10 Image Segmentation

The whole is equal to the sum of its parts.

Euclid

The whole is greater than the sum of its parts.

Max Wertheimer

Preview

The material in the previous chapter began a transition from image processing methods whose inputs and outputs are images, to methods in which the inputs are images but the outputs are attributes extracted from those images (in the sense defined in Section 1.1). Segmentation is another major step in that direction.

Segmentation subdivides an image into its constituent regions or objects. The level of detail to which the subdivision is carried depends on the problem being solved. That is, segmentation should stop when the objects or regions of interest in an application have been detected. For example, in the automated inspection of electronic assemblies, interest lies in analyzing images of products with the objective of determining the presence or absence of specific anomalies, such as missing components or broken connection paths. There is no point in carrying segmentation past the level of detail required to identify those elements.

Segmentation of nontrivial images is one of the most difficult tasks in image processing. Segmentation accuracy determines the eventual success or failure of computerized analysis procedures. For this reason, considerable care should be taken to improve the probability of accurate segmentation. In some situations, such as in industrial inspection applications, at least some measure of control over the environment typically is possible. The experienced image processing system designer invariably pays considerable attention to such opportunities. In other applications, such as autonomous target acquisition, the system designer has no control over the operating environment, and the usual

approach is to focus on selecting the types of sensors most likely to enhance the objects of interest while diminishing the contribution of irrelevant image detail. A good example is the use of infrared imaging by the military to detect objects with strong heat signatures, such as equipment and troops in motion.

See Sections 6.7 and 10.3.8 for a discussion of segmentation techniques based on more than just gray (intensity) values.

Most of the segmentation algorithms in this chapter are based on one of two basic properties of intensity values: discontinuity and similarity. In the first category, the approach is to partition an image based on abrupt changes in intensity, such as edges. The principal approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria. Thresholding, region growing, and region splitting and merging are examples of methods in this category. In this chapter, we discuss and illustrate a number of these approaches and show that improvements in segmentation performance can be achieved by combining methods from distinct categories, such as techniques in which edge detection is combined with thresholding. We discuss also image segmentation based on morphology. This approach is particularly attractive because it combines several of the positive attributes of segmentation based on the techniques presented in the first part of the chapter. We conclude the chapter with a brief discussion on the use of motion cues for segmentation.

10.1 Fundamentals

Let R represent the entire spatial region occupied by an image. We may view image segmentation as a process that partitions R into n subregions, $R_1, R_2, \ldots, R_n$, such that

(a) $\bigcup_{i=1}^{n} R_i = R$.

See Section 2.5.2 regarding connected sets.

(b) R_i is a connected set, $i = 1, 2, \ldots, n$.

(c) $R_i \cap R_j = \varnothing$ for all i and j, $i \neq j$.

(d) $Q(R_i) = \text{TRUE}$ for $i = 1, 2, \ldots, n$.

(e) $Q(R_i \cup R_j) = \text{FALSE}$ for any adjacent regions R_i and R_j.

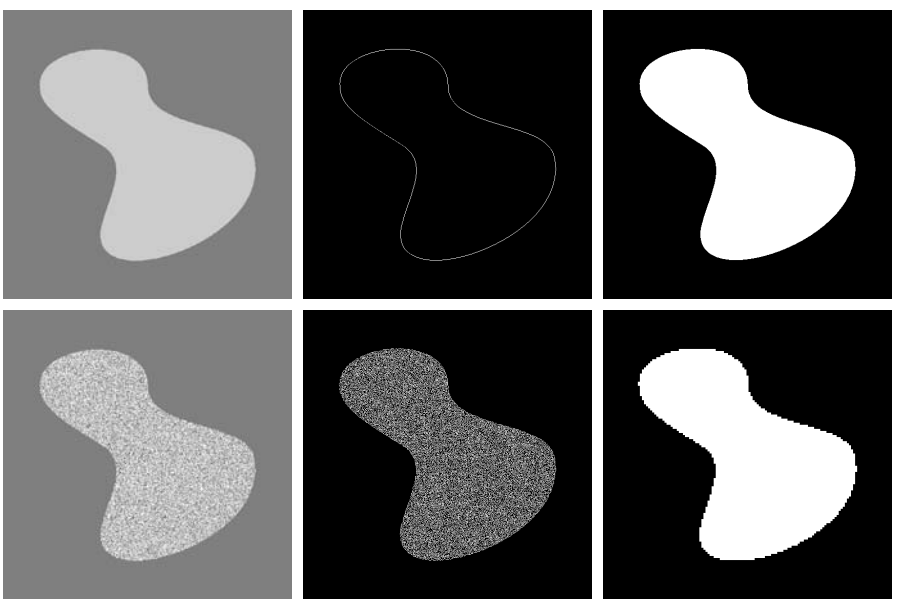
Here, $Q(R_k)$ is a logical predicate defined over the points in set R_k, and $\varnothing$ is the null set. The symbols $\cup$ and $\cap$ represent set union and intersection, respectively, as defined in Section 2.6.4. Two regions R_i and R_j are said to be *adjacent* if their union forms a connected set, as discussed in Section 2.5.2.

Condition (a) indicates that the segmentation must be complete; that is, every pixel must be in a region. Condition (b) requires that points in a region be connected in some predefined sense (e.g., the points must be 4- or 8-connected, as defined in Section 2.5.2). Condition (c) indicates that the regions must be disjoint. Condition (d) deals with the properties that must be satisfied by the pixels in a segmented region—for example, $Q(R_i) = \text{TRUE}$ if all pixels in R_i have the same intensity level. Finally, condition (e) indicates that two adjacent regions R_i and R_j must be different in the sense of predicate Q.[†]

[†]In general, Q can be a compound expression such as, for example, $Q(R_i) = \text{TRUE}$ if the average intensity of the pixels in R_i is less than m_i, AND if the standard deviation of their intensity is greater than σ_i, where m_i and σ_i are specified constants.

Thus, we see that the fundamental problem in segmentation is to partition an image into regions that satisfy the preceding conditions. Segmentation algorithms for monochrome images generally are based on one of two basic categories dealing with properties of intensity values: discontinuity and similarity. In the first category, the assumption is that boundaries of regions are sufficiently different from each other and from the background to allow boundary detection based on local discontinuities in intensity. *Edge-based segmentation* is the principal approach used in this category. *Region-based segmentation* approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria.

Figure 10.1 illustrates the preceding concepts. Figure 10.1(a) shows an image of a region of constant intensity superimposed on a darker background, also of constant intensity. These two regions comprise the overall image region. Figure 10.1(b) shows the result of computing the boundary of the inner region based on intensity discontinuities. Points on the inside and outside of the boundary are black (zero) because there are no discontinuities in intensity in those regions. To segment the image, we assign one level (say, white) to the pixels on, or interior to, the boundary and another level (say, black) to all points exterior to the boundary. Figure 10.1(c) shows the result of such a procedure. We see that conditions (a) through (c) stated at the beginning of this section are satisfied by



a b c
d e f

FIGURE 10.1 (a) Image containing a region of constant intensity. (b) Image showing the boundary of the inner region, obtained from intensity discontinuities. (c) Result of segmenting the image into two regions. (d) Image containing a textured region. (e) Result of edge computations. Note the large number of small edges that are connected to the original boundary, making it difficult to find a unique boundary using only edge information. (f) Result of segmentation based on region properties.

this result. The predicate of condition (d) is: If a pixel is on, or inside the boundary, label it white; otherwise label it black. We see that this predicate is TRUE for the points labeled black and white in Fig. 10.1(c). Similarly, the two segmented regions (object and background) satisfy condition (e).

The next three images illustrate region-based segmentation. Figure 10.1(d) is similar to Fig. 10.1(a), but the intensities of the inner region form a textured pattern. Figure 10.1(e) shows the result of computing the edges of this image. Clearly, the numerous spurious changes in intensity make it difficult to identify a unique boundary for the original image because many of the nonzero intensity changes are connected to the boundary, so edge-based segmentation is not a suitable approach. We note however, that the outer region is constant, so all we need to solve this simple segmentation problem is a predicate that differentiates between textured and constant regions. The standard deviation of pixel values is a measure that accomplishes this, because it is nonzero in areas of the texture region and zero otherwise. Figure 10.1(f) shows the result of dividing the original image into subregions of size 4×4. Each subregion was then labeled white if the standard deviation of its pixels was positive (i.e., if the predicate was TRUE) and zero otherwise. The result has a "blocky" appearance around the edge of the region because groups of 4×4 squares were labeled with the same intensity. Finally, note that these results also satisfy the five conditions stated at the beginning of this section.

10.2 Point, Line, and Edge Detection

The focus of this section is on segmentation methods that are based on detecting sharp, *local* changes in intensity. The three types of image features in which we are interested are isolated points, lines, and edges. *Edge pixels* are pixels at which the intensity of an image function changes abruptly, and *edges* (or *edge segments*) are sets of connected edge pixels (see Section 2.5.2 regarding connectivity). *Edge detectors* are local image processing methods designed to detect edge pixels. A line may be viewed as an edge segment in which the intensity of the background on either side of the line is either much higher or much lower than the intensity of the line pixels. In fact, as we discuss in the following section and in Section 10.2.4, lines give rise to so-called "roof edges." Similarly, an isolated point may be viewed as a line whose length and width are equal to one pixel.

When we refer to lines, we are referring to thin structures, typically just a few pixels thick. Such lines may correspond, for example, to elements of a digitized architectural drawing or roads in a satellite image.

10.2.1 Background

As we saw in Sections 2.6.3 and 3.5, local averaging smooths an image. Given that averaging is analogous to integration, it should come as no surprise that abrupt, local changes in intensity can be detected using derivatives. For reasons that will become evident shortly, first- and second-order derivatives are particularly well suited for this purpose.

Derivatives of a digital function are defined in terms of differences. There are various ways to approximate these differences but, as explained in Section 3.6.1, we require that any approximation used for a first derivative (1) must be zero in areas of constant intensity; (2) must be nonzero at the onset of an intensity step or ramp; and (3) must be nonzero at points along an intensity

ramp. Similarly, we require that an approximation used for a second derivative (1) must be zero in areas of constant intensity; (2) must be nonzero at the onset *and* end of an intensity step or ramp; and (3) must be zero along intensity ramps. Because we are dealing with digital quantities whose values are finite, the maximum possible intensity change is also finite, and the shortest distance over which a change can occur is between adjacent pixels.

We obtain an approximation to the first-order derivative at point x of a one-dimensional function $f(x)$ by expanding the function $f(x + \Delta x)$ into a Taylor series about x, letting $\Delta x = 1$, and keeping only the linear terms (Problem 10.1). The result is the digital difference

Recall from Section 2.4.2 that increments between image samples are defined as unity for notational clarity, hence the use of $\Delta x = 1$ in the derivation of Eq. (10.2-1).

$$\frac{\partial f}{\partial x} = f'(x) = f(x + 1) - f(x) \tag{10.2-1}$$

We used a partial derivative here for consistency in notation when we consider an image function of two variables, $f(x, y)$, at which time we will be dealing with partial derivatives along the two spatial axes. Clearly, $\partial f/\partial x = df/dx$ when f is a function of only one variable.

We obtain an expression for the second derivative by differentiating Eq. (10.2-1) with respect to x:

$$\begin{aligned}\frac{\partial^2 f}{\partial x^2} &= \frac{\partial f'(x)}{\partial x} = f'(x + 1) - f'(x) \\ &= f(x + 2) - f(x + 1) - f(x + 1) + f(x) \\ &= f(x + 2) - 2f(x + 1) + f(x)\end{aligned}$$

where the second line follows from Eq. (10.2-1). This expansion is about point $x + 1$. Our interest is on the second derivative about point x, so we subtract 1 from the arguments in the preceding expression and obtain the result

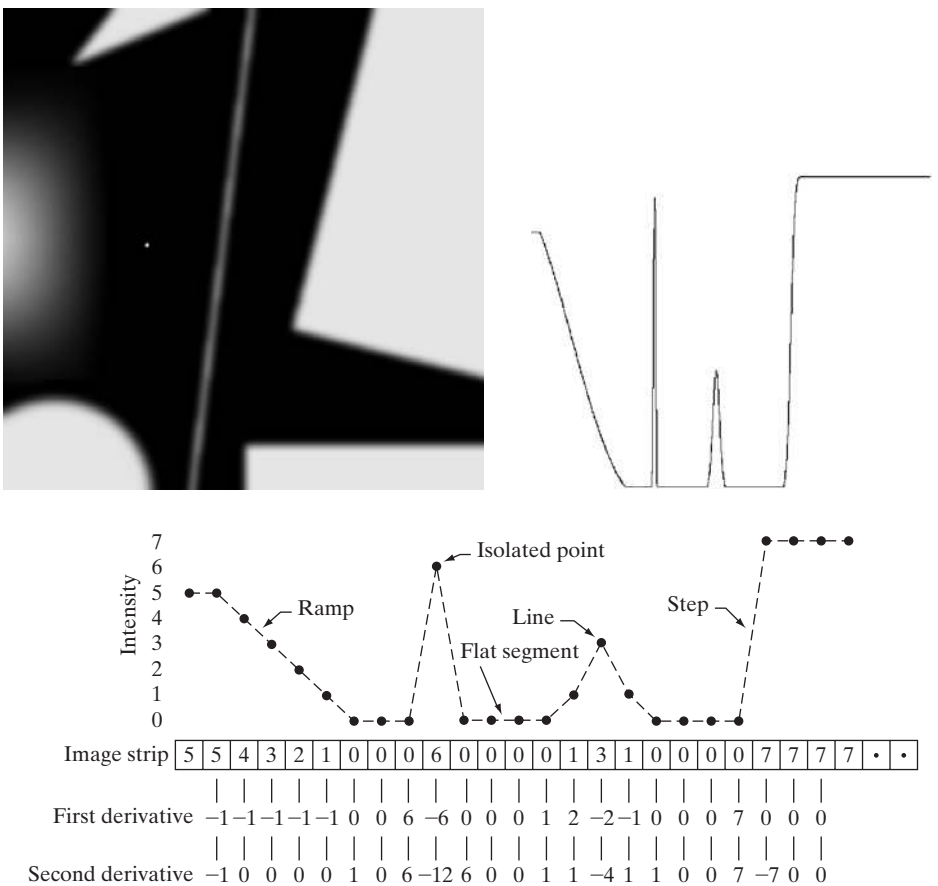
$$\frac{\partial^2 f}{\partial x^2} = f''(x) = f(x + 1) + f(x - 1) - 2f(x) \tag{10.2-2}$$

It easily is verified that Eqs. (10.2-1) and (10.2-2) satisfy the conditions stated at the beginning of this section regarding derivatives of the first and second order. To illustrate this, and also to highlight the fundamental similarities and differences between first- and second-order derivatives in the context of image processing, consider Fig. 10.2.

Figure 10.2(a) shows an image that contains various solid objects, a line, and a single noise point. Figure 10.2(b) shows a horizontal intensity profile (scan line) of the image approximately through its center, including the isolated point. Transitions in intensity between the solid objects and the background along the scan line show two types of edges: *ramp edges* (on the left) and *step edges* (on the right). As we discuss later, intensity transitions involving thin objects such as lines often are referred to as *roof edges*. Figure 10.2(c) shows a simplification of the profile, with just enough points to make it possible for us to analyze numerically how the first- and second-order derivatives behave as they encounter a noise point, a line, and the edges of objects. In this simplified diagram the

transition in the ramp spans four pixels, the noise point is a single pixel, the line is three pixels thick, and the transition of the intensity step takes place between adjacent pixels. The number of intensity levels was limited to eight for simplicity.

Consider the properties of the first and second derivatives as we traverse the profile from left to right. Initially, we note that the first-order derivative is nonzero at the onset and along the entire intensity ramp, while the second-order derivative is nonzero only at the onset and end of the ramp. Because edges of digital images resemble this type of transition, we conclude that first-order derivatives produce "thick" edges and second-order derivatives much finer ones. Next we encounter the isolated noise point. Here, the magnitude of the response at the point is much stronger for the second- than for the first-order derivative. This is not unexpected, because a second-order derivative is much



a b
c

FIGURE 10.2 (a) Image. (b) Horizontal intensity profile through the center of the image, including the isolated noise point. (c) Simplified profile (the points are joined by dashes for clarity). The image strip corresponds to the intensity profile, and the numbers in the boxes are the intensity values of the dots shown in the profile. The derivatives were obtained using Eqs. (10.2-1) and (10.2-2).

more aggressive than a first-order derivative in enhancing sharp changes. Thus, we can expect second-order derivatives to enhance fine detail (including noise) much more than first-order derivatives. The line in this example is rather thin, so it too is fine detail, and we see again that the second derivative has a larger magnitude. Finally, note in both the ramp and step edges that the second derivative has opposite signs (negative to positive or positive to negative) as it transitions into and out of an edge. This "double-edge" effect is an important characteristic that, as we show in Section 10.2.6, can be used to locate edges. The sign of the second derivative is used also to determine whether an edge is a transition from light to dark (negative second derivative) or from dark to light (positive second derivative), where the sign is observed as we move *into* the edge.

In summary, we arrive at the following conclusions: (1) First-order derivatives generally produce thicker edges in an image. (2) Second-order derivatives have a stronger response to fine detail, such as thin lines, isolated points, and noise. (3) Second-order derivatives produce a double-edge response at ramp and step transitions in intensity. (4) The sign of the second derivative can be used to determine whether a transition into an edge is from light to dark or dark to light.

The approach of choice for computing first and second derivatives at every pixel location in an image is to use spatial filters. For the 3×3 filter mask in Fig. 10.3, the procedure is to compute the sum of products of the mask coefficients with the intensity values in the region encompassed by the mask. That is, with reference to Eq. (3.4.3), the *response* of the mask at the center point of the region is

$$\begin{aligned} R &= w_1 z_1 + w_2 z_2 + \cdots + w_9 z_9 \\ &= \sum_{k=1}^{9} w_k z_k \end{aligned} \tag{10.2-3}$$

where z_k is the intensity of the pixel whose spatial location corresponds to the location of the kth coefficient in the mask. The details of implementing this operation over all pixels in an image are discussed in detail in Sections 3.4 and 3.6. In other words, computation of derivatives based on spatial masks is spatial filtering of an image with those masks, as explained in those sections.[†]

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

FIGURE 10.3 A general 3×3 spatial filter mask.

[†]As explained in Section 3.4.3, Eq. (10.2-3) is simplified notation either for spatial correlation, given by Eq. (3.4-1), or spatial convolution, given by Eq. (3.4-2). Therefore, when R is evaluated at all locations in an image, the result is an array. All spatial filtering in this chapter is done using correlation. In some instances, we use the term *convolving a mask with an image* as a matter of convention. However, we use this terminology only when the filter masks are symmetric, in which case correlation and convolution yield the same result.

10.2.2 Detection of Isolated Points

Based on the conclusions reached in the preceding section, we know that point detection should be based on the second derivative. From the discussion in Section 3.6.2, this implies using the Laplacian:

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{10.2-4}$$

where the partials are obtained using Eq. (10.2-2):

$$\frac{\partial^2 f(x, y)}{\partial x^2} = f(x + 1, y) + f(x - 1, y) - 2f(x, y) \tag{10.2-5}$$

and

$$\frac{\partial^2 f(x, y)}{\partial y^2} = f(x, y + 1) + f(x, y - 1) - 2f(x, y) \tag{10.2-6}$$

The Laplacian is then

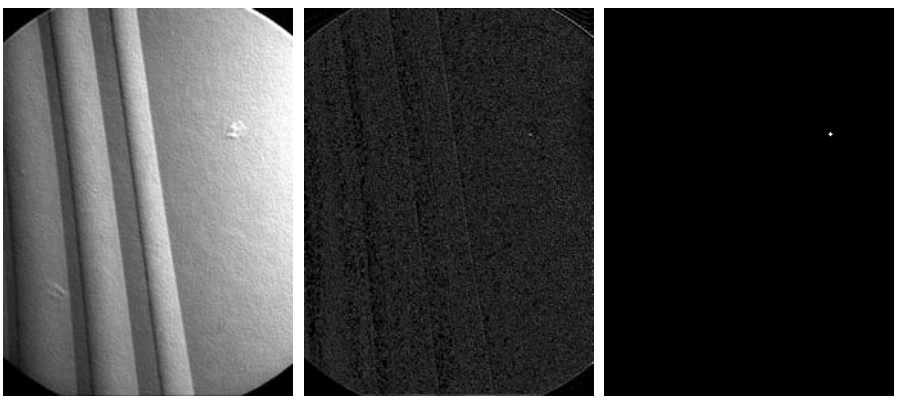
$$\begin{aligned} \nabla^2 f(x, y) = {} & f(x + 1, y) + f(x - 1, y) + f(x, y + 1) \\ & + f(x, y - 1) - 4f(x, y) \end{aligned} \tag{10.2-7}$$

As explained in Section 3.6.2, this expression can be implemented using the mask in Fig. 3.37(a). Also, as explained in that section, we can extend Eq. (10.2-7) to include the diagonal terms, and use the mask in Fig. 3.37(b). Using the Laplacian mask in Fig. 10.4(a), which is the same as the mask in Fig. 3.37(b), we say that a point has been detected at the location (x, y) on which the mask is centered if the absolute value of the response of the mask at that point exceeds a specified threshold. Such points are labeled 1 in the output image and all others are labeled 0, thus producing a binary image. In other words, the output is obtained using the following expression:

$$g(x, y) = \begin{cases} 1 & \text{if } |R(x, y)| \geq T \\ 0 & \text{otherwise} \end{cases} \tag{10.2-8}$$

where g is the output image, T is a nonnegative threshold, and R is given by Eq. (10.2-3). This formulation simply measures the weighted differences between a pixel and its 8-neighbors. Intuitively, the idea is that the intensity of an isolated point will be quite different from its surroundings and thus will be easily detectable by this type of mask. The only differences in intensity that are considered of interest are those large enough (as determined by T) to be considered isolated points. Note that, as usual for a derivative mask, the coefficients sum to zero, indicating that the mask response will be zero in areas of constant intensity.

1	1	1
1	−8	1
1	1	1



a
b c d

FIGURE 10.4
(a) Point detection (Laplacian) mask. (b) X-ray image of turbine blade with a porosity. The porosity contains a single black pixel. (c) Result of convolving the mask with the image. (d) Result of using Eq. (10.2-8) showing a single point (the point was enlarged to make it easier to see). (Original image courtesy of X-TEK Systems, Ltd.)

EXAMPLE 10.1: Detection of isolated points in an image.

■ We illustrate segmentation of isolated points in an image with the aid of Fig. 10.4(b), which is an X-ray image of a turbine blade from a jet engine. The blade has a porosity in the upper-right quadrant of the image, and there is a single black pixel embedded within the porosity. Figure 10.4(c) is the result of applying the point detector mask to the X-ray image, and Fig. 10.4(d) shows the result of using Eq. (10.2-8) with T equal to 90% of the highest absolute pixel value of the image in Fig. 10.4(c). The single pixel is clearly visible in this image (the pixel was enlarged manually to enhance its visibility). This type of detection process is rather specialized, because it is based on abrupt intensity changes at single-pixel locations that are surrounded by a homogeneous background in the area of the detector mask. When this condition is not satisfied, other methods discussed in this chapter are more suitable for detecting intensity changes. ■

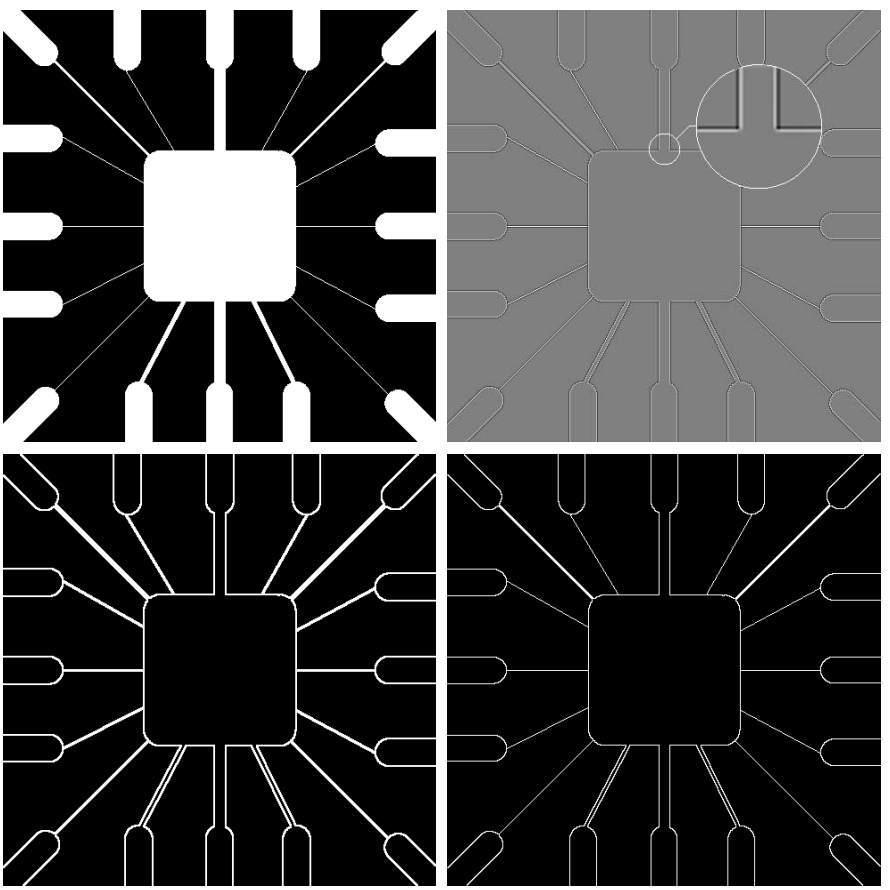
10.2.3 Line Detection

The next level of complexity is line detection. Based on the discussion in Section 10.2.1, we know that for line detection we can expect second derivatives to result in a stronger response and to produce thinner lines than first derivatives. Thus, we can use the Laplacian mask in Fig. 10.4(a) for line detection also, keeping in mind that the double-line effect of the second derivative must be handled properly. The following example illustrates the procedure.

EXAMPLE 10.2: Using the Laplacian for line detection.

■ Figure 10.5(a) shows a 486 × 486 (binary) portion of a wire-bond mask for an electronic circuit, and Fig. 10.5(b) shows its Laplacian image. Because the Laplacian image contains negative values,[†] scaling is necessary for display. As the magnified section shows, mid gray represents zero, darker shades of gray represent negative values, and lighter shades are positive. The double-line effect is clearly visible in the magnified region.

At first, it might appear that the negative values can be handled simply by taking the absolute value of the Laplacian image. However, as Fig. 10.5(c) shows, this approach doubles the thickness of the lines. A more suitable approach is to use only the positive values of the Laplacian (in noisy situations we use the values that exceed a positive threshold to eliminate random variations about zero caused by the noise). As the image in Fig. 10.5(d) shows, this approach results in thinner lines, which are considerably more useful. Note in Figs. 10.5(b) through (d) that when the lines are wide with respect to the size of the Laplacian mask, the lines are separated by a zero "valley."



a b
c d

FIGURE 10.5
(a) Original image. (b) Laplacian image; the magnified section shows the positive/negative double-line effect characteristic of the Laplacian. (c) Absolute value of the Laplacian. (d) Positive values of the Laplacian.

[†]When a mask whose coefficients sum to zero is convolved with an image, the pixels in the resulting image will sum to zero also (Problem 3.16), implying the existence of both positive and negative pixels in the result. Scaling so that all values are nonnegative is required for display purposes.

This is not unexpected. For example, when the 3×3 filter is centered on a line of constant intensity 5 pixels wide, the response will be zero, thus producing the effect just mentioned. When we talk about line detection, the assumption is that lines are thin with respect to the size of the detector. Lines that do not satisfy this assumption are best treated as regions and handled by the edge detection methods discussed later in this section. ■

The Laplacian detector in Fig. 10.4(a) is isotropic, so its response is independent of direction (with respect to the four directions of the 3×3 Laplacian mask: vertical, horizontal, and two diagonals). Often, interest lies in detecting lines in *specified* directions. Consider the masks in Fig. 10.6. Suppose that an image with a constant background and containing various lines (oriented at 0°, ±45°, and 90°) is filtered with the first mask. The maximum responses would occur at image locations in which a horizontal line passed through the middle row of the mask. This is easily verified by sketching a simple array of 1s with a line of a different intensity (say, 5s) running horizontally through the array. A similar experiment would reveal that the second mask in Fig. 10.6 responds best to lines oriented at +45°; the third mask to vertical lines; and the fourth mask to lines in the −45° direction. The preferred direction of each mask is weighted with a larger coefficient (i.e., 2) than other possible directions. The coefficients in each mask sum to zero, indicating a zero response in areas of constant intensity.

Recall from Section 2.4.2 that the image axis convention has the origin at the top left, with the positive x-axis pointing down and the positive y-axis extending to the right. The angles of the lines discussed in this section are measured with respect to the positive x-axis. For example, a vertical line has an angle of 0°, and a +45° line extends downward and to the right.

Let R_1, R_2, R_3, and R_4 denote the responses of the masks in Fig. 10.6, from left to right, where the Rs are given by Eq. (10.2-3). Suppose that an image is filtered (individually) with the four masks. If, at a given point in the image, $|R_k| > |R_j|$, for all $j \neq k$, that point is said to be more likely associated with a line in the direction of mask k. For example, if at a point in the image, $|R_1| > |R_j|$ for $j = 2, 3, 4$, that particular point is said to be more likely associated with a horizontal line. Alternatively, we may be interested in detecting lines in a specified direction. In this case, we would use the mask associated with that direction and threshold its output, as in Eq. (10.2-8). In other words, if we are interested in detecting all the lines in an image in the direction defined by a given mask, we simply run the mask through the image and threshold the absolute value of the result. The points that are left are the strongest responses which, for lines 1 pixel thick, correspond closest to the direction defined by the mask. The following example illustrates this procedure.

Do not confuse our use of R to designate mask response with the same symbol to denote regions in Section 10.1.

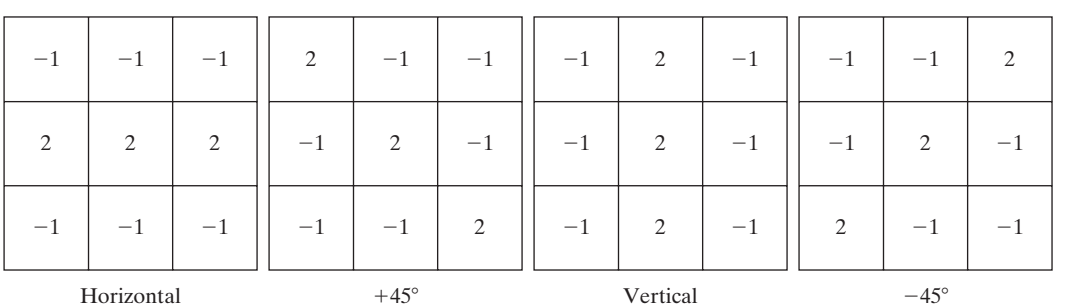


FIGURE 10.6 Line detection masks. Angles are with respect to the axis system in Fig. 2.18(b).

EXAMPLE 10.3: Detection of lines in specified directions.

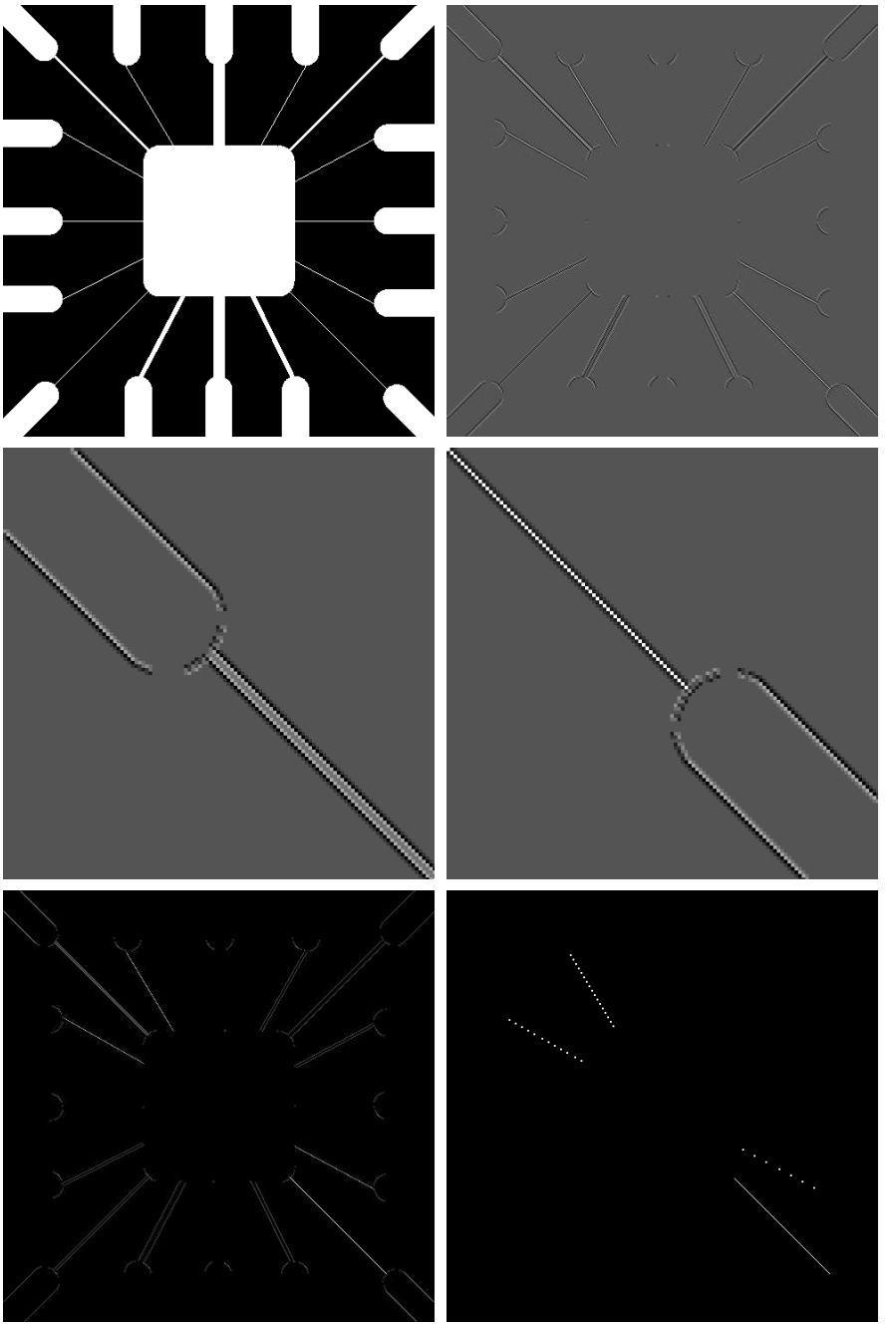
■ Figure 10.7(a) shows the image used in the previous example. Suppose that we are interested in finding all the lines that are 1 pixel thick and oriented at +45°. For this purpose, we use the second mask in Fig. 10.6. Figure 10.7(b) is the result of filtering the image with that mask. As before, the shades darker than the gray background in Fig. 10.7(b) correspond to negative values. There are two principal segments in the image oriented in the +45° direction, one at the top left and one at the bottom right. Figures 10.7(c) and (d) show zoomed sections of Fig. 10.7(b) corresponding to these two areas. Note how much brighter the straight line segment in Fig. 10.7(d) is than the segment in Fig. 10.7(c). The reason is that the line segment in the bottom right of Fig. 10.7(a) is 1 pixel thick, while the one at the top left is not. The mask is "tuned" to detect 1-pixel-thick lines in the +45° direction, so we expect its response to be stronger when such lines are detected. Figure 10.7(e) shows the positive values of Fig. 10.7(b). Because we are interested in the strongest response, we let T equal the maximum value in Fig. 10.7(e). Figure 10.7(f) shows in white the points whose values satisfied the condition $g \geq T$, where g is the image in Fig. 10.7(e). The isolated points in the figure are points that also had similarly strong responses to the mask. In the original image, these points and their immediate neighbors are oriented in such a way that the mask produced a maximum response at those locations. These isolated points can be detected using the mask in Fig. 10.4(a) and then deleted, or they can be deleted using morphological operators, as discussed in the last chapter. ■

10.2.4 Edge Models

Edge detection is the approach used most frequently for segmenting images based on abrupt (local) changes in intensity. We begin by introducing several ways to model edges and then discuss a number of approaches for edge detection.

Edge models are classified according to their intensity profiles. A *step edge* involves a transition between two intensity levels occurring ideally over the distance of 1 pixel. Figure 10.8(a) shows a section of a vertical step edge and a horizontal intensity profile through the edge. Step edges occur, for example, in images generated by a computer for use in areas such as solid modeling and animation. These clean, *ideal* edges can occur over the distance of 1 pixel, provided that no additional processing (such as smoothing) is used to make them look "real." Digital step edges are used frequently as edge models in algorithm development. For example, the Canny edge detection algorithm discussed in Section 10.2.6 was derived using a step-edge model.

In practice, digital images have edges that are blurred and noisy, with the degree of blurring determined principally by limitations in the focusing mechanism (e.g., lenses in the case of optical images), and the noise level determined principally by the electronic components of the imaging system. In such situations, edges are more closely modeled as having an intensity *ramp* profile, such as the edge in Fig. 10.8(b). The slope of the ramp is inversely proportional to the degree of blurring in the edge. In this model, we no longer have a thin (1 pixel thick) path. Instead, an edge point now is any point contained in the ramp, and an edge segment would then be a set of such points that are connected.



a b
c d
e f

FIGURE 10.7
(a) Image of a wire-bond template. (b) Result of processing with the +45° line detector mask in Fig. 10.6. (c) Zoomed view of the top left region of (b). (d) Zoomed view of the bottom right region of (b). (e) The image in (b) with all negative values set to zero. (f) All points (in white) whose values satisfied the condition $g \geq T$, where g is the image in (e). (The points in (f) were enlarged to make them easier to see.)

A third model of an edge is the so-called *roof edge*, having the characteristics illustrated in Fig. 10.8(c). Roof edges are models of lines through a region, with the base (width) of a roof edge being determined by the thickness and sharpness of the line. In the limit, when its base is 1 pixel wide, a roof edge is

a b c

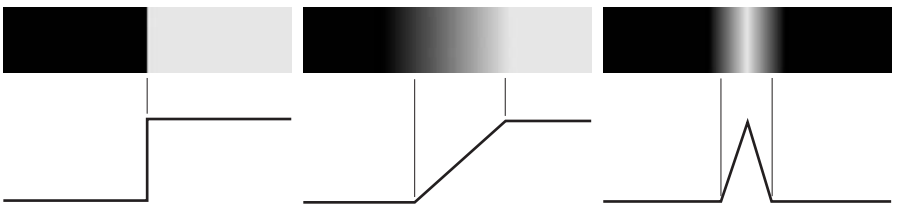


FIGURE 10.8
From left to right, models (ideal representations) of a step, a ramp, and a roof edge, and their corresponding intensity profiles.

really nothing more than a 1-pixel-thick line running through a region in an image. Roof edges arise, for example, in range imaging, when thin objects (such as pipes) are closer to the sensor than their equidistant background (such as walls). The pipes appear brighter and thus create an image similar to the model in Fig. 10.8(c). As mentioned earlier, other areas in which roof edges appear routinely are in the digitization of line drawings and also in satellite images, where thin features, such as roads, can be modeled by this type of edge.

It is not unusual to find images that contain all three types of edges. Although blurring and noise result in deviations from the ideal shapes, edges in images that are reasonably sharp and have a moderate amount of noise do *resemble* the characteristics of the edge models in Fig. 10.8, as the profiles in Fig. 10.9 illustrate.[†] What the models in Fig. 10.8 allow us to do is write mathematical expressions for edges in the development of image processing algorithms. The performance of these algorithms will depend on the differences between actual edges and the models used in developing the algorithms.

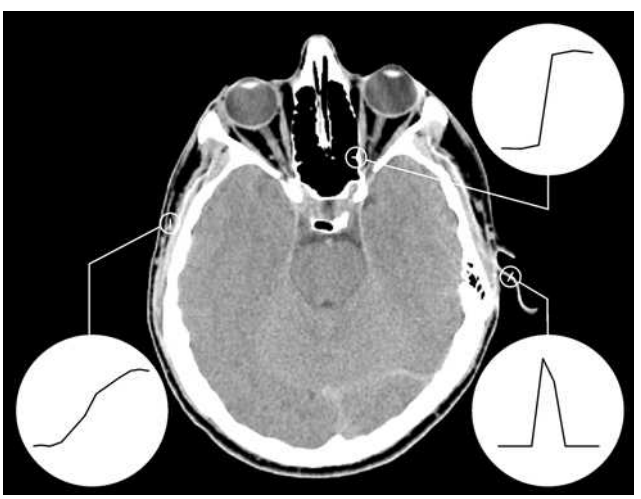
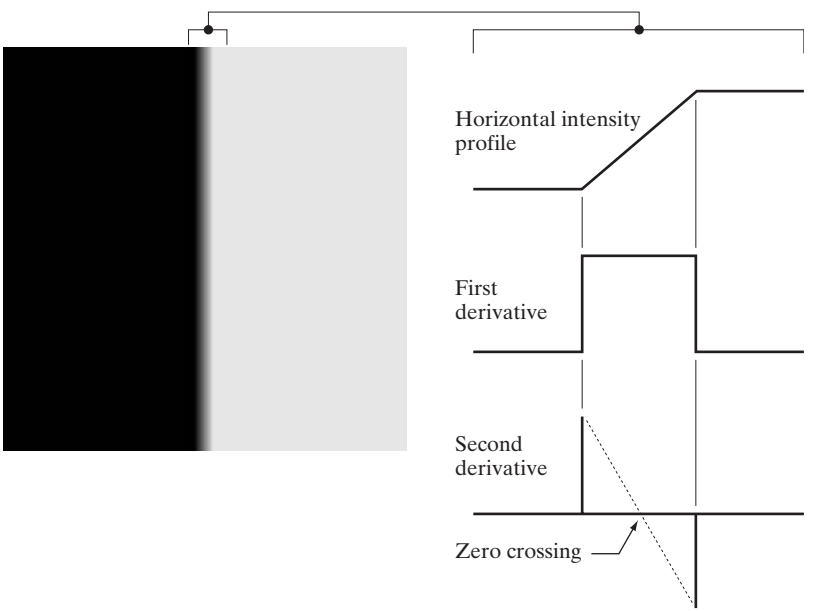


FIGURE 10.9 A 1508 × 1970 image showing (zoomed) actual ramp (bottom, left), step (top, right), and roof edge profiles. The profiles are from dark to light, in the areas indicated by the short line segments shown in the small circles. The ramp and "step" profiles span 9 pixels and 2 pixels, respectively. The base of the roof edge is 3 pixels. (Original image courtesy of Dr. David R. Pickens, Vanderbilt University.)

[†]Ramp edges with a sharp slope of a few pixels often are treated as step edges in order to differentiate them from ramps in the same image whose slopes are more gradual.

Figure 10.10(a) shows the image from which the segment in Fig. 10.8(b) was extracted. Figure 10.10(b) shows a horizontal intensity profile. This figure shows also the first and second derivatives of the intensity profile. As in the discussion in Section 10.2.1, moving from left to right along the intensity profile, we note that the first derivative is positive at the onset of the ramp and at points on the ramp, and it is zero in areas of constant intensity. The second derivative is positive at the beginning of the ramp, negative at the end of the ramp, zero at points on the ramp, and zero at points of constant intensity. The signs of the derivatives just discussed would be reversed for an edge that transitions from light to dark. The intersection between the zero intensity axis and a line extending between the extrema of the second derivative marks a point called the *zero crossing* of the second derivative.

We conclude from these observations that the *magnitude* of the first derivative can be used to detect the presence of an edge at a point in an image. Similarly, the *sign* of the second derivative can be used to determine whether an edge pixel lies on the dark or light side of an edge. We note two additional properties of the second derivative around an edge: (1) it produces two values for every edge in an image (an undesirable feature); and (2) its zero crossings can be used for locating the centers of thick edges, as we show later in this section. Some edge models make use of a smooth transition into and out of the ramp (Problem 10.7). However, the conclusions reached using those models are the same as with an ideal ramp, and working with the latter simplifies theoretical formulations. Finally, although attention thus far has been limited to a 1-D horizontal profile, a similar argument applies to an edge of any orientation in an image. We simply define a profile perpendicular to the edge direction at any desired point and interpret the results in the same manner as for the vertical edge just discussed.



a b

FIGURE 10.10
(a) Two regions of constant intensity separated by an ideal vertical ramp edge. (b) Detail near the edge, showing a horizontal intensity profile, together with its first and second derivatives.

EXAMPLE 10.4: Behavior of the first and second derivatives of a noisy edge.

■ The edges in Fig. 10.8 are free of noise. The image segments in the first column in Fig. 10.11 show close-ups of four ramp edges that transition from a black region on the left to a white region on the right (keep in mind that the entire transition from black to white is a single edge). The image segment at the top left is free of noise. The other three images in the first column are corrupted by additive Gaussian noise with zero mean and standard deviation of 0.1, 1.0, and 10.0 intensity levels, respectively. The graph below each image is a horizontal intensity profile passing through the center of the image. All images have 8 bits of intensity resolution, with 0 and 255 representing black and white, respectively.

Computation of the derivatives for the entire image segment is discussed in the following section. For now, our interest lies on analyzing just the intensity profiles.

Consider the image at the top of the center column. As discussed in connection with Fig. 10.10(b), the derivative of the scan line on the left is zero in the constant areas. These are the two black bands shown in the derivative image. The derivatives at points on the ramp are constant and equal to the slope of the ramp. These constant values in the derivative image are shown in gray. As we move down the center column, the derivatives become increasingly different from the noiseless case. In fact, it would be difficult to associate the last profile in the center column with the first derivative of a ramp edge. What makes these results interesting is that the noise is almost invisible in the images on the left column. These examples are good illustrations of the sensitivity of derivatives to noise.

As expected, the second derivative is even more sensitive to noise. The second derivative of the noiseless image is shown at the top of the right column. The thin white and black vertical lines are the positive and negative components of the second derivative, as explained in Fig. 10.10. The gray in these images represents zero (as discussed earlier, scaling causes zero to show as gray). The only noisy second derivative image that barely resembles the noiseless case is the one corresponding to noise with a standard deviation of 0.1. The remaining second-derivative images and profiles clearly illustrate that it would be difficult indeed to detect their positive and negative components, which are the truly useful features of the second derivative in terms of edge detection.

The fact that such little visual noise can have such a significant impact on the two key derivatives used for detecting edges is an important issue to keep in mind. In particular, image smoothing should be a serious consideration prior to the use of derivatives in applications where noise with levels similar to those we have just discussed is likely to be present. ■

We conclude this section by noting that there are three fundamental steps performed in edge detection:

1. *Image smoothing for noise reduction.* The need for this step is amply illustrated by the results in the second and third columns of Fig. 10.11.
2. *Detection of edge points.* As mentioned earlier, this is a local operation that extracts from an image all points that are potential candidates to become edge points.
3. *Edge localization.* The objective of this step is to select from the candidate edge points only the points that are true members of the set of points comprising an edge.

The remainder of this section deals with techniques for achieving these objectives.

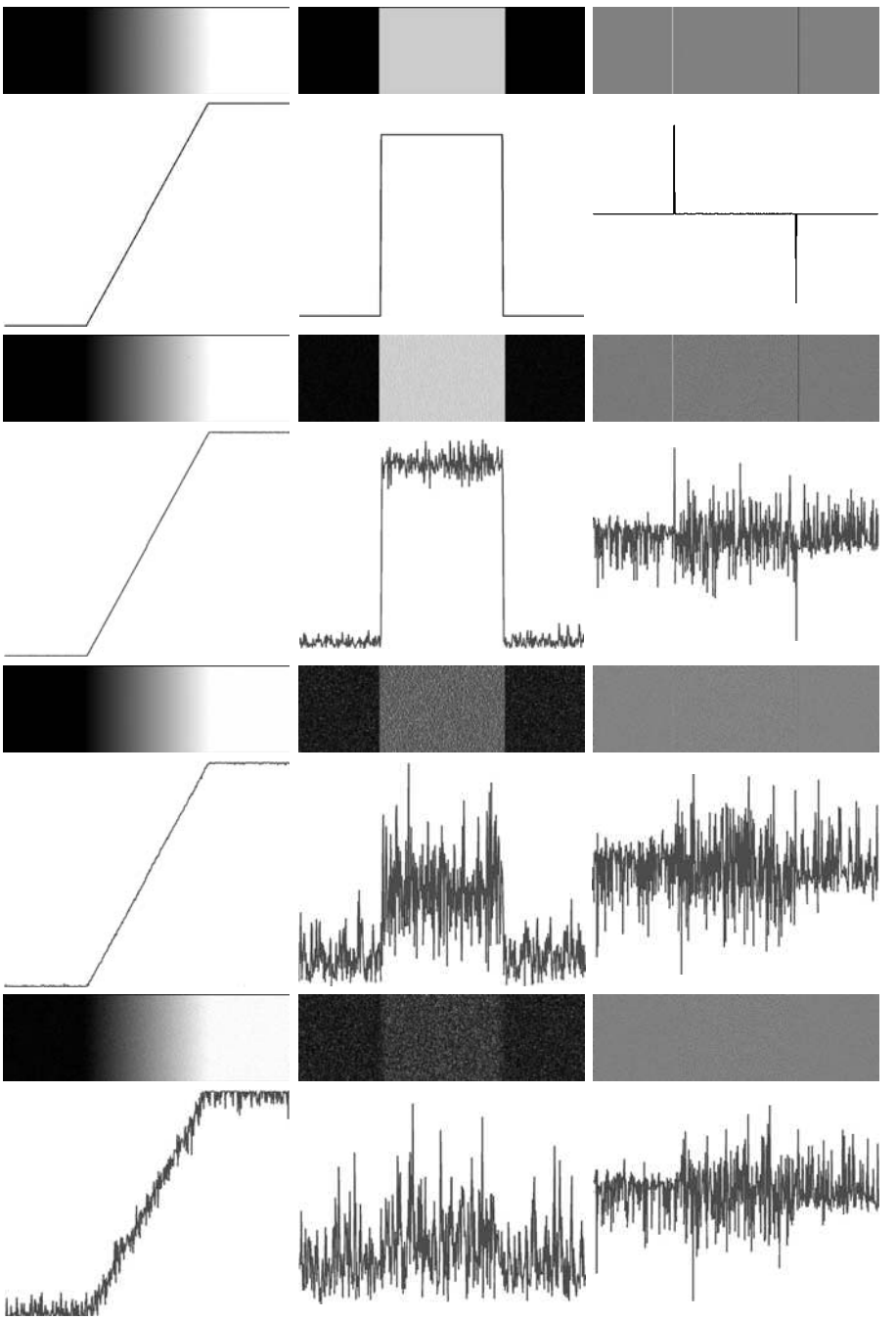


FIGURE 10.11 First column: Images and intensity profiles of a ramp edge corrupted by random Gaussian noise of zero mean and standard deviations of 0.0, 0.1, 1.0, and 10.0 intensity levels, respectively. Second column: First-derivative images and intensity profiles. Third column: Second-derivative images and intensity profiles.

10.2.5 Basic Edge Detection

As illustrated in the previous section, detecting changes in intensity for the purpose of finding edges can be accomplished using first- or second-order derivatives. We discuss first-order derivatives in this section and work with second-order derivatives in Section 10.2.6.

The image gradient and its properties

The tool of choice for finding edge strength *and* direction at location (x, y) of an image, f, is the gradient, denoted by ∇f, and defined as the *vector*

For convenience, we repeat here some equations from Section 3.6.4.

$$\nabla f \equiv \text{grad}(f) \equiv \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix} \tag{10.2-9}$$

This vector has the important geometrical property that it points in the direction of the greatest rate of change of f at location (x, y).

The *magnitude* (*length*) of vector ∇f, denoted as $M(x, y)$, where

$$M(x, y) = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2} \tag{10.2-10}$$

is the *value* of the rate of change in the direction of the gradient vector. Note that g_x, g_y, and $M(x, y)$ are images of the same size as the original, created when x and y are allowed to vary over all pixel locations in f. It is common practice to refer to the latter image as the *gradient image*, or simply as the *gradient* when the meaning is clear. The summation, square, and square root operations are *array operations*, as defined in Section 2.6.1.

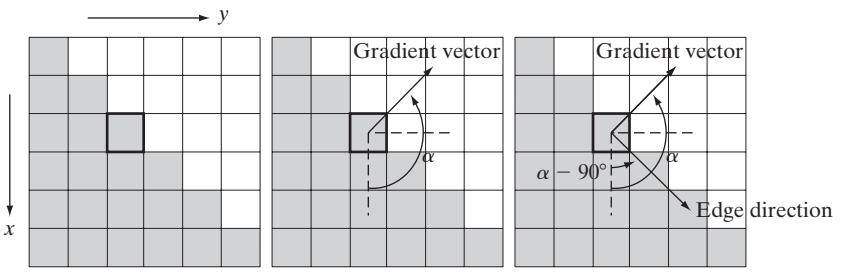
The *direction* of the gradient vector is given by the angle

$$\alpha(x, y) = \tan^{-1}\left[\frac{g_y}{g_x}\right] \tag{10.2-11}$$

measured with respect to the x-axis. As in the case of the gradient image, $\alpha(x, y)$ also is an image of the same size as the original created by the array division of image g_y by image g_x. The direction of an edge at an arbitrary point (x, y) is *orthogonal* to the direction, $\alpha(x, y)$, of the gradient vector at the point.

EXAMPLE 10.5: Properties of the gradient.

■ Figure 10.12(a) shows a zoomed section of an image containing a straight edge segment. Each square shown corresponds to a pixel, and we are interested in obtaining the strength and direction of the edge at the point highlighted with a box. The pixels in gray have value 0 and the pixels in white have value 1. We show following this example that an approach for computing the derivatives in the x- and y-directions using a 3×3 neighborhood centered about a point consists simply of subtracting the pixels in the top row of the neighborhood from the pixels in the bottom row to obtain the partial derivative in the x-direction. Similarly, we subtract the pixels in the left column from the pixels in the right column to obtain the partial derivative in the y-direction. It then



a b c

FIGURE 10.12 Using the gradient to determine edge strength and direction at a point. Note that the edge is perpendicular to the direction of the gradient vector at the point where the gradient is computed. Each square in the figure represents one pixel.

follows, using these differences as our estimates of the partials, that $\partial f/\partial x = -2$ and $\partial f/\partial y = 2$ at the point in question. Then,

Recall from Section 2.4.2 that the origin of the image coordinate system is at the top left, with the positive x- and y-axes extending down and to the right, respectively.

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \end{bmatrix}$$

from which we obtain $M(x, y) = 2\sqrt{2}$ at that point. Similarly, the direction of the gradient vector at the same point follows from Eq. (10.2-11): $\alpha(x, y) = \tan^{-1}(g_y/g_x) = -45°$, which is the same as 135° measured in the positive direction with respect to the x-axis. Figure 10.12(b) shows the gradient vector and its direction angle.

Figure 10.12(c) illustrates the important fact mentioned earlier that the edge at a point is orthogonal to the gradient vector at that point. So the direction angle of the edge in this example is $\alpha - 90° = 45°$. All edge points in Fig. 10.12(a) have the same gradient, so the entire edge segment is in the same direction. The gradient vector sometimes is called the *edge normal*. When the vector is normalized to unit length by dividing it by its magnitude [Eq. (10.2-10)], the resulting vector is commonly referred to as the *edge unit normal.* ■

Gradient operators

Obtaining the gradient of an image requires computing the partial derivatives $\partial f/\partial x$ and $\partial f/\partial y$ at every pixel location in the image. We are dealing with digital quantities, so a digital approximation of the partial derivatives over a neighborhood about a point is required. From Section 10.2.1 we know that

$$g_x = \frac{\partial f(x, y)}{\partial x} = f(x + 1, y) - f(x, y) \tag{10.2-12}$$

and

$$g_y = \frac{\partial f(x, y)}{\partial y} = f(x, y + 1) - f(x, y) \tag{10.2-13}$$

a b

FIGURE 10.13 One-dimensional masks used to implement Eqs. (10.2-12) and (10.2-13).

−1
1

−1	1

These two equations can be implemented for all pertinent values of x and y by filtering $f(x, y)$ with the 1-D masks in Fig. 10.13.

When diagonal edge direction is of interest, we need a 2-D mask. The *Roberts cross-gradient operators* (Roberts [1965]) are one of the earliest attempts to use 2-D masks with a diagonal preference. Consider the 3×3 region in Fig. 10.14(a). The Roberts operators are based on implementing the diagonal differences

$$g_x = \frac{\partial f}{\partial x} = (z_9 - z_5) \tag{10.2-14}$$

In the remainder of this section we assume implicitly that f is a function of two variables, and omit the variables to simplify the notation.

and

$$g_y = \frac{\partial f}{\partial y} = (z_8 - z_6) \tag{10.2-15}$$

a
b c
d e
f g

FIGURE 10.14 A 3×3 region of an image (the z's are intensity values) and various masks used to compute the gradient at the point labeled z_5.

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

−1	0	0	−1
0	1	1	0

Roberts

−1	−1	−1	−1	0	1
0	0	0	−1	0	1
1	1	1	−1	0	1

Prewitt

−1	−2	−1	−1	0	1
0	0	0	−2	0	2
1	2	1	−1	0	1

Sobel

Filter masks used to compute the derivatives needed for the gradient are often called *gradient operators, difference operators, edge operators*, or *edge detectors*.

These derivatives can be implemented by filtering an image with the masks in Figs. 10.14(b) and (c).

Masks of size 2×2 are simple conceptually, but they are not as useful for computing edge direction as masks that are symmetric about the center point, the smallest of which are of size 3×3. These masks take into account the nature of the data on opposite sides of the center point and thus carry more information regarding the direction of an edge. The simplest digital approximations to the partial derivatives using masks of size 3×3 are given by

$$g_x = \frac{\partial f}{\partial x} = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3) \tag{10.2-16}$$

Although these equations encompass a larger neighborhood, we are still dealing with *differences* between intensity values, so the conclusions from earlier discussions regarding first-order derivatives still apply.

and

$$g_y = \frac{\partial f}{\partial y} = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7) \tag{10.2-17}$$

In these formulations, the difference between the third and first *rows* of the 3×3 region approximates the derivative in the x-direction, and the difference between the third and first columns approximate the derivate in the y-direction. Intuitively, we would expect these approximations to be more accurate than the approximations obtained using the Roberts operators. Equations (10.2-16) and (10.2-17) can be implemented over an entire image by filtering f with the two masks in Figs. 10.14(d) and (e). These masks are called the *Prewitt operators* (Prewitt [1970]).

A slight variation of the preceding two equations uses a weight of 2 in the center coefficient:

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{10.2-18}$$

and

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{10.2-19}$$

It can be shown (Problem 10.10) that using a 2 in the center location provides image smoothing. Figures 10.14(f) and (g) show the masks used to implement Eqs. (10.2-18) and (10.2-19). These masks are called the *Sobel operators* (Sobel [1970]).

The Prewitt masks are simpler to implement than the Sobel masks, but, the slight computational difference between them typically is not an issue. The fact that the Sobel masks have better noise-suppression (smoothing) characteristics makes them preferable because, as mentioned in the previous section, noise suppression is an important issue when dealing with derivatives. Note that the coefficients of all the masks in Fig. 10.14 sum to zero, thus giving a response of zero in areas of constant intensity, as expected of a derivative operator.

a b
c d

FIGURE 10.15
Prewitt and Sobel masks for detecting diagonal edges.

0	1	1	−1	−1	0
−1	0	1	−1	0	1
−1	−1	0	0	1	1

Prewitt

0	1	2	−2	−1	0
−1	0	1	−1	0	1
−2	−1	0	0	1	2

Sobel

The masks just discussed are used to obtain the gradient components g_x and g_y at every pixel location in an image. These two partial derivatives are then used to estimate edge strength and direction. Computing the magnitude of the gradient requires that g_x and g_y be combined in the manner shown in Eq. (10.2-10). However, this implementation is not always desirable because of the computational burden required by squares and square roots. An approach used frequently is to approximate the magnitude of the gradient by absolute values:

$$M(x, y) \approx |g_x| + |g_y| \tag{10.2-20}$$

This equation is more attractive computationally, and it still preserves relative changes in intensity levels. The price paid for this advantage is that the resulting filters will not be isotropic (invariant to rotation) in general. However, this is not an issue when masks such as the Prewitt and Sobel masks are used to compute g_x and g_y, because these masks give isotropic results only for vertical and horizontal edges. Results would be isotropic only for edges in those two directions, regardless of which of the two equations is used. In addition, Eqs. (10.2-10) and (10.2-20) give identical results for vertical and horizontal edges when the Sobel or Prewitt masks are used (Problem 10.8).

It is possible to modify the 3×3 masks in Fig. 10.14 so that they have their strongest responses along the diagonal directions. Figure 10.15 shows the two additional Prewitt and Sobel masks needed for detecting edges in the diagonal directions.

