2

THE USE OF DIGITAL MORPHOLOGY

2.1 MORPHOLOGY DEFINED

To be completely precise, the word morphology means "the form and structure of an object," or the arrangements and interrelationships between the parts of an object. Morphology is related to shape, and digital morphology is a way to describe or analyze the shape of a digital (most often raster) object.

The oldest uses of the word relate to language and to biology. In linguistics, morphology is the study of the structure of words, and this has been an area of study for a great many years. In biology, morphology relates more directly to the shape of an organism—the shape of a leaf can be used to identify a plant, and the shape of a colony of bacteria can be used to identify its variety. In each case, there is an intricate scheme for classification based on overall shape (elliptical, circular, etc.), type and degree of irregularities (convex, rough or smooth outline, etc.), and internal structures (holes, linear or curved features, etc.) that has been accumulated over many years of observation.

The science of digital morphology is relatively recent, since it is only recently that digital computers have made it practical. On the other hand, the mathematics behind it is simply set theory, which is a well studied area. The idea underlying digital morphology is that images consist of a set of picture elements (pixels) that collect into groups having a two-dimensional structure (shape). Certain mathematical operations on the set of pixels can be used to enhance specific aspects of the shapes so that they might be (for example) counted or recognized. Basic operations are *erosion*, in which pixels matching a given pattern are deleted from the image, and *dilation*, in which a small area about a pixel is set to a given

gical Operations

Morphological operations are methods for processing binary images based on shapes. These operations take a binary image as input, and return a binary image as output. The value of each pixel in the output image is based on the corresponding input pixel and its neighbors. By choosing the neighborhood shape appropriately, you can construct a morphological operation that is sensitive to specific shapes in the input image.

Dilation and Erosion

The main morphological operations are *dilation* and *erosion*. Dilation and erosion are related operations, although they produce very different results. Dilation adds pixels to the boundaries of objects (i.e., changes them from `off` to `on`), while erosion removes pixels on object boundaries (changes them from `on` to `off`).

Each dilation or erosion operation uses a specified neighborhood. The state of any given pixel in the output image is determined by applying a rule to the neighborhood of the corresponding pixel in the input image. The rule used defines the operation as a dilation or an erosion:

- For dilation, if *any* pixel in the input pixel's neighborhood is `on`, the output pixel is `on`. Otherwise, the output pixel is `off`.
- For erosion, if *every* pixel in the input pixel's neighborhood is `on`, the output pixel is `on`. Otherwise, the output pixel is `off`.

The neighborhood for a dilation or erosion operation can be of arbitrary shape and size. The neighborhood is represented by a *structuring element*, which is a matrix consisting of only 0's and 1's. The *center pixel* in the structuring element represents the pixel of interest, while the elements in the matrix that are `on` (i.e., = 1) define the neighborhood.

The center pixel is defined as `floor((size(SE)+1)/2)`, where `SE` is the structuring element. For example, in a 4-by-7 structuring element, the center pixel is (2,4). When you construct the structuring element, you should make sure that the pixel of interest is actually the center pixel. You can do this by adding rows or columns of 0's, if necessary. For example, suppose you want the neighborhood to consist of a 3-by-3 block of pixels, with the pixel of interest in the upper-left corner of the block. The structuring element would not be

ELEMENTS OF DIGITAL MORPHOLOGY—BINARY OPERATIONS

Binary morphological operations are defined on bilevel images; that is, images that consist of either black or white pixels only. For the purpose of beginning the discussion, consider the image seen in Figure 2.1a. The set of black pixels form a square object. The object in Figure 2.1b is also square, but is one pixel larger in all directions. It was obtained from the previous square by simply setting all white neighbors of any black pixel to black. This amounts to a simple *binary dilation*, so named because it causes the original object to grow larger. Figure 2.1c shows the result of dilating Figure 2.1b by one pixel, which is the same as dilating Figure 2.1a by two pixels; this process could be continued until the entire image consisted entirely of black pixels, at which point the image would stop showing any change.

This is a very simple example of digital morphology, and one that can be implemented directly by first marking all white pixels having at least one black neighbor, and then setting all of the marked pixels to black. This is not, however, how morphological operators are usually implemented. In general the object is considered to be a mathematical set of black pixels; since each pixel is identified by its row and column indices, a pixel is said to be a point in two-dimensional space (E^2). The set of pixels comprising the object in Figure 2.1a can now be written as {(3,3)(3,4)(4,3)(4,4)} if the upper left pixel in the image has the index (0,0). This set is too awkward to write out in full all of the time, so it will simply be called A.

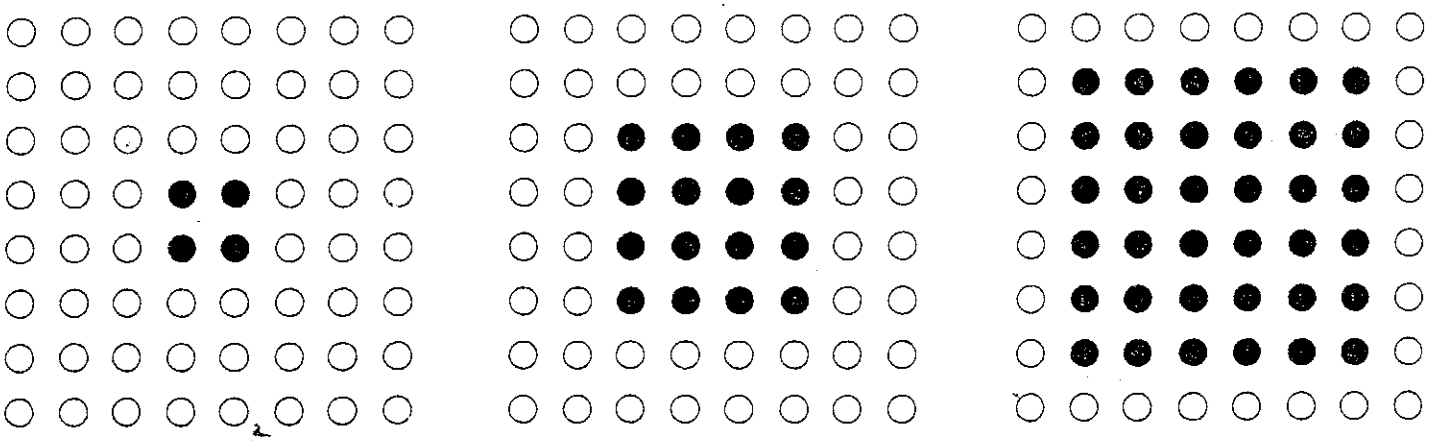


Figure 2.1 The effects of a simple binary dilation on a small object. (a) Original image. (b) Dilation of the original by 1 pixel. (c) Dilation of the original by 2 pixels (dilation of (b) by 1 pixel).

{(0,0)(0,1)}. The pixels in the set C = A + B are computed using Equation 2.7, which can be rewritten in this case as:

$$A \oplus B = (A + \{(0,0)\}) \cup (A + \{(0,1)\}) \qquad \text{(EQ 2.8)}$$

There are four pixels in the set A, and since any pixel translated by (0,0) does not change, those four will also be in the resulting set C after computing C = A + {(0,0)}:

$$(3,3) + (0,0) = (3,3) \quad (3,4) + (0,0) = (3,4)$$

$$(4,3) + (0,0) = (4,3) \quad (4,4) + (0,0) = (4,3)$$

The result of A + {(0,1)} is:

$$(3,3) + (0,1) = (3,4) \quad (3,4) + (0,1) = (3,5)$$

$$(4,3) + (0,1) = (4,4) \quad (4,4) + (0,1) = (4,5)$$

The set C is the result of the dilation of A using structuring element B, and consists of all of the pixels listed above (some of which are duplicates). Figure 2.2 illustrates this operation, showing graphically the effect of the dilation. The

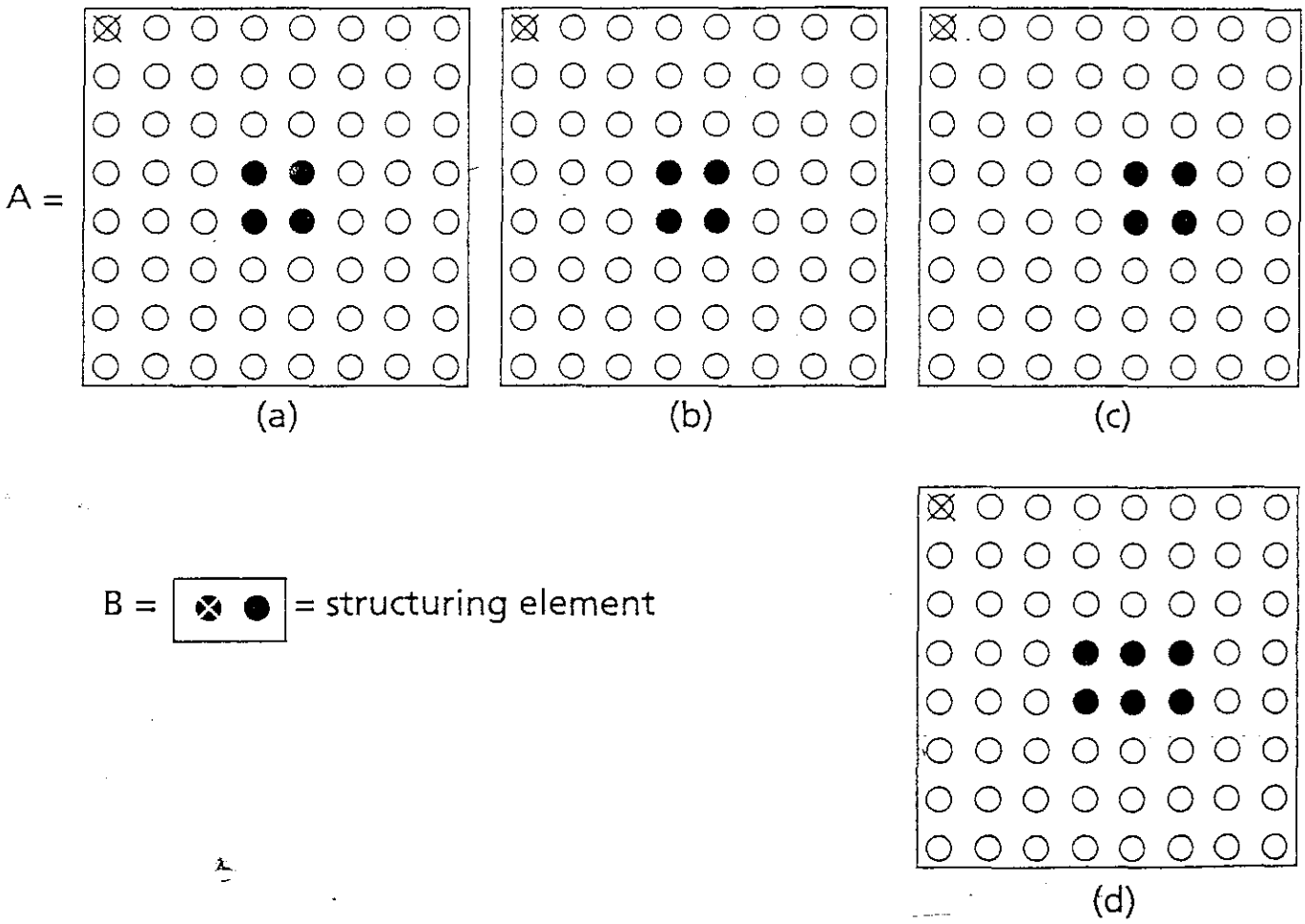


Figure 2.2 Dilation of the set A (Figure 2.1a) by the set B. (a) The two sets. (b) The set obtained by adding (0,0) to all element of A. (c) The set obtained by adding (0,1) to all elements of A. (d) The union of the two sets is the result of the dilation.

Binary Dilation

Now some definitions of simple set operations can be stated, with the goal being to define dilation in a more general fashion in terms of sets. The *translation* of the set A by the point x is defined, in set notation, as:

$$(A)_x = \{c|c = a + x, a \in A\} \quad \text{(EQ 2.1)}$$

For example, if x were at (1,2) then the first (upper left) pixel in A_x would be (3,3) + (1,2) = (4,5); all of the pixels in A shift down by one row and right by two columns in this case. This is a translation in the same sense that is seen in computer graphics—a change in position by a specified amount.

The *reflection* of the set A is defined as:

$$\hat{A} = \{c|c = -a, a \in A\} \quad \text{(EQ 2.2)}$$

This is really a rotation of the object A by 180 degrees about the origin. The *complement* of the set A is the set of pixels not belonging to A. This would correspond to the white pixels in the figure, or in the language of set theory:

$$A^c = \{c|c \notin A\} \quad \text{(EQ 2.3)}$$

The *intersection* of two sets A and B is the set of elements (pixels) belonging to both A and B:

$$A \cap B = \{c|((c \in A) \wedge (c \in B))\} \quad \text{(EQ 2.4)}$$

The *union* of two sets A and B is the set of pixels that belong to either A or B, or to both:

$$A \cup G = \{c|(c \in A) \vee (c \in B)\} \quad \text{(EQ 2.5)}$$

Finally, completing this collection of basic definitions, the *difference* between the set A and the set B is:

$$A - B = \{c|(c \in A) \wedge (c \notin B)\} \quad \text{(EQ 2.6)}$$

which is the set of pixels that belong to A but not also to B. This is really just the intersection of A with the complement of B or $A \cap B^c$.

It is now possible to define more formally what is meant by a dilation. A dilation of the set A by the set B is:

$$A \oplus B = \{c|c = a + b, a \in A, b \in B\} \quad \text{(EQ 2.7)}$$

where A represents the image being operated on, and B is a second set of pixels, a shape that operates on the pixels of A to produce the result; the set B is called a *structuring element*, and its composition defines the nature of the specific dilation. To explore this idea, let A be the set of Figure 2.1a, and let B be the set

As a further example, consider the object and structuring element shown Figure 2.3. In this case, the origin of the structuring element B_1 contains a whi pixel, implying that *the origin is not included* in the set B_1. There is no rule again this, but it is more difficult to see what will happen, so the example will be doi in detail. The image to be dilated, A_1, has the following set representation:

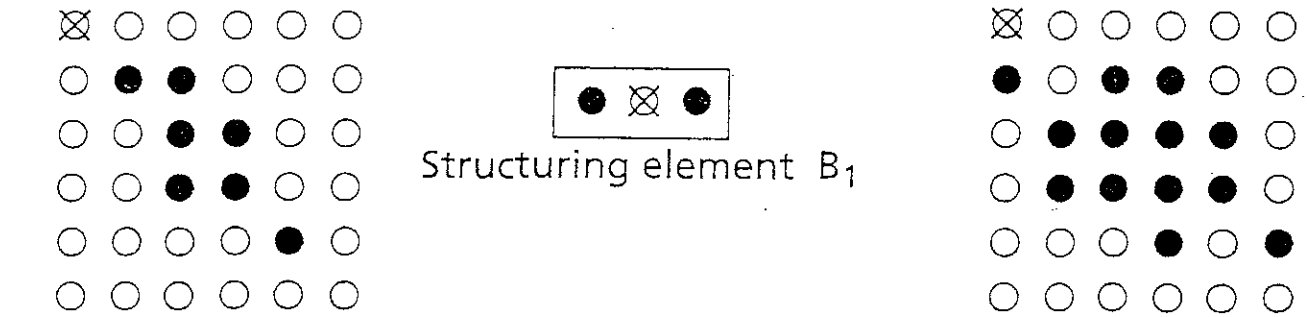
$$A_1 = \{(1,1)(2,2)(2,3)(3,2)(3,3)(4,4)\}.$$

The structuring element B_1 is:

$$B_1 = \{(0,-1)(0,1)\}.$$

The translation of A_1 by $(0,-1)$ yields

$$(A_1)_{(0,-1)} = \{(1,0)(2,1)(2,2)(3,1)(3,2)(4,3)\}$$



The object image A_1. A_1 dilated B_1.

Figure 2.3 Dilation by a structuring element that does not include the origin. Some pixels that are set in the original image are not set in the dilated image.

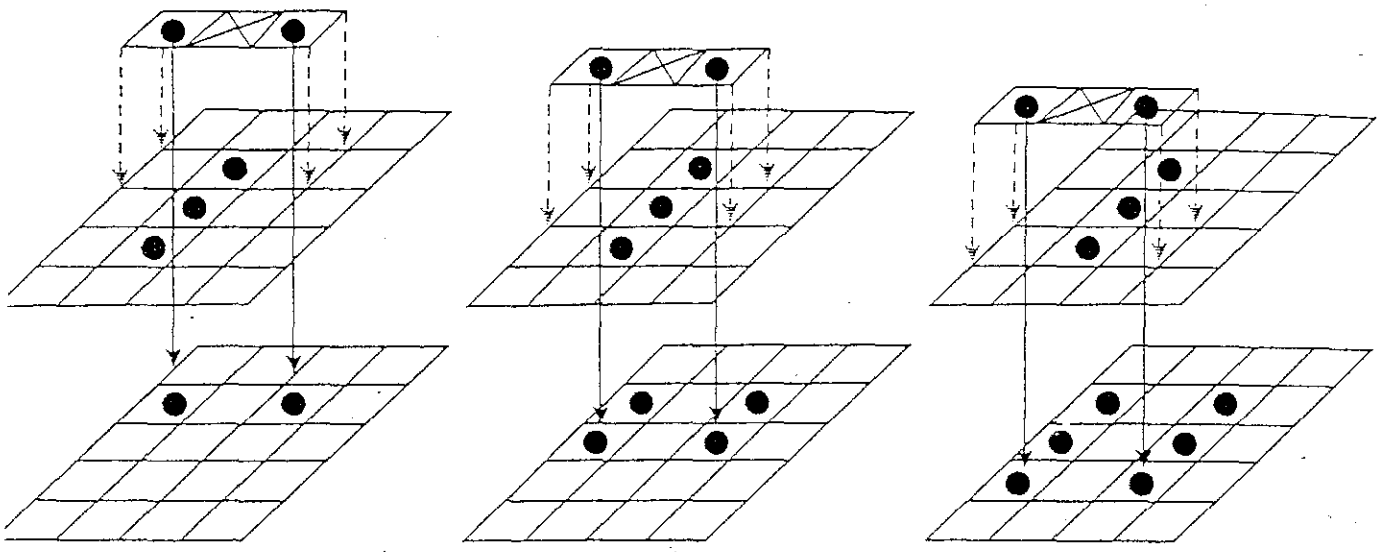


Figure 2.4 Dilating an image using a structuring element. (a) The origin of the structuring element is placed over the first black pixel in the image, and the pixels in the structuring element are copied into their corresponding positions in the result image. (b) Then the structuring element is placed over the next black pixel in the image and the process is repeated. (c) This is done for every black pixel in the image.

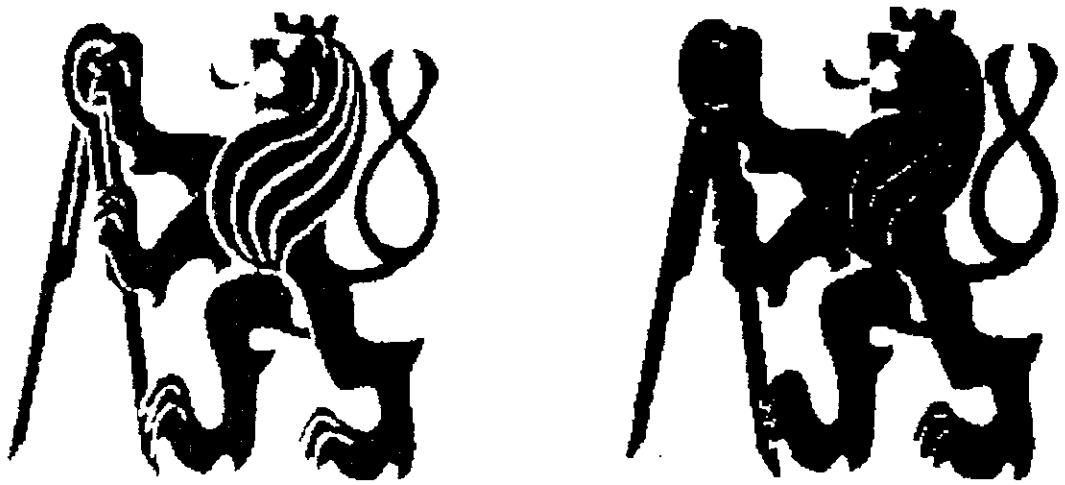


Figure 11.5: *Dilation as isotropic expansion.*

s several interesting properties that may ease its hardware or softw
present some here without proof. The interested reader may consult
paper [Haralick et al. 87].
n operation is commutative,

$$X \oplus B = B \oplus X$$

ociative,

$$X \oplus (B \oplus D) = (X \oplus B) \oplus D$$

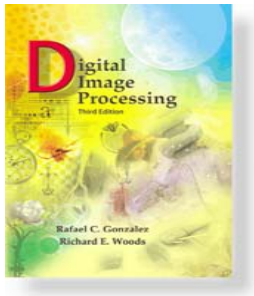
lso be expressed as a union of shifted point sets,

$$X \oplus B = \bigcup_{b \in B} X_b$$

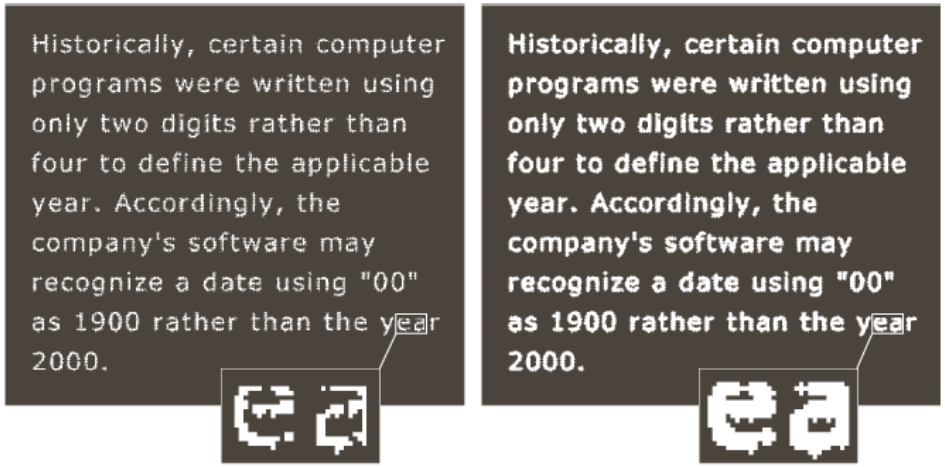
t to translation,

$$X_h \oplus B = (X \oplus B)_h$$

12) and (11.13) show the importance of shifts in speeding up imple



Chapter 9
Morphological Image Processing



0	1	0
1	1	1
0	1	0

a c
b

FIGURE 9.7
(a) Sample text of poor resolution with broken characters (see magnified view). (b) Structuring element. (c) Dilation of (a) by (b). Broken segments were joined.

Erosion

With A and B as sets in Z^2, the erosion of A by B, denoted $A \ominus B$, is defined as

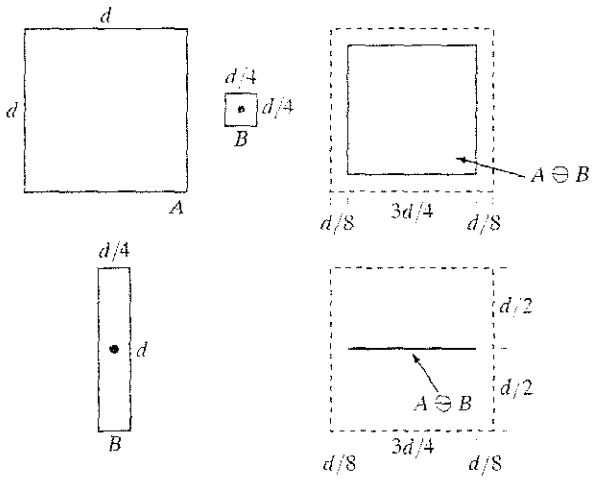
$$A \ominus B = \{z | (B)_z \subseteq A\} \tag{9.2-1}$$

In words, this equation indicates that the erosion of A by B is the set of all points z such that B, translated by z, is contained in A. In the following discussion, set B is assumed to be a structuring element. Equation (9.2-1) is the mathematical formulation of the example in Fig. 9.3(e), discussed at the end of the last section. Because the statement that B has to be contained in A is equivalent to B not sharing any common elements with the background, we can express erosion in the following equivalent form:

$$A \ominus B = \{z | (B)_z \cap A^c = \emptyset\} \tag{9.2-2}$$

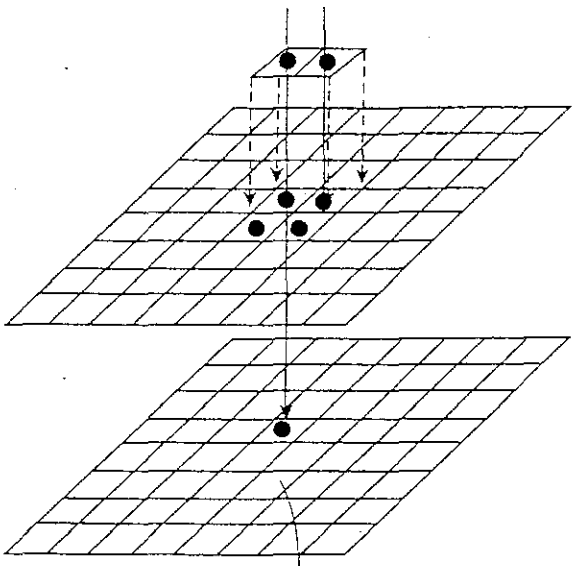
where, as defined in Section 2.6.4, A^c is the complement of A and $\emptyset$ is the empty set.

Figure 9.4 shows an example of erosion. The elements of A and B are shown shaded and the background is white. The solid boundary in Fig. 9.4(c) is the limit beyond which further displacements of the origin of B would cause the structuring element to cease being completely contained in A. Thus, the locus of points (locations of the origin of B) within (and including) this boundary, constitutes the erosion of A by B. We show the erosion shaded in Fig. 9.4(c). Keep in mind that that erosion is simply the *set* of

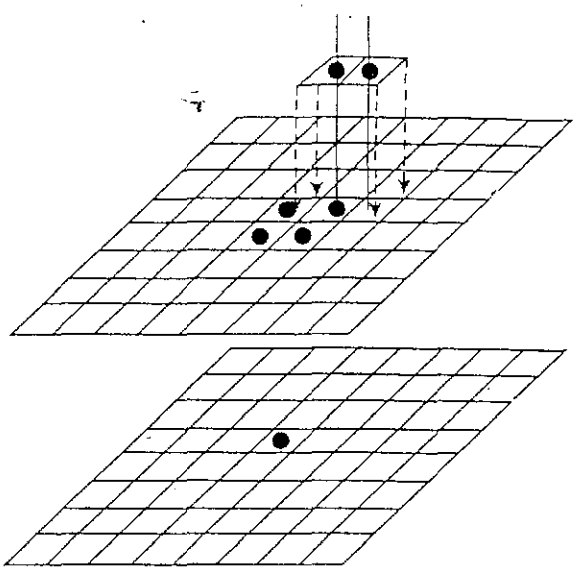


a b c
d e

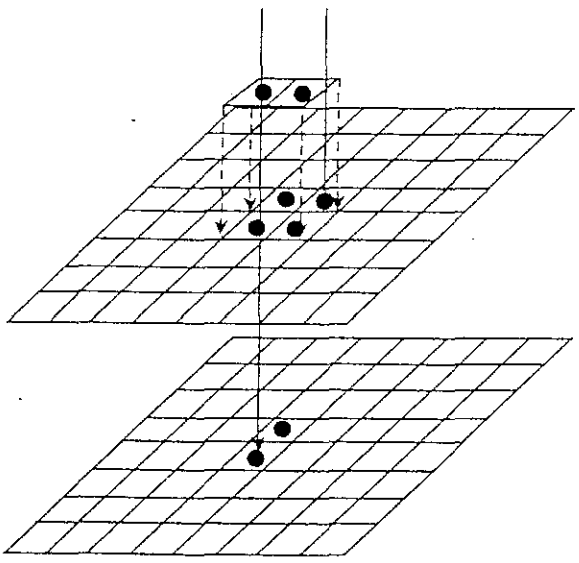
FIGURE 9.4 (a) Set A. (b) Square structuring element, B. (c) Erosion of A by B, shown shaded. (d) Elongated structuring element. (e) Erosion of A by B using this element. The dotted border in (c) and (e) is the boundary of set A, shown only for reference.



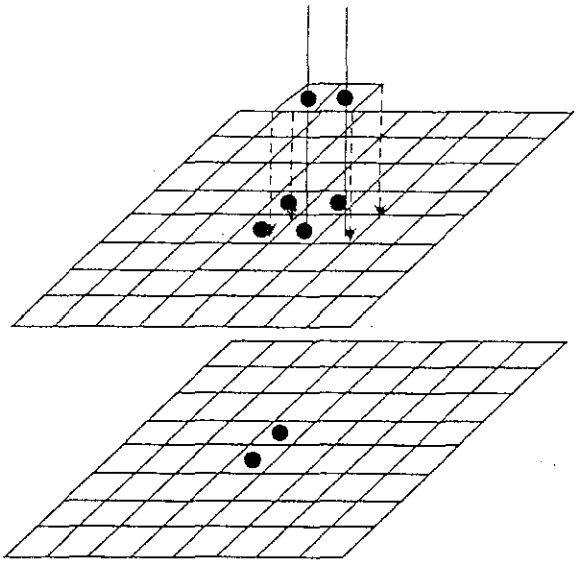
The structuring element is translated to the position of a black pixel in the image. In this case all members of the structuing element correspond to black image pixels, so the result is a black pixel.



Now the structuring element is translated to the next black pixel in the image, and there is one pixel that does not match. The result is a white pixel.

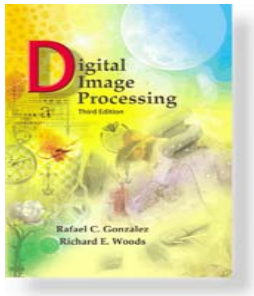


At the next translation there is another match so, again, the pixel in the output image that corresponds to the translated origin of the structuring element is set to black.

