

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 relatively 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 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.

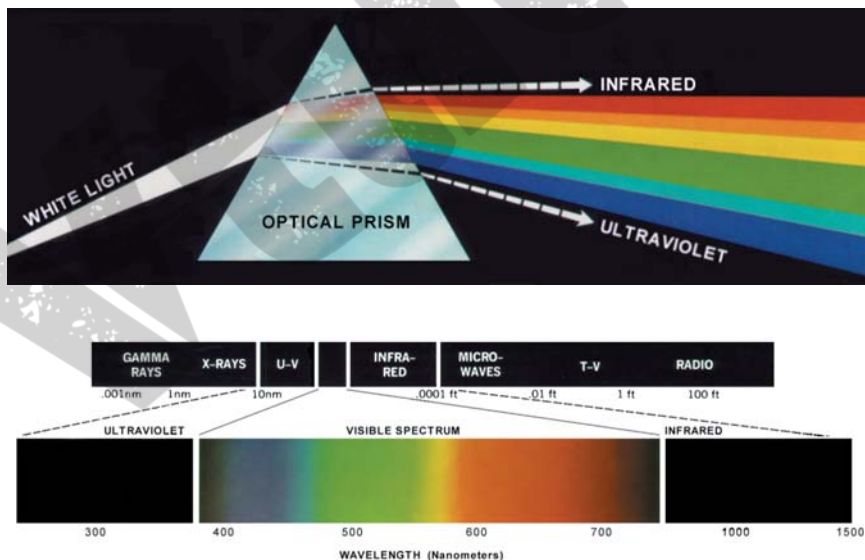


FIGURE 6.1 Color spectrum seen by passing white light through a prism. (Courtesy of the General Electric Co., Lamp Business Division.)

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

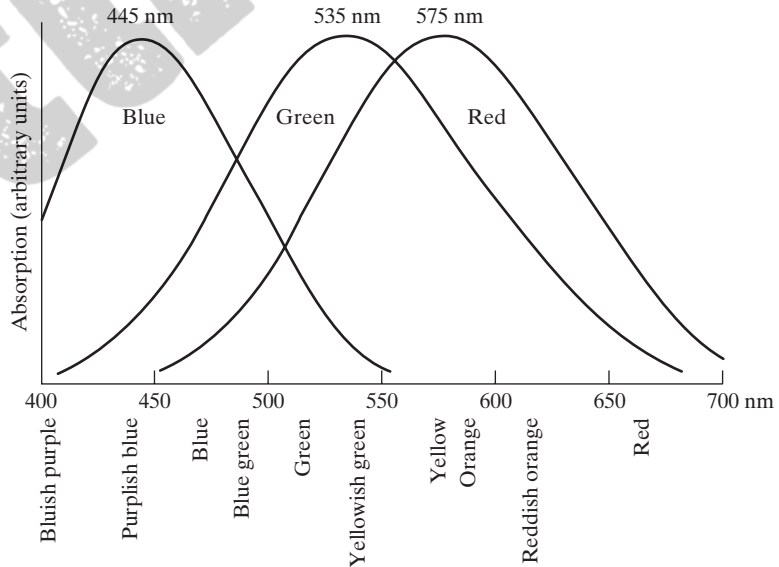
Characterization of light is central to the science of color. If the light is achromatic (void of color), its only attribute is its *intensity*, or amount. Achromatic light is what viewers see on a black and white television set, and it has been an implicit component of our discussion of image processing 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.

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

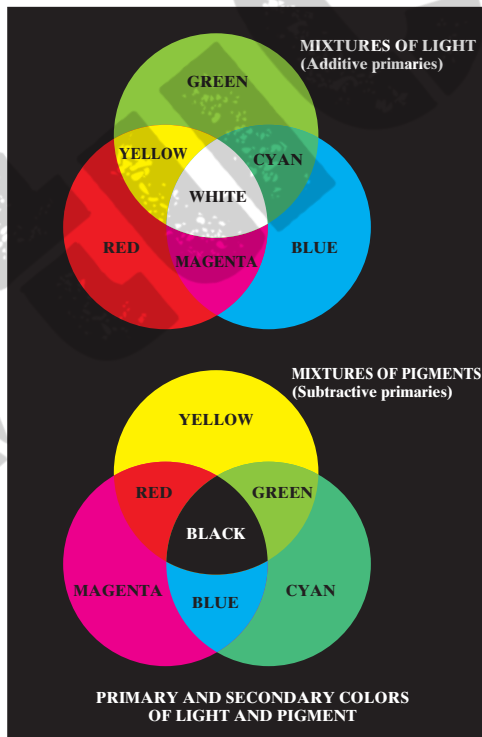
FIGURE 6.3

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



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 in 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 you will see shortly, this interpretation is not 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.



a
b

FIGURE 6.4

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

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 CRT (cathode ray tube) color TV screens is composed of a large array of triangular dot patterns of electron-sensitive phosphor. When excited, each dot in a triad produces 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 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.

CRT displays are being replaced by “flat panel” digital technologies, such as *liquid crystal displays* (LCDs) and *plasma* devices. Although they are fundamentally different from CRTs, these and similar technologies use the same principle in the sense that they all require three subpixels (red, green, and blue) to generate a single color pixel. LCDs use properties of polarized light to block or pass light through the LCD screen and, in the case of active matrix display technology, thin film transistors (TFTs) are used to provide the proper signals to address each pixel on the screen. Light filters are used to produce the three primary colors of light at each pixel triad location. In plasma units, pixels are tiny gas cells coated with phosphor to produce one of the three primary colors. The individual cells are addressed in a manner analogous to LCDs. This individual pixel triad coordinate addressing capability is the foundation of digital displays.

The characteristics generally used to distinguish one color from another are *brightness*, *hue*, and *saturation*. As indicated earlier in this section, brightness embodies the achromatic 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 referring to 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

$$x = \frac{X}{X + Y + Z} \quad (6.1-1)$$

$$y = \frac{Y}{X + Y + Z} \quad (6.1-2)$$

and

$$z = \frac{Z}{X + Y + Z} \quad (6.1-3)$$

It is noted from these equations that[†]

$$x + y + z = 1 \quad (6.1-4)$$

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%.

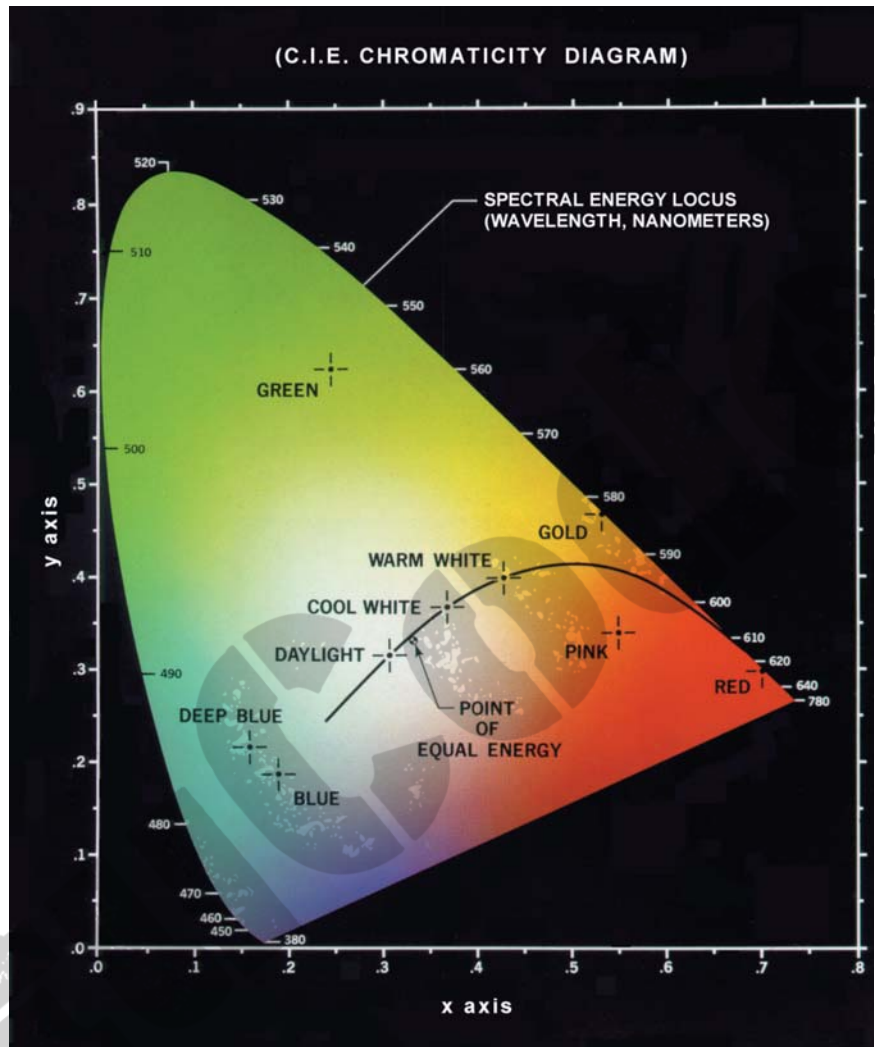
The positions of the various spectrum colors—from violet at 380 nm to red at 780 nm—are indicated around the boundary of the tongue-shaped chromaticity diagram. These are the pure colors shown in the spectrum of Fig. 6.2. Any point not actually on the boundary but within the diagram represents some mixture 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

[†]The use of 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.)



energy to any point on the boundary of the chart will define all the shades of that particular spectrum color.

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 on the boundary or 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

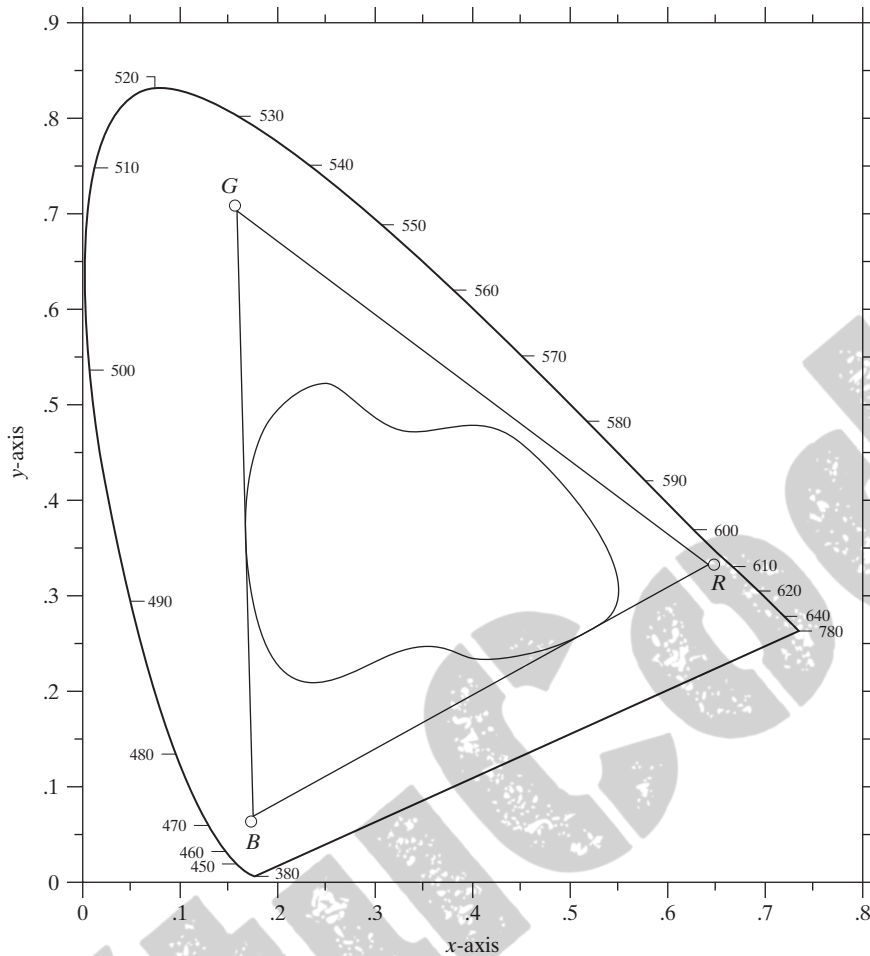


FIGURE 6.6
Typical color gamut of color monitors (triangle) and color printing devices (irregular region).

devices. 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, you 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 primary values are at three corners; the secondary colors 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 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 $[0, 1]$.

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 $(1, 1, 1)$.

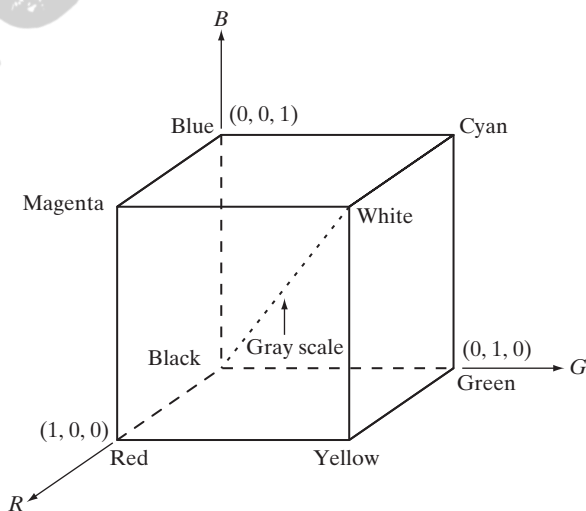




FIGURE 6.8 RGB
24-bit color cube.

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 screen to produce a composite color image, as explained in Section 6.1. 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.

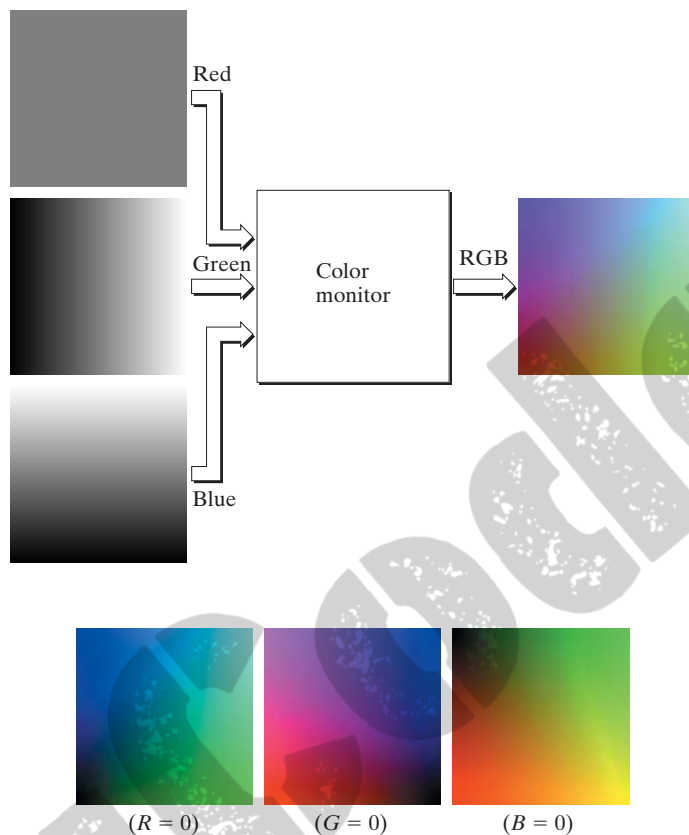
■ The cube shown in Fig. 6.8 is a solid, composed of the $(2^8)^3 = 16,777,216$ colors mentioned in the preceding paragraph. A convenient way to view these colors is to generate color planes (faces or cross sections of the cube). This is accomplished simply by fixing one of the three colors and allowing the other two to vary. For instance, a cross-sectional plane through the center of the cube and parallel to the *GB*-plane in Fig. 6.8 is the plane $(127, G, B)$ for $G, B = 0, 1, 2, \dots, 255$. 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. 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. ■

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



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. 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.

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.

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 FFFFFFFF 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 FFFFFFFF (white), the second square to its right has value FFFFCC, the third square has value FFFF99, and

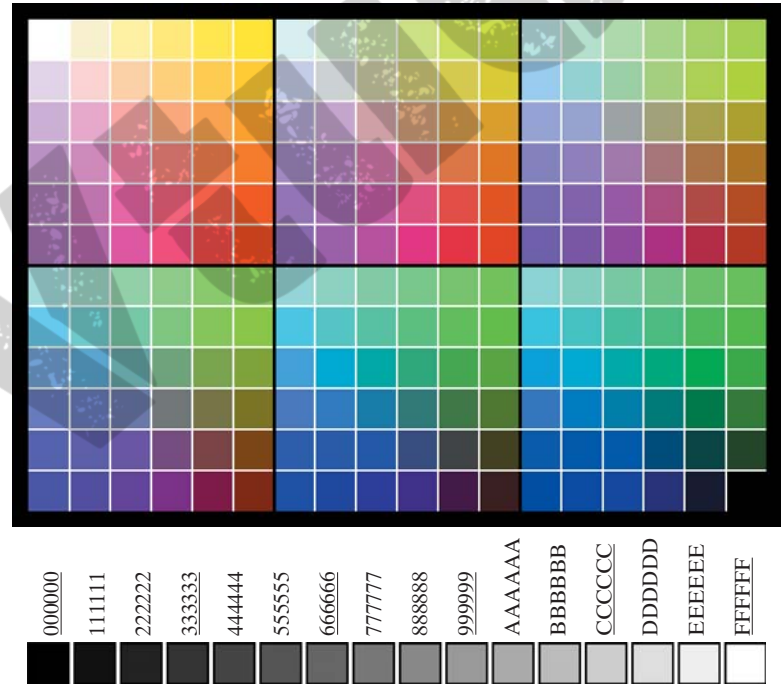


FIGURE 6.10
(a) The 216 safe RGB colors.
(b) All the grays in the 256-color RGB system (grays that are part of the safe color group are shown underlined).

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 of 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

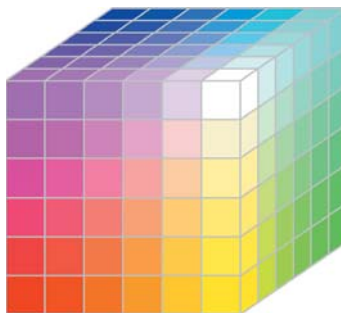
As indicated in Section 6.1, cyan, magenta, and yellow are the secondary colors of light or, alternatively, the primary colors of pigments. For example, when a surface coated with cyan pigment is illuminated with white light, no red light is reflected from the surface. That is, cyan subtracts red light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

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

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

FIGURE 6.11
The RGB safe-color cube.



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.

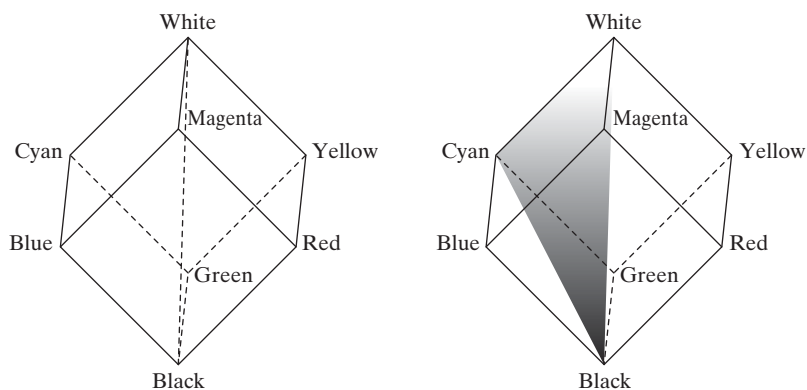
When humans view a color object, we describe it by its hue, saturation, and brightness. Recall from the discussion in Section 6.1 that hue is a color attribute that describes a pure color (pure yellow, orange, or red), whereas saturation gives a measure of the degree to which a pure color is diluted 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 an 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 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 in the HSI color model by working out the geometrical formulas describing the reasoning outlined in the preceding discussion.

a b

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



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 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 in terms of the hexagon just discussed, a triangle, or even a circle, as Figs. 6.13(c) and (d) show. The shape chosen does not matter because 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.

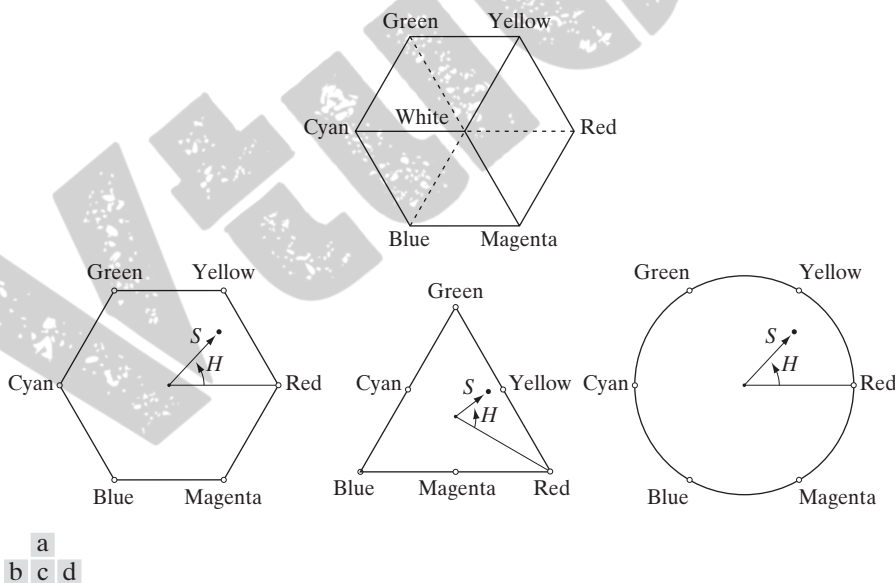
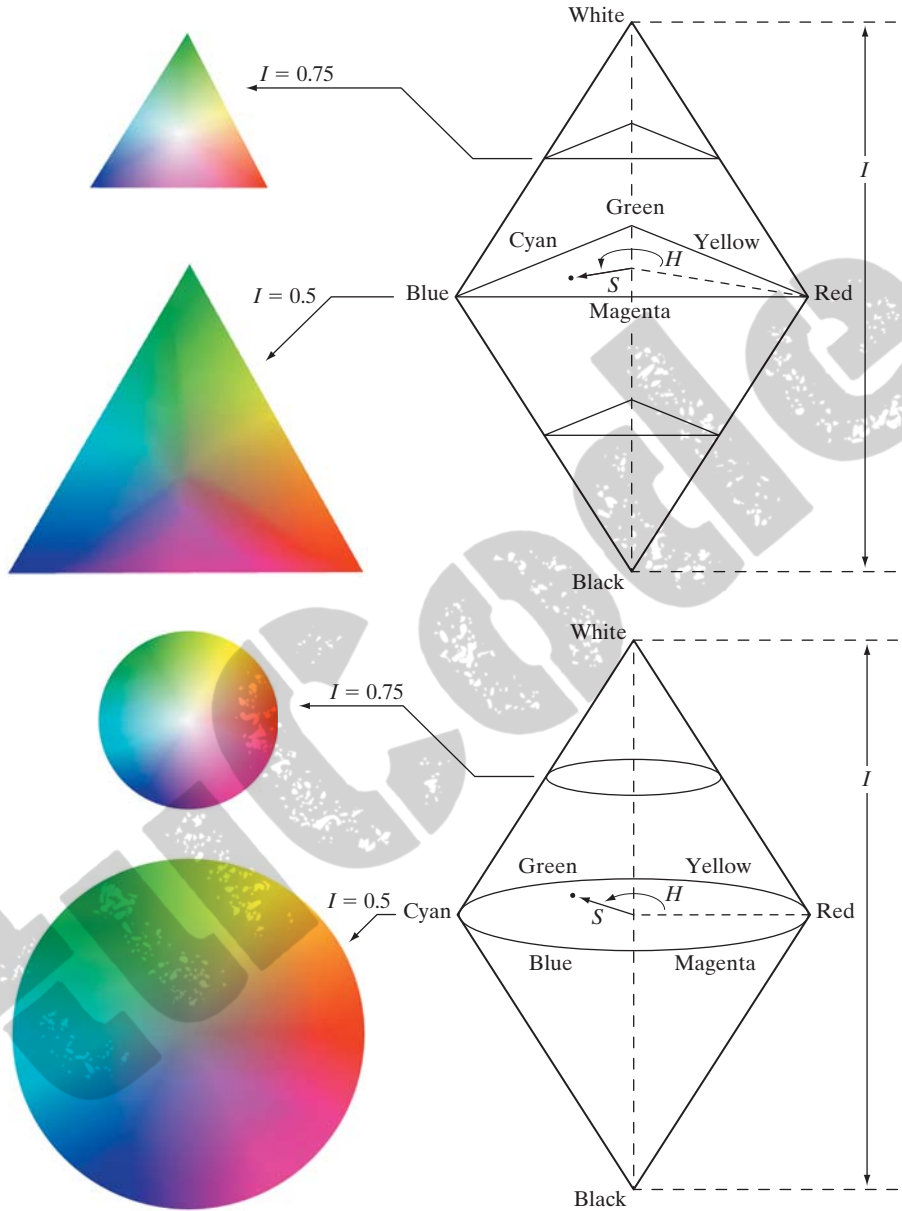


FIGURE 6.13 Hue and saturation in the HSI color model. The dot is an arbitrary color point. The angle from the red axis gives the hue, and the length of the vector is the saturation. The intensity of all colors in any of these planes is given by the position of the plane on the vertical intensity axis.

a
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

Computations from RGB to HSI and back are carried out on a per-pixel basis. We omitted the dependence on (x, y) of the conversion equations for notational clarity.

