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

Image compression, the art and science of reducing the amount of data required to represent an image, is one of the most useful and commercially successful technologies in the field of digital image processing. The number of images that are compressed and decompressed daily is staggering, and the compressions and decompressions themselves are virtually invisible to the user. Anyone who owns a digital camera, surfs the web, or watches the latest Hollywood movies on *Digital Video Disks* (DVDs) benefits from the algorithms and standards discussed in this chapter.

To better understand the need for compact image representations, consider the amount of data required to represent a two-hour *standard definition* (SD) *television* movie using $720 \times 480 \times 24$ bit pixel arrays. A digital movie (or *video*) is a sequence of *video frames* in which each frame is a full-color still image. Because video players must display the frames sequentially at rates near 30 fps (frames per second), SD digital video data must be accessed at

$$30\frac{\text{frames}}{\text{sec}} \times (720 \times 480)\frac{\text{pixels}}{\text{frame}} \times 3\frac{\text{bytes}}{\text{pixel}} = 31{,}104{,}000 \text{ bytes/sec}$$

and a two-hour movie consists of

$$31{,}104{,}000\frac{\text{bytes}}{\text{sec}} \times \left(60^2\right)\frac{\text{sec}}{\text{hr}} \times 2 \text{ hrs} \cong 2.24 \times 10^{11} \text{ bytes}$$

or 224 GB (gigabytes) of data. Twenty-seven 8.5 GB dual-layer DVDs (assuming conventional 12 cm disks) are needed to store it. To put a two-hour movie on a single DVD, each frame must be compressed—on average—by a factor of 26.3. The compression must be even higher for *high definition* (HD) *television*, where image resolutions reach 1920 × 1080 × 24 bits/image.

Web page images and high-resolution digital camera photos also are compressed routinely to save storage space and reduce transmission time. For example, residential Internet connections deliver data at speeds ranging from 56 Kbps (kilobits per second) via conventional phone lines to more than 12 Mbps (megabits per second) for broadband. The time required to transmit a small 128 × 128 × 24 bit full-color image over this range of speeds is from 7.0 to 0.03 seconds. Compression can reduce transmission time by a factor of 2 to 10 or more. In the same way, the number of uncompressed full-color images that an 8-megapixel digital camera can store on a 1-GB flash memory card [about forty-one 24 MB (megabyte) images] can be similarly increased. In addition to these applications, image compression plays an important role in many other areas, including televideo conferencing, remote sensing, document and medical imaging, and facsimile transmission (FAX). An increasing number of applications depend on the efficient manipulation, storage, and transmission of binary, gray-scale, and color images.

In this chapter, we introduce the theory and practice of digital image compression. We examine the most frequently used compression techniques and describe the industry standards that make them useful. The material is introductory in nature and applicable to both still image and video applications. The chapter concludes with an introduction to *digital image watermarking*, the process of inserting visible and invisible data (like copyright information) into images.

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. In this definition, *data* and *information* are not the same thing; data are the means by which information is conveyed. Because various amounts of data can be used to represent the same amount of information, representations that contain irrelevant or repeated information are said to contain *redundant data*. If we let b and b' denote the number of bits (or information-carrying units) in two representations of the same information, the *relative data redundancy* R of the representation with b bits is

$$R = 1 - \frac{1}{C} \tag{8.1-1}$$

where C, commonly called the *compression ratio*, is defined as

$$C = \frac{b}{b'} \tag{8.1-2}$$

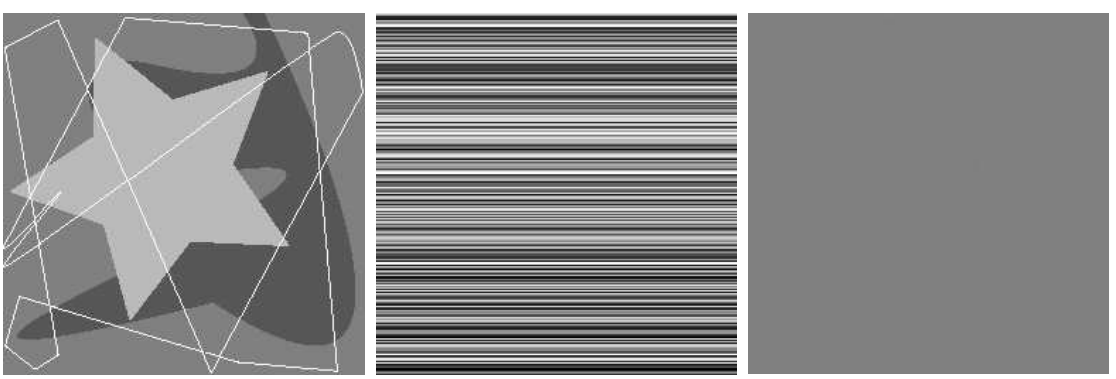
If $C = 10$ (sometimes written 10:1), for instance, the larger representation has 10 bits of data for every 1 bit of data in the smaller representation.

The corresponding relative data redundancy of the larger representation is 0.9 ($R = 0.9$), indicating that 90% of its data is redundant.

In the context of digital image compression, b in Eq. (8.1-2) usually is the number of bits needed to represent an image as a 2-D array of intensity values. The 2-D intensity arrays introduced in Section 2.4.2 are the preferred formats for human viewing and interpretation—and the standard by which all other representations are judged. When it comes to compact image representation, however, these formats are far from optimal. Two-dimensional intensity arrays suffer from three principal types of data redundancies that can be identified and exploited:

1. *Coding redundancy.* 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*. The 8-bit codes that are used to represent the intensities in most 2-D intensity arrays contain more bits than are needed to represent the intensities.
2. *Spatial* and *temporal redundancy*. Because the pixels of most 2-D intensity arrays are correlated spatially (i.e., each pixel is similar to or dependent on neighboring pixels), information is unnecessarily replicated in the representations of the correlated pixels. In a video sequence, temporally correlated pixels (i.e., those similar to or dependent on pixels in nearby frames) also duplicate information.
3. *Irrelevant information.* Most 2-D intensity arrays contain information that is ignored by the human visual system and/or extraneous to the intended use of the image. It is redundant in the sense that it is not used.

The computer-generated images in Figs. 8.1(a) through (c) exhibit each of these fundamental redundancies. As will be seen in the next three sections, compression is achieved when one or more redundancy is reduced or eliminated.



a b c

FIGURE 8.1 Computer generated 256 × 256 × 8 bit images with (a) coding redundancy, (b) spatial redundancy, and (c) irrelevant information. (Each was designed to demonstrate one principal redundancy but may exhibit others as well.)

8.1.1 Coding Redundancy

In Chapter 3, we developed techniques for image enhancement by histogram processing, assuming that the intensity values of an image are random quantities. In this section, we use a similar formulation to introduce optimal information coding.

Assume that a discrete random variable r_k in the interval $[0, L-1]$ is used to represent the intensities of an $M \times N$ image and that each r_k occurs with probability $p_r(r_k)$. As in Section 3.3,

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

where L is the number of intensity values, and n_k is the number of times that the kth intensity appears 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 intensity values is found by summing the products of the number of bits used to represent each intensity and the probability that the intensity occurs. The total number of bits required to represent an $M \times N$ image is MNL_{avg}. If the intensities are represented using a *natural* m-bit fixed-length code,[†] the right-hand side of Eq. (8.1-4) reduces to m bits. That is, $L_{\text{avg}} = m$ when m is substituted for $l(r_k)$. The constant m can be taken outside the summation, leaving only the sum of the $p_r(r_k)$ for $0 \le k \le L-1$, which, of course, equals 1.

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

■ The computer-generated image in Fig. 8.1(a) has the intensity distribution shown in the second column of Table 8.1. If a natural 8-bit binary code (denoted as code 1 in Table 8.1) is used to represent its 4 possible intensities, L_{avg}—the average number of bits for code 1—is 8 bits, because $l_1(r_k) = 8$ bits for all r_k.

TABLE 8.1 Example of variable-length coding.

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_{87} = 87$	0.25	01010111	8	01	2
$r_{128} = 128$	0.47	10000000	8	1	1
$r_{186} = 186$	0.25	11000100	8	000	3
$r_{255} = 255$	0.03	11111111	8	001	3
r_k for $k \neq 87, 128, 186, 255$	0	—	8	—	0

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

On the other hand, if the scheme designated as code 2 in Table 8.1 is used, the average length of the encoded pixels is, in accordance with Eq. (8.1-4),

$$L_{\text{avg}} = 0.25(2) + 0.47(1) + 0.25(3) + 0.03(3) = 1.81 \text{ bits}$$

The total number of bits needed to represent the entire image is $MNL_{\text{avg}} = 256 \times 256 \times 1.81$ or 118,621. From Eqs. (8.1-2) and (8.1-1), the resulting compression and corresponding relative redundancy are

$$C = \frac{256 \times 256 \times 8}{118{,}621} = \frac{8}{1.81} \approx 4.42$$

and

$$R = 1 - \frac{1}{4.42} = 0.774$$

respectively. Thus 77.4% of the data in the original 8-bit 2-D intensity array is redundant.

The compression achieved by code 2 results from assigning fewer bits to the more probable intensity values than to the less probable ones. In the resulting *variable-length code*, r_{128}—the image's most probable intensity—is assigned the 1-bit code word 1 [of length $l_2(r_{128}) = 1$], while r_{255}—its least probable occurring intensity—is assigned the 3-bit code word 001 [of length $l_2\left(r_{255}\right) = 3$]. Note that the best *fixed-length code* that can be assigned to the intensities of the image in Fig. 8.1(a) is the natural 2-bit counting sequence {00, 01, 10, 11}, but the resulting compression is only 8/2 or 4:1—about 10% less than the 4.42:1 compression of the variable-length code. ■

As the preceding example shows, *coding redundancy* is present when the codes assigned to a set of events (such as intensity values) do not take full advantage of the probabilities of the events. Coding redundancy is almost always present when the intensities of an image are represented using a natural binary code. The reason is that most images are composed of objects that have a regular and somewhat predictable morphology (shape) and reflectance, and are sampled so that the objects being depicted are much larger than the picture elements. The natural consequence is that, for most images, certain intensities are more probable than others (that is, the histograms of most images are not uniform). A natural binary encoding assigns the same number of bits to both the most and least probable values, failing to minimize Eq. (8.1-4) and resulting in coding redundancy.

8.1.2 Spatial and Temporal Redundancy

Consider the computer-generated collection of constant intensity lines in Fig. 8.1(b). In the corresponding 2-D intensity array:

1. All 256 intensities are equally probable. As Fig. 8.2 shows, the histogram of the image is uniform.

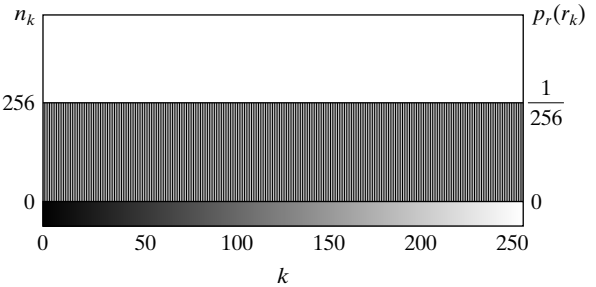


FIGURE 8.2 The intensity histogram of the image in Fig. 8.1(b).

2. Because the intensity of each line was selected randomly, its pixels are independent of one another in the vertical direction.
3. Because the pixels along each line are identical, they are maximally correlated (completely dependent on one another) in the horizontal direction.

The first observation tells us that the image in Fig. 8.1(b)—when represented as a conventional 8-bit intensity array—cannot be compressed by variable-length coding alone. Unlike the image of Fig. 8.1(a) (and Example 8.1), whose histogram was *not* uniform, a fixed-length 8-bit code in this case minimizes Eq. (8.1-4). Observations 2 and 3 reveal a significant spatial redundancy that can be eliminated, for instance, by representing the image in Fig. 8.1(b) as a sequence of *run-length pairs*, where each run-length pair specifies the start of a new intensity and the number of consecutive pixels that have that intensity. A run-length based representation compresses the original 2-D, 8-bit intensity array by $(256 \times 256 \times 8)/[(256 + 256) \times 8]$ or 128:1. Each 256-pixel line of the original representation is replaced by a single 8-bit intensity value and length 256 in the run-length representation.

In most images, pixels are correlated spatially (in both x and y) and in time (when the image is part of a video sequence). Because most pixel intensities can be predicted reasonably well from neighboring intensities, the information carried by a single pixel is small. Much of its visual contribution is redundant in the sense that it can be inferred from its neighbors. To reduce the redundancy associated with spatially and temporally correlated pixels, a 2-D intensity array must be transformed into a more efficient but usually "non-visual" representation. For example, run-lengths or the differences between adjacent pixels can be used. Transformations of this type are called *mappings*. A mapping is said to be *reversible* if the pixels of the original 2-D intensity array can be reconstructed without error from the transformed data set; otherwise the mapping is said to be *irreversible*.

8.1.3 Irrelevant Information

One of the simplest ways to compress a set of data is to remove superfluous data from the set. In the context of digital image compression, information that is ignored by the human visual system or is extraneous to the intended use of an image are obvious candidates for omission. Thus, the computer-generated image in Fig. 8.1(c), because it appears to be a homogeneous field of gray, can

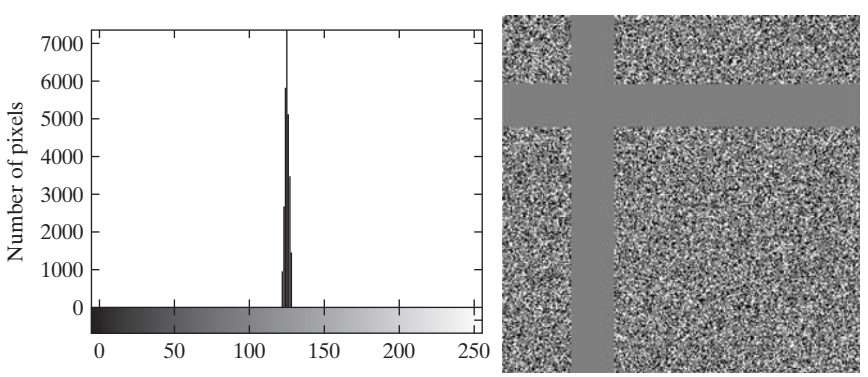
be represented by its average intensity alone—a single 8-bit value. The original $256 \times 256 \times 8$ bit intensity array is reduced to a single byte; and the resulting compression is $(256 \times 256 \times 8)/8$ or 65,536:1. Of course, the original $256 \times 256 \times 8$ bit image must be recreated to view and/or analyze it—but there would be little or no perceived decrease in reconstructed image quality.

Figure 8.3(a) shows the histogram of the image in Fig. 8.1(c). Note that there are several intensity values (intensities 125 through 131) actually present. The human visual system averages these intensities, perceives only the average value, and ignores the small changes in intensity that are present in this case. Figure 8.3(b), a histogram equalized version of the image in Fig. 8.1(c), makes the intensity changes visible *and* reveals two previously undetected regions of constant intensity—one oriented vertically and the other horizontally. If the image in Fig. 8.1(c) is represented by its average value alone, this "invisible" structure (i.e., the constant intensity regions) and the random intensity variations surrounding them—real information—is lost. Whether or not this information should be preserved is application dependent. If the information is important, as it might be in a medical application (like digital X-ray archival), it should not be omitted; otherwise, the information is redundant and can be excluded for the sake of compression performance.

We conclude the section by noting that the redundancy examined here is fundamentally different from the redundancies discussed in Sections 8.1.1 and 8.1.2. Its elimination is possible because the information itself is not essential for normal visual processing and/or the intended use of the image. Because its omission results in a loss of quantitative information, its removal is commonly referred to as *quantization*. This terminology is consistent with normal use of the word, which generally means the mapping of a broad range of input values to a limited number of output values (see Section 2.4). Because information is lost, quantization is an irreversible operation.

8.1.4 Measuring Image Information

In the previous sections, we introduced several ways to reduce the amount of data used to represent an image. The question that naturally arises is this: How



a b

FIGURE 8.3
(a) Histogram of the image in Fig. 8.1(c) and (b) a histogram equalized version of the image.

Consult the book Web site for a brief review of information and probability theory.

few bits are actually needed to represent the information in an image? That is, is there a minimum amount of data that is sufficient to describe an image without losing information? *Information theory* provides the mathematical framework to answer this and related questions. Its fundamental premise 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 with probability $P(E)$ is said to contain

$$I(E) = \log \frac{1}{P(E)} = -\log P(E) \tag{8.1-5}$$

units of information. If $P(E) = 1$ (that is, the event always occurs), $I(E) = 0$ and no information is attributed to it. Because no uncertainty is associated with the event, no information would be transferred by communicating that the event has occurred [it *always* occurs if $P(E) = 1$].

The base of the logarithm in Eq. (8.1-5) 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 unit of information is the *bit*. Note that if $P(E) = \frac{1}{2}$, $I(E) = -\log_2 \frac{1}{2}$, 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 is flipping a coin and communicating the result.

Given a source of statistically independent random events from a discrete set of possible events $\{a_1, a_2, \ldots, a_J\}$ with associated probabilities $\{P(a_1), P(a_2), \ldots, P(a_J)\}$, the average information per source output, called the *entropy* of the source, is

$$H = -\sum_{j=1}^{J} P\left(a_j\right) \log P\left(a_j\right) \tag{8.1-6}$$

The a_j in this equation are called *source symbols*. Because they are statistically independent, the source itself is called a *zero-memory source*.

If an image is considered to be the output of an imaginary zero-memory "intensity source," we can use the histogram of the observed image to estimate the symbol probabilities of the source. Then the intensity source's entropy becomes

Equation (8.1-6) is for zero-memory sources with J source symbols; Eq. (8.1-7) uses probability estimates for the $L - 1$ intensity values in an image.

$$\tilde{H} = -\sum_{k=0}^{L-1} p_r\left(r_k\right) \log_2 p_r\left(r_k\right) \tag{8.1-7}$$

where variables L, r_k, and $p_r(r_k)$ are as defined in Sections 8.1.1 and 3.3. Because the base 2 logarithm is used, Eq. (8.1-7) is the average information per intensity output of the imaginary intensity source in bits. It is not possible to code the *intensity values* of the imaginary source (and thus the sample image) with fewer than $\tilde{H}$ bits/pixel.

EXAMPLE 8.2: Image entropy estimates.

■ The entropy of the image in Fig. 8.1(a) can be estimated by substituting the intensity probabilities from Table 8.1 into Eq. (8.1-7):

$$\begin{aligned}\tilde{H} &= -[0.25 \log_2 0.25 + 0.47 \log_2 0.47 + 0.25 \log_2 0.25 + 0.03 \log_2 0.03] \\ &\approx -[0.25(-2) + 0.47(-1.09) + 0.25(-2) + 0.03(-5.06)] \\ &\approx 1.6614 \text{ bits/pixel}\end{aligned}$$

In a similar manner, the entropies of the images in Fig. 8.1(b) and (c) can be shown to be 8 bits/pixel and 1.566 bits/pixel, respectively. Note that the image in Fig. 8.1(a) appears to have the most visual information, but has almost the lowest computed entropy—1.66 bits/pixel. The image in Fig. 8.1(b) has almost five times the entropy of the image in (a), but appears to have about the same (or less) visual information; and the image in Fig. 8.1(c), which seems to have little or no information, has almost the same entropy as the image in (a). The obvious conclusion is that the amount of entropy and thus information in an image is far from intuitive. ■

Shannon's first theorem

Recall that the variable-length code in Example 8.1 was able to represent the intensities of the image in Fig. 8.1(a) using only 1.81 bits/pixel. Although this is higher than the 1.6614 bits/pixel entropy estimate from Example 8.2, *Shannon's first theorem*—also called the *noiseless coding theorem* (Shannon [1948])—assures us that the image in Fig. 8.1(a) can be represented with as few as 1.6614 bits/pixel. To prove it in a general way, Shannon looked at representing groups of n consecutive source symbols with a single code word (rather than one code word per source symbol) and showed that

$$\lim_{n \to \infty} \left[\frac{L_{\text{avg},n}}{n} \right] = H \tag{8.1-8}$$

where $L_{\text{avg},n}$ is the average number of code symbols required to represent all n-symbol groups. In the proof, he defined the *nth extension* of a zero-memory source to be the hypothetical source that produces n-symbol blocks[†] using the symbols of the original source; and computed $L_{\text{avg},n}$ by applying Eq. (8.1-4) to the code words used to represent the n-symbol blocks. Equation (8.1-8) tells us that $L_{\text{avg},n}/n$ can be made arbitrarily close to H by encoding infinitely long extensions of the single-symbol source. That is, it is possible to represent the output of a zero-memory source with an average of H information units per source symbol.

[†]The output of the nth extension is an n-tuple of symbols from the underlying *single-symbol* source. It was considered a *block random variable* in which the probability of each n-tuple is the product of the probabilities of its individual symbols. The entropy of the nth extension is then n times the entropy of the single-symbol source from which it is derived.

If we now return to the idea that an image is a "sample" of the intensity source that produced it, a block of n source symbols corresponds to a group of n adjacent pixels. To construct a variable-length code for n-pixel blocks, the relative frequencies of the blocks must be computed. But the nth extension of a hypothetical intensity source with 256 intensity values has 256^n possible n-pixel blocks. Even in the simple case of $n = 2$, a 65,536 element histogram and up to 65,536 variable-length code words must be generated. For $n = 3$, as many as 16,777,216 code words are needed. So even for small values of n, computational complexity limits the usefulness of the extension coding approach in practice.

Finally, we note that although Eq. (8.1-7) provides a lower bound on the compression that can be achieved when coding statistically independent pixels directly, it breaks down when the pixels of an image are correlated. Blocks of correlated pixels can be coded with fewer average bits per pixel than the equation predicts. Rather than using source extensions, less correlated descriptors (like intensity run-lengths) are normally selected and coded without extension. This was the approach used to compress Fig. 8.1(b) in Section 8.1.2. When the output of a source of information depends on a finite number of preceding outputs, the source is called a *Markov* or *finite memory source*.

8.1.5 Fidelity Criteria

In Section 8.1.3, it was noted that the removal of "irrelevant visual" information involves a loss of real or quantitative image information. Because information is lost, a means of quantifying the nature of the loss is needed. Two types of criteria can be used for such an assessment: (1) objective fidelity criteria and (2) subjective fidelity criteria.

When information loss can be expressed as a mathematical function of the input and output of a compression process, it is said to be based on an *objective fidelity criterion*. An example is the root-mean-square (rms) error between two images. Let $f(x,y)$ be an input image and $\hat{f}(x,y)$ be an 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 $\hat{f}(x,y)$ is

$$e(x,y) = \hat{f}(x,y) - f(x,y) \tag{8.1-9}$$

so that the total error between the two images is

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

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

$$e_{\text{rms}} = \left[\frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y) - f(x,y)\right]^2\right]^{1/2} \tag{8.1-10}$$

If $\hat{f}(x,y)$ is considered [by a simple rearrangement of the terms in Eq. (8.1-9)] to be the sum of the original image $f(x,y)$ and an error or "noise" signal $e(x,y)$, the *mean-square signal-to-noise ratio* of the output image, denoted SNR_{ms}, can be defined as in Section 5.8:

$$\text{SNR}_{\text{ms}} = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y) - f(x,y)\right]^2} \tag{8.1-11}$$

The rms value of the signal-to-noise ratio, denoted SNR_{rms}, is obtained by taking the square root of Eq. (8.1-11).

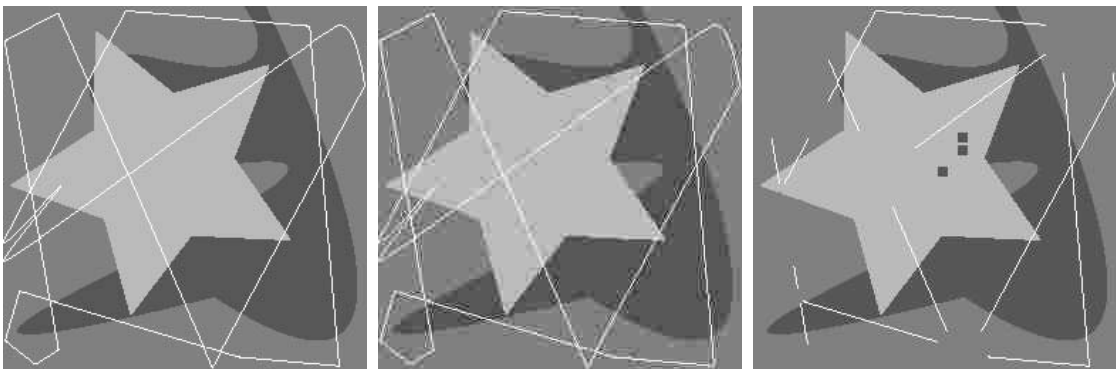
While objective fidelity criteria offer a simple and convenient way to evaluate information loss, decompressed images are ultimately viewed by humans. So, measuring image quality by the subjective evaluations of people is often more appropriate. This can be done by presenting a decompressed image to a 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 8.2 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 based on *subjective fidelity criteria.*

EXAMPLE 8.3: Image quality comparisons.

■ Figure 8.4 shows three different approximations of the image in Fig. 8.1(a). Using Eq. (8.1-10) with Fig. 8.1(a) for $f(x,y)$ and the images in Figs. 8.4(a) through (c) as $\hat{f}(x,y)$, the computed rms errors are 5.17, 15.67, and 14.17 intensity levels, respectively. In terms of rms error—an objective fidelity criterion—the three images in Fig. 8.4 are ranked in order of decreasing quality as {(a), (c), (b)}.

TABLE 8.2 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.



a b c

FIGURE 8.4 Three approximations of the image in Fig. 8.1(a).

Figures 8.4(a) and (b) are typical of images that have been compressed and subsequently reconstructed. Both retain the essential information of the original image—like the spatial and intensity characteristics of its objects. And their rms errors correspond roughly to perceived quality. Figure 8.4(a), which is practically as good as the original image, has the lowest rms error, while Fig. 8.4(b) has more error but noticeable degradation at the boundaries between objects. This is exactly as one would expect.

Figure 8.4(c) is an artificially generated image that demonstrates the limitations of objective fidelity criteria. Note that the image is missing large sections of several important lines (i.e., visual information), and has small dark squares (i.e., artifacts) in the upper right quadrant. The visual content of the image is misleading and certainly not as accurate as the image in (b), but it has less rms error—14.17 versus 15.67 intensity values. A subjective evaluation of the three images using Table 8.2 might yield an *excellent* rating for (a), a *passable* or *marginal* rating for (b), and an *inferior* of *unusable* rating for (c). The rms error measure, on the other hand, ranks (c) ahead of (b). ■

8.1.6 Image Compression Models

As Fig. 8.5 shows, an image compression system is composed of two distinct functional components: an *encoder* and a *decoder*. The encoder performs compression, and the decoder performs the complementary operation of decompression. Both operations can be performed in software, as is the case in Web browsers and many commercial image editing programs, or in a combination of hardware and firmware, as in commercial DVD players. A *codec* is a device or program that is capable of both encoding and decoding.

Here, the notation $f(x, \ldots)$ is used to denote both $f(x, y)$ and $f(x, y, t)$.

Input image $f(x, \ldots)$ is fed into the encoder, which creates a compressed representation of the input. This representation is stored for later use, or transmitted for storage and use at a remote location. When the compressed representation is presented to its complementary decoder, a reconstructed output image $\hat{f}(x, \ldots)$ is generated. In still-image applications, the encoded input and decoder output are $f(x, y)$ and $\hat{f}(x, y)$, respectively; in video applications, they

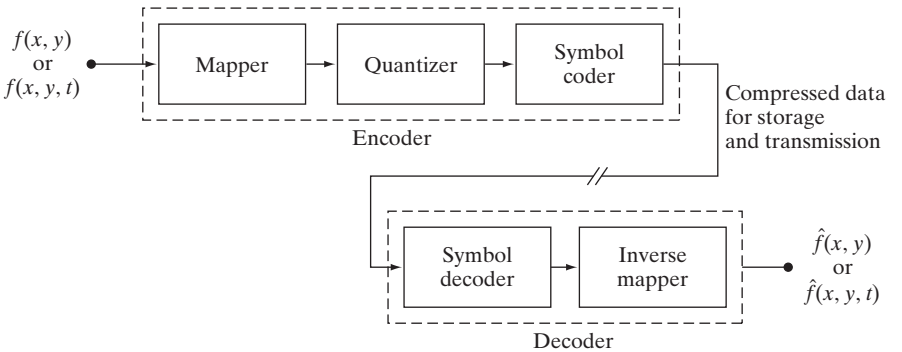


FIGURE 8.5 Functional block diagram of a general image compression system.

are $f(x, y, t)$ and $\hat{f}(x, y, t)$, where discrete parameter t specifies time. In general, $\hat{f}(x, \ldots)$ may or may not be an exact replica of $f(x, \ldots)$. If it is, the compression system is called *error free*, *lossless*, or *information preserving*. If not, the reconstructed output image is distorted and the compression system is referred to as *lossy*.

The encoding or compression process

The encoder of Fig. 8.5 is designed to remove the redundancies described in Sections 8.1.1–8.1.3 through a series of three independent operations. In the first stage of the encoding process, a *mapper* transforms $f(x, \ldots)$ into a (usually nonvisual) format designed to reduce spatial and temporal redundancy. This operation generally is reversible and may or may not reduce directly the amount of data required to represent the image. Run-length coding (see Sections 8.1.2 and 8.2.5) is an example of a mapping that normally yields compression in the first step of the encoding process. The mapping of an image into a set of less correlated transform coefficients (see Section 8.2.8) is an example of the opposite case (the coefficients must be further processed to achieve compression). In video applications, the mapper uses previous (and in some cases future) video frames to facilitate the removal of temporal redundancy.

The *quantizer* in Fig. 8.5 reduces the accuracy of the mapper's output in accordance with a pre-established fidelity criterion. The goal is to keep irrelevant information out of the compressed representation. As noted in Section 8.1.3, this operation is irreversible. It must be omitted when error-free compression is desired. In video applications, the *bit rate* of the encoded output is often measured (in bits/second) and used to adjust the operation of the quantizer so that a predetermined average output rate is maintained. Thus, the visual quality of the output can vary from frame to frame as a function of image content.

In the third and final stage of the encoding process, the *symbol coder* of Fig. 8.5 generates a fixed- or variable-length code to represent the quantizer output and maps the output in accordance with the code. In many cases, a variable-length code is used. The shortest code words are assigned to the most frequently occurring quantizer output values—thus minimizing coding redundancy. This operation is reversible. Upon its completion, the input image has been processed for the removal of each of the three redundancies described in Sections 8.1.1 to 8.1.3.

The decoding or decompression process

The decoder of Fig. 8.5 contains only two components: a *symbol decoder* and an *inverse mapper*. They perform, in reverse order, the inverse operations of the encoder's symbol encoder and mapper. Because quantization results in irreversible information loss, an inverse quantizer block is not included in the general decoder model. In video applications, decoded output frames are maintained in an internal frame store (not shown) and used to reinsert the temporal redundancy that was removed at the encoder.

8.1.7 Image Formats, Containers, and Compression Standards

In the context of digital imaging, an *image file format* is a standard way to organize and store image data. It defines how the data is arranged and the type of compression—if any—that is used. An *image container* is similar to a file format but handles multiple types of image data. Image *compression standards*, on the other hand, define procedures for compressing and decompressing images—that is, for reducing the amount of data needed to represent an image. These standards are the underpinning of the widespread acceptance of image compression technology.

Figure 8.6 lists the most important image compression standards, file formats, and containers in use today, grouped by the type of image handled. The entries in black are international standards sanctioned by the *International Standards Organization* (ISO), the *International Electrotechnical Commission* (IEC), and/or the *International Telecommunications Union* (ITU-T)—a *United Nations* (UN) organization that was once called the *Consultative Committee of the International Telephone and Telegraph* (CCITT). Two video compression standards, VC-1 by the *Society of Motion Pictures and Television Engineers* (SMPTE) and AVS by the Chinese *Ministry of Information Industry* (MII), are

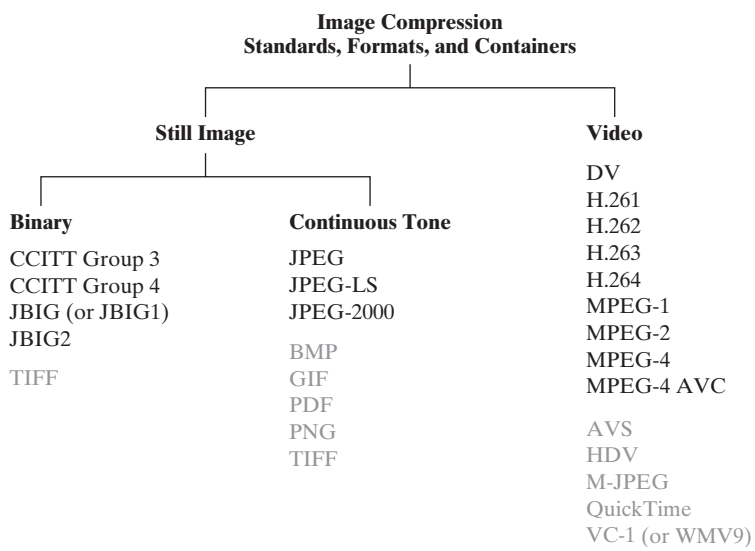


FIGURE 8.6 Some popular image compression standards, file formats, and containers. Internationally sanctioned entries are shown in black; all others are grayed.

also included. Note that they are shown in gray, which is used in Fig. 8.6 to denote entries that are not sanctioned by an international standards organization.

Tables 8.3 and 8.4 summarize the standards, formats, and containers listed in Fig. 8.6. Responsible organizations, targeted applications, and key compression methods are identified. The compression methods themselves are the subject of the next section. In both tables, forward references to the relevant subsections of Section 8.2 are enclosed in square brackets.

TABLE 8.3 Internationally sanctioned image compression standards. The numbers in brackets refer to sections in this chapter.

Name	Organization	Description
Bi-Level Still Images		
CCITT Group 3	ITU-T	Designed as a facsimile (FAX) method for transmitting binary documents over telephone lines. Supports 1-D and 2-D run-length [8.2.5] and Huffman [8.2.1] coding.
CCITT Group 4	ITU-T	A simplified and streamlined version of the CCITT Group 3 standard supporting 2-D run-length coding only.
JBIG *or* JBIG1	ISO/IEC/ ITU-T	A *Joint Bi-level Image Experts Group* standard for progressive, lossless compression of bi-level images. Continuous-tone images of up to 6 bits/pixel can be coded on a bit-plane basis [8.2.7]. Context sensitive arithmetic coding [8.2.3] is used and an initial low resolution version of the image can be gradually enhanced with additional compressed data.
JBIG2	ISO/IEC/ ITU-T	A follow-on to JBIG1 for bi-level images in desktop, Internet, and FAX applications. The compression method used is content based, with dictionary based methods [8.2.6] for text and halftone regions, and Huffman [8.2.1] or arithmetic coding [8.2.3] for other image content. It can be lossy or lossless.
Continuous-Tone Still Images		
JPEG	ISO/IEC/ ITU-T	A *Joint Photographic Experts Group* standard for images of photographic quality. Its lossy *baseline coding system* (most commonly implemented) uses quantized discrete cosine transforms (DCT) on 8×8 image blocks [8.2.8], Huffman [8.2.1], and run-length [8.2.5] coding. It is one of the most popular methods for compressing images on the Internet.
JPEG-LS	ISO/IEC/ ITU-T	A lossless to near-lossless standard for continuous tone images based on adaptive prediction [8.2.9], context modeling [8.2.3], and Golomb coding [8.2.2].
JPEG-2000	ISO/IEC/ ITU-T	A follow-on to JPEG for increased compression of photographic quality images. Arithmetic coding [8.2.3] and quantized discrete wavelet transforms (DWT) [8.2.10] are used. The compression can be lossy or lossless.

(*Continues*)

TABLE 8.3
(*Continued*)

Name	Organization	Description
Video		
DV	IEC	*Digital Video*. A video standard tailored to home and semiprofessional video production applications and equipment—like electronic news gathering and camcorders. Frames are compressed independently for uncomplicated editing using a DCT-based approach [8.2.8] similar to JPEG.
H.261	ITU-T	A two-way videoconferencing standard for ISDN (*integrated services digital network*) lines. It supports non-interlaced 352 × 288 and 176 × 144 resolution images, called CIF (*Common Intermediate Format*) and QCIF (*Quarter CIF*), respectively. A DCT-based compression approach [8.2.8] similar to JPEG is used, with frame-to-frame prediction differencing [8.2.9] to reduce temporal redundancy. A block-based technique is used to compensate for motion between frames.
H.262	ITU-T	See MPEG-2 below.
H.263	ITU-T	An enhanced version of H.261 designed for ordinary telephone modems (i.e., 28.8 Kb/s) with additional resolutions: SQCIF (*Sub-Quarter* CIF 128 × 96), 4CIF (704 × 576), and 16CIF (1408 × 512).
H.264	ITU-T	An extension of H.261–H.263 for videoconferencing, Internet streaming, and television broadcasting. It supports prediction differences within frames [8.2.9], variable block size integer transforms (rather than the DCT), and context adaptive arithmetic coding [8.2.3].
MPEG-1	ISO/IEC	A *Motion Pictures Expert Group* standard for CD-ROM applications with non-interlaced video at up to 1.5 Mb/s. It is similar to H.261 but frame predictions can be based on the previous frame, next frame, or an interpolation of both. It is supported by almost all computers and DVD players.
MPEG-2	ISO/IEC	An extension of MPEG-1 designed for DVDs with transfer rates to 15 Mb/s. Supports interlaced video and HDTV. It is the most successful video standard to date.
MPEG-4	ISO/IEC	An extension of MPEG-2 that supports variable block sizes and prediction differencing [8.2.9] within frames.
MPEG-4 AVC	ISO/IEC	MPEG-4 Part 10 *Advanced Video Coding* (AVC). Identical to H.264 above.

TABLE 8.4 Popular image compression standards, file formats, and containers, not included in Table 8.3.

Name	Organization	Description
Continuous-Tone Still Images		
BMP	Microsoft	*Windows Bitmap*. A file format used mainly for simple uncompressed images.
GIF	CompuServe	*Graphic Interchange Format*. A file format that uses lossless LZW coding [8.2.4] for 1- through 8-bit images. It is frequently used to make small animations and short low resolution films for the World Wide Web.
PDF	Adobe Systems	*Portable Document Format*. A format for representing 2-D documents in a device and resolution independent way. It can function as a container for JPEG, JPEG 2000, CCITT, and other compressed images. Some PDF versions have become ISO standards.
PNG	*World Wide Web Consortium* (W3C)	*Portable Network Graphics*. A file format that losslessly compresses full color images with transparency (up to 48 bits/pixel) by coding the difference between each pixel's value and a predicted value based on past pixels [8.2.9].
TIFF	Aldus	*Tagged Image File Format*. A flexible file format supporting a variety of image compression standards, including JPEG, JPEG-LS, JPEG-2000, JBIG2, and others.
Video		
AVS	MII	*Audio-Video Standard*. Similar to H.264 but uses exponential Golomb coding [8.2.2]. Developed in China.
HDV	Company consortium	*High Definition Video*. An extension of DV for HD television that uses MPEG-2 like compression, including temporal redundancy removal by prediction differencing [8.2.9].
M-JPEG	Various companies	*Motion JPEG*. A compression format in which each frame is compressed independently using JPEG.
Quick-Time	Apple Computer	A media container supporting DV, H.261, H.262, H.264, MPEG-1, MPEG-2, MPEG-4, and other video compression formats.
VC-1 WMV9	SMPTE Microsoft	The most used video format on the Internet. Adopted for HD and *Blu-ray* high-definition DVDs. It is similar to H.264/AVC, using an integer DCT with varying block sizes [8.2.8 and 8.2.9] and context dependent variable-length code tables [8.2.1]—but no predictions within frames.

8.2 Some Basic Compression Methods

In this section, we describe the principal lossy and error-free compression methods in use today. Our focus is on methods that have proven useful in mainstream binary, continuous-tone still images, and video compression standards. The standards themselves are used to demonstrate the methods presented.

8.2.1 Huffman Coding

With reference to Tables 8.3 and 8.4, Huffman codes are used in

- CCITT
- JBIG2
- JPEG
- MPEG-1,2,4
- H.261, H.262, H.263, H.264

and other compression standards.

One of the most popular techniques 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 Shannon's first theorem (see Section 8.1.4), 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*. In practice, the source symbols may be either the intensities of an image or the output of an intensity mapping operation (pixel differences, run lengths, and so on).

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 8.7 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 also are 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, are the symbols 0 and 1. As Fig. 8.8 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 appended to each to distinguish them from each other. This operation is then repeated for

FIGURE 8.7 Huffman source reductions.

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				

Original source			Source reduction			
Symbol	Probability	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				

FIGURE 8.8 Huffman code assignment procedure.

each reduced source until the original source is reached. The final code appears at the far left in Fig. 8.8. The average length of this code is

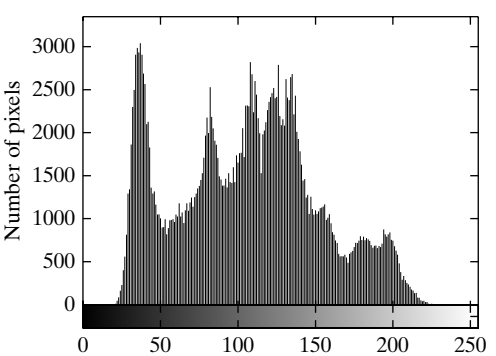
$$L_{\text{avg}} = (0.4)(1) + (0.3)(2) + (0.1)(3) + (0.1)(4) + (0.06)(5) + (0.04)(5)$$
$$= 2.2 \text{ bits/pixel}$$

and the entropy of the source is 2.14 bits/symbol.

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 error-free 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. 8.8, 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 $a_3a_1a_2a_2a_6$.

EXAMPLE 8.4: Huffman coding.

■ The 512 × 512 × 8 bit monochrome image in Fig. 8.9(a) has the intensity histogram shown in Fig. 8.9(b). Because the intensities are not equally probable,



a b

FIGURE 8.9 (a) A 512 × 512 8-bit image, and (b) its histogram.

a MATLAB implementation of Huffman's procedure was used to encode them with 7.428 bits/pixel—including the Huffman code table that is required to reconstruct the original 8-bit image intensities. The compressed representation exceeds the estimated entropy of the image [7.3838 bits/pixel from Eq. (8.1-7)] by $512^2 \times (7.428 - 7.3838)$ or 11,587 bits—about 0.6%. The resulting compression ratio and corresponding relative redundancy are $C = 8/7.428 = 1.077$ and $R = 1 - (1/1.077) = 0.0715$, respectively. Thus 7.15% of the original 8-bit fixed-length intensity representation was removed as coding redundancy. ■

When a large number of symbols is to be coded, the construction of an optimal Huffman code is a nontrivial task. For the general case of J source symbols, J symbol probabilities, $J - 2$ source reductions, and $J - 2$ code assignments are required. When source symbol probabilities can be estimated in advance, "near optimal" coding can be achieved with pre-computed Huffman codes. Several popular image compression standards, including the JPEG and MPEG standards discussed in Sections 8.2.8 and 8.2.9, specify default Huffman coding tables that have been pre-computed based on experimental data.

8.2.2 Golomb Coding

With reference to Tables 8.3 and 8.4, Golomb codes are used in

- JPEG-LS
- AVS

compression.

In this section we consider the coding of nonnegative integer inputs with exponentially decaying probability distributions. Inputs of this type can be optimally encoded (in the sense of Shannon's first theorem) using a family of codes that are computationally simpler than Huffman codes. The codes themselves were first proposed for the representation of nonnegative run lengths (Golomb [1966]). In the discussion that follows, the notation $\lfloor x \rfloor$ denotes the largest integer less than or equal to x, $\lceil x \rceil$ means the smallest integer greater than or equal to x, and $x \bmod y$ is the remainder of x divided by y.

Given a nonnegative integer n and a positive integer *divisor* $m > 0$, the *Golomb code* of n with respect to m, denoted $G_m(n)$, is a combination of the unary code of *quotient* $\lfloor n/m \rfloor$ and the binary representation of *remainder* $n \bmod m$. $G_m(n)$ is constructed as follows:

Step 1. Form the unary code of quotient $\lfloor n/m \rfloor$. (The *unary code* of an integer q is defined as q 1s followed by a 0.)

Step 2. Let $k = \lceil \log_2 m \rceil$, $c = 2^k - m$, $r = n \bmod m$, and compute truncated remainder r' such that

$$r' = \begin{cases} r \text{ truncated to } k - 1 \text{ bits} & 0 \le r < c \\ r + c \text{ truncated to } k \text{ bits} & \text{otherwise} \end{cases} \tag{8.2-1}$$

Step 3. Concatenate the results of steps 1 and 2.

To compute $G_4(9)$, for example, begin by determining the unary code of the quotient $\lfloor 9/4 \rfloor = \lfloor 2.25 \rfloor = 2$, which is 110 (the result of step 1). Then let $k = \lceil \log_2 4 \rceil = 2$, $c = 2^2 - 4 = 0$, and $r = 9 \bmod 4$, which in binary is $1001 \bmod 0100$ or 0001. In accordance with Eq. (8.2-1), r' is then r (i.e., 0001) truncated to 2 bits, which is 01 (the result of step 2). Finally, concatenate 110 from step 1 and 01 from step 2 to get 11001, which is $G_4(9)$.

For the special case of $m = 2^k$, $c = 0$ and $r' = r = n \bmod m$ truncated to k bits in Eq. (8.2-1) for all n. The divisions required to generate the resulting Golomb codes become binary shift operations and the computationally simpler codes are called *Golomb-Rice* or *Rice codes* (Rice [1975]). Columns 2, 3, and 4 of Table 8.5 list the G_1, G_2, and G_4 codes of the first ten nonnegative integers. Because m is a power of 2 in each case (i.e., $1 = 2^0$, $2 = 2^1$, and $4 = 2^2$), they are the first three Golomb-Rice codes as well. Moreover, G_1 is the unary code of the nonnegative integers because $\lfloor n/1 \rfloor = n$ and $n \bmod 1 = 0$ for all n.

Keeping in mind that Golomb codes can only be used to represent nonnegative integers and that there are many Golomb codes to choose from, a key step in their effective application is the selection of divisor m. When the integers to be represented are *geometrically* distributed with *probability mass function* (PMF)[†]

$$P(n) = (1 - \rho)\rho^n \tag{8.2-2}$$

The discrete probability distribution defined by the PMF in Eq. (8.2-2) is called the *geometric probability distribution*. Its continuous counterpart is the *exponential distribution*.

for some $0 < \rho < 1$, Golomb codes can be shown to be optimal—in the sense that $G_m(n)$ provides the shortest average code length of all uniquely decipherable codes—when (Gallager and Voorhis [1975])

$$m = \left\lceil \frac{\log_2(1 + \rho)}{\log_2(1/\rho)} \right\rceil \tag{8.2-3}$$

Figure 8.10(a) plots Eq. (8.2-2) for three values of ρ and illustrates graphically the symbol probabilities that Golomb codes handle well (that is, code efficiently). As is shown in the figure, small integers are much more probable than large ones.

The graphical representation of a PMF is a histogram.

Because the probabilities of the intensities in an image [see, for example, the histogram of Fig. 8.9(b)] are unlikely to match the probabilities specified in Eq. (8.2-2) and shown in Fig. 8.10(a), Golomb codes are seldom used for the coding of intensities. When intensity differences are to be coded, however, the

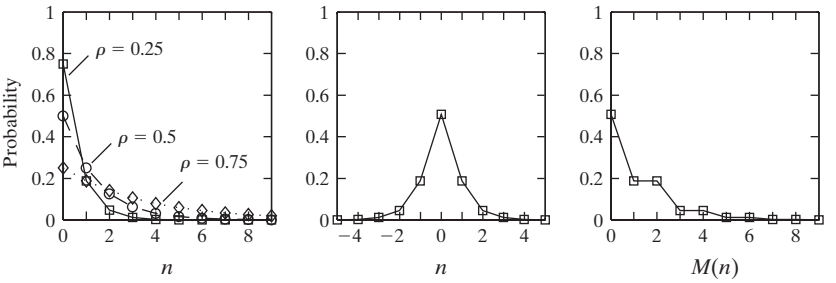
TABLE 8.5
Several Golomb codes for the integers 0 – 9.

n	$G_1(n)$	$G_2(n)$	$G_4(n)$	$G^0_{\exp}(n)$
0	0	00	000	0
1	10	01	001	100
2	110	100	010	101
3	1110	101	011	11000
4	11110	1100	1000	11001
5	111110	1101	1001	11010
6	1111110	11100	1010	11011
7	11111110	11101	1011	1110000
8	111111110	111100	11000	1110001
9	1111111110	111101	11001	1110010

[†]A *probability mass function* (PMF) is a function that defines the probability that a discrete random variable is exactly equal to some value. A PMF differs from a PDF in that a PDF's values are not probabilities; rather, the integral of a PDF over a specified interval is a probability.

a b c

FIGURE 8.10 (a) Three one-sided geometric distributions from Eq. (8.2-2); (b) a two-sided exponentially decaying distribution; and (c) a reordered version of (b) using Eq. (8.2-4).



probabilities of the resulting "difference values" (see Section 8.2.9)—with the notable exception of the negative differences—often resemble those of Eq. (8.2-2) and Fig. 8.10(a). To handle negative differences in Golomb coding, which can only represent nonnegative integers, a mapping like

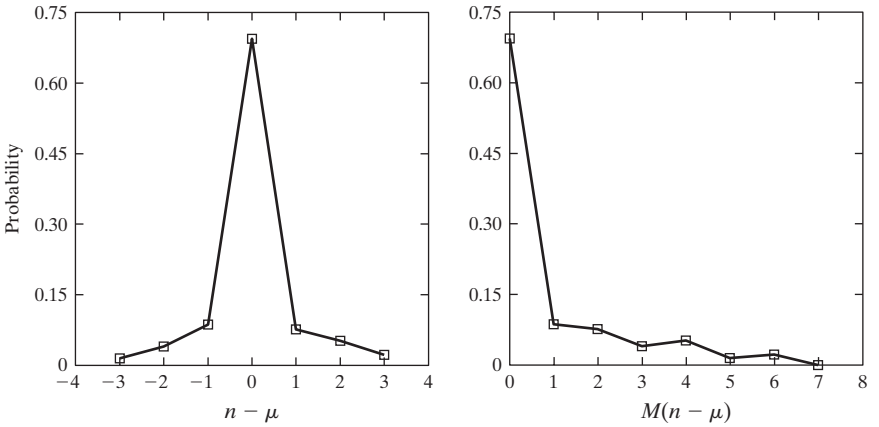
$$M(n) = \begin{cases} 2n & n \geq 0 \\ 2|n| - 1 & n < 0 \end{cases} \tag{8.2-4}$$

typically is used. Using this mapping, for example, the two-sided PMF shown in Fig. 8.10(b) can be transformed into the one-sided PMF in Fig. 8.10(c). Its integers are reordered, alternating the negative and positive integers so that the negative integers are mapped into the odd positive integer positions. If $P(n)$ is two-sided and centered at zero, $P(M(n))$ will be one-sided. The mapped integers, $M(n)$, can then be efficiently encoded using an appropriate Golomb-Rice code (Weinberger et al. [1996]).

