# CHAPTER 13

# MORPHOLOGICAL IMAGE PROCESSING

## WHAT WILL WE LEARN?

- What is mathematical morphology and how is it used in image processing?
- What are the main morphological operations and what is the effect of applying them to binary and grayscale images?
- What is a structuring element (SE) and how does it impact the result of a morphological operation?
- What are some of the most useful morphological image processing algorithms?

## 13.1 INTRODUCTION

Mathematical morphology is a branch of image processing that has been successfully used to provide tools for representing, describing, and analyzing shapes in images. It was initially developed by Jean Serra in the early 1980s [SC82] and—because of its emphasis on studying the geometrical structure of the components of an image—named after the branch of biology that deals with the form and structure of animals and plants. In addition to providing useful tools for extracting image components, morphological algorithms have been used for pre- or postprocessing the images containing shapes of interest.

*Practical Image and Video Processing Using MATLAB®*. By Oge Marques.

The basic principle of mathematical morphology is the extraction of geometrical and topological information from an unknown set (an image) through transformations using another, well-defined, set known as *structuring element*. In morphological image processing, the design of SEs, their shape and size, is crucial to the success of the morphological operations that use them.

The IPT in MATLAB has an extensive set of built-in morphological functions, which will be introduced throughout the chapter. You will have a chance to work with many of them in the tutorials at the end of the chapter.

## 13.2 FUNDAMENTAL CONCEPTS AND OPERATIONS

The basic concepts of mathematical morphology can be introduced with the help of set theory and its standard operations: *union* ($\cup$), *intersection* ($\cap$), and *complement*, defined as

$$A^c = \{z | z \notin A\} \tag{13.1}$$

and the *difference* of two sets $A$ and $B$:

$$A - B = \{z | z \in A, z \notin B\} = A \cap B^c \tag{13.2}$$

Let $A$ be a set (of pixels in a binary image) and $w = (x, y)$ be a particular coordinate point. The *translation* of set $A$ by point $w$ is denoted by $A_w$ and defined as

$$A_w = \{c | c = a + w, \quad \text{for} \quad a \in A\} \tag{13.3}$$

The *reflection* of set $A$ relative to the origin of a coordinate system, denoted $\hat{A}$, is defined as

$$\hat{A} = \{z | z = -a, \quad \text{for} \quad a \in A\} \tag{13.4}$$

Figure 13.1 shows a graphical representation of the basic set operations defined above. The black dot represents the origin of the coordinate system.

Binary mathematical morphology theory views binary images as a set of its foreground pixels (whose values are assumed to be 1), the elements of which are in $\mathcal{Z}^2$. Classical image processing refers to a binary image as a function of $x$ and $y$, whose only possible values are 0 and 1. To avoid any potential confusion that this dual view may cause, here is an example of how a statement expressed in set theory notation can be translated into a set of logical operations applied to binary images:

The statement $C = A \cap B$, from a set theory perspective, means

$$C = \{(x, y) | (x, y) \in A \text{ and } (x, y) \in B\} \tag{13.5}$$

The equivalent expression using conventional image processing notation would be

$$C(x, y) = \begin{cases} 1 & \text{if } A(x, y) \text{ and } B(x, y) \text{ are both 1} \\ 0 & \text{otherwise} \end{cases} \tag{13.6}$$

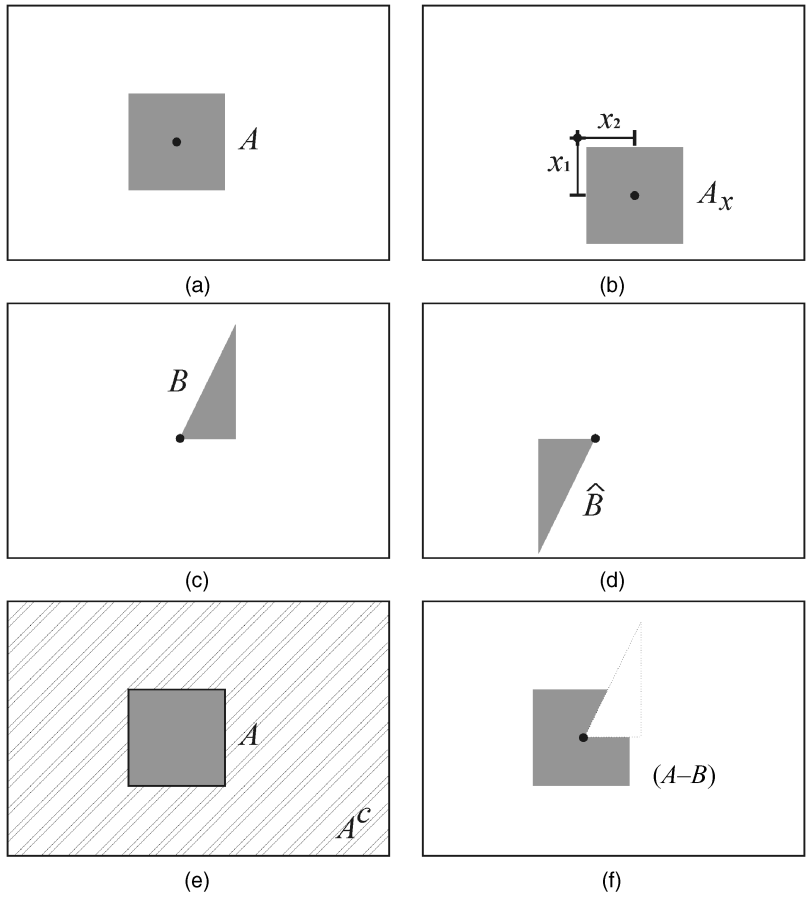


**FIGURE 13.1** Basic set operations: (a) set $A$; (b) translation of $A$ by $x = (x_1, x_2)$; (c) set $B$; (d) reflection of $B$; (e) set $A$ and its complement $A^c$; (f) set difference $(A-B)$.

This expression leads quite easily to a single MATLAB statement that performs the intersection operation using the logical operator AND (`&`). Similarly, complement can be obtained using the unary NOT (`~`) operator, set union can be implemented using the logical operator OR (`|`), and set difference ($A - B$) can be expressed as `(A & ~B)`. Figure 13.2 shows representative results for two binary input images. Note that we have followed the IPT convention, representing foreground (1-valued) pixels as white pixels against a black background.

### 13.2.1 The Structuring Element

The structuring element is the basic neighborhood structure associated with morphological image operations. It is usually represented as a small matrix, whose shape

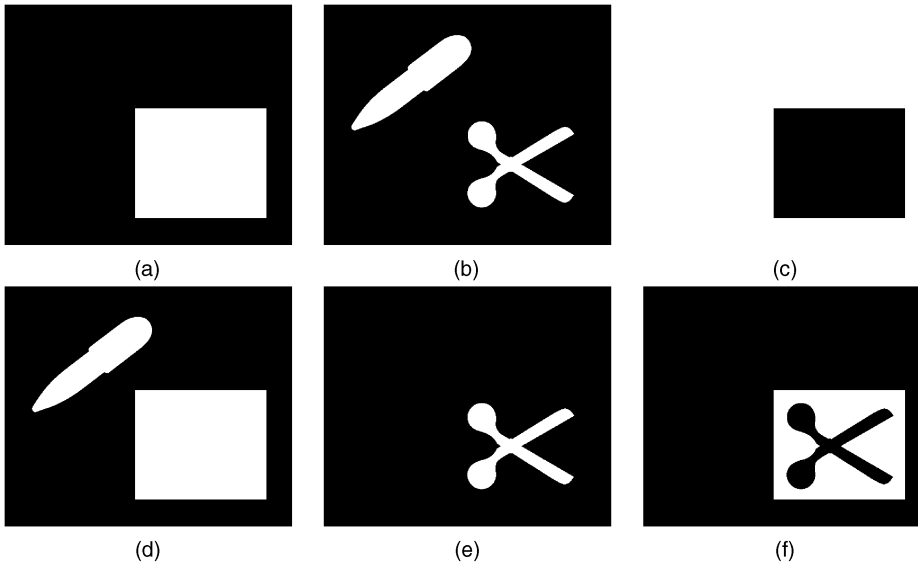


**FIGURE 13.2** Logical equivalents of set theory operations: (a) Binary image ($A$); (b) Binary image ($B$); (c) Complement ($A^c$); (d) Union ($A \cup B$); (e) Intersection ($A \cap B$); (f) Set difference ($A-B$).

and size impact the results of applying a certain morphological operator to an image. Figure 13.3 shows two examples of SEs and how they will be represented in this chapter: the black dot corresponds to their origin (reference point), the gray squares represent 1 (true), and the white squares represent 0 (false). Although a structuring element can have any shape, its implementation requires that it should be converted to a rectangular array. For each array, the shaded squares correspond to the members of the SE, whereas the empty squares are used for padding, only.

### In MATLAB

MATLAB's IPT provides a function for creating structuring elements, `strel`, which supports arbitrary shapes, as well as commonly used ones, such as square, diamond, line, and disk. The result is stored as a variable of class `strel`.