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

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

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

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

Finally, the intensity component is given by

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

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.

Consult the Tutorials section of 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 ($0^\circ \leq H < 120^\circ$): When H is in this sector, the RGB components are given by the equations

$$B = I(1 - S) \quad (6.2-5)$$

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

and

$$G = 3I - (R + B) \quad (6.2-7)$$

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

$$H = H - 120^\circ \quad (6.2-8)$$

[†]It is good practice to add a small number in the denominator of this expression to avoid dividing by 0 when $R = G = B$, in which case θ will be 90°. Note that when all RGB components are equal, Eq. (6.2-3) gives $S = 0$. In addition, the conversion from HSI back to RGB in Eqs. (6.2-5) through (6.2-7) will give $R = G = B = I$, as expected, because when $R = G = B$, we are dealing with a gray-scale image.

Then the RGB components are

$$R = I(1 - S) \quad (6.2-9)$$

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

and

$$B = 3I - (R + G) \quad (6.2-11)$$

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

$$H = H - 240^\circ \quad (6.2-12)$$

Then the RGB components are

$$G = I(1 - S) \quad (6.2-13)$$

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

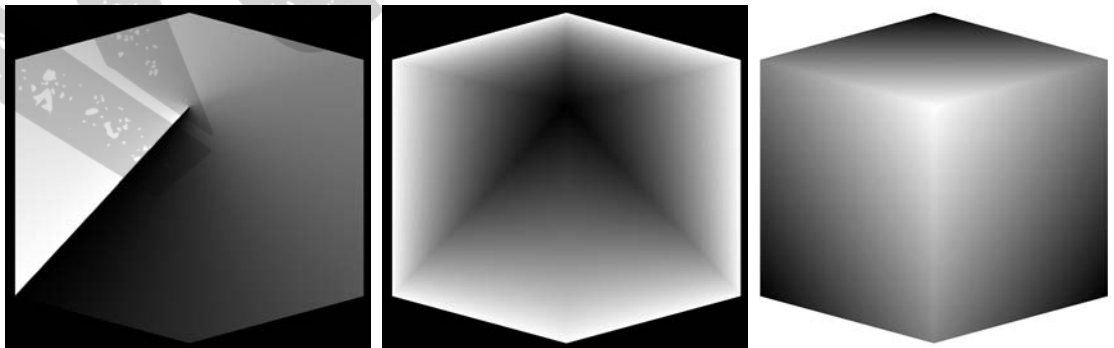
and

$$R = 3I - (G + B) \quad (6.2-15)$$

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°



a b c

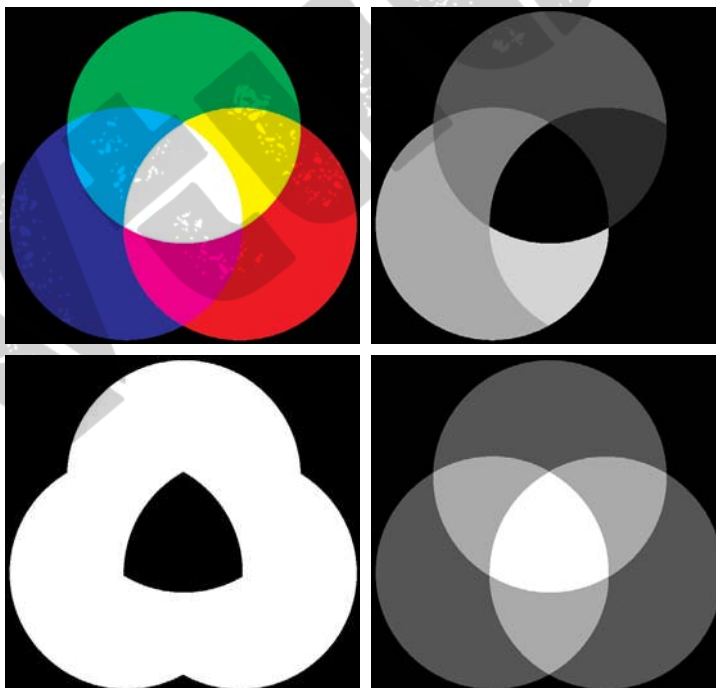
FIGURE 6.15 HSI components of the image in Fig. 6.8. (a) Hue, (b) saturation, and (c) intensity images.

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 saturated as they approach white. Finally, every pixel in the intensity image shown in Fig. 6.15(c) 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 you develop familiarity with these components and also help you deepen your 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, 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) is 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.



a	b
c	d

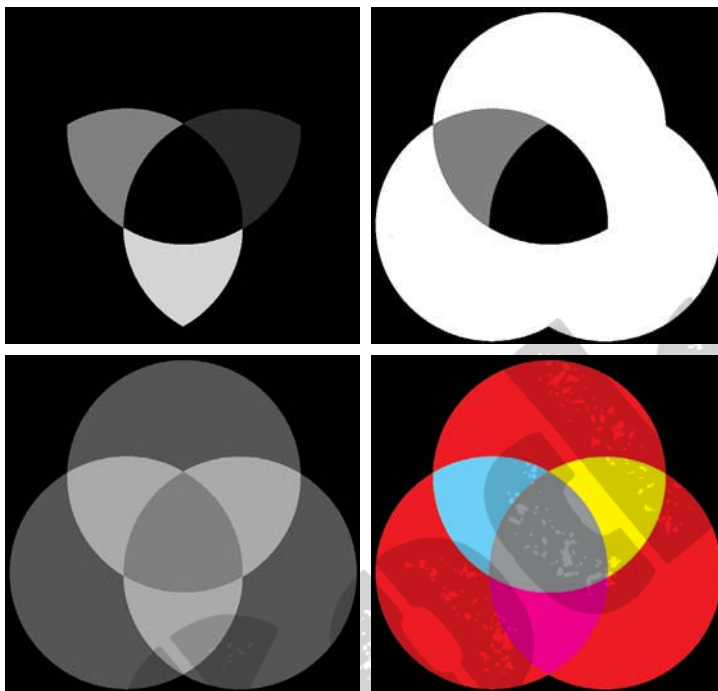
FIGURE 6.16

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

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.)



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 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 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 [see Fig. 2.18(a)], 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 intensity 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 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 intensity axis.

In general, the technique may be summarized as follows. Let $[0, L - 1]$ represent the gray scale, 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, \dots, l_P . Then, assuming that $0 < P < L - 1$, the P planes partition the gray scale into $P + 1$ intervals, V_1, V_2, \dots, V_{P+1} . Intensity to color assignments are made according to the relation

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

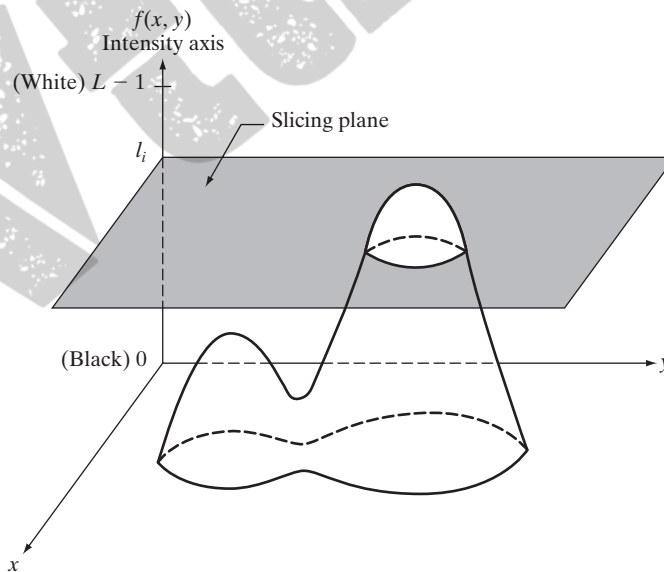
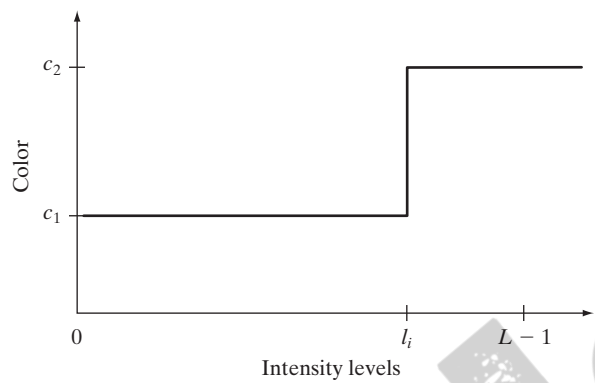


FIGURE 6.18
Geometric
interpretation of
the intensity-
slicing technique.

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



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

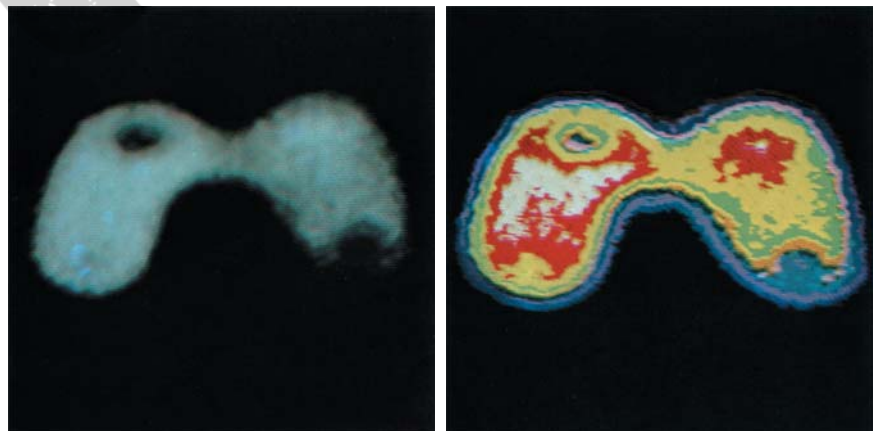
The idea of planes is useful primarily for a geometric interpretation of the intensity-slicing technique. Figure 6.19 shows an alternative representation that defines the same mapping as in Fig. 6.18. According to the mapping function shown in Fig. 6.19, any input intensity level is assigned one of two 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.

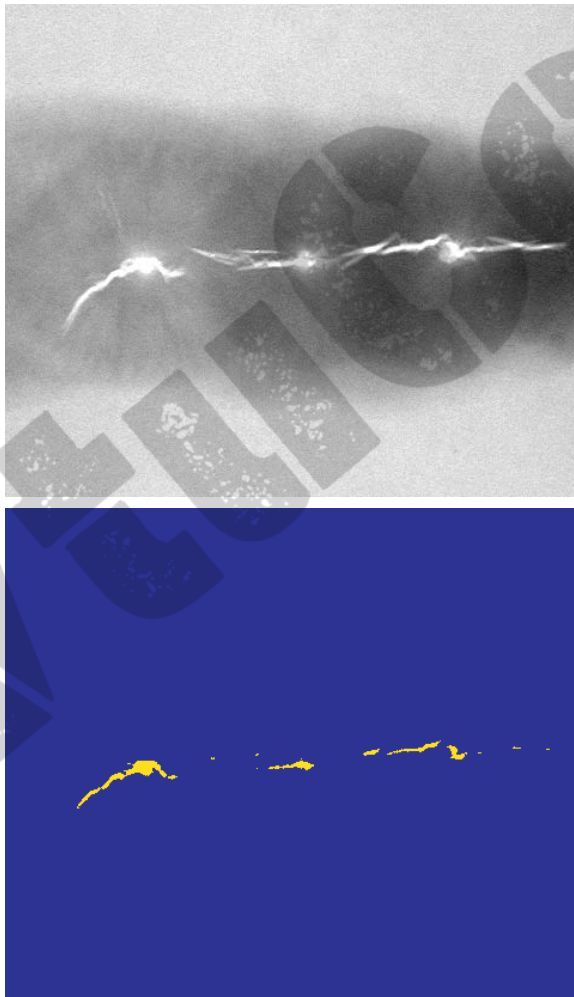
■ A simple, but practical, use of intensity slicing is shown in Fig. 6.20. Figure 6.20(a) is a monochrome image of the Picker Thyroid Phantom (a radiation test pattern), and Fig. 6.20(b) is the result of intensity slicing this image into eight color regions. Regions that appear of constant intensity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

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.)



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 (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. Thus, intensity values of 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



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.)

level 255 and another to all other intensity 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 intensity value or range of values one is looking for is known, intensity slicing is a simple but powerful aid in visualization, especially if numerous images are involved. The following is a 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 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 intensity 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 Intensity 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 intensity 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

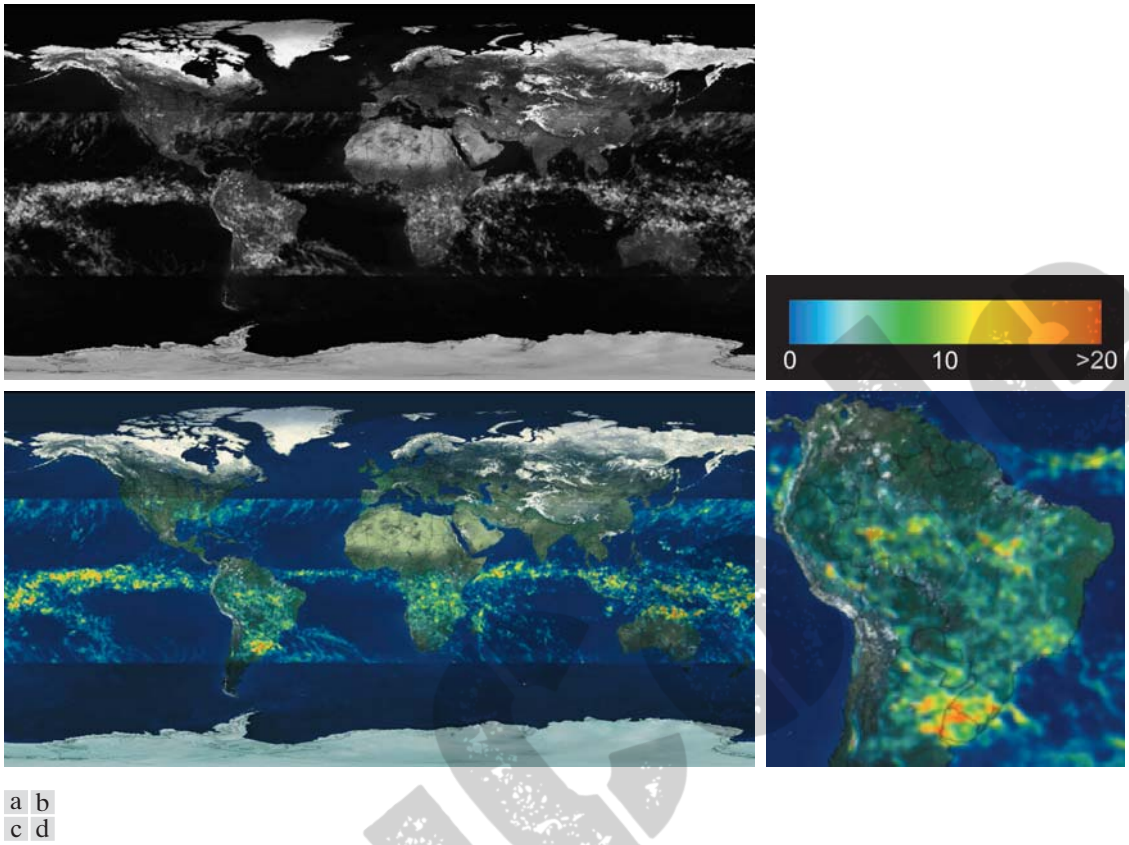


FIGURE 6.22 (a) Gray-scale image in which intensity (in the lighter horizontal band shown) corresponds to average monthly rainfall. (b) Colors assigned to intensity values. (c) Color-coded image. (d) Zoom of the South American region. (Courtesy of NASA.)

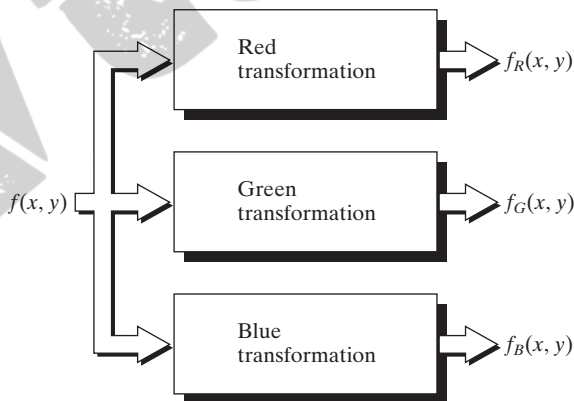


FIGURE 6.23 Functional block diagram for pseudocolor image processing. f_R , f_G , and f_B are fed into the corresponding red, green, and blue inputs of an RGB color monitor.

functions. Note that these are transformations on the intensity 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 intensity 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.

EXAMPLE 6.5:

Use of pseudocolor for highlighting explosives contained in luggage.

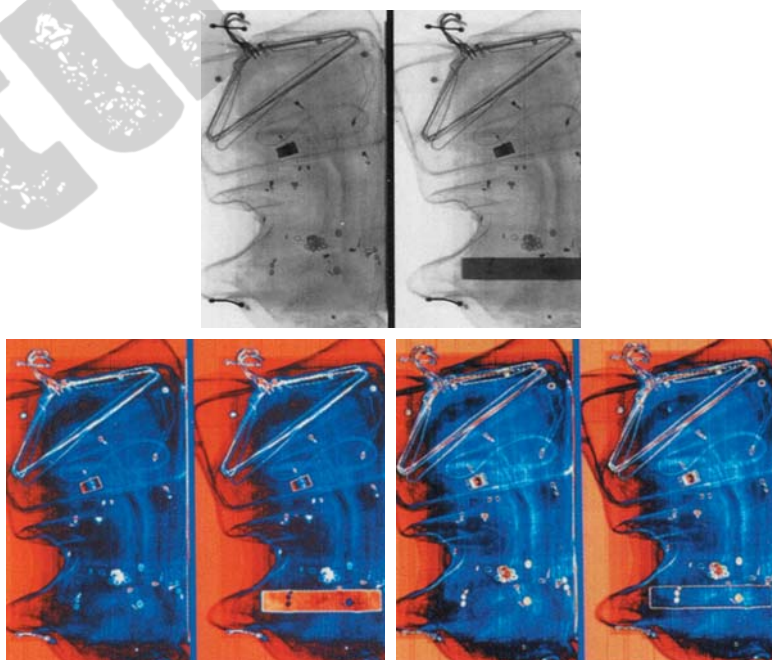
■ Figure 6.24(a) shows two monochrome images of luggage obtained from an airport X-ray scanning system. The image on the left contains ordinary articles. The image on the right contains the same articles, as well as a block of simulated plastic explosives. The purpose of this example is to illustrate the use of intensity 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 intensities correspond to peaks in the sinusoids, especially if the sinusoids have broad profiles (low frequencies). Pixels with intensity 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.

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.)



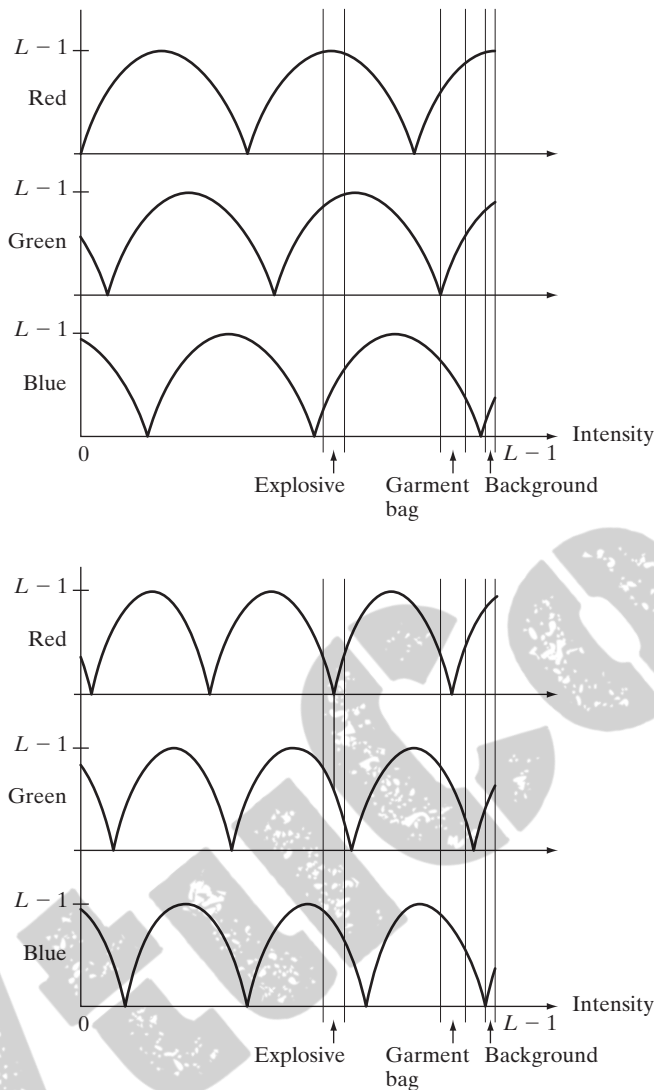
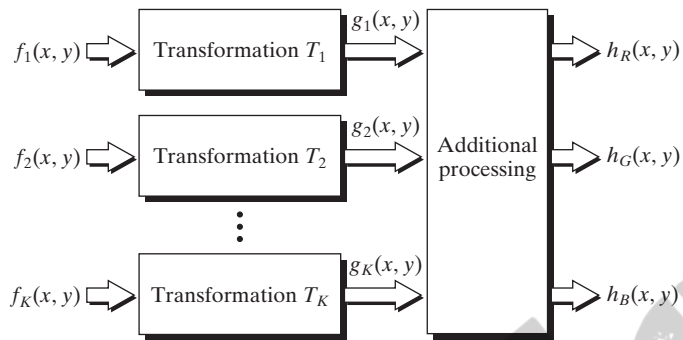


FIGURE 6.25
Transformation
functions used to
obtain the images
in Fig. 6.24.

