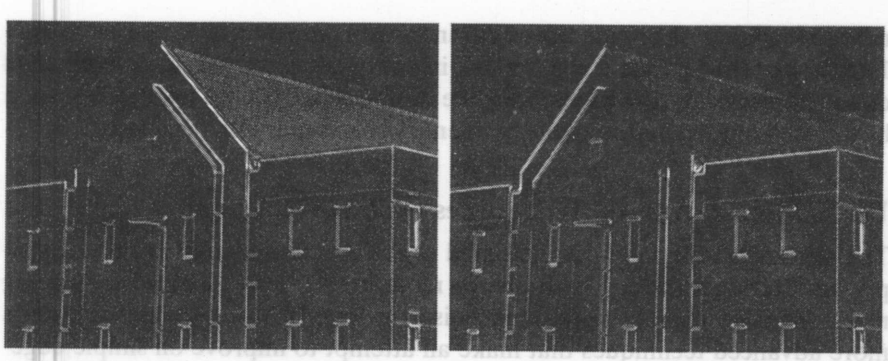


a b

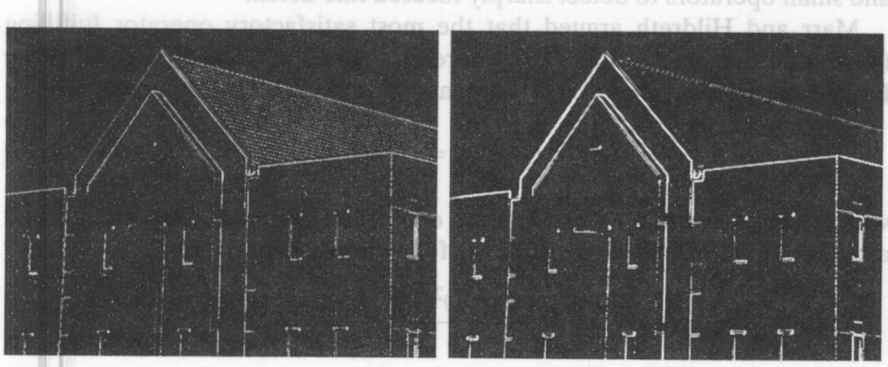
FIGURE 10.19 Diagonal edge detection. (a) Result of using the mask in Fig. 10.15(c). (b) Result of using the mask in Fig. 10.15(d). The input image in both cases was Fig. 10.18(a).

Combining the gradient with thresholding

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

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

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



a b

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

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

10.2.6 More Advanced Techniques for Edge Detection

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

The Marr-Hildreth edge detector

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

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

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

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

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

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

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

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

Collecting terms gives the final expression:

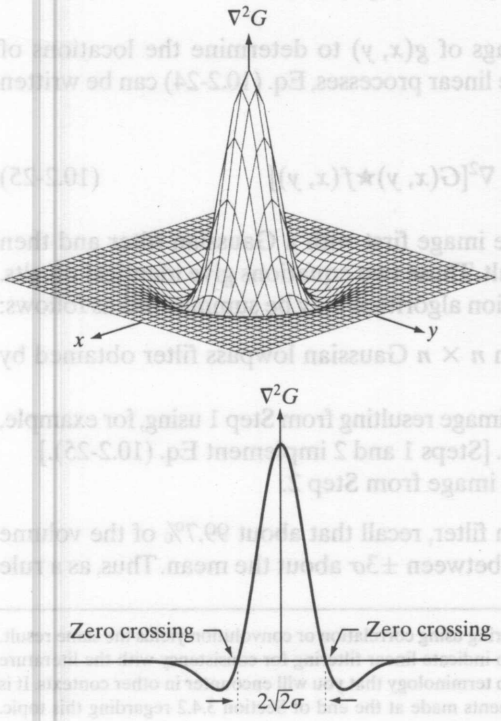
$$\nabla^2 G(x, y) = \left[\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (10.2\text{-}23)$$

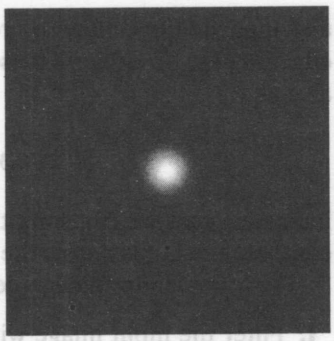
This expression is called the *Laplacian of a Gaussian* (LoG).

