approximation input to the leftmost filter bank. To compute the $W_\psi(1, k)$ coefficients that appear at the end of the upper branch of Fig. 7.17, for example, we first convolve $f(n)$ with $h_\psi(-n)$. As explained in Section 4.6.3, this requires flipping one of the functions about the origin, sliding it past the other, and computing the sum of the point-to-point product of the two functions. For sequences $\{1, 4, -3, 0\}$ and $\{-1/\sqrt{2}, 1/\sqrt{2}\}$, this produces $\{-1/\sqrt{2}, -3/\sqrt{2}, 7/\sqrt{2}, -3/\sqrt{2}, 0\}$, where the second term corresponds to index $k = 2n = 0$. (In Fig. 7.17, underlined values represent negative indices, i.e., $n < 0$.) When downsampled by taking the even-indexed points, we get $W_\psi(1, k) = \{-3/\sqrt{2}, -3/\sqrt{2}\}$ for $k = \{0, 1\}$. We can alternately use Eq. (7.4-9) to compute

$$\begin{aligned} W_\psi(1, k) &= h_\psi(-n) * W_\varphi(2, n)\Big|_{n=2k, k\geq 0} = h_\psi(-n) * f(n)\Big|_{n=2k, k\geq 0} \\ &= \sum_l h_\psi(2k - l)x(l)\Big|_{k=0,1} \\ &= \frac{1}{\sqrt{2}}x(2k) - \frac{1}{\sqrt{2}}x(2k+1)\Big|_{k=0,1}. \end{aligned}$$

Here, we have substituted $2k$ for n in the convolution and employed l as a dummy variable of convolution (i.e., for displacing the two sequences relative to one another). There are only two terms in the expanded sum because there are only two nonzero values in the time-reversed wavelet vector $h_\psi(-n)$. Substituting $k = 0$, we find that $W_\psi(1, 0) = -3/\sqrt{2}$; for $k = 1$, we get $W_\psi(1, 1) = -3/\sqrt{2}$. Thus, the filtered and downsampled sequence is $\{-3/\sqrt{2}, -3/\sqrt{2}\}$, which matches the earlier result. The remaining convolutions and downsamplings are performed in a similar manner. ■

As one might expect, an equally efficient inverse transform for the reconstruction of $f(x)$ from DWT/FWT approximation and detail coefficients, $W_\varphi(j, k)$ and $W_\psi(j, k)$, can also be formulated. Called the *inverse fast wavelet transform* (FWT^{-1}), it uses the scaling and wavelet vectors employed in the forward transform, together with the level j approximation and detail coefficients, to generate the level $j + 1$ approximation coefficients. Noting the similarity between the FWT analysis bank in Fig. 7.15 and the two-band subband analysis portion of Fig. 7.4(a), we can immediately postulate the required FWT^{-1} *synthesis filter bank*. Figure 7.18 details its structure, which is identical to the synthesis portion of the two-band subband coding and decoding system in Fig. 7.4(a). Equation (7.1-23) of Section 7.1.2 defines the

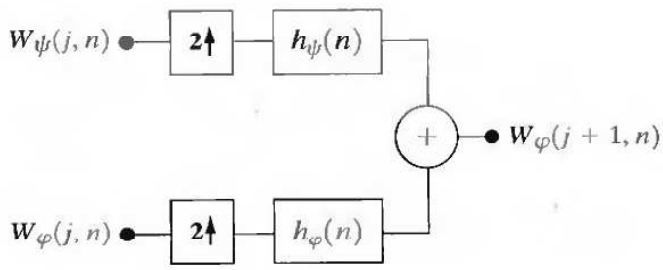


FIGURE 7.18 The FWT^{-1} synthesis filter bank.

relevant synthesis filters. As noted there, perfect reconstruction (for two-band orthonormal filters) requires $g_i(n) = h_i(-n)$ for $i = \{0, 1\}$. That is, the synthesis and analysis filters must be time-reversed versions of one another. Since the FWT analysis filters (see Fig. 7.15) are $h_0(n) = h_\varphi(-n)$ and $h_1(n) = h_\psi(-n)$, the required FWT^{-1} synthesis filters are $g_0(n) = h_0(-n) = h_\varphi(n)$ and $g_1(n) = h_1(-n) = h_\psi(n)$. It should be remembered, however, that it is also possible to use biorthogonal analysis and synthesis filters, which are not time-reversed versions of one another. Biorthogonal analysis and synthesis filters are cross-modulated per Eqs. (7.1-14) and (7.1-15).

The FWT^{-1} filter bank in Figure 7.18 implements the computation

$$W_\varphi(j+1, k) = h_\varphi(k) * W_\varphi^{\text{up}}(j, k) + h_\psi(k) * W_\psi^{\text{up}}(j, k)\Big|_{k \geq 0} \tag{7.4-13}$$

where W^{up} signifies upsampling by 2 (i.e., inserting zeros between the elements of W so that it is twice its original length). The upsampled coefficients are filtered, by convolution with $h_\varphi(n)$ and $h_\psi(n)$, and added to generate a higher scale approximation. In essence, a better approximation of $f(x)$, with greater detail and resolution, is created. As with the forward FWT, the inverse filter bank can be iterated as shown in Fig. 7.19, where a two-scale structure for computing the final two scales of a FWT^{-1} reconstruction is depicted. This coefficient combining process can be extended to any number of scales and guarantees perfect reconstruction of function $f(x)$.

EXAMPLE 7.11: Computing a 1-D inverse fast wavelet transform.

■ Computation of the inverse fast wavelet transform mirrors its forward counterpart. Figure 7.20 illustrates the process for the sequence considered in Example 7.10. To begin the calculation, the level 0 approximation and detail coefficients are upsampled to yield $\{1, 0\}$ and $\{4, 0\}$, respectively. Convolution with filters $g_0(n) = h_\varphi(n) = \{1/\sqrt{2}, 1/\sqrt{2}\}$ and $g_1(n) = h_\psi(n) = \{1/\sqrt{2}, -1/\sqrt{2}\}$ produces $\{1/\sqrt{2}, 1/\sqrt{2}, 0\}$ and $\{4/\sqrt{2}, -4/\sqrt{2}, 0\}$, which when added give $W_\varphi(1, n) = \{5/\sqrt{2}, -3/\sqrt{2}\}$. Thus, the level 1 approximation of Fig. 7.20, which matches the computed approximation in Fig. 7.17, is reconstructed. Continuing in this manner, $f(n)$ is formed at the right of the second synthesis filter bank. ■

We conclude our discussion of the fast wavelet transform by noting several differences between the FWT and the FFT—the first being their numerical complexity. The number of mathematical operations involved in the computa-

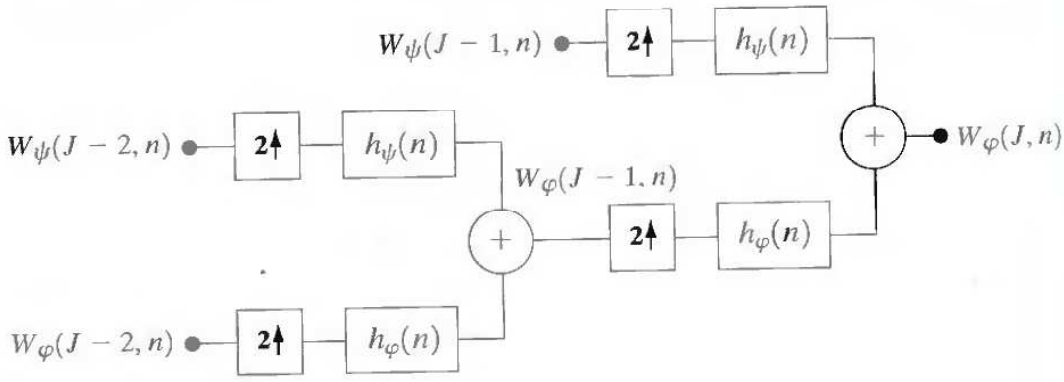


FIGURE 7.19 A two-stage or two-scale FWT^{-1} synthesis bank.

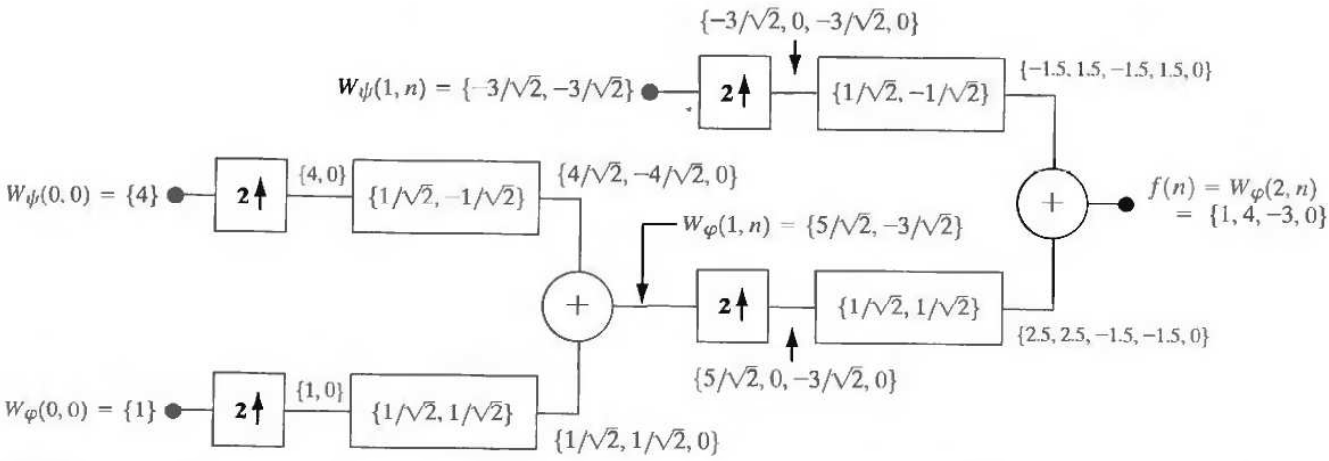


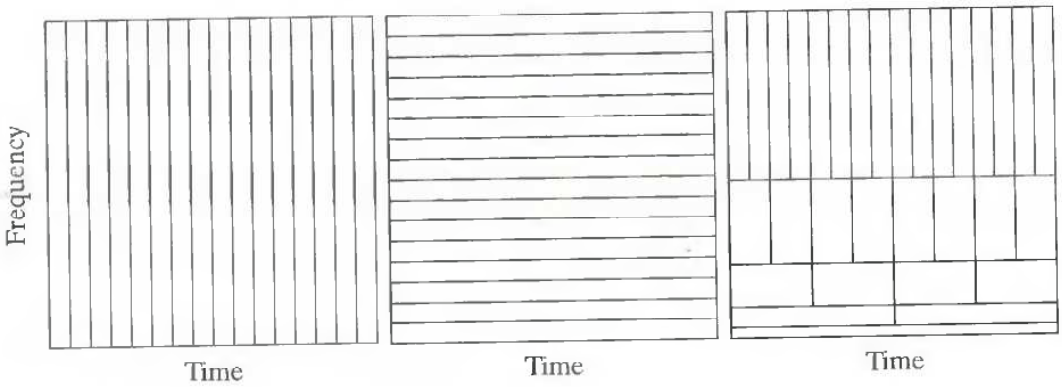
FIGURE 7.20 Computing a two-scale inverse fast wavelet transform of sequence $\{1, 4, -1.5\sqrt{2}, -1.5\sqrt{2}\}$ with Haar scaling and wavelet vectors.

tion of the FWT of a length $M = 2^J$ sequence is on the order of $O(M)$. That is, the number of floating-point multiplications and additions (using filter banks) is linear with respect to the length of the sequence. This compares favorably with the FFT algorithm, which requires $O(M \log M)$.

The second difference relates to the transforms' basis functions. While the Fourier basis functions (i.e., sinusoids) guarantee the existence of the FFT, the existence of the FWT depends upon the availability of a scaling function for the wavelets being used, as well as the orthogonality (or biorthogonality) of the scaling function and corresponding wavelets. Thus, the Mexican hat wavelet of Eq. (7.3-12), which does not have a companion scaling function, cannot be used in the computation of the FWT. In other words, we cannot construct a filter bank like that of Fig. 7.15 for the Mexican hat wavelet; it does not satisfy the underlying assumptions of the FWT approach.

Finally, we note that while time and frequency are usually viewed as different domains when representing functions, they are inextricably linked. When you try to analyze a function simultaneously in time and frequency, you run into the following problem: If you want precise information about time, you must put up with some vagueness about frequency, and vice versa. This is the *Heisenberg uncertainty principle* applied to information processing. To illustrate graphically the principle, each basis function used in the representation of a function is viewed schematically as a *tile* in a *time-frequency plane*. The tile, also called a *Heisenberg cell* or *Heisenberg box*, shows where the basis function's energy is concentrated. Basis functions that are orthonormal are characterized by nonoverlapping tiles.

Figure 7.21 shows the time-frequency tiles for (a) a delta function (i.e., conventional time domain) basis, (b) a sinusoidal (FFT) basis, and (c) an FWT basis. Note that the standard time domain basis pinpoints the instants when events occur but provides no frequency information. A sinsusoidal basis, on the other hand, pinpoints the frequencies that are present in events that occur over long periods but provides no time resolution. The time and frequency resolution of



a b c

FIGURE 7.21 Time-frequency tilings for (a) sampled data, (b) FFT, and (c) FWT basis functions.

the FWT tiles vary, but the area of each tile is the same. That is, each tile represents an equal portion of the time-frequency plane. At low frequencies, the tiles are shorter (i.e., have better frequency resolution or less ambiguity regarding frequency) but are wider (which corresponds to poorer time resolution or more ambiguity regarding time). At high frequencies, tile width is smaller (so the time resolution is improved) and tile height is greater (which means the frequency resolution is poorer). This fundamental difference between the FFT and FWT was noted in the introduction to the chapter and is important in the analysis of nonstationary functions whose frequencies vary in time.

7.5 Wavelet Transforms in Two Dimensions

The one-dimensional transforms of the previous sections are easily extended to two-dimensional functions like images. In two dimensions, a two-dimensional scaling function, $\varphi(x, y)$, and three two-dimensional wavelets, $\psi^H(x, y)$, $\psi^V(x, y)$, and $\psi^D(x, y)$, are required. Each is the product of a one-dimensional scaling function φ and corresponding wavelet ψ. Excluding products that produce one-dimensional results, like $\varphi(x)\psi(x)$, the four remaining products produce the *separable* scaling function

$$\varphi(x, y) = \varphi(x)\varphi(y) \tag{7.5-1}$$

and separable, "directionally sensitive" wavelets

$$\psi^H(x, y) = \psi(x)\varphi(y) \tag{7.5-2}$$

$$\psi^V(x, y) = \varphi(x)\psi(y) \tag{7.5-3}$$

$$\psi^D(x, y) = \psi(x)\psi(y). \tag{7.5-4}$$

These wavelets measure functional variations—intensity or gray-level variations for images—along different directions: ψ^H measures variations along columns (for example, horizontal edges), ψ^V responds to variations along rows (like vertical edges), and ψ^D corresponds to variations along diagonals. The directional sensitivity is a natural consequence of the separability imposed by

Eqs. (7.5-2) to (7.5-4); it does not increase the computational complexity of the two-dimensional transform discussed in this section.

Given separable two-dimensional scaling and wavelet functions, extension of the one-dimensional DWT to two dimensions is straightforward. We first define the scaled and translated basis functions:

$$\varphi_{j,m,n}(x, y) = 2^{j/2}\varphi(2^j x - m, 2^j y - n), \tag{7.5-5}$$

$$\psi^i_{j,m,n}(x, y) = 2^{j/2}\psi^i(2^j x - m, 2^j y - n), \qquad i = \{H, V, D\} \tag{7.5-6}$$

where index i identifies the directional wavelets in Eqs. (7.5-2) to (7.5-4). Rather than an exponent, i is a superscript that assumes the values H, V, and D. The discrete wavelet transform of function $f(x, y)$ of size $M \times N$ is then

$$W_\varphi(j_0, m, n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\varphi_{j_0,m,n}(x, y) \tag{7.5-7}$$

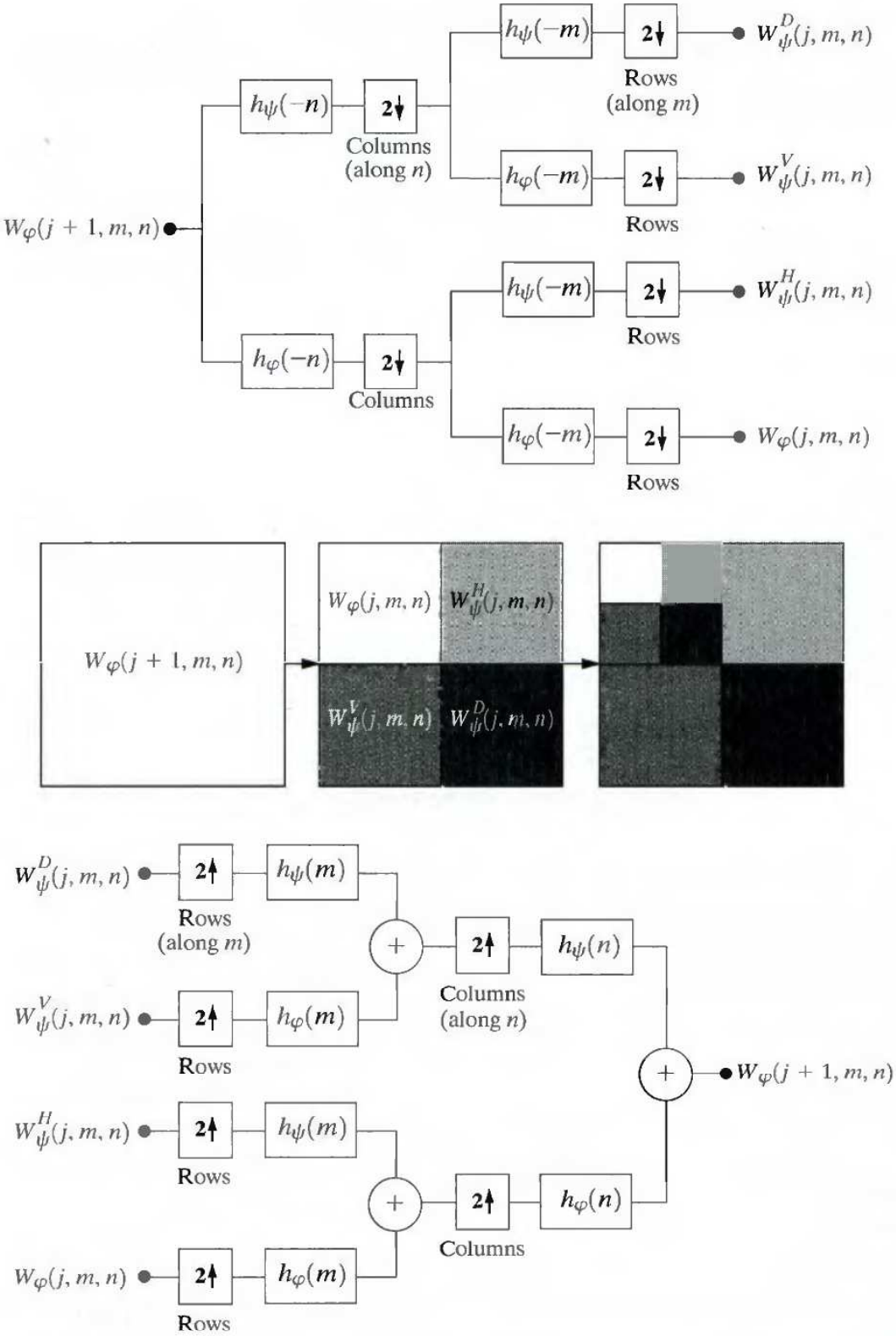
$$W^i_\psi(j, m, n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\psi^i_{j,m,n}(x, y) \qquad i = \{H, V, D\}. \tag{7.5-8}$$

As in the one-dimensional case, j_0 is an arbitrary starting scale and the $W_\varphi(j_0, m, n)$ coefficients define an approximation of $f(x, y)$ at scale j_0. The $W^i_\psi(j, m, n)$ coefficients add horizontal, vertical, and diagonal details for scales $j \geq j_0$. We normally let $j_0 = 0$ and select $N = M = 2^J$ so that $j = 0, 1, 2, \ldots, J - 1$ and $m, n = 0, 1, 2, \ldots, 2^j - 1$. Given the W_φ and W^i_ψ of Eqs. (7.5-7) and (7.5-8), $f(x, y)$ is obtained via the inverse discrete wavelet transform

$$f(x, y) = \frac{1}{\sqrt{MN}} \sum_m \sum_n W_\varphi(j_0, m, n)\varphi_{j_0,m,n}(x, y) + \frac{1}{\sqrt{MN}} \sum_{i=H,V,D} \sum_{j=j_0}^{\infty} \sum_m \sum_n W^i_\psi(j, m, n)\psi^i_{j,m,n}(x, y). \tag{7.5-9}$$

Like the one-dimensional discrete wavelet transform, the two-dimensional DWT can be implemented using digital filters and downsamplers. With separable two-dimensional scaling and wavelet functions, we simply take the one-dimensional FWT of the rows of $f(x, y)$, followed by the one-dimensional FWT of the resulting columns. Figure 7.22(a) shows the process in block diagram form. Note that, like its one-dimensional counterpart in Fig. 7.15, the two-dimensional FWT "filters" the scale $j + 1$ approximation coefficients to construct the scale j approximation and detail coefficients. In the two-dimensional case, however, we get three sets of detail coefficients—the horizontal, vertical, and diagonal details.

The single-scale filter bank of Fig. 7.22(a) can be "iterated" (by tying the approximation output to the input of another filter bank) to produce a P scale transform in which scale $j = J - 1, J - 2, \ldots, J - P$. As in the one-dimensional case, image $f(x, y)$ is used as the $W_\varphi(J, m, n)$ input. Convolving its rows with $h_\varphi(-n)$ and $h_\psi(-n)$ and downsampling its columns, we get two subimages whose horizontal resolutions are reduced by a factor of 2. The highpass or detail component characterizes the image's high-frequency information with vertical orientation; the lowpass, approximation component contains its low-frequency, vertical information. Both subimages are then filtered columnwise and downsampled to yield four quarter-size output subimages—W_φ, W^H_ψ, W^V_ψ, and W^D_ψ.



a
b
c

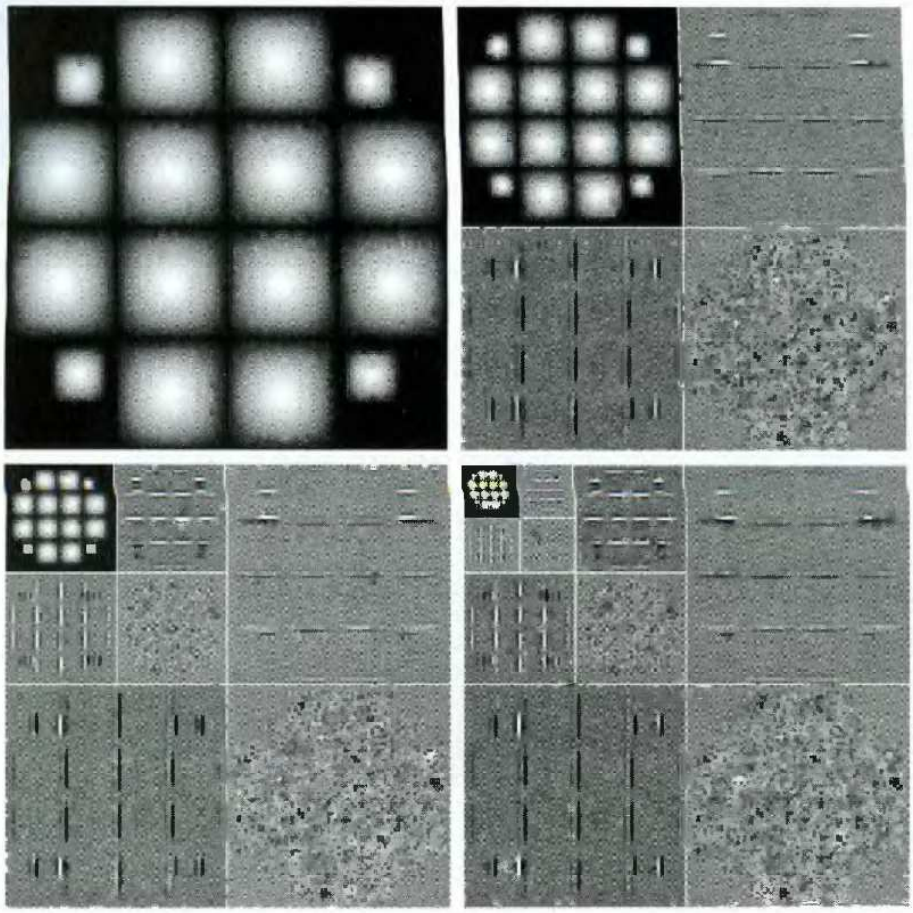
FIGURE 7.22 The two-dimensional fast wavelet transform: (a) the analysis filter bank; (b) the resulting decomposition; and (c) the synthesis filter bank.

These subimages, which are shown in the middle of Fig. 7.22(b), are the inner products of $f(x, y)$ and the two-dimensional scaling and wavelet functions in Eqs. (7.5-1) through (7.5-4), followed by downsampling by two in each dimension. Two iterations of the filtering process produces the two-scale decomposition at the far right of Fig. 7.22(b).

Figure 7.22(c) shows the synthesis filter bank that reverses the process described above. As would be expected, the reconstruction algorithm is similar to the one-dimensional case. At each iteration, four scale j approximation and detail subimages are upsampled and convolved with two one-dimensional filters—one operating on the subimages' columns and the other on its rows. Addition of the results yields the scale $j + 1$ approximation, and the process is repeated until the original image is reconstructed.

EXAMPLE 7.12: Computing a 2-D fast wavelet transform.

■ Consider the two-dimensional FWT shown in Fig. 7.23. Here, we see a sequence of filter-based decompositions of the 128×128 computer-generated image in Fig. 7.23(a). To generate the results, the two-dimensional filter bank of Fig. 7.22(a) and the *symlet* reconstruction filters of Figs. 7.24(a) and (b) were used. Figures 7.23(b), (c), and (d) are the resulting decompositions. In the first decomposition [Fig. 7.23(b)], the original image—a series of sinusoidal pulses on a black background—is the filter bank input; in all subsequent decompositions, approximation image W_φ—a subimage from the upper-left-hand corner of



a b
c d

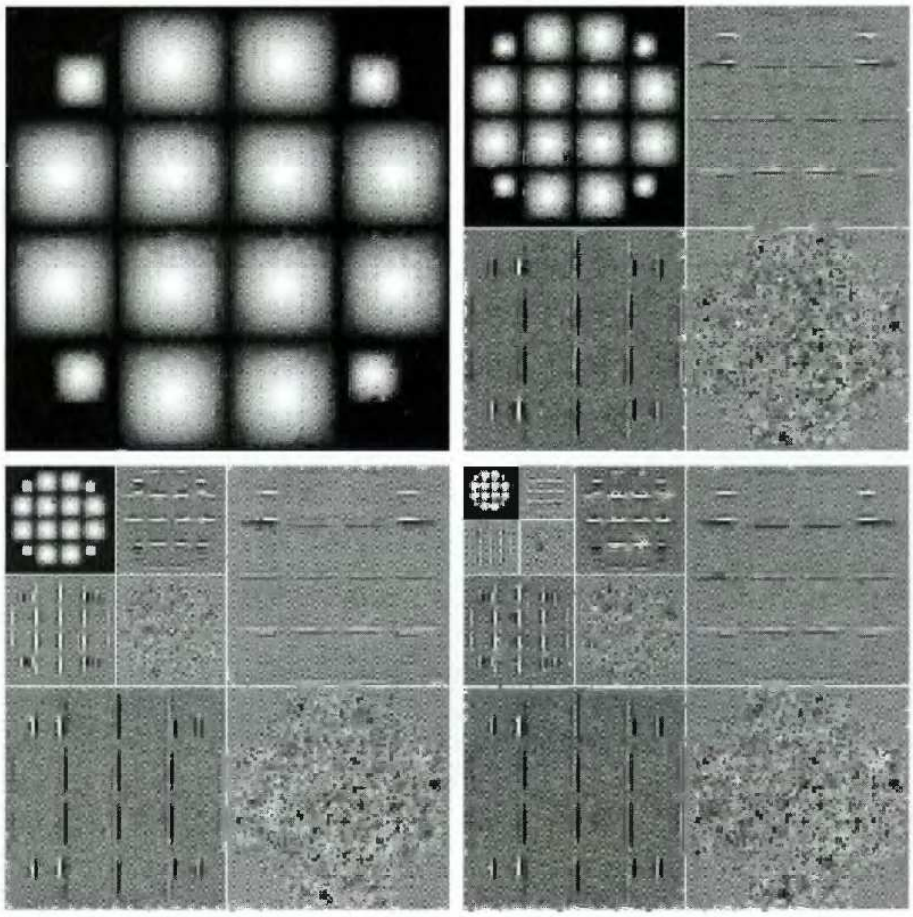
FIGURE 7.23 A three-scale FWT.

These subimages, which are shown in the middle of Fig. 7.22(b), are the inner products of $f(x, y)$ and the two-dimensional scaling and wavelet functions in Eqs. (7.5-1) through (7.5-4), followed by downsampling by two in each dimension. Two iterations of the filtering process produces the two-scale decomposition at the far right of Fig. 7.22(b).

Figure 7.22(c) shows the synthesis filter bank that reverses the process described above. As would be expected, the reconstruction algorithm is similar to the one-dimensional case. At each iteration, four scale j approximation and detail subimages are upsampled and convolved with two one-dimensional filters—one operating on the subimages' columns and the other on its rows. Addition of the results yields the scale $j + 1$ approximation, and the process is repeated until the original image is reconstructed.

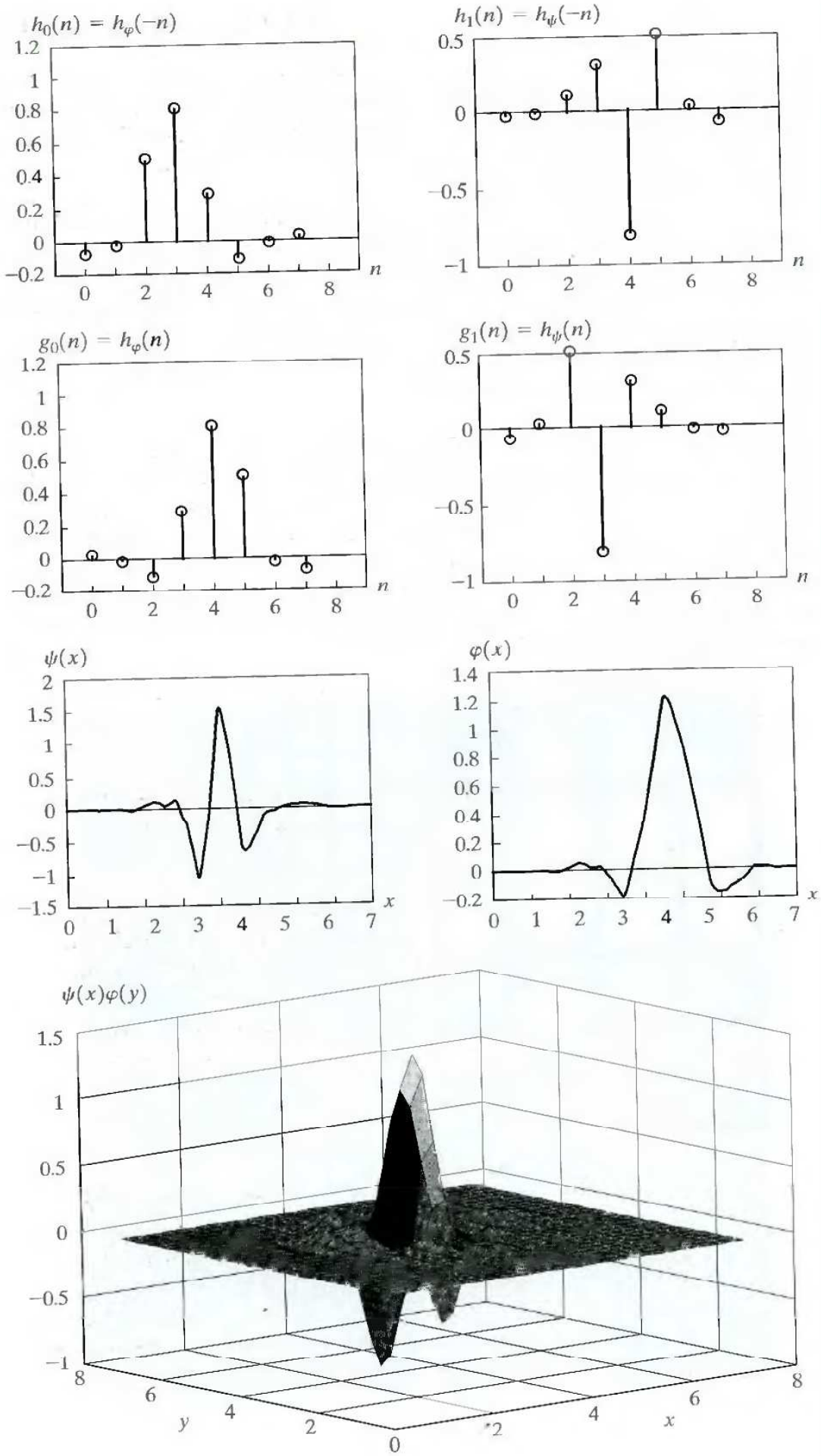
EXAMPLE 7.12: Computing a 2-D fast wavelet transform.

■ Consider the two-dimensional FWT shown in Fig. 7.23. Here, we see a sequence of filter-based decompositions of the 128×128 computer-generated image in Fig. 7.23(a). To generate the results, the two-dimensional filter bank of Fig. 7.22(a) and the *symlet* reconstruction filters of Figs. 7.24(a) and (b) were used. Figures 7.23(b), (c), and (d) are the resulting decompositions. In the first decomposition [Fig. 7.23(b)], the original image—a series of sinusoidal pulses on a black background—is the filter bank input; in all subsequent decompositions, approximation image W_φ—a subimage from the upper-left-hand corner of



a b
c d

FIGURE 7.23 A three-scale FWT.



a b
c d
e f
g

FIGURE 7.24 Fourth-order symlets: (a)–(b) decomposition filters; (c)–(d) reconstruction filters; (e) the one-dimensional wavelet; (f) the one-dimensional scaling function; and (g) one of three two-dimensional wavelets, $\psi^H(x, y)$.

the previous decomposition—is the input. Each decomposition produces four quarter-size output images that are arranged as shown in Fig. 7.22(b) and substituted for the input from which they were derived. Note the directional nature of the wavelet-based subimages, W_ψ^H, W_ψ^V, and W_ψ^D, at each scale. ■

The decomposition filters used in the preceding example are part of a well-known family of wavelets called *symlets*, short for "symmetrical wavelets." Although they are not perfectly symmetrical, they are designed to have the least asymmetry and highest number of vanishing moments† for a given compact support (Daubechies [1992]). Figures 7.24(e) and (f) show the fourth-order one-dimensional symlets (i.e., wavelet and scaling functions). Figures 7.24(a)–(d) show the corresponding decomposition and reconstruction filters. The coefficients of lowpass reconstruction filter $g_0(n) = h_\varphi(n)$ are 0.0322, −0.0126, −0.0992, 0.2979, 0.8037, 0.4976, −0.0296, and −0.0758 for $0 \leq n \leq 7$. The coefficients of the remaining orthonormal filters can be obtained using Eq. (7.1-23). Figure 7.24(g), a low-resolution graphic depiction of wavelet $\psi^H(x, y)$, is provided as an illustration of how a one-dimensional scaling and wavelet function combine to form a separable, two-dimensional wavelet.

We conclude the section with two examples that demonstrate the usefulness of wavelets in image processing. As in the Fourier domain, the basic approach is to

1. Compute the two-dimensional wavelet transform of an image.
2. Alter the transform.
3. Compute the inverse transform.

Because the DWT's scaling and wavelet vectors are used as lowpass and highpass filters, most Fourier-based filtering techniques have an equivalent "wavelet domain" counterpart.

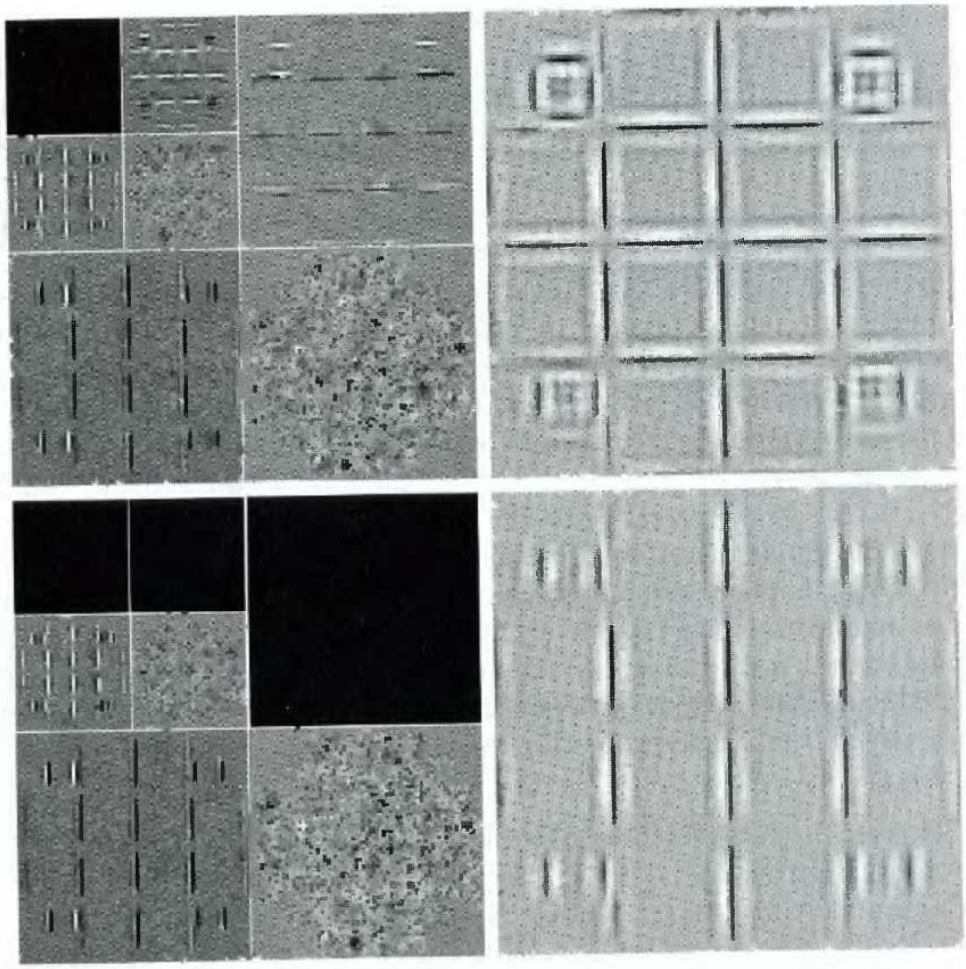
EXAMPLE 7.13: Wavelet-based edge detection.

