Chapter 9

Mathematical morphology (1)

Introduction

Morphology, or *morphology* for short, is a branch of image processing which is particularly useful for analyzing shapes in images. We shall develop basic morphological tools for investigation of binary images, and then show how to extend these tools to greyscale images. MATLAB has many tools for binary morphology in the image processing toolbox; most of which can be used for greyscale morphology as well.

Basic ideas

The theory of mathematical morphology can be developed in many different ways. We shall adopt one standard method which uses operations on sets of points. A very solid and detailed account can be found in Haralick and Shapiro [5].

Translation

Suppose that is a set of pixels in a binary image, and is a particular coordinate point. Then is the set "translated" in direction . That is

For example, in figure 1, is the cross shaped set, and . The set has been shifted in the and directions by the values given in . Note that here we are using matrix coordinates, rather than Cartesian coordinates, so that the origin is at the top left, goes down and goes across.

Reflection

If is set of pixels, then its *reflection*, denoted , is obtained by reflecting in the origin:

For examples, in figure 2, the open and closed circles form sets which are reflections of each other.

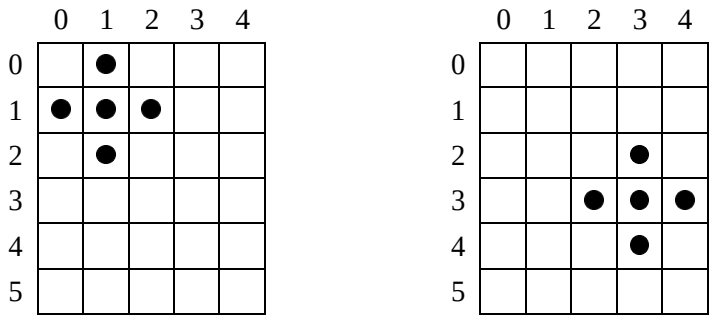


Figure 1: Translation

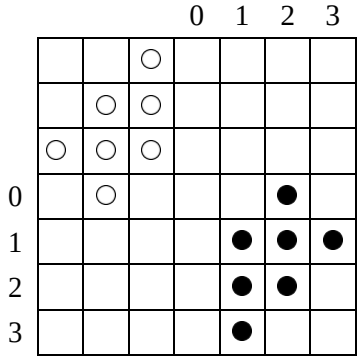


Figure 2: Reflection

Dilation and erosion

These are the basic operations of morphology, in the sense that all other operations are built from a combination of these two.

Dilation

Suppose and are sets of pixels. Then the *dilation of by* , denoted , is defined as

What this means is that for every point , we translate by those coordinates. Then we take the union of all these translations.

An equivalent definition is that

From this last definition, dilation is shown to be commutative; that

An example of a dilation is given in figure 3. In the translation diagrams, the grey squares show the original position of the object. Note that is of course just itself. In this example, we have

and those these are the coordinates by which we translate .

In general, can be obtained by replacing every point in with a copy of , placing the point of at . Equivalently, we can replace every point of with a copy of .

Dilation is also known as *Minkowski addition*; see Haralick and Shapiro [5] for more information.

As you see in figure 9.3, dilation has the effect of increasing the size of an object. However, it is not necessarily true that the original object will lie within its dilation . Depending on the coordinates of , may end up quite a long way from . Figure 4 gives an example of this:is the same as in figure 3; has the same shape but a different position. In this figure, we have

so that

For dilation, we generally assume that is the image being processed, and is a small set of pixels. In this case is referred to as a *structuring element* or as a *kernel*.

Dilation in MATLAB is performed with the command

```
>> imdilate(image,kernel)
```

To see an example of dilation, consider the commands:

