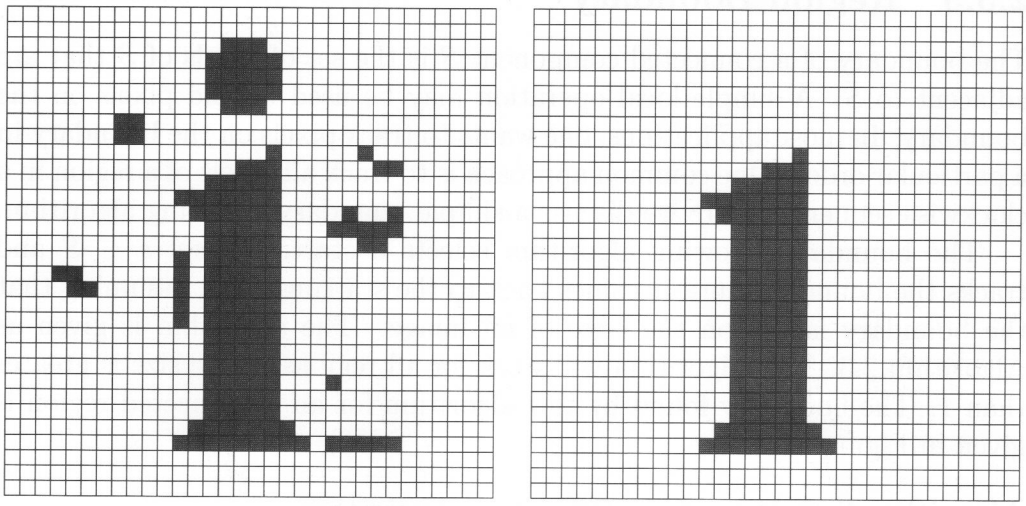
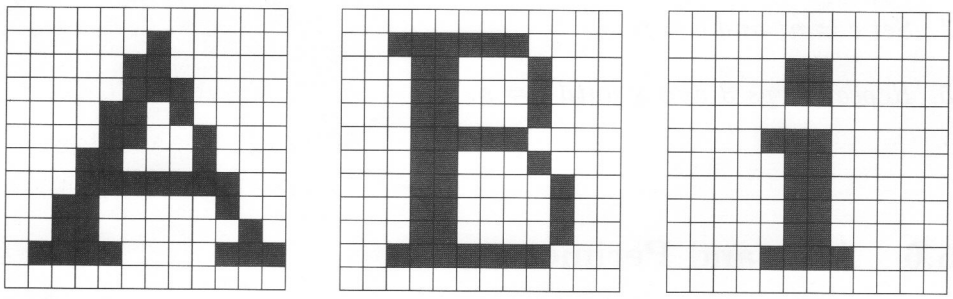


Figure 2.14: The possible error resulting from a poor choice of threshold for the size filter ($T = 25$). Note that the "dot" is missing from the letter "i."



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

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.

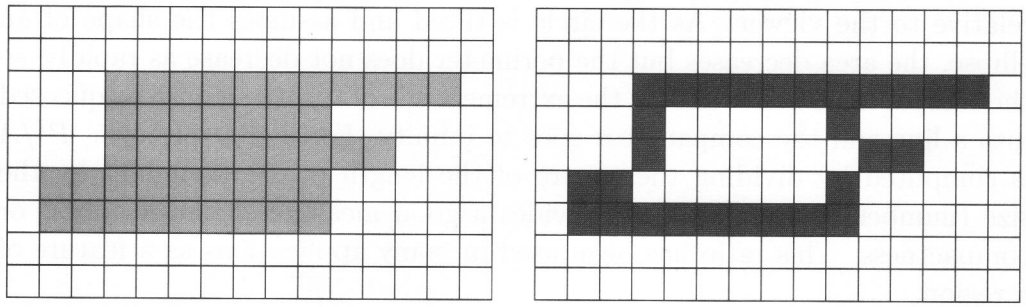


Figure 2.16: Results of a boundary-following algorithm. *Left:* Original binary object. *Right:* Calculated boundary.