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

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

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





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

a b
c d

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

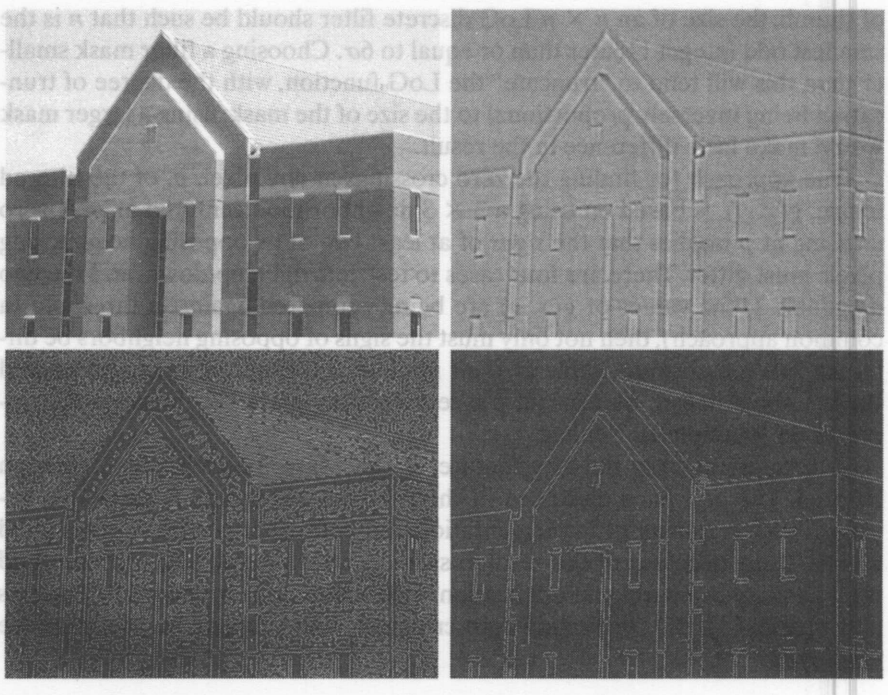
Zero crossings are the key feature of the Marr-Hildreth edge-detection method. The approach discussed in the previous paragraph is attractive because of its simplicity of implementation and because it generally gives good results. If the accuracy of the zero-crossing locations found using this method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

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

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

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

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



a b
c d

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

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

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

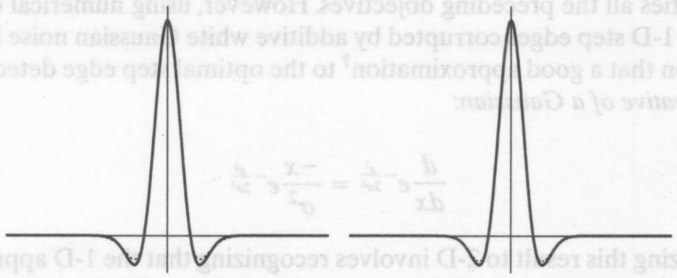
The difference of Gaussians is a highpass filter, as discussed in Section 4.7.4.

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

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

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

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



a b

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

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

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

The Canny edge detector

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

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

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

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

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

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.

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

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

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

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

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

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

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

and

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

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

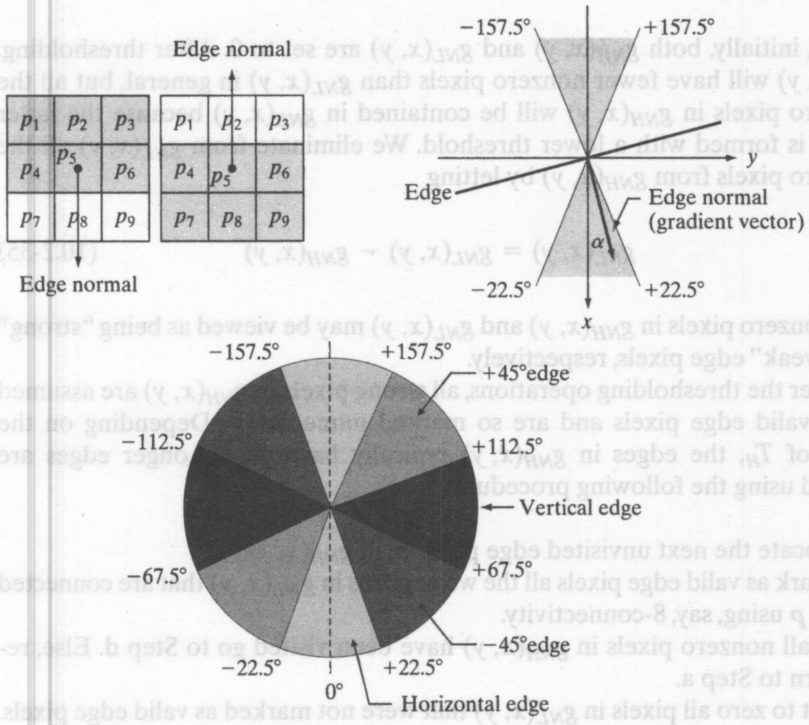
Because it is generated using the gradient, $M(x, y)$ typically contains wide ridges around local maxima (recall the discussion in Section 10.2.1 regarding edges obtained using the gradient). The next step is to thin those ridges. One approach is to use *nonmaxima suppression*. This can be done in several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