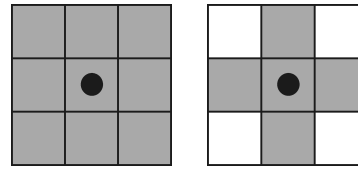


**FIGURE 13.3** Examples of structuring elements: (a) square; (b) cross.

## ■ EXAMPLE 13.1

This example shows the creation of a square SE using `strel` and interpretation of its results.

```
>> se1 = strel('square',4)

se1 =

Flat STREL object containing 16 neighbors.
Decomposition: 2 STREL objects containing a total of 8 neighbors

Neighborhood:
     1     1     1     1
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

The $4 \times 4$ square SE has been stored in a variable `se1`. The results displayed on the command window also indicate that the STREL object contains 16 neighbors, which can be decomposed (for faster execution) into 2 STREL objects of 8 elements each.

We can then use the `getsequence` function to inspect the decomposed structuring elements.

```
>> decomp = getsequence(se1)

decomp =

2x1 array of STREL objects

>> decomp(1)

ans =

Flat STREL object containing 4 neighbors.

Neighborhood:
     1
     1
     1
     1

>> decomp(2)
```

```
ans =

Flat STREL object containing 4 neighbors.

Neighborhood:
     1     1     1     1
```

### ■ EXAMPLE 13.2

This example shows the creation of two additional SEs of different shape and size using `strel`. Note that the rectangular SE does not require decomposition.

```
>> se2 = strel('diamond',4)

se2 =

Flat STREL object containing 41 neighbors.
Decomposition: 3 STREL objects containing a total of 13 neighbors

Neighborhood:
     0     0     0     0     1     0     0     0     0
     0     0     0     1     1     1     0     0     0
     0     0     1     1     1     1     1     0     0
     0     1     1     1     1     1     1     1     0
     1     1     1     1     1     1     1     1     1
     0     1     1     1     1     1     1     1     0
     0     0     1     1     1     1     1     0     0
     0     0     0     1     1     1     0     0     0
     0     0     0     0     1     0     0     0     0

>> se3 = strel('rectangle',[1 3])

se3 =

Flat STREL object containing 3 neighbors.

Neighborhood:
     1     1     1
```

## 13.3 DILATION AND EROSION

In this section, we discuss the two fundamental morphological operations upon which all other operations and algorithms are built: dilation and erosion.

### 13.3.1 Dilation

Dilation is a morphological operation whose effect is to "grow" or "thicken" objects in a binary image. The extent and direction of this thickening are controlled by the size and shape of the structuring element.

Mathematically, the dilation of a set $A$ by $B$, denoted $A \oplus B$, is defined as

$$A \oplus B = \left\{ z | (\hat{B})_z \cap A \neq \emptyset \right\} \tag{13.7}$$

Figure 13.4 illustrates how dilation works using the same input object and three different structuring elements: rectangular SEs cause greater dilation along their longer dimension, as expected.

#### In MATLAB

Morphological dilation is implemented by function `imdilate`, which takes two parameters: an image and a structuring element. In Tutorial 13.1, you will have an opportunity to learn more about this function and apply it to binary images.

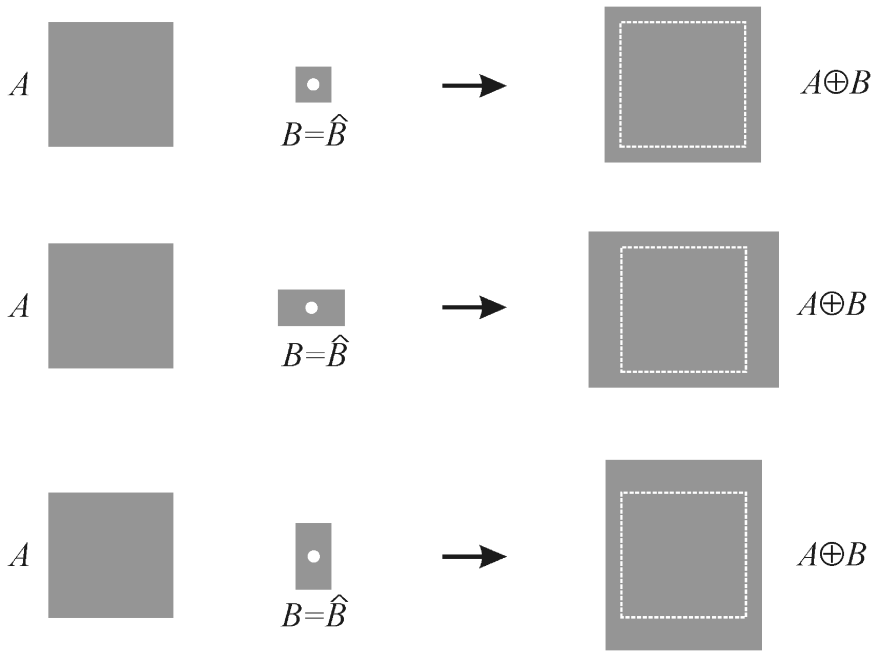


**FIGURE 13.4** Example of dilation using three different rectangular structuring elements.

## ■ EXAMPLE 13.3

This example shows the application of `imdilate` to a small binary test image with two different SEs.

```
>> a = [ 0 0 0 0 0 ; 0 1 1 0 0; 0 1 1 0 0; 0 0 1 0 0; 0 0 0 0 0]

a =

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

>> se1 = strel('square',2)

se1 =

Flat STREL object containing 4 neighbors.

Neighborhood:
     1     1
     1     1


>> b = imdilate(a,se1)

b =

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

The dilation of `a` with SE `se1` (whose reference pixel is the top left corner) caused every pixel in `a` whose value was 1 to be replaced with a $2 \times 2$ square of pixels equal to 1.

```
>> se2 = strel('rectangle', [1 2])

se2 =
```

```
Flat STREL object containing 2 neighbors.

Neighborhood:
     1     1


>> c = imdilate(a,se2)

c =

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

The dilation of `a` with SE `se2` (whose reference pixel is the leftmost one) caused every pixel in `a` whose value was 1 to be replaced with a $1 \times 2$ rectangle of pixels equal to 1, that is, a predominantly horizontal dilation.

### 13.3.2 Erosion

Erosion is a morphological operation whose effect is to "shrink" or "thin" objects in a binary image. The direction and extent of this thinning is controlled by the shape and size of the structuring element.

Mathematically, the erosion of a set $A$ by $B$, denoted $A \ominus B$, is defined as

$$A \ominus B = \left\{ z | (\hat{B})_z \cup A^c \neq \emptyset \right\} \tag{13.8}$$

Figure 13.5 illustrates how erosion works using the same input object and three different structuring elements. Erosion is more severe in the direction of the longer dimension of the rectangular SE.

#### In MATLAB

Morphological erosion is implemented by function `imerode`, which takes two parameters: an image and a structuring element. In Tutorial 13.1, you will have an opportunity to learn more about this function.

#### ■ EXAMPLE 13.4

This example shows the application of `imerode` to a small binary test image with two different SEs.

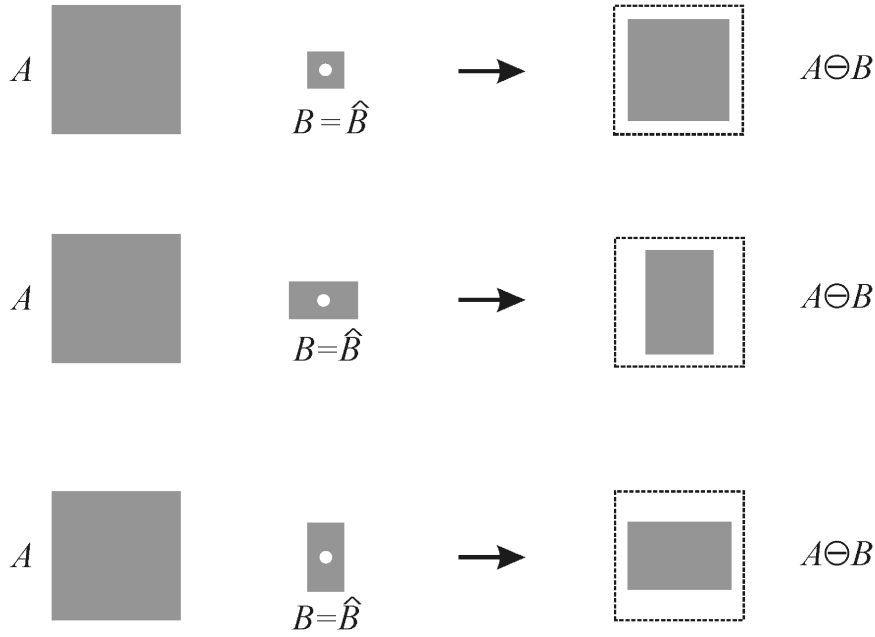


**FIGURE 13.5** Example of erosion using three different rectangular structuring elements.

```
>> a = [ 0 0 0 0 0 ; 0 1 1 1 0; 1 1 1 0 0; 0 1 1 1 1; 0 0 0 0 0]

a =

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

>> se1 = strel('square',2)

se1 =

Flat STREL object containing 4 neighbors.

