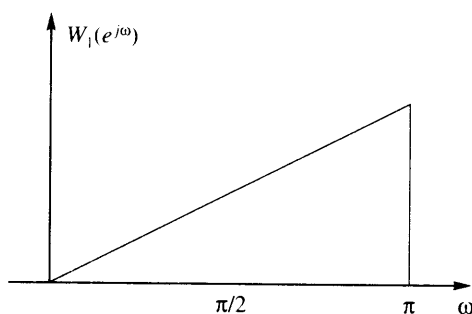
$$= \frac{1}{2} Y_1(z^{\frac{1}{2}}) + \frac{1}{2} Y_1(-z^{\frac{1}{2}}). \quad (14.35)$$

Why didn't we simply write the Z-transform of  $w_{1,n}$  directly in terms of  $y_{1,n}$  and use the substitution  $m = 2n$ ? If we had, the equivalent equation to (14.33) would contain the odd indexed terms of  $y_{1,n}$ , which we know do not appear at the output of the downsampler. In Equation (14.33), we also get the odd indexed terms of  $y'_{1,n}$ ; however, as these terms are all zero (see Equation (14.26)), they do not contribute to the Z-transform.

Substituting  $z = e^{j\omega}$  we get

$$W_1(e^{j\omega}) = \frac{1}{2} Y_1(e^{j\frac{\omega}{2}}) + \frac{1}{2} Y_1(-e^{j\frac{\omega}{2}}). \quad (14.36)$$

Plotting this for the  $Y_1(e^{j\omega})$  of Figure 14.13, we get the spectral shape shown in Figure 14.14; that is, the spectral shape of the downsampled signal is a stretched version of the spectral shape of the original signal. A similar situation exists for the downsampled signal  $w_{2,n}$ .



**FIGURE 14. 14** Spectrum of the downsampled low-pass filter output.



### 14.5.2 Upsampling ★

Let's take a look now at what happens after the upsampling. The upsampled sequence  $v_{1,n}$  can be written as

$$v_{1,n} = \begin{cases} w_{1,\frac{n}{2}} & n \text{ even} \\ 0 & n \text{ odd.} \end{cases} \quad (14.37)$$

The Z-transform  $V_1(z)$  is thus

$$V_1(z) = \sum_{n=-\infty}^{\infty} v_{1,n} z^{-n} \quad (14.38)$$

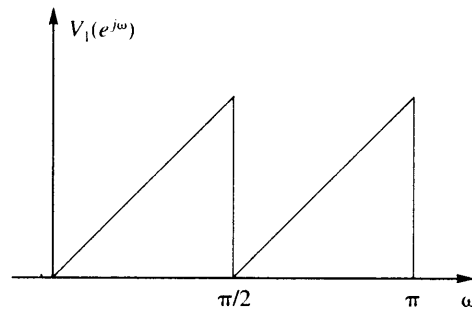
$$= \sum_{n=-\infty}^{\infty} w_{1,\frac{n}{2}} z^{-n} \quad n \text{ even} \quad (14.39)$$

$$= \sum_{m=-\infty}^{\infty} w_{1,m} z^{-2m} \quad (14.40)$$

$$= W_1(z^2). \quad (14.41)$$

The spectrum is sketched in Figure 14.15. The "stretching" of the sequence in the time domain has led to a compression in the frequency domain. This compression has also resulted in a replication of the spectrum in the  $[0, \pi]$  interval. This replication effect is called *imaging*. We remove the images by using an ideal low-pass filter in the top branch and an ideal high-pass filter in the bottom branch.

Because the use of the filters prior to sampling reduces the bandwidth, which in turn allows the downsampling operation to proceed without aliasing, these filters are called *anti-aliasing* filters. Because they decompose the source output into components, they are also called *analysis* filters. The filters after the upsampling operation are used to recombine the original signal; therefore, they are called *synthesis* filters. We can also view these filters as interpolating between nonzero values to recover the signal at the point that we have inserted zeros. Therefore, these filters are also called *interpolation* filters.



**FIGURE 14.15** Spectrum of the upsampled signal.

Although the use of ideal filters would give us perfect reconstruction of the source output, in practice we do not have ideal filters available. When we use more realistic filters in place of the ideal filters, we end up introducing distortion. In the next section we look at this situation and discuss how we can reduce or remove this distortion.

## 14.6 Perfect Reconstruction Using Two-Channel Filter Banks ★

Suppose we replace the ideal low-pass filter in Figure 14.11 with a more realistic filter with the magnitude response shown in Figure 14.4. The spectrum of the output of the low-pass filter is shown in Figure 14.16. Notice that we now have nonzero values for frequencies above  $\frac{\pi}{2}$ . If we now subsample by two, we will end up sampling at *less* than twice the highest frequency, or in other words, we will be sampling at below the Nyquist rate. This will result in the introduction of aliasing distortion, which will show up in the reconstruction. A similar situation will occur when we replace the ideal high-pass filter with a realistic high-pass filter.

In order to get perfect reconstruction after synthesis, we need to somehow get rid of the aliasing and imaging effects. Let us look at the conditions we need to impose upon the filters  $H_1(z)$ ,  $H_2(z)$ ,  $K_1(z)$ , and  $K_2(z)$  in order to accomplish this. These conditions are called *perfect reconstruction (PR)* conditions.

Consider Figure 14.17. Let's obtain an expression for  $\hat{X}(z)$  in terms of  $H_1(z)$ ,  $H_2(z)$ ,  $K_1(z)$ , and  $K_2(z)$ . We start with the reconstruction:

$$\hat{X}(z) = U_1(z) + U_2(z) \quad (14.42)$$

$$= V_1(z)K_1(z) + V_2(z)K_2(z). \quad (14.43)$$

Therefore, we need to find  $V_1(z)$  and  $V_2(z)$ . The sequence  $v_{1,n}$  is obtained by upsampling  $w_{1,n}$ . Therefore, from Equation (14.41),

$$V_1(z) = W_1(z^2). \quad (14.44)$$

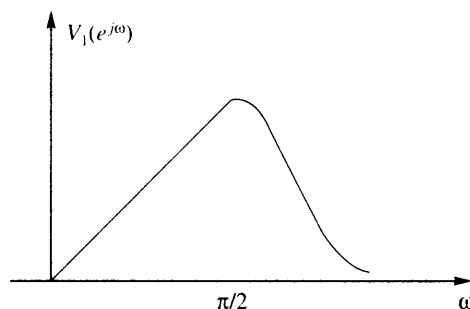
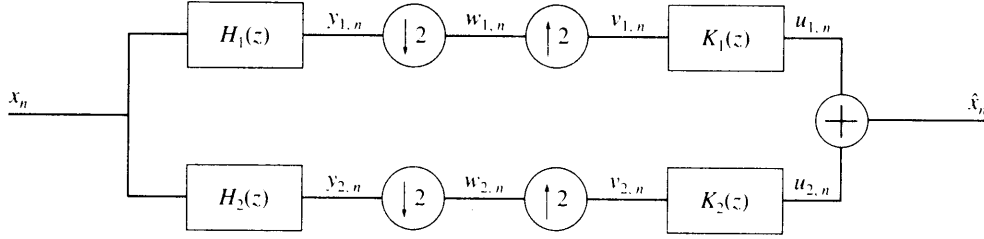


FIGURE 14.16 Output of the low-pass filter.



**FIGURE 14.17** Two-channel subband decimation and interpolation.

The sequence  $w_{1,n}$  is obtained by downsampling  $y_{1,n}$ ,

$$Y_1(z) = X(z)H_1(z).$$

Therefore, from Equation (14.35),

$$W_1(z) = \frac{1}{2} [X(z^{\frac{1}{2}})H_1(z^{\frac{1}{2}}) + X(-z^{\frac{1}{2}})H_1(-z^{\frac{1}{2}})] \quad (14.45)$$

and

$$V_1(z) = \frac{1}{2} [X(z)H_1(z) + X(-z)H_1(-z)]. \quad (14.46)$$

Similarly, we can also show that

$$V_2(z) = \frac{1}{2} [X(z)H_2(z) + X(-z)H_2(-z)]. \quad (14.47)$$

Substituting the expressions for  $V_1(z)$  and  $V_2(z)$  into Equation (14.43) we obtain

$$\begin{aligned} \hat{X}(z) &= \frac{1}{2} [H_1(z)K_1(z) + H_2(z)K_2(z)] X(z) \\ &\quad + \frac{1}{2} [H_1(-z)K_1(z) + H_2(-z)K_2(z)] X(-z). \end{aligned} \quad (14.48)$$

For perfect reconstruction we would like  $\hat{X}(z)$  to be a delayed and perhaps amplitude-scaled version of  $X(z)$ ; that is,

$$\hat{X}(z) = cX(z)z^{-n_0}. \quad (14.49)$$

In order for this to be true, we need to impose conditions on  $H_1(z)$ ,  $H_2(z)$ ,  $K_1(z)$ , and  $K_2(z)$ . There are several ways we can do this, with each approach providing a different solution. One approach involves writing Equation (14.48) in matrix form as

$$\hat{X}(z) = \frac{1}{2} \begin{bmatrix} K_1(z) & K_2(z) \end{bmatrix} \begin{bmatrix} H_1(z) & H_1(-z) \\ H_2(z) & H_2(-z) \end{bmatrix} \begin{bmatrix} X(z) \\ X(-z) \end{bmatrix} \quad (14.50)$$

For perfect reconstruction, we need

$$\begin{bmatrix} K_1(z) & K_2(z) \end{bmatrix} \begin{bmatrix} H_1(z) & H_1(-z) \\ H_2(z) & H_2(-z) \end{bmatrix} = \begin{bmatrix} cz^{-n_0} & 0 \end{bmatrix} \quad (14.51)$$

where we have absorbed the factor of  $\frac{1}{2}$  into the constant  $c$ . This means that the synthesis filters  $K_1(z)$  and  $K_2(z)$  satisfy

$$\begin{bmatrix} K_1(z) & K_2(z) \end{bmatrix} = \frac{cz^{-n_0}}{\det[\mathcal{H}(z)]} \begin{bmatrix} H_2(-z) & -H_1(-z) \end{bmatrix} \quad (14.52)$$

where

$$\mathcal{H}(z) = \begin{bmatrix} H_1(z) & H_1(-z) \\ H_2(z) & H_2(-z) \end{bmatrix} \quad (14.53)$$

If  $H_1(z)$  and/or  $H_2(z)$  are IIR filters, the reconstruction filters can become quite complex. Therefore, we would like to have both the analysis and synthesis filters be FIR filters. If we select the analysis filters to be FIR, then in order to guarantee that the synthesis filters are also FIR we need

$$\det[\mathcal{H}(z)] = \gamma z^{-n_1}$$

where  $\gamma$  is a constant. Examining  $\det[\mathcal{H}(z)]$

$$\begin{aligned} \det[\mathcal{H}(z)] &= H_1(z)H_2(-z) - H_1(-z)H_2(z) \\ &= P(z) - P(-z) = \gamma z^{-n_1} \end{aligned} \quad (14.54)$$

where  $P(z) = H_1(z)H_2(-z)$ . If we examine Equation (14.54), we can see that  $n_1$  has to be odd because all terms containing even powers of  $z$  in  $P(z)$  will be canceled out by the corresponding terms in  $P(-z)$ . Thus,  $P(z)$  can have an arbitrary number of even-indexed coefficients (as they will get canceled out), but there must be only one nonzero coefficient of an odd power of  $z$ . By choosing any valid factorization of the form

$$P(z) = P_1(z)P_2(z) \quad (14.55)$$

we can obtain many possible solutions of perfect reconstruction FIR filter banks with

$$H_1(z) = P_1(z) \quad (14.56)$$

and

$$H_2(z) = P_2(-z). \quad (14.57)$$

Although these filters are perfect reconstruction filters, for applications in data compression they suffer from one significant drawback. Because these filters may be of unequal bandwidth, the output of the larger bandwidth filter suffers from severe aliasing. If the output of both bands is available to the receiver, this is not a problem because the aliasing is canceled out in the reconstruction process. However, in many compression applications we discard the subband containing the least amount of energy, which will generally be the output of the filter with the smaller bandwidth. In this case the reconstruction will contain a large amount of aliasing distortion. In order to avoid this problem for compression applications, we generally wish to minimize the amount of aliasing in each subband. A class of filters that is useful in this situation is the *quadrature mirror filters* (QMF). We look at these filters in the next section.