EXAMPLE 10.6: Illustration of the 2-D gradient magnitude and angle.

■ Figure 10.16 illustrates the absolute value response of the two components of the gradient, $|g_x|$ and $|g_y|$, as well as the gradient image formed from the sum of these two components. The directionality of the horizontal and vertical components of the gradient is evident in Figs. 10.16(b) and (c). Note, for example, how strong the roof tile, horizontal brick joints, and horizontal segments of the windows are in Fig. 10.16(b) compared to other edges. By contrast,

a b
c d

FIGURE 10.16
(a) Original image of size 834×1114 pixels, with intensity values scaled to the range $[0, 1]$. (b) $|g_x|$, the component of the gradient in the x-direction, obtained using the Sobel mask in Fig. 10.14(f) to filter the image. (c) $|g_y|$, obtained using the mask in Fig. 10.14(g). (d) The gradient image, $|g_x| + |g_y|$.

Fig. 10.16(c) favors features such as the vertical components of the façade and windows. It is common terminology to use the term *edge map* when referring to an image whose principal features are edges, such as gradient magnitude images. The intensities of the image in Fig. 10.16(a) were scaled to the range $[0, 1]$. We use values in this range to simplify parameter selection in the various methods for edge detection discussed in this section.

Figure 10.17 shows the gradient angle image computed using Eq. (10.2-11). In general, angle images are not as useful as gradient magnitude images for edge detection, but they do complement the information extracted from an image using the magnitude of the gradient. For instance, the constant intensity areas in Fig. 10.16(a), such as the front edge of the sloping roof and top horizontal bands of the front wall, are constant in Fig. 10.17, indicating that the gradient vector direction at all the pixel locations in those regions is the same.

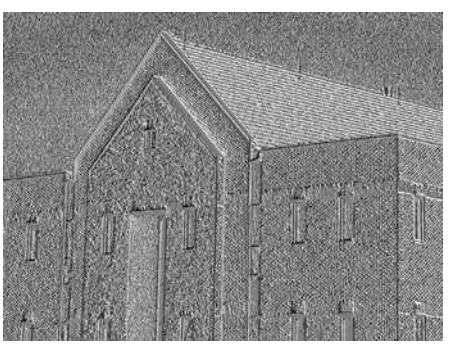


FIGURE 10.17
Gradient angle image computed using Eq. (10.2-11). Areas of constant intensity in this image indicate that the direction of the gradient vector is the same at all the pixel locations in those regions.

As we show in Section 10.2.6, angle information plays a key supporting role in the implementation of the Canny edge detection algorithm, the most advanced edge detection method we discuss in this chapter. ■

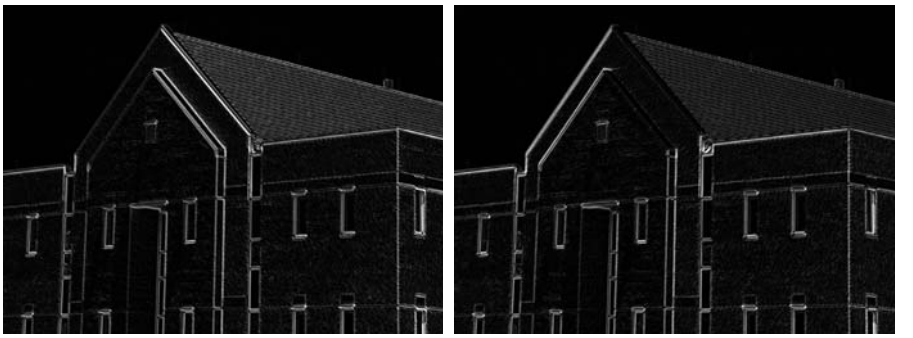
The maximum edge strength (magnitude) of a smoothed image decreases inversely as a function of the size of the smoothing mask (Problem 10.13).

The original image in Fig. 10.16(a) is of reasonably high resolution (834 × 1114 pixels), and at the distance the image was acquired, the contribution made to image detail by the wall bricks is significant. This level of fine detail often is undesirable in edge detection because it tends to act as noise, which is enhanced by derivative computations and thus complicates detection of the principal edges in an image. One way to reduce fine detail is to smooth the image. Figure 10.18 shows the same sequence of images as in Fig. 10.16, but with the original image smoothed first using a 5 × 5 averaging filter (see Section 3.5 regarding smoothing filters). The response of each mask now shows almost no contribution due to the bricks, with the results being dominated mostly by the principal edges.

It is evident in Figs. 10.16 and 10.18 that the horizontal and vertical Sobel masks do not differentiate between edges oriented in the ±45° directions. If it is important to emphasize edges along the diagonal directions, then one of the masks in Fig. 10.15 should be used. Figures 10.19(a) and (b) show the absolute responses of the 45° and −45° Sobel masks, respectively. The stronger diagonal response of these masks is evident in these figures. Both diagonal masks have similar response to horizontal and vertical edges but, as expected, their response in these directions is weaker than the response of the horizontal and vertical masks, as discussed earlier.

a b
c d

FIGURE 10.18
Same sequence as in Fig. 10.16, but with the original image smoothed using a 5 × 5 averaging filter prior to edge detection.



a b

FIGURE 10.19 Diagonal edge detection. (a) Result of using the mask in Fig. 10.15(c). (b) Result of using the mask in Fig. 10.15(d). The input image in both cases was Fig. 10.18(a).

Combining the gradient with thresholding

The results in Fig. 10.18 show that edge detection can be made more selective by smoothing the image prior to computing the gradient. Another approach aimed at achieving the same basic objective is to threshold the gradient image. For example, Fig. 10.20(a) shows the gradient image from Fig. 10.16(d) thresholded, in the sense that pixels with values greater than or equal to 33% of the maximum value of the gradient image are shown in white, while pixels below the threshold value are shown in black. Comparing this image with Fig. 10.18(d), we see that there are fewer edges in the thresholded image, and that the edges in this image are much sharper (see, for example, the edges in the roof tile). On the other hand, numerous edges, such as the 45° line defining the far edge of the roof, are broken in the thresholded image.

The threshold used to generate Fig. 10.20(a) was selected so that most of the small edges caused by the bricks were eliminated. Recall that this was the original objective for smoothing the image in Fig. 10.16 prior to computing the gradient.

When interest lies both in highlighting the principal edges and on maintaining as much connectivity as possible, it is common practice to use both smoothing and thresholding. Figure 10.20(b) shows the result of thresholding Fig. 10.18(d), which is the gradient of the smoothed image. This result shows a

a b

FIGURE 10.20 (a) Thresholded version of the image in Fig. 10.16(d), with the threshold selected as 33% of the highest value in the image; this threshold was just high enough to eliminate most of the brick edges in the gradient image. (b) Thresholded version of the image in Fig. 10.18(d), obtained using a threshold equal to 33% of the highest value in that image.

reduced number of broken edges; for instance, compare the 45° edges in Figs. 10.20(a) and (b). Of course, edges whose intensity values were severely attenuated due to blurring (e.g., the edges in the tile roof) are likely to be totally eliminated by thresholding. We return to the problem of broken edges in Section 10.2.7.

10.2.6 More Advanced Techniques for Edge Detection

The edge-detection methods discussed in the previous section are based simply on filtering an image with one or more masks, with no provisions being made for edge characteristics and noise content. In this section, we discuss more advanced techniques that make an attempt to improve on simple edge-detection methods by taking into account factors such as image noise and the nature of edges themselves.

The Marr-Hildreth edge detector

One of the earliest successful attempts at incorporating more sophisticated analysis into the edge-finding process is attributed to Marr and Hildreth [1980]. Edge-detection methods in use at the time were based on using small operators (such as the Sobel masks), as discussed in the previous section. Marr and Hildreth argued (1) that intensity changes are not independent of image scale and so their detection requires the use of operators of different sizes; and (2) that a sudden intensity change will give rise to a peak or trough in the first derivative or, equivalently, to a zero crossing in the second derivative (as we saw in Fig. 10.10).

To convince yourself that edge detection is not independent of scale, consider, for example, the roof edge in Fig. 10.8(c). If the scale of the image is reduced, the edge will appear thinner.

These ideas suggest that an operator used for edge detection should have two salient features. First and foremost, it should be a differential operator capable of computing a digital approximation of the first or second derivative at every point in the image. Second, it should be capable of being "tuned" to act at any desired scale, so that large operators can be used to detect blurry edges and small operators to detect sharply focused fine detail.

Marr and Hildreth argued that the most satisfactory operator fulfilling these conditions is the filter $\nabla^2 G$ where, as defined in Section 3.6.2, ∇^2 is the Laplacian operator, $(\partial^2/\partial x^2 + \partial^2/\partial y^2)$, and G is the 2-D Gaussian function

$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{10.2-21}$$

It is customary for Eq. (10.2-21) to differ from the definition of a 2-D Gaussian PDF by the constant term $1/2\pi\sigma^2$. If an exact expression is desired in a given application, then the multiplying constant can be appended to the final result in Eq. (10.2-23).

with standard deviation σ (sometimes σ is called the *space constant*). To find an expression for $\nabla^2 G$ we perform the following differentiations:

$$\begin{aligned}\nabla^2 G(x, y) &= \frac{\partial^2 G(x, y)}{\partial x^2} + \frac{\partial^2 G(x, y)}{\partial y^2} \\ &= \frac{\partial}{\partial x}\left[\frac{-x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}\right] + \frac{\partial}{\partial y}\left[\frac{-y}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}\right] \\ &= \left[\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} + \left[\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}\end{aligned} \tag{10.2-22}$$

Collecting terms gives the final expression:

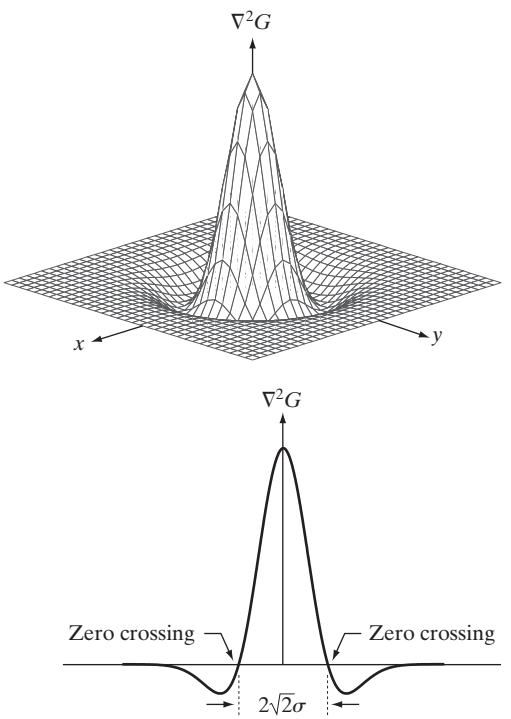
$$\nabla^2 G(x, y) = \left[\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (10.2\text{-}23)$$

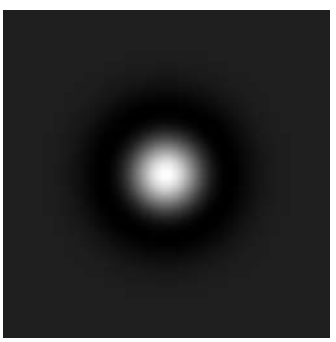
This expression is called the *Laplacian of a Gaussian* (LoG).

Figures 10.21(a) through (c) show a 3-D plot, image, and cross section of the *negative* of the LoG function (note that the zero crossings of the LoG occur at $x^2 + y^2 = 2\sigma^2$, which defines a circle of radius $\sqrt{2}\sigma$ centered on the origin). Because of the shape illustrated in Fig. 10.21(a), the LoG function sometimes is called the *Mexican hat* operator. Figure 10.21(d) shows a 5×5 mask that approximates the shape in Fig. 10.21(a) (in practice we would use the *negative* of this mask). This approximation is not unique. Its purpose is to capture the essential *shape* of the LoG function; in terms of Fig. 10.21(a), this means a positive, central term surrounded by an adjacent, negative region whose values increase as a function of distance from the origin, and a zero outer region. The coefficients must sum to zero so that the response of the mask is zero in areas of constant intensity.

Note the similarity between the cross section in Fig. 10.21(c) and the highpass filter in Fig. 4.37(d). Thus, we can expect the LoG to behave as a highpass filter.

Masks of arbitrary size can be generated by sampling Eq. (10.2-23) and scaling the coefficients so that they sum to zero. A more effective approach for generating a LoG filter is to sample Eq. (10.2-21) to the desired $n \times n$ size and





0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

a b
c d

FIGURE 10.21 (a) Three-dimensional plot of the *negative* of the LoG. (b) Negative of the LoG displayed as an image. (c) Cross section of (a) showing zero crossings. (d) 5×5 mask approximation to the shape in (a). The negative of this mask would be used in practice.

then convolve[†] the resulting array with a Laplacian mask, such as the mask in Fig. 10.4(a). Because convolving an image array with a mask whose coefficients sum to zero yields a result whose elements also sum to zero (see Problems 3.16 and 10.14), this approach automatically satisfies the requirement that the sum of the LoG filter coefficients be zero. We discuss the issue of selecting the size of LoG filter later in this section.

There are two fundamental ideas behind the selection of the operator $\nabla^2 G$. First, the Gaussian part of the operator blurs the image, thus reducing the intensity of structures (including noise) at scales much smaller than σ. Unlike averaging of the form discussed in Section 3.5 and used in Fig. 10.18, the Gaussian function is smooth in both the spatial and frequency domains (see Section 4.8.3), and is thus less likely to introduce artifacts (e.g., ringing) not present in the original image. The other idea concerns ∇^2, the second derivative part of the filter. Although first derivatives can be used for detecting abrupt changes in intensity, they are directional operators. The Laplacian, on the other hand, has the important advantage of being isotropic (invariant to rotation), which not only corresponds to characteristics of the human visual system (Marr [1982]) but also responds equally to changes in intensity in any mask direction, thus avoiding having to use multiple masks to calculate the strongest response at any point in the image.

The Marr-Hildreth algorithm consists of convolving the LoG filter with an input image, $f(x, y)$,

$$g(x, y) = [\nabla^2 G(x, y)] \star f(x, y) \tag{10.2-24}$$

and then finding the zero crossings of $g(x, y)$ to determine the locations of edges in $f(x, y)$. Because these are linear processes, Eq. (10.2-24) can be written also as

$$g(x, y) = \nabla^2 [G(x, y) \star f(x, y)] \tag{10.2-25}$$

This expression is implemented in the spatial domain using Eq. (3.4-2). It can be implemented also in the frequency domain using Eq. (4.7-1).

indicating that we can smooth the image first with a Gaussian filter and then compute the Laplacian of the result. These two equations give identical results.

The Marr-Hildreth edge-detection algorithm may be summarized as follows:

1. Filter the input image with an $n \times n$ Gaussian lowpass filter obtained by sampling Eq. (10.2-21).
2. Compute the Laplacian of the image resulting from Step 1 using, for example, the 3×3 mask in Fig. 10.4(a). [Steps 1 and 2 implement Eq. (10.2-25).]
3. Find the zero crossings of the image from Step 2.

To specify the size of the Gaussian filter, recall that about 99.7% of the volume under a 2-D Gaussian surface lies between $\pm 3\sigma$ about the mean. Thus, as a rule

[†]The LoG is a symmetric filter, so spatial filtering using correlation or convolution yields the same result. We use the convolution terminology here to indicate linear filtering for consistency with the literature on this topic. Also, this gives you exposure to terminology that you will encounter in other contexts. It is important that you keep in mind the comments made at the end of Section 3.4.2 regarding this topic.

of thumb, the size of an $n \times n$ LoG discrete filter should be such that n is the smallest odd integer greater than or equal to 6σ. Choosing a filter mask smaller than this will tend to "truncate" the LoG function, with the degree of truncation being inversely proportional to the size of the mask; using a larger mask would make little difference in the result.

One approach for finding the zero crossings at any pixel, p, of the filtered image, $g(x, y)$, is based on using a 3×3 neighborhood centered at p. A zero crossing at p implies that the *signs* of at least two of its opposing neighboring pixels must differ. There are four cases to test: left/right, up/down, and the two diagonals. If the values of $g(x, y)$ are being compared against a threshold (a common approach), then not only must the signs of opposing neighbors be different, but the absolute value of their numerical difference must also exceed the threshold before we can call p a zero-crossing pixel. We illustrate this approach in Example 10.7 below.

Attempting to find the zero crossings by finding the coordinates (x, y), such that $g(x, y) = 0$ is impractical because of noise and/or computational inaccuracies.

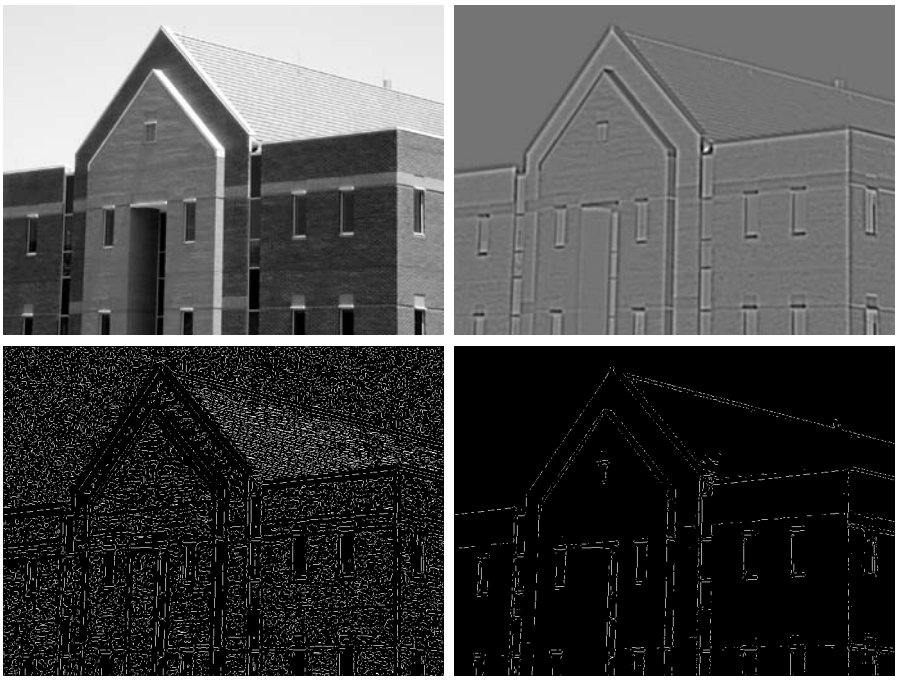
Zero crossings are the key feature of the Marr-Hildreth edge-detection method. The approach discussed in the previous paragraph is attractive because of its simplicity of implementation and because it generally gives good results. If the accuracy of the zero-crossing locations found using this method is inadequate in a particular application, then a technique proposed by Huertas and Medioni [1986] for finding zero crossings with subpixel accuracy can be employed.

EXAMPLE 10.7: Illustration of the Marr-Hildreth edge-detection method.

■ Figure 10.22(a) shows the original building image used earlier and Fig. 10.22(b) is the result of Steps 1 and 2 of the Marr-Hildreth algorithm, using $\sigma = 4$ (approximately 0.5% of the short dimension of the image) and $n = 25$ (the smallest odd integer greater than or equal to 6σ, as discussed earlier). As in Fig. 10.5, the gray tones in this image are due to scaling. Figure 10.22(c) shows the zero crossings obtained using the 3×3 neighborhood approach discussed above with a threshold of zero. Note that all the edges form closed loops. This so-called "spaghetti" effect is a serious drawback of this method when a threshold value of zero is used (Problem 10.15). We avoid closed-loop edges by using a positive threshold.

Figure 10.22(d) shows the result of using a threshold approximately equal to 4% of the maximum value of the LoG image. Note that the majority of the principal edges were readily detected and "irrelevant" features, such as the edges due to the bricks and the tile roof, were filtered out. As we show in the next section, this type of performance is virtually impossible to obtain using the gradient-based edge-detection techniques discussed in the previous section. Another important consequence of using zero crossings for edge detection is that the resulting edges are 1 pixel thick. This property simplifies subsequent stages of processing, such as edge linking. ■

A procedure used sometimes to take into account the fact mentioned earlier that intensity changes are scale dependent is to filter an image with various values of σ. The resulting zero-crossings edge maps are then combined by keeping only the edges that are common to all maps. This approach can yield



a b
c d

FIGURE 10.22
(a) Original image of size 834 × 1114 pixels, with intensity values scaled to the range [0, 1]. (b) Results of Steps 1 and 2 of the Marr-Hildreth algorithm using $\sigma = 4$ and $n = 25$. (c) Zero crossings of (b) using a threshold of 0 (note the closed-loop edges). (d) Zero crossings found using a threshold equal to 4% of the maximum value of the image in (b). Note the thin edges.

useful information, but, due to its complexity, it is used in practice mostly as a design tool for selecting an appropriate value of σ to use with a single filter.

Marr and Hildreth [1980] noted that it is possible to approximate the LoG filter in Eq. (10.2-23) by a difference of Gaussians (DoG):

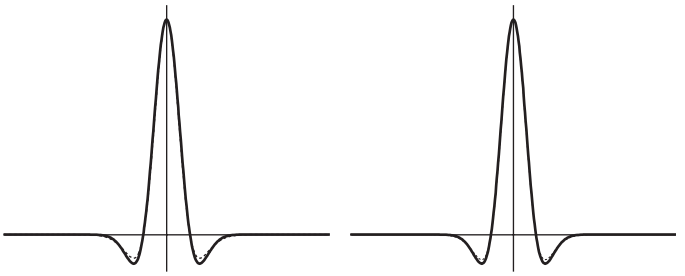
The difference of Gaussians is a highpass filter, as discussed in Section 4.7.4.

$$\text{DoG}(x, y) = \frac{1}{2\pi\sigma_1^2} e^{-\frac{x^2+y^2}{2\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2} e^{-\frac{x^2+y^2}{2\sigma_2^2}} \tag{10.2-26}$$

with $\sigma_1 > \sigma_2$. Experimental results suggest that certain "channels" in the human vision system are selective with respect to orientation and frequency, and can be modeled using Eq. (10.2-26) with a ratio of standard deviations of 1.75:1. Marr and Hildreth suggested that using the ratio 1.6:1 preserves the basic characteristics of these observations and also provides a closer "engineering" approximation to the LoG function. To make meaningful comparisons between the LoG and DoG, the value of σ for the LoG must be selected as in the following equation so that the LoG and DoG have the same zero crossings (Problem 10.17):

$$\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 - \sigma_2^2} \ln\left[\frac{\sigma_1^2}{\sigma_2^2}\right] \tag{10.2-27}$$

Although the zero crossings of the LoG and DoG will be the same when this value of σ is used, their amplitude scales will be different. We can make them compatible by scaling both functions so that they have the same value at the origin.



a b

FIGURE 10.23
(a) Negatives of the LoG (solid) and DoG (dotted) profiles using a standard deviation ratio of 1.75:1. (b) Profiles obtained using a ratio of 1.6:1.

The profiles in Figs. 10.23(a) and (b) were generated with standard deviation ratios of 1:1.75 and 1:1.6, respectively (by convention, the curves shown are inverted, as in Fig. 10.21). The LoG profiles are shown as solid lines while the DoG profiles are dotted. The curves shown are intensity profiles through the center of LoG and DoG arrays generated by sampling Eq. (10.2-23) (with the constant in $1/2\pi\sigma^2$ in front) and Eq. (10.2-26), respectively. The amplitude of all curves at the origin were normalized to 1. As Fig. 10.23(b) shows, the ratio 1:1.6 yielded a closer approximation between the LoG and DoG functions.

Both the LoG and the DoG filtering operations can be implemented with 1-D convolutions instead of using 2-D convolutions directly (Problem 10.19). For an image of size $M \times N$ and a filter of size $n \times n$, doing so reduces the number of multiplications and additions for each convolution from being proportional to n^2MN for 2-D convolutions to being proportional to nMN for 1-D convolutions. This implementation difference is significant. For example, if $n = 25$, a 1-D implementation will require on the order of 12 times fewer multiplication and addition operations than using 2-D convolution.

The Canny edge detector

Although the algorithm is more complex, the performance of the Canny edge detector (Canny [1986]) discussed in this section is superior in general to the edge detectors discussed thus far. Canny's approach is based on three basic objectives:

1. *Low error rate.* All edges should be found, and there should be no spurious responses. That is, the edges detected must be as close as possible to the true edges.
2. *Edge points should be well localized.* The edges located must be as close as possible to the true edges. That is, the distance between a point marked as an edge by the detector and the center of the true edge should be minimum.
3. *Single edge point response.* The detector should return only one point for each true edge point. That is, the number of local maxima around the true edge should be minimum. This means that the detector should not identify multiple edge pixels where only a single edge point exists.

The essence of Canny's work was in expressing the preceding three criteria mathematically and then attempting to find optimal solutions to these formulations. In general, it is difficult (or impossible) to find a closed-form solution

Recall that *white noise* is noise having a frequency spectrum that is continuous and uniform over a specified frequency band. White Gaussian noise is white noise in which the distribution of amplitude values is Gaussian. Gaussian white noise is a good approximation of many real-world situations and generates mathematically tractable models. It has the useful property that its values are statistically independent.

that satisfies all the preceding objectives. However, using numerical optimization with 1-D step edges corrupted by additive white Gaussian noise led to the conclusion that a good approximation[†] to the optimal step edge detector is the *first derivative of a Gaussian:*

$$\frac{d}{dx}e^{-\frac{x^2}{2\sigma^2}} = \frac{-x}{\sigma^2}e^{-\frac{x^2}{2\sigma^2}} \tag{10.2-28}$$

Generalizing this result to 2-D involves recognizing that the 1-D approach *still applies* in the direction of the edge normal (see Fig. 10.12). Because the direction of the normal is unknown beforehand, this would require applying the 1-D edge detector in all possible directions. This task can be approximated by first smoothing the image with a *circular* 2-D Gaussian function, computing the gradient of the result, and then using the gradient magnitude and direction to estimate edge strength and direction at every point.

Let $f(x, y)$ denote the input image and $G(x, y)$ denote the Gaussian function:

$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{10.2-29}$$

We form a smoothed image, $f_s(x, y)$, by convolving G and f:

$$f_s(x, y) = G(x, y)\star f(x, y) \tag{10.2-30}$$

This operation is followed by computing the gradient magnitude and direction (angle), as discussed in Section 10.2.5:

$$M(x, y) = \sqrt{g_x^2 + g_y^2} \tag{10.2-31}$$

and

$$\alpha(x, y) = \tan^{-1}\left[\frac{g_y}{g_x}\right] \tag{10.2-32}$$

with $g_x = \partial f_s/\partial x$ and $g_y = \partial f_s/\partial y$. Any of the filter mask pairs in Fig. 10.14 can be used to obtain g_x and g_y. Equation (10.2-30) is implemented using an $n \times n$ Gaussian mask whose size is discussed below. Keep in mind that $M(x, y)$ and $\alpha(x, y)$ are arrays of the same size as the image from which they are computed.

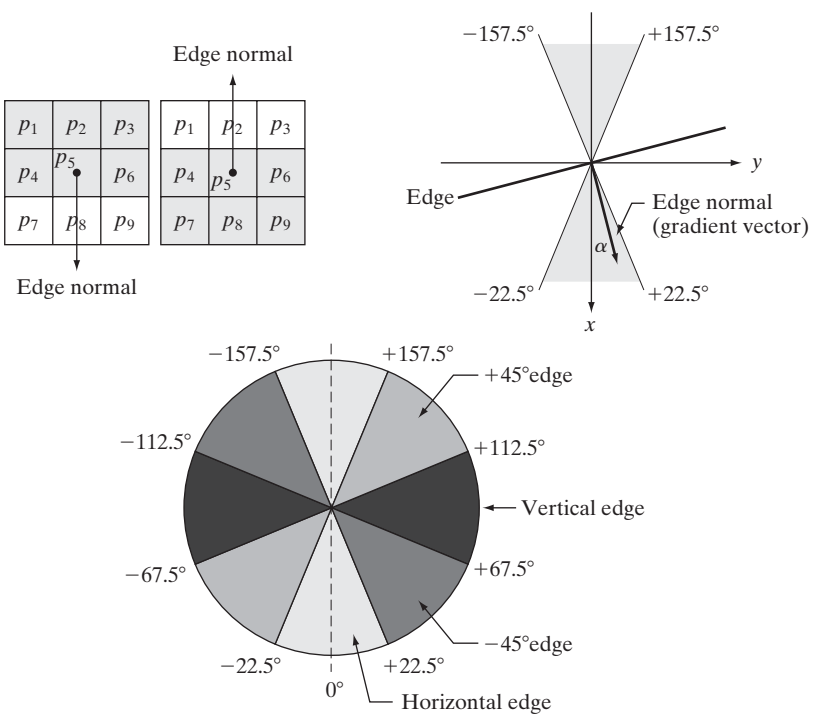
Because it is generated using the gradient, $M(x, y)$ typically contains wide ridges around local maxima (recall the discussion in Section 10.2.1 regarding edges obtained using the gradient). The next step is to thin those ridges. One approach is to use *nonmaxima suppression*. This can be done in several ways, but the essence of the approach is to specify a number of discrete orientations

[†]Canny [1986] showed that using a Gaussian approximation proved only about 20% worse than using the optimized numerical solution. A difference of this magnitude generally is imperceptible in most applications.

of the edge normal (gradient vector). For example, in a 3×3 region we can define four orientations† for an edge passing through the center point of the region: horizontal, vertical, $+45°$ and $-45°$. Figure 10.24(a) shows the situation for the two possible orientations of a horizontal edge. Because we have to quantize all possible edge directions into four, we have to define a range of directions over which we consider an edge to be horizontal. We determine edge direction from the direction of the edge normal, which we obtain directly from the image data using Eq. (10.2-32). As Fig. 10.24(b) shows, if the edge normal is in the range of directions from $-22.5°$ to $22.5°$ or from $-157.5°$ to $157.5°$, we call the edge a horizontal edge. Figure 10.24(c) shows the angle ranges corresponding to the four directions under consideration.

Let d_1, d_2, d_3, and d_4 denote the four basic edge directions just discussed for a 3×3 region: horizontal, $-45°$, vertical, and $+45°$, respectively. We can formulate the following nonmaxima suppression scheme for a 3×3 region centered at *every* point (x, y) in $\alpha(x, y)$:

1. Find the direction d_k that is closest to $\alpha(x, y)$.
2. If the value of $M(x, y)$ is less than at least one of its two neighbors along d_k, let $g_N(x, y) = 0$ (suppression); otherwise, let $g_N(x, y) = M(x, y)$



a b
c

FIGURE 10.24
(a) Two possible orientations of a horizontal edge (in gray) in a 3×3 neighborhood. (b) Range of values (in gray) of α, the direction angle of the *edge normal*, for a horizontal edge. (c) The angle ranges of the edge normals for the four types of edge directions in a 3×3 neighborhood. Each edge direction has two ranges, shown in corresponding shades of gray.

†Keep in mind that every edge has two possible orientations. For example, an edge whose normal is oriented at 0° and an edge whose normal is oriented at 180° are the same *horizontal* edge.

where $g_N(x, y)$ is the nonmaxima-suppressed image. For example, with reference to Fig. 10.24(a), letting (x, y) be at p_5 and assuming a horizontal edge through p_5, the pixels in which we would be interested in Step 2 are p_2 and p_8. Image $g_N(x, y)$ contains only the thinned edges; it is equal to $M(x, y)$ with the nonmaxima edge points suppressed.

The final operation is to threshold $g_N(x, y)$ to reduce false edge points. In Section 10.2.5 we did this using a single threshold, in which all values below the threshold were set to 0. If we set the threshold too low, there will still be some false edges (called *false positives*). If the threshold is set too high, then actual valid edge points will be eliminated (*false negatives*). Canny's algorithm attempts to improve on this situation by using *hysteresis thresholding* which, as we discuss in Section 10.3.6, uses two thresholds: a low threshold, T_L, and a high threshold, T_H. Canny suggested that the ratio of the high to low threshold should be two or three to one.

We can visualize the thresholding operation as creating two additional images

$$g_{NH}(x, y) = g_N(x, y) \geq T_H \tag{10.2-33}$$

and

$$g_{NL}(x, y) = g_N(x, y) \geq T_L \tag{10.2-34}$$

where, initially, both $g_{NH}(x, y)$ and $g_{NL}(x, y)$ are set to 0. After thresholding, $g_{NH}(x, y)$ will have fewer nonzero pixels than $g_{NL}(x, y)$ in general, but all the nonzero pixels in $g_{NH}(x, y)$ will be contained in $g_{NL}(x, y)$ because the latter image is formed with a lower threshold. We eliminate from $g_{NL}(x, y)$ all the nonzero pixels from $g_{NH}(x, y)$ by letting

$$g_{NL}(x, y) = g_{NL}(x, y) - g_{NH}(x, y) \tag{10.2-35}$$

The nonzero pixels in $g_{NH}(x, y)$ and $g_{NL}(x, y)$ may be viewed as being "strong" and "weak" edge pixels, respectively.

After the thresholding operations, all strong pixels in $g_{NH}(x, y)$ are assumed to be valid edge pixels and are so marked immediately. Depending on the value of T_H, the edges in $g_{NH}(x, y)$ typically have gaps. Longer edges are formed using the following procedure:

(a) Locate the next unvisited edge pixel, p, in $g_{NH}(x, y)$.
(b) Mark as valid edge pixels all the weak pixels in $g_{NL}(x, y)$ that are connected to p using, say, 8-connectivity.
(c) If all nonzero pixels in $g_{NH}(x, y)$ have been visited go to Step d. Else, return to Step a.
(d) Set to zero all pixels in $g_{NL}(x, y)$ that were not marked as valid edge pixels.

At the end of this procedure, the final image output by the Canny algorithm is formed by appending to $g_{NH}(x, y)$ all the nonzero pixels from $g_{NL}(x, y)$.

We used two additional images, $g_{NH}(x, y)$ and $g_{NL}(x, y)$, to simplify the discussion. In practice, hysteresis thresholding can be implemented directly during nonmaxima suppression, and thresholding can be implemented directly on $g_N(x, y)$ by forming a list of strong pixels and the weak pixels connected to them.

Summarizing, the Canny edge detection algorithm consists of the following basic steps:

1. Smooth the input image with a Gaussian filter.
2. Compute the gradient magnitude and angle images.
3. Apply nonmaxima suppression to the gradient magnitude image.
4. Use double thresholding and connectivity analysis to detect and link edges.

Although the edges after nonmaxima suppression are thinner than raw gradient edges, edges thicker than 1 pixel can still remain. To obtain edges 1 pixel thick, it is typical to follow Step 4 with one pass of an edge-thinning algorithm (see Section 9.5.5).

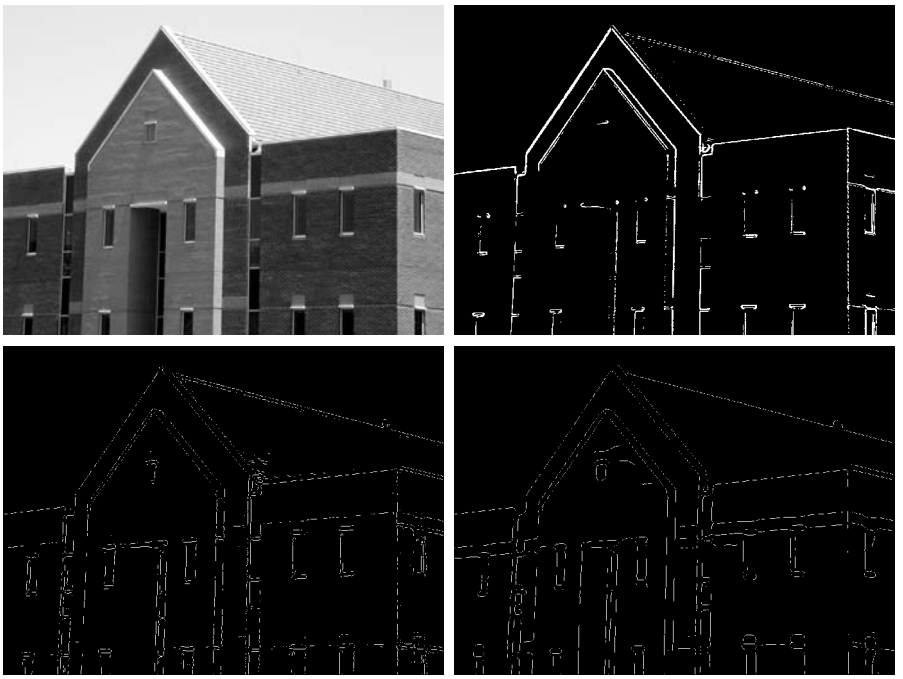
As mentioned earlier, smoothing is accomplished by convolving the input image with a Gaussian mask whose size, $n \times n$, must be specified. We can use the approach discussed in the previous section in connection with the Marr-Hildreth algorithm to determine a value of n. That is, a filter mask generated by sampling Eq. (10.2-29) so that n is the smallest odd integer greater than or equal to 6σ provides essentially the "full" smoothing capability of the Gaussian filter. If practical considerations require a smaller filter mask, then the tradeoff is less smoothing for smaller values of n.

Some final comments on implementation: As noted earlier in the discussion of the Marr-Hildreth edge detector, the 2-D Gaussian function in Eq. (10.2-29) is separable into a product of two 1-D Gaussians. Thus, Step 1 of the Canny algorithm can be formulated as 1-D convolutions that operate on the rows (columns) of the image one at a time and then work on the columns (rows) of the result. Furthermore, if we use the approximations in Eqs. (10.2-12) and (10.2-13), we can also implement the gradient computations required for Step 2 as 1-D convolutions (Problem 10.20).

EXAMPLE 10.8: Illustration of the Canny edge-detection method.

■ Figure 10.25(a) shows the familiar building image. For comparison, Figs. 10.25(b) and (c) show, respectively, the results obtained earlier in Fig. 10.20(b) using the thresholded gradient and Fig. 10.22(d) using the Marr-Hildreth detector. Recall that the parameters used in generating those two images were selected to detect the principal edges while attempting to reduce "irrelevant" features, such as the edges due to the bricks and the tile roof.

Figure 10.25(d) shows the result obtained with the Canny algorithm using the parameters $T_L = 0.04$, $T_H = 0.10$ (2.5 times the value of the low threshold), $\sigma = 4$ and a mask of size 25×25, which corresponds to the smallest odd integer greater than 6σ. These parameters were chosen interactively to achieve the objectives stated in the previous paragraph for the gradient and Marr-Hildreth images. Comparing the Canny image with the other two images, we



a b
c d

FIGURE 10.25
(a) Original image of size 834 × 1114 pixels, with intensity values scaled to the range [0, 1].
(b) Thresholded gradient of smoothed image.
(c) Image obtained using the Marr-Hildreth algorithm.
(d) Image obtained using the Canny algorithm. Note the significant improvement of the Canny image compared to the other two.

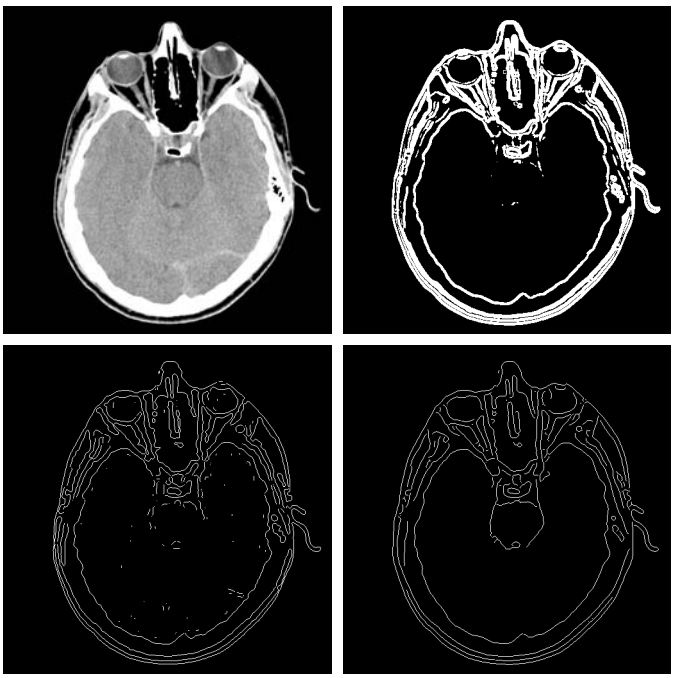
The threshold values given here should be considered only in relative terms. Implementation of most algorithms involves various scaling steps, such as scaling the range of values of the input image to the range [0, 1]. Different scaling schemes obviously would require different values of thresholds from those used in this example.

see significant improvements in detail of the principal edges and, at the same time, more rejection of irrelevant features in the Canny result. Note, for example, that both edges of the concrete band lining the bricks in the upper section of the image were detected by the Canny algorithm, whereas the thresholded gradient lost both of these edges and the Marr-Hildreth image contains only the upper one. In terms of filtering out irrelevant detail, the Canny image does not contain a single edge due to the roof tiles; this is not true in the other two images. The quality of the lines with regard to continuity, thinness, and straightness is also superior in the Canny image. Results such as these have made the Canny algorithm a tool of choice for edge detection. ■

EXAMPLE 10.9: Another illustration of the three principal edge detection methods discussed in this section.

■ As another comparison of the three principal edge-detection methods discussed in this section, consider Fig. 10.26(a) which shows a 512 × 512 head CT image. Our objective in this example is to extract the edges of the outer contour of the brain (the gray region in the image), the contour of the spinal region (shown directly behind the nose, toward the front of the brain), and the outer contour of the head. We wish to generate the thinnest, continuous contours possible, while eliminating edge details related to the gray content in the eyes and brain areas.

Figure 10.26(b) shows a thresholded gradient image that was first smoothed with a 5 × 5 averaging filter. The threshold required to achieve the result shown was 15% of the maximum value of the gradient image. Figure 10.26(c) shows the result obtained with the Marr-Hildreth edge-detection algorithm with a threshold of 0.002, $\sigma = 3$, and a mask of size 19 × 19 pixels. Figure 10.26(d) was obtained using the Canny algorithm with $T_L = 0.05$, $T_H = 0.15$ (3 times the



a b
c d

FIGURE 10.26
(a) Original head CT image of size 512 × 512 pixels, with intensity values scaled to the range [0, 1]. (b) Thresholded gradient of smoothed image. (c) Image obtained using the Marr-Hildreth algorithm. (d) Image obtained using the Canny algorithm. (Original image courtesy of Dr. David R. Pickens, Vanderbilt University.)

value of the low threshold), $\sigma = 2$, and a mask of size 13 × 13, which, as in the Marr-Hildreth case, corresponds to the smallest odd integer greater than 6σ.

The results in Fig. 10.26 correspond closely to the results and conclusions in the previous example in terms of edge quality and the ability to eliminate irrelevant detail. Note also that the Canny algorithm was the only procedure capable of yielding a totally unbroken edge for the posterior boundary of the brain. It was also the only procedure capable of finding the best contours while eliminating all the edges associated with the gray matter in the original image. ■

As might be expected, the price paid for the improved performance of the Canny algorithm is a more complex implementation than the two approaches discussed earlier, requiring also considerably more execution time. In some applications, such as real-time industrial image processing, cost and speed requirements usually dictate the use of simpler techniques, principally the thresholded gradient approach. When edge quality is the driving force, then the Marr-Hildreth and Canny algorithms, especially the latter, offer superior alternatives.

10.2.7 Edge Linking and Boundary Detection

Ideally, edge detection should yield sets of pixels lying only on edges. In practice, these pixels seldom characterize edges completely because of noise, breaks in the edges due to nonuniform illumination, and other effects that introduce spurious discontinuities in intensity values. Therefore, edge detection typically is followed by linking algorithms designed to assemble edge pixels into meaningful edges and/or region boundaries. In this section, we discuss three fundamental approaches to edge linking that are representative of techniques used in practice.

The first requires knowledge about edge points in a local region (e.g., a 3×3 neighborhood); the second requires that points on the boundary of a region be known; and the third is a global approach that works with an entire edge image.

Local processing

One of the simplest approaches for linking edge points is to analyze the characteristics of pixels in a small neighborhood about every point (x, y) that has been declared an edge point by one of the techniques discussed in the previous section. All points that are similar according to predefined criteria are linked, forming an edge of pixels that share common properties according to the specified criteria.

The two principal properties used for establishing similarity of edge pixels in this kind of analysis are (1) the strength (magnitude) and (2) the direction of the gradient vector. The first property is based on Eq. (10.2-10). Let S_{xy} denote the set of coordinates of a neighborhood centered at point (x, y) in an image. An edge pixel with coordinates (s, t) in S_{xy} is similar in *magnitude* to the pixel at (x, y) if

$$|M(s, t) - M(x, y)| \leq E \tag{10.2-36}$$

where E is a positive threshold.

The direction angle of the gradient vector is given by Eq. (10.2-11). An edge pixel with coordinates (s, t) in S_{xy} has an *angle* similar to the pixel at (x, y) if

$$|\alpha(s, t) - \alpha(x, y)| \leq A \tag{10.2-37}$$

where A is a positive angle threshold. As noted in Section 10.2.5, the direction of the edge at (x, y) is *perpendicular* to the direction of the gradient vector at that point.

A pixel with coordinates (s, t) in S_{xy} is linked to the pixel at (x, y) if both magnitude and direction criteria are satisfied. This process is repeated at every location in the image. A record must be kept of linked points as the center of the neighborhood is moved from pixel to pixel. A simple bookkeeping procedure is to assign a different intensity value to each set of linked edge pixels.

The preceding formulation is computationally expensive because all neighbors of every point have to be examined. A simplification particularly well suited for real time applications consists of the following steps:

1. Compute the gradient magnitude and angle arrays, $M(x, y)$ and $\alpha(x, y)$, of the input image, $f(x, y)$.
2. Form a binary image, g, whose value at any pair of coordinates (x, y) is given by:

$$g(x, y) = \begin{cases} 1 & \text{if } M(x, y) > T_M \text{ AND } \alpha(x, y) = A \pm T_A \\ 0 & \text{otherwise} \end{cases}$$

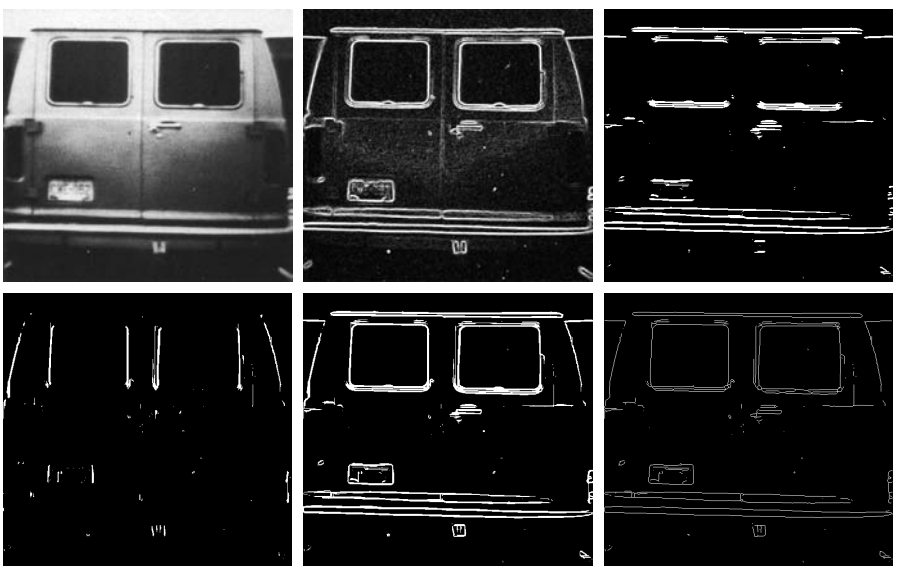
where T_M is a threshold, A is a specified angle direction, and $\pm T_A$ defines a "band" of acceptable directions about A.

3. Scan the rows of g and fill (set to 1) all gaps (sets of 0s) in each row that do not exceed a specified length, K. Note that, by definition, a gap is bounded at both ends by one or more 1s. The rows are processed individually, with no memory between them.
4. To detect gaps in any other direction, θ, rotate g by this angle and apply the horizontal scanning procedure in Step 3. Rotate the result back by $-\theta$.

When interest lies in horizontal and vertical edge linking, Step 4 becomes a simple procedure in which g is rotated ninety degrees, the rows are scanned, and the result is rotated back. This is the application found most frequently in practice and, as the following example shows, this approach can yield good results. In general, image rotation is an expensive computational process so, when linking in numerous angle directions is required, it is more practical to combine Steps 3 and 4 into a single, radial scanning procedure.

EXAMPLE 10.10: Edge linking using local processing.

■ Figure 10.27(a) shows an image of the rear of a vehicle. The objective of this example is to illustrate the use of the preceding algorithm for finding rectangles whose sizes makes them suitable candidates for license plates. The formation of these rectangles can be accomplished by detecting strong horizontal and vertical edges. Figure 10.27(b) shows the gradient magnitude image, $M(x, y)$, and Figs. 10.27(c) and (d) show the result of Steps (3) and (4) of the algorithm obtained by letting T_M equal to 30% of the maximum gradient value,



a b c
d e f

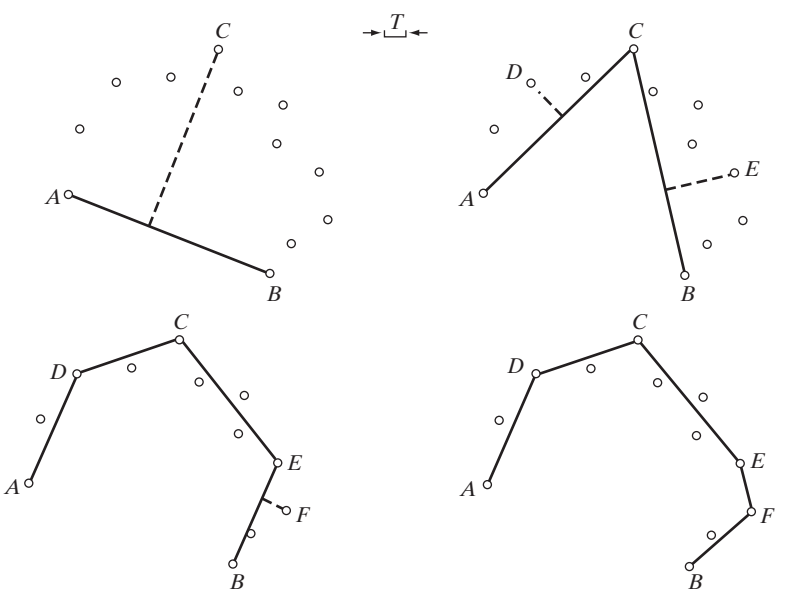
FIGURE 10.27 (a) A 534 × 566 image of the rear of a vehicle. (b) Gradient magnitude image. (c) Horizontally connected edge pixels. (d) Vertically connected edge pixels. (e) The logical OR of the two preceding images. (f) Final result obtained using morphological thinning. (Original image courtesy of Perceptics Corporation.)

$A = 90°$, $T_A = 45°$, and filling in all gaps of 25 or fewer pixels (approximately 5% of the image width). Use of a large range of allowable angle directions was required to detect the rounded corners of the license plate enclosure, as well as the rear windows of the vehicle. Figure 10.27(e) is the result of forming the logical OR of the two preceding images, and Fig. 10.27(f) was obtained by thinning 10.27(e) with the thinning procedure discussed in Section 9.5.5. As Fig. 10.16(f) shows, the rectangle corresponding to the license plate was clearly detected in the image. It would be a simple matter to isolate the license plate from all the rectangles in the image using the fact that the width-to-height ratio of license plates in the U.S. has a distinctive 2:1 proportion. ■

Regional processing

Often, the location of regions of interest in an image are known or can be determined. This implies that knowledge is available regarding the regional membership of pixels in the corresponding edge image. In such situations, we can use techniques for linking pixels on a regional basis, with the desired result being an approximation to the boundary of the region. One approach to this type of processing is functional approximation, where we fit a 2-D curve to the known points. Typically, interest lies in fast-executing techniques that yield an approximation to essential features of the boundary, such as extreme points and concavities. Polygonal approximations are particularly attractive because they can capture the essential shape features of a region while keeping the representation of the boundary (i.e., the vertices of the polygon) relatively simple. In this section, we develop and illustrate an algorithm suitable for this purpose.

Before stating the algorithm, we discuss the mechanics of the procedure using a simple example. Figure 10.28 shows a set of points representing an open curve in which the end points have been labeled A and B. These two



a b
c d

FIGURE 10.28 Illustration of the iterative polygonal fit algorithm.

points are by definition vertices of the polygon. We begin by computing the parameters of a straight line passing through A and B. Then, we compute the perpendicular distance from all other points in the curve to this line and select the point that yielded the maximum distance (ties are resolved arbitrarily). If this distance exceeds a specified threshold, T, the corresponding point, labeled C, is declared a vertex, as Fig. 10.28(a) shows. Lines from A to C and from C to B are then established, and distances from all points between A and C to line AC are obtained. The point corresponding to the maximum distance is declared a vertex, D, if the distance exceeds T; otherwise no new vertices are declared for that segment. A similar procedure is applied to the points between C and B. Figure 10.28(b) shows the result and Fig. 10.28(c) shows the next step. This iterative procedure is continued until no points satisfy the threshold test. Figure 10.28(d) shows the final result which, as you can see, is a reasonable approximation to the shape of a curve fitting the given points.

Two important requirements are implicit in the procedure just explained. First, two starting points must be specified; second, all the points must be ordered (e.g., in a clockwise or counterclockwise direction). When an arbitrary set of points in 2-D does not form a connected path (as is typically the case in edge images) it is not always obvious whether the points belong to a boundary segment (open curve) or a boundary (closed curve). Given that the points are ordered, we can infer whether we are dealing with an open or closed curve by analyzing the distances between points. A large distance between two consecutive points in the ordered sequence relative to the distance between other points as we traverse the sequence of points is a good indication that the curve is open. The end points are then used to start the procedure. If the separation between points tends to be uniform, then we are most likely dealing with a closed curve. In this case, we have several options for selecting the two starting points. One approach is to choose the rightmost and leftmost points in the set. Another is to find the extreme points of the curve (we discuss a way to do this in Section 11.2.1). An algorithm for finding a polygonal fit to open and closed curves may be stated as follows:

1. Let P be a sequence of ordered, distinct, 1-valued points of a binary image. Specify two starting points, A and B. These are the two starting vertices of the polygon.
2. Specify a threshold, T, and two empty stacks, OPEN and CLOSED.
3. If the points in P correspond to a closed curve, put A into OPEN and put B into OPEN *and* into CLOSED. If the points correspond to an open curve, put A into OPEN and B into CLOSED.
4. Compute the parameters of the line passing from the last vertex in CLOSED to the last vertex in OPEN.
5. Compute the distances from the line in Step 4 to all the points in P whose sequence places them between the vertices from Step 4. Select the point, $V_{\max}$, with the maximum distance, $D_{\max}$ (ties are resolved arbitrarily).
6. If $D_{\max} > T$, place $V_{\max}$ at the end of the OPEN stack as a new vertex. Go to Step 4.

See Section 11.1.1 for an algorithm that creates ordered point sequences.

The use of OPEN and CLOSED for the stack names is *not* related to open and closed curves. The stack names indicate simply a stack to store final (CLOSED) vertices or vertices that are in transition (OPEN).

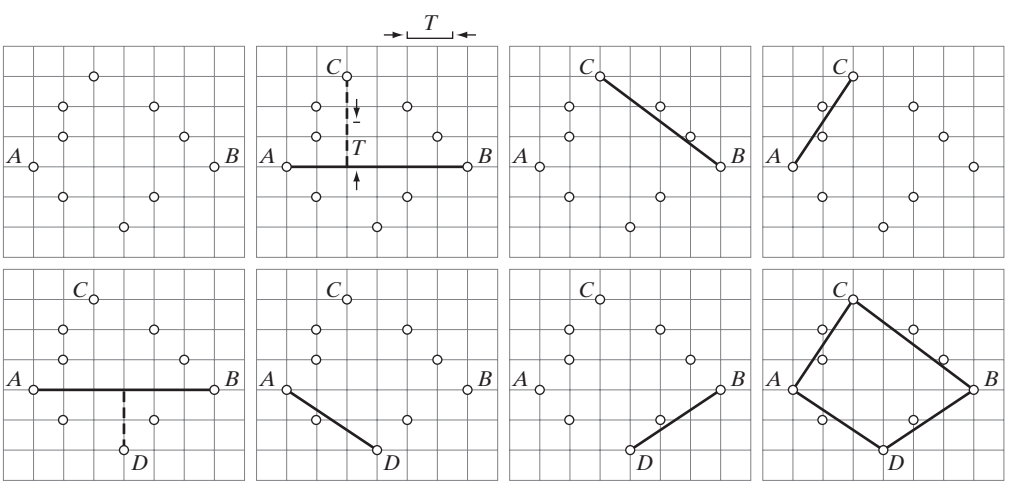
7. Else, remove the last vertex from OPEN and insert it as the last vertex of CLOSED.
8. If OPEN is not empty, go to Step 4.
9. Else, exit. The vertices in CLOSED are the vertices of the polygonal fit to the points in P.

The mechanics of the algorithm are illustrated in the following two examples.

EXAMPLE 10.11: Edge linking using a polygonal approximation.

■ Consider the set of points, P, in Fig. 10.29(a). Assume that these points belong to a closed curve, that they are ordered in a clockwise direction (note that some of the points are not adjacent), and that A and B are selected to be the leftmost and rightmost points in P, respectively. These are the starting vertices, as Table 10.1 shows. Select the first point in the sequence to be the leftmost point, A. Figure 10.29(b) shows the only point (labeled C) in the upper curve segment between A and B that satisfied Step 6 of the algorithm, so it is designated as a new vertex and added to the vertices in the OPEN stack. The second row in Table 10.1 shows C being detected, and the third row shows it being added as the last vertex in OPEN. The threshold, T, in Fig. 10.29(b) is approximately equal to 1.5 subdivisions in the figure grid.

Note in Fig. 10.29(b) that there is a point below line AB that also satisfies Step 6. However, because the points are ordered, only one subset of the points between these two vertices is detected at one time. The other point in the lower segment will be detected later, as Fig. 10.29(e) shows. The key is always to follow the points in the order in which they are given.



a b c d
e f g h

FIGURE 10.29 (a) A set of points in a clockwise path (the points labeled A and B were chosen as the starting vertices). (b) The distance from point C to the line passing through A and B is the largest of all the points between A and B and also passed the threshold test, so C is a new vertex. (d)–(g) Various stages of the algorithm. (h) The final vertices, shown connected with straight lines to form a polygon. Table 10.1 shows step-by-step details.

CLOSED	OPEN	Curve segment processed	Vertex generated
B	B, A	—	A, B
B	B, A	(BA)	C
B	B, A, C	(BC)	—
B, C	B, A	(CA)	—
B, C, A	B	(AB)	D
B, C, A	B, D	(AD)	—
B, C, A, D	B	(DB)	—
B, C, A, D, B	Empty	—	—

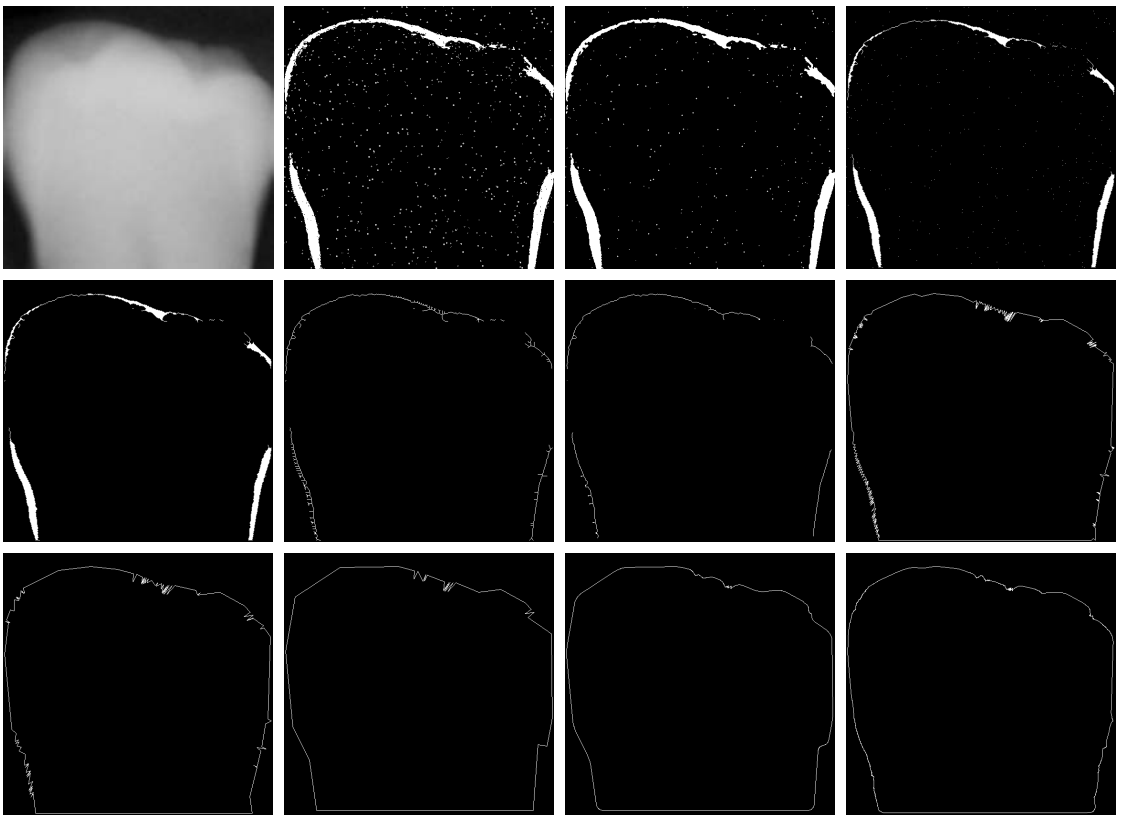
TABLE 10.1 Step-by-step details of the mechanics in Example 10.11.

