Lecture 3: Basic Morphological Image Processing

Harvey Rhody
Chester F. Carlson Center for Imaging Science
Rochester Institute of Technology
rhody@cis.rit.edu

September 13, 2005

Abstract

Morphological processing is constructed with operations on sets of pixels. Binary morphology uses only set membership and is indifferent to the value, such as gray level or color, of a pixel. We will examine some basic set operations and their usefulness in image processing.

Morphology and Sets

We will deal here only with morphological operations for binary images. This will provide a basic understanding of the techniques. Morphological processing for gray scale images requires more sophisticated mathematical development.

Morphological processing is described almost entirely as operations on sets. In this discussion, a set is a collection of pixels in the context of an image.

Our sets will be collections of points on an image grid G of size $N \times M$ pixels.

Pixel Location

A set A of pixels in G can be described by giving a list of the pixel locations. The locations can be given as a list of index positions from G that are included in A.

If G is of size $N \times M$ then the index positions are the integers in the range $[0, MN-1]$. For $[N, M] = [7, 3]$ the index grid is

$$G = \begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ \hline 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ \hline \end{array}$$

A pixel in column x and row y has index $p = x + Ny$. Given p one can find the column and row coordinates by

$$\begin{aligned} x &= p \mod N \\ y &= p/N \end{aligned}$$

Set Operations

Let A and B be sets. If a is the index of a pixel in A, then we write $a \in A$. If a is not in A we write $a \notin A$.

If every element that is in A is also in B then A is a *subset* of B, written

$$A \subseteq B$$

This is equivalent to the statement $a \in A \Rightarrow a \in B$.

The *union* of A and B is the collection of all elements that are in one or both set. The union is the set represented by $C = A \cup B$. We can write

$$C = \{p | p \in A \text{ or } p \in B \text{ (or both)}\}$$

Set Operations

Set A is the white object in the black box on the right. The image grid G is the $N \times M$ black rectangle that holds the set.

In `IDL` programs we write a set A as an array whose background values are zero and object values are not zero.

We can find the coordinates of the object by

$$
\begin{aligned}
p &= \text{WHERE}(A) \\
x &= p \mod N \\
y &= p/N
\end{aligned}
$$

Set Operations - Union

The operation

$$C = A \cup B$$

produces a set that contains the elements of both A and B. The fact that some elements are represented in both sets (red pixels) does not affect the result.

Note that B itself contains two disjoint subsets.

Set A

Set B

Overlap (red)

$A \cup B$

Set Operations (cont)

The *intersection* of two sets A and B is

$$D = A \cap B = \{p | p \in A \text{ and } p \in B\}$$

It is possible that A and B have no common elements. In that case, we say that they are *disjoint* and write

$$A \cap B = \emptyset$$

where $\emptyset$ is the name for the set with no members.

The *complement* of a set A is the set of elements in the image grid G that are not in A:

$$A^c = \{w | w \in G \text{ and } w \notin A\}$$

Set Operations - Intersection

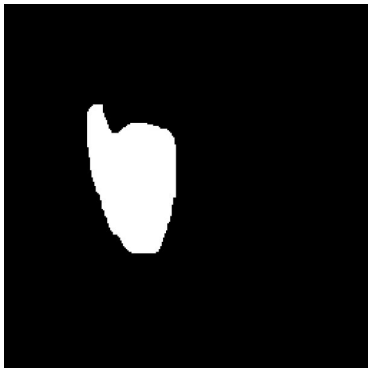
The operation $C = A \cap B$ produces a set that contains the elements that are in both A and B.

If A and B are binary image arrays, then the intersection is computed in IDL by

$$D = A \text{ AND } B$$

The pixels in the intersection can be found by

$$p = \text{WHERE(A AND B)}$$



Set A

Set B

Overlap (red)

$A \cap B$

Set Operations - Complement

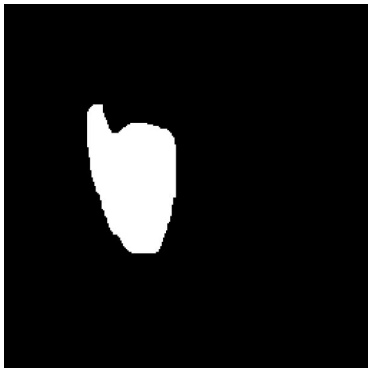
The complement A^c is the set of elements that are not contained in A.