### 14.6.1 Two-Channel PR Quadrature Mirror Filters ★

Before we introduce the quadrature mirror filters, let's rewrite Equation (14.48) as

$$\hat{X}(z) = T(z)X(z) + S(z)X(-z) \quad (14.58)$$

where

$$T(z) = \frac{1}{2} [H_1(z)K_1(z) + H_2(z)K_2(z)] \quad (14.59)$$

$$S(z) = \frac{1}{2} [H_1(-z)K_1(z) + H_2(-z)K_2(z)]. \quad (14.60)$$

In order for the reconstruction of the input sequence  $\{x_n\}$  to be a delayed, and perhaps scaled, version of  $\{x_n\}$ , we need to get rid of the aliasing term  $X(-z)$  and have  $T(z)$  be a pure delay. To get rid of the aliasing term, we need

$$S(z) = 0, \quad \forall z.$$

From Equation (14.60), this will happen if

$$K_1(z) = H_2(-z) \quad (14.61)$$

$$K_2(z) = -H_1(-z). \quad (14.62)$$

After removing the aliasing distortion, a delayed version of the input will be available at the output if

$$T(z) = cz^{-n_0} \quad c \text{ is a constant.} \quad (14.63)$$

Replacing  $z$  by  $e^{j\omega}$ , this means that we want

$$|T(e^{j\omega})| = \text{constant} \quad (14.64)$$

$$\arg(T(e^{j\omega})) = K\omega \quad K \text{ constant.} \quad (14.65)$$

The first requirement eliminates amplitude distortion, while the second, the linear phase requirement, is necessary to eliminate phase distortion. If these requirements are satisfied,

$$\hat{x}(n) = cx(n - n_0). \quad (14.66)$$

That is, the reconstructed signal is a delayed version of input signal  $x(n)$ . However, meeting both requirements simultaneously is not a trivial task.

Consider the problem of designing  $T(z)$  to have linear phase. Substituting (14.61) and (14.62) into Equation (14.59), we obtain

$$T(z) = \frac{1}{2} [H_1(z)H_2(-z) - H_1(-z)H_2(z)]. \quad (14.67)$$

Therefore, if we choose  $H_1(z)$  and  $H_2(z)$  to be linear phase FIR,  $T(z)$  will also be a linear phase FIR filter. In the QMF approach, we first select the low-pass filter  $H_1(z)$ , then define the high-pass filter  $H_2(z)$  to be a mirror image of the low-pass filter:

$$H_2(z) = H_1(-z). \quad (14.68)$$

This is referred to as a *mirror* condition and is the original reason for the name of the QMF filters [200]. We can see that this condition will force both filters to have equal bandwidth.

Given the mirror condition and  $H_1(z)$ , a linear phase FIR filter, we will have linear phase and

$$T(z) = \frac{1}{2}[H_1^2(z) - H_1^2(-z)]. \quad (14.69)$$

It is not clear that  $|T(e^{j\omega})|$  is a constant. In fact, we will show in Section 14.8 that a linear phase two-channel FIR QMF bank with the filters chosen as in Equation (14.68) can have PR property if and only if  $H_1(z)$  is in the simple two-tap form

$$H_1(z) = h_0 z^{-2k_0} + h_1 z^{-(2k_1+1)}. \quad (14.70)$$

Then,  $T(z)$  is given by

$$T(z) = 2h_0 h_1 z^{-(2k_0+2k_1+1)} \quad (14.71)$$

which is of the desired form  $c z^{-n}$ . However, if we look at the magnitude characteristics of the two filters, we see that they have poor cutoff characteristics. The magnitude of the low-pass filter is given by

$$|H_1(e^{j\omega})|^2 = h_0^2 + h_1^2 + 2h_0 h_1 \cos(2k_0 - 2k_1 - 1)\omega \quad (14.72)$$

and the high-pass filter is given by

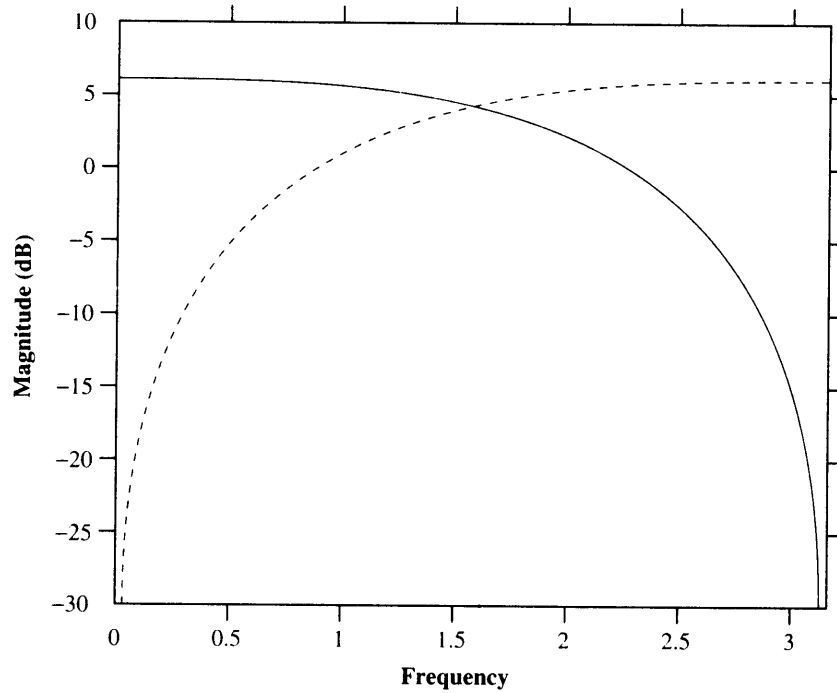
$$|H_2(e^{j\omega})|^2 = h_0^2 + h_1^2 - 2h_0 h_1 \cos(2k_0 - 2k_1 - 1)\omega. \quad (14.73)$$

For  $h_0 = h_1 = k_0 = k_1 = 1$ , the magnitude responses are plotted in Figure 14.18. Notice the poor cutoff characteristics of these two filters.

Thus, for perfect reconstruction with no aliasing and no amplitude or phase distortion, the mirror condition does not seem like such a good idea. However, if we slightly relax these rather strict conditions, we can obtain some very nice designs. For example, instead of attempting to eliminate all phase and amplitude distortion, we could elect to eliminate only the phase distortion and *minimize* the amplitude distortion. We can optimize the coefficients of  $H_1(z)$  such that  $|T(e^{j\omega})|$  is made as close to a constant as possible, while minimizing the stopband energy of  $H_1(z)$  in order to have a good low-pass characteristic. Such an optimization has been suggested by Johnston [198] and Jain and Crochiere [202]. They construct the objective function

$$J = \alpha \int_{\omega_s}^{\pi} |H_1(e^{j\omega})|^2 d\omega + (1 - \alpha) \int_0^{\pi} (1 - |T(e^{j\omega})|^2) d\omega \quad (14.74)$$

which has to be minimized to obtain  $H_1(z)$  and  $T_1(z)$ , where  $\omega_s$  is the cutoff frequency of the filter.



**FIGURE 14.18** Magnitude characteristics of the two-tap PR filters.

We can also go the other way and eliminate the amplitude distortion, then attempt to minimize the phase distortion. A review of these approaches can be found in [201, 200].

### 14.6.2 Power Symmetric FIR Filters ★

Another approach, independently discovered by Smith and Barnwell [199] and Mintzer [203], can be used to design a two-channel filter bank in which aliasing, amplitude distortion, and phase distortion can be completely eliminated. As discussed earlier, choosing

$$\begin{aligned} K_1(z) &= -H_2(-z) \\ K_2(z) &= H_1(-z) \end{aligned} \quad (14.75)$$

eliminates aliasing. This leaves us with

$$T(z) = \frac{1}{2}[H_1(-z)H_2(z) - H_1(z)H_2(-z)].$$

In the approach due to Smith and Barnwell [199] and Mintzer [203], with  $N$  an odd integer, we select

$$H_2(z) = z^{-N}H_1(-z^{-1}) \quad (14.76)$$

so that

$$T(z) = \frac{1}{2}z^{-N}[H_1(z)H_1(z^{-1}) + H_1(-z)H_1(-z^{-1})]. \quad (14.77)$$

Therefore, the perfect reconstruction requirement reduces to finding a prototype low-pass filter  $H(z) = H_1(z)$  such that

$$Q(z) = H(z)H(z^{-1}) + H(-z)H(-z^{-1}) = \text{constant}. \quad (14.78)$$

Defining

$$R(z) = H(z)H(z^{-1}), \quad (14.79)$$

the perfect reconstruction requirement becomes

$$Q(z) = R(z) + R(-z) = \text{constant}. \quad (14.80)$$

But  $R(z)$  is simply the Z-transform of the autocorrelation sequence of  $h(n)$ . The autocorrelation sequence  $\rho(n)$  is given by

$$\rho(n) = \sum_{k=0}^N h_k h_{k+n}. \quad (14.81)$$

The Z-transform of  $\rho(n)$  is given by

$$R(z) = \mathcal{Z}[\rho(n)] = \mathcal{Z}\left[\sum_{k=0}^N h_k h_{k+n}\right]. \quad (14.82)$$