Table 10.1 shows the individual steps leading to the solution in Fig. 10.29(h). Four vertices were detected, and the figure shows them connected with straight line segments to form a polygon approximating the given boundary points. Note in the table that the vertices detected, B, C, A, D, B are in the counterclockwise direction, even though the points were followed in a clockwise direction to generate the vertices. Had the input been an open curve, the vertices would have been in a clockwise order. The reason for the discrepancy is the way in which the OPEN and CLOSED stacks are initialized. The difference in which stack CLOSED is formed for open and closed curves also leads to the first and last vertices in a closed curve being repeated. This is consistent with how one would differentiate between open and closed polygons given only the vertices. ■

EXAMPLE 10.12: Polygonal fitting of an image boundary.

■ Figure 10.30 shows a more practical example of polygonal fitting. The input image in Fig. 10.30(a) is a 550×566 X-ray image of a human tooth with intensities scaled to the interval $[0, 1]$. The objective of this example is to extract the boundary of the tooth, a process useful in areas such as matching against a database for forensics purposes. Figure 10.30(b) is a gradient image obtained using the Sobel masks and thresholded with $T = 0.1$ (10% of the maximum intensity). As expected for an X-ray image, the noise content is high, so the first step is noise reduction. Because the image is binary, morphological techniques are well suited for this purpose. Figure 10.30(c) shows the result of *majority filtering*, which sets a pixel to 1 if five or more pixels in its 3×3 neighborhood are 1 and sets the pixel to 0 otherwise. Although the noise was reduced, some noise points are still clearly visible. Figure 10.30(d) shows the result of morphological shrinking, which further reduced the noise to isolated points. These were eliminated [Fig. 10.30(e)] by morphological filtering in the manner described in Example 9.4. At this point, the image consists of thick boundaries, which can be thinned by obtaining the morphological skeleton, as Fig. 10.30(f) shows. Finally, Fig. 10.30(g) shows the last step in preprocessing using spur reduction, as discussed in Section 9.5.8.

Next, we fit the points in Fig. 10.30(g) with a polygon. Figures 10.30(h)–(j) show the result of using the polygon fitting algorithm with thresholds equal to 0.5%, 1%, and 2% of the image width ($T = 3$, 6, and 12). The first two results are good approximations to the boundary, but the third is marginal. Excessive jaggedness in all three cases clearly indicates that boundary smoothing is



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

FIGURE 10.30 (a) A 550 × 566 X-ray image of a human tooth. (b) Gradient image. (c) Result of majority filtering. (d) Result of morphological shrinking. (e) Result of morphological cleaning. (f) Skeleton. (g) Spur reduction. (h)–(j) Polygonal fit using thresholds of approximately 0.5%, 1%, and 2% of image width ($T = 3$, 6, and 12). (k) Boundary in (j) smoothed with a 1-D averaging filter of size 1×31 (approximately 5% of image width). (l) Boundary in (h) smoothed with the same filter.

required. Figures 10.30(k) and (l) show the result of convolving a 1-D averaging mask with the boundaries in (j) and (h), respectively. The mask used was a 1×31 array of 1s, corresponding approximately to 5% of the image width. As expected, the result in Fig. 10.30(k) again is marginal in terms of preserving important shape features (e.g., the right side is severely distorted). On the other hand, the result in Fig. 10.30(l) shows significant boundary smoothing and reasonable preservation of shape features. For example, the roundness of the left-upper cusp and the details of the right-upper cusp were preserved with reasonable fidelity. ■

The results in the preceding example are typical of what can be achieved with the polygon fitting algorithm discussed in this section. The advantage of this

algorithm is that it is simple to implement and yields results that generally are quite acceptable. In Section 11.1.3, we discuss a more sophisticated procedure capable of yielding closer fits by computing minimum-perimeter polygons.

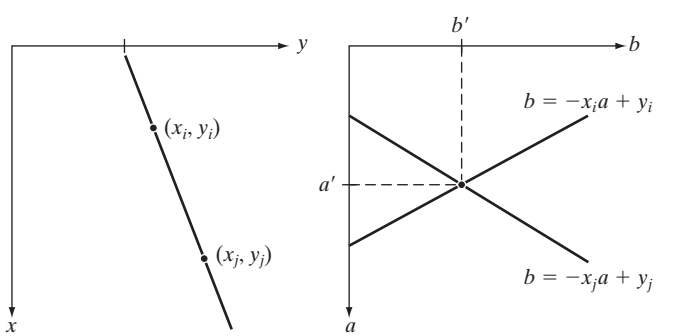
Global processing using the Hough transform

The methods discussed in the previous two sections are applicable in situations where knowledge about pixels belonging to individual objects is at least partially available. For example, in regional processing, it makes sense to link a given set of pixels only if we know that they are part of the boundary of a meaningful region. Often, we have to work with unstructured environments in which all we have is an edge image and no knowledge about where objects of interest might be. In such situations, all pixels are candidates for linking and thus have to be accepted or eliminated based on predefined *global* properties. In this section, we develop an approach based on whether sets of pixels lie on curves of a specified shape. Once detected, these curves form the edges or region boundaries of interest.

Given n points in an image, suppose that we want to find subsets of these points that lie on straight lines. One possible solution is to find first all lines determined by every pair of points and then find all subsets of points that are close to particular lines. This approach involves finding $n(n-1)/2 \sim n^2$ lines and then performing $(n)(n(n-1))/2 \sim n^3$ comparisons of every point to all lines. This is a computationally prohibitive task in all but the most trivial applications.

Hough [1962] proposed an alternative approach, commonly referred to as the *Hough transform*. Consider a point (x_i, y_i) in the xy-plane and the general equation of a straight line in slope-intercept form, $y_i = ax_i + b$. Infinitely many lines pass through (x_i, y_i), but they all satisfy the equation $y_i = ax_i + b$ for varying values of a and b. However, writing this equation as $b = -x_i a + y_i$ and considering the ab-plane (also called *parameter space*) yields the equation of a *single* line for a fixed pair (x_i, y_i). Furthermore, a second point (x_j, y_j) also has a line in parameter space associated with it, and, unless they are parallel, this line intersects the line associated with (x_i, y_i) at some point (a', b'), where a' is the slope and b' the intercept of the line containing *both* (x_i, y_i) and (x_j, y_j) in the xy-plane. In fact, *all* the points on this line have lines in parameter space that intersect at (a', b'). Figure 10.31 illustrates these concepts.

In principle, the parameter-space lines corresponding to all points (x_k, y_k) in the xy-plane could be plotted, and the principal lines in that plane could be found by identifying points in parameter space where large numbers of parameter-space lines intersect. A practical difficulty with this approach, however, is that *a*



a b

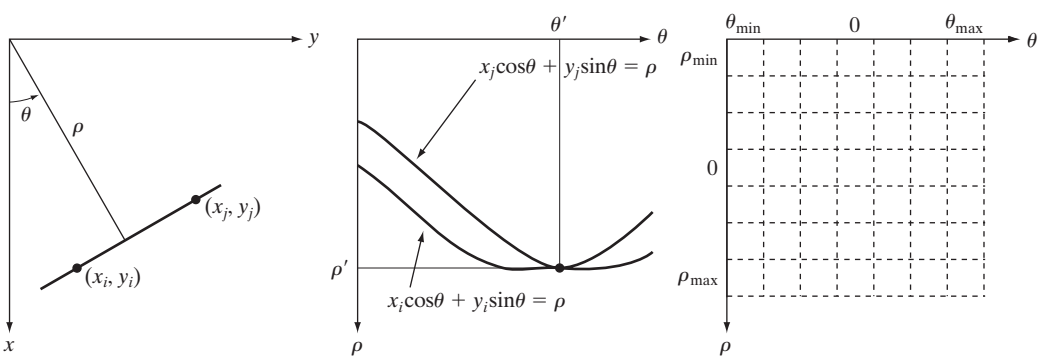
FIGURE 10.31 (a) xy-plane. (b) Parameter space.

(the slope of a line) approaches infinity as the line approaches the vertical direction. One way around this difficulty is to use the normal representation of a line:

$$x \cos\theta + y \sin\theta = \rho \tag{10.2-38}$$

Figure 10.32(a) illustrates the geometrical interpretation of the parameters ρ and θ. A horizontal line has $\theta = 0°$, with ρ being equal to the positive x-intercept. Similarly, a vertical line has $\theta = 90°$, with ρ being equal to the positive y-intercept, or $\theta = -90°$, with ρ being equal to the negative y-intercept. Each sinusoidal curve in Figure 10.32(b) represents the family of lines that pass through a particular point (x_k, y_k) in the xy-plane. The intersection point (ρ', θ') in Fig. 10.32(b) corresponds to the line that passes through both (x_i, y_i) and (x_j, y_j) in Fig. 10.32(a).

The computational attractiveness of the Hough transform arises from subdividing the $\rho\theta$ parameter space into so-called *accumulator cells*, as Fig. 10.32(c) illustrates, where $(\rho_{\min}, \rho_{\max})$ and $(\theta_{\min}, \theta_{\max})$ are the expected ranges of the parameter values: $-90° \le \theta \le 90°$ and $-D \le \rho \le D$, where D is the maximum distance between opposite corners in an image. The cell at coordinates (i, j), with accumulator value $A(i, j)$, corresponds to the square associated with parameter-space coordinates (ρ_i, θ_j). Initially, these cells are set to zero. Then, for every non-background point (x_k, y_k) in the xy-plane, we let θ equal each of the allowed subdivision values on the θ-axis and solve for the corresponding ρ using the equation $\rho = x_k \cos\theta + y_k \sin\theta$. The resulting ρ values are then rounded off to the nearest allowed cell value along the ρ axis. If a choice of θ_p results in solution ρ_q, then we let $A(p, q) = A(p, q) + 1$. At the end of this procedure, a value of P in $A(i, j)$ means that P points in the xy-plane lie on the line $x \cos\theta_j + y \sin\theta_j = \rho_i$. The number of subdivisions in the $\rho\theta$-plane determines the accuracy of the colinearity of these points. It can be shown (Problem 10.24) that the number of computations in the method just discussed is linear with respect to n, the number of non-background points in the xy-plane.



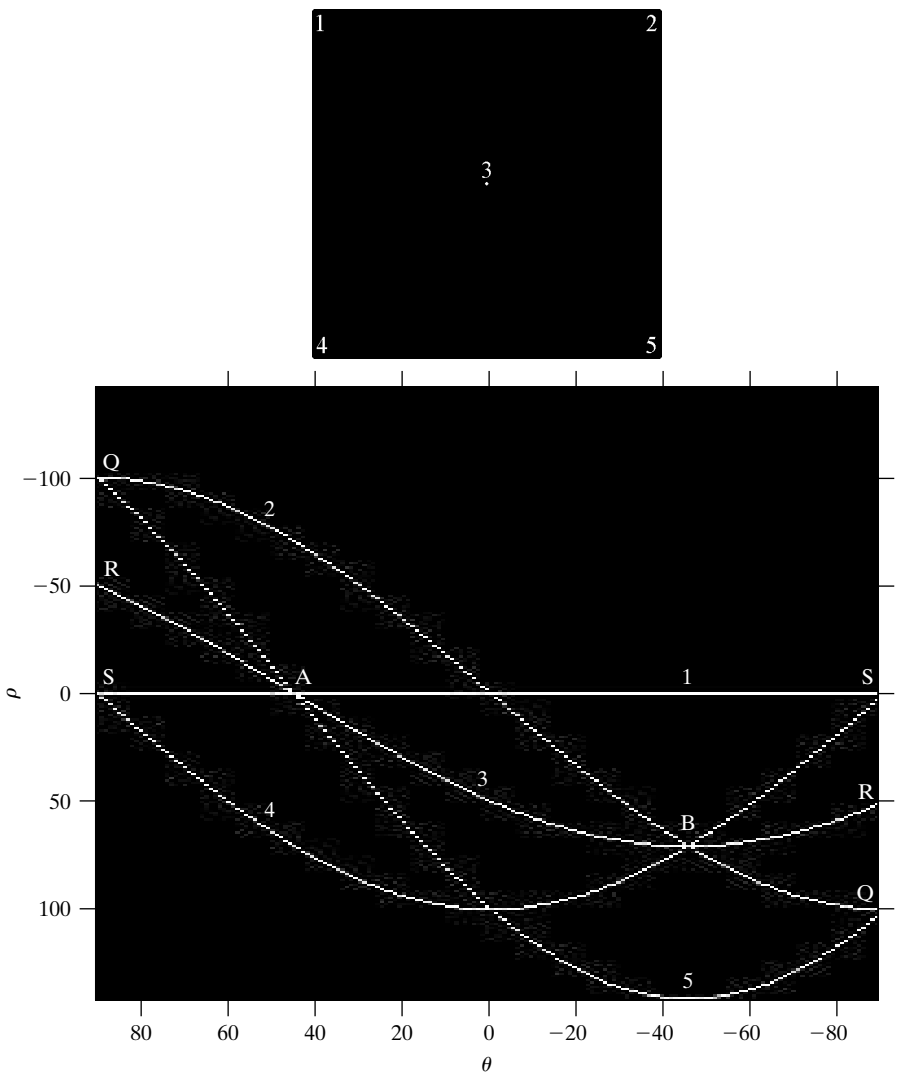
a b c

FIGURE 10.32 (a) (ρ, θ) parameterization of line in the xy-plane. (b) Sinusoidal curves in the $\rho\theta$-plane; the point of intersection (ρ', θ') corresponds to the line passing through points (x_i, y_i) and (x_j, y_j) in the xy-plane. (c) Division of the $\rho\theta$-plane into accumulator cells.

EXAMPLE 10.13: An illustration of basic Hough transform properties.

■ Figure 10.33 illustrates the Hough transform based on Eq. (10.2-38). Figure 10.33(a) shows an image of size 101 × 101 pixels with five labeled points, and Fig. 10.33(b) shows each of these points mapped onto the $\rho\theta$-plane using subdivisions of one unit for the ρ and θ axes. The range of θ values is $\pm 90°$, and the range of the ρ axis is $\pm\sqrt{2}D$, where D is the distance between corners in the image. As Fig. 10.33(c) shows, each curve has a different sinusoidal shape. The horizontal line resulting from the mapping of point 1 is a special case of a sinusoid with zero amplitude.

The points labeled A (not to be confused with accumulator values) and B in Fig. 10.33(b) show the colinearity detection property of the Hough transform.



a
b

FIGURE 10.33
(a) Image of size 101 × 101 pixels, containing five points.
(b) Corresponding parameter space. (The points in (a) were enlarged to make them easier to see.)

Point A denotes the intersection of the curves corresponding to points 1, 3, and 5 in the xy image plane. The location of point A indicates that these three points lie on a straight line passing through the origin ($\rho = 0$) and oriented at 45° [see Fig. 10.32(a)]. Similarly, the curves intersecting at point B in the parameter space indicate that points 2, 3, and 4 lie on a straight line oriented at −45°, and whose distance from the origin is $\rho = 71$ (one-half the diagonal distance from the origin of the image to the opposite corner, rounded to the nearest integer value). Finally, the points labeled Q, R, and S in Fig. 10.33(b) illustrate the fact that the Hough transform exhibits a reflective adjacency relationship at the right and left edges of the parameter space. This property is the result of the manner in which θ and ρ change sign at the $\pm 90°$ boundaries. ■

Although the focus thus far has been on straight lines, the Hough transform is applicable to any function of the form $g(\mathbf{v}, \mathbf{c}) = 0$, where $\mathbf{v}$ is a vector of coordinates and $\mathbf{c}$ is a vector of coefficients. For example, points lying on the circle

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2 \tag{10.2-39}$$

can be detected by using the basic approach just discussed. The difference is the presence of three parameters (c_1, c_2, and c_3), which results in a 3-D parameter space with cube-like cells and accumulators of the form $A(i, j, k)$. The procedure is to increment c_1 and c_2, solve for the c_3 that satisfies Eq. (10.2-39), and update the accumulator cell associated with the triplet (c_1, c_2, c_3). Clearly, the complexity of the Hough transform depends on the number of coordinates and coefficients in a given functional representation. Further generalizations of the Hough transform to detect curves with no simple analytic representations are possible, as is the application of the transform to gray-scale images. Several references dealing with these extensions are included at the end of this chapter.

We return now to the edge-linking problem. An approach based on the Hough transform is as follows:

1. Obtain a *binary* edge image using any of the techniques discussed earlier in this section.
2. Specify subdivisions in the $\rho\theta$-plane.
3. Examine the counts of the accumulator cells for high pixel concentrations.
4. Examine the relationship (principally for continuity) between pixels in a chosen cell.

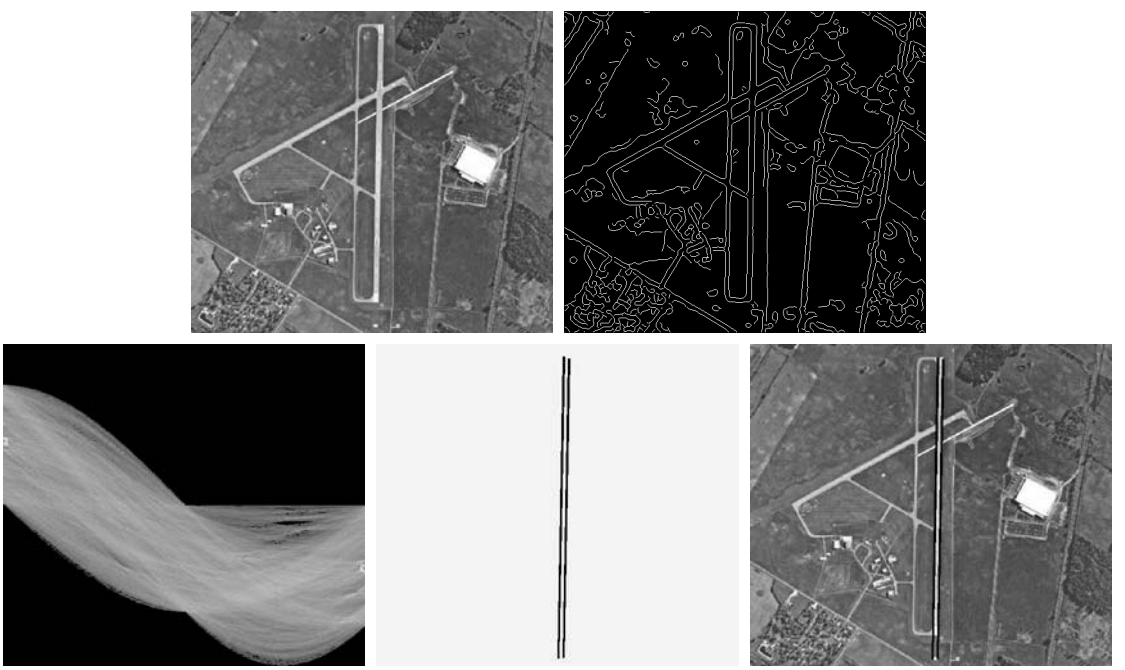
Continuity in this case usually is based on computing the distance between disconnected pixels corresponding to a given accumulator cell. A gap in a line associated with a given cell is bridged if the length of the gap is less than a specified threshold. Note that the mere fact of being able to group lines based on direction is a *global* concept applicable over the entire image, requiring only that we examine pixels associated with specific accumulator cells. This is a significant advantage over the methods discussed in the previous two sections. The following example illustrates these concepts.

EXAMPLE 10.14: Using the Hough transform for edge linking.

■ Figure 10.34(a) shows an aerial image of an airport. The objective of this example is to use the Hough transform to extract the two edges of the principal runway. A solution to such a problem might be of interest, for instance, in applications involving autonomous navigation of air vehicles.

The first step is to obtain an edge image. Figure 10.34(b) shows the edge image obtained using Canny's algorithm with the same parameters and procedure used in Example 10.9. For the purpose of computing the Hough transform, similar results can be obtained using any of the edge-detection techniques discussed in Sections 10.2.5 or 10.2.6. Figure 10.34(c) shows the Hough parameter space obtained using 1° increments for θ and 1 pixel increments for ρ.

The runway of interest is oriented approximately 1° off the north direction, so we select the cells corresponding to $\pm 90°$ and containing the highest count because the runways are the longest lines oriented in these directions. The small white boxes on the edges of Fig. 10.34(c) highlight these cells. As mentioned earlier in connection with Fig. 10.33(b), the Hough transform exhibits adjacency at the edges. Another way of interpreting this property is that a line oriented at $+90°$ and a line oriented at $-90°$ are equivalent (i.e., they are both vertical). Figure 10.34(d) shows the lines corresponding to the two accumulator cells just discussed, and Fig. 10.34(e) shows the lines superimposed on the



a b
c d e

FIGURE 10.34 (a) A 502 × 564 aerial image of an airport. (b) Edge image obtained using Canny's algorithm. (c) Hough parameter space (the boxes highlight the points associated with long vertical lines). (d) Lines in the image plane corresponding to the points highlighted by the boxes). (e) Lines superimposed on the original image.

original image. The lines were obtained by joining all gaps not exceeding 20% of the image height (approximately 100 pixels). These lines clearly correspond to the edges of the runway of interest.

Note that the only key knowledge needed to solve this problem was the orientation of the runway and the observer's position relative to it. In other words, a vehicle navigating autonomously would know that if the runway of interest faces north, and the vehicle's direction of travel also is north, the runway should appear vertically in the image. Other relative orientations are handled in a similar manner. The orientations of runways throughout the world are available in flight charts, and direction of travel is easily obtainable using GPS (Global Positioning System) information. This information also could be used to compute the distance between the vehicle and the runway, thus allowing estimates of parameters such as expected length of lines relative to image size, as we did in this example. ■

10.3 Thresholding

Because of its intuitive properties, simplicity of implementation, and computational speed, image thresholding enjoys a central position in applications of image segmentation. Thresholding was introduced in Section 3.1.1, and we have used it in various discussions since then. In this section, we discuss thresholding in a more formal way and develop techniques that are considerably more general than what has been presented thus far.

10.3.1 Foundation

In the previous section, regions were identified by first finding edge segments and then attempting to link the segments into boundaries. In this section, we discuss techniques for partitioning images directly into regions based on intensity values and/or properties of these values.

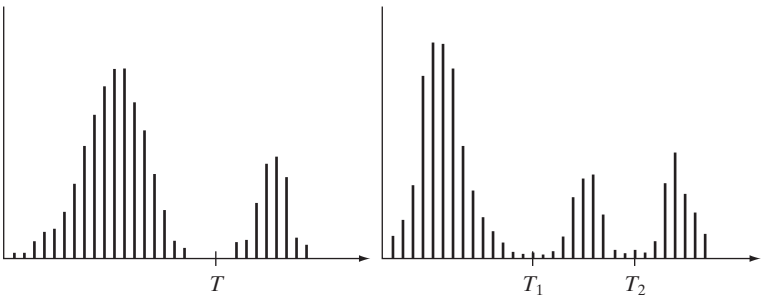
The basics of intensity thresholding

Suppose that the intensity histogram in Fig. 10.35(a) corresponds to an image, $f(x, y)$, composed of light objects on a dark background, in such a way that object and background pixels have intensity values grouped into two dominant modes. One obvious way to extract the objects from the background is to select a threshold, T, that separates these modes. Then, any point (x, y) in the image at which $f(x, y) > T$ is called an *object point*; otherwise, the point is called a *background point*. In other words, the segmented image, $g(x, y)$, is given by

Although we follow convention in using 0 intensity for the background and 1 for object pixels, any two distinct values can be used in Eq. (10.3-1).

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T \end{cases} \tag{10.3-1}$$

When T is a constant applicable over an entire image, the process given in this equation is referred to as *global thresholding*. When the value of T changes over an image, we use the term *variable thresholding*. The term *local* or *regional thresholding* is used sometimes to denote variable thresholding in



a b

FIGURE 10.35 Intensity histograms that can be partitioned (a) by a single threshold, and (b) by dual thresholds.

which the value of T at any point (x, y) in an image depends on properties of a neighborhood of (x, y) (for example, the average intensity of the pixels in the neighborhood). If T depends on the spatial coordinates (x, y) themselves, then variable thresholding is often referred to as *dynamic* or *adaptive* thresholding. Use of these terms is not universal, and one is likely to see them used interchangeably in the literature on image processing.

Figure 10.35(b) shows a more difficult thresholding problem involving a histogram with three dominant modes corresponding, for example, to two types of light objects on a dark background. Here, *multiple thresholding* classifies a point (x, y) as belonging to the background if $f(x, y) \leq T_1$, to one object class if $T_1 < f(x, y) \leq T_2$, and to the other object class if $f(x, y) > T_2$. That is, the segmented image is given by

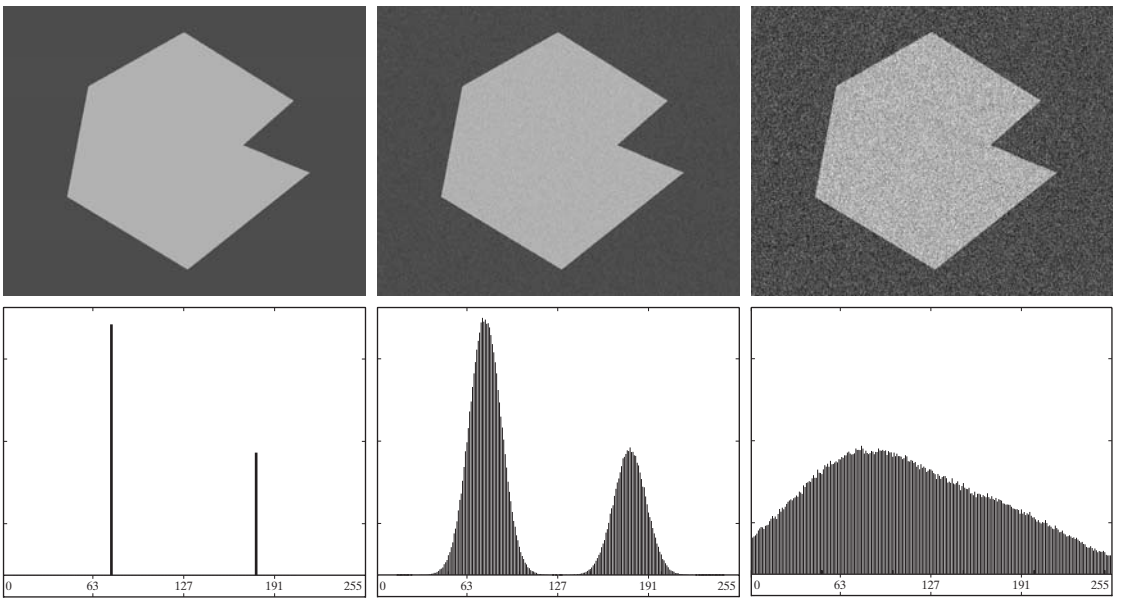
$$g(x, y) = \begin{cases} a & \text{if } f(x, y) > T_2 \\ b & \text{if } T_1 < f(x, y) \leq T_2 \\ c & \text{if } f(x, y) \leq T_1 \end{cases} \tag{10.3-2}$$

where a, b, and c are any three distinct intensity values. We discuss dual thresholding in Section 10.3.6. Segmentation problems requiring more than two thresholds are difficult (often impossible) to solve, and better results usually are obtained using other methods, such as variable thresholding, as discussed in Section 10.3.7, or region growing, as discussed in Section 10.4.

Based on the preceding discussion, we may infer intuitively that the success of intensity thresholding is directly related to the width and depth of the valley(s) separating the histogram modes. In turn, the key factors affecting the properties of the valley(s) are: (1) the separation between peaks (the further apart the peaks are, the better the chances of separating the modes); (2) the noise content in the image (the modes broaden as noise increases); (3) the relative sizes of objects and background; (4) the uniformity of the illumination source; and (5) the uniformity of the reflectance properties of the image.

The role of noise in image thresholding

As an illustration of how noise affects the histogram of an image, consider Fig. 10.36(a). This simple synthetic image is free of noise, so its histogram consists of two "spike" modes, as Fig. 10.36(d) shows. Segmenting this image into two regions is a trivial task involving a threshold placed anywhere between the two



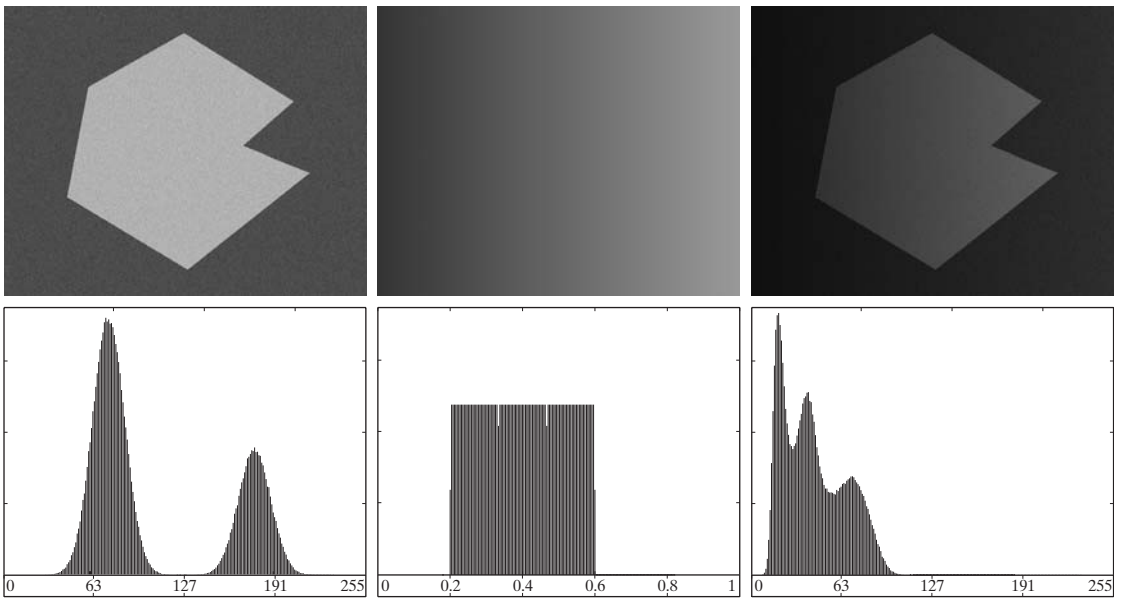
a b c
d e f

FIGURE 10.36 (a) Noiseless 8-bit image. (b) Image with additive Gaussian noise of mean 0 and standard deviation of 10 intensity levels. (c) Image with additive Gaussian noise of mean 0 and standard deviation of 50 intensity levels. (d)–(f) Corresponding histograms.

modes. Figure 10.36(b) shows the original image corrupted by Gaussian noise of zero mean and a standard deviation of 10 intensity levels. Although the corresponding histogram modes are now broader [Fig. 10.36(e)], their separation is large enough so that the depth of the valley between them is sufficient to make the modes easy to separate. A threshold placed midway between the two peaks would do a nice job of segmenting the image. Figure 10.36(c) shows the result of corrupting the image with Gaussian noise of zero mean and a standard deviation of 50 intensity levels. As the histogram in Fig. 10.36(f) shows, the situation is much more serious now, as there is no way to differentiate between the two modes. Without additional processing (such as the methods discussed in Sections 10.3.4 and 10.3.5) we have little hope of finding a suitable threshold for segmenting this image.

The role of illumination and reflectance

Figure 10.37 illustrates the effect that illumination can have on the histogram of an image. Figure 10.37(a) is the noisy image from Fig. 10.36(b), and Fig. 10.37(d) shows its histogram. As before, this image is easily segmentable with a single threshold. We can illustrate the effects of nonuniform illumination by multiplying the image in Fig. 10.37(a) by a variable intensity function, such as the intensity ramp in Fig. 10.37(b), whose histogram is shown in Fig. 10.37(e). Figure 10.37(c) shows the product of the image and this shading pattern. As Fig. 10.37(f) shows, the deep valley between peaks was corrupted to the point



a b c
d e f

FIGURE 10.37 (a) Noisy image. (b) Intensity ramp in the range [0.2, 0.6]. (c) Product of (a) and (b). (d)–(f) Corresponding histograms.

where separation of the modes without additional processing (see Sections 10.3.4 and 10.3.5) is no longer possible. Similar results would be obtained if the illumination was perfectly uniform, but the reflectance of the image was not, due, for example, to natural reflectivity variations in the surface of objects and/or background.

In theory, the histogram of a ramp image is uniform. In practice, achieving perfect uniformity depends on the size of the image and number of intensity bits. For example, a 256 × 256, 256-level ramp image has a uniform histogram, but a 256 × 257 ramp image with the same number of intensities does not.

The key point in the preceding paragraph is that illumination and reflectance play a central role in the success of image segmentation using thresholding or other segmentation techniques. Therefore, controlling these factors when it is possible to do so should be the first step considered in the solution of a segmentation problem. There are three basic approaches to the problem when control over these factors is not possible. One is to correct the shading pattern directly. For example, nonuniform (but fixed) illumination can be corrected by multiplying the image by the inverse of the pattern, which can be obtained by imaging a flat surface of constant intensity. The second approach is to attempt to correct the global shading pattern via processing using, for example, the top-hat transformation introduced in Section 9.6.3. The third approach is to "work around" nonuniformities using variable thresholding, as discussed in Section 10.3.7.

10.3.2 Basic Global Thresholding

As noted in the previous section, when the intensity distributions of objects and background pixels are sufficiently distinct, it is possible to use a single (*global*) threshold applicable over the entire image. In most applications, there

is usually enough variability between images that, even if global thresholding is a suitable approach, an algorithm capable of estimating automatically the threshold value for each image is required. The following iterative algorithm can be used for this purpose:

1. Select an initial estimate for the global threshold, T.
2. Segment the image using T in Eq. (10.3-1). This will produce two groups of pixels: G_1 consisting of all pixels with intensity values $> T$, and G_2 consisting of pixels with values $\leq T$.
3. Compute the average (mean) intensity values m_1 and m_2 for the pixels in G_1 and G_2, respectively.
4. Compute a new threshold value:

$$T = \frac{1}{2}(m_1 + m_2)$$

5. Repeat Steps 2 through 4 until the difference between values of T in successive iterations is smaller than a predefined parameter ΔT.

This simple algorithm works well in situations where there is a reasonably clear valley between the modes of the histogram related to objects and background. Parameter ΔT is used to control the number of iterations in situations where speed is an important issue. In general, the larger ΔT is, the fewer iterations the algorithm will perform. The initial threshold must be chosen greater than the minimum and less than maximum intensity level in the image (Problem 10.28). The average intensity of the image is a good initial choice for T.

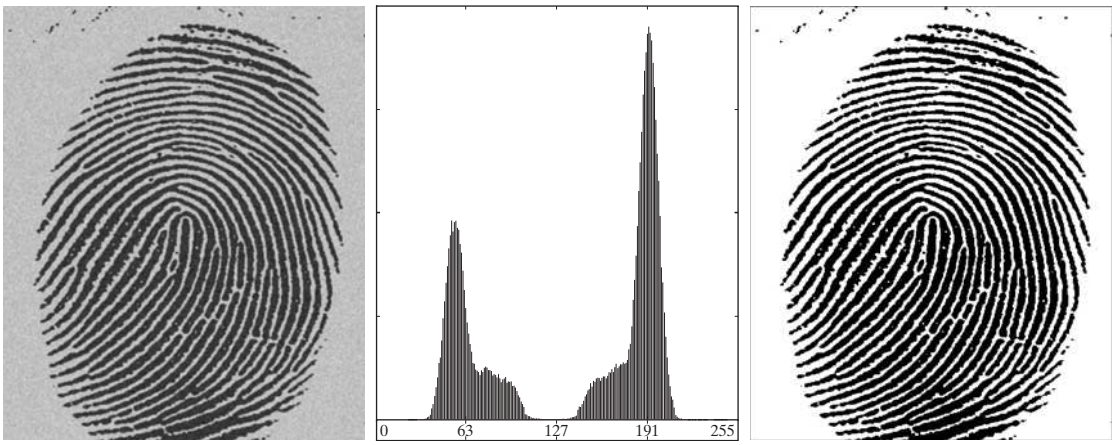
EXAMPLE 10.15: Global thresholding.

■ Figure 10.38 shows an example of segmentation based on a threshold estimated using the preceding algorithm. Figure 10.38(a) is the original image, and Fig. 10.38(b) is the image histogram, showing a distinct valley. Application of the preceding iterative algorithm resulted in the threshold $T = 125.4$ after three iterations, starting with $T = m$ (the average image intensity), and using $\Delta T = 0$. Figure 10.38(c) shows the result obtained using $T = 125$ to segment the original image. As expected from the clear separation of modes in the histogram, the segmentation between object and background was quite effective. ■

The preceding algorithm was stated in terms of successively thresholding the input image and calculating the means at each step because it is more intuitive to introduce it in this manner. However, it is possible to develop a more efficient procedure by expressing all computations in the terms of the image histogram, which has to be computed only once (Problem 10.26).

10.3.3 Optimum Global Thresholding Using Otsu's Method

Thresholding may be viewed as a statistical-decision theory problem whose objective is to minimize the average error incurred in assigning pixels to two or more groups (also called *classes*). This problem is known to have an elegant closed-form solution known as the *Bayes decision rule* (see Section 12.2.2). The solution is based on only two parameters: the probability density function (PDF) of the intensity levels of each class and the probability that each class occurs in a given application. Unfortunately, estimating PDFs is not a trivial



a b c

FIGURE 10.38 (a) Noisy fingerprint. (b) Histogram. (c) Segmented result using a global threshold (the border was added for clarity). (Original courtesy of the National Institute of Standards and Technology.)

matter, so the problem usually is simplified by making workable assumptions about the form of the PDFs, such as assuming that they are Gaussian functions. Even with simplifications, the process of implementing solutions using these assumptions can be complex and not always well-suited for practical applications.

The approach discussed in this section, called *Otsu's method* (Otsu [1979]), is an attractive alternative. The method is optimum in the sense that it maximizes the *between-class variance*, a well-known measure used in statistical discriminant analysis. The basic idea is that well-thresholded classes should be distinct with respect to the intensity values of their pixels and, conversely, that a threshold giving the best separation between classes in terms of their intensity values would be the best (optimum) threshold. In addition to its optimality, Otsu's method has the important property that it is based entirely on computations performed on the histogram of an image, an easily obtainable 1-D array.

Let $\{0, 1, 2, \ldots, L-1\}$ denote the L distinct intensity levels in a digital image of size $M \times N$ pixels, and let n_i denote the number of pixels with intensity i. The total number, MN, of pixels in the image is $MN = n_0 + n_1 + n_2 + \cdots + n_{L-1}$. The normalized histogram (see Section 3.3) has components $p_i = n_i/MN$, from which it follows that

$$\sum_{i=0}^{L-1} p_i = 1, \qquad p_i \geq 0 \tag{10.3-3}$$

Now, suppose that we select a threshold $T(k) = k, 0 < k < L-1$, and use it to threshold the input image into two classes, C_1 and C_2, where C_1 consists of all the pixels in the image with intensity values in the range $[0, k]$ and C_2 consists of the pixels with values in the range $[k+1, L-1]$. Using this threshold, the probability, $P_1(k)$, that a pixel is assigned to (i.e., thresholded into) class C_1 is given by the cumulative sum

$$P_1(k) = \sum_{i=0}^{k} p_i \tag{10.3-4}$$

Viewed another way, this is the probability of class C_1 occurring. For example, if we set $k = 0$, the probability of class C_1 having any pixels assigned to it is zero. Similarly, the probability of class C_2 occurring is

$$P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1(k) \tag{10.3-5}$$

From Eq. (3.3-18), the mean intensity value of the pixels assigned to class C_1 is

$$\begin{aligned} m_1(k) &= \sum_{i=0}^{k} iP(i/C_1) \\ &= \sum_{i=0}^{k} iP(C_1/i)P(i)/P(C_1) \\ &= \frac{1}{P_1(k)} \sum_{i=0}^{k} ip_i \end{aligned} \tag{10.3-6}$$

where $P_1(k)$ is given in Eq. (10.3-4). The term $P(i/C_1)$ in the first line of Eq. (10.3-6) is the probability of value i, given that i comes from class C_1. The second line in the equation follows from Bayes' formula:

$$P(A/B) = P(B/A)P(A)/P(B)$$

The third line follows from the fact that $P(C_1/i)$, the probability of C_1 given i, is 1 because we are dealing only with values of i from class C_1. Also, $P(i)$ is the probability of the ith value, which is simply the ith component of the histogram, p_i. Finally, $P(C_1)$ is the probability of class C_1, which we know from Eq. (10.3-4) is equal to $P_1(k)$.

Similarly, the mean intensity value of the pixels assigned to class C_2 is

$$\begin{aligned} m_2(k) &= \sum_{i=k+1}^{L-1} iP(i/C_2) \\ &= \frac{1}{P_2(k)} \sum_{i=k+1}^{L-1} ip_i \end{aligned} \tag{10.3-7}$$

The cumulative mean (average intensity) up to level k is given by

$$m(k) = \sum_{i=0}^{k} ip_i \tag{10.3-8}$$

and the average intensity of the entire image (i.e., the *global* mean) is given by

$$m_G = \sum_{i=0}^{L-1} ip_i \tag{10.3-9}$$

The validity of the following two equations can be verified by direct substitution of the preceding results:

$$P_1 m_1 + P_2 m_2 = m_G \tag{10.3-10}$$

and

$$P_1 + P_2 = 1 \tag{10.3-11}$$

where we have omitted the ks temporarily in favor of notational clarity.

In order to evaluate the "goodness" of the threshold at level k we use the normalized, dimensionless metric

$$\eta = \frac{\sigma_B^2}{\sigma_G^2} \tag{10.3-12}$$

where σ_G^2 is the *global variance* [i.e., the intensity variance of all the pixels in the image, as given in Eq. (3.3-19)],

$$\sigma_G^2 = \sum_{i=0}^{L-1} (i - m_G)^2 p_i \tag{10.3-13}$$

and σ_B^2 is the *between-class variance*, defined as

$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 \tag{10.3-14}$$

This expression can be written also as

$$\begin{aligned} \sigma_B^2 &= P_1 P_2 (m_1 - m_2)^2 \\ &= \frac{(m_G P_1 - m)^2}{P_1(1 - P_1)} \end{aligned} \tag{10.3-15}$$

The second step in Eq. (10.3-15) makes sense only if P_1 is greater than 0 and less than 1, which, in view of Eq. (10.3-11), implies that P_2 must satisfy the same condition.

where m_G and m are as stated earlier. The first line of this equation follows from Eqs. (10.3-14), (10.3-10), and (10.3-11). The second line follows from Eqs. (10.3-5) through (10.3-9). This form is slightly more efficient computationally because the global mean, m_G, is computed only once, so only two parameters, m and P_1, need to be computed for any value of k.

We see from the first line in Eq. (10.3-15) that the farther the two means m_1 and m_2 are from each other the larger σ_B^2 will be, indicating that the between-class variance is a measure of *separability* between classes. Because σ_G^2 is a constant, it follows that η also is a measure of separability, and maximizing this metric is equivalent to maximizing σ_B^2. The objective, then, is to determine the threshold value, k, that maximizes the between-class variance, as stated at the beginning of this section. Note that Eq. (10.3-12) assumes implicitly that $\sigma_G^2 > 0$. This variance can be zero only when all the intensity levels in the image are the same, which implies the existence of only one class of pixels. This in turn means that $\eta = 0$ for a constant image since the separability of a single class from itself is zero.

Reintroducing k, we have the final results:

$$\eta(k) = \frac{\sigma_B^2(k)}{\sigma_G^2} \tag{10.3-16}$$

and

$$\sigma_B^2(k) = \frac{\left[m_G P_1(k) - m(k)\right]^2}{P_1(k)\left[1 - P_1(k)\right]} \tag{10.3-17}$$

Then, the optimum threshold is the value, k^*, that maximizes $\sigma_B^2(k)$:

$$\sigma_B^2(k^*) = \max_{0 \le k \le L-1} \sigma_B^2(k) \tag{10.3-18}$$

In other words, to find k^* we simply evaluate Eq. (10.3-18) for all *integer* values of k (such that the condition $0 < P_1(k) < 1$ holds) and select that value of k that yielded the maximum $\sigma_B^2(k)$. If the maximum exists for more than one value of k, it is customary to average the various values of k for which $\sigma_B^2(k)$ is maximum. It can be shown (Problem 10.33) that a maximum always exists, subject to the condition that $0 < P_1(k) < 1$. Evaluating Eqs. (10.3-17) and (10.3-18) for all values of k is a relatively inexpensive computational procedure, because the maximum number of integer values that k can have is L.

Once k^* has been obtained, the input image $f(x, y)$ is segmented as before:

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > k^* \\ 0 & \text{if } f(x, y) \le k^* \end{cases} \tag{10.3-19}$$

for $x = 0, 1, 2, \ldots, M - 1$ and $y = 0, 1, 2, \ldots, N - 1$. Note that all the quantities needed to evaluate Eq. (10.3-17) are obtained using only the histogram of $f(x, y)$. In addition to the optimum threshold, other information regarding the segmented image can be extracted from the histogram. For example, $P_1(k^*)$ and $P_2(k^*)$, the class probabilities evaluated at the optimum threshold, indicate the portions of the areas occupied by the classes (groups of pixels) in the thresholded image. Similarly, the means $m_1(k^*)$ and $m_2(k^*)$ are estimates of the average intensity of the classes in the original image.

The normalized metric η, evaluated at the optimum threshold value, $\eta(k^*)$, can be used to obtain a quantitative estimate of the separability of classes, which in turn gives an idea of the ease of thresholding a given image. This measure has values in the range

Although our interest is in the value of η at the optimum threshold, k^*, this inequality holds in general for any value of k in the range $[0, L - 1]$.

$$0 \le \eta(k^*) \le 1 \tag{10.3-20}$$

The lower bound is attainable only by images with a single, constant intensity level, as mentioned earlier. The upper bound is attainable only by 2-valued images with intensities equal to 0 and $L - 1$ (Problem 10.34).

Otsu's algorithm may be summarized as follows:

1. Compute the normalized histogram of the input image. Denote the components of the histogram by p_i, $i = 0, 1, 2, \ldots, L - 1$.
2. Compute the cumulative sums, $P_1(k)$, for $k = 0, 1, 2, \ldots, L - 1$, using Eq. (10.3-4).
3. Compute the cumulative means, $m(k)$, for $k = 0, 1, 2, \ldots, L - 1$, using Eq. (10.3-8).
4. Compute the global intensity mean, m_G, using (10.3-9).
5. Compute the between-class variance, $\sigma_B^2(k)$, for $k = 0, 1, 2, \ldots, L - 1$, using Eq. (10.3-17).
6. Obtain the Otsu threshold, k^*, as the value of k for which $\sigma_B^2(k)$ is maximum. If the maximum is not unique, obtain k^* by averaging the values of k corresponding to the various maxima detected.
7. Obtain the separability measure, η^*, by evaluating Eq. (10.3-16) at $k = k^*$.

The following example illustrates the preceding concepts.

EXAMPLE 10.16: Optimum global thresholding using Otsu's method.

■ Figure 10.39(a) shows an optical microscope image of polymersome cells, and Fig. 10.39(b) shows its histogram. The objective of this example is to segment the molecules from the background. Figure 10.39(c) is the result of using the basic global thresholding algorithm developed in the previous section. Because the histogram has no distinct valleys and the intensity difference between the background and objects is small, the algorithm failed to achieve the desired segmentation. Figure 10.39(d) shows the result obtained using Otsu's method. This result obviously is superior to Fig. 10.39(c). The threshold value computed by the basic algorithm was 169, while the threshold computed by Otsu's method was 181, which is closer to the lighter areas in the image defining the cells. The separability measure η was 0.467.

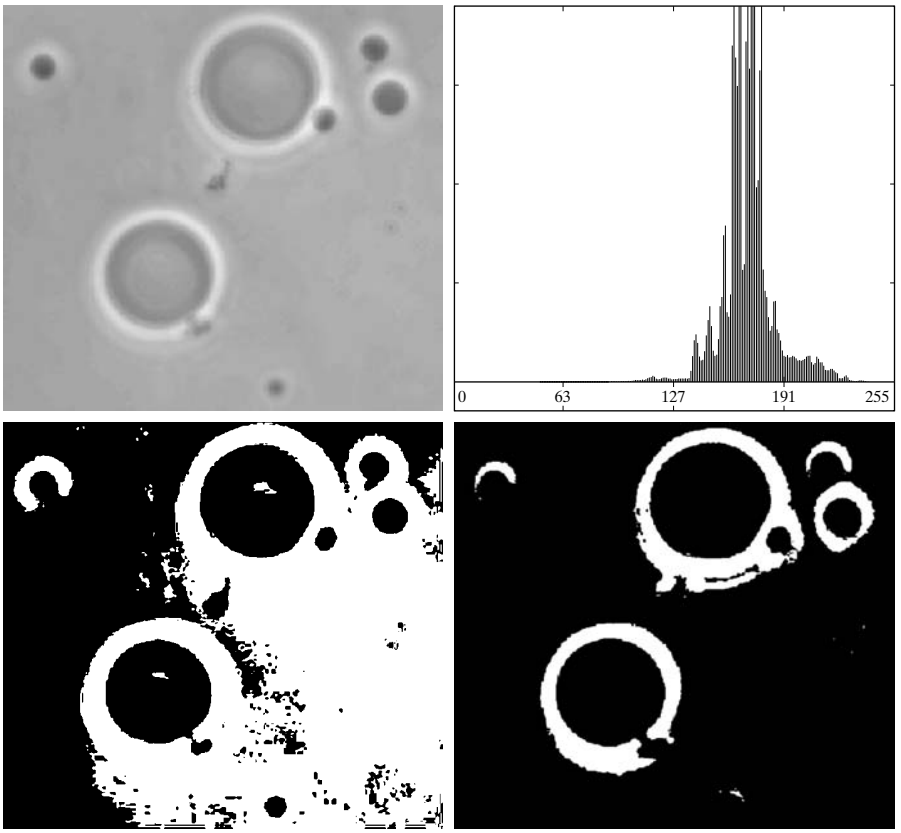
Polymersomes are cells artificially engineered using polymers. Polymorsomes are invisible to the human immune system and can be used, for example, to deliver medication to targeted regions of the body.

As a point of interest, applying Otsu's method to the fingerprint image in Example 10.15 yielded a threshold of 125 and a separability measure of 0.944. The threshold is identical to the value (rounded to the nearest integer) obtained with the basic algorithm. This is not unexpected, given the nature of the histogram. In fact, the separability measure is high due primarily to the relatively large separation between modes and the deep valley between them. ■

10.3.4 Using Image Smoothing to Improve Global Thresholding

As noted in Fig. 10.36, noise can turn a simple thresholding problem into an unsolvable one. When noise cannot be reduced at the source, and thresholding is the segmentation method of choice, a technique that often enhances performance is to smooth the image prior to thresholding. We illustrate the approach with an example.

Figure 10.40(a) is the image from Fig. 10.36(c), Fig. 10.40(b) shows its histogram, and Fig. 10.40(c) is the image thresholded using Otsu's method. Every black point in the white region and every white point in the black region is a

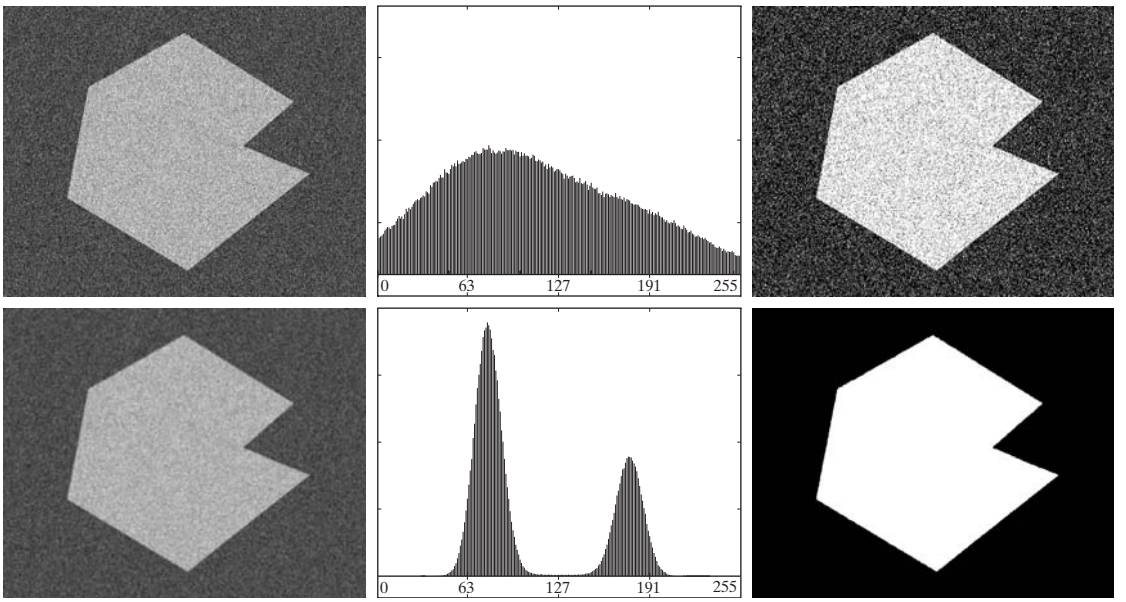


a b
c d

FIGURE 10.39
(a) Original image.
(b) Histogram (high peaks were clipped to highlight details in the lower values).
(c) Segmentation result using the basic global algorithm from Section 10.3.2.
(d) Result obtained using Otsu's method.
(Original image courtesy of Professor Daniel A. Hammer, the University of Pennsylvania.)

thresholding error, so the segmentation was highly unsuccessful. Figure 10.40(d) shows the result of smoothing the noisy image with an averaging mask of size 5×5 (the image is of size 651×814 pixels), and Fig. 10.40(e) is its histogram. The improvement in the shape of the histogram due to smoothing is evident, and we would expect thresholding of the smoothed image to be nearly perfect. As Fig. 10.40(f) shows, this indeed was the case. The slight distortion of the boundary between object and background in the segmented, smoothed image was caused by the blurring of the boundary. In fact, the more aggressively we smooth an image, the more boundary errors we should anticipate in the segmented result.

Next we consider the effect of reducing the size of the region in Fig. 10.40(a) with respect to the background. Figure 10.41(a) shows the result. The noise in this image is additive Gaussian noise with zero mean and a standard deviation of 10 intensity levels (as opposed to 50 in the previous example). As Fig. 10.41(b) shows, the histogram has no clear valley, so we would expect segmentation to fail, a fact that is confirmed by the result in Fig. 10.41(c). Figure 10.41(d) shows the image smoothed with an averaging mask of size 5×5, and Fig. 10.40(e) is the corresponding histogram. As expected, the net effect was to reduce the spread of the histogram, but the distribution still is unimodal. As Fig. 10.40(f) shows, segmentation failed again. The reason for the failure can be traced to the fact that the region is so small that its contribution to the histogram is insignificant compared to the intensity spread caused by noise. In



a b c
d e f

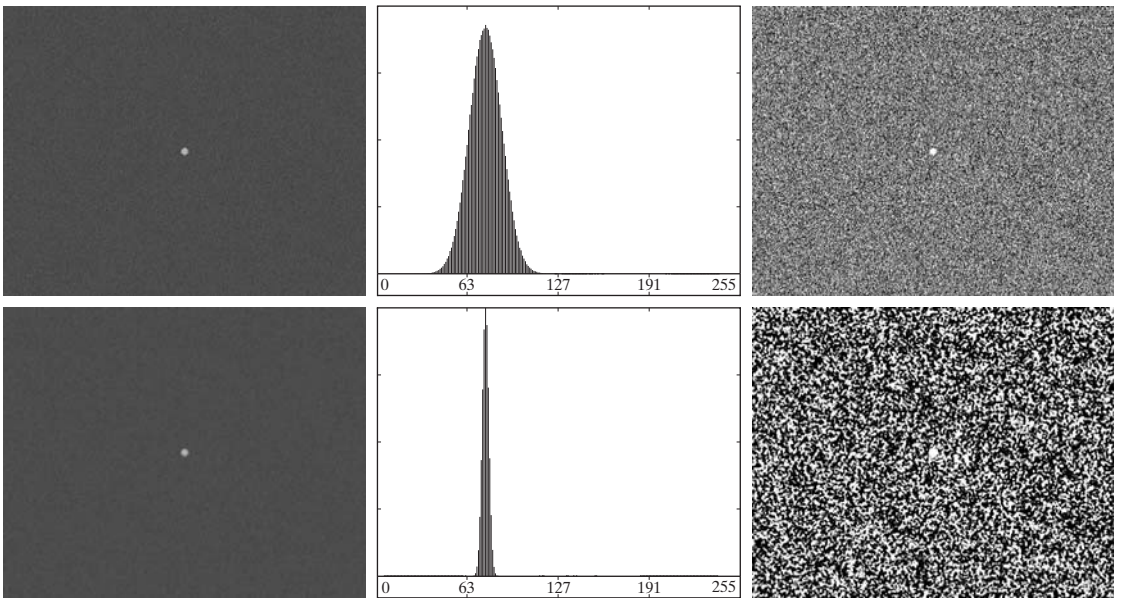
FIGURE 10.40 (a) Noisy image from Fig. 10.36 and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5×5 averaging mask and (e) its histogram. (f) Result of thresholding using Otsu's method.

situations such as this, the approach discussed in the following section is more likely to succeed.

10.3.5 Using Edges to Improve Global Thresholding

Based on the discussion in the previous four sections, we conclude that the chances of selecting a "good" threshold are enhanced considerably if the histogram peaks are tall, narrow, symmetric, and separated by deep valleys. One approach for improving the shape of histograms is to consider only those pixels that lie on or near the edges between objects and the background. An immediate and obvious improvement is that histograms would be less dependent on the relative sizes of objects and the background. For instance, the histogram of an image composed of a small object on a large background area (or vice versa) would be dominated by a large peak because of the high concentration of one type of pixels. We saw in the previous section that this can lead to failure in thresholding.

If only the pixels on or near the edges between objects and background were used, the resulting histogram would have peaks of approximately the same height. In addition, the probability that any of those pixels lies on an object would be approximately equal to the probability that it lies on the background, thus improving the symmetry of the histogram modes. Finally, as indicated in the following paragraph, using pixels that satisfy some simple measures based on gradient and Laplacian operators has a tendency to deepen the valley between histogram peaks.



a b c
d e f

FIGURE 10.41 (a) Noisy image and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5×5 averaging mask and (e) its histogram. (f) Result of thresholding using Otsu's method. Thresholding failed in both cases.

The approach just discussed assumes that the edges between objects and background are known. This information clearly is not available during segmentation, as finding a division between objects and background is precisely what segmentation is all about. However, with reference to the discussion in Section 10.2, an indication of whether a pixel is on an edge may be obtained by computing its gradient or Laplacian. For example, the average value of the Laplacian is 0 at the transition of an edge (see Fig. 10.10), so the valleys of histograms formed from the pixels selected by a Laplacian criterion can be expected to be sparsely populated. This property tends to produce the desirable deep valleys discussed above. In practice, comparable results typically are obtained using either the gradient or Laplacian images, with the latter being favored because it is computationally more attractive and is also an isotropic edge detector.

The preceding discussion is summarized in the following algorithm, where $f(x, y)$ is the input image:

1. Compute an edge image as either the magnitude of the gradient, or absolute value of the Laplacian, of $f(x, y)$ using any of the methods discussed in Section 10.2.
2. Specify a threshold value, T.
3. Threshold the image from Step 1 using the threshold from Step 2 to produce a binary image, $g_T(x, y)$. This image is used as a *mask image* in the following step to select pixels from $f(x, y)$ corresponding to "strong" edge pixels.

It is possible to modify this algorithm so that both the magnitude of the gradient and the absolute value of the Laplacian images are used. In this case, we would specify a threshold for each image and form the logical OR of the two results to obtain the marker image. This approach is useful when more control is desired over the points deemed to be valid edge points.

