

# Chapter 9:

# Image

# Segmentation

## 9.1 Introduction

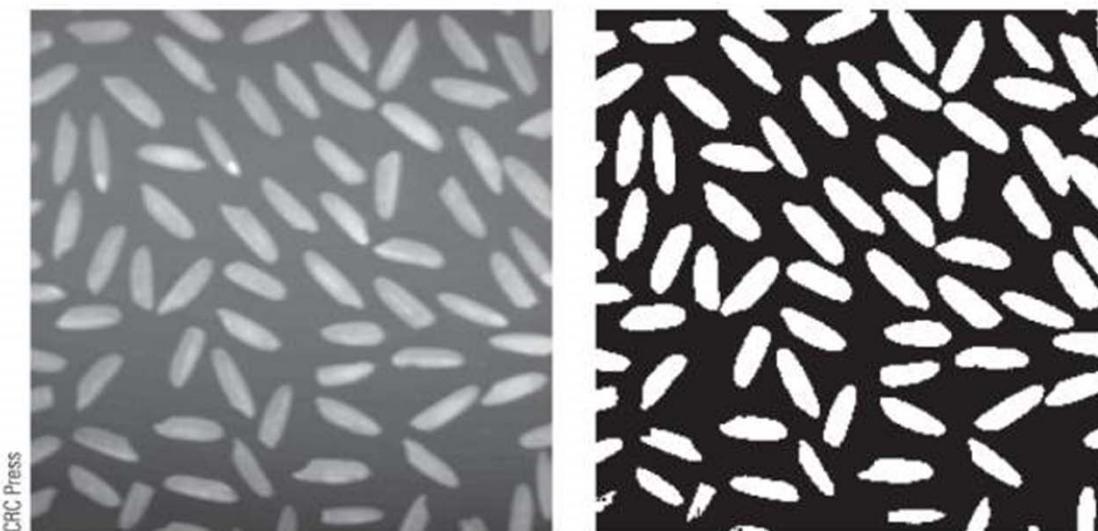
- **Segmentation** is the operation of partitioning an image into component parts or into separate objects.
- In this chapter, we will investigate two very important topics:  
**thresholding** and **edge detection**
  - Single & double thresholding
  - How to determine the threshold value
  - Adaptive thresholding
  - Edge detection: 1st and 2nd derivatives
  - Canny edge detector
  - Hough Transform

# Single Thresholding

- grayscale image -> binary image

A pixel becomes  $\begin{cases} \text{white if its gray level is } > T, \\ \text{black if its gray level is } \leq T. \end{cases}$

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



CRC Press

FIGURE 9.1 Thresholded image of rice grains.

# Single Thresholding

```
>> b=imread('bacteria.tif');  
>> imshow(b), figure, imshow(b>100)
```

CRC Press

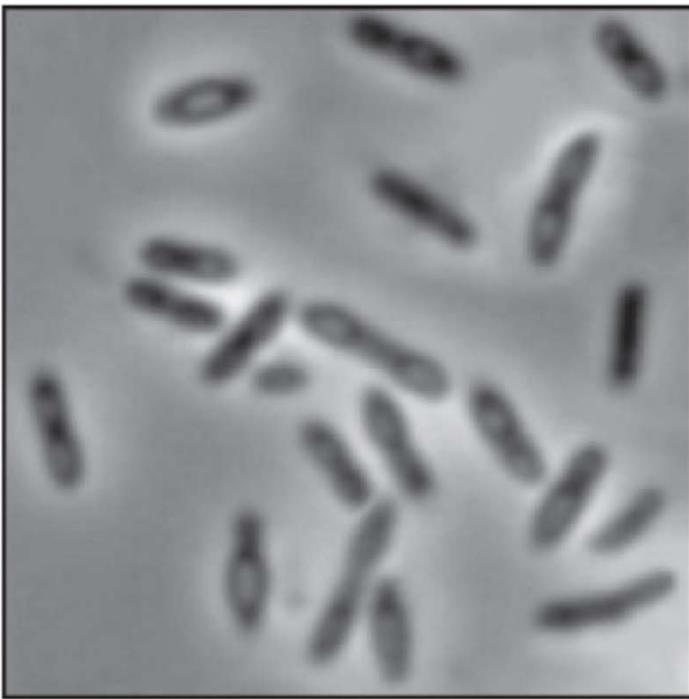


FIGURE 9.1 Thresholded image of bacteria.

# Single Thresholding in Matlab: im2bw()

- MATLAB has the **im2bw** function, which thresholds an image, using the general syntax.
- It works on grayscale, colored, and indexed images of data type uint8, uint16, or double.

```
im2bw(image,level)
```

e.g.

```
>> im2bw(r,0.43);  
>> im2bw(b,0.39);
```

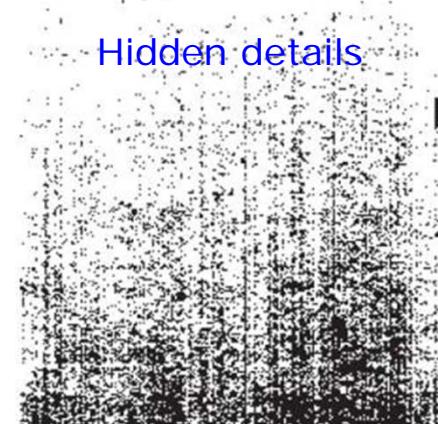
- choosing an appropriate value of T is important.

```
>> p=imread('paper1.tif');  
>> imshow(p),figure,imshow(p>241)
```

Nearly all gray  
values are very  
high.



Hidden details



# Double Thresholding

- grayscale image with two thresholds -> binary image

a pixel becomes  $\begin{cases} \text{white if its gray level is between } T_1 \text{ and } T_2, \\ \text{black if its gray level is otherwise.} \end{cases}$

```
>> [x,map]=imread('spine.tif');
>> s=ind2gray(x,map);
>> imshow(s),figure,imshow(s>115 & s<125)
```

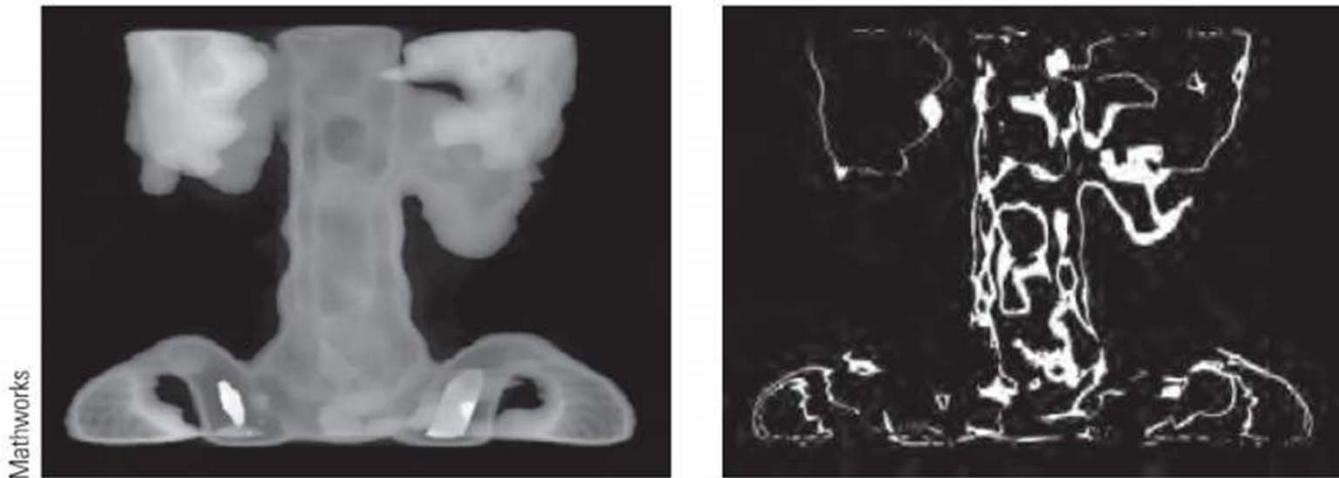


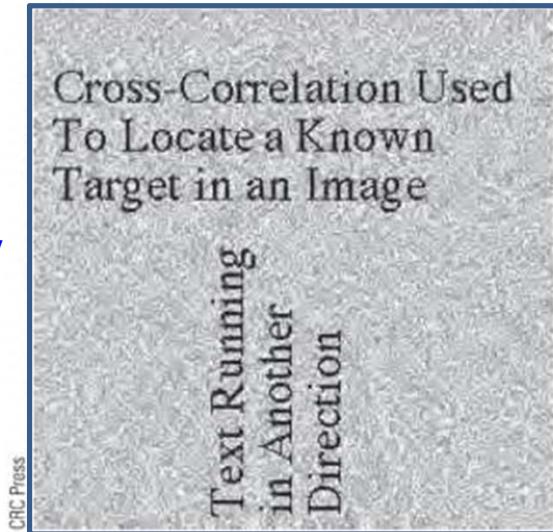
FIGURE 9.4 The image *spine.tif* as the result after double thresholding.

# Applications of Thresholding

- Remove unnecessary detail: e.g. rice and bacteria images
- bring out hidden detail : e.g. paper and spine images
- remove a varying background from text or a drawing

```
>> r=rand(256)*128+127;  
>> t=imread('text.tif');  
>> tr=uint8(r.*double(not(t)));  
>> imshow(tr)  
>> imshow(tr>100)
```

text in noisy  
background



segmentation  
by single  
thresholding

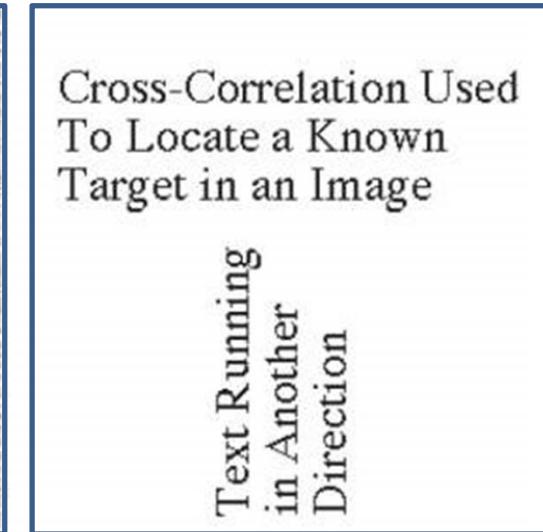
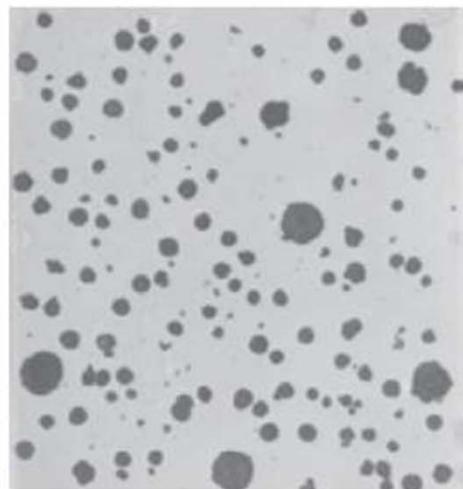


FIGURE 9.5 Text on a varying background and thresholding.

# Choosing an Appropriate Thresholding Value

```
>> n=imread('nodules1.tif');
>> imshow(n);
>> n1=im2bw(n,0.35);
>> n2=im2bw(n,0.75);
>> figure,imshow(n1),figure,imshow(n2)
```

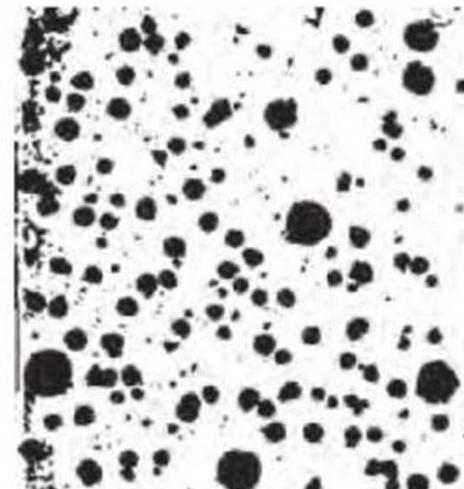
Mathworks



n: Original image



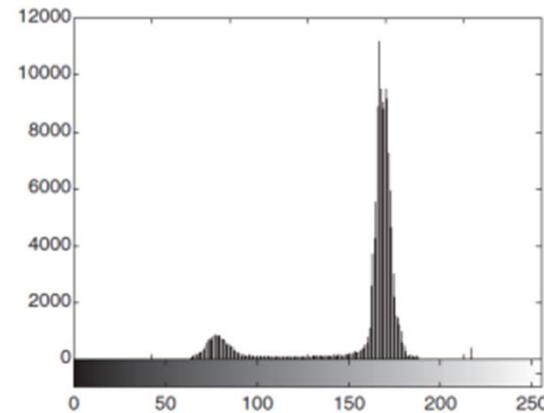
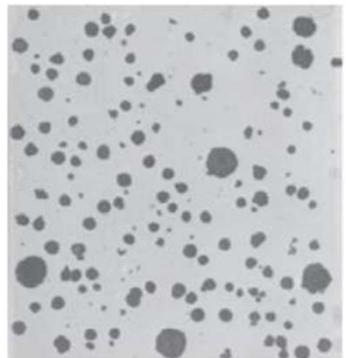
n1: Threshold too low



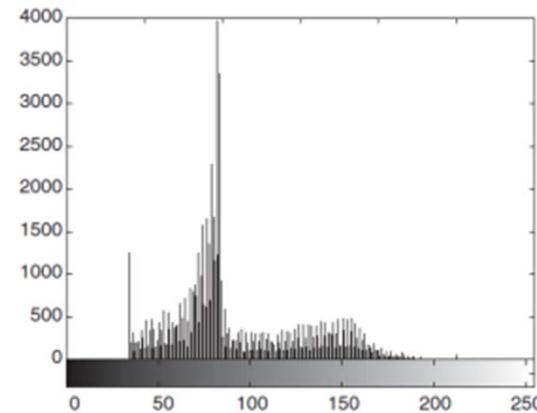
n2: Threshold too high

FIGURE 9.6 Attempts at thresholding.

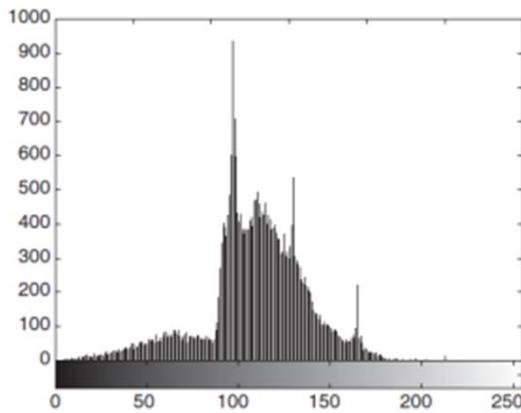
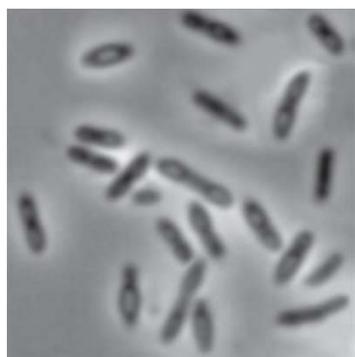
# Histogram Analysis



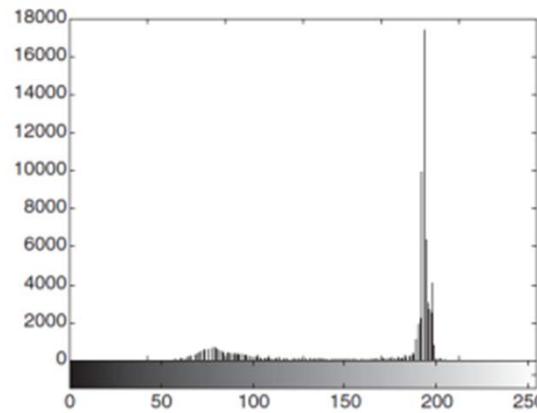
Nodules image : nodules.tif



Rice image : rice.tif



Bacteria image : bacteria.tif



# Histogram Analysis

- Choosing a threshold value is easy only if we have a prior knowledge of individual histograms of objects and background.
- However, in general, histogram of object and background is overlaped so that we cannot easily determine the appropriate threshold values.

-> needs more intelligent way to choose an optimal threshold.