EXAMPLE 8.5: Golomb-Rice coding.

■ Consider again the image from Fig. 8.1(c) and note that its histogram—see Fig. 8.3(a)—is similar to the two-sided distribution in Fig. 8.10(b) above. If we let n be some nonnegative integer intensity in the image, where $0 \leq n \leq 255$, and μ be the mean intensity, $P(n - \mu)$ is the two-sided distribution shown in Fig. 8.11(a). This plot was generated by normalizing the histogram in Fig. 8.3(a) by the total number of pixels in the image and shifting the normalized values to the left by 128 (which in effect subtracts the mean intensity from the image). In accordance with Eq. (8.2-4), $P(M(n - \mu))$ is then the one-sided distribution shown in Fig. 8.11(b). If the reordered intensity values are Golomb coded using a MATLAB implementation of code G_1 in column 2 of Table 8.5, the encoded representation is 4.5 times smaller than the original image (i.e., $C = 4.5$). The G_1 code realizes 4.5/5.1 or 88% of the theoretical compression possible with variable-length coding. (Based on the entropy calculated in Example 8.2, the maximum possible compression ratio through variable-length coding is $C = 8/1.566 \approx 5.1$.) Moreover, Golomb coding achieves 96% of the compression provided by a MATLAB implementation of Huffman's approach—and doesn't require the computation of a custom Huffman coding table.

Now consider the image in Fig. 8.9(a). If its intensities are Golomb coded using the same G_1 code as above, $C = 0.0922$. That is, there is *data expansion*.



a b

FIGURE 8.11 (a) The probability distribution of the image in Fig. 8.1(c) after subtracting the mean intensity from each pixel, and (b) a mapped version of (a) using Eq. (8.2-4).

When C is less than 1 in Eq. (8.1-2), there is data expansion.

This is due to the fact that the probabilities of the intensities of the image in Fig. 8.9(a) are much different than the probabilities defined in Eq. (8.2-2). In a similar manner, Huffman codes can produce data expansion when used to encode symbols whose probabilities are different from those for which the code was computed. In practice, the further you depart from the input probability assumptions for which a code is designed, the greater the risk of poor compression performance and data expansion. ■

To conclude our coverage of Golomb codes, we note that Column 5 of Table 8.5 contains the first 10 codes of the zeroth order *exponential-Golomb code*, denoted $G^0_{\exp}(n)$. Exponential-Golomb codes are useful for the encoding of run lengths, because both short and long runs are encoded efficiently. An order-k exponential-Golomb code $G^k_{\exp}(n)$ is computed as follows:

Step 1. Find an integer $i \geq 0$ such that

$$\sum_{j=0}^{i-1} 2^{j+k} \leq n < \sum_{j=0}^{i} 2^{j+k} \tag{8.2-5}$$

and form the unary code of i. If $k = 0, i = \lfloor \log_2(n + 1) \rfloor$ and the code is also known as the *Elias gamma code*.

Step 2. Truncate the binary representation of

$$n - \sum_{j=0}^{i-1} 2^{j+k} \tag{8.2-6}$$

to $k + i$ least significant bits.

Step 3. Concatenate the results of steps 1 and 2.

To find $G^0_{\text{exp}}(8)$, for example, we let $i = \lfloor \log_2 9 \rfloor$ or 3 in step 1 because $k = 0$. Equation (8.2-5) is then satisfied because

$$\sum_{j=0}^{3-1} 2^{j+0} \le 8 < \sum_{j=0}^{3} 2^{j+0}$$

$$\sum_{j=0}^{2} 2^{j} \le 8 < \sum_{j=0}^{3} 2^{j}$$

$$2^0 + 2^1 + 2^2 \le 8 < 2^0 + 2^1 + 2^2 + 2^3$$

$$7 \le 8 < 15$$

The unary code of 3 is 1110 and Eq. (8.2-6) of step 2 yields

$$8 - \sum_{j=0}^{3-1} 2^{j+0} = 8 - \sum_{j=0}^{2} 2^{j} = 8 - (2^0 + 2^1 + 2^2) = 8 - 7 = 1 = 0001$$

which when truncated to its $3 + 0$ least significant bits becomes 001. The concatenation of the results from steps 1 and 2 then yields 1110001. Note that this is the entry in column 4 of Table 8.5 for $n = 8$. Finally, we note that like the Huffman codes of the last section, the Golomb codes of Table 8.5 are variable-length, instantaneous uniquely decodable block codes.

8.2.3 Arithmetic Coding

With reference to Tables 8.3 and 8.4, arithmetic coding is used in

- JBIG1
- JBIG2
- JPEG-2000
- H.264
- MPEG-4 AVC

and other compression standards.

Unlike the variable-length codes of the previous two sections, *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 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 established by Shannon's first theorem of Section 8.1.4.

Figure 8.12 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 subdivided initially 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. 8.12

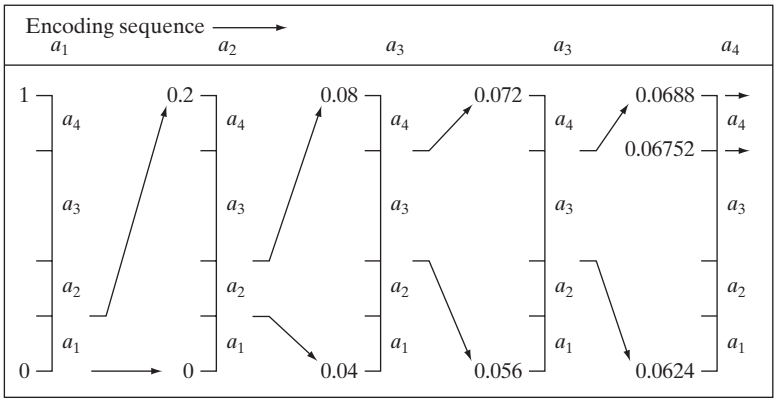


FIGURE 8.12
Arithmetic coding procedure.

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.

[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.

In the arithmetically-coded message of Fig. 8.12, three decimal digits are used to represent the five-symbol message. This translates into 0.6 decimal digits per source symbol and compares favorably with the entropy of the source, which, from Eq. (8.1-6), is 0.58 decimal digits per source symbol. As the length of the sequence being coded increases, the resulting arithmetic code approaches the bound established by Shannon's first 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.

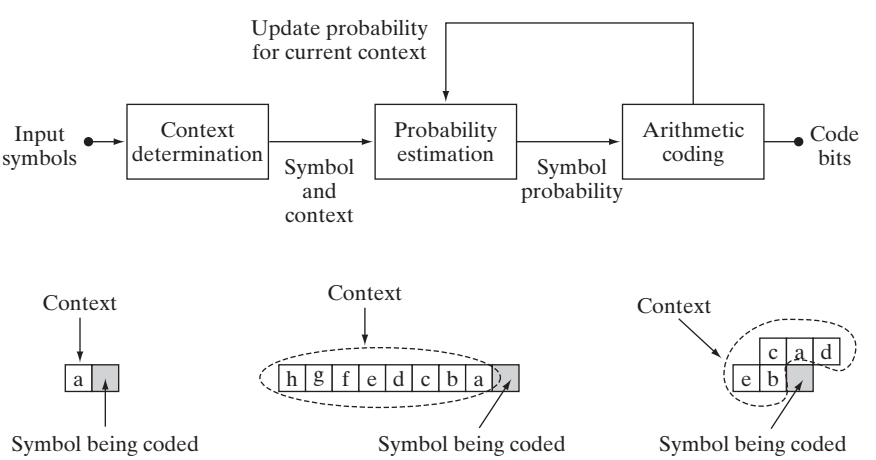
Adaptive context dependent probability estimates

With accurate input symbol *probability models*, that is, models that provide the true probabilities of the symbols being coded, arithmetic coders are near optimal in the sense of minimizing the average number of code symbols required to represent the symbols being coded. Like in both Huffman and Golomb coding, however, inaccurate probability models can lead to non-optimal results. A simple way to improve the accuracy of the probabilities employed is to use an adaptive, context dependent probability model. *Adaptive* probability models update symbol probabilities as symbols are coded or become known. Thus, the probabilities adapt to the local statistics of the symbols being coded. *Context dependent* models provide probabilities that are based on a predefined neighborhood of pixels—called the *context*—around the symbols being coded. Normally, a *causal context*—one limited to symbols that have already been coded—is used. Both the Q-coder (Pennebaker et al. [1988]) and MQ-coder (ISO/IEC [2000]), two well-known arithmetic coding techniques that have been incorporated into the JBIG, JPEG-2000, and other important image compression standards, use probability models that are both adaptive and context dependent. The Q-coder dynamically updates symbol probabilities during the interval renormalizations that are part of the arithmetic coding process. Adaptive context dependent models also have been used in Golomb coding—for example, in the JPEG-LS compression standard.

Figure 8.13(a) diagrams the steps involved in adaptive, context-dependent arithmetic coding of *binary* source symbols. Arithmetic coding often is used when binary symbols are to be coded. As each symbol (or bit) begins the coding process, its context is formed in the *Context determination* block of Fig. 8.13(a). Figures 8.13(b) through (d) show three possible contexts that can be used: (1) the immediately preceding symbol, (2) a group of preceding symbols, and (3) some number of preceding symbols plus symbols on the previous scan line. For the three cases shown, the *Probability estimation* block must manage 2^1 (or 2), 2^8 (or 256), and 2^5 (or 32) contexts and their associated probabilities. For instance, if the context in Fig. 8.13(b) is used, conditional probabilities

a
b c d

FIGURE 8.13
(a) An adaptive, context-based arithmetic coding approach (often used for binary source symbols). (b)–(d) Three possible context models.



$P(0|a = 0)$ (the probability that the symbol being coded is a 0 given that the preceding symbol is a 0), $P(1|a = 0)$, $P(0|a = 1)$, and $P(1|a = 1)$ must be tracked. The appropriate probabilities are then passed to the *Arithmetic coding* block as a function of the current context and drive the generation of the arithmetically coded output sequence in accordance with the process illustrated in Fig. 8.12. The probabilities associated with the context involved in the current coding step are then updated to reflect the fact that another symbol within that context has been processed.

Finally, we note that a variety of arithmetic coding techniques are protected by United States patents (and may in addition be protected in other jurisdictions). Because of these patents and the possibility of unfavorable monetary judgments for their infringement, most implementations of the JPEG compression standard, which contains options for both Huffman and arithmetic coding, typically support Huffman coding alone.

8.2.4 LZW Coding

The techniques covered in the previous sections are focused on the removal of coding redundancy. In this section, we consider an error-free compression approach that also addresses spatial redundancies in an image. The technique, called *Lempel-Ziv-Welch* (*LZW*) *coding*, assigns fixed-length code words to variable length sequences of source symbols. Recall from Section 8.1.4 that Shannon used the idea of coding sequences of source symbols, rather than individual source symbols, in the proof of his first theorem. A key feature of LZW coding is that it requires no a priori knowledge of the probability of occurrence of the symbols to be encoded. Despite the fact that until recently it was protected under a United States patent, LZW compression has been integrated into a variety of mainstream imaging file formats, including GIF, TIFF, and PDF. The PNG format was created to get around LZW licensing requirements.

With reference to Tables 8.3 and 8.4, LZW coding is used in the

- GIF
- TIFF
- PDF

formats, but not in any of the internationally sanctioned compression standards.

EXAMPLE 8.6: LZW coding Fig. 8.9(a).

■ Consider again the 512 × 512, 8-bit image from Fig. 8.9(a). Using Adobe Photoshop, an uncompressed TIFF version of this image requires 286,740 bytes of disk space—262,144 bytes for the 512 × 512 8-bit pixels plus 24,596 bytes of overhead. Using TIFF's LZW compression option, however, the resulting file is 224,420 bytes. The compression ratio is $C = 1.28$. Recall that for the Huffman encoded representation of Fig. 8.9(a) in Example 8.4, $C = 1.077$. The additional compression realized by the LZW approach is due the removal of some of the image's spatial redundancy. ■

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 intensities 0, 1, 2, ..., 255. As the encoder sequentially examines image pixels, intensity 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 intensity 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. Clearly, the size of the dictionary is an important system parameter. If it is too small, the detection of matching intensity-level sequences will be less likely; if it is too large, the size of the code words will adversely affect compression performance.

EXAMPLE 8.7: LZW coding.

■ 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 initially are unused.

The image is encoded by processing its pixels in a left-to-right, top-to-bottom manner. Each successive intensity 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 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 intensity 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 intensity 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. ■

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		

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 8.20) 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. Alternatively, the least used dictionary entries can be tracked and replaced when necessary.

8.2.5 Run-Length Coding

As was noted in Section 8.1.2, images with repeating intensities along their rows (or columns) can often be compressed by representing runs of identical intensities as *run-length pairs*, where each run-length pair specifies the start of a new intensity and the number of consecutive pixels that have that intensity. The technique, referred to as *run-length encoding* (RLE), was developed in the 1950s and became, along with its 2-D extensions, the standard compression approach in facsimile (FAX) coding. Compression is achieved by eliminating a simple form of spatial redundancy—groups of identical intensities. When there are few (or no) runs of identical pixels, run-length encoding results in data expansion.

With reference to Tables 8.3 and 8.4, the coding of run-lengths is used in

- CCITT
- JBIG2
- JPEG
- M-JPEG
- MPEG-1,2,4
- BMP

and other compression standards and file formats.

EXAMPLE 8.8: RLE in the BMP file format.

■ The BMP file format uses a form of run-length encoding in which image data is represented in two different modes: encoded and absolute—and either mode can occur anywhere in the image. In *encoded* mode, a two byte RLE representation is used. The first byte specifies the number of consecutive pixels that have the color index contained in the second byte. The 8-bit color index selects the run's intensity (color or gray value) from a table of 256 possible intensities.

In *absolute* mode, the first byte is 0 and the second byte signals one of four possible conditions, as shown in Table 8.8. When the second byte is 0 or 1, the end of a line or the end of the image has been reached. If it is 2, the next two bytes contain unsigned horizontal and vertical offsets to a new spatial position (and pixel) in the image. If the second byte is between 3 and 255, it specifies the number of uncompressed pixels that follow—with each subsequent byte containing the color index of one pixel. The total number of bytes must be aligned on a 16-bit word boundary.

Note that due to differences in overhead, the uncompressed BMP file is smaller than the uncompressed TIFF file in Example 8.7.

An uncompressed BMP file (saved using Photoshop) of the $512 \times 512 \times 8$ bit image shown in Fig. 8.9(a) requires 263,244 bytes of memory. Compressed using BMP's RLE option, the file expands to 267,706 bytes—and the compression ratio is $C = 0.98$. There are not enough equal intensity runs to make run-length compression effective; a small amount of expansion occurs. For the image in Fig. 8.1(c), however, the BMP RLE option results in a compression ratio $C = 1.35$. ■

Run-length encoding is particularly effective when compressing binary images. Because there are only two possible intensities (black and white), adjacent pixels are more likely to be identical. In addition, each image row can be represented by a sequence of lengths only—rather than length-intensity pairs as was used in Example 8.8. The basic idea is to code each contiguous group (i.e., run) of 0s or 1s 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 conventions 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 encoding is in itself an effective method of compressing binary images, additional compression can be achieved by variable-length coding the run lengths themselves. The black and white run lengths can 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

TABLE 8.8 BMP absolute coding mode options. In this mode, the first byte of the BMP pair is 0.

Second Byte Value	Condition
0	End of line
1	End of image
2	Move to a new position
3–255	Specify pixels individually

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. (8.1-6). A similar argument holds for the entropy of the white runs, denoted H_1. The approximate run-length entropy of the image is then

$$H_{RL} = \frac{H_0 + H_1}{L_0 + L_1} \tag{8.2-7}$$

where the variables L_0 and L_1 denote the average values of black and white run lengths, respectively. Equation (8.2-7) 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 of the oldest and most widely used image compression standards are the CCITT Group 3 and 4 standards for binary image compression. Although they have been used in a variety of computer applications, they were originally designed as facsimile (FAX) coding methods for transmitting documents over telephone networks. The Group 3 standard uses a 1-D run-length coding technique in which the last $K - 1$ lines of each group of K lines (for $K = 2$ or 4) can be optionally coded in a 2-D manner. The Group 4 standard is a simplified or streamlined version of the Group 3 standard in which only 2-D coding is allowed. Both standards use the same 2-D coding approach, which is two-dimensional in the sense that information from the previous line is used to encode the current line. Both 1-D and 2-D coding are discussed next.

One-dimensional CCITT compression

In the 1-D CCITT Group 3 compression standard, each line of an image[†] is encoded as a series of variable-length Huffman code words that represent the run lengths of alternating white and black runs in a left-to-right scan of the line. The compression method employed is commonly referred to as *Modified Huffman* (MH) coding. The code words themselves are of two types, which the standard refers to as *terminating codes* and *makeup codes*. If run length r is less than 63, a terminating code from Table A.1 in Appendix A is used to represent it. Note that the standard specifies different terminating codes for black and white runs. If $r > 63$, two codes are used—a makeup code for quotient $\lfloor r/64 \rfloor$ and terminating code for remainder $r \bmod 64$. Makeup codes are listed in Table A.2 and may or may not depend on the intensity (black or white) of the run being coded. If $\lfloor r/64 \rfloor < 1792$, separate black and white run makeup codes are specified; otherwise, makeup codes are independent of run intensity. The standard requires that each line begin with a white run-length code word, which may in fact be 00110101, the code for a white run of length zero. Finally, a unique end-of-line (EOL) code word 000000000001 is used to terminate each line, as well as to signal the first line of each new image. The end of a sequence of images is indicated by six consecutive EOLs.

Recall from Section 8.2.2 that the notation $\lfloor x \rfloor$ denotes the largest integer less than or equal to x.

[†]In the standard, images are referred to as *pages* and sequences of images are called *documents*.

Two-dimensional CCITT compression

The 2-D compression approach adopted for both the CCITT Group 3 and 4 standards is a line-by-line method in which the position of each black-to-white or white-to-black run transition is coded with respect to the position of a *reference element* a_0 that is situated on the current *coding line*. The previously coded line is called the *reference line*; the reference line for the first line of each new image is an imaginary white line. The 2-D coding technique that is used is called *Relative Element Address Designate* (READ) coding. In the Group 3 standard, one or three READ coded lines are allowed between successive MH coded lines and the technique is called *Modified READ* (MR) coding. In the Group 4 standard, a greater number of READ coded lines are allowed and the method is called *Modified Modified READ* (MMR) coding. As was previously noted, the coding is two-dimensional in the sense that information from the previous line is used to encode the current line. Two-dimensional transforms are not involved.

Figure 8.14 shows the basic 2-D coding process for a single scan line. Note that the initial steps of the procedure are directed at locating several key *changing elements*: a_0, a_1, a_2, b_1, and b_2. A changing element is defined by the standard as a pixel whose value is different from that of the previous pixel on the same line. The most important changing element is a_0 (the reference element), which is either set to the location of an imaginary white changing element to the left of the first pixel of each new coding line or determined from the previous coding mode. Coding modes are discussed in the following paragraph. After a_0 is located, a_1 is identified as the location of the next changing element to the right of a_0 on the current coding line, a_2 as the next changing element to the right of a_1 on the coding line, b_1 as the changing element of the opposite value (of a_0) and to the right of a_0 on the reference (or previous) line, and b_2 as the next changing element to the right of b_1 on the reference line. If any of these changing elements are not detected, they are set to the location of an imaginary pixel to the right of the last pixel on the appropriate line. Figure 8.15 provides two illustrations of the general relationships between the various changing elements.

After identification of the current reference element and associated changing elements, two simple tests are performed to select one of three possible coding modes: *pass mode*, *vertical mode*, or *horizontal mode*. The initial test, which corresponds to the first branch point in the flowchart in Fig. 8.14, compares the location of b_2 to that of a_1. The second test, which corresponds to the second branch point in Fig. 8.14, computes the distance (in pixels) between the locations of a_1 and b_1 and compares it against 3. Depending on the outcome of these tests, one of the three outlined coding blocks of Fig. 8.14 is entered and the appropriate coding procedure is executed. A new reference element is then established, as per the flowchart, in preparation for the next coding iteration.

Table 8.9 defines the specific codes utilized for each of the three possible coding modes. In pass mode, which specifically excludes the case in which b_2 is directly above a_1, only the pass mode code word 0001 is needed. As Fig. 8.15(a) shows, this mode identifies white or black reference line runs that do not overlap

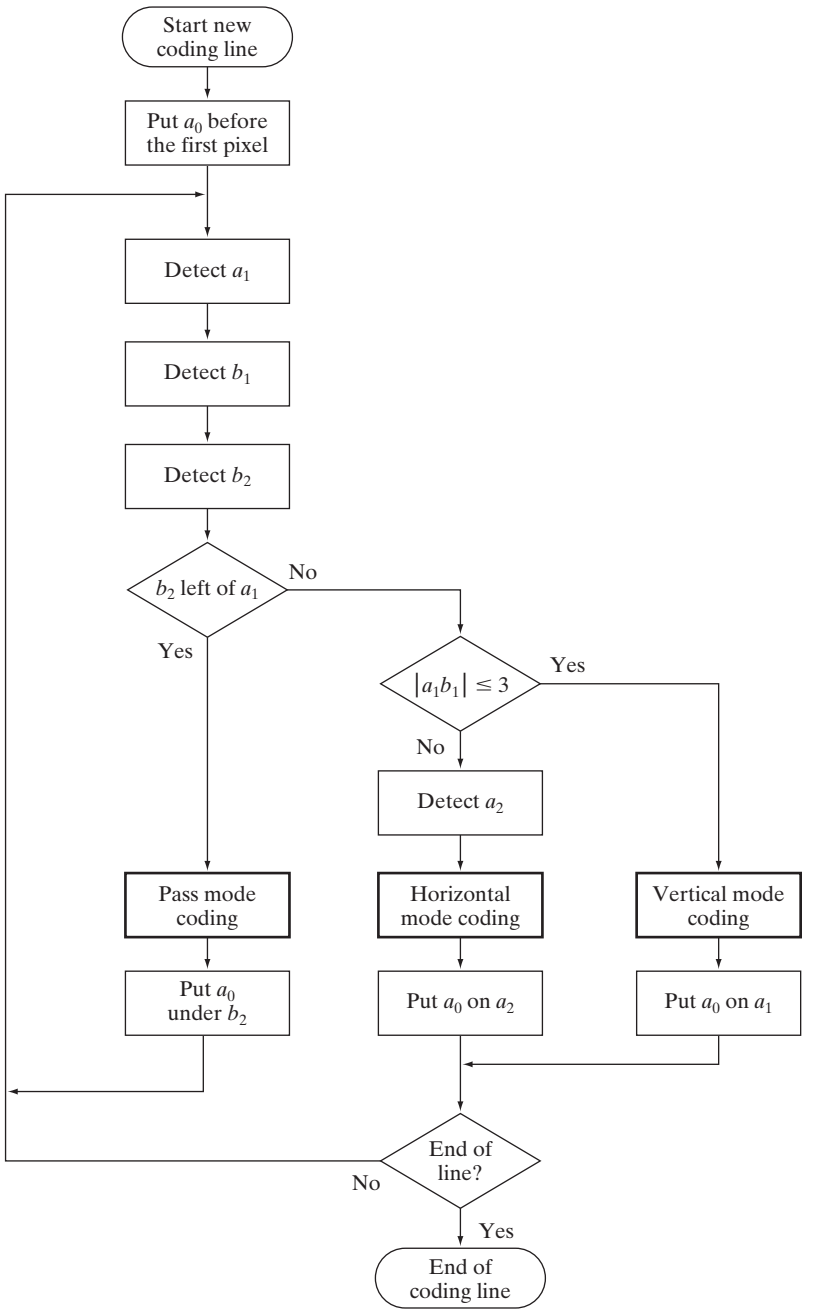


FIGURE 8.14
CCITT 2-D READ coding procedure. The notation $|a_1b_1|$ denotes the absolute value of the distance between changing elements a_1 and b_1.

TABLE 8.9
CCITT two-dimensional code table.

Mode	Code Word
Pass	0001
Horizontal	$001 + M(a_0a_1) + M(a_1a_2)$
Vertical	
a_1 below b_1	1
a_1 one to the right of b_1	011
a_1 two to the right of b_1	000011
a_1 three to the right of b_1	0000011
a_1 one to the left of b_1	010
a_1 two to the left of b_1	000010
a_1 three to the left of b_1	0000010
Extension	0000001xxx

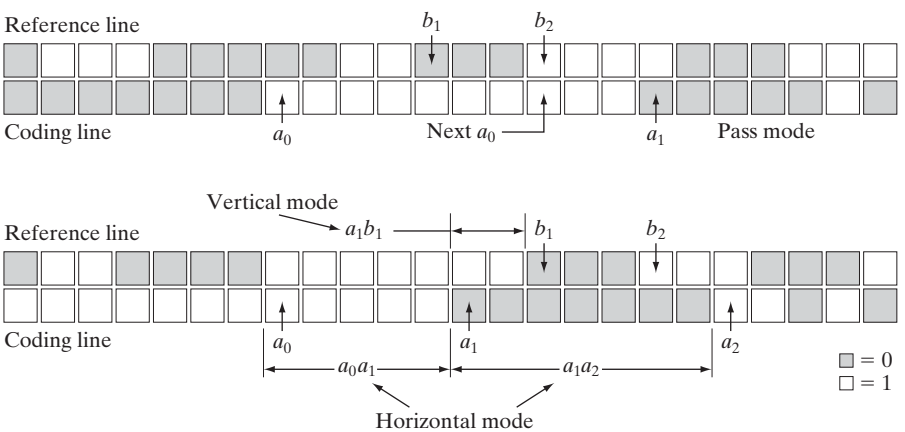
the current white or black coding line runs. In horizontal coding mode, the distances from a_0 to a_1 and a_1 to a_2 must be coded in accordance with the termination and makeup codes of Tables A.1 and A.2 of Appendix A and then appended to the horizontal mode code word 001. This is indicated in Table 8.9 by the notation $001 + M(a_0a_1) + M(a_1a_2)$, where a_0a_1 and a_1a_2 denote the distances from a_0 to a_1 and a_1 to a_2, respectively. Finally, in vertical coding mode, one of six special variable-length codes is assigned to the distance between a_1 and b_1. Figure 8.15(b) illustrates the parameters involved in both horizontal and vertical mode coding. The extension mode code word at the bottom of Table 8.9 is used to enter an optional facsimile coding mode. For example, the 0000001111 code is used to initiate an uncompressed mode of transmission.

EXAMPLE 8.9: CCITT vertical mode coding example.

■ Although Fig. 8.15(b) is annotated with the parameters for both horizontal and vertical mode coding (to facilitate the discussion above), the depicted pattern of black and white pixels is a case for vertical mode coding. That is, because b_2 is to the right of a_1, the first (or pass mode) test in Fig. 8.14 fails. The second test, which determines whether the vertical or horizontal coding mode

a
b

FIGURE 8.15
CCITT (a) pass mode and (b) horizontal and vertical mode coding parameters.



is entered, indicates that vertical mode coding should be used, because the distance from a_1 to b_1 is less than 3. In accordance with Table 8.9, the appropriate code word is 000010, implying that a_1 is two pixels left of b_1. In preparation for the next coding iteration, a_0 is moved to the location of a_1. ■

EXAMPLE 8.10: CCITT compression example.

■ Figure 8.16(a) is a 300 dpi scan of a 7 × 9.25 inch book page displayed at about 1/3 scale. Note that about half of the page contains text, around 9% is occupied by a halftone image, and the rest is white space. A section of the page is enlarged in Fig. 8.16(b). Keep in mind that we are dealing with a binary image; the illusion of gray tones is created, as was described in Section 4.5.4, by the halftoning process used in printing. If the binary pixels of the image in Fig. 8.16(a) are stored in groups of 8 pixels per byte, the 1952 × 2697 bit scanned image, commonly called a *document*, requires 658,068 bytes. An uncompressed PDF file of the document (created in Photoshop) requires 663,445 bytes. CCITT Group 3 compression reduces the file to 123,497 bytes—resulting in a compression ratio $C = 5.37$; CCITT Group 4 compression reduces the file to 110,456 bytes, increasing the compression ratio to about 6. ■

Do not confuse the PDF used here, which stands for *Portable Document Format*, with the PDF used in previous sections and chapters for probability density function.

8.2.6 Symbol-Based Coding

In *symbol-* or *token-based* coding, an image is represented as a collection of frequently occurring sub-images, called *symbols*. Each such symbol is stored in a *symbol dictionary* and the image is coded as a set of triplets $\{(x_1, y_1, t_1), (x_2, y_2, t_2), \ldots\}$, where each (x_i, y_i) pair specifies the location of a symbol in

With reference to Tables 8.3 and 8.4, symbol-based coding is used in

- JBIG2

compression.

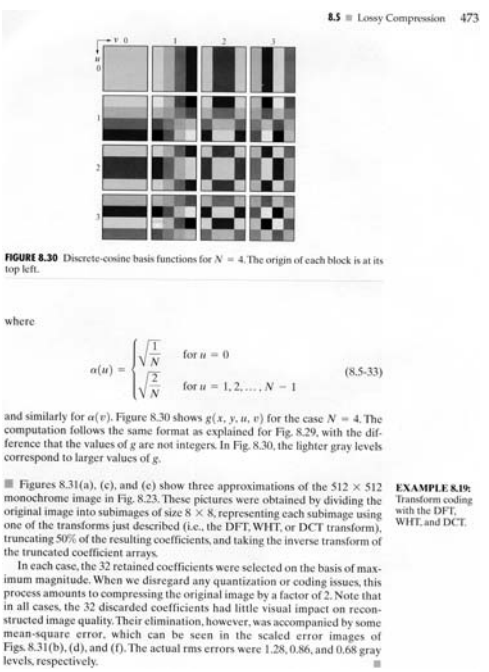

8.5 ■ Lossy Compression 473

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

where

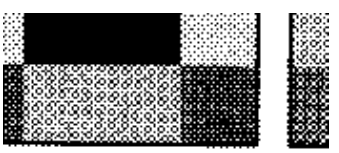
$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{for } u = 0 \\ \sqrt{\frac{2}{N}} & \text{for } u = 1, 2, \ldots, N-1 \end{cases} \quad (8.5\text{-}33)$$

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. ■



r $N = 4$. Th

a b

FIGURE 8.16 A binary scan of a book page: (a) scaled to show the general page content; (b) scaled to show the binary pixels used in dithering.

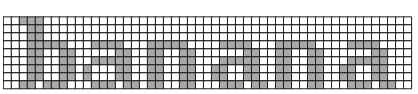
the image and *token* t_i is the address of the symbol or sub-image in the dictionary. That is, each triplet represents an instance of a dictionary symbol in the image. Storing repeated symbols only once can compress images significantly—particularly in document storage and retrieval applications, where the symbols are often character bitmaps that are repeated many times.

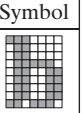
Consider the simple bilevel image in Fig. 8.17(a). It contains the single word, *banana*, which is composed of three unique symbols: a *b*, three *a*'s, and two *n*'s. Assuming that the *b* is the first symbol identified in the coding process, its 9×7 bitmap is stored in location 0 of the symbol dictionary. As Fig. 8.17(b) shows, the token identifying the *b* bitmap is 0. Thus, the first triplet in the encoded image's representation [see Fig. 8.17(c)] is $(0, 2, 0)$—indicating that the upper-left corner (an arbitrary convention) of the rectangular bitmap representing the *b* symbol is to be placed at location $(0, 2)$ in the decoded image. After the bitmaps for the *a* and *n* symbols have been identified and added to the dictionary, the remainder of the image can be encoded with five additional triplets. As long as the six triplets required to locate the symbols in the image, together with the three bitmaps required to define them, are smaller than the original image, compression occurs. In this case, the starting image has $9 \times 51 \times 1$ or 459 bits and, assuming that each triplet is composed of 3 bytes, the compressed representation has $(6 \times 3 \times 8) + [(9 \times 7) + (6 \times 7) + (6 \times 6)]$ or 285 bits; the resulting compression ratio $C = 1.61$. To decode the symbol-based representation in Fig. 8.17(c), you simply read the bitmaps of the symbols specified in the triplets from the symbol dictionary and place them at the spatial coordinates specified in each triplet.

Symbol-based compression was proposed in the early 1970s (Ascher and Nagy [1974]), but has become practical only recently. Advances in symbol matching algorithms (see Chapter 12) and increased CPU computer processing speeds have made it possible both to select dictionary symbols and to find where they occur in an image in a timely manner. And like many other compression methods, symbol-based decoding is significantly faster than encoding. Finally, we note that both the symbol bitmaps that are stored in the dictionary and the triplets used to reference them can themselves be encoded to further improve compression performance. If—as in Fig. 8.17—only exact symbol matches are allowed, the resulting compression is lossless; if small differences are permitted, some level of reconstruction error will be present.

a b c

FIGURE 8.17
(a) A bi-level document, (b) symbol dictionary, and (c) the triplets used to locate the symbols in the document.



Token	Symbol
0	
1	
2	

Triplet
(0, 2, 0)
(3, 10, 1)
(3, 18, 2)
(3, 26, 1)
(3, 34, 2)
(3, 42, 1)

JBIG2 compression

JBIG2 is an international standard for bilevel image compression. By segmenting an image into overlapping and/or non-overlapping regions of *text*, *halftone*, and *generic* content, compression techniques that are specifically optimized for each type of content are employed:

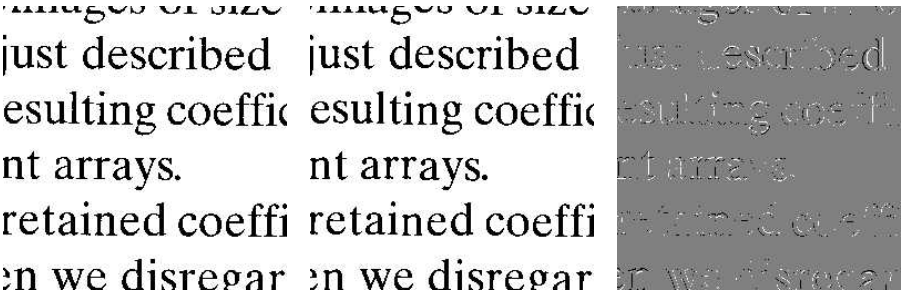
- *Text regions* are composed of characters that are ideally suited for a symbol-based coding approach. Typically, each symbol will correspond to a character bitmap—a subimage representing a character of text. There is normally only one character bitmap (or subimage) in the symbol dictionary for each upper- and lowercase character of the font being used. For example, there would be one "a" bitmap in the dictionary, one "A" bitmap, one "b" bitmap, and so on.

  In lossy JBIG2 compression, often called *perceptually lossless* or *visually lossless*, we neglect differences between dictionary bitmaps (i.e., the reference character bitmaps or character templates) and specific instances of the corresponding characters in the image. In lossless compression, the differences are stored and used in conjunction with the triplets encoding each character (by the decoder) to produce the actual image bitmaps. All bitmaps are encoded either arithmetically or using MMR (see Section 8.2.5); the triplets used to access dictionary entries are either arithmetically or Huffman encoded.
- *Halftone regions* are similar to text regions in that they are composed of patterns arranged in a regular grid. The symbols that are stored in the dictionary, however, are not character bitmaps but periodic patterns that represent intensities (e.g., of a photograph) that have been dithered to produce bilevel images for printing.
- *Generic regions* contain non-text, non-halftone information, like line art and noise, and are compressed using either arithmetic or MMR coding.

As is true of many image compression standards, JBIG2 defines decoder behavior. It does not explicitly define a standard encoder, but is flexible enough to allow various encoder designs. Although the design of the encoder is left unspecified, it is nevertheless important, because it determines the level of compression that is achieved. After all, the encoder must segment the image into regions, choose the text and halftone symbols that are stored in the dictionaries, and decide when those symbols are essentially the same as, or different from, potential instances of the symbols in the image. The decoder simply uses that information to recreate the original image.

EXAMPLE 8.11: JBIG2 compression example.

■ Consider again the bilevel image in Fig. 8.16(a). Figure 8.18(a) shows a reconstructed section of the image after lossless JBIG2 encoding (by a commercially available document compression application). It is an exact replica of the original image. Note that the *d*s in the reconstructed text vary slightly, despite the fact that they were generated from the same *d* entry in the dictionary. The differences between that *d* and the *d*s in the image were used to refine the output of the dictionary. The standard defines an algorithm for accomplishing



a b c

FIGURE 8.18 JBIG2 compression comparison: (a) lossless compression and reconstruction; (b) perceptually lossless; and (c) the scaled difference between the two.

this during the decoding of the encoded dictionary bitmaps. For the purposes of our discussion, you can think of it as adding the difference between a dictionary bitmap and a specific instance of the corresponding character in the image to the bitmap read from the dictionary.

Figure 8.18(b) is another reconstruction of the area in (a) after perceptually lossless JBIG2 compression. Note that the *d*s in this figure are identical. They have been copied directly from the symbol dictionary. The reconstruction is called perceptually lossless because the text is readable and the font is even the same. The small differences—shown in Fig. 8.18(c)—between the *d*s in the original image and the *d* in the dictionary are considered unimportant because they do not affect readability. Remember that we are dealing with bilevel images, so there are only three intensities in Fig. 8.18(c). Intensity 128 indicates areas where there is no difference between the corresponding pixels of the images in Figs. 8.18(a) and (b); intensities 0 (black) and 255 (white) indicate pixels of opposite intensities in the two images—for example, a black pixel in one image that is white in the other, and vice versa.

The lossless JBIG2 compression that was used to generate Fig. 8.18(a) reduces the original 663,445 byte uncompressed PDF image to 32,705 bytes; the compression ratio is $C = 20.3$. Perceptually lossless JBIG2 compression reduces the image to 23,913 bytes, increasing the compression ratio to about 27.7. These compressions are 4 to 5 times greater than the CCITT Group 3 and 4 results from Example 8.10. ■

8.2.7 Bit-Plane Coding

With reference to Tables 8.3 and 8.4, bit-plane coding is used in the

- JBIG1
- JPEG-2000

compression standards.

The run-length and symbol-based techniques of the previous sections can be applied to images with more than two intensities by processing their 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 (see Section 3.2.4) and compressing each binary image via one of several well-known binary compression methods. In this section, we describe the two most popular decomposition approaches.

The intensities of an m-bit monochrome image can be represented in the form of the base-2 polynomial

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \ldots + a_1 2^1 + a_0 2^0 \qquad (8.2\text{-}8)$$

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 Section 3.2.4, the lowest order bit plane (the plane corresponding to the least significant bit) is generated by collecting the a_0 bits of each pixel, while the highest order bit plane contains the a_{m-1} bits or coefficients. In general, each bit plane 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 decomposition approach is that small changes in intensity 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, because the most significant bits of the binary codes for 127 and 128 are different, the highest bit plane 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 intensity variations) is to first represent the image by an m-bit *Gray code*. The m-bit Gray code $g_{m-1} \ldots g_2 g_1 g_0$ that corresponds to the polynomial in Eq. (8.2-8) can be computed from

$$\begin{aligned} g_i &= a_i \oplus a_{i+1} \quad 0 \le i \le m-2 \\ g_{m-1} &= a_{m-1} \end{aligned} \tag{8.2-9}$$

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 intensity are less likely to affect all m bit planes. For instance, when intensity levels 127 and 128 are adjacent, only the highest order 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 8.12: Bit-plane coding.

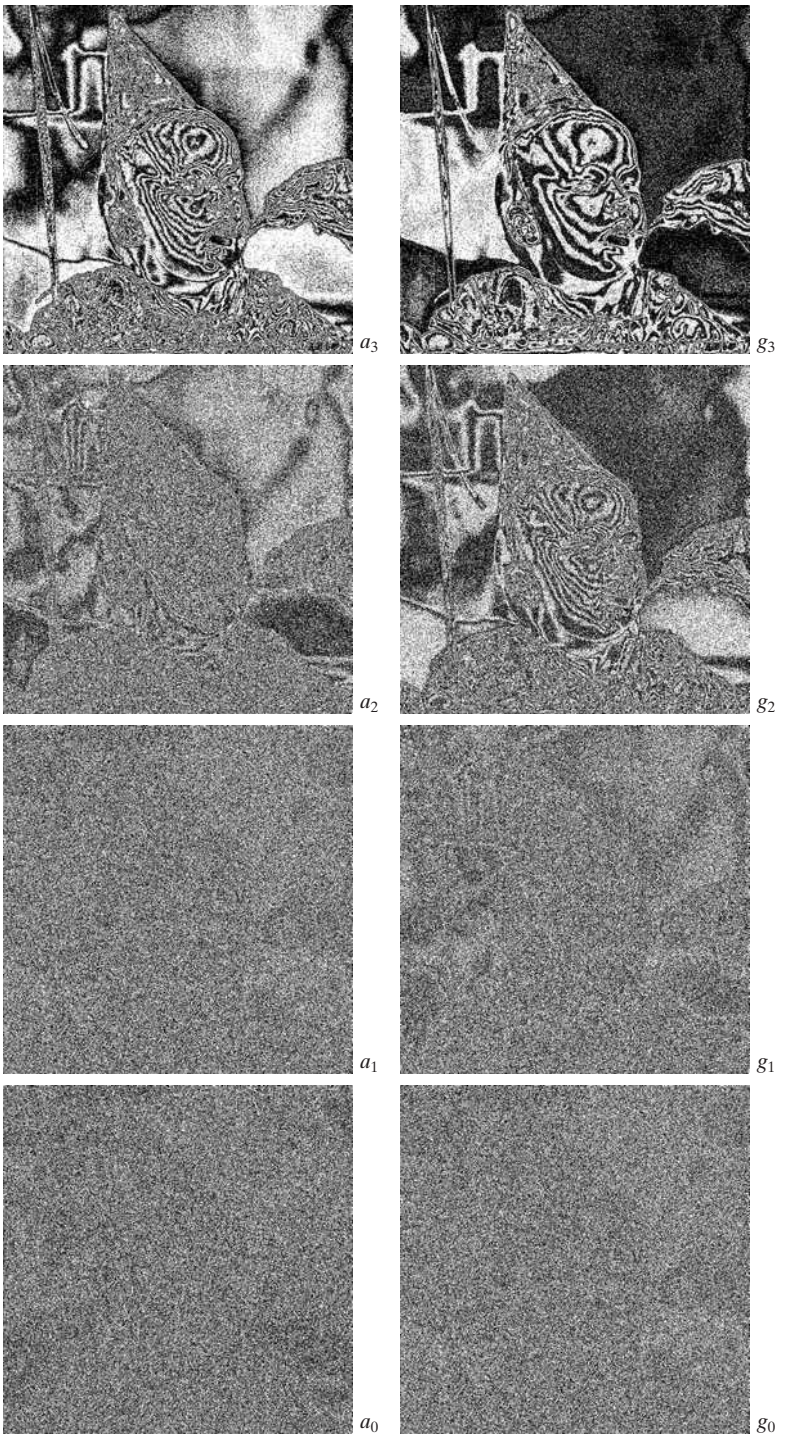
■ Figures 8.19 and 8.20 show the eight binary and Gray-coded bit planes of the 8-bit monochrome image of the child in Fig. 8.19(a). Note that the high-order bit planes are far less complex than their low-order counterparts. That is, they contain large uniform areas of significantly less detail, busyness, or randomness. In addition, the Gray-coded bit planes are less complex than the corresponding binary bit planes. Both observations are reflected in the JBIG2 coding results of Table 8.10. Note, for instance, that the a_5 and g_5 results are

TABLE 8.10 JBIG2 lossless coding results for the binary and Gray-coded bit planes of Fig. 8.19(a). These results include the overhead of each bit plane's PDF representation.

Coefficient m	Binary Code (PDF bits)	Gray Code (PDF bits)	Compression Ratio
7	6,999	6,999	1.00
6	12,791	11,024	1.16
5	40,104	36,914	1.09
4	55,911	47,415	1.18
3	78,915	67,787	1.16
2	101,535	92,630	1.10
1	107,909	105,286	1.03
0	99,753	107,909	0.92

a b
c d
e f
g h

FIGURE 8.19
(a) A 256-bit monochrome image. (b)–(h) The four most significant binary and Gray-coded bit planes of the image in (a).



a b
c d
e f
g h

FIGURE 8.20
(a)–(h) The four least significant binary (left column) and Gray-coded (right column) bit planes of the image in Fig. 8.19(a).

significantly larger than the a_6 and g_6 compressions; and that both g_5 and g_6 are smaller than their a_5 and a_6 counterparts. This trend continues throughout the table, with the single exception of a_0. Gray-coding provides a compression advantage of about 1.06:1 on average. Combined together, the Gray-coded files compress the original monochrome image by 678,676/475,964 or 1.43:1; the non-Gray-coded files compress the image by 678,676/503,916 or 1.35:1.

Finally, we note that the two least significant bits in Fig. 8.20 have little apparent structure. Because this is typical of most 8-bit monochrome images, bit-plane coding is usually restricted to images of 6 bits/pixel or less. JBIG1, the predecessor to JBIG2, imposes such a limit. ■

8.2.8 Block Transform Coding

With reference to Tables 8.3 and 8.4, block transform coding is used in

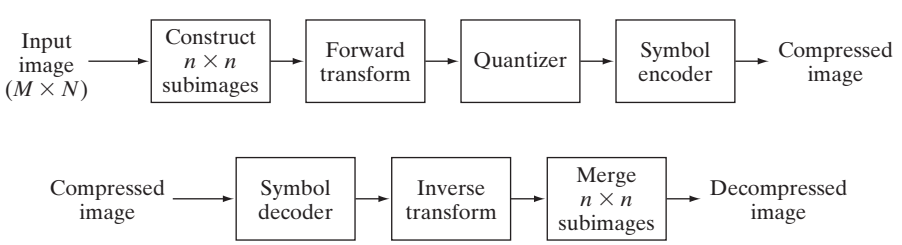
- JPEG
- M-JPEG
- MPEG-1, 2, 4
- H.261, H.262, H.263, and H.264
- DV and HDV
- VC-1

and other compression standards.

In this section, we consider a compression technique that divides an image into small non-overlapping blocks of equal size (e.g., 8×8) and processes the blocks independently using a 2-D transform. In *block transform coding*, a reversible, linear transform (such as the Fourier transform) is used to map each *block* or *subimage* into a set of transform coefficients, which are then quantized and coded. For most 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 8.21 shows a typical block 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 $M \times N$ input image is subdivided first into subimages of size $n \times n$, which are then transformed to generate MN/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 amount of information in a predefined sense (several methods are discussed later in the section). 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*.

In this section, we restrict our attention to square subimages (the most commonly used). It is assumed that the input image is padded, if necessary, so that both M and N are multiples of n.



a
b

FIGURE 8.21 A block transform coding system: (a) encoder; (b) decoder.

Transform selection

Block 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).

With reference to the discussion in Section 2.6.7, consider a subimage $g(x, y)$ of size $n \times n$ whose forward, discrete transform, $T(u, v)$, can be expressed in terms of the general relation

We use $g(x, y)$ to differentiate a subimage from the input image $f(x, y)$. Thus, the summation limits become n rather than M and N.

$$T(u, v) = \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} g(x, y)\, r(x, y, u, v) \tag{8.2-10}$$

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

$$g(x, y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v)\, s(x, y, u, v) \tag{8.2-11}$$

for $x, y = 0, 1, 2, \ldots, n - 1$. In these equations, $r(x, y, u, v)$ and $s(x, y, u, v)$ 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. (8.2-10) are called *transform coefficients*; they can be viewed as the expansion coefficients—see Section 7.2.1—of a series expansion of $g(x, y)$ with respect to basis functions $s(x, y, u, v)$.

As explained in Section 2.6.7, the kernel in Eq. (8.2-10) is separable if

$$r(x, y, u, v) = r_1(x, u)\, r_2(y, v) \tag{8.2-12}$$

In addition, the kernel is symmetric if r_1 is functionally equal to r_2. In this case, Eq. (8.2-12) can be expressed in the form

$$r(x, y, u, v) = r_1(x, u)\, r_1(y, v) \tag{8.2-13}$$

Identical comments apply to the inverse kernel if $r(x, y, u, v)$ is replaced by $s(x, y, u, v)$ in Eqs. (8.2-12) and (8.2-13). 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 4.11.1.

The forward and inverse transformation kernels in Eqs. (8.2-10) and (8.2-11) determine the type of transform that is computed and the overall computational complexity and reconstruction error of the block transform coding system in which they are employed. The best known transformation kernel pair is

$$r(x, y, u, v) = e^{-j2\pi(ux+vy)/n} \tag{8.2-14}$$

and

$$s(x, y, u, v) = \frac{1}{n^2} e^{j2\pi(ux+vy)/n} \tag{8.2-15}$$

where $j = \sqrt{-1}$. These are the transformation kernels defined in Eqs. (2.6-34) and (2.6-35) of Chapter 2 with $M = N = n$. Substituting these kernels into Eqs. (8.2-10) and (8.2-11) yields a simplified version of the discrete Fourier transform pair introduced in Section 4.5.5.

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

To compute the WHT of an $N \times N$ input image $f(x, y)$, rather than a subimage, change n to N in Eq. (8.2-16).

$$r(x, y, u, v) = s(x, y, u, v) = \frac{1}{n}(-1)^{\sum_{i=0}^{m-1} [b_i(x)p_i(u)+b_i(y)p_i(v)]} \tag{8.2-16}$$

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. (8.2-16) are computed using:

$$\begin{aligned} p_0(u) &= b_{m-1}(u) \\ p_1(u) &= b_{m-1}(u) + b_{m-2}(u) \\ p_2(u) &= b_{m-2}(u) + b_{m-3}(u) \\ &\vdots \\ p_{m-1}(u) &= b_1(u) + b_0(u) \end{aligned} \tag{8.2-17}$$

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. (8.2-14) and (8.2-15)], the Walsh-Hadamard kernels consist of alternating plus and minus 1s arranged in a checkerboard pattern. Figure 8.22 shows the kernel for $n = 4$. Each block consists of $4 \times 4 = 16$ elements (subsquares).

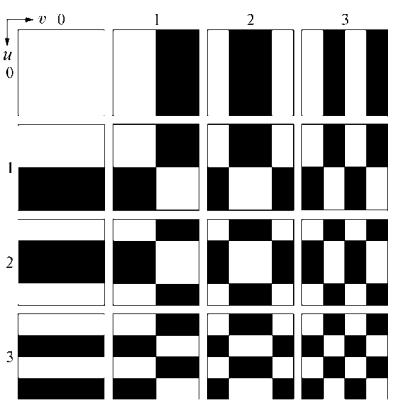


FIGURE 8.22 Walsh-Hadamard basis functions for $n = 4$. The origin of each block is at its top left.

White denotes +1 and black denotes −1. To obtain the top left block, we let $u = v = 0$ and plot values of $r(x, y, 0, 0)$ 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 $r(x, y, 0, 1)$ 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 transformations used most frequently for image compression is the *discrete cosine transform* (DCT). It is obtained by substituting the following (equal) kernels into Eqs. (8.2-10) and (8.2-11)

$$r(x, y, u, v) = s(x, y, u, v) = \alpha(u)\alpha(v)\cos\left[\frac{(2x + 1)u\pi}{2n}\right]\cos\left[\frac{(2y + 1)v\pi}{2n}\right] \quad (8.2\text{-}18)$$

To compute the DCT of an $N \times N$ input image $f(x, y)$, rather than a subimage, change n to N in Eqs. (8.2-18) and (8.2-19).

where

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{n}} & \text{for } u = 0 \\ \sqrt{\frac{2}{n}} & \text{for } u = 1, 2, \ldots, n - 1 \end{cases} \quad (8.2\text{-}19)$$

and similarly for $\alpha(v)$. Figure 8.23 shows $r(x, y, u, v)$ for the case $n = 4$. The computation follows the same format as explained for Fig. 8.22, with the difference that the values of r are not integers. In Fig. 8.23, the lighter intensity values correspond to larger values of r.

