6 Color Image Processing

It is only after years of preparation that the young artist should touch color—not color used descriptively, that is, but as a means of personal expression.

Henri Matisse

For a long time I limited myself to one color—as a form of discipline.

Pablo Picasso

Preview

The use of color in image processing is motivated by two principal factors. First, color is a powerful descriptor that often simplifies object identification and extraction from a scene. Second, humans can discern thousands of color shades and intensities, compared to about only two dozen shades of gray. This second factor is particularly important in manual (i.e., when performed by humans) image analysis.

Color image processing is divided into two major areas: *full-color* and *pseudocolor* processing. In the first category, the images in question typically are acquired with a full-color sensor, such as a color TV camera or color scanner. In the second category, the problem is one of assigning a color to a particular monochrome intensity or range of intensities. Until recently, most digital color image processing was done at the pseudocolor level. However, in the past decade, color sensors and hardware for processing color images have become available at reasonable prices. The result is that full-color image processing techniques are now used in a broad range of applications, including publishing, visualization, and the Internet.

It will become evident in the discussions that follow that some of the gray-scale methods covered in previous chapters are directly applicable to color images. Others require reformulation to be consistent with the properties of the color spaces developed in this chapter. The techniques described here are

far from exhaustive; they illustrate the range of methods available for color image processing.

6.1 Color Fundamentals

Although the process followed by the human brain in perceiving and interpreting color is a physiopsychological phenomenon that is not yet fully understood, the physical nature of color can be expressed on a formal basis supported by experimental and theoretical results.

In 1666, Sir Isaac Newton discovered that when a beam of sunlight passes through a glass prism, the emerging beam of light is not white but consists instead of a continuous spectrum of colors ranging from violet at one end to red at the other. As Fig. 6.1 shows, the color spectrum may be divided into six broad regions: violet, blue, green, yellow, orange, and red. When viewed in full color (Fig. 6.2), no color in the spectrum ends abruptly, but rather each color blends smoothly into the next.

Basically, the colors that humans and some other animals perceive in an object are determined by the nature of the light reflected from the object. As illustrated in Fig. 6.2, visible light is composed of a relatively narrow band of frequencies in the electromagnetic spectrum. A body that reflects light that is balanced in all visible wavelengths appears white to the observer. However, a body that favors reflectance in a limited range of the visible spectrum exhibits some shades of color. For example, green objects reflect light with wavelengths primarily in the 500 to 570 nm range while absorbing most of the energy at other wavelengths.

Characterization of light is central to the science of color. If the light is achromatic (void of color), its only attribute is its *intensity*, or amount. Achromatic light is what viewers see on a black and white television set, and it has been an implicit component of our discussion of image processing thus far. As defined in Chapter 2, and used numerous times since, the term *gray level* refers to a scalar measure of intensity that ranges from black, to grays, and finally to white.

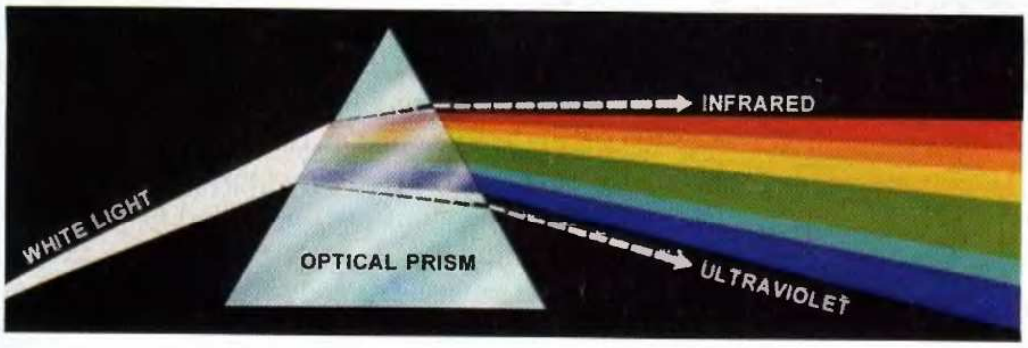


FIGURE 6.1 Color spectrum seen by passing white light through a prism. (Courtesy of the General Electric Co., Lamp Business Division.)

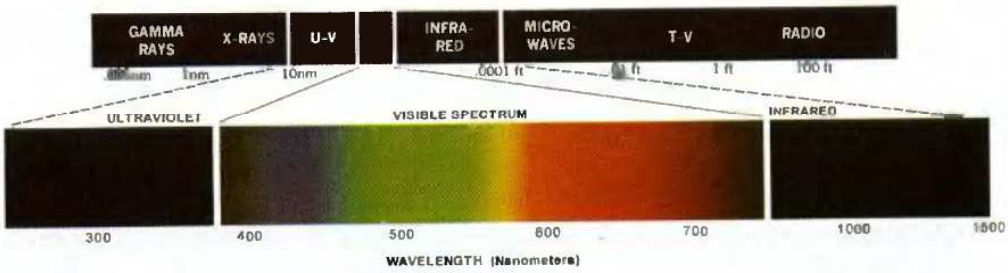


FIGURE 6.2 Wavelengths comprising the visible range of the electromagnetic spectrum. (Courtesy of the General Electric Co., Lamp Business Division.)

Chromatic light spans the electromagnetic spectrum from approximately 400 to 700 nm. Three basic quantities are used to describe the quality of a chromatic light source: radiance, luminance, and brightness. *Radiance* is the total amount of energy that flows from the light source, and it is usually measured in watts (W). *Luminance*, measured in lumens (lm), gives a measure of the amount of energy an observer *perceives* from a light source. For example, light emitted from a source operating in the far infrared region of the spectrum could have significant energy (radiance), but an observer would hardly perceive it; its luminance would be almost zero. Finally, *brightness* is a subjective descriptor that is practically impossible to measure. It embodies the achromatic notion of intensity and is one of the key factors in describing color sensation.

As noted in Section 2.1.1, cones are the sensors in the eye responsible for color vision. Detailed experimental evidence has established that the 6 to 7 million cones in the human eye can be divided into three principal sensing categories, corresponding roughly to red, green, and blue. Approximately 65% of all cones are sensitive to red light, 33% are sensitive to green light, and only about 2% are sensitive to blue (but the blue cones are the most sensitive). Figure 6.3 shows average experimental curves detailing the absorption of light by the red, green, and blue cones in the eye. Due to these absorption characteristics of the human eye, colors are seen as variable combinations of the so-called *primary colors* red (R), green (G), and blue (B). For the purpose of standardization, the CIE (Commission Internationale de l'Eclairage—the International Commission on Illumination) designated in 1931 the following specific wavelength values to the three primary colors: blue = 435.8 nm, green = 546.1 nm, and red = 700 nm. This standard was set before the detailed experimental curves shown Fig. 6.3 became available in 1965. Thus, the CIE standards correspond only approximately with experimental data. We note from Figs. 6.2 and 6.3 that no single color may be called red, green, or blue. Also, it is important to keep in mind that having three specific primary color wavelengths for the purpose of standardization does not mean that these three fixed RGB components acting alone can generate all spectrum colors. Use of the word *primary* has been widely misinterpreted to mean that the three standard primaries, when mixed in various intensity proportions, can produce *all* visible colors. As we will see shortly, this interpretation is not

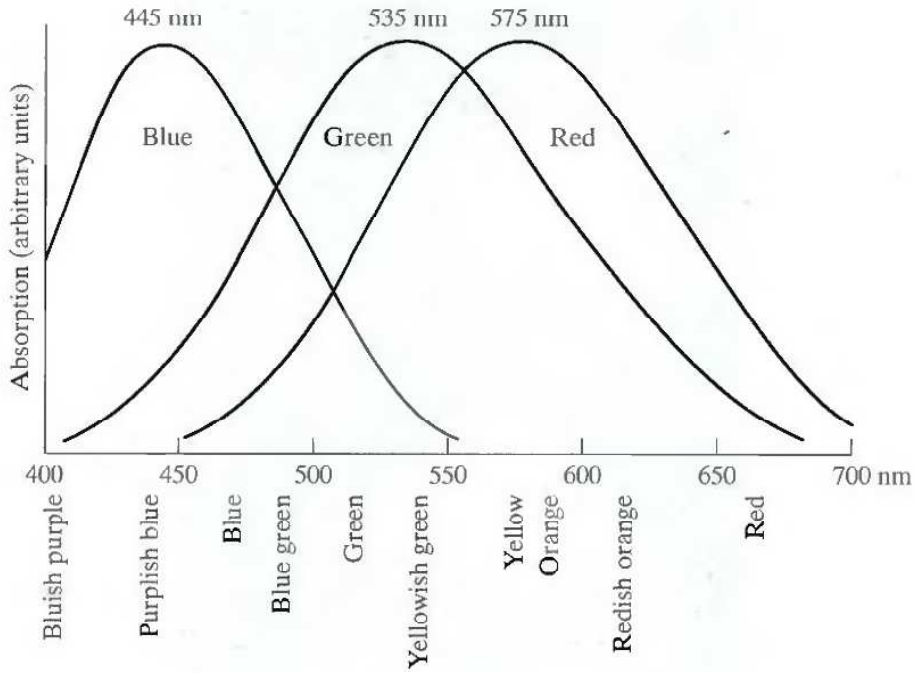


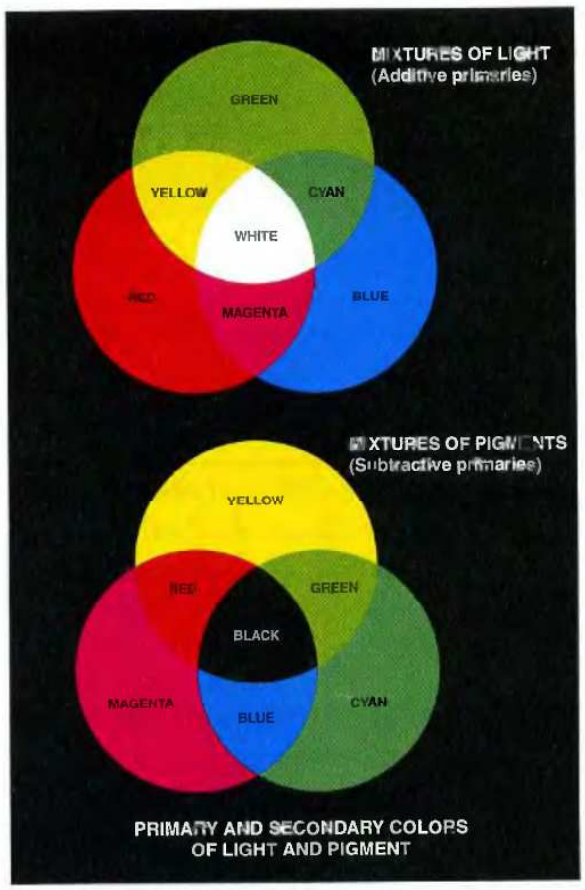
FIGURE 6.3 Absorption of light by the red, green, and blue cones in the human eye as a function of wavelength.

correct unless the wavelength also is allowed to vary, in which case we would no longer have three fixed, standard primary colors.

The primary colors can be added to produce the *secondary* colors of light—magenta (red plus blue), cyan (green plus blue), and yellow (red plus green). Mixing the three primaries, or a secondary with its opposite primary color, in the right intensities produces white light. This result is shown in Fig. 6.4(a), which also illustrates the three primary colors and their combinations to produce the secondary colors.

Differentiating between the primary colors of light and the primary colors of pigments or colorants is important. In the latter, a primary color is defined as one that subtracts or absorbs a primary color of light and reflects or transmits the other two. Therefore, the primary colors of pigments are magenta, cyan, and yellow, and the secondary colors are red, green, and blue. These colors are shown in Fig. 6.4(b). A proper combination of the three pigment primaries, or a secondary with its opposite primary, produces black.

Color television reception is an example of the additive nature of light colors. The interior of many color TV tubes is composed of a large array of triangular dot patterns of electron-sensitive phosphor. When excited, each dot in a triad is capable of producing light in one of the primary colors. The intensity of the red-emitting phosphor dots is modulated by an electron gun inside the tube, which generates pulses corresponding to the "red energy" seen by the TV camera. The green and blue phosphor dots in each triad are modulated in the same manner. The effect, viewed on the television receiver, is that the three primary colors from each phosphor triad are "added" together and received by



a
b

FIGURE 6.4 Primary and secondary colors of light and pigments. (Courtesy of the General Electric Co., Lamp Business Division.)

the color-sensitive cones in the eye as a full-color image. Thirty successive image changes per second in all three colors complete the illusion of a continuous image display on the screen.

The characteristics generally used to distinguish one color from another are *brightness, hue*, and *saturation*. As indicated earlier in this section, brightness embodies the chromatic notion of intensity. Hue is an attribute associated with the dominant wavelength in a mixture of light waves. Hue represents dominant color as perceived by an observer. Thus, when we call an object red, orange, or yellow, we are specifying its hue. Saturation refers to the relative purity or the amount of white light mixed with a hue. The pure spectrum colors are fully saturated. Colors such as pink (red and white) and lavender (violet and white) are less saturated, with the degree of saturation being inversely proportional to the amount of white light added.

Hue and saturation taken together are called *chromaticity*, and, therefore, a color may be characterized by its brightness and chromaticity. The amounts of red, green, and blue needed to form any particular color are called the *tristimulus* values and are denoted, X, Y, and Z, respectively. A color is then specified

by its *trichromatic coefficients*, defined as

$$x = \frac{X}{X + Y + Z} \tag{6.1-1}$$

$$y = \frac{Y}{X + Y + Z} \tag{6.1-2}$$

and

$$z = \frac{Z}{X + Y + Z}. \tag{6.1-3}$$

It is noted from these equations that†

$$x + y + z = 1. \tag{6.1-4}$$

For any wavelength of light in the visible spectrum, the tristimulus values needed to produce the color corresponding to that wavelength can be obtained directly from curves or tables that have been compiled from extensive experimental results (Poynton [1996]. See also the early references by Walsh [1958] and by Kiver [1965]).

Another approach for specifying colors is to use the CIE *chromaticity diagram* (Fig. 6.5), which shows color composition as a function of x (red) and y (green). For any value of x and y, the corresponding value of z (blue) is obtained from Eq. (6.1-4) by noting that $z = 1 - (x + y)$. The point marked green in Fig. 6.5, for example, has approximately 62% green and 25% red content. From Eq. (6.1-4), the composition of blue is approximately 13%.

The positions of the various spectrum colors—from violet at 380 nm to red at 780 nm—are indicated around the boundary of the tongue-shaped chromaticity diagram. These are the pure colors shown in the spectrum of Fig. 6.2. Any point not actually on the boundary but within the diagram represents some mixture of spectrum colors. The point of equal energy shown in Fig. 6.5 corresponds to equal fractions of the three primary colors; it represents the CIE standard for white light. Any point located on the boundary of the chromaticity chart is fully saturated. As a point leaves the boundary and approaches the point of equal energy, more white light is added to the color and it becomes less saturated. The saturation at the point of equal energy is zero.

The chromaticity diagram is useful for color mixing because a straight-line segment joining any two points in the diagram defines all the different color variations that can be obtained by combining these two colors additively. Consider, for example, a straight line drawn from the red to the green points shown in Fig. 6.5. If there is more red light than green light, the exact point representing the new color will be on the line segment, but it will be closer to the red point than to the green point. Similarly, a line drawn from the point of equal energy to any point on the boundary of the chart will define all the shades of that particular spectrum color.

†The use x, y, z in this context follows notational convention. These should not be confused with the use of (x, y) to denote spatial coordinates in other sections of the book.

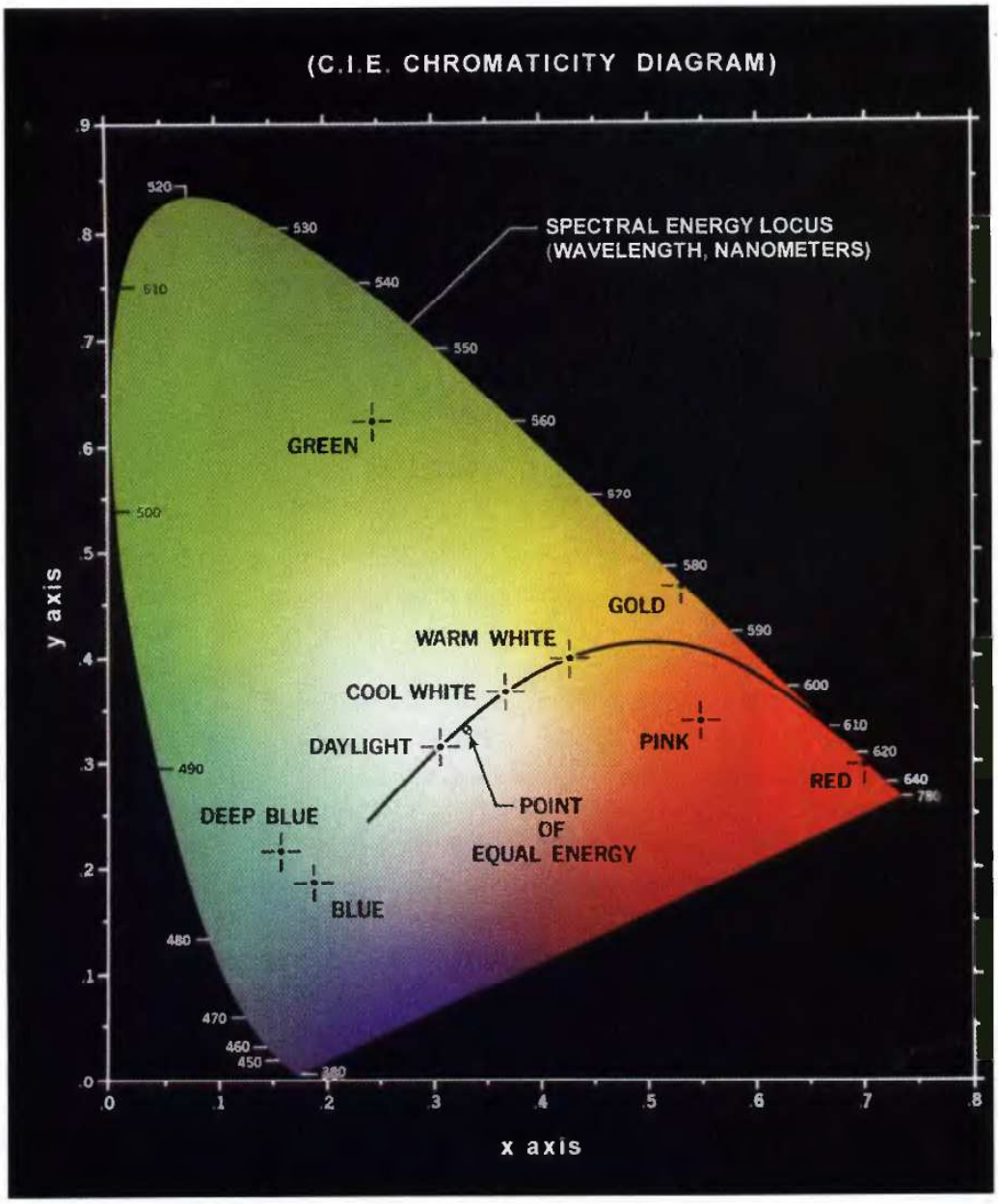


FIGURE 6.5 Chromaticity diagram. (Courtesy of the General Electric Co., Lamp Business Division.)

Extension of this procedure to three colors is straightforward. To determine the range of colors that can be obtained from any three given colors in the chromaticity diagram, we simply draw connecting lines to each of the three color points. The result is a triangle, and any color inside the triangle can be produced by various combinations of the three initial colors. A triangle with vertices at any three *fixed* colors cannot enclose the entire color region in Fig. 6.5. This observation supports graphically the remark made earlier that not all colors can be obtained with three single, fixed primaries.

The triangle in Figure 6.6 shows a typical range of colors (called the *color gamut*) produced by RGB monitors. The irregular region inside the triangle is representative of the color gamut of today's high-quality color printing devices.

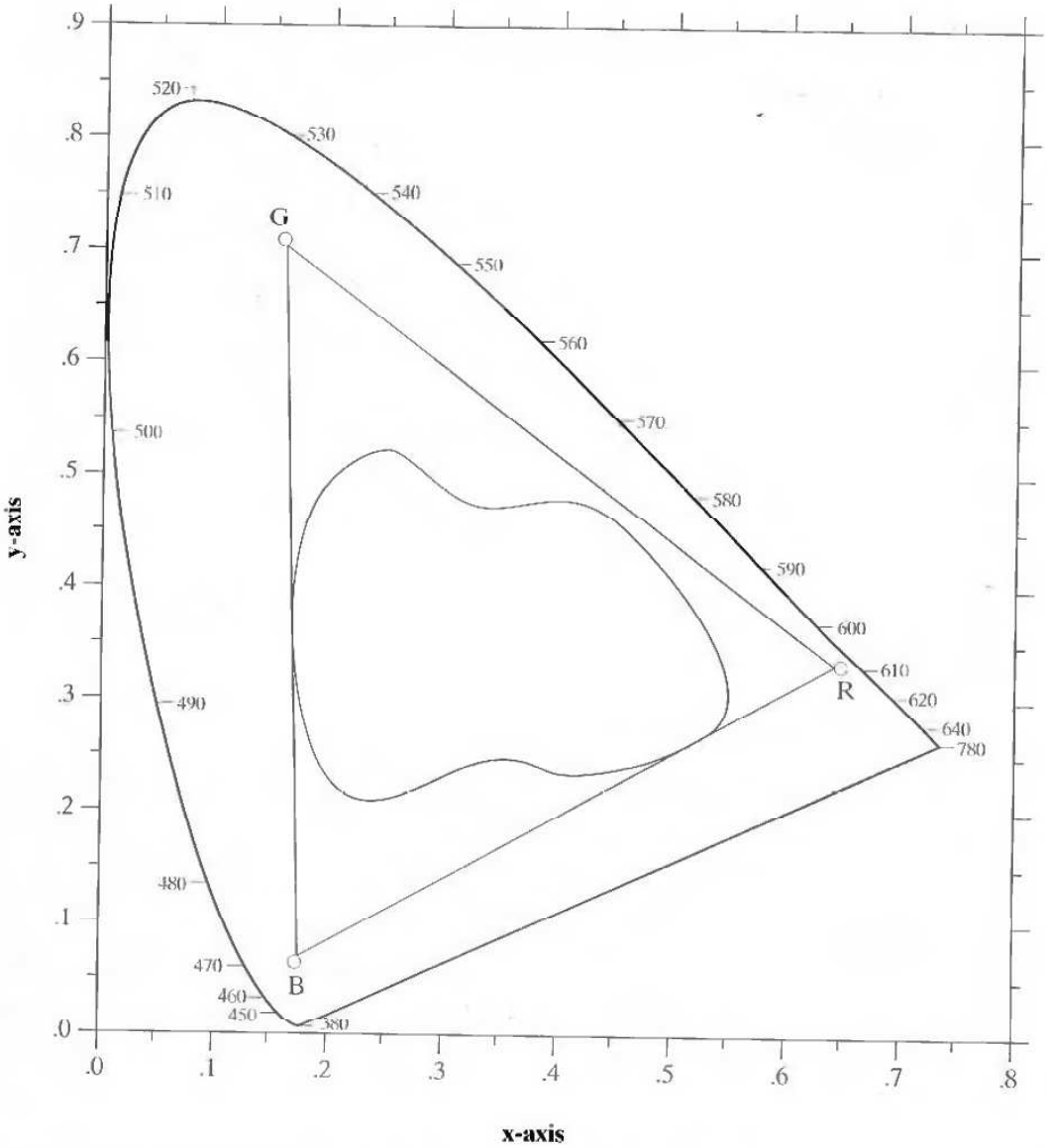


FIGURE 6.6 Typical color gamut of color monitors (triangle) and color printing devices (irregular region).

The boundary of the color printing gamut is irregular because color printing is a combination of additive and subtractive color mixing, a process that is much more difficult to control than that of displaying colors on a monitor, which is based on the addition of three highly controllable light primaries.

6.2 Color Models

The purpose of a color model (also called *color space* or *color system*) is to facilitate the specification of colors in some standard, generally accepted way. In essence, a color model is a specification of a coordinate system and a subspace within that system where each color is represented by a single point.

Most color models in use today are oriented either toward hardware (such as for color monitors and printers) or toward applications where color manipulation is a goal (such as in the creation of color graphics for animation). In terms of digital image processing, the hardware-oriented models most commonly used in practice are the RGB (red, green, blue) model for color monitors and a broad class of color video cameras; the CMY (cyan, magenta, yellow) and CMYK (cyan, magenta, yellow, black) models for color printing; and the HSI (hue, saturation, intensity) model, which corresponds closely with the way humans describe and interpret color. The HSI model also has the advantage that it decouples the color and gray-scale information in an image, making it suitable for many of the gray-scale techniques developed in this book. There are numerous color models in use today due to the fact that color science is a broad field that encompasses many areas of application. It is tempting to dwell on some of these models here simply because they are interesting and informative. However, keeping to the task at hand, the models discussed in this chapter are leading models for image processing. Having mastered the material in this chapter, the reader will have no difficulty in understanding additional color models in use today.

6.2.1 The RGB Color Model

In the RGB model, each color appears in its primary spectral components of red, green, and blue. This model is based on a Cartesian coordinate system. The color subspace of interest is the cube shown in Fig. 6.7, in which RGB values are at three corners; cyan, magenta, and yellow are at three other corners; black is at the origin; and white is at the corner farthest from the origin. In this model, the gray scale (points of equal RGB values) extends from black to white along the line joining these two points. The different colors in this model are points on or inside the cube, and are defined by vectors extending from the origin. For

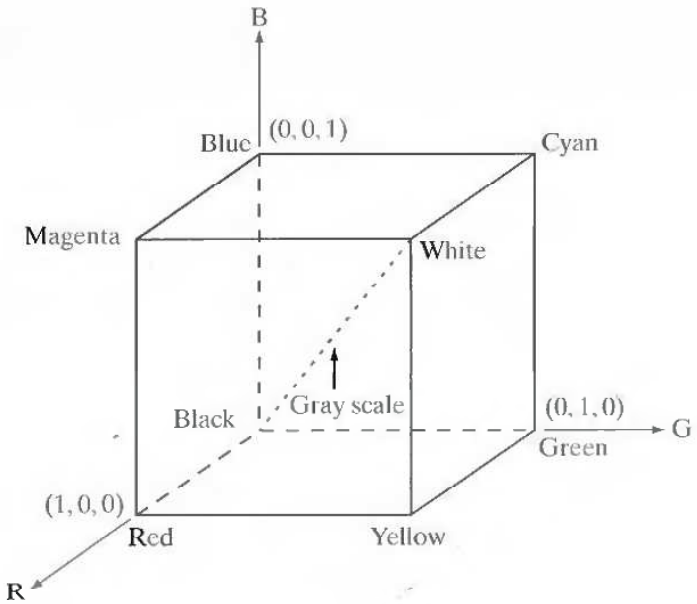


FIGURE 6.7 Schematic of the RGB color cube. Points along the main diagonal have gray values, from black at the origin to white at point (1, 1, 1).

convenience, the assumption is that all color values have been normalized so that the cube shown in Fig. 6.7 is the unit cube. That is, all values of R, G, and B are assumed to be in the range $[0, 1]$.

Images represented in the RGB color model consist of three component images, one for each primary color. When fed into an RGB monitor, these three images combine on the phosphor screen to produce a composite color image. The number of bits used to represent each pixel in RGB space is called the *pixel depth*. Consider an RGB image in which each of the red, green, and blue images is an 8-bit image. Under these conditions each RGB *color* pixel [that is, a triplet of values (R, G, B)] is said to have a depth of 24 bits (3 image planes times the number of bits per plane). The term *full-color* image is used often to denote a 24-bit RGB color image. The total number of colors in a 24-bit RGB image is $(2^8)^3 = 16{,}777{,}216$. Figure 6.8 shows the 24-bit RGB color cube corresponding to the diagram in Fig. 6.7.

EXAMPLE 6.1: Generating the hidden face planes and a cross section of the RGB color cube.

■ The cube shown in Fig. 6.8 is a solid, composed of the $(2^8)^3 = 16{,}777{,}216$ colors mentioned in the preceding paragraph. A convenient way to view these colors is to generate color planes (faces or cross sections of the cube). This is accomplished simply by fixing one of the three colors and allowing the other two to vary. For instance, a cross-sectional plane through the center of the cube and parallel to the GB-plane in Figs. 6.8 and 6.7 is the plane $(127, G, B)$ for $G, B = 0, 1, 2, \ldots, 255$. Here we used the actual pixel values rather than the mathematically convenient normalized values in the range $[0, 1]$ because the former values are the ones actually used in a computer to generate colors. Figure 6.9(a) shows that an image of the cross-sectional plane is viewed simply by feeding the three individual component images into a color monitor. In the component images, 0 represents black and 255 represents white (note that these are gray-scale images). Finally, Fig. 6.9(b) shows the three hidden surface planes of the cube in Fig. 6.8, generated in the same manner.

It is of interest to note that *acquiring* a color image is basically the process shown in Fig. 6.9 in reverse. A color image can be acquired by using three filters, sensitive to red, green, and blue, respectively. When we view a color scene with a monochrome camera equipped with one of these filters, the result is a monochrome image whose intensity is proportional to the response of that filter.

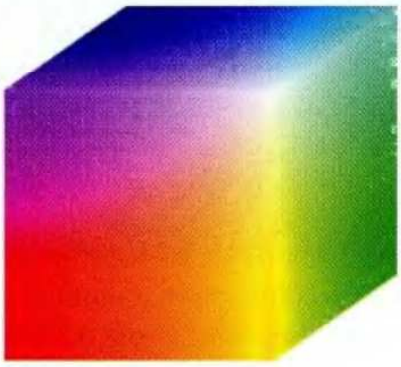
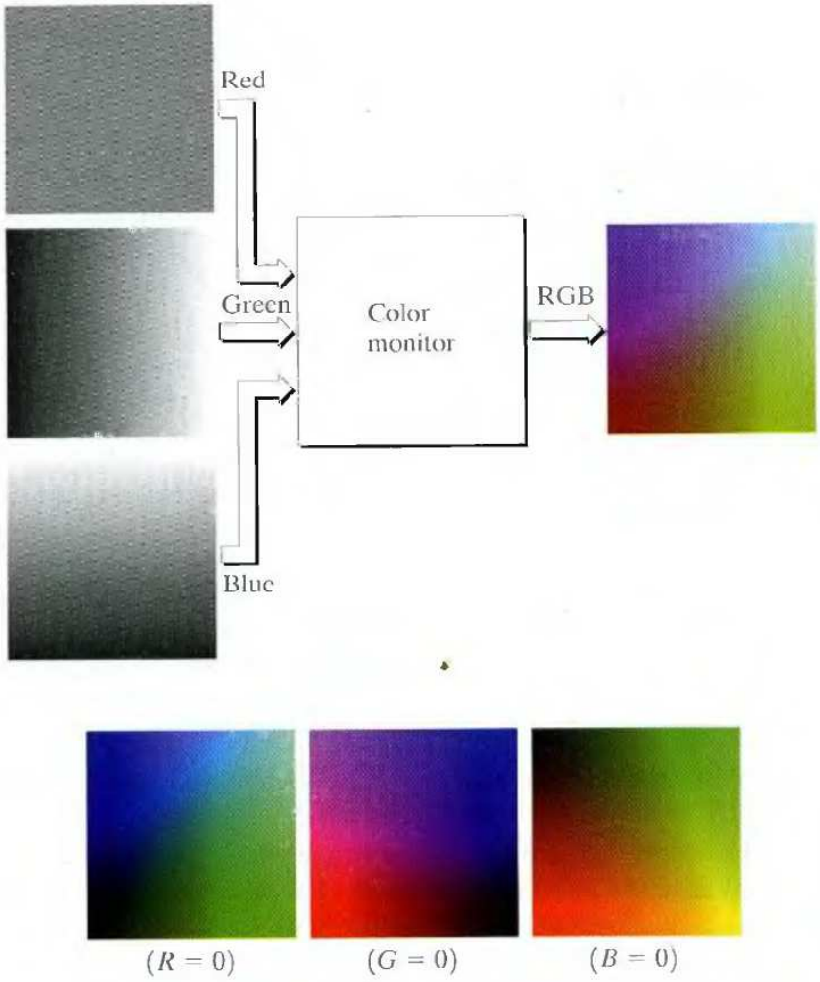


FIGURE 6.8 RGB 24-bit color cube.

a
b

FIGURE 6.9
(a) Generating the RGB image of the cross-sectional color plane (127, G, B).
(b) The three hidden surface planes in the color cube of Fig. 6.8.



Repeating this process with each filter produces three monochrome images that are the RGB component images of the color scene. (In practice, RGB color image sensors usually integrate this process into a single device.) Clearly, displaying these three RGB component images in the form shown in Fig. 6.9(a) would yield an RGB color rendition of the original color scene. ■

While high-end display cards and monitors provide a reasonable rendition of the colors in a 24-bit RGB image, many systems in use today are limited to 256 colors. Also, there are numerous applications in which it simply makes no sense to use more than a few hundred, and sometimes fewer, colors. A good example of this is provided by the pseudocolor image processing techniques discussed in Section 6.3. Given the variety of systems in current use, it is of considerable interest to have a subset of colors that are likely to be reproduced faithfully, reasonably independently of viewer hardware capabilities. This subset of colors is called the set of *safe RGB colors*, or the set of *all-systems-safe colors*. In Internet applications, they are called *safe Web colors* or *safe browser colors*.

On the assumption that 256 colors is the minimum number of colors that can be reproduced faithfully by any system in which a desired result is likely to be displayed, it is useful to have an accepted standard notation to refer to these colors. Forty of these 256 colors are known to be processed differently by various operating systems, leaving only 216 colors that are common to most systems.

Number System	Color Equivalents					
Hex	00	33	66	99	CC	FF
Decimal	0	51	102	153	204	255

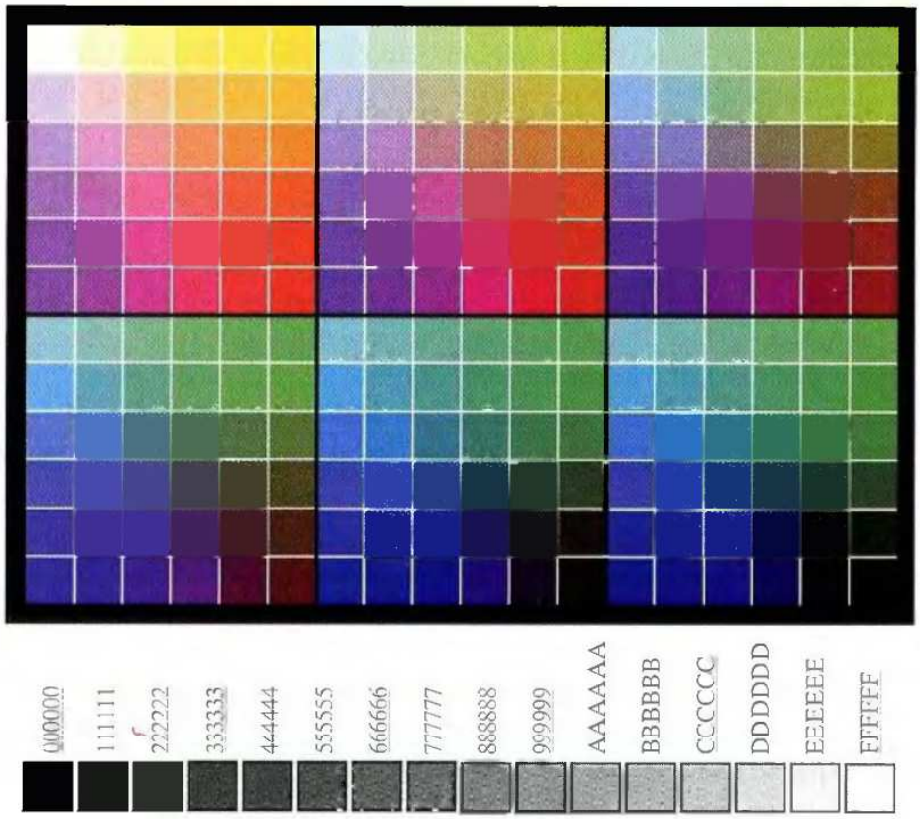
TABLE 6.1 Valid values of each RGB component in a safe color.