The complement is computed in `IDL` by

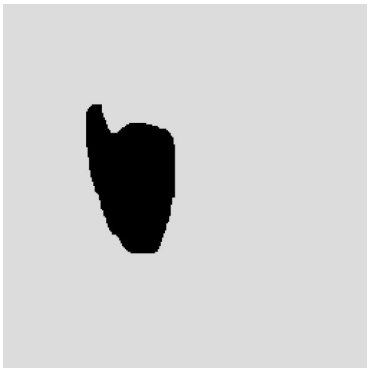
$$A^c = A \text{ EQ } 0$$

The pixels in the intersection can be found by

$$p = \text{WHERE(A EQ 0)}$$



Set A

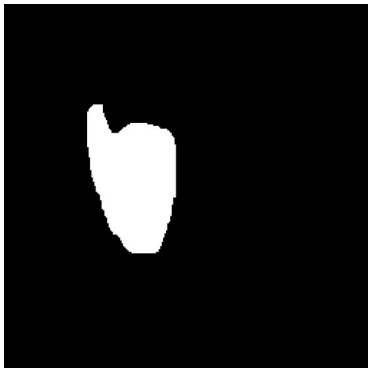


Set A^c

Set Operations - Difference

The *difference* of two sets A and B, denoted by $A - B$ is

$$\begin{aligned} A - B &= \{w | w \in A \text{ and } w \notin B\} \\ &= A \cap B^c \end{aligned}$$



Set A

Set B

The set difference does *not* involve subtraction of pixel values.

The pixels in the set can be computed by

$p =$ `WHERE(A AND (B EQ 0))`

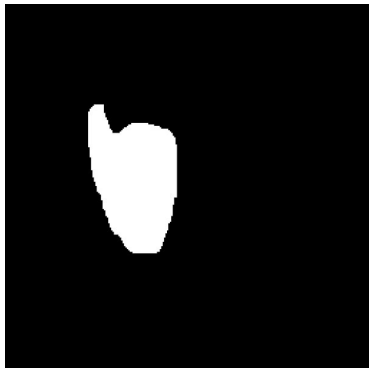
Set B^c

$A - B$

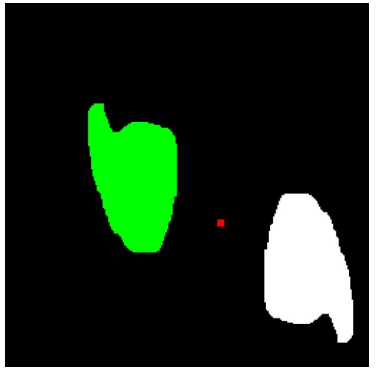
Set Operations - Reflection

A standard morphological operation is the reflection of all of the points in a set about the origin of the set. The origin of a set is not necessarily the origin of the base.

Shown at the right is an image and its reflection about a point (shown in red), with the original image in green and the reflected image in white.



Set A

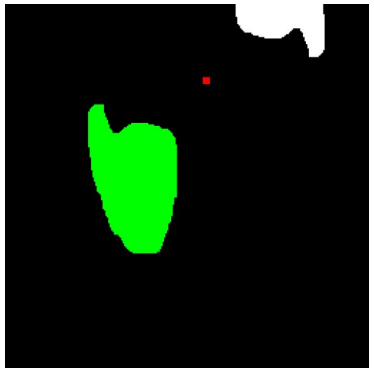


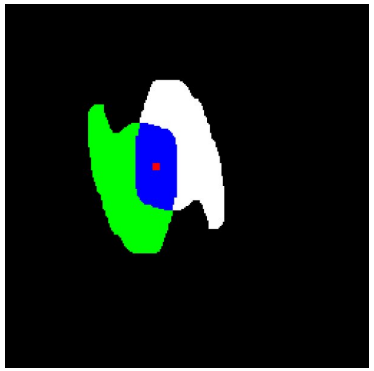
Set $\hat{A}$

Set Operations - Reflection

The reflected image (white) may fall partially or entirely outside the image grid as shown in the top figure.

The reflected image (white) may overlap the original image (green) as shown in the lower figure. The overlap is blue.





Set Operations - Reflection Program

```
FUNCTION REFLECTION,k,kc,ncols,nrows,row=row,column=column
;+
;khat=REFLECTION(k,kc,ncols,nrows) is the reflection of image set k
;about the origin represented by kc.  The number of columns and rows in the
;base image must be provided.  Keywords /row and /column may be set to
;reflect the image around the row or column that contains kc instead of the point
;kc.
;
;If the reflected set is empty then khat=-1 is returned.
;
;HISTORY
;Default reflection written by HR Sept.  1999 for SIMG-782
;Modified to reflect about row and column by JH Oct.  2000 for SIMG-782
;-
IF N_PARAMS() LT 4 THEN MESSAGE,'REFLECTION has too few arguments'
;Find the rectangular coordinats of the set points and the origin.
x=k MOD ncols
y=k/ncols
xc=kc MOD ncols
yc=kc/ncols
```

Set Operations - Reflection Program (cont)

```
;Check for keyword specifying desired reflection and then carry that reflection out.
if keyword_set(row) then begin
;Doing row reflection
xr=x
yr=2*yc-y
endif else begin
if keyword_set(column) then begin
;Doing column reflction
xr=2*xc-x
yr=y
endif else begin
;Reflect about the origin.  If x=xc + dx then xr=xc-dx=2*xc-x.
;Same for yr.
xr=2*xc-x
yr=2*yc-y
endelse
endelse
;Keep the points that fall within the base image frame.
is=WHERE(xr GE 0 AND xr LT ncols AND yr GE 0 and yr LT nrows)
if min(is) ge 0 then khat=yr[is]*ncols+xr[is] else khat=-1
RETURN,khat
END
```

Dilation and Erosion

Dilation and erosion are basic morphological processing operations. They are defined in terms of more elementary set operations, but are employed as the basic elements of many algorithms.