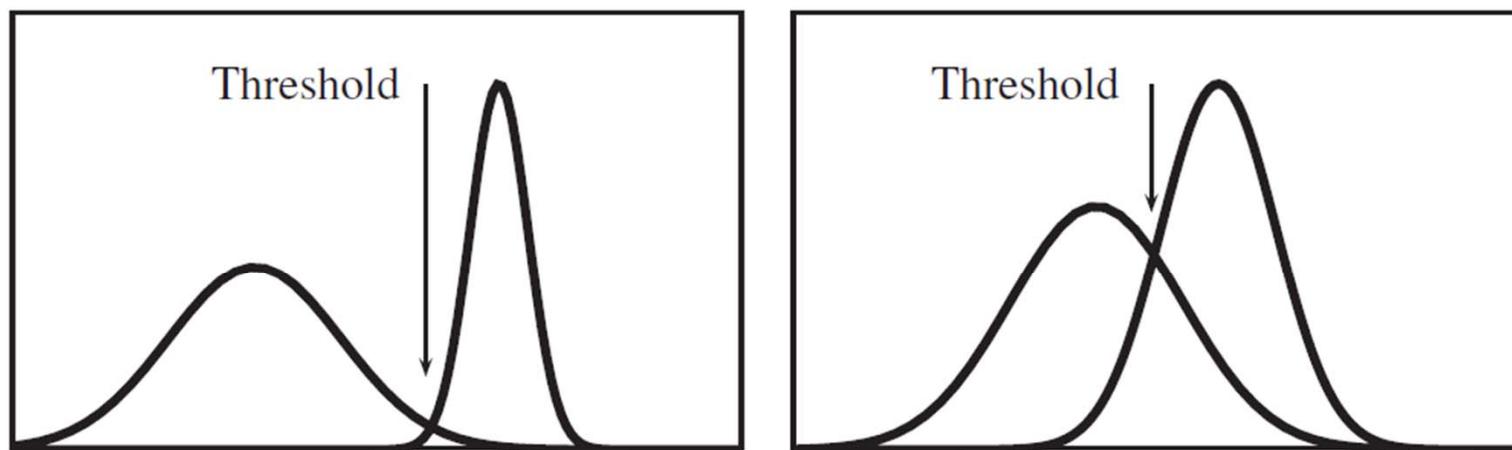


FIGURE 9.8 *Splitting up a histogram for thresholding.*

# Otsu's Method: Choosing an Appropriate Thresholding Value

- Good classifier should be able to divide two classes so that each class has a small variance.
- method 1: Find threshold value  $t$  that makes sum of variances of two classes becomes minimum.

$$t = \arg \min_k [\omega_1(k) \text{var}_1(k) + \omega_2(k) \text{var}_2(k)]$$

minimizes within-class variance

$$\omega_1(k) = \sum_{i=0}^k p_i, \quad \omega_2(k) = \sum_{i=k+1}^{255} p_i$$

- method 2(faster): Find threshold value  $t$  that makes difference of weighted average of two classes becomes maximum.

$$t = \arg \max_k [\omega_1(k) \omega_2(k) (\mu_1(k) - \mu_2(k))^2]$$

maximize between-class variance

- Matlab command: `graythresh()`
- [http://en.wikipedia.org/wiki/Otsu's\\_method](http://en.wikipedia.org/wiki/Otsu's_method)

# Example of Otsu's Method

```
>> tn=graythresh(n)  
  
tn =  
  
    0.5804  
  
>> r imread('rice.tif');  
>> tr=graythresh(r)  
  
tr =  
  
    0.4902  
  
>> b imread('bacteria.tif');  
>> tb=graythresh(b)  
  
tb =  
  
    0.3765  
  
>> e imread('eight.tif');  
>> te=graythresh(e)  
  
te =  
  
    0.6490  
  
>> imshow(im2bw(n,tn))  
>> figure,imshow(im2bw(r,tr))  
>> figure,imshow(im2bw(b,tb))  
>> figure,imshow(im2bw(e,te))
```

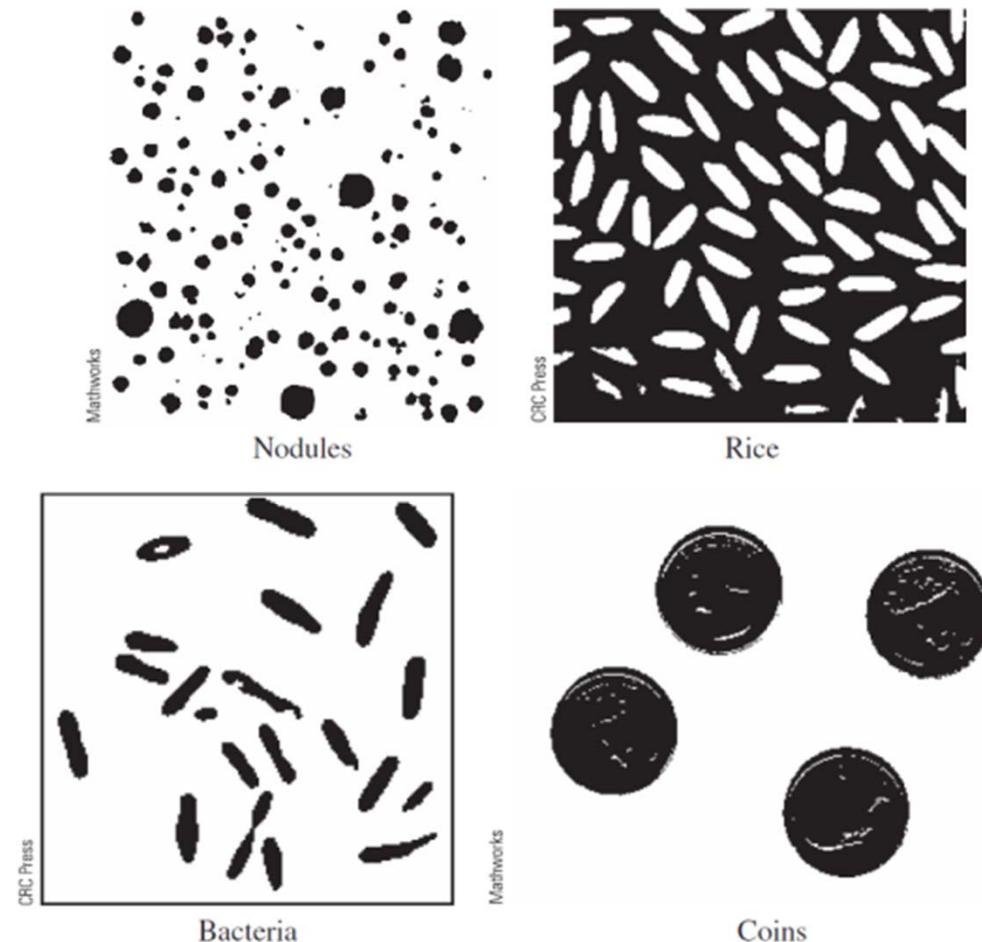


FIGURE 9.9 Thresholding with values from *graythresh*.

# Adaptive Thresholding

- Sometimes a single global threshold values (even if it was extracted by Otsu's method) does not work.
- Needs to determine the threshold values in each image block depending on the characteristic of each block.
  - > adaptively thresholding

```
>> c=imread('circles.tif');
>> x=ones(256,1)*[1:256];
>> c2=double(c).* (x/2+50)+(1-double(c)).*x/2;
>> c3=uint8(255*mat2gray(c2));
```



c3 from circles.tif



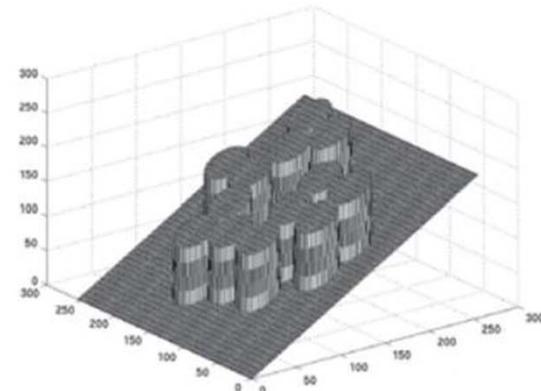
global thresholding

```
>> t=graythresh(c3)
t =
0.4196
>> ct=im2bw(c3,t);
```

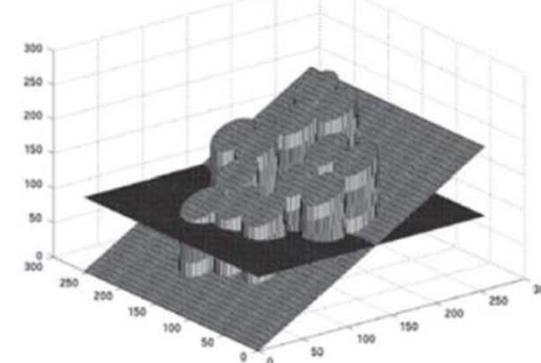
# Adaptive Thresholding

Problem  
determination by  
3D surface plots

surface plot of c3



global threshold  
(black plane) onto c3



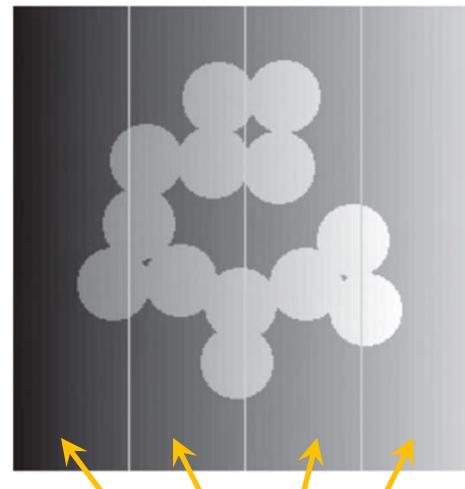
**solution:** apply blockwise thresholding adaptively

blkproc and  
im2bw: pp233

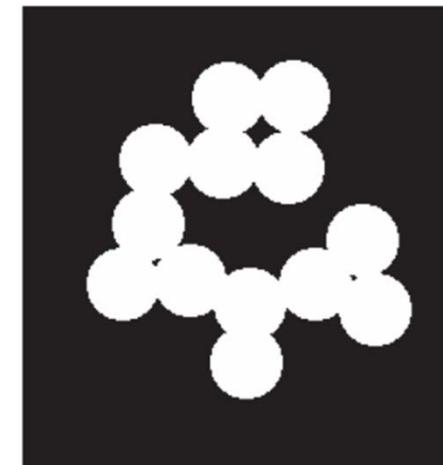
```
>> p1=c3(:,1:64);  
>> p2=c3(:,65:128);  
>> p3=c3(:,129:192);  
>> p4=c3(:,193:256);
```

```
>> g1=im2bw(p1,graythresh(p1));  
>> g2=im2bw(p2,graythresh(p2));  
>> g3=im2bw(p3,graythresh(p3));  
>> g4=im2bw(p4,graythresh(p4));
```

```
>> imshow([g1 g2 g3 g4])
```



4 sub-blocks



# Edge Detection in Matlab

- The general MATLAB command for finding edges is

```
edge(image, 'method', parameters . . . )
```

51	52	53	59
54	52	53	62
50	52	53	68
55	52	53	55

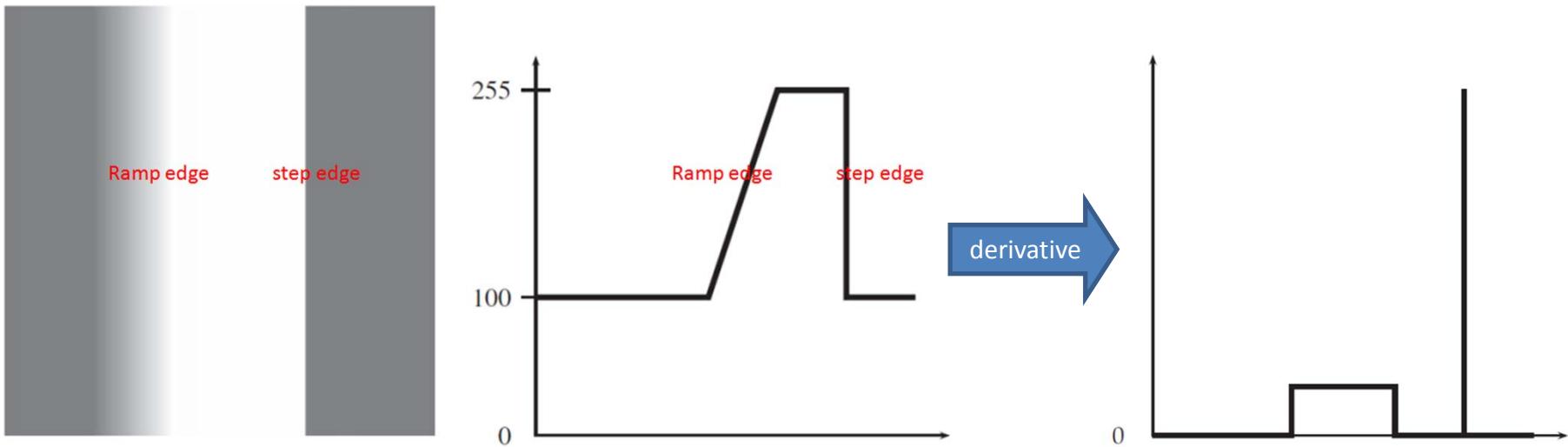
(a)

50	53	155	160
51	53	160	170
52	53	167	190
51	53	162	155

(b)

FIGURE 9.13 Blocks of pixels.

# Derivatives and Edges



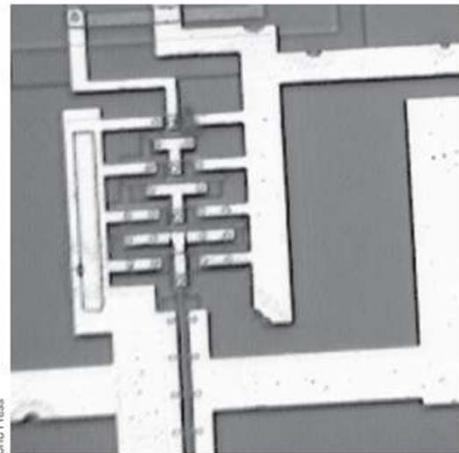
- derivative  $f'(x) = df/dx$
- discrete version:  $f(x+1)-f(x)$ ,  $f(x)-f(x-1)$ ,  $(f(x+1)-f(x-1))/2$
- Expansion into 2D image -> **gradient**

$$\begin{bmatrix} \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} \end{bmatrix} \xrightarrow{\text{magnitude}} \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

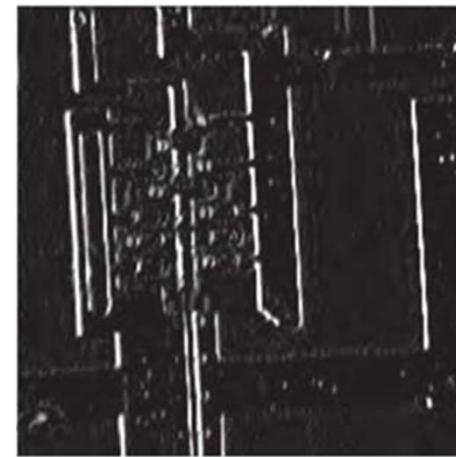
# Edge Detection Filters: Prewitt

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

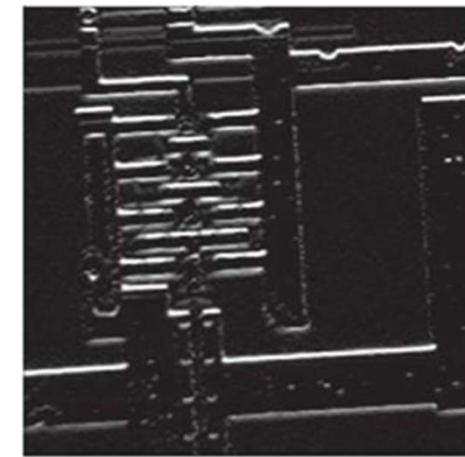
```
>> ic=imread('ic.tif');
```



```
>> px=[-1 0 1;-1 0 1;-1 0 1];  
>> icx=filter2(px,ic);  
>> figure,imshow(icx/255)
```



```
>> py=px';  
>> icy=filter2(py,ic);  
>> figure,imshow(icy/255)
```

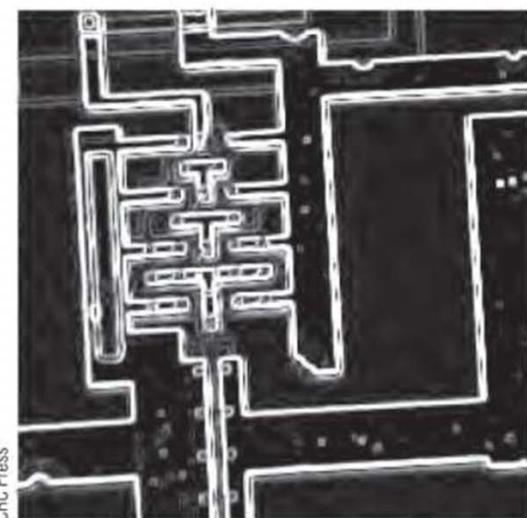


Prewitt filter: Px

Prewitt filter: Py

# Edge Detection by Prewitt Filter

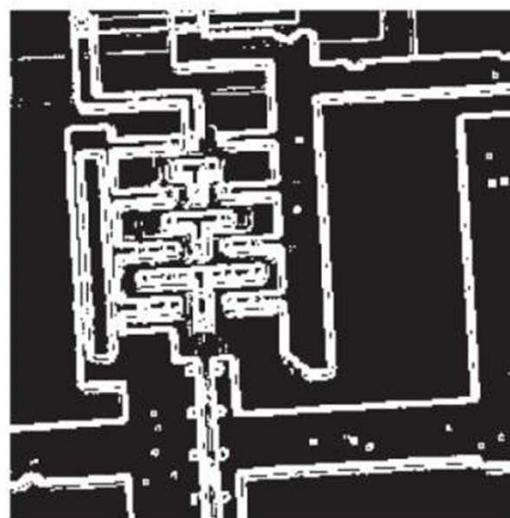
magnitude of  
gradient images  
(icx and icy)  
-> gray scale



(a)

```
>> edge_p=sqrt(icx.^2+icy.^2);  
>> figure,imshow(edge_p/255)
```

Thresholding  
magnitude of  
gradient images  
-> binary



(b)

```
>> edge_t=im2bw(edge_p/255,0.3);
```

[edge\(\)](#) in matlab:  
Prewitt magnitude +  
thresholding + some  
extra processing

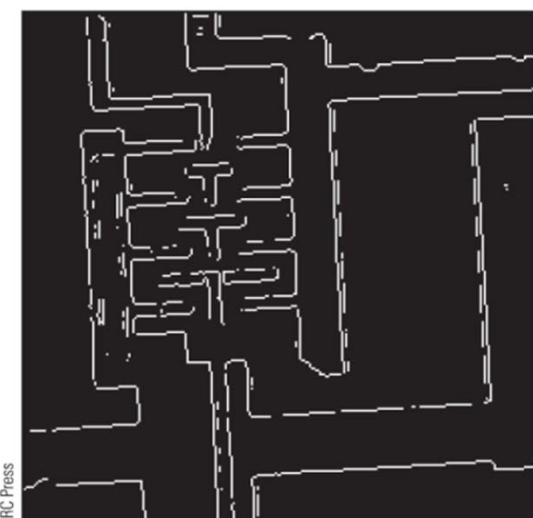


FIGURE 9.19 The prewitt option of `edge`.

```
>> edge_p=edge(ic,'prewitt');
```

# More Edge Detection Filters in Matlab

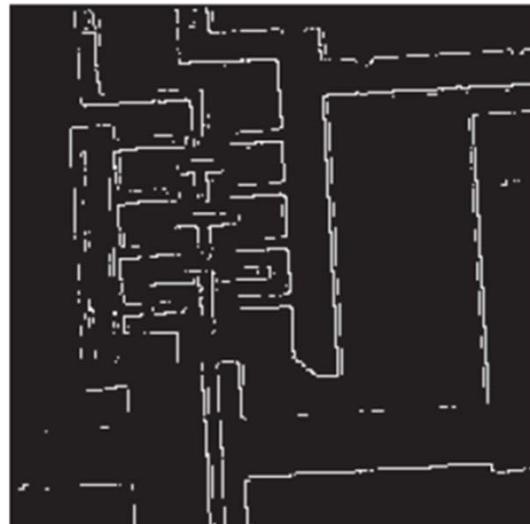
- Roberts cross-gradient filters:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