We can express the sum  $\sum_{k=0}^N h_k h_{k+n}$  as a convolution:

$$h_n \otimes h_{-n} = \sum_{k=0}^N h_k h_{k+n}. \quad (14.83)$$

Using the fact that the Z-transform of a convolution of two sequences is the product of the Z-transforms of the individual sequences, we obtain

$$R(z) = \mathcal{Z}[h_n]\mathcal{Z}[h_{-n}] = H(z)H(z^{-1}). \quad (14.84)$$

Writing out  $R(z)$  as the Z-transform of the sequence  $\{\rho(n)\}$  we obtain

$$R(z) = \rho(N)z^N + \rho(N-1)z^{N-1} + \cdots + \rho(0) + \cdots + \rho(N-1)z^{-N-1} + \rho(N)z^{-N}. \quad (14.85)$$

Then  $R(-z)$  is

$$R(-z) = -\rho(N)z^N + \rho(N-1)z^{N-1} - \cdots + \rho(0) - \cdots + \rho(N-1)z^{-N-1} - \rho(N)z^{-N}. \quad (14.86)$$

Adding  $R(z)$  and  $R(-z)$ , we obtain  $Q(z)$  as

$$Q(z) = 2\rho(N-1)z^{N-1} + 2\rho(N-1)z^{N-3} + \cdots + \rho(0) + \cdots + 2\rho(N-1)z^{-N-1}. \quad (14.87)$$



Notice that the terms containing the odd powers of  $z$  got canceled out. Thus, for  $Q(z)$  to be a constant all we need is that for even values of the lag  $n$  (except for  $n = 0$ ),  $\rho(n)$  be zero. In other words

$$\rho(2n) = \sum_{k=0}^N h_k h_{k+2n} = 0, \quad n \neq 0. \quad (14.88)$$

Writing this requirement in terms of the impulse response:

$$\sum_{k=0}^N h_k h_{k-2n} = \begin{cases} 0 & n \neq 0 \\ \rho(0) & n = 0. \end{cases} \quad (14.89)$$

If we now normalize the impulse response,

$$\sum_{k=0}^N |h_k|^2 = 1 \quad (14.90)$$

we obtain the perfect reconstruction requirement

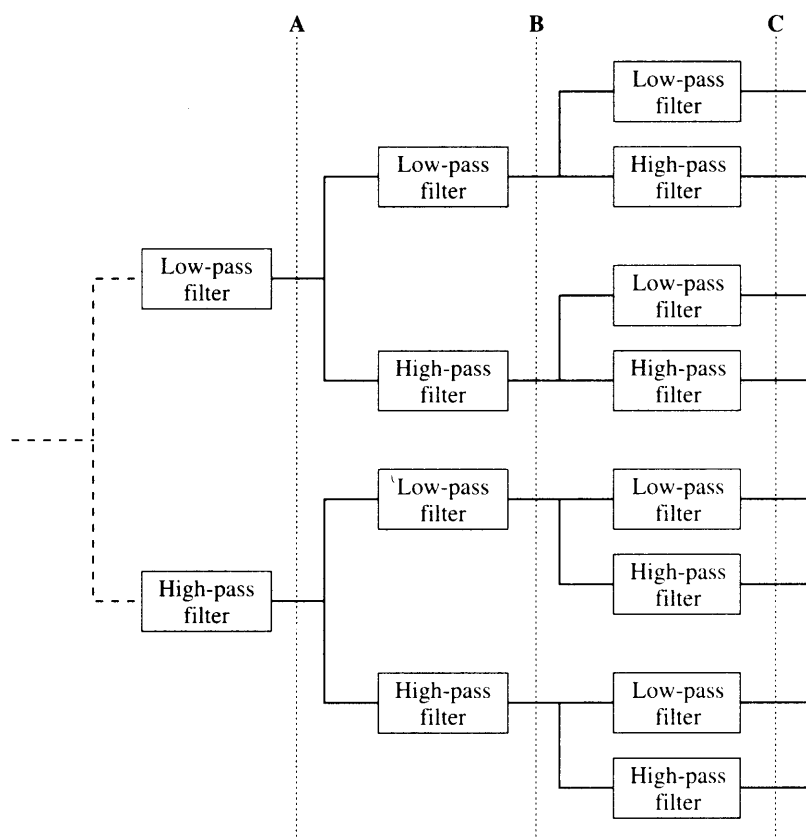
$$\sum_{k=0}^N h_k h_{k+2n} = \delta_n. \quad (14.91)$$

In other words, for perfect reconstruction, the impulse response of the prototype filter is orthogonal to the twice-shifted version of itself.

## 14.7 *M*-Band QMF Filter Banks ★

We have looked at how we can decompose an input signal into two bands. In many applications it is necessary to divide the input into multiple bands. We can do this by using a recursive two-band splitting as shown in Figure 14.19, or we can obtain banks of filters that directly split the input into multiple bands. Given that we have good filters that provide two-band splitting, it would seem that using a recursive splitting, as shown in Figure 14.19, would be an efficient way of obtaining an  $M$ -band split. Unfortunately, even when the spectral characteristics of the filters used for the two-band split are quite good, when we employ them in the tree structure shown in Figure 14.19, the spectral characteristics may not be very good. For example, consider the four-tap filter with filter coefficients shown in Table 14.6. In Figure 14.20 we show what happens to the spectral characteristics when we look at the two-band split (at point **A** in Figure 14.19), the four-band split (at point **B** in Figure 14.19), and the eight-band split (at point **C** in Figure 14.19). For a two-band split the magnitude characteristic is flat, with some aliasing. When we employ these same filters to obtain a four-band split from the two-band split, there is an increase in the aliasing. When we go one step further to obtain an eight-band split, the magnitude characteristics deteriorate substantially, as evidenced by Figure 14.20. The various bands are no longer clearly distinct. There is significant overlap between the bands, and hence there will be a significant amount of aliasing in each band.

In order to see why there is an increase in distortion, let us follow the top branch of the tree. The path followed by the signal is shown in Figure 14.21a. As we will show later



**FIGURE 14. 19** Decomposition of an input sequence into multiple bands by recursively using a two-band split.

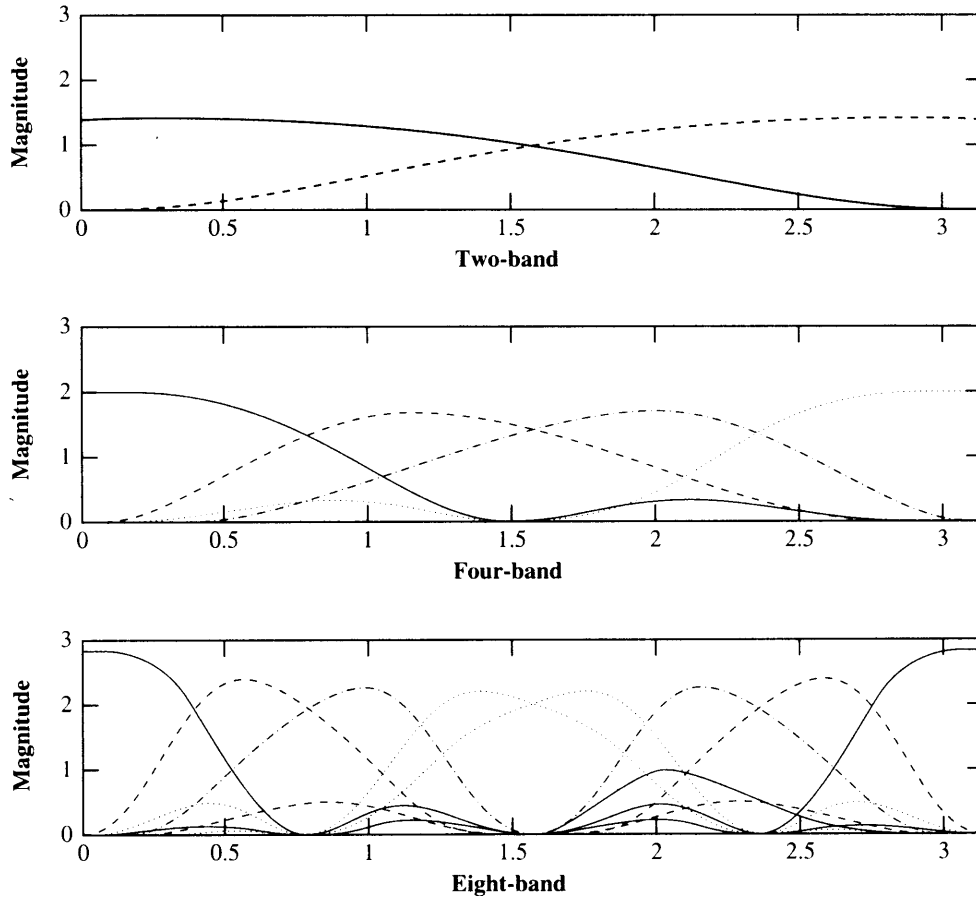
**TABLE 14. 6** Coefficients for the four-tap Daubechies low-pass filter.

$h_0$	0.4829629131445341
$h_1$	0.8365163037378079
$h_2$	0.2241438680420134
$h_3$	-0.1294095225512604

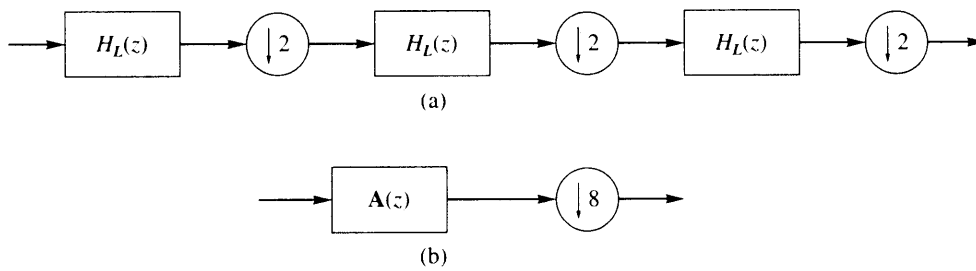
(Section 14.8), the three filters and downsamplers can be replaced by a single filter and downsampler as shown in Figure 14.21b, where

$$A(z) = H_L(z)H_L(z^2)H_L(z^4). \quad (14.92)$$

If  $H_L(z)$  corresponds to a 4-tap filter, then  $A(z)$  corresponds to a  $3 \times 6 \times 12 = 216$ -tap filter! However, this is a severely constrained filter because it was generated using only



**FIGURE 14. 20** Spectral characteristics at points A, B, and C.



**FIGURE 14. 21** Equivalent structures for recursive filtering using a two-band split.

four coefficients. If we had set out to design a 216-tap filter from scratch, we would have had significantly more freedom in selecting the coefficients. This is a strong motivation for designing filters directly for the  $M$ -band case.