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

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

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

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



a b
c

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

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

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

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

We can visualize the thresholding operation as creating two additional images

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

and

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

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

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

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

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

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

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

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

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

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

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

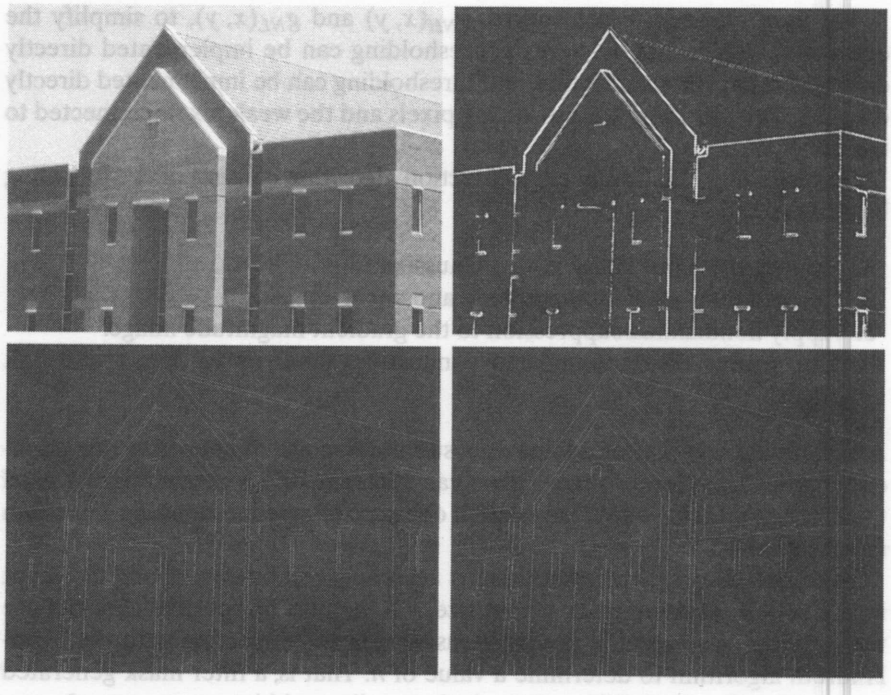
As mentioned earlier, smoothing is accomplished by convolving the input image with a Gaussian mask whose size, $n \times n$, must be specified. We can use the approach discussed in the previous section in connection with the Marr-Hildreth algorithm to determine a value of n. That is, a filter mask generated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

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

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

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

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



a b
c d

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

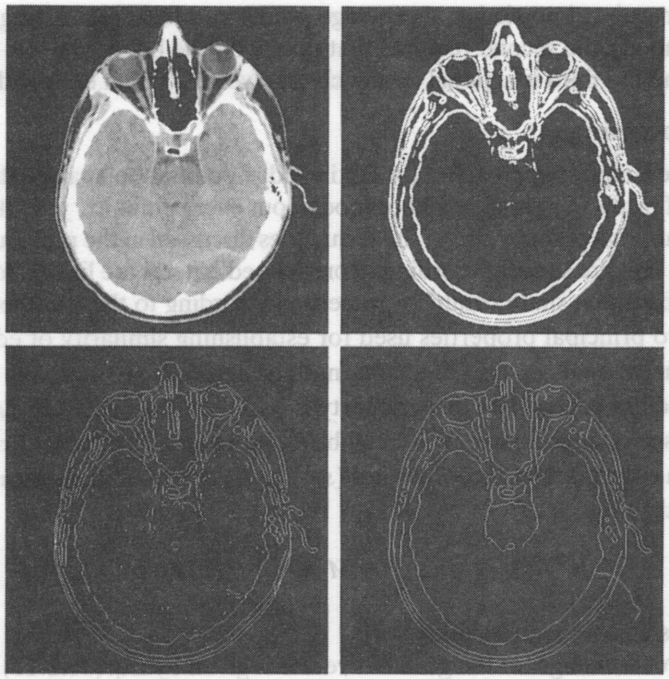
The threshold values given here should be considered only in relative terms. Implementation of most algorithms involves various scaling steps, such as scaling the range of values of the input image to the range [0, 1]. Different scaling schemes obviously would require different values of thresholds from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

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

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

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

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



a b
c d

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

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

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

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

10.2.7 Edge Linking and Boundary Detection

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

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

Local processing

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

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

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

where E is a positive threshold.