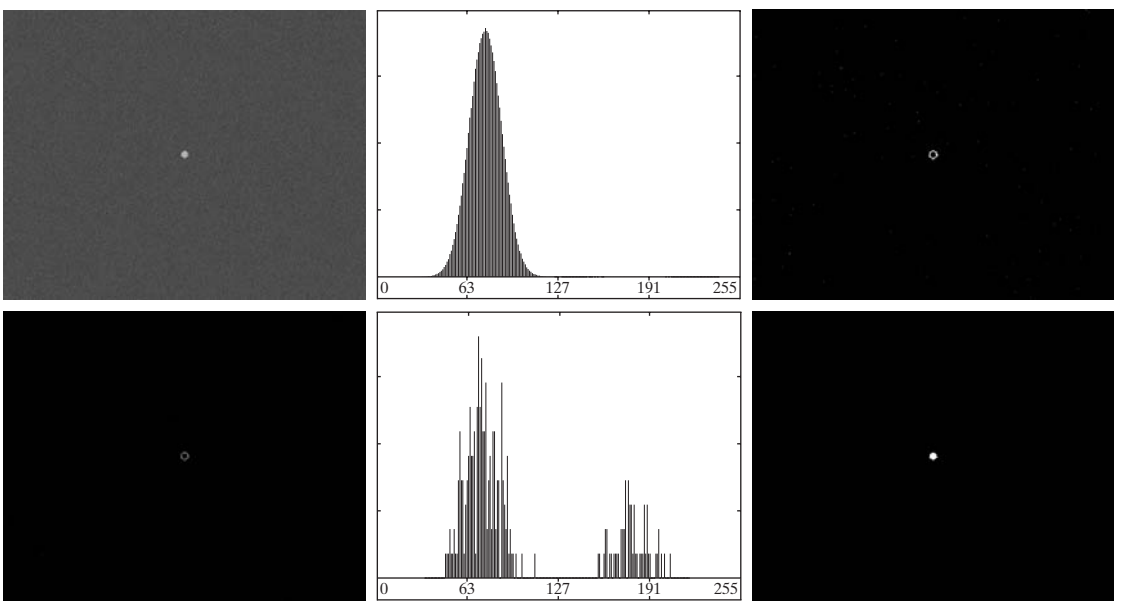
4. Compute a histogram using only the pixels in $f(x, y)$ that correspond to the locations of the 1-valued pixels in $g_T(x, y)$.
5. Use the histogram from Step 4 to segment $f(x, y)$ globally using, for example, Otsu's method.

If T is set to the maximum value of the edge image then, according to Eq. (10.3-1), $g_T(x, y)$ will consist of all 0s, implying that all pixels of $f(x, y)$ will be used to compute the image histogram. In this case, the preceding algorithm becomes global thresholding in which the histogram of the original image is used without modification. It is customary to specify the value of T corresponding to a percentile, which typically is set high (e.g., in the high 90s) so that few pixels in the gradient/Laplacian image will be used in the computation. The following examples illustrate the concepts just discussed. The first example uses the gradient and the second uses the Laplacian. Similar results can be obtained in both examples using either approach. The important issue is to generate a suitable derivative image.

The nth percentile is the smallest number that is greater than n% of the numbers in a given set. For example, if you received a 95 in a test and this score was greater than 85% of all the students taking the test, then you would be in the 85th percentile with respect to the test scores.

EXAMPLE 10.17: Using edge information based on the gradient to improve global thresholding.

■ Figures 10.42(a) and (b) show the image and histogram from Fig. 10.41. You saw that this image could not be segmented by smoothing followed by thresholding. The objective of this example is to solve the problem using edge information. Figure 10.42(c) is the gradient magnitude image thresholded at the



a b c
d e f

FIGURE 10.42 (a) Noisy image from Fig. 10.41(a) and (b) its histogram. (c) Gradient magnitude image thresholded at the 99.7 percentile. (d) Image formed as the product of (a) and (c). (e) Histogram of the nonzero pixels in the image in (d). (f) Result of segmenting image (a) with the Otsu threshold based on the histogram in (e). The threshold was 134, which is approximately midway between the peaks in this histogram.

99.7 percentile. Figure 10.42(d) is the image formed by multiplying this (mask) image by the input image. Figure 10.42(e) is the histogram of the nonzero elements in Fig. 10.42(d). Note that this histogram has the important features discussed earlier; that is, it has reasonably symmetrical modes separated by a deep valley. Thus, while the histogram of the original noisy image offered no hope for successful thresholding, the histogram in Fig. 10.42(e) indicates that thresholding of the small object from the background is indeed possible. The result in Fig. 10.42(f) shows that indeed this is the case. This image was obtained by using Otsu's method to obtain a threshold based on the histogram in Fig. 10.42(e) and then applying this threshold globally to the noisy image in Fig. 10.42(a). The result is nearly perfect. ■

EXAMPLE 10.18: Using edge information based on the Laplacian to improve global thresholding.

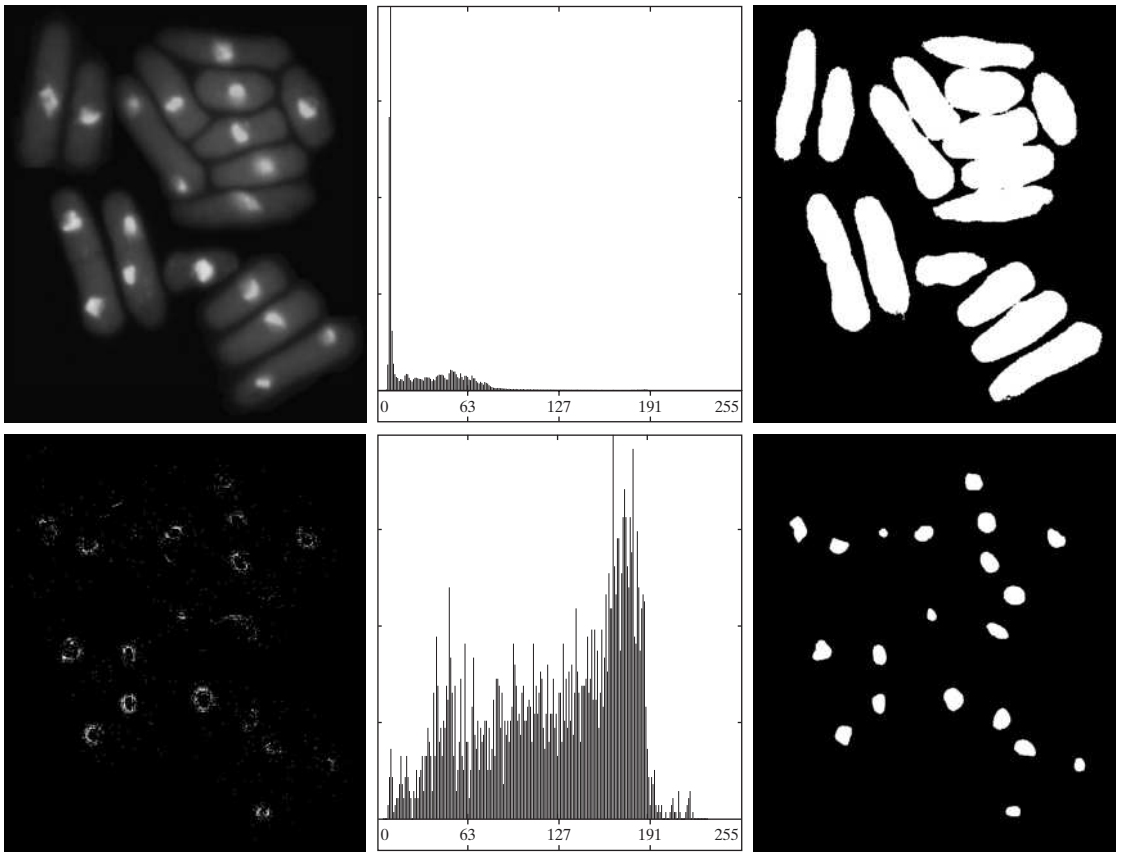
■ In this example we consider a more complex thresholding problem. Figure 10.43(a) shows an 8-bit image of yeast cells in which we wish to use global thresholding to obtain the regions corresponding to the bright spots. As a starting point, Fig. 10.43(b) shows the image histogram, and Fig. 10.43(c) is the result obtained using Otsu's method directly on the image, using the histogram shown. We see that Otsu's method failed to achieve the original objective of detecting the bright spots, and, although the method was able to isolate some of the cell regions themselves, several of the segmented regions on the right are not disjoint. The threshold computed by the Otsu method was 42 and the separability measure was 0.636.

Figure 10.43(d) shows the image $g_T(x, y)$ obtained by computing the absolute value of the Laplacian image and then thresholding it with T set to 115 on an intensity scale in the range $[0, 255]$. This value of T corresponds approximately to the 99.5 percentile of the values in the absolute Laplacian image, so thresholding at this level should result in a sparse set of pixels, as Fig. 10.43(d) shows. Note in this image how the points cluster near the edges of the bright spots, as expected from the preceding discussion. Figure 10.43(e) is the histogram of the nonzero pixels in the product of (a) and (d). Finally, Fig. 10.43(f) shows the result of globally segmenting the original image using Otsu's method based on the histogram in Fig. 10.43(e). This result agrees with the locations of the bright spots in the image. The threshold computed by the Otsu method was 115 and the separability measure was 0.762, both of which are higher than the values obtained by using the original histogram.

By varying the percentile at which the threshold is set we can even improve on the segmentation of the cell regions. For example, Fig. 10.44 shows the result obtained using the same procedure as in the previous paragraph, but with the threshold set at 55, which is approximately 5% of the maximum value of the absolute Laplacian image. This value is at the 53.9 percentile of the values in that image. This result clearly is superior to the result in Fig. 10.43(c) obtained using Otsu's method with the histogram of the original image. ■

10.3.6 Multiple Thresholds

Thus far, we have focused attention on image segmentation using a single global threshold. The thresholding method introduced in Section 10.3.3 can be extended to an arbitrary number of thresholds, because the separability measure



a b c
d e f

FIGURE 10.43 (a) Image of yeast cells. (b) Histogram of (a). (c) Segmentation of (a) with Otsu's method using the histogram in (b). (d) Thresholded absolute Laplacian. (e) Histogram of the nonzero pixels in the product of (a) and (d). (f) Original image thresholded using Otsu's method based on the histogram in (e). (Original image courtesy of Professor Susan L. Forsburg, University of Southern California.)

FIGURE 10.44 Image in Fig. 10.43(a) segmented using the same procedure as explained in Figs. 10.43(d)–(f), but using a lower value to threshold the absolute Laplacian image.

on which it is based also extends to an arbitrary number of classes (Fukunaga [1972]). In the case of K classes, $C_1, C_2, \ldots C_K$, the between-class variance generalizes to the expression

$$\sigma_B^2 = \sum_{k=1}^{K} P_k (m_k - m_G)^2 \tag{10.3-21}$$

where

$$P_k = \sum_{i \in C_k} p_i \tag{10.3-22}$$

$$m_k = \frac{1}{P_k} \sum_{i \in C_k} i p_i \tag{10.3-23}$$

and m_G is the global mean given in Eq. (10.3-9). The K classes are separated by $K - 1$ thresholds whose values, $k_1^*, k_2^*, \ldots, k_{K-1}^*$, are the values that maximize Eq. (10.3-21):

$$\sigma_B^2(k_1^*, k_2^*, \ldots, k_{K-1}^*) = \max_{0<k_1<k_2<\ldots k_{n-1}<L-1} \sigma_B^2(k_1, k_2, \ldots k_{K-1}) \tag{10.3-24}$$

Although this result is perfectly general, it begins to lose meaning as the number of classes increases, because we are dealing with only one variable (intensity). In fact, the between-class variance usually is cast in terms of multiple variables expressed as vectors (Fukunaga [1972]). In practice, using multiple global thresholding is considered a viable approach when there is reason to believe that the problem can be solved effectively with two thresholds. Applications that require more than two thresholds generally are solved using more than just intensity values. Instead, the approach is to use additional descriptors (e.g., color) and the application is cast as a pattern recognition problem, as explained in Section 10.3.8.

Thresholding with two thresholds sometimes is referred to as *hysteresis thresholding*.

For three classes consisting of three intensity intervals (which are separated by two thresholds) the between-class variance is given by:

$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 + P_3(m_3 - m_G)^2 \tag{10.3-25}$$

where

$$\begin{aligned} P_1 &= \sum_{i=0}^{k_1} p_i \\ P_2 &= \sum_{i=k_1+1}^{k_2} p_i \\ P_3 &= \sum_{i=k_2+1}^{L-1} p_i \end{aligned} \tag{10.3-26}$$

and

$$m_1 = \frac{1}{P_1}\sum_{i=0}^{k_1} ip_i$$
$$m_2 = \frac{1}{P_2}\sum_{i=k_1+1}^{k_2} ip_i \tag{10.3-27}$$
$$m_3 = \frac{1}{P_3}\sum_{i=k_2+1}^{L-1} ip_i$$

As in Eqs. (10.3-10) and (10.3-11), the following relationships hold:

$$P_1 m_1 + P_2 m_2 + P_3 m_3 = m_G \tag{10.3-28}$$

and

$$P_1 + P_2 + P_3 = 1 \tag{10.3-29}$$

We see that the P and m terms and, therefore σ_B^2, are functions of k_1 and k_2. The two optimum threshold values, k_1^* and k_2^*, are the values that maximize $\sigma_B^2(k_1, k_2)$. In other words, as in the single-threshold case discussed in Section 10.3.3, we find the optimum thresholds by finding

$$\sigma_B^2(k_1^*, k_2^*) = \max_{0<k_1<k_2<L-1} \sigma_B^2(k_1, k_2) \tag{10.3-30}$$

The procedure starts by selecting the first value of k_1 (that value is 1 because looking for a threshold at 0 intensity makes no sense; also, keep in mind that the increment values are integers because we are dealing with intensities). Next, k_2 is incremented through all its values greater than k_1 and less than $L - 1$ (i.e., $k_2 = k_1 + 1, \ldots, L - 2$). Then k_1 is incremented to its next value and k_2 is incremented again through all its values greater than k_1. This procedure is repeated until $k_1 = L - 3$. The result of this process is a 2-D array, $\sigma_B^2(k_1, k_2)$, and the last step is to look for the maximum value in this array. The values of k_1 and k_2 corresponding to that maximum are the optimum thresholds, k_1^* and k_2^*. If there are several maxima, the corresponding values of k_1 and k_2 are averaged to obtain the final thresholds. The thresholded image is then given by

$$g(x, y) = \begin{cases} a & \text{if } f(x, y) \le k_1^* \\ b & \text{if } k_1^* < f(x, y) \le k_2^* \\ c & \text{if } f(x, y) > k_2^* \end{cases} \tag{10.3-31}$$

where a, b, and c are any three valid intensity values.

Finally, we note that the separability measure defined in Section 10.3.3 for one threshold extends directly to multiple thresholds:

$$\eta(k_1^*, k_2^*) = \frac{\sigma_B^2(k_1^*, k_2^*)}{\sigma_G^2} \tag{10.3-32}$$

where σ_G^2 is the total image variance from Eq. (10.3-13).

EXAMPLE 10.19: Multiple global thresholding.

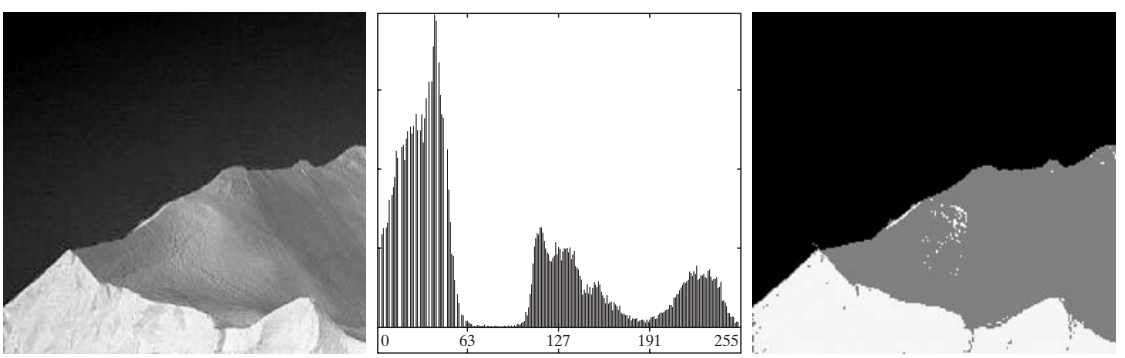
■ Figure 10.45(a) shows an image of an iceberg. The objective of this example is to segment the image into three regions: the dark background, the illuminated area of the iceberg, and the area in shadows. It is evident from the image histogram in Fig. 10.45(b) that two thresholds are required to solve this problem. The procedure discussed above resulted in the thresholds $k_1^* = 80$ and $k_2^* = 177$, which we note from Fig. 10.45(b) are near the centers of the two histogram valleys. Figure 10.45(c) is the segmentation that resulted using these two thresholds in Eq. (10.3-31). The separability measure was 0.954. The principal reason this example worked out so well can be traced to the histogram having three distinct modes separated by reasonably wide, deep valleys. ■

10.3.7 Variable Thresholding

As discussed in Section 10.3.1, factors such as noise and nonuniform illumination play a major role in the performance of a thresholding algorithm. We showed in Sections 10.3.4 and 10.3.5 that image smoothing and using edge information can help significantly. However, it frequently is the case that this type of preprocessing is either impractical or simply ineffective in improving the situation to the point where the problem is solvable by any of the methods discussed thus far. In such situations, the next level of thresholding complexity involves variable thresholding. In this section, we discuss various techniques for choosing variable thresholds.

Image partitioning

One of the simplest approaches to variable thresholding is to subdivide an image into nonoverlapping rectangles. This approach is used to compensate for non-uniformities in illumination and/or reflectance. The rectangles are chosen small enough so that the illumination of each is approximately uniform. We illustrate this approach with an example.



a b c

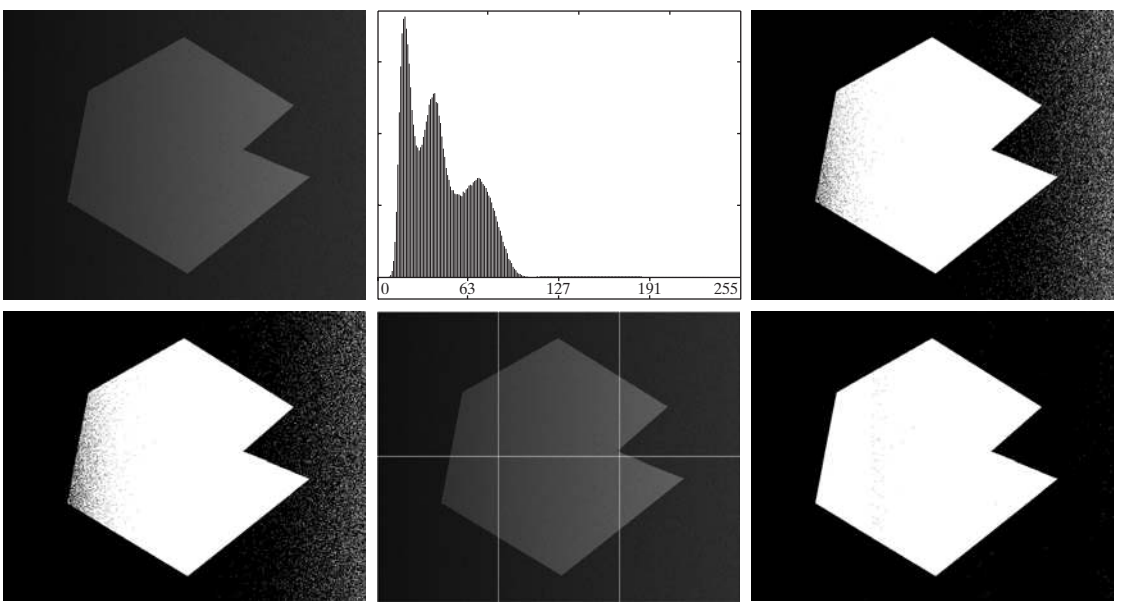
FIGURE 10.45 (a) Image of iceberg. (b) Histogram. (c) Image segmented into three regions using dual Otsu thresholds. (Original image courtesy of NOAA.)

EXAMPLE 10.20: Variable thresholding via image partitioning.

■ Figure 10.46(a) shows the image from Fig. 10.37(c), and Fig. 10.46(b) shows its histogram. When discussing Fig. 10.37(c) we concluded that this image could not be segmented with a global threshold, a fact confirmed by Figs. 10.46(c) and (d), which show the results of segmenting the image using the iterative scheme discussed in Section 10.3.2 and Otsu's method, respectively. Both methods produced comparable results, in which numerous segmentation errors are visible.

Figure 10.46(e) shows the original image subdivided into six rectangular regions, and Fig. 10.46(f) is the result of applying Otsu's global method to each subimage. Although some errors in segmentation are visible, image subdivision produced a reasonable result on an image that is quite difficult to segment. The reason for the improvement is explained easily by analyzing the histogram of each subimage. As Fig. 10.47 shows, each subimage is characterized by a bimodal histogram with a deep valley between the modes, a fact that we know will lead to effective global thresholding.

Image subdivision generally works well when the objects of interest and the background occupy regions of reasonably comparable size, as in Fig. 10.46. When this is not the case, the method typically fails because of the likelihood of subdivisions containing only object or background pixels. Although this situation can be addressed by using additional techniques to determine when a subdivision contains both types of pixels, the logic required to address different



a b c
d e f

FIGURE 10.46 (a) Noisy, shaded image and (b) its histogram. (c) Segmentation of (a) using the iterative global algorithm from Section 10.3.2. (d) Result obtained using Otsu's method. (e) Image subdivided into six subimages. (f) Result of applying Otsu's method to each subimage individually.

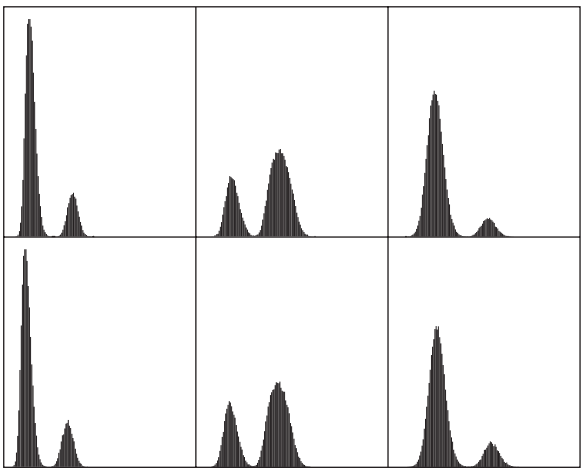


FIGURE 10.47 Histograms of the six subimages in Fig. 10.46(e).

scenarios can get complicated. In such situations, methods such as those discussed in the remainder of this section typically are preferable. ■

Variable thresholding based on local image properties

A more general approach than the image subdivision method discussed in the previous section is to compute a threshold at every point, (x, y), in the image based on one or more specified properties computed in a neighborhood of (x, y). Although this may seem like a laborious process, modern algorithms and hardware allow for fast neighborhood processing, especially for common functions such as logical and arithmetic operations.

We illustrate the basic approach to local thresholding using the standard deviation and mean of the pixels in a neighborhood of every point in an image. These two quantities are quite useful for determining local thresholds because they are descriptors of local contrast and average intensity. Let σ_{xy} and m_{xy} denote the standard deviation and mean value of the set of pixels contained in a neighborhood, S_{xy}, centered at coordinates (x, y) in an image (see Section 3.3.4 regarding computation of the local mean and standard deviation). The following are common forms of variable, local thresholds:

$$T_{xy} = a\sigma_{xy} + bm_{xy} \tag{10.3-33}$$

where a and b are nonnegative constants, and

$$T_{xy} = a\sigma_{xy} + bm_G \tag{10.3-34}$$

where m_G is the global image mean. The segmented image is computed as

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T_{xy} \\ 0 & \text{if } f(x, y) \leq T_{xy} \end{cases} \tag{10.3-35}$$

where $f(x, y)$ is the input image. This equation is evaluated for all pixel locations in the image, and a different threshold is computed at each location (x, y) using the pixels in the neighborhood S_{xy}.

Significant power (with a modest increase in computation) can be added to local thresholding by using predicates based on the parameters computed in the neighborhoods of (x, y):

$$g(x, y) = \begin{cases} 1 & \text{if } Q(\text{local parameters}) \text{ is true} \\ 0 & \text{if } Q(\text{local parameters}) \text{ is false} \end{cases} \tag{10.3-36}$$

where Q is a *predicate* based on parameters computed using the pixels in neighborhood S_{xy}. For example, consider the following predicate, $Q(\sigma_{xy}, m_{xy})$, based on the local mean and standard deviation:

$$Q(\sigma_{xy}, m_{xy}) = \begin{cases} \text{true} & \text{if } f(x, y) > a\sigma_{xy} \text{ AND } f(x, y) > bm_{xy} \\ \text{false} & \text{otherwise} \end{cases} \tag{10.3-37}$$

Note that Eq. (10.3-35) is a special case of Eq. (10.3-36), obtained by letting Q be true if $f(x, y) > T_{xy}$ and false otherwise. In this case, the predicate is based simply on the intensity at a point.

EXAMPLE 10.21: Variable thresholding based on local image properties.

■ Figure 10.48(a) shows the yeast image from Example 10.18. This image has three predominant intensity levels, so it is reasonable to assume that perhaps dual thresholding could be a good segmentation approach. Figure 10.48(b) is the result of using the dual thresholding method explained in Section 10.3.6. As the figure shows, it was possible to isolate the bright areas from the background, but the mid-gray regions on the right side of the image were not segmented properly (recall that we encountered a similar problem with Fig. 10.43(c) in Example 10.18). To illustrate the use of local thresholding, we computed the local standard deviation σ_{xy} for all (x, y) in the input image using a neighborhood of size 3×3. Figure 10.48(c) shows the result. Note how the faint outer lines correctly delineate the boundaries of the cells. Next, we formed a predicate of the form shown in Eq. (10.3-37) but using the global mean instead of m_{xy}. Choosing the global mean generally gives better results when the background is nearly constant and all the object intensities are above or below the background intensity. The values $a = 30$ and $b = 1.5$ were used in completing the specification of the predicate (these values were determined experimentally, as is usually the case in applications such as this). The image was then segmented using Eq. (10.3-36). As Fig. 10.48(d) shows, the result agrees quite closely with the two types of intensity regions prevalent in the input image. Note in particular that all the outer regions were segmented properly and that most of the inner, brighter regions were isolated correctly. ■

Using moving averages

A special case of the local thresholding method just discussed is based on computing a moving average along scan lines of an image. This implementation is quite useful in document processing, where speed is a fundamental requirement. The scanning typically is carried out line by line in a zigzag pattern to

a b
c d

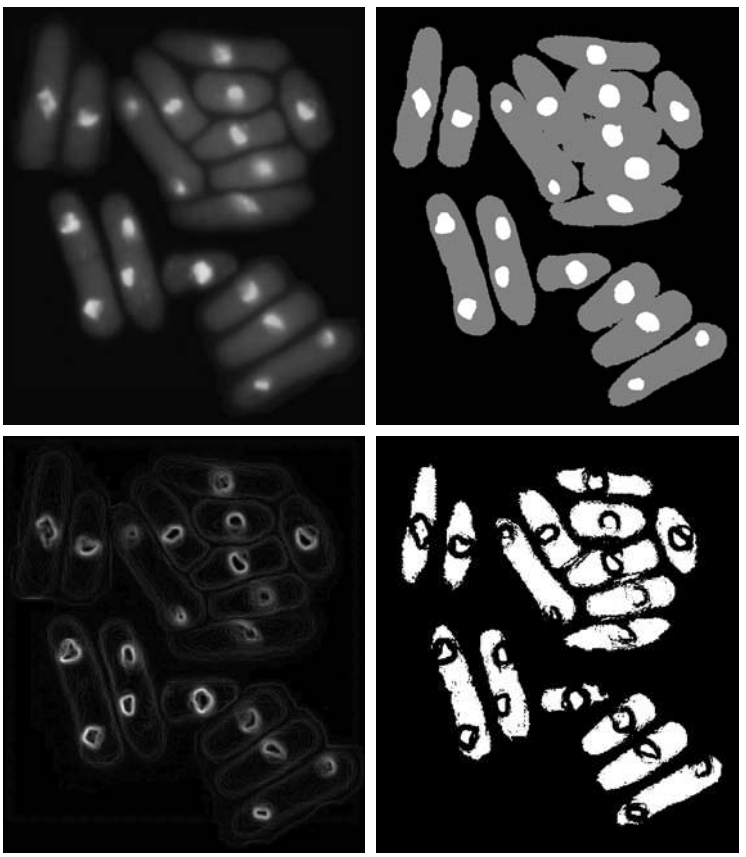


FIGURE 10.48
(a) Image from Fig. 10.43. (b) Image segmented using the dual thresholding approach discussed in Section 10.3.6. (c) Image of local standard deviations. (d) Result obtained using local thresholding.

reduce illumination bias. Let z_{k+1} denote the intensity of the point encountered in the scanning sequence at step $k + 1$. The moving average (mean intensity) at this new point is given by

$$\begin{aligned} m(k+1) &= \frac{1}{n}\sum_{i=k+2-n}^{k+1} z_i \\ &= m(k) + \frac{1}{n}(z_{k+1} - z_{k-n}) \end{aligned} \tag{10.3-38}$$

The first expression is valid for $k \geq n - 1$. When k is less than $n - 1$, averages are formed with the available points. Similarly, the second expression is valid for $k \geq n + 1$.

where n denotes the number of points used in computing the average and $m(1) = z_1/n$. This initial value is not strictly correct because the average of a single point is the value of the point itself. However, we use $m(1) = z_1/n$ so that no special computations are required when Eq. (10.3-38) first starts up. Another way of viewing it is that this is the value we would obtain if the border of the image were padded with $n - 1$ zeros. The algorithm is initialized only once, not at every row. Because a moving average is computed for every point in the image, segmentation is implemented using Eq. (10.3-35) with $T_{xy} = bm_{xy}$ where b is constant and m_{xy} is the moving average from Eq. (10.3-38) at point (x, y) in the input image.

EXAMPLE 10.22: Document thresholding using moving averages.

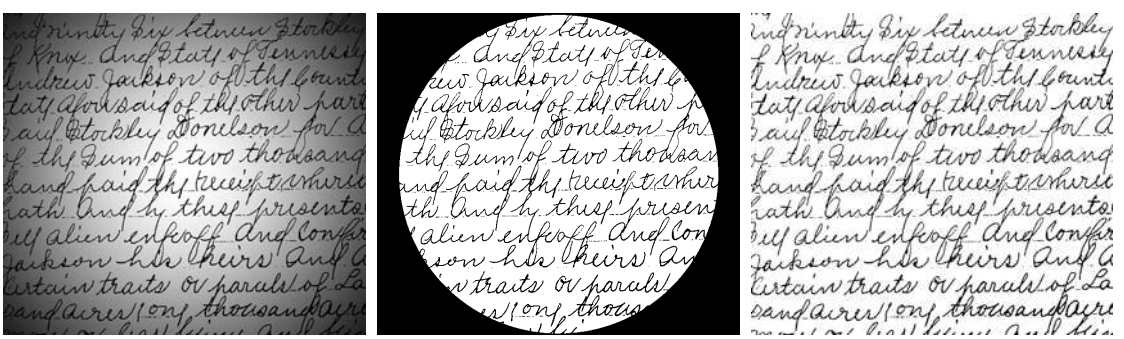
■ Figure 10.49(a) shows an image of handwritten text shaded by a spot intensity pattern. This form of intensity shading is typical of images obtained with a photographic flash. Figure 10.49(b) is the result of segmentation using the Otsu global thresholding method. It is not unexpected that global thresholding could not overcome the intensity variation. Figure 10.49(c) shows successful segmentation with local thresholding using moving averages. A rule of thumb is to let n equal 5 times the average stroke width. In this case, the average width was 4 pixels, so we let $n = 20$ in Eq. (10.3-38) and used $b = 0.5$.

As another illustration of the effectiveness of this segmentation approach we used the same parameters as in the previous paragraph to segment the image in Fig. 10.50(a), which is corrupted by a sinusoidal intensity variation typical of the variation that may occur when the power supply in a document scanner is not grounded properly. As Figs. 10.50(b) and (c) show, the segmentation results are comparable to those in Fig. 10.49.

It is of interest to note that successful segmentation results were obtained in both cases using the same values for n and b, which shows the relative ruggedness of the approach. In general, thresholding based on moving averages works well when the objects of interest are small (or thin) with respect to the image size, a condition satisfied by images of typed or handwritten text. ■

10.3.8 Multivariable Thresholding

Thus far, we have been concerned with thresholding based on a single variable: gray-scale intensity. In some cases, a sensor can make available more than one variable to characterize each pixel in an image, and thus allow *multivariable thresholding*. A notable example is color imaging, where red (R), green (G), and blue (B) components are used to form a composite color image (see Chapter 6). In this case, each "pixel" is characterized by three values, and can be represented as a 3-D vector, $\mathbf{z} = (z_1, z_2, z_3)^T$, whose components are the RGB colors at a point. These 3-D points often are referred to as *voxels*, to denote *volumetric* elements, as opposed to *image* elements.



a b c

FIGURE 10.49 (a) Text image corrupted by spot shading. (b) Result of global thresholding using Otsu's method. (c) Result of local thresholding using moving averages.

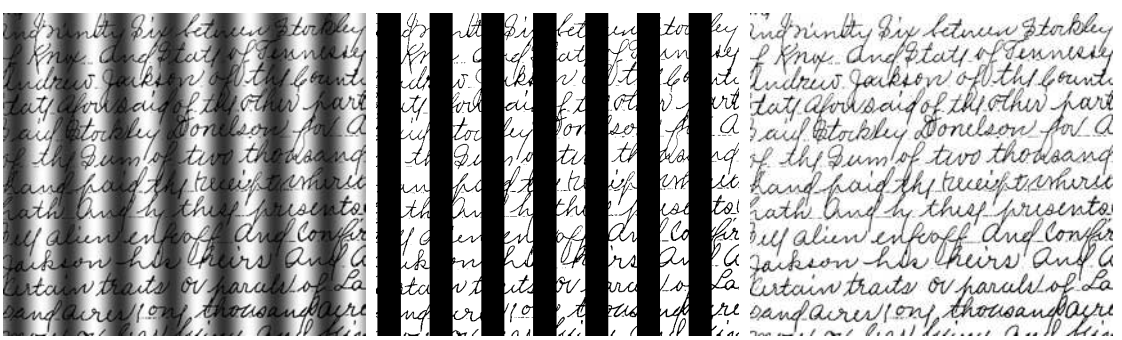
As discussed in some detail in Section 6.7, multivariable thresholding may be viewed as a distance computation. Suppose that we want to extract from a color image all regions having a specified color range: say, reddish hues. Let **a** denote the average reddish color in which we are interested. One way to segment a color image based on this parameter is to compute a distance measure, $D(\mathbf{z}, \mathbf{a})$, between an arbitrary color point, $\mathbf{z}$, and the average color, $\mathbf{a}$. Then, we segment the input image as follows:

$$g = \begin{cases} 1 & \text{if } D(\mathbf{z}, \mathbf{a}) < T \\ 0 & \text{otherwise} \end{cases} \tag{10.3-39}$$

where T is a threshold, and it is understood that the distance computation is performed at all coordinates in the input image to generate the corresponding segmented values in g. Note that the inequalities in this equation are the opposite of the inequalities we used in Eq. (10.3-1) for thresholding a single variable. The reason is that the equation $D(\mathbf{z}, \mathbf{a}) = T$ defines a volume (see Fig. 6.43) and it is more intuitive to think of segmented pixel values as being contained within the volume and background pixel values as being on the surface or outside the volume. Equation (10.3-39) reduces to Eq. (10.3-1) by letting $D(\mathbf{z}, \mathbf{a}) = -f(x, y)$.

Observe that the condition $f(x, y) > T$ basically says that the Euclidean distance between the value of f and the origin of the real line exceeds the value of T. Thus, thresholding is based on the computation of a distance measure, and the form of Eq. (10.3-39) depends on the measure used. If, in general, $\mathbf{z}$ in an n-dimensional vector, we know from Section 2.6.6 that the n-dimensional *Euclidean distance* is defined as

$$\begin{aligned} D(\mathbf{z}, \mathbf{a}) &= \|\mathbf{z} - \mathbf{a}\| \\ &= \left[(\mathbf{z} - \mathbf{a})^T(\mathbf{z} - \mathbf{a})\right]^{\frac{1}{2}} \end{aligned} \tag{10.3-40}$$



a b c

FIGURE 10.50 (a) Text image corrupted by sinusoidal shading. (b) Result of global thresholding using Otsu's method. (c) Result of local thresholding using moving averages.

The equation $D(\mathbf{z}, \mathbf{a}) = T$ describes a sphere (called a *hypersphere*) in n-dimensional Euclidean space (Fig. 6.43 shows a 3-D example). A more powerful distance measure is the so-called *Mahalanobis distance*, defined as

$$D(\mathbf{z}, \mathbf{a}) = \left[(\mathbf{z} - \mathbf{a})^T \mathbf{C}^{-1} (\mathbf{z} - \mathbf{a})\right]^{\frac{1}{2}} \tag{10.3-41}$$

where $\mathbf{C}$ is the covariance matrix of the $\mathbf{z}$s, as discussed Section 12.2.2. $D(\mathbf{z}, \mathbf{a}) = T$ describes an n-dimensional hyperellipse (Fig. 6.43 shows a 3-D example). This expression reduces to Eq. (10.3-40) when $\mathbf{C} = \mathbf{I}$, the identity matrix.

We gave a detailed example in Section 6.7 regarding the use of these expressions. We also discuss in Section 12.2 the problem of segmenting regions out of an image using pattern recognition techniques based on decision functions, which may be viewed as a multiclass, multivariable thresholding problem.

10.4 Region-Based Segmentation

As discussed in Section 10.1, the objective of segmentation is to partition an image into regions. In Section 10.2, we approached this problem by attempting to find boundaries between regions based on discontinuities in intensity levels, whereas in Section 10.3, segmentation was accomplished via thresholds based on the distribution of pixel properties, such as intensity values or color. In this section, we discuss segmentation techniques that are based on finding the regions directly.

You should review the terminology introduced in Section 10.1 before proceeding.

10.4.1 Region Growing

As its name implies, *region growing* is a procedure that groups pixels or subregions into larger regions based on predefined criteria for growth. The basic approach is to start with a set of "seed" points and from these grow regions by appending to each seed those neighboring pixels that have predefined properties similar to the seed (such as specific ranges of intensity or color).

Selecting a set of one or more starting points often can be based on the nature of the problem, as shown later in Example 10.23. When a priori information is not available, the procedure is to compute at every pixel the same set of properties that ultimately will be used to assign pixels to regions during the growing process. If the result of these computations shows clusters of values, the pixels whose properties place them near the centroid of these clusters can be used as seeds.

The selection of similarity criteria depends not only on the problem under consideration, but also on the type of image data available. For example, the analysis of land-use satellite imagery depends heavily on the use of color. This problem would be significantly more difficult, or even impossible, to solve without the inherent information available in color images. When the images are monochrome, region analysis must be carried out with a set of descriptors based on intensity levels and spatial properties (such as moments or texture). We discuss descriptors useful for region characterization in Chapter 11.

Descriptors alone can yield misleading results if connectivity properties are not used in the region-growing process. For example, visualize a random arrangement of pixels with only three distinct intensity values. Grouping pixels with the same intensity level to form a "region" without paying attention to connectivity would yield a segmentation result that is meaningless in the context of this discussion.

Another problem in region growing is the formulation of a stopping rule. Region growth should stop when no more pixels satisfy the criteria for inclusion in that region. Criteria such as intensity values, texture, and color are local in nature and do not take into account the "history" of region growth. Additional criteria that increase the power of a region-growing algorithm utilize the concept of size, likeness between a candidate pixel and the pixels grown so far (such as a comparison of the intensity of a candidate and the average intensity of the grown region), and the shape of the region being grown. The use of these types of descriptors is based on the assumption that a model of expected results is at least partially available.

Let: $f(x, y)$ denote an input image array; $S(x, y)$ denote a *seed* array containing 1s at the locations of seed points and 0s elsewhere; and Q denote a predicate to be applied at each location (x, y). Arrays f and S are assumed to be of the same size. A basic region-growing algorithm based on 8-connectivity may be stated as follows.

See Sections 2.5.2 and 9.5.3 regarding connected components, and Section 9.2.1 regarding erosion.

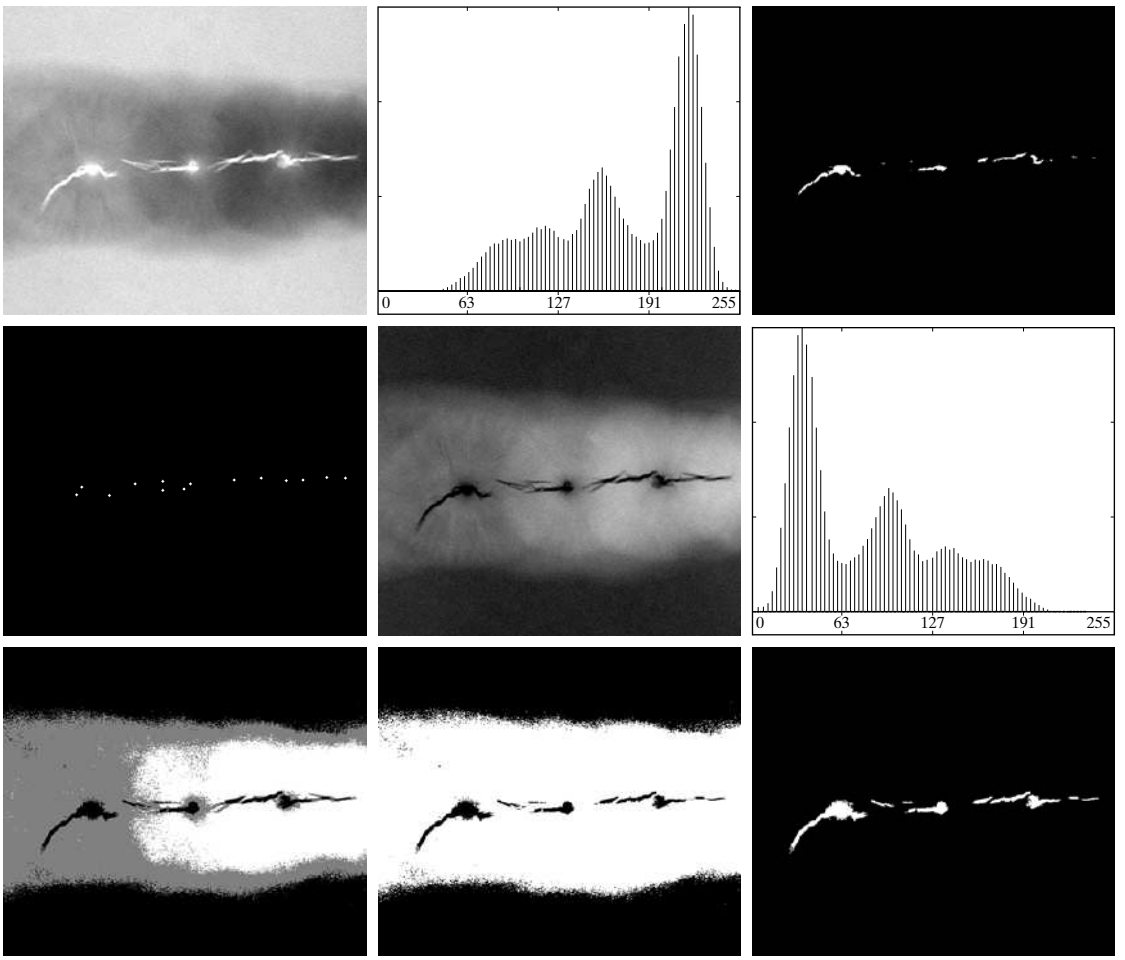
1. Find all connected components in $S(x, y)$ and erode each connected component to one pixel; label all such pixels found as 1. All other pixels in S are labeled 0.
2. Form an image f_Q such that, at a pair of coordinates (x, y), let $f_Q(x, y) = 1$ if the input image satisfies the given predicate, Q, at those coordinates; otherwise, let $f_Q(x, y) = 0$.
3. Let g be an image formed by appending to each seed point in S all the 1-valued points in f_Q that are 8-connected to that seed point.
4. Label each connected component in g with a different region label (e.g., $1, 2, 3, \ldots$). This is the segmented image obtained by region growing.

We illustrate the mechanics of this algorithm by an example.

EXAMPLE 10.23: Segmentation by region growing.

■ Figure 10.51(a) shows an 8-bit X-ray image of a weld (the horizontal dark region) containing several cracks and porosities (the bright regions running horizontally through the center of the image). We illustrate the use of region growing by segmenting the defective weld regions. These regions could be used in applications such as weld inspection, for inclusion in a database of historical studies, or for controlling an automated welding system.

The first order of business is to determine the seed points. From the physics of the problem, we know that cracks and porosities will attenuate X-rays considerably less than solid welds, so we expect the regions containing these types of defects to be significantly brighter than other parts of the X-ray image. We can extract the seed points by thresholding the original image, using a threshold set at a high percentile. Figure 10.51(b) shows the histogram of the image



a b c
d e f
g h i

FIGURE 10.51 (a) X-ray image of a defective weld. (b) Histogram. (c) Initial seed image. (d) Final seed image (the points were enlarged for clarity). (e) Absolute value of the difference between (a) and (c). (f) Histogram of (e). (g) Difference image thresholded using dual thresholds. (h) Difference image thresholded with the smallest of the dual thresholds. (i) Segmentation result obtained by region growing. (Original image courtesy of X-TEK Systems, Ltd.)

and Fig. 10.51(c) shows the thresholded result obtained with a threshold equal to the 99.9 percentile of intensity values in the image, which in this case was 254 (see Section 10.3.5 regarding percentiles). Figure 10.51(d) shows the result of morphologically eroding each connected component in Fig. 10.51(c) to a single point.

Next, we have to specify a predicate. In this example, we are interested in appending to each seed all the pixels that (a) are 8-connected to that seed and

(b) are "similar" to it. Using intensity differences as a measure of similarity, our predicate applied at each location (x, y) is

$$Q = \begin{cases} \text{TRUE} & \text{if the absolute difference of the intensities between the seed and the pixel at } (x, y) \text{ is } \leq T \\ \text{FALSE} & \text{otherwise} \end{cases}$$

where T is a specified threshold. Although this predicate is based on intensity differences and uses a single threshold, we could specify more complex schemes in which a different threshold is applied to each pixel, and properties other than differences are used. In this case, the preceding predicate is sufficient to solve the problem, as the rest of this example shows.

From the previous paragraph, we know that the smallest seed value is 255 because the image was thresholded with a threshold of 254. Figure 10.51(e) shows the absolute value of the difference between the images in Figs. 10.51(a) and (c). The image in Fig. 10.51(e) contains all the differences needed to compute the predicate at each location (x, y). Figure 10.51(f) shows the corresponding histogram. We need a threshold to use in the predicate to establish similarity. The histogram has three principal modes, so we can start by applying to the difference image the dual thresholding technique discussed in Section 10.3.6. The resulting two thresholds in this case were $T_1 = 68$ and $T_2 = 126$, which we see correspond closely to the valleys of the histogram. (As a brief digression, we segmented the image using these two thresholds. The result in Fig. 10.51(g) shows that the problem of segmenting the defects cannot be solved using dual thresholds, even though the thresholds are in the main valleys.)

Figure 10.51(h) shows the result of thresholding the difference image with only T_1. The black points are the pixels for which the predicate was TRUE; the others failed the predicate. The important result here is that the points in the good regions of the weld failed the predicate, so they will not be included in the final result. The points in the outer region will be considered by the region-growing algorithm as candidates. However, Step 3 will reject the outer points, because they are not 8-connected to the seeds. In fact, as Fig. 10.51(i) shows, this step resulted in the correct segmentation, indicating that the use of connectivity was a fundamental requirement in this case. Finally, note that in Step 4 we used the same value for all the regions found by the algorithm. In this case, it was visually preferable to do so. ■

10.4.2 Region Splitting and Merging

The procedure discussed in the last section grows regions from a set of seed points. An alternative is to subdivide an image initially into a set of arbitrary, disjoint regions and then merge and/or split the regions in an attempt to satisfy the conditions of segmentation stated in Section 10.1. The basics of splitting and merging are discussed next.

Let R represent the entire image region and select a predicate Q. One approach for segmenting R is to subdivide it successively into smaller and smaller quadrant regions so that, for any region R_i, $Q(R_i) = \text{TRUE}$. We start with the entire region. If $Q(R) = \text{FALSE}$, we divide the image into quadrants. If Q is FALSE for any quadrant, we subdivide that quadrant into subquadrants, and so on. This particular splitting technique has a convenient representation in the form of so-called *quadtrees*, that is, trees in which each node has exactly four descendants, as Fig. 10.52 shows (the images corresponding to the nodes of a quadtree sometimes are called *quadregions* or *quadimages*). Note that the root of the tree corresponds to the entire image and that each node corresponds to the subdivision of a node into four descendant nodes. In this case, only R_4 was subdivided further.

If only splitting is used, the final partition normally contains adjacent regions with identical properties. This drawback can be remedied by allowing merging as well as splitting. Satisfying the constraints of segmentation outlined in Section 10.1 requires merging only adjacent regions whose combined pixels satisfy the predicate Q. That is, two adjacent regions R_j and R_k are merged only if $Q(R_j \cup R_k) = \text{TRUE}$.

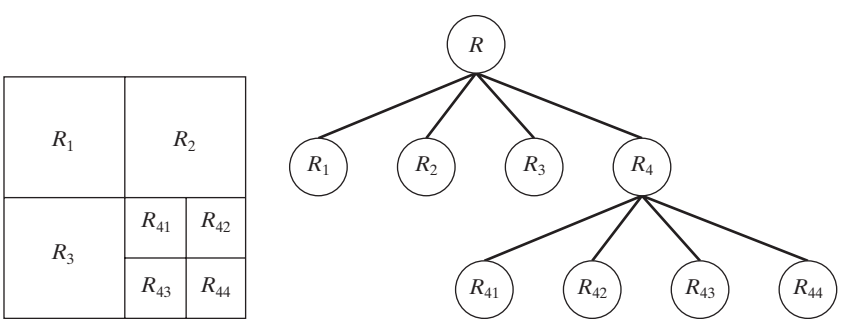
See Section 2.5.2 regarding region adjacency.

The preceding discussion can be summarized by the following procedure in which, at any step, we

1. Split into four disjoint quadrants any region R_i for which $Q(R_i) = \text{FALSE}$.
2. When no further splitting is possible, merge any adjacent regions R_j and R_k for which $Q(R_j \cup R_k) = \text{TRUE}$.
3. Stop when no further merging is possible.

It is customary to specify a minimum quadregion size beyond which no further splitting is carried out.

Numerous variations of the preceding basic theme are possible. For example, a significant simplification results if in Step 2 we allow merging of any two adjacent regions R_i and R_j if each one satisfies the predicate individually. This results in a much simpler (and faster) algorithm, because testing of the predicate is limited to individual quadregions. As the following example shows, this simplification is still capable of yielding good segmentation results.



a b

FIGURE 10.52 (a) Partitioned image. (b) Corresponding quadtree. R represents the entire image region.

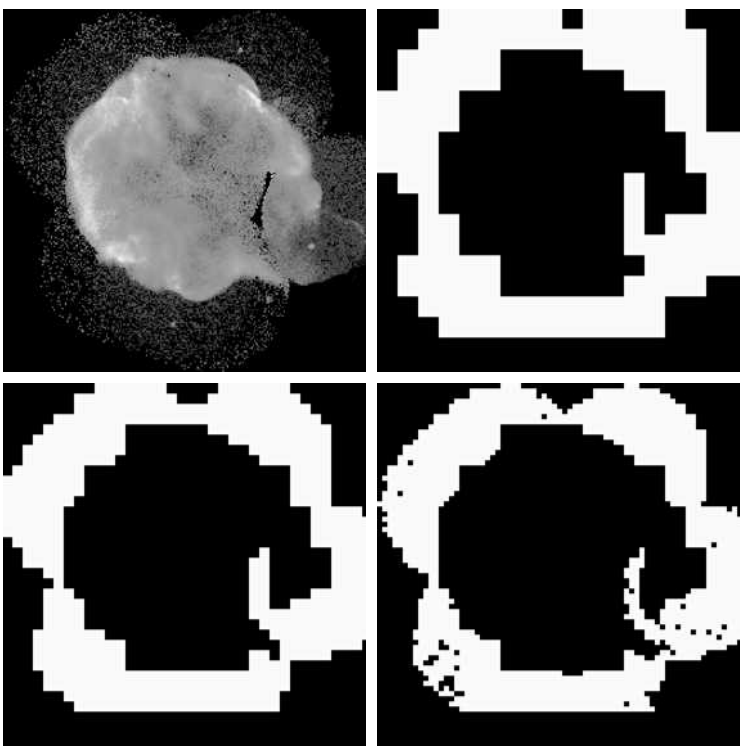
EXAMPLE 10.24: Segmentation by region splitting and merging.

■ Figure 10.53(a) shows a 566 × 566 X-ray band image of the Cygnus Loop. The objective of this example is to segment out of the image the "ring" of less dense matter surrounding the dense center. The region of interest has some obvious characteristics that should help in its segmentation. First, we note that the data in this region has a random nature, indicating that its standard deviation should be greater than the standard deviation of the background (which is near 0) and of the large central region, which is fairly smooth. Similarly, the mean value (average intensity) of a region containing data from the outer ring should be greater than the mean of the darker background and less than the mean of the large, lighter central region. Thus, we should be able to segment the region of interest using the following predicate:

$$Q = \begin{cases} \text{TRUE} & \text{if } \sigma > a \quad \text{AND} \quad 0 < m < b \\ \text{FALSE} & \text{otherwise} \end{cases}$$

where m and σ are the mean and standard deviation of the pixels in a quadregion, and a and b are constants.

Analysis of several regions in the outer area of interest revealed that the mean intensity of pixels in those regions did not exceed 125 and the standard deviation was always greater than 10. Figures 10.53(b) through (d) show the results obtained using these values for a and b, and varying the minimum size allowed for the quadregions from 32 to 8. The pixels in a quadregion whose



a b
c d

FIGURE 10.53 (a) Image of the Cygnus Loop supernova, taken in the X-ray band by NASA's Hubble Telescope. (b)–(d) Results of limiting the smallest allowed quadregion to sizes of 32 × 32, 16 × 16, and 8 × 8 pixels, respectively. (Original image courtesy of NASA.)

pixels satisfied the predicate were set to white; all others in that region were set to black. The best result in terms of capturing the shape of the outer region was obtained using quadregions of size 16×16. The black squares in Fig. 10.53(d) are quadregions of size 8×8 whose pixels did not satisfied the predicate. Using smaller quadregions would result in increasing numbers of such black regions. Using regions larger than the one illustrated here results in a more "block-like" segmentation. Note that in all cases the segmented regions (white pixels) completely separate the inner, smoother region from the background. Thus, the segmentation effectively partitioned the image into three distinct areas that correspond to the three principal features in the image: background, dense, and sparse regions. Using any of the white regions in Fig. 10.53 as a mask would make it a relatively simple task to extract these regions from the original image (Problem 10.40). As in Example 10.23, these results could not have been obtained using edge- or threshold-based segmentation. ■

As used in the preceding example, properties based on the mean and standard deviation of pixel intensities in a region attempt to quantify the *texture* of the region (see Section 11.3.3 for a discussion on texture). The concept of *texture segmentation* is based on using measures of texture in the predicates. In other words, we can perform texture segmentation by any of the methods discussed in this section simply by specifying predicates based on texture content.

10.5 Segmentation Using Morphological Watersheds

Thus far, we have discussed segmentation based on three principal concepts: (a) edge detection, (b) thresholding, and (c) region growing. Each of these approaches was found to have advantages (for example, speed in the case of global thresholding) and disadvantages (for example, the need for post-processing, such as edge linking, in edge-based segmentation). In this section we discuss an approach based on the concept of so-called *morphological watersheds*. As will become evident in the following discussion, segmentation by watersheds embodies many of the concepts of the other three approaches and, as such, often produces more stable segmentation results, including connected segmentation boundaries. This approach also provides a simple framework for incorporating knowledge-based constraints (see Fig. 1.23) in the segmentation process.

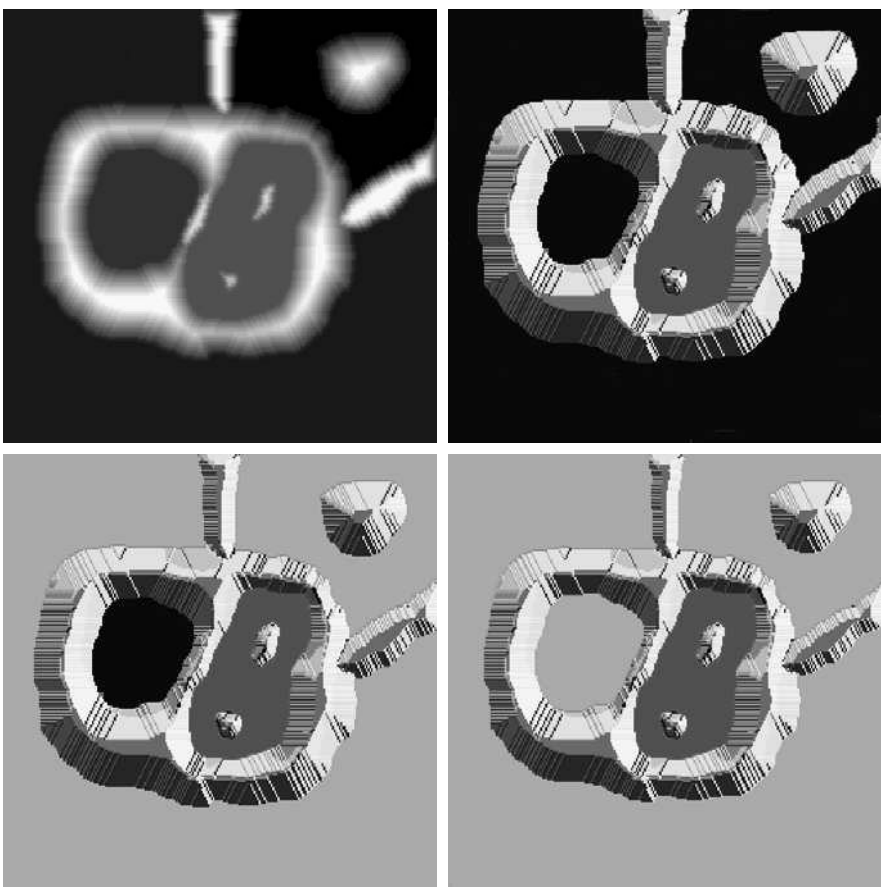
10.5.1 Background

The concept of watersheds is based on visualizing an image in three dimensions: two spatial coordinates versus intensity, as in Fig. 2.18(a). In such a "topographic" interpretation, we consider three types of points: (a) points belonging to a regional minimum; (b) points at which a drop of water, if placed at the location of any of those points, would fall with certainty to a single minimum; and (c) points at which water would be equally likely to fall to more than one such minimum. For a particular regional minimum, the set of points satisfying condition (b) is called the *catchment basin* or *watershed* of that

minimum. The points satisfying condition (c) form crest lines on the topographic surface and are termed *divide lines* or *watershed lines*.

The principal objective of segmentation algorithms based on these concepts is to find the watershed lines. The basic idea is simple, as the following analogy illustrates. Suppose that a hole is punched in each regional minimum and that the entire topography is flooded from below by letting water rise through the holes at a uniform rate. When the rising water in distinct catchment basins is about to merge, a dam is built to prevent the merging. The flooding will eventually reach a stage when only the tops of the dams are visible above the water line. These dam boundaries correspond to the divide lines of the watersheds. Therefore, they are the (connected) boundaries extracted by a watershed segmentation algorithm.