EXAMPLE 8.13: Block transform coding with the DFT, WHT, and DCT.

■ Figures 8.24(a) through (c) show three approximations of the 512 × 512 monochrome image in Fig. 8.9(a). 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.

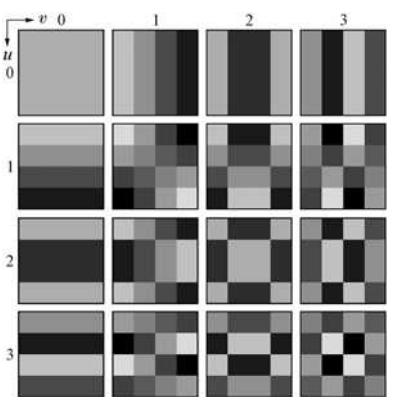


FIGURE 8.23 Discrete-cosine basis functions for $n = 4$. The origin of each block is at its top left.

a b c
d e f

FIGURE 8.24 Approximations of Fig. 8.9(a) using the (a) Fourier, (b) Walsh-Hadamard, and (c) cosine transforms, together with the corresponding scaled error images in (d)–(f).

In each case, the 32 retained coefficients were selected on the basis of maximum magnitude. Note that in all cases, the 32 discarded coefficients had little visual impact on the quality of the reconstructed image. Their elimination, however, was accompanied by some mean-square error, which can be seen in the scaled error images of Figs. 8.24(d) through (f). The actual rms errors were 2.32, 1.78, and 1.13 intensities, respectively. ■

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. (8.2-11), an $n \times n$ subimage $g(x, y)$ can be expressed as a function of its 2-D transform $T(u, v)$:

$$g(x, y) = \sum_{u=0}^{n-1} \sum_{v=0}^{n-1} T(u, v)\, s(x, y, u, v) \tag{8.2-20}$$

for $x, y = 0, 1, 2, \ldots, n - 1$. Because the inverse kernel $s(x, y, u, v)$ in Eq. (8.2-20) depends only on the indices x, y, u, v, and not on the values of $g(x, y)$ or $T(u, v)$, it can be viewed as defining a set of *basis functions* or *basis*

images for the series defined by Eq. (8.2-20). This interpretation becomes clearer if the notation used in Eq. (8.2-20) is modified to obtain

$$\mathbf{G} = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u, v)\mathbf{S}_{uv} \tag{8.2-21}$$

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

$$\mathbf{S}_{uv} = \begin{bmatrix} s(0,0,u,v) & s(0,1,u,v) & \cdots & s(0,n-1,u,v) \\ s(1,0,u,v) & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & & & \\ s(n-1,0,u,v) & s(n-1,1,u,v) & \cdots & s(n-1,n-1,u,v) \end{bmatrix} \tag{8.2-22}$$

Then $\mathbf{G}$, 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 $\mathbf{S}_{uv}$ for $u, v = 0, 1, 2, \ldots, n-1$ in Eq. (8.2-22). These matrices in fact are the basis images (or functions) of the series expansion in Eq. (8.2-20); the associated $T(u, v)$ are the expansion coefficients. Figures 8.22 and 8.23 illustrate graphically the WHT and DCT basis images for the case of $n = 4$.

If we now define a transform coefficient *masking function*

$$\chi(u, v) = \begin{cases} 0 & \text{if } T(u, v) \text{ satisfies a specified truncation criterion} \\ 1 & \text{otherwise} \end{cases} \tag{8.2-23}$$

for $u, v = 0, 1, 2, \ldots, n-1$, an approximation of $\mathbf{G}$ can be obtained from the truncated expansion

$$\hat{\mathbf{G}} = \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} \chi(u, v)T(u, v)\mathbf{S}_{uv} \tag{8.2-24}$$

where $\chi(u, v)$ is constructed to eliminate the basis images that make the smallest contribution to the total sum in Eq. (8.2-21). The mean-square error between subimage $\mathbf{G}$ and approximation $\hat{\mathbf{G}}$ then is

$$\begin{aligned} e_{ms} &= E\left\{\|\mathbf{G} - \hat{\mathbf{G}}\|^2\right\} \\ &= E\left\{\left\|\sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u, v)\mathbf{S}_{uv} - \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} \chi(u, v)T(u, v)\mathbf{S}_{uv}\right\|^2\right\} \\ &= E\left\{\left\|\sum_{u=0}^{n-1}\sum_{v=0}^{n-1} T(u, v)\mathbf{S}_{uv}\left[1 - \chi(u, v)\right]\right\|^2\right\} \\ &= \sum_{u=0}^{n-1}\sum_{v=0}^{n-1} \sigma^2_{T(u,v)}\left[1 - \chi(u, v)\right] \end{aligned} \tag{8.2-25}$$

where $\|\mathbf{G} - \hat{\mathbf{G}}\|$ is the norm of matrix $\left(\mathbf{G} - \hat{\mathbf{G}}\right)$ 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 $\mathbf{G}$ 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 $\chi(u, v) = 0$, so that $\left[1 - \chi(u, v)\right]$ in Eq. (8.2-25) 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. (8.2-25), the mean-square error of the MN/n^2 subimages of an $M \times N$ image are identical. Thus the mean-square error (being a measure of *average* error) of the $M \times N$ image equals the mean-square error of a single subimage.

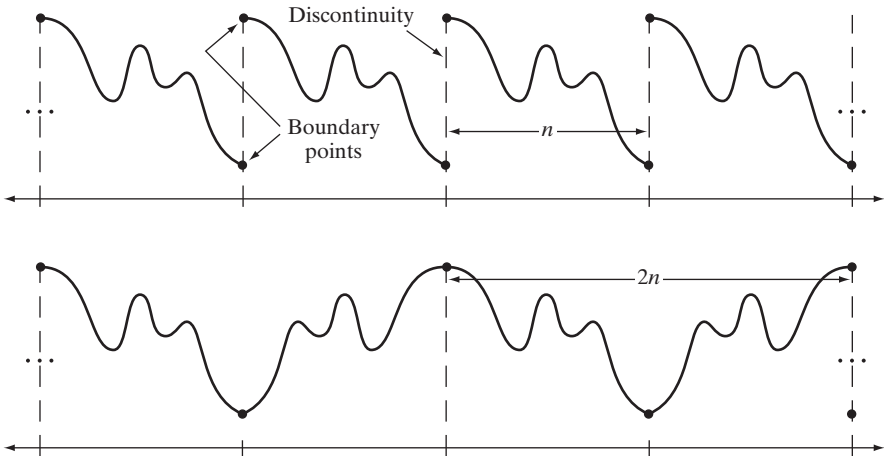
In Example 8.13, 50% of a DFT, WHT, and DCT block transform coded image's coefficients were discarded (using 8×8 blocks). After decoding, the DCT-based result had the smallest rms error, indicating that with respect to rms error the least amount of information was discarded.

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 images, the Karhunen-Loève transform (see Chapter 11), not the DCT, is the optimal transform in an information packing sense. This is due to the fact that the KLT minimizes the mean-square error in Eq. (8.2-25) 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 used infrequently 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, most 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 systems. 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 images), and minimizing the block-like 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. 8.25(a) shows, the implicit n-point periodicity (see Section 4.6.3) of the DFT gives rise to boundary discontinuities that result in substantial high-frequency transform

[†] An additional condition for optimality is that the masking function of Eq. (8.2-23) selects the KLT coefficients of maximum variance.

[‡] 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]).



a
b

FIGURE 8.25 The periodicity implicit in the 1-D (a) DFT and (b) DCT.

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. 8.25(a)]. The DCT of Fig. 8.25(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 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 method discussed in Section 4.11.3). 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 8.14: Effects of subimage size on transform coding.

■ Figure 8.26 illustrates graphically the impact of subimage size on transform coding reconstruction error. The data plotted were obtained by dividing the monochrome image of Fig. 8.9(a) into subimages of size $n \times n$, for $n = 2, 4, 8, 16, \ldots, 256, 512$, 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

[†]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 of the points of discontinuity.

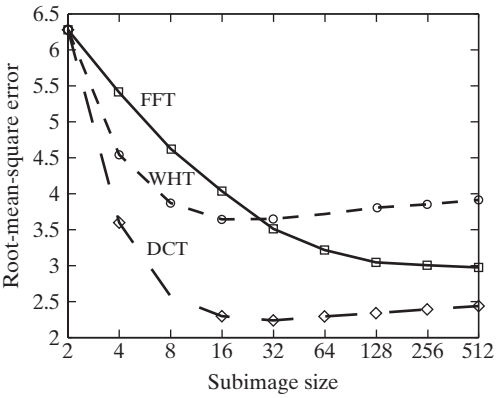


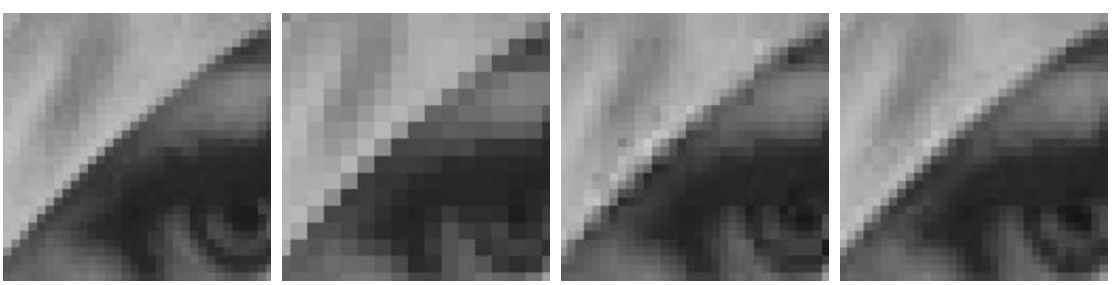
FIGURE 8.26 Reconstruction error versus subimage size.

error continues to decrease in this region. As n further increases, the Fourier reconstruction error crosses the Walsh-Hadamard curve and approaches the cosine result. 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. (4.6-21)]. This condition is evident in Fig. 8.27(b), 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. 8.27(c) and (d). Figure 8.27(a) shows a zoomed portion of the original image for reference. ■

Bit allocation

The reconstruction error associated with the truncated series expansion of Eq. (8.2-24) is a function of the number and relative importance of the



a b c d

FIGURE 8.27 Approximations of Fig. 8.27(a) using 25% of the DCT coefficients and (b) 2×2 subimages, (c) 4×4 subimages, and (d) 8×8 subimages. The original image in (a) is a zoomed section of Fig. 8.9(a).

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. (8.2-23) 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 8.15: Bit allocation.

■ Figures 8.28(a) and (c) show two approximations of Fig. 8.9(a) 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. 8.28(b) contains less error than the zonal coding result in Fig. 8.28(d). Both images have been scaled to make the errors more visible. The corresponding rms errors are 4.5 and 6.5 intensities, respectively. ■

a b
c d

FIGURE 8.28 Approximations of Fig. 8.9(a) using 12.5% of the 8×8 DCT coefficients: (a)—(b) threshold coding results; (c)—(d) zonal coding results. The difference images are scaled by 4.

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 MN/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. (8.2-24), 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. 8.29(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. 8.29(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 (see Optimal quantizers in Section 8.2.9), is designed for each coefficient. To construct the required quantizers, the zeroth or dc coefficient normally is modeled by a Rayleigh density function, whereas the remaining coefficients are modeled by a Laplacian or

a b
c d

FIGURE 8.29
A typical (a) zonal mask, (b) zonal bit allocation, (c) threshold mask, and (d) thresholded coefficient ordering sequence. Shading highlights the coefficients that are retained.

1	1	1	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

8	7	6	4	3	2	1	0
7	6	5	4	3	2	1	0
6	5	4	3	3	1	1	0
4	4	3	3	2	1	0	0
3	3	3	2	1	1	0	0
2	2	1	1	1	0	0	0
1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

1	1	0	1	1	0	0	0
1	1	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Gaussian density.[†] The number of quantization levels (and thus the number of bits) allotted to each quantizer is made proportional to $\log_2 \sigma^2_{T(u,v)}$. Thus the retained coefficients in Eq. (8.2-24)—which (in the context of the current discussion) are selected on the basis of maximum variance—are assigned bits in proportion to the logarithm of the coefficient variances.

Threshold coding implementation Zonal coding usually is implemented by using a single fixed mask for all subimages. Threshold coding, however, is inherently adaptive in the sense that the location of the transform coefficients retained for each subimage vary from one subimage to another. In fact, threshold coding is the adaptive transform coding approach most often used in practice because of its computational simplicity. The underlying concept is that, for any subimage, the transform coefficients of largest magnitude make the most significant contribution to reconstructed subimage quality, as demonstrated in the last example. Because the locations of the maximum coefficients vary from one subimage to another, the elements of $\chi(u, v)T(u, v)$ normally are reordered (in a predefined manner) to form a 1-D, run-length coded sequence. Figure 8.29(c) shows a typical *threshold mask* for one subimage of a hypothetical image. This mask provides a convenient way to visualize the threshold coding process for the corresponding subimage, as well as to mathematically describe the process using Eq. (8.2-24). When the mask is applied [via Eq. (8.2-24)] to the subimage for which it was derived, and the resulting $n \times n$ array is reordered to form an n^2-element coefficient sequence in accordance with the zigzag ordering pattern of Fig. 8.29(d), the reordered 1-D sequence contains several long runs of 0s [the zigzag pattern becomes evident by starting at 0 in Fig. 8.29(d) and following the numbers in sequence]. These runs normally are run-length coded. The nonzero or retained coefficients, corresponding to the mask locations that contain a 1, are represented using a variable-length code.

There are three basic ways to threshold a transformed subimage or, stated differently, to create a subimage threshold masking function of the form given in Eq. (8.2-23): (1) A single global threshold can be applied to all subimages; (2) a different threshold can be used for each subimage; or (3) the threshold can be varied as a function of the location of each coefficient within the subimage. In the first approach, the level of compression differs from image to image, depending on the number of coefficients that exceed the global threshold. In the second, called *N-largest coding*, the same number of coefficients is discarded for each subimage. As a result, the code rate is constant and known in advance. The third technique, like the first, results in a variable code rate, but offers the advantage that thresholding *and* quantization can be combined

The *N* in "*N*-largest coding" is not an image dimension, but refers to the number of coefficients that are kept.

[†]As each coefficient is a linear combination of the pixels in its subimage [see Eq. (8.2-10)], the central-limit theorem suggests that, as subimage size increases, the coefficients tend to become Gaussian. This result does not apply to the dc coefficient, however, because nonnegative images always have positive dc coefficients.

by replacing $\chi(u, v)T(u, v)$ in Eq. (8.2-24) with

$$\hat{T}(u, v) = \text{round}\left[\frac{T(u, v)}{Z(u, v)}\right] \tag{8.2-26}$$

where $\hat{T}(u, v)$ is a thresholded and quantized approximation of $T(u, v)$, and $Z(u, v)$ is an element of the transform normalization array

$$\mathbf{Z} = \begin{bmatrix} Z(0,0) & Z(0,1) & \cdots & Z(0,n-1) \\ Z(1,0) & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ Z(n-1,0) & Z(n-1,1) & \cdots & Z(n-1,n-1) \end{bmatrix} \tag{8.2-27}$$

Before a normalized (thresholded and quantized) subimage transform, $\hat{T}(u, v)$, can be inverse transformed to obtain an approximation of subimage $g(x, y)$, it must be multiplied by $Z(u, v)$. The resulting denormalized array, denoted $\dot{T}(u, v)$ is an approximation of $\hat{T}(u, v)$:

$$\dot{T}(u, v) = \hat{T}(u, v)Z(u, v) \tag{8.2-28}$$

The inverse transform of $\dot{T}(u, v)$ yields the decompressed subimage approximation.

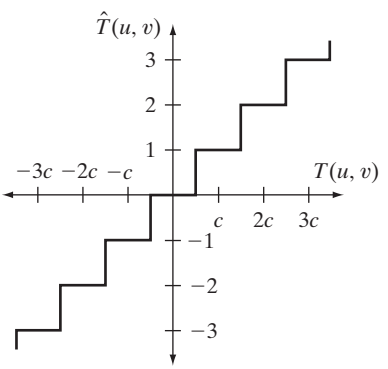
Figure 8.30(a) depicts Eq. (8.2-26) graphically for the case in which $Z(u, v)$ is assigned a particular value c. Note that $\hat{T}(u, v)$ assumes integer value k if and only if

$$kc - \frac{c}{2} \leq T(u, v) < kc + \frac{c}{2} \tag{8.2-29}$$

If $Z(u, v) > 2T(u, v)$, then $\hat{T}(u, v) = 0$ and the transform coefficient is completely truncated or discarded. When $\hat{T}(u, v)$ is represented with a variable-length code that increases in length as the magnitude of k increases, the number of bits used to represent $T(u, v)$ is controlled by the value of c. Thus the elements of $\mathbf{Z}$

a b

FIGURE 8.30
(a) A threshold coding quantization curve [see Eq. (8.2-29)]. (b) A typical normalization matrix.



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

can be scaled to achieve a variety of compression levels. Figure 8.30(b) shows a typical normalization array. This array, which has been used extensively in the JPEG standardization efforts (see the next section), weighs each coefficient of a transformed subimage according to heuristically determined perceptual or psychovisual importance.

EXAMPLE 8.16: Illustration of threshold coding.

■ Figures 8.31(a) through (f) show six threshold-coded approximations of the monochrome image in Fig. 8.9(a). All images were generated using an 8×8 DCT and the normalization array of Fig. 8.30(b). The first result, which provides a compression ratio of about 12 to 1 (i.e., $C = 12$), was obtained by direct application of that normalization array. The remaining results, which compress the original image by 19, 30, 49, 85, and 182 to 1, were generated after multiplying (scaling) the normalization arrays by 2, 4, 8, 16, and 32, respectively. The corresponding rms errors are 3.83, 4.93, 6.62, 9.35, 13.94, and 22.46 intensity levels. ■

JPEG

One of the most popular and comprehensive continuous tone, still frame compression standards is the JPEG standard. It defines three different coding systems: (1) a lossy baseline coding system, which is based on the DCT and is adequate for most compression applications; (2) an extended coding system for

a b c
d e f

FIGURE 8.31 Approximations of Fig. 8.9(a) using the DCT and normalization array of Fig. 8.30(b): (a) **Z**, (b) 2**Z**, (c) 4**Z**, (d) 8**Z**, (e) 16**Z**, and (f) 32**Z**.

greater compression, higher precision, or progressive reconstruction applications; and (3) a lossless independent coding system for reversible compression. To be JPEG compatible, a product or system must include support for the baseline system. No particular file format, spatial resolution, or color space model is specified.

In the baseline system, often called the *sequential baseline system*, the input and output data precision is limited to 8 bits, whereas the quantized DCT values are restricted to 11 bits. The compression itself is performed in three sequential steps: DCT computation, quantization, and variable-length code assignment. The image is first subdivided into pixel blocks of size 8×8, which are processed left to right, top to bottom. As each 8×8 block or subimage is encountered, its 64 pixels are level-shifted by subtracting the quantity 2^{k-1}, where 2^k is the maximum number of intensity levels. The 2-D discrete cosine transform of the block is then computed, quantized in accordance with Eq. (8.2-26), and reordered, using the zigzag pattern of Fig. 8.29(d), to form a 1-D sequence of quantized coefficients.

Because the one-dimensionally reordered array generated under the zigzag pattern of Fig. 8.29(d) is arranged qualitatively according to increasing spatial frequency, the JPEG coding procedure is designed to take advantage of the long runs of zeros that normally result from the reordering. In particular, the nonzero AC[†] coefficients are coded using a variable-length code that defines the coefficient values and number of preceding zeros. The DC coefficient is difference coded relative to the DC coefficient of the previous subimage. Tables A.3, A.4, and A.5 in Appendix A provide the default JPEG Huffman codes for the luminance component of a color image or intensity of a monochrome image. The JPEG recommended luminance quantization array is given in Fig. 8.30(b) and can be scaled to provide a variety of compression levels. The scaling of this array allows users to select the "quality" of JPEG compressions. Although default coding tables and quantization arrays are provided for both color and monochrome processing, the user is free to construct custom tables and/or arrays, which may in fact be adapted to the characteristics of the image(s) being compressed.

EXAMPLE 8.17: JPEG baseline coding and decoding.

■ Consider compression and reconstruction of the following 8×8 subimage with the JPEG baseline standard:

52	55	61	66	70	61	64	73
63	59	66	90	109	85	69	72
62	59	68	113	144	104	66	73
63	58	71	122	154	106	70	69
67	61	68	104	126	88	68	70
79	65	60	70	77	63	58	75
85	71	64	59	55	61	65	83
87	79	69	68	65	76	78	94

[†]In the standard, the term AC denotes all transform coefficients with the exception of the zeroth or DC coefficient.

The original image consists of 256 or 2^8 possible intensities, so the coding process begins by level shifting the pixels of the original subimage by -2^7 or -128 intensity levels. The resulting shifted array is

$$\begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -62 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -65 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix}$$

which, when transformed in accordance with the forward DCT of Eqs. (8.2-10) and (8.2-18) for $n = 8$, becomes

$$\begin{bmatrix} -415 & -29 & -62 & 25 & 55 & -20 & -1 & 3 \\ 7 & -21 & -62 & 9 & 11 & -7 & -6 & 6 \\ -46 & 8 & 77 & -25 & -30 & 10 & 7 & -5 \\ -50 & 13 & 35 & -15 & -9 & 6 & 0 & 3 \\ 11 & -8 & -13 & -2 & -1 & 1 & -4 & 1 \\ -10 & 1 & 3 & -3 & -1 & 0 & 2 & -1 \\ -4 & -1 & 2 & -1 & 2 & -3 & 1 & -2 \\ -1 & -1 & -1 & -2 & -1 & -1 & 0 & -1 \end{bmatrix}$$

If the JPEG recommended normalization array of Fig. 8.30(b) is used to quantize the transformed array, the scaled and truncated [that is, normalized in accordance with Eq. (8.2-26)] coefficients are

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\ 1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

where, for instance, the DC coefficient is computed as

$$\begin{aligned} \hat{T}(0,0) &= \text{round}\left[\frac{T(0,0)}{Z(0,0)}\right] \\ &= \text{round}\left[\frac{-415}{16}\right] = -26 \end{aligned}$$

Note that the transformation and normalization process produces a large number of zero-valued coefficients. When the coefficients are reordered in

accordance with the zigzag ordering pattern of Fig. 8.29(d), the resulting 1-D coefficient sequence is

$$[-26\ -3\ 1\ -3\ -2\ -6\ 2\ -4\ 1\ -4\ 1\ 1\ 5\ 0\ 2\ 0\ 0\ -1\ 2\ 0\ 0\ 0\ 0\ 0\ -1\ -1\ \text{EOB}]$$

where the EOB symbol denotes the end-of-block condition. A special EOB Huffman code word (see category 0 and run-length 0 in Table A.5) is provided to indicate that the remainder of the coefficients in a reordered sequence are zeros.

The construction of the default JPEG code for the reordered coefficient sequence begins with the computation of the difference between the current DC coefficient and that of the previously encoded subimage. Assuming the DC coefficient of the transformed and quantized subimage to its immediate left was 17, the resulting DPCM difference is $[-26 - (-17)]$ or -9, which lies in DC difference category 4 of Table A.3. In accordance with the default Huffman difference code of Table A.4, the proper base code for a category 4 difference is 101 (a 3-bit code), while the total length of a completely encoded category 4 coefficient is 7 bits. The remaining 4 bits must be generated from the least significant bits (LSBs) of the difference value. For a general DC difference category (say, category K), an additional K bits are needed and computed as either the K LSBs of the positive difference or the K LSBs of the negative difference minus 1. For a difference of -9, the appropriate LSBs are $(0111) - 1$ or 0110, and the complete DPCM coded DC code word is 1010110.

The nonzero AC coefficients of the reordered array are coded similarly from Tables A.3 and A.5. The principal difference is that each default AC Huffman code word depends on the number of zero-valued coefficients preceding the nonzero coefficient to be coded, as well as the magnitude category of the nonzero coefficient. (See the column labeled Run/Category in Table A.5.) Thus the first nonzero AC coefficient of the reordered array (-3) is coded as 0100. The first 2 bits of this code indicate that the coefficient was in magnitude category 2 and preceded by no zero-valued coefficients (see Table A.3); the last 2 bits are generated by the same process used to arrive at the LSBs of the DC difference code. Continuing in this manner, the completely coded (reordered) array is

$$1010110\ 0100\ 001\ 0100\ 0101\ 100001\ 0110\ 100011\ 001\ 100011\ 001$$
$$001\ 100101\ 11100110\ 110110\ 0110\ 11110100\ 000\ 1010$$

where the spaces have been inserted solely for readability. Although it was not needed in this example, the default JPEG code contains a special code word for a run of 15 zeros followed by a zero (see category 0 and run-length F in Table A.5). The total number of bits in the completely coded reordered array (and thus the number of bits required to represent the entire 8×8, 8-bit subimage of this example) is 92. The resulting compression ratio is 512/92, or about 5.6:1.

To decompress a JPEG compressed subimage, the decoder must first recreate the normalized transform coefficients that led to the compressed bit stream. Because a Huffman-coded binary sequence is instantaneous and uniquely decodable, this step is easily accomplished in a simple lookup table manner.

Here the regenerated array of quantized coefficients is

$$\begin{matrix} -26 & -3 & -6 & 2 & 2 & 0 & 0 & 0 \\ 1 & -2 & -4 & 0 & 0 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

After denormalization in accordance with Eq. (8.2-28), the array becomes

$$\begin{matrix} -416 & -33 & -60 & 32 & 48 & 0 & 0 & 0 \\ 12 & -24 & -56 & 0 & 0 & 0 & 0 & 0 \\ -42 & 13 & 80 & -24 & -40 & 0 & 0 & 0 \\ -56 & 17 & 44 & -29 & 0 & 0 & 0 & 0 \\ 18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}$$

where, for example, the DC coefficient is computed as

$$\dot{T}(0, 0) = \hat{T}(0, 0)Z(0, 0) = (-26)(16) = -416$$

The completely reconstructed subimage is obtained by taking the inverse DCT of the denormalized array in accordance with Eqs. (8.2-11) and (8.2-18) to obtain

$$\begin{matrix} -70 & -64 & -61 & -64 & -69 & -66 & -58 & -50 \\ -72 & -73 & -61 & -39 & -30 & -40 & -54 & -59 \\ -68 & -78 & -58 & -9 & 13 & -12 & -48 & -64 \\ -59 & -77 & -57 & 0 & 22 & -13 & -51 & -60 \\ -54 & -75 & -64 & -23 & -13 & -44 & -63 & -56 \\ -52 & -71 & -72 & -54 & -54 & -71 & -71 & -54 \\ -45 & -59 & -70 & -68 & -67 & -67 & -61 & -50 \\ -35 & -47 & -61 & -66 & -60 & -48 & -44 & -44 \end{matrix}$$

and level shifting each inverse transformed pixel by 2^7 (or +128) to yield

$$\begin{matrix} 58 & 64 & 67 & 64 & 59 & 62 & 70 & 78 \\ 56 & 55 & 67 & 89 & 98 & 88 & 74 & 69 \\ 60 & 50 & 70 & 119 & 141 & 116 & 80 & 64 \\ 69 & 51 & 71 & 128 & 149 & 115 & 77 & 68 \\ 74 & 53 & 64 & 105 & 115 & 84 & 65 & 72 \\ 76 & 57 & 56 & 74 & 75 & 57 & 57 & 74 \\ 83 & 69 & 59 & 60 & 61 & 61 & 67 & 78 \\ 93 & 81 & 67 & 62 & 69 & 80 & 84 & 84 \end{matrix}$$

Any differences between the original and reconstructed subimage are a result of the lossy nature of the JPEG compression and decompression process. In this example, the errors range from −14 to +11 and are distributed as follows:

−6	−9	−6	2	11	−1	−6	−5
7	4	−1	1	11	−3	−5	3
2	9	−2	−6	−3	−12	−14	9
−6	7	0	−4	−5	−9	−7	1
−7	8	4	−1	6	4	3	−2
3	8	4	−4	2	6	1	1
2	2	5	−1	−6	0	−2	5
−6	−2	2	6	−4	−4	−6	10

The root-mean-square error of the overall compression and reconstruction process is approximately 5.8 intensity levels. ■

EXAMPLE 8.18: Illustration of JPEG coding.

■ Figures 8.32(a) and (d) show two JPEG approximations of the monochrome image in Fig. 8.9(a). The first result provides a compression of 25:1; the second compresses the original image by 52:1. The differences between the original image and the reconstructed images in Figs. 8.30(a) and (d) are shown in Figs. 8.30(b) and (e), respectively. The corresponding rms errors are 5.4 and 10.7 intensities. The errors are clearly visible in the zoomed images in Figs. 8.32(c) and (f). These images show a magnified section of Figs. 8.32(a) and (d), respectively. Note that the JPEG blocking artifact increases with compression. ■

8.2.9 Predictive Coding

With reference to Tables 8.3 and 8.4, predictive coding is used in

- JBIG2
- JPEG
- JPEG-LS
- MPEG-1,2,4
- H.261, H.262, H.263, and H.264
- HDV
- VC-1

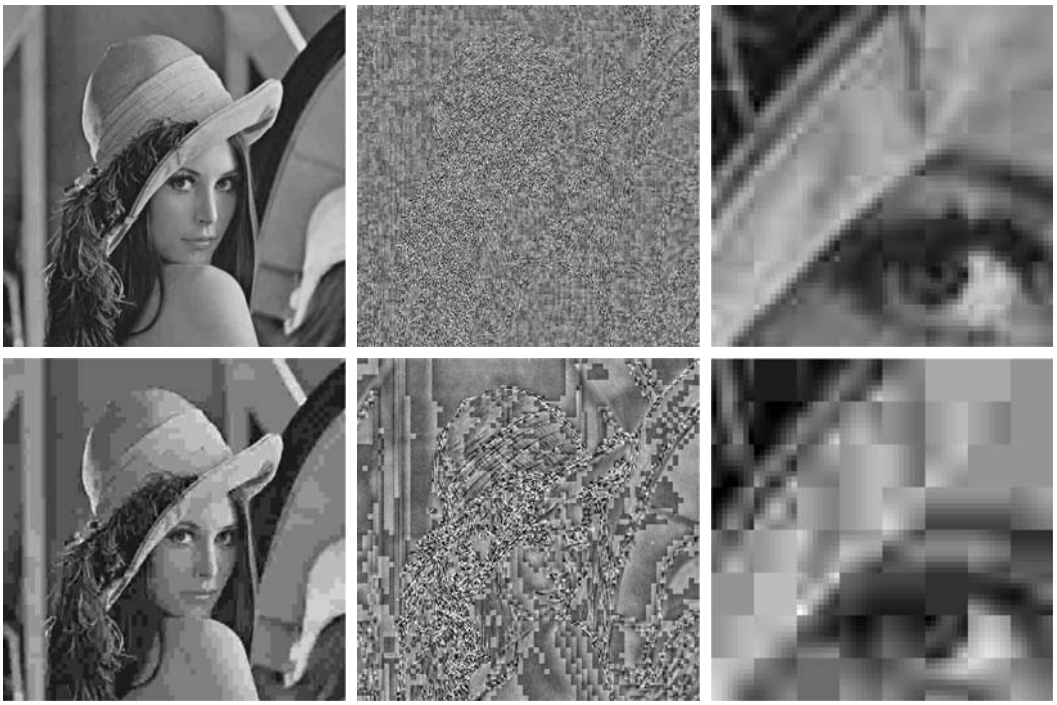
and other compression standards and file formats.

We now turn to a simpler compression approach that achieves good compression without significant computational overhead *and* can be either error-free or lossy. The approach, commonly referred to as *predictive coding*, is based on eliminating the redundancies of closely spaced pixels—in space and/or time—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 the pixel.

Lossless predictive coding

Figure 8.33 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 successive samples of discrete time input signal, $f(n)$, are introduced to the encoder, the predictor generates the anticipated value of each sample based on a specified number of past samples. 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*

$$e(n) = f(n) - \hat{f}(n) \tag{8.2-30}$$



a b c
d e f

FIGURE 8.32 Two JPEG approximations of Fig. 8.9(a). Each row contains a result after compression and reconstruction, the scaled difference between the result and the original image, and a zoomed portion of the reconstructed image.

which is encoded using a variable-length code (by the symbol encoder) to generate the next element of the compressed data stream. The decoder in Fig. 8.33(b) reconstructs $e(n)$ from the received variable-length code words and performs the inverse operation

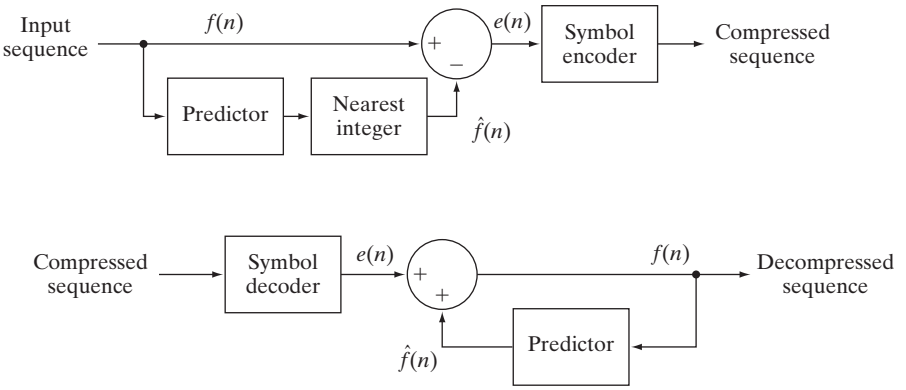
$$f(n) = e(n) + \hat{f}(n) \tag{8.2-31}$$

to decompress or recreate the original input sequence.

Various local, global, and adaptive methods (see the later subsection entitled Lossy predictive coding) can be used to generate $\hat{f}(n)$. In many cases, the prediction is formed as a linear combination of m previous samples. That is,

$$\hat{f}(n) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(n-i)\right] \tag{8.2-32}$$

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. If the input sequence in Fig. 8.33(a) is considered to be



a
b

FIGURE 8.33
A lossless predictive coding model:
(a) encoder;
(b) decoder.

samples of an image, the $f(n)$ in Eqs. (8.2-30) through (8.2-32) are pixels—and the m samples used to predict the value of each pixel come from the current scan line (called 1-D linear predictive coding), from the current and previous scan lines (called 2-D linear predictive coding), or from the current image and previous images in a sequence of images (called 3-D linear predictive coding). Thus, for 1-D linear predictive image coding, Eq. (8.2-32) can be written as

$$\hat{f}(x, y) = \text{round}\left[\sum_{i=1}^{m} \alpha_i f(x, y - i)\right] \tag{8.2-33}$$

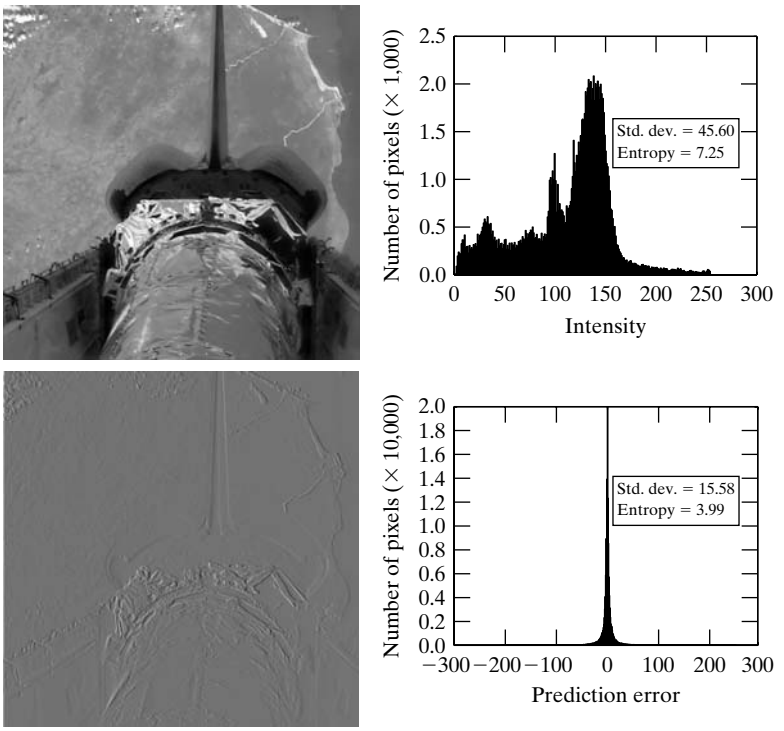
where each sample is now expressed explicitly as a function of the input image's spatial coordinates, x and y. Note that Eq. (8.2-33) indicates that the 1-D linear prediction is a function of the previous pixels on the current line alone. In 2-D predictive coding, the prediction is a function of the 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 (8.2-33) cannot be evaluated for the first m pixels of each line, so those pixels must be coded by using other means (such as a Huffman code) and considered as an overhead of the predictive coding process. Similar comments apply to the higher-dimensional cases.

EXAMPLE 8.19:
Predictive coding and spatial redundancy.

■ Consider encoding the monochrome image of Fig. 8.34(a) using the simple first-order (i.e., $m = 1$) linear predictor from Eq. (8.2-33)

$$\hat{f}(x, y) = \text{round}\left[\alpha f(x, y - 1)\right] \tag{8.2-34}$$

This equation is a simplification of Eq. (8.2-33) with $m = 1$ and the subscript of lone prediction coefficient α_1 dropped as unnecessary. A predictor of this general form is called a *previous pixel* predictor, and the corresponding predictive coding procedure is known as *differential coding* or *previous pixel coding*. Figure 8.34(c) shows the prediction error image, $e(x, y) = f(x, y) - \hat{f}(x, y)$, that results from Eq. (8.2-34) with $\alpha = 1$. The scaling of this image is such that intensity 128 represents a prediction error of zero, while all nonzero positive



a b
c d

FIGURE 8.34
(a) A view of the Earth from an orbiting space shuttle. (b) The intensity histogram of (a). (c) The prediction error image resulting from Eq. (8.2-34). (d) A histogram of the prediction error. (Original image courtesy of NASA.)

and negative prediction errors (under and over estimates) are displayed as lighter and darker shades of gray, respectively. The mean value of the prediction image is 128.26. Because intensity 128 corresponds to a prediction error of 0, the average prediction error is only 0.26 bits.

Figures 8.34(b) and (d) show the intensity histogram of the image in Fig. 8.34(a) and the histogram of prediction error $e(x, y)$, respectively. Note that the standard deviation of the prediction error in Fig. 8.34(d) is much smaller than the standard deviation of the intensities in the original image. Moreover, the entropy of the prediction error—as estimated using Eq. (8.1-7)—is significantly less than the estimated entropy of the original image (3.99 bits/pixel as opposed to 7.25 bits/pixel). This decrease in entropy reflects removal of a great deal of spatial redundancy, despite the fact that for k-bit images, $(k + 1)$-bit numbers are needed to represent accurately prediction error sequence $e(x, y)$. In general, the maximum compression of a predictive coding approach can be estimated by dividing the average number of bits used to represent each pixel in the original image by an estimate of the entropy of the prediction error. In this example, any variable-length coding procedure can be used to code $e(x, y)$, but the resulting compression will be limited to about $8/3.99$ or 2:1. ■

Note that the variable-length encoded prediction error is the compressed image.

The preceding example illustrates that the compression achieved in predictive coding is related directly to the entropy reduction that results from mapping

an input image into a prediction error sequence—often called a *prediction residual*. Because spatial redundancy is removed by the prediction and differencing process, the probability density function of the prediction residual is, in general, highly peaked at zero and characterized by a relatively small (in comparison to the input intensity distribution) variance. In fact, it is often modeled by a zero mean uncorrelated Laplacian PDF

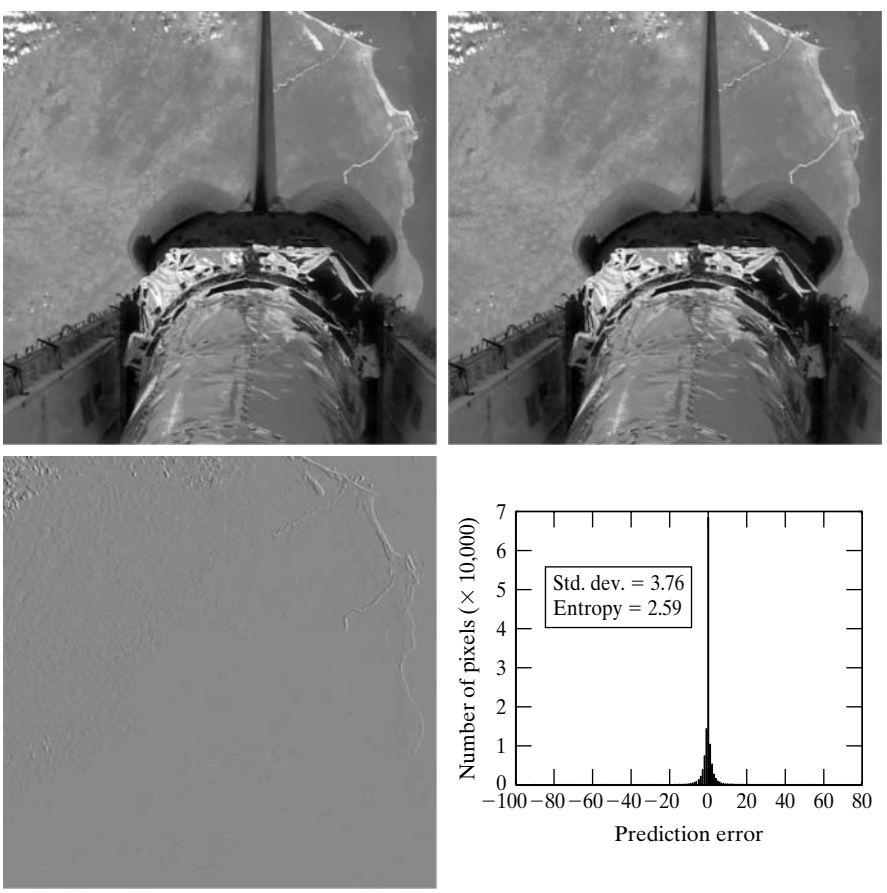
$$p_e(e) = \frac{1}{\sqrt{2}\sigma_e} e^{\frac{-\sqrt{2}|e|}{\sigma_e}} \tag{8.2-35}$$

where σ_e is the standard deviation of e.

EXAMPLE 8.20: Predictive coding and temporal redundancy.

■ The image in Fig. 8.34(a) is a portion of a frame of NASA video in which the Earth is moving from left to right with respect to a stationary camera attached to the space shuttle. It is repeated in Fig. 8.35(b)—along with its immediately preceding frame in Fig. 8.35(a). Using the first-order linear predictor

$$\hat{f}(x, y, t) = \text{round}\left[\alpha f(x, y, t - 1)\right] \tag{8.2-36}$$



a b
c d

FIGURE 8.35 (a) and (b) Two views of Earth from an orbiting space shuttle video. (c) The prediction error image resulting from Eq. (8.2-36). (d) A histogram of the prediction error. (Original images courtesy of NASA.)

with $\alpha = 1$, the intensities of the pixels in Fig. 8.35(b) can be predicted from the corresponding pixels in (a). Figure 8.34(c) is the resulting prediction residual image, $e(x, y, t) = f(x, y, t) - \hat{f}(x, y, t)$. Figure 8.34(d) is the histogram of $e(x, y, t)$. Note that there is very little prediction error. The standard deviation of the error is much smaller than in the previous example—3.76 bits/pixel as opposed to 15.58 bits/pixel. In addition, the entropy of the prediction error [computed using Eq. (8.1-7)] has decreased from 3.99 to 2.59 bits/pixel. By variable-length coding the resulting prediction residual, the original image is compressed by approximately 8/2.59 or 3.1:1—a 50% improvement over the 2:1 compression obtained using the spatially-oriented previous pixel predictor in Example 8.19. ■

Recall again that the variable-length encoded prediction error is the compressed image.

Motion compensated prediction residuals

As you saw in Example 8.20, successive frames in a video sequence often are very similar. Coding their differences can reduce temporal redundancy and provide significant compression. However, when a sequence of frames contains rapidly moving objects—or involves camera zoom and pan, sudden scene changes, or fade-ins and fade-outs—the similarity between neighboring frames is reduced and compression is affected negatively. That is, like most compression techniques (see Example 8.5), temporally-based predictive coding works best with certain kinds of inputs—namely, a sequence of images with significant temporal redundancy. When used on images with little temporal redundancy, data expansion can occur. Video compression systems avoid the problem of data expansion in two ways:

1. By tracking object movement and compensating for it during the prediction and differencing process.
2. By switching to an alternate coding method when there is insufficient *interframe* correlation (similarity between frames) to make predictive coding advantageous.

The first of these—called *motion compensation*—is the subject of the remainder of this section. Before proceeding, however, we note that when there is insufficient interframe correlation to make predictive coding effective, the second problem is typically addressed using a block-oriented 2-D transform, like JPEG's DCT-based coding (see Section 8.2.8). Frames compressed in this way (i.e., without a prediction residual) are called *intraframes* or *Independent frames* (*I-frames*). They can be decoded without access to other frames in the video to which they belong. I-frames usually resemble JPEG encoded images and are ideal starting points for the generation of prediction residuals. Moreover, they provide a high degree of random access, ease of editing, and resistance to the propagation of transmission error. As a result, all standards require the periodic insertion of I-frames into the compressed video codestream.

Figure 8.36 illustrates the basics of motion compensated predictive coding. Each video frame is divided into non-overlapping rectangular regions—typically of size 4×4 to 16×16—called *macroblocks*. (Only one macroblock is shown in Fig. 8.36.) The "movement" of each macroblock with respect to its "most likely" position in the previous (or subsequent) video frame, called the *reference frame*,

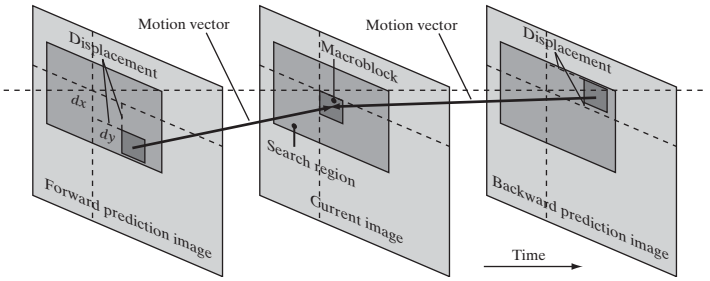


FIGURE 8.36 Macroblock motion specification.

The "most likely" position is the one that minimizes an error measure between the reference macroblock and macroblock being encoded. The two blocks do not have to be representations of the same object, but they must minimize the error measure.

is encoded in a *motion vector*. The vector describes the motion by defining the horizontal and vertical *displacement* from the "most likely" position. The displacements typically are specified to the nearest pixel, $\frac{1}{2}$ pixel, or $\frac{1}{4}$ pixel precision. If sub-pixel precision is used, the predictions must be interpolated [e.g., using bilinear interpolation (see Section 2.4.4)] from a combination of pixels in the reference frame. An encoded frame that is based on the previous frame (a *forward prediction* in Fig. 8.36) is called a *Predictive frame* (*P-frame*); one that is based on the subsequent frame (a *backward prediction* in Fig. 8.36) is called a *Bidirectional frame* (*B-frame*). B-frames require the compressed codestream to be reordered so that frames are presented to the decoder in the proper decoding sequence—rather than the natural display order.

As you might expect, *motion estimation* is the key component of motion compensation. During motion estimation, the motion of objects is measured and encoded into motion vectors. The search for the "best" motion vector requires that a criterion of optimality be defined. For example, motion vectors may be selected on the basis of maximum correlation or minimum error between macroblock pixels and the predicted pixels (or interpolated pixels for sub-pixel motion vectors) from the chosen reference frame. One of the most commonly used error measures is *mean absolute distortion* (*MAD*)

$$MAD(x, y) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} \left| f(x + i, y + j) - p(x + i + dx, y + j + dy) \right| \tag{8.2-37}$$

where x and y are the coordinates of the upper-left pixel of the $m \times n$ macroblock being coded, dx and dy are displacements from the reference frame as shown in Fig. 8.36, and p is an array of predicted macroblock pixel values. For sub-pixel motion vector estimation, p is interpolated from pixels in the reference frame. Typically, dx and dy must fall within a limited search region (see Fig. 8.36) around each macroblock. Values from ± 8 to ± 64 pixels are common, and the horizontal search area often is slightly larger than the vertical area. A more computationally efficient error measure, called the *sum of absolute distortions* (*SAD*), omits the $1/mn$ factor in Eq. (8.2-37).

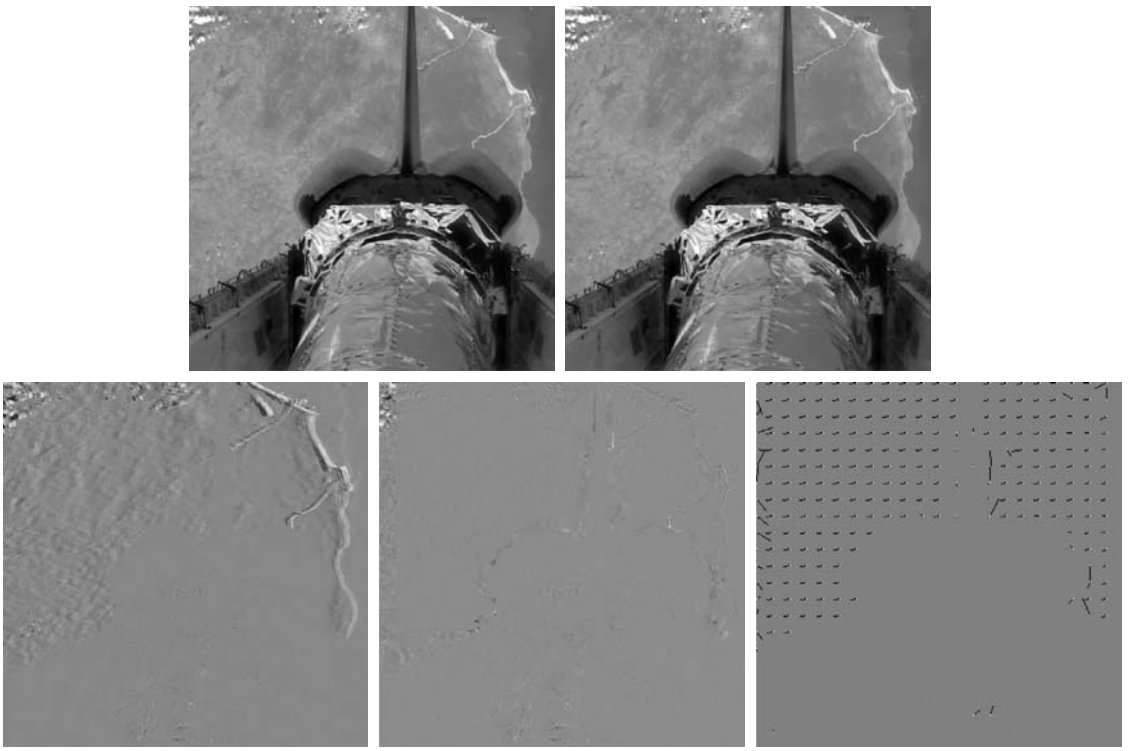
Given a selection criterion like that of Eq. (8.2-37), motion estimation is performed by searching for the dx and dy that minimize $MAD(x, y)$ over the

allowed range of motion vector displacements—including sub-pixel displacements. This process often is called *block matching*. An exhaustive search guarantees the best possible result, but is computationally expensive, because every possible motion must be tested over the entire displacement range. For 16×16 macroblocks and a ± 32 pixel displacement range (not out of the question for action films and sporting events), 4225 16×16 *MAD* calculations must be performed for each macroblock in a frame when integer displacement precision is used. If $\frac{1}{2}$ or $\frac{1}{4}$ pixel precision is desired, the number of calculations is multiplied by a factor of 4 or 16, respectively. Fast search algorithms can reduce the computational burden but may or may not yield optimal motion vectors. A number of fast block-based motion estimation algorithms have been proposed and studied in the literature (see, for example, Furht et al. [1997] or Mitchell et al. [1997]).

