Chapter 2

Binary Image Processing

An image contains a continuum of intensity values before it is quantized to obtain a digital image. The information in an image is in these gray values. To interpret an image, the variations in the intensity values must be analyzed. The most commonly used number of quantization levels for representing image intensities is 256 different gray levels. It is not uncommon, however, to see digital images quantized to 32, 64, 128, or 512 intensity levels for certain applications, and even up to 4096 (12 bits) are used in medicine. Clearly, more intensity levels allow better representation of the scene at the cost of more storage.

In the early days of machine vision, the memory and computing power available was very limited and expensive. These limitations encouraged designers of vision applications to focus their efforts on binary vision systems. A binary image contains only two gray levels. The difference this makes in the representation of a scene is shown in Figure 2.1.

In addition, designers noted that people have no difficulty in understanding line drawings, silhouettes, and other images formed using only two gray levels. Encouraged by this human capability, they used binary images in many applications.

Even though computers have become much more powerful, binary vision systems are still useful. First of all, the algorithms for computing properties of binary images are well understood. They also tend to be less expensive and faster than vision systems that operate on gray level or color images. This is due to the significantly smaller memory and processing requirements of binary vision. The memory requirements of a gray level system working

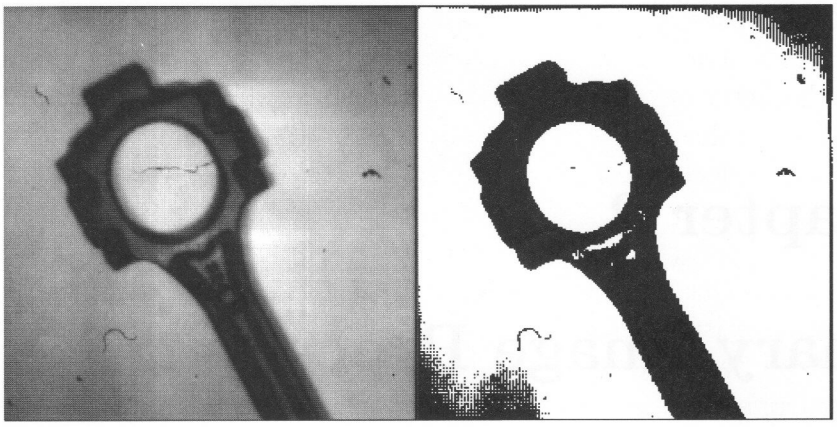


Figure 2.1: A gray level image and its corresponding binary image.

with 256 gray levels will be eight times that of a system working with a binary image of the same size. The storage size may be further reduced by using techniques such as run-length encoding, covered in Section 2.4. The processing time requirements are lower because many operations on binary images may be performed as logical operations instead of integer arithmetic operations.

Smaller memory requirements and faster execution times are not the only reasons for studying binary vision systems. Many techniques developed for these systems are also applicable to vision systems which use gray scale images. A convenient way to represent an object in a gray level or color image is to use its mask. The mask of an object is a binary picture in which the object points are 1 and other points are 0. After an object has been separated from the background, its geometric and topological properties may be required in decision making. These properties can be computed from its binary image. All the techniques discussed in this chapter can be applied to a region in a gray image. Thus, though we will discuss these techniques in the context of binary images, their application is not limited to binary images.

In general, binary vision systems are useful in cases where a silhouette contains enough information to allow recognition of an object and where the environment can be adequately controlled. To obtain a good silhouette, the objects must be easily separated from the background. This can be achieved by using special illumination techniques and by having only a few

objects in the scene. There are many industrial situations that fulfill these requirements. For example, binary vision systems have found application in optical character recognition, chromosome analysis, and recognition of industrial parts. In these cases, the binary vision system usually uses a threshold to separate objects from the background. The proper value of this threshold depends on illumination and on reflectance characteristics of objects. The resulting binary picture allows computation of geometric and topological properties (*features*) of objects for the given task. In many applications, these characteristics are enough for recognition of objects.

It should be mentioned here, however, that with the increase in the complexity of applications, more and more vision systems are using gray scale images. This is due to the fact that in many material handling and assembly tasks, the illumination cannot be controlled to obtain good contrast between objects and background. Care has to be exercised to make the system insensitive to small changes in illumination and reflectance characteristics of other objects in a scene. In many applications, this becomes a formidable task. Similarly, in inspection tasks, it may not be possible to recover subtle information using only two intensity levels. Internal details of an object may be lost in thresholding and may make the task of detecting surface defects very difficult.

Certain generally used conventions concerning binary images will be followed in this chapter. Object pixels will have the value 1 and background pixels will have 0. In displaying pictures, 0 is white and 1 is black; thus, in binary images, the background is white and objects are black. We will also assume that pictures are of size $n \times m$ pixels and are represented in a computer as a two-dimensional array. This representation allows us to visualize images with the spatial relationships between points maintained in the form familiar to people.

The techniques discussed in this chapter, though simple, have played a very important role in industrial vision. We will study the following aspects of binary vision systems in this chapter:

- Formation of binary images
- Geometric properties
- Topological properties

- Object recognition in binary images

Many concepts discussed here are used in all aspects of machine vision. Many definitions are related to digital geometry and are useful in discussions related to sampled images. In general, after an image has been segmented into several objects, each object is represented as a region. Discussions related to these object regions use the terminology and concepts discussed in this chapter.

2.1 Thresholding

One of the most important problems in a vision system is to identify the subimages that represent objects. This operation, which is so natural and so easy for people, is surprisingly difficult for computers. The partitioning of an image into regions is called segmentation. Ideally, a partition represents an object or part of an object. Formally, segmentation can be defined as a method to partition an image, $F[i,j]$, into subimages, called regions, $P_1, \ldots, P_k$, such that each subimage is an object candidate.

Definition 2.1 *A region is a subset of an image.*

Definition 2.2 *Segmentation is grouping pixels into regions, such that*

- $\cup_{i=1}^{k} P_i =$ Entire image ($\{P_i\}$ is an *exhaustive* partitioning.)
- $P_i \cap P_j = \emptyset$, $i \neq j$ ($\{P_i\}$ is an *exclusive* partitioning.)
- Each region P_i satisfies a predicate; that is, all points of the partition have *some* common property.
- Pixels belonging to adjacent regions, when taken jointly, do not satisfy the predicate.

As shown above, a partition satisfies a predicate. This predicate may be as simple as *has uniform intensity* but is more complex in most applications. Segmentation is a very important step in understanding images.

A binary image is obtained using an appropriate segmentation of a gray scale image. If the intensity values of an object are in an interval and the

intensity values of the background pixels are outside this interval, a binary image can be obtained using a thresholding operation that sets the points in that interval to 1 and points outside that range to 0. Thus, for binary vision, segmentation and thresholding are synonymous. Many cameras have been designed to perform this thresholding operation in hardware. The output of such a camera is a binary image. In most applications, however, cameras give a gray scale image and the binary image is obtained using thresholding.

Thresholding is a method to convert a gray scale image into a binary image so that objects of interest are separated from the background. For thresholding to be effective in *object-background separation*, it is necessary that the objects and background have sufficient contrast and that we know the intensity levels of either the objects or the background. In a fixed thresholding scheme, these intensity characteristics determine the value of the threshold.

Let us assume that a binary image $B[i,j]$ is the same as a thresholded gray image $F_T[i,j]$ which is obtained using a threshold T for the original gray image $F[i,j]$. Thus,

$$B[i,j] = F_T[i,j] \tag{2.1}$$

where for a darker object on a lighter background

$$F_T[i,j] = \begin{cases} 1 & \text{if } F[i,j] \leq T \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

If it is known that the object intensity values are in a range $[T_1, T_2]$, then we may use

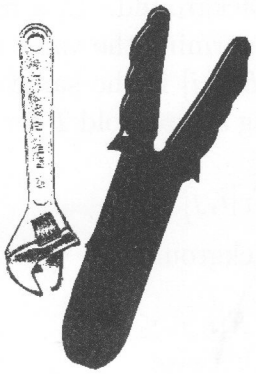
$$F_T[i,j] = \begin{cases} 1 & \text{if } T_1 \leq F[i,j] \leq T_2 \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

A general thresholding scheme in which the intensity levels for an object may come from several disjoint intervals may be represented as

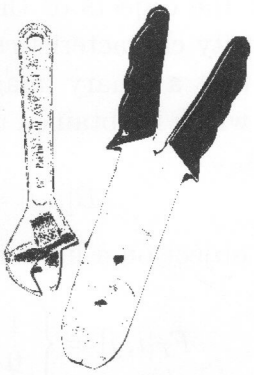
$$F_T[i,j] = \begin{cases} 1 & \text{if } F[i,j] \in Z \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

where Z is a set of intensity values for object components. The results of producing an image using different thresholds are shown in Figure 2.2.

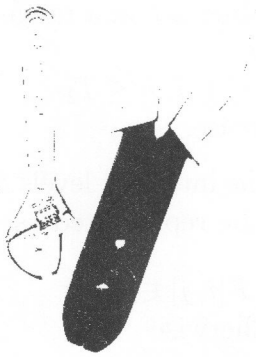
Note how knowledge about the application domain is incorporated into the thresholding algorithm. It has, in fact, been tailored for the domain; therefore, the same threshold values may not work in a new domain. The



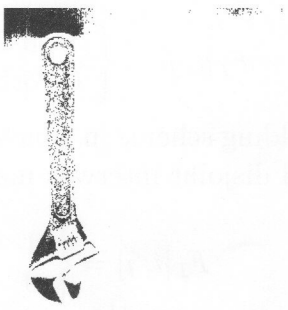
$T = 48$



$T_1 = 21, T_2 = 48$



$T = 21$



$T_1 = 135, T_2 = 255$

Figure 2.2: A gray level image and its resulting binary images using different thresholds. *Top:* Original gray-level image. *Middle left:* Original image thresholded with $T = 48$. *Middle right:* $T_1 = 21$ and $T_2 = 48$. *Bottom left:* $T = 21$. *Bottom right:* $T = 135$ and $T_2 = 255$.

threshold is usually selected on the basis of experience with the application domain. In some cases, the first few runs of the system may be used for interactively analyzing a scene and determining an appropriate value for the threshold.

Automatic thresholding of images is often the first step in the analysis of images in machine vision systems. Many techniques have been developed for utilizing the intensity distribution in an image and the knowledge about the objects of interest for selecting a proper threshold value automatically. This was briefly introduced in Figure 1.11, where an image and its histogram are given. Many such automatic methods for thresholding an image are presented in Section 3.2.

2.2 Geometric Properties

Suppose that a thresholding scheme has given us objects in an image. The next step is to recognize and locate objects. In most industrial applications, the camera location and the environment are known. Using simple geometry, one may find the three-dimensional locations of objects from their two-dimensional positions in images. Moreover, in most applications the number of different objects is not large. If the objects are different in size and shape, the size and shape features of objects may be determined from their images to help the system recognize them. Many applications in industry have utilized some simple features of regions for determining the locations of objects and for recognizing them (e.g., size, position, orientation).

If there are several objects, one can compute these features for each object. An object is usually represented by a connected component or a region. The concept of connectedness and the algorithms for finding connected components in an image will be discussed later in this chapter. For the present discussion, let us assume that an image has only one object.