■ Figure 7.25 provides a simple illustration. In Fig. 7.25(a), the lowest scale approximation component of the discrete wavelet transform shown in Fig. 7.23(c) has been eliminated by setting its values to zero. When the inverse transform is computed using these modified transform coefficients, as seen in Fig. 7.25(b), the effect is to emphasize or highlight the reconstructed image's edges. This provides considerable insight into the location of the original image's edges, despite the fact that they are relatively soft, sinusoidal transitions. By zeroing the horizontal details as well—see Figs. 7.25(c) and (d)—we can isolate the vertical edges. ■

EXAMPLE 7.14: Wavelet-based noise removal.

■ As a second example, consider the magnetic resonance image (MRI) of a human head shown in Fig. 7.26(a). As can be seen in the background, the image has been uniformly corrupted with a form of additive or multiplicative white

†The kth moment of wavelet $\psi(x)$ is $m(k) = \int x^k \psi(x)\, dx$. Zero moments impact the smoothness of the scaling and wavelet functions and our ability to represent them as polynomials. An order-N symlet has N vanishing moments.



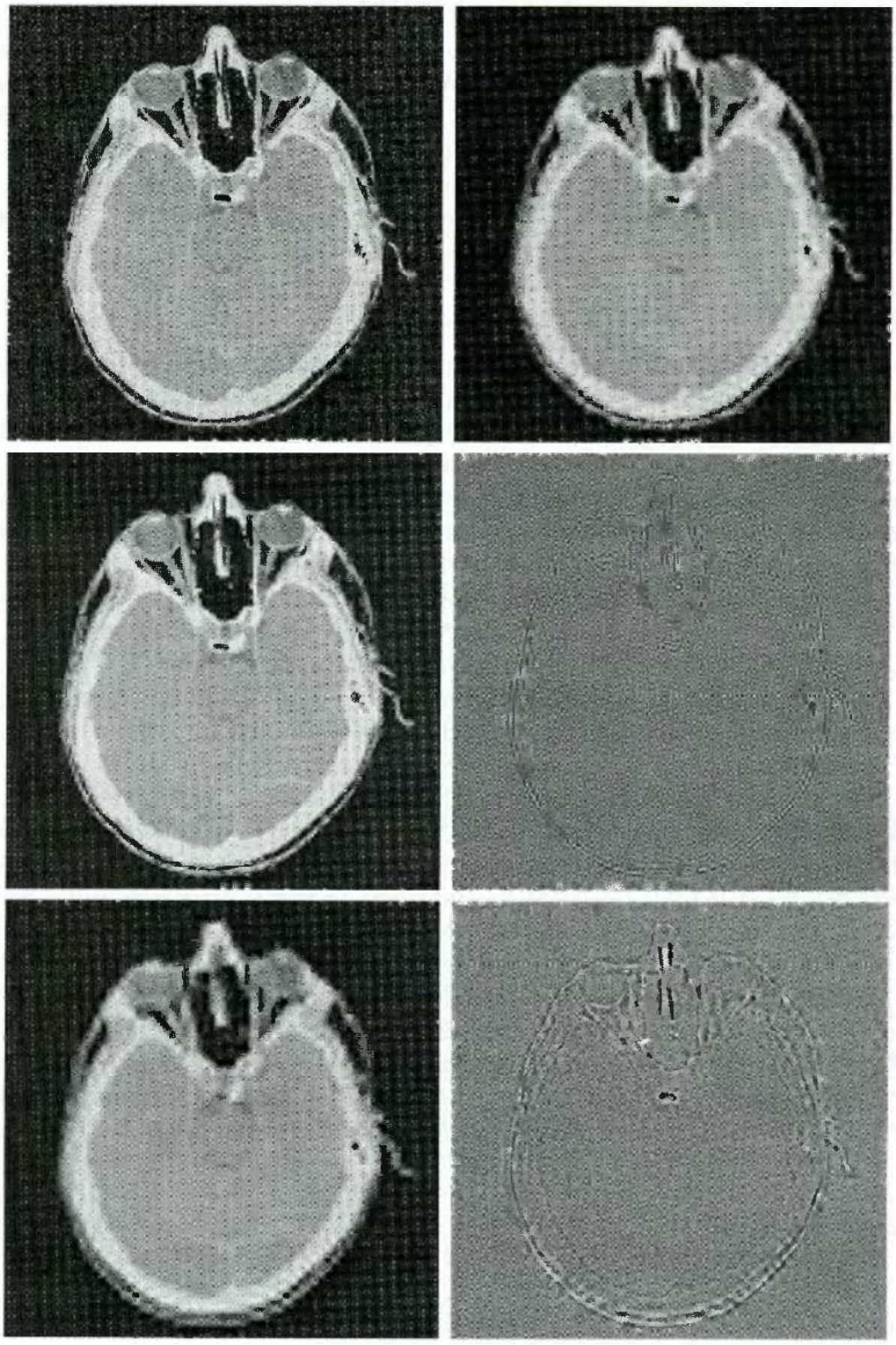
a b
c d

FIGURE 7.25 Modifying a DWT for edge detection: (a) and (c) two-scale decompositions with selected coefficients deleted; (b) and (d) the corresponding reconstructions.

noise. The general wavelet-based procedure for *denoising* the image (i.e., suppressing the noise part) is as follows:

1. Choose a wavelet (e.g., Haar, symlet, ...) and number of levels or scales, P, for the decomposition. Then compute the FWT of the noisy image.
2. Threshold the detail coefficients. That is, select and apply a threshold to the detail coefficients from scales $J - 1$ to $J - P$. This can be accomplished by *hard thresholding*, which means setting to zero the elements whose absolute values are lower than the threshold, or by *soft thresholding*, which involves first setting to zero the elements whose absolute values are lower than the threshold and then scaling the nonzero coefficients toward zero. Soft thresholding eliminates the discontinuity (at the threshold) that is inherent in hard thresholding.
3. Perform a wavelet reconstruction based on the original approximation coefficients at level $J - P$ and the modified detail coefficients for levels $J - 1$ to $J - P$.

Figure 7.26(b) shows the result of performing these operations with fourth-order symlets, two scales (i.e., $P = 2$), and a global threshold of 94.9093. Note the reduction in noise and corresponding loss of quality at the image edges. This loss of edge detail is greatly reduced in Fig. 7.26(c), which was generated by zeroing the highest-resolution detail coefficients (not the lower-resolution de-



a b
c d
e f

FIGURE 7.26 Modifying a DWT for noise removal: (a) a noisy MRI of a human head; (b), (c) and (e) various reconstructions after thresholding the detail coefficients; (d) and (f) the information removed during the reconstruction of (c) and (e). (Original image courtesy Vanderbuilt University Medical Center.)

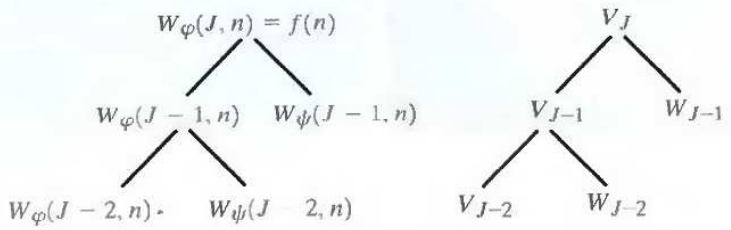
tails) and reconstructing the image. Here, almost all of the background noise has been eliminated and the edges are only slightly disturbed. Figure 7.26(d) shows the information that is lost in the process. This result was generated by computing the inverse FWT of the decomposed image with all but the highest-resolution detail coefficients zeroed. As can be seen, it contains most of the

noise in the original image and some of the edge information. Figures 7.26(e) and (f) are a similar set of images involving all detail coefficients. That is, Fig. 7.26(e) is a reconstruction of the DWT in which the details at both levels of the decomposition have been zeroed; Fig. 7.26(f) is a reconstruction in which all but the detail coefficients (i.e., only the lowest level approximation coefficients) have been eliminated. Note the significant increase in edge information in Fig. 7.26(f) and the corresponding decrease in edge detail in Fig. 7.26(e). ■

7.6 Wavelet Packets

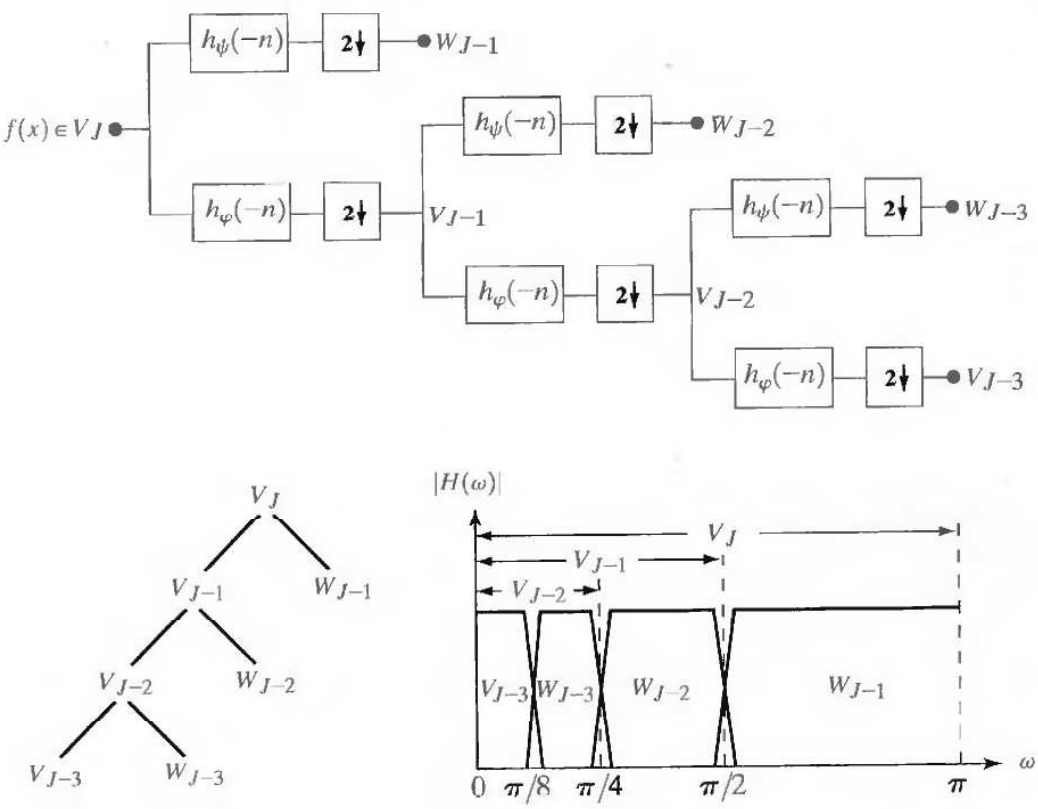
The fast wavelet transform decomposes a function into a series of logarithmically related frequency bands. That is, the low frequencies are grouped into narrow bands, while the high frequencies are grouped into wider bands. If you look along the frequency axis of the time-frequency plane in Fig. 7.21(c), this is immediately apparent. It is a defining characteristic of what are commonly called *constant-Q filters*. If we want greater control over the partitioning of the time-frequency plane (e.g., smaller bands at the higher frequencies), the FWT must be generalized to yield a more flexible decomposition—called a *wavelet packet* (Coifman and Wickerhauser [1992]). The cost of this generalization is an increase in computational complexity from $O(M)$ for the FWT to $O(M \log M)$.

Consider again the two-scale filter bank of Fig. 7.16(a)—but picture the decomposition as a *binary tree*. Figure 7.27(a) details the structure of the tree and links the appropriate FWT scaling and wavelet coefficients [from Fig. 7.16(a)] to its *nodes*. The *root node* is assigned the highest-scale approximation coefficients, which are samples of the function itself, while the *leaves* inherit the transform's approximation and detail coefficient outputs. The lone intermediate node, $W_\varphi(J-1, n)$, is a filter bank approximation that is ultimately filtered to become two leaf nodes. Note that the coefficients of each node are the weights of a linear expansion that produces a band-limited "piece" of root node $f(n)$. Since any such piece is an element of a known scaling or wavelet subspace (see Sections 7.2.2 and 7.2.3), we can replace the generating coefficients in Fig. 7.27(a) by the corresponding subspace. The result is the *subspace analysis tree* of Fig. 7.27(b). Although the variable W is used to denote both coefficients and subspaces, the two quantities are distinguishable by the format of their subscripts.



a b

FIGURE 7.27 A coefficient (a) and analysis (b) tree for the two-scale FWT analysis bank of Fig. 7.16.



a
b c

FIGURE 7.28 A three-scale FWT filter bank: (a) block diagram; (b) decomposition space tree; and (c) spectrum splitting characteristics.

These concepts are further illustrated in Fig. 7.28, where a three-scale FWT analysis bank, analysis tree, and corresponding frequency spectrum are depicted. Unlike Fig. 7.16(a), the block diagram of Fig. 7.28(a) is labeled to resemble the analysis tree in Fig. 7.28(b)—as well as the spectrum in Fig. 7.28(c). Thus, while the output of the upper-left filter and subsampler is, to be accurate, $W_\psi(J-1, n)$, it has been labeled W_{J-1}—the subspace of the function that is generated by the $W_\psi(J-1, n)$ transform coefficients. This subspace corresponds to the upper-right leaf of the associated analysis tree, as well as the rightmost or widest bandwidth segment of the corresponding frequency spectrum.

Analysis trees provide a compact and informative way of representing multiscale wavelet transforms. They are simple to draw, take less space than their corresponding filter and subsampler-based block diagrams, and make it relatively easy to spot valid decompositions. The three-scale analysis tree of Fig. 7.28(b), for example, offers the following three expansion options:

$$V_J = V_{J-1} \oplus W_{J-1} \tag{7.6-1}$$

$$V_J = V_{J-2} \oplus W_{J-2} \oplus W_{J-1} \tag{7.6-2}$$

$$V_J = V_{J-3} \oplus W_{J-3} \oplus W_{J-2} \oplus W_{J-1}. \tag{7.6-3}$$

They correspond to the one-, two-, and three-scale FWT decompositions of Section 7.4 and may be obtained from Eq. (7.2-27) of Section 7.2.3 by letting $j_0 = J - P$ for $P = \{1, 2, 3\}$. In general, a P-scale FWT analysis tree supports P unique decompositions.

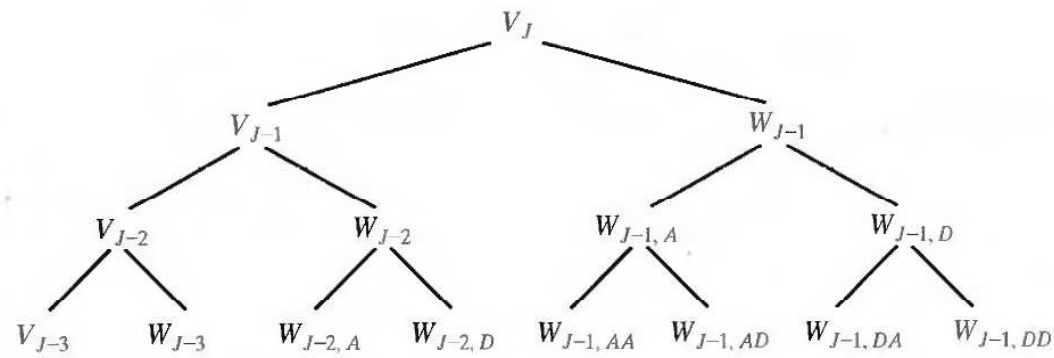


FIGURE 7.29 A three-scale wavelet packet analysis tree.

Analysis trees are also an efficient mechanism for representing *wavelet packets*, which are nothing more than *conventional wavelet transforms in which the details are iteratively filtered*. Thus, the three-scale FWT analysis tree of Fig. 7.28(b) becomes the three-scale *wavelet packet* tree of Fig. 7.29. Note the additional subscripting that is introduced. The first subscript of a double-subscripted node identifies the scale of the FWT *parent* node from which it descended. The second—a variable length string of A's and D's—encodes the path from the parent to the node. An A designates approximation filtering, while a D indicates detail filtering. Subspace $W_{J-1,DA}$, for example, is obtained by "filtering" the scale $J-1$ FWT coefficients (i.e., parent W_{J-1} in Fig. 7.29) through an additional detail filter (yielding $W_{J-1,D}$), followed by an approximation filter (giving $W_{J-1,DA}$). Figures 7.30(a) and (b) are the filter bank and spectrum splitting characteristics of the analysis tree in Fig. 7.29. Note the evenly spaced frequency bands that are characteristic of full packet decompositions.

The three-scale packet tree in Fig. 7.29 almost triples the number of decompositions (and associated time-frequency tilings) that are available from the three-scale FWT tree. Recall that in a normal FWT, we split, filter, and downsample the lowpass bands alone. This creates a fixed logarithmic relationship between frequency bands. Thus, while the three-scale FWT analysis tree of Fig. 7.28(a) offers three possible decompositions—see Eqs. (7.6-1) to (7.6-3)—the wavelet packet tree of Fig. 7.29 supports 26 different decompositions. For instance, V_J [and therefore function $f(n)$] can be expanded as

$$V_J = V_{J-3} \oplus W_{J-3} \oplus W_{J-2,A} \oplus W_{J-2,D} \oplus W_{J-1,AA} \oplus W_{J-1,AD} \oplus W_{J-1,DA} \oplus W_{J-1,DD} \quad (7.6\text{-}4)$$

whose spectrum is shown in Fig. 7.30(b), or

$$V_J = V_{J-1} \oplus W_{J-1,D} \oplus W_{J-1,AA} \oplus W_{J-1,AD} \quad (7.6\text{-}5)$$

whose spectrum is depicted in Fig. 7.31. Note the difference between this last spectrum and the full packet spectrum of Fig. 7.30(b), or the three-scale FWT spectrum of Fig. 7.28(c). In general, P-scale, one-dimensional wavelet packet transforms (and associated $P+1$-level analysis trees) support

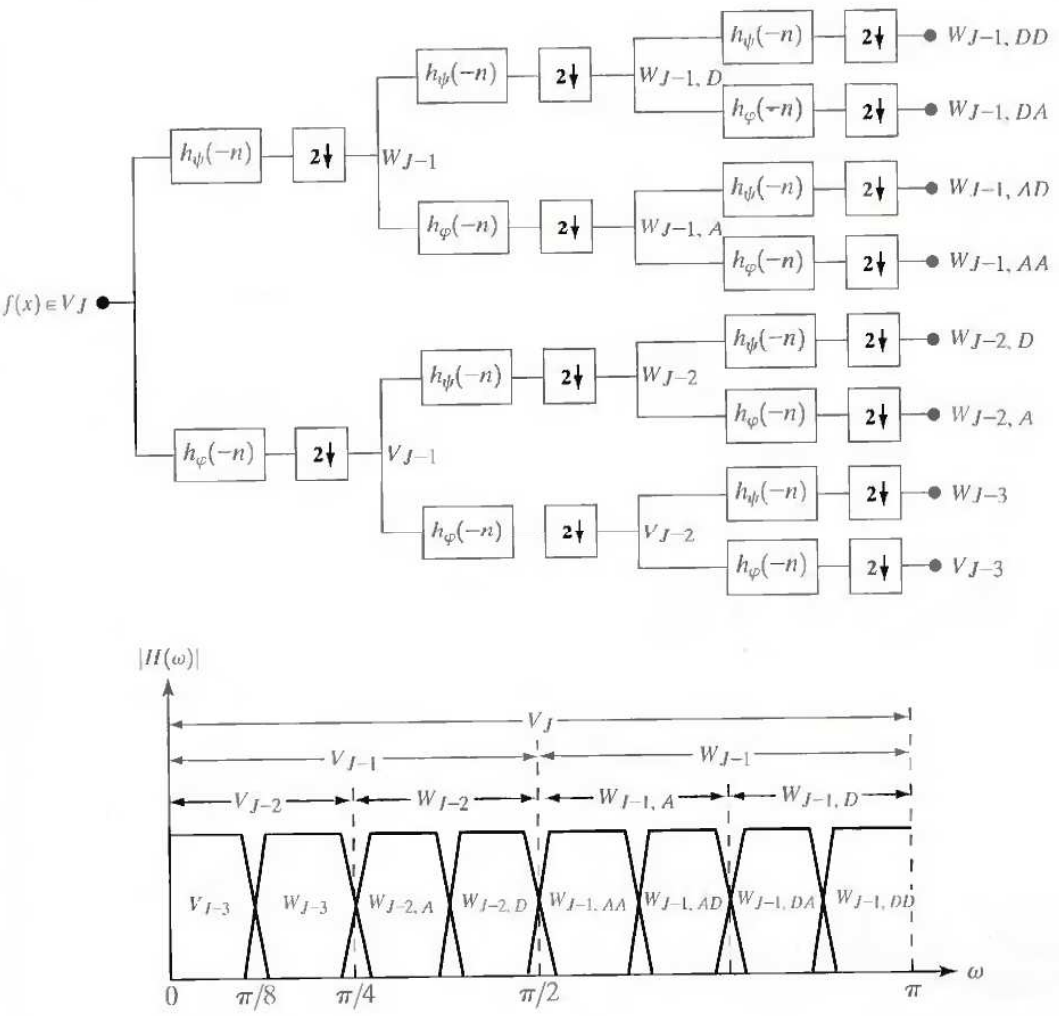


FIGURE 7.30 The (a) filter bank and (b) spectrum splitting characteristics of a three-scale full wavelet packet analysis tree.

$$D(P+1) = \left[D(P)\right]^2 + 1 \tag{7.6-6}$$

unique decompositions, where $D(1) = 1$. With such a large number of valid expansions, packet-based transforms provide improved control of the partitioning of the decomposed function's spectrum. Of course, the cost of this control is an increase in computational complexity [compare the filter bank in Fig. 7.28(a) to that of Fig. 7.30(a)].

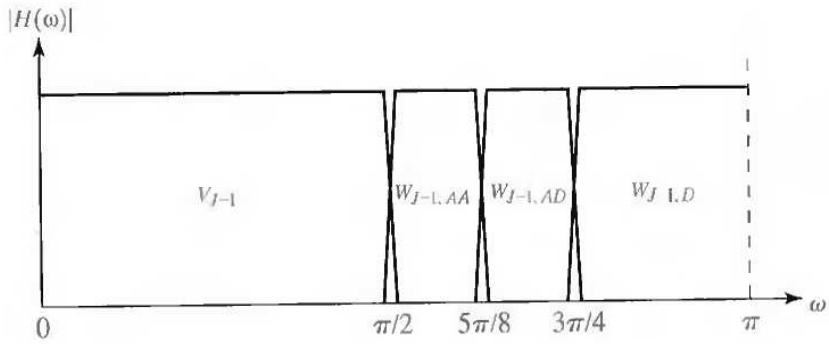
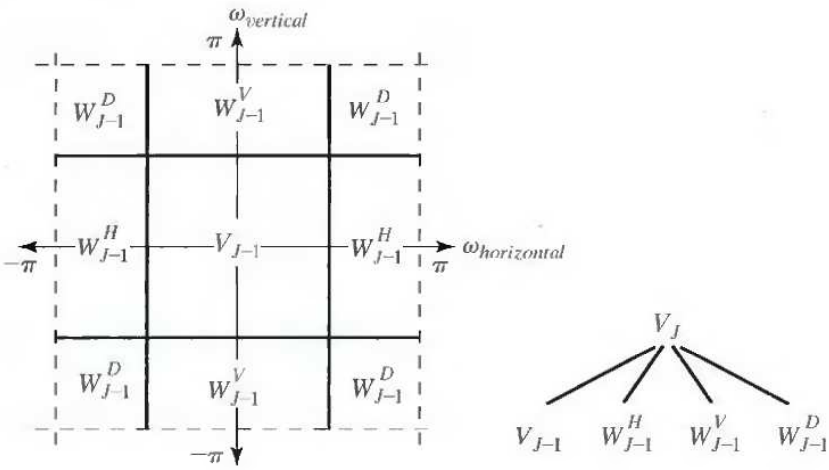


FIGURE 7.31 The spectrum of the decomposition in Eq. (7.6-5).

a b

FIGURE 7.32 The first decomposition of a two-dimensional FWT: (a) the spectrum and (b) the subspace analysis tree.



Now consider the two-dimensional, four-band filter bank of Fig. 7.22(a). As was noted in Section 7.5, it splits approximation $W_\varphi(j+1, m, n)$ into outputs, $W_\varphi(j, m, n)$, $W^H_\psi(j, m, n)$, $W^V_\psi(j, m, n)$, and $W^D_\psi(j, m, n)$. As in the one-dimensional case, it can be "iterated" to generate P scale transforms for scales $j = J-1, J-2, \ldots, J-P$, with $W_\varphi(J, m, n) = f(m, n)$. The spectrum resulting from the first iteration [i.e., using $j+1 = J$ in Fig. 7.22(a)] is shown in Fig. 7.32(a). Note that it divides the frequency plane into four equal areas. The low-frequency quarter band in the center of the plane coincides with transform coefficients $W_\varphi(J-1, m, n)$ and scaling space V_{J-1}. (This nomenclature is consistent with the one-dimensional case.) To accommodate the two-dimensional nature of the input, however, we now have three (rather than one) wavelet subspaces. They are denoted W^H_{J-1}, W^V_{J-1}, and W^D_{J-1} and correspond to coefficients $W^H_\psi(J-1, m, n)$, $W^V_\psi(J-1, m, n)$, and $W^D_\psi(J-1, m, n)$, respectively. Figure 7.32(b) shows the resulting four-band, single-scale *quaternary FWT analysis tree*. Note the superscripts that link the wavelet subspace designations to their transform coefficient counterparts.

Figure 7.33 shows a portion of a three-scale, two-dimensional wavelet packet analysis tree. Like its one-dimensional counterpart in Fig. 7.29, the first subscript of every node that is a descendent of a conventional FWT detail node is

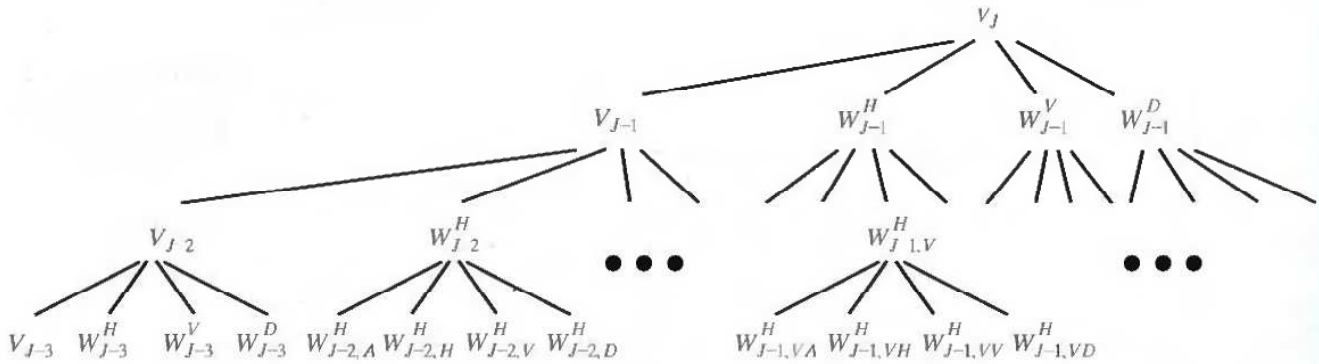


FIGURE 7.33 A three-scale, full wavelet packet decomposition tree. Only a portion of the tree is provided.

the scale of that *parent* detail node. The second subscript—a variable length string of A's and D's—encodes the path from the parent to the node under consideration. The node labeled $W^H_{J-1,VD}$, for example, is obtained by "row/column filtering" the scale $J-1$ FWT horizontal detail coefficients (i.e., parent W^H_{J-1} in Fig. 7.33) through an additional detail/approximation filter (yielding $W^H_{J-1,V}$), followed by a detail/detail filter (giving $W^H_{J-1,VD}$). A P-scale, two-dimensional wavelet packet tree supports

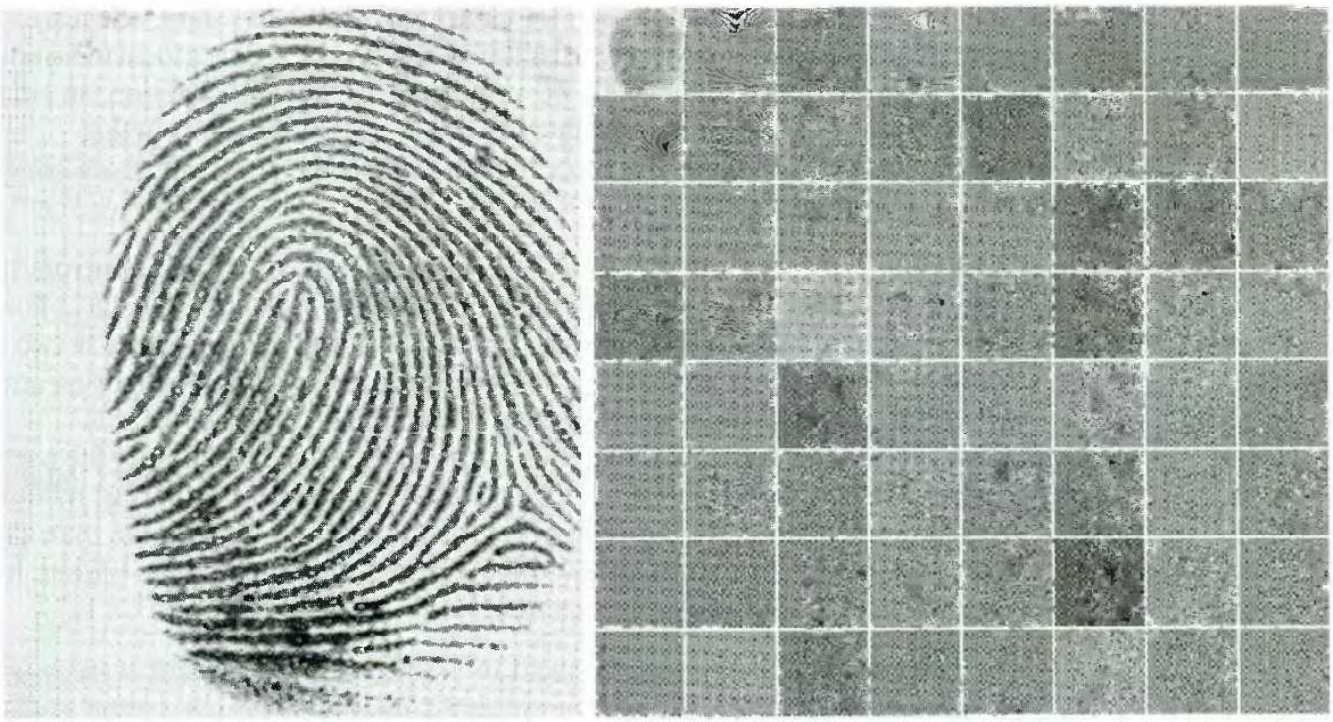
$$D(P+1) = [D(P)]^4 + 1 \tag{7.6-7}$$

unique expansions, where $D(1) = 1$. Thus, the three-scale tree of Fig. 7.33 offers 83,522 possible decompositions!

EXAMPLE 7.15: Two-dimensional wavelet packet decompositions.

■ A single wavelet packet tree presents numerous decomposition options. In fact, the number of possible decompositions is often so large that it is impractical, if not impossible, to enumerate or examine them individually. An efficient algorithm for finding optimal decompositions with respect to application specific criteria is highly desirable. As will be seen, classical entropy-based criteria are applicable in many situations and well suited to binary and quartenary tree searching algorithms.

Consider the problem of compressing the fingerprint image in Fig. 7.34(a). Using three-scale wavelet packet trees, there are 83,522 [see Eq. (7.6-7)]



a b

FIGURE 7.34 (a) A scanned fingerprint and (b) its three-scale, full wavelet packet decomposition. (Original image courtesy of the National Institute of Standards and Technology.)

potential decompositions that could serve as the starting point for the compression process. Figure 7.34(b) shows one of them—a full wavelet packet, 64-leaf decomposition like the analysis tree of Fig. 7.33. Note that the leaves of the tree correspond to the subbands of the 8×8 array of decomposed subimages in Fig. 7.34(b). The probability that this particular 64-leaf decomposition is in some way optimal for the purpose of compression, however, is relatively low. In the absence of a suitable optimality criterion, we can neither confirm nor deny it.

One reasonable criterion for selecting a decomposition for the compression of the image of Fig. 7.34(a) is the additive cost function

$$E(f) = \sum_{m,n} |f(m, n)|. \tag{7.6-8}$$

This function measures the entropy or information content of two-dimensional function f. Entropies near 0 indicate functions with little to no information. For example, the entropy of function $f(m, n) = 0$ for all m and n is 0. It provides no information. High entropy values, on the other hand, are indicative of functions with many nonzero values. Since most transform-based compression schemes work by truncating or thresholding the small coefficients to zero, a cost function that maximizes the number of near-zero values is a reasonable criterion for selecting a "best," from a compression viewpoint, decomposition.

The cost function just described is both computationally simple and easily adapted to tree optimization routines. The optimization algorithm must use the function to minimize the "cost" of the decomposition tree's leaf nodes. Minimal entropy leaf nodes should be favored because they have more near-zero values, which leads to greater compression. Because the cost function of Eq. (7.6-8) is a local measure that uses only the information available at the node under consideration, an efficient algorithm for finding minimal entropy solutions is easily constructed:

For each node of the analysis tree, beginning with the root and proceeding level by level to the leaves:

1. Compute both the entropy of the node, denoted E_P (for parent entropy), and the entropy of its four offspring—denoted E_A, E_H, E_V, and E_D. For two-dimensional wavelet packet decompositions, the parent is a two-dimensional array of approximation or detail coefficients; the offspring are the filtered approximation, horizontal, vertical, and diagonal details.
2. If the combined entropy of the offspring is less than the entropy of the parent—that is, $E_A + E_H + E_V + E_D < E_P$—include the offspring in the analysis tree. If the combined entropy of the offspring is greater than or equal to that of the parent, prune the offspring, keeping only the parent. It is a leaf of the optimized analysis tree.

The preceding algorithm can be used to (1) prune wavelet packet trees or (2) design procedures for computing optimal trees from scratch. In the latter case, nonessential siblings—descendants of nodes that would be eliminated in Step 2 of the algorithm—would not be computed. Figure 7.35 shows the optimized decomposition that results from applying the algorithm to the image of Fig. 7.34(a) with the cost function of Eq. (7.6-8). The corresponding analysis tree is given in Fig. 7.36. Note that many of the original full packet decomposi-

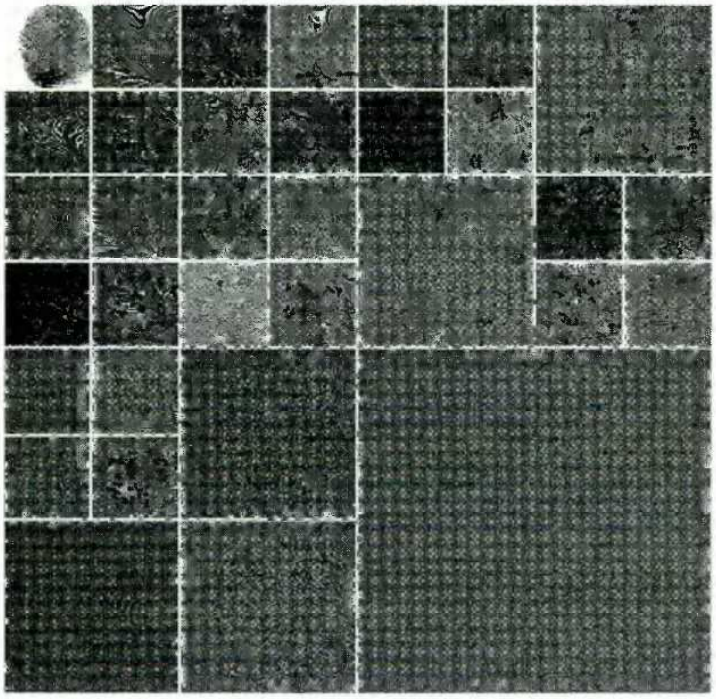


FIGURE 7.35 An optimal wavelet packet decomposition for the fingerprint of Fig. 7.34(a).

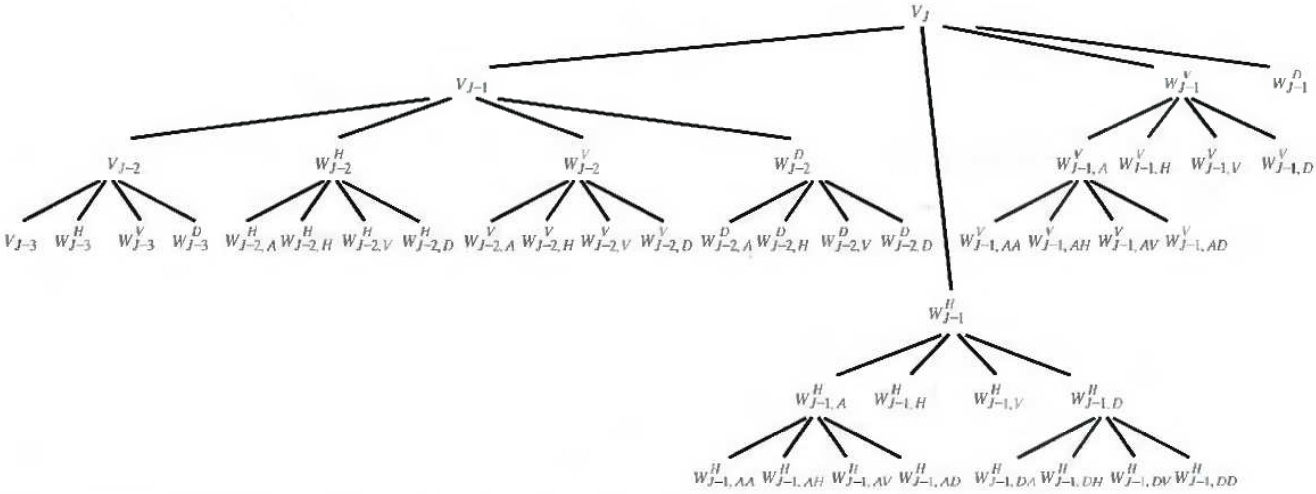


FIGURE 7.36 The optimal wavelet packet analysis tree for the decomposition in Fig. 7.35.

tion's 64 subbands in Fig. 7.34(b) (and corresponding analysis tree's 64 leaves in Fig. 7.33) have been eliminated. In addition, the subimages that are not split (or further decomposed) in Fig. 7.35 are relatively smooth and composed of pixels that are middle gray in value. Since all but the approximation subimage of this figure have been scaled so that gray level 128 indicates a zero-valued coefficient, these subimages contain little information. There would be no overall decrease in entropy realized by splitting them. ■

The preceding example is based on a real-world problem that was solved through the use of wavelets. The Federal Bureau of Investigation (FBI) currently maintains a large database of fingerprints and has established a wavelet-based national standard for the digitization and compression of fingerprint images (FBI [1993]). Using biorthogonal wavelets, the standard achieves a typical compression ratio of 15:1. The advantages of wavelet-based compression over the more traditional JPEG approach are examined in the next chapter.