EXAMPLE 8.21: Motion compensated prediction.

■ Figures 8.37(a) and (b) were taken from the same NASA video sequence used in Examples 8.19 and 8.20. Figure 8.37(b) is identical to Figs. 8.34(a) and 8.35(b); Fig. 8.37(a) is the corresponding section of a frame occurring thirteen frames earlier. Figure 8.37(c) is the difference between the two frames, scaled to the full intensity range. Note that the difference is 0 in the area of the stationary (with respect to the camera) space shuttle, but there are significant differences in the remainder of the image due to the relative motion of the Earth. The standard deviation of the prediction residual in Fig. 8.37(c) is 12.73 intensity levels; its entropy [using Eq. (8.1-7)] is 4.17 bits/pixel. The maximum compression achievable when variable-length coding the prediction residual is $C = 8/4.17 = 1.92$.

Figure 8.37(d) shows a motion compensated prediction residual with a much lower standard deviation (5.62 as opposed to 12.73 intensity levels) and slightly lower entropy (3.04 vs. 4.17 bits/pixel). The entropy was computed using Eq. (8.1-7). If the prediction residual in Fig. 8.37(d) is variable-length coded, the resulting compression ratio is $C = 8/3.04 = 2.63$. To generate this prediction residual, we divided Fig. 8.37(b) into non-overlapping 16×16 macroblocks and compared each macroblock against every 16×16 region in Fig. 8.37(a)—the reference frame—that fell within ± 16 pixels of the macroblock's position in (b). We used Eq. (8.2-37) to determine the best match by selecting displacement (dx, dy) with the lowest *MAD*. The resulting displacements are the x and y components of the motion vectors shown in Fig. 8.37(e). The white dots in the figure are the heads of the motion vectors; they indicate the upper-left-hand corner of the coded macroblocks. As you can see from the pattern of the vectors, the predominant motion in the image is from left to right. In the lower portion of the image, which corresponds to the area of the space shuttle in the original image, there is no motion and therefore no motion vectors displayed. Macroblocks in this area are predicted from similarly located (i.e., the corresponding) macroblocks in the reference frame. Because the motion vectors in Fig. 8.37(e) are highly correlated, they can be variable-length coded to reduce their storage and transmission requirements. ■



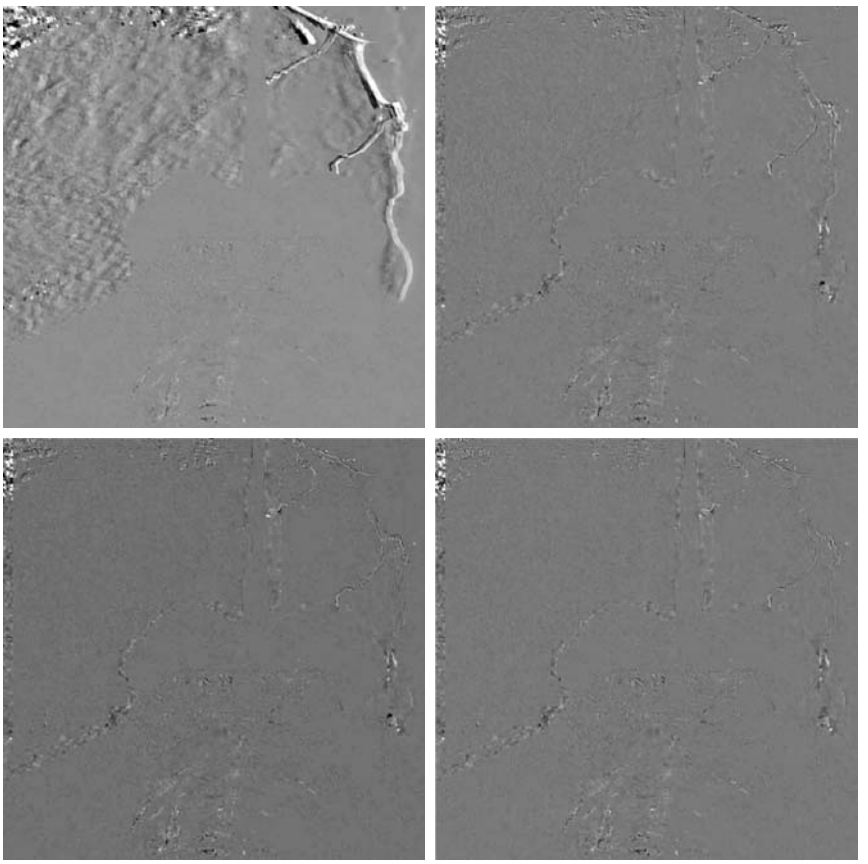
a b
c d e

FIGURE 8.37 (a) and (b) Two views of Earth that are thirteen frames apart in an orbiting space shuttle video. (c) A prediction error image without motion compensation. (d) The prediction residual with motion compensation. (e) The motion vectors associated with (d). The white dots in (d) represent the arrow heads of the motion vectors that are depicted. (Original images courtesy of NASA.)

The visual difference between Figs. 8.37(c) and 8.38(a) is due to scaling. The image in Fig. 8.38(a) has been scaled to match Figs. 8.38(b)–(d).

Figure 8.38 illustrates the increased prediction accuracy that is possible with sub-pixel motion compensation. Figure 8.38(a) is repeated from Fig. 8.37(c) and included as a point of reference; it shows the prediction error that results without motion compensation. The images in Figs. 8.38(b), (c), and (d) are motion compensated prediction residuals. They are based on the same two frames that were used in Example 8.21 and computed with macroblock displacements to 1, $\frac{1}{2}$, and $\frac{1}{4}$ pixel resolution (i.e., precision), respectively. Macroblocks of size 8×8 were used; displacements were limited to ± 8 pixels.

The most significant visual difference between the prediction residuals in Fig. 8.38 is the number and size of intensity peaks and valleys—their darkest and lightest areas of intensity. The $\frac{1}{4}$ pixel residual in Fig. 8.38(d) is the "flattest" of the four images, with the fewest excursions to black or white. As would be expected, it has the narrowest histogram. The standard deviations of the prediction residuals in Figs. 8.38(a) through (d) decrease as motion vector precision increases—from 12.7 to 4.4, 4, and 3.8 pixels, respectively. The entropies of the



a b
c d

FIGURE 8.38
Sub-pixel motion compensated prediction residuals: (a) without motion compensation; (b) single pixel precision; (c) $\frac{1}{2}$ pixel precision; and (d) $\frac{1}{4}$ pixel precision. (All prediction errors have been scaled to the full intensity range and then multiplied by 2 to increase their visibility.)

residuals, as determined using Eq. (8.1-7), are 4.17, 3.34, 3.35, and 3.34 bits/pixel, respectively. Thus, the motion compensated residuals contain about the same amount of information, despite the fact that the residuals in Figs. 8.38(c) and (d) use additional bits to accommodate $\frac{1}{2}$ and $\frac{1}{4}$ pixel interpolation. Finally, we note that there is an obvious strip of increased prediction error on the left side of each motion compensated residual. This is due to the left-to-right motion of the Earth, which introduces new or previously unseen areas of the Earth's terrain into the left side of each image. Because these areas are absent from the previous frames, they cannot be accurately predicted, regardless of the precision used to compute motion vectors.

Motion estimation is a computationally demanding task. Fortunately, only the encoder must estimate macroblock motion. Given the motion vectors of the macroblocks, the decoder simply accesses the areas of the reference frames that were used in the encoder to form the prediction residuals. Because of this, motion estimation is not included in most video compression standards. Compression standards focus on the decoder—placing constraints on macroblock dimensions, motion vector precision, horizontal and vertical

displacement ranges, and the like. Table 8.11 gives the key predictive coding parameters of some the most important video compression standards. Note that most of the standards use an 8×8 DCT for I-frame encoding, but specify a larger area (i.e., 16×16 macroblock) for motion compensation. In addition, even the P- and B-frame prediction residuals are transform coded due to the effectiveness of DCT coefficient quantization. Finally, we note that the H.264 and MPEG-4 AVC standards support intraframe predictive coding (in I-frames) to reduce spatial redundancy.

Figure 8.39 shows a typical motion compensated video encoder. It exploits redundancies within and between adjacent video frames, motion uniformity between frames, and the psychovisual properties of the human visual system. We can think of the input to the encoder as sequential macroblocks of video. For color video, each macroblock is composed of a luminance block and two chrominance blocks. Because the eye has far less spatial acuity for color than for luminance, the chrominance blocks often are sampled at half the horizontal and vertical resolution of the luminance block. The grayed elements in the figure parallel the transformation, quantization, and variable-length coding operations of a JPEG encoder. The principal difference is the input, which may be a conventional macroblock of image data (for I-frames)

TABLE 8.11
Predictive coding in video compression standards.

Parameter	H.261	MPEG-1	H.262 MPEG-2	H.263	MPEG-4	VC-1 WMV-9	H.264 MPEG-4 AVC
Motion vector precision	1	1/2	1/2	1/2	1/4	1/4	1/4
Macroblock sizes	16×16	16×16	16×16 16×8	16×16 8×8	16×16 8×8	16×16 8×8	16×16 16×8 8×16 8×8 8×4 4×8 4×4
Transform	8×8 DCT	8×8 DCT	8×8 DCT	8×8 DCT	8×8 DCT	8×8 8×4 4×8 4×4 Integer DCT	4×4 8×8 Integer
Interframe predictions	P	P, B	P, B	P, B	P, B	P, B	P, B
I-frame intra-predictions	No	No	No	No	No	No	Yes

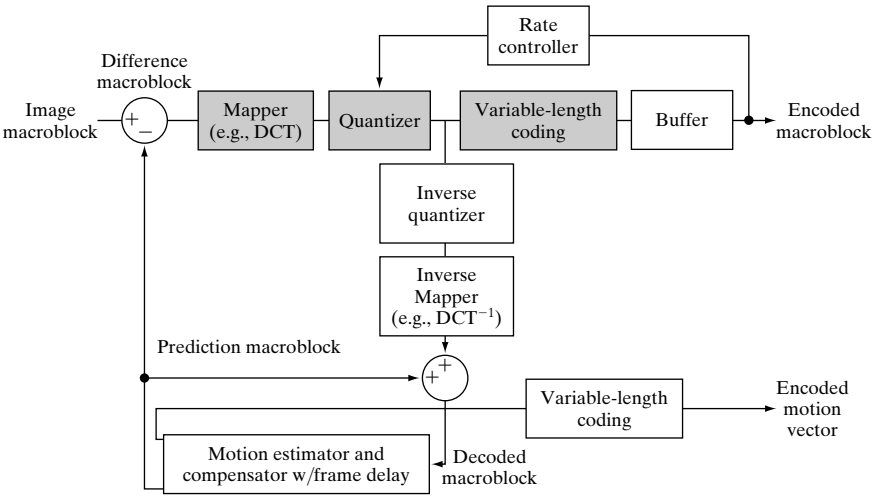


FIGURE 8.39
A typical motion compensated video encoder.

or the difference between a conventional macroblock and a prediction of it based on previous and/or subsequent video frames (for P- and B-frames). The encoder includes an *inverse quantizer* and *inverse mapper* (e.g., inverse DCT) so that its predictions match those of the complementary decoder. Also, it is designed to produce compressed bit streams that match the capacity of the intended video channel. To accomplish this, the quantization parameters are adjusted by a *rate controller* as a function of the occupancy of an output *buffer*. As the buffer becomes fuller, the quantization is made coarser, so that fewer bits stream into the buffer.

Quantization as defined earlier in the chapter is irreversible. The "inverse quantizer" in Fig. 8.39 does not prevent information loss.

■ We conclude our discussion of motion compensated predictive coding with an example illustrating the kind of compression that is possible with modern video compression methods. Figure 8.40 shows fifteen frames of a 1 minute HD (1280 × 720) full-color NASA video, parts of which have been used throughout this section. Although the images shown are monochrome, the video is a sequence of 1,829 full-color frames. Note that there are a variety of scenes, a great deal of motion, and multiple fade effects. For example, the video opens with a 150 frame fade-in from black, which includes frames 21 and 44 in Fig. 8.40, and concludes with a fade sequence containing frames 1595, 1609, and 1652 in Fig. 8.40, followed by a final fade to black. There are also several abrupt scene changes, like the change involving frames 1303 and 1304 in Fig. 8.40.

EXAMPLE 8.22:
Video compression example.

See the book Web site for the NASA video segment used in this section.

An H.264 compressed version of the NASA video stored as a Quicktime file (see Table 8.4) requires 44.56 MB of storage—plus another 1.39 MB for the associated audio. The video quality is excellent. About 5 GB of data would be needed to store the video frames as uncompressed full-color images. It should be noted that the video contains sequences involving both rotation and scale change (e.g., the sequence including frames 959, 1023, and 1088 in Fig. 840). The discussion in this section, however, has been limited to translation alone. ■

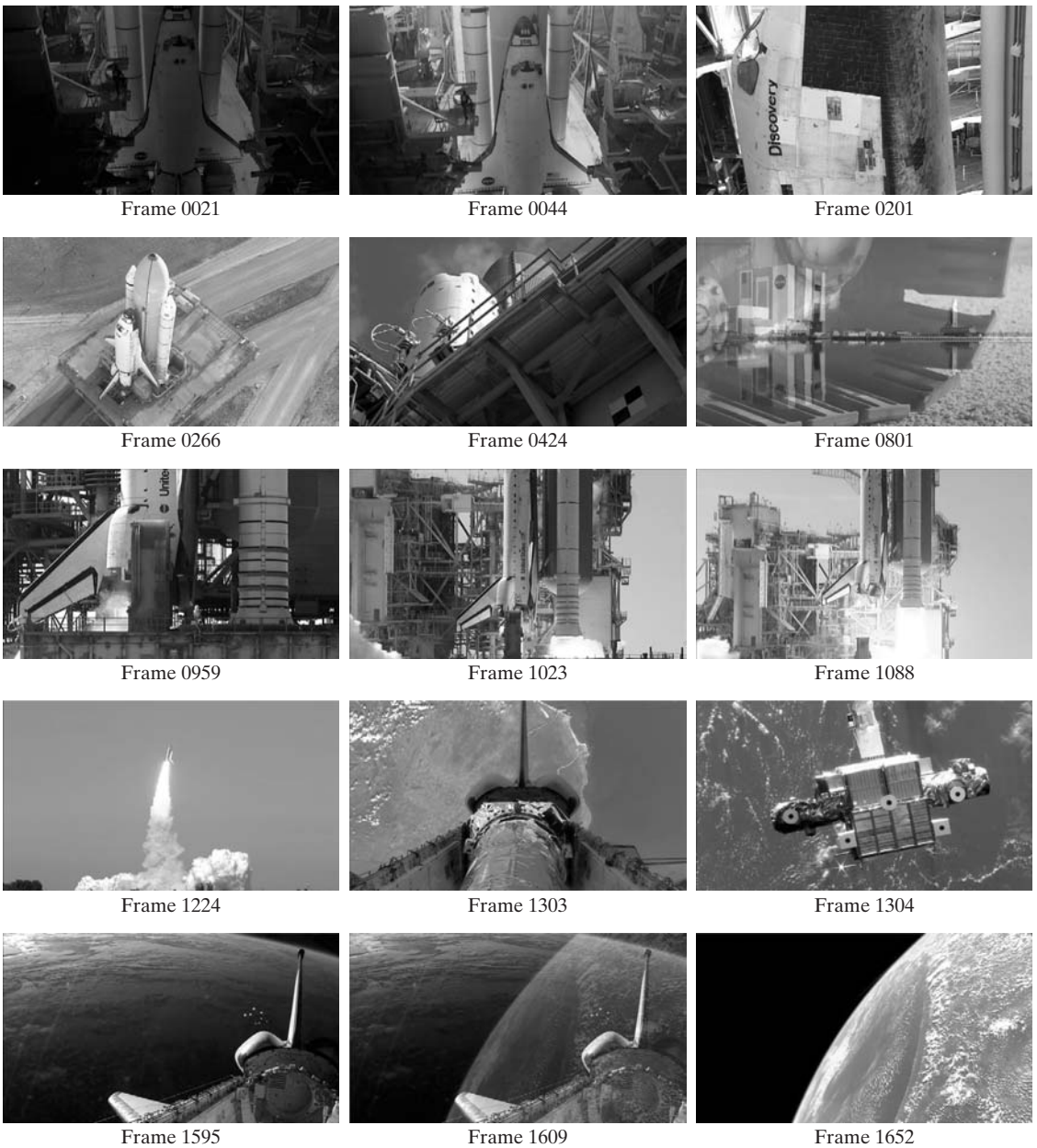
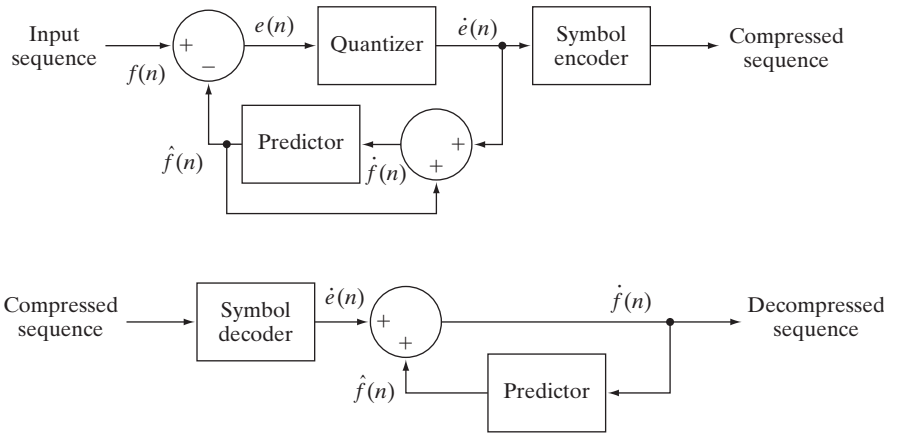


FIGURE 8.40 Fifteen frames from an 1829-frame, 1-minute NASA video. The original video is in HD full color. (Courtesy of NASA.)

Lossy predictive coding

In this section, we add a quantizer to the lossless predictive coding model introduced earlier and examine the trade-off between reconstruction accuracy and compression performance within the context of spatial predictors. As Fig. 8.41 shows, the quantizer, which replaces the nearest integer function of the error-free



a
b

FIGURE 8.41 A lossless predictive coding model: (a) encoder; (b) decoder.

encoder, is inserted between the symbol 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 that occurs.

In order to accommodate the insertion of the quantization step, the error-free encoder of Fig. 8.33(a) must be altered so that the predictions generated by the encoder and decoder are equivalent. As Fig. 8.41(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,

$$\dot{f}(n) = \dot{e}(n) + \hat{f}(n) \tag{8.2-38}$$

where $\hat{f}(n)$ is as defined earlier. This closed loop configuration prevents error buildup at the decoder's output. Note in Fig. 8.41(b) that the output of the decoder is given also by Eq. (8.2-38).

EXAMPLE 8.23: 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

$$\hat{f}(n) = \alpha \dot{f}(n-1) \tag{8.2-39}$$

and

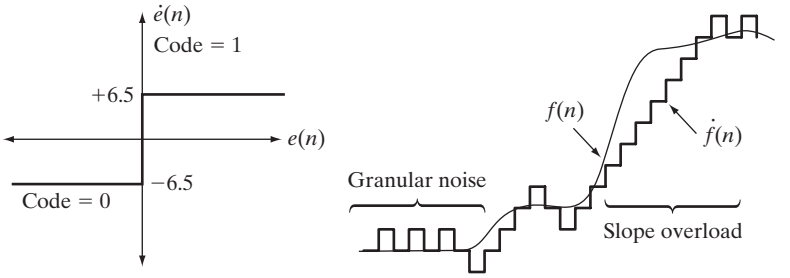
$$\dot{e}(n) = \begin{cases} +\zeta & \text{for } e(n) > 0 \\ -\zeta & \text{otherwise} \end{cases} \tag{8.2-40}$$

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. 8.42(a)], so the symbol encoder of Fig. 8.41(a) can utilize a 1-bit fixed-length code. The resulting DM code rate is 1 bit/pixel.

Figure 8.42(c) illustrates the mechanics of the delta modulation process, where the calculations needed to compress and reconstruct input sequence

a b
c

FIGURE 8.42
An example of delta modulation.



Input		Encoder				Decoder		Error
n	$f(n)$	$\hat{f}(n)$	$e(n)$	$\dot{e}(n)$	$\dot{f}(n)$	$\hat{f}(n)$	$\dot{f}(n)$	$f(n) - \dot{f}(n)$
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
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

$\{14, 15, 14, 15, 13, 15, 15, 14, 20, 26, 27, 28, 27, 27, 29, 37, 47, 62, 75, 77, 78, 79, 80, 81, 81, 82, 82\}$ with $\alpha = 1$ and $\zeta = 6.5$ are tabulated. The process begins with the error-free transfer of the first input sample 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. (8.2-39), (8.2-30), (8.2-40), and (8.2-38). Thus, when $n = 1$, for example, $\hat{f}(1) = (1)(14) = 14$, $e(1) = 15 - 14 = 1$, $\dot{e}(1) = +6.5$ (because $e(1) > 0$), $\dot{f}(1) = 6.4 + 14 = 20.5$, and the resulting reconstruction error is $(15 - 20.5)$, or -5.5.

Figure 8.42(b) shows graphically the tabulated data in Fig. 8.42(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 images, these two phenomena lead to blurred object edges and grainy or noisy surfaces (that is, distorted smooth areas). ■

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 with 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

In many predictive coding applications, the predictor is chosen to minimize the encoder's mean-square prediction error[†]

$$E\left\{e^2(n)\right\} = E\left\{\left[f(n) - \hat{f}(n)\right]^2\right\} \tag{8.2-41}$$

subject to the constraint that

$$\dot{f}(n) = \dot{e}(n) + \hat{f}(n) \approx e(n) + \hat{f}(n) = f(n) \tag{8.2-42}$$

and

$$\hat{f}(n) = \sum_{i=1}^{m} \alpha_i f(n - i) \tag{8.2-43}$$

That is, the optimization criterion is minimal 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 samples.[‡] 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

$$E\{e^2(n)\} = E\left\{\left[f(n) - \sum_{i=1}^{m} \alpha_i f(n - i)\right]^2\right\} \tag{8.2-44}$$

Differentiating Eq. (8.2-44) 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

$$\boldsymbol{\alpha} = \mathbf{R}^{-1}\mathbf{r} \tag{8.2-45}$$

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

$$\mathbf{R} = \begin{bmatrix} E\{f(n-1)\,f(n-1)\} & E\{f(n-1)f(n-2)\} & \cdots & E\{f(n-1)f(n-m)\} \\ E\{f(n-2)\,f(n-1)\} & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & & & \\ E\{f(n-m)\,f(n-1)\} & E\{f(n-m)f(n-2)\} & \cdots & E\{f(n-m)f(n-m)\} \end{bmatrix} \tag{8.2-46}$$

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

[‡]In general, the optimal predictor for a non-Gaussian sequence is a nonlinear function of the samples used to form the estimate.

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

$$\mathbf{r} = \begin{bmatrix} E\{f(n)f(n-1)\} \\ E\{f(n)f(n-2)\} \\ \vdots \\ E\{f(n)f(n-m)\} \end{bmatrix} \text{ and } \boldsymbol{\alpha} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix} \tag{8.2-47}$$

Thus for any input sequence, the coefficients that minimize Eq. (8.2-44) can be determined via a series of elementary matrix operations. Moreover, the coefficients depend only on the autocorrelations of the samples in the original sequence. The variance of the prediction error that results from the use of these optimal coefficients is

$$\sigma_e^2 = \sigma^2 - \boldsymbol{\alpha}^T\mathbf{r} = \sigma^2 - \sum_{i=1}^{m} E\{f(n)f(n-i)\}\alpha_i \tag{8.2-48}$$

Although the mechanics of evaluating Eq. (8.2-45) are quite simple, computation of the autocorrelations needed to form $\mathbf{R}$ and $\mathbf{r}$ is so difficult in practice that *local* predictions (those in which the prediction coefficients are computed for each input sequence) are almost never used. In most cases, a set of *global* coefficients is computed by assuming a simple input model and substituting the corresponding autocorrelations into Eqs. (8.2-46) and (8.2-47). For instance, when a 2-D Markov image source (see Section 8.1.4) with separable autocorrelation function

$$E\{f(x, y)f(x-i, y-j)\} = \sigma^2\rho_v^i\rho_h^j \tag{8.2-49}$$

and generalized fourth-order linear predictor

$$\begin{aligned} \hat{f}(x, y) = {} & \alpha_1 f(x, y-1) + \alpha_2 f(x-1, y-1) \\ & + \alpha_3 f(x-1, y) + \alpha_4 f(x-1, y+1) \end{aligned} \tag{8.2-50}$$

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

$$\alpha_1 = \rho_h \quad \alpha_2 = -\rho_v\rho_h \quad \alpha_3 = \rho_v \quad \alpha_4 = 0 \tag{8.2-51}$$

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. (8.2-43) normally is required to be less than or equal to one. That is,

$$\sum_{i=1}^{m} \alpha_i \leq 1 \tag{8.2-52}$$

This restriction is made to ensure that the output of the predictor falls within the allowed range of the input and to reduce the impact of transmission noise [which generally is seen as horizontal streaks in reconstructed images when the input to Fig. 8.41(a) is an 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. (8.2-52) to be strictly less than 1 confines the impact of an input error to a small number of outputs.

EXAMPLE 8.24: Comparison of prediction techniques.

■ Consider the prediction error that results from DPCM coding the monochrome image of Fig. 8.9(a) under the assumption of zero quantization error and with each of four predictors:

$$\hat{f}(x, y) = 0.97f(x, y - 1) \tag{8.2-53}$$

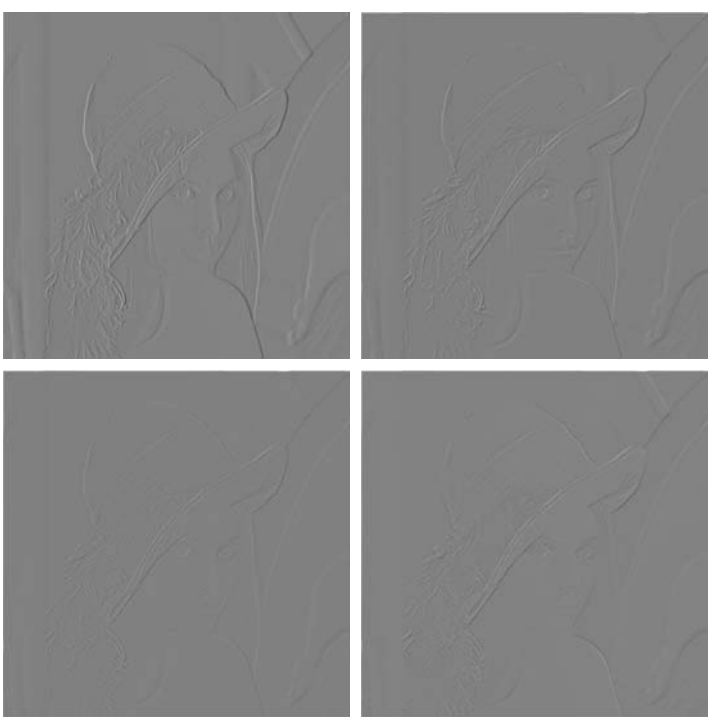
$$\hat{f}(x, y) = 0.5f(x, y - 1) + 0.5f(x - 1, y) \tag{8.2-54}$$

$$\hat{f}(x, y) = 0.75f(x, y - 1) + 0.75f(x - 1, y) - 0.5f(x - 1, y - 1) \tag{8.2-55}$$

$$\hat{f}(x, y) = \begin{cases} 0.97f(x, y - 1) & \text{if } \Delta h \leq \Delta v \\ 0.97f(x - 1, y) & \text{otherwise} \end{cases} \tag{8.2-56}$$

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 (8.2-53) through (8.2-56) define a relatively robust set of α_i that provide satisfactory performance over a wide range of images. The adaptive predictor of Eq. (8.2-56) 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 8.43(a) through (d) show the prediction error images that result from using the predictors of Eqs. (8.2-53) through (8.2-56). Note that the



a b
c d

FIGURE 8.43 A comparison of four linear prediction techniques.

visually perceptible error decreases as the order of the predictor increases.[†] The standard deviations of the prediction errors follow a similar pattern. They are 11.1, 9.8, 9.1, and 9.7 intensity levels, respectively. ■

Optimal quantization

The staircase quantization function $t = q(s)$ in Fig. 8.44 is an odd function of s [that is, $q(-s) = -q(s)$] that can be described completely 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, $E\{(s_i - t_i)^2\}$) and $p(s)$ is an even function, the conditions for minimal error (Max [1960]) are

$$\int_{s_{i-1}}^{s_i} (s - t_i)p(s)\,ds \quad i = 1, 2, \ldots, \frac{L}{2} \tag{8.2-57}$$

$$s_i = \begin{cases} 0 & i = 0 \\ \dfrac{t_i + t_{i+1}}{2} & i = 1, 2, \ldots, \dfrac{L}{2} - 1 \\ \infty & i = \dfrac{L}{2} \end{cases} \tag{8.2-58}$$

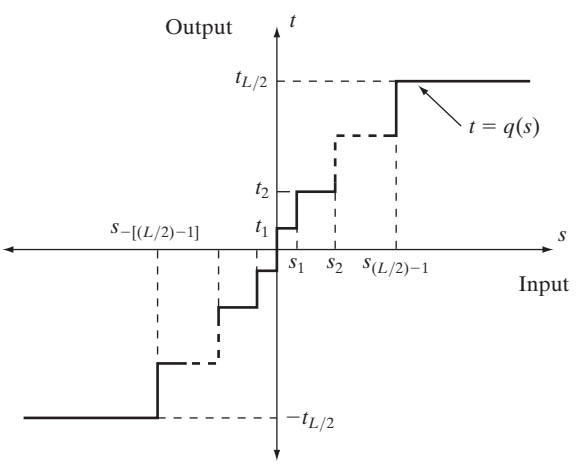


FIGURE 8.44 A typical quantization function.

[†]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.

and

$$s_{-i} = -s_i \quad t_{-i} = -t_i \tag{8.2-59}$$

Equation (8.2-57) indicates that the reconstruction levels are the centroids of the areas under $p(s)$ over the specified decision intervals, whereas Eq. (8.2-58) indicates that the decision levels are halfway between the reconstruction levels. Equation (8.2-59) is a consequence of the fact that q is an odd function. For any L, the s_i and t_i that satisfy Eqs. (8.2-57) through (8.2-59) are optimal in the mean-square error sense; the corresponding quantizer is called an L-level *Lloyd-Max* quantizer.

Table 8.12 lists the 2-, 4-, and 8-level Lloyd-Max decision and reconstruction levels for a unit variance Laplacian probability density function [see Eq. (8.2-35)]. Because obtaining an explicit or closed-form solution to Eqs. (8.2-57) through (8.2-59) 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 8.12 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. (8.2-57) through (8.5-59) *and* the additional constraint that

$$t_i - t_{i-1} = s_i - s_{i-1} = \theta \tag{8.2-60}$$

If a symbol encoder that utilizes a variable-length code is used in the general lossy predictive encoder of Fig. 8.41(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.

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	

TABLE 8.12 Lloyd-Max quantizers for a Laplacian probability density function of unit variance.

8.2.10 Wavelet Coding

With reference to Tables 8.3 and 8.4, wavelet coding is used in the

- JPEG-2000

compression standard.

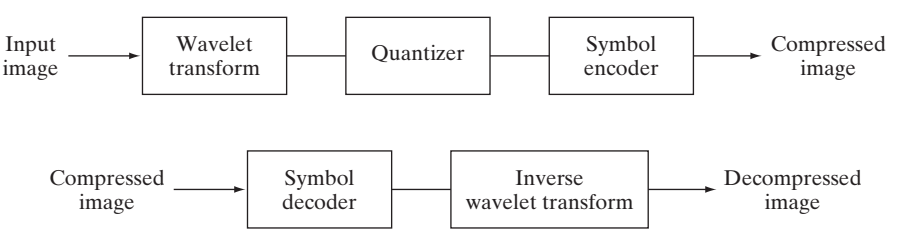
As with the transform coding techniques of Section 8.2.8, wavelet coding is based on the idea that the coefficients of a transform that decorrelates the pixels of an image can be coded more efficiently than the original pixels themselves. If the basis functions of the transform—in this case wavelets—pack most of the important visual information into a small number of coefficients, the remaining coefficients can be quantized coarsely or truncated to zero with little image distortion.

Figure 8.45 shows a typical wavelet coding system. To encode a $2^J \times 2^J$ image, an analyzing wavelet, ψ, and minimum decomposition level, $J - P$, are selected and used to compute the discrete wavelet transform of the image. If the wavelet has a complementary scaling function φ, the fast wavelet transform (see Sections 7.4 and 7.5) can be used. In either case, the computed transform converts a large portion of the original image to horizontal, vertical, and diagonal decomposition coefficients with zero mean and Laplacian-like probabilities. Recall the image of Fig. 7.1 and the dramatically simpler statistics of its wavelet transform in Fig. 7.10(a). Because many of the computed coefficients carry little visual information, they can be quantized and coded to minimize intercoefficient and coding redundancy. Moreover, the quantization can be adapted to exploit any positional correlation across the P decomposition levels. One or more lossless coding methods, like run-length, Huffman, arithmetic, and bit-plane coding, can be incorporated into the final symbol coding step. Decoding is accomplished by inverting the encoding operations—with the exception of quantization, which cannot be reversed exactly.

The principal difference between the wavelet-based system of Fig. 8.45 and the transform coding system of Fig. 8.21 is the omission of the subimage processing stages of the transform coder. Because wavelet transforms are both computationally efficient and inherently local (i.e., their basis functions are limited in duration), subdivision of the original image is unnecessary. As you will see later in this section, the removal of the subdivision step eliminates the blocking artifact that characterizes DCT-based approximations at high compression ratios.

Wavelet selection

The wavelets chosen as the basis of the forward and inverse transforms in Fig. 8.45 affect all aspects of wavelet coding system design and performance. They impact directly the computational complexity of the transforms and, less directly, the system's ability to compress and reconstruct



a
b

FIGURE 8.45
A wavelet coding system:
(a) encoder;
(b) decoder.

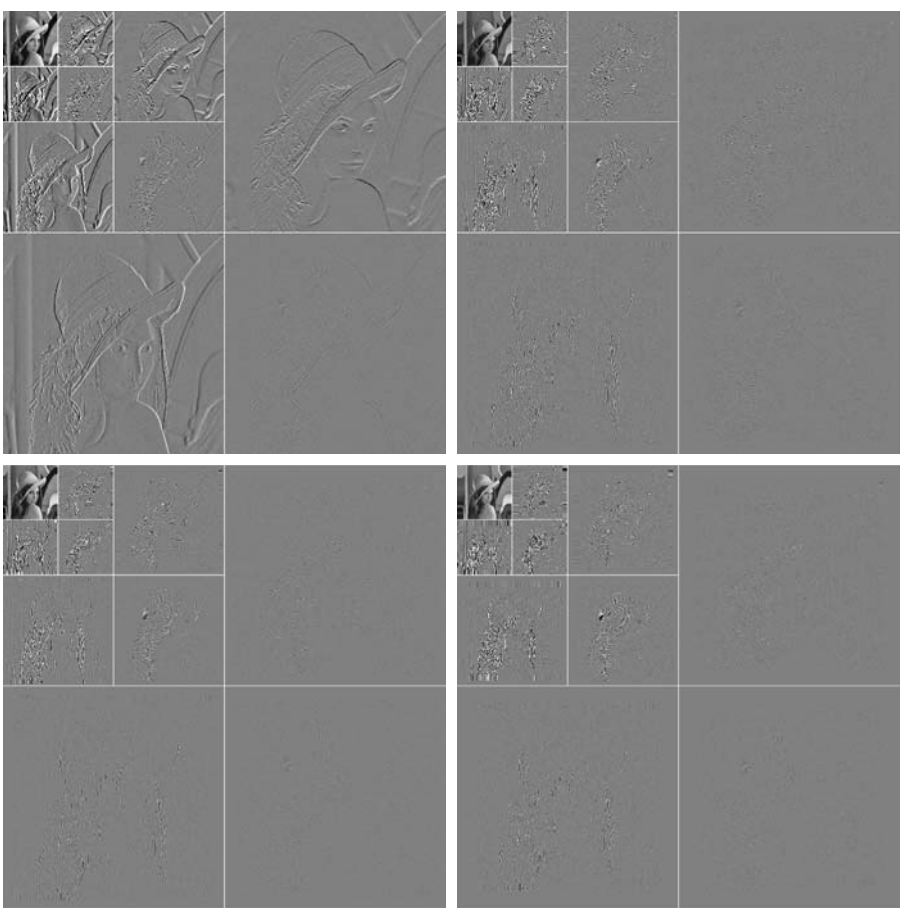
images of acceptable error. When the transforming wavelet has a companion scaling function, the transformation can be implemented as a sequence of digital filtering operations, with the number of filter taps equal to the number of nonzero wavelet and scaling vector coefficients. The ability of the wavelet to pack information into a small number of transform coefficients determines its compression and reconstruction performance.

In digital filtering, each filter tap multiplies a filter coefficient by a delayed version of the signal being filtered.

The most widely used expansion functions for wavelet-based compression are the Daubechies wavelets and biorthogonal wavelets. The latter allow useful analysis properties, like the number of zero moments (see Section 7.5), to be incorporated into the decomposition filters, while important synthesis properties, like smoothness of reconstruction, are built into the reconstruction filters.

EXAMPLE 8.25: Wavelet bases in wavelet coding.

■ Figure 8.46 contains four discrete wavelet transforms of Fig. 8.9(a). Haar wavelets, the simplest and only discontinuous wavelets considered in this example, were used as the expansion or basis functions in Fig. 8.46(a). Daubechies wavelets, among the most popular imaging wavelets, were used in Fig. 8.46(b),



a b
c d

FIGURE 8.46 Three-scale wavelet transforms of Fig. 8.9(a) with respect to (a) Haar wavelets, (b) Daubechies wavelets, (c) symlets, and (d) Cohen-Daubechies Feauveau biorthogonal wavelets.

and symlets, which are an extension of the Daubechies wavelets with increased symmetry, were used in Fig. 8.46(c). The Cohen-Daubechies-Feauveau wavelets that were employed in Fig. 8.46(d) are included to illustrate the capabilities of biorthogonal wavelets. As in previous results of this type, all detail coefficients were scaled to make the underlying structure more visible—with intensity 128 corresponding to coefficient value 0.

DWT detail coefficients are discussed in Section 7.3.2.

As you can see in Table 8.13, the number of operations involved in the computation of the transforms in Fig. 8.46 increases from 4 to 28 multiplications and additions per coefficient (for each decomposition level) as you move from Fig. 8.46(a) to (d). All four transforms were computed using a fast wavelet transform (i.e., filter bank) formulation. Note that as the computational complexity (i.e., the number of filter taps) increases, the information packing performance does as well. When Haar wavelets are employed and the detail coefficients below 1.5 are truncated to zero, 33.8% of the total transform is zeroed. With the more complex biorthogonal wavelets, the number of zeroed coefficients rises to 42.1%, increasing the potential compression by almost 10%. ■

Decomposition level selection

Another factor affecting wavelet coding computational complexity and reconstruction error is the number of transform decomposition levels. Because a P-scale fast wavelet transform involves P-filter bank iterations, the number of operations in the computation of the forward and inverse transforms increases with the number of decomposition levels. Moreover, quantizing the increasingly lower-scale coefficients that result with more decomposition levels affects increasingly larger areas of the reconstructed image. In many applications, like searching image databases or transmitting images for progressive reconstruction, the resolution of the stored or transmitted images and the scale of the lowest useful approximations normally determine the number of transform levels.

EXAMPLE 8.26: Decomposition levels in wavelet coding.

■ Table 8.14 illustrates the effect of decomposition level selection on the coding of Fig. 8.9(a) using biorthogonal wavelets and a fixed global threshold of 25. As in the previous wavelet coding example, only detail coefficients are truncated. The table lists both the percentage of zeroed coefficients and the resulting rms reconstruction errors from Eq. (8.1-10). Note that the initial decompositions are responsible for the majority of the data compression. There is little change in the number of truncated coefficients above three decomposition levels. ■

TABLE 8.13 Wavelet transform filter taps and zeroed coefficients when truncating the transforms in Fig. 8.46 below 1.5.

Wavelet	Filter Taps (Scaling + Wavelet)	Zeroed Coefficients
Haar (see Ex. 7.10)	2 + 2	33.8%
Daubechies (see Fig. 7.8)	8 + 8	40.9%
Symlet (see Fig. 7.26)	8 + 8	41.2%
Biorthogonal (see Fig. 7.39)	17 + 11	42.1%

Decomposition Level (Scales or Filter Bank Iterations)	Approximation Coefficient Image	Truncated Coefficients (%)	Reconstruction Error (rms)
1	256 × 256	74.7%	3.27
2	128 × 128	91.7%	4.23
3	64 × 64	95.1%	4.54
4	32 × 32	95.6%	4.61
5	16 × 16	95.5%	4.63

TABLE 8.14 Decomposition level impact on wavelet coding the 512 × 512 image of Fig. 8.9(a).

Quantizer design

The most important factor affecting wavelet coding compression and reconstruction error is coefficient quantization. Although the most widely used quantizers are uniform, the effectiveness of the quantization can be improved significantly by (1) introducing a larger quantization interval around zero, called a *dead zone*, or (2) adapting the size of the quantization interval from scale to scale. In either case, the selected quantization intervals must be transmitted to the decoder with the encoded image bit stream. The intervals themselves may be determined heuristically or computed automatically based on the image being compressed. For example, a global coefficient threshold could be computed as the median of the absolute values of the first-level detail coefficients or as a function of the number of zeroes that are truncated and the amount of energy that is retained in the reconstructed image.

One measure of the energy of a digital signal is the sum of the squared samples.

■ Figure 8.47 illustrates the impact of dead zone interval size on the percentage of truncated detail coefficients for a three-scale biorthogonal wavelet-based encoding of Fig. 8.9(a). As the size of the dead zone increases, the number of truncated coefficients does as well. Above the knee of the curve (i.e., beyond 5), there is little gain. This is due to the fact that the histogram of the detail coefficients is highly peaked around zero (see, for example, Fig. 7.10).

EXAMPLE 8.27: Dead zone interval selection in wavelet coding.

The rms reconstruction errors corresponding to the dead zone thresholds in Fig. 8.47 increase from 0 to 1.94 intensity levels at a threshold of 5 and to 3.83 intensity levels for a threshold of 18, where the number of zeroes reaches 93.85%. If every detail coefficient were eliminated, that percentage would increase to about 97.92% (about 4%), but the reconstruction error would grow to 12.3 intensity levels. ■

JPEG-2000

JPEG-2000 extends the popular JPEG standard to provide increased flexibility in both the compression of continuous-tone still images and access to the compressed data. For example, portions of a JPEG-2000 compressed image can be extracted for retransmission, storage, display, and/or editing. The standard is based on the wavelet coding techniques just described. Coefficient quantization is adapted to individual scales and subbands and the quantized

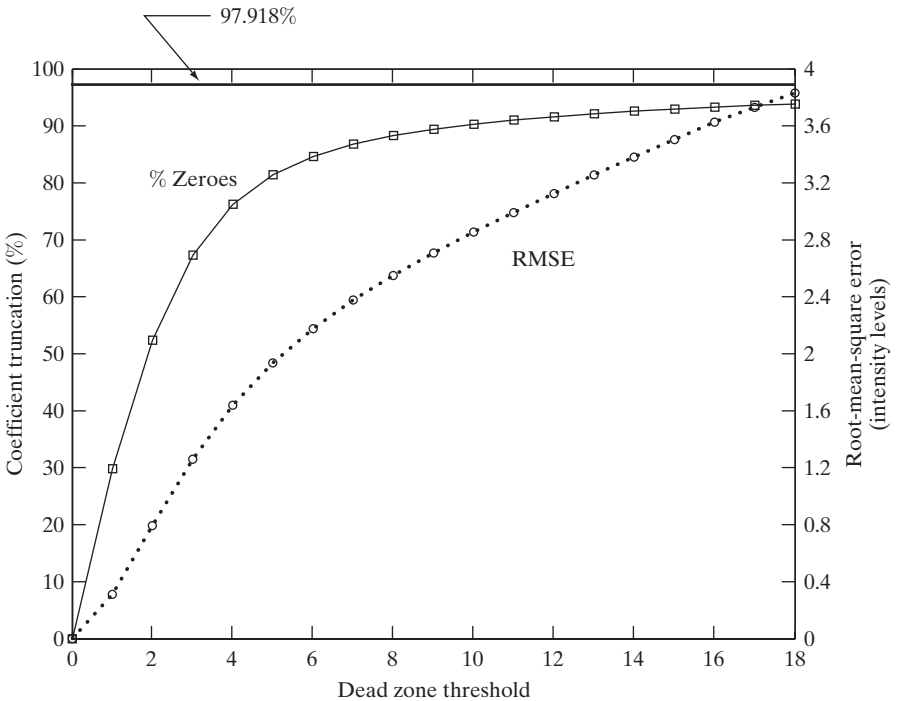


FIGURE 8.47 The impact of dead zone interval selection on wavelet coding.

coefficients are arithmetically coded on a bit-plane basis (see Sections 8.2.3 and 8.2.7). Using the notation of the standard, an image is encoded as follows (ISO/IEC [2000]).

Ssiz is used in the standard to denote intensity resolution.

The first step of the encoding process is to DC level shift the samples of the *Ssiz*-bit unsigned image to be coded by subtracting 2^{Ssiz-1}. If the image has more than one *component*—like the red, green, and blue planes of a color image—each component is shifted individually. If there are exactly three components, they may be optionally decorrelated using a reversible or nonreversible linear combination of the components. The *irreversible component transform* of the standard, for example, is

The irreversible component transform is the component transform used for lossy compression. The component transform itself is not irreversible. A different component transform is used for reversible compression.

$$\begin{aligned} Y_0(x, y) &= 0.299I_0(x, y) + 0.587I_1(x, y) + 0.114I_2(x, y) \\ Y_1(x, y) &= -0.16875I_0(x, y) - 0.33126I_1(x, y) + 0.5I_2(x, y) \\ Y_2(x, y) &= 0.5I_0(x, y) - 0.41869I_1(x, y) - 0.08131I_2(x, y) \end{aligned} \quad (8.2\text{-}61)$$

where I_0, I_1, and I_2 are the level-shifted input components and Y_0, Y_1, and Y_2 are the corresponding decorrelated components. If the input components are the red, green, and blue planes of a color image, Eq. (8.2-61) approximates the $R'G'B'$ to $Y'C_bC_r$ color video transform (Poynton [1996]).[†] The goal of the transformation is to improve compression efficiency; transformed components Y_1 and Y_2 are difference images whose histograms are highly peaked around zero.

[†] $R'G'B'$ is a gamma corrected, nonlinear version of a linear CIE (International Commission on Illumination) RGB colorimetry value. Y' is luminance and C_b and C_r are color differences (i.e., scaled $B' - Y'$ and $R' - Y'$ values).

After the image has been level shifted and optionally decorrelated, its components can be divided into *tiles*. Tiles are rectangular arrays of pixels that are processed independently. Because an image can have more than one component (e.g., it could be made up of three color components), the tiling process creates *tile components*. Each tile component can be reconstructed independently, providing a simple mechanism for accessing and/or manipulating a limited region of a coded image. For example, an image having a 16:9 aspect ratio could be subdivided into tiles so that one of its tiles is a subimage with a 4:3 aspect ratio. That tile could then be reconstructed without accessing the other tiles in the compressed image. If the image is not subdivided into tiles, it is a single tile.

The 1-D discrete wavelet transform of the rows and columns of each tile component is then computed. For error-free compression, the transform is based on a biorthogonal, 5-3 coefficient scaling and wavelet vector (Le Gall and Tabatabai [1988]). A rounding procedure is defined for non-integer-valued transform coefficients. In lossy applications, a 9-7 coefficient scaling-wavelet vector (Antonini, Barlaud, Mathieu, and Daubechies [1992]) is employed. In either case, the transform is computed using the fast wavelet transform of Section 7.4 or via a complementary *lifting-based* approach (Mallat [1999]). For example, in lossy applications, the coefficients used to construct the 9-7 FWT analysis filter bank are given in Table 8.15. The complementary lifting-based implementation involves six sequential "lifting" and "scaling" operations:

Lifting-based implementations are another way to compute wavelet transforms. The coefficients used in the approach are directly related to the FWT filter bank coefficients.

$$
\begin{aligned}
Y(2n+1) &= X(2n+1) + \alpha\big[X(2n) + X(2n+2)\big], && i_0 - 3 \le 2n + 1 < i_1 + 3\\
Y(2n) &= X(2n) + \beta\big[Y(2n-1) + Y(2n+1)\big], && i_0 - 2 \le 2n < i_1 + 2\\
Y(2n+1) &= Y(2n+1) + \gamma\big[Y(2n) + Y(2n+2)\big], && i_0 - 1 \le 2n + 1 < i_1 + 1\\
Y(2n) &= Y(2n) + \delta\big[Y(2n-1) + Y(2n+1)\big], && i_0 \le 2n < i_1\\
Y(2n+1) &= -K \cdot Y(2n+1), && i_0 \le 2n + 1 < i_1\\
Y(2n) &= Y(2n)/K, && i_0 \le 2n < i_1
\end{aligned}
\qquad (8.2\text{-}62)
$$

Here, X is the tile component being transformed, Y is the resulting transform, and i_0 and i_1 define the position of the tile component within a component. That is, they are the indices of the first sample of the tile-component row or column being transformed and the one immediately following the last sample. Variable n assumes values based on i_0, i_1, and which of the six operations is

TABLE 8.15
Impulse responses of the low- and highpass analysis filters for an irreversible 9-7 wavelet transform.