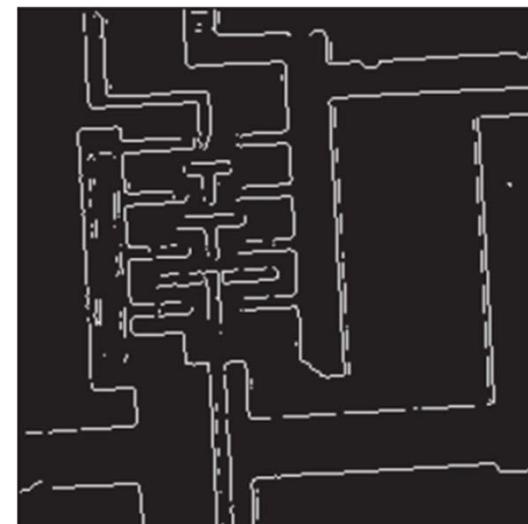
- Sobel filters:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} -1 & -2 & 1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

```
>> edge_r=edge(ic,'roberts');  
>> figure,imshow(edge_r)
```



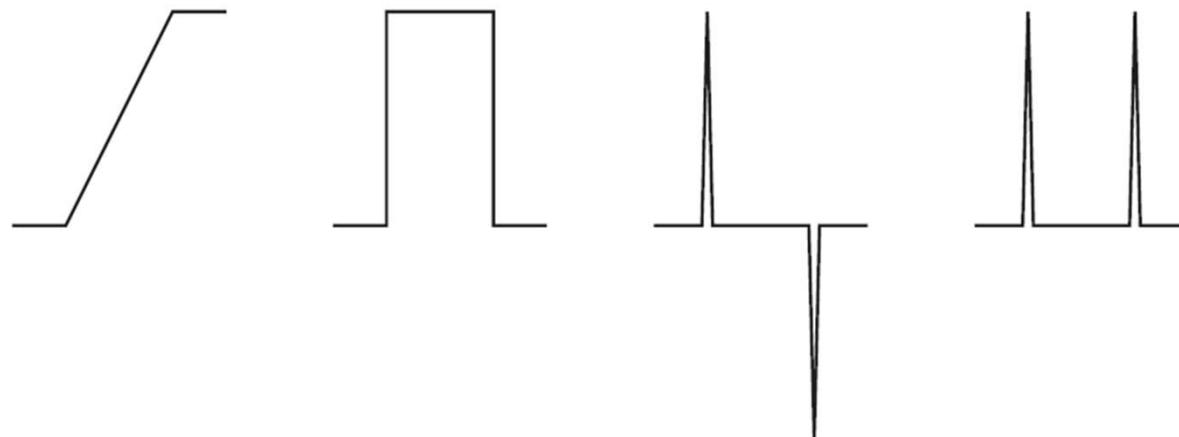
```
>> edge_s=edge(ic,'sobel');  
>> figure,imshow(edge_s)
```



# Second Derivatives for Edge Detection

- The Laplacian:  $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$  discrete version  $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

- ✓ Pros: isotropic (= rotation invariant) filter
- ✓ Cons: very sensitive to noise

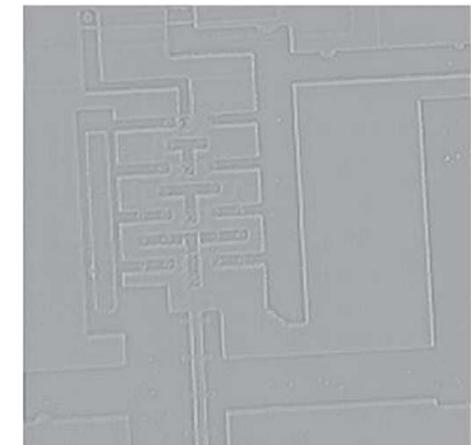


The edge

First derivative

Second derivative

Absolute values



```
>> l=fspecial('laplacian',0);
>> ic_l=filter2(l,ic);
>> figure,imshow(mat2gray(ic_l))
```

# Edge Detection by Zero Crossing

- The position where the result of the filter **changes sign**.
- E.g., the simple image given below and the result after filtering with a Laplacian mask
- **What if we simply take all zero crossing points as edges ?**

50	50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50	50
50	50	200	200	200	200	200	200	50	50	50
50	50	200	200	200	200	200	200	50	50	50
50	50	200	200	200	200	200	200	50	50	50
50	50	200	200	200	200	200	200	50	50	50
50	50	50	50	200	200	200	200	50	50	50
50	50	50	50	200	200	200	200	50	50	50
50	50	50	50	50	50	50	50	50	50	50
50	50	50	50	50	50	50	50	50	50	50

(a)

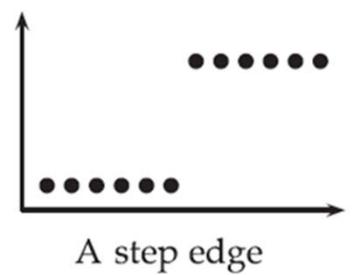
-100	-50	-50	-50	-50	-50	-50	-50	-50	-100
-50	0	150	150	150	150	150	150	0	-50
-50	150	-300	-150	-150	-150	-150	-300	150	-50
-50	150	-150	0	0	0	0	-150	150	-50
-50	150	-150	0	0	0	0	-150	150	-50
-50	150	-300	-150	0	0	0	-150	150	-50
-50	0	150	300	-150	0	0	-150	150	-50
-50	0	0	150	-300	-150	-150	-300	150	-50
-50	0	0	0	150	150	150	150	0	-50
-100	-50	-50	-50	-50	-50	-50	-50	-50	-100

(b)

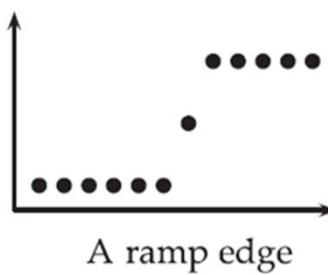
FIGURE 9.23 Locating zero crossings in an image. (a) A simple image. (b) After laplace filtering.

# Edge Detection by Zero Crossing

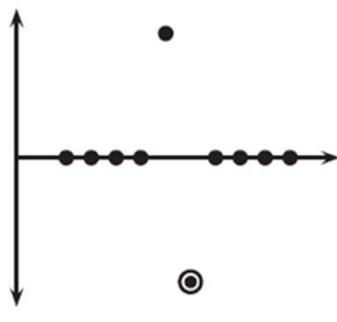
- Two cases of zero crossing
  - ✓ They have a negative gray value and are orthogonally adjacent to a pixel whose gray value is positive.
  - ✓ They have a value of zero and are between negative- and positive-valued pixels



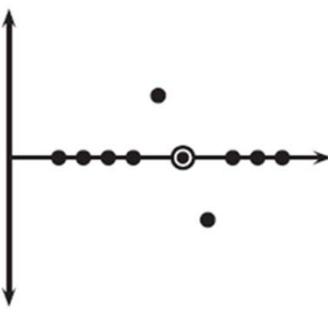
A step edge



A ramp edge

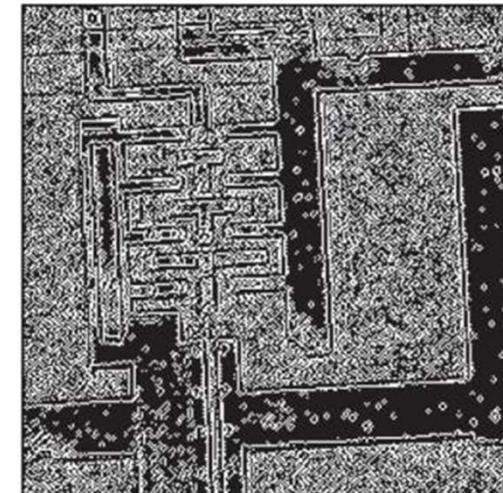


Its second differences



Its second differences

```
>> l=fspecial('laplace',0);  
>> icz=edge(ic,'zerocross',1);  
>> imshow(icz)
```



Too many edges are detected.  
How can we improve the results?  
-> [Marr-Hildreth Method](#)

# Edge Detection by Zero Crossing

- **Marr-Hildreth method**

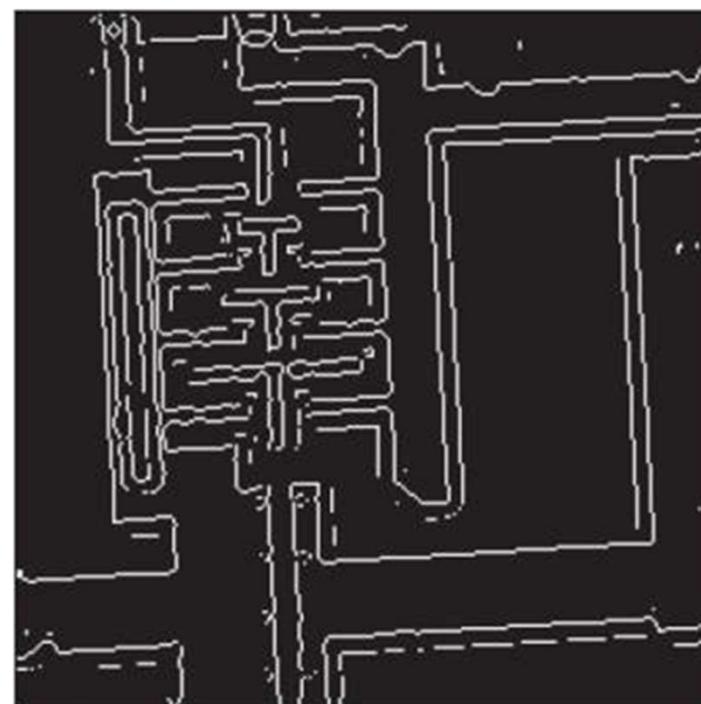
- ✓ Smooth the image with a Gaussian filter.
- ✓ Convolve the result with a Laplacian filter.
- ✓ Find the zero crossings.

Laplacian of  
Gaussian  
(LoG)

```
>> fspecial('log', 13, 2);  
>> edge(ic, 'log');
```

or

```
>> log=fspecial('log',13,2);  
>> edge(ic,'zerocross',log);
```



# Canny Edge Detector

- Proposed by John Canny in 1986.
- The most popular edge filter ever !
- Stage 1: 1D DoG filters -> result:  $x_e(x,y)$ 
  - ✓ Take our image  $x$
  - ✓ Create a one-dimensional Gaussian filter  $g$
  - ✓ Create a **1D filter**  $dg$  corresponding to the expression given in

$$\frac{d}{dx} e^{-\frac{x^2}{2\sigma^2}} = \left( -\frac{x^2}{\sigma^2} \right) e^{-\frac{x^2}{2\sigma^2}}$$



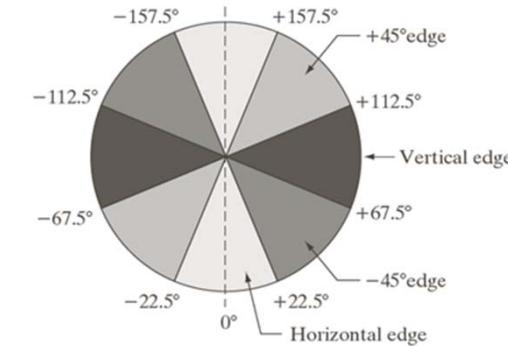
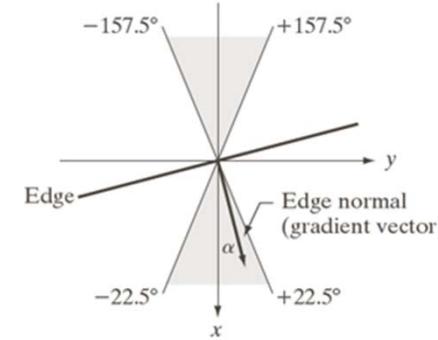
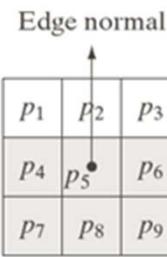
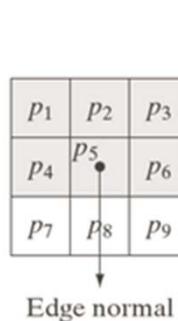
- Gaussian, then derivative of Gaussian
- to smooth the noise and find possible candidate pixels for edges
- separable (1D filtering twice)

- ✓ Convolve  $g$  with  $dg$  to obtain  $gdg$
- ✓ Apply  $gdg$  to  $x$  producing  $x_1$
- ✓ Apply  $gdg'$  to  $x$  producing  $x_2$
- ✓ Form an magnitude of edge  $x_e$  with the equation

$$x_e = \sqrt{x_1^2 + x_2^2}$$

# Canny Edge Detector

- Stage 2: Non-maxima suppression -> result:  $g_N(x,y)$ 
  - Rejects pixels detected as edges in the Stage 1 by detecting the maximum derivative among adjacent pixels.
  - Thresholding alone is not a good way to detection edges in the result  $xe$ .
  - ✓ Calculate edge normal(gradient vector)  $xg$  from the 1D gradient images  $xl$  and  $x2$ .
 
$$xg = \tan^{-1} \left( \frac{x2}{x1} \right)$$
  - ✓ Because it is generated using the gradient,  $xg$  typically contains wide ridges around local maxima. Thus, we needs to thin those ridges.
  - Based on  $xg$ , quantize edge orientations into the predetermined steps. (e.g. four orientations in 3x3 region: horizontal, vertical,  $+45^\circ$ ,  $-45^\circ$ )
  - ✓ If a value of  $xe(x,y)$  is less than at least one of its two neighbors along  $xg_i$ , let  $g_N(x,y) = 0$  (suppression); otherwise, let  $g_N(x,y) = xe(x,y)$ .



# Canny Edge Detector

- Stage 3: Hysteresis thresholding for binary edge image
  - uses two threshold values:  $T_L$  and  $T_H$  for lower and higher bounds respectively.
    - ✓  $g_{NH}(x,y) = g_N(x,y) \geq T_H$  : strong edge pixels
    - ✓  $g_{NL}(x,y) = g_N(x,y) \geq T_L$
    - ✓  $g_{NL}(x,y) = g_{NL}(x,y) - g_{NH}(x,y)$  : weak edge pixels
    - ✓ Connectivity analysis in  $g_{NH}(x,y)$ 
      1. Locate the next unvisited edge pixel, p, in  $g_{NH}(x,y)$ .
      2. Mark as valid edge pixels all the weak pixels in  $g_{NL}(x,y)$  that are connected to p using 8 connectivity.
      3. If all nonzero pixels in  $g_{NH}(x,y)$  have been processed, go to the next step. Otherwise, return to the first step.
      4. Set to 0 all pixels in  $g_{NL}(x,y)$  that were not marked as valid edge pixels.
    - ✓ Append all the nonzero pixels from  $g_{NL}(x,y)$  to  $g_{NH}(x,y)$  .

# Canny Edge Detector in Matlab

```
>> [icc,t]=edge(ic,'canny');  
>> t  
  
t =  
  
    0.0500    0.1250  
  
>> imshow(icc)
```

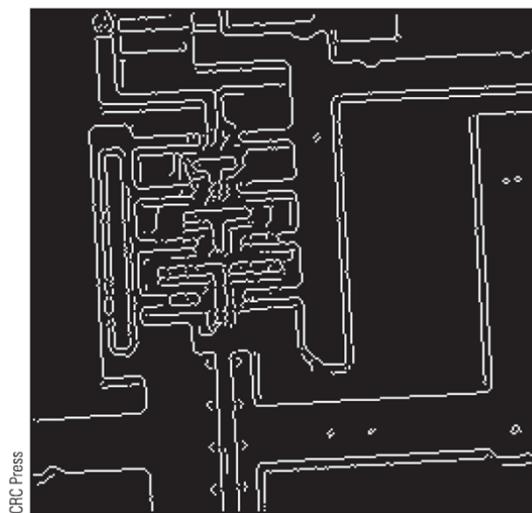


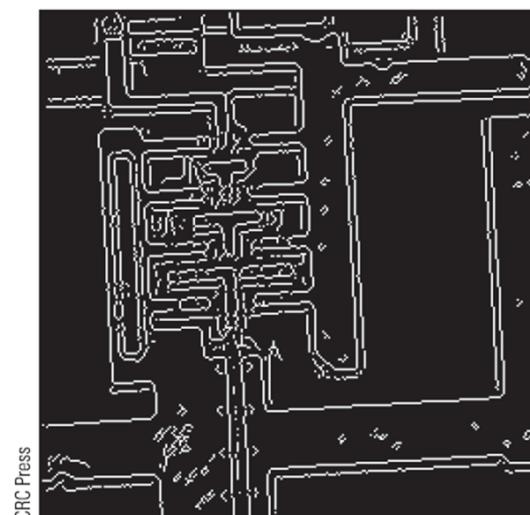
FIGURE 9.29 Canny edge detection.

default threshold and standard deviation

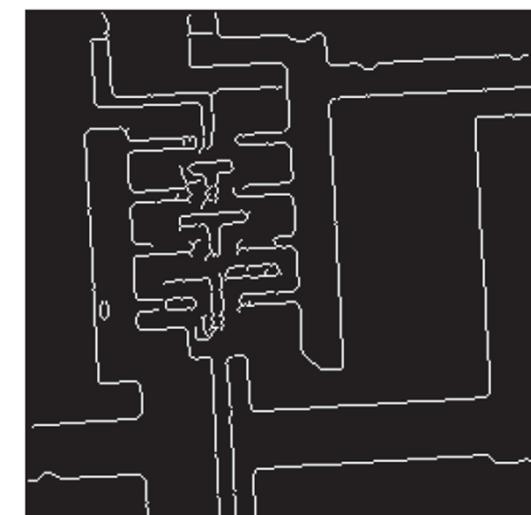
- $BW = \text{EDGE}(I, 'canny', [\text{low}, \text{high}], \text{SIGMA})$

threshold limits      standard deviation

- The higher we make the upper threshold, the fewer edges will be shown.