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

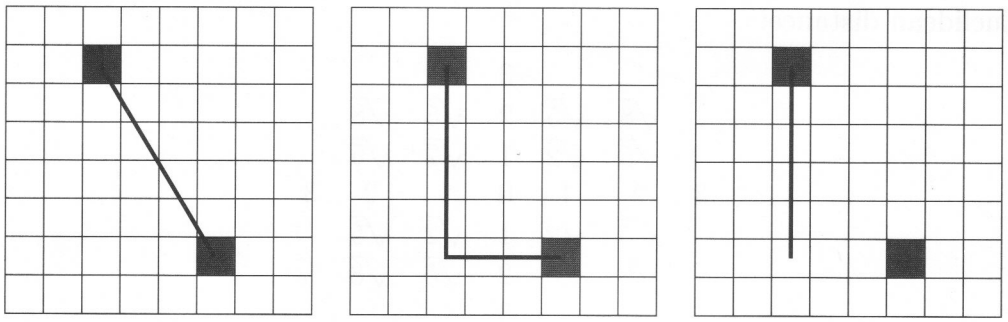
$$d_{\text{Euclidean}}([i_1, j_1], [i_2, j_2]) = \sqrt{(i_1 - i_2)^2 + (j_1 - j_2)^2} \tag{2.38}$$