Filter Tap	Highpass Wavelet Coefficient	Lowpass Scaling Coefficient
0	−1.115087052456994	0.6029490182363579
±1	0.5912717631142470	0.2668641184428723
±2	0.05754352622849957	−0.07822326652898785
±3	−0.09127176311424948	−0.01686411844287495
±4	0	0.02674875741080976

being performed. If $n < i_0$ or $n \geq i_1$, $X(n)$ is obtained by symmetrically extending X. For example, $X(i_0 - 1) = X(i_0 + 1)$, $X(i_0 - 2) = X(i_0 + 2)$, $X(i_1) = X(i_1 - 2)$, and $X(i_1 + 1) = X(i_1 - 3)$. At the conclusion of the lifting and scaling operations, the even-indexed values of Y are equivalent to the FWT lowpass filtered output; the odd-indexed values of Y correspond to the highpass FWT filtered result. Lifting parameters α, β, γ, and δ are -1.586134342, -0.052980118, 0.882911075, and 0.433506852, respectively. Scaling factor K is 1.230174105.

These lifting-based coefficients are specified in the standard.

Recall from Chapter 7 that the DWT decomposes an image into a set of band-limited components called subbands.

The transformation just described produces four subbands—a low-resolution approximation of the tile component and the component's horizontal, vertical, and diagonal frequency characteristics. Repeating the transformation N_L times, with subsequent iterations restricted to the previous decomposition's approximation coefficients, produces an N_L-scale wavelet transform. Adjacent scales are related spatially by powers of 2 and the lowest scale contains the only explicitly defined approximation of the original tile component. As can be surmised from Fig. 8.48, where the notation of the JPEG-2000 standard is summarized for the case of $N_L = 2$, a general N_L-scale transform contains $3N_L + 1$ subbands whose coefficients are denoted a_b, for $b = N_LLL, N_LHL, \ldots, 1HL, 1LH, 1HH$. The standard does not specify the number of scales to be computed.

When each of the tile components has been processed, the total number of transform coefficients is equal to the number of samples in the original image—but the important visual information is concentrated in a few coefficients. To reduce the number of bits needed to represent the transform, coefficient $a_b(u, v)$ of subband b is quantized to value $q_b(u, v)$ using

$$q_b(u, v) = \text{sign}\left[a_b(u, v)\right] \cdot \text{floor}\left[\frac{|a_b(u, v)|}{\Delta_b}\right] \tag{8.2-63}$$

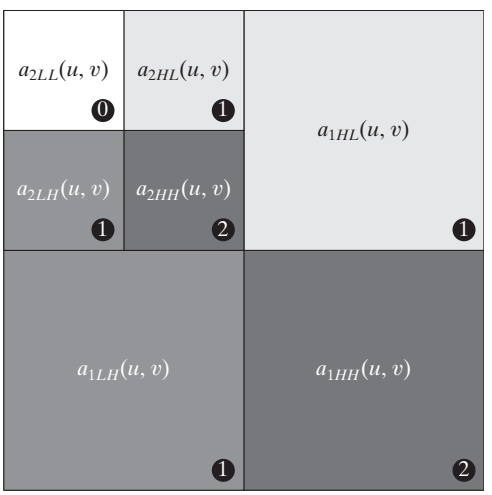


FIGURE 8.48 JPEG 2000 two-scale wavelet transform tile-component coefficient notation and analysis gain.

where the *quantiztion step size* Δ_b is

$$\Delta_b = 2^{R_b - \varepsilon_b}\left(1 + \frac{\mu_b}{2^{11}}\right) \tag{8.2-64}$$

R_b is the *nominal dynamic range* of subband b, and ε_b and μ_b are the number of bits allotted to the *exponent* and *mantissa* of the subband's coefficients. The nominal dynamic range of subband b is the sum of the number of bits used to represent the original image and the *analysis gain* bits for subband b. Subband analysis gain bits follow the simple pattern shown in Fig. 8.48. For example, there are two analysis gain bits for subband $b = 1HH$.

Do not confuse the standard's definition of nominal dynamic range with the closely related definition in Chapter 2.

For error-free compression, $\mu_b = 0$, $R_b = \varepsilon_b$, and $\Delta_b = 1$. For irreversible compression, no particular quantization step size is specified in the standard. Instead, the number of exponent and mantissa bits must be provided to the decoder on a subband basis, called *expounded quantization*, or for the $N_L LL$ subband only, called *derived quantization*. In the latter case, the remaining subbands are quantized using extrapolated $N_L LL$ subband parameters. Letting ε_0 and μ_0 be the number of bits allocated to the $N_L LL$ subband, the extrapolated parameters for subband b are

$$\begin{aligned} \mu_b &= \mu_0 \\ \varepsilon_b &= \varepsilon_0 + n_b - N_L \end{aligned} \tag{8.2-65}$$

where n_b denotes the number of subband decomposition levels from the original image tile component to subband b.

In the final steps of the encoding process, the coefficients of each transformed tile-component's subbands are arranged into rectangular blocks called *code blocks*, which are coded individually, one bit plane at a time. Starting from the most significant bit plane with a nonzero element, each bit plane is processed in three passes. Each bit (in a bit plane) is coded in only one of the three passes, which are called *significance propagation*, *magnitude refinement*, and *cleanup*. The outputs are then arithmetically coded and grouped with similar passes from other code blocks to form *layers*. A layer is an arbitrary number of groupings of coding passes from each code block. The resulting layers finally are partitioned into *packets*, providing an additional method of extracting a spatial region of interest from the total code stream. Packets are the fundamental unit of the encoded code stream.

JPEG-2000 decoders simply invert the operations described previously. After reconstructing the subbands of the tile-components from the arithmetically coded JPEG-2000 packets, a user-selected number of the subbands is decoded. Although the encoder may have encoded M_b bit planes for a particular subband, the user—due to the embedded nature of the code stream—may choose to decode only N_b bit planes. This amounts to quantizing the coefficients of the code block using a step size of $2^{M_b - N_b} \cdot \Delta_b$. Any nondecoded bits are set to zero and the resulting coefficients, denoted

Quantization as defined earlier in the chapter is irreversible. The term "inverse quantized" does not mean that there is no information loss. This process is lossy except for the case of reversible JPEG-2000 compression, where $\mu_b = 0$, $R_b = \varepsilon_b$, and $\Delta_b = 1$.

$\overline{q}_b(u, v)$, are inverse quantized using

$$R_{q_b}(u, v) = \begin{cases} (\overline{q}_b(u, v) + r \cdot 2^{M_b - N_b(u, v)}) \cdot \Delta_b & \overline{q}_b(u, v) > 0 \\ (\overline{q}_b(u, v) - r \cdot 2^{M_b - N_b(u, v)}) \cdot \Delta_b & \overline{q}_b(u, v) < 0 \\ 0 & \overline{q}_b(u, v) = 0 \end{cases} \quad (8.2\text{-}66)$$

where $R_{q_b}(u, v)$ denotes an inverse-quantized transform coefficient and $N_b(u, v)$ is the number of decoded bit planes for $\overline{q}_b(u, v)$. *Reconstruction parameter r* is chosen by the decoder to produce the best visual or objective quality of reconstruction. Generally $0 \leq r < 1$, with a common value being $r = 1/2$. The inverse-quantized coefficients then are inverse-transformed by column and by row using an FWT^{-1} filter bank whose coefficients are obtained from Table 8.15 and Eq. (7.1-11), or via the following lifting-based operations:

$$\begin{aligned}
&X(2n) = K \cdot Y(2n), && i_0 - 3 \leq 2n < i_1 + 3 \\
&X(2n+1) = (-1/K) \cdot Y(2n+1), && i_0 - 2 \leq 2n - 1 < i_1 + 2 \\
&X(2n) = X(2n) - \delta\big[X(2n-1) + X(2n+1)\big], && i_0 - 3 \leq 2n < i_1 + 3 \\
&X(2n+1) = X(2n+1) - \gamma\big[X(2n) + X(2n+2)\big], && i_0 - 2 \leq 2n + 1 < i_1 + 2 \\
&X(2n) = X(2n) - \beta\big[X(2n-1) + X(2n+1)\big], && i_0 - 1 \leq 2n < i_1 + 1 \\
&X(2n+1) = X(2n+1) - \alpha\big[X(2n) + X(2n+2)\big], && i_0 \leq 2n + 1 < i_1
\end{aligned} \quad (8.2\text{-}67)$$

where parameters α, β, γ, δ, and K are as defined for Eq. (8.2-62). Inverse-quantized coefficient row or column element $Y(n)$ is symmetrically extended when necessary. The final decoding steps are the assembly of the component tiles, inverse component transformation (if required), and DC level shifting. For irreversible coding, the inverse component transformation is

$$\begin{aligned}
I_0(x, y) &= Y_0(x, y) + 1.402Y_2(x, y) \\
I_1(x, y) &= Y_0(x, y) - 0.34413Y_1(x, y) - 0.71414Y_2(x, y) \\
I_2(x, y) &= Y_0(x, y) + 1.772Y_1(x, y)
\end{aligned} \quad (8.2\text{-}68)$$

and the transformed pixels are shifted by $+2^{Ssiz-1}$.

EXAMPLE 8.28: A comparison of JPEG-2000 wavelet-based coding and JPEG DCT-based compression.

■ Figure 8.49 shows four JPEG-2000 approximations of the monochrome image in Figure 8.9(a). Successive rows of the figure illustrate increasing levels of compression—including $C = 25, 52, 75$, and 105. The images in column 1 are decompressed JPEG-2000 encodings. The differences between these images and the original image [Fig. 8.9(a)] are shown in the second column, and the third column contains a zoomed portion of the reconstructions in column 1. Because the compression ratios for the first two rows are virtually identical to the compression ratios in Example 8.18, these results can be compared—both qualitatively and quantitatively—to the JPEG transform-based results in Figs. 8.32(a) through (f).

FIGURE 8.49 Four JPEG-2000 approximations of Fig. 8.9(a). Each row contains a result after compression and reconstruction, the scaled difference between the result and the original image, and a zoomed portion of the reconstructed image. (Compare the results in rows 1 and 2 with the JPEG results in Fig. 8.32.)

A visual comparison of the error images in rows 1 and 2 of Fig. 8.49 with the corresponding images in Figs. 8.32(b) and (e) reveals a noticeable decrease of error in the JPEG-2000 results—3.86 and 5.77 intensity levels as opposed to 5.4 and 10.7 intensity levels for the JPEG results. The computed errors favor the wavelet-based results at both compression levels. Besides decreasing reconstruction error, wavelet coding dramatically increases (in a subjective sense) image quality. Note that the blocking artifact that dominated the JPEG results [see Figs. 8.32(c) and (f)] is not present in Fig. 8.49. Finally, we note that the compression achieved in rows 3 and 4 of Fig. 8.49 is not practical with JPEG. JPEG-2000 provides useable images that are compressed by more than 100:1—with the most objectionable degradation being increased image blur. ■

8.3 Digital Image Watermarking

The methods and standards of Section 8.2 make the distribution of images (whether in photographs or videos) on digital media and over the Internet practical. Unfortunately, the images so distributed can be copied repeatedly and without error, putting the rights of their owners at risk. Even when encrypted for distribution, images are unprotected after decryption. One way to discourage illegal duplication is to insert one or more items of information, collectively called a *watermark*, into potentially vulnerable images in such a way that the watermarks are inseparable from the images themselves. As integral parts of the *watermarked images*, they protect the rights of their owners in a variety of ways, including:

1. *Copyright identification.* Watermarks can provide information that serves as proof of ownership when the rights of the owner have been infringed.
2. *User identification* or *fingerprinting.* The identity of legal users can be encoded in watermarks and used to identify sources of illegal copies.
3. *Authenticity determination.* The presence of a watermark can guarantee that an image has not been altered—assuming the watermark is designed to be destroyed by any modification of the image.
4. *Automated monitoring.* Watermarks can be monitored by systems that track when and where images are used (e.g., programs that search the Web for images placed on Web pages). Monitoring is useful for royalty collection and/or the location of illegal users.
5. *Copy protection.* Watermarks can specify rules of image usage and copying (e.g., to DVD players).

In this section, we provide a brief overview of *digital image watermarking*—the process of inserting data into an image in such a way that it can be used to make an assertion about the image. The methods described have little in common with the compression techniques presented in the previous sections—although they do involve the coding of information. In fact, watermarking and compression are in some ways opposites. While the objective in compression is to reduce the amount of data used to represent images, the goal in watermarking is to add information and thus data (i.e., watermarks) to them. As will be seen in

the remainder of the section, the watermarks themselves can be either visible or invisible.

A *visible watermark* is an opaque or semi-transparent sub-image or image that is placed on top of another image (i.e., the image being watermarked) so that it is obvious to the viewer. Television networks often place visible watermarks (fashioned after their logos) in the upper- or lower-right hand corner of the television screen. As the following example illustrates, visible watermarking typically is performed in the spatial domain.

EXAMPLE 8.29: A simple visible watermark.

■ The image in Fig. 8.50(b) is the lower-right-hand quadrant of the image in Fig. 8.9(a) with a scaled version of the watermark in Fig. 8.50(a) overlaid on top of it. Letting f_w denote the watermarked image, we can express it as a linear combination of the unmarked image f and watermark w using

$$f_w = (1 - \alpha)f + \alpha w \tag{8.3-1}$$

where constant α controls the relative visibility of the watermark and the underlying image. If α is 1, the watermark is opaque and the underlying image is completely obscured. As α approaches 0, more of the underlying image and less of the watermark is seen. In general, $0 < \alpha \leq 1$; in Fig. 8.50(b), $\alpha = 0.3$. Figure 8.50(c) is the computed difference (scaled in intensity) between the watermarked image in (b) and the unmarked image in Fig. 8.9(a). Intensity 128 represents a difference of 0. Note that the underlying image is clearly visible through the "semi-transparent" watermark. This is evident in both Fig. 8.50(b) and the difference image in (c). ■

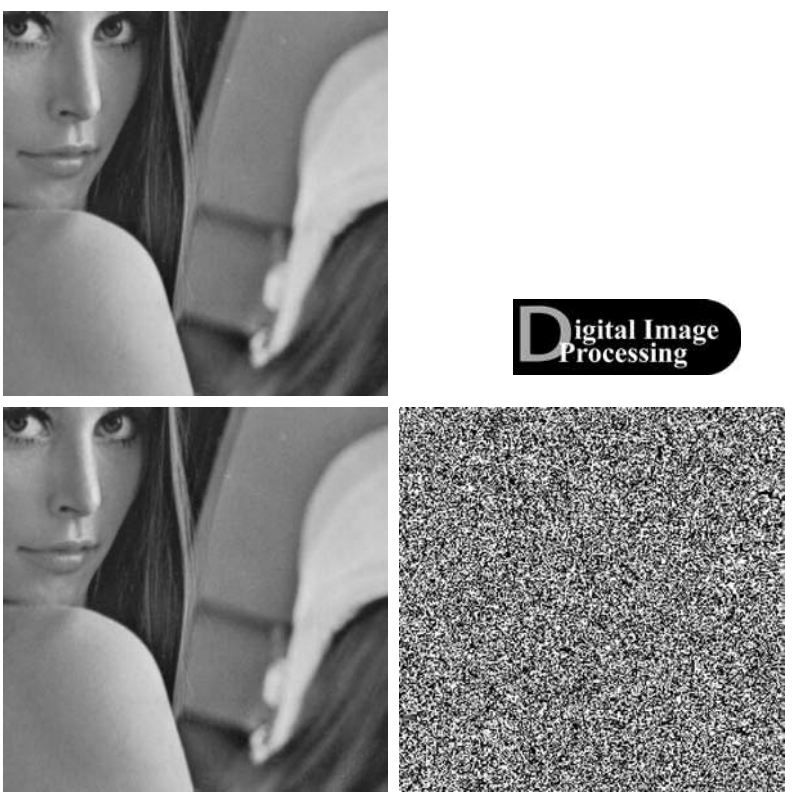
a
b c

FIGURE 8.50 A simple visible watermark: (a) watermark; (b) the watermarked image; and (c) the difference between the watermarked image and the original (non-watermarked) image.

Unlike the visible watermark of the previous example, *invisible watermarks* cannot be seen with the naked eye. They are imperceptible—but can be recovered with an appropriate decoding algorithm. Invisibility is assured by inserting them as visually redundant information—as information that the human visual system ignores or cannot perceive (see Section 8.1.3). Figure 8.51(a) provides a simple example. Because the least significant bits of an 8-bit image have virtually no effect on our perception of the image, the watermark from Fig. 8.50(a) was inserted or "hidden" in its two least significant bits. Using the notation introduced above, we let

$$f_w = 4\left(\frac{f}{4}\right) + \frac{w}{64} \tag{8.3-2}$$

and use unsigned integer arithmetic to perform the calculations. Dividing and multiplying by 4 sets the two least significant bits of f to 0, dividing w by 64 shifts its two most significant bits into the two least significant bit positions, and adding the two results generates the *LSB watermarked image*. Note that the embedded watermark is not visible in Fig. 8.51(a). By zeroing the most significant 6 bits of this image and scaling the remaining values to the full intensity range, however, the watermark can be extracted as in Fig. 8.51(b).



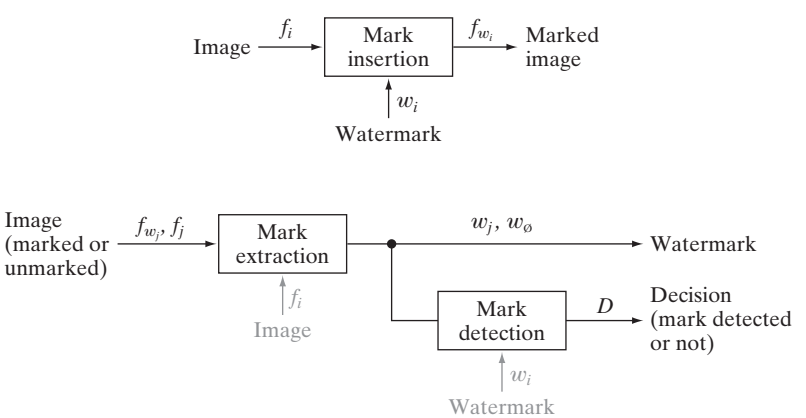
a b
c d

FIGURE 8.51 A simple invisible watermark: (a) watermarked image; (b) the extracted watermark; (c) the watermarked image after high quality JPEG compression and decompression; and (d) the extracted watermark from (c).

An important property of invisible watermarks is their resistance to both accidental and intentional attempts to remove them. *Fragile invisible watermarks* are destroyed by any modification of the images in which they are embedded. In some applications, like image authentication, this is a desirable characteristic. As Figs. 8.51(c) and (d) show, the LSB watermarked image in Fig. 8.51(a) contains a fragile invisible watermark. If the image in (a) is compressed and decompressed using lossy JPEG, the watermark is destroyed. Figure 8.51(c) is the result after compressing and decompressing Fig. 8.51(a); the rms error is 2.1 bits. If we try to extract the watermark from this image using the same method as in (b), the result is unintelligible [see Fig. 8.51(d)]. Although lossy compression and decompression preserved the important visual information in the image, the fragile watermark was destroyed.

Robust invisible watermarks are designed to survive image modification, whether the so called *attacks* are inadvertent or intentional. Common inadvertent attacks include lossy compression, linear and non-linear filtering, cropping, rotation, resampling, and the like. Intentional attacks range from printing and rescanning to adding additional watermarks and/or noise. Of course, it is unnecessary to withstand attacks that leave the image itself unusable.

Figure 8.52 shows the basic components of a typical image watermarking system. The encoder in Fig. 8.52(a) inserts watermark w_i into image f_i, producing watermarked image f_{w_i}; the complementary decoder in (b) extracts and validates the presence of w_i in watermarked input f_{w_j} or unmarked input f_j. If w_i is visible, the decoder is not needed. If it is invisible, the decoder may or may not require a copy of f_i and w_i [shown in gray in Fig. 8.52(b)] to do its job. If f_i and/or w_i are used, the watermarking system is known as a *private* or *restricted-key* system; if not, it is a *public* or *unrestricted-key* system. Because the decoder must process both marked and unmarked images, $w_\varnothing$ is used in Fig. 8.52(b) to denote the absence of a mark. Finally, we note that to determine the presence of w_i in an image, the decoder must correlate extracted watermark w_j with w_i and compare the result to a predefined threshold. The threshold sets the degree of similarity that is acceptable for a "match."



a
b

FIGURE 8.52
A typical image watermarking system:
(a) encoder;
(b) decoder.

EXAMPLE 8.30: A DCT-based invisible robust watermark.

■ *Mark insertion* and *extraction* can be performed in the spatial domain, as in the previous examples, or in the transform domain. Figures 8.53(a) and (c) show two watermarked versions of the image in Fig. 8.9(a) using the DCT-based watermarking approach outlined below (Cox et al. [1997]):

Step 1. Compute the 2-D DCT of the image to be watermarked.

Step 2. Locate its K largest coefficients, $c_1, c_2, \ldots, c_K$, by magnitude.

a b
c d

FIGURE 8.53 (a) and (c) Two watermarked versions of Fig. 8.9(a); (b) and (d) the differences (scaled in intensity) between the watermarked versions and the unmarked image. These two images show the intensity contribution (although scaled dramatically) of the pseudo-random watermarks on the original image.

Step 3. Create a watermark by generating a K-element pseudo-random sequence of numbers, $\omega_1, \omega_2, \ldots, \omega_K$, taken from a Gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$.

A pseudo-random number sequence approximates the properties of random numbers. It is not truly random because it depends on a predetermined initial value.

Step 4. Embed the watermark from step 3 into the K largest DCT coefficients from step 2 using the following equation

$$c_i' = c_i \cdot (1 + \alpha\omega_i) \quad 1 \le i \le K \tag{8.3-3}$$

for a specified constant $\alpha > 0$ (that controls the extent to which ω_i alters c_i). Replace the original c_i with the computed c_i' from Eq. (8.3-3).

For the images in Fig. 8.53, $\alpha = 0.1$ and $K = 1000$.

Step 5. Compute the inverse DCT of the result from step 4.

By employing watermarks made from pseudo-random numbers and spreading them across an image's perceptually significant frequency components, α can be made small, reducing watermark visibility. At the same time, watermark security is kept high because (1) the watermarks are composed of pseudo-random numbers with no obvious structure, (2) the watermarks are embedded in multiple frequency components with spatial impact over the entire 2-D image (so their location is not obvious) and (3) attacks against them tend to degrade the image as well (i.e., the image's most important frequency components must be altered to affect the watermarks).

Figures 8.53(b) and (d) make the changes in image intensity that result from the pseudo-random numbers that are embedded in the DCT coefficients of the watermarked images in Figs. 8.53(a) and (c) visible. Obviously, the pseudo-random numbers must have an effect—even if too small to see—on the watermarked images. To display the effect, the images in Figs. 8.53(a) and (c) were subtracted from the unmarked image in Fig. 8.9(a) and scaled in intensity to the range [0, 255]. Figures 8.53(b) and (d) are the resulting images; they show the 2-D spatial contributions of the pseudo-random numbers. Because they have been scaled, however, you cannot simply add these images to the image in Fig. 8.9(a) and get the watermarked images in Figs. 8.53(a) and (c). As can be seen in Figs. 8.53(a) and (c), their actual intensity perturbations are small to negligible.

To determine whether a particular image is a copy of a previously watermarked image with watermark $\omega_1, \omega_2, \ldots, \omega_K$ and DCT coefficients $c_1, c_2, \ldots, c_K$, we use the following procedure:

Step 1. Compute the 2-D DCT of the image in question.

Step 2. Extract the K DCT coefficients (in the positions corresponding to $c_1, c_2, \ldots, c_K$ of step 2 in the watermarking procedure) and denote the coefficients as $\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_K$. If the image in question is the previously watermarked image (without modification), $\hat{c}_i = c_i'$ for $1 \le i \le K$. If it is a modified copy of the watermarked image (i.e., it has undergone some sort of attack), $\hat{c}_i \approx c_i'$ for $1 \le i \le K$ (the $\hat{c}_i$ will be approximations of the c_i'). Otherwise, the image in question will be an unmarked image or an image with a completely different watermark—and the $\hat{c}_i$ will bear no resemblance to the original c_i'.

Step 3. Compute watermark $\hat{\omega}_1, \hat{\omega}_2, \ldots, \hat{\omega}_K$ using

$$\hat{\omega}_i = \hat{c}_i - c_i \quad \text{for} \quad 1 \le i \le K \tag{8.3-4}$$

Recall that watermarks are a sequence of pseudo-random numbers.

Step 4. Measure the similarity of $\hat{\omega}_1, \hat{\omega}_2, \ldots, \hat{\omega}_K$ (from step 2) and $\omega_1, \omega_2, \ldots, \omega_K$ (from step 3 of the watermarking procedure) using a metric such as the correlation coefficient

We discuss the correlation coefficient in detail in Section 12.2.1.

$$\gamma = \frac{\sum_{i=1}^{K}(\hat{\omega}_i - \overline{\hat{\omega}})(\omega_i - \overline{\omega})}{\sqrt{\sum_{i=1}^{K}(\hat{\omega}_i - \overline{\hat{\omega}})^2 \cdot \sum_{i=1}^{K}(\omega_i - \overline{\omega})^2}} \quad 1 \le i \le K \tag{8.3-5}$$

where $\overline{\omega}$ and $\overline{\hat{\omega}}$ are the means of the two K-element watermarks.

Step 5. Compare the measured similarity, γ, to a predefined threshold, T, and make a binary detection decision

$$D = \begin{cases} 1 & \text{if } \gamma \ge T \\ 0 & \text{otherwise} \end{cases} \tag{8.3-6}$$

In other words, $D = 1$ indicates that watermark $\omega_1, \omega_2, \ldots, \omega_K$ is present (with respect to the specified threshold, T); $D = 0$ indicates that it was not.

Using this procedure, the original watermarked image in Fig. 8.53(a)—measured against itself—yields a correlation coefficient of 0.9999, i.e., $\gamma = 0.9999$. It is an unmistakable match. In a similar manner, the image in Fig. 8.53(b), when measured against the image in Fig. 8.53(a), results in a γ of 0.0417—it could not be mistaken for the watermarked image in Fig. 8.53(a) because the correlation coefficient is so low. ■

To conclude the section, we note that the DCT-based watermarking approach of the previous example is fairly resistant to watermark attacks, partly because it is a private or restricted-key method. Restricted-key methods are always more resilient than their unrestricted-key counterparts. Using the watermarked image in Fig. 8.53(a), Fig. 8.54 illustrates the ability of the method to withstand a variety of common attacks. As can be seen in the figure, watermark detection is quite good over the range of attacks that were implemented—the resulting correlation coefficients (shown under each image in the figure) vary from 0.3113 to 0.9945. When subjected to a high quality but lossy (resulting in an rms error of 7 intensities) JPEG compression and decompression, $\gamma = 0.9945$. Even when the compression and reconstructed yields an rms error of 10 intensity levels, $\gamma = 0.7395$—and the usability of this image has been significantly degraded. Significant smoothing by spatial filtering and the addition of Gaussian noise do not reduce the correlation coefficient below 0.8230. However, histogram equalization reduces γ to 0.5210; and rotation has the largest effect—reducing γ to 0.3313. All attacks, except for the lossy JPEG

a b c
d e f

FIGURE 8.54 Attacks on the watermarked image in Fig. 8.53(a): (a) lossy JPEG compression and decompression with an rms error of 7 intensity levels; (b) lossy JPEG compression and decompression with an rms error of 10 intensity levels (note the blocking artifact); (c) smoothing by spatial filtering; (d) the addition of Gaussian noise; (e) histogram equalization; and (f) rotation. Each image is a modified version of the watermarked image in Fig. 8.53(a). After modification, they retain their watermarks to varying degrees, as indicated by the correlation coefficients below each image.

compression and reconstruction in (a), have significantly reduced the usability of the original watermarked image.

Summary

The principal objectives of this chapter were to present the theoretic foundation of digital image compression, to describe the most commonly used compression methods, and to introduce the related area of digital image watermarking. Although the level of the presentation is introductory in nature, the references provide an entry into the extensive body of literature dealing with the topics discussed. As evidenced by the international standards listed in Tables 8.3 and 8.4, compression plays a key role in

7 Wavelets and Multiresolution Processing

All this time, the guard was looking at her, first through a telescope, then through a microscope, and then through an opera glass.

Lewis Carrol, Through the Looking Glass

Preview

Although the Fourier transform has been the mainstay of transform-based image processing since the late 1950s, a more recent transformation, called the *wavelet transform*, is now making it even easier to compress, transmit, and analyze many images. Unlike the Fourier transform, whose basis functions are sinusoids, wavelet transforms are based on small waves, called *wavelets*, of varying frequency *and limited duration*. This allows them to provide the equivalent of a musical score for an image, revealing not only what notes (or frequencies) to play but also when to play them. Fourier transforms, on the other hand, provide only the notes or frequency information; temporal information is lost in the transformation process.

In 1987, wavelets were first shown to be the foundation of a powerful new approach to signal processing and analysis called *multiresolution* theory (Mallat [1987]). Multiresolution theory incorporates and unifies techniques from a variety of disciplines, including subband coding from signal processing, quadrature mirror filtering from digital speech recognition, and pyramidal image processing. As its name implies, multiresolution theory is concerned with the representation and analysis of signals (or images) at more than one resolution. The appeal of such an approach is obvious—features that might go undetected at one resolution may be easy to detect at another. Although the imaging community's interest in multiresolution analysis was limited until the late 1980s, it is now difficult to keep up with the number of papers, theses, and books devoted to the subject.

In this chapter, we examine wavelet-based transformations from a multiresolution point of view. Although such transformations can be presented in other ways, this approach simplifies both their mathematical and physical interpretations. We begin with an overview of imaging techniques that influenced the formulation of multiresolution theory. Our objective is to introduce the theory's fundamental concepts within the context of image processing and simultaneously provide a brief historical perspective of the method and its application. The bulk of the chapter is focused on the development and use of the discrete wavelet transform. To demonstrate the usefulness of the transform, examples ranging from image coding to noise removal and edge detection are provided. In the next chapter, wavelets will be used for image compression, an application in which they have received considerable attention.

7.1 Background

When we look at images, generally we see connected regions of similar texture and intensity levels that combine to form objects. If the objects are small in size or low in contrast, we normally examine them at high resolutions; if they are large in size or high in contrast, a coarse view is all that is required. If both small and large objects—or low- and high-contrast objects—are present simultaneously, it can be advantageous to study them at several resolutions. This, of course, is the fundamental motivation for multiresolution processing.

From a mathematical viewpoint, images are two-dimensional arrays of intensity values with locally varying statistics that result from different combinations of abrupt features like edges and contrasting homogeneous regions. As illustrated in Fig. 7.1—an image that will be examined repeatedly in the remainder of the

Local histograms are histograms of the pixels in a neighborhood (see Section 3.3.3).

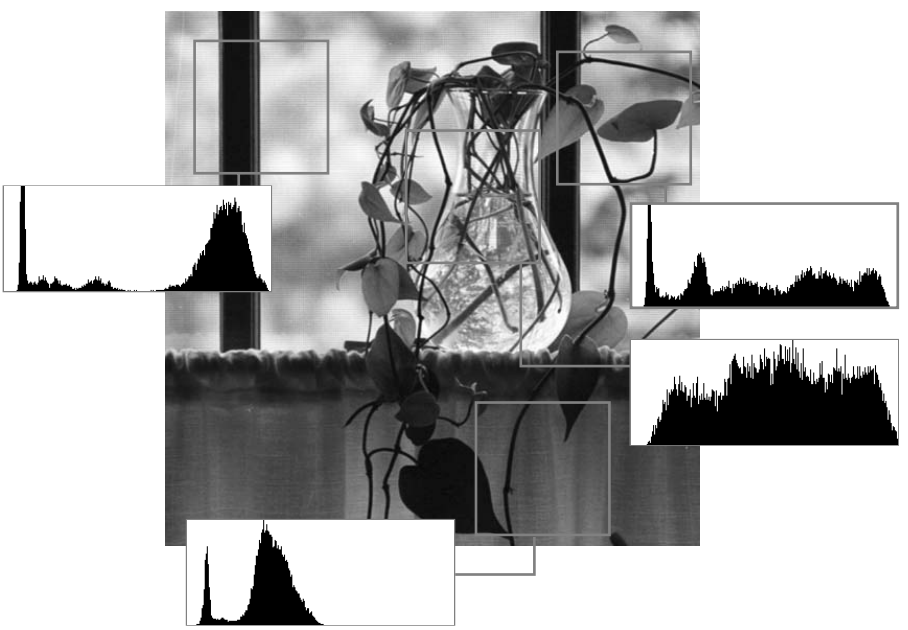


FIGURE 7.1 An image and its local histogram variations.

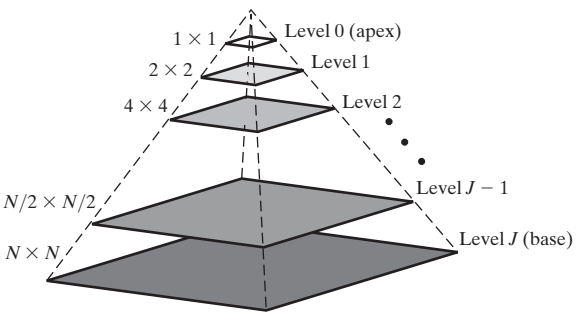
section—local histograms can vary significantly from one part of an image to another, making statistical modeling over the span of an entire image a difficult, or impossible task.

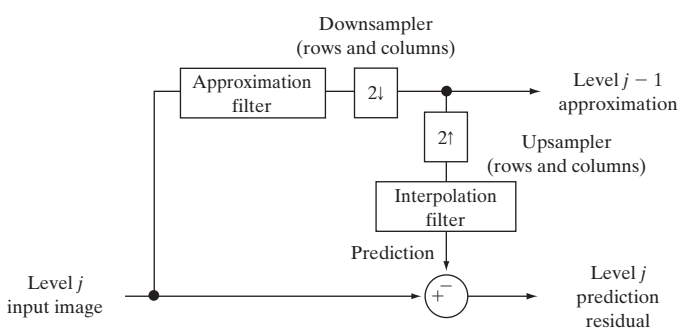
7.1.1 Image Pyramids

A powerful, yet conceptually simple structure for representing images at more than one resolution is the *image pyramid* (Burt and Adelson [1983]). Originally devised for machine vision and image compression applications, an image pyramid is a collection of decreasing resolution images arranged in the shape of a pyramid. As can be seen in Fig. 7.2(a), the base of the pyramid contains a high-resolution representation of the image being processed; the apex contains a low-resolution approximation. As you move up the pyramid, both size and resolution decrease. Base level J is of size $2^J \times 2^J$ or $N \times N$, where $J = \log_2 N$, apex level 0 is of size 1×1, and general level j is of size $2^j \times 2^j$, where $0 \leq j \leq J$. Although the pyramid shown in Fig. 7.2(a) is composed of $J + 1$ resolution levels from $2^J \times 2^J$ to $2^0 \times 2^0$, most image pyramids are truncated to $P + 1$ levels, where $1 \leq P \leq J$ and $j = J - P, \ldots, J - 2, J - 1, J$. That is, we normally limit ourselves to P reduced resolution approximations of the original image; a 1×1 (i.e., single pixel) approximation of a 512×512 image, for example, is of little value. The total number of pixels in a $P + 1$ level pyramid for $P > 0$ is

$$N^2\left(1 + \frac{1}{(4)^1} + \frac{1}{(4)^2} + \cdots + \frac{1}{(4)^P}\right) \leq \frac{4}{3}N^2$$

Figure 7.2(b) shows a simple system for constructing two intimately related image pyramids. The *Level $j - 1$ approximation* output provides the images





a
b

FIGURE 7.2 (a) An image pyramid. (b) A simple system for creating approximation and prediction residual pyramids.

needed to build an *approximation pyramid* (as described in the preceding paragraph), while the *Level j prediction residual* output is used to build a complementary *prediction residual pyramid*. Unlike approximation pyramids, prediction residual pyramids contain only one reduced-resolution approximation of the input image (at the top of the pyramid, level $J - P$). All other levels contain prediction residuals, where the level j *prediction residual* (for $J - P + 1 \leq j \leq J$) is defined as the difference between the level j approximation (the input to the block diagram) and an estimate of the level j approximation based on the level $j - 1$ approximation (the approximation output in the block diagram).

In general, a prediction residual can be defined as the difference between an image and a predicted version of the image. As will be seen in Section 8.2.9, prediction residuals can often be coded more efficiently than 2-D intensity arrays.

As Fig. 7.2(b) suggests, both approximation and prediction residual pyramids are computed in an iterative fashion. Before the first iteration, the image to be represented in pyramidal form is placed in level J of the approximation pyramid. The following three-step procedure is then executed P times—for $j = J, J - 1, \ldots,$ and $J - P + 1$ (in that order):

Step 1. Compute a reduced-resolution approximation of the *Level j input image* [the input on the left side of the block diagram in Fig. 7.2(b)]. This is done by filtering and downsampling the filtered result by a factor of 2. Both of these operations are described in the next paragraph. Place the resulting approximation at level $j - 1$ of the approximation pyramid.

Step 2. Create an estimate of the *Level j input image* from the reduced-resolution approximation generated in step 1. This is done by upsampling and filtering (see the next paragraph) the generated approximation. The resulting prediction image will have the same dimensions as the *Level j input image*.

Step 3. Compute the difference between the prediction image of step 2 and the input to step 1. Place this result in level j of the prediction residual pyramid.

At the conclusion of P iterations (i.e., following the iteration in which $j = J - P + 1$), the level $J - P$ approximation output is placed in the prediction residual pyramid at level $J - P$. If a prediction residual pyramid is not needed, this operation—along with steps 2 and 3 and the upsampler, interpolation filter, and summer of Fig. 7.2(b)—can be omitted.

A variety of approximation and interpolation filters can be incorporated into the system of Fig. 7.2(b). Typically, the filtering is performed in the spatial domain (see Section 3.4). Useful approximation filtering techniques include neighborhood averaging (see Section 3.5.1.), which produces *mean pyramids*; lowpass Gaussian filtering (see Sections 4.7.4 and 4.8.3), which produces *Gaussian pyramids*; and no filtering, which results in *subsampling pyramids*. Any of the interpolation methods described in Section 2.4.4, including nearest neighbor, bilinear, and bicubic, can be incorporated into the interpolation filter. Finally, we note that the upsampling and downsampling blocks of Fig. 7.2(b) are used to double and halve the spatial dimensions of the approximation and prediction images that are computed. Given an integer variable n and 1-D sequence of samples $f(n)$, *upsampled* sequence $f_{2\uparrow}(n)$ is defined as

$$f_{2\uparrow}(n) = \begin{cases} f(n/2) & \text{if } n \text{ is even} \\ 0 & \text{otherwise} \end{cases} \tag{7.1-1}$$

In this chapter, we will be working with both continuous and discrete functions and variables. With the notable exception of 2-D image $f(x, y)$ and unless otherwise noted, $x, y, z, \ldots$ are continuous variables; $i, j, k, l, m, n, \ldots$ are discrete variables.

where, as is indicated by the subscript, the upsampling is by a factor of 2. The complementary operation of *downsampling* by 2 is defined as

$$f_{2\downarrow}(n) = f(2n) \tag{7.1-2}$$

Upsampling can be thought of as inserting a 0 after every sample in a sequence; downsampling can be viewed as discarding every other sample. The upsampling and downsampling blocks in Fig. 7.2(b), which are labeled $2\uparrow$ and $2\downarrow$, respectively, are annotated to indicate that both the rows and columns of the 2-D inputs on which they operate are to be up- and downsampled. Like the separable 2-D DFT in Section 4.11.1, 2-D upsampling and downsampling can be performed by successive passes of the 1-D operations defined in Eqs. (7.1-1) and (7.1-2).

EXAMPLE 7.1: Approximation and prediction residual pyramids.

■ Figure 7.3 shows both an approximation pyramid and a prediction residual pyramid for the vase of Fig. 7.1. A lowpass Gaussian smoothing filter (see Section 4.7.4) was used to produce the four-level approximation pyramid in Fig. 7.3(a). As you can see, the resulting pyramid contains the original 512×512 resolution image (at its base) and three low-resolution approximations (of resolution 256×256, 128×128, and 64×64). Thus, P is 3 and levels 9, 8, 7, and 6 out of a possible $\log_2(512) + 1$ or 10 levels are present. Note the reduction in detail that accompanies the lower resolutions of the pyramid. The level 6 (i.e., 64×64) approximation image is suitable for locating the window stiles (i.e., the window pane framing), for example, but not for finding the stems of the plant. In general, the lower-resolution levels of a pyramid can be used for the analysis of large structures or overall image context; the high-resolution images are appropriate for analyzing individual object characteristics. Such a coarse-to-fine analysis strategy is particularly useful in pattern recognition.

A bilinear interpolation filter was used to produce the prediction residual pyramid in Fig. 7.3(b). In the absence of quantization error, the resulting prediction residual pyramid can be used to generate the complementary approximation pyramid in Fig. 7.3(a), including the original image, without error. To do so, we begin with the level 6 64×64 approximation image (the only approximation image in the prediction residual pyramid), predict the level 7 128×128 resolution approximation (by upsampling and filtering), and add the level 7 prediction residual. This process is repeated using successively computed approximation images until the original 512×512 image is generated. Note that the prediction residual histogram in Fig. 7.3(b) is highly peaked around zero; the approximation histogram in Fig. 7.3(a) is not. Unlike approximation images, prediction residual images can be highly compressed by assigning fewer bits to the more probable values (see the variable-length codes of Section 8.2.1). Finally, we note that the prediction residuals in Fig. 7.3(b) are scaled to make small prediction errors more visible; the prediction residual histogram, however, is based on the original residual values, with level 128 representing zero error. ■

a
b

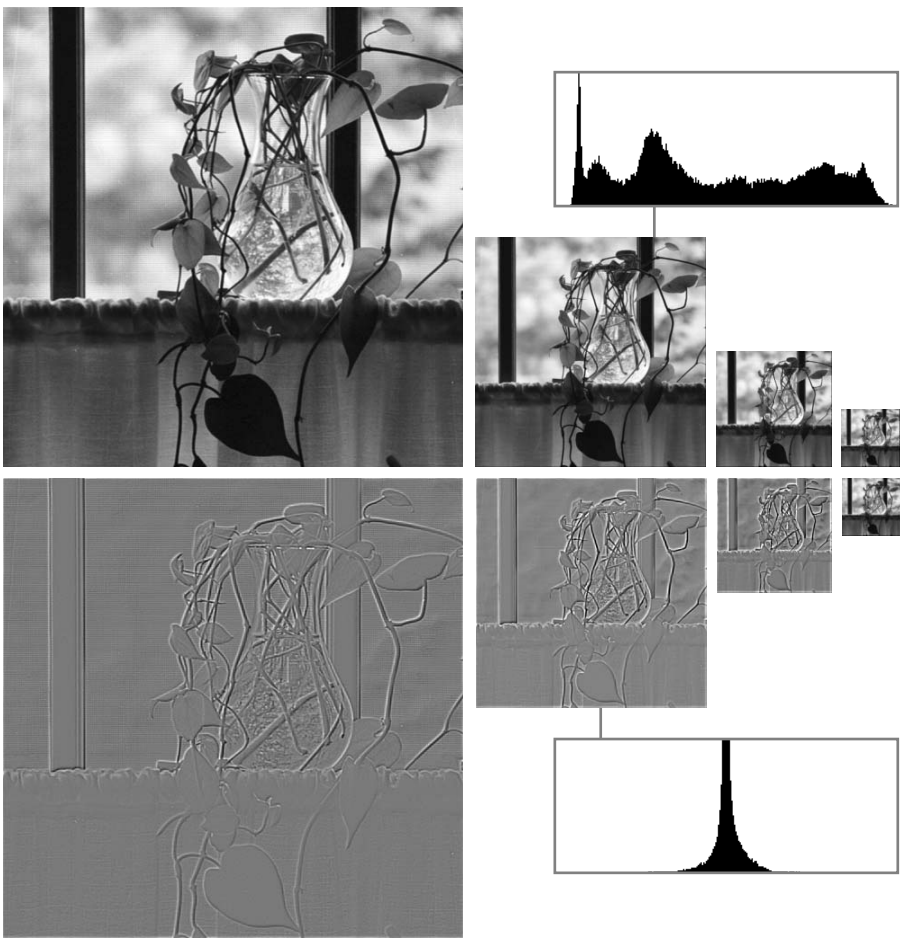


FIGURE 7.3
Two image pyramids and their histograms: (a) an approximation pyramid; (b) a prediction residual pyramid.

The approximation pyramid in (a) is called a Gaussian pyramid because a Gaussian filter was used to construct it. The prediction residual pyramid in (b) is often called a Laplacian pyramid; note the similarity in appearance with the Laplacian filtered images in Chapter 3.

7.1.2 Subband Coding

Another important imaging technique with ties to multiresolution analysis is *subband coding*. In subband coding, an image is decomposed into a set of bandlimited components, called subbands. The decomposition is performed so that the subbands can be reassembled to reconstruct the original image without error. Because the decomposition and reconstruction are performed by means of digital filters, we begin our discussion with a brief introduction to *digital signal processing* (DSP) and *digital signal filtering*.

The term "delay" implies a time-based input sequence and reflects the fact that in digital signal filtering, the input is usually a sampled analog signal.

Consider the simple *digital filter* in Fig. 7.4(a) and note that it is constructed from three basic components—*unit delays*, *multipliers*, and *adders*. Along the top of the filter, unit delays are connected in series to create $K-1$ delayed (i.e., right shifted) versions of the input sequence $f(n)$. Delayed sequence $f(n-2)$, for example, is

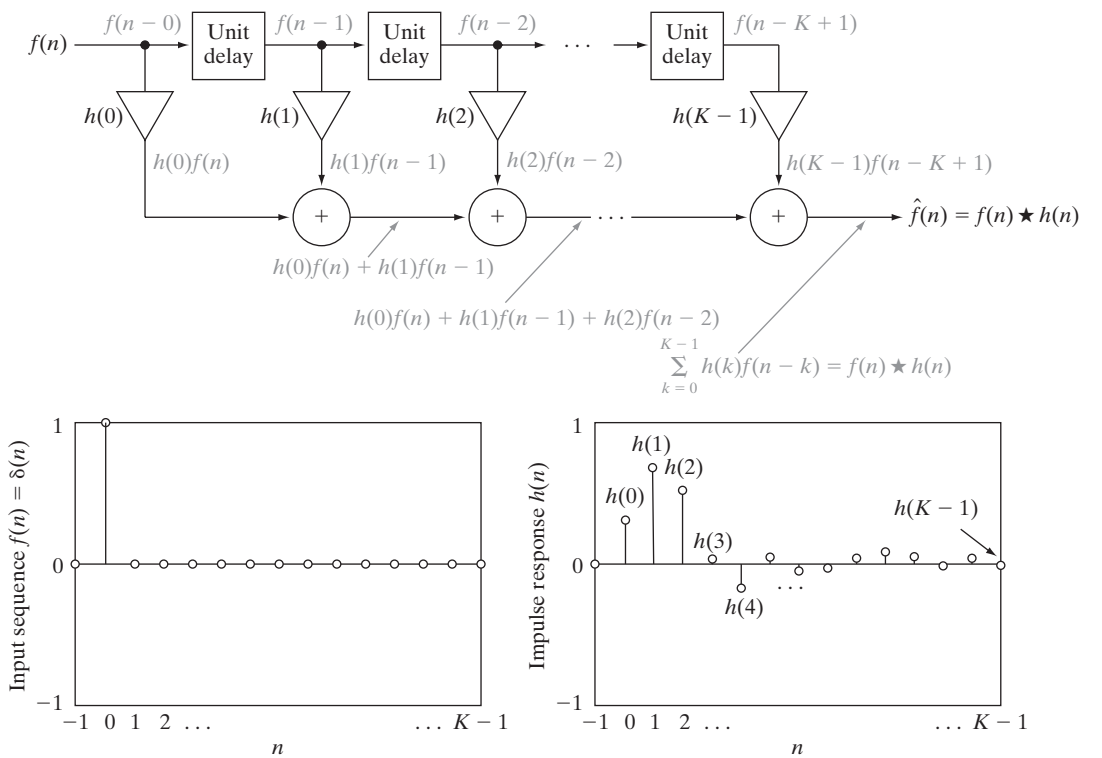
$$f(n-2) = \begin{cases} \vdots & \\ f(0) & \text{for } n = 2 \\ f(1) & \text{for } n = 2 + 1 = 3 \\ \vdots & \end{cases}$$

As the grayed annotations in Fig. 7.4(a) indicate, input sequence $f(n) = f(n-0)$ and the $K-1$ delayed sequences at the outputs of the unit delays, denoted $f(n-1), f(n-2), \ldots, f(n-K+1)$, are multiplied by constants $h(0), h(1), \ldots, h(K-1)$, respectively, and summed to produce the filtered output sequence

$$\begin{aligned} \hat{f}(n) &= \sum_{k=-\infty}^{\infty} h(k) f(n-k) \\ &= f(n) \star h(n) \end{aligned} \tag{7.1-3}$$

where ★ denotes convolution. Note that—except for a change in variables—Eq. (7.1-3) is equivalent to the discrete convolution defined in Eq. (4.4-10) of Chapter 4. The K multiplication constants in Fig. 7.4(a) and Eq. (7.1-3) are

If the coefficients of the filter in Fig. 7.4(a) are indexed using values of n between 0 and $K-1$ (as we have done), the limits on the sum in Eq. (7.1-3) can be reduced to 0 to $K-1$ [like Eq. (4.4-10)].



a
b c

FIGURE 7.4 (a) A digital filter; (b) a unit discrete impulse sequence; and (c) the impulse response of the filter.

called *filter coefficients*. Each coefficient defines a *filter tap*, which can be thought of as the components needed to compute one term of the sum in Eq. (7.1-3), and the filter is said to be of *order K*.

If the input to the filter of Fig. 7.4(a) is the unit discrete impulse of Fig. 7.4(b) and Section 4.2.3, Eq. (7.1-3) becomes

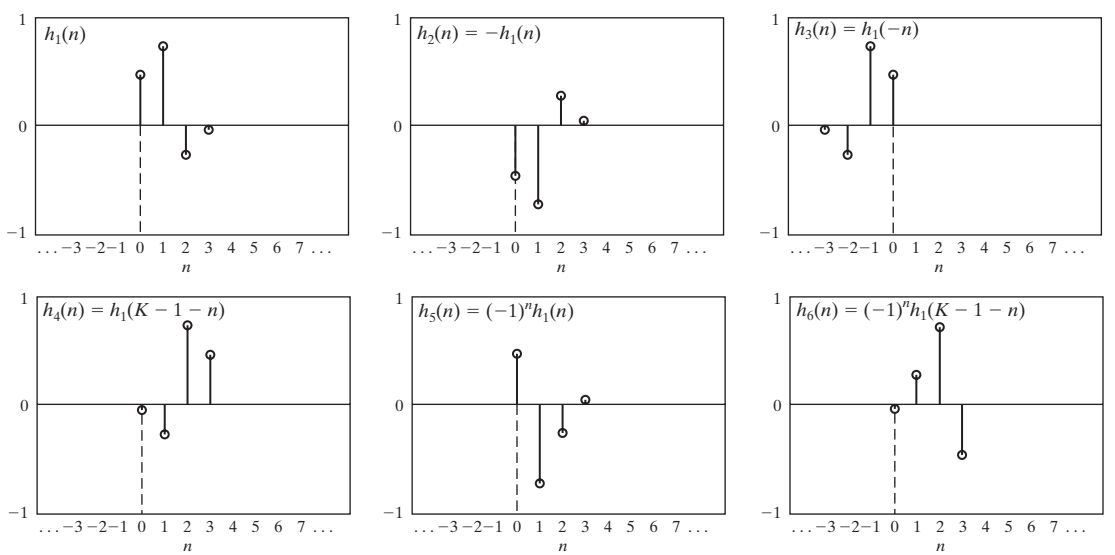
$$\begin{aligned}\hat{f}(n) &= \sum_{k=-\infty}^{\infty} h(k)\delta(n-k) \\ &= h(n)\end{aligned} \tag{7.1-4}$$

That is, by substituting $\delta(n)$ for input $f(n)$ in Eq. (7.1-3) and making use of the sifting property of the unit discrete impulse as defined in Eq. (4.2-13), we find that the *impulse response* of the filter in Fig. 7.4(a) is the K-element sequence of filter coefficients that define the filter. Physically, the unit impulse is shifted from left to right across the top of the filter (from one unit delay to the next), producing an output that assumes the value of the coefficient at the location of the delayed impulse. Because there are K coefficients, the impulse response is of length K and the filter is called a *finite impulse response* (FIR) filter.