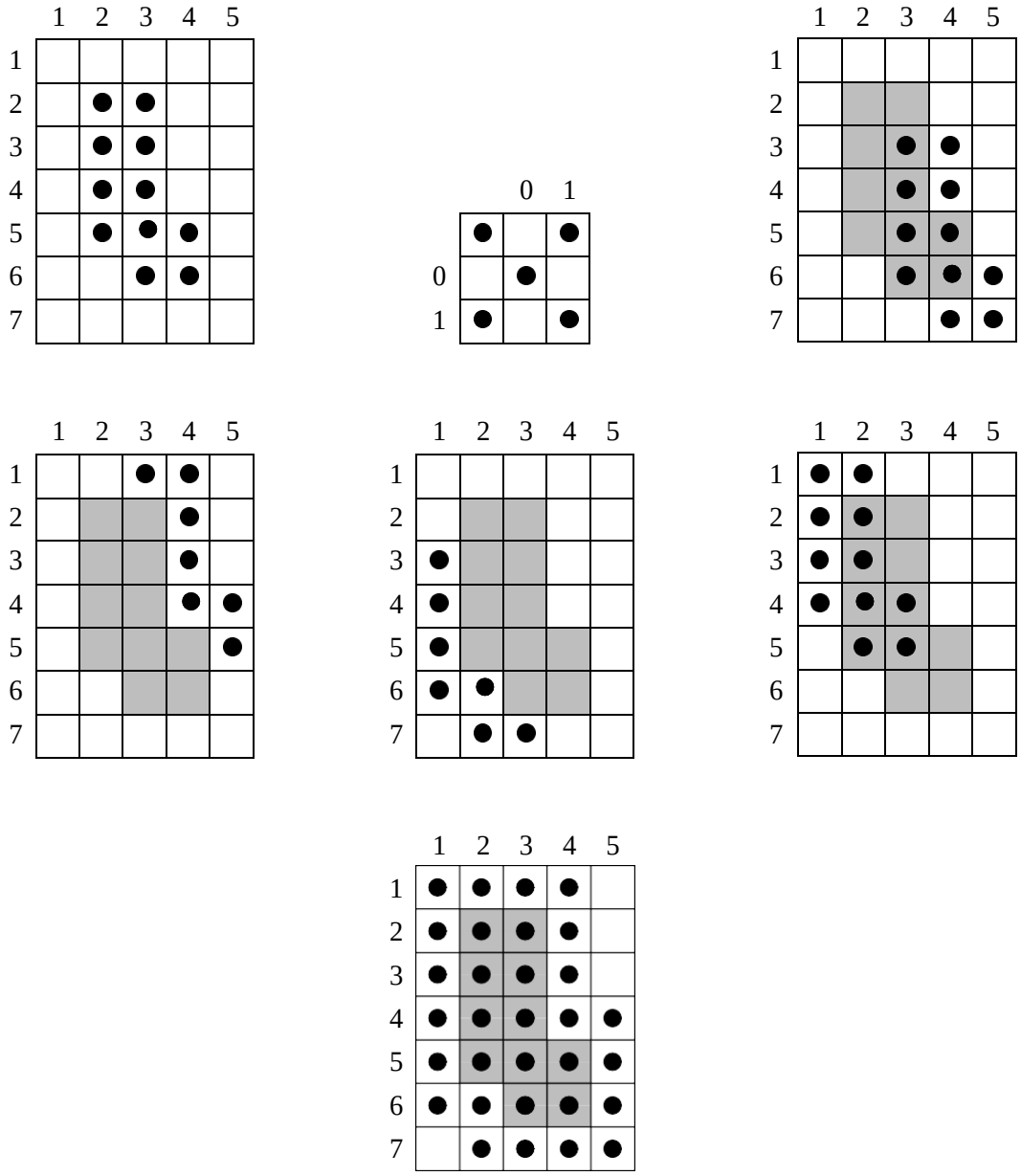


Figure 3: Dilation

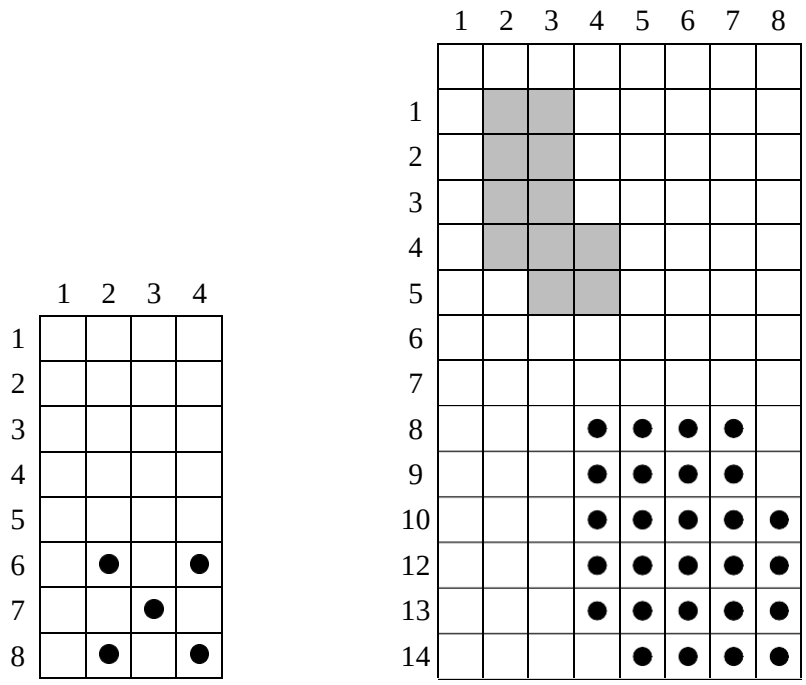


Figure 4: A dilation for which

```
>> t=imread(' text.tif' );
>> sq=ones(3,3);
>> td=imdilate(t,sq);
>> subplot(1,2,1),imshow(t)
>> subplot(1,2,2),imshow(td)
```

The result is shown in figure 5. Notice how the image has been “thickened”. This is really what dilation does; hence its name.

Erosion

Given sets and , the *erosion of by* , written , is defined as:

In other words the erosion of by consists of all points for which is in . To perform an erosion, we can move over , and find all the places it will fit, and for each such place mark down the corresponding point of . The set of all such points will form the erosion.

An example of erosion is given in figures 6.

Note that in the example, the erosion was a subset of . This is not necessarily the case; it depends on the position of the origin in . If contains the origin (as it did in figure 9.6), then the erosion will be a subset of the original object.

Figure 7 shows an example where does not contain the origin. In this figure, the open circles in the right hand figure form the erosion.

Figure 5: Dilation of a binary image

Note that in figure 7, the *shape* of the erosion is the same as that in figure 6; however its *position* is different. Since the origin of in figure 7 is translated by from its position in figure 6, we can assume that the erosion will be translated by the same amount. And if we compare figures 6 and 7, we can see that the second erosion has indeed been shifted by from the first.

For erosion, as for dilation, we generally assume that is the image being processed, and is a small set of pixels: the structuring element or kernel.

Erosion is related to *Minkowski subtraction*: the Minkowski subtraction of from is defined as

Erosion in MATLAB is performed with the command

```
>> imerode(image,kernel)
```

We shall give an example; using a different binary image:

```
>> c=imread(' circbw.tif' );
>> ce=imerode(c,sq);
>> subplot(1,2,1),imshow(c)
>> subplot(1,2,2),imshow(ce)
```

The result is shown in figure 8. Notice how the image has been "thinned". This is the expected result of an erosion; hence its name. If we kept on eroding the image, we would end up with a completely black result.

Relationship between erosion and dilation

It can be shown that erosion and dilation are "inverses" of each other; more precisely, the complement of an erosion is equal to the dilation of the complement. Thus:

——— —

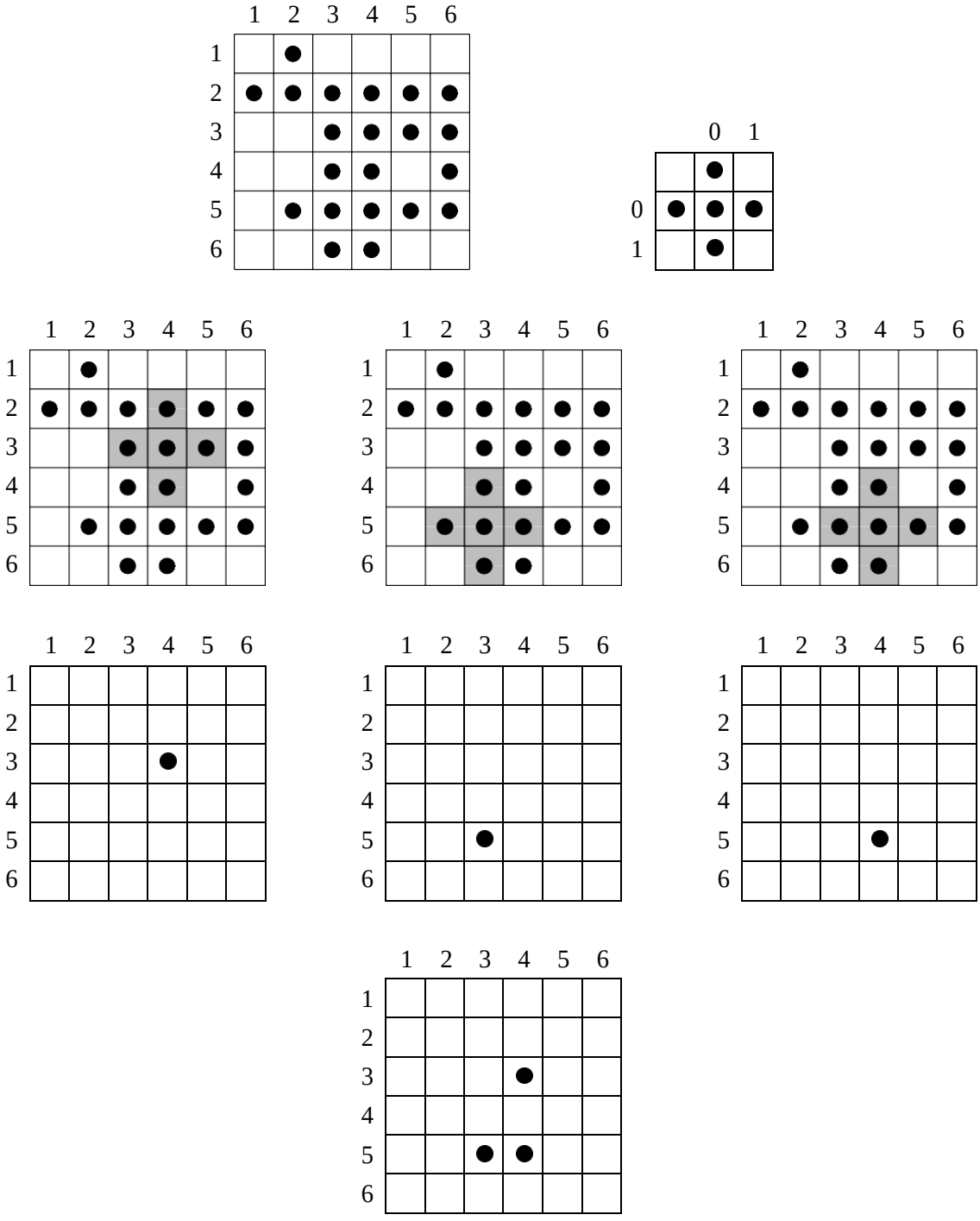


Figure 6: Erosion with a cross-shaped structuring element

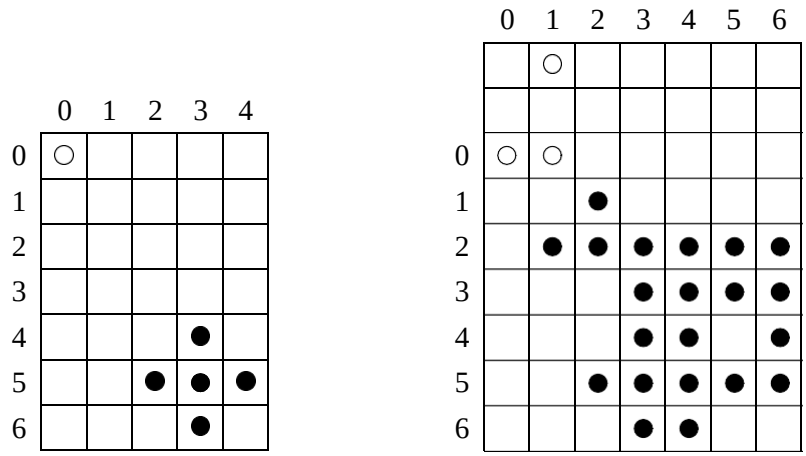


Figure 7: Erosion with a structuring element not containing the origin

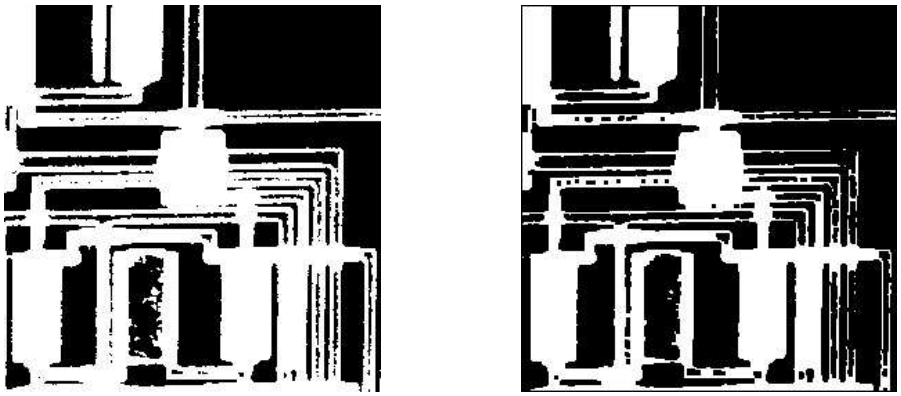


Figure 8: Erosion of a binary image

A proof of this can be found in Haralick and Shapiro [5].

It can be similarly shown that the same relationship holds if erosion and dilation are interchanged; that

——— —

We can demonstrate the truth of these using MATLAB commands; all we need to know is that the complement of a binary image

b

is obtained using

```
>> ~b
```

and that given two images a and b; their equality is determined with

```
>> all(a(:)==b(:))
```

To demonstrate the equality

——— —

pick a binary image, say the text image, and a structuring element. Then the left hand side of this equation is produced with

```
>> lhs=~imerode(t,sq);
```

and the right hand side with

```
>> rhs=imdilate(~t,sq);
```

Finally, the command

```
>> all(lhs(:)==rhs(:))
```

should return 1, for true.

An application: boundary detection

If is an image, and a small structuring element consisting of point symmetrically places about the origin, then we can define the boundary of by any of the following methods:

(i) " internal boundary"
(ii) "external boundary"
(iii) "morphological gradient"

In each definition the minus refers to set difference. For some examples, see figure 9. Note that the internal boundary consists of those pixels in which are at its edge; the external boundary consists of pixels outside which are just next to it, and that the morphological gradient is a combination of both the internal and external boundaries.

