ROBOTICS:
Control, Sensing, Vision, and Intelligence

K. S. Fu
School of Electrical Engineering
Purdue University

R. C. Gonzalez
Department of Electrical Engineering
University of Tennessee
and
Perceptics Corporation
Knoxville, Tennessee

C. S. G. Lee
School of Electrical Engineering
Purdue University

McGraw-Hill Book Company
New York St. Louis San Francisco Auckland Bogotá
Hamburg Johannesburg London Madrid Mexico Milan Montreal New Delhi
Panama Paris São Paulo Singapore Sydney Tokyo Toronto

CHAPTER
EIGHT

HIGHER-LEVEL VISION

The artist is one who gives
form to difficult visions.
Theodore Gill

8.1 INTRODUCTION

For the purpose of categorizing the various techniques and approaches used in machine vision, we introduced in Sec. 7.1 three broad subdivisions: low-, medium-, and high-level vision. Low-level vision deals with basic sensing and preprocessing, topics which were covered in some detail in Chap. 7. We may view the material in that chapter as being instrumental in providing image and other relevant information that is in a form suitable for subsequent intelligent visual processing.

Although the concept of "intelligence" is somewhat vague, particularly when one is referring to a machine, it is not difficult to conceptualize the type of behavior that we may, however grudgingly, characterize as intelligent. Several characteristics come immediately to mind: (1) the ability to extract pertinent information from a background of irrelevant details, (2) the capability to learn from examples and to generalize this knowledge so that it will apply in new and different circumstances, (3) the ability to infer facts from incomplete information, and (4) the capability to generate self-motivated goals, and to formulate plans for meeting these goals.

While it is possible to design and implement a vision system with these characteristics in a *limited* environment, we do not yet know how to endow it with a range and depth of adaptive performance that comes even close to emulating human vision. Although research in biological systems is continually uncovering new and promising concepts, the state of the art in machine vision is for the most part based on analytical formulations tailored to meet specific tasks. The time frame in which we may have machines that approach human visual and other sensory capabilities is open to speculation. It is of interest to note, however, that imitating nature is not the only solution to this problem. The reader is undoubtedly familiar with early experimental airplanes equipped with flapping wings and other birdlike features. Given that the objective is to fly between two points, our present solution is quite different from the examples provided by nature. In terms of speed and achievable altitude, this solution exceeds the capabilities of these examples by a wide margin.

As indicated in Sec. 7.1, medium-level vision deals with topics in segmentation, description, and recognition of individual objects. It will be seen in the following sections that these topics encompass a variety of approaches that are well-founded on analytical concepts. High-level vision deals with issues such as those discussed in the preceding paragraph. Our knowledge of these areas and their relationship to low- and medium-level vision is significantly more vague and speculative, leading to the formulation of constraints and idealizations intended to simplify the complexity of this task.

The material discussed in this chapter introduces the reader to a broad range of topics in state-of-the-art machine vision, with a strong orientation toward techniques that are suitable for robotic vision. The material is subdivided into four principal areas. We begin the discussion with a detailed treatment of segmentation. This is followed by a discussion of object description techniques. We then discuss the principal approaches used in the recognition stage of a vision system. We conclude the chapter with a discussion of issues on the interpretation of visual information.

8.2 SEGMENTATION

Segmentation is the process that subdivides a sensed scene into its constituent parts or objects. Segmentation is one of the most important elements of an automated vision system because it is at this stage of processing that objects are extracted from a scene for subsequent recognition and analysis. Segmentation algorithms are generally based on one of two basic principles: discontinuity and similarity. The principal approach in the first category is based on edge detection; the principal approaches in the second category are based on thresholding and region growing. These concepts are applicable to both static and dynamic (time-varying) scenes. In the latter case, however, motion can often be used as a powerful cue to improve the performance of segmentation algorithms.

8.2.1 Edge Linking and Boundary Detection

The techniques discussed in Sec. 7.6.4 detect intensity discontinuities. Ideally, these techniques should yield only pixels lying on the boundary between objects and the background. In practice, this set of pixels seldom characterizes a boundary completely because of noise, breaks in the boundary due to nonuniform illumination, and other effects that introduce spurious intensity discontinuities. Thus, edge detection algorithms are typically followed by linking and other boundary detection procedures designed to assemble edge pixels into a meaningful set of object boundaries. In the following discussion we consider several techniques suited for this purpose.

Local Analysis. One of the simplest approaches for linking edge points is to analyze the characteristics of pixels in a small neighborhood (e.g., 3×3 or 5×5)

about every point (x, y) in an image that has undergone an edge detection process. All points that are similar (as defined below) are linked, thus forming a boundary of pixels that share some common properties.

There are two principal properties used for establishing similarity of edge pixels in this kind of analysis: (1) the strength of the response of the gradient operator used to produce the edge pixel, and (2) the direction of the gradient. The first property is given by the value of $G[f(x, y)]$, as defined in Eqs. (7.6-38) or (7.6-39). Thus, we say that an edge pixel with coordinates (x', y') and in the predefined neighborhood of (x, y) is similar in magnitude to the pixel at (x, y) if

$$|G[f(x, y)] - G[f(x', y')]| \leqslant T \tag{8.2-1}$$

where T is a threshold.

The direction of the gradient may be established from the angle of the gradient vector given in Eq. (7.6-37). That is,

$$\theta = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{8.2-2}$$

where θ is the angle (measured with respect to the x axis) along which the rate of change has the greatest magnitude, as indicated in Sec. 7.6.4. Then, we say that an edge pixel at (x', y') in the predefined neighborhood of (x, y) has an angle similar to the pixel at (x, y) if

$$|\theta - \theta'| < A \tag{8.2-3}$$

where A is an angle threshold. It is noted that the direction of the edge at (x, y) is, in reality, perpendicular to the direction of the gradient vector at that point. However, for the purpose of comparing directions, Eq. (8.2-3) yields equivalent results.

Based on the foregoing concepts, we link a point in the predefined neighborhood of (x, y) to the pixel at (x, y) if both the magnitude and direction criteria are satisfied. This process is repeated for every location in the image, keeping a record of linked points as the center of the neighborhood is moved from pixel to pixel. A simple bookkeeping procedure is to assign a different gray level to each set of linked edge pixels.

Example: As an illustration of the foregoing procedure, consider Fig. 8.1*a*, which shows an image of the rear of a vehicle. The objective is to find rectangles whose sizes makes them suitable license plate candidates. The formation of these rectangles can be accomplished by detecting strong horizontal and vertical edges. Figure 8.1*b* and *c* shows the horizontal and vertical components of the Sobel operators discussed in Sec. 7.6.4. Finally, Fig. 8.1*d* shows the results of linking all points which, simultaneously, had a gradient value greater than 25 and whose gradient directions did not differ by more

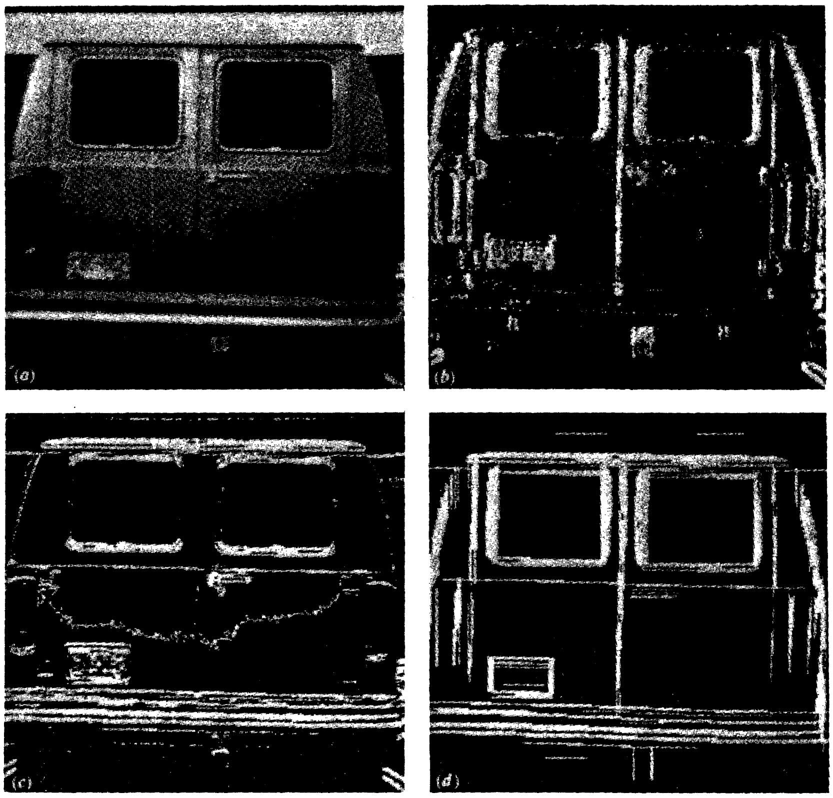


Figure 8.1 (*a*) Input image. (*b*) Horizontal component of the gradient. (*c*) Vertical component of the gradient. (*d*) Result of edge linking. (Courtesy of Perceptics Corporation.)

than 15°. The horizontal lines were formed by sequentially applying these criteria to every row of Fig. 8.1*c*, while a sequential column scan of Fig. 8.1*b* yielded the vertical lines. Further processing consisted of linking edge segments separated by small breaks and deleting isolated short segments. □

Global Analysis via the Hough Transform. In this section we consider the linking of boundary points by determining whether or not they lie on a curve of specified shape. Suppose initially that, given n points in the xy plane of an image, we wish to find subsets that lie on straight lines. One possible solution is to first find all lines determined by every pair of points and then find all subsets of points that are close to particular lines. The problem with this procedure is that it 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 computationally prohibitive in all but the most trivial applications.

This problem may be viewed in a different way using an approach proposed by Hough [1962] and commonly referred to as the *Hough transform*. Consider a point (x_i, y_i) and the general equation of a straight line in slope-intercept form, $y_i = ax_i + b$. There is an infinite number of lines that 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, if we write this equation as $b = -x_i a + y_i$, and consider the ab plane (also called *parameter space*), then we have the equation of a *single* line for a fixed pair (x_i, y_i). Furthermore, a second point (x_j, y_j) will also have a line in parameter space associated with it, and this line will intersect the line associated with (x_i, y_i) at (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 points contained on this line will have lines in parameter space which intercept at (a', b'). These concepts are illustrated in Fig. 8.2.

The computational attractiveness of the Hough transform arises from subdividing the parameter space into so-called *accumulator cells*, as illustrated in Fig. 8.3, where $(a_{\max}, a_{\min})$ and $(b_{\max}, b_{\min})$ are the expected ranges of slope and intercept values. Accumulator cell $A(i, j)$ corresponds to the square associated with parameter space coordinates (a_i, b_j). Initially, these cells are set to zero. Then, for every point (x_k, y_k) in the image plane, we let the parameter a equal each of the allowed subdivision values on the a axis and solve for the corresponding b using the equation $b = -x_k a + y_k$. The resulting b's are then rounded off to the nearest allowed value in the b axis. If a choice of a_p results in solution b_q, we let $A(p, q) = A(p, q) + 1$. At the end of this procedure, a value of M in cell $A(i, j)$ corresponds to M points in the xy plane lying on the line $y = a_i x + b_j$. The accuracy of the colinearity of these points is established by the number of subdivisions in the ab plane.

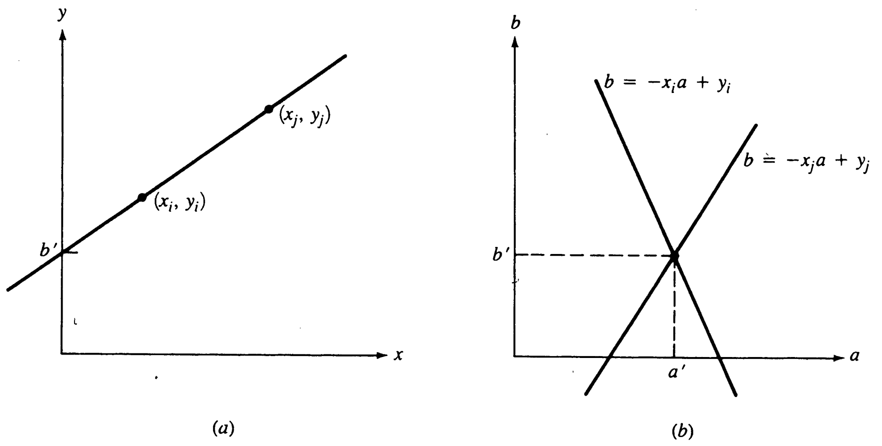


Figure 8.2 (*a*) *xy* Plane. (*b*) Parameter space.

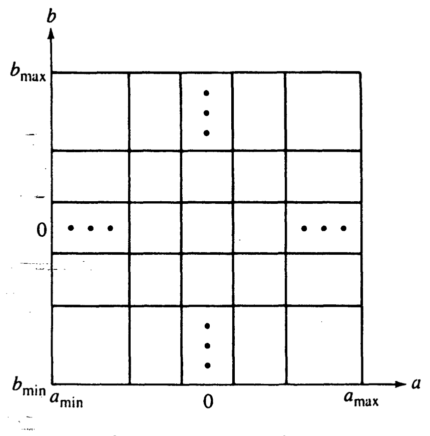


Figure 8.3 Quantization of the parameter plane into cells for use in the Hough transform.

It is noted that if we subdivide the a axis into K increments, then for every point (x_k, y_k) we obtain K values of b corresponding to the K possible values of a. Since there are n image points, this involves nK computations. Thus, the procedure just discussed is *linear* in n, and the product nK does not approach the number of computations discussed at the beginning of this section unless K approaches or exceeds n.

A problem with using the equation $y = ax + b$ to represent a line is that both the slope and intercept approach infinity as the line approaches a vertical position. One way around this difficulty is to use the normal representation of a line, given by

$$x \cos\theta + y \sin\theta = \rho \tag{8.2-4}$$

The meaning of the parameters used in Eq. (8.2-4) is illustrated in Fig. 8.4*a*. The use of this representation in constructing a table of accumulators is identical to the method discussed above for the slope-intercept representation; the only difference is that, instead of straight lines, we now have sinusoidal curves in the $\theta\rho$ plane. As before, M colinear points lying on a line $x \cos\theta_i + y \sin\theta_i = \rho_j$ will yield M sinusoidal curves which intercept at (θ_i, ρ_j) in the parameter space. When we use the method of incrementing θ and solving for the corresponding ρ, the procedure will yield M entries in accumulator $A(i, j)$ associated with the cell determined by (θ_i, ρ_j). The subdivision of the parameter space is illustrated in Fig. 8.4*b*.

Example: An illustration of using the Hough transform based on Eq. (8.2-4) is shown in Fig. 8.5. Figure 8.5*a* shows an image of an industrial piece, Fig. 8.5*b* is the gradient image, and Fig. 8.5*c* shows the $\theta\rho$ plane displayed as an image in which brightness level is proportional to the number of counts in the accumulators. The abscissa in this image corresponds to θ and the ordinate to ρ, with ranges $\pm 90°$ and $\pm\rho_{max}$, respectively. In this case, ρ_{max} was set

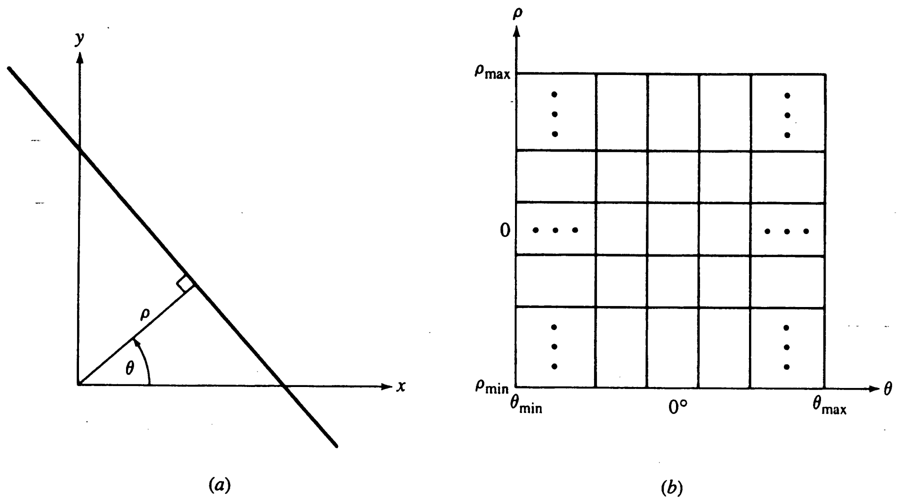


Figure 8.4 (*a*) Normal representation of a line. (*b*) Quantization of the $\theta\rho$ plane into cells.

equal to the distance from corner to corner in the original image. The center of the square in Fig. 8.5*c* thus corresponds to $\theta = 0°$ and $\rho = 0$. It is of interest to note the bright spots (high accumulator counts) near $0°$ corresponding to the vertical lines, and near $\pm 90°$ corresponding to the horizontal lines in Fig. 8.5*b*. The lines detected by this method are shown in Fig. 8.5*d* superimposed on the original image. The discrepancy is due to the quantization error of θ and ρ in the parameter space. □

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

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2 \tag{8.2-5}$$

can easily be detected by using the approach discussed above. The basic difference is that we now have three parameters, c_1, c_2, and c_3, which result in a three-dimensional parameter space with cubelike 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. (8.2-5), and update the accumulator corresponding to the cell associated with the triple (c_1, c_2, c_3). Clearly, the complexity of the Hough transform is strongly dependent on the number of coordinates and coefficients in a given functional representation.

Before leaving this section, we point out that further generalizations of the Hough transform to detect curves with no simple analytic representations are possible. These concepts, which are extensions of the material presented above, are treated in detail by Ballard [1981].

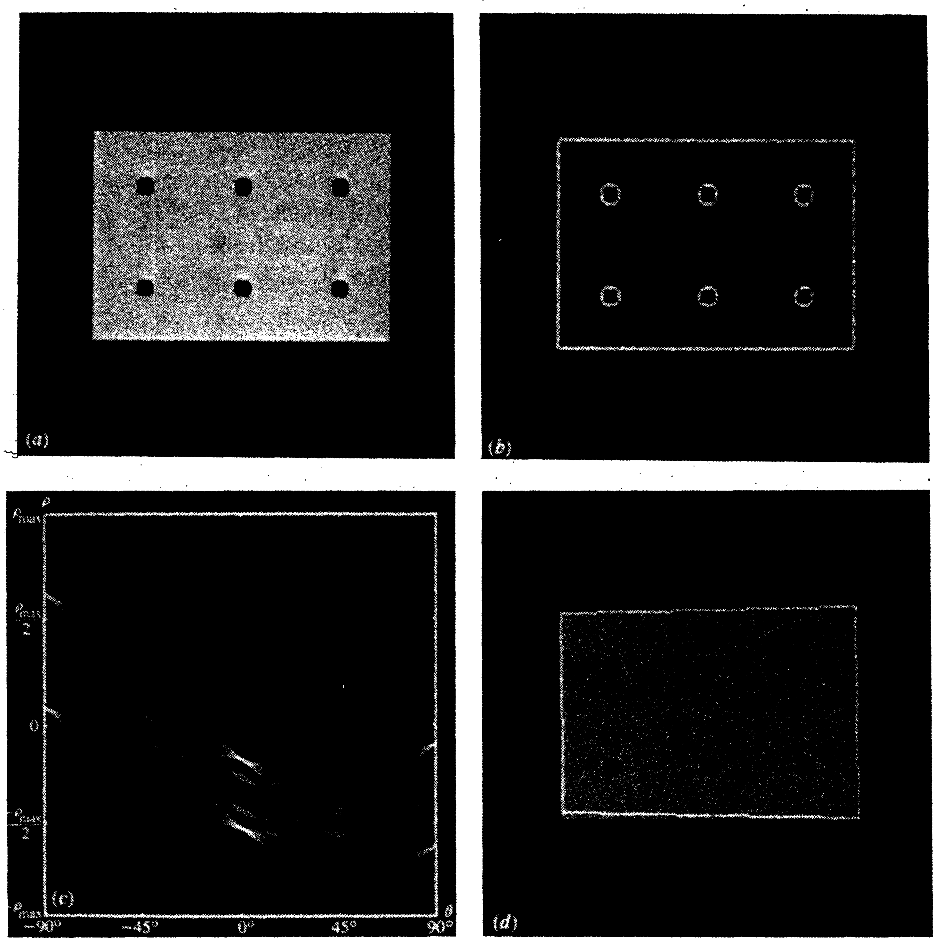


Figure 8.5 (*a*) Image of a work piece. (*b*) Gradient image. (*c*) Hough transform table. (*d*) Detected lines superimposed on the original image. (Courtesy of D. Cate, Texas Instruments, Inc.)

Global Analysis via Graph-Theoretic Techniques. The method discussed in the previous section is based on having a set of edge points obtained typically through a gradient operation. Since the gradient is a derivative, it enhances sharp variations in intensity and, therefore, is seldom suitable as a preprocessing step in situations characterized by high noise content. We now discuss a global approach based on representing edge segments in the form of a graph structure and searching the graph for low-cost paths which correspond to significant edges. This representation provides a rugged approach which performs well in the presence of noise. As might be expected, the procedure is considerably more complicated and requires more processing time than the methods discussed thus far.

We begin the development with some basic definitions. A *graph* $G = (N, A)$ is a finite, nonempty set of nodes N, together with a set A of unordered pairs of distinct elements of N. Each pair (n_i, n_j) of A is called an *arc*. A graph in which its arcs are directed is called a *directed graph*. If an arc is directed from node n_i to node n_j, then n_j is said to be a *successor* of its *parent* node n_i. The process of identifying the successors of a node is called *expansion* of the node. In each graph we will define levels, such that level 0 consists of a single node, called the *start* node, and the nodes in the last level are called *goal* nodes. A *cost* $c(n_i, n_j)$ can be associated with every arc (n_i, n_j). A sequence of nodes $n_1, n_2, \ldots, n_k$ with each node n_i being a successor of node n_{i-1} is called a *path* from n_1 to n_k, and the cost of the path is given by

$$c = \sum_{i=2}^{k} c(n_{i-1}, n_i) \tag{8.2-6}$$

Finally, we define an *edge element* as the boundary between two pixels p and q, such that p and q are 4-neighbors, as illustrated in Fig. 8.6. In this context, an *edge* is a sequence of edge elements.

In order to illustrate how the foregoing concepts apply to edge detection, consider the 3 × 3 image shown in Fig. 8.7, where the outer numbers are pixel coordinates and the numbers in parentheses represent intensity. With each edge element defined by pixels p and q we associate the cost

$$c(p, q) = H - [f(p) - f(q)] \tag{8.2-7}$$

where H is the highest intensity value in the image (7 in this example), $f(p)$ is the intensity value of p, and $f(q)$ is the intensity value of q. As indicated above, p and q are 4-neighbors.

The graph for this problem is shown in Fig. 8.8. Each node in this graph corresponds to an edge element, and an arc exists between two nodes if the two corresponding edge elements taken in succession can be part of an edge. The cost of each edge element, computed using Eq. (8.2-7), is shown by the arc leading into it, and goal nodes are shown in double rectangles. Each path between the start node and a goal node is a possible edge. For simplicity, it has been assumed that the edge starts in the top row and terminates in the last row, so that the first ele-

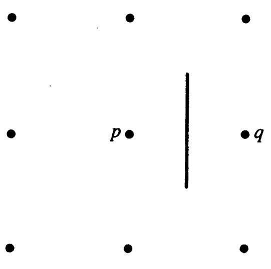


Figure 8.6 Edge element between pixels p and q.

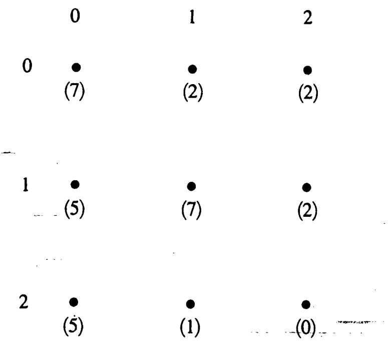


Figure 8.7 A 3 × 3 image.

ment of an edge can only be [(0, 0), (0, 1)] or [(0, 1), (0, 2)] and the last element [(2, 0), (2, 1)] or [(2, 1), (2, 2)]. The minimum-cost path, computed using Eq. (8.2-6), is shown dashed, and the corresponding edge is shown in Fig. 8.9.

In general, the problem of finding a minimum-cost path is not trivial from a computational point of view. Typically, the approach is to sacrifice optimality for the sake of speed, and the algorithm discussed below is representative of a class of

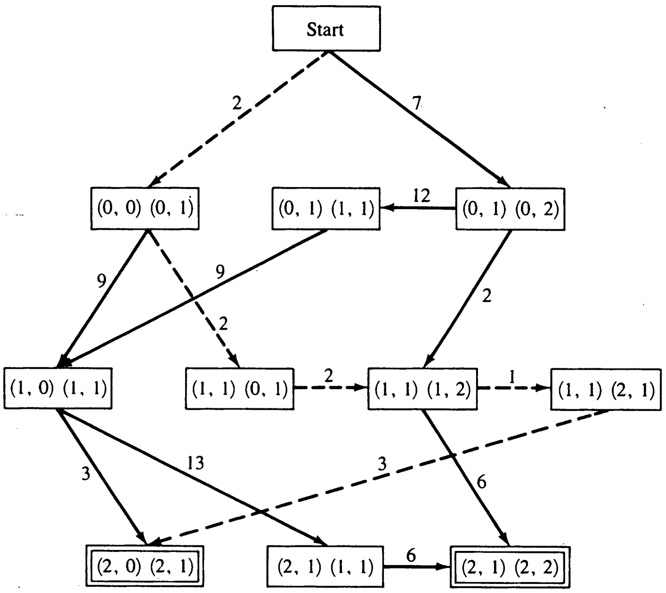


Figure 8.8 Graph used for finding an edge in the image of Fig. 8.7. The pair $(a, b)(c, d)$ in each box refers to points p and q, respectively. Note that p is assumed to be to the right of the path as the image is traversed from top to bottom. The dashed lines indicate the minimum-cost path. (Adapted from Martelli [1972], © Academic Press.)

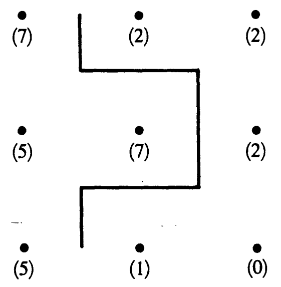


Figure 8.9 Edge corresponding to minimum-cost path in Fig. 8.8.

procedures which use heuristics in order to reduce the search effort. Let $r(n)$ be an estimate of the cost of a minimum-cost path from the start node s to a goal node, where the path is constrained to go through n. This cost can be expressed as the estimate of the cost of a minimum-cost path from s to n, plus an estimate of the cost of that path from n to a goal node; that is,

$$r(n) = g(n) + h(n) \tag{8.2-8}$$

Here, $g(n)$ can be chosen as the lowest-cost path from s to n found so far, and $h(n)$ is obtained by using any available heuristic information (e.g., expanding only certain nodes based on previous costs in getting to that node). An algorithm that uses $r(n)$ as the basis for performing a graph search is as follows:

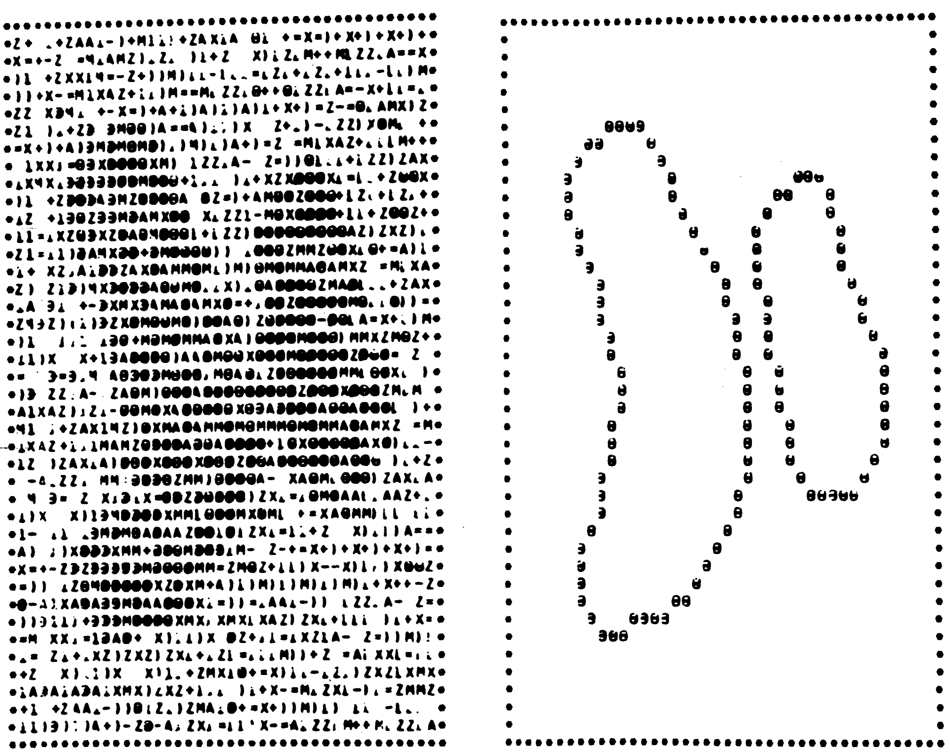
Step 1. Mark the start node OPEN and set $g(s) = 0$.
Step 2. If no node is OPEN, exit with failure; otherwise continue.
Step 3. Mark CLOSED the OPEN node n whose estimate $r(n)$ computed from Eq. (8.2-8) is smallest. (Ties for minimum r values are resolved arbitrarily, but always in favor of a goal node.)
Step 4. If n is a goal node, exit with the solution path obtained by tracing back through the pointers; otherwise continue.
Step 5. Expand node n, generating all its successors. (If there are no successors, go to step 2.)
Step 6. If a successor n_i is not marked, set

$$r(n_i) = g(n) + c(n, n_i)$$

mark it OPEN, and direct pointers from it back to n.
Step 7. If a successor n_i is marked CLOSED or OPEN, update its value by letting

$$g'(n_i) = \min[g(n_i), g(n) + c(n, n_i)]$$

Mark OPEN those CLOSED successors whose g' values were thus lowered and redirect to n the pointers from all nodes whose g' values were lowered. Go to step 2.