Both dilation and erosion are produced by the interaction of a set called a *structuring element* with a set of pixels of interest in the image.

The structuring element has both a shape and an origin.

Let A be a set of pixels and let B be a structuring element. Let $(\hat{B})_s$ be the reflection of B about its origin and followed by a shift by s. Dilation, written $A \oplus B$, is the set of all shifts that satisfy the following:

$$A \oplus B = \{s | (\hat{B})_s \cap A \neq \emptyset\}$$

Equivalently,

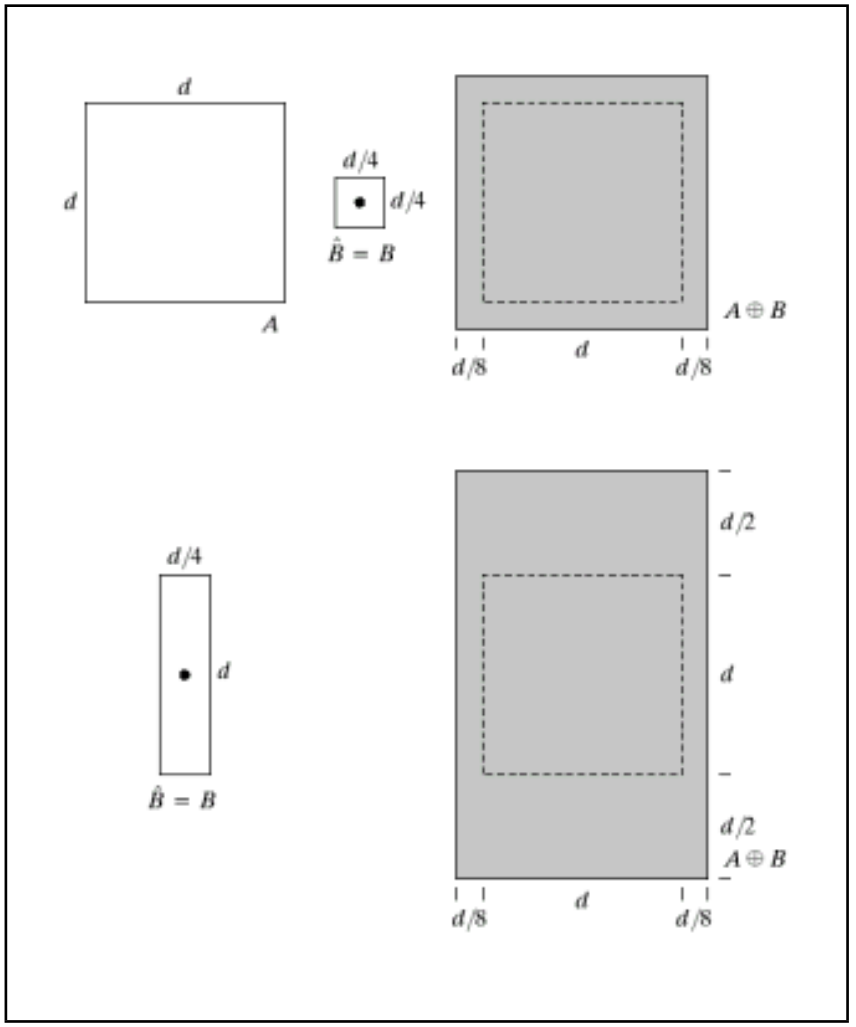
$$A \oplus B = \{s | \left((\hat{B})_s \cap A \right) \subseteq A\}$$

Morphological Dilation

Any pixel in the output image touched by the · in the structuring element is set to ON when any point of the structuring element touches a ON pixel in the original image.

This tends to close up holes in an image by expanding the ON regions. It also makes objects larger.

Note that the result depends upon both the shape of the structuring element and the location of its origin.