These 216 colors have become the de facto standard for safe colors, especially in Internet applications. They are used whenever it is desired that the colors viewed by most people appear the same.

Each of the 216 safe colors is formed from three RGB values as before, but each value can only be 0, 51, 102, 153, 204, or 255. Thus, RGB triplets of these values give us $(6)^3 = 216$ possible values (note that all values are divisible by 3). It is customary to express these values in the hexagonal number system, as shown in Table 6.1. Recall that hex numbers 0, 1, 2, ..., 9, A, B, C, D, E, F correspond to decimal numbers 0, 1, 2, ..., 9, 10, 11, 12, 13, 14, 15. Recall also that $(0)_{16} = (0000)_2$ and $(F)_{16} = (1111)_2$. Thus, for example, $(FF)_{16} = (255)_{10} = (11111111)_2$ and we see that a grouping of two hex numbers forms an 8-bit byte.

Since it takes three numbers to form an RGB color, each safe color is formed from three of the two digit hex numbers in Table 6.1. For example, the purest red is FF0000. The values 000000 and FFFFFF represent black and white, respectively. Keep in mind that the same result is obtained by using the more familiar decimal notation. For instance, the brightest red in decimal notation has $R = 255$ (FF) and $G = B = 0$.

Figure 6.10(a) shows the 216 safe colors, organized in descending RGB values. The square in the top left array has value FFFFFF (white), the second



a
b

FIGURE 6.10 (a) The 216 safe RGB colors. (b) All the grays in the 256-color RGB system (grays that are part of the safe color group are shown underlined).

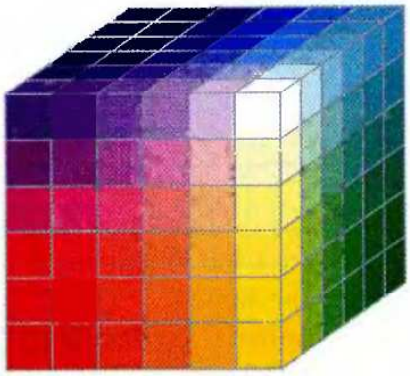


FIGURE 6.11 The RGB safe-color cube.

square to its right has value FFFFCC, the third square has value FFFF99, and so on for the first row. The second row of that same array has values FFCCFF, FFCCCC, FFCC99, and so on. The final square of that array has value FF0000 (the brightest possible red). The second array to the right of the one just examined starts with value CCFFFF and proceeds in the same manner, as do the other remaining four arrays. The final (bottom right) square of the last array has value 000000 (black). It is important to note that not all possible 8-bit gray colors are included in the 216 safe colors. Figure 6.10(b) shows the hex codes for *all* the possible gray colors in a 256-color RGB system. Some of these values are outside of the safe color set but are represented properly (in terms on their relative intensities) by most display systems. The grays from the safe color group, $(KKKKKK)_{16}$, for $K = 0, 3, 6, 9, C, F$, are shown underlined in Fig. 6.10(b).

Figure 6.11 shows the RGB safe-color cube. Unlike the full-color cube in Fig. 6.8, which is solid, the cube in Fig. 6.11 has valid colors only on the surface planes. As shown in Fig. 6.10(a), each plane has a total of 36 colors, so the entire surface of the safe-color cube is covered by 216 different colors, as expected.

6.2.2 The CMY and CMYK Color Models

As indicated in Section 6.1, cyan, magenta, and yellow are the secondary colors of light or, alternatively, the primary colors of pigments. For example, when a surface coated with cyan pigment is illuminated with white light, no red light is reflected from the surface. That is, cyan subtracts red light from reflected white light, which itself is composed of equal amounts of red, green, and blue light.

Most devices that deposit colored pigments on paper, such as color printers and copiers, require CMY data input or perform an RGB to CMY conversion internally. This conversion is performed using the simple operation

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{6.2-1}$$

where, again, the assumption is that all color values have been normalized to the range [0, 1]. Equation (6.2-1) demonstrates that light reflected from a surface coated with pure cyan does not contain red (that is, $C = 1 - R$ in the equation).

Similarly, pure magenta does not reflect green, and pure yellow does not reflect blue. Equation (6.2-1) also reveals that RGB values can be obtained easily from a set of CMY values by subtracting the individual CMY values from 1. As indicated earlier, in image processing this color model is used in connection with generating hardcopy output, so the inverse operation from CMY to RGB generally is of little practical interest.

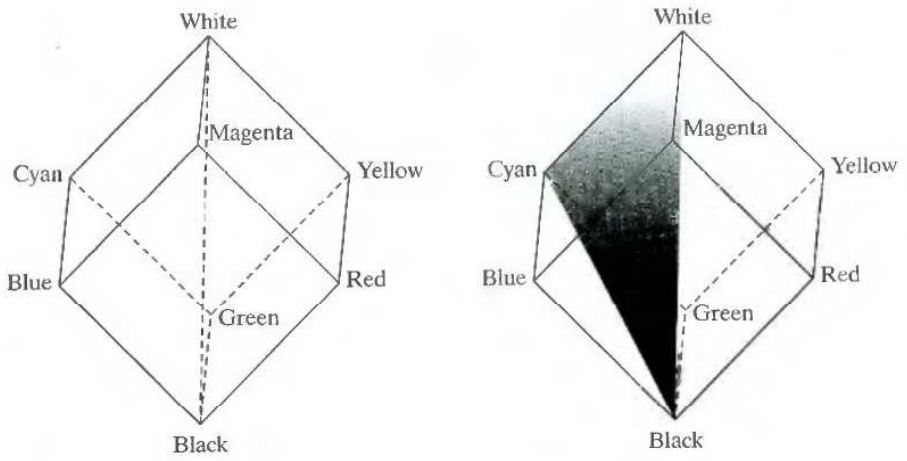
According to Fig. 6.4, equal amounts of the pigment primaries, cyan, magenta, and yellow should produce black. In practice, combining these colors for printing produces a muddy-looking black. So, in order to produce true black (which is the predominant color in printing), a fourth color, *black*, is added, giving rise to the CMYK color model. Thus, when publishers talk about "four-color printing," they are referring to the three colors of the CMY color model plus black.

6.2.3 The HSI Color Model

As we have seen, creating colors in the RGB and CMY models and changing from one model to the other is a straightforward process. As noted earlier, these color systems are ideally suited for hardware implementations. In addition, the RGB system matches nicely with the fact that the human eye is strongly perceptive to red, green, and blue primaries. Unfortunately, the RGB, CMY, and other similar color models are not well suited for *describing* colors in terms that are practical for human interpretation. For example, one does not refer to the color of an automobile by giving the percentage of each of the primaries composing its color. Furthermore, we do not think of color images as being composed of three primary images that combine to form that single image.

When humans view a color object, we describe it by its hue, saturation, and brightness. Recall from the discussion in Section 6.1 that hue is a color attribute that describes a pure color (pure yellow, orange, or red), whereas saturation gives a measure of the degree to which a pure color is diluted by white light. Brightness is a subjective descriptor that is practically impossible to measure. It embodies the achromatic notion of *intensity* and is one of the key factors in describing color sensation. We do know that intensity (gray level) is a most useful descriptor of monochromatic images. This quantity definitely is measurable and easily interpretable. The model we are about to present, called the *HSI* (hue, saturation, intensity) *color model*, decouples the intensity component from the color-carrying information (hue and saturation) in a color image. As a result, the HSI model is an ideal tool for developing image processing algorithms based on color descriptions that are natural and intuitive to humans, who, after all, are the developers and users of these algorithms. We can summarize by saying that RGB is ideal for image color generation (as in image capture by a color camera or image display in a monitor screen), but its use for color *description* is much more limited. The material that follows provides a very effective way to do this.

As discussed in Example 6.1, an RGB color image can be viewed as three monochrome intensity images (representing red, green, and blue), so it should come as no surprise that we should be able to extract intensity from an RGB image. This becomes rather clear if we take the color cube from Fig. 6.7 and stand it on the black (0, 0, 0) vertex, with the white vertex (1, 1, 1) directly



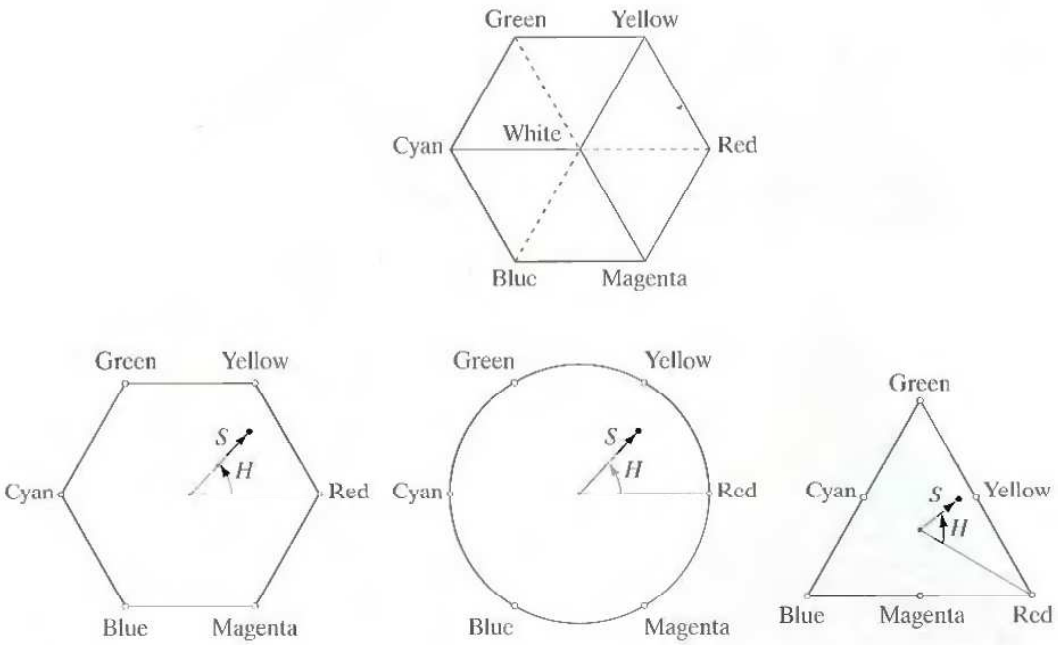
a b

FIGURE 6.12 Conceptual relationships between the RGB and HSI color models.

above it, as shown in Fig. 6.12(a). As noted in connection with Fig. 6.7, the intensity (gray scale) is along the line joining these two vertices. In the arrangement shown in Fig. 6.12, the line (intensity axis) joining the black and white vertices is vertical. Thus, if we wanted to determine the intensity component of any color point in Fig. 6.12, we would simply pass a plane *perpendicular* to the intensity axis and containing the color point. The intersection of the plane with the intensity axis would give us a point with intensity value in the range $[0, 1]$. We also note with a little thought that the saturation (purity) of a color increases as a function of distance from the intensity axis. In fact, the saturation of points on the intensity axis is zero, as evidenced by the fact that all points along this axis are gray.

In order to see how hue can be determined also from a given RGB point, consider Fig. 6.12(b), which shows a plane defined by three points (black, white, and cyan). The fact that the black and white points are contained in the plane tells us that the intensity axis also is contained in the plane. Furthermore, we see that *all* points contained in the plane segment defined by the intensity axis and the boundaries of the cube have the *same* hue (cyan in this case). We would arrive at the same conclusion by recalling from Section 6.1 that all colors generated by three colors lie in the triangle defined by those colors. If two of those points are black and white and the third is a color point, all points on the triangle would have the same hue because the black and white components cannot change the hue (of course, the intensity and saturation of points in this triangle would be different). By rotating the shaded plane about the vertical intensity axis, we would obtain different hues. From these concepts we arrive at the conclusion that the hue, saturation, and intensity values required to form the HSI space can be obtained from the RGB color cube. That is, we can convert any RGB point to a corresponding point is the HSI color model by working out the geometrical formulas describing the reasoning outlined in the preceding discussion.

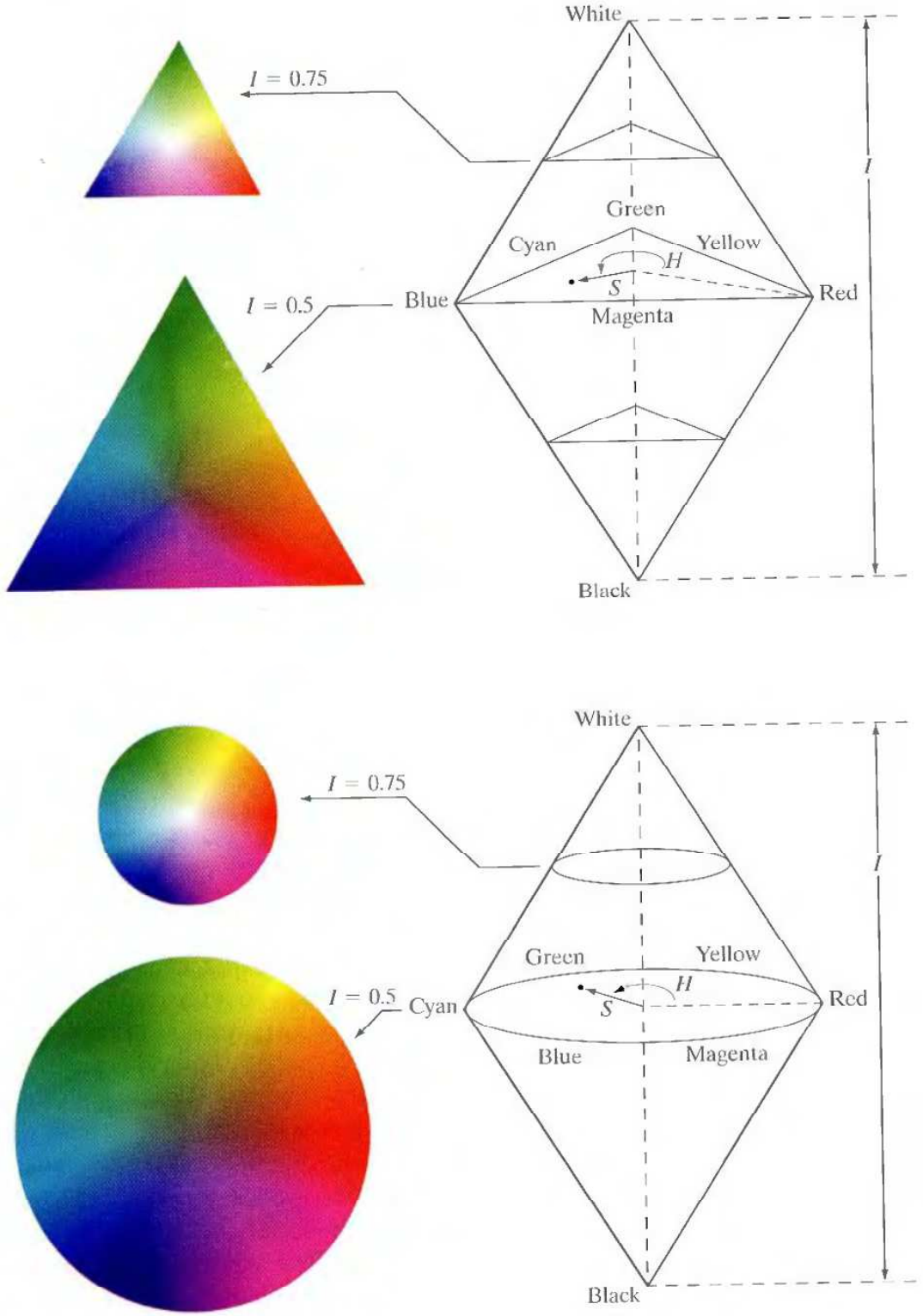
The key point to keep in mind regarding the cube arrangement in Fig. 6.12 and its corresponding HSI color space is that the HSI space is represented by



a
b c d

FIGURE 6.13 Hue and saturation in the HSI color model. The dot is an arbitrary color point. The angle from the red axis gives the hue, and the length of the vector is the saturation. The intensity of all colors in any of these planes is given by the position of the plane on the vertical intensity axis.

a vertical intensity axis and the locus of color points that lie on planes *perpendicular* to this axis. As the planes move up and down the intensity axis, the boundaries defined by the intersection of each plane with the faces of the cube have either a triangular or hexagonal shape. This can be visualized much more readily by looking at the cube down its gray-scale axis, as shown in Fig. 6.13(a). In this plane we see that the primary colors are separated by 120°. The secondary colors are 60° from the primaries, which means that the angle between secondaries also is 120°. Figure 6.13(b) shows the same hexagonal shape and an arbitrary color point (shown as a dot). The hue of the point is determined by an angle from some reference point. Usually (but not always) an angle of 0° from the red axis designates 0 hue, and the hue increases counterclockwise from there. The saturation (distance from the vertical axis) is the length of the vector from the origin to the point. Note that the origin is defined by the intersection of the color plane with the vertical intensity axis. The important components of the HSI color space are the vertical intensity axis, the length of the vector to a color point, and the angle this vector makes with the red axis. Therefore, it is not unusual to see the HSI planes defined is terms of the hexagon just discussed, a triangle, or even a circle, as Figs. 6.13(c) and (d) show. The shape chosen really does not matter, since any one of these shapes can be warped into one of the other two by a geometric transformation. Figure 6.14 shows the HSI model based on color triangles and also on circles.



a
b

FIGURE 6.14 The HSI color model based on (a) triangular and (b) circular color planes. The triangles and circles are perpendicular to the vertical intensity axis.

Converting colors from RGB to HSI

Given an image in RGB color format, the H component of each RGB pixel is obtained using the equation

$$H = \begin{cases} \theta & \text{if } B \leq G \\ 360 - \theta & \text{if } B > G \end{cases} \tag{6.2-2}$$

with

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}[(R-G)+(R-B)]}{[(R-G)^2+(R-B)(G-B)]^{1/2}}\right\}.$$

The saturation component is given by

$$S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)]. \tag{6.2-3}$$

Finally, the intensity component is given by

$$I = \frac{1}{3}(R+G+B). \tag{6.2-4}$$

It is assumed that the RGB values have been normalized to the range $[0, 1]$ and that angle θ is measured with respect to the red axis of the HSI space, as indicated in Fig. 6.13. Hue can be normalized to the range $[0, 1]$ by dividing by 360° all values resulting from Eq. (6.2-2). The other two HSI components already are in this range if the given RGB values are in the interval $[0, 1]$.

The results in Eqs. (6.2-2) through (6.2-4) can be derived from the geometry shown in Figs. 6.12 and 6.13. The derivation is tedious and would not add significantly to the present discussion. The interested reader can consult the book's references or web site for a proof of these equations, as well as for the following HSI to RGB conversion results.

See inside front cover

Consult the book web site for a detailed derivation of the conversion equations between RGB and HSI, and vice versa.

Converting colors from HSI to RGB

Given values of HSI in the interval $[0, 1]$, we now want to find the corresponding RGB values in the same range. The applicable equations depend on the values of H. There are three sectors of interest, corresponding to the 120° intervals in the separation of primaries (see Fig. 6.13). We begin by multiplying H by 360°, which returns the hue to its original range of $[0°, 360°]$.

RG sector $(0° \leq H < 120°)$: When H is in this sector, the RGB components are given by the equations

$$B = I(1 - S) \tag{6.2-5}$$

$$R = I\left[1 + \frac{S\cos H}{\cos(60° - H)}\right] \tag{6.2-6}$$

and

$$G = 3I - (R + B). \tag{6.2-7}$$

GB sector $(120° \leq H < 240°)$: If the given value of H is in this sector, we first subtract 120° from it:

$$H = H - 120°. \tag{6.2-8}$$

Then the RGB components are

$$R = I(1 - S) \tag{6.2-9}$$

$$G = I\left[1 + \frac{S \cos H}{\cos(60° - H)}\right] \tag{6.2-10}$$

and

$$B = 3I - (R + G). \tag{6.2-11}$$

BR sector $(240° \leq H \leq 360°)$: Finally, if H is in this range, we subtract 240° from it:

$$H = H - 240°. \tag{6.2-12}$$

Then the RGB components are

$$G = I(1 - S) \tag{6.2-13}$$

$$B = I\left[1 + \frac{S \cos H}{\cos(60° - H)}\right] \tag{6.2-14}$$

and

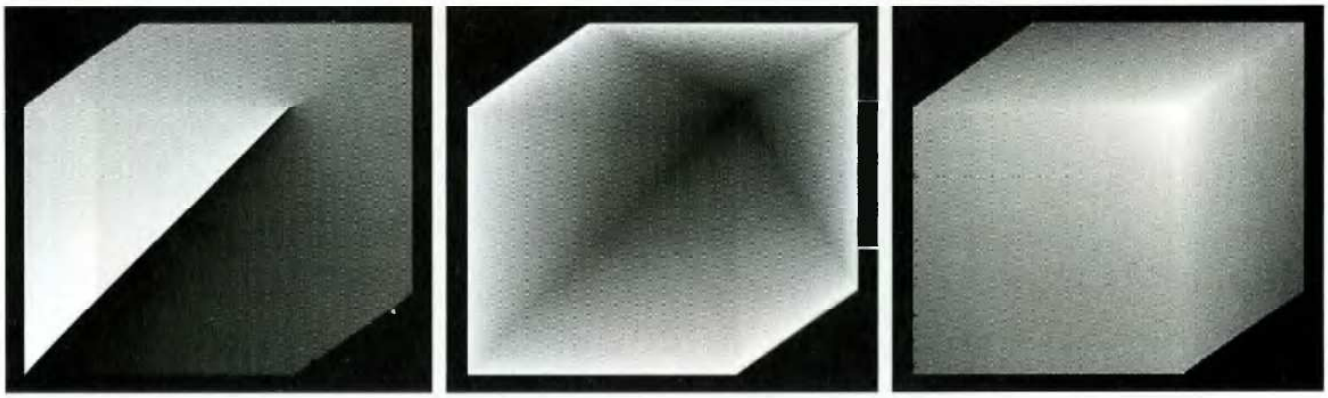
$$R = 3I - (G + B). \tag{6.2-15}$$

Uses of these equations for image processing are discussed in several of the following sections.

EXAMPLE 6.2: The HSI values corresponding to the image of the RGB color cube.

■ Figure 6.15 shows the hue, saturation, and intensity images for the RGB values shown in Fig. 6.8. Figure 6.15(a) is the hue image. Its most distinguishing feature is the discontinuity in value along a 45° line in the front (red) plane of the cube. To understand the reason for this discontinuity, refer to Fig. 6.8, draw a line from the red to the white vertices of the cube, and select a point in the middle of this line. Starting at that point, draw a path to the right, following the cube around until you return to the starting point. The major colors encountered in this path are yellow, green, cyan, blue, magenta, and back to red. According to Fig. 6.13, the values of hue along this path should increase from 0° to 360° (i.e., from the lowest to highest possible values of hue). This is precisely what Fig. 6.15(a) shows because the lowest value is represented as black and the highest value as white in the gray scale. In fact, the hue image was originally normalized to the range $[0, 1]$ and then scaled to 8 bits; that is, it was converted to the range $[0, 255]$; for display.

The saturation image in Fig. 6.15(b) shows progressively darker values toward the white vertex of the RGB cube, indicating that colors become less and less



a b c

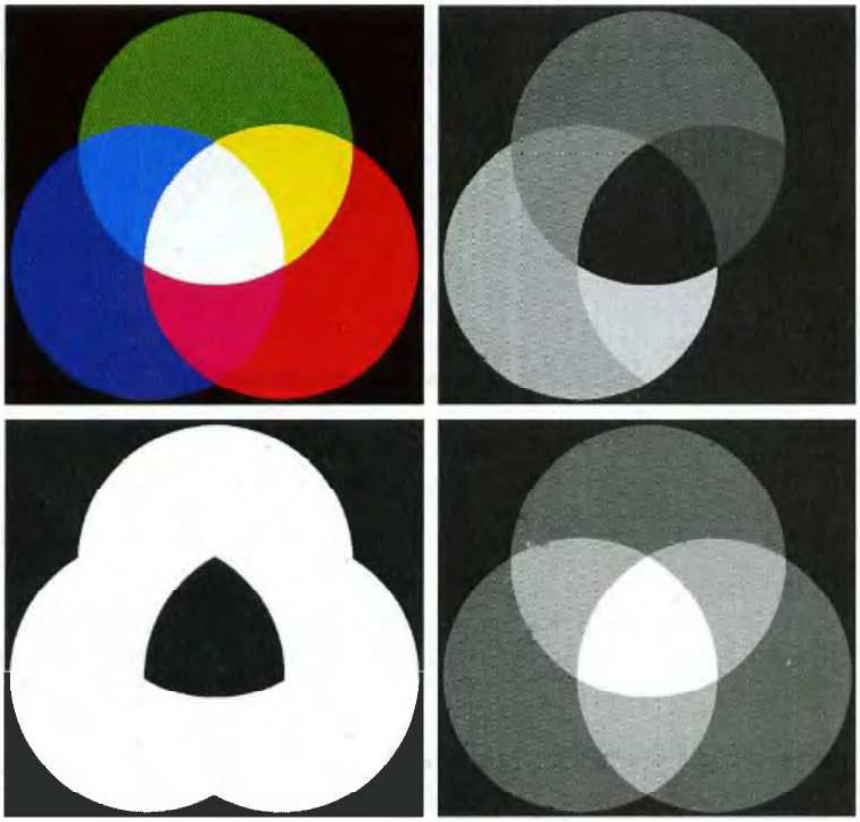
FIGURE 6.15 HSI components of the image in Fig. 6.8. (a) Hue, (b) saturation, and (c) intensity images.

saturated as they approach white. Finally, every pixel in the intensity image shown in Fig. 6.15(d) is the average of the RGB values at the corresponding pixel in Fig. 6.8. ■

Manipulating HSI component images

In the following discussion, we take a look at some simple techniques for manipulating HSI component images. This will help develop familiarity with these components and also help deepen our understanding of the HSI color model. Figure 6.16(a) shows an image composed of the primary and secondary RGB colors. Figures 6.16(b) through (d) show the H, S, and I components of this image. These images were generated using Eqs. (6.2-2) through (6.2-4). Recall from the discussion earlier in this section that the gray-level values in Fig. 6.16(b) correspond to angles; thus, for example, because red corresponds to 0°, the red region in Fig. 6.16(a) mapped to a black region in the hue image. Similarly, the gray levels in Fig. 6.16(c) correspond to saturation (they were scaled to [0, 255] for display), and the gray levels in Fig. 6.16(d) are average intensities.

To change the individual color of any region in the RGB image, we change the values of the corresponding region in the hue image of Fig. 6.16(b). Then we convert the new H image, along with the unchanged S and I images, back to RGB using the procedure explained in connection with Eqs. (6.2-5) through (6.2-15). To change the saturation (purity) of the color in any region, we follow the same procedure, except that we make the changes in the saturation image in HSI space. Similar comments apply to changing the average intensity of any region. Of course, these changes can be made simultaneously. For example, the image in Fig. 6.17(a) was obtained by changing to 0 the pixels corresponding to the blue and green regions in Fig. 6.16(b). In Fig. 6.17(b) we reduced by half the saturation of the cyan region in component image S from Fig. 6.16(c). In Fig. 6.17(c) we reduced by half the intensity of the central white region in the intensity image of Fig. 6.16(d). The result of converting this modified HSI image back to RGB is shown in Fig. 6.17(d). As expected, we see in this figure that



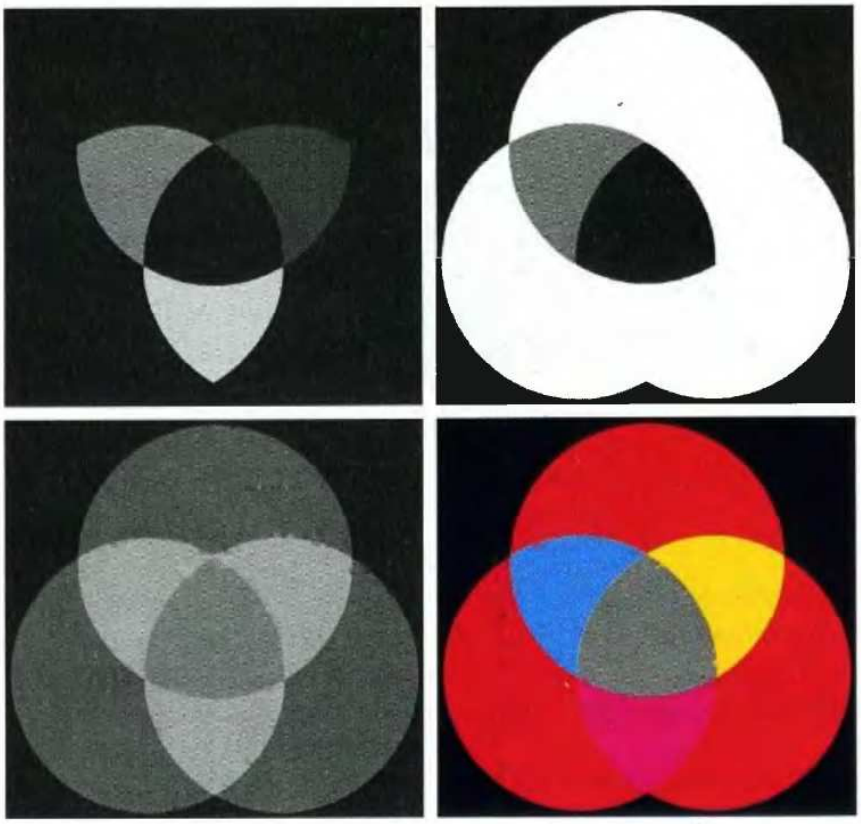
a b
c d

FIGURE 6.16 (a) RGB image and the components of its corresponding HSI image: (b) hue, (c) saturation, and (d) intensity.

the outer portions of all circles are now red; the purity of the cyan region was diminished, and the central region became gray rather than white. Although these results are simple, they illustrate clearly the power of the HSI color model in allowing *independent* control over hue, saturation, and intensity, quantities with which we are quite familiar when describing colors.

6.3 Pseudocolor Image Processing

Pseudocolor (also called *false color*) image processing consists of assigning colors to gray values based on a specified criterion. The term *pseudo* or *false* color is used to differentiate the process of assigning colors to monochrome images from the processes associated with true color images, a topic discussed starting in Section 6.4. The principal use of pseudocolor is for human visualization and interpretation of gray-scale events in an image or sequence of images. As noted



a b
c d

FIGURE 6.17 (a)–(c) Modified HSI component images. (d) Resulting RGB image. (See Fig. 6.16 for the original HSI images.)

at the beginning of this chapter, one of the principal motivations for using color is the fact that humans can discern thousands of color shades and intensities, compared to only two dozen or so shades of gray.

6.3.1 Intensity Slicing

The technique of *intensity* (sometimes called *density*) *slicing* and color coding is one of the simplest examples of pseudocolor image processing. If an image is interpreted as a 3-D function (intensity versus spatial coordinates), the method can be viewed as one of placing planes parallel to the coordinate plane of the image; each plane then "slices" the function in the area of intersection. Figure 6.18 shows an example of using a plane at $f(x, y) = l_i$ to slice the image function into two levels.

If a different color is assigned to each side of the plane shown in Fig. 6.18, any pixel whose gray level is above the plane will be coded with one color, and any pixel below the plane will be coded with the other. Levels that lie on the plane

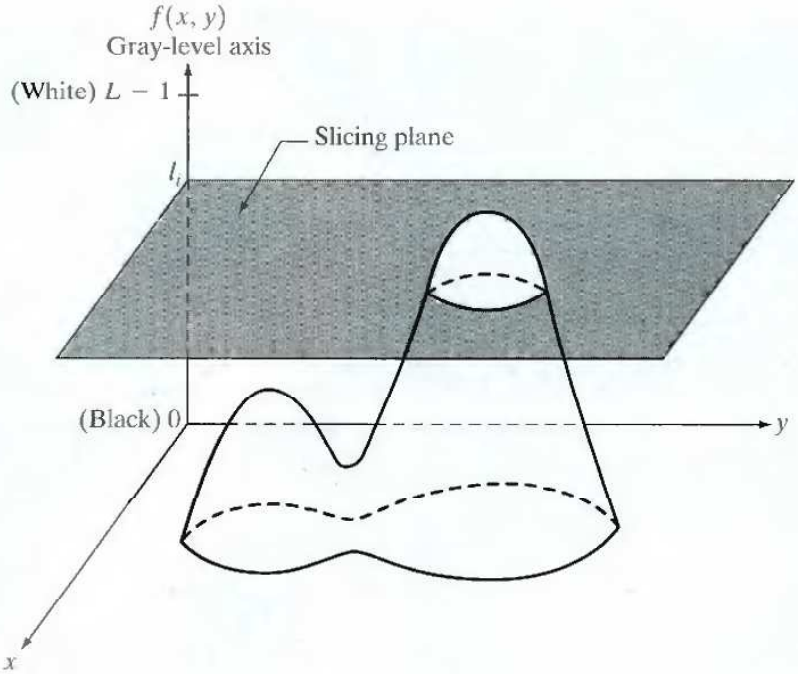


FIGURE 6.18 Geometric interpretation of the intensity-slicing technique.

itself may be arbitrarily assigned one of the two colors. The result is a two-color image whose relative appearance can be controlled by moving the slicing plane up and down the gray-level axis.

In general, the technique may be summarized as follows. Let $[0, L - 1]$ represent the gray scale (see Section 2.3.4), let level l_0 represent black $[f(x, y) = 0]$, and level l_{L-1} represent white $[f(x, y) = L - 1]$. Suppose that P planes perpendicular to the intensity axis are defined at levels $l_1, l_2, \ldots, l_P$. Then, assuming that $0 < P < L - 1$, the P planes partition the gray scale into $P + 1$ intervals, $V_1, V_2, \ldots, V_{P+1}$. Gray-level to color assignments are made according to the relation

$$f(x, y) = c_k \qquad \text{if } f(x, y) \in V_k \tag{6.3-1}$$

where c_k is the color associated with the kth intensity interval V_k defined by the partitioning planes at $l = k - 1$ and $l = k$.

The idea of planes is useful primarily for a geometric interpretation of the intensity-slicing technique. Figure 6.19 shows an alternative representation that defines the same mapping as in Fig. 6.18. According to the mapping function shown in Fig. 6.19, any input gray level is assigned one of two colors, depending on whether it is above or below the value of l_i. When more levels are used, the mapping function takes on a staircase form.

EXAMPLE 6.3: Intensity slicing.

■ A simple, but practical, use of intensity slicing is shown in Fig. 6.20. Figure 6.20(a) is a monochrome image of the Picker Thyroid Phantom (a radiation test pattern), and Fig. 6.20(b) is the result of intensity slicing this image into eight color regions. Regions that appear of constant intensity in the monochrome image

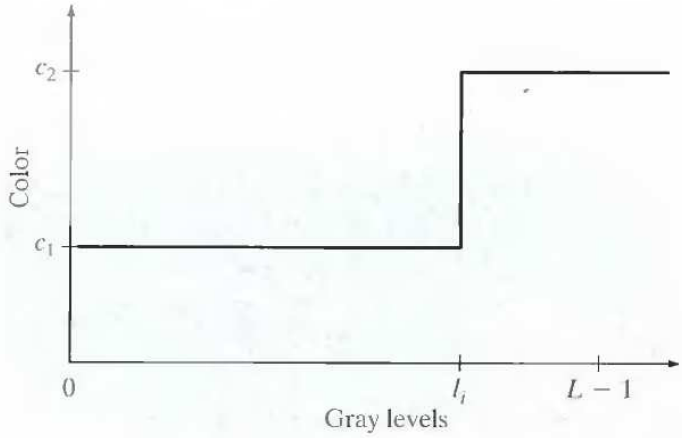
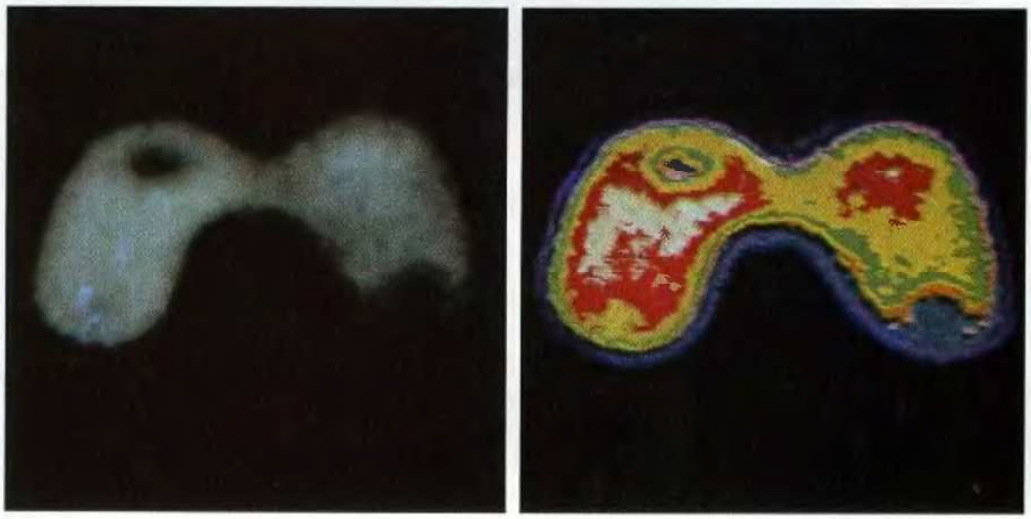


FIGURE 6.19 An alternative representation of the intensity-slicing technique.