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

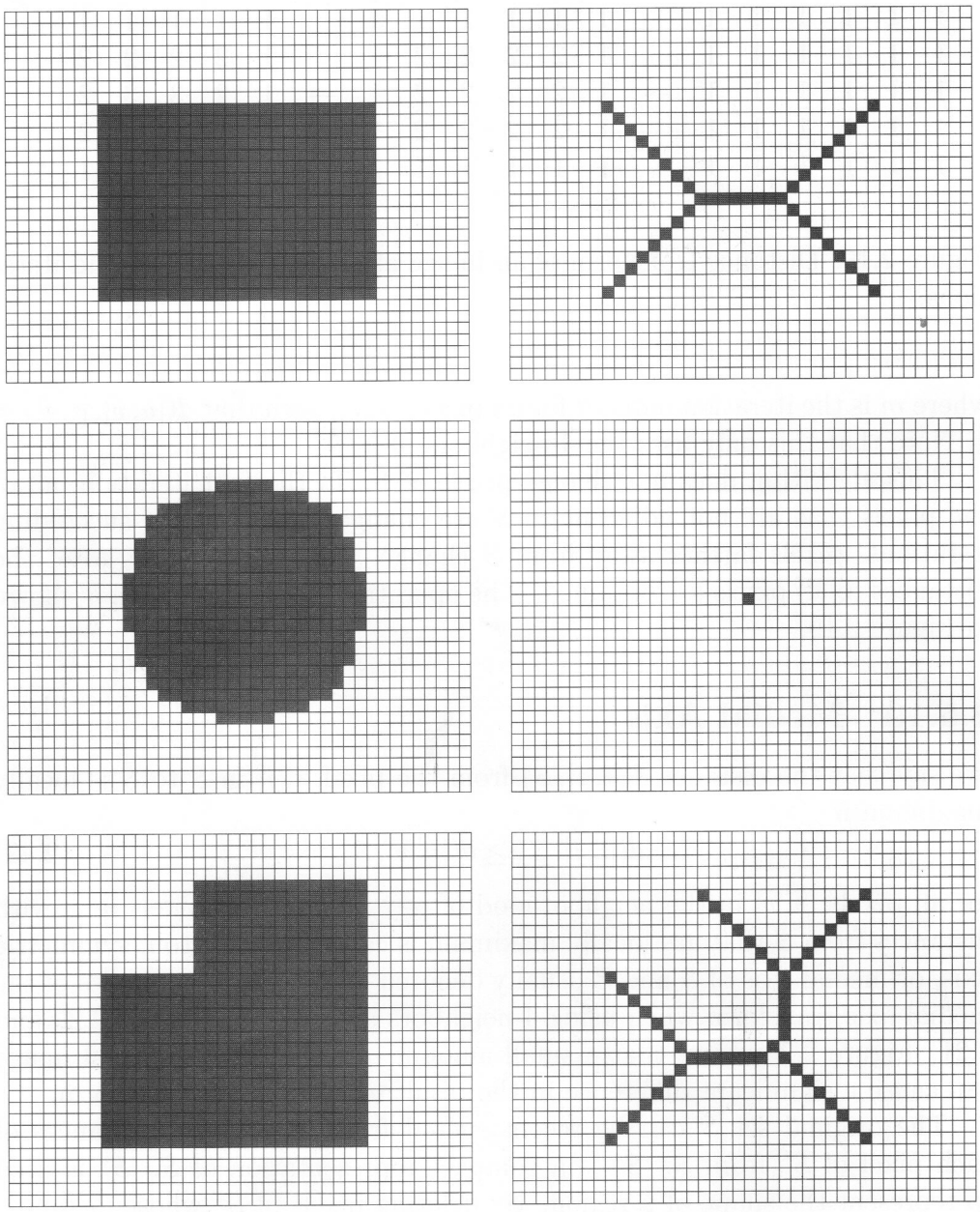


Figure 2.20: Examples of the medial axis transform.

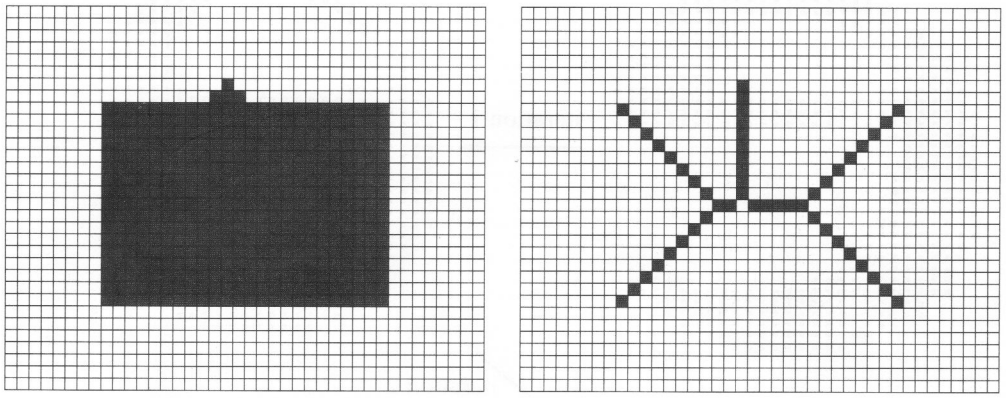


Figure 2.21: The result of the medial axis transform on a noisy image. Note that this result is very different from the first example in Figure 2.20.

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.