`edge(ic, 'canny', [0, 0.05])`



`edge(ic, 'canny', [0.01, 0.5])`

FIGURE 9.30 Canny edge detection with different thresholds.

# Hough Transform

- If the detected edge points are sparse, we might need to fit a line to those points. However it is a time-consuming and computationally inefficient process.
- Is there any way to find boundary lines to compensate the problem described above ? => Hough Transform
- Basic idea of Hough Transform
  - A coordinate  $(x,y)$  in an image can be transformed into a line  $y = ax + b$  with all possible pairs of  $(a,b)$ .
  - Every pixel at  $(x,y)$  coordinate in the image can be mapped into a line in the transform array.

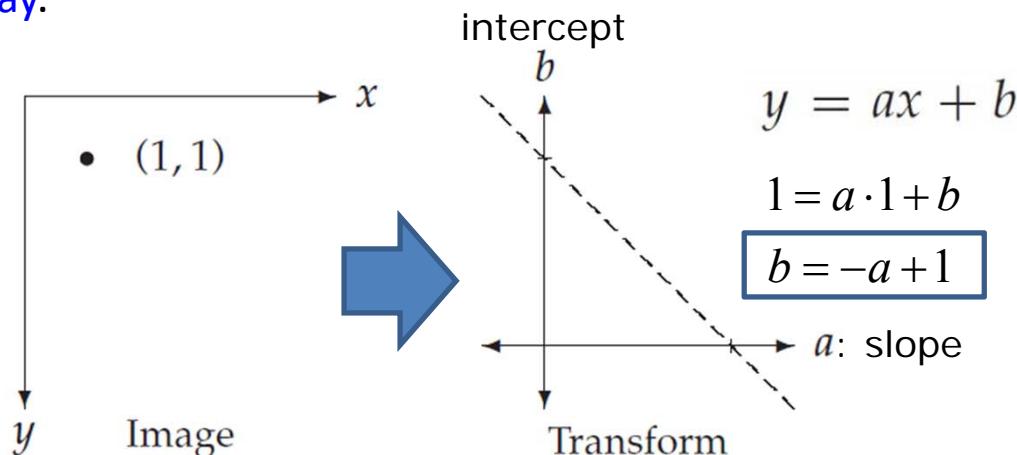


FIGURE 9.31 A point in an image and its corresponding line in the transform.

# Basic Idea of Hough Transform

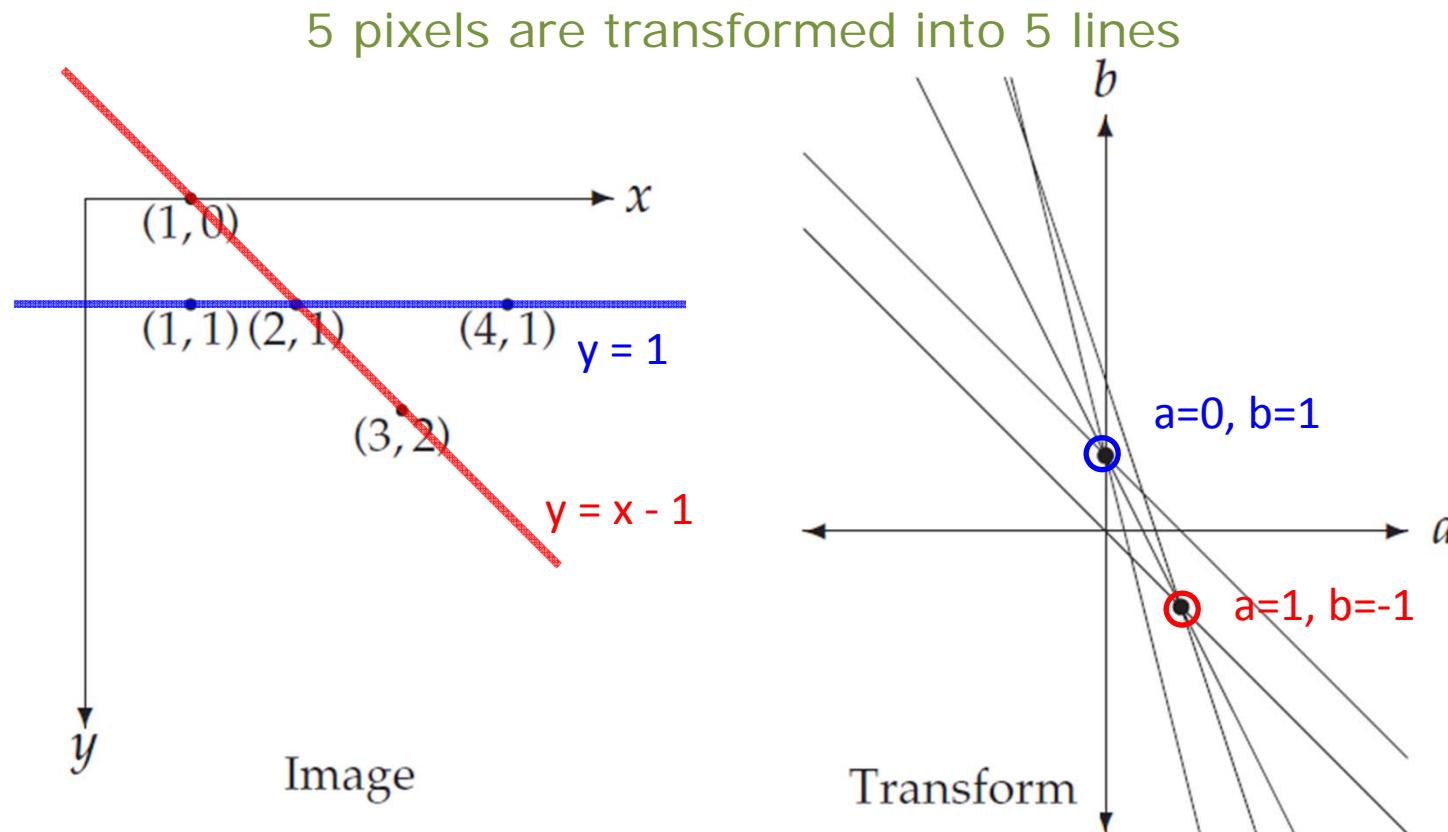
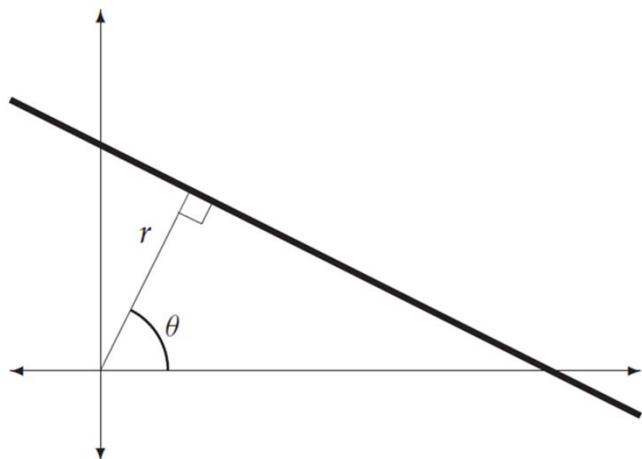


FIGURE 9.32 An image and its corresponding lines in the transform.

- Higher value of cross point  $(a,b)$  in the transformed array indicates more distinguished lines (more pixels through the line) in the image !

# Hough Transform for Generalization

- We cannot express a vertical line in the form  $y = mx + c$ , because  $m$  represents the gradient and a vertical line has infinite gradient.
- Any line can be described in terms of the two parameters  $r$  and  $\theta$ .
  - ✓  $r$  is the perpendicular distance from the line to the origin
  - ✓  $\theta$  is the angle of the line's perpendicular to the  $x$  axis



$$-90 < \theta \leq 90$$

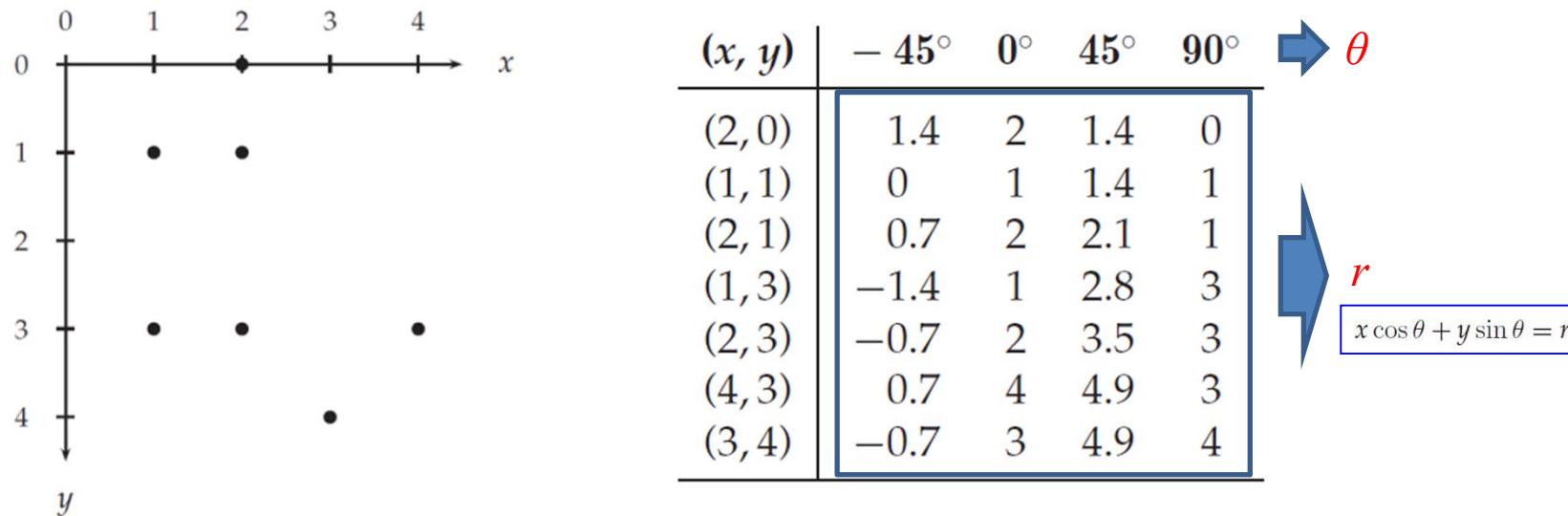
$$x \cos \theta + y \sin \theta = r$$

We can calculate  $\theta$  and  $r$  based on the image coordinate  $(x, y)$ .  
See pp.253 for the derivation.

- Higher value of  $(r, \theta)$  in the transformed array indicates stronger lines (more pixels through the line) in the image.

# Hough Transform Example

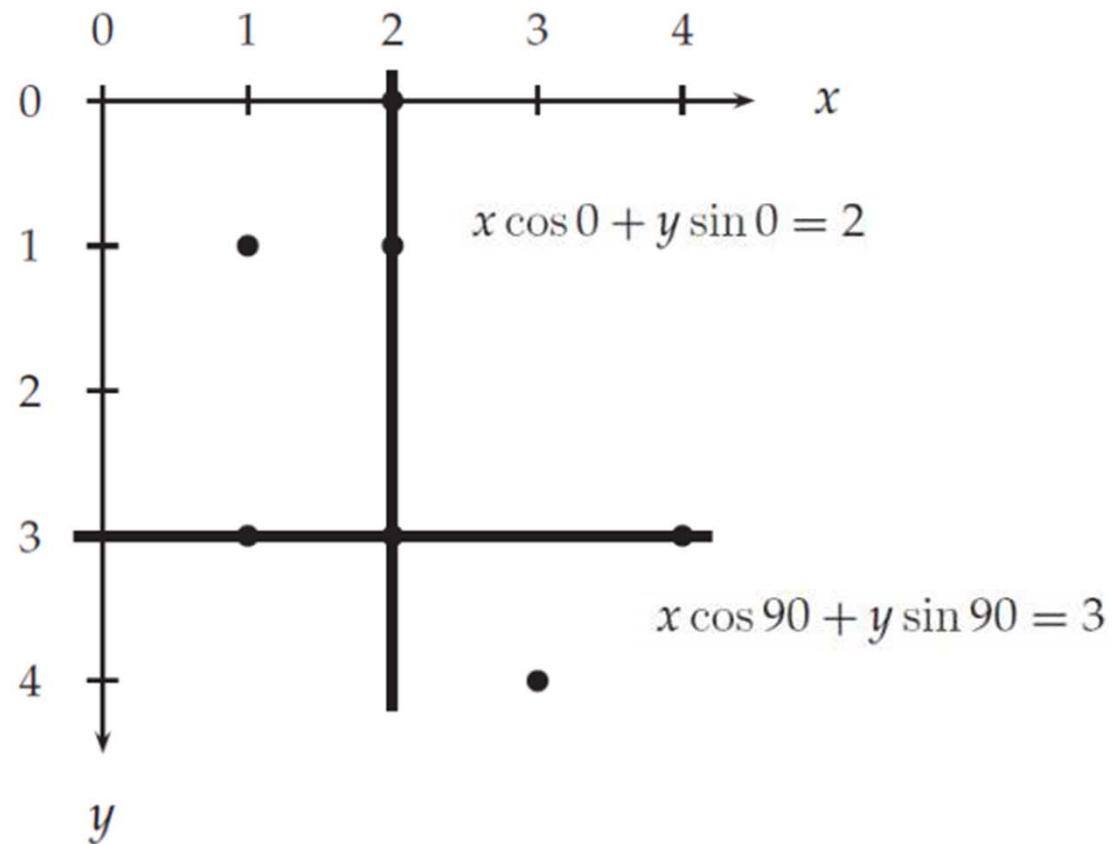
- If we discretize  $\theta$  to use only four values:  $-45^\circ, 0^\circ, 45^\circ, 90^\circ$



- The **accumulator array** contains the **number of times each value of  $(r, \theta)$  appears** in the above table.

	-1.4	-0.7	0	0.7	1	1.4	2	2.1	2.8	3	3.5	4	4.9
$-45^\circ$	1	2	1	2		1							
$0^\circ$					2		3			1		1	
$45^\circ$						2		1	1	1	1	2	
$90^\circ$					1	2				3		2	

# Hough Transform Example



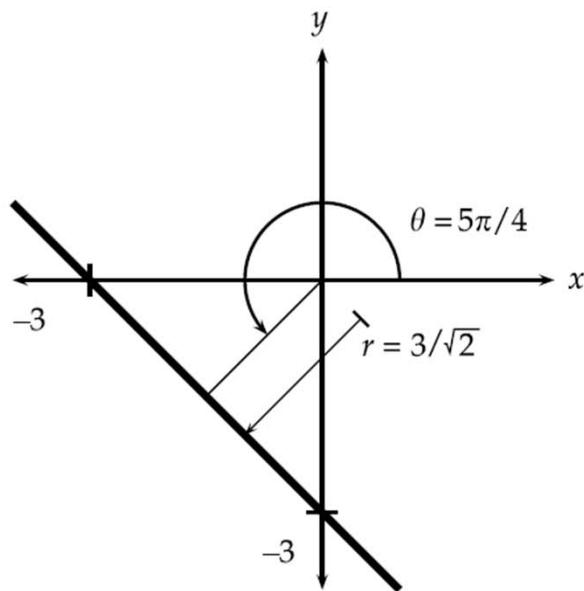
Detected lines

# Implementing Hough Transform in MATLAB

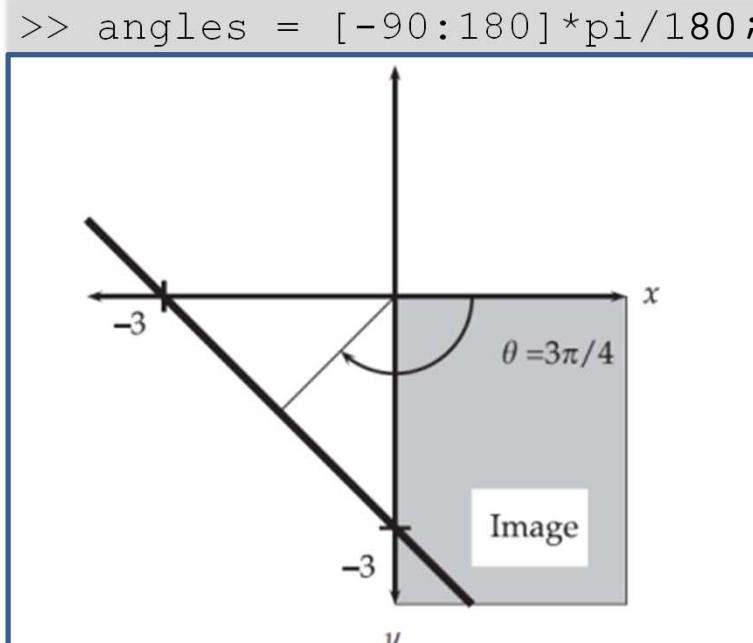
1. Decide on a discrete set of values of  $\theta$  and  $r$  to use.
2. At each foreground pixel  $(x, y)$  in the image, calculate, the values  $r = x\cos\theta + y\sin\theta$  **for all  $\theta$** .
3. Create an **accumulator array** whose sizes are the numbers of angles  $\theta$  and values  $r$  in the chosen discretizations from Step 1, and
4. Step through all of our  $r$  values, updating the accumulator array as we go.

# Hough Transform in MATLAB

- Discretizing  $r$  and  $\theta$ 
  - Let  $\theta$  with the range of  $0 \leq \theta < 2\pi$ .
  - Restrict  $r$  to non-negative value.



(a)



(b)

FIGURE 9.37 A line parameterized with  $r$  and  $\theta$ . (a) Using ordinary Cartesian axes.  
(b) Using matrix axes.