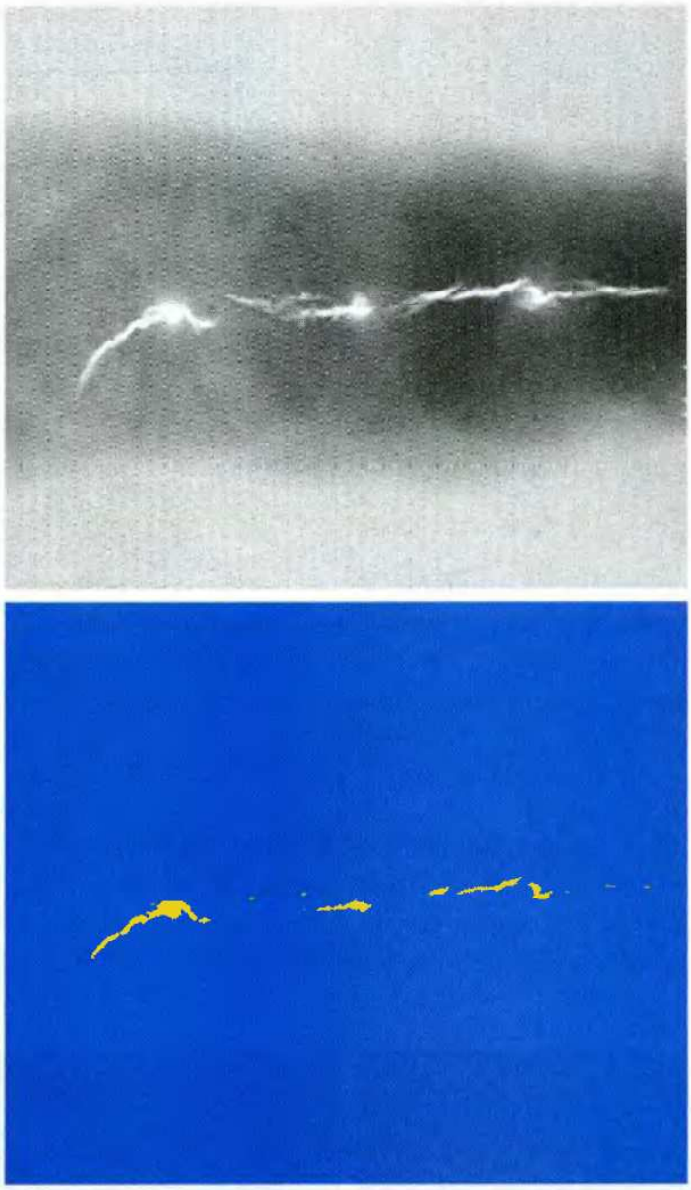
are really quite variable, as shown by the various colors in the sliced image. The left lobe, for instance, is a dull gray in the monochrome image, and picking out variations in intensity is difficult. By contrast, the color image clearly shows eight different regions of constant intensity, one for each of the colors used. ■

In the preceding simple example, the gray scale was divided into intervals and a different color was assigned to each region, without regard for the meaning of the gray levels in the image. Interest in that case was simply to view the different gray levels constituting the image. Intensity slicing assumes a much more meaningful and useful role when subdivision of the gray scale is based on physical characteristics of the image. For instance, Fig. 6.21(a) shows an X-ray image of a weld



a b

FIGURE 6.20 (a) Monochrome image of the Picker Thyroid Phantom. (b) Result of density slicing into eight colors. (Courtesy of Dr. J. L. Blankenship, Instrumentation and Controls Division, Oak Ridge National Laboratory.)



a
b

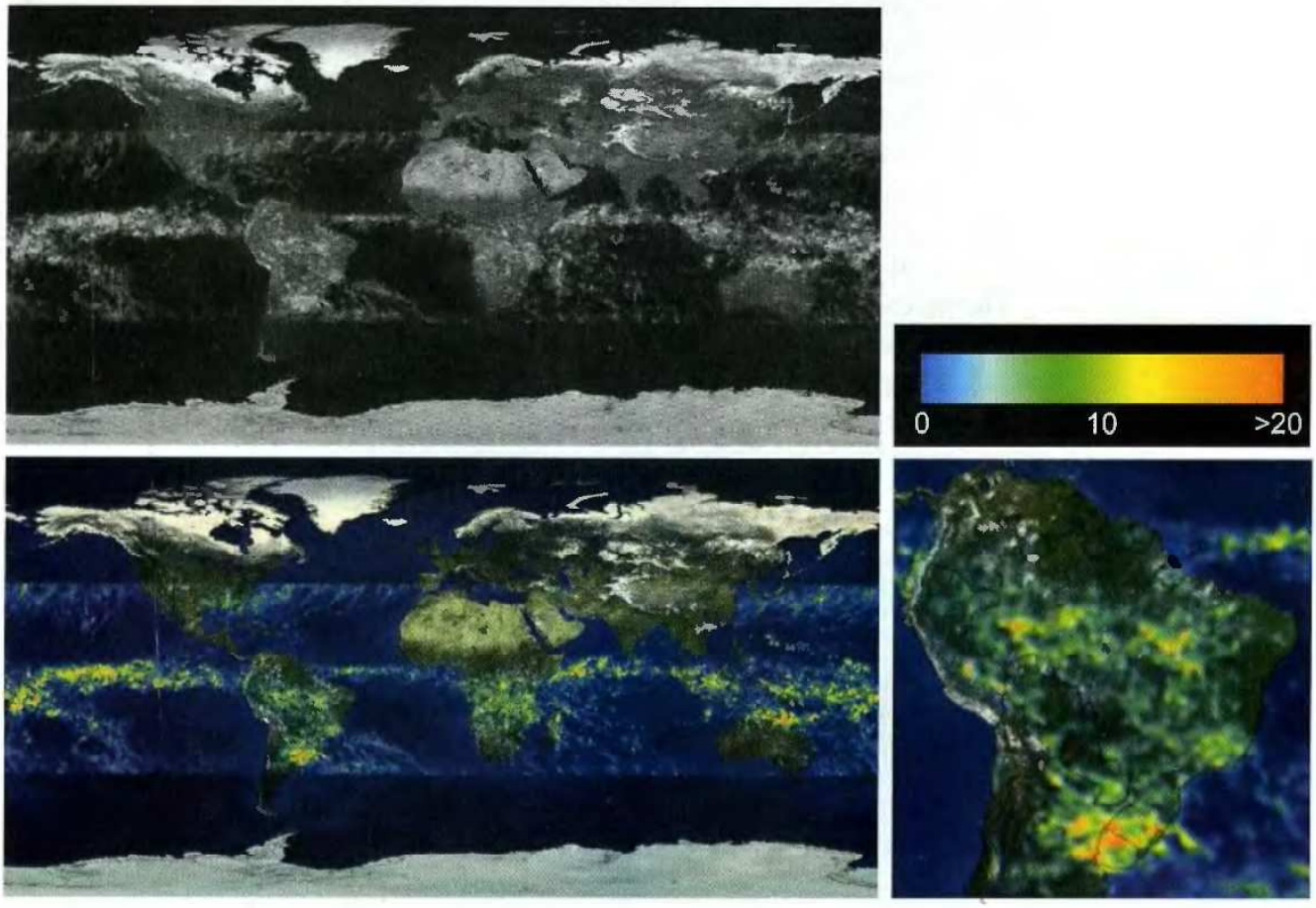
FIGURE 6.21 (a) Monochrome X-ray image of a weld. (b) Result of color coding. (Original image courtesy of X-TEK Systems, Ltd.)

(the horizontal dark region) containing several cracks and porosities (the bright, white streaks running horizontally through the middle of the image). It is known that when there is a porosity or crack in a weld, the full strength of the X-rays going through the object saturates the imaging sensor on the other side of the object (see Section 2.3). Thus, gray levels of value 255 in an 8-bit image coming from such a system automatically imply a problem with the weld. If a human were to be the ultimate judge of the analysis and manual processes were employed to inspect welds (still a common procedure today), a simple color coding that assigns one color to level 255 and another to all other gray levels would simplify the inspector's job considerably. Figure 6.21(b) shows the result. No explanation is required to arrive at the conclusion that human error rates would be lower if images were displayed in the form of Fig. 6.21(b), instead of the form shown in Fig. 6.21(a). In other words, if the exact values of gray levels one is looking for are known, intensity slicing is a simple but powerful aid in visualization, especially if numerous images are involved. The following is a significantly more complex example.

EXAMPLE 6.4: Use of color to highlight rainfall levels.

■ Measurement of rainfall levels, especially in the tropical regions of the Earth, is of interest in diverse applications dealing with the environment. Accurate measurements using ground-based sensors are difficult and expensive to acquire, and total rainfall figures are even more difficult to obtain because a significant portion of precipitation occurs over the ocean. One approach for obtaining rainfall figures is to use a satellite. The TRMM (Tropical Rainfall Measuring Mission) satellite utilizes, among others, three sensors specially designed to detect rain: a precipitation radar, a microwave imager, and a visible and infrared scanner (see Sections 1.3 and 2.3 regarding image sensing modalities).

The results from the various rain sensors are processed, resulting in estimates of average rainfall over a given time period in the area monitored by the sensors. From these estimates, it is not difficult to generate gray-scale images whose intensity values correspond directly to rainfall, with each pixel representing a physical land area whose size depends on the resolution of the sensors. Such an intensity image is shown in Fig. 6.22(a), where the area monitored by the satellite



a b
c d

FIGURE 6.22 (a) Gray-scale image in which intensity (in the lighter horizontal band shown) corresponds to average monthly rainfall. (b) Colors assigned to intensity values. (c) Color-coded image. (d) Zoom of the South America region. (Courtesy of NASA.)

is the slightly lighter horizontal band in the middle one-third of the picture (these are the tropical regions). In this particular example, the rainfall values are average monthly values (in inches) over a three-year period.

Visual examination of this picture for rainfall patterns is quite difficult, if not impossible. However, suppose that we code gray levels from 0 to 255 using the colors shown in Fig. 6.22(b). Values toward the blues signify low values of rainfall, with the opposite being true for red. Note that the scale tops out at pure red for values of rainfall greater than 20 inches. Figure 6.22(c) shows the result of color coding the gray image with the color map just discussed. The results are much easier to interpret, as shown in this figure and in the zoomed area of Fig. 6.22(d). In addition to providing global coverage, this type of data allows meteorologists to calibrate ground-based rain monitoring systems with greater precision than ever before. ■

6.3.2 Gray Level to Color Transformations

Other types of transformations are more general and thus are capable of achieving a wider range of pseudocolor enhancement results than the simple slicing technique discussed in the preceding section. An approach that is particularly attractive is shown in Fig. 6.23. Basically, the idea underlying this approach is to perform three independent transformations on the gray level of any input pixel. The three results are then fed separately into the red, green, and blue channels of a color television monitor. This method produces a composite image whose color content is modulated by the nature of the transformation functions. Note that these are transformations on the gray-level values of an image and are not functions of position.

The method discussed in the previous section is a special case of the technique just described. There, piecewise linear functions of the gray levels (Fig. 6.19) are used to generate colors. The method discussed in this section, on the other hand, can be based on smooth, nonlinear functions, which, as might be expected, gives the technique considerable flexibility.

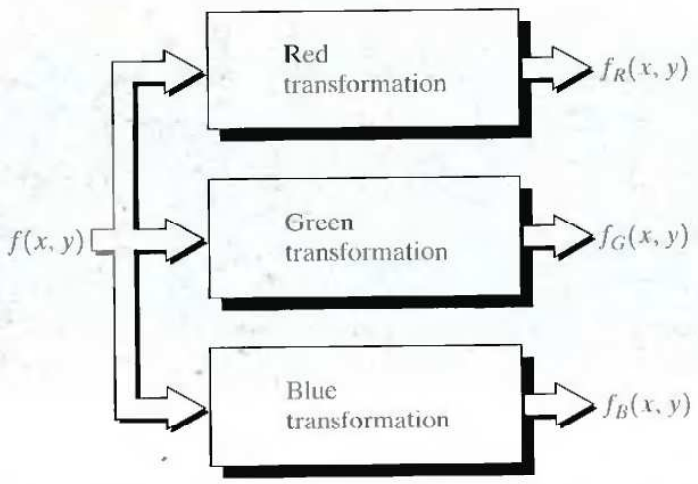


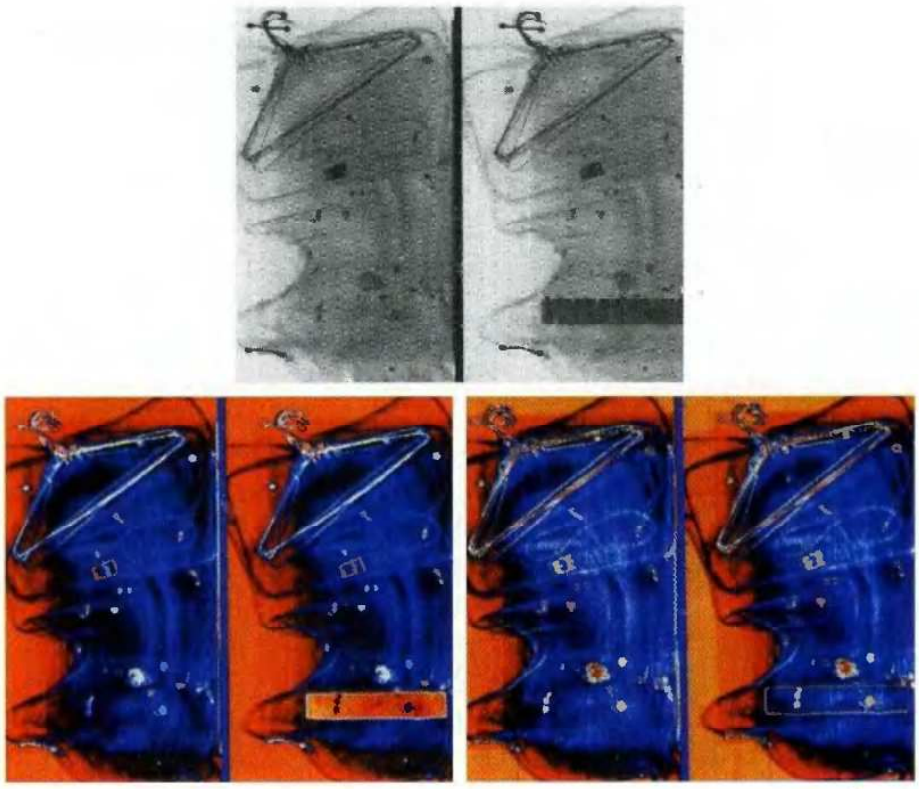
FIGURE 6.23 Functional block diagram for pseudocolor image processing. f_R, f_G, and f_B are fed into the corresponding red, green, and blue inputs of an RGB color monitor.

EXAMPLE 6.5: Use of pseudocolor for highlighting explosives contained in luggage.

■ Figure 6.24(a) shows two monochrome images of luggage obtained from an airport X-ray scanning system. The image on the left contains ordinary articles. The image on the right contains the same articles, as well as a block of simulated plastic explosives. The purpose of this example is to illustrate the use of gray level to color transformations to obtain various degrees of enhancement.

Figure 6.25 shows the transformation functions used. These sinusoidal functions contain regions of relatively constant value around the peaks as well as regions that change rapidly near the valleys. Changing the phase and frequency of each sinusoid can emphasize (in color) ranges in the gray scale. For instance, if all three transformations have the same phase and frequency, the output image will be monochrome. A small change in the phase between the three transformations produces little change in pixels whose gray levels correspond to peaks in the sinusoids, especially if the sinusoids have broad profiles (low frequencies). Pixels with gray-level values in the steep section of the sinusoids are assigned a much stronger color content as a result of significant differences between the amplitudes of the three sinusoids caused by the phase displacement between them.

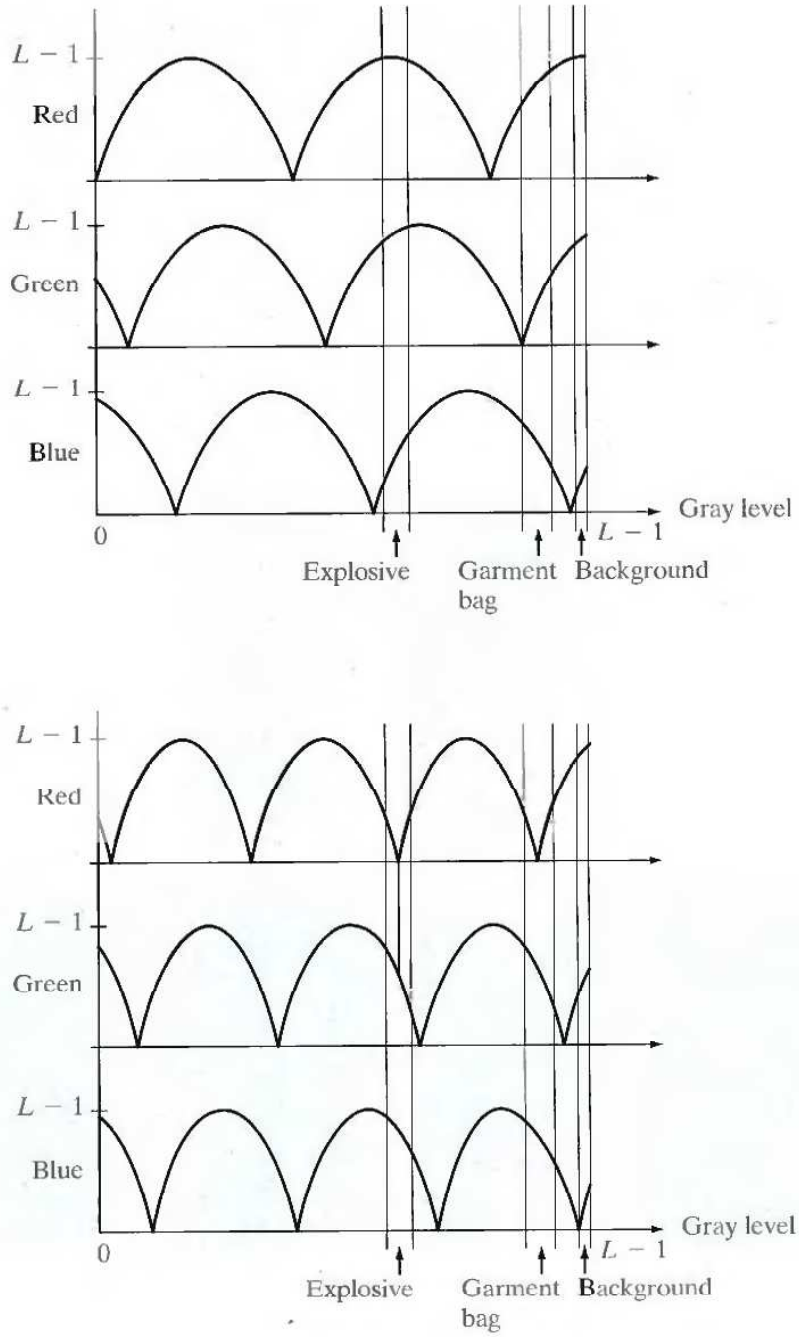
The image shown in Fig. 6.24(b) was obtained with the transformation functions in Fig. 6.25(a), which shows the gray-level bands corresponding to the explosive, garment bag, and background, respectively. Note that the explosive and background have quite different gray levels, but they were both coded with



a
b c

FIGURE 6.24 Pseudocolor enhancement by using the gray-level to color transformations in Fig. 6.25. (Original image courtesy of Dr. Mike Hurwitz, Westinghouse.)

approximately the same color as a result of the periodicity of the sine waves. The image shown in Fig. 6.24(c) was obtained with the transformation functions in Fig. 6.25(b). In this case the explosives and garment bag intensity bands were mapped by similar transformations and thus received essentially the same color assignments. Note that this mapping allows an observer to "see" through the explosives. The background mappings were about the same as those used for Fig. 6.24(b), producing almost identical color assignments. ■



a
b

FIGURE 6.25 Transformation functions used to obtain the images in Fig. 6.24.

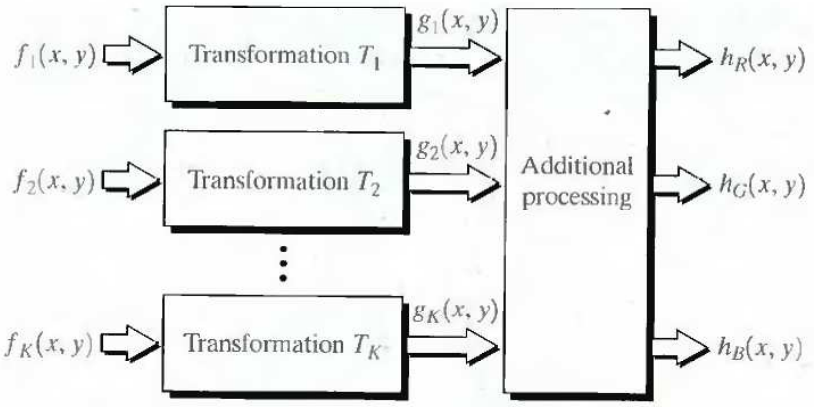


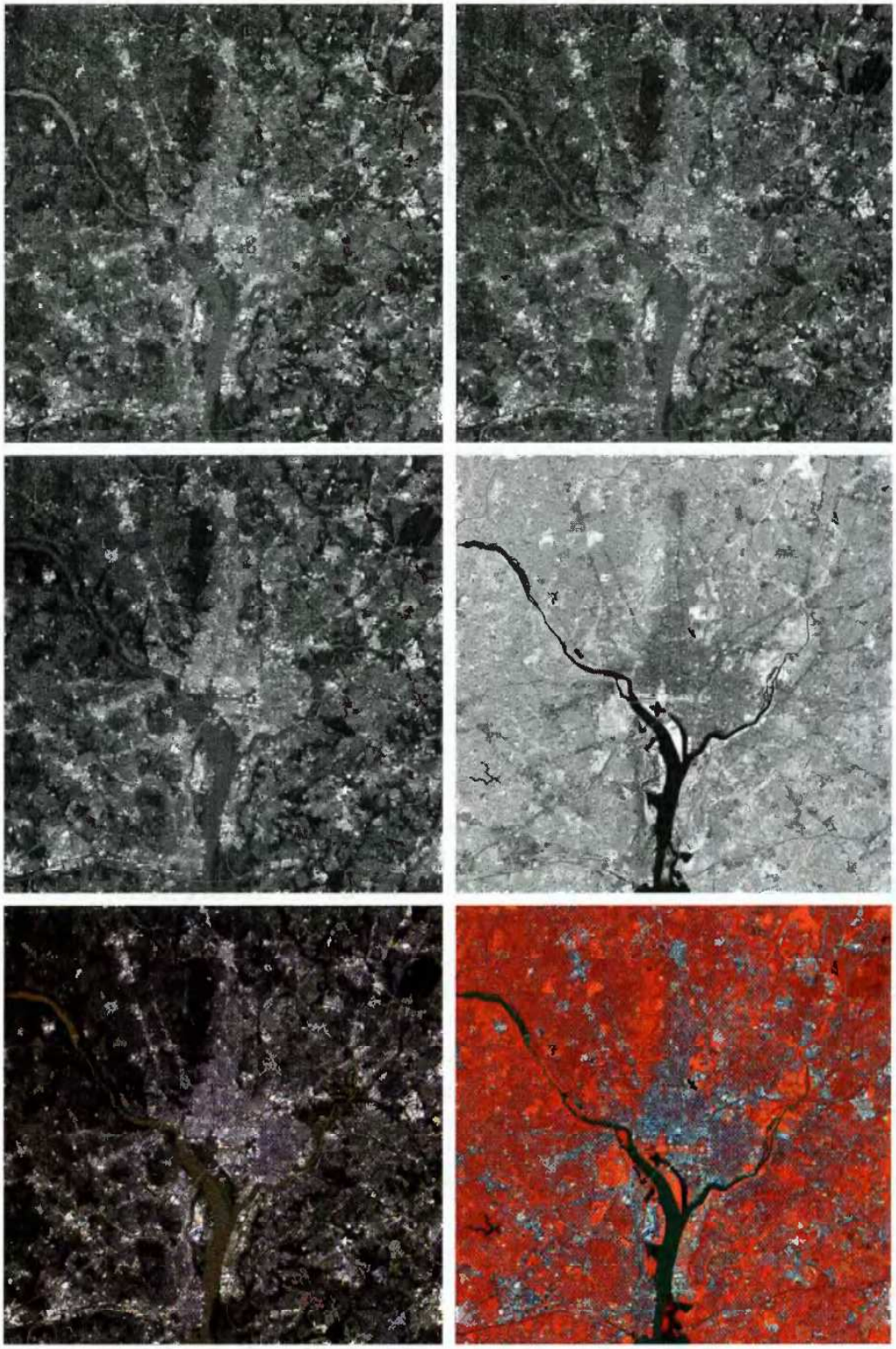
FIGURE 6.26 A pseudocolor coding approach used when several monochrome images are available.

The approach shown in Fig. 6.23 is based on a single monochrome image. Often, it is of interest to combine several monochrome images into a single color composite, as shown in Fig. 6.26. A frequent use of this approach (illustrated in Example 6.6) is in multispectral image processing, where different sensors produce individual monochrome images, each in a different spectral band. The types of additional processes shown in Fig. 6.26 can be techniques such as color balancing (see Section 6.5.4), combining images, and selecting the three images for display based on knowledge about response characteristics of the sensors used to generate the images.

EXAMPLE 6.6: Color coding of multispectral images.

■ Figures 6.27(a) through (d) show four spectral satellite images of Washington, D.C., including part of the Potomac River. The first three images are in the visible red, green, and blue, and the fourth is in the near infrared (see Table 1.1 and Fig. 1.10). Figure 6.27(e) is the full-color image obtained by combining the first three images into an RGB image. Full-color images of dense areas are difficult to interpret, but one notable feature of this image is the difference in color in various parts of the Potomac River. Figure 6.27(f) is a little more interesting. This image was formed by replacing the red component of Fig. 6.27(e) with the near-infrared image. From Table 1.1, we know that this band is strongly responsive to the biomass components of a scene. Figure 6.27(f) shows quite clearly the difference between biomass (in red) and the human-made features in the scene, composed primarily of concrete and asphalt, which appear bluish in the image.

The type of processing just illustrated is quite powerful in helping visualize events of interest in complex images, especially when those events our beyond our normal sensing capabilities. Figure 6.28 is an excellent illustration of this. These are images of the Jupiter moon Io, shown in pseudocolor by combining several of the sensor images from the *Galileo* spacecraft, some of which are in spectral regions not visible to the eye. However, by understanding the physical and chemical processes likely to affect sensor response, it is possible to combine the sensed images into a meaningful pseudocolor map. One way to combine the sensed image data is by how they show either differences in surface chemical composition or changes in the way the surface reflects sunlight. For example, in the pseudocolor image in Fig. 6.28(b), bright red depicts material newly ejected



a b
c d
e f

FIGURE 6.27 (a) (d) Images in bands 1 4 in Fig. 1.10 (see Table 1.1). (e) Color composite image obtained by treating (a), (b), and (c) as the red, green, blue components of an RGB image. (f) Image obtained in the same manner, but using in the red channel the near-infrared image in (d). (Original multispectral images courtesy of NASA.)

a
b

FIGURE 6.28 (a) Pseudocolor rendition of Jupiter Moon Io. (b) A close-up. (Courtesy of NASA.)

from an active volcano on Io, and the surrounding yellow materials are older sulfur deposits. This image conveys these characteristics much more readily than would be possible by analyzing the component images individually. ■

6.4 Basics of Full-Color Image Processing

In this section be begin the study of processing techniques applicable to full-color images. Although they are far from being exhaustive, the techniques developed in the sections that follow are illustrative of how full-color images are handled for a variety of image processing tasks. Full-color image processing approaches fall into two major categories. In the first category, we process each component image individually and then form a composite processed color image

from the individually processed components. In the second category, we work with color pixels directly. Because full-color images have at least three components, color pixels really are vectors. For example, in the RGB system, each color point can be interpreted as a vector extending from the origin to that point in the RGB coordinate system (see Fig. 6.7).

Let $\mathbf{c}$ represent an arbitrary vector in RGB color space:

$$\mathbf{c} = \begin{bmatrix} c_R \\ c_G \\ c_B \end{bmatrix} = \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{6.4-1}$$

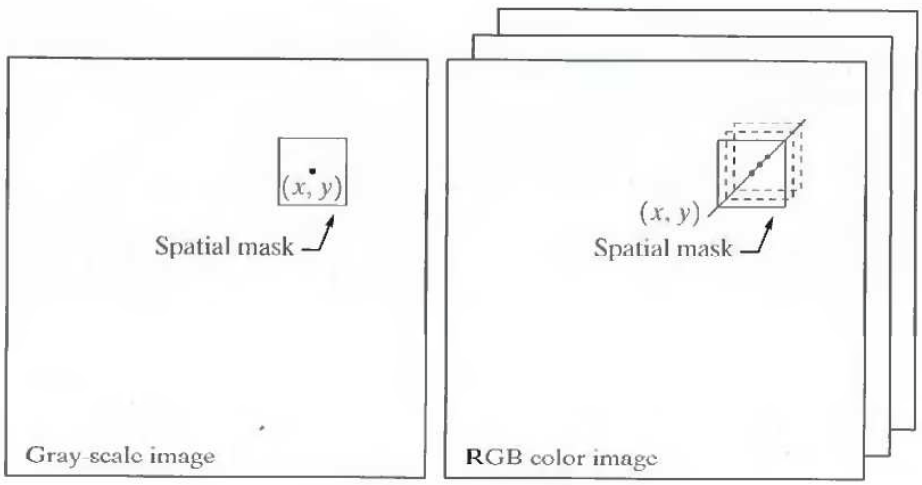
This equation indicates that the components of $\mathbf{c}$ are simply the RGB components of a color image at a point. We take into account the fact that the color components are a function of coordinates (x, y) by using the notation

$$\mathbf{c}(x, y) = \begin{bmatrix} c_R(x, y) \\ c_G(x, y) \\ c_B(x, y) \end{bmatrix} = \begin{bmatrix} R(x, y) \\ G(x, y) \\ B(x, y) \end{bmatrix}. \tag{6.4-2}$$

For an image of size $M \times N$, there are MN such vectors, $\mathbf{c}(x, y)$, for $x = 0, 1, 2, \ldots, M - 1$; $y = 0, 1, 2, \ldots, N - 1$.

It is important to keep clearly in mind that Eq. (6.4-2) depicts a vector whose components are *spatial* variables in x and y. This is a frequent source of confusion that can be avoided by focusing on the fact that our interest lies on spatial processes. That is, we are interested in image processing techniques formulated in x and y. The fact that the pixels are now color pixels introduces a factor that, in its easiest formulation, allows us to process a color image by processing each of its component images separately, using standard gray-scale image processing methods. However, the results of individual color component processing are not always equivalent to direct processing in color vector space, in which case we must formulate new approaches.

In order for per-color-component and vector-based processing to be equivalent, two conditions have to be satisfied: First, the process has to be applicable to both vectors and scalars. Second, the operation on each component of a vector must be independent of the other components. As an illustration, Fig. 6.29



a b

FIGURE 6.29 Spatial masks for gray-scale and RGB color images.

shows neighborhood spatial processing of gray-scale and full-color images. Suppose that the process is neighborhood averaging. In Fig. 6.29(a), averaging would be accomplished by summing the gray levels of all the pixels in the neighborhood and dividing by the total number of pixels in the neighborhood. In Fig. 6.29(b), averaging would be done by summing all the vectors in the neighborhood and dividing each component by the total number of vectors in the neighborhood. But each component of the average vector is the sum of the pixels in the image corresponding to that component, which is the same as the result that would be obtained if the averaging were done on a per-color-component basis and then the vector was formed. We show this in more detail in the following sections. We also show methods in which the results of the two approaches are not the same.

6.5 Color Transformations

The techniques described in this section, collectively called *color transformations*, deal with processing the components of a color image within the context of a *single* color model, as opposed to the conversion of those components between models (like the RGB-to-HSI and HSI-to-RGB conversion transformations of Section 6.2.3).

6.5.1 Formulation

As with the gray-level transformation techniques of Chapter 3, we model color transformations using the expression

$$g(x, y) = T[f(x, y)] \tag{6.5-1}$$

where $f(x, y)$ is a color input image, $g(x, y)$ is the transformed or processed color output image, and T is an operator on f over a spatial neighborhood of (x, y). The principal difference between this equation and Eq. (3.1-1) is in its interpretation. The pixel values here are triplets or quartets (i.e., groups of three or four values) from the color space chosen to represent the images, as illustrated in Fig. 6.29(b).

Analogous to the approach we used to introduce the basic gray-level transformations in Section 3.2, we will restrict attention in this section to color transformations of the form

$$s_i = T_i(r_1, r_2, \ldots, r_n), \qquad i = 1, 2, \ldots, n \tag{6.5-2}$$

where, for notational simplicity, r_i and s_i are variables denoting the color components of $f(x, y)$ and $g(x, y)$ at any point (x, y), n is the number of color components, and $\{T_1, T_2, \ldots, T_n\}$ is a set of *transformation* or *color mapping functions* that operate on r_i to produce s_i. Note that n transformations, T_i, combine to implement the single transformation function, T, in Eq. (6.5-1). The color space chosen to describe the pixels of f and g determines the value of n. If the RGB color space is selected, for example, $n = 3$ and r_1, r_2, and r_3 denote the red, green, and blue components of the input image, respectively. If the CMYK or HSI color spaces are chosen, $n = 4$ or $n = 3$.

Figure 6.30(a) shows a high-resolution color image of a bowl of strawberries and cup of coffee that was digitized from a large format (4″ × 5″) color negative. The second row of the figure contains the components of the initial CMYK scan. In these images, black represents 0 and white represents 1 in each CMYK color component. Thus, we see that the strawberries are composed of large amounts of magenta and yellow because the images corresponding to these two CMYK components are the brightest. Black is used sparingly and is generally confined to the coffee and shadows within the bowl of strawberries. When the CMYK image is converted to RGB, as shown in the third row of the figure, the strawberries are seen to contain a large amount of red and very little (although some) green and blue. The last row of Fig. 6.30 shows the HSI components of Fig. 6.30(a)—computed using Eqs. (6.2-2) through (6.2-4). As expected, the intensity component is a monochrome rendition of the full-color original. In addition, the strawberries are relatively pure in color; they possess the highest saturation or least dilution by white light of any of the hues in the image. Finally, we note some difficulty in interpreting the hue component. The problem is compounded by the fact that (1) there is a discontinuity in the HSI model where 0° and 360° meet, and (2) hue is undefined for a saturation of 0 (i.e., for white, black, and pure grays). The discontinuity of the model is most apparent around the strawberries, which are depicted in gray level values near both black (0) and white (1). The result is an unexpected mixture of highly contrasting gray levels to represent a single color—red.