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

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

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

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

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

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

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

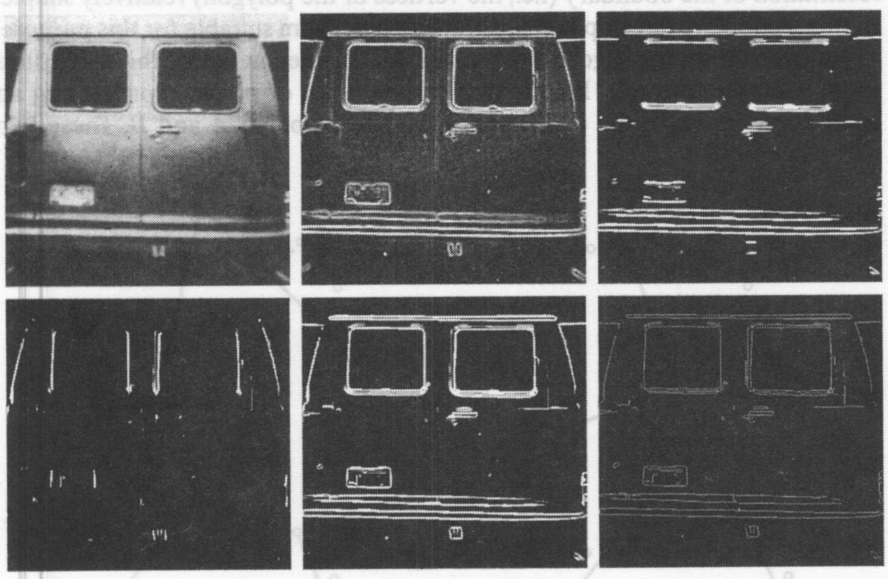
where T_M is a threshold, A is a specified angle direction, and $\pm T_A$ defines a "band" of acceptable directions about A.

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

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

EXAMPLE 10.10: Edge linking using local processing.

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



a b c
d e f

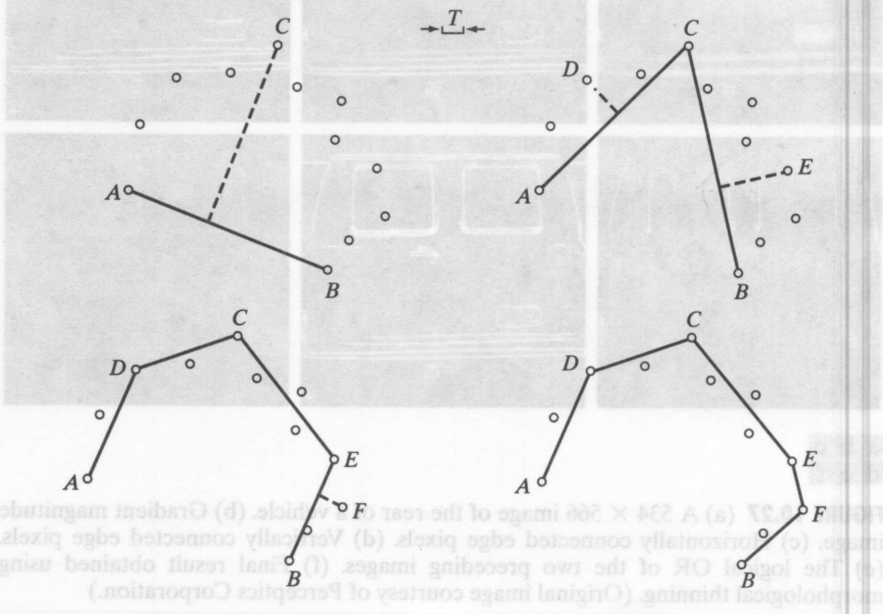
FIGURE 10.27 (a) A 534 × 566 image of the rear of a vehicle. (b) Gradient magnitude image. (c) Horizontally connected edge pixels. (d) Vertically connected edge pixels. (e) The logical OR of the two preceding images. (f) Final result obtained using morphological thinning. (Original image courtesy of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

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

Regional processing

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

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



a b
c d

FIGURE 10.28 Illustration of the iterative polygonal fit algorithm.

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

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

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

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

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

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

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

EXAMPLE 10.11: Edge linking using a polygonal approximation.

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

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

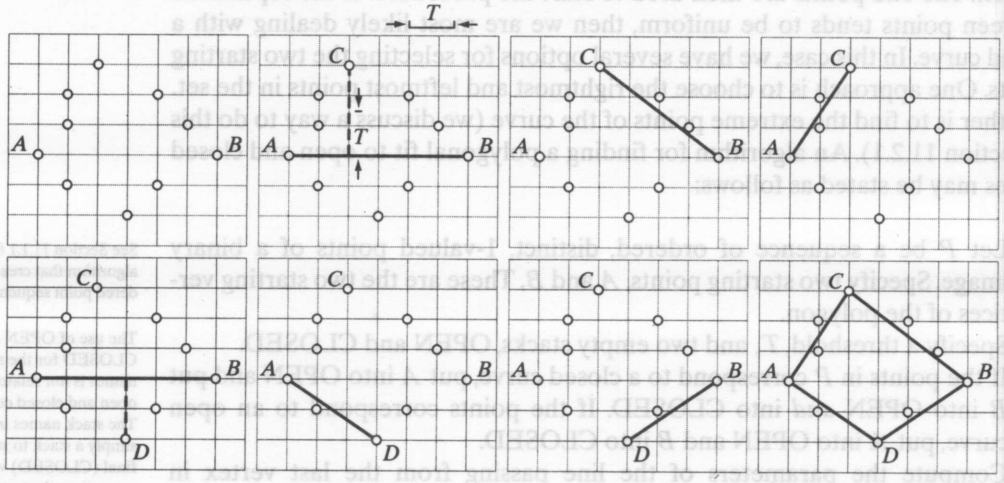


FIGURE 10.29 (a) A set of points in a clockwise path (the points labeled A and B were chosen as the starting vertices). (b) The distance from point C to the line passing through A and B is the largest of all the points between A and B and also passed the threshold test, so C is a new vertex. (d)–(g)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