An  $M$ -band filter bank has two sets of filters that are arranged as shown in Figure 14.7. The input signal  $x(n)$  is split into  $M$  frequency bands using an analysis bank of  $M$  filters of bandwidth  $\pi/M$ . The signal in any of these  $M$  channels is then downsampled by a factor  $L$ . This constitutes the analysis bank. The subband signals  $y_k(n)$  are encoded and transmitted. At the synthesis stage the subband signals are then decoded, upsampled by a factor of  $L$  by interlacing adjacent samples with  $L - 1$  zeros, and then passed through the synthesis or interpolation filters. The output of all these synthesis filters is added together to obtain the reconstructed signal. This constitutes the synthesis filter bank. Thus, the analysis and synthesis filter banks together take an input signal  $x(n)$  and produce an output signal  $\hat{x}(n)$ . These filters could be any combination of FIR and IIR filters.

Depending on whether  $M$  is less than, equal to, or greater than  $L$ , the filter bank is called an *underdecimated*, *critically (maximally) decimated*, or *overdecimated* filter bank. For most practical applications, maximal decimation or "critical subsampling" is used.

A detailed study of  $M$ -band filters is beyond the scope of this chapter. Suffice it to say that in broad outline much of what we said about two-band filters can be generalized to  $M$ -band filters. (For more on this subject, see [200].)

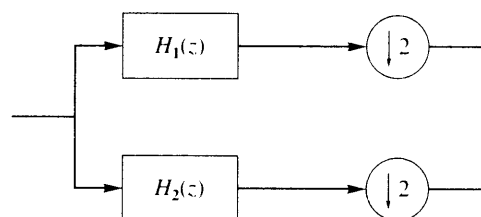
## 14.8 The Polyphase Decomposition ★

A major problem with representing the combination of filters and downsamplers is the time-varying nature of the up- and downsamplers. An elegant way of solving this problem is with the use of *polyphase decomposition*. In order to demonstrate this concept, let us first consider the simple case of two-band splitting. We will first consider the analysis portion of the system shown in Figure 14.22. Suppose the analysis filter  $H_1(z)$  is given by

$$H_1(z) = h_0 + h_1z^{-1} + h_2z^{-2} + h_3z^{-3} + \dots \quad (14.93)$$

By grouping the odd and even terms together, we can write this as

$$H_1(z) = (h_0 + h_2z^{-2} + h_4z^{-4} + \dots) + z^{-1}(h_1 + h_3z^{-2} + h_5z^{-4} + \dots). \quad (14.94)$$



**FIGURE 14.22** Analysis portion of a two-band subband coder.

Define

$$H_{10}(z) = h_0 + h_2z^{-1} + h_4z^{-2} + \dots \quad (14.95)$$

$$H_{11}(z) = h_1 + h_3z^{-1} + h_5z^{-2} + \dots \quad (14.96)$$

Then  $H_1(z) = H_{10}(z^2) + z^{-1}H_{11}(z^2)$ . Similarly, we can decompose the filter  $H_2(z)$  into components  $H_{20}(z)$  and  $H_{21}(z)$ , and we can represent the system of Figure 14.22 as shown in Figure 14.23. The filters  $H_{10}(z)$ ,  $H_{11}(z)$  and  $H_{20}(z)$ ,  $H_{21}(z)$  are called the polyphase components of  $H_1(z)$  and  $H_2(z)$ .

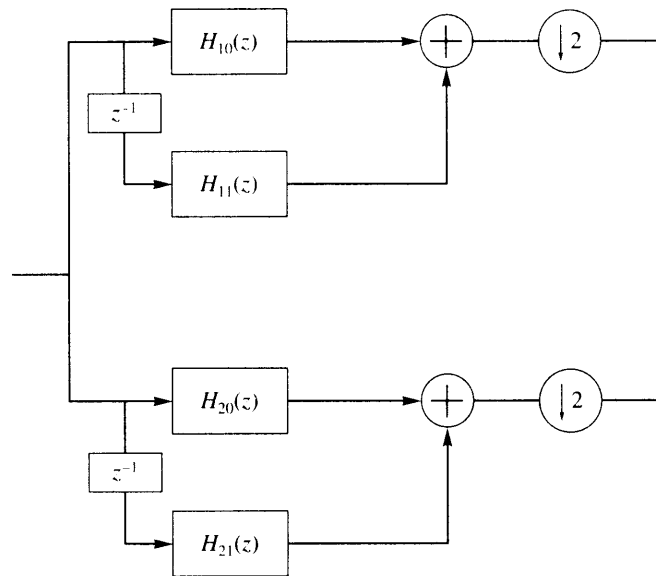
Let's take the inverse Z-transform of the polyphase components of  $H_1(z)$ :

$$h_{10}(n) = h_{2n} \quad n = 0, 1, \dots \quad (14.97)$$

$$h_{11}(n) = h_{2n+1} \quad n = 0, 1, \dots \quad (14.98)$$

Thus,  $h_{10}(n)$  and  $h_{11}(n)$  are simply the impulse response  $h_n$  downsampled by two. Consider the output of the downsampler for a given input  $X(z)$ . The input to the downsampler is  $X(z)H_1(z)$ ; thus, the output from Equation (14.35) is

$$Y_1(z) = \frac{1}{2}X\left(z^{\frac{1}{2}}\right)H_1\left(z^{\frac{1}{2}}\right) + \frac{1}{2}X\left(-z^{\frac{1}{2}}\right)H_1\left(-z^{\frac{1}{2}}\right). \quad (14.99)$$



**FIGURE 14. 23** Alternative representation of the analysis portion of a two-band subband coder.

Replacing  $H_1(z)$  with its polyphase representation, we get

$$\begin{aligned} Y_1(z) &= \frac{1}{2}X\left(z^{\frac{1}{2}}\right)\left[H_{10}(z) + z^{-\frac{1}{2}}H_{11}(z)\right] + \frac{1}{2}X\left(-z^{\frac{1}{2}}\right)\left[H_{10}(z) - z^{-\frac{1}{2}}H_{11}(z)\right] \quad (14.100) \\ &= H_{10}(z)\left[\frac{1}{2}X\left(z^{\frac{1}{2}}\right) + \frac{1}{2}X\left(-z^{\frac{1}{2}}\right)\right] + H_{11}(z)\left[\frac{1}{2}z^{-\frac{1}{2}}X\left(z^{\frac{1}{2}}\right) - \frac{1}{2}z^{-\frac{1}{2}}X\left(-z^{\frac{1}{2}}\right)\right] \quad (14.101) \end{aligned}$$

Note that the first expression in square brackets is the output of a downsampler whose input is  $X(z)$ , while the quantity in the second set of square brackets is the output of a downsampler whose input is  $z^{-1}X(z)$ . Therefore, we could implement this system as shown in Figure 14.24.

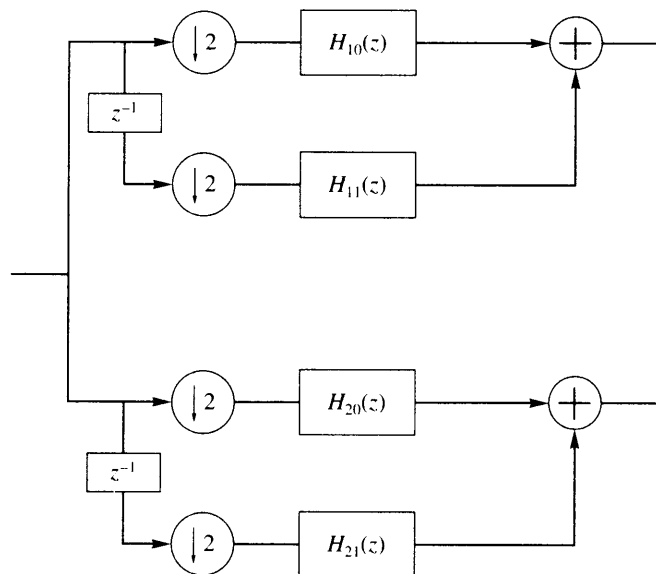
Now let us consider the synthesis portion of the two-band system shown in Figure 14.25. As in the case of the analysis portion, we can write the transfer functions in terms of their polyphase representation. Thus,

$$G_1(z) = G_{10}(z^2) + z^{-1}G_{11}(z^2) \quad (14.102)$$

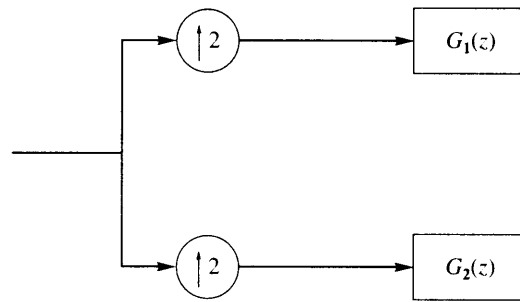
$$G_2(z) = G_{20}(z^2) + z^{-1}G_{21}(z^2). \quad (14.103)$$

Consider the output of the synthesis filter  $G_1(z)$  given an input  $Y_1(z)$ . From Equation (14.41), the output of the upsampler is

$$U_1(z) = Y_1(z^2) \quad (14.104)$$



**FIGURE 14. 24** Polyphase representation of the analysis portion of a two-band subband coder.



**FIGURE 14. 25** The synthesis portion of a two-band subband coder.

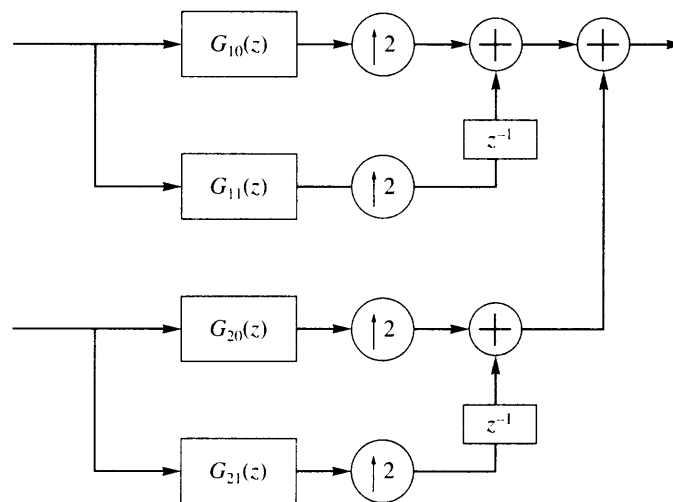
and the output of  $G_1(z)$  is

$$V_1(z) = Y_1(z^2)G_1(z) \quad (14.105)$$

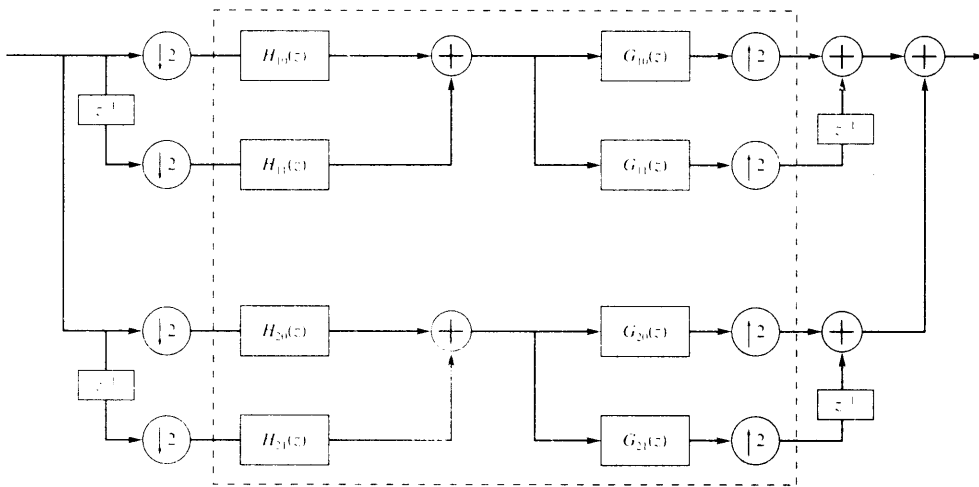
$$= Y_1(z^2)G_{10}(z^2) + z^{-1}Y_1(z^2)G_{11}(z^2). \quad (14.106)$$

The first term in the equation above is the output of an upsampler that *follows* a filter with transfer function  $G_{10}(z)$  with input  $Y(z)$ . Similarly,  $Y_1(z^2)G_{11}(z^2)$  is the output of an upsampler that follows a filter with transfer function  $G_{11}(z)$  with input  $Y(z)$ . Thus, this system can be represented as shown in Figure 14.26.