In the remainder of the chapter, "filter $h(n)$" will be used to refer to the filter whose impulse response is $h(n)$.

Figure 7.5 shows the impulse responses of six functionally related filters. Filter $h_2(n)$ in Fig. 7.5(b) is a *sign-reversed* (i.e., reflected about the horizontal axis) version of $h_1(n)$ in Fig. 7.5(a). That is,

$$h_2(n) = -h_1(n) \tag{7.1-5}$$



a b c
d e f

FIGURE 7.5 Six functionally related filter impulse responses: (a) reference response; (b) sign reversal; (c) and (d) order reversal (differing by the delay introduced); (e) modulation; and (f) order reversal and modulation.

Filters $h_3(n)$ and $h_4(n)$ in Figs. 7.5(c) and (d) are *order-reversed* versions of $h_1(n)$:

Order reversal is often called *time reversal* when the input sequence is a sampled analog signal.

$$h_3(n) = h_1(-n) \tag{7.1-6}$$

$$h_4(n) = h_1(K - 1 - n) \tag{7.1-7}$$

Filter $h_3(n)$ is a reflection of $h_1(n)$ about the vertical axis; filter $h_4(n)$ is a reflected and translated (i.e., shifted) version of $h_1(n)$. Neglecting translation, the responses of the two filters are identical. Filter $h_5(n)$ in Fig. 7.5(e), which is defined as

$$h_5(n) = (-1)^n h_1(n) \tag{7.1-8}$$

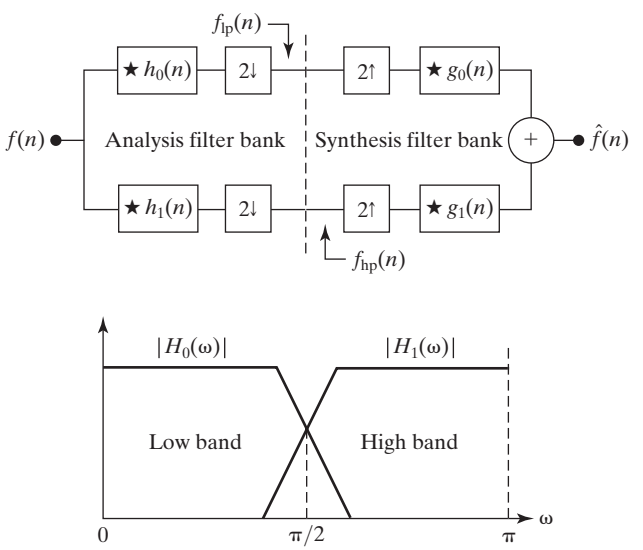
is called a *modulated* version of $h_1(n)$. Because modulation changes the signs of all odd-indexed coefficients [i.e., the coefficients for which n is odd in Fig. 7.5(e)], $h_5(1) = -h_1(1)$ and $h_5(3) = -h_1(3)$, while $h_5(0) = h_1(0)$ and $h_5(2) = h_1(2)$. Finally, the sequence shown in Fig. 7.5(f) is an order-reversed version of $h_1(n)$ that is also modulated:

$$h_6(n) = (-1)^n h_1(K - 1 - n) \tag{7.1-9}$$

This sequence is included to illustrate the fact that sign reversal, order reversal, and modulation are sometimes combined in the specification of the relationship between two filters.

With this brief introduction to digital signal filtering, consider the two-band subband coding and decoding system in Fig. 7.6(a). As indicated in the figure, the system is composed of two *filter banks*, each containing two FIR filters of the type shown in Fig. 7.4(a). Note that each of the four FIR filters is depicted

A *filter bank* is a collection of two or more filters.



a
b

FIGURE 7.6
(a) A two-band subband coding and decoding system, and (b) its spectrum splitting properties.

as a single block in Fig. 7.6(a), with the impulse response of each filter (and the convolution symbol) written inside it. The *analysis* filter bank, which includes filters $h_0(n)$ and $h_1(n)$, is used to break input sequence $f(n)$ into two half-length sequences $f_{lp}(n)$ and $f_{hp}(n)$, the *subbands* that represent the input. Note that filters $h_0(n)$ and $h_1(n)$ are half-band filters whose idealized transfer characteristics, H_0 and H_1, are shown in Fig. 7.6(b). Filter $h_0(n)$ is a lowpass filter whose output, subband $f_{lp}(n)$, is called an *approximation* of $f(n)$; filter $h_1(n)$ is a highpass filter whose output, subband $f_{hp}(n)$, is called the high frequency or *detail* part of $f(n)$. *Synthesis* bank filters $g_0(n)$ and $g_1(n)$ combine $f_{lp}(n)$ and $f_{hp}(n)$ to produce $\hat{f}(n)$. The goal in subband coding is to select $h_0(n)$, $h_1(n)$, $g_0(n)$, and $g_1(n)$ so that $\hat{f}(n) = f(n)$. That is, so that the input and output of the subband coding and decoding system are identical. When this is accomplished, the resulting system is said to employ *perfect reconstruction filters*.

By *real-coefficient*, we mean that the filter coefficients are real (not complex) numbers.

There are many two-band, real-coefficient, FIR, perfect reconstruction filter banks described in the filter bank literature. In all of them, the synthesis filters are modulated versions of the analysis filters—with one (and only one) synthesis filter being sign reversed as well. For perfect reconstruction, the impulse responses of the synthesis and analysis filters must be related in one of the following two ways:

Equations (7.1-10) through (7.1-14) are described in detail in the filter bank literature (see, for example, Vetterli and Kovacevic [1995]).

$$\begin{aligned} g_0(n) &= (-1)^n h_1(n) \\ g_1(n) &= (-1)^{n+1} h_0(n) \end{aligned} \tag{7.1-10}$$

or

$$\begin{aligned} g_0(n) &= (-1)^{n+1} h_1(n) \\ g_1(n) &= (-1)^n h_0(n) \end{aligned} \tag{7.1-11}$$

Filters $h_0(n)$, $h_1(n)$, $g_0(n)$, and $g_1(n)$ in Eqs. (7.1-10) and (7.1-11) are said to be *cross-modulated* because diagonally opposed filters in the block diagram of Fig. 7.6(a) are related by modulation [and sign reversal when the modulation factor is $-(-1)^n$ or $(-1)^{n+1}$]. Moreover, they can be shown to satisfy the following *biorthogonality* condition:

$$\langle h_i(2n - k), g_j(k) \rangle = \delta(i - j)\delta(n), \quad i, j = \{0, 1\} \tag{7.1-12}$$

Here, $\langle h_i(2n - k), g_j(k) \rangle$ denotes the inner product of $h_i(2n - k)$ and $g_j(k)$.[†] When i is not equal to j, the inner product is 0; when i and j are equal, the product is the unit discrete impulse function, $\delta(n)$. Biorthogonality will be considered again in Section 7.2.1.

Of special interest in subband coding—and in the development of the fast wavelet transform of Section 7.4—are filters that move beyond biorthogonality and require

[†]The vector inner product of sequences $f_1(n)$ and $f_2(n)$ is $\langle f_1, f_2 \rangle = \sum_n f_1^*(n) f_2(n)$, where the * denotes the complex conjugate operation. If $f_1(n)$ and $f_2(n)$ are real, $\langle f_1, f_2 \rangle = \langle f_2, f_1 \rangle$.

$$\langle g_i(n), g_j(n + 2m)\rangle = \delta(i - j)\delta(m), \quad i, j = \{0, 1\} \tag{7.1-13}$$

which defines *orthonormality* for perfect reconstruction filter banks. In addition to Eq. (7.1-13), orthonormal filters can be shown to satisfy the following two conditions:

$$\begin{aligned} g_1(n) &= (-1)^n g_0(K_{\text{even}} - 1 - n) \\ h_i(n) &= g_i(K_{\text{even}} - 1 - n), \quad i = \{0, 1\} \end{aligned} \tag{7.1-14}$$

where the subscript on K_{even} is used to indicate that the number of filter coefficients must be divisible by 2 (i.e., an even number). As Eq. (7.1-14) indicates, synthesis filter g_1 is related to g_0 by order reversal and modulation. In addition, both h_0 and h_1 are order-reversed versions of synthesis filters, g_0 and g_1, respectively. Thus, an orthonormal filter bank can be developed around the impulse response of a single filter, called the *prototype*; the remaining filters can be computed from the specified prototype's impulse response. For biorthogonal filter banks, two prototypes are required; the remaining filters can be computed via Eq. (7.1-10) or (7.1-11). The generation of useful prototype filters, whether orthonormal or biorthogonal, is beyond the scope of this chapter. We simply use filters that have been presented in the literature and provide references for further study.

Before concluding the section with a 2-D subband coding example, we note that 1-D orthonormal and biorthogonal filters can be used as 2-D separable filters for the processing of images. As can be seen in Fig. 7.7, the separable filters are first applied in one dimension (e.g., vertically) and then in the other (e.g., horizontally) in the manner introduced in Section 2.6.7. Moreover, downsampling is performed in two stages—once before the second filtering operation to reduce the overall number of computations. The resulting filtered

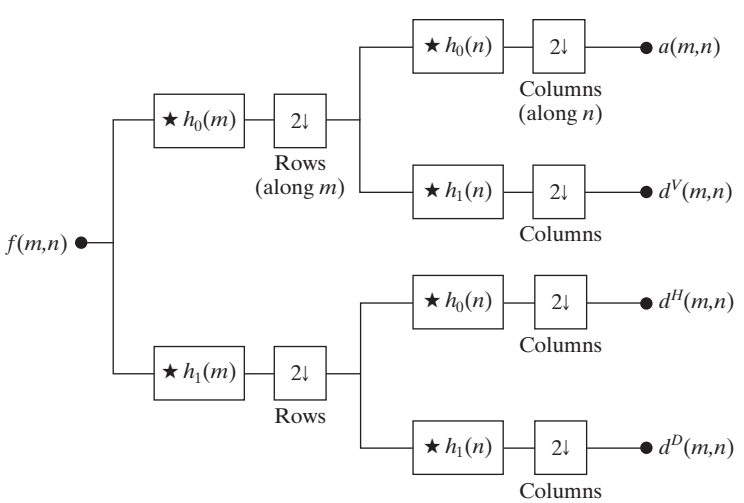


FIGURE 7.7
A two-dimensional, four-band filter bank for subband image coding.

outputs, denoted $a(m, n)$, $d^V(m, n)$, $d^H(m, n)$, and $d^D(m, n)$ in Fig. 7.7, are called the *approximation*, *vertical detail*, *horizontal detail*, and *diagonal detail* subbands of the input image, respectively. These subbands can be split into four smaller subbands, which can be split again, and so on—a property that will be described in greater detail in Section 7.4.

EXAMPLE 7.2: A four-band subband coding of the vase in Fig. 7.1.

■ Figure 7.8 shows the impulse responses of four 8-tap orthonormal filters. The coefficients of prototype synthesis filter $g_0(n)$ for $0 \leq n \leq 7$ [in Fig. 7.8(c)] are defined in Table 7.1 (Daubechies [1992]). The coefficients of the remaining orthonormal filters can be computed using Eq. (7.1-14). With the help of Fig. 7.5, note (by visual inspection) the cross modulation of the analysis and synthesis filters in Fig. 7.8. It is relatively easy to show numerically that the filters are

TABLE 7.1 Daubechies 8-tap orthonormal filter coefficients for $g_0(n)$ (Daubechies [1992]).

n	$g_0(n)$
0	0.23037781
1	0.71484657
2	0.63088076
3	−0.02798376
4	−0.18703481
5	0.03084138
6	0.03288301
7	−0.01059740

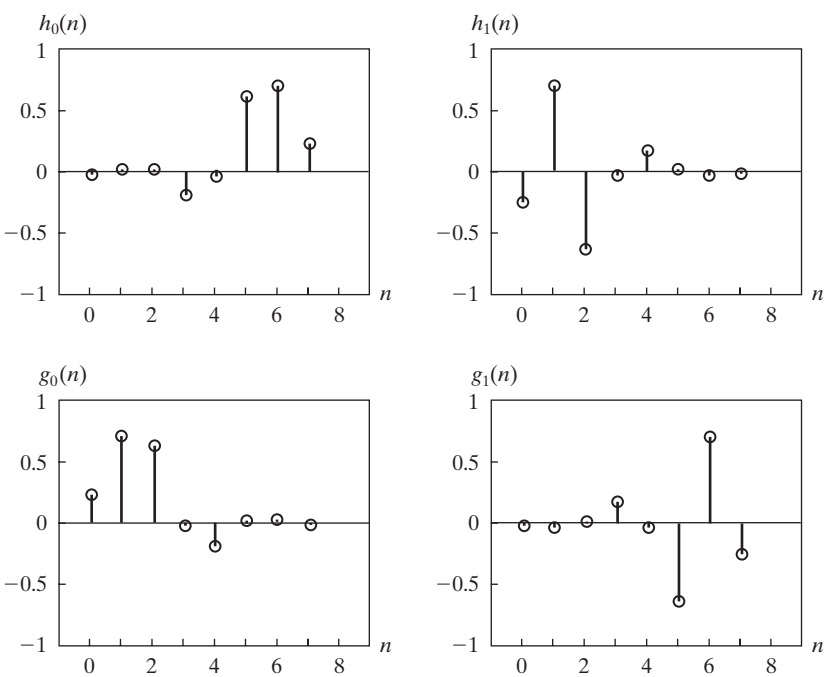
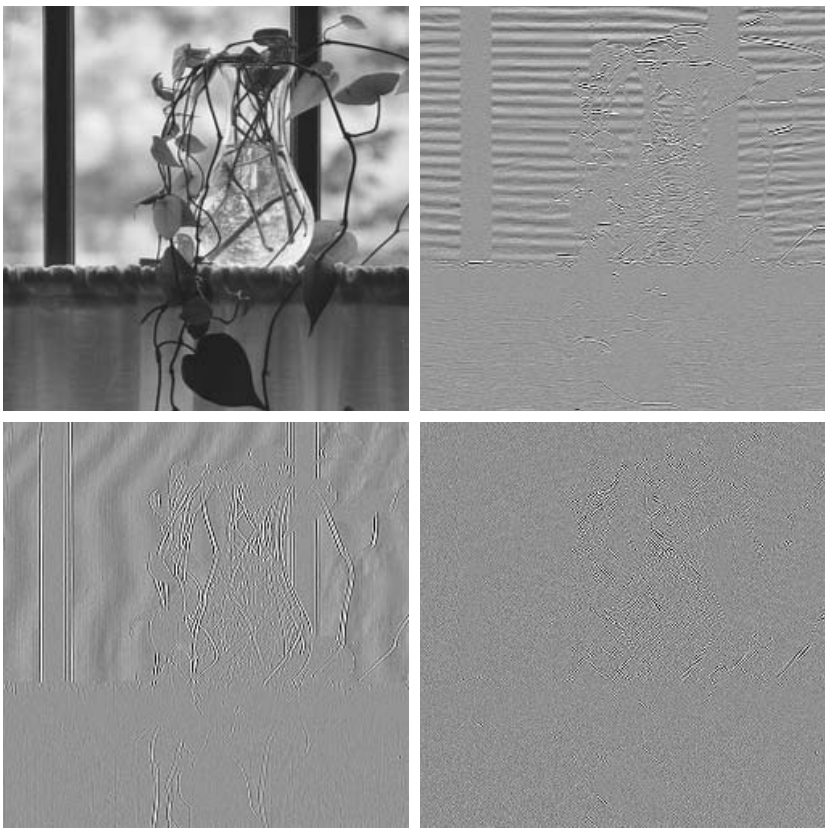


FIGURE 7.8 The impulse responses of four 8-tap Daubechies orthonormal filters. See Table 7.1 for the values of $g_0(n)$ for $0 \leq n \leq 7$.



a b
c d

FIGURE 7.9
A four-band split of the vase in Fig. 7.1 using the subband coding system of Fig. 7.7. The four subbands that result are the (a) approximation, (b) horizontal detail, (c) vertical detail, and (d) diagonal detail subbands.

both biorthogonal (they satisfy Eq. 7.1-12) and orthonormal (they satisfy Eq. 7.1-13). As a result, the Daubechies 8-tap filters in Fig. 7.8 support error-free reconstruction of the decomposed input.

A four-band split of the 512×512 image of a vase in Fig. 7.1, based on the filters in Fig. 7.8, is shown in Fig. 7.9. Each quadrant of this image is a subband of size 256×256. Beginning with the upper-left corner and proceeding in a clockwise manner, the four quadrants contain approximation subband a, horizontal detail subband d^H, diagonal detail subband d^D, and vertical detail subband d^V, respectively. All subbands, except the approximation subband in Fig. 7.9(a), have been scaled to make their underlying structure more visible. Note the visual effects of aliasing that are present in Figs. 7.9(b) and (c)—the d^H and d^V subbands. The wavy lines in the window area are due to the downsampling of a barely discernable window screen in Fig. 7.1. Despite the aliasing, the original image can be reconstructed from the subbands in Fig. 7.9 without error. The required synthesis filters, $g_0(n)$ and $g_1(n)$, are determined from Table 7.1 and Eq. (7.1-14), and incorporated into a filter bank that roughly mirrors the system in Fig. 7.7. In the new filter bank, filters $h_i(n)$ for $i = \{0, 1\}$ are replaced by their $g_i(n)$ counterparts, and upsamplers and summers are added. ■

See Section 4.5.4 for more on aliasing.

7.1.3 The Haar Transform

The third and final imaging-related operation with ties to multiresolution analysis that we will look at is the Haar transform (Haar [1910]). Within the context of this chapter, its importance stems from the fact that its basis functions (defined below) are the oldest and simplest known orthonormal wavelets. They will be used in a number of examples in the sections that follow.

With reference to the discussion in Section 2.6.7, the Haar transform can be expressed in the following matrix form

$$\mathbf{T} = \mathbf{HFH}^T \tag{7.1-15}$$

where $\mathbf{F}$ is an $N \times N$ image matrix, $\mathbf{H}$ is an $N \times N$ Haar transformation matrix, and $\mathbf{T}$ is the resulting $N \times N$ transform. The transpose is required because $\mathbf{H}$ is not symmetric; in Eq. (2.6-38) of Section 2.6.7, the transformation matrix is assumed to be symmetric. For the Haar transform, $\mathbf{H}$ contains the Haar basis functions, $h_k(z)$. They are defined over the continuous, closed interval $z \in [0, 1]$ for $k = 0, 1, 2, \ldots, N - 1$, where $N = 2^n$. To generate $\mathbf{H}$, we define the integer k such that $k = 2^p + q - 1$, where $0 \le p \le n - 1$, $q = 0$ or 1 for $p = 0$, and $1 \le q \le 2^p$ for $p \ne 0$. Then the *Haar basis functions* are

$$h_0(z) = h_{00}(z) = \frac{1}{\sqrt{N}}, \quad z \in [0, 1] \tag{7.1-16}$$

and

$$h_k(z) = h_{pq}(z) = \frac{1}{\sqrt{N}} \begin{cases} 2^{p/2} & (q - 1)/2^p \le z < (q - 0.5)/2^p \\ -2^{p/2} & (q - 0.5)/2^p \le z < q/2^p \\ 0 & \text{otherwise, } z \in [0, 1] \end{cases} \tag{7.1-17}$$

The ith row of an $N \times N$ Haar transformation matrix contains the elements of $h_i(z)$ for $z = 0/N, 1/N, 2/N, \ldots, (N - 1)/N$. For instance, if $N = 2$, the first row of the 2×2 Haar matrix is computed using $h_0(z)$ with $z = 0/2, 1/2$. From Eq. (7.1-16), $h_0(z)$ is equal to $1/\sqrt{2}$, independent of z, so the first row of $\mathbf{H}_2$ has two identical $1/\sqrt{2}$ elements. The second row is obtained by computing $h_1(z)$ for $z = 0/2, 1/2$. Because $k = 2^p + q - 1$, when $k = 1$, $p = 0$ and $q = 1$. Thus, from Eq. (7.1-17), $h_1(0) = 2^0/\sqrt{2} = 1/\sqrt{2}$, $h_1(1/2) = -2^0/\sqrt{2} = -1/\sqrt{2}$, and the 2×2 Haar matrix is

$$\mathbf{H}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{7.1-18}$$

If $N = 4$, k, q, and p assume the values

k	p	q
0	0	0
1	0	1
2	1	1
3	1	2

and the 4×4 transformation matrix, $\mathbf{H}_4$, is

$$\mathbf{H}_4 = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix} \tag{7.1-19}$$

Our principal interest in the Haar transform is that the rows of $\mathbf{H}_2$ can be used to define the analysis filters, $h_0(n)$ and $h_1(n)$, of a 2-tap perfect reconstruction filter bank (see the previous section), as well as the scaling and wavelet vectors (defined in Sections 7.2.2 and 7.2.3, respectively) of the simplest and oldest wavelet transform (see Example 7.10 in Section 7.4). Rather than concluding the section with the computation of a Haar transform, we close with an example that illustrates the influence of the decomposition methods that have been considered to this point on the methods that will be developed in the remainder of the chapter.

EXAMPLE 7.3: Haar functions in a discrete wavelet transform.

■ Figure 7.10(a) shows a decomposition of the 512×512 image in Fig. 7.1 that combines the key features of pyramid coding, subband coding, and the Haar transform (the three techniques we have discussed so far). Called the discrete wavelet transform (and developed later in the chapter), the representation is characterized by the following important features:

1. With the exception of the subimage in the upper-left corner of Fig. 7.10(a), the local histograms are very similar. Many of the pixels are close to zero. Because the subimages (except for the subimage in the upper-left corner) have been scaled to make their underlying structure more visible, the displayed histograms are peaked at intensity 128 (the zeroes have been scaled to mid-gray). The large number of zeroes in the decomposition makes the image an excellent candidate for compression (see Chapter 8).
2. In a manner that is similar to the way in which the levels of the prediction residual pyramid of Fig. 7.3(b) were used to create approximation images of differing resolutions, the subimages in Fig. 7.10(a) can be used to construct both coarse and fine resolution approximations of the original vase image in Fig. 7.1. Figures 7.10(b) through (d), which are of size

a
b c d

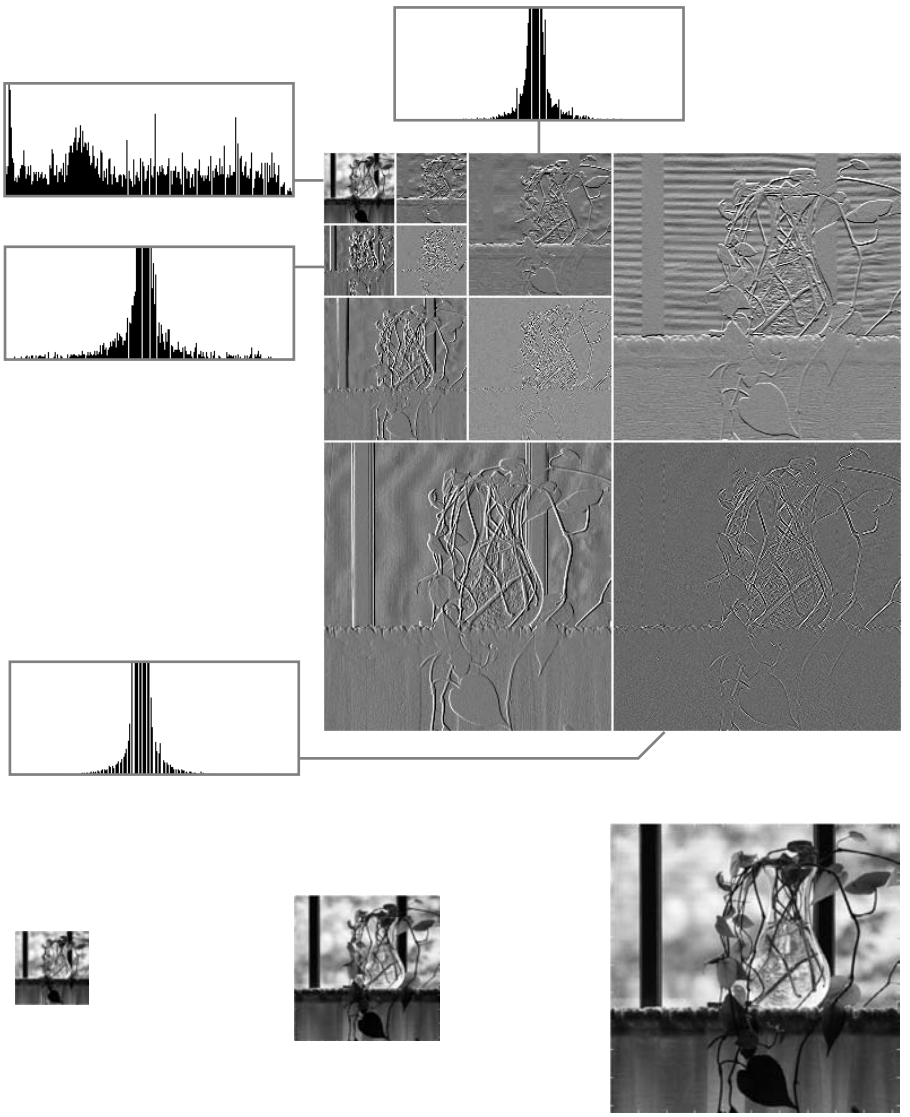


FIGURE 7.10
(a) A discrete wavelet transform using Haar $\mathbf{H}_2$ basis functions. Its local histogram variations are also shown. (b)–(d) Several different approximations (64×64, 128×128, and 256×256) that can be obtained from (a).

64×64, 128×128, and 256×256, respectively, were generated from the subimages in Fig. 7.10(a). A perfect 512×512 reconstruction of the original image is also possible.

3. Like the subband coding decomposition in Fig. 7.9, a simple real-coefficient, FIR filter bank of the form given in Fig. 7.7 was used to produce Fig. 7.10(a). After the generation of a four subband image like that of Fig. 7.9, the 256×256 approximation subband was decomposed and replaced by four 128×128 subbands (using the same filter bank), and the resulting approximation subband was again decomposed and replaced by four 64×64 subbands. This process produced the unique arrangement of subimages that

characterizes discrete wavelet transforms. The subimages in Fig. 7.10(a) become smaller in size as you move from the lower-right-hand to upper-left-hand corner of the image.

4. Figure 7.10(a) is not the Haar transform of the image in Fig. 7.1. Although the filter bank coefficients that were used to produce this decomposition were taken from Haar transformation matrix $\mathbf{H}_2$, a variety of othronormal and biorthogonal filter bank coefficients can be used in discrete wavelet transforms.
5. As will be shown in Section 7.4, each subimage in Fig. 7.10(a) represents a specific band of spatial frequencies in the original image. In addition, many of the subimages demonstrate directional sensitivity [e.g., the subimage in the upper-right corner of Fig. 7.10(a) captures horizontal edge information in the original image].

Considering this impressive list of features, it is remarkable that the discrete wavelet transform of Fig. 7.10(a) was generated using two 2-tap digital filters with a total of four filter coefficients. ■

7.2 Multiresolution Expansions

The previous section introduced three well-known imaging techniques that play an important role in a mathematical framework called *multiresolution analysis* (MRA). In MRA, a *scaling function* is used to create a series of approximations of a function or image, each differing by a factor of 2 in resolution from its nearest neighboring approximations. Additional functions, called *wavelets*, are then used to encode the difference in information between adjacent approximations.

7.2.1 Series Expansions

A signal or function $f(x)$ can often be better analyzed as a linear combination of expansion functions

$$f(x) = \sum_k \alpha_k \varphi_k(x) \tag{7.2-1}$$

where k is an integer index of a finite or infinite sum, the α_k are real-valued *expansion coefficients*, and the $\varphi_k(x)$ are real-valued *expansion functions*. If the expansion is unique—that is, there is only one set of α_k for any given $f(x)$—the $\varphi_k(x)$ are called *basis functions*, and the *expansion set*, $\{\varphi_k(x)\}$, is called a *basis* for the class of functions that can be so expressed. The expressible functions form a *function space* that is referred to as the *closed span* of the expansion set, denoted

$$V = \overline{\underset{k}{\text{Span}}\{\varphi_k(x)\}} \tag{7.2-2}$$

To say that $f(x) \in V$ means that $f(x)$ is in the closed span of $\{\varphi_k(x)\}$ and can be written in the form of Eq. (7.2-1).

For any function space V and corresponding expansion set $\{\varphi_k(x)\}$, there is a set of *dual* functions denoted $\{\tilde{\varphi}_k(x)\}$ that can be used to compute the α_k coefficients of Eq. (7.2-1) for any $f(x) \in V$. These coefficients are computed by taking the *integral inner products*[†] of the dual $\tilde{\varphi}_k(x)$ and function $f(x)$. That is,

$$\alpha_k = \langle \tilde{\varphi}_k(x), f(x) \rangle = \int \tilde{\varphi}_k^*(x) f(x)\, dx \tag{7.2-3}$$

where the * denotes the complex conjugate operation. Depending on the orthogonality of the expansion set, this computation assumes one of three possible forms. Problem 7.10 at the end of the chapter illustrates the three cases using vectors in two-dimensional Euclidean space.

Case 1: If the expansion functions form an orthonormal basis for V, meaning that

$$\langle \varphi_j(x), \varphi_k(x) \rangle = \delta_{jk} = \begin{cases} 0 & j \neq k \\ 1 & j = k \end{cases} \tag{7.2-4}$$

the basis and its dual are equivalent. That is, $\varphi_k(x) = \tilde{\varphi}_k(x)$ and Eq. (7.2-3) becomes

$$\alpha_k = \langle \varphi_k(x), f(x) \rangle \tag{7.2-5}$$

The α_k are computed as the inner products of the basis functions and $f(x)$.

Case 2: If the expansion functions are not orthonormal, but are an orthogonal basis for V, then

$$\langle \varphi_j(x), \varphi_k(x) \rangle = 0 \quad j \neq k \tag{7.2-6}$$

and the basis functions and their duals are called *biorthogonal*. The α_k are computed using Eq. (7.2-3), and the biorthogonal basis and its dual are such that

$$\langle \varphi_j(x), \tilde{\varphi}_k(x) \rangle = \delta_{jk} = \begin{cases} 0 & j \neq k \\ 1 & j = k \end{cases} \tag{7.2-7}$$

Case 3: If the expansion set is not a basis for V, but supports the expansion defined in Eq. (7.2-1), it is a spanning set in which there is more than one set of α_k for any $f(x) \in V$. The expansion functions and their duals are said to be *overcomplete* or redundant. They form a *frame* in which[‡]

$$A\|f(x)\|^2 \leq \sum_k |\langle \varphi_k(x), f(x) \rangle|^2 \leq B\|f(x)\|^2 \tag{7.2-8}$$

[†]The integral inner product of two real or complex-valued functions $f(x)$ and $g(x)$ is $\langle f(x), g(x) \rangle = \int f^*(x) g(x)\, dx$. If $f(x)$ is real, $f^*(x) = f(x)$ and $\langle f(x), g(x) \rangle = \int f(x) g(x)\, dx$.

[‡]The norm of $f(x)$, denoted $\|f(x)\|$, is defined as the square root of the absolute value of the inner product of $f(x)$ with itself.

for some $A > 0$, $B < \infty$, and all $f(x) \in V$. Dividing this equation by the norm squared of $f(x)$, we see that A and B "frame" the normalized inner products of the expansion coefficients and the function. Equations similar to (7.2-3) and (7.2-5) can be used to find the expansion coefficients for frames. If $A = B$, the expansion set is called a *tight frame* and it can be shown that (Daubechies [1992])

$$f(x) = \frac{1}{A} \sum_k \langle \varphi_k(x), f(x) \rangle \varphi_k(x) \tag{7.2-9}$$

Except for the A^{-1} term, which is a measure of the frame's redundancy, this is identical to the expression obtained by substituting Eq. (7.2-5) (for orthonormal bases) into Eqs. (7.2-1).

7.2.2 Scaling Functions

Consider the set of expansion functions composed of integer translations and binary scalings of the real, square-integrable function $\varphi(x)$; this is the set $\{\varphi_{j,k}(x)\}$, where

$$\varphi_{j,k}(x) = 2^{j/2} \varphi(2^j x - k) \tag{7.2-10}$$

for all $j, k \in \mathbf{Z}$ and $\varphi(x) \in L^2(\mathbf{R})$.[†] Here, k determines the position of $\varphi_{j,k}(x)$ along the x-axis, and j determines the width of $\varphi_{j,k}(x)$—that is, how broad or narrow it is along the x-axis. The term $2^{j/2}$ controls the amplitude of the function. Because the shape of $\varphi_{j,k}(x)$ changes with j, $\varphi(x)$ is called a *scaling function*. By choosing $\varphi(x)$ properly, $\{\varphi_{j,k}(x)\}$ can be made to span $L^2(\mathbf{R})$, which is the set of all measurable, square-integrable functions.

If we restrict j in Eq. (7.2-10) to a specific value, say $j = j_0$, the resulting expansion set, $\{\varphi_{j_0,k}(x)\}$, is a subset of $\{\varphi_{j,k}(x)\}$ that spans a subspace of $L^2(\mathbf{R})$. Using the notation of the previous section, we can define that subspace as

$$V_{j_0} = \overline{\underset{k}{\text{Span}}\{\varphi_{j_0,k}(x)\}} \tag{7.2-11}$$

That is, V_{j_0} is the span of $\varphi_{j_0,k}(x)$ over k. If $f(x) \in V_{j_0}$, we can write

$$f(x) = \sum_k \alpha_k \varphi_{j_0,k}(x) \tag{7.2-12}$$

More generally, we will denote the subspace spanned over k for any j as

$$V_j = \overline{\underset{k}{\text{Span}}\{\varphi_{j,k}(x)\}} \tag{7.2-13}$$

As will be seen in the following example, increasing j increases the size of V_j, allowing functions with smaller variations or finer detail to be included in the subspace. This is a consequence of the fact that, as j increases, the $\varphi_{j,k}(x)$ that are used to represent the subspace functions become narrower and separated by smaller changes in x.

[†]The notation $L^2(\mathbf{R})$, where $\mathbf{R}$ is the set of real numbers, denotes the set of measurable, square-integrable, one-dimensional functions; $\mathbf{Z}$ is the set of integers.

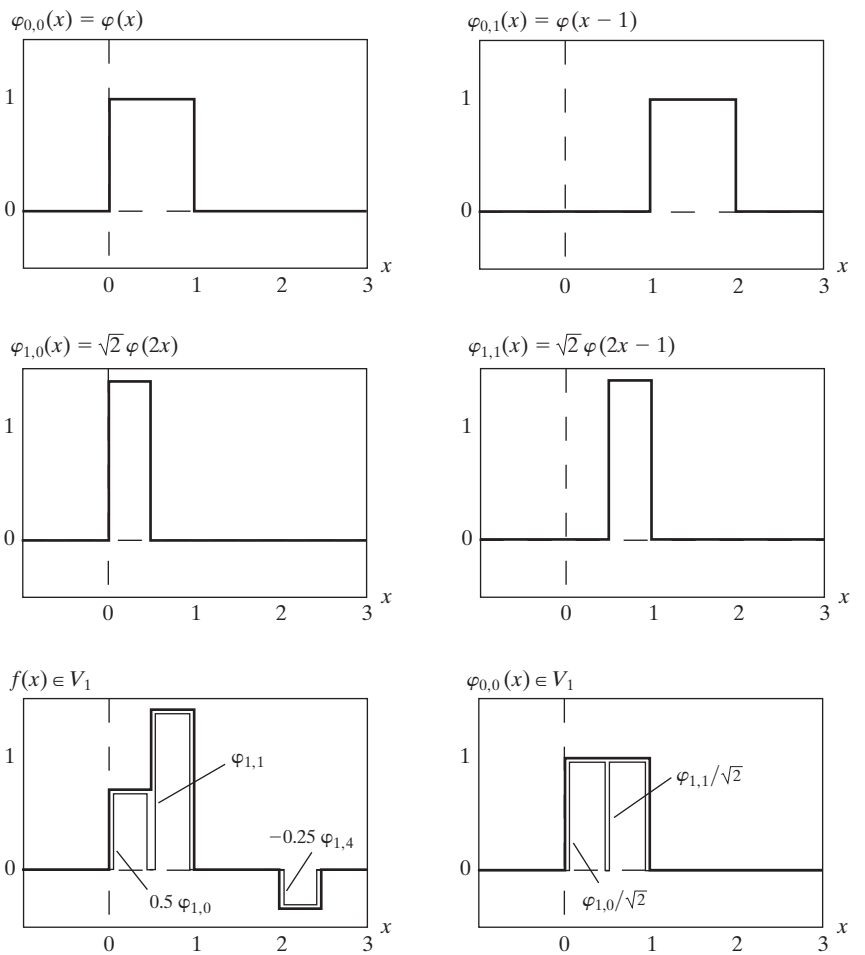
EXAMPLE 7.4: The Haar scaling function.

■ Consider the unit-height, unit-width scaling function (Haar [1910])

$$\varphi(x) = \begin{cases} 1 & 0 \le x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{7.2-14}$$

Figures 7.11(a) through (d) show four of the many expansion functions that can be generated by substituting this pulse-shaped scaling function into Eq. (7.2-10). Note that the expansion functions for $j = 1$ in Figs. 7.11(c) and (d) are half as wide as those for $j = 0$ in Figs. 7.11(a) and (b). For a given interval on x, we can define twice as many V_1 scaling functions as V_0 scaling functions (e.g., $\varphi_{1,0}$ and $\varphi_{1,1}$ of V_1 versus $\varphi_{0,0}$ of V_0 for the interval $0 \le x < 1$).

Figure 7.11(e) shows a member of subspace V_1. This function does not belong to V_0, because the V_0 expansion functions in 7.11(a) and (b) are too coarse to represent it. Higher-resolution functions like those in 7.11(c) and (d)



a b
c d
e f

FIGURE 7.11 Some Haar scaling functions.

are required. They can be used, as shown in (e), to represent the function by the three-term expansion

$$f(x) = 0.5\varphi_{1,0}(x) + \varphi_{1,1}(x) - 0.25\varphi_{1,4}(x)$$

To conclude the example, Fig. 7.11(f) illustrates the decomposition of $\varphi_{0,0}(x)$ as a sum of V_1 expansion functions. In a similar manner, any V_0 expansion function can be decomposed using

$$\varphi_{0,k}(x) = \frac{1}{\sqrt{2}}\varphi_{1,2k}(x) + \frac{1}{\sqrt{2}}\varphi_{1,2k+1}(x)$$

Thus, if $f(x)$ is an element of V_0, it is also an element of V_1. This is because all V_0 expansion functions are contained in V_1. Mathematically, we write that V_0 is a subspace of V_1, denoted $V_0 \subset V_1$. ■

The simple scaling function in the preceding example obeys the four fundamental requirements of multiresolution analysis (Mallat [1989a]):

MRA Requirement 1: The scaling function is orthogonal to its integer translates.

This is easy to see in the case of the Haar function, because whenever it has a value of 1, its integer translates are 0, so that the product of the two is 0. The Haar scaling function is said to have *compact support*, which means that it is 0 everywhere outside a finite interval called the *support*. In fact, the width of the support is 1; it is 0 outside the half open interval [0, 1). It should be noted that the requirement for orthogonal integer translates becomes harder to satisfy as the width of support of the scaling function becomes larger than 1.

MRA Requirement 2: The subspaces spanned by the scaling function at low scales are nested within those spanned at higher scales.

As can be seen in Fig. 7.12, subspaces containing high-resolution functions must also contain all lower resolution functions. That is,

$$V_{-\infty} \subset \cdots \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots \subset V_{\infty} \qquad (7.2\text{-}15)$$

Moreover, the subspaces satisfy the intuitive condition that if $f(x) \in V_j$, then $f(2x) \in V_{j+1}$. The fact that the Haar scaling function meets this requirement

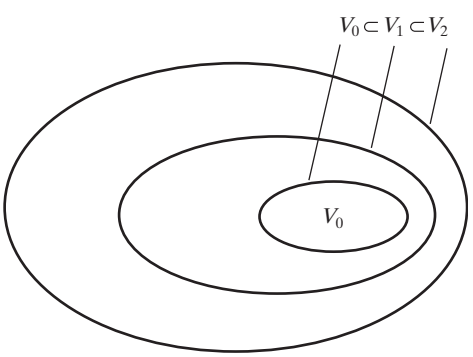


FIGURE 7.12 The nested function spaces spanned by a scaling function.

should not be taken to indicate that any function with a support width of 1 automatically satisfies the condition. It is left as an exercise for the reader to show that the equally simple function

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

is not a valid scaling function for a multiresolution analysis (see Problem 7.11).

MRA Requirement 3: The only function that is common to all V_j is $f(x) = 0$. If we consider the coarsest possible expansion functions (i.e., $j = -\infty$), the only representable function is the function of no information. That is,

$$V_{-\infty} = \{0\} \tag{7.2-16}$$

MRA Requirement 4: Any function can be represented with arbitrary precision. Though it may not be possible to expand a particular $f(x)$ at an arbitrarily coarse resolution, as was the case for the function in Fig. 7.11(e), all measurable, square-integrable functions can be represented by the scaling functions in the limit as $j \to \infty$. That is,

$$V_{\infty} = \left\{L^2(\mathbf{R})\right\} \tag{7.2-17}$$

Under these conditions, the expansion functions of subspace V_j can be expressed as a weighted sum of the expansion functions of subspace V_{j+1}. Using Eq. (7.2-12), we let

$$\varphi_{j,k}(x) = \sum_n \alpha_n \varphi_{j+1,n}(x)$$

where the index of summation has been changed to n for clarity. Substituting for $\varphi_{j+1,n}(x)$ from Eq. (7.2-10) and changing variable α_n to $h_\varphi(n)$, this becomes

The α_n are changed to $h_\varphi(n)$ because they are used later (see Section 7.4) as filter bank coefficients.

$$\varphi_{j,k}(x) = \sum_n h_\varphi(n) 2^{(j+1)/2} \varphi(2^{j+1}x - n)$$

Because $\varphi(x) = \varphi_{0,0}(x)$, both j and k can be set to 0 to obtain the simpler nonsubscripted expression

$$\varphi(x) = \sum_n h_\varphi(n) \sqrt{2} \varphi(2x - n) \tag{7.2-18}$$

The $h_\varphi(n)$ coefficients in this recursive equation are called *scaling function coefficients*; h_φ is referred to as a *scaling vector*. Equation (7.2-18) is fundamental to multiresolution analysis and is called the *refinement equation*, the *MRA equation*, or the *dilation equation*. It states that the expansion functions of any subspace can be built from double-resolution copies of themselves—that is, from expansion functions of the next higher resolution space. The choice of a reference subspace, V_0, is arbitrary.

EXAMPLE 7.5: Haar scaling function coefficients.

■ The scaling function coefficients for the Haar function of Eq. (7.2-14) are $h_\varphi(0) = h_\varphi(1) = 1/\sqrt{2}$, the first row of matrix $\mathbf{H}_2$ in Eq. (7.1-18). Thus, Eq. (7.2-18) yields

$$\varphi(x) = \frac{1}{\sqrt{2}}\left[\sqrt{2}\varphi(2x)\right] + \frac{1}{\sqrt{2}}\left[\sqrt{2}\varphi(2x-1)\right]$$

This decomposition was illustrated graphically for $\varphi_{0,0}(x)$ in Fig. 7.11(f), where the bracketed terms of the preceding expression are seen to be $\varphi_{1,0}(x)$ and $\varphi_{1,1}(x)$. Additional simplification yields $\varphi(x) = \varphi(2x) + \varphi(2x-1)$. ■

7.2.3 Wavelet Functions

Given a scaling function that meets the MRA requirements of the previous section, we can define a *wavelet function* $\psi(x)$ that, together with its integer translates and binary scalings, spans the difference between any two adjacent scaling subspaces, V_j and V_{j+1}. The situation is illustrated graphically in Fig. 7.13. We define the set $\{\psi_{j,k}(x)\}$ of wavelets

$$\psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k) \tag{7.2-19}$$

for all $k \in \mathbf{Z}$ that span the W_j spaces in the figure. As with scaling functions, we write

$$W_j = \overline{\operatorname{Span}_k\{\psi_{j,k}(x)\}} \tag{7.2-20}$$

and note that if $f(x) \in W_j$,

$$f(x) = \sum_k \alpha_k \psi_{j,k}(x) \tag{7.2-21}$$

The scaling and wavelet function subspaces in Fig. 7.13 are related by

$$V_{j+1} = V_j \oplus W_j \tag{7.2-22}$$

where $\oplus$ denotes the union of spaces (like the union of sets). The orthogonal complement of V_j in V_{j+1} is W_j, and all members of V_j are orthogonal to the members of W_j. Thus,

$$\langle \varphi_{j,k}(x), \psi_{j,l}(x) \rangle = 0 \tag{7.2-23}$$

for all appropriate $j, k, l \in \mathbf{Z}$.

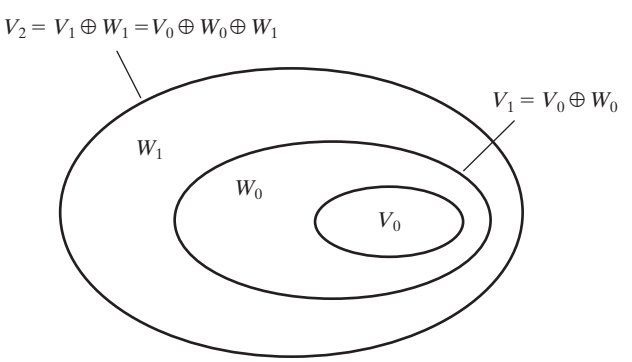


FIGURE 7.13 The relationship between scaling and wavelet function spaces.

We can now express the space of all measurable, square-integrable functions as

$$L^2(\mathbf{R}) = V_0 \oplus W_0 \oplus W_1 \oplus \ldots \tag{7.2-24}$$

or

$$L^2(\mathbf{R}) = V_1 \oplus W_1 \oplus W_2 \oplus \ldots \tag{7.2-25}$$

or even

$$L^2(\mathbf{R}) = \cdots \oplus W_{-2} \oplus W_{-1} \oplus W_0 \oplus W_1 \oplus W_2 \oplus \ldots \tag{7.2-26}$$

which eliminates the scaling function, and represents a function in terms of wavelets alone [i.e., there are only wavelet function spaces in Eq. (7.2-26)]. Note that if $f(x)$ is an element of V_1, but not V_0, an expansion using Eq. (7.2-24) contains an *approximation* of $f(x)$ using V_0 scaling functions. Wavelets from W_0 would encode the *difference* between this approximation and the actual function. Equations (7.2-24) through (7.2-26) can be generalized to yield

$$L^2(\mathbf{R}) = V_{j_0} \oplus W_{j_0} \oplus W_{j_0+1} \oplus \ldots \tag{7.2-27}$$

where j_0 is an arbitrary starting scale.

Since wavelet spaces reside within the spaces spanned by the next higher resolution scaling functions (see Fig. 7.13), any wavelet function—like its scaling function counterpart of Eq. (7.2-18)—can be expressed as a weighted sum of shifted, double-resolution scaling functions. That is, we can write

$$\psi(x) = \sum_n h_\psi(n)\sqrt{2}\varphi(2x - n) \tag{7.2-28}$$

where the $h_\psi(n)$ are called the *wavelet function coefficients* and h_ψ is the *wavelet vector*. Using the condition that wavelets span the orthogonal complement spaces in Fig. 7.13 and that integer wavelet translates are orthogonal, it can be shown that $h_\psi(n)$ is related to $h_\varphi(n)$ by (see, for example, Burrus, Gopinath, and Guo [1998])

$$h_\psi(n) = (-1)^n h_\varphi(1 - n) \tag{7.2-29}$$

Note the similarity of this result and Eq. (7.1-14), the relationship governing the impulse responses of orthonormal subband coding and decoding filters.

EXAMPLE 7.6: The Haar wavelet function coefficients.

■ In the previous example, the Haar scaling vector was defined as $h_\varphi(0) = h_\varphi(1) = 1/\sqrt{2}$. Using Eq. (7.2-29), the corresponding wavelet vector is $h_\psi(0) = (-1)^0 h_\varphi(1 - 0) = 1/\sqrt{2}$ and $h_\psi(1) = (-1)^1 h_\varphi(1 - 1) = -1/\sqrt{2}$. Note that these coefficients correspond to the second row of matrix $\mathbf{H}_2$ in Eq. (7.1-18). Substituting these values into Eq. (7.2-28), we get

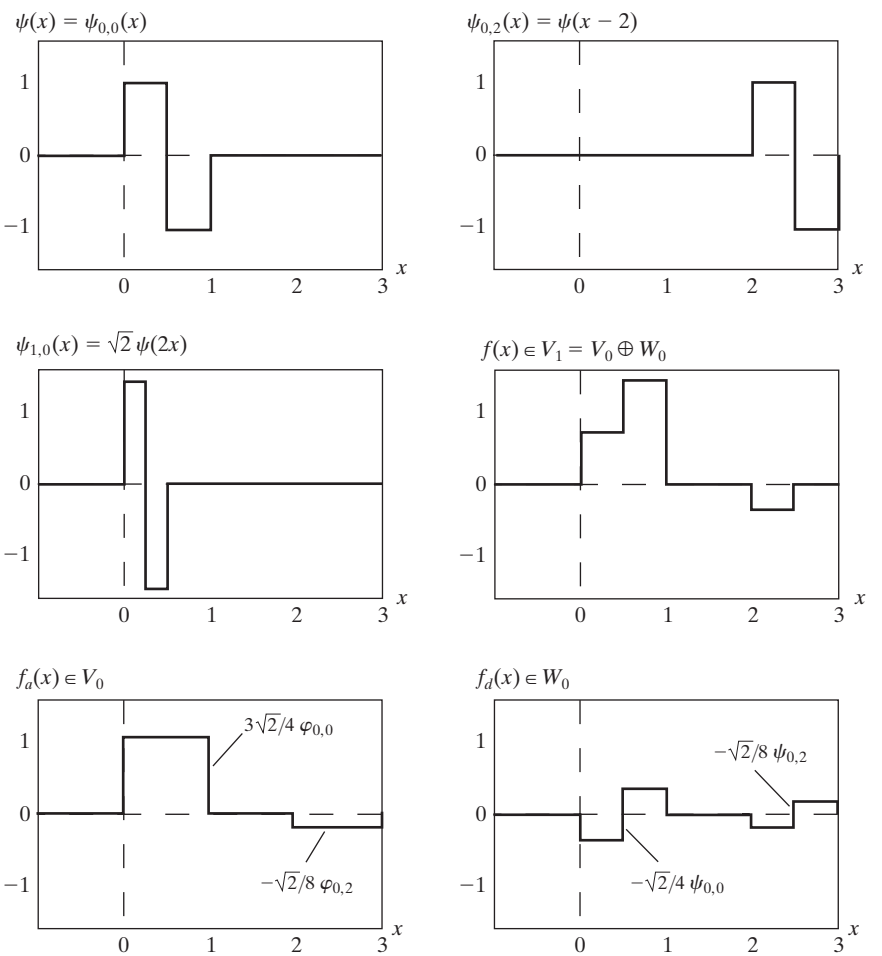
$\psi(x) = \varphi(2x) - \varphi(2x - 1)$, which is plotted in Fig. 7.14(a). Thus, the Haar wavelet function is

$$\psi(x) = \begin{cases} 1 & 0 \le x < 0.5 \\ -1 & 0.5 \le x < 1 \\ 0 & \text{elsewhere} \end{cases} \tag{7.2-30}$$

Using Eq. (7.2-19), we can now generate the universe of scaled and translated Haar wavelets. Two such wavelets, $\psi_{0,2}(x)$ and $\psi_{1,0}(x)$, are plotted in Figs. 7.14(b) and (c), respectively. Note that wavelet $\psi_{1,0}(x)$ for space W_1 is narrower than $\psi_{0,2}(x)$ for W_0; it can be used to represent finer detail.