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 intensity levels, but they were both coded with 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.

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 are 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

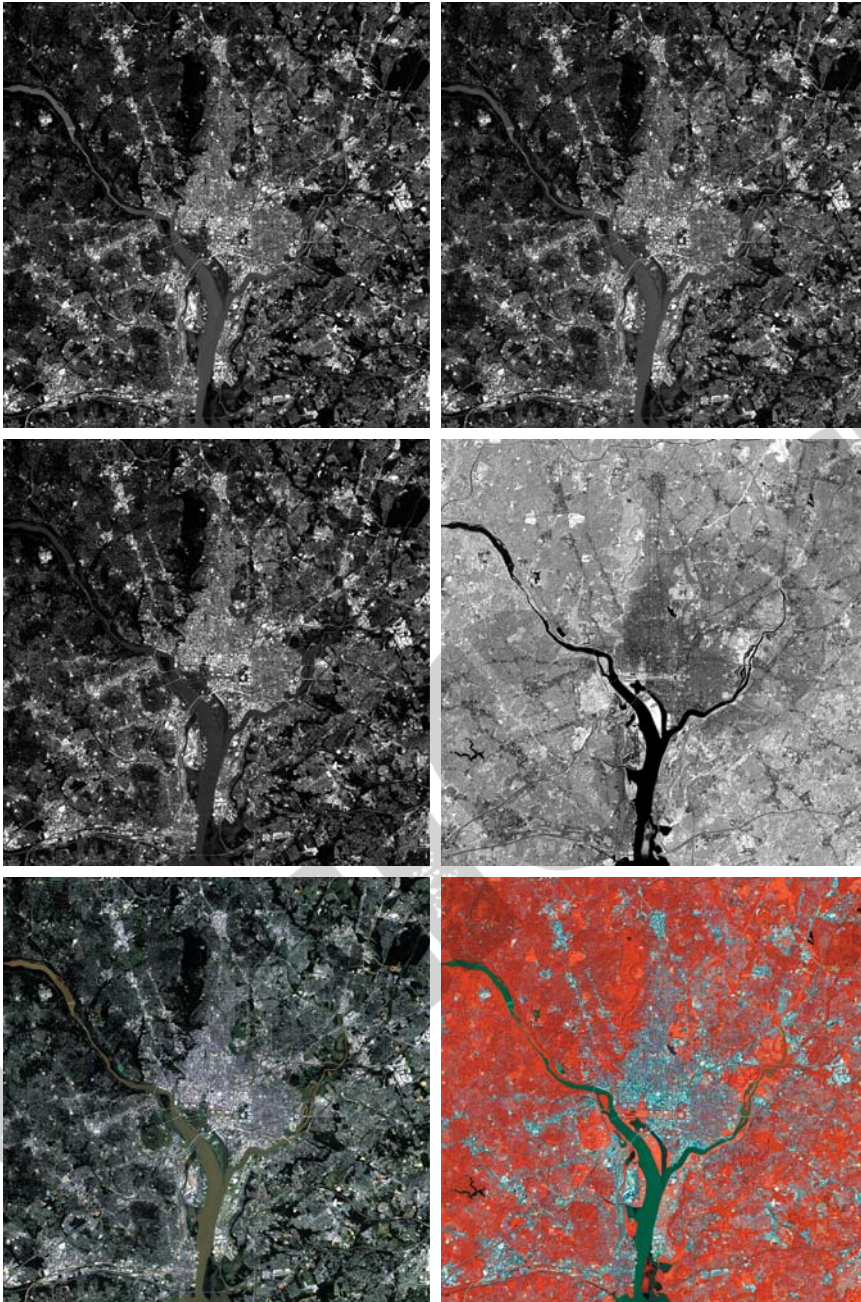


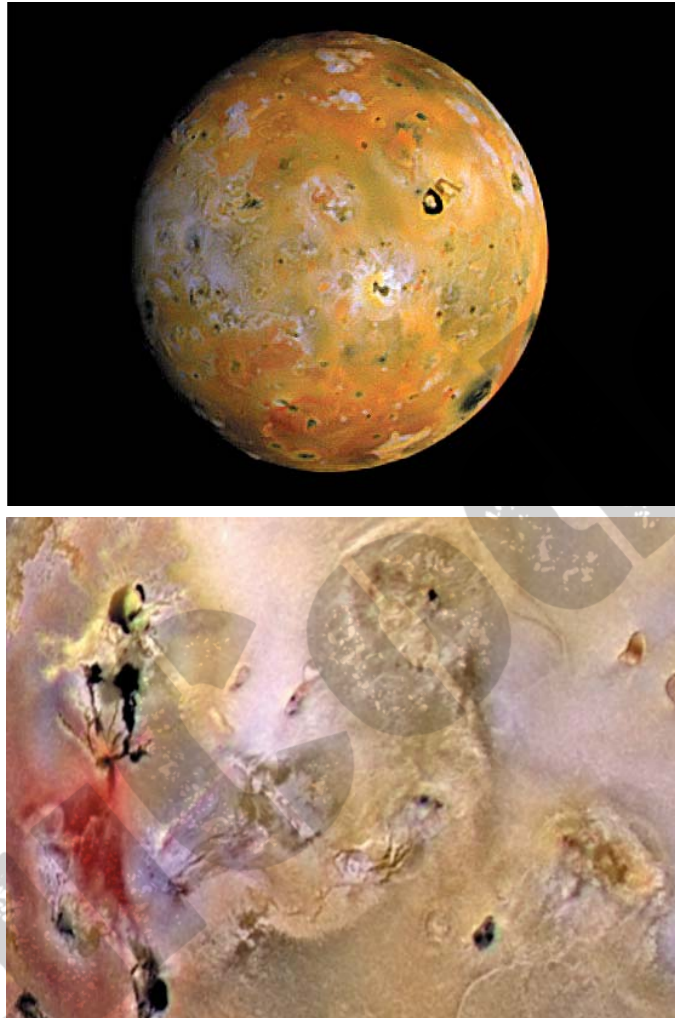
FIGURE 6.27 (a)–(d) Images in bands 1–4 in Fig. 1.10 (see Table 1.1). (e) Color composite image obtained by treating (a), (b), and (c) as the red, green, blue components of an RGB image. (f) Image obtained in the same manner, but using in the red channel the near-infrared image in (d). (Original multispectral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

a	b
c	d
e	f

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. ■

7.1 Background

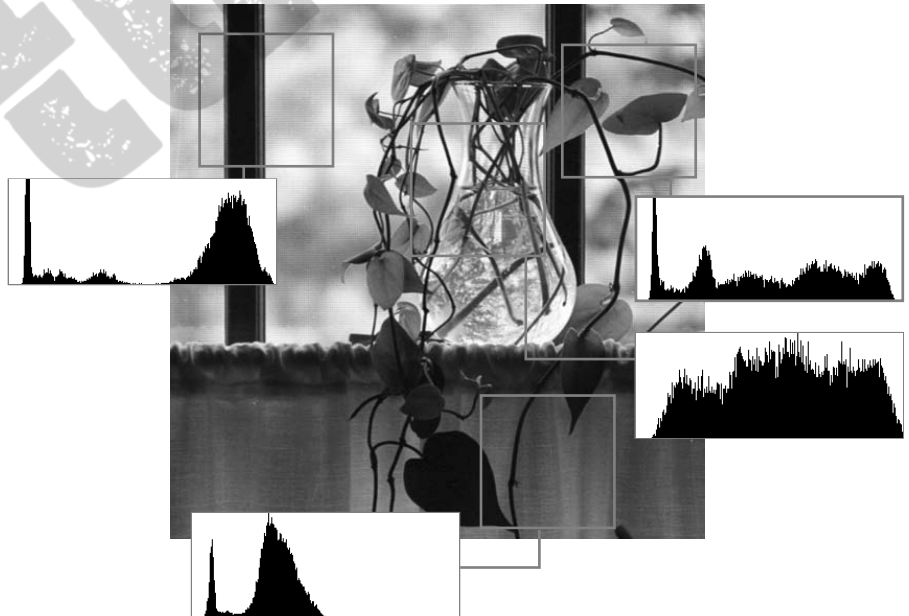
When we look at images, generally we see connected regions of similar texture and intensity levels that combine to form objects. If the objects are small in size or low in contrast, we normally examine them at high resolutions; if they are large in size or high in contrast, a coarse view is all that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

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

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

FIGURE 7.1

An image and its local histogram variations.



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

7.1.1 Image Pyramids

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

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

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

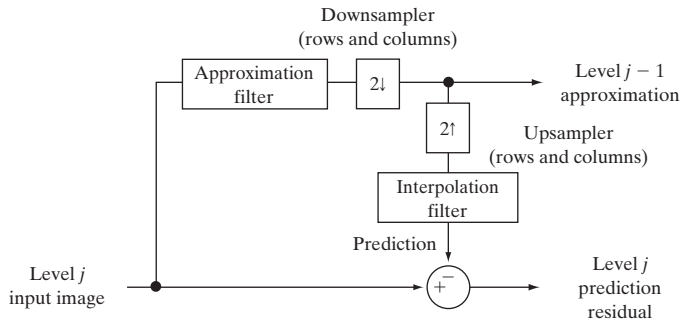
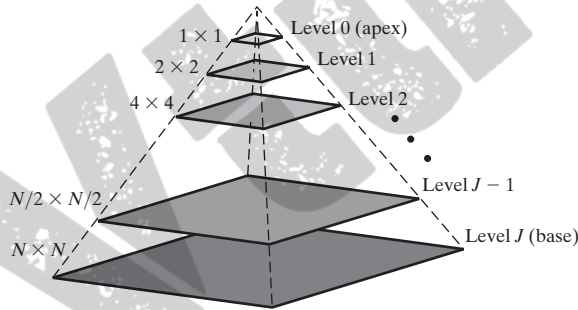


FIGURE 7.2
(a) An image pyramid. (b) A simple system for creating approximation and prediction residual pyramids.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

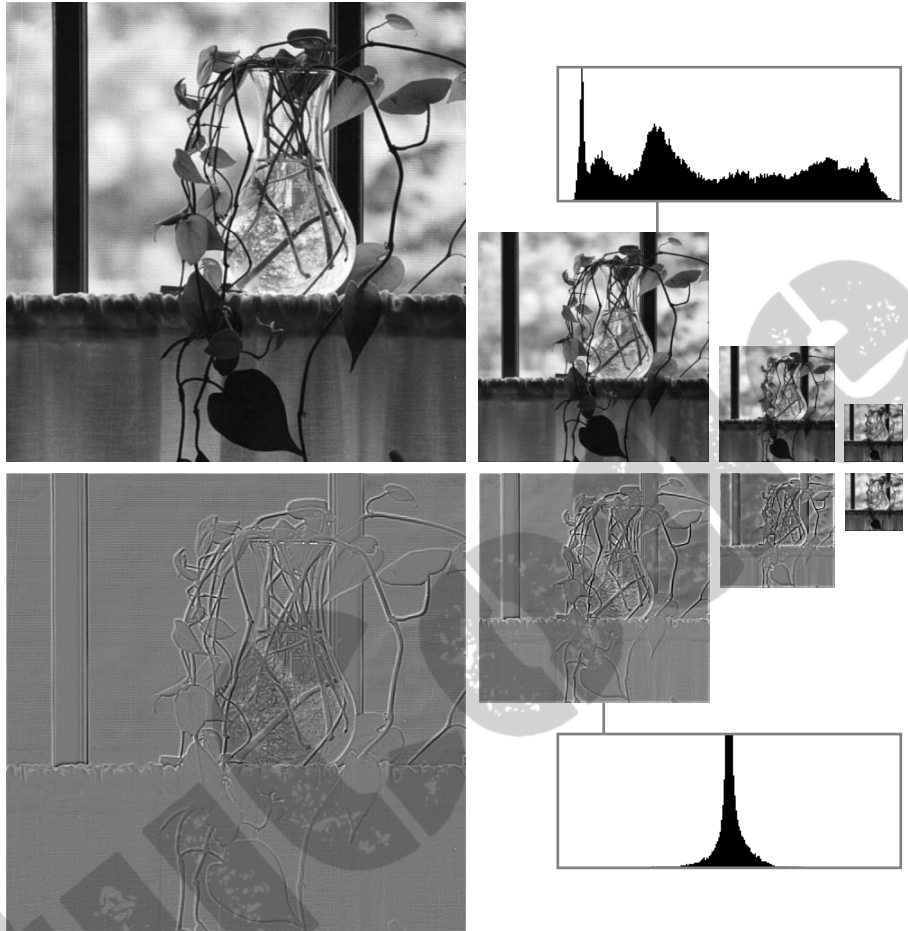
EXAMPLE 7.1:
Approximation
and prediction
residual pyramids.

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

a
b**FIGURE 7.3**

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

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



7.1.2 Subband Coding

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

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

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

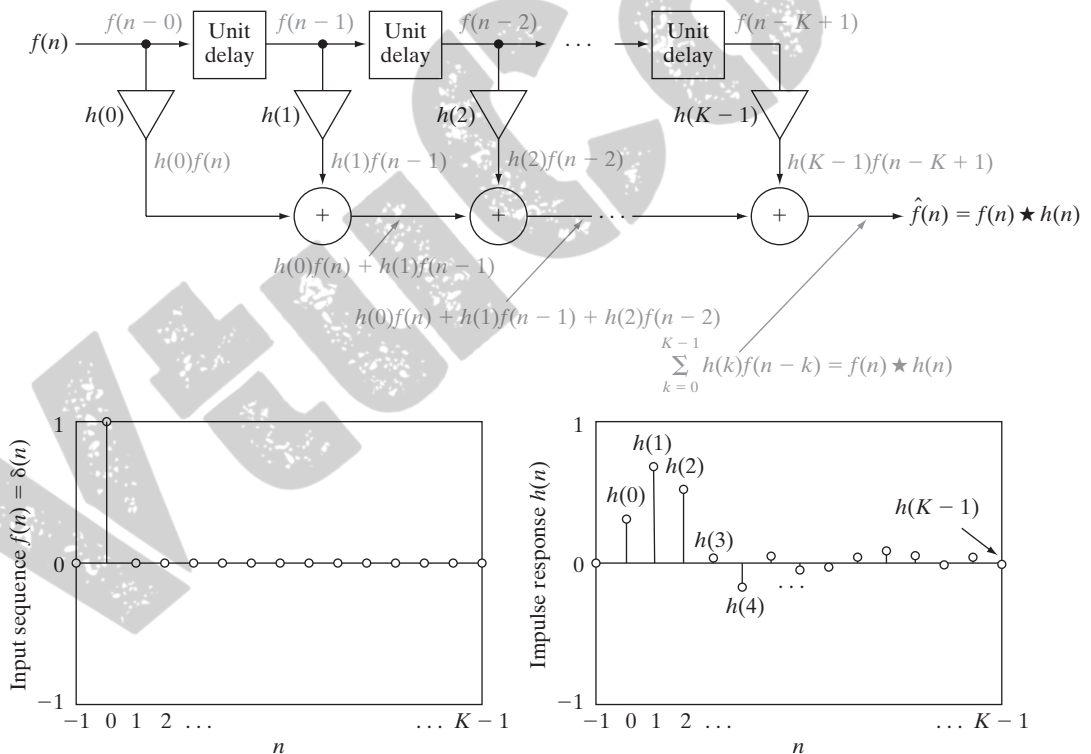
$$f(n-2) = \begin{cases} \vdots & \\ f(0) & \text{for } n = 2 \\ f(1) & \text{for } n = 2 + 1 = 3 \\ \vdots & \end{cases}$$

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

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

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

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



a
b c

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

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

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

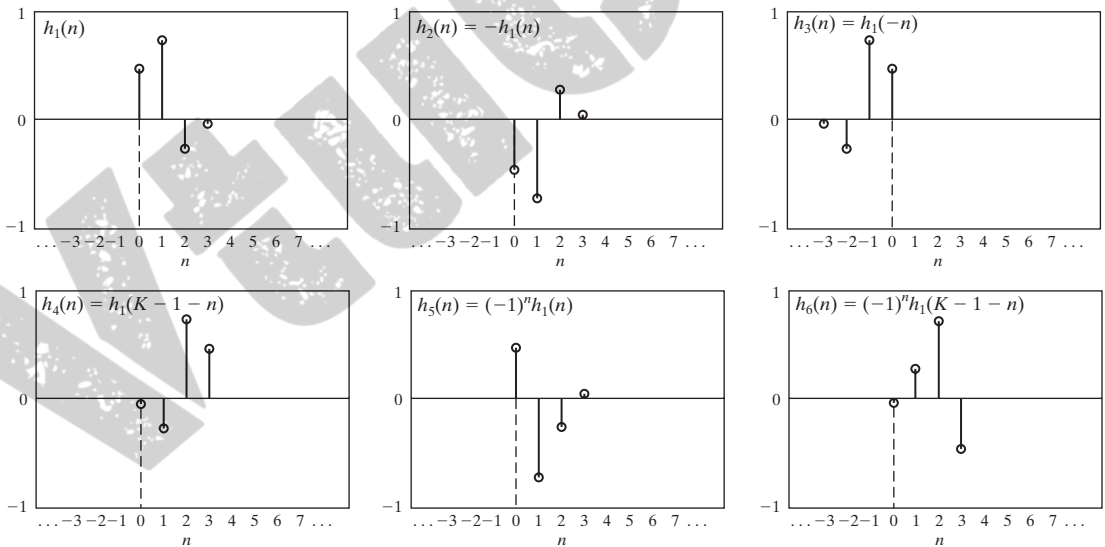
$$\begin{aligned}\hat{f}(n) &= \sum_{k=-\infty}^{\infty} h(k)\delta(n-k) \\ &= h(n)\end{aligned}\quad (7.1-4)$$

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

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

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

$$h_2(n) = -h_1(n) \quad (7.1-5)$$



a b c
d e f

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

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

$$h_3(n) = h_1(-n) \quad (7.1-6)$$

$$h_4(n) = h_1(K - 1 - n) \quad (7.1-7)$$

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

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

$$h_5(n) = (-1)^n h_1(n) \quad (7.1-8)$$

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

$$h_6(n) = (-1)^n h_1(K - 1 - n) \quad (7.1-9)$$

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

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

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

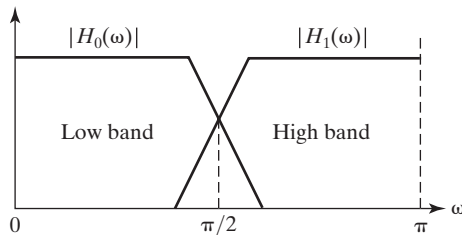
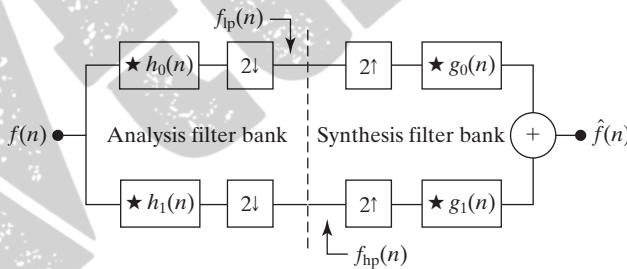


FIGURE 7.6
(a) A two-band subband coding and decoding system, and (b) its spectrum splitting properties.

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

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

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

OR

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

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

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

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

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

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

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

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

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

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

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

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

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

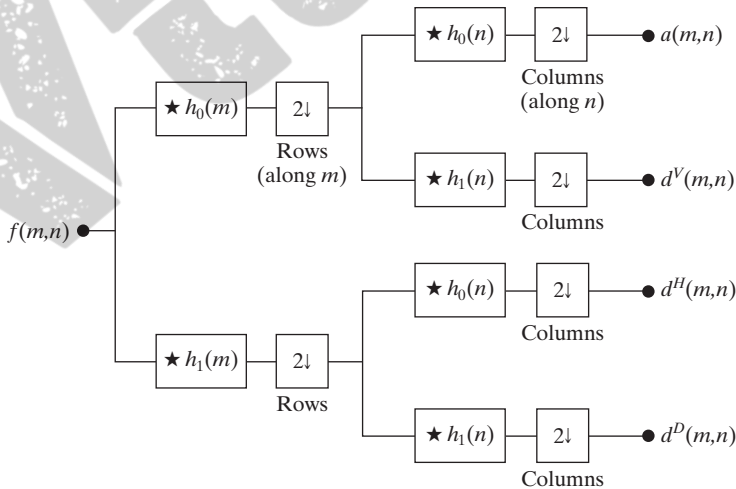


FIGURE 7.7

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

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

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

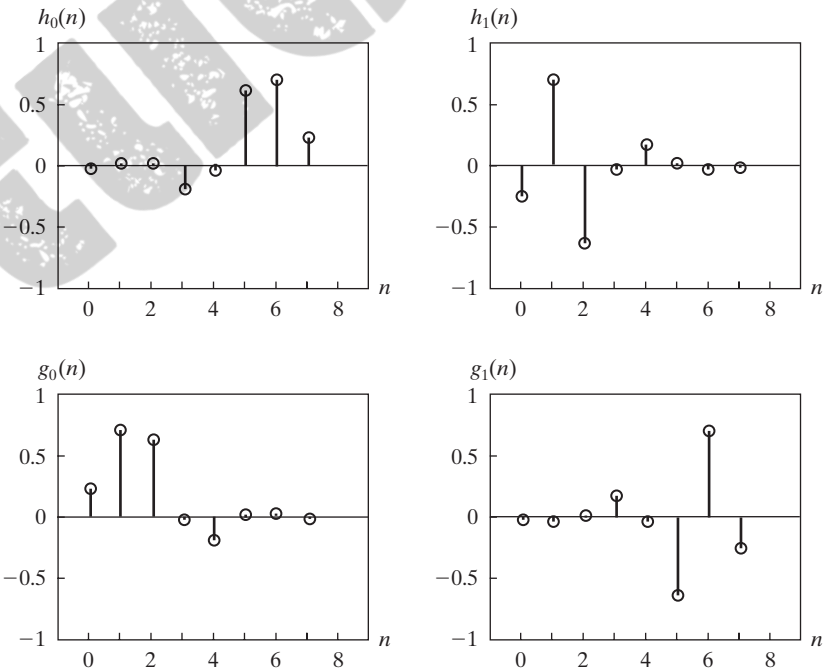
■ Figure 7.8 shows the impulse responses of four 8-tap orthonormal filters. The coefficients of prototype synthesis filter $g_0(n)$ for $0 \leq n \leq 7$ [in Fig. 7.8(c)] are defined in Table 7.1 (Daubechies [1992]). The coefficients of the remaining orthonormal filters can be computed using Eq. (7.1-14).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

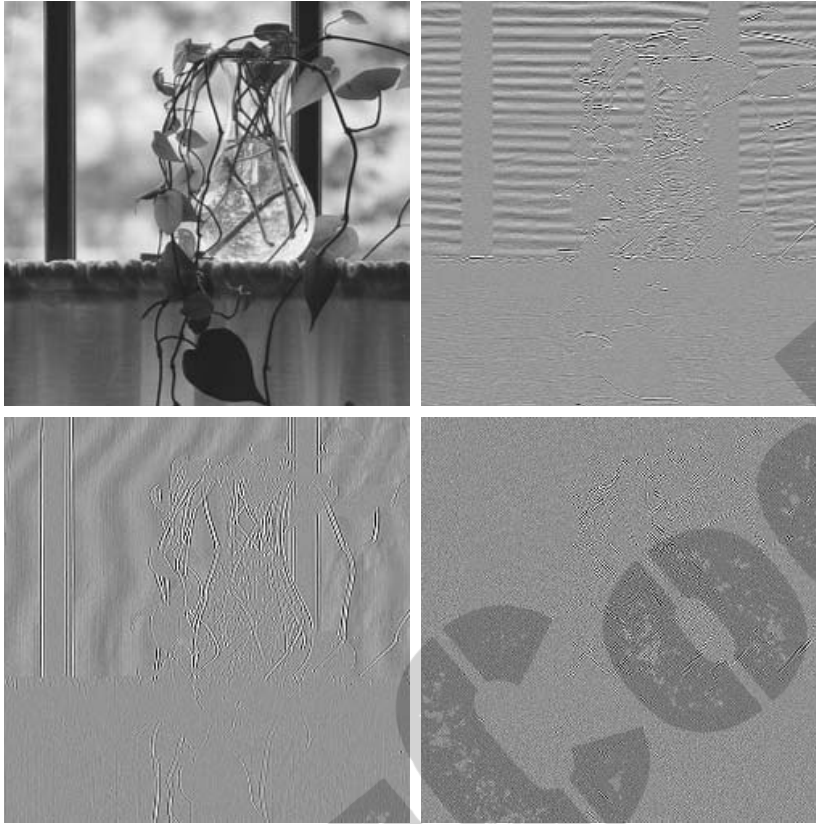
TABLE 7.1
Daubechies 8-tap
orthonormal filter
coefficients for
 $g_0(n)$ (Daubechies
[1992]).