These ideas can be explained further with the aid of Fig. 10.54. Figure 10.54(a) shows a gray-scale image and Fig. 10.54(b) is a topographic view, in which the height of the "mountains" is proportional to intensity values in the input image. For ease of interpretation, the backsides of structures are shaded. This is not to be confused with intensity values; only the general topography of the three-dimensional representation is of interest. In order to prevent the rising water from spilling out through the edges of the image, we imagine the

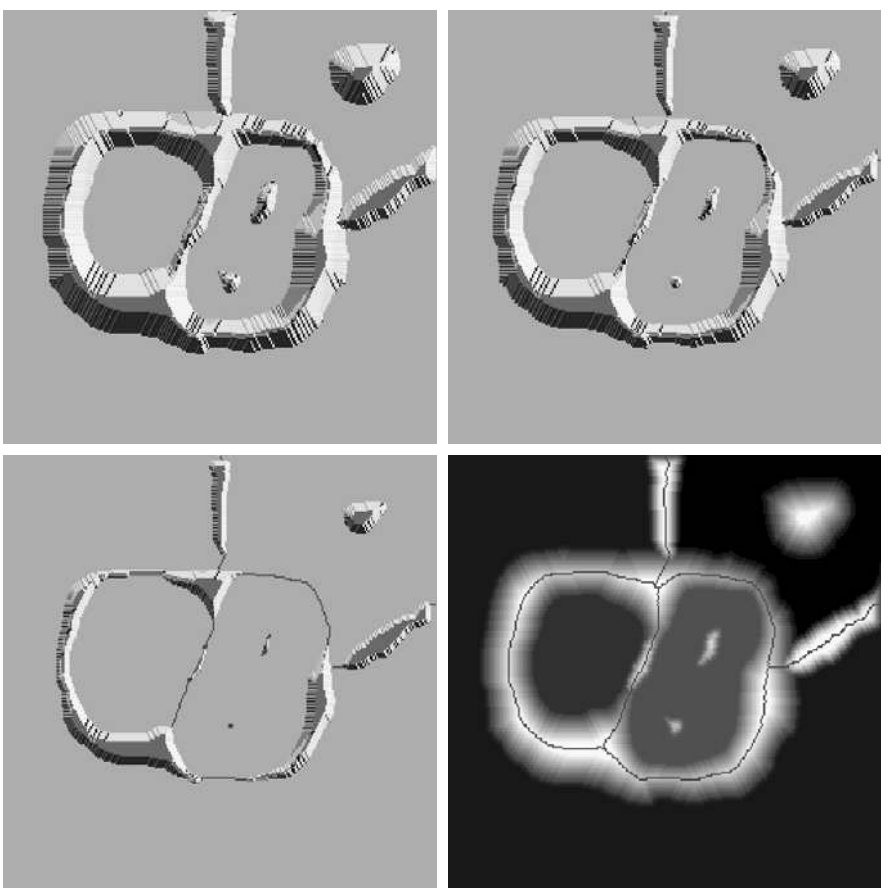


a b
c d

FIGURE 10.54
(a) Original image. (b) Topographic view. (c)–(d) Two stages of flooding.

perimeter of the entire topography (image) being enclosed by dams of height greater than the highest possible mountain, whose value is determined by the highest possible intensity value in the input image.

Suppose that a hole is punched in each regional minimum [shown as dark areas in Fig. 10.54(b)] and that the entire topography is flooded from below by letting water rise through the holes at a uniform rate. Figure 10.54(c) shows the first stage of flooding, where the "water," shown in light gray, has covered only areas that correspond to the very dark background in the image. In Figs. 10.54(d) and (e) we see that the water now has risen into the first and second catchment basins, respectively. As the water continues to rise, it will eventually overflow from one catchment basin into another. The first indication of this is shown in 10.54(f). Here, water from the left basin actually overflowed into the basin on the right and a short "dam" (consisting of single pixels) was built to prevent water from merging at that level of flooding (the details of dam building are discussed in the following section). The effect is more pronounced as water continues to rise, as shown in Fig. 10.54(g). This figure shows a longer dam between the two catchment basins and another dam in the top part of the right basin. The latter dam was built to prevent merging of water from that basin with water from areas corresponding to the background. This process is continued until the maximum



e f
g h

FIGURE 10.54
(Continued)
(e) Result of further flooding. (f) Beginning of merging of water from two catchment basins (a short dam was built between them). (g) Longer dams. (h) Final watershed (segmentation) lines. (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

level of flooding (corresponding to the highest intensity value in the image) is reached. The final dams correspond to the watershed lines, which are the desired segmentation result. The result for this example is shown in Fig. 10.54(h) as dark, 1-pixel-thick paths superimposed on the original image. Note the important property that the watershed lines form connected paths, thus giving continuous boundaries between regions.

One of the principal applications of watershed segmentation is in the extraction of nearly uniform (bloblike) objects from the background. Regions characterized by small variations in intensity have small gradient values. Thus, in practice, we often see watershed segmentation applied to the gradient of an image, rather than to the image itself. In this formulation, the regional minima of catchment basins correlate nicely with the small value of the gradient corresponding to the objects of interest.

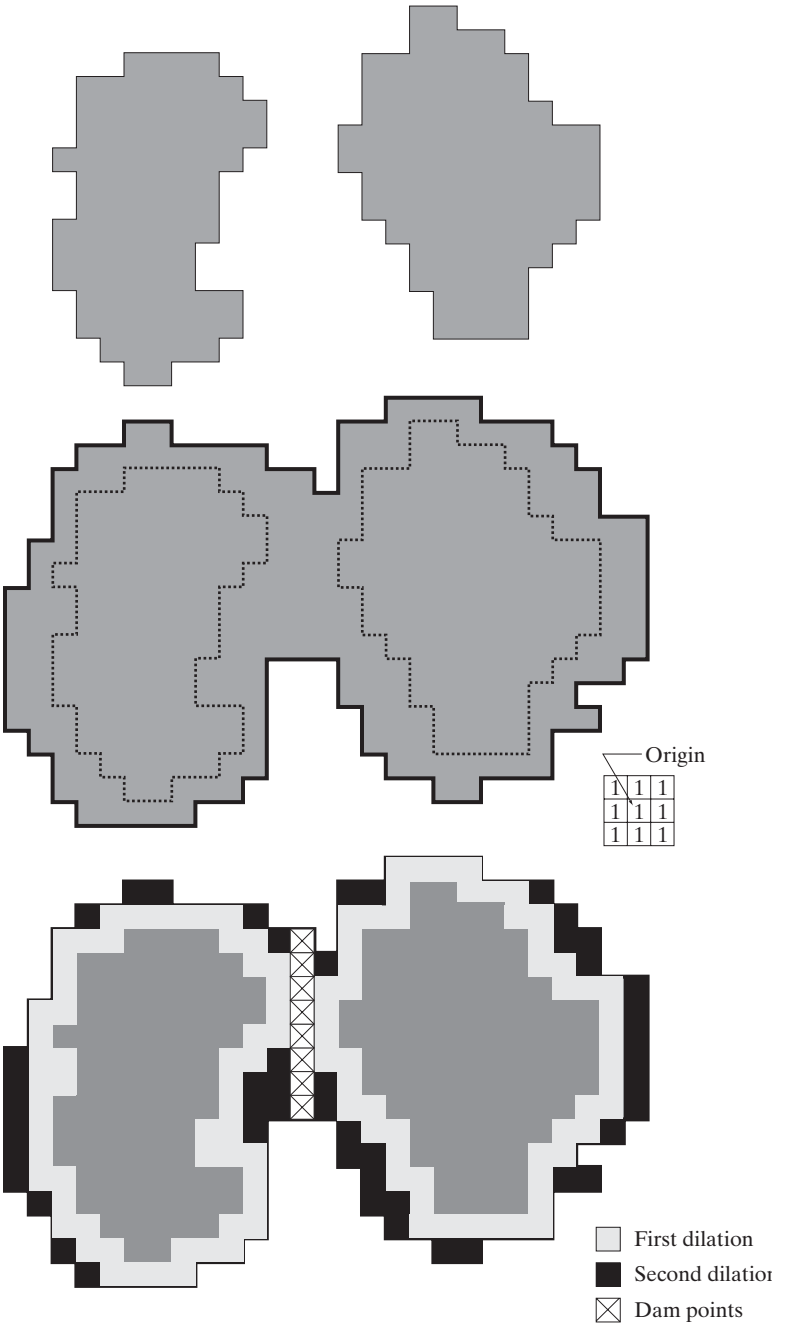
10.5.2 Dam Construction

Before proceeding, let us consider how to construct the dams or watershed lines required by watershed segmentation algorithms. Dam construction is based on binary images, which are members of 2-D integer space Z^2 (see Section 2.4.2). The simplest way to construct dams separating sets of binary points is to use morphological dilation (see Section 9.2.2).

The basics of how to construct dams using dilation are illustrated in Fig. 10.55. Figure 10.55(a) shows portions of two catchment basins at flooding step $n - 1$ and Fig. 10.55(b) shows the result at the next flooding step, n. The water has spilled from one basin to the other and, therefore, a dam must be built to keep this from happening. In order to be consistent with notation to be introduced shortly, let M_1 and M_2 denote the sets of coordinates of points in two regional minima. Then let the set of coordinates of points in the *catchment basin* associated with these two minima at stage $n - 1$ of flooding be denoted by $C_{n-1}(M_1)$ and $C_{n-1}(M_2)$, respectively. These are the two gray regions in Fig. 10.55(a).

Let $C[n - 1]$ denote the union of these two sets. There are two connected components in Fig. 10.55(a) (see Section 2.5.2 regarding connected components) and only one connected component in Fig. 10.55(b). This connected component encompasses the earlier two components, shown dashed. The fact that two connected components have become a *single* component indicates that water between the two catchment basins has merged at flooding step n. Let this connected component be denoted q. Note that the two components from step $n - 1$ can be extracted from q by performing the simple AND operation $q \cap C[n - 1]$. We note also that all points belonging to an individual catchment basin form a single connected component.

Suppose that each of the connected components in Fig. 10.55(a) is dilated by the structuring element shown in Fig. 10.55(c), subject to two conditions: (1) The dilation has to be constrained to q (this means that the center of the structuring element can be located only at points in q during dilation), and (2) the dilation cannot be performed on points that would cause the sets being dilated to merge (become a single connected component). Figure 10.55(d) shows that a first dilation pass (in light gray) expanded the boundary of each original connected component. Note that condition (1) was satisfied by every point



a
b c
d

FIGURE 10.55 (a) Two partially flooded catchment basins at stage $n - 1$ of flooding. (b) Flooding at stage n, showing that water has spilled between basins. (c) Structuring element used for dilation. (d) Result of dilation and dam construction.

during dilation, and condition (2) did not apply to any point during the dilation process; thus the boundary of each region was expanded uniformly.

In the second dilation (shown in black), several points failed condition (1) while meeting condition (2), resulting in the broken perimeter shown in the figure. It also is evident that the only points in q that satisfy the two conditions under consideration describe the 1-pixel-thick connected path shown crossed-hatched in Fig. 10.55(d). This path constitutes the desired separating dam at stage n of flooding. Construction of the dam at this level of flooding is completed by setting all the points in the path just determined to a value greater than the maximum intensity value of the image. The height of all dams is generally set at 1 plus the maximum allowed value in the image. This will prevent water from crossing over the part of the completed dam as the level of flooding is increased. It is important to note that dams built by this procedure, which are the desired segmentation boundaries, are connected components. In other words, this method eliminates the problems of broken segmentation lines.

Although the procedure just described is based on a simple example, the method used for more complex situations is exactly the same, including the use of the 3×3 symmetric structuring element shown in Fig. 10.55(c).

10.5.3 Watershed Segmentation Algorithm

Let $M_1, M_2, \ldots, M_R$ be sets denoting the *coordinates* of the points in the regional minima of an image $g(x, y)$. As indicated at the end of Section 10.5.1, this typically will be a gradient image. Let $C(M_i)$ be a set denoting the coordinates of the points in the catchment basin associated with regional minimum M_i (recall that the points in any catchment basin form a connected component). The notation min and max will be used to denote the minimum and maximum values of $g(x, y)$. Finally, let $T[n]$ represent the set of coordinates (s, t) for which $g(s, t) < n$. That is,

$$T[n] = \{(s, t) \,|\, g(s, t) < n\} \tag{10.5-1}$$

Geometrically, $T[n]$ is the set of coordinates of points in $g(x, y)$ lying below the plane $g(x, y) = n$.

The topography will be flooded in *integer* flood increments, from $n = \min + 1$ to $n = \max + 1$. At any step n of the flooding process, the algorithm needs to know the number of points below the flood depth. Conceptually, suppose that the coordinates in $T[n]$ that are below the plane $g(x, y) = n$ are "marked" black, and all other coordinates are marked white. Then when we look "down" on the xy-plane at any increment n of flooding, we will see a binary image in which black points correspond to points in the function that are below the plane $g(x, y) = n$. This interpretation is quite useful in helping clarify the following discussion.

Let $C_n(M_i)$ denote the set of coordinates of points in the catchment basin associated with minimum M_i that are flooded at stage n. With reference to the discussion in the previous paragraph, $C_n(M_i)$ may be viewed as a binary image given by

$$C_n(M_i) = C(M_i) \cap T[n] \tag{10.5-2}$$

In other words, $C_n(M_i) = 1$ at location (x, y) if $(x, y) \in C(M_i)$ AND $(x, y) \in T[n]$; otherwise $C_n(M_i) = 0$. The geometrical interpretation of this result is straightforward. We are simply using the AND operator to isolate at stage n of flooding the portion of the binary image in $T[n]$ that is associated with regional minimum M_i.

Next, we let $C[n]$ denote the union of the flooded catchment basins at stage n:

$$C[n] = \bigcup_{i=1}^{R} C_n(M_i) \tag{10.5-3}$$

Then $C[\max + 1]$ is the union of all catchment basins:

$$C[\max + 1] = \bigcup_{i=1}^{R} C(M_i) \tag{10.5-4}$$

It can be shown (Problem 10.41) that the elements in both $C_n(M_i)$ and $T[n]$ are never replaced during execution of the algorithm, and that the number of elements in these two sets either increases or remains the same as n increases. Thus, it follows that $C[n - 1]$ is a subset of $C[n]$. According to Eqs. (10.5-2) and (10.5-3), $C[n]$ is a subset of $T[n]$, so it follows that $C[n - 1]$ is a subset of $T[n]$. From this we have the important result that each connected component of $C[n - 1]$ is contained in exactly one connected component of $T[n]$.

The algorithm for finding the watershed lines is initialized with $C[\min + 1] = T[\min + 1]$. The algorithm then proceeds recursively, computing $C[n]$ from $C[n - 1]$. A procedure for obtaining $C[n]$ from $C[n - 1]$ is as follows. Let Q denote the set of connected components in $T[n]$. Then, for each connected component $q \in Q[n]$, there are three possibilities:

1. $q \cap C[n - 1]$ is empty.
2. $q \cap C[n - 1]$ contains one connected component of $C[n - 1]$.
3. $q \cap C[n - 1]$ contains more than one connected component of $C[n - 1]$.

Construction of $C[n]$ from $C[n - 1]$ depends on which of these three conditions holds. Condition 1 occurs when a new minimum is encountered, in which case connected component q is incorporated into $C[n - 1]$ to form $C[n]$. Condition 2 occurs when q lies within the catchment basin of some regional minimum, in which case q is incorporated into $C[n - 1]$ to form $C[n]$. Condition 3 occurs when all, or part, of a ridge separating two or more catchment basins is encountered. Further flooding would cause the water level in these catchment basins to merge. Thus a dam (or dams if more than two catchment basins are involved) must be built within q to prevent overflow between the catchment basins. As explained in the previous section, a one-pixel-thick dam can be constructed when needed by dilating $q \cap C[n - 1]$ with a 3×3 structuring element of 1s, and constraining the dilation to q.

Algorithm efficiency is improved by using only values of n that correspond to existing intensity values in $g(x, y)$; we can determine these values, as well as the values of min and max, from the histogram of $g(x, y)$.

a b
c d

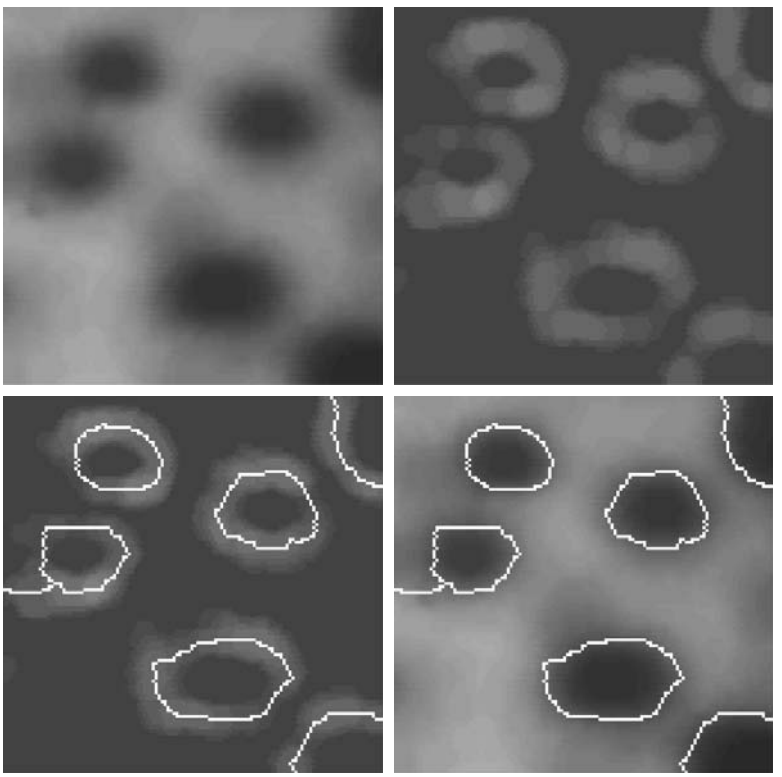


FIGURE 10.56
(a) Image of blobs. (b) Image gradient. (c) Watershed lines. (d) Watershed lines superimposed on original image. (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

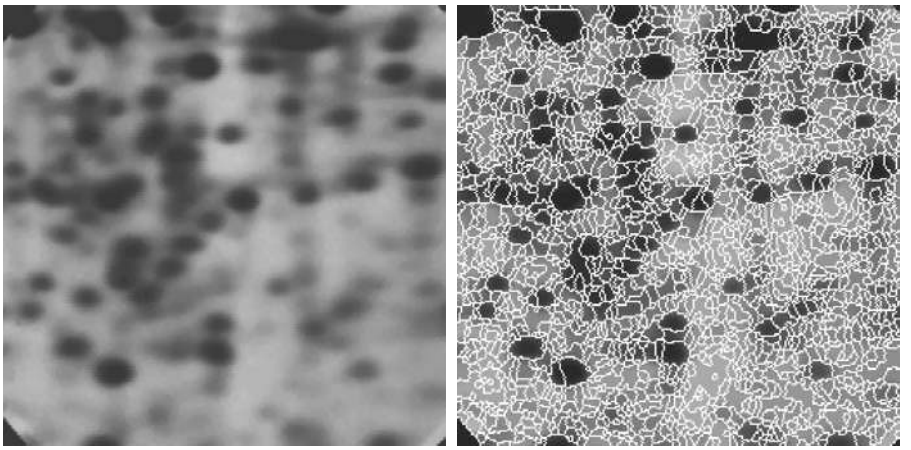
EXAMPLE 10.25: Illustration of the watershed segmentation algorithm.

■ Consider the image and its gradient in Figs. 10.56(a) and (b), respectively. Application of the watershed algorithm just described yielded the watershed lines (white paths) of the gradient image in Fig. 10.56(c). These segmentation boundaries are shown superimposed on the original image in Fig. 10.56(d). As noted at the beginning of this section, the segmentation boundaries have the important property of being connected paths. ■

10.5.4 The Use of Markers

Direct application of the watershed segmentation algorithm in the form discussed in the previous section generally leads to *oversegmentation* due to noise and other local irregularities of the gradient. As Fig. 10.57 shows, oversegmentation can be serious enough to render the result of the algorithm virtually useless. In this case, this means a large number of segmented regions. A practical solution to this problem is to limit the number of allowable regions by incorporating a preprocessing stage designed to bring additional knowledge into the segmentation procedure.

An approach used to control oversegmentation is based on the concept of markers. A *marker* is a connected component belonging to an image. We have *internal* markers, associated with objects of interest, and *external* markers, associated with the background. A procedure for marker selection typically will consist of two principal steps: (1) preprocessing; and (2) definition of a set of criteria that markers must satisfy. To illustrate, consider Fig. 10.57(a) again.

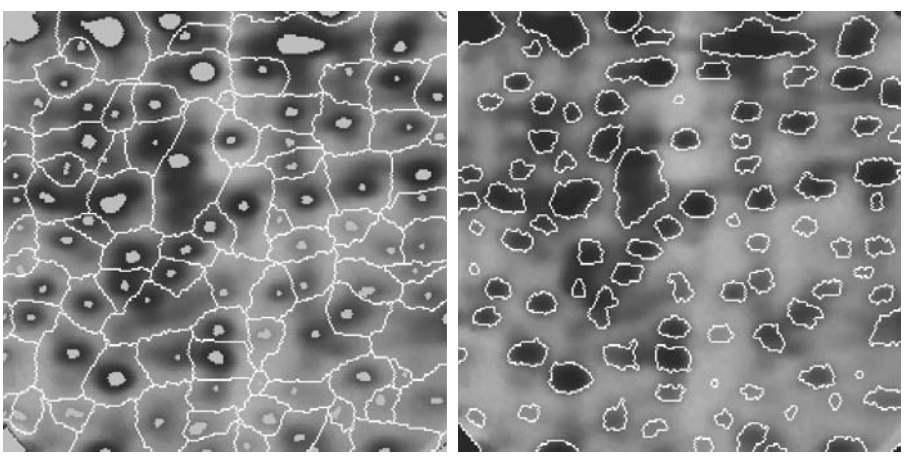


a b

FIGURE 10.57 (a) Electrophoresis image. (b) Result of applying the watershed segmentation algorithm to the gradient image. Oversegmentation is evident. (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

Part of the problem that led to the oversegmented result in Fig. 10.57(b) is the large number of potential minima. Because of their size, many of these minima are irrelevant detail. As has been pointed out several times in earlier discussions, an effective method for minimizing the effect of small spatial detail is to filter the image with a smoothing filter. This is an appropriate preprocessing scheme in this particular case.

Suppose that we define an *internal marker* as (1) a region that is surrounded by points of higher "altitude"; (2) such that the points in the region form a connected component; and (3) in which all the points in the connected component have the same intensity value. After the image was smoothed, the internal markers resulting from this definition are shown as light gray, bloblike regions in Fig. 10.58(a). Next, the watershed algorithm was applied to the



a b

FIGURE 10.58 (a) Image showing internal markers (light gray regions) and external markers (watershed lines). (b) Result of segmentation. Note the improvement over Fig. 10.47(b). (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

smoothed image, under the restriction that these internal markers be the *only* allowed regional minima. Figure 10.58(a) shows the resulting watershed lines. These watershed lines are defined as the *external markers*. Note that the points along the watershed line pass along the highest points between neighboring markers.

The external markers in Fig. 10.58(a) effectively partition the image into regions, with each region containing a single internal marker and part of the background. The problem is thus reduced to partitioning each of these regions into two: a single object and its background. We can bring to bear on this simplified problem many of the segmentation techniques discussed earlier in this chapter. Another approach is simply to apply the watershed segmentation algorithm to each individual region. In other words, we simply take the gradient of the smoothed image [as in Fig. 10.56(b)] and then restrict the algorithm to operate on a single watershed that contains the marker in that particular region. The result obtained using this approach is shown in 10.58(b). The improvement over the image in 10.57(b) is evident.

Marker selection can range from simple procedures based on intensity values and connectivity, as was just illustrated, to more complex descriptions involving size, shape, location, relative distances, texture content, and so on (see Chapter 11 regarding descriptors). The point is that using markers brings a priori knowledge to bear on the segmentation problem. The reader is reminded that humans often aid segmentation and higher-level tasks in everyday vision by using a priori knowledge, one of the most familiar being the use of context. Thus, the fact that segmentation by watersheds offers a framework that can make effective use of this type of knowledge is a significant advantage of this method.

10.6 The Use of Motion in Segmentation

Motion is a powerful cue used by humans and many other animals to extract objects or regions of interest from a background of irrelevant detail. In imaging applications, motion arises from a relative displacement between the sensing system and the scene being viewed, such as in robotic applications, autonomous navigation, and dynamic scene analysis. In the following sections we consider the use of motion in segmentation both spatially and in the frequency domain.

10.6.1 Spatial Techniques

Basic approach

One of the simplest approaches for detecting changes between two image frames $f(x, y, t_i)$ and $f(x, y, t_j)$ taken at times t_i and t_j, respectively, is to compare the two images pixel by pixel. One procedure for doing this is to form a difference image. Suppose that we have a reference image containing only stationary components. Comparing this image against a subsequent image of the same scene, but including a moving object, results in the difference of the two images canceling the stationary elements, leaving only nonzero entries that correspond to the nonstationary image components.

A difference image between two images taken at times t_i and t_j may be defined as

$$d_{ij}(x, y) = \begin{cases} 1 & \text{if } |f(x, y, t_i) - f(x, y, t_j)| > T \\ 0 & \text{otherwise} \end{cases} \tag{10.6-1}$$

where T is a specified threshold. Note that $d_{ij}(x, y)$ has a value of 1 at spatial coordinates (x, y) only if the intensity difference between the two images is *appreciably different* at those coordinates, as determined by the specified threshold T. It is assumed that all images are of the same size. Finally, we note that the values of the coordinates (x, y) in Eq. (10.6-1) span the dimensions of these images, so that the difference image $d_{ij}(x, y)$ is of the same size as the images in the sequence.

In dynamic image processing, all pixels in $d_{ij}(x, y)$ with value 1 are considered the result of object motion. This approach is applicable only if the two images are registered spatially and if the illumination is relatively constant within the bounds established by T. In practice, 1-valued entries in $d_{ij}(x, y)$ may arise as a result of noise. Typically, these entries are isolated points in the difference image, and a simple approach to their removal is to form 4- or 8-connected regions of 1s in $d_{ij}(x, y)$ and then ignore any region that has less than a predetermined number of elements. Although it may result in ignoring small and/or slow-moving objects, this approach improves the chances that the remaining entries in the difference image actually are the result of motion.

Accumulative differences

Consider a sequence of image frames $f(x, y, t_1), f(x, y, t_2), \ldots, f(x, y, t_n)$ and let $f(x, y, t_1)$ be the *reference image*. An *accumulative difference image* (ADI) is formed by comparing this reference image with every subsequent image in the sequence. A counter for each pixel location in the accumulative image is incremented every time a difference occurs at that pixel location between the reference and an image in the sequence. Thus when the kth frame is being compared with the reference, the entry in a given pixel of the accumulative image gives the number of times the intensity at that position was different [as determined by T in Eq. (10.6-1)] from the corresponding pixel value in the reference image.

Consider the following three types of accumulative difference images: *absolute*, *positive*, and *negative* ADIs. Assuming that the intensity values of the moving objects are larger than the background, these three types of ADIs are defined as follows. Let $R(x, y)$ denote the reference image and, to simplify the notation, let k denote t_k, so that $f(x, y, k) = f(x, y, t_k)$. We assume that $R(x, y) = f(x, y, 1)$. Then, for any $k > 1$, and keeping in mind that the values of the ADIs are *counts*, we define the following for all relevant values of (x, y):

$$A_k(x, y) = \begin{cases} A_{k-1}(x, y) + 1 & \text{if } |R(x, y) - f(x, y, k)| > T \\ A_{k-1}(x, y) & \text{otherwise} \end{cases} \tag{10.6-2}$$

$$P_k(x, y) = \begin{cases} P_{k-1}(x, y) + 1 & \text{if } \big[R(x, y) - f(x, y, k)\big] > T \\ P_{k-1}(x, y) & \text{otherwise} \end{cases} \tag{10.6-3}$$

and

$$N_k(x, y) = \begin{cases} N_{k-1}(x, y) + 1 & \text{if } \big[R(x, y) - f(x, y, k)\big] < -T \\ N_{k-1}(x, y) & \text{otherwise} \end{cases} \tag{10.6-4}$$

where $A_k(x, t)$, $P_k(x, y)$, and $N_k(x, y)$ are the absolute, positive, and negative ADIs, respectively, after the kth image in the sequence is encountered.

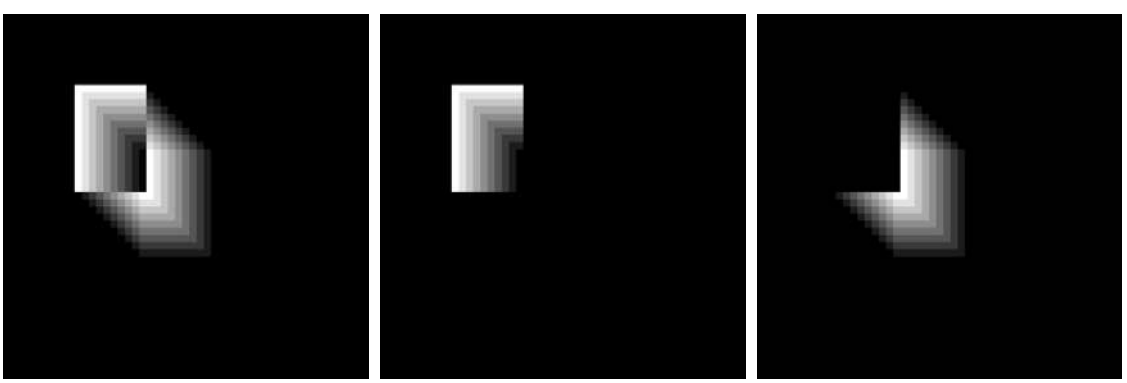
It is understood that these ADIs start out with all zero values (counts). Note also that the ADIs are of the same size as the images in the sequence. Finally, we note that the order of the inequalities and signs of the thresholds in Eqs. (10.6-3) and (10.6-4) are reversed if the intensity values of the background pixels are greater than the values of the moving objects.

EXAMPLE 10.26: Computation of the absolute, positive, and negative accumulative difference images.

■ Figure 10.59 shows the three ADIs displayed as intensity images for a rectangular object of dimension 75×50 pixels that is moving in a southeasterly direction at a speed of $5\sqrt{2}$ pixels per frame. The images are of size 256×256 pixels. We note the following: (1) The nonzero area of the positive ADI is equal to the size of the moving object. (2) The location of the positive ADI corresponds to the location of the moving object in the reference frame. (3) The number of counts in the positive ADI stops increasing when the moving object is displaced completely with respect to the same object in the reference frame. (4) The absolute ADI contains the regions of the positive and negative ADI. (5) The direction and speed of the moving object can be determined from the entries in the absolute and negative ADIs. ■

Establishing a reference image

A key to the success of the techniques discussed in the preceding two sections is having a reference image against which subsequent comparisons can be



a b c

FIGURE 10.59 ADIs of a rectangular object moving in a southeasterly direction. (a) Absolute ADI. (b) Positive ADI. (c) Negative ADI.

made. The difference between two images in a dynamic imaging problem has the tendency to cancel all stationary components, leaving only image elements that correspond to noise and to the moving objects.

In practice, obtaining a reference image with only stationary elements is not always possible, and building a reference from a set of images containing one or more moving objects becomes necessary. This applies particularly to situations describing busy scenes or in cases where frequent updating is required. One procedure for generating a reference image is as follows. Consider the first image in a sequence to be the reference image. When a nonstationary component has moved completely out of its position in the reference frame, the corresponding background in the present frame can be duplicated in the location originally occupied by the object in the reference frame. When all moving objects have moved completely out of their original positions, a reference image containing only stationary components will have been created. Object displacement can be established by monitoring the changes in the positive ADI, as indicated in the preceding section.

EXAMPLE 10.27: Building a reference image.

■ Figures 10.60(a) and (b) show two image frames of a traffic intersection. The first image is considered the reference, and the second depicts the same scene some time later. The objective is to remove the principal moving objects in the reference image in order to create a static image. Although there are other smaller moving objects, the principal moving feature is the automobile at the intersection moving from left to right. For illustrative purposes we focus on this object. By monitoring the changes in the positive ADI, it is possible to determine the initial position of a moving object, as explained previously. Once the area occupied by this object is identified, the object can be removed from the image by subtraction. By looking at the frame in the sequence at which the positive ADI stopped changing, we can copy from this image the area previously occupied by the moving object in the initial frame. This area then is pasted onto the image from which the object was cut out, thus restoring the background of that area. If this is done for all moving objects, the result is a reference image with only static components against which we can compare subsequent frames for motion detection. The result of removing the eastbound moving vehicle in this case is shown in Fig. 10.60(c). ■

a b c

FIGURE 10.60 Building a static reference image. (a) and (b) Two frames in a sequence. (c) Eastbound automobile subtracted from (a) and the background restored from the corresponding area in (b). (Jain and Jain.)

10.6.2 Frequency Domain Techniques

In this section we consider the problem of determining motion via a Fourier transform formulation. Consider a sequence $f(x, y, t), t = 0, 1, \ldots, K - 1$, of K digital image frames of size $M \times N$ generated by a stationary camera. We begin the development by assuming that all frames have a homogeneous background of zero intensity. The exception is a single, 1-pixel object of unit intensity that is moving with constant velocity. Suppose that for frame one ($t = 0$), the object is at location (x', y') and that the image plane is *projected* onto the x-axis; that is, the pixel intensities are summed across the columns in the image. This operation yields a 1-D array with M entries that are zero, except at x', which is the x-coordinate of the single-point object. If we now multiply all the components of the 1-D array by the quantity $\exp[j2\pi a_1 x \Delta t]$ for $x = 0, 1, 2, \ldots, M - 1$ and sum the results, we obtain the single term $\exp[j2\pi a_1 x' \Delta t]$. In this notation, a_1 is a positive integer, and Δt is the time interval between frames.

Suppose that in frame two ($t = 1$) the object has moved to coordinates $(x' + 1, y')$; that is, it has moved 1 pixel parallel to the x-axis. Then repeating the projection procedure discussed in the previous paragraph yields the sum $\exp[j2\pi a_1(x' + 1)\Delta t]$. If the object continues to move 1 pixel location per frame, then, at any integer instant of time, t, the result is $\exp[j2\pi a_1(x' + t)\Delta t]$, which, using Euler's formula, may be expressed as

$$e^{j2\pi a_1(x'+t)\Delta t} = \cos[2\pi a_1(x' + t)\Delta t] + j \sin[2\pi a_1(x' + t)\Delta t] \qquad (10.6\text{-}5)$$

for $t = 0, 1, \ldots, K - 1$. In other words, this procedure yields a complex sinusoid with frequency a_1. If the object were moving V_1 pixels (in the x-direction) between frames, the sinusoid would have frequency $V_1 a_1$. Because t varies between 0 and $K - 1$ in integer increments, restricting a_1 to integer values causes the discrete Fourier transform of the complex sinusoid to have two peaks—one located at frequency $V_1 a_1$ and the other at $K - V_1 a_1$. This latter peak is the result of symmetry in the discrete Fourier transform, as discussed in Section 4.6.4, and may be ignored. Thus a peak search in the Fourier spectrum yields $V_1 a_1$. Division of this quantity by a_1 yields V_1, which is the velocity component in the x-direction, as the frame rate is assumed to be known. A similar argument would yield V_2, the component of velocity in the y-direction.

A sequence of frames in which no motion takes place produces identical exponential terms, whose Fourier transform would consist of a single peak at a frequency of 0 (a single dc term). Therefore, because the operations discussed so far are linear, the general case involving one or more moving objects in an arbitrary static background would have a Fourier transform with a peak at dc corresponding to static image components and peaks at locations proportional to the velocities of the objects.

These concepts may be summarized as follows. For a sequence of K digital images of size $M \times N$, the sum of the weighted projections onto the x axis at any integer instant of time is

$$g_x(t, a_1) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y, t) e^{j2\pi a_1 x \Delta t} \quad t = 0, 1, \ldots, K - 1 \qquad (10.6\text{-}6)$$

Similarly, the sum of the projections onto the y-axis is

$$g_y(t, a_2) = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} f(x, y, t) e^{j2\pi a_2 y \Delta t} \quad t = 0, 1, \ldots, K-1 \tag{10.6-7}$$

where, as noted already, a_1 and a_2 are positive integers.

The 1-D Fourier transforms of Eqs. (10.6-6) and (10.6-7), respectively, are

$$G_x(u_1, a_1) = \sum_{t=0}^{K-1} g_x(t, a_1) e^{-j2\pi u_1 t/K} \quad u_1 = 0, 1, \ldots, K-1 \tag{10.6-8}$$

and

$$G_y(u_2, a_2) = \sum_{t=0}^{K-1} g_y(t, a_2) e^{-j2\pi u_2 t/K} \quad u_2 = 0, 1, \ldots, K-1 \tag{10.6-9}$$

In practice, computation of these transforms is carried out using an FFT algorithm, as discussed in Section 4.11.

The frequency-velocity relationship is

$$u_1 = a_1 V_1 \tag{10.6-10}$$

and

$$u_2 = a_2 V_2 \tag{10.6-11}$$

In this formulation the unit of velocity is in pixels per total frame time. For example, $V_1 = 10$ is interpreted as a motion of 10 pixels in K frames. For frames that are taken uniformly, the actual physical speed depends on the frame rate and the distance between pixels. Thus if $V_1 = 10$, $K = 30$, the frame rate is two images per second, and the distance between pixels is 0.5 m, then the actual physical speed in the x-direction is

$$\begin{aligned} V_1 &= (10 \text{ pixels})(0.5 \text{ m/pixel})(2 \text{ frames/s})/(30 \text{ frames}) \\ &= 1/3 \text{ m/s} \end{aligned}$$

The sign of the x-component of the velocity is obtained by computing

$$S_{1x} = \left. \frac{d^2 \text{Re}\left[g_x(t, a_1) \right]}{dt^2} \right|_{t=n} \tag{10.6-12}$$

and

$$S_{2x} = \left. \frac{d^2 \text{Im}\left[g_x(t, a_1) \right]}{dt^2} \right|_{t=n} \tag{10.6-13}$$

Because g_x is sinusoidal, it can be shown (Problem 10.47) that S_{1x} and S_{2x} will have the same sign at an arbitrary point in time, n, if the velocity component V_1 is positive. Conversely, opposite signs in S_{1x} and S_{2x} indicate a negative component. If either S_{1x} or S_{2x} is zero, we consider the next closest point in time, $t = n \pm \Delta t$. Similar comments apply to computing the sign of V_2.

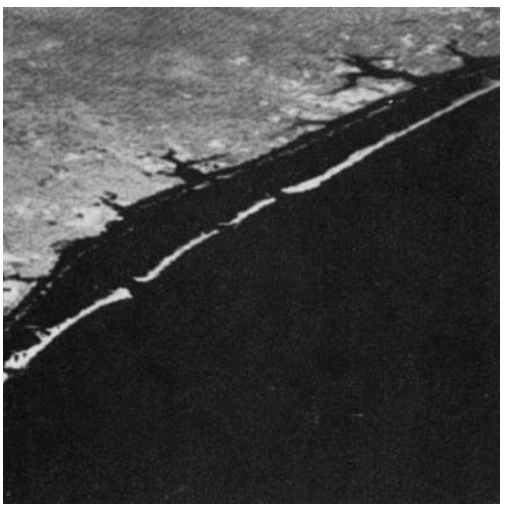


FIGURE 10.61 LANDSAT frame. (Cowart, Snyder, and Ruedger.)

EXAMPLE 10.28: Detection of a small moving object via the frequency domain.

■ Figures 10.61 through 10.64 illustrate the effectiveness of the approach just derived. Figure 10.61 shows one of a 32-frame sequence of LANDSAT images generated by adding white noise to a reference image. The sequence contains a superimposed target moving at 0.5 pixel per frame in the x-direction and 1 pixel per frame in the y-direction. The target, shown circled in Fig. 10.62, has a Gaussian intensity distribution spread over a small (9-pixel) area and is not easily discernible by eye. Figures 10.63 and 10.64 show the results of computing Eqs. (10.6-8) and (10.6-9) with $a_1 = 6$ and $a_2 = 4$, respectively. The peak at $u_1 = 3$ in Fig. 10.63 yields $V_1 = 0.5$ from Eq. (10.6-10). Similarly, the peak at $u_2 = 4$ in Fig. 10.64 yields $V_2 = 1.0$ from Eq. (10.6-11). ■

Guidelines for the selection of a_1 and a_2 can be explained with the aid of Figs. 10.63 and 10.64. For instance, suppose that we had used $a_2 = 15$ instead of $a_2 = 4$. In that case the peaks in Fig. 10.64 would now be at $u_2 = 15$ and 17 because $V_2 = 1.0$, which would be a seriously aliased result. As discussed in Section 4.5.4, aliasing is caused by undersampling (too few frames in the present discussion, as the range of u is determined by K). Because $u = aV$, one possibility is to select

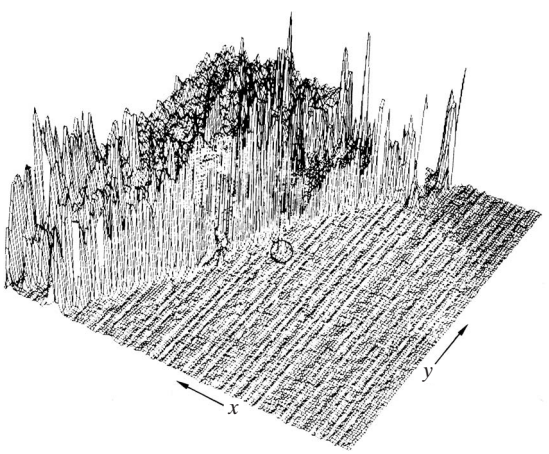


FIGURE 10.62 Intensity plot of the image in Fig. 10.61, with the target circled. (Rajala, Riddle, and Snyder.)

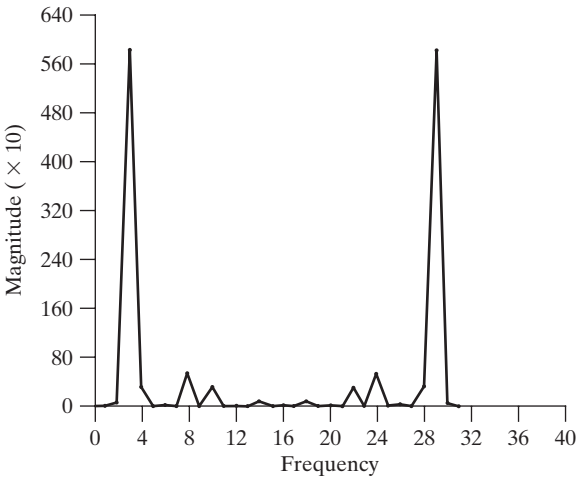


FIGURE 10.63
Spectrum of Eq. (10.6-8) showing a peak at $u_1 = 3$. (Rajala, Riddle, and Snyder.)

a as the integer closest to $a = u_{max}/V_{max}$, where u_{max} is the aliasing frequency limitation established by K and V_{max} is the maximum expected object velocity.

Summary

Image segmentation is an essential preliminary step in most automatic pictorial pattern recognition and scene analysis applications. As indicated by the range of examples presented in the previous sections, the choice of one segmentation technique over another is dictated mostly by the peculiar characteristics of the problem being considered. The methods discussed in this chapter, although far from exhaustive, are representative of techniques commonly used in practice. The following references can be used as the basis for further study of this topic.

References and Further Reading

Because of its central role in autonomous image processing, segmentation is a topic covered in most books dealing with image processing, image analysis, and computer vision. The following books provide complementary and/or supplementary reading for our coverage of this topic: Umbaugh [2005]; Davies [2005]; Gonzalez, Woods, and Eddins [2004]; Shapiro and Stockman [2001]; Sonka et al. [1999]; and Petrou and Bosdogianni [1999].

Work dealing with the use of masks to detect intensity discontinuities (Section 10.2) has a long history. Numerous masks have been proposed over the years: Roberts [1965], Prewitt [1970], Kirsh [1971], Robinson [1976], Frei and Chen [1977], and Canny [1986]. A review article by Fram and Deutsch [1975] contains numerous masks and an evaluation of

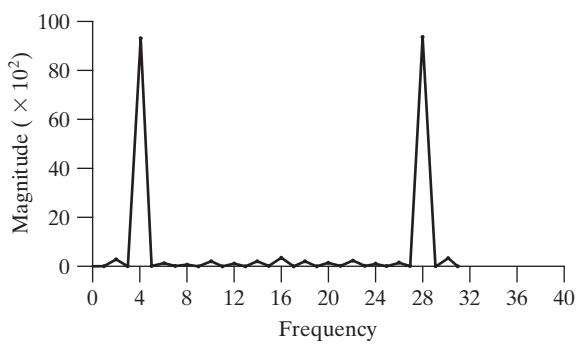


FIGURE 10.64
Spectrum of Eq. (10.6-9) showing a peak at $u_2 = 4$. (Rajala, Riddle, and Snyder.)

9 Morphological Image Processing

In form and feature, face and limb,
I grew so like my brother
That folks got taking me for him
And each for one another.

Henry Sambrooke Leigh, Carols of Cockayne, The Twins

Preview

The word *morphology* commonly denotes a branch of biology that deals with the form and structure of animals and plants. We use the same word here in the context of *mathematical morphology* as a tool for extracting image components that are useful in the representation and description of region shape, such as boundaries, skeletons, and the convex hull. We are interested also in morphological techniques for pre- or postprocessing, such as morphological filtering, thinning, and pruning.

In the following sections we develop and illustrate several important concepts in mathematical morphology. Many of the ideas introduced here can be formulated in terms of n-dimensional Euclidean space, E^n. However, our interest initially is on binary images whose components are elements of Z^2 (see Section 2.4.2). We discuss extensions to gray-scale images in Section 9.6.

The material in this chapter begins a transition from a focus on purely image processing methods, whose input and output are images, to processes in which the inputs are images, but the outputs are attributes extracted from those images, in the sense defined in Section 1.1. Tools such as morphology and related concepts are a cornerstone of the mathematical foundation that is utilized for extracting "meaning" from an image. Other approaches are developed and applied in the remaining chapters of the book.

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. Grayscale 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).

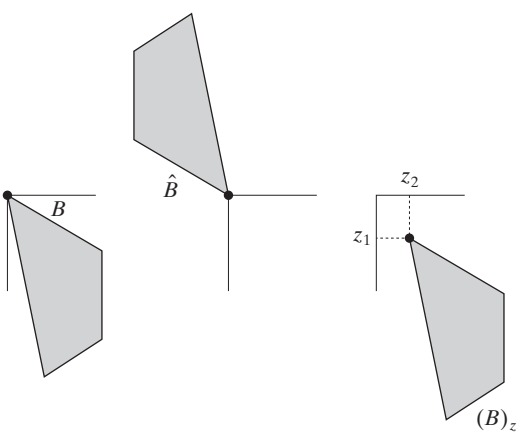
In addition to the basic set definitions in Section 2.6.4, the concepts of set reflection and translation are used extensively in morphology. The *reflection* of a set B, denoted $\hat{B}$, is defined as

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

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

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

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

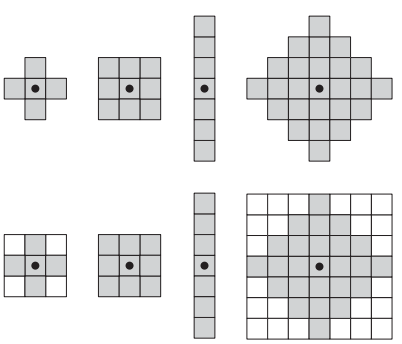
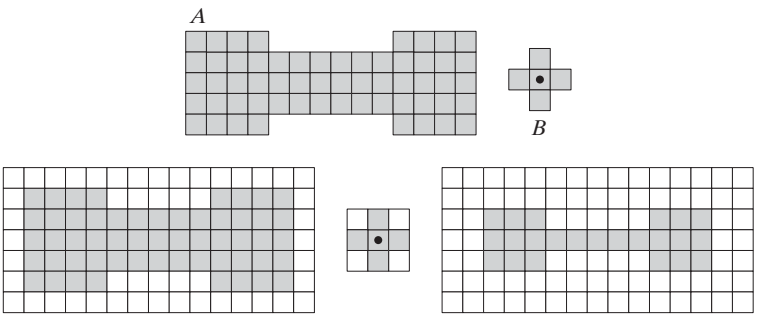


FIGURE 9.2 First row: Examples of structuring elements. Second row: Structuring elements converted to rectangular arrays. The dots denote the centers of the SEs.



a b
c d e

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

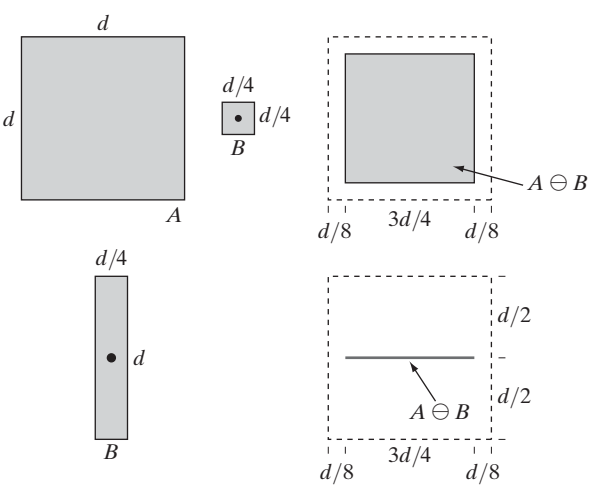
$$A \ominus B = \{z | (B)_z \subseteq A\} \tag{9.2-1}$$

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:

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

where, as defined in Section 2.6.4, A^c is the complement of A and $\varnothing$ 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



a b c
d e

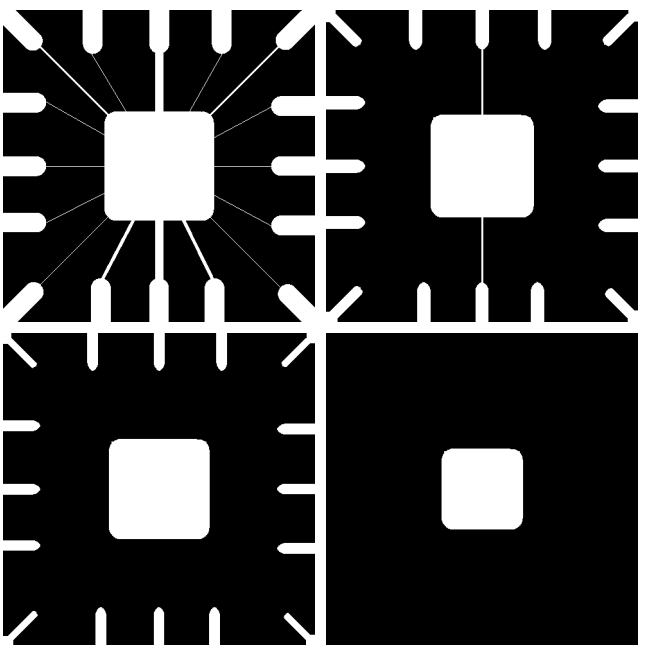
FIGURE 9.4 (a) Set A. (b) Square structuring element, B. (c) Erosion of A by B, shown shaded. (d) Elongated structuring element. (e) Erosion of A by B using this element. The dotted border in (c) and (e) is the boundary of set A, shown only for reference.

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 wirebond 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

$$A \oplus B = \left\{ z | (\hat{B})_z \cap A \neq \varnothing \right\} \tag{9.2-3}$$

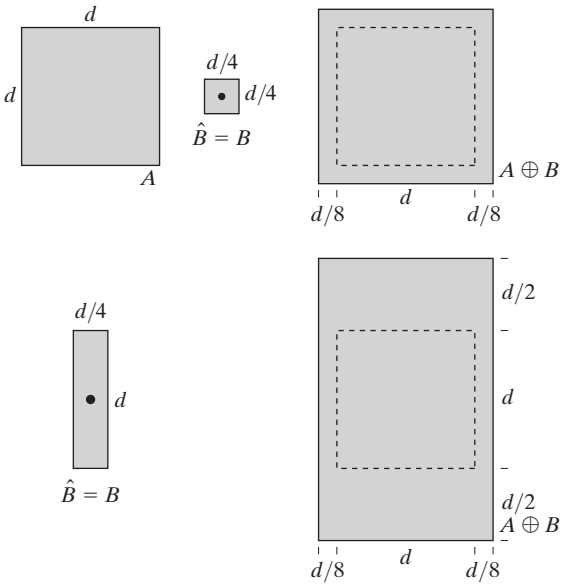
This equation is based on reflecting B about its origin, and shifting this reflection by z (see Fig. 9.1). The dilation of A by B then is the set of all displacements, z, such that $\hat{B}$ and A overlap by at least one element. Based on this interpretation, Eq. (9.2-3) can be written equivalently as

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

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.



a b c
d e

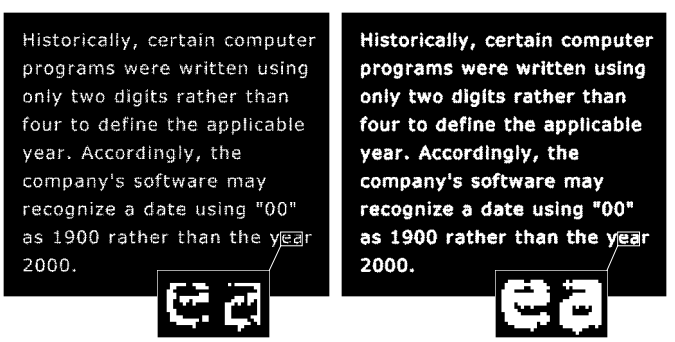
FIGURE 9.6
(a) Set A. (b) Square structuring element (the dot denotes the origin). (c) Dilation of A by B, shown shaded. (d) Elongated structuring element. (e) Dilation of A using this element. The dotted border in (c) and (e) is the boundary of set A, shown only for reference

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

a c
b

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.



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,

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

and

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

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

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

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

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

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

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

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

$$A \circ B = (A \ominus B) \oplus B \tag{9.3-1}$$

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

$$A \bullet B = (A \oplus B) \ominus B \tag{9.3-2}$$

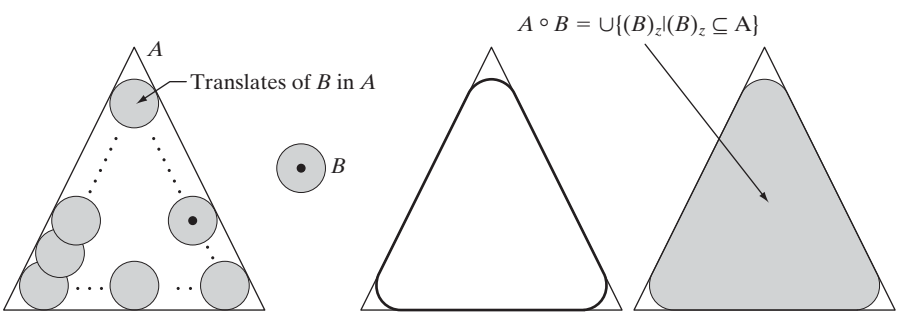
which says that the closing of A by B is simply the dilation of A by B, followed by the erosion of the result by B.

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

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

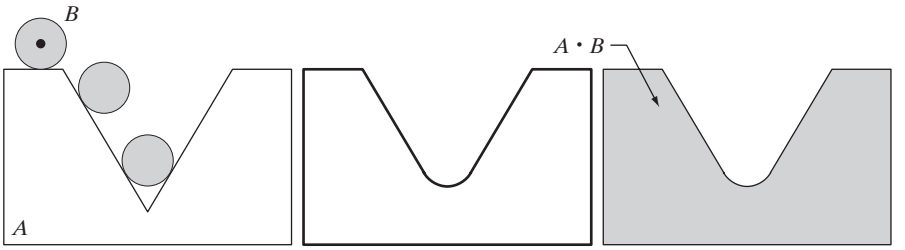
where $\cup\{\cdot\}$ denotes the union of all the sets inside the braces.

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 $(B)_z \cap A \neq \varnothing$ for any translate of $(B)_z$ that contains w. Figure 9.9 illustrates the basic geometrical properties of closing.



a b c d

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.

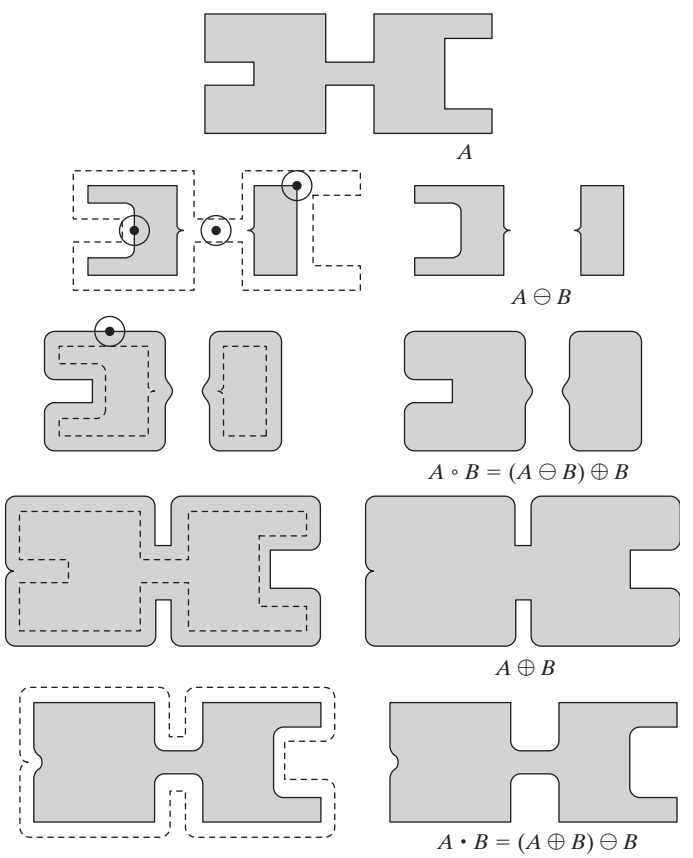


a b c

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.



a
b c
d e
f g
h i

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,

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

and

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

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:

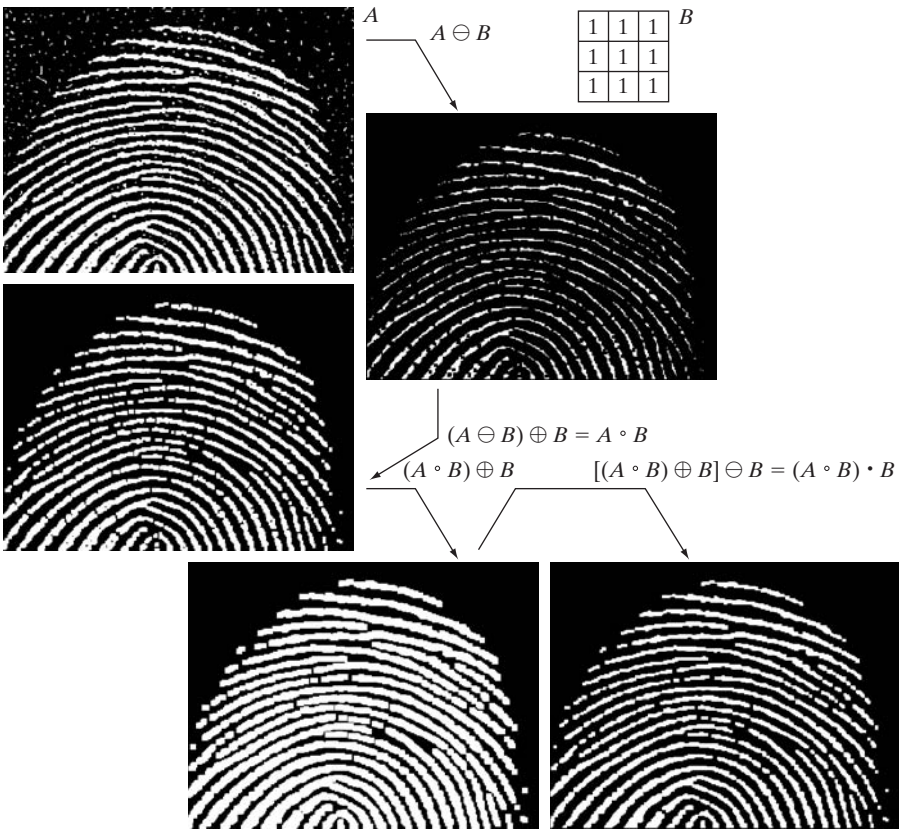
(a) A is a subset (subimage) of $A \bullet B$.
(b) If C is a subset of D, then $C \bullet B$ is a subset of $D \bullet B$.
(c) $(A \bullet B) \bullet B = A \bullet B$.