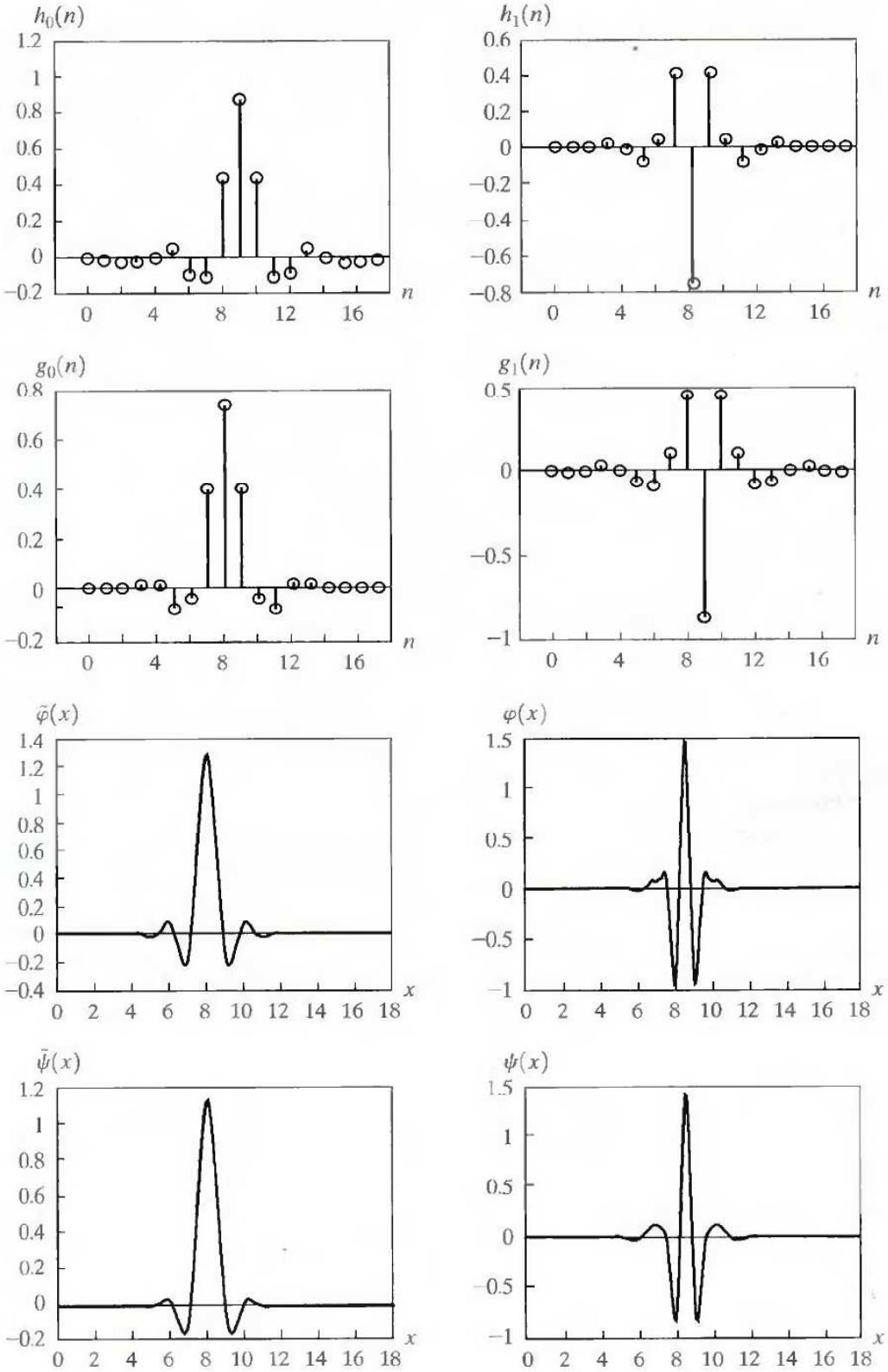
The decomposition filters used in Example 7.15, as well as by the FBI, are part of a well-known family of wavelets called Cohen-Daubechies-Feauveau biorthogonal wavelets (Cohen, Daubechies, and Feauveau [1992]). Because the family's scaling and wavelet functions are symmetrical and have similar lengths, they are among the most widely used biorthogonal wavelets. Figures 7.37(e)–(h) show the dual scaling and wavelet functions that are characteristic of biorthogonal bases. Figures 7.37(a)–(d) are the corresponding decomposition and reconstruction filters. The coefficients of lowpass decomposition filter $h_0(n)$ are 0, 0.0019, −0.0019, −0.017, 0.0119, 0.0497, −0.0773, −0.0941, 0.4208, 0.8259, 0.4208, −0.0941, −0.0773, 0.0497, 0.0119, −0.017, −0.0019, and 0.0010 for $0 \leq n \leq 17$. Highpass decomposition filter $h_1(n)$ has coefficients 0, 0, 0, 0.0144, −0.0145, −0.0787, 0.0404, 0.4178, −0.7589, 0.4178, 0.0404, −0.0787, −0.0145, 0.0144, 0, 0, 0, and 0 for $0 \leq n \leq 17$. The corresponding coefficients of the biorthogonal synthesis filters can be computed using $g_0(n) = (-1)^{n+1}h_1(n)$ and $g_1(n) = (-1)^n h_0(n)$ of Eq. (7.1-15). That is, they are cross-modulated versions of the decomposition filters. Note that zero padding is employed to make the filters the same length.

Summary

The material of this chapter establishes a solid mathematical foundation for understanding and accessing the role of wavelets and multiresolution analysis in image processing. Wavelets and wavelet transforms are relatively new imaging tools that are being rapidly applied to a wide variety of image processing problems. Because of their similarity to the Fourier transform, many of the techniques in Chapter 4 have wavelet domain counterparts. A partial listing of the imaging applications that have been approached from a wavelet point of view includes image matching, registration, segmentation, denoising, restoration, enhancement, compression, morphological filtering, and computed tomography. Since it is impractical to cover all of these applications in a single chapter, the topics included were chosen for their value in introducing or clarifying fundamental concepts and preparing the reader for further study in the field. In Chapter 8, we will apply wavelets to the compression of images.

References and Further Reading

There are many good texts on wavelets and their application. Several complement our treatment and were relied on during the development of the core sections of the chapter. The material in Section 7.1.2 on subband coding and digital filtering follows the book by Vetterli and Kovacevic [1995], while Sections 7.2 and 7.4 on multiresolution expansions and the fast wavelet transform follow the treatment of these subjects in Burrus, Gopinath, and Guo [1998]. The remainder of the material in the chapter is based on the



a b
c d
e f
g h

FIGURE 7.37 A member of the Cohen-Daubechies-Feauveau biorthogonal wavelet family: (a) and (b) decomposition filter coefficients; (c) and (d) reconstruction filter coefficients; (e)–(h) dual wavelet and scaling functions.

references cited in the text. Experimental work and many of the examples were done using the MATLAB wavelet toolbox (Misiti, Misiti, Oppenheim, and Poggi [1996]).

The history of wavelet analysis is recorded in a book by Hubbard [1998]. The early predecessors of wavelets were developed simultaneously in different fields and unified in a paper by Mallat [1987]. It brought a mathematical framework to the field. Much of the history of wavelets can be traced through the works of Meyer [1987] [1990] [1992a, b] [1993], Mallat [1987] [1989a–c] [1998], and Daubechies [1988] [1990] [1992] [1993] [1996]. The current interest in wavelets was stimulated by many of their publications. The book by Daubechies [1992] is a classic source for the mathematical details of wavelet theory.

The application of wavelets to image processing is addressed in general image processing texts, like Castleman [1996], and many application-specific books, some of which are conference proceedings. In this latter category, for example, are Rosenfeld [1984], Prasad and Iyengar [1997], and Topiwala [1998]. Recent articles that can serve as starting points for further research into specific imaging applications include Thévenaz and Unser [2000] for image registration; Chang and Kuo [1993] and Unser [1995] on texture-based classification; Heijmans and Goutsias [2000] on morphological wavelets; Banham et al. [1994], Wang, Zhang, and Pan [1995], and Banham and Kastaggelos [1996] on image restoration; Xu et al. [1994] and Chang, Yu, and Vetterli [2000] on image enhancement; Delaney and Bresler [1995] and Westenberg and Roerdink [2000] on computed tomography; and Lee, Sun, and Chen [1995], Liang and Kuo [1999], Wang, Lee, and Toraichi [1999], and You and Bhattacharya [2000] on image description and matching. One of the most important applications of wavelets is image compression—see, for example, Antonini et al. [1992], Wei et al. [1998], and the book by Topiwala [1998]. Finally, there have been a number of special issues devoted to wavelets, including a special issue on wavelet transforms and multiresolution signal anaysis in the *IEEE Transactions on Information Theory* [1992], a special issue on wavelets and signal processing in the *IEEE Transactions on Signal Processing* [1993], and a special section on multiresolution representation in the *IEEE Transactions on Pattern Analysis and Machine Intelligence* [1989].

Although the chapter focuses on the fundamentals of wavelets and their application to image processing, there is considerable interest in the construction of wavelets themselves. The interested reader is referred to the work of Battle [1987] [1988], Daubechies [1988] [1992], Cohen and Daubechies [1992], Meyer [1990], Mallat [1989b], Unser, Aldroubi, and Eden [1993], and Gröchenig and Madych [1992]. This is not an exhaustive list but should serve as a starting point for further reading. See also the general references on subband coding and filter banks, including Strang and Nguyen [1996] and Vetterli and Kovacevic [1995], and the references included in the chapter with respect to the wavelets we used as examples.

Problems

See inside front cover

Detailed solutions to the problems marked with a star can be found in the book web site. The site also contains suggested projects based on the material in this chapter.

7.1 Design a system for decoding the prediction residual pyramid generated by the encoder of Fig. 7.2(b) and draw its block diagram. Assume there is no quantization error introduced by the encoder.

★7.2 Construct a fully populated approximation pyramid and corresponding prediction residual pyramid for the image

$$f(x, y) = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}.$$

Use 2×2 block neighborhood averaging for the approximation filter in Fig. 7.2(b) and assume the interpolation filter is omitted.

★**7.3** Given a $2^J \times 2^J$ image, does a $J + 1$-level pyramid reduce or expand the amount of data required to represent the image? What is the compression or expansion ratio?

7.4 Prove that the following filters from Table 7.1 form perfect reconstruction filter banks:

★ **(a)** Quadrature mirror filter (QMF)

(b) Orthonormal filter

7.5 Are quadrature mirror filters biorthogonal or othonormal or both?

7.6 Compute the coefficients of the Daubechies synthesis filters $g_0(n)$ and $g_1(n)$ for Example 7.2. Using Eq. (7.1-22) with $m = 0$ only, show that the filters are orthonormal.

★**7.7** Draw a two-dimensional four-band filter bank decoder to reconstruct input $x(m, n)$ in Fig. 7.5.

7.8 Obtain the Haar transformation matrix for $N = 8$.

7.9 **(a)** Compute the Haar transform of the 2×2 image

$$\mathbf{F} = \begin{bmatrix} 3 & -1 \\ 6 & 2 \end{bmatrix}.$$

(b) The inverse Haar transform is $\mathbf{F} = \mathbf{H}^{-1}\mathbf{T}\mathbf{H}^{-1}$, where $\mathbf{T}$ is the Haar transform and $\mathbf{H}^{-1}$ denotes the matrix inverse of Haar tranformation matrix $\mathbf{H}$. Find $\mathbf{H}_2^{-1}$ for Haar transformation matrix $\mathbf{H}_2$ and use it to compute the inverse Haar transform of the result in (a).

7.10 Compute the expansion coefficients of 2-tuple $[3, 2]^T$ for the following bases and write the corresponding expansions:

★ **(a)** Basis $\varphi_0 = [1/\sqrt{2}, 1/\sqrt{2}]^T$ and $\varphi_1 = [1/\sqrt{2}, -1/\sqrt{2}]^T$ on $\mathbf{R}^2$, the set of real 2-tuples.

(b) Basis $\varphi_0 = [1, 0]^T$ and $\varphi_1 = [1, 1]^T$, and its dual, $\tilde{\varphi}_0 = [1, -1]^T$ and $\tilde{\varphi}_1 = [0, 1]^T$, on $\mathbf{R}^2$.

(c) Basis $\varphi_0 = [1, 0]^T, \varphi_1 = [-1/2, \sqrt{3}/2]^T$, and $\varphi_1 = [-1/2, -\sqrt{3}/2]^T$, and their duals, $\tilde{\varphi}_i = 2\varphi_i/3$ for $i = \{0, 1, 2\}$, on $\mathbf{R}^2$.

Hint: Vector inner products must be used in place of the integral inner products of Section 7.2.1.

7.11 Show that scaling function

$$\varphi(x) = \begin{cases} 1 & 0.25 \le x < 0.75 \\ 0 & \text{elsewhere} \end{cases}$$

does not satisfy the second requirement of a multiresolution analysis.

7.12 Write an expression for scaling space V_3 as a function of scaling function $\varphi(x)$. Use the Haar scaling function definition of Eq. (7.2-14) to draw the Haar V_3 scaling functions at translations $k = \{0, 1, 2\}$.

★**7.13** Draw wavelet $\psi_{3,3}(x)$ for the Haar wavelet function. Write an expression for $\psi_{3,3}(x)$ in terms of the Haar scaling function.

7.14 Suppose function $f(x)$ is a member of Haar scaling space V_3—that is, $f(x) \in V_3$. Use Eq. (7.2-22) to express V_3 as a function of scaling space V_0 and any required wavelet spaces. If $f(x)$ is 0 outside the interval $[0, 1)$, sketch the scaling and wavelet functions required for a linear expansion of $f(x)$ based on your expression.

7.15 Compute the first four terms of the wavelet series expansion of the function used in Example 7.7 with starting scale $j_0 = 1$. Write the resulting expansion in terms

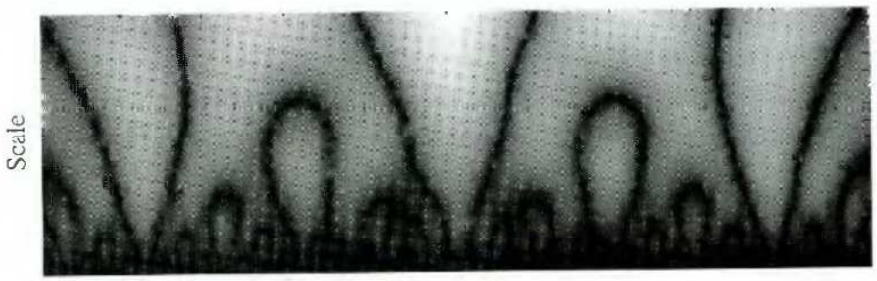
of the scaling and wavelet functions involved. How does your result compare to the example, where the starting scale was $j_0 = 0$?

7.16 The DWT in Eqs. (7.3-5) and (7.3-6) is a function of starting scale j_0.

(a) Recompute the one-dimensional DWT of function $f(n) = \{1, 4, -3, 0\}$ for $0 \le n \le 3$ in Example 7.8 with $j_0 = 1$ (rather than 0).

(b) Use the result from (a) to compute $f(1)$ from the transform values.

★7.17 What does the following continuous wavelet transform reveal about the one-dimensional function upon which it was based?



7.18 **(a)** The continuous wavelet transform of Problem 7.17 is computer generated. The function upon which it is based was first sampled at discrete intervals. What is continuous about the transform—or what distinguishes it from the discrete wavelet transform of the function?

★ (b) Under what circumstances is the DWT a better choice than the CWT? Are there times when the CWT is better than the DWT?

★7.19 Draw the FWT filter bank required to compute the transform in Problem 7.16. Label all inputs and outputs with the appropriate sequences.

7.20 The computational complexity of an M-point fast wavelet transform is $O(M)$. That is, the number of operations is proportional M. What determines the constant of proportionality?

7.21 ★ (a) If the input to the three-scale FWT filter bank of Fig. 7.28(a) is the Haar scaling function $\varphi(n) = 1$ for $n = 0, 1, \ldots, 7$ and 0 elsewhere, what is the resulting transform with respect to Haar wavelets?

(b) What is the transform if the input is the corresponding Haar wavelet function $\psi(n) = \{1, 1, 1, 1, -1, -1, -1, -1\}$ for $n = 0, 1, \ldots, 7$?

(c) What input sequence produces transform $\{0, 0, 0, 0, 0, 0, B, 0\}$ with nonzero coefficient $W_\psi(2, 2) = B$?

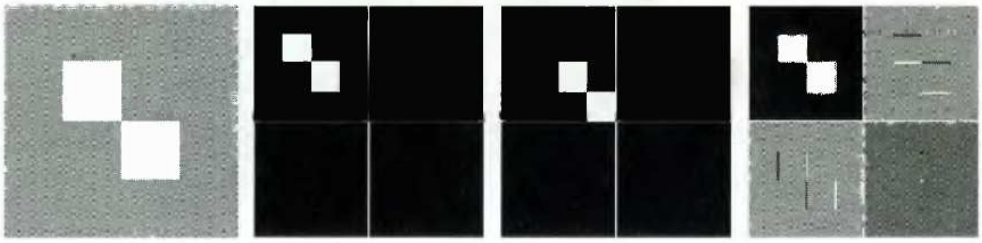
★7.22 The two-dimensional fast wavelet transform is similar to the pyramidal coding scheme of Section 7.2.1. How are they similar? Given the three-scale wavelet transform in Fig. 7.8(a), how would you construct the corresponding approximation pyramid? How many levels would it have?

7.23 Compute the two-dimensional wavelet transform with respect to Haar wavelets of the 2×2 image in Problem 7.9. Draw the required filter bank and label all inputs and outputs with the proper arrays.

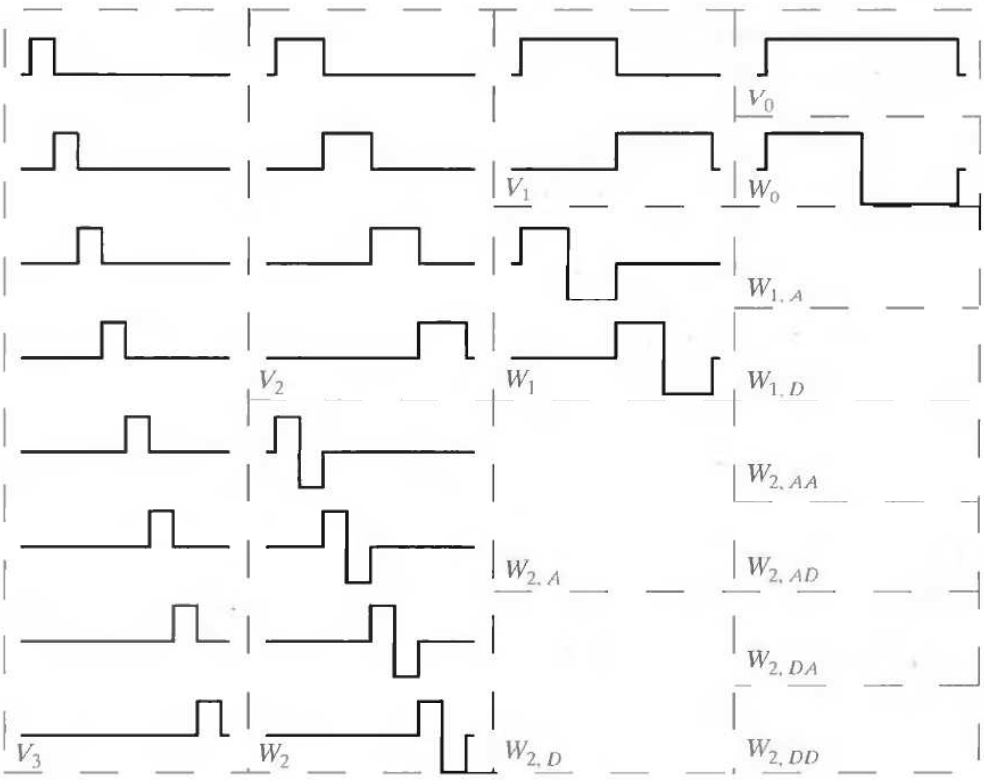
★**7.24** In the Fourier domain

$$f(x - x_0, y - y_0) \Leftrightarrow F(u, v)e^{-2\pi(ux_0/M + vy_0/N)}$$

and translation does not effect the display of $|F(u, v)|$. Using the following sequence of images, explain the translation property of wavelet transforms. The leftmost image contains two 32×32 white squares centered on a 128×128 gray background. The second image (from the left) is its single-scale wavelet transform with respect to Haar wavelets. The third is the wavelet transform of the original image after shifting it 32 pixels to the right and downward, and the final (rightmost) image is the wavelet transform of the original image after it has been shifted 1 pixel to the right and downward.



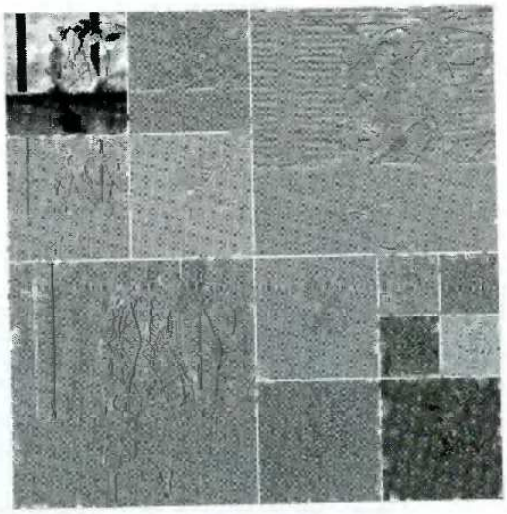
7.25 The following table shows the Haar wavelet and scaling functions for a four-scale fast wavelet transform. Sketch the additional basis functions needed for a full three-scale packet decomposition. Give the mathematical expression or expressions for determining them.



7.26 A wavelet packet decomposition of the vase from Fig. 7.1 is shown below.

(a) Draw the corresponding decomposition analysis tree, labeling all nodes with the names of the proper scaling and wavelet spaces.

(b) Draw and label the decomposition's frequency spectrum.



7.27 Using the Haar wavelet, determine the minimum entropy packet decomposition for the function $f(n) = 0.25$ for $n = 0, 1, 2, \ldots, 15$. Employ the nonnormalized Shannon entropy,

$$E[f(n)] = \sum_n f^2(n) \ln[f^2(n)]$$

as the minimization criterion. Draw the optimal tree, labeling the nodes with the computed entropy values.

8 Image Compression

But life is short and information endless ...
Abbreviation is a necessary evil and the abbreviator's business is to make the best of a job which, although intrinsically bad, is still better than nothing.

Aldous Huxley

Preview

Every day, an enormous amount of information is stored, processed, and transmitted digitally. Companies provide business associates, investors, and potential customers with financial data, annual reports, inventory, and product information over the Internet. Order entry and tracking, two of the most basic on-line transactions, are routinely performed from the comfort of one's own home. The U.S., as part of its *digital-* or *e-government* initiative, has made the entire catalog (and some of the holdings) of the Library of Congress, the world's largest library, electronically accessible; and cable television programming on demand is on the verge of becoming a reality. Because much of this on-line information is graphical or pictorial in nature, the storage (see Section 2.4.2) and communications requirements are immense. Methods of compressing the data prior to storage and/or transmission are of significant practical and commercial interest.

Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is the removal of redundant data. From a mathematical viewpoint, this amounts to transforming a 2-D pixel array into a statistically uncorrelated data set. The transformation is applied prior to storage or transmission of the image. At some later time, the compressed image is decompressed to reconstruct the original image or an approximation of it.

Interest in image compression dates back more than 35 years. The initial focus of research efforts in this field was on the development of analog methods for reducing video transmission bandwidth, a process called *bandwidth compression*. The advent of the digital computer and subsequent development of advanced integrated circuits, however, caused interest to shift from analog to digital compression approaches. With the relatively recent adoption of several key international image compression standards, the field has undergone significant growth through the practical application of the theoretic work that began in the 1940s, when C. E. Shannon and others first formulated the probabilistic view of information and its representation, transmission, and compression.

Currently, image compression is recognized as an "enabling technology." In addition to the areas just mentioned, image compression is the natural technology for handling the increased spatial resolutions of today's imaging sensors and evolving broadcast television standards. Furthermore, image compression plays a major role in many important and diverse applications, including televideoconferencing, remote sensing (the use of satellite imagery for weather and other earth-resource applications), document and medical imaging, facsimile transmission (FAX), and the control of remotely piloted vehicles in military, space, and hazardous waste management applications. In short, an ever-expanding number of applications depend on the efficient manipulation, storage, and transmission of binary, gray-scale, and color images.

In this chapter, we examine both the theoretic and practical aspects of the image compression process. Sections 8.1 through 8.3 constitute an introduction to the fundamentals that collectively form the theory of this discipline. Section 8.1 describes the data redundancies that may be exploited by image compression algorithms. A model-based paradigm for the general compression-decompression process is presented in Section 8.2. Section 8.3 examines in some detail a number of basic concepts from information theory and their role in establishing fundamental limits on the representation of information.

Sections 8.4 through 8.6 cover the practical aspects of image compression, including both the principal techniques in use and the standards that have been instrumental in increasing the scope and acceptance of this discipline. Compression techniques fall into two broad categories: *information preserving* and *lossy*. Section 8.4 addresses methods in the first category, which are particularly useful in image archiving (as in the storage of legal or medical records). These methods allow an image to be compressed and decompressed without losing information. Section 8.5 describes methods in the second category, which provide higher levels of data reduction but result in a less than perfect reproduction of the original image. Lossy image compression is useful in applications such as broadcast television, videoconferencing, and facsimile transmission, in which a certain amount of error is an acceptable trade-off for increased compression performance. Finally, Section 8.6 deals with existing and proposed image compression standards.

8.1 Fundamentals

The term *data compression* refers to the process of reducing the amount of data required to represent a given quantity of information. A clear distinction must be made between *data* and *information*. They are not synonymous. In fact, data are the means by which information is conveyed. Various amounts of data may be used to represent the same amount of information. Such might be the case, for example, if a long-winded individual and someone who is short and to the point were to relate the same story. Here, the information of interest is the story; words are the data used to relate the information. If the two individuals use a different number of words to tell the same basic story, two different versions of the story are created, and at least one includes nonessential data. That is, it contains data (or words) that either provide no relevant information or simply restate that which is already known. It is thus said to contain *data redundancy*.

Data redundancy is a central issue in digital image compression. It is not an abstract concept but a mathematically quantifiable entity. If n_1 and n_2 denote the number of information-carrying units in two data sets that represent the same information, the *relative data redundancy* R_D of the first data set (the one characterized by n_1) can be defined as

$$R_D = 1 - \frac{1}{C_R} \tag{8.1-1}$$

where C_R, commonly called the *compression ratio*, is

$$C_R = \frac{n_1}{n_2}. \tag{8.1-2}$$

For the case $n_2 = n_1$, $C_R = 1$ and $R_D = 0$, indicating that (relative to the second data set) the first representation of the information contains no redundant data. When $n_2 \ll n_1$, $C_R \to \infty$ and $R_D \to 1$, implying significant compression and highly redundant data. Finally, when $n_2 \gg n_1$, $C_R \to 0$ and $R_D \to -\infty$, indicating that the second data set contains much more data than the original representation. This, of course, is the normally undesirable case of data expansion. In general, C_R and R_D lie in the open intervals $(0, \infty)$ and $(-\infty, 1)$, respectively. A practical compression ratio, such as 10 (or 10:1), means that the first data set has 10 information carrying units (say, bits) for every 1 unit in the second or compressed data set. The corresponding redundancy of 0.9 implies that 90% of the data in the first data set is redundant.

In digital image compression, three basic data redundancies can be identified and exploited: *coding* redundancy, *interpixel* redundancy, and *psychovisual* redundancy. Data compression is achieved when one or more of these redundancies are reduced or eliminated.

8.1.1 Coding Redundancy

In Chapter 3 we developed the technique for image enhancement by histogram processing on the assumption that the gray levels of an image are random quantities. We showed that a great deal of information about the appearance of an image could be obtained from a histogram of its gray levels. In this section, we utilize a similar formulation to show how the gray-level histogram of an image also can provide a great deal of insight into the construction of codes† to reduce the amount of data used to represent it.

Let us assume, once again, that a discrete random variable r_k in the interval $[0, 1]$ represents the gray levels of an image and that each r_k occurs with probability $p_r(r_k)$. As in Chapter 3,

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0, 1, 2, \ldots, L - 1 \tag{8.1-3}$$

where L is the number of gray levels, n_k is the number of times that the kth gray level appears in the image, and n is the total number of pixels in the image. If the number of bits used to represent each value of r_k is $l(r_k)$, then the average number of bits required to represent each pixel is

$$L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k)p_r(r_k). \tag{8.1-4}$$

That is, the average length of the code words assigned to the various gray-level values is found by summing the product of the number of bits used to represent each gray level and the probability that the gray level occurs. Thus the total number of bits required to code an $M \times N$ image is MNL_{avg}.

Representing the gray levels of an image with a natural m-bit binary code‡ reduces the right-hand side of Eq. (8.1-4) to m bits. That is, $L_{\text{avg}} = m$ when m is substituted for $l(r_k)$. Then the constant m may be taken outside the summation, leaving only the sum of the $p_r(r_k)$ for $0 \leq k \leq L - 1$, which, of course, equals 1.

EXAMPLE 8.1: A simple illustration of variable-length coding.

■ An 8-level image has the gray-level distribution shown in Table 8.1. If a natural 3-bit binary code [see code 1 and $l_1(r_k)$ in Table 8.1] is used to represent the 8 possible gray levels, L_{avg} is 3 bits, because $l_1(r_k) = 3$ bits for all r_k. If code 2 in Table 8.1 is used, however, the average number of bits required to code the image is reduced to

† A *code* is a system of symbols (letters, numbers, bits, and the like) used to represent a body of information or set of events. Each piece of information or event is assigned a sequence of *code symbols*, called a *code word*. The number of symbols in each code word is its *length*. One of the most famous codes was used by Paul Revere on April 18, 1775. The phrase "one if by land, two if by sea" is often used to describe that code, in which one or two lights were used to indicate whether the British were traveling by land or sea.

‡ A *natural* (or *straight*) binary code is one in which each event or piece of information to be encoded (such as gray-level value) is assigned one of 2^m m-bit binary codes from an m-bit binary counting sequence.

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_0 = 0$	0.19	000	3	11	2
$r_1 = 1/7$	0.25	001	3	01	2
$r_2 = 2/7$	0.21	010	3	10	2
$r_3 = 3/7$	0.16	011	3	001	3
$r_4 = 4/7$	0.08	100	3	0001	4
$r_5 = 5/7$	0.06	101	3	00001	5
$r_6 = 6/7$	0.03	110	3	000001	6
$r_7 = 1$	0.02	111	3	000000	6

TABLE 8.1 Example of variable-length coding.

$$
\begin{aligned}
L_{\text{avg}} &= \sum_{k=0}^{7} l_2(r_k)p_r(r_k) \\
&= 2(0.19) + 2(0.25) + 2(0.21) + 3(0.16) + 4(0.08) \\
&\quad + 5(0.06) + 6(0.03) + 6(0.02) \\
&= 2.7 \text{ bits.}
\end{aligned}
$$

From Eq. (8.1-2), the resulting compression ratio C_R is 3/2.7 or 1.11. Thus approximately 10% of the data resulting from the use of code 1 is redundant. The exact level of redundancy can be determined from Eq. (8.1-1):

$$R_D = 1 - \frac{1}{1.11} = 0.099.$$

Figure 8.1 illustrates the underlying basis for the compression achieved by code 2. It shows both the histogram of the image [a plot of $p_r(r_k)$ versus r_k] and $l_2(r_k)$. Because these two functions are inversely proportional—that is, $l_2(r_k)$ increases as $p_r(r_k)$ decreases—the shortest code words in code 2 are assigned to the gray levels that occur most frequently in an image. ■

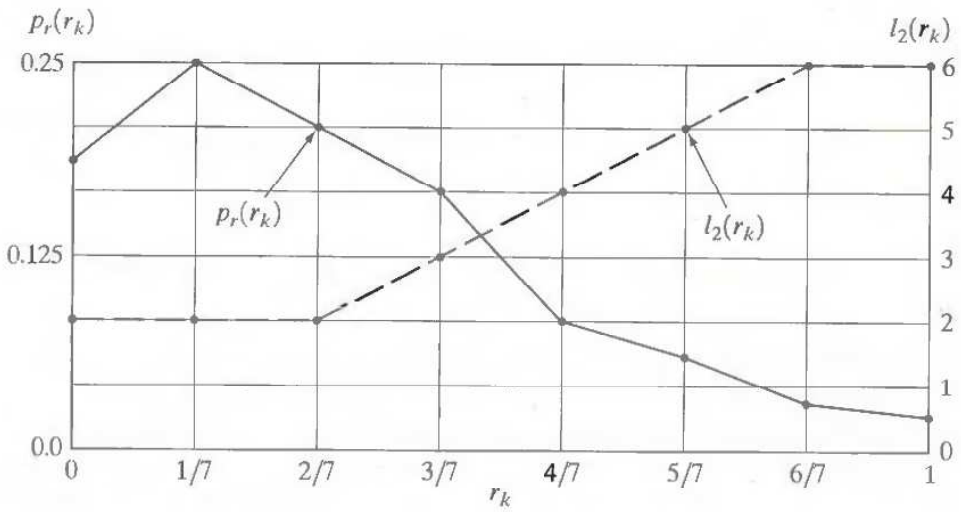


FIGURE 8.1 Graphic representation of the fundamental basis of data compression through variable-length coding.

8.1.1 Coding Redundancy

In Chapter 3 we developed the technique for image enhancement by histogram processing on the assumption that the gray levels of an image are random quantities. We showed that a great deal of information about the appearance of an image could be obtained from a histogram of its gray levels. In this section, we utilize a similar formulation to show how the gray-level histogram of an image also can provide a great deal of insight into the construction of codes[†] to reduce the amount of data used to represent it.

Let us assume, once again, that a discrete random variable r_k in the interval $[0, 1]$ represents the gray levels of an image and that each r_k occurs with probability $p_r(r_k)$. As in Chapter 3,

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0, 1, 2, \ldots, L - 1 \tag{8.1-3}$$

where L is the number of gray levels, n_k is the number of times that the kth gray level appears in the image, and n is the total number of pixels in the image. If the number of bits used to represent each value of r_k is $l(r_k)$, then the average number of bits required to represent each pixel is

$$L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k)p_r(r_k). \tag{8.1-4}$$

That is, the average length of the code words assigned to the various gray-level values is found by summing the product of the number of bits used to represent each gray level and the probability that the gray level occurs. Thus the total number of bits required to code an $M \times N$ image is MNL_{avg}.

Representing the gray levels of an image with a natural m-bit binary code[‡] reduces the right-hand side of Eq. (8.1-4) to m bits. That is, $L_{\text{avg}} = m$ when m is substituted for $l(r_k)$. Then the constant m may be taken outside the summation, leaving only the sum of the $p_r(r_k)$ for $0 \leq k \leq L - 1$, which, of course, equals 1.

EXAMPLE 8.1: A simple illustration of variable-length coding.

■ An 8-level image has the gray-level distribution shown in Table 8.1. If a natural 3-bit binary code [see code 1 and $l_1(r_k)$ in Table 8.1] is used to represent the 8 possible gray levels, L_{avg} is 3 bits, because $l_1(r_k) = 3$ bits for all r_k. If code 2 in Table 8.1 is used, however, the average number of bits required to code the image is reduced to

[†] A *code* is a system of symbols (letters, numbers, bits, and the like) used to represent a body of information or set of events. Each piece of information or event is assigned a sequence of *code symbols*, called a *code word*. The number of symbols in each code word is its *length*. One of the most famous codes was used by Paul Revere on April 18, 1775. The phrase "one if by land, two if by sea" is often used to describe that code, in which one or two lights were used to indicate whether the British were traveling by land or sea.

[‡] A *natural* (or *straight*) binary code is one in which each event or piece of information to be encoded (such as gray-level value) is assigned one of 2^m m-bit binary codes from an m-bit binary counting sequence.

In the preceding example, assigning fewer bits to the more probable gray levels than to the less probable ones achieves data compression. This process commonly is referred to as *variable-length coding*. If the gray levels of an image are coded in a way that uses more code symbols than absolutely necessary to represent each gray level [that is, the code fails to minimize Eq. (8.1-4)], the resulting image is said to contain *coding redundancy*. In general, coding redundancy is present when the codes assigned to a set of events (such as gray-level values) have not been selected to take full advantage of the probabilities of the events. It is almost always present when an image's gray levels are represented with a straight or natural binary code. In this case, the underlying basis for the coding redundancy is that images are typically composed of objects that have a regular and somewhat predictable morphology (shape) and reflectance, and are generally sampled so that the objects being depicted are much larger than the picture elements. The natural consequence is that, in most images, certain gray levels are more probable than others (that is, the histograms of most images are not uniform). A natural binary coding of their gray levels assigns the same number of bits to both the most and least probable values, thus failing to minimize Eq. (8.1-4) and resulting in coding redundancy.

8.1.2 Interpixel Redundancy

Consider the images shown in Figs. 8.2(a) and (b). As Figs. 8.2(c) and (d) show, these images have virtually identical histograms. Note also that both histograms are trimodal, indicating the presence of three dominant ranges of gray-level values. Because the gray levels in these images are not equally probable, variable-length coding can be used to reduce the coding redundancy that would result from a straight or natural binary encoding of their pixels. The coding process, however, would not alter the level of correlation between the pixels within the images. In other words, the codes used to represent the gray levels of each image have nothing to do with the correlation between pixels. These correlations result from the structural or geometric relationships between the objects in the image.