# Example of Hough Transform in MATLAB

- Calculating the  $r$  values
  - ✓ If  $\text{im}$  is a **binary image**

```
>> [x,y]=find(im);
```

We can create a binary edge image by use of the `edge` function beforehand.

```
>> r=floor(x*cos(angles)+y*sin(angles));
```

- Forming the Accumulator Array

```
>> rmax=max(r(find(r>0)));
>> acc=zeros(rmax+1,270);
```

- Updating the Accumulator Array

```
function res=hough2(image)
%
% HOUGH2(IMAGE) creates the Hough transform corresponding to the image IMAGE
%

if ~isbw(image)
    edges=edge(image,'canny');
else
    edges=image;
end;
[x,y]=find(edges);
angles=[-90:180]*pi/180;
r=floor(x*cos(angles)+y*sin(angles));
rmax=max(r(find(r>0)));
acc=zeros(rmax+1,270);
for i=1:length(x),
    for j=1:270,
        if r(i,j)>=0
            acc(r(i,j)+1,j)=acc(r(i,j)+1,j)+1;
        end;
    end;
end;
res=acc;
```

FIGURE 9.38 A simple MATLAB function for implementing the Hough transform.

# Example of Hough Transform in MATLAB

```
>> c=imread('cameraman.tif');  
>> hc=hough2(c);  
  
>> imshow(mat2gray(hc)*1.5)  
  
>> max(hc(:))  
ans =  
91  
  
>> [r,theta]=find(hc==91)  
r =  
138  
  
theta =  
181
```

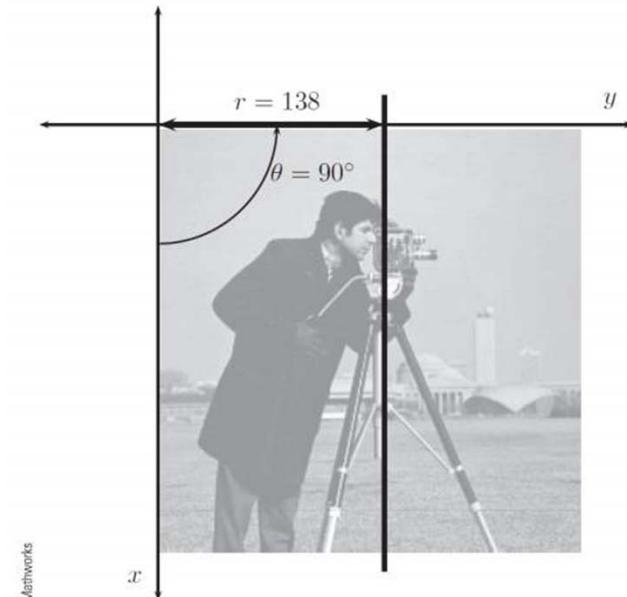
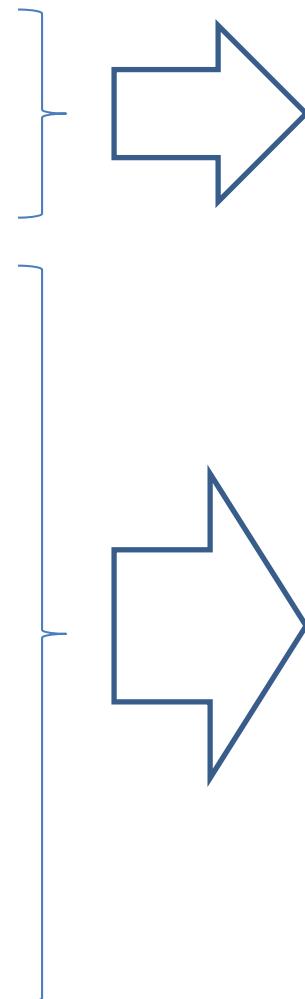


FIGURE 9.40 A line from the Hough Transform.

# How to Display the Detected Line in Matlab

```
function houghline(image,r,theta)
%
% Draws a line at perpendicular distance R from the upper left corner of the
% current figure, with perpendicular angle THETA to the left vertical axis.
% THETA is assumed to be in degrees.
%
[x,y]=size(image);
angle=pi*(181-theta)/180;
X=[1:x];
if sin(angle)==0
    line([r r],[0,y],'Color','black')
else
    line([0,y],[r/sin(angle),(r-y*cos(angle))/sin(angle)],'Color','black')
end;
```



See pp. 261 for more detailed information to use houghline().

# Summary

- **Chap 9. Image Segmentation**
  - Single & double thresholding
  - How to determine the threshold value
  - Adaptive thresholding
  - Edge detection: 1st and 2nd derivatives
  - Canny edge detector
  - Hough Transform
- **Chap 10. Mathematical Morphology**
  - Erosion & dilation -> boundary detection
  - Opening & closing -> noise removal
  - Hit-or-miss transform -> shape detection
  - Binary applications: region filling, connected components, skeletons
  - Grayscale morphology -> edge detection, noise removal



# **Chapter 10:**

# **Mathematical**

# **Morphology**

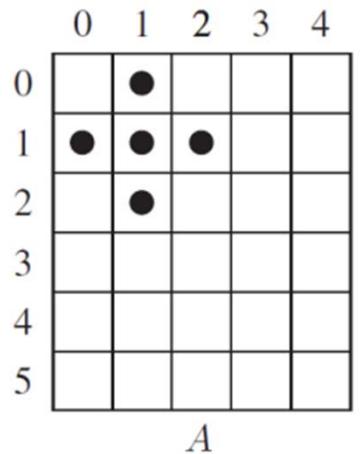
## 10.1 Introduction

- *Morphology* is a branch of image processing that is particularly useful for analyzing **shapes** in images.
- We will develop basic morphological tools for investigation of **binary images** and then show how to extend these tools to **grayscale images**.
- In this chapter, we will learn,
  - What is morphology?
  - Simple morphological operations: **erosion**, **dilation**
  - Compound operations: **opening**, **closing**
  - Morphological algorithms: **boundary detection**, **region filling**, **connected components**, **skeleton**

# Translation & Reflection

- Translation

$$A_w = \{(a, b) + (x, y) : (a, b) \in A\}$$



$$w = (2, 2)$$

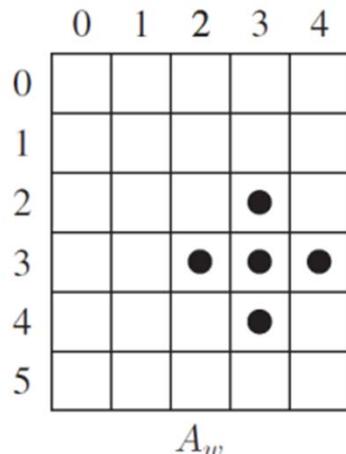


FIGURE 10.1 *Translation.*

- Reflection

$$\hat{A} = \{(-x, -y) : (x, y) \in A\}$$

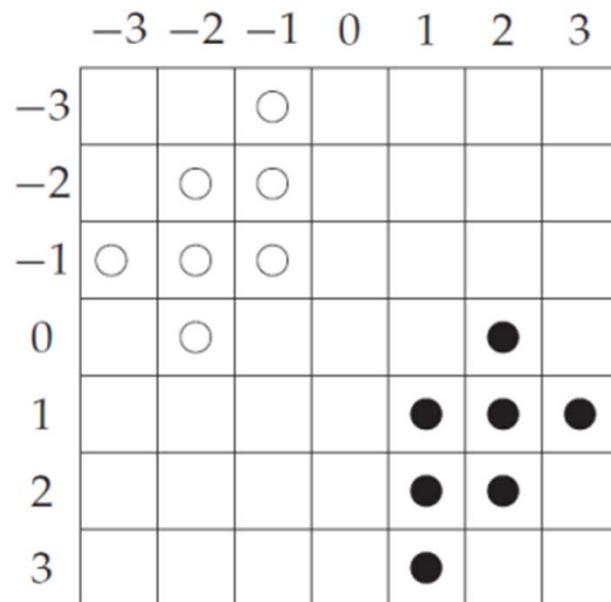


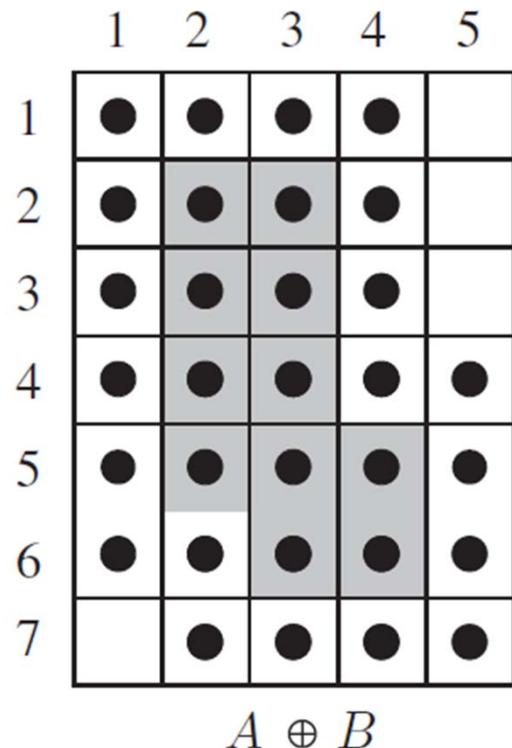
FIGURE 10.2 *Reflection.*

# Dilation

- $A$  and  $B$  are sets of pixels
- Also known as **Minkowski addition**

$$A \oplus B = \{(x, y) + (u, v) : (x, y) \in A, (u, v) \in B\}$$

$$A \oplus B = \bigcup_{x \in B} A_x$$



structuring  
element (SE)

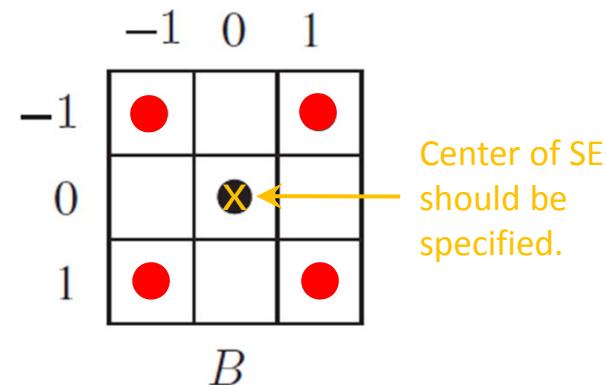


FIGURE 10.3 *Dilation*.

# Dilation in Matlab

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

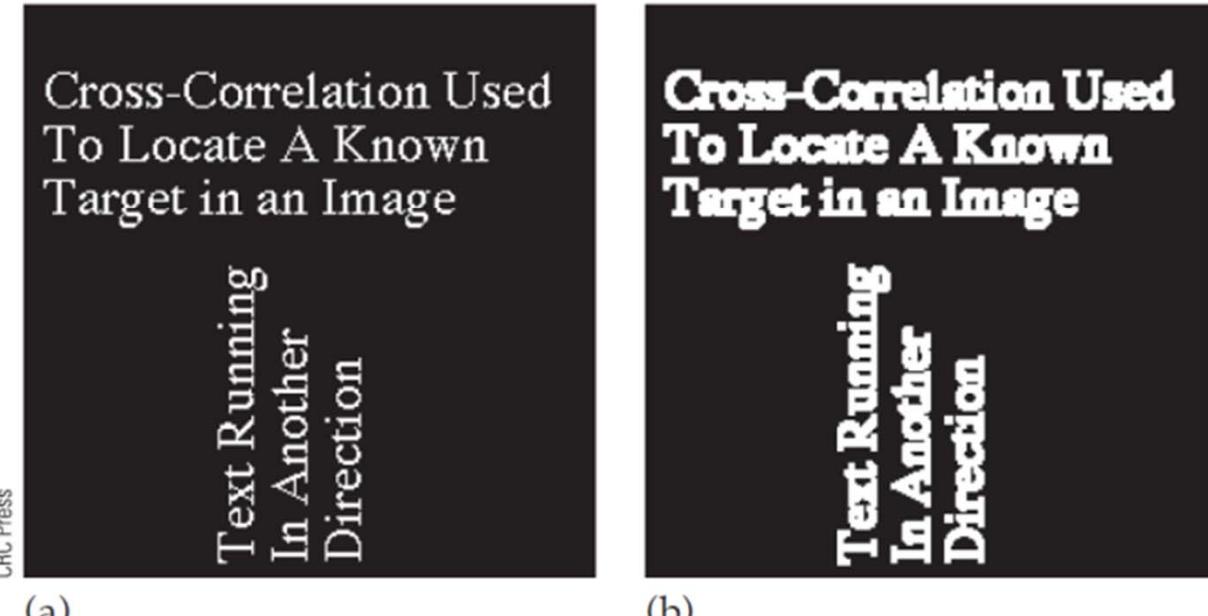
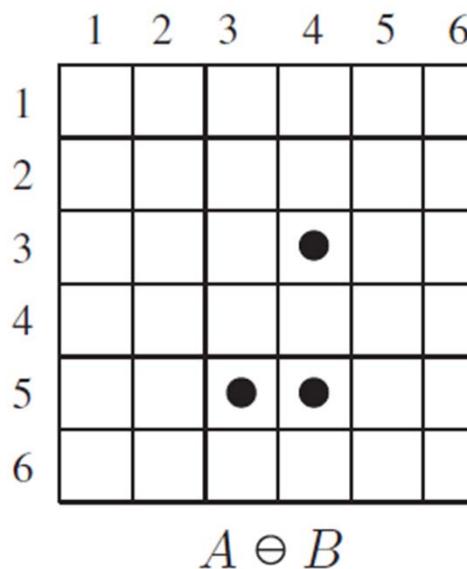


FIGURE 10.5 Dilation of a binary image. (a) Text image. (b) Result of dilation.

# Erosion

- Also known as Minkowski subtraction  $A \ominus B = \{w : B_w \subseteq A\}$



$$A - B = \bigcap_{b \in B} A_b$$

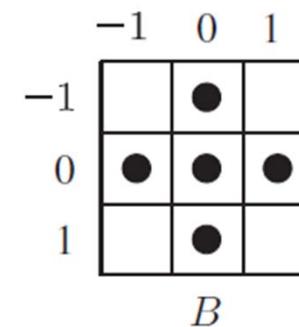


FIGURE 10.6 Erosion with a cross-shaped structuring element.

# Erosion in Matlab

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

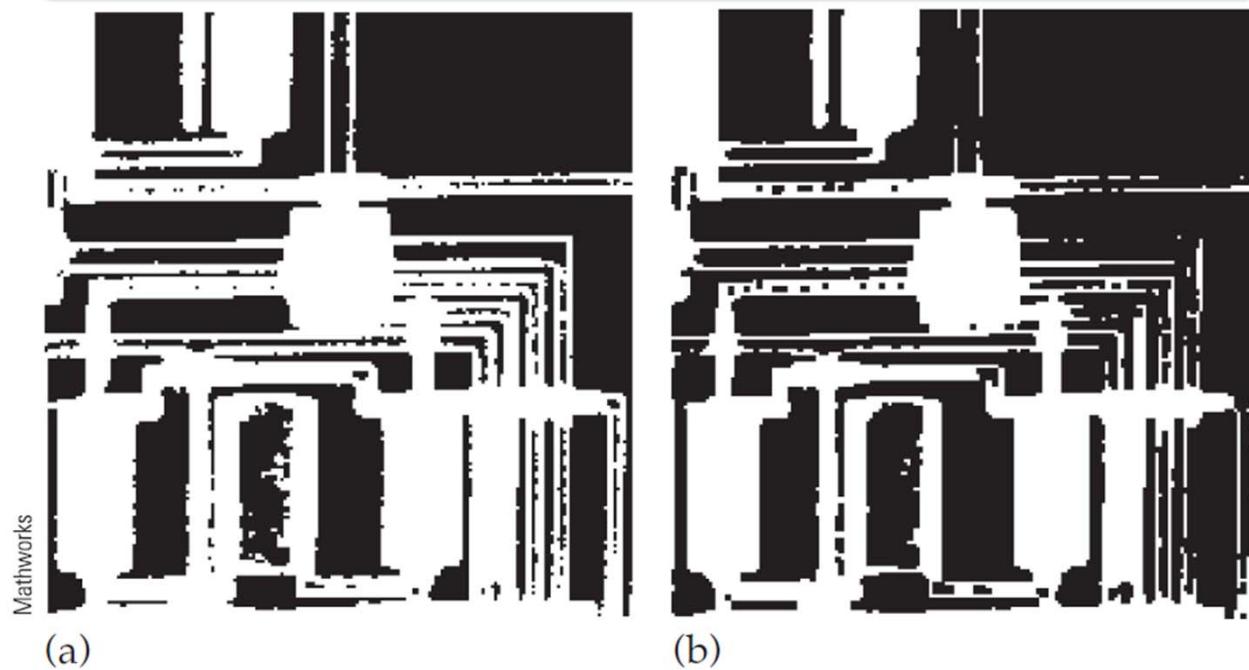
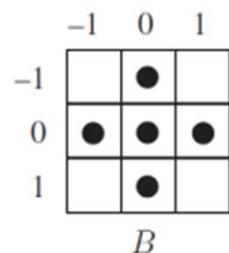
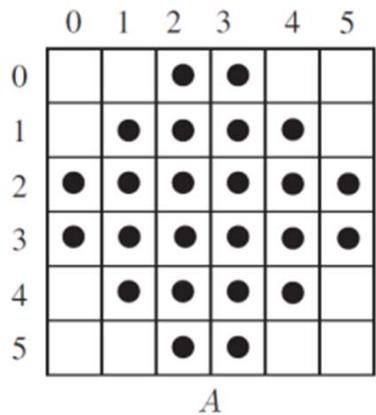


FIGURE 10.8 Erosion of a binary image. (a) Original image. (b) Result of erosion.