Neighborhood:
     1     1
     1     1


>> b = imerode(a,se1)
```

```
b =

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

The erosion of `a` with SE `se1` (whose reference pixel is the top left corner) caused the disappearance of many pixels in `a`; the ones that remained were the ones corresponding to the top left corner of a $2 \times 2$ square of pixels equal to 1.

```
>> se2 = strel('rectangle', [1 2])

se2 =

Flat STREL object containing 2 neighbors.

Neighborhood:
     1     1


>> c = imerode(a,se2)

c =

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

The erosion of `a` with SE `se2` (whose reference pixel is the leftmost one) caused the disappearance of many pixels in `a`; the ones that remained were the ones corresponding to the leftmost pixel of a $1 \times 2$ rectangle of pixels equal to 1.

Erosion is the dual operation of dilation and vice versa:

$$(A \ominus B)^c = A^c \oplus \hat{B} \tag{13.9}$$

$$A \oplus B = (A^c \ominus \hat{B})^c \tag{13.10}$$

Erosion and dilation can also be interpreted in terms of whether a SE *hits* or *fits* an image (region), as follows [Eff00].

For dilation, the resulting image $g(x, y)$, given an input image $f(x, y)$ and a SE *se*, will be

$$g(x, y) = \begin{cases} 1 & \text{if } se \text{ hits } f \\ 0 & \text{otherwise} \end{cases} \tag{13.11}$$

for all $x$ and $y$.

For erosion, the resulting image $g(x, y)$, given an input image $f(x, y)$ and a SE *se*, will be

$$g(x, y) = \begin{cases} 1 & \text{if } se \text{ fits } f \\ 0 & \text{otherwise} \end{cases} \tag{13.12}$$

for all $x$ and $y$.

## 13.4 COMPOUND OPERATIONS

In this section, we present morphological operations that combine the two fundamental operations (erosion and dilation) in different ways.

### 13.4.1 Opening

The morphological opening of set $A$ by $B$, represented as $A \circ B$, is the erosion of $A$ by $B$ followed by the dilation of the result by $B$. Mathematically,

$$A \circ B = (A \ominus B) \oplus B \tag{13.13}$$

Alternatively, the opening operation can be expressed using set notation as

$$A \circ B = \bigcup \{(B)_z | (B)_z \subseteq A\} \tag{13.14}$$

where $\bigcup \{\bullet\}$ represents the union of all sets within the curly braces, and the symbol $\subseteq$ means "is a subset of."

The opening operation is *idempotent*, that is, once an image has been opened with a certain SE, subsequent applications of the opening algorithm with the same SE will not cause any effect on the image. Mathematically,

$$(A \circ B) \circ B = A \circ B \tag{13.15}$$

Morphological opening is typically used to remove thin protrusions from objects and to open up a gap between objects connected by a thin bridge without shrinking the objects (as erosion would have done). It also causes a smoothening of the object's contour (Figure 13.6).

The geometric interpretation of the opening operation is straightforward: $A \circ B$ is the union of all translations of $B$ that fit entirely within $A$ (Figure 13.7).

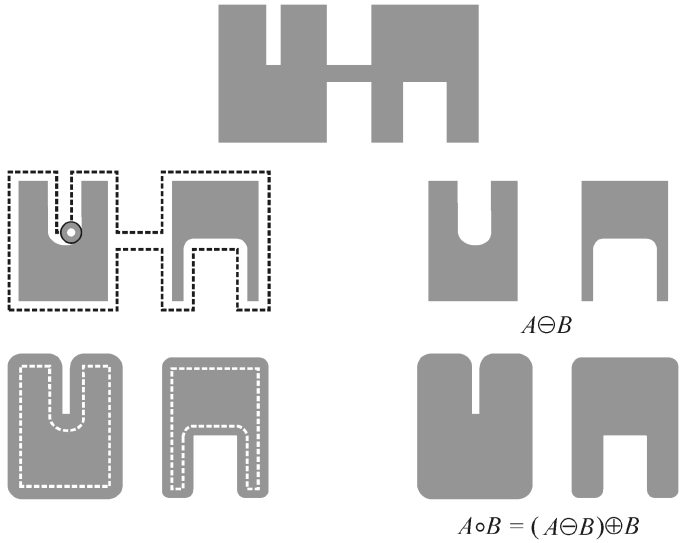


**FIGURE 13.6** Example of morphological opening.

### In MATLAB

Morphological opening is implemented by function `imopen`, which takes two parameters: input image and structuring element. In Tutorial 13.1, you will have an opportunity to learn more about this function.

### 13.4.2 Closing

The morphological closing of set $A$ by $B$, represented as $A \bullet B$, is the dilation of $A$ by $B$ followed by the erosion of the result by $B$. Mathematically,

$$A \bullet B = (A \oplus B) \ominus B \tag{13.16}$$

Just as with opening, the closing operation is *idempotent*, that is, once an image has been closed with a certain SE, subsequent applications of the opening algorithm

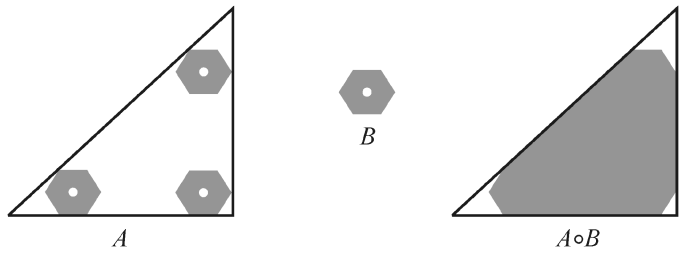


**FIGURE 13.7** Geometric interpretation of the morphological opening operation.

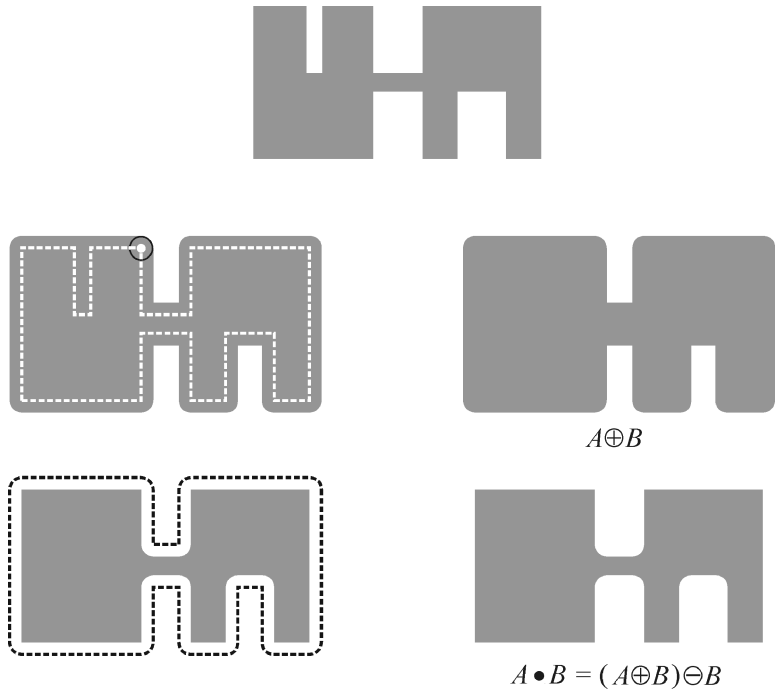


**FIGURE 13.8** Example of morphological closing.

with the same SE will not cause any effect on the image. Mathematically,

$$(A \bullet B) \bullet B = A \bullet B \tag{13.17}$$

Morphological closing is typically used to fill small holes, fuse narrow breaks, and close thin gaps in the objects within an image, without changing the objects' size (as dilation would have done). It also causes a smoothening of the object's contour (Figure 13.8).

The geometric interpretation of the closing operation is as follows: $A \bullet B$ is the complement of the union of all translations of $B$ that do not overlap $A$ (Figure 13.9).

Closing is the dual operation of opening and vice versa:

$$A \bullet B = (A^c \circ B)^c \tag{13.18}$$

$$A \circ B = (A^c \bullet B)^c \tag{13.19}$$

## In MATLAB

Morphological closing is implemented by function `imclose`, which takes two parameters: input image and structuring element. In Tutorial 13.1, you will have an opportunity to learn more about this function.

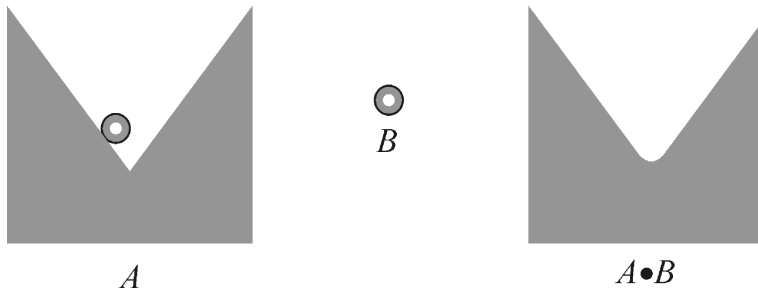