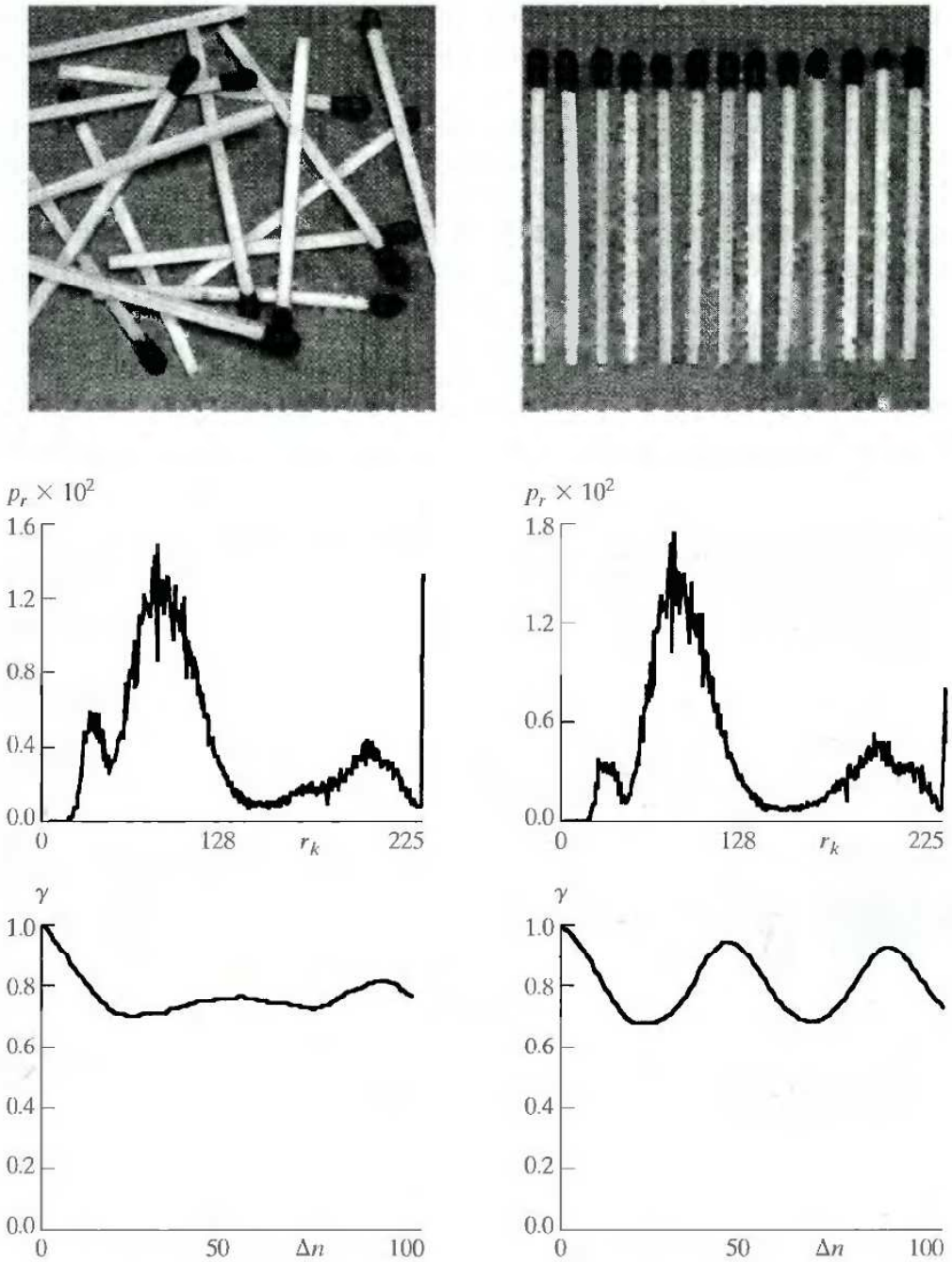
Figures 8.2(e) and (f) show the respective *autocorrelation coefficients* computed along one line of each image. These coefficients were computed using a normalized version of Eq. (4.6-30) in which

$$\gamma(\Delta n) = \frac{A(\Delta n)}{A(0)} \tag{8.1-5}$$

where

$$A(\Delta n) = \frac{1}{N - \Delta n} \sum_{y=0}^{N-1-\Delta n} f(x, y) f(x, y + \Delta n). \tag{8.1-6}$$

The scaling factor in Eq. (8.1-6) accounts for the varying number of sum terms that arise for each integer value of Δn. Of course, Δn must be strictly less than N, the number of pixels on a line. The variable x is the coordinate of the line used in the computation. Note the dramatic difference between the shape of the functions shown in Figs. 8.2(e) and (f). Their shapes can be qualitatively related to the structure in the images in Figs. 8.2(a) and (b). This relationship is particularly noticeable



a b
c d
e f

FIGURE 8.2 Two images and their gray-level histograms and normalized autocorrelation coefficients along one line.

in Fig. 8.2(f), where the high correlation between pixels separated by 45 and 90 samples can be directly related to the spacing between the vertically oriented matches of Fig. 8.2(b). In addition, the adjacent pixels of both images are highly correlated. When Δn is 1, γ is 0.9922 and 0.9928 for the images of Figs. 8.2(a) and (b), respectively. These values are typical of most properly sampled television images.

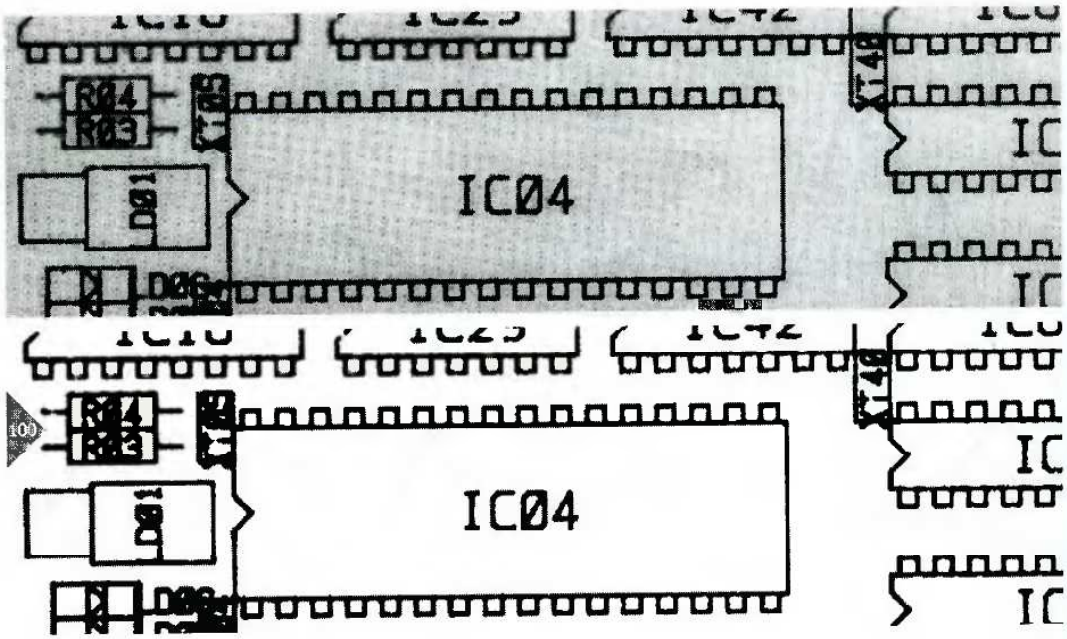
These illustrations reflect another important form of data redundancy—one directly related to the interpixel correlations within an image. Because the value of any given pixel can be reasonably predicted from the value of its neighbors, the information carried by individual pixels is relatively small. Much of the visual contribution of a single pixel to an image is redundant; it could have been guessed on the basis of the values of its neighbors. A variety of names, including

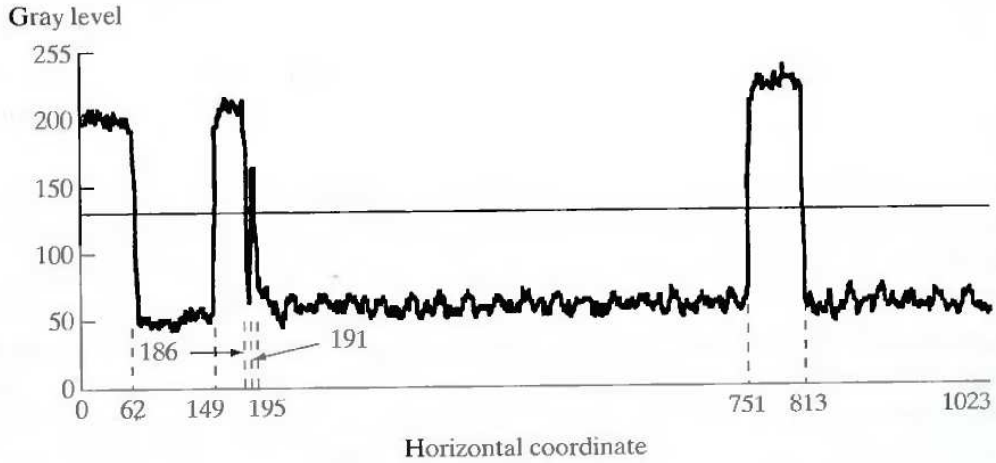
spatial redundancy, geometric redundancy, and *interframe redundancy*, have been coined to refer to these interpixel dependencies. We use the term *interpixel redundancy* to encompass them all.

In order to reduce the interpixel redundancies in an image, the 2-D pixel array normally used for human viewing and interpretation must be transformed into a more efficient (but usually "nonvisual") format. For example, the differences between adjacent pixels can be used to represent an image. Transformations of this type (that is, those that remove interpixel redundancy) are referred to as *mappings*. They are called *reversible mappings* if the original image elements can be reconstructed from the transformed data set.

EXAMPLE 8.2: A simple illustration of run-length coding.

■ Figure 8.3 illustrates a simple mapping procedure. Figure 8.3(a) depicts a 1-in. by 3-in. section of an electrical assembly drawing that has been sampled at





a
b
c
d

FIGURE 8.3 Illustration of run-length coding: (a) original image. (b) Binary image with line 100 marked. (c) Line profile and binarization threshold. (d) Run-length code.

Line 100: (1, 63) (0, 87) (1, 37) (0, 5) (1, 4) (0, 556) (1, 62) (0, 210)

approximately 330 dpi (dots per inch). Figure 8.3(b) shows a binary version of this drawing, and Fig. 8.3(c) depicts the gray-level profile of one line of the image and the threshold used to obtain the binary version (see Section 3.1). Because the binary image contains many regions of constant intensity, a more efficient representation can be constructed by mapping the pixels along each scan line $f(x, 0), f(x, 1), \ldots, f(x, N-1)$ into a sequence of pairs (g_1, w_1), $(g_2, w_2), \ldots$, in which g_i denotes the ith gray level encountered along the line and w_i the run length of the ith run. In other words, the thresholded image can be more efficiently represented by the value and length of its constant gray-level runs (a nonvisual representation) than by a 2-D array of binary pixels.

Figure 8.3(d) shows the run-length encoded data corresponding to the thresholded line profile of Fig. 8.3(c). Only 88 bits are needed to represent the 1024 bits of binary data. In fact, the entire 1024×343 section shown in Fig. 8.3(b) can be reduced to 12,166 runs. As 11 bits are required to represent each run-length pair, the resulting compression ratio and corresponding relative redundancy are

$$C_R = \frac{(1024)(343)(1)}{(12166)(11)} = 2.63$$

and

$$R_D = 1 - \frac{1}{2.63} = 0.62.$$

■

8.1.3 Psychovisual Redundancy

We noted in Section 2.1 that the brightness of a region, as perceived by the eye, depends on factors other than simply the light reflected by the region. For example, intensity variations (Mach bands) can be perceived in an area of constant intensity. Such phenomena result from the fact that the eye does not respond with equal sensitivity to all visual information. Certain information simply has less relative importance than other information in normal visual processing. This information is said to be *psychovisually redundant*. It can be eliminated without significantly impairing the quality of image perception.

That psychovisual redundancies exist should not come as a surprise, because human perception of the information in an image normally does not involve quantitative analysis of every pixel value in the image. In general, an observer searches for distinguishing features such as edges or textural regions and mentally combines them into recognizable groupings. The brain then correlates these groupings with prior knowledge in order to complete the image interpretation process.

Psychovisual redundancy is fundamentally different from the redundancies discussed earlier. Unlike coding and interpixel redundancy, psychovisual redundancy is associated with real or quantifiable visual information. Its elimination is possible only because the information itself is not essential for normal visual processing. Since the elimination of psychovisually redundant data results in a loss of quantitative information, it is commonly referred to as *quantization*. This terminology is consistent with normal usage of the word, which generally

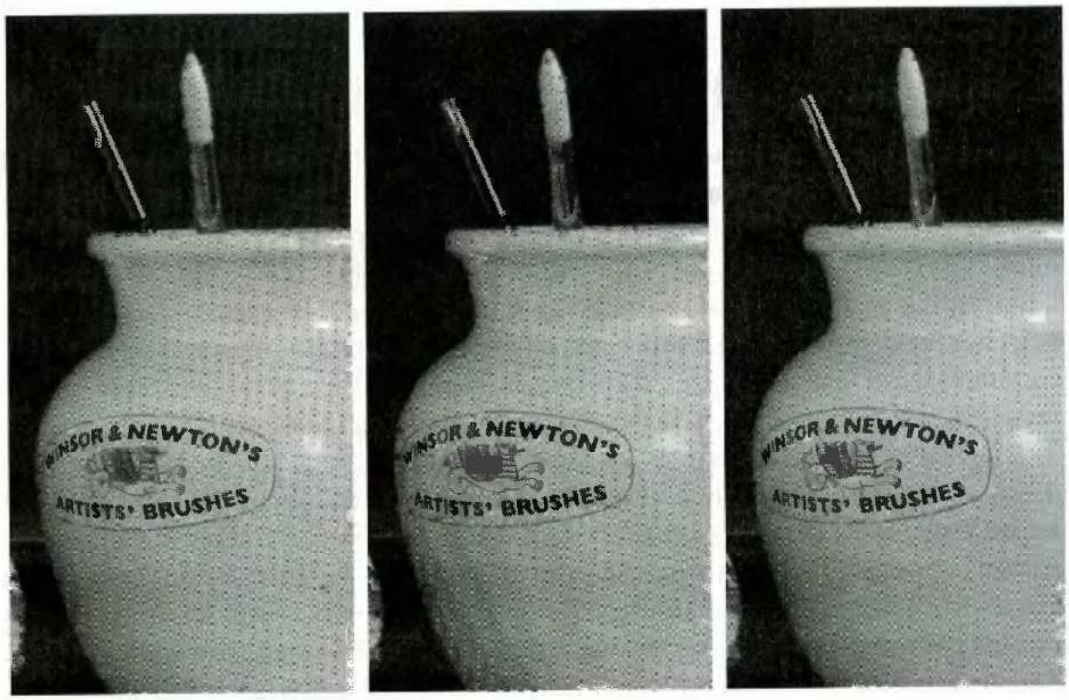
means the mapping of a broad range of input values to a limited number of output values, as discussed in Section 2.4. As it is an irreversible operation (visual information is lost), quantization results in lossy data compression.

EXAMPLE 8.3: Compression by quantization.

■ Consider the images in Fig. 8.4. Figure 8.4(a) shows a monochrome image with 256 possible gray levels. Figure 8.4(b) shows the same image after uniform quantization to four bits or 16 possible levels. The resulting compression ratio is 2:1. Note, as discussed in Section 2.4, that false contouring is present in the previously smooth regions of the original image. This is the natural visual effect of more coarsely representing the gray levels of the image.

Figure 8.4(c) illustrates the significant improvements possible with quantization that takes advantage of the peculiarities of the human visual system. Although the compression ratio resulting from this second quantization procedure also is 2:1, false contouring is greatly reduced at the expense of some additional but less objectionable graininess. The method used to produce this result is known as *improved gray-scale* (IGS) *quantization*. It recognizes the eye's inherent sensitivity to edges and breaks them up by adding to each pixel a pseudorandom number, which is generated from the low-order bits of neighboring pixels, before quantizing the result. Because the low-order bits are fairly random (see the bit planes in Section 3.2.4), this amounts to adding a level of randomness, which depends on the local characteristics of the image, to the artificial edges normally associated with false contouring.

Table 8.2 illustrates this method. A sum—initially set to zero—is first formed from the current 8-bit gray-level value and the four least significant bits of a previously generated sum. If the four most significant bits of the current value are 1111_2, however, 0000_2 is added instead. The four most significant bits of the resulting sum are used as the coded pixel value. ■



a b c

FIGURE 8.4 (a) Original image. (b) Uniform quantization to 16 levels. (c) IGS quantization to 16 levels.

Pixel	Gray Level	Sum	IGS Code
$i-1$	N/A	00000000	N/A
i	01101100	01101100	0110
$i+1$	10001011	10010111	1001
$i+2$	10000111	10001110	1000
$i+3$	11110100	11110100	1111

TABLE 8.2 IGS quantization procedure.

Improved gray-scale quantization is typical of a large group of quantization procedures that operate directly on the gray levels of the image to be compressed. They usually entail a decrease in the image's spatial and/or gray-scale resolution. The resulting false contouring or other related effects necessitates the use of heuristic techniques to compensate for the visual impact of quantization. The normal 2:1 line interlacing approach used in commercial broadcast television, for example, is a form of quantization in which interleaving portions of adjacent frames allows reduced video scanning rates with little decrease in perceived image quality.

8.1.4 Fidelity Criteria

As noted previously, removal of psychovisually redundant data results in a loss of real or quantitative visual information. Because information of interest may be lost, a repeatable or reproducible means of quantifying the nature and extent of information loss is highly desirable. Two general classes of criteria are used as the basis for such an assessment: (1) objective fidelity criteria and (2) subjective fidelity criteria.

When the level of information loss can be expressed as a function of the original or input image and the compressed and subsequently decompressed output image, it is said to be based on an *objective fidelity criterion*. A good example is the root-mean-square (rms) error between an input and output image. Let $f(x, y)$ represent an input image and let $\hat{f}(x, y)$ denote an estimate or approximation of $f(x, y)$ that results from compressing and subsequently decompressing the input. For any value of x and y, the error $e(x, y)$ between $f(x, y)$ and $\hat{f}(x, y)$ can be defined as

$$e(x, y) = \hat{f}(x, y) - f(x, y) \tag{8.1-7}$$

so that the total error between the two images is

$$\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x, y) - f(x, y)]$$

where the images are of size $M \times N$. The *root-mean-square error*, e_{rms}, between $f(x, y)$ and $\hat{f}(x, y)$ then is the square root of the squared error averaged over the $M \times N$ array, or

$$e_{rms} = \left[\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}[\hat{f}(x, y) - f(x, y)]^2\right]^{1/2} \tag{8.1-8}$$

A closely related objective fidelity criterion is the mean-square signal-to-noise ratio of the compressed-decompressed image. If $\hat{f}(x, y)$ is considered [by a simple rearrangement of the terms in Eq. (8.1-7)] to be the sum of the original image $f(x, y)$ and a noise signal $e(x, y)$, the *mean-square signal-to-noise ratio* of the output image, denoted SNR_{ms}, is

$$SNR_{\text{ms}} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x, y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x, y) - f(x, y)\right]^2}. \tag{8.1-9}$$

The rms value of the signal-to-noise ratio, denoted SNR_{rms}, is obtained by taking the square root of Eq. (8.1-9).

Although objective fidelity criteria offer a simple and convenient mechanism for evaluating information loss, most decompressed images ultimately are viewed by humans. Consequently, measuring image quality by the subjective evaluations of a human observer often is more appropriate. This can be accomplished by showing a "typical" decompressed image to an appropriate cross section of viewers and averaging their evaluations. The evaluations may be made using an absolute rating scale or by means of side-by-side comparisons of $f(x, y)$ and $\hat{f}(x, y)$. Table 8.3 shows one possible absolute rating scale. Side-by-side comparisons can be done with a scale such as $\{-3, -2, -1, 0, 1, 2, 3\}$ to represent the subjective evaluations $\{$*much worse, worse, slightly worse, the same, slightly better, better, much better*$\}$, respectively. In either case, the evaluations are said to be based on *subjective fidelity criteria*.

EXAMPLE 8.4: Comparisons of image quality.

■ The rms errors in the quantized images of Figs. 8.4(b) and (c) are 6.93 and 6.78 gray levels, respectively. The corresponding rms signal-to-noise ratios are 10.25 and 10.39. Although these values are quite similar, a subjective evaluation of the visual quality of the two coded images might result in a *marginal* rating for the image in Fig. 8.4(b) and a *passable* rating for that in Fig. 8.4(c). ■

TABLE 8.3 Rating scale of the Television Allocations Study Organization. (Frendendall and Behrend.)

Value	Rating	Description
1	Excellent	An image of extremely high quality, as good as you could desire.
2	Fine	An image of high quality, providing enjoyable viewing. Interference is not objectionable.
3	Passable	An image of acceptable quality. Interference is not objectionable.
4	Marginal	An image of poor quality; you wish you could improve it. Interference is somewhat objectionable.
5	Inferior	A very poor image, but you could watch it. Objectionable interference is definitely present.
6	Unusable	An image so bad that you could not watch it.

8.2 Image Compression Models

In Section 8.1 we discussed individually three general techniques for reducing or compressing the amount of data required to represent an image. However, these techniques typically are combined to form practical image compression systems. In this section, we examine the overall characteristics of such a system and develop a general model to represent it.

As Fig. 8.5 shows, a compression system consists of two distinct structural blocks: an *encoder* and a *decoder*.† An input image $f(x, y)$ is fed into the encoder, which creates a set of symbols from the input data. After transmission over the *channel*, the encoded representation is fed to the decoder, where a reconstructed output image $\hat{f}(x, y)$ is generated. In general, $\hat{f}(x, y)$ may or may not be an exact replica of $f(x, y)$. If it is, the system is error free or information preserving; if not, some level of distortion is present in the reconstructed image.

Both the encoder and decoder shown in Fig. 8.5 consist of two relatively independent functions or subblocks. The encoder is made up of a *source encoder*, which removes input redundancies, and a *channel encoder*, which increases the noise immunity of the source encoder's output. As would be expected, the decoder includes a *channel decoder* followed by a *source decoder*. If the channel between the encoder and decoder is noise free (not prone to error), the channel encoder and decoder are omitted, and the general encoder and decoder become the source encoder and decoder, respectively.

8.2.1 The Source Encoder and Decoder

The source encoder is responsible for reducing or eliminating any coding, interpixel, or psychovisual redundancies in the input image. The specific application and associated fidelity requirements dictate the best encoding approach to use in any given situation. Normally, the approach can be modeled by a series of three independent operations. As Fig. 8.6(a) shows, each operation is designed to reduce one of the three redundancies described in Section 8.1. Figure 8.6(b) depicts the corresponding source decoder.

In the first stage of the source encoding process, the *mapper* transforms the input data into a (usually nonvisual) format designed to reduce interpixel redundancies in the input image. This operation generally is reversible and may or may not reduce directly the amount of data required to represent the image. Run-length coding (Sections 8.1.2 and 8.4.3) is an example of a mapping that

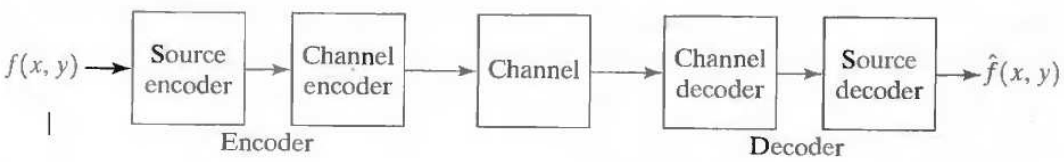
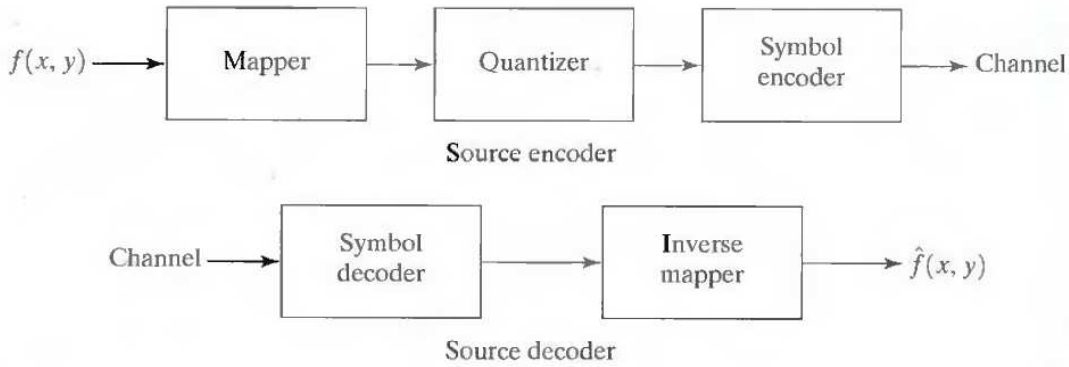


FIGURE 8.5 A general compression system model.

†It would be reasonable to expect these blocks to be called the "compressor" and "decompressor." The terms *encoder* and *decoder* reflect the influence of information theory (to be discussed in Section 8.3) on the field of image compression.



a
b

FIGURE 8.6 (a) Source encoder and (b) source decoder model.

directly results in data compression in this initial stage of the overall source encoding process. The representation of an image by a set of transform coefficients (Section 8.5.2) is an example of the opposite case. Here, the mapper transforms the image into an array of coefficients, making its interpixel redundancies more accessible for compression in later stages of the encoding process.

The second stage, or *quantizer* block in Fig. 8.6(a), reduces the accuracy of the mapper's output in accordance with some preestablished fidelity criterion. This stage reduces the psychovisual redundancies of the input image. As noted in Section 8.1.3, this operation is irreversible. Thus it must be omitted when error-free compression is desired.

In the third and final stage of the source encoding process, the *symbol coder* creates a fixed- or variable-length code to represent the quantizer output and maps the output in accordance with the code. The term *symbol coder* distinguishes this coding operation from the overall source encoding process. In most cases, a variable-length code is used to represent the mapped and quantized data set. It assigns the shortest code words to the most frequently occurring output values and thus reduces coding redundancy. The operation, of course, is reversible. Upon completion of the symbol coding step, the input image has been processed to remove each of the three redundancies described in Section 8.1.

Figure 8.6(a) shows the source encoding process as three successive operations, but all three operations are not necessarily included in every compression system. Recall, for example, that the quantizer must be omitted when error-free compression is desired. In addition, some compression techniques normally are modeled by merging blocks that are physically separate in Fig. 8.6(a). In the predictive compression systems of Section 8.5.1, for instance, the mapper and quantizer are often represented by a single block, which simultaneously performs both operations.

The source decoder shown in Fig. 8.6(b) contains only two components: a *symbol decoder* and an *inverse mapper*. These blocks perform, in reverse order, the inverse operations of the source encoder's symbol encoder and mapper blocks. Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general source decoder model shown in Fig. 8.6(b).

8.2.2 The Channel Encoder and Decoder

The channel encoder and decoder play an important role in the overall encoding-decoding process when the channel of Fig. 8.5 is noisy or prone to error. They are designed to reduce the impact of channel noise by inserting a controlled form of redundancy into the source encoded data. As the output of the source encoder contains little redundancy, it would be highly sensitive to transmission noise without the addition of this "controlled redundancy."

One of the most useful channel encoding techniques was devised by R. W. Hamming (Hamming [1950]). It is based on appending enough bits to the data being encoded to ensure that some minimum number of bits must change between valid code words. Hamming showed, for example, that if 3 bits of redundancy are added to a 4-bit word, so that the *distance*† between any two valid code words is 3, all single-bit errors can be detected *and* corrected. (By appending additional bits of redundancy, multiple-bit errors can be detected and corrected.) The 7-bit *Hamming (7,4) code* word $h_1 h_2 \ldots h_5 h_6 h_7$ associated with a 4-bit binary number $b_3 b_2 b_1 b_0$ is

$$\begin{aligned} h_1 &= b_3 \oplus b_2 \oplus b_0 & h_3 &= b_3 \\ h_2 &= b_3 \oplus b_1 \oplus b_0 & h_5 &= b_2 \\ h_4 &= b_2 \oplus b_1 \oplus b_0 & h_6 &= b_1 \\ & & h_7 &= b_0 \end{aligned} \tag{8.2-1}$$

where $\oplus$ denotes the exclusive OR operation. Note that bits h_1, h_2, and h_4 are even-parity bits for the bit fields $b_3 b_2 b_0$, $b_3 b_1 b_0$, and $b_2 b_1 b_0$, respectively. (Recall that a string of binary bits has even parity if the number of bits with a value of 1 is even.)

To decode a Hamming encoded result, the channel decoder must check the encoded value for odd parity over the bit fields in which even parity was previously established. A single-bit error is indicated by a nonzero parity word $c_4 c_2 c_1$, where

$$\begin{aligned} c_1 &= h_1 \oplus h_3 \oplus h_5 \oplus h_7 \\ c_2 &= h_2 \oplus h_3 \oplus h_6 \oplus h_7 \\ c_4 &= h_4 \oplus h_5 \oplus h_6 \oplus h_7. \end{aligned} \tag{8.2-2}$$

If a nonzero value is found, the decoder simply complements the code word bit position indicated by the parity word. The decoded binary value is then extracted from the corrected code word as $h_3 h_5 h_6 h_7$.

EXAMPLE 8.5: Hamming encoding.

■ Consider the transmission of the 4-bit IGS data of Table 8.2 over a noisy communication channel. A single-bit error could cause a decompressed pixel to deviate from its correct value by as many as 128 gray levels.‡ A Hamming

†The *distance* between two code words is defined as the minimum number of digits that must change in one word so that the other word results. For example, the distance between 101101 and 011101 is 2. The *minimum distance* of a code is the smallest number of digits by which any two code words differ.

‡A simple procedure for decompressing 4-bit IGS data is to multiply the decimal equivalent of the IGS value by 16. For example, if the IGS value is 1110, the decompressed gray level is (14)(16) or 224. If the most significant bit of this IGS value was incorrectly transmitted as a 0, the decompressed gray level becomes 96. The resulting error is 128 gray levels.

channel encoder can be utilized to increase the noise immunity of this source encoded IGS data by inserting enough redundancy to allow the detection and correction of single-bit errors. From Eq. (8.2-1), the Hamming encoded value for the first IGS value in Table 8.2 is 1100110_2. Because the Hamming channel encoder increases the number of bits required to represent the IGS value from 4 to 7, the 2:1 compression ratio noted in the IGS example is reduced to 8/7 or 1.14:1. This reduction in compression is the price paid for increased noise immunity. ■

8.3 Elements of Information Theory

In Section 8.1 we introduced several ways to reduce the amount of data used to represent an image. The question that naturally arises is: How few data actually are needed to represent an image? That is, is there a minimum amount of data that is sufficient to describe completely an image without loss of information? Information theory provides the mathematical framework to answer this and related questions.

8.3.1 Measuring Information

See inside front cover
Consult the book web site for a brief review of probability theory.

The fundamental premise of information theory is that the generation of information can be modeled as a probabilistic process that can be measured in a manner that agrees with intuition. In accordance with this supposition, a random event E that occurs with probability $P(E)$ is said to contain

$$I(E) = \log \frac{1}{P(E)} = -\log P(E) \tag{8.3-1}$$

units of information. The quantity $I(E)$ often is called the *self-information* of E. Generally speaking, the amount of self-information attributed to event E is inversely related to the probability of E. If $P(E) = 1$ (that is, the event always occurs), $I(E) = 0$ and no information is attributed to it. That is, because no uncertainty is associated with the event, no information would be transferred by communicating that the event has occurred. However, if $P(E) = 0.99$, communicating that E has occurred conveys some small amount of information. Communicating that E has *not* occurred conveys more information, because this outcome is less likely.

The base of the logarithm in Eq. (8.3-1) determines the unit used to measure information.† If the base m logarithm is used, the measurement is said to be in m-ary units. If the base 2 is selected, the resulting unit of information is called a *bit*. Note that if $P(E) = 1/2$, $I(E) = -\log_2 1/2$, or 1 bit. That is, 1 bit is the amount of information conveyed when one of two possible equally likely events occurs. A simple example of such a situation is flipping a coin and communicating the result.

† When we do not explicitly specify the base of the log used in an expression, the result may be interpreted in any base and corresponding information unit.

8.3.2 The Information Channel

When self-information is transferred between an information source and a user of the information, the source of information is said to be connected to the user of information by an *information channel*. The information channel is the physical medium that links the source to the user. It may be a telephone line, an electromagnetic energy propagation path, or a wire in a digital computer. Figure 8.7 shows a simple mathematical model for a discrete information system. Here, the parameter of particular interest is the system's *capacity*, defined as its ability to transfer information.

Let us assume that the information source in Fig. 8.7 generates a random sequence of symbols from a finite or countably infinite set of possible symbols. That is, the output of the source is a discrete random variable. The set of source symbols $\{a_1, a_2, \ldots, a_J\}$ is referred to as the *source alphabet* A, and the elements of the set, denoted a_j, are called *symbols* or *letters*. The probability of the event that the source will produce symbol a_j is $P(a_j)$, and

$$\sum_{j=1}^{J} P(a_j) = 1. \tag{8.3-2}$$

A $J \times 1$ vector $\mathbf{z} = [P(a_1), P(a_2), \ldots, P(a_J)]^T$ customarily is used to represent the set of all source symbol probabilities $\{P(a_1), P(a_2), \ldots, P(a_J)\}$. The finite *ensemble* $(A, \mathbf{z})$ describes the information source completely.

The probability that the discrete source will emit symbol a_j is $P(a_j)$, so the self-information generated by the production of a single source symbol is, in accordance with Eq. (8.3-1), $I(a_j) = -\log P(a_j)$. If k source symbols are generated, the law of large numbers stipulates that, for a sufficiently large value of k, symbol a_j will (on average) be output $kP(a_j)$ times. Thus the average self-information obtained from k outputs is

$$-kP(a_1)\log P(a_1) - kP(a_2)\log P(a_2) - \ldots - kP(a_J)\,kP(a_J)$$

or

$$-k\sum_{j=1}^{J} P(a_j)\log P(a_j).$$

The average information per source output, denoted $H(\mathbf{z})$, is

$$H(\mathbf{z}) = -\sum_{j=1}^{J} P(a_j)\log P(a_j). \tag{8.3-3}$$

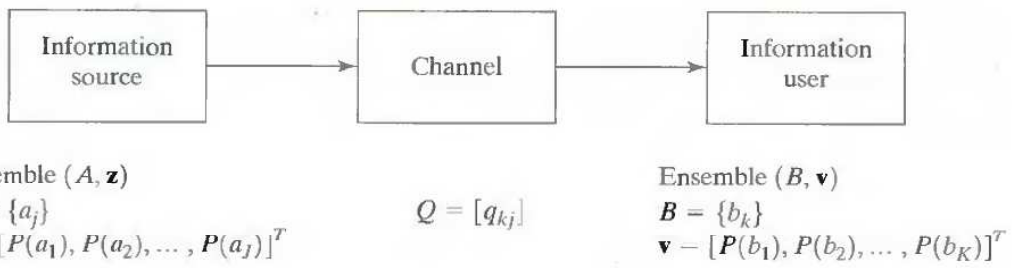


FIGURE 8.7 A simple information system.

This quantity is called the *uncertainty* or *entropy* of the source. It defines the average amount of information (in m-ary units per symbol) obtained by observing a single source output. As its magnitude increases, more uncertainty and thus more information is associated with the source. If the source symbols are equally probable, the entropy or uncertainty of Eq. (8.3-3) is maximized and the source provides the greatest possible average information per source symbol.

Having modeled the information source, we can develop the input-output characteristics of the information channel rather easily. Because we modeled the input to the channel in Fig. 8.7 as a discrete random variable, the information transferred to the output of the channel is also a discrete random variable. Like the source random variable, it takes on values from a finite or countably infinite set of symbols $\{b_1, b_2, \ldots, b_K\}$ called the *channel alphabet, B*. The probability of the event that symbol b_k is presented to the *information user* is $P(b_k)$. The finite ensemble $(B, \mathbf{v})$, where $\mathbf{v} = [P(b_1), P(b_2), \ldots, P(b_K)]^T$, describes the channel output completely and thus the information received by the user.

The probability $P(b_k)$ of a given channel output and the probability distribution of the source $\mathbf{z}$ are related by the expression†

$$P(b_k) = \sum_{j=1}^{J} P(b_k \mid a_j) P(a_j) \tag{8.3-4}$$

where $P(b_k \mid a_j)$ is the conditional probability that output symbol b_k is received, given that source symbol a_j was generated. If the conditional probabilities referenced in Eq. (8.3-4) are arranged in a matrix $K \times J$ matrix $\mathbf{Q}$, such that

$$\mathbf{Q} = \begin{bmatrix} P(b_1 \mid a_1) & P(b_1 \mid a_2) & \cdots & P(b_1 \mid a_J) \\ P(b_2 \mid a_1) & \cdot & \cdots & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & & & \\ P(b_K \mid a_1) & P(b_K \mid a_2) & \cdots & P(b_K \mid a_J) \end{bmatrix} \tag{8.3-5}$$

then the probability distribution of the complete output alphabet can be computed from

$$\mathbf{v} = \mathbf{Q}\mathbf{z}. \tag{8.3-6}$$

Matrix $\mathbf{Q}$, with elements $q_{kj} = P(b_k \mid a_j)$, is referred to as the *forward channel transition matrix* or by the abbreviated term *channel matrix*.

To determine the capacity of an information channel with forward channel transition matrix $\mathbf{Q}$, the entropy of the information source must first be computed under the assumption that the information user observes a particular output b_k. Equation (8.3-4) defines a distribution of source symbols for any observed b_k, so each b_k has one *conditional entropy function*. Based on the steps leading to Eq. (8.3-3), this conditional entropy function, denoted $H(\mathbf{z} \mid b_k)$, can be written as

† One of the fundamental laws of probability theory is that, for an arbitrary event D and t mutually exclusive events $C_1, C_2, \ldots, C_t$, the total probability of D is $P(D) = P(D \mid C_1)P(C_1) + \ldots + P(D \mid C_t)P(C_t)$.

$$H(\mathbf{z}|b_k) = -\sum_{j=1}^{J} P(a_j|b_k)\log P(a_j|b_k) \tag{8.3-7}$$

where $P(a_j|b_k)$ is the probability that symbol a_j was transmitted by the source, given that the user received b_k. The expected (average) value of this expression over all b_k is

$$H(\mathbf{z}|\mathbf{v}) = \sum_{k=1}^{K} H(\mathbf{z}|b_k)P(b_k), \tag{8.3-8}$$

which, after substitution of Eq. (8.3-7) for $H(\mathbf{z}|b_k)$ and some minor rearrangement,† can be written as

$$H(\mathbf{z}|\mathbf{v}) = -\sum_{j=1}^{J}\sum_{k=1}^{K} P(a_j, b_k)\log P(a_j|b_k). \tag{8.3-9}$$

Here, $P(a_j, b_k)$ is the joint probability of a_j and b_k. That is, $P(a_j, b_k)$ is the probability that a_j is transmitted *and* b_k is received.

The term $H(\mathbf{z}|\mathbf{v})$ is called the *equivocation* of $\mathbf{z}$ with respect to $\mathbf{v}$. It represents the average information of one source symbol, assuming observation of the output symbol that resulted from its generation. Because $H(\mathbf{z})$ is the average information of one source symbol, assuming no knowledge of the resulting output symbol, the difference between $H(\mathbf{z})$ and $H(\mathbf{z}|\mathbf{v})$ is the average information received upon observing a single output symbol. This difference, denoted $I(\mathbf{z}, \mathbf{v})$ and called the *mutual information* of $\mathbf{z}$ and $\mathbf{v}$, is

$$I(\mathbf{z}, \mathbf{v}) = H(\mathbf{z}) - H(\mathbf{z}|\mathbf{v}). \tag{8.3-10}$$

Substituting Eqs. (8.3-3) and (8.3-9) for $H(\mathbf{z})$ and $H(\mathbf{z}|\mathbf{v})$, and recalling that $P(a_j) = P(a_j, b_1) + P(a_j, b_2) + \cdots + P(a_j, b_K)$, yields

$$I(\mathbf{z}, \mathbf{v}) = \sum_{j=1}^{J}\sum_{k=1}^{K} P(a_j, b_k)\log\frac{P(a_j, b_k)}{P(a_j)P(b_k)}, \tag{8.3-11}$$

which, after further manipulation, can be written as

$$I(\mathbf{z}, \mathbf{v}) = \sum_{j=1}^{J}\sum_{k=1}^{K} P(a_j)q_{kj}\log\frac{q_{kj}}{\sum_{i=1}^{J} P(a_i)q_{ki}}. \tag{8.3-12}$$

Thus the average information received upon observing a single output of the information channel is a function of the input or source symbol probability vector $\mathbf{z}$ and channel matrix $\mathbf{Q}$. The minimum possible value of $I(\mathbf{z}, \mathbf{v})$ is zero and occurs when the input and output symbols are statistically independent, in which case $P(a_j, b_k) = P(a_j)P(b_k)$ and the log term in Eq. (8.3-11) is 0 for all j and k. The maximum value of $I(\mathbf{z}, \mathbf{v})$ over all possible choices of source probabilities in vector $\mathbf{z}$ is the *capacity*, C, of the channel described by channel matrix $\mathbf{Q}$. That is,

$$C = \max_{\mathbf{z}}[I(\mathbf{z}, \mathbf{v})] \tag{8.3-13}$$

† Use is made of the fact that the joint probability of two events, C and D, is $P(C, D) = P(C)P(D|C) = P(D)P(C|D)$.

where the maximum is taken over all possible input symbol probabilities. The capacity of the channel defines the maximum rate (in m-ary information units per source symbol) at which information can be transmitted reliably through the channel. Moreover, the capacity of a channel does not depend on the input probabilities of the source (that is, on how the channel is used) but is a function of the conditional probabilities defining the channel alone.