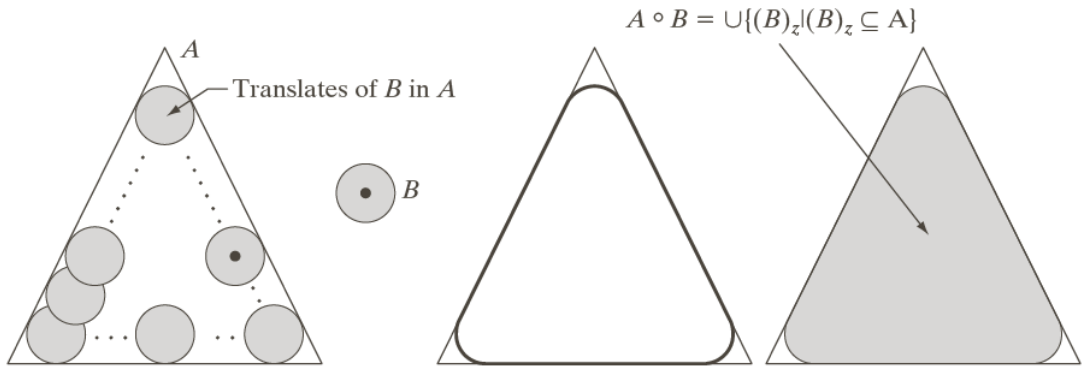


The final translation is not a match, and the result is a white pixel. The remaining image pixels are white and could not match the origin of the structuring element; they need not be considered.

Figure 2.6 Binary erosion using a simple structuring element.



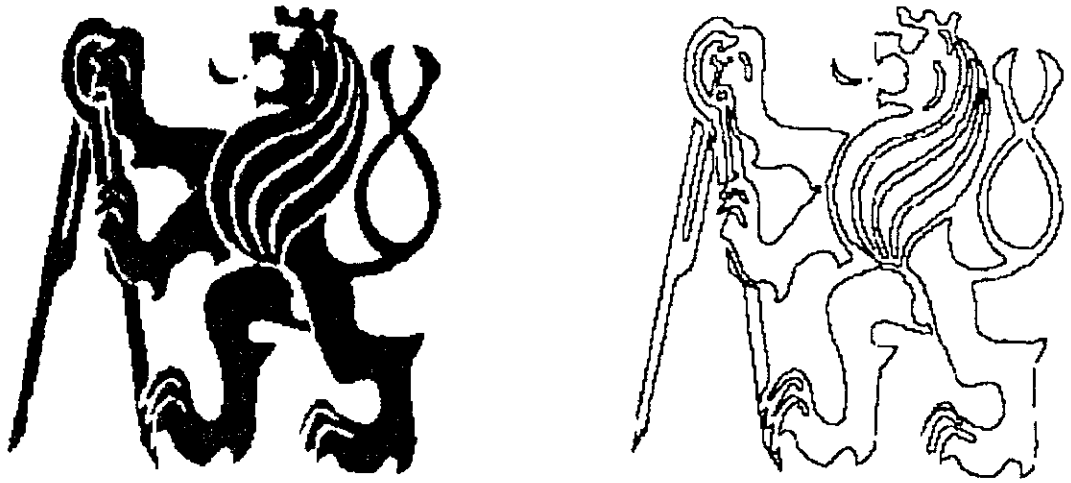
Chapter 9
Morphological Image Processing



a b c d

FIGURE 9.8 (a) Structuring element B "rolling" along the inner boundary of A (the dot indicates the origin of B). (b) Structuring element. (c) The heavy line is the outer boundary of the opening. (d) Complete opening (shaded). We did not shade A in (a) for clarity.

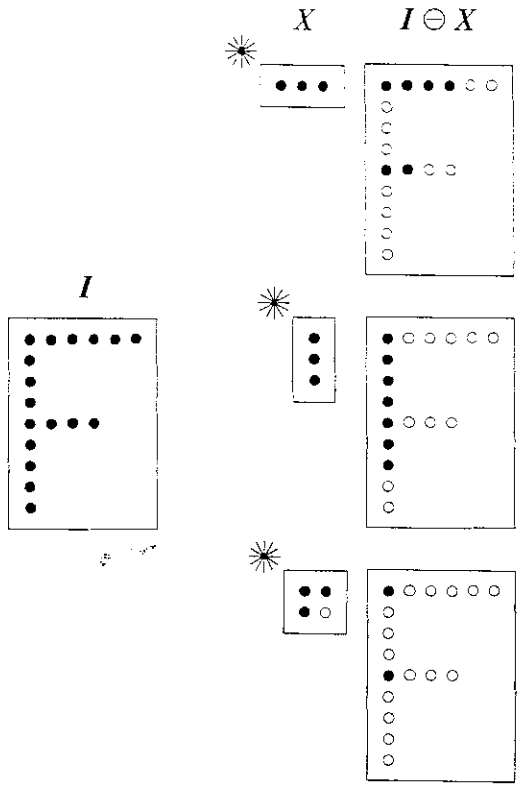
Figure 11.8: *Erosion as isotropic shrink.*



11.9: *Contours obtained by subtraction of an eroded image from a*

ıion might be interpreted by structuring element B sliding across th
ıslated by the vector p is contained in the image X, the point corı
tative point of B belongs to the erosion $X \ominus B$.
mplementation of erosion might be simplified by noting that an im
:turing element B can be expressed as an intersection of all translat
e vector[1] $-b \in B$:

$$X \ominus B = \bigcap_{b \in B} X_{-b}$$



$\boldsymbol{X}$, the structuring element does not contain the origin. In Figure 13.3 a more elaborate example shows that erosion is 'marking' occurrences of patterns in the image. It is as if the structuring element was a probe which is moved over the image, recording a 'hit' whenever it corresponds to the same structure.

Combining dilation and erosion – boundaries and contours

Because isotropic shrinking is accomplished by erosion, we can use the composite operation:

$$\boldsymbol{I} - (\boldsymbol{I} \ominus \boldsymbol{X})$$

to perform boundary extraction. Here we are shrinking the image and subtracting the result from the original. The subtrahend is thus the set of pixels 'lost' by the erosion process. For isotropic shrinking we can use a 3×3 or a 5×5 structuring element of 1's. The dimension of the structuring element will determine the width of the boundary detected. For example, using a 5×5 structuring element will result in a boundary set that is between 2 and 3 pixels thick.

Figure 2.7 gives some examples of erosions of a simple image by a collectio
of different structuring elements. The basic shape of the structuring elements is

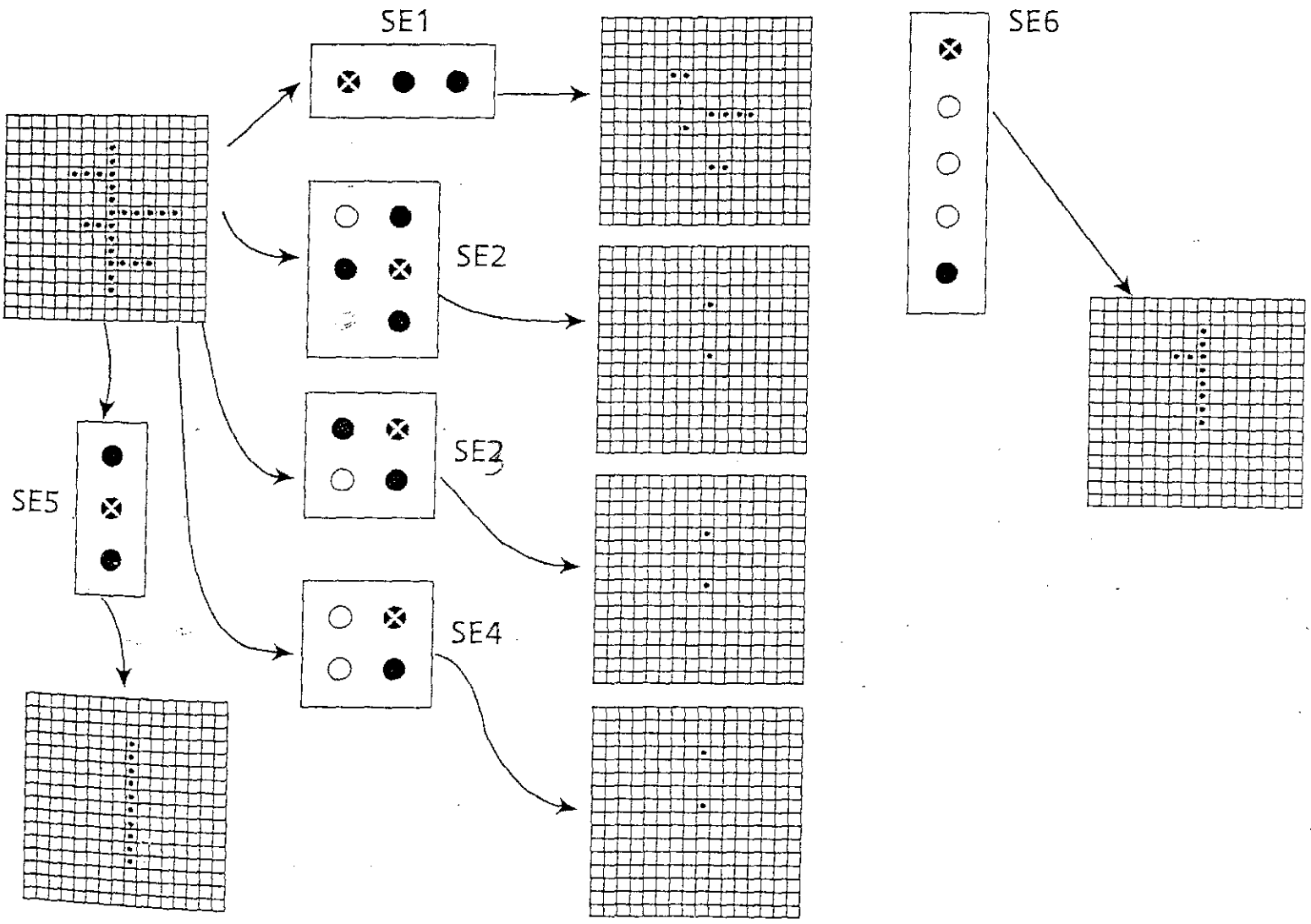


Figure 2.7 Examples of erosions by various structuring elements. The binary structuring elements are labelled SE1 through SE6.

pixel in the input image. For example, to determine the state of the pixel (4,6) in the output image:

- Overlay the structuring element on the input image, with the center pixel of the structuring element covering the pixel (4,6).
- Look at the pixels in the neighborhood of the input pixel. These are the five pixels covered by 1's in the structuring element. In this case the pixels are: (2,4), (3,5), (4,6), (5,7), (6,8). If all of these pixels are `on`, then set the pixel in the output image (4,6) to `on`. If any of these pixels is `off`, then set the pixel (4,6) in the output image to `off`.

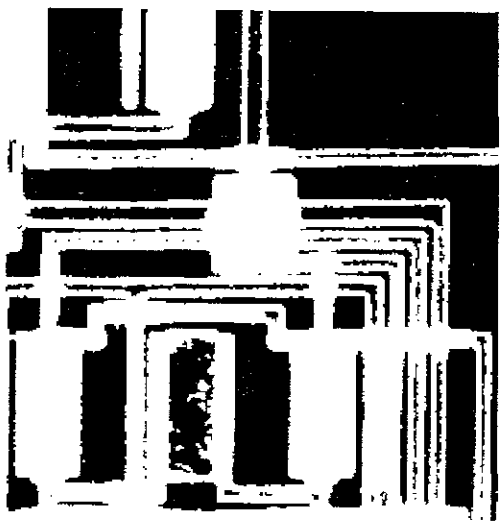
You perform this procedure for each pixel in the input image to determine the state of each corresponding pixel in the output image.

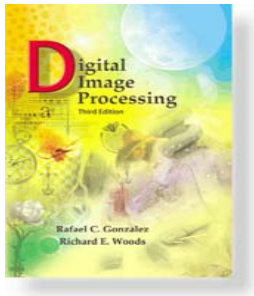
Note that for pixels on borders of the image, some of the 1's in the structuring element are actually outside the image. These elements are assumed to cover `off` pixels. As a result, the output image will usually have a black border, as in the example below.

The Image Processing Toolbox performs dilation through the `dilate` function, and erosion through the `erode` function. Each of these functions takes an input image and a structuring element as input, and returns an output image.

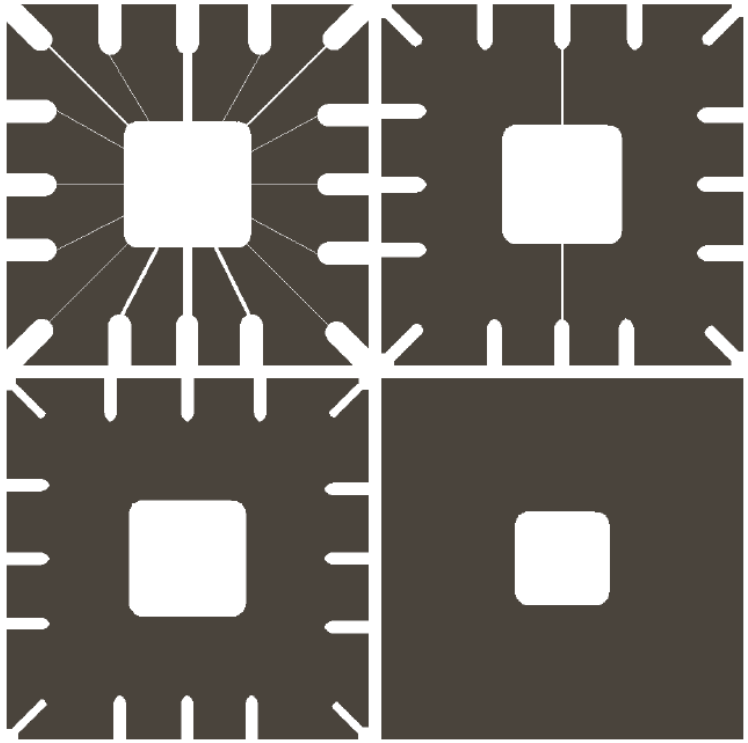
This example illustrates the erosion operation described above:

```
BW1 = imread('circbw.tif');
SE = eye(5);
BW2 = erode(BW1,SE);
imshow(BW1)
figure, imshow(BW2)
```





Chapter 9
Morphological Image Processing



a b
c d

FIGURE 9.5 Using erosion to remove image components. (a) A 486 × 486 binary image of a wire-bond mask. (b)–(d) Image eroded using square structuring elements of sizes 11 × 11, 15 × 15, and 45 × 45, respectively. The elements of the SEs were all 1s.

OPENING/CLOSING

Dilation: $A \oplus B$

Erosion: $\mathrm{A} \ominus \mathrm{B}$

Opening: $A \circ B = (A \ominus B) \oplus B$

Closing: $A \bullet B = (A \oplus B) \ominus B$

Thus

Opening

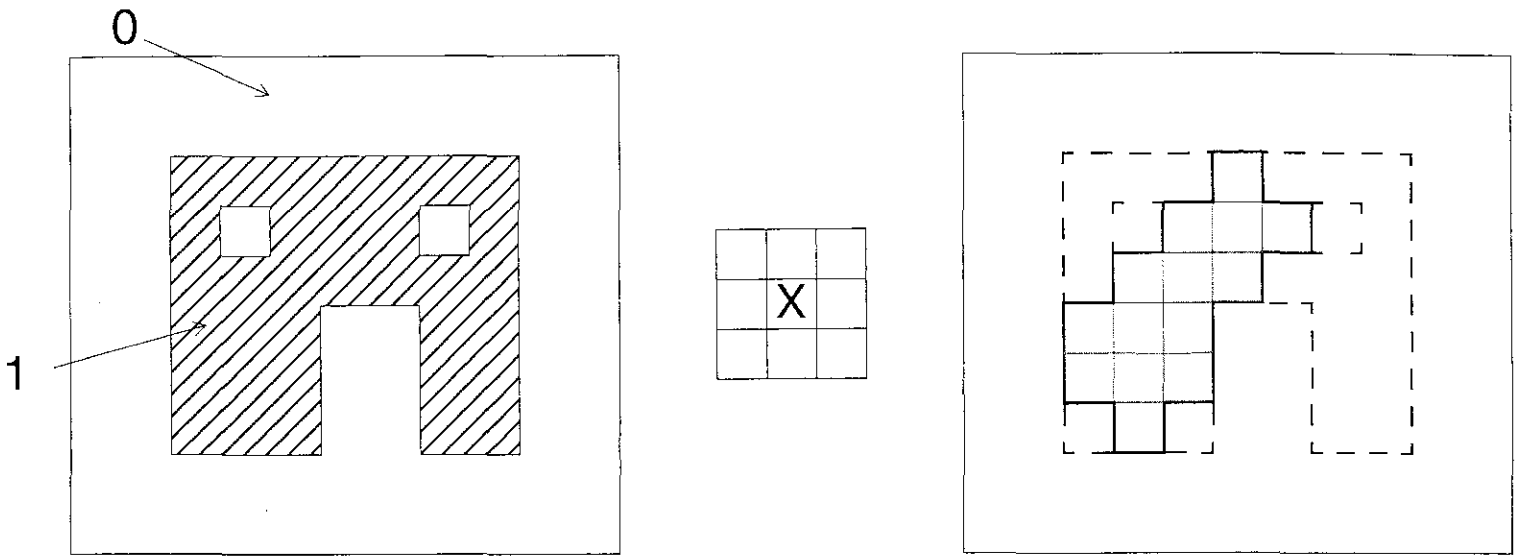
1. $A \circ B \subset A$
2. If $C \subset D$ then $\mathrm{C} \circ \mathrm{B} \subset \mathrm{D} \circ \mathrm{B}$
3. $(A \circ B) \circ B = A \circ B$

Closing

1. $A \subset A \bullet B$
2. If $C \subset D$ then $\mathrm{C} \bullet \mathrm{B} \subset \mathrm{D} \bullet \mathrm{B}$
3. $(A \bullet B) \bullet B = A \bullet B$

So, repeated opening or closing does nothing.
Only alternating them them accomplishes something

Figure 2.4-13 Opening

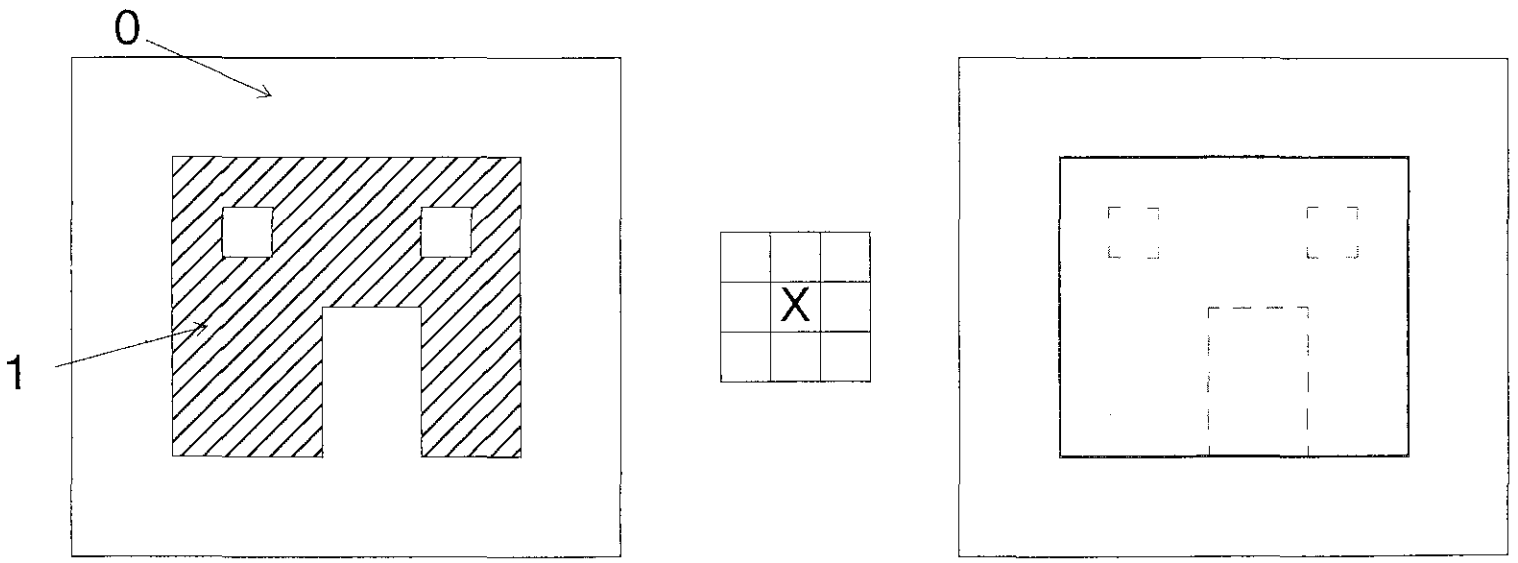


a. Original image.

b. Structural element; x = origin.

c. Image after opening; erosion followed by dilation.

Figure 2.4-14 Closing



a. Original image.

b. Structural element; x = origin.

c. Image after closing; dilation followed by erosion; original in dashes.

3. The number of iterations n.

The function $L(\ )$ and the values of a and b are all functions of the row and column, (r, c), but for concise notation this is implied. Set S can contain any or all of the 14 surrounds defined in Figure 2.4-16. $L(a, b)$ can be any logic function, but it turns out that the most useful are the AND and OR functions. The AND function tends to etch away at object boundaries (erosion), and the OR function tends to grow objects (dilation).

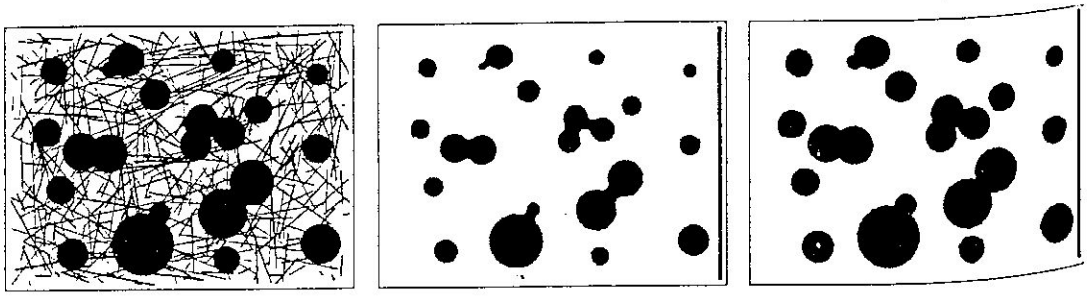


Figure 13.5
Opening a binary image with a disk structuring element.

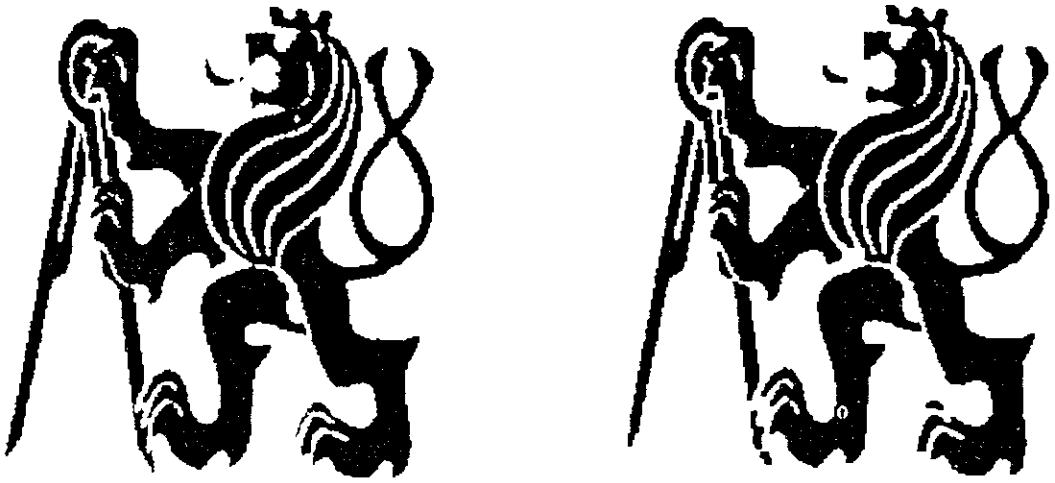


Figure 11.10: *Opening (original on the left).*

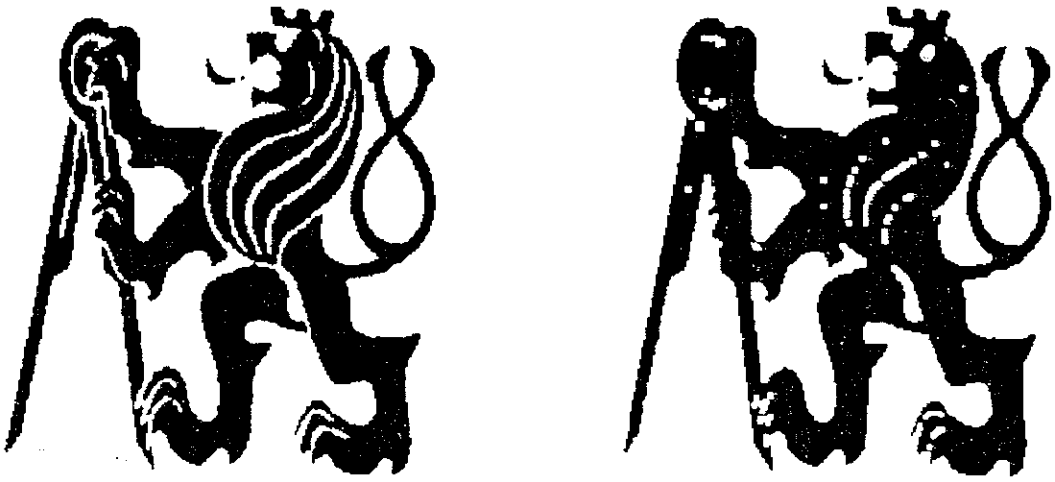


Figure 11.11: *Closing (original on the left).*

ınd erosion, opening and closing are invariant to translation of the s
ɹuations (11.14) and (11.20) imply that both opening and closin
ıations. Opening is anti-extensive ($X \circ B \subseteq X$) and closing is exte

ɔsing, like dilation and erosion, are dual transformations:

$$(X \bullet B)^C = X^C \circ \breve{B} \qquad (1$$

Opening and Closing

The application of an erosion immediately followed by a dilation using the same structuring element is referred to as an *opening* operation. The name opening is a descriptive one, describing the observation that the operation tends to "open" small gaps or spaces between touching objects in an image. This effect is most easily observed when using the simple structuring element. Figure 2.10 shows an image having a collection of small objects, some of them touching each other. After an opening using simple the objects are better isolated, and might now be counted or classified.

Figure 2.10 also illustrates another, and quite common, using of opening: the removal of noise. When a noisy grey-level image is thresholded some of the noise pixels are above the threshold, and result in isolated pixels in random locations. The erosion step in an opening will remove isolated pixels as well as boundaries of objects, and the dilation step will restore most of the boundary pixels without restoring the noise. This process seems to be successful at removing spurious black pixels, but does not remove the white ones.

The example in Figure 2.8 is actually an example of opening, albeit with a more complex structuring element. The image was eroded (leaving only a horizontal line), and then dilated by the same structuring element, which is certainly an opening. What is being eroded in this case is all portions of the image that are not staff lines, which the dilation subsequently restores. The same description of the process applies to Figure 2.10: What is being eroded is all parts of the image that

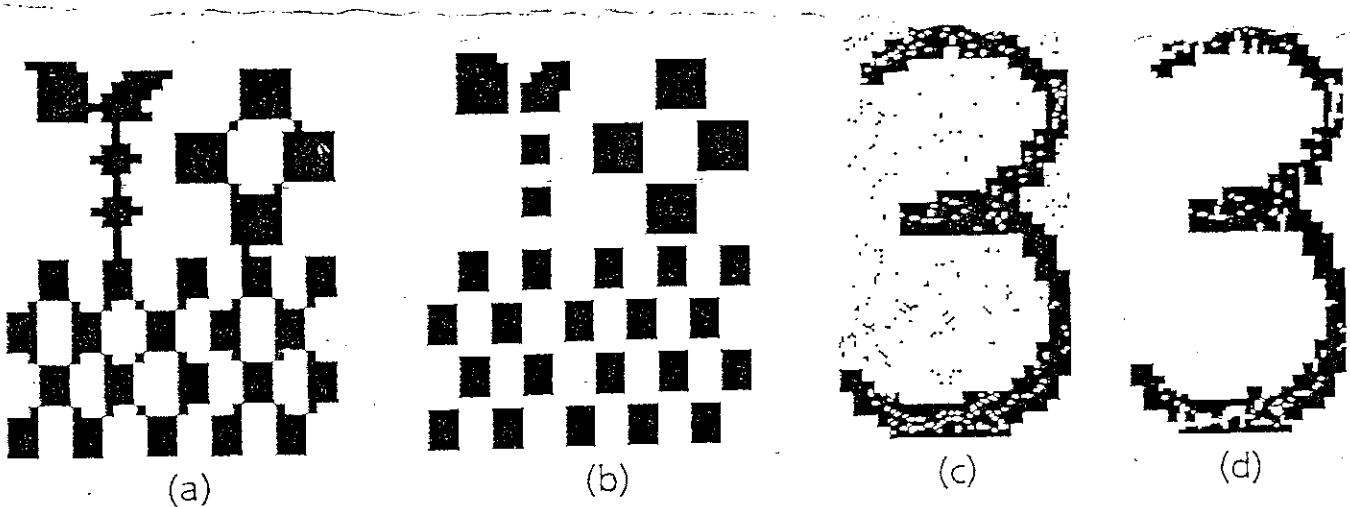


Figure 2.10 The use of opening. (a) An image having many connected objects. (b) Objects can be isolated by opening using the simple structuring element. (c) An image that has been subjected to noise. (d) The noisy image after opening showing that the black noise pixels have been removed.

are not small black squares, which are restored by the dilation, thus removing everything except that in which we are interested.

A *closing* is similar to an opening except that the dilation is performed first, followed by an erosion using the same structuring element. If an opening creates small gaps in the image, a closing will fill them, or "close" the gaps. Figure 2.11a shows a closing applied to the image of Figure 2.10d, which you may remember was opened in an attempt to remove noise. The closing removes much of the white pixel noise, giving a fairly clean image.