Putting the polyphase representations of the analysis and synthesis portions together, we get the system shown in Figure 14.27. Looking at the portion in the dashed box, we can see that this is a completely linear time-invariant system.



**FIGURE 14. 26** Polyphase representation of the synthesis portion of a two-band subband coder.



**FIGURE 14. 27 Polyphase representation of the two-band subband coder.**

The polyphase representation can be a very useful tool for the design and analysis of filters. While many of its uses are beyond the scope of this chapter, we can use this representation to prove our statement about the two-band perfect reconstruction QMF filters.

Recall that we want

$$T(z) = \frac{1}{2} [H_1(z)H_2(-z) - H_1(-z)H_2(z)] = cz^{-n_0}.$$

If we impose the mirror condition  $H_2(z) = H_1(-z)$ ,  $T(z)$  becomes

$$T(z) = \frac{1}{2} [H_1^2(z) - H_1^2(-z)]. \tag{14.107}$$

The polyphase decomposition of  $H_1(z)$  is

$$H_1(z) = H_{10}(z^2) + z^{-1}H_{11}(z^2).$$

Substituting this into Equation (14.107) for  $H_1(z)$  and

$$H_1(-z) = H_{10}(z^2) - z^{-1}H_{11}(z^2)$$

for  $H_1(-z)$ , we obtain

$$T(z) = 2z^{-1}H_{10}(z^2)H_{11}(z^2). \tag{14.108}$$

Clearly, the only way  $T(z)$  can have the form  $cz^{-n_0}$  is if both  $H_{10}(z)$  and  $H_{11}(z)$  are simple delays; that is,

$$H_{10}(z) = h_0z^{-k_0} \tag{14.109}$$

$$H_{11}(z) = h_1z^{-k_1}. \tag{14.110}$$



This results in

$$T(z) = 2h_0h_1z^{-(2k_0+2k_1+1)} \quad (14.111)$$

which is of the form  $cz^{-n_0}$  as desired. The resulting filters have the transfer functions

$$H_1(z) = h_0z^{-2k_0} + h_1z^{-(2k_1+1)} \quad (14.112)$$

$$H_2(z) = h_0z^{-2k_0} - h_1z^{-(2k_1+1)}. \quad (14.113)$$

## 14.9 Bit Allocation

Once we have separated the source output into the constituent sequences, we need to decide how much of the coding resource should be used to encode the output of each synthesis filter. In other words, we need to allocate the available bits between the subband sequences. In the previous chapter we described a bit allocation procedure that uses the variances of the transform coefficient. In this section we describe a bit allocation approach that attempts to use as much information about the subbands as possible to distribute the bits.

Let's begin with some notation. We have a total of  $B_T$  bits that we need to distribute among  $M$  subbands. Suppose  $R$  corresponds to the average rate in bits per sample for the overall system, and  $R_k$  is the average rate for subband  $k$ . Let's begin with the case where the input is decomposed into  $M$  equal bands, each of which is decimated by a factor of  $M$ . Finally, let's assume that we know the rate distortion function for each band. (If you recall from Chapter 8, this is a rather strong assumption and we will relax it shortly.) We also assume that the distortion measure is such that the total distortion is the sum of the distortion contribution of each band.

We want to find the bit allocation  $R_k$  such that

$$R = \frac{1}{M} \sum_{k=1}^M R_k \quad (14.114)$$

and the reconstruction error is minimized. Each value of  $R_k$  corresponds to a point on the rate distortion curve. The question is where on the rate distortion curve for each subband should we operate to minimize the average distortion. There is a trade-off between rate and distortion. If we decrease the rate (that is, move down the rate distortion curve), we will increase the distortion. Similarly, if we want to move to the left on the rate distortion curve and minimize the distortion, we end up increasing the rate. We need a formulation that incorporates both rate and distortion and the trade-off involved. The formulation we use is based on a landmark paper in 1988 by Yaacov Shoham and Allen Gersho [204]. Let's define a functional  $J_k$ :

$$J_k = D_k + \lambda R_k \quad (14.115)$$

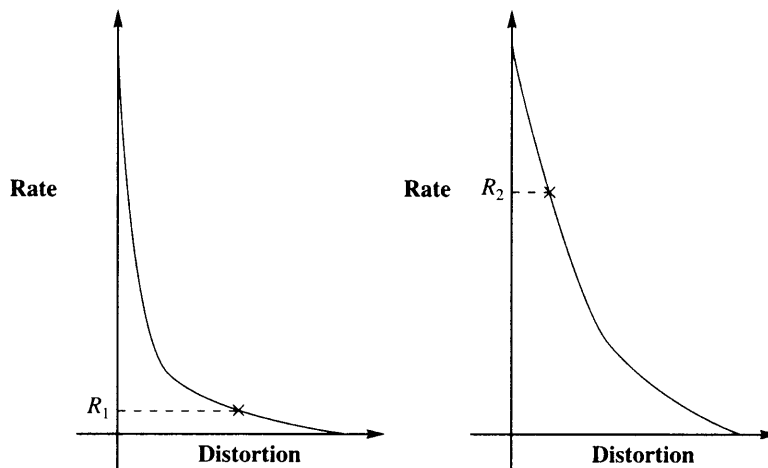
where  $D_k$  is the distortion contribution from the  $k$ th subband and  $\lambda$  is a Lagrangian parameter. This is the quantity we wish to minimize. In this expression the parameter  $\lambda$  in some sense specifies the trade-off. If we are primarily interested in minimizing the distortion, we can set  $\lambda$  to a small value. If our primary interest is in minimizing the rate, we keep the value of

$\lambda$  large. We can show that the values of  $D_k$  and  $R_k$  that minimize  $J_k$  occur where the slope of the rate distortion curve is  $\lambda$ . Thus, given a value of  $\lambda$  and the rate distortion function, we can immediately identify the values of  $R_k$  and  $D_k$ . So what should the value of  $\lambda$  be, and how should it vary between subbands?

Let's take the second question first. We would like to allocate bits in such a way that any increase in any of the rates will have the same impact on the distortion. This will happen when we pick  $R_k$  in such a way that the slopes of the rate distortion functions for the different subbands are the same; that is, we want to use the same  $\lambda$  for each subband. Let's see what happens if we do not. Consider the two rate distortion functions shown in Figure 14.28. Suppose the points marked  $x$  on the rate distortion functions correspond to the selected rates. Obviously, the slopes, and hence the values of  $\lambda$ , are different in the two cases. Because of the differences in the slope, an increase by  $\Delta R$  in the rate  $R_1$  will result in a much larger decrease in the distortion than the increase in distortion if we decreased  $R_2$  by  $\Delta R$ . Because the total distortion is the sum of the individual distortions, we can therefore reduce the overall distortions by increasing  $R_1$  and decreasing  $R_2$ . We will be able to keep doing this until the slope corresponding to the rates are the same in both cases. Thus, the answer to our second question is that we want to use the same value of  $\lambda$  for all the subbands.

Given a set of rate distortion functions and a value of  $\lambda$ , we automatically get a set of rates  $R_k$ . We can then compute the average and check if it satisfies our constraint on the total number of bits we can spend. If it does not, we modify the value of  $\lambda$  until we get a set of rates that satisfies our rate constraint.

However, generally we do not have rate distortion functions available. In these cases we use whatever is available. For some cases we might have *operational* rate distortion curves available. By "operational" we mean performance curves for particular types of encoders operating on specific types of sources. For example, if we knew we were going to be using *pdf*-optimized nonuniform quantizers with entropy coding, we could estimate the distribution of the subband and use the performance curve for *pdf*-optimized nonuniform quantizers for



**FIGURE 14. 28** Two rate distortion functions.

that distribution. We might only have the performance of the particular encoding scheme for a limited number of rates. In this case we need to have some way of obtaining the slope from a few points. We could estimate this numerically from these points. Or we could fit the points to a curve and estimate the slope from the curve. In these cases we might not be able to get exactly the average rate we wanted.

Finally, we have been talking about a situation where the number of samples in each subband is exactly the same, and therefore the total rate is simply the sum of the individual rates. If this is not true, we need to weight the rates of the individual subbands. The functional to be minimized becomes

$$J = \sum D_k + \lambda \sum \beta_k R_k \quad (14.116)$$

where  $\beta_k$  is the weight reflecting the relative length of the sequence generated by the  $k$ th filter. The distortion contribution from each subband might not be equally relevant, perhaps because of the filter construction or because of the perceptual weight attached to those frequencies [205]. In these cases we can modify our functional still further to include the unequal weighting of the distortion:

$$J = \sum w_k D_k + \lambda \sum \beta_k R_k. \quad (14.117)$$

## 14.10 Application to Speech Coding—G.722

The ITU-T recommendation G.722 provides a technique for wideband coding of speech signals that is based on subband coding. The basic objective of this recommendation is to provide high-quality speech at 64 kbits per second (kbps). The recommendation also contains two other modes that encode the input at 56 and 48 kbps. These two modes are used when an auxiliary channel is needed. The first mode provides for an auxiliary channel of 8 kbps; the second mode, for an auxiliary channel of 16 kbps.

The speech output or audio signal is filtered to 7 kHz to prevent aliasing, then sampled at 16,000 samples per second. Notice that the cutoff frequency for the anti-aliasing filter is 7 kHz, not 8 kHz, even though we are sampling at 16,000 samples per second. One reason for this is that the cutoff for the anti-aliasing filter is not going to be sharp like that of the ideal low-pass filter. Therefore, the highest frequency component in the filter output will be greater than 7 kHz. Each sample is encoded using a 14-bit uniform quantizer. This 14-bit input is passed through a bank of two 24-coefficient FIR filters. The coefficients of the low-pass QMF filter are shown in Table 14.7.