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

a b
c d

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





a b
c d

FIGURE 7.9

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

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

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

See Section 4.5.4 for more on aliasing.

7.1.3 The Haar Transform

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

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

$$\mathbf{T} = \mathbf{H}\mathbf{F}\mathbf{H}^T \quad (7.1-15)$$

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

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

and

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

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

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

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

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

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

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

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

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

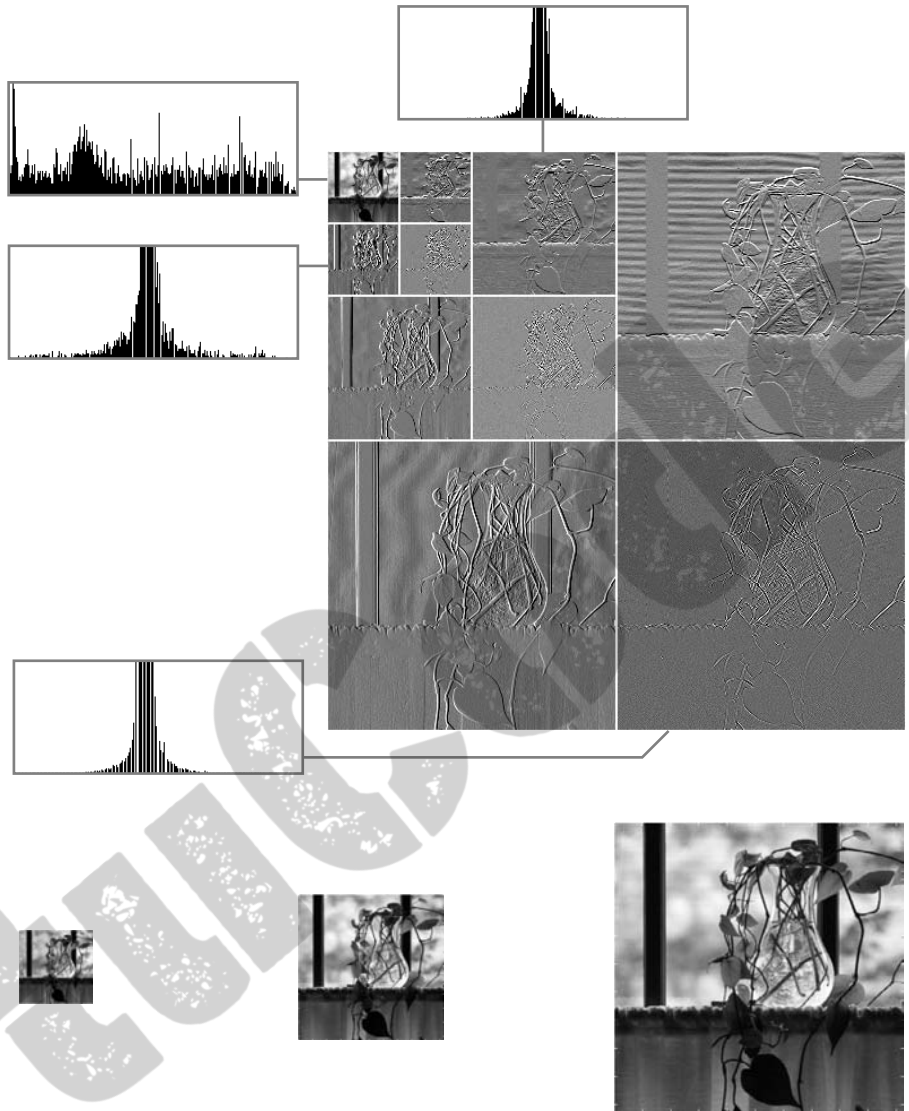
EXAMPLE 7.3: Haar functions in a discrete wavelet transform.

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

a
b c d

FIGURE 7.10

(a) A discrete wavelet transform using Haar H_2 basis functions. Its local histogram variations are also shown. (b)–(d) Several different approximations (64×64 , 128×128 , and 256×256) that can be obtained from (a).



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

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

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

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

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

7.2 Multiresolution Expansions

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

7.2.1 Series Expansions

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

$$f(x) = \sum_k \alpha_k \varphi_k(x) \quad (7.2-1)$$

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

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

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

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

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

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

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

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

$$\alpha_k = \langle \varphi_k(x), f(x) \rangle \quad (7.2-5)$$

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

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

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

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

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

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[‡]

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

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

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

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

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

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

7.2.2 Scaling Functions

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

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

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

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

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

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

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

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

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

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

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

EXAMPLE 7.4:
The Haar scaling
function.

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

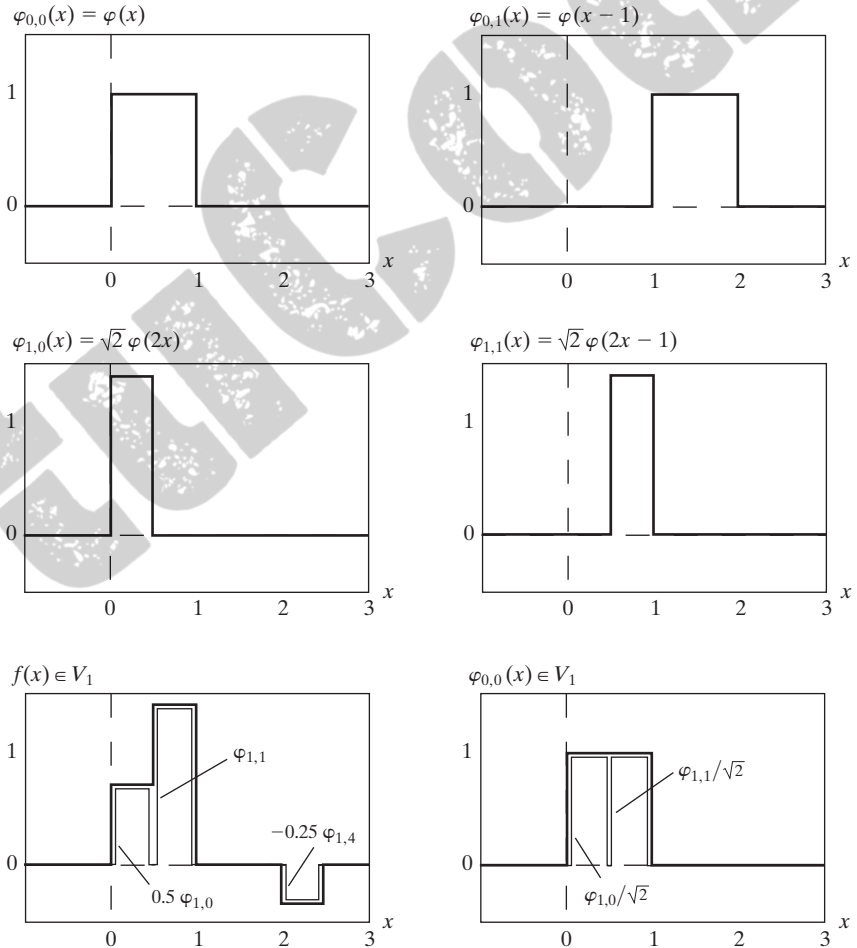
$$\varphi(x) = \begin{cases} 1 & 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases} \quad (7.2-14)$$

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

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

a b
c d
e f

FIGURE 7.11
Some Haar
scaling functions.



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

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

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

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

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

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

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

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

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

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

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

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

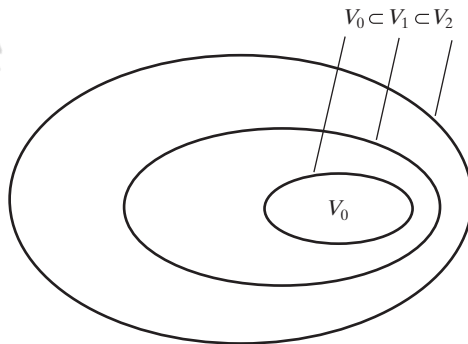


FIGURE 7.12
The nested function spaces spanned by a scaling function.

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

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

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,

$$V_{-\infty} = \{0\} \quad (7.2-16)$$

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

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

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

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

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

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

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

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

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

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

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

7.2.3 Wavelet Functions

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

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

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

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

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

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

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

$$V_{j+1} = V_j \oplus W_j \quad (7.2-22)$$

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,

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

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

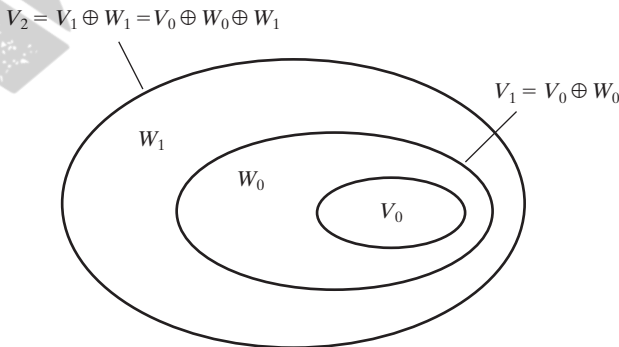


FIGURE 7.13
The relationship between scaling and wavelet function spaces.

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

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

or

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

or even

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

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

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

where j_0 is an arbitrary starting scale.

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

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

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

$$h_\psi(n) = (-1)^n h_\varphi(1 - n) \quad (7.2-29)$$

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

EXAMPLE 7.6:
The Haar wavelet function coefficients.

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

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

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

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

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

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

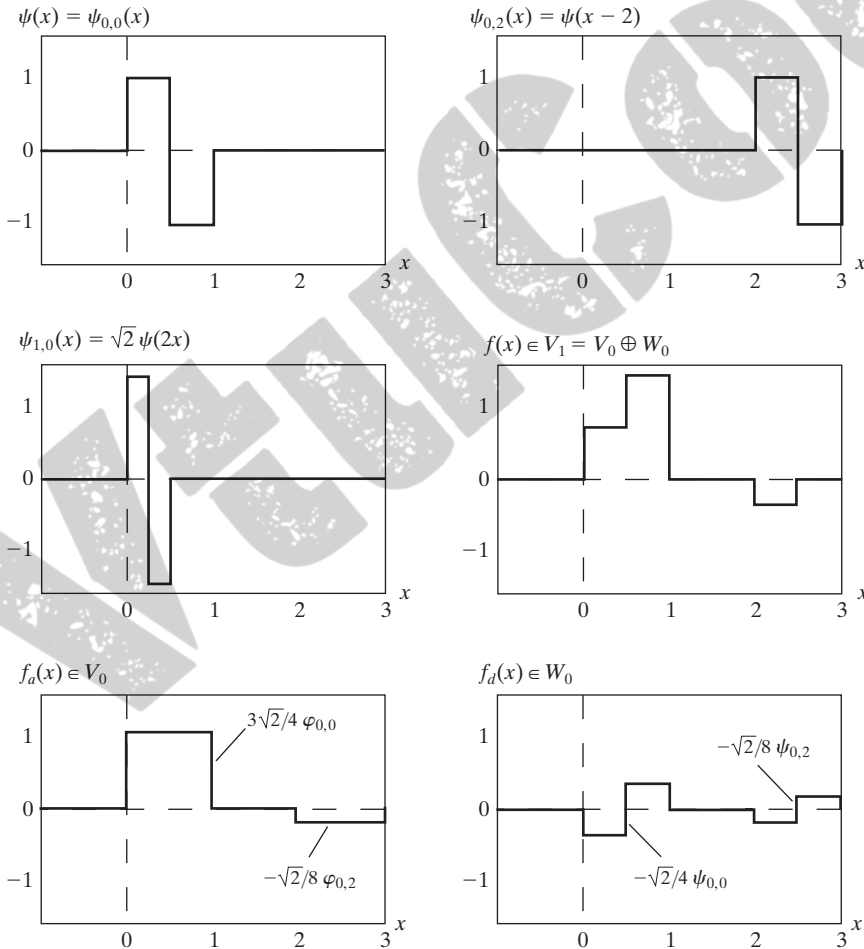


FIGURE 7.14
Haar wavelet
functions in W_0
and W_1 .

where

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

and

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

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

9.1 Preliminaries

You will find it helpful to review Sections 2.4.2 and 2.6.4 before proceeding.

The language of mathematical morphology is set theory. As such, morphology offers a unified and powerful approach to numerous image processing problems. Sets in mathematical morphology represent objects in an image. For example, the set of all white pixels in a binary image is a complete morphological description of the image. In binary images, the sets in question are members of the 2-D integer space Z^2 (see Section 2.4.2), where each element of a set is a tuple (2-D vector) whose coordinates are the (x, y) coordinates of a white (or black, depending on convention) pixel in the image. Gray-scale digital images of the form discussed in the previous chapters can be represented as sets whose components are in Z^3 . In this case, two components of each element of the set refer to the coordinates of a pixel, and the third corresponds to its discrete intensity value. Sets in higher dimensional spaces can contain other image attributes, such as color and time varying components.

The set reflection operation is analogous to the flipping (rotating) operation performed in spatial convolution (Section 3.4.2).

In addition to the basic set definitions in Section 2.6.4, the concepts of set reflection and translation are used extensively in morphology. The *reflection* of a set B , denoted \hat{B} , is defined as

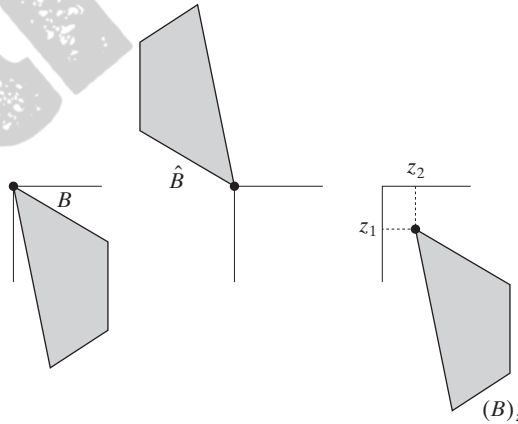
$$\hat{B} = \{w | w = -b, \text{ for } b \in B\} \quad (9.1-1)$$

If B is the set of pixels (2-D points) representing an object in an image, then \hat{B} is simply the set of points in B whose (x, y) coordinates have been replaced by $(-x, -y)$. Figures 9.1(a) and (b) show a simple set and its reflection.[†]

a b c

FIGURE 9.1

(a) A set, (b) its reflection, and (c) its translation by z .



[†]When working with graphics, such as the sets in Fig. 9.1, we use shading to indicate points (pixels) that are members of the set under consideration. When working with binary images, the sets of interest are pixels corresponding to objects. We show these in white, and all other pixels in black. The terms *foreground* and *background* are used often to denote the sets of pixels in an image defined to be objects and non-objects, respectively.

The *translation* of a set B by point $z = (z_1, z_2)$, denoted $(B)_z$, is defined as

$$(B)_z = \{c | c = b + z, \text{ for } b \in B\} \quad (9.1-2)$$

If B is the set of pixels representing an object in an image, then $(B)_z$ is the set of points in B whose (x, y) coordinates have been replaced by $(x + z_1, y + z_2)$. Figure 9.1(c) illustrates this concept using the set B from Fig. 9.1(a).

Set reflection and translation are employed extensively in morphology to formulate operations based on so-called *structuring elements* (SEs): small sets or subimages used to probe an image under study for properties of interest. The first row of Fig. 9.2 shows several examples of structuring elements where each shaded square denotes a member of the SE. When it does not matter whether a location in a given structuring element is or is not a member of the SE set, that location is marked with an “×” to denote a “don’t care” condition, as defined later in Section 9.5.4. In addition to a definition of which elements are members of the SE, the origin of a structuring element also must be specified. The origins of the various SEs in Fig. 9.2 are indicated by a black dot (although placing the center of an SE at its center of gravity is common, the choice of origin is problem dependent in general). When the SE is symmetric and no dot is shown, the assumption is that the origin is at the center of symmetry.

When working with images, we require that structuring elements be rectangular arrays. This is accomplished by appending the smallest possible number of background elements (shown nonshaded in Fig. 9.2) necessary to form a rectangular array. The first and last SEs in the second row of Fig. 9.2 illustrate the procedure. The other SEs in that row already are in rectangular form.

As an introduction to how structuring elements are used in morphology, consider Fig. 9.3. Figures 9.3(a) and (b) show a simple set and a structuring element. As mentioned in the previous paragraph, a computer implementation requires that set A be converted also to a rectangular array by adding background elements. The background border is made large enough to accommodate the entire structuring element when its origin is on the border of the

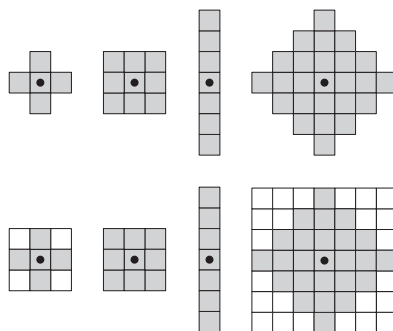


FIGURE 9.2 First row: Examples of structuring elements. Second row: Structuring elements converted to rectangular arrays. The dots denote the centers of the SEs.

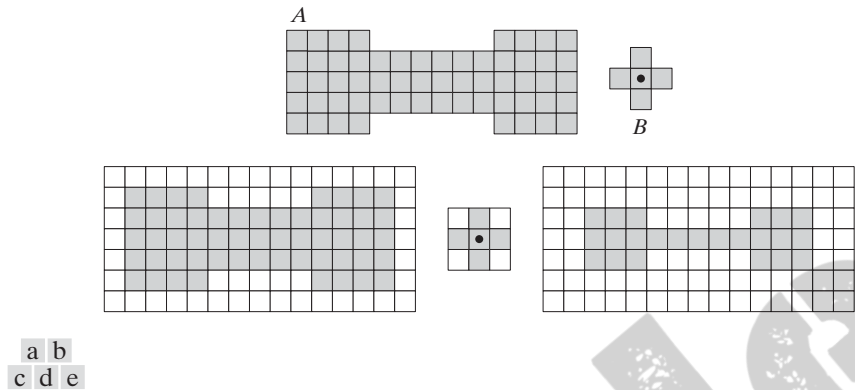


FIGURE 9.3 (a) A set (each shaded square is a member of the set). (b) A structuring element. (c) The set padded with background elements to form a rectangular array and provide a background border. (d) Structuring element as a rectangular array. (e) Set processed by the structuring element.

In future illustrations, we add enough background points to form rectangular arrays, but let the padding be implicit when the meaning is clear in order to simplify the figures.

original set (this is analogous to padding for spatial correlation and convolution, as discussed in Section 3.4.2). In this case, the structuring element is of size 3×3 with the origin in the center, so a one-element border that encompasses the entire set is sufficient, as Fig. 9.3(c) shows. As in Fig. 9.2, the structuring element is filled with the smallest possible number of background elements necessary to make it into a rectangular array [Fig. 9.3(d)].

Suppose that we define an operation on set A using structuring element B , as follows: Create a new set by running B over A so that the origin of B visits every element of A . At each location of the origin of B , if B is completely contained in A , mark that location as a member of the new set (shown shaded); else mark it as not being a member of the new set (shown not shaded). Figure 9.3(e) shows the result of this operation. We see that, when the origin of B is on a border element of A , part of B ceases to be contained in A , thus eliminating the location on which B is centered as a possible member for the new set. The net result is that the boundary of the set is *eroded*, as Fig. 9.3(e) shows. When we use terminology such as “the structuring element is contained in the set,” we mean specifically that the elements of A and B fully overlap. In other words, although we showed A and B as arrays containing both shaded and nonshaded elements, only the shaded elements of both sets are considered in determining whether or not B is contained in A . These concepts form the basis of the material in the next section, so it is important that you understand the ideas in Fig. 9.3 fully before proceeding.

9.2 Erosion and Dilation

We begin the discussion of morphology by studying two operations: *erosion* and *dilation*. These operations are fundamental to morphological processing. In fact, many of the morphological algorithms discussed in this chapter are based on these two primitive operations.

9.2.1 Erosion

With A and B as sets in Z^2 , the erosion of A by B , denoted $A \ominus B$, is defined as

$$A \ominus B = \{z | (B)_z \subseteq A\} \quad (9.2-1)$$

In words, this equation indicates that the erosion of A by B is the set of all points z such that B , translated by z , is contained in A . In the following discussion, set B is assumed to be a structuring element. Equation (9.2-1) is the mathematical formulation of the example in Fig. 9.3(e), discussed at the end of the last section. Because the statement that B has to be contained in A is equivalent to B not sharing any common elements with the background, we can express erosion in the following equivalent form:

$$A \ominus B = \{z | (B)_z \cap A^c = \emptyset\} \quad (9.2-2)$$

where, as defined in Section 2.6.4, A^c is the complement of A and \emptyset is the empty set.

Figure 9.4 shows an example of erosion. The elements of A and B are shown shaded and the background is white. The solid boundary in Fig. 9.4(c) is the limit beyond which further displacements of the origin of B would cause the structuring element to cease being completely contained in A . Thus, the locus of points (locations of the origin of B) within (and including) this boundary, constitutes the erosion of A by B . We show the erosion shaded in Fig. 9.4(c). Keep in mind that that erosion is simply the *set* of

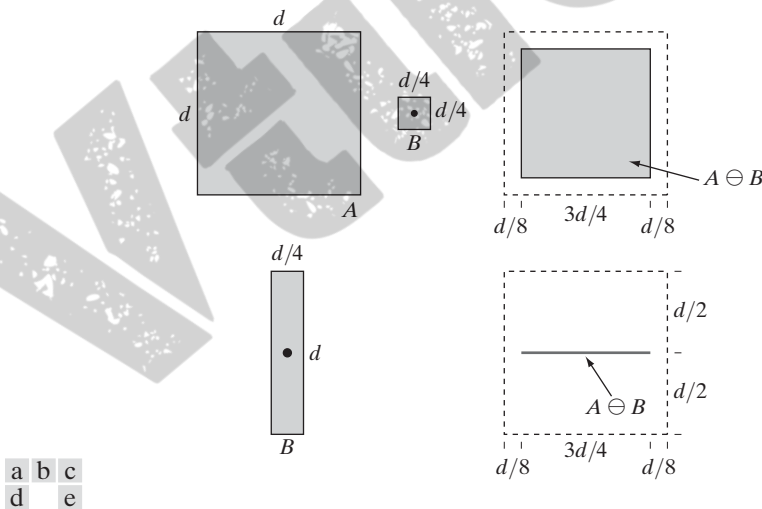


FIGURE 9.4 (a) Set A . (b) Square structuring element, B . (c) Erosion of A by B , shown shaded. (d) Elongated structuring element. (e) Erosion of A by B using this element. The dotted border in (c) and (e) is the boundary of set A , shown only for reference.

values of z that satisfy Eq. (9.2-1) or (9.2-2). The boundary of set A is shown dashed in Figs. 9.4(c) and (e) only as a reference; it is not part of the erosion operation. Figure 9.4(d) shows an elongated structuring element, and Fig. 9.4(e) shows the erosion of A by this element. Note that the original set was eroded to a line.