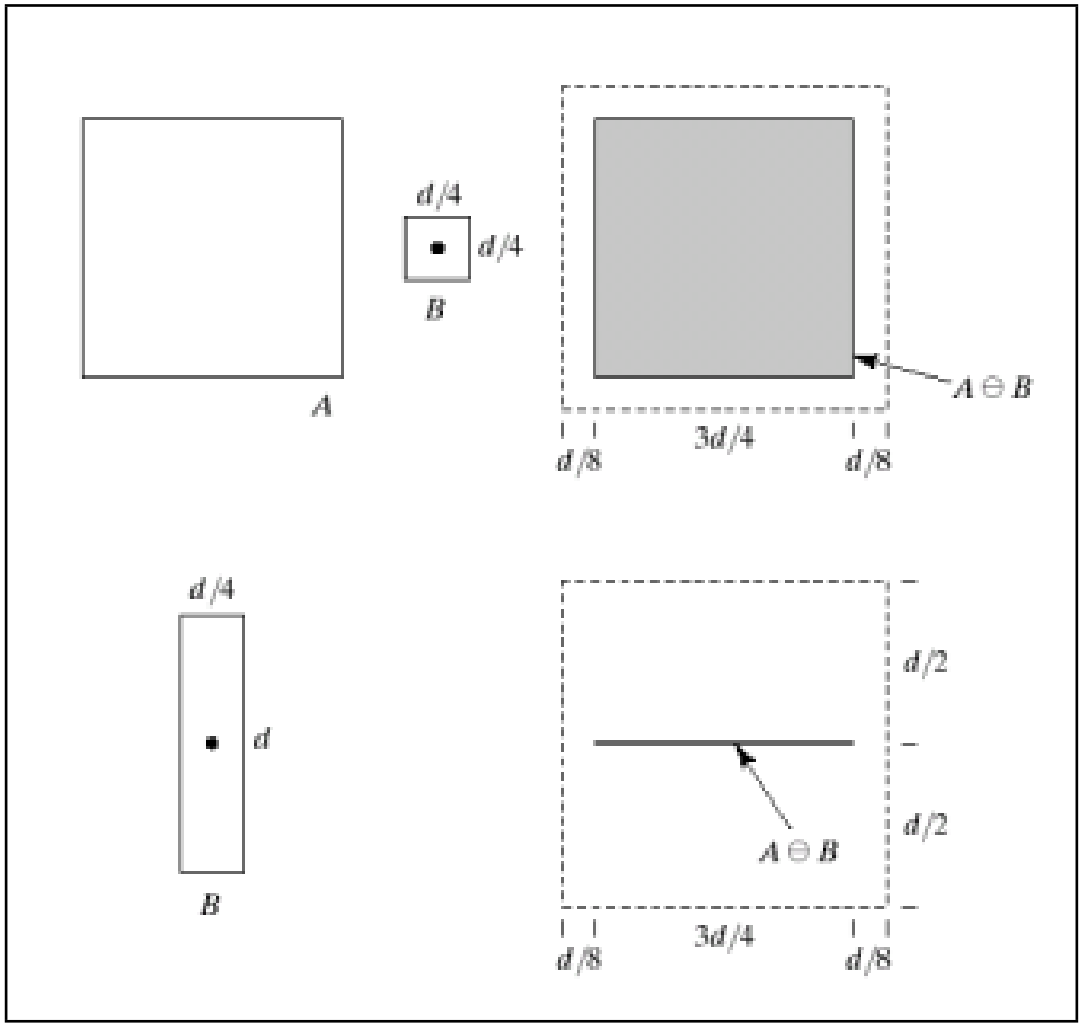


Morphological Erosion

Any pixel in the output image touched by the · in the structuring element is set to ON when every point of the structuring element touches a ON pixel in the original image.

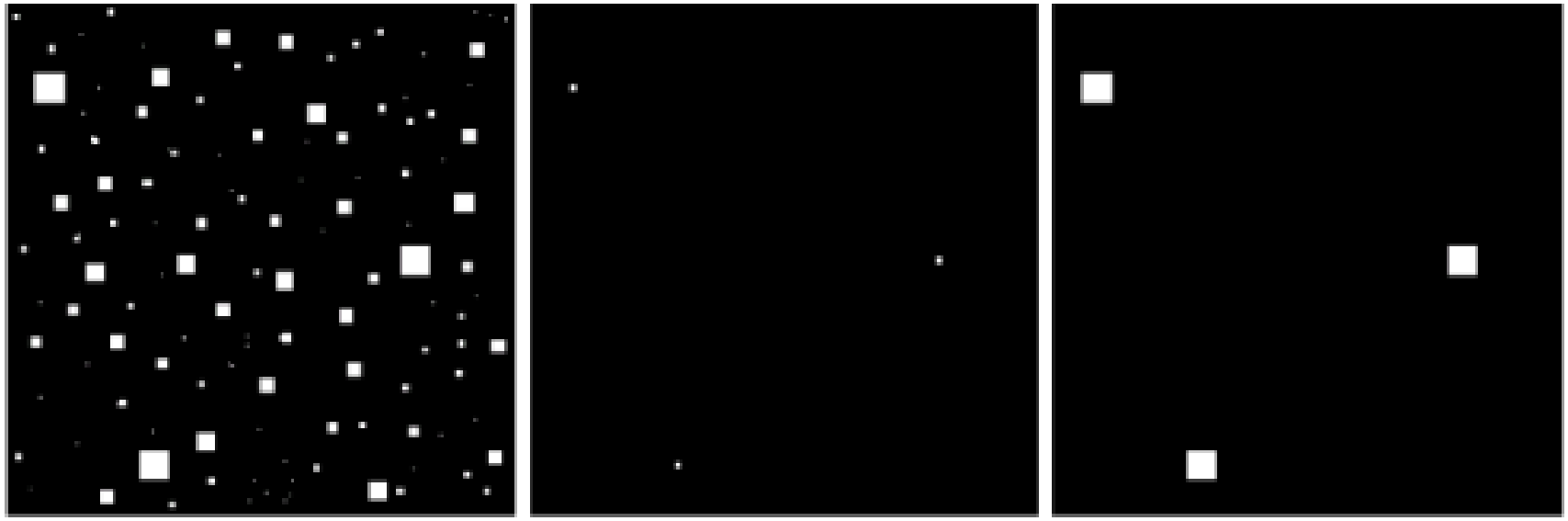
This tends to makes objects smaller by removing pixels.