2.2.1 Size

In general, for a binary image it is well known that the area A is given by

$$A = \sum_{i=1}^{n} \sum_{j=1}^{m} B[i, j]. \tag{2.5}$$

This is the zeroth-order moment.

2.2.2 Position

The position of an object in an image plays an important role in many applications. There are different ways to specify the position of an object, such as using its enclosing rectangle or centroid. In industrial applications, objects usually appear on a known surface, such as a table, and the position of the camera is known with respect to the table. In such cases, an object's position in the image determines its spatial location. The position of an object in an image may be defined using the center of area of the object image. Though other methods such as the enclosing rectangle of the object image may be used, the center of area is a point and is relatively insensitive to noise in the image.

The center of area in binary images is the same as the center of mass if we consider the intensity at a point as the mass at that point. To calculate the position of the object, we use

$$\bar{x} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} B[i,j] = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} jB[i,j] \tag{2.6}$$

$$\bar{y} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} B[i,j] = -\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} iB[i,j] \tag{2.7}$$

where $\bar{x}$ and $\bar{y}$ are the coordinates of the center of the region measured with respect to the top left pixel. Thus, the position of an object is

$$\bar{x} = \frac{\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} jB[i,j]}{A} \tag{2.8}$$

$$\bar{y} = \frac{-\sum_{i=0}^{n-1} \sum_{j=0}^{m-1} iB[i,j]}{A}. \tag{2.9}$$

These are the first-order moments. The position calculated using first moments is not necessarily an integer and usually lies between the integer values of the image array indices. We emphasize that this does *not* imply that the calculated position is better than the resolution of pixel coordinates.

2.2.3 Orientation

Calculating the orientation of an object is a little more complex than calculating its position. For some shapes (such as circles), orientation is not unique. To define unique orientation, an object must be elongated. If so, the orientation of the axis of elongation can be used to define the orientation of the object. Usually, the axis of least second moment, which in 2-D is equivalent to the axis of least inertia, is used as the axis of elongation.

The axis of second moment for an object image is that line for which the sum of the squared distances between object points and the line is minimum. Given a binary image, $B[i,j]$, compute the least-squares fit of a line to the object points in the binary image. Minimize the sum of the squared perpendicular distances of all object points from the line,

$$\chi^2 = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} r_{ij}^2 B[i,j], \tag{2.10}$$

where r_{ij} is the perpendicular distance from an object point $[i,j]$ to the line. To avoid numerical problems when the line is nearly vertical, represent the line in polar coordinates:

$$\rho = x\cos\theta + y\sin\theta. \tag{2.11}$$

As shown in Figure 2.3, θ is the orientation of the normal to the line with the x axis, and ρ is the distance of the line from the origin. The distance r of a point (x,y) is obtained by plugging the coordinates of the point into the equation for the line:

$$r^2 = (x\cos\theta + y\sin\theta - \rho)^2. \tag{2.12}$$

Plugging the representation of the line into the minimization criterion yields the regression problem for fitting a straight line to the object points. Determine the model parameters ρ and θ by minimizing

$$\chi^2 = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} (x_{ij}\cos\theta + y_{ij}\sin\theta - \rho)^2 B[i,j]. \tag{2.13}$$

Setting the derivative of χ^2 with respect to ρ to zero and solving for ρ yields

$$\rho = (\bar{x}\cos\theta + \bar{y}\sin\theta), \tag{2.14}$$

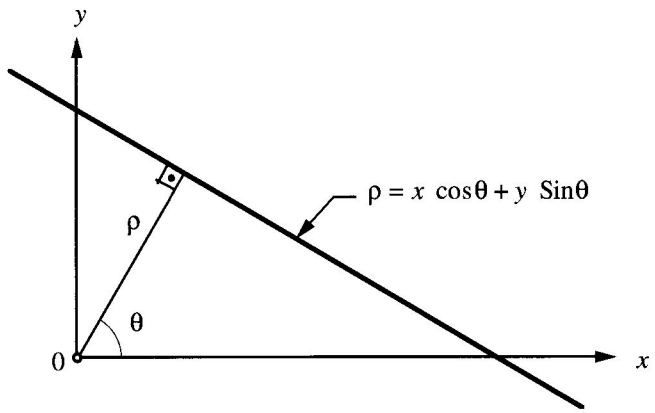


Figure 2.3: Polar representation of a straight line.

which shows that the regression line passes through the center of object points $(\bar{x}, \bar{y})$. After substituting this value of ρ in the above equation for χ^2 and replacing

$$x' = x - \bar{x} \qquad y' = y - \bar{y}, \tag{2.15}$$

the minimization problem becomes

$$\chi^2 = a\cos^2\theta + b\sin\theta\cos\theta + c\sin^2\theta. \tag{2.16}$$

The parameters

$$a = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}(x'_{ij})^2 B[i,j] \tag{2.17}$$

$$b = 2\sum_{i=0}^{n-1}\sum_{j=0}^{m-1} x'_{ij}y'_{ij}B[i,j] \tag{2.18}$$

$$c = \sum_{i=0}^{n-1}\sum_{j=0}^{m-1}(y'_{ij})^2 B[i,j] \tag{2.19}$$

are the second-order moments. The expression for χ^2 can be rewritten as

$$\chi^2 = \frac{1}{2}(a+c) + \frac{1}{2}(a-c)\cos 2\theta + \frac{1}{2}b\sin 2\theta. \tag{2.20}$$

Differentiating χ^2, setting the result to zero, and solving for θ yields

$$\tan 2\theta = \frac{b}{a-c}. \tag{2.21}$$

The orientation of the axis is given by

$$\sin 2\theta = \pm \frac{b}{\sqrt{b^2 + (a-c)^2}} \tag{2.22}$$

$$\cos 2\theta = \pm \frac{a-c}{\sqrt{b^2 + (a-c)^2}}. \tag{2.23}$$

The axis of orientation is obtained for the minimal value of χ^2. Note that if $b = 0$ and $a = c$, the object does not have a unique axis of orientation. The elongation E of the object is the ratio of the largest to smallest values for χ:

$$E = \frac{\chi_{\max}}{\chi_{\min}}. \tag{2.24}$$

When the expressions for $\sin 2\theta$ and $\cos 2\theta$ are substituted into Equation 2.20, their signs determine whether χ^2 is a maximum or minimum. Note that the elongation is 1 for a circle and that this is the lower bound on E.

2.3 Projections

The projection of a binary image onto a line may be obtained by partitioning the line into bins and finding the number of 1 pixels that are on lines perpendicular to each bin. A simple example of this is shown in Figure 2.4. Projections are compact representations of images, since much useful information is retained in the projection. However, projections are not unique in the sense that more than one image may have the same projection. Horizontal and vertical projections can be easily obtained by finding the number of 1 pixels for each bin in the vertical and horizontal directions, respectively, as shown in Figure 2.5.

The projection $H[i]$ along the rows and the projection $V[j]$ along the columns of a binary image are given by

$$H[i] = \sum_{j=0}^{m-1} B[i,j] \tag{2.25}$$

$$V[j] = \sum_{i=0}^{n-1} B[i,j]. \tag{2.26}$$

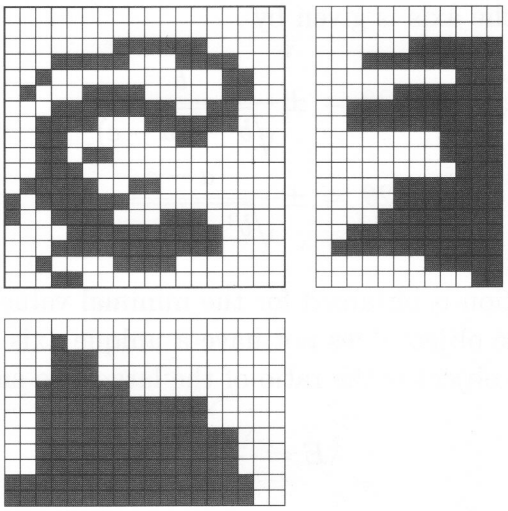


Figure 2.4: The binary image of a lizard with its horizontal and vertical projections.

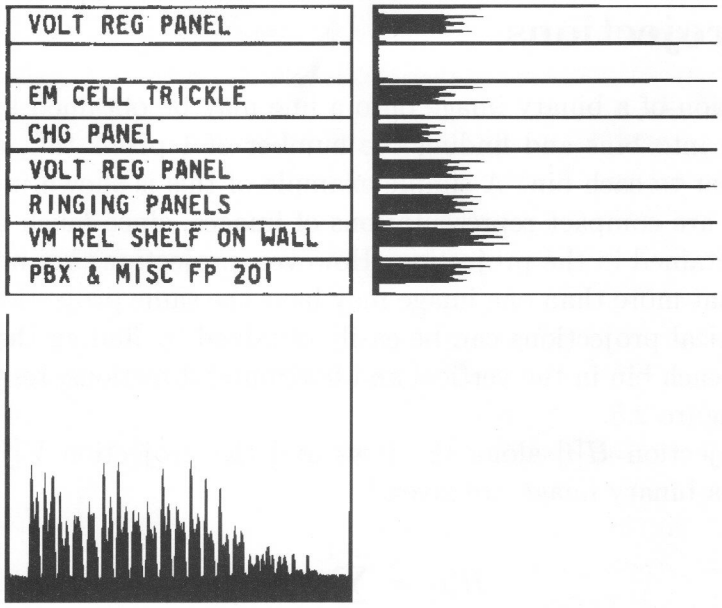


Figure 2.5: Horizontal and vertical projections of an image.

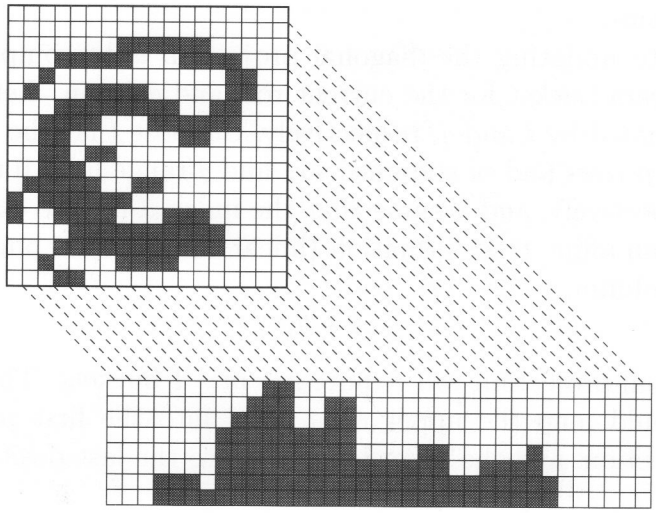


Figure 2.6: The binary lizard image with its diagonal projection.

A general projection onto any line may be defined. An example of a diagonal projection is given in Figure 2.6.

It can be shown that the first moments of an image equal the first moments of its projections. Since calculating the position of an object requires only the first moments, the position can be computed from the horizontal and vertical projections. Thus,

$$A = \sum_{j=0}^{m-1} V[j] = \sum_{i=0}^{n-1} H[i] \tag{2.27}$$

$$\bar{y} = \frac{\sum_{i=0}^{n-1} iH[i]}{A} \tag{2.28}$$

$$\bar{x} = \frac{\sum_{j=0}^{m-1} jV[j]}{A}. \tag{2.29}$$

As we saw in the preceding section, the orientation of an object requires knowledge of the second moment. The second moment can be computed from the diagonal projection of an image. Thus, orientation may be computed using horizontal, vertical, and diagonal projections.