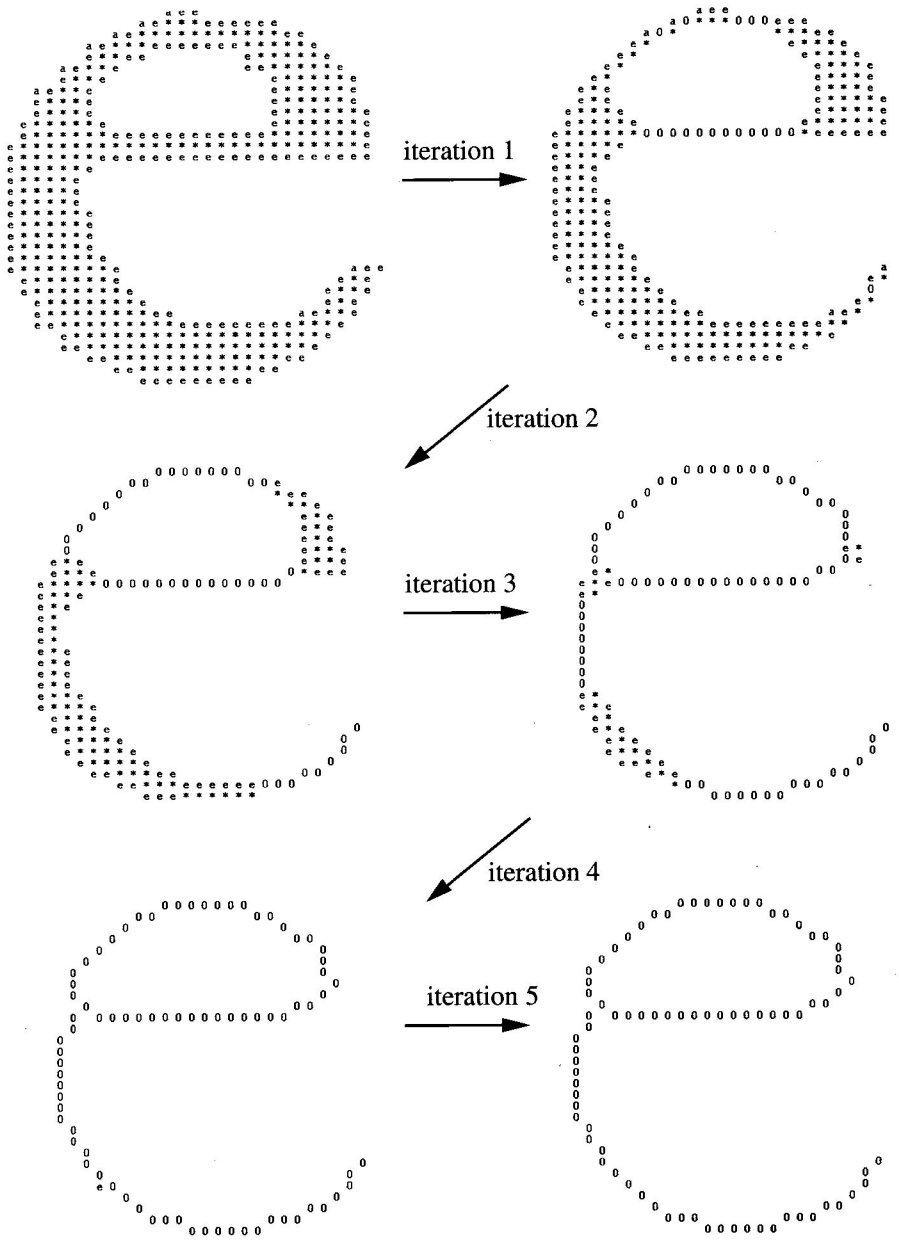


Figure 2.22: Sequence of five iterations of thinning the letter "e." On each iteration a layer of the boundary is peeled off. On iteration 5, the end is detected because no pixels change. (*Courtesy of Lawrence O'Gorman, AT&T Bell Laboratories.*)

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.

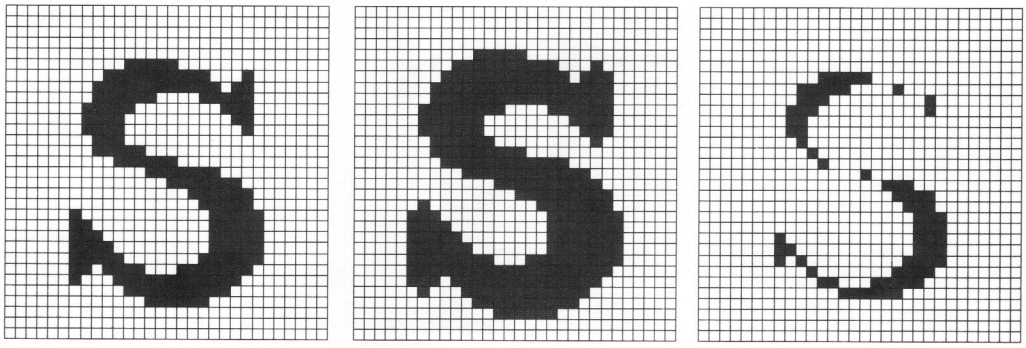


Figure 2.23: An example of expanding and shrinking operations on the letter "s." *Left*: The original image. *Middle*: Expanded image. *Right*: Shrunken image.