To see some examples, choose the image rice.tif, and threshold it to obtain a binary image:

```
>> rice=imread('rice.tif');
>> r=rice>110;
```

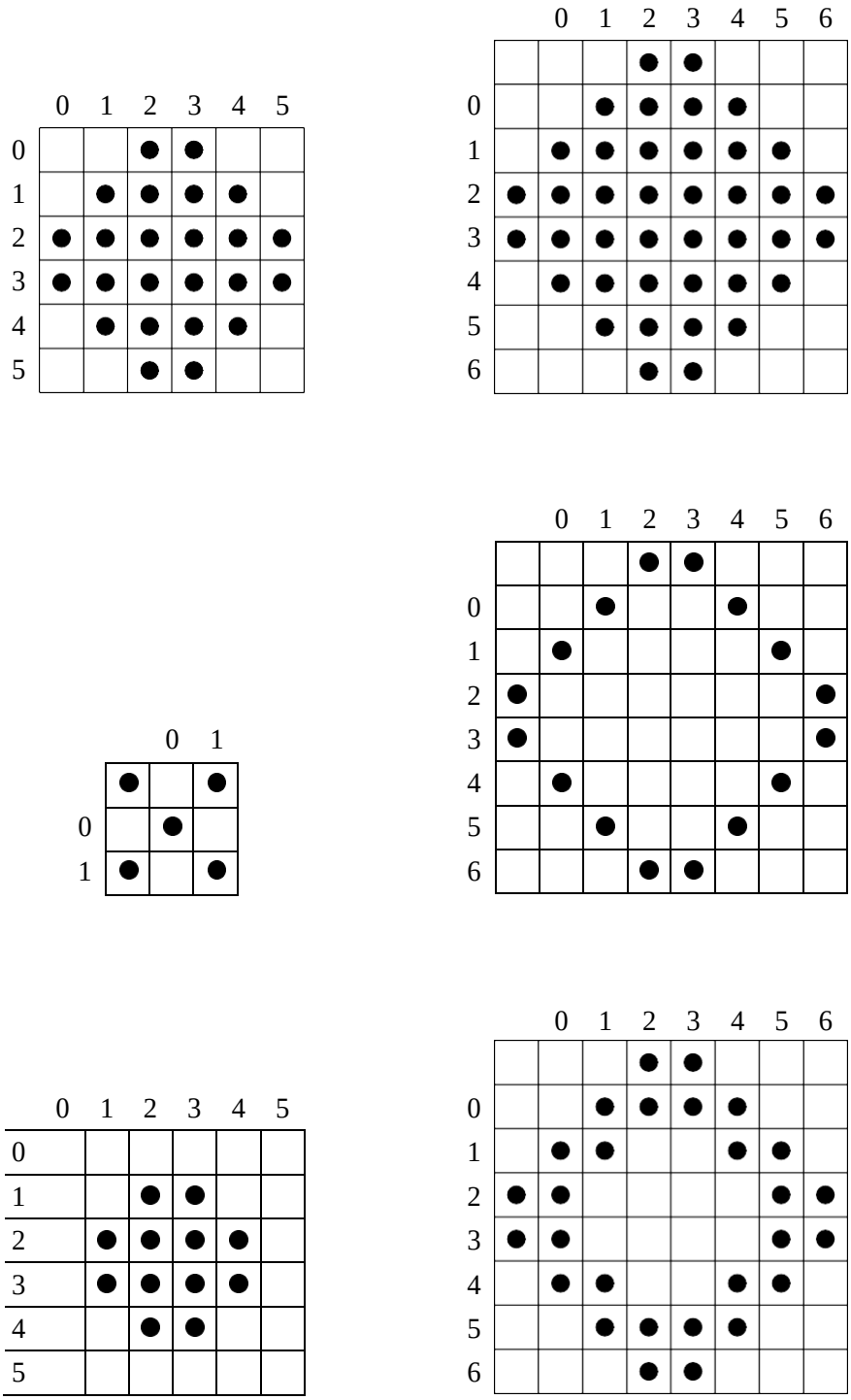


Figure 9: Boundaries

Then the internal boundary is obtained with:

```
>> re=imerode(r,sq);
>> r_int=r&~re;
>> subplot(1,2,1),imshow(r)
>> subplot(1,2,2),imshow(r_int)
```

The result is shown in figure 10.

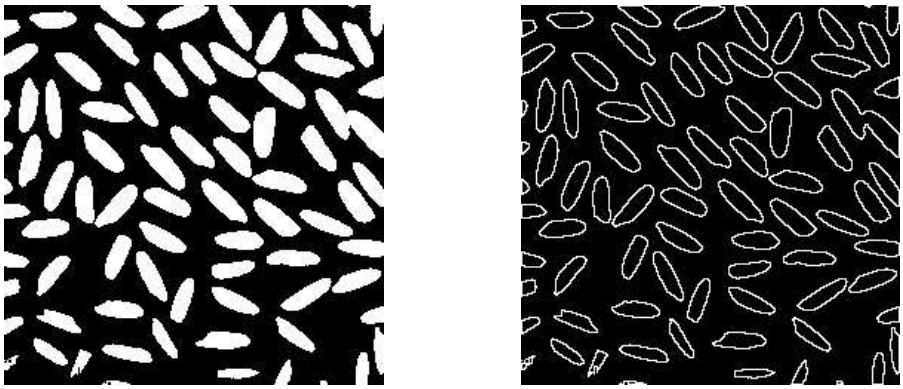


Figure 10: “Internal boundary” of a binary image

The external boundary and morphological gradients can be obtained similarly:

```
>> rd=imdilate(r,sq);
>> r_ext=rd&~r;
>> r_grad=rd&~re;
>> subplot(1,2,1),imshow(r_ext)
>> subplot(1,2,2),imshow(r_grad)
```

The results are shown in figure 11.

Note that the external boundaries are larger than the internal boundaries. This is because the internal boundaries show the outer edge of the image components; whereas the external boundaries show the pixels just outside the components. The morphological gradient is thicker than either, and is in fact the union of both.

Exercises

1. For each of the following images and structuring elements :

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

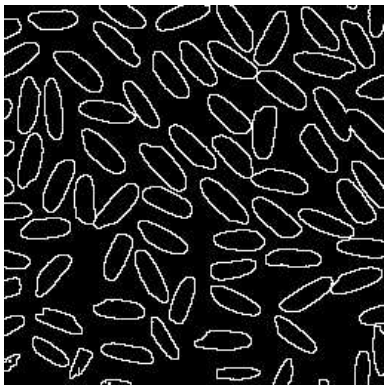


Figure 11: “External boundary” and the morphological gradient of a binary image

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

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

calculate the erosion , the dilation , the opening and the closing .

Check your answers with MATLAB.

2. Suppose a square object was eroded by a circle whose radius was about one quarter the side of the square. Draw the result.

3. Repeat the previous question with dilation.

4. Using the binary images `circbw.tif`, `circles.tif`, `circlesm.tif`, `logo.tif` and `testpat2.tif`, view the erosion and dilation with both the square and the cross structuring elements.

   Can you see any differences?

Chapter 10

Mathematical morphology (2)

Opening and closing

These operations may be considered as "second level" operations; in that they build on the basic operations of dilation and erosion. They are also, as we shall see, better behaved mathematically.