(*a*) (*b*)

Figure 8.10 (*a*) Noisy image. (*b*) Result of edge detection by using the heuristic graph search. (From Martelli [1976], ©ACM.)

In general, this algorithm is not guaranteed to find a minimum-cost path; its advantage is speed via the use of heuristics. It can be shown, however, that if $h(n)$ is a lower bound on the cost of the minimal-cost path from node n to a goal node, then the procedure will indeed find an optimal path to a goal (Hart et al. [1968]). If no heuristic information is available (i.e., $h \equiv 0$) then the procedure reduces to the *uniform-cost algorithm* of Dijkstra [1959].

Example: A typical result obtainable with this procedure is shown in Fig. 8.10. Part (*a*) of this figure shows a noisy image and Fig. 8.10*b* is the result of edge segmentation by searching the corresponding graph for low-cost paths. Heuristics were brought into play by not expanding those nodes whose cost exceeded a given threshold. □

8.2.2 Thresholding

The concept of thresholding was introduced in Sec. 7.6.5 as an operation involving tests against a function of the form

$$T = T[x,\ y,\ p(x,\ y),\ f(x,\ y)] \tag{8.2-9}$$

where $f(x, y)$ is the intensity of point (x, y) and $p(x, y)$ denotes some local property measured in a neighborhood of this point. A thresholded image, $g(x, y)$ is created by defining

$$g(x,\ y) = \begin{cases} 1 & \text{if } f(x,\ y) > T \\ 0 & \text{if } f(x,\ y) \leqslant T \end{cases} \tag{8.2-10}$$

so that pixels in $g(x, y)$ labeled 1 correspond to objects, while pixels labeled 0 correspond to the background. Equation (8.2-10) assumes that the intensity of objects is greater than the intensity of the background. The opposite condition is handled by reversing the sense of the inequalities.

Global vs. Local Thresholds. As indicated in Sec. 7.6.5, when T in Eq. (8.2-9) depends only on $f(x, y)$, the threshold is called *global.* If T depends on both $f(x, y)$ and $p(x, y)$, then it is called a *local threshold.* If, in addition, T depends on the spatial coordinates x and y, it is called a *dynamic threshold.*

Global thresholds have application in situations where there is clear definition between objects and background, and where illumination is relatively uniform. The backlighting and structured lighting techniques discussed in Sec. 7.3 usually yield images that can be segmented by global thresholds. For the most part, arbitrary illumination of a work space yields images that, if handled by thresholding, require some type of local analysis to compensate for effects such as nonuniformities in illumination, shadows, and reflections.

In the following discussion we consider a number of techniques for selecting segmentation thresholds. Although some of these techniques can be used for global threshold selection, they are usually employed in situations requiring local threshold analysis.

Optimum Threshold Selection. It is often possible to consider a histogram as being formed by the sum of probability density functions. In the case of a bimodal histogram the overall function approximating the histogram is given by

$$p(z) = P_1 p_1(z) + P_2 p_2(z) \tag{8.2-11}$$

where z is a random variable denoting intensity, $p_1(z)$ and $p_2(z)$ are the probability density functions, and P_1 and P_2 are called *a priori* probabilities. These last two quantities are simply the probabilities of occurrence of two types of intensity levels in an image. For example, consider an image whose histogram is shown in Fig. 8.11*a*. The overall histogram may be approximated by the sum of two proba-

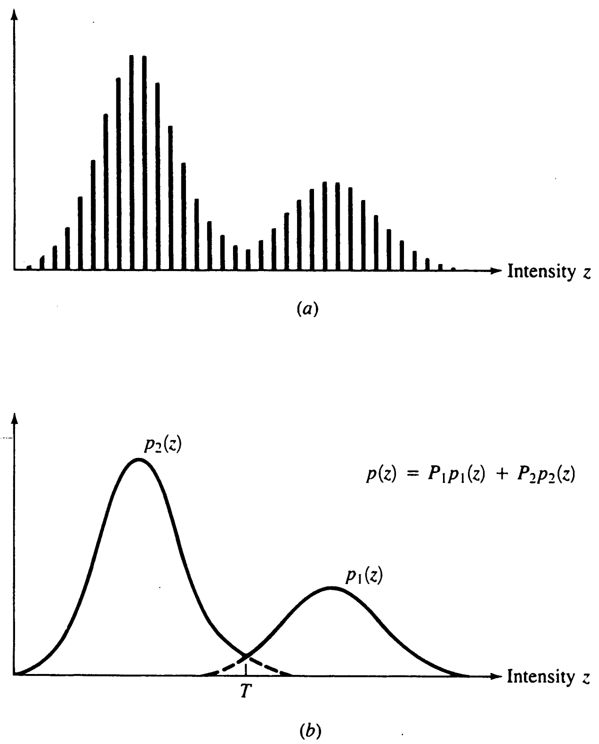


Figure 8.11 (*a*) Intensity histogram. (*b*) Approximation as the sum of two probability density functions.

bility density functions, as shown in Fig. 8.11*b*. If it is known that light pixels represent objects and also that 20 percent of the image area is occupied by object pixels, then $P_1 = 0.2$. It is required that

$$P_1 + P_2 = 1 \tag{8.2-12}$$

which simply says that, in this case, the remaining 80 percent are background pixels.

Let us form two functions of z, as follows:

$$d_1(z) = P_1 p_1(z) \tag{8.2-13}$$

and

$$d_2(z) = P_2 p_2(z) \tag{8.2-14}$$

It is known from decision theory (Tou and Gonzalez [1974]) that the average error of misclassifying an object pixel as background, or vice versa, is minimized by using the following rule: Given a pixel with intensity value z, we substitute that value of z into Eqs. (8.2-13) and (8.2-14). Then, we classify the pixel as an object pixel if $d_1(z) > d_2(z)$ or as background pixel if $d_2(z) > d_1(z)$. The optimum threshold is then given by the value of z for which $d_1(z) = d_2(z)$. That is, setting $z = T$ in Eqs. (8.2-13) and (8.2-14), we have that the optimum threshold satisfies the equation

$$P_1 p_1(T) = P_2 p_2(T) \tag{8.2-15}$$

Thus, if the functional forms of $p_1(z)$ and $p_2(z)$ are known, we can use this equation to solve for the optimum threshold that separates objects from the background. Once this threshold is known, Eq. (8.2-10) can be used to segment a given image.

As an important illustration of the use of Eq. (8.2-15), suppose that $p_1(z)$ and $p_2(z)$ are gaussian probability density functions; that is,

$$p_1(z) = \frac{1}{\sqrt{2\pi}\sigma_1} \exp - \left[\frac{(z - m_1)^2}{2\sigma_1^2} \right] \tag{8.2-16}$$

and

$$p_2(z) = \frac{1}{\sqrt{2\pi}\sigma_2} \exp - \left[\frac{(z - m_2)^2}{2\sigma_2^2} \right] \tag{8.2-17}$$

Letting $z = T$ in these expressions, substituting into Eq. (8.2-15), and simplifying yields a quadratic equation in T:

$$AT^2 + BT + C = 0 \tag{8.2-18}$$

where

$$\begin{aligned} A &= \sigma_1^2 - \sigma_2^2 \\ B &= 2(m_1\sigma_2^2 - m_2\sigma_1^2) \\ C &= \sigma_1^2 m_2^2 - \sigma_2^2 m_1^2 + 2\sigma_1^2\sigma_2^2 \ln \frac{\sigma_2 P_1}{\sigma_1 P_2} \end{aligned} \tag{8.2-19}$$

The possibility of two solutions indicates that two threshold values may be required to obtain an optimal solution.

If the standard deviations are equal, $\sigma_1 = \sigma_2 = \sigma$, a single threshold is sufficient:

$$T = \frac{m_1 + m_2}{2} + \frac{\sigma^2}{m_1 - m_2} \ln \frac{P_2}{P_1} \tag{8.2-20}$$

If $\sigma = 0$ or $P_1 = P_2$, the optimum threshold is just the average of the means. The former condition simply means that both the object and background intensities are constant throughout the image. The latter condition means that object and background pixels are equally likely to occur, a condition met whenever the number of object pixels is equal to the number of background pixels in an image.

Example: As an illustration of the concepts just discussed, consider the segmentation of the mechanical parts shown in Fig. 8.12*a*, where, for the moment, we ignore the grid superimposed on the image. Figure 8.12*b* shows the result of computing a global histogram, fitting it with a bimodal gaussian density, establishing an optimum global threshold, and finally using this threshold in Eq. (8.2-10) to segment the image. As expected, the variations in intensity rendered this approach virtually useless. A similar approach, however, can be carried out on a local basis by subdividing the image into subimages, as defined by the grid in Fig. 8.12*a*.

After the image has been subdivided, a histogram is computed for each subimage and a test of bimodality is conducted. The bimodal histograms are fitted by a mixed gaussian density and the corresponding optimum threshold is computed using Eqs. (8.2-18) and (8.2-19). No thresholds are computed for subimages without bimodal histograms; instead, these regions are assigned thresholds computed by interpolating the thresholds from neighboring subimages that are bimodal. The histograms for each subimage are shown in Fig. 8.12*c*, where the horizontal lines provide an indication of the relative scales of these histograms. At the end of this procedure a second interpolation is carried out on a point-by-point manner using neighboring thresholds so that every point is assigned a threshold value, $T(x, y)$. Note that this is a dynamic threshold since it depends on the spatial coordinates (x, y). A display of how $T(x, y)$ varies as a function of position is shown in Fig. 8.12*d*.

Finally, a thresholded image is created by comparing every pixel in the original image against its corresponding threshold. The result of using this method in this particular case is shown in Fig. 8.12*e*. The improvement over a single, global threshold is evident. It is of interest to note that this method involves local analysis to establish the threshold for each cell, and that these local thresholds are interpolated to create a dynamic threshold which is finally used for segmentation. □

The approach developed above is applicable to the selection of multiple thresholds. Suppose that we can model a multimodal histogram as the sum of n probability density functions so that

$$p(z) = P_1 p_1(z) + \cdots + P_n p_n(z) \qquad (8.2\text{-}21)$$

Then, the optimum thresholding problem may be viewed as classifying a given pixel as belonging to one of n possible categories. The minimum-error decision rule is now based on n functions of the form

$$d_i(z) = P_i p_i(z) \qquad i = 1, 2, \ldots, n \qquad (8.2\text{-}22)$$

A given pixel with intensity z is assigned to the kth category if $d_k(z) > d_j(z)$, $j = 1, 2, \ldots, n$; $j \neq k$. As before, the optimum threshold between category k

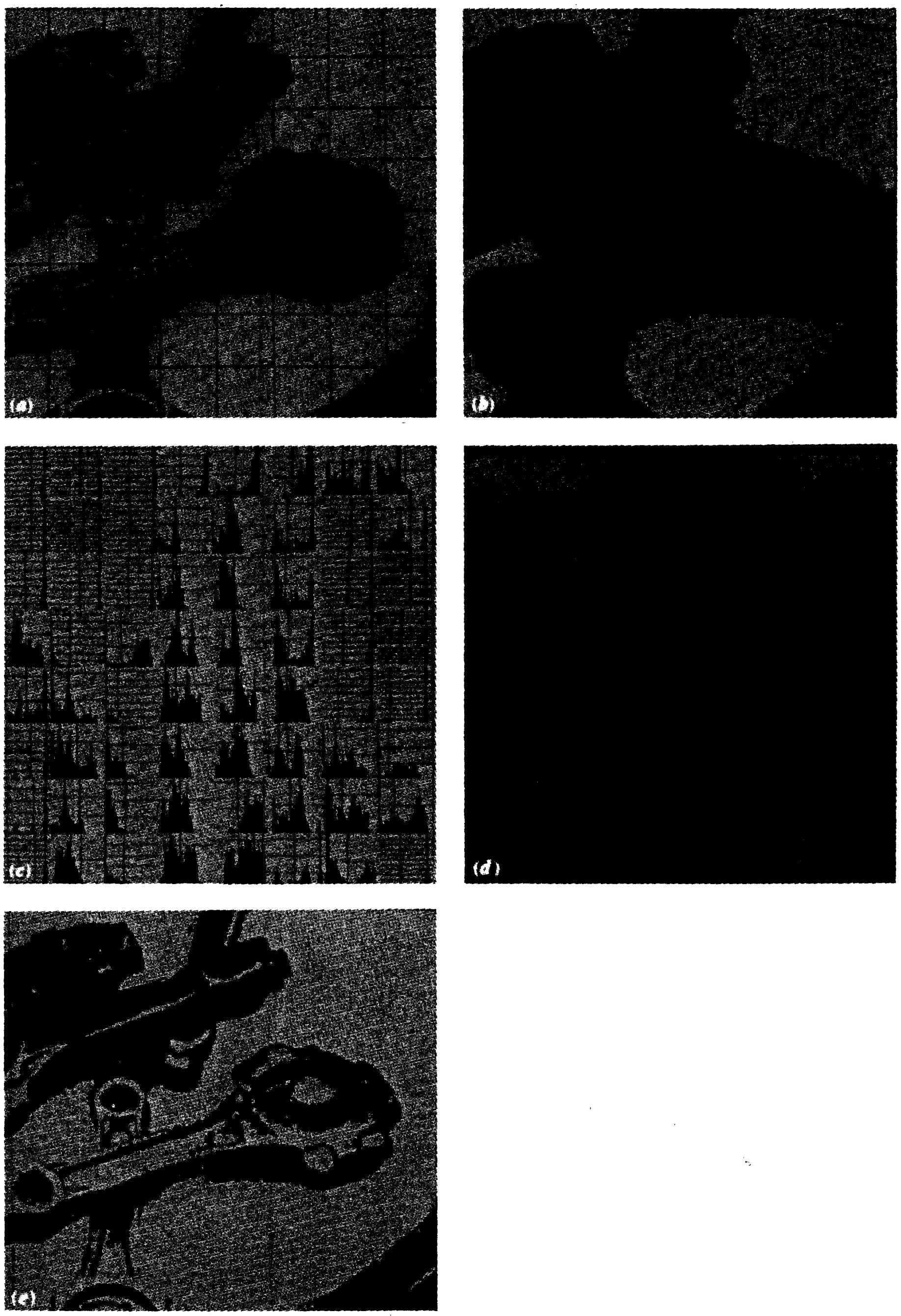


Figure 8.12 (*a*) Image of mechanical parts showing local-region grid. (*b*) Result of global thresholding. (*c*) Histograms of subimages. (*d*) Display of dynamic threshold. (*e*) Result of dynamic thresholding. (From Rosenfeld and Kak [1982], courtesy of A. Rosenfeld.)

and category j, denoted by T_{kj}, is obtained by solving the equation

$$P_k p_k(T_{kj}) = P_j p_j(T_{kj}) \qquad (8.2\text{-}23)$$

As indicated in Sec. 7.6.5, the real problem with using multiple histogram thresholds lies in establishing meaningful histogram modes.

Threshold Selection Based on Boundary Characteristics. One of the most important aspects of selecting a threshold is the capability to reliably identify the mode peaks in a given histogram. This is particularly important for automatic threshold selection in situations where image characteristics can change over a broad range of intensity distributions. Based on the discussion in the last two sections, it is intuitively evident that the chances of selecting a "good" threshold should be considerably enhanced 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 boundary between objects and the background. One immediate and obvious improvement is that this makes histograms less dependent on the relative size between objects and the background. For instance, the intensity histogram of an image composed of a large, nearly constant background area and one small object would be dominated by a large peak due to the concentration of background pixels. If, on the other hand, only the pixels on or near the boundary between the object and the background were used, the resulting histogram would have peaks whose heights are more balanced. In addition, the probability that a given pixel lies near the edge of an object is usually equal to the probability that it lies on the edge of the background, thus improving the symmetry of the histogram peaks. Finally, as will be seen below, using pixels that satisfy some simple measures based on gradient and Laplacian operators has a tendency to deepen the valley between histogram peaks.

The principal problem with the foregoing comments is that they implicitly assume that the boundary between objects and background is known. This information is clearly not available during segmentation since finding a division between objects and background is precisely the ultimate goal of the procedures discussed here. However, we know from the material in Sec. 7.6.4 that an indication of whether a pixel is on an edge may be obtained by computing its gradient. In addition, use of the Laplacian can yield information regarding whether a given pixel lies on the dark (e.g., background) or light (object) side of an edge. Since, as discussed in Sec. 7.6.4, the Laplacian is zero on the interior of an ideal ramp edge, we may expect in practice that the valleys of histograms formed from the pixels selected by a gradient/Laplacian criterion to be sparsely populated. This property produces the highly desirable deep valleys mentioned earlier in this section.

The gradient, $G[f(x, y)]$, at any point in an image is given by Eq. (7.6-38) or (7.6-39). Similarly, the Laplacian $L[f(x, y)]$ is given by Eq. (7.6-47). We may use these two quantities to form a three-level image, as follows:

$$s(x, y) = \begin{cases} 0 & \text{if } G[f(x, y)] < T \\ + & \text{if } G[f(x, y)] \geqslant T \text{ and } L[f(x, y)] \geqslant 0 \\ - & \text{if } G[f(x, y)] \geqslant T \text{ and } L[f(x, y)] < 0 \end{cases} \qquad (8.2\text{-}24)$$

where the symbols 0, +, and − represent any three distinct gray levels, and T is a threshold. Assuming a dark object on a light background, and with reference to Fig. 7-34*b*, the use of Eq. (8.2-24) produces an image $s(x, y)$ in which all pixels which are not on an edge (as determined by $G[f(x, y)]$ being less than T) are labeled "0," all pixels on the dark side of an edge are labeled "+," and all pixels on the light side of an edge are labeled "−." The symbols + and − in Eq. (8.2-24) are reversed for a light object on a dark background. Figure 8.13 shows the labeling produced by Eq. (8.2-24) for an image of a dark, underlined stroke written on a light background.