In some applications, projections can be used as features for recognition of objects. Projections offer a compact representation and allow application

of fast algorithms.

The trick to updating the diagonal projection is to compute the index for the histogram bucket for the current row and column. Let the row and column be denoted by i and j, respectively. Suppose that the dimensions of the image are n rows and m columns, so i and j range from 0 to $n-1$ and 0 to $m-1$, respectively, and assume that the index d for the diagonal can be computed by an affine transformation (linear combination plus constant) of the row and column:

$$d = ai + bj + c. \tag{2.30}$$

The diagonal projection will require $n+m-1$ buckets. The affine transformation should map the upper right pixel into the first position of the diagonal projection, and the lower left pixel into the last position. Solve the equations

$$\begin{aligned} a \cdot 0 + b\,(m-1) + c &= 0 && (2.31)\\ a\,(n-1) + b \cdot 0 + c &= n+m-2 && (2.32)\\ a &= -b && (2.33) \end{aligned}$$

to obtain the formula

$$d = i - j + m - 1. \tag{2.34}$$

2.4 Run-Length Encoding

Another compact representation of a binary image is its run-length encoding. In this representation, numbers indicating the lengths of the runs of 1 pixels in the image are used to represent the image. This coding has been used for image transmission. Additionally, some properties, such as the area of objects, may be directly computed from their run-length codes.

Two approaches are commonly used in run-length encoding. In the first, the start position and lengths of runs of 1s for each row are used. The other approach uses only lengths of runs, starting with the length of the 1 run. We will use the second convention and represent run lengths for each row of an image. Thus, $r_{i,k}$ denotes the length of the kth run in the ith row of an image. The run-length codes for an image are shown in Figure 2.7.

Binary image:

1	1	1	0	0	0	1	1	0	0	0	1	1	1	1	0	1	1	0	1	1	1
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Start and length of 1 runs: $(1,3)\ (7,2)\ (12,4)\ (17,2)\ (20,3)$
$(5,13)\ (19,4)$
$(1,3)\ (17,6)$

Length of 1 and 0 runs: $3,3,2,3,4,1,2,1,3$
$0,4,13,1,4$
$3,13,6$

Figure 2.7: Run-length codes for a simple binary image.

The area of all objects can be obtained by summing the lengths of all 1 runs.

$$A = \sum_{i=0}^{n-1} \sum_{k=0}^{\left(\frac{m_i-1}{2}\right)} r_{i,2k+1} \tag{2.35}$$

where m_i is the number of runs in the ith row.

The horizontal projection can be easily computed from the run-length code, without generating the image (see Figure 2.8). The vertical and diagonal projections can also be computed from the run-length code without generating the image by using clever code. Area and first and second moments can be calculated from the projections as explained in Section 2.3. Calculating other properties of images may require partial or complete regeneration of the image.

2.5 Binary Algorithms

Segmenting object pixels from background pixels is a difficult problem. We will not address this problem here. Let us assume here that somehow an object can be defined and, using a predicate, the points of an image belonging

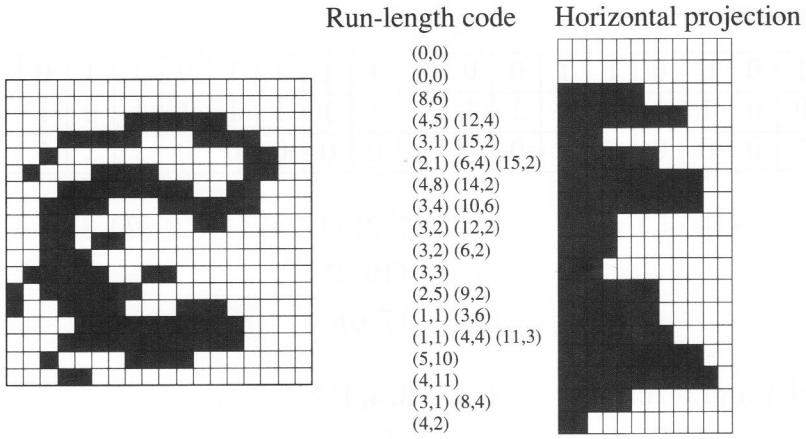


Figure 2.8: Horizontal projection calculated from the run-length code.

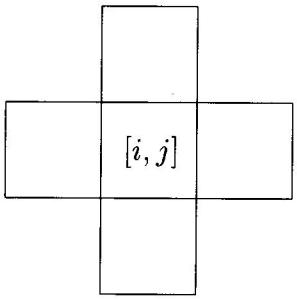
to an object may be labeled. The problem then is to group together all points of an image that are labeled as object points into an object image. In this chapter we will assume that all such points are spatially close. This notion of *spatial proximity* requires a more precise definition so that an algorithm may be devised to group spatially close points into a component. For this purpose, let us introduce some definitions.

2.5.1 Definitions

Neighbors

A pixel in a digital image is spatially close to several other pixels. In a digital image represented on a square grid, a pixel has a common boundary with four pixels and shares a corner with four additional pixels. We say that two pixels are *4-neighbors* if they share a common boundary. Similarly, two pixels are *8-neighbors* if they share at least one corner. For example, the pixel at location $[i, j]$ has 4-neighbors $[i+1, j]$, $[i-1, j]$, $[i, j+1]$, and $[i, j-1]$. The 8-neighbors of the pixel include the 4-neighbors plus $[i+1, j+1]$, $[i+1, j-1]$, $[i-1, j+1]$ and $[i-1, j-1]$. A pixel is said to be *4-connected* to its 4-neighbors and *8-connected* to its 8-neighbors (see Figure 2.9).

4-neighbors $[i+1, j]$, $[i-1, j]$, $[i, j+1]$, $[i, j-1]$



8-neighbors $[i+1, j+1]$, $[i+1, j-1]$, $[i-1, j+1]$, $[i-1, j-1]$ plus all of the 4-neighbors

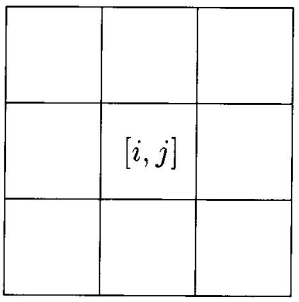
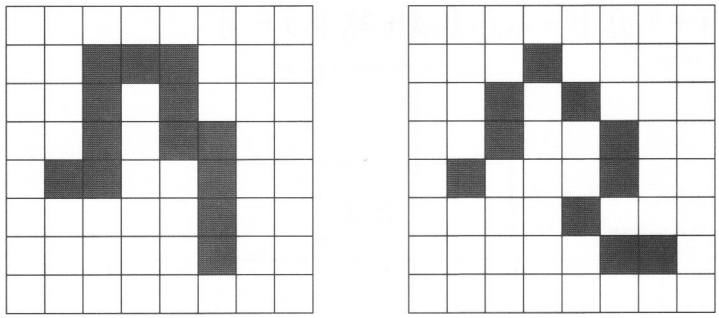


Figure 2.9: The 4- and 8-neighborhoods for a rectangular image tessellation. Pixel $[i, j]$ is located in the center of each figure.



(a) 4-path (b) 8-path

Figure 2.10: Examples of a 4-path and an 8-path.

Path

A *path* from the pixel at $[i_0, j_0]$ to the pixel at $[i_n, j_n]$ is a sequence of pixel indices $[i_0, j_0], [i_1, j_1], [i_2, j_2], \ldots, [i_n, j_n]$ such that the pixel at $[i_k, j_k]$ is a neighbor of the pixel at $[i_{k+1}, j_{k+1}]$ for all k with $0 \leq k \leq n-1$. If the neighbor relation uses 4-connection, then the path is a 4-path; for 8-connection, the path is an 8-path. Simple examples of these are shown in Figure 2.10.

Foreground

The set of all 1 pixels in an image is called the *foreground* and is denoted by S.

Connectivity

A pixel $p \in S$ is said to be *connected* to $q \in S$ if there is a path from p to q consisting entirely of pixels of S.

Note that connectivity is an equivalence relation. For any three pixels p, q, and r in S, we have the following properties:

1. Pixel p is connected to p (reflexivity).

2. If p is connected to q, then q is connected to p (commutativity).

3. If p is connected to q and q is connected to r, then p is connected to r (transitivity).

Connected Components

A set of pixels in which each pixel is connected to all other pixels is called a *connected component.*

Background

The set of all connected components of $\bar{S}$ (the complement of S) that have points on the border of an image is called the *background.* All other components of $\bar{S}$ are called *holes.*

Let us consider the simple picture shown below:

	1	
1		1
	1	

How many objects and how many holes are in this figure? If we consider 4-connectedness for both foreground and background, there are four objects that are 1 pixel in size and there is one hole. If we use 8-connectedness, then there is one object and no hole. Intuitively, in both cases we have an ambiguous situation. A similar ambiguous situation arises in a simple case like:

1	0
0	1

where if the 1s are connected, then the 0s should not be.

To avoid this awkward situation, different connectedness should be used for objects and backgrounds. If we use 8-connectedness for S, then 4-connectedness should be used for $\bar{S}$.

Boundary

The *boundary* of S is the set of pixels of S that have 4-neighbors in $\bar{S}$. The boundary is usually denoted by S'.

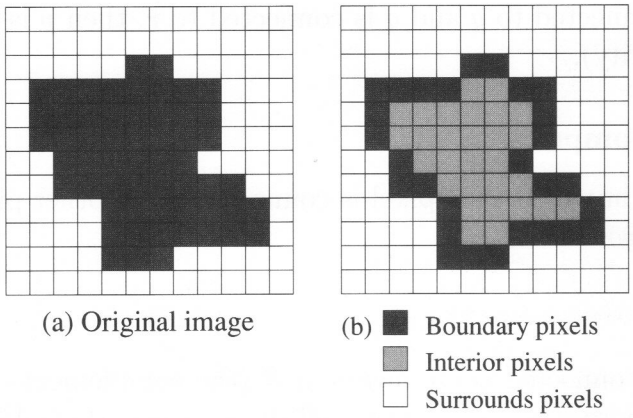


Figure 2.11: A binary image with its boundary, interior, and surrounds.

Interior

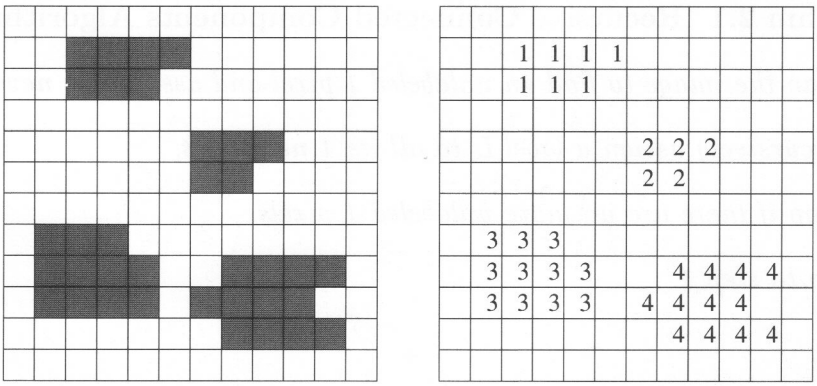
The *interior* is the set of pixels of S that are not in its boundary. The interior of S is $(S - S')$.