Any of the color space components in Fig. 6.30 can be used in conjunction with Eq. (6.5-2). In theory, any transformation can be performed in any color model. In practice, however, some operations are better suited to specific models. For a given transformation, the cost of converting between representations must be factored into the decision regarding the color space in which to implement it. Suppose, for example, that we wish to modify the intensity of the image in Fig. 6.30(a) using

$$g(x, y) = kf(x, y) \tag{6.5-3}$$

where $0 < k < 1$. In the HSI color space, this can be done with the simple transformation

$$s_3 = kr_3 \tag{6.5-4}$$

where $s_1 = r_1$ and $s_2 = r_2$. Only HSI intensity component r_3 is modified. In the RGB color space, three components must be transformed:

$$s_i = kr_i \qquad i = 1, 2, 3. \tag{6.5-5}$$

The CMY space requires a similar set of linear transformations:

$$s_i = kr_i + (1 - k) \qquad i = 1, 2, 3. \tag{6.5-6}$$

Although the HSI transformation involves the fewest number of operations, the computations required to convert an RGB or CMY(K) image to the HSI space more than offsets (in this case) the advantages of the simpler transformation—

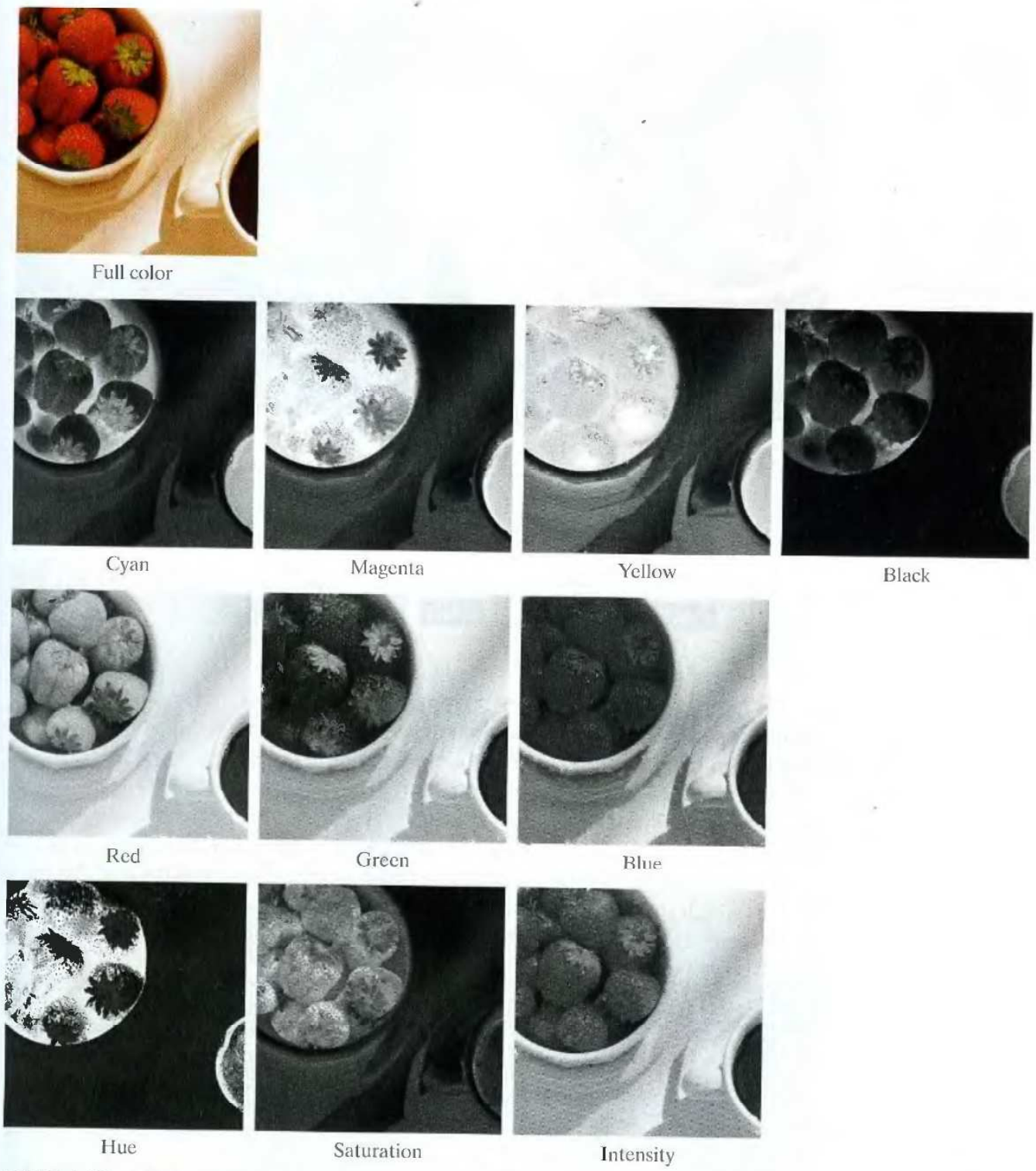
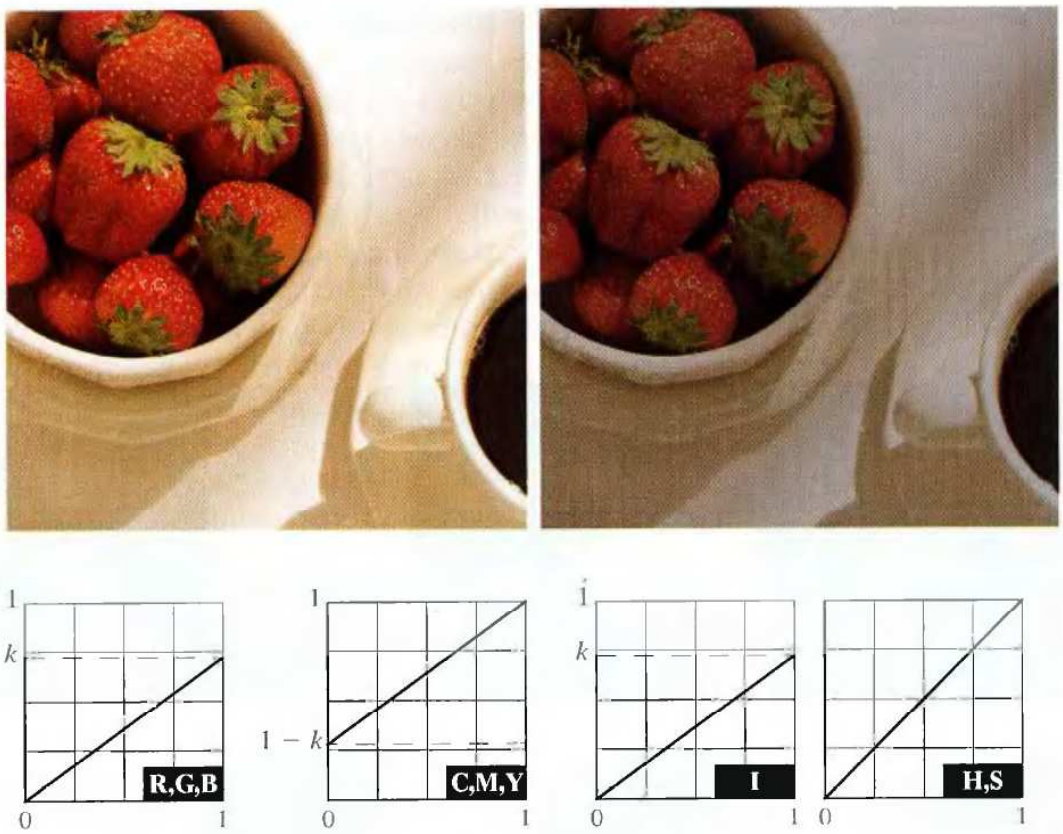


FIGURE 6.30 A full-color image and its various color-space components. (Original image courtesy of MedData Interactive.)

the conversion calculations are more computationally intense than the intensity transformation itself. Regardless of the color space selected, however, the output is the same. Figure 6.31(b) shows the result of applying any of the transformations in Eqs. (6.5-4) through (6.5-6) to the image of Fig. 6.30(a) using $k = 0.7$. The mapping functions themselves are depicted graphically in Figs. 6.31(c) through (e).



a b
c d e

FIGURE 6.31 Adjusting the intensity of an image using color transformations. (a) Original image. (b) Result of decreasing its intensity by 30% (i.e., letting $k = 0.7$). (c)–(e) The required RGB, CMY, and HSI transformation functions. (Original image courtesy of MedData Interactive.)

It is important to note that each transformation defined in Eqs. (6.5-4) through (6.5-6) depends only on one component within its color space. For example, the red output component, s_1, in Eq. (6.5-5) is independent of the green (r_2) and blue (r_3) inputs; it depends only on the red (r_1) input. Transformations of this type are among the simplest and most used color processing tools and can be carried out on a per-color-component basis, as mentioned at the beginning of our discussion. In the remainder of this section we examine several such transformations and discuss a case in which the component transformation functions are dependent on all the color components of the input image and, therefore, cannot be done on an individual color component basis.

6.5.2 Color Complements

The hues directly opposite one another on the *color circle*† of Fig. 6.32 are called *complements*. Our interest in complements stems from the fact that they are analogous to the gray-scale negatives of Section 3.2.1. As in the gray-scale case, color complements are useful for enhancing detail that is embedded in dark regions of a color image—particularly when the regions are dominant in size.

EXAMPLE 6.7: Computing color image complements.

■ Figures 6.33(a) and (c) show the image from Fig. 6.30(a) and its color complement. The RGB transformations used to compute the complement are plotted in Fig. 6.33(b). They are identical to the gray-scale negative

†The color circle originated with Sir Isaac Newton, who in the seventeenth century joined the ends of the color spectrum to form the first color circle.

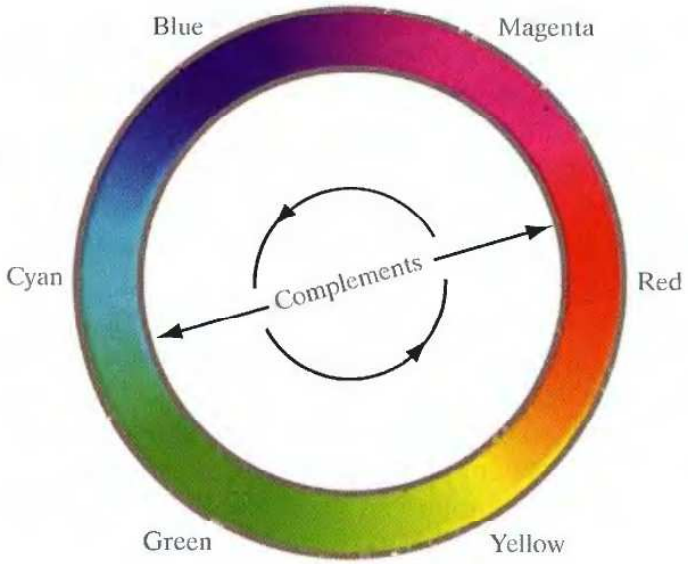
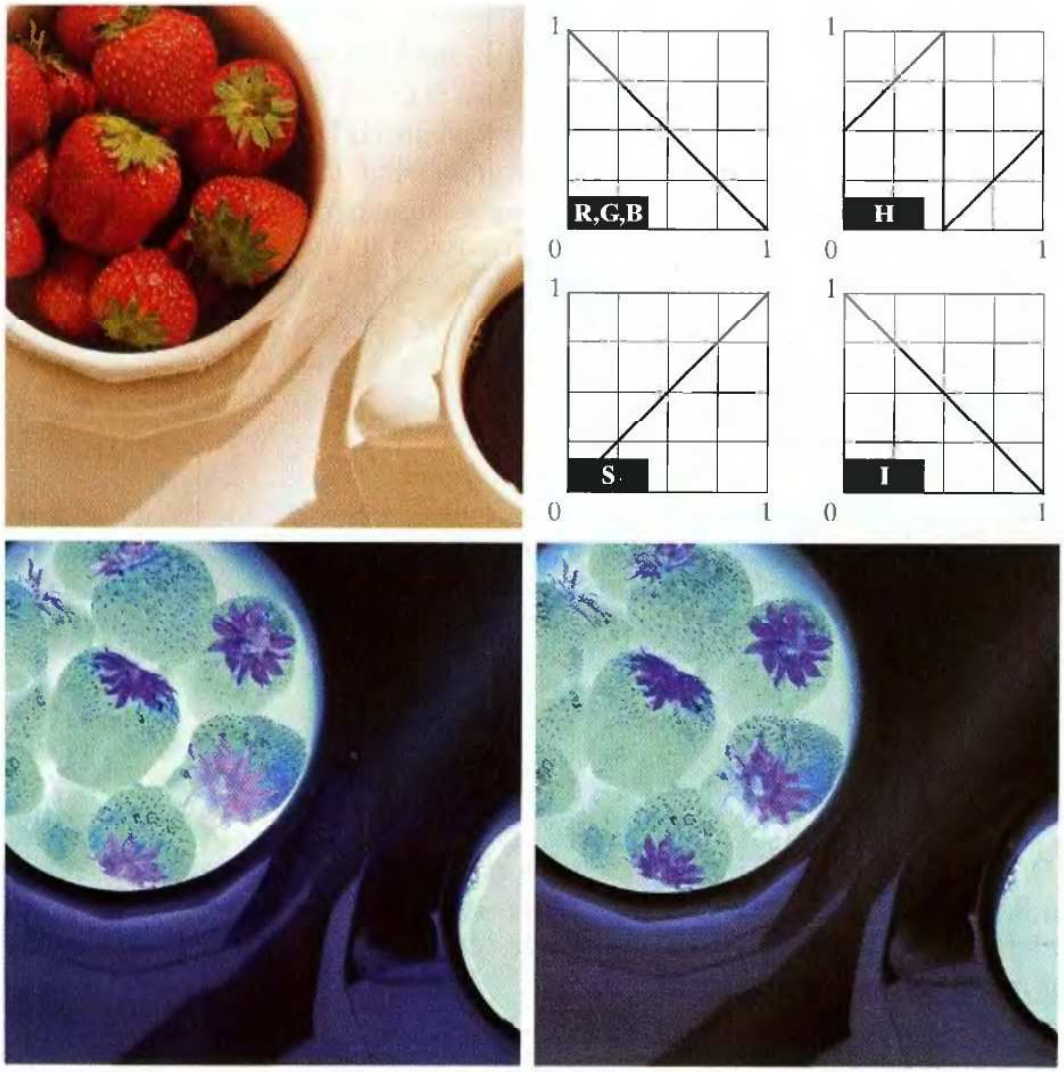


FIGURE 6.32 Complements on the color circle.



a b
c d

FIGURE 6.33 Color complement transformations. (a) Original image. (b) Complement transformation functions. (c) Complement of (a) based on the RGB mapping functions. (d) An approximation of the RGB complement using HSI transformations.

transformation defined in Section 3.2.1. Note that the computed complement is reminiscent of conventional photographic color film negatives. Reds of the original image are replaced by cyans in the complement. When the original image is black, the complement is white, and so on. Each of the hues in the complement image can be predicted from the original image using the color circle of Fig. 6.32. And each of the RGB component transforms involved in the computation of the complement is a function of *only* the corresponding input color component.

Unlike the intensity transformations of Fig. 6.31, the RGB complement transformation functions used in this example do not have a straightforward HSI space equivalent. It is left as an exercise for the reader (see Problem 6.18) to show that the saturation component of the complement cannot be computed from the saturation component of the input image alone. Figure 6.33(d) provides an approximation of the complement using the hue, saturation, and intensity transformations given in Fig. 6.33(b). Note that the saturation component of the input image is unaltered; it is responsible for the visual differences between Figs. 6.33(c) and (d). ■

6.5.3 Color Slicing

Highlighting a specific range of colors in an image is useful for separating objects from their surroundings. The basic idea is either to (1) display the colors of interest so that they stand out from the background or (2) use the region defined by the colors as a mask for further processing. The most straightforward approach is to extend the gray-level slicing techniques of Section 3.2.4. Because a color pixel is an n-dimensional quantity, however, the resulting color transformation functions are more complicated than their gray-scale counterparts in Fig. 3.11. In fact, the required transformations are more complex than the color component transforms considered thus far. This is because all practical color slicing approaches require each pixel's transformed color components to be a function of all n original pixel's color components.

One of the simplest ways to "slice" a color image is to map the colors outside some range of interest to a nonprominent neutral color. If the colors of interest are enclosed by a cube (or *hypercube* for $n > 3$) of width W and centered at a prototypical (e.g., average) color with components $(a_1, a_2, \ldots, a_n)$, the necessary set of transformations is

$$s_i = \begin{cases} 0.5 & \text{if } \left[|r_j - a_j| > \dfrac{W}{2} \right]_{\text{any } 1 \le j \le n} \\ r_i & \text{otherwise} \end{cases}, \qquad i = 1, 2, \ldots, n. \quad (6.5\text{-}7)$$

These transformations highlight the colors around the prototype by forcing all other colors to the midpoint of the reference color space (an arbitrarily chosen neutral point). For the RGB color space, for example, a suitable neutral point is middle gray or color (0.5, 0.5, 0.5).

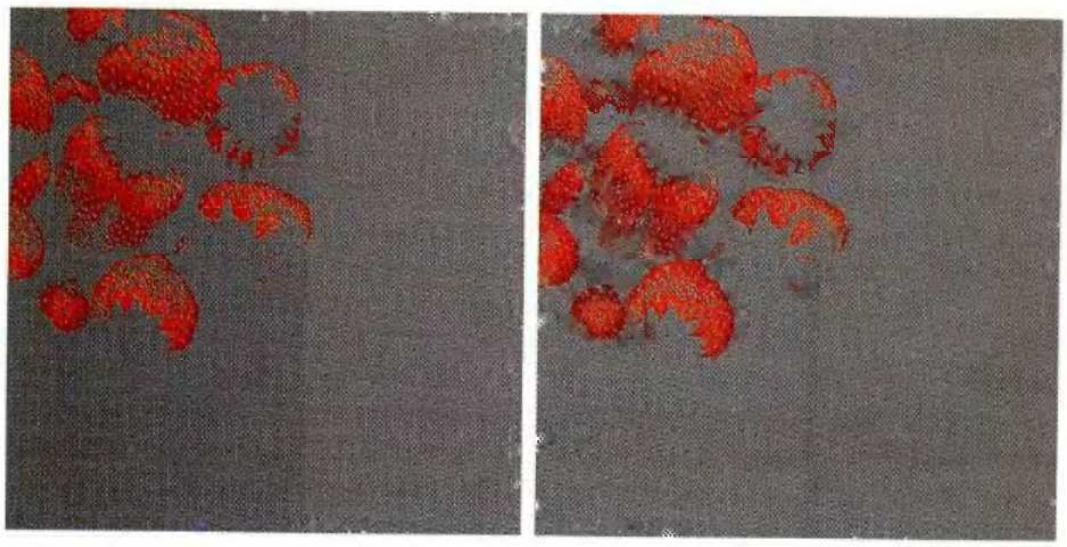
If a sphere is used to specify the colors of interest, Eq. (6.5-7) becomes

$$s_i = \begin{cases} 0.5 & \text{if } \sum_{j=1}^{n} (r_j - a_j)^2 > R_0^2 \\ r_i & \text{otherwise} \end{cases}, \quad i = 1, 2, \ldots, n. \tag{6.5-8}$$

Here, R_0 is the radius of the enclosing sphere (or hypersphere for $n > 3$) and $(a_1, a_2, \ldots, a_n)$ are the components of its center (i.e., the prototypical color). Other useful variations of Eqs. (6.5-7) and (6.5-8) include implementing multiple color prototypes and reducing the intensity of the colors outside the region of interest—rather than setting them to a neutral constant.

EXAMPLE 6.8: An illustration of color slicing.

■ Equations (6.5-7) and (6.5-8) can be used to separate the edible part of the strawberries in Fig. 6.30(a) from the background cups, bowl, coffee, and table. Figures 6.34(a) and (b) show the results of applying both transformations. In each case, a prototype red with RGB color coordinate (0.6863, 0.1608, 0.1922) was selected from the most prominent strawberry; W and R_0 were chosen so that the highlighted region would not expand to undesirable portions of the image. The actual values, $W = 0.2549$ and $R_0 = 0.1765$, were determined interactively. Note that the sphere-based transformation of Eq. (6.5-8) is slightly better, in the sense that it includes more of the strawberries' red areas. A sphere of radius 0.1765 does not completely enclose a cube of width 0.2549 but is itself not completely enclosed by the cube. ■



a b

FIGURE 6.34 Color slicing transformations that detect (a) reds within an RGB cube of width $W = 0.2549$ centered at (0.6863, 0.1608, 0.1922), and (b) reds within an RGB sphere of radius 0.1765 centered at the same point. Pixels outside the cube and sphere were replaced by color (0.5, 0.5, 0.5).

6.5.4 Tone and Color Corrections

Color transformations can be performed on most desktop computers. In conjunction with digital cameras, flatbed scanners, and inkjet printers, they turn a personal computer into a *digital darkroom*—allowing tonal adjustments and color corrections, the mainstays of high-end color reproduction systems, to be performed without the need for traditionally outfitted wet processing (i.e., darkroom) facilities. Although tone and color corrections are useful in other areas of imaging, the focus of the current discussion is on the most common uses—photo enhancement and color reproduction.

The effectiveness of the transformations examined in this section is judged ultimately in print. Since these transformations are developed, refined, and evaluated on monitors, it is necessary to maintain a high degree of color consistency between the monitors used and the eventual output devices. In fact, the colors of the monitor should represent accurately any digitally scanned source images, as well as the final printed output. This is best accomplished with a *device-independent color model* that relates the color gamuts (see Section 6.1) of the monitors and output devices, as well as any other devices being used, to one another. The success of this approach is a function of the quality of the *color profiles* used to map each device to the model and the model itself. The model of choice for many *color management systems* (CMS) is the CIE $L^*a^*b^*$ model, also called CIELAB (CIE [1978], Robertson [1977]). The $L^*a^*b^*$ color components are given by the following equations:

$$L^* = 116 \cdot h\left(\frac{Y}{Y_W}\right) - 16 \tag{6.5-9}$$

$$a^* = 500\left[h\left(\frac{X}{X_W}\right) - h\left(\frac{Y}{Y_W}\right)\right] \tag{6.5-10}$$

$$b^* = 200\left[h\left(\frac{Y}{Y_W}\right) - h\left(\frac{Z}{Z_W}\right)\right] \tag{6.5-11}$$

where

$$h(q) = \begin{cases} \sqrt[3]{q} & q > 0.008856 \\ 7.787q + 16/116 & q < 0.008856 \end{cases} \tag{6.5-12}$$

and X_W, Y_W, and Z_W are reference white tristimulus values—typically the white of a perfectly reflecting diffuser under CIE standard $D65$ illumination (defined by $x = 0.3127$ and $y = 0.3290$ in the CIE chromaticity diagram of Fig. 6.5). The $L^*a^*b^*$ color space is *colorimetric* (i.e., colors perceived as matching are encoded identically), *perceptually uniform* (i.e., color differences among various hues are perceived uniformly—see the classic paper by MacAdams [1942]), and *device independent*. While not a directly displayable format (conversion to another color space is required), its gamut encompasses the entire visible spectrum and can represent accurately the colors of any display, print, or input device. Like the HSI system, the $L^*a^*b^*$ system is an excellent decoupler of intensity (represented by lightness L^*) and color (represented by a^* for red minus green

and b^* for green minus blue), making it useful in both image manipulation (tone and contrast editing) and image compression applications.[†]

The principal benefit of calibrated imaging systems is that they allow tonal and color imbalances to be corrected interactively and independently—that is, in two sequential operations. Before color irregularities, like over- and undersaturated colors, are resolved, problems involving the image's tonal range are corrected. The *tonal range* of an image, also called its *key type*, refers to its general distribution of color intensities. Most of the information in *high-key* images is concentrated at high (or light) intensities; the colors of *low-key* images are located predominantly at low intensities; *middle-key* images lie in between. As in the monochrome case, it is often desirable to distribute the intensities of a color image equally between the highlights and the shadows. The following examples demonstrate a variety of color transformations for the correction of tonal and color imbalances.

EXAMPLE 6.9: Tonal transformations.

■ Transformations for modifying image tones normally are selected interactively. The idea is to adjust experimentally the image's brightness and contrast to provide maximum detail over a suitable range of intensities. The colors themselves are not changed. In the RGB and CMY(K) spaces, this means mapping all three (or four) color components with the same transformation function; in the HSI color space, only the intensity component is modified.

Figure 6.35 shows typical transformations used for correcting three common tonal imbalances—flat, light, and dark images. The S-shaped curve in the first row of the figure is ideal for boosting contrast [see Fig. 3.2(a)]. Its midpoint is anchored so that highlight and shadow areas can be lightened and darkened, respectively. (The inverse of this curve can be used to correct excessive contrast.) The transformations in the second and third rows of the figure correct light and dark images and are reminiscent of the power-law transformations in Fig. 3.6. Although the color components are discrete, as are the actual transformation functions, the transformation functions themselves are displayed and manipulated as continuous quantities—typically constructed from piecewise linear or higher order (for smoother mappings) polynomials. Note that the keys of the images in Fig. 6.35 are directly observable; they could also be determined using the histograms of the images' color components. ■

EXAMPLE 6.10: Color balancing.

■ After the tonal characteristics of an image have been properly established, any color imbalances can be addressed. Although color imbalances can be determined objectively by analyzing—with a color spectrometer—a known color in an image, accurate visual assessments are possible when white areas, where the RGB or CMY(K) components should be equal, are present. As can be seen in Fig. 6.36, skin tones also are excellent subjects for visual color assessments because humans are highly perceptive of proper skin color. Vivid colors, such as bright red objects, are of little value when it comes to visual color assessment.

[†] Studies indicate that the degree to which the luminance (lightness) information is separated from the color information in $L^*a^*b^*$ is greater than in other color models—such as CIELUV, YIQ, YUV, YCC, and XYZ (Kasson and Plouffe [1992]).

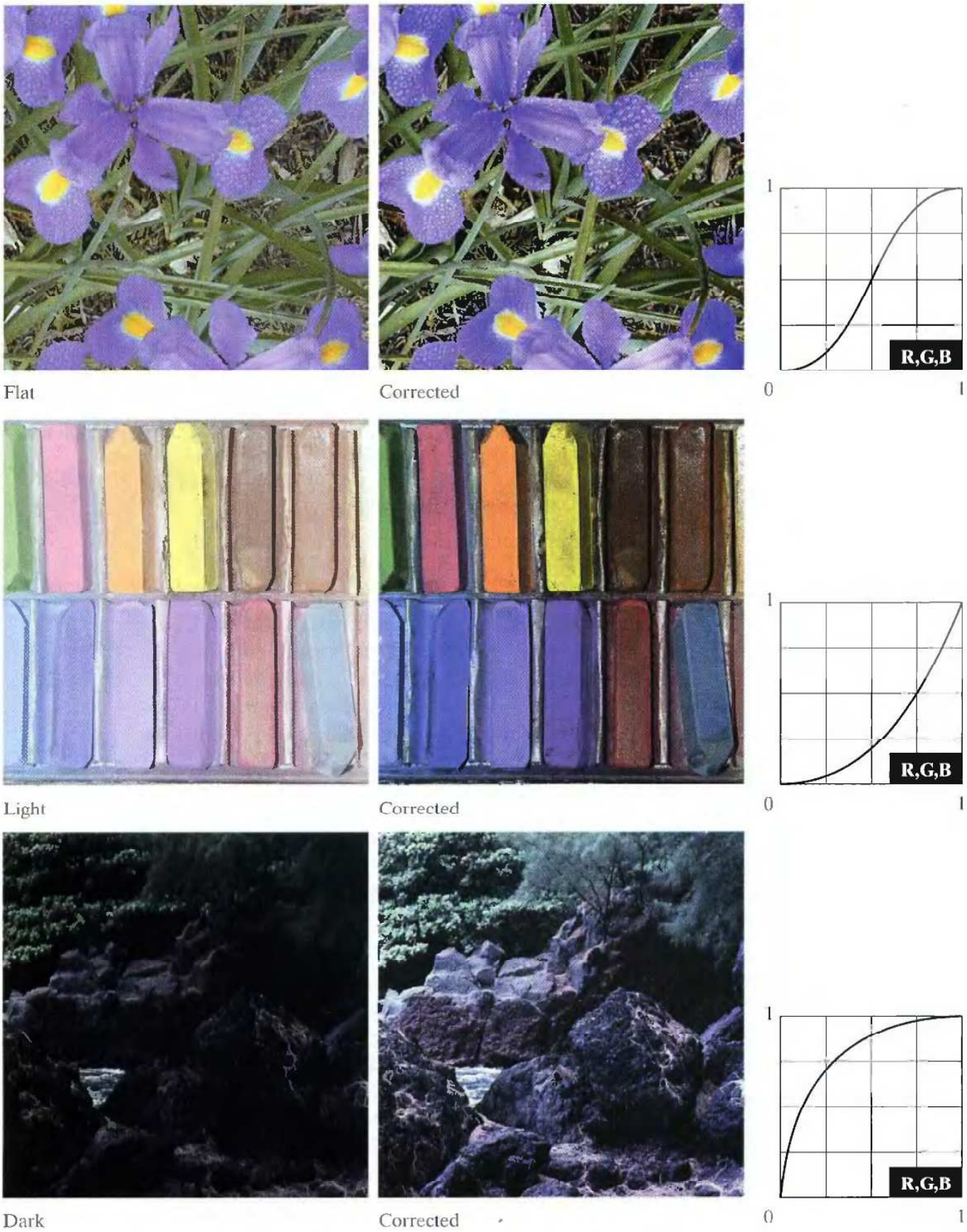
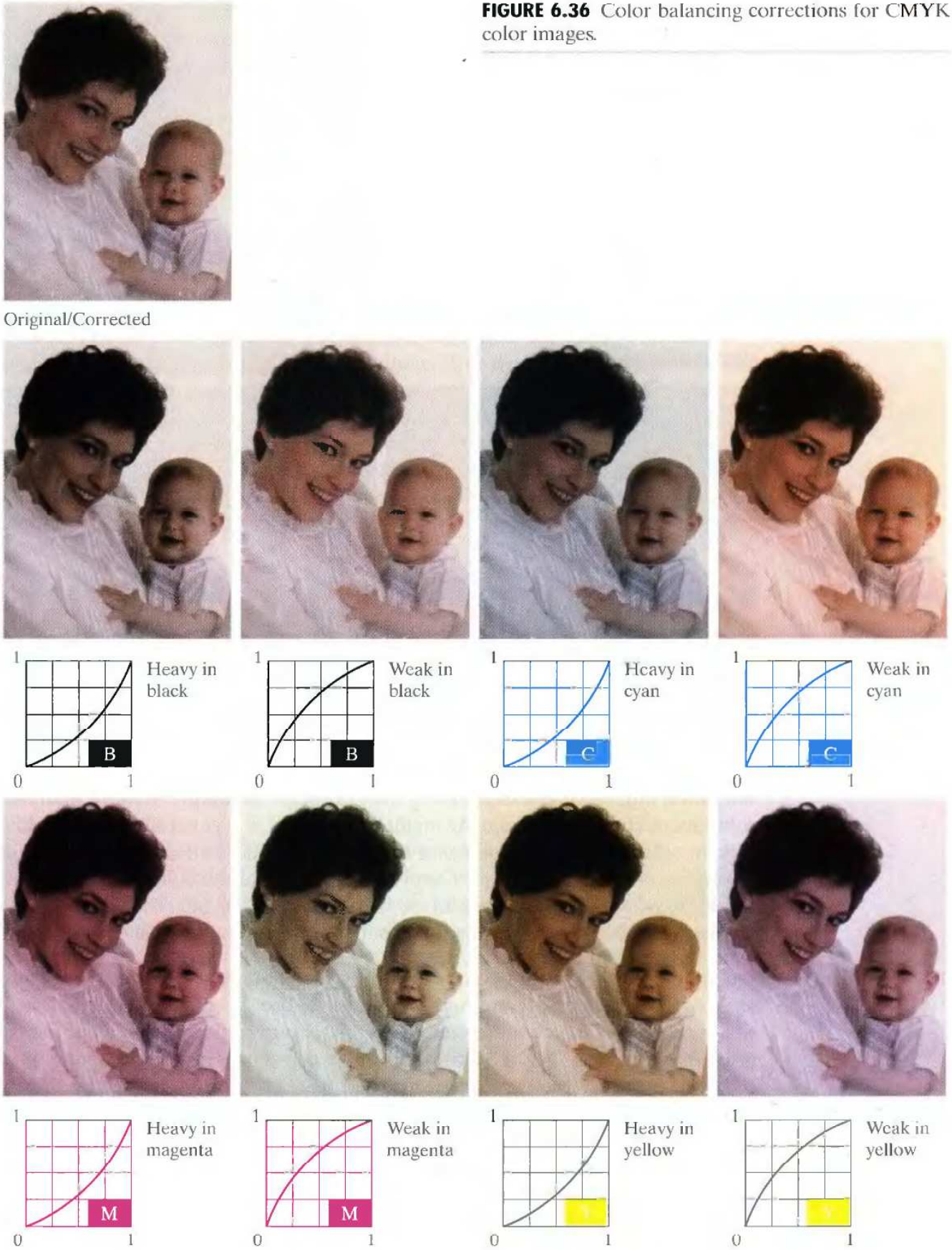


FIGURE 6.35 Tonal corrections for flat, light (high key), and dark (low key) color images. Adjusting the red, green, and blue components equally does not alter the image hues.

FIGURE 6.36 Color balancing corrections for CMYK color images.



When a color imbalance is noted, there are a variety of ways to correct it. When adjusting the color components of an image, it is important to realize that every action affects the overall color balance of the image. That is, the perception of one color is affected by its surrounding colors. Nevertheless, the color wheel of Fig. 6.32 can be used to predict how one color component will affect others. Based on the color wheel, for example, the proportion of any color can be increased by decreasing the amount of the opposite (or complementary) color in the image. Similarly, it can be increased by raising the proportion of the two immediately adjacent colors or decreasing the percentage of the two colors adjacent to the complement. Suppose, for instance, that there is an abundance of magenta in an RGB image. It can be decreased by (1) removing both red and blue or (2) adding green.

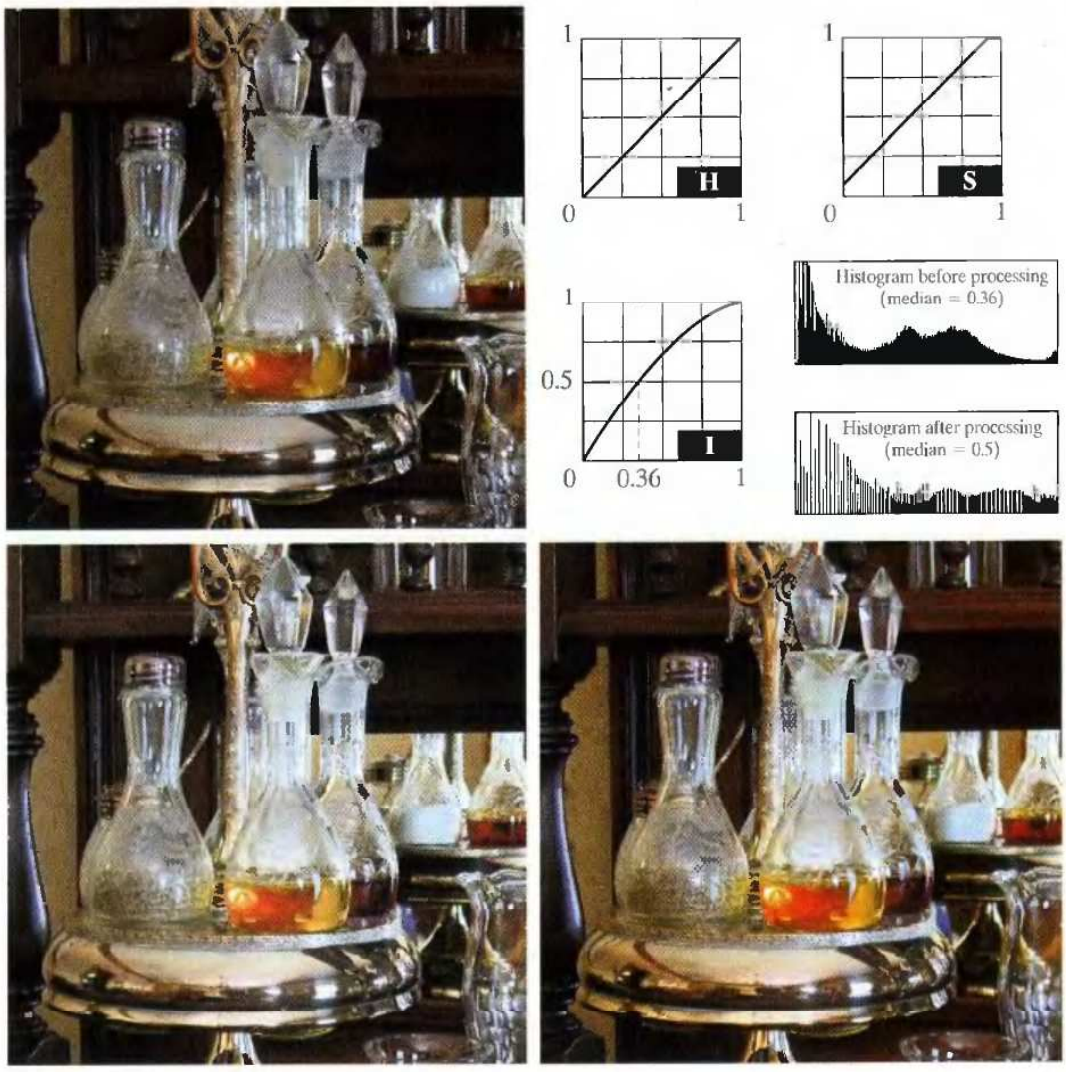
Figure 6.36 shows the transformations used to correct simple CMYK output imbalances. Note that the transformations depicted are the functions required for correcting the images; the inverses of these functions were used to generate the associated color imbalances. Together, the images are analogous to a color ring-around print of a darkroom environment and are useful as a reference tool for identifying color printing problems. Note, for example, that too much red can be due to excessive magenta (per the bottom left image) or too little cyan (as shown in the rightmost image of the second row). ■

6.5.5 Histogram Processing

Unlike the interactive enhancement approaches of the previous section, the gray-level histogram processing transformations of Section 3.3 can be applied to color images in an automated way. Recall that histogram equalization automatically determines a transformation that seeks to produce an image with a uniform histogram of intensity values. In the case of monochrome images, it was shown (see Fig. 3.17) to be reasonably successful at handling low-, high-, and middle-key images. Since color images are composed of multiple components, however, consideration must be given to adapting the gray-scale technique to more than one component and/or histogram. As might be expected, it is generally unwise to histogram equalize the components of a color image independently. This results in erroneous color. A more logical approach is to spread the color intensities uniformly, leaving the colors themselves (e.g., hues) unchanged. The following example shows that the HSI color space is ideally suited to this type of approach.