The coefficients for the high-pass QMF filter can be obtained by the relationship

$$h_{HP,n} = (-1)^n h_{LP,n}. \quad (14.118)$$

The low-pass filter passes all frequency components in the range of 0 to 4 kHz, while the high-pass filter passes all remaining frequencies. The output of the filters is downsampled by a factor of two. The downsampled sequences are encoded using adaptive differential PCM (ADPCM) systems.

The ADPCM system that encodes the downsampled output of the low-frequency filter uses 6 bits per sample, with the option of dropping 1 or 2 least significant bits in order to

**TABLE 14.7** Transmit and receive QMF coefficient values.

$h_0, h_{23}$	$3.66211 \times 10^{-4}$
$h_1, h_{22}$	$-1.34277 \times 10^{-3}$
$h_2, h_{21}$	$-1.34277 \times 10^{-3}$
$h_3, h_{20}$	$6.46973 \times 10^{-3}$
$h_4, h_{19}$	$1.46484 \times 10^{-3}$
$h_5, h_{18}$	$-1.90430 \times 10^{-2}$
$h_6, h_{17}$	$3.90625 \times 10^{-3}$
$h_7, h_{16}$	$4.41895 \times 10^{-2}$
$h_8, h_{15}$	$-2.56348 \times 10^{-2}$
$h_9, h_{14}$	$-9.82666 \times 10^{-2}$
$h_{10}, h_{13}$	$1.16089 \times 10^{-1}$
$h_{11}, h_{12}$	$4.73145 \times 10^{-1}$

provide room for the auxiliary channel. The output of the high-pass filter is encoded using 2 bits per sample. Because the 2 least significant bits of the quantizer output of the low-pass ADPCM system could be dropped and then not available to the receiver, the adaptation and prediction at both the transmitter and receiver are performed using only the 4 most significant bits of the quantizer output.

If all 6 bits are used in the encoding of the low-frequency subband, we end up with a rate of 48 kbps for the low band. Since the high band is encoded at 2 bits per sample, the output rate for the high subband is 16 kbps. Therefore, the total output rate for the subband-ADPCM system is 64 kbps.

The quantizer is adapted using a variation of the Jayant algorithm [110]. Both ADPCM systems use the past two reconstructed values and the past six quantizer outputs to predict the next sample, in the same way as the predictor for recommendation G.726 described in Chapter 11. The predictor is adapted in the same manner as the predictor used in the G.726 algorithm.

At the receiver, after being decoded by the ADPCM decoder, each output signal is upsampled by the insertion of a zero after each sample. The upsampled signals are passed through the reconstruction filters. These filters are identical to the filters used for decomposing the signal. The low-pass reconstruction filter coefficients are given in Table 14.7, and the coefficients for the high-pass filter can be obtained using Equation (14.118).

### 14.11 Application to Audio Coding—MPEG Audio

The Moving Picture Experts Group (MPEG) has proposed an audio coding scheme that is based in part on subband coding. Actually, MPEG has proposed three coding schemes, called Layer I, Layer II, and Layer III coding. Each is more complex than the previous and provides higher compression. The coders are also “upward” compatible; a Layer  $N$  decoder is able to decode the bitstream generated by the Layer  $N - 1$  encoder. In this section we will look primarily at the Layer 1 and Layer 2 coders.

The Layer 1 and Layer 2 coders both use a bank of 32 filters, splitting the input into 32 bands, each with a bandwidth of  $f_s/64$ , where  $f_s$  is the sampling frequency. Allowable



**TABLE 14.9 Filtered and decimated output.**

Decimated Low-Pass Output				Decimated High-Pass Output			
5	12	13	11	5	-2	1	3
5	10	11	8	5	-2	-1	2
6	9	7	11	6	-1	1	1
4	5	5	7	4	-1	-1	1
7	11	7	5	7	-1	-1	1
6	10	8	6	6	2	-2	0
6	8	6	6	6	-2	0	0
3	6	6	6	3	0	0	0

**TABLE 14.10 Four subimages.**

Low-Low Image				Low-High Image			
2.5	6	6.5	5.5	2.5	6	6.5	5.5
5.5	9.5	9	9.5	0.5	-0.5	-2	1.5
5.5	8	6	6	1.5	3	1	-1
6	9	7	6	0	-1	-1	0
High-Low Image				High-High Image			
2.5	-1	0.5	1.5	2.5	-1	0.5	1.5
5.5	-1.5	0	1.5	0.5	0.5	1	-0.5
5.5	-1	-1	1	1.5	0	0	0
6	0	-1	0	0	-2	1	0

is the top element in each row. We assume that there is a zero row of pixels right above this row in order to provide the filter with "past" values. After filtering and decimation, we get four subimages (Table 14.10). The subimage obtained by low-pass filtering of the columns of the subimage (which was the output of the row low-pass filtering) is called the low-low (LL) image. Similarly, the other images are called the low-high (LH), high-low (HL), and high-high (HH) images. ♦

If we look closely at the final set of subimages in the previous example, we notice that there is a difference in the characteristics of the values in the left or top row and the interiors of some of the subimages. For example, in the high-low subimage, the values in the first column are significantly larger than the other values in the subimage. Similarly, in the low-high subimage, the values in the first row are generally very different than the other values in the subimage. The reason for this variance is our assumption that the "past" of the image above the first row and to the left of the column was zero. The difference between zero and the image values was much larger than the normal pixel-to-pixel differences. Therefore, we ended up adding some spurious structure to the image reflected in the subimages. Generally, this is undesirable because it is easier to select appropriate compression schemes when the characteristics of the subimages are as uniform as possible. For example, if we did not have

**TABLE 14.11** Alternate four subimages.

Low-Low Image				Low-High Image			
10	12	13	11	0	0	-0.5	-0.5
11	9.5	9	9.5	1	-0.5	-2	1.5
11	8	6	6	3	3	1	-1
12	9	7	6	0	-1	-1	0
High-Low Image				High-High Image			
0	-2	1	3	0	0	0	0
0	-1.5	0	1.5	0	0.5	1	-0.5
0	-1	-1	1	0	0	0	0
0	0	-1	0	0	-2	1	0

the relatively large values in the first column of the high-low subimage, we could choose a quantizer with a smaller step size.

In this example, this effect was limited to a single row or column because the filters used a single past value. However, most filters use a substantially larger number of past values in the filtering operation, and a larger portion of the subimage is affected.

We can avoid this problem by assuming a different “past.” There are a number of ways this can be done. A simple method that works well is to reflect the values of the pixels at the boundary. For example, for the sequence 6 9 5 4 7 2 . . . , which was to be filtered with a three-tap filter, we would assume the past as 9 6 6 9 5 4 7 2 . . . . If we use this approach for the image in Example 14.12.1, the four subimages would be as shown in Table 14.11.

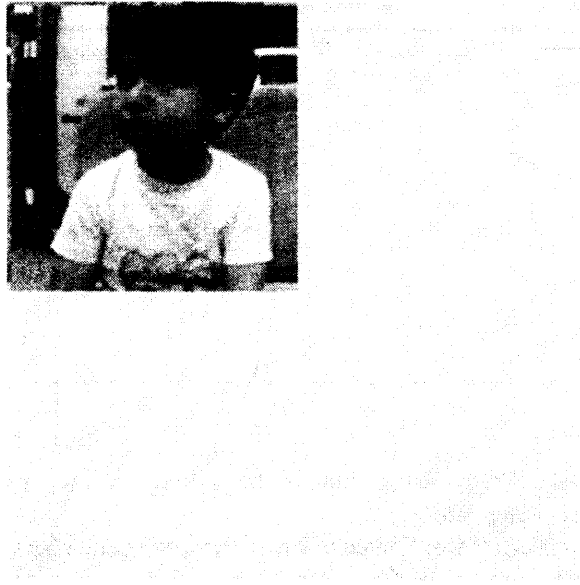
Notice how much sparser each image is, except for the low-low image. Most of the energy in the original image has been compacted into the low-low image. Since the other subimages have very few values that need to be encoded, we can devote most of our resources to the low-low subimage.

### 14.12.1 Decomposing an Image

Earlier a set of filters was provided to be used in one-dimensional subband coding. We can use those same filters to decompose an image into its subbands.

#### Example 14.12.2:

Let’s use the eight-tap Johnston filter to decompose the Sinan image into four subbands. The results of the decomposition are shown in Figure 14.29. Notice that, as in the case of the image in Example 14.12.1, most of the signal energy is concentrated in the low-low subimage. However, there remains a substantial amount of energy in the higher bands. To see this more clearly, let’s look at the decomposition using the 16-tap Johnston filter. The results are shown in Figure 14.30. Notice how much less energy there is in the higher



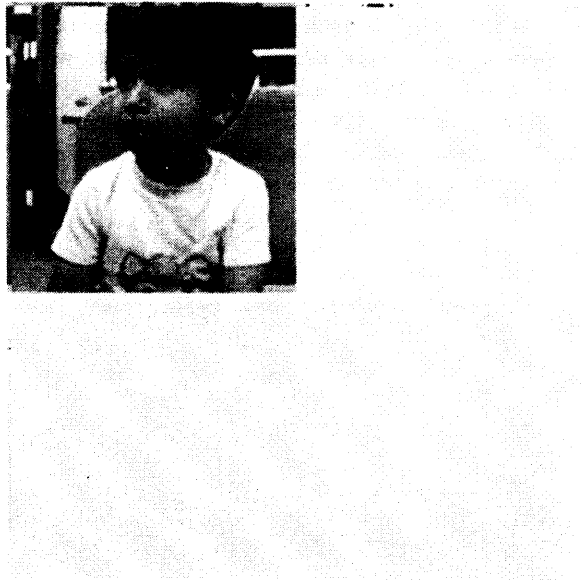
**FIGURE 14. 29** Decomposition of Sinan image using the eight-tap Johnston filter.



**FIGURE 14. 30** Decomposition of Sinan image using the 16-tap Johnston filter.



subbands. In fact, the high-high subband seems completely empty. As we shall see later, this difference in *energy compaction* can have a drastic effect on the reconstruction.



**FIGURE 14.31** Decomposition of Sinan image using the the eight-tap Smith-Barnwell filter.

Increasing the size of the filter is not necessarily the only way of improving the energy compaction. Figure 14.31 shows the decomposition obtained using the eight-tap Smith-Barnwell filter. The results are almost identical to the 16-tap Johnston filter. Therefore, rather than increase the computational load by going to a 16-tap filter, we can keep the same computational load and simply use a different filter. ♦

### 14.12.2 Coding the Subbands

Once we have decomposed an image into subbands, we need to find the best encoding scheme to use with each subband. The coding schemes we have studied to date are scalar quantization, vector quantization, and differential encoding. Let us encode some of the decomposed images from the previous section using two of the coding schemes we have studied earlier, scalar quantization and differential encoding.