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 $[i - k, j - l]$.

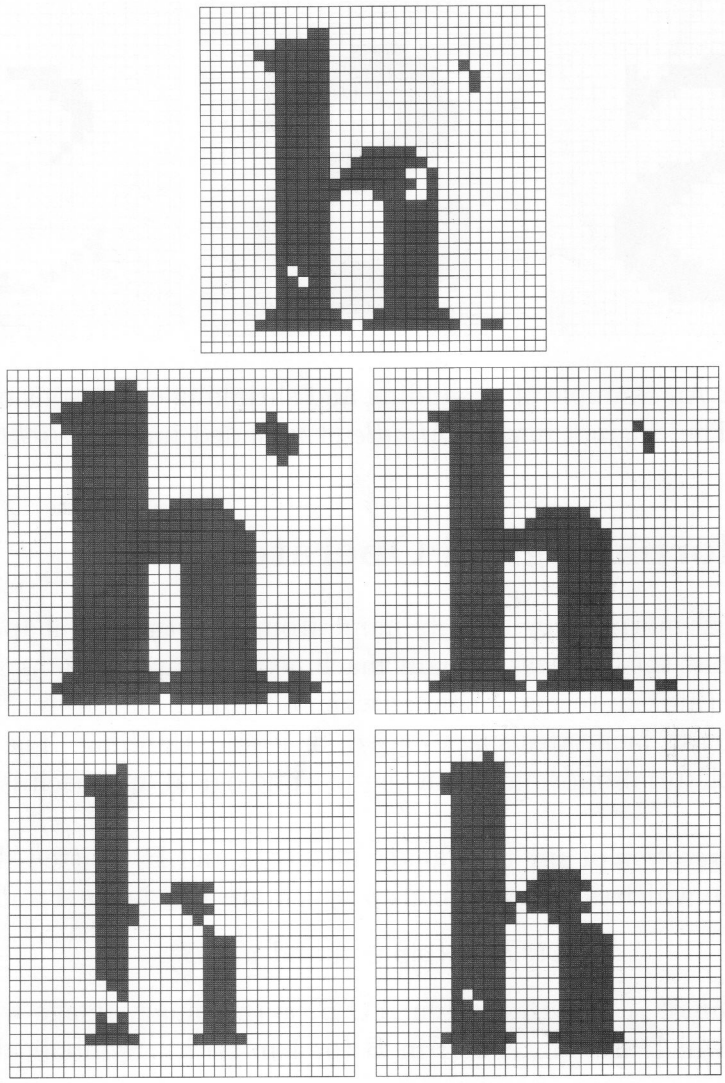


Figure 2.24: Sequences of expanding and shrinking the letter "h." *Top:* The original noisy image. *Middle:* Expanding followed by shrinking. *Bottom:* Shrinking followed by expanding. Note that expanding followed by shrinking effectively filled the holes but did not eliminate the noise. Conversely,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

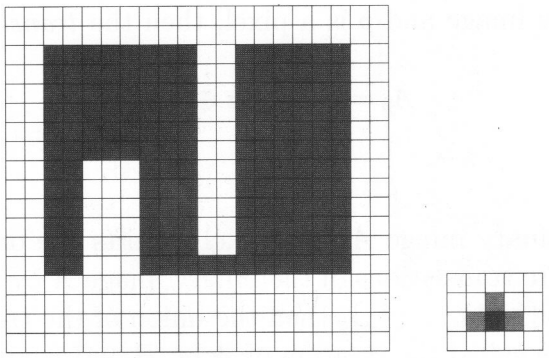


Figure 2.25: The original test image A (left) and structuring element B (right). Note that the origin of the structuring element is darker than the other pixels in B.

