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. Most of the segmentation algorithms in this chapter are based on one of two basic properties of image intensity values: *discontinuity* and *similarity*. In the first category, the approach is to partition an image into regions based on abrupt changes in intensity, such as edges. 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. We 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 using clustering and superpixels, and give an introduction to graph cuts, an approach ideally suited for extracting the principal regions of an image. This is followed by a discussion of image segmentation based on morphology, an approach that combines several of the 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.

Upon completion of this chapter, readers should:

- Understand the characteristics of various types of edges found in practice.
- Understand how to use spatial filtering for edge detection.
- Be familiar with other types of edge detection methods that go beyond spatial filtering.
- Understand image thresholding using several different approaches.
- Know how to combine thresholding and spatial filtering to improve segmentation.
- Be familiar with region-based segmentation, including clustering and superpixels.
- Understand how graph cuts and morphological watersheds are used for segmentation.
- Be familiar with basic techniques for utilizing motion in image segmentation.

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

(b) R_i is a connected set, for $i = 0, 1, 2, \ldots, n$.

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

(d) $Q(R_i) = \text{TRUE}$ for $i = 0, 1, 2, \ldots, n$.

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

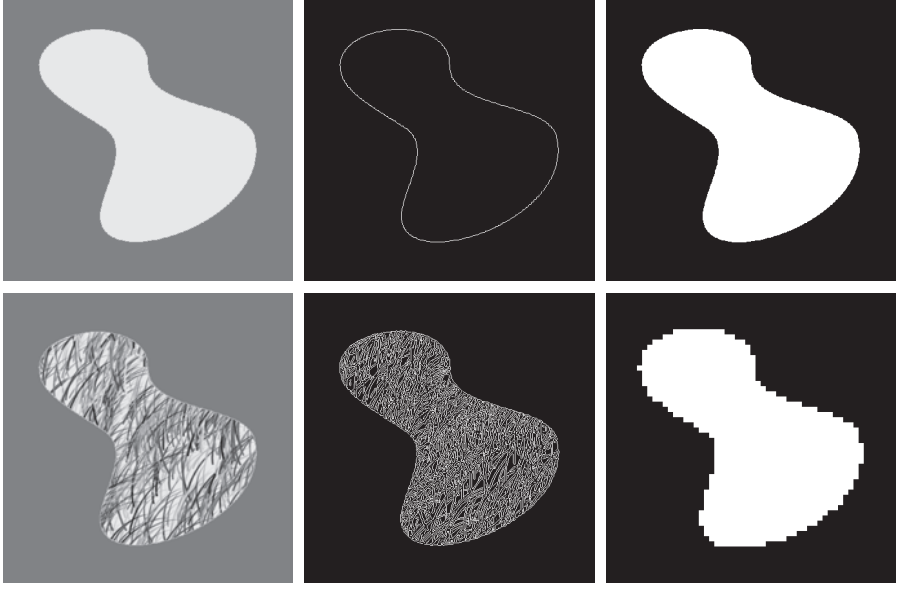
where $Q(R_k)$ is a logical predicate defined over the points in set R_k, and $\varnothing$ is the null set. The symbols $\cup$ and $\cap$ represent set union and intersection, respectively, as defined in Section 2.6. Two regions R_i and R_j are said to be *adjacent* if their union forms a connected set, as defined in Section 2.5. If the set formed by the union of two regions is not connected, the regions are said to *disjoint*.

Condition (a) indicates that the segmentation must be *complete*, in the sense that 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 8-connected). Condition (c) says 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. Finally, condition (e) indicates that two adjacent regions R_i and R_j must be different in the sense of predicate Q.[†]

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, we assume 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. 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 inside the boundary, and another level (e.g., 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

[†] In general, Q can be a compound expression such as, "$Q(R_i) = \text{TRUE}$ if the average intensity of the pixels in region 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.



a b c
d e f

FIGURE 10.1
(a) Image of a constant intensity region. (b) Boundary based on intensity discontinuities. (c) Result of segmentation. (d) Image of a texture region. (e) Result of intensity discontinuity computations (note the large number of small edges). (f) Result of segmentation based on region properties.

section are satisfied by 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 or 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 intensity discontinuities in this image. 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. However, we note that the outer region is constant, so all we need to solve this 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 8×8. 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 8×8 squares were labeled with the same intensity (smaller squares would have given a smoother region boundary). Finally, note that these results also satisfy the five segmentation 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 characteristics in which

we are interested are isolated points, lines, and edges. *Edge pixels* are pixels at which the intensity of an image changes abruptly, and *edges* (or *edge segments*) are sets of connected edge pixels (see Section 2.5 regarding connectivity). *Edge detectors* are local image processing tools designed to detect edge pixels. A *line* may be viewed as a (typically) thin 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 will discuss later, lines give rise to so-called "roof edges." Finally, an *isolated point* may be viewed as a foreground (background) pixel surrounded by background (foreground) pixels.

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.

BACKGROUND

As we saw in Section 3.5, local averaging smoothes an image. Given that averaging is analogous to integration, it is intuitive 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 *finite differences*. There are various ways to compute these differences but, as explained in Section 3.6, we require that any approximation used for first derivatives (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 second derivatives (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 an arbitrary point x of a one-dimensional function $f(x)$ by expanding the function $f(x + \Delta x)$ into a Taylor series about x

$$\begin{aligned} f(x+\Delta x) &= f(x) + \Delta x\frac{\partial f(x)}{\partial x} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 f(x)}{\partial x^2} + \frac{(\Delta x)^3}{3!}\frac{\partial^3 f(x)}{\partial x^3} + \cdots\cdots \\ &= \sum_{n=0}^{\infty}\frac{(\Delta x)^n}{n!}\frac{\partial^n f(x)}{\partial x^n} \end{aligned} \tag{10-1}$$

Remember, the notation $n!$ means "n factorial": $n! = 1\times 2\times\cdots\times n$.

where Δx is the separation between samples of f. For our purposes, this separation is measured in pixel units. Thus, following the convention in the book, $\Delta x = 1$ for the sample preceding x and $\Delta x = -1$ for the sample following x. When $\Delta x = 1$, Eq. (10-1) becomes

$$\begin{aligned} f(x+1) &= f(x) + \frac{\partial f(x)}{\partial x} + \frac{1}{2!}\frac{\partial^2 f(x)}{\partial x^2} + \frac{1}{3!}\frac{\partial^3 f(x)}{\partial x^3} + \cdots\cdots \\ &= \sum_{n=0}^{\infty}\frac{1}{n!}\frac{\partial^n f(x)}{\partial x^n} \end{aligned} \tag{10-2}$$

Although this is an expression of only one variable, we used partial derivatives notation for consistency when we discuss functions of two variables later in this section.

Similarly, when $\Delta x = -1$,

$$\begin{aligned} f(x-1) &= f(x) - \frac{\partial f(x)}{\partial x} + \frac{1}{2!}\frac{\partial^2 f(x)}{\partial x^2} - \frac{1}{3!}\frac{\partial^3 f(x)}{\partial x^3} + \cdots\cdots \\ &= \sum_{n=0}^{\infty} \frac{(-1)^n}{n!}\frac{\partial^n f(x)}{\partial x^n} \end{aligned} \tag{10-3}$$

In what follows, we compute *intensity differences* using just a few terms of the Taylor series. For first-order derivatives we use only the linear terms, and we can form differences in one of three ways.

The *forward difference* is obtained from Eq. (10-2):

$$\frac{\partial f(x)}{\partial x} = f'(x) = f(x+1) - f(x) \tag{10-4}$$

where, as you can see, we kept only the linear terms. The *backward difference* is similarly obtained by keeping only the linear terms in Eq. (10-3):

$$\frac{\partial f(x)}{\partial x} = f'(x) = f(x) - f(x-1) \tag{10-5}$$

and the *central difference* is obtained by subtracting Eq. (10-3) from Eq. (10-2):

$$\frac{\partial f(x)}{\partial x} = f'(x) = \frac{f(x+1) - f(x-1)}{2} \tag{10-6}$$

The higher terms of the series that we did not use represent the error between an exact and an approximate derivative expansion. In general, the more terms we use from the Taylor series to represent a derivative, the more accurate the approximation will be. To include more terms implies that more points are used in the approximation, yielding a lower error. However, it turns out that central differences have a lower error for the same number of points (see Problem 10.1). For this reason, derivatives are usually expressed as central differences.

The *second order* derivative based on a central difference, $\partial^2 f(x)/\partial x^2$, is obtained by adding Eqs. (10-2) and (10-3):

$$\frac{\partial^2 f(x)}{\partial x^2} = f''(x) = f(x+1) - 2f(x) + f(x-1) \tag{10-7}$$

To obtain the *third order, central derivative* we need one more point on either side of x. That is, we need the Taylor expansions for $f(x+2)$ and $f(x-2)$, which we obtain from Eqs. (10-2) and (10-3) with $\Delta x = 2$ and $\Delta x = -2$, respectively. The strategy is to combine the two Taylor expansions to eliminate all derivatives lower than the third. The result after ignoring all higher-order terms [see Problem 10.2(a)] is

$$\frac{\partial^3 f(x)}{\partial x^3} = f'''(x) = \frac{f(x+2) - 2f(x+1) + 0f(x) + 2f(x-1) - f(x-2)}{2} \quad (10\text{-}8)$$

Similarly [see Problem 10.2(b)], the *fourth* finite difference (the highest we use in the book) after ignoring all higher order terms is given by

$$\frac{\partial^4 f(x)}{\partial x^4} = f''''(x) = f(x+2) - 4f(x+1) + 6f(x) - 4f(x-1) + f(x-2) \quad (10\text{-}9)$$

Table 10.1 summarizes the first four central derivatives just discussed. Note the symmetry of the coefficients about the center point. This symmetry is at the root of why central differences have a lower approximation error for the same number of points than the other two differences. For two variables, we apply the results in Table 10.1 to each variable independently. For example,

$$\frac{\partial^2 f(x,y)}{\partial x^2} = f(x+1,y) - 2f(x,y) + f(x-1,y) \quad (10\text{-}10)$$

and

$$\frac{\partial^2 f(x,y)}{\partial y^2} = f(x,y+1) - 2f(x,y) + f(x,y-1) \quad (10\text{-}11)$$

It is easily verified that the first and second-order derivatives in Eqs. (10-4) through (10-7) satisfy the conditions stated at the beginning of this section regarding derivatives of the first and second order. To illustrate this, consider Fig. 10.2. Part (a) shows an image of various objects, a line, and an isolated point. Figure 10.2(b) shows a horizontal intensity profile (scan line) through the center of the image, 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 will discuss later, intensity transitions involving thin objects such as lines often are referred to as *roof edges*.

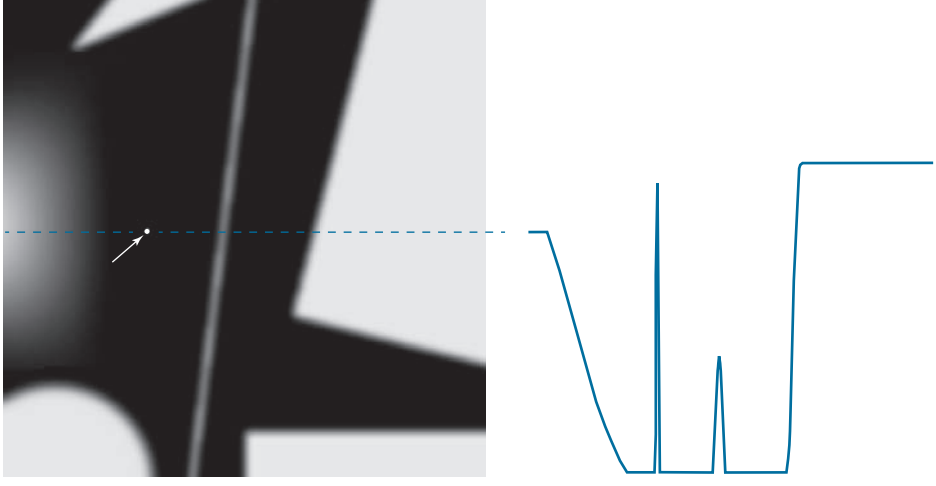
Figure 10.2(c) shows a simplified profile, with just enough points to make it possible for us to analyze manually how the first- and second-order derivatives behave as they encounter a point, a line, and the edges of objects. In this diagram the transition

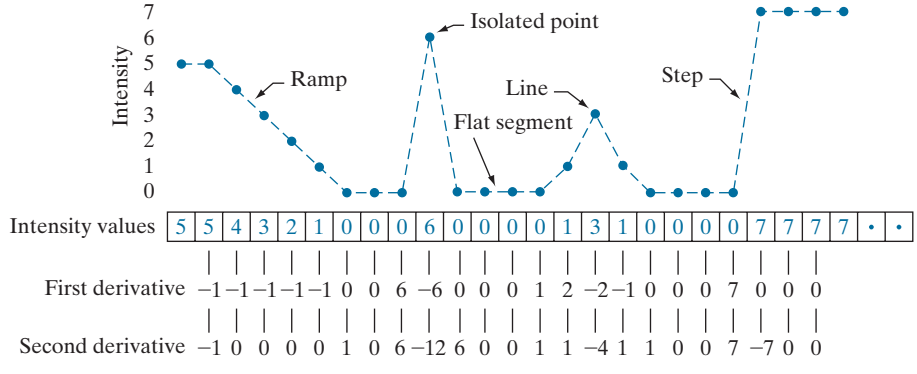
TABLE 10.1
First four central digital derivatives (finite differences) for samples taken uniformly, $\Delta x = 1$ units apart.

	$f(x+2)$	$f(x+1)$	$f(x)$	$f(x-1)$	$f(x-2)$
$2f'(x)$		1	0	–1	
$f''(x)$		1	–2	1	
$2f'''(x)$	1	–2	0	2	–1
$f''''(x)$	1	–4	6	–4	1

a b
c

FIGURE 10.2
(a) Image.
(b) Horizontal intensity profile that includes the isolated point indicated by the arrow.
(c) Subsampled profile; the dashes were added for clarity. The numbers in the boxes are the intensity values of the dots shown in the profile. The derivatives were obtained using Eqs. (10-4) for the first derivative and Eq. (10-7) for the second.





in the ramp spans four pixels, the noise point is a single pixel, the line is three pixels thick, and the transition of the step edge 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, 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 the edges of digital images resemble this type of transition, we conclude that first-order derivatives produce "thick" edges, and second-order derivatives much thinner 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 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

FIGURE 10.3
A general 3×3 spatial filter kernel. The w's are the kernel coefficients (weights).

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

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 can be used to locate edges, as we will show later in this section. As we move into the edge, 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)

In summary, we arrive at the following conclusions: (1) First-order derivatives generally produce thicker edges. (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 convolution. For the 3×3 filter kernel in Fig. 10.3, the procedure is to compute the sum of products of the kernel coefficients with the intensity values in the region encompassed by the kernel, as we explained in Section 3.4. That is, the response of the filter at the center point of the kernel is

This equation is an expansion of Eq. (3-35) for a 3×3 kernel, valid at one point, and using simplified subscript notation for the kernel coefficients.

$$\begin{aligned} Z &= w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9 \\ &= \sum_{k=1}^{9} w_k z_k \end{aligned} \tag{10-12}$$

where z_k is the intensity of the pixel whose spatial location corresponds to the location of the kth kernel coefficient.

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 which, from the discussion in Section 3.6, means using the Laplacian:

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{10-13}$$

where the partial derivatives are computed using the second-order finite differences in Eqs. (10-10) and (10-11). The Laplacian is then

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y) \quad (10\text{-}14)$$

As explained in Section 3.6, this expression can be implemented using the Laplacian kernel in Fig. 10.4(a) in Example 10.1. We then say that a point has been detected at a location (x,y) on which the kernel is centered if the absolute value of the response of the filter at that point exceeds a specified threshold. Such points are labeled 1 and all others are labeled 0 in the output image, thus producing a binary image. In other words, we use the expression:

$$g(x,y) = \begin{cases} 1 & \text{if } |Z(x,y)| > T \\ 0 & \text{otherwise} \end{cases} \quad (10\text{-}15)$$

where $g(x,y)$ is the output image, T is a nonnegative threshold, and Z is given by Eq. (10-12). 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 kernel. 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 kernel, the coefficients sum to zero, indicating that the filter response will be zero in areas of constant intensity.

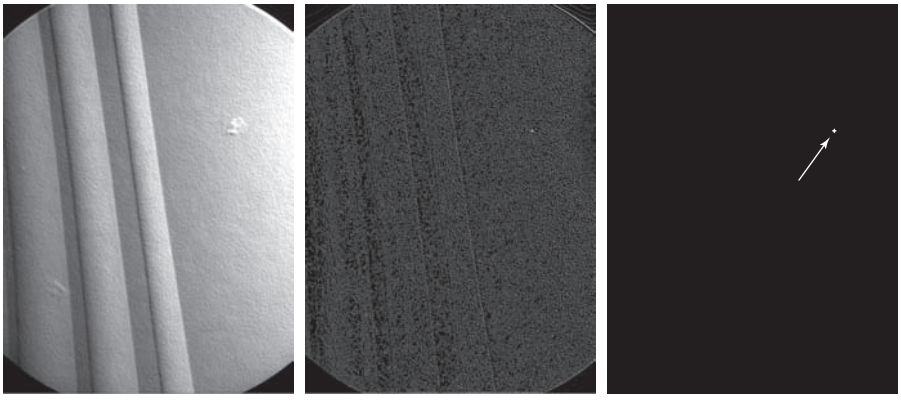
EXAMPLE 10.1: Detection of isolated points in an image.

Figure 10.4(b) is an X-ray image of a turbine blade from a jet engine. The blade has a porosity manifested by a single black pixel in the upper-right quadrant of the image. Figure 10.4(c) is the result of filtering the image with the Laplacian kernel, and Fig. 10.4(d) shows the result of Eq. (10-15) 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 at the tip of the arrow (the pixel was enlarged to enhance its visibility). This type of detection process is 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 kernel. When this condition is not satisfied, other methods discussed in this chapter are more suitable for detecting intensity changes.

LINE DETECTION

The next level of complexity is line detection. Based on the discussion earlier in this section, we know that for line detection we can expect second derivatives to result in a stronger filter response, and to produce thinner lines than first derivatives. Thus, we can use the Laplacian kernel 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.

1	1	1
1	−8	1
1	1	1



a
b c d

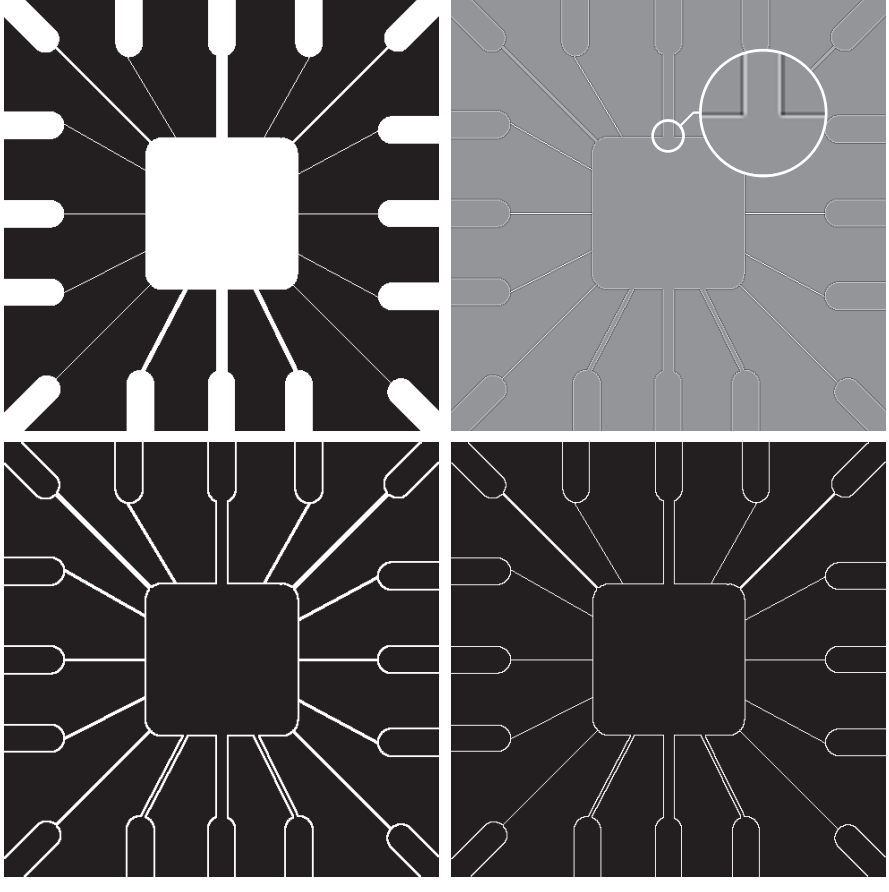
FIGURE 10.4
(a) Laplacian kernel used for point detection. (b) X-ray image of a turbine blade with a porosity manifested by a single black pixel. (c) Result of convolving the kernel with the image. (d) Result of using Eq. (10-15) was a single point (shown enlarged at the tip of the arrow). (Original image courtesy of X-TEK Systems, Ltd.)

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 (see the discussion after Example 3.18), 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 Fig. 10.5(d) shows, this approach results in thinner lines that generally are more useful. Note in Figs. 10.5(b) through (d) that when the lines are wide with respect to the size of the Laplacian kernel, the lines are separated by a zero "valley." This is not unexpected. For example, when the 3×3 kernel 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 in the following section.

The Laplacian detector kernel in Fig. 10.4(a) is isotropic, so its response is independent of direction (with respect to the four directions of the 3×3 kernel: vertical, horizontal, and two diagonals). Often, interest lies in detecting lines in *specified*



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.

directions. Consider the kernels 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 kernel. The maximum responses would occur at image locations in which a horizontal line passes through the middle row of the kernel. This is easily verified by sketching a simple array of 1's with a line of a different intensity (say, 5s) running horizontally through the array. A similar experiment would reveal that the second kernel in Fig. 10.6 responds best to lines oriented at +45°; the third kernel to vertical lines; and the fourth kernel to lines in the −45° direction. The preferred direction of each kernel is weighted with a larger coefficient (i.e., 2) than other possible directions. The coefficients in each kernel sum to zero, indicating a zero response in areas of constant intensity.

Let Z_1, Z_2, Z_3, and Z_4 denote the responses of the kernels in Fig. 10.6, from left to right, where the Zs are given by Eq. (10-12). Suppose that an image is filtered with these four kernels, one at a time. If, at a given point in the image, $|Z_k| > |Z_j|$, for all $j \neq k$, that point is said to be more likely associated with a line in the direction of kernel k. For example, if at a point in the image, $|Z_1| > |Z_j|$ for $j = 2,3,4$, that

−1	−1	−1	2	−1	−1	−1	2	−1	−1	−1	2
2	2	2	−1	2	−1	−1	2	−1	−1	2	−1
−1	−1	−1	−1	−1	2	−1	2	−1	2	−1	−1
Horizontal			+45°			Vertical			−45°		

a b c d

FIGURE 10.6 Line detection kernels. Detection angles are with respect to the axis system in Fig. 2.19, with positive angles measured counterclockwise with respect to the (vertical) x-axis.

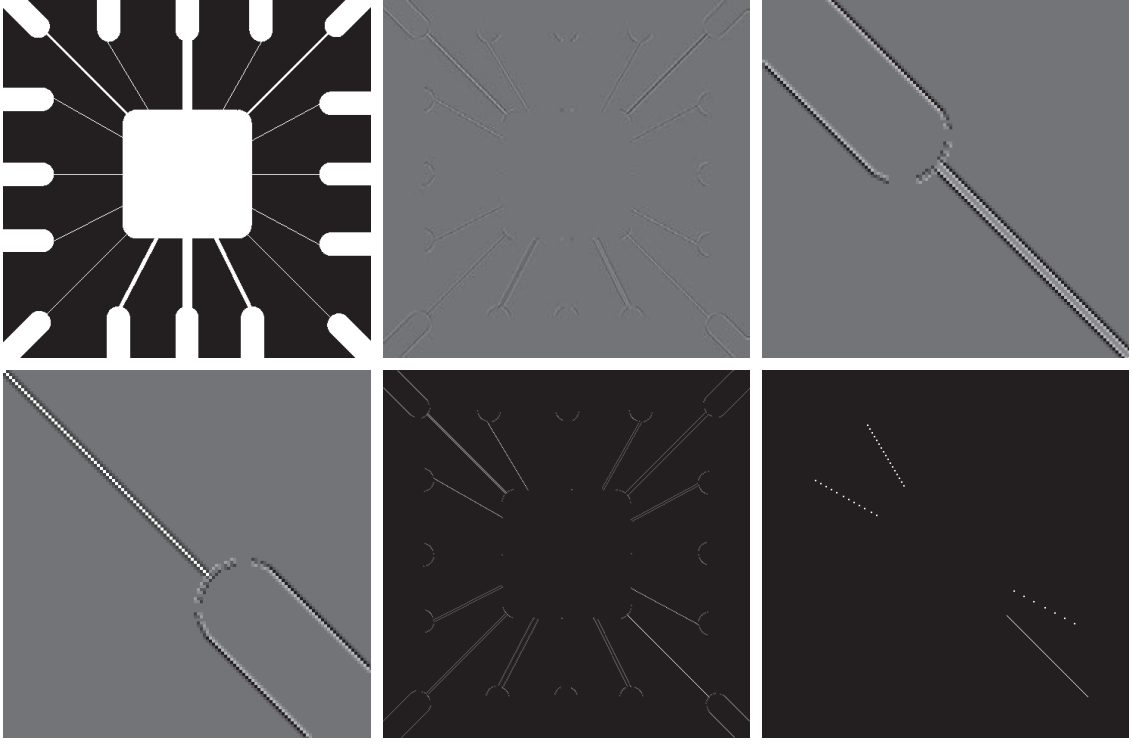
point is said to be more likely associated with a horizontal line. If we are interested in detecting all the lines in an image in the direction defined by a given kernel, we simply run the kernel through the image and threshold the absolute value of the result, as in Eq. (10-15). The nonzero points remaining after thresholding are the strongest responses which, for lines one pixel thick, correspond closest to the direction defined by the kernel. The following example illustrates this procedure.

EXAMPLE 10.3: Detecting 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 one pixel thick and oriented at +45°. For this purpose, we use the kernel in Fig. 10.6(b). Figure 10.7(b) is the result of filtering the image with that kernel. 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 in 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. The straight line segment in Fig. 10.7(d) is brighter than the segment in Fig. 10.7(c) because the line segment in the bottom right of Fig. 10.7(a) is one pixel thick, while the one at the top left is not. The kernel is "tuned" to detect one-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 254 (the maximum value in Fig. 10.7(e) minus one). Figure 10.7(f) shows in white the points whose values satisfied the condition $g > T$, 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 kernel. In the original image, these points and their immediate neighbors are oriented in such a way that the kernel produced a maximum response at those locations. These isolated points can be detected using the kernel in Fig. 10.4(a) and then deleted, or they can be deleted using morphological operators, as discussed in the last chapter.

EDGE MODELS

Edge detection is an approach used 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.



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 kernel 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 $g > T$, where g is the image in (e) and $T = 254$ (the maximum pixel value in the image minus 1). (The points in (f) were enlarged to make them easier to see.)

Edge models are classified according to their intensity profiles. A *step edge* is characterized by a transition between two intensity levels occurring ideally over the distance of one 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 one 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 later in this section was derived originally 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

a b c

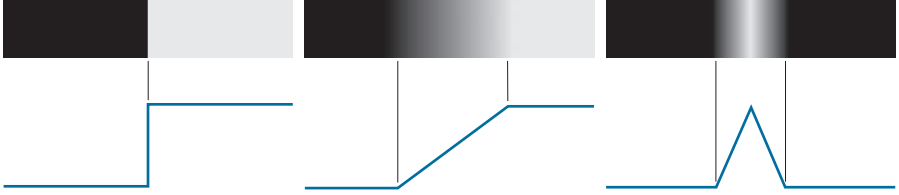


FIGURE 10.8
From left to right, models (ideal representations) of a step, a ramp, and a roof edge, and their corresponding intensity profiles.

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 to which the edge is blurred. In this model, we no longer have a single "edge point" along the profile. 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 third type of 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 the edge being determined by the thickness and sharpness of the line. In the limit, when its base is one pixel wide, a roof edge is nothing more than a one-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 the background (such as walls). The pipes appear brighter and thus create an image similar to the model in Fig. 10.8(c). 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 10.10(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. 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

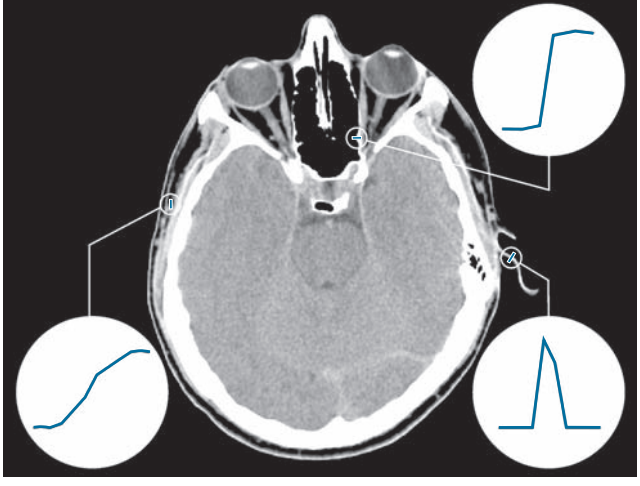
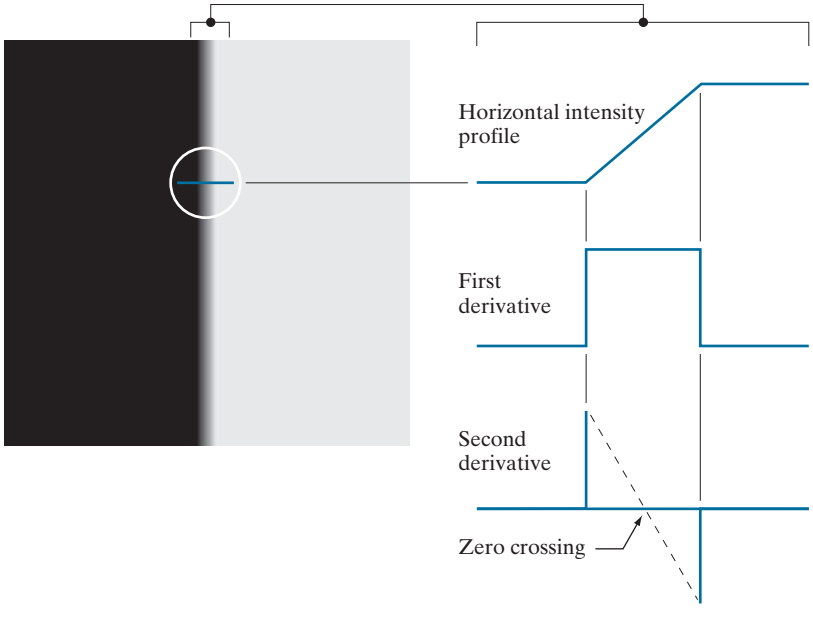


FIGURE 10.9 A 1508×1970 image showing (zoomed) actual ramp (bottom, left), step (top, right), and roof edge profiles. The profiles are from dark to light, in the areas enclosed by the small circles. The ramp and step profiles span 9 pixels and 2 pixels, respectively. The base of the roof edge is 3 pixels.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

the dark or light side of an edge. Two additional properties of the second derivative around an edge are: (1) it produces two values for every edge in an image; and (2) its zero crossings can be used for locating the centers of thick edges, as we will show later in this section. Some edge models utilize a smooth transition into and out of

a b

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



the ramp (see Problem 10.9). 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.