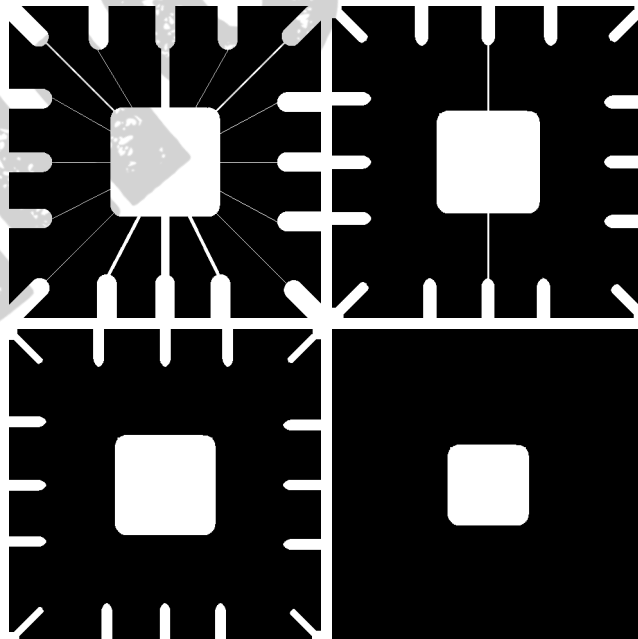
Equations (9.2-1) and (9.2-2) are not the only definitions of erosion (see Problems 9.9 and 9.10 for two additional, equivalent definitions.) However, these equations have the distinct advantage over other formulations in that they are more intuitive when the structuring element B is viewed as a spatial mask (see Section 3.4.1).

EXAMPLE 9.1:
Using erosion to
remove image
components.

■ Suppose that we wish to remove the lines connecting the center region to the border pads in Fig. 9.5(a). Eroding the image with a square structuring element of size 11×11 whose components are all 1s removed most of the lines, as Fig. 9.5(b) shows. The reason the two vertical lines in the center were thinned but not removed completely is that their width is greater than 11 pixels. Changing the SE size to 15×15 and eroding the original image again did remove all the connecting lines, as Fig. 9.5(c) shows (an alternate approach would have been to erode the image in Fig. 9.5(b) again using the same 11×11 SE). Increasing the size of the structuring element even more would eliminate larger components. For example, the border pads can be removed with a structuring element of size 45×45 , as Fig. 9.5(d) shows.

a b
c d

FIGURE 9.5 Using erosion to remove image components. (a) A 486×486 binary image of a wire-bond mask. (b)–(d) Image eroded using square structuring elements of sizes 11×11 , 15×15 , and 45×45 , respectively. The elements of the SEs were all 1s.



We see from this example that erosion shrinks or thins objects in a binary image. In fact, we can view erosion as a *morphological filtering* operation in which image details smaller than the structuring element are filtered (removed) from the image. In Fig. 9.5, erosion performed the function of a “line filter.” We return to the concept of a morphological filter in Sections 9.3 and 9.6.3. ■

9.2.2 Dilation

With A and B as sets in Z^2 , the *dilation* of A by B , denoted $A \oplus B$, is defined as

$$A \oplus B = \{z | (\hat{B})_z \cap A \neq \emptyset\} \quad (9.2-3)$$

This equation is based on reflecting B about its origin, and shifting this reflection by z (see Fig. 9.1). The dilation of A by B then is the set of all displacements, z , such that \hat{B} and A overlap by at least one element. Based on this interpretation, Eq. (9.2-3) can be written equivalently as

$$A \oplus B = \{z | [(\hat{B})_z \cap A] \subseteq A\} \quad (9.2-4)$$

As before, we assume that B is a structuring element and A is the set (image objects) to be dilated.

Equations (9.2-3) and (9.2-4) are not the only definitions of dilation currently in use (see Problems 9.11 and 9.12 for two different, yet equivalent, definitions). However, the preceding definitions have a distinct advantage over other formulations in that they are more intuitive when the structuring element B is viewed as a convolution mask. The basic process of flipping (rotating) B about its origin and then successively displacing it so that it slides over set (image) A is analogous to spatial convolution, as introduced in Section 3.4.2. Keep in mind, however, that dilation is based on set operations and therefore is a nonlinear operation, whereas convolution is a linear operation.

Unlike erosion, which is a shrinking or thinning operation, dilation “grows” or “thickens” objects in a binary image. The specific manner and extent of this thickening is controlled by the shape of the structuring element used. Figure 9.6(a) shows the same set used in Fig. 9.4, and Fig. 9.6(b) shows a structuring element (in this case $\hat{B} = B$ because the SE is symmetric about its origin). The dashed line in Fig. 9.6(c) shows the original set for reference, and the solid line shows the limit beyond which any further displacements of the origin of \hat{B} by z would cause the intersection of \hat{B} and A to be empty. Therefore, all points on and inside this boundary constitute the dilation of A by B . Figure 9.6(d) shows a structuring element designed to achieve more dilation vertically than horizontally, and Fig. 9.6(e) shows the dilation achieved with this element.



FIGURE 9.6
(a) Set A .
(b) Square structuring element (the dot denotes the origin).
(c) Dilation of A by B , shown shaded.
(d) Elongated structuring element. (e) Dilation of A using this element. The dotted border in (c) and (e) is the boundary of set A , shown only for reference


EXAMPLE 9.2:
An illustration of dilation.

■ One of the simplest applications of dilation is for bridging gaps. Figure 9.7(a) shows the same image with broken characters that we studied in Fig. 4.49 in connection with lowpass filtering. The maximum length of the breaks is known to be two pixels. Figure 9.7(b) shows a structuring element that can be used for repairing the gaps (note that instead of shading, we used 1s to denote the elements of the SE and 0s for the background; this is because the SE is now being treated as a subimage and not as a graphic). Figure 9.7(c) shows the result of dilating the original image with this structuring element. The gaps were bridged. One immediate advantage of the morphological approach over the lowpass filtering method we used to bridge the gaps in Fig. 4.49 is




FIGURE 9.7
(a) Sample text of poor resolution with broken characters (see magnified view).
(b) Structuring element.
(c) Dilation of (a) by (b). Broken segments were joined.

Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.



Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.



0	1	0
1	1	1
0	1	0

that the morphological method resulted directly in a binary image. Lowpass filtering, on the other hand, started with a binary image and produced a gray-scale image, which would require a pass with a thresholding function to convert it back to binary form. ■

9.2.3 Duality

Erosion and dilation are duals of each other with respect to set complementation and reflection. That is,

$$(A \ominus B)^c = A^c \oplus \hat{B} \quad (9.2-5)$$

and

$$(A \oplus B)^c = A^c \ominus \hat{B} \quad (9.2-6)$$

Equation (9.2-5) indicates that erosion of A by B is the complement of the dilation of A^c by \hat{B} , and vice versa. The duality property is useful particularly when the structuring element is symmetric with respect to its origin (as often is the case), so that $\hat{B} = B$. Then, we can obtain the erosion of an image by B simply by dilating its background (i.e., dilating A^c) with the same structuring element and complementing the result. Similar comments apply to Eq. (9.2-6).

We proceed to prove formally the validity of Eq. (9.2-5) in order to illustrate a typical approach for establishing the validity of morphological expressions. Starting with the definition of erosion, it follows that

$$(A \ominus B)^c = \{z | (B)_z \subseteq A\}^c$$

If set $(B)_z$ is contained in A , then $(B)_z \cap A^c = \emptyset$, in which case the preceding expression becomes

$$(A \ominus B)^c = \{z | (B)_z \cap A^c = \emptyset\}^c$$

But the *complement* of the set of z 's that satisfy $(B)_z \cap A^c = \emptyset$ is the set of z 's such that $(B)_z \cap A^c \neq \emptyset$. Therefore,

$$\begin{aligned} (A \ominus B)^c &= \{z | (B)_z \cap A^c \neq \emptyset\} \\ &= A^c \oplus \hat{B} \end{aligned}$$

where the last step follows from Eq. (9.2-3). This concludes the proof. A similar line of reasoning can be used to prove Eq. (9.2-6) (see Problem 9.13).

9.3 Opening and Closing

As you have seen, dilation expands the components of an image and erosion shrinks them. In this section we discuss two other important morphological operations: opening and closing. *Opening* generally smoothes the contour of an object, breaks narrow isthmuses, and eliminates thin protrusions. *Closing* also tends to smooth sections of contours but, as opposed to opening, it generally fuses narrow breaks and long thin gulfs, eliminates small holes, and fills gaps in the contour.

The *opening* of set A by structuring element B , denoted $A \circ B$, is defined as

$$A \circ B = (A \ominus B) \oplus B \quad (9.3-1)$$

Thus, the opening A by B is the erosion of A by B , followed by a dilation of the result by B .

Similarly, the *closing* of set A by structuring element B , denoted $A \bullet B$, is defined as

$$A \bullet B = (A \oplus B) \ominus B \quad (9.3-2)$$

which says that the closing of A by B is simply the dilation of A by B , followed by the erosion of the result by B .

The opening operation has a simple geometric interpretation (Fig. 9.8). Suppose that we view the structuring element B as a (flat) “rolling ball.” The *boundary* of $A \circ B$ is then established by the points in B that reach the *farthest* into the boundary of A as B is rolled around the *inside* of this boundary. This geometric *fitting* property of the opening operation leads to a set-theoretic formulation, which states that the opening of A by B is obtained by taking the union of all translates of B that fit into A . That is, opening can be expressed as a fitting process such that

$$A \circ B = \bigcup \{ (B)_z \mid (B)_z \subseteq A \} \quad (9.3-3)$$

where $\bigcup \{ \cdot \}$ denotes the union of all the sets inside the braces.

Closing has a similar geometric interpretation, except that now we roll B on the outside of the boundary (Fig. 9.9). As discussed below, opening and closing are duals of each other, so having to roll the ball on the outside is not unexpected. Geometrically, a point w is an element of $A \bullet B$ if and only if $(B)_z \cap A \neq \emptyset$ for any translate of $(B)_z$ that contains w . Figure 9.9 illustrates the basic geometrical properties of closing.

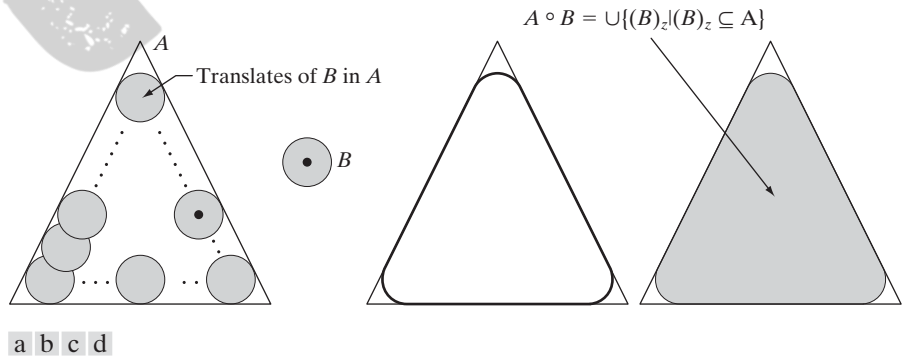
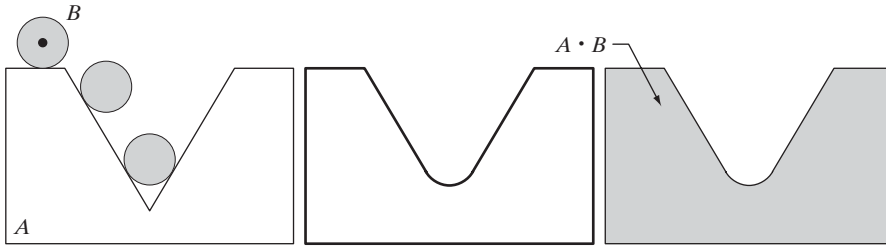


FIGURE 9.8 (a) Structuring element B “rolling” along the inner boundary of A (the dot indicates the origin of B). (b) Structuring element. (c) The heavy line is the outer boundary of the opening. (d) Complete opening (shaded). We did not shade A in (a) for clarity.



a b c

FIGURE 9.9 (a) Structuring element B “rolling” on the outer boundary of set A . (b) The heavy line is the outer boundary of the closing. (c) Complete closing (shaded). We did not shade A in (a) for clarity.

Figure 9.10 further illustrates the opening and closing operations. Figure 9.10(a) shows a set A , and Fig. 9.10(b) shows various positions of a disk structuring element during the erosion process. When completed, this process resulted in the disjoint figure in Fig. 9.10(c). Note the elimination of the bridge between the two main sections. Its width was thin in relation to the diameter of

EXAMPLE 9.3:

A simple illustration of morphological opening and closing.

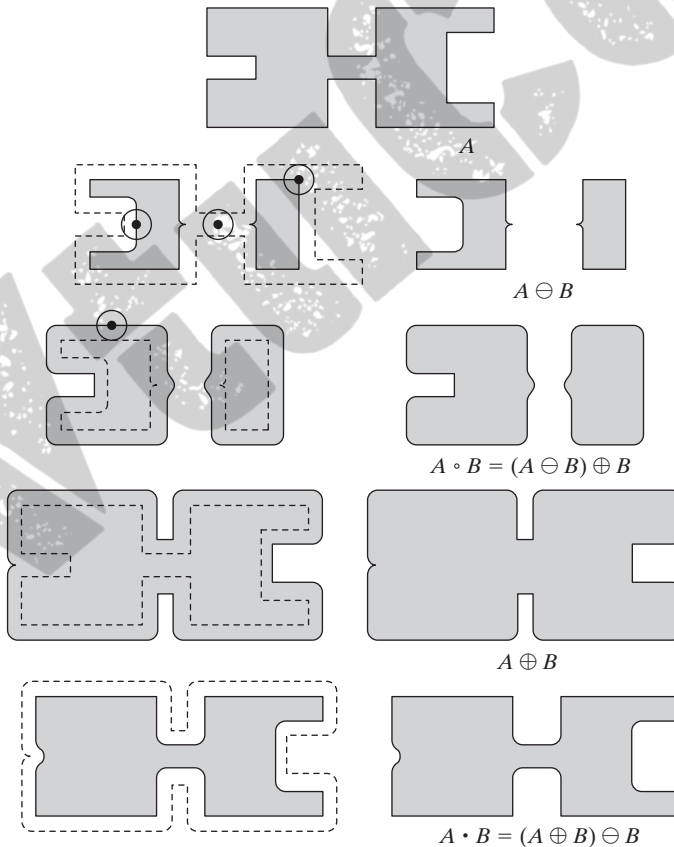


FIGURE 9.10

Morphological opening and closing. The structuring element is the small circle shown in various positions in (b). The SE was not shaded here for clarity. The dark dot is the center of the structuring element.

the structuring element; that is, the structuring element could not be completely contained in this part of the set, thus violating the conditions of Eq. (9.2-1). The same was true of the two rightmost members of the object. Protruding elements where the disk did not fit were eliminated. Figure 9.10(d) shows the process of dilating the eroded set, and Fig. 9.10(e) shows the final result of opening. Note that outward pointing corners were rounded, whereas inward pointing corners were not affected.

Similarly, Figs. 9.10(f) through (i) show the results of closing A with the same structuring element. We note that the inward pointing corners were rounded, whereas the outward pointing corners remained unchanged. The leftmost intrusion on the boundary of A was reduced in size significantly, because the disk did not fit there. Note also the smoothing that resulted in parts of the object from both opening and closing the set A with a circular structuring element. ■

As in the case with dilation and erosion, opening and closing are duals of each other with respect to set complementation and reflection. That is,

$$(A \bullet B)^c = (A^c \circ \hat{B}) \quad (9.3-4)$$

and

$$(A \circ B)^c = (A^c \bullet \hat{B}) \quad (9.3-5)$$

We leave the proof of this result as an exercise (Problem 9.14).

The opening operation satisfies the following properties:

- (a) $A \circ B$ is a subset (subimage) of A .
- (b) If C is a subset of D , then $C \circ B$ is a subset of $D \circ B$.
- (c) $(A \circ B) \circ B = A \circ B$.

Similarly, the closing operation satisfies the following properties:

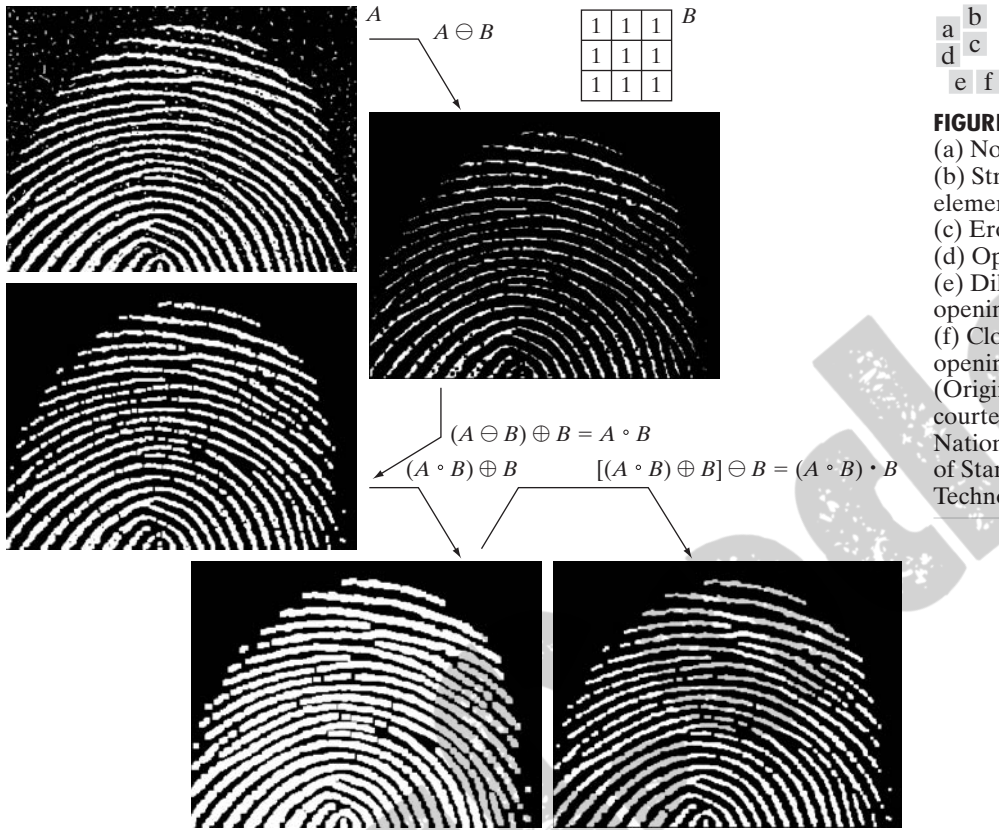
- (a) A is a subset (subimage) of $A \bullet B$.
- (b) If C is a subset of D , then $C \bullet B$ is a subset of $D \bullet B$.
- (c) $(A \bullet B) \bullet B = A \bullet B$.

Note from condition (c) in both cases that multiple openings or closings of a set have no effect after the operator has been applied once.

EXAMPLE 9.4:
Use of opening
and closing for
morphological
filtering.

■ Morphological operations can be used to construct filters similar in concept to the spatial filters discussed in Chapter 3. The binary image in Fig. 9.11(a) shows a section of a fingerprint corrupted by noise. Here the noise manifests itself as random light elements on a dark background and as dark elements on the light components of the fingerprint. The objective is to eliminate the noise and its effects on the print while distorting it as little as possible. A morphological filter consisting of opening followed by closing can be used to accomplish this objective.

Figure 9.11(b) shows the structuring element used. The rest of Fig. 9.11 shows a step-by-step sequence of the filtering operation. Figure 9.11(c) is the

**FIGURE 9.11**

(a) Noisy image.
 (b) Structuring element.
 (c) Eroded image.
 (d) Opening of A .
 (e) Dilation of the opening.
 (f) Closing of the opening.
 (Original image courtesy of the National Institute of Standards and Technology.)

result of eroding A with the structuring element. The background noise was completely eliminated in the erosion stage of opening because in this case all noise components are smaller than the structuring element. The size of the noise elements (dark spots) contained within the fingerprint actually increased in size. The reason is that these elements are inner boundaries that increase in size as the object is eroded. This enlargement is countered by performing dilation on Fig. 9.11(c). Figure 9.11(d) shows the result. The noise components contained in the fingerprint were reduced in size or deleted completely.

The two operations just described constitute the opening of A by B . We note in Fig. 9.11(d) that the net effect of opening was to eliminate virtually all noise components in both the background and the fingerprint itself. However, new gaps between the fingerprint ridges were created. To counter this undesirable effect, we perform a dilation on the opening, as shown in Fig. 9.11(e). Most of the breaks were restored, but the ridges were thickened, a condition that can be remedied by erosion. The result, shown in Fig. 9.11(f), constitutes the closing of the opening of Fig. 9.11(d). This final result is remarkably clean of noise specks, but it has the disadvantage that some of the print ridges were not fully repaired, and thus contain breaks. This is not totally unexpected, because no conditions were built into the procedure for maintaining connectivity (we discuss this issue again in Example 9.8 and demonstrate ways to address it in Section 11.1.7). ■

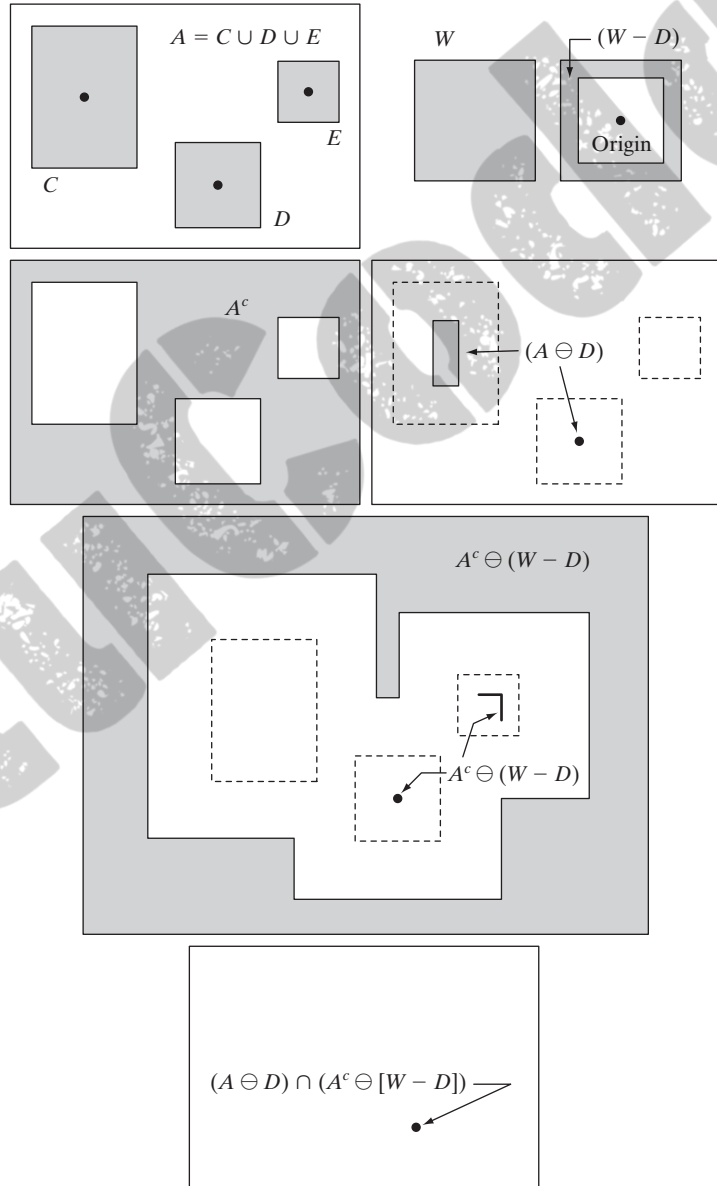
9.4 The Hit-or-Miss Transformation

The morphological hit-or-miss transform is a basic tool for shape detection. We introduce this concept with the aid of Fig. 9.12, which shows a set A consisting of three shapes (subsets), denoted C , D , and E . The shading in Figs. 9.12(a) through (c) indicates the original sets, whereas the shading in Figs. 9.12(d) and (e) indicates the result of morphological operations. The objective is to find the location of one of the shapes, say, D .

a b
c d
e
f

FIGURE 9.12

(a) Set A . (b) A window, W , and the local background of D with respect to W , $(W - D)$. (c) Complement of A . (d) Erosion of A by D . (e) Erosion of A^c by $(W - D)$. (f) Intersection of (d) and (e), showing the location of the origin of D , as desired. The dots indicate the origins of C , D , and E .