**FIGURE 13.9** Geometric interpretation of the morphological closing operation. Adapted and redrawn from [GW08].

### 13.4.3 Hit-or-Miss Transform

The hit-or-miss (HoM) transform is a combination of morphological operations that uses two structuring elements ($B_1$ and $B_2$) designed in such a way that the output image will consist of all locations that match the pixels in $B_1$ (a *hit*) and that have none of the pixels in $B_2$ (a *miss*). Mathematically, the HoM transform of image $A$ by the structuring element set $B$ ($B = (B_1, B_2)$), denoted $A \otimes B$, is defined as

$$A \otimes B = (A \ominus B_1) \cap (A^c \ominus B_2) \tag{13.20}$$

Alternatively, the HoM transform can be expressed as

$$A \otimes B = (A \ominus B_1) - (A \oplus \hat{B}_2) \tag{13.21}$$

Figure 13.10[1] shows how the HoM transform can be used to locate squares of a certain size in a binary image. Figure 13.10a shows the input image ($A$), consisting of two squares against a background, and Figure 13.10b shows its complement ($A^c$). The two structuring elements, in this case, have been chosen to be a square of the same size as the smallest square in the input image, $B_1$ (Figure 13.10c), and a square of the larger size, indicating that the smallest square should be surrounded by pixels of opposite color, $B_2$ (Figure 13.10d). Figure 13.10e shows the partial result after the first erosion: the smallest square is hit at a single spot, magnified, and painted red for viewing purposes, whereas the largest square is hit at many points. Figure 13.10f shows the final result, containing the set of all points[2] for which the HoM transform found a hit for $B_1$ in $A$ *and* a hit for $B_2$ in $A^c$ (which is equivalent to a miss for $B_2$ in $A$).

#### In MATLAB

The binary hit-or-miss transformation is implemented by function `bwhitmiss`, whose basic syntax is `J = bwhitmiss(I, B1, B2)`, where `I` is the input

[1]The thin black borders in parts (a), (d), (e), and (f) were added for viewing purposes only.

[2]In this case, this set contains only one point, which was enlarged and painted red for viewing purposes.

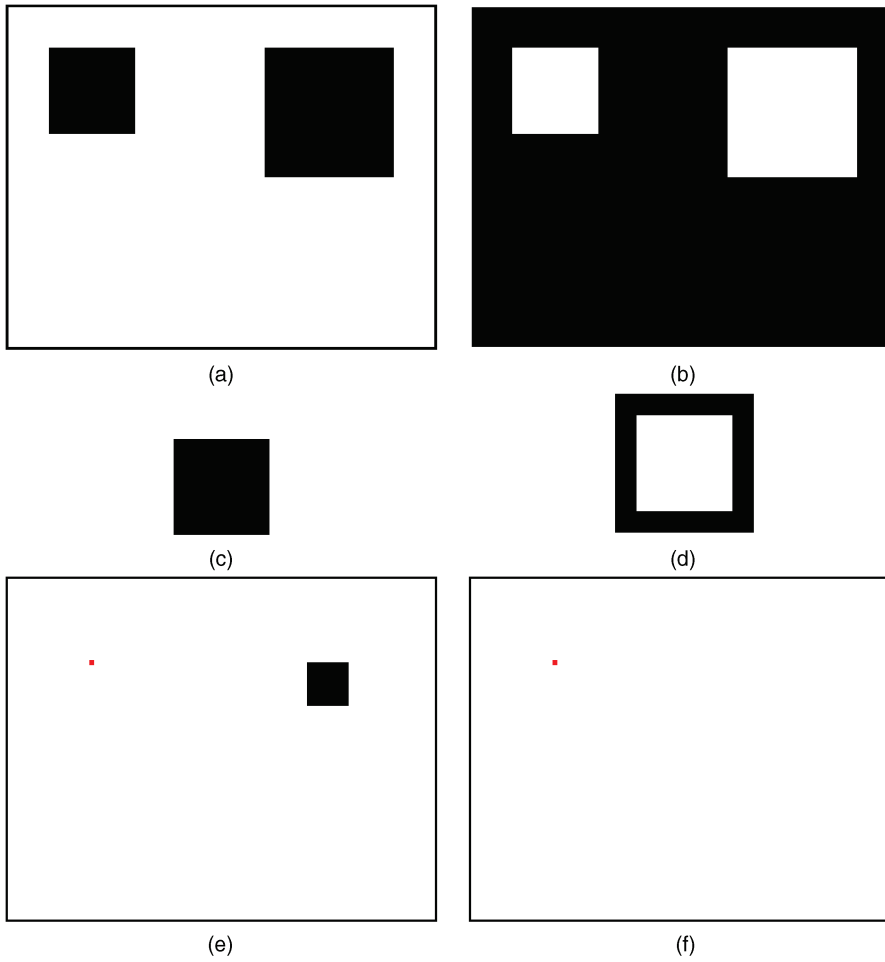


**FIGURE 13.10** Example of HoM transform.

image, `B1` and `B2` are the structuring elements, and `J` is the resulting image. You will learn how to use this function in Tutorial 13.1.

## 13.5 MORPHOLOGICAL FILTERING

Morphological filters are Boolean filters that apply a many-to-one binary (or Boolean) function $h$ within a window $W$ in the binary input image $f(x, y)$, producing at the output an image $g(x, y)$ given by

$$g(x, y) = h\left[Wf(x, y)\right] \tag{13.22}$$

Examples of Boolean operations (denoted as $h$ in equation (13.22)) are as follows:

- OR: This (as we have seen in Section 13.2) is equivalent to a morphological dilation with a square SE of the same size as $W$.
- AND: This (as we have seen in Section 13.2) is equivalent to a morphological erosion with a square SE of the same size as $W$.
- MAJ (Majority): This is the morphological equivalent to a median filter (introduced in Section 12.3.2) applicable to binary images.

Morphological filters can also be used in noise reduction. Let $A$ be a binary image corrupted by impulse (salt and pepper) noise. The application of a morphological opening operation followed by a morphological closing will remove a significant amount of noise present in the input image, resulting in an image $C$ given by

$$C = ((A \circ B) \bullet B) \tag{13.23}$$

where $B$ is the chosen SE.

■ **EXAMPLE 13.5**

Figure 13.11 shows an example of morphological filtering for noise removal on a binary image using a circular SE of radius equal to 2 pixels. Part (a) shows the ($641 \times 535$) input image, contaminated by salt and pepper noise. Part (b) shows the partial result (after the opening operation): at this stage, all the "salt" portion of the noise has been removed, but the "pepper" noise remains. Finally, part (c) shows the result of applying closing to the result of the opening operation. The noise has been completely removed, at the expense of imperfections at the edges of the objects. Such imperfections would be even more severe for larger SEs, as demonstrated in part (d), where the radius of the circular SE is equal to 4 pixels.

## 13.6 BASIC MORPHOLOGICAL ALGORITHMS

In this section, we present a collection of simple and useful morphological algorithms and show how they can be implemented using MATLAB and the IPT. Several of these algorithms will also be covered in Tutorial 13.2.

### In MATLAB

The IPT function `bwmorph` implements a number of useful morphological operations and algorithms, listed in Table 13.1. This function takes three arguments: input image, desired operation, and the number of times the operation is to be repeated.

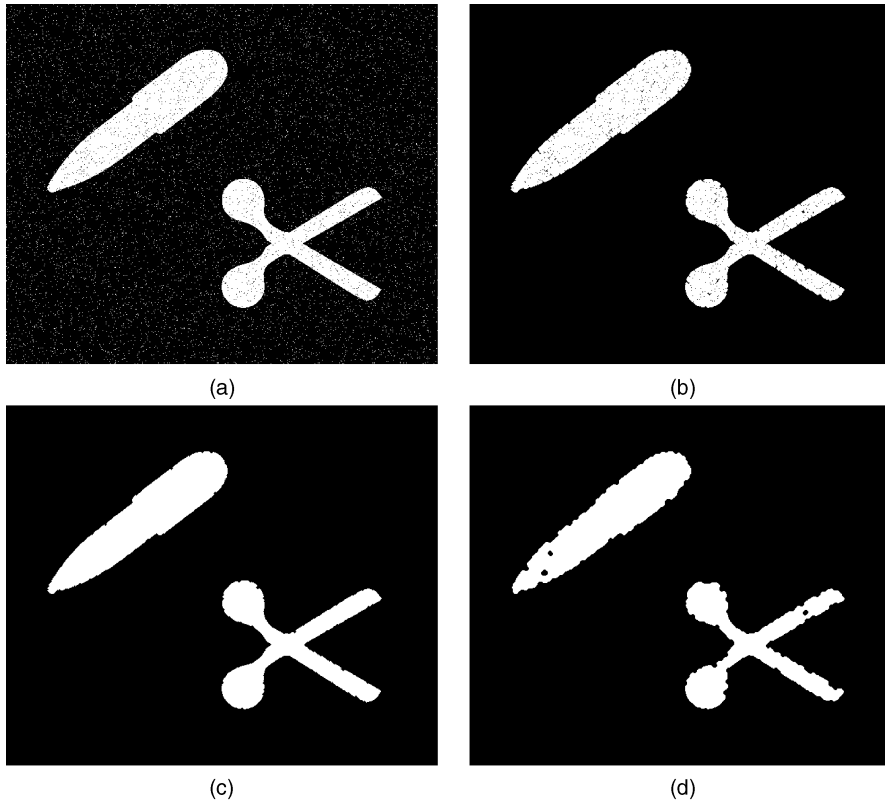


**FIGURE 13.11** Morphological filtering. (a) input (noisy) image; (b) partial result (after opening) with SE of radius = 2 pixels; (c) final result with SE of radius = 2 pixels; (d) final result with SE of radius = 4 pixels.

## ■ EXAMPLE 13.6

In this example, we use `bwmorph` to apply different operations to the same small test image (Figure 13.12a).[3]

Here is the sequence of steps used to obtain the results in Figure 13.12(b–f):