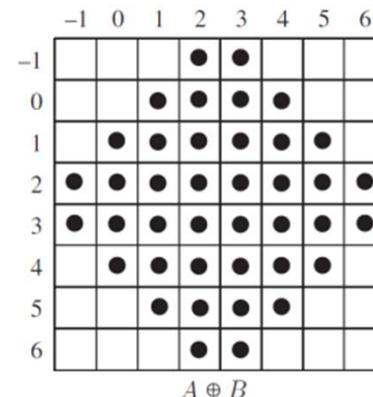
# Application of Dilation and Erosion: Boundary Detection

- If  $A$  is an image and  $B$  a small structuring element,

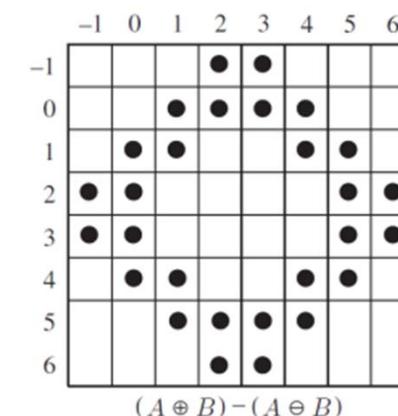
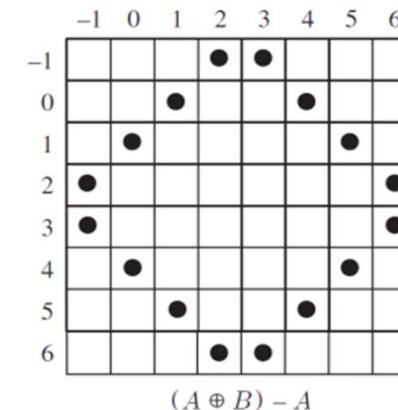
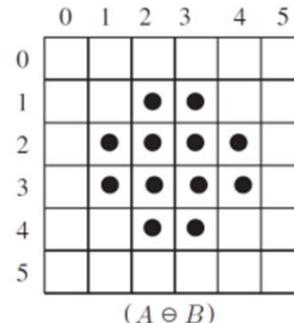
- |       |                                |                        |
|-------|--------------------------------|------------------------|
| (i)   | $A - (A \ominus B)$            | internal boundary      |
| (ii)  | $(A \oplus B) - A$             | external boundary      |
| (iii) | $(A \oplus B) - (A \ominus B)$ | morphological gradient |



external  
boundary  
detection:  
**outside of  
the object**



morphological  
gradient:  
**both int. and ext.  
boundaries**



# Internal Boundary Detection in Matlab

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

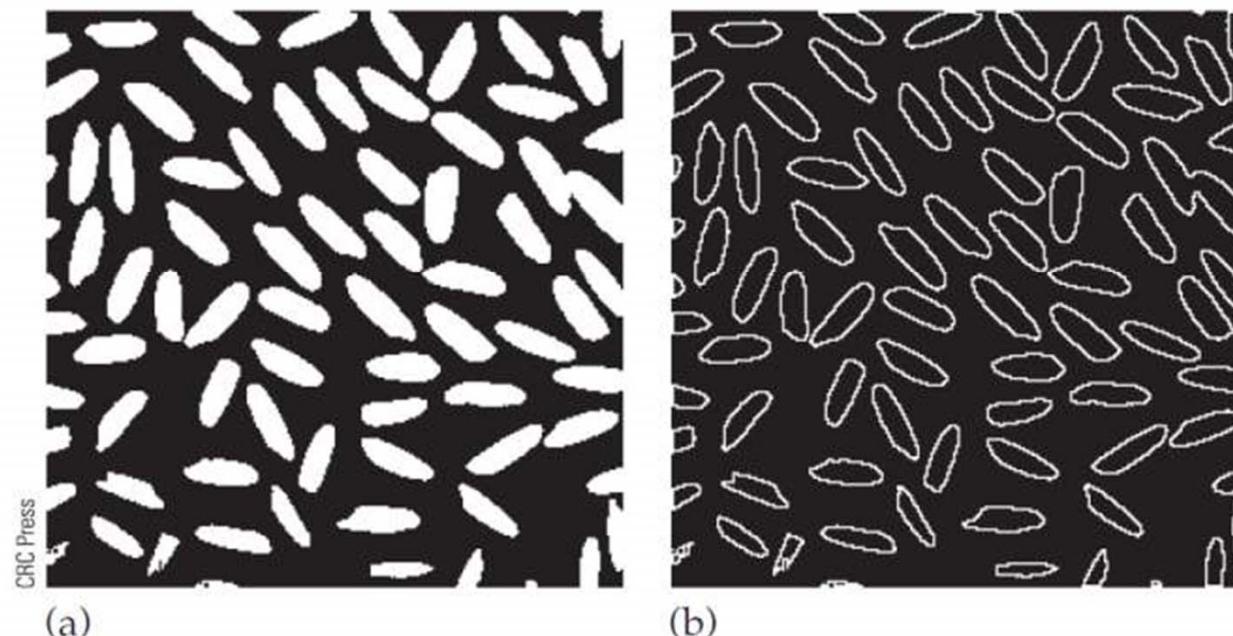
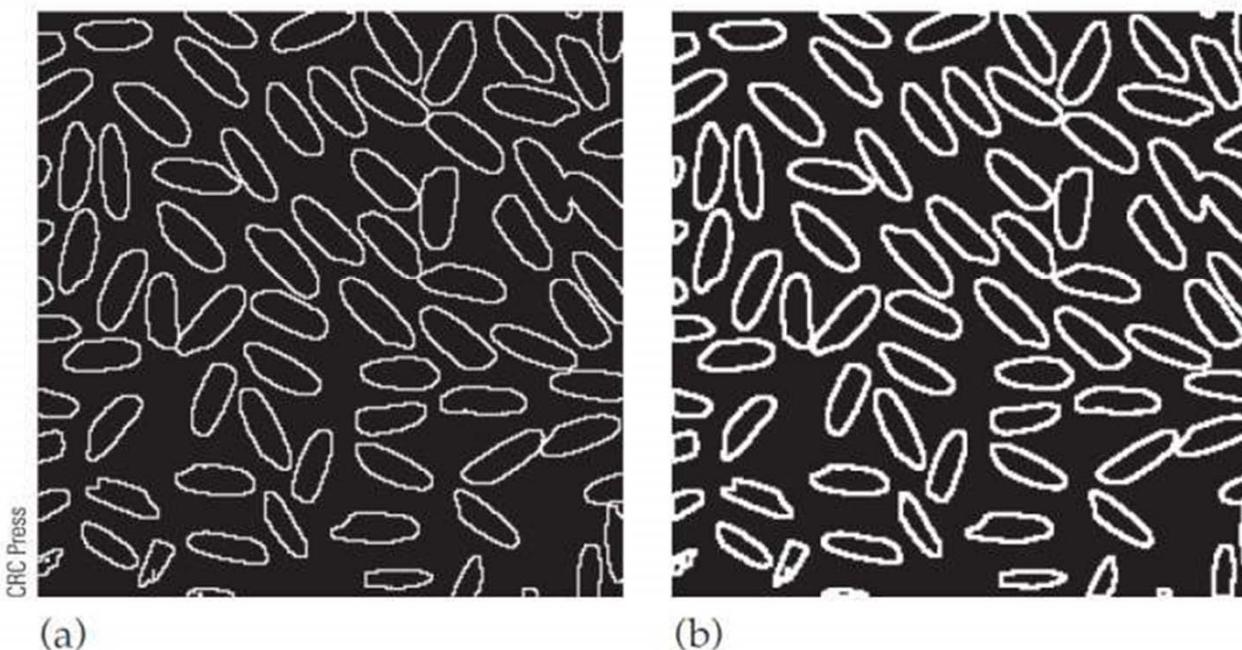


FIGURE 10.10 Morphological edge detection. (a) The rice grains image. (b) The internal boundary.

# External and Gradient Boundary Detection

```
>> 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)
```



**FIGURE 10.11** More morphological edge detection. (a) External boundary. (b) Morphological gradient.

# Opening

- **Opening** (function: `imopen( )`)

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

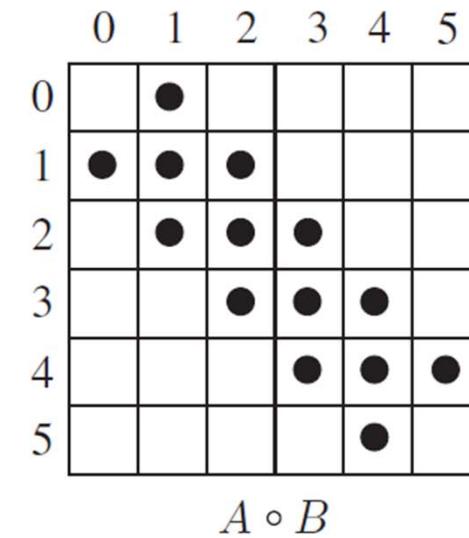
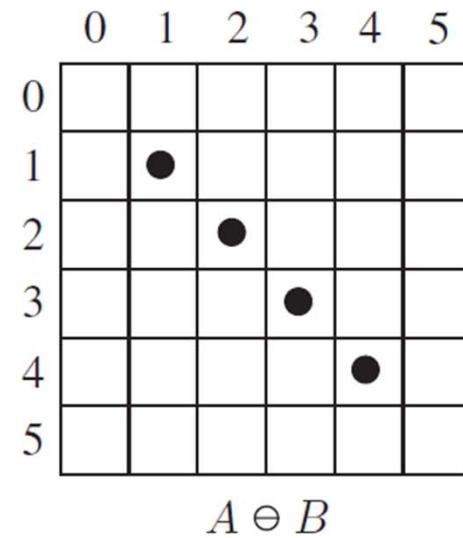
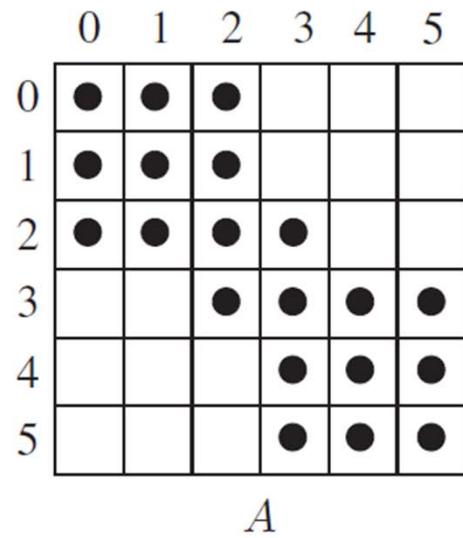


FIGURE 10.12 Opening.

# Properties of Opening

✓  $(A \circ B) \subseteq A$

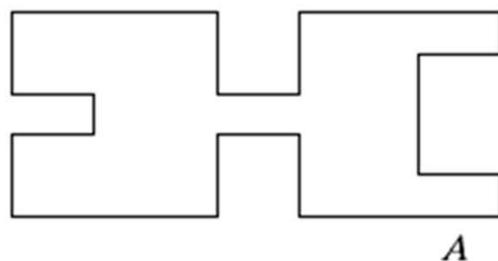
✓ Note that this is not the case with erosion. As we have seen, an erosion may not necessarily be a subset.

✓  $(A \circ B) \circ B = A \circ B$

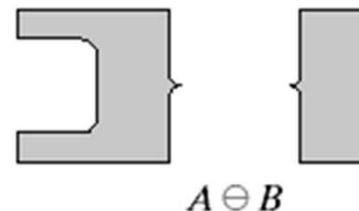
✓ That is, an opening can never be done more than once (**idempotent**).

✓ If  $A \subseteq C$ , then  $(A \circ B) \subseteq (C \circ B)$

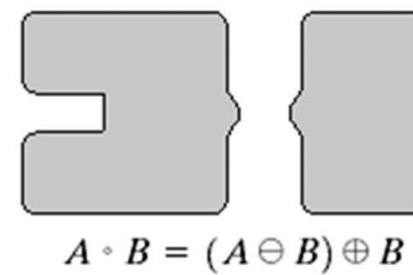
✓ Opening tends to **smooth an image**, to **break narrow joins**, and to **remove thin protrusions**.



Original shape  
 $A$



After erosion  
 $A \ominus B$



After dilation  
(opening)  
 $A \circ B = (A \ominus B) \oplus B$

# Opening Example

Original  
Image

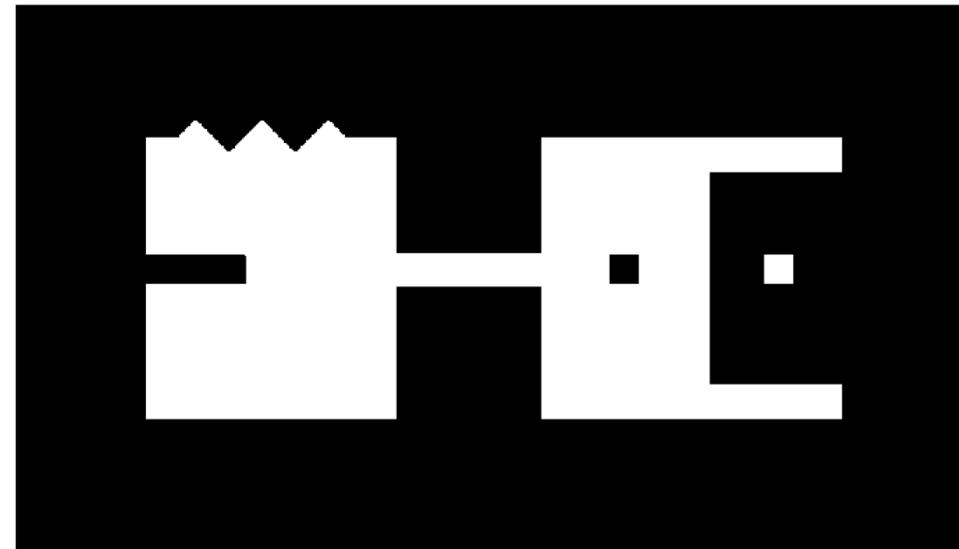


Image  
After  
Opening



# Closing

- **Closing** (function: `imclose( )`)

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

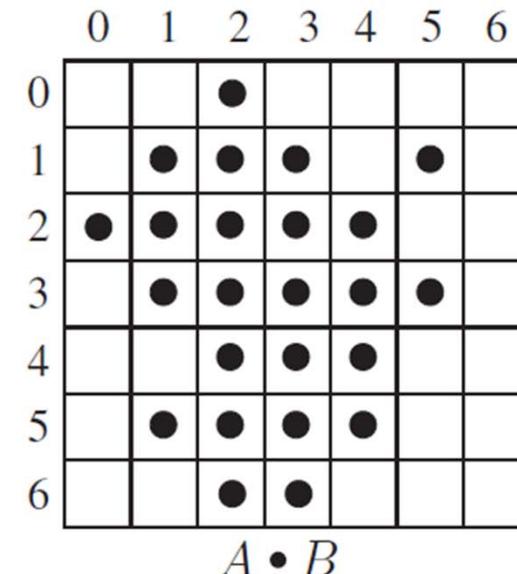
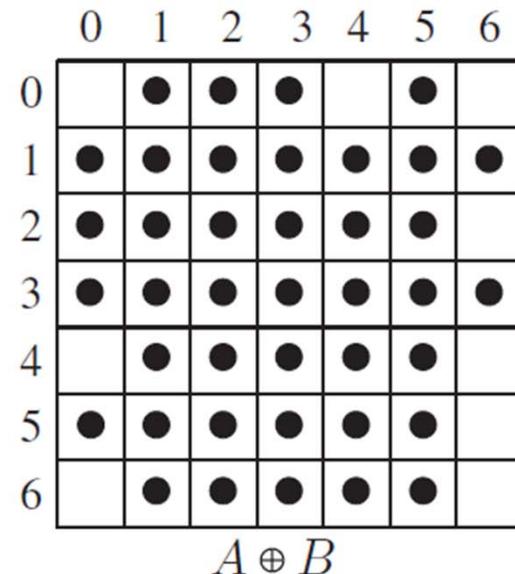
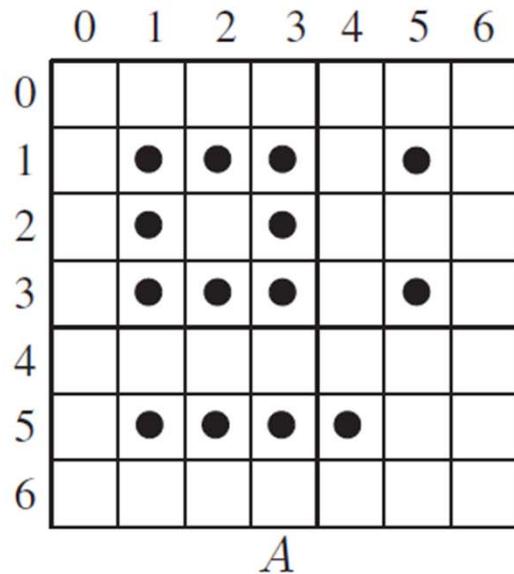
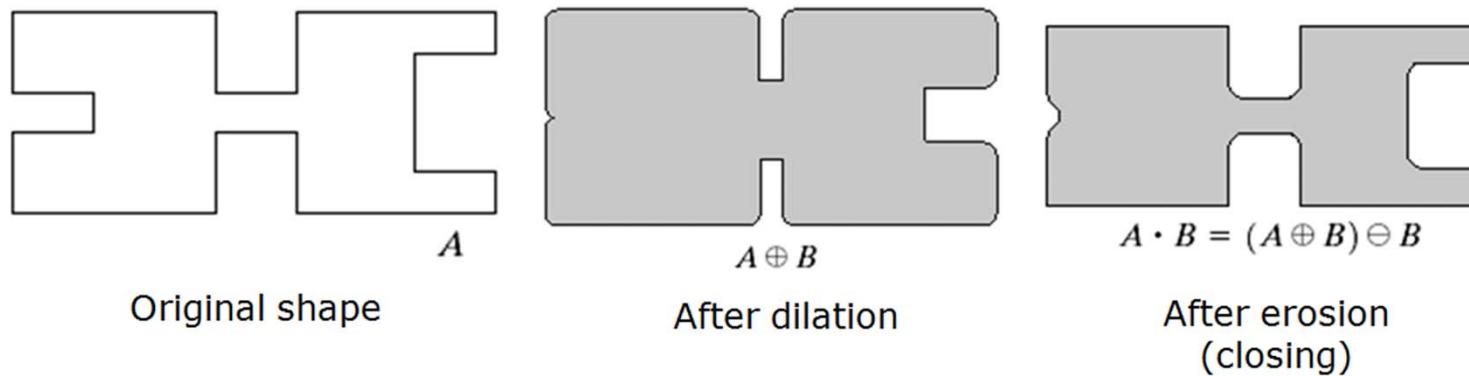


FIGURE 10.13 Closing.