Figure 7.14(d) shows a function of subspace V_1 that is not in subspace V_0. This function was considered in an earlier example [see Fig. 7.11(e)]. Although the function cannot be represented accurately in V_0, Eq. (7.2-22) indicates that it can be expanded using V_0 and W_0 expansion functions. The resulting expansion is

$$f(x) = f_a(x) + f_d(x)$$



a b
c d
e f

FIGURE 7.14 Haar wavelet functions in W_0 and W_1.

where

$$f_a(x) = \frac{3\sqrt{2}}{4}\varphi_{0,0}(x) - \frac{\sqrt{2}}{8}\varphi_{0,2}(x)$$

and

$$f_d(x) = \frac{-\sqrt{2}}{4}\psi_{0,0}(x) - \frac{\sqrt{2}}{8}\psi_{0,2}(x)$$

Here, $f_a(x)$ is an approximation of $f(x)$ using V_0 scaling functions, while $f_d(x)$ is the difference $f(x) - f_a(x)$ as a sum of W_0 wavelets. The two expansions, which are shown in Figs. 7.14(e) and (f), divide $f(x)$ in a manner similar to a lowpass and highpass filter as discussed in connection with Fig. 7.6. The low frequencies of $f(x)$ are captured in $f_a(x)$—it assumes the average value of $f(x)$ in each integer interval—while the high-frequency details are encoded in $f_d(x)$. ■

7.3 Wavelet Transforms in One Dimension

We can now formally define several closely related wavelet transformations: the generalized *wavelet series expansion*, the *discrete wavelet transform*, and the *continuous wavelet transform*. Their counterparts in the Fourier domain are the Fourier series expansion, the discrete Fourier transform, and the integral Fourier transform, respectively. In Section 7.4, we develop a computationally efficient implementation of the discrete wavelet transform called the *fast wavelet transform*.

7.3.1 The Wavelet Series Expansions

We begin by defining the *wavelet series expansion* of function $f(x) \in L^2(\mathbf{R})$ relative to wavelet $\psi(x)$ and scaling function $\varphi(x)$. In accordance with Eq. (7.2-27), $f(x)$ can be represented by a scaling function expansion in subspace V_{j_0} [Eq. (7.2-12) defines such an expansion] and some number of wavelet function expansions in subspaces $W_{j_0}, W_{j_0+1}, \ldots$ [as defined in Eq. (7.2-21)]. Thus,

$$f(x) = \sum_k c_{j_0}(k)\varphi_{j_0,k}(x) + \sum_{j=j_0}^{\infty}\sum_k d_j(k)\psi_{j,k}(x) \tag{7.3-1}$$

where j_0 is an arbitrary starting scale and the $c_{j_0}(k)$ and $d_j(k)$ are relabeled α_k from Eqs. (7.2-12) and (7.2-21), respectively. The $c_{j_0}(k)$ are normally called *approximation* and/or *scaling coefficients*; the $d_j(k)$ are referred to as *detail* and/or *wavelet coefficients*. This is because the first sum in Eq. (7.3-1) uses scaling functions to provide an approximation of $f(x)$ at scale j_0 [unless $f(x) \in V_{j_0}$ so that the sum of the scaling functions is equal to $f(x)$]. For each higher scale $j \geq j_0$ in the second sum, a finer resolution function—a sum of wavelets—is added to the approximation to provide increasing detail. If the expansion

functions form an orthonormal basis or tight frame, which is often the case, the expansion coefficients are calculated—based on Eqs. (7.2-5) and (7.2-9)—as

$$c_{j_0}(k) = \langle f(x), \varphi_{j_0,k}(x) \rangle = \int f(x)\varphi_{j_0,k}(x)\,dx \tag{7.3-2}$$

Because f is real, no conjugates are needed in the inner products of Eqs. (7.3-2) and (7.3-3).

and

$$d_j(k) = \langle f(x), \psi_{j,k}(x) \rangle = \int f(x)\psi_{j,k}(x)\,dx \tag{7.3-3}$$

In Eqs. (7.2-5) and (7.2-9), the expansion coefficients (i.e., the α_k) are defined as inner products of the function being expanded and the expansion functions being used. In Eqs. (7.3-2) and (7.3-3), the expansion functions are the $\varphi_{j_0,k}$ and $\psi_{j,k}$; the expansion coefficients are the c_{j_0} and d_j. If the expansion functions are part of a biorthogonal basis, the φ and ψ terms in these equations must be replaced by their dual functions, $\widetilde{\varphi}$ and $\widetilde{\psi}$, respectively.

EXAMPLE 7.7: The Haar wavelet series expansion of $y = x^2$.

■ Consider the simple function

$$y = \begin{cases} x^2 & 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}$$

shown in Fig. 7.15(a). Using Haar wavelets—see Eqs. (7.2-14) and (7.2-30)—and a starting scale $j_0 = 0$, Eqs. (7.3-2) and (7.3-3) can be used to compute the following expansion coefficients:

$$c_0(0) = \int_0^1 x^2\varphi_{0,0}(x)\,dx = \int_0^1 x^2\,dx = \frac{x^3}{3}\bigg|_0^1 = \frac{1}{3}$$

$$d_0(0) = \int_0^1 x^2\psi_{0,0}(x)\,dx = \int_0^{0.5} x^2\,dx - \int_{0.5}^1 x^2\,dx = -\frac{1}{4}$$

$$d_1(0) = \int_0^1 x^2\psi_{1,0}(x)\,dx = \int_0^{0.25} x^2\sqrt{2}\,dx - \int_{0.25}^{0.5} x^2\sqrt{2}\,dx = -\frac{\sqrt{2}}{32}$$

$$d_1(1) = \int_0^1 x^2\psi_{1,1}(x)\,dx = \int_{0.5}^{0.75} x^2\sqrt{2}\,dx - \int_{0.75}^1 x^2\sqrt{2}\,dx = -\frac{3\sqrt{2}}{32}$$

Substituting these values into Eq. (7.3-1), we get the wavelet series expansion

$$y = \underbrace{\underbrace{\frac{1}{3}\varphi_{0,0}(x)}_{V_0} + \underbrace{\left[-\frac{1}{4}\psi_{0,0}(x)\right]}_{W_0}}_{V_1 = V_0 \oplus W_0} + \underbrace{\left[-\frac{\sqrt{2}}{32}\psi_{1,0}(x) - \frac{3\sqrt{2}}{32}\psi_{1,1}(x)\right]}_{W_1} + \cdots$$

$$V_2 = V_1 \oplus W_1 = V_0 \oplus W_0 \oplus W_1$$

a b
c d
e f

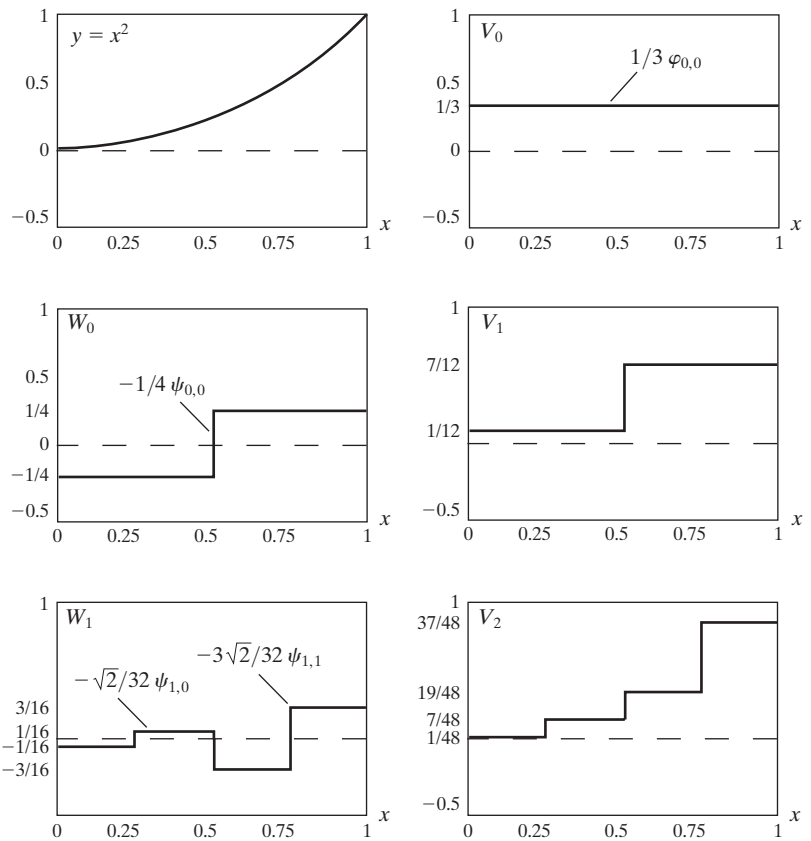


FIGURE 7.15
A wavelet series expansion of $y = x^2$ using Haar wavelets.

The first term in this expansion uses $c_0(0)$ to generate a subspace V_0 approximation of the function being expanded. This approximation is shown in Fig. 7.15(b) and is the average value of the original function. The second term uses $d_0(0)$ to refine the approximation by adding a level of detail from subspace W_0. The added detail and resulting V_1 approximation are shown in Figs. 7.15(c) and (d), respectively. Another level of detail is added by the subspace W_1 coefficients $d_1(0)$ and $d_1(1)$. This additional detail is shown in Fig. 7.15(e), and the resulting V_2 approximation is depicted in 7.15(f). Note that the expansion is now beginning to resemble the original function. As higher scales (greater levels of detail) are added, the approximation becomes a more precise representation of the function, realizing it in the limit as $j \rightarrow \infty$. ■

7.3.2 The Discrete Wavelet Transform

Like the Fourier series expansion, the wavelet series expansion of the previous section maps a function of a continuous variable into a sequence of coefficients. If the function being expanded is discrete (i.e., a sequence of numbers), the resulting coefficients are called the *discrete wavelet transform* (DWT). For example, if $f(n) = f(x_0 + n\,\Delta x)$ for some x_0, Δx, and $n = 0, 1, 2, \ldots, M - 1$,

the wavelet series expansion coefficients for $f(x)$ [defined by Eqs. (7.3-2) and (7.3-3)] become the *forward* DWT coefficients for sequence $f(n)$:

$$W_\varphi(j_0, k) = \frac{1}{\sqrt{M}} \sum_n f(n)\varphi_{j_0,k}(n) \tag{7.3-5}$$

$$W_\psi(j, k) = \frac{1}{\sqrt{M}} \sum_n f(n)\psi_{j,k}(n) \quad \text{for } j \geq j_0 \tag{7.3-6}$$

The $\varphi_{j_0,k}(n)$ and $\psi_{j,k}(n)$ in these equations are sampled versions of basis functions $\varphi_{j_0,k}(x)$ and $\psi_{j,k}(x)$. For example, $\varphi_{j_0,k}(n) = \varphi_{j_0,k}(x_s + n\Delta x_s)$ for some x_s, Δx_s, and $n = 0, 1, 2, \ldots, M - 1$. Thus, we employ M equally spaced samples over the support of the basis functions (see Example 7.8 below). In accordance with Eq. (7.3-1), the complementary *inverse* DWT is

$$f(n) = \frac{1}{\sqrt{M}} \sum_k W_\varphi(j_0, k)\varphi_{j_0,k}(n) + \frac{1}{\sqrt{M}} \sum_{j=j_0}^{\infty} \sum_k W_\psi(j, k)\psi_{j,k}(n) \tag{7.3-7}$$

Normally, we let $j_0 = 0$ and select M to be a power of 2 (i.e., $M = 2^J$) so that the summations in Eqs. (7.3-5) through (7.3-7) are performed over $n = 0, 1, 2, \ldots, M - 1$, $j = 0, 1, 2, \ldots, J - 1$, and $k = 0, 1, 2, \ldots, 2^j - 1$. For Haar wavelets, the discretized scaling and wavelet functions employed in the transform (i.e., the basis functions) correspond to the rows of the $M \times M$ Haar transformation matrix of Section 7.1.3. The transform itself is composed of M coefficients, the minimum scale is 0, and the maximum scale is $J - 1$. For reasons noted in Section 7.3.1 and illustrated in Example 7.6, the coefficients defined in Eqs. (7.3-5) and (7.3-6) are usually called *approximation* and *detail coefficients*, respectively.

The $W_\varphi(j_0, k)$ and $W_\psi(j, k)$ in Eqs. (7.3-5) to (7.3-7) correspond to the $c_{j_0}(k)$ and $d_j(k)$ of the wavelet series expansion in the previous section. (This change of variables is not necessary but paves the way for the standard notation used for the continuous wavelet transform of the next section.) Note that the integrations in the series expansion have been replaced by summations, and a $1/\sqrt{M}$ normalizing factor, reminiscent of the DFT in Section 4.4.1, has been added to both the forward and inverse expressions. This factor alternately could be incorporated into the forward or inverse alone as $1/M$. Finally, it should be remembered that Eqs. (7.3-5) through (7.3-7) are valid for orthonormal bases and tight frames alone. For biorthogonal bases, the φ and ψ terms in Eqs. (7.3-5) and (7.3-6) must be replaced by their duals, $\tilde{\varphi}$ and $\tilde{\psi}$, respectively.

EXAMPLE 7.8: Computing a one-dimensional discrete wavelet transform.

■ To illustrate the use of Eqs. (7.3-5) through (7.3-7), consider the discrete function of four points: $f(0) = 1$, $f(1) = 4$, $f(2) = -3$, and $f(3) = 0$. Because $M = 4$, $J = 2$ and, with $j_0 = 0$, the summations are performed over $x = 0, 1, 2, 3$, $j = 0, 1$, and $k = 0$ for $j = 0$ or $k = 0, 1$ for $j = 1$. We will use the Haar scaling and wavelet functions and assume that the four samples of

$f(x)$ are distributed over the support of the basis functions, which is 1 in width. Substituting the four samples into Eq. (7.3-5), we find that

$$W_\varphi(0,0) = \frac{1}{2}\sum_{n=0}^{3} f(n)\varphi_{0,0}(n)$$
$$= \frac{1}{2}\left[1\cdot 1 + 4\cdot 1 - 3\cdot 1 + 0\cdot 1\right] = 1$$

because $\varphi_{0,0}(n) = 1$ for $n = 0, 1, 2, 3$. Note that we have employed uniformly spaced samples of the Haar scaling function for $j = 0$ and $k = 0$. The values correspond to the first row of Haar transformation matrix $\mathbf{H}_4$ of Section 7.1.3. Continuing with Eq. (7.3-6) and similarly spaced samples of $\psi_{j,k}(x)$, which correspond to rows 2, 3, and 4 of $\mathbf{H}_4$, we get

$$W_\psi(0,0) = \frac{1}{2}\left[1\cdot 1 + 4\cdot 1 - 3\cdot(-1) + 0\cdot(-1)\right] = 4$$

$$W_\psi(1,0) = \frac{1}{2}\left[1\cdot\sqrt{2} + 4\cdot\left(-\sqrt{2}\right) - 3\cdot 0 + 0\cdot 0\right] = -1.5\sqrt{2}$$

$$W_\psi(1,1) = \frac{1}{2}\left[1\cdot 0 + 4\cdot 0 - 3\cdot\sqrt{2} + 0\cdot\left(-\sqrt{2}\right)\right] = -1.5\sqrt{2}$$

Thus, the discrete wavelet transform of our simple four-sample function relative to the Haar wavelet and scaling function is $\left\{1, 4, -1.5\sqrt{2}, -1.5\sqrt{2}\right\}$, where the transform coefficients have been arranged in the order in which they were computed.

Equation (7.3-7) lets us reconstruct the original function from its transform. Iterating through its summation indices, we get

$$f(n) = \frac{1}{2}\left[W_\varphi(0,0)\,\varphi_{0,0}(n) + W_\psi(0,0)\psi_{0,0}(n) + W_\psi(1,0)\,\psi_{1,0}(n) + W_\psi(1,1)\psi_{1,1}(n)\right]$$

for $n = 0, 1, 2, 3$. If $n = 0$, for instance,

$$f(0) = \frac{1}{2}\left[1\cdot 1 + 4\cdot 1 - 1.5\sqrt{2}\cdot\left(\sqrt{2}\right) - 1.5\sqrt{2}\cdot 0\right] = 1$$

As in the forward case, uniformly spaced samples of the scaling and wavelet functions are used in the computation of the inverse. ■

The four-point DWT in the preceding example is an illustration of a two-scale decomposition of $f(n)$—that is, $j = \{0, 1\}$. The underlying assumption was that starting scale j_0 was zero, but other starting scales are possible. It is left as an exercise for the reader (see Problem 7.16) to compute the single-scale transform $\left\{2.5\sqrt{2}, -1.5\sqrt{2}, -1.5\sqrt{2}, -1.5\sqrt{2}\right\}$, which results when the starting scale is 1. Thus, Eqs. (7.3-5) and (7.3-6) define a "family" of transforms that differ in starting scale j_0.

7.3.3 The Continuous Wavelet Transform

The natural extension of the discrete wavelet transform is the *continuous wavelet transform* (CWT), which transforms a continuous function into a highly redundant function of two continuous variables—translation and scale. The resulting transform is easy to interpret and valuable for time-frequency analysis. Although our interest is in discrete images, the continuous transform is covered here for completeness.

The continuous wavelet transform of a continuous, square-integrable function, $f(x)$, relative to a real-valued wavelet, $\psi(x)$, is defined as

$$W_\psi(s, \tau) = \int_{-\infty}^{\infty} f(x)\psi_{s,\tau}(x)\, dx \tag{7.3-8}$$

where

$$\psi_{s,\tau}(x) = \frac{1}{\sqrt{s}}\psi\left(\frac{x - \tau}{s}\right) \tag{7.3-9}$$

and s and τ are called *scale* and *translation* parameters, respectively. Given $W_\psi(s, \tau)$, $f(x)$ can be obtained using the *inverse continuous wavelet transform*

$$f(x) = \frac{1}{C_\psi}\int_0^{\infty}\int_{-\infty}^{\infty} W_\psi(s, \tau)\frac{\psi_{s,\tau}(x)}{s^2}d\tau\, ds \tag{7.3-10}$$

where

$$C_\psi = \int_{-\infty}^{\infty}\frac{|\Psi(\mu)|^2}{|\mu|}d\mu \tag{7.3-11}$$

and $\Psi(\mu)$ is the Fourier transform of $\psi(x)$. Equations (7.3-8) through (7.3-11) define a reversible transformation as long as the so-called *admissibility criterion*, $C_\psi < \infty$, is satisfied (Grossman and Morlet [1984]). In most cases, this simply means that $\Psi(0) = 0$ and $\Psi(\mu) \to 0$ as $\mu \to \infty$ fast enough to make $C_\psi < \infty$.

The preceding equations are reminiscent of their discrete counterparts—Eqs. (7.2-19), (7.3-1), (7.3-3), (7.3-6), and (7.3-7). The following similarities should be noted:

1. The continuous translation parameter, τ, takes the place of the integer translation parameter, k.
2. The continuous scale parameter, s, is inversely related to the binary scale parameter, 2^j. This is because s appears in the denominator of $\psi\big((x - \tau)/s\big)$ in Eq. (7.3-9). Thus, wavelets used in continuous transforms are compressed or reduced in width when $0 < s < 1$ and dilated or expanded when $s > 1$. Wavelet scale and our traditional notion of frequency are inversely related.

3. The continuous transform is similar to a series expansion [see Eq. (7.3-1)] or discrete transform [see Eq. (7.3-6)] in which the starting scale $j_0 = -\infty$. This—in accordance with Eq. (7.2-26)—eliminates explicit scaling function dependence, so that the function is represented in terms of wavelets alone.
4. Like the discrete transform, the continuous transform can be viewed as a set of transform coefficients, $\{W_\psi(s, \tau)\}$, that measure the similarity of $f(x)$ with a set of basis functions, $\{\psi_{s,\tau}(x)\}$. In the continuous case, however, both sets are infinite. Because $\psi_{s,\tau}(x)$ is real valued and $\psi_{s,\tau}(x) = \psi^*_{s,\tau}(x)$, each coefficient from Eq. (7.3-8) is the integral inner product, $\langle f(x), \psi_{s,\tau}(x)\rangle$, of $f(x)$ and $\psi_{s,\tau}(x)$.

EXAMPLE 7.9: A one-dimensional continuous wavelet transform.

■ The *Mexican hat* wavelet,

$$\psi(x) = \left(\frac{2}{\sqrt{3}}\pi^{-1/4}\right)(1 - x^2)e^{-x^2/2} \tag{7.3-12}$$

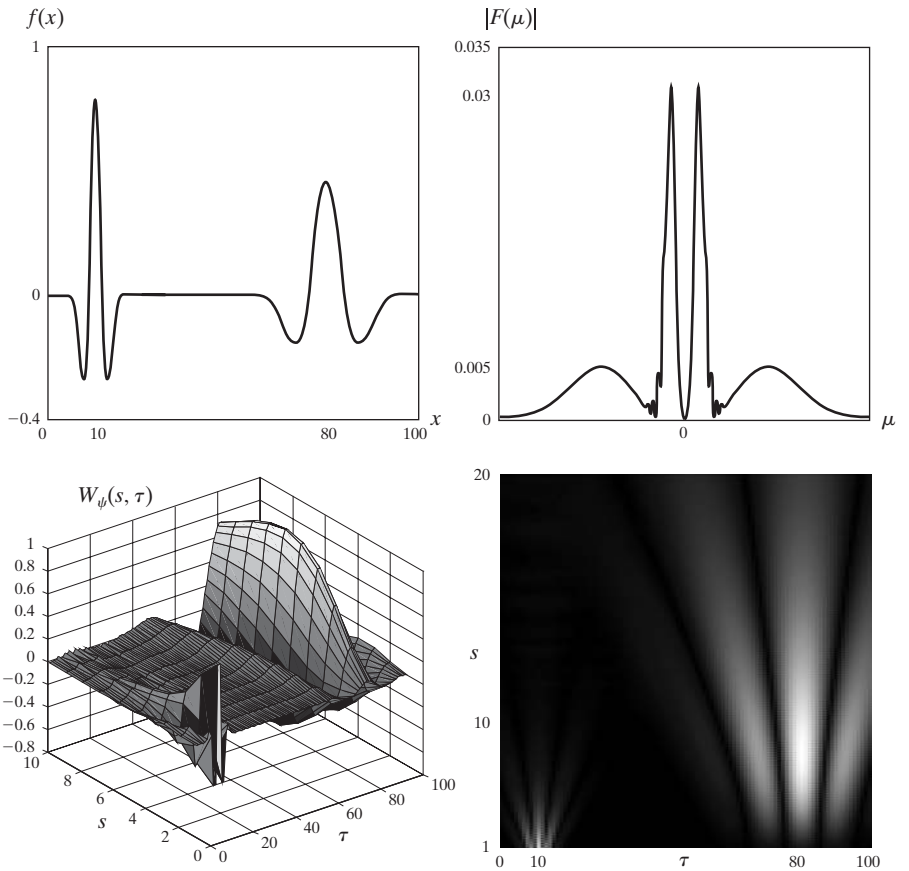
gets its name from its distinctive shape [see Fig. 7.16(a)]. It is proportional to the second derivative of the Gaussian probability function, has an average value of 0, and is compactly supported (i.e., dies out rapidly as $|x| \to \infty$). Although it satisfies the admissibility requirement for the existence of continuous, reversible transforms, there is not an associated scaling function, and the computed transform does not result in an orthogonal analysis. Its most distinguishing features are its symmetry and the existence of the explicit expression of Eq. (7.3-12).

The continuous, one-dimensional function in Fig. 7.16(a) is the sum of two Mexican hat wavelets:

$$f(x) = \psi_{1,10}(x) + \psi_{6,80}(x)$$

Its Fourier spectrum, shown in Fig. 7.16(b), reveals the close connection between scaled wavelets and Fourier frequency bands. The spectrum contains two broad frequency bands (or peaks) that correspond to the two Gaussian-like perturbations of the function.

Figure 7.16(c) shows a portion ($1 \le s \le 10$ and $\tau \le 100$) of the CWT of the function in Fig. 7.16(a) relative to the Mexican hat wavelet. Unlike the Fourier spectrum in Fig. 7.16(b), it provides both spatial and frequency information. Note, for example, that when $s = 1$, the transform achieves a maximum at $\tau = 10$, which corresponds to the location of the $\psi_{1,10}(x)$ component of $f(x)$. Because the transform provides an objective measure of the similarity between $f(x)$ and the wavelets for which it is computed, it is easy to see how it can be used for feature detection. We simply need wavelets that match the features of interest. Similar observations can be drawn from the intensity plot in Fig. 7.16(d), where the absolute value of the transform $|W_\psi(s, \tau)|$ is displayed as intensities between black and white. Note that the continuous wavelet transform turns a 1-D function into a 2-D result. ■



a b
c d

FIGURE 7.16
The continuous wavelet transform (c and d) and Fourier spectrum (b) of a continuous 1-D function (a).

7.4 The Fast Wavelet Transform

The *fast wavelet transform* (FWT) is a computationally efficient implementation of the discrete wavelet transform (DWT) that exploits a surprising but fortunate relationship between the coefficients of the DWT at adjacent scales. Also called *Mallat's herringbone algorithm* (Mallat [1989a, 1989b]), the FWT resembles the two-band subband coding scheme of Section 7.1.2.

Consider again the multiresolution refinement equation

$$\varphi(x) = \sum_n h_\varphi(n)\sqrt{2}\varphi(2x - n) \tag{7.4-1}$$

Equation (7.4-1) is Eq. (7.2-18) of Section 7.2.2.

Scaling x by 2^j, translating it by k, and letting $m = 2k + n$ gives

$$\begin{aligned}\varphi(2^j x - k) &= \sum_n h_\varphi(n)\sqrt{2}\varphi\left(2(2^j x - k) - n\right) \\ &= \sum_m h_\varphi(n)\sqrt{2}\varphi(2^{j+1}x - 2k - n) \\ &= \sum_m h_\varphi(m - 2k)\sqrt{2}\varphi(2^{j+1}x - m)\end{aligned} \tag{7.4-2}$$

Note that scaling vector h_φ can be thought of as the "weights" used to expand $\varphi(2^j x - k)$ as a sum of scale $j + 1$ scaling functions. A similar sequence of operations—beginning with Eq. (7.2-28)—provides an analogous result for $\psi(2^j x - k)$. That is,

$$\psi(2^j x - k) = \sum_m h_\psi(m - 2k)\sqrt{2}\varphi(2^{j+1}x - m) \tag{7.4-3}$$

where scaling vector $h_\varphi(n)$ in Eq. (7.4-2) corresponds to wavelet vector $h_\psi(n)$ in Eq. (7.4-3).

The wavelet series expansion coefficients become the DWT coefficient when f is discrete. Here, we begin with the series expansion coefficients to simplify the derivation; we will be able to substitute freely from earlier results (like the scaling and wavelet function definitions).

Now consider Eqs. (7.3-2) and (7.3-3) of Section 7.3.1. They define the wavelet series expansion coefficients of continuous function $f(x)$. Substituting Eq. (7.2-19)—the wavelet defining equation—into Eq. (7.3-3), we get

$$d_j(k) = \int f(x)2^{j/2}\psi(2^j x - k)\,dx \tag{7.4-4}$$

which, upon replacing $\psi(2^j x - k)$ with the right side of Eq. (7.4-3), becomes

$$d_j(k) = \int f(x)2^{j/2}\left[\sum_m h_\psi(m - 2k)\sqrt{2}\varphi(2^{j+1}x - m)\right]dx \tag{7.4-5}$$

Interchanging the sum and integral and rearranging terms then gives

$$d_j(k) = \sum_m h_\psi(m - 2k)\left[\int f(x)2^{(j+1)/2}\varphi(2^{j+1}x - m)\right] \tag{7.4-6}$$

where the bracketed quantity is $c_{j_0}(k)$ of Eq. (7.3-2) with $j_0 = j + 1$ and $k = m$. To see this, substitute Eq. (7.2-10) into Eq. (7.3-2) and replace j_0 and k with $j + 1$ and m, respectively. Therefore, we can write

$$d_j(k) = \sum_m h_\psi(m - 2k)c_{j+1}(m) \tag{7.4-7}$$

and note that the detail coefficients at scale j are a function of the approximation coefficients at scale $j + 1$. Using Eqs. (7.4-2) and (7.3-2) as the starting point of a similar derivation involving the wavelet series expansion (and DWT) approximation coefficients, we find similarly that

$$c_j(k) = \sum_m h_\varphi(m - 2k)c_{j+1}(m) \tag{7.4-8}$$

Because the $c_j(k)$ and $d_j(k)$ coefficients of the wavelet series expansion become the $W_\varphi(j, k)$ and $W_\psi(j, k)$ coefficients of the DWT when $f(x)$ is discrete (see Section 7.3.2), we can write

$$W_\psi(j, k) = \sum_m h_\psi(m - 2k) W_\varphi(j + 1, m) \tag{7.4-9}$$

$$W_\varphi(j, k) = \sum_m h_\varphi(m - 2k) W_\varphi(j + 1, m) \tag{7.4-10}$$

Equations (7.4-9) and (7.4-10) reveal a remarkable relationship between the DWT coefficients of adjacent scales. Comparing these results to Eq. (7.1-7), we see that both $W_\varphi(j, k)$ and $W_\psi(j, k)$, the scale j approximation and the detail coefficients, can be computed by convolving $W_\varphi(j + 1, k)$, the scale $j + 1$ approximation coefficients, with the order-reversed scaling and wavelet vectors, $h_\varphi(-n)$ and $h_\psi(-n)$, and subsampling the results. Figure 7.17 summarizes these operations in block diagram form. Note that this diagram is identical to the analysis portion of the two-band subband coding and decoding system of Fig. 7.6, with $h_0(n) = h_\varphi(-n)$ and $h_1(n) = h_\psi(-n)$. Therefore, we can write

$$W_\psi(j, k) = h_\psi(-n) \star W_\varphi(j + 1, n)\Big|_{n=2k, k\geq 0} \tag{7.4-11}$$

and

$$W_\varphi(j, k) = h_\varphi(-n) \star W_\varphi(j + 1, n)\Big|_{n=2k, k\geq 0} \tag{7.4-12}$$

where the convolutions are evaluated at instants $n = 2k$ for $k \geq 0$. As will be shown in Example 7.10, evaluating convolutions at nonnegative, even indices is equivalent to filtering and downsampling by 2.

If $h_\varphi(m - 2k)$ in Eq. (7.4-9) is rewritten as $h_\varphi(-(2k - m))$, we see that the first minus sign is responsible for the order reversal [see Eq. (7.1-6)], the $2k$ is responsible for the subsampling [see Eq. (7.1-2)], and m is the dummy variable for convolution [see Eq. (7.1-7)].

Equations (7.4-11) and (7.4-12) are the defining equations for the computation of the fast wavelet transform. For a sequence of length $M = 2^J$, the number of mathematical operations involved is on the order of $O(M)$. That is, the number of multiplications and additions is linear with respect to the length of the input sequence—because the number of multiplications and additions involved in the convolutions performed by the FWT analysis bank in Fig. 7.17 is proportional to the length of the sequences being convolved. Thus, the FWT compares favorably with the FFT algorithm, which requires on the order of $O(M \log_2 M)$ operations.

To conclude the development of the FWT, we simply note that the filter bank in Fig. 7.17 can be "iterated" to create multistage structures for computing DWT coefficients at two or more successive scales. For example, Fig. 7.18(a) shows a two-stage filter bank for generating the coefficients at the two highest scales of the transform. Note that the highest scale coefficients are assumed to be samples of the function itself. That is, $W_\varphi(J, n) = f(n)$, where J is the highest

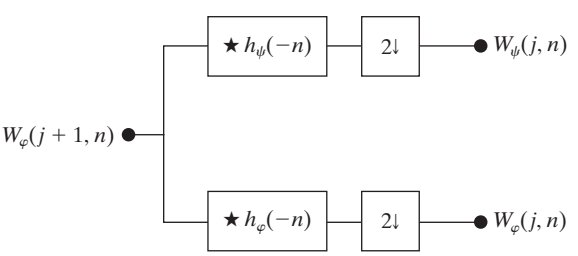


FIGURE 7.17 An FWT analysis bank.

a
b

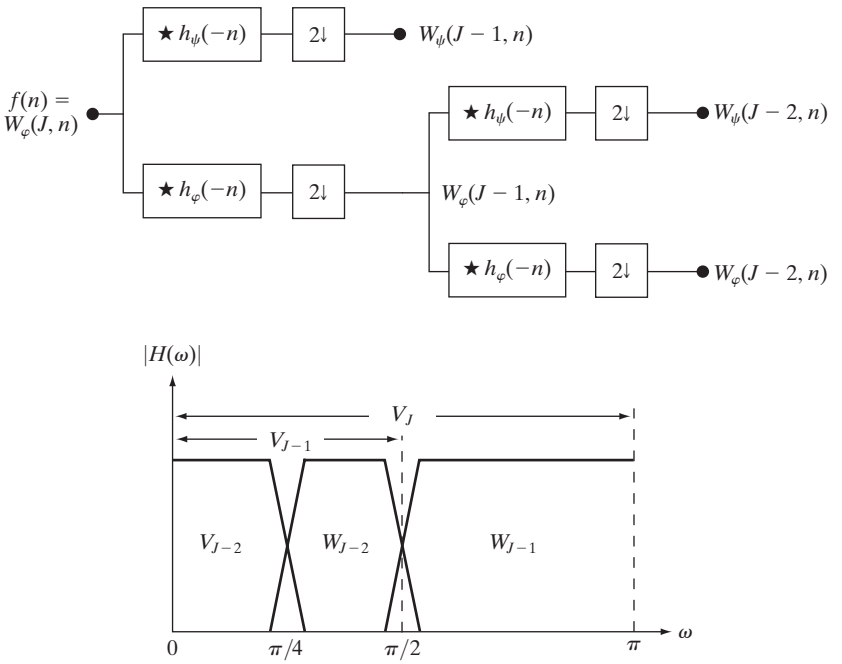


FIGURE 7.18
(a) A two-stage or two-scale FWT analysis bank and (b) its frequency splitting characteristics.

scale. [In accordance with Section 7.2.2, $f(x) \in V_J$, where V_J is the scaling space in which $f(x)$ resides.] The first filter bank in Fig. 7.18(a) splits the original function into a lowpass, approximation component, which corresponds to scaling coefficients $W_\varphi(J-1, n)$; and a highpass, detail component, corresponding to coefficients $W_\psi(J-1, n)$. This is illustrated graphically in Fig. 7.18(b), where scaling space V_J is split into wavelet subspace W_{J-1} and scaling subspace V_{J-1}. The spectrum of the original function is split into two half-band components. The second filter bank of Fig. 7.18(a) splits the spectrum and subspace V_{J-1}, the lower half-band, into quarter-band subspaces W_{J-2} and V_{J-2} with corresponding DWT coefficients $W_\psi(J-2, n)$ and $W_\varphi(J-2, n)$, respectively.

The two-stage filter bank of Fig. 7.18(a) is extended easily to any number of scales. A third filter bank, for example, would operate on the $W_\varphi(J-2, n)$ coefficients, splitting scaling space V_{J-2} into two eighth-band subspaces W_{J-3} and V_{J-3}. Normally, we choose 2^J samples of $f(x)$ and employ P filter banks (as in Fig. 7.17) to generate a P-scale FWT at scales $J-1, J-2, \ldots, J-P$. The highest scale (i.e., $J-1$) coefficients are computed first; the lowest scale (i.e., $J-P$) last. If function $f(x)$ is sampled above the Nyquist rate, as is usually the case, its samples are good approximations of the scaling coefficients at the sampling resolution and can be used as the starting high-resolution scaling coefficient inputs. In other words, no wavelet or detail coefficients are needed at the sampling scale. The highest-resolution scaling functions act as unit discrete impulse functions in Eqs. (7.3-5) and (7.3-6), allowing $f(n)$ to be used as the scaling (approximation) input to the first two-band filter bank (Odegard, Gopinath, and Burrus [1992]).

EXAMPLE 7.10: Computing a 1-D fast wavelet transform.

■ To illustrate the preceding concepts, consider the discrete function $f(n) = \{1, 4, -3, 0\}$ from Example 7.8. As in that example, we will compute the transform based on Haar scaling and wavelet functions. Here, however, we will not use the basis functions directly, as was done in the DWT of Example 7.8. Instead, we will use the corresponding scaling and wavelet vectors from Examples 7.5 and 7.6:

$$h_\varphi(n) = \begin{cases} 1/\sqrt{2} & n = 0, 1 \\ 0 & \text{otherwise} \end{cases} \tag{7.4-13}$$

and

$$h_\psi(n) = \begin{cases} 1/\sqrt{2} & n = 0 \\ -1/\sqrt{2} & n = 1 \\ 0 & \text{otherwise} \end{cases} \tag{7.4-14}$$

These are the functions used to build the FWT filter banks; they provide the filter coefficients. Note that because Haar scaling and wavelet functions are orthonormal, Eq. (7.1-14) can be used to generate the FWT filter coefficients from a single prototype filter—like $h_\varphi(n)$ in Table 7.2, which corresponds to $g_0(n)$ in Eq. (7.1-14):

Since the DWT computed in Example 7.8 was composed of elements $\{W_\varphi(0, 0), W_\psi(0, 0), W_\psi(1, 0), W_\psi(1, 1)\}$, we will compute the corresponding two-scale FWT for scales $j = \{0, 1\}$. That is, $J = 2$ (there are $2^J = 2^2$ samples) and $P = 2$ (we are working with scales $J - 1 = 2 - 1 = 1$ and $J - P = 2 - 2 = 0$ in that order). The transform will be computed using the two-stage filter bank of Fig. 7.18(a). Figure 7.19 shows the sequences that result from the required FWT convolutions and downsamplings. Note that function $f(n)$ itself is the scaling (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. 7.19, for example, we first convolve $f(n)$ with $h_\psi(-n)$. As explained in Section 3.4.2, this requires flipping one of the functions about the origin, sliding it past the other, and computing the sum of the point-wise 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. 7.19, underlined values represent negative indices, i.e., $n < 0$.) When downsampled by

n	$h_\varphi(n)$
0	$1/\sqrt{2}$
1	$1/\sqrt{2}$

TABLE 7.2 Orthonormal Haar filter coefficients for $h_\varphi(n)$.

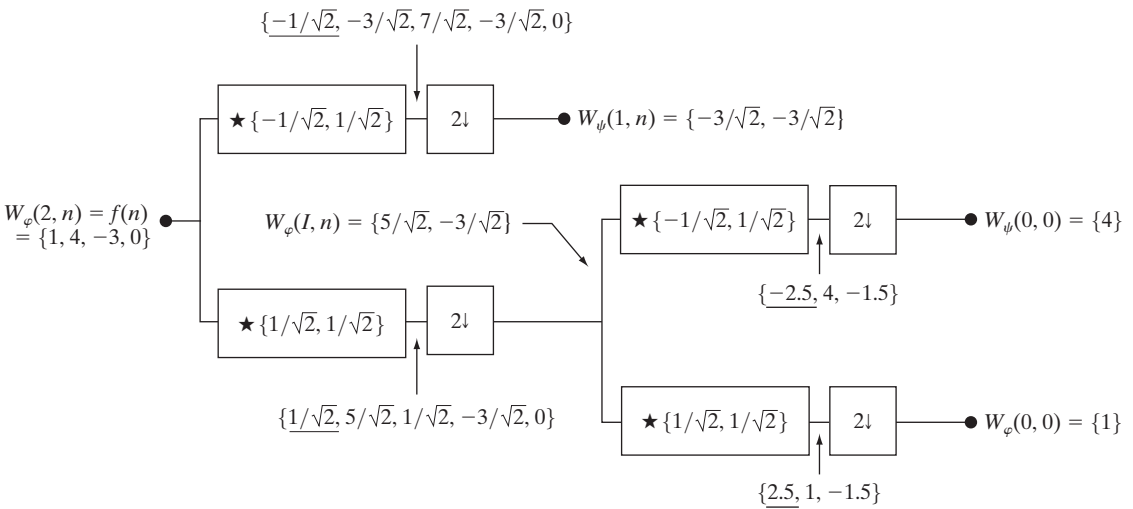


FIGURE 7.19 Computing a two-scale fast wavelet transform of sequence $\{1, 4, -3, 0\}$ using Haar scaling and wavelet vectors.

taking the even-indexed points, we get $W_\psi(1, k) = \left\{-3/\sqrt{2}, -3/\sqrt{2}\right\}$ for $k = \{0, 1\}$. Alternatively, we can use Eq. (7.4-12) to compute

$$W_\psi(1, k) = h_\psi(-n) \star W_\varphi(2, n)\Big|_{n=2k, k\geq 0} = h_\psi(-n) \star f(n)\Big|_{n=2k, k\geq 0}$$

$$= \sum_l h_\psi(l - 2k)x(l)\Big|_{k=0,1}$$

$$= \frac{1}{\sqrt{2}}\, x(2k) - \frac{1}{\sqrt{2}}\, x(2k + 1)\Big|_{k=0,1}$$

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 order-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 $\left\{-3/\sqrt{2}, -3/\sqrt{2}\right\}$, which matches the earlier result. The remaining convolutions and downsamplings are performed in a similar manner. ■

As one might expect, a fast inverse transform for the reconstruction of $f(n)$ from the results of the forward transform can 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

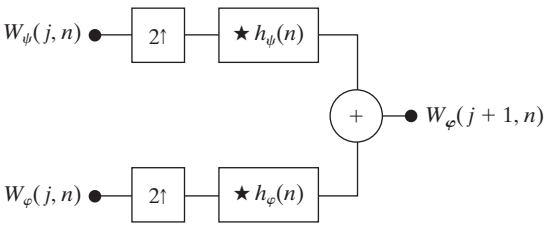


FIGURE 7.20 The FWT^{-1} synthesis filter bank.

and detail coefficients, to generate the level $j + 1$ approximation coefficients. Noting the similarity between the FWT analysis bank in Fig. 7.17 and the two-band subband analysis portion of Fig. 7.6(a), we can immediately postulate the required FWT^{-1} *synthesis filter bank*. Figure 7.20 details its structure, which is identical to the synthesis portion of the two-band subband coding and decoding system in Fig. 7.6(a). Equation (7.1-14) of Section 7.1.2 defines the 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 order-reversed versions of one another. Since the FWT analysis filters (see Fig. 7.17) 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 possible also to use biorthogonal analysis and synthesis filters, which are not order-reversed versions of one another. Biorthogonal analysis and synthesis filters are cross-modulated per Eqs. (7.1-10) and (7.1-11).

The FWT^{-1} filter bank in Fig. 7.20 implements the computation

$$W_\varphi(j+1, k) = h_\varphi(k) \star W_\varphi^{2\uparrow}(j, k) + h_\psi(k) \star W_\psi^{2\uparrow}(j, k) \Big|_{k \geq 0} \tag{7.4-15}$$

where $W^{2\uparrow}$ signifies upsampling by 2 [i.e., inserting zeros in W as defined by Eq. (7.1-1) 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 sequence $f(n)$ with greater detail and resolution is created. As with the forward FWT, the inverse filter bank can be iterated as shown in Fig. 7.21, 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 sequence $f(n)$.

Remember that like in pyramid coding (see Section 7.1.1), wavelet transforms can be computed at a user-specified number of scales. For a $2^J \times 2^J$ image, for example, there are $1 + \log_2 J$ possible scales.

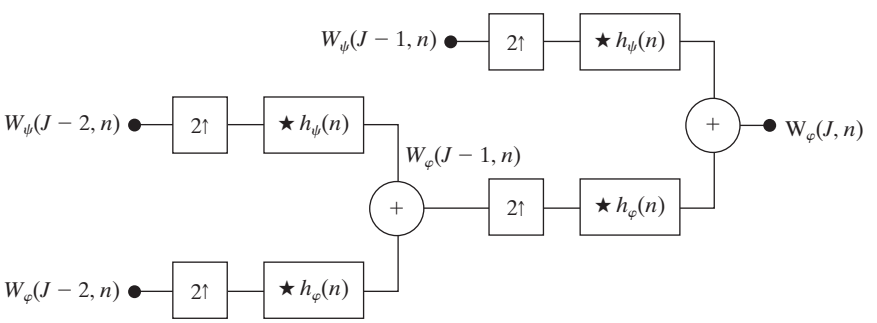


FIGURE 7.21 A two-stage or two-scale FWT^{-1} synthesis bank.

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

■ Computation of the inverse fast wavelet transform mirrors its forward counterpart. Figure 7.22 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 $g_0(n) = h_\varphi(n) = \left\{1/\sqrt{2}, 1/\sqrt{2}\right\}$ and $g_1(n) = h_\psi(n) = \left\{1/\sqrt{2}, -1/\sqrt{2}\right\}$ produces $\left\{1/\sqrt{2}, 1/\sqrt{2}, 0\right\}$ and $\left\{4/\sqrt{2}, -4/\sqrt{2}, 0\right\}$, which when added give $W_\varphi(1, n) = \left\{5/\sqrt{2}, -3/\sqrt{2}\right\}$. Thus, the level 1 approximation of Fig. 7.22, which matches the computed approximation in Fig. 7.19, 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 that 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. 7.17 for the Mexican hat wavelet; it does not satisfy the underlying assumptions of the FWT approach.

Finally, we note that while time and frequency usually are 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 accept some vagueness about frequency, and vice versa. This is the *Heisenberg uncertainty principle* applied to information processing. To illustrate the principle graphically, each basis function used in the representation of a function can be viewed schematically as a *tile* in a *time-frequency plane*. The tile, also called a *Heisenberg cell* or *Heisenberg box*, shows the frequency content of the basis function that it represents and where the basis function resides in time. Basis functions that are orthonormal are characterized by nonoverlapping tiles.

Figure 7.23 shows the time-frequency tiles for (a) an impulse function (i.e., conventional time domain) basis, (b) a sinusoidal (FFT) basis, and (c) an FWT

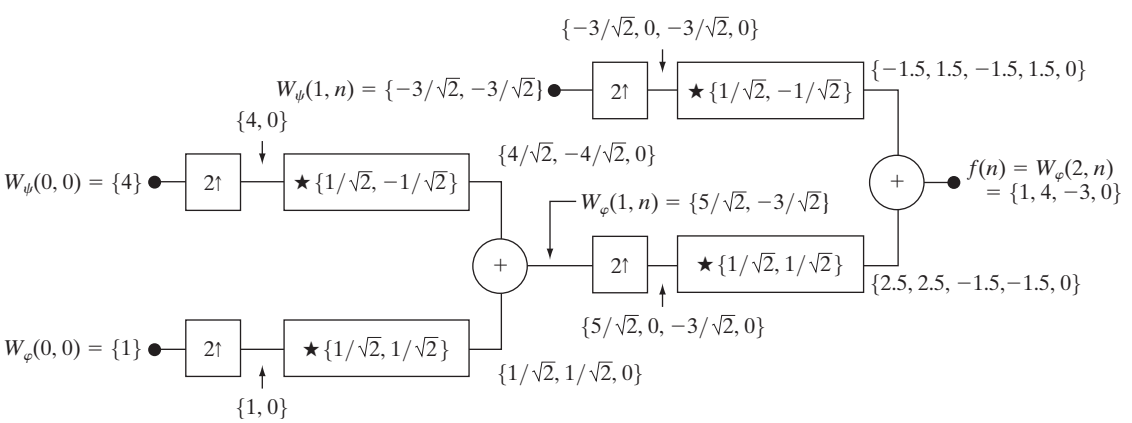
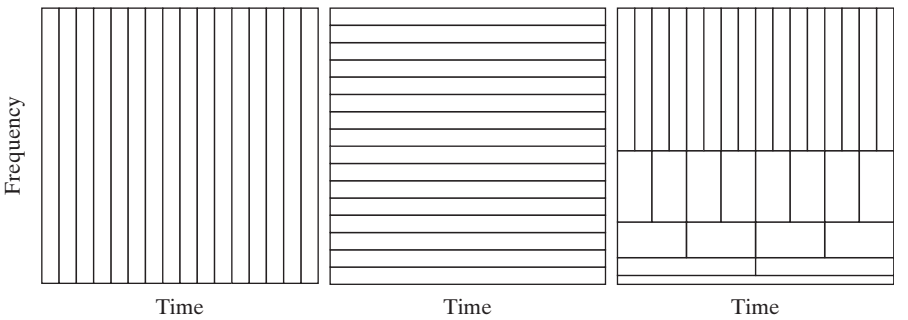


FIGURE 7.22 Computing a two-scale inverse fast wavelet transform of sequence $\left\{1, 4, -1.5\sqrt{2}, -1.5\sqrt{2}\right\}$ with Haar scaling and wavelet functions.



a b c

FIGURE 7.23 Time-frequency tilings for the basis functions associated with (a) sampled data, (b) the FFT, and (c) the FWT. Note that the horizontal strips of equal height rectangles in (c) represent FWT scales.

basis. Each tile is a rectangular region in Figs. 7.23(a) through (c); the height and width of the region defines the frequency and time characteristics of the functions that can be represented using the basis function. Note that the standard time domain basis in Fig. 7.23(a) pinpoints the instants when events occur but provides no frequency information [the width of each rectangle in Fig. 7.23(a) should be considered one instant in time]. Thus, to represent a single frequency sinusoid as an expansion using impulse basis functions, every basis function is required. The sinusoidal basis in Fig. 7.23(b), on the other hand, pinpoints the frequencies that are present in events that occur over long periods but provides no time resolution [the height of each rectangle in Fig. 7.23(b) should be considered a single frequency]. Thus, the single frequency sinusoid that was represented by an infinite number of impulse basis functions can be represented as an expansion involving one sinusoidal basis function. The time and frequency resolution of the FWT tiles in Fig. 7.23(c) vary, but the area of each tile (rectangle) is the same. 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). Thus, the FWT basis functions provide a compromise between the two limiting cases in Fig. 7.23(a) and (b). 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 two one-dimensional functions. 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 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 in Eqs. (7.5-2) to (7.5-4); it does not increase the computational complexity of the 2-D transform discussed in this section.