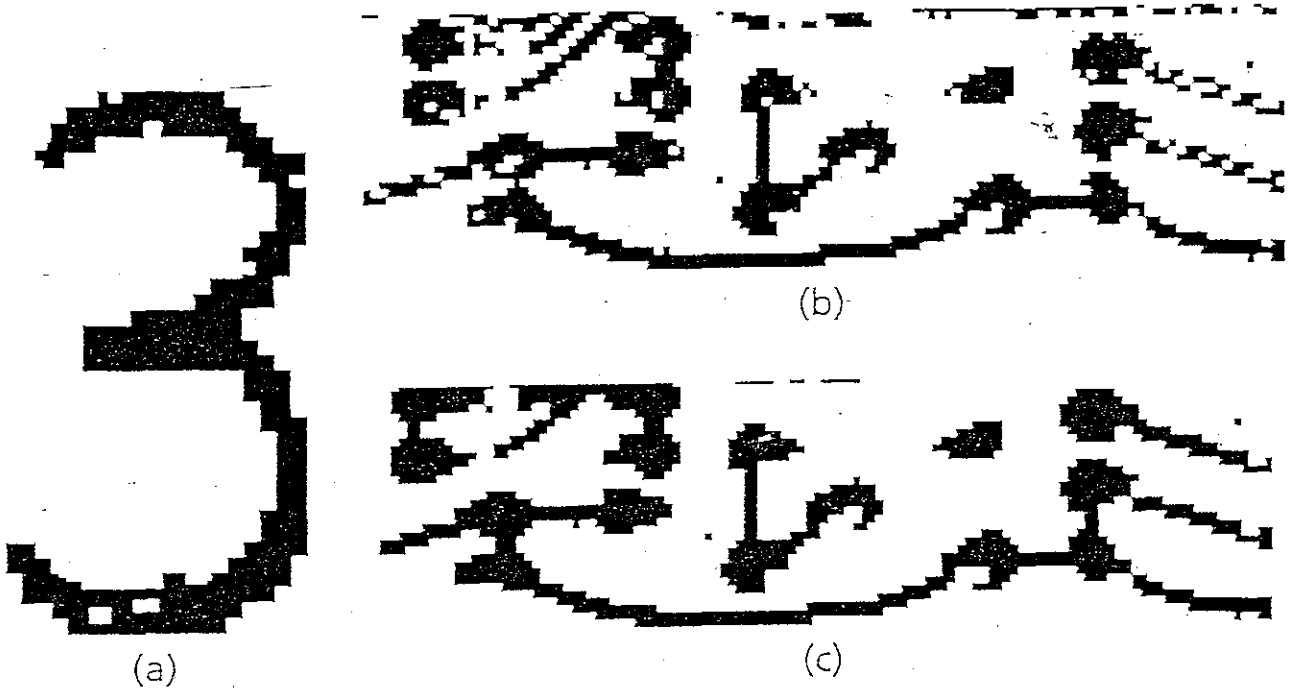


Figure 2.11 The closing operation. (a) The result of closing Figure 2.10d using the simple structuring element. (b) A thresholded image of a circuit board, showing broken traces. (c) The same image after closing, showing that most of the breaks have been repaired.

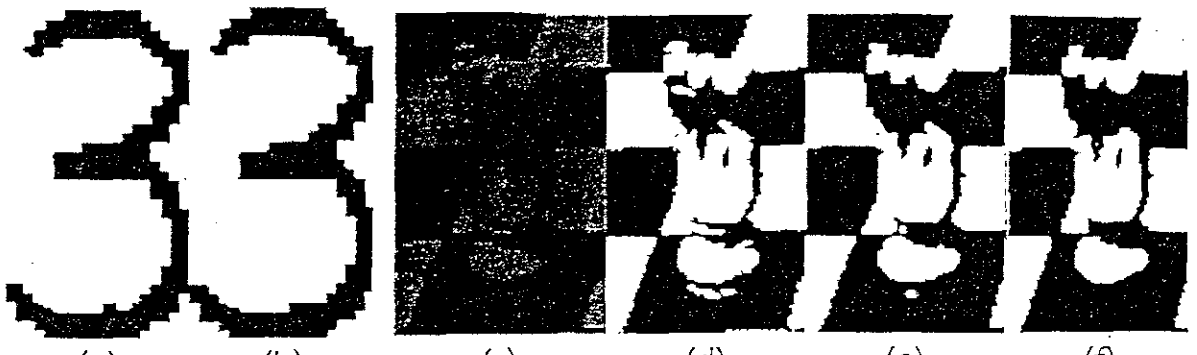


Figure 2.12 Multiple closings for outline smoothing. (a) Glyph from Figure 2.11a after a depth 2 closing. (b) After a depth 3 closing. (c) A chess piece. (d) Thresholded chess piece showing irregularities in the outline and some holes. (e) Chess piece after closing. (f) Chess piece after a depth 2 closing.

for this latter option can be prohibitive, and if file storage is used the I/O time can be large also.

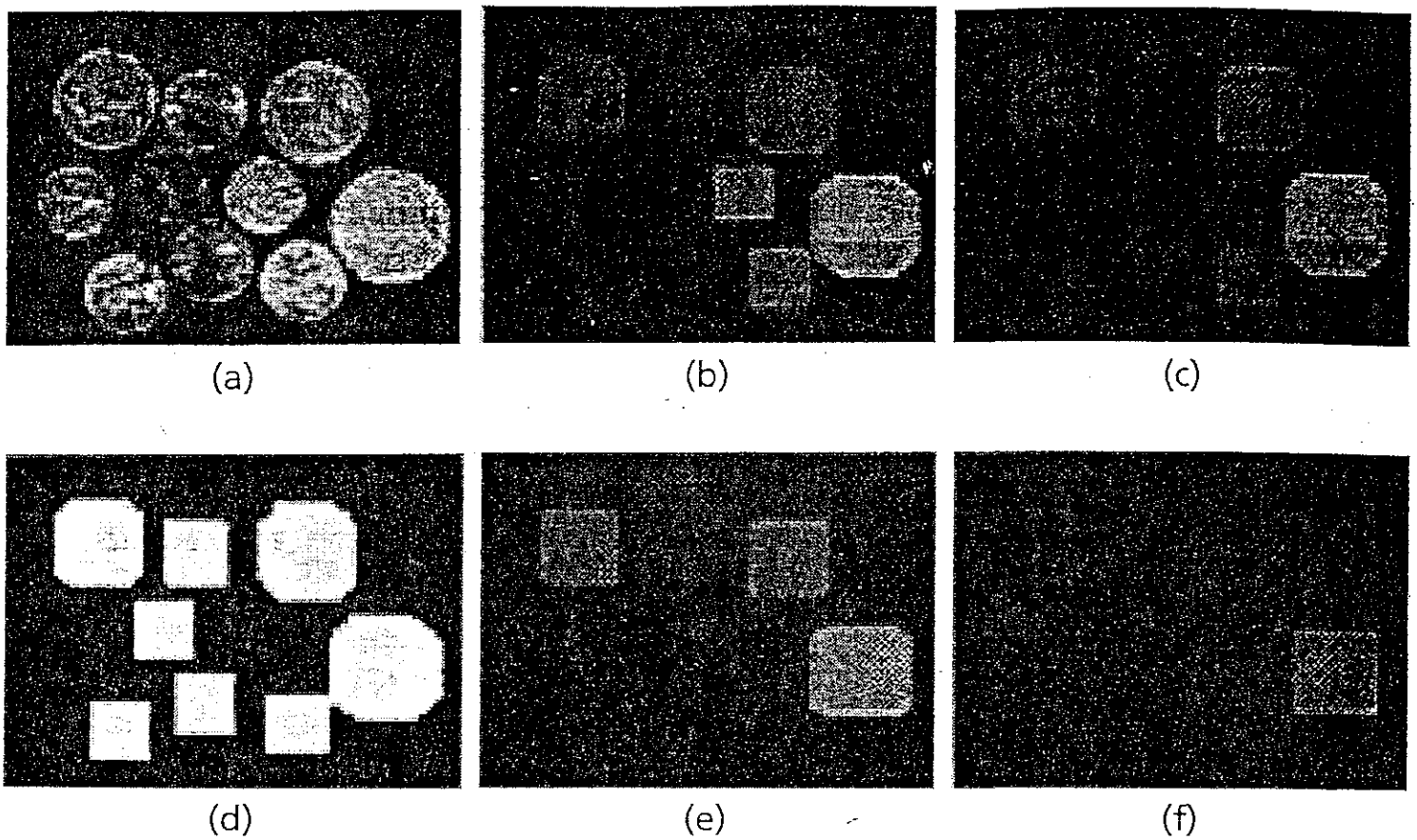


Figure 2.31 Classifying coins by their size. (a) The image containing coins to be classified. (b) After opening by a structuring element of radius 6. (c) After opening by radius 6.5. (d) Thresholded version of (c), showing that the dimes can be removed. (e) After opening by radius 8, showing that the pennies have been removed. (f) After opening by radius 10; the only coin remaining is the one-dollar coin.

shape would require quite a bit of experimenting with different structuring elements, size classification will be explored here. Quite a variety of objects are regularly classified according to their size, from biological objects under a microscope to eggs and apples. A "grade A large" egg, for example, should be noticeably bigger than a "medium" egg, and it should be possible to create a program for classifying eggs using grey-level morphology. However, since eggs are often graded using their weight, we will examine another case close to all of us—that of money.

As it happens, and not by accident, coins vary in size according to their value. A dime is the smallest, and a one-dollar coin is (if you can find one) the biggest. Figure 2.31a shows an image of a small collection of coins on a dark background. It is a mixture of U.S. and Canadian coins, since it was easy to obtain a Canadian one-dollar coin (called a *loon*).

Figure 2.8 is a better illustration of the use of erosion elements in a practical sense. The problem is to design a structuring element that will locate the staff lines in a raster image of printed music. The basic problem is to isolate the symbols so that, once identified, the staff lines will be removed. The structuring element consists of five horizontal straight line segments separated by "don't care" pixels—the latter corresponds to whatever occupies the space between the staff lines: note heads, sharps, etc. In effect, these elements act as spacers, permitting the combination of five distinct structuring elements into one.

After an erosion by the structuring element, each short section of staff lines has been reduced to a single pixel. The staff lines can be regenerated by a dilation of the eroded image by the same structuring element (Figure 2.8d). If it is necessary to remove the staff lines, this can be done by subtracting this image from the original (Figure 2.8e). There are now gaps in the image where the lines used to be, but otherwise the music symbols are free of the lines. A further morphological step can fill in some of the gaps (Figure 2.8f).

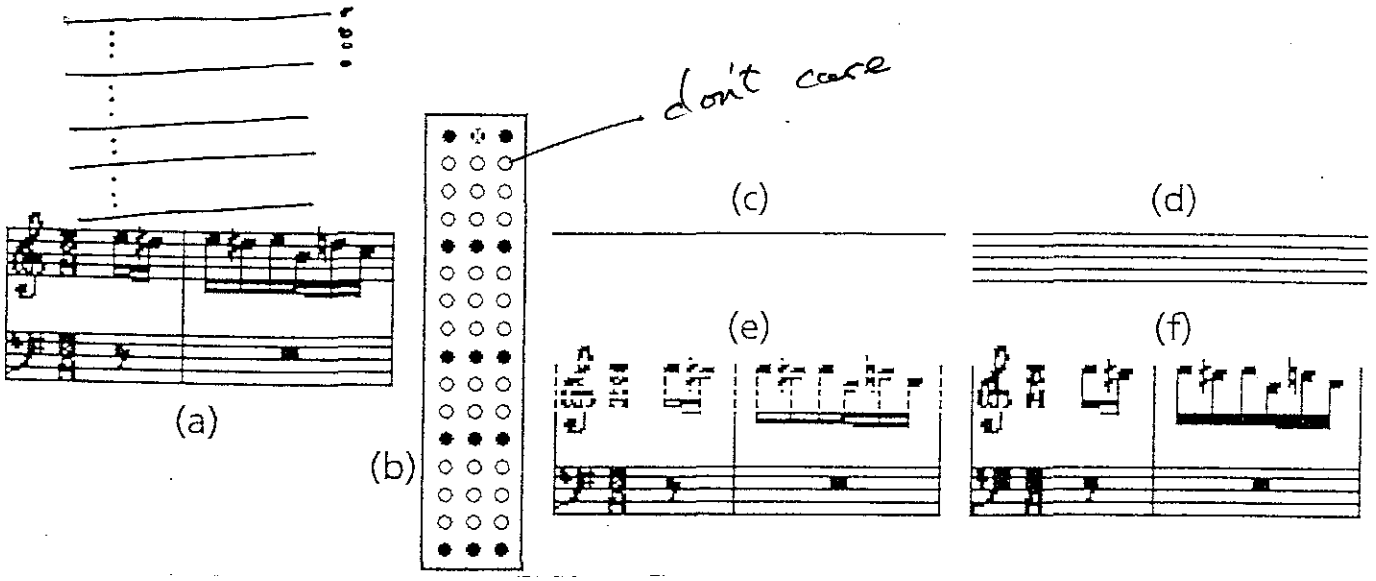


Figure 2.8 Morphological removal of staff lines from printed music. (a) The original image. (b) The structuring element. (c) Result of the erosion of (a) by (b). (d) Result of dilating again by the same structuring element. (e) Subtract (d) from (a). (f) Use a simple morphological operator to fill in the gaps.

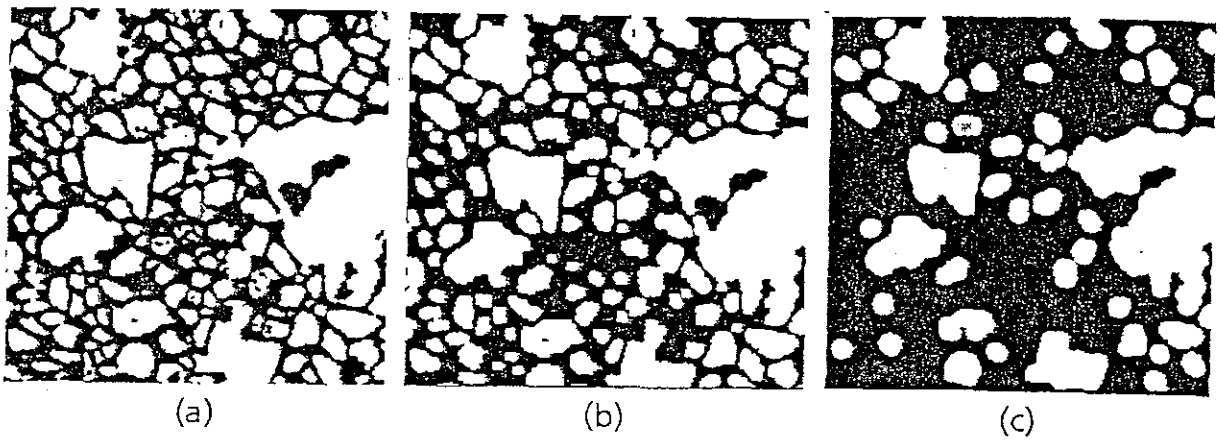


Figure 2.15 Computing the roughness of reservoir rock pores. (a) A pore image. (b) Opening of depth 3. Grey areas are pixels that have been changed. (c) Opening of depth 6.

Figure 13.6
A series of opening/subtracting operations (based on an illustration by Haralick and Shapiro (1992)).

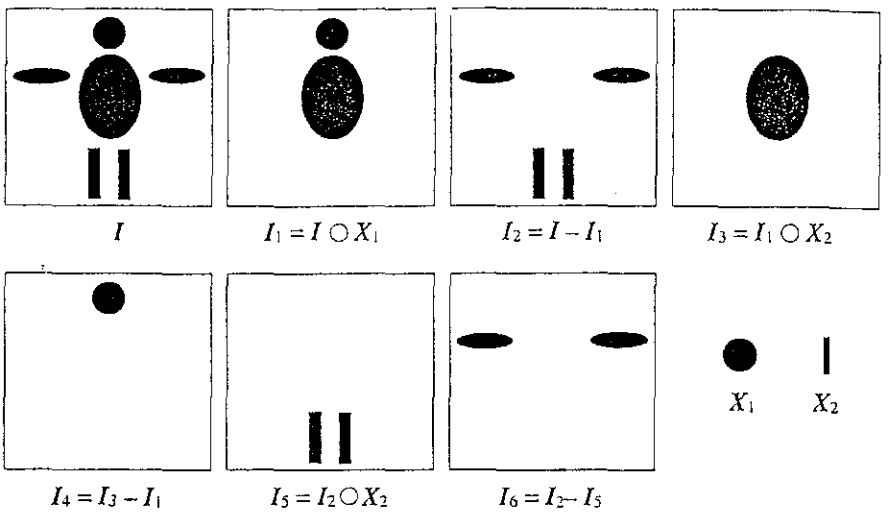
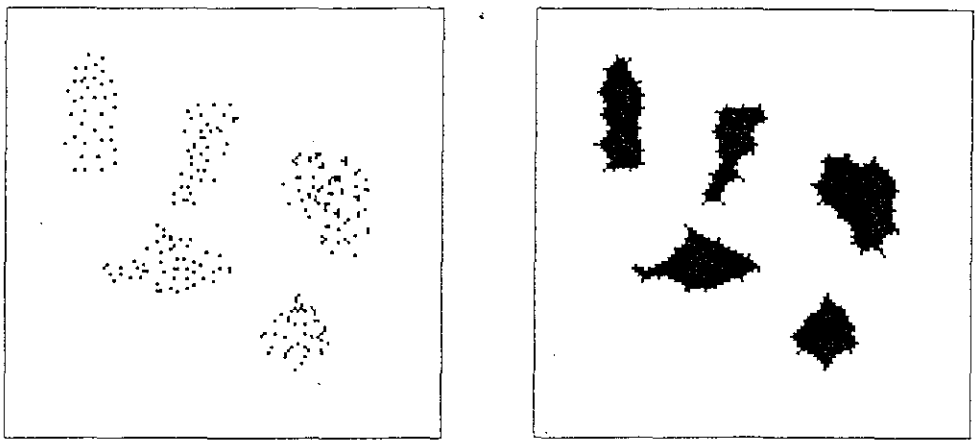
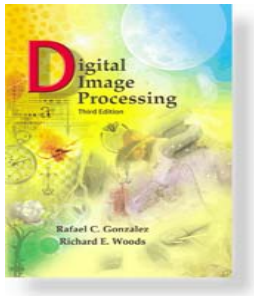


Figure 13.7
Using the closing operation to cluster points.



An example of an application of closing is shown in Figure 13.7. Here clusters of points are changed into connected sets. Thus each point can now be labelled as belonging to a certain cluster.



Chapter 9
Morphological Image Processing

a b
d c
e f

FIGURE 9.11
(a) Noisy image.
(b) Structuring element.
(c) Eroded image.
(d) Opening of A.
(e) Dilation of the opening.
(f) Closing of the opening.
(Original image courtesy of the National Institute of Standards and Technology.)

3.8.1 Basic morphological gradients

A common assumption in image analysis consists in considering image objects as regions of rather homogeneous grey levels. It follows that object boundaries or edges are located where there are high grey level variations. Gradient operators are used to enhance these variations. When the image signal is disturbed by some noise signal, it should be filtered before applying a gradient operator so as to avoid enhancing the noise component.

Many gradient operators have been proposed in image analysis because there is no unique discrete equivalent of the gradient operator defined for differentiable continuous functions. Morphological gradients are operators enhancing variations of pixel intensity in a neighbourhood determined by a structuring element. The erosion/dilation outputs for each pixel the minimum/maximum value of the image in the neighbourhood defined by the SE. Variations are therefore enhanced by combining these elementary operators. Three combinations are currently used:

1. arithmetic difference between the dilation and the erosion;
2. arithmetic difference between the dilation and the original image;
3. arithmetic difference between the original image and its erosion.

Only symmetric structuring elements containing their origin are considered. By doing so, we make sure that the arithmetic difference is always nonnegative.

The basic morphological gradient, also called *Beucher gradient*, is defined as the arithmetic difference between the dilation and the erosion by the elementary structuring element B, of the considered grid. This morphological gradient is denoted by ρ:

$$\rho_B = \delta_B - \varepsilon_B. \tag{3.16}$$

From this latter equation, it can be seen that the morphological gradient outputs the maximum variation of the grey level intensities within the neighbourhood defined by the SE rather than a local slope. In the continuous case, the Beucher gradient is defined for a disc B whose radius λ tends to zero:

$$\rho_B = \lim_{\lambda \to 0} \frac{\delta_{B_\lambda} - \varepsilon_{B_\lambda}}{2\lambda}.$$

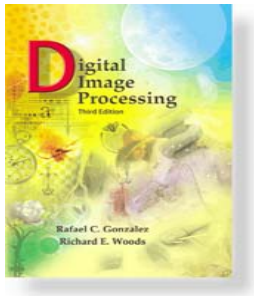
It can be shown that this definition is equivalent to the norm of the gradient vector of the image considered as a differentiable function:

$$\rho(f) = \|\nabla f\|,$$

where $\nabla f = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$. Note that the Beucher gradient is invariant to complementation: $\rho = \rho\complement$. It is therefore a self-complementary operator.

In situations where the components of the gradient vector need to be determined, alternative non-morphological methods should be considered. For example, an estimation of the $x-$ and $y-$components of the gradient vector can be obtained by the so-called Sobel operator whose convolution masks are given hereafter:

$$\frac{\partial f}{\partial x} = f * \frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline\end{array} \quad \text{and} \quad \frac{\partial f}{\partial y} = f * \frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline\end{array}.$$



Chapter 9

Morphological Image Processing

a b

FIGURE 9.14
(a) A simple binary image, with 1s represented in white. (b) Result of using Eq. (9.5-1) with the structuring element in Fig. 9.13(b).

These properties can be understood by the fact that once a hole or crack has been filled in, it remains filled in and cannot be filled in again. Likewise, once a hair or protrusion has been removed, it remains removed and cannot be recreated without evidence (e.g. without appealing to the original image data).

There are a number of other properties of closing and opening; amongst the most important ones are the following set containment properties:

$$A \oplus B \supseteq A \bullet B \supseteq A \tag{B.27}$$

$$A \ominus B \subseteq A \circ B \subseteq A \tag{B.28}$$

In particular, objects in a closed image will if anything be larger than the original objects, and objects in an opened image will if anything be smaller than the original objects.

B.3.5 Hit-and-miss transform

We now move on to a transform that can be used to search an image for particular instances of a shape or other characteristic image feature. It is defined by the expression:

$$A \otimes (B, C) = (A \ominus B) \cap (A^c \ominus C) \tag{B.29}$$

The fact that this expression contains an *and* (intersection) sign would appear to make the description "hit-and-miss" transform more appropriate than the name "hit-or-miss" transform which is sometimes used.

We first of all consider the image analysing properties of the $A \ominus B$ bracket. Applying B in this way (i.e. using erosion) leads to the location of a set of pixels where the 1s in the image match the 1s in the mask. Now notice that applying C to image A^c reveals a set of pixels where the 1s in C match the 1s in A^c or the 0s in A. However, it is usual to take a C mask which does not intersect with B (i.e. $B \cap C = \varnothing$). This indicates that we are matching a set of 0s in a more general mask D against the 0s in A. Thus D is an augmented form of B containing a specified set of 0s (corresponding to the 1s in C) and a specified set of 1s (corresponding to the 1s in B). Certain other locations in D contain neither 0s nor 1s and

can be regarded as "don't care" elements (which we will mark with an "×"). As an example, take the following lower left concave corner locating masks:

$B =$

		1
	1	1

$C =$

1	1	
1	1	

$D =$

0	0	×
0	0	1
×	1	1

It will be seen that, in reality, this transform is equivalent to a code coincidence detector, with the power to ignore irrelevant pixels. Thus it searches for instances where specified pixels have the required 0 values and other specified pixels have the required 1 values: when it encounters such a pixel, it marks it with a 1 in the new image space, and otherwise returns a 0. This approach is quite general and can be used to detect a wide variety of local image conditions, including particular features, structures, shapes or objects. It is by no means restricted to 3×3 masks as the above example might indicate. In short, it is a general binary template matching operator. Its power might serve to indicate that erosion is ultimately a more important concept than dilation—at least in its discriminatory, pattern recognition properties.

B.3.6 Template matching

As indicated in the previous subsection, the hit-and-miss transform is essentially a template matching operator and can be used to search images for characteristic features or structures. However, it needs to be generalized significantly before it can be used for practical feature detection tasks, because features may appear in a number of different orientations or guises, and any one of them will have to trigger the detector. To achieve this we merely need to take the union of the results obtained from the various pairs of masks:

$$\cup_i A \otimes (B_i, C_i) = \cup_i (A \ominus B_i) \cap (A^c \ominus C_i) \tag{B.30}$$

or, equivalently, to take the union of the results obtained from the various D-type masks (see previous subsection).

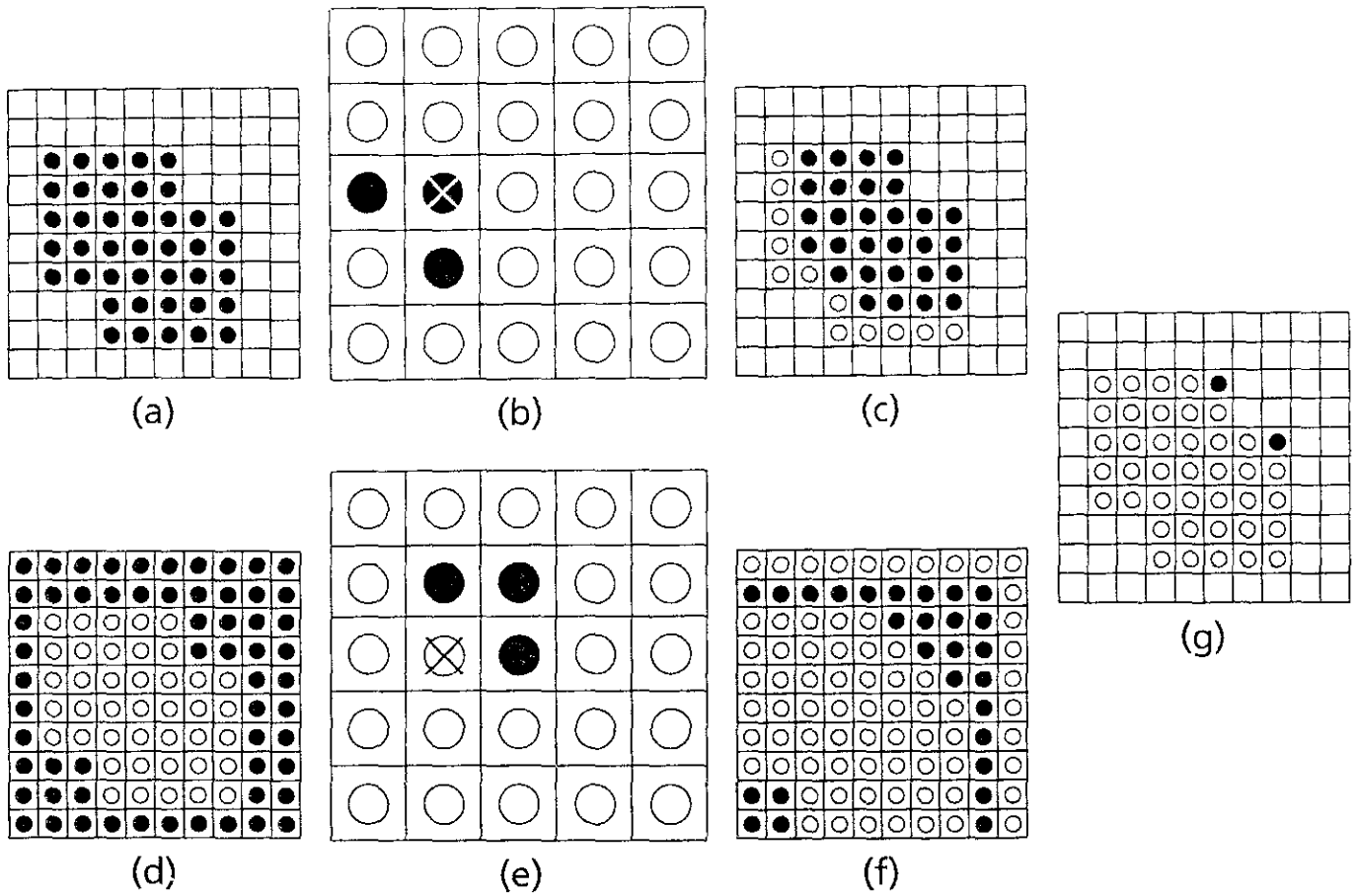


Figure 2.17 Illustration of the hit and miss transform. (a) The image to be examined. (b) Foreground structuring element for the location of upper right corners. (c) The erosion of (a) by (b)—the "hit" portion of the computation. (d) The complement of (a). (e) The background structuring element, showing that the three pixels to the upper right of the corner must be background pixels. (f) The erosion of (d) by (e), or the "miss" portion of the computation. (g) The intersection of (c) and (f)—the result, showing the location of each of the two upper right corners in the original image.

the result of the intersection of the "hit" and the "miss." The set pixels in the result both correspond to corners in the image.

Also notice that the background structuring element is *not* the complement of the foreground structuring element; indeed, if it had been, then the result would have been an empty image since there is no match to its peculiar shape in the complement image. The set pixels in the background structuring element are those that *must* be background pixels in the image in order for a match to take place. Overspecification of these pixels results in few matches, and underspecification results in too many. Careful selection, possibly through experimentation, is needed.

By the way, the upper and right pixels in Figure 2.17f are white because they

Morphological Image Processing Lecture 22 (page 2)

• Objective is to find a disjoint region (set) in an image

• If B denotes the set composed of X and its background, the match/hit (or set of matches/hits) of B in A, is

$$A \circledast B = (A \ominus X) \cap [A^c \ominus (W - X)]$$

• Generalized notation: $B = (B_1, B_2)$

- B_1: Set formed from elements of B associated with an object
- B_2: Set formed from elements of B associated with the corresponding background

[Preceeding discussion: $B_1 = X$ and $B_2 = (W - X)$]

- More general definition:

$$A \circledast B = (A \ominus B_1) \cap [A^c \ominus B_2]$$

- $A \circledast B$ contains all the origin points at which, simultaneously, B_1 found a hit in A and B_2 found a hit in A^c

• Alternative definition:

$$A \circledast B = (A \ominus B_1) - (A \oplus \hat{B}_2)$$

• A background is necessary to detect disjoint sets

• When we only aim to detect certain patterns within a set, a background is not required, and simple erosion is sufficient

9.5 Some basic morphological algorithms

When dealing with **binary images**, the principle application of morphology is extracting image components that are useful in the representation and description of shape

9.5.1 Boundary extraction

The boundary $\beta(A)$ of a set A is

$$\beta(A) = A - (A \ominus B),$$

where B is a suitable structuring element

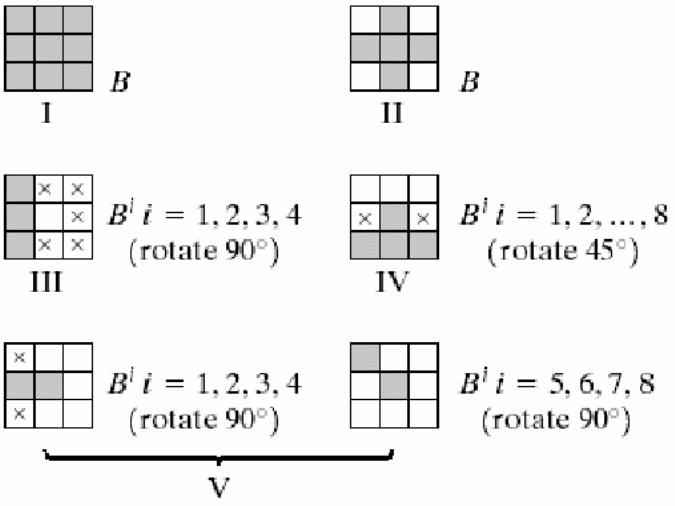


FIGURE 9.26 Five basic types of structuring elements used for binary morphology. The origin of each element is at its center and the ×'s indicate "don't care" values.