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

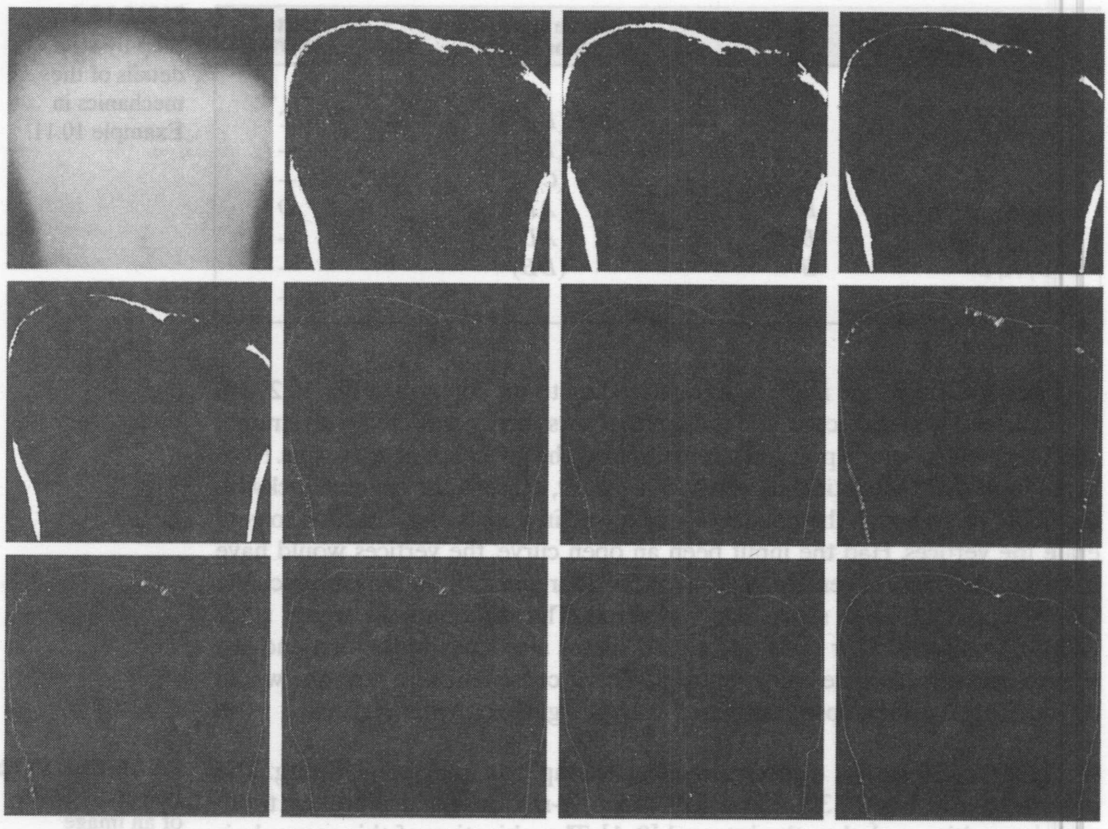
TABLE 10.1 Step-by-step details of the mechanics in Example 10.11.

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

EXAMPLE 10.12: Polygonal fitting of an image boundary.

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

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



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

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

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

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

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

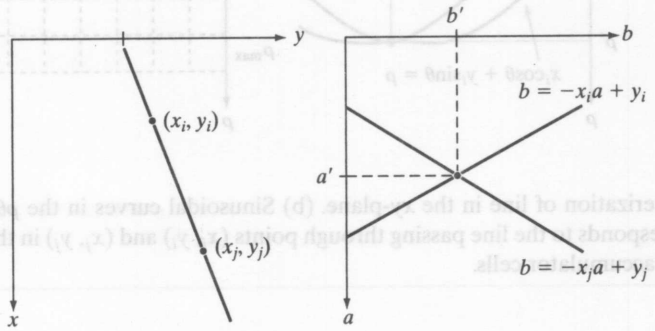
Global processing using the Hough transform

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

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

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

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



a b

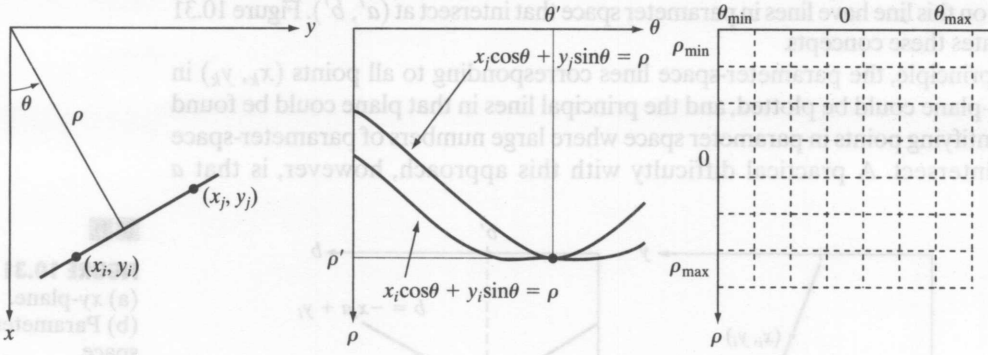
FIGURE 10.31
(a) xy-plane. (b) Parameter space.