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

The edge models 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.

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 visually undetectable 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 corresponds 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.

In summary, the three steps performed typically for edge detection are:

1. *Image smoothing for noise reduction.* The need for this step is 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 edge-point candidates.
3. *Edge localization.* The objective of this step is to select from the candidate points only the points that are members of the set of points comprising an edge.

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

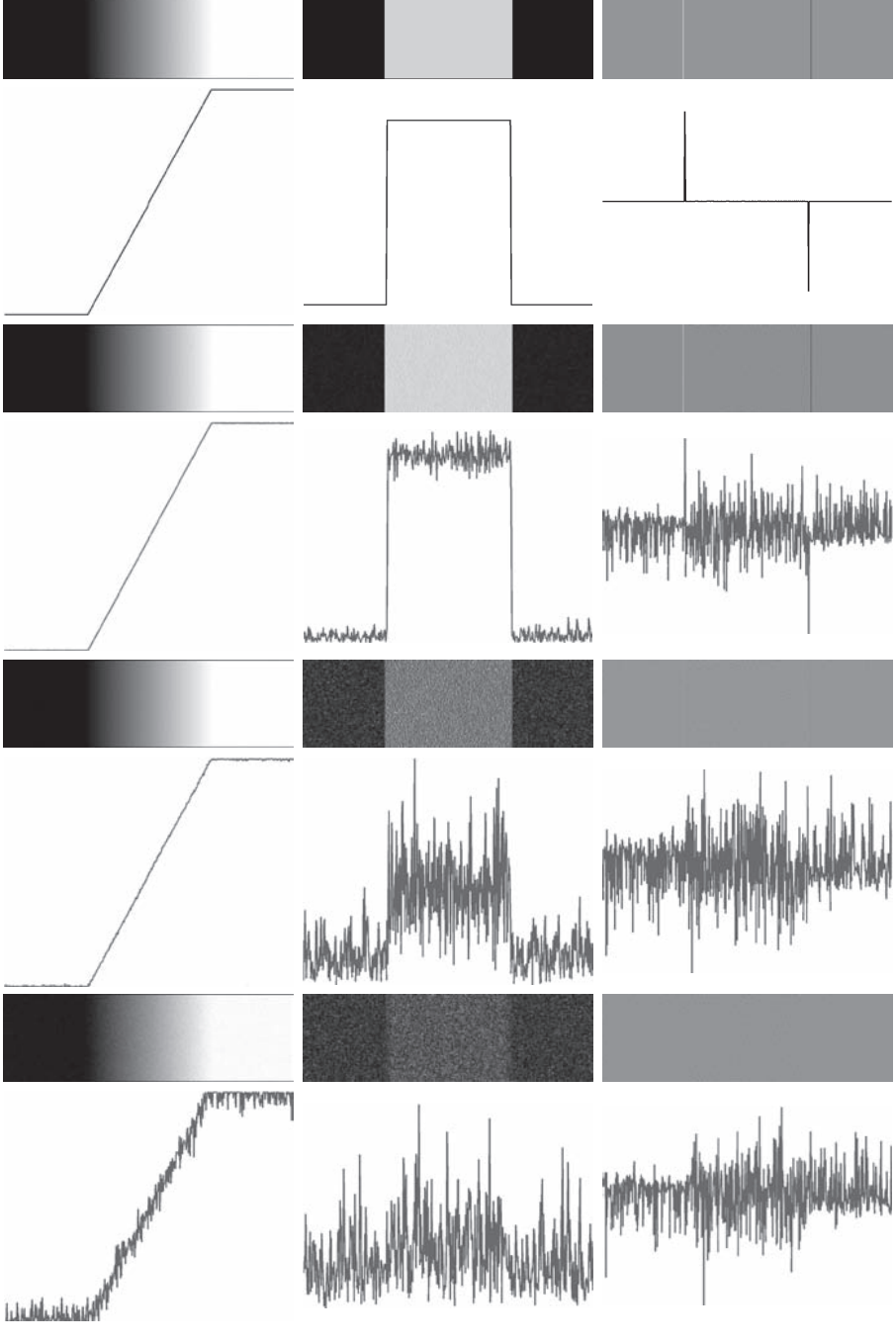


FIGURE 10.11 First column: 8-bit images with values in the range [0, 255], and intensity profiles of a ramp edge corrupted by Gaussian noise of zero mean and standard deviations of 0.0, 0.1, 1.0, and 10.0 intensity levels, respectively. Second column: First-derivative images and intensity profiles. Third column: Second-derivative images and intensity profiles.

As illustrated in the preceding discussion, detecting changes in intensity for the purpose of finding edges can be accomplished using first- or second-order derivatives. We begin with first-order derivatives, and work with second-order derivatives in the following subsection.

The Image Gradient and Its Properties

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

For convenience, we repeat here some of the gradient concepts and equations introduced in Chapter 3.

$$\nabla f(x,y) \equiv \text{grad}\left[f(x,y)\right] \equiv \begin{bmatrix} g_x(x,y) \\ g_y(x,y) \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f(x,y)}{\partial x} \\ \dfrac{\partial f(x,y)}{\partial y} \end{bmatrix} \tag{10-16}$$

This vector has the well-known property that it points in the direction of maximum rate of change of f at (x, y) (see Problem 10.10). Equation (10-16) is valid at an arbitrary (but *single*) point (x, y). When evaluated for all applicable values of x and y, $\nabla f(x, y)$ becomes a *vector image*, each element of which is a vector given by Eq. (10-16). The *magnitude*, $M(x, y)$, of this gradient vector at a point (x, y) is given by its Euclidean vector norm:

$$M(x,y) = \left\|\nabla f(x,y)\right\| = \sqrt{g_x^2(x,y) + g_y^2(x,y)} \tag{10-17}$$

This is the *value* of the rate of change in the direction of the gradient vector at point (x, y). Note that $M(x, y)$, $\left\|\nabla f(x, y)\right\|$, $g_x(x, y)$, and $g_y(x, y)$ are arrays of the same size as f, created when x and y are allowed to vary over all pixel locations in f. It is common practice to refer to $M(x, y)$ and $\left\|\nabla f(x, y)\right\|$ as the *gradient image*, or simply as the *gradient* when the meaning is clear. The summation, square, and square root operations are elementwise operations, as defined in Section 2.6.

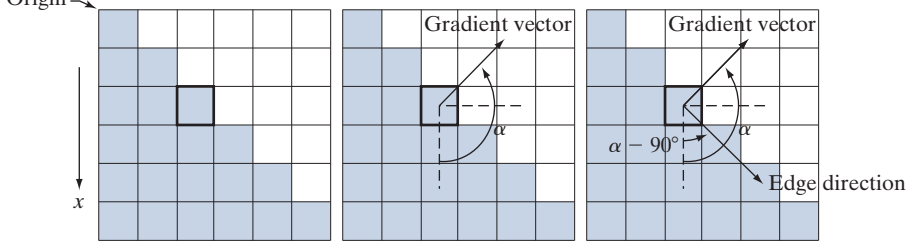
The *direction* of the gradient vector at a point (x, y) is given by

$$\alpha(x,y) = \tan^{-1}\left[\frac{g_y(x,y)}{g_x(x,y)}\right] \tag{10-18}$$

Angles are measured in the counterclockwise direction with respect to the x-axis (see Fig. 2.19). This is also an image of the same size as f, created by the elementwise division of g_x and g_y over all applicable values of x and y. The following example illustrates, the direction of an edge at a point (x, y) is orthogonal to the direction, $\alpha(x, y)$, of the gradient vector at the point.

EXAMPLE 10.5: Computing the gradient.

Figure 10.12(a) shows a zoomed section of an image containing a straight edge segment. Each square 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 shaded pixels in this figure are assumed to have value 0, and the white



a b c

FIGURE 10.12 Using the gradient to determine edge strength and direction at a point. Note that the edge direction is perpendicular to the direction of the gradient vector at the point where the gradient is computed. Each square represents one pixel. (Recall from Fig. 2.19 that the origin of our coordinate system is at the top, left.)

pixels have value 1. We discuss after this example an approach for computing the derivatives in the x- and y-directions using a 3×3 neighborhood centered at a point. The method consists 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 of the neighborhood to obtain the partial derivative in the y-direction. It then 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,

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

from which we obtain $\|\nabla f\| = 2\sqrt{2}$ at that point. Similarly, the direction of the gradient vector at the same point follows from Eq. (10-18): $\alpha = \tan^{-1}\left(g_y/g_x\right) = -45°$, which is the same as 135° measured in the positive (counterclockwise) direction with respect to the x-axis in our image coordinate system (see Fig. 2.19). Figure 10.12(b) shows the gradient vector and its direction angle.

As mentioned earlier, the direction of an edge at a point is orthogonal to the gradient vector at that point. So the direction angle of the edge in this example is $\alpha - 90° = 135° - 90° = 45°$, as Fig. 10.12(c) shows. 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, the resulting vector is 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. For the gradient, we typically use a forward or centered finite difference (see Table 10.1). Using forward differences we obtain

$$g_x(x,y) = \frac{\partial f(x,y)}{\partial x} = f(x+1,y) - f(x,y) \tag{10-19}$$

a b

FIGURE 10.13
1-D kernels used to implement Eqs. (10-19) and (10-20).

−1
1

−1	1

and

$$g_y(x,y) = \frac{\partial f(x,y)}{\partial y} = f(x,y+1) - f(x,y) \tag{10-20}$$

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

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

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

$$g_x = \frac{\partial f}{\partial x} = (z_9 - z_5) \tag{10-21}$$

and

$$g_y = \frac{\partial f}{\partial y} = (z_8 - z_6) \tag{10-22}$$

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

Kernels of size 2×2 are simple conceptually, but they are not as useful for computing edge direction as kernels that are symmetric about their centers, the smallest of which are of size 3×3. These kernels 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 kernels of size 3×3 are given by

Observe that these two equations are first-order central differences as given in Eq. (10-6), but multiplied by 2.

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

and

$$g_y = \frac{\partial f}{\partial y} = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7) \tag{10-23}$$

In this formulation, 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 derivative in the y-direction. Intuitively, we would expect these approximations to be more accurate than the approximations obtained using the Roberts operators. Equations (10-22) and (10-23) can be implemented over an entire image by filtering it with the two kernels in Figs. 10.14(d) and (e). These kernels are called the *Prewitt operators* (Prewitt [1970]).

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

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 kernels 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

$$g_x = \frac{\partial f}{\partial x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3) \tag{10-24}$$

and

$$g_y = \frac{\partial f}{\partial y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) \tag{10-25}$$

It can be demonstrated (see Problem 10.12) that using a 2 in the center location provides image smoothing. Figures 10.14(f) and (g) show the kernels used to implement Eqs. (10-24) and (10-25). These kernels are called the *Sobel operators* (Sobel [1970]).

The Prewitt kernels are simpler to implement than the Sobel kernels, but the slight computational difference between them typically is not an issue. The fact that the Sobel kernels have better noise-suppression (smoothing) characteristics makes them preferable because, as mentioned earlier in the discussion of Fig. 10.11, noise suppression is an important issue when dealing with derivatives. Note that the

Recall the important result in Problem 3.32 that using a kernel whose coefficients sum to zero produces a filtered image whose pixels also sum to zero. This implies in general that some pixels will be negative. Similarly, if the kernel coefficients sum to 1, the sum of pixels in the original and filtered images will be the same (see Problem 3.31).

coefficients of all the kernels in Fig. 10.14 sum to zero, thus giving a response of zero in areas of constant intensity, as expected of derivative operators.

Any of the pairs of kernels from Fig. 10.14 are convolved with an image to obtain the gradient components g_x and g_y at every pixel location. These two partial derivative arrays are then used to estimate edge strength and direction. Obtaining the magnitude of the gradient requires the computations in Eq. (10-17). This implementation is not always desirable because of the computational burden required by squares and square roots, and an approach used frequently is to approximate the magnitude of the gradient by absolute values:

$$M(x, y) \approx |g_x| + |g_y| \tag{10-26}$$

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 kernels such as the Prewitt and Sobel kernels are used to compute g_x and g_y because these kernels give isotropic results only for vertical and horizontal edges. This means that results would be isotropic only for edges in those two directions anyway, regardless of which of the two equations is used. That is, Eqs. (10-17) and (10-26) give identical results for vertical and horizontal edges when either the Sobel or Prewitt kernels are used (see Problem 10.11).

The 3×3 kernels in Fig. 10.14 exhibit their strongest response predominantly for vertical and horizontal edges. The *Kirsch compass kernels* (Kirsch [1971]) in Fig. 10.15, are designed to detect edge magnitude *and* direction (angle) in all eight compass directions. Instead of computing the magnitude using Eq. (10-17) and angle using Eq. (10-18), Kirsch's approach was to determine the edge magnitude by convolving an image with all eight kernels and assign the edge magnitude at a point as the response of the kernel that gave strongest convolution value at that point. The edge angle at that point is then the direction associated with that kernel. For example, if the strongest value at a point in the image resulted from using the north (N) kernel, the edge magnitude at that point would be assigned the response of that kernel, and the direction would be 0° (because compass kernel pairs differ by a rotation of 180°; choosing the maximum response will always result in a positive number). Although when working with, say, the Sobel kernels, we think of a north or south edge as being vertical, the N and S compass kernels differentiate between the two, the difference being the direction of the intensity transitions defining the edge. For example, assuming that intensity values are in the range $[0, 1]$, the binary edge in Fig. 10.8(a) is defined by black (0) on the left and white (1) on the right. When all Kirsch kernels are applied to this edge, the N kernel will yield the highest value, thus indicating an edge oriented in the north direction (at the point of the computation).

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

Figure 10.16 illustrates the Sobel 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

a b c d
e f g h

FIGURE 10.15
Kirsch compass kernels. The edge direction of strongest response of each kernel is labeled below it.

−3	−3	5	−3	5	5	5	5	5	5	5	−3
−3	0	5	−3	0	5	−3	0	−3	5	0	−3
−3	−3	5	−3	−3	−3	−3	−3	−3	−3	−3	−3
N			NW			W			SW		
5	−3	−3	−3	−3	−3	−3	−3	−3	−3	−3	−3
5	0	−3	5	0	−3	−3	0	−3	−3	0	5
5	−3	−3	5	5	−3	5	5	5	−3	5	5
S			SE			E			NE		

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. In contrast, 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.

a b
c d

FIGURE 10.16
(a) 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 kernel in Fig. 10.14(f) to filter the image. (c) $|g_y|$, obtained using the kernel in Fig. 10.14(g). (d) The gradient image, $|g_x| + |g_y|$.

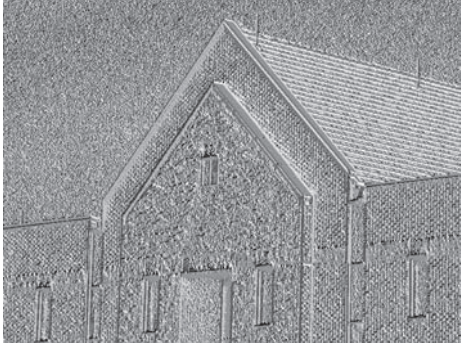


FIGURE 10.17 Gradient angle image computed using Eq. (10-18). Areas of constant intensity in this image indicate that the direction of the gradient vector is the same at all the pixel locations in those regions.

Figure 10.17 shows the gradient angle image computed using Eq. (10-18). 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. As we will show later in this section, angle information plays a key supporting role in the implementation of the Canny edge detection algorithm, a widely used edge detection scheme.

The original image in Fig. 10.16(a) is of reasonably high resolution, 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. One way to reduce fine detail is to smooth the image prior to computing the edges. 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 kernel now shows almost no contribution due to the bricks, with the results being dominated mostly by the principal edges in the image.

Figures 10.16 and 10.18 show that the horizontal and vertical Sobel kernels do not differentiate between edges in the $\pm 45°$ directions. If it is important to emphasize edges oriented in particular diagonal directions, then one of the Kirsch kernels in Fig. 10.15 should be used. Figures 10.19(a) and (b) show the responses of the $45°$ (NW) and $-45°$ (SW) Kirsch kernels, respectively. The stronger diagonal selectivity of these kernels is evident in these figures. Both kernels have similar responses to horizontal and vertical edges, but the response in these directions is weaker.

Combining the Gradient with Thresholding

The threshold used to generate Fig. 10.20(a) was selected so that most of the small edges caused by the bricks were eliminated. This was the same objective as when the image in Fig. 10.16(a) was smoothed prior to computing the gradient.

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 objective is to threshold the gradient image. For example, Fig. 10.20(a) shows the gradient image from Fig. 10.16(d), thresholded so 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

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 kernel prior to edge detection.

black. Comparing this image with Fig. 10.16(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 sloping line defining the far edge of the roof (see arrow), are broken in the thresholded image.

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 reduced number of broken edges; for instance, compare the corresponding edges identified by the arrows in Figs. 10.20(a) and (b).

a b

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

a b

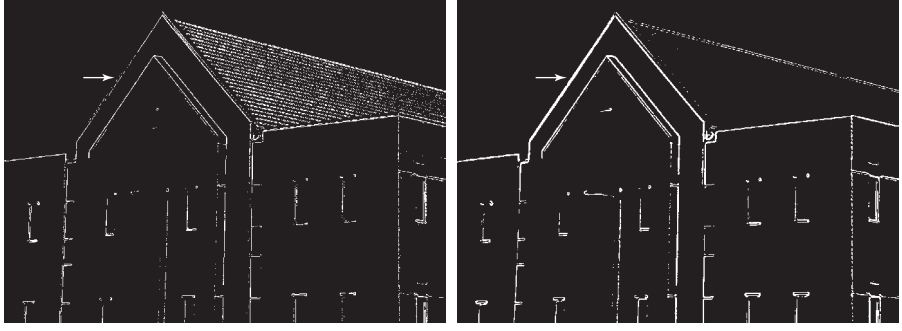


FIGURE 10.20
(a) Result of thresholding Fig. 10.16(d), the gradient of the original image. (b) Result of thresholding Fig. 10.18(d), the gradient of the smoothed image.

MORE ADVANCED TECHNIQUES FOR EDGE DETECTION

The edge-detection methods discussed in the previous subsections are based on filtering an image with one or more kernels, with no provisions made for edge characteristics and noise content. In this section, we discuss more advanced techniques that 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 small operators, such as the Sobel kernels discussed earlier. Marr and Hildreth argued (1) that intensity changes are not independent of image scale, implying that their detection requires using 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).

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 suggested that the most satisfactory operator fulfilling these conditions is the filter $\nabla^2 G$ where, as defined in Section 3.6, ∇^2 is the Laplacian, and G is the 2-D Gaussian function

Equation (10-27) differs from the definition of a Gaussian function by a multiplicative constant [see Eq. (3-45)]. Here, we are interested only in the general shape of the Gaussian function.

$$G(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}} \tag{10-27}$$

with standard deviation σ (sometimes σ is called the *space constant* in this context). We find an expression for $\nabla^2 G$ by applying the Laplacian to Eq. (10-27):

$$\nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}$$
$$= \frac{\partial}{\partial x}\left(\frac{-x}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}\right) + \frac{\partial}{\partial y}\left(\frac{-y}{\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}\right) \quad (10\text{-}28)$$
$$= \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}} + \left(\frac{y^2}{\sigma^4} - \frac{1}{\sigma^2}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Collecting terms, we obtain

$$\nabla^2 G(x,y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right) e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (10\text{-}29)$$

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

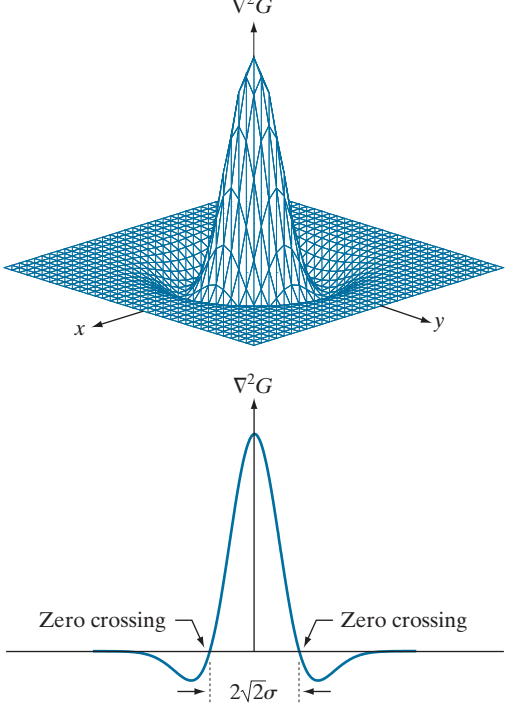
Figures 10.21(a) through (c) show a 3-D plot, image, and cross-section of the negative of the LoG function (note that the zero crossings of the LoG occur at $x^2 + y^2 = 2\sigma^2$, which defines a circle of radius $\sqrt{2}\sigma$ centered on the peak of the Gaussian function). Because of the shape illustrated in Fig. 10.21(a), the LoG function sometimes is called the *Mexican hat operator*. Figure 10.21(d) shows a 5×5 kernel that approximates the shape in Fig. 10.21(a) (normally, we would use the negative of this kernel). 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 decrease 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 kernel is zero in areas of constant intensity.

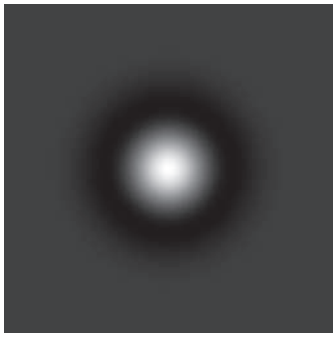
Filter kernels of arbitrary size (but fixed σ) can be generated by sampling Eq. (10-29), and scaling the coefficients so that they sum to zero. A more effective approach for generating a LoG kernel is sampling Eq. (10-27) to the desired size, then convolving the resulting array with a Laplacian kernel, such as the kernel in Fig. 10.4(a). Because convolving an image with a kernel whose coefficients sum to zero yields an image whose elements also sum to zero (see Problems 3.32 and 10.16), this approach automatically satisfies the requirement that the sum of the LoG kernel coefficients be zero. We will discuss size selection for 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 the averaging filter used in Fig. 10.18, the Gaussian function is smooth in both the spatial and frequency domains (see Section 4.8), and is thus less likely to introduce artifacts (e.g., ringing) not present in the original image. The other idea concerns the second-derivative properties of the Laplacian operator, ∇^2. 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 kernel

a b
c d

FIGURE 10.21
(a) 3-D 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 kernel approximation to the shape in (a). The negative of this kernel would be used in practice.





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

direction, thus avoiding having to use multiple kernels to calculate the strongest response at any point in the image.

The Marr-Hildreth algorithm consists of convolving the LoG kernel with an input image,

$$g(x,y) = \left[\nabla^2 G(x,y)\right] \star f(x,y) \tag{10-30}$$

This expression is implemented in the spatial domain using Eq. (3-35). It can be implemented also in the frequency domain using Eq. (4-104).

and then finding the zero crossings of $g(x,y)$ to determine the locations of edges in $f(x,y)$. Because the Laplacian and convolution are linear processes, we can write Eq. (10-30) as

$$g(x,y) = \nabla^2 \left[G(x,y) \star f(x,y)\right] \tag{10-31}$$

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 kernel obtained by sampling Eq. (10-27).
2. Compute the Laplacian of the image resulting from Step 1 using, for example, the 3×3 kernel in Fig. 10.4(a). [Steps 1 and 2 implement Eq. (10-31).]
3. Find the zero crossings of the image from Step 2.

To specify the size of the Gaussian kernel, recall from our discussion of Fig. 3.35 that the values of a Gaussian function at a distance larger than 3σ from the mean are small enough so that they can be ignored. As discussed in Section 3.5, this implies using a Gaussian kernel of size $\lceil 6\sigma \rceil \times \lceil 6\sigma \rceil$, where $\lceil 6\sigma \rceil$ denotes the ceiling of 6σ; that is, smallest integer not less than 6σ. Because we work with kernels of odd dimensions, we would use the smallest *odd* integer satisfying this condition. Using a kernel smaller than this will "truncate" the LoG function, with the degree of truncation being inversely proportional to the size of the kernel. Using a larger kernel would make little difference in the result.

As explained in Section 3.5, $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ denote the *ceiling* and *floor* functions. That is, the ceiling and floor functions map a real number to the smallest following, or the largest previous, integer, respectively.

One approach for finding the zero crossings at any pixel, p, of the filtered image, $g(x, y)$, is to use 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.

Attempts to find zero crossings by finding the coordinates (x, y) where $g(x, y) = 0$ are impractical because of noise and other computational inaccuracies.

Computing zero crossings is 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.

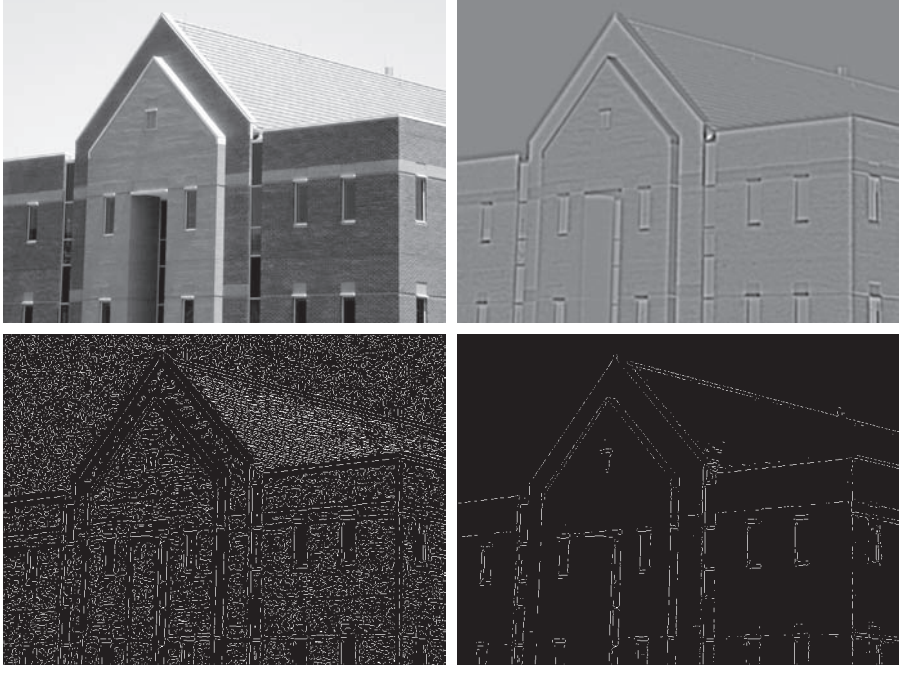
EXAMPLE 10.7: Illustration of the Marr-Hildreth edge-detection method.

Figure 10.22(a) shows the 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 $n = 25$ to satisfy the size condition stated above. 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 just discussed, 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 (see Problem 10.17). 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. 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. This type of performance is virtually impossible to obtain using the gradient-based edge-detection techniques discussed earlier. 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.

It is possible to approximate the LoG function in Eq. (10-29) by a *difference of Gaussians* (DoG):

$$D_G(x, y) = \frac{1}{2\pi\sigma_1^2} e^{-\frac{x^2+y^2}{2\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2} e^{-\frac{x^2+y^2}{2\sigma_2^2}} \tag{10-32}$$



a b
c d

FIGURE 10.22
(a) Image of size 834×1114 pixels, with intensity values scaled to the range $[0, 1]$. (b) Result 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.

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. (10-32) with a ratio of standard deviations of 1.75:1. Using the ratio 1.6:1 preserves the basic characteristics of these observations and also provides a closer "engineering" approximation to the LoG function (Marr and Hildreth [1980]). In order for the LoG and DoG to have the same zero crossings, the value of σ for the LoG must be selected based on the following equation (see Problem 10.19):

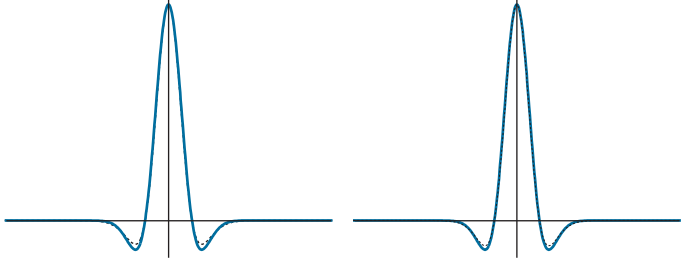
$$\sigma^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 - \sigma_2^2} \ln\left[\frac{\sigma_1^2}{\sigma_2^2}\right] \tag{10-33}$$

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.

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 the solid lines, and the DoG profiles are dotted. The curves shown are intensity profiles through the center of the LoG and DoG arrays, generated by sampling Eqs. (10-29) and (10-32), 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 slightly closer approximation of the LoG and DoG functions (for example, compare the bottom lobes of the two figures).

a b

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



Gaussian kernels are separable (see Section 3.4). Therefore, both the LoG and the DoG filtering operations can be implemented with 1-D convolutions instead of using 2-D convolutions directly (see Problem 10.19). For an image of size $M \times N$ and a kernel 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.
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 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*,

$$\frac{d}{dx}e^{-\frac{x^2}{2\sigma^2}} = \frac{-x}{\sigma^2}e^{-\frac{x^2}{2\sigma^2}} \tag{10-34}$$

[†]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.

where the approximation was only about 20% worse that using the optimized numerical solution (a difference of this magnitude generally is visually imperceptible in most applications).

Generalizing the preceding 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:

$$G(x, y) = e^{-\frac{x^2 + y^2}{2\sigma^2}} \tag{10-35}$$

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

$$f_s(x, y) = G(x, y) \star f(x, y) \tag{10-36}$$

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

$$M_s(x, y) = \|\nabla f_s(x, y)\| = \sqrt{g_x^2(x, y) + g_y^2(x, y)} \tag{10-37}$$

and

$$\alpha(x, y) = \tan^{-1}\left[\frac{g_y(x, y)}{g_x(x, y)}\right] \tag{10-38}$$

with $g_x(x, y) = \partial f_s(x, y)/\partial x$ and $g_y(x, y) = \partial f_s(x, y)/\partial y$. Any of the derivative filter kernel pairs in Fig. 10.14 can be used to obtain $g_x(x, y)$ and $g_y(x, y)$. Equation (10-36) is implemented using an $n \times n$ Gaussian kernel whose size is discussed below. Keep in mind that $\|\nabla f_s(x, y)\|$ and $\alpha(x, y)$ are arrays of the same size as the image from which they are computed.