Surrounds

Region T *surrounds* region S (or S is inside T), if any 4-path from any point of S to the border of the picture must intersect T. Figure 2.11 shows an example of a simple binary image with its boundary, interior, and surrounds.

2.5.2 Component Labeling

One of the most common operations in machine vision is finding the connected components in an image. The points in a connected component form a candidate region for representing an object. As mentioned earlier, in computer vision most objects have surfaces. Points belonging to a surface project to spatially close points. The notion of "spatially close" is captured by connected components in digital images. It should be mentioned here that connected component algorithms usually form a bottleneck in a binary vision system. The algorithm is sequential in nature, because the operation of finding connected components is a global operation. If there is only one



(a) (b)

Figure 2.12: An image (a) and its connected component image (b).

object in an image, then there may not be a need for finding the connected components; however, if there are many objects in an image and the object properties and locations need to be found, then the connected components must be determined.

A component labeling algorithm finds all connected components in an image and assigns a unique label to all points in the same component. Figure 2.12 shows an image and its labeled connected components. In many applications, it is desirable to compute characteristics (such as size, position, orientation, and bounding rectangle) of the components while labeling these components. There are two algorithms for component labeling: recursive and sequential.

Recursive Algorithm

A recursive algorithm is given as Algorithm 2.1. On a sequential processor, this algorithm is very inefficient. Due to its inefficiency, this algorithm is rarely used on general-purpose computers; but is commonly used on parallel machines.

Algorithm 2.1 Recursive Connected Components Algorithm

1. *Scan the image to find an unlabeled 1 pixel and assign it a new label L.*
2. *Recursively assign a label L to all its 1 neighbors.*
3. *Stop if there are no more unlabeled 1 pixels.*
4. *Go to step 1.*

Sequential Algorithm

The sequential algorithm usually requires two passes over the image. Since this algorithm works with only two rows of an image at a time, it can be used even when images are stored as a file and space limitations do not allow the full image to be brought into memory. This algorithm, given as Algorithm 2.2, looks at the neighborhood of a pixel and tries to assign already used labels to a 1 pixel. In case of two different labels in the neighborhood of a pixel, an *equivalence table* is prepared to keep track of all labels that are equivalent. This table is used in the second pass to assign a unique label to all pixels of a component.

In the algorithm, there are three cases of interest when scanning an image left to right and top to bottom. The algorithm looks at only two of a pixel's 4-neighbors: the ones above and to the left of it. Note that these two pixels have already been seen by the algorithm. If none of these pixels is 1, then the pixel requires a new label. If only one of the two pixels is 1 and has been assigned a label L, then the pixel will be assigned L. If both pixels are 1s and have been assigned the same label L, then the new pixel will be assigned L; however, in the case where the neighbors have been assigned different labels M and N, then the two labels have been used for the same component and they must be merged. In this case the pixel is assigned one of the labels, usually the smaller label, and both labels are recorded as equivalent labels in the equivalence table.

The equivalence table contains the information to assign unique labels to each connected component. In the first scan, all those labels that belong to one component are declared equivalent. In the second pass, one label from an equivalent set is selected to be assigned to all pixels of a component. Usually

Algorithm 2.2 Sequential Connected Components Algorithm using 4-connectivity

1. *Scan the image left to right, top to bottom.*
2. *If the pixel is 1, then*
   (a) *If only one of its upper and left neighbors has a label, then copy the label.*
   (b) *If both have the same label, then copy the label.*
   (c) *If both have different labels, then copy the upper's label and enter the labels in the equivalence table as equivalent labels.*
   (d) *Otherwise assign a new label to this pixel and enter this label in the equivalence table.*
3. *If there are more pixels to consider, then go to step 2.*
4. *Find the lowest label for each equivalent set in the equivalence table.*
5. *Scan the picture. Replace each label by the lowest label in its equivalent set.*

the smallest label is assigned to a component. The second pass assigns an unique label to each component.

After all of the connected components have been found, the equivalence table should be renumbered so that gaps in the labels are eliminated; then the image is rescanned using the equivalence table as a lookup table for renumbering the labels in the image.

Area, first moments, and second moments can be calculated for each component as part of the sequential connected components algorithm. Of course, separate variables must be used to accumulate the moment information for each region. When regions are merged, the moment accumulations for each region are just added together.

2.5.3 Size Filter

It is very common to use thresholding for finding a binary image. In most cases, there are some regions in an image that are due to noise. Usually, such

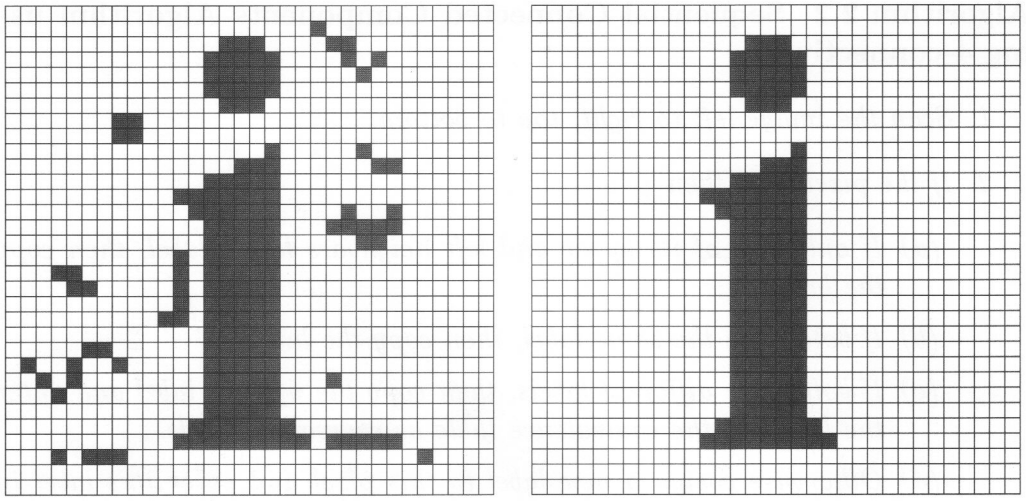


Figure 2.13: A noisy binary image of the letter "i" (left) and the resulting image after the application of a size filter (right). $T = 10$.

regions are small. In many applications, it is known that objects of interest are of size greater than T_0 pixels. In such cases one may use a size filter to remove noise after component labeling. All components below T_0 in size are removed by changing the corresponding pixels to 0. This simple filtering mechanism is very effective in removing noise. Figures 2.13 and 2.14 show two examples of the application of a size filter to a noisy character image.

2.5.4 Euler Number

In many applications, the genus or *Euler number* is used as a feature of an object. *Genus* is defined as the number of components minus the number of holes. Thus,

$$E = C - H \tag{2.36}$$

where E, C, and H are the Euler number, the number of connected components, and the number of holes, respectively. This provides a simple topological feature that is invariant to translation, rotation, and scaling. Several algorithms have been developed that allow computation of the genus using local structures such as runs. Figure 2.15 gives examples of objects with their corresponding Euler numbers.

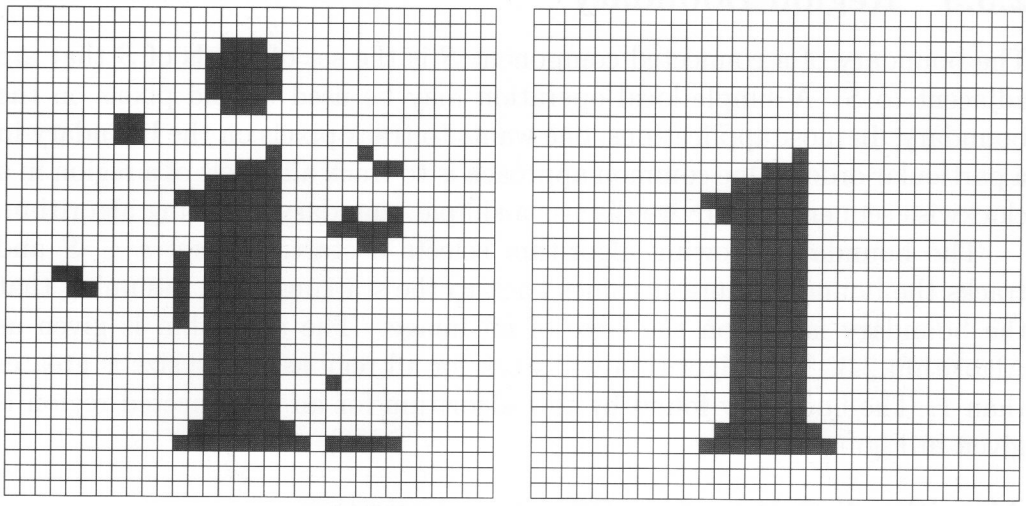
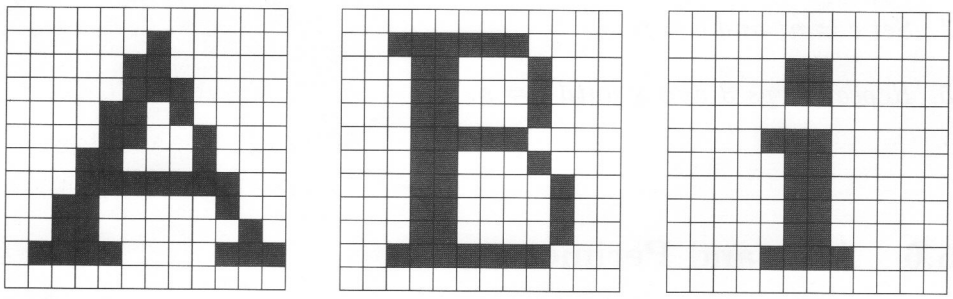


Figure 2.14: The possible error resulting from a poor choice of threshold for the size filter ($T = 25$). Note that the "dot" is missing from the letter "i."



$E = 0$ $E = -1$ $E = 2$

Figure 2.15: The letters "A," "B," and "i" and their Euler numbers. Note that 8-connectivity is used for the foreground and 4-connectivity for the background.

2.5.5 Region Boundary

The boundary of a connected component S is the set of pixels of S that are adjacent to $\bar{S}$. A simple local operation may be used to find pixels on the boundary. In most applications, one wants to track pixels on the boundary in a particular order. One common approach is to track all pixels of a region in a clockwise sequence. Here we discuss a simple boundary-following algorithm.

The boundary-following algorithm selects a starting pixel $s \in S$ and tracks the boundary until it comes back to the starting pixel, assuming that the boundary is not on the edge of an image. The algorithm is given as Algorithm 2.3. This algorithm will work for all regions whose size is greater than 1. The boundary found by this algorithm for an 8-connected region is given in Figure 2.16.