The information obtained by using the procedure just discussed can be used to generate a segmented, binary image in which 1's correspond to objects of interest

```
000000000000000000--00+0+000000000000000000000000000000000000000000000000
00000000000000000000000000000+000000+0000000+0000000000000000000000000000
0000000000000000000000000-00000000000--000000-00++00++0000000000000000000
000000000000000000000000000000000000000000000000000000000000+000000000000
00000000000000000000000000000000000000000000000000000-0000-00000+00000000
00000000000000000000+0000000000000000000000000000000000000000-00000+00000
00000000000000000000000+000000000000000000000000000000000000000000-0-00+0
00000000-00000000000-00000000++000000000000000000000000000000000000000000
0000000000000000000000000000000000000+000+0000000000000000000000000000-0-
000000000000000000000000000000000-000-0000000-0+0000000000000000000000000
0000000000000000000000000000000000000000000000000000000+00000+0000000000
000000000000000000000000000000000000000000000000000000-0000-0000+00000000
000000000000000000000000000000000000000000000000000000000000-00+000000000
0000000000000000-------00000000000000000000000000000000000000000000000000
00000000000000---------------00000000000000000000000000000000000000000000
0000000000000---+++++++---00000000000000000000000000000000000000000000000
00000000000---++++00++++--00000000000000000------000000000000000000000000
00000000000--+++++++++++++--000000000000------------000000000000000000000
0000000000---+++-----+----00000000000------++++++++---0000000000000000000
00000000---++++----------00000000000----+++++++++++++---000000000000000000
00000000---+++--0-+++-----00000000---+++++++++++++++--000000000000000000
00000000---+++-00--+++++++--00000-----+++-------++++++++-00000000000000000
00000000--++++--00--+++0+++-0000----+++----------++++++++---00000000000000
00000000--++++--000--++00+++--00---+++++-----00------++++++---000000000000
00000000---++++--000--++++++++-----++++++--00000000---++++++--00000000000
000000000--+++--000--++++-++++-----++++----00000000000--++++-----00000---
000000000--++++--00--++++----++++++++++---000000000000---+++++++--------+
00000000000--++++---0--++++++----++++++++---000000000000000---++++++++++++
000000000000--++++---++++++--------------0000000000000000000---++++++++++
0000000000000--+++++++++++++--000000000000000000000000000000000------------
00000000000000---+++++++---00000000000000000000000000000000000--------0
0000000000000000-----------0000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
----000---00000-0000-000000000000000000000000000000000000000000000000000
-------------------------------------------------------------------------
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
-------------------------------------------------------------------------
000-00000000--00-000000-------0000000-----------------0-----------------
0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure 8.13 Image of a handwritten stroke coded by using Eq.(8.2-24). (From White and Rohrer [1983], ©IBM.)

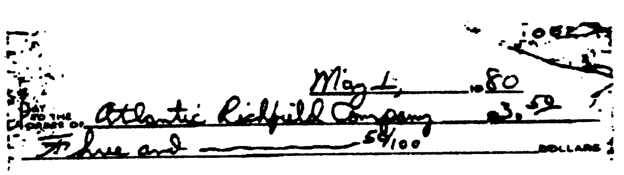
and 0's correspond to the background. First we note that the transition (along a horizontal or vertical scan line) from a light background to a dark object must be characterized by the occurrence of a $-$ followed by a $+$ in $s(x, y)$. The interior of the object is composed of pixels which are labeled either 0 or $+$. Finally, the transition from the object back to the background is characterized by the occurrence of a $+$ followed by a $-$. Thus we have that a horizontal or vertical scan line containing a section of an object has the following structure:

$$(\cdots)(-, +)(0 \text{ or } +)(+, -)(\cdots)$$

where $(\cdots)$ represents any combination of $+$, $-$, or 0. The innermost parentheses contain object points and are labeled 1. All other pixels along the same scan line are labeled 0, with the exception of any sequence of (0 or $+$) bounded by $(-, +)$ and $(+, -)$.

Example: As an illustration of the concepts just discussed, consider Fig. 8.14*a* which shows an image of an ordinary scenic bank check. Figure 8.15 shows the histogram as a function of gradient values for pixels with gradients greater than 5. It is noted that this histogram has the properties discussed earlier. That is, it has two dominant modes which are symmetric, nearly of the same height, and are separated by a distinct valley. Finally, Fig. 8.14*b* shows the segmented image obtained by using Eq. (8.2-24) with T near the midpoint of the valley. The result was made binary by using the sequence analysis discussed above. □

(*a*)



(*b*)

Figure 8.14 (*a*) Original image. (*b*) Segmented image. (From White and Rohrer [1983], ©IBM.)

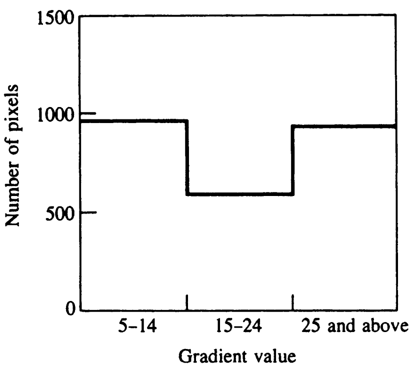


Figure 8.15 Histogram of pixels with gradients greater than 5. (From White and Rohrer [1983, ©IBM].)

Thresholds Based on Several Variables. The techniques discussed thus far deal with thresholding a single intensity variable. In some applications it is possible to use more than one variable to characterize each pixel in an image, thus enhancing the capability to differentiate not only between objects and background, but also to distinguish between objects themselves. A notable example is color sensing, where red, green, and blue (RGB) components are used to form a composite color image. In this case, each pixel is characterized by three values and it becomes possible to construct a three-dimensional histogram. The basic procedure is the same as that used for one variable. For example, given three 16-level images corresponding to the RGB components of a color sensor, we form a $16 \times 16 \times 16$ grid (cube) and insert in each cell of the cube the number of pixels whose RGB components have intensities corresponding to the coordinates defining the location of that particular cell. Each entry can then be divided by the total number of pixels in the image to form a normalized histogram.

The concept of threshold selection now becomes that of finding clusters of points in three-dimensional space, where each "tight" cluster is analogous to a dominant mode in a one-variable histogram. Suppose, for example, that we find two significant clusters of points in a given histogram, where one cluster corresponds to objects and the other to the background. Keeping in mind that each pixel now has three components and, therefore, may be viewed as a point in three-dimensional space, we can segment an image by using the following procedure: For every pixel in the image we compute the distance between that pixel and the centroid of each cluster. Then, if the pixel is closer to the centroid of the object cluster, we label it with a 1; otherwise, we label it with a 0. This concept is easily extendible to more pixel components and, certainly, to more clusters. The principal difficulty is that finding meaningful clusters generally becomes an increasingly complex task as the number of variables is increased. The reader interested in further pursuing techniques for cluster seeking can consult, for example, the book by Tou and Gonzalez [1974]. This and other related techniques for segmentation are surveyed by Fu and Mui [1981].

Example: As an illustration of the multivariable histogram approach, consider Fig. 8.16. Part (*a*) of this image is a monochrome image of a color photograph. The original color image was composed of three 16-level RGB images. For our purposes, it is sufficient to note that the scarf and one of the flowers were a vivid red, and that the hair and facial colors were light and different in spectral characteristics from the window and other background features.

Figure 8.16*b* was obtained by thresholding about a histogram cluster which was known to contain RGB components representative of flesh tones. It is important to note that the window, which in the monochrome image has a range of intensities close to those of the hair, does not appear in the segmented image because its multispectral characteristics are quite different. The fact that some small regions on top of the subject's hair appear in the segmented image indicates that their color is similar to flesh tones. Figure 8.16*c* was obtained by thresholding about a cluster close to the red axis. In this case

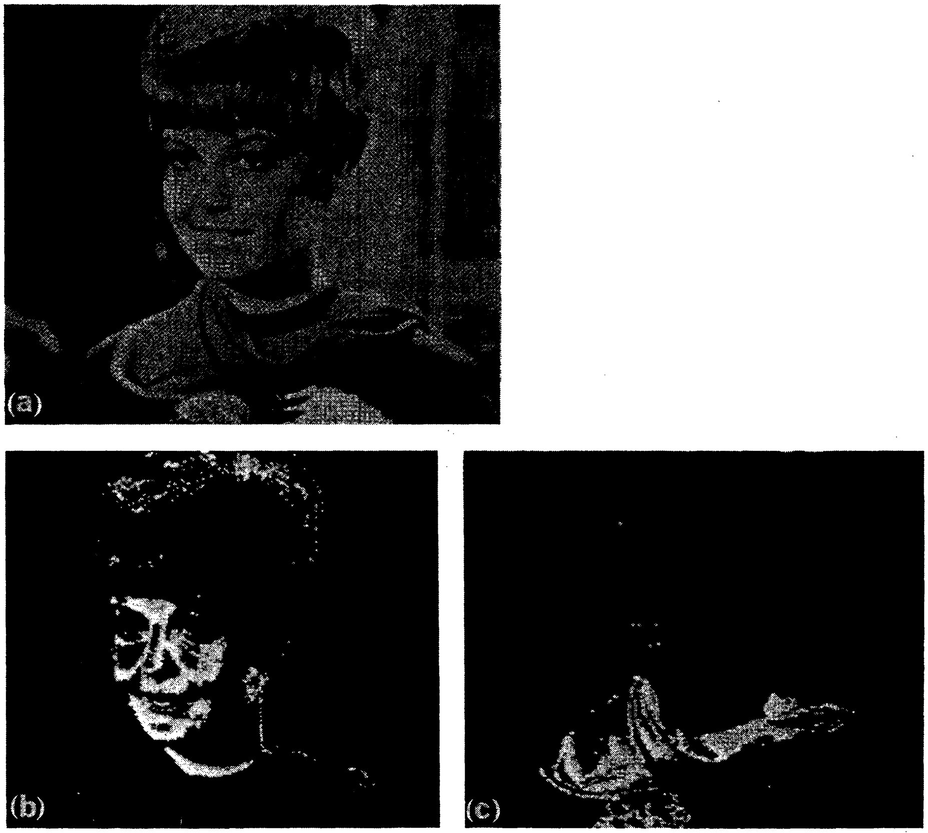


Figure 8.16 Segmentation by multivariable threshold approach. (From Gonzalez and Wintz [1977], © Addison-Wesley.)

only the scarf, the red flower, and a few isolated points appeared in the segmented image. The threshold used to obtain both results was a distance of one cell. Thus, any pixels whose components placed them within a unit distance from the centroid of the cluster under consideration were coded white. All other pixels were coded black. □

8.2.3 Region-Oriented Segmentation

Basic Formulation. The objective of segmentation is to partition an image into regions. In Sec. 8.2.1 we approached this problem by finding boundaries between regions based on intensity discontinuities, while in Sec. 8.2.2 segmentation was accomplished via thresholds based on the distribution of pixel properties, such as intensity or color. In this section we discuss segmentation techniques that are based on finding the regions directly.

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

1. $\bigcup_{i=1}^{n} R_i = R$
2. R_i is a connected region, $i = 1, 2, \ldots, n$
3. $R_i \cap R_j = \phi$ for all i and j, $i \neq j$
4. $P(R_i) = \text{TRUE}$ for $i = 1, 2, \ldots, n$
5. $P(R_i \cup R_j) = \text{FALSE}$ for $i \neq j$

where $P(R_i)$ is a logical predicate defined over the points in set R_i, and ϕ is the null set.

Condition 1 indicates that the segmentation must be complete; that is, every pixel must be in a region. The second condition requires that points in a region must be connected (see Sec. 7.5.2 regarding connectivity). Condition 3 indicates that the regions must be disjoint. Condition 4 deals with the properties that must be satisfied by the pixels in a segmented region. One simple example is: $P(R_i) =$ TRUE if all pixels in R_i have the same intensity. Finally, condition 5 indicates that regions R_i and R_j are different in the sense of predicate P. The use of these conditions in segmentation algorithms is discussed in the following subsections.

Region Growing by Pixel Aggregation. As implied by its name, region growing is a procedure that groups pixels or subregions into larger regions. The simplest of these approaches is *pixel aggregation*, where we start with a set of "seed" points and from these grow regions by appending to each seed point those neighboring pixels that have similar properties (e.g., intensity, texture, or color). As a simple illustration of this procedure consider Fig. 8.17*a*, where the numbers inside the cells represent intensity values. Let the points with coordinates (3, 2) and (3, 4) be used as seeds. Using two starting points will result in a segmentation consisting of, at most, two regions: R_1 associated with seed (3, 2) and R_2 associated

	1	2	3	4	5
1	0	0	5	6	7
2	1	1	5	8	7
3	0	$\underline{1}$	6	$\underline{7}$	7
4	2	0	7	6	6
5	0	1	5	6	5

(*a*)

a	a	b	b	b
a	a	b	b	b
a	a	b	b	b
a	a	b	b	b
a	a	b	b	b

(*b*)

a	a	a	a	a
a	a	a	a	a
a	a	a	a	a
a	a	a	a	a
a	a	a	a	a

(*c*)

Figure 8.17 Example of region growing using known starting points. (*a*) Original image array. (*b*) Segmentation result using an absolute difference of less than 3 between intensity levels. (*c*) Result using an absolute difference less than 8. (From Gonzalez and Wintz [1977], © Addison-Wesley.)

with seed (3, 4). The property P that we will use to include a pixel in either region is that the absolute difference between the intensity of the pixel and the intensity of the seed be less than a threshold T (any pixel that satisfies this property simultaneously for both seeds is arbitrarily assigned to regions R_1). The result obtained using $T = 3$ is shown in Fig. 8.17*b*. In this case, the segmentation consists of two regions, where the points in R_1 are denoted by a's and the points in R_2 by b's. It is noted that any starting point in either of these two resulting regions would have yielded the same result. If, on the other hand, we had chosen $T = 8$, a single region would have resulted, as shown in Fig. 8.17*c*.

The preceding example, while simple in nature, points out some important problems in region growing. Two immediate problems are the selection of initial seeds that properly represent regions of interest and the selection of suitable properties for including points in the various regions during the growing process. Selecting a set of one or more starting points can often be based on the nature of the problem. For example, in military applications of infrared imaging, targets of interest are hotter (and thus appear brighter) than the background. Choosing the brightest pixels is then a natural starting point for a region-growing algorithm. When a priori information is not available, one may proceed by computing at every pixel the same set of properties that will ultimately be used to assign pixels to regions during the growing process. If the result of this computation shows clusters of values, then the pixels whose properties place them near the centroid of these clusters can be used as seeds. For instance, in the example given above, a histogram of intensities would show that points with intensity of 1 and 7 are the most predominant.

The selection of similarity criteria is dependent 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 is heavily dependent on the use of color. This problem would be significantly more difficult to handle by using monochrome images alone. Unfortunately, the availability of multispectral and other complementary image data is the exception, rather than the rule, in industrial computer vision. Typically, region analysis must be carried out using a set of descriptors based on intensity and spatial properties (e.g., moments, texture) of a single image source. A discussion of descriptors useful for region characterization is given in Sec. 8.3.

It is important to note that descriptors alone can yield misleading results if connectivity or adjacency information is not used in the region growing process. An illustration of this is easily visualized by considering a random arrangement of pixels with only three distinct intensity values. Grouping pixels with the same intensity to form a "region" without paying attention to connectivity would yield a segmentation result that is meaningless in the context of this discussion.

Another important problem in region growing is the formulation of a stopping rule. Basically, we stop growing a region when no more pixels satisfy the criteria for inclusion in that region. We mentioned above criteria such as intensity, texture, and color, which are local in nature and do not take into account the "history" of region growth. Additional criteria that increase the power of a region-growing algorithm incorporate the concept of size, likeness between a candidate pixel and the pixels grown thus far (e.g., a comparison of the intensity of a candidate and the average intensity of the region), and the shape of a given 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.

Region Splitting and Merging. The procedure discussed above grows regions starting from a given set of seed points. An alternative is to initially subdivide an image into a set of arbitrary, disjoint regions and then merge and/or split the regions in an attempt to satisfy the conditions stated at the beginning of this section. A split and merge algorithm which iteratively works toward satisfying these constraints may be explained as follows.

Let R represent the entire image region, and select a predicate P. Assuming a square image, one approach for segmenting R is to successively subdivide it into smaller and smaller quadrant regions such that, for any region R_i, $P(R_i)$ = TRUE. The procedure starts with the entire region R. If $P(R)$ = FALSE, we divide the image into quadrants. If P is FALSE for any quadrant, we subdivide that quadrant into subquadrants, and so on. This particular splitting technique has a convenient representation in the form of a so-called *quadtree* (i.e., a tree in which each node has exactly four descendants). A simple illustration is shown in Fig. 8.18. It is noted that the root of the tree corresponds to the entire image and that each node corresponds to a subdivision. In this case, only R_4 was subdivided further.

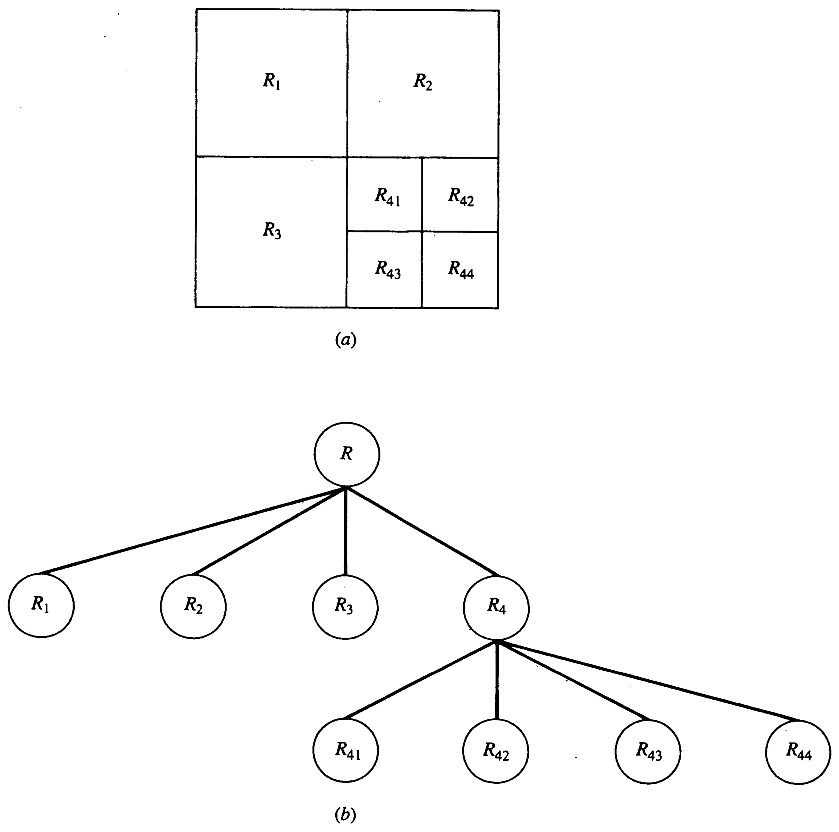


Figure 8.18 (*a*) Partitioned image. (*b*) Corresponding quadtree.

If we used only splitting, it is likely that the final partition would contain adjacent regions with identical properties. This may be remedied by allowing merging, as well as splitting. In order to satisfy the segmentation conditions stated earlier, we merge only adjacent regions whose combined pixels satisfy the predicate P; that is, we merge two adjacent regions R_i and R_k only if $P(R_i \cup R_k) =$ TRUE.

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

1. Split into four disjoint quadrants any region R_i for which $P(R_i) =$ FALSE
2. Merge any adjacent regions R_j and R_k for which $P(R_j \cup R_k) =$ TRUE
3. Stop when no further merging or splitting is possible

A number of variations of this basic theme are possible (Horowitz and Pavlidis [1974]). For example, one possibility is to initially split the image into a set of square blocks. Further splitting is carried out as above, but merging is initially limited to groups of four blocks which are descendants in the quadtree representation and which satisfy the predicate P. When no further mergings of this type are possible, the procedure is terminated by one final merging of regions satisfying step 2 above. At this point, the regions that are merged may be of different sizes. The principal advantage of this approach is that it uses the same quadtree for splitting and merging, until the final merging step.

Example: An illustration of the split and merge algorithm discussed above is shown in Fig. 8.19. The image under consideration consists of a single object and background. For simplicity, we assume that both the object and background have constant intensities and that $P(R_i) =$ TRUE if all pixels in R_i have the same intensity. Then, for the entire image region R, it follows that $P(R) =$ FALSE, so the image is split as shown in Fig. 8.19*a*. In the next step, only the top left region satisfies the predicate so it is not changed, while the other three quadrant regions are split into subquadrants, as shown in Fig. 8.19*b*. At this point several regions can be merged, with the exception of the two subquadrants that include the lower part of the object; these do not satisfy the predicate and must be split further. The results of the split and merge operation are shown in Fig. 8.19*c*. At this point all regions satisfy P, and merging the appropriate regions from the last split operation yields the final, segmented result shown in Fig. 8.19*d*. □

8.2.4 The Use of Motion

Motion is a powerful cue used by humans and other animals in extracting objects of interest from the background. In robot vision, motion arises in conveyor belt applications, by motion of a sensor mounted on a moving arm or, more rarely, by motion of the entire robot system. In this subsection we discuss the use of motion for segmentation from the point of view of image differencing.

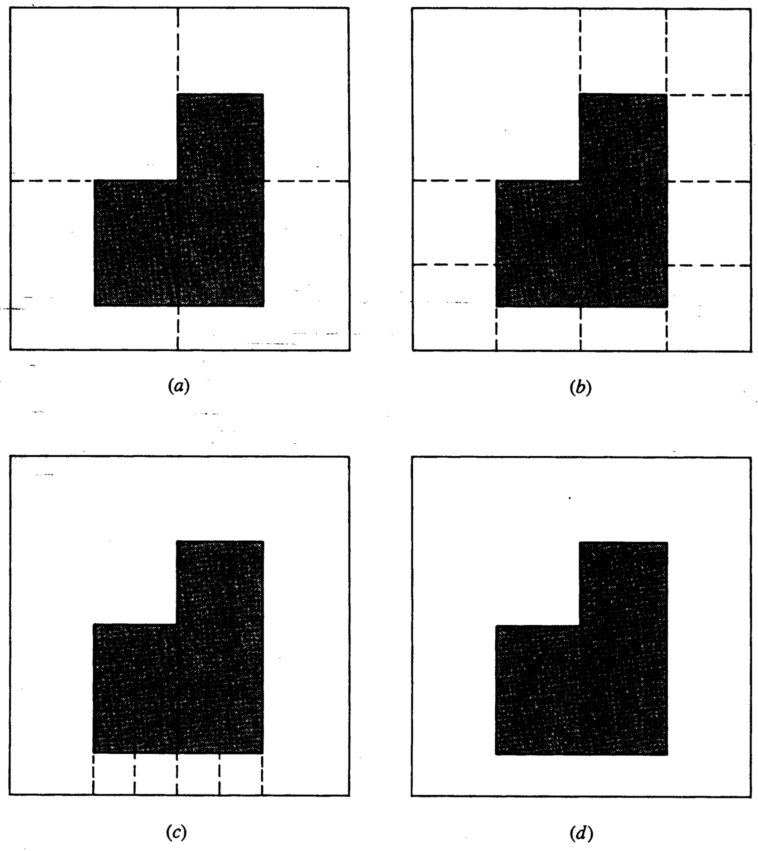


Figure 8.19 Example of split and merge algorithm.

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 on a pixel-by-pixel basis. One procedure for doing this is to form a *difference image*.

Suppose that we have a reference image containing only stationary components. If we compare this image against a subsequent image having the same environment but including a moving object, the difference of the two images will cancel the stationary components, leaving only nonzero entries that correspond to the nonstationary image components.

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

$$d_{ij}(x, y) = \begin{cases} 1 & \text{if } |f(x, y, t_i) - f(x, y, t_j)| > \theta \\ 0 & \text{otherwise} \end{cases} \tag{8.2-25}$$

where θ is a threshold. It is noted that $d_{ij}(x, y)$ has a 1 at spatial coordinates (x, y) only if the intensity difference between the two images is appreciably different at those coordinates, as determined by the threshold θ.

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

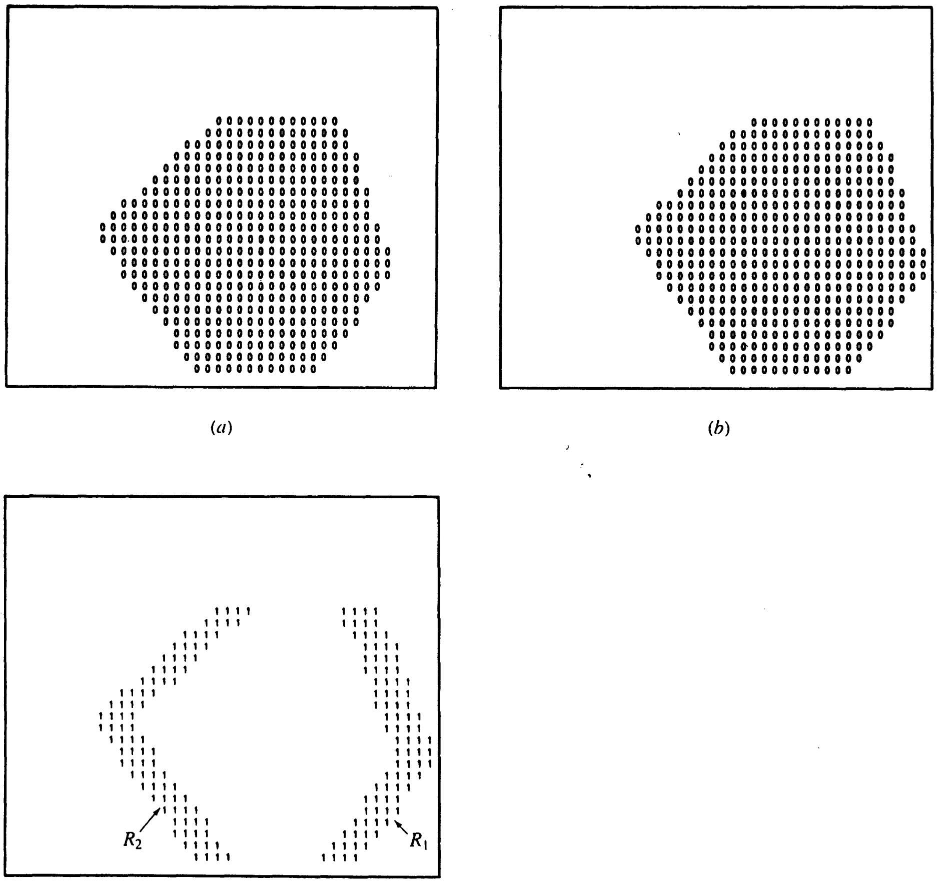


Figure 8.20 (*a*) Image taken at time t_i. (*b*) Image taken at time t_j. (*c*) Difference image. (From Jain [1981], ©IEEE.)

The foregoing concepts are illustrated in Fig. 8.20. Part (*a*) of this figure shows a reference image frame taken at time t_i and containing a single object of constant intensity that is moving with uniform velocity over a background surface, also of constant intensity. Figure 8.20*b* shows a current frame taken at time t_j, and Fig. 8.20*c* shows the difference image computed using Eq. (8.2-25) with a threshold larger than the constant background intensity. It is noted that two disjoint regions were generated by the differencing process: one region is the result of the leading edge and the other of the trailing edge of the moving object.

Accumulative Differences. As indicated above, a difference image will often contain isolated entries that are due to noise. Although the number of these entries can be reduced or completely eliminated by a thresholded connectivity analysis, this filtering process can also remove small or slow-moving objects. The approach discussed in this section addresses this problem by considering changes at a pixel location on several frames, thus introducing a "memory" into the process. The basic idea is to ignore those changes which occur only sporadically over a frame sequence and can, therefore, be attributed to random noise.

Consider a sequence of image frames $f(x, y, t_1)$, $f(x, y, t_2), \ldots,$ $f(x, y, t_n)$, and let $f(x, y, t_1)$ be the *reference image*. An *accumulative difference image* 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 that there is a difference at that pixel location between the reference and an image in the sequence. Thus, when the *k*th 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 from the corresponding pixel value in the reference image. Differences are established, for example, by use of Eq. (8.2-25).

The foregoing concepts are illustrated in Fig. 8.21. Parts (*a*) through (*e*) of this figure show a rectangular object (denoted by 0's) that is moving to the right with constant velocity of 1 pixel/frame. The images shown represent instants of time corresponding to one pixel displacement. Figure 8.21*a* is the reference image frame, Figs. 8.21*b* to *d* are frames 2 to 4 in the sequence, and Fig. 8.21*e* is the eleventh frame. Figures 8.21*f* to *i* are the corresponding accumulative images, which may be explained as follows. In Fig. 8.21*f*, the left column of 1's is due to differences between the object in Fig. 8.21*a* and the background in Fig. 8.21*b*. The right column of 1's is caused by differences between the background in the reference image and the leading edge of the moving object. By the time of the fourth frame (Fig. 8.21*d*), the first nonzero column of the accumulative difference image shows three counts, indicating three total differences between that column in the reference image and the corresponding column in the subsequent frames. Finally, Fig. 8.21*a* shows a total of 10 (represented by "A" in hexadecimal) changes at that location. The other entries in that figure are explained in a similar manner.

It is often useful to consider three types of accumulative difference images: absolute (AADI), positive (PADI), and negative (NADI). The latter two quantities

```
        9
       10   00000000
       11   00000000
(a)    12   00000000
       13   00000000
       14   00000000
       15   00000000
       16
        9                                 9
       10    00000000                    10   1         1
       11    00000000                    11   1         1
(b)    12    00000000                    12   1         1             (f)
       13    00000000                    13   1         1
       14    00000000                    14   1         1
       15    00000000                    15   1         1
       16                                16
        9                                 9
       10     00000000                   10   21        21
       11     00000000                   11   21        21
(c)    12     00000000                   12   21        21            (g)
       13     00000000                   13   21        21
       14     00000000                   14   21        21
       15     00000000                   15   21        21
       16                                16
        9                                 9
       10      00000000                  10   321       321
       11      00000000                  11   321       321
(d)    12      00000000                  12   321       321           (h)
       13      00000000                  13   321       321
       14      00000000                  14   321       321
       15      00000000                  15   321       321
       16                                16
        9                                 9
       10            00000000            10   A98765438887654321
       11            00000000            11   A98765438887654321
(e)    12            00000000            12   A98765438887654321     (i)
       13            00000000            13   A98765438887654321
       14            00000000            14   A98765438887654321
       15            00000000            15   A98765438887654321
       16                                16
```

Figure 8.21 (*a*) Reference image frame. (*b*) to (*e*) Frames 2, 3, 4, and 11. (*f*) to (*i*) Accumulative difference images for frames 2, 3, 4, and 11. (From Jain [1981], ©IEEE.)

are obtained by using Eq. (8.2-25) without the absolute value and by using the reference frame instead of $f(x, y, t_i)$. Assuming that the intensities of an object are numerically greater than the background, if the difference is positive, it is compared with a positive threshold; if it is negative, the difference is compared with a negative threshold. This definition is reversed if the intensities of the object are less than the background.

Example: Figure 8.22*a* to *c* show the AADI, PADI, and NADI for a 20×20 pixel object whose intensity is greater than the background, and which is moving with constant velocity in a south-easterly direction. It is important to note that the spatial growth of the PADI stops when the object is displaced from its original position. In other words, when an object whose intensities are greater than the background is completely displaced from its position in the reference image, there will be no new entries generated in the positive accumulative difference image. Thus, when its growth stops, the PADI gives the initial location of the object in the reference frame. As will be seen below, this property can be used to advantage in creating a reference from a dynamic sequence of images. It is also noted in Fig. 8.22 that the AADI contains the regions of both the PADI and NADI, and that the entries in these images give an indication of the speed and direction of object movement. The images in Fig. 8.22 are shown in intensity-coded form in Fig. 8.23. □

Establishing a Reference Image. A key to the success of the techniques discussed in the previous two sections is having a reference image against which subsequent comparisons can be made. As indicated earlier, 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 mov-

```
99999999999999999999
99999999999999999999
99999999999999999999
9988888888888888888811
9988888888888888888811
9988888888888888888811
998877777777777777772211
998877777777777777772211
998877777777777777772211
99887766666666666666332211
99887766666666666666332211
99887766666666666666332211
9988776655555555555544332211
9988776655555555555544332211
9988776655555555555544332211
998877665544444444445544332211
998877665544444444445544332211
998877665544444444445544332211
99887766554433333333665544332211
99887766554433333333665544332211
  112233445566666666665544332211
  11223344556677777777665544332211
  11223344556677777777665544332211
    112233445566666666665544332211
    11223344556677777777665544332211
    11223344556677777777665544332211
      112233445566666666665544332211
      11223344556677777777665544332211
      11223344556677777777665544332211
        112233445566666666665544332211
        112233445566666666665544332211
        112233445566666666665544332211
          1122334455555555555544332211
          1122334455555555555544332211
          1122334455555555555544332211
            11223344444444444444332211
            11223344444444444444332211
            11223344444444444444332211
              112233333333333333332211
              112233333333333333332211
              112233333333333333332211
                1122222222222222222211
                1122222222222222222211
                1122222222222222222211
                  11111111111111111111
                  11111111111111111111
                  11111111111111111111
```

(*a*)

```
99999999999999999999
99999999999999999999
99999999999999999999
99888888888888888888
99888888888888888888
99888888888888888888
99887777777777777777
99887777777777777777
99887777777777777777
99887766666666666666
99887766666666666666
99887766666666666666
99887766555555555555
99887766555555555555
99887766555555555555
99887766554444444444
99887766554444444444
99887766554444444444
99887766554433333333
99887766554433333333
```

(*b*)