Given separable two-dimensional scaling and wavelet functions, extension of the 1-D DWT to two dimensions is straightforward. We first define the scaled and translated basis functions:

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

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

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 image $f(x, y)$ of size $M \times N$ is then

Now that we are dealing with 2-D images, $f(x, y)$ is a discrete function or sequence of values and x and y are discrete variables. The scaling and wavelet functions in Eq. (7.5-7) and (7.5-8) are sampled over their support (as was done in the 1-D case in Section 7.3.2).

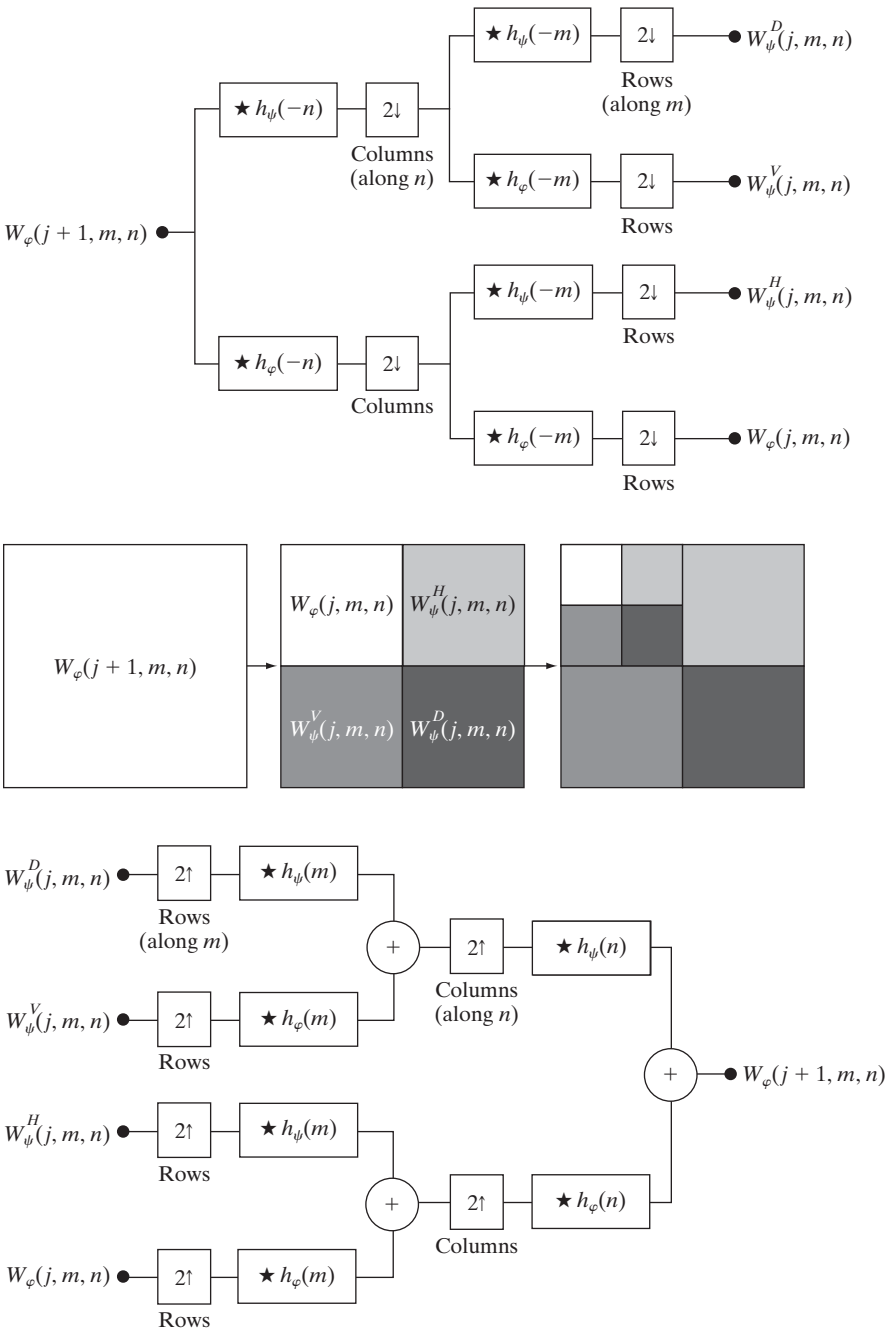
$$W_\varphi(j_0, m, n) = \frac{1}{\sqrt{MN}} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\varphi_{j_0,m,n}(x, y) \tag{7.5-7}$$

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

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 1-D discrete wavelet transform, the 2-D DWT can be implemented using digital filters and downsamplers. With separable two-dimensional scaling and wavelet functions, we simply take the 1-D FWT of the rows of $f(x, y)$, followed by the 1-D FWT of the resulting columns. Figure 7.24(a) shows the



a
b
c

FIGURE 7.24 The 2-D fast wavelet transform: (a) the analysis filter bank; (b) the resulting decomposition; and (c) the synthesis filter bank.

process in block diagram form. Note that, like its one-dimensional counterpart in Fig. 7.17, the 2-D 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. 7.24(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 is equal to $J - 1, J - 2, \ldots, J - P$. 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 high-pass 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_ψ^H, W_ψ^V, and W_ψ^D. These subimages, which are shown in the middle of Fig. 7.24(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. 7.24(b).

Note how W_φ, W_ψ^H, W_ψ^V, and W_ψ^D are arranged in Fig. 7.24(b). For each scale that is computed, they replace the previous scale approximation on which they were based.

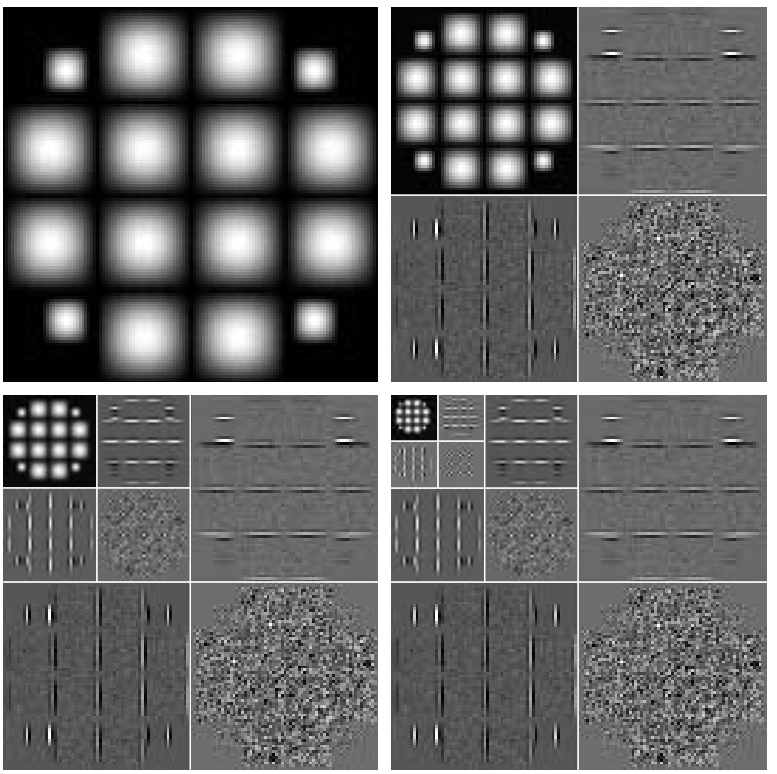
Figure 7.24(c) shows the synthesis filter bank that reverses the process just described. As would be expected, the reconstruction algorithm is similar to the one-dimensional case. At each iteration, four scale j approximation and detail subimages are upsampled and convolved with two one-dimensional filters—one 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.

■ Figure 7.25(a) is a 128×128 computer-generated image consisting of 2-D sine-like pulses on a black background. The objective of this example is to illustrate the mechanics involved in computing the 2-D FWT of this image. Figures 7.25(b) through (d) show three FWTs of the image in Fig. 7.25(a). The 2-D filter bank of Fig. 7.24(a) and the decomposition filters shown in Figs. 7.26(a) and (b) were used to generate all three results.

The scaling and wavelet vectors used in this example are described later. Our focus here is on the mechanics of the transform computation, which are independent of the filter coefficients employed.

Figure 7.25(b) shows the one-scale FWT of the image in Fig. 7.25(a). To compute this transform, the original image was used as the input to the filter bank of Fig. 7.24(a). The four resulting quarter-size decomposition outputs (i.e., the approximation and horizontal, vertical, and diagonal details) were then arranged in accordance with Fig. 7.24(b) to produce the image in Fig. 7.25(b). A similar process was used to generate the two-scale FWT in Fig. 7.25(c), but the input to the filter bank was changed to the quarter-size approximation subimage from the upper-left-hand corner of Fig. 7.25(b). As can be seen in Fig. 7.25(c), that quarter-size subimage was then replaced by the four quarter-size



a b
c d

FIGURE 7.25
Computing a 2-D three-scale FWT: (a) the original image; (b) a one-scale FWT; (c) a two-scale FWT; and (d) a three-scale FWT.

(now 1/16th of the size of the original image) decomposition results that were generated in the second filtering pass. Finally, Fig. 7.25(d) is the three-scale FWT that resulted when the subimage from the upper-left-hand corner of Fig. 7.25(c) was used as the filter bank input. Each pass through the filter bank produced four quarter-size output images that were 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 7.26(e) and (f) show the fourth-order

Recall that the compact support of a function is the interval in which the function has non-zero values.

[†]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
e f
g

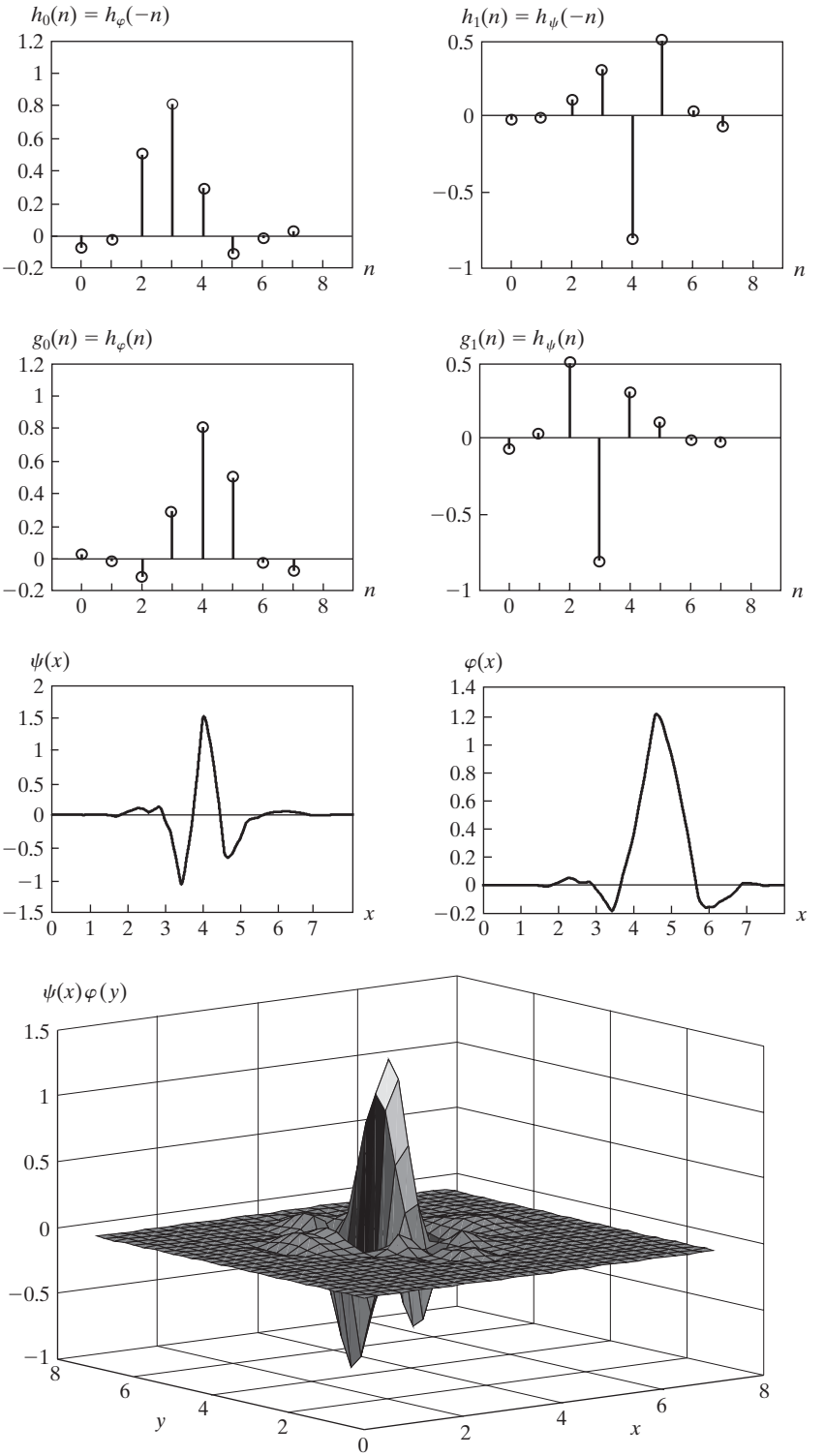


FIGURE 7.26 Fourth-order symlets: (a)–(b) decomposition filters; (c)–(d) reconstruction filters; (e) the one-dimensional wavelet; (f) the one-dimensional scaling function; and (g) one of three two-dimensional wavelets, $\psi^V(x, y)$. See Table 7.3 for the values of $h_\varphi(n)$ for $0 \leq n \leq 7$.

n	$h_\varphi(n)$
0	0.0322
1	−0.0126
2	−0.0992
3	0.2979
4	0.8037
5	0.4976
6	−0.0296
7	−0.0758

TABLE 7.3 Orthonormal fourth-order symlet filter coefficients for $h_\varphi(n)$. (Daubechies [1992].)

1-D symlets (i.e., wavelet and scaling functions). Figures 7.26(a) through (d) show the corresponding decomposition and reconstruction filters. The coefficients of lowpass reconstruction filter $g_0(n) = h_\varphi(n)$ for $0 \leq n \leq 7$ are given in Table 7.3. The coefficients of the remaining orthonormal filters are obtained using Eq. (7.1-14). Figure 7.26(g), a low-resolution graphic depiction of wavelet $\psi^V(x, y)$, is provided as an illustration of how a one-dimensional scaling and wavelet function can combine to form a separable, two-dimensional wavelet.

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

Step 1. Compute a 2-D wavelet transform of an image.
Step 2. Alter the transform.
Step 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 7.27 provides a simple illustration of the preceding three steps. In Fig. 7.27(a), the lowest scale approximation component of the discrete wavelet transform shown in Fig. 7.25(c) has been eliminated by setting its values to zero. As Fig. 7.27(b) shows, the net effect of computing the inverse wavelet transform using these modified coefficients is edge enhancement, reminiscent of the Fourier-based image sharpening results discussed in Section 4.9. Note how well the transitions between signal and background are delineated, despite the fact that they are relatively soft, sinusoidal transitions. By zeroing the horizontal details as well—see Figs. 7.27(c) and (d)—we can isolate the vertical edges. ■

a b
c d

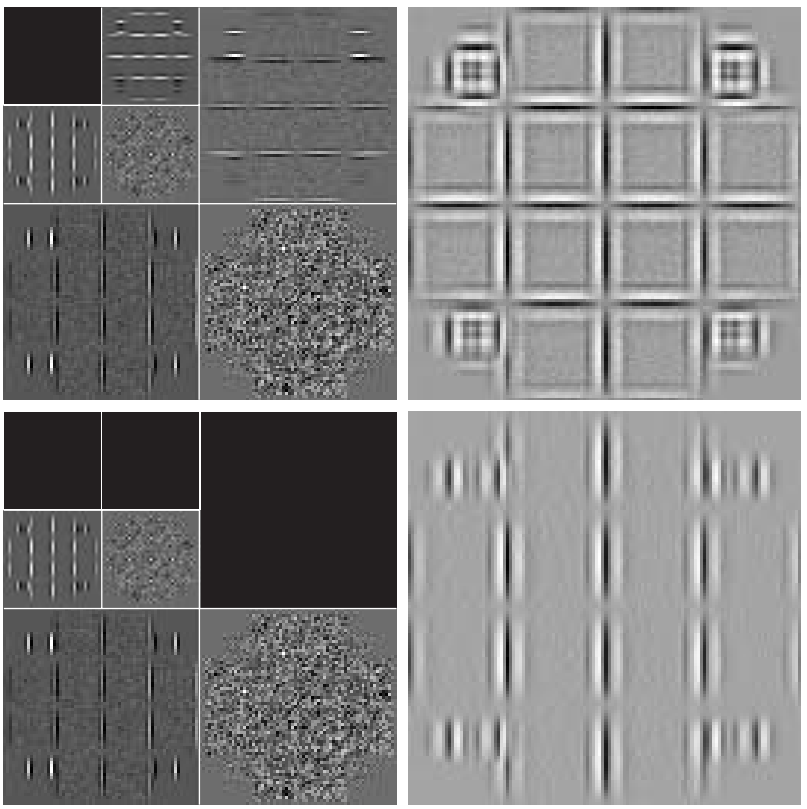


FIGURE 7.27 Modifying a DWT for edge detection: (a) and (c) two-scale decompositions with selected coefficients deleted; (b) and (d) the corresponding reconstructions.

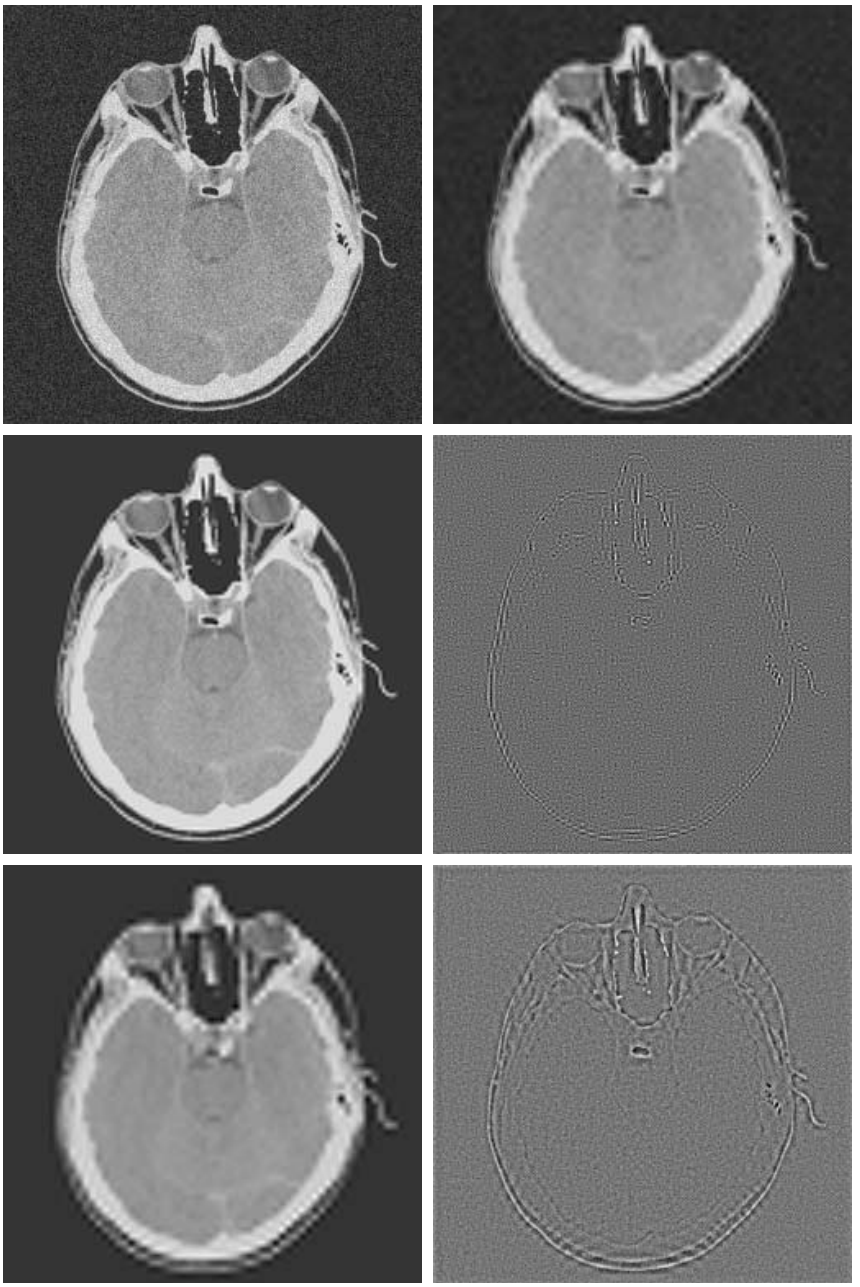
EXAMPLE 7.14: Wavelet-based noise removal.

■ As a second example, consider the CT image of a human head shown in Fig. 7.28(a). As can be seen in the background, the image has been uniformly corrupted with additive white noise. A general wavelet-based procedure for *denoising* the image (i.e., suppressing the noise part) is as follows:

Step 1. Choose a wavelet (e.g. Haar, symlet, ...) and number of levels (scales), P, for the decomposition. Then compute the FWT of the noisy image.

Step 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. (See Chapter 10 for a discussion of thresholding.)

Step 3. Compute the inverse wavelet transform (i.e., perform a wavelet reconstruction) using the original approximation coefficients at level $J - P$ and the modified detail coefficients for levels $J - 1$ to $J - P$.



a b
c d
e f

FIGURE 7.28
Modifying a DWT for noise removal: (a) a noisy CT 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 Vanderbilt University Medical Center.)

Figure 7.28(b) shows the result of performing these operations with fourth-order symlets, two scales (i.e., $P = 2$), and a global threshold that was determined interactively. Note the reduction in noise and blurring of image edges. This loss of edge detail is reduced significantly in Fig. 7.28(c), which

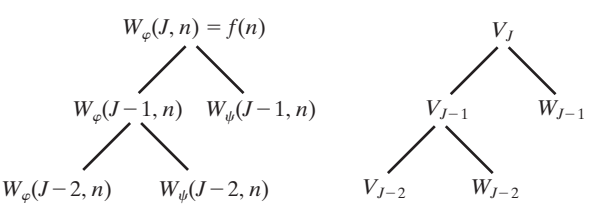
was generated by simply zeroing the highest-resolution detail coefficients (not thresholding the lower-resolution details) and reconstructing the image. Here, almost all of the background noise has been eliminated and the edges are only slightly disturbed. The difference image in Fig. 7.28(d) shows the information that is lost in the process. This result was generated by computing the inverse FWT of the two-scale transform with all but the highest-resolution detail coefficients zeroed. As can be seen, the resulting image contains most of the noise in the original image and some of the edge information. Figures 7.28(e) and (f) are included to show the negative effect of deleting all the detail coefficients. That is, Fig. 7.28(e) is a reconstruction of the DWT in which the details at both levels of the two-scale transform have been zeroed; Fig. 7.28(f) shows the information that is lost. Note the significant increase in edge information in Fig. 7.28(f) and the corresponding decrease in edge detail in Fig. 7.28(e). ■

Because only the highest resolution detail coefficients were kept when generating Fig. 7.28(d), the inverse transform is their contribution to the image. In the same way, Fig. 7.28(f) is the contribution of all the detail coefficients.

7.6 Wavelet Packets

The fast wavelet transform decomposes a function into a sum of scaling and wavelet functions whose bandwidths are logarithmically related. That is, the low frequency content (of the function) is represented using (scaling and wavelet) functions with narrow bandwidths, while the high-frequency content is represented using functions with wider bandwidths. If you look along the frequency axis of the time-frequency plane in Fig. 7.23(c), this is immediately apparent. Each horizontal strip of constant height tiles, which contains the basis functions for a single FWT scale, increases logarithmically in height as you move up the frequency axis. 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 $O(M \log_2 M)$ for a wavelet packet.

Consider again the two-scale filter bank of Fig. 7.18(a)—but imagine the decomposition as a *binary tree*. Figure 7.29(a) details the structure of the tree, and links the appropriate FWT scaling and wavelet coefficients [from Fig. 7.18(a)] to its *nodes*. The *root node* is assigned the highest-scale approximation coefficients,



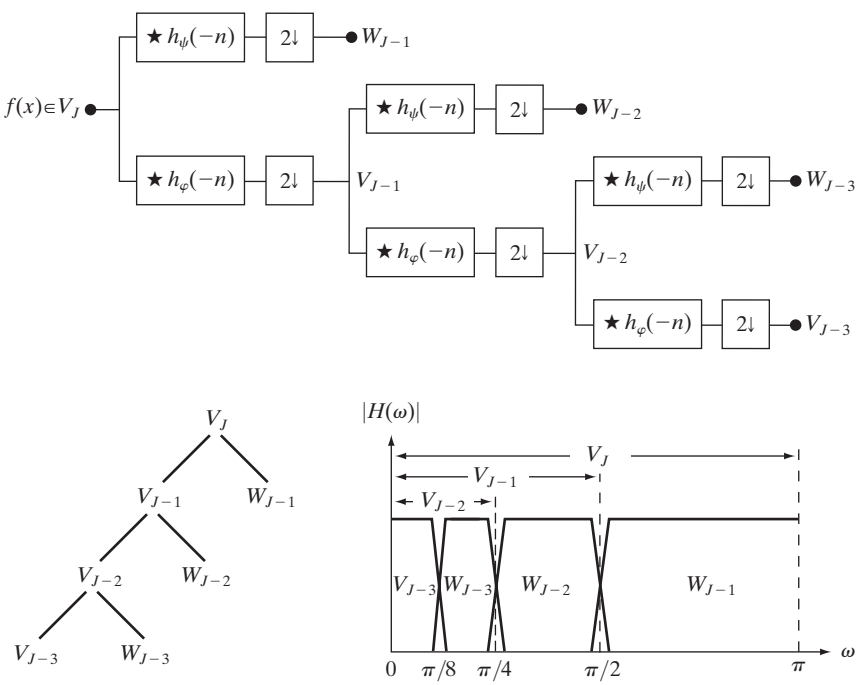
a b

FIGURE 7.29 An (a) coefficient tree and (b) analysis tree for the two-scale FWT analysis bank of Fig. 7.18.

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 $f(n)$. Because 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. 7.29(a) by the corresponding subspace. The result is the *subspace analysis tree* of Fig. 7.29(b). Although the variable W is used to denote both coefficients and subspaces, the two quantities are distinguishable by the format of their subscripts.

These concepts are further illustrated in Fig. 7.30, where a three-scale FWT analysis bank, analysis tree, and corresponding frequency spectrum are depicted. Unlike Fig. 7.18(a), the block diagram of Fig. 7.30(a) is labeled to resemble the analysis tree in Fig. 7.30(b)—as well as the spectrum in Fig. 7.30(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 (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



a
b c

FIGURE 7.30 A three-scale FWT filter bank: (a) block diagram; (b) decomposition space tree; and (c) spectrum splitting characteristics.

easy to detect valid decompositions. The three-scale analysis tree of Fig. 7.30(b), for example, makes possible 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.

Analysis trees also are an efficient mechanism for representing *wavelet packets*, which are nothing more than *conventional wavelet transforms in which the details are filtered iteratively*. Thus, the three-scale FWT analysis tree of Fig. 7.30(b) becomes the three-scale *wavelet packet* tree of Fig. 7.31. 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 As and Ds—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. 7.31) through an additional detail filter (yielding $W_{J-1,D}$), followed by an approximation filter (giving $W_{J-1,DA}$). Figures 7.32(a) and (b) are the filter bank and spectrum splitting characteristics of the analysis tree in Fig. 7.31. Note that the "naturally ordered" outputs of the filter bank in Fig. 7.32(a) have been reordered based on frequency content in Fig. 7.32(b) (see Problem 7.25 for more on "frequency ordered" wavelets).

The three-scale packet tree in Fig. 7.31 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 (base 2) relationship between the bandwidths of the scaling and wavelet spaces used in the representation of a function [see Figure 7.30(c)]. Thus, while the three-scale FWT analysis tree of Fig. 7.30(a) offers three possible decompositions—defined by Eqs. (7.6-1) to (7.6-3)—the wavelet packet tree of Fig. 7.31 supports 26 different decompositions. For instance, V_J [and therefore function $f(n)$] can be expanded as

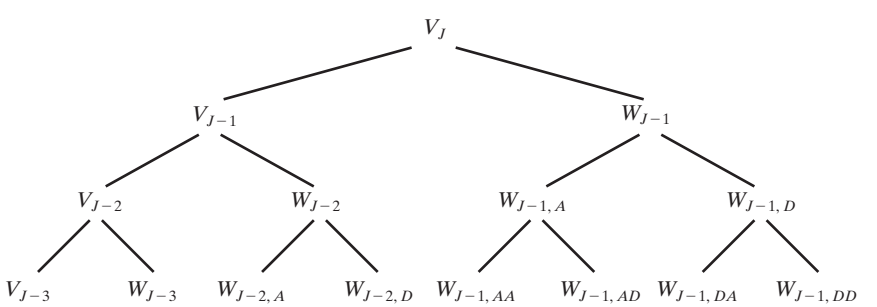
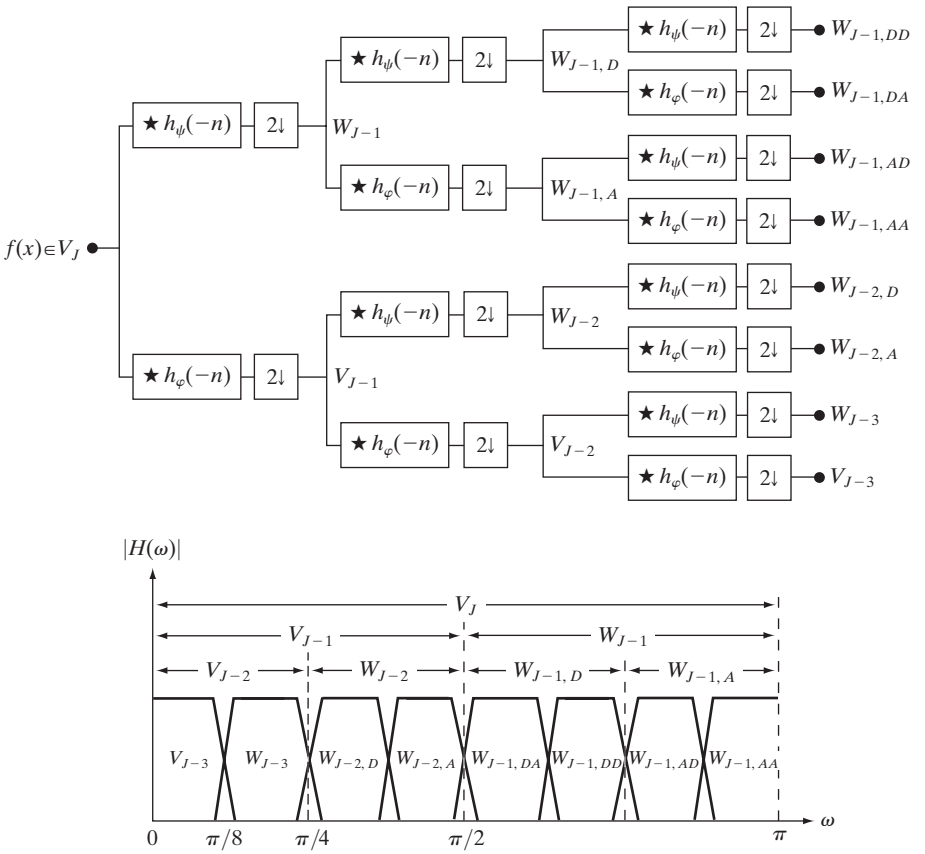


FIGURE 7.31
A three-scale wavelet packet analysis tree.



a
b

FIGURE 7.32 The (a) filter bank and (b) spectrum splitting characteristics of a three-scale full wavelet packet analysis tree.

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

Recall that $\oplus$ denotes the union of spaces (like the union of sets). The 26 decompositions associated with Fig. 7.31 are determined by various combinations of nodes (spaces) that can be combined to represent the root node (space) at the top of the tree. Eqs. (7.6-4) and (7.6-5) define two of them.

whose spectrum is shown in Fig. 7.32(b), or

$$V_J = V_{J-1} \oplus W_{J-1,A} \oplus W_{J-1,DA} \oplus W_{J-1,DD} \quad (7.6\text{-}5)$$

whose spectrum is depicted in Fig. 7.33. Note the difference between this last spectrum and the full packet spectrum of Fig. 7.32(b), or the three-scale FWT

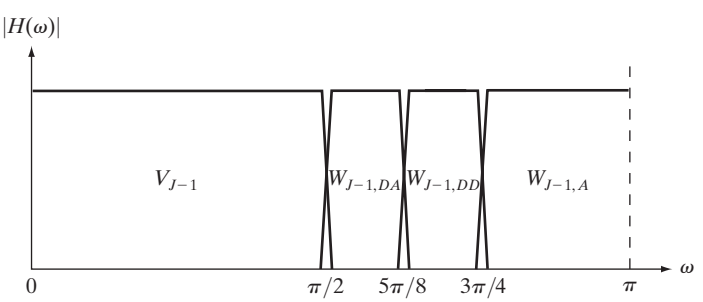


FIGURE 7.33 The spectrum of the decomposition in Eq. (7.6-5).

spectrum of Fig. 7.30(c). In general, P-scale, one-dimensional wavelet packet transforms (and associated $P + 1$-level analysis trees) support

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

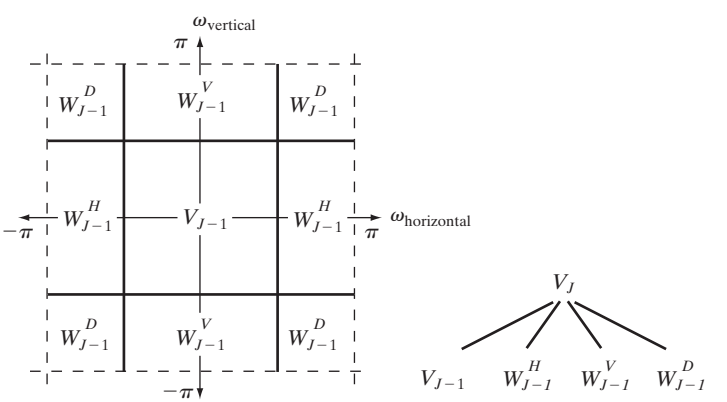
unique decompositions, where $D(1) = 1$. With such a large number of valid expansions, packet-based transforms provide improved control over partitioning the spectrum of the decomposed function. The cost of this control is an increase in computational complexity [compare the filter bank in Fig. 7.30(a) to that of Fig. 7.32(a)].

Now consider the two-dimensional, four-band filter bank of Fig. 7.24(a). As was noted in Section 7.5, it splits approximation $W_\varphi(j + 1, m, n)$ into outputs, $W_\varphi(j, m, n)$, $W_\psi^H(j, m, n)$, $W_\psi^V(j, m, n)$, and $W_\psi^D(j, m, n)$. 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. 7.24(a)] is shown in Fig. 7.34(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_{J-1}^H, W_{J-1}^V, and W_{J-1}^D and correspond to coefficients $W_\psi^H(J - 1, m, n)$, $W_\psi^V(J - 1, m, n)$, and $W_\psi^D(J - 1, m, n)$, respectively. Figure 7.34(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 7.35 shows a portion of a three-scale, two-dimensional wavelet packet analysis tree. Like its one-dimensional counterpart in Fig. 7.31, the first subscript of every node that is a descendant of a conventional FWT detail node is the scale of that *parent* detail node. The second subscript—a variable length string of *A*s, *H*s, *V*s, and *D*s—encodes the path from the parent to the node under consideration. The node labeled $W_{J-1,VD}^H$, for example, is obtained by "row/column filtering" the

a b

FIGURE 7.34 The first decomposition of a two-dimensional FWT: (a) the spectrum and (b) the subspace analysis tree.



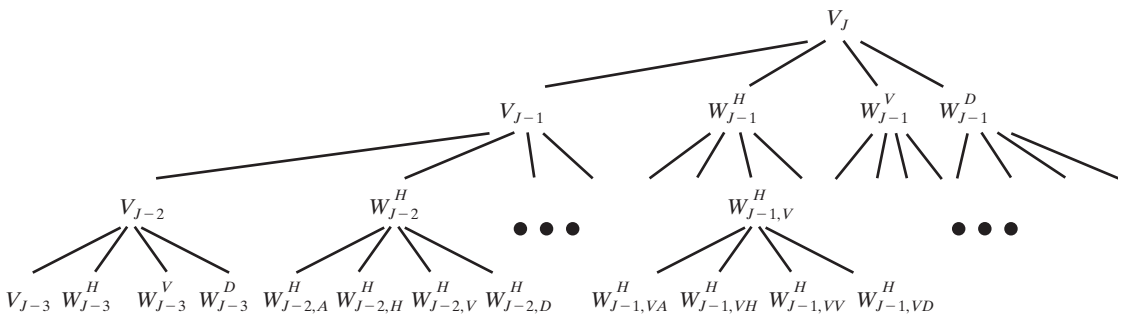


FIGURE 7.35 A three-scale, full wavelet packet decomposition tree. Only a portion of the tree is provided.

scale $J - 1$ FWT horizontal detail coefficients (i.e., parent W^H_{J-1} in Fig. 7.35) 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

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

unique expansions, where $D(1) = 1$. Thus, the three-scale tree of Fig. 7.35 offers 83,522 possible decompositions. The problem of selecting among them is the subject of the next example.

EXAMPLE 7.15: Two-dimensional wavelet packet decompositions.

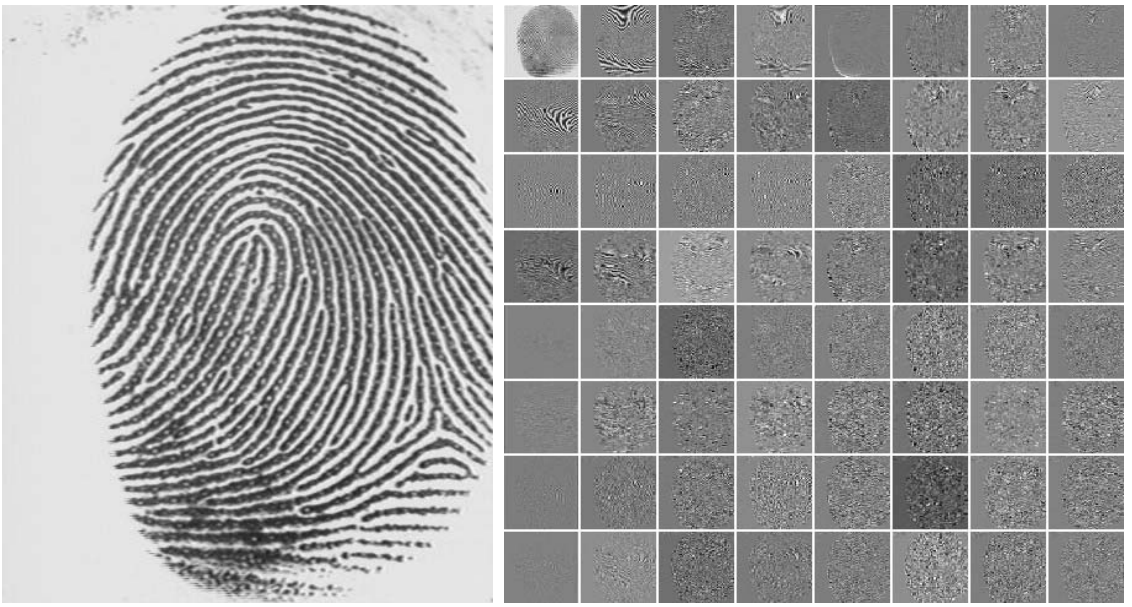
■ As noted in the above discussion, a single wavelet packet tree presents numerous decomposition options. In fact, the number of possible decompositions is often so large that it is impractical, if not impossible, to enumerate or examine them individually. An efficient algorithm for finding optimal decompositions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

Consider the problem of reducing the amount of data needed to represent the 400×480 fingerprint image in Fig. 7.36(a). Image compression is discussed in detail in Chapter 8. In this example, we want to select the "best" three-scale wavelet packet decomposition as a starting point for the compression process. Using three-scale wavelet packet trees, there are 83,522 [see Eq. (7.6-7)] potential decompositions. Figure 7.36(b) shows one of them—a full wavelet packet, 64-leaf decomposition like the analysis tree of Fig. 7.35. Note that the leaves of the tree correspond to the subbands of the 8×8 array of decomposed subimages in Fig. 7.36(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.

The 64 leaf nodes in Fig. 7.35 correspond to the 8×8 array of 64 subimages in Fig. 7.36(b). Despite appearances, they are not square. The distortion (particularly noticeable in the approximation subimage) is due to the program used to produce the result.

One reasonable criterion for selecting a decomposition for the compression of the image of Fig. 7.36(a) is the additive cost function

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



a b

FIGURE 7.36 (a) A scanned fingerprint and (b) its three-scale, full wavelet packet decomposition. (Original image courtesy of the National Institute of Standards and Technology.)

Other possible energy measures include the sum of the squares of $f(x, y)$, the sum of the log of the squares, etc. Problem 7.27 defines one possible entropy-based cost function.

This function provides one possible measure of the energy content of two-dimensional function f. Under this measure, the energy of function $f(m, n) = 0$ for all m and n is 0. High values of E, 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 "good" decomposition from a compression point of view.

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 leaf nodes in the decomposition tree. Minimal energy 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 energy solutions is easily constructed as follows:

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

Step 1. Compute both the energy of the node, denoted E_P (for parent energy), and the energy 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.

Step 2. If the combined energy of the offspring is less than the energy 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 energy 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 7.37 shows the optimized decomposition that results from applying the algorithm to the image of Fig. 7.36(a) with the cost function of Eq. (7.6-8). The corresponding analysis tree is given in Fig. 7.38. Note that many of the original full packet decomposition's 64 subbands in Fig. 7.36(b) (and corresponding 64 leaves of the analysis tree in Fig. 7.35) have been eliminated. In addition, the subimages that are not split (further decomposed) in Fig. 7.37 are relatively smooth and composed of pixels that are middle gray in value. Because 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 energy. There would be no overall decrease in energy 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

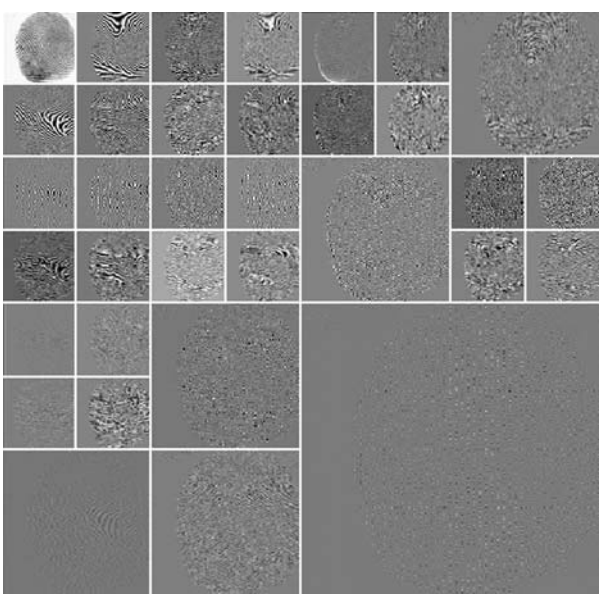


FIGURE 7.37 An optimal wavelet packet decomposition for the fingerprint of Fig. 7.36(a).

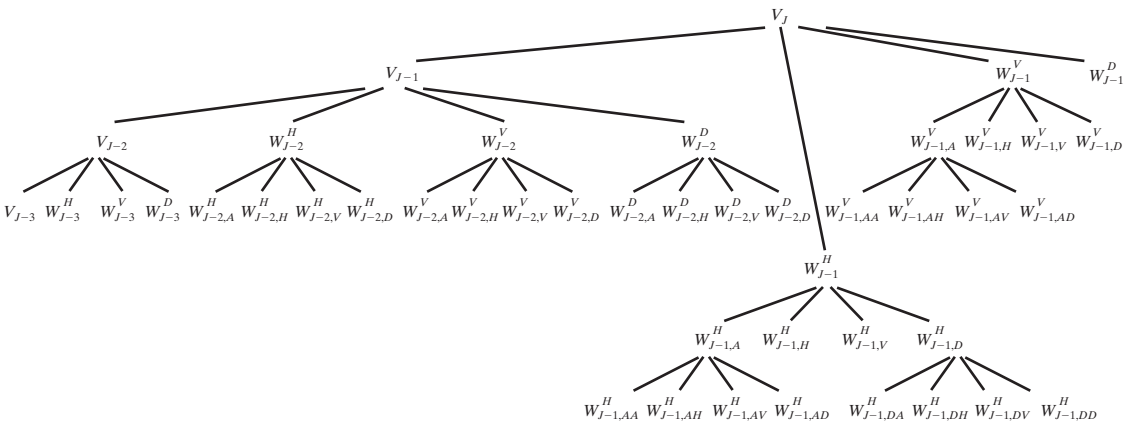


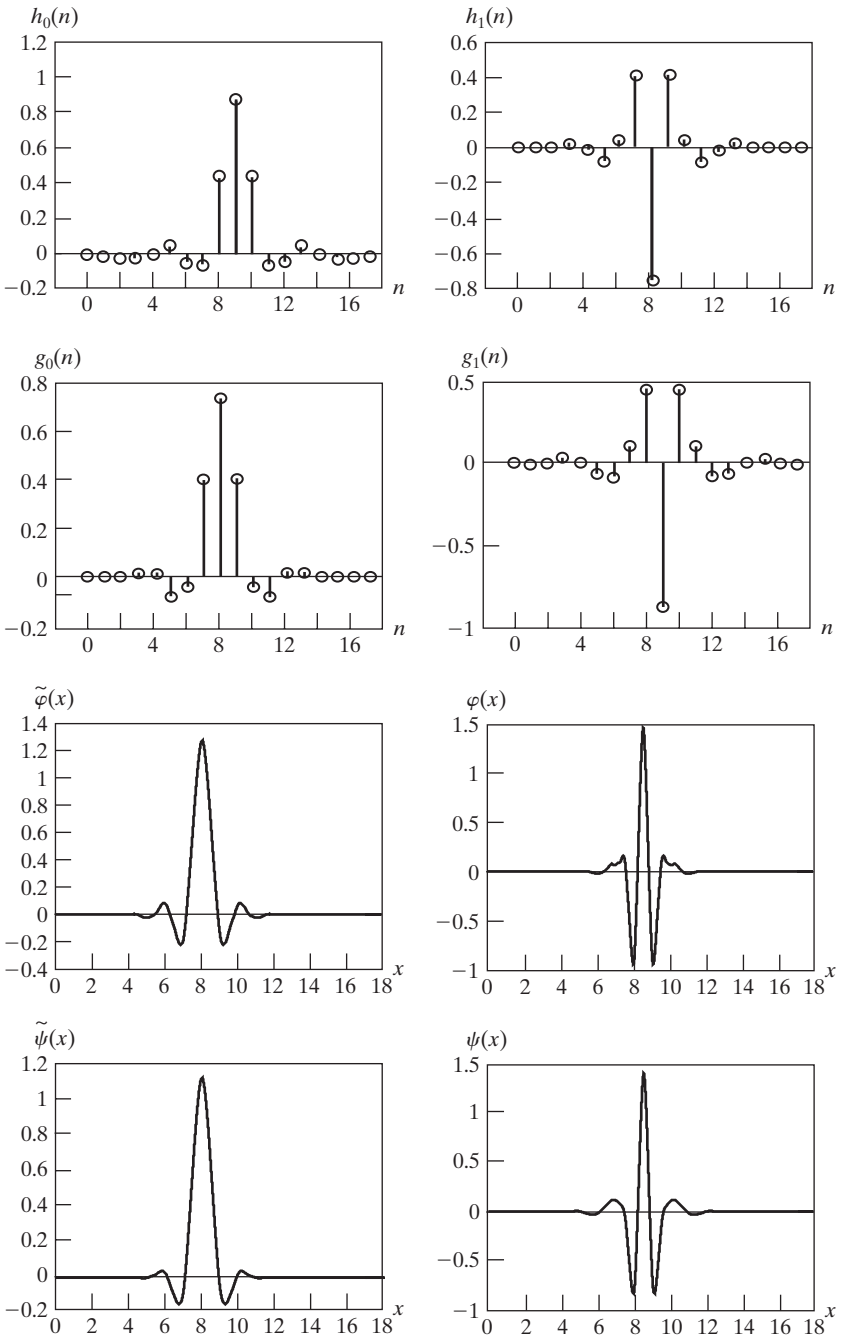
FIGURE 7.38 The optimal wavelet packet analysis tree for the decomposition in Fig. 7.37.

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 scaling and wavelet functions of the family are symmetrical and have similar lengths, they are among the most widely used biorthogonal wavelets. Figures 7.39(e) through (h) show the dual scaling and wavelet functions. Figures 7.39(a) through (d) are the corresponding decomposition and reconstruction filters. The coefficients of the lowpass and highpass decomposition filters, $h_0(n)$ and $h_1(n)$ for $0 \le n \le 17$ are shown in Table 7.4. 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. (7.1-11). That is, they are cross-modulated versions of the decomposition filters. Note that zero padding is employed to make the filters the same length and that Table 7.4 and Fig. 7.39 define them with respect to the subband coding and decoding system of Fig. 7.6(a); with respect to the FWT, $h_\varphi(-n) = h_0(n)$ and $h_\psi(-n) = h_1(n)$.

TABLE 7.4 Biorthogonal Cohen-Daubechies-Feauveau filter coefficients (Cohen, Daubechies, and Feauveau [1992]).

n	$h_0(n)$	$h_1(n)$	n	$h_0(n)$	$h_1(n)$
0	0	0	9	0.8259	0.4178
1	0.0019	0	10	0.4208	0.0404
2	−0.0019	0	11	−0.0941	−0.0787
3	−0.017	0.0144	12	−0.0773	−0.0145
4	0.0119	−0.0145	13	0.0497	0.0144
5	0.0497	−0.0787	14	0.0119	0
6	−0.0773	0.0404	15	−0.017	0
7	−0.0941	0.4178	16	−0.0019	0
8	0.4208	−0.7589	17	0.0010	0



a b
c d
e f
g h

FIGURE 7.39 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. See Table 7.3 for the values of $h_0(n)$ and $h_1(n)$ for $0 \leq n \leq 17$.

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 upon 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 references cited in the text. All of the examples in the chapter were done using MATLAB (see Gonzalez et al. [2004]).

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, 1992b] [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 Gao et al. [2007] on corner detection; Olkkonen and Olkkonen [2007] on lattice implementations; Selesnick et al. [2005] and Kokare et al. [2005] on complex wavelets; 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, Brechet et al. [2007], Demin Wang et al. [2006], 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*