Algorithm 2.3 Boundary-Following Algorithm

1. *Find the starting pixel $s \in S$ for the region using a systematic scan, say from left to right and from top to bottom of the image.*
2. *Let the current pixel in boundary tracking be denoted by c. Set $c = s$ and let the 4-neighbor to the west of s be $b \in \bar{S}$.*
3. *Let the eight 8-neighbors of c starting with b in clockwise order be $n_1, n_2, \ldots, n_8$. Find n_i, for the first i that is in S.*
4. *Set $c = n_i$ and $b = n_{i-1}$.*
5. *Repeat steps 3 and 4 until $c = s$.*

2.5.6 Area and Perimeter

As we discussed earlier, the area is the number of pixels in S. If there are several components $S_1, S_2, \ldots, S_n$, then the area of each component is the number of pixels in that component. The number of pixels in each component may be obtained along with the labeling of the components. In a general case, the area of each of the n components may be obtained in one scan of an image.

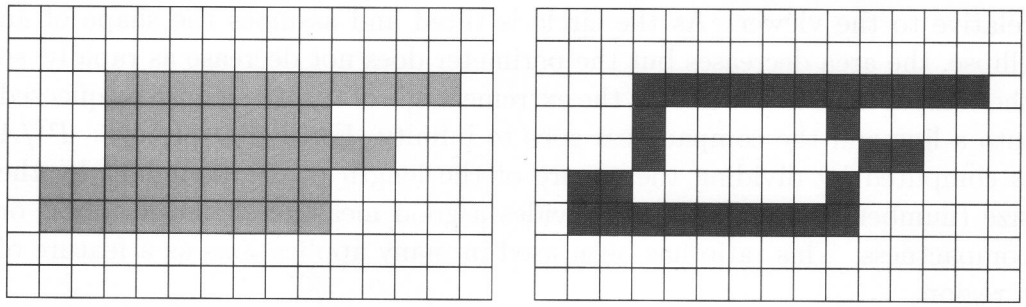


Figure 2.16: Results of a boundary-following algorithm. *Left:* Original binary object. *Right:* Calculated boundary.

The perimeter of a component may be defined in many different ways. Some common definitions are:

1. The sum of lengths of the "cracks" separating pixels of S from pixels of $\bar{S}$. A crack is a line that separates a pair of pixels p and q such that $p \in S$ and $q \in \bar{S}$.

2. The number of steps taken by a boundary-following algorithm.

3. The number of boundary pixels of S.

The measured perimeter will be very different according to different definitions. In general, the perimeter obtained using definition 1 is much longer than the perimeter measurements obtained using the other two definitions.

2.5.7 Compactness

It is well known that the compactness of a continuous geometric figure is measured by the isoperimetric inequality

$$\frac{P^2}{A} \geq 4\pi \tag{2.37}$$

where P and A are the figure's perimeter and area, respectively. A circle is the most compact figure (i.e., has the smallest compactness value) according to this measure. In the case of a circle, the ratio P^2/A achieves its minimal value of 4π; for other figures the ratio is larger. Consider a circle at an angle

relative to the viewer. As the circle is tilted and assumes the shape of an ellipse, the area decreases but the perimeter does not decrease as rapidly, so the compactness increases. At the extreme angle of tilt, the ellipse is squeezed into a line and the compactness goes to infinity. For digital pictures, P^2/A is computed by dividing the square of the length of the boundary by the size (number of pixels). This provides a good measure of dispersedness or compactness. This ratio has been used in many applications as a feature of a region.

Another way of viewing compactness is that a more compact region encloses a larger amount of area for a given perimeter. Note that a square is more compact than a rectangle with the same perimeter.

2.5.8 Distance Measures

In many applications, it is necessary to find the distance between two pixels or two components of an image. Unfortunately, there is no unique method of defining distance in digital images. One can define distance in many different ways. For all pixels p, q, and r, any distance metric must satisfy all of the following three properties:

1. $d(p,q) \geq 0 \quad$ and $\quad d(p,q) = 0 \quad$ iff $\quad p = q$
2. $d(q,p) = d(p,q)$
3. $d(p,r) \leq d(p,q) + d(q,r)$

Several distance functions have been used in digital geometry. Some of the more common distance functions are:

Euclidean

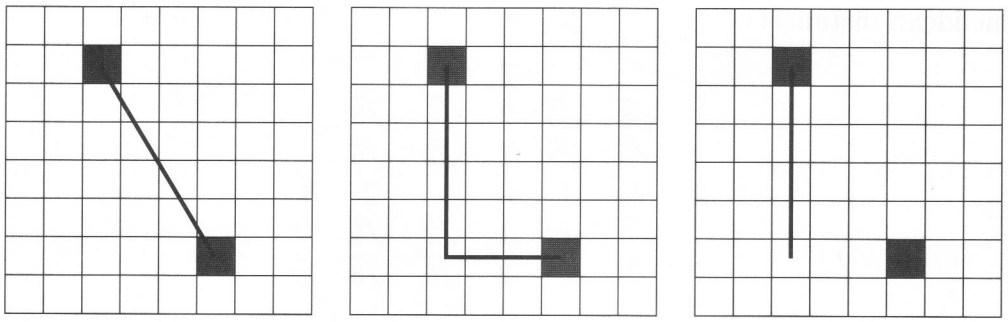
$$d_{\text{Euclidean}}([i_1, j_1], [i_2, j_2]) = \sqrt{(i_1 - i_2)^2 + (j_1 - j_2)^2} \tag{2.38}$$

City-block

$$d_{\text{city}} = |i_1 - i_2| + |j_1 - j_2| \tag{2.39}$$

Chessboard

$$d_{\text{chess}} = \max(|i_1 - i_2|, |j_1 - j_2|) \tag{2.40}$$



(a) (b) (c)

Figure 2.17: Examples of (a) Euclidean, (b) city-block, and (c) chessboard distance measures.

The city-block and chessboard distance measures are preferred over Euclidean due to their simplicity of calculation. These three functions are illustrated in Figure 2.17.

A set of pixels at distance $\leq k$ according to a distance metric is called a *disc* of radius k under that metric. Figure 2.18 shows discs of size 3 according to the above three measures. Note that the shapes of the neighborhood are significantly different. An important consideration in using a distance measure is the fact that though the Euclidean distance is closest to the continuous case, it is computationally the most expensive and results in real-valued distances. The integer-valued square of the Euclidean distance can also be used as a distance measure.

2.5.9 Distance Transforms

In certain applications, such as character recognition, the minimum distance between a pixel of an object component and the background is used. Thus, given an object region, S, we must compute the distance to the background region, $\bar{S}$, for all pixels in S. The transform for obtaining an image representing such distances is called a *distance transform*. A parallel iterative algorithm to compute the distance transform is obtained using the equations

$$f^0[i,j] = f[i,j] \tag{2.41}$$

$$f^m[i,j] = f^0[i,j] + \min\left(f^{m-1}[u,v]\right) \tag{2.42}$$

Euclidean distance:

$$\begin{array}{ccccccc} & & & 3 & & & \\ & \sqrt{8} & \sqrt{5} & 2 & \sqrt{5} & \sqrt{8} & \\ & \sqrt{5} & \sqrt{2} & 1 & \sqrt{2} & \sqrt{5} & \\ 3 & 2 & 1 & 0 & 1 & 2 & 3 \\ & \sqrt{5} & \sqrt{2} & 1 & \sqrt{2} & \sqrt{5} & \\ & \sqrt{8} & \sqrt{5} & 2 & \sqrt{5} & \sqrt{8} & \\ & & & 3 & & & \end{array}$$

City-block distance:

$$\begin{array}{ccccccc} & & & 3 & & & \\ & & 3 & 2 & 3 & & \\ & 3 & 2 & 1 & 2 & 3 & \\ 3 & 2 & 1 & 0 & 1 & 2 & 3 \\ & 3 & 2 & 1 & 2 & 3 & \\ & & 3 & 2 & 3 & & \\ & & & 3 & & & \end{array}$$

Chessboard distance:

$$\begin{array}{ccccccc} 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 3 & 2 & 2 & 2 & 2 & 2 & 3 \\ 3 & 2 & 1 & 1 & 1 & 2 & 3 \\ 3 & 2 & 1 & 0 & 1 & 2 & 3 \\ 3 & 2 & 1 & 1 & 1 & 2 & 3 \\ 3 & 2 & 2 & 2 & 2 & 2 & 3 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 \end{array}$$

Figure 2.18: Different distance measures.

$$\begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array} \rightarrow \begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array} \rightarrow \begin{array}{cccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 & 2 & 1 \\ 1 & 2 & 3 & 3 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{array}$$

Figure 2.19: Distance transform of an image after the first and second iterations.

where m is the iteration number for all pixels $[u, v]$, such that $d([u, v], [i, j]) = 1$. Note that this only uses the 4-neighbors of $[i, j]$.

This algorithm does not change pixels of $\bar{S}$. In the first iteration, all of the pixels that are not adjacent to $\bar{S}$ are changed to 2. In the succeeding iterations, pixels farther away from $\bar{S}$ change. No pixel changes after the distances to all pixels are obtained. The operation of this algorithm is shown in Figure 2.19.

2.5.10 Medial Axis

We say that the distance $d([i, j], \bar{S})$ from the pixel $[i, j]$ in S to $\bar{S}$ is locally maximum if

$$d([i, j], \bar{S}) \geq d([u, v], \bar{S}) \tag{2.43}$$

for all pixels $[u, v]$ in the neighborhood of $[i, j]$. The set of pixels in S with distances from $\bar{S}$ that are locally maximum is called the skeleton, symmetric axis, or medial axis of S and is usually denoted by S^*. Some examples of the medial axis transform when using 4-neighbors are given in Figures 2.20 and 2.21. Figure 2.22 shows that a small amount of noise in the original image can cause a significant difference in the resulting medial axis transform.

The original set S can be reconstructed from S^* and the distances of each pixel of S^* from $\bar{S}$. S^* is a compact representation of S. S^* is used to represent the shape of a region. By deleting pixels of S^* whose distances from $\bar{S}$ are small, we can create a simplified version of S^*.