EXAMPLE 8.6: The binary case.

■ Consider a binary information source with source alphabet $A = \{a_1, a_2\} = \{0, 1\}$. The probabilities that the source will produce symbols a_1 and a_2 are $P(a_1) = p_{bs}$ and $P(a_2) = 1 - p_{bs} = \bar{p}_{bs}$, respectively. From Eq. (8.3-3), the entropy of the source is

$$H(\mathbf{z}) = -p_{bs}\log_2 p_{bs} - \bar{p}_{bs}\log_2 \bar{p}_{bs}.$$

Because $\mathbf{z} = [P(a_1), P(a_2)]^T = [p_{bs}, 1 - p_{bs}]^T$, $H(\mathbf{z})$ depends on the single parameter p_{bs}, and the right-hand side of the equation is called the *binary entropy function*, denoted $H_{bs}(\cdot)$. Thus, for example, $H_{bs}(t)$ is the function $-t\log_2 t - \bar{t}\log_2 \bar{t}$. Figure 8.8(a) shows a plot of $H_{bs}(p_{bs})$ for $0 \le p_{bs} \le 1$. Note that H_{bs} obtains its maximum value (of 1 bit) when p_{bs} is $1/2$. For all other values of p_{bs}, the source provides less than 1 bit of information.

Now assume that the information is to be transmitted over a noisy binary information channel and let the probability of an error during the transmission of any symbol be p_e. Such a channel is called a *binary symmetric channel* (BSC) and is defined by the channel matrix

$$\mathbf{Q} = \begin{bmatrix} 1 - p_e & p_e \\ p_e & 1 - p_e \end{bmatrix} = \begin{bmatrix} \bar{p}_e & p_e \\ p_e & \bar{p}_e \end{bmatrix}.$$

For each input or source symbol, the BSC produces one output b_j from the output alphabet $B = \{b_1, b_2\} = \{0, 1\}$. The probabilities of receiving output symbols b_1 and b_2 can be determined from Eq. (8.3-6):

$$\mathbf{v} = \mathbf{Qz} = \begin{bmatrix} \bar{p}_e & p_e \\ p_e & \bar{p}_e \end{bmatrix}\begin{bmatrix} p_{bs} \\ \bar{p}_{bs} \end{bmatrix} = \begin{bmatrix} \bar{p}_e p_{bs} + p_e \bar{p}_{bs} \\ p_e p_{bs} + \bar{p}_e \bar{p}_{bs} \end{bmatrix}.$$

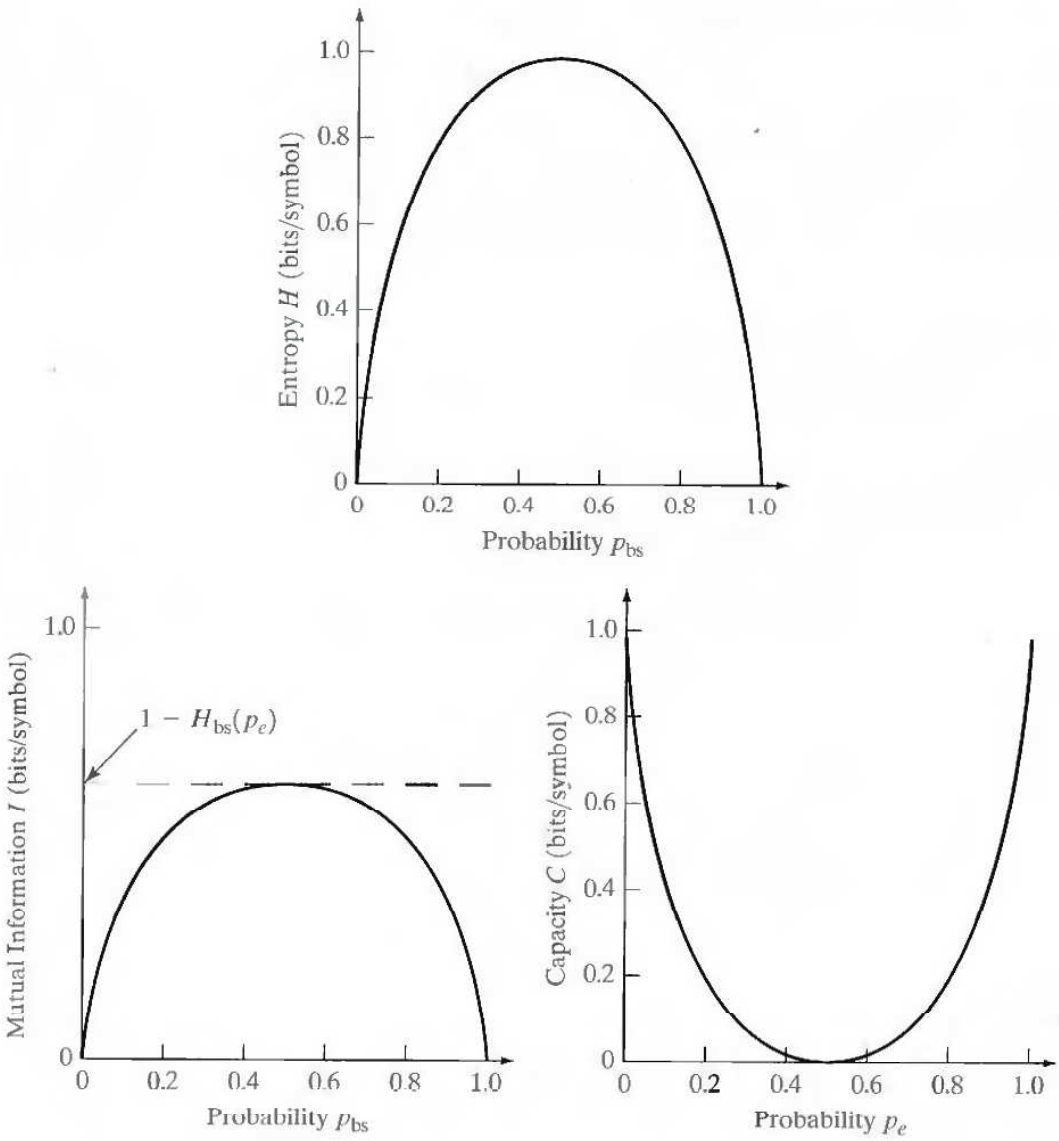
Consequently, because $\mathbf{v} = [P(b_1), P(b_2)]^T = [P(0), P(1)]^T$, the probability that the output is a 0 is $\bar{p}_e p_{bs} + p_e \bar{p}_{bs}$, and the probability that it is a 1 is $p_e p_{bs} + \bar{p}_e \bar{p}_{bs}$.

The mutual information of the BSC can now be computed from Eq. (8.3-12). Expanding the summations of this equation and collecting the appropriate terms gives

$$I(\mathbf{z}, \mathbf{v}) = H_{bs}(p_{bs}p_e + \bar{p}_{bs}\bar{p}_e) - H_{bs}(p_e)$$

where $H_{bs}(\cdot)$ is the binary entropy function of Fig. 8.8(a). For a fixed value of p_e, $I(\mathbf{z}, \mathbf{v})$ is 0 when p_{bs} is 0 or 1. Moreover, $I(\mathbf{z}, \mathbf{v})$ achieves its maximum value when the binary source symbols are equally probable. Figure 8.8(b) shows $I(\mathbf{z}, \mathbf{v})$ for all values of p_{bs} and a given channel error p_e.

In accordance with Eq. (8.3-13), the capacity of the BSC is obtained from the maximum of the mutual information over all possible source distributions. From



a
b c

FIGURE 8.8 Three binary information functions: (a) the binary entropy function; (b) the mutual information of a binary symmetric channel (BSC); (c) the capacity of the BSC.

Fig. 8.8(b), which plots $I(\mathbf{z}, \mathbf{v})$ for all possible binary source distributions (that is, for $0 \leq p_{bs} \leq 1$ or for $\mathbf{z} = [0, 1]^T$ to $\mathbf{z} = [1, 0]^T$), we see that $I(\mathbf{z}, \mathbf{v})$ is maximum (for any p_e) when $p_{bs} = 1/2$. This value of p_{bs} corresponds to source probabilities vector $\mathbf{z} = [1/2, 1/2]^T$. The corresponding value of $I(\mathbf{z}, \mathbf{v})$ is $1 - H_{bs}(p_e)$. Thus the capacity of the BSC, plotted in Fig. 8.8(c), is

$$C = 1 - H_{bs}(p_e).$$

Note that when there is no possibility of a channel error $(p_e = 0)$—as well as when a channel error is a certainty $(p_e = 1)$—the capacity of the channel obtains its maximum value of 1 bit/symbol. In either case, maximum information transfer is possible because the channel's output is completely predictable. However, when $p_e = 1/2$, the channel's output is completely unpredictable and no information can be transferred through it. ■

8.3.3 Fundamental Coding Theorems

The overall mathematical framework introduced in Section 8.3.2 is based on the model shown in Fig. 8.7, which contains an information source, channel, and user. In this section, we add a communication system to the model and examine three basic theorems regarding the coding or representation of information. As Fig. 8.9 shows, the communication system is inserted between the source and the user and consists of an encoder and decoder.

The noiseless coding theorem

When both the information channel and communication system are error free, the principal function of the communication system is to represent the source as compactly as possible. Under these circumstances, the *noiseless coding theorem*, also called *Shannon's first theorem* (Shannon [1948]), defines the minimum average code word length per source symbol that can be achieved.

A source of information with finite ensemble $(A, \mathbf{z})$ and statistically independent source symbols is called a *zero-memory* source. If we consider its output to be an n-tuple of symbols from the source alphabet (rather than a single symbol), the source output then takes on one of J^n possible values, denoted α_i, from the set of all possible n element sequences $A' = \{\alpha_1, \alpha_2, \ldots, \alpha_{J^n}\}$. In other words, each α_i (called a *block random variable*) is composed of n symbols from A. (The notation A' distinguishes the set of block symbols from A, the set of single symbols.) The probability of a given α_i is $P(\alpha_i)$, ...which is related to the single-symbol probabilities $P(a_j)$ by

$$P(\alpha_i) = P(a_{j1})P(a_{j2})\cdots P(a_{jn}) \tag{8.3-14}$$

where subscripts $j1, j2, \ldots, jn$ are used to index the n symbols from A that make up an α_i. As before, the vector $\mathbf{z}'$ (the prime is added to indicate the use of the block random variable) denotes the set of all source probabilities $\{P(\alpha_1), P(\alpha_2), \ldots, P(\alpha_{J^n})\}$, and the entropy of the source is

$$H(\mathbf{z}') = -\sum_{i=1}^{J^n} P(\alpha_i) \log P(\alpha_i).$$

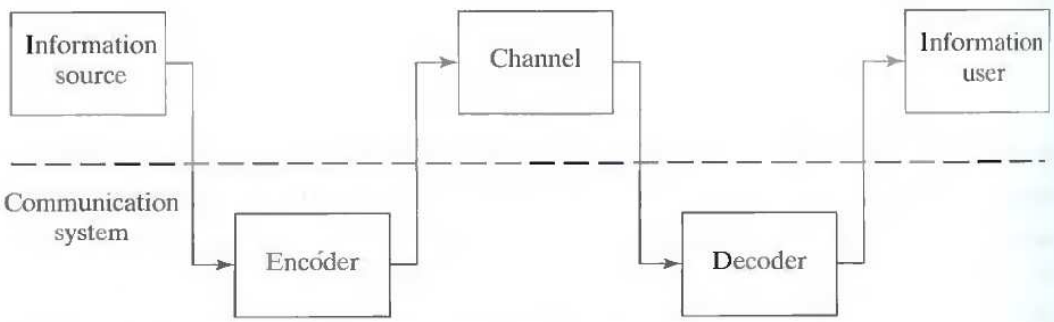


FIGURE 8.9 A communication system model.

is a lower bound on L'_{avg}/n [that is, the limit of L'_{avg}/n as n becomes large in Eq. (8.3-20) is $H(\mathbf{z})$], the *efficiency* η of any encoding strategy can be defined as

$$\eta = n\frac{H(\mathbf{z})}{L'_{avg}}. \tag{8.3-21}$$

EXAMPLE 8.7: Extension coding.

■ A zero-memory information source with source alphabet $A = \{a_1, a_2\}$ has symbol probabilities $P(a_1) = 2/3$ and $P(a_2) = 1/3$. From Eq. (8.3-3), the entropy of this source is 0.918 bits/symbol. If symbols a_1 and a_2 are represented by the binary code words 0 and 1, $L'_{avg} = 1$ bit/symbol and the resulting code efficiency is $\eta = (1)(0.918)/1$, or 0.918.

Table 8.4 summarizes the code just described and an alternative encoding based on the second extension of the source. The lower portion of Table 8.4 lists the four block symbols ($\alpha_1, \alpha_2, \alpha_3$, and α_4) in the second extension of the source. From Eq. (8.3-14) their probabilities are $4/9$, $2/9$, $2/9$, and $1/9$, respectively. In accordance with Eq. (8.3-18), the average word length of the second encoding is $17/9$ or 1.89 bits/symbol. The entropy of the second extension is twice the entropy of the nonextended source, or 1.83 bits/symbol, so the efficiency of the second encoding is $\eta = 1.83/1.89 = 0.97$. It is slightly better than the nonextended coding efficiency of 0.92. Encoding the second extension of the source reduces the average number of code bits per source symbol from 1 bit/symbol to 1.89/2 or 0.94 bits/symbol. ■

The noisy coding theorem

If the channel of Fig. 8.9 is noisy or prone to error, interest shifts from representing the information as compactly as possible to encoding it so that reliable communication is possible. The question that naturally arises is: How small can the error in communication be made?

EXAMPLE 8.8: Noisy binary channel.

■ Suppose that a BSC has a probability of error $p_e = 0.01$ (that is, 99% of all source symbols are transmitted through the channel correctly). A simple method for increasing the reliability of the communication is to repeat each message or binary symbol several times. Suppose, for example, that rather than transmitting a 0 or a 1, the coded messages 000 and 111 are used. The

TABLE 8.4 Extension coding example.

α_i	Source Symbols	$P(\alpha_i)$ Eq. (8.3-14)	$I(\alpha_i)$ Eq. (8.3-1)	$l(\alpha_i)$ Eq. (8.3-16)	Code Word	Code Length
First Extension						
α_1	a_1	2/3	0.59	1	0	1
α_2	a_2	1/3	1.58	2	1	1
Second Extension						
α_1	a_1a_1	4/9	1.17	2	0	1
α_2	a_1a_2	2/9	2.17	3	10	2
α_3	a_2a_1	2/9	2.17	3	110	3
α_4	a_2a_2	1/9	3.17	4	111	3

probability that no errors will occur during the transmission of a three-symbol message is or $(1 - p_e)^3$ or $\bar{p}_e^3$. The probability of a single error is $3p_e\bar{p}_e^2$, the probability of two errors is $3p_e^2\bar{p}_e$, and the probability of three errors is p_e^3. Because the probability of a single symbol transmission error is less than 50%, received messages can be decoded by using a majority vote of the three received symbols. Thus the probability of incorrectly decoding a three-symbol code word is the sum of the probabilities of two symbol errors and three symbol errors, or $p_e^3 + 3p_e^2\bar{p}_e$. When no errors or a single error occurs, the majority vote decodes the message correctly. For $p_e = 0.01$, the probability of a communication error is reduced to 0.0003. ■

By extending the repetitive coding scheme just described, we can make the overall error in communication as small as desired. In the general case, we do so by encoding the nth extension of the source using K-ary code sequences of length r, where $K^r \leq J^n$. The key to this approach is to select only φ of the K^r possible code sequences as valid code words and devise a decision rule that optimizes the probability of correct decoding. In the preceding example, repeating each source symbol three times is equivalent to block encoding the nonextended binary source using two out of 2^3, or 8, possible binary code words. The two valid code words are 000 and 111. If a nonvalid code word is presented to the decoder, a majority vote of the three code bits determines the output.

A zero-memory information source generates information at a *rate* (in information units per symbol) equal to its entropy $H(\mathbf{z})$. The nth extension of the source provides information at a rate of $H(\mathbf{z}')/n$ information units per symbol. If the information is coded, as in the preceding example, the maximum rate of coded information is $\log(\varphi/r)$ and occurs when the φ valid code words used to code the source are equally probable. Hence, a code of size φ and block length r is said to have a rate of

$$R = \log \frac{\varphi}{r} \tag{8.3-22}$$

information units per symbol. *Shannon's second theorem* (Shannon [1948]), also called the *noisy coding theorem*, tells us that for any $R < C$, where C is the capacity of the *zero-memory channel* with matrix $\mathbf{Q}$,[†] there exists an integer r, and code of block length r and rate R such that the probability of a block decoding error is less than or equal to ε for any $\varepsilon > 0$. Thus the probability of error can be made arbitrarily small so long as the coded message rate is less than the capacity of the channel.

The source coding theorem

The theorems described thus far establish fundamental limits on error-free communication over both reliable and unreliable channels. In this section, we turn to the case in which the channel is error free but the communication process itself is lossy. Under these circumstances, the principal function of the communication

[†] A zero-memory channel is one in which the channel's response to the current input symbol is independent of its response to previous input symbols.

system is "information compression." In most cases, the average error introduced by the compression is constrained to some maximum allowable level D. We want to determine the smallest rate, subject to a given fidelity criterion, at which information about the source can be conveyed to the user. This problem is specifically addressed by a branch of information theory known as *rate distortion theory*.

Let the information source and decoder outputs in Fig. 8.9 be defined by the finite ensembles $(A, \mathbf{z})$ and $(B, \mathbf{z})$, respectively. The assumption now is that the channel of Fig. 8.9 is error free, so a channel matrix $\mathbf{Q}$, which relates $\mathbf{z}$ to $\mathbf{v}$ in accordance with Eq. (8.3-6), can be thought of as modeling the encoding-decoding process alone. Because the encoding-decoding process is deterministic, $\mathbf{Q}$ describes an artificial zero-memory channel that models the effect of the information compression and decompression. Each time the source produces source symbol a_j, it is represented by a code symbol that is then decoded to yield output symbol b_k with probability q_{kj} (see Section 8.3.2).

Addressing the problem of encoding the source so that the average distortion is less than D requires that a rule be formulated to assign quantitatively a distortion value to every possible approximation at the source output. For the simple case of a nonextended source, a nonnegative cost function $\rho(a_j, b_k)$, called a *distortion measure*, can be used to define the penalty associated with reproducing source output a_j with decoder output b_k. The output of the source is random, so the distortion also is a random variable whose average value, denoted $d(\mathbf{Q})$, is

$$\begin{aligned} d(\mathbf{Q}) &= \sum_{j=1}^{J} \sum_{k=1}^{K} \rho(a_j, b_k) P(a_j, b_k) \\ &= \sum_{j=1}^{J} \sum_{k=1}^{K} \rho(a_j, b_k) P(a_j) q_{kj}. \end{aligned} \tag{8.3-23}$$

The notation $d(\mathbf{Q})$ emphasizes that the average distortion is a function of the encoding-decoding procedure, which (as noted previously) is modeled by $\mathbf{Q}$. A particular encoding-decoding procedure is said to be *D-admissible* if and only if the average distortion associated with $\mathbf{Q}$ is less than or equal to D. The set of all D-admissible encoding-decoding procedures therefore is

$$\mathbf{Q}_D = \{q_{kj} \mid d(\mathbf{Q}) \leq D\}. \tag{8.3-24}$$

Because every encoding-decoding procedure is defined by an artificial channel matrix $\mathbf{Q}$, the average information obtained from observing a single decoder output can be computed in accordance with Eq. (8.3-12). Hence, we can define a *rate distortion function*

$$R(D) = \min_{\mathbf{Q} \in \mathbf{Q}_D} [I(\mathbf{z}, \mathbf{v})], \tag{8.3-25}$$

which assumes the minimum value of Eq. (8.3-12) over all D-admissible codes. Note that the minimum can be taken over $\mathbf{Q}$, because $I(\mathbf{z}, \mathbf{v})$ is a function of the probabilities in vector $\mathbf{z}$ and elements in matrix $\mathbf{Q}$. If $D = 0$, $R(D)$ is less than or equal to the entropy of the source, or $R(0) \leq H(\mathbf{z})$.

Equation (8.3-25) defines the minimum rate at which information about the source can be conveyed to the user subject to the constraint that the average

distortion be less than or equal to D. To compute this rate [that is, $R(D)$], we simply minimize $I(\mathbf{z}, \mathbf{v})$ [Eq. (8.3-12)] by appropriate choice of $\mathbf{Q}$ (or q_{kj}) subject to the constraints

$$q_{kj} \geq 0, \tag{8.3-26}$$

$$\sum_{k=1}^{K} q_{kj} = 1, \tag{8.3-27}$$

and

$$d(\mathbf{Q}) = D. \tag{8.3-28}$$

Equations (8.3-26) and (8.3-27) are fundamental properties of channel matrix $\mathbf{Q}$. The elements of $\mathbf{Q}$ must be positive and, because some output must be received for any input symbol generated, the terms in any one column of $\mathbf{Q}$ must sum to 1. Equation (8.3-28) indicates that the minimum information rate occurs when the maximum possible distortion is allowed.

EXAMPLE 8.9: Computing the rate distortion function for a zero-memory binary source.

■ Consider a zero-memory binary source with equally probable source symbols $\{0, 1\}$ and the simple distortion measure

$$\rho(a_j, b_k) = 1 - \delta_{jk}$$

where δ_{jk} is the unit delta function. Because $\rho(a_j, b_k)$ is 1 if $a_j \neq b_k$ but is 0 otherwise, each encoding-decoding error is counted as one unit of distortion. The calculus of variations can be used to compute $R(D)$. Letting $\mu_1, \mu_2, \ldots, \mu_{J+1}$ be Lagrangian multipliers, we form the augmented criterion function

$$J(\mathbf{Q}) = I(\mathbf{z}, \mathbf{v}) - \sum_{j=1}^{J} \mu_j \sum_{k=1}^{K} q_{kj} - \mu_{J+1} d(\mathbf{Q}),$$

equate its JK derivatives with respect to q_{kj} to 0 (that is, $dJ/dq_{kj} = 0$), and solve the resulting equations, together with the $J + 1$ equations associated with Eqs. (8.3-27) and (8.3-28), for unknowns q_{kj} and $\mu_1, \mu_2, \ldots, \mu_{J+1}$. If the resulting q_{kj} are nonnegative [or satisfy Eq. (8.3-26)], a valid solution is found. For the source and distortion pair defined above, we get the following 7 equations (with 7 unknowns):

$$\begin{aligned}
2q_{11} &= (q_{11} + q_{12}) \exp[2\mu_1] & 2q_{22} &= (q_{21} + q_{22}) \exp[2\mu_2] \\
2q_{12} &= (q_{11} + q_{12}) \exp[2\mu_1 + \mu_3] & 2q_{21} &= (q_{21} + q_{22}) \exp[2\mu_2 + \mu_3] \\
q_{11} + q_{21} &= 1 & q_{12} + q_{22} &= 1 \\
q_{21} + q_{12} &= 2D.
\end{aligned}$$

A series of tedious but straightforward algebraic steps then yields

$$\begin{aligned}
q_{12} &= q_{21} = D \\
q_{11} &= q_{22} = 1 - D \\
\mu_1 &= \mu_2 = \log \sqrt{2(1 - D)} \\
\mu_3 &= \log \frac{D}{1 - D}
\end{aligned}$$

Substituting Eq. (8.3-14) for $P(\alpha_i)$ and simplifying yields

$$H(\mathbf{z}') = nH(\mathbf{z}). \tag{8.3-15}$$

Thus the entropy of the zero-memory information source (which produces the block random variable) is n times the entropy of the corresponding single symbol source. Such a source is referred to as the *nth extension* of the single symbol or nonextended source. Note that the first extension of any source is the nonextended source itself.

Because the self-information of source output α_i is $\log[1/P(\alpha_i)]$, it seems reasonable to code α_i with a code word of integer length $l(\alpha_i)$ such that

$$\log \frac{1}{P(\alpha_i)} \leq l(\alpha_i) < \log \frac{1}{P(\alpha_i)} + 1. \tag{8.3-16}$$

Intuition suggests that the source output α_i be represented by a code word whose length is the smallest integer exceeding the self-information of α_i.† Multiplying this result by $P(\alpha_i)$ and summing over all i gives

$$\sum_{i=1}^{J^n} P(\alpha_i) \log \frac{1}{P(\alpha_i)} \leq \sum_{i=1}^{J^n} P(\alpha_i) l(\alpha_i) < \sum_{i=1}^{J^n} P(\alpha_i) \log \frac{1}{P(\alpha_i)} + 1$$

or

$$H(\mathbf{z}') \leq L'_{\text{avg}} < H(\mathbf{z}') + 1 \tag{8.3-17}$$

where L'_{avg} represents the average word length of the code corresponding to the nth extension of the nonextended source. That is,

$$L'_{\text{avg}} = \sum_{i=1}^{J^n} P(\alpha_i) l(\alpha_i). \tag{8.3-18}$$

Dividing Eq. (8.3-17) by n and noting from Eq. (8.3-15) that $H(\mathbf{z}')/n$ is equal to $H(\mathbf{z})$ yields

$$H(\mathbf{z}) \leq \frac{L'_{\text{avg}}}{n} < H(\mathbf{z}) + \frac{1}{n} \tag{8.3-19}$$

which, in the limiting case, becomes

$$\lim_{n \to \infty} \left[\frac{L'_{\text{avg}}}{n} \right] = H(\mathbf{z}). \tag{8.3-20}$$

Equation (8.3-19) states Shannon's first theorem for a zero-memory source. It shows that it is possible to make L'_{avg}/n arbitrarily close to $H(\mathbf{z})$ by coding infinitely long extensions of the source. Although derived under the assumption of statistically independent source symbols, the result is easily extended to more general sources, where the occurrence of source symbol a_j may depend on a finite number of preceding symbols. These types of sources (called *Markov sources*) commonly are used to model interpixel correlations in an image. Because $H(\mathbf{z})$

† A uniquely decodable code can be constructed subject to this constraint.

so that

$$\mathbf{Q} = \begin{bmatrix} 1 - D & D \\ D & 1 - D \end{bmatrix}.$$

It is given that the source symbols are equally probable, so the maximum possible distortion is $\frac{1}{2}$. Thus $0 \leq D \leq \frac{1}{2}$ and the elements of $\mathbf{Q}$ satisfy Eq. (8.3-12) for all D. The mutual information associated with $\mathbf{Q}$ and the previously defined binary source is computed by using Eq. (8.3-12). Noting the similarity between $\mathbf{Q}$ and the binary symmetric channel matrix, however, we can immediately write

$$I(\mathbf{z}, \mathbf{v}) = 1 - H_{\text{bs}}(D).$$

This result follows from Example 8.6 by substituting $p_{\text{bs}} = \frac{1}{2}$ and $p_e = D$ into $I(\mathbf{z}, \mathbf{v}) = H_{\text{bs}}(p_{\text{bs}} p_e + \bar{p}_{\text{bs}} \bar{p}_e) - H_{\text{bs}}(p_e)$. The rate distortion function follows immediately from Eq. (8.3-25):

$$R(D) = \min_{\mathbf{Q} \in \mathbf{Q}_D} [1 - H_{\text{bs}}(D)] = 1 - H_{\text{bs}}(D).$$

The final simplification is based on the fact that, for a given D, $1 - H_{\text{bs}}(D)$ assumes a single value, which, by default, is the minimum. The resulting function is plotted in Fig. 8.10. Its shape is typical of most rate distortion functions. Note the maximum value of D, denoted $D_{\max}$, such that $R(D) = 0$ for all $D \geq D_{\max}$. In addition, $R(D)$ is always positive, monotonically decreasing, and convex in the interval $(0, D_{\max})$. ■

Rate distortion functions can be computed analytically for simple sources and distortion measures, as in the preceding example. Moreover, convergent iterative algorithms suitable for implementation on digital computers can be used when

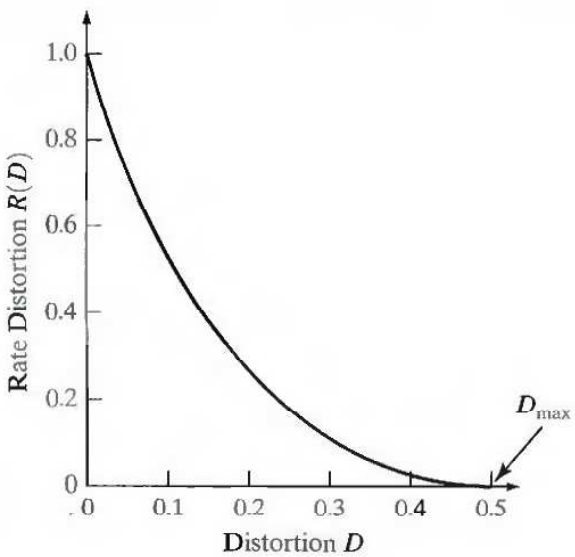


FIGURE 8.10 The rate distortion function for a binary symmetric source.

analytical methods fail or are impractical. After $R(D)$ is computed (for any zero-memory source and *single-letter* distortion measure[†]), the *source coding theorem* tells us that, for any $\varepsilon > 0$, there exists an r, and code of block length r and rate $R < R(D) + \varepsilon$, such that the average per-letter distortion satisfies the condition $d(\mathbf{Q}) \leq D + \varepsilon$. An important practical consequence of this theorem and the noisy coding theorem is that the source output can be recovered at the decoder with an arbitrarily small probability of error provided that the channel has capacity $C > R(D) + \varepsilon$. This latter result is known as the *information transmission theorem*.

8.3.4 Using Information Theory

Information theory provides the basic tools needed to deal with information representation and manipulation directly and quantitatively. In this section we explore the application of these tools to the specific problem of image compression. Because the fundamental premise of information theory is that the generation of information can be modeled as a probabilistic process, we first develop a statistical model of the image generation process.

EXAMPLE 8.10: Computing the entropy of an image.

■ Consider the problem of estimating the information content (or entropy) of the simple 8-bit image:

21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243
21	21	21	95	169	243	243	243

One relatively simple approach is to assume a particular source model and compute the entropy of the image based on that model. For example, we can assume that the image was produced by an imaginary "8-bit gray-level source" that sequentially emitted statistically independent pixels in accordance with a predefined probability law. In this case, the source symbols are gray levels, and the source alphabet is composed of 256 possible symbols. If the symbol probabilities are known, the average information content or entropy of each pixel in the image can be computed by using Eq. (8.3-3). In the case of a uniform probability density, for instance, the source symbols are equally probable, and the source is characterized by an entropy of 8 bits/pixel. That is, the average information per source output (pixel) is 8 bits. Then the total entropy of the preceding 4×8 image is 256 bits. This particular image is but one of $2^{8 \times 4 \times 8}$, or 2^{256} $(\sim 10^{77})$, equally probable 4×8 images that can be produced by the source.

An alternative method of estimating information content is to construct a source model based on the relative frequency of occurrence of the gray levels in the image under consideration. That is, an observed image can be interpreted as a sample of the behavior of the gray-level source that generated it. Because

[†] A single-letter distortion measure is one in which the distortion associated with a block of letters (or symbols) is the sum of the distortions for each letter (or symbol) in the block.

the observed image is the only available indicator of source behavior, modeling the probabilities of the source symbols using the gray-level histogram of the sample image is reasonable:

Gray Level	Count	Probability
21	12	3/8
95	4	1/8
169	4	1/8
243	12	3/8

An estimate, called the *first-order estimate*, of the entropy of the source can be computed with Eq. (8.3-3). The first-order estimate in this example is 1.81 bits/pixel. The entropy of the source and/or image thus is approximately 1.81 bits/pixel, or 58 total bits.

Better estimates of the entropy of the gray-level source that generated the sample image can be computed by examining the relative frequency of pixel blocks in the sample image, where a block is a grouping of adjacent pixels. As block size approaches infinity, the estimate approaches the source's true entropy. (This result can be shown with the procedure utilized to prove the validity of the noiseless coding theorem in Section 8.3.3.) Thus by assuming that the sample image is connected from line to line and end to beginning, we can compute the relative frequency of pairs of pixels (that is, the second extension of the source):

Gray-level Pair	Count	Probability
(21, 21)	8	1/4
(21, 95)	4	1/8
(95, 169)	4	1/8
(169, 243)	4	1/8
(243, 243)	8	1/4
(243, 21)	4	1/8

The resulting entropy estimate [again using Eq. (8.3-3)] is 2.5/2, or 1.25 bits/pixel, where division by 2 is a consequence of considering two pixels at a time. This estimate is called the *second-order estimate* of the source entropy, because it was obtained by computing the relative frequencies of 2-pixel blocks. Although the third-, fourth-, and higher-order estimates would provide even better approximations of source entropy, convergence of these estimates to the true source entropy is slow and computationally involved. For instance, a general 8-bit image has $(2^8)^2$, or 65,536, possible symbol pairs whose relative frequency must be computed. If 5-pixel blocks are considered, the number of possible 5-tuples is $(2^8)^5$, or $\sim 10^{12}$. ■

Although computing the actual entropy of an image is difficult, estimates such as those in the preceding example provide insight into image compressibility.

The first-order estimate of entropy, for example, is a lower bound on the compression that can be achieved through variable-length coding alone. (Recall from Section 8.1.1 that variable-length coding is used to reduce coding redundancies.) In addition, the differences between the higher-order estimates of entropy and the first-order estimate indicate the presence or absence of interpixel redundancies. That is, they reveal whether the pixels in an image are statistically independent. If the pixels are statistically independent (that is, there is no interpixel redundancy), the higher-order estimates are equivalent to the first-order estimate, and variable-length coding provides optimal compression. For the image considered in the preceding example, the numerical difference between the first- and second-order estimates indicates that a mapping can be created that allows an additional $1.81 - 1.25 = 0.56$ bits/pixel to be eliminated from the image's representation.

EXAMPLE 8.11: Using mappings to reduce entropy.

■ Consider mapping the pixels of the image in the preceding example to create the representation:

21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0
21	0	0	74	74	74	0	0

Here, we construct a difference array by replicating the first column of the original image and using the arithmetic difference between adjacent columns for the remaining elements. For example, the element in the first row, second column of the new representation is $(21 - 21)$, or 0. The resulting difference distribution is

Gray Level or Difference	Count	Probability
0	12	1/2
21	4	1/8
74	12	3/8

If we now consider the mapped array to be generated by a "difference source," we can again use Eq. (8.3-3) to compute a first-order estimate of the entropy of the array, which is 1.41 bits/pixel. Thus by variable-length coding the mapped difference image, the original image can be represented with only 1.41 bits/pixel or a total of about 46 bits. This value is greater than the 1.25 bits/pixel second-order estimate of entropy computed in the preceding example, so we know that we can find an even better mapping. ■

The preceding examples illustrate that the first-order estimate of the entropy of an image is not necessarily the minimum code rate for the image. The reason is that pixels in an image generally are not statistically independent. As noted

in Section 8.2, the process of minimizing the actual entropy of an image is called source coding. In the error-free case it encompasses the two operations of mapping and symbol coding. If information loss can be tolerated, it also includes the third step of quantization.

The slightly more complicated problem of lossy image compression can also be approached using the tools of information theory. In this case, however, the principal result is the source coding theorem. As indicated in Section 8.3.3, this theorem reveals that any zero-memory source can be encoded by using a code of rate $R < R(D)$ such that the average per symbol distortion is less than D. To apply this result correctly to lossy image compression requires identifying an appropriate source model, devising a meaningful distortion measure, and computing the resulting rate distortion function $R(D)$. The first step of this process has already been considered. The second step can be conveniently approached through the use of an objective fidelity criterion from Section 8.1.4. The final step involves finding a matrix $\mathbf{Q}$ whose elements minimize Eq. (8.3-12), subject to the constraints imposed by Eqs. (8.3-24) through (8.3-28). Unfortunately, this task is particularly difficult—and only a few cases of any practical interest have been solved. One is when the images are Gaussian random fields and the distortion measure is a weighted square error function. In this case, the optimal encoder must expand the image into its Karhunen-Loève components (see Section 11.4) and represent each component with equal mean-square error (Davisson [1972]).

8.4 Error-Free Compression

In numerous applications error-free compression is the only acceptable means of data reduction. One such application is the archival of medical or business documents, where lossy compression usually is prohibited for legal reasons. Another is the processing of satellite imagery, where both the use and cost of collecting the data makes any loss undesirable. Yet another is digital radiography, where the loss of information can compromise diagnostic accuracy. In these and other cases, the need for error-free compression is motivated by the intended use or nature of the images under consideration.

In this section, we focus on the principal error-free compression strategies currently in use. They normally provide compression ratios of 2 to 10. Moreover, they are equally applicable to both binary and gray-scale images. As indicated in Section 8.2, error-free compression techniques generally are composed of two relatively independent operations: (1) devising an alternative representation of the image in which its interpixel redundancies are reduced; and (2) coding the representation to eliminate coding redundancies. These steps correspond to the mapping and symbol coding operations of the source coding model discussed in connection with Fig. 8.6.

8.4.1 Variable-Length Coding

The simplest approach to error-free image compression is to reduce *only* coding redundancy. Coding redundancy normally is present in any natural binary encoding of the gray levels in an image. As we noted in Section 8.1.1, it can be

eliminated by coding the gray levels so that Eq. (8.1-4) is minimized. To do so requires construction of a variable-length code that assigns the shortest possible code words to the most probable gray levels. Here, we examine several optimal and near optimal techniques for constructing such a code. These techniques are formulated in the language of information theory. In practice, the source symbols may be either the gray levels of an image or the output of a gray-level mapping operation (pixel differences, run lengths, and so on).

Huffman coding

The most popular technique for removing coding redundancy is due to Huffman (Huffman [1952]). When coding the symbols of an information source individually, *Huffman coding* yields the smallest possible number of code symbols per source symbol. In terms of the noiseless coding theorem (see Section 8.3.3), the resulting code is optimal for a fixed value of n, subject to the constraint that the source symbols be coded *one at a time*.