Opening

Given and a structuring element , the *opening of by* , denoted , is defined as:

So an opening consists of an erosion followed by a dilation. An equivalent definition is

That is, is the union of all translations of which fit inside . Note the difference with erosion: the erosion consists only of the point of for those translations which fit inside ; the opening consists of all of . An example of opening is given in figure 1.

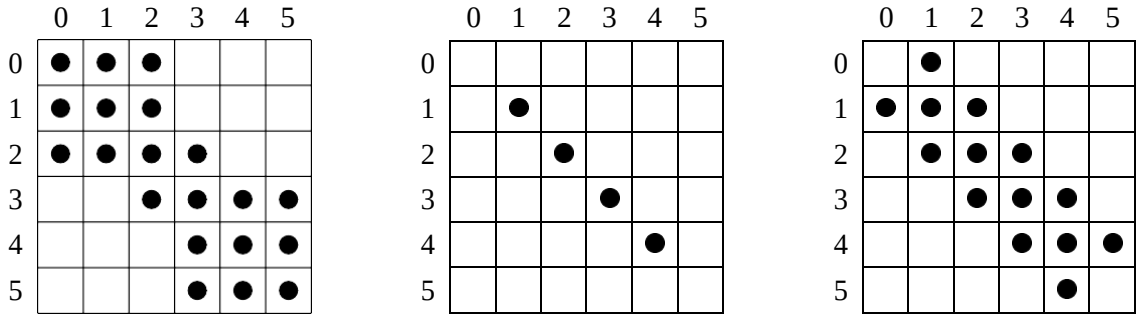


Figure 1: Opening

The opening operation satisfies the following properties:

1. . Note that this is not the case with erosion; as we have seen, an erosion may not necessarily be a subset.

2. . That is, an opening can never be done more than once. This property is called *idempotence*. Again, this is not the case with erosion; you can keep on applying a sequence of erosions to an image until nothing is left.

3. If , then .

4. Opening tends to "smooth" an image, to break narrow joins, and to remove thin protrusions.

Closing

Analogous to opening we can define *closing*, which may be considered as a dilation followed by an erosion, and is denoted :

Another definition of closing is that if *all* translations which contain have non-empty intersections with . An example of closing is given in figure 2. The closing operation satisfies

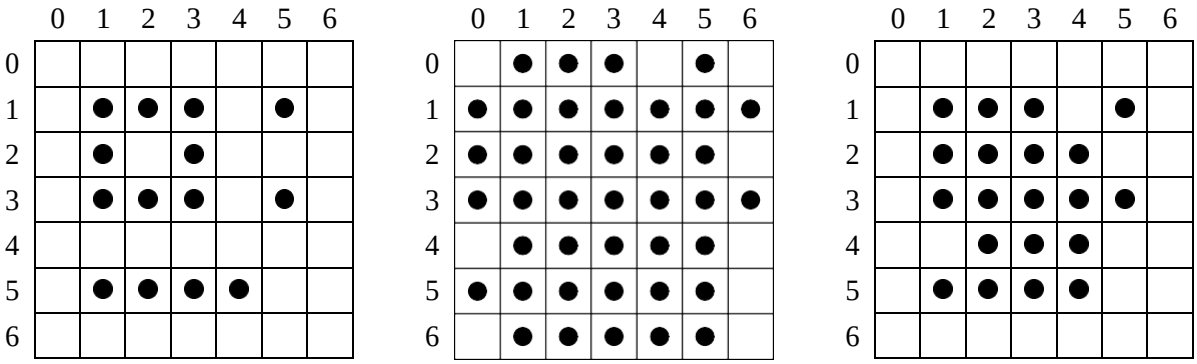


Figure 2: Closing

the following properties:

1. .

2. ; that is, closing, like opening, is idempotent.

3. If , then .

4. Closing tends also to smooth an image, but it fuses narrow breaks and thin gulfs, and eliminates small holes.

Opening and closing are implemented by the `imopen` and `imclose` functions respectively. We can see the effects on a simple image using the square and cross structuring elements.

```
>> cr=[0 1 0;1 1 1;0 1 0];
>> >> test=zeros(10,10);test(2:6,2:4)=1;test(3:5,6:9)=1;test(8:9,4:8)=1;test(4,5)=1

test =
```

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

>> imopen(test,sq)

ans =

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

>> imopen(test,cr)

ans =

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

Note that in each case the image has been separated into distinct components, and the lower part has been removed completely.

```
>> imclose(test,sq)

ans =
```

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

>> imclose(test,cr)

ans =

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

With closing, the image is now fully "joined up". We can obtain a joining-up effect with the text image, using a diagonal structuring element.

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

diag =

     0     0     1
     0     1     0
     1     0     0

>> tc=imclose(t,diag);
>> imshow(tc)
```

The result is shown in figure 3.

An application: noise removal

Suppose is a binary image corrupted by impulse noise—some of the black pixels are white, and some of the white pixels are back. An example is given in figure 4. Then will remove the single black pixels, but will enlarge the holes. We can fill the holes by dilating twice:

Figure 3: An example of closing

The first dilation returns the holes to their original size; the second dilation removes them. But this will enlarge the objects in the image. To reduce them to their correct size, perform a final erosion:

The inner two operations constitute an opening; the outer two operations a closing. Thus this noise removal method is in fact an opening followed by a closing:

This is called *morphological filtering*.

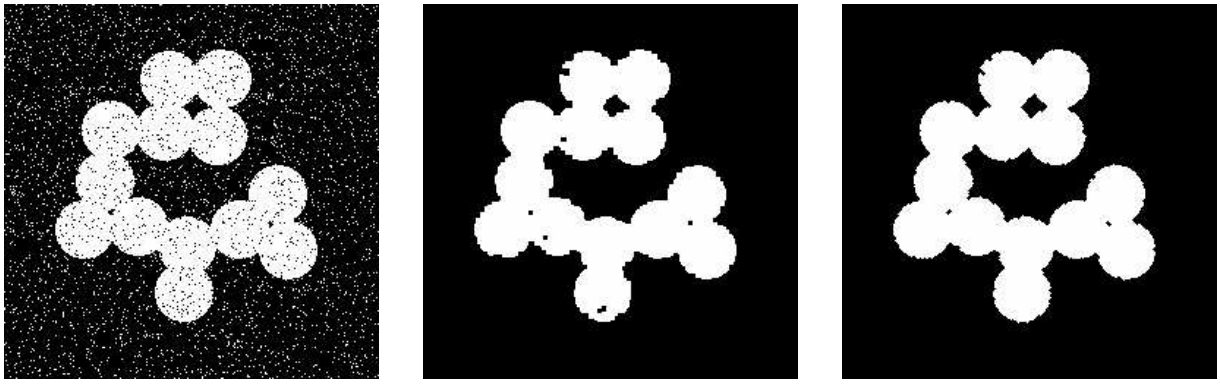
Suppose we take an image and apply shot noise to it:

```
>> c=imread('circles.tif');
>> x=rand(size(c));
>> d1=find(x<=0.05);
>> d2=find(x>=0.95);
>> c(d1)=0;
>> c(d2)=1;
>> imshow(c)
```

The result is shown as figure 4(a). The filtering process can be implemented with

```
>> cf1=imclose(imopen(c,sq),sq);
>> figure,imshow(cf1)
>> cf2=imclose(imopen(c,cr),cr);
>> figure,imshow(cf2)
```

and the results are shown as figures 4(b) and (c). The results are rather “blocky”; although lessso with the cross structuring element.



(a) (b) (c)

Figure 4: A noisy binary image and results after morphological filtering with different structuring elements.

Relationship between opening and closing

Opening and closing share a relationship very similar to that of erosion and dilation: the complement of an opening is equal to the closing of a complement, and complement of an closing is equal to the opening of a complement. Specifically:

$$\overline{A \circ B} = \overline{A} \bullet B$$

and

$$\overline{A \bullet B} = \overline{A} \circ B$$

Again see Haralick and Shapiro [5] for a formal proof.

The hit-or-miss transform

This is a powerful method for finding shapes in images. As with all other morphological algorithms, it can be defined entirely in terms of dilation and erosion; in this case, erosion only.

Suppose we wish to locate square shapes, such as is in the centre of the image in figure 5.

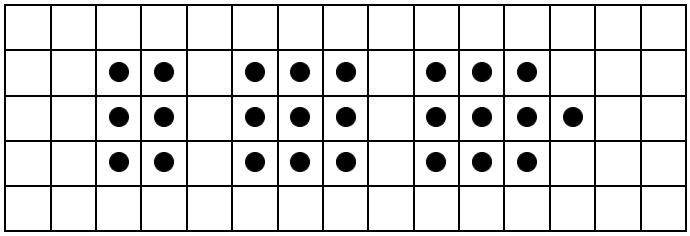


Figure 5: An image containing a shape to be found

If we performed an erosion with being the square structuring element, we would obtain the result given in figure 6.

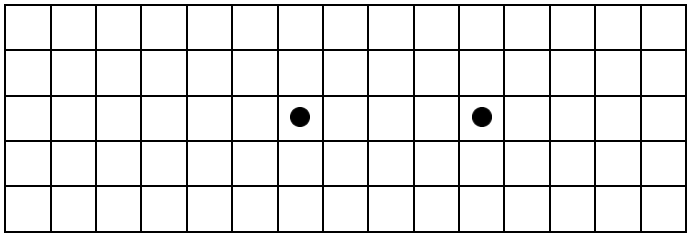


Figure 6: The erosion

The result contains two pixels, as there are exactly two places in where will fit. Now suppose we also erode the complement of with a structuring element which fits exactly around the square; ¯ and are shown in figure 7. (We assume that is at the centre of .)

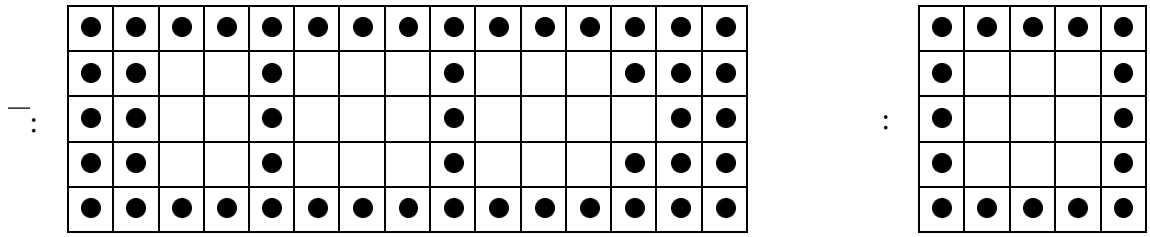


Figure 7: The complement and the second structuring element

If we now perform the erosion ¯ we would obtain the result shown in figure 8.

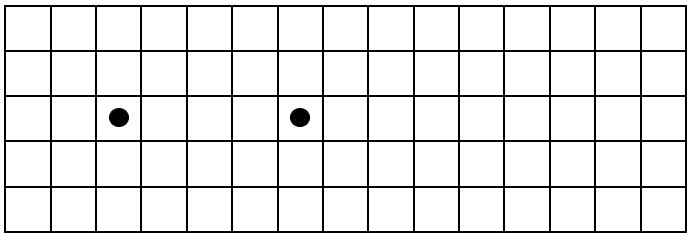


Figure 8: The erosion ¯

The intersection of the two erosion operations would produce just one pixel at the position of the centre of the square in , which is just what we want. If had contained more than one square, the final result would have been single pixels at the positions of the centres of each. This combination of erosions forms the hit-or-miss transform.

In general, if we are looking for a particular shape in an image, we design two structuring elements: which is the same shape, and which fits around the shape. We then write

and

¯

for the hit-or-miss transform.

As an example, we shall attempt to find the hyphen in “Cross-Correlation” in the text image shown in figure 5 in chapter 9. This is in fact a line of pixels of length six. We thus can create our two structuring elements as:

```
>> b1=ones(1,6);
>> b2=[1 1 1 1 1 1 1;1 0 0 0 0 0 0; 1 1 1 1 1 1 1];
>> tb1=erode(t,b1);
>> tb2=erode(~t,b2);
>> hit_or_miss=tb1&tb2;
>> [x,y]=find(hit_or_miss==1)
```

and this returns a coordinate of , which is right in the middle of the hyphen. Note that the command