9.6 Extensions to grey-scale images

$f(x, y)$: Input image
$b(x, y)$: Structuring element image

9.6.1 Dilation

Grey-scale dilation of f by b, is defined as

$$(f \oplus b)(s, t) = \max \{f(s - x, t - y) + b(x, y) \\ | (s - x), (t - y) \in D_f; \, (x, y) \in D_b\},$$

where D_f and D_b are the domains of f and b, respectively

erode

Purpose

Perform erosion on a binary image

Syntax

```
BW2 = erode(BW1,SE)
BW2 = erode(BW1,SE,alg)
BW2 = erode(BW1,SE,...,n)
```

Description

`BW2 = erode(BW1,SE)` performs erosion on the binary image `BW1`, using the binary structuring element `SE`. `SE` is a matrix containing only 1's and 0's.

`BW2 = erode(BW1,SE,alg)` performs erosion using the specified algorithm. a is a string that can have one of these values:

- `'spatial'` (default) – processes the image in the spatial domain.
- `'frequency'` – processes the image in the frequency domain.

Both algorithms produce the same result, but they make different tradeoffs between speed and memory use. The frequency algorithm is faster for large images and structuring elements than the spatial algorithm, but uses much more memory.

`BW2 = erode(BW1,SE,...,n)` performs the erosion operation `n` times.

Class Support

The input image `BW1` can be of class `double` or `uint8`. The output image `BW2` of class `uint8`.

Remarks

You should use the frequency algorithm only if you have a large amount of memory on your system. If you use this algorithm with insufficient memory may actually be *slower* than the spatial algorithm, due to virtual memory paging. If the frequency algorithm slows down your system excessively, or you receive "out of memory" messages, use the spatial algorithm instead.

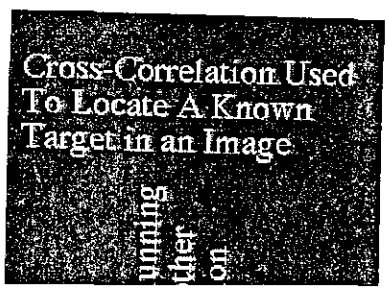
nple

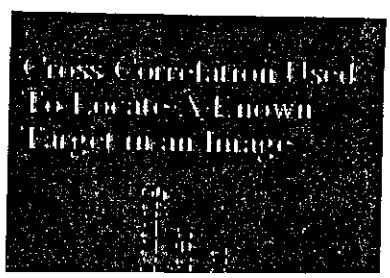
```
BW1 = imread('text.tif')
SE = ones(3,1)

SE =

     1
     1
     1

BW2 = erode(BW1,SE);
imshow(BW1)
figure, imshow(BW2)
```





Purpose Perform morphological operations on binary images

Syntax

```
BW2 = bwmorph(BW1,operation)
BW2 = bwmorph(BW1,operation,n)
```

Description `BW2 = bwmorph(BW1,operation)` applies a specific morphological operation to the binary image `BW1`.

`BW2 = bwmorph(BW1,operation,n)` applies the operation `n` times. `n` can be `Inf`, in which case the operation is repeated until the image no longer changes.

`operation` is a string that can have one of these values:

`'bothat'`	`'erode'`	`'shrink'`
`'bridge'`	`'fill'`	`'skel'`
`'clean'`	`'hbreak'`	`'spur'`
`'close'`	`'majority'`	`'thicken'`
`'diag'`	`'open'`	`'thin'`
`'dilate'`	`'remove'`	`'tophat'`

`'bothat'` ("bottom hat") performs binary closure (dilation followed by erosion) and subtracts the original image.

`'bridge'` bridges previously unconnected pixels. For example:

```
1  0  0                 1  0  0
1  0  1     becomes     1  1  1
0  0  1                 0  0  1
```

`'clean'` removes isolated pixels (individual 1's that are surrounded by 0's), such as the center pixel in this pattern:

```
0  0  0
0  1  0
0  0  0
```

`'close'` performs binary closure (dilation followed by erosion).

'diag' uses diagonal fill to eliminate 8-connectivity of the background. For example:

```
0  1  0                    0  1  0
1  0  0      becomes       1  1  0
0  0  0                    0  0  0
```

'dilate' performs dilation using the structuring element ones(3).

'erode' performs dilation using the structuring element ones(3).

'fill' fills isolated interior pixels (individual 0's that are surrounded by 1's), such as the center pixel in this pattern:

```
1  1  1
1  0  1
1  1  1
```

'hbreak' removes H-connected pixels. For example:

```
1  1  1                    1  1  1
0  1  0      becomes       0  0  0
1  1  1                    1  1  1
```

'majority' sets a pixel to 1 if five or more pixels in its 3-by-3 neighborhood are 1's; otherwise, it sets the pixel to 0.

'open' implements binary opening (erosion followed by dilation).

'remove' removes interior pixels. This option sets a pixel to 0 if all of its 4-connected neighbors are 1, thus leaving only the boundary pixels on.

'shrink', with n = Inf, shrinks objects to points. It removes pixels so that objects without holes shrink to a point, and objects with holes shrink to a connected ring halfway between each hole and the outer boundary. This option preserves the Euler number.

'skel', with n = Inf, removes pixels on the boundaries of objects but does not allow objects to break apart. The pixels remaining make up the image skeleton. This option preserves the Euler number.

`'spur'` removes spur pixels. For example:

```
0  0  0  0                 0  0  0  0
0  0  0  0                 0  0  0  0
0  0  1  0    becomes      0  0  0  0
0  1  0  0                 0  1  0  0
1  1  0  0                 1  1  0  0
```

`'thicken'`, with `n = Inf`, thickens objects by adding pixels to the exterior of objects until doing so would result in previously unconnected objects being 8-connected. This option preserves the Euler number.

`'thin'`, with `n = Inf`, thins objects to lines. It removes pixels so that an object without holes shrinks to a minimally connected stroke, and an object with holes shrinks to a connected ring halfway between each hole and the outer boundary. This option preserves the Euler number.

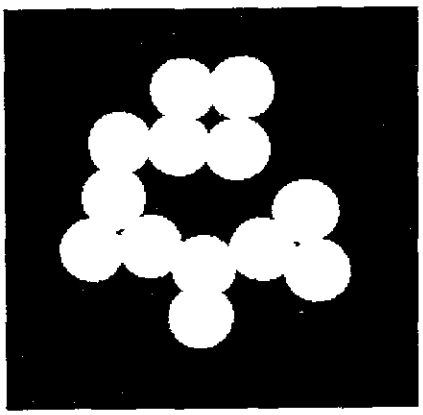
`'tophat'` ("top hat") returns the image minus the binary opening of the image.

Class Support

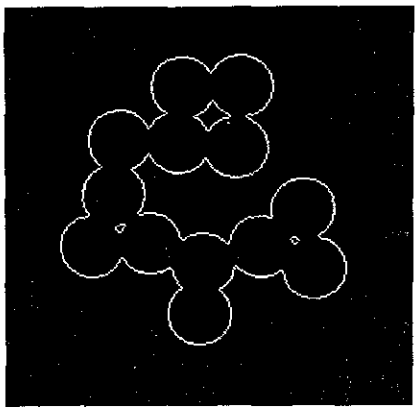
The input image `BW1` can be of class `double` or `uint8`. The output image `BW2` is of class `uint8`.

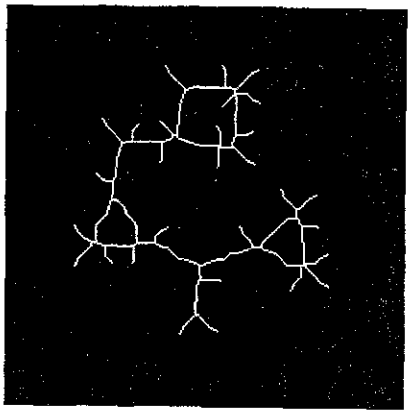
Examples

```
BW1 = imread('circles.tif');
imshow(BW1);
```



```
BW2 = bwmorph(BW1,'remove');
BW3 = bwmorph(BW1,'skel',Inf);
imshow(BW2)
figure, imshow(BW3)
```





See Also

`bweuler`, `bwperim`, `dilate`, `erode`

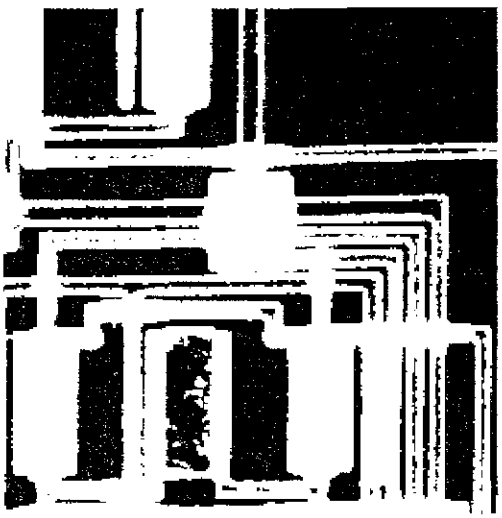
References

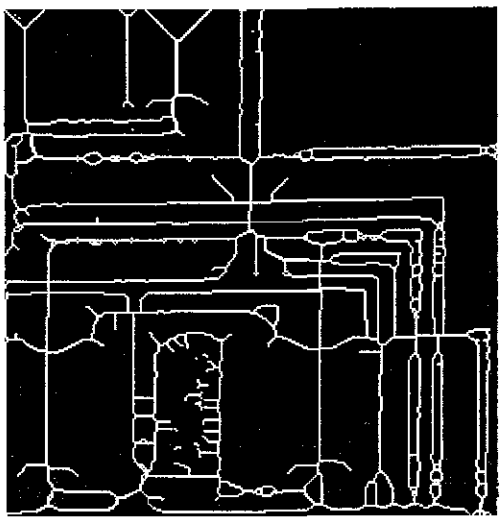
Haralick, Robert M., and Linda G. Shapiro. *Computer and Robot Vision, Volume I.* Addison-Wesley, 1992.

Pratt, William K. *Digital Image Processing*. John Wiley & Sons, Inc., 1991.

For example, suppose you want to reduce all objects in the circuit image to lines, without changing the essential structure (topology) of the image. This process is known as *skeletonization*. You can use `bwmorph` to do this:

```
BW1 = imread('circbw.tif');
BW2 = bwmorph(BW1,'skel',Inf);
imshow(BW1)
figure, imshow(BW2)
```





The third argument to `bwmorph` indicates the number of times to perform the operation. For example, if this value is 2, the operation is performed twice, with the result of the first operation being used as the input for the second operation. In the example above, the value is `Inf`. In this case `bwmorph` performs the operation repeatedly until it no longer changes.

For more information about the predefined operations available, see the reference entry for `bwmorph` in Chapter 11.

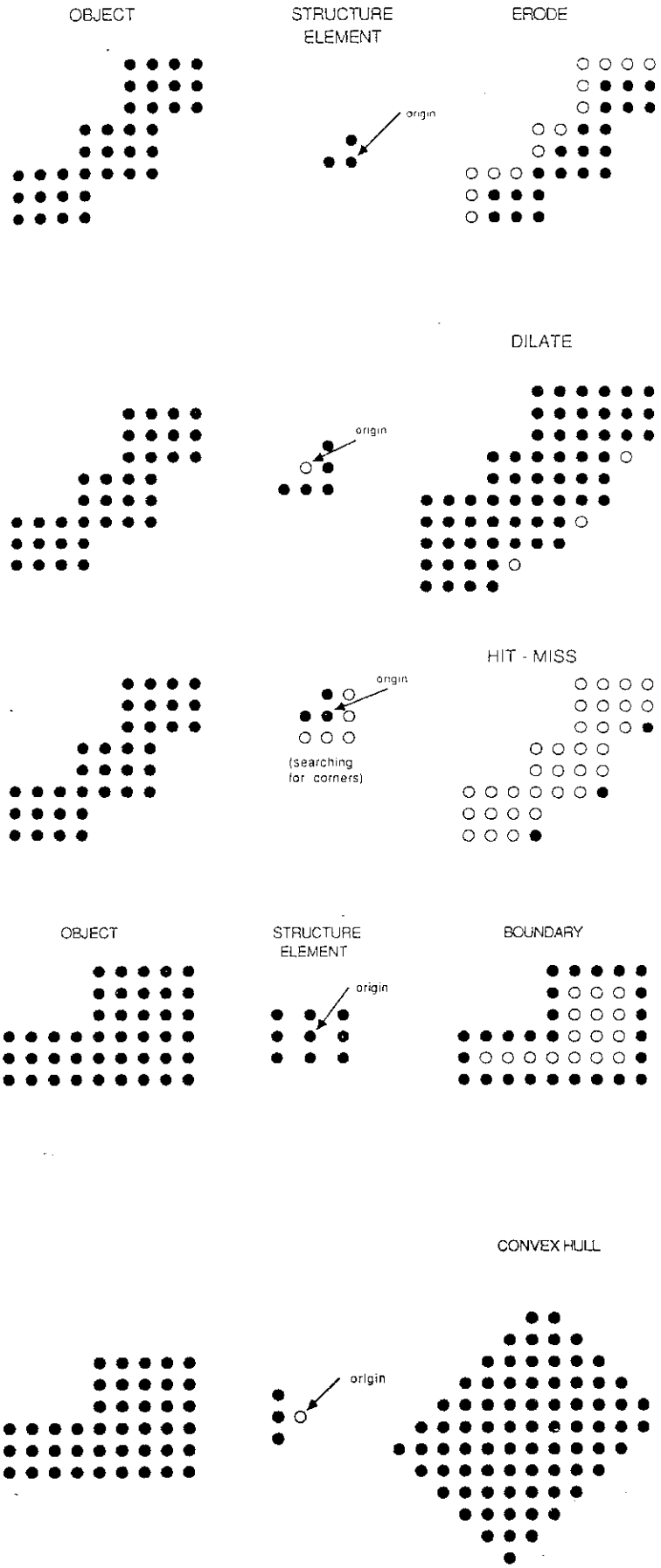


Figure 9.33 Examples of some morphological operations.

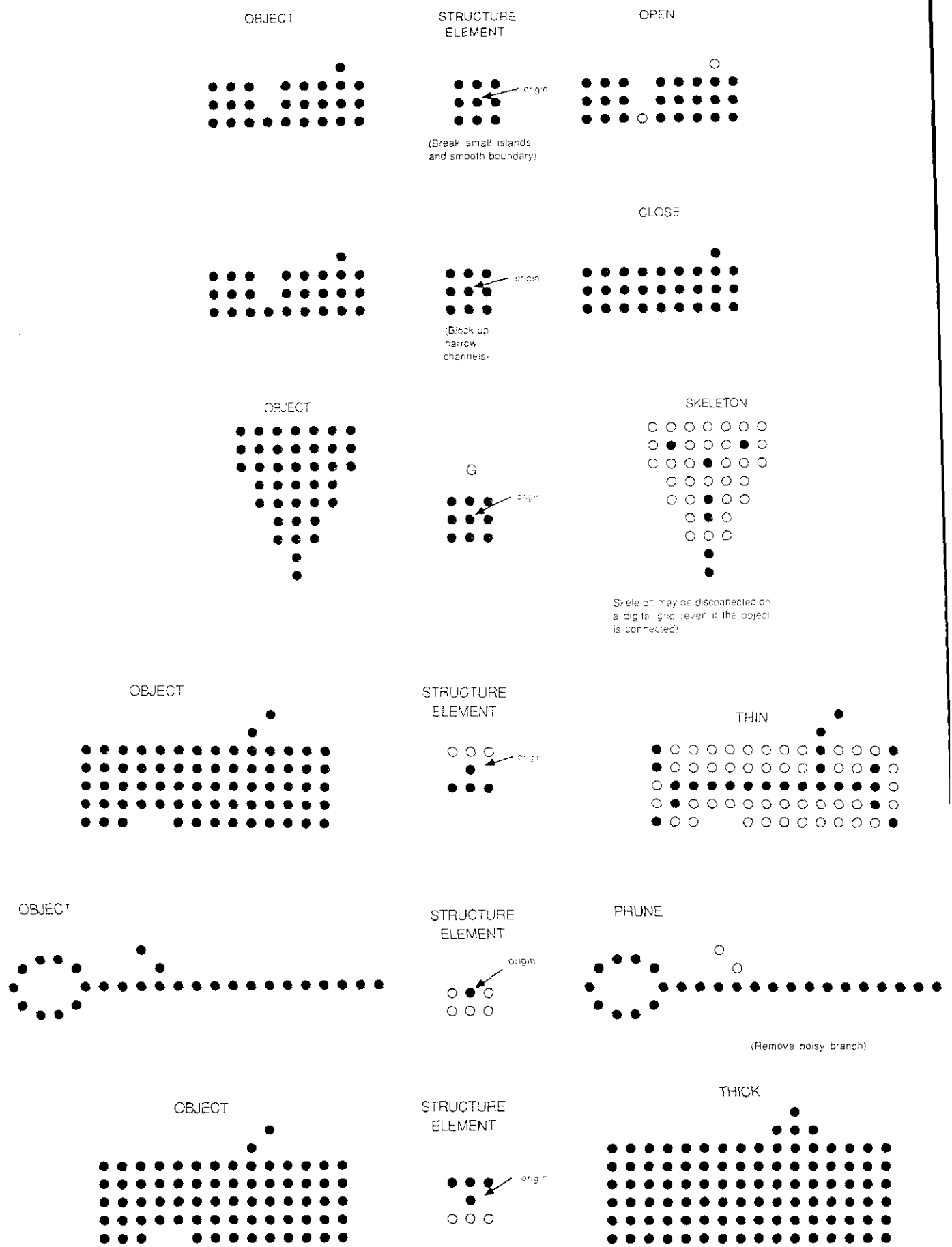


Figure 9.33 Cont'd.

Operation	Equation	Comments (The Roman numerals refer to the structuring elements shown in Fig. 9.26).
Translation	$(A)_z = \{w \mid w = a + z, \quad \text{for } a \in A\}$	Translates the origin of A to point z.
Reflection	$\hat{B} = \{w \mid w = -b, \quad \text{for } b \in B\}$	Reflects all elements of B about the origin of this set.
Complement	$A^c = \{w \mid w \notin A\}$	Set of points not in A.
Difference	$A - B = \{w \mid w \in A, w \notin B\} = A \cap B^c$	Set of points that belong to A but not to B.
Dilation	$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \varnothing\}$	"Expands" the boundary of A. (I)
Erosion	$A \ominus B = \{z \mid (B)_z \subseteq A\}$	"Contracts" the boundary of A. (I)
Opening	$A \circ B = (A \ominus B) \oplus B$	Smoothes contours, breaks narrow isthmuses, and eliminates small islands and sharp peaks. (I)
Closing	$A \bullet B = (A \oplus B) \ominus B$	Smoothes contours, fuses narrow breaks and long thin gulfs, and eliminates small holes. (I)
Hit-or-miss transform	$A \circledast B = (A \ominus B_1) \cap (A^c \ominus B_2) = (A \ominus B_1) - (A \oplus \hat{B}_2)$	The set of points (coordinates) at which, simultaneously, B_1 found a match ("hit") in A and B_2 found a match in A^c.
Boundary extraction	$\beta(A) = A - (A \ominus B)$	Set of points on the boundary of set A. (I)
Region filling	$X_k = (X_{k-1} \oplus B) \cap A^c$; $X_0 = p$ and $k = 1, 2, 3, \ldots$	Fills a region in A, given a point p in the region. (II)
Connected components	$X_k = (X_{k-1} \oplus B) \cap A$; $X_0 = p$ and $k = 1, 2, 3, \ldots$	Finds a connected component Y in A, given a point p in Y. (I)
Convex hull	$X_k^i = (X_{k-1}^i \circledast B^i) \cup A$; $i = 1, 2, 3, 4$; $k = 1, 2, 3, \ldots$; $X_0^i = A$; and $D^i = X_{\text{conv}}^i$.	Finds the convex hull $C(A)$ of set A, where "conv" indicates convergence in the sense that $X_k^i = X_{k-1}^i$. (III)

Operation	Equation	Comments (The Roman numerals refer to the structuring elements shown in Fig. 9.26).
Thinning	$A \otimes B = A - (A \circledast B)$ $= A \cap (A \circledast B)^c$ $A \otimes \{B\} =$ $((\dots((A \otimes B^1) \otimes B^2)\dots) \otimes B^n)$ $\{B\} = \{B^1, B^2, B^3, \dots, B^n\}$	Thins set A. The first two equations give the basic definition of thinning. The last two equations denote thinning by a sequence of structuring elements. This method is normally used in practice. (IV)
Thickening	$A \odot B = A \cup (A \circledast B)$ $A \odot \{B\} =$ $((\dots(A \odot B^1) \odot B^2 \dots) \odot B^n)$	Thickens set A. (See preceding comments on sequences of structuring elements.) Uses IV with 0's and 1's reversed.
Skeletons	$S(A) = \bigcup_{k=0} S_k(A)$ $S_k(A) = \bigcup_{k=0}^{K} \{(A \ominus kB)$ $- [(A \ominus kB) \circ B]\}$ Reconstruction of A: $A = \bigcup_{k=0}^{K} (S_k(A) \oplus kB)$	Finds the skeleton $S(A)$ of set A. The last equation indicates that A can be reconstructed from its skeleton subsets $S_k(A)$. In all three equations, K is the value of the iterative step after which the set A erodes to the empty set. The notation $(A \ominus kB)$ denotes the kth iteration of successive erosion of A by B. (I)
Pruning	$X_1 = A \otimes \{B\}$ $X_2 = \bigcup_{k=1}^{8} (X_1 \circledast B^k)$ $X_3 = (X_2 \oplus H) \cap A$ $X_4 = X_1 \cup X_3$	X_4 is the result of pruning set A. The number of times that the first equation is applied to obtain X_1 must be specified. Structuring elements V are used for the first two equations. In the third equation H denotes structuring element I.

re Extraction

When you process an image, you may want to obtain information about how certain features of the image change. For example, when you perform dilation, you may want to determine how many pixels are changed from `off` to `on` by the operation, or to see if the number of objects in the image changes. This section describes two functions, `bwarea` and `bweuler`, that return two common measures of binary images: the image area and the Euler number.

Image Area

The `bwarea` function returns the area of a binary image. The area is a measure of the size of the foreground of the image. Roughly speaking, the area is the number of `on` pixels in the image.

`bwarea` does not simply count the number of `on` pixels, however. Rather, `bwarea` weights different pixel patterns unequally when computing the area. This weighting compensates for the distortion that is inherent in representing a continuous image with discrete pixels. For example, a diagonal line of 50 pixels is longer than a horizontal line of 50 pixels. As a result of the weighting `bwarea` uses, the horizontal line has area of 50, but the diagonal line has area of 62.5.

This example uses `bwarea` to determine how much the area of the circuit image increases after a dilation operation:

```
BW1 = imread('circbw.tif');
SE = ones(5);
BW2 = dilate(BW1,SE);
increase = (bwarea(BW2) - bwarea(BW1))/bwarea(BW1);

increase =

    0.3456
```

The dilation increases the area by about 35 percent.

For more information about the weighting pattern used by `bwarea`, see the reference entry for `bwarea` in Chapter 11.

As an example, we consider how the ends of thin lines may be located. To achieve this we need eight masks of the following types:

$D_1 =$

0	0	0
0	1	1
0	0	0

$D_2 =$

0	0	1
0	1	0
0	0	0

...

It is of interest that a much simpler algorithm could be used in this case to count the number of neighbours of each pixel and check whether the number is unity. (It is worth bearing in mind that alternative approaches can sometimes be highly beneficial in cutting down on excessive computational loads: on the other hand, morphological processing is of value in this general context because of the extreme simplicity of the masks being used for analysing images.) Further examples of template matching will appear in the following section.

B.4 Connectedness-based analysis of images

In this appendix there is only space for one illustration of the application of morphology to connectedness-based analysis. We have chosen the case of skeletonization and thinning.

B.4.1 Skeletons and thinning

A widely used thinning algorithm uses eight D-type hit-and-miss transform masks (see Section B.3.5):

$T_1 =$

0	×	1
0	1	1
0	×	1

$T_2 =$

×	1	1
0	1	1
0	0	×

$T_3 =$

1	1	1
×	1	×
0	0	0

$T_4 =$

1	1	×
1	1	0
×	0	0

$T_5 =$

1	×	0
1	1	0
1	×	0

$T_6 =$

×	0	0
1	1	0
1	1	×

$T_7 =$

0	0	0
×	1	×
1	1	1

$T_8 =$

0	0	×
0	1	1
×	1	1

Each of these masks recognizes a particular situation where the central pixel is unnecessary for maintaining connectedness and can be eliminated (this assumes that the foreground is regarded as 8-connected). Thus all pixels can be subjected to the T_1 mask and eliminated if the mask matches the neighbourhood; then all pixels can be subjected to the T_2 mask and eliminated if appropriate; and so on until upon repeating with all the masks there is no change in the output image (see Chapter 6 for a full description of the process).

However, although this particular procedure is guaranteed not to disconnect objects, it does not terminate the skeleton location procedure, as there remain a number of locations where pixels can be removed without disconnecting the skeleton. The advantage of the masks given above is that they are very simple and produce fast convergence towards the skeleton. Their disadvantage is that a final stage of processing is required to complete the process (for example, one of the crossing number based methods of Chapter 6 may be used for this purpose, but we do not pursue the matter further here).

P_3	P_2	P_9
P_4	P_1	P_8
P_5	P_6	P_7

(a) Labeling point P_1 and its neighbors.

1	1	0
1	P_1	1
0	0	0

(i)

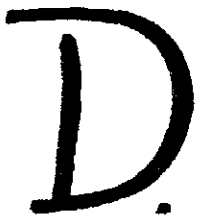
0	0	0
1	P_1	0
0	0	0

(ii)

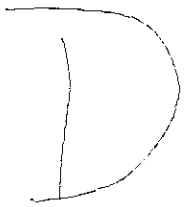
1	0	1
0	P_1	0
1	1	1

(iii)

(b) Examples where P_1 is not deletable ($P_1 = 1$).
(i) Deleting P_1 will tend to split the region;
(ii) deleting P_1 will shorten arc ends;
(iii) $2 \leq NZ(P_1) \leq 6$ but P_1 is not deletable.



(i)



(ii)

(c) Example of thinning.
(i) Original;
(ii) thinned.

Figure 9.32 A thinning algorithm.

tained is not influenced by small contour inflections that may be present on the initial contour. The basic approach [42] is to *delete* from the object **X** simple border points that have more than one neighbor in **X** and whose deletion does not locally disconnect **X**. Here a *connected region* is defined as one in which any two points in the region can be connected by a curve that lies entirely in the region. In this way, endpoints of thin arcs are not deleted. A simple algorithm that yields connected arcs while being insensitive to contour noise is as follows [43].

Referring to Figure 9.32a, let $Z0(P_1)$ be the number of zero to nonzero transitions in the ordered set $P_2, P_3, P_4, \ldots, P_9, P_2$. Let $NZ(P_1)$ be the number of nonzero neighbors of P_1. Then P_1 is deleted if (Fig. 9.32b)

$$\left.\begin{aligned} & 2 \leq NZ(P_1) \leq 6 \\ \text{and} \quad & Z0(P_1) = 1 \\ \text{and} \quad & P_2 \cdot P_4 \cdot P_8 = 0 \quad \text{or} \quad Z0(P_2) \neq 1 \\ \text{and} \quad & P_2 \cdot P_4 \cdot P_6 = 0 \quad \text{or} \quad Z0(P_4) \neq 1 \end{aligned}\right\} \tag{9.86}$$

The procedure is repeated until no further changes occur in the image. Figure 9.32c gives an example of applying this algorithm. Note that at each location such as P_1 we end up examining pixels from a 5×5 neighborhood.

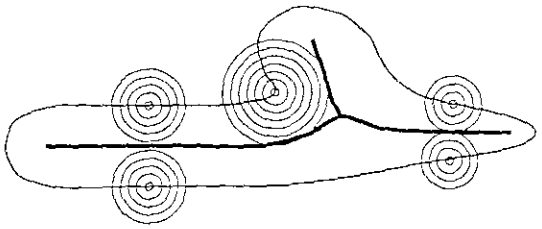


Figure 11.22: *Skeleton as points where two or more wavefronts of grassfire meet.*

A more formal definition of skeleton is based on the concept of maximal ball. A **ball** $B(p,r)$ with center p and radius r, $r \geq 0$, is the set of points with distances d from the center less than or equal to r.

The ball B included in a set X is said to be **maximal** if and only if there is no larger ball included in X that contains B, i.e., each ball B', $B \subseteq B' \subseteq X \Longrightarrow B' = B$. Balls and maximal balls are illustrated in Figure 11.23.

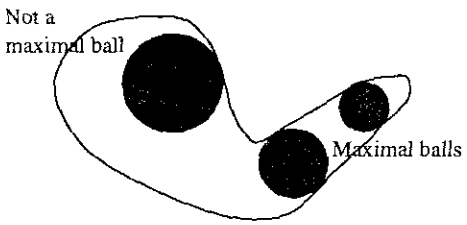


Figure 11.23: *Ball and maximal balls in Euclidean plane.*

The distance metric d that is used depends on the grid and definition of connectivity. Unit balls in a plane (i.e., unit disks) are shown in Figure 11.24.