Gradient image $\|\nabla f_s(x, y)\|$ typically contains wide ridges around local maxima. The next step is to thin those ridges. One approach is to use *nonmaxima suppression*. The essence of this approach is to specify a number of discrete orientations 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 ranges, 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. (10-38). As Fig. 10.24(b) shows, if the edge normal is in the range of directions from −22.5° to

[†]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.

a b
c

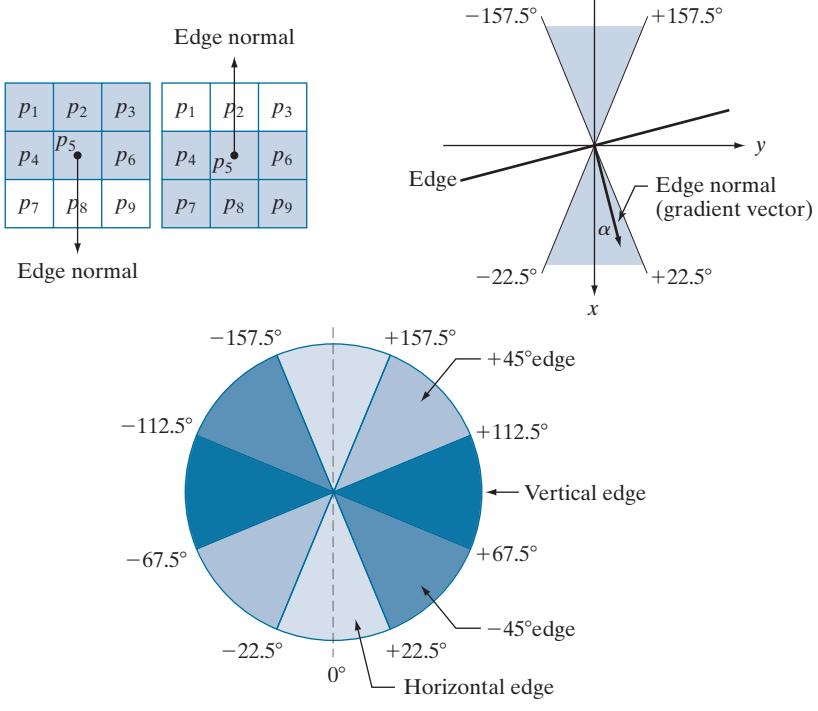


FIGURE 10.24
(a) Two possible orientations of a horizontal edge (shaded) in a 3×3 neighborhood. (b) Range of values (shaded) of α, the direction angle of the edge normal for a horizontal edge. (c) The angle ranges of the edge normals for the four types of edge directions in a 3×3 neighborhood. Each edge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

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 an arbitrary point (x, y) in α:

1. Find the direction d_k that is closest to $\alpha(x, y)$.
2. Let K denote the value of $\|\nabla f_s\|$ at (x, y). If K is less than the value of $\|\nabla f_s\|$ at one or both of the neighbors of point (x, y) along d_k, let $g_N(x, y) = 0$ (suppression); otherwise, let $g_N(x, y) = K$.

When repeated for all values of x and y, this procedure yields a nonmaxima suppressed image $g_N(x, y)$ that is of the same size as $f_s(x, y)$. 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 of interest in Step 2 would be p_2 and p_8. Image $g_N(x, y)$ contains only the thinned edges; it is equal to image $\|\nabla f_s(x, y)\|$ with the nonmaxima edge points suppressed.

The final operation is to threshold $g_N(x, y)$ to reduce false edge points. In the Marr-Hildreth algorithm 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 valid edge points will be eliminated (*false negatives*). Canny's algorithm attempts to

improve on this situation by using *hysteresis thresholding* which, as we will discuss in Section 10.3, uses two thresholds: a low threshold, T_L and a high threshold, T_H. Experimental evidence (Canny [1986]) suggests that the ratio of the high to low threshold should be in the range of 2:1 to 3:1.

We can visualize the thresholding operation as creating two additional images:

$$g_{NH}(x, y) = g_N(x, y) \geq T_H \tag{10-39}$$

and

$$g_{NL}(x, y) = g_N(x, y) \geq T_L \tag{10-40}$$

Initially, $g_{NH}(x, y)$ and $g_{NL}(x, y)$ are set to 0. After thresholding, $g_{NH}(x, y)$ will usually have fewer nonzero pixels than $g_{NL}(x, y)$, 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

$$g_{NL}(x, y) = g_{NL}(x, y) - g_{NH}(x, y) \tag{10-41}$$

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 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, the former can still be thicker than one pixel. To obtain edges one pixel thick, it is typical to follow Step 4 with one pass of an edge-thinning algorithm (see Section 9.5).

As mentioned earlier, smoothing is accomplished by convolving the input image with a Gaussian kernel whose size, $n \times n$, must be chosen. Once a value of σ has been specified, we can use the approach discussed in connection with the Marr-Hildreth algorithm to determine an odd value of n that provides the "full" smoothing capability of the Gaussian filter for the specified value of σ.

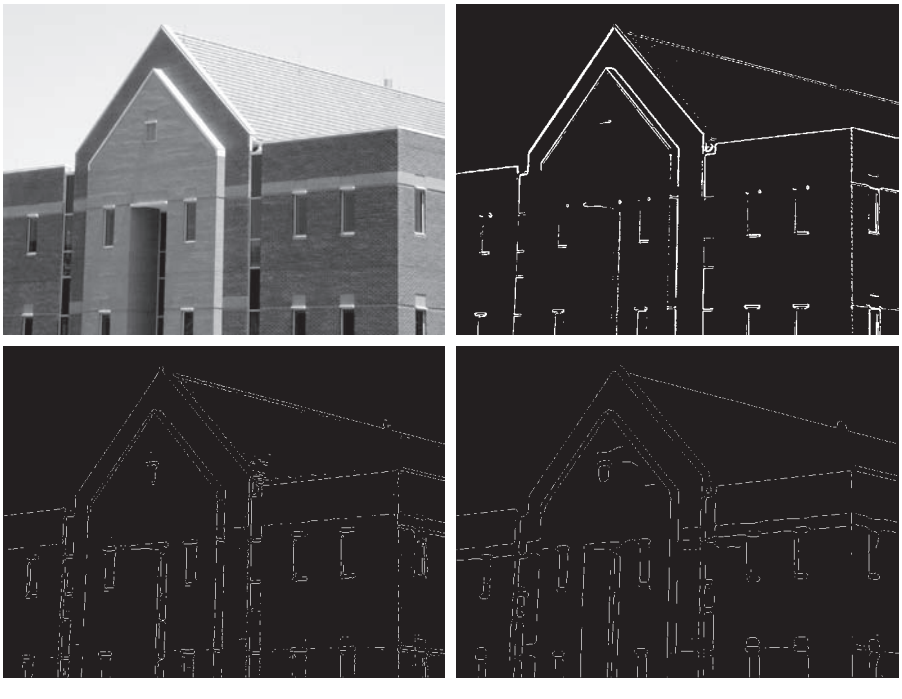
Usually, selecting a suitable value of σ for the first time in an application requires experimentation.

Some final comments on implementation: As noted earlier in the discussion of the Marr-Hildreth edge detector, the 2-D Gaussian function in Eq. (10-35) 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. (10-19) and (10-20), we can also implement the gradient computations required for Step 2 as 1-D convolutions (see Problem 10.22).

EXAMPLE 10.8: Illustration and comparison 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 result in Fig. 10.20(b) obtained 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 of the bricks and the roof tiles.

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 kernel of size 25×25, which corresponds to the smallest odd integer not less than 6σ. These parameters were chosen experimentally



a b
c d

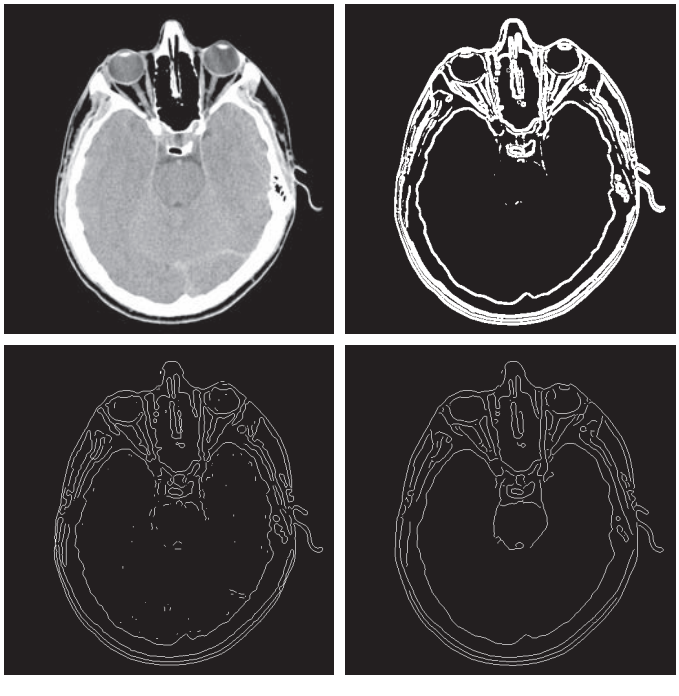
FIGURE 10.25
(a) Original image of size 834×1114 pixels, with intensity values scaled to the range $[0, 1]$.
(b) Thresholded gradient of the smoothed image.
(c) Image obtained using the Marr-Hildreth algorithm.
(d) Image obtained using the Canny algorithm. Note the significant improvement of the Canny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

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 see in the Canny result significant improvements in detail of the principal edges and, at the same time, more rejection of irrelevant features. For example, note 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 method detected 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 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 using a 5×5 averaging kernel. 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 kernel of size 19×19. Figure 10.26(d) was obtained using the Canny algorithm with $T_L = 0.05, T_H = 0.15$ (3 times the value of the low threshold), $\sigma = 2$, and a kernel of size 13×13.



a b
c d

FIGURE 10.26
(a) Head CT image of size 512×512 pixels, with intensity values scaled to the range $[0, 1]$.
(b) Thresholded gradient of the 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.)

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

The price paid for the improved performance of the Canny algorithm is a significantly more complex implementation than the two approaches discussed earlier. 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, the Marr-Hildreth and Canny algorithms, especially the latter, offer superior alternatives.

LINKING EDGE POINTS

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 caused by nonuniform illumination, and other effects that introduce 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 two 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), and the second is a global approach that works with an entire edge map. As it turns out, linking points along the boundary of a region is also an important aspect of some of the segmentation methods discussed in the next chapter, and in extracting features from a segmented image, as we will do in Chapter 11. Thus, you will encounter additional edge-point linking methods in the next two chapters.

Local Processing

A simple approach 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 preceding sections. 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 local analysis are (1) the strength (magnitude) and (2) the direction of the gradient vector. The first property is based on Eq. (10-17). 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

$$|M(s,t) - M(x,y)| \leq E \tag{10-42}$$

where E is a positive threshold.

The direction angle of the gradient vector is given by Eq. (10-18). An edge pixel with coordinates (s,t) in S_{xy} has an *angle* similar to the pixel at (x,y) if

$$|\alpha(s,t) - \alpha(x,y)| \leq A \tag{10-43}$$

where A is a positive angle threshold. As noted earlier, 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 considered to be linked to the pixel at (x,y) if both magnitude and direction criteria are satisfied. This process is repeated for every edge pixel. As the center of the neighborhood is moved from pixel to pixel, a record of linked points is kept. 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, $g(x,y)$, whose value at any point (x,y) is given by:

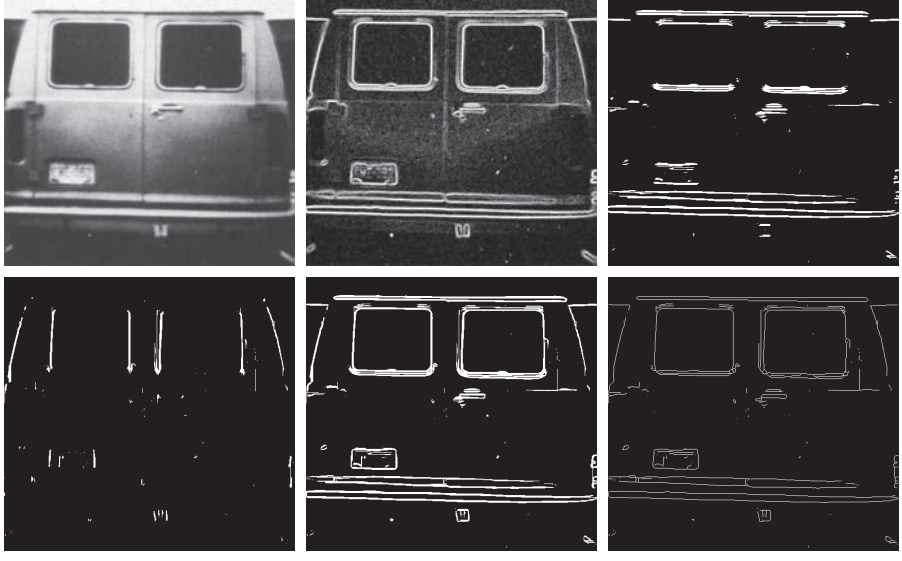
$$g(x,y) = \begin{cases} 1 & \text{if } M(x,y) > T_M \text{ AND } \alpha(x,y) = A \pm T_A \\ 0 & \text{otherwise} \end{cases}$$

   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 0's) in each row that do not exceed a specified length, L. Note that, by definition, a gap is bounded at both ends by one or more 1's. The rows are processed individually, with no "memory" kept 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 a 534×566 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



a b c
d e f

FIGURE 10.27
(a) 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 (c) and (d). (f) Final result, using morphological thinning. (Original image courtesy of Perceptics Corporation.)

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, $A = 90°$, $T_A = 45°$, and filling all gaps of 25 or fewer pixels (approximately 5% of the image width). 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. As Fig. 10.27(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 have distinctive proportions (e.g., a 2:1 ratio in U.S. plates).

Global Processing Using the Hough Transform

The method discussed in the previous section is applicable in situations in which knowledge about pixels belonging to individual objects is available. Often, we have to work in unstructured environments in which all we have is an edge map 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 all lines determined by every pair of points, then find all subsets of points that are close to particular lines. This approach involves finding $n(n-1)/2 \sim n^2$ lines, then performing $(n)(n(n-1))/2 \sim n^3$

comparisons of every point to all lines. This is a computationally prohibitive task in most applications.

The original formulation of the Hough transform presented here works with straight lines. For a generalization to arbitrary shapes, see Ballard [1981].

Hough [1962] proposed an alternative approach, commonly referred to as the *Hough transform*. Let (x_i, y_i) denote a point in the xy-plane and consider 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 point (x_i, y_i). Furthermore, a second point (x_j, y_j) also has a single line in parameter space associated with it, which intersects the line associated with (x_i, y_i) at some point (a', b') in parameter space, 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 (we are assuming, of course, that the lines are not parallel). In fact, *all* points on this line have lines in parameter space that intersect at (a', b'). Figure 10.28 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. However, a difficulty with this approach is that a, (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:

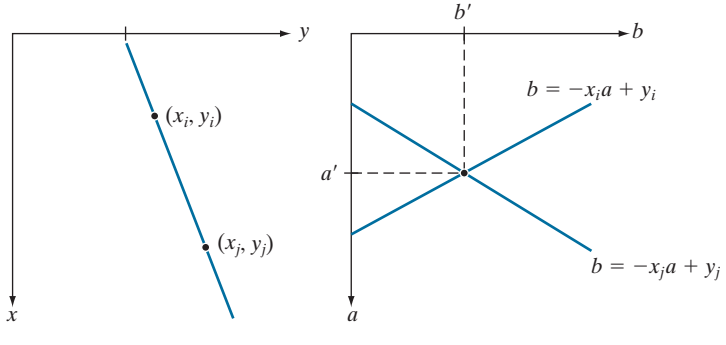
$$x\cos\theta + y\sin\theta = \rho \tag{10-44}$$

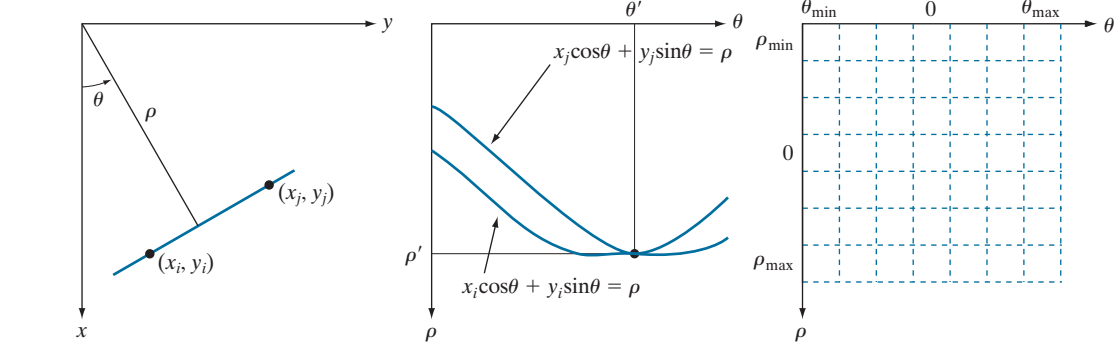
Figure 10.29(a) illustrates the geometrical interpretation of the parameters ρ and θ. A horizontal line has $\theta = 0°$, with ρ being equal to the positive x-intercept. Similarly, a vertical line has $\theta = 90°$, with ρ being equal to the positive y-intercept, or $\theta = -90°$, with ρ being equal to the negative y-intercept (we limit the angle to the range $-90° \le \theta \le 90°$). Each sinusoidal curve in Figure 10.29(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. 10.29(b) corresponds to the line that passes through both (x_i, y_i) and (x_j, y_j) in Fig. 10.29(a).

The computational attractiveness of the Hough transform arises from subdividing the $\rho\theta$ parameter space into so-called *accumulator cells*, as Fig. 10.29(c) illustrates, where $(\rho_{\min}, \rho_{\max})$ and $(\theta_{\min}, \theta_{\max})$ are the expected ranges of the parameter values:

a b

FIGURE 10.28
(a) xy-plane. (b) Parameter space.





a b c

FIGURE 10.29 (a) (ρ,θ) parameterization of a 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.

$-90° \le \theta \le 90°$ 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_q results in the solution ρ_p, then we let $A(p,q) = A(p,q) + 1$. At the end of the procedure, a value of K in a cell $A(i,j)$ means that K 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 (see Problem 10.27) 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.

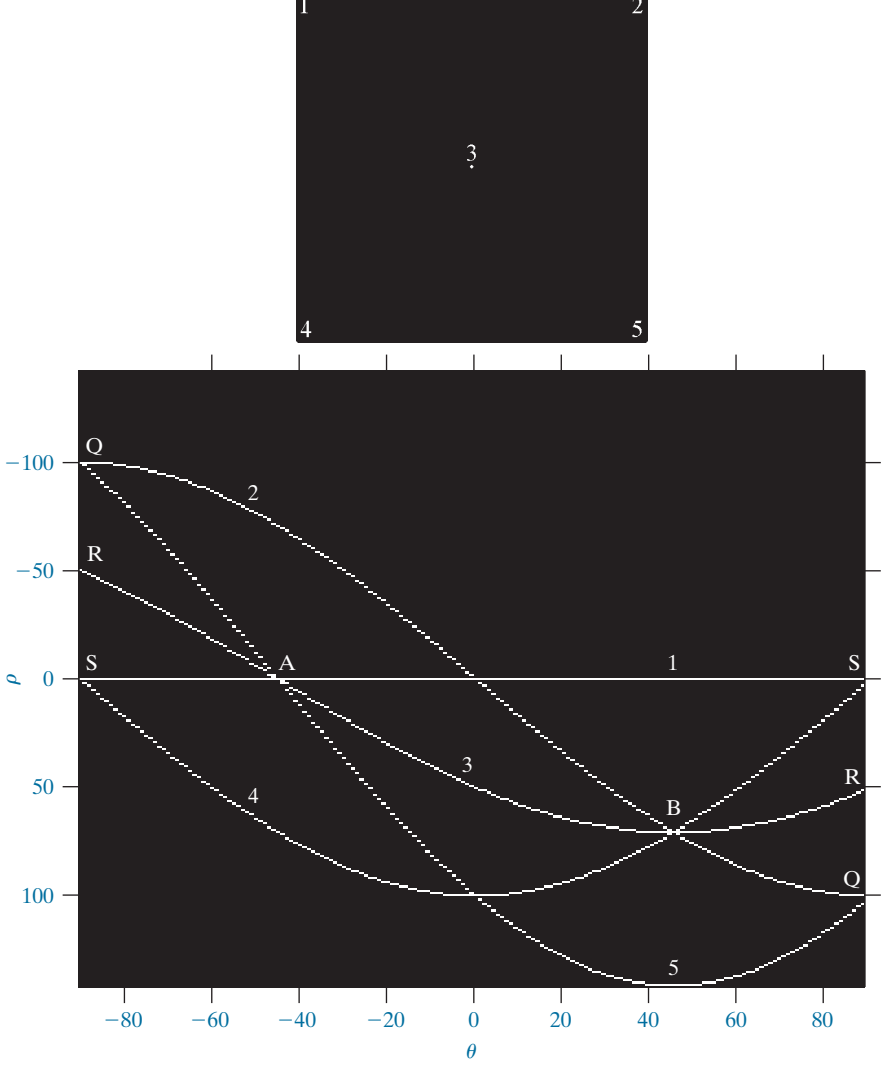
EXAMPLE 10.11: Some basic properties of the Hough transform.

Figure 10.30 illustrates the Hough transform based on Eq. (10-44). Figure 10.30(a) shows an image of size $M \times M$ ($M = 101$) with five labeled white points, and Fig. 10.30(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 ρ values is $\pm\sqrt{2}M$. As Fig. 10.30(b) shows, each curve has a different sinusoidal shape. The horizontal line resulting from the mapping of point 1 is a sinusoid of zero amplitude.

The points labeled A (not to be confused with accumulator values) and B in Fig. 10.30(b) illustrate the colinearity detection property of the Hough transform. For example, point B, marks the intersection of the curves corresponding to points 2, 3, and 4 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. 10.29(a)]. Similarly, the curves intersecting at point B in 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

a
b

FIGURE 10.30
(a) Image of size 101×101 pixels, containing five white points (four in the corners and one in the center). (b) Corresponding parameter space.



value). Finally, the points labeled Q, R, and S in Fig. 10.30(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 ρ and θ 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

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2 \tag{10-45}$$

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 that result 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 value of c_3 that satisfies Eq. (10-45), 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. As noted earlier, generalizations of the Hough transform to detect curves with no simple analytic representations are possible, as is the application of the transform to grayscale images.

Returning to the edge-linking problem, an approach based on the Hough transform is as follows:

1. Obtain a binary edge map using any of the methods 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. 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. The following example illustrates these concepts.

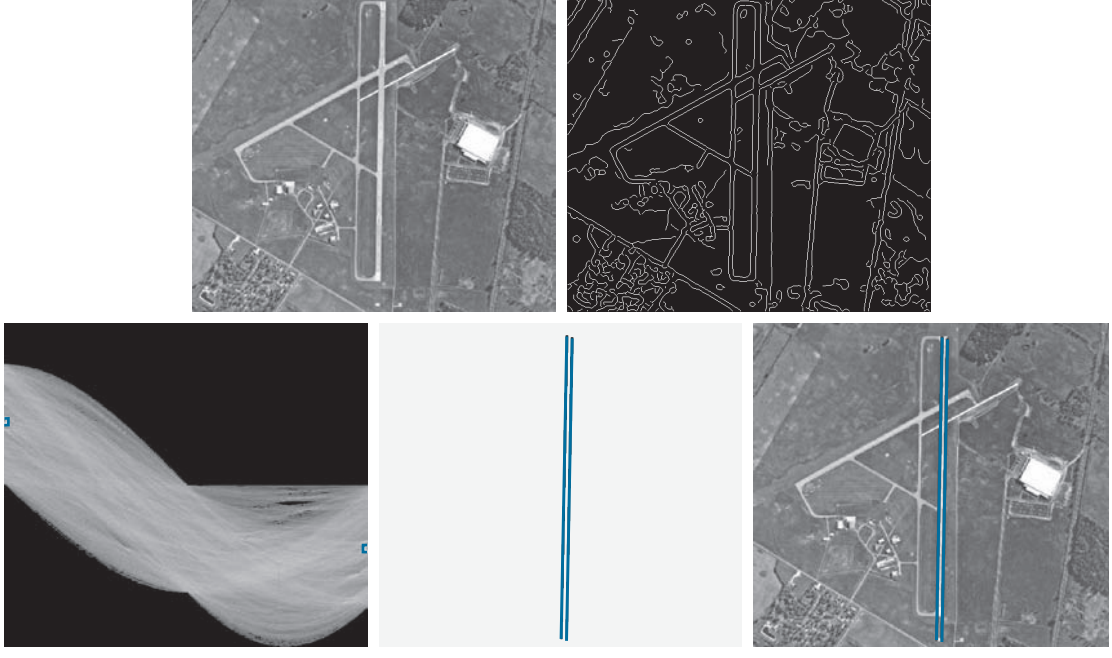
EXAMPLE 10.12: Using the Hough transform for edge linking.

Figure 10.31(a) shows an aerial image of an airport. The objective of this example is to use the Hough transform to extract the two edges defining the principal runway. A solution to such a problem might be of interest, for instance, in applications involving autonomous air navigation.

The first step is to obtain an edge map. Figure 10.31(b) shows the edge map 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 other edge-detection techniques discussed earlier. Figure 10.31(c) shows the Hough parameter space obtained using 1° increments for θ, and one-pixel increments for ρ.

The runway of interest is oriented approximately 1° off the north direction, so we select the cells corresponding to ±90° and containing the highest count because the runways are the longest lines oriented in these directions. The small boxes on the edges of Fig. 10.31(c) highlight these cells. As mentioned earlier in connection with Fig. 10.30(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 10.31(d) shows the lines corresponding to the two accumulator cells just discussed, and Fig. 10.31(e) shows the lines superimposed on the original image. The lines were obtained by joining all gaps not exceeding 20% (approximately 100 pixels) of the image height. These lines clearly correspond to the edges of the runway of interest.

Note that the only information 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



a b
c d e

FIGURE 10.31 (a) A 502×564 aerial image of an airport. (b) Edge map 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.

orientations of runways throughout the world are available in flight charts, and the 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, 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.

FOUNDATION

In the previous section, regions were identified by first finding edge segments, 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. 10.32(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, denoted by $g(x, y)$, is given by

Remember, $f(x, y)$ denotes the intensity of f at coordinates (x, y).

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. (10-46).

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T \\ 0 & \text{if } f(x, y) \leq T \end{cases} \tag{10-46}$$

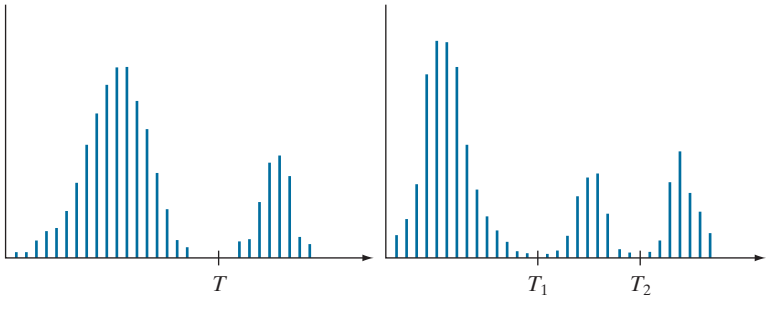
When T is a constant applicable over an entire image, the process given in this equation is referred to as *global thresholding*. When the value of T changes over an image, we use the term *variable thresholding*. The terms *local* or *regional* thresholding are used sometimes to denote variable thresholding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

Figure 10.32(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

$$g(x, y) = \begin{cases} a & \text{if } f(x, y) > T_2 \\ b & \text{if } T_1 < f(x, y) \leq T_2 \\ c & \text{if } f(x, y) \leq T_1 \end{cases} \tag{10-47}$$

a b

FIGURE 10.32 Intensity histograms that can be partitioned (a) by a single threshold, and (b) by dual thresholds.

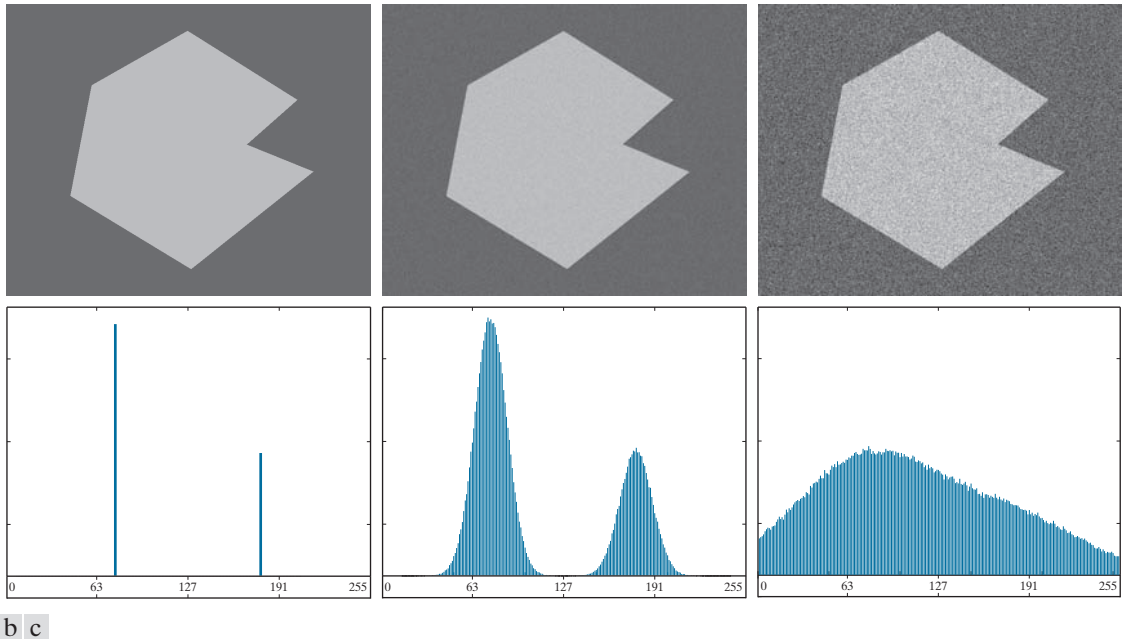


where a, b, and c are any three distinct intensity values. We will discuss dual thresholding later in this section. Segmentation problems requiring more than two thresholds are difficult (or often impossible) to solve, and better results usually are obtained using other methods, such as variable thresholding,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

Based on the preceding discussion, we may infer intuitively that the success of intensity thresholding is related directly 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

The simple synthetic image in Fig. 10.33(a) is free of noise, so its histogram consists of two "spike" modes, as Fig. 10.33(d) shows. Segmenting this image into two regions is a trivial task: we just select a threshold anywhere between the two modes. Figure 10.33(b) shows the original image corrupted by Gaussian noise of zero mean and a standard deviation of 10 intensity levels. The modes are broader now



a b c
d e f

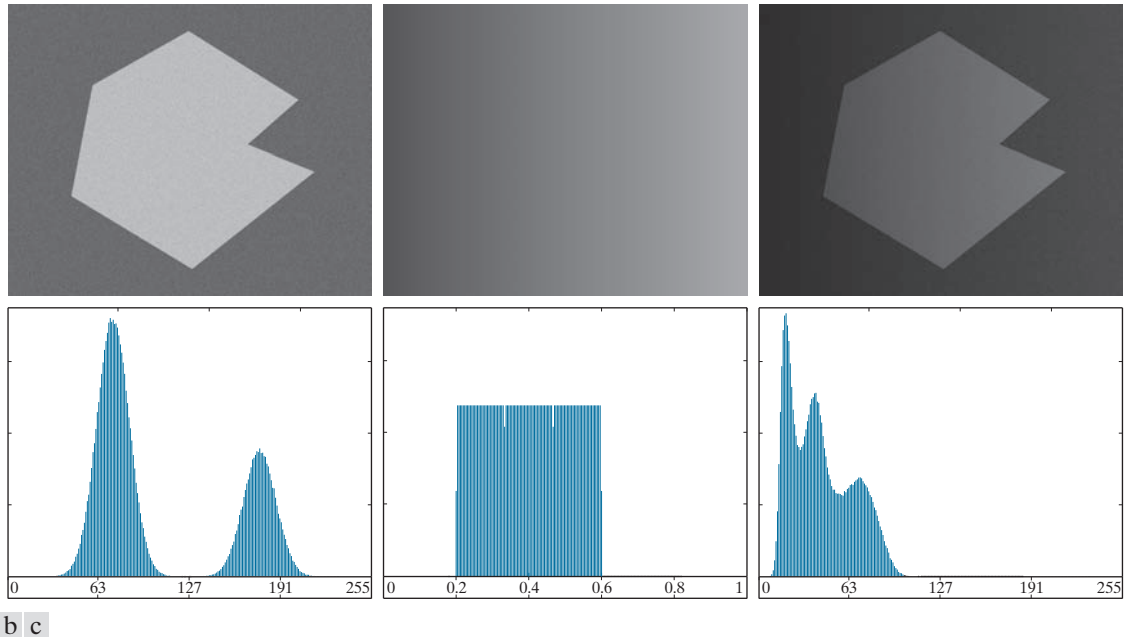
FIGURE 10.33 (a) Noiseless 8-bit image. (b) Image with additive Gaussian noise of mean 0 and standard deviation of 10 intensity levels. (c) Image with additive Gaussian noise of mean 0 and standard deviation of 50 intensity levels. (d) through (f) Corresponding histograms.