# Properties of Closing

- ✓  $A \subseteq (A \bullet B)$
- ✓  $(A \bullet B) \bullet B = A \bullet B$ 
  - ✓ That is, closing, like opening, is **idempotent**.
- ✓ If  $A \subseteq C$ , then  $(A \bullet B) \subseteq (C \bullet B)$
- ✓ Closing also tends to **smooth an image**, but it fuses **narrow breaks** and **thin gulfs** and **eliminates small holes**.



# Closing Example

Original  
Image



se

Image  
After  
Closing

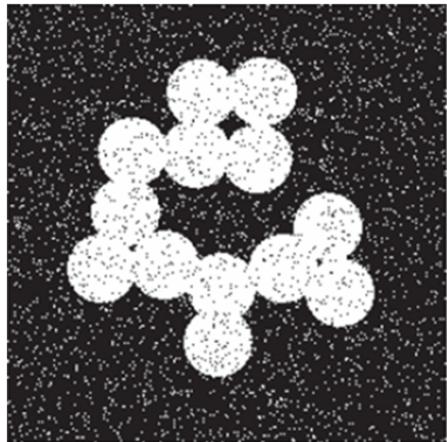


# Application of Opening and Closing: Noise Removal

- Morphological filtering

```
>> 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)
```

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



square SE

cross SE

# Hit-or-Miss Transform

- Very useful for **finding same shapes** in the image
- defined with dilation and erosion
- If  $B$  is the  $3 \times 3$  square structuring element,

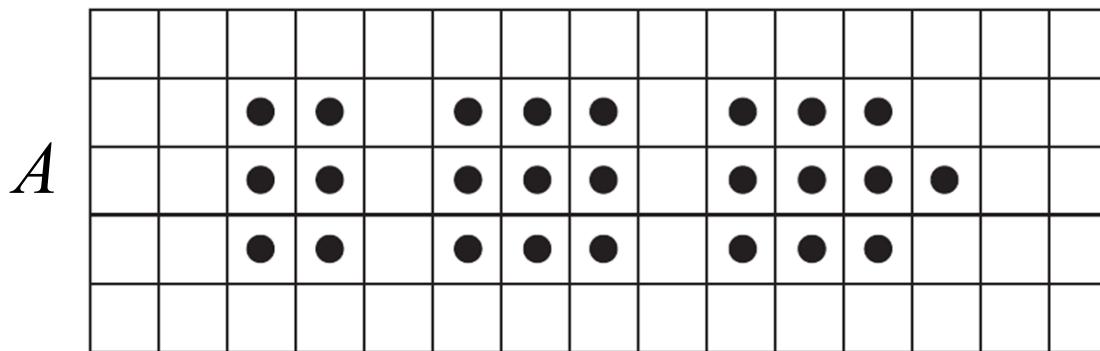
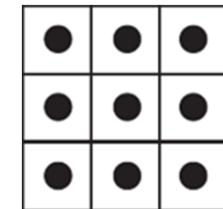
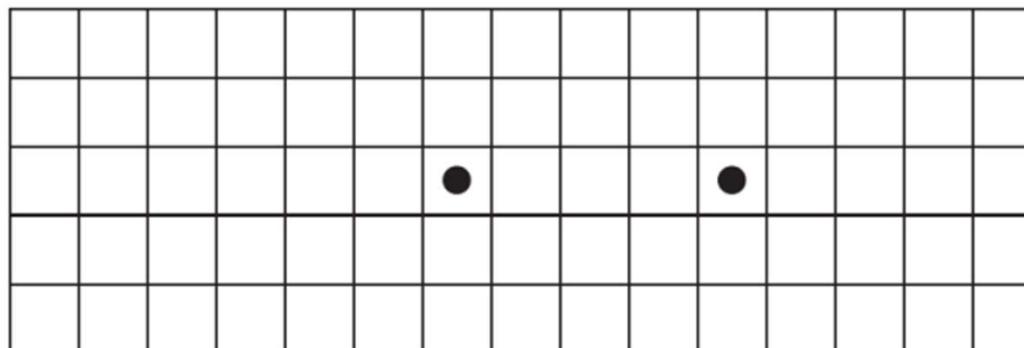


FIGURE 10.16 An image  $A$  containing a shape to be found.



$B$   
square SE

FIGURE 10.17 The erosion  $A \ominus B$ .

# Hit-or-Miss Transform

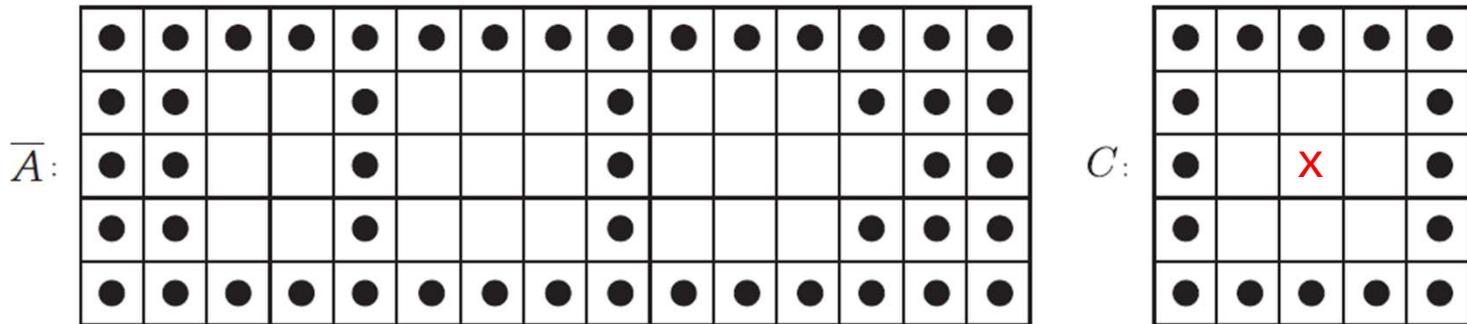


FIGURE 10.18 The complement and the second structuring element.

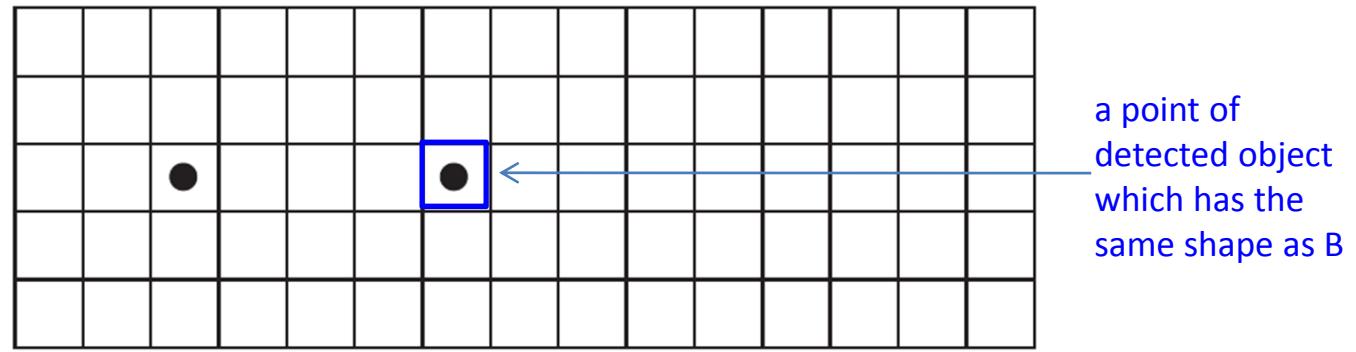


FIGURE 10.19 The erosion  $\bar{A} \ominus C$ .

$$A \circledast B = (A \ominus B_1) \cap (\bar{A} \ominus B_2)$$

In this example,  
 $B_1=B$ ,  $B_2=C$ .

# Hit-or-Miss Transform

- In general, if we are looking for a particular shape in an image, we can design two structuring elements, B1 and B2.
- B1 should be the same shape that we want to find.
- B2 should fit around the shape.

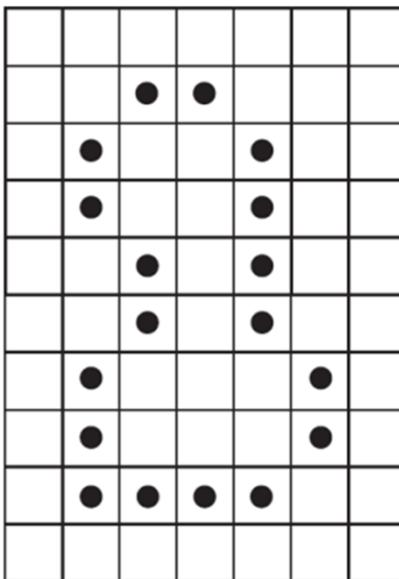
$$A \circledast B = (A \ominus B_1) \cap (\overline{A} \ominus B_2)$$

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

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

# Region Filling

- How to fill inside of the boundary ?



- Given a pixel  $p$  within the region, we wish to fill up the entire region.

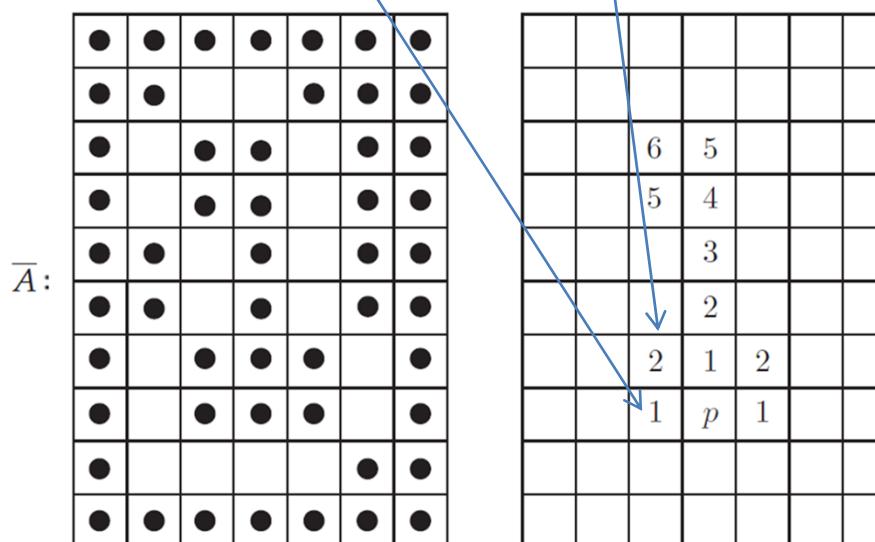
1. Dilate  $p$  with the **cross-shaped** structuring element  $B$ .
2. Take an intersection with complement of  $A$ .

$$\{p\} = X_0, X_1, X_2, \dots, X_k = X_{k+1},$$

$$X_n = (X_{n-1} \oplus B) \cap \bar{A}$$

3. Repeat 1 and 2 until the result image is converged.  $X_n = X_{n-1}$

$$X_0 = \{p\}, X_1 = \{p, 1\}, X_2 = \{p, 1, 2\}, \dots$$



# Region Filling in Matlab: **regfill( )**

```
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 10.24 A simple program for filling regions.

# Region Filling in Matlab

```
>> 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)
```



(a)



(b)



(c)



(d)

FIGURE 10.25 Region filling.

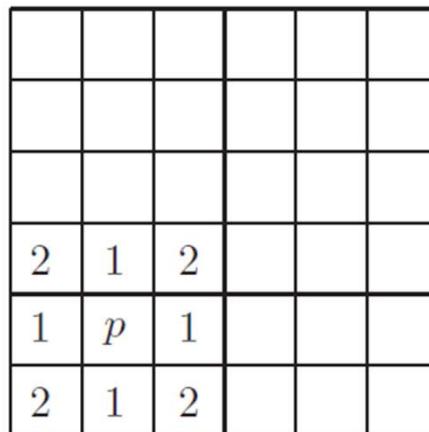
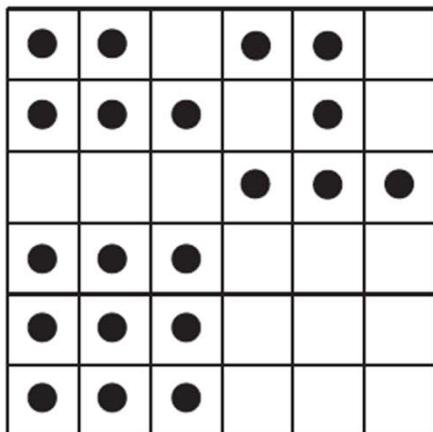
# Connected Components

- Cross shape SE for 4-connected components
- Square shape SE for 8-connected components

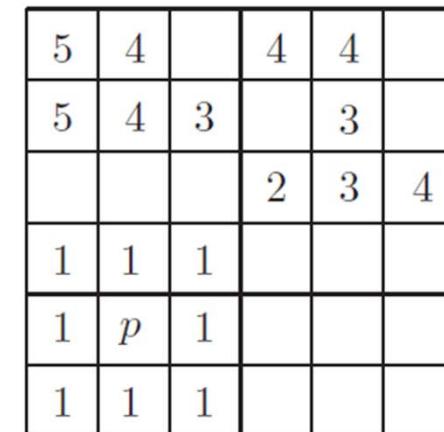
$$X_0 = p$$

It is not  $A^c$  !

$$X_n = (X_{n-1} \oplus B) \cap A, \quad n = 1, 2, 3, \dots \quad \text{Until} \quad X_n = X_{n-1}$$



Using the cross



Using the square

FIGURE 10.23 Filling connected components.

# Connected Components in Matlab: **components( )**

```
function out=components(im,pos,kernel)
% COMPONENTS(IM,POS,KERNEL) produces the connected component
% of binary image IMAGE which includes 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 10.26** A simple program for connected components.

# Connected Components in Matlab

```
>> sq2=ones(11,11);  
>> nc=components(n,[57,97],sq);  
>> imshow(nc)  
>> nc2=components(n,[57,97],sq2);  
>> figure,imshow(nc2)
```



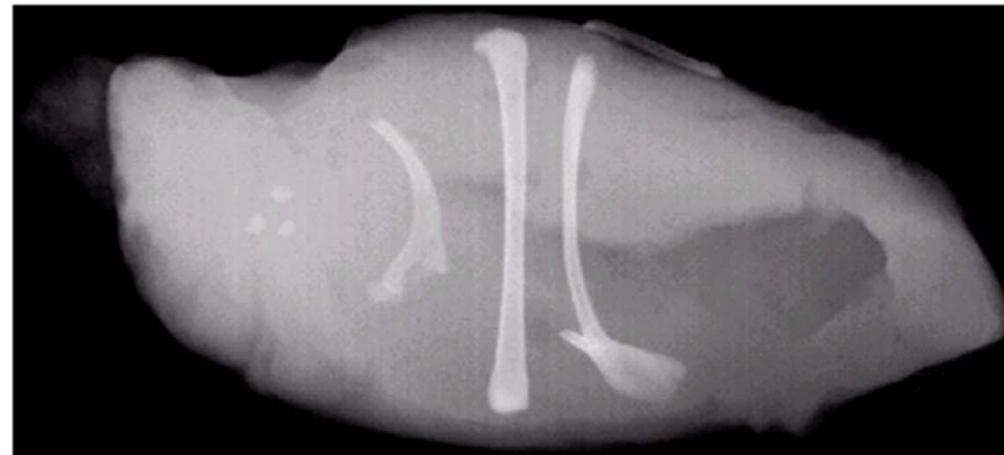
(a) 3x3 square SE

(b) 11x11 square SE

FIGURE 10.21 *Connected components.*

# Connected Components Example

original image



thresholding



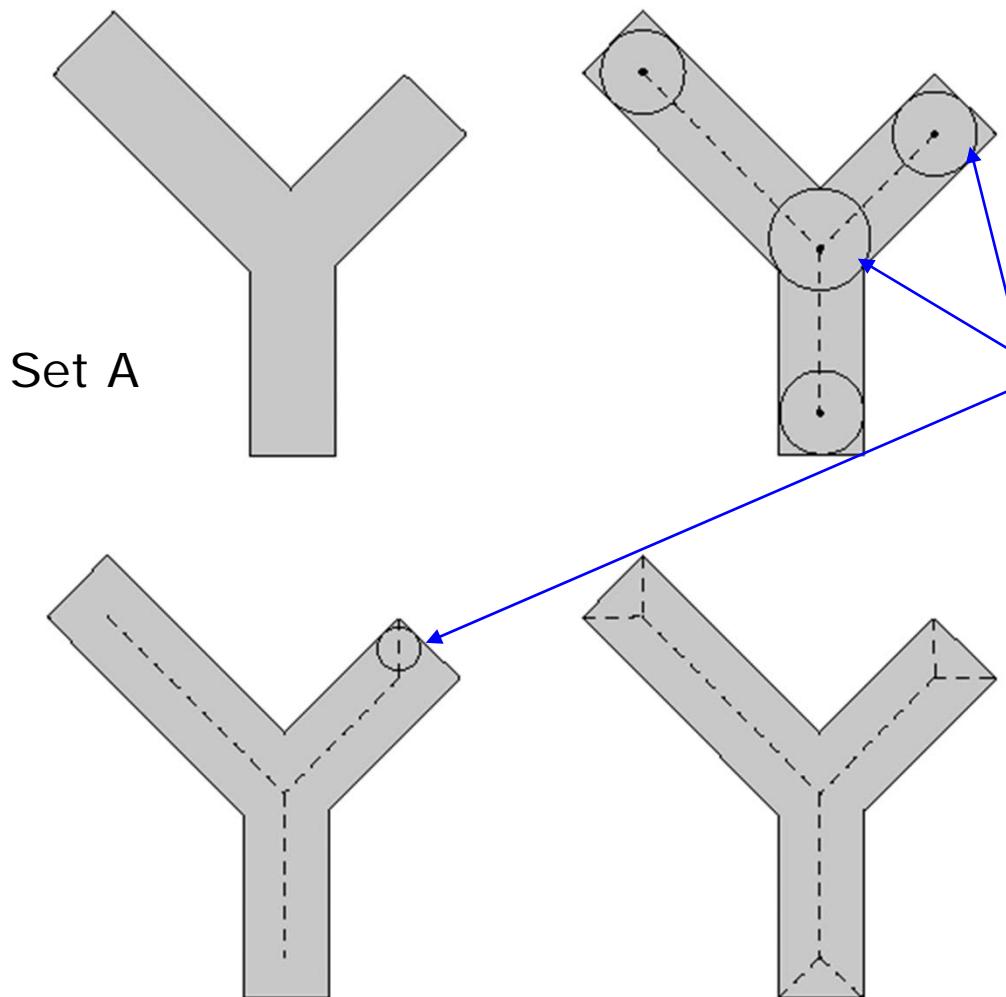
erosion  
with a  
5x5 SE



extraction of  
connected  
component

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

# Skeletons



How to define a  
Skeletons?