```
                  11
                  11
                  11
                  2211
                  2211
                  2211
                  332211
                  332211
                  332211
                  44332211
                  44332211
                  44332211
                  5544332211
                  5544332211
                  5544332211
                  665544332211
                  665544332211
112233445566666666665544332211
11223344556677777777665544332211
11223344556677777777665544332211
  112233445566666666665544332211
  11223344556677777777665544332211
  11223344556677777777665544332211
    112233445566666666665544332211
    11223344556677777777665544332211
    11223344556677777777665544332211
      112233445566666666665544332211
      112233445566666666665544332211
      112233445566666666665544332211
        1122334455555555555544332211
        1122334455555555555544332211
        1122334455555555555544332211
          11223344444444444444332211
          11223344444444444444332211
          11223344444444444444332211
            112233333333333333332211
            112233333333333333332211
            112233333333333333332211
              1122222222222222222211
              1122222222222222222211
              1122222222222222222211
                11111111111111111111
                11111111111111111111
                11111111111111111111
```

(*c*)

Figure 8.22 (*a*) Absolute, (*b*) positive, and (*c*) negative accumulative difference images for a 20 × 20 pixel object with intensity greater than the background and moving in a southeasterly direction. (From Jain [1983], courtesy of R. Jain.)

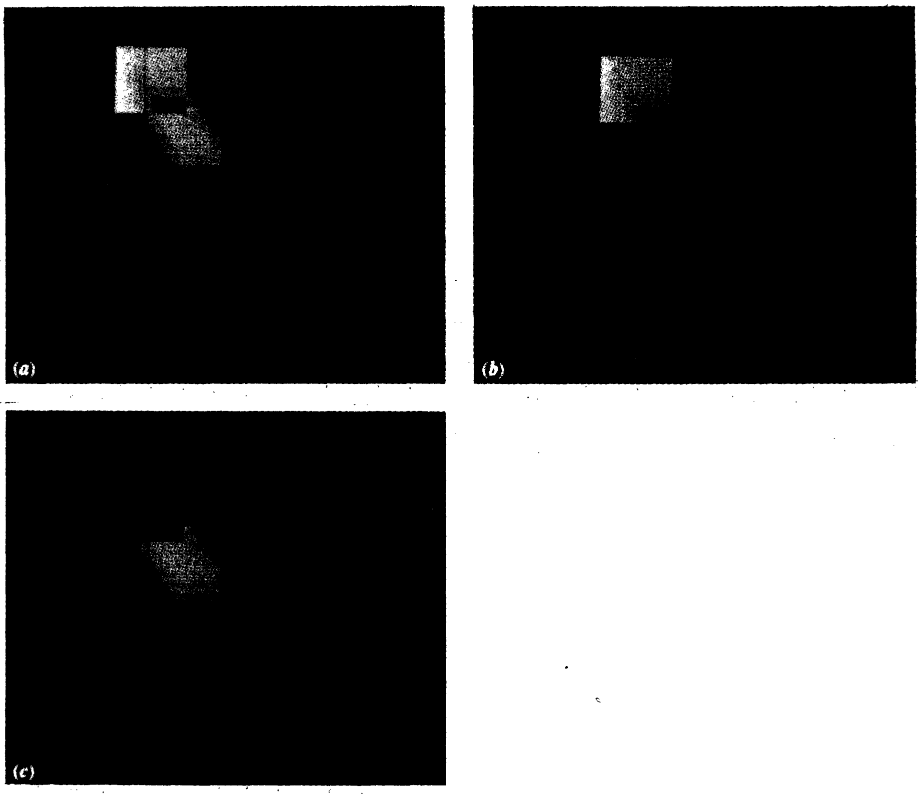


Figure 8.23 Intensity-coded accumulative difference images for Fig. 8.22. (*a*) AADI, (*b*) PADI, and (*c*) NADI. (From Jain [1983], courtesy of R. Jain.)

ing objects. The noise problem can be handled by the filtering approach discussed earlier or by forming an accumulative difference image.

In practice, it is not always possible to obtain a reference image with only stationary elements and it becomes necessary to build a reference from a set of images containing one or more moving objects. This is particularly true in situations describing busy scenes or in cases where frequent updating is required. One procedure for generating a reference image is as follows: Suppose that we 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 growth of the PADI.

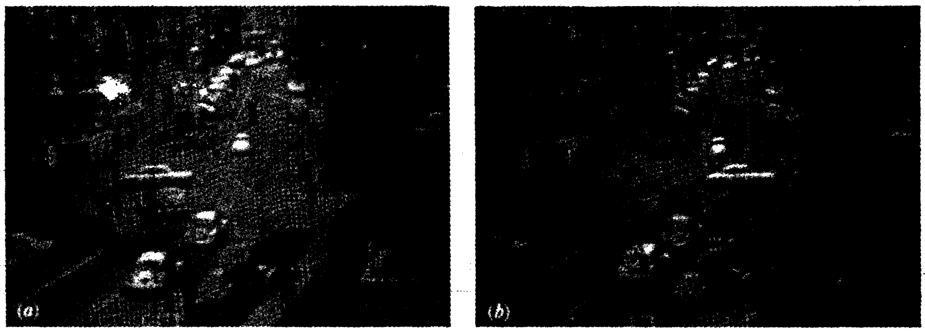


Figure 8.24 Two image frames of a traffic scene. There are two principal moving objects: a white car in the middle of the picture and a pedestrian on the lower left. (From Jain [1981], ©IEEE.)

Example: An illustration of the approach just discussed is shown in Figs. 8.24 and 8.25. Figure 8.24 shows 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 principal moving features are the automobile moving from left to right and a pedestrian crossing the street in the bottom left of the picture. Removal of the moving automobile is shown in Fig. 8.25*a*. The pedestrian is removed in Fig. 8.25*b*. □

8.3 DESCRIPTION

The description problem in vision is one of extracting features from an object for the purpose of recognition. Ideally, descriptors should be independent of object size, location, and orientation and should contain enough discriminatory informa-

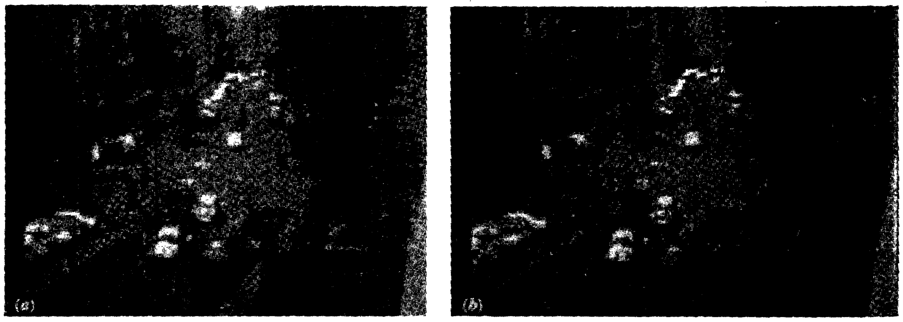


Figure 8.25 (*a*) Image with automobile removed and background restored. (*b*) Image with pedestrian removed and background restored. The latter image can be used as a reference. (From Jain [1981], ©IEEE.)

tion to uniquely identify one object from another. Description is a central issue in the design of vision systems in the sense that descriptors affect not only the complexity of recognition algorithms but also their performance. In Secs. 8.3.1, 8.3.2, and 8.4, respectively, we subdivide descriptors into three principal categories: boundary descriptors, regional descriptors, and descriptors suitable for representing three-dimensional structures.

8.3.1 Boundary Descriptors

Chain Codes. Chain codes are used to represent a boundary as a set of straight line segments of specified length and direction. Typically, this representation is established on a rectangular grid using 4- or 8-connectivity, as shown in Fig. 8.26. The length of each segment is established by the resolution of the grid, and the directions are given by the code chosen. It is noted that two bits are sufficient to represent all directions in the 4-code, and three bits are needed for the 8-code. Of course, it is possible to specify chain codes with more directions, but the codes shown in Fig. 8.26 are the ones most often used in practice.

To generate the chain code of a given boundary we first select a grid spacing, as shown in Fig. 8.27*a*. Then, if a cell is more than a specified amount (usually 50 percent) inside the boundary, we assign a 1 to that cell; otherwise, we assign it a 0. Figure 8.27*b* illustrates this process, where cells with value 1 are shown dark. Finally, we code the boundary between the two regions using the direction codes given in Fig. 8.26*a*. The result is shown in Fig. 8.27*c*, where the coding was started at the dot and proceeded in a clockwise direction. An alternate procedure is to subdivide the boundary into segments of equal length (i.e., each segment having the same number of pixels), connecting the endpoints of each segment with a straight line, and assigning to each line the direction closest to one of the allowed chain-code directions. An example of this approach using four directions is shown in Fig. 8.28.

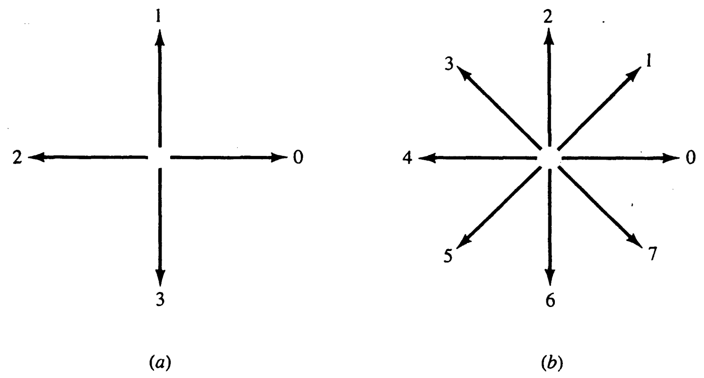


Figure 8.26 (*a*) 4-directional chain code. (*b*) 8-directional chain code.

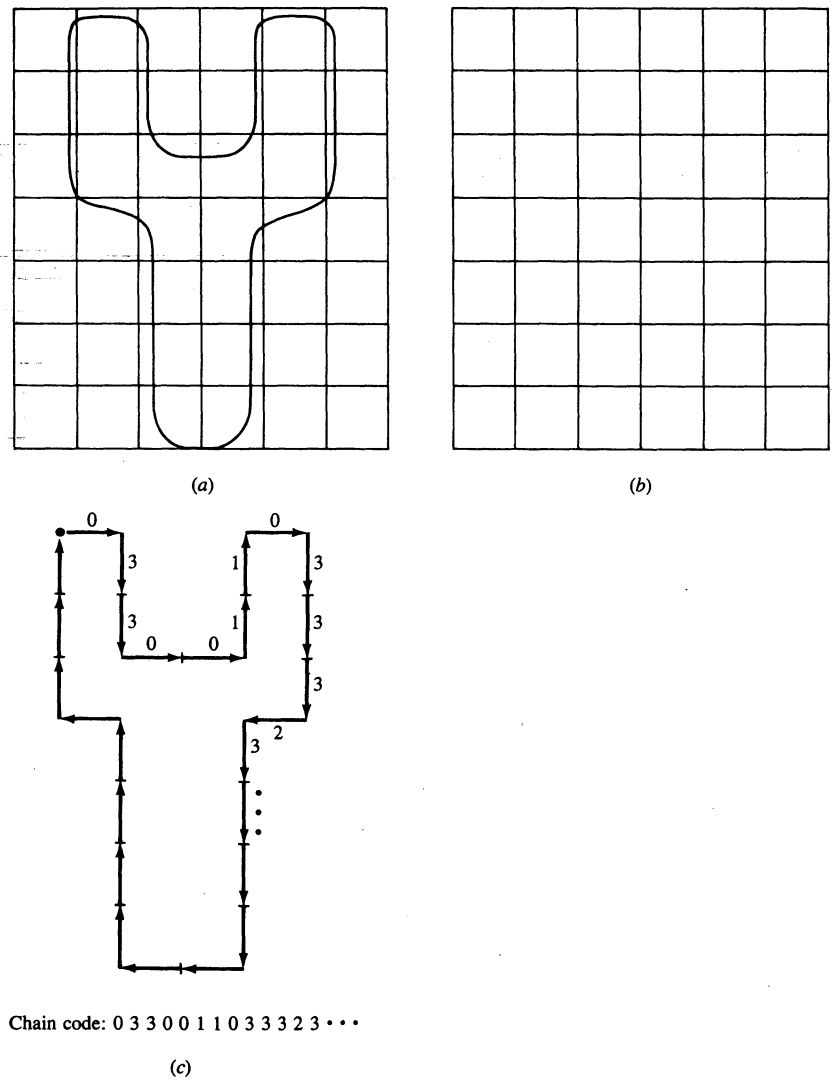


Figure 8.27 Steps in obtaining a chain code. The dot in (*c*) indicates the starting point.

It is important to note that the chain code of a given boundary depends upon the starting point. It is possible, however, to normalize the code by a straightforward procedure: Given a chain code generated by starting in an arbitrary position, we treat it as a circular sequence of direction numbers and redefine the starting point so that the resulting sequence of numbers forms an integer of minimum magnitude. We can also normalize for rotation by using the first difference of the chain code, instead of the code itself. The difference is computed simply by counting (in a counterclockwise manner) the number of directions that separate two

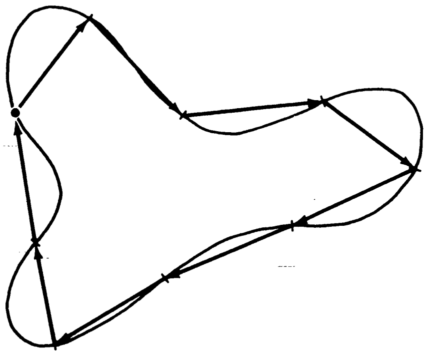


Figure 8.28 Generation of chain code by boundary subdivision.

adjacent elements of the code. For instance, the first difference of the 4-direction chain code 10103322 is 3133030. If we treat the code as a circular sequence, then the first element of the difference is computed using the transition between the last and first components of the chain. In this example the result is 33133030. Size normalization can be achieved by subdividing all object boundaries into the same number of equal segments and adjusting the code segment lengths to fit these subdivision, as illustrated in Fig. 8.28.

The preceding normalizations are exact only if the boundaries themselves are invariant to rotation and scale change. In practice, this is seldom the case. For instance, the same object digitized in two different orientations will in general have different boundary shapes, with the degree of dissimilarity being proportional to image resolution. This effect can be reduced by selecting chain elements which are large in proportion to the distance between pixels in the digitized image or by orienting the grid in Fig. 8.27 along the principal axes of the object to be coded. This is discussed below in the section on shape numbers.

Signatures. A signature is a one-dimensional functional representation of a boundary. There are a number of ways to generate signatures. One of the simplest is to plot the distance from the centroid to the boundary as a function of angle, as illustrated in Fig. 8.29. Signatures generated by this approach are obviously dependent on size and starting point. Size normalization can be achieved simply by normalizing the $r(\theta)$ curve to, say, unit maximum value. The starting-point problem can be solved by first obtaining the chain code of the boundary and then using the approach discussed in the previous section.

Distance vs. angle is, of course, not the only way to generate a signature. We could, for example, traverse the boundary and plot the angle between a line tangent to the boundary and a reference line as a function of position along the boundary (Ambler et al. [1975]). The resulting signature, although quite different

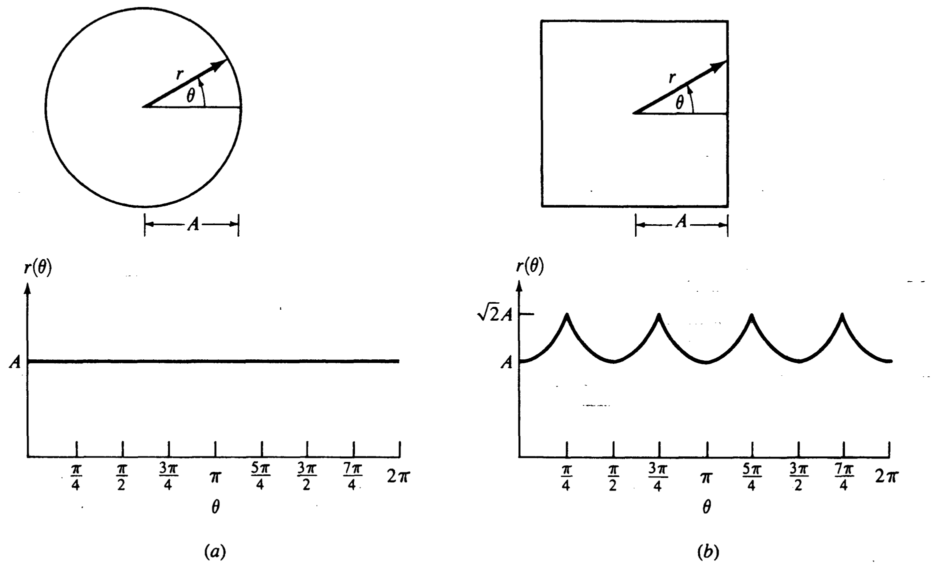


Figure 8.29 Two simple boundary shapes and their corresponding distance vs. angle signatures. In (*a*), $r(\theta)$ is constant, while in (*b*), $r(\theta) = A \sec \theta$.

from the $r(\theta)$ curve, would carry information about basic shape characteristics. For instance, horizontal segments in the curve would correspond to straight lines along the boundary since the tangent angle would be constant there. A variation of this approach is to use the so-called *slope density function* as a signature (Nahin [1974]). This function is simply a histogram of tangent angle values. Since a histogram is a measure of concentration of values, the slope density function would respond strongly to sections of the boundary with constant tangent angles (straight or nearly straight segments) and have deep valleys in sections producing rapidly varying angles (corners or other sharp inflections).

Once a signature has been obtained, we are still faced with the problem of describing it in a way that will allow us to differentiate between signatures corresponding to different boundary shapes. This problem, however, is generally easier because we are now dealing with one-dimensional functions. An approach often used to characterize a signature is to compute its *moments*. Suppose that we treat a as a discrete random variable denoting amplitude variations in a signature, and let $p(a_i)$, $i = 1, 2, \ldots, K$, denote the corresponding histogram, where K is the number of discrete amplitude increments of a. The nth moment of a about its mean is defined as

$$\mu_n(a) = \sum_{i=1}^{K} (a_i - m)^n p(a_i) \tag{8.3-1}$$

where

$$m = \sum_{i=1}^{K} a_i p(a_i) \tag{8.3-2}$$

The quantity m is recognized as the mean or average value of a and μ_2 as its variance. Only the first few moments are generally required to differentiate between signatures of clearly distinct shapes.

Polygonal Approximations. A digital boundary can be approximated with arbitrary accuracy by a polygon. For a closed curve, the approximation is exact when the number of segments in the polygon is equal to the number of points in the boundary so that each pair of adjacent points defines a segment in the polygon. In practice, the goal of a polygonal approximation is to capture the "essence" of the boundary shape with the fewest possible polygonal segments. Although this problem is in general not trivial and can very quickly turn into a time-consuming iterative search, there are a number of polygonal approximation techniques whose modest complexity and processing requirements makes them well-suited for robot vision applications. Several of these techniques are presented in this section.

We begin the discussion with a method proposed by Sklansky et al. [1972] for finding minimum-perimeter polygons. The procedure is best explained by means of an example. With reference to Fig. 8.30, suppose that we enclose a given boundary by a set of concatenated cells, as shown in Fig. 8.30*a*. We can visualize this enclosure as consisting of two walls corresponding to the outside and inside boundaries of the strip of cells, and we can think of the object boundary as a rubberband contained within the walls. If we now allow the rubberband to shrink, it will take the shape shown in Fig. 8.30*b*, thus producing a polygon of minimum perimeter which fits in the geometry established by the cell strip. If the cells are chosen so that each cell encompasses only one point on the boundary, then the error in each cell between the original boundary and the rubberband approximation would be at most $\sqrt{2}d$, where d is the distance between pixels. This error can be reduced in half by forcing each cell to be centered on its corresponding pixel.

Merging techniques based on error or other criteria have been applied to the problem of polygonal approximation. One approach is to merge points along a boundary until the least-squares error line fit of the points merged thus far exceeds a preset threshold. When this occurs, the parameters of the line are stored, the error is set to zero, and the procedure is repeated, merging new points along the boundary until the error again exceeds the threshold. At the end of the procedure the intersections of adjacent line segments form the vertices of a polygon. One of the principal difficulties with this method is that vertices do not generally correspond to inflections (such as corners) in the boundary because a new line is not started until the error threshold is exceeded. If, for instance, a long straight line were being tracked and it turned a corner, a number (depending on the threshold) of points past the corner would be absorbed before the threshold is exceeded.

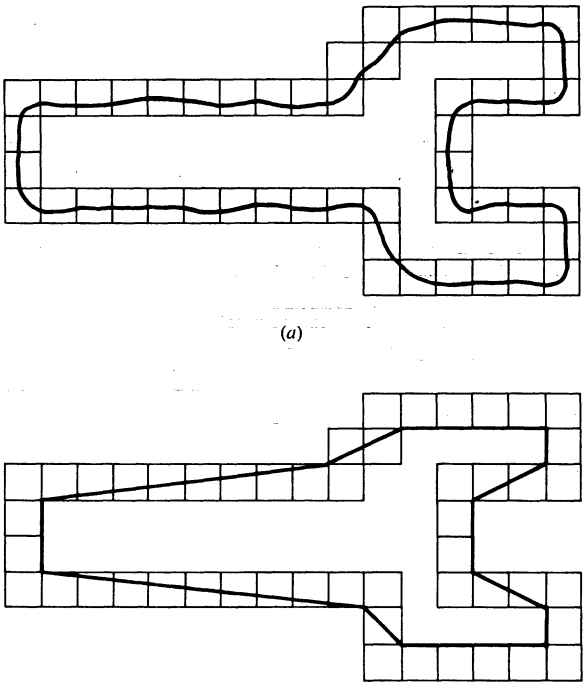


Figure 8.30 (*a*) Object boundary enclosed by cells. (*b*) Minimum-perimeter polygon.

It is possible, however, to use splitting along with merging to alleviate this difficulty.

One approach to boundary segment *splitting* is to successively subdivide a segment into two parts until a given criterion is satisfied. For instance, we might require that the maximum perpendicular distance from a boundary segment to the line joining its two endpoints not exceed a preset threshold. If it does, the furthest point becomes a vertex, thus subdividing the initial segment into two subsegments. This approach has the advantage that it "seeks" prominent inflection points. For a closed boundary, the best starting pair of points is usually the two furthest points in the boundary. An example is shown in Fig. 8.31. Part (*a*) of this figure shows an object boundary, and Fig. 8.31*b* shows a subdivision of this boundary (solid line) about its furthest points. The point marked *c* has the largest perpendicular distance from the top segment to line *ab*. Similarly, point *d* has the largest distance in the bottom segment. Figure 8.31*c* shows the result of using the splitting procedure with a threshold equal to 0.25 times the length of line *ab*. Since no point in the new boundary segments has a perpendicular distance (to its corresponding straight-line segment) which exceeds this threshold, the procedure terminates with the polygon shown in Fig. 8.31*d*.

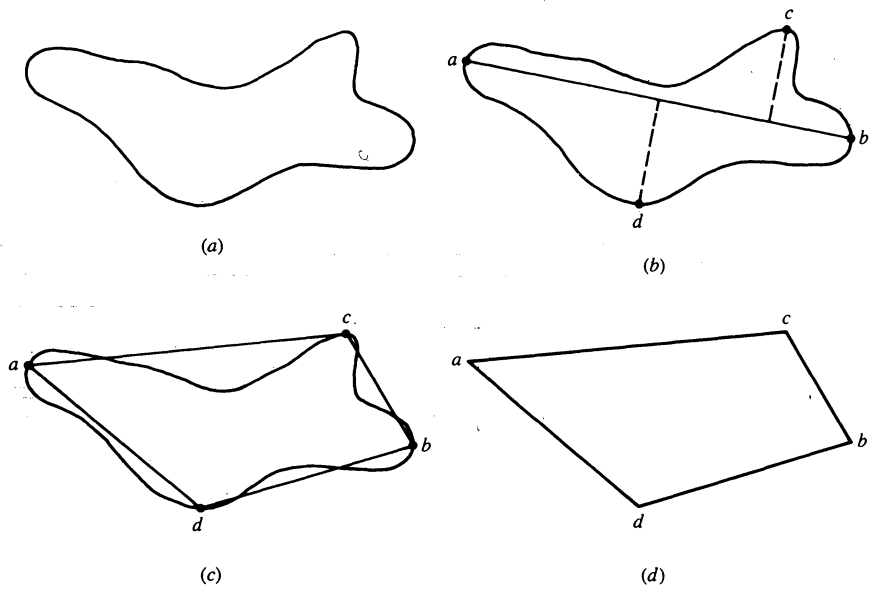
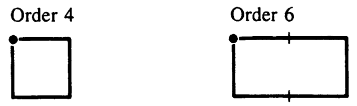


Figure 8.31 (*a*) Original boundary. (*b*) Boundary subdivided along furthest points. (*c*) Joining of vertices by straight line segments. (*d*) Resulting polygon.

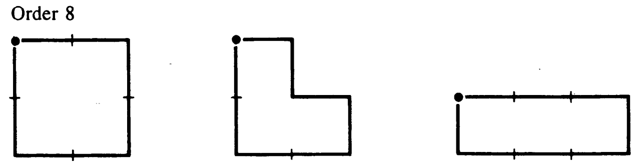
We point out before leaving this section that a considerable amount of work has been done in the development of techniques which combine merging and splitting. A comprehensive discussion of these methods is given by Pavlidis [1977].

Shape Numbers. A chain-coded boundary has several first differences, depending on the starting point. The *shape number* of such a boundary, based on the 4-directional code of Fig. 8.26*a* is defined as the first difference of smallest magnitude. The *order*, *n*, of a shape number is defined as the number of digits in its representation. It is noted that *n* is even for a closed boundary, and that its value limits the number of possible different shapes. Figure 8.32 shows all the shapes of orders 4, 6, and 8, along with their chain-code representations, first differences, and corresponding shape numbers. Note that the first differences were computed by treating the chain codes as a circular sequence in the manner discussed earlier.

Although the first difference of a chain code is independent of rotation, the coded boundary in general will depend on the orientation of the coding grid shown in Fig. 8.27*a*. One way to normalize the grid orientation is as follows. The *major axis* of a boundary is the straight-line segment joining the two points furthest away from each other. The *minor axis* is perpendicular to the major axis and of length such that a box could be formed that just encloses the boundary. The ratio of the major to minor axis is called the *eccentricity* of the boundary, and the rectangle just described is called the *basic rectangle*. In most cases a unique shape number will be obtained by aligning the chain-code grid with the sides of the basic rectan-



	Order 4	Order 6
Chain code:	0 3 2 1	0 0 3 2 2 1
Difference:	3 3 3 3	3 0 3 3 0 3
Shape number:	3 3 3 3	0 3 3 0 3 3



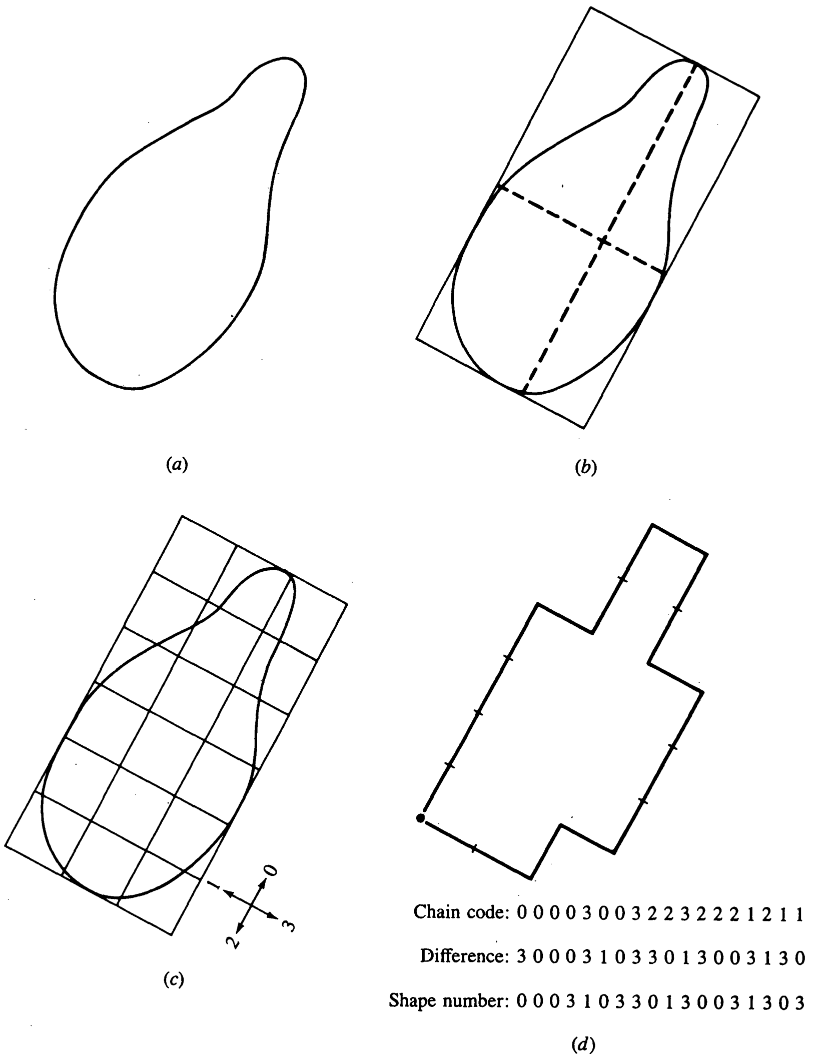
Chain code:	0 0 3 3 2 2 1 1	0 3 0 3 2 2 1 1	0 0 0 3 2 2 2 1
Difference:	3 0 3 0 3 0 3 0	3 3 1 3 3 0 3 0	3 0 0 3 3 0 0 3
Shape number:	0 3 0 3 0 3 0 3	0 3 0 3 3 1 3 3	0 0 3 3 0 0 3 3

Figure 8.32 All shapes of order 4, 6, and 8. The directions are from Fig. 8.26, and the dot indicates the starting point.

gle. Freeman and Shapira [1975] give an algorithm for finding the basic rectangle of a closed, chain-coded curve.

In practice, given a desired shape order, we find the rectangle of order n whose eccentricity best approximates that of the basic rectangle, and use this new rectangle to establish the grid size. For example, if $n = 12$, all the rectangles of order 12 (i.e., those whose perimeter length is 12) are 2×4, 3×3, and 1×5. If the eccentricity of the 2×4 rectangle best matches the eccentricity of the basic rectangle for a given boundary, we establish a 2×4 grid centered on the basic rectangle and use the procedure already outlined to obtain the chain code. The shape number follows from the first difference of this code, as indicated above. Although the order of the resulting shape number will usually be equal to n because of the way the grid spacing was selected, boundaries with depressions comparable with this spacing will sometimes yield shape numbers of order greater than n. In this case, we specify a rectangle of order lower than n and repeat the procedure until the resulting shape number is of order n.

Example: Suppose that we specify $n = 18$ for the boundary shown in Fig. 8.33*a*. In order to obtain a shape number of this order we follow the steps discussed above. First we find the basic rectangle, as shown in Fig. 8.33*b*. The closest rectangle of order 18 is a 3×6 rectangle, and so we subdivide the basic rectangle as shown in Fig. 8.33*c*, where it is noted that the chain code directions are aligned with the resulting grid. Finally, we obtain the chain code and use its first difference to compute the shape number, as shown in Fig. 8.33*d*. □



Chain code: 0 0 0 0 3 0 0 3 2 2 3 2 2 2 1 2 1 1

Difference: 3 0 0 0 3 1 0 3 3 0 1 3 0 0 3 1 3 0

Shape number: 0 0 0 3 1 0 3 3 0 1 3 0 0 3 1 3 0 3

Figure 8.33 Steps in the generation of a shape number.

Fourier Descriptors. The discrete, one-dimensional Fourier transform given in Eq. (7.6-4) can often be used to describe a two-dimensional boundary. Suppose that M points on a boundary are available. If, as shown in Fig. 8.34, we view this boundary as being in the complex plane, then each two-dimensional boundary point (x, y) is reduced to the one-dimensional complex number $x + jy$. The sequence of points along the boundary forms a function whose Fourier transform is $F(u)$, $u = 0, 1, 2, \ldots, M - 1$. If M is an integer power of 2, $F(u)$ can be computed using an FFT algorithm, as discussed in Sec. 7.6.1. The motivation for this approach is that only the first few components of $F(u)$ are generally required