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

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

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

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



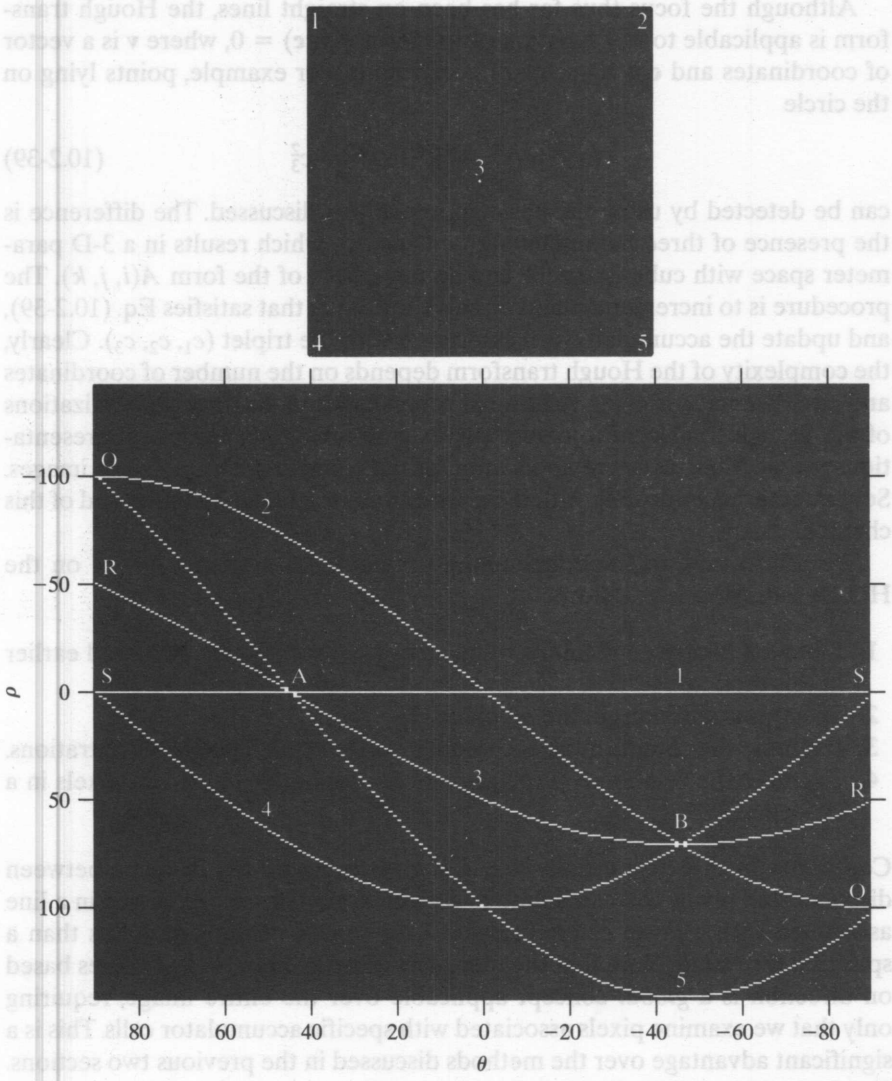
a b c

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

EXAMPLE 10.13: An illustration of basic Hough transform properties.