[see Fig. 10.33(e)], but their separation is 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 the job. Figure 10.33(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. 10.33(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 later in this section) we have little hope of finding a suitable threshold for segmenting this image.

The Role of Illumination and Reflectance in Image Thresholding

Figure 10.34 illustrates the effect that illumination can have on the histogram of an image. Figure 10.34(a) is the noisy image from Fig. 10.33(b), and Fig. 10.34(d) shows its histogram. As before, this image is easily segmentable with a single threshold. With reference to the image formation model discussed in Section 2.3, suppose that we multiply the image in Fig. 10.34(a) by a nonuniform intensity function, such as the intensity ramp in Fig. 10.37(b), whose histogram is shown in Fig. 10.34(e). Figure 10.34(c) shows the product of these two images, and Fig. 10.34(f) is the resulting histogram. The deep valley between peaks was corrupted to the point where separation of the modes without additional processing (to be discussed later in this section) is no longer possible. Similar results would be obtained if the illumination was

In theory, the histogram of a ramp image is uniform. In practice, the degree of uniformity depends on the size of the image and number of intensity levels.



a b c
d e f

FIGURE 10.34 (a) Noisy image. (b) Intensity ramp in the range [0.2, 0.6]. (c) Product of (a) and (b). (d) through (f) Corresponding histograms.

perfectly uniform, but the reflectance of the image was not, as a result, for example, of natural reflectivity variations in the surface of objects and/or background.

The important point 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 possible 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. The first 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 is to attempt to correct the global shading pattern via processing using, for example, the top-hat transformation introduced in Section 9.8. The third approach is to "work around" nonuniformities using variable thresholding, as discussed later in this section.

BASIC GLOBAL THRESHOLDING

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 the threshold value for each image is required. The following iterative algorithm can be used for this purpose:

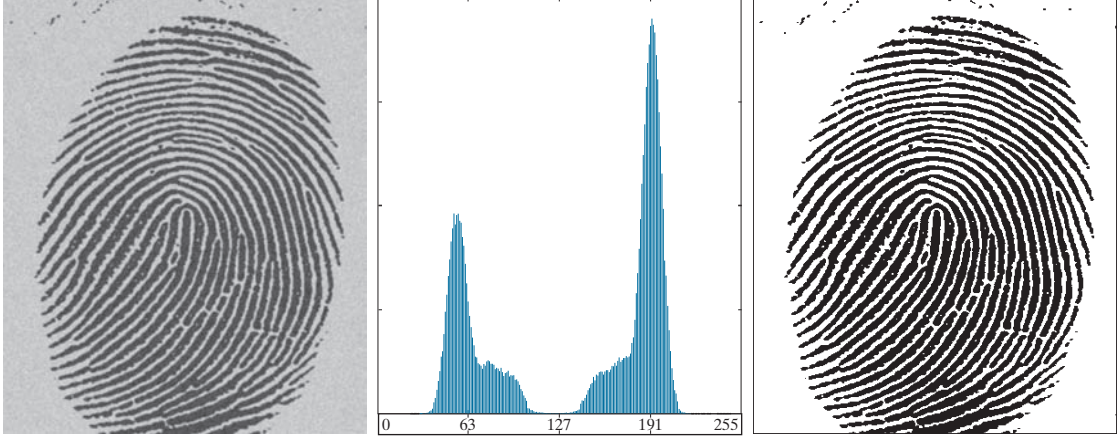
1. Select an initial estimate for the global threshold, T.
2. Segment the image using T in Eq. (10-46). This will produce two groups of pixels: G_1, consisting of pixels with intensity values $> T$; and G_2, consisting of pixels with values $\leq T$.
3. Compute the average (mean) intensity values m_1 and m_2 for the pixels in G_1 and G_2, respectively.
4. Compute a new threshold value midway between m_1 and m_2:

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

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

The algorithm is stated here 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 an equivalent (and more efficient) procedure by expressing all computations in the terms of the image histogram, which has to be computed only once (see Problem 10.29).

The preceding 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 stop iterating when the changes in threshold values is small. The initial threshold must be chosen greater than the minimum and less than the maximum intensity level in the image (the average intensity of the image is a good



a b c

FIGURE 10.35 (a) Noisy fingerprint. (b) Histogram. (c) Segmented result using a global threshold (thin image border added for clarity). (Original image courtesy of the National Institute of Standards and Technology.).

initial choice for T). If this condition is met, the algorithm converges in a finite number of steps, whether or not the modes are separable (see Problem 10.30).

EXAMPLE 10.13: Global thresholding.

Figure 10.35 shows an example of segmentation using the preceding iterative algorithm. Figure 10.35(a) is the original image and Fig. 10.35(b) is the image histogram, showing a distinct valley. Application of the basic global algorithm resulted in the threshold $T = 125.4$ after three iterations, starting with T equal to the average intensity of the image, and using $\Delta T = 0$. Figure 10.35(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 perfect.

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 function* (see Section 12.4). 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 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 real-time applications.

The approach in the following discussion, 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 properly 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 (see Section 3.3).

Let $\{0, 1, 2, \ldots, L-1\}$ denote the set of L distinct integer 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

$$\sum_{i=0}^{L-1} p_i = 1 \qquad p_i \geq 0 \tag{10-48}$$

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

$$P_1(k) = \sum_{i=0}^{k} p_i \tag{10-49}$$

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

$$P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1(k) \tag{10-50}$$

From Eq. (3-25), the *mean intensity* value of the pixels in c_1 is

$$\begin{aligned} m_1(k) &= \sum_{i=0}^{k} iP(i/c_1) = \sum_{i=0}^{k} iP(c_1/i)P(i)/P(c_1) \\ &= \frac{1}{P_1(k)} \sum_{i=0}^{k} ip_i \end{aligned} \tag{10-51}$$

where $P_1(k)$ is given by Eq. (10-49). The term $P(i/c_1)$ in Eq. (10-51) is the probability of intensity value i, given that i comes from class c_1. The rightmost term in the first line of the equation follows from Bayes' formula:

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

The second 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 the ith component of the histogram, p_i. Finally, $P(c_1)$ is the probability of class c_1 which, from Eq. (10-49), is equal to $P_1(k)$.

Similarly, the *mean intensity* value of the pixels assigned to class c_2 is

$$\begin{aligned} m_2(k) &= \sum_{i=k+1}^{L-1} iP(i/c_2) \\ &= \frac{1}{P_2(k)} \sum_{i=k+1}^{L-1} ip_i \end{aligned} \tag{10-52}$$

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

$$m(k) = \sum_{i=0}^{k} ip_i \tag{10-53}$$

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

$$m_G = \sum_{i=0}^{L-1} ip_i \tag{10-54}$$

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

$$P_1 m_1 + P_2 m_2 = m_G \tag{10-55}$$

and

$$P_1 + P_2 = 1 \tag{10-56}$$

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

In order to evaluate the effectiveness of the threshold at level k, we use the normalized, dimensionless measure

$$\eta = \frac{\sigma_B^2}{\sigma_G^2} \tag{10-57}$$

where σ_G^2 is the *global variance* [i.e., the intensity variance of all the pixels in the image, as given in Eq. (3-26)],

$$\sigma_G^2 = \sum_{i=0}^{L-1} (i - m_G)^2 p_i \tag{10-58}$$

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

$$\sigma_B^2 = P_1 (m_1 - m_G)^2 + P_2 (m_2 - m_G)^2 \tag{10-59}$$

The second step in this equation makes sense only if P_1 is greater than 0 and less than 1, which, in view of Eq. (10-56), implies that P_2 must satisfy the same condition.

This expression can also be written as

$$\begin{aligned} \sigma_B^2 &= P_1 P_2 (m_1 - m_2)^2 \\ &= \frac{(m_G P_1 - m)^2}{P_1 (1 - P_1)} \end{aligned} \tag{10-60}$$

The first line of this equation follows from Eqs. (10-55), (10-56), and (10-59). The second line follows from Eqs. (10-50) through (10-54). This form is slightly more efficient computationally because the global mean, m_G, is computed only once, so only two parameters, m_1 and P_1, need to be computed for any value of k.

The first line in Eq. (10-60) indicates that the farther the two means m_1 and m_2 are from each other, the larger σ_B^2 will be, implying 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 earlier. Note that Eq. (10-57) 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 because the separability of a single class from itself is zero.

Reintroducing k, we have the final results:

$$\eta(k) = \frac{\sigma_B^2(k)}{\sigma_G^2} \tag{10-61}$$

and

$$\sigma_B^2(k) = \frac{\left[m_G P_1(k) - m(k)\right]^2}{P_1(k)\left[1 - P_1(k)\right]} \tag{10-62}$$

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

$$\sigma_B^2(k^*) = \max_{0 \le k \le L-1} \sigma_B^2(k) \tag{10-63}$$

To find k^* we simply evaluate this equation for all *integer* values of k (subject to the condition $0 < P_1(k) < 1$) and select the 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 (see Problem 10.36) that a maximum always exists, subject to the condition $0 < P_1(k) < 1$. Evaluating Eqs. (10-62) and (10-63) for all values of k is a relatively inexpensive computational procedure, because the maximum number of integer values that k can have is L, which is only 256 for 8-bit images.

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

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) > k^* \\ 0 & \text{if } f(x,y) \le k^* \end{cases} \tag{10-64}$$

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. (10-62) 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.

In general, the measure in Eq.(10-61) has values in the range

$$0 \leq \eta(k) \leq 1 \tag{10-65}$$

for values of k in the range $[0, L-1]$. When evaluated at the optimum threshold k^*, this measure is a quantitative estimate of the separability of classes, which in turn gives us an idea of the accuracy of thresholding a given image with k^*. The lower bound in Eq. (10-65) is attainable only by images with a single, constant intensity level. The upper bound is attainable only by two-valued images with intensities equal to 0 and $L-1$ (see Problem 10.37).

Otsu's algorithm may be summarized as follows:

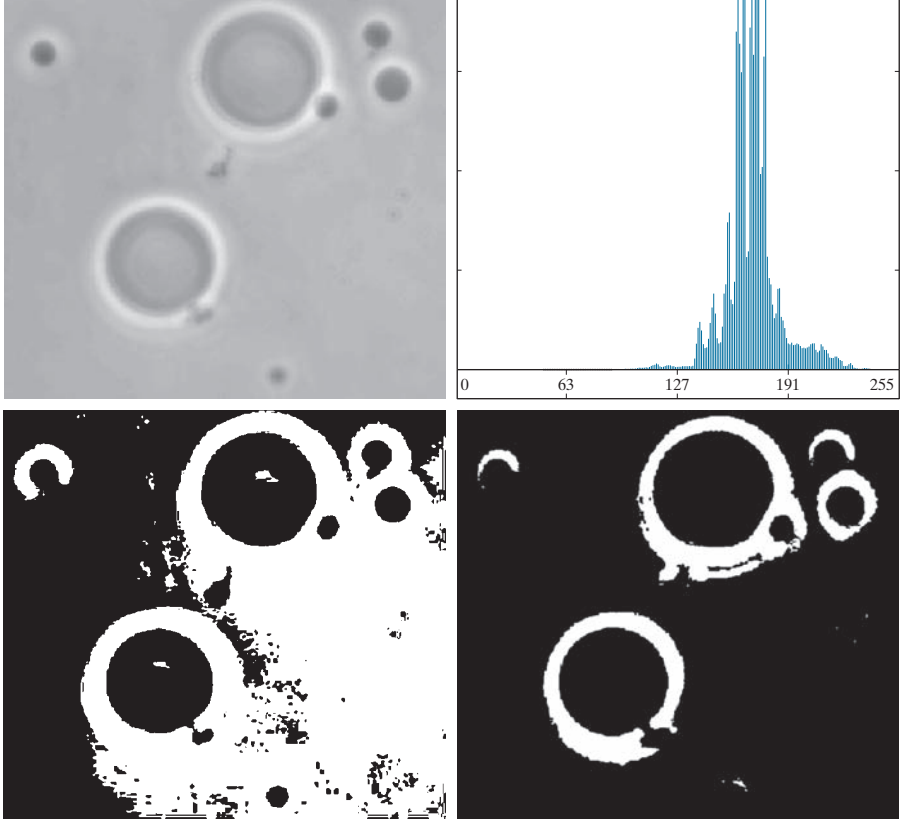
1. Compute the normalized histogram of the input image. Denote the components of the histogram by p_i, $i = 0, 1, 2, \ldots, L-1$.
2. Compute the cumulative sums, $P_1(k)$, for $k = 0, 1, 2, \ldots, L-1$, using Eq. (10-49).
3. Compute the cumulative means, $m(k)$, for $k = 0, 1, 2, \ldots, L-1$, using Eq. (10-53).
4. Compute the global mean, m_G, using Eq. (10-54).
5. Compute the between-class variance term, $\sigma_B^2(k)$, for $k = 0, 1, 2, \ldots, L-1$, using Eq. (10-62).
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. Compute the global variance, σ_G^2, using Eq. (10-58), and then obtain the separability measure, η^*, by evaluating Eq. (10-61) with $k = k^*$.

The following example illustrates the use of this algorithm.

EXAMPLE 10.14: Optimum global thresholding using Otsu's method.

Figure 10.36(a) shows an optical microscope image of polymersome cells. These are cells artificially engineered using polymers. They are invisible to the human immune system and can be used, for example, to deliver medication to targeted regions of the body. Figure 10.36(b) shows the image histogram. The objective of this example is to segment the molecules from the background. Figure 10.36(c) is the result of using the basic global thresholding algorithm discussed earlier. 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 10.36(d) shows the result obtained using Otsu's method. This result obviously is superior to Fig. 10.36(c). The threshold value computed by the basic algorithm was 169, while the threshold computed by Otsu's method was 182, which is closer to the lighter areas in the image defining the cells. The separability measure η^* was 0.467.

As a point of interest, applying Otsu's method to the fingerprint image in Example 10.13 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 because of the relatively large separation between modes and the deep valley between them.



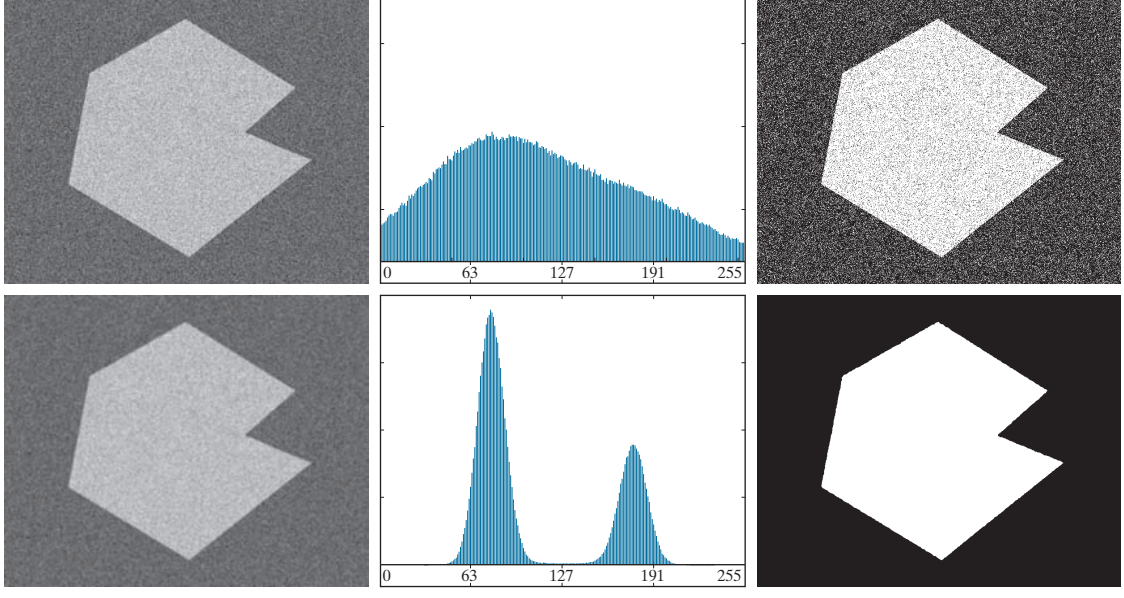
a b
c d

FIGURE 10.36
(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.
(d) Result using Otsu's method.
(Original image courtesy of Professor Daniel A. Hammer, the University of Pennsylvania.)

USING IMAGE SMOOTHING TO IMPROVE GLOBAL THRESHOLDING

As illustrated in Fig. 10.33, noise can turn a simple thresholding problem into an unsolvable one. When noise cannot be reduced at the source, and thresholding is the preferred segmentation method, a technique that often enhances performance is to smooth the image prior to thresholding. We illustrate this approach with an example.

Figure 10.37(a) is the image from Fig. 10.33(c), Fig. 10.37(b) shows its histogram, and Fig. 10.37(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 thresholding error, so the segmentation was highly unsuccessful. Figure 10.37(d) shows the result of smoothing the noisy image with an averaging kernel of size 5×5 (the image is of size 651×814 pixels), and Fig. 10.37(e) is its histogram. The improvement in the shape of the histogram as a result of smoothing is evident, and we would expect thresholding of the smoothed image to be nearly perfect. Figure 10.37(f) shows this to be 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.



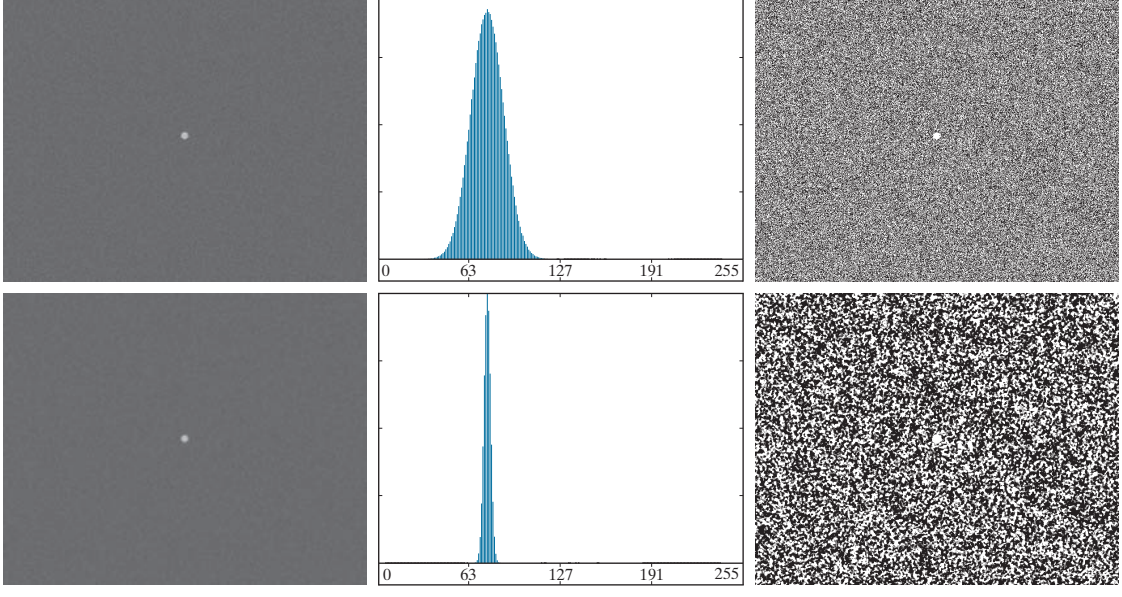
a b c
d e f

FIGURE 10.37 (a) Noisy image from Fig. 10.33(c) and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5×5 averaging kernel and (e) its histogram. (f) Result of thresholding using Otsu's method.

Next, we investigate the effect of severely reducing the size of the foreground region with respect to the background. Figure 10.38(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. 10.38(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. 10.38(c). Figure 10.38(d) shows the image smoothed with an averaging kernel of size 5×5, and Fig. 10.38(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. 10.38(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 situations such as this, the approach discussed in the following section is more likely to succeed.

USING EDGES TO IMPROVE GLOBAL THRESHOLDING

Based on the discussion thus far, we conclude that the chances of finding 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



a b c
d e f

FIGURE 10.38 (a) Noisy image and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5×5 averaging kernel and (e) its histogram. (f) Result of thresholding using Otsu's method. Thresholding failed in both cases to extract the object of interest. (See Fig. 10.39 for a better solution.)

objects and the background. An immediate and obvious improvement is that histograms should be less dependent on the relative sizes of objects and 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 Fig. 10.38 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.

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 aims to do. However, 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 created using an isotropic edge detector.

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

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.

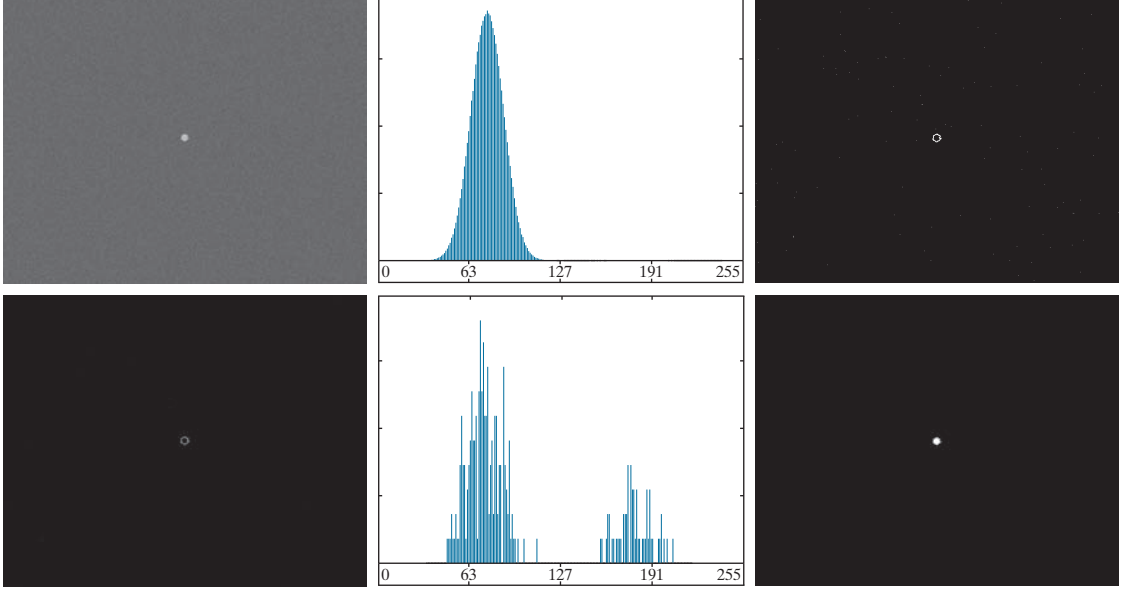
1. Compute an edge image as either the magnitude of the gradient, or absolute value of the Laplacian, of $f(x, y)$ using any of the methods in Section 10.2.
2. Specify a threshold value, T.
3. Threshold the image from Step 1 using T from Step 2 to produce a binary image, $g_T(x, y)$. This image is used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

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.

If T is set to any value less than the minimum value of the edge image then, according to Eq. (10-46), $g_T(x, y)$ will consist of all 1's, 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 using the histogram of the original image. It is customary to specify the value of T to correspond to a percentile, which typically is set high (e.g., in the high 90's) 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.

EXAMPLE 10.15: Using edge information based on the gradient to improve global thresholding.

Figures 10.39(a) and (b) show the image and histogram from Fig. 10.38. 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 10.39(c) is the mask image, $g_T(x, y)$, formed as gradient magnitude image thresholded at the 99.7 percentile. Figure 10.39(d) is the image formed by multiplying the mask by the input image. Figure 10.39(e) is the histogram of the nonzero elements in Fig. 10.39(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. 10.39(e) indicates that thresholding of the small object from the background is indeed possible. The result in Fig. 10.39(f) shows that this is the case. This image was generated using Otsu's method [to obtain a threshold based on the histogram in Fig. 10.42(e)], and then applying the Otsu threshold globally to the noisy image in Fig. 10.39(a). The result is nearly perfect.



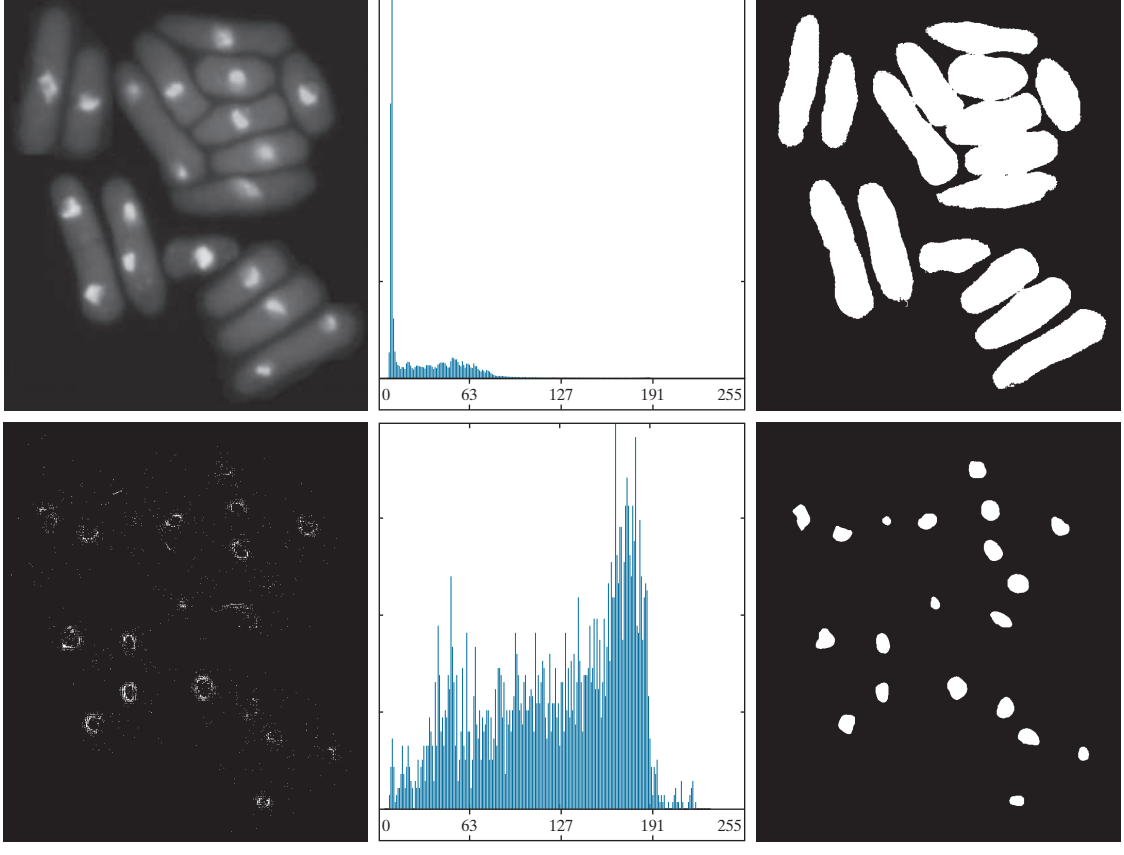
a b c
d e f

FIGURE 10.39 (a) Noisy image from Fig. 10.38(a) and (b) its histogram. (c) Mask image formed as the 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.

EXAMPLE 10.16: Using edge information based on the Laplacian to improve global thresholding.

In this example, we consider a more complex thresholding problem. Figure 10.40(a) shows an 8-bit image of yeast cells for which we want to use global thresholding to obtain the regions corresponding to the bright spots. As a starting point, Fig. 10.40(b) shows the image histogram, and Fig. 10.40(c) is the result obtained using Otsu's method directly on the image, based on the histogram shown. We see that Otsu's method failed to achieve the original objective of detecting the bright spots. Although the method was able to isolate some of the cell regions themselves, several of the segmented regions on the right were actually joined. The threshold computed by the Otsu method was 42, and the separability measure was 0.636.

Figure 10.40(d) shows the mask image $g_T(x, y)$ obtained by computing the absolute value of the Laplacian image, 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 results in a sparse set of pixels, as Fig. 10.40(d) shows. Note in this image how the points cluster near the edges of the bright spots, as expected from the preceding discussion. Figure 10.40(e) is the histogram of the nonzero pixels in the product of (a) and (d). Finally, Fig. 10.40(f) shows the result of globally segmenting the original image using Otsu's method based on the histogram in Fig. 10.40(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.



a b c
d e f

FIGURE 10.40 (a) Image of yeast cells. (b) Histogram of (a). (c) Segmentation of (a) with Otsu's method using the histogram in (b). (d) Mask image formed by thresholding the absolute Laplacian image. (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.)

By varying the percentile at which the threshold is set, we can even improve the segmentation of the complete cell regions. For example, Fig. 10.41 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. 10.40(c) obtained using Otsu's method with the histogram of the original image.

MULTIPLE THRESHOLDS

Thus far, we have focused attention on image segmentation using a single global threshold. Otsu's method can be extended to an arbitrary number of thresholds

FIGURE 10.41 Image in Fig. 10.40(a) segmented using the same procedure as explained in Figs. 10.40(d) through (f), but using a lower value to threshold the absolute Laplacian image.

In applications involving more than one variable (for example the RGB components of a color image), thresholding can be implemented using a distance measure, such as the *Euclidean distance*, or *Mahalanobis distance* discussed in Section 6.7 (see Eqs. (6-48), (6-49), and Example 6.15).

because the separability measure 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

$$\sigma_B^2 = \sum_{k=1}^{K} P_k \left(m_k - m_G\right)^2 \tag{10-66}$$

where

$$P_k = \sum_{i \in c_k} p_i \tag{10-67}$$

and

$$m_k = \frac{1}{P_k} \sum_{i \in c_k} i p_i \tag{10-68}$$

As before, m_G is the global mean given in Eq. (10-54). 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. (10-66):

$$\sigma_B^2\left(k_1^*, k_2^*, \ldots, k_{K-1}^*\right) = \max_{0<k_1<k_2<\ldots k_K<L-1} \sigma_B^2\left(k_1, k_2, \ldots k_{K-1}\right) \tag{10-69}$$

Although this result is applicable to an arbitrary number of classes, 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 you will learn shortly in the discussion on multivariable thresholding.