$$A \ominus B = \{s|(B)_s \subseteq A\}$$



Morphological Erosion + Dilation

The effect of erosion followed by dilation is illustrated below.

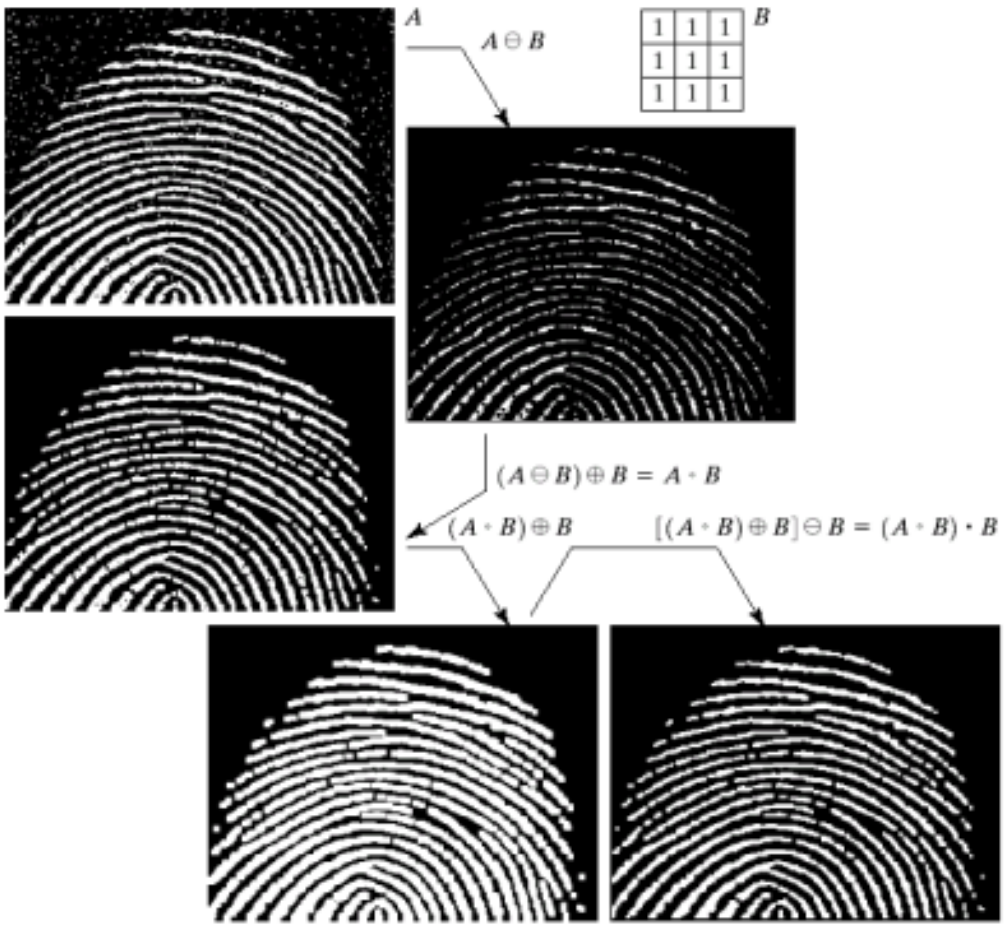


a b c

FIGURE 9.7 (a) Image of squares of size 1, 3, 5, 7, 9, and 15 pixels on the side. (b) Erosion of (a) with a square structuring element of 1's, 13 pixels on the side. (c) Dilation of (b) with the same structuring element.