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

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



a
b

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

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

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

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

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

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

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

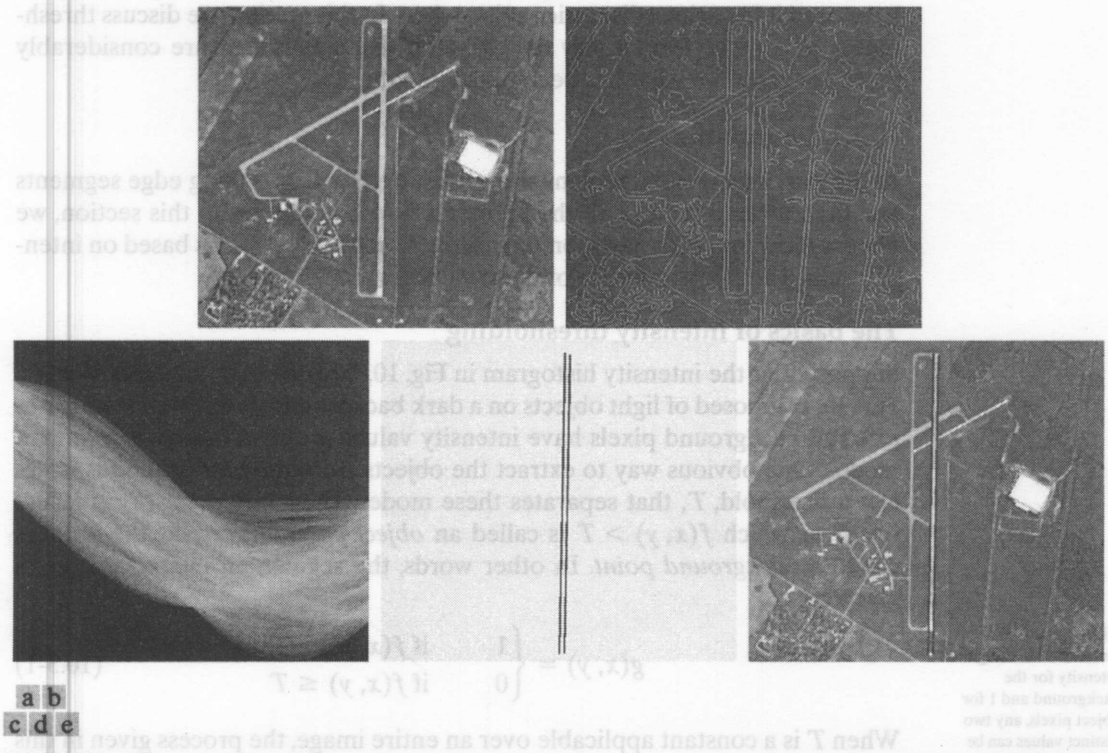
Continuity in this case usually is based on computing the distance between disconnected pixels corresponding to a given accumulator cell. A gap in a line associated with a given cell is bridged if the length of the gap is less than a specified threshold. Note that the mere fact of being able to group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

EXAMPLE 10.14: Using the Hough transform for edge linking.

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

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

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



a b
c d e

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

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

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

10.3 Thresholding

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

10.3.1 Foundation

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

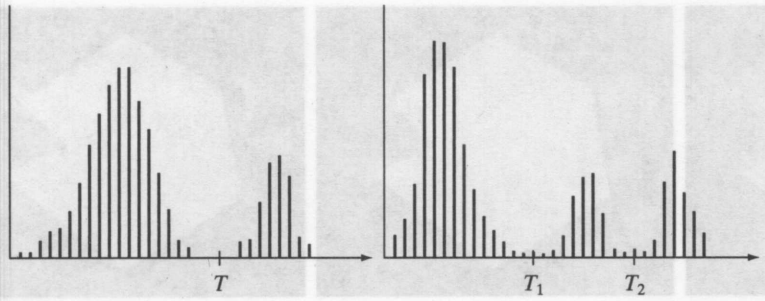
The basics of intensity thresholding

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

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

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

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



a b

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

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

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

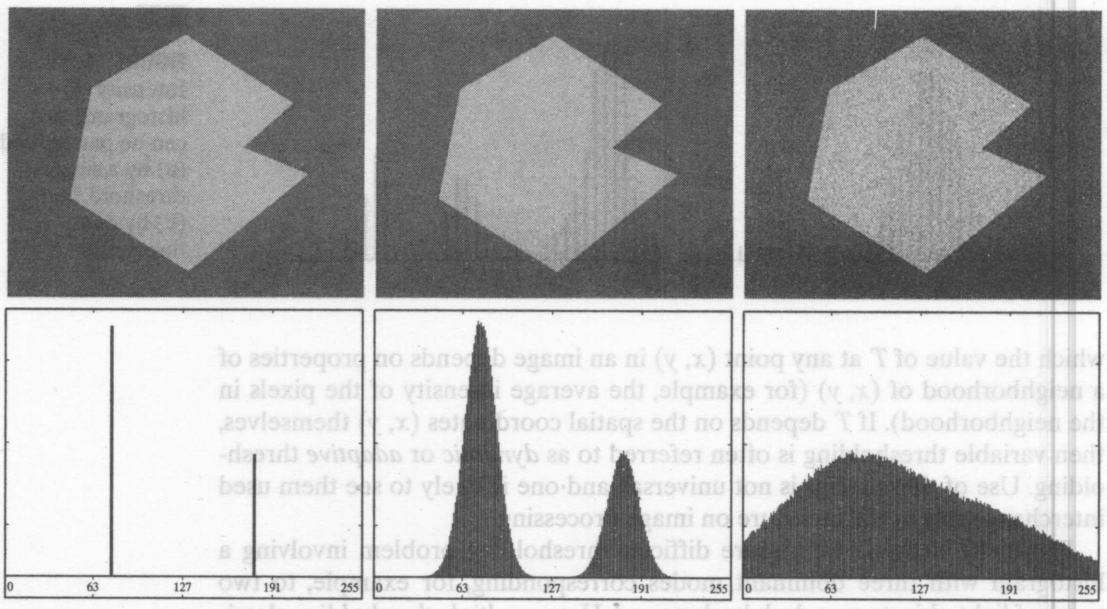
$$g(x, y) = \begin{cases} a & \text{if } f(x, y) > T_2 \\ b & \text{if } T_1 < f(x, y) \leq T_2 \\ c & \text{if } f(x, y) \leq T_1 \end{cases} \qquad (10.3\text{-}2)$$