```
B = bwmorph(A,'skel', Inf);
C = bwmorph(B,'spur',Inf);
D = bwmorph(A,'remove');
E = bwmorph(D,'thicken',3);
F = bwmorph(E,'thin',3);
```

[3]The goal is to give you a quick glance at some of the most useful morphological algorithms and their effect on an input test image. A detailed formulation of each operation is beyond the scope of this book. Refer to "Learn More About It" section at the end of the chapter.

**TABLE 13.1 Operations Supported by bwmorph**

| Operation | Description |
|---|---|
| `bothat` | Subtract the input image from its closing |
| `bridge` | Bridge previously unconnected pixels |
| `clean` | Remove isolated pixels (1s surrounded by 0s) |
| `close` | Perform binary closure (dilation followed by erosion) |
| `diag` | Diagonal fill to eliminate 8-connectivity of background |
| `dilate` | Perform dilation using the structuring element 1s(3) |
| `erode` | Perform erosion using the structuring element 1s(3) |
| `fill` | Fill isolated interior pixels (0s surrounded by 1s) |
| `hbreak` | Remove H-connected pixels |
| `majority` | Set a pixel to 1 if five or more pixels in its $3 \times 3$ neighborhood are 1s |
| `open` | Perform binary opening (erosion followed by dilation) |
| `remove` | Set a pixel to 0 if its 4-connected neighbors are all 1s, thus leaving only boundary pixels |
| `shrink` | With N = Inf, shrink objects to points; shrink objects with holes to connected rings |
| `skel` | With N = Inf, remove pixels on the boundaries of objects without allowing objects to break apart |
| `spur` | Remove endpoints of lines without removing small objects completely |
| `thicken` | With N = Inf, thicken objects by adding pixels to the exterior of objects without connecting previously unconnected objects |
| `thin` | With N = Inf, remove pixels so that an object without holes shrinks to a minimally connected stroke, and an object with holes shrinks to a ring halfway between the hold and outer boundary |
| `tophat` | Subtract the opening from the input image |

### 13.6.1 Boundary Extraction

Dilation, erosion, and set difference operations can be combined to perform boundary extraction of a set $A$, denoted by $\mathcal{BE}(A)$, as follows:

- *Internal Boundary*: Consists of the pixels in $A$ that sit at the edge of $A$.

$$\mathcal{BE}(A) = A - (A \ominus B) \tag{13.24}$$

- *External Boundary*: Consists of the pixels outside $A$ that sit immediately next to $A$.

$$\mathcal{BE}(A) = (A \oplus B) - A \tag{13.25}$$

- *Morphological Gradient*: Consists of the combination of internal and external boundaries.

$$\mathcal{BE}(A) = (A \oplus B) - (A \ominus B) \tag{13.26}$$

where $B$ is a suitable structuring element.

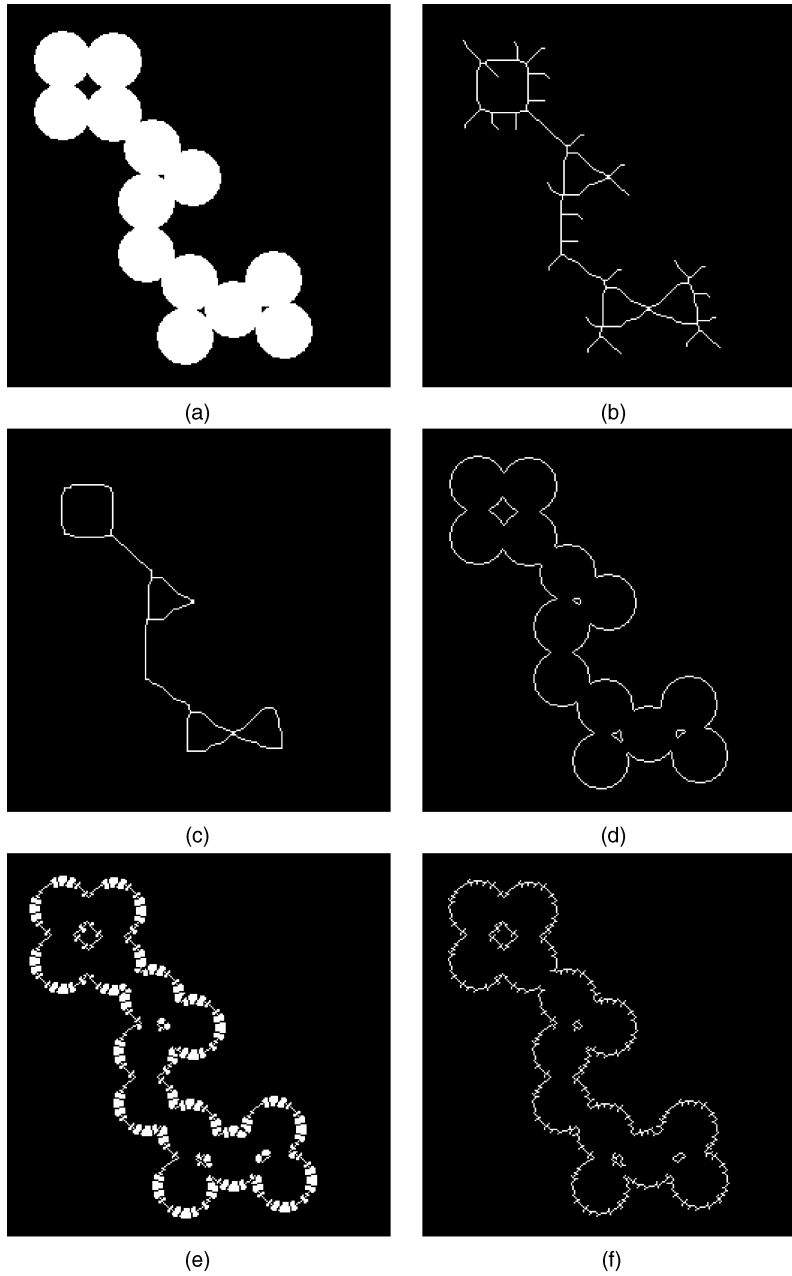


**FIGURE 13.12** Morphological algorithms. (a) input image; (b) skeleton of (a); (c) pruning spurious pixels from (b); (d) removing interior pixels from (a); (e) thickening the image in (d); (f) thinning the image in (e). Original image: courtesy of MathWorks.

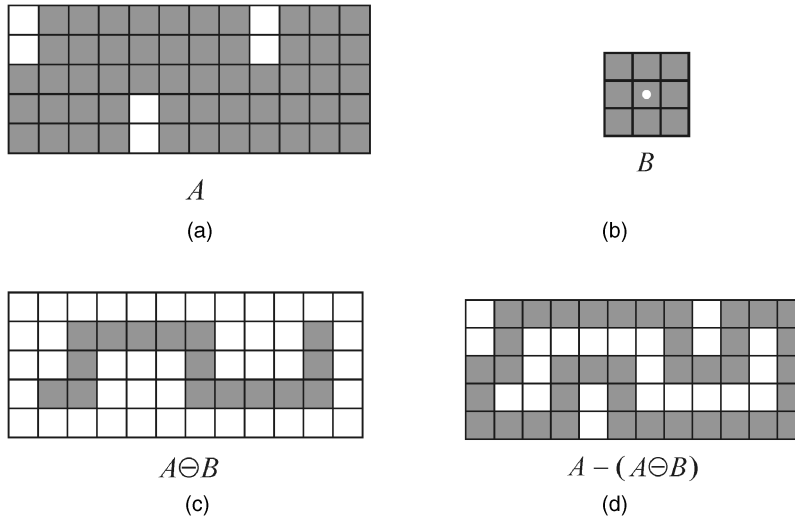


**FIGURE 13.13** Boundary extraction.

Figure 13.13 shows an example of internal boundary extraction of a small (5 × 12) object using a 3 × 3 square as SE.

### In MATLAB

The IPT function `bwperim` returns a binary image containing only the perimeter pixels of objects in the input image.

### ■ EXAMPLE 13.7

This example shows the application of `bwperim` (with 8-connectivity, to be consistent with the SE in Figure 13.13b) to extract the internal boundary of a small binary test image identical to the image in Figure 13.13a.