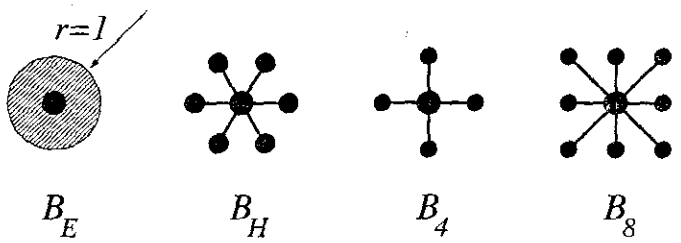


Figure 11.24: *Unit-size disk for different distances, from left side: Euclidean distance, 6-, 4-, and 8-connectivity, respectively.*

The plane $\mathcal{R}^2$ with the usual Euclidean distance gives the ball B_E. Three distances and balls are often defined in the discrete plane $\mathcal{Z}^2$. If a hexagonal grid and 6-connectivity is used, the hexagonal ball B_H is obtained. If the support is a square grid, two unit balls are possible: B_4 for 4-connectivity and B_8 for 8-connectivity.

The **skeleton by maximal balls** $S(X)$ of a set $X \subset \mathcal{Z}^2$ is the set of centers p of maximal balls:

$$S(X) = \{p \in X : \exists r \geq 0,\ B(p,r) \text{ is a maximal ball of } X\}$$

This definition of skeleton has an intuitive meaning in the Euclidean plane. The skeleton of a disk reduces to its center, the skeleton of a stripe with rounded endings is a unit thickness

line at its center, etc.

Figure 11.25 shows several objects together with their skeletons—a rectangle, two touching balls, and a ring. The properties of the (Euclidean) skeleton can be seen here—in particular, the skeleton of two adjacent circles consists of two distinct points instead of a straight line joining these two points, as might be intuitively expected.

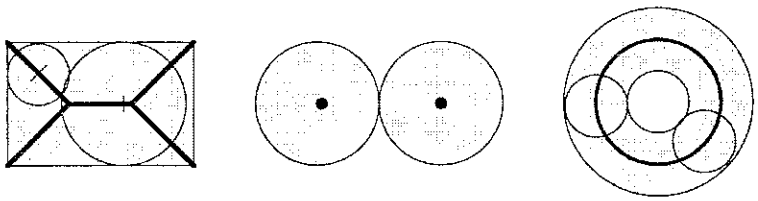


Figure 11.25: *Skeletons of rectangle, two touching balls, and a ring.*

The skeleton by maximal balls has two unfortunate properties in practical applications. First, it does not necessarily preserve the homotopy (connectivity) of the original set; and second, some of the skeleton lines may be wider than one pixel in the discrete plane. We shall see later that the skeleton is often substituted by sequential homotopic thinning that does not have these two properties.

Dilation can be used in any of the three discrete connectivities to create balls of varying radii. Let nB be the ball of radius n:

$$nB = B \oplus B \oplus \ldots \oplus B \tag{11.44}$$

The skeleton by maximal balls can be obtained as the union of the residues of opening of the set X at all scales [Serra 82]:

$$S(X) = \bigcup_{n=0}^{\infty} [(X \ominus nB) \setminus (X \ominus nB) \circ B] \tag{11.45}$$

The trouble with this is that the resulting skeleton is completely disconnected and this property is not useful in many applications. Thus **homotopic skeletons** that preserve connectivity are often preferred. We present an intuitive homotopic skeletonization algorithm based on consecutive erosions (thinning) in Section 11.5.3.

11.5.3 Thinning, thickening, and homotopic skeleton

One application of the hit-or-miss transformation (Section 11.3.3) is **thinning** and **thickening** of point sets. For an image X and a composite structuring element $B = (B_1, B_2)$ (notice that B here is not a ball), *thinning* is defined as

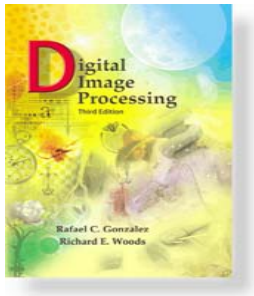
$$X \oslash B = X \setminus (X \otimes B) \tag{11.46}$$

and *thickening* is defined as

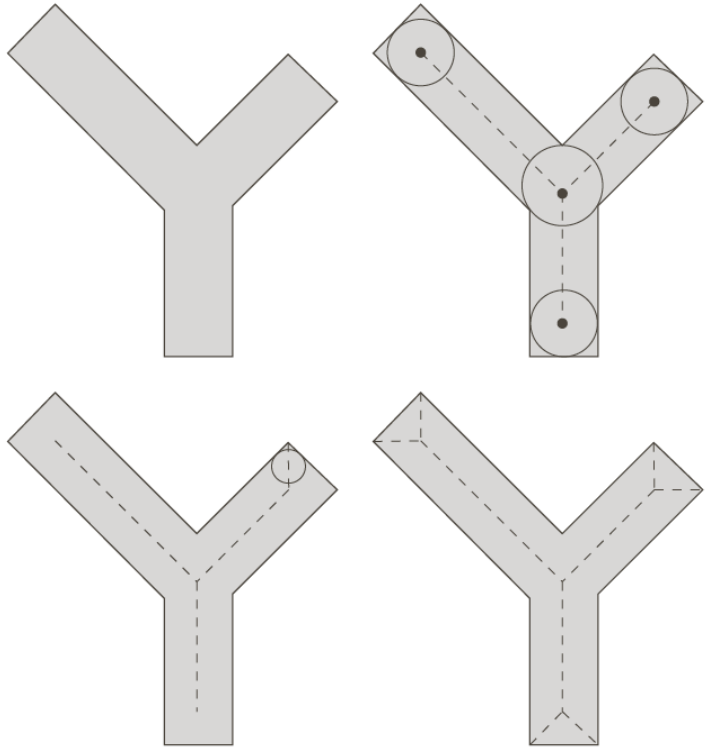
$$X \odot B = X \cup (X \otimes B) \tag{11.47}$$

When thinning, a part of the boundary of the object is subtracted from it by the set difference operation. When thickening, a part of the boundary of the background is added to the object. Thinning and thickening are dual transformations:

$$(X \odot B)^c = X^c \bigcirc B \quad B = (B_2, B_1) \tag{11.48}$$



Chapter 9
Morphological Image Processing



a b
c d

FIGURE 9.23
(a) Set A.
(b) Various positions of maximum disks with centers on the skeleton of A.
(c) Another maximum disk on a different segment of the skeleton of A.
(d) Complete skeleton.

Thinning reduces a curvilinear object to a single-pixel-wide line, showing its topology graphically. In Figure 18–23, thinning a group of chromosomes, some of which are touching, produces a graph with one segment for each chromosome. This can be used as the basis for a separation algorithm for objects that are in contact.

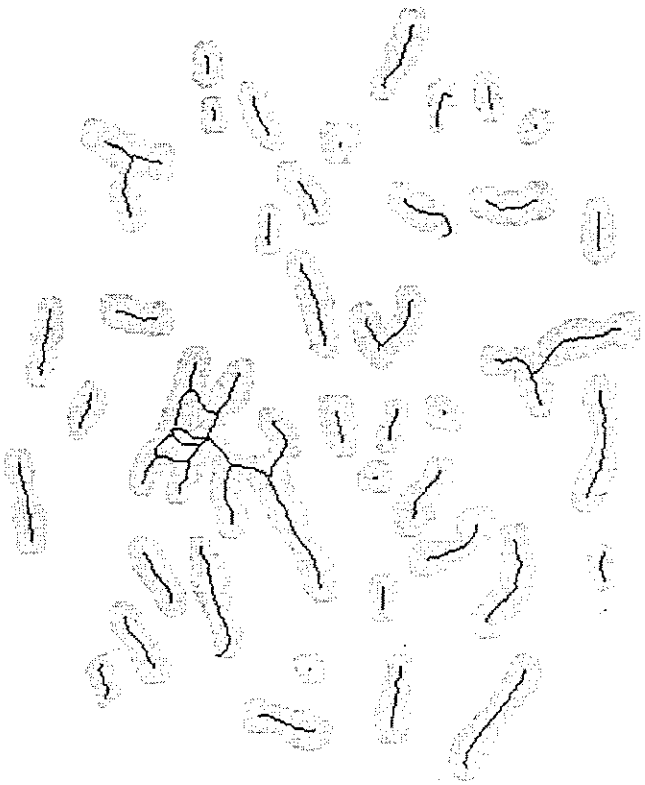


Figure 18–23 Thinning

18.7.4.3 Skeletonization

An operation related to thinning is *skeletonization,* also known as the *medial axis transform* or the *grass-fire technique* [46–50]. The medial axis is the locus of the centers of all the circles that are tangent to the boundary of the object at two or more disjoint points. Skeletonization is seldom implemented, however, by actually fitting circles inside the object.

Conceptually, the medial axis can be thought of as being formed in the following way. Imagine that a patch of grass, in the shape of the object, is set on fire all around the periphery at once. As the fire progresses inward, the locus of points where advancing fire lines meet is the medial axis.

Skeletonization can be implemented with a two-pass conditional erosion, as with thinning. The rule for deleting pixels, however, is slightly different. Figure 18–24 compares thinning with skeletonization. The primary difference is that the medial axis skeleton extends to the boundary at corners, while the skeleton obtained by thinning does not.

18.7.4.4 Pruning

Often, the thinning or skeletonizing process will leave *spurs* on the resulting figure. These are short branches having an endpoint located within three or so pixels of an intersection.

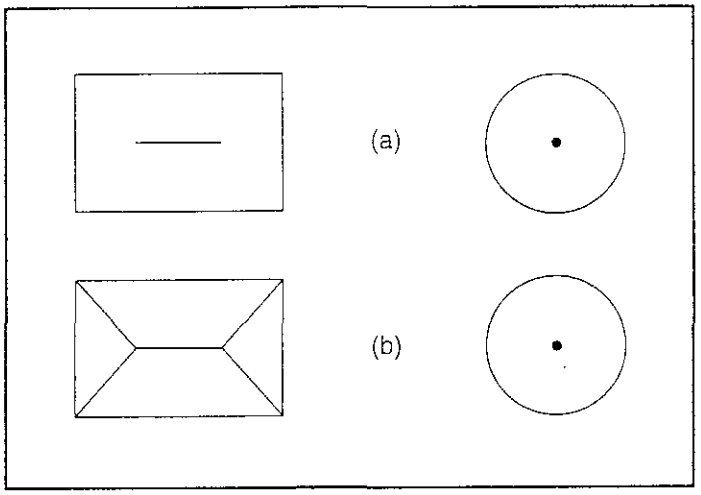


Figure 18–24 Thinning and skeletonization: (a) thinned skeleton; (b) medial axis

Spurs result from single-pixel-sized undulations in the boundary that give rise to a short branch. They can be removed by a series of three-by-three operations that remove endpoints (thereby shortening all the branches), followed by reconstruction of the branches that still exist. A three-pixel spur, for example, disappears after three iterations of removing endpoints. Not having an endpoint to grow back from, the spur is not reconstructed.

18.7.4.5 Thickening

Dilation can be implemented so as not to merge nearby objects. This can be done in two passes, similarly to thinning. An alternative is to complement the image and use the thinning operation on the background. In fact, each of the variants of erosion has a companion dilation-type operation obtained when it is run on a complemented image.

Some segmentation techniques tend to fit rather tight boundaries to objects so as to avoid erroneously merging them. Often, the best boundary for isolating objects is too tight for subsequent measurement. Thickening can correct this by enlarging the boundaries without merging separate objects.

18.7.4.6 An example

Figure 18–25 illustrates how morphological operations can be concatenated to implement a complex process. Here an image of a printed circuit board is analyzed to locate a break point in the traces.

18.7.5 The Distance Transformation

Another related operation that can be performed on binary images is the distance transformation. It results, however, not in another binary image, but in a gray-level image. The gray level at each pixel is the distance from that pixel to the nearest background pixel.

An approximate distance transformation can be computed by an erosion-like operation wherein, on each pass, pixels are labeled with the iteration number rather than being eliminated from the object. The so-called *chamfer algorithm* computes a distance transformation in only two passes over the image [51,52].

10.5 Morphological Reconstruction

See Sections 11.4.2 and 11.4.3 for additional applications of morphological reconstruction.

Reconstruction is a morphological transformation involving two images and a structuring element (instead of a single image and structuring element). One image, the *marker*, is the starting point for the transformation. The other image, the *mask*, constrains the transformation. The structuring element used defines connectivity. In this section we use 8-connectivity (the default), which implies that B in the following discussion is a 3×3 matrix of 1s, with the center defined at coordinates (2, 2). In this section we deal with binary images; gray-scale reconstruction is discussed in Section 10.6.3.

This definition of reconstruction is based on dilation. It is possible to define a similar operation using erosion. The results are duals of each other with respect to set complementation. These concepts are developed in detail in Gonzalez and Woods [2008].

If G is the mask and F is the marker, the reconstruction of G from F, denoted $R_G(F)$, is defined by the following iterative procedure:

1. Initialize h_1 to be the marker image, F.
2. Create the structuring element: `B = ones(3)`.
3. Repeat:

$$h_{k+1} = (h_k \oplus B) \cap G$$

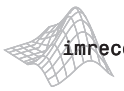
   until $h_{k+1} = h_k$.

4. $R_G(F) = h_{k+1}$.

Marker F must be a subset of G:

$$F \subseteq G$$

Figure 10.21 illustrates the preceding iterative procedure. Although this iterative formulation is useful conceptually, much faster computational algorithms exist. Toolbox function `imreconstruct` uses the "fast hybrid reconstruction" algorithm described in Vincent [1993]. The calling syntax for `imreconstruct` is




```
out = imreconstruct(marker, mask)
```

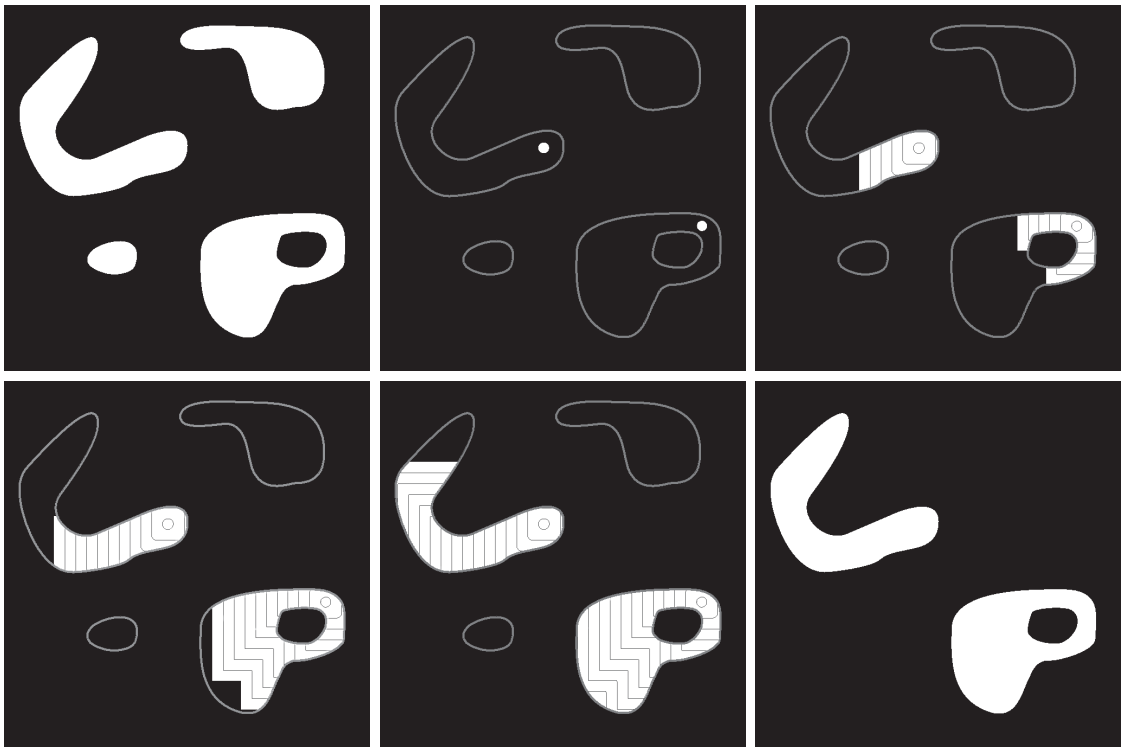
where `marker` and `mask` are as defined at the beginning of this section.

10.5.1 Opening by Reconstruction

In morphological opening, erosion typically removes small objects, and the subsequent dilation tends to restore the shape of the objects that remain. However, the accuracy of this restoration depends on the similarity between the shapes and the structuring element. The method discussed in this section, *opening by reconstruction*, restores the original shapes of the objects that remain after erosion. The opening by reconstruction of an image G using structuring element B, is defined as $R_G(G \ominus B)$.

EXAMPLE 10.8: Opening by reconstruction.

■ A comparison between opening and opening by reconstruction for an image containing text is shown in Fig. 10.22. In this example, we are interested in extracting from Fig. 10.22(a) the characters that contain long vertical strokes.



a b c
d e f

FIGURE 10.21 Morphological reconstruction. (a) Original image (the mask). (b) Marker image. (c)–(e) Intermediate result after 100, 200, and 300 iterations, respectively. (f) Final result. (The outlines of the objects in the mask image are superimposed on (b)–(e) as visual references.)

Because both opening and opening by reconstruction have erosion in common, we perform that step first, using a thin, vertical structuring element of length proportional to the height of the characters:

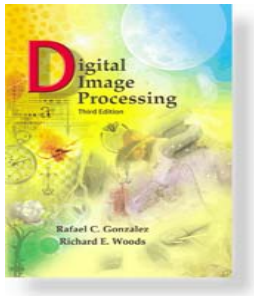
```
>> f = imread('book_text_bw.tif');
>> fe = imerode(f, ones(51, 1));
```

Figure 10.22(b) shows the result. The opening, shown in Fig. 10.22(c), is computed using `imopen`:

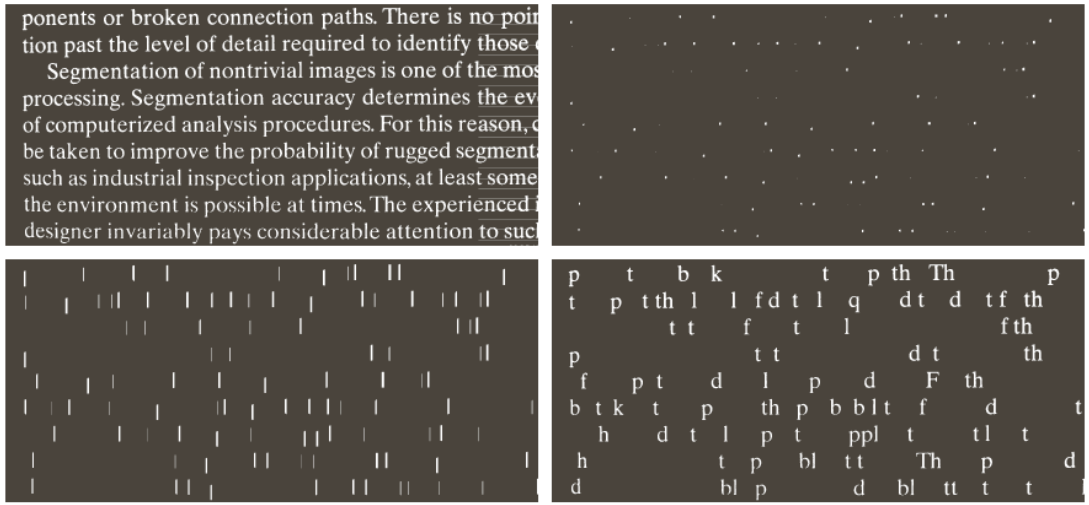
```
>> fo = imopen(f,ones(51, 1));
```

Note that the vertical strokes were restored, but not the rest of the characters containing the strokes. Finally, we obtain the reconstruction:

```
>> fobr = imreconstruct(fe, f);
```

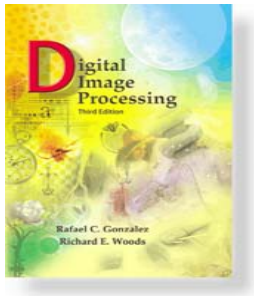


Chapter 9
Morphological Image Processing

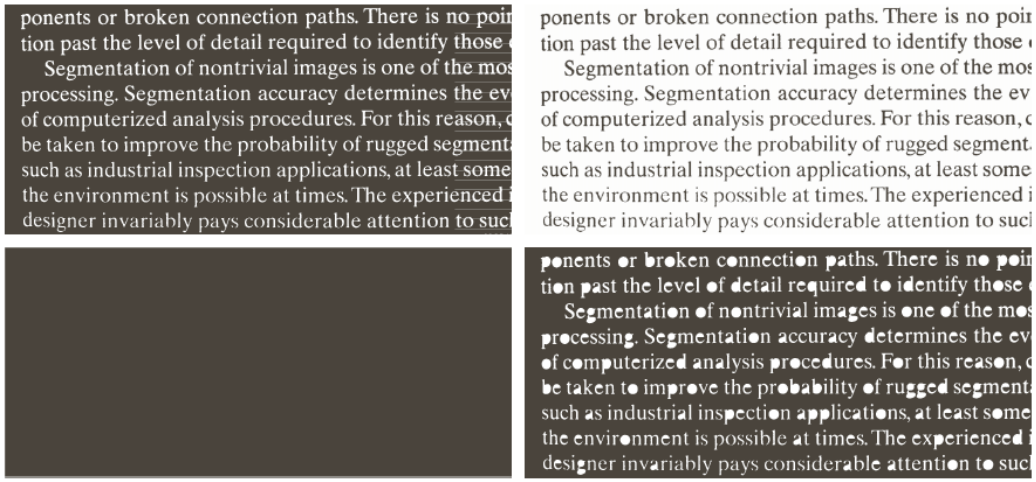


a b
c d

FIGURE 9.29 (a) Text image of size 918 × 2018 pixels. The approximate average height of the tall characters is 50 pixels. (b) Erosion of (a) with a structuring element of size 51 × 1 pixels. (c) Opening of (a) with the same structuring element, shown for reference. (d) Result of opening by reconstruction.



Chapter 9
Morphological Image Processing



a b
c d

FIGURE 9.31
(a) Text image of size 918 × 2018 pixels. (b) Complement of (a) for use as a mask image. (c) Marker image. (d) Result of hole-filling using Eq. (9.5-29).

a b
c d
e f
g

FIGURE 10.22 Morphological reconstruction: (a) Original image. (b) Image eroded with vertical line; (c) opened with a vertical line; and (d) opened by reconstruction with a vertical line. (e) Holes filled. (f) Characters touching the border (see right border). (g) Border characters removed.

The result in Fig. 10.22(d) shows that characters containing long vertical strokes were restored exactly; all other characters were removed. The remaining parts of Fig. 10.22 are explained in the following two sections. ■

10.5.2 Filling Holes

Morphological reconstruction has a broad spectrum of practical applications, each characterized by the selection of the marker and mask images. For example, let I denote a binary image and suppose that we choose the marker image, F, to be 0 everywhere except on the image border, where it is set to $1 - I$:

$$F(x, y) = \begin{cases} 1 - I(x, y) & \text{if } (x, y) \text{ is on the border of } I \\ 0 & \text{otherwise} \end{cases}$$

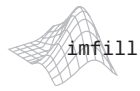
Then,

$$H = \left[R_{I^c}(F) \right]^c$$

is a binary image equal to I with all holes filled, as illustrated in Fig. 10.22(e).

Toolbox function `imfill` performs this computation automatically when the optional argument `'holes'` is used:

```
g = imfill(f, 'holes')
```



This function is discussed in more detail in Section 12.1.2.

10.5.3 Clearing Border Objects

Another useful application of reconstruction is removing objects that touch the border of an image. Again, the key task is to select the appropriate marker to achieve the desired effect. Suppose we define the marker image, F, as

$$F(x,y) = \begin{cases} I(x,y) & \text{if } (x,y) \text{ is on the border of } I \\ 0 & \text{otherwise} \end{cases}$$

where I is the original image. Then, using I as the mask image, the reconstruction

$$H = R_I(F)$$

yields and image, H, that contains only the objects touching the border, as Fig. 10.22(f) shows. The difference, $1 - H$, shown in Fig. 10.22(g), contains only the objects from the original image that do not touch the border. Toolbox function `imclearborder` performs this entire procedure automatically. Its syntax is

```
g = imclearborder(f, conn)
```

where `f` is the input image and `g` is the result. The value of `conn` can be either 4 or 8 (the default). This function suppresses structures that are lighter than their surroundings and that are connected to the image border.

5.2 THE MEDIAL-AXIS TRANSFORM

Possibly the first definition of a skeleton is that of Blum (1967) in defining the *medial axis function* (MAF). The MAF treats all boundary pixels as point sources of a wave front. Each of these pixels excites its neighbors with a delay time proportional to distance, so that they, too, become part of the wave front. The wave passes through each point only once, and when two waves meet they cancel each other, producing a *corner*. The *medial axis* (MA) is the locus of the corners, and forms the skeleton (Blum says *line of symmetry*) of the object. The MAF uses both time and space information, and can be inverted to give back the original picture. It is possible to implement this directly, but it is difficult: What is needed is to convert the continuous transform to a discrete one. This involves various approximations involving the distance function on a discrete grid. This allows the MAF to be applied to a raster image, for which the medial axis is not defined.

One way to find the medial axis is to use the boundary of the object. For any point P in the object, locate the closest point on the boundary. If there is more than one boundary point at the minimum distance, then P is on the medial axis. The set of all such points is the medial axis of the object. Unfortunately this must

---

be done at a very high resolution, or Euclidean distances will not be equal when they should be, and skeletal pixels will be missed.

An approximation to the medial axis on a sampled grid is more easily obtained in two steps. First, compute the distance from each object pixel to the nearest boundary pixel. This involves computing the distance to all boundary pixels. Next, the Laplacian of the distance image is calculated, and pixels having large values are thought to belong to the medial axis.

The way that distance is measured has an impact on the result, as seen in Figure 5.1. The medial axis was found for a T-shaped object using Euclidean distance, *4-distance*, and *8-distance*. 4-distance between pixels A and B is defined to be the minimum number of horizontal and vertical moves needed to get from A to B. 8-distance is the minimum number of pixel moves, in any of the standard eight directions, needed to get from A to B. There are clear differences in the medial axis depending on which way distance is calculated, but any of them could be used as a skeleton.

The skeleton of the T produced by the medial axis does not have the same shape as the T, nor does it need it. The main concern is whether the skeleton characterizes the basic shape of the object somehow. On the other hand, a simple example

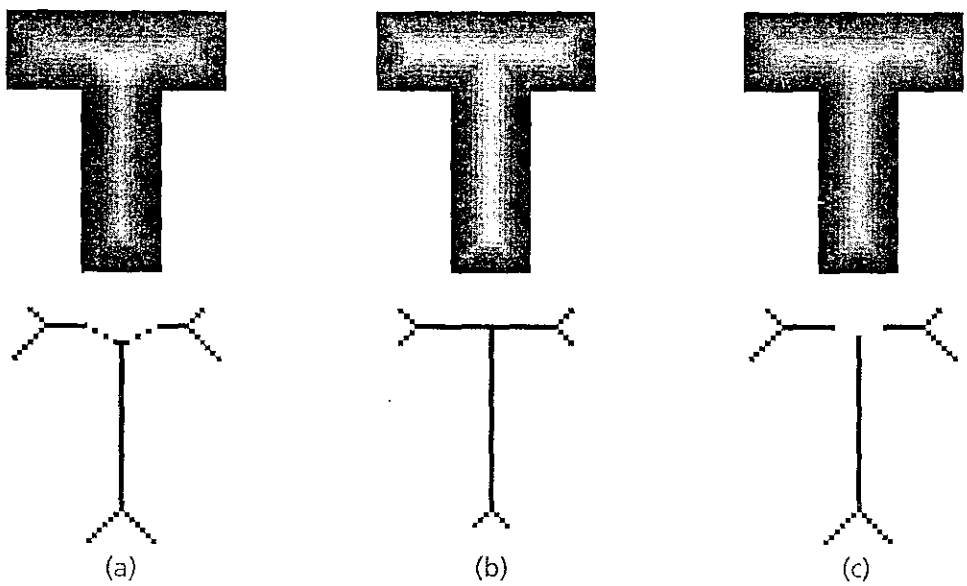


Figure 5.1 The effect of the distance function on the medial axis. (a) Medial axis (above) and skeleton (below) of the T-shaped object, using 4-distance. (b) Medial axis and skeleton computed using 8-distance. (c) Computed using Euclidean distance.

9.5.2 Region filling

• Begin with a point p inside the boundary, and then fill the entire region with 1's

• All non-boundary (background) points are labeled 0

• Assign a value of 1 to p to begin...