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

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

The role of noise in image thresholding

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



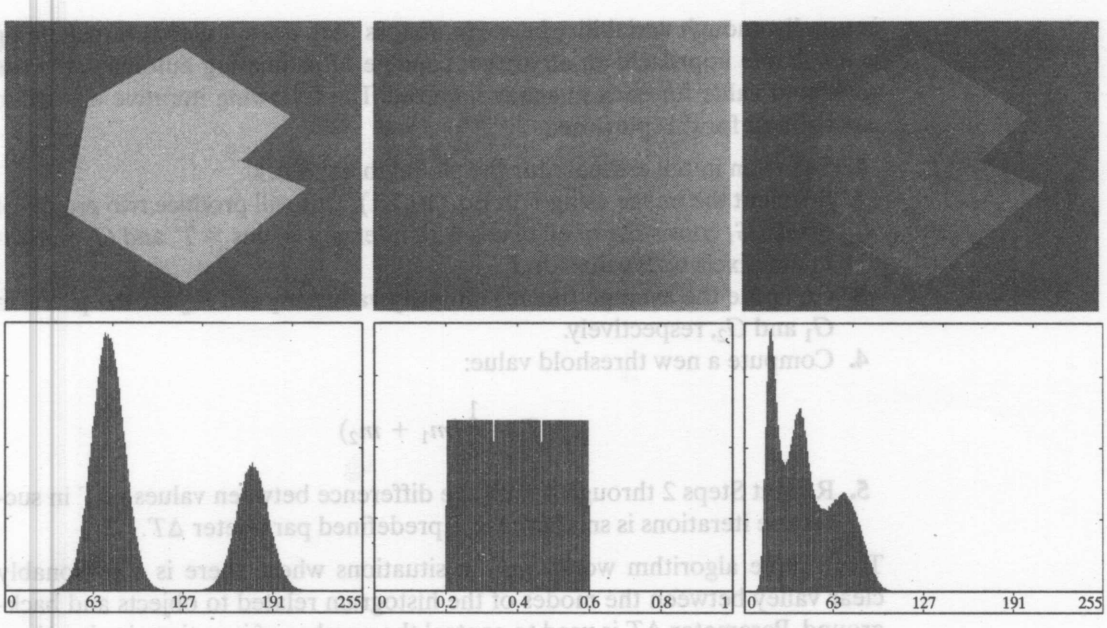
a b c
d e f

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

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

The role of illumination and reflectance

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



a b c
d e f

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

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

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

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

10.3.2 Basic Global Thresholding

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

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

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

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

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

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

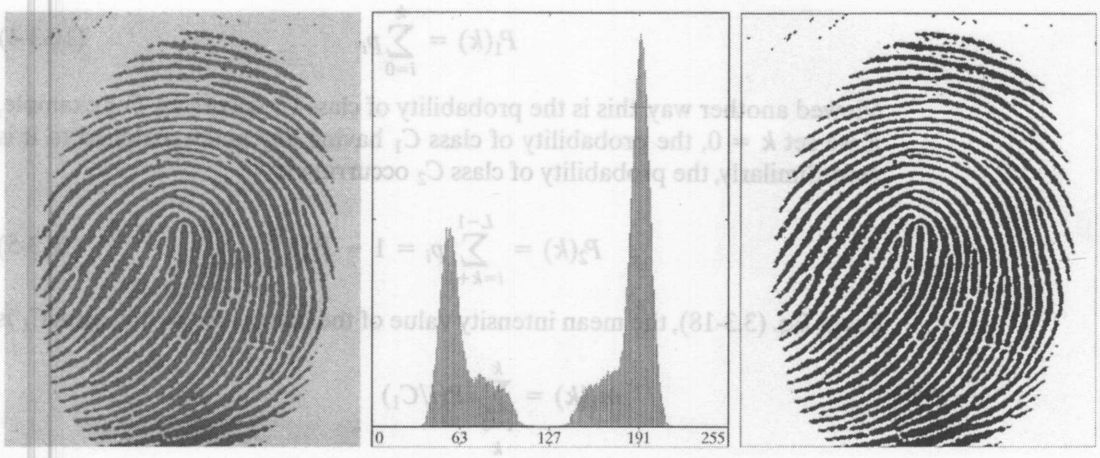
EXAMPLE 10.15: Global thresholding.

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

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

10.3.3 Optimum Global Thresholding Using Otsu's Method

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



a b c

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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