Recall from the discussion of the Canny edge detector that thresholding with two thresholds 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:

$$\sigma_B^2 = P_1\left(m_1 - m_G\right)^2 + P_2\left(m_2 - m_G\right)^2 + P_3\left(m_3 - m_G\right)^2 \tag{10-70}$$

where

$$\begin{aligned} P_1 &= \sum_{i=0}^{k_1} p_i \\ P_2 &= \sum_{i=k_1+1}^{k_2} p_i \\ P_3 &= \sum_{i=k_2+1}^{L-1} p_i \end{aligned} \tag{10-71}$$

and

$$\begin{aligned} m_1 &= \frac{1}{P_1}\sum_{i=0}^{k_1} i p_i \\ m_2 &= \frac{1}{P_2}\sum_{i=k_1+1}^{k_2} i p_i \\ m_3 &= \frac{1}{P_3}\sum_{i=k_2+1}^{L-1} i p_i \end{aligned} \tag{10-72}$$

As in Eqs. (10-55) and (10-56), the following relationships hold:

$$P_1 m_1 + P_2 m_2 + P_3 m_3 = m_G \tag{10-73}$$

and

$$P_1 + P_2 + P_3 = 1 \tag{10-74}$$

We see from Eqs. (10-71) and (10-72) that P and m, 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 $\sigma_B^2(k_1, k_2)$. That is, as indicated in Eq. (10-69), we find the optimum thresholds by finding

$$\sigma_B^2\left(k_1^*, k_2^*\right) = \max_{0<k_1<k_2<L-1} \sigma_B^2\left(k_1, k_2\right) \tag{10-75}$$

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 integer intensity values). 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 procedure is a 2-D array, $\sigma_B^2\left(k_1, k_2\right)$, 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 in the array 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

$$g(x,y) = \begin{cases} a & \text{if } f(x,y) \le k_1^* \\ b & \text{if } k_1^* < f(x,y) \le k_2^* \\ c & \text{if } f(x,y) > k_2^* \end{cases} \tag{10-76}$$

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

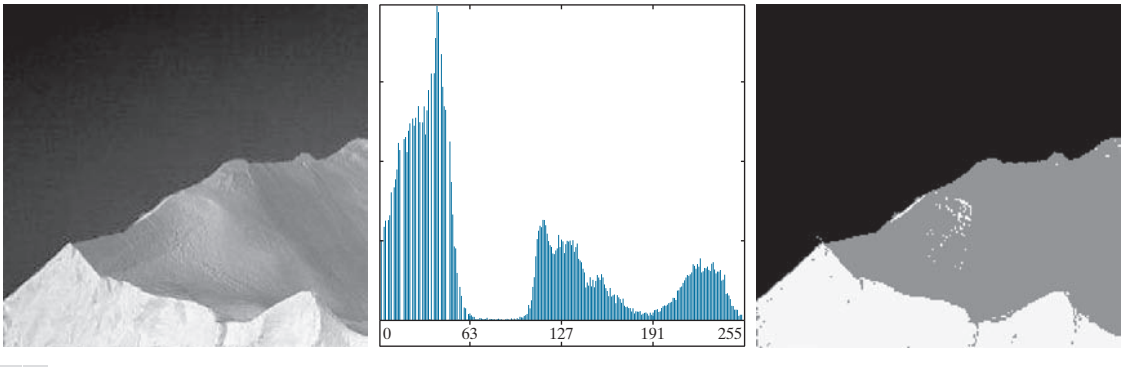
Finally, the separability measure defined earlier for one threshold extends directly to multiple thresholds:

$$\eta\left(k_1^*, k_2^*\right) = \frac{\sigma_B^2\left(k_1^*, k_2^*\right)}{\sigma_G^2} \tag{10-77}$$

where σ_G^2 is the total image variance from Eq. (10-58).

EXAMPLE 10.17: Multiple global thresholding.

Figure 10.42(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. 10.42(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 10.42(c) is the segmentation that resulted using these two thresholds in Eq. (10-76). 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. But we can do even better using superpixels, as you will see in Section 10.5.



a b c

FIGURE 10.42 (a) Image of an iceberg. (b) Histogram. (c) Image segmented into three regions using dual Otsu thresholds. (Original image courtesy of NOAA.)

VARIABLE THRESHOLDING

As discussed earlier in this section, factors such as noise and nonuniform illumination play a major role in the performance of a thresholding algorithm. We showed that image smoothing and the use of edge information can help significantly. However, sometimes this type of preprocessing is either impractical or ineffective in improving the situation, to the point where the problem cannot be solved by any of the thresholding methods discussed thus far. In such situations, the next level of thresholding complexity involves variable thresholding, as we will illustrate in the following discussion.

Variable Thresholding Based on Local Image Properties

A basic approach to variable thresholding is to compute a threshold at every point, (x, y), in the image based on one or more specified properties 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 approach using the mean and standard deviation of the pixel values in a neighborhood of every point in an image. These two quantities are useful for determining local thresholds because, as you know from Chapter 3, they are descriptors of average intensity and contrast. Let m_{xy} and σ_{xy} denote the mean and standard deviation of the set of pixel values in a neighborhood, S_{xy}, centered at coordinates (x, y) in an image (see Section 3.3 regarding computation of the local mean and standard deviation). The following are common forms of variable thresholds based on the local image properties:

We simplified the notation slightly from the form we used in Eqs. (3-27) and (3-28) by letting xy imply a neighborhood S, centered at coordinates (x, y).

$$T_{xy} = a\sigma_{xy} + bm_{xy} \tag{10-78}$$

where a and b are nonnegative constants, and

$$T_{xy} = a\sigma_{xy} + bm_G \tag{10-79}$$

Note that T_{xy} is a threshold *array* of the same size as the image from which it was obtained. The threshold at a location (x, y) in the array is used to segment the value of an image at that location.

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

$$g(x, y) = \begin{cases} 1 & \text{if } f(x, y) > T_{xy} \\ 0 & \text{if } f(x, y) \leq T_{xy} \end{cases} \tag{10-80}$$

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 variable thresholding by using predicates based on the parameters computed in the neighborhood of a point (x, y):

$$g(x, y) = \begin{cases} 1 & \text{if } Q(\text{local parameters}) \text{ is TRUE} \\ 0 & \text{if } Q(\text{local parameters}) \text{ is FALSE} \end{cases} \tag{10-81}$$

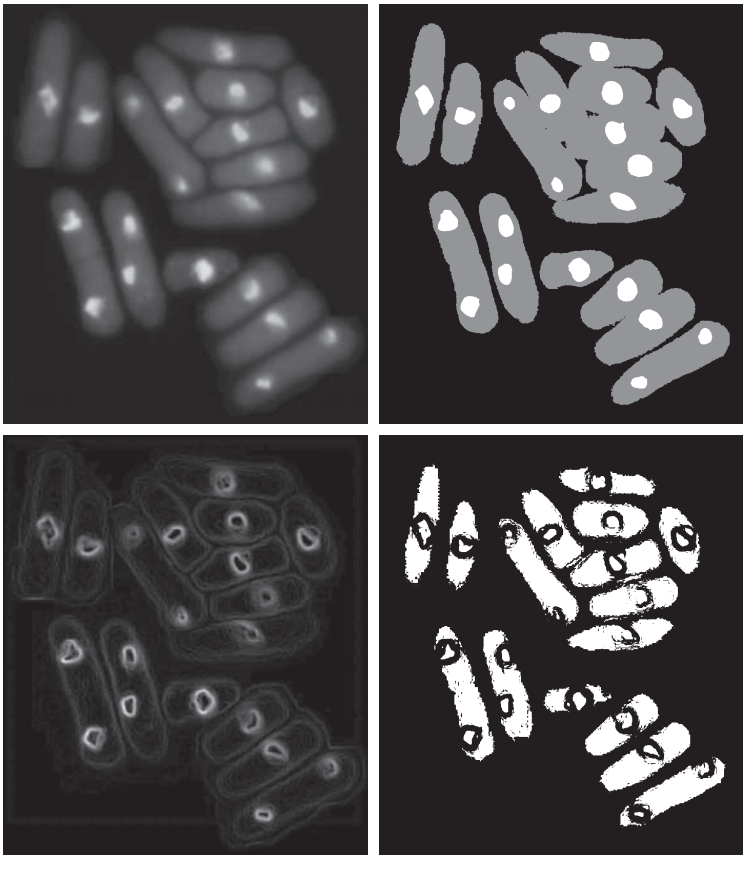
where Q is a *predicate* based on parameters computed using the pixels in neighborhood S_{xy}. For example, consider the following predicate, $Q(\sigma_{xy}, m_{xy})$, based on the local mean and standard deviation:

$$Q(\sigma_{xy}, m_{xy}) = \begin{cases} \text{TRUE} & \text{if } f(x,y) > a\sigma_{xy} \text{ AND } f(x,y) > bm_{xy} \\ \text{FALSE} & \text{otherwise} \end{cases} \quad (10\text{-}82)$$

Note that Eq. (10-80) is a special case of Eq. (10-81), 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 10.18: Variable thresholding based on local image properties.

Figure 10.43(a) shows the yeast image from Example 10.16. This image has three predominant intensity levels, so it is reasonable to assume that perhaps dual thresholding could be a good segmentation approach. Figure 10.43(b) is the result of using the dual thresholding method summarized in Eq. (10-76). 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 (i.e., separated) properly. To illustrate the use



a b
c d

FIGURE 10.43
(a) Image from Fig. 10.40. (b) Image segmented using the dual thresholding approach given by Eq. (10-76). (c) Image of local standard deviations. (d) Result obtained using local thresholding.

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 10.43(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. (10-82), 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 to complete 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. (10-82). As Fig. 10.43(d) shows, the segmentation was quite successful. Note in particular that all the outer regions were segmented properly, and that most of the inner, brighter regions were isolated correctly.

Variable Thresholding Based on Moving Averages

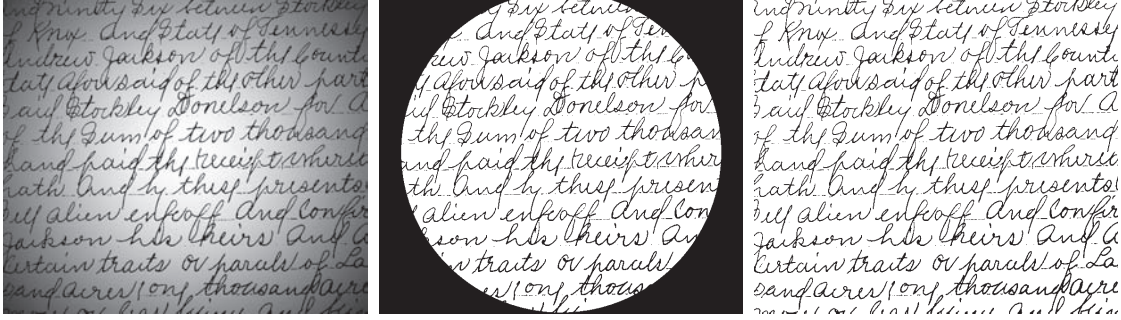
A special case of the variable thresholding method discussed in the previous section is based on computing a moving average along scan lines of an image. This implementation is useful in applications such as document processing, where speed is a fundamental requirement. The scanning typically is carried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

$$\begin{aligned} m(k+1) &= \frac{1}{n}\sum_{i=k+2-n}^{k+1} z_i \qquad \text{for } k \geq n-1 \\ &= m(k) + \frac{1}{n}\left(z_{k+1} - z_{k-n}\right) \qquad \text{for } k \geq n+1 \end{aligned} \tag{10-83}$$

where n is the number of points used in computing the average, and $m(1) = z_1$. The conditions imposed on k are so that all subscripts on z_k are positive. All this means is that n points must be available for computing the average. When k is less than the limits shown (this happens near the image borders) the averages are formed with the available image points. Because a moving average is computed for every point in the image, segmentation is implemented using Eq. (10-80) with $T_{xy} = cm_{xy}$, where c is positive scalar, and m_{xy} is the moving average from Eq. (10-83) at point (x, y) in the input image.

EXAMPLE 10.19: Document thresholding using moving averages.

Figure 10.44(a) shows an image of handwritten text shaded by a spot intensity pattern. This form of intensity shading is typical of images obtained using spot illumination (such as a photographic flash). Figure 10.44(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 because the method generally performs poorly when the areas of interest are embedded in a nonuniform illumination field. Figure 10.44(c) shows successful segmentation with local thresholding using moving averages. For images of written material, a rule of thumb is to let n equal five times the average stroke width. In this case, the average width was 4 pixels, so we let $n = 20$ in Eq. (10-83) and used $c = 0.5$.



a b c

FIGURE 10.44 (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 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. 10.45(a), which is corrupted by a sinusoidal intensity variation typical of the variations that may occur when the power supply in a document scanner is not properly grounded. As Figs. 10.45(b) and (c) show, the segmentation results are comparable to those in Fig. 10.44.

Note that successful segmentation results were obtained in both cases using the same values for n and c, 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.

10.4 SEGMENTATION BY REGION GROWING AND BY REGION SPLITTING AND MERGING

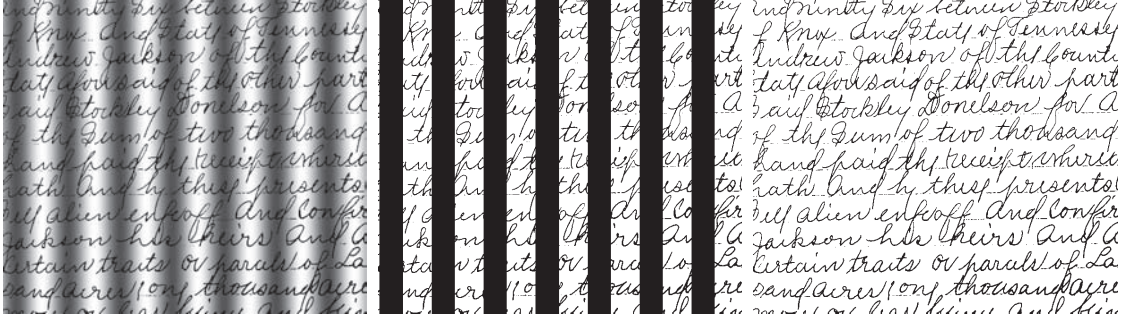
You should review the terminology introduced in Section 10.1 before proceeding.

As we 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 and in Sections 10.5 and 10.6, we discuss segmentation techniques that find the regions directly. In Section 10.7, we will discuss a method that finds the regions and their boundaries simultaneously.

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 ranges of intensity or color).

Selecting a set of one or more starting points can often be based on the nature of the problem, as we show later in Example 10.20. When a priori information is not



a b c

FIGURE 10.45 (a) Text image corrupted by sinusoidal shading. (b) Result of global thresholding using Otsu's method. (c) Result of local thresholding using moving averages..

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 will 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 that have three distinct intensity values. Grouping pixels with the same intensity value 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 can 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; $S(x, y)$ denote a *seed* array containing 1's at the locations of seed points and 0's 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 and 9.5 regarding connected components, and Section 9.2 regarding erosion.

1. Find all connected components in $S(x, y)$ and reduce 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 each point (x, y), $f_Q(x, y) = 1$ if the input image satisfies a given predicate, Q, at those coordinates, and $f_Q(x, y) = 0$ otherwise.
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.,integers or letters). This is the segmented image obtained by region growing.

The following example illustrates the mechanics of this algorithm.

EXAMPLE 10.20: Segmentation by region growing.

Figure 10.46(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.

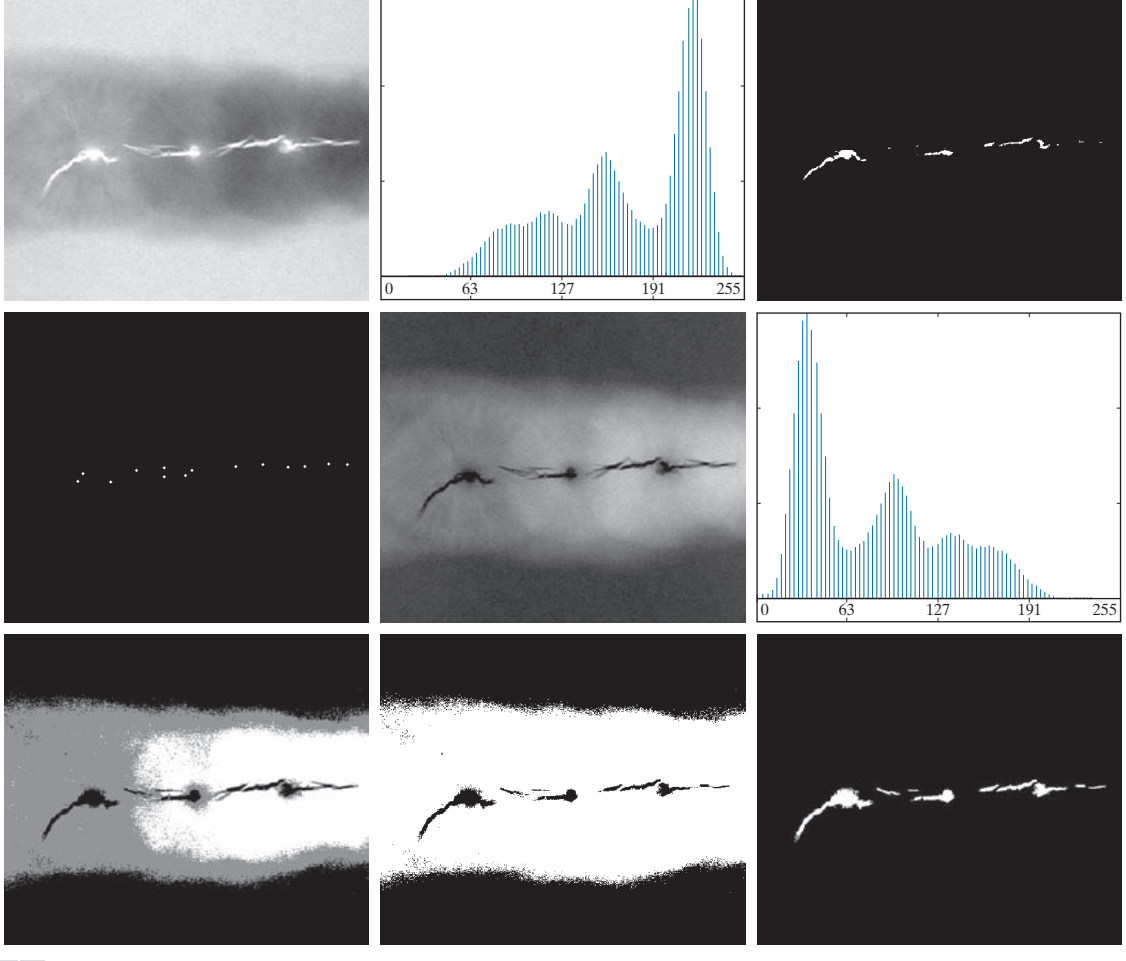
The first thing we do is determine the seed points. From the physics of the problem, we know that cracks and porosities will attenuate X-rays considerably less than solid welds, so we expect the regions containing these types of defects to be significantly brighter than other parts of the X-ray image. We can extract the seed points by thresholding the original image, using a threshold set at a high percentile. Figure 10.46(b) shows the histogram of the image, and Fig. 10.46(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 regarding percentiles). Figure 10.46(d) shows the result of morphologically eroding each connected component in Fig. 10.46(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 absolute intensity differences as a measure of similarity, our predicate applied at each location (x, y) is

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

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 all seed values are 255 because the image was thresholded with a threshold of 254. Figure 10.46(e) shows the difference between the seed value (255) and Fig. 10.46(a). The image in Fig. 10.46(e) contains all the differences needed to compute the predicate at each location (x, y). Figure 10.46(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. 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



a b c
d e f
g h i

Figure **10.46** (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 the seed value (255) and (a). (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.)

Fig. 10.46(g) shows that segmenting the defects cannot be accomplished using dual thresholds, despite the fact that the thresholds are in the deep valleys of the histogram.)

Figure 10.46(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. 10.46(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 because all those regions have the same physical meaning in this application—they all represent porosities.

REGION SPLITTING AND MERGING

The procedure just discussed grows regions from seed points. An alternative is to subdivide an image initially into a set of 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 region splitting and merging are discussed next.

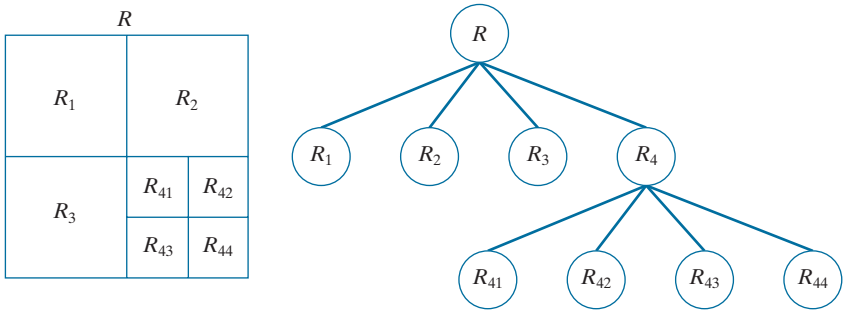
Let R represent the entire image region and select a predicate Q. One approach for segmenting R is to subdivide it successively into smaller and smaller quadrant regions so that, for any region R_i, $Q(R_i) = \text{TRUE}$. We start with the entire region, R. If $Q(R) = \text{FALSE}$, we divide the image into quadrants. If Q is FALSE for any quadrant, we subdivide that quadrant into sub-quadrants, and so on. This 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. 10.47 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 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 $Q(R_j \cup R_k) = \text{TRUE}$.



a b

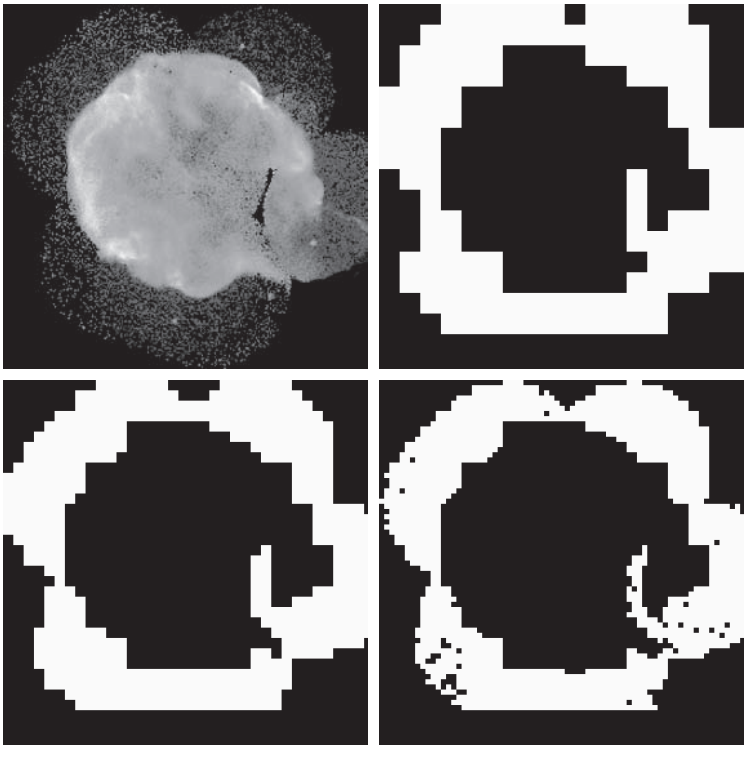
FIGURE 10.47
(a) Partitioned image.
(b) Corresponding quadtree.
R represents the entire image region.

3. Stop when no further merging is possible.

Numerous variations of this basic theme are possible. For example, a significant simplification results if in Step 2 we allow merging of any two adjacent regions R_j and R_k 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.

EXAMPLE 10.21: Segmentation by region splitting and merging.

Figure 10.48(a) shows a 566×566 X-ray image of the Cygnus Loop supernova. The objective of this example is to segment (extract from the image) the "ring" of less dense matter surrounding the dense inner region. 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 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 lighter central region. Thus, we should be able to segment the region of interest using the following predicate:



a b
c d

FIGURE 10.48
(a) Image of the Cygnus Loop supernova, taken in the X-ray band by NASA's Hubble Telescope. (b) through (d) Results of limiting the smallest allowed quadregion to be of sizes of 32×32, 16×16, and 8×8 pixels, respectively. (Original image courtesy of NASA.)

$$Q(R) = \begin{cases} \text{TRUE} & \text{if } \sigma_R > a \text{ AND } 0 < m_R < b \\ \text{FALSE} & \text{otherwise} \end{cases}$$

where σ_R and m_R are the standard deviation and mean of the region being processed, and a and b are nonnegative 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 10.48(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 that 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 small black squares in Fig. 10.48(d) are quadregions of size 8×8 whose pixels did not satisfy the predicate. Using smaller quadregions would result in increasing numbers of such black regions. Using regions larger than the one illustrated here would result in a more "block-like" segmentation. Note that in all cases the segmented region (white pixels) was a connected region that completely separates 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, a dense region, and a sparse region. Using any of the white regions in Fig. 10.48 as a mask would make it a relatively simple task to extract these regions from the original image (see Problem 10.43). As in Example 10.20, 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 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 REGION SEGMENTATION USING CLUSTERING AND SUPERPIXELS

In this section, we discuss two related approaches to region segmentation. The first is a classical approach based on seeking clusters in data, related to such variables as intensity and color. The second approach is significantly more modern, and is based on using clustering to extract "superpixels" from an image.

A more general form of clustering is *unsupervised clustering*, in which a clustering algorithm attempts to find a meaningful set of clusters in a given set of samples. We do not address this topic, as our focus in this brief introduction is only to illustrate how *supervised clustering* is used for image segmentation.

REGION SEGMENTATION USING K-MEANS CLUSTERING

The basic idea behind the clustering approach used in this chapter is to partition a set, Q, of observations into a specified number, k, of clusters. In k-means clustering, each observation is assigned to the cluster with the nearest mean (hence the name of the method), and each mean is called the *prototype* of its cluster. A *k-means algorithm* is an iterative procedure that successively refines the means until convergence is achieved.

Let $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_Q\}$ be set of vector observations (samples). These vectors have the form

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \tag{10-84}$$

In image segmentation, each component of a vector $\mathbf{z}$ represents a numerical pixel attribute. For example, if segmentation is based on just grayscale intensity, then $\mathbf{z} = z$ is a scalar representing the intensity of a pixel. If we are segmenting RGB color images, $\mathbf{z}$ typically is a 3-D vector, each component of which is the intensity of a pixel in one of the three primary color images, as we discussed in Chapter 6. The objective of k-means clustering is to partition the set Q of observations into k ($k \leq Q$) disjoint cluster sets $C = \{C_1, C_2, \ldots, C_k\}$, so that the following criterion of optimality is satisfied:[†]

$$\arg\min_{C} \left(\sum_{i=1}^{k} \sum_{\mathbf{z} \in C_i} \|\mathbf{z} - \mathbf{m}_i\|^2 \right) \tag{10-85}$$

where $\mathbf{m}_i$ is the *mean vector* (or *centroid*) of the samples in set C_i and $\|\arg\|$ is the vector norm of the argument. Typically, the Euclidean norm is used, so the term $\|\mathbf{z} - \mathbf{m}_i\|$ is the familiar *Euclidean distance* from a sample in C_i to mean $\mathbf{m}_i$. In words, this equation says that we are interested in finding the sets $C = \{C_1, C_2, \ldots, C_k\}$ such that the *sum of the distances* from each point in a set to the mean of that set is minimum.

Unfortunately, finding this minimum is an NP-hard problem for which no practical solution is known. As a result, a number of heuristic methods that attempt to find approximations to the minimum have been proposed over the years. In this section, we discuss what is generally considered to be the "standard" k-means algorithm, which is based on the Euclidean distance (see Section 2.6). Given a set $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_Q\}$ of vector observation and a specified value of k, the algorithm is as follows:

These initial means are the initial cluster centers. They are also called *seeds*.

1. ***Initialize the algorithm:*** Specify an initial set of means, $\mathbf{m}_i(1)$, $i = 1, 2, \ldots, k$.
2. ***Assign samples to clusters:*** Assign each sample to the cluster set whose mean is the closest (ties are resolved arbitrarily, but samples are assigned to only *one* cluster):

   $$\mathbf{z}_q \rightarrow C_i \quad \text{if} \quad \|\mathbf{z}_q - \mathbf{m}_i\|^2 < \|\mathbf{z}_q - \mathbf{m}_j\|^2 \quad j = 1, 2, \ldots, k \; (j \neq i); \quad q = 1, 2, \ldots, Q$$

3. ***Update the cluster centers (means)***:

   $$\mathbf{m}_i = \frac{1}{|C_i|} \sum_{\mathbf{z} \in C_i} \mathbf{z} \qquad i = 1, 2, \ldots, k$$

   where $|C_i|$ is the number of samples in cluster set C_i.
4. ***Test for completion:*** Compute the Euclidean norms of the differences between the mean vectors in the current and previous steps. Compute the residual error, E, as the sum of the k norms. Stop if $E \leq T$, where T a specified, nonnegative threshold. Else, go back to Step 2.

[†] Remember, $\min_x(h(x))$ is the minimum of h with respected to x, whereas $\arg\min_x(h(x))$ is the value (or values) of x at which h is minimum.

a b

FIGURE 10.49
(a) Image of size 688×688 pixels. (b) Image segmented using the k-means algorithm with $k = 3$.

When $T = 0$, this algorithm is known to converge in a finite number of iterations to a local minimum. It is not guaranteed to yield the global minimum required to minimize Eq. (10-85). The result at convergence does depend on the initial values chosen for $\mathbf{m}_i$. An approach used frequently in data analysis is to specify the initial means as k randomly chosen samples from the given sample set, and to run the algorithm several times, with a new random set of initial samples each time. This is to test the "stability" of the solution. In image segmentation, the important issue is the value selected for k because this determines the number of segmented regions; thus, multiple passes are rarely used.

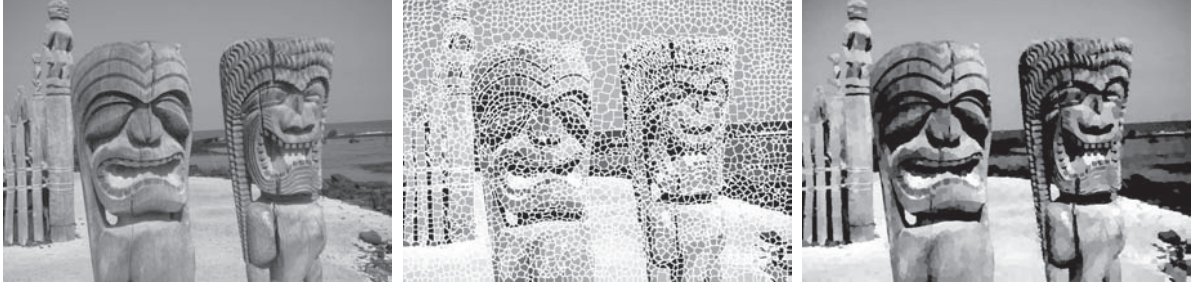
EXAMPLE 10.22: Using k-means clustering for segmentation.

Figure 10.49(a) shows an image of size 688×688 pixels, and Fig. 10.49(b) is the segmentation obtained using the k-means algorithm with $k = 3$. As you can see, the algorithm was able to extract all the meaningful regions of this image with high accuracy. For example, compare the quality of the characters in both images. It is important to realize that the entire segmentation was done by clustering of a single variable (intensity). Because k-means works with vector observations in general, its power to discriminate between regions increases as the number of components of vector $\mathbf{z}$ in Eq. (10-84) increases.

REGION SEGMENTATION USING SUPERPIXELS

The idea behind *superpixels* is to replace the standard pixel grid by grouping pixels into primitive regions that are more perceptually meaningful than individual pixels. The objectives are to lessen computational load, and to improve the performance of segmentation algorithms by reducing irrelevant detail. A simple example will help explain the basic approach of superpixel representations.