Let the origin of each shape be located at its center of gravity. Let D be enclosed by a small window, W . The *local background* of D with respect to W is defined as the set difference $(W - D)$, as shown in Fig. 9.12(b). Figure 9.12(c) shows the complement of A , which is needed later. Figure 9.12(d) shows the erosion of A by D (the dashed lines are included for reference). Recall that the erosion of A by D is the set of locations of the *origin* of D , such that D is completely contained in A . Interpreted another way, $A \ominus D$ may be viewed geometrically as the set of all locations of the origin of D at which D found a match (hit) in A . Keep in mind that in Fig. 9.12 A consists only of the three *disjoint* sets C , D , and E .

Figure 9.12(e) shows the erosion of the complement of A by the local background set $(W - D)$. The outer shaded region in Fig. 9.12(e) is part of the erosion. We note from Figs. 9.12(d) and (e) that the set of locations for which D *exactly* fits inside A is the *intersection* of the erosion of A by D and the erosion of A^c by $(W - D)$ as shown in Fig. 9.12(f). This intersection is precisely the location sought. In other words, if B denotes the set composed of D and its background, the match (or set of matches) of B in A , denoted $A \otimes B$, is

$$A \otimes B = (A \ominus D) \cap [A^c \ominus (W - D)] \quad (9.4-1)$$

We can generalize the notation somewhat by letting $B = (B_1, B_2)$, where B_1 is the set formed from elements of B associated with an object and B_2 is the set of elements of B associated with the corresponding background. From the preceding discussion, $B_1 = D$ and $B_2 = (W - D)$. With this notation, Eq. (9.4-1) becomes

$$A \otimes B = (A \ominus B_1) \cap (A^c \ominus B_2) \quad (9.4-2)$$

Thus, set $A \otimes B$ contains all the (origin) points at which, simultaneously, B_1 found a match (“hit”) in A and B_2 found a match in A^c . By using the definition of set differences given in Eq. (2.6-19) and the dual relationship between erosion and dilation given in Eq. (9.2-5), we can write Eq. (9.4-2) as

$$A \otimes B = (A \ominus B_1) - (A \oplus \hat{B}_2) \quad (9.4-3)$$

However, Eq. (9.4-2) is considerably more intuitive. We refer to any of the preceding three equations as the *morphological hit-or-miss transform*.

The reason for using a structuring element B_1 associated with objects and an element B_2 associated with the background is based on an assumed definition that two or more objects are distinct only if they form disjoint (disconnected) sets. This is guaranteed by requiring that each object have at least a one-pixel-thick background around it. In some applications, we may be interested in detecting certain patterns (combinations) of 1s and 0s within a set, in which case a background is not required. In such instances, the hit-or-miss transform reduces to simple erosion. As indicated previously, erosion is still a set of matches, but without the additional requirement of a background match for detecting individual objects. This simplified pattern detection scheme is used in some of the algorithms developed in the following section.

9.5 Some Basic Morphological Algorithms

With the preceding discussion as foundation, we are now ready to consider some practical uses of morphology. When dealing with binary images, one of the principal applications of morphology is in extracting image components that are useful in the representation and description of shape. In particular, we consider morphological algorithms for extracting boundaries, connected components, the convex hull, and the skeleton of a region. We also develop several methods (for region filling, thinning, thickening, and pruning) that are used frequently in conjunction with these algorithms as pre- or post-processing steps. We make extensive use in this section of “mini-images,” designed to clarify the mechanics of each morphological process as we introduce it. These images are shown graphically with 1s shaded and 0s in white.

9.5.1 Boundary Extraction

The boundary of a set A , denoted by $\beta(A)$, can be obtained by first eroding A by B and then performing the set difference between A and its erosion. That is,

$$\beta(A) = A - (A \ominus B)$$
 (9.5-1)

where B is a suitable structuring element.

Figure 9.13 illustrates the mechanics of boundary extraction. It shows a simple binary object, a structuring element B , and the result of using Eq. (9.5-1). Although the structuring element in Fig. 9.13(b) is among the most frequently used, it is by no means unique. For example, using a 5×5 structuring element of 1s would result in a boundary between 2 and 3 pixels thick.

From this point on, we do not show border padding explicitly.

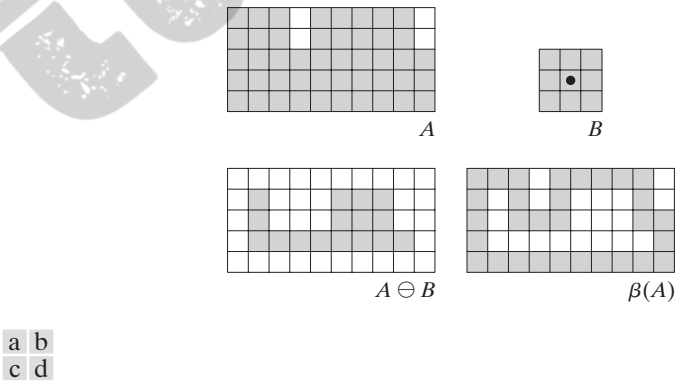
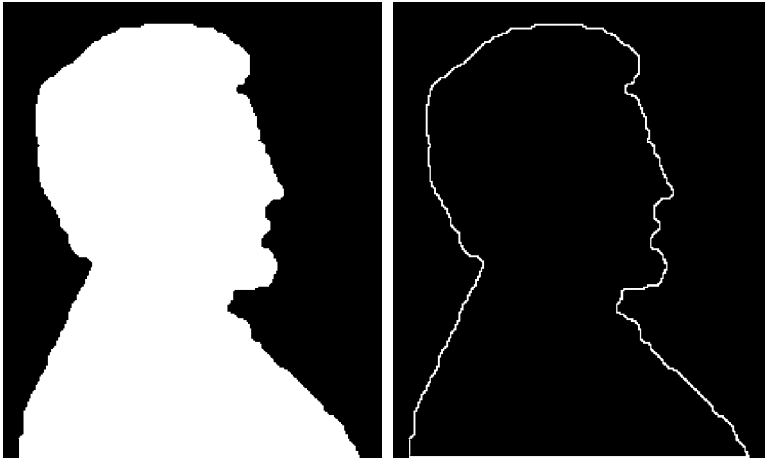


FIGURE 9.13 (a) Set A . (b) Structuring element B . (c) A eroded by B . (d) Boundary, given by the set difference between A and its erosion.



a b

FIGURE 9.14

(a) A simple binary image, with 1s represented in white. (b) Result of using Eq. (9.5-1) with the structuring element in Fig. 9.13(b).

■ Figure 9.14 further illustrates the use of Eq. (9.5-1) with a 3×3 structuring element of 1s. As for all binary images in this chapter, binary 1s are shown in white and 0s in black, so the elements of the structuring element, which are 1s, also are treated as white. Because of the size of the structuring element used, the boundary in Fig. 9.14(b) is one pixel thick.

EXAMPLE 9.5:

Boundary extraction by morphological processing.

9.5.2 Hole Filling

A *hole* may be defined as a background region surrounded by a connected border of foreground pixels. In this section, we develop an algorithm based on set dilation, complementation, and intersection for filling holes in an image. Let A denote a set whose elements are 8-connected boundaries, each boundary enclosing a background region (i.e., a hole). Given a point in each hole, the objective is to fill all the holes with 1s.

We begin by forming an array, X_0 , of 0s (the same size as the array containing A), except at the locations in X_0 corresponding to the given point in each hole, which we set to 1. Then, the following procedure fills all the holes with 1s:

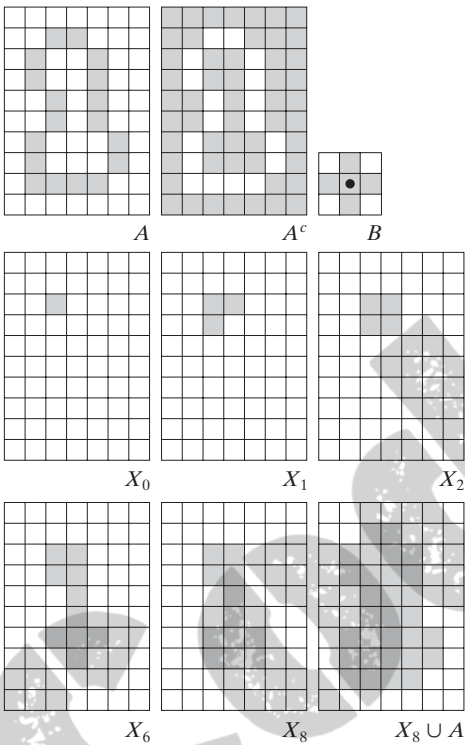
$$X_k = (X_{k-1} \oplus B) \cap A^c \quad k = 1, 2, 3, \dots \quad (9.5-2)$$

where B is the symmetric structuring element in Fig. 9.15(c). The algorithm terminates at iteration step k if $X_k = X_{k-1}$. The set X_k then contains all the filled holes. The set union of X_k and A contains all the filled holes and their boundaries.

The dilation in Eq. (9.5-2) would fill the entire area if left unchecked. However, the intersection at each step with A^c limits the result to inside the region of interest. This is our first example of how a morphological process can be *conditioned* to meet a desired property. In the current application, it is appropriately called *conditional dilation*. The rest of Fig. 9.15 illustrates further the mechanics of Eq. (9.5-2). Although this example only has one hole, the concept clearly applies to any finite number of holes, assuming that a point inside each hole region is given.

a	b	c
d	e	f
g	h	i

FIGURE 9.15 Hole filling. (a) Set A (shown shaded). (b) Complement of A . (c) Structuring element B . (d) Initial point inside the boundary. (e)–(h) Various steps of Eq. (9.5-2). (i) Final result [union of (a) and (h)].



EXAMPLE 9.6: Morphological hole filling.

Figure 9.16(a) shows an image composed of white circles with black inner spots. An image such as this might result from thresholding into two levels a scene containing polished spheres (e.g., ball bearings). The dark spots inside the spheres could be the result of reflections. The objective is to eliminate the reflections by hole filling. Figure 9.16(a) shows one point selected inside one of the spheres, and Fig. 9.16(b) shows the result of filling that component. Finally,

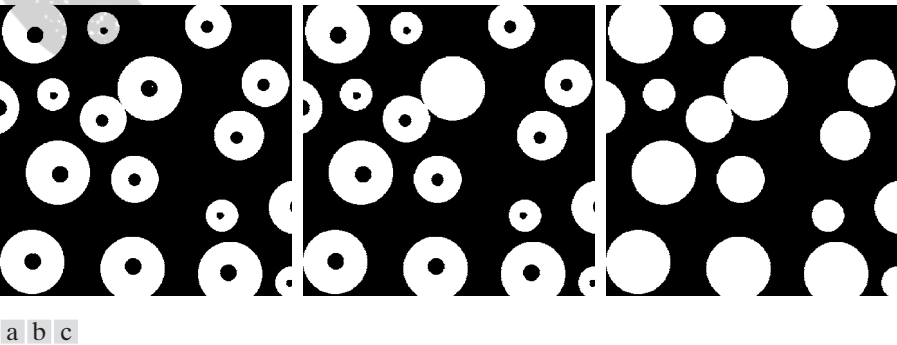


FIGURE 9.16 (a) Binary image (the white dot inside one of the regions is the starting point for the hole-filling algorithm). (b) Result of filling that region. (c) Result of filling all holes.

Fig. 9.16(c) shows the result of filling all the spheres. Because it must be known whether black points are background points or sphere inner points, fully automating this procedure requires that additional “intelligence” be built into the algorithm. We give a fully automatic approach in Section 9.5.9 based on morphological reconstruction. (See also Problem 9.23.) ■

9.5.3 Extraction of Connected Components

The concepts of connectivity and connected components were introduced in Section 2.5.2. Extraction of connected components from a binary image is central to many automated image analysis applications. Let A be a set containing one or more connected components, and form an array X_0 (of the same size as the array containing A) whose elements are 0s (background values), except at each location known to correspond to a point in each connected component in A , which we set to 1 (foreground value). The objective is to start with X_0 and find all the connected components. The following iterative procedure accomplishes this objective:

$$X_k = (X_{k-1} \oplus B) \cap A \quad k = 1, 2, 3, \dots \quad (9.5-3)$$

where B is a suitable structuring element (as in Fig. 9.17). The procedure terminates when $X_k = X_{k-1}$, with X_k containing all the connected components

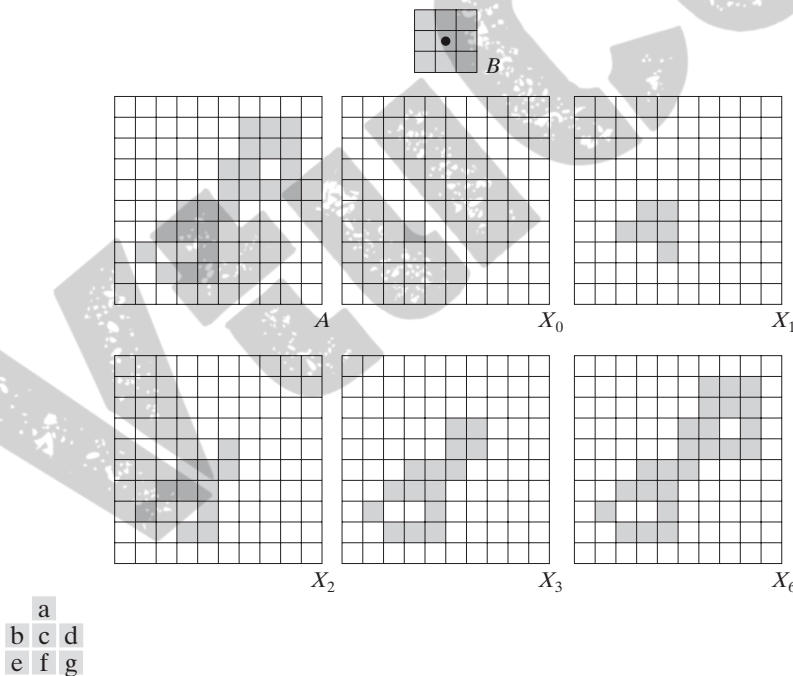


FIGURE 9.17 Extracting connected components. (a) Structuring element. (b) Array containing a set with one connected component. (c) Initial array containing a 1 in the region of the connected component. (d)–(g) Various steps in the iteration of Eq. (9.5-3).

of the input image. Note the similarity in Eqs. (9.5-3) and (9.5-2), the only difference being the use of A as opposed to A^c . This is not surprising, because here we are looking for foreground points, while the objective in Section 9.5.2 was to find background points.

Figure 9.17 illustrates the mechanics of Eq. (9.5-3), with convergence being achieved for $k = 6$. Note that the shape of the structuring element used is based on 8-connectivity between pixels. If we had used the SE in Fig. 9.15, which is based on 4-connectivity, the leftmost element of the connected component toward the bottom of the image would not have been detected because it is 8-connected to the rest of the figure. As in the hole-filling algorithm, Eq. (9.5-3) is applicable to any finite number of connected components contained in A , assuming that a point is known in each.

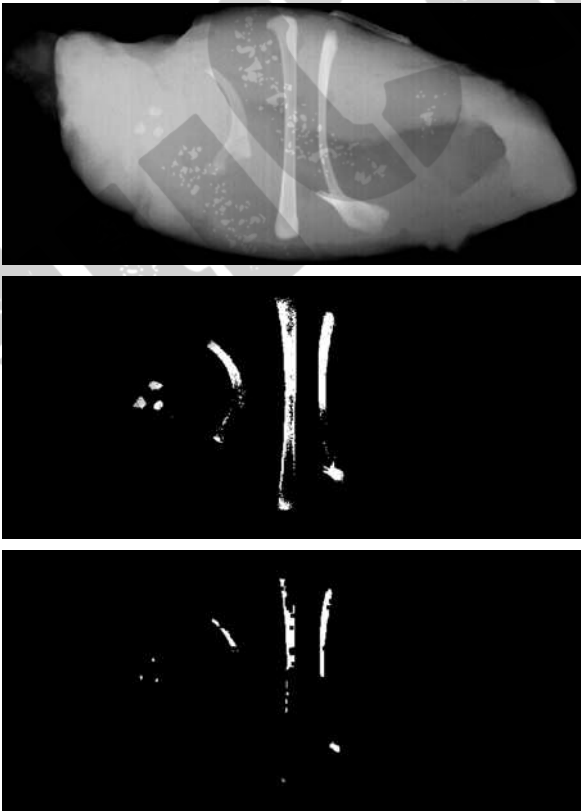
See Problem 9.24 for an algorithm that does not require that a point in each connected component be known a priori.

EXAMPLE 9.7: Using connected components to detect foreign objects in packaged food.

■ Connected components are used frequently for automated inspection. Figure 9.18(a) shows an X-ray image of a chicken breast that contains bone fragments. It is of considerable interest to be able to detect such objects in processed food before packaging and/or shipping. In this particular case, the density of the bones is such that their nominal intensity values are different from the background. This makes extraction of the bones from the background

a
b
c d

FIGURE 9.18
(a) X-ray image of chicken filet with bone fragments.
(b) Thresholded image. (c) Image eroded with a 5×5 structuring element of 1s.
(d) Number of pixels in the connected components of (c).
(Image courtesy of NTB Elektronische Geraete GmbH, Diepholz, Germany, www.ntbxray.com.)



Connected component	No. of pixels in connected comp
01	11
02	9
03	9
04	39
05	133
06	1
07	1
08	743
09	7
10	11
11	11
12	9
13	9
14	674
15	85

a simple matter by using a single threshold (thresholding was introduced in Section 3.1 and is discussed in more detail in Section 10.3). The result is the binary image in Fig. 9.18(b).

The most significant feature in this figure is the fact that the points that remain are clustered into objects (bones), rather than being isolated, irrelevant points. We can make sure that only objects of “significant” size remain by eroding the thresholded image. In this example, we define as significant any object that remains after erosion with a 5×5 structuring element of 1s. The result of erosion is shown in Fig. 9.18(c). The next step is to analyze the size of the objects that remain. We label (identify) these objects by extracting the connected components in the image. The table in Fig. 9.18(d) lists the results of the extraction. There are a total of 15 connected components, with four of them being dominant in size. This is enough to determine that significant undesirable objects are contained in the original image. If needed, further characterization (such as shape) is possible using the techniques discussed in Chapter 11. ■

9.5.4 Convex Hull

A set A is said to be *convex* if the straight line segment joining any two points in A lies entirely within A . The *convex hull* H of an arbitrary set S is the smallest convex set containing S . The set difference $H - S$ is called the *convex deficiency* of S . As discussed in more detail in Sections 11.1.6 and 11.3.2, the convex hull and convex deficiency are useful for object description. Here, we present a simple morphological algorithm for obtaining the convex hull, $C(A)$, of a set A .

Let $B^i, i = 1, 2, 3, 4$, represent the four structuring elements in Fig. 9.19(a). The procedure consists of implementing the equation:

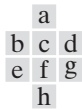
$$X_k^i = (X_{k-1} \otimes B^i) \cup A \quad i = 1, 2, 3, 4 \quad \text{and} \quad k = 1, 2, 3, \dots \quad (9.5-4)$$

with $X_0^i = A$. When the procedure converges (i.e., when $X_k^i = X_{k-1}^i$), we let $D^i = X_k^i$. Then the convex hull of A is

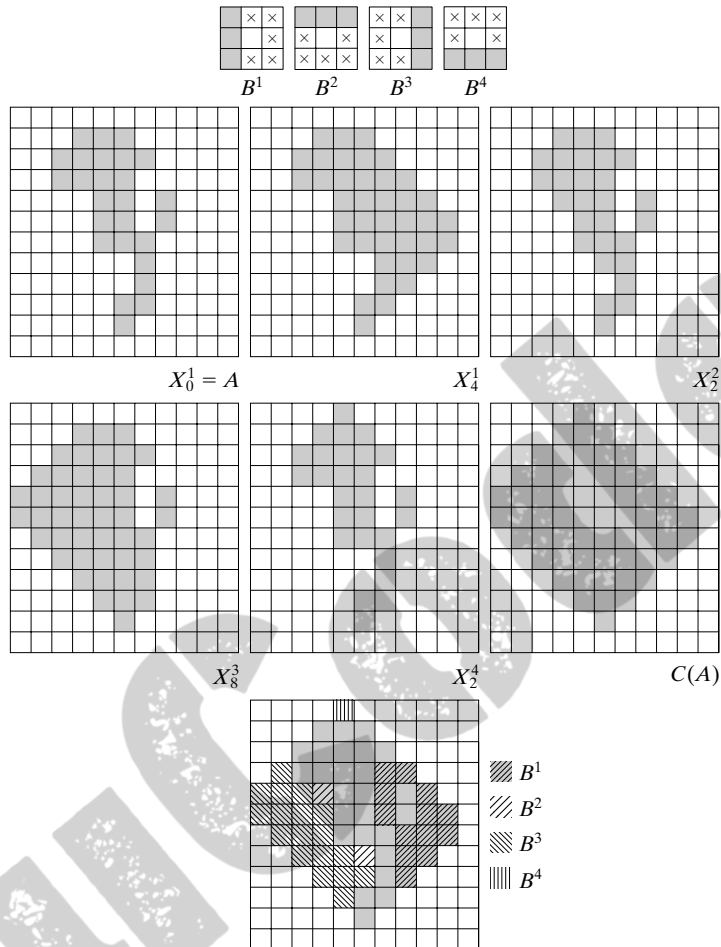
$$C(A) = \bigcup_{i=1}^4 D^i \quad (9.5-5)$$

In other words, the method consists of iteratively applying the hit-or-miss transform to A with B^1 ; when no further changes occur, we perform the union with A and call the result D^1 . The procedure is repeated with B^2 (applied to A) until no further changes occur, and so on. The union of the four resulting D s constitutes the convex hull of A . Note that we are using the simplified implementation of the hit-or-miss transform in which no background match is required, as discussed at the end of Section 9.4.

Figure 9.19 illustrates the procedure given in Eqs. (9.5-4) and (9.5-5). Figure 9.19(a) shows the structuring elements used to extract the convex hull. The origin of each element is at its center. The \times entries indicate “don’t care” conditions. This means that a structuring element is said to have found a match

**FIGURE 9.19**

(a) Structuring elements. (b) Set A . (c)–(f) Results of convergence with the structuring elements shown in (a). (g) Convex hull. (h) Convex hull showing the contribution of each structuring element.



in A if the 3×3 region of A under the structuring element mask at that location matches the pattern of the mask. For a particular mask, a pattern match occurs when the center of the 3×3 region in A is 0, and the three pixels under the shaded mask elements are 1. The values of the other pixels in the 3×3 region do not matter. Also, with respect to the notation in Fig. 9.19(a), B^i is a clockwise rotation of B^{i-1} by 90° .