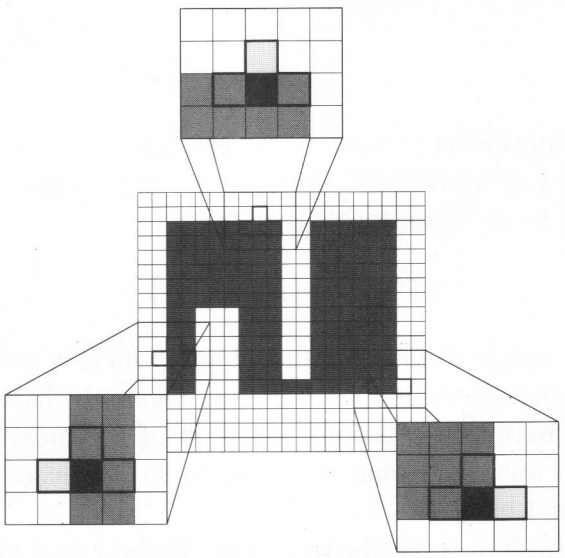
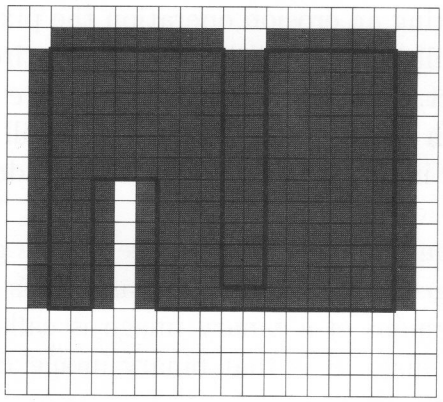
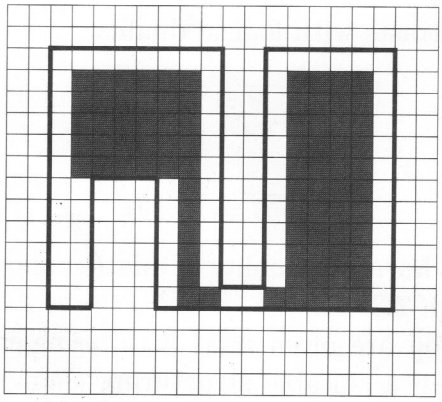


Figure 2.26: Translations of the structuring element B to 1 pixels in A where the entire structuring element is *not* contained within A. During a dilation operation, every pixel in the structuring element will be present in the final image. During an erosion operation, the pixel at the origin of the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

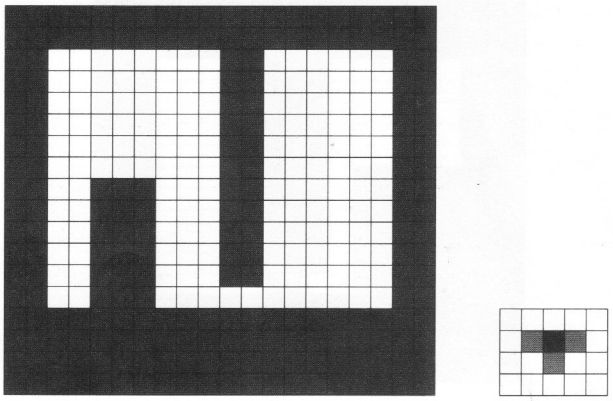
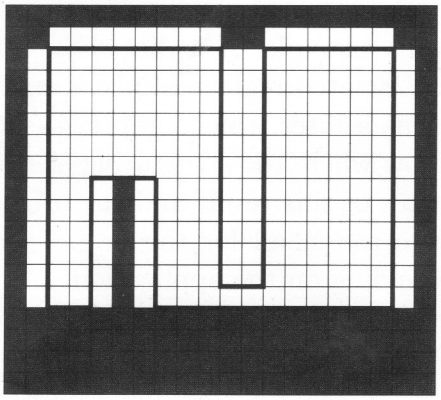
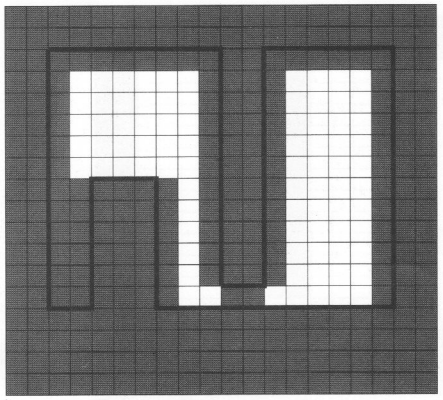