The first step in Huffman's approach is to create a series of source reductions by ordering the probabilities of the symbols under consideration and combining the lowest probability symbols into a single symbol that replaces them in the next source reduction. Figure 8.11 illustrates this process for binary coding (K-ary Huffman codes can also be constructed). At the far left, a hypothetical set of source symbols and their probabilities are ordered from top to bottom in terms of decreasing probability values. To form the first source reduction, the bottom two probabilities, 0.06 and 0.04, are combined to form a "compound symbol" with probability 0.1. This compound symbol and its associated probability are placed in the first source reduction column so that the probabilities of the reduced source are also ordered from the most to the least probable. This process is then repeated until a reduced source with two symbols (at the far right) is reached.

The second step in Huffman's procedure is to code each reduced source, starting with the smallest source and working back to the original source. The minimal length binary code for a two-symbol source, of course, is the symbols 0 and 1. As Fig. 8.12 shows, these symbols are assigned to the two symbols on the right (the assignment is arbitrary; reversing the order of the 0 and 1 would work just as well). As the reduced source symbol with probability 0.6 was generated by combining two symbols in the reduced source to its left, the 0 used to code it is now assigned to *both* of these symbols, and a 0 and 1 are arbitrarily

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

FIGURE 8.11 Huffman source reductions.

FIGURE 8.12 Huffman code assignment procedure.

Original source			Source reduction			
Sym.	Prob.	Code	1	2	3	4
a_2	0.4	1	0.4 1	0.4 1	0.4 1	0.6 0
a_6	0.3	00	0.3 00	0.3 00	0.3 00	0.4 1
a_1	0.1	011	0.1 011	0.2 010	0.3 01	
a_4	0.1	0100	0.1 0100	0.1 011		
a_3	0.06	01010	0.1 0101			
a_5	0.04	01011				

appended to each to distinguish them from each other. This operation is then repeated for each reduced source until the original source is reached. The final code appears at the far left in Fig. 8.12. The average length of this code is

$$L_{avg} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5)$$
$$= 2.2 \text{ bits/symbol}$$

and the entropy of the source is 2.14 bits/symbol. In accordance with Eq. (8.3-21), the resulting Huffman code efficiency is 0.973.

Huffman's procedure creates the optimal code for a set of symbols and probabilities *subject to* the constraint that the symbols be coded one at a time. After the code has been created, coding and/or decoding is accomplished in a simple lookup table manner. The code itself is an instantaneous uniquely decodable block code. It is called a *block code* because each source symbol is mapped into a fixed sequence of code symbols. It is *instantaneous*, because each code word in a string of code symbols can be decoded without referencing succeeding symbols. It is *uniquely decodable*, because any string of code symbols can be decoded in only one way. Thus, any string of Huffman encoded symbols can be decoded by examining the individual symbols of the string in a left to right manner. For the binary code of Fig. 8.12, a left-to-right scan of the encoded string 010100111100 reveals that the first valid code word is 01010, which is the code for symbol a_3. The next valid code is 011, which corresponds to symbol a_1. Continuing in this manner reveals the completely decoded message to be $a_3 a_1 a_2 a_2 a_6$.

Other near optimal variable length codes

When a large number of symbols is to be coded, the construction of the optimal binary Huffman code is a nontrivial task. For the general case of J source symbols, $J - 2$ source reductions must be performed (see Fig. 8.11) and $J - 2$ code assignments made (see Fig. 8.12). Thus construction of the optimal Huffman code for an image with 256 gray levels requires 254 source reductions and 254 code assignments. In view of the computational complexity of this task, sacrificing coding efficiency for simplicity in code construction sometimes is necessary.

Table 8.5 illustrates four variable-length codes that provide such a trade-off. Note that the average length of the Huffman code—the last row of the table—is lower than the other codes listed. The natural binary code has the greatest average length. In addition, the 4.05 bits/symbol code rate achieved by Huffman's technique approaches the 4.0 bits/symbol entropy bound of the source, com-

TABLE 8.5 Variable-length codes.

Source symbol	Probability	Binary Code	Huffman	Truncated Huffman	B_2-Code	Binary Shift	Huffman Shift
Block 1							
a_1	0.2	00000	10	11	C00	000	10
a_2	0.1	00001	110	011	C01	001	11
a_3	0.1	00010	111	0000	C10	010	110
a_4	0.06	00011	0101	0101	C11	011	100
a_5	0.05	00100	00000	00010	C00C00	100	101
a_6	0.05	00101	00001	00011	C00C01	101	1110
a_7	0.05	00110	00010	00100	C00C10	110	1111
Block 2							
a_8	0.04	00111	00011	00101	C00C11	111 000	00 10
a_9	0.04	01000	00110	00110	C01C00	111 001	00 11
a_{10}	0.04	01001	00111	00111	C01C01	111 010	00 110
a_{11}	0.04	01010	00100	01000	C01C10	111 011	00 100
a_{12}	0.03	01011	01001	01001	C01C11	111 100	00 101
a_{13}	0.03	01100	01110	100000	C10C00	111 101	00 1110
a_{14}	0.03	01101	01111	100001	C10C01	111 110	00 1111
Block 3							
a_{15}	0.03	01110	01100	100010	C10C10	111 111 000	000 10
a_{16}	0.02	01111	010000	100011	C10C11	111 111 001	000 11
a_{17}	0.02	10000	010001	100100	C11C00	111 111 010	000 110
a_{18}	0.02	10001	001010	100101	C11C01	111 111 011	000 100
a_{19}	0.02	10010	001011	100110	C11C10	111 111 100	000 101
a_{20}	0.02	10011	011010	100111	C11C11	111 111 101	000 1110
a_{21}	0.01	10100	011011	101000	C00C00C00	111 111 110	000 1111
Entropy	4.0						
Average length		5.0	4.05	4.24	4.65	4.59	4.13

puted by using Eq. (8.3-3) and given at the bottom of the table. Although none of the remaining codes in Table 8.5 achieve the Huffman coding efficiency, all are easier to construct. Like Huffman's technique, they assign the shortest code words to the most likely source symbols.

Column 5 of Table 8.5 illustrates a simple modification of the basic Huffman coding strategy known as *truncated Huffman coding*. A truncated Huffman code is generated by Huffman coding only the most probable ψ symbols of the source, for some positive integer ψ less than J. A prefix code followed by a suitable fixed-length code is used to represent all other source symbols. In Table 8.5, ψ arbitrarily was selected as 12 and the prefix code was generated as the 13th Huffman code word. That is, a "prefix symbol" whose probability was the sum of the probabilities of symbols a_{13} through a_{21} was included as a 13th symbol during the Huffman coding of the 12 most probable source symbols. The remaining 9 symbols were then coded using the prefix code, which turned out to be 10, and a 4-bit binary value equal to the symbol subscript minus 13.

Column 6 of Table 8.5 illustrates a second, near optimal, and variable-length code known as a B-code. It is close to optimal when the source symbol probabilities obey a power law of the form

$$P(a_j) = cj^{-\beta} \tag{8.4-1}$$

for some positive constant β and normalizing constant $c = 1/\sum_{j=0}^{J} j^{-\beta}$. For example, the distribution of run lengths in a binary representation of a typical typewritten text document is nearly exponential. As Table 8.5 shows, each code word is made up of *continuation* bits, denoted C, and *information* bits, which are natural binary numbers. The only purpose of the continuation bits is to separate individual code words, so they simply alternate between 0 and 1 for each code word in a string. The B-code shown in Table 8.5 is called a B_2-code, because two information bits are used per continuation bit. The sequence of B_2-codes corresponding to the source symbol string $a_{11}a_2a_7$ is 001 010 101 000 010 or 101 110 001 100 110, depending on whether the first continuation bit is assumed to be 0 or 1.

The two remaining variable-length codes in Table 8.5 are referred to as *shift codes*. A shift code is generated by (1) arranging the source symbols so that their probabilities are monotonically decreasing, (2) dividing the total number of symbols into symbol blocks of equal size, (3) coding the individual elements within all blocks identically, and (4) adding special *shift-up* and/or *shift-down* symbols to identify each block. Each time a shift-up or shift-down symbol is recognized at the decoder, it moves one block up or down with respect to a predefined reference block.

To generate the 3-bit binary shift code in column 7 of Table 8.5, the 21 source symbols are first ordered in accordance with their probabilities of occurrence and divided into three blocks of seven symbols. The individual symbols (a_1 through a_7) of the upper block—considered the reference block—are then coded with the binary codes 000 through 110. The eighth binary code (111) is not included in the reference block; instead, it is used as a single shift-up control that identifies the remaining blocks (in this case, a shift-down symbol is not used). The symbols in the remaining two blocks are then coded by one or two shift-up symbols in combination with the binary codes used to code the reference block. For example, source symbol a_{19} is coded as 111 111 100.

The Huffman shift code in column 8 of Table 8.5 is generated in a similar manner. The principal difference is in the assignment of a probability to the shift symbol prior to Huffman coding the reference block. Normally, this assignment is accomplished by summing the probabilities of all the source symbols outside the reference block; that is, by using the same concept utilized to define the prefix symbol in the truncated Huffman code. Here, the sum is taken over symbols a_8 through a_{21} and is 0.39. The shift symbol is thus the most probable symbol and is assigned one of the shortest Huffman code words (00).

Arithmetic coding

Unlike the variable-length codes described previously, *arithmetic coding* generates nonblock codes. In arithmetic coding, which can be traced to the work of Elias (see Abramson [1963]), a one-to-one correspondence between source symbols and code words does not exist. Instead, an entire sequence of source symbols (or message) is assigned a single arithmetic code word. The code word itself defines an interval of real numbers between 0 and 1. As the number of symbols in the message increases, the interval used to represent it becomes smaller

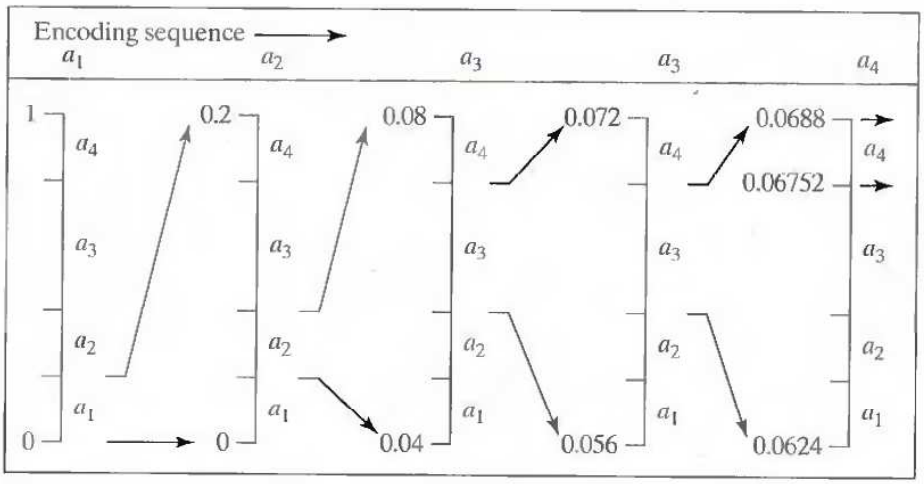


FIGURE 8.13 Arithmetic coding procedure.

and the number of information units (say, bits) required to represent the interval becomes larger. Each symbol of the message reduces the size of the interval in accordance with its probability of occurrence. Because the technique does not require, as does Huffman's approach, that each source symbol translate into an integral number of code symbols (that is, that the symbols be coded one at a time), it achieves (but only in theory) the bound cstablished by the noiseless coding theorem of Section 8.3.3.

Figure 8.13 illustrates the basic arithmetic coding process. Here, a five-symbol sequence or message, $a_1a_2a_3a_3a_4$, from a four-symbol source is coded. At the start of the coding process, the message is assumed to occupy the entire half-open interval $[0, 1)$. As Table 8.6 shows, this interval is initially subdivided into four regions based on the probabilities of each source symbol. Symbol a_1, for example, is associated with subinterval $[0, 0.2)$. Because it is the first symbol of the message being coded, the message interval is initially narrowed to $[0, 0.2)$. Thus in Fig. 8.13 $[0, 0.2)$ is expanded to the full height of the figure and its end points labeled by the values of the narrowed range. The narrowed range is then subdivided in accordance with the original source symbol probabilities and the process continues with the next message symbol. In this manner, symbol a_2 narrows the subinterval to $[0.04, 0.08)$, a_3 further narrows it to $[0.056, 0.072)$, and so on. The final message symbol, which must be reserved as a special end-of-message indicator, narrows the range to $[0.06752, 0.0688)$. Of course, any number within this subinterval—for example, 0.068—can be used to represent the message.

Source Symbol	Probability	Initial Subinterval
a_1	0.2	$[0.0, 0.2)$
a_2	0.2	$[0.2, 0.4)$
a_3	0.4	$[0.4, 0.8)$
a_4	0.2	$[0.8, 1.0)$

TABLE 8.6 Arithmetic coding example.

In the arithmetically coded message of Fig. 8.13, three decimal digits are used to represent the five-symbol message. This translates into $^3/_5$ or 0.6 decimal digits per source symbol and compares favorably with the entropy of the source, which, from Eq. (8.3-3), is 0.58 decimal digits or 10-ary units/symbol. As the length of the sequence being coded increases, the resulting arithmetic code approaches the bound established by the noiseless coding theorem. In practice, two factors cause coding performance to fall short of the bound: (1) the addition of the end-of-message indicator that is needed to separate one message from another; and (2) the use of finite precision arithmetic. Practical implementations of arithmetic coding address the latter problem by introducing a scaling strategy and a rounding strategy (Langdon and Rissanen [1981]). The scaling strategy renormalizes each subinterval to the $[0, 1)$ range before subdividing it in accordance with the symbol probabilities. The rounding strategy guarantees that the truncations associated with finite precision arithmetic do not prevent the coding subintervals from being represented accurately.

8.4.2 LZW Coding

Having examined the principal methods for removing coding redundancy, we now consider one of several error-free compression techniques that also attack an image's interpixel redundancies. The technique, called Lempel-Ziv-Welch (LZW) coding, assigns fixed-length code words to variable length sequences of source symbols but requires no a priori knowledge of the probability of occurrence of the symbols to be encoded. Recall from Section 8.3.3 that Shannon's first theorem states that the nth extension of a zero-memory source can be coded with fewer average bits per source symbol than the nonextended source itself. Despite the fact that it must be licensed under United States Patent No. 4,558,302, LZW compression has been integrated into a variety of mainstream imaging file formats, including the *graphic interchange format* (GIF), *tagged image file format* (TIFF), and the *portable document format* (PDF).

LZW coding is conceptually very simple (Welch [1984]). At the onset of the coding process, a codebook or "dictionary" containing the source symbols to be coded is constructed. For 8-bit monochrome images, the first 256 words of the dictionary are assigned to the gray values $0, 1, 2, \ldots, 255$. As the encoder sequentially examines the image's pixels, gray-level sequences that are not in the dictionary are placed in algorithmically determined (e.g., the next unused) locations. If the first two pixels of the image are white, for instance, sequence "255-255" might be assigned to location 256, the address following the locations reserved for gray levels 0 through 255. The next time that two consecutive white pixels are encountered, code word 256, the address of the location containing sequence 255-255, is used to represent them. If a 9-bit, 512-word dictionary is employed in the coding process, the original $(8 + 8)$ bits that were used to represent the two pixels are replaced by a single 9-bit code word. Cleary, the size of the dictionary is an important system parameter. If it is too small, the detection of matching gray-level sequences will be less likely; if it is too large, the size of the code words will adversely affect compression performance.

EXAMPLE 8.12: An LZW coding example.

■ Consider the following 4×4, 8-bit image of a vertical edge:

39	39	126	126
39	39	126	126
39	39	126	126
39	39	126	126

Table 8.7 details the steps involved in coding its 16 pixels. A 512-word dictionary with the following starting content is assumed:

Dictionary Location	Entry
0	0
1	1
⋮	⋮
255	255
256	—
⋮	⋮
511	—

Locations 256 through 511 are initially unused.

The image is encoded by processing its pixels in a left-to-right, top-to-bottom manner. Each successive gray-level value is concatenated with a variable—column 1 of Table 8.7—called the "currently recognized sequence." As can be seen, this variable is initially null or empty. The dictionary is searched for each

TABLE 8.7 LZW coding example.

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39-39
39	126	39	257	39-126
126	126	126	258	126-126
126	39	126	259	126-39
39	39			
39-39	126	256	260	39-39-126
126	126			
126-126	39	258	261	126-126-39
39	39			
39-39	126			
39-39-126	126	260	262	39-39-126-126
126	39			
126-39	39	259	263	126-39-39
39	126			
39-126	126	257	264	39-126-126
126		126		

concatenated sequence and if found, as was the case in the first row of the table, is replaced by the newly concatenated and recognized (i.e., located in the dictionary) sequence. This was done in column 1 of row 2. No output codes are generated, nor is the dictionary altered. If the concatenated sequence is not found, however, the address of the currently recognized sequence is output as the next encoded value, the concatenated but unrecognized sequence is added to the dictionary, and the currently recognized sequence is initialized to the current pixel value. This occurred in row 2 of the table. The last two columns detail the gray-level sequences that are added to the dictionary when scanning the entire 4×4 image. Nine additional code words are defined. At the conclusion of coding, the dictionary contains 265 code words and the LZW algorithm has successfully identified several repeating gray-level sequences—leveraging them to reduce the original 128-bit image to 90 bits (i.e., 10 9-bit codes). The encoded output is obtained by reading the third column from top to bottom. The resulting compression ratio is 1.42:1. ■

A unique feature of the LZW coding just demonstrated is that the coding dictionary or code book is created while the data are being encoded. Remarkably, an LZW decoder builds an identical decompression dictionary as it decodes simultaneously the encoded data stream. It is left as an exercise to the reader (see Problem 8.16) to decode the output of the preceding example and reconstruct the code book. Although not needed in this example, most practical applications require a strategy for handling dictionary overflow. A simple solution is to flush or reinitialize the dictionary when it becomes full and continue coding with a new initialized dictionary. A more complex option is to monitor compression performance and flush the dictionary when it becomes poor or unacceptable. Alternately, the least used dictionary entries can be tracked and replaced when necessary.

8.4.3 Bit-Plane Coding

Another effective technique for reducing an image's interpixel redundancies is to process the image's bit planes individually. The technique, called *bit-plane coding*, is based on the concept of decomposing a multilevel (monochrome or color) image into a series of binary images and compressing each binary image via one of several well-known binary compression methods. In this section, we describe the most popular decomposition approaches and review several of the more commonly used compression methods.

Bit-plane decomposition

The gray levels of an m-bit gray-scale image can be represented in the form of the base 2 polynomial

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \cdots + a_1 2^1 + a_0 2^0. \tag{8.4-2}$$

Based on this property, a simple method of decomposing the image into a collection of binary images is to separate the m coefficients of the polynomial into m 1-bit *bit planes*. As noted in Chapter 3, the zeroth-order bit plane is generated

by collecting the a_0 bits of each pixel, while the $(m-1)$st-order bit plane contains the a_{m-1} bits or coefficients. In general, each bit plane is numbered from 0 to $m-1$ and is constructed by setting its pixels equal to the values of the appropriate bits or polynomial coefficients from each pixel in the original image. The inherent disadvantage of this approach is that small changes in gray level can have a significant impact on the complexity of the bit planes. If a pixel of intensity 127 (01111111) is adjacent to a pixel of intensity 128 (10000000), for instance, every bit plane will contain a corresponding 0 to 1 (or 1 to 0) transition. For example, as the most significant bits of the two binary codes for 127 and 128 are different, bit plane 7 will contain a zero-valued pixel next to a pixel of value 1, creating a 0 to 1 (or 1 to 0) transition at that point.

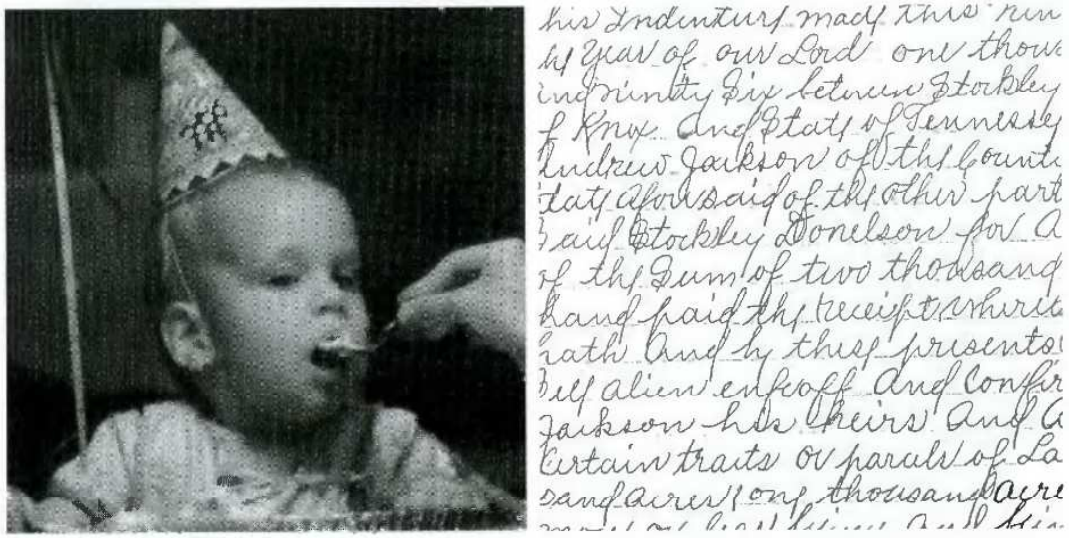
An alternative decomposition approach (which reduces the effect of small gray-level variations) is to first represent the image by an m-bit *Gray code*. The m-bit Gray code $g_{m-1} \cdots g_2 g_1 g_0$ that corresponds to the polynomial in Eq. (8.4-2) can be computed from

$$g_i = a_i \oplus a_{i+1} \qquad 0 \le i \le m-2 \tag{8.4-3}$$
$$g_{m-1} = a_{m-1}.$$

Here, $\oplus$ denotes the exclusive OR operation. This code has the unique property that successive code words differ in only one bit position. Thus, small changes in gray level are less likely to affect all m bit planes. For instance, when gray levels 127 and 128 are adjacent, only the 7th bit plane will contain a 0 to 1 transition, because the Gray codes that correspond to 127 and 128 are 11000000 and 01000000, respectively.

EXAMPLE 8.13: Bit-plane coding.

■ The 1024×1024 images shown in Figs. 8.14(a) and (b) are used to illustrate the compression techniques described in the remainder of this section. The 8-bit monochrome image of a child was generated with a high-resolution CCD camera. The binary image of a warranty deed prepared by President Andrew Jackson in 1796 was produced on a flatbed document scanner. Figures 8.15 and 8.16 show the eight binary and Gray-coded bit planes of the image of the child.



a b

FIGURE 8.14 A 1024×1024 (a) 8-bit monochrome image and (b) binary image.

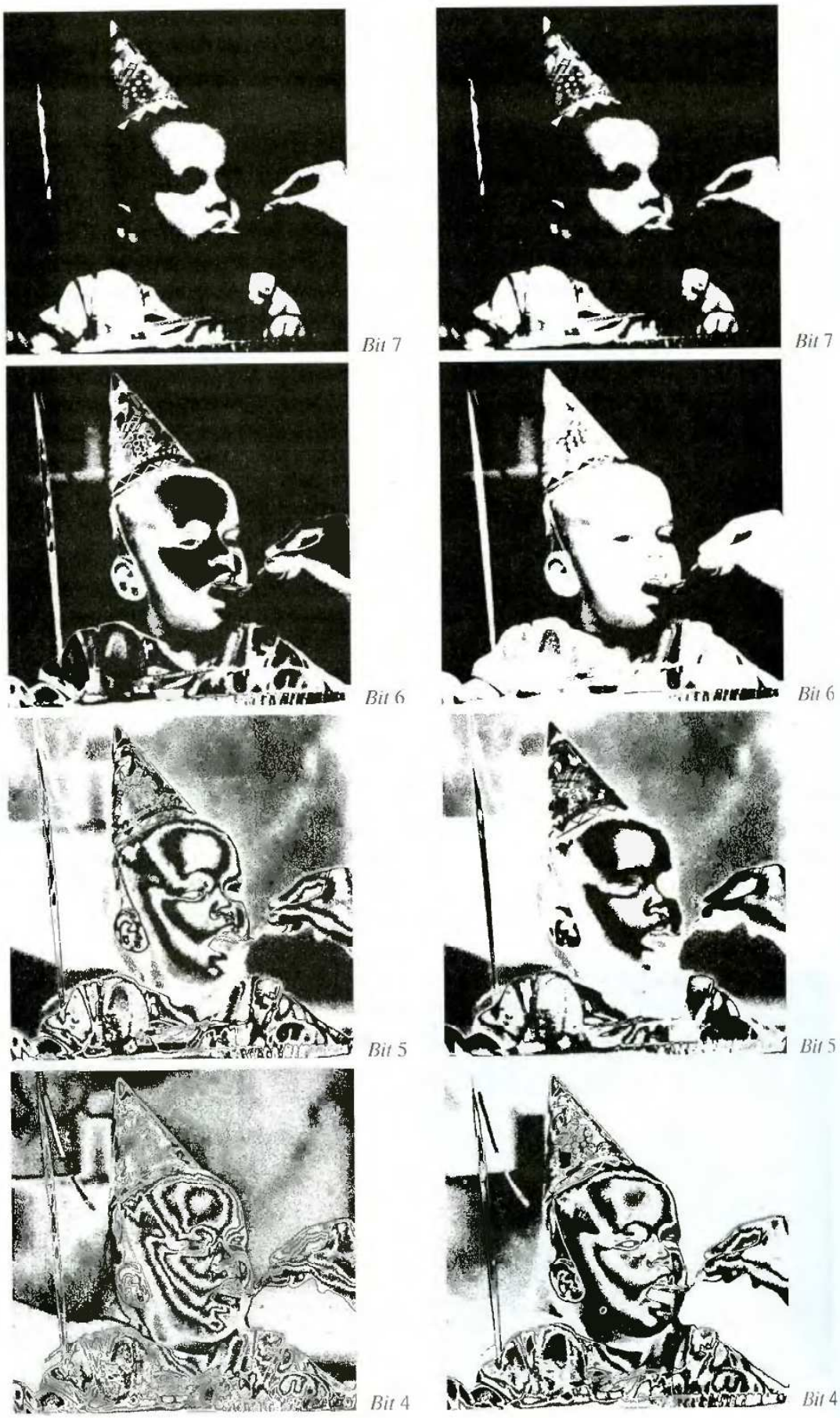


FIGURE 8.15 The four most significant binary (left column) and Gray-coded (right column) bit planes of the image in Fig. 8.14(a).

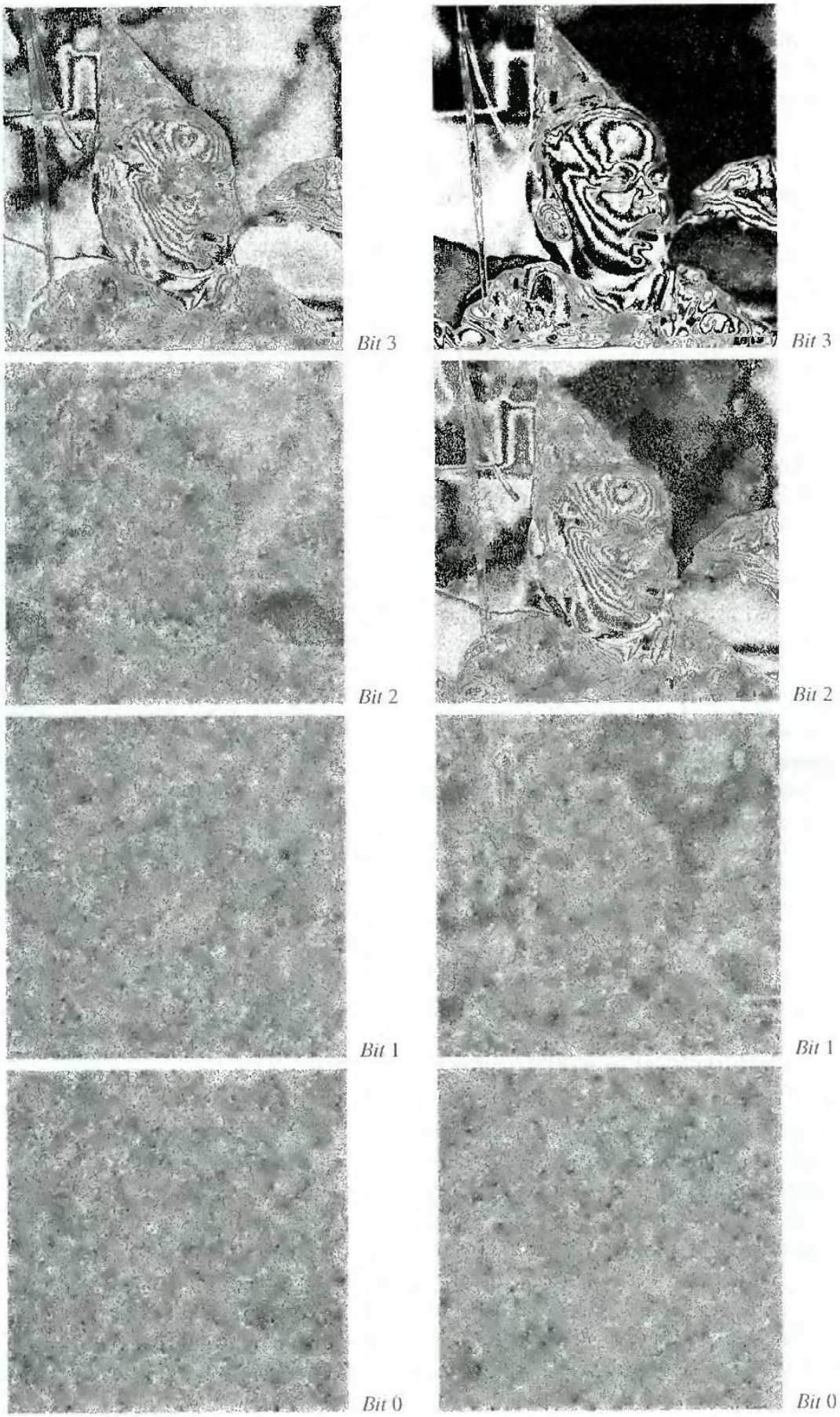


FIGURE 8.16 The four least significant binary (left column) and Gray-coded (right column) bit planes of the image in Fig. 8.14(a).

Note that the high-order bit planes are far less complex than their low-order counterparts. That is, they contain large uniform areas of significantly less detail, busyness, or randomness. In addition, the Gray-coded bit planes are less complex than the corresponding binary bit planes. ■

Constant area coding

A simple but effective method of compressing a binary image or bit plane is to use special code words to identify large areas of contiguous 1's or 0's. In one such approach, called *constant area coding* (CAC), the image is divided into blocks of size $p \times q$ pixels, which are classified as all white, all black, or mixed intensity. The most probable or frequently occurring category is then assigned the 1-bit code word 0, and the other two categories are assigned the 2-bit codes 10 and 11. Compression is achieved because the pq bits that normally would be used to represent each constant area are replaced by a 1-bit or 2-bit code word. Of course, the code assigned to the mixed intensity category is used as a prefix, which is followed by the pq-bit pattern of the block.

When predominantly white text documents are being compressed, a slightly simpler approach is to code the solid white areas as 0 and all other blocks (including the solid black blocks) by a 1 followed by the bit pattern of the block. This approach, called *white block skipping* (WBS), takes advantage of the anticipated structural tendencies of the image to be compressed. As few solid black areas are expected, they are grouped with the mixed intensity regions, allowing a 1-bit code word to be used for the highly probable white blocks. A particularly effective modification of this procedure (with blocks of size $1 \times q$) is to code the solid white lines as 0's and all other lines with a 1 followed by the normal WBS code sequence. Another is to employ an iterative approach in which the binary image or bit plane is decomposed into successively smaller and smaller subblocks. For 2-D blocks, a solid white image is coded as a 0, and all other images are divided into subblocks that are assigned a prefix of 1 and similarly coded. That is, if a subblock is all white, it is represented by the prefix 1, indicating that it is a first iteration subblock, followed by a 0, indicating that it is solid white. If the subblock is not solid white, the decomposition process is repeated until a predefined subblock size is reached and coded as either a 0 (if it is all white) or a 1 followed by the block bit pattern.

One-dimensional run-length coding

An effective alternative to constant area coding is to represent each row of an image or bit plane by a sequence of lengths that describe successive runs of black and white pixels. This technique, referred to as *run-length coding*, was developed in the 1950s and has become, along with its 2-D extensions, the standard compression approach in facsimile (FAX) coding. The basic concept is to code each contiguous group of 0's or 1's encountered in a left to right scan of a row by its length and to establish a convention for determining the value of the run. The most common approaches for determining the value of a run are (1) to specify the value of the first run of each row, or (2) to assume that each row begins with a white run, whose run length may in fact be zero.

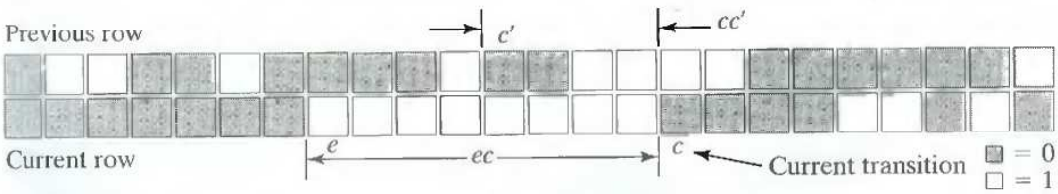
Although run-length coding is in itself an effective method of compressing an image (see the example in Section 8.1.2), additional compression usually can be realized by variable-length coding the run lengths themselves. In fact, the black and white run lengths may be coded separately using variable-length codes that are specifically tailored to their own statistics. For example, letting symbol a_j represent a black run of length j, we can estimate the probability that symbol a_j was emitted by an imaginary black run-length source by dividing the number of black run lengths of length j in the entire image by the total number of black runs. An estimate of the entropy of this black run-length source, denoted H_0, follows by substituting these probabilities into Eq. (8.3-3). A similar argument holds for the entropy of the white runs, denoted H_1. The approximate run-length entropy of the image is

$$H_{RL} = \frac{H_0 + H_1}{L_0 + L_1} \tag{8.4-4}$$

where the variables L_0 and L_1 denote the average values of black and white run lengths, respectively. Equation (8.4-4) provides an estimate of the average number of bits per pixel required to code the run lengths in a binary image using a variable-length code.

Two-dimensional run-length coding

One-dimensional run-length coding concepts are easily extended to create a variety of 2-D coding procedures. One of the better known results is *relative address coding* (RAC), which is based on the principle of tracking the binary transitions that begin and end each black and white run. Figure 8.17(a) illustrates one implementation of this approach. Note that ec is the distance from the current transition c to the last transition of the current line e, whereas cc' is the distance from c to the first similar (in the same direction) transition past e, denoted c', on the previous line. If $ec \leq cc'$, the RAC coded distance d is set equal to ec and used to represent the current transition at c. But if $cc' < ec$, d is set equal to cc'.



a
b

FIGURE 8.17 A relative address coding (RAC) illustration.

Distance measured	Distance	Code	Distance range	Code $h(d)$
cc'	0	0	1 – 4	0 ××
ec or cc' (left)	1	100	5 – 20	10 ××××
cc' (right)	1	101	21 – 84	110 ××××××
ec	$d(d>1)$	111 $h(d)$	85 – 340	1110 ××××××××
cc' (c' to left)	$d(d>1)$	1100 $h(d)$	341 – 364	11110 ×××××××××××
cc' (c' to right)	$d(d>1)$	1101 $h(d)$	1365 5460	111110 ××××××××××××

Like run-length coding, relative address coding requires the adoption of a convention for determining run values. In addition, imaginary transitions at the start and end of each line, as well as an imaginary starting line (say, an all-white line), must be assumed so that image boundaries can be handled properly. Finally, since the probability distributions of the RAC distances of most images are not uniform in practice (see Section 8.1.1), the final step of the RAC process is to code the RAC distance selected (that is, the shortest) and its distance, d, by using a suitable variable-length code. As Fig. 8.17(b) shows, a code similar to a B_1-code can be utilized. The smallest distances are assigned the shortest code words, and all other distances are coded by using a prefix to indicate the shortest RAC distance, a second prefix that assigns d to a specific range of distances, and the binary representation [denoted $\times\times\times \ldots \times$ in Fig. 8.17(b)] of d minus the base distance of the range itself. If ec and cc' are +8 and +4, as in Fig. 8.17(a), the proper RAC code word is 1100011. Finally, if $d = 0$, c is directly below c', whereas if $d = 1$, the decoder may have to determine the closest transition point, because the 100 code does not specify whether the measurement is relative to the current row or to the previous row.

Contour tracing and coding

Relative address coding is one approach for representing intensity transitions that make up the contours in a binary image. Another approach is to represent each contour by a set of boundary points or by a single boundary point and a set of directionals. The latter sometimes is referred to as *direct contour tracing*. In this section, we describe yet another method, called *predictive differential quantizing* (PDQ), which demonstrates the essential characteristics of both approaches. It is a scan-line-oriented contour tracing procedure.

In predictive differential quantizing, the front and back contours (Fig. 8.18) of each object of an image are traced simultaneously to generate a sequence of pairs (Δ', Δ''). The term Δ' is the difference between the starting coordinates of the front contours on adjacent lines, and Δ'' is the difference between the front-to-back contour lengths. These differences, together with special messages that indicate the start of new contours (the *new start* message) and the end of old contours (the *merge* message), represent each object. If Δ'' is replaced by the

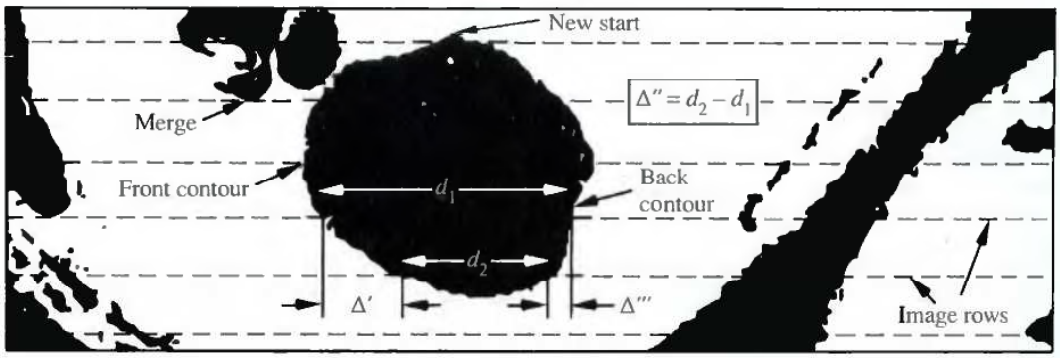


FIGURE 8.18 Parameters of the PDQ algorithm.

difference between the back contour coordinates of adjacent lines, denoted Δ''', the technique is referred to as *double delta coding* (DDC).

The new start and merge messages allow the (Δ', Δ'') or (Δ', Δ''') pairs generated on a scan line basis to be linked properly to the corresponding pairs in the previous and next rows. Without these messages, the decoder would be unable to reference one difference pair to another or to position correctly the contours within the image. To avoid encoding both the row and column coordinates of each new start and merge message, a unique code often is used to identify scan lines that do not contain object pixels. The final step in both PDQ and DDC coding is to represent Δ', Δ'' or Δ''', and the coordinates of the new starts and merges with a suitable variable-length code.