Figure 9.19(b) shows a set A for which the convex hull is sought. Starting with $X_0^1 = A$ resulted in the set in Fig. 9.19(c) after four iterations of Eq. (9.5-4). Then, letting $X_0^2 = A$ and again using Eq. (9.5-4) resulted in the set in Fig. 9.19(d) (convergence was achieved in only two steps in this case). The next two results were obtained in the same way. Finally, forming the union of the sets in Figs. 9.19(c), (d), (e), and (f) resulted in the convex hull shown in Fig. 9.19(g). The contribution of each structuring element is highlighted in the composite set shown in Fig. 9.19(h).

One obvious shortcoming of the procedure just outlined is that the convex hull can grow beyond the minimum dimensions required to guarantee

convexity. One simple approach to reduce this effect is to limit growth so that it does not extend past the vertical and horizontal dimensions of the original set of points. Imposing this limitation on the example in Fig. 9.19 resulted in the image shown in Fig. 9.20. Boundaries of greater complexity can be used to limit growth even further in images with more detail. For example, we could use the maximum dimensions of the original set of points along the vertical, horizontal, and diagonal directions. The price paid for refinements such as this is additional complexity and increased computational requirements of the algorithm.

9.5.5 Thinning

The thinning of a set A by a structuring element B , denoted $A \otimes B$, can be defined in terms of the hit-or-miss transform:

$$\begin{aligned} A \otimes B &= A - (A \circledast B) \\ &= A \cap (A \circledast B)^c \end{aligned} \quad (9.5-6)$$

As in the previous section, we are interested only in pattern matching with the structuring elements, so no background operation is required in the hit-or-miss transform. A more useful expression for thinning A symmetrically is based on a *sequence* of structuring elements:

$$\{B\} = \{B^1, B^2, B^3, \dots, B^n\} \quad (9.5-7)$$

where B^i is a rotated version of B^{i-1} . Using this concept, we now define thinning by a sequence of structuring elements as

$$A \otimes \{B\} = (((\dots((A \otimes B^1) \otimes B^2) \dots) \otimes B^n) \quad (9.5-8)$$

The process is to thin A by *one pass* with B^1 , then thin the result with one pass of B^2 , and so on, until A is thinned with one pass of B^n . The entire process is repeated until no further changes occur. Each individual thinning pass is performed using Eq. (9.5-6).

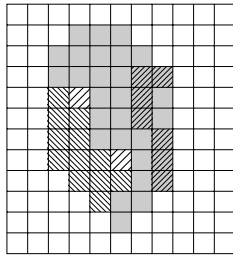


FIGURE 9.20

Result of limiting growth of the convex hull algorithm to the maximum dimensions of the original set of points along the vertical and horizontal directions.

Figure 9.21(a) shows a set of structuring elements commonly used for thinning, and Fig. 9.21(b) shows a set A to be thinned by using the procedure just discussed. Figure 9.21(c) shows the result of thinning after one pass of A with B^1 , and Figs. 9.21(d) through (k) show the results of passes with the other structuring elements. Convergence was achieved after the second pass of B^6 . Figure 9.21(l) shows the thinned result. Finally, Fig. 9.21(m) shows the thinned set converted to m -connectivity (see Section 2.5.2) to eliminate multiple paths.

9.5.6 Thickening

Thickening is the morphological dual of thinning and is defined by the expression

$$A \odot B = A \cup (A \otimes B) \quad (9.5-9)$$

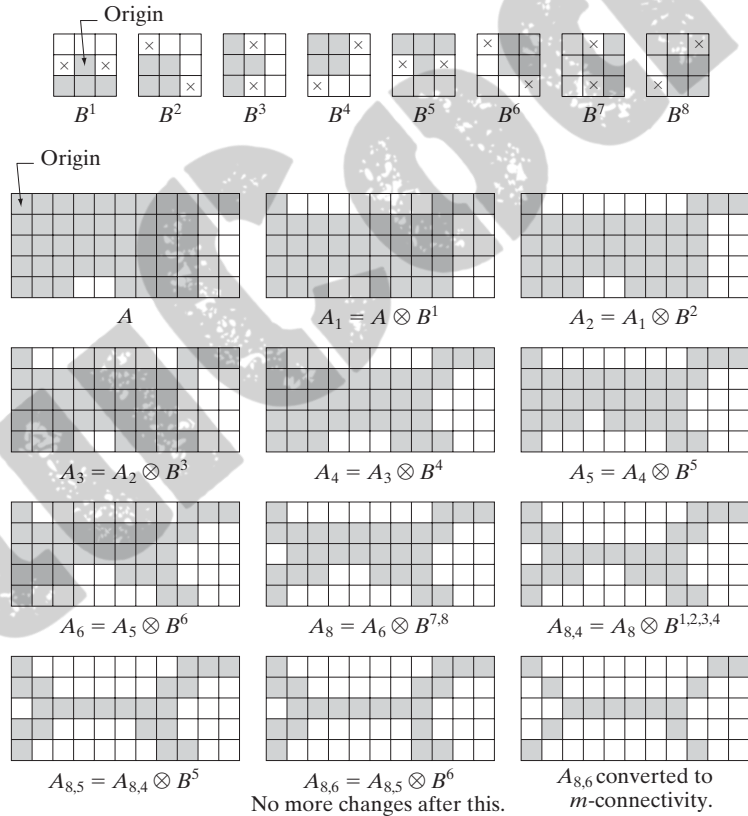


FIGURE 9.21 (a) Sequence of rotated structuring elements used for thinning. (b) Set A . (c) Result of thinning with the first element. (d)–(i) Results of thinning with the next seven elements (there was no change between the seventh and eighth elements). (j) Result of using the first four elements again. (l) Result after convergence. (m) Conversion to m -connectivity.

a
b c d
e f g
h i j
k l m

where B is a structuring element suitable for thickening. As in thinning, thickening can be defined as a sequential operation:

$$A \odot \{B\} = ((\dots((A \odot B^1) \odot B^2) \dots) \odot B^n) \quad (9.5-10)$$

The structuring elements used for thickening have the same form as those shown in Fig. 9.21(a), but with all 1s and 0s interchanged. However, a separate algorithm for thickening is seldom used in practice. Instead, the usual procedure is to thin the background of the set in question and then complement the result. In other words, to thicken a set A , we form $C = A^c$, thin C , and then form C^c . Figure 9.22 illustrates this procedure.

Depending on the nature of A , this procedure can result in disconnected points, as Fig. 9.22(d) shows. Hence thickening by this method usually is followed by postprocessing to remove disconnected points. Note from Fig. 9.22(c) that the thinned background forms a boundary for the thickening process. This useful feature is not present in the direct implementation of thickening using Eq. (9.5-10), and it is one of the principal reasons for using background thinning to accomplish thickening.

9.5.7 Skeletons

As Fig. 9.23 shows, the notion of a skeleton, $S(A)$, of a set A is intuitively simple. We deduce from this figure that

- (a) If z is a point of $S(A)$ and $(D)_z$ is the largest disk centered at z and contained in A , one cannot find a larger disk (not necessarily centered at z) containing $(D)_z$ and included in A . The disk $(D)_z$ is called a *maximum disk*.
- (b) The disk $(D)_z$ touches the boundary of A at two or more different places.

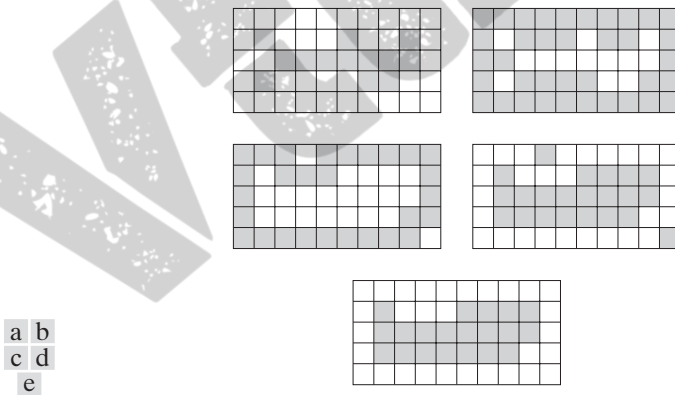
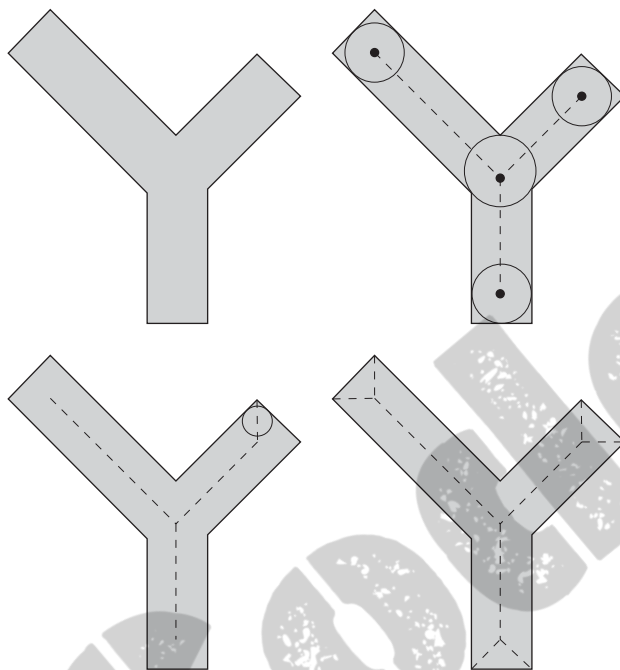


FIGURE 9.22 (a) Set A . (b) Complement of A . (c) Result of thinning the complement of A . (d) Thickened set obtained by complementing (c). (e) Final result, with no disconnected points.

a	b
c	d

FIGURE 9.23

(a) Set A .
 (b) Various positions of maximum disks with centers on the skeleton of A .
 (c) Another maximum disk on a different segment of the skeleton of A .
 (d) Complete skeleton.



The skeleton of A can be expressed in terms of erosions and openings. That is, it can be shown (Serra [1982]) that

$$S(A) = \bigcup_{k=0}^K S_k(A) \quad (9.5-11)$$

with

$$S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B \quad (9.5-12)$$

where B is a structuring element, and $(A \ominus kB)$ indicates k successive erosions of A :

$$(A \ominus kB) = ((\dots((A \ominus B) \ominus B) \ominus \dots) \ominus B) \quad (9.5-13)$$

k times, and K is the last iterative step before A erodes to an empty set. In other words,

$$K = \max\{k | (A \ominus kB) \neq \emptyset\} \quad (9.5-14)$$

The formulation given in Eqs. (9.5-11) and (9.5-12) states that $S(A)$ can be obtained as the union of the *skeleton subsets* $S_k(A)$. Also, it can be shown that A can be *reconstructed* from these subsets by using the equation

$$A = \bigcup_{k=0}^K (S_k(A) \oplus kB) \quad (9.5-15)$$

where $(S_k(A) \oplus kB)$ denotes k successive dilations of $S_k(A)$; that is,

$$(S_k(A) \oplus kB) = ((\dots((S_k(A) \oplus B) \oplus B) \oplus \dots) \oplus B) \quad (9.5-16)$$

■ Figure 9.24 illustrates the concepts just discussed. The first column shows the original set (at the top) and two erosions by the structuring element B . Note that one more erosion of A would yield the empty set, so $K = 2$ in this case. The second column shows the opening of the sets in the first column by B . These results are easily explained by the fitting characterization of the opening operation discussed in connection with Fig. 9.8. The third column simply contains the set differences between the first and second columns.

The fourth column contains two partial skeletons and the final result (at the bottom of the column). The final skeleton not only is thicker than it needs to be but, more important, it is not connected. This result is not unexpected, as nothing in the preceding formulation of the morphological skeleton guarantees connectivity. Morphology produces an elegant formulation in terms of erosions and openings of the given set. However, heuristic formulations such as the algorithm developed in Section 11.1.7 are needed if, as is usually the case, the skeleton must be maximally thin, connected, and minimally eroded.

EXAMPLE 9.8:

Computing the skeleton of a simple figure.

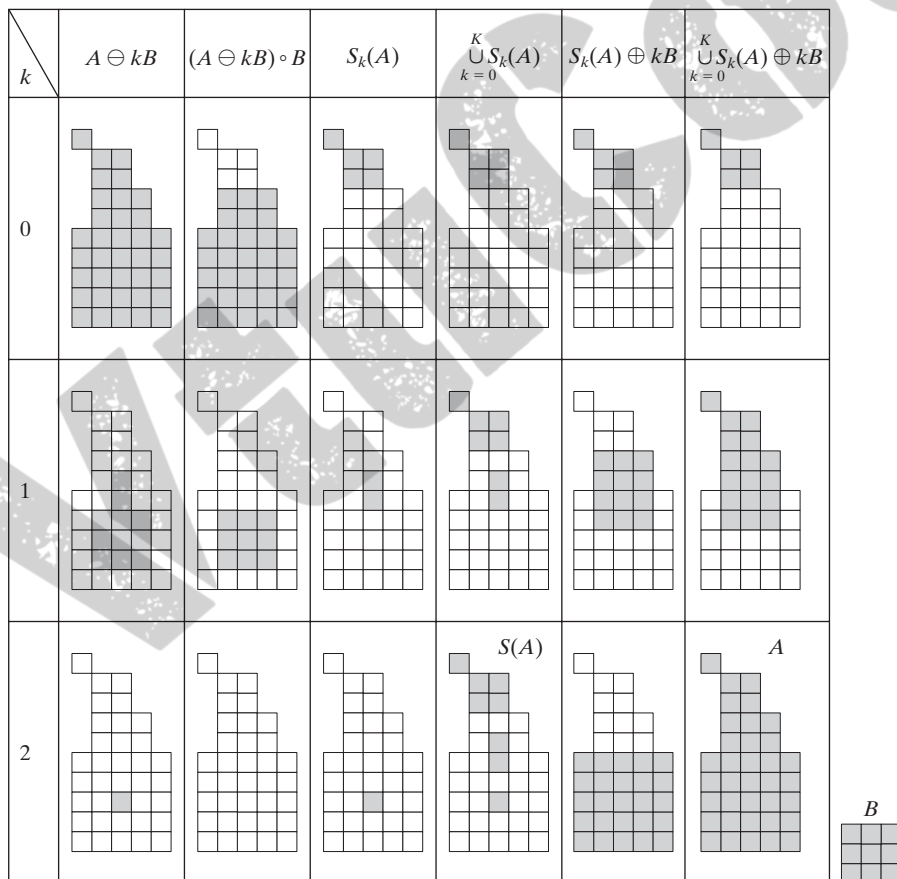


FIGURE 9.24

Implementation of Eqs. (9.5-11) through (9.5-15). The original set is at the top left, and its morphological skeleton is at the bottom of the fourth column. The reconstructed set is at the bottom of the sixth column.

The fifth column shows $S_0(A)$, $S_1(A) \oplus B$, and $(S_2(A) \oplus 2B) = (S_2(A) \oplus B) \oplus B$. Finally, the last column shows reconstruction of set A , which, according to Eq. (9.5-15), is the union of the dilated skeleton subsets shown in the fifth column. ■

9.5.8 Pruning

Pruning methods are an essential complement to thinning and skeletonizing algorithms because these procedures tend to leave parasitic components that need to be “cleaned up” by postprocessing. We begin the discussion with a pruning problem and then develop a morphological solution based on the material introduced in the preceding sections. Thus, we take this opportunity to illustrate how to go about solving a problem by combining several of the techniques discussed up to this point.

A common approach in the automated recognition of hand-printed characters is to analyze the shape of the skeleton of each character. These skeletons often are characterized by “spurs” (parasitic components). Spurs are caused during erosion by non uniformities in the strokes composing the characters. We develop a morphological technique for handling this problem, starting with the assumption that the length of a parasitic component does not exceed a specified number of pixels.

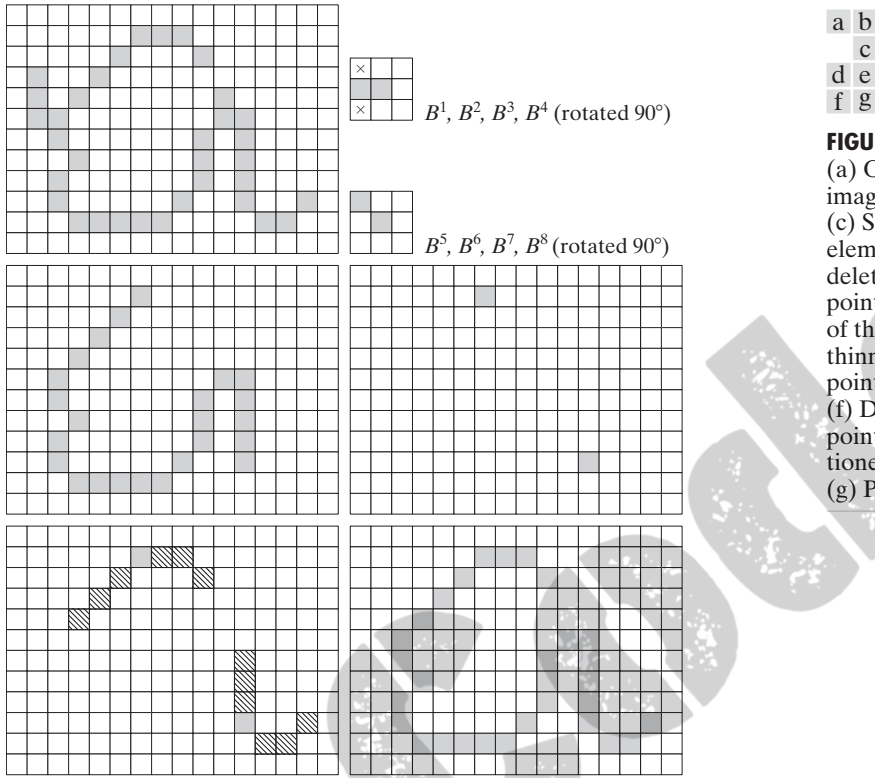
Figure 9.25(a) shows the skeleton of a hand-printed “a.” The parasitic component on the leftmost part of the character is illustrative of what we are interested in removing. The solution is based on suppressing a parasitic branch by successively eliminating its end point. Of course, this also shortens (or eliminates) other branches in the character but, in the absence of other structural information, the assumption in this example is that any branch with three or less pixels is to be eliminated. Thinning of an input set A with a sequence of structuring elements designed to detect only end points achieves the desired result. That is, let

$$X_1 = A \otimes \{B\} \quad (9.5-17)$$

where $\{B\}$ denotes the structuring element sequence shown in Figs. 9.25(b) and (c) [see Eq. (9.5-7) regarding structuring-element sequences]. The sequence of structuring elements consists of two different structures, each of which is rotated 90° for a total of eight elements. The \times in Fig. 9.25(b) signifies a “don’t care” condition, in the sense that it does not matter whether the pixel in that location has a value of 0 or 1. Numerous results reported in the literature on morphology are based on the use of a *single* structuring element, similar to the one in Fig. 9.25(b), but having “don’t care” conditions along the entire first column. This is incorrect. For example, this element would identify the point located in the eighth row, fourth column of Fig. 9.25(a) as an end point, thus eliminating it and breaking connectivity in the stroke.

Applying Eq. (9.5-17) to A three times yields the set X_1 in Fig. 9.25(d). The next step is to “restore” the character to its original form, but with the parasitic

We may define an *end point* as the center point of a 3×3 region that satisfies any of the arrangements in Figs. 9.25(b) or (c).


FIGURE 9.25

(a) Original image. (b) and (c) Structuring elements used for deleting end points. (d) Result of three cycles of thinning. (e) End points of (d). (f) Dilation of end points conditioned on (a). (g) Pruned image.

branches removed. To do so first requires forming a set X_2 containing all end points in X_1 [Fig. 9.25(e)]:

$$X_2 = \bigcup_{k=1}^8 (X_1 \otimes B^k) \quad (9.5-18)$$

where the B^k are the same end-point detectors shown in Figs. 9.25(b) and (c). The next step is dilation of the end points three times, using set A as a delimiter:

$$X_3 = (X_2 \oplus H) \cap A \quad (9.5-19)$$

where H is a 3×3 structuring element of 1s and the intersection with A is applied after each step. As in the case of region filling and extraction of connected components, this type of conditional dilation prevents the creation of 1-valued elements outside the region of interest, as evidenced by the result shown in Fig. 9.25(f). Finally, the union of X_3 and X_1 yields the desired result,

$$X_4 = X_1 \cup X_3 \quad (9.5-20)$$

in Fig. 9.25(g).

In more complex scenarios, use of Eq. (9.5-19) sometimes picks up the “tips” of some parasitic branches. This condition can occur when the end

Equation (9.5-19) is the basis for morphological reconstruction by dilation, as explained in the next section.

points of these branches are near the skeleton. Although Eq. (9.5-17) may eliminate them, they can be picked up again during dilation because they are valid points in A . Unless entire parasitic elements are picked up again (a rare case if these elements are short with respect to valid strokes), detecting and eliminating them is easy because they are disconnected regions.

A natural thought at this juncture is that there must be easier ways to solve this problem. For example, we could just keep track of all deleted points and simply reconnect the appropriate points to all end points left after application of Eq. (9.5-17). This option is valid, but the advantage of the formulation just presented is that the use of simple morphological constructs solved the entire problem. In practical situations when a set of such tools is available, the advantage is that no new algorithms have to be written. We simply combine the necessary morphological functions into a sequence of operations.

9.5.9 Morphological Reconstruction

The morphological concepts discussed thus far involve an image and a structuring element. In this section, we discuss a powerful morphological transformation called *morphological reconstruction* that involves two images and a structuring element. One image, the *marker*, contains the starting points for the transformation. The other image, the *mask*, constrains the transformation. The structuring element is used to define connectivity.[†]

Geodesic dilation and erosion

Central to morphological reconstruction are the concepts of geodesic dilation and geodesic erosion. Let F denote the marker image and G the mask image. It is assumed in this discussion that both are binary images and that $F \subseteq G$. The *geodesic dilation* of size 1 of the marker image with respect to the mask, denoted by $D_G^{(1)}(F)$, is defined as

$$D_G^{(1)}(F) = (F \oplus B) \cap G \quad (9.5-21)$$

where \cap denotes the set intersection (here \cap may be interpreted as a logical AND because the set intersection and logical AND operations are the same for binary sets). The geodesic dilation of size n of F with respect to G is defined as

$$D_G^{(n)}(F) = D_G^{(1)}[D_G^{(n-1)}(F)] \quad (9.5-22)$$

with $D_G^{(0)}(F) = F$. In this recursive expression, the set intersection in Eq. (9.5-21) is performed at each step.[‡] Note that the intersection operator guarantees that

[†]In much of the literature on morphological reconstruction, the structuring element is tacitly assumed to be isotropic and typically is called an *elementary isotropic structuring element*. In the context of this chapter, an example of such an SE is simply a 3×3 array of 1s with the origin at the center.

[‡]Although it is more intuitive to develop morphological-reconstruction methods using recursive formulations (as we do here), their practical implementation typically is based on more computationally efficient algorithms (see, for example, Vincent [1993] and Soille [2003]). All image-based examples in this section were generated using such algorithms.