• The following procedure fills the region with 1's,

$$X_k = (X_{k-1} \oplus B) \cap A^c, \quad k = 1, 2, 3, \ldots,$$

where $X_0 = p$, and B is the symmetric structuring element in figure 9.15 (c)

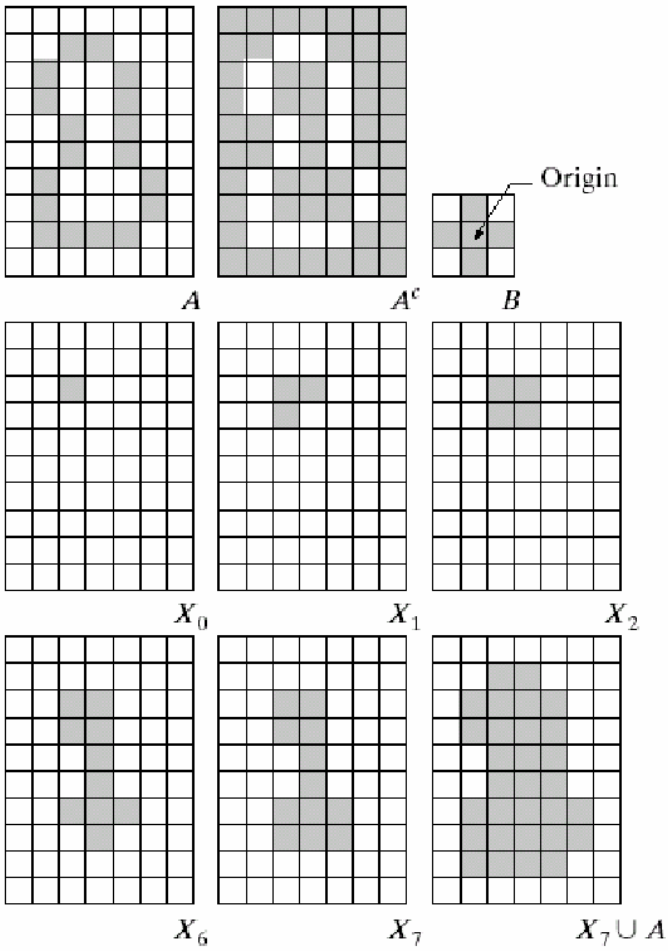
• The algorithm terminates at iteration step k if $X_k = X_{k-1}$

• The set union of X_k and A contains the filled set and its boundary

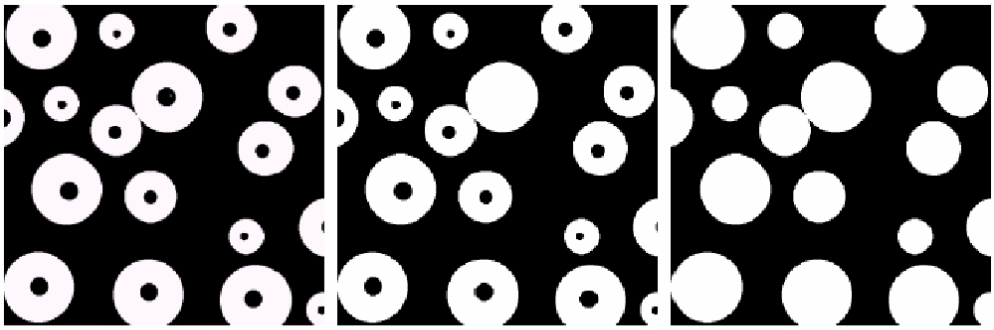
Note that the intersection at each step with A^c limits the dilation result to inside the region of interest

a b c
d e f
g h i

FIGURE 9.15
Region filling.
(a) Set A.
(b) Complement of A.
(c) Structuring element B.
(d) Initial point inside the boundary.
(e)–(h) Various steps of Eq. (9.5-2).
(i) Final result [union of (a) and (h)].



Example 9.6: Morphological region filling



a b c

FIGURE 9.16 (a) Binary image (the white dot inside one of the regions is the starting point for the region-filling algorithm). (b) Result of filling that region (c) Result of filling all regions.

9.5.3 Extraction of connected components

Let Y represent a connected component contained in a set A and assume that a point p of Y is known. Then the following iterative expression yields all the elements of Y:

$$X_k = (X_{k-1} \oplus B) \cap A \quad k = 1, 2, 3, \ldots,$$

where $X_0 = p$, and B is a suitable structuring element. If $X_k = X_{k-1}$, the algorithm has converged and we let $Y = X_k$.

This algorithm is applicable to any finite number of sets of connected components contained in A, assuming that a point is known in **each** connected component

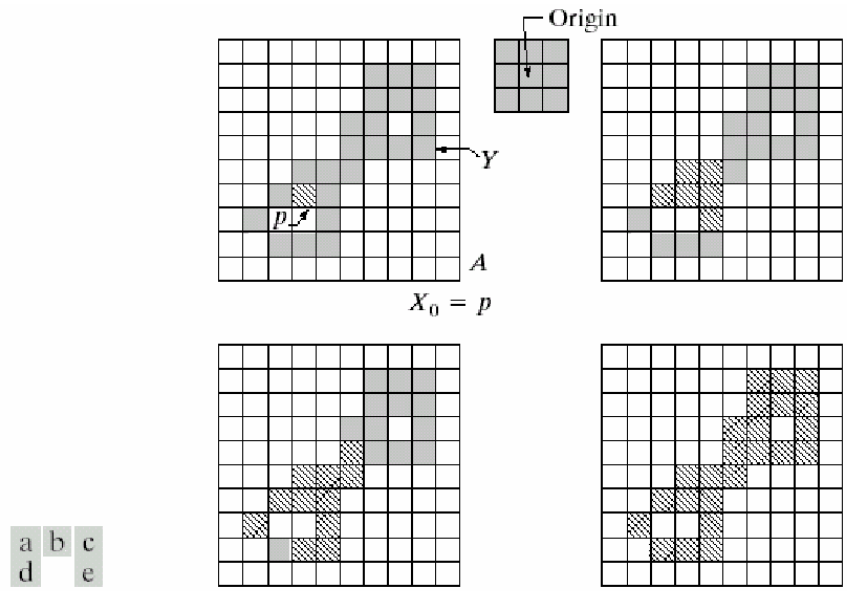
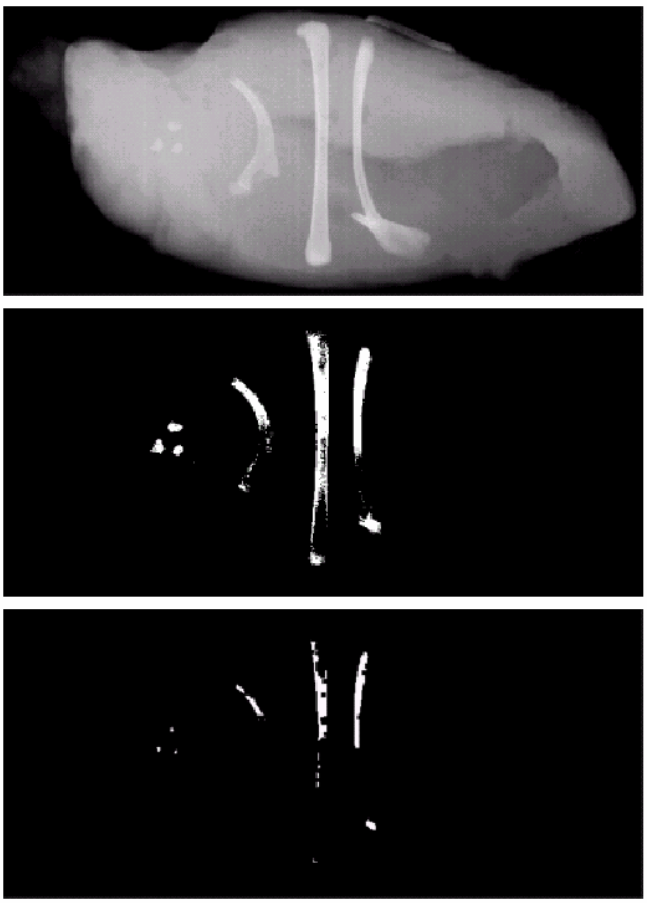


FIGURE 9.17 (a) Set A showing initial point p (all shaded points are valued 1, but are shown different from p to indicate that they have not yet been found by the algorithm). (b) Structuring element. (c) Result of first iterative step. (d) Result of second step. (e) Final result.

Example 9.7:

a
b
c d

FIGURE 9.18
(a) X-ray image of chicken filet with bone fragments. (b) Thresholded image. (c) Image eroded with a 5×5 structuring element of 1's. (d) Number of pixels in the connected components of (c). (Image courtesy of NTB Elektronische Geraete GmbH, Diepholz, Germany, www.ntbxray.com.)



Connected component	No. of pixels in connected comp
01	11
02	9
03	9
04	39
05	133
06	1
07	1
08	743
09	7
10	11
11	11
12	9
13	9
14	674
15	85

9.5.4 Convex hull

Morphological algorithm for obtaining the convex hull, $C(A)$, of a set A...

Let B_1, B_2, B_3 and B_4 represent the four structuring elements in figure 9.19 (a), and then implement the equation ...

$$X_k^i = (X_{k-1} \circledast B^i) \cup A, \ i = 1, 2, 3, 4, \ k = 1, 2, \ldots, \ X_0^i = A$$

Now let $D^i = X_{\text{conv}}^i$, where "conv" indicates convergence in the sense that $X_k^i = X_{k-1}^i$. Then the convex hull of A is

$$C(A) = \cup_{i=1}^4 D^i$$

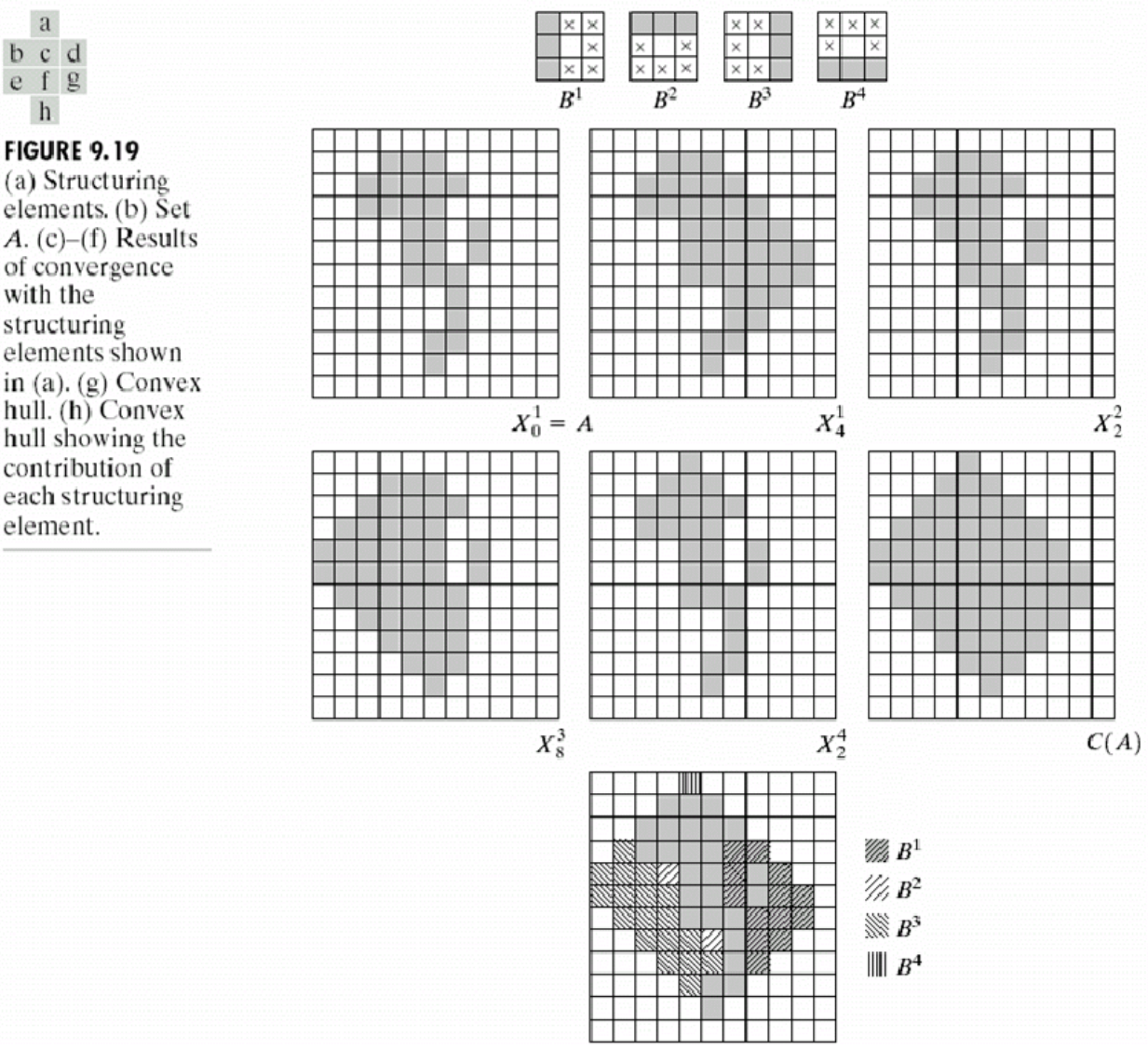


FIGURE 9.19 (a) Structuring elements. (b) Set A. (c)–(f) Results of convergence with the structuring elements shown in (a). (g) Convex hull. (h) Convex hull showing the contribution of each structuring element.

Shortcoming of above algorithm: convex hull can grow beyond the minimum dimensions required to guarantee convexity

Possible solution: Limit growth so that it does not extend past the vertical and horizontal dimensions of the original set of points

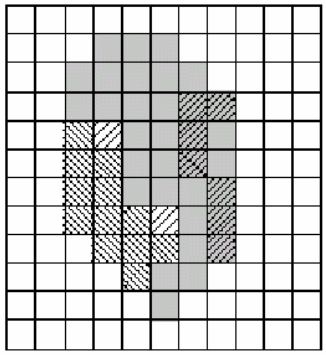


FIGURE 9.20 Result of limiting growth of convex hull algorithm to the maximum dimensions of the original set of points along the vertical and horizontal directions.

Boundaries of greater complexity can be used to limit growth even further in images with more detail

9.5.5 Thinning

The thinning of a set A by a structuring element B:

$$A \otimes B = A - (A \circledast B) = A \cap (A \circledast B)^c$$

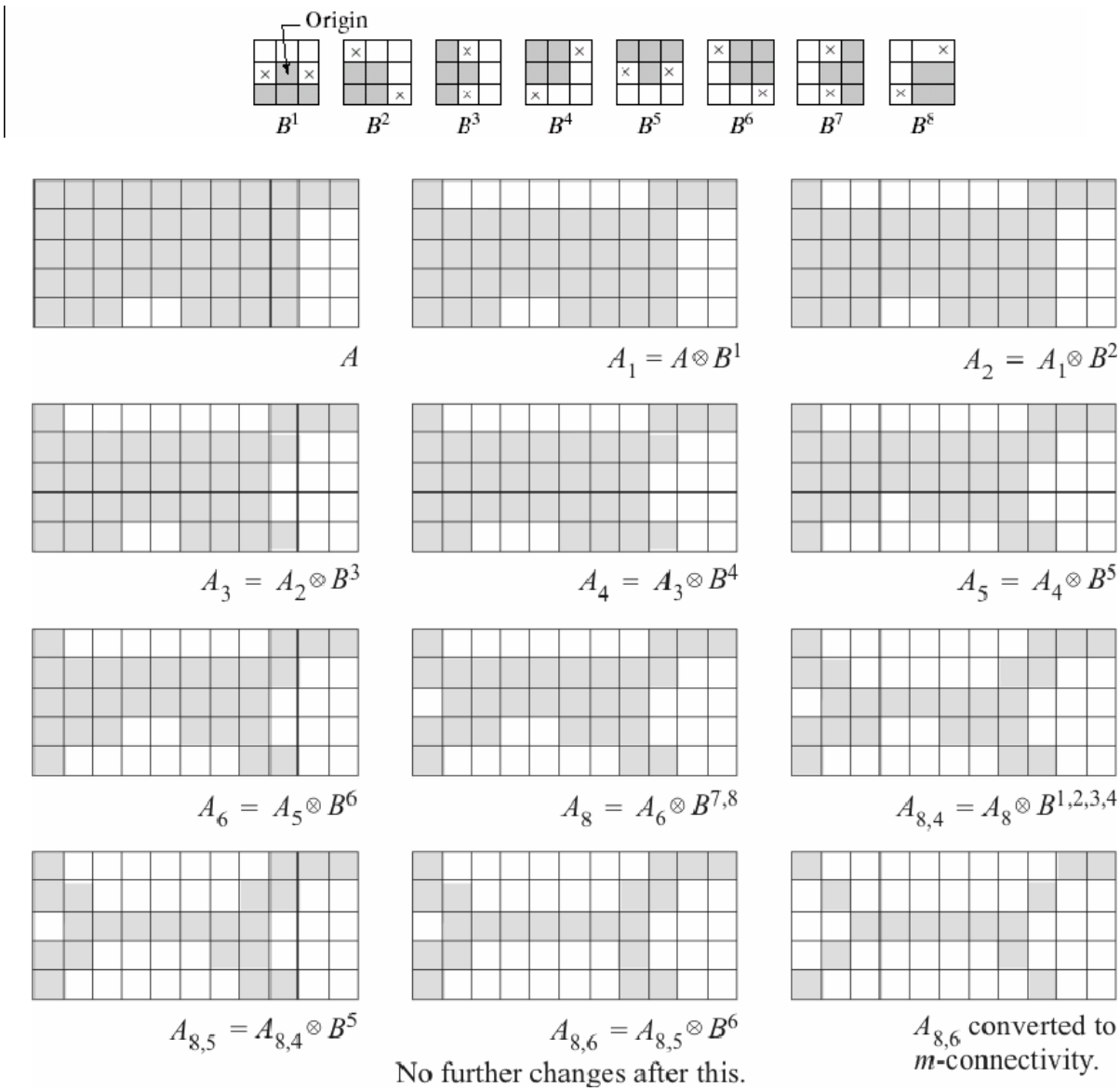
Symmetric thinning: sequence of structuring elements,

$$\{B\} = \left\{B^1, B^2, B^3, \ldots, B^n\right\},$$

where B^i is a rotated version of B^{i-1}

$$A \otimes \{B\} = ((\ldots((A \otimes B^1) \otimes B^2)\ldots) \otimes B^n)$$

Illustration: Note that figure 9.21 (in the handbook) has many errors — this one is correct...



9.5.6 Thickening

Thickening is the morphological dual of thinning and is defined by

$$A \odot B = A \cup (A \circledast B),$$

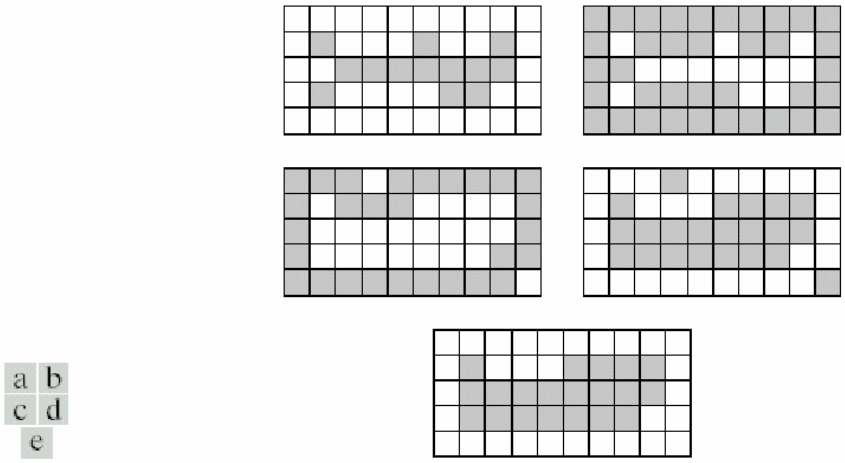
where B is a structuring element

Similar to thinning...

$$A \odot \{B\} = ((\ldots((A \odot B^1) \odot B^2)\ldots) \odot B^n)$$

Structuring elements for thickening are similar to those of figure 9.21 (a), but with all 1's and 0's interchanged

A separate algorithm for thickening is seldom used in practice — we thin the background instead, and then complement the result



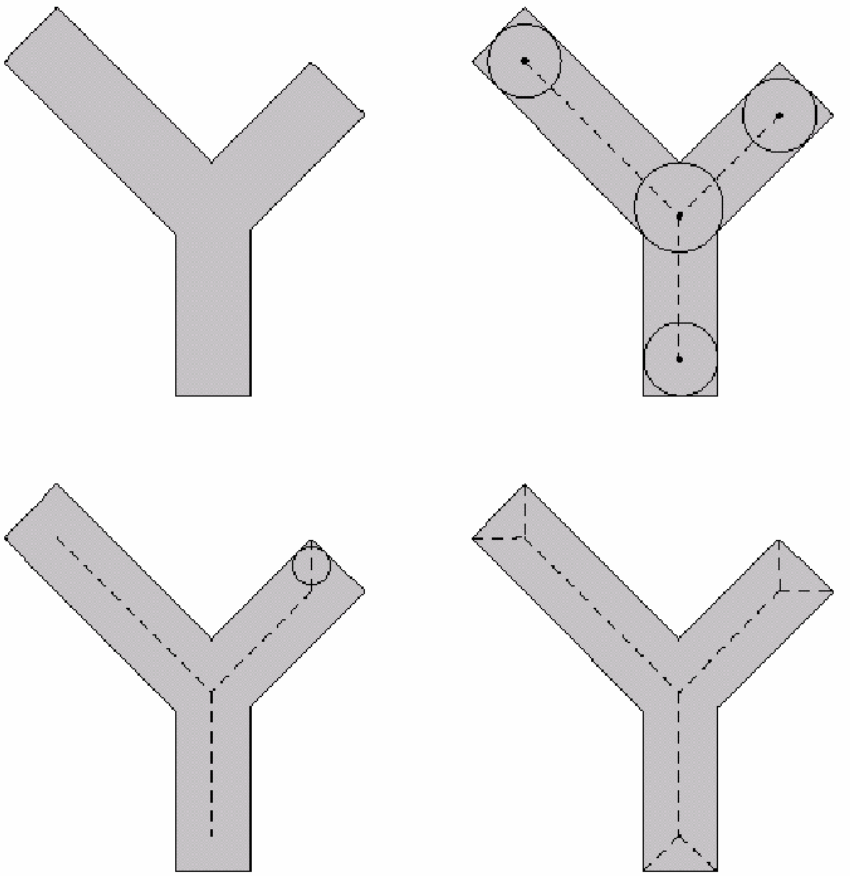
a b
c d
e

FIGURE 9.22 (a) Set A. (b) Complement of A. (c) Result of thinning the complement of A. (d) Thickened set obtained by complementing (c). (e) Final result, with no disconnected points.

9.5.7 Skeletons

The algorithm proposed in this section is similar to the medial axis transformation (MAT). The MAT transformation is discussed in section 11.1.5 and is far inferior to the skeletonization algorithm introduced in section 11.1.5. The skeletonization algorithm proposed in this section also does not guarantee connectivity. We therefore do not discuss this algorithm.

Illustration of the above comments...



a b
c d

FIGURE 9.23
(a) Set A.
(b) Various positions of maximum disks with centers on the skeleton of A.
(c) Another maximum disk on a different segment of the skeleton of A.
(d) Complete skeleton.

A further illustration...

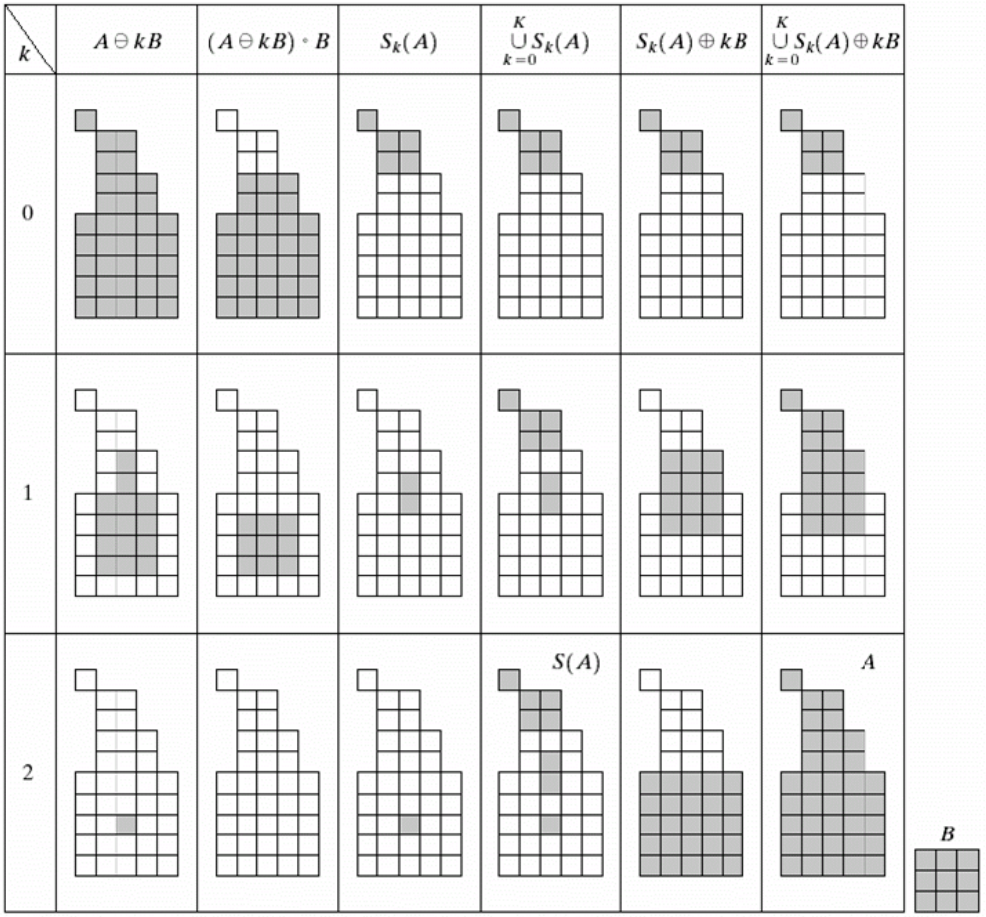


FIGURE 9.24 Implementation of Eqs. (9.5-11) through (9.5-15). The original set is at the top left, and its morphological skeleton is at the bottom of the fourth column. The reconstructed set is at the bottom of the sixth column.

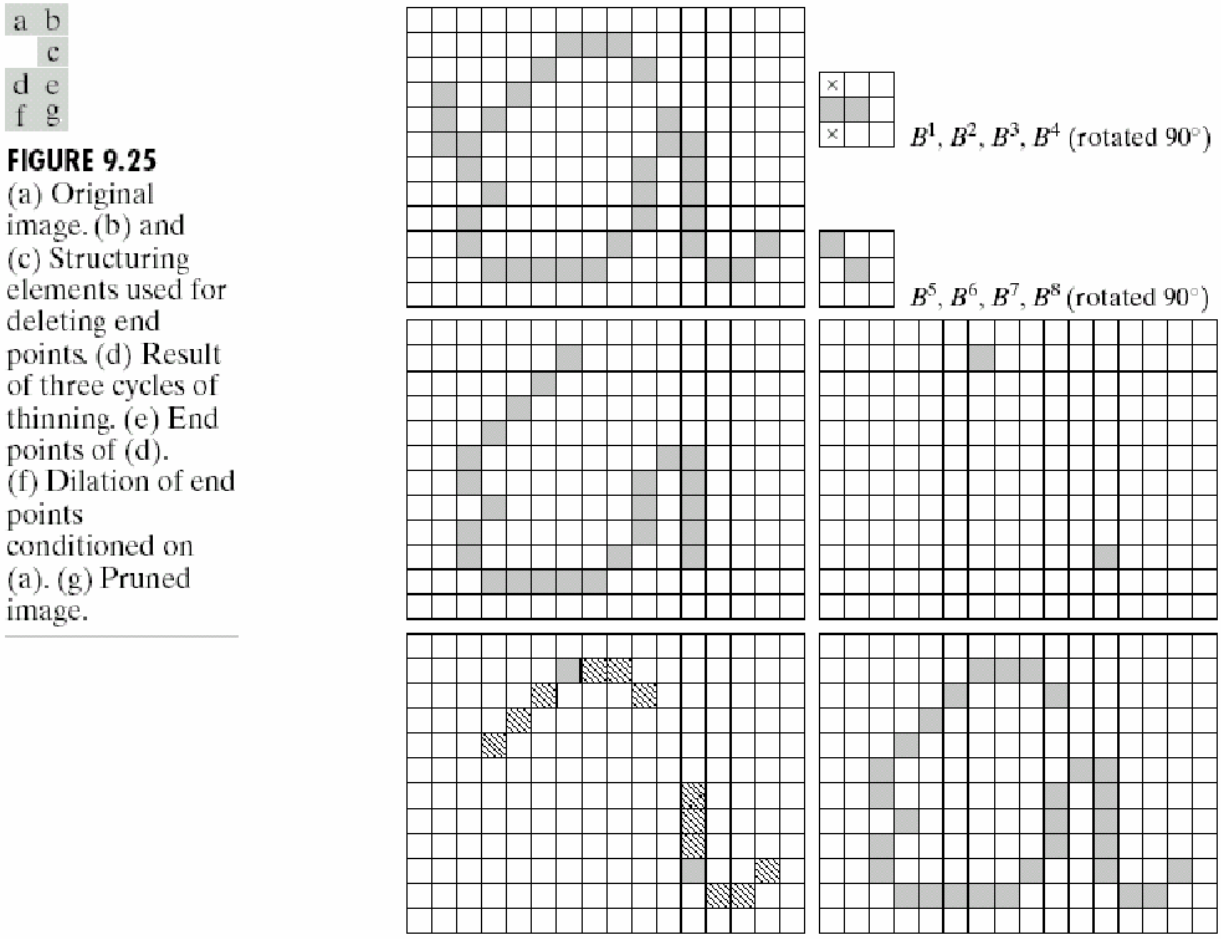
9.5.8 Pruning

• Cleans up "parasitic" components left by thinning and skeletonization

• Use combination of morphological techniques

Illustrative problem: hand-printed character recognition

- Analyze shape of skeleton of character
- Skeletons characterized by spurs ("parasitic" components)
- Spurs caused during erosion of non-uniformities in strokes
- We assume that the length of a parasitic component does not exceed a specified number of pixels



a b
c
d e
f g

FIGURE 9.25
(a) Original image. (b) and (c) Structuring elements used for deleting end points. (d) Result of three cycles of thinning. (e) End points of (d). (f) Dilation of end points conditioned on (a). (g) Pruned image.

Any branch with three or less pixels is to be eliminated

(1) Three iterations of:

$$X_1 = A \otimes \{B\}$$

(2) Find all the end points in X_1:

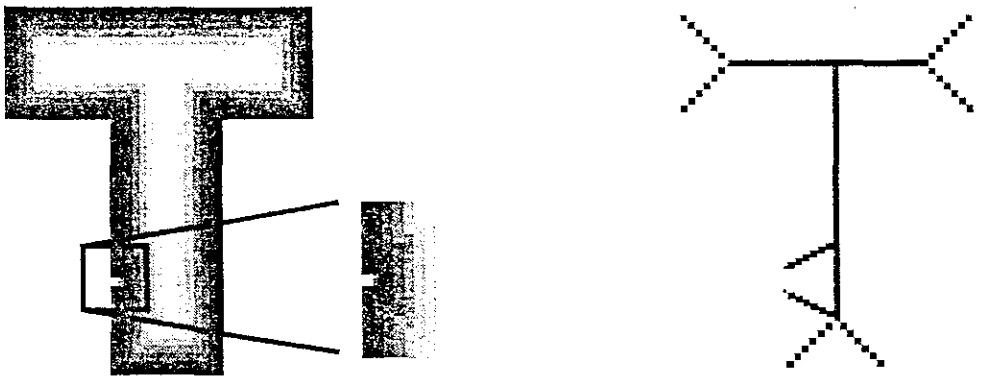
$$X_2 = \cup_{k=1}^{8} \left(X_1 \circledast B^k\right)$$

(3) Dilate end points three times, using A as a delimiter:

$$X_3 = (X_2 \oplus H) \cap A, \quad H = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

(4) Finally:

$$X_4 = X_1 \cup X_3$$



(a) (b)

Figure 5.2 A single pixel difference between two objects can create a large difference in their skeletons. (a) The T-shaped object, but with one less black pixel. (b) The skeleton of the new object, quite different from those in Figure 5.1.

exposes a fundamental problem with the medial axis as a skeleton. Most people would agree that the skeletons of two objects that are similar to each other should, in turn, be similar. Figure 5.2 shows an object that differs from Figure 5.1a in only a single pixel; the medial axes of these objects, on the other hand, differ substantially.