#### Example 14.12.3:

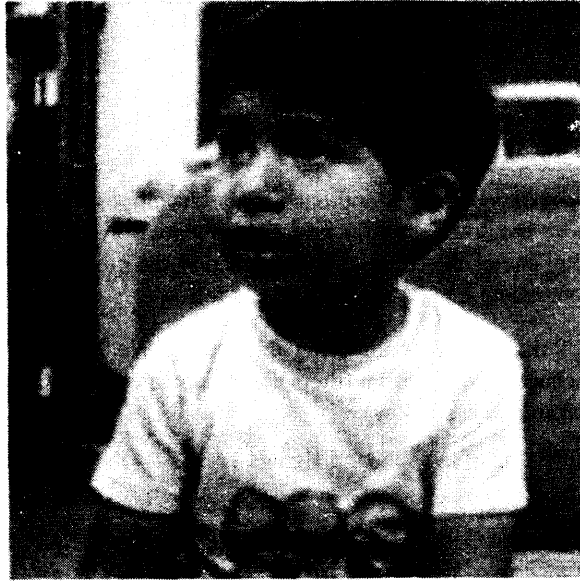
In the previous example we noted the fact that the eight-tap Johnston filter did not compact the energy as well as the 16-tap Johnston filter or the eight-tap Smith-Barnwell filter. Let's see how this affects the encoding of the decomposed images.

When we encode these images at an average rate of 0.5 bits per pixel, there are  $4 \times 0.5 = 2$  bits available to encode four values, one value from each of the four subbands. If we use the recursive bit allocation procedure on the eight-tap Johnston filter outputs, we end up allocating 1 bit to the low-low band and 1 bit to the high-low band. As the pixel-to-pixel difference in the low-low band is quite small, we use a DPCM encoder for the low-low band. The high-low band does not show this behavior, which means we can simply use scalar quantization for the high-low band. As there are no bits available to encode the other two bands, these bands can be discarded. This results in the image shown in Figure 14.32, which is far from pleasing. However, if we use the same compression approach with the image decomposed using the eight-tap Smith-Barnwell filter, the result is Figure 14.33, which is much more pleasing.



**FIGURE 14. 32** Sinan image coded at 0.5 bits per pixel using the eight-tap Johnston filter.

To understand why we get such different results from using the two filters, we need to look at the way the bits were allocated to the different bands. In this implementation, we used the recursive bit allocation algorithm. In the image decomposed using the Johnston filter, there was significant energy in the high-low band. The algorithm allocated 1 bit to the low-low band and 1 bit to the high-low band. This resulted in poor encoding for both, and subsequently poor reconstruction. There was very little signal content in any of the bands other than the low-low band for the image decomposed using the Smith-Barnwell filter. Therefore, the bit allocation algorithm assigned both bits to the low-low band, which provided a reasonable reconstruction.



**FIGURE 14.33** Sinan image coded at 0.5 bits per pixel using the eight-tap Smith-Barnwell filter.

If the problem with the encoding of the image decomposed by the Johnston filter is an insufficient number of bits for encoding the low-low band, why not simply assign both bits to the low-low band? The problem is that the bit allocation scheme assigned a bit to the high-low band because there was a significant amount of information in that band. If both bits were assigned to the low-low band, we would have no bits left for use in encoding the high-low band, and we would end up throwing away information necessary for the reconstruction. ♦

The issue of energy compaction becomes a very important factor in reconstruction quality. Filters that allow for more energy compaction permit the allocation of bits to a smaller number of subbands. This in turn results in a better reconstruction.

The coding schemes used in this example were DPCM and scalar quantization, the techniques generally preferred in subband coding. The advantage provided by subband coding is readily apparent if we compare the result shown in Figure 14.33 to results in the previous chapters where we used either DPCM or scalar quantization without prior decomposition.

It would appear that the subband approach lends itself naturally to vector quantization. After decomposing an image into subbands, we could design separate codebooks for each subband to reflect the characteristics of that particular subband. The only problem with this idea is that the low-low subband generally requires a large number of bits per pixel. As we mentioned in Chapter 10, it is generally not feasible to operate the nonstructured vector quantizers at high rates. Therefore, when vector quantizers are used, they are generally

used only for encoding the higher frequency bands. This may change as vector quantization algorithms that operate at higher rates are developed.

### 14.13 Summary

In this chapter we introduced another approach to the decomposition of signals. In subband coding we decompose the source output into components. Each of these components can then be encoded using one of the techniques described in the previous chapters. The general subband encoding procedure can be summarized as follows:

- Select a set of filters for decomposing the source. We have provided a number of filters in this chapter. Many more filters can be obtained from the published literature (we give some references below).
- Using the filters, obtain the subband signals  $\{y_{k,n}\}$ :

$$y_{k,n} = \sum_{i=0}^{N-1} h_{k,i} x_{n-i} \quad (14.119)$$

where  $\{h_{k,n}\}$  are the coefficients of the  $k$ th filter.

- Decimate the output of the filters.
- Encode the decimated output.

The decoding procedure is the inverse of the encoding procedure. When encoding images the filtering and decimation operations have to be performed twice, once along the rows and once along the columns. Care should be taken to avoid problems at edges, as described in Section 14.12.

#### Further Reading

1. *Handbook for Digital Signal Processing*, edited by S.K. Mitra and J.F. Kaiser [162], is an excellent source of information about digital filters.
2. *Multirate Systems and Filter Banks*, by P.P. Vaidyanathan [200], provides detailed information on QMF filters, as well as the relationship between wavelets and filter banks and much more.
3. The topic of subband coding is also covered in *Digital Coding of Waveforms*, by N.S. Jayant and P. Noll [123].
4. The MPEG-1 audio coding algorithm is described in "ISO-MPEG-1 Audio: A Generic Standard for Coding of High-Quality Digital Audio," by K. Brandenburg and G. Stoll [28], in the October 1994 issue of the *Journal of the Audio Engineering Society*.
5. A review of the rate distortion method of bit allocation is provided in "Rate Distortion Methods for Image and Video Compression," by A. Ortega and K. Ramachandran, in the November 1998 issue of *IEEE Signal Processing Magazine* [169].

## 14.14 Projects and Problems

1. A linear shift invariant system has the following properties:

- If for a given input sequence  $\{x_n\}$  the output of the system is the sequence  $\{y_n\}$ , then if we delay the input sequence by  $k$  units to obtain the sequence  $\{x_{n-k}\}$ , the corresponding output will be the sequence  $\{y_n\}$  delayed by  $k$  units.
- If the output corresponding to the sequence  $\{x_n^{(1)}\}$  is  $\{y_n^{(1)}\}$ , and the output corresponding to the sequence  $\{x_n^{(2)}\}$  is  $\{y_n^{(2)}\}$ , then the output corresponding to the sequence  $\{\alpha x_n^{(1)} + \beta x_n^{(2)}\}$  is  $\{\alpha y_n^{(1)} + \beta y_n^{(2)}\}$ .

Use these two properties to show the convolution property given in Equation (14.18).

2. Let's design a set of simple four-tap filters that satisfies the perfect reconstruction condition.

- (a) We begin with the low-pass filter. Assume that the impulse response of the filter is given by  $\{h_{1,k}\}_{k=0}^{k=3}$ . Further assume that

$$|h_{1,k}| = |h_{1,j}| \quad \forall j, k.$$

Find a set of values for  $\{h_{1,j}\}$  that satisfies Equation (14.91).

- (b) Plot the magnitude of the transfer function  $H_1(z)$ .
- (c) Using Equation (14.23), find the high-pass filter coefficients  $\{h_{2,k}\}$ .
- (d) Find the magnitude of the transfer function  $H_2(z)$ .

3. Given an input sequence

$$x_n = \begin{cases} (-1)^n & n = 0, 1, 2, \dots \\ 0 & \text{otherwise} \end{cases}$$

- (a) Find the output sequence  $y_n$  if the filter impulse response is

$$h_n = \begin{cases} \frac{1}{\sqrt{2}} & n = 0, 1 \\ 0 & \text{otherwise.} \end{cases}$$

- (b) Find the output sequence  $w_n$  if the impulse response of the filter is

$$h_n = \begin{cases} \frac{1}{\sqrt{2}} & n = 0 \\ -\frac{1}{\sqrt{2}} & n = 1 \\ 0 & \text{otherwise.} \end{cases}$$

- (c) Looking at the sequences  $y_n$  and  $w_n$ , what can you say about the sequence  $x_n$ ?

4. Given an input sequence

$$x_n = \begin{cases} 1 & n = 0, 1, 2, \dots \\ 0 & \text{otherwise} \end{cases}$$

- (a) Find the output sequence  $y_n$  if the filter impulse response is

$$h_n = \begin{cases} \frac{1}{\sqrt{2}} & n = 0, 1 \\ 0 & \text{otherwise.} \end{cases}$$

- (b) Find the output sequence  $w_n$  if the impulse response of the filter is

$$h_n = \begin{cases} \frac{1}{\sqrt{2}} & n = 0 \\ -\frac{1}{\sqrt{2}} & n = 1 \\ 0 & \text{otherwise.} \end{cases}$$

- (c) Looking at the sequences  $y_n$  and  $w_n$ , what can you say about the sequence  $x_n$ ?
5. Write a program to perform the analysis and downsampling operations and another to perform the upsampling and synthesis operations for an image compression application. The programs should read the filter parameters from a file. The synthesis program should read the output of the analysis program and write out the reconstructed images. The analysis program should also write out the subimages scaled so that they can be displayed. Test your program using the Johnston eight-tap filter and the Sena image.
6. In this problem we look at some of the many ways we can encode the subimages obtained after subsampling. Use the eight-tap Johnston filter to decompose the Sena image into four subimages.
- (a) Encode the low-low band using an adaptive delta modulator (CFDM or CVSD). Encode all other bands using a 1-bit scalar quantizer.
- (b) Encode the low-low band using a 2-bit adaptive DPCM system. Encode the low-high and high-low bands using a 1-bit scalar quantizer.
- (c) Encode the low-low band using a 3-bit adaptive DPCM system. Encode the low-high and high-low band using a 0.5 bit/pixel vector quantizer.
- (d) Compare the reconstructions obtained using the different schemes.