```
a = ones(5,12)
a (1:2,1)=0
a (1:2,9)=0
a (4:5,5)=0
b = bwperim(a,8)
```

When you execute the steps above, you will confirm that the result (stored in variable `b`) is identical to the image in Figure 13.13d.

## 13.6.2 Region Filling

In this section, we present an algorithm that uses morphological and set operations to fill regions (which can be thought of as *holes*) in a binary image.

Let $p$ be a pixel in a region surrounded by an 8-connected boundary, $A$. The goal of a region filling algorithm is to fill up the entire region with 1s using $p$ as a starting point (i.e., setting it as 1). Region filling can be accomplished using an iterative procedure, mathematically expressed as follows:

$$X_k = (X_{k-1} \oplus B) \cap A^c \quad k = 1, 2, 3, \ldots \tag{13.27}$$

where $X_0 = p$ and $B$ is the cross-shaped structuring element. The algorithm stops at the $k$th iteration if $X_k = X_{k-1}$. The union of $X_k$ and $A$ contains the original boundary ($A$) and all the pixels within it labeled as 1.

Figure 13.14 illustrates the process. Part (a) shows the input image, whose complement is in part (b). Part (c) shows the partial results of the algorithm after each iteration (the number inside the square): the initial pixel ($p$, top left) corresponds to iteration 0. This example requires six iterations to complete and uses the SE shown in part (e). Part (d) shows the union of $X_6$ and $A$.

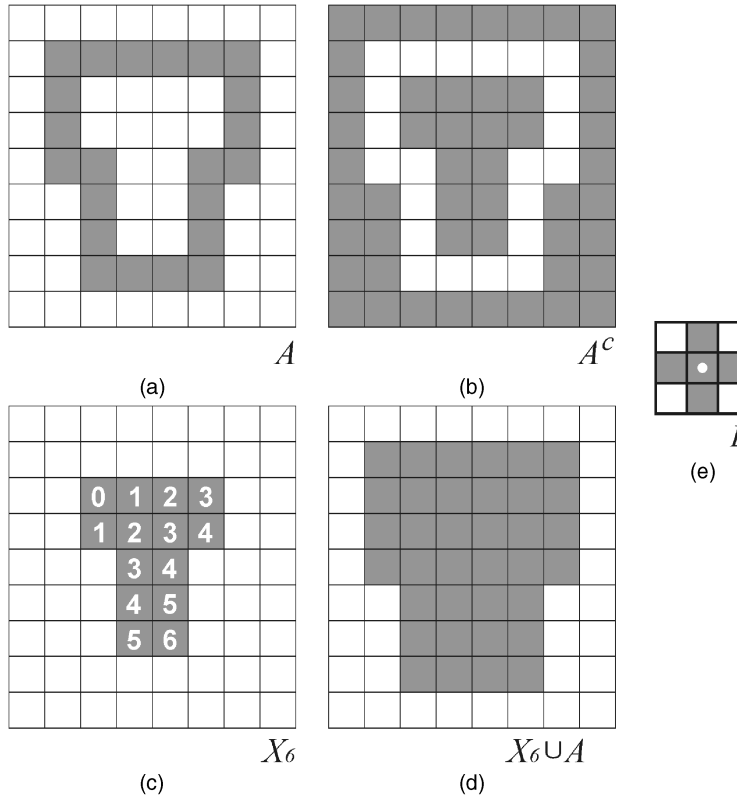


**FIGURE 13.14** Region filling: (a) input image; (b) complement of (a); (c) partial results (numbered according to the iteration in the algorithm described by equation (13.27); (d) final result; (e) structuring element.

### In MATLAB

The IPT function `imfill` implements region filling. It can be used in an interactive mode (where the user clicks on the image pixels that should be used as starting points) or by passing the coordinates of the starting points. You will use `imfill` in Tutorial 13.2.

### 13.6.3 Extraction and Labeling of Connected Components

Morphological concepts and operations can be used to extract and label connected components[4] in a binary image.

The morphological algorithm for extraction of connected components is very similar to the region filling algorithm described in Section 13.6.2. Let $A$ be a set containing one or more connected components and $p$ ($p \in A$) be a starting pixel. The process of finding all other pixels in a component can be accomplished using an iterative procedure, mathematically expressed as follows:

$$X_k = (X_{k-1} \oplus B) \cap A, \quad k = 1, 2, 3, \ldots \tag{13.28}$$

where $X_0 = p$ and $B$ is a suitable structuring element: cross-shaped for 4-connectivity, $3 \times 3$ square for 8-connectivity.

The algorithm stops at the $k$th iteration if $X_k = X_{k-1}$.

Figure 13.15 shows an example of extraction of connected components. Part (a) shows the initial set $A$ and the starting pixel $p$, represented by the number 0. Part (e) shows the SE used in this case. Parts (b) and (c) show the result of the first and second iterations, respectively. The final result (after six iterations) is shown in Figure 13.15d.

### In MATLAB

The IPT has a very useful function for computing connected components in a binary image: `bwlabel`. The function takes two parameters (input image and connectivity criterion: 4 or 8 (default)) and returns a matrix of the same size as the input image, containing labels for the connected objects in the image. Pixels labeled 0 correspond to the background; pixels labeled 1 and higher correspond to the connected components in the image. You will learn how to use this function in Tutorial 13.2.

Visualization of the connected components extracted by `bwlabel` is often achieved by pseudocoloring the label matrix (assigning a different color to each component) using the `label2rgb` function.

The IPT also has a function for selecting objects, that is, connected components, in a binary image: `bwselect`. It can be used in an interactive mode (where the user clicks on the image pixels that should be used as starting points) or by passing the coordinates of the starting points. You will use `bwselect` in Tutorial 13.2.

[4]The concept of *connected component* (a set of pixels that are 4- or 8- connected to each other) was introduced in Section 2.3.

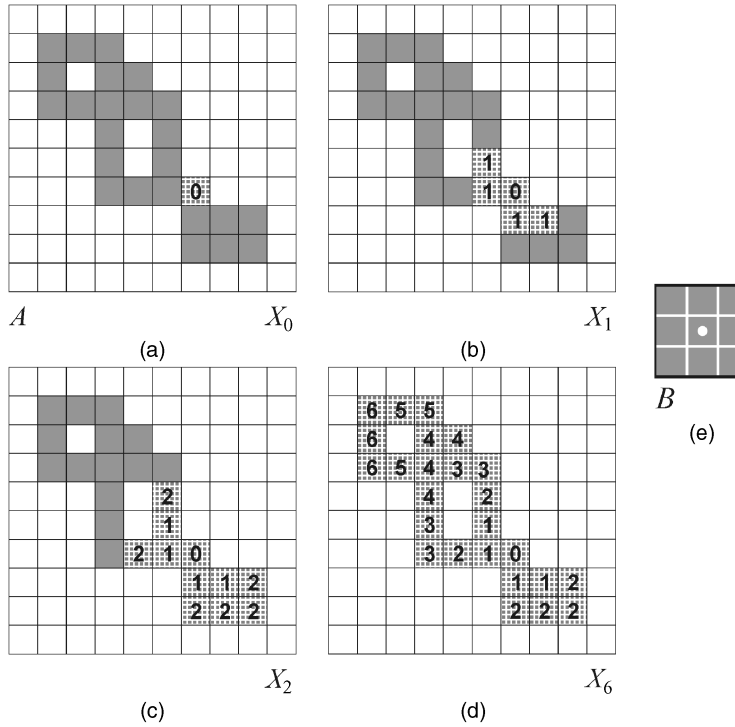


**FIGURE 13.15** Extraction of connected components: (a) input image; (b) first iteration; (c) second iteration; (d) final result, showing the contribution of each iteration (indicated by the numbers inside the squares); (e) structuring element.

## 13.7 GRAYSCALE MORPHOLOGY

Many morphological operations originally developed for binary images can be extended to grayscale images. This section presents the grayscale version of the basic morphological operations (erosion, dilation, opening, and closing) and introduces the top-hat and bottom-hat transformations.

The mathematical formulation of grayscale morphology uses an input image $f(x, y)$ and a structuring element (SE) $b(x, y)$. Structuring elements in grayscale morphology come in two categories: nonflat and flat.

### In MATLAB

Nonflat SEs can be created with the same function used to create flat SEs: `strel`. In the case of nonflat SEs, we must also pass a second matrix (containing the height values) as a parameter.

### 13.7.1 Dilation and Erosion

The dilation of an image $f(x, y)$ by a flat SE $b(x, y)$ is defined as

$$[A \oplus B](x, y) = \max\{f(x + s, y + t) | (s, t) \in D_b\} \tag{13.29}$$

where $D_b$ is called the *domain* of $b$.
For a nonflat SE, $b_N(x, y)$, equation (13.29) becomes

$$[A \oplus B](x, y) = \max\{f(x + s, y + t) + b_N(s, t) | (s, t) \in D_b\} \tag{13.30}$$

Some references use $f(x - s, y - t)$ in equations (13.29) and (13.30), which just requires the SE to be rotated by 180° or mirrored around its origin, that is, $\hat{B}(x, y) = b(-x, -y)$.

The erosion of an image $f(x, y)$ by a flat SE $b(x, y)$ is defined as

$$[A \ominus B](x, y) = \min\{f(x + s, y + t) | (s, t) \in D_b\} \tag{13.31}$$

where $D_b$ is called the *domain* of b.
For a nonflat SE, $b_N(x, y)$, equation (13.31) becomes

$$[A \ominus B](x, y) = \min\{f(x + s, y + t) - b_N(s, t) | (s, t) \in D_b\} \tag{13.32}$$

■ **EXAMPLE 13.8**

Figure 13.16 shows examples of grayscale erosion and dilation with a nonflat ball-shaped structuring element with radius 5 (`se = strel('ball',5,5);`).

### 13.7.2 Opening and Closing

Grayscale opening and closing are as defined exactly as they were for binary morphology.

The opening of grayscale image $f(x, y)$ by SE $b(x, y)$ is given by

$$f \circ b = (f \ominus b) \oplus b \tag{13.33}$$

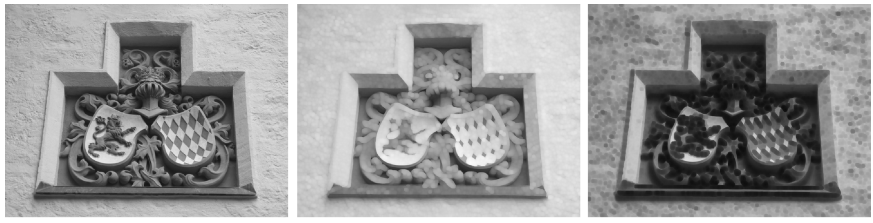


**FIGURE 13.16** Grayscale erosion and dilation with a nonflat ball-shaped structuring element with radius 5: (a) input image; (b) result of dilation; (c) result of erosion.

The closing of grayscale image $f(x, y)$ by SE $b(x, y)$ is given by

$$f \bullet b = (f \oplus b) \ominus b \tag{13.34}$$

Grayscale opening and closing are used in combination for the purpose of smoothing and noise reduction, similar to what we saw in Section 13.5 for binary images. This technique, known as *morphological smoothing*, is discussed in the following example.

## ■ EXAMPLE 13.9

Figure 13.17 shows examples of applying grayscale opening and closing with a flat disk-shaped structuring element with radius 3 (`se = strel('disk',3);`) for noise reduction. The upper part of the figure refers to an input image corrupted by

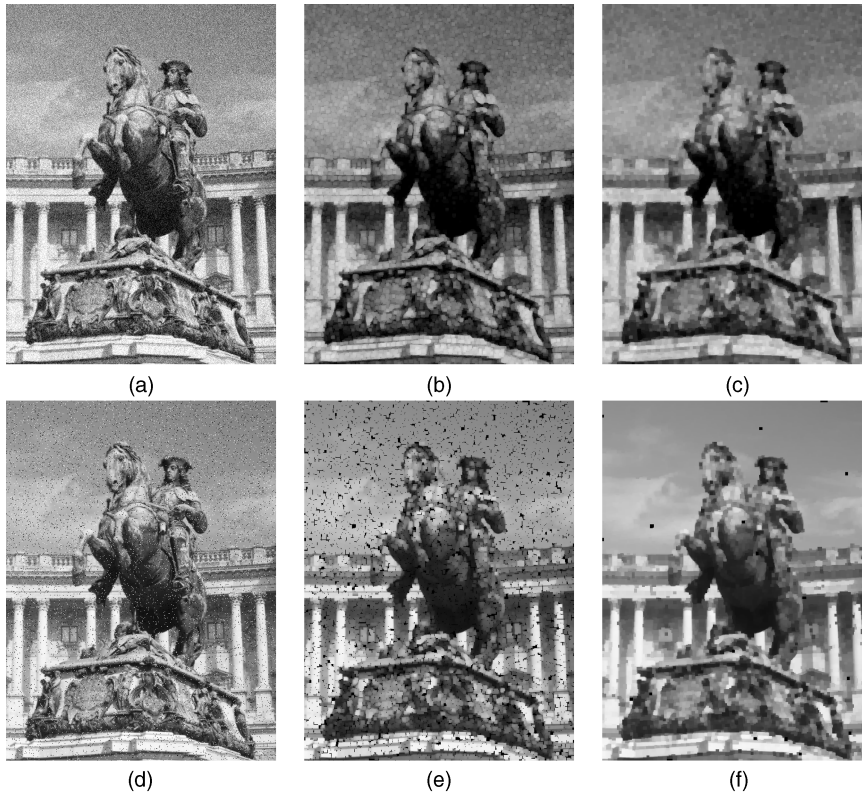


**FIGURE 13.17** Grayscale opening and closing with a flat disk-shaped structuring element with radius 3: (a) input image (Gaussian noise); (b) result of opening image (a); (c) result of closing image (b); (d) input image (salt and pepper noise); (e) result of opening image (d); (f) result of closing image (e).

Gaussian noise of mean 0 and variance 0.01, whereas the bottom part refers to an input image corrupted by salt and pepper noise.

### 13.7.3 Top-Hat and Bottom-Hat Transformations

The top-hat and bottom-hat transformations are operations that combine morphological openings and closings with image subtraction. The top-hat transformation is often used in the process of *shading correction*, which consists in compensating for nonuniform illumination of the scene. Top-hat and bottom-hat operations can be combined for contrast enhancement purposes.

Mathematically, the top-hat transformation of a grayscale image $f$ is defined as the subtraction of $f$ from its opening with SE $b$:

$$\text{Top-hat}\,(f) = f - (f \circ b) \tag{13.35}$$

and the bottom-hat transformation of $f$ is given by the subtraction of the closing of $f$ with SE $b$ from $f$:

$$\text{Bottom-hat}\,(f) = (f \bullet b) - f \tag{13.36}$$

#### In MATLAB

IPT functions `imtophat` and `imbothat` implement top-hat and bottom-hat filtering, respectively.

#### ■ EXAMPLE 13.10

Figure 13.18 shows an example of applying `imtophat` and `imbothat` to improve the contrast of an input image `I` using a flat disk-shaped SE of radius 3. The output image was obtained as follows:

```
J = imsubtract(imadd(I,imtophat(I,se)), imbothat(I,se));
```

## 13.8 TUTORIAL 13.1: BINARY MORPHOLOGICAL IMAGE PROCESSING

### Goal

The goal of this tutorial is to learn how to implement the basic binary morphological operations in MATLAB.

### Objectives

- Learn how to dilate an image using the `imdilate` function.
- Learn how to erode an image using the `imerode` function.

(a) (b)

**FIGURE 13.18** Example of using top-hat and bottom-hat filtering for contrast improvement: (a) input image; (b) output image.

- Learn how to open an image using the `imopen` function.
- Learn how to close an image using the `imclose` function.
- Explore the hit-or-miss transformation using the `bwhitmiss` function.

### Procedure

1. Load and display the `blobs` test image.

```
I = imread('blobs.png');
figure, imshow(I), title('Original Image');
```

### Dilation

2. Create a $3 \times 3$ structuring element with all coefficients equal to 1.

```
SE_1 = strel('square',3);
```

3. Perform dilation using the generated SE and display the results.

```
I_dil_1 = imdilate(I,SE_1);
figure,  imshow(I_dil_1), title('Dilated with 3x3');
```

***Question 1*** What happened when we dilated the image with a square $3 \times 3$ SE?

Let us now see what happens when using a SE shape other than a square.

4. Create a $1 \times 7$ SE with all elements equal to 1 and dilate the image.

```
SE_2 = strel('rectangle', [1 7]);
I_dil_2 = imdilate(I, SE_2);
figure, imshow(I_dil_2), title('Dilated with 1x7');
```

***Question 2*** What is the difference in results between the dilation using the $3 \times 3$ SE and the $1 \times 7$ SE?

***Question 3*** How would the results change if we use a $7 \times 1$ SE? Verify your prediction.

***Question 4*** What other SE shapes does the `strel` function support?

### Erosion

The procedure for erosion is similar to that for dilation. First, we create a structuring element with the `strel` function, followed by eroding the image using the `imerode` function. We will use the same two SEs already created in the previous steps.

5. Erode the original image with a $3 \times 3$ structuring element and display the results.

```
I_ero_1 = imerode(I, SE_1);
figure, imshow(I), title('Original Image');
figure, imshow(I_ero_1), title('Eroded with 3x3');
```

6. Erode the original image with a $1 \times 7$ structuring element.

```
I_ero_2 = imerode(I, SE_2);
figure, imshow(I_ero_2), title('Eroded with 1x7');
```

***Question 5*** What is the effect of eroding the image?

***Question 6*** How does the size and shape of the SE affect the results?

### Opening

7. Perform morphological opening on the original image using the square $3 \times 3$ SE created previously.

```
I_open_1 = imopen(I, SE_1);
figure, imshow(I), title('Original Image');
figure, imshow(I_open_1), title('Opening the image');
```

***Question 7*** What is the overall effect of opening a binary image?

8. Compare opening with eroding.

```
figure, subplot(2,2,1), imshow(I), title('Original Image');
subplot(2,2,2), imshow(I_ero_1), title('Result of Erosion');
subplot(2,2,3), imshow(I_open_1), title('Result of Opening (3x3)');
```

***Question 8*** How do the results of opening and erosion compare?

9. Open the image with a $1 \times 7$ SE.

```
I_open_2 = imopen(I, SE_2);
subplot(2,2,4), imshow(I_open_2), title('Result of Opening (1x7)');
```

***Question 9*** How does the shape of the SE affect the result of opening?

### Closing

10. Create a square $5 \times 5$ SE and perform morphological closing on the image.

```
SE_3 = strel('square',5);
I_clo_1 = imclose(I, SE_3);
figure,  imshow(I), title('Original Image');
figure,  imshow(I_clo_1), title('Closing the image');
```

***Question 10*** What was the overall effect of closing this image?

11. Compare closing with dilation.