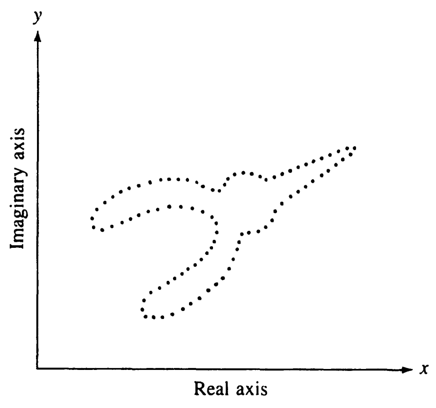


Figure 8.34 Representation of a region boundary in the frequency domain.

to distinguish between shapes that are reasonably distinct. For example, the objects shown in Fig. 8.35 can be differentiated by using less than 10 percent of the elements of the complete Fourier transform of their boundaries.

The Fourier transform is easily normalized for size, rotation, and starting point on the boundary. To change the size of a contour we simply multiply the components of $F(u)$ by a constant. Due to the linearity of the Fourier transform pair, this is equivalent to multiplying (scaling) the boundary by the same factor. Rotation by an angle θ is similarly handled by multiplying the elements of $F(u)$ by $\exp(j\theta)$. Finally, it can be shown that shifting the starting point of the contour in the spatial domain corresponds to multiplying the kth component of $F(u)$ by $\exp(jkT)$, where T is in the interval $[0, 2\pi]$. As T goes from 0 to 2π, the starting point traverses the entire contour once. This information can be used as the basis for normalization (Gonzalez and Wintz [1977]).

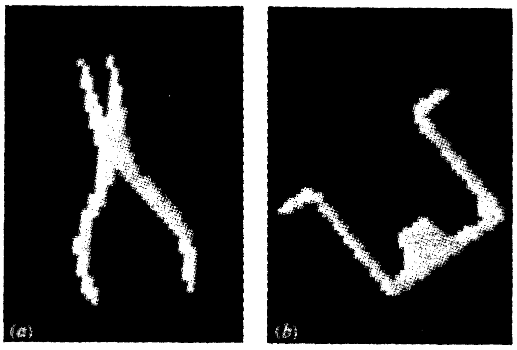


Figure 8.35 Two shapes easily distinguishable by Fourier descriptors. (From Persoon and Fu [1977], ©IEEE.)

8.3.2 Regional Descriptors

A region of interest can be described by the shape of its boundary, as discussed in Sec. 8.3.1, or by its internal characteristics, as indicated in the following discussion. It is thus important to note that the methods developed in both of these sections are applicable to region descriptions.

Some Simple Descriptors. A number of existing industrial vision systems are based on regional descriptors which are rather simple in nature and thus are attractive from a computational point of view. As might be expected, the use of these descriptors is limited to situations in which the objects of interest are so distinct that a few global descriptors are sufficient for their characterization.

The *area* of a region is defined as the number of pixels contained within its boundary. This is a useful descriptor when the viewing geometry is fixed and objects are always analyzed approximately the same distance from the camera. A typical application is the recognition of objects moving on a conveyor belt past a vision station.

The *major* and *minor* axes of a region are defined in terms of its boundary (see Sec. 8.3.1) and are useful for establishing the orientation of an object. The ratio of the lengths of these axes, called the *eccentricity* of the region, is also an important global descriptor of its shape.

The *perimeter* of a region is the length of its boundary. Although the perimeter is sometimes used as a descriptor, its most frequent application is in establishing a measure of *compactness* of a region, defined as perimeter^2/area. It is of interest to note that compactness is a dimensionless quantity (and thus is insensitive to scale changes) and that it is minimum for a disk-shaped region.

A *connected region* is a region in which all pairs of points can be connected by a curve lying entirely in the region. For a set of connected regions, some of which may have holes, it is useful to consider the *Euler number* as a descriptor. The Euler number is defined simply as the number of connected regions minus the number of holes. As an example, the Euler numbers of the letters A and B are 0 and -1, respectively. A number of other regional descriptors are discussed below.

Texture. The identification of objects or regions in an image can often be accomplished, at least partially, by the use of texture descriptors. Although no formal definition of texture exists, we intuitively view this descriptor as providing quantitative measures of properties such as smoothness, coarseness, and regularity (some examples are shown in Fig. 8.36). The two principal approaches to texture description are statistical and structural. Statistical approaches yield characterizations of textures as being smooth, coarse, grainy, and so on. Structural techniques, on the other hand, deal with the arrangement of image primitives, such as the description of texture based on regularly spaced parallel lines.

One of the simplest approaches for describing texture is to use moments of the intensity histogram of an image or region. Let z be a random variable denoting

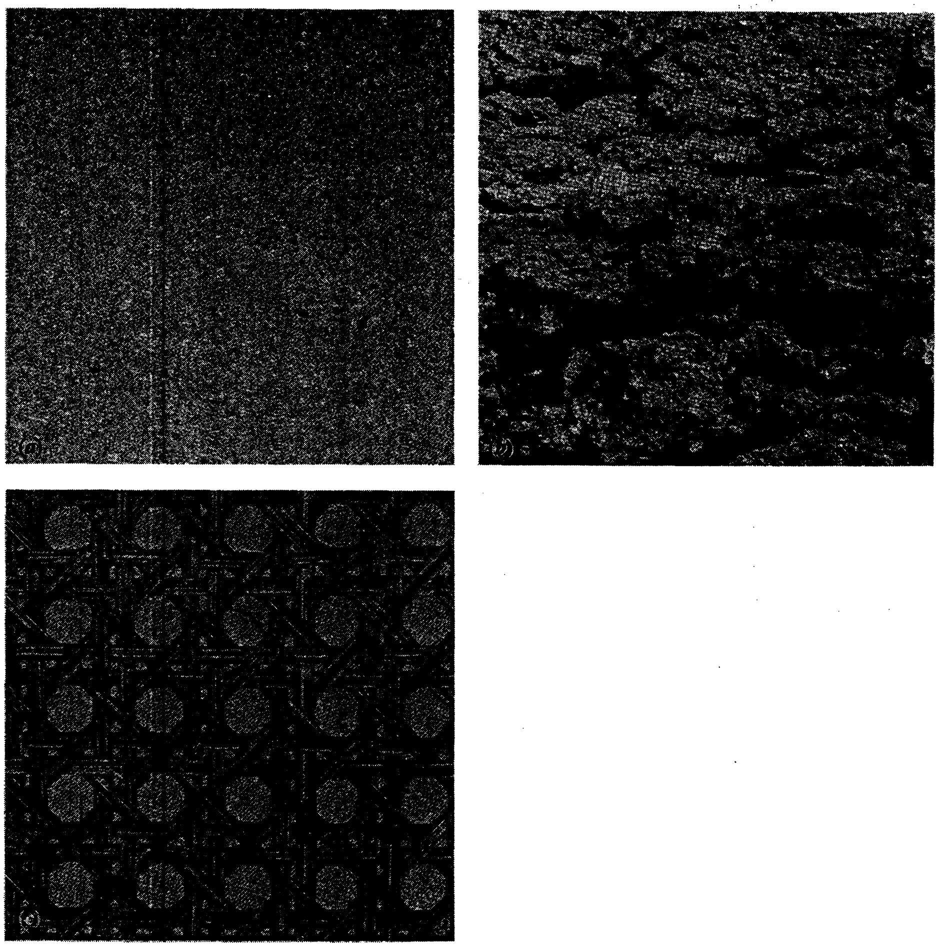


Figure 8.36 Examples of (*a*) smooth, (*b*) coarse, and (*c*) regular texture.

discrete image intensity, and let $p(z_i)$, $i = 1, 2, \ldots, L$, be the corresponding histogram, where L is the number of distinct intensity levels. As indicated in Sec. 8.3.1, the nth moment of z about the mean is defined as

$$\mu_n(z) = \sum_{i=1}^{L} (z_i - m)^n p(z_i) \tag{8.3-3}$$

where m is the mean value of z (i.e., the average image intensity):

$$m = \sum_{i=1}^{L} z_i p(z_i) \tag{8.3-4}$$

It is noted from Eq. (8.3-3) that $\mu_0 = 1$ and $\mu_1 = 0$. The second moment [also called the *variance* and denoted by $\sigma^2(z)$] is of particular importance in texture description. It is a measure of intensity contrast which can be used to establish descriptors of relative smoothness. For example, the measure

$$R = 1 - \frac{1}{1 + \sigma^2(z)} \tag{8.3-5}$$

is 0 for areas of constant intensity [$\sigma^2(z) = 0$ if all z_i have the same value] and approaches 1 for large values of $\sigma^2(z)$. The third moment is a measure of the skewness of the histogram while the fourth moment is a measure of its relative flatness. The fifth and higher moments are not so easily related to histogram shape, but they do provide further quantitative discrimination of texture content.

Measures of texture computed using only histograms suffer from the limitation that they carry no information regarding the relative position of pixels with respect to each other. One way to bring this type of information into the texture analysis process is to consider not only the distribution of intensities but also the positions of pixels with equal or nearly equal intensity values. Let P be a position operator and let $\mathbf{A}$ be a $k \times k$ matrix whose element a_{ij} is the number of times that points with intensity z_i occur (in the position specified by P) relative to points with intensity z_j, with $1 \leqslant i, j \leqslant k$. For instance, consider an image with three intensities, $z_1 = 0$, $z_2 = 1$, and $z_3 = 2$, as follows:

$$\begin{matrix} 0 & 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 1 & 1 \\ 2 & 2 & 1 & 0 & 0 \\ 1 & 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{matrix}$$

If we define the position operator P as "one pixel to the right and one pixel below," then we obtain the following 3×3 matrix $\mathbf{A}$:

$$\mathbf{A} = \begin{bmatrix} 4 & 2 & 1 \\ 2 & 3 & 2 \\ 0 & 2 & 0 \end{bmatrix}$$

where, for example, a_{11} (top left) is the number of times that a point with intensity level $z_1 = 0$ appears one pixel location below and to the right of a pixel with the same intensity, while a_{13} (top right) is the number of times that a point with level $z_1 = 0$ appears one pixel location below and to the right of a point with intensity $z_3 = 2$. It is important to note that the size of $\mathbf{A}$ is determined strictly by the number of distinct intensities in the input image. Thus, application of the concepts discussed in this section usually require that intensities be requantized into a few bands in order to keep the size of $\mathbf{A}$ manageable.

Let n be the total number of point pairs in the image which satisfy P (in the above example $n = 16$). If we define a matrix $\mathbf{C}$ formed by dividing every element of **A** by n, then c_{ij} is an estimate of the joint probability that a pair of points satisfying P will have values (z_i, z_j). The matrix **C** is called a *gray-level co-occurrence matrix*, where "gray level" is used interchangeably to denote the intensity of a monochrome pixel or image. Since **C** depends on P, it is possible to detect the presence of given texture patterns by choosing an appropriate position operator. For instance, the operator used in the above example is sensitive to bands of constant intensity running at $-45°$ (note that the highest value in **A** was $a_{11} = 4$, partially due to a streak of points with intensity 0 and running at $-45°$). In a more general situation, the problem is to analyze a given **C** matrix in order to categorize the texture of the region over which **C** was computed. A set of descriptors proposed by Haralick [1979] include

1. Maximum probability:

$$\max_{i,j} (c_{ij})$$

2. Element-difference moment of order k:

$$\sum_i \sum_j (i - j)^k c_{ij}$$

3. Inverse element-difference moment of order k:

$$\frac{\sum_i \sum_j c_{ij}}{(i - j)^k} \qquad i \neq j$$

4. Entropy:

$$-\sum_i \sum_j c_{ij} \log c_{ij}$$

5. Uniformity:

$$\sum_i \sum_j c_{ij}^2$$

The basic idea is to characterize the "content" of **C** via these descriptors. For example, the first property gives an indication of the strongest response to P (as in the above example). The second descriptor has a relatively low value when the high values of **C** are near the main diagonal since the differences $(i - j)$ are smaller there. The third descriptor has the opposite effect. The fourth descriptor is a measure of randomness, achieving its highest value when all elements of **C** are equal. Conversely, the fifth descriptor is lowest when the c_{ij} are all equal. One approach for using these descriptors is to "teach" a system representative descriptor values for a set of different textures. The texture of an unknown region is then subsequently determined by how closely its descriptors match those stored in the system memory. This approach is discussed in more detail in Sec. 8.4.

The approaches discussed above are statistical in nature. As mentioned at the beginning of this section, a second major category of texture description is based on structural concepts. Suppose that we have a rule of the form $S \rightarrow aS$ which indicates that the symbol S may be rewritten as aS (e.g., three applications of this rule would yield the string $aaaS$). If we let a represent a circle (Fig. 8.37*a*) and assign the meaning of "circles to the right" to a string of the form $aaa \cdots$, then the rule $S \rightarrow aS$ allows us to generate a texture pattern of the form shown in Fig. 8.37*b*.

Suppose next that we add some new rules to this scheme: $S \rightarrow bA$, $A \rightarrow cA$, $A \rightarrow c$, $A \rightarrow bS$, $S \rightarrow a$, such that the presence of a b means "circle down" and the presence of a c means "circle to the left." We can now generate a string of the form *aaabccbaa* which corresponds to a three-by-three matrix of circles. Larger texture patterns, such as the one shown in Fig. 8.37*c* can easily be generated in the same way. (It is noted, however, that these rules can also generate structures that are not rectangular).

The basic idea in the foregoing discussion is that a simple "texture primitive" can be used to form more complex texture patterns by means of some rules which limit the number of possible arrangements of the primitive(s). These concepts lie at the heart of structural pattern generation and recognition, a topic which will be treated in considerably more detail in Sec. 8.5.

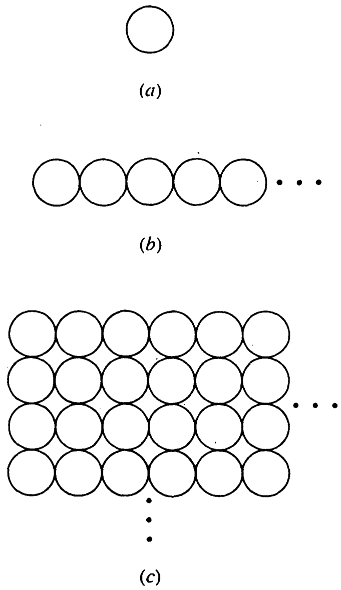


Figure 8.37 (*a*) Texture primitive. (*b*) Pattern generated by the rule $S \rightarrow aS$. (*c*) Two-dimensional texture pattern generated by this plus other rules.

Skeleton of a Region. An important approach for representing the structural shape of a plane region is to reduce it to a graph. This is often accomplished by obtaining the *skeleton* of the region via a thinning (also called *skeletonizing*) algorithm. Thinning procedures play a central role in a broad range of problems in computer vision, ranging from automated inspection of printed circuit boards to counting of asbestos fibers on air filters.

The skeleton of a region may be defined via the *medial axis transformation* (MAT) proposed by Blum [1967]. The MAT of a region R with border B is as follows. For each point p in R, we find its closest neighbor in B. If p has more than one such neighbor, then it is said to belong to the *medial axis* (skeleton) of R. It is important to note that the concept of "closest" depends on the definition of a distance (see Sec. 7.5.3) and, therefore, the results of a MAT operation will be influenced by the choice of a given metric. Some examples using the euclidean distance are shown in Fig. 8.38.

Although the MAT of a region yields an intuitively pleasing skeleton, a direct implementation of the above definition is typically prohibitive from a computational point of view because it potentially involves calculating the distance from every interior point to every point on the boundary of a region. A number of algorithms have been proposed for improving computational efficiency while, at the same time, attempting to produce a medial axis representation of a given region. Typically, these are thinning algorithms that iteratively delete edge points of a region subject to the constraints that the deletion of these points (1) does not remove endpoints, (2) does not break connectedness, and (3) does not cause excessive erosion of the region. Although some attempts have been made to use skeletons in gray-scale images (Dyer and Rosenfeld [1979], Salari and Siy [1984]) this type of representation is usually associated with binary data.

In the following discussion, we present an algorithm developed by Naccache and Shinghal [1984]. This procedure is fast, straightforward to implement, and, as will be seen below, yields skeletons that are in many cases superior to those

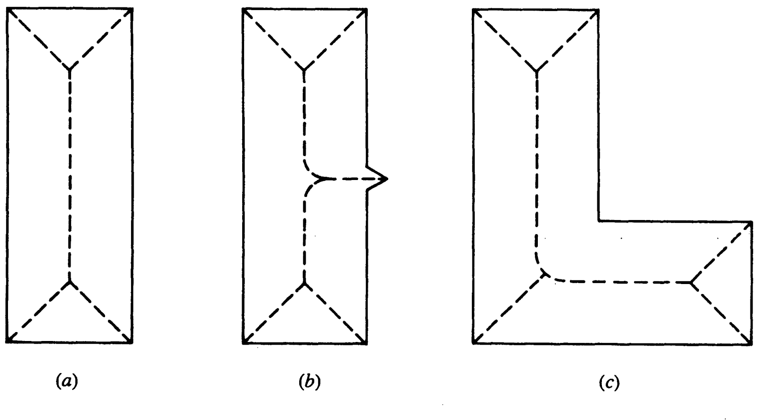


Figure 8.38 Medial axes of three simple regions.

obtained with other thinning algorithms. We begin the development with a few definitions. Assuming binary data, region points will be denoted by 1's and background points by 0's. These will be called *dark* and *light* points, respectively. An *edge point* is a dark point which has at least one light 4-neighbor. An *endpoint* is a dark point which has one and only one dark 8-neighbor. A *breakpoint* is a dark point whose deletion would break connectedness. As is true with all thinning algorithms, noise and other spurious variations along the boundary can significantly alter the resulting skeleton (Fig. 8.38*b* shows this effect quite clearly). Consequently, it is assumed that the boundaries of all regions have been smoothed prior to thinning by using, for example, the procedure discussed in Sec. 7.6.2.

With reference to the neighborhood arrangement shown in Fig. 8.39, the thinning algorithm identifies an edge point p as one or more of the following four types: (1) a *left edge point* having its left neighbor n_4 light; (2) a *right edge point* having n_0 light; (3) a *top edge point* having n_2 light; and (4) a *bottom edge point* having n_6 light. It is possible for p to be classified into more than one of these types. For example, a dark point p having n_0 and n_4 light will be a right edge point and a left edge point simultaneously. The following discussion initially addresses the identification (flagging) of left edge points that should be deleted. The procedure is then extended to the other types.

An edge point p is flagged if it is *not* an endpoint or breakpoint, or if its deletion would cause excessive erosion (as discussed below). The test for these conditions is carried out by comparing the 8-neighborhood of p against the windows shown in Fig. 8.40, where p and the asterisk are dark points and d and e are "don't care" points; that is, they can be either dark or light. If the neighborhood of p matches windows (a) to (c), two cases may arise: (1) If all d's are light, then p is an endpoint, or (2) if at least one of the d's is dark, then p is a breakpoint. In either case p should not be flagged.

The analysis of window (d) is slightly more complicated. If at least one d and e are dark, then p is a break point and should not be flagged. Other arrangements need to be considered, however. Suppose that all d's are light and the e's can be either dark or light. This condition yields the eight possibilities shown in Fig. 8.41. Configurations (a) through (c) make p an endpoint, and configuration (d) makes it a breakpoint. If p were deleted in configurations (e) and (f), it is easy to show by example that its deletion would cause excessive erosion in slanting regions of width 2. In configuration (g), p is what is commonly referred to as a

n_3	n_2	n_1
n_4	p	n_0
n_5	n_6	n_7

Figure 8.39 Notation for the neighbors of p used by the thinning algorithm. (From Naccache and Shinghal [1984], ©IEEE.)

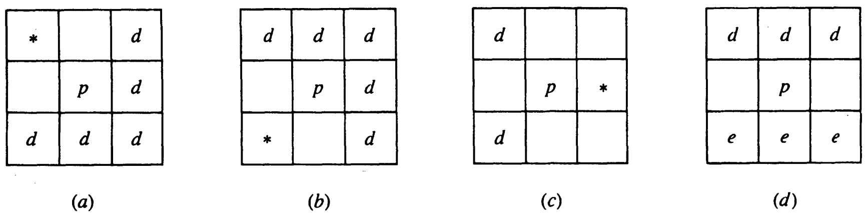


Figure 8.40 If the 8-neighborhood of a dark point p matches any of the above windows, then p is not flagged. The asterisk denotes a dark point, and d and e can be either dark or light. (From Naccache and Shinghal [1984], ©IEEE.)

spur, typically due to a short tail or protrusion in a region. Since it is assumed that the boundary of the region has been smoothed initially, the appearance of a spur during thinning is considered an important description of shape and p should not be deleted. Finally, if all isolated points are removed initially, the appearance of configuration (h) during thinning indicates that a region has been reduced to a single point; its deletion would erase the last remaining portion of the region. Similar arguments apply if the roles of d and e were reversed or if the d's and e's were allowed to assume dark and light values. The essence of the preceding discussion is that any left edge point p whose 8-neighborhood matches any of the windows shown in Fig. 8.40 should not be flagged.

Testing the 8-neighborhood of p against the four windows in Fig. 8.40 has a particularly simple boolean representation given by

$$B_4 = n_0 \cdot (n_1 + n_2 + n_6 + n_7) \cdot (n_2 + \bar{n}_3) \cdot (\bar{n}_5 + n_6) \qquad (8.3\text{-}6)$$

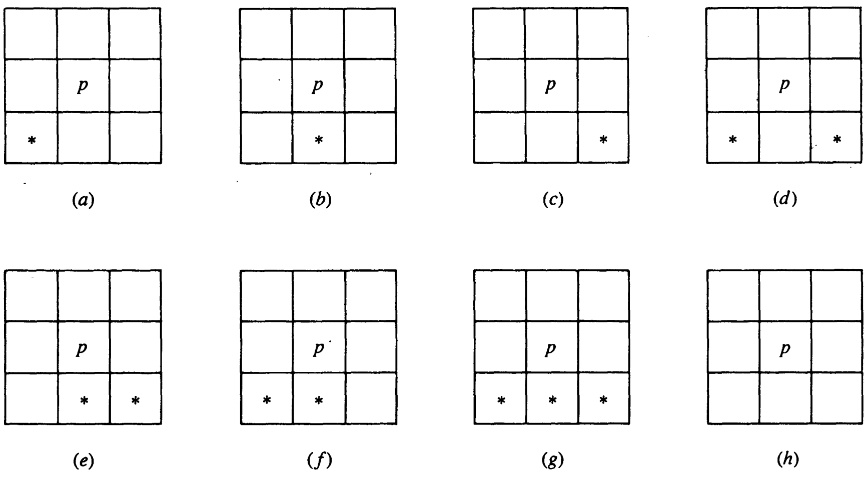


Figure 8.41 All the configurations that could exist if d is light in Fig. 8.40, and e can be dark, $*$, or light. (From Naccache and Shinghal [1984], ©IEEE.)

where the subscript on B indicates that n_4 is light (i.e., p is a left edge point), "$\cdot$" is the logical AND, "$+$" is the logical OR, "$-$" is the logical COMPLEMENT, and the n's are as defined in Fig. 8.39. Equation (8.3-6) is evaluated by letting dark, *previously unflagged* points be valued 1 (TRUE), and light or flagged points be valued 0 (FALSE). Then if B_4 is 1 (TRUE), we flag p. Otherwise, p is left unflagged. It is not difficult to show that these conditions on B_4 implement all four windows in Fig. 8.40 simultaneously.

Similar expressions are obtained for right edge points,

$$B_0 = n_4 \cdot (n_2 + n_3 + n_5 + n_6) \cdot (n_6 + \bar{n}_7) \cdot (\bar{n}_1 + n_2) \qquad (8.3\text{-}7)$$

for top edge points,

$$B_2 = n_6 \cdot (n_0 + n_4 + n_5 + n_7) \cdot (n_0 + \bar{n}_1) \cdot (\bar{n}_3 + n_4) \qquad (8.3\text{-}8)$$

and for the bottom edge points,

$$B_6 = n_2 \cdot (n_0 + n_1 + n_3 + n_4) \cdot (n_4 + \bar{n}_5) \cdot (n_0 + \bar{n}_7) \qquad (8.3\text{-}9)$$