Figure 10.50(a) shows an image of size 600×800 (480,000) pixels containing various levels of detail that could be described verbally as: "This is an image of two large carved figures in the foreground, and at least three, much smaller, carved figures resting on a fence behind the large figures. The figures are on a beach, with



a b c

FIGURE 10.50 (a) Image of size 600×480 (480,000) pixels. (b) Image composed of 4,000 superpixels (the boundaries between superpixels (in white) are superimposed on the superpixel image for reference—the boundaries are not part of the data). (c) Superpixel image. (Original image courtesy of the U.S. National Park Services.).

Figures 10.50(b) and (c) were obtained using a method to be discussed later in this section.

the ocean and sky in the background." Figure 10.50(b) shows the same image represented by 4,000 superpixels and their boundaries (the boundaries are shown for reference—they are not part of the data), and Fig. 10.50(c) shows the superpixel image. One could argue that the level of detail in the superpixel image would lead to the same description as the original, but the former contains only 4,000 primitive units, as opposed to 480,000 in the original. Whether the superpixel representation is "adequate" depends on the application. If the objective is to describe the image at the level of detail mentioned above, then the answer is yes. On the other hand, if the objective is to detect imperfections at pixel-level resolutions, then the answer obviously is no. And there are application, such as computerized medical diagnosis, in which approximate representations of any kind are not acceptable. Nevertheless, numerous application areas, such as image-database queries, autonomous navigation, and certain branches of robotics, in which economy of implementation and potential improvements in segmentation performance far outweigh any appreciable loss of image detail.

One important requirement of any superpixel representation is *adherence to boundaries*. This means that boundaries between regions of interest must be preserved in a superpixel image. We can see that this indeed is the case with the image in Fig. 10.50(c). Note, for example, how clear the boundaries between the figures and the background are. The same is true of the boundaries between the beach and the ocean, and between the ocean and the sky. Other important characteristics are the preservations of topological properties and, of course, computational efficiency. The superpixel algorithm discussed in this section meets these requirements.

As another illustration, we show the results of severely decreasing the number of superpixels to 1,000, 500, and 250. The results in Fig. 10.51, show a significant loss of detail compared to Fig. 10.50(a), but the first two images contain most of the detail relevant to the image description discussed earlier. A notable difference is that two of the three small carvings on the fence in the back were eliminated. The 250-element superpixel image even lost the third. However, the boundaries between the principal regions, as well as the basic topology of the images, were preserved.

FIGURE 10.51 Top row: Results of using 1,000, 500, and 250 superpixels in the representation of Fig. 10.50(a). As before, the boundaries between superpixels are superimposed on the images for reference. Bottom row: Superpixel images.

SLIC Superpixel Algorithm

In this section we discuss an algorithm for generating superpixels, called *simple linear iterative clustering* (SLIC). This algorithm, developed by Achanta et al. [2012], is conceptually simple, and has computational and other performance advantages over other superpixels techniques. SLIC is a modification of the k-means algorithm discussed in the previous section. SLIC observations typically use (but are not limited to) 5-dimensional vectors containing three color components and two spatial coordinates. For example, if we are using the RGB color system, the 5-dimensional vector associated with an image pixel has the form

As you will learn in Chapter 11, vectors containing image attributes are called *feature vectors*.

$$\mathbf{z} = \begin{bmatrix} r \\ g \\ b \\ x \\ y \end{bmatrix} \tag{10-86}$$

where (r, g, b) are the three color components of a pixel, and (x, y) are its two spatial coordinates. Let n_{sp} denote the desired number of superpixels and let n_{tp} denote the total number of pixels in the image. The initial superpixel centers, $\mathbf{m}_i = \begin{bmatrix} r_i & g_i & b_i & x_i & y_i \end{bmatrix}^T$, $i = 1, 2, \ldots, n_{sp}$, are obtained by sampling the image on a regular grid spaced s units apart. To generate superpixels approximately equal in size (i.e., area), the grid spac-

ing interval is selected as $s = [n_{tp}/n_{sp}]^{1/2}$. To prevent centering a superpixel on the edge of the image, and to reduce the chances of starting at a noisy point, the initial cluster centers are moved to the lowest gradient position in the 3×3 neighborhood about each center.

The SLIC superpixel algorithm consists of the following steps. Keep in mind that superpixels are vectors in general. When we refer to a "pixel" in the algorithm, we are referring to the (x, y) *location* of the superpixel relative to the image.

1. ***Initialize the algorithm:*** Compute the initial superpixel cluster centers,

$$\mathbf{m}_i = \left[r_i \;\; g_i \;\; b_i \;\; x_i \;\; y_i\right]^T, \;\; i = 1, 2, \ldots, n_{sp}$$

   by sampling the image at regular grid steps, s. Move the cluster centers to the lowest gradient position in a 3×3 neighborhood. For each pixel location, p, in the image, set a label $L(p) = -1$ and a distance $d(p) = \infty$.
2. ***Assign samples to cluster centers:*** For each cluster center $\mathbf{m}_i$, $i = 1, 2, \ldots, n_{sp}$, compute the distance, $D_i(p)$ between $\mathbf{m}_i$ and *each* pixel p in a $2s \times 2s$ neighborhood about $\mathbf{m}_i$. Then, for each p and $i = 1, 2, \ldots, n_{sp}$, if $D_i < d(p)$, let $d(p) = D_i$ and $L(p) = i$.
3. ***Update the cluster centers:*** Let C_i denote the set of pixels in the image with label $L(p) = i$. Update $\mathbf{m}_i$:

$$\mathbf{m}_i = \frac{1}{|C_i|} \sum_{\mathbf{z} \in C_i} \mathbf{z} \qquad i = 1, 2, \ldots, n_{sp}$$

   where $|C_i|$ is the number of pixels in set C_i, and the $\mathbf{z}$'s are given by Eq. (10-86).
4. ***Test for convergence:*** Compute the Euclidean norms of the differences between the mean vectors in the current and previous steps. Compute the residual error, E, as the sum of the n_{sp} norms. If $E < T$, where T a specified nonnegative threshold, go to Step 5. Else, go back to Step 2.
5. ***Post-process the superpixel regions:*** Replace all the superpixels in each region, C_i, by their average value, $\mathbf{m}_i$.

Note in Step 5 that superpixels end up as contiguous regions of constant value. The average value is not the only way to compute this constant, but it is the most widely used. For graylevel images, the average is just the average intensity of all the pixels in the region spanned by the superpixel. This algorithm is similar to the k-means algorithm in the previous section, with the exceptions that the distances, D_i, are not specified as Euclidean distances (see below), and that these distances are computed for regions of size $2s \times 2s$, rather than for all the pixels in the image, thus reducing computation time significantly. In practice, SLIC convergence with respect to E can be achieved with fairly large values of T. For example, all results reported by Achanta et al. [2012] were obtained using $T = 10$.

Specifying the Distance Measure

SLIC superpixels correspond to clusters in a space whose coordinates are colors and spatial variables. It would be senseless to use a single Euclidean distance in this case, because the scales in the axes of this coordinate system are different and unrelated. In other words, spatial and color distances must be treated separately. This is accomplished by normalizing the distance of the various components, then combining them into a single measure. Let d_c and d_s denote the color and spatial Euclidean distances between two points in a cluster, respectively:

$$d_c = \left[(r_j - r_i)^2 + (g_j - g_i)^2 + (b_j - b_i)^2\right]^{1/2} \tag{10-87}$$

and

$$d_s = \left[(x_j - x_i)^2 + (y_j - y_i)^2\right]^{1/2} \tag{10-88}$$

We then define D as the *composite* distance

$$D = \left[\left(\frac{d_c}{d_{cm}}\right)^2 + \left(\frac{d_s}{d_{sm}}\right)^2\right]^{1/2} \tag{10-89}$$

where d_{cm} and d_{sm} are the maximum expected values of d_c and d_s. The maximum spatial distance should correspond to the sampling interval; that is, $d_{sm} = s = [n_{tp}/n_{sp}]^{1/2}$. Determining the maximum color distance is not as straightforward, because these distances can vary significantly from cluster to cluster, and from image to image. A solution is to set d_{cm} to a constant c so that Eq. (10-89) becomes

$$D = \left[\left(\frac{d_c}{c}\right)^2 + \left(\frac{d_s}{s}\right)^2\right]^{1/2} \tag{10-90}$$

We can write this equation as

$$D = \left[d_c^2 + \left(\frac{d_s}{s}\right)^2 c^2\right]^{1/2} \tag{10-91}$$

This is the distance measure used for each cluster in the algorithm. Constant c can be used to weigh the relative importance between color similarity and spatial proximity. When c is large, spatial proximity is more important, and the resulting superpixels are more compact. When c is small, the resulting superpixels adhere more tightly to image boundaries, but have less regular size and shape.

For grayscale images, as in Example 10.23 below, we use

$$d_c = \left[(l_j - l_i)^2\right]^{1/2} \tag{10-92}$$

in Eq. (10-91), where the I's are intensity levels of the points for which the distance is being computed.

In 3-D, superpixels become *supervoxels*, which are handled by defining

$$d_s = \left[(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2\right]^{1/2} \tag{10-93}$$

where the z's are the coordinates of the third spatial dimension. We must also add the third spatial variable, z, to the vector in Eq. (10-86).

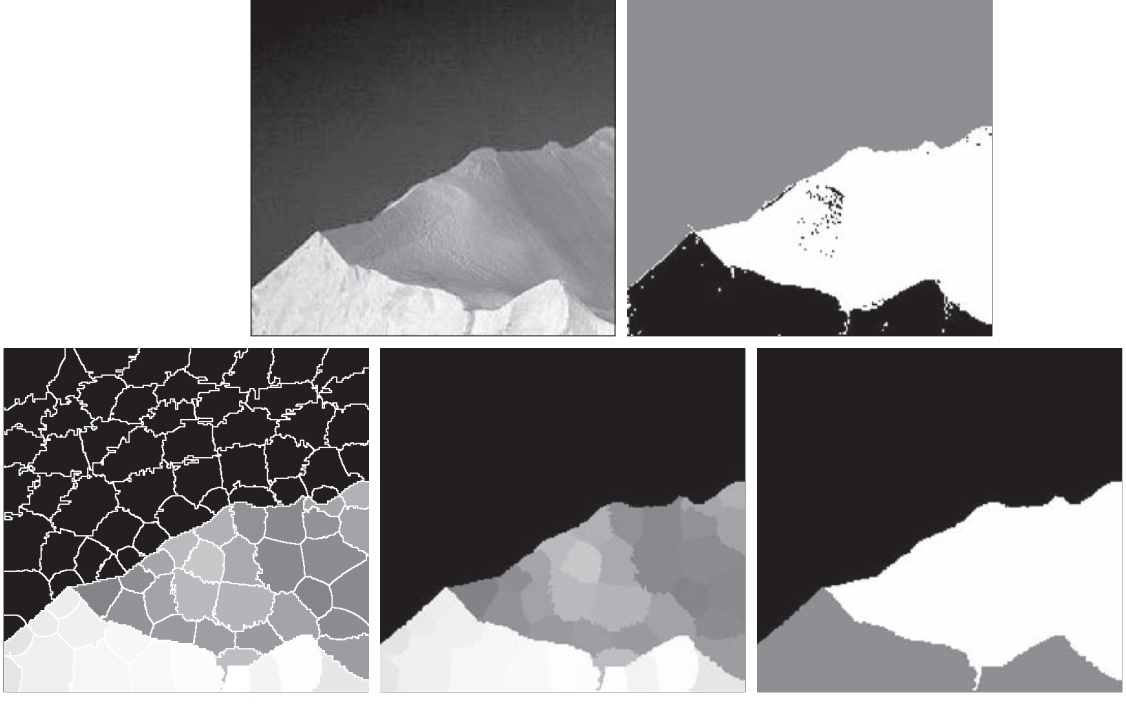
Because no provision is made in the algorithm to enforce connectivity, it is possible for isolated pixels to remain after convergence. These are assigned the label of the nearest cluster using a connected components algorithm (see Section 9.6). Although we explained the algorithm in the context of RGB color components, the method is equally applicable to other colors systems. In fact, other components of vector $\mathbf{z}$ in Eq. (10-86) (with the exception of the spatial variables) could be other real-valued feature values, provided that a meaningful distance measure can be defined for them.

EXAMPLE 10.23: Using superpixels for image segmentation.

Figure 10.52(a) shows an image of an iceberg, and Fig. 10.52(b) shows the result of segmenting this image using the k-means algorithm developed in the last section, with $k = 3$. Although the main regions of the image were segmented, there are numerous segmentation errors in both regions of the iceberg, and also on the boundary separating it from the background. Errors are visible as isolated pixels (and also as small groups of pixels) with the wrong shade (e.g., black pixels within a white region). Figure 10.52(c) shows a 100-superpixel representation of the image with the superpixel boundaries superimposed for reference, and Fig. 10.52(d) shows the same image without the boundaries. Figure 10.52(e) is the segmentation of (d) using the k-means algorithm with $k = 3$ as before. Note the significant improvement over the result in (b), indicating that the original image has considerably more (irrelevant) detail than is needed for a proper segmentation. In terms of computational advantage, consider that generating Fig. 10.52(b) required individual processing of over 300K pixels, while (e) required processing of 100 pixels with considerably fewer shades of gray.

10.6 REGION SEGMENTATION USING GRAPH CUTS

In this section, we discuss an approach for partitioning an image into regions by expressing the pixels of the image as nodes of a graph, and then finding an optimum partition (*cut*) of the graph into groups of nodes. Optimality is based on criteria whose values are high for members within a group (i.e., a region) and low across members of different groups. As you will see later in this section, graph-cut segmentation is capable in some cases of results that can be superior to the results achievable by any of the segmentation methods studied thus far. The price of this potential benefit is added complexity in implementation, which generally translates into slower execution.



a b
c d e

FIGURE 10.52 (a) Image of size 533×566 (301,678) pixels. (b) Image segmented using the k-means algorithm. (c) 100-element superpixel image showing boundaries for reference. (d) Same image without boundaries. (e) Superpixel image (d) segmented using the k-means algorithm. (Original image courtesy of NOAA.)

IMAGES AS GRAPHS

Nodes and edges are also referred to as *vertices* and *links*, respectively.

A graph, G, is a mathematical structure consisting of a set V of *nodes* and a set E of *edges* connecting those vertices:

$$G = (V, E) \tag{10-94}$$

where V is a set and

See Section 2.5 for an explanation of the Cartesian product $V \times V$ and for a review of the set symbols used in this section.

$$E \subseteq V \times V \tag{10-95}$$

is a set of ordered pairs of elements from V. If $(u, v) \in E$ implies that $(v, u) \in E$, and vice versa, the graph is said to be *undirected*; otherwise the graph is *directed*. For example, we may consider a street map as a graph in which the nodes are street intersections, and the edges are the streets connecting those intersections. If all streets are bidirectional, the graph is undirected (meaning that we can travel both ways from any two intersections). Otherwise, if at least one street is a one-way street, the graph is directed.

The types of graphs in which we are interested are undirected graphs whose edges are further characterized by a matrix, $\mathbf{W}$, whose element $w(i,j)$ is a weight associated with the edge that connects nodes i and j. Because the graph is undirected, $w(i,j) = w(j,i)$, which means that $\mathbf{W}$ is a symmetric matrix. The weights are selected to be proportional to one or more similarity measures between all pairs of nodes. A graph whose edges are associated with weights is called a *weighted graph*.

Superpixels are also well suited for use as graph nodes. Thus, when we refer in this section to "pixels" in an image, we are, by implication, also referring to superpixels.

The essence of the material in this section is to represent an image to be segmented as a weighted, undirected graph, where the nodes of the graph are the pixels in the image, and an edge is formed between every pair of nodes. The weight, $w(i,j)$, of each edge is a function of the similarity between nodes i and j. We then seek to partition the nodes of the graph into disjoint subsets $V_1, V_2, \ldots, V_K$ where, by some measure, the similarity among the nodes within a subset is high, and the similarity across the nodes of different subsets is low. The nodes of the partitioned subsets correspond to the regions in the segmented image.

Set V is partitioned into subsets by cutting the graph. A *cut* of a graph is a partition of V into two subsets A and B such that

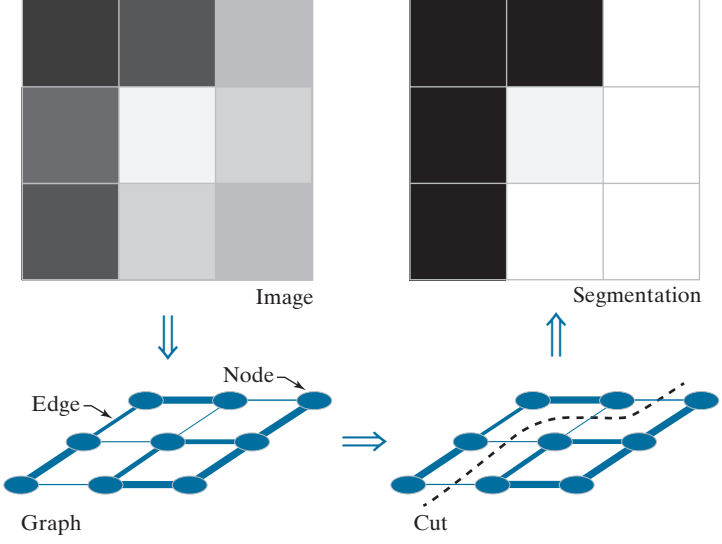
$$A \cup B = V \text{ and } A \cap B = \varnothing \tag{10-96}$$

where the cut is implemented by removing the edges connecting subgraphs A and B. There are two key aspects of using graph cuts for image segmentation: (1) how to associate a graph with an image; and (2) how to cut the graph in a way that makes sense in terms of partitioning the image into background and foreground (object) pixels. We address these two questions next.

Figure 10.53 shows a simplified approach for generating a graph from an image. The nodes of the graph correspond to the pixels in the image and, to keep the explanation simple, we allow edges only between adjacent pixels using 4-connectivity, which means that there are no diagonal edges linking the pixels. But, keep in mind that, in general, edges are specified between every pair of pixels. The weights for the edges typically are formed from spatial relationships (for example, distance from the vertex pixel) and intensity measures (for example, texture and color), consistent with exhibiting similarity between pixels. In this simple example, we define the degree of similarity between two pixels as the inverse of the difference in their intensities. That is, for two nodes (pixels) n_i and n_j, the weight of the edge between them is $w(i,j) = 1/(|I(n_i) - I(n_j)| + c)$, where $I(n_i)$ and $I(n_j)$, are the intensities of the two nodes (pixels) and c is a constant included to prevent division by 0. Thus, the closer the values of intensity between adjacent pixels is, the larger the value of w will be.

For illustrative purposes, the thickness of each edge in Fig. 10.53 is shown proportional to the degree of similarity between the pixels that it connects (see Problem 10.44). As you can see in the figure, the edges between the dark pixels are stronger than the edges between dark and light pixels, and vice versa. Conceptually, segmentation is achieved by cutting the graph along its weak edges, as illustrated by the dashed line in Fig. 10.53(d). Figure 10.53(c) shows the segmented image.

Although the basic structure in Fig. 10.53 is the focus of the discussion in this section, we mention for completeness another common approach for constructing



a b
c d

FIGURE 10.53
(a) A 3×3 image. (c) A corresponding graph. (d) Graph cut. (c) Segmented image.

image graphs. Figure 10.54 shows the same graph as the one we just discussed, but here you see two additional nodes called the *source* and *sink terminal* nodes, respectively, each connected to all nodes in the graph via unidirectional links called *t-links*. The terminal nodes are not part of the image; their role, for example, is to associate with each pixel a probability that it is a background or foreground (object) pixel. The probabilities are the weights of the t-links. In Figs. 10.54(c) and (d), the thickness of each t-link is proportional to the value of the probability that the graph node to which it is connected is a foreground or background pixel (the thicknesses shown are so that the segmentation result would be the same as in Fig. 10.53). Which of the two nodes we call background or foreground is arbitrary.

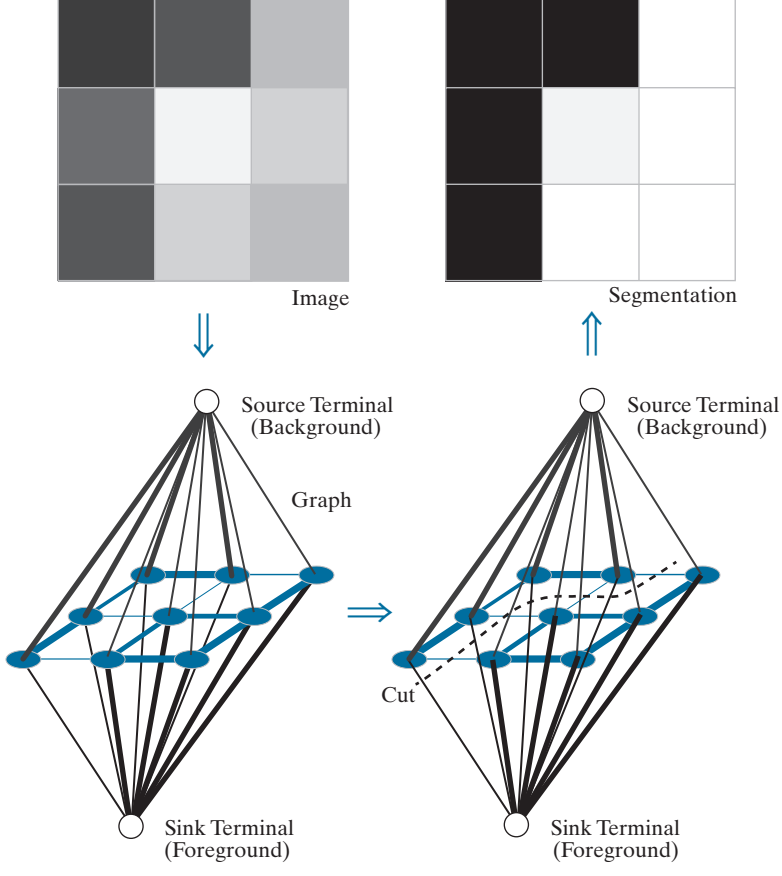
MINIMUM GRAPH CUTS

Once an image has been expressed as a graph, the next step is to cut the graph into two or more subgraphs. The nodes (pixels) in each resulting subgraph correspond to a region in the segmented image. Approaches based on Fig. 10.54 rely on interpreting the graph as a flow network (of pipes, for example) and obtaining what is commonly referred to as a *minimum graph cut*. This formulation is based on the so-called *Max-Flow, Min-Cut Theorem*. This theorem states that, in a flow network, the maximum amount of flow passing from the source to the sink is equal to the *minimum cut*. This minimum cut is defined as the smallest *total* weight of the edges that, if removed, would disconnect the sink from the source:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v) \tag{10-97}$$

a b
c d

FIGURE 10.54
(a) Same image as in Fig. 10.53(a). (c) Corresponding graph and terminal nodes. (d) Graph cut. (b) Segmented image.



where A and B satisfy Eq. (10-96). The optimum partition of a graph is the one that minimizes this cut value. There is an exponential number of such partitions, which would present us with an intractable computational problem. However, efficient algorithms that run in polynomial time have been developed for solving max-flow problems. Therefore, based on the Max-Flow, Min-Cut Theorem, we can apply these algorithms to image segmentation, provided that we cast segmentation as a flow problem and select the weights for the edges and t-links such that minimum graph cuts will result in meaningful segmentations.

Although the min-cut approach offers an elegant solution, it can result in groupings that favor cutting small sets of isolated nodes in a graph, leading to improper segmentations. Figure 10.55 shows an example, in which the two regions of interest are characterized by the tightness of the pixel groupings. Meaningful edge weights that reflect this property would be inversely proportional to the distance between pairs of points. But this would lead to weights that would be smaller for isolated points, resulting in min cuts such as the example in Fig. 10.55. In fact, any cut that partitions out individual points on the left of the figure will have a smaller cut value in Eq. (10-4) than a cut that properly partitions the points into two groups based on

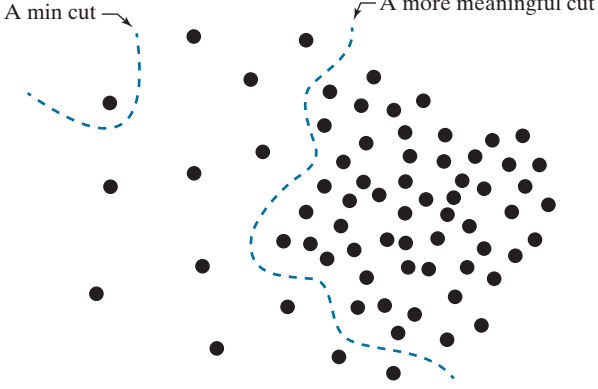


FIGURE 10.55
An example showing how a min cut can lead to a meaningless segmentation. In this example, the similarity between pixels is defined as their spatial proximity, which results in two distinct regions.

their proximity, such as the partition shown in Fig. 10.55. The approach presented in this section, proposed by Shi and Malik [2000] (see also Hochbaum [2010]), is aimed at avoiding this type of behavior by redefining the concept of a cut.

Instead of looking at the total weight value of the edges that connect two partitions, the idea is to work with a measure of "disassociation" that computes the cost as a fraction of the total edge connections to all nodes in the graph. This measure, called the *normalized cut* (*Ncut*), is defined as

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)} \tag{10-98}$$

where $cut(A,B)$ is given by Eq. (10-97) and

$$assoc(A,V) = \sum_{u \in A, z \in V} w(u,z) \tag{10-99}$$

is the sum of the weights of all the edges from the nodes of subgraph A to the nodes of the entire graph. Similarly,

$$assoc(B,V) = \sum_{v \in B, z \in V} w(v,z) \tag{10-100}$$

is the sum of the weights of the edges from all the edges in B to the entire graph. As you can see, $assoc(A,V)$ is simply the cut of A from the rest of the graph, and similarly for $assoc(B,V)$.

By using $Ncut(A,B)$ instead of $cut(A,B)$, the cut that partitions isolated points will no longer have small values. You can see this, for example, by noting in Fig. 10.55 that if A is the single node shown, $cut(A,B)$ and $assoc(A,V)$ will have the same value. Thus, independently of how small $cut(A,B)$ is, $Ncut(A,B)$ will always be greater than or equal to 1, thus providing normalization for "pathological" cases such as this.

Based on similar concepts, we can define a measure for total *normalized association* within graph partitions as

$$Nassoc(A,B) = \frac{assoc(A,A)}{assoc(A,V)} + \frac{assoc(B,B)}{assoc(B,V)} \tag{10-101}$$

where $assoc(A,A)$ and $assoc(B,B)$ are the total weights connecting the nodes within A and within B, respectively. It is not difficult to show (see Problem 10.46) that

$$Ncut(A,B) = 2 - Nassoc(A,B) \tag{10-102}$$

which implies that minimizing $Ncut(A,B)$ simultaneously maximizes $Nassoc(A,B)$.

Based on the preceding discussion, image segmentation using graph cuts is now based on finding a partition that minimizes $Ncut(A,B)$. Unfortunately, minimizing this quantity exactly is an NP-complete computational task, and we can no longer rely on the solutions available for max flow because the approach being followed now is based on the concepts explained in connection with Fig. 10.53. However, Shi and Malik [2000] (see also Hochbaum [2010]) were able to find an approximate discrete solution to minimizing $Ncut(A,B)$ by formulating minimization as a generalized eigenvalue problem, for which numerous implementations exist.

COMPUTING MINIMAL GRAPH CUTS

If the nodes of graph G are the pixels in an image, then $K = M \times N$, where M and N are the number of rows and columns in the image.

As above, let V denote the nodes of a graph G, and let A and B be two subsets of V satisfying Eq. (10-96). Let K denote the number of nodes in V and define a K-dimensional *indicator* vector, $\mathbf{x}$, whose element x_i has the property $x_i = 1$ if node n_i of V is in A and $x_i = -1$ if it is in B. Let

$$d_i = \sum_j w(i,j) \tag{10-103}$$

be the sum of the weights from node n_i to all other nodes in V. Using these definitions, we can write Eq. (10-98) as

$$\begin{aligned} Ncut(A,B) &= \frac{cut(A,B)}{cut(A,V)} + \frac{cut(A,B)}{cut(B,V)} \\ &= \frac{\sum_{x_i>0,\, x_j<0} -w(i,j)x_i x_j}{\sum_{x_i>0} d_i} + \frac{\sum_{x_i<0,\, x_j>0} -w(i,j)x_i x_j}{\sum_{x_i<0} d_i} \end{aligned} \tag{10-104}$$

The objective is to find a vector, $\mathbf{x}$, that minimizes $Ncut(A,B)$. A closed-form solution that minimizes Eq. (10-104) can be found, but only if the elements of $\mathbf{x}$ are allowed to be real, continuous numbers instead of being constrained to be ± 1. The solution derived by Shi and Malik [2000] is given by solving the generalized eigensystem expression

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y} \tag{10-105}$$

where $\mathbf{D}$ is a $K \times K$ diagonal matrix with main-diagonal elements d_i, $i = 1, 2, \ldots, K$, and $\mathbf{W}$ is a $K \times K$ weight matrix with elements $w(i,j)$, as defined earlier. Solving

Eq. (10-105) gives K eigenvalues and K eigenvectors, each corresponding to one eigenvalue. The solution to our problem is the eigenvector corresponding the *second* smallest eigenvalue.

We can convert the preceding generalized eigenvalue formulation into a standard eigenvalue problem by writing Eq. (10-105) as (see Problem 10.45):

$$\mathbf{A}\mathbf{z} = \lambda\mathbf{z} \tag{10-106}$$

where

$$\mathbf{A} = \mathbf{D}^{-\frac{1}{2}}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-\frac{1}{2}} \tag{10-107}$$

and

$$\mathbf{z} = \mathbf{D}^{\frac{1}{2}}\mathbf{y} \tag{10-108}$$

from which it follows that

$$\mathbf{y} = \mathbf{D}^{-\frac{1}{2}}\mathbf{z} \tag{10-109}$$

Thus, we can find the (continuous-valued) eigenvector corresponding to the second smallest eigenvalue using either a generalized or a standard eigenvalue solver. The desired (discrete) vector $\mathbf{x}$ can be generated from the resulting, continuous valued solution vector by finding a splitting point that divides the values of the continuous eigenvector elements into two parts. We do this by finding the splitting point that yields the smallest value of $Ncut(A, B)$, since this is the quantity we are trying to minimize. To simplify the search, we divide the range of values in the continuous vector into Q evenly spaced values, evaluate Eq. (10-104) for each value, and choose the splitting point that yields the smallest value of $Ncut(A, B)$. Then, all values of the eigenvector with values above the split point are assigned the value 1; all others are assigned the value –1. The result is the desired vector $\mathbf{x}$. Then, partition A is the set nodes in V corresponding to 1's in $\mathbf{x}$; the remaining nodes correspond to partition B. This partitioning is carried out only if the stability criterion discussed in the following paragraph is met.

Searching for a splitting point implies computing a total of Q values of $Ncut(A, B)$ and selecting the smallest one. A region that is not clearly segmentable into two subregions using the specified weights will usually result in many splitting points with similar values of $Ncut(A, B)$. Trying to segment such a region is likely to result in a meaningless partition. To avoid this behavior, a region (i.e., subgraph) is split only if it satisfies a *stability criterion*, obtained by first computing the histogram of the eigenvector values, then forming the ratio of the minimum to the maximum bin counts. In an "uncertain" eigenvector, the values in the histogram will stay relatively the same, and the ratio will be relatively high. Shi and Malik [2000] found experimentally that thresholding the ratio at 0.06 was a effective criterion for not splitting the region in question.