**Maximum disk**

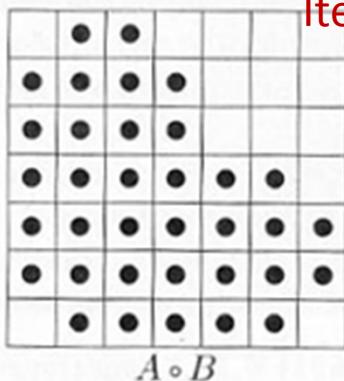
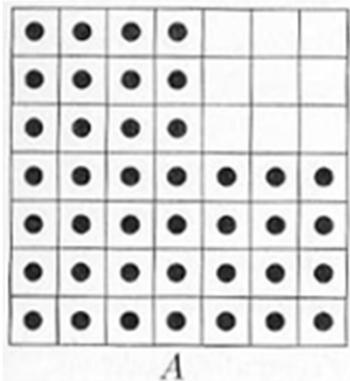
1. The largest disk centered at a pixel
2. Touch the boundary of A at two or more places

# Skeleton

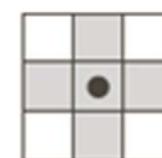
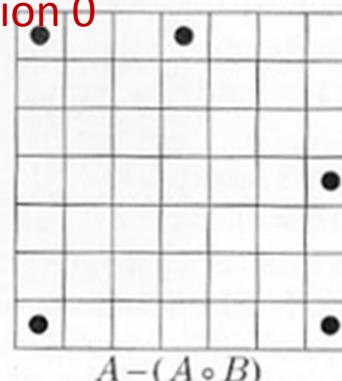
1. iterative erosions followed by an opening
  2. eroded image - opened image
- 

Erosions	Openings	Set differences
$A$	$A \circ B$	$A - (A \circ B)$
$A \ominus B$	$(A \ominus B) \circ B$	$(A \ominus B) - ((A \ominus B) \circ B)$
$A \ominus 2B$	$(A \ominus 2B) \circ B$	$(A \ominus 2B) - ((A \ominus 2B) \circ B)$
$A \ominus 3B$	$(A \ominus 3B) \circ B$	$(A \ominus 3B) - ((A \ominus 3B) \circ B)$
$\vdots$	$\vdots$	$\vdots$
$A \ominus kB$	$(A \ominus kB) \circ B$	$(A \ominus kB) - ((A \ominus kB) \circ B)$

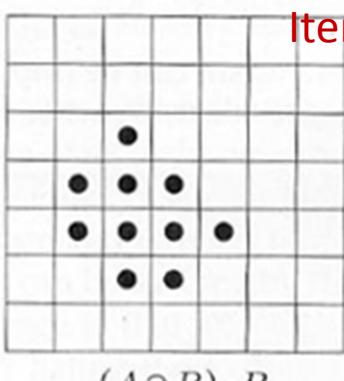
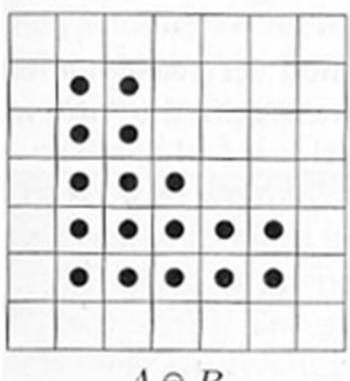
---



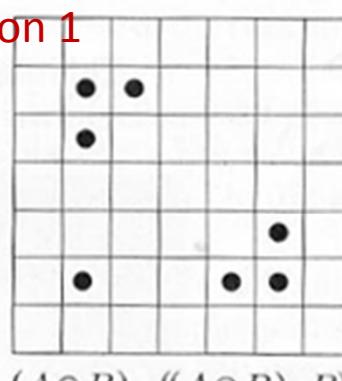
Iteration 0



$B$



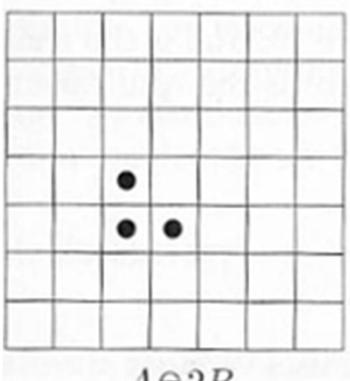
Iteration 1



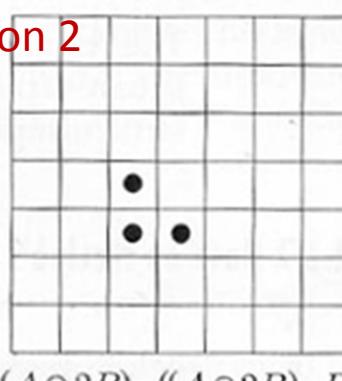
$(A \ominus B) - ((A \ominus B) \circ B)$

$$S_k(A) = (A \Theta kB) - (A \Theta kB) \circ B$$

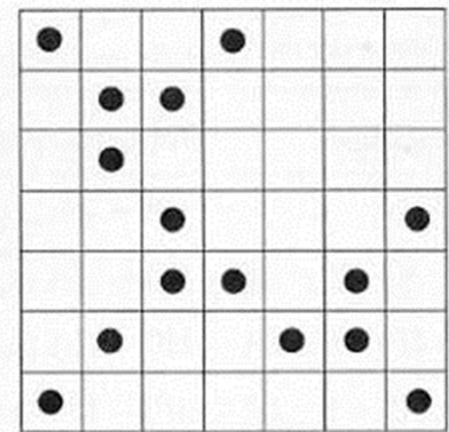
$$S(A) = \bigcup_{k=0}^K S_k(A)$$



Iteration 2



$(A \ominus 2B) - ((A \ominus 2B) \circ B)$



Final skeleton,  
 $S(A)$

# Skeleton in Matlab

```
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 10.30 A simple program for computing skeletons.

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

# FIGURE 10.31

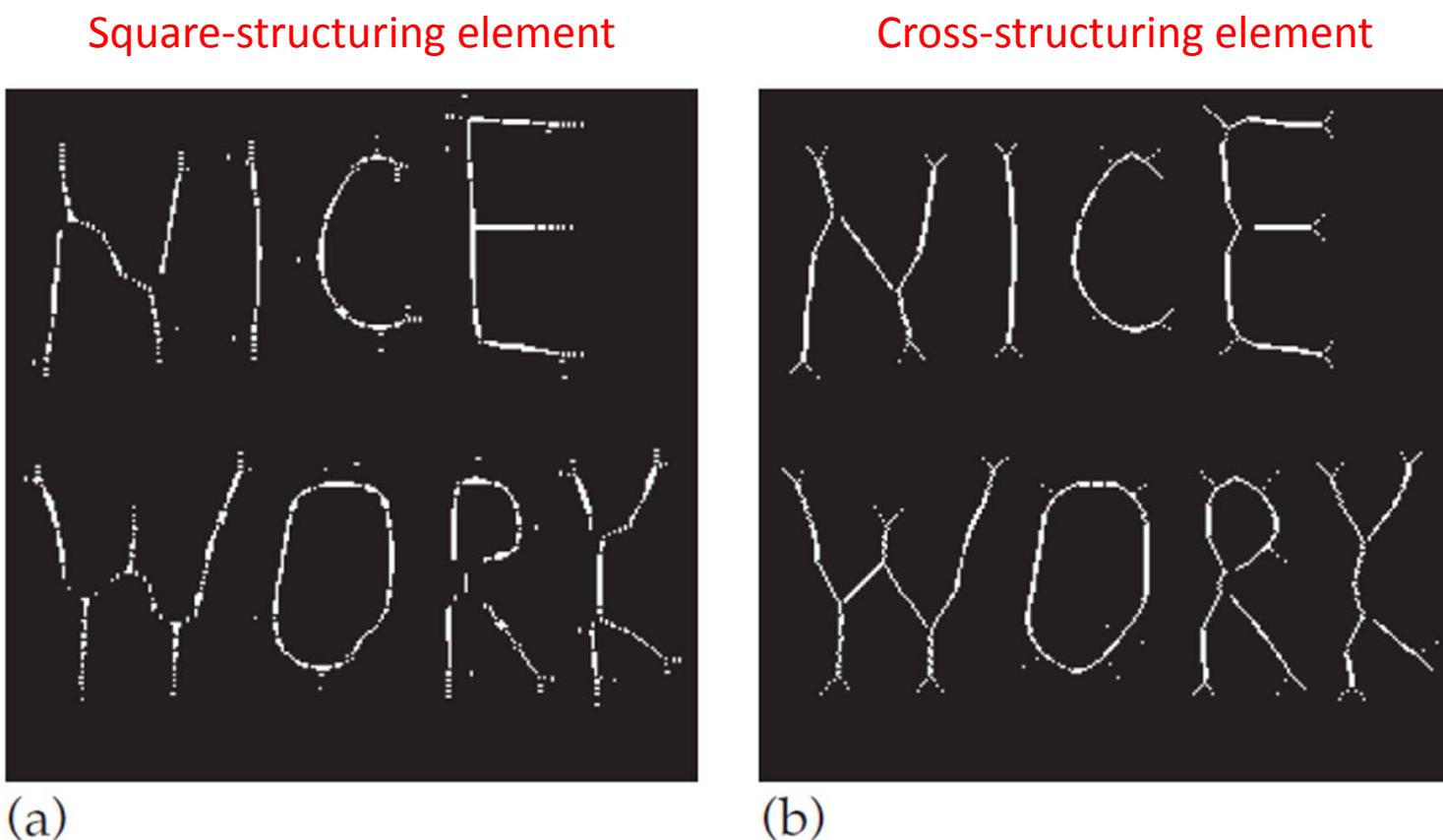


FIGURE 10.31 *Skeletonization of a binary image.*

# A Note on MATLAB's bwmorph Function

- Based on **lookup tables** (ch11)
  - ✓ Consider the  $3 \times 3$  neighborhood of a pixel.
  - ✓ Since each pixel in the neighborhood can have only two values, there are  $2^9 = 512$  different possible neighborhoods.
  - ✓ Define a morphological operation to be a function that maps these neighborhoods to the values 0 and 1.
  - ✓ Each possible neighborhood state can be associated with a numeric value from 0 (all pixels have value 0) to 511 (all pixels have value 1).
  - ✓ The lookup table is then a binary vector of length 512. Its  $k$ th element is the value of the function for state  $k$ .
- Many other operations can be defined by this method (see the help file for `bwmorph`)

# Grayscale Morphology

- Another way for Erosion
  - ✓ 1. Find the  $3 \times 3$  neighborhood  $N_p$  of  $p$
  - ✓ 2. Compute the matrix  $\{N_p - B\}$
  - ✓ 3. Find the minimum value in the matrix  $\{N_p - B\}$

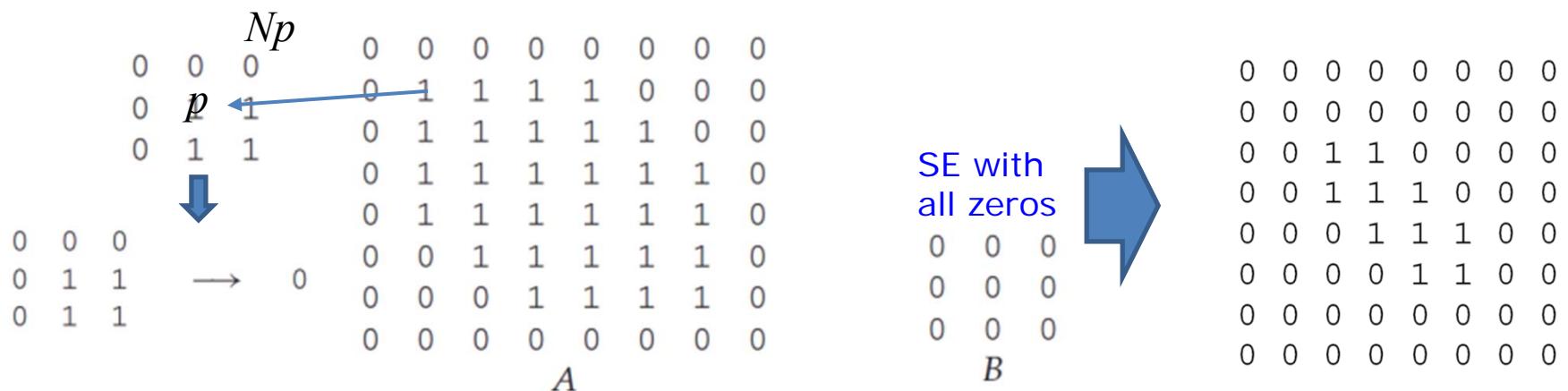


FIGURE 10.32 An example for erosion.

# Grayscale Morphology

- **Another way for dilation**
  - ✓ 1. Find the  $3 \times 3$  neighborhood  $N_p$  of  $p$
  - ✓ 2. Compute the matrix  $N_p + B$
  - ✓ 3. Find the maximum of that result
- **Summary**

$$(A \ominus B)(x, y) = \min\{A(x + s, y + t) - B(s, t), (s, t) \in D_B\},$$

$$(A \oplus B)(x, y) = \max\{A(x + s, y + t) + B(s, t), (s, t) \in D_B\}.$$

# Grayscale Morphology Example

	$y$					$t$			
	1	2	3	4	5	-1	0	1	
$x$	1	10	20	20	20	30	-1	1	2
	2	20	30	30	40	50	0	4	5
	3	20	30	30	50	60	1	7	8
	4	20	40	50	50	60			9
	5	30	50	60	60	70			

$A$                              $B$

FIGURE 10.33 An example for grayscale erosion and dilation.

$$A \ominus B = \begin{matrix} 5 & 6 & 14 & 15 & 16 \\ 8 & 9 & 17 & 18 & 19 \\ 12 & 13 & 25 & 26 & 39 \\ 15 & 16 & 28 & 29 & 46 \\ 18 & 19 & 39 & 48 & 49 \end{matrix} \quad A \oplus B = \begin{matrix} 39 & 39 & 49 & 59 & 58 \\ 39 & 39 & 59 & 69 & 68 \\ 49 & 59 & 59 & 69 & 68 \\ 59 & 69 & 69 & 79 & 78 \\ 56 & 66 & 66 & 76 & 75 \end{matrix}$$

# Grayscale Morphology in Matlab

- The arbitrary parameter of strel allows us to create a structuring element containing any values we like.
- ones(3,3) provides the neighborhood.
- Matrix as third parameter provides the values of 3x3 SE.

```
>> str=strel('arbitrary',ones(3,3),[1 2 3;4 5 6;7 8 9])
```

```
str =
```

```
Nonflat STREL object containing 9 neighbors.
```

```
Neighborhood:
```

```
1 1 1  
1 1 1  
1 1 1
```

```
Height:
```

```
1 2 3  
4 5 6  
7 8 9
```

```
>> A=[10 20 20 20 30;20 30 30 40 50;20 30 30 50 60;20 40 50 50 60;30 50 60 60 70];  
>> imerode(A,str)
```

```
ans =
```

```
5 6 14 15 16  
8 9 17 18 19  
12 13 25 26 39  
15 16 28 29 46  
18 19 39 48 49
```

# Grayscale Morphology in Matlab

```
>> str2=strel('arbitrary',ones(3,3),[9 8 7;6 5 4;3 2 1])
str2 =
Nonflat STREL object containing 9 neighbors.

Neighborhood:
 1     1     1
 1     1     1
 1     1     1

Height:
 9     8     7
 6     5     4
 3     2     1

>> imdilate(A,str2)

ans =

 39    39    49    59    58
 39    39    59    69    68
 49    59    59    69    68
 59    69    69    79    78
 56    66    66    76    75
```

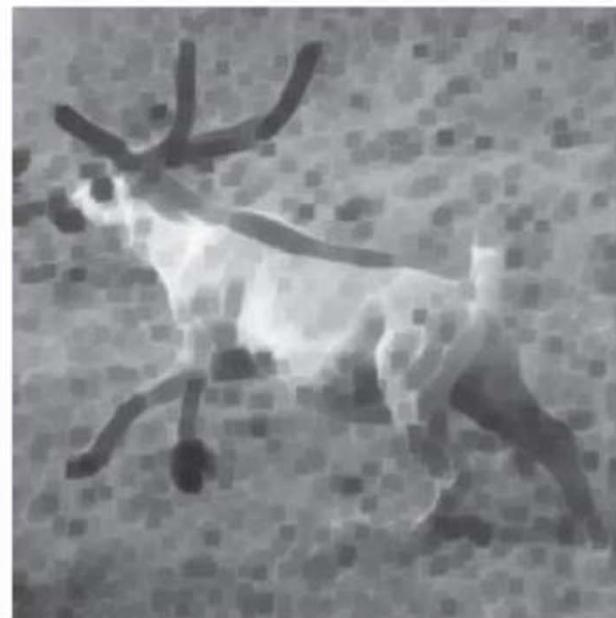
# Grayscale Morphology in Matlab: Erosion and Dilation

```
>> c=imread('caribou.tif');
>> str=strel('square',5)
>> cd=imdilate(c,str);
>> ce=imerode(c,str);
>> imshow(cd), figure, imshow(ce)
```

Capt. Budd Christman/NOAA Corps.



(a)



(b)

FIGURE 10.34 Morphology. (a) Dilation. (b) Erosion.

# Grayscale Morphology: Opening and Closing

```
>> co=imopen(c,str);
>> cc=imclose(c,str);
>> imshow(co),figure,imshow(cc)
```

Capt. Budd Christman/NOAA Corps.



(a)