Fingerprint Image Cleanup

The use of `ERODE` + `DILATE` is illustrated by this example



a b
d c
e f

FIGURE 9.11
(a) Noisy image.
(c) Eroded image.
(d) Opening of A.
(d) Dilation of the opening.
(e) Closing of the opening. (Original image for this example courtesy of the National Institute of Standards and Technology.)

Boundary Extraction

Let A be an $N \times M$ binary image array with the background represented by the value 0. The goal of boundary extraction is to find the pixels that are on the boundary of objects in the image.

The boundary pixels are the set $\beta(A)$ which can be found by

$$\beta(A) = A \bigcap (A \ominus B)^c$$

where B is a 3×3 structuring element.

Boundary extraction can be implemented with the `IDL` statement

```
B=A AND NOT ERODE(A,B)
```

Example

Let A be the array

$$A = \begin{bmatrix} 1110111110 \\ 1110111110 \\ 1111111111 \\ 1111111111 \\ 1111111111 \end{bmatrix}$$

The object takes up most of the image, with just four background pixels. Now, erode with a 3×3 structuring element. The result is

$$E = A \ominus B = \begin{bmatrix} 0000000000 \\ 0100011100 \\ 0100011100 \\ 0111111110 \\ 0000000000 \end{bmatrix}$$

Example (cont)

$$E^c = \begin{bmatrix} 1111111111 \\ 1011100011 \\ 1011100011 \\ 1000000001 \\ 1111111111 \end{bmatrix} \quad A = \begin{bmatrix} 1110111110 \\ 1110111110 \\ 1111111111 \\ 1111111111 \\ 1111111111 \end{bmatrix}$$

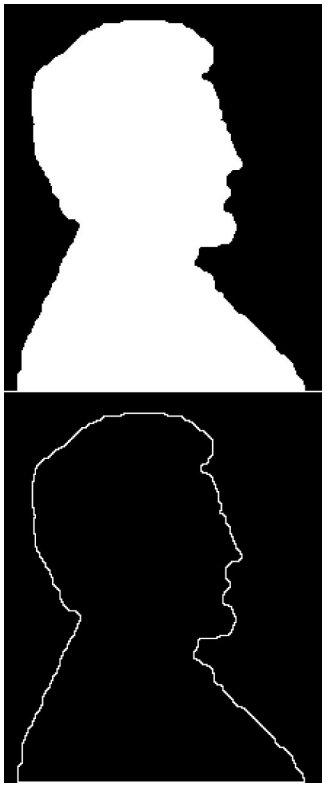
$$\beta(A) = A \bigcap E^c = \begin{bmatrix} 1110111110 \\ 1010100010 \\ 1011100011 \\ 1000000001 \\ 1111111111 \end{bmatrix}$$

The boundary $\beta(A)$ is shown as an image above. As a set, $\beta(A)$ is a list of the foreground pixels in the above array.

Example: Lincoln Image

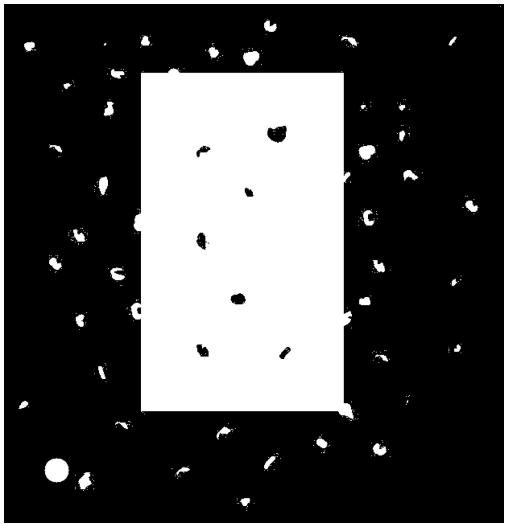
The following program reproduces the example in G&W Figure 9.14.

```
;Demonstration of Boundary Extraction
fname=datapath+'lincoln.png'
A=Read_Image(fname,rr,gg,bb)
TVLCT,rr,gg,bb
sa=Size(A,/DIM)
Window,1,Xsize=sa[0],Ysize=2*sa[1]+1
Erase,255
TV,A,0,sa[1]+1
B=Replicate(1b,3,3)
A1=A AND NOT Erode(A,B)
TV,A1 & WSHOW
```



Example: G&W Problem 9.17

Extract an object represented by the white box from a noisy image, represented by the blobs.