Most vision researchers would agree that the medial-axis transform often does not yield an ideal skeleton, and takes too long to compute. It does, however, form the basis of a great many thinning methods, and in that regard is a very important concept.

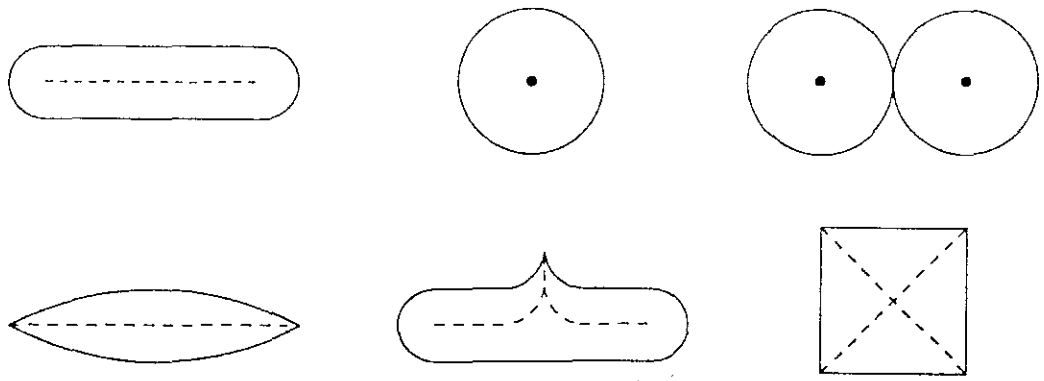


Figure 6.32: *Region skeletons; small changes in border can have a significant effect on the skeleton.*

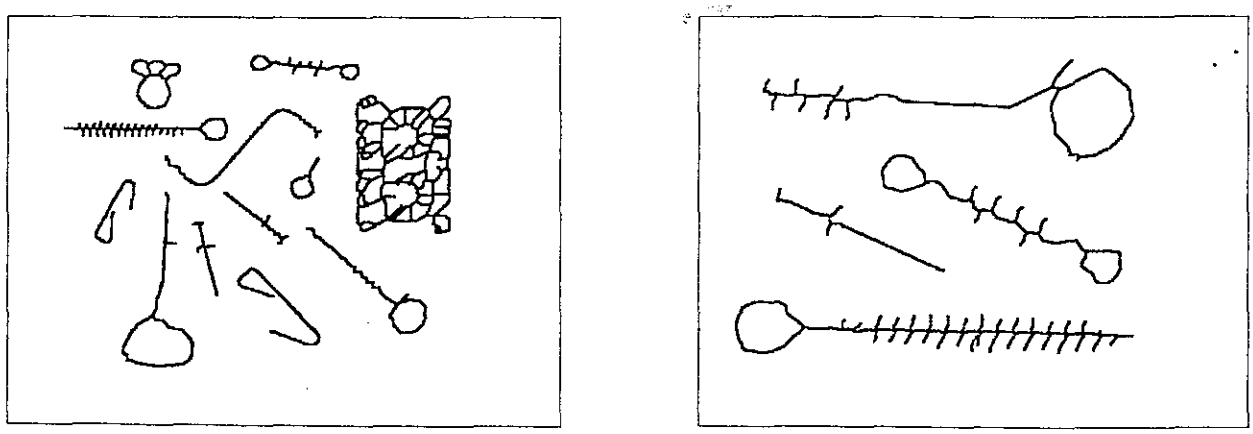


Figure 6.33: *Region skeletons, see Figures 5.1a and 6.2a for original images; thickened for visibility.*

and small-scale local deformation.

Skeleton construction algorithms do not result in graphs, but the transformation from skeletons to graphs is relatively straightforward. Consider first the medial axis skeleton, and assume that a minimum radius circle has been drawn from each point of the skeleton which has at least one point common with a region boundary. Let *contact* be each contiguous subset of the circle which is common to the circle and to the boundary. If a circle drawn from its center A has one contact only, A is a skeleton end point. If the point A has two contacts, it is a normal skeleton point. If A has three or more contacts, the point A is a skeleton node point.

Algorithm 6.9: Region graph construction from skeleton

1. Assign a point description to all skeleton points—end point, node point, normal point.

2. Let graph node points be all end points and node points. Connect any two graph nodes by a graph edge if they are connected by a sequence of normal points in the region skeleton.

The majority of thinning algorithms are based on a repeated stripping away of layers of pixels until no more layers can be removed. There is a set of rules defining which pixels may be removed, and frequently some sort of template-matching scheme is used to implement these rules. Often, the rules are designed so that it is easy to tell when to stop: when no change occurs after two consecutive passes through the image.

The first such algorithm to be described (Stentiford 1983) is typical of the genre. It uses 3×3 templates, where a match of the template in the image means to delete (set to white) the center pixel. The basic algorithm is:

1. Find a pixel location (i,j) where the pixels in the image I match those in template M1 Figure 5.3a).
2. If the central pixel *is not an endpoint*, and has *connectivity number* $= 1$, then mark this pixel for later deletion.

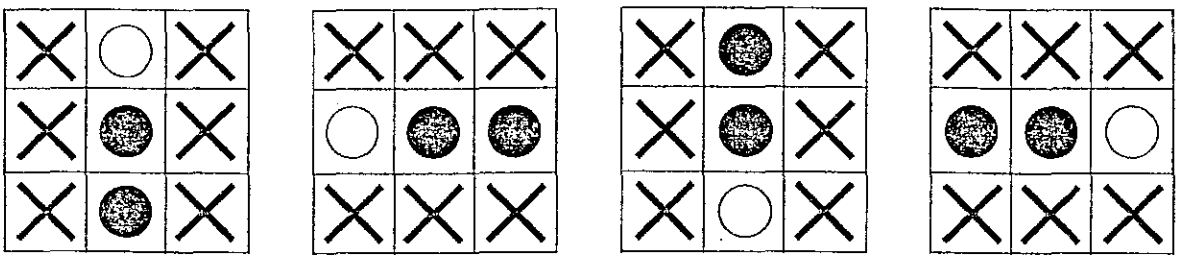


Figure 5.3 Templates for identifying pixels that can be deleted in the Stentiford thinning algorithm. (a) Template M1. (b) Template M2. (c) Template M3. (d) Template M4. The specified black and white pixels in the templates must correspond to pixels of an identical color in the image; the Xs indicate places where we don't care what color the image pixel is.

3. Repeat steps 1 and 2 for all pixel locations matching the template M1.
4. Repeat steps 1–3 for the remaining templates in turn: M2, M3, and M4.
5. If any pixels have been marked for deletion, then delete them by setting them to white.
6. If any pixels were deleted in step 5, then repeat the entire process from step 1; otherwise stop.

The image must be scanned for a template match in a particular order for each template. The purpose of template M_1 is to find removable pixels along the top edge of an object, and we search for a match from left to right, then from top to bottom. M_2 will match a pixel on the left side of an object; this template moves from the bottom to the top of the image, left to right. M_3 will locate pixels along the bottom edge, and moves from right to left, bottom to top. Finally, to find pixels on the right side of an object, match template M_4 in a top-to-bottom, right-to-left fashion. This specific order and direction for applying the templates ensures that the pixels will be removed in a symmetrical way, without any significant directional bias.

There are still two issues to be resolved, both from step 2. A pixel is an endpoint if it is connected to just one other pixel; that is, if a black pixel has only one black

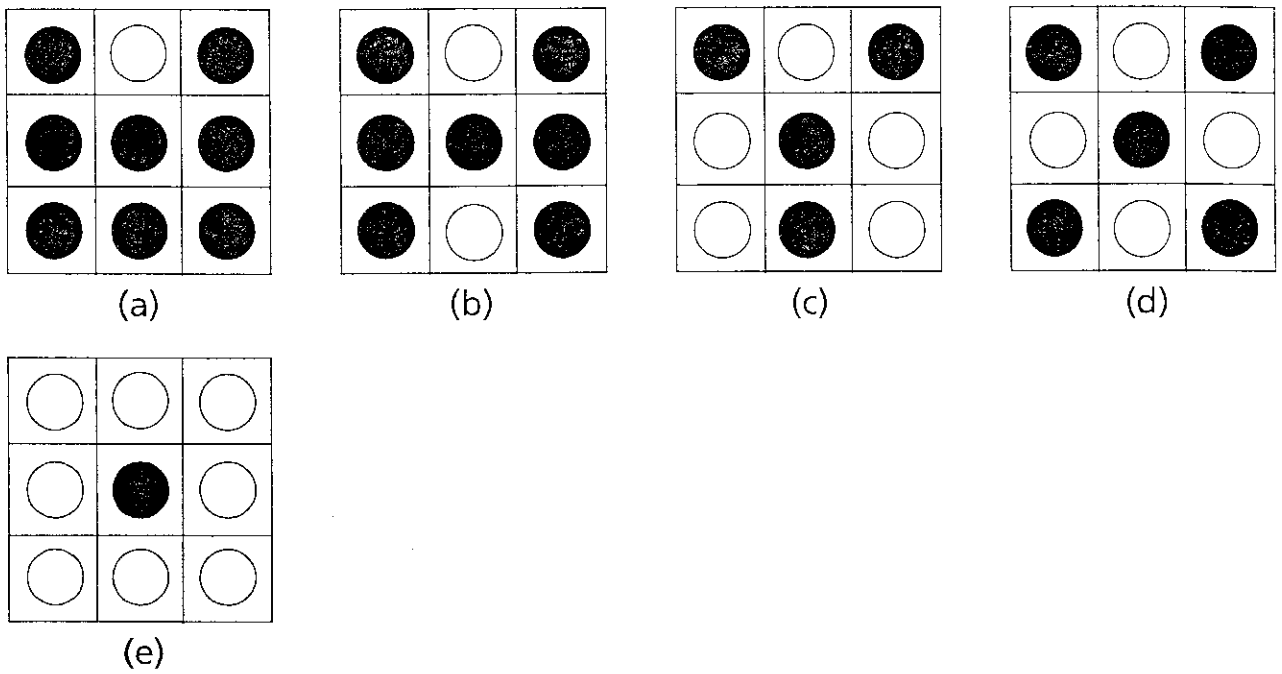


Figure 5.4 An illustration of the connectivity number. (a) The central pixel does not connect any regions, and can be removed. Connectivity number = 1. (b) If the central pixel were to be deleted, then the left and right halves would (might) become disconnected. Connectivity number = 2. (c) Connectivity = 3. (d) Connectivity = 4, the maximum. (e) Connectivity = 0.

A connectivity number is a measure of how many objects a particular pixel *might* connect.

One such connectivity measure, as seen in Figure 5.4, is (Yokoi 1973):

$$C_n = \sum_{k \in S} N_k - (N_k \cdot N_{k+1} \cdot N_{k+2}) \qquad \text{(EQ 5.1)}$$

Where N_k is the color value of one of the eight neighbors of the pixel involved, and S = {1,3,5,7}. N_1 is the color value of pixel to the right of the central pixel,

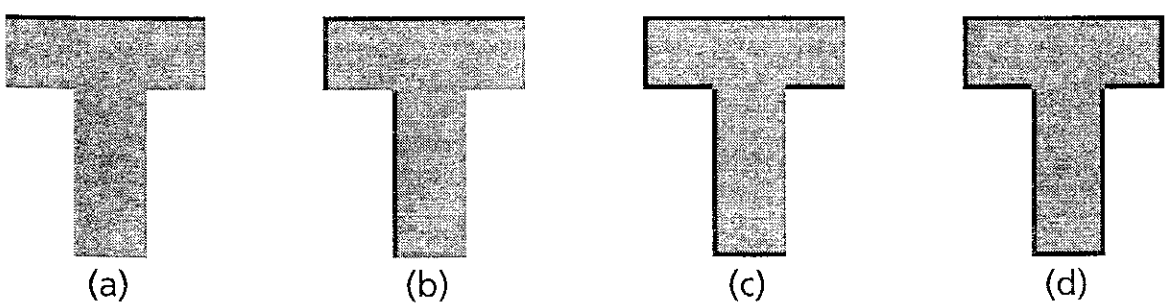


Figure 5.5 The four parts of each iteration of the Stentiford thinning method. (a) After applying template M1. (b) After M2. (c) After M3. (d) After M4. In each case, the black pixels represent those to be deleted in this iteration.

and they are numbered in counterclockwise order around the center. The value of N_k is one if the pixel is white (background) and zero if black (object). The center pixel is N_0, and $N_k = N_{k-8}$ if $k > 8$. Another way that connectivity can be computed is by visiting the neighbors in the order $N_1, N_2 \ldots N_8, N_1$. The number of color changes (black-white) counts the number of regions the central pixel connects.

Figure 5.5 shows one iteration (the first) of this thinning algorithm applied to the T-shaped object of Figure 5.1. One iteration includes one pass for each of the four templates. The black pixels are those marked for deletion, and it is clear from

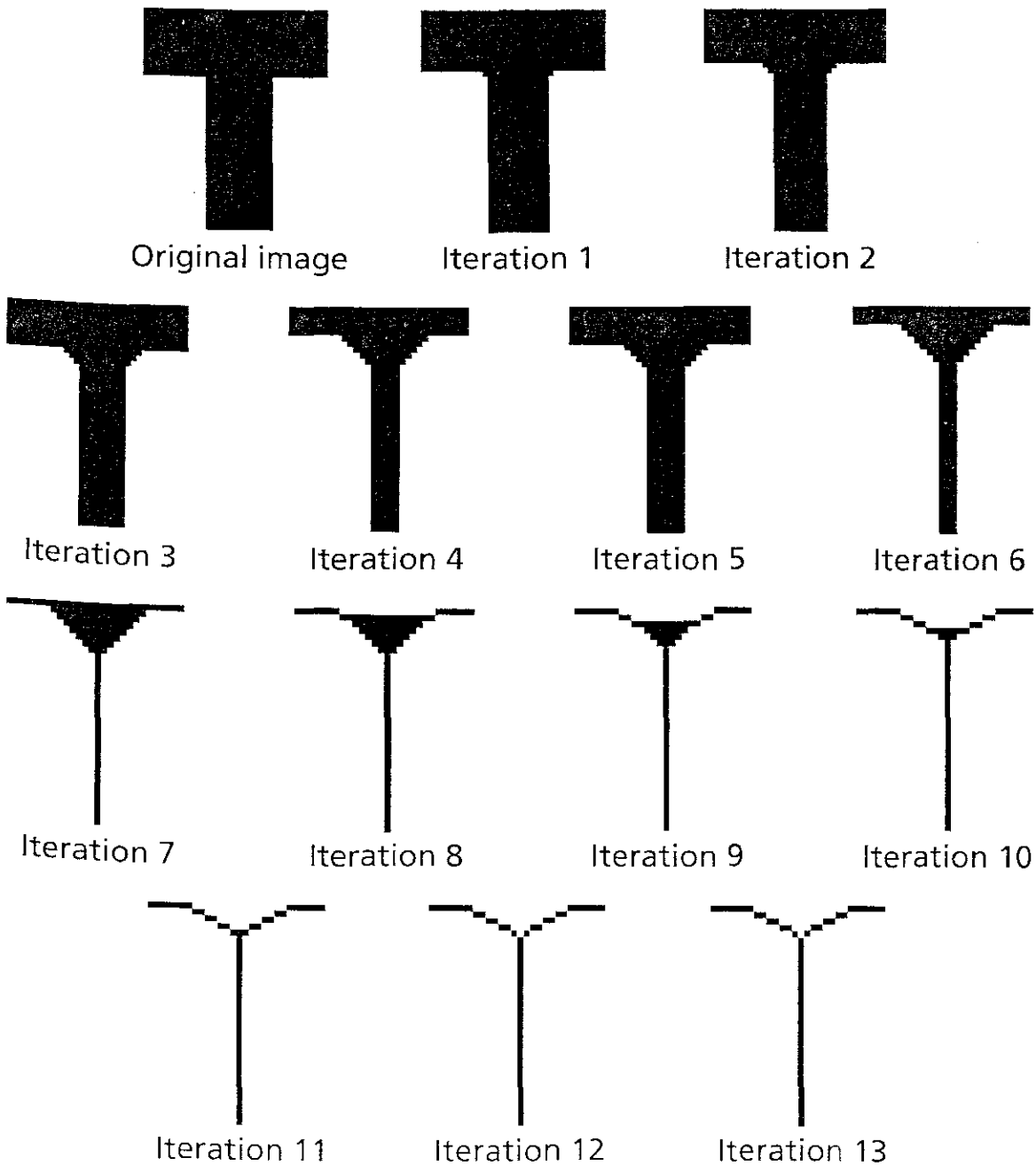


Figure 5.6 All iterations of the Stentiford thinning algorithm applied to the T. The last two iterations are the same, since one extra pass is needed to ensure that the skeleton is complete.

the figure exactly what each template accomplishes. Each complete iteration effectively erodes a layer of pixels from the outside of the object. but unlike standard morphological erosion, the deletion of a pixel is contingent upon meeting the endpoint and connectedness constraints.

Complete thinning of this object requires 13 iterations (counting the final iteration, which does nothing except show that we are finished). Figure 5.6 shows the resulting image after each iteration. One iteration makes four passes through the image, which in this case is 60 × 60 pixels, or 3600 pixels. Thus, 187,000 pixels were examined in order to thin this simple image. It gets worse: Each template application looks at three pixels (the maximum is 561,600), and each time a template match occurs, another 18 pixels are looked at (the upper limit is 10,108,800 pixels, but will be a fraction of that in practice). Finally, there will be one extra pass each iteration to delete the marked pixels (10,152,000). This is an expensive way to thin a small image, but is quite typical of template-based mark-and-delete algorithms.

There are a few classic problems with this thinning algorithm that show up as *artifacts* in the skeleton. They are classic because they tend to appear in a great variety of algorithms of this type, and researchers in the area have learned to anticipate them. The first of these is called *necking*, in which a narrow point at the intersection of two lines is stretched into a small line segment (Figure 5.7a).

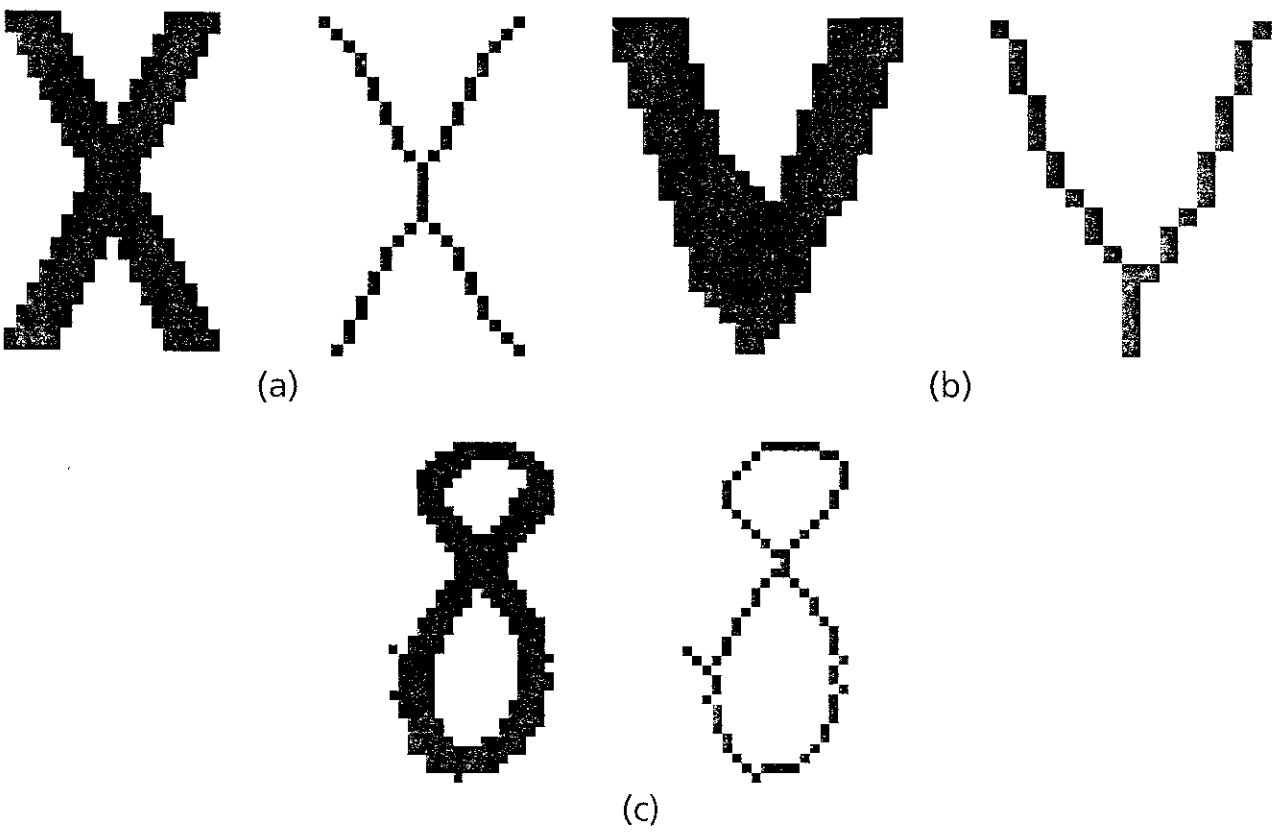


Figure 5.7 Classic thinning artifacts. (a) Necking. (b) Tailing. (c) Spurious projection (line fuzz).

Tails can be created where none exist because of excess thinning where two lines meet at an acute angle (Figure 5.7b). Finally, and perhaps most common, is the creation of extra line segments joining a real skeletal segment; this has been called a *spurious projection*, *hairs*, or *line fuzz* (5.7c).

Stentiford suggests a preprocessing stage to minimize these thinning artifacts. Since line fuzz is frequently caused by small irregularities in the object outline, a smoothing step is suggested before thinning to remove them. Basically, a pass is made over all pixels, deleting those having two or fewer black neighbors and having a connectivity number less than two.

For dealing with necking, he suggests a procedure called *acute angle emphasis*, in which pixels near the joint between two lines are set to white if they "plug up" an acute angle. This is done using the templates seen in Figure 5.8. A match to any template marks the central pixel for deletion, and causes another iteration of less severe acute angle emphasis using only the first three templates of each type. If any pixels were deleted, one last pass using only the first templates of each type is performed.

Smoothing is done first, followed by all passes of acute angle emphasis, followed finally by the thinning steps. Figure 5.9 shows the final skeletons of the characters from Figure 5.7 when the preprocessing steps are included.

As good as these skeletons appear to be, the method is still flawed. The use of three stages of acute angle emphasis will not be sufficient for very thick characters, and the templates do not match all situations that can cause necking and tailing. Also, the smoothing step will not catch all irregularities that can cause line fuzz.

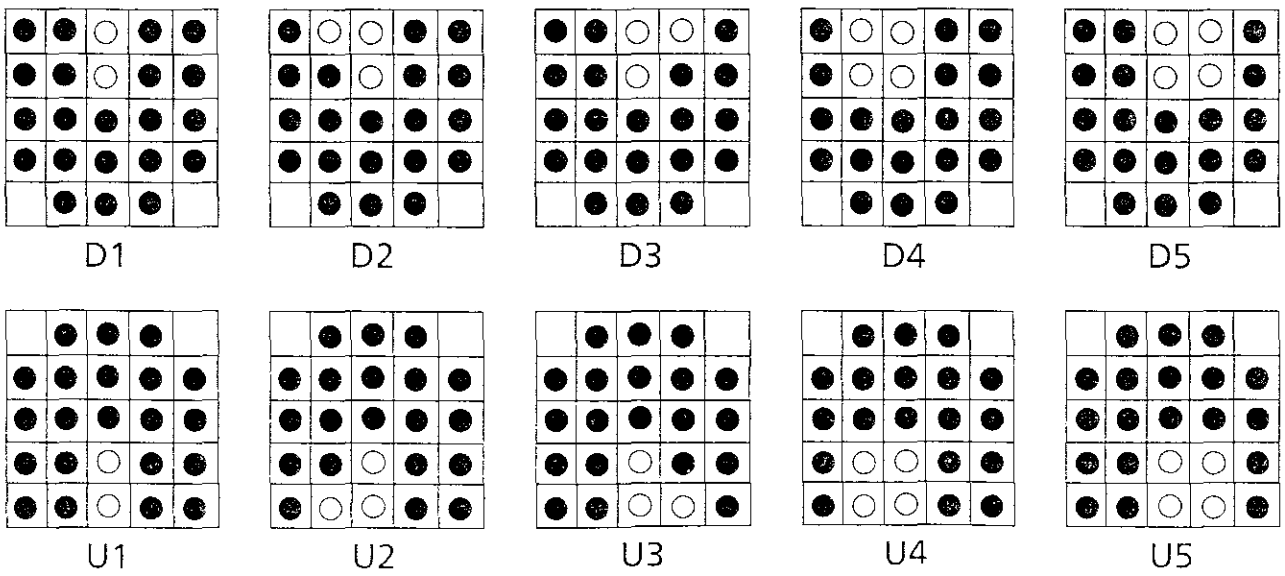


Figure 5.8 Templates used for the acute angle emphasis preprocessing step.

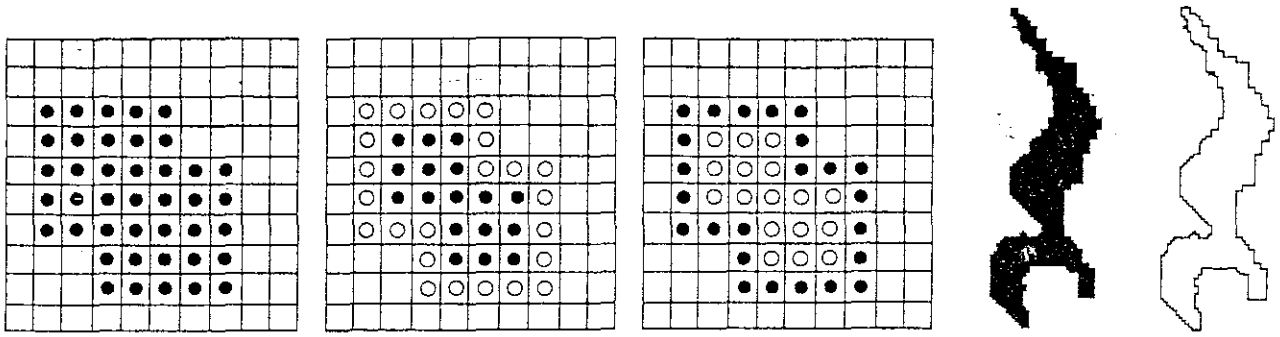


Figure 2.18 Morphological boundary extraction. (a) The squares image. (b) The squares image after an erosion by the simple structuring element. (c) Difference between the squares image and the eroded image: the boundary. (d) A musical quarter rest, scanned from a document. (e) The boundary of the quarter rest as found by this algorithm.

Figure 2.18 shows this method used to extract the boundaries of the "squares" image of Figure 2.17a. A larger example, that of a quarter rest scanned from a page of sheet music, also appears in the figure.

onal Dilation

There are occasions when it is desirable to dilate an object in such a way that certain pixels remain immune. If, for example, an object cannot occupy certain parts of an image, then a dilation of the object must not intrude into that area. In that case, a *conditional dilation* can be performed. The forbidden area of the image is specified as a second image in which the forbidden pixels are black (1). The notation for conditional dilation will be:

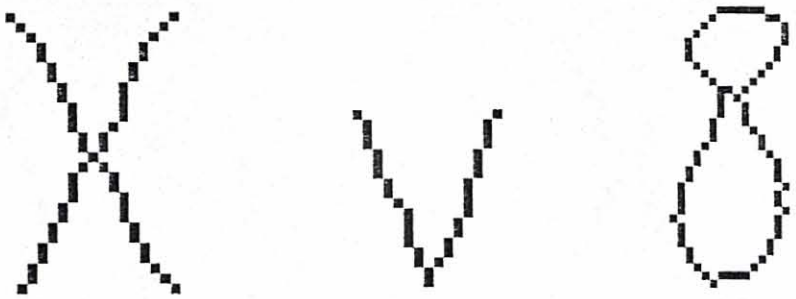
$$A \oplus (S_e, A') \qquad \text{(EQ 2.15)}$$

where S_e is the structuring element to be used in the dilation, and A' is the image representing the set of forbidden pixels.

One place where this is useful is in segmenting an image. Determining a good threshold for grey-level segmentation can be difficult, as discussed later in Chapter 3. However, sometimes two bad thresholds can be used instead of one good one. If a very high threshold is applied to an image, then only those pixels that are certainly supposed to belong to an object will remain. Of course, a great many will be missed. Now a very low threshold can be applied to the original image, giving an image that has too many object pixels, but where the background is marked with some certainty. Then the following conditional dilation is performed:

$$R = I_{high} \oplus (\text{simple}, I_{low}) \qquad \text{(EQ 2.16)}$$

The image R is now a segmented version of the original, and it is in some cases



(a) (b) (c)

Figure 5.9 Final thinned characters, after both preprocessing steps and thinning.

Still, perfection should not be expected, and the method does pretty well, particularly as a preprocessing step for character recognition.

One thinning algorithm that seems to be in everybody's toolkit is the Zhang-Suen (Zhang 1984) algorithm. It has been used as a basis of comparison for thinning methods for many years, and is fast and simple to implement. It is a *parallel* method, meaning that the new value for any pixel can be computed using only the values known from the previous iteration. Therefore, if a computer having one CPU per pixel were available, it could determine the entire next iteration simultaneously. Since most of us don't have a computer of that size, let's consider only the version of the program that uses one CPU.