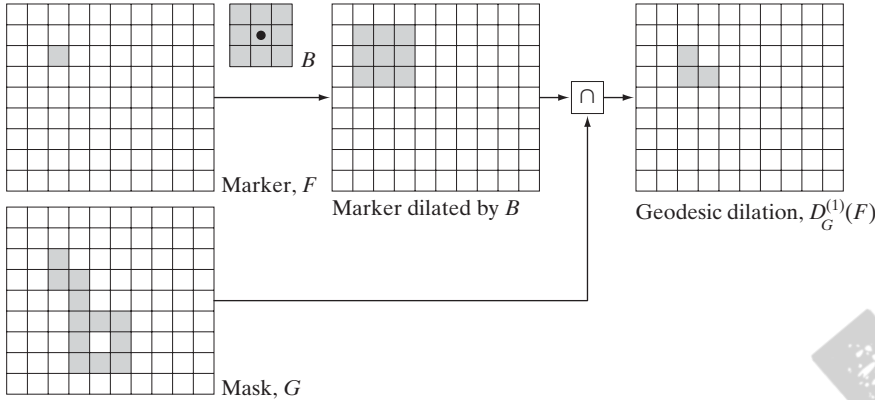


FIGURE 9.26
Illustration of
geodesic dilation.

mask G will limit the growth (dilation) of marker F . Figure 9.26 shows a simple example of a geodesic dilation of size 1. The steps in the figure are a direct implementation of Eq. (9.5-21).

Similarly, the *geodesic erosion* of size 1 of marker F with respect to mask G is defined as

$$E_G^{(1)}(F) = (F \ominus B) \cup G \quad (9.5-23)$$

where \cup denotes set union (or OR operation). The geodesic erosion of size n of F with respect to G is defined as

$$E_G^{(n)}(F) = E_G^{(1)}[E_G^{(n-1)}(F)] \quad (9.5-24)$$

with $E_G^{(0)}(F) = F$. The set union operation in Eq. (9.5-23) is performed at each iterative step, and guarantees that geodesic erosion of an image remains greater than or equal to its mask image. As expected from the forms in Eqs. (9.5-21) and (9.5-23), geodesic dilation and erosion are *duals* with respect to set complementation (see Problem 9.29). Figure 9.27 shows a simple example of geodesic erosion of size 1. The steps in the figure are a direct implementation of Eq. (9.5-23).

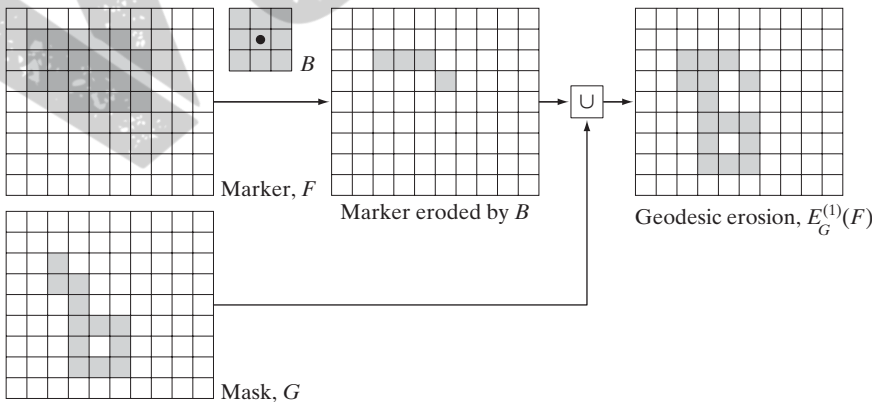


FIGURE 9.27
Illustration of
geodesic erosion.

Geodesic dilation and erosion of finite images always converge after a finite number of iterative step because propagation or shrinking of the marker image is constrained by the mask.

Morphological reconstruction by dilation and by erosion

Based on the preceding concepts, *morphological reconstruction by dilation* of a mask image G from a marker image F , denoted $R_G^D(F)$, is defined as the geodesic dilation of F with respect to G , iterated until stability is achieved; that is,

$$R_G^D(F) = D_G^k(F) \quad (9.5-25)$$

with k such that $D_G^k(F) = D_G^{k+1}(F)$.

Figure 9.28 illustrates reconstruction by dilation. Figure 9.28(a) continues the process begun in Fig. 9.26; that is, the next step in reconstruction after obtaining $D_G^{(1)}(F)$ is to dilate this result and then AND it with the mask G to yield $D_G^{(2)}(F)$, as Fig. 9.28(b) shows. Dilation of $D_G^{(2)}(F)$ and masking with G then yields $D_G^{(3)}(F)$, and so on. This procedure is repeated until stability is reached. If we carried this example one more step, we would find that $D_G^{(5)}(F) = D_G^{(6)}(F)$, so the morphologically reconstructed image by dilation is given by $R_G^D(F) = D_G^{(5)}(F)$, as indicated in Eq. (9.5-25). Note that the reconstructed image in this case is identical to the mask because F contained a single 1-valued pixel (this is analogous to convolution of an image with an impulse, which simply copies the image at the location of the impulse, as explained in Section 3.4.2).

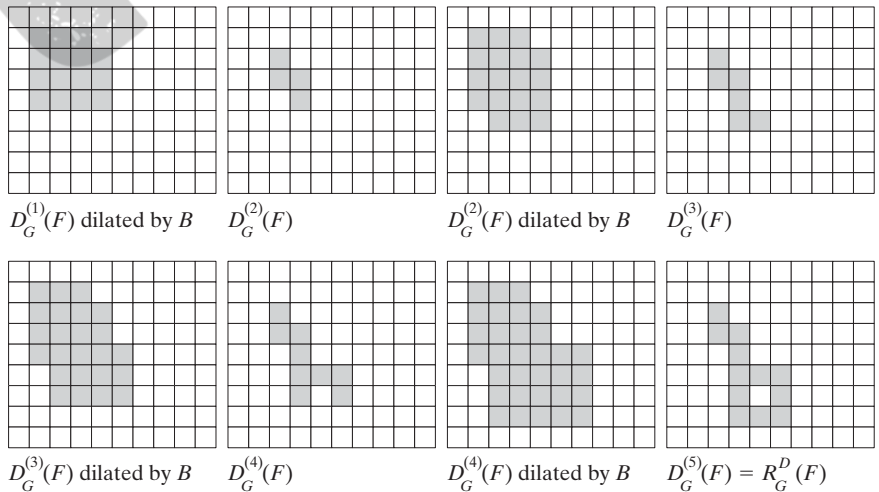
In a similar manner, the *morphological reconstruction by erosion* of a mask image G from a marker image F , denoted $R_G^E(F)$, is defined as the geodesic erosion of F with respect to G , iterated until stability; that is,

$$R_G^E(F) = E_G^k(F) \quad (9.5-26)$$

with k such that $E_G^k(F) = E_G^{k+1}(F)$. As an exercise, you should generate a figure similar to Fig. 9.28 for morphological reconstruction by erosion.

a b c d
e f g h

FIGURE 9.28
Illustration of morphological reconstruction by dilation. F , G , B and $D_G^{(1)}(F)$ are from Fig. 9.26.



Reconstruction by dilation and erosion are duals with respect to set complementation (see Problem 9.30).

Sample applications

Morphological reconstruction has a broad spectrum of practical applications, each determined by the selection of the marker and mask images, by the structuring elements used, and by combinations of the primitive operations defined in the preceding discussion. The following examples illustrate the usefulness of these concepts.

Opening by reconstruction: In a morphological opening, erosion removes small objects and the subsequent dilation attempts to restore the shape of objects that remain. However, the accuracy of this restoration is highly dependent on the similarity of the shapes of the objects and the structuring element used. *Opening by reconstruction* restores *exactly* the shapes of the objects that remain after erosion. The opening by reconstruction of size n of an image F is defined as the reconstruction by dilation of F from the erosion of size n of F ; that is,

$$O_R^{(n)}(F) = R_F^D[(F \ominus nB)] \quad (9.5-27)$$

where $(F \ominus nB)$ indicates n erosions of F by B , as explained in Section 9.5.7. Note that F is used as the mask in this application. A similar expression can be written for closing by reconstruction (see Table 9.1).

Figure 9.29 shows an example of opening by reconstruction. In this illustration, we are interested in extracting from Fig. 9.29(a) the characters that contain long, vertical strokes. Opening by reconstruction requires at least one erosion, so we perform that step first. Figure 9.29(b) shows the erosion

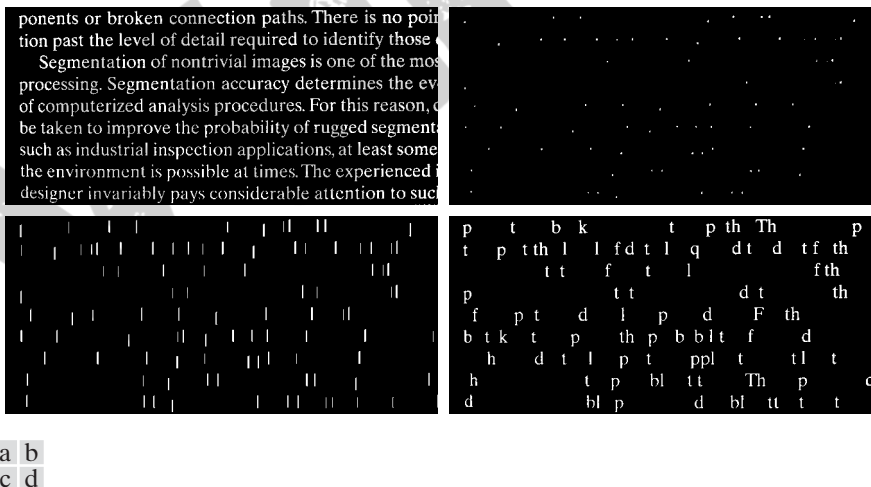


FIGURE 9.29 (a) Text image of size 918×2018 pixels. The approximate average height of the tall characters is 50 pixels. (b) Erosion of (a) with a structuring element of size 51×1 pixels. (c) Opening of (a) with the same structuring element, shown for reference. (d) Result of opening by reconstruction.

of Fig. 9.29(a) with a structuring element of length proportional to the average height of the tall characters (51 pixels) and width of one pixel. For the purpose of comparison, we computed the opening of the image using the same structuring element. Figure 9.29(c) shows the result. Finally, Fig. 9.29(d) is the opening by reconstruction (of size 1) of F [i.e., $O_R^{(1)}(F)$] given in Eq. (9.5-27). This result shows that characters containing long vertical strokes were restored accurately; all other characters were removed.

Filling holes: In Section 9.5.2, we developed an algorithm for filling holes based on knowing a starting point in each hole in the image. Here, we develop a fully automated procedure based on morphological reconstruction. Let $I(x, y)$ denote a binary image and suppose that we form a marker image F that is 0 everywhere, except at the image border, where it is set to $1 - I$; that is,

$$F(x, y) = \begin{cases} 1 - I(x, y) & \text{if } (x, y) \text{ is on the border of } I \\ 0 & \text{otherwise} \end{cases} \quad (9.5-28)$$

Then

$$H = [R_{I^c}^D(F)]^c \quad (9.5-29)$$

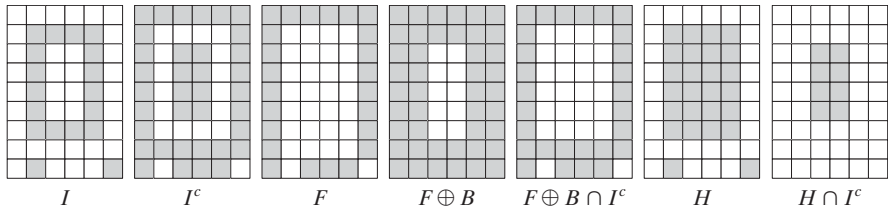
is a binary image equal to I with all holes filled.

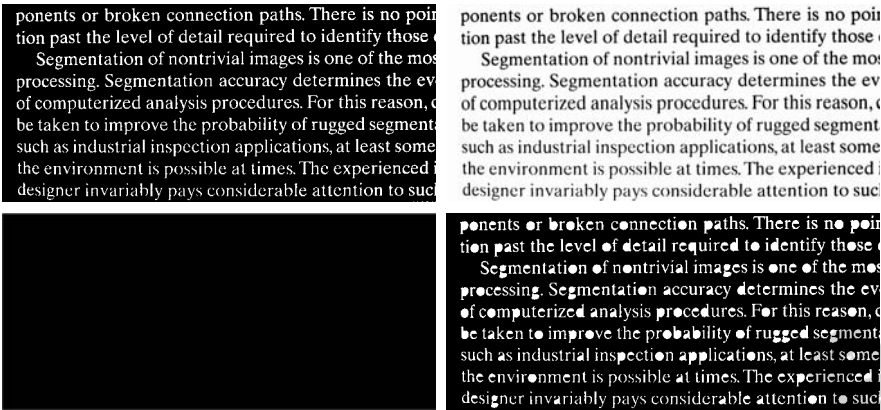
Let us consider the individual components of Eq. (9.5-29) to see how this expression in fact leads to all holes in an image being filled. Figure 9.30(a) shows a simple image I containing one hole, and Fig. 9.30(b) shows its complement. Note that because the complement of I sets all foreground (1-valued) pixels to background (0-valued) pixels, and vice versa, this operation in effect builds a “wall” of 0s around the hole. Because I^c is used as an AND mask, all we are doing here is protecting all foreground pixels (including the wall around the hole) from changing during iteration of the procedure. Figure 9.30(c) is array F formed according to Eq. (9.5-28) and Fig. 9.30(d) is F dilated with a 3×3 SE whose elements are all 1s. Note that marker F has a border of 1s (except at locations where I is 1), so the dilation of F of the marker points starts at the border and proceeds inward. Figure 9.30(e) shows the geodesic dilation of F using I^c as the mask. As was just indicated, we see that all locations in this result corresponding to foreground pixels from I are 0, and that this is true now for the hole pixels as well. Another iteration will yield the same result which, when complemented as required by Eq. (9.5-29), gives the result in Fig. 9.30(f). As desired, the hole is now filled and the rest of image I was unchanged. The operation $H \cap I^c$ yields an image containing 1-valued pixels in the locations corresponding to the holes in I , as Fig. 9.30(g) shows.

a b c d e f g

FIGURE 9.30

Illustration of hole filling on a simple image.



a b
c d**FIGURE 9.31**

(a) Text image of size 918×2018 pixels. (b) Complement of (a) for use as a mask image. (c) Marker image. (d) Result of hole-filling using Eq. (9.5-29).

Figure 9.31 shows a more practical example. Figure 9.31(b) shows the complement of the text image in Fig. 9.31(a), and Fig. 9.31(c) is the marker image, F , generated using Eq. (9.5-28). This image has a border of 1s, except at locations corresponding to 1s in the border of the original image. Finally, Fig. 9.31(d) shows the image with all the holes filled.

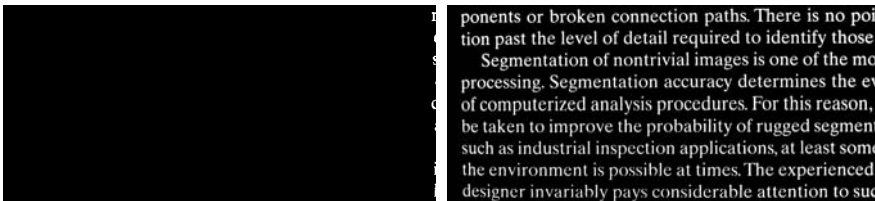
Border clearing: The extraction of objects from an image for subsequent shape analysis is a fundamental task in automated image processing. An algorithm for removing objects that touch (i.e., are connected to) the border is a useful tool because (1) it can be used to screen images so that only complete objects remain for further processing, or (2) it can be used as a signal that partial objects are present in the field of view. As a final illustration of the concepts introduced in this section, we develop a border-clearing procedure based on morphological reconstruction. In this application, we use the original image as the mask and the following marker image:

$$F(x, y) = \begin{cases} I(x, y) & \text{if } (x, y) \text{ is on the border of } I \\ 0 & \text{otherwise} \end{cases} \quad (9.5-30)$$

The border-clearing algorithm first computes the morphological reconstruction $R_F^D(I)$ (which simply extracts the objects touching the border) and then computes the difference

$$X = I - R_F^D(I) \quad (9.5-31)$$

to obtain an image, X , with no objects touching the border.

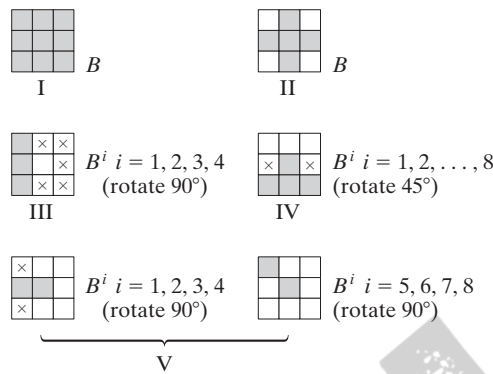


a b

FIGURE 9.32

Border clearing. (a) Marker image. (b) Image with no objects touching the border. The original image is Fig. 9.29(a).

FIGURE 9.33 Five basic types of structuring elements used for binary morphology. The origin of each element is at its center and the ×'s indicate “don't care” values.



As an example, consider the text image again. Figure 9.32(a) in the previous page shows the reconstruction $R_I^D(F)$ obtained using a 3×3 structuring element of all 1s (note the objects touching the boundary on the right side), and Fig. 9.32(b) shows image X , computed using Eq. (9.5-31). If the task at hand were automated character recognition, having an image in which no characters touch the border is most useful because the problem of having to recognize partial characters (a difficult task at best) is avoided.

9.5.10 Summary of Morphological Operations on Binary Images

Table 9.1 summarizes the morphological results developed in the preceding sections, and Fig. 9.33 summarizes the basic types of structuring elements used in the various morphological processes discussed thus far. The Roman numerals in the third column of Table 9.1 refer to the structuring elements in Fig. 9.33.

TABLE 9.1
Summary of morphological operations and their properties.

Operation	Equation	Comments (The Roman numerals refer to the structuring elements in Fig. 9.33.)
Translation	$(B)_z = \{w w = b + z, \text{ for } b \in B\}$	Translates the origin of B to point z .
Reflection	$\hat{B} = \{w w = -b, \text{ for } b \in B\}$	Reflects all elements of B about the origin of this set.
Complement	$A^c = \{w w \notin A\}$	Set of points not in A .
Difference	$A - B = \{w w \in A, w \notin B\}$ $= A \cap B^c$	Set of points that belong to A but not to B .
Dilation	$A \oplus B = \{z (\hat{B}_z) \cap A \neq \emptyset\}$	“Expands” the boundary of A . (I)
Erosion	$A \ominus B = \{z (B)_z \subseteq A\}$	“Contracts” the boundary of A . (I)
Opening	$A \circ B = (A \ominus B) \oplus B$	Smooths contours, breaks narrow isthmuses, and eliminates small islands and sharp peaks. (I)

(Continued)

TABLE 9.1
(Continued)

Operation	Equation	Comments (The Roman numerals refer to the structuring elements in Fig. 9.33.)
Closing	$A \bullet B = (A \oplus B) \ominus B$	Smooths contours, fuses narrow breaks and long thin gulfs, and eliminates small holes. (I)
Hit-or-miss transform	$A \otimes B = (A \ominus B_1) \cap (A^c \ominus B_2)$ $= (A \ominus B_1) - (A \oplus \hat{B}_2)$	The set of points (coordinates) at which, simultaneously, B_1 found a match ("hit") in A and B_2 found a match in A^c
Boundary extraction	$\beta(A) = A - (A \ominus B)$	Set of points on the boundary of set A . (I)
Hole filling	$X_k = (X_{k-1} \oplus B) \cap A^c$; $k = 1, 2, 3, \dots$	Fills holes in A ; X_0 = array of 0s with a 1 in each hole. (II)
Connected components	$X_k = (X_{k-1} \oplus B) \cap A$; $k = 1, 2, 3, \dots$	Finds connected components in A ; X_0 = array of 0s with a 1 in each connected component. (I)
Convex hull	$X_k^i = (X_{k-1}^i \otimes B^i) \cup A$; $i = 1, 2, 3, 4$; $k = 1, 2, 3, \dots$; $X_0^i = A$; and $D^i = X_{\text{conv}}^i$	Finds the convex hull $C(A)$ of set A , where "conv" indicates convergence in the sense that $X_k^i = X_{k-1}^i$. (III)
Thinning	$A \otimes B = A - (A \otimes B)$ $= A \cap (A \otimes B)^c$ $A \otimes \{B\} =$ $((\dots((A \otimes B^1) \otimes B^2) \dots) \otimes B^n)$ $\{B\} = \{B^1, B^2, B^3, \dots, B^n\}$	Thins set A . The first two equations give the basic definition of thinning. The last equations denote thinning by a sequence of structuring elements. This method is normally used in practice. (IV)
Thickening	$A \odot B = A \cup (A \otimes B)$ $A \odot \{B\} =$ $((\dots(A \odot B^1) \odot B^2 \dots) \odot B^n)$	Thickens set A . (See preceding comments on sequences of structuring elements.) Uses IV with 0s and 1s reversed.
Skeletons	$S(A) = \bigcup_{k=0}^K S_k(A)$ $S_k(A) = \bigcup_{k=0}^K \{(A \ominus kB)$ $- [(A \ominus kB) \circ B]\}$ Reconstruction of A : $A = \bigcup_{k=0}^K (S_k(A) \oplus kB)$	Finds the skeleton $S(A)$ of set A . The last equation indicates that A can be reconstructed from its skeleton subsets $S_k(A)$. In all three equations, K is the value of the iterative step after which the set A erodes to the empty set. The notation $(A \ominus kB)$ denotes the k th iteration of successive erosions of A by B . (I)

(Continued)

TABLE 9.1

(Continued)

Operation	Equation	Comments (The Roman numerals refer to the structuring elements in Fig. 9.33.)
Pruning	$X_1 = A \otimes \{B\}$ $X_2 = \bigcup_{k=1}^8 (X_1 \otimes B^k)$ $X_3 = (X_2 \oplus H) \cap A$ $X_4 = X_1 \cup X_3$	X_4 is the result of pruning set A . The number of times that the first equation is applied to obtain X_1 must be specified. Structuring elements V are used for the first two equations. In the third equation H denotes structuring element I .
Geodesic dilation of size 1	$D_G^{(1)}(F) = (F \oplus B) \cap G$	F and G are called the <i>marker</i> and <i>mask</i> images, respectively.
Geodesic dilation of size n	$D_G^{(n)}(F) = D_G^{(1)}[D_G^{(n-1)}(F)];$ $D_G^{(0)}(F) = F$	
Geodesic erosion of size 1	$E_G^{(1)}(F) = (F \ominus B) \cup G$	
Geodesic erosion of size n	$E_G^{(n)}(F) = E_G^{(1)}[E_G^{(n-1)}(F)];$ $E_G^{(0)}(F) = F$	
Morphological reconstruction by dilation	$R_G^D(F) = D_G^{(k)}(F)$	k is such that $D_G^{(k)}(F) = D_G^{(k+1)}(F)$
Morphological reconstruction by erosion	$R_G^E(F) = E_G^{(k)}(F)$	k is such that $E_G^{(k)}(F) = E_G^{(k+1)}(F)$
Opening by reconstruction	$O_R^{(n)}(F) = R_F^D[(F \ominus nB)]$	$(F \ominus nB)$ indicates n erosions of F by B .
Closing by reconstruction	$C_R^{(n)}(F) = R_F^E[(F \oplus nB)]$	$(F \oplus nB)$ indicates n dilations of F by B .
Hole filling	$H = [R_F^D(F)]^c$	H is equal to the input image I , but with all holes filled. See Eq. (9.5-28) for the definition of the marker image F .
Border clearing	$X = I - R_I^D(F)$	X is equal to the input image I , but with all objects that touch (are connected to) the boundary removed. See Eq. (9.5-30) for the definition of the marker image F .