```
figure, imshow(I), title('Original Image');
figure, imshow(I_dil_1), title('Dilating the image');
figure, imshow(I_clo_1), title('Closing the image');
```

***Question 11*** How does closing differ from dilation?

### Hit-or-Miss Transformation

The HoM transformation is implemented through the `bwhitmiss` function, which takes three variables: an image and two structuring elements. Technically, the operation will keep pixels whose neighbors match the first structuring element, and will also keep any pixels whose neighbors do not match the second structuring element. This justifies the name of the operations: a hit for the first structuring element or a miss for the second structuring element.

Normally we would use `strel` to generate the structuring element; but in this case, we are not simply generating a matrix of 1s. Therefore, we will define the structuring elements manually.

12. Close any open windows.
13. Define the two structuring elements.

```
 SE1 = [    0 0 0 0 0
        0 0 0 0 0
        0 1 1 0 0
        0 0 1 0 0
        0 0 0 0 0]

SE2 = [ 0 0 0 0 0
        1 1 1 1 0
        0 0 0 1 0
        0 0 0 1 0
        0 0 0 1 0]
```

14. Apply the HoM operation on the original image.

```
I_hm = bwhitmiss(I,SE1,SE2);
figure,  imshow(I), title('Original Image');
figure,  imshow(I_hm), title('Hit-or-miss operation');
```

***Question 12*** What was the result of applying the HoM operation to the image with the given structuring elements?

***Question 13*** How could we define the structuring elements so that the bottom left corner pixels of objects are shown in the result?

The `bwhitmiss` function offers a shortcut for creating structuring elements like the one above. Because the two structuring elements above do not have any values in common, we can actually use one array to represent both structuring elements.

In the array in Figure 13.19, 1 refers to ones in the first structuring element, $-1$ refers to ones in the second structuring element, and all 0s are ignored. This array is called an *interval*.

15. Define an equivalent interval to that of the two structuring elements previously defined.

```
interval = [ 0 0 0 0 0
             -1 -1 -1 -1 0
             0 1 1 -1 0
             0 0 1 -1 0
             0 0 0 -1 0]
```

$$Interval = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & -1 & -1 & 0 \\ 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 \end{bmatrix}$$

**FIGURE 13.19** Combining two structuring elements into one for the HoM transformation.

```
I_hm2 = bwhitmiss(I,interval);
figure, imshow(I_hm), title('Using two SEs');
figure, imshow(I_hm2), title('Using interval');
```

***Question 14*** Will this technique work if the two structuring elements have elements in common? Explain.

## 13.9 TUTORIAL 13.2: BASIC MORPHOLOGICAL ALGORITHMS

### Goal

The goal of this tutorial is to learn how to implement basic morphological algorithms in MATLAB.

### Objectives

- Learn how to perform boundary extraction.
- Explore the `bwperim` function.
- Learn how to fill object holes using the `imfill` function.
- Explore object selection using the `bwselect` function.
- Learn how to label objects in a binary image using the `bwlabel` function.
- Explore the `bwmorph` function to perform thinning, thickening, and skeletonization.

### What You Will Need

- `morph.bmp`

### Procedure

### Boundary Extraction

1. Load and display the test image.

```
I = imread('morph.bmp');
figure, imshow(I), title('Original image');
```

2. Subtract the original image from its eroded version to get the boundary image.

```
se = strel('square',3);
I_ero = imerode(I,se);
I_bou = imsubtract(I,I_ero);
figure, imshow(I_bou), title('Boundary Extraction');
```

3. Perform boundary extraction using the `bwperim` function.

```
I_perim = bwperim(I,8);
figure, imshow(I_perim), title('Boundary using bwperim');
```

***Question 1*** Show that the `I_perim` image is exactly the same as `I_bou`.

***Question 2*** If we specify 4-connectivity in the `bwperim` function call, how will this affect the output image?

### Region Filling

We can use the `imfill` function to fill holes within objects (among other operations).

4. Close any open figures.
5. Fill holes in the image using the `imfill` function.

```
I_fill1 = imfill(I,'holes');
figure, imshow(I_fill1), title('Holes filled');
```

Function `imfill` can also be used in an interactive mode.

6. Pick two of the three holes interactively by executing this statement. After selecting the points, press Enter.

```
I_fill2 = imfill(I);
imshow(I_fill2), title('Interactive fill');
```

***Question 3*** What other output parameters can be specified when using `imfill`?

### Selecting and Labeling Objects

The `bwselect` function allows the user to interactively select connected components—which often correspond to objects of interest—in a binary image.

7. Close any open figures.
8. Select any of the white objects and press Enter.

```
bwselect(I);
```

***Question 4*** In the last step, we did not save the output image into a workspace variable. Does this function allow us to do this? If so, what is the syntax?

In many cases, we need to label the connected components so that we can reference them individually. We can use the function `bwlabel` for this purpose.

9. Label the objects in an image using `bwlabel`.

```
I_label = bwlabel(I);
figure, imshow(I_label,[]), title('Labeled image');
```

***Question 5*** What do the different shades of gray represent when the image is displayed?

***Question 6*** What other display options can you choose to make it easier to tell the different labeled regions apart?

### Thinning

Thinning is one of the many operations that can be achieved through the use of the `bwmorph` function (see Table 13.1 for a complete list).

10. Use `bwmorph` to thin the original image with five iterations.

```
I_thin = bwmorph(I,'thin',5);
figure, imshow(I_thin), title('Thinning, 5 iterations');
```

***Question 7*** What happens if we specify 10, 15, or `Inf` (infinitely many) iterations instead?

### Thickening

11. Thicken the original image with five iterations.

```
I_thick = bwmorph(I,'thicken',5);
figure, imshow(I_thick), title('Thicken, 5 iterations');
```

***Question 8*** What happens when we specify a higher number of iterations?

***Question 9*** How does MATLAB know when to stop thickening an object? (*Hint*: Check to see what happens if we use `Inf` (infinitely many) iterations?

### Skeletonization

12. Close any open figures.
13. Generate the skeleton using the `bwmorph` function.

```
I_skel = bwmorph(I,'skel',Inf);
figure, imshow(I_skel), title('Skeleton of image');
```

***Question 10*** How does skeletonization compare with thinning? Explain.

## WHAT HAVE WE LEARNED?

- Mathematical morphology is a branch of image processing that has been successfully used to represent, describe, and analyze shapes in images. It offers useful tools for extracting image components, such as boundaries and skeletons, as well as pre- or postprocessing images containing shapes of interest.
- The main morphological operations are erosion (implemented by the `imerode` function in MATLAB), dilation (`imdilate`), opening (`imopen`), closing (`imclose`), and the hit-or-miss transform (`bwhitmiss`).
- The structuring element is the basic neighborhood structure associated with morphological image operations. It is usually represented as a small matrix, whose shape and size impact the results of applying a certain morphological operator to an image.
- Morphological operators can be combined into useful algorithms for commonly used image processing operations, for example, boundary extraction, region filling, and extraction (and labeling) of connected components within an image.

## LEARN MORE ABOUT IT

- The books by Serra ( [SC82], [Ser88], and [SS96]) are considered the primary references in the field of mathematical morphology.
- Book-length treatment of the topics discussed in this chapter can also be found in [GD88] and [Dou92], among others.
- Section 14.2 of [Pra07] and Section 9.3.3 of [GWE04] discuss the implementation of the HoM transform using lookup tables (LUTs).
- Chapter 2.2 of [Bov00a] provides examples to illustrate the operation of basic morphological filters.

- Maragos and Pessoa (Chapter 3.3 of [Bov00a]) provide an extensive coverage of morphological filtering and applications.
- For additional discussions and examples of morphological algorithms, refer to Sections 9.5 and 9.6.3 of [GW08] and Section 10.6 of [McA04].
- The topic of morphological reconstruction [Vin93] is discussed in Sections 9.5.9 and 9.6.4 of [GW08] and Sections 9.5 and 9.6.3 of [GWE04].
- The paper by Jang and Chin [JC90] discusses morphological thinning algorithms.
- Section 3.4 of [SS01] and Section 11.1 of [BB08] contain additional algorithms for extraction and labeling of connected components.
- Chapter 13 of [SKH08] has MATLAB implementations of morphological algorithms for object detection, thinning, and granulometry applications, among others.

## 13.10 PROBLEMS

**13.1** Use `strel` and `getsequence` to create SEs of different shape and size and inspect their decomposed structuring elements.

**13.2** Use the equations and definitions from Section 13.2 to prove that equation (13.21) is, indeed, equivalent to equation (13.20).

**13.3** Write a MATLAB script to demonstrate that erosion and dilation are dual operations.

**13.4** Repeat Example 13.7 using MATLAB commands that correspond to equation (13.26).

**13.5** Write a MATLAB script (or function) that extracts the connected components of a binary image and displays the results using different colors for each component and overlays a cross-shaped symbol on top of each component's center of gravity.

**13.6** How many connected components would the `bwlabel` function find, if presented with the negative (i.e., the image obtained by swapping white and black pixels) of the image in Figure 2.8a as an input image? Explain.

**13.7** Explore the MATLAB IPT and try to find at least two other functions—besides `bwlabel`—whose results depend on whether 4- or 8-connectivity has been selected.