Using the above expressions, the thinning algorithm iteratively performs two scans through the data. The scanning sequence can be either along the rows or columns of the image, but the choice will generally affect the final result. In the first scan we use B_4 and B_0 to flag left and right edge points; in the second scan we use B_2 and B_6 to flag top and bottom edge points. If no new edge points were flagged during the two scans, the algorithm stops, with the unflagged points constituting the skeleton; otherwise, the procedure is repeated. It is again noted that previously flagged dark points are treated as 0 in evaluating the boolean expressions. An alternate procedure is to set any flagged point at zero during execution of the algorithm, thus producing only skeleton and background points at the end. This approach is easier to implement, at the cost of losing all other points in the region.

Example: Figure 8.42*a* shows a binary region, and Fig. 8.42*b* shows the skeleton obtained by using the algorithm developed above. As a point of interest, Fig. 8.42*c* shows the skeleton obtained by applying to the same data another, well-known thinning algorithm (Pavlidis [1982]). The fidelity of the skeleton in Fig. 8.42*b* over that shown in Fig. 8.42*c* is evident. □

Moment Invariants. It was noted in Sec. 8.3.1 that Fourier descriptors which are insensitive to translation, rotation, and scale change can be used to describe the boundary of a region. When the region is given in terms of its interior points, we can describe it by a set of moments which are invariant to these effects.

Let $f(x, y)$ represent the intensity at point (x, y) in a region. The *moment* of order $(p + q)$ for the region is defined as

$$m_{pq} = \sum_x \sum_y x^p y^q f(x, y) \qquad (8.3\text{-}10)$$

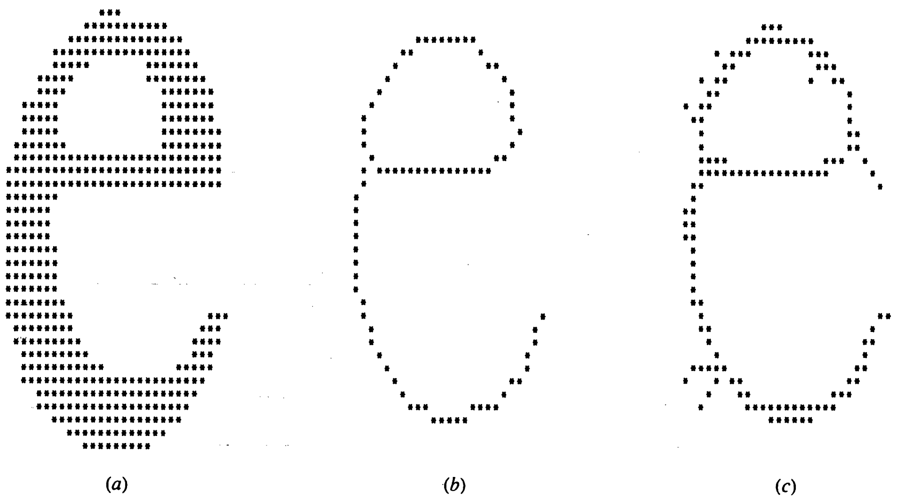


Figure 8.42 (*a*) Binary region. (*b*) Skeleton obtained using the thinning algorithm discussed in this section. (*c*) Skeleton obtained by using another algorithm. (From Naccache and Shinghal [1984], ©IEEE.)

where the summation is taken over all spatial coordinates (x, y) of points in the region. The *central moment* of order $(p + q)$ is given by

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y) \tag{8.3-11}$$

where

$$\bar{x} = \frac{m_{10}}{m_{00}} \qquad \bar{y} = \frac{m_{01}}{m_{00}} \tag{8.3-12}$$

The *normalized central moments* of order $(p + q)$ are defined as

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}} \tag{8.3-13}$$

where

$$\gamma = \frac{p + q}{2} + 1 \qquad \text{for } (p + q) = 2, 3, \ldots \tag{8.3-14}$$

The following set of *moment invariants* can be derived using only the normalized central moments of orders 2 and 3:

$$\phi_1 = \eta_{20} + \eta_{02} \tag{8.3-15}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{8.3-16}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{8.3-17}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{8.3-18}$$

$$\begin{aligned}\phi_5 = {} & (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\ & + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]\end{aligned} \tag{8.3-19}$$

$$\begin{aligned}\phi_6 = {} & (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\ & + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})\end{aligned} \tag{8.3-20}$$

$$\begin{aligned}\phi_7 = {} & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] \\ & + (3\eta_{12} - \eta_{30})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]\end{aligned} \tag{8.3-21}$$

This set of moments has been shown to be invariant to translation, rotation, and scale change (Hu [1962]).

8.4 SEGMENTATION AND DESCRIPTION OF THREE-DIMENSIONAL STRUCTURES

Attention was focused in the previous two sections on techniques for segmenting and describing two-dimensional structures. In this section we consider the problem of performing these tasks on three-dimensional (3D) scene data.

As indicated in Sec. 7.1, vision is inherently a 3D problem. It is thus widely accepted that a key to the development of versatile vision systems capable of operating in unconstrained environments lies in being able to process three-dimensional scene information. Although research in this area spans more than a 10-year history, we point out that factors such as cost, speed, and complexity have inhibited the use of three-dimensional vision techniques in industrial applications.

Three-dimensional information about a scene may be obtained in three principal forms. If range sensing is used, we obtain the (x, y, z) coordinates of points on the surface of objects. The use of stereo imaging devices yields 3D coordinates, as well as intensity information about each point. In this case, we represent each point in the form $f(x, y, z)$, where the value of f at (x, y, z) gives the intensity of that point (the term *voxel* is often used to denote a 3D point and its intensity). Finally, we may *infer* 3D relationships from a single two-dimensional image of a scene. In other words, it is often possible to deduce relationships between objects such as "above," "behind," and "in front of." Since the exact 3D location of scene points generally cannot be computed from a single view, the relationships obtained from this type of analysis are sometimes referred to as 2½ D information.

8.4.1 Fitting Planar Patches to Range Data

One of the simplest approaches for segmenting and describing a three-dimensional structure given in terms of range data points (x, y, z) is to first subdivide it into small planar "patches" and then combine these patches into larger surface elements according to some criterion. This approach is particularly attractive for polyhedral objects whose surfaces are smooth with respect to the resolution of the sensed scene.

We illustrate the basic concepts underlying this approach by means of the example shown in Fig. 8.43. Part (*a*) of this figure shows a simple scene and Fig. 8.43*b* shows a set of corresponding 3D points. These points can be assembled into small surface elements by, for example, subdividing the 3D space into cells and grouping points according to the cell which contains them. Then, we fit a plane to the group of points in each cell and calculate a unit vector which is normal to the plane and passes through the centroid of the group of points in that cell. A planar

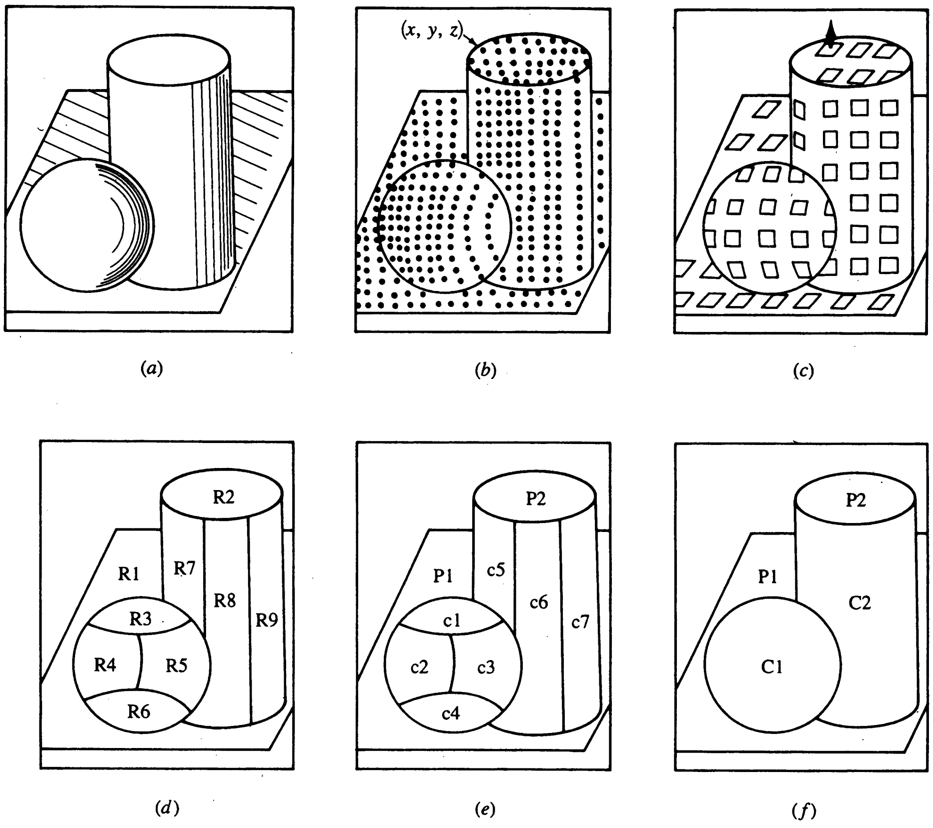


Figure 8.43 Three-dimensional surface description based on planar patches. (From Shirai [1979], ©Plenum Press.)

patch is established by the intersection of the plane and the walls of the cell, with the direction of the patch being given by the unit normal, as illustrated in Fig. 8.43*c*. All patches whose directions are similar within a specified threshold are grouped into elementary regions (R), as shown in Fig. 8.43*d*. These regions are then classified as planar (P), curved (C), or undefined (U) by using the directions of the patches within each region (for example, the patches in a planar surface will all point in essentially the same direction). This type of region classification is illustrated in Fig. 8.43*e*. Finally (and this is the hardest step), the classified regions are assembled into global surfaces by grouping adjacent regions of the same classification, as shown in Fig. 8.43*f*. It is noted that, at the end of this procedure, the scene has been segmented into distinct surfaces, and that each surface has been assigned a descriptor (e.g., curved or planar).

8.4.2 Use of the Gradient

When a scene is given in terms of voxels, the 3D gradient can be used to obtain patch representations (similar to those discussed in Sec. 8.4.1) which can then be combined to form surface descriptors. As indicated in Sec. 7.6.4, the gradient vector is normal to the direction of maximum rate of change of a function, and the magnitude of this vector is proportional to the strength of that change. These concepts are just as applicable in three dimensions and they can be used to segment 3D structures in a manner analogous to that used for two-dimensional data.

Given a function $f(x, y, z)$, its gradient vector at coordinates (x, y, z) is given by

$$\mathbf{G}[f(x, y, z)] = \begin{bmatrix} G_x \\ G_y \\ G_z \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \\ \dfrac{\partial f}{\partial z} \end{bmatrix} \tag{8.4-1}$$

The magnitude of **G** is given by

$$G[f(x, y, z)] = (G_x^2 + G_y^2 + G_z^2)^{1/2} \tag{8.4-2}$$

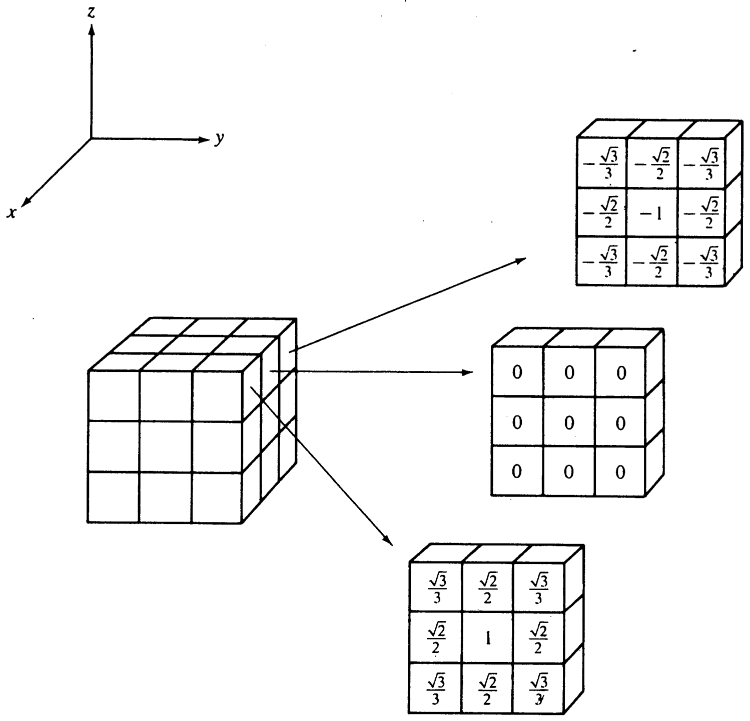
which, as indicated in Eq. (7.6-39), is often approximated by absolute values to simplify computation:

$$G[f(x, y, z)] \approx |G_x| + |G_y| + |G_z$$

The implementation of the 3D gradient can be carried out using operators analogous in form to those discussed in Sec. 7.6.4. Figure 8.44 shows a $3 \times 3 \times 3$ operator proposed by Zucker and Hummel [1981] for computing G_x. The same operator oriented along the y axis is used to compute G_y, and oriented along the z axis to compute G_z. A key property of these operators is that they yield the best (in a least-squares error sense) planar edge between two regions of different intensities in a 3D neighborhood.

The center of each operator is moved from voxel to voxel and applied in exactly the same manner as their two-dimensional counterparts, as discussed in Sec. 7.6.4. That is, the responses of these operators at any point (x, y, z) yield G_x, G_y, and G_z, which are then substituted into Eq. (8.4-1) to obtain the gradient vector at (x, y, z) and into Eq. (8.4-2) or (8.4-3) to obtain the magnitude. It is of interest to note that the operator shown in Fig. 8.44 yields a zero output in a $3 \times 3 \times 3$ region of constant intensity.

It is a straightforward procedure to utilize the gradient approach for segmenting a scene into planar patches analogous to those discussed in the previous section. It is not difficult to show that the gradient vector of a plane $ax + by + cz = 0$ has components $G_x = a$, $G_y = b$, and $G_z = c$. Since the operators discussed above yield an optimum planar fit in a $3 \times 3 \times 3$ neighborhood, it follows that the components of the vector **G** establish the direction of a planar patch in each neighborhood, while the magnitude of **G** gives an indication of abrupt changes of intensity within the patch; that is, it indicates the presence of



z
y
x
$-\frac{\sqrt{3}}{3}$ $-\frac{\sqrt{2}}{2}$ $-\frac{\sqrt{3}}{3}$
$-\frac{\sqrt{2}}{2}$ -1 $-\frac{\sqrt{2}}{2}$
$-\frac{\sqrt{3}}{3}$ $-\frac{\sqrt{2}}{2}$ $-\frac{\sqrt{3}}{3}$
0 0 0
0 0 0
0 0 0
$\frac{\sqrt{3}}{3}$ $\frac{\sqrt{2}}{2}$ $\frac{\sqrt{3}}{3}$
$\frac{\sqrt{2}}{2}$ 1 $\frac{\sqrt{2}}{2}$
$\frac{\sqrt{3}}{3}$ $\frac{\sqrt{2}}{2}$ $\frac{\sqrt{3}}{3}$

Figure 8.44 A $3 \times 3 \times 3$ operator for computing the gradient component G_x. (Adapted from Zucker and Hummel [1981], ©IEEE.)

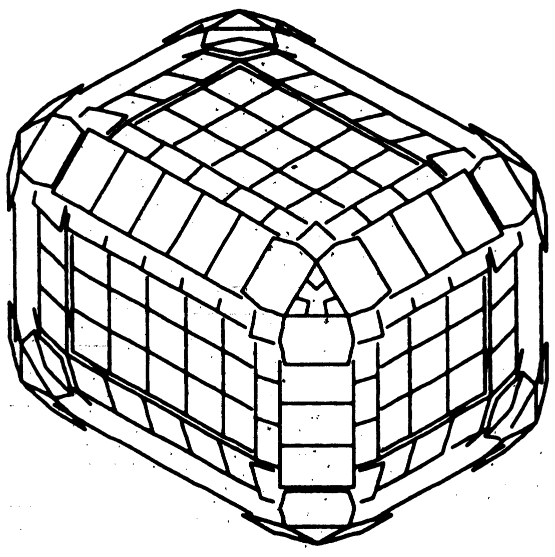


Figure 8.45 Planar patch approximation of a cube using the gradient. (From Zucker and Hummel [1981], ©IEEE.)

an intensity edge within the patch. An example of such a patch representation using the gradient operators is shown in Fig. 8.45. Since each planar patch surface passes through the center of a voxel, the borders of these patches may not always coincide. Patches that coincide are shown as larger uniform regions in Fig. 8.45.

Once patches have been obtained, they can be grouped and described in the form of global surfaces as discussed in Sec. 8.4.1. Note, however, that additional information in the form of intensity and intensity discontinuities is now available to aid the merging and description process.

8.4.3 Line and Junction Labeling

With reference to the discussion in the previous two sections, edges in a 3D scene are determined by discontinuities in range and/or intensity data. Given a set of surfaces and the edges between them, a finer description of a scene may be obtained by labeling the lines corresponding to these edges and the junctions which they form.

As illustrated in Fig. 8.46, we consider basic types of lines. A *convex line* (labeled +) is formed by the intersection of two surfaces which are part of a convex solid (e.g., the line formed by the intersection of two sides of a cube). A *concave line* (labeled −) is formed by the intersection of two surfaces belonging to two different solids (e.g., the intersection of one side of a cube with the floor). An *occluding line* (labeled with an arrow) is the edge of a surface which obscures a

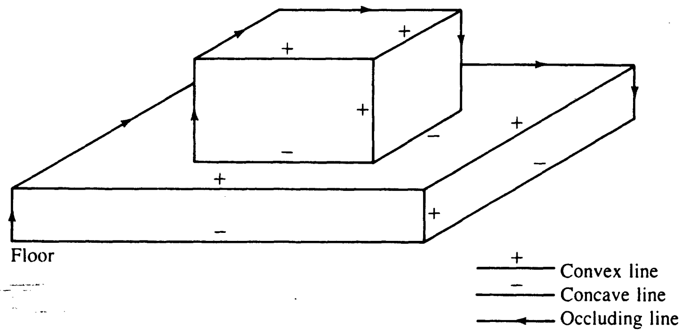


Figure 8.46 Three basic line labels.

surface. The occluding matter is to the right of the line looking in the direction of the arrow, and the occluded surface is to the left.

After the lines in a scene have been labeled, their junctions provide clues as to the nature of the 3D solids in the scene. Physical constraints allow only a few possible combinations of line labels at a junction. For example, in a polyhedral scene, no line can change its label between vertices. Violation of this rule leads to impossible physical objects, as illustrated in Fig. 8.47.

The key to using junction analysis is to form a dictionary of allowed junction types. For example, it is easily shown that the junction dictionary shown in Fig. 8.48 contains all valid labeled vertices of trihedral solids (i.e., solids in which exactly three plane surfaces come together at each vertex). Once the junctions in a scene have been classified according to their match in the dictionary, the objective is to group the various surfaces into objects. This is typically accomplished via a set of heuristic rules designed to interpret the labeled lines and sequences of neighboring junctions. The basic concept underlying this approach can be illustrated with the aid of Fig. 8.49. We note in Fig. 8.49*b* that the blob is composed entirely of an occluding boundary, with the exception of a short concave line,

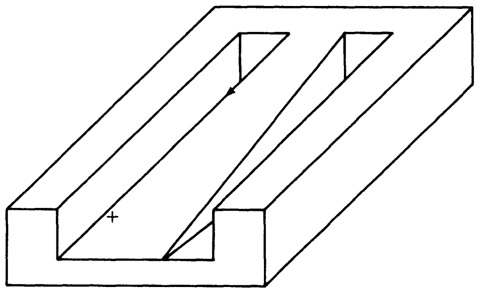


Figure 8.47 An impossible physical object. Note that one of the lines changes label from occluding to convex between two vertices.

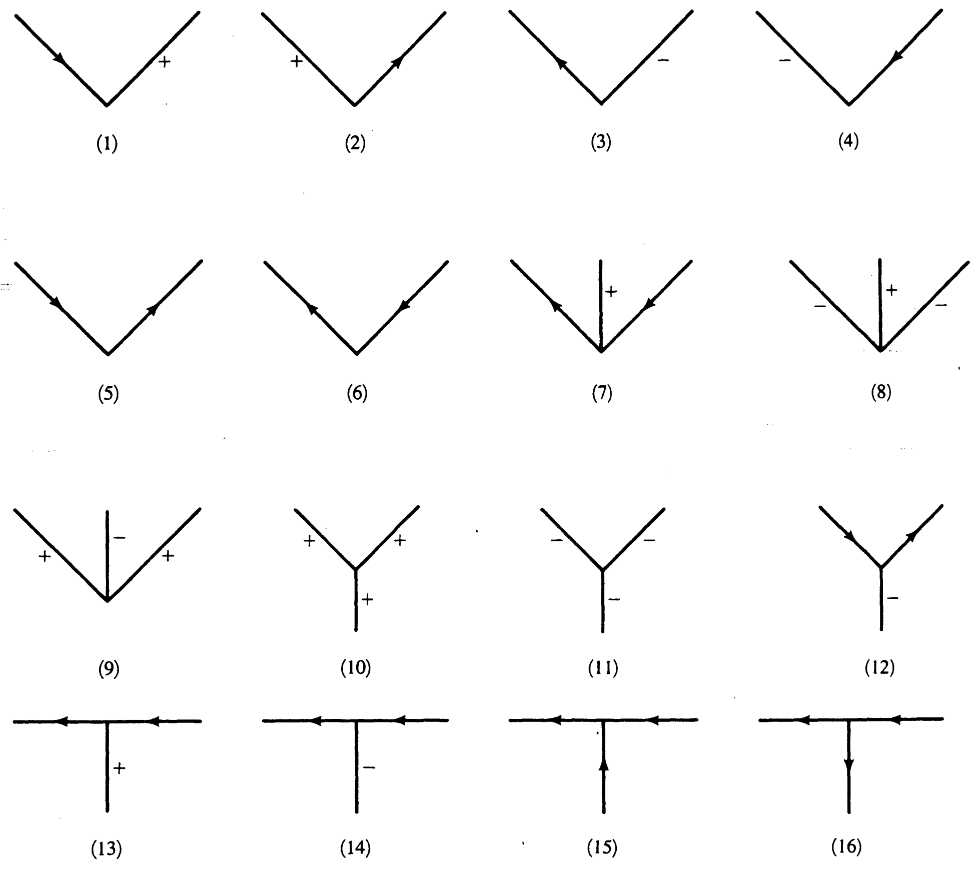


Figure 8.48 Junction dictionary for trihedral solids.

indicating where it touches the base. Thus, there is nothing in front of it and it can be extracted from the scene. We also note that there is a vertex of type (10) from the dictionary in Fig. 8.48. This is strong evidence (if we know we are dealing with trihedral objects) that the three surfaces involved in that vertex form a cube. Similar comments apply to the base after the cube surfaces are removed. Removing the base leaves the single object in the background, which completes the decomposition of the scene.

Although the preceding short explanation gives an overall view of how line and junction analysis are used to describe 3D objects in a scene, we point out that formulation of an algorithm capable of handling more complex scenes is far from a trivial task. Several comprehensive efforts in this area are referenced at the end of this chapter.

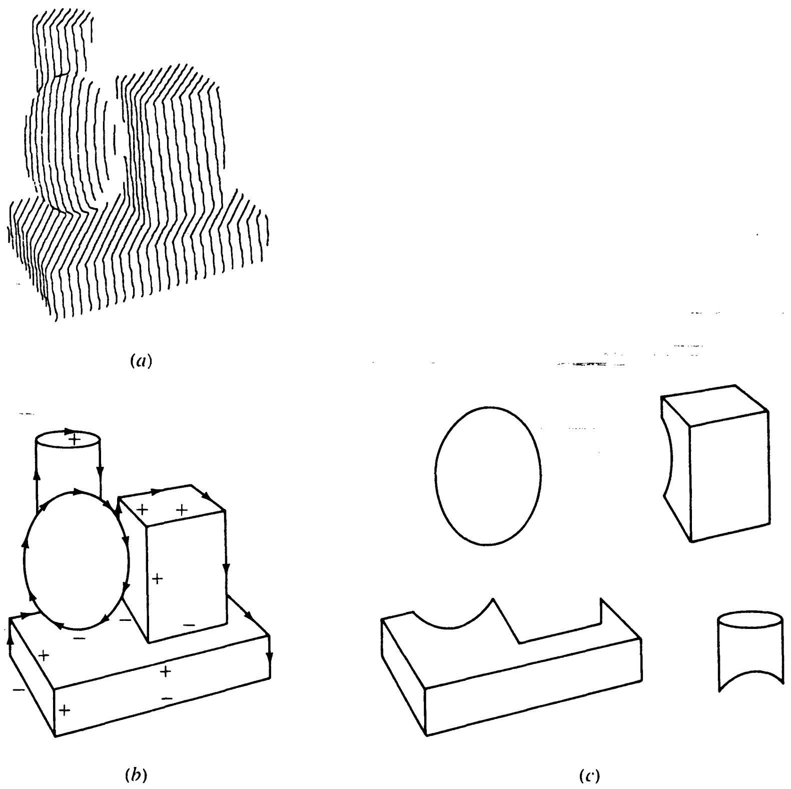


Figure 8.49 (*a*) Scene. (*b*) Labeled lines. (*c*) Decomposition via line and junction analysis. (Adapted from Shirai [1979], ©Plenum Press.)

8.4.4 Generalized Cones

A *generalized cone* (or *cylinder*) is the volume described by a planar cross section as it is translated along an arbitrary space curve (the *spine*), held at a constant angle to the curve, and transformed according to a *sweeping rule*. In machine vision, generalized cones provide viewpoint-independent representations of three-dimensional structures which are useful for description and model-based matching purposes.

Figure 8.50 illustrates the procedure for generating generalized cones. In Fig. 8.50*a* the cross section is a ring, the spine is a straight line and the sweeping rule is to translate the cross section normal to the spine while keeping its diameter constant. The result is a hollow cylinder. In Fig. 8.50*b* we have essentially the same situation, with the exception that the sweeping rule holds the diameter of the cross section constant and then allows it to increase linearly past the midpoint of the spine.

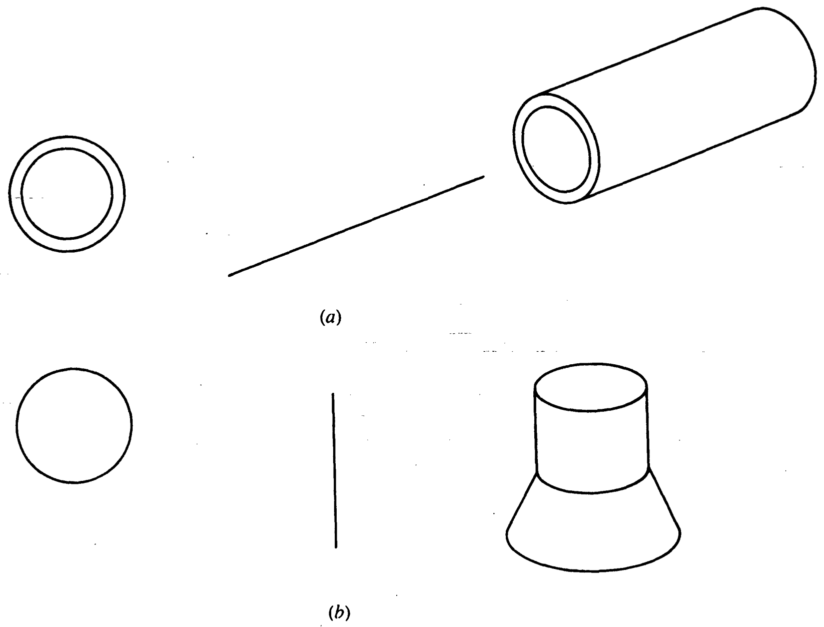


Figure 8.50 Cross sections, spines, and their corresponding generalized cones. In (*a*) the cross section remained constant during the sweep, while in (*b*) its diameter increased linearly past the midpoint in the spine.

When matching a set of 3D points against a set of known generalized cones, we first determine the center axis of the points and then find the closest set of cross sections that will fit the data as we travel along the spine. In general, considerable trial and error is required, particularly when one is dealing with incomplete data.

8.5 RECOGNITION

Recognition is a labeling process; that is, the function of recognition algorithms is to identify each segmented object in a scene and to assign a label (e.g., wrench, seal, bolt) to that object. For the most part, the recognition stages of present industrial vision systems operate on the assumption that objects in a scene have been segmented as individual units. Another common constraint is that images be acquired in a known viewing geometry (usually perpendicular to the work space). This decreases variability in shape characteristics and simplifies segmentation and description by reducing the possibility of occlusion. Variability in object orientation is handled by choosing rotation-invariant descriptors or by using the principal axis of an object to orient it in a predefined direction.

Recognition approaches in use today can be divided into two principal categories: *decision-theoretic* and *structural*. As will be seen in the following discussion, decision-theoretic methods are based on quantitative descriptions (e.g., statistical texture) while structural methods rely on symbolic descriptions and their relationships (e.g., sequences of directions in a chain-coded boundary). With a few exceptions, the procedures discussed in this section are generally used to recognize two-dimensional object representations.

8.5.1 Decision-Theoretic Methods

Decision-theoretic pattern recognition is based on the use of *decision* (*discriminant*) *functions*. Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ represent a column *pattern vector* with real components, where x_i is the ith descriptor of a given object (e.g., area, average intensity, perimeter length). Given M object classes, denoted by $\omega_1, \omega_2, \ldots, \omega_M$, the basic problem in decision-theoretic pattern recognition is to identify M decision functions, $d_1(\mathbf{x}), d_2(\mathbf{x}), \ldots, d_M(\mathbf{x})$, with the property that the following relationship holds for any pattern vector $\mathbf{x}^*$ belonging to class ω_i:

$$d_i(\mathbf{x}^*) > d_j(\mathbf{x}^*) \qquad j = 1, 2, \ldots, M;\ j \neq i \tag{8.5-1}$$

In other words, an unknown object represented by vector $\mathbf{x}^*$ is recognized as belonging to the ith object class if, upon substitution of $\mathbf{x}^*$ into all decision functions, $d_i(\mathbf{x}^*)$ yields the largest value.

The predominant use of decision functions in industrial vision systems is for *matching*. Suppose that we represent each object class by a *prototype* (or *average*) vector:

$$\mathbf{m}_i = \frac{1}{N}\sum_{k=1}^{N} \mathbf{x}_k \qquad i = 1, 2, \ldots, M \tag{8.5-2}$$

where the $\mathbf{x}_k$ are sample vectors known to belong to class ω_i. Given an unknown $\mathbf{x}^*$, one way to determine its class membership is to assign it to the class of its closest prototype. If we use the euclidean distance to determine closeness, the problem reduces to computing the following distance measures:

$$D_j(\mathbf{x}^*) = \|\mathbf{x}^* - \mathbf{m}_j\| \qquad j = 1, 2, \ldots, M \tag{8.5-3}$$

where $\|\mathbf{a}\| = (\mathbf{a}^T\mathbf{a})^{1/2}$ is the euclidean norm. We then assign $\mathbf{x}^*$ to class ω_i if $D_i(\mathbf{x}^*)$ is the smallest distance. It is not difficult to show that this is equivalent to evaluating the functions