The algorithm is broken up into two subiterations instead of, for example, the four subiterations of the Stentiford method. In one subiteration, a pixel I(i,j) is deleted (or marked for deletion) if the following four conditions are all true:

1. Its connectivity number is one (1).
2. It has at least two black neighbors and not more than six.
3. At least one of I(i,j+1), I(i−1,j) and I(i,j−1) are background (white).
4. At least one of I(i−1,j), I(i+1,j) and I(i,j−1) are background.

At the end of this subiteration the marked pixels are deleted. The next subiteration is the same except for steps 3 and 4:

1. At least one of I(i−1,j), I(i,j+1) and I(i+1,j) are background.
2. At least one of I(i,j+1), I(i+1,j) and I(i,j−1) are background.

and again, any marked pixels are deleted. If at the end of either subiteration there are no pixels to be deleted, then the skeleton is complete, and the program stops.

Figure 5.10 shows the skeletons found by the Zhang-Suen algorithm applied to the four example images seen so far: the T, X, V, and 8. The T skeleton is exceptionally good, and the V skeleton does not show any signs of tailing. The X

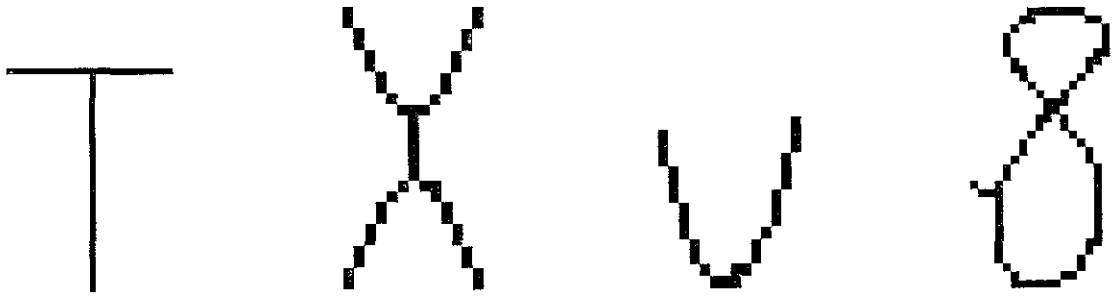


Figure 5.10 The skeletons produced by the standard Zhang-Suen thinning algorithm when applied to the test images of Figure 5.2 and 5.7.

skeleton does still show necking, and the 8 skeleton still has line fuzz. The preprocessing steps suggested by Stentiford may clear this up.

Before trying this, an improvement of the algorithm was suggested (Holt 1987) that is faster and does not involve subiterations. First, the two subiterations are written as logical expressions which use the 3 × 3 neighborhood about the pixel concerned. The first subiteration above can be written as:

$$v(C) \wedge (\sim\text{edge}(C) \vee (v(E) \wedge v(S) \wedge (v(N) \vee v(W)))) \qquad \text{(EQ 5.2)}$$

which is the condition under which the center pixel C survives the first subiteration. The v function gives the value of the pixel (1 = true for an object pixel, 0 = false for background), and the edge function is true if C is on the edge of the object—this corresponds to having between two and six neighbors and connectivity number = 1. The letters E, S, N, and W correspond to pixels in a particular direction from the center pixel C; E means east (as in $I(i,j+1)$) S means south (as in $I(i+1,j)$) and so on.

The second subiteration would be written as:

$$v(C) \wedge (\sim\text{edge}(C) \vee (v(W) \wedge v(N) \wedge (v(S) \vee v(E)))) \qquad \text{(EQ 5.3)}$$

Holt and company combined the two expressions for survival (Eqs. 5.2 and 5.3) with a connectedness-preserving condition (needed for parallel execution) and came up with the following single expression for pixel survival:

$$\begin{aligned} v(C) \wedge (&\sim\text{edge}(C) \vee \\ &(\text{edge}(E) \wedge v(N) \wedge v(S)) \vee \\ &(\text{edge}(S) \wedge v(W) \wedge v(E)) \vee \\ &(\text{edge}(E) \wedge \text{edge}(SE) \wedge \text{edge}(S)) \end{aligned} \qquad \text{(EQ 5.4)}$$

This expression is not a daunting as it appears: the v functions are simply pixel values, and the edge function is just about as complex as the connectivity function used in the Stentiford algorithm. The result from this are good, but not identical to the standard Zhang-Suen. However, there is still more to come.

Sometimes, when thinning is complete, there are still pixels that could be deleted. Principal among these are pixels that form a staircase; clearly half of the pixels in a staircase could be removed without affecting the shape or connectedness of the overall object. Basically, the central pixel in one of the following windows can be deleted:

0	1	x		x	1	0		0	x	x		x	x	0
1	1	x		x	1	1		x	1	1		1	1	x
x	x	0		0	x	x		x	1	0		0	1	x

To avoid creating a new hole, we simply add a condition that one of the x values be 0. For windows having a northward bias (the first two above) the expression for survival of a pixel in the staircase-removal iteration is:

$$v(C) \wedge \sim(v(N) \wedge$$
$$((v(E) \wedge \sim v(NE) \wedge \sim v(SW) \wedge (\sim v(W) \vee \sim v(S)) \vee \quad \text{(EQ 5.5)}$$
$$(v(W) \wedge \sim v(NW) \wedge \sim v(SE) \wedge (\sim v(E) \vee \sim v(S))))))$$

The pass having a southward bias is the same, but with north and south exchanged. None of the example images seen so far possess any significant amount of staircasing, but the image introduced in Figure 5.11 does. The version thinned using staircase removal seems more smooth and symmetrical than the other skeletons. Figure 5.12 shows the result of applying this method to the four test images

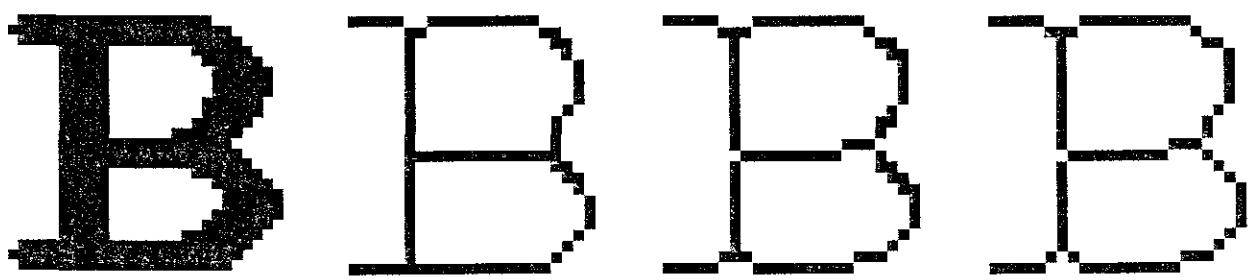


Figure 5.11 Variations on the Zhang-Suen thinning algorithm. (a) Original image (b) Thinned using the standard algorithm. (c) Thinned using Holt's variation. (d) Holt's variation plus staircase removal.

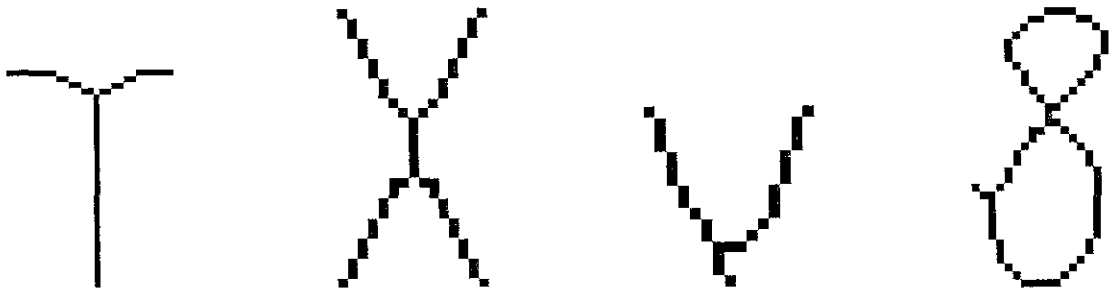


Figure 5.12 Results from Holt's algorithm with staircase removal applied to the standard test images.

we have been using. The basic problems are still present; in fact, this method does not deal with tails as well as the standard Zhang-Suen method, and the T skeleton is not as good.

If simple speed is what is of importance, then the Holt variation of Zhang-Suen is the better of the methods seen so far. On the other hand, if the quality of the skeleton is of prime importance, it is probable that a merging of the three methods is in order: Stentiford's preprocessing scheme feeding images into Zhang-Suen's basic algorithm, with Holt's staircase removal as a post-processor. The code for this sequence of operations appears in Section 5.8, since it includes all of the techniques of importance that have been discussed to this point. It is available on the accompanying CD as the program thnbest, and does appear to generate the best skeletons of all of the methods seen so far; of course, this is a subjective measure. The best skeletons can be seen in Figure 5.13.

5.4 USE OF CONTOURS

The iterative mark-and-delete methods seen so far have in common that they always delete pixels from the outer layer. These are on the boundary of the object, and form a contour having a distance of zero pixels from the background. A

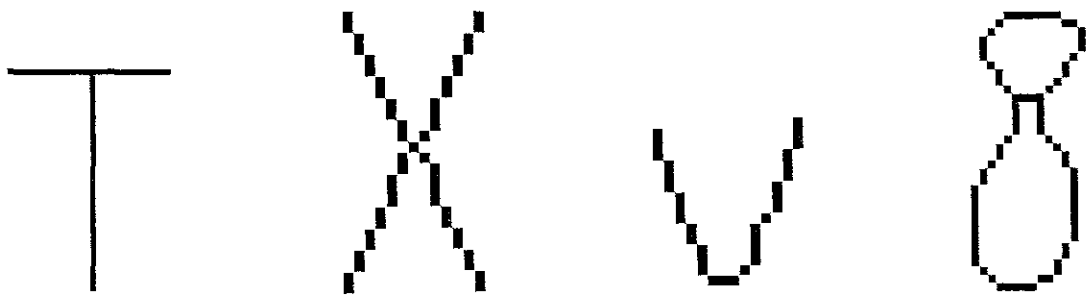


Figure 5.13 Skeletons obtained from using the Stentiford preprocessing steps combined with the Zhang-Suen thinning algorithm and Holt's staircase-elimination procedure.

Erosion: $A \ominus B$

Dilation: $A \oplus B$

Opening: $A \circ B$

Closing: $A \bullet B$

Hit or Miss: $A \otimes B = (A \ominus \mathcal{B}_1) \cap [A^c \ominus \mathcal{B}_2]$

Boundary of a set: $A - (A \ominus B)$

Hole filling: set one point X_0 in hole to 1

$$\text{then } X_k = (X_{k-1} \oplus B) \cap A^c$$

Extraction Connected Components: start with X_0

$$\text{then } X_k = (X_{k-1} \oplus B) \cap A$$

Convex Hull: Choose 4 masks B_i

$$\text{then } X_k^i = (X_{k-1} \otimes B) \cup A$$

after convergence the convex hull is given by

$$C(A) = \bigcup_{i=1}^{4} D^i$$

Thinning:
$$A \otimes B = A \cap (A \otimes B)$$
$$= A - (A \otimes B)^c \qquad \otimes\text{=hit\&miss}$$

$$A \otimes \{B\} = (...((A \otimes B^1) \otimes B^2)... \otimes B^n)$$

$$A \backslash B = X \cap Y^c$$

Thinning and thickening transformations are very often used sequentially. Let $\{B_{(1)}, B_{(2)}, B_{(3)}, \ldots, B_{(n)}\}$ denote a sequence of composite structuring elements $B_{(i)} = (B_{i_1}, B_{i_2})$. **Sequential thinning** can then be expressed as a sequence of eight structuring elements for square rasters,

$$X \oslash \{B_{(i)}\} = (((X \oslash B_{(1)}) \oslash B_{(2)}) \ldots \oslash B_{(n)}) \tag{11.49}$$

and **sequential thickening** as

$$X \odot \{B_{(i)}\} = (((X \odot B_{(1)}) \odot B_{(2)}) \ldots \odot B_{(n)}) \tag{11.50}$$

There are several sequences of structuring elements $\{B_{(i)}\}$ that are useful in practice. Many of them are given by a permissible rotation of a structuring element in the appropriate digital raster (e.g., hexagonal, square, or octagonal). These sequences, sometimes called the **Golay alphabet** [Golay 69], are summarized for the hexagonal raster in [Serra 82]. We shall present structuring elements of the Golay alphabet for octagonal rasters. 3×3 matrices will be shown for the first two rotations, from which the other rotations can easily be derived.

A composite structuring element can be expressed by a single matrix only. A value of one in it means that this element belongs to B_1 (it is a subset of objects in the hit-or-miss transformation), and a value zero belongs to B_2 and is a subset of the background. An asterisk $*$ in the matrix denotes an element that is not used in the matching process, so its value is not significant.

Thinning and thickening sequential transformations converge to some image—the number of iterations needed depends on the objects in the image and the structuring element used. If two successive images in the sequence are identical, the thinning (or thickening) is stopped.

Sequential thinning by structuring element L

This sequential thinning is quite important, as it serves as the homotopic substitute of the skeleton; the final thinned image consists only of lines of width one and isolated points.

The structuring element L from the Golay alphabet is given by

$$L_1 = \begin{bmatrix} 0 & 0 & 0 \\ * & 1 & * \\ 1 & 1 & 1 \end{bmatrix} \quad L_2 = \begin{bmatrix} * & 0 & 0 \\ 1 & 1 & 0 \\ * & 1 & * \end{bmatrix} \ldots \tag{11.51}$$

(The other six elements are given by rotation). Figure 11.26 shows the result of thinning with the structuring element L, after five iterations to illustrate an intermediate result, and Figure 11.27 shows the homotopic substitute of the skeleton when the idempotency was reached (in both cases, the original is shown on the left).

Sequential thinning by structuring element E

Assume that the homotopic substitute of the skeleton by element L has been found. The skeleton is usually jagged, because of sharp points on the outline of the object, but it is possible to 'smooth' the skeleton by sequential thinning by structuring element E. Using n iterations, several points (whose number depends on n) from the lines of width one (and isolated points as well) are removed from free ends. If thinning by element E is performed until the image does not change, then only closed contours remain.

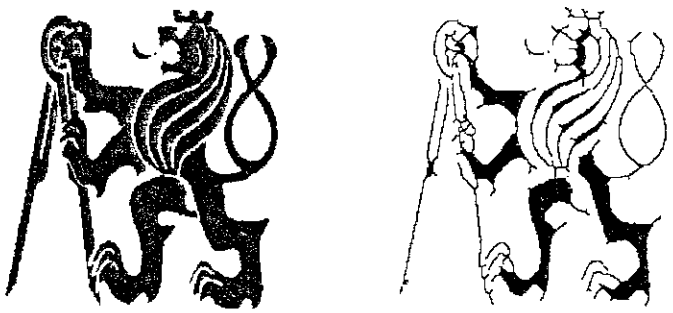


Figure 11.26: *Sequential thinning using element L after five iterations.*

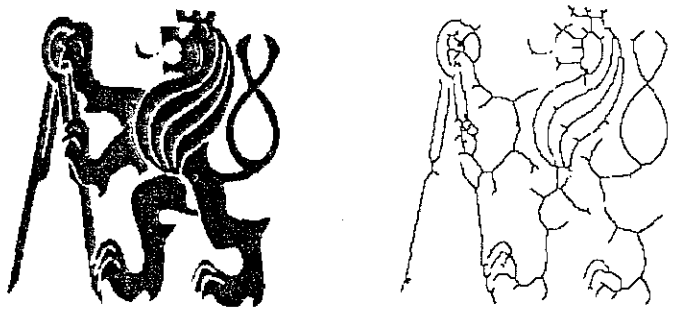


Figure 11.27: *Homotopic substitute of the skeleton (element L).*

The structuring element E from the Golay alphabet is given again by eight rotated masks,

$$E_1 = \begin{bmatrix} * & 1 & * \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad E_2 = \begin{bmatrix} 0 & * & * \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \ldots \tag{11.52}$$

Figure 11.28 shows sequential thinning (five iterations) by the element E of the skeleton from Figure 11.27. Notice that lines have been shortened from their free ends.

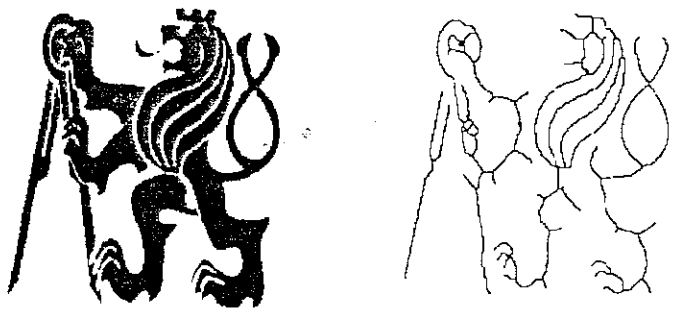


Figure 11.28: *Five iterations of sequential thinning by element E.*

There are three other elements M, D, C in the Golay alphabet [Golay 69]. These are not much used in practice at present, and other morphological algorithms are used instead to find skeletons, convex hulls, and homotopic markers.

The computationally most efficient algorithm of which we are aware creates the connected skeleton as the minimal superset of the skeleton by maximal balls [Vincent 91]. Its performance is shown in Figure 11.29. The homotopy is preserved.

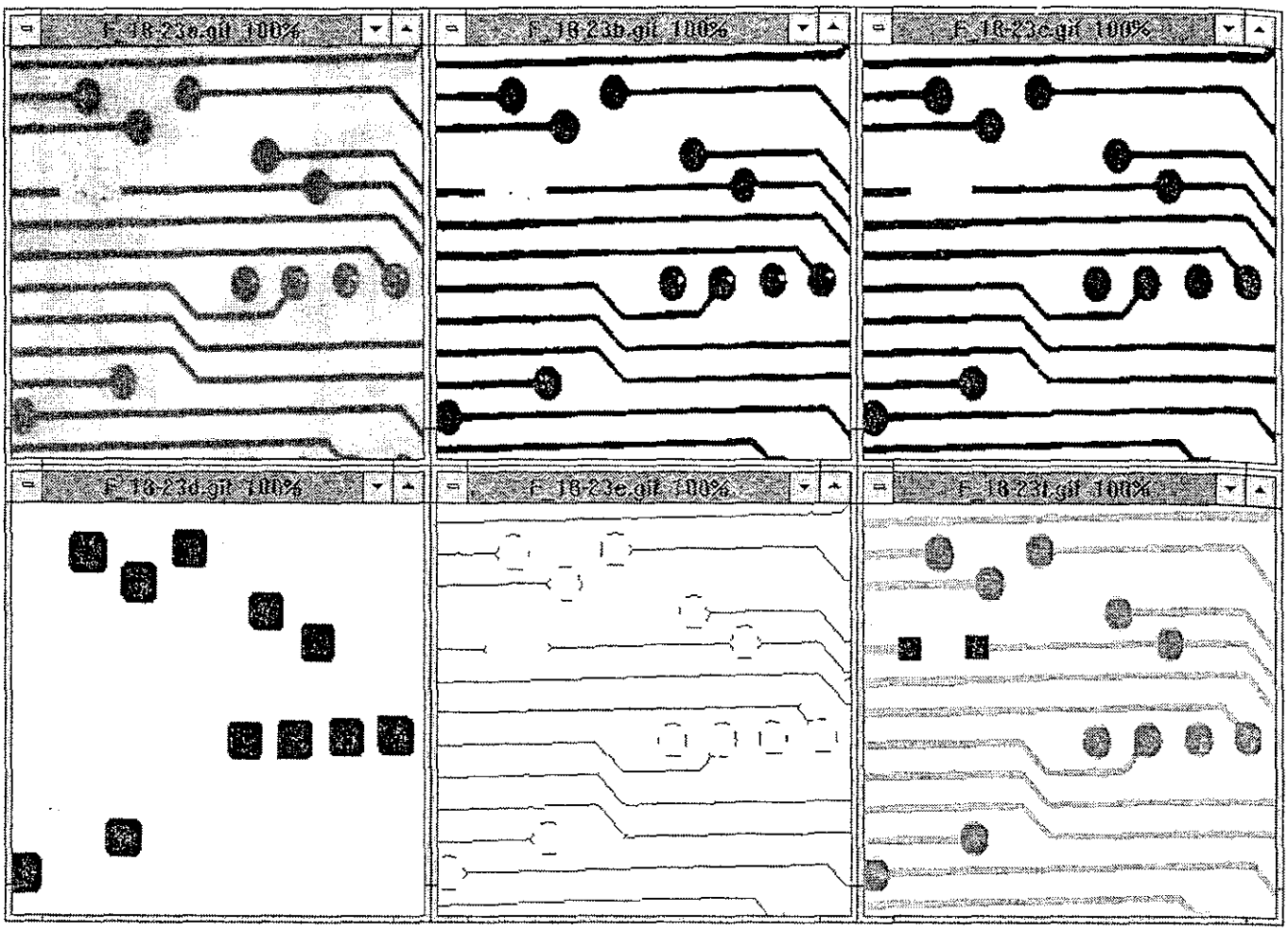


Figure 18–25 Morphological analysis of a printed circuit board image: (a) grayscale image; (b) thresholded image; (c) cleanup by opening; (d) isolation of pads by erosion and dilation; (e) isolation of traces by skeletonization; (f) final display of traces, pads and break points (Courtesy Luc Nocente, Noesis Vision)

Pruning

Pruning methods are an essential complement of thinning and skeletonizing algorithms, because these procedures tend to leave parasitic components that need to be cleaned up by postprocessing. We begin the discussion with a pruning problem and then develop a morphological solution based on the material introduced in the preceding sections. Thus we take advantage of what has been developed so far to illustrate how to go about solving a problem by combining several of the techniques discussed so far.

A common approach to the automated recognition of hand-printed characters is to analyze the shape of the skeleton of each character. These skeletons are often characterized by "spurs" (parasitic components), caused during erosion by nonuniformities in the strokes composing the characters. We develop a morphological technique for handling this problem, starting with the assumption that the length of a parasitic component does not exceed three pixels.

Figure 8.40(a) shows the skeleton of a hand-printed "a." The parasitic component on the leftmost part of the character is typical of what we are interested in removing. The solution is based on suppression of a parasitic branch by successively eliminating its end point. Of course, this also shortens (or eliminates) other branches in the character but, in the absence of other structural information, the assumption is that any branch with three or less pixels is to be eliminated. For an input set A, thinning A with a sequence of structuring elements designed to detect only end points achieves the desired

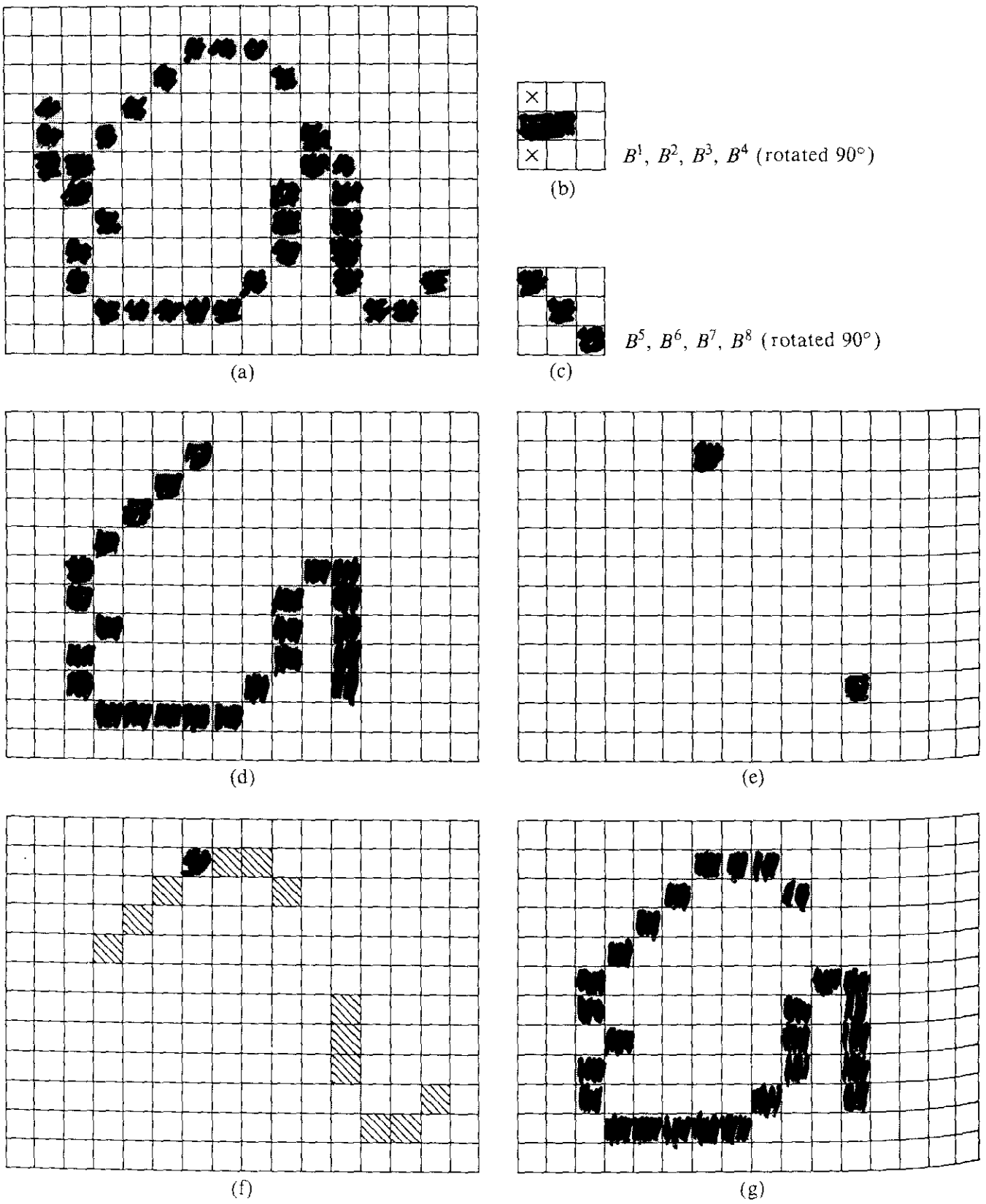


Figure 8.40 *Example of pruning: (a) original image; (b) and (c) structuring elements used for deleting (thinning) end points; (d) result of three cycles of thinning; (e) end points of (d); (f) dilation of end points conditioned on (a); (g) pruned image.*

end;

Another application of conditional dilation is that of filling a region with pixels, which is the inverse operation of boundary extraction. Given an outline of black pixels and a pixel inside of the outline, we are to fill the region with black pixels. In this case the forbidden image will consist of the boundary pixels; we want to fill the region up to the boundary, but never set a pixel that is outside. Since the outside pixels and the inside pixels have the same value, the boundary pixels are forbidden and the dilation continues until the inside region is all black. Then this image and the boundary image are combined to form the final result.

The conditional dilation is:

$$\text{Fill} = P \oplus (S_{cross}, A^c) \qquad \text{(EQ 2.18)}$$

where P is an image containing only the seed pixel, which is any pixel known to be inside the region to be filled, and A is the boundary image for the region to be filled. S_{cross} is the cross-shaped structuring element seen in Figure 2.20b. The same figure shows the steps in the conditional dilation that fills the same boundary that was identified earlier in this section. The seed pixel used in the example is [3,3], but any of the white pixels inside the boundary could have been used.

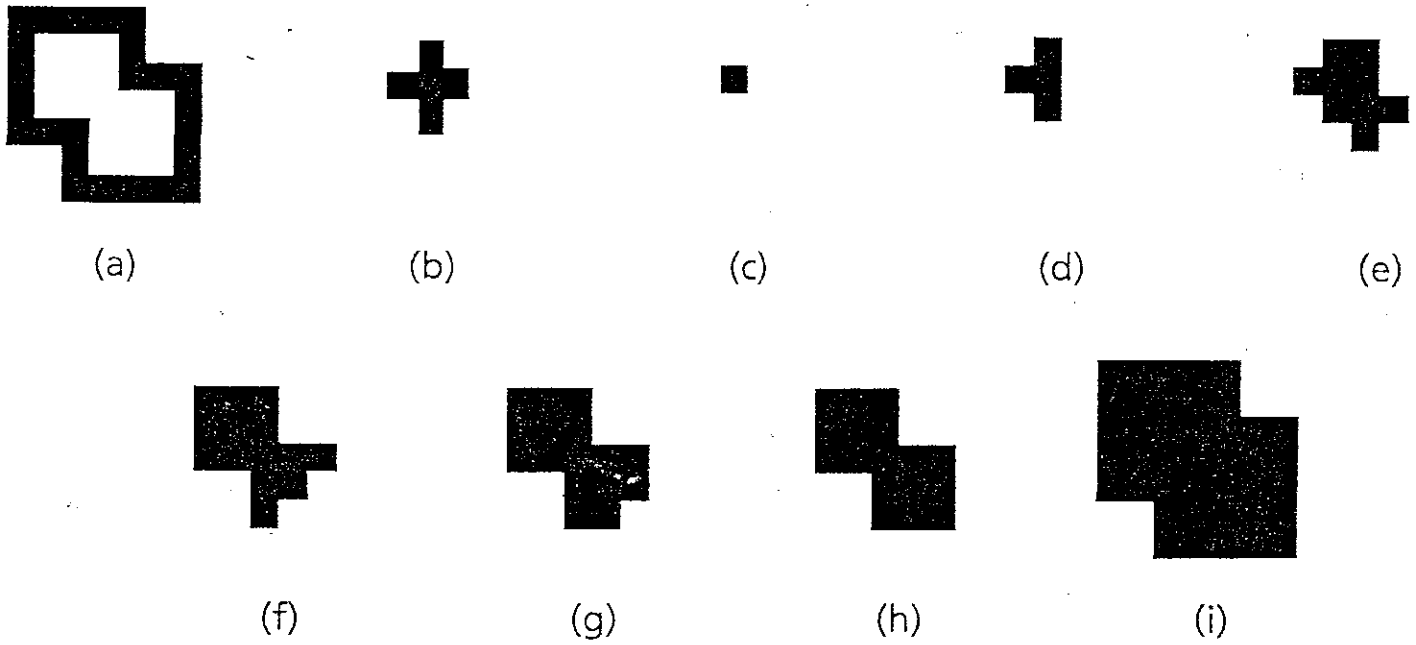


Figure 2.20 Filling a region using conditional dilation. (a) The boundary of the region to be filled. This is the boundary found in Figure 2.18. (b) The structuring element. (c) The seed pixel, and iteration 0 of the process. (d) After iteration 1. (e) After iteration 2. (f) After iteration 3. (g) After iteration 4. (h) After iteration 5 the dilation is complete. (i) Union of (h) with (a) is the result.