GRAPH CUT SEGMENTATION ALGORITHM

In the preceding discussion, we illustrated two ways in which edge weights can be generated from an image. In Figs. 10.53 and 10.54, we looked at weights generated using image intensity values, and in Fig. 10.55 we considered weights based on the distance between pixels. But these are just two examples of the many ways that we can generate a graph and corresponding weights from an image. For example, we could use color, texture, statistical moments about a region, and other types of features to be discussed in Chapter 11. In general, then, graphs can be constructed from image *features*, of which pixel intensities are a special case. With this concept as background, we can summarize the discussion thus far in this section as the following algorithm:

1. Given a set of features, specify a weighted graph, $G = (V, E)$ in which V contains the points in the feature space, and E contains the edges of the graph. Compute the edge weights and use them to construct matrices $\mathbf{W}$ and $\mathbf{D}$. Let K denote the desired number of partitions of the graph.
2. Solve the eigenvalue system $(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y}$ to find the eigenvector with the second smallest eigenvalue.
3. Use the eigenvector from Step 2 to bipartition the graph by finding the splitting point such that $Ncut(A, B)$ is minimized.
4. If the number of cuts has not reached K, decide if the current partition should be subdivided by checking the stability of the cut.
5. Recursively repartition the segmented parts if necessary.

Note that the algorithm works by recursively generating two-way cuts. The number of groups (e.g., regions) in the segmented image is controlled by K. Other criteria, such as the maximum size allowed for each cut, can further refine the final segmentation. For example, when using pixels and their intensities as the basis for constructing the graph, we can specify the maximum and/or minimum size allowed for each region.

EXAMPLE 10.24: Specifying weights for graph cut segmentation.

In Fig. 10.53, we illustrated how to generate graph weights using intensity values, and in Fig. 10.55 we discussed briefly how to generate weights based on the distance between pixels. In this example, we give a more practical approach for generating weights that include both intensity and distance from a pixel, thus introducing the concept of a neighborhood in graph segmentation.

Let n_i and n_j denote two nodes (image pixels). As mentioned earlier in this section, weights are supposed to reflect the similarity between nodes in a graph. When considering segmentation, one of the principal ways to establish how likely two pixels in an image are to be a part of the same region or object is to determine the difference in their intensity values, and how close the pixels are to each other. The weight value of the edge between two pixels should be large when the pixels are very close in intensity and proximity (i.e., when the pixels are "similar), and should decrease as their intensity difference and distance from each other increases. That is, the weight value should be a function of how similar the pixels are in intensity and distance. These two concepts can be embedded into a single weight function using the following expression:

$$w(i,j) = \begin{cases} e^{-\frac{[I(n_i)-I(n_j)]^2}{\sigma_I^2}} e^{-\frac{dist(n_i,n_j)}{\sigma_d^2}} & \text{if } dist(n_i,n_j) < r \\ 0 & \text{otherwise} \end{cases}$$

where $I(n_i)$ is the intensity of node n_i, σ_I^2 and σ_d^2 are constants determining the spread of the two Gaussian-like functions, $dist(n_i, n_j)$ is the distance (e.g., the Euclidean distance) between the two nodes, and r is a radial constant that establishes how far away we are willing to consider similarity. The exponential terms decrease as a function of dissimilarity in intensity and as function of distance between the nodes, as required of our measure of similarity in this case.

EXAMPLE 10.25: Segmentation using graph cuts.

Graph cuts are ideally suited for obtaining a rough segmentation of the principal regions in an image. Figure 10.56 shows a typical result. Figure 10.56(a) is the familiar building image. Consistent with the idea of extracting the principal regions of an image, Fig. 10.56(b) shows the image smoothed with a simple 25×25 box kernel. Observe how the fine detail is smoothed out, leaving only major regional features such as the facade and sky. Figure 10.56(c) is the result of segmentation using the graph cut algorithm just developed, with weights of the form discussed in the previous example, and allowing only two partitions. Note how well the region corresponding to the building was extracted, with none of the details characteristic of the methods discussed earlier in this chapter. In fact, it would have been nearly impossible to obtain comparable results using any of the methods we have discussed thus far without significant additional processing. This type of result is ideal for tasks such as providing broad cues for autonomous navigation, for searching image databases, and for low-level image analysis.

10.7 SEGMENTATION USING MORPHOLOGICAL WATERSHEDS

Thus far, we have discussed segmentation based on three principal concepts: edge detection, thresholding, and region extraction. 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*. 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, as we discuss at the end of this section.

BACKGROUND

The concept of a watershed 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: (1) points belonging to a regional minimum; (2) points at which a drop of water, if placed at the location of any of those

a b c

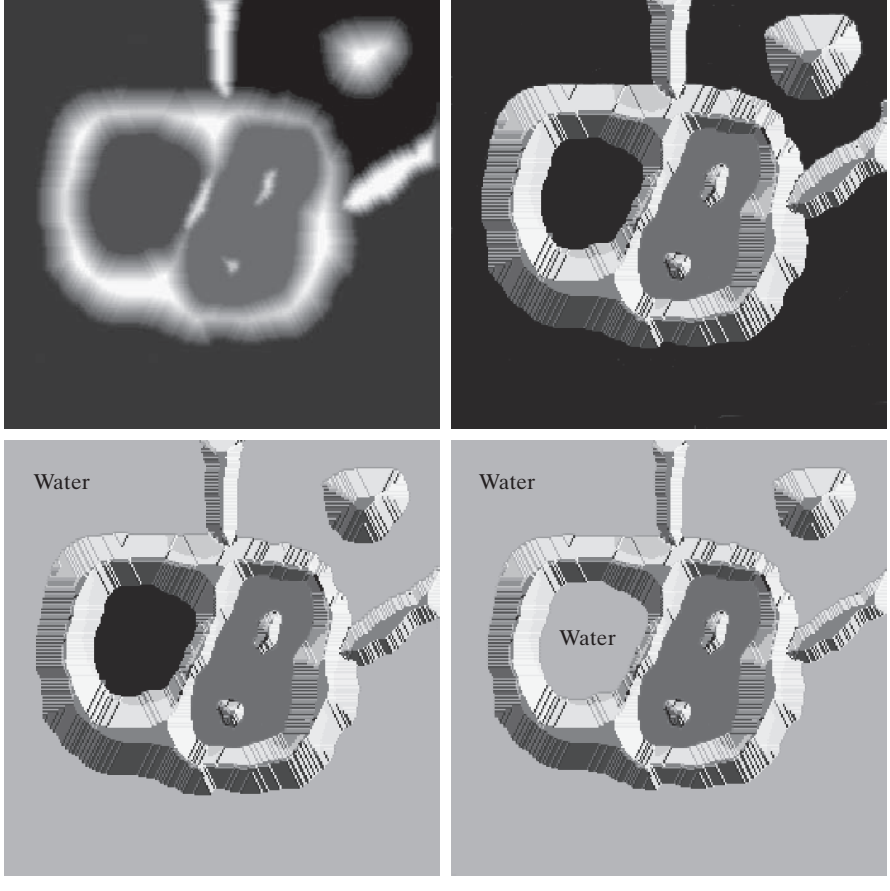
FIGURE 10.56 (a) Image of size 600×600 pixels. (b) Image smoothed with a 25×25 box kernel. (c) Graph cut segmentation obtained by specifying two regions.

points, would fall with certainty to a single minimum; and (3) 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 (2) is called the *catchment basin* or *watershed* of that minimum. The points satisfying condition (3) form crest lines on the topographic surface, and are referred to as *divide lines* or *watershed lines*.

The principal objective of segmentation algorithms based on these concepts is to find the watershed lines. The method for doing this can be explained with the aid of Fig. 10.57. Figure 10.57(a) shows a gray-scale image and Fig. 10.57(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 perimeter of the entire topography (image) being enclosed by dams that are higher 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. 10.57(b)] and that the entire topography is flooded from below by letting water rise through the holes at a uniform rate. Figure 10.57(c) shows the first stage of flooding, where the "water," shown in light gray, has covered only areas that correspond to the black *background* in the image. In Figs. 10.57(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 10.57(f). Here, water from the lower part of the left basin 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 mathematical details of dam building are discussed in the following section). The

Because of neighboring contrast, the leftmost basin in Fig. 10.57(c) appears black, but it is a few shades lighter than the black background. The mid-gray in the second basin is a natural gray from the image in (a).



a c
b d

FIGURE 10.57
(a) Original image. (b) Topographic view. Only the background is *black*. The basin on the left is slightly lighter than black. (c) and (d) Two stages of flooding. All constant dark values of gray are intensities in the original image. Only constant *light gray* represents "water." (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.) (*Continued on next page*.)

effect is more pronounced as water continues to rise, as shown in Fig. 10.57(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 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 boundaries. The result for this example is shown in Fig. 10.57(h) as dark, one-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 (blob-like) 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.

e f
g h

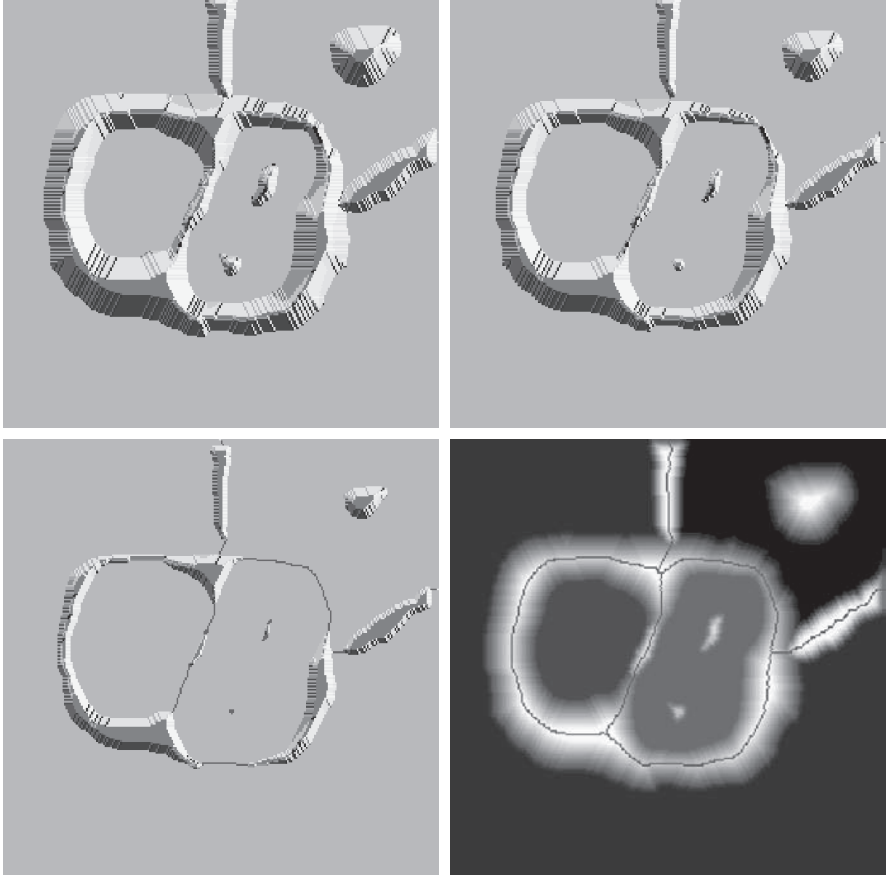


FIGURE 10.57
(Continued)
(e) Result of further flooding. (f) Beginning of merging of water from two catchment basins (a short dam was built between them). (g) Longer dams. (h) Final watershed (segmentation) lines superimposed on the original image. (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

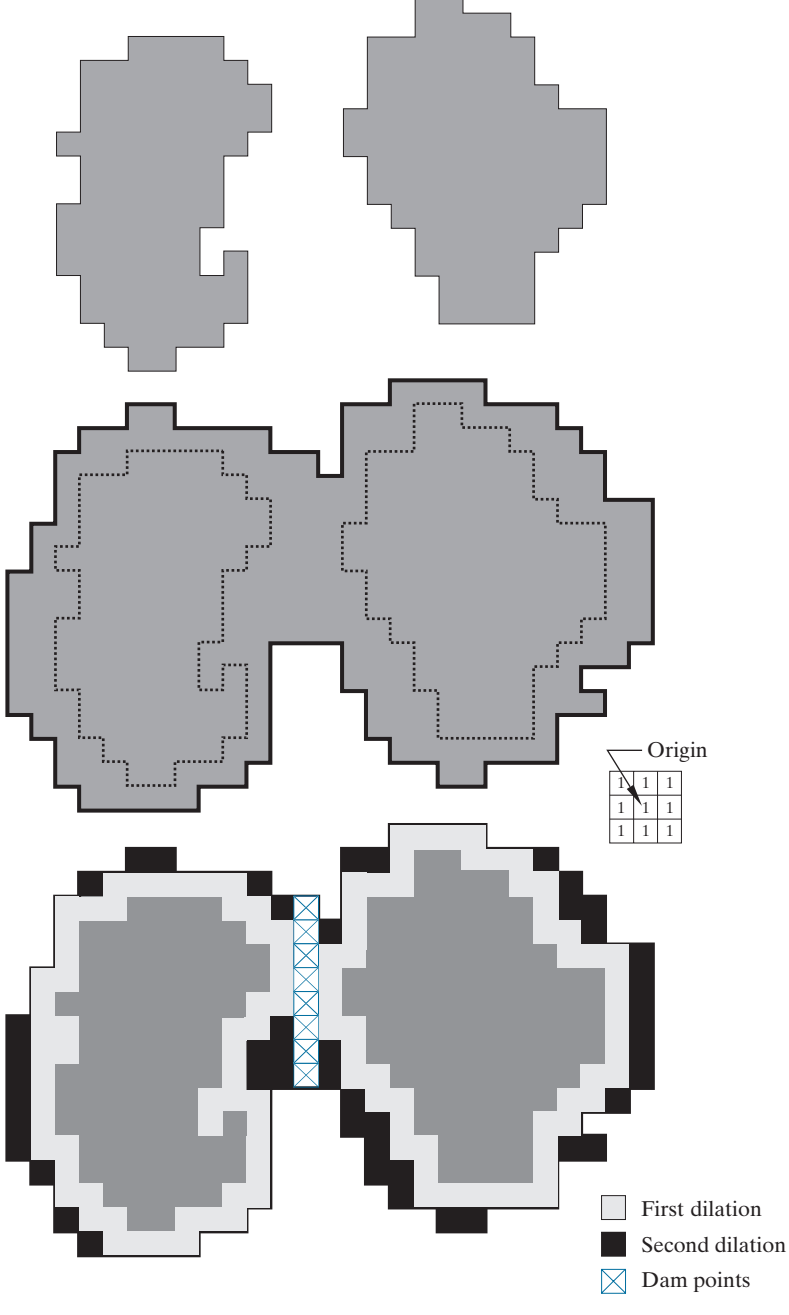
DAM CONSTRUCTION

Dam construction is based on binary images, which are members of 2-D integer space Z^2 (see Sections 2.4 and 2.6). The simplest way to construct dams separating sets of binary points is to use morphological dilation (see Section 9.2).

Figure 10.58 illustrates the basics of dam construction using dilation. Part (a) shows portions of two catchment basins at flooding step $n-1$, and Fig. 10.58(b) shows the result at the next flooding step, n. The water has spilled from one basin to the another 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. 10.58(a).

See Sections 2.5 and 9.5 regarding connected components.

Let $C[n-1]$ denote the union of these two sets. There are two connected components in Fig. 10.58(a), and only one component in Fig. 10.58(b). This connected



a
b c
d

FIGURE 10.58 (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.

component encompasses the earlier two components, which are shown dashed. Two connected components having become a *single* component indicates that water between the two catchment basins has merged at flooding step n. Let this connected component be denoted by q. Note that the two components from step $n-1$ can be extracted from q by performing a logical AND operation, $q \cap C[n-1]$. Observe also that all points belonging to an individual catchment basin form a single connected component.

Suppose that each of the connected components in Fig. 10.58(a) is dilated by the structuring element in Fig. 10.58(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 (i.e., become a single connected component). Figure 10.58(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 during dilation, and that 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 in 10.58(d), several points failed condition (1) while meeting condition (2), resulting in the broken perimeter shown in the figure. It is evident that the only points in q that satisfy the two conditions under consideration describe the one-pixel-thick connected path shown crossed-hatched in Fig. 10.58(d). This path is 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 possible intensity value of the image (e.g., greater than 255 for an 8-bit image). This will prevent water from crossing over the part of the completed dam as the level of flooding is increased. As noted earlier, 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 in Fig. 10.58(c).

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 mentioned earlier, 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,

$$T[n] = \{(s,t) \,|\, g(s,t) < n\} \tag{10-110}$$

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, and will make it easier to understand 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, we may view $C_n(M_i)$ as a binary image given by

$$C_n(M_i) = C(M_i) \cap T[n] \tag{10-111}$$

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, let B denote the number of number of flooded catchment basins at stage n, and let $C[n]$ denote the union of these basins at stage n :

$$C[n] = \bigcup_{i=1}^{B} C_n(M_i) \tag{10-112}$$

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

$$C[\max + 1] = \bigcup_{i=1}^{B} C(M_i) \tag{10-113}$$

It can be shown (see Problem 10.47) 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. (10-112) and (10-113), $C[n]$ is a subset of $T[n]$, so it follows that $C[n-1]$ is also 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 by letting $C[\min+1] = T[\min + 1]$. The procedure then proceeds recursively, successively computing $C[n]$ from $C[n-1]$, using the following approach. 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 $C[n-1]$.

The 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 earlier, a one-pixel-thick dam can be constructed when needed by dilating $q \cap C[n-1]$ with a 3×3 structuring element of 1's, 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 $g(x, y)$.

EXAMPLE 10.26: Illustration of the watershed segmentation algorithm.

Consider the image and its gradient in Figs. 10.59(a) and (b), respectively. Application of the watershed algorithm just described yielded the watershed lines (white paths) shown superimposed on the gradient image in Fig. 10.59(c). These segmentation boundaries are shown superimposed on the original image in Fig. 10.59(d). As noted at the beginning of this section, the segmentation boundaries have the important property of being connected paths.

THE USE OF MARKERS

Direct application of the watershed segmentation algorithm in the form discussed in the previous section generally leads to over-segmentation, caused by noise and other local irregularities of the gradient. As Fig. 10.60 illustrates, over-segmentation 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 over-segmentation 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. 10.60(a) again. Part of the problem that led to the over-segmented result in Fig. 10.60(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 case also.

a b
c d

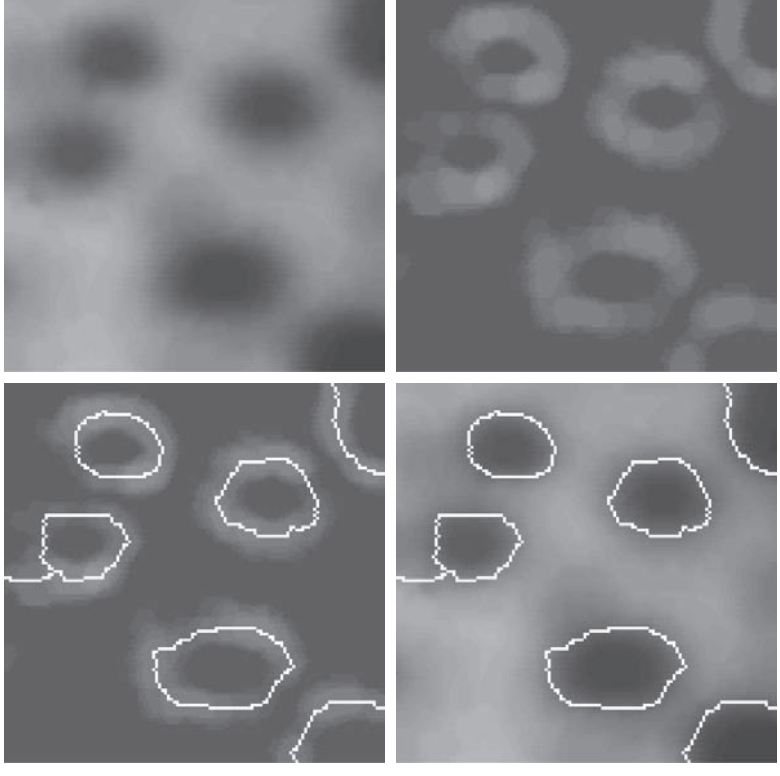


FIGURE 10.59
(a) Image of blobs. (b) Image gradient. (c) Watershed lines, superimposed on the gradient image. (d) Watershed lines superimposed on the original image. (Courtesy of Dr. S. Beucher, CMM/ Ecole des Mines de Paris.)

a b

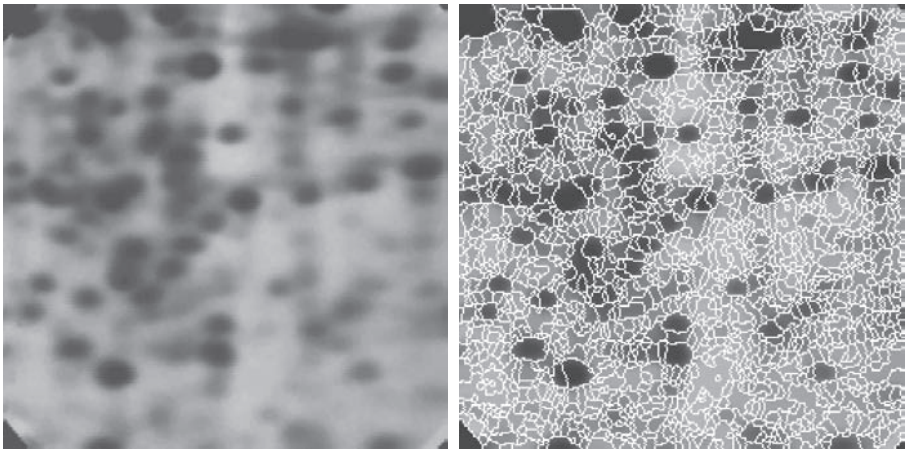


FIGURE 10.60
(a) Electrophoresis image. (b) Result of applying the watershed segmentation algorithm to the gradient image. Over-segmentation is evident. (Courtesy of Dr. S. Beucher, CMM/ Ecole des Mines de Paris.)

a b

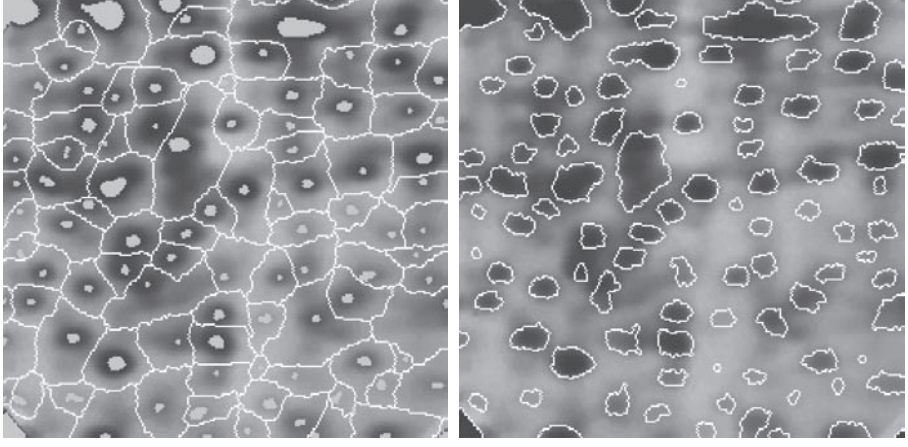


FIGURE 10.61
(a) Image showing internal markers (light gray regions) and external markers (watershed lines).
(b) Result of segmentation. Note the improvement over Fig. 10.60(b). (Courtesy of Dr. S. Beucher, CMM/ Ecole des Mines de Paris.)

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, blob-like regions in Fig. 10.61(a). Next, the watershed algorithm was applied to the smoothed image, under the restriction that these internal markers be the only allowed regional minima. Figure 10.61(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. 10.61(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. 10.59(b)] and restrict the algorithm to operate on a single watershed that contains the marker in that particular region. Figure 10.61(b) shows the result obtained using this approach. The improvement over the image in 10.60(b) is evident.

Marker selection can range from simple procedures based on intensity values and connectivity, as we just illustrated, to more complex descriptions involving size, shape, location, relative distances, texture content, and so on (see Chapter 11 regarding feature descriptors). The point is that using markers brings a priori knowledge to bear on the segmentation problem. Keep in mind 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.

Motion is a powerful cue used by humans and many 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 discussion we consider the use of motion in segmentation both spatially and in the frequency domain.

SPATIAL TECHNIQUES

In what follows, we will consider two approaches for detecting motion, working directly in the spatial domain. The key objective is to give you an idea how to measure changes in digital images using some straightforward techniques.

A 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 one or more moving objects, 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* of two images (of the same size) taken at times t_i and t_j may be defined as

$$d_{ij}(x,y) = \begin{cases} 1 & \text{if } \left|f(x,y,t_i) - f(x,y,t_j)\right| > T \\ 0 & \text{otherwise} \end{cases} \tag{10-114}$$

where T is a nonnegative 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 T. Note also that coordinates (x,y) in Eq. (10-114) span the dimensions of the two images, so the difference image is of the same size as the images in the sequence.

In the discussion that follows, all pixels in $d_{ij}(x,y)$ that have 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 also. 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 1's in image $d_{ij}(x,y)$, 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, and not noise.

Although the method just described is simple, it is used frequently as the basis of imaging systems designed to detect changes in controlled environments, such as in surveillance of parking facilities, buildings, and similar fixed locales.

Accumulative Differences

Consider a sequence of image frames denoted by $f(x,y,t_1), f(x,y,t_2), \ldots, f(x,y,t_n)$, 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. (10-114)] from the corresponding pixel value in the reference image.

Assuming that the intensity values of the moving objects are greater than the background, we consider three types of ADIs. 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 accumulative differences for all relevant values of (x,y):

$$A_k(x,y) = \begin{cases} A_{k-1}(x,y) + 1 & \text{if } |R(x,y) - f(x,y,k)| > T \\ A_{k-1}(x,y) & \text{otherwise} \end{cases} \tag{10-115}$$

$$P_k(x,y) = \begin{cases} P_{k-1}(x,y) + 1 & \text{if } |R(x,y) - f(x,y,k)| > T \\ P_{k-1}(x,y) & \text{otherwise} \end{cases} \tag{10-116}$$

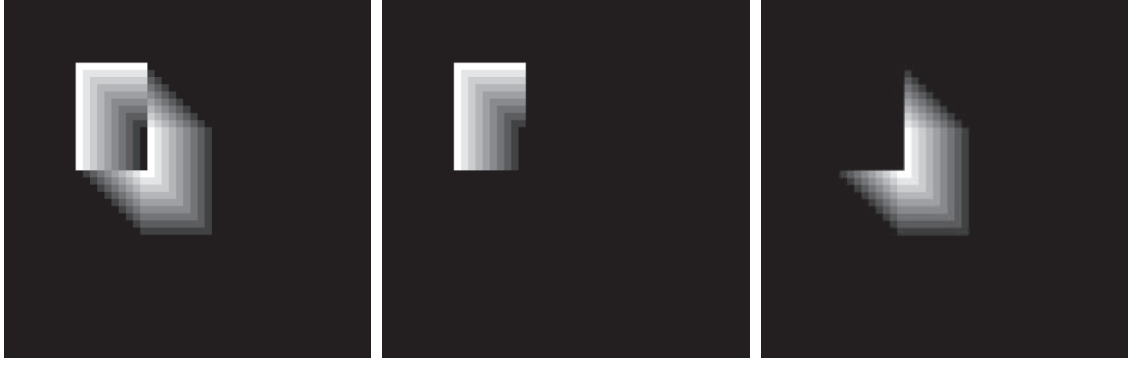
and

$$N_k(x,y) = \begin{cases} N_{k-1}(x,y) + 1 & \text{if } |R(x,y) - f(x,y,k)| < -T \\ N_{k-1}(x,y) & \text{otherwise} \end{cases} \tag{10-117}$$

where $A_k(x,y)$, $P_k(x,y)$, and $N_k(x,y)$ are the *absolute*, *positive*, and *negative* ADIs, respectively, computed using the kth image in the sequence. All three ADIs start out with zero counts and are of the same size as the images in the sequence. The order of the inequalities and signs of the thresholds in Eqs. (10-116) and (10-117) are reversed if the intensity values of the background pixels are greater than the values of the moving objects.

EXAMPLE 10.27: Computation of the absolute, positive, and negative accumulative difference images.

Figure 10.62 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



a b c

FIGURE 10.62 ADIs of a rectangular object moving in a southeasterly direction. (a) Absolute ADI. (b) Positive ADI. (c) Negative ADI.

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; and (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 just discussed is having a reference image against which subsequent comparisons can be 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.

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 earlier. The following example illustrates how to build a reference frame using the approach just described.

EXAMPLE 10.28: Building a reference image.

Figures 10.63(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 above. 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 reference image resulting from removing the east-bound moving vehicle and restoring the background is shown in Fig. 10.63(c).

FREQUENCY DOMAIN TECHNIQUES

In this section, we consider the problem of determining motion via a Fourier transform formulation. Consider a sequence $f(x, y, t)$, $t = 0, 1, 2, \ldots, K - 1$, of K digital image frames of size $M \times N$ pixels, 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 the image plane is projected onto the x-axis; that is, the pixel intensities are summed (for each row) 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 $\exp\left[j2\pi a_1 x \Delta t\right]$ for $x = 0, 1, 2, \ldots, M - 1$ and add the results, we obtain the single term $\exp\left[j2\pi a_1 x' \Delta t\right]$ because there is only one nonzero point in the array. In this notation, a_1 is a positive integer, and Δt is the time interval between frames.

a b c

FIGURE 10.63 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.)

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 $\exp\left[j2\pi a_1 (x' + 1)\Delta t\right]$. If the object continues to move 1 pixel location per frame then, at any integer instant of time, t, the result will be $\exp\left[j2\pi a_1 (x' + t)\Delta t\right]$, which, using Euler's formula, may be expressed as

$$e^{j2\pi a_1 (x' + t)\Delta t} = \cos\left[2\pi a_1 (x' + t)\Delta t\right] + j\sin\left[2\pi a_1 (x' + t)\Delta t\right] \qquad (10\text{-}118)$$

for $t = 0, 1, 2, \ldots, K - 1$. 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 have 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, and may be ignored. Thus a peak search in the Fourier spectrum would yield one peak with value $V_1 a_1$. Dividing 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 analysis 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 $M \times N$ pixels, the sum of the weighted projections onto the x-axis at any integer instant of time is

$$g_x(t, a_1) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y, t) e^{j2\pi a_1 x \Delta t} \quad t = 0, 1, \ldots, K - 1 \qquad (10\text{-}119)$$

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

$$g_y(t, a_2) = \sum_{y=0}^{N-1} \sum_{x=0}^{M-1} f(x, y, t) e^{j2\pi a_2 y \Delta t} \quad t = 0, 1, \ldots, K - 1 \qquad (10\text{-}120)$$

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

The 1D Fourier transforms of Eqs. (10-119) and (10-120), respectively, are

$$G_x(u_1, a_1) = \sum_{t=0}^{K-1} g_x(t, a_1) e^{-j2\pi u_1 t / K} \quad u_1 = 0, 1, \ldots, K - 1 \qquad (10\text{-}121)$$

and

$$G_y(u_2,a_2) = \sum_{t=0}^{K-1} g_y(t,a_2)e^{-j2\pi u_2 t/K} \quad u_2 = 0,1,\ldots,K-1 \tag{10-122}$$

These transforms are computed using an FFT algorithm, as discussed in Section 4.11.