EXAMPLE 6.11: Histogram equalization in the HSI color space.

■ Figure 6.37(a) shows a color image of a caster stand containing cruets and shakers whose intensity component spans the entire (normalized) range of possible values, [0, 1]. As can be seen in the histogram of its intensity component prior to processing [Fig. 6.37(b)], the image contains a large number of dark colors that reduce the median intensity to 0.36. Histogram equalizing the intensity component, without altering the hue and saturation, resulted in the image shown in Fig. 6.37(c). Note that the overall image is significantly brighter and that several moldings and the grain of the wooden table on which the caster is sitting are now visible. Figure 6.37(b) shows the intensity histogram of the new image, as well as the intensity transformation used to equalize the intensity component [see Eq. (3.3-8)].



a b
c d

FIGURE 6.37 Histogram equalization (followed by saturation adjustment) in the HSI color space.

Although the intensity equalization process did not alter the values of hue and saturation of the image, it did impact the overall color perception. Note, in particular, the loss of vibrancy in the oil and vinegar in the cruets. Figure 6.37(d) shows the result of correcting this partially by increasing the image's saturation component, subsequent to histogram equalization, using the transformation in Fig. 6.37(b). This type of adjustment is common when working with the intensity component in HSI space because changes in intensity usually affect the relative appearance of colors in an image. ■

6.6 Smoothing and Sharpening

The next step beyond transforming each pixel of a color image without regard to its neighbors (as in the previous section) is to modify its value based on the characteristics of the surrounding pixels. In this section, the basics of this type of neighborhood processing are illustrated within the context of color image smoothing and sharpening.

6.6.1 Color Image Smoothing

With reference to Fig. 6.29(a) and the discussion in Section 3.6, gray-scale image smoothing can be viewed as a spatial filtering operation in which the coefficients of the filtering mask are all 1's. As the mask is slid across the image to be smoothed, each pixel is replaced by the average of the pixels in the neighborhood defined by the mask. As can be seen in Fig. 6.29(b), this concept is easily extended to the processing of full-color images. The principal difference is that instead of scalar gray-level values we must deal with component vectors of the form given in Eq. (6.4-2).

Let S_{xy} denote the set of coordinates defining a neighborhood centered at (x, y) in an RGB color image. The average of the RGB component vectors in this neighborhood is

$$\bar{\mathbf{c}}(x, y) = \frac{1}{K} \sum_{(x,y)\in S_{xy}} \mathbf{c}(x, y). \tag{6.6-1}$$

It follows from Eq. (6.4-2) and the properties of vector addition that

See inside front cover
Consult the book web site for a brief review of vectors and matrices.

$$\bar{\mathbf{c}}(x, y) = \begin{bmatrix} \dfrac{1}{K} \displaystyle\sum_{(x,y)\in S_{xy}} R(x, y) \\ \dfrac{1}{K} \displaystyle\sum_{(x,y)\in S_{xy}} G(x, y) \\ \dfrac{1}{K} \displaystyle\sum_{(x,y)\in S_{xy}} B(x, y) \end{bmatrix}. \tag{6.6-2}$$

We recognize the components of this vector as the scalar images that would be obtained by independently smoothing each plane of the starting RGB image using conventional gray-scale neighborhood processing. Thus, we conclude that smoothing by neighborhood averaging can be carried out on a per-color-plane basis. The result is the same as when the averaging is performed using RGB color vectors.

EXAMPLE 6.12: Color image smoothing by neighborhood averaging.

■ Consider the color image shown in Fig. 6.38(a). The red, green, and blue planes of this image are depicted in Figs. 6.38(b) through (d). Figures 6.39(a) through (c) show the image's HSI components. In accordance with the discussion in the preceding paragraph, we can smooth the RGB image in Fig. 6.38 using the 5×5 gray-level averaging mask of Section 3.6. We simply smooth independently each of the RGB color planes and then combine the processed planes to form a smoothed full-color result. The image so computed is shown in Fig. 6.40(a). Note that it appears as we would expect from the discussion and examples in Section 3.6.

In Section 6.2 it was noted that an important advantage of the HSI color model is that it decouples intensity (closely related to gray scale) and color information. This makes it suitable for many gray-scale processing techniques and suggests that it might be more efficient to smooth only the intensity component of the HSI representation in Fig. 6.39. To illustrate the merits and/or consequences of this approach, we next smooth only the intensity component

a b
c d

FIGURE 6.38 (a) RGB image. (b) Red component image. (c) Green component. (d) Blue component.

a b c

FIGURE 6.39 HSI components of the RGB color image in Fig. 6.38(a). (a) Hue. (b) Saturation. (c) Intensity.

a b c

FIGURE 6.40 Image smoothing with a 5×5 averaging mask. (a) Result of processing each RGB component image. (b) Result of processing the intensity component of the HSI image and converting to RGB. (c) Difference between the two results.

(leaving the hue and saturation components unmodified) and convert the processed result to an RGB image for display. The smoothed color image is shown in Fig. 6.40(b). Note that it is similar to Fig. 6.40(a), but, as can be seen from the difference image in Fig. 640(c), is not identical. This is due to the fact that the average of two pixels of differing color is a mixture of the two colors, not either of the original colors. By smoothing only the intensity image, the pixels in Fig. 6.40(b) maintain their original hue and saturation—and thus their original color. Finally, we note that the difference (between the smoothed results in this example) would increase as the size of the smoothing mask increases. ■

6.6.2 Color Image Sharpening

In this section we consider image sharpening using the Laplacian (see Section 3.7.2). From vector analysis, we know that the Laplacian of a vector is defined as a vector whose components are equal to the Laplacian of the individual scalar components of the input vector. In the RGB color system, the Laplacian of vector $\mathbf{c}$ in Eq. (6.4-2) is

$$\nabla^2[\mathbf{c}(x, y)] = \begin{bmatrix} \nabla^2 R(x, y) \\ \nabla^2 G(x, y) \\ \nabla^2 B(x, y) \end{bmatrix}, \tag{6.6-3}$$

which, as in the previous section, tells us that we can compute the Laplacian of a full-color image by computing the Laplacian of each component image separately.

EXAMPLE 6.13: Sharpening with the Laplacian.

■ Figure 6.41(a) was obtained using Eq. (3.7-6) to compute the Laplacians of the RGB component images in Fig. 6.38 and combining them to produce the sharpened full-color result. Figure 6.41(b) shows a similarly sharpened image based on the HSI components in Fig. 6.39. This result was generated by combining the Laplacian of the intensity component with the unchanged hue and saturation

a b c

FIGURE 6.41 Image sharpening with the Laplacian. (a) Result of processing each RGB channel. (b) Result of processing the intensity component and converting to RGB. (c) Difference between the two results.

components. The difference between the RGB and HSI-based results is shown in Fig. 6.41(c); it results from the same factors explained in Example 6.12. ■

6.7 Color Segmentation

Segmentation is a process that partitions an image into regions. Although segmentation is the topic of Chapter 10, we consider color segmentation briefly here for the sake of continuity. The reader will have no difficulty in following the discussion.

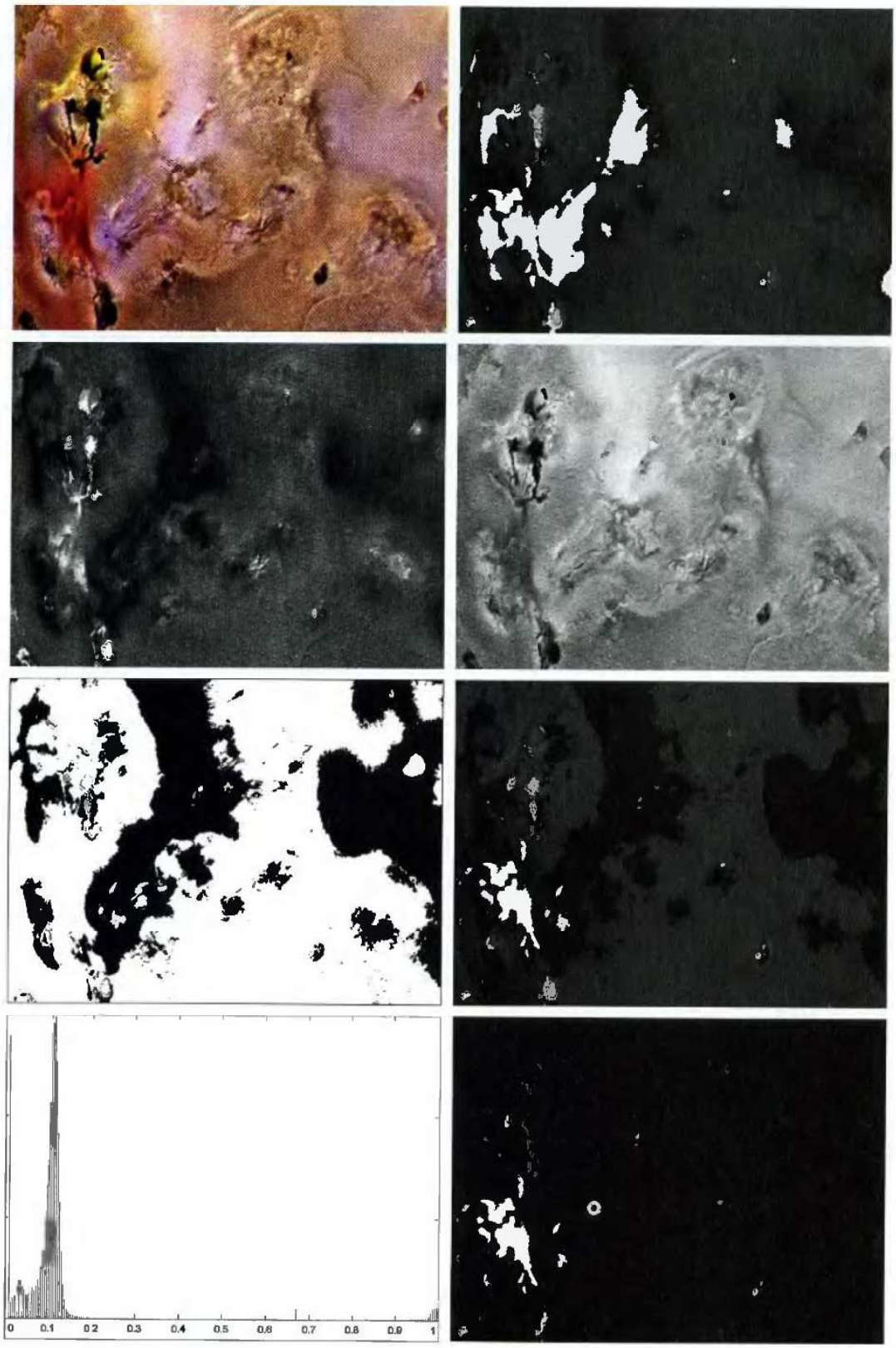
6.7.1 Segmentation in HSI Color Space

If we wish to segment an image based on color, and, in addition, we want to carry out the process on individual planes, it is natural to think first of the HSI space because color is conveniently represented in the hue image. Typically, saturation is used as a masking image in order to isolate further regions of interest in the hue image. The intensity image is used less frequently for segmentation of color images because it carries no color information. The following example is typical of how segmentation is performed in the HSI system.

EXAMPLE 6.14: Segmentation in HSI space.

■ Suppose that it is of interest to segment the reddish region in the lower left of the image in Fig. 6.42(a). Although it was generated by pseudocolor methods, this image can be processed (segmented) as a full-color image without loss of generality. Figures 6.42(b) through (d) are its HSI component images. Note by comparing Fig. 6.42(a) and (b) that the region in which we are interested has relatively high values of hue, indicating that the colors are on the blue-magenta side of red (see Fig. 6.13). Figure 6.42(e) shows a binary mask generated by thresholding the saturation image with a threshold equal to 10% of the maximum value in the saturation image. Any pixel value greater than the threshold was set to 1 (white). All others were set to 0 (black).

Figure 6.42(f) is the product of the mask with the hue image, and Fig. 6.42(g) is the histogram of the product image (note that the gray scale is in the range [0, 1]).



a b
c d
e f
g h

FIGURE 6.42 Image segmentation in HSI space. (a) Original. (b) Hue. (c) Saturation. (d) Intensity. (e) Binary saturation mask (black = 0). (f) Product of (b) and (e). (g) Histogram of (f). (h) Segmentation of red components in (a).

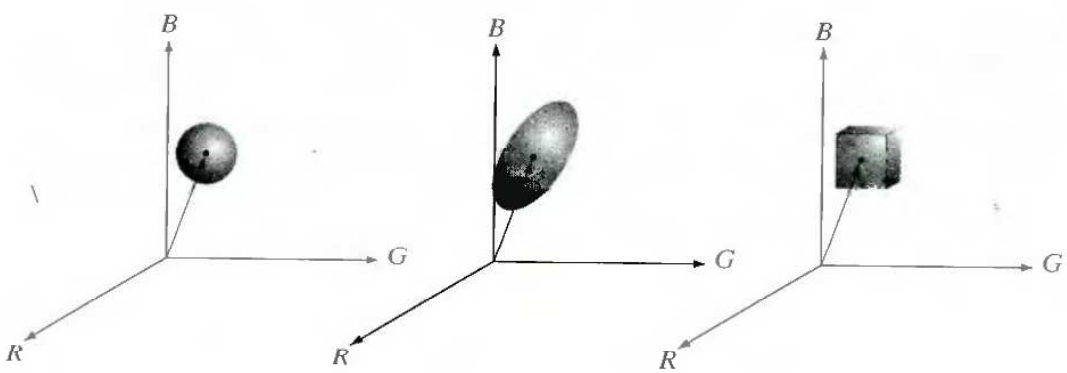
We see in the histogram that high values (which are the values of interest) are grouped at the very high end of the gray scale, near 1.0. The result of thresholding the product image with threshold value of 0.9 resulted in the binary image shown in Fig. 6.42(h). The spatial location of the white points in this image identifies the points in the original image that have the reddish hue of interest. This was far from a perfect segmentation because there are points in the original image that we certainly would say have a reddish hue, but that were not identified by this segmentation method. However, it can be shown by experimentation that the regions shown in white in Fig. 6.42(h) are about the best this method can do in identifying the reddish components of the original image. The segmentation method discussed in the following section is capable of yielding considerably better results. ■

6.7.2 Segmentation in RGB Vector Space

Although, as mentioned numerous times in this chapter, working in HSI space is more intuitive, segmentation is one area in which better results generally are obtained by using RGB color vectors. The approach is straightforward. Suppose that the objective is to segment objects of a specified color range in an RGB image. Given a set of sample color points representative of the colors of interest, we obtain an estimate of the "average" color that we wish to segment. Let this average color be denoted by the RGB vector **a**. The objective of segmentation is to classify each RGB pixel in a given image as having a color in the specified range or not. In order to perform this comparison, it is necessary to have a measure of similarity. One of the simplest measures is the Euclidean distance. Let **z** denote an arbitrary point in RGB space. We say that **z** is similar to **a** if the distance between them is less than a specified threshold, D_0. The Euclidean distance between **z** and **a** is given by

$$\begin{aligned} D(\mathbf{z}, \mathbf{a}) &= \|\mathbf{z} - \mathbf{a}\| \\ &= \left[(\mathbf{z} - \mathbf{a})^T(\mathbf{z} - \mathbf{a})\right]^{\frac{1}{2}} \qquad (6.7\text{-}1) \\ &= \left[(z_R - a_R)^2 + (z_G - a_G)^2 + (z_B - a_B)^2\right]^{\frac{1}{2}} \end{aligned}$$

where the subscripts R, G, and B, denote the RGB components of vectors **a** and **z**. The locus of points such that $D(\mathbf{z}, \mathbf{a}) \leq D_0$ is a solid sphere of radius D_0, as illustrated in Fig. 6.43(a). Points contained within or on the surface of the



a b c

FIGURE 6.43 Three approaches for enclosing data regions for RGB vector segmentation.

sphere satisfy the specified color criterion; points outside the sphere do not. Coding these two sets of points in the image with, say, black and white, produces a binary segmented image.

A useful generalization of Eq. (6.7-1) is a distance measure of the form

$$D(\mathbf{z}, \mathbf{a}) = \left[(\mathbf{z} - \mathbf{a})^T \mathbf{C}^{-1} (\mathbf{z} - \mathbf{a})\right]^{\frac{1}{2}} \tag{6.7-2}$$

where $\mathbf{C}$ is the covariance matrix[†] of the samples representative of the color we wish to segment. The locus of points such that $D(\mathbf{z}, \mathbf{a}) \leq D_0$ describes a solid 3-D elliptical body [Fig. 6.43(b)] with the important property that its principal axes are oriented in the direction of maximum data spread. When $\mathbf{C} = \mathbf{I}$, the 3×3 identity matrix, Eq. (6.7-2) reduces to Eq. (6.7-1). Segmentation is as described in the preceding paragraph.

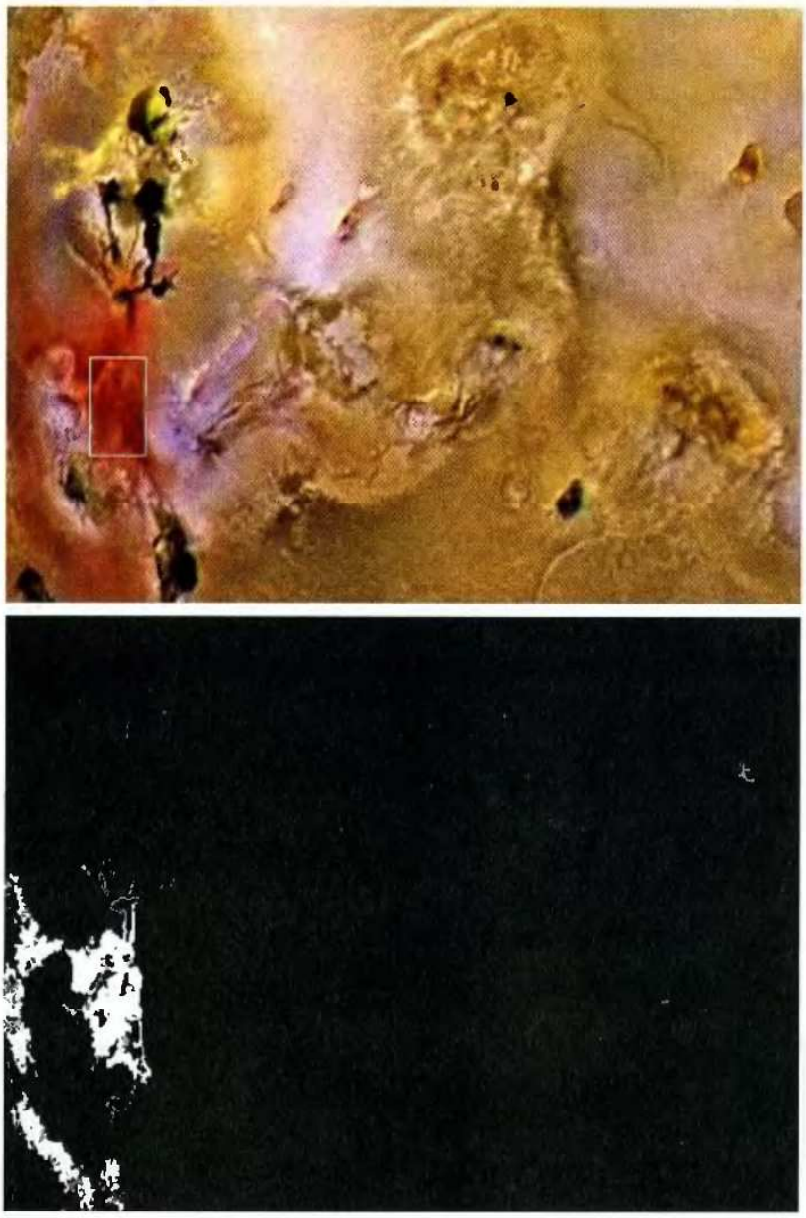
Because distances are positive and monotonic, we can work with the distance squared instead, thus avoiding root computations. However, implementing Eq. (6.7-1) or (6.7-2) is computationally expensive for images of practical size, even if the square roots are not computed. A compromise is to use a bounding box, as illustrated in Fig. 6.43(c). In this approach, the box is centered on $\mathbf{a}$, and its dimensions along each of the color axes is chosen proportional to the standard deviation of the samples along each of the axis. Computation of the standard deviations is done only once using sample color data.

Given an arbitrary color point, we segment it by determining whether or not it is on the surface or inside the box, as with the distance formulations. However, determining whether a color point is inside or outside the box is much simpler computationally when compared to a spherical or elliptical enclosure. Note that the preceding discussion is a generalization of the method introduced in Section 6.5.3 in connection with color slicing.

EXAMPLE 6.15: Color image segmentation in RGB space.

■ The rectangular region shown Fig. 6.44(a) contains samples of reddish colors we wish to segment out of the color image. This is the same problem we considered in Example 6.14 using hue, but here we approach the problem using RGB color vectors. The approach followed was to compute the mean vector $\mathbf{a}$ using the color points contained within the rectangle in Fig. 6.44(a), and then to compute the standard deviation of the red, green, and blue values of those samples. A box was centered at $\mathbf{a}$, and its dimensions along each of the RGB axes were selected as 1.25 times the standard deviation of the data along the corresponding axis. For example, let σ_R denote the standard deviation of the red components of the sample points. Then the dimensions of the box along the R-axis extended from $(a_R - 1.25\sigma_R)$ to $(a_R + 1.25\sigma_R)$, where a_R denotes the red component of average vector $\mathbf{a}$. The result of coding each point in the entire color image as white if it was on the surface or inside the box, and as black otherwise, is shown in Fig. 6.44(b). Note how the segmented region was generalized from the color samples enclosed by the rectangle. In fact, by comparing

[†] Computation of the covariance matrix of a set of vector samples is discussed in detail in Section 11.4.



a
b

FIGURE 6.44 Segmentation in RGB space. (a) Original image with colors of interest shown enclosed by a rectangle. (b) Result of segmentation in RGB vector space. Compare with Fig. 6.42(h).

Figs. 6.44(b) and. 6.42(h), we see that segmentation in the RGB vector space yielded results that are much more accurate, in the sense that they correspond much more closely with what we would define as "reddish" points in the original color image. ■

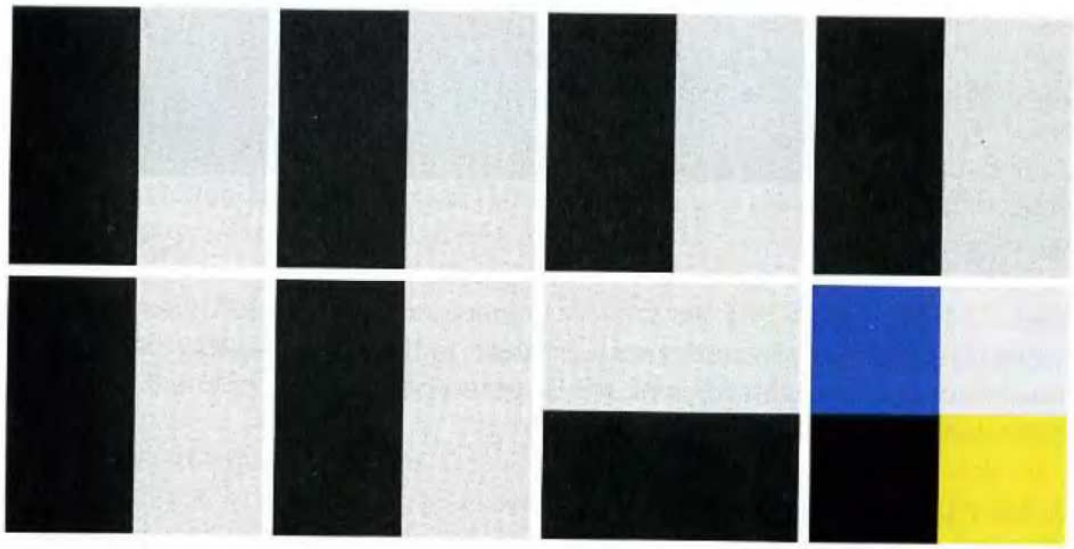
6.7.3 Color Edge Detection

As discussed in Chapter 10, edge detection is an important tool for image segmentation. In this section, we are interested in the issue of computing edges on an individual-image basis versus computing edges directly in color vector space. The details of edge-based segmentation are given in Section 10.1.3.

Edge detection by gradient operators was introduced in Section 3.7.3 in connection with edge enhancement. Unfortunately, the gradient discussed in Section 3.7.3 is not defined for vector quantities. Thus, we know immediately that computing the gradient on individual images and then using the results to form a color image will lead to erroneous results. A simple example will help illustrate the reason why.

Consider the two $M \times M$ color images (M odd) in Fig. 6.45(d) and (h), composed of the three component images in Figs. 6.45(a) through (c) and 6.45(e) through (g), respectively. If, for example, we compute the gradient image of each of the component images [see Eq. (3.7-13)] and add the results to form the two corresponding RGB gradient images, the value of the gradient at point $[(M+1)/2, (M+1)/2]$ would be the same in both cases. Intuitively, we would expect the gradient at that point to be stronger for the image in Fig. 6.45(d) because the edges of the R, G, and B images are in the same direction in that image, as opposed to the image in Fig. 6.45(h), in which only two of the edges are in the same direction. Thus we see from this simple example that processing the three individual planes to form a composite gradient image can yield erroneous results. If the problem is one of just detecting edges, then the individual-component approach usually yields acceptable results. If accuracy is an issue, however, then obviously we need a new definition of the gradient applicable to vector quantities. We discuss next a method proposed by Di Zenzo [1986] for doing this.

The problem at hand is to define the gradient (magnitude and direction) of the vector $\mathbf{c}$ in Eq. (6.4-2) at any point (x, y). As was just mentioned, the gradient we studied in Section 3.7.3 is applicable to a *scalar* function $f(x, y)$; it is



a b c d
e f g h

FIGURE 6.45 (a)–(c) R, G, and B component images and (d) resulting RGB color image. (f)–(g) R, G, and B component images and (h) resulting RGB color image.

not applicable to vector functions. The following is one of the various ways in which we can extend the concept of a gradient to vector functions. Recall that for a scalar function $f(x, y)$, the gradient is a vector pointing in the direction of maximum rate of change of f at coordinates (x, y).

Let $\mathbf{r}$, $\mathbf{g}$, and $\mathbf{b}$ be unit vectors along the R, G, and B axis of RGB color space (Fig. 6.7), and define the vectors

$$\mathbf{u} = \frac{\partial R}{\partial x}\mathbf{r} + \frac{\partial G}{\partial x}\mathbf{g} + \frac{\partial B}{\partial x}\mathbf{b} \tag{6.7-3}$$

and

$$\mathbf{v} = \frac{\partial R}{\partial y}\mathbf{r} + \frac{\partial G}{\partial y}\mathbf{g} + \frac{\partial B}{\partial y}\mathbf{b}. \tag{6.7-4}$$

Let the quantities g_{xx}, g_{yy}, and g_{xy} be defined in terms of the dot product of these vectors, as follows:

$$g_{xx} = \mathbf{u} \cdot \mathbf{u} = \mathbf{u}^T\mathbf{u} = \left|\frac{\partial R}{\partial x}\right|^2 + \left|\frac{\partial G}{\partial x}\right|^2 + \left|\frac{\partial B}{\partial x}\right|^2 \tag{6.7-5}$$

$$g_{yy} = \mathbf{v} \cdot \mathbf{v} = \mathbf{v}^T\mathbf{v} = \left|\frac{\partial R}{\partial y}\right|^2 + \left|\frac{\partial G}{\partial y}\right|^2 + \left|\frac{\partial B}{\partial y}\right|^2 \tag{6.7-6}$$

and

$$g_{xy} = \mathbf{u} \cdot \mathbf{v} = \mathbf{u}^T\mathbf{v} = \frac{\partial R}{\partial x}\frac{\partial R}{\partial y} + \frac{\partial G}{\partial x}\frac{\partial G}{\partial y} + \frac{\partial B}{\partial x}\frac{\partial B}{\partial y}. \tag{6.7-7}$$

Keep in mind that R, G, and B, and consequently the g's, are functions of x and y. Using this notation, it can be shown (Di Zenzo [1986]) that the direction of maximum rate of change of $\mathbf{c}(x, y)$ is given by the angle

$$\theta = \frac{1}{2}\tan^{-1}\left[\frac{2g_{xy}}{(g_{xx} - g_{yy})}\right] \tag{6.7-8}$$

and that the value of the rate of change at (x, y), in the direction of θ, is given by

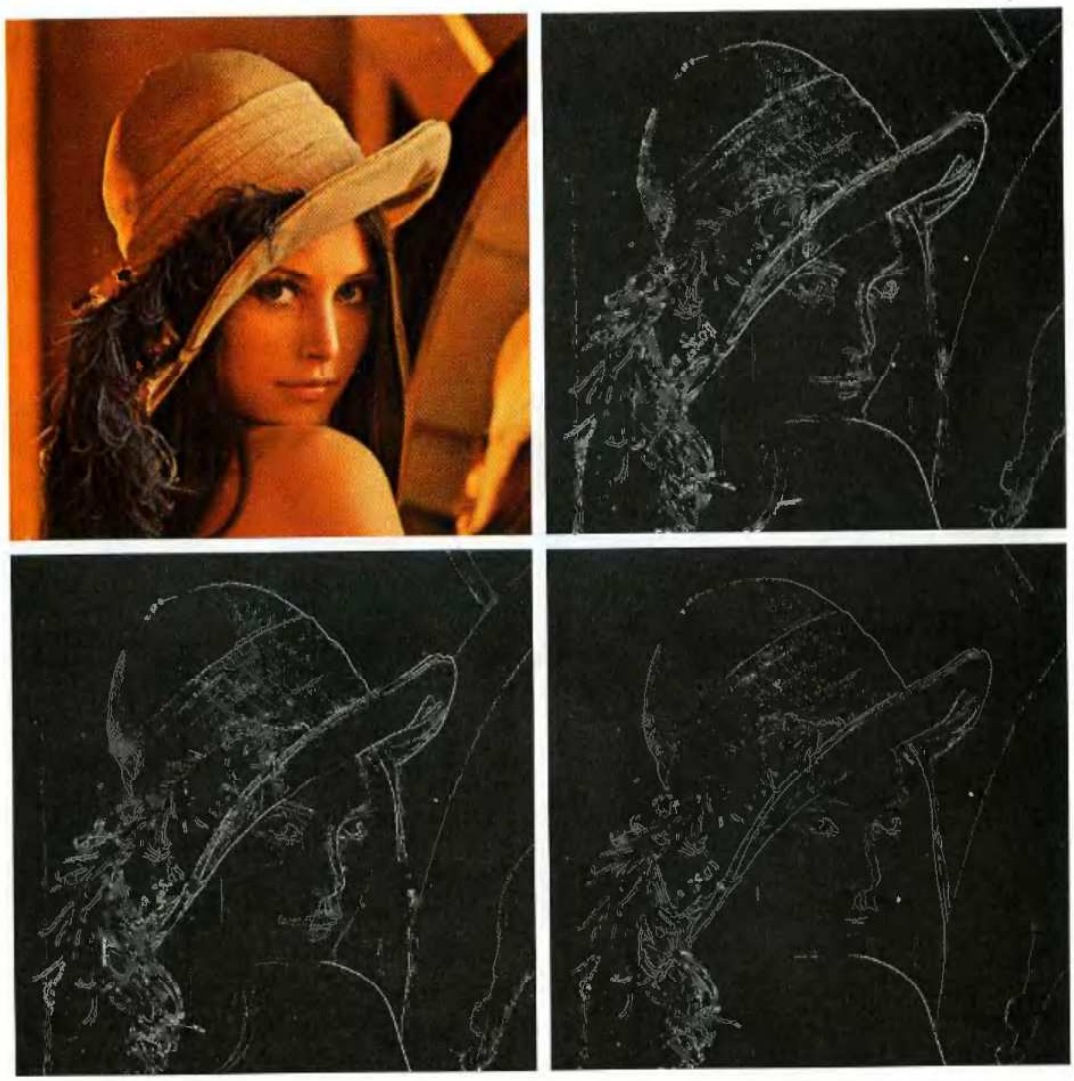
$$F(\theta) = \left\{\frac{1}{2}\left[(g_{xx} + g_{yy}) + (g_{xx} - g_{yy})\cos 2\theta + 2g_{xy}\sin 2\theta\right]\right\}^{\frac{1}{2}}. \tag{6.7-9}$$

Because $\tan(\alpha) = \tan(\alpha \pm \pi)$, if θ_0 is a solution to Eq. (6.7-8), so is $\theta_0 \pm \pi/2$. Furthermore, $F(\theta) = F(\theta + \pi)$, so F needs to be computed only for values of θ in the half-open interval $[0, \pi)$. The fact that Eq. (6.7-8) provides two values 90° apart means that this equation associates with each point (x, y) a pair of orthogonal directions. Along one of those directions F is maximum, and it is minimum along the other. The derivation of these results is rather lengthy, and we would gain little in terms of the fundamental objective of our current discussion by detailing it here. The interested reader should consult the paper

by Di Zenzo [1986] for details. The partial derivatives required for implementing Eqs. (6.7-5) through (6.7-7) can be computed using, for example, the Sobel operators discussed in Section 3.7.3.

EXAMPLE 6.16: Edge detection in vector space.

■ Figure 6.46(b) is the gradient of the image in Fig. 6.46(a), obtained using the vector method just discussed. Figure 6.46(c) shows the image obtained by computing the gradient of each RGB component image and forming a composite gradient image by adding the corresponding values of the three component images at each coordinate (x, y). The edge detail of the vector gradient image is more complete than the detail in the individual-plane gradient image in Fig. 6.46(c); for example, see the detail around the subject's right eye. The image in Fig. 6.46(d) shows the difference between the two gradient images at each point (x, y). It is important to note that both approaches yielded reasonable results. Whether the extra detail in Fig. 6.46(b) is worth the added computational burden (as opposed to implementation of the Sobel operators, which



a b
c d

FIGURE 6.46 (a) RGB image. (b) Gradient computed in RGB color vector space. (c) Gradients computed on a per-image basis and then added. (d) Difference between (b) and (c).

a b c

FIGURE 6.47 Component gradient images of the color image in Fig. 6.46. (a) Red component, (b) green component, and (c) blue component. These three images were added and scaled to produce the image in Fig. 6.46(c).