EXAMPLE 8.14: Comparison of binary compression techniques.

We conclude this section by comparing the binary compression techniques described previously. Each approach was used to compress the images of Fig. 8.14. The resulting code rates and compression ratios are provided in Tables 8.8 and 8.9. When interpreting these results, note that first-order estimates (see Section 8.3.4) of the entropies of the RLC run lengths and PDQ and DDC distances were computed and used as an approximation of the compression performance that could be achieved under the variable-length coding approaches of Section 8.4.1.

The results in Tables 8.8 and 8.9 show that all the techniques were able to eliminate some amount of interpixel redundancy. That is, the resulting code rates

TABLE 8.8 Error-free bit-plane coding results for Fig. 8.14(a): $H \approx 6.82$ bits/pixel

	Bit-plane code rate (bits/pixel)									
Method	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Code Rate**	**Compression Ratio**
Binary Bit Plane Coding										
CBC (4 × 4)	0.14	0.24	0.60	0.79	0.99	—	—	—	5.75	1.4:1
RLC	0.09	0.19	0.51	0.68	0.87	1.00	1.00	1.00	5.33	1.5:1
PDQ	0.07	0.18	0.79	—	—	—	—	—	6.04	1.3:1
DDC	0.07	0.18	0.79	—	—	—	—	—	6.03	1.3:1
RAC	0.06	0.15	0.62	0.91	—	—	—	—	5.17	1.4:1
Gray Bit Plane Coding										
CBC (4 × 4)	0.14	0.18	0.48	0.40	0.61	0.98	—	—	4.80	1.7:1
RLC	0.09	0.13	0.40	0.33	0.51	0.85	1.00	1.00	4.29	1.9:1
PDQ	0.07	0.12	0.61	0.40	0.82	—	—	—	5.02	1.6:1
DDC	0.07	0.11	0.61	0.40	0.81	—	—	—	5.00	1.6:1
RAC	0.06	0.10	0.49	0.31	0.62	—	—	—	4.05	1.8:1

TABLE 8.9 Error-free binary image compression results for Fig. 8.14(b): $H \approx 0.55$ bits/pixel.

	WBS (1 × 8)	**WBS (4 × 4)**	**RLC**	**PDQ**	**DDC**	**RAC**
Code rate (bits/pixel)	0.48	0.39	0.32	0.23	0.22	0.23
Compression ratio	2.1:1	2.6:1	3.1:1	4.4:1	4.7:1	4.4:1

were less than the first-order entropy estimate of each image. Run-length coding proved to be the best coding method for the bit-plane coded images, whereas the 2-D techniques (such as PDQ, DDC, and RAC) performed better when compressing the binary image. Furthermore, the relatively straightforward procedure of Gray coding the image of Fig. 8.14(a) improved the achievable coding performance by about 1 bit/pixel. Finally, note that the five compression methods were able to compress the monochrome image only by a factor of 1 to 2, while compressing the binary image of Fig. 8.14(b) by a factor of 2 to 5. As Table 8.8 shows, the reason for this performance difference is that the algorithms were unable to compress the lower-order bit planes of the bit-plane coded images. In fact, the dashed entries of the table indicate instances in which the algorithms caused data expansion. In these cases, the raw data were used to represent the bit plane, and only 1 bit/pixel was added to the total code rate. ■

8.4.4 Lossless Predictive Coding

Let us now turn to an error-free compression approach that does not require decomposition of an image into a collection of bit planes. The approach, commonly referred to as *lossless predictive coding*, is based on eliminating the interpixel redundancies of closely spaced pixels by extracting and coding *only* the new information in each pixel. The *new information* of a pixel is defined as the difference between the actual and predicted value of that pixel.

Figure 8.19 shows the basic components of a lossless predictive coding system. The system consists of an encoder and a decoder, each containing an identical *predictor*. As each successive pixel of the input image, denoted f_n, is introduced to the encoder, the predictor generates the anticipated value of that pixel based on some number of past inputs. The output of the predictor is then rounded to the nearest integer, denoted $\hat{f}_n$, and used to form the difference or *prediction error*

$$e_n = f_n - \hat{f}_n, \tag{8.4-5}$$

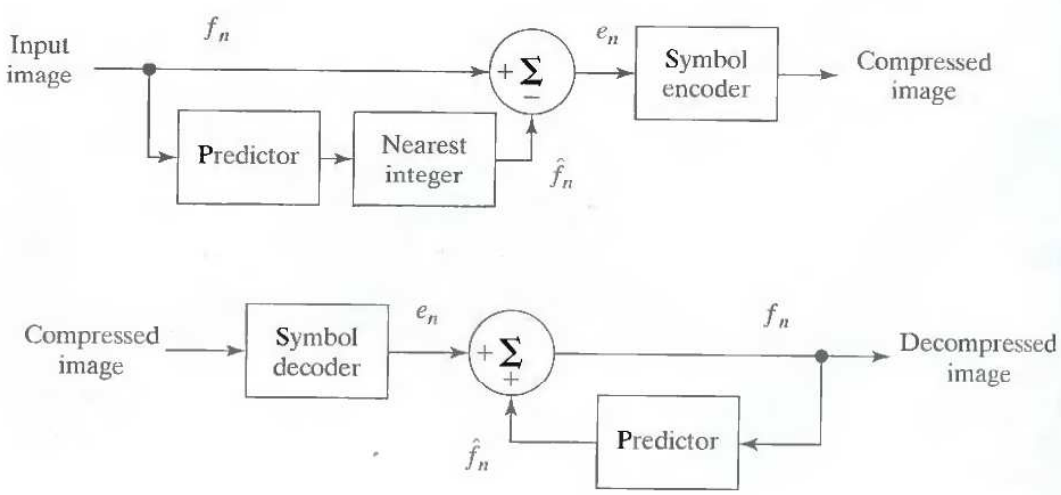


FIGURE 8.19 A lossless predictive coding model: (a) encoder; (b) decoder.

which is coded using a variable-length code (by the symbol encoder) to generate the next element of the compressed data stream. The decoder of Fig. 8.19(b) reconstructs e_n from the received variable-length code words and performs the inverse operation

$$f_n = e_n + \hat{f}_n. \tag{8.4-6}$$

Various local, global, and adaptive (see Section 8.5.1) methods can be used to generate $\hat{f}_n$. In most cases, however, the prediction is formed by a linear combination of m previous pixels. That is,

$$\hat{f}_n = \text{round}\left[\sum_{i=1}^{m} \alpha_i f_{n-i}\right] \tag{8.4-7}$$

where m is the order of the linear predictor, round is a function used to denote the rounding or nearest integer operation, and the α_i for $i = 1, 2, \ldots, m$ are prediction coefficients. In raster scan applications, the subscript n indexes the predictor outputs in accordance with their time of occurrence. That is, f_n, $\hat{f}_n$, and e_n in Eqs. (8.4-5) through (8.4-7) could be replaced with the more explicit notation $f(t)$, $\hat{f}(t)$, and $e(t)$, where t represents time. In other cases, n is used as an index on the spatial coordinates and/or frame number (in a time sequence of images) of an image. In 1-D linear predictive coding, for example, Eq. (8.4-7) can be written

$$\hat{f}_n(x, y) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(x, y - i)\right] \tag{8.4-8}$$

where each subscripted variable is now expressed explicitly as a function of spatial coordinates x and y. Note that Eq. (8.4-8) indicates that the 1-D linear prediction $\hat{f}(x, y)$ is a function of the previous pixels on the current line alone. In 2-D predictive coding, the prediction is a function of the previous pixels in a left-to-right, top-to-bottom scan of an image. In the 3-D case, it is based on these pixels and the previous pixels of preceding frames. Equation (8.4-8) cannot be evaluated for the first m pixels of each line, so these pixels must be coded by using other means (such as a Huffman code) and considered as an overhead of the predictive coding process. A similar comment applies to the higher-dimensional cases.

EXAMPLE 8.15: Predictive coding.

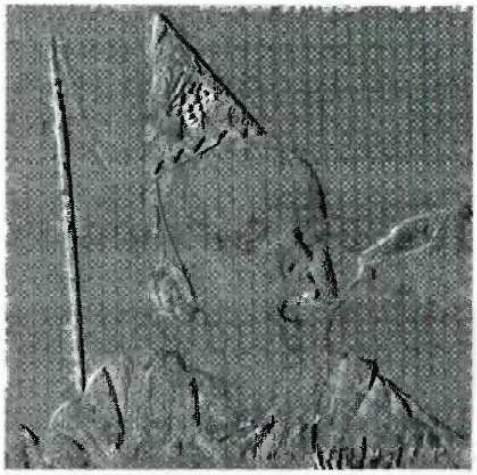
Consider encoding the monochrome image of Fig. 8.14(a) using the simple first-order linear predictor

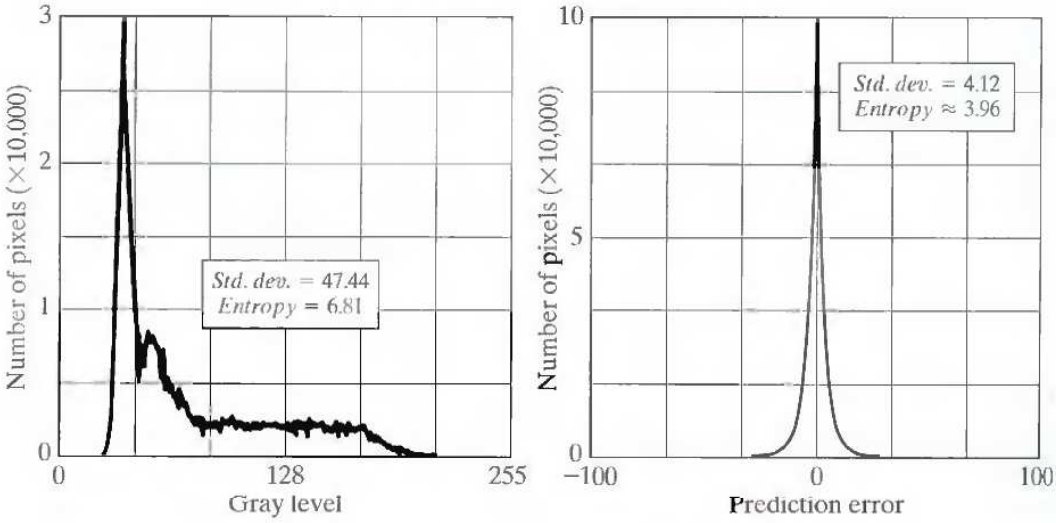
$$\hat{f}(x, y) = \text{round}[\alpha f(x, y - 1)]. \tag{8.4-9}$$

A predictor of this general form commonly is called a *previous pixel* predictor, and the corresponding predictive coding procedure is referred to as *differential coding* or *previous pixel coding*. Figure 8.20(a) shows the prediction error image that results from Eq. (8.4-9) with $\alpha = 1$. In this image, gray-level 128 represents a prediction error of zero, whereas all nonzero positive and negative prediction errors (under and over estimates) are multiplied by 8 and displayed as lighter and darker shades of gray, respectively. The mean value of the prediction image is 128.02, which corresponds to an average prediction error of only 0.02 bits.

a
b c

FIGURE 8.20
(a) The prediction error image resulting from Eq. (8.4-9).
(b) Gray-level histogram of the original image.
(c) Histogram of the prediction error.





Figures 8.20(b) and (c) show the gray-level histogram of the image in Fig. 8.14(a) and the histogram of the prediction error resulting from Eq. (8.4-9). Note that the variance of the prediction error in Fig. 8.20(c) is much smaller than the variance of the gray levels in the original image. Moreover, the first-order estimate of the entropy of the prediction error image is significantly less than the corresponding first-order estimate for the original image (3.96 bits/pixel as opposed to 6.81 bits/pixel). This decrease in entropy reflects removal of a great deal of redundancy by the predictive coding process, despite the fact that for m-bit images, $(m + 1)$-bit numbers are needed to represent accurately the error sequence that results from Eq. (8.4-5). Although any of the variable-length coding procedures of Section 8.4.1 can be used to code this error sequence, the resulting compression will be limited to approximately 8/3.96, or about 2:1. In general, an estimate of the maximum compression of any lossless predictive coding approach may be obtained by dividing the average number of bits used to represent each pixel in the original image by a first-order estimate of the entropy of the prediction error data. ■

The preceding example emphasizes that the amount of compression achieved in lossless predictive coding is related directly to the entropy reduction that results

from mapping the input image into the prediction error sequence. Because a great deal of interpixel redundancy is removed by the prediction and differencing process, the probability density function of the prediction error is, in general, highly peaked at zero and characterized by a relatively small (in comparison to the input gray-level distribution) variance. In fact, the density function of the prediction error often is modeled by the zero mean uncorrelated Laplacian pdf

$$p_e(e) = \frac{1}{\sqrt{2}\sigma_e} e^{\frac{-\sqrt{2}|e|}{\sigma_e}} \tag{8.4-10}$$

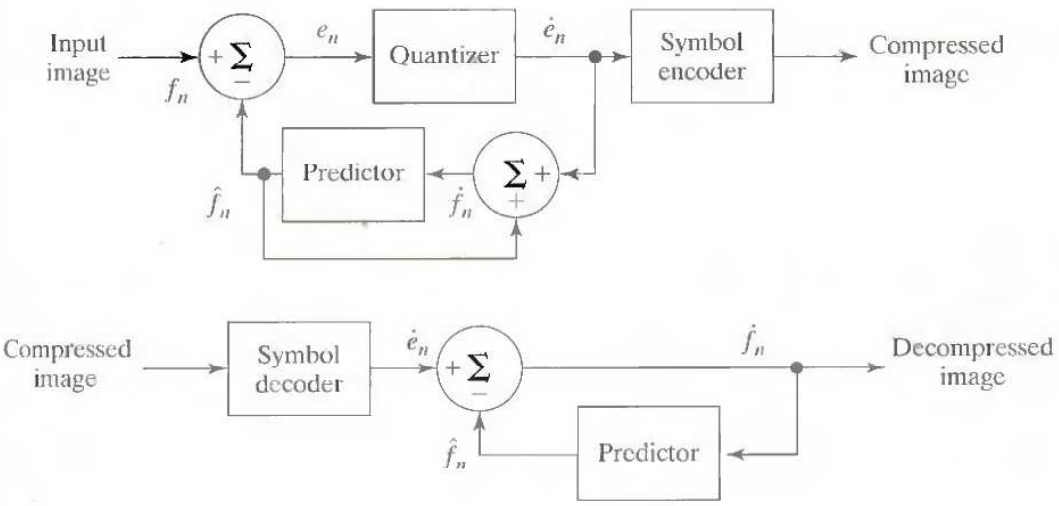
where σ_e is the standard deviation of e.

8.5 Lossy Compression

Unlike the error-free approaches outlined in the previous section, lossy encoding is based on the concept of compromising the accuracy of the reconstructed image in exchange for increased compression. If the resulting distortion (which may or may not be visually apparent) can be tolerated, the increase in compression can be significant. In fact, many lossy encoding techniques are capable of reproducing recognizable monochrome images from data that have been compressed by more than 100:1 and images that are virtually indistinguishable from the originals at 10:1 to 50:1. Error-free encoding of monochrome images, however, seldom results in more than a 3:1 reduction in data. As indicated in Section 8.2, the principal difference between these two approaches is the presence or absence of the quantizer block of Fig. 8.6.

8.5.1 Lossy Predictive Coding

In this section, we add a quantizer to the model introduced in Section 8.4.4 and examine the resulting trade-off between reconstruction accuracy and compression performance. As Fig. 8.21 shows, the quantizer, which absorbs the nearest integer function of the error-free encoder, is inserted between the symbol



a
b

FIGURE 8.21 A lossy predictive coding model: (a) encoder and (b) decoder.

encoder and the point at which the prediction error is formed. It maps the prediction error into a limited range of outputs, denoted $\dot{e}_n$, which establish the amount of compression and distortion associated with lossy predictive coding.

In order to accommodate the insertion of the quantization step, the error-free encoder of Fig. 8.19(a) must be altered so that the predictions generated by the encoder and decoder are equivalent. As Fig. 8.21(a) shows, this is accomplished by placing the lossy encoder's predictor within a feedback loop, where its input, denoted $\dot{f}_n$, is generated as a function of past predictions and the corresponding quantized errors. That is,

$$\dot{f}_n = \dot{e}_n + \hat{f}_n \tag{8.5-1}$$

where $\hat{f}_n$ is as defined in Section 8.4.4. This closed loop configuration prevents error buildup at the decoder's output. Note from Fig. 8.21(b) that the output of the decoder also is given by Eq. (8.5-1).

EXAMPLE 8.16: Delta modulation.

■ *Delta modulation* (DM) is a simple but well-known form of lossy predictive coding in which the predictor and quantizer are defined as

$$\hat{f}_n = \alpha \dot{f}_{n-1} \tag{8.5-2}$$

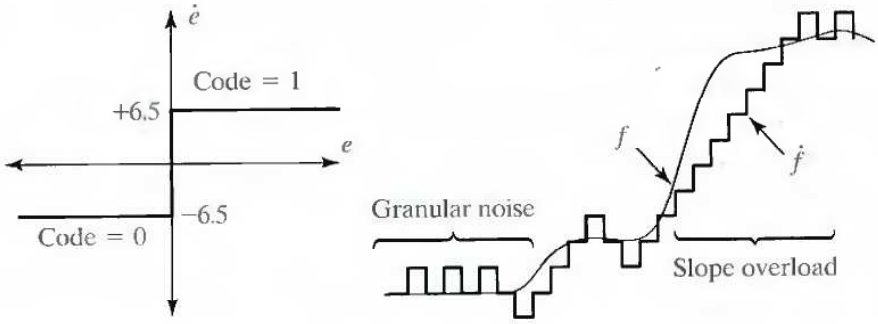
and

$$\dot{e}_n = \begin{cases} +\zeta & \text{for } e_n > 0 \\ -\zeta & \text{otherwise} \end{cases} \tag{8.5-3}$$

where α is a prediction coefficient (normally less than 1) and ζ is a positive constant. The output of the quantizer, $\dot{e}_n$, can be represented by a single bit (Fig. 8.22a), so the symbol encoder of Fig. 8.21(a) can utilize a 1-bit fixed-length code. The resulting DM code rate is 1 bit/pixel.

Figure 8.22(c) illustrates the mechanics of the delta modulation process, where the calculations needed to compress and reconstruct the input sequence $\{14, 15, 14, 15, 13, 15, 15, 14, 20, 26, 27, 28, 27, 27, 29, 37, 47, 62, 75, 77, 78, 79, 80, 81, 81, 82, 82\}$ with $\alpha = 1$ and $\zeta = 6.5$ are tabulated. The process begins with the error-free transfer of the first input pixel to the decoder. With the initial condition $\dot{f}_0 = f_0 = 14$ established at both the encoder and decoder, the remaining outputs can be computed by repeatedly evaluating Eqs. (8.5-2), (8.4-5), (8.5-3), and (8.5-1). Thus, when $n = 1$, for example, $\hat{f}_1 = (1)(14) = 14$, $e_1 = 15 - 14 = 1$, $\dot{e}_1 = +6.5$ (because $e_1 > 0$), $\dot{f}_1 = 6.5 + 14 = 20.5$, and the resulting reconstruction error is $(15 - 20.5)$, or -5.5 gray levels.

Figure 8.22(b) shows graphically the tabulated data shown in Fig. 8.22(c). Both the input and completely decoded output (f_n and $\dot{f}_n$) are shown. Note that in the rapidly changing area from $n = 14$ to 19, where ζ was too small to represent the input's largest changes, a distortion known as *slope overload* occurs. Moreover, when ζ was too large to represent the input's smallest changes, as in the relatively smooth region from $n = 0$ to $n = 7$, *granular noise* appears. In most images, these two phenomena lead to blurred object edges and grainy or noisy surfaces (that is, distorted smooth areas). ■



a b
c

FIGURE 8.22 An example of delta modulation.

Input		Encoder				Decoder		Error
n	f	$\hat{f}$	e	$\dot{e}$	$\dot{f}$	$\hat{f}$	$\dot{f}$	$[f-\dot{f}]$
0	14	—	—	—	14.0	—	14.0	0.0
1	15	14.0	1.0	6.5	20.5	14.0	20.5	−5.5
2	14	20.5	−6.5	−6.5	14.0	20.5	14.0	0.0
3	15	14.0	1.0	6.5	20.5	14.0	20.5	−5.5
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
14	29	20.5	8.5	6.5	27.0	20.5	27.0	2.0
15	37	27.0	10.0	6.5	33.5	27.0	33.5	3.5
16	47	33.5	13.5	6.5	40.0	33.5	40.0	7.0
17	62	40.0	22.0	6.5	46.5	40.0	46.5	15.5
18	75	46.5	28.5	6.5	53.0	46.5	53.0	22.0
19	77	53.0	24.0	6.5	59.6	53.0	59.6	17.5
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

The distortions noted in the preceding example are common to all forms of lossy predictive coding. The severity of these distortions depends on a complex set of interactions between the quantization and prediction methods employed. Despite these interactions, the predictor normally is designed under the assumption of no quantization error, and the quantizer is designed to minimize its own error. That is, the predictor and quantizer are designed independently of each other.

Optimal predictors

The optimal predictor used in most predictive coding applications minimizes the encoder's mean-square prediction error†

$$E\{e_n^2\} = E\{[f_n - \hat{f}_n]^2\} \tag{8.5-4}$$

subject to the constraint that

$$\dot{f}_n = \dot{e}_n + \hat{f}_n \approx e_n + \hat{f}_n = f_n \tag{8.5-5}$$

and

$$\hat{f}_n = \sum_{i=1}^{m} \alpha_i f_{n-i}. \tag{8.5-6}$$

†The notation $E\{\cdot\}$ denotes the statistical expectation operator.

That is, the optimization criterion is chosen to minimize the mean-square prediction error, the quantization error is assumed to be negligible ($\dot{e}_n \approx e_n$), and the prediction is constrained to a linear combination of m previous pixels.† These restrictions are not essential, but they simplify the analysis considerably and, at the same time, decrease the computational complexity of the predictor. The resulting predictive coding approach is referred to as *differential pulse code modulation* (DPCM).

Under these conditions, the optimal predictor design problem is reduced to the relatively straightforward exercise of selecting the m prediction coefficients that minimize the expression

$$E\{e_n^2\} = E\left\{\left[f_n - \sum_{i=1}^{m} \alpha_i f_{n-1}\right]^2\right\}. \tag{8.5-7}$$

Differentiating Eq. (8.5-7) with respect to each coefficient, equating the derivatives to zero, and solving the resulting set of simultaneous equations under the assumption that f_n has mean zero and variance σ^2 yields

$$\boldsymbol{\alpha} = \mathbf{R}^{-1}\mathbf{r} \tag{8.5-8}$$

where $\mathbf{R}^{-1}$ is the inverse of the $m \times m$ autocorrelation matrix

$$\mathbf{R} = \begin{bmatrix} E\{f_{n-1}f_{n-1}\} & E\{f_{n-1}f_{n-2}\} & \cdots & E\{f_{n-1}f_{n-m}\} \\ E\{f_{n-2}f_{n-1}\} & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & & & \\ E\{f_{n-m}f_{n-1}\} & E\{f_{n-m}f_{n-2}\} & \cdots & E\{f_{n-m}f_{n-m}\} \end{bmatrix} \tag{8.5-9}$$

and $\mathbf{r}$ and $\boldsymbol{\alpha}$ are the m-element vectors

$$\mathbf{r} = \begin{bmatrix} E\{f_n f_{n-1}\} \\ E\{f_n f_{n-2}\} \\ \vdots \\ E\{f_n f_{n-m}\} \end{bmatrix} \quad \text{and} \quad \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}. \tag{8.5-10}$$

Thus for any input image, the coefficients that minimize Eq. (8.5-7) can be determined via a series of elementary matrix operations. Moreover, the coefficients depend only on the autocorrelations of the pixels in the original image. The variance of the prediction error that results from the use of these optimal coefficients is

$$\sigma_e^2 = \sigma^2 - \boldsymbol{\alpha}^T\mathbf{r} = \sigma^2 - \sum_{i=1}^{m} E\{f_n f_{n-i}\}\alpha_i. \tag{8.5-11}$$

Although the mechanics of evaluating Eq. (8.5-8) are quite simple, computation of the autocorrelations needed to form $\mathbf{R}$ and $\mathbf{r}$ is so difficult in practice

†In general, the optimal predictor for a non-Gaussian image is a nonlinear function of the pixels used to form the estimate.

that *local* predictions (those in which the prediction coefficients are computed image by image) are almost never used. In most cases, a set of *global* coefficients is computed by assuming a simple image model and substituting the corresponding autocorrelations into Eqs. (8.5-9) and (8.5-10). For instance, when a 2-D Markov source (see Section 8.3.3) with separable autocorrelation function

$$E\{f(x, y)f(x - i, y - i)\} = \sigma^2 \rho_v^i \rho_h^j \tag{8.5-12}$$

and generalized fourth-order linear predictor

$$\begin{aligned}\hat{f}(x, y) = {} & \alpha_1 f(x, y - 1) + \alpha_2 f(x - 1, y - 1) \\ & + \alpha_3 f(x - 1, y) + \alpha_4 f(x - 1, y + 1)\end{aligned} \tag{8.5-13}$$

are assumed, the resulting optimal coefficients (Jain [1989]) are

$$\alpha_1 = \rho_h \quad \alpha_2 = -\rho_v \rho_h \quad \alpha_3 = \rho_v \quad \alpha_4 = 0 \tag{8.5-14}$$

where ρ_h and ρ_v are the horizontal and vertical correlation coefficients, respectively, of the image under consideration.

Finally, the sum of the prediction coefficients in Eq. (8.5-6) normally is required to be less than or equal to one. That is,

$$\sum_{i=1}^{m} \alpha_i \leq 1. \tag{8.5-15}$$

This restriction is made to ensure that the predictor's output falls within the allowed range of gray levels and to reduce the impact of transmission noise, which is generally seen as horizontal streaks in the reconstructed image. Reducing the DPCM decoder's susceptibility to input noise is important, because a single error (under the right circumstances) can propagate to all future outputs. That is, the decoder's output may become unstable. By further restricting Eq. (8.5-15) to be strictly less than 1 confines the impact of an input error to a small number of outputs.

EXAMPLE 8.17: Comparison of prediction techniques.

■ Consider the prediction error that results from DPCM coding the monochrome image of Fig. 8.23 under the assumption of zero quantization error and with each of four predictors:

FIGURE 8.23 A 512 × 512 8-bit monochrome image.

$$\hat{f}(x, y) = 0.97f(x, y - 1) \tag{8.5-16}$$

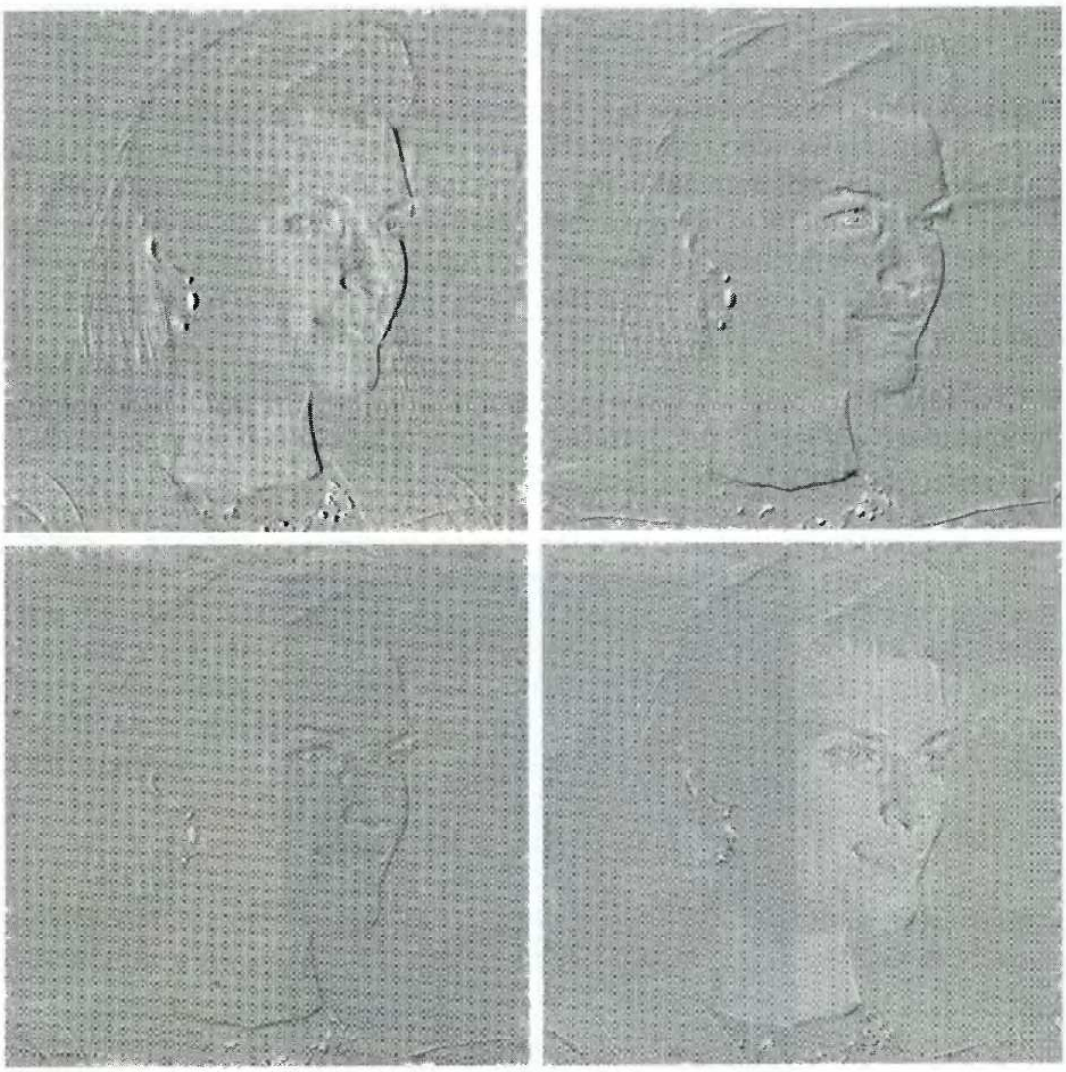
$$\hat{f}(x, y) = 0.5f(x, y - 1) + 0.5f(x - 1, y) \tag{8.5-17}$$

$$\hat{f}(x, y) = 0.75f(x, y - 1) + 0.75f(x - 1, y) - 0.5f(x - 1, y - 1) \tag{8.5-18}$$

$$\hat{f}(x, y) = \begin{cases} 0.97f(x, y - 1) & \text{if } \Delta h \leq \Delta v \\ 0.97f(x - 1, y) & \text{otherwise} \end{cases} \tag{8.5-19}$$

where $\Delta h = |f(x - 1, y) - f(x - 1, y - 1)|$ and $\Delta v = |f(x, y - 1) - f(x - 1, y - 1)|$ denote the horizontal and vertical gradients at point (x, y). Equations (8.5-16) through (8.5-18) define a relatively robust set of α_i, which provide satisfactory performance over a wide range of images. The adaptive predictor of Eq. (8.5-19) is designed to improve edge rendition by computing a local measure of the directional properties of an image (Δh and Δv) and selecting a predictor specifically tailored to the measured behavior.

Figures 8.24(a) through (d) show the prediction error images that result from using the predictors of Eqs. (8.5-16) through (8.5-19). Note that the visually



a b
c d

FIGURE 8.24 A comparison of four linear prediction techniques.

perceptible error decreases as the order of the predictor increases.[†] The standard deviations of the prediction error distributions follow a similar pattern. They are 4.9, 3.7, 3.3, and 4.1 gray levels, respectively. ■

Optimal quantization

The staircase quantization function $t = q(s)$ shown in Fig. 8.25 is an odd function of s [that is, $q(-s) = -q(s)$] that can be completely described by the $L/2$ values of s_i and t_i shown in the first quadrant of the graph. These break points define function discontinuities and are called the *decision* and *reconstruction levels* of the quantizer. As a matter of convention, s is considered to be mapped to t_i if it lies in the half-open interval $(s_i, s_{i+1}]$.

The quantizer design problem is to select the best s_i and t_i for a particular optimization criterion and input probability density function $p(s)$. If the optimization criterion, which could be either a statistical or psychovisual measure,[‡] is the minimization of the mean-square quantization error (that is, $E\{(s - t_i)^2\}$) and $p(s)$ is an even function, the conditions for minimal error (Max [1960]) are

$$\int_{s_{i-1}}^{s_i} (s - t_i)p(s)\, ds = 0 \qquad i = 1, 2, \ldots, \frac{L}{2} \tag{8.5-20}$$

$$s_i = \begin{cases} 0 & i = 0 \\ \dfrac{t_i + t_{i+1}}{2} & i = 1, 2, \ldots, \dfrac{L}{2} - 1 \\ \infty & i = \dfrac{L}{2} \end{cases} \tag{8.5-21}$$

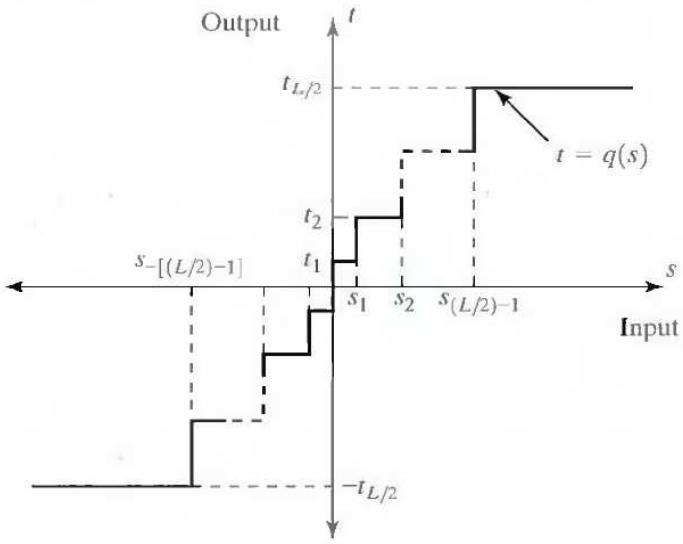


FIGURE 8.25 A typical quantization function.

[†] Predictors that use more than three or four previous pixels provide little compression gain for the added predictor complexity (Habibi [1971]).

[‡] See Netravali [1977] and Limb and Rubinstein [1978] for more on psychovisual measures.

TABLE 8.10 Lloyd-Max quantizers for a Laplacian probability density function of unit variance.

Levels	2		4		8	
i	s_i	t_i	s_i	t_i	s_i	t_i
1	∞	0.707	1.102	0.395	0.504	0.222
2			∞	1.810	1.181	0.785
3					2.285	1.576
4					∞	2.994
θ	1.414		1.087		0.731	

and

$$s_{-i} = -s_i \quad t_{-i} = -t_i. \tag{8.5-22}$$

Equation (8.5-20) indicates that the reconstruction levels are the centroids of the areas under $p(s)$ over the specified decision intervals, whereas Eq. (8.5-21) indicates that the decision levels are halfway between the reconstruction levels. Equation (8.5-22) is a consequence of the fact that q is an odd function. For any L, the s_i and t_i that satisfy Eqs. (8.5-20) through (8.5-22) are optimal in the mean-square error sense; the corresponding quantizer is called an L-level *Lloyd-Max* quantizer.

Table 8.10 lists the 2-, 4-, and 8-level Lloyd-Max decision and reconstruction levels for a unit variance Laplacian probability density function [see Eq. (8.4-10)]. Because obtaining an explicit or closed-form solution to Eqs. (8.5-20) through (8.5-22) for most nontrivial $p(s)$ is difficult, these values were generated numerically (Paez and Glisson [1972]). The three quantizers shown provide fixed output rates of 1, 2, and 3 bits/pixel, respectively. As Table 8.10 was constructed for a unit variance distribution, the reconstruction and decision levels for the case of $\sigma \neq 1$ are obtained by multiplying the tabulated values by the standard deviation of the probability density function under consideration. The final row of the table lists the step size, θ, that simultaneously satisfies Eqs. (8.5-20) through (8.5-22) *and* the additional constraint that

$$t_i - t_{i-1} = s_i - s_{i-1} = \theta. \tag{8.5-23}$$

If a symbol encoder that utilizes a variable-length code is used in the general lossy predictive encoder of Fig. 8.21(a), an *optimum uniform quantizer* of step size θ will provide a lower code rate (for a Laplacian pdf) than a fixed-length coded Lloyd-Max quantizer with the same output fidelity (O'Neil [1971]).

Although the Lloyd-Max and optimum uniform quantizers are not adaptive, much can be gained from adjusting the quantization levels based on the local behavior of an image. In theory, slowly changing regions can be finely quantized, while the rapidly changing areas are quantized more coarsely. This approach simultaneously reduces both granular noise and slope overload, while requiring only a minimal increase in code rate. The trade-off is increased quantizer complexity.

EXAMPLE 8.18: Illustration of quantization and reconstruction.

■ Figures 8.26(a), (c), and (e) show the DPCM reconstructed images that resulted from combining the 2-, 4-, and 8-level Lloyd-Max quantizers in Table 8.10 with the planar predictor of Eq. (8.5-18). The quantizers were generated by

multiplying the tabulated Lloyd-Max decision and reconstruction levels by the standard deviation of the nonquantized planar prediction error from the preceding example (that is, 3.3 gray levels). Note that the edges of the decoded images are blurred from slope overload. This result is particularly noticeable in Fig. 8.26(a), which was generated using a two-level quantizer, but is less apparent in Figs. 8.26(c) and (e), where four and eight quantization levels were applied. Figures 8.27(a), (c), and (e) show the scaled differences between these decoded images and the original image of Fig. 8.23.

In order to generate the decoded images in Figs. 8.26(b), (d), and (f), and the resulting error images in Figs. 8.27(b), (d), and (f), we used an adaptive quantization method in which the best (in a mean-square error sense) of four possible quantizers was selected for each block of 16 pixels. The four quantizers were scaled versions of the optimal Lloyd-Max quantizers previously described. The scaling factors were 0.5, 1.0, 1.75, and 2.5. Because a 2-bit code was appended to each block in order to specify the selected quantizer, the overhead associated with the quantizer switching was $^2/_{16}$ or 0.125 bits/pixel. Note the substantial decrease in perceived error that resulted from this relatively small increase in code rate.