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



a b
d c
e f

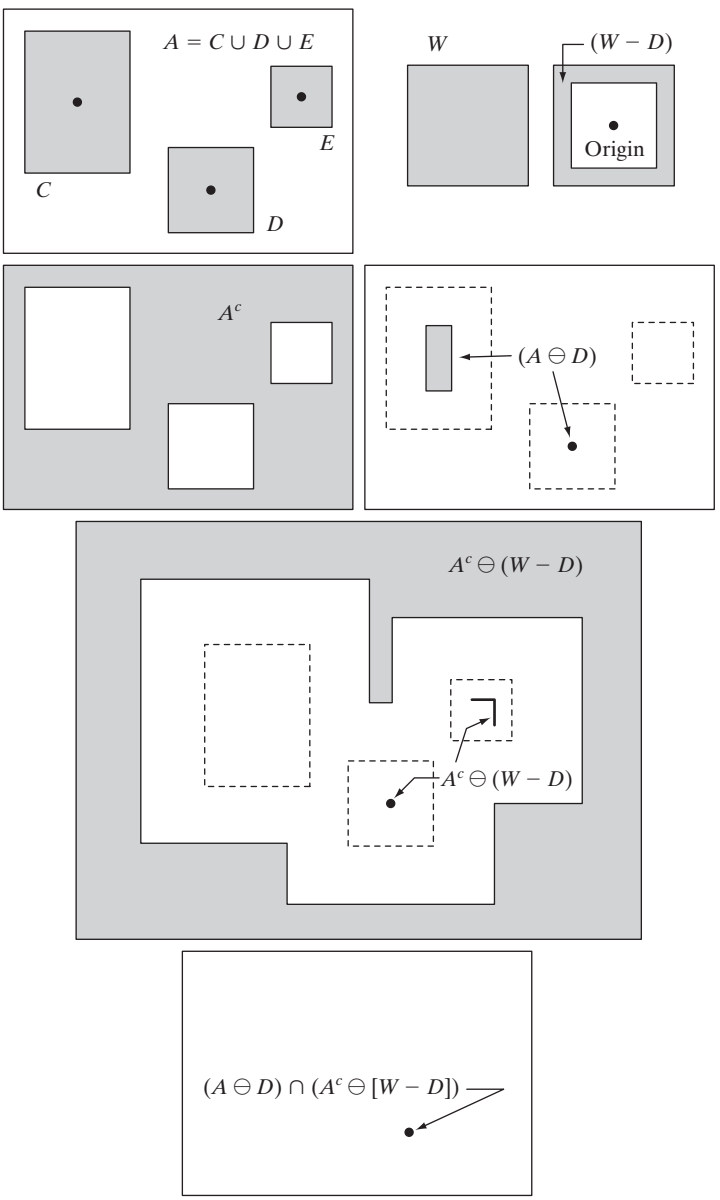
FIGURE 9.11
(a) Noisy image. (b) Structuring element. (c) Eroded image. (d) Opening of A. (e) Dilation of the opening. (f) Closing of the opening. (Original image courtesy of the National Institute of Standards and Technology.)

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 $A \circledast B$, is

$$A \circledast B = (A \ominus D) \cap \left[A^c \ominus (W - D)\right] \tag{9.4-1}$$

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

$$A \circledast B = (A \ominus B_1) \cap (A^c \ominus B_2) \tag{9.4-2}$$

Thus, set $A \circledast B$ 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

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

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

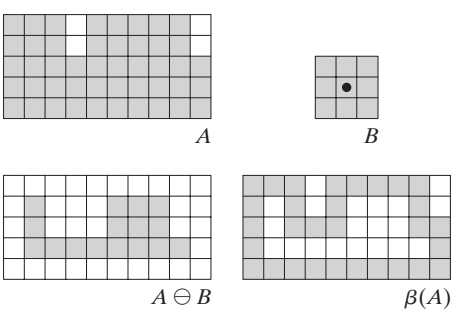
The boundary of a set A, denoted by $\beta(A)$, can be obtained by first eroding A by B and then performing the set difference between A and its erosion. That is,

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

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.



a b
c d

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).

EXAMPLE 9.5: Boundary extraction by morphological processing.

■ 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. ■

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:

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

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

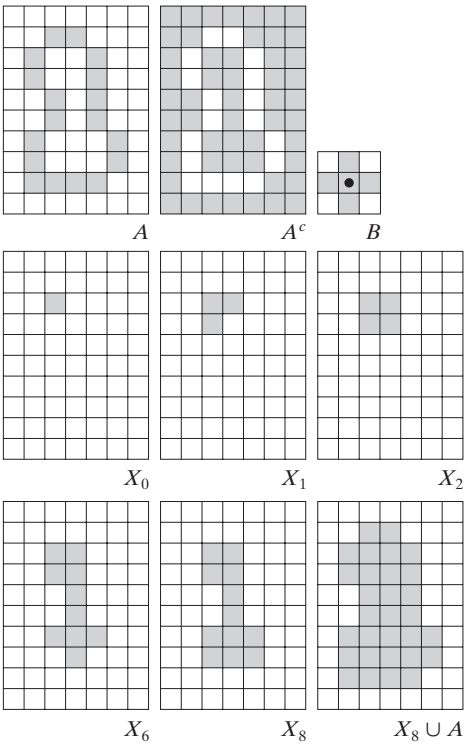
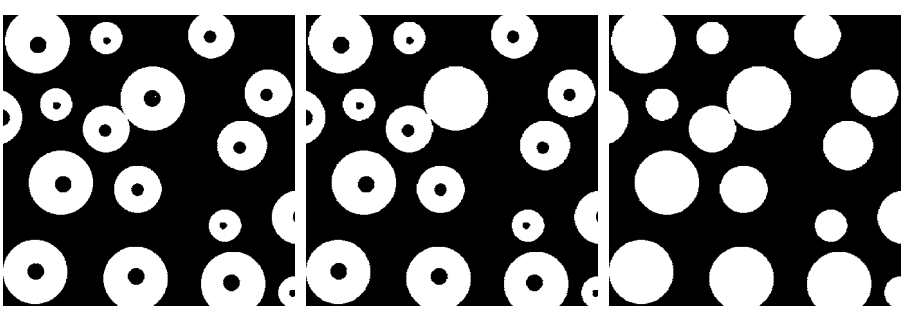


FIGURE 9.15 Hole filling. (a) Set A (shown shaded). (b) Complement of A. (c) Structuring element B. (d) Initial point inside the boundary. (e)–(h) Various steps of Eq. (9.5-2). (i) Final result [union of (a) and (h)].

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,



a b c

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.

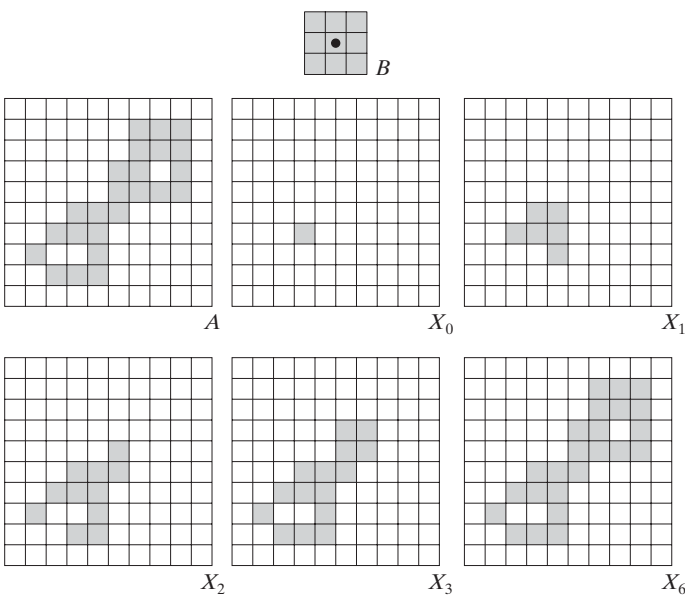
Fig. 9.16(c) shows the result of filling all the spheres. Because it must be known whether black points are background points or sphere inner points, fully automating this procedure requires that additional "intelligence" be built into the algorithm. We give a fully automatic approach in Section 9.5.9 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:

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

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



a
b c d
e f g

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:

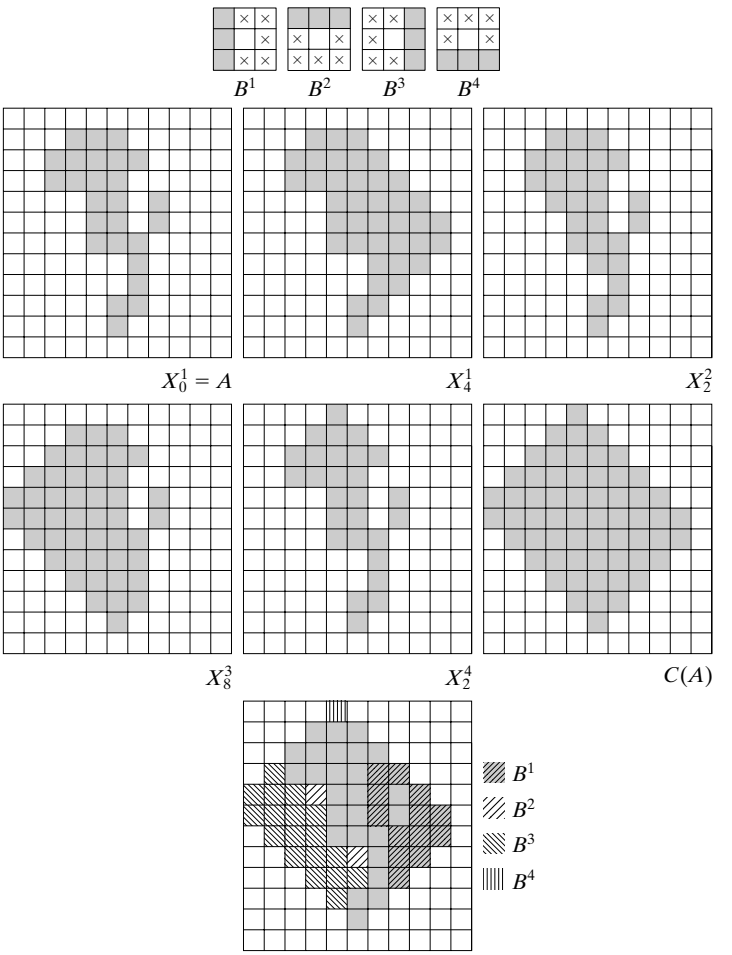
$$X_k^i = (X_{k-1} \circledast B^i) \cup A \quad i = 1, 2, 3, 4 \quad \text{and} \quad k = 1, 2, 3, \ldots \tag{9.5-4}$$

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

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

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 Ds 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



a
b c d
e f g
h

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:

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

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:

$$\{B\} = \{B^1, B^2, B^3, \ldots, B^n\} \tag{9.5-7}$$

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

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

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).

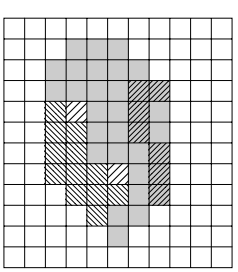


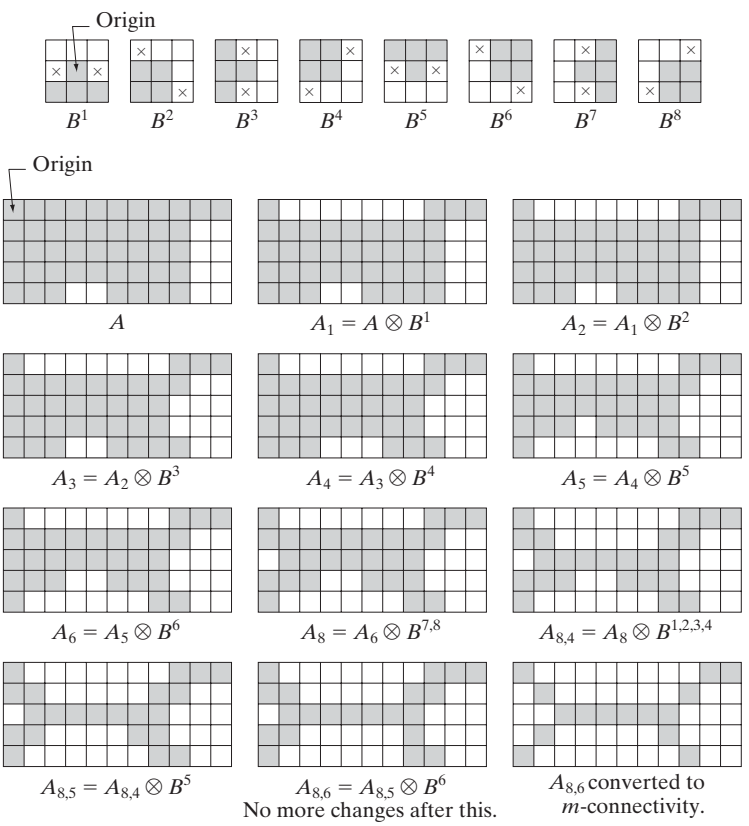
FIGURE 9.20 Result of limiting growth of the convex hull algorithm to the maximum dimensions of the original set of points along the vertical and horizontal directions.

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

$$A \odot B = A \cup (A \circledast B) \tag{9.5-9}$$



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

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.

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

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

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.

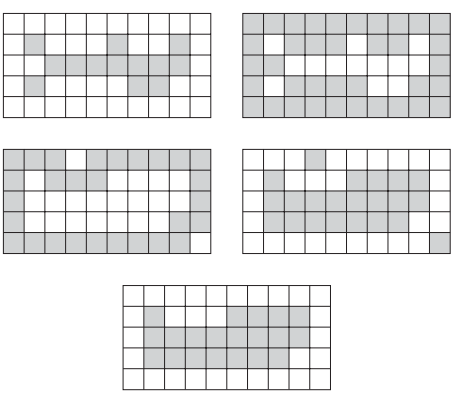
Depending on the nature of A, this procedure can result in disconnected points, as Fig. 9.22(d) shows. Hence thickening by this method usually is followed by postprocessing to remove disconnected points. Note from Fig. 9.22(c) that the thinned background forms a boundary for the thickening process. This 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.



a b
c d
e

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

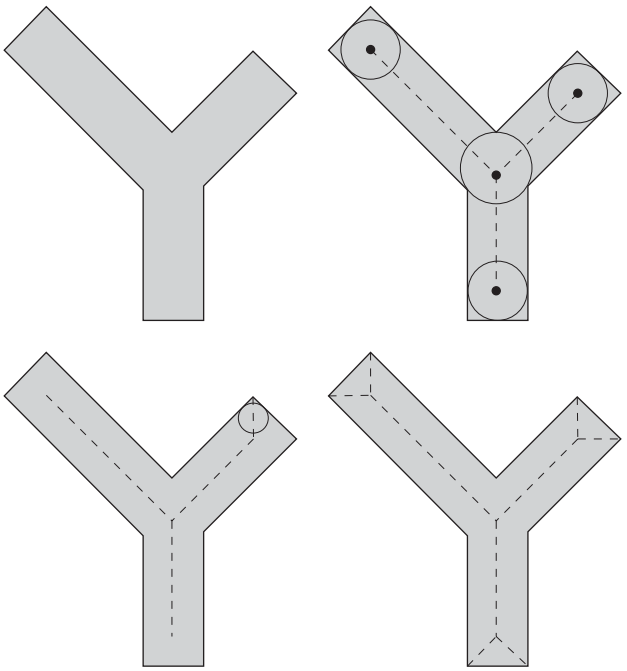


FIGURE 9.23
(a) Set A. (b) Various positions of maximum disks with centers on the skeleton of A. (c) Another maximum disk on a different segment of the skeleton of A. (d) Complete skeleton.

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

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

with

$$S_k(A) = (A \ominus kB) - (A \ominus kB) \circ B \tag{9.5-12}$$

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

$$(A \ominus kB) = ((\ldots((A \ominus B) \ominus B) \ominus \ldots) \ominus B) \tag{9.5-13}$$

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

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

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

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

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

$$(S_k(A) \oplus kB) = ((\ldots((S_k(A) \oplus B) \oplus B) \oplus \ldots) \oplus B) \tag{9.5-16}$$

EXAMPLE 9.8: Computing the skeleton of a simple figure.

■ 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.

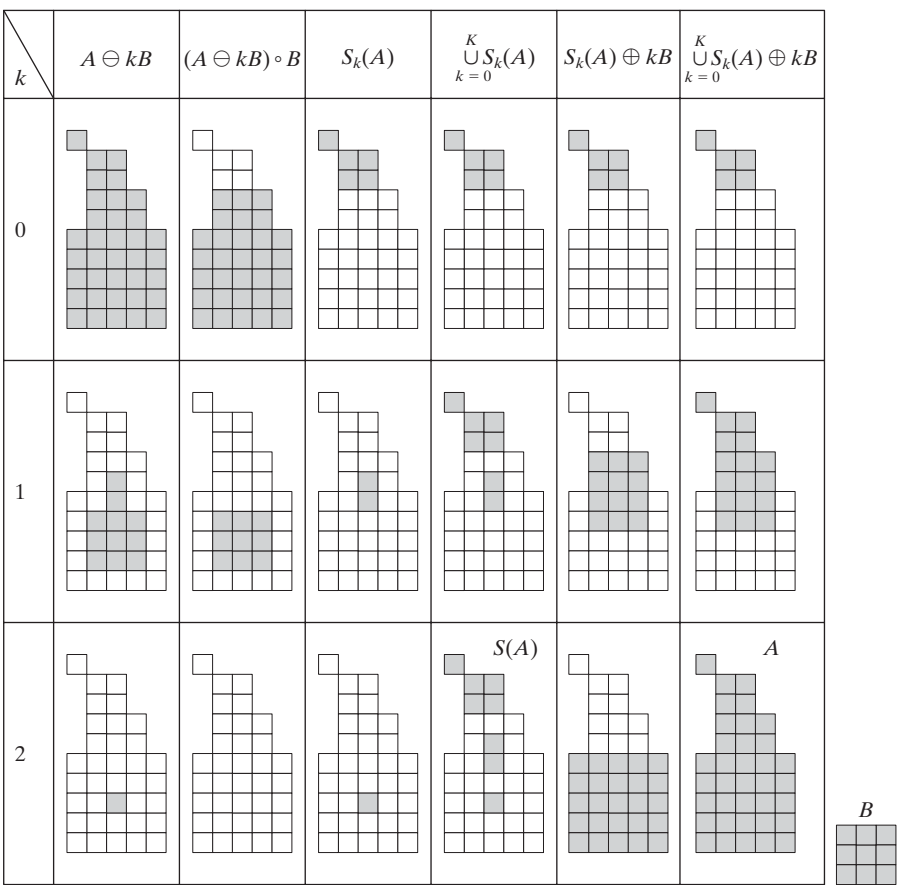


FIGURE 9.24 Implementation of Eqs. (9.5-11) through (9.5-15). The original set is at the top left, and its morphological skeleton is at the bottom of the fourth column. The reconstructed set is at the bottom of the sixth column.

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.

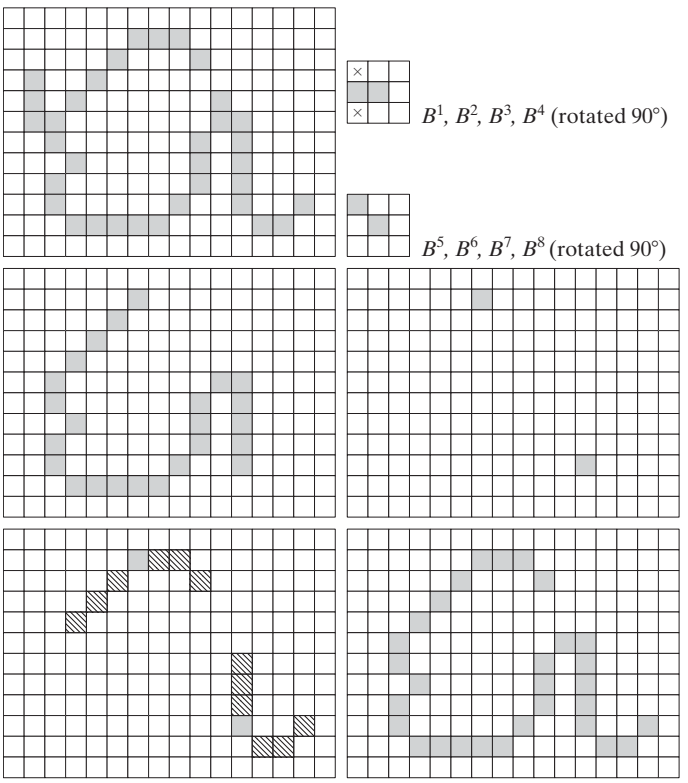
Figure 9.25(a) shows the skeleton of a hand-printed "a." The parasitic component on the leftmost part of the character is illustrative of what we are interested in removing. The solution is based on suppressing a parasitic branch by successively eliminating its end point. Of course, this also shortens (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

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).

$$X_1 = A \otimes \{B\} \tag{9.5-17}$$

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



a b
c
d e
f g

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)]:

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

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:

$$X_3 = (X_2 \oplus H) \cap A \tag{9.5-19}$$

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

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,

$$X_4 = X_1 \cup X_3 \tag{9.5-20}$$

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

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

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

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

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

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.

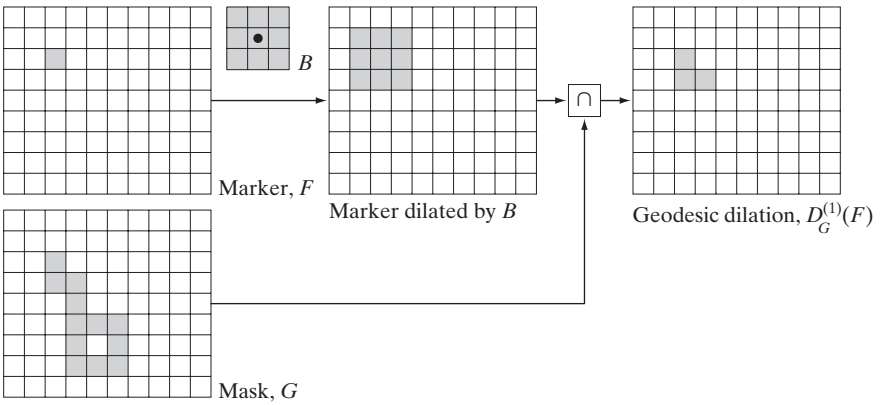


FIGURE 9.26 Illustration of geodesic dilation.

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

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

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

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

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).

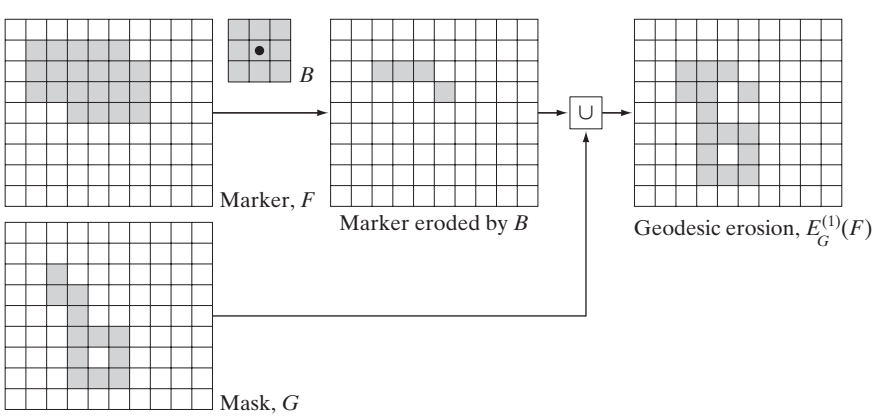


FIGURE 9.27 Illustration of geodesic erosion.

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,

$$R_G^D(F) = D_G^{(k)}(F) \tag{9.5-25}$$

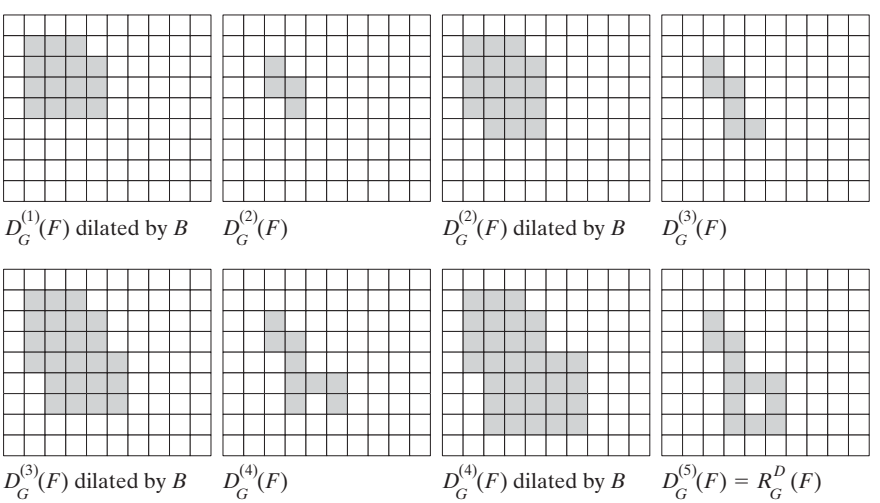
with k such that $D_G^{(k)}(F) = D_G^{(k+1)}(F)$.

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,

$$R_G^E(F) = E_G^{(k)}(F) \tag{9.5-26}$$

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

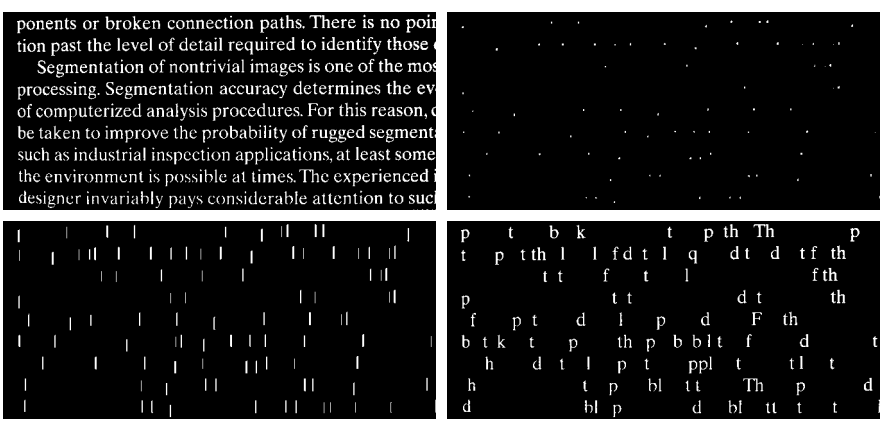
Morphological reconstruction has a broad spectrum of practical applications, each determined by the selection of the marker and mask images, by the structuring elements used, and by combinations of the primitive operations defined in the preceding discussion. The following examples illustrate the usefulness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

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

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



a b
c d

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,

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

Then

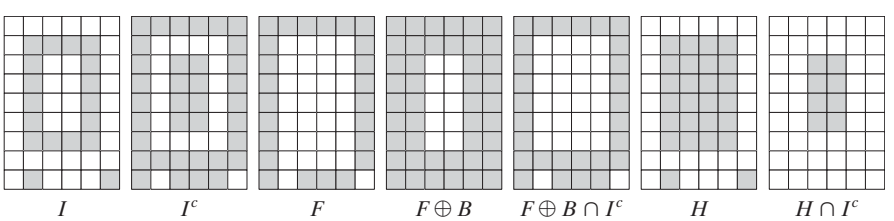
$$H = \left[R_{I^c}^D(F)\right]^c \tag{9.5-29}$$

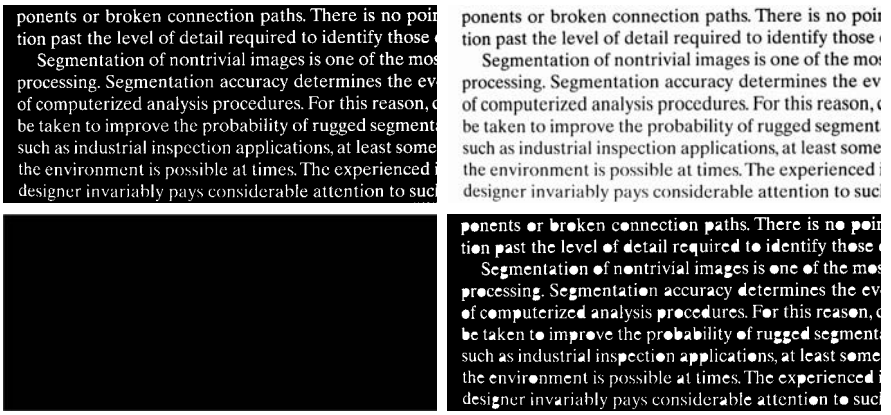
is a binary image equal to I with all holes filled.

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.

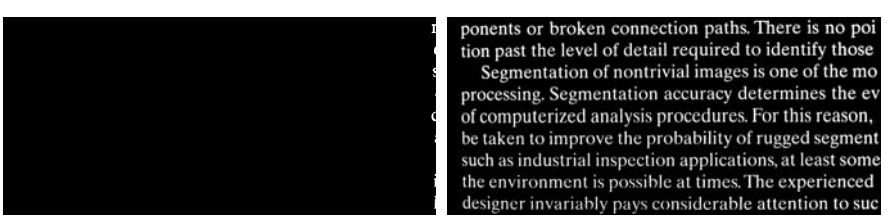
Border clearing: The extraction of objects from an image for subsequent shape analysis is a fundamental task in automated image processing. An algorithm for removing objects that touch (i.e., are connected to) the border is a useful tool because (1) it can be used to screen images so that only complete 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:

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

The border-Eclearing algorithm first computes the morphological reconstruction $R_I^D(F)$ (which simply extracts the objects touching the border) and then computes the difference

$$X = I - R_I^D(F) \tag{9.5-31}$$

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).

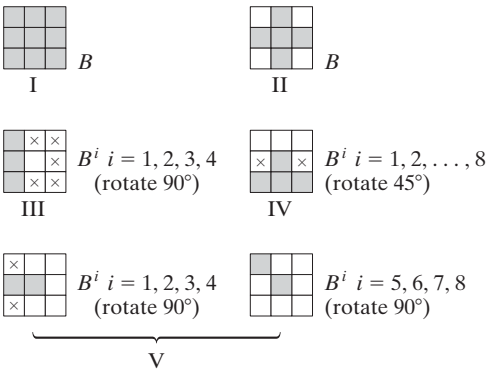


FIGURE 9.33 Five basic types of structuring elements used for binary morphology. The origin of each element is at its center and the ×'s indicate "don't care" values.

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	$(B)_z = \{w \mid w = b + z, \text{ for } b \in B\}$	Translates the origin of B to point z.
Reflection	$\hat{B} = \{w \mid w = -b, \text{ for } b \in B\}$	Reflects all elements of B about the origin of this set.
Complement	$A^c = \{w \mid w \notin A\}$	Set of points not in A.
Difference	$A - B = \{w \mid w \in A, w \notin B\} = A \cap B^c$	Set of points that belong to A but not to B.
Dilation	$A \oplus B = \left\{z \mid (\hat{B}_z) \cap A \neq \varnothing\right\}$	"Expands" the boundary of A. (I)
Erosion	$A \ominus B = \left\{z \mid (B)_z \subseteq A\right\}$	"Contracts" the boundary of A. (I)
Opening	$A \circ B = (A \ominus B) \oplus B$	Smoothes 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	$A \bullet B = (A \oplus B) \ominus B$	Smoothes contours, fuses narrow breaks and long thin gulfs, and eliminates small holes. (I)
Hit-or-miss transform	$A \circledast B = (A \ominus B_1) \cap (A^c \ominus B_2)$ $= (A \ominus B_1) - (A \oplus \hat{B}_2)$	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	$X_k = (X_{k-1} \oplus B) \cap A^c$; $k = 1, 2, 3, \ldots$	Fills holes in A; X_0 = array of 0s with a 1 in each hole. (II)
Connected components	$X_k = (X_{k-1} \oplus B) \cap A$; $k = 1, 2, 3, \ldots$	Finds connected components in A; X_0 = array of 0s with a 1 in each connected component. (I)
Convex hull	$X_k^i = (X_{k-1}^i \circledast B^i) \cup A$; $i = 1, 2, 3, 4$; $k = 1, 2, 3, \ldots$; $X_0^i = A$; 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	$A \otimes B = A - (A \circledast B)$ $= A \cap (A \circledast B)^c$ $A \otimes \{B\} =$ $((\ldots((A \otimes B^1) \otimes B^2)\ldots) \otimes B^n)$ $\{B\} = \{B^1, B^2, B^3, \ldots, B^n\}$	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	$A \odot B = A \cup (A \circledast B)$ $A \odot \{B\} =$ $((\ldots(A \odot B^1) \odot B^2 \ldots) \odot B^n)$	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 kth iteration of successive erosions of A by B. (I)

(*Continued*)

TABLE 9.1
(Continued)

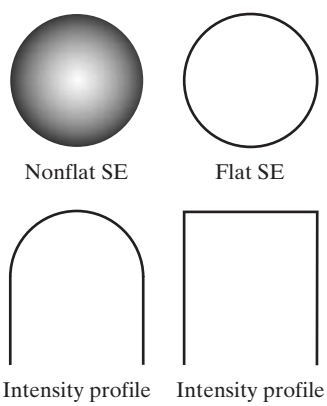
Operation	Equation	Comments (The Roman numerals refer to the structuring elements in Fig. 9.33.)
Pruning	$X_1 = A \otimes \{B\}$ $X_2 = \bigcup_{k=1}^{8} (X_1 \circledast B^k)$ $X_3 = (X_2 \oplus H) \cap A$ $X_4 = X_1 \cup X_3$	X_4 is the result of pruning set A. The number of times that the first equation 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 *marker* and *mask* images, respectively.
Geodesic dilation of size n	$D_G^{(n)}(F) = D_G^{(1)}\left[D_G^{(n-1)}(F)\right]$; $D_G^{(0)}(F) = F$	
Geodesic erosion of size 1	$E_G^{(1)}(F) = (F \ominus B) \cup G$	
Geodesic erosion of size n	$E_G^{(n)}(F) = E_G^{(1)}\left[E_G^{(n-1)}(F)\right]$; $E_G^{(0)}(F) = F$	
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	$H = \left[R_{I^c}^D(F)\right]^c$	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.

9.6 Gray-Scale Morphology

In this section, we extend to gray-scale images the basic operations of dilation, erosion, opening, and closing. We then use these operations to develop several basic gray-scale morphological algorithms.

Throughout the discussion that follows, we deal with digital functions of the form $f(x, y)$ and $b(x, y)$, where $f(x, y)$ is a gray-scale image and $b(x, y)$ is a structuring element. The assumption is that these functions are discrete in the sense introduced in Section 2.4.2. That is, if Z denotes the set of real integers, then the coordinates (x, y) are integers from the Cartesian product Z^2 and f and b are functions that assign an intensity value (a real number from the set of real numbers, R) to each distinct pair of coordinates (x, y). If the intensity levels are integers also, then Z replaces R.

Structuring elements in gray-scale morphology perform the same basic functions as their binary counterparts: They are used as "probes" to examine a given image for specific properties. Structuring elements in gray-scale morphology belong to one of two categories: *nonflat* and *flat*. Figure 9.34 shows an example of each. Figure 9.34(a) is a hemispherical gray-scale SE shown as an image, and Fig. 9.34(c) is a horizontal intensity profile through its center. Figure 9.34(b) shows a flat structuring element in the shape of a disk and Fig. 9.34(d) is its corresponding intensity profile (the shape of this profile explains the origin of the word "flat"). The elements in Fig. 9.34 are shown as continuous quantities for clarity; their computer implementation is based on digital approximations (e.g., see the rightmost disk SE in Fig. 9.2). Due to a number of difficulties discussed later in this section, gray-scale SEs are used infrequently in practice. Finally, we mention that, as in the binary case, the origin of structuring elements must be clearly identified. Unless mentioned otherwise, all the examples in this section are based on symmetrical, flat structuring elements of unit height whose origins are at the center. The *reflection* of an SE in gray-scale morphology is as defined in Section 9.1, and we denote it in the following discussion by $\hat{b}(x, y) = b(-x - y)$.



a b
c d

FIGURE 9.34 Nonflat and flat structuring elements, and corresponding horizontal intensity profiles through their center. All examples in this section are based on flat SEs.

9.6.1 Erosion and Dilation

The *erosion* of f by a *flat* structuring element b at any location (x, y) is defined as the *minimum* value of the image in the region coincident with b when the origin of b is at (x, y). In equation form, the erosion at (x, y) of an image f by a structuring element b is given by

$$\big[f \ominus b\big](x, y) = \min_{(s,t)\in b}\big\{f(x + s, y + t)\big\} \tag{9.6-1}$$

where, in a manner similar to the correlation procedure discussed in Section 3.4.2, x and y are incremented through all values required so that the origin of b visits every pixel in f. That is, to find the erosion of f by b, we place the origin of the structuring element at every pixel location in the image. The erosion at any location is determined by selecting the minimum value of f from all the values of f contained in the region coincident with b. For example, if b is a square structuring element of size 3×3, obtaining the erosion at a point requires finding the minimum of the nine values of f contained in the 3×3 region defined by b when its origin is at that point.

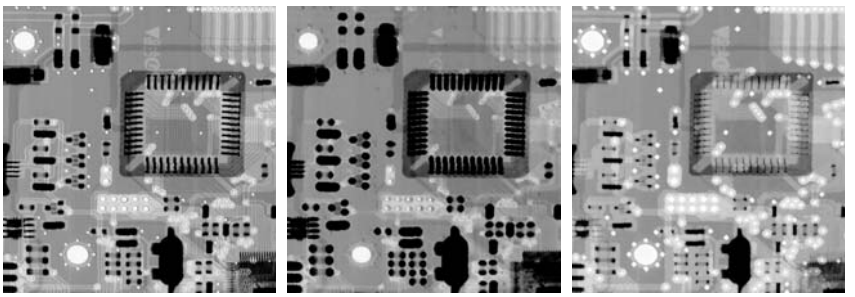
Similarly, the *dilation* of f by a *flat* structuring element b at any location (x, y) is defined as the *maximum* value of the image in the window outlined by $\hat{b}$ when the origin of $\hat{b}$ is at (x, y). That is,

$$\big[f \oplus b\big](x, y) = \max_{(s,t)\in b}\big\{f(x - s, y - t)\big\} \tag{9.6-2}$$

where we used the fact stated earlier that $\hat{b} = b(-x, -y)$. The explanation of this equation is identical to the explanation in the previous paragraph, but using the maximum, rather than the minimum, operation and keeping in mind that the structuring element is reflected about its origin, which we take into account by using $(-s, -t)$ in the argument of the function. This is analogous to spatial convolution, as explained in Section 3.4.2.

EXAMPLE 9.9: Illustration of gray-scale erosion and dilation.

■ Because gray-scale erosion with a flat SE computes the minimum intensity value of f in every neighborhood of (x, y) coincident with b, we expect in general that an eroded gray-scale image will be darker than the original, that the sizes (with respect to the size of the SE) of bright features will be reduced, and that the sizes of dark features will be increased. Figure 9.35(b) shows the erosion of Fig. 9.35(a) using a disk SE of unit height and a radius of two pixels. The effects just mentioned are clearly visible in the eroded image. For instance, note how the intensities of the small bright dots were reduced, making them barely visible in Fig. 9.35(b), while the dark features grew in thickness. The general background of the eroded image is slightly darker than the background of the original image. Similarly, Fig. 9.35(c) shows the result of dilation with the same SE. The effects are the opposite of those obtained with erosion. The bright features were thickened and the intensities of the dark features were reduced. Note in particular how the thin black connecting wires in the left, middle, and right, bottom of Fig. 9.35(a) are barely visible in Fig. 9.35(c). The sizes of the dark dots were reduced as a result of dilation but, unlike the eroded small white dots in Fig. 9.35(b), they still are easily visible in the dilated



a b c

FIGURE 9.35 (a) A gray-scale X-ray image of size 448 × 425 pixels. (b) Erosion using a flat disk SE with a radius of two pixels. (c) Dilation using the same SE. (Original image courtesy of Lixi, Inc.)

image. The reason is that the black dots were originally larger than the white dots with respect to the size of the SE. Finally, note that the background of the dilated image is slightly lighter than that of Fig. 9.35(a). ■

Nonflat SEs have gray-scale values that vary over their domain of definition. The erosion of image f by nonflat structuring element, b_N, is defined as

$$\left[f \ominus b_N\right](x, y) = \min_{(s,t)\in b_N}\left\{f(x + s, y + t) - b_N(s, t)\right\} \tag{9.6-3}$$

Here, we actually subtract values from f to determine the erosion at any point. This means that, unlike Eq. (9.6-1), erosion using a nonflat SE is not bounded in general by the values of f, which can present problems in interpreting results. Gray-scale SEs are seldom used in practice because of this, in addition to potential difficulties in selecting meaningful elements for b_N, and the added computational burden when compared with Eq. (9.6-1).

In a similar manner, dilation using a nonflat SE is defined as

$$\left[f \oplus b_N\right](x, y) = \max_{(s,t)\in b_N}\left\{f(x - s, y - t) + b_N(s, t)\right\} \tag{9.6-4}$$

The same comments made in the previous paragraph are applicable to dilation with nonflat SEs. When all the elements of b_N are constant (i.e., the SE is flat), Eqs. (9.6-3) and (9.6-4) reduce to Eqs. (9.6-1) and (9.6-2), respectively, within a scalar constant equal to the amplitude of the SE.

As in the binary case, erosion and dilation are duals with respect to function complementation and reflection; that is,

$$(f \ominus b)^c(x, y) = (f^c \oplus \hat{b})(x, y)$$

where $f^c = -f(x, y)$ and $\hat{b} = b(-x, -y)$. The same expression holds for nonflat structuring elements. Except as needed for clarity, we simplify the notation in the following discussion by omitting the arguments of all functions, in which case the preceding equation is written as

$$(f \ominus b)^c = (f^c \oplus \hat{b}) \tag{9.6-5}$$

Similarly,

$$(f \oplus b)^c = (f^c \ominus \hat{b}) \tag{9.6-6}$$

Erosion and dilation by themselves are not particularly useful in gray-scale image processing. As with their binary counterparts, these operations become powerful when used in combination to derive higher-level algorithms, as the material in the following sections demonstrates.

Although we deal with flat SEs in the examples in the remainder of this section, the concepts discussed are applicable also to nonflat structuring elements.

9.6.2 Opening and Closing

The expressions for opening and closing gray-scale images have the same form as their binary counterparts. The *opening* of image f by structuring element b, denoted $f \circ b$, is

$$f \circ b = (f \ominus b) \oplus b \tag{9.6-7}$$

As before, opening is simply the erosion of f by b, followed by a dilation of the result with b. Similarly, the *closing* of f by b, denoted $f \cdot b$, is

$$f \bullet b = (f \oplus b) \ominus b \tag{9.6-8}$$

The opening and closing for gray-scale images are duals with respect to complementation and SE reflection:

$$(f \bullet b)^c = f^c \circ \hat{b} \tag{9.6-9}$$

and

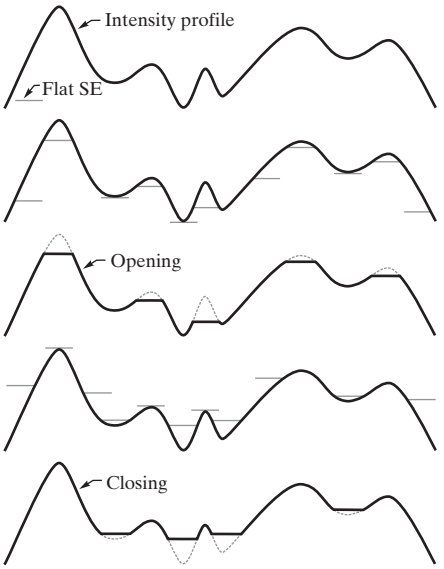
$$(f \circ b)^c = f^c \bullet \hat{b} \tag{9.6-10}$$

Because $f^c = -f(x, y)$, Eq. (9.6-9) can be written also as $-(f \bullet b) = (-f \circ \hat{b})$ and similarly for Eq. (9.6-10).

Opening and closing of images have a simple geometric interpretation. Suppose that an image function $f(x, y)$ is viewed as a 3-D surface; that is, its intensity values are interpreted as height values over the xy-plane, as in Fig. 2.18(a). Then the opening of f by b can be interpreted geometrically as pushing the structuring element up from below against the undersurface of f. At each location of the origin of b, the opening is the highest value reached by any part of b as it pushes up against the undersurface of f. The complete opening is then the set of all such values obtained by having the origin of b visit every (x, y) coordinate of f.

Sometimes opening and closing are illustrated by rolling a circle on the under and upper sides of a curve. In 3-D, the circle becomes a sphere and the resulting procedures are called *rolling-ball* algorithms.

Figure 9.36 illustrates the concept in one dimension. Suppose that the curve in Fig. 9.36(a) is the intensity profile along a single row of an image. Figure 9.36(b) shows a flat structuring element in several positions, pushed up against the bottom of the curve. The solid curve in Fig. 9.36(c) is the complete opening. Because the structuring element is too large to fit completely inside the upward peaks of the curve, the tops of the peaks are clipped by the opening, with the amount removed being proportional to how far the structuring element was able to reach into the peak. In general, openings are used to remove small, bright details, while leaving the overall intensity levels and larger bright features relatively undisturbed.



a
b
c
d
e

FIGURE 9.36 Opening and closing in one dimension. (a) Original 1-D signal. (b) Flat structuring element pushed up underneath the signal. (c) Opening. (d) Flat structuring element pushed down along the top of the signal. (e) Closing.

Figure 9.36(d) is a graphical illustration of closing. Observe that the structuring element is pushed down on top of the curve while being translated to all locations. The closing, shown in Fig. 9.36(e), is constructed by finding the lowest points reached by any part of the structuring element as it slides against the upper side of the curve.

The gray-scale opening operation satisfies the following properties:

(a) $f \circ b \leftarrow\!\lrcorner f$
(b) If $f_1 \leftarrow\!\lrcorner f_2$, then $(f_1 \circ b) \leftarrow\!\lrcorner (f_2 \circ b)$
(c) $(f \circ b) \circ b = f \circ b$

The notation $e \leftarrow\!\lrcorner r$ is used to indicate that the domain of e is a subset of the domain of r, and also that $e(x, y) \leq r(x, y)$ for any (x, y) in the domain of e.

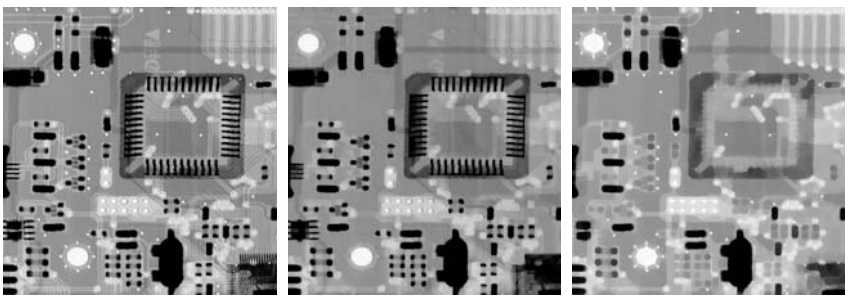
Similarly, the closing operation satisfies the following properties:

(a) $f \leftarrow\!\lrcorner f \bullet b$
(b) If $f_1 \leftarrow\!\lrcorner f_2$, then $(f_1 \bullet b) \leftarrow\!\lrcorner (f_2 \bullet b)$
(c) $(f \bullet b) \bullet b = f \bullet b$

The usefulness of these properties is similar to that of their binary counterparts.

EXAMPLE 9.10: Illustration of gray-scale opening and closing.

■ Figure 9.37 extends to 2-D the 1-D concepts illustrated in Fig. 9.36. Figure 9.37(a) is the same image we used in Example 9.9, and Fig. 9.37(b) is the opening obtained using a disk structuring element of unit height and radius of 3 pixels. As expected, the intensity of all bright features decreased, depending on the sizes of the features relative to the size of the SE. Comparing this figure with Fig. 9.35(b), we see that, unlike the result of erosion, opening had negligible effect on the dark features of the image, and the effect on the background was negligible. Similarly, Fig. 9.37(c) shows the closing of the image with a disk of radius 5 (the small round



a b c

FIGURE 9.37 (a) A gray-scale X-ray image of size 448 × 425 pixels. (b) Opening using a disk SE with a radius of 3 pixels. (c) Closing using an SE of radius 5.

black dots are larger than the small white dots, so a larger disk was needed to achieve results comparable to the opening). In this image, the bright details and background were relatively unaffected, but the dark features were attenuated, with the degree of attenuation being dependent on the relative sizes of the features with respect to the SE. ■

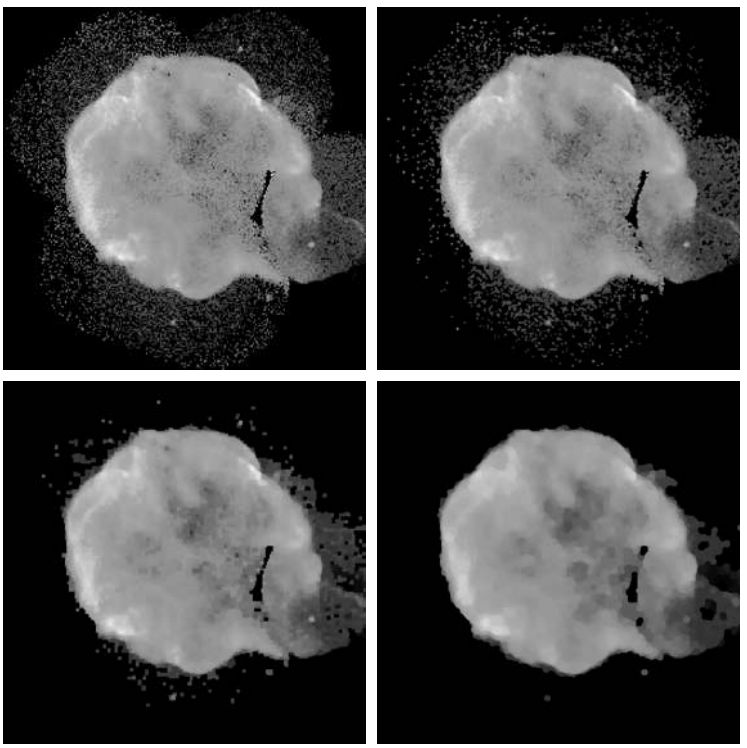
9.6.3 Some Basic Gray-Scale Morphological Algorithms

Numerous morphological techniques are based on the gray-scale morphological concepts introduced thus far. We illustrate some of these algorithms in the following discussion.

Morphological smoothing

Because opening suppresses bright details smaller than the specified SE, and closing suppresses dark details, they are used often in combination as *morphological filters* for image smoothing and noise removal. Consider Fig. 9.38(a), which shows an image of the Cygnus Loop supernova taken in the X-ray band (see Fig. 1.7 for details about this image). For purposes of the present discussion, suppose that the central light region is the object of interest and that the smaller components are noise. The objective is to remove the noise. Figure 9.38(b) shows the result of opening the original image with a flat disk of radius 2 and then closing the opening with an SE of the same size. Figures 9.38(c) and (d) show the results of the same operation using SEs of radii 3 and 5, respectively. As expected, this sequence shows progressive removal of small components as a function of SE size. In the last result, we see that the object of interest has been extracted. The noise components on the lower side of the image could not be removed completely because of their density.

The results in Fig. 9.38 are based on opening the original image and then closing the opening. A procedure used sometimes is to perform *alternating sequential filtering*, in which the opening–closing sequence starts with the original image, but subsequent steps perform the opening and closing on the results



a b
c d

FIGURE 9.38
(a) 566 × 566 image of the Cygnus Loop supernova, taken in the X-ray band by NASA's Hubble Telescope. (b)–(d) Results of performing opening and closing sequences on the original image with disk structuring elements of radii, 1, 3, and 5, respectively. (Original image courtesy of NASA.)

of the previous step. This type of filtering is useful in automated image analysis, in which results at each step are compared against a specified metric. Generally, this approach produces more blurring for the same size SE than the method illustrated in Fig. 9.38.

Morphological gradient

Dilation and erosion can be used in combination with image subtraction to obtain the morphological gradient of an image, denoted by g, where

See Section 3.6.4 for a definition of the image gradient.

$$g = (f \oplus b) - (f \ominus b) \tag{9.6-11}$$

The dilation thickens regions in an image and the erosion shrinks them. Their difference emphasizes the boundaries between regions. Homogenous areas are not affected (as long as the SE is relatively small) so the subtraction operation tends to eliminate them. The net result is an image in which the edges are enhanced and the contribution of the homogeneous areas are suppressed, thus producing a "derivative-like" (gradient) effect.

Figure 9.39 shows an example. Figure 9.39(a) is a head CT scan, and the next two figures are the opening and closing with a 3 × 3 SE of all 1s. Note the thickening and shrinking just mentioned. Figure 9.39(d) is the morphological gradient obtained using Eq. (9.6-11), in which the boundaries between regions are clearly delineated, as expected of a 2-D derivative image.

a b
c d

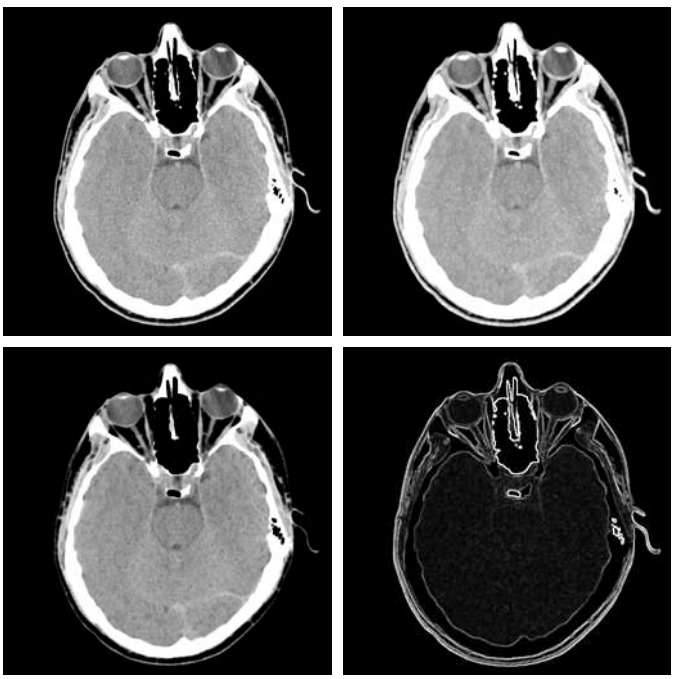


FIGURE 9.39
(a) 512 × 512 image of a head CT scan. (b) Dilation. (c) Erosion. (d) Morphological gradient, computed as the difference between (b) and (c). (Original image courtesy of Dr. David R. Pickens, Vanderbilt University.)

Top-hat and bottom-hat transformations

Combining image subtraction with openings and closings results in so-called *top-hat and bottom-hat* transformations. The *top-hat transformation* of a grayscale image f is defined as f minus its opening:

$$T_{\text{hat}}(f) = f - (f \circ b) \tag{9.6-12}$$

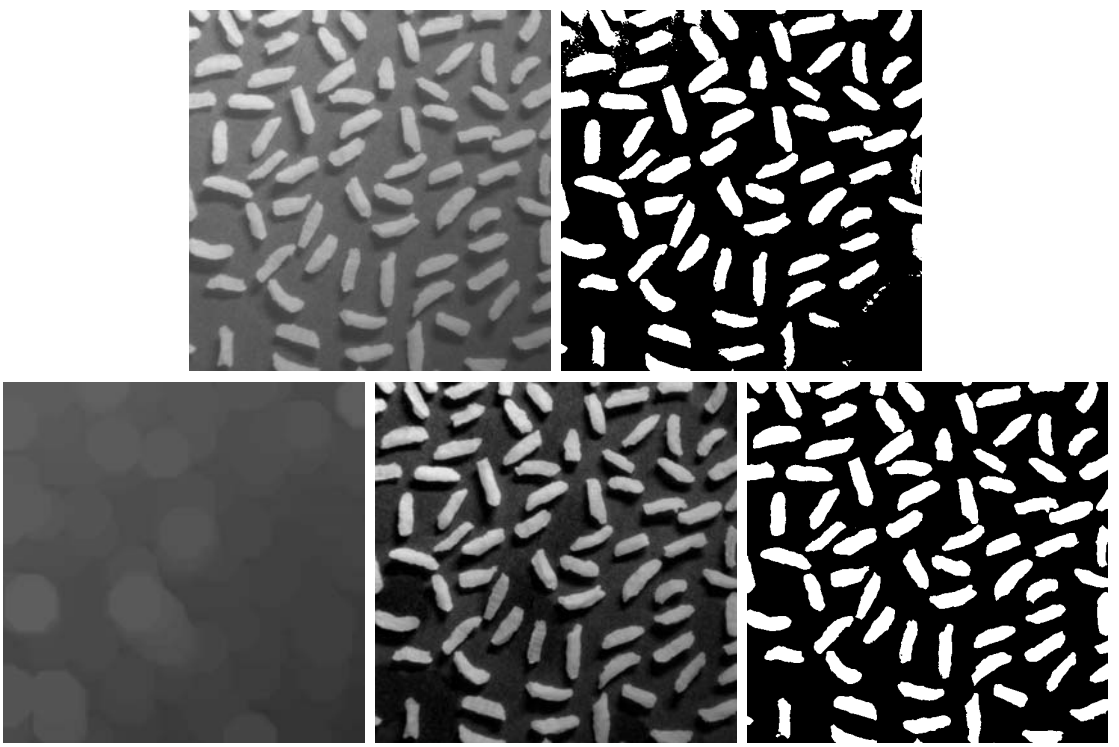
Similarly, the *bottom-hat transformation* of f is defined as the closing of f minus f:

$$B_{\text{hat}}(f) = (f \bullet b) - f \tag{9.6-13}$$

One of the principal applications of these transformations is in removing objects from an image by using a structuring element in the opening or closing operation that does not fit the objects to be removed. The difference operation then yields an image in which only the removed components remain. The top-hat transform is used for light objects on a dark background, and the bottom-hat transform is used for the converse. For this reason, the names *white top-hat* and *black top-hat*, respectively, are used frequently when referring to these two transformations.

An important use of top-hat transformations is in correcting the effects of nonuniform illumination. As we will see in the next chapter, proper (uniform) illumination plays a central role in the process of extracting objects from the background. This process, called *segmentation*, is one of the first steps performed in automated image analysis. A commonly used segmentation approach is to threshold the input image.

To illustrate, consider Fig. 9.40(a), which shows a 600 × 600 image of grains of rice. This image was obtained under nonuniform lighting, as evidenced by the darker area in the bottom, rightmost part of the image. Figure 9.40(b) shows the result of thresholding using Otsu's method, an optimal thresholding method discussed in Section 10.3.3. The net result of nonuniform illumination was to cause segmentation errors in the dark area (several grains of rice were not extracted from the background), as well as in the top left part of the image, where parts of the background were misclassified. Figure 9.40(c) shows the opening of the image with a disk of radius 40. This SE was large enough so that it would not fit in any of the objects. As a result, the objects were eliminated, leaving only an approximation of the background. The shading pattern is clear in this image. By subtracting this image from the original (i.e., performing a top-hat transformation), the background should become more uniform. This is indeed the case, as Fig. 9.40(d) shows. The background is not perfectly uniform, but the differences between light and dark extremes are less, and this was enough to yield a correct



a b
c d e

FIGURE 9.40 Using the top-hat transformation for *shading correction*. (a) Original image of size 600 × 600 pixels. (b) Thresholded image. (c) Image opened using a disk SE of radius 40. (d) Top-hat transformation (the image minus its opening). (e) Thresholded top-hat image.

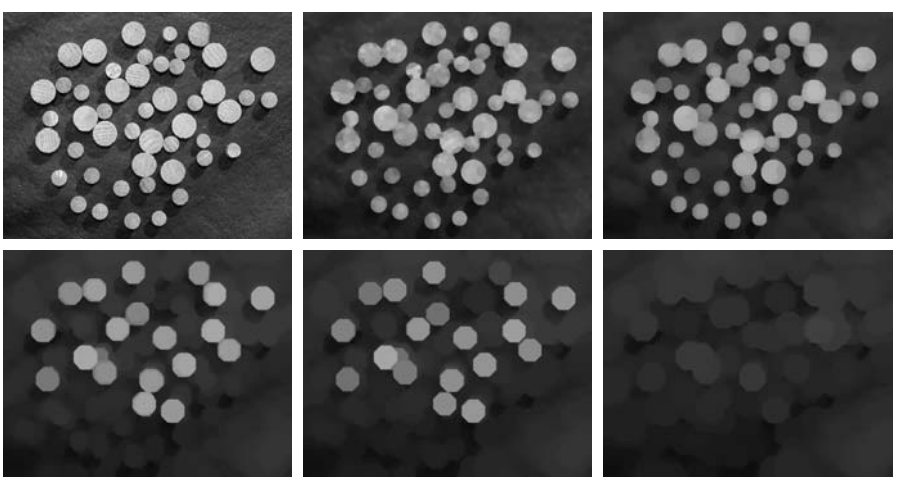
segmentation result in which all rice grains were detected, as Fig. 9.40(e) shows. This image was obtained using Otsu's method, as before.

Granulometry

In terms of image processing, *granulometry* is a field that deals with determining the size distribution of particles in an image. In practice, particles seldom are neatly separated, which makes particle counting by identifying individual particles a difficult task. Morphology can be used to estimate particle size distribution indirectly, without having to identify and measure every particle in the image.

The approach is simple in principle. With particles having regular shapes that are lighter than the background, the method consists of applying openings with SEs of increasing size. The basic idea is that opening operations of a particular size should have the most effect on regions of the input image that contain particles of similar size. For each opening, the *sum* of the pixel values in the opening is computed. This sum, sometimes called the *surface area*, decreases as a function of increasing SE size because, as we noted earlier, openings decrease the intensity of light features. This procedure yields a 1-D array of such numbers, with each element in the array being equal to the sum of the pixels in the opening for the size SE corresponding to that location in the array. To emphasize changes between successive openings, we compute the difference between adjacent elements of the 1-D array. To visualize the results, the differences are plotted. The peaks in the plot are an indication of the predominant size distributions of the particles in the image.