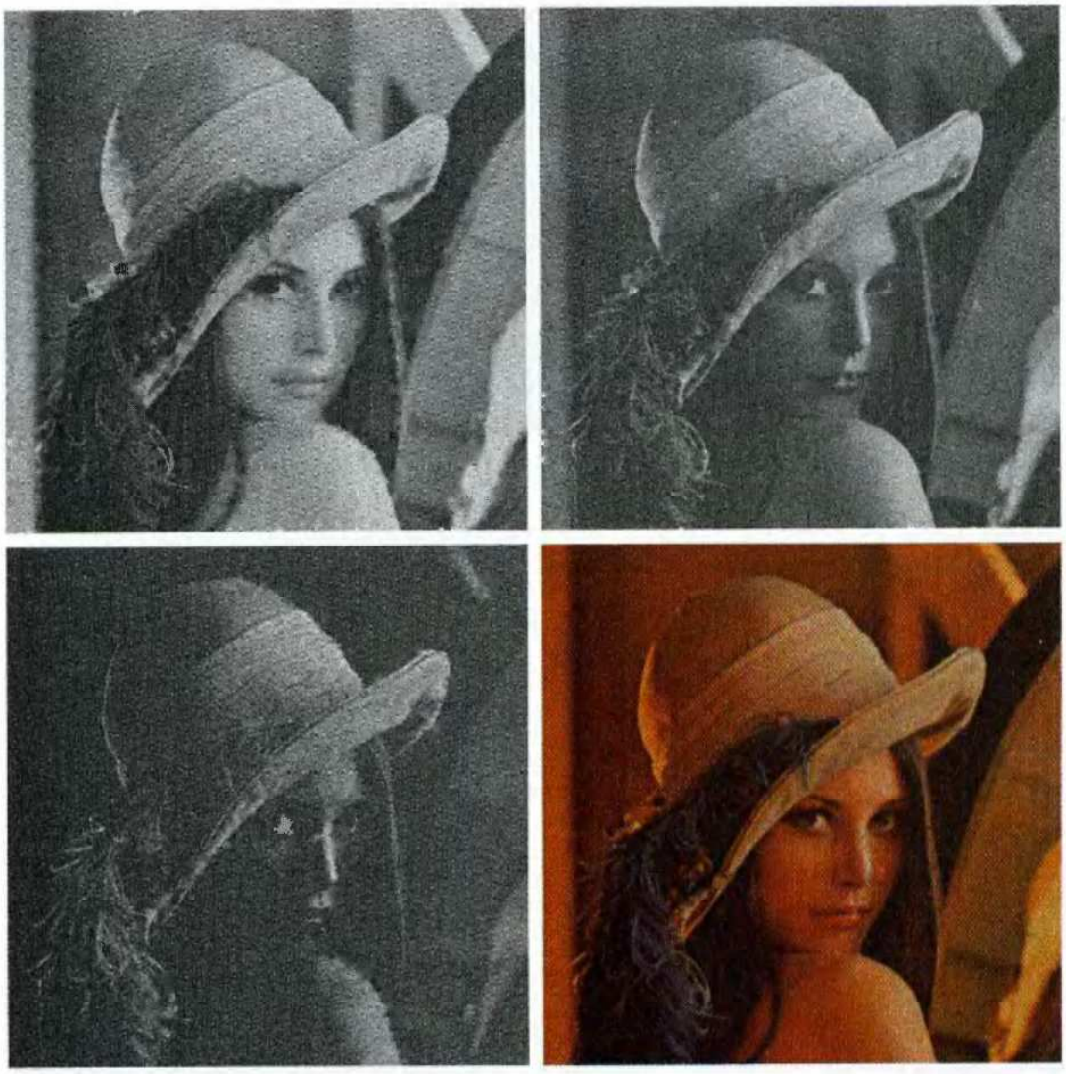
were used to generate the gradient of the individual planes) can only be determined by the requirements of a given problem. Figure 6.47 shows the three component gradient images, which, when added and scaled, were used to obtain Fig. 6.46(c). ■

6.8 Noise in Color Images

The noise models discussed in Section 5.2 are applicable to color images. Usually, the noise content of a color image has the same characteristics in each color channel, but it is possible for color channels to be affected differently by noise. One possibility is for the electronics of a particular channel to malfunction. However, different noise levels are more likely to be caused by differences in the relative strength of illumination available to each of the color channels. For example, use of a red (reject) filter in a CCD camera will reduce the strength of illumination available to the red sensor. CCD sensors are noisier at lower levels of illumination, so the resulting red component of an RGB image would tend to be noisier than the other two component images in this situation.

EXAMPLE 6.17: Illustration of the effects of converting noisy RGB images to HSI.

■ In this example we take a brief look at noise in color images and how noise carries over when converting from one color model to another. Figures 6.48(a) through (c) show the three color planes of an RGB image corrupted by Gaussian noise, and Fig. 6.48(d) is the composite RGB image. Note that fine grain noise such as this tends to be less visually noticeable in a color image than it is in a monochrome image. Figures 6.49(a) through (c) show the result of converting the RGB image in Fig. 6.48(d) to HSI. Compare these results with the HSI components of the original image (Fig. 6.39) and note how significantly

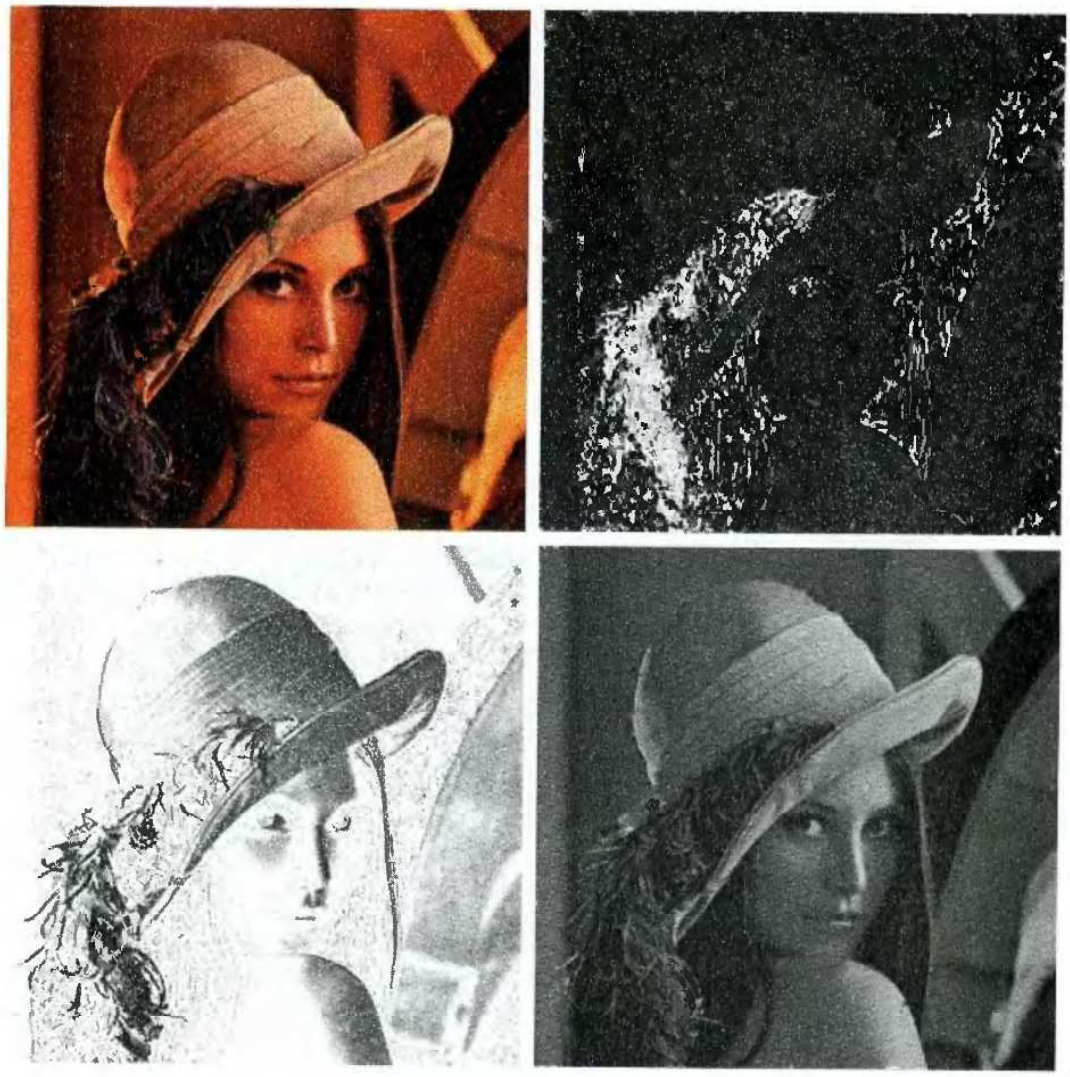


a b
c d

FIGURE 6.48 (a)–(c) Red, green, and blue component images corrupted by additive Gaussian noise of mean 0 and variance 800. (d) Resulting RGB image. [Compare (d) with Fig. 6.46(a).]

a b c

FIGURE 6.49 HSI components of the noisy color image in Fig. 6.48(d). (a) Hue. (b) Saturation. (c) Intensity.



a b
c d

FIGURE 6.50 (a) RGB image with green plane corrupted by salt-and-pepper noise. (b) Hue component of HSI image. (c) Saturation component. (d) Intensity component.

degraded the hue and saturation components of the noisy image are. This is due to the nonlinearity of the cos and min operations in Eqs. (6.2-2) and (6.2-3), respectively. On the other hand, the intensity component in Fig. 6.49(c) is slightly smoother than any of the three noisy RGB component images. This is due to the fact that the intensity image is the average of the RGB images, as indicated in Eq. (6.2-4). (Recall the discussion in Section 3.4.2 regarding the fact that image averaging reduces random noise.)

In cases when, say, only one RGB channel is affected by noise, conversion to HSI spreads the noise to all HSI component images. Figure 6.50 shows an example. Figure 6.50(a) shows an RGB image whose green image is corrupted by salt-and-pepper noise, in which the probability of either salt or pepper is 0.05. The HSI component images in Figs. 6.50(b) through (d) show clearly how the noise spread from the green RGB channel to all the HSI images. Of course, this is not unexpected because computation of the HSI components makes use of all RGB components, as shown in Section 6.2.3. ■

As is true of the processes we have discussed thus far, filtering of full-color images can be carried out on a per-image basis or directly in color vector space, depending on the process. For example, noise reduction by using an averaging filter is the process discussed in Section 6.6.1, which we know gives the same result in vector space as it does if the component images are processed independently. Other filters, however, cannot be formulated in this manner. Examples include the class of order statistics filters discussed in Section 5.3.2. For instance, to implement a median filter in color vector space it is necessary to find a scheme for ordering vectors in a way that the median makes sense. While this was a simple process when dealing with scalars, the process is considerably more complex when dealing with vectors. A discussion of vector ordering is beyond the scope of our discussion here, but the book by Plataniotis and Venetsanopoulos [2000] is a good reference on vector ordering and some of the filters based on the ordering concept.

6.9 Color Image Compression

Since the number of bits required to represent color is typically three to four times greater than the number employed in the representation of gray levels, *data compression* plays a central role in the storage and transmission of color images. With respect to the RGB, CMY(K), and HSI images of the previous sections, the *data* that are the object of any compression are the components of each color pixel (e.g., the red, green, and blue components of the pixels in an RGB image); they are the means by which the color information is conveyed. *Compression* is the process of reducing or eliminating redundant and/or irrelevant data. Although compression is the topic of Chapter 8, we illustrate the concept briefly in the following example using a color image.

EXAMPLE 6.18: A color image compression example.

■ Figure 6.51(a) shows a 24-bit RGB full-color image of an iris in which 8 bits each are used to represent the red, green, and blue components. Figure 6.51(b) was reconstructed from a compressed version of the image in (a) and is, in fact, a compressed and subsequently decompressed approximation of it. Although the compressed image is not directly displayable—it must be decompressed before input to a color monitor—the compressed image contains only 1 data bit (and thus 1 storage bit) for every 230 bits of data in the original image. Assuming that the compressed image could be transmitted over, say, the Internet, in 1 minute, transmission of the original image would require almost 4 hours. Of course, the transmitted data would have to be decompressed for viewing, but the decompression could be done in a matter of seconds. The JPEG 2000 compression algorithm used to generate Fig. 6.51(b) is a recently introduced standard that is described in detail in Section 8.6.2. Note that the reconstructed approximation image is slightly blurred. This is a characteristic of many *lossy* compression techniques; it can be reduced or eliminated by altering the level of compression. ■

a b
c d

FIGURE 6.51 Color image compression. (a) Original RGB image. (b) Result of compressing and decompressing the image in (a).

Summary

The material in this chapter is an introduction to color image processing and covers topics selected to give the reader a solid background in the techniques used in this branch of image processing. Our treatment of color fundamentals and color models was prepared as foundation material for a field that is in its own right wide in technical scope and areas of application. In particular, we focused on color models that we felt are not only useful in digital image processing but would also provide the tools necessary for further study in this area of color image processing. The discussion of pseudocolor and full-color processing on an individual image basis provides a tie to techniques that were covered in some detail in Chapters 3 through 5.

The material on color vector spaces is a departure from methods that we had studied before and highlights some important differences between gray-scale and full-color

processing. In terms of techniques, the areas of direct color vector processing are numerous and include processes such as median and other order filters, adaptive and morphological filters, image restoration, image compression, and many others. These processes are not equivalent to color processing carried out on the individual component images of a color image. The references in the following section provide a pointer to further results in this field.

Our treatment of noise in color images also points out that the vector nature of the problem, along with the fact that color images are routinely transformed from one working space to another, has implications on the issue of how to reduce noise in these images. In some cases, noise filtering can be done on a per-image basis, but others, such as median filtering, require special treatment to reflect the fact that color pixels are vector quantities, as mentioned in the previous paragraph.

Although segmentation is the topic of Chapter 10 and image data compression is the topic of Chapter 8, we gained the advantage of continuity by introducing them here in the context of color image processing. As will become evident in subsequent discussions, many of the techniques developed in those chapters are applicable to the discussion in this chapter.

References and Further Reading

For a comprehensive reference on the science of color, see Malacara [2001]. Regarding the physiology of color, see Gegenfurtner and Sharpe [1999]. These two references, along with the early books by Walsh [1958] and by Kiver [1965], provide ample supplementary material for the discussion in Section 6.1. For further reading on color models (Section 6.2), see Fortner and Meyer [1997], Poynton [1996], and Fairchild [1998]. For a detailed derivation of the HSI model equations in Section 6.2.3 see the paper by Smith [1978] or consult the book web site. The topic of pseudocolor (Section 6.3) is closely tied to the general area of data visualization. Wolff and Yaeger [1993] is a good basic reference on the use of pseudocolor. The book by Thorell and Smith [1990] also is of interest. For a discussion on the vector representation of color signals (Section 6.4), see Plataniotis and Venetsanopoulos [2000].

References for Section 6.5 are Benson [1985], Robertson [1977], and CIE [1978]. See also the classic paper by MacAdam [1942]. The material on color image filtering (Section 6.6) is based on the vector formulation introduced in Section 6.4 and on our discussion of spatial filtering in Chapter 3. Segmentation of color images (Section 6.7) has been a topic of much attention during the past ten years. The papers by Liu and Yang [1994] and by Shafarenko et al. [1998] are representative of work in this field. A special issue of the *IEEE Transactions on Image Processing* [1997] also is of interest. The discussion on color edge detection (Section 6.7.3) is from Di Zenzo [1986]. The book by Plataniotis and Venetsanopoulos [2000] does a good job of summarizing a variety of approaches to the segmentation of color images. The discussion in Section 6.8 is based on the noise models introduced in Section 5.2. References on image compression (Section 6.9) are listed at the end of Chapter 8.

See inside front cover

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

Problems

6.1 Give the percentages of red (X), green (Y), and blue (Z) light required to generate the point labeled "warm white" in Fig. 6.5.

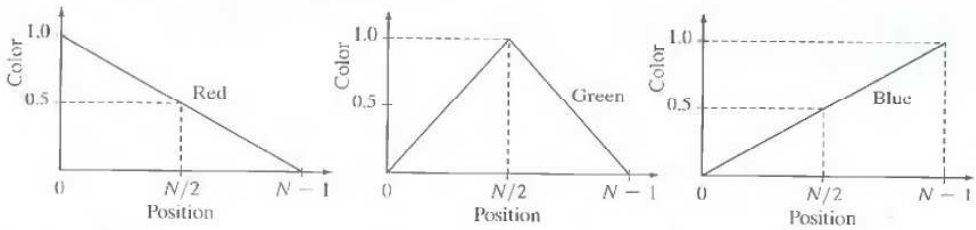
★6.2 Consider any two valid colors c_1 and c_2 with coordinates (x_1, y_1) and (x_2, y_2) in the chromaticity diagram of Fig. 6.5. Derive the necessary general expression(s)

for computing the relative percentages of colors c_1 and c_2 composing a given color that is known to lie on the straight line joining these two colors.

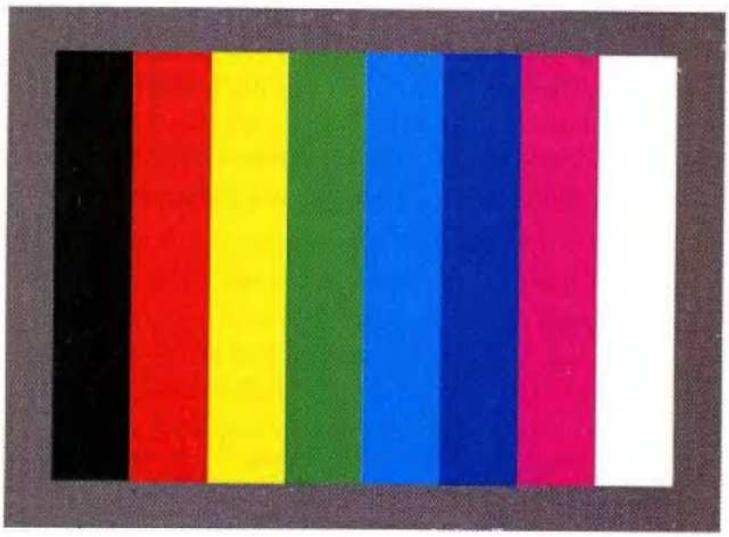
6.3 Consider any three valid colors c_1, c_2, and c_3 with coordinates $(x_1, y_1), (x_2, y_2)$, and (x_3, y_3) in the chromaticity diagram of Fig. 6.5. Derive the necessary general expression(s) for computing the relative percentages of c_1, c_2, and c_3 composing a given color that is known to lie within the triangle whose vertices are at the coordinates of c_1, c_2, and c_3.

★6.4 In an automated assembly application, three classes of parts are to be color coded in order to simplify detection. However, only a monochrome TV camera is available to acquire digital images. Propose a technique for using this camera to detect the three different colors.

6.5 In a simple RGB image, the R, G, and B component images have the horizontal intensity profiles shown in the following diagram. What color would a person see in the middle column of this image?



★6.6 Sketch the RGB components of the following image as they would appear on a monochrome monitor. All colors are at maximum intensity and saturation. In working this problem, consider the middle gray border as part of the image.



6.7 How many different shades of gray are there in a color RGB system in which each RGB image is an 8-bit image?

6.8 Consider the RGB color cube shown in Fig 6.8, and answer each of the following:

★ (a) Describe how the gray levels vary in the R, G, and B primary images that make up the front face of the color cube.

(b) Suppose that we replace every color in the RGB cube by its CMY color. This new cube is displayed on an RGB monitor. Label with a color name the eight vertices of the new cube that you would see on the screen.

(c) What can you say about the colors on the edges of the RGB color cube regarding saturation?

6.9 **(a)** Sketch the CMY components of the image in Problem 6.6 as they would appear on a monochrome monitor.

(b) If the CMY components sketched in (a) are fed into the red, green, and blue inputs of a color monitor, respectively, describe the resulting image.

★6.10 Derive the CMY intensity mapping function of Eq. (6.5-6) from its RGB counterpart in Eq. (6.5-5).

6.11 Consider the entire 216 safe-color array shown in Fig. 6.10(a). Label each cell by its (row, column) designation, so that the top left cell is (1, 1) and the rightmost bottom cell is (12, 18). At which cells will you find

(a) The purest green?

(b) The purest blue?

★6.12 Sketch the HSI components of the image in Problem 6.6 as they would appear on a monochrome monitor.

6.13 Propose a method for generating a color band similar to the one shown in the zoomed section entitled *Visible Spectrum* in Fig. 6.2. Note that the band starts at a dark purple on the left and proceeds toward pure red on the right. (*Hint:* Use the HSI color model.)

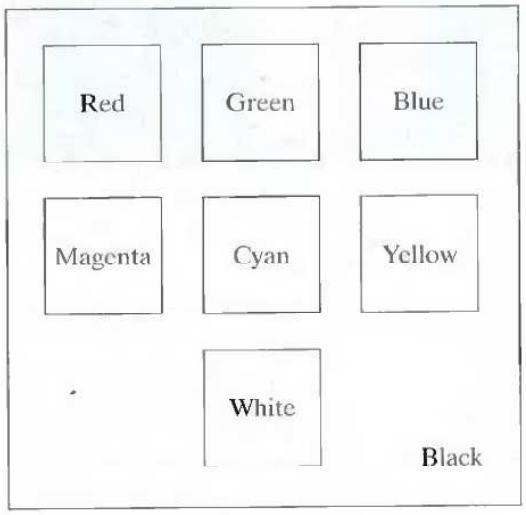
★6.14 Propose a method for generating a color version of the image shown diagrammatically in Fig. 6.13(c). Give your answer in the form of a flow chart. Assume that the intensity value is fixed and given. (*Hint:* Use the HSI color model.)

6.15 Consider the following image composed of solid color squares. For discussing your answer, choose a gray scale consisting of eight shades of gray, 0 through 7, where 0 is black and 7 is white. Suppose that the image is converted to HSI color space. In answering the following questions, use specific numbers for the grade shades if they make sense. Otherwise, the relationships "same as," "lighter than," or "darker than" are sufficient. If you cannot assign a specific gray level or one of these relationships to the image you are discussing, give the reason.

(a) Sketch the hue image.

(b) Sketch the saturation image.

(c) Sketch the intensity image.

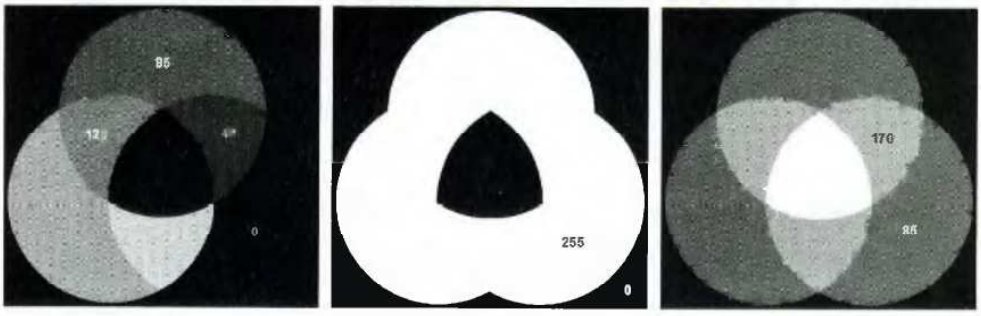


6.16 The following 8-bit images are (left to right) the H, S, and I component images from Fig. 6.16. The numbers indicate gray-level values. Answer the following questions, explaining the basis for your answer in each. If it is not possible to answer a question based on the given information, state why you cannot do so.

★ **(a)** Give the gray-level values of all regions in the hue image.

(b) Give the gray-level value of all regions in the saturation image.

(c) Give the gray-level values of all regions in the intensity image.



(a) (b) (c)

6.17 Refer to Fig. 6.27 in answering the following:

(a) Why does the image in Fig. 6.27(f) exhibit predominantly red tones?

(b) Suggest an automated procedure for coding the water in Fig. 6.27 in a bright-blue color.

(c) Suggest an automated procedure for coding the predominantly man-made components in a bright yellow color. [*Hint*: Work with Fig. 6.27(f).]

★ **6.18** Show that the saturation component of the complement of a color image cannot be computed from the saturation component of the input image alone.

6.19 Explain the shape of the hue transformation function for the complement approximation in Fig. 6.33(b) using the HSI color model.

★ **6.20** Derive the CMY transformations to generate the complement of a color image.

6.21 Draw the general shape of the transformation functions used to correct excessive contrast in the RGB color space.

★ **6.22** Assume that the monitor and printer of an imaging system are imperfectly calibrated. An image that looks balanced on the monitor appears yellowish in print. Describe general transformations that might correct the imbalance.

6.23 Compute the $L^*a^*b^*$ components of the image in Problem 6.6 assuming

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.588 & 0.179 & 0.183 \\ 0.29 & 0.606 & 0.105 \\ 0 & 0.068 & 1.021 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

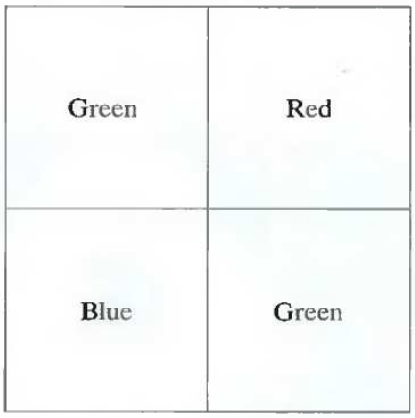
This matrix equation defines the tristimulus values of the colors generated by standard National Television System Committee (NTSC) color TV phosphors viewed under $D65$ standard illumination (Benson [1985]).

★ **6.24** How would you implement the color equivalent of gray scale histogram matching (specification) from Section 3.3.2?

6.25 Consider the following 500×500 RGB color image, where the squares are pure red, green, and blue.

(a) Suppose that we convert this image to HSI, blur the H component image with a 25×25 averaging mask, and convert back to RGB. What would the result look like?

(b) Repeat, blurring only the saturation component this time.



6.26 Show that Eq. (6.7-2) reduces to Eq. (6.7-1) when $\mathbf{C} = \mathbf{I}$, the identity matrix.

6.27 ★ (a) With reference to the discussion in Section 6.7.2, give a procedure (in flow chart form) for determining whether a color vector (point) $\mathbf{z}$ is inside a cube with sides W, centered at an average color vector $\mathbf{a}$. Distance computations are not allowed.

(b) This process also can be implemented on an image-by-image basis if the box is lined up with the axes. Show how you would do it.

6.28 Sketch the surface in RGB space for the points that satisfy the equation

$$D(\mathbf{z}, \mathbf{a}) = \left[(\mathbf{z} - \mathbf{a})^T \mathbf{C}^{-1} (\mathbf{z} - \mathbf{a})\right]^{\frac{1}{2}} = D_0$$

where D_0 is a specified nonzero constant. Assume that $\mathbf{a} = \mathbf{0}$ and that

$$\mathbf{C} = \begin{bmatrix} 8 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

6.29 Refer to Section 6.7.3. One might think that a logical approach for defining the gradient of an RGB image at any point (x, y) would be to compute the gradient *vector* (see Section 3.7.3) of each component image and then form a gradient vector for the color image by summing the three individual gradient vectors. Unfortunately, this method can at times yield erroneous results. Specifically, it is possible for a color image with clearly defined edges to have a zero gradient if this method were used. Give an example of such an image. (*Hint:* Set one of the color planes to a constant value to simplify your analysis.)

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. Conventional 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 spot 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 they can be presented in other ways, this approach simplifies both the mathematics 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 of a multiresolution-based toolkit for the representation and processing of images. To demonstrate the usefulness of the toolkit, 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 gray level 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

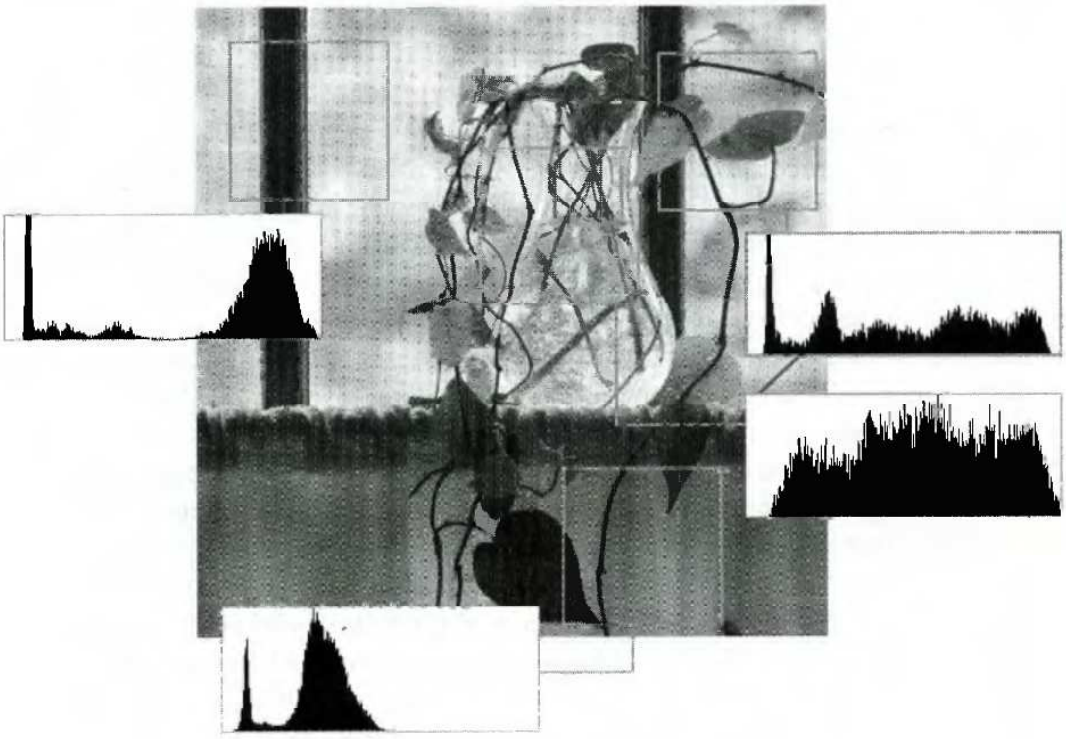


FIGURE 7.1 A natural image and its local histogram variations.

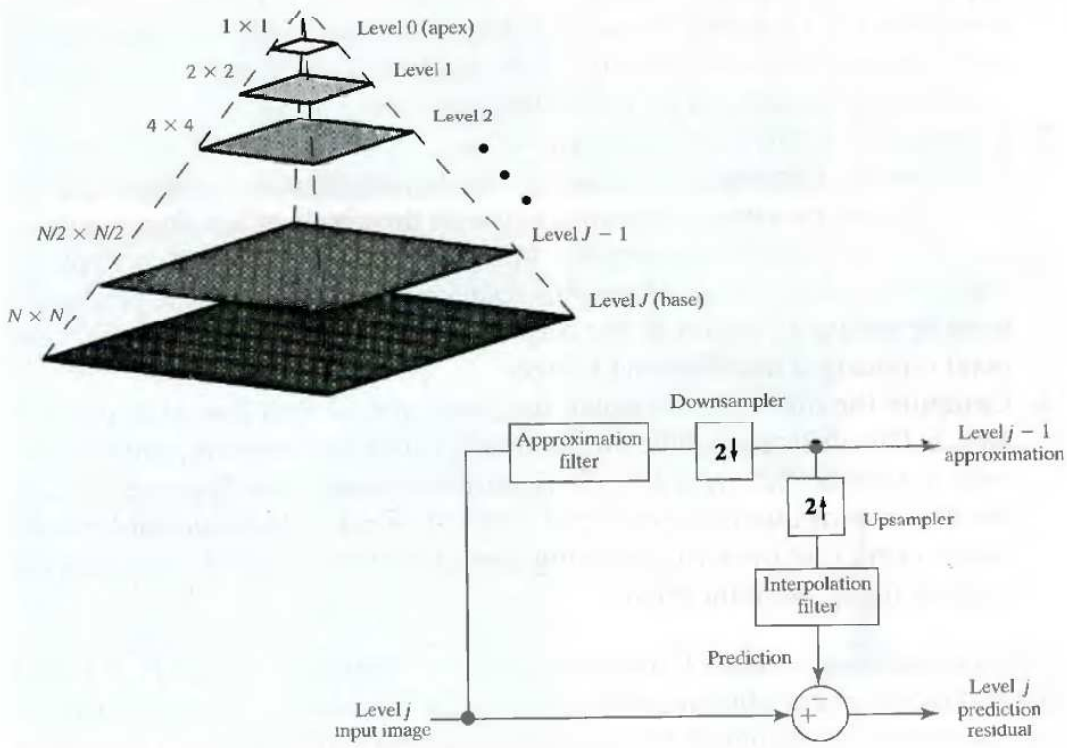
section—even first-order statistics vary significantly from one part of an image to another. They defy simple statistical modeling over the entire image.

7.1.1 Image Pyramids

A powerful, but 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. Since base level J is size $2^J \times 2^J$ or $N \times N$, where $J = \log_2 N$, intermediate level j is size $2^j \times 2^j$, where $0 \leq j < J$. Fully populated pyramids are composed of $J + 1$ resolution levels from $2^J \times 2^J$ to $2^0 \times 2^0$, but most pyramids are truncated to $P + 1$ levels, where $j = J - P, \ldots, J - 2, J - 1, J$ and $1 \leq P \leq J$. That is, we normally limit ourselves to P reduced resolution approximations of the original image; a 1×1 or single pixel approximation of a 512×512 image, for example, is of little value. The total number of elements 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 image pyramids. The *level $j - 1$ approximation* output is used to create *approximation pyramids*,



a
b

FIGURE 7.2 (a) A pyramidal image structure and (b) system block diagram for creating it.

which contain one or more approximations of the original image. Both the original image, which is at the base of the pyramid, and its P reduced resolution approximations can be accessed and manipulated directly. The *level j prediction residual* output of Fig. 7.2(b) is used to build *prediction residual pyramids*. These pyramids contain a low-resolution approximation of the original image at level $J - P$ and information for the construction of P higher-resolution approximations at the other levels. The information at level j is the difference between the level j approximation of the corresponding approximation pyramid and an estimate of that approximation based on the level $j - 1$ prediction residual. This difference can be coded—and therefore stored and transmitted—more efficiently than the approximation.