```
>> tb1=erode(t,b1);
```

is not sufficient, as there are quite a few lines of length six in this image. We can see this by viewing the image `tb1`, which is given in figure 9.

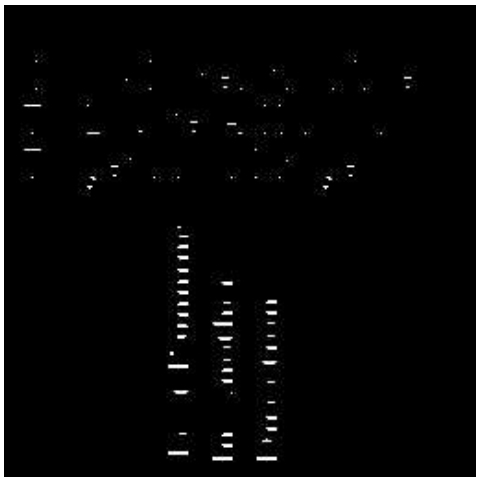


Figure 9: Text eroded by a hyphen-shaped structuring element

Some morphological algorithms

In this section we shall investigate some simple algorithms which use some of the morphological techniques we have discussed in previous sections.

Region filling

Suppose in an image we have a region bounded by an -connected boundary, as shown in figure 10.10. Given a pixel within the region, we wish to fill up the entire region. To do this, we start with , and dilate as many times as necessary with the cross-shaped structuring element (as used in

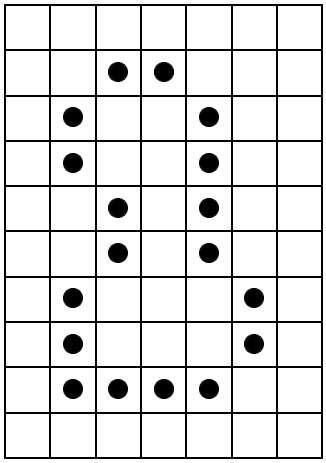


Figure 10: An -connected boundary of a region to be filled

figure 6) in ch9, each time taking an intersection with before continuing. We thus create a sequenceof sets:

for which

Finally is the filled region. Figure 11 shows how this is done.

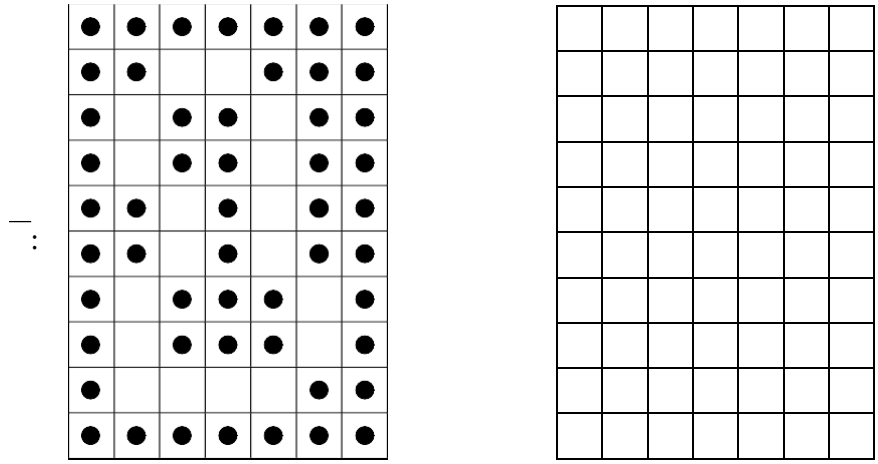


Figure 11: The process of filling a region

In the right hand grid, we have

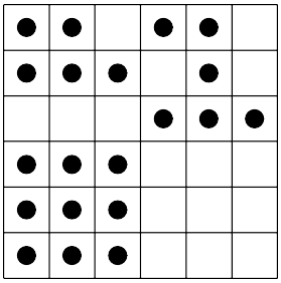
Note that the use of the cross-shaped structuring element means that we never cross the boundary.

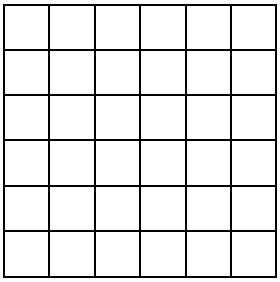
Connected components

We use a very similar algorithm to fill a connected component; we use the cross-shaped structuring element for -connected components, and the square structuring element for -connected components. Starting with a pixel , we fill up the rest of the component by creating a sequence of sets

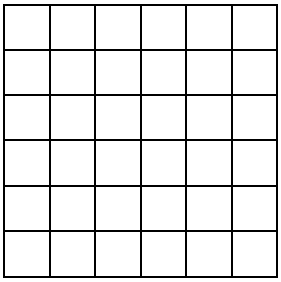
such that

until . Figure 12 shows an example.





Using the cross



Using the square

Figure 12: Filling connected components

In each case we are starting in the centre of the square in the lower left. As this square is itself a -connected component, the cross structuring element cannot go beyond it.

Both of these algorithms can be very easily implemented by MATLAB functions. To implement region filling, we keep track of two images: `current` and `previous`, and stop when there is no difference between them. We start with `previous` being the single point in the region, and `current` the dilation —. At the next step we set

—

Given , we can implement the last step in MATLAB by

```
imdilate(current,B)&~A.
```

The function is shown in figure 13 We can use this to fill a particular region delineated by a boundary.

```
>> n=imread(' nicework.tif' );
>> imshow(n),pixval on
>> nb=n&~imerode(n,sq);
>> figure,imshow(nb)
>> nf=regfill(nb,[74,52],sq);
>> figure,imshow(nf)
```

```
function out=regfill(im,pos,kernel)
% REGFILL(IM,POS,KERNEL) performs region filling of binary image IMAGE,
% with kernel KERNEL, starting at point with coordinates given by POS.
%
% Example:
%           n=imread(' nicework.tif' );
%           nb=n&~imerode(n,ones(3,3));
%           nr=regfill(nb,[74,52],ones(3,3));
%
current=zeros(size(im));
last=zeros(size(im));
last(pos(1),pos(2))=1;
current=imdilate(last,kernel)&~im;
while any(current(:)~=last(:)),
  last=current;
  current=imdilate(last,kernel)&~im;
end;
out=current;
```

Figure 13: A simple program for filling regions

The results are shown in figure 14. Image (a) is the original; (b) the boundary, and (c) the result of a region fill. Figure (d) shows a variation on the region filling, we just include all boundaries. This was obtained with

```
>> figure,imshow(nf|nb)
```

(a) (b) (c) (d)

Figure 14: Region filling

The function for connected components is almost exactly the same as that for region filling, except that whereas for region filling we took an intersection with the *complement* of our image, for connected components we take the intersection with the image itself. Thus we need only change one line, and the resulting function is shown in 15 We can experiment with this function with the "nice work" image. We shall use the square structuring element, and also a larger structuring element of size .