$$d_j(\mathbf{x}^*) = (\mathbf{x}^*)^T\mathbf{m}_j - ½\mathbf{m}_j^T\mathbf{m}_j \qquad j = 1, 2, \ldots, M \tag{8.5-4}$$

and selecting the largest value. This formulation agrees with the concept of a decision function, as defined in Eq. (8.5-1).

Another application of matching is in searching for an instance of a subimage $w(x, y)$ in a larger image $f(x, y)$. At each location (x, y) of $f(x, y)$ we define the *correlation coefficient* as

$$\gamma(x, y) = \frac{\sum_s \sum_t [w(s, t) - m_w][f(s, t) - m_f]}{\left\{ \sum_s \sum_t [w(s, t) - m_w]^2 \sum_s \sum_t [f(s, t) - m_f]^2 \right\}^{1/2}} \tag{8.5-5}$$

where it is assumed that $w(s, t)$ is centered at coordinates (x, y). The summations are taken over the image coordinates common to both regions, m_w is the average intensity of w, and m_f is the average intensity of f in the region coincident with w. It is noted that, in general, $\gamma(x, y)$ will vary from one location to the next and that its values are in the range $[-1, 1]$, with a value of 1 corresponding to a perfect match. The procedure, then, is to compute $\gamma(x, y)$ at each location

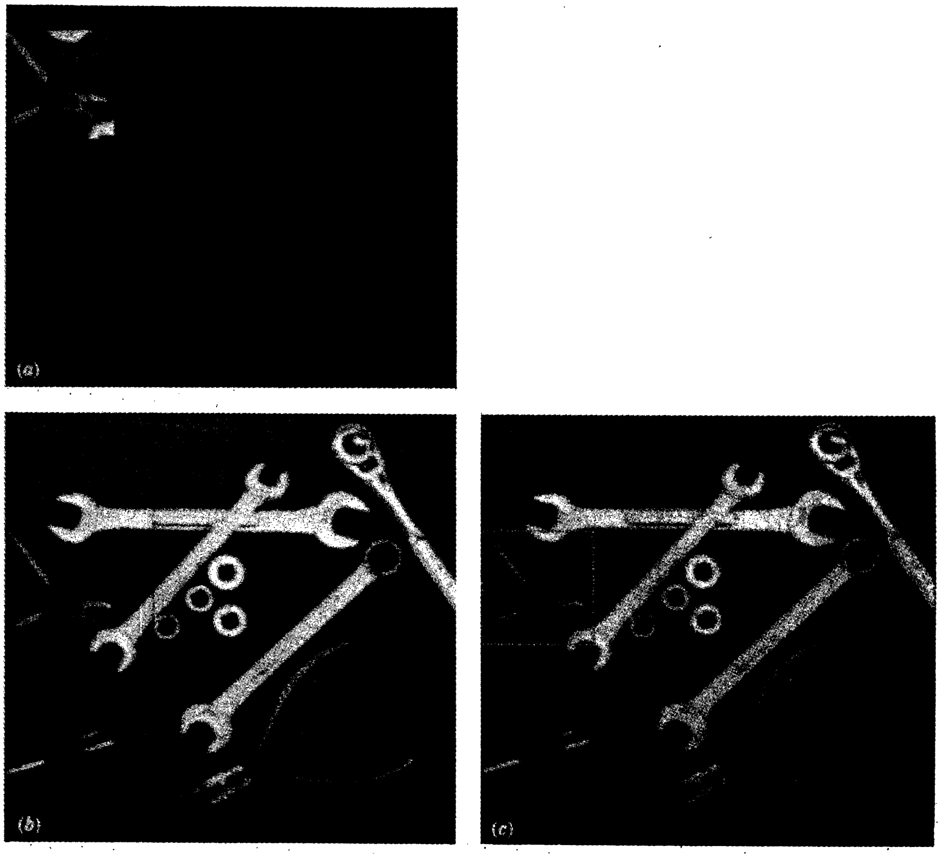


Figure 8.51 (*a*) Subimage $w(x, y)$. (*b*) Image $f(x, y)$. (*c*) Location of the best match of w in f, as determined by the largest correlation coefficient.

(x, y) and to select its largest value to determine the best match of w in f [the procedure of moving $w(x, y)$ throughout $f(x, y)$ is analogous to Fig. 7.20].

The quality of the match can be controlled by accepting a correlation coefficient only if it exceeds a preset value (for example, .9). Since this method consists of directly comparing two regions, it is clearly sensitive to variations in object size and orientation. Variations in intensity are normalized by the denominator in Eq. (8.5-5). An example of matching by correlation is shown in Fig. 8.51.

8.5.2 Structural Methods

The techniques discussed in Sec. 8.5.1 deal with patterns on a quantitative basis, ignoring any geometrical relationships which may be inherent in the shape of an object. Structural methods, on the other hand, attempt to achieve object discrimination by capitalizing on these relationships.

Central to the structural recognition approach is the decomposition of an object into *pattern primitives*. This idea is easily explained with the aid of Fig. 8.52. Part (*a*) of this figure shows a simple object boundary, and Fig. 8.52*b* shows a set of primitive elements of specified length and direction. By starting at the top left, tracking the boundary in a clockwise direction, and identifying instances of these primitives, we obtain the coded boundary shown in Fig. 8.52*c*. Basically, what we have done is represent the boundary by the string *aaabcbbbcdddcd*. The known length and direction of these primitives, together with the order in which they occur, establishes the structure of the object in terms of this particular representation. The objective of this section is to introduce the reader to techniques suitable for handling this and other types of structural pattern descriptions.

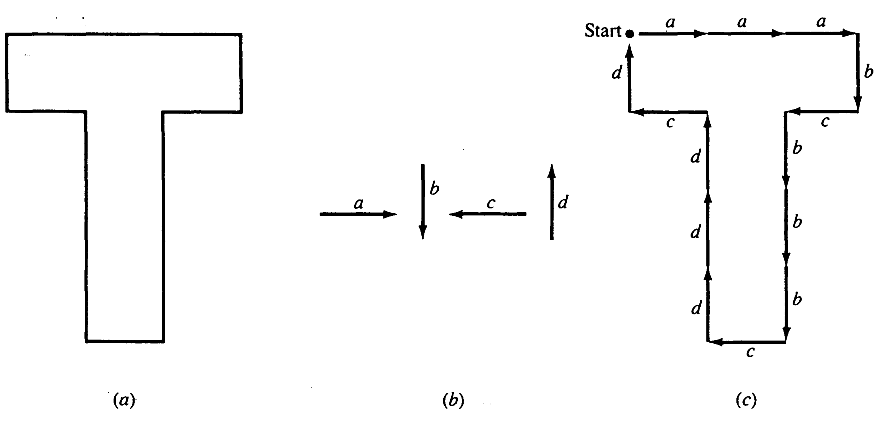


Figure 8.52 (*a*) Object boundary. (*b*) Primitives. (*c*) Boundary coded in terms of primitives, resulting in the string *aaabcbbbcdddcd*.

Matching Shape Numbers. A procedure analogous to the minimum-distance concept introduced in Sec. 8.5.1 for vector representations can be formulated for the comparison of two object boundaries that are described in terms of shape numbers. With reference to the discussion in Sec. 8.3.1, the *degree of similarity* k between two object boundaries, A and B, is defined as the largest order of which their shape numbers still coincide. That is, we have $s_4(A) = s_4(B)$, $s_6(A) = s_6(B)$, $s_8(A) = s_8(B), \ldots,$ $s_k(A) = s_k(B)$, $s_{k+2}(A) \neq s_{k+2}(B)$, $s_{k+4}(A) \neq s_{k+4}(B), \ldots,$ where s indicates shape number and the subscript indicates the order. The *distance* between two shapes A and B is defined as the inverse of their degree of similarity:

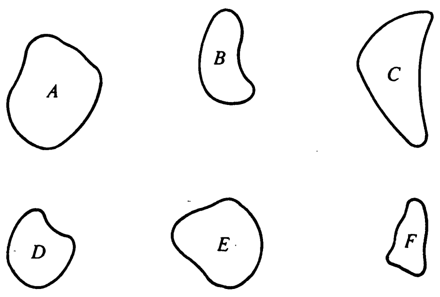
$$D(A, B) = \frac{1}{k} \tag{8.5-6}$$

This distance satisfies the following properties:

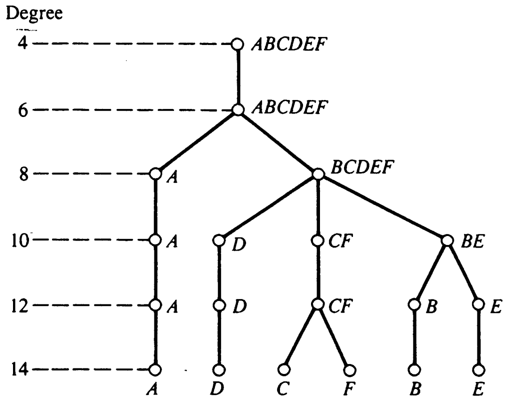
$$\begin{aligned} &(a)\ D(A, B) \geqslant 0 \\ &(b)\ D(A, B) = 0 \qquad \text{iff } A = B \\ &(c)\ D(A, C) \leqslant \max\,[D(A, B), D(B, C)] \end{aligned} \tag{8.5-7}$$

In order to compare two shapes, we can use either k or D. If the degree of similarity is used, then we know from the above discussion that the larger k is, the more similar the shapes are (note that k is infinite for identical shapes). The reverse is true when the distance measure is used.

Example: As an illustration of the preceding concepts, suppose that we wish to find which of the five shapes (A, B, D, E, F) in Fig. 8.53*a* best matches shape C. This is analogous to having five prototype shapes whose identities are known and trying to determine which of these constitutes the best match to an unknown shape. The search may be visualized with the aid of the *similarity tree* shown in Fig. 8.53*b*. The root of the tree corresponds to the lowest degree of similarity considered, which in this example is 4. As shown in the tree, all shapes are identical up to degree 6, with the exception of shape A. That is, the degree of similarity of this shape with respect to all the others is 6. Proceeding down the tree we find that shape D has degree 8 with respect to the remaining shapes, and so on. In this particular case, shape F turned out to be a unique match for C and, furthermore, their degree of similarity is higher than any of the other shapes. If E had been the unknown, a unique match would have also been found, but with a lower degree of similarity. If A had been the unknown, all we could have said using this method is that it is similar to the other five figures with degree 6. The same information can be summarized in the form of a *similarity matrix*, as shown in Fig. 8.53*c*. □



(*a*)



(*b*)

	A	B	C	D	E	F
A	∞	6	6	6	6	6
B		∞	8	8	10	8
C			∞	8	8	12
D				∞	8	8
E					∞	8
F						∞

(*c*)

Figure 8.53 (*a*) Shapes. (*b*) Similarity tree. (*c*) Similarity matrix. (From Bribiesca and Guzman [1980], ©Pergamon Press.)

String Matching. Suppose that two object contours C_1 and C_2 are coded into strings $a_1 a_2 \cdots a_n$ and $b_1 b_2 \cdots b_m$, respectively. Let A represent the number of matches between the two strings, where we say that a match has occurred in the jth position if $a_j = b_j$. The number of symbols that do not match up is given by

$$B = \max(|C_1|, |C_2|) - A \tag{8.5-8}$$

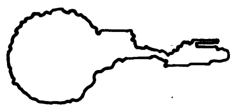
where $|C|$ is the length (number of symbols) of string C. It can be shown that $B = 0$ if and only if C_1 and C_2 are identical.

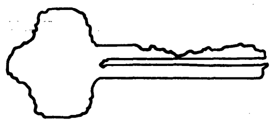
A simple measure of similarity between strings C_1 and C_2 is defined as the ratio

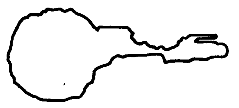
$$R = \frac{A}{B} = \frac{A}{\max(|C_1|, |C_2|) - A} \tag{8.5-9}$$

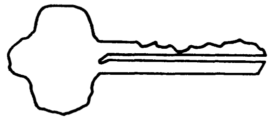
Based on the above comment regarding B, R is infinite for a perfect match and zero when none of the symbols in C_1 and C_2 match (i.e., $A = 0$ in this case). Since the matching is done on a symbol-by-symbol basis, the starting point on each boundary when creating the string representation is important. Alternatively, we can start at arbitrary points on each boundary, shift one string (with wraparound), and compute Eq. (8.5-9) for each shift. The number of shifts required to perform all necessary comparisons is max $(|C_1|, |C_2|)$.

Example: Figure 8.54*a* and *b* shows a sample boundary from each of two classes of objects. The boundaries were approximated by a polygonal fit (Fig. 8.54*c* and *d*) and then strings were formed by computing the interior angle


(*a*)


(*b*)


(*c*)


(*d*)

A/B	1.a	1.b	1.c	1.d	1.e
1.b	16.0				
1.c	9.6	26.3			
1.d	5.07	8.1	10.3		
1.e	4.67	7.2	10.3	14.2	
1.f	4.67	7.2	10.3	8.5	23.7

(*e*)

A/B	2.a	2.b	2.c	2.d	2.e
2.b	33.5				
2.c	4.75	5.8			
2.d	3.6	4.23	19.3		
2.e	2.83	3.25	9.17	18.3	
2.f	2.63	3.0	7.71	13.5	27.0

(*f*)

A/B	1.a	1.b	1.c	1.d	1.e	1.f
2.a	1.24	1.5	1.32	1.47	1.55	1.48
2.b	1.18	1.43	1.32	1.47	1.55	1.48
2.c	1.02	1.18	1.19	1.32	1.39	1.48
2.d	1.02	1.18	1.19	1.32	1.39	1.40
2.e	0.93	1.07	1.08	1.19	1.24	1.25
2.f	0.89	1.02	1.02	1.14	1.11	1.18

(*g*)

Figure 8.54 (*a*), (*b*) Sample boundaries of two different object classes. (*c*), (*d*) Their corresponding polygonal approximations. (*e*)–(*g*) Tabulations of $R = A/B$. (Adapted from Sze and Yang [1981], ©IEEE.)

between the polygon segments as the polygon was traversed in a clockwise direction. Angles were coded into one of eight possible symbols which correspond to 45° increments, $s_1: 0° < \theta \leqslant 45°$, $s_2: 45° < \theta \leqslant 90°$, ..., $s_8: 315° < \theta \leqslant 360°$.

The results of computing the measure R for five samples of object 1 against themselves are shown in Fig. 8.54*e*, where the entries correspond to values of $R = A/B$ and, for example, the notation 1.c refers to the third string for object class 1. Figure 8.54*f* shows the results for the strings of the second object class. Finally, Fig. 8.54*g* is a tabulation of R values obtained by comparing strings of one class against the other. The important thing to note is that all values of R in this last table are considerably smaller than any entry in the preceding two tables, indicating that the R measure achieved a high degree of discrimination between the two classes of objects. For instance, if string 1.a had been an unknown, the smallest value in comparing it with the other strings of class 1 would have been 4.67. By contrast, the largest value in a comparison against class 2 would have been 1.24. Thus, classification of this string into class 1 based on the maximum value of R would have been a simple, unambiguous matter. □

Syntactic Methods. Syntactic techniques are by far the most prevalent concepts used for handling structural recognition problems. Basically, the idea behind syntactic pattern recognition is the specification of structural pattern primitives and a set of rules (in the form of a *grammar*) which govern their interconnection. We consider first string grammars and then extend these ideas to higher-dimensional grammars.

String Grammars. Suppose that we have two classes of objects, ω_1 and ω_2, which are represented as strings of primitives, as outlined at the beginning of Sec. 8.5.2. We may interpret each primitive as being a symbol permissible in some grammar, where a *grammar* is a set of rules of syntax (hence the name syntactic pattern recognition) for the generation of *sentences* formed from the given symbols. In the context of the present discussion, these sentences are strings of symbols which in turn represent patterns. It is further possible to envision two grammars, G_1 and G_2, whose rules are such that G_1 only allows the generation of sentences which correspond to objects of class ω_1 while G_2 only allows generation of sentences corresponding to objects of class ω_2. The set of sentences generated by a grammar G is called its *language*, and denoted by $L(G)$.

Once the two grammars G_1 and G_2 have been established, the syntactic pattern recognition process is, in principle, straightforward. Given a sentence representing an unknown pattern, the problem is one of deciding in which language the pattern represents a valid sentence. If the sentence belongs to $L(G_1)$, we say that the pattern belongs to object class ω_1. Similarly, we say that the object comes from class ω_2 if the sentence is in $L(G_2)$. A unique decision cannot be made if the sentence belongs to both $L(G_1)$ and $L(G_2)$. If the sentence is found to be invalid over both languages it is rejected.

When there are more than two pattern classes, the syntactic classification approach is the same as described above, except that more grammars (at least one per class) are involved in the process. In this case the pattern is assigned to class ω_i if it is a sentence of *only* $L(G_i)$. A unique decision cannot be made if the sentence belongs to more than one language, and (as above) a pattern is rejected if it does not belong to any of the languages under consideration.

When dealing with strings, we define a grammar as the four-tuple

$$G = (N, \Sigma, P, S) \tag{8.5-10}$$

where

N = finite set of *nonterminals* or variables
Σ = finite set of *terminals* or constants
P = finite set of *productions* or rewriting rules
S in N = the *starting symbol*

It is required that N and Σ be disjoint sets. In the following discussion nonterminals will be denoted by capital letters: $A, B, \ldots, S, \ldots$. Lowercase letters at the beginning of the alphabet will be used for terminals: $a, b, c, \ldots$. Strings of terminals will be denoted by lowercase letters toward the end of the alphabet: v, w, x, y, z. Strings of mixed terminals and nonterminals will be denoted by lowercase Greek letters: $\alpha, \beta, \theta, \ldots$. The *empty sentence* (the sentence with no symbols) will be denoted by λ. Finally, given a set V of symbols, we will use the notation V^* to denote the set of all sentences composed of elements from V.

String grammars are characterized primarily by the form of their productions. Of particular interest in syntactic pattern recognition are *regular grammars*, whose productions are always of the form $A \rightarrow aB$ or $A \rightarrow a$ with A and B in N, and a in Σ, and *context-free* grammars, with productions of the form $A \rightarrow \alpha$ with A in N, and α in the set $(N \cup \Sigma)^* - \lambda$; that is, α can be any string composed of terminals and nonterminals, except the empty string.

Example: The preceding concepts are best clarified by an example. Suppose that the object shown in Fig. 8.55*a* is represented by its skeleton, and that we define the primitives shown in Fig. 8.55*b* to describe the structure of this and similar skeletons. Consider the grammar $G = (N, \Sigma, P, S)$ with $N = \{A, B, S\}$, $\Sigma = \{a, b, c\}$, and production rules

1. $S \rightarrow aA$
2. $A \rightarrow bA$
3. $A \rightarrow bB$
4. $B \rightarrow c$

where the terminals a, b, and c are as shown in Fig. 8.55*b*. As indicated earlier, S is the starting symbol from which we generate all strings in $L(G)$.

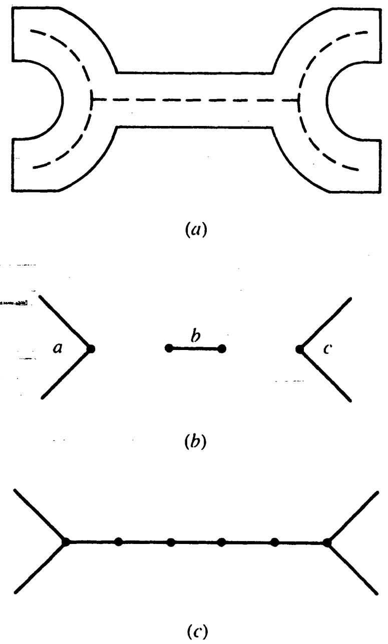


Figure 8.55 (*a*) Object represented by its skeleton. (*b*) Primitives. (*c*) Structure generated using a regular string grammar.

If, for instance, we apply production 1 followed by two applications of production 2, we obtain: $S \Rightarrow aA \Rightarrow abA \Rightarrow abbA$, where " $\Rightarrow$ " indicates a string derivation starting from S and using production rules from P. It is noted that we interpret the production $S \rightarrow aA$ and $A \rightarrow bA$ as "S can be rewritten as aA" and "A can be rewritten as bA." Since we have a nonterminal in the string $abbA$ and a rule which allows us to rewrite it, we can continue the derivation. For example, if we apply production 2 two more times, followed by production 3 and then production 4, we obtain the string *abbbbbc* which corresponds to the structure shown in Fig. 8.55*c*. It is important to note that no nonterminals are left after application of production 4 so the derivation terminates after this production is used. A little thought will reveal that the grammar given above has the language $L(G) = \{ab^nc \mid n \geqslant 1\}$, where b^n indicates n repetitions of the symbol b. In other words, G is capable of generating the skeletons of wrenchlike structures with bodies of arbitrary length within the resolution established by the length of primitive b. □

Use of Semantics. In the above example we have implicitly assumed that the interconnection between primitives takes place only at the dots shown in Fig. 8.55*b*. In more complicated situations the rules of connectivity, as well as other information regarding factors such as primitive length and direction, and the number of times a production can be applied, must be made explicit. This is usually accomplished via the use of *semantics*. Basically, syntax establishes the structure of an object or expression, while semantics deal with its meaning. For example, the FORTRAN statement $A = B/C$ is syntactically correct, but it is semantically correct only if $C \neq 0$.

In order to fix these ideas, suppose that we attach semantic information to the wrench grammar just discussed. This information may be attached to the productions as follows:

Production	Semantic information
$S \rightarrow aA$	Connections to a are made only at the dot. The direction of a, denoted by θ, is given by the direction of the perpendicular bisector of the line joining the endpoints of the two undotted segments. These line segments are 3 cm each.
$A \rightarrow bA$	Connections to b are made only at the dots. No multiple connections are allowed. The direction of b must be the same as that of a and the length of b is 0.25 cm. This production cannot be applied more than 10 times.
$A \rightarrow bB$	The direction of a and b must be the same. Connections must be simple and made only at the dots.
$B \rightarrow c$	The direction of c and a must be the same. Connections must be simple and made only at the dots.

It is noted that, by using semantic information, we are able to use a few rules of syntax to describe a broad (although limited as desired) class of patterns. For instance, by specifying the direction θ, we avoid having to specify primitives for each possible object orientation. Similarly, by requiring that all primitives be oriented in the same direction, we eliminate from consideration nonsensical wrenchlike structures.

Recognition. Thus far, we have seen that grammars are *generators* of patterns. In the following discussion we consider the problem of recognizing if a given pattern string belongs to the language $L(G)$ generated by a grammar G. The basic concepts underlying syntactic recognition can be illustrated by the development of mathematical models of computing machines, called *automata*. Given an input pattern string, these automata have the capability of recognizing whether or not the pattern belongs to a specified language or class. We will focus attention only on *finite automata*, which are the recognizers of languages generated by regular grammars.

A *finite automaton* is defined as a five-tuple

$$A = (Q, \Sigma, \delta, q_0, F) \tag{8.5-11}$$

where Q is a finite, nonempty set of *states*, Σ is a finite input *alphabet*, δ is a mapping from $Q \times \Sigma$ (the set of ordered pairs formed from elements of Q and Σ) into the collection of all subsets of Q, q_0 is the *starting state*, and F (a subset of Q) is a set of *final* or *accepting states*. The terminology and notation associated with Eq. (8.5-11) are best illustrated by a simple example.

Example: Consider an automaton given by Eq. (8.5-11) where $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $F = \{q_0\}$, and the mappings are given by $\delta(q_0, a) = \{q_2\}$, $\delta(q_0, b) = \{q_1\}$, $\delta(q_1, a) = \{q_2\}$, $\delta(q_1, b) = \{q_0\}$, $\delta(q_2, a) = \{q_0\}$, $\delta(q_2, b) = \{q_1\}$. If, for example, the automaton is in state q_0 and an a is input, its state changes to q_2. Similarly, if a b is input next, the automaton moves to state q_1, and so forth. It is noted that, in this case, the initial and final states are the same. □