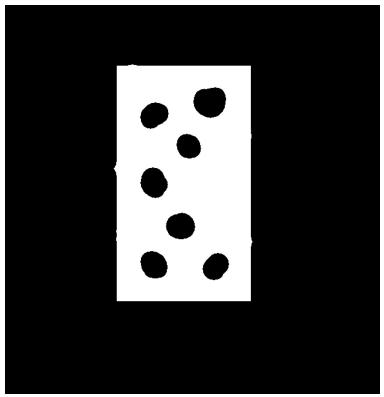
Construct a circular structuring element of radius `d=15` (shown in lower left corner of the figure)

```
d=15 & B=shift(dist(2*d+1),d,d) LE d
```

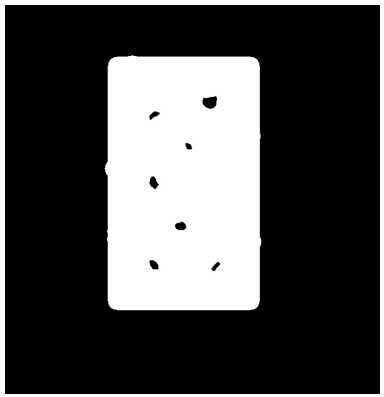
Perform a sequence of erosions and dilations using the structuring element.

```
A1=ERODE(A,B)
A2=DILATE(A1,B)
A3=DILATE(A2,B)
A4=ERODE(A3,B)
```

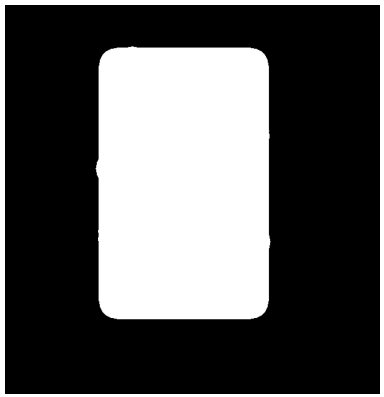
G&W Problem 9.17 (cont)



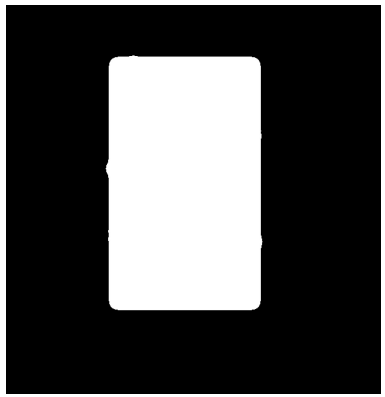
A1=ERODE(A,B)



A2=DILATE(A1,B)

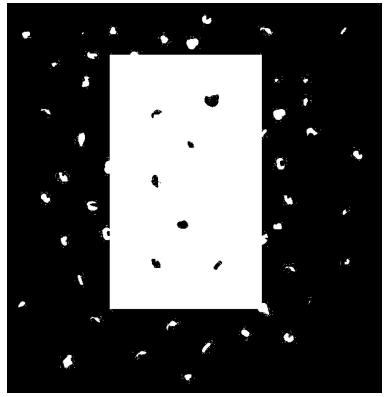


A3=DILATE(A2,B)



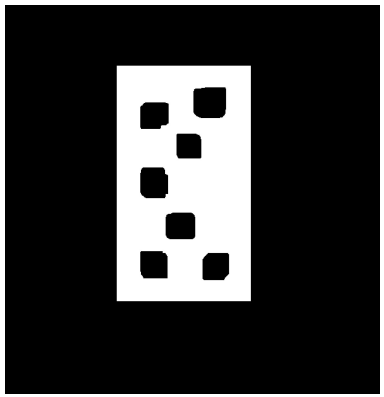
A4=ERODE(A3,B)

The algorithm does a reasonable job of extracting the object. However, note the rounded corners and small bumps on the sides of image A4. Compare to the original shown on the right.

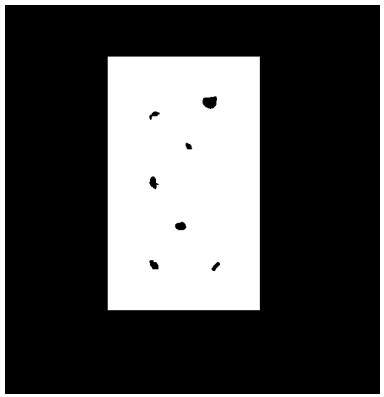


Original Image

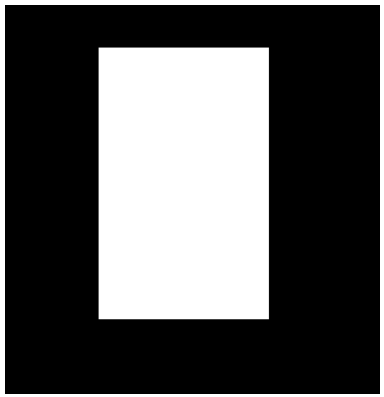
G&W Problem 9.17 (cont)