The medial axis has been used for compact representation of objects. However, a region in a binary image may also be represented using its boundary. A boundary-following algorithm may be used to obtain a sequence of

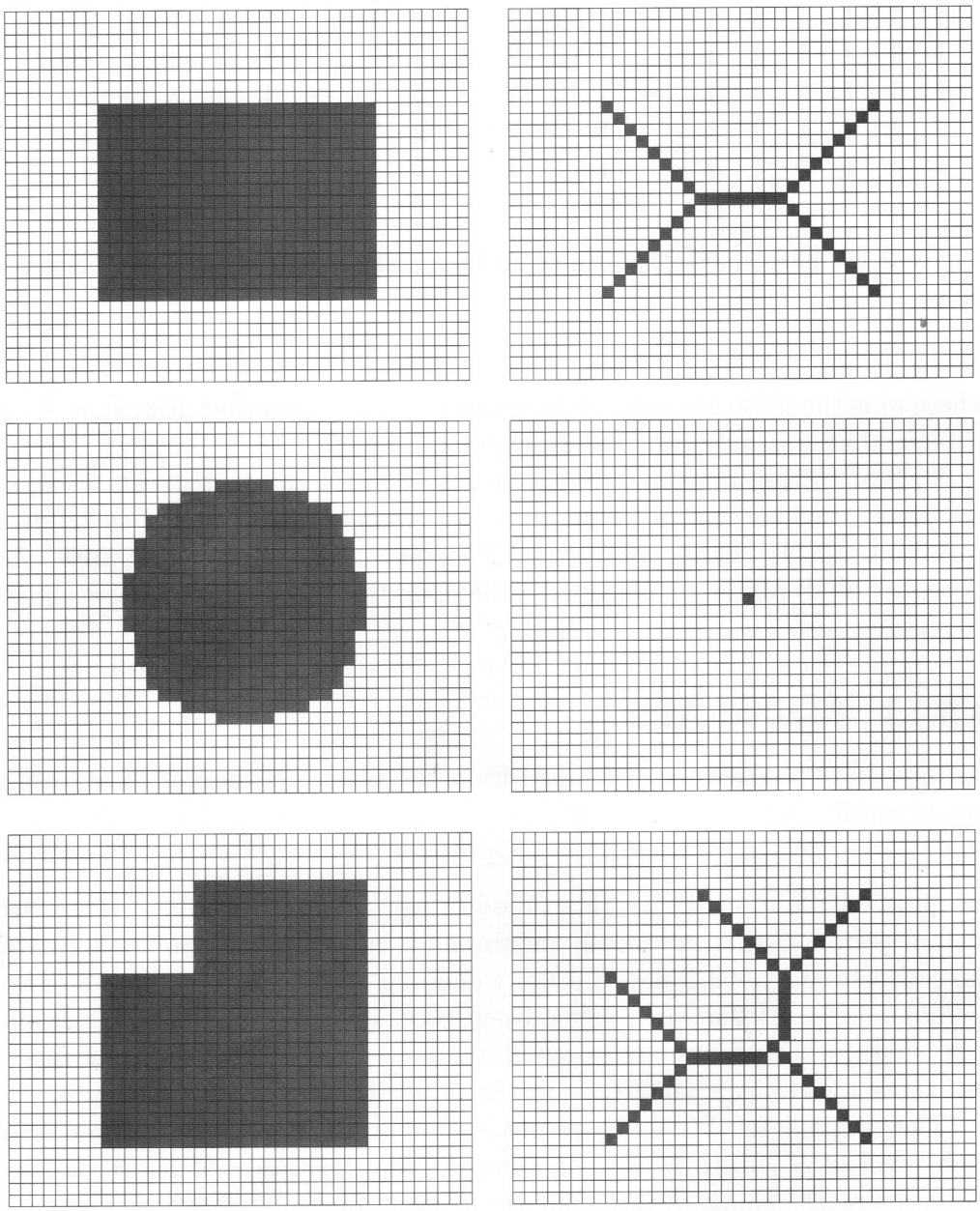


Figure 2.20: Examples of the medial axis transform.

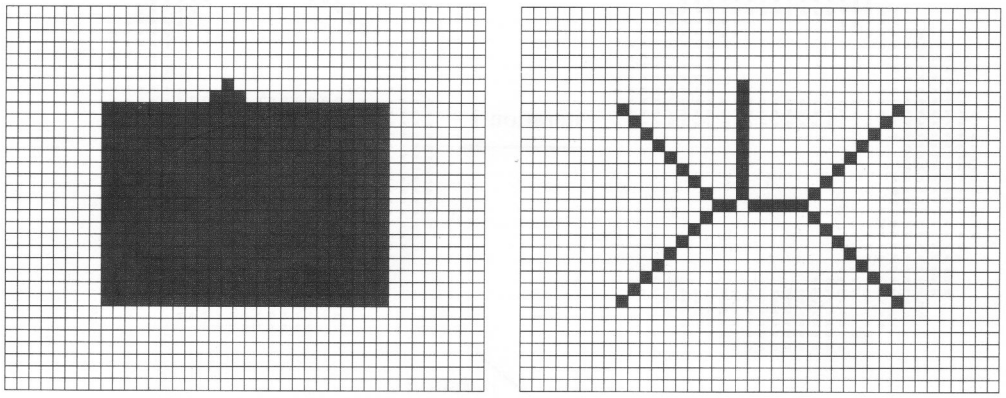


Figure 2.21: The result of the medial axis transform on a noisy image. Note that this result is very different from the first example in Figure 2.20.

pixels representing the boundary. The boundary may be very compactly represented using chain codes, to be discussed in a later section. For arbitrary objects, a boundary is a more compact representation of a region. However, to find whether a given pixel is in the region or not, the medial axis is a better representation. This is due to the fact that pixels that are within a region represented by the medial axis can be easily detected using pixels on the axis and the maximal disc at each pixel, as given by the distance transform.

2.5.11 Thinning

Thinning is an image-processing operation in which binary-valued image regions are reduced to lines that approximate their center lines, also called "skeletons" or core lines. The purpose of thinning is to reduce the image components to their essential information so that further analysis and recognition are facilitated. Although the thinning operation can be applied to binary images containing regions of any shape, it is useful primarily for "elongated" shapes as opposed to convex, or "bloblike" shapes. Thinning is commonly used in the preprocessing stage of document analysis applications for representing lines in a drawing or character strokes in a text image.

The thinning requirements are formally stated as follows:

1. Connected image regions must thin to connected line structures.

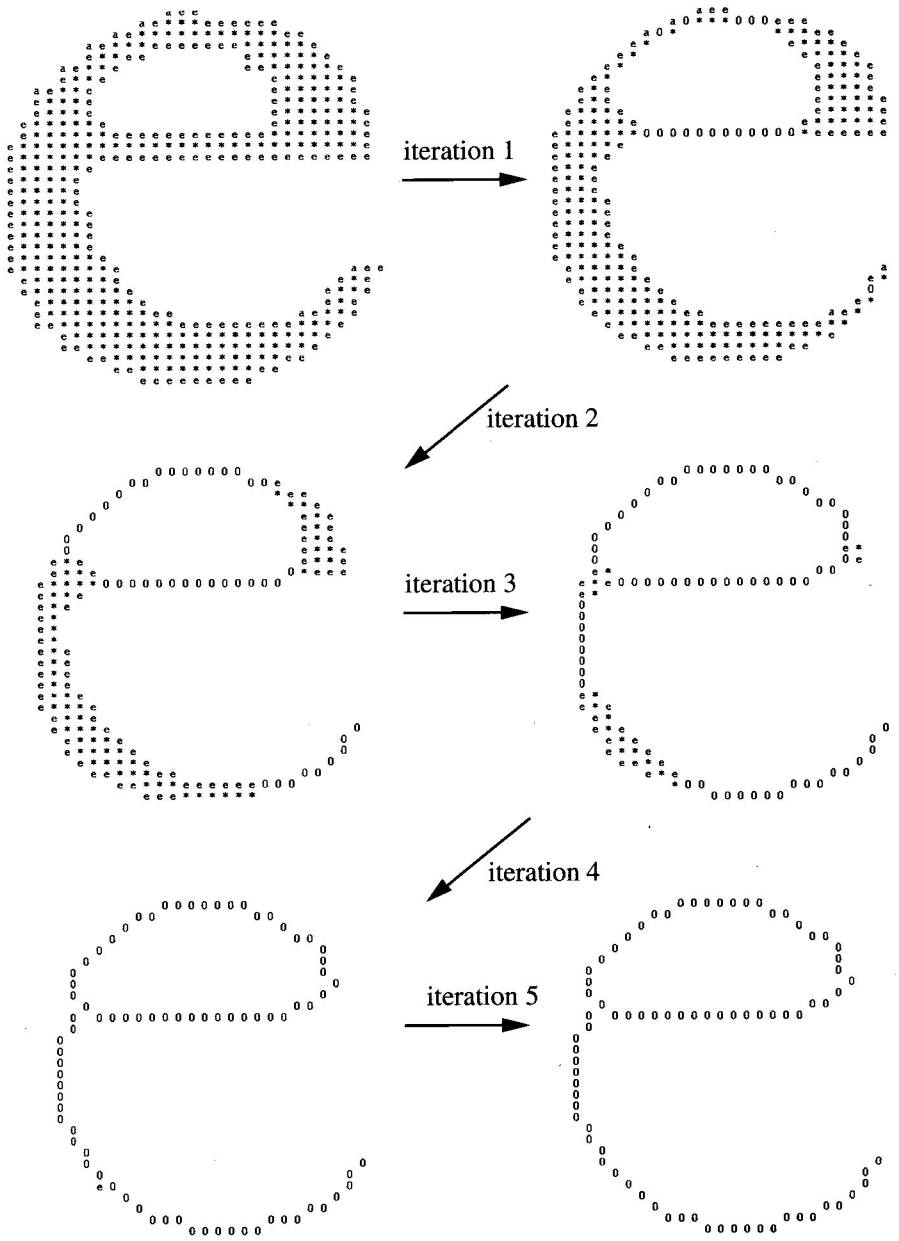


Figure 2.22: Sequence of five iterations of thinning the letter "e." On each iteration a layer of the boundary is peeled off. On iteration 5, the end is detected because no pixels change. (*Courtesy of Lawrence O'Gorman, AT&T Bell Laboratories.*)

2. The thinned result should be minimally 8-connected (explained below).

3. Approximate endline locations should be maintained.

4. The thinning results should approximate the medial lines.

5. Extraneous spurs (short branches) caused by thinning should be minimized.

That the results of thinning must maintain connectivity as specified by requirement 1 is essential. This guarantees a number of connected line structures equal to the number of connected regions in the original image. Requirement 2 stipulates that the resulting lines should always contain a minimal number of pixels that maintain 8-connectedness. Requirement 3 states that the locations of endlines should be maintained. Since thinning can be achieved by iteratively removing outer boundary pixels, it is important not to also iteratively remove the last pixels of a line. This would shorten the line and fail to preserve its location. Requirement 4 states that the resultant lines should best approximate the medial lines of the original regions. Unfortunately, in digital space, the true medial lines can only be approximated. For example, for a 2-pixel-wide vertical or horizontal line, the true medial line should run at the half-pixel spacing along the middle of the original line. Since it is impossible to represent this in digital images, the result will be a single line running at one side of the original. With respect to requirement 5, it is obvious that noise should be minimized, but it is often difficult to say what is noise and what isn't. We don't want spurs to result from every small bump on the original region, but we do want to recognize when a somewhat larger bump is a feature. Though some thinning algorithms have parameters to remove spurs, we believe that thinning and noise removal should be performed separately. Since one person's undesired spur may be another's desired short line, it is best to perform thinning first and then, in a separate process, remove any spurs whose length is less than a specified minimum.

A common thinning approach is to examine each pixel in the image within the context of its neighborhood region of 3×3 pixels and to "peel" the region boundaries, one pixel layer at a time, until the regions have been reduced to thin lines. This process is performed iteratively: on each iteration, every image pixel is inspected within 3×3 windows, and single-pixel-thick boundaries that are not required to maintain connectivity or the position of

a line end are erased. In Figure 2.22 you can see how, on each iteration, the outside layer of a 1-valued region is peeled off in this manner. When no changes are made on an iteration, the process is complete and the image is thinned.

2.5.12 Expanding and Shrinking

A component of an image can be systematically expanded or contracted. When a component is allowed to change such that some background pixels are converted to 1, the operation is called expanding. If object pixels are systematically deleted or converted to 0, then it is called shrinking. A simple implementation of expanding and shrinking may be:

Expanding: Change a pixel from 0 to 1 if any neighbors of the pixel are 1.

Shrinking: Change a pixel from 1 to 0 if any neighbors of the pixel are 0.