A *state diagram* for the automaton just discussed is shown in Fig. 8.56. The state diagram consists of a node for each state, and directed arcs showing the possible transitions between states. The final state is shown as a double circle and each arc is labeled with the symbol that causes that transition. A string w of terminal symbols is said to be *accepted* or *recognized* by the automaton if, starting in state q_0, the sequence of symbols in w causes the automaton to be in a final state after the last symbol in w has been input. For example, the automaton in Fig. 8.56 recognizes the string w = *abbabb*, but rejects the string w = *aabab*.

There is a one-to-one correspondence between regular grammars and finite automata. That is, a language is recognized by a finite automaton if and only if it is generated by a regular grammar. The procedure for obtaining the automaton corresponding to a given regular grammar is straightforward. Let the grammar be denoted by $G = (N, \Sigma, P, X_0)$, where $X_0 \equiv S$, and suppose that N is composed of X_0 plus n additional nonterminals $X_1, X_2, \ldots, X_n$. The state set Q for the automaton is formed by introducing $n + 2$ states, $\{q_0, q_1, \ldots, q_n, q_{n+1}\}$ such that q_i corresponds to X_i for $0 \leqslant i \leqslant n$ and q_{n+1} is the final state. The set of

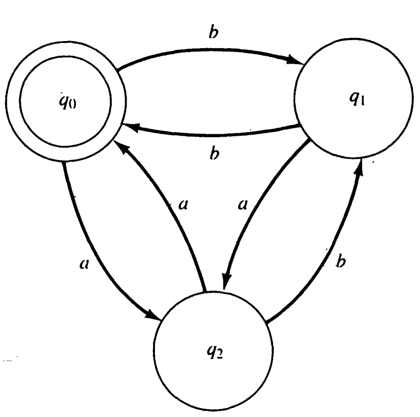


Figure 8.56 A finite automaton.

input symbols is identical to the set of terminals in G. The mappings in δ are defined by two rules based on the production of G; namely, for each i and j, $0 \leqslant i \leqslant n$, $0 \leqslant j \leqslant n$:

1. If $X_i \rightarrow aX_j$ is in P, then $\delta\ (q_i, a)$ contains q_j.
2. If $S_i \rightarrow a$ is in P, then $\delta\ (q_i, a)$ contains q_{n+1}.

On the other hand, given a finite automaton $A = (Q, \Sigma, \delta, q_0, F)$, we obtain the corresponding regular grammar $G = (N, \Sigma, P, X_0)$ by letting N be identified with the state set Q, with the starting symbol X_0 corresponding to q_0, and the productions of G obtained as follows:

1. If q_j is in $\delta(q_i, a)$, there is a production $X_i \rightarrow aX_j$ in P.
2. If a state in F is in $\delta(q_i, a)$, there is a production $X_i \rightarrow a$ in P.

Example: The finite automaton corresponding to the wrench grammar given earlier is obtained by writing the productions as $X_0 \rightarrow aX_1$, $X_1 \rightarrow bX_1$, $X_1 \rightarrow bX_2$, $X_2 \rightarrow c$. Then, from the above discussion, we have $A = (Q, \Sigma, \delta, q_0, F)$ with $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b, c\}$, $F = \{q_3\}$, and mappings $\delta(q_0, a) = \{q_1\}$, $\delta(q_1, b) = \{q_1, q_2\}$, $\delta(q_2, c) = \{q_3\}$. For completeness, we can write $\delta(q_0, b) = \delta(q_0, c) = \delta(q_1, a) = \delta(q_1, c) = \delta(q_2, a) = \delta(q_2, b) = \phi$, where ϕ is the null set, indicating that these transitions are not defined for this automaton. □

Higher-Dimensional Grammars. The grammars discussed above are best suited for applications where the connectivity of primitives can be conveniently expressed in a stringlike manner. In the following discussion we consider two examples of grammars capable of handling more general interconnections between primitives and subpatterns.

A *tree grammar* is defined as the five-tuple

$$G = (N, \Sigma, P, r, S) \tag{8.5-12}$$

where N and Σ are, as before, sets of nonterminals and terminals, respectively; S is the start symbol which can, in general, be a tree; P is a set of productions of the form $T_i \rightarrow T_j$, where T_i and T_j are trees; and r is a *ranking function* which denotes the number of direct descendants of a node whose label is a terminal in the grammar. An *expansive* tree grammar has productions of the form

$$\begin{tikzcd} A \arrow[rr] & & a \arrow[dll, dash] \arrow[d, dash] \\ A_1 & \cdots & A_n \end{tikzcd}$$

where $A, A_1, \ldots, A_n$ are nonterminals, and a is a terminal.

Example: The skeleton of the structure shown in Fig. 8.57*a* can be generated by means of a tree grammar with productions

(1) $S \rightarrow \begin{array}{c} a \\ | \\ A_1 \end{array}$ (2) $A_1 \rightarrow \begin{array}{c} b \\ | \\ A_1 \end{array}$ (3) $A_1 \rightarrow \begin{array}{c} c \\ \wedge \\ A_2 \quad A_3 \end{array}$

(4) $A_2 \rightarrow \begin{array}{c} d \\ | \\ A_2 \end{array}$ (5) $A_2 \rightarrow e$ (6) $A_3 \rightarrow \begin{array}{c} e \\ | \\ A_3 \end{array}$ (7) $A_3 \rightarrow a$

where connectivity between linear primitives is head to tail, and connections to the circle primitive can be made anywhere on its circumference. The ranking functions in this case are $r(a) = \{0, 1\}$, $r(b) = r(d) = r(e) = \{1\}$, $r(c) = \{2\}$. It is noted that restricting the use of productions 2, 4, and 6 to

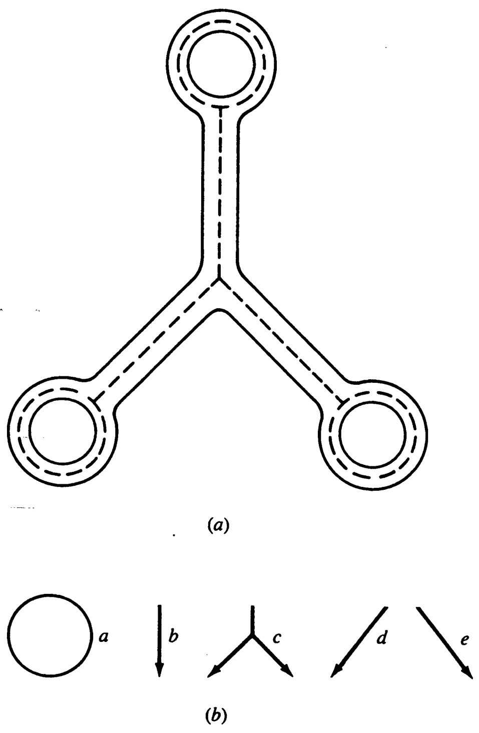


Figure 8.57 (*a*) An object and (*b*) primitives used for representing the skeleton by means of a tree grammar.

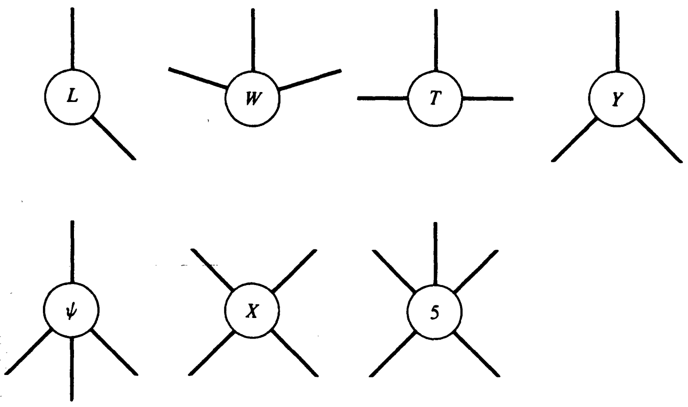


Figure 8.58 Vertex primitives. (From Gips [1974], ©Pergamon Press.)

be applied the same number of times generates a structure in which all three legs are of the same length. Similarly, requiring that productions 4 and 6 be applied the same number of times produces a structure that is symmetrical about the vertical axis in Fig. 8.57*a*. □

We conclude this section with a brief discussion of a grammar proposed by Gips [1974] for generating three-dimensional objects consisting of cube structures. As in the previous discussion, the key to object generation by syntactic techniques is the specification of a set of primitives and their interconnections. In this case, the primitives are the vertices shown in Fig. 8.58. Vertices of type T are further classified as either T_1 or T_3, using local information. If a T vertex is contained in a parallelogram of vertices it is classified as type T_3. If a T vertex is not contained in a parallelogram of vertices, it is classified as type T_1. Figure 8.59 shows this classification.

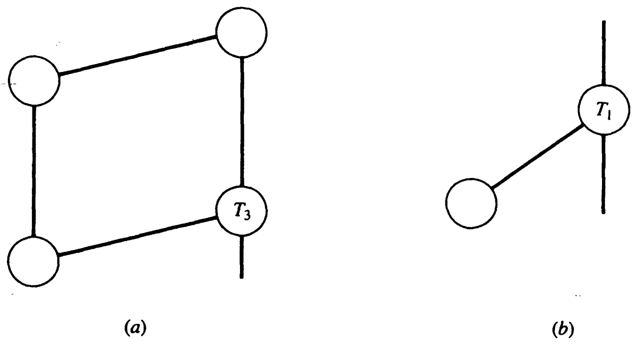


Figure 8.59 Further classification of T vertices.

The rules of the grammar consist of specifying valid interconnections between structures, as detailed in Fig. 8.60. The vertices denoted by double circles denote central vertices of the end cube of an object where further connections can be made. This is illustrated in Fig. 8.61 which shows a typical derivation using the rules of Fig. 8.60. Figure 8.62 illustrates the range of structures that can be generated with these rules.

8.6 INTERPRETATION

In this discussion, we view interpretation as the process which endows a vision system with a higher level of cognition about its environment than that offered by any of the concepts discussed thus far. When viewed in this way, interpretation clearly encompasses all these methods as an integral part of understanding a visual scene. Although this is one of the most active research topics in machine vision, the reader is reminded of the comments made in Secs. 7.1 and 8.1 regarding the fact that our understanding of this area is really in its infancy. In this section we touch briefly upon a number of topics which are representative of current efforts toward advancing the state of the art in machine vision.

The power of a machine vision system is determined by its ability to extract meaningful information from a scene under a broad range of viewing conditions and using minimal knowledge about the objects being viewed. There are a number of factors which make this type of processing a difficult task, including variations in illumination, occluding bodies, and viewing geometry. In Sec. 7.3 we spent considerable time discussing techniques designed to reduce variability in illumination and thus provide a relatively constant input to a vision system. The back- and structured-lighting approaches discussed in that section are indicative of the extreme levels of specialization employed by current industrial systems to reduce the difficulties associated with arbitrary lighting of the work space. Among these difficulties we find shadowing effects which complicate edge finding, and the introduction of nonuniformities on smooth surfaces which often results in their being detected as distinct bodies. Clearly, many of these problems result from the fact that relatively little is known about modeling the illumination-reflectance properties of 3D scenes. The line and junction labeling techniques discussed in Sec. 8.4 represent an attempt in this direction, but they fall short of explaining the interaction of illumination and reflectivity in quantitative terms. A more promising approach is based on mathematical models which attempt to infer intrinsic relationships between illumination, reflectance, and surface characteristics such as orientation (Horn [1977]; Marr [1979]; Katsushi and Horn [1981]).

Occlusion problems come into play when we are dealing with a multiplicity of objects in an unconstrained working environment. Consider, for example, the scene shown in Fig. 8.63. A human observer would have little difficulty, say, in determining the presence of two wrenches behind the sockets. For a machine, however, interpretation of this scene is a totally different story. Even if the system were able to perform a perfect segmentation of object clusters from the back-

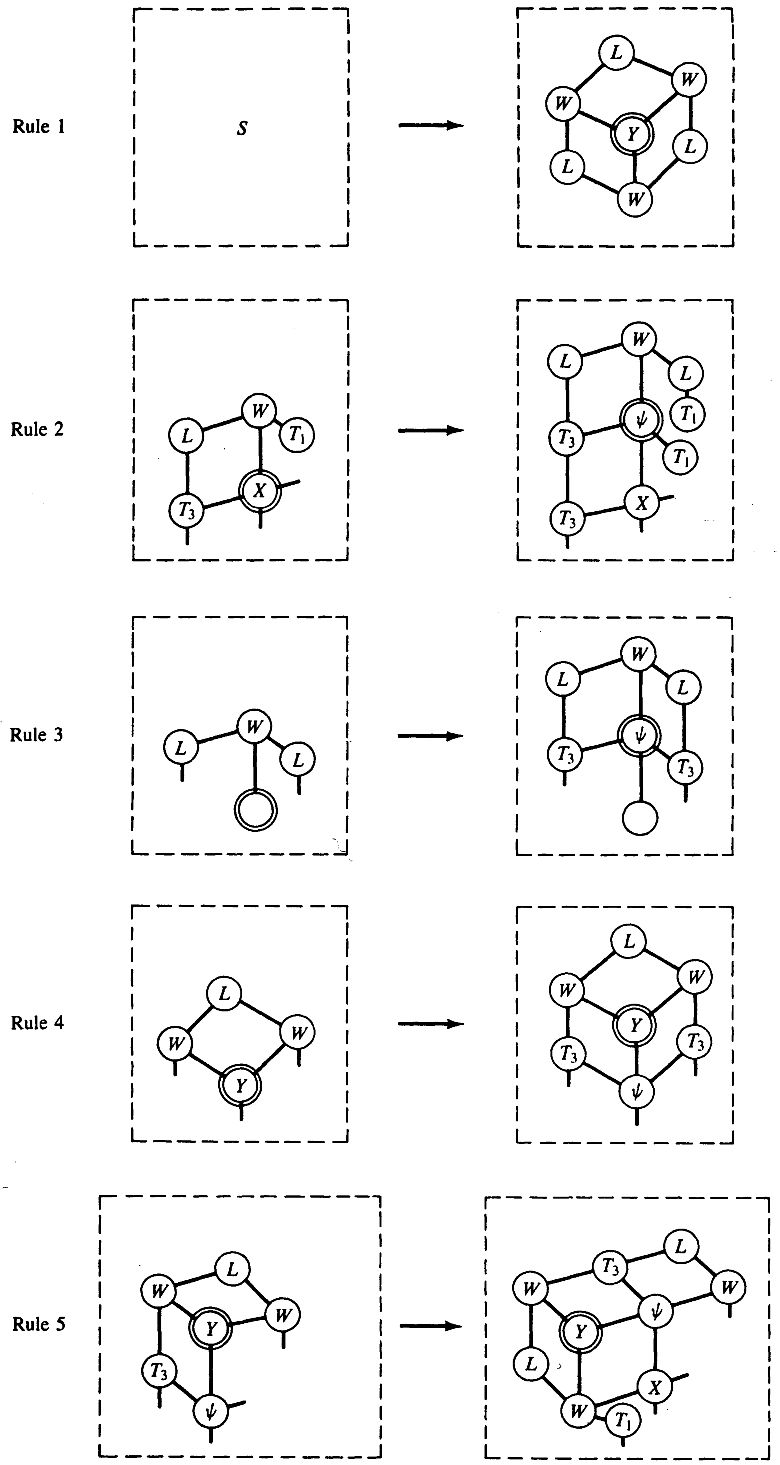


Figure 8.60 Rules used to generate three-dimensional structures. The blank circles indicate that more than one vertex type is allowed. (Adapted from Gips [1974], ©Pergamon Press.)

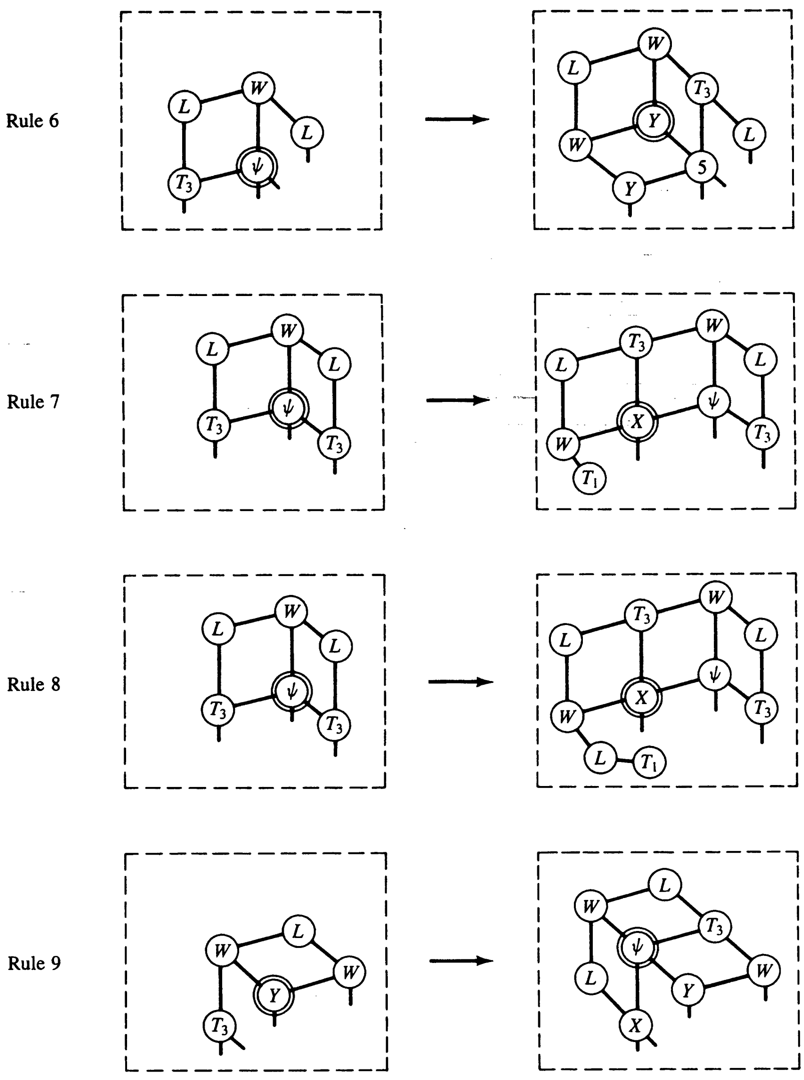


Figure 8.60 (continued)

ground, all the two-dimensional procedures discussed thus far for description and recognition would perform poorly on most of the occluded objects. The three-dimensional descriptors discussed in Sec. 8.4 would have a better chance, but even they would yield incomplete information. For instance, several of the sockets would appear as partial cylindrical surfaces, and the middle wrench would appear as two separate objects.

Processing scenes such as the one shown in Fig. 8.63 requires the capability to obtain descriptions which inherently carry shape and volumetric information, and procedures for establishing relationships between these descriptions, even when

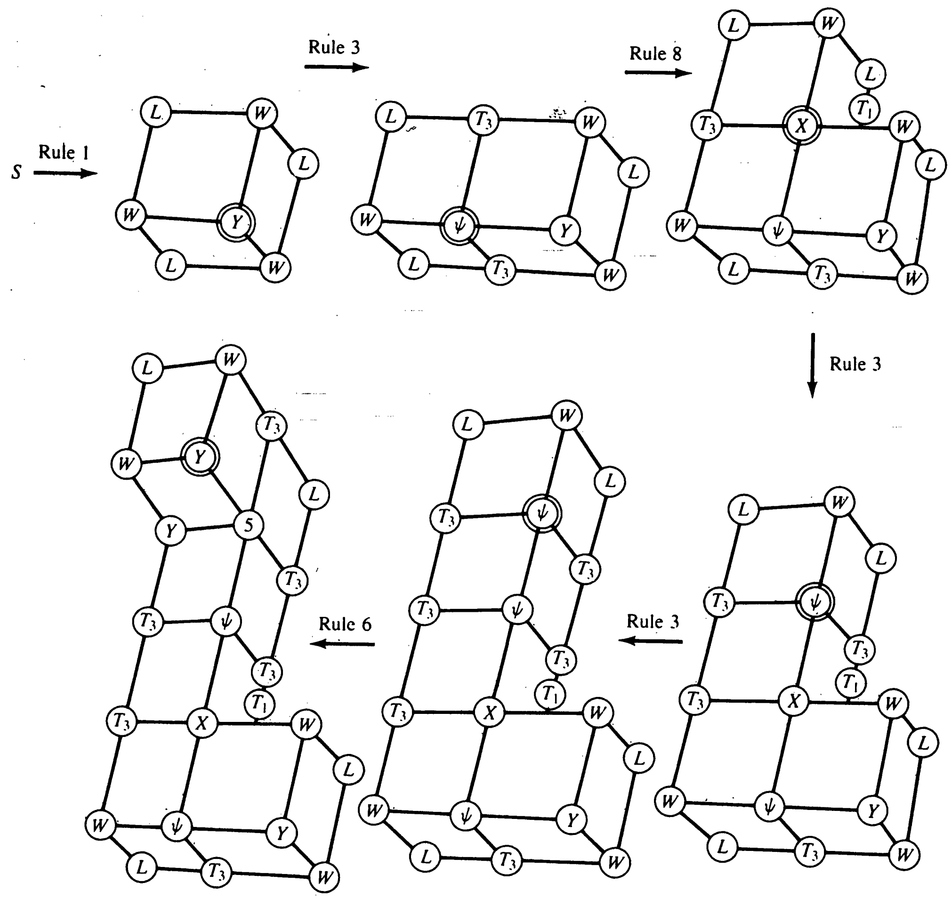


Figure 8.61 Sample derivation using the rules in Fig. 8.60. (Adapted from Gips [1974], © Pergamon Press.)

they are incomplete. Ultimately, these issues will be resolved only through the development of methods capable of handling 3D information obtained either by means of direct measurements or via geometric reasoning techniques capable of inferring (but not necessarily quantifying) 3D relationships from intensity imagery.

As an example of this type of reasoning, the reader would have little difficulty in arriving at a detailed interpretation of the objects in Fig. 8.63 with the exception of the object occluded by the screwdriver. The capability to know when interpretation of a scene or part of a scene is not an achievable task is just as important as correctly analyzing the scene. The decision to look at the scene from a different viewpoint (Fig. 8.64) to resolve the issue would be a natural reaction in an intelligent observer.

One of the most promising approaches in this direction is research in model-driven vision (Brooks [1981]). The basic idea behind this approach is to base the

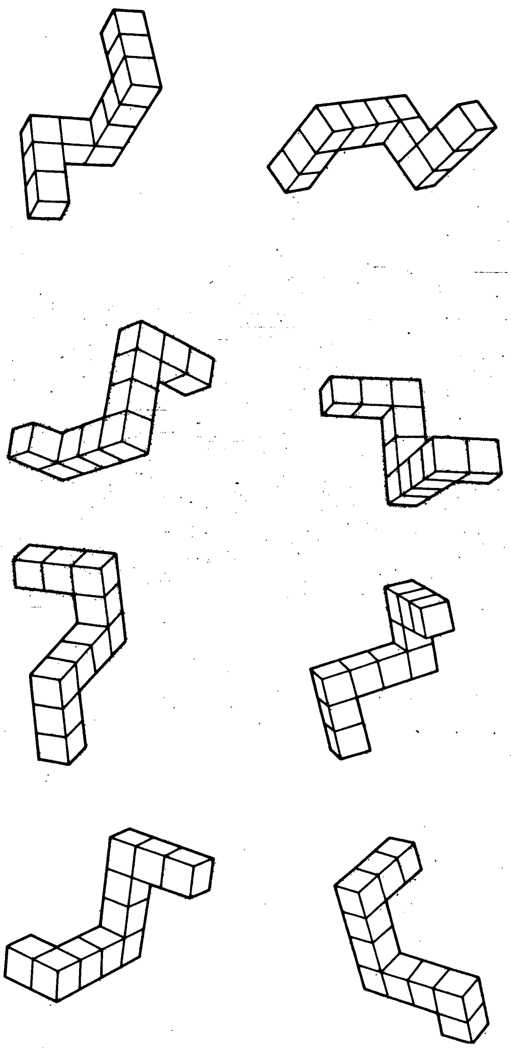


Figure 8.62 Sample three-dimensional structures generated by the rules given in Fig. 8.60. (From Gips [1974], ©Pergamon Press.)

interpretation of a scene on discovering instances of matches between image data and 3D models of volumetric primitives or entire objects of interest. Vision based on 3D models has another important advantage: It provides an approach for handling variances in viewing geometry. Variability in the appearance of an object when viewed from different positions is one of the most serious problems in machine vision. Even in two-dimensional situations where the viewing geometry is fixed, object orientation can strongly influence recognition performance if not handled properly (the reader will recall numerous comments made about this in Sec.

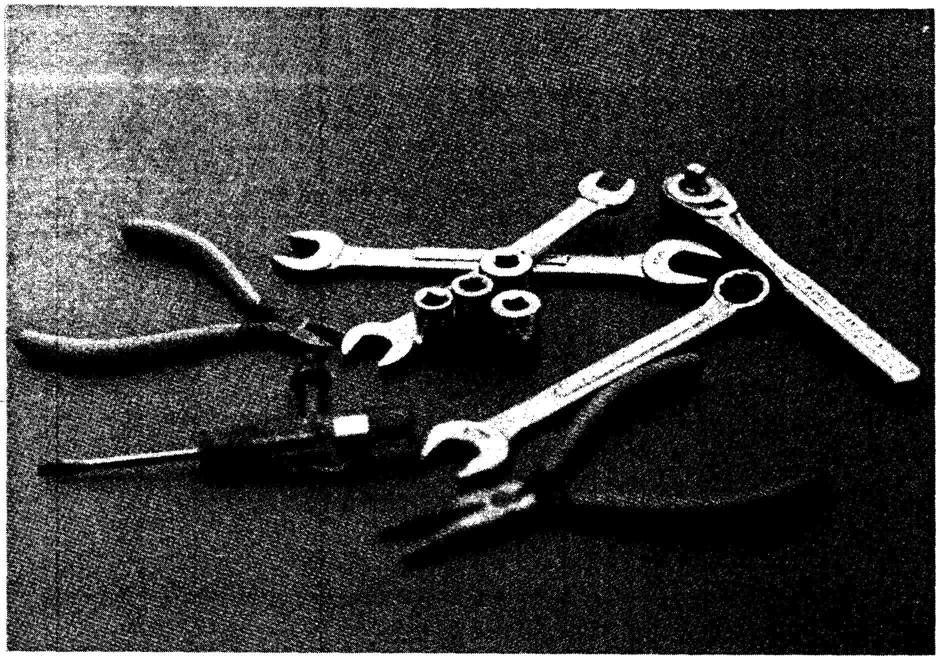


Figure 8.63 Oblique view of a three-dimensional scene.

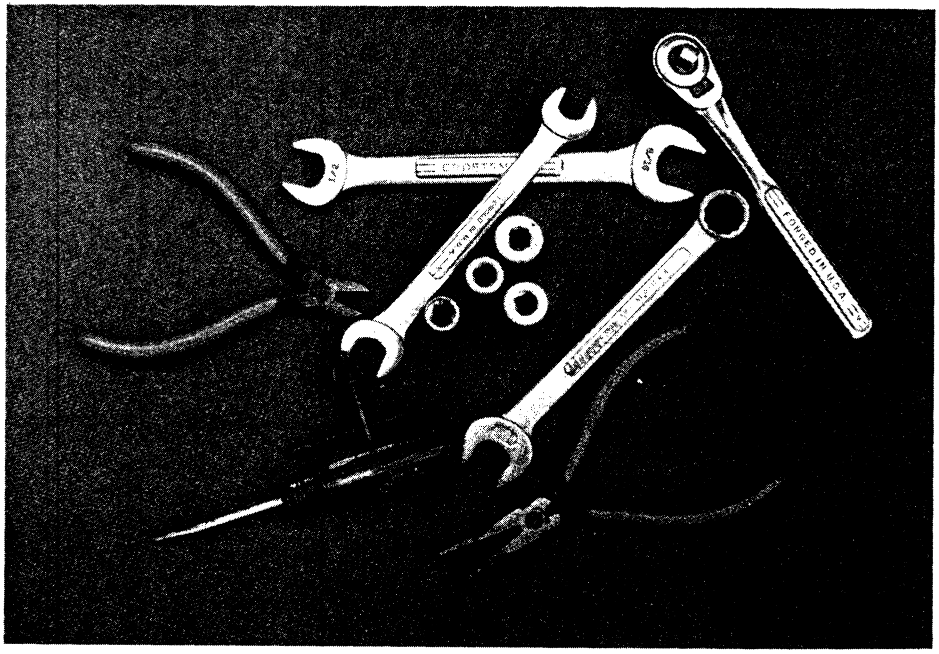


Figure 8.64 Another view of the scene shown in Fig. 8.63.

8.3). One of the advantages of a model-driven approach is that, depending on a known viewing geometry, it is possible to project the 3D model onto an imaging plane (see Sec. 7.4) in that orientation and thus simplify the match between an unknown object and what the system would expect to see from a given viewpoint.

8.7 CONCLUDING REMARKS

The focus of the discussion in this chapter is on concepts and techniques of machine vision with a strong bias toward industrial applications. As indicated in Sec. 8.2, segmentation is one of the most important processes in the early stages of a machine vision system. Consequently, a significant portion of this chapter is dedicated to this topic. Following segmentation, the next task of a vision system is to form a set of descriptors which will uniquely identify the objects of a particular class. As indicated in Sec. 8.3, the key in selecting descriptors is to minimize their dependence on object size, location, and orientation.

Although vision is inherently a three-dimensional problem, most present industrial systems operate on image data which are often idealized via the use of specialized illumination techniques and a fixed viewing geometry. The problems encountered when these constraints are relaxed are addressed briefly in Secs. 8.4 and 8.6.

Our treatment of recognition techniques has been at an introductory level. This is a broad area in which dozens of books and thousands of articles have been written. The references at the end of this chapter provide a pointer for further reading on both the decision-theoretic and structural aspects of pattern recognition and related topics.

REFERENCES

Further reading on the local analysis concepts discussed in Sec. 8.2.1 may be found in the book by Rosenfeld and Kak [1982]. The Hough transform was first proposed by P. V. C. Hough [1962] in a U.S. patent and later popularized by Duda and Hart [1972]. A generalization of the Hough transform for detecting arbitrary shape has been proposed by Ballard [1981]. The material on graph-theoretic techniques is based on two papers by Martelli [1972, 1976]. Another interesting approach based on a minimum-cost search is given in Ramer [1975]. Additional reading on graph searching techniques may be found in Nilsson [1971, 1980]. Edge following may also be approached from a dynamic programming point of view. For further details on this topic see Ballard and Brown [1982].

The optimum thresholding approach discussed in Sec. 8.2.2 was first utilized by Chow and Kaneko [1972] for detecting boundaries in cineagiograms (x-ray pictures of a heart which has been injected with a dye). Further reading on optimum discrimination may be found in Tou and Gonzalez [1974]. The book by Rosenfeld and Kak [1982] contains a number of approaches for threshold selection structural pattern recognition see the books by Pavlidis [1977], Gonzalez and Thomason [1978], and Fu [1982].

Further reading for the material in Sec. 8.6 may be found in Dodd and Rossol [1979] and in Ballard and Brown [1982]. A set of survey papers on the topics discussed in that section has been compiled by Brady [1981].

PROBLEMS

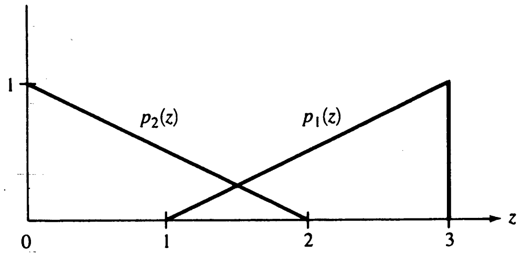
8.1 (*a*) Develop a general procedure for obtaining the normal representation of a line given its slope-intercept equation $y = ax + b$. (*b*) Find the normal representation of the line $y = -2x + 1$.

8.2 (*a*) Superimpose on Fig. 8.7 all the possible edges given by the graph in Fig. 8.8. (*b*) Compute the cost of the minimum-cost path.

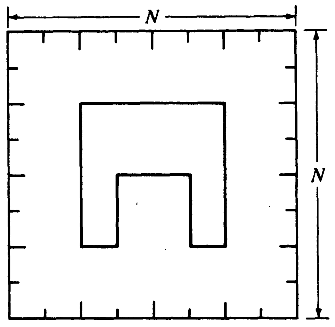
8.3 Find the edge corresponding to the minimum-cost path in the subimage shown below, where the numbers in parentheses indicate intensity. Assume that the edge starts on the first column and ends in the last column.

$$\begin{array}{cccc}
 & 0 & 1 & 2 \\
 & \bullet & \bullet & \bullet \\
0 & (2) & (1) & (0) \\
 & \bullet & \bullet & \bullet \\
1 & (1) & (1) & (7) \\
 & \bullet & \bullet & \bullet \\
2 & (6) & (8) & (2)
\end{array}$$

8.4 Suppose that an image has the following intensity distributions, where $p_1(z)$ corresponds to the intensity of objects and $p_2(z)$ corresponds to the intensity of the background. Assuming that $P_1 = P_2$, find the optimum threshold between object and background pixels.



8.5 Segment the image on page 448 using the split and merge procedure discussed in Sec. 8.2.3. Let $P(R_i)$ = TRUE if all pixels in R_i have the same intensity. Show the quadtree corresponding to your segmentation.



8.6 (*a*) Show that redefining the starting point of a chain code so that the resulting sequence of numbers forms an integer of minimum magnitude makes the code independent of where we initially start on the boundary. (*b*) What would be the normalized starting point of the chain code 11076765543322?

8.7 (*a*) Show that using the first difference of a chain code normalizes it to rotation, as explained in Section 8.3.1. (*b*) Compute the first difference of the code 0101030303323232212111.

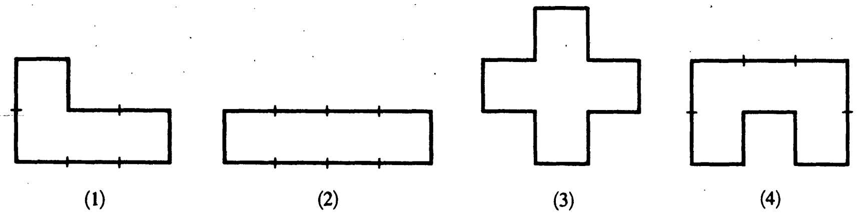
8.8 (*a*) Plot the signature of a square boundary using the tangent angle method discussed in Sec. 8.3.1. (*b*) Repeat for the slope density function. Assume that the square is aligned with the x and y axes and let the x axis be the reference line. Start at the corner closest to the origin.

8.9 Give the fewest number of moment descriptors that would be needed to differentiate between the shapes shown in Fig. 8.29.

8.10 (*a*) Show that the rubberband polygonal approximation approach discussed in Sec. 8.3.1 yields a polygon with minimum perimeter. (*b*) Show that if each cell corresponds to a pixel on the boundary, then the maximum possible error in that cell is $\sqrt{2}d$, where d is the grid distance between pixels.

8.11 (*a*) What would be the effect on the resulting polygon if the error threshold were set to zero in the merging method discussed in Sec. 8.3.1? (*b*) What would be the effect on the splitting method?

8.12 (*a*) What is the order of the shape number in each of the following figures? (*b*) Obtain the shape number for the fourth figure.



8.13 Compute the mean and variance of a four-level image with histogram $p(z_1) = 0.1$, $p(z_2) = 0.4$, $p(z_3) = 0.3$, $p(z_4) = 0.2$. Assume that $z_1 = 0$, $z_2 = 1$, $z_3 = 2$, and $z_4 = 3$.

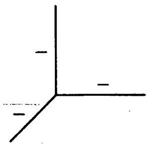
8.14 Obtain the gray-level co-occurrence matrix of a 5×5 image composed of a checkerboard of alternating 1's and 0's if (*a*) P is defined as "one pixel to the right," and (*b*) "two pixels to the right." Assume that the top, left pixel has value 0.

8.15 Consider a checkerboard image composed of alternating black and white squares, each of size $m \times m$. Give a position operator that would yield a diagonal co-occurrence matrix?

8.16 (*a*) Show that the medial axis of a circular region is a single point at its center. (*b*) Sketch the medial axis of a rectangle, the region between two concentric circles, and an equilateral triangle.

8.17 (*a*) Show that the boolean expression given in Eq. (8.3-6) implements the conditions given by the four windows in Fig. 8.40. (*b*) Draw the windows corresponding to B_0 in Eq. (8.3-7).

8.18 Draw a trihedral object which has a junction of the form



8.19 Show that using Eq. (8.5-4) to classify an unknown pattern vector $\mathbf{x}^*$ is equivalent to using Eq. (8.5-3).

8.20 Show that $D(A, B) = 1/k$ satisfies the three conditions given in Eq. (8.5-7).

8.21 Show that $B = \max(|C_1|, |C_2|) - A$ in Eq. (8.5-8) is zero if and only if C_1 and C_2 are identical strings.