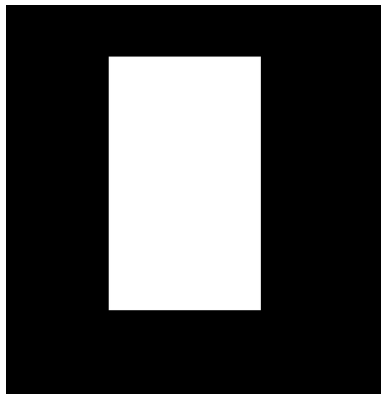
A1=ERODE(A,B)



A2=DILATE(A1,B)

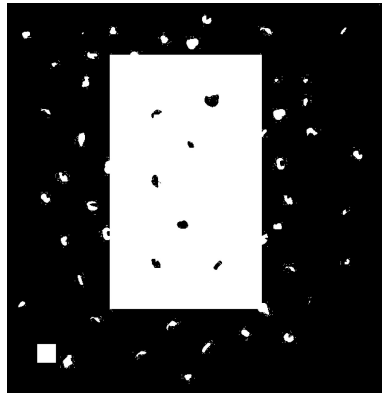


A3=DILATE(A2,B)



A4=ERODE(A3,B)

Using a square structuring element of size 31x31 is an improvement. The white rectangle is now recovered perfectly.



Original Image

Region Filling

Develop an algorithm to fill in bounded regions of background color.

1. Let B be a structuring element of type N_4, N_d or N_8, depending on the desired connectivity.

2. Select a pixel p inside a background color region.

3. Initialize X_0 to be an array of background pixels except $X_0[p] = 1$.

4. Perform the iteration

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

   until $X_k = X_{k-1}$.

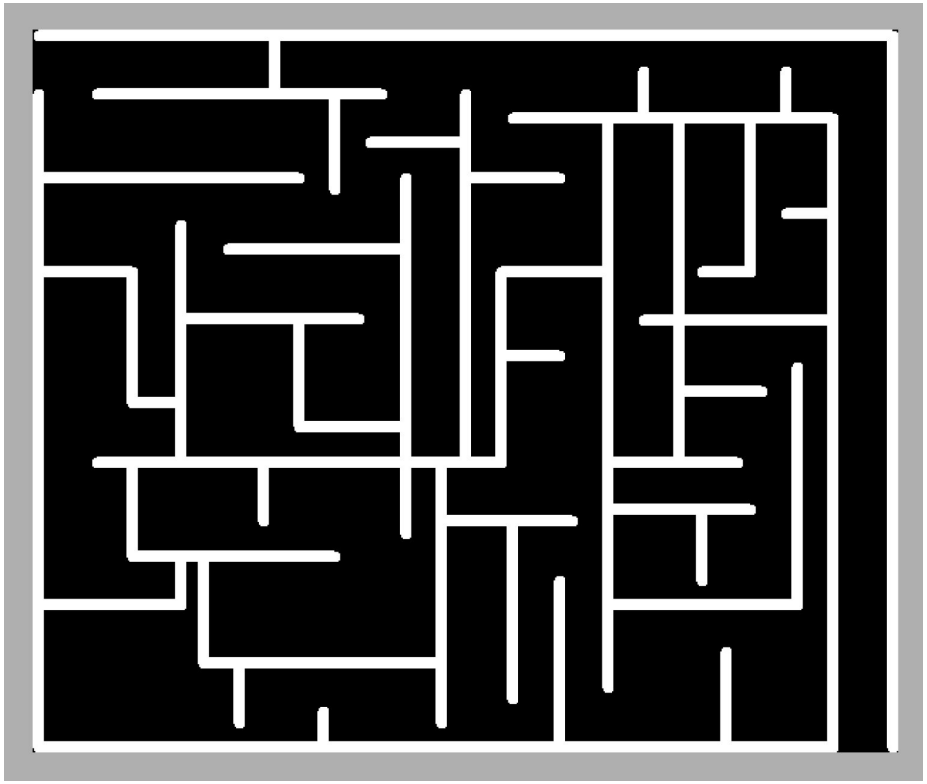
Region Filling Program

```
Function Region_Fill,A,start,filter=B,max_iterations=max_iterations


IF n_params() LT 2 THEN Message,'Too Few Arguments to Function'
IF NOT KEYWORD_SET(B) THEN B=[[0b,1b,0b],[1b,1b,1b],[0b,1b,0b]]
IF NOT KEYWORD_SET(max_iterations) THEN max_iterations=10000L


sa=Size(A,/dim)
X=BytArr(sa[0],sa[1]) & X[start]=1b
Y=BytArr(sa[0],sa[1])
Ac=A EQ 0
count=0
WHILE ((min(X EQ Y) EQ 0) AND (count LT max_iterations)) DO BEGIN
    count=count+1
    Y=X
    X=Dilate(Y,B) AND Ac
ENDWHILE
RETURN,X
END
```

Example: Connected Regions in Maze



Example: Connected Regions in Maze