```
>> sq2=ones(11,11);
>> nc=components(n,[57,97],sq);
```

```
function out=components(im,pos,kernel)
% COMPONENTS(IM,POS,KERNEL) produces the connected component of binary image
% IMAGE which nicludes the point with coordinates given by POS, using
% kernel KERNEL.
%
% Example:
%           n=imread(' nicework.tif' );
%           nc=components(nb,[74,52],ones(3,3));
%
current=zeros(size(im));
last=zeros(size(im));
last(pos(1),pos(2))=1;
current=imdilate(last,kernel)&im;
while any(current(:)~=last(:)),
  last=current;
  current=imdilate(last,kernel)&im;
end;
out=current;
```

Figure 15: A simple program for connected components

```
>> imshow(nc)
>> nc2=components(n,[57,97],sq2);
>> figure,imshow(nc2)
```

and the results are shown in figure 16. Image (a) uses the square; image (b) uses the square.

(a) (b)

Figure 16: Connected components

Skeletonization

Recall that the *skeleton* of an object can be defined by the “medial axis transform”; we may imagine fires burning in along all edges of the object. The places where the lines of fire meet form the skeleton. The skeleton may be produced by morphological methods.

Consider the table of operations as shown in table 1.

Erosions	Openings	Set differences
.	.	.

Table 1: Operations used to construct the skeleton

Here we use the convention that a sequence of erosions using the same structuring element is denoted . We continue the table until is empty. The skeleton is then obtained by taking the unions of all the set differences. An example is given in figure 17, using the cross structuring element.

Since is empty, we stop here. The skeleton is the union of all the sets in the third column; it is shown in figure 18. This method of skeletonization is called *Lantuéjoul's method*;for details see Serra [12].

This algorithm again can be implemented very easily; a function to do so is shown in figure 19. We shall experiment with the nice work image.

```
>> nk=imskel(n,sq);
>> imshow(nk)
>> nk2=imskel(n,cr);
>> figure,imshow(nk2)
```

The result is shown in figure 20. Image (a) is the result using the square structuring element;Image (b) is the result using the cross structuring element.

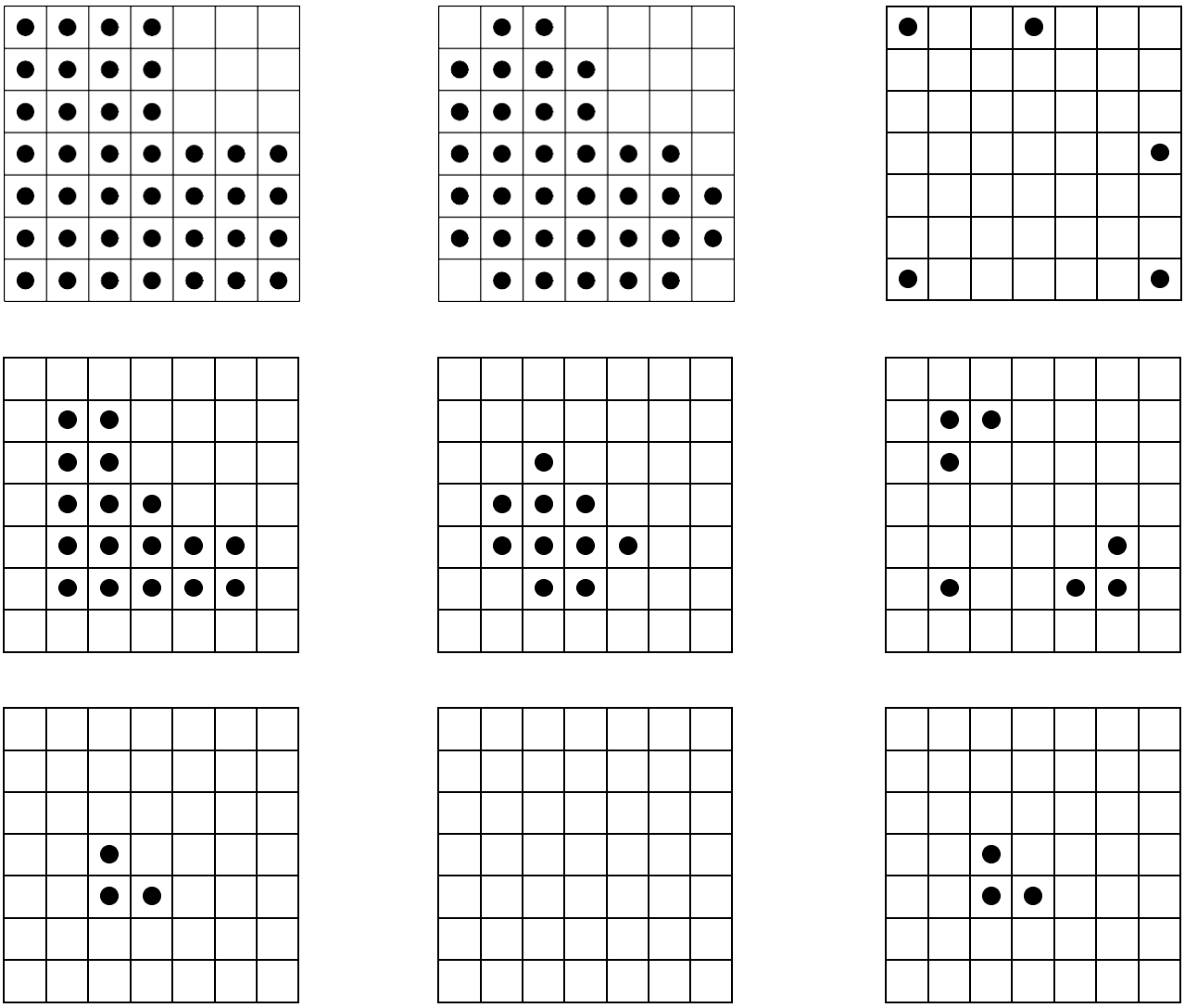


Figure 17: Skeletonization

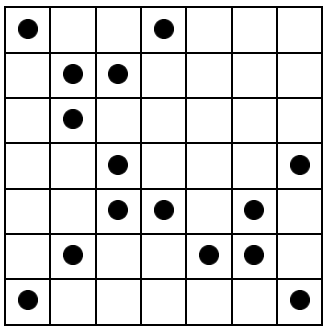


Figure 18: The final skeleton

```
function skel = imskel(image,str)
% IMSKEL(IMAGE,STR) - Calculates the skeleton of binary image IMAGE using
% structuring element STR.  This function uses Lantejoul's algorithm.
%
skel=zeros(size(image));
e=image;
while (any(e(:))),
    o=imopen(e,str);
    skel=skel | (e&~o);
    e=imerode(e,str);
end
```

Figure 19: A simple program for computing skeletons

(a) (b)

Figure 20: Skeletonization of a binary image

Image Processing Report

الاسم :عمر وليد محمد الدمرداش
سكشن:2