As the block diagram of Fig. 7.2(b) suggests, approximation and prediction residual pyramids are computed in an iterative fashion. A $P + 1$ level pyramid is built by executing the operations in the block diagram P times. During the first iteration or pass, $j = J$ and the original $2^J \times 2^J$ image is applied as the *level J input image*. This produces the *level J − 1 approximation* and *level J prediction residual* results. For passes $j = J - 1, J - 2, \ldots, J - P + 1$ (in that order), the previous iteration's *level j − 1 approximation* output is used as the input. Each pass is composed of three sequential steps:

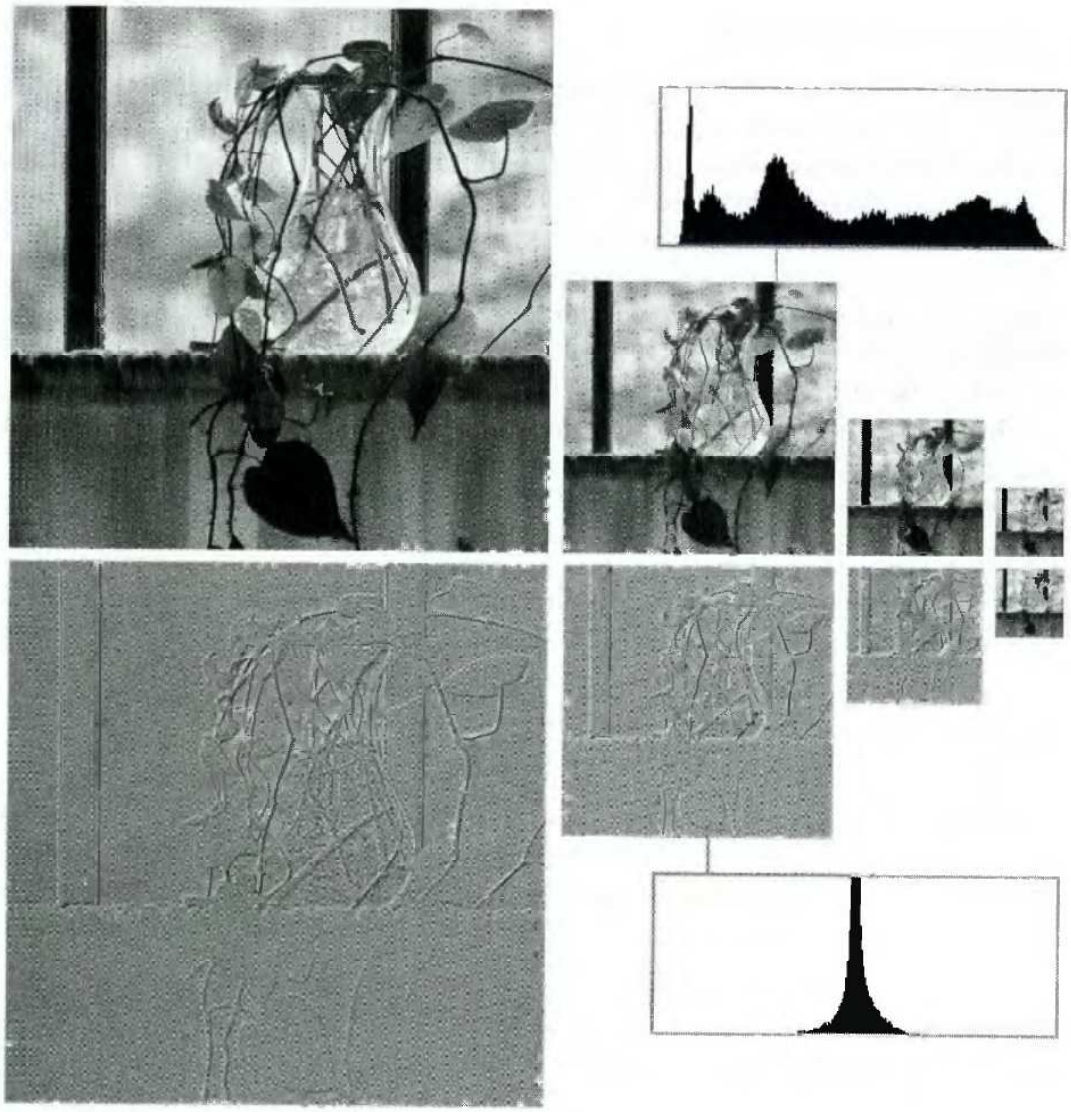
1. Compute a reduced-resolution approximation of the input image. This is done by filtering the input and downsampling (i.e., subsampling) the filtered result by a factor of 2. A variety of filtering operations can be used, including neighborhood averaging, which produces a *mean pyramid*, lowpass Gaussian filtering (see Section 4.2.4), which produces a *Gaussian pyramid*, or no filtering, which results in a *subsampling pyramid*. The quality of the generated approximation, labeled the *level j − 1 approximation* in Fig. 7.2(b), is a function of the filter selected. Without the filter, aliasing can become pronounced in the upper levels of the pyramid, as the subsampled points may not be good representatives of the areas from which they were taken.
2. Upsample the output of the previous step—again by a factor of 2—and filter the result. This creates a *prediction* image with the same resolution as the input. By interpolating intensities between the pixels of the *Step 1* output, the *interpolation filter* determines how accurately the *prediction* approximates the input to *Step 1*. If the *interpolation filter* is omitted, the prediction is an upsampled version of the *Step 1* output and the blocking effects of pixel replication may become visible.
3. Compute the difference between the *prediction* of *Step 2* and the input to *Step 1*. This difference, labeled the *level j prediction residual*, can be later used to reconstruct progressively the original image (see Example 7.1). In the absence of quantization error, a prediction residual pyramid can be used to generate the corresponding approximation pyramid, including the original image, without error.

Executing this procedure P times produces two intimately related $P + 1$ level approximation and prediction residual pyramids. The *level j − 1 approximation* outputs are used to populate the approximation pyramid; the *level j prediction*

residual outputs are placed in the prediction residual pyramid. If a prediction residual pyramid is not needed, *Steps 2* and *3*, together with the upsampler, *interpolation filter*, and summer of Fig. 7.2(b), can be omitted.

EXAMPLE 7.1: Gaussian and Laplacian pyramids.

■ Figure 7.3 shows one possible approximation and prediction residual pyramid for the vase of Fig. 7.1. The approximation pyramid in Fig. 7.3(a) is a Gaussian pyramid. Filtering was performed in the spatial domain using a 5×5 lowpass Gaussian convolution kernel of the type depicted in Fig. 4.9(c) of Section 4.2.4. As can be seen, 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). That is, $P = 3$ and the pyramid has been truncated to four levels—levels 9, 8, 7, and 6 out of a possible $\log_2(512) + 1$ or 10 levels. Note the reduction in detail that accompanies the lower resolutions of the pyramid. The level 6 (i.e., 64×64) approximation is suitable for locating the window stiles, for example, but not for finding the philodendron's stems. In general, a pyramid's lower-resolution levels can be used for the analysis of large



a
b

FIGURE 7.3 Two image pyramids and their statistics: (a) a Gaussian (approximation) pyramid and (b) a Laplacian (prediction residual) pyramid.

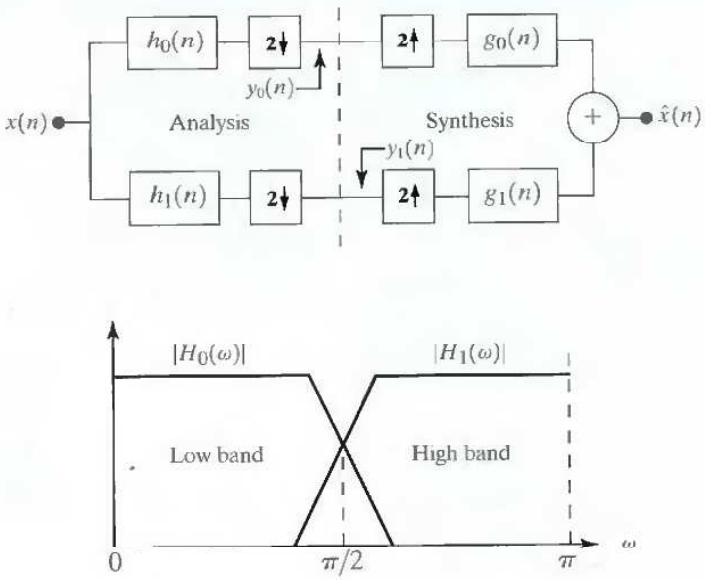
structures or overall image context; its high-resolution images are appropriate for analyzing individual object characteristics. Such a coarse to fine analysis strategy is particularly useful in pattern recognition.

The Laplacian pyramid in Fig. 7.3(b) contains the prediction residuals needed to compute its Gaussian counterpart in 7.3(a). To build the Gaussian pyramid, we begin with the Laplacian pyramid's level 6 64×64 approximation image, predict the Gaussian pyramid's level 7 128×128 resolution approximation (by upsampling and filtering), and add the Laplacian's level 7 prediction residual. This process is repeated using successively computed approximation images until the original 512×512 image is generated. Note that the first-order statistics of the prediction residual images in the Laplacian pyramid are highly peaked around zero. Unlike their Gaussian counterparts, these images can be highly compressed by assigning fewer bits to the more probable values (see the variable length codes of Section 8.1.1). Finally, we note that the prediction residuals in Fig. 7.3(b) are scaled to make the smaller prediction errors more visible; the prediction residual histogram, however, is based on prescaled residuals, with level 128 representing zero error. ■

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 band-limited components, called *subbands*, which can be reassembled to reconstruct the original image without error. Originally developed for speech and image compression, each subband is generated by bandpass filtering the input. Since the bandwidth of the resulting subbands is smaller than that of the original image, the subbands can be downsampled without loss of information. Reconstruction of the original image is accomplished by upsampling, filtering, and summing the individual subbands.

Figure 7.4(a) shows the principal components of a two-band subband coding and decoding system. The input of the system is a one-dimensional, band-limited



a
b

FIGURE 7.4 (a) A two-band filter bank for one-dimensional subband coding and decoding, and (b) its spectrum splitting properties.

discrete-time signal $x(n)$ for $n = 0, 1, 2, \ldots$; the output sequence, $\hat{x}(n)$, is formed through the decomposition of $x(n)$ into $y_0(n)$ and $y_1(n)$ via *analysis filters* $h_0(n)$ and $h_1(n)$, and subsequent recombination via *synthesis filters* $g_0(n)$ and $g_1(n)$. Note that filters $h_0(n)$ and $h_1(n)$ are half-band digital filters whose idealized transfer characteristics, H_0 and H_1, are shown in Fig. 7.4(b). Filter H_0 is a lowpass filter whose output is an *approximation* of $x(n)$; filter H_1 is a highpass filter whose output is the high frequency or *detail* part of $x(n)$. All filtering is performed in the time domain by convolving each filter's input with its impulse response—its response to a unit amplitude impulse function, $\delta(n)$. We wish to select $h_0(n)$, $h_1(n)$, $g_0(n)$, and $g_1(n)$ (or, alternately H_0, H_1, G_0, and G_1) so that the input can be reconstructed perfectly. That is, so that $\hat{x}(n) = x(n)$.

The *Z-transform*, a generalization of the discrete Fourier transform, is the ideal tool for studying discrete-time, sampled-data systems like the one in Fig. 7.4(a). The Z-transform of sequence $x(n)$ for $n = 0, 1, 2, \ldots$ is

$$X(z) = \sum_{-\infty}^{\infty} x(n) z^{-n} \tag{7.1-1}$$

where z is a complex variable. [If $e^{j\omega}$ is substituted for z, Eq. (7.1-1) becomes the discrete Fourier transform of $x(n)$]. Our interest in the Z-transform stems from the ease with which it handles sampling rate changes. Downsampling by a factor of 2 in the time domain corresponds to the simple Z-domain operation

$$x_{\text{down}}(n) = x(2n) \Leftrightarrow X_{\text{down}}(z) = \frac{1}{2}\left[X(z^{1/2}) + X(-z^{1/2})\right] \tag{7.1-2}$$

where the double arrow indicates that the expressions on the left and right form a Z-transform pair. In a similar manner, upsampling—again by a factor of 2—is defined by the transform pair

$$x^{\text{up}}(n) = \begin{cases} x(n/2) & n = 0, 2, 4, \ldots \\ 0 & \text{otherwise} \end{cases} \Leftrightarrow X^{\text{up}}(z) = X(z^2). \tag{7.1-3}$$

If sequence $x(n)$ is downsampled and subsequently upsampled to yield $\hat{x}(n)$, Eqs. (7.1-2) and (7.1-3) combine to yield

$$\hat{X}(z) = \frac{1}{2}\left[X(z) + X(-z)\right] \tag{7.1-4}$$

where $\hat{x}(n) = Z^{-1}\left[\hat{X}(z)\right]$ is the resulting downsampled-upsampled sequence. The $X(-z)$ term in this equation is the Z-transform of an *aliased* or *modulated* version of sequence $x(n)$. Its inverse Z-transform is

$$Z^{-1}\left[X(-z)\right] = (-1)^n x(n). \tag{7.1-5}$$

With this brief introduction to the Z-transform, consider again the subband coding and decoding system of Fig. 7.4(a). In accordance with Eq. (7.1-4), we can express the system's output as

$$\hat{X}(z) = \frac{1}{2}G_0(z)[H_0(z)X(z) + H_0(-z)X(-z)] \tag{7.1-6}$$
$$+ \frac{1}{2}G_1(z)[H_1(z)X(z) + H_1(-z)X(-z)]$$

where, for example, the output of filter $h_0(n)$ in Fig. 7.4(a) is defined by the transform pair

$$h_0(n) * x(n) = \sum_k h_0(n-k)x(k) \Leftrightarrow H_0(z)X(z). \tag{7.1-7}$$

As with Fourier transforms, convolution in the time (or spatial) domain is equivalent to multiplication in the Z-domain. Rearranging terms in Eq. (7.1-6), we then get

$$\hat{X}(z) = \frac{1}{2}[H_0(z)G_0(z) + H_1(z)G_1(z)]X(z) \tag{7.1-8}$$
$$+ \frac{1}{2}[H_0(-z)G_0(z) + H_1(-z)G_1(z)]X(-z)$$

where the second component—by virtue of the fact that it contains the $-z$ dependence—represents the aliasing that is introduced by the downsampling-upsampling process.

For error-free reconstruction of the input, $\hat{x}(n) = x(n)$ and $\hat{X}(z) = X(z)$. Thus, we impose the following conditions:

$$H_0(-z)G_0(z) + H_1(-z)G_1(z) = 0 \tag{7.1-9}$$
$$H_0(z)G_0(z) + H_1(z)G_1(z) = 2. \tag{7.1-10}$$

Equation (7.1-9) eliminates aliasing by forcing the second term of Eq. (7.1-8) to zero; Eq. (7.1-10) eliminates amplitude distortion by reducing the first term to $X(z)$. Both can be incorporated into the single matrix expression

$$[G_0(z) \quad G_1(z)]\mathbf{H}_m(z) = [2 \quad 0] \tag{7.1-11}$$

where *analysis modulation matrix* $\mathbf{H}_m(z)$ is

$$\mathbf{H}_m(z) = \begin{bmatrix} H_0(z) & H_0(-z) \\ H_1(z) & H_1(-z) \end{bmatrix}. \tag{7.1-12}$$

Assuming $\mathbf{H}_m(z)$ is nonsingular (i.e., it has a common left and right inverse), we can transpose Eq. (7.1-11) and left multiply by inverse $(\mathbf{H}_m^T(z))^{-1}$ to get

$$\begin{bmatrix} G_0(z) \\ G_1(z) \end{bmatrix} = \frac{2}{\det(\mathbf{H}_m(z))}\begin{bmatrix} H_1(-z) \\ -H_0(-z) \end{bmatrix} \tag{7.1-13}$$

where $\det(\mathbf{H}_m(z))$ denotes the determinate of $\mathbf{H}_m(z)$.

Equations (7.1-9) through (7.1-13) reveal several important characteristics of perfect reconstruction filter banks. Matrix Eq. (7.1-13), for instance, tells us that $G_1(z)$ is a function of $H_0(-z)$, while $G_0(z)$ is a function of $H_1(-z)$. The analysis

and synthesis filters are *cross-modulated*. That is, diagonally opposed filters in the block diagram of Fig. 7.4(a) are functionally related by $-z$ in the Z-domain. For finite impulse response (FIR) filters, the determinate of the modulation matrix is a pure delay, i.e., $\det(\mathbf{H}_m(z)) = \alpha z^{-(2k+1)}$ (see, for example, Vetterli and Kovacevic [1995]). Thus, the exact form of the cross-modulation is a function of α. The $z^{-(2k+1)}$ term can be considered arbitrary since it is a shift that only changes the overall delay of the filter. Ignoring the delay, letting $\alpha = 2$, and taking the inverse Z-transform of Eq. (7.1-13), for instance, we get

$$\begin{aligned} g_0(n) &= (-1)^n h_1(n) \\ g_1(n) &= (-1)^{n+1} h_0(n). \end{aligned} \tag{7.1-14}$$

If $\alpha = -2$, the resulting expressions are sign reversed:

$$\begin{aligned} g_0(n) &= (-1)^{n+1} h_1(n) \\ g_1(n) &= (-1)^n h_0(n). \end{aligned} \tag{7.1-15}$$

Thus, FIR synthesis filters are cross-modulated copies of the analysis filters—with one (and only one) being sign reversed.

Equations (7.1-9) through (7.1-13) can also be used to demonstrate the biorthogonality of the analysis and synthesis filters. To do this, let $P(z)$ be the product of the lowpass analysis and synthesis filter transfer functions. Substituting for G_0 from Eq. (7.1-13), we get

$$P(z) = G_0(z)H_0(z) = \frac{2}{\det(\mathbf{H}_m(z))} H_0(z)H_1(-z). \tag{7.1-16}$$

Since $\det(\mathbf{H}_m(z)) = -\det(\mathbf{H}_m(-z))$, product $G_1(z)H_1(z)$ can be similarly defined as

$$G_1(z)H_1(z) = \frac{-2}{\det(\mathbf{H}_m(z))} H_0(-z)H_1(z) = P(-z). \tag{7.1-17}$$

Thus, $G_1(z)H_1(z) = P(-z) = G_0(-z)H_0(-z)$ and Eq. (7.1-10) becomes

$$G_0(z)H_0(z) + G_0(-z)H_0(-z) = 2. \tag{7.1-18}$$

Taking the inverse Z-transform, we see that

$$\sum_k g_0(k)h_0(n-k) + (-1)^n \sum_k g_0(k)h_0(n-k) = 2\delta(n)$$

where, as usual, impulse function $\delta(n)$ is 1 if $n = 0$ and 0 otherwise. Since the odd indexed terms cancel, additional simplification yields[†]

$$\sum_k g_0(k)h_0(2n-k) = \langle g_0(k), h_0(2n-k)\rangle = \delta(n). \tag{7.1-19}$$

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

By starting over from Eqs. (7.1-9) and (7.1-10) and expressing G_0 and H_0 as a function of G_1 and H_1, we can also show that

$$\begin{aligned}\langle g_1(k), h_1(2n-k)\rangle &= \delta(n) \\ \langle g_0(k), h_1(2n-k)\rangle &= 0\end{aligned} \tag{7.1-20}$$

and

$$\langle g_1(k), h_0(2n-k)\rangle = 0,$$

which, in combination with Eq. (7.1-19), establishes the more general expression

$$\langle h_i(2n-k), g_j(k)\rangle = \delta(i-j)\delta(n), \quad i, j = \{0, 1\}. \tag{7.1-21}$$

Filter banks satisfying this condition are called *biorthogonal*. Moreover, the analysis and synthesis filter impulse responses of all two-band, real-coefficient, perfect reconstruction filter banks are subject to the biorthogonality constraint. Example biorthogonal, FIR filters include the *biorthogonal spline* family (Cohen, Daubechies, and Feauveau [1992]) and the *biorthogonal coiflet* family (Tian and Wells [1995]).

Table 7.1 gives three general solutions to Eqs. (7.1-9) and (7.1-10). While each satisfies the biorthogonality requirement of Eq. (7.1-21), they are generated in different ways and define unique classes of perfect reconstruction filters. For each class, a "prototype" filter is designed to a particular specification and the remaining filters are computed from the prototype. Columns 1 and 2 of Table 7.1 are classic results from the filter bank literature—namely, *quadrature mirror filters* (QMFs) (Croisier, Estaban, and Galand [1976]) and *conjugate quadrature filters* (CQFs) (Smith and Barnwell [1986]). The filters in column three, which are later used in the development of the fast wavelet transform (see Section 7.4), are called *orthonormal*. They move a step beyond biorthogonality and require

$$\langle g_i(n), g_j(n+2m)\rangle = \delta(i-j)\delta(m), \quad i, j = \{0, 1\}, \tag{7.1-22}$$

which defines orthonormality for perfect reconstruction filter banks. Note that in the expression for $G_1(z)$ in row three of the table, $2K$ denotes the length or number of coefficients (i.e., filter taps) in each filter. As can be seen, G_1 is related to lowpass synthesis filter G_0 by modulation (see Eq. (7.1-5)), time reversal, and odd shift†. In addition, both H_0 and H_1 are time-reversed versions of the corresponding synthesis filters, G_0 and G_1, respectively. Taking the inverse Z-transform of the appropriate entries from column 3 of Table 7.1, we get that

$$\begin{aligned}g_1(n) &= (-1)^n g_0(2K-1-n) \\ h_i(n) &= g_i(2K-1-n), \quad i = \{0, 1\}\end{aligned} \tag{7.1-23}$$

where h_0, h_1, g_0, and g_1 are the impulse responses of the defined orthonormal filters. Examples include the Smith and Barnwell filter (Smith and Barnwell

†The Z-transform pairs for time reversal and shift are $x(-n) \Leftrightarrow X(z^{-1})$ and $x(n-k) \Leftrightarrow z^{-k}X(z)$, respectively.

Filter	QMF	CQF	Orthonormal
$H_0(z)$	$H_0^2(z) - H_0^2(-z) = 2$	$H_0(z)H_0(z^{-1}) + H_0(-z)H_0(-z^{-1}) = 2$	$G_0(z^{-1})$
$H_1(z)$	$H_0(-z)$	$z^{-1}H_0(-z^{-1})$	$G_1(z^{-1})$
$G_0(z)$	$H_0(z)$	$H_0(z^{-1})$	$G_0(z)G_0(z^{-1}) + G_0(-z)G_0(-z^{-1}) = 2$
$G_1(z)$	$-H_0(-z)$	$zH_0(-z)$	$-z^{-2K+1}G_0(-z^{-1})$

TABLE 7.1 Perfect reconstruction filter families.

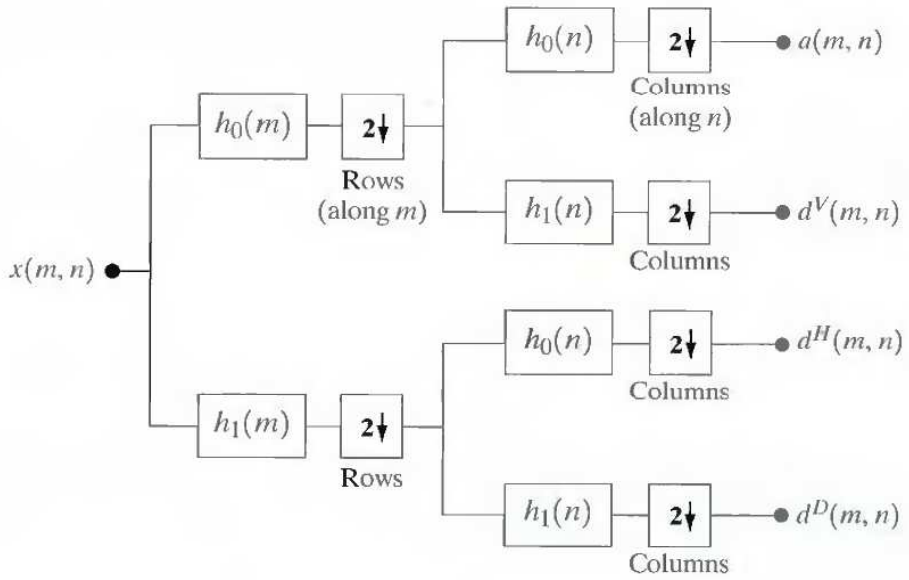


FIGURE 7.5 A two-dimensional, four-band filter bank for subband image coding.

[1984]), Daubechies filters (Daubechies [1988]), and the Vaidyanathan and Hoang filter (Vaidyanathan and Hoang [1988]).

The one-dimensional filters in Table 7.1 can be used as two-dimensional separable filters for the processing of images. As can be seen in Fig. 7.5, separable filters are first applied in one dimension (e.g., vertically) and then in the other (e.g., horizontally). Moreover, downsampling is performed in two stages—once before the second filtering operation to reduce the overall number of computations. The resulting filtered outputs, denoted $a(m, n)$, $d^V(m, n)$, $d^H(m, n)$, and $d^D(m, n)$ in Fig. 7.5, are called the approximation, vertical detail, horizontal detail, and diagonal detail subbands of the image, respectively. One or more of these subbands can be split into four smaller subbands, which can be split again, and so on.

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

■ Figure 7.6 shows the impulse responses of four 8-tap orthonormal filters. The coefficients of lowpass filter $h_0(n)$ for $0 \le n \le 7$ are −0.01059740, 0.03288301, 0.03084138, −0.18703481, −0.02798376, 0.63088076, 0.71484657, and 0.23037781 (Daubechies [1992]); the coefficients of the remaining orthonormal filters can be computed using Eq. (7.1-23). Note (by visual inspection) the cross-modulation of the analysis and synthesis filters in Fig. 7.6. It is relatively easy to

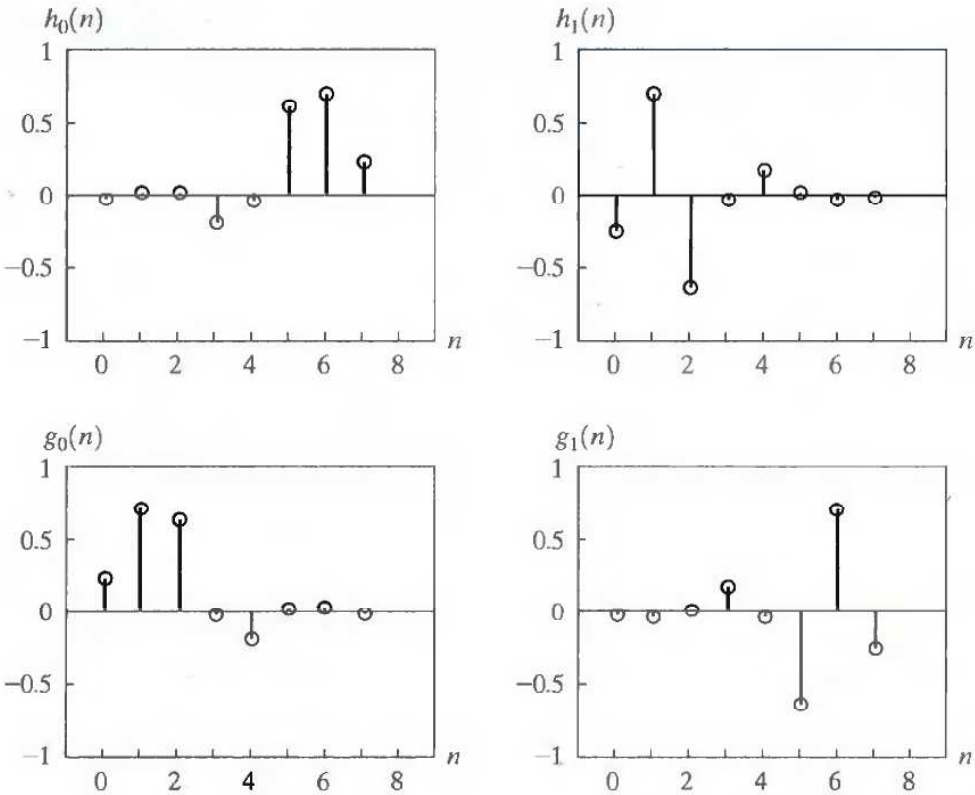


FIGURE 7.6 The impulse responses of four 8-tap Daubechies orthonormal filters.

show numerically that the filters are both biorthogonal (satisfy Eq. 7.1-21) and orthogonal (satisfy Eq. 7.1-22). Moreover, they fulfill Eqs. (7.1-9) and (7.1-10) and 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. 7.6, is shown in Fig. 7.7. 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 the upper left quadrant, have been scaled to make their underlying structure more visible. Note the aliasing that is present in d^H and d^V, which is due to downsampling the barely discernable window screen in Fig. 7.1. Reconstruction of the original image from the subbands via synthesis filters $g_0(n)$ and $g_1(n)$ will cancel this aliasing in accordance with Eq. (7.1-9). To perform the reconstruction, a filter bank that roughly mirrors the system in Fig. 7.5 is required. In the new filter bank, the $h_i(n)$ for $i = \{0, 1\}$ filters are replaced by their $g_i(n)$ counterparts, and upsamplers and summers are added. ■

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 are the oldest and simplest known orthonormal wavelets. They will be used in a number of examples in the sections that follow.

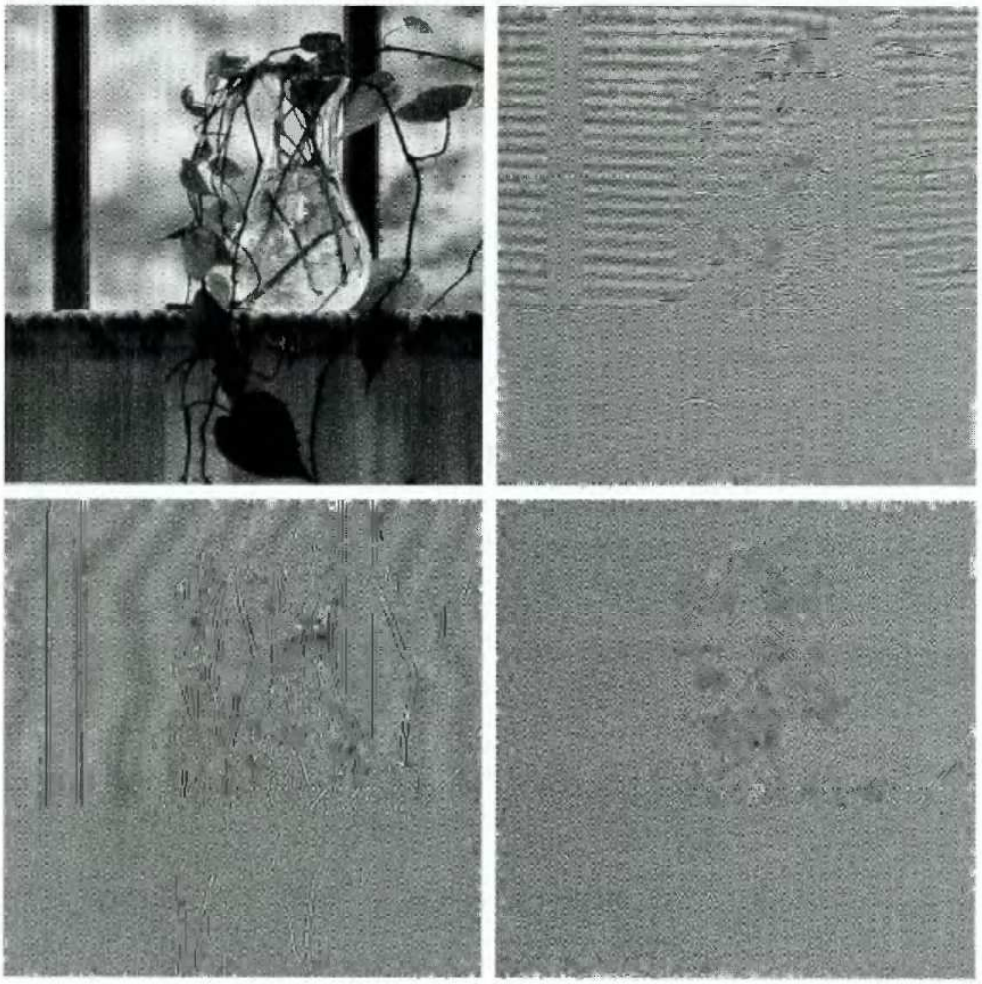


FIGURE 7.7 A four-band split of the vase in Fig. 7.1 using the subband coding system of Fig. 7.5.

The Haar transform itself is both separable and symmetric and can be expressed in matrix form

$$\mathbf{T} = \mathbf{HFH} \tag{7.1-24}$$

where $\mathbf{F}$ is an $N \times N$ image matrix, $\mathbf{H}$ is an $N \times N$ transformation matrix, and $\mathbf{T}$ is the resulting $N \times N$ transform. For the Haar transform, transformation matrix $\mathbf{H}$ 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

$$h_0(z) = h_{00}(z) = \frac{1}{\sqrt{N}}, \quad z \in [0, 1] \tag{7.1-25}$$

and

$$h_k(z) = h_{pq}(z) = \frac{1}{\sqrt{N}} \begin{cases} 2^{p/2} & (q-1)/2^p \le z < (q-0.5)/2^p \\ -2^{p/2} & (q-0.5)/2^p \le z < q/2^p \\ 0 & \text{otherwise}, z \in [0, 1]. \end{cases} \tag{7.1-26}$$

The ith row of an $N \times N$ Haar transformation matrix contains the elements of $h_i(z)$ for $z = 0/N, 1/N, 2/N, \ldots, (N-1)/N$. If $N = 4$, for example, 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

$$\mathbf{H}_4 = \frac{1}{\sqrt{4}} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & -\sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}. \tag{7.1-27}$$

In a similar manner, the 2×2 transformation matrix, $\mathbf{H}_2$, is

$$\mathbf{H}_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{7.1-28}$$

Its basis functions define the only two-tap FIR filter bank that will satisfy the QMF prototype filter specification in row 1 and column 1 of Table 7.1. The coefficients of the corresponding QMF analysis filters, $h_0(n)$ and $h_1(n)$, are the elements of the first and second rows of matrix $\mathbf{H}_2$, respectively.

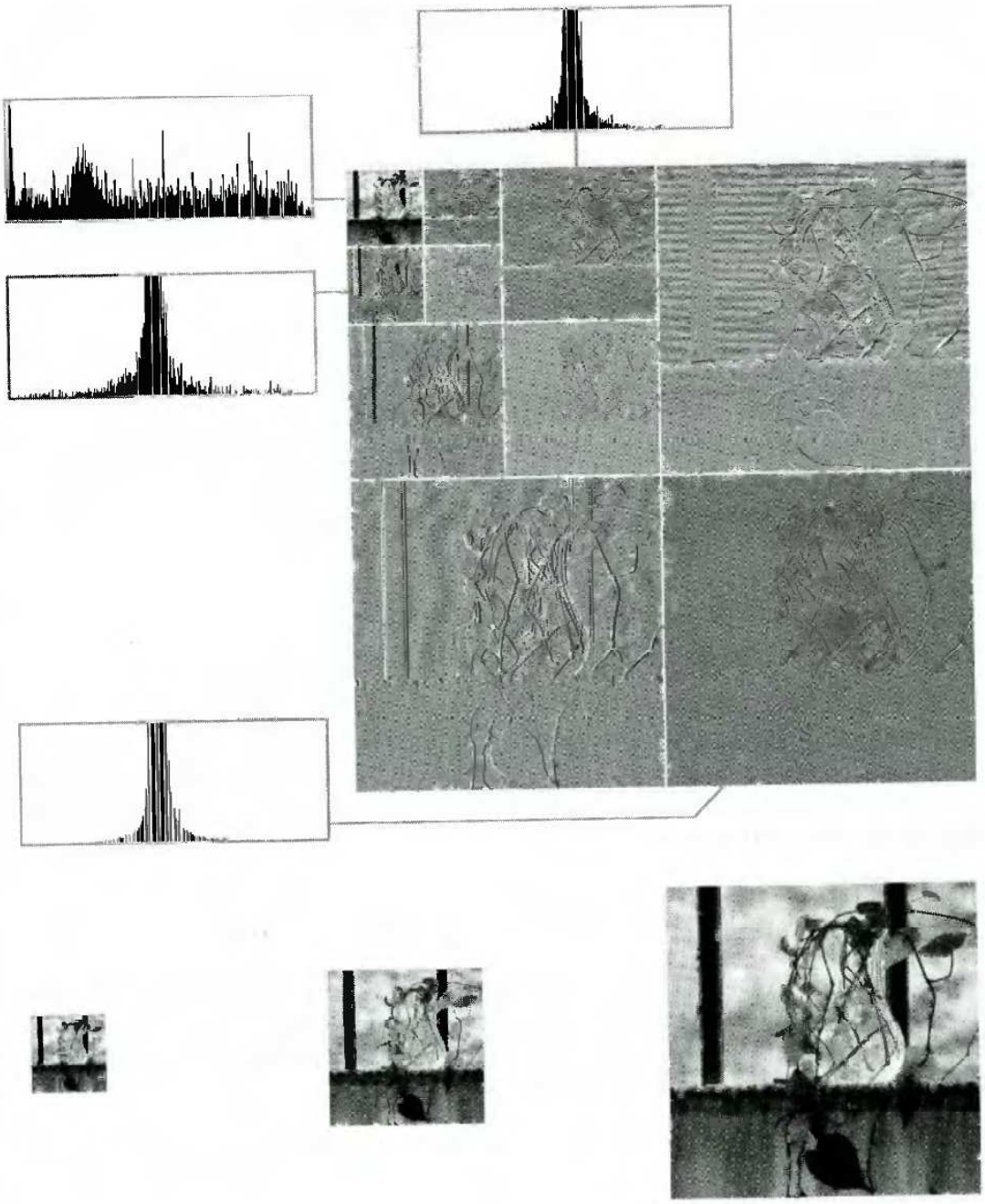
EXAMPLE 7.3: Haar functions in a discrete wavelet transform.

■ Figure 7.8(a) shows a multiresolution-based decomposition of Fig. 7.1 using the Haar basis functions of Eqs. (7.1-25) and (7.1-26). Unlike the pyramidal structure in Fig. 7.3, this representation, called the *discrete wavelet transform* and developed later in the chapter, contains the same number of pixels as the original image. In addition,

1. Its local statistics are relatively constant and easily modeled. See Fig. 7.8(a).
2. Many of its values are close to zero. This makes it an excellent candidate for image compression.
3. Both coarse and fine resolution approximations of the original image can be extracted from it. Figures 7.8(b)–(d) were reconstructed from the subimages of Fig. 7.8(a).