As an example, consider Fig. 9.41(a) which is an image of wood dowel plugs of two dominant sizes. The wood grain in the dowels are likely to introduce variations in the openings, so smoothing is a sensible pre-processing step. Figure 9.41(b) shows the image smoothed using the morphological smoothing



a b c
d e f

FIGURE 9.41 (a) 531 × 675 image of wood dowels. (b) Smoothed image. (c)–(f) Openings of (b) with disks of radii equal to 10, 20, 25, and 30 pixels, respectively. (Original image courtesy of Dr. Steve Eddins, The MathWorks, Inc.)

filter discussed earlier, with a disk of radius 5. Figures 9.41(c) through (f) show examples of image openings with disks of radii 10, 20, 25, and 30. Note in Fig. 9.41(d) that the intensity contribution due to the small dowels has been almost eliminated. In Fig. 9.41(e) the contribution of the large dowels has been significantly reduced, and in Fig. 9.41(f) even more so. (Observe in Fig. 9.41(e) that the large dowel near the top right of the image is much darker than the others because of its smaller size. This would be useful information if we had been attempting to detect defective dowels.)

Figure 9.42 shows a plot of the difference array. As mentioned previously, we expect significant differences (peaks in the plot) around radii at which the SE is large enough to encompass a set of particles of approximately the same diameter. The result in Fig. 9.42 has two distinct peaks, clearly indicating the presence of two dominant object sizes in the image.

Textural segmentation

Figure 9.43(a) shows a noisy image of dark blobs superimposed on a light background. The image has two textural regions: a region composed on large blobs on the right and a region on the left composed of smaller blobs. The objective is to find a boundary between the two regions based on their textural content (we discuss texture in Section 11.3.3). As noted earlier, the process of subdividing an image into regions is called *segmentation*, which is the topic of Chapter 10.

The objects of interest are darker than the background, and we know that if we close the image with a structuring element larger than the small blobs, these blobs will be removed. The result in Fig. 9.43(b), obtained by closing the input image using a disk with a radius of 30 pixels, shows that indeed this is the case (the radius of the blobs is approximately 25 pixels). So, at this point, we have an image with large, dark blobs on a light background. If we *open* this image with a structuring element that is large relative to the separation between these blobs, the net result should be an image in which the light patches between the blobs are removed, leaving the dark blobs and now equally dark patches between these blobs. Figure 9.43(c) shows the result, obtained using a disk of radius 60.

Performing a morphological gradient on this image with, say, a 3×3 SE of 1s, will give us the boundary between the two regions. Figure 9.43(d) shows the boundary obtained from the morphological gradient operation superimposed

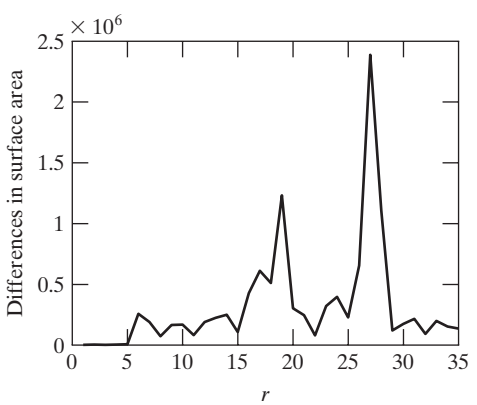


FIGURE 9.42 Differences in surface area as a function of SE disk radius, r. The two peaks are indicative of two dominant particle sizes in the image.

a b
c d

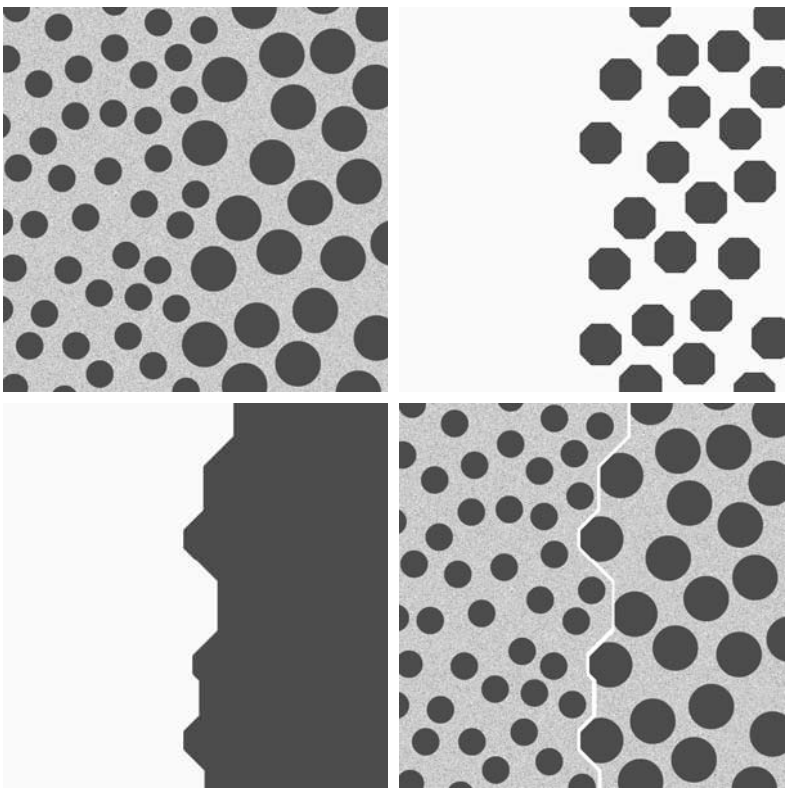


FIGURE 9.43 Textural segmentation. (a) A 600 × 600 image consisting of two types of blobs. (b) Image with small blobs removed by closing (a). (c) Image with light patches between large blobs removed by opening (b). (d) Original image with boundary between the two regions in (c) superimposed. The boundary was obtained using a morphological gradient operation.

on the original image. All pixels to the right of this boundary are said to belong to the texture region characterized by large blobs, and conversely for the pixels on the left of the boundary. You will find it instructive to work through this example in more detail using the graphical analogy for opening and closing illustrated in Fig. 9.36.

9.6.4 Gray-Scale Morphological Reconstruction

Gray-scale morphological reconstruction is defined basically in the same manner introduced in Section 9.5.9 for binary images. Let f and g denote the *marker* and *mask* images, respectively. We assume that both are gray-scale images of the same size and that $f \leq g$. The *geodesic dilation* of size 1 of f with respect to g is defined as

It is understood that these expressions are functions of (x, y). We omit the coordinates to simplify the notation.

$$D_g^{(1)}(f) = (f \oplus b) \wedge g \tag{9.6-14}$$

where $\wedge$ denotes the point-wise minimum operator. This equation indicates that the geodesic dilation of size 1 is obtained by first computing the dilation of f by b and then selecting the minimum between the result and g at every point (x,y). The dilation is given by Eq. (9.6-2) if b is a flat SE or by Eq. (9.6-4) if it is not. The geodesic dilation of size n of f with respect to g is defined as

$$D_g^{(n)}(f) = D_g^{(1)}[D_g^{(n-1)}(f)] \tag{9.6-15}$$

with $D_g^{(0)}(f) = f$.

Similarly, the *geodesic erosion* of size 1 of f with respect to g is defined as

$$E_g^{(1)}(f) = (f \ominus b) \vee g \tag{9.6-16}$$

where $\vee$ denotes the point-wise maximum operator. The geodesic erosion of size n is defined as

$$E_g^{(n)}(f) = E_g^{(1)}\left[E_g^{(n-1)}(f)\right] \tag{9.6-17}$$

with $E_g^{(0)}(f) = f$.

See Problem 9.33 for a list of dual relationships between expressions in this section.

The *morphological reconstruction by dilation* of a gray-scale mask image, g, by a gray-scale marker image, f, is defined as the geodesic dilation of f with respect to g, iterated until stability is reached; that is,

$$R_g^D(f) = D_g^{(k)}(f) \tag{9.6-18}$$

with k *such* that $D_g^{(k)}(f) = D_g^{(k+1)}(f)$. The *morphological reconstruction by erosion* of g by f is similarly defined as

$$R_g^E(f) = E_g^{(k)}(f) \tag{9.6-19}$$

with k *such* that $E_g^{(k)}(f) = E_g^{(k+1)}(f)$.

As in the binary case, opening by reconstruction of gray-scale images first erodes the input image and uses it as a marker. 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,

$$O_R^{(n)}(f) = R_f^D\left[(f \ominus nb)\right] \tag{9.6-20}$$

where $(f \ominus nb)$ denotes n erosions of f by b, as explained in Section 9.5.7. Recall from the discussion of Eq. (9.5-27) for binary images that the objective of opening by reconstruction is to preserve the shape of the image components that remain after erosion.

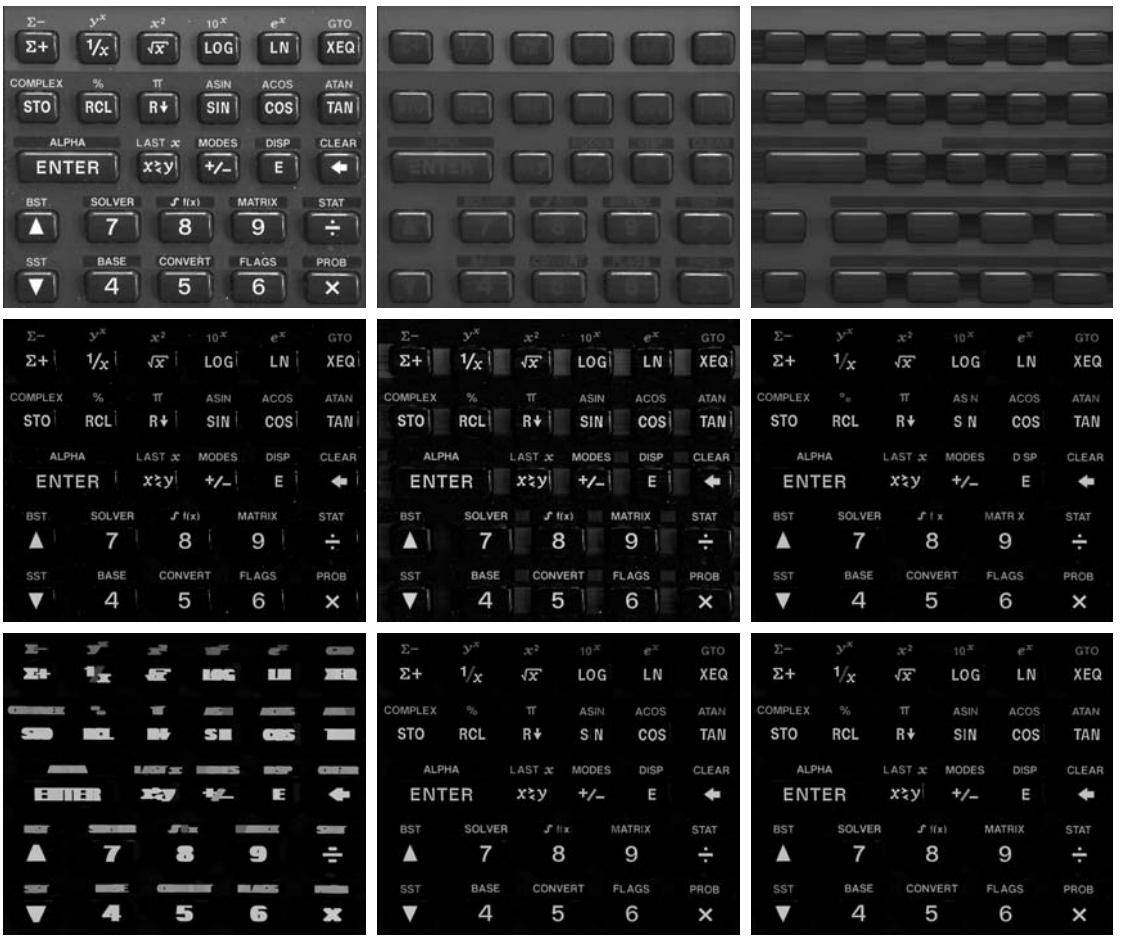
Similarly, the *closing by reconstruction* of size n of an image f is defined as the reconstruction by erosion of f from the dilation of size n of f; that is,

$$C_R^{(n)}(f) = R_f^E\left[(f \oplus nb)\right] \tag{9.6-21}$$

where $(f \oplus nb)$ denotes n dilations of f by b. Because of duality, the closing by reconstruction of an image can be obtained by complementing the image, obtaining the opening by reconstruction, and complementing the result. Finally, as the following example shows, a useful technique called *top-hat by reconstruction* consists of subtracting from an image its opening by reconstruction.

EXAMPLE 9.11: Using morphological reconstruction to flatten a complex background.

■ In this example, we illustrate the use of gray-scale reconstruction in several steps to normalize the irregular background of the image in Fig. 9.44(a), leaving only the text on a background of constant intensity. The solution of this problem is a good illustration of the power of morphological concepts. We begin by suppressing the horizontal reflection on the top of the keys. The reflections are wider than any single character in the image, so we should be able to suppress them by performing an opening by reconstruction using a long horizontal line in the erosion operation. This operation will yield the background containing the keys and their reflections. Subtracting this from



a b c
d e f
g h i

FIGURE 9.44 (a) Original image of size 1134 × 1360 pixels. (b) Opening by reconstruction of (a) using a horizontal line 71 pixels long in the erosion. (c) Opening of (a) using the same line. (d) Top-hat by reconstruction. (e) Top-hat. (f) Opening by reconstruction of (d) using a horizontal line 11 pixels long. (g) Dilation of (f) using a horizontal line 21 pixels long. (h) Minimum of (d) and (g). (i) Final reconstruction result. (Images courtesy of Dr. Steve Eddins, The MathWorks, Inc.)

the original image (i.e., performing a top-hat by reconstruction) will eliminate the horizontal reflections and variations in background from the original image.

Figure 9.44(b) shows the result of opening by reconstruction of the original image using a horizontal line of size 1 × 71 pixels in the erosion operation. We could have used just an opening to remove the characters, but the resulting background would not have been as uniform, as Fig. 9.44(c) shows (for example, compare the regions between the keys in the two images). Figure 9.44(d)

shows the result of subtracting Fig. 9.44(b) from Fig. 9.44(a). As expected, the horizontal reflections and variations in background were suppressed. For comparison, Fig. 9.44(e) shows the result of performing just a top-hat transformation (i.e., subtracting the "standard" opening from the image, as discussed earlier in this section). As expected from the characteristics of the background in Fig. 9.44(c), the background in Fig. 9.44(e) is not nearly as uniform as in Fig. 9.44(d).

The next step is to remove the vertical reflections from the edges of keys, which are quite visible in Fig. 9.44(d). We can do this by performing an opening by reconstruction with a line SE whose width is approximately equal to the reflections (about 11 pixels in this case). Figure 9.44(f) shows the result of performing this operation on Fig. 9.44(d). The vertical reflections were suppressed, but so were thin, vertical strokes that are valid characters (for example, the I in SIN), so we have to find a way to restore the latter. The suppressed characters are very close to the other characters so, if we dilate the remaining characters horizontally, the dilated characters will overlap the area previously occupied by the suppressed characters. Figure 9.44(g), obtained by dilating Fig. 9.44(f) with a line SE of size 1×21, shows that indeed this is case.

All that remains at this point is to restore the suppressed characters. Consider an image formed as the point-wise minimum between the dilated image in Fig. 9.44(g) and the top-hat by reconstruction in Fig. 9.44(d). Figure 9.44(h) shows the minimum image (although this result appears to be close to our objective, note that the I in SIN is still missing). By using this image as a marker and the dilated image as the mask in gray-scale reconstruction [Eq. (9.6-18)] we obtain the final result in Fig. 9.44(i). This image shows that all characters were properly extracted from the original, irregular background, including the background of the keys. The background in Fig. 9.44(i) is uniform throughout. ■

Summary

The morphological concepts and techniques introduced in this chapter constitute a powerful set of tools for extracting features of interest in an image. One of the most appealing aspects of morphological image processing is the extensive set-theoretic foundation from which morphological techniques have evolved. A significant advantage in terms of implementation is the fact that dilation and erosion are primitive operations that are the basis for a broad class of morphological algorithms. As shown in the following chapter, morphology can be used as the basis for developing image segmentation procedures with numerous applications. As discussed in Chapter 11, morphological techniques also play a major role in procedures for image description.

References and Further Reading

The book by Serra [1982] is a fundamental reference on morphological image processing. See also Serra [1988], Giardina and Dougherty [1988], and Haralick and Shapiro [1992]. Additional early references relevant to our discussion include Blum [1967], Lantuéjoul [1980], Maragos [1987], and Haralick et al. [1987]. For an overview of both binary and gray-scale morphology, see Basart and Gonzalez [1992] and Basart et al.

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.

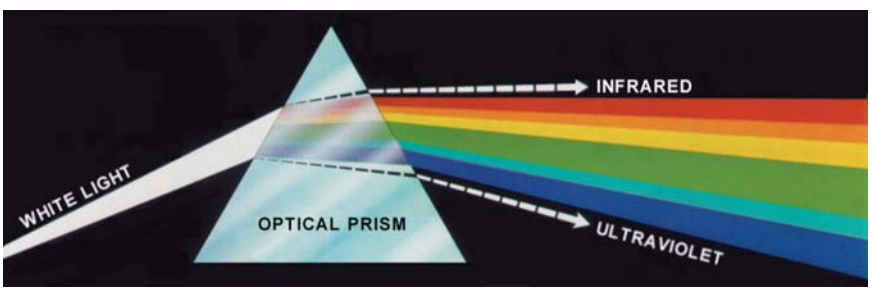


FIGURE 6.1 Color spectrum seen by passing white light through a prism. (Courtesy of the General Electric Co., Lamp Business Division.)

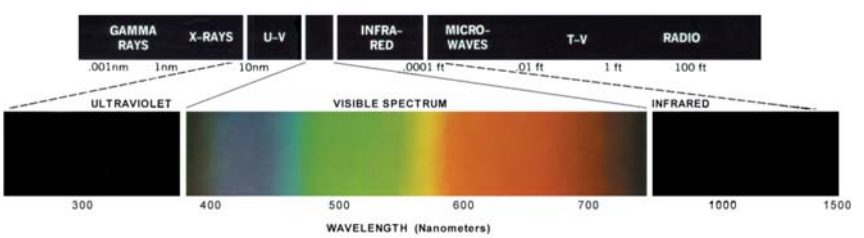


FIGURE 6.2 Wavelengths comprising the visible range of the electromagnetic spectrum. (Courtesy of the General Electric Co., Lamp Business Division.)

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

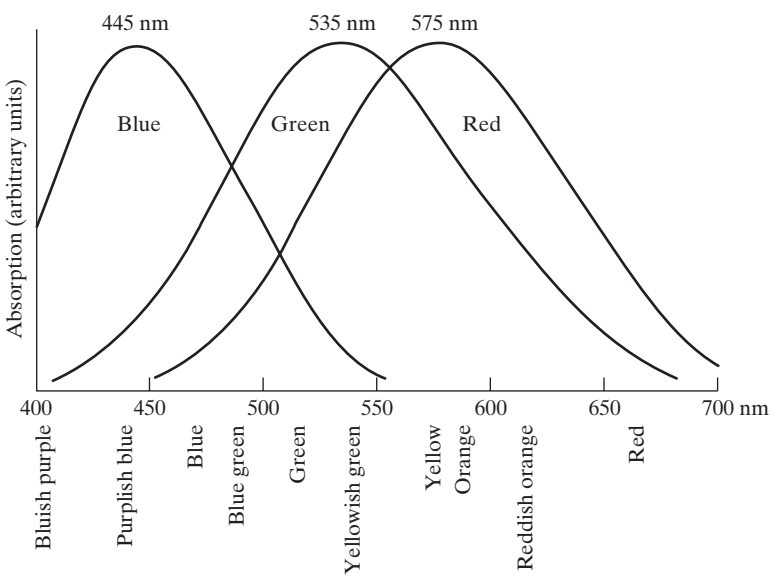
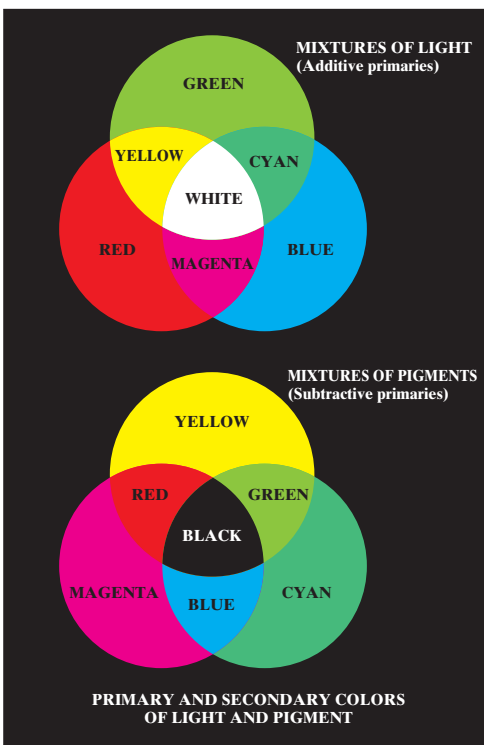


FIGURE 6.3 Absorption of light by the red, green, and blue cones in the human eye as a function of wavelength.

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

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

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

and

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

It is noted from these equations that[†]

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

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

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

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

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

[†]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.

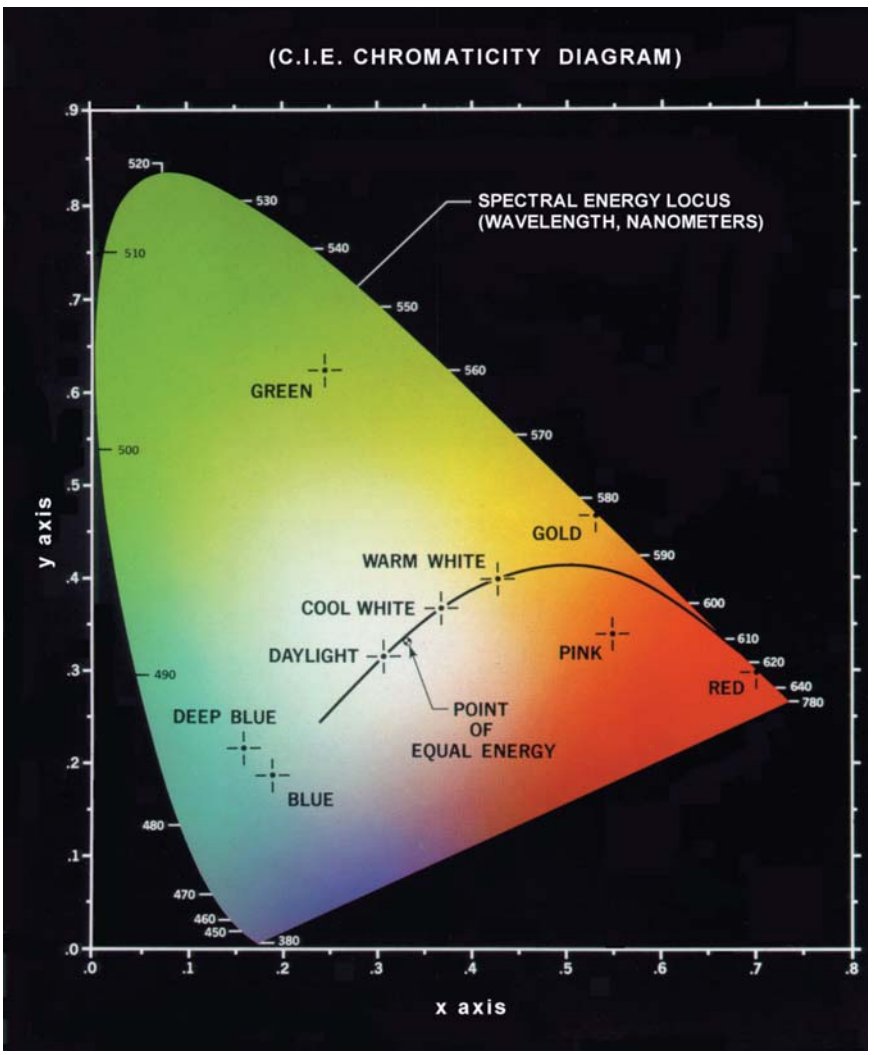


FIGURE 6.5 Chromaticity diagram. (Courtesy of the General Electric Co., Lamp Business Division.)

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

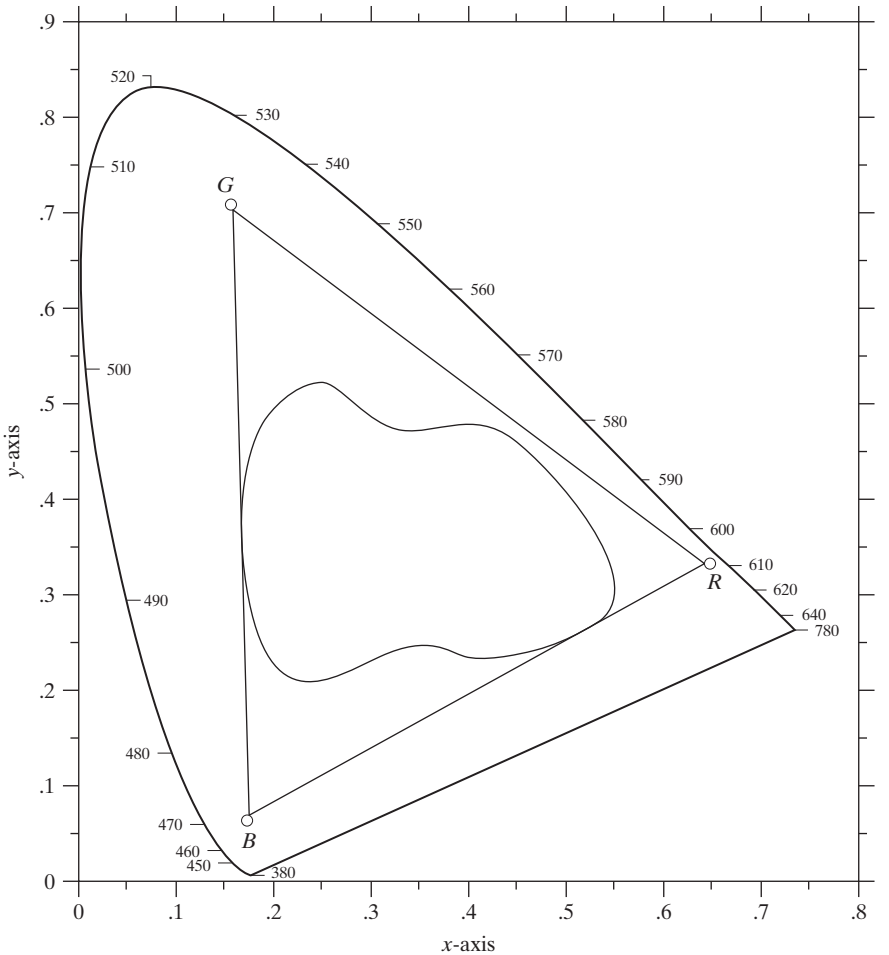


FIGURE 6.6
Typical color gamut of color monitors (triangle) and color printing devices (irregular region).

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 $[0, 1]$.

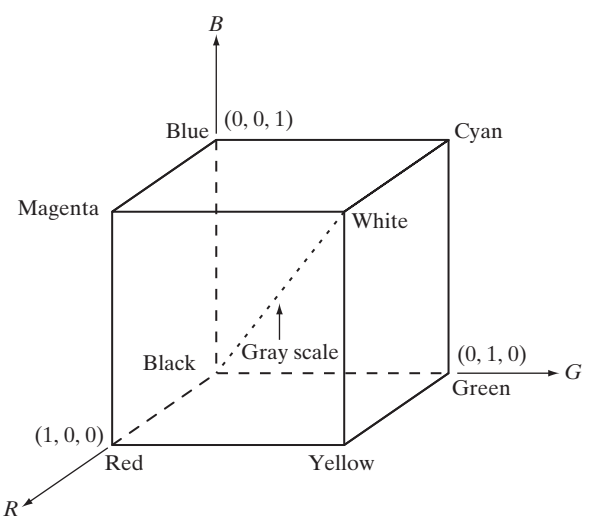


FIGURE 6.7 Schematic of the RGB color cube. Points along the main diagonal have gray values, from black at the origin to white at point (1, 1, 1).

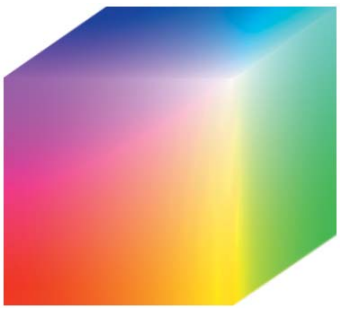


FIGURE 6.8 RGB 24-bit color cube.

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.

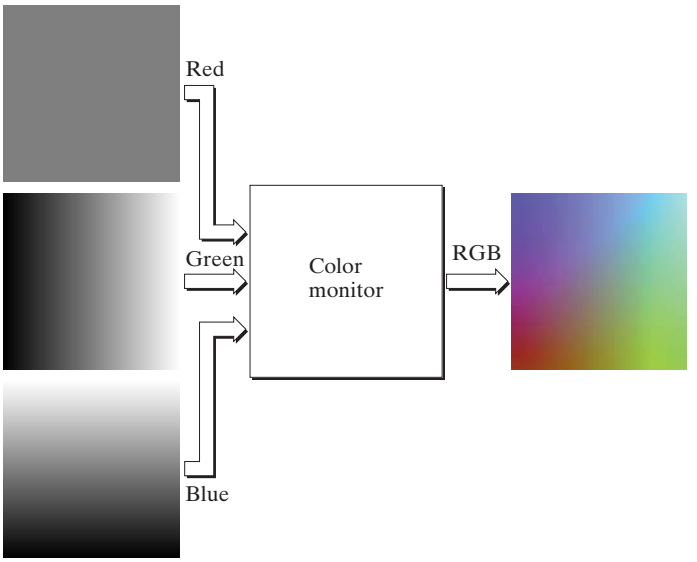
EXAMPLE 6.1: Generating the hidden face planes and a cross section of the RGB color cube.

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

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

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.

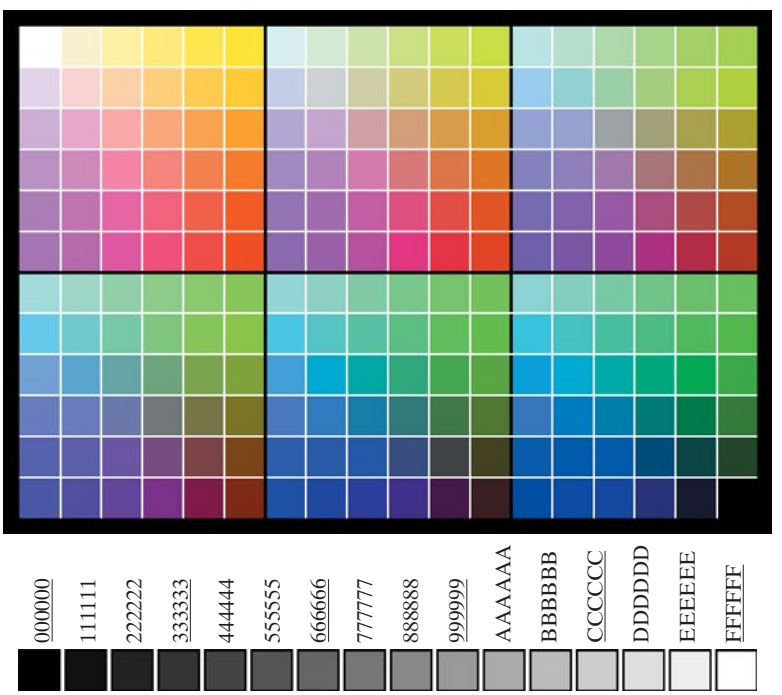
Number System	Color Equivalents					
Hex	00	33	66	99	CC	FF
Decimal	0	51	102	153	204	255

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 FFFFFF represent black and white, respectively. Keep in mind that the same result is obtained by using the more familiar decimal notation. For instance, the brightest red in decimal notation has $R = 255$ (FF) and $G = B = 0$.

Figure 6.10(a) shows the 216 safe colors, organized in descending RGB values. The square in the top left array has value FFFFFF (white), the second square to its right has value FFFFCC, the third square has value FFFF99, and



a
b

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

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

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

where, again, the assumption is that all color values have been normalized to the range [0, 1]. Equation (6.2-1) demonstrates that light reflected from a

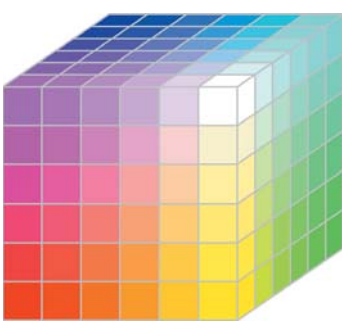


FIGURE 6.11 The RGB safe-color cube.

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.

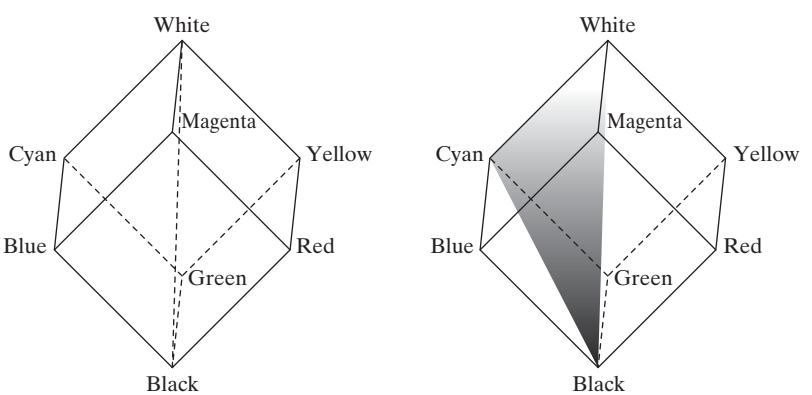
When humans view a color object, we describe it by its hue, saturation, and brightness. Recall from the discussion in Section 6.1 that hue is a color attribute that describes a pure color (pure yellow, orange, or red), whereas saturation gives a measure of the degree to which a pure color is diluted 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.

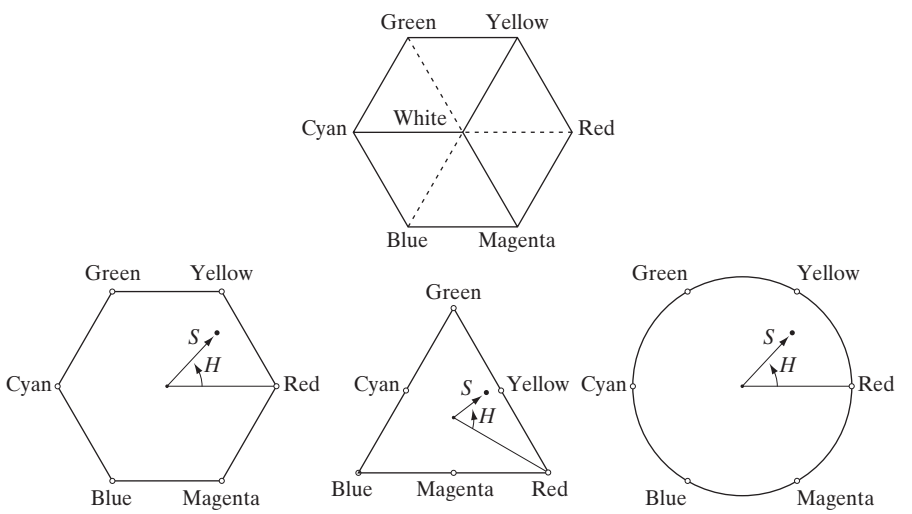
In order to see how hue can be determined also from a given RGB point, consider Fig. 6.12(b), which shows a plane defined by three points (black, white, and cyan). The fact that the black and white points are contained in the plane tells us that the intensity axis also is contained in the plane. Furthermore, 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 is 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.



a
b c d

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

a
b

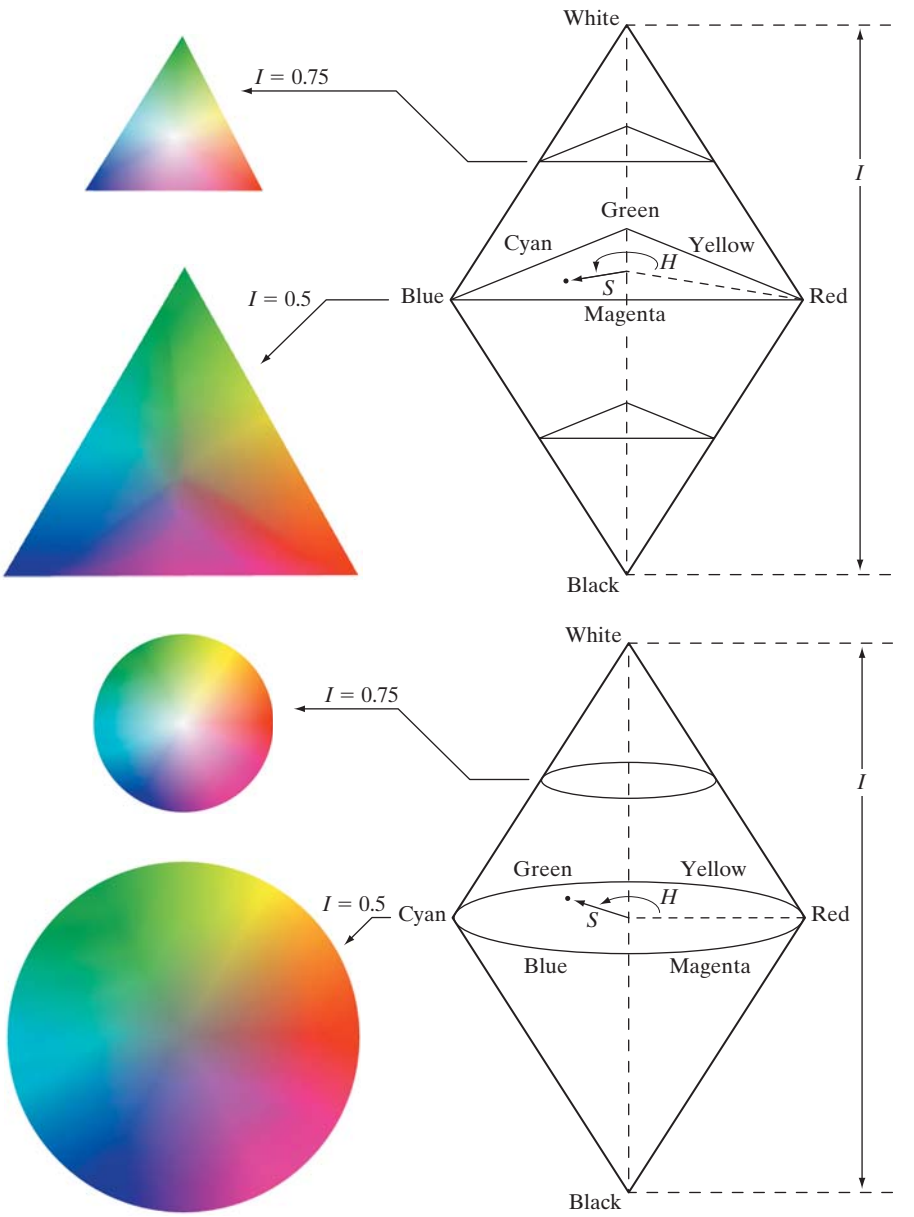


FIGURE 6.14 The HSI color model based on (a) triangular and (b) circular color planes. The triangles and circles are perpendicular to the vertical intensity axis.

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

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

with[†]

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

The saturation component is given by

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

Finally, the intensity component is given by

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

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

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

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

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

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

and

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

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

$$H = H - 120° \tag{6.2-8}$$

[†]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

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

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

and

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

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

$$H = H - 240^\circ \tag{6.2-12}$$

Then the RGB components are

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

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

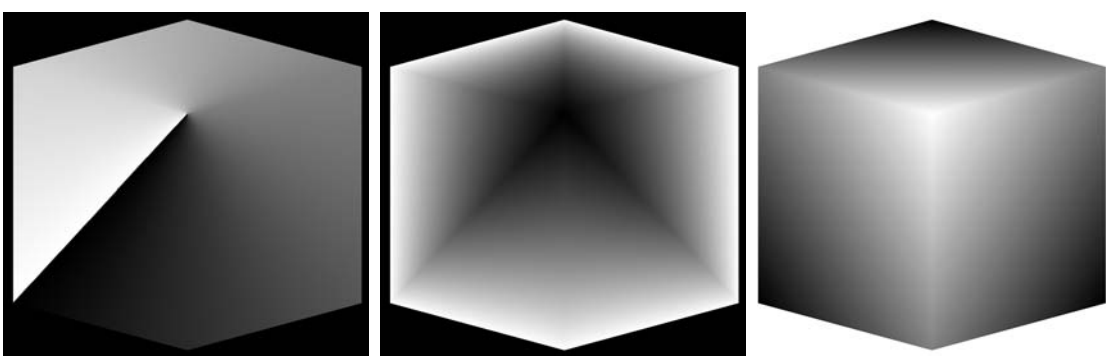
and

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

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

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

■ Figure 6.15 shows the hue, saturation, and intensity images for the RGB values shown in Fig. 6.8. Figure 6.15(a) is the hue image. Its most distinguishing feature is the discontinuity in value along a 45° line in the front (red) plane of the cube. To understand the reason for this discontinuity, refer to Fig. 6.8, draw a line from the red to the white vertices of the cube, and select a point in the middle of this line. Starting at that point, draw a path to the right, following the cube around until you return to the starting point. The major colors encountered in this path are yellow, green, cyan, blue, magenta, and back to red. According to Fig. 6.13, the values of hue along this path should increase from 0°



a b c

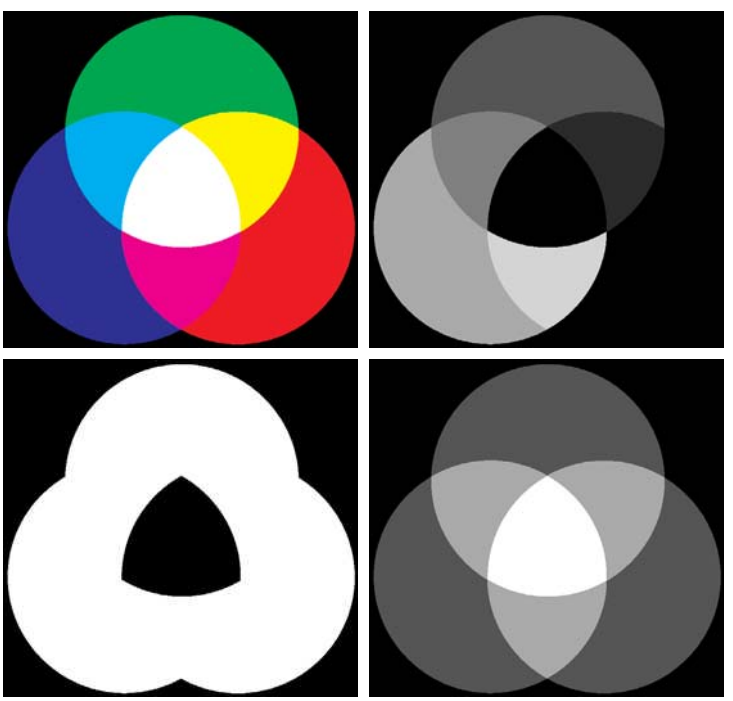
FIGURE 6.15 HSI components of the image in Fig. 6.8. (a) Hue, (b) saturation, and (c) intensity images.

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

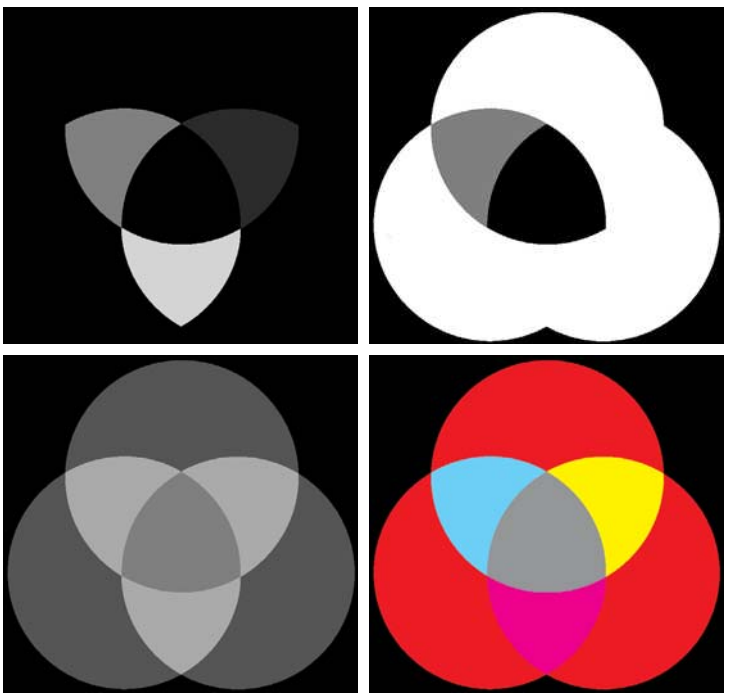


FIGURE 6.17
(a)–(c) Modified HSI component images.
(d) Resulting RGB image. (See Fig. 6.16 for the original HSI images.)

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 $l_1, l_2, \ldots, l_P$. Then, assuming that $0 < P < L - 1$, the P planes partition the gray scale into $P + 1$ intervals, $V_1, V_2, \ldots, V_{P+1}$. Intensity to color assignments are made according to the relation

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

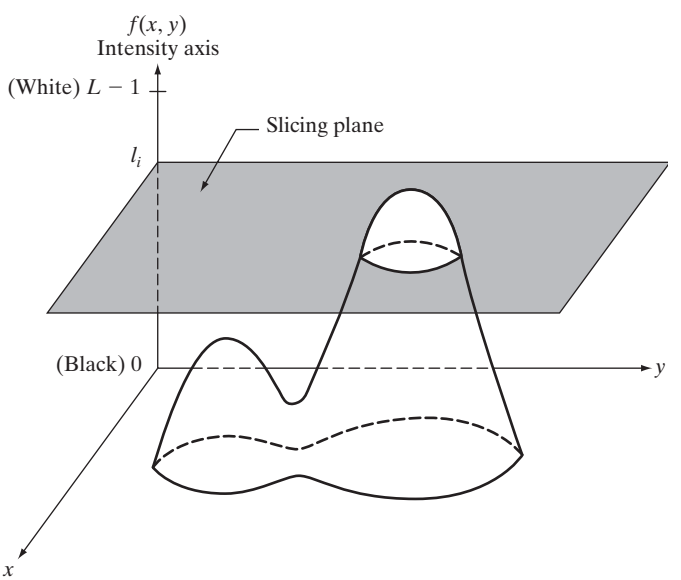


FIGURE 6.18 Geometric interpretation of the intensity-slicing technique.

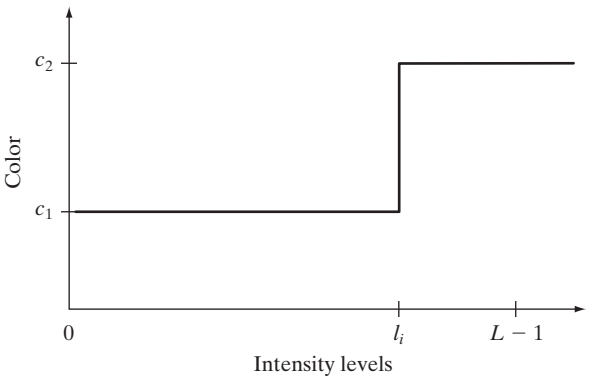


FIGURE 6.19 An alternative representation of the intensity-slicing technique.

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

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

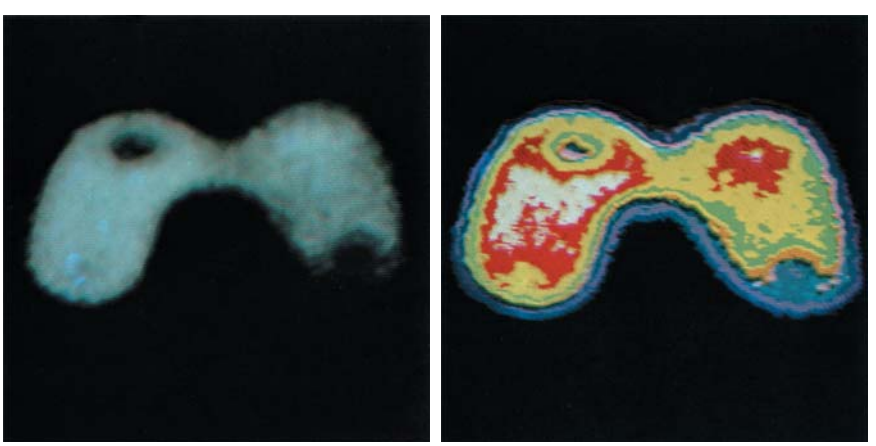
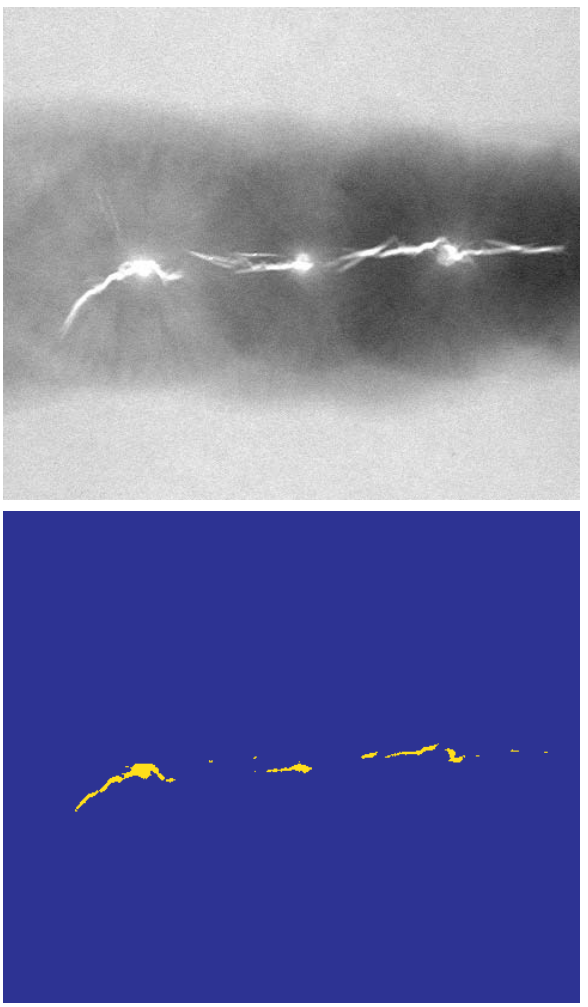


FIGURE 6.20 (a) Monochrome image of the Picker Thyroid Phantom. (b) Result of density slicing into eight colors. (Courtesy of Dr. J. L. Blankenship, Instrumentation and Controls Division, Oak Ridge National Laboratory.)

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.

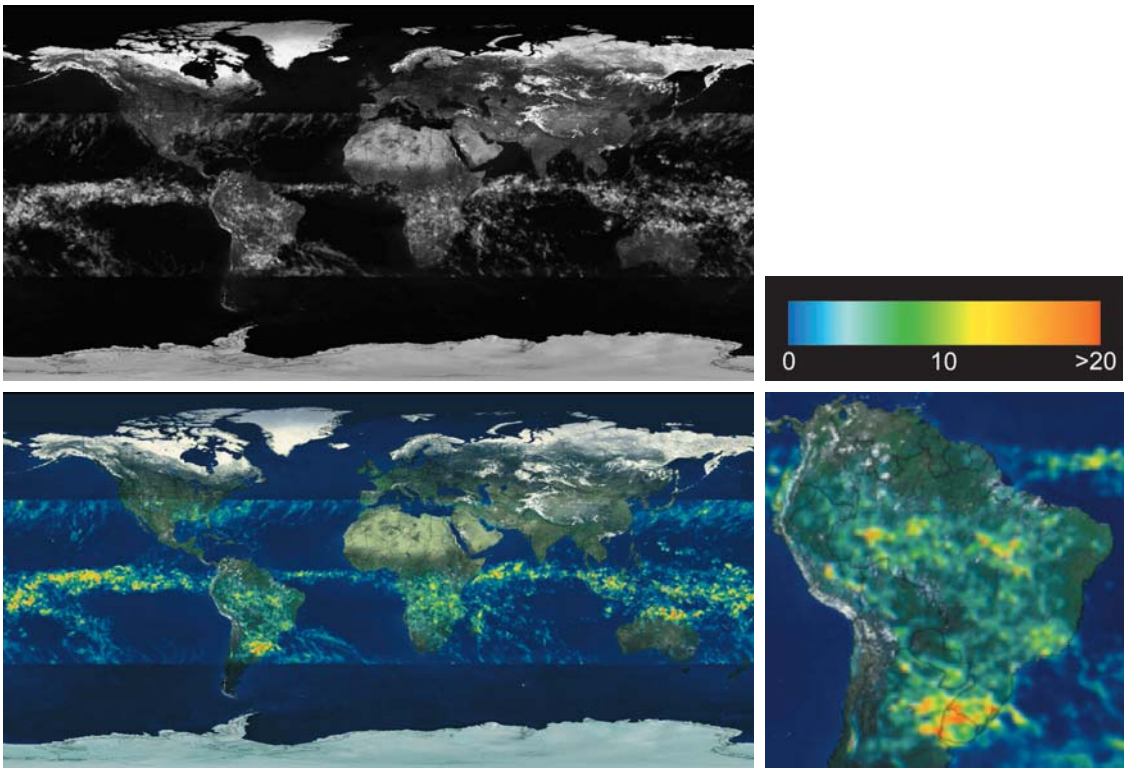
■ Measurement of rainfall levels, especially in the tropical regions of the Earth, is of interest in diverse applications dealing with the environment. Accurate measurements using ground-based sensors are difficult and expensive to acquire, and total rainfall figures are even more difficult to obtain 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



a b
c d

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

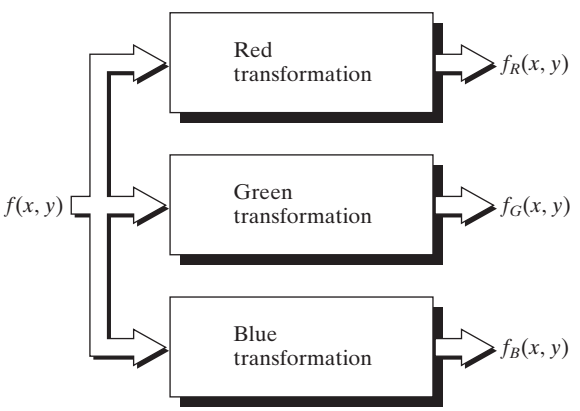


FIGURE 6.23 Functional block diagram for pseudocolor image processing. f_R, f_G, and f_B are fed into the corresponding red, green, and blue inputs of an RGB color monitor.

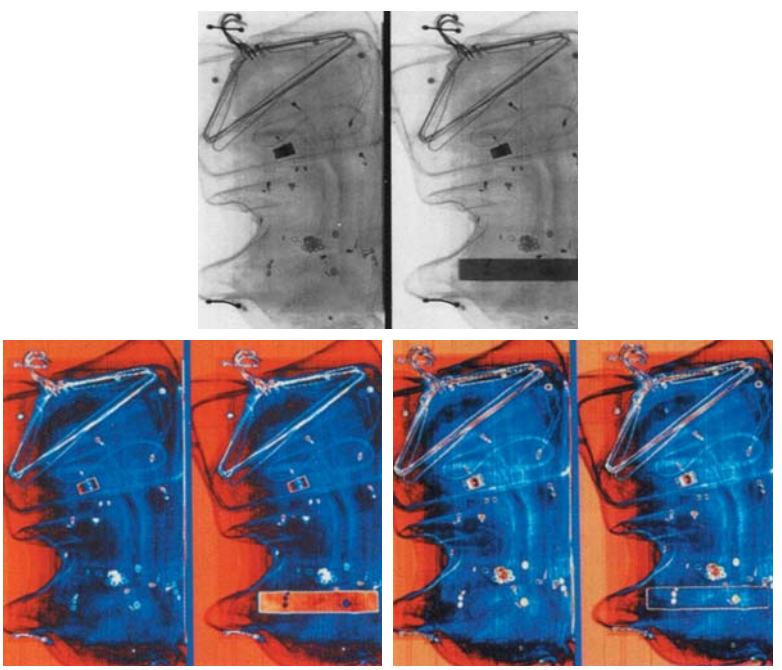
functions. Note that these are transformations on the intensity values of an image and are not functions of position.

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.

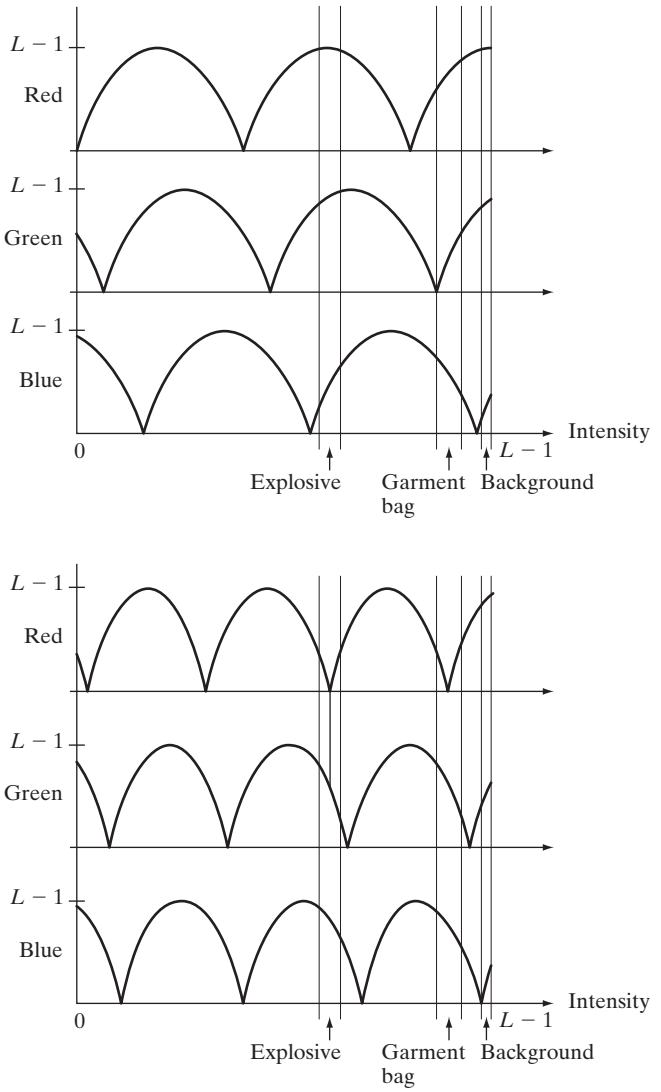
■ Figure 6.24(a) shows two monochrome images of luggage obtained from an airport X-ray scanning system. The image on the left contains ordinary articles. The image on the right contains the same articles, as well as a block of simulated plastic explosives. The purpose of this example is to illustrate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



a
b

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. ■

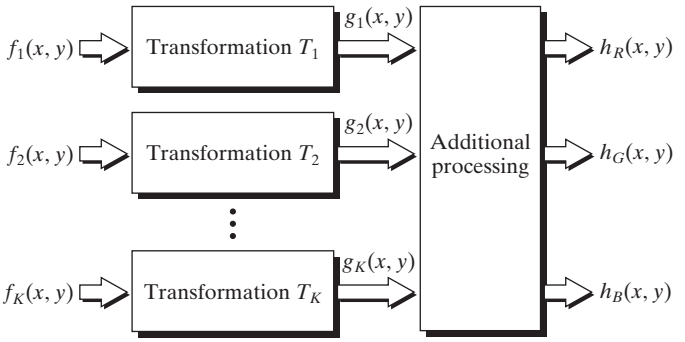


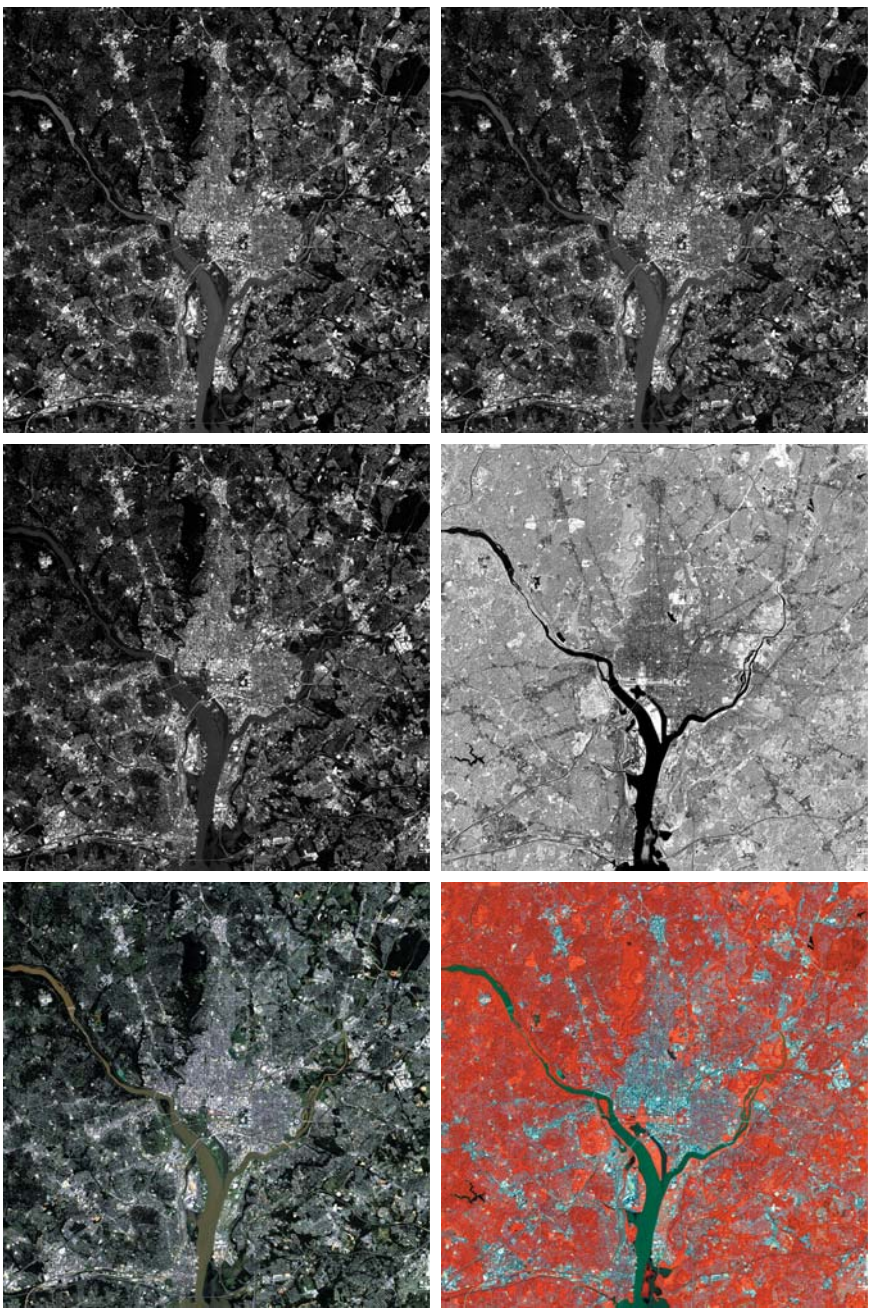
FIGURE 6.26 A pseudocolor coding approach used when several monochrome images are available.