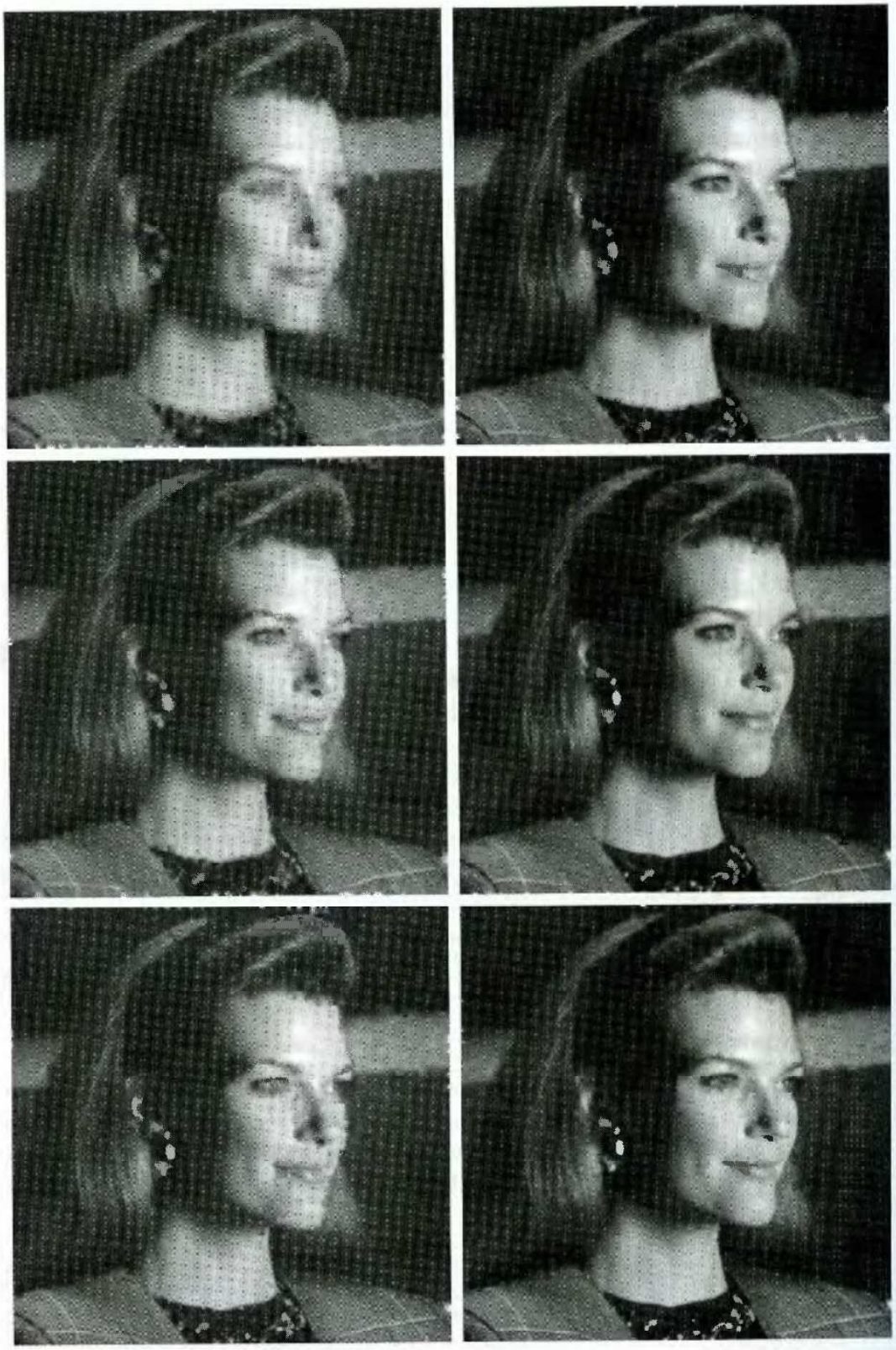
Table 8.11 lists the rms errors of the difference images in Figs. 8.27(a) through (f), as well as for a number of other combinations of predictors and quantizers. Note that in a mean-square error sense, the two-level adaptive quantizers performed about as well as the four-level nonadaptive versions. Moreover, the four-level adaptive quantizers outperformed the eight-level nonadaptive approaches. In general, the numerical results indicate that the predictors of Eqs. (8.5-15), (8.5-17), and (8.5-19) exhibit the same overall characteristics as the predictor of Eq. (8.5-18). The compression that resulted under each of the quantization methods is listed in the last row of Table 8.11. Note that the substantial decrease in rms error [Eq. (8.1-8)] achieved by the adaptive approaches did not significantly affect compression performance. ■

8.5.2 Transform Coding

The predictive coding techniques discussed in Section 8.5.1 operate directly on the pixels of an image and thus are *spatial domain methods*. In this section, we consider compression techniques that are based on modifying the transform of an image. In *transform coding*, a reversible, linear transform (such as the Fourier

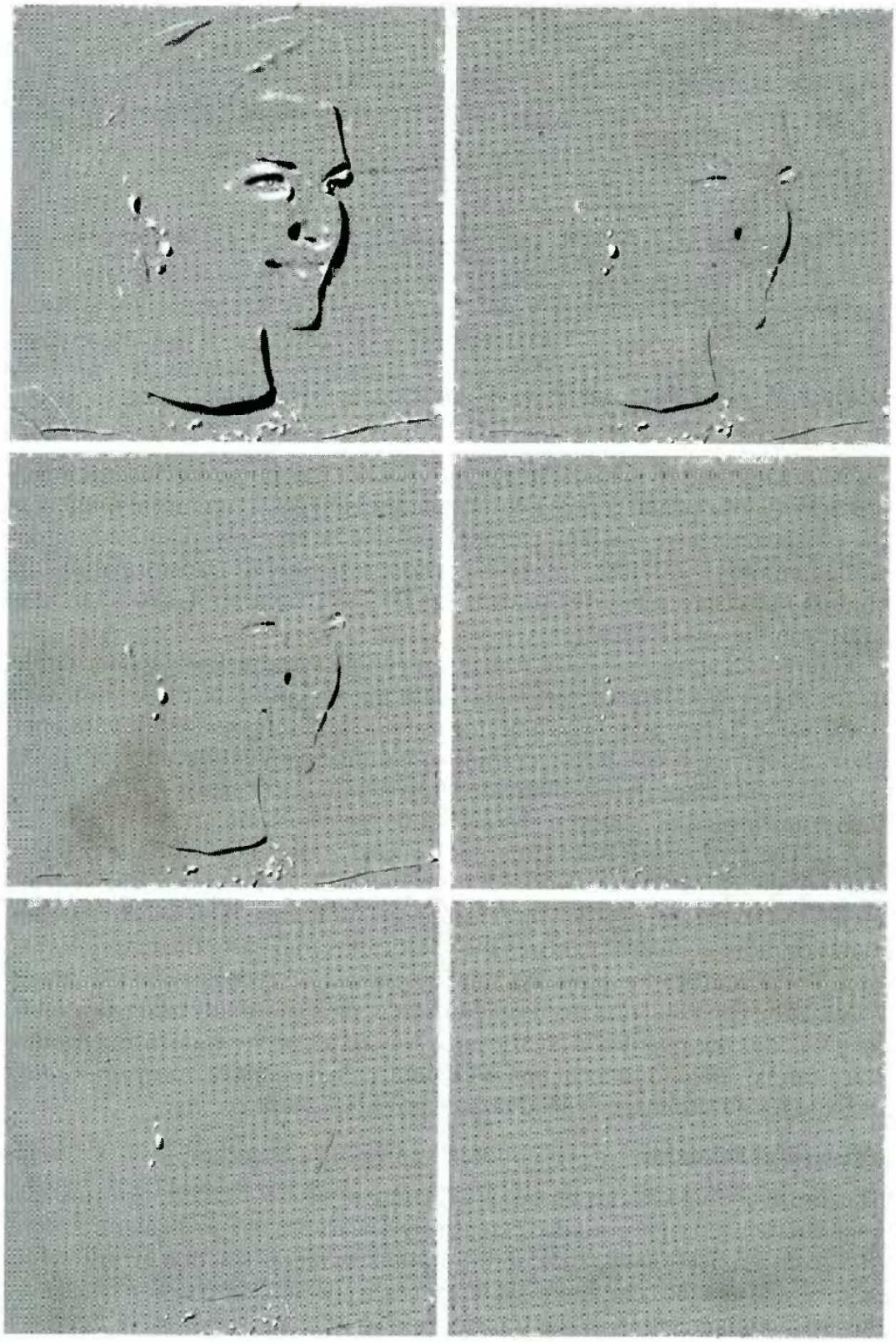
TABLE 8.11 Lossy DPCM root-mean-square error summary.

	Lloyd-Max Quantizer			Adaptive Quantizer		
Predictor	**2-level**	**4-level**	**8-level**	**2-level**	**4-level**	**8-level**
Eq. (8.5-16)	30.88	6.86	4.08	7.49	3.22	1.55
Eq. (8.5-17)	14.59	6.94	4.09	7.53	2.49	1.12
Eq. (8.5-18)	9.90	4.30	2.31	4.61	1.70	0.76
Eq. (8.5-19)	38.18	9.25	3.36	11.46	2.56	1.14
Compression	8.00:1	4.00:1	2.70:1	7.11:1	3.77:1	2.56:1



a b
c d
e f

FIGURE 8.26 DPCM result images: (a) 1.0; (b) 1.125; (c) 2.0; (d) 2.125; (e) 3.0; (f) 3.125 bits/pixel.



a b
c d
e f

FIGURE 8.27 The scaled (×8) DPCM error images that correspond to Figs. 8.26(a) through (f).

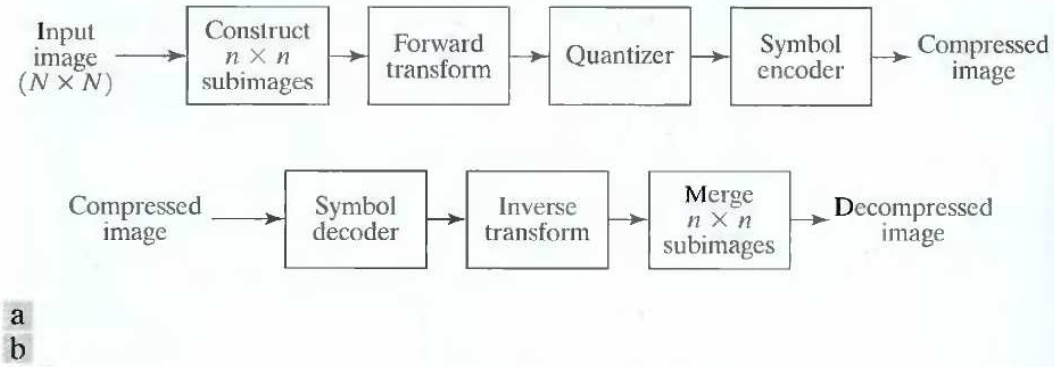


FIGURE 8.28 A transform coding system: (a) encoder; (b) decoder.

transform) is used to map the image into a set of transform coefficients, which are then quantized and coded. For most natural images, a significant number of the coefficients have small magnitudes and can be coarsely quantized (or discarded entirely) with little image distortion. A variety of transformations, including the discrete Fourier transform (DFT) of Chapter 4, can be used to transform the image data.

Figure 8.28 shows a typical transform coding system. The decoder implements the inverse sequence of steps (with the exception of the quantization function) of the encoder, which performs four relatively straightforward operations: subimage decomposition, transformation, quantization, and coding. An $N \times N$ input image first is subdivided into subimages of size $n \times n$, which are then transformed to generate $(N/n)^2$ subimage transform arrays, each of size $n \times n$. The goal of the transformation process is to decorrelate the pixels of each subimage, or to pack as much information as possible into the smallest number of transform coefficients. The quantization stage then selectively eliminates or more coarsely quantizes the coefficients that carry the least information. These coefficients have the smallest impact on reconstructed subimage quality. The encoding process terminates by coding (normally using a variable-length code) the quantized coefficients. Any or all of the transform encoding steps can be adapted to local image content, called *adaptive transform coding*, or fixed for all subimages, called *nonadaptive transform coding*.

Transform selection

Transform coding systems based on a variety of discrete 2-D transforms have been constructed and/or studied extensively. The choice of a particular transform in a given application depends on the amount of reconstruction error that can be tolerated and the computational resources available. Compression is achieved during the quantization of the transformed coefficients (not during the transformation step).

Consider an image $f(x, y)$ of size $N \times N$ whose forward, discrete transform, $T(u, v)$, can be expressed in terms of the general relation

$$T(u, v) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) g(x, y, u, v) \tag{8.5-24}$$

for $u, v = 0, 1, 2, \ldots, N-1$. Given $T(u, v)$, $f(x, y)$ similarly can be obtained using the generalized inverse discrete transform

$$f(x, y) = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} T(u, v) h(x, y, u, v) \tag{8.5-25}$$

for $x, y = 0, 1, 2, \ldots, N-1$. In these equations, $g(x, y, u, v)$ and $h(x, y, u, v)$ are called the *forward* and *inverse transformation kernels*, respectively. For reasons that will become clear later in the section, they also are referred to as *basis functions* or *basis images*. The $T(u, v)$ for $u, v = 0, 1, 2, \ldots, N-1$ in Eq. (8.5-25) are called *transform coefficients*; they can be viewed as the expansion coefficients—see Section 7.2.1—of a series expansion of $f(x, y)$ with respect to basis functions $h(x, y, u, v)$.

The forward kernel in Eq. (8.5-24) is said to be *separable* if

$$g(x, y, u, v) = g_1(x, u) g_2(y, v). \tag{8.5-26}$$

In addition, the kernel is *symmetric* if g_1 is functionally equal to g_2. In this case, Eq. (8.5-26) can be expressed in the form

$$g(x, y, u, v) = g_1(x, u) g_1(y, v). \tag{8.5-27}$$

Identical comments apply to the inverse kernel if $g(x, y, u, v)$ is replaced by $h(x, y, u, v)$ in Eqs. (8.5-26) and (8.5-27). It is not difficult to show that a 2-D transform with a separable kernel can be computed using row-column or column-row passes of the corresponding 1-D transform, in the manner explained in Section 4.6.1.

The forward and inverse transformation kernels in Eqs. (8.5-24) and (8.5-25) determine the type of transform that is computed and the overall computational complexity and reconstruction error of the transform coding system in which they are employed. The most well-known transform kernel pair is

$$g(x, y, u, v) = \frac{1}{N^2} e^{-j2\pi(ux+vy)/N} \tag{8.5-28}$$

and

$$h(x, y, u, v) = e^{j2\pi(ux+vy)/N} \tag{8.5-29}$$

where $j = \sqrt{-1}$. Substituting these kernels into Eqs. (8.5-24) and (8.5-25) yields a simplified version (in which $M = N$) of the discrete Fourier transform pair introduced in Section 4.2.2.

A computationally simpler transformation that is also useful in transform coding, called the *Walsh-Hadamard transform* (WHT), is derived from the functionally identical kernels

$$g(x, y, u, v) = h(x, y, u, v) = \frac{1}{N} (-1)^{\sum_{i=0}^{m-1} [b_i(x) p_i(u) + b_i(y) p_i(v)]} \tag{8.5-30}$$

where $N = 2^m$. The summation in the exponent of this expression is performed in modulo 2 arithmetic and $b_k(z)$ is the kth bit (from right to left) in the binary

representation of z. If $m = 3$ and $z = 6$ (110 in binary), for example, $b_0(z) = 0$, $b_1(z) = 1$, and $b_2(z) = 1$. The $p_i(u)$ in Eq. (8.5-30) are computed using:

$$\begin{aligned} p_0(u) &= b_{m-1}(u) \\ p_1(u) &= b_{m-1}(u) + b_{m-2}(u) \\ p_2(u) &= b_{m-2}(u) + b_{m-3}(u) \\ &\vdots \\ p_{m-1}(u) &= b_1(u) + b_0(u) \end{aligned} \tag{8.5-31}$$

where the sums, as noted previously, are performed in modulo 2 arithmetic. Similar expressions apply to $p_i(v)$.

Unlike the kernels of the DFT, which are sums of sines and cosines [see Eqs. (8.5-28) and (8.5-29)], the Walsh-Hadamard kernels consist of alternating plus and minus 1's arranged in a checkerboard pattern. Figure 8.29 shows the kernel for $N = 4$. Each block consists of $4 \times 4 = 16$ elements (subsquares). White denotes +1 and black denotes −1. To obtain the top, left block, we let $u = v = 0$ and plot values of $g(x, y, 0, 0)$ for $x, y = 0, 1, 2, 3$. All values in this case are +1. The second block on the top row is a plot of values of $g(x, y, 0, 1)$ for $x, y = 0, 1, 2, 3$, and so on. As already noted, the importance of the Walsh-Hadamard transform is its simplicity of implementation—all kernel values are +1 or −1.

One of the most frequently used transformations for image compression is the *discrete cosine transform* (DCT). It is obtained by substituting the following (equal) kernels into Eqs. (8.5-24) and (8.5-25)

$$\begin{aligned} g(x, y, u, v) &= h(x, y, u, v) \\ &= \alpha(u)\alpha(v) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right] \end{aligned} \tag{8.5-32}$$

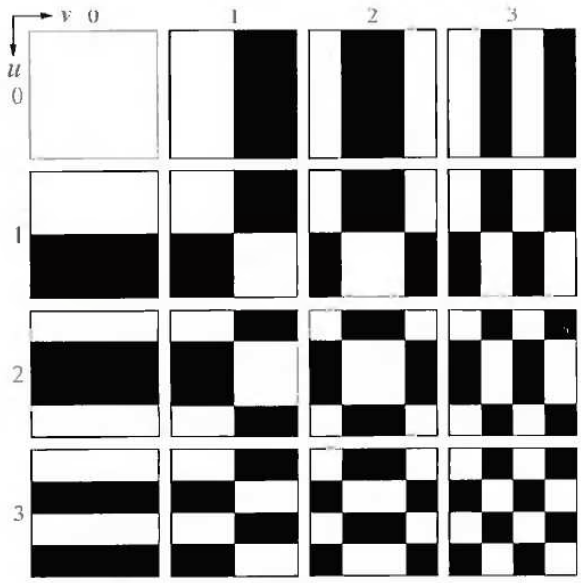


FIGURE 8.29 Walsh-Hadamard basis functions for $N = 4$. The origin of each block is at its top left.

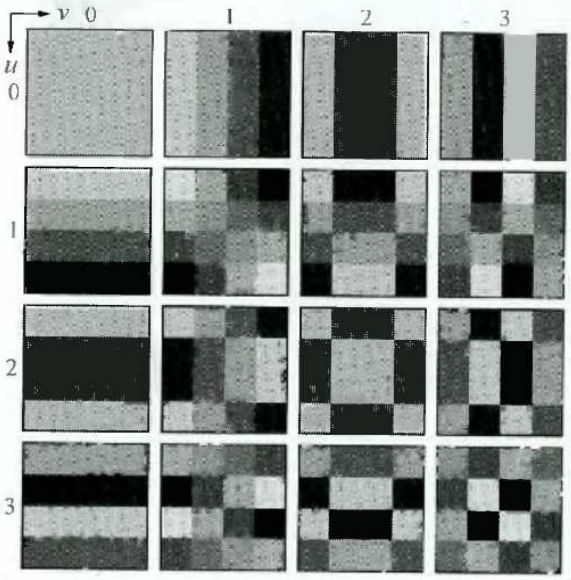


FIGURE 8.30 Discrete-cosine basis functions for $N = 4$. The origin of each block is at its top left.

where

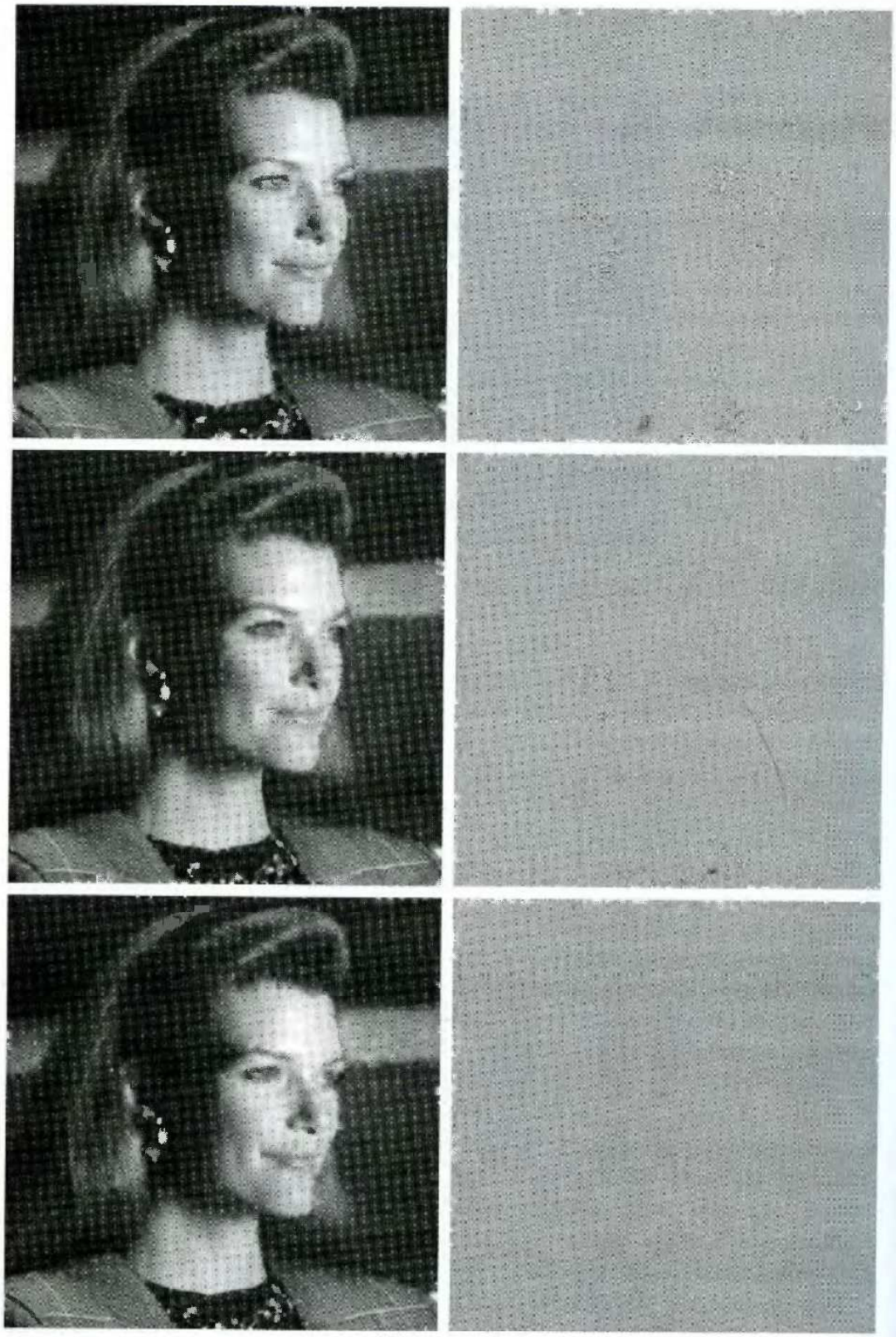
$$\alpha(u) = \begin{cases} \sqrt{\dfrac{1}{N}} & \text{for } u = 0 \\ \sqrt{\dfrac{2}{N}} & \text{for } u = 1, 2, \ldots, N-1 \end{cases} \tag{8.5-33}$$

and similarly for $\alpha(v)$. Figure 8.30 shows $g(x, y, u, v)$ for the case $N = 4$. The computation follows the same format as explained for Fig. 8.29, with the difference that the values of g are not integers. In Fig. 8.30, the lighter gray levels correspond to larger values of g.

EXAMPLE 8.19: Transform coding with the DFT, WHT, and DCT.

■ Figures 8.31(a), (c), and (e) show three approximations of the 512×512 monochrome image in Fig. 8.23. These pictures were obtained by dividing the original image into subimages of size 8×8, representing each subimage using one of the transforms just described (i.e., the DFT, WHT, or DCT transform), truncating 50% of the resulting coefficients, and taking the inverse transform of the truncated coefficient arrays.

In each case, the 32 retained coefficients were selected on the basis of maximum magnitude. When we disregard any quantization or coding issues, this process amounts to compressing the original image by a factor of 2. Note that in all cases, the 32 discarded coefficients had little visual impact on reconstructed image quality. Their elimination, however, was accompanied by some mean-square error, which can be seen in the scaled error images of Figs. 8.31(b), (d), and (f). The actual rms errors were 1.28, 0.86, and 0.68 gray levels, respectively. ■



a b
c d
e f

FIGURE 8.31 Approximations of Fig. 8.23 using the (a) Fourier, (c) Hadamard, and (e) cosine transforms, together with the corresponding scaled error images.

The small differences in mean-square reconstruction error noted in the preceding example are related directly to the energy or information packing properties of the transforms employed. In accordance with Eq. (8.5-25), an $n \times n$ image $f(x, y)$ can be expressed as a function of its 2-D transform $T(u, v)$:

$$f(x, y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) h(x, y, u, v) \tag{8.5-34}$$

for $x, y = 0, 1, \ldots, n - 1$. Note that we merely replaced N in Eq. (8.5-25) with n and now consider $f(x, y)$ to represent a subimage of the image being compressed. Since the inverse kernel $h(x, y, u, v)$ in Eq. (8.5-34) depends only on the indices x, y, u, v, and not on the values of $f(x, y)$ or $T(u, v)$, it can be viewed as defining a set of *basis functions* or *basis images* for the series defined by Eq. (8.5-34). This interpretation becomes clearer if the notation used in Eq. (8.5-34) is modified to obtain

$$\mathbf{F} = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v) \mathbf{H}_{uv} \tag{8.5-35}$$

where $\mathbf{F}$ is an $n \times n$ matrix containing the pixels of $f(x, y)$ and

$$\mathbf{H}_{uv} = \begin{bmatrix} h(0, 0, u, v) & h(0, 1, u, v) & \cdots & h(0, n-1, u, v) \\ h(1, 0, u, v) & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & & & \\ h(n-1, 0, u, v) & h(n-1, 1, u, v) & \cdots & h(n-1, n-1, u, v) \end{bmatrix}. \tag{8.5-36}$$

Then $\mathbf{F}$, the matrix containing the pixels of the input subimage, is explicitly defined as a linear combination of n^2 matrices of size $n \times n$; that is, the $\mathbf{H}_{uv}$ for $u, v = 0, 1, \ldots, n - 1$ in Eq. (8.5-36). These matrices in fact are the basis images (or functions) of the series expansion in Eq. (8.5-35); the associated $T(u, v)$ are the expansion coefficients. Figures 8.29 and 8.30 illustrate graphically the WHT and DCT basis images for the case of $n = 4$.

If we now define a transform coefficient *masking function*

$$\gamma(u, v) = \begin{cases} 0 & \text{if } T(u, v) \text{ satisfies a specified truncation criterion} \\ 1 & \text{otherwise} \end{cases} \tag{8.5-37}$$

for $u, v = 0, 1, \ldots, n - 1$, an approximation of $\mathbf{F}$ can be obtained from the truncated expansion

$$\hat{\mathbf{F}} = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} \gamma(u, v) T(u, v) \mathbf{H}_{uv} \tag{8.5-38}$$

where $\gamma(u, v)$ is constructed to eliminate the basis images that make the smallest contribution to the total sum in Eq. (8.5-35). The mean-square error between subimage $\mathbf{F}$ and approximation $\hat{\mathbf{F}}$ then is

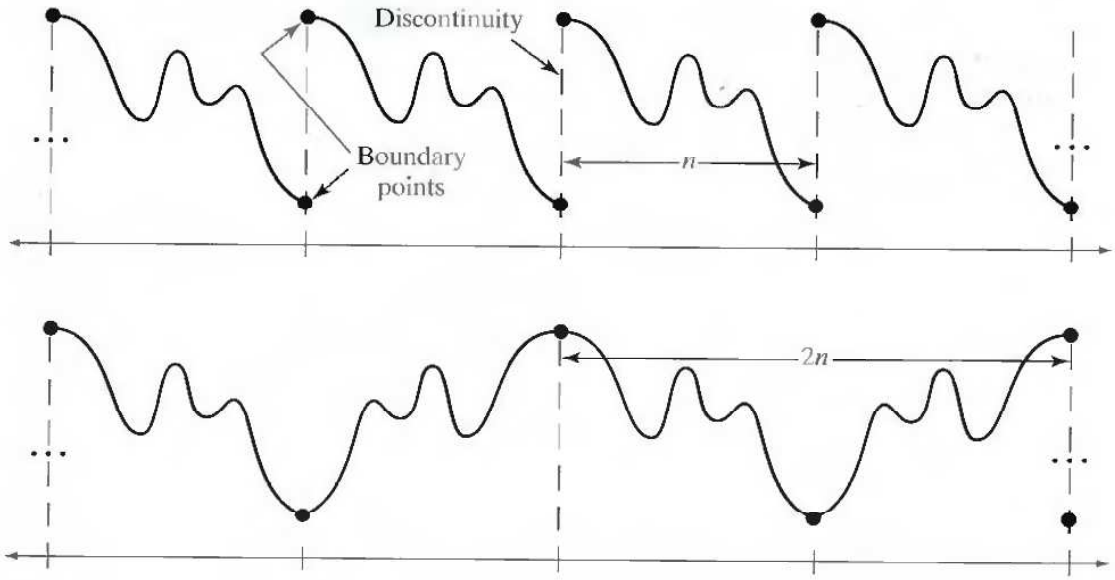
$$\begin{aligned} e_{\text{ms}} &= E\{\|\mathbf{F} - \hat{\mathbf{F}}\|^2\} \\ &= E\left\{\left\|\sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u, v)\mathbf{H}_{uv} - \sum_{u=0}^{n-1}\sum_{v=0}^{n-1}\gamma(u, v)T(u, v)\mathbf{H}_{uv}\right\|^2\right\} \\ &= E\left\{\left\|\sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u, v)\mathbf{H}_{uv}[1 - \gamma(u, v)]\right\|^2\right\} \\ &= \sum_{u=0}^{n-1}\sum_{v=0}^{n-1}\sigma^2_{T(u,v)}[1 - \gamma(u, v)] \end{aligned} \tag{8.5-39}$$

where $\|\mathbf{F} - \hat{\mathbf{F}}\|$ is the norm of matrix $(\mathbf{F} - \hat{\mathbf{F}})$, and $\sigma^2_{T(u,v)}$ is the variance of the coefficient at transform location (u, v). The final simplification is based on the orthonormal nature of the basis images and the assumption that the pixels of $\mathbf{F}$ are generated by a random process with zero mean and known covariance. The total mean-square approximation error thus is the sum of the variances of the discarded transform coefficients; that is, the coefficients for which $\gamma(u, v) = 0$, so that $[1 - \gamma(u, v)]$ in Eq. (8.5-39) is 1. Transformations that redistribute or pack the most information into the fewest coefficients provide the best subimage approximations and, consequently, the smallest reconstruction errors. Finally, under the assumptions that led to Eq. (8.5-39), the mean-square error of the $(N/n)^2$ subimages of an $N \times N$ image are identical. Thus the mean-square error (being a measure of *average* error) of the $N \times N$ image equals the mean-square error of a single subimage.

The earlier example showed that the information packing ability of the DCT is superior to that of the DFT and WHT. Although this condition usually holds for most natural images, the Karhunen-Loève transform (the KLT is discussed in Chapter 11), not the DCT, is the optimal transform in an information packing sense. That is, the KLT minimizes the mean-square error in Eq. (8.5-39) for any input image and any number of retained coefficients (Kramer and Mathews [1956]).[†] However, because the KLT is data dependent, obtaining the KLT basis images for each subimage, in general, is a nontrivial computational task. For this reason, the KLT is seldom used in practice for image compression. Instead, a transform, such as the DFT, WHT, or DCT, whose basis images are fixed (input independent), normally is used. Of the possible input independent transforms, the nonsinusoidal transforms (such as the WHT transform) are the simplest to implement. The sinusoidal transforms (such as the DFT or DCT) more closely approximate the information packing ability of the optimal KLT.

Hence many transform coding systems are based on the DCT, which provides a good compromise between information packing ability and computational complexity. In fact, the properties of the DCT have proved to be of such practical value that the DCT has become an international standard for transform coding

[†] An additional condition for optimality is that the masking function of Eq. (8.5-37) selects the KLT coefficients of maximum variance.



a
b

FIGURE 8.32 The periodicity implicit in the 1-D (a) DFT and (b) DCT.

systems (see Section 8.6). Compared to the other input independent transforms, it has the advantages of having been implemented in a single integrated circuit, packing the most information into the fewest coefficients[†] (for most natural images), and minimizing the blocklike appearance, called *blocking artifact*, that results when the boundaries between subimages become visible. This last property is particularly important in comparisons with the other sinusoidal transforms. As Fig. 8.32(a) shows, the implicit n-point periodicity (see Section 4.6) of the DFT gives rise to boundary discontinuities that result in substantial high-frequency transform content. When the DFT transform coefficients are truncated or quantized, the Gibbs phenomenon[‡] causes the boundary points to take on erroneous values, which appear in an image as blocking artifact. That is, the boundaries between adjacent subimages become visible because the boundary pixels of the subimages assume the mean values of discontinuities formed at the boundary points [see Fig. 8.32(a)]. The DCT of Fig. 8.32(b) reduces this effect, because its implicit $2n$-point periodicity does not inherently produce boundary discontinuities.

Subimage size selection

Another significant factor affecting transform coding error and computational complexity is subimage size. In most applications, images are subdivided so that the correlation (redundancy) between adjacent subimages is reduced to some

[†] Ahmed et al. [1974] first noticed that the KLT basis images of a first order Markov image source closely resemble the DCT's basis images. As the correlation between adjacent pixels approaches one, the input dependent KLT basis images become identical to the input independent DCT basis images (Clarke [1985]).

[‡] This phenomenon, described in most electrical engineering texts on circuit analysis, occurs because the Fourier transform fails to converge uniformly at discontinuities. At discontinuities, Fourier expansions take the mean values.

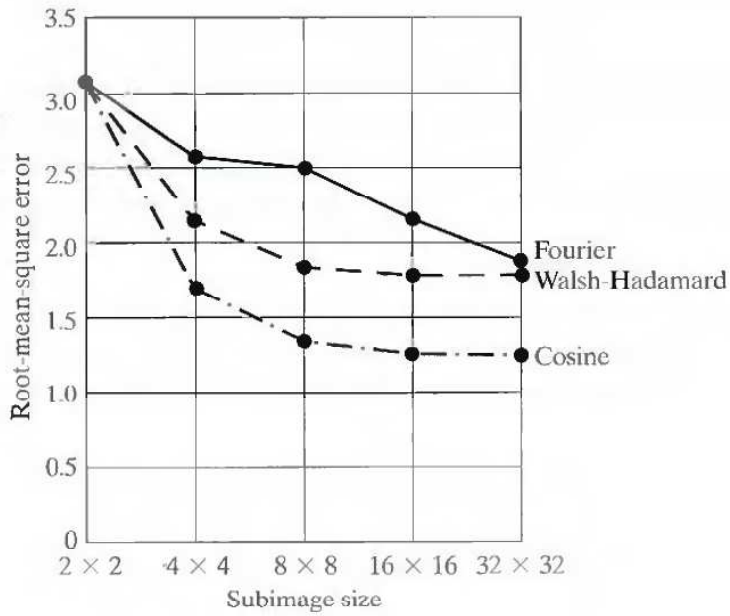
acceptable level and so that n is an integer power of 2 where, as before, n is the subimage dimension. The latter condition simplifies the computation of the subimage transforms (see the base-2 successive doubling methods in Section 4.6.6). In general, both the level of compression and computational complexity increase as the subimage size increases. The most popular subimage sizes are 8×8 and 16×16.

EXAMPLE 8.20: Effects of subimage size on transform coding.

■ Figure 8.33 illustrates graphically the impact of subimage size on transform coding reconstruction error. The data plotted were obtained by dividing the monochrome image of Fig. 8.23 into subimages of size $n \times n$, for $n = 2, 4, 8, 16$, and 32, computing the transform of each subimage, truncating 75% of the resulting coefficients, and taking the inverse transform of the truncated arrays. Note that the Hadamard and cosine curves flatten as the size of the subimage becomes greater than 8×8, whereas the Fourier reconstruction error decreases even more rapidly in this region. Extrapolation of these curves to larger values of n suggests that the Fourier reconstruction error will cross the Walsh-Hadamard curve and converge to the cosine result. In fact, this result is consistent with the theoretical and experimental findings reported by Netravali and Limb [1980] and by Pratt [1991] for a 2-D Markov image source.

All three curves intersect when 2×2 subimages are used. In this case, only one of the four coefficients (25%) of each transformed array was retained. The coefficient in all cases was the dc component, so the inverse transform simply replaced the four subimage pixels by their average value [see Eq. (4.2-22)]. This condition is evident in Fig. 8.34(d), which shows a zoomed portion of the 2×2 DCT result. Note that the blocking artifact that is prevalent in this result decreases as the subimage size increases to 4×4 and 8×8 in Figs. 8.34(e) and (f). Figure 8.34(c) shows a zoomed portion of the original image for reference. In addition, Figs. 8.34(a) and (b) facilitate comparison of these results to those of the preceding example. ■

FIGURE 8.33 Reconstruction error versus subimage size.



a b
c d
e f

FIGURE 8.34 Approximations of Fig. 8.23 using 25% of the DCT coefficients: (a) and (b) 8×8 subimage results; (c) zoomed original; (d) 2×2 result; (e) 4×4 result; and (f) 8×8 result.

Bit allocation

The reconstruction error associated with the truncated series expansion of Eq. (8.5-38) is a function of the number and relative importance of the transform coefficients that are discarded, as well as the precision that is used to represent the retained coefficients. In most transform coding systems, the retained coefficients are selected [that is, the masking function of Eq. (8.5-37) is constructed] on the basis of maximum variance, called *zonal coding*, or on the basis of maximum magnitude, called *threshold coding*. The overall process of truncating, quantizing, and coding the coefficients of a transformed subimage is commonly called *bit allocation*.

EXAMPLE 8.21: Bit allocation.

■ Figures 8.35(a) and (b) show two approximations of Fig. 8.23 in which 87.5% of the DCT coefficients of each 8×8 subimage were discarded. The first result was obtained via threshold coding by keeping the eight largest transform coefficients, and the second image was generated by using a zonal coding approach. In the latter case, each DCT coefficient was considered a random variable whose distribution could be computed over the ensemble of all transformed subimages. The 8 distributions of largest variance (12.5% of the 64 coefficients in the transformed 8×8 subimage) were located and used to determine the coordinates, u and v, of the coefficients, $T(u, v)$, that were retained for all subimages. Note that the threshold coding difference image of Fig. 8.35(c) contains far less error than the zonal coding result in Fig. 8.35(d). Figures 8.35(e) and (f) provide a closer view of a small portion of the reconstructed images in (a) and (c). ■

Zonal coding implementation Zonal coding is based on the information theory concept of viewing information as uncertainty. Therefore the transform coefficients of maximum variance carry the most image information and should be retained in the coding process. The variances themselves can be calculated directly from the ensemble of $(N/n)^2$ transformed subimage arrays, as in the preceding example, or based on an assumed image model (say, a Markov autocorrelation function). In either case, the zonal sampling process can be viewed, in accordance with Eq. (8.5-38), as multiplying each $T(u, v)$ by the corresponding element in a *zonal mask*, which is constructed by placing a 1 in the locations of maximum variance and a 0 in all other locations. Coefficients of maximum variance usually are located around the origin of an image transform, resulting in the typical zonal mask shown in Fig. 8.36(a).

The coefficients retained during the zonal sampling process must be quantized and coded, so zonal masks are sometimes depicted showing the number of bits used to code each coefficient [Fig. 8.36(b)]. In most cases, the coefficients are allocated the same number of bits, or some fixed number of bits is distributed among them unequally. In the first case, the coefficients generally are normalized by their standard deviations and uniformly quantized. In the second case, a quantizer, such as an optimal Lloyd-Max quantizer, is designed for each coefficient. To construct the required quantizers, the zeroth or dc coefficient normally is