In database applications, these properties make it easy for users to access low-quality versions of images during searches and later retrieve additional data to refine them.

Finally, we note that Fig. 7.8(a) bears a close resemblance to both the subband coding result of Fig. 7.7 and the Laplacian pyramid of Fig. 7.3(b). As in the previous results, the subimages in Fig. 7.8(a) are scaled to make their underlying structure more visible. The approximation images in Fig. 7.8(b)–(d) are of size



a
b c d

FIGURE 7.8 (a) A discrete wavelet transform using Haar 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. A perfect 512×512 reconstruction of the original image is also possible. ■

7.2 Multiresolution Expansions

The previous section introduced three well-known imaging techniques that played an important role in the development of a unique mathematical theory 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 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 the 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

$$V = \overline{\operatorname{Span}_k \{\varphi_k(x)\}}. \tag{7.2-2}$$

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 $\tilde{\varphi}_k(x)$'s 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 $f(x)$.

†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 $\langle f(x), g(x) \rangle = \int f(x) g(x)\, dx$.

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

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

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†

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

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

Now consider the set of expansion functions composed of integer translations and binary scalings of the real, square-integrable function $\varphi(x)$; that 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, j determines $\varphi_{j,k}(x)$'s width—how broad or narrow it is along the x-axis—and $2^{j/2}$ controls its height or amplitude. Because the shape of $\varphi_{j,k}(x)$ changes with j, $\varphi(x)$ is called a *scaling function*. By choosing $\varphi(x)$ wisely, $\{\varphi_{j,k}(x)\}$ can be made to span $L^2(\mathbf{R})$, the set of all measurable, square-integrable functions.

---

†The norm of $f(x)$, denoted $\|f(x)\|$, is defined as the square root of the inner product of $f(x)$ with itself.

‡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.

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 $\{\varphi_{j,k}(x)\}$. It will not span $L^2(\mathbf{R})$, but a subspace within it. 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 $f(x) \in V_{j_0}$, it can be written

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

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.

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 7.9(a)–(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 when $j = 1$, as opposed to $j = 0$, the resulting expansion functions are narrower and closer together.

Figure 7.9(e) shows a member of subspace V_1. This function does not belong to V_0 because the V_0 expansion functions in 7.9(a) and (b) are too coarse to represent it. Higher-resolution functions like those in 7.9(c) and (d) are required. They can be used, as shown in (e), to represent the function by the three-term expansion

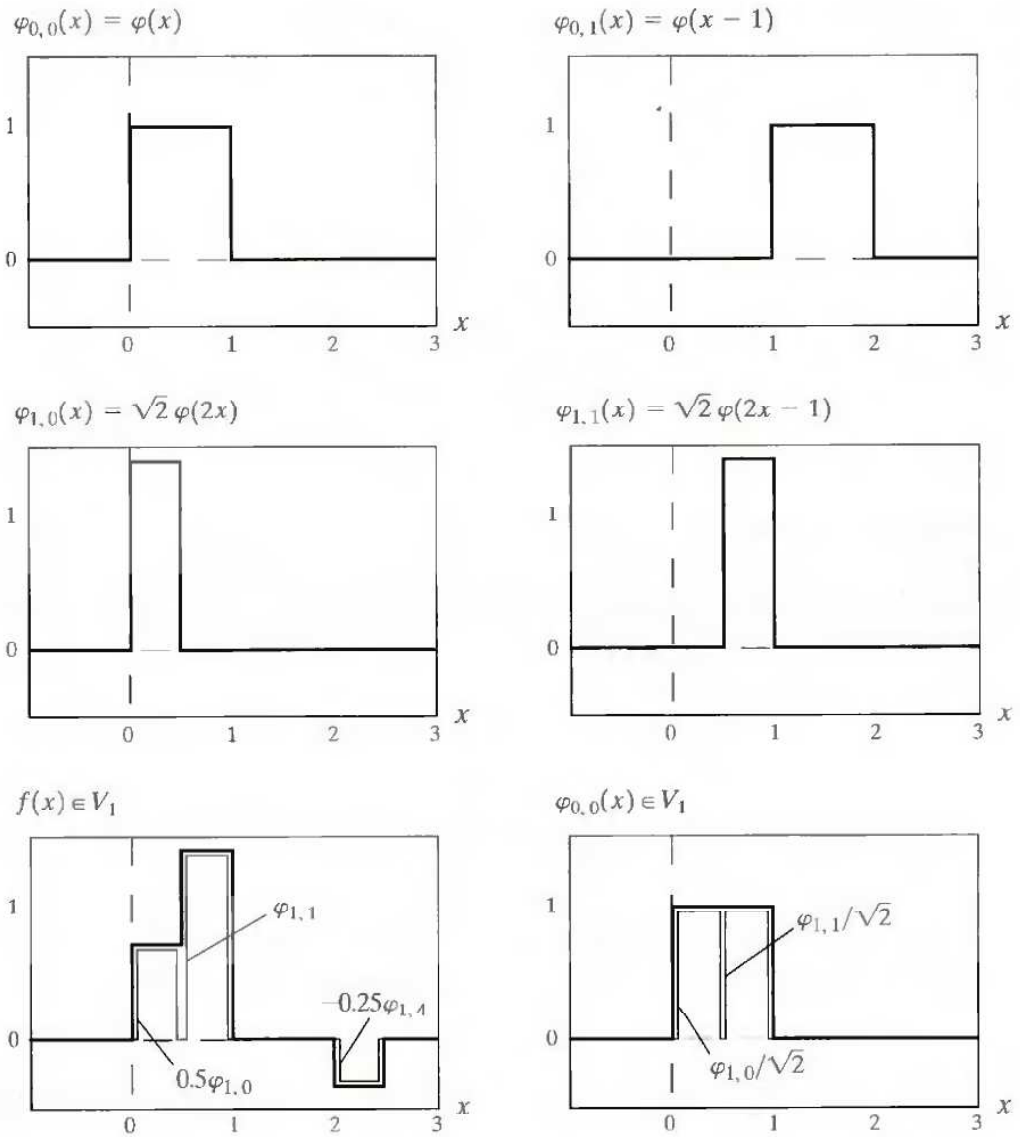
$$f(x) = 0.5\varphi_{1,0}(x) + \varphi_{1,1}(x) - 0.25\varphi_{1,4}(x).$$

To conclude the example, Fig. 7.9(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 a part of 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]):



a b
c d
e f

FIGURE 7.9 Haar scaling functions in V_0 in V_1.

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

This is easy to see in the case of the Haar function, since 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 is 0 everywhere outside a finite interval called the support. In fact, its 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 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. 7.10, subspaces containing high-resolution functions must also contain all lower resolution functions. That is,

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

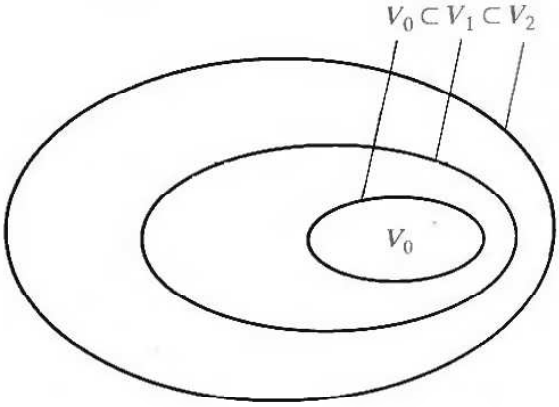


FIGURE 7.10 The nested function spaces spanned by a scaling function.

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 should not be taken to indicate that any function with a support 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. 7.9(e), all measurable, square-integrable functions can be represented in the limit as $j \to \infty$. That is,

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

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

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

Since $\varphi(x) = \varphi_{0,0}(x)$, both j and k can be set to 0 to obtain the simpler non-subscripted 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*. The alternate notations $h(n)$ and $h_0(n)$ are often used in the literature, but we wish to avoid confusion with our earlier discussion of subband analysis filters. 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. (7.1-28). 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. 7.9(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. 7.11. We define the set $\{\psi_{j,k}(x)\}$ of wavelets

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

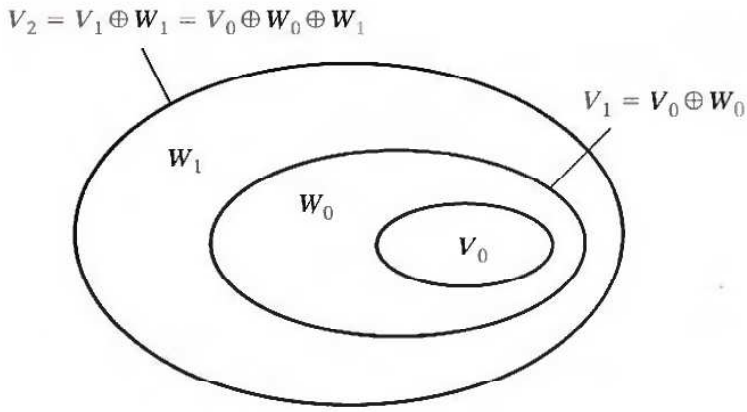


FIGURE 7.11 The relationship between scaling and wavelet function spaces.

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

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

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. 7.11 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 $j, k, l \in \mathbf{Z}$.

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. 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

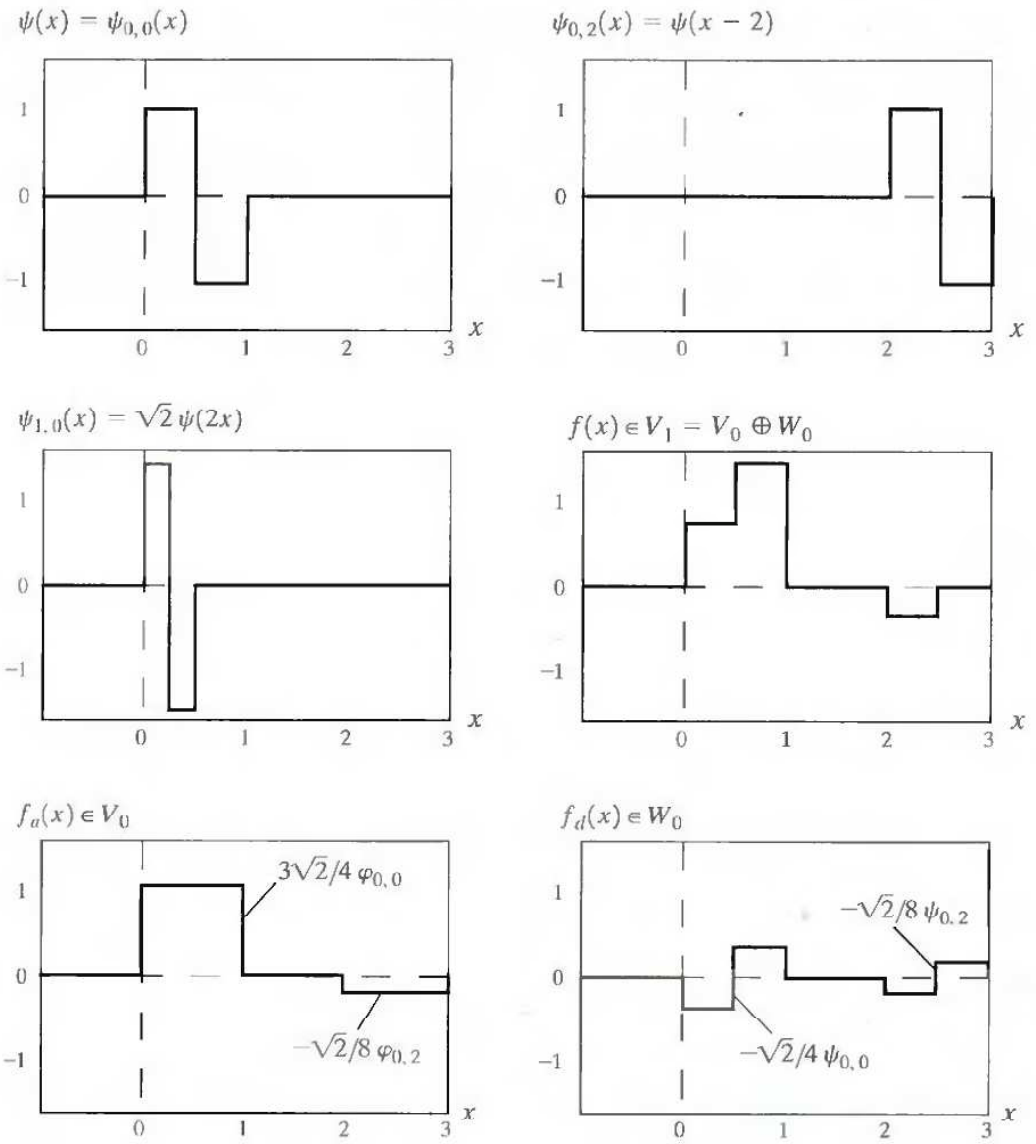
$$L^2(\mathbf{R}) = V_{j_0} \oplus W_{j_0} \oplus W_{j_0+1} \oplus \ldots \tag{7.2-27}$$

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. 7.11), 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* [the notation $h_1(n)$ is often used in the literature]. Using the condition that wavelets span the orthogonal complement spaces in Fig. 7.11, 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])



a b
c d
e f

FIGURE 7.12 Haar wavelet functions in W_0 and W_1.

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

Note the similarity of this result and Eq. (7.1-23), 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. (7.1-28). Substituting these values into Eq. (7.2-28), we get $\psi(x) = \varphi(2x) - \varphi(2x-1)$, which is plotted in Fig. 7.12(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 with 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. 7.12(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 7.12(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. 7.9(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

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

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. 7.12(e) and (f), divide $f(x)$ in a manner similar to a lowpass and highpass filter. 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 will define 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), we can write

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

where j_0 is an arbitrary starting scale and the $c_{j_0}(k)$'s and $d_j(k)$'s are relabeled α_k's from Eqs. (7.2-12) and (7.2-21), respectively. The $c_{j_0}(k)$'s are normally called the *approximation* or *scaling coefficients*; the $d_j(k)$'s are referred to as the *detail*

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 $f(x) \in V_{j_0}$ and it is exact]. 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

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

and

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

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, $\tilde{\varphi}$ and $\tilde{\psi}$, respectively.

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

■ Consider the simple function

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

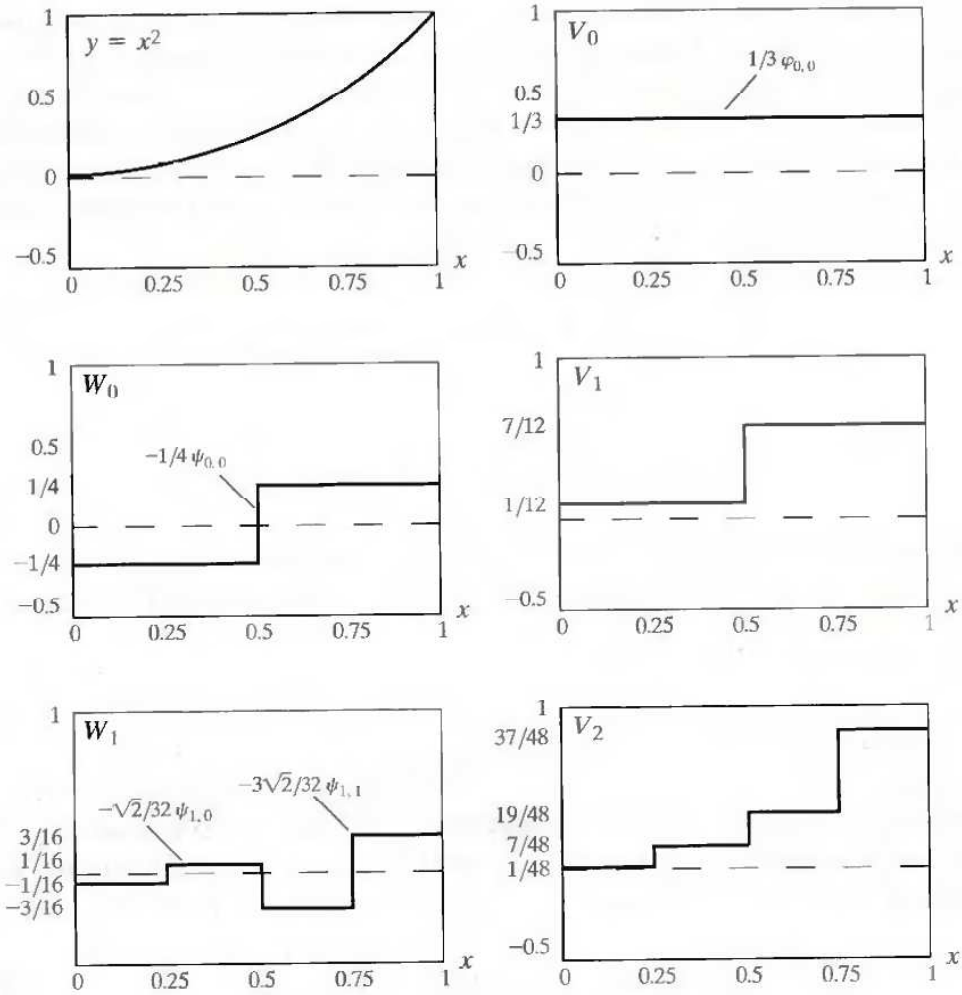
shown in Fig. 7.13(a). Using Haar wavelets—see Eqs. (7.2-14) and (7.2-30)—and a starting scale $j_0 = 0$, Eqs. (7.3-2) and (7.3-3) can be used to compute the following expansion coefficients:

$$\begin{aligned}
c_0(0) &= \int_0^1 x^2\varphi_{0,0}(x)\, dx = \int_0^1 x^2\, dx = \frac{x^3}{3}\bigg|_0^1 = \frac{1}{3} \\
d_0(0) &= \int_0^1 x^2\psi_{0,0}(x)\, dx = \int_0^{0.5} x^2\, dx - \int_{0.5}^1 x^2\, dx = -\frac{1}{4} \\
d_1(0) &= \int_0^1 x^2\psi_{1,0}(x)\, dx = \int_0^{0.25} x^2\sqrt{2}\, dx - \int_{0.25}^{0.5} x^2\sqrt{2}\, dx = -\frac{\sqrt{2}}{32} \\
d_1(1) &= \int_0^1 x^2\psi_{1,1}(x)\, dx = \int_{0.5}^{0.75} x^2\sqrt{2}\, dx - \int_{0.75}^1 x^2\sqrt{2}\, dx = -\frac{3\sqrt{2}}{32}
\end{aligned}$$

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

$$y = \underbrace{\underbrace{\underbrace{\frac{1}{3}\varphi_{0,0}(x)}_{V_0} + \underbrace{\left[-\frac{1}{4}\psi_{0,0}(x)\right]}_{W_0}}_{V_1 = V_0 \oplus W_0} + \underbrace{\left[-\frac{\sqrt{2}}{32}\psi_{1,0}(x) - \frac{3\sqrt{2}}{32}\psi_{1,1}(x)\right]}_{W_1}}_{V_2 = V_1 \oplus W_1 = V_0 \oplus W_0 \oplus W_1} + \cdots$$

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. 7.13(b) and is the average value of the original function. The second term



a b
c d
e f

FIGURE 7.13 A wavelet series expansion of $y = x^2$ using Haar wavelets.

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. 7.13(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. 7.13(e), and the resulting V_2 approximation is depicted in 7.13(f). Note that the expansion is now beginning to resemble the original function. As higher scales (or greater levels of detail) are added, the approximation becomes a more precise representation of the function, realizing it in the limit as $j \to \infty$. ■

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 a sequence of numbers, like samples of a continuous function $f(x)$, the resulting coefficients are called the *discrete wavelet*

transform (DWT) of $f(x)$. For this case, the series expansion defined in Eqs. (7.3-1) through (7.3-3) becomes the DWT transform pair

$$W_\varphi(j_0, k) = \frac{1}{\sqrt{M}} \sum_x f(x)\varphi_{j_0,k}(x) \tag{7.3-5}$$

$$W_\psi(j, k) = \frac{1}{\sqrt{M}} \sum_x f(x)\psi_{j,k}(x) \tag{7.3-6}$$

for $j \geq j_0$ and

$$f(x) = \frac{1}{\sqrt{M}} \sum_k W_\varphi(j_0, k)\varphi_{j_0,k}(x) + \frac{1}{\sqrt{M}} \sum_{j=j_0}^{\infty} \sum_k W_\psi(j, k)\psi_{j,k}(x). \tag{7.3-7}$$

Here, $f(x)$, $\varphi_{j_0,k}(x)$, and $\psi_{j,k}(x)$ are functions of the discrete variable $x = 0, 1, 2, \ldots, M - 1$. For example, $f(x) = f(x_0 + x\Delta x)$ for some x_0, Δx, and $x = 0, 1, 2, \ldots, M - 1$. Normally, we let $j_0 = 0$ and select M to be a power of 2 (i.e., $M = 2^J$) so that the summations are performed over $x = 0, 1, 2, \ldots, M - 1$, $j = 0, 1, 2, \ldots, J - 1$, 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 $W_\varphi(j_0, k)$'s and $W_\psi(j, k)$'s in Eqs. (7.3-5) to (7.3-7) correspond to the $c_{j_0}(k)$'s and $d_j(k)$'s 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 4.2.1, has been added to both the forward and inverse expressions. This factor could alternately 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 $f(3) = 0$. Since $M = 4$, $J = 2$ 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. Substituting the four samples into Eq. (7.3-5), we find that

$$\begin{aligned} W_\varphi(0, 0) &= \frac{1}{2} \sum_{x=0}^{3} f(x)\varphi_{0,0}(x) \\ &= \frac{1}{2}[1 \cdot 1 + 4 \cdot 1 - 3 \cdot 1 + 0 \cdot 1] = 1 \end{aligned}$$

because $\varphi_{0,0}(x) = 1$ for $x = 0, 1, 2, 3$. 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

$$W_\psi(0,0) = \frac{1}{2}\left[1 \cdot 1 + 4 \cdot 1 - 3 \cdot (-1) + 0 \cdot (-1)\right] = 4$$

$$W_\psi(1,0) = \frac{1}{2}\left[1 \cdot \sqrt{2} + 4 \cdot (-\sqrt{2}) - 3 \cdot 0 + 0 \cdot 0\right] = -1.5\sqrt{2}$$

$$W_\psi(1,1) = \frac{1}{2}\left[1 \cdot 0 + 4 \cdot 0 - 3 \cdot \sqrt{2} + 0 \cdot (-\sqrt{2})\right] = -1.5\sqrt{2}.$$

Thus, the discrete wavelet transform of our simple four-sample function relative to the Haar wavelet and scaling function is $\{1, 4, -1.5\sqrt{2}, -1.5\sqrt{2}\}$, 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

$$f(x) = \frac{1}{2}\left[W_\varphi(0,0)\varphi_{0,0}(x) + W_\psi(0,0)\psi_{0,0}(x) + W_\psi(1,0)\psi_{1,0}(x) + W_\psi(1,1)\psi_{1,1}(x)\right]$$

for $x = 0, 1, 2, 3$. If $x = 0$, for instance,

$$f(0) = \frac{1}{2}\left[1 \cdot 1 + 4 \cdot 1 - 1.5\sqrt{2} \cdot (\sqrt{2}) - 1.5\sqrt{2} \cdot 0\right] = 1.$$

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 $f(x)$—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 $\{2.5\sqrt{2}, -1.5\sqrt{2}, -1.5\sqrt{2}, -1.5\sqrt{2}\}$, which results when the starting scale is one. 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, it 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

$$W_\psi(s,\tau) = \int_{-\infty}^{\infty} f(x)\psi_{s,\tau}(x)\,dx \tag{7.3-8}$$

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

$$C_\psi = \int_{-\infty}^{\infty}\frac{|\Psi(u)|^2}{|u|}\,du \tag{7.3-11}$$

and $\Psi(u)$ 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 $\Psi(u) \to 0$ as $u \to \infty$ 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 $\psi((x-\tau)/s)$ 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,

$$\psi(x) = \left(\frac{2}{\sqrt{3}}\pi^{-1/4}\right)(1-x^2)e^{-x^2/2} \tag{7.3-12}$$

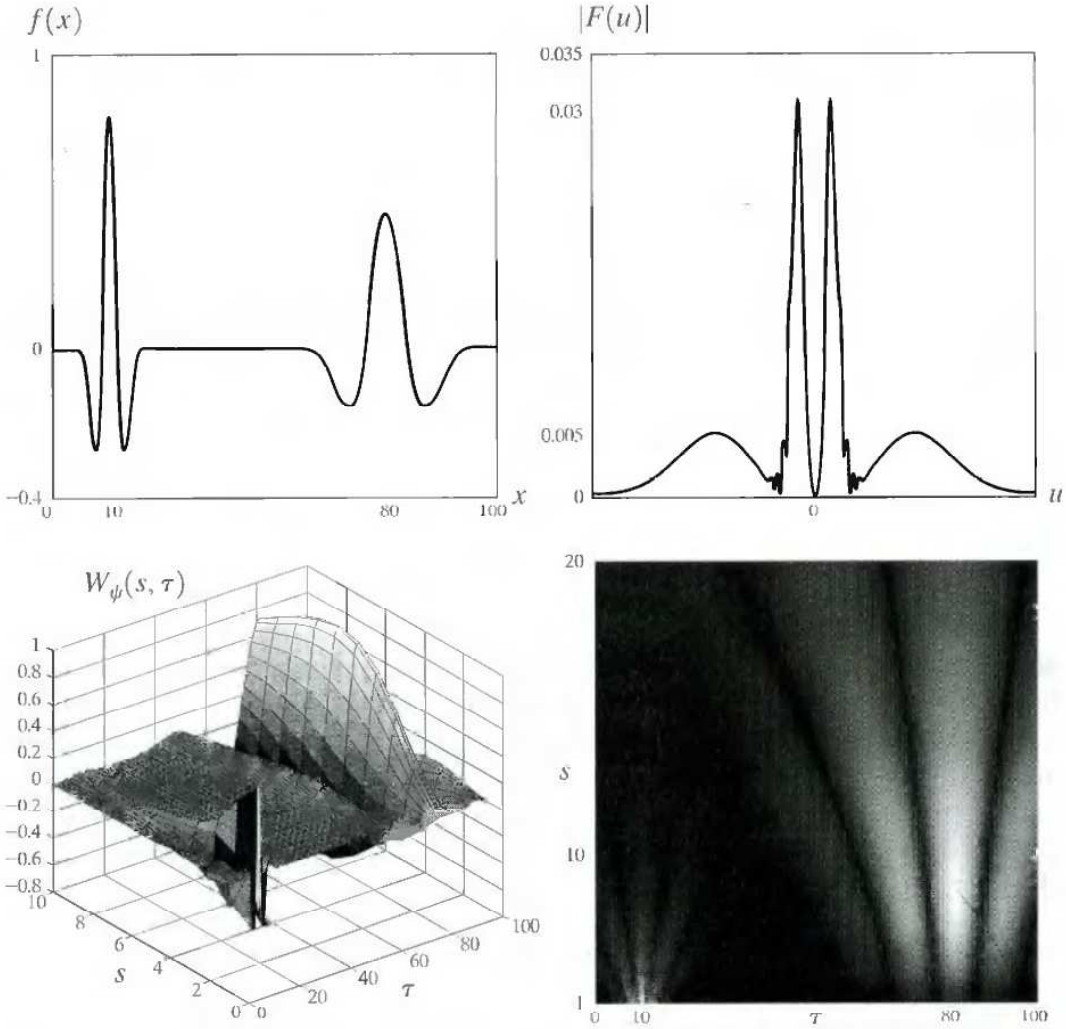


FIGURE 7.14 The continuous wavelet transform (c and d) and Fourier spectrum (b) of a continuous one-dimensional function (a).

gets its name from its distinctive shape [see Fig. 7.14(a)]. It is proportional to the second derivative of the Gaussian probability function, has an average value of zero, 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. 7.14(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. 7.14(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 function's two Gaussian-like perturbations.

Figure 7.14(c) shows a portion ($1 \leq s \leq 10$ and $\tau \leq 100$) of the CWT of the function in Fig. 7.14(a) relative to the Mexican hat wavelet. Unlike the Fourier

spectrum in Fig. 7.14(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. 7.14(d), where the absolute value of the transform, $|W_\psi(s, \tau)|$, is displayed as gray levels between black and white. Note that the continuous wavelet transform turns a one-dimensional function into a two-dimensional result. ■

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, b]), the FWT resembles the two-band subband coding scheme of Section 7.1.2.

Consider again the multiresolution refinement equation

$$\varphi(x) = \sum_n h_\varphi(n)\sqrt{2}\varphi(2x - n). \tag{7.4-1}$$

Scaling x by 2^j, translating it by k, and letting $m = 2k + n$ gives

$$\begin{aligned}\varphi(2^j x - k) &= \sum_n h_\varphi(n)\sqrt{2}\varphi(2(2^j x - k) - n)\\ &= \sum_m h_\varphi(m - 2k)\sqrt{2}\varphi(2^{j+1}x - m).\end{aligned} \tag{7.4-2}$$

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) is replaced by wavelet vector $h_\psi(n)$ in Eq. (7.4-3).

Now consider Eqs. (7.3-5) and (7.3-6) of Section 7.2.2. They define the discrete wavelet transform. Substituting Eq. (7.2-19)—the wavelet defining equation—into Eq. (7.3-6), we get

$$W_\psi(j, k) = \frac{1}{\sqrt{M}} \sum_x f(x)2^{j/2}\psi(2^j x - k), \tag{7.4-4}$$

which, upon replacing $\psi(2^j x - k)$ with the right side of Eq. (7.4-3), becomes

$$W_\psi(j, k) = \frac{1}{\sqrt{M}} \sum_x f(x)2^{j/2}\left[\sum_m h_\psi(m - 2k)\sqrt{2}\varphi(2^{j+1}x - m)\right]. \tag{7.4-5}$$

Interchanging the sum and integral and rearranging terms then gives

$$W_\psi(j,k) = \sum_m h_\psi(m-2k)\left[\frac{1}{\sqrt{M}}\sum_x f(x)2^{(j+1)/2}\varphi(2^{j+1}x-m)\right] \tag{7.4-6}$$

where the bracketed quantity is identical to Eq. (7.3-5) with $j_0 = j+1$. To see this, substitute Eq. (7.2-10) into Eq. (7.3-5) and let j_0 be $j+1$. We can therefore write

$$W_\psi(j,k) = \sum_m h_\psi(m-2k)W_\varphi(j+1,m) \tag{7.4-7}$$

and note that the DWT detail coefficients at scale j are a function of the DWT approximation coefficients at scale $j+1$. Recognizing Eqs. (7.4-2) and (7.3-5) as the starting point of a similar derivation involving the DWT approximation coefficients, we find similarly that

$$W_\varphi(j,k) = \sum_m h_\varphi(m-2k)W_\varphi(j+1,m). \tag{7.4-8}$$

Equations (7.4-7) and (7.4-8) 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 time-reversed scaling and wavelet vectors, $h_\varphi(-n)$ and $h_\psi(-n)$, and subsampling the results. Figure 7.15 reduces these operations to block diagram form. Note that it is identical to the analysis portion of the two-band subband coding and decoding system of Fig. 7.4, with $h_0(n) = h_\varphi(-n)$ and $h_1(n) = h_\psi(-n)$. We can therefore write

$$W_\psi(j,k) = h_\psi(-n) * W_\varphi(j+1,n)\Big|_{n=2k,\,k\geq 0} \tag{7.4-9}$$

and

$$W_\varphi(j,k) = h_\varphi(-n) * W_\varphi(j+1,n)\Big|_{n=2k,\,k\geq 0} \tag{7.4-10}$$

where the convolutions are evaluated at instants $n = 2k$ for $k \geq 0$. Evaluating convolutions at nonnegative, even indices is equivalent to filtering and downsampling by 2.

To conclude the development of the FWT, we simply note that the filter bank in Fig. 7.15 can be "iterated" to create multistage structures for computing DWT

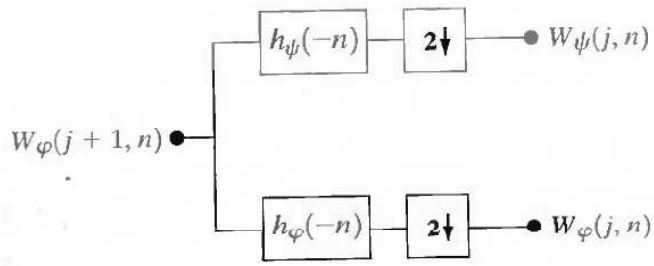


FIGURE 7.15 An FWT analysis bank.

highest-resolution scaling functions act as "delta functions" in Eqs. (7.3-5) and (7.3-6), allowing $f(n)$ to be used as the scale J approximation or scaling coefficient 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 these concepts, consider the discrete function $f(n) = \{1, 4, -3, 0\}$ from Example 7.8. As in that example, we will recompute 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, but the corresponding scaling and wavelet vectors from Examples 7.5 and 7.6:

$$h_\varphi(n) = \begin{cases} 1/\sqrt{2} & n = 0, 1 \\ 0 & \text{otherwise} \end{cases} \tag{7.4-11}$$

and

$$h_\psi(n) = \begin{cases} 1/\sqrt{2} & n = 0 \\ -1/\sqrt{2} & n = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{7.4-12}$$

These are the functions used to build the FWT filter banks; they provide the filter coefficients.

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. 7.16(a). Figure 7.17 shows the sequences that result from the required FWT convolutions and downsamplings. Note that function $f(n)$ itself is the scaling or

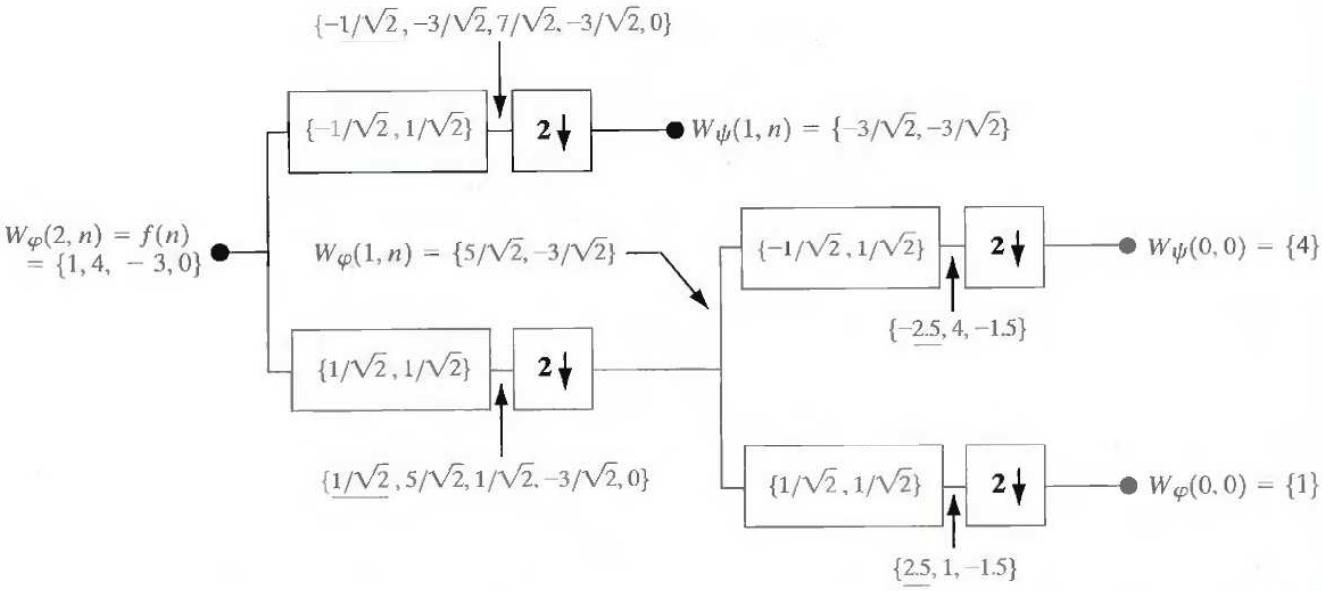


FIGURE 7.17 Computing a two-scale fast wavelet transform of sequence $\{1, 4, -3, 0\}$ using Haar scaling and wavelet vectors.