# Wavelet-Based Compression

## 15.1 Overview

**I**n 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

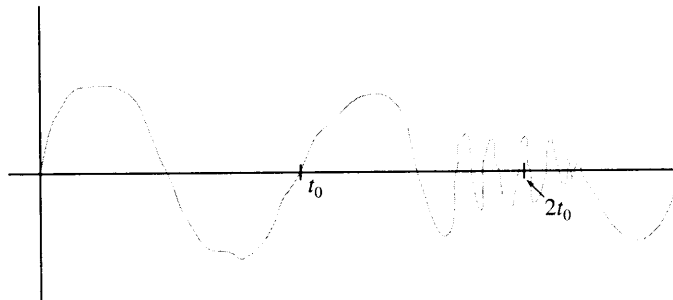
$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{j\omega t} dt.$$

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

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

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  $\sigma_t^2$  and the frequency spread  $\sigma_\omega^2$  is lower bounded by  $\sqrt{1/2}$ , where

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

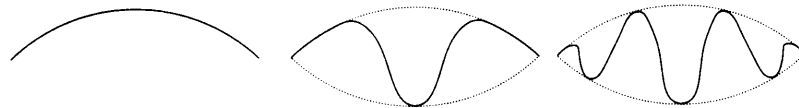
$$\sigma_\omega^2 = \frac{\int \omega^2 |G(\omega)|^2 d\omega}{\int |G(\omega)|^2 d\omega} \tag{15.3}$$

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

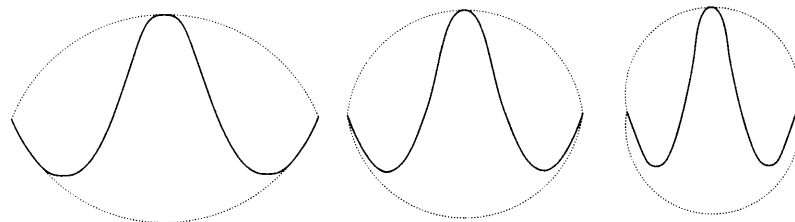
Let's take a look at the discrete STFT in terms of basis expansion, and for the moment, let's look at just one interval:

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

The basis functions are  $g(t)$ ,  $g(t)e^{j\omega_0 t}$ ,  $g(t)e^{j2\omega_0 t}$ , 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

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

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

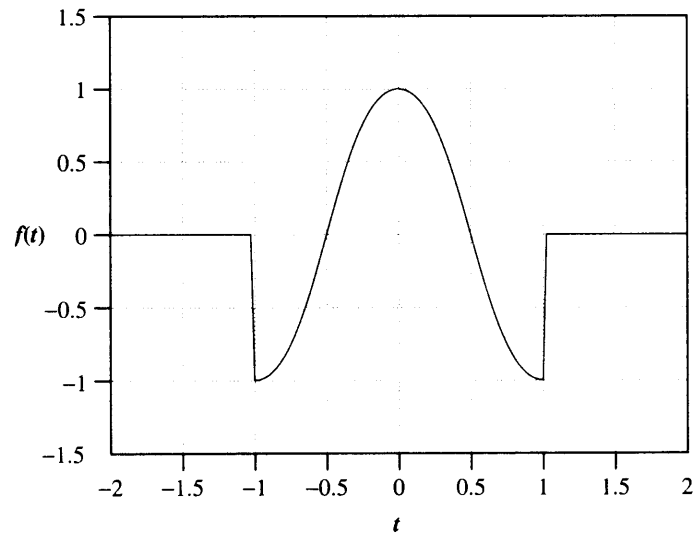
$$\begin{aligned} f\left(\frac{t}{0.5}\right) &= \begin{cases} \cos(\pi \frac{t}{0.5}) & -1 \leq \frac{t}{0.5} \leq 1 \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} \cos(2\pi t) & -\frac{1}{2} \leq t \leq \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

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

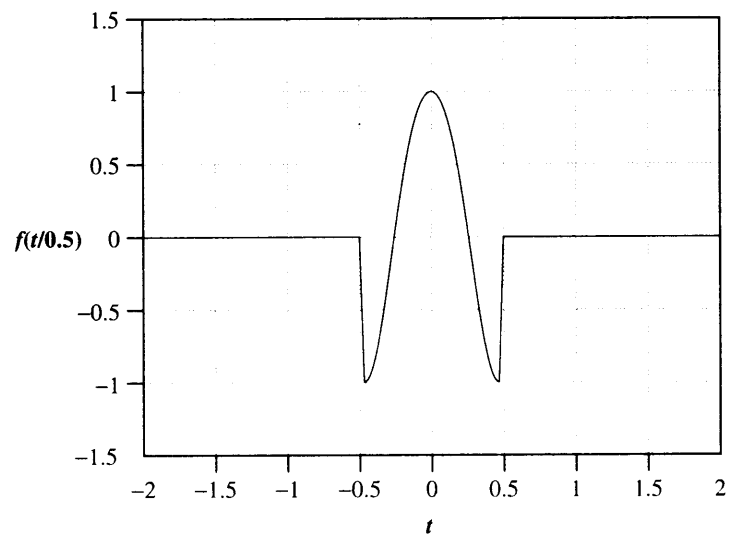
$$\|f(t)\|^2 = \int_{-\infty}^{\infty} f^2(t) dt$$

scaling obviously changes the norm of the function:

$$\begin{aligned} \left\|f\left(\frac{t}{a}\right)\right\|^2 &= \int_{-\infty}^{\infty} f^2\left(\frac{t}{a}\right) dt \\ &= a \int_{-\infty}^{\infty} f^2(x) dx \end{aligned}$$



**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,

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

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

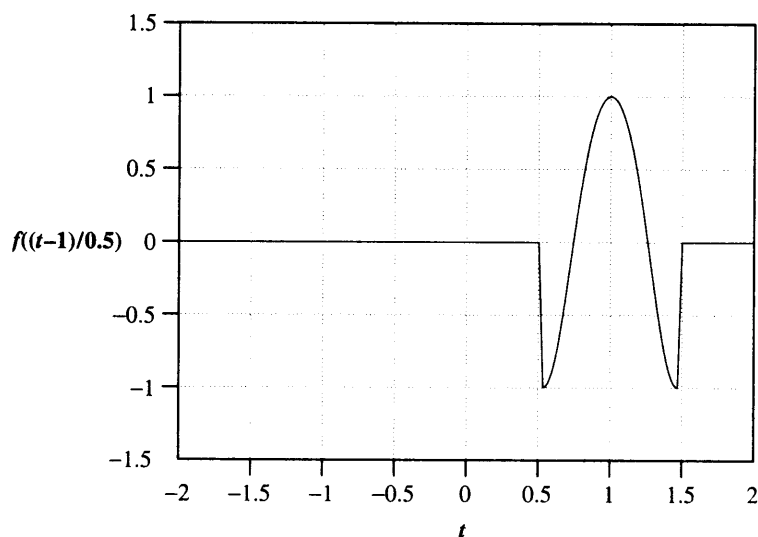
$$\begin{aligned} f\left(\frac{t-1}{0.5}\right) &= \begin{cases} \cos(2\pi(t-1)) & -\frac{1}{2} \leq t-1 \leq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \\ &= \begin{cases} \cos(2\pi(t-1)) & \frac{1}{2} \leq t \leq \frac{3}{2} \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

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

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

with Fourier transforms

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



**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:

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

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

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

where

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

For integral (15.8) to exist, we need  $C_\psi$  to be finite. For  $C_\psi$  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  $C_\psi$  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  $\omega$  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

$$a = a_0^{-m}, \quad b = nb_0 a_0^{-m} \quad (15.10)$$

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

$$\psi_{m,n}(t) = a_0^{m/2} \psi(a_0^m t - nb_0), \quad m, n \in \mathbb{Z}. \quad (15.11)$$

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

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

(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

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

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

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

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

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

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

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

$$f(t) = \sum_k a_k \phi(t - k). \quad (15.17)$$

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:

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

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

Let's define

$$\phi_k(t) = \phi(t-k). \quad (15.19)$$

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

$$f(t) = \sum_k a_k \phi_k(t) \quad (15.20)$$

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:

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

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,

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

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

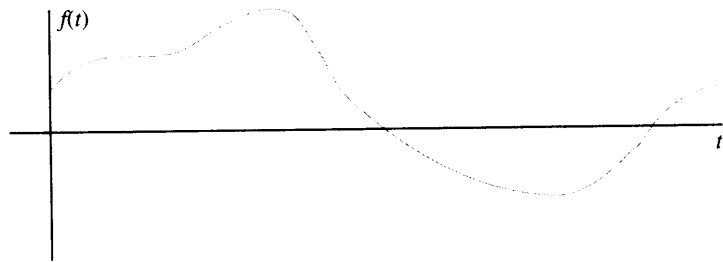
$$V_1 = \overline{\text{Span}\{\phi_{1,k}(t)\}} \quad (15.23)$$

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

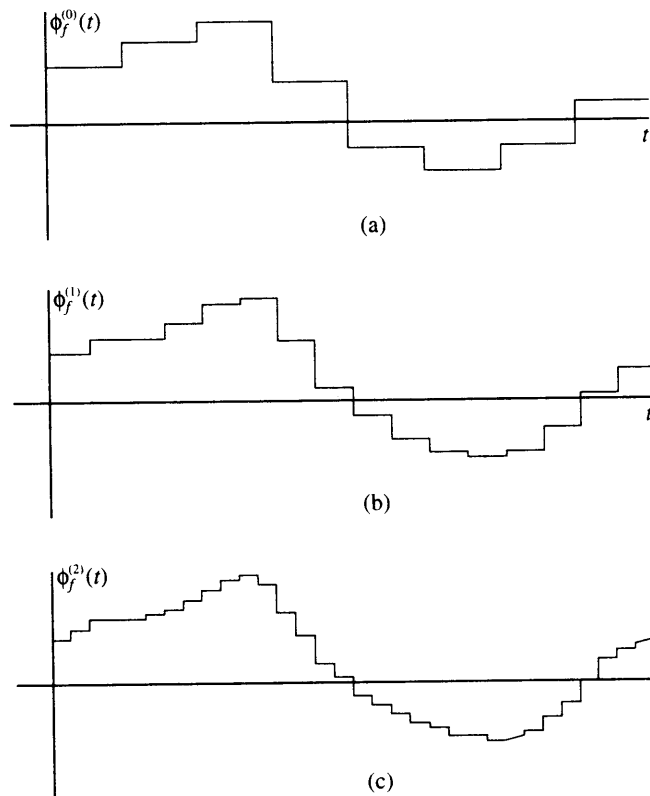
#### Example 15.4.1:

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

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



**FIGURE 15.7** A sample function.



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



where

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

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  $\{\phi_{1,k}(t)\}$ :

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

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

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

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

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

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

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

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:

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

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

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

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

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

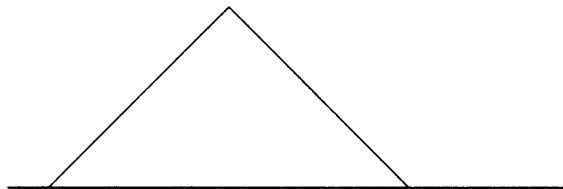
satisfies the recursion equation. ♦

**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  $[k, k+1)$ :

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

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

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

Defining

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

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

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

where

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

But this is simply the  $k$ th 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:

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

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

$$V_1 = V_0 \oplus W_0. \quad (15.37)$$

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

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

or

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

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

$$V_{j+1} = V_j \oplus W_j. \quad (15.40)$$

But, as  $j$  is arbitrary,

$$V_j = V_{j-1} \oplus W_{j-1} \quad (15.41)$$

and

$$V_{j+1} = V_{j-1} \oplus W_{j-1} \oplus W_j. \quad (15.42)$$

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

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

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

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

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

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

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

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