The frequency-velocity relationship is

$$u_1 = a_1 V_1 \tag{10-123}$$

and

$$u_2 = a_2 V_2 \tag{10-124}$$

In the preceding formulation, the unit of velocity is in pixels per total frame time. For example, $V_1 = 10$ indicates 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 $V_1 = 10$, and $K = 30$, 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

$$V_1 = (10\,\text{pixels})(0.5\,\text{m/pixel})(2\,\text{frames/s})(30\ \text{frames})$$

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

$$S_{1x} = \left.\frac{d^2\,\text{Re}\left[g_x(t,a_1)\right]}{dt^2}\right|_{t=n} \tag{10-125}$$

and

$$S_{2x} = \left.\frac{d^2\,\text{Im}\left[g_x(t,a_1)\right]}{dt^2}\right|_{t=n} \tag{10-126}$$

Because g_x is sinusoidal, it can be shown (see Problem 10.53) 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 velocity 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.

EXAMPLE 10.29: Detection of a small moving object via frequency-domain analysis.

Figures 10.64 through 10.66 illustrate the effectiveness of the approach just developed. Figure 10.64 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. 10.65, has a Gaussian intensity distribution spread over a small (9-pixel) area, and is not easily discernible by eye. Figure 10.66 shows

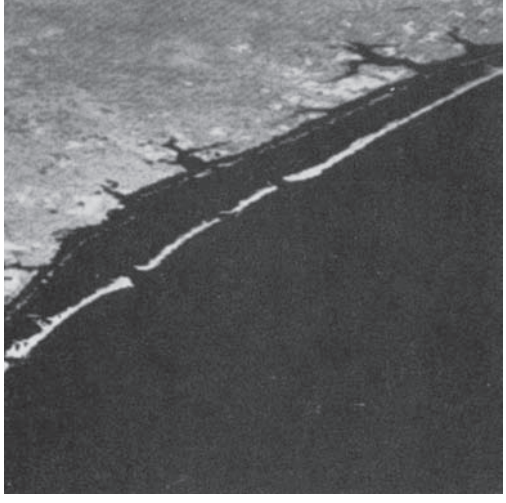


FIGURE 10.64
LANDSAT frame. (Cowart, Snyder, and Ruedger.)

the results of computing Eqs. (10-121) and (10-122) with $a_1 = 6$ and $a_2 = 4$, respectively. The peak at $u_1 = 3$ in Fig. 10.66(a) yields $V_1 = 0.5$ from Eq. (10-123). Similarly, the peak at $u_2 = 4$ in Fig. 10.66(b) yields $V_2 = 1.0$ from Eq. (10-124).

Guidelines for selecting a_1 and a_2 can be explained with the aid of Fig. 10.66. For instance, suppose that we had used $a_2 = 15$ instead of $a_2 = 4$. In that case, the peaks in Fig. 10.66(b) would now be at $u_2 = 15$ and 17 because $V_2 = 1.0$. This would be a seriously aliased result. As discussed in Section 4.5, aliasing is caused by under-sampling (too few frames in the present discussion, as the range of u is determined by K). Because $u = aV$, one possibility is to select a as the integer closest to $a = u_{\max}/V_{\max}$,

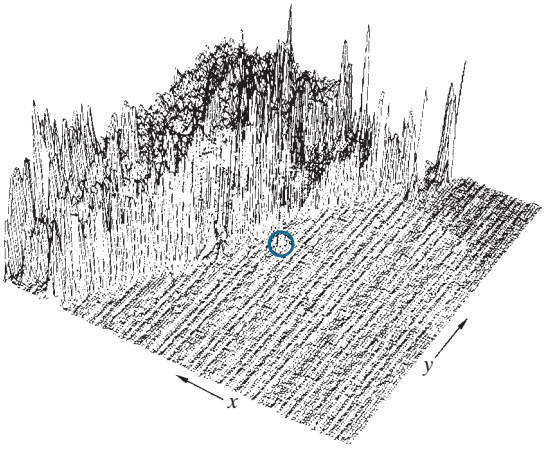
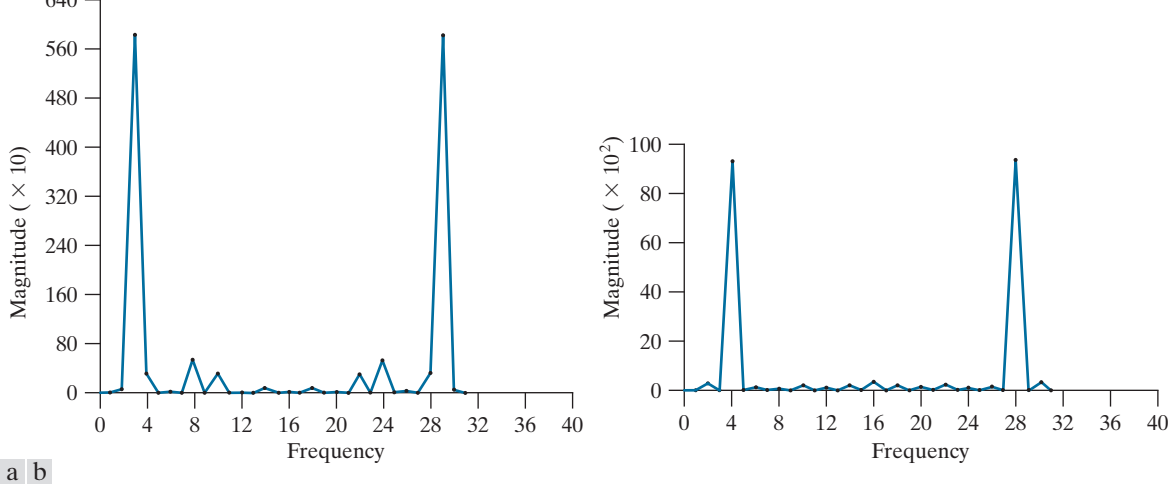


FIGURE 10.65
Intensity plot of the image in Fig. 10.64, with the target circled. (Rajala, Riddle, and Snyder.)



a b

FIGURE 10.66 (a) Spectrum of Eq. (10-121) showing a peak at $u_1 = 3$. (b) Spectrum of Eq. (10-122) showing a peak at $u_2 = 4$. (Rajala, Riddle, and Snyder.)

where u_{max} is the aliasing frequency limitation established by K, and V_{max} is the maximum expected object velocity.

Summary, 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 [2010]; Prince [2012]; Nixon and Aguado, A [2012]; Pratt [2014]; and Petrou and Petrou [2010].

Work dealing with the use of kernels to detect intensity discontinuities (see Section 10.2) has a long history. Numerous kernels have been proposed over the years: Roberts [1965]; Prewitt [1970]; and Kirsh [1971]. The Sobel operators are from [Sobel]; see also Danielsson and Seger [1990]. Our presentation of the zero-crossing properties of the Laplacian is based on Marr [1982]. The Canny edge detector discussed in Section 10.2 is due to Canny [1986]. The basic reference for the Hough transform is Hough [1962]. See Ballard [1981], for a generalization to arbitrary shapes.

Other approaches used to deal with the effects of illumination and reflectance on thresholding are illustrated by the work of Perez and Gonzalez [1987], Drew et al. [1999], and Toro and Funt [2007]. The optimum thresholding approach due to Otsu [1979] has gained considerable acceptance because it combines excellent performance with simplicity of implementation, requiring only estimation of image histograms. The basic idea of using preprocessing to improve thresholding dates back to an early paper by White and Rohrer [1983]), which combined thresholding, the gradient, and the Laplacian in the solution of a difficult segmentation problem.

See Fu and Mui [1981] for an early survey on the topic of region-oriented segmentation. The work of Haddon and Boyce [1990] and of Pavlidis and Liow [1990] are among the earliest efforts to integrate region and boundary information for the purpose of segmentation. Region growing is still an active area of research in image processing, as exemplified by Liangjia et al. [2013]. The basic reference on the *k*-means algorithm presented in Section 10.5 goes way back several decades to an obscure 1957 Bell Labs report by Lloyd, who subsequenty published in Lloyd [1982]. This algorithm was already being in used in areas such as pattern recognition in the 1960s and '70s (Tou and

Gonzalez [1974]). The superpixel algorithm presented in Section 10.5 is from Achanta et al. [2012]. See their paper for a listing and comparison of other superpixel approaches. The material on graph cuts is based on the paper by Shi and Malik [2000]. See Hochbaum [2010] for an example of faster implementations.

Segmentation by watersheds was shown in Section 10.7 to be a powerful concept. Early references dealing with segmentation by watersheds are Serra [1988], and Beucher and Meyer [1992]. As indicated in our discussion in Section 10.7, one of the key issues with watersheds is the problem of over-segmentation. The papers by Bleau and Leon [2000] and by Gaetano et al. [2015] are illustrative of approaches for dealing with this problem.

The material in Section 10.8 dealing with accumulative differences is from Jain, R. [1981]. See also Jain, Kasturi, and Schunck [1995]. The material dealing with motion via Fourier techniques is from Rajala, Riddle, and Snyder [1983]. The books by Snyder and Qi [2004], and by Chakrabarti et al. [2015], provide additional reading on motion estimation. For details on the software aspects of many of the examples in this chapter, see Gonzalez, Woods, and Eddins [2009].

Problems

Solutions to the problems marked with an asterisk (*) are in the DIP4E Student Support Package (consult the book website: www.ImageProcessingPlace.com).

10.1* In a Taylor series approximation, the *remainder* (also called the *truncation error*) consists of all the terms not used in the approximation. The first term in the remainder of a finite difference approximation is indicative of the error in the approximation. The higher the derivative order of that term is, the lower the error will be in the approximation. All three approximations to the first derivative given in Eqs. (10-4)-(10-6) are computed using the same number of sample points. However, the error of the central difference approximation is less than the other two. Show that this is true.

10.2 Do the following:

(a)* Show how Eq. (10-8) was obtained.

(b) Show how Eq. (10-9) was obtained.

10.3 A binary image contains straight lines oriented horizontally, vertically, at 45°, and at –45°. Give a set of 3×3 kernels that can be used to detect one-pixel breaks in these lines. Assume that the intensities of the lines and background are 1 and 0, respectively.

10.4 Propose a technique for detecting gaps of length ranging between 1 and K pixels in line segments of a binary image. Assume that the lines are one pixel thick. Base your technique on 8-neighbor connectivity analysis, rather than attempting to construct kernels for detecting the gaps.

10.5* With reference to Fig. 10.6, what are the angles (measured with respect to the x-axis of the book axis convention in Fig. 2.19) of the horizontal and vertical lines to which the kernels in Figs. 10.6(a) and (c) are most responsive?

10.6 Refer to Fig. 10.7 in answering the following questions.

(a)* Some of the lines joining the pads and center element in Fig. 10.7(e) are single lines, while others are double lines. Explain why.

(b) Propose a method for eliminating the components in Fig. 10.7(f) that are not part of the line oriented at –45°.

(c)

10.7 With reference to the edge models in Fig. 10.8, answer the following without generating the gradient and angle images. Simply provide sketches of the profiles that show what you would expect the profiles of the magnitude and angle images to look like.

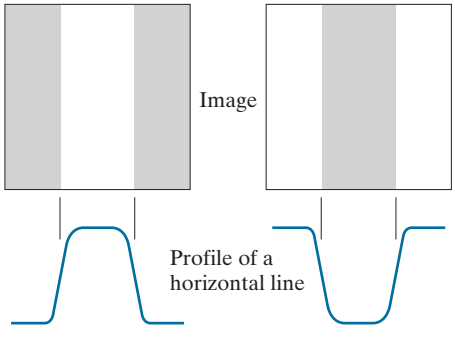
(a)* Suppose that we compute the gradient magnitude of each of these models using the Prewitt kernels in Fig. 10.14. Sketch what a horizontal profile through the center of each gradient image would look like.

(b) Sketch a horizontal profile for each corresponding angle image.

10.8 Consider a horizontal intensity profile through

the middle of a binary image that contains a vertical step edge through the center of the image. Draw what the profile would look like after the image has been blurred by an averaging kernel of size $n \times n$ with coefficients equal to $1/n^2$. For simplicity, assume that the image was scaled so that its intensity levels are 0 on the left of the edge and 1 on its right. Also, assume that the size of the kernel is much smaller than the image, so that image border effects are not a concern near the center of the image.

10.9* Suppose that we had used the edge models in the following image, instead of the ramp in Fig. 10.10. Sketch the gradient and Laplacian of each profile.



10.10 Do the following:

(a)* Show that the *direction* of steepest (maximum) ascent of a function f at point (x, y) is given by the vector $\nabla f(x, y)$ in Eq. (10-16), and that the rate of that descent is $\|\nabla f(x, y)\|$, defined in Eq. (10-17).

(b) Show that the *direction* of *steepest descent* is given by the vector $-\nabla f(x, y)$, and that the *rate* of the steepest descent is $\|\nabla f(x, y)\|$.

(c) Give the description of an image whose gradient magnitude image would be the same, whether we computed it using Eq. (10-17) or (10-26). A constant image is not acceptable answer.

10.11 Do the following.

(a) How would you modify the Sobel and Prewitt kernels in Fig. 10.14 so that they give their strongest gradient response for edges oriented at ±45°?

(b)* Show that the Sobel and Prewitt kernels in Fig. 10.14, and in (a) above, and give isotropic results only for horizontal and vertical edges, and for edges oriented at ±45°, respectively.

10.12 The results obtained by a single pass through an image of some 2-D kernels can be achieved also by two passes using 1-D kernels. For example, the same result of using a 3×3 smoothing kernel with coefficients 1/9 can be obtained by a pass of the kernel [1 1 1] through an image, followed by a pass of the result with the kernel $[1\ 1\ 1]^T$. The final result is then scaled by 1/9. Show that the response of Sobel kernels (Fig. 10.14) can be implemented similarly by one pass of the *differencing* kernel [–1 0 1] (or its vertical counterpart) followed by the smoothing kernel [1 2 1] (or its vertical counterpart).

10.13 A popular variation of the compass kernels shown in Fig. 10.15 is based on using coefficients with values 0, 1, and –1.

(a)* Give the form of the eight compass kernels using these coefficients. As in Fig. 10.15, let N, NW, . . . denote the direction of the edge that gives the strongest response.

(b) Specify the gradient vector direction of the edges detected by each kernel in (a).

10.14 The rectangle in the following binary image is of size $m \times n$ pixels.

(a)* What would the magnitude of the gradient of this image look like based on using the approximation in Eq. (10-26)? Assume that g_x and g_y are obtained using the Sobel kernels. Show all relevant different pixel values in the gradient image.

(b) With reference to Eq. (10-18) and Fig. 10.12,

sketch the histogram of edge *directions*. Be precise in labeling the height of each component of the histogram.

(c) What would the Laplacian of this image look like based on using Eq. (10-14)? Show all relevant different pixel values in the Laplacian image.

10.15 Suppose that an image $f(x,y)$ is convolved with a kernel of size $n \times n$ (with coefficients $1/n^2$) to produce a smoothed image $\bar{f}(x,y)$.

(a)* Derive an expression for edge strength (edge magnitude) as a function of n. Assume that n is odd and that the partial derivatives are computed using Eqs. (10-19) and (10-20).

(b) Show that the ratio of the maximum edge strength of the smoothed image to the maximum edge strength of the original image is $1/n$. In other words, edge strength is inversely proportional to the size of the smoothing kernel, as one would expect.

10.16 With reference to Eq. (10-29),

(a)* Show that the average value of the LoG operator, $\nabla^2 G(x,y)$, is zero.

(b) Show that the average value of any image convolved with this operator also is zero. (*Hint:* Consider solving this problem in the frequency domain, using the convolution theorem and the fact that the average value of a function is proportional to its Fourier transform evaluated at the origin.)

(c) Suppose that we: (1) used the kernel in Fig. 10.4(a) to approximate the Laplacian of a Gaussian, and (2) convolved this result with any image. What would be true in general of the values of the resulting image? Explain. (*Hint:* Take a look at Problem 3.32.)

10.17 Refer to Fig. 10.22(c).

(a) Explain why the edges form closed contours.

(b)* Does the zero-crossing method for finding edge location always result in closed contours? Explain.

10.18 One often finds in the literature a derivation of the Laplacian of a Gaussian (LoG) that starts with the expression

$$G(r) = e^{-r^2/2\sigma^2}$$

where $r^2 = x^2 + y^2$. The LoG is then derived by taking the second partial derivative with respect to r: $\nabla^2 G(r) = \partial^2 G(r)/\partial r^2$. Finally, $x^2 + y^2$ is substituted for r^2 to get the final (*incorrect*) result:

$$\nabla^2 G(x,y) = \left[\left(x^2 + y^2 - \sigma^2\right)/\sigma^4\right] \exp\left[-\left(x^2 + y^2\right)/2\sigma^2\right]$$

Derive this result and explain the reason for the difference between this expression and Eq. (10-29).

10.19 Do the following:

(a)* Derive Eq. (10-33).

(b) Let $k = \sigma_1/\sigma_2$ denote the standard deviation ratio discussed in connection with the DoG function, and express Eq. (10-33) in terms of k and σ_2.

10.20 In the following, assume that G and f are discrete arrays of size $n \times n$ and $M \times N$, respectively.

(a) Show that the 2-D convolution of the Gaussian function $G(x,y)$ in Eq. (10-27) with an image $f(x,y)$ can be expressed as a 1-D convolution along the rows (columns) of $f(x,y)$, followed by a 1-D convolution along the columns (rows) of the result. (*Hints:* See Section 3.4 regarding discrete convolution and separability).

(b)* Derive an expression for the computational advantage using the 1-D convolution approach in (a) as opposed to implementing the 2-D convolution directly. Assume that $G(x,y)$ is sampled to produce an array of size $n \times n$ and that $f(x,y)$ is of size $M \times N$. The computational advantage is the ratio of the number of multiplications required for 2-D convolution to the number required for 1-D convolution. (*Hint:* Review the subsection on separable kernels in Section 3.4.)

10.21 Do the following.

(a) Show that Steps 1 and 2 of the Marr-Hildreth algorithm can be implemented using four 1-D convolutions. (*Hints:* Refer to Problem 10.20(a) and express the Laplacian operator as the sum of two partial derivatives, given

by Eqs. (10-10) and (10-11), and implement each derivative using a 1-D kernel, as in Problem 10.12.)

(b) Derive an expression for the computational advantage of using the 1-D convolution approach in (a) as opposed to implementing the 2-D convolution directly. Assume that $G(x, y)$ is sampled to produce an array of size $n \times n$ and that $f(x, y)$ is of size $M \times N$. The computational advantage is the ratio of the number of multiplications required for 2-D convolution to the number required for 1-D convolution (see Problem 10.20).

10.22 Do the following.

(a)* Formulate Step 1 and the gradient magnitude image computation in Step 2 of the Canny algorithm using 1-D instead of 2-D convolutions.

(b) What is the computational advantage of using the 1-D convolution approach as opposed to implementing a 2-D convolution. Assume that the 2-D Gaussian filter in Step 1 is sampled into an array of size $n \times n$ and that the input image is of size $M \times N$. Express the computational advantage as the ratio of the number of multiplications required by each method.

10.23 With reference to the three vertical edge models and corresponding profiles in Fig. 10.8 provide sketches of the profiles that would result from each of the following methods. You may sketch the profiles manually.

(a)* Suppose that we compute the gradient magnitude of each of the three edge model images using the Sobel kernels. Sketch the horizontal intensity profiles of the three resulting gradient images.

(b) Sketch the horizontal intensity profiles that would result from using the 3×3 Laplacian kernel in Fig. 10.10.4(a).

(c)* Repeat (b) using only the first two steps of the Marr-Hildreth edge detector.

(d) Repeat (b) using the first two steps of the Canny edge detector. You may ignore the angle images.

(e) Sketch the horizontal profiles of the angle images resulting from using the Canny edge detector.

10.24 In Example 10.9, we used a smoothing kernel of size 19×19 to generate Fig. 10.26(c) and a kernel of size 13×13 to generate Fig. 10.26(d). What was the rationale that led to choosing these values? (*Hint:* Observe that both are Gaussian kernels, and refer to the discussion of lowpass Gaussian kernels in Section 3.5.)

10.25 Refer to the Hough transform in Section 10.2.

(a) Propose a general procedure for obtaining the normal representation of a line from its slope-intercept form, $y = ax + b$.

(b)* Find the normal representation of the line $y = -2x + 1$.

10.26 Refer to the Hough transform in Section 10.2.

(a)* Explain why the Hough mapping of the point labeled 1 in Fig. 10.30(a) is a straight line in Fig. 10.30(b).

(b)* Is this the only point that would produce that result? Explain.

(c) Explain the reflective adjacency relationship illustrated by, for example, the curve labeled Q in Fig. 10.30(b).

10.27 Show that the number of operations required to implement the accumulator-cell approach discussed in Section 10.2 is linear in n, the number of non-background points in the image plane (i.e., the xy-plane).

10.28 An important application of image segmentation is in processing images resulting from so-called *bubble chamber* events. These images arise from experiments in high-energy physics in which a beam of particles of known properties is directed onto a target of known nuclei. A typical event consists of incoming tracks, any one of which, upon a collision, branches out into secondary tracks of particles emanating from the point of collision. Propose a segmentation approach for detecting all tracks angled at any of the following six directions off the horizontal: $\pm 25°$, $\pm 50°$, and $\pm 75°$. The estimation error allowed in any of these six directions is $\pm 5°$. For a track to be valid it must be at least 100 pixels long and have no more than three gaps, each not exceeding 10 pixels. You may

so that they are binary and that all tracks are 1 thick, except at the point of collision from which they emanate. Your procedure should be able to differentiate between tracks that have the same direction but different origins. (*Hint:* Base your solution on the Hough transform.)

10.29* Restate the basic global thresholding algorithm in Section 10.3 so that it uses the histogram of an image instead of the image itself.

10.30* Prove that the basic global thresholding algorithm in Section 10.3 converges in a finite number of steps. (*Hint:* Use the histogram formulation from Problem 10.29.)

10.31 Give an explanation why the initial threshold in the basic global thresholding algorithm in Section 10.3 must be between the minimum and maximum values in the image. (*Hint:* Construct an example that shows the algorithm failing for a threshold value selected outside this range.)

10.32* Assume that the initial threshold in the basic global thresholding algorithm in Section 10.3 is selected as a value between the minimum and maximum intensity values in an image. Do you think the final value of the threshold at convergence depends on the specific initial value used? Explain. (You can use a simple image example to support your conclusion.)

10.33 You may assume in both of the following cases that the initial threshold is in the open interval $(0, L-1)$.

(a)* Show that if the histogram of an image is uniform over all possible intensity levels, the basic global thresholding algorithm converges to the average intensity of the image.

(b) Show that if the histogram of an image is bimodal, with identical modes that are symmetric about their means, then the basic global thresholding algorithm will converge to the point halfway between the means of the modes.

10.34 Refer to the basic global thresholding algorithm in Section 10.3. Assume that in a given problem, the histogram is bimodal with modes that are Gaussian curves of the form $A_1 \exp[-(z-m_1)^2/2\sigma_1^2]$ and $A_2 \exp[-(z-m_2)^2/2\sigma_2^2]$. Assume that m_1 is greater than m_2, and that the initial T is between the max and min image intensities. Give conditions (in terms of the parameters of these curves) for the following to be true when the algorithm converges:

(a)* The threshold is equal to $(m_1 + m_2)/2$.

(b)* The threshold is to the left of m_2.

(c) The threshold is in the interval given by the equation $(m_1 + m_2/2) < T < m_1$.

10.35 Do the following:

(a)* Show how the first line in Eq. (10-60) follows from Eqs. (10-55), (10-56), and (10-59).

(b) Show how the second line in Eq. (10-60) follows from the first.

10.36 Show that a maximum value for Eq. (10-63) always exists for k in the range $0 \le k \le L-1$.

10.37* With reference to Eq. (10-65), advance an argument that establishes that $0 \le \eta(k) \le 1$ for k in the range $0 \le k \le L-1$, where the minimum is achievable only by images with constant intensity, and the maximum occurs only for 2-valued images with values 0 and $(L-1)$.

10.38 Do the following:

(a)* Suppose that the intensities of a digital image $f(x,y)$ are in the range $[0, 1]$ and that a threshold, T, successfully segmented the image into objects and background. Show that the threshold $T' = 1 - T$ will successfully segment the negative of $f(x,y)$ into the same regions. The term negative is used here in the sense defined in Section 3.2.

(b) The intensity transformation function in (a) that maps an image into its negative is a linear function with negative slope. State the conditions that an arbitrary intensity transformation function must satisfy for the segmentability of the original image with respect to a threshold, T, to be preserved. What would be the value of the threshold after the intensity transformation?

10.39 The objects and background in the image below have a mean intensity of 170 and 60, respectively, on a [0, 255] scale. The image is corrupted by Gaussian noise with 0 mean and a standard deviation of 10 intensity levels. Propose a thresholding

method capable of a correct segmentation rate of 90% or higher. (Recall that 99.7% of the area of a Gaussian curve lies in a $\pm 3\sigma$ interval about the mean, where σ is the standard deviation.)

10.40 Refer to the intensity ramp image in Fig. 10.34(b) and the moving-average algorithm discussed in Section 10.3. Assume that the image is of size 500×700 pixels and that its minimum and maximum values are 0 and 1, where 0's are contained only in the first column.

(a)* What would be the result of segmenting this image with the moving-average algorithm using $b = 0$ and an arbitrary value for n. Explain what the segmented image would look like.

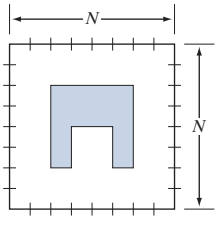
(b) Now reverse the direction of the ramp so that its leftmost value is 1 and the rightmost value is 0 and repeat (a).

(c) Repeat (a) but with $b = 1$ and $n = 2$.

(d) Repeat (a) but with $b = 1$ and $n = 100$.

10.41 Propose a region-growing algorithm to segment the image in Problem 10.39.

10.42* Segment the image shown by using the split and merge procedure discussed in Section 10.4. Let $Q(R_i)$ = TRUE if all pixels in R_i have the same intensity. Show the quadtree corresponding to your segmentation.



10.43 Consider the region of 1's resulting from the segmentation of the sparse regions in the image of the Cygnus Loop in Example 10.21. Propose a technique for using this region as a mask to isolate the three main components of the image: (1) background; (2) dense inner region; and (3) sparse outer region.

10.44 Let the pixels in the first row of a 3×3 image, like the one in Fig. 10.53(a), be labeled as 1, 2, 3, and the pixels in the second and third rows be labeled as 4, 5, 6 and 7, 8, 9, respectively. Let the intensity of these pixels be [90, 80, 30; 70, 5, 20; 80 20 30] where, for example, the intensity of pixel 2 is 80 and of pixel 4 it is 70. Compute the weights for the edges for the graph in Fig. 10.53(c), using the formula $w(i,j) = 30[1/(|I(n_i) - I(n_j)| + c)]$ explained in the text in connection with that figure (we scaled the formula by 30 to make the numerical results easier to interpret). Let $c = 0$ in this case.

10.45* Show how Eqs. (10-106) through (10-108) follow from Eq. (10-105).

10.46 Demonstrate the validity of Eq. (10-102).

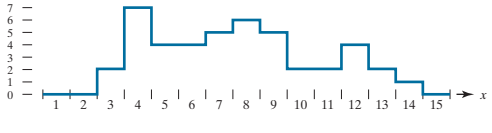
10.47 Refer to the discussion in Section 10.7.

(a)* Show that the elements of $C_n(M_i)$ and $T[n]$ are never replaced during execution of the watershed segmentation algorithm.

(b) Show that the number of elements in sets $C_n(M_i)$ and $T[n]$ either increases or remains the same as n increases.

10.48 You saw in Section 10.7 that the boundaries obtained using the watershed segmentation algorithm form closed loops (for example, see Figs. 10.59 and 10.61). Advance an argument that establishes whether or not closed boundaries *always* result from application of this algorithm.

10.49* Give a step-by-step implementation of the dam-building procedure for the one-dimensional intensity cross section shown below. Show a drawing of the cross section at each step, showing "water" levels and dams constructed.



10.50 What would the negative ADI image shown in Fig. 10.62(c) look like if we tested against T (instead of testing against $-T$) in Eq. (10-117)?

10.51 Are the following statements true or false? Explain the reason for your answer in each.

(a)* The nonzero entries in the absolute ADI continue to grow in dimension, provided that the object is moving.

(b) The nonzero entries in the positive ADI always occupy the same area, regardless of the motion undergone by the object.

(c) The nonzero entries in the negative ADI continue to grow in dimension, provided that the object is moving.

10.52 Suppose that in Example 10.29 motion along the x-axis is set to zero. The object now moves only along the y-axis at 1 pixel per frame for 32 frames and then (instantaneously) reverses direction and moves in exactly the opposite direction for another 32 frames. What would Figs. 10.66(a) and (b) look like under these conditions?

10.53* Advance an argument that demonstrates that when the signs of S_{1x} and S_{2x} in Eqs. (10-125) and (10-126) are the same, velocity component V_1 is positive.

10.54 An automated pharmaceutical plant uses image processing to measure the shapes of medication tablets for the purpose of quality control. The segmentation stage of the system is based on Otsu's method. The speed of the inspection lines is so high that a very high rate flash illumination is required to "stop" motion. When new, the illumination lamps project a uniform pattern of light. However, as the lamps age, the illumination pattern deteriorates as a function of time and spatial coordinates according to the equation

$$i(x,y) = A(t) - t^2 e^{-[(x - M/2)^2 + (y - N/2)^2]}$$

where $(M/2, N/2)$ is the center of the viewing area and t is time measured in increments of months. The lamps are still experimental and the behavior of $A(t)$ is not fully understood by the manufacturer. All that is known is that, during the life of the lamps, $A(t)$ is always greater than the negative component in the preceding equation because illumination cannot be negative. It has been observed that Otsu's algorithm works well when the lamps are new, and their pattern of illumination is nearly constant over the entire image. However, segmentation performance deteriorates with time. Being experimental, the lamps are exceptionally expensive, so you are employed as a consultant to help solve the problem using digital image processing techniques to compensate for the changes in illumination, and thus extend the useful life of the lamps. You are given flexibility to install any special markers or other visual cues in the viewing area of the imaging cameras. Propose a solution in sufficient detail that the engineering plant manager can understand your approach. (*Hint:* Review the image model discussed in Section 2.3 and consider using one or more targets of known reflectivity.)

10.55 The speed of a bullet in flight is to be estimated by using high-speed imaging techniques. The method of choice involves the use of a CCD camera and flash that exposes the scene for K seconds. The bullet is 2.5 cm long, 1 cm wide, and its range of speed is 750 ± 250 m/s. The camera optics produce an image in which the bullet occupies 10% of the horizontal resolution of a 256×256 digital image.

(a)* Determine the maximum value of K that will guarantee that the blur from motion does not exceed 1 pixel.

(b) Determine the minimum number of frames per second that would have to be acquired in order to guarantee that at least two complete images of the bullet are obtained during its path through the field of view of the camera.

(c)* Propose a segmentation procedure for automatically extracting the bullet from a sequence of frames.

(d) Propose a method for automatically determining the speed of the bullet.