Thus, shrinking may be considered as expanding the background. Example of these operations are given in Figure 2.23.

It is interesting that simple operations like expanding and shrinking can be used to do some very useful and seemingly complex operations on images. Let us denote:

$S^{(k)}$: S expanded k times
$S^{(-k)}$: S shrunk k times

It can be shown that the following properties hold:

$$\begin{aligned}(S^m)^{-n} &\neq (S^{-n})^m \\ &\neq S^{(m-n)}\end{aligned}$$

$$\begin{aligned}S &\subset (S^k)^{-k} \\ S &\supset (S^{-k})^k\end{aligned}$$

Expanding followed by shrinking can be used for filling undesirable holes, and shrinking followed by expanding can be used for removing isolated noise pixels (see Figure 2.24). Expanding and shrinking can be used to determine isolated components and clusters. In morphological image processing and dilation and erosion operations, generalized forms of expanding and shrinking are used extensively to do many tasks.

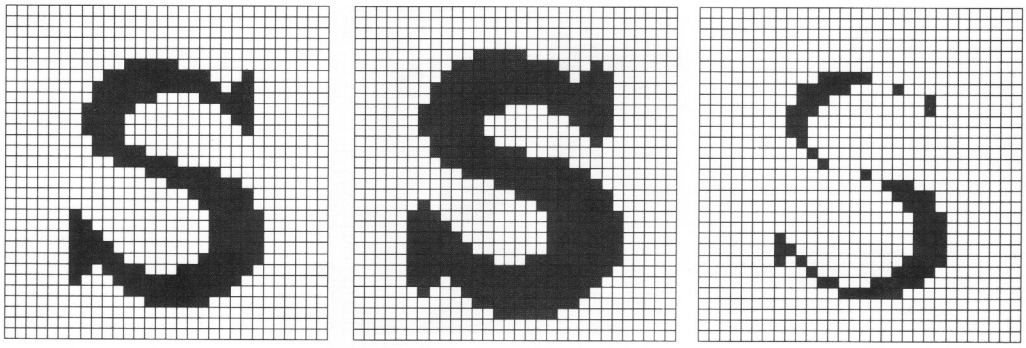


Figure 2.23: An example of expanding and shrinking operations on the letter "s." *Left*: The original image. *Middle*: Expanded image. *Right*: Shrunken image.

2.6 Morphological Operators

Mathematical morphology gets its name from the study of shape. This approach exploits the fact that in many machine vision applications, it is natural and easy to think in terms of shapes when designing algorithms. A morphological approach facilitates shape-based, or iconic, thinking. The fundamental unit of pictorial information in the morphological approach is the binary image.

The *intersection* of any two binary images A and B, written $A \cap B$, is the binary image which is 1 at all pixels p which are 1 in both A and B. Thus,

$$A \cap B = \{p | p \in A \text{ and } p \in B\}. \tag{2.44}$$

The *union* of A and B, written $A \cup B$, is the binary image which is 1 at all pixels p which are 1 in A or 1 in B (or 1 in both). Symbolically,

$$A \cup B = \{p | p \in A \text{ or } p \in B\}. \tag{2.45}$$

Let Ω be a universal binary image (all 1) and A a binary image. The *complement* of A is the binary image which interchanges the 1s and 0s in A. Thus,

$$\overline{A} = \{p | p \in \Omega \text{ and } p \notin A\}. \tag{2.46}$$

The *vector sum* of two pixels p and q with indices $[i, j]$ and $[k, l]$ is the pixel $p + q$ with indices $[i + k, j + l]$. The *vector difference* $p - q$ is the pixel with indices $[i - k, j - l]$.

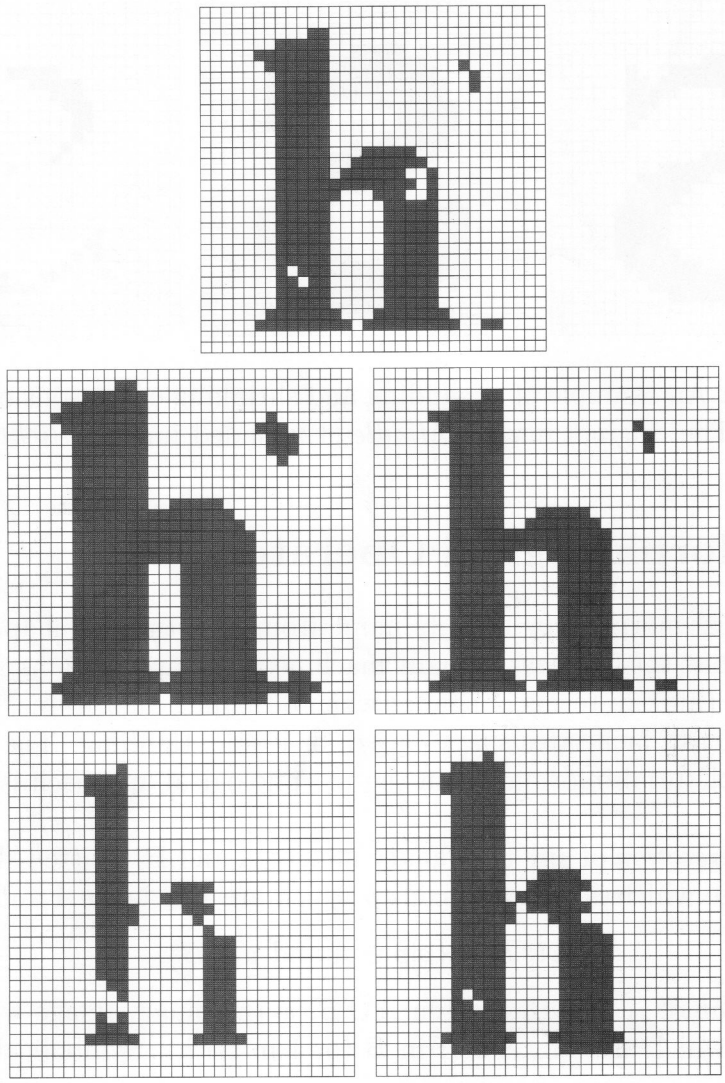


Figure 2.24: Sequences of expanding and shrinking the letter "h." *Top:* The original noisy image. *Middle:* Expanding followed by shrinking. *Bottom:* Shrinking followed by expanding. Note that expanding followed by shrinking effectively filled the holes but did not eliminate the noise. Conversely, shrinking followed by expanding eliminated the noise but did not fill the holes.

If A is a binary image and p is a pixel, then the *translation* of A by p is an image given by

$$A_p = \{a + p | a \in A\}. \tag{2.47}$$

Dilation

Translation of a binary image A by a pixel p shifts the origin of A to p. If $A_{b_1}, A_{b_2}, \ldots, A_{b_n}$ are translations of the binary image A by the 1 pixels of the binary image $B = \{b_1, b_2, \ldots, b_n\}$, then the union of the translations of A by the 1 pixels of B is called the *dilation* of A by B and is given by

$$A \oplus B = \bigcup_{b_i \in B} A_{b_i} = \bigcup_{a_i \in A} B_{a_i} \tag{2.48}$$

Dilation has both associative and commutative properties. Thus, in a sequence of dilation steps the order of performing operations is not important. This fact allows breaking a complex shape into several simpler shapes which can be recombined as a sequence of dilations.

Erosion

The opposite of dilation is *erosion*. The erosion of a binary image A by a binary image B is 1 at a pixel p if and only if every 1 pixel in the translation of B to p is also 1 in A. Erosion is given by

$$A \ominus B = \{p | B_p \subseteq A\}. \tag{2.49}$$

Often the binary image B is a regular shape which is used as a probe on image A and is referred to as a *structuring element*. Erosion plays a very important role in many applications. Erosion of an image by a structuring element results in an image that gives all locations where the structuring element is contained in the image.

Figures 2.25 through 2.28 illustrate the dilation and erosion operations with a simple binary object and an upside-down "T-shaped" structuring element. Figure 2.26 shows examples of translations of the structuring element to 1 pixels of the original figure where the entire structuring element does *not* fit entirely inside the original object. In this case, during a dilation, every pixel in the structuring element will be present in the final dilated image, including the pixel not contained in the original object (shown as a lightly

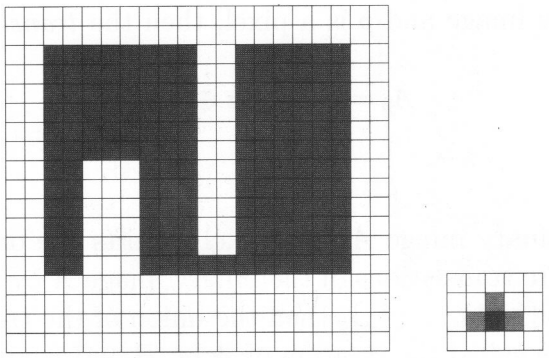


Figure 2.25: The original test image A (left) and structuring element B (right). Note that the origin of the structuring element is darker than the other pixels in B.

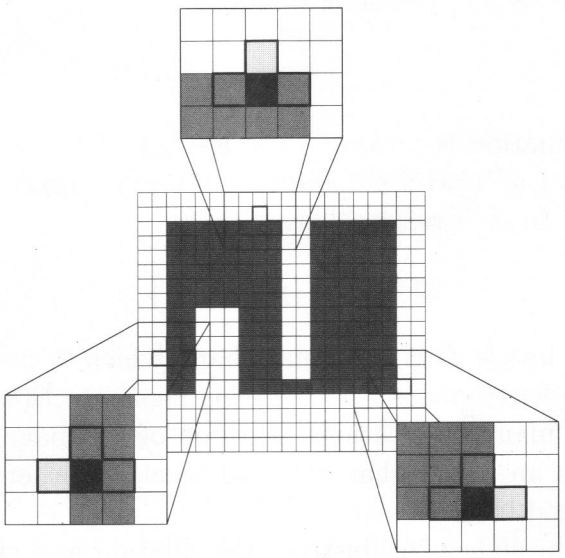
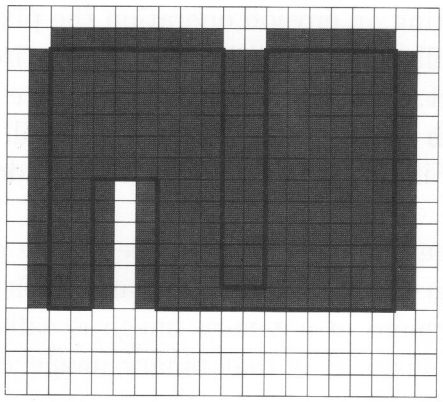
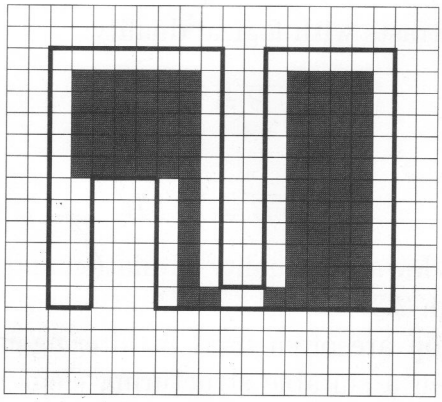


Figure 2.26: Translations of the structuring element B to 1 pixels in A where the entire structuring element is *not* contained within A. During a dilation operation, every pixel in the structuring element will be present in the final image. During an erosion operation, the pixel at the origin of the structuring element will be deleted.



$$A \oplus B = \bigcup_{b_i \in B} A_{b_i}$$

Figure 2.27: The dilation of A by B. The boundary of the original figure A is shown as a bold line.



$$A \ominus B = \{p | B_p \subseteq A\}$$

Figure 2.28: The erosion of A by B. The boundary of the original figure A is shown as a bold line.

shaded pixel). But during an erosion operation the pixel at the origin of the structuring element will be removed because the entire structuring element is not within the object. Conversely, in the case where the entire structuring element *does* fit within the original object, there will be no change to the final dilated or eroded image (i.e., no pixels will be added or deleted at that point).

Dilation and erosion exhibit a dual nature that is geometric rather than logical and involves a geometric complement as well as a logical complement. The geometric complement of a binary image is called its *reflection*. The reflection of a binary image B is that binary image B' which is symmetric with B about the origin, that is

$$B' = \{-p | p \in B\}. \tag{2.50}$$

The geometric duality of dilation and erosion is expressed by the relationships

$$\overline{A \oplus B} = \overline{A} \ominus B' \tag{2.51}$$

and

$$\overline{A \ominus B} = \overline{A} \oplus B'. \tag{2.52}$$

Geometric duality contrasts with logical duality:

$$\overline{A \cup B} = \overline{A} \cap \overline{B} \tag{2.53}$$

and

$$\overline{A \cap B} = \overline{A} \cup \overline{B}, \tag{2.54}$$

also called deMorgan's law. The duality of dilation and erosion are illustrated in Figures 2.29 through 2.31.

Erosion and dilation are often used in filtering images. If the nature of noise is known, then a suitable structuring element can be used and a sequence of erosion and dilation operations can be applied for removing the noise. Such filters affect the shape of the objects in the image.

The basic operations of mathematical morphology can be combined into complex sequences. For example, an erosion followed by a dilation with the same structuring element (probe) will remove all of the pixels in regions which are too small to contain the probe, and it will leave the rest. This sequence

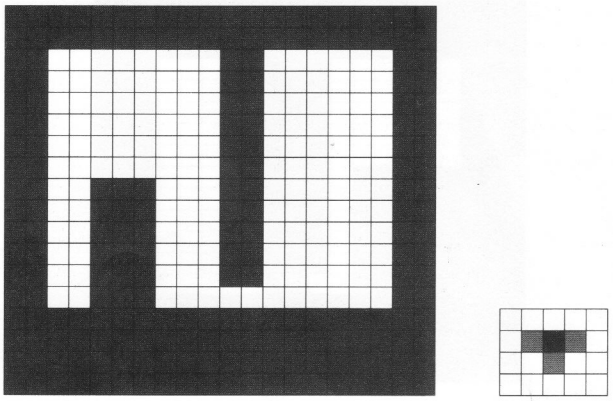
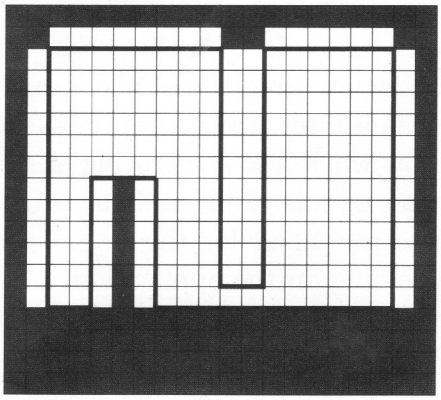
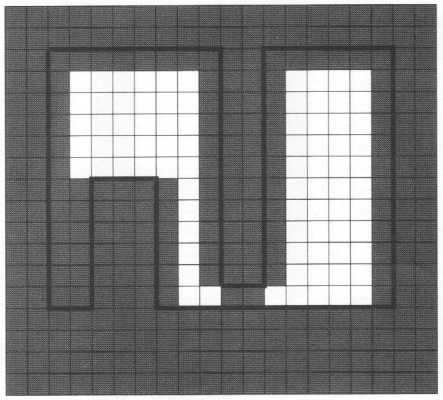


Figure 2.29: The complement of A and the reflection of B. Note that the origin of the reflected structuring element is still the darker pixel.



$\overline{A} \ominus B'$

Figure 2.30: The dual of dilation: the result of eroding the background of A with the reflection of B. The original boundary is shown as a bold line.



$\overline{A} \oplus B'$

Figure 2.31: The dual of erosion: the result of dilating the background of A with the reflection of B. The original boundary is shown as a bold line.

is called *opening*. As an example, if a disc-shaped probe image is used, then all of the convex or isolated regions of pixels smaller than the disc will be eliminated. This forms a filter which suppresses positive spatial details. The remaining pixels show where the structuring element is contained in the foreground. The difference of this result and the original image would show those regions which were too small for the probe, and these could be the features of interest, depending on the application.

The opposite sequence, a dilation followed by an erosion, will fill in holes and concavities smaller than the probe. This is referred to as *closing*. These operations are illustrated in Figures 2.32 and 2.33 with the same T-shaped structuring element. Again, what is removed may be just as important as what remains. Such filters can be used to suppress spatial features or discriminate against objects based upon their size. The structuring element used does not have to be compact or regular, and can be any pattern of pixels. In this way, features made up of distributed pixels can be detected.

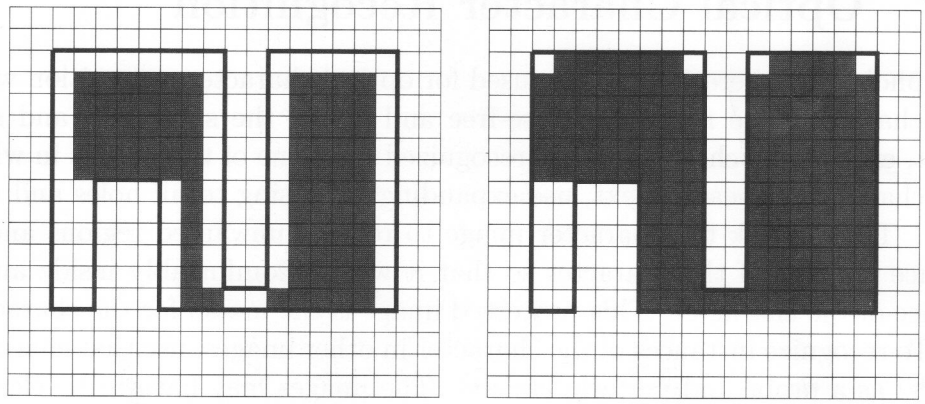


Figure 2.32: Opening operation. *Left:* Initial erosion. *Right:* Succeeding dilation. The boundary of the original figure A is shown as a bold line.

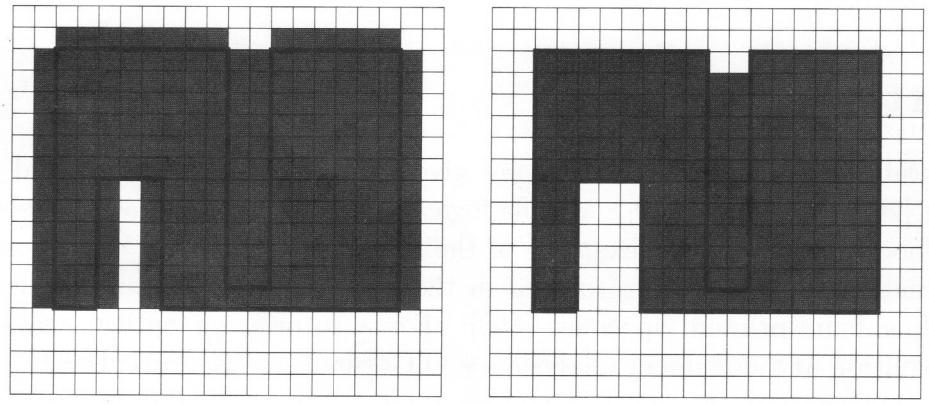


Figure 2.33: Closing operation. *Left:* Initial dilation. *Right:* Succeeding erosion. The boundary of the original figure A is shown as a bold line.

2.7 Optical Character Recognition

Morphological operations can be used for optical character recognition when the characters are relatively noise-free and are of the same font and size. First, extract the character to be recognized from one of the images in which the character appears. Next, use expanding or closing to fill holes and cavities. Then shrink the character image to remove unwanted regions and to reduce the size of the character so that it will fit comfortably inside an instance of the character. This processed image is the model for the character.

To recognize instances of the character in other images, use that character model as a probe and perform erosion. The images may have to be cleaned (holes filled and unwanted clutter removed) before performing erosion. After erosion, compute connected components, apply the size filter to discard regions that are too small, and compute the position of each region that passes through the size filter. This provides the position of each recognized instance of the character model in the image. Good character models obtained after cleaning, filling, and shrinking will match most instances of the character, including instances in a slightly different font and size. However, omnifont recognition has been a very challenging problem for researchers. Optical character recognition can be implemented in real time with special-purpose hardware.

Further Reading

Several aspects of binary vision are given in the books by Rosenfeld and Kak [206] and Horn [109]. Morphological image processing is discussed in the books [68, 103]. An example of the use of morphological operations in a practical application is provided in the paper by Mitchell and Gillies on reading hand-printed zip codes [168]. For a tutorial on optical character recognition and document analysis see O'Gorman and Kasturi [188].

Exercises

2.1 We saw in this chapter that the zeroth-, first-, and second-order moments for a binary region provide important information about the image's size, location, and orientation. Define higher moments of a

region. Do you think that these moments will provide any useful characteristics for the region? If so, what will be the nature of information provided by these higher moments?

2.2 In many applications an early step is to determine where in an image are interesting objects. After these objects are located, most processing is focused only on these areas. How can you use projections for such focus-of-attention computations?

2.3 A component labeling algorithm is a computation bottleneck in many applications. This can be considered a bridge between lower levels and higher (semantic) levels in a vision system. How can you develop a fast algorithm to compute connected components? Can you develop a parallel algorithm?

Computer Projects

2.1 Use a thresholding program (or write your own) and see an image at many thresholding levels. Find a suitable threshold to get the best representation of an object in the image. Now select another object and find the best threshold for it. Repeat this for each object. Are these thresholds the same? Why? Repeat this experiment with several images.

2.2 Develop algorithms to compute area and first and second moments of a region from its run-length code.

2.3 Develop a medial axis algorithm. Apply it to several binary images of irregularly shaped objects to study the strengths and weaknesses of this technique to represent shapes of objects.

2.4 In many robotic applications, for a given domain a medial axis can be used for path planning. Consider the map of a building floor. Find the medial axis for the hallways and see which hallways can be navigated by a robot of specific size.

2.5 Construct algorithms to implement expanding and shrinking operations. Use these algorithms to implement different types of noise removal in binary images.

2.6 Modify your shrink algorithm to make it intelligent so that it does not completely eliminate a region. This feature can then be used to compute the number of connected components in a region. Implement this algorithm.

2.7 Design a machine vision system to identify objects from their binary images. Consider common objects such as coins, pens, notebooks, and other desk accessories. Develop a recognition strategy based on the features that you studied in this chapter. Implement all operations and test your system.