Figure 2.29: The complement of A and the reflection of B. Note that the origin of the reflected structuring element is still the darker pixel.



$\overline{A} \ominus B'$

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.

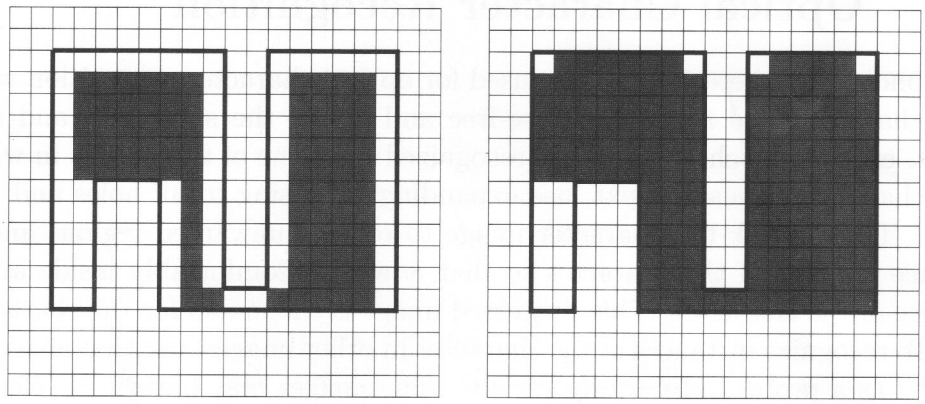


Figure 2.32: Opening operation. *Left:* Initial erosion. *Right:* Succeeding dilation. The boundary of the original figure A is shown as a bold line.

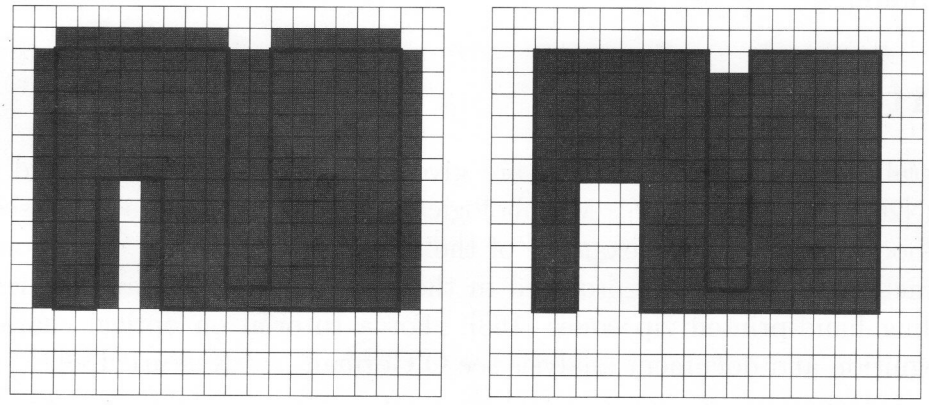


Figure 2.33: Closing operation. *Left:* Initial dilation. *Right:* Succeeding erosion. The boundary of the original figure A is shown as a bold line.

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.