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



a b
c d
e f

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

a
b

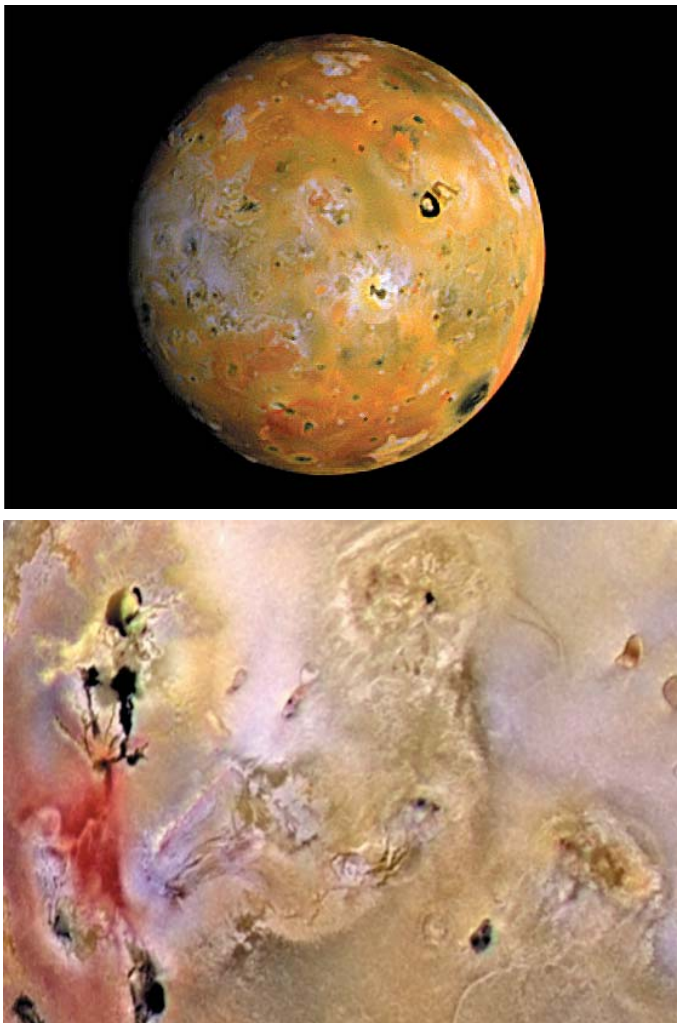


FIGURE 6.28
(a) Pseudocolor rendition of Jupiter Moon Io. (b) A close-up. (Courtesy of NASA.)

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

6.4 Basics of Full-Color Image Processing

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

three components, color pixels are vectors. For example, in the RGB system, each color point can be interpreted as a vector extending from the origin to that point in the RGB coordinate system (see Fig. 6.7).

Let **c** represent an arbitrary vector in RGB color space:

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

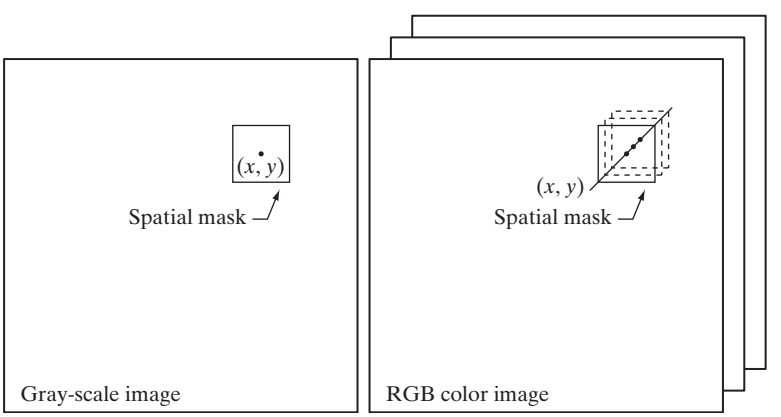
This equation indicates that the components of **c** are simply the RGB components of a color image at a point. We take into account the fact that the color components are a function of coordinates (x, y) by using the notation

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

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

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

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



a b

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

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

6.5 Color Transformations

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

6.5.1 Formulation

As with the intensity transformation techniques of Chapter 3, we model color transformations using the expression

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

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

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

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

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

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

The full-color image in Fig. 6.30 shows a high-resolution color image of a bowl of strawberries and a cup of coffee that was digitized from a large format ($4'' \times 5''$) color negative. The second row of the figure contains the components

of the initial CMYK scan. In these images, black represents 0 and white represents 1 in each CMYK color component. Thus, we see that the strawberries are composed of large amounts of magenta and yellow because the images corresponding to these two CMYK components are the brightest. Black is used sparingly and is generally confined to the coffee and shadows within the bowl of strawberries. When the CMYK image is converted to RGB, as shown in the third row of the figure, the strawberries are seen to contain a large amount of red and very little (although some) green and blue. The last row of Fig. 6.30 shows the HSI components of the full-color image—computed using Eqs. (6.2-2) through (6.2-4). As expected, the intensity component is a monochrome rendition of the full-color original. In addition, the strawberries are relatively pure in color; they possess the highest saturation or least dilution by white light of any of the hues in the image. Finally, we note some difficulty in interpreting the hue component. The problem is compounded by the fact that (1) there is a discontinuity in the HSI model where 0° and 360° meet (see Fig. 6.15), and (2) hue is undefined for a saturation of 0 (i.e., for white, black, and pure grays). The discontinuity of the model is most apparent around the strawberries, which are depicted in gray level values near both black (0) and white (1). The result is an unexpected mixture of highly contrasting gray levels to represent a single color—red.

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

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

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

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

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

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

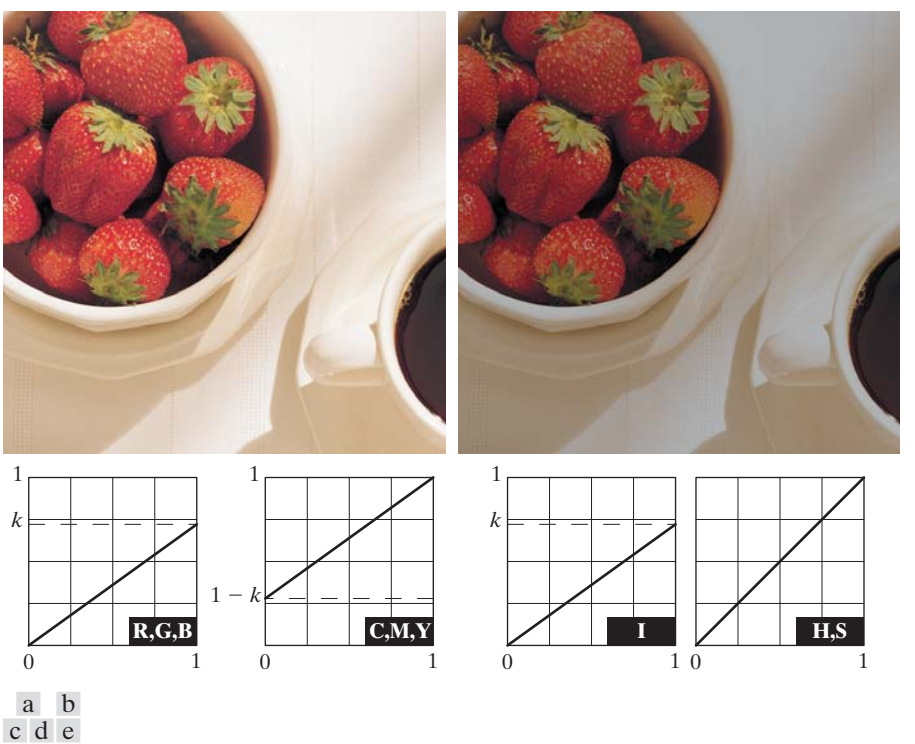
The CMY space requires a similar set of linear transformations:

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

Although the HSI transformation involves the fewest number of operations, the computations required to convert an RGB or CMY(K) image to the HSI space more than offsets (in this case) the advantages of the simpler transformation—the conversion calculations are more computationally intense than the intensity transformation itself. Regardless of the color space

selected, however, the output is the same. Figure 6.31(b) shows the result of applying any of the transformations in Eqs. (6.5-4) through (6.5-6) to the full-color image of Fig. 6.30 using $k = 0.7$. The mapping functions themselves are depicted graphically in Figs. 6.31(c) through (e).

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



a b
c d e

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

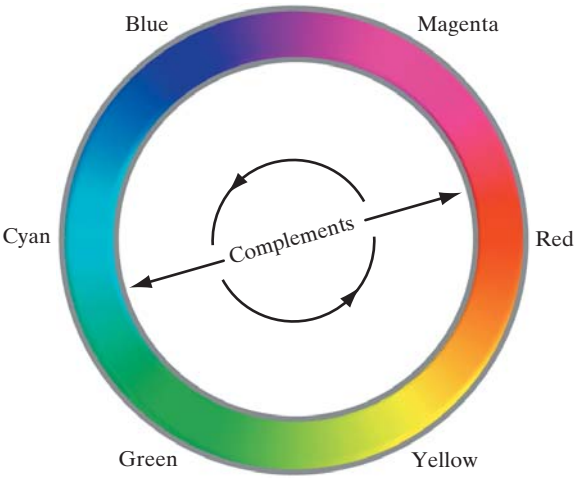


FIGURE 6.32 Complements on the color circle.

6.5.2 Color Complements

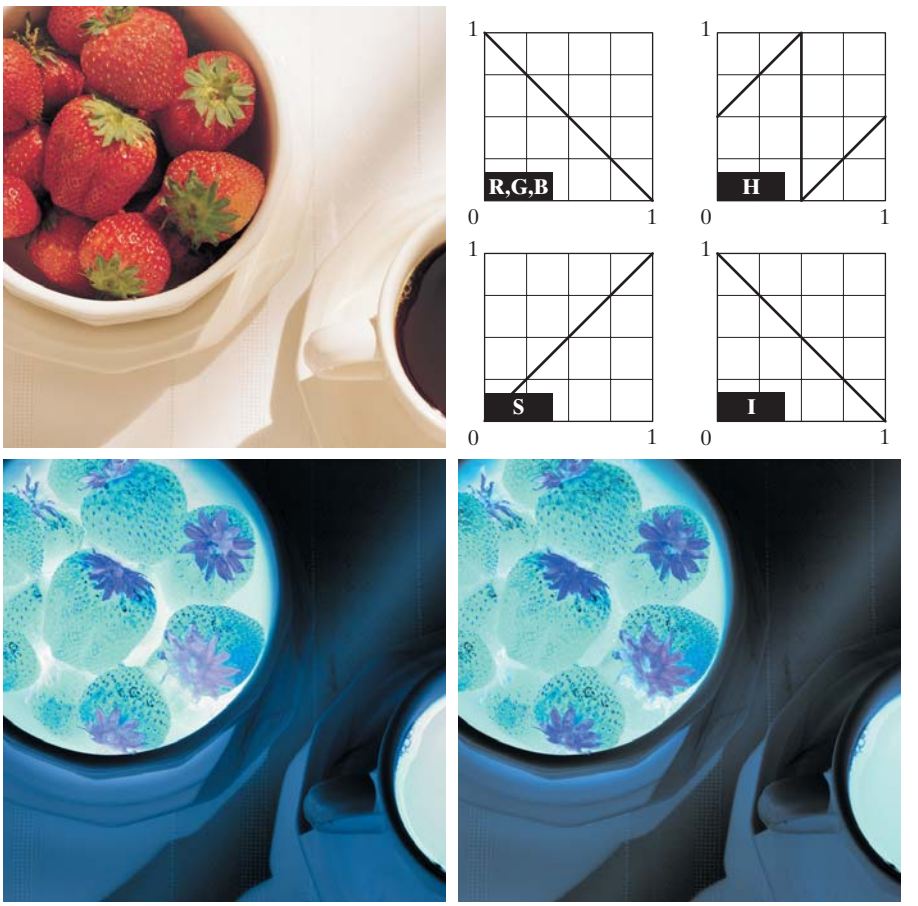
The hues directly opposite one another on the *color circle*[†] of Fig. 6.32 are called *complements*. Our interest in complements stems from the fact that they are analogous to the gray-scale negatives of Section 3.2.1. As in the gray-scale case, color complements are useful for enhancing detail that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

EXAMPLE 6.7: Computing color image complements.

■ Figures 6.33(a) and (c) show the full-color image from Fig. 6.30 and its color complement. The RGB transformations used to compute the complement are plotted in Fig. 6.33(b). They are identical to the gray-scale negative transformation defined in Section 3.2.1. Note that the computed complement is reminiscent of conventional photographic color film negatives. Reds of the original image are replaced by cyans in the complement. When the original image is black, the complement is white, and so on. Each of the hues in the complement image can be predicted from the original image using the color circle of Fig. 6.32, and each of the RGB component transforms involved in the computation of the complement is a function of *only* the corresponding input color component.

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

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



a b
c d

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

6.5.3 Color Slicing

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

One of the simplest ways to "slice" a color image is to map the colors outside some range of interest to a nonprominent neutral color. If the colors of interest are enclosed by a cube (or *hypercube* for $n > 3$) of width W and centered at a

prototypical (e.g., average) color with components $(a_1, a_2, \ldots, a_n)$, the necessary set of transformations is

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

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

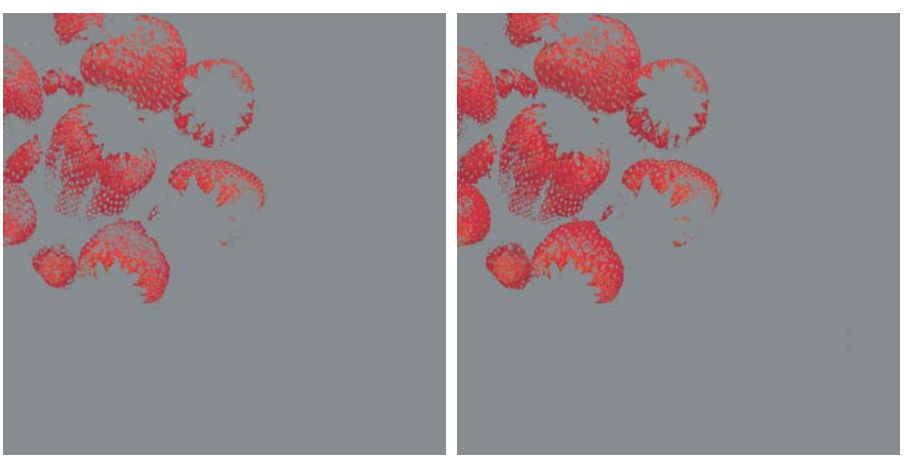
If a sphere is used to specify the colors of interest, Eq. (6.5-7) becomes

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

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

EXAMPLE 6.8: An illustration of color slicing.

■ Equations (6.5-7) and (6.5-8) can be used to separate the edible part of the strawberries in Fig. 6.31(a) from the background cups, bowl, coffee, and table. Figures 6.34(a) and (b) show the results of applying both transformations. In



a b

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

each case, a prototype red with RGB color coordinate (0.6863, 0.1608, 0.1922) was selected from the most prominent strawberry; W and R_0 were chosen so that the highlighted region would not expand to undesirable portions of the image. The actual values, $W = 0.2549$ and $R_0 = 0.1765$, were determined interactively. Note that the sphere-based transformation of Eq. (6.5-8) is slightly better, in the sense that it includes more of the strawberries' red areas. A sphere of radius 0.1765 does not completely enclose a cube of width 0.2549 but is itself not completely enclosed by the cube. ■

6.5.4 Tone and Color Corrections

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

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

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

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

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

where

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

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

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

EXAMPLE 6.9: Tonal transformations.

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

Figure 6.35 shows typical transformations used for correcting three common tonal imbalances—flat, light, and dark images. The S-shaped curve in the

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

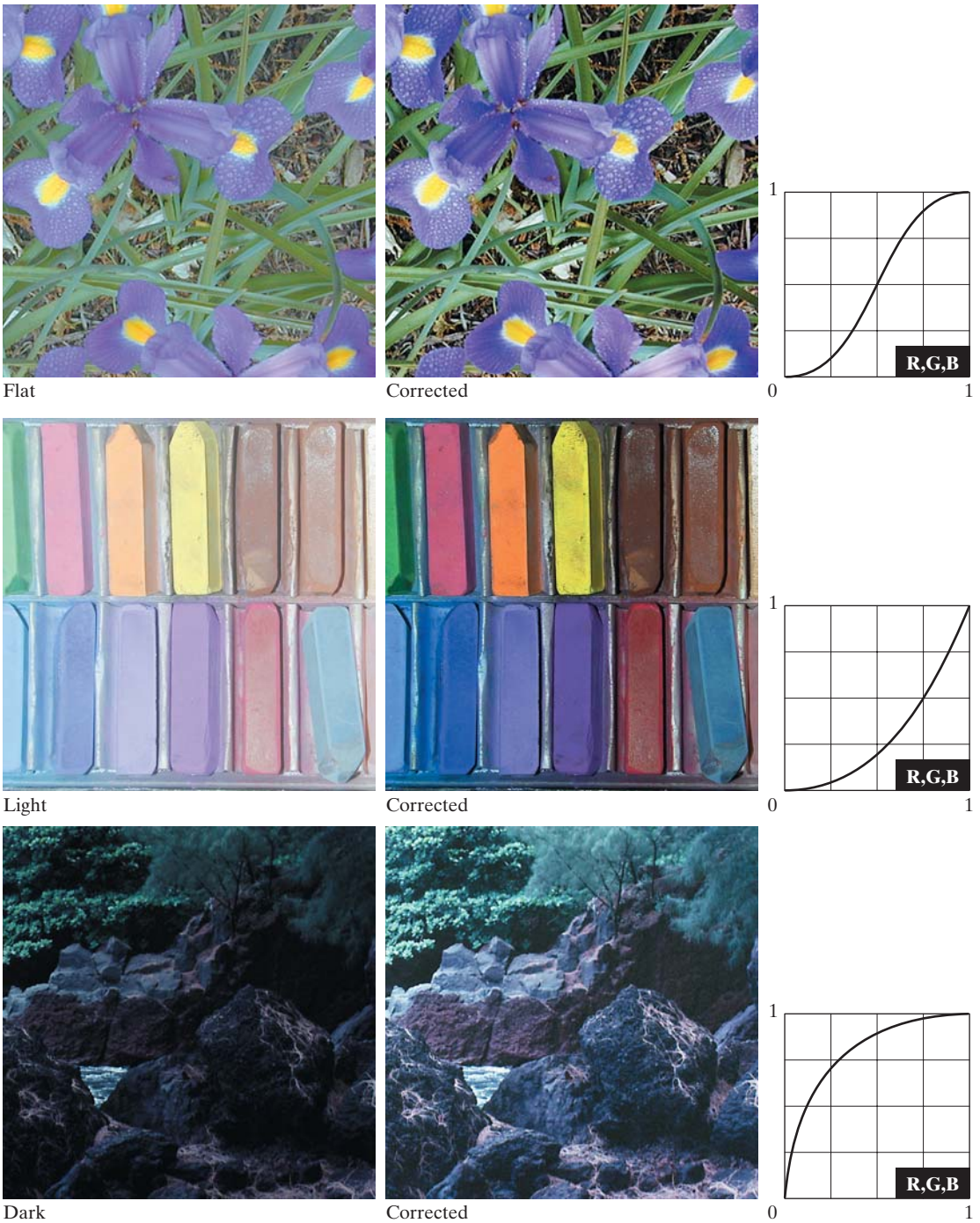


FIGURE 6.35 Tonal corrections for flat, light (high key), and dark (low key) color images. Adjusting the red, green, and blue components equally does not always alter the image hues significantly.

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

EXAMPLE 6.10: Color balancing.

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

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

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

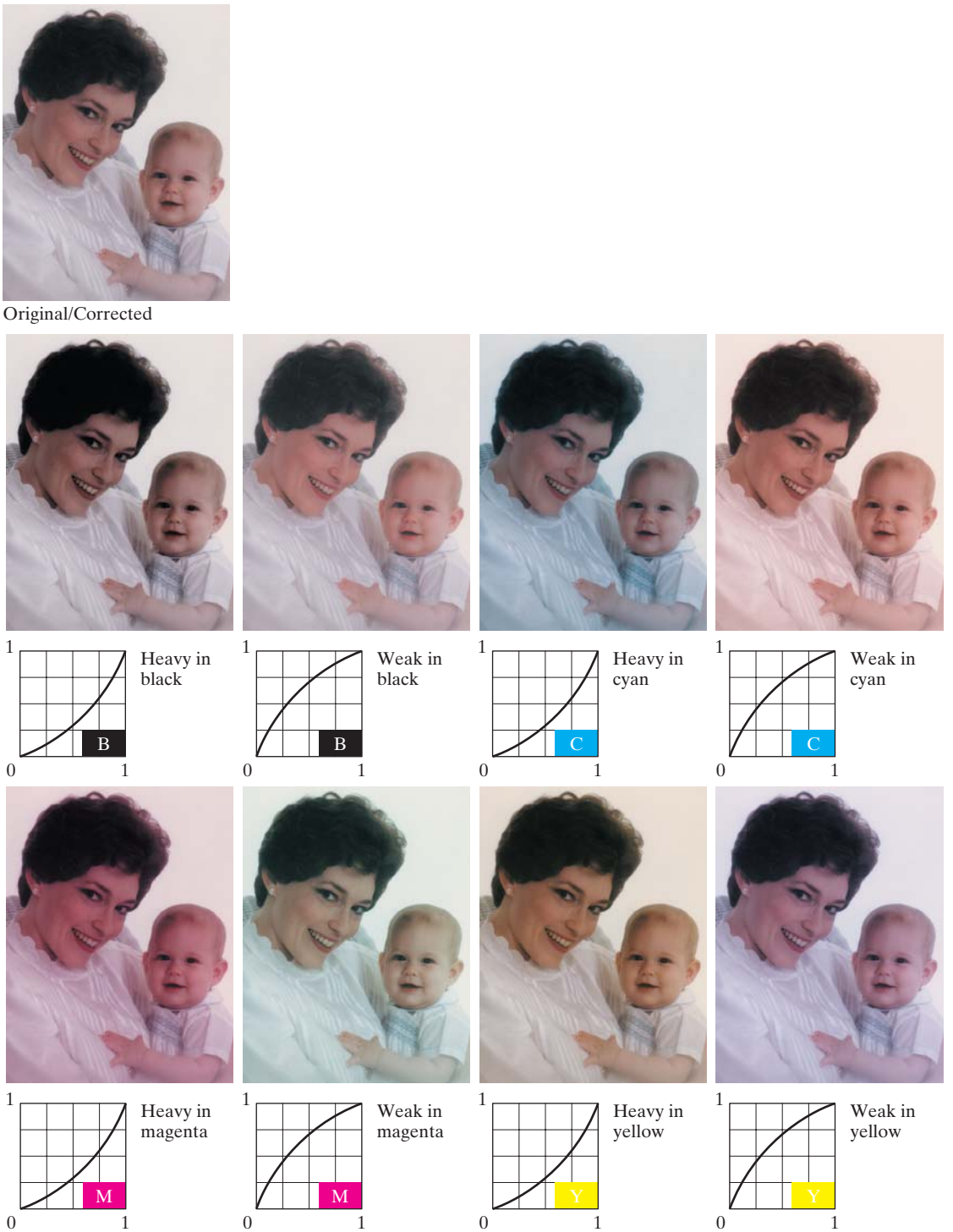
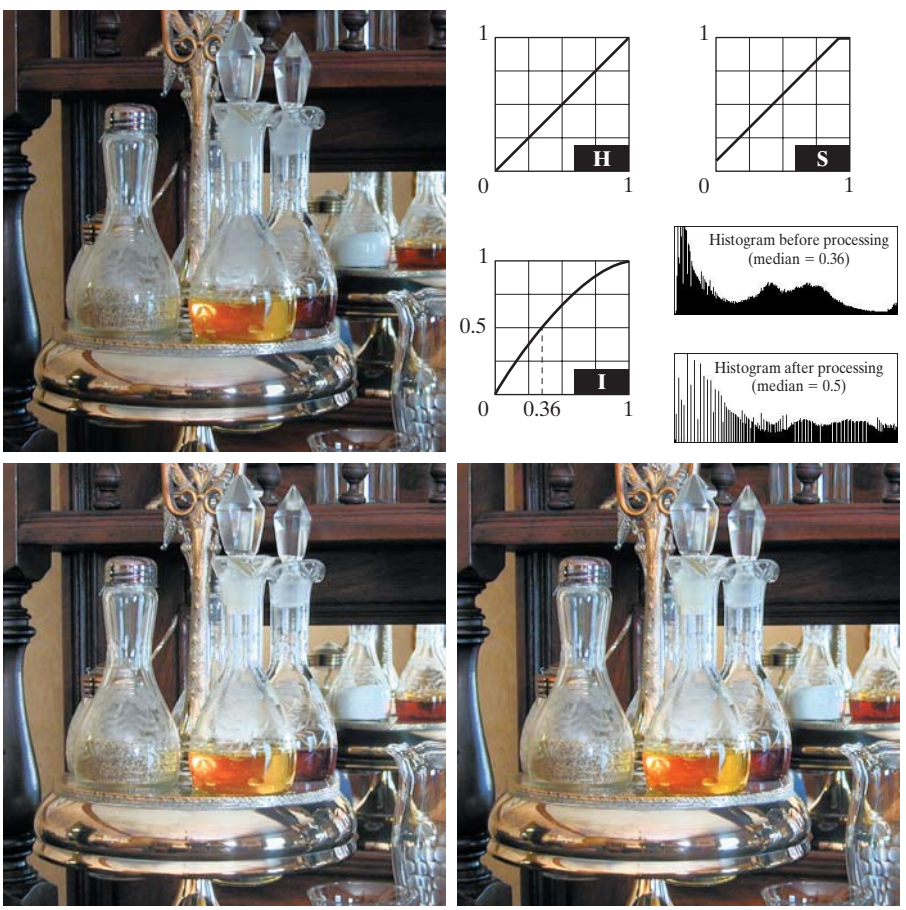


FIGURE 6.36 Color balancing corrections for CMYK color images.

6.5.5 Histogram Processing

Unlike the interactive enhancement approaches of the previous section, the gray-level histogram processing transformations of Section 3.3 can be applied to color images in an automated way. Recall that histogram equalization automatically determines a transformation that seeks to produce an image with a uniform histogram of intensity values. In the case of monochrome images, it was shown (see Fig. 3.20) to be reasonably successful at handling low-, high-, and middle-key images. Since color images are composed of multiple components, however, consideration must be given to adapting the gray-scale technique to more than one component and/or histogram. As might be expected, it is generally unwise to histogram equalize the components of a color image independently. This results in erroneous color. A more logical approach is to spread the color intensities uniformly, leaving the colors themselves (e.g., hues) unchanged. The following example shows that the HSI color space is ideally suited to this type of approach.



a b
c d

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

EXAMPLE 6.11: Histogram equalization in the HSI color space.

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

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

6.6 Smoothing and Sharpening

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

6.6.1 Color Image Smoothing

With reference to Fig. 6.29(a) and the discussion in Sections 3.4 and 3.5, gray-scale image smoothing can be viewed as a spatial filtering operation in which the coefficients of the filtering mask have the same value. As the mask is slid across the image to be smoothed, each pixel is replaced by the average of the pixels in the neighborhood defined by the mask. As can be seen in Fig. 6.29(b), this concept is easily extended to the processing of full-color images. The principal difference is that instead of scalar intensity values we must deal with component vectors of the form given in Eq. (6.4-2).

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

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

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

Consult the book Web site for a brief review of vectors and matrices.

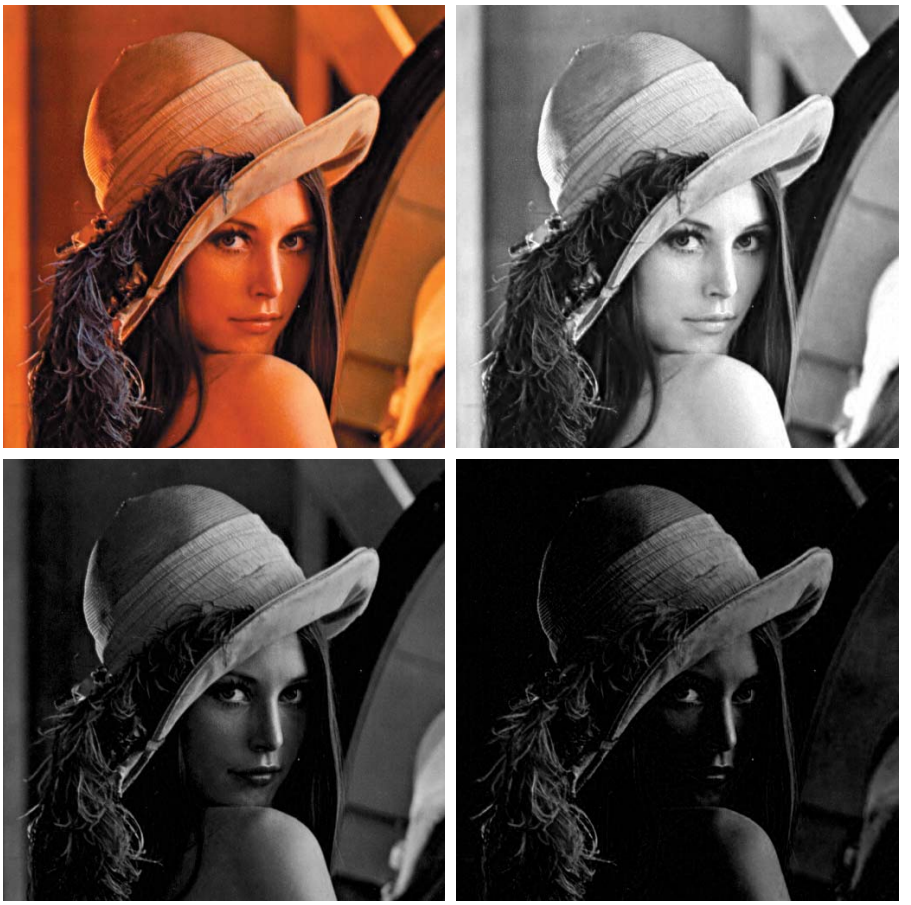
$$\bar{\mathbf{c}}(x, y) = \begin{bmatrix} \dfrac{1}{K} \displaystyle\sum_{(s,t)\in S_{xy}} R(s, t) \\ \dfrac{1}{K} \displaystyle\sum_{(s,t)\in S_{xy}} G(s, t) \\ \dfrac{1}{K} \displaystyle\sum_{(s,t)\in S_{xy}} B(s, t) \end{bmatrix} \tag{6.6-2}$$

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

EXAMPLE 6.12: Color image smoothing by neighborhood averaging.

■ Consider the RGB color image in Fig. 6.38(a). Its red, green, and blue component images are shown in Figs. 6.38(b) through (d). Figures 6.39(a) through (c) show the HSI components of the image. Based on the discussion in the previous paragraph, we smoothed each component image of the RGB image in Fig. 6.38 independently using a 5×5 spatial averaging mask. We then combined the individually smoothed images to form the smoothed, full-color RGB result shown in Fig. 6.40(a). Note that this image appears as we would expect from performing a spatial smoothing operation, as in the examples given in Section 3.5.

In Section 6.2, we noted that an important advantage of the HSI color model is that it decouples intensity and color information. This makes it suitable for many gray-scale processing techniques and suggests that it might be more efficient to smooth only the intensity component of the HSI representation in Fig. 6.39. To illustrate the merits and/or consequences of this approach, we next smooth only the intensity component (leaving the hue and saturation components unmodified) and convert the processed result to an RGB image for display. The smoothed color image is shown in Fig. 6.40(b). Note that it is similar to Fig. 6.40(a), but, as you can see from the difference image in Fig. 6.40(c), the two smoothed images are not identical. This is because in Fig. 6.40(a) the color of each pixel is the average color of the pixels in the neighborhood. On the other hand, by smoothing only the intensity component image in Fig. 6.40(b), the hue and saturation of each pixel was not affected and, therefore, the pixel colors did not change. It follows from this observation that the difference between the two smoothing approaches would become more pronounced as a function of increasing filter size. ■

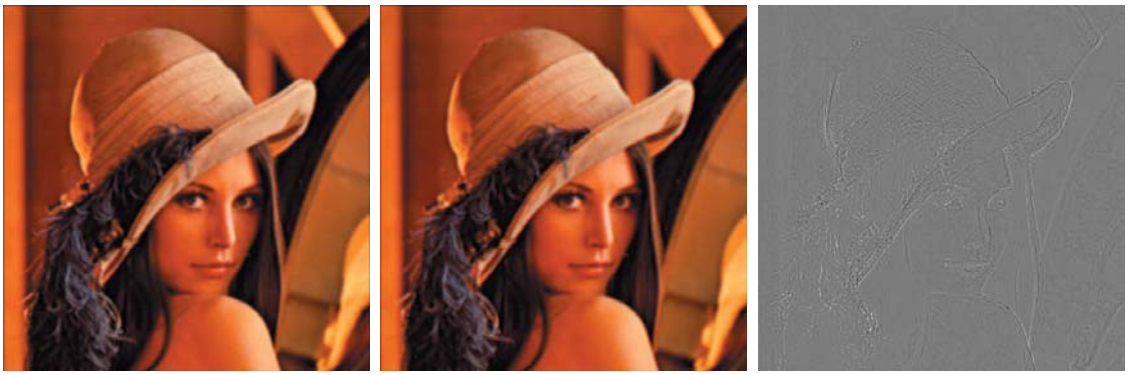


a b
c d

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

a b c

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



a b c

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

6.6.2 Color Image Sharpening

In this section we consider image sharpening using the Laplacian (see Section 3.6.2). From vector analysis, we know that the Laplacian of a vector is defined as a vector whose components are equal to the Laplacian of the individual scalar components of the input vector. In the RGB color system, the Laplacian of vector $\mathbf{c}$ in Eq. (6.4-2) is

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

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

a b c

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

EXAMPLE 6.13: Sharpening with the Laplacian.

■ Figure 6.41(a) was obtained using Eq. (3.6-7) and the mask in Fig. 3.37(c) to compute the Laplacians of the RGB component images in Fig. 6.38. These results were combined to produce the sharpened full-color result. Figure 6.41(b) shows a similarly sharpened image based on the HSI components in Fig. 6.39. This result was generated by combining the Laplacian of the intensity component with the unchanged hue and saturation components. The difference between the RGB and HSI sharpened images is shown in Fig. 6.41(c). The reason for the discrepancies between the two images is as in Example 6.12. ■

6.7 Image Segmentation Based on Color

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

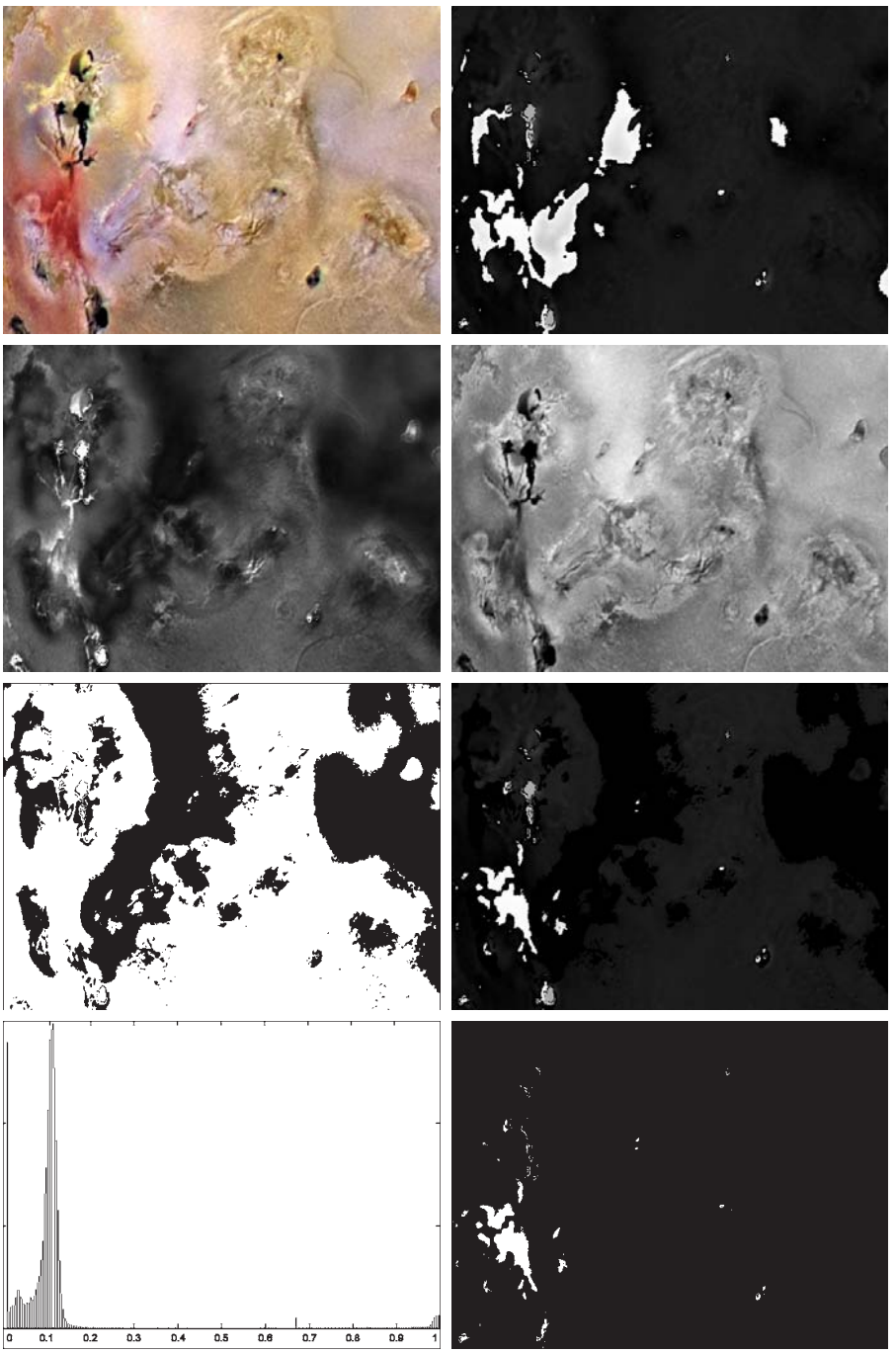
6.7.1 Segmentation in HSI Color Space

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

EXAMPLE 6.14: Segmentation in HSI space.

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

Figure 6.42(f) is the product of the mask with the hue image, and Fig. 6.42(g) is the histogram of the product image (note that the gray scale is in the range $[0, 1]$). We see in the histogram that high values (which are the values of interest) are grouped at the very high end of the gray scale, near 1.0. The result of thresholding the product image with threshold value of 0.9 resulted in the binary image shown in Fig. 6.42(h). The spatial location of the white points in this image identifies the points in the original image that have the reddish hue of interest. This was far from a perfect segmentation because there are points in the original image that we certainly would say have a reddish hue, but that were not identified by this segmentation method. However, it can be determined



a b
c d
e f
g h

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

by experimentation that the regions shown in white in Fig. 6.42(h) are about the best this method can do in identifying the reddish components of the original image. The segmentation method discussed in the following section is capable of yielding considerably better results. ■

6.7.2 Segmentation in RGB Vector Space

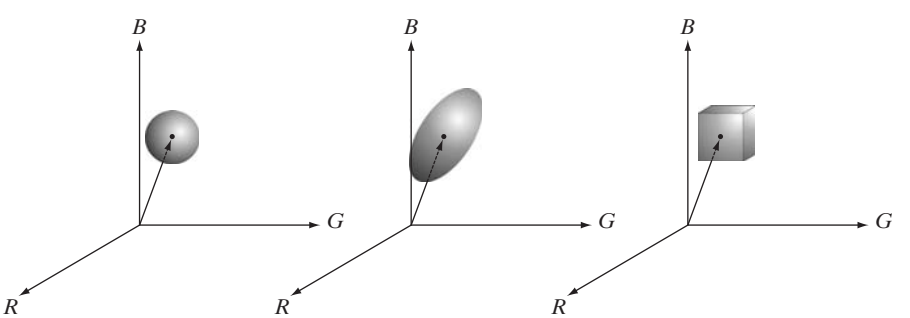
Although, as mentioned numerous times in this chapter, working in HSI space is more intuitive, segmentation is one area in which better results generally are obtained by using RGB color vectors. The approach is straightforward. Suppose that the objective is to segment objects of a specified color range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

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

where the subscripts R, G, and B denote the RGB components of vectors $\mathbf{a}$ and $\mathbf{z}$. The locus of points such that $D(\mathbf{z}, \mathbf{a}) \leq D_0$ is a solid sphere of radius D_0, as illustrated in Fig. 6.43(a). Points contained within the sphere satisfy the specified color criterion; points outside the sphere do not. Coding these two sets of points in the image with, say, black and white, produces a binary segmented image.

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

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



a b c

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

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

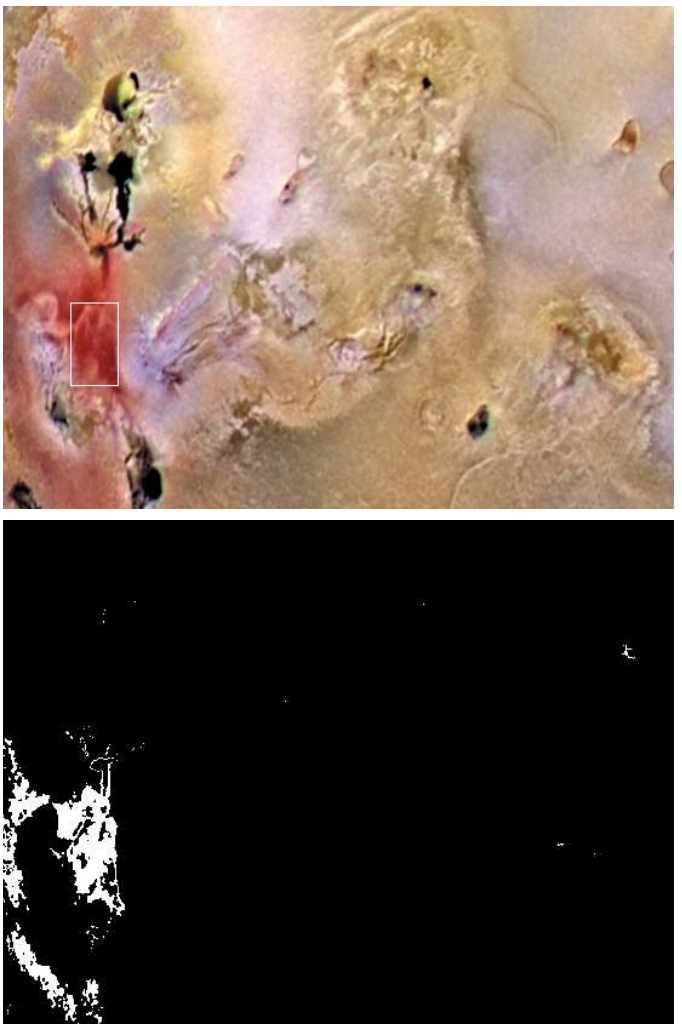
Because distances are positive and monotonic, we can work with the distance squared instead, thus avoiding square root computations. However, implementing Eq. (6.7-1) or (6.7-2) is computationally expensive for images of practical size, even if the square roots are not computed. A compromise is to use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

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

EXAMPLE 6.15: Color image segmentation in RGB space.

■ The rectangular region shown Fig. 6.44(a) contains samples of reddish colors we wish to segment out of the color image. This is the same problem we considered in Example 6.14 using hue, but here we approach the problem using RGB color vectors. The approach followed was to compute the mean vector $\mathbf{a}$ using the color points contained within the rectangle in Fig. 6.44(a), and then to compute the standard deviation of the red, green, and blue values of those samples. A box was centered at $\mathbf{a}$, and its dimensions along each of the RGB axes were selected as 1.25 times the standard deviation of the data along the corresponding axis. For example, let σ_R denote the standard deviation of the red components of the sample points. Then the dimensions of the box along the R-axis extended from $(a_R - 1.25\sigma_R)$ to $(a_R + 1.25\sigma_R)$, where a_R denotes the red component of average vector $\mathbf{a}$. The result of coding each point in the entire color image as white if it was on the surface or inside the box, and as black otherwise, is shown in Fig. 6.44(b). Note how the segmented region was generalized from the color samples enclosed by the rectangle. In fact, by comparing Figs. 6.44(b) and. 6.42(h), we see that segmentation in the RGB vector space yielded results that are much more accurate, in the sense that they correspond much more closely with what we would define as "reddish" points in the original color image. ■

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



a
b

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

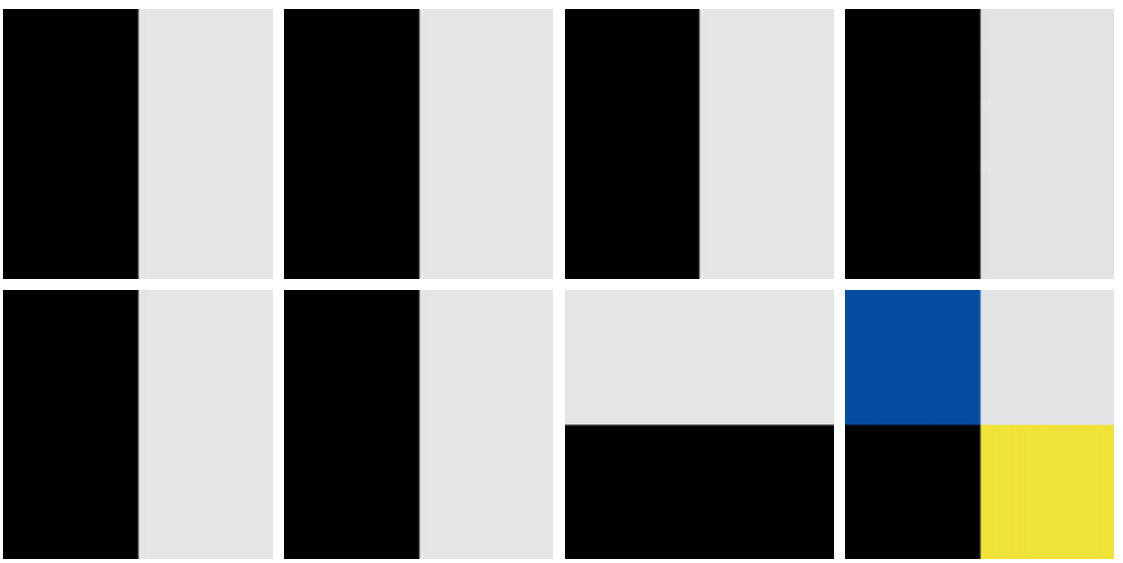
6.7.3 Color Edge Detection

As discussed in Chapter 10, edge detection is an important tool for image segmentation. In this section, we are interested in the issue of computing edges on an individual-image basis versus computing edges directly in color vector space. The details of edge-based segmentation are given in Section 10.2.

Edge detection by gradient operators was introduced in Section 3.6.4 in connection with image sharpening. Unfortunately, the gradient discussed in Section 3.6.4 is not defined for vector quantities. Thus, we know immediately that computing the gradient on individual images and then using the results to form a color image will lead to erroneous results. A simple example will help illustrate the reason why.

Consider the two $M \times M$ color images (M odd) in Figs. 6.45(d) and (h), composed of the three component images in Figs. 6.45(a) through (c) and (e) through (g), respectively. If, for example, we compute the gradient image of each of the component images [see Eq. (3.6-11)] and add the results to form the two corresponding RGB gradient images, the value of the gradient at point $[(M + 1)/2, (M + 1)/2]$ would be the same in both cases. Intuitively, we would expect the gradient at that point to be stronger for the image in Fig. 6.45(d) because the edges of the R, G, and B images are in the same direction in that image, as opposed to the image in Fig. 6.45(h), in which only two of the edges are in the same direction. Thus we see from this simple example that processing the three individual planes to form a composite gradient image can yield erroneous results. If the problem is one of just detecting edges, then the individual-component approach usually yields acceptable results. If accuracy is an issue, however, then obviously we need a new definition of the gradient applicable to vector quantities. We discuss next a method proposed by Di Zenzo [1986] for doing this.

The problem at hand is to define the gradient (magnitude and direction) of the vector $\mathbf{c}$ in Eq. (6.4-2) at any point (x, y). As was just mentioned, the gradient we studied in Section 3.6.4 is applicable to a *scalar* function $f(x, y)$; it is not applicable to vector functions. The following is one of the various ways in which we can extend the concept of a gradient to vector functions. Recall that for a scalar function $f(x, y)$, the gradient is a vector pointing in the direction of maximum rate of change of f at coordinates (x, y).



a b c d
e f g h

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

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

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

and

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

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

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

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

and

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

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

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

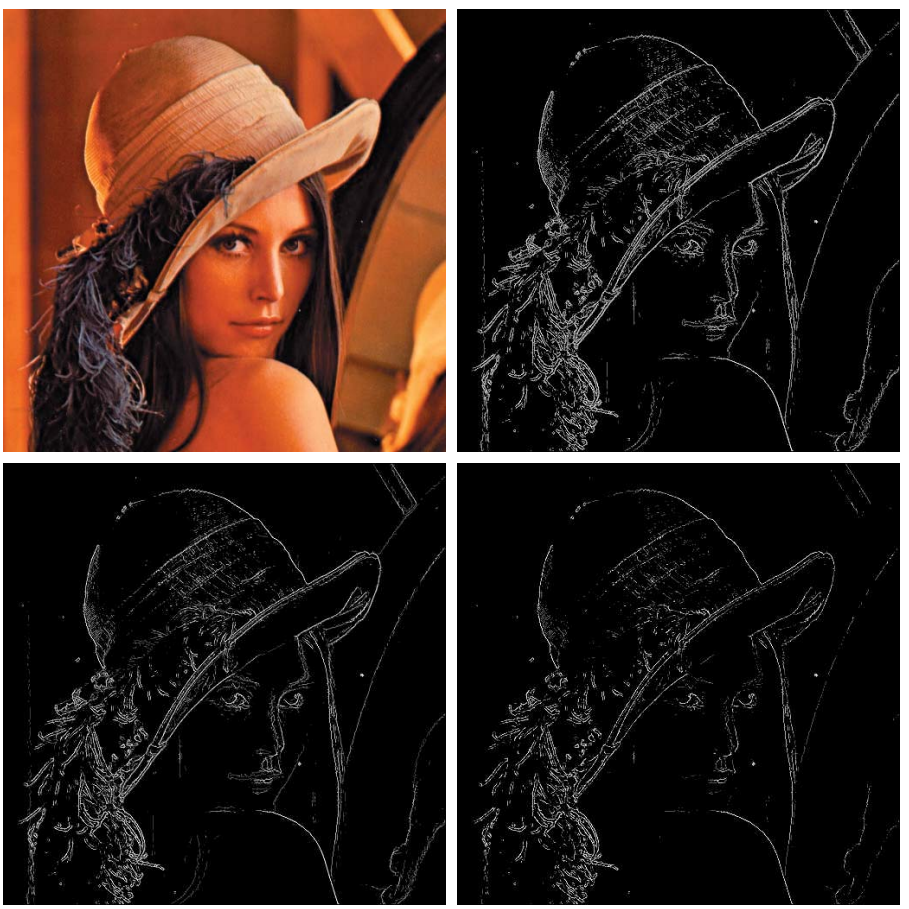
and that the value of the rate of change at (x, y), in the direction of $\theta(x, y)$, is given by

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

Because $\tan(\alpha) = \tan(\alpha \pm \pi)$, if θ_0 is a solution to Eq. (6.7-8), so is $\theta_0 \pm \pi/2$. Furthermore, $F_\theta = F_{\theta+\pi}$, so F has to be computed only for values of θ in the half-open interval $[0, \pi)$. The fact that Eq. (6.7-8) provides two values 90° apart means that this equation associates with each point (x, y) a pair of orthogonal directions. Along one of those directions F is maximum, and it is minimum along the other. The derivation of these results is rather lengthy, and we would gain little in terms of the fundamental objective of our current discussion by detailing it here. Consult the paper by Di Zenzo [1986] for details. The partial derivatives required for implementing Eqs. (6.7-5) through (6.7-7) can be computed using, for example, the Sobel operators discussed in Section 3.6.4.

EXAMPLE 6.16: Edge detection in vector space.

■ Figure 6.46(b) is the gradient of the image in Fig. 6.46(a), obtained using the vector method just discussed. Figure 6.46(c) shows the image obtained by computing the gradient of each RGB component image and forming a composite gradient image by adding the corresponding values of the three component images at each coordinate (x, y). The edge detail of the vector gradient image is more complete than the detail in the individual-plane gradient image in Fig. 6.46(c); for example, see the detail around the subject's right eye. The image in Fig. 6.46(d) shows the difference between the two gradient images at each point (x, y). It is important to note that both approaches yielded reasonable results. Whether the extra detail in Fig. 6.46(b) is worth the added computational burden (as opposed to implementation of the Sobel operators, which were used to generate the gradient of the individual planes) can only be determined by the requirements of a given problem. Figure 6.47 shows the three component gradient images, which, when added and scaled, were used to obtain Fig. 6.46(c). ■



a b
c d

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

a b c

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

6.8 Noise in Color Images

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

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

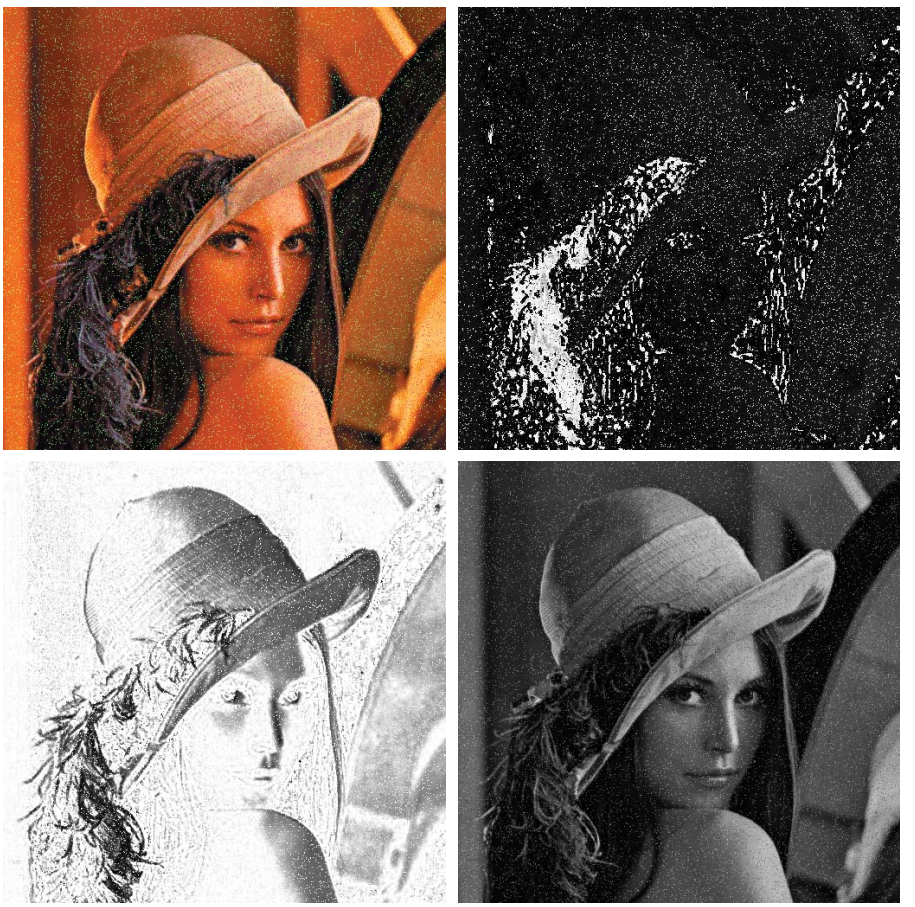
■ In this example we take a brief look at noise in color images and how noise carries over when converting from one color model to another. Figures 6.48(a) through (c) show the three color planes of an RGB image corrupted by Gaussian noise, and Fig. 6.48(d) is the composite RGB image. Note that fine grain noise such as this tends to be less visually noticeable in a color image than it is in a monochrome image. Figures 6.49(a) through (c) show the result of converting the RGB image in Fig. 6.48(d) to HSI. Compare these results with the HSI components of the original image (Fig. 6.39) and note how significantly degraded the hue and saturation components of the noisy image are. This is due to the nonlinearity of the cos and min operations in Eqs. (6.2-2) and (6.2-3), respectively. On the other hand, the intensity component in Fig. 6.49(c) is slightly smoother than any of the three noisy RGB component images. This is due to the fact that the intensity image is the average of the RGB images, as indicated in Eq. (6.2-4). (Recall the discussion in Section 2.6.3 regarding the fact that image averaging reduces random noise.)

a b
c d

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

a b c

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



a b
c d

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

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

As is true of the processes we have discussed thus far, filtering of full-color images can be carried out on a per-image basis or directly in color vector

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

6.9 Color Image Compression

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

EXAMPLE 6.18: A color image compression example.

■ Figure 6.51(a) shows a 24-bit RGB full-color image of an iris in which 8 bits each are used to represent the red, green, and blue components. Figure 6.51(b) was reconstructed from a compressed version of the image in (a) and is, in fact, a compressed and subsequently decompressed approximation of it. Although the compressed image is not directly displayable—it must be decompressed before input to a color monitor—the compressed image contains only 1 data bit (and thus 1 storage bit) for every 230 bits of data in the original image. Assuming that the compressed image could be transmitted over, say, the Internet, in 1 minute, transmission of the original image would require almost 4 hours. Of course, the transmitted data would have to be decompressed for viewing, but the decompression can be done in a matter of seconds. The JPEG 2000 compression algorithm used to generate Fig. 6.51(b) is a recently introduced standard that is described in detail in Section 8.2.10. Note that the reconstructed approximation image is slightly blurred. This is a characteristic of many *lossy* compression techniques; it can be reduced or eliminated by altering the level of compression. ■

a
b

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

Summary

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

The material on color vector spaces is a departure from methods that we had studied before and highlights some important differences between gray-scale and full-color processing. In terms of techniques, the areas of direct color vector processing are numerous and include processes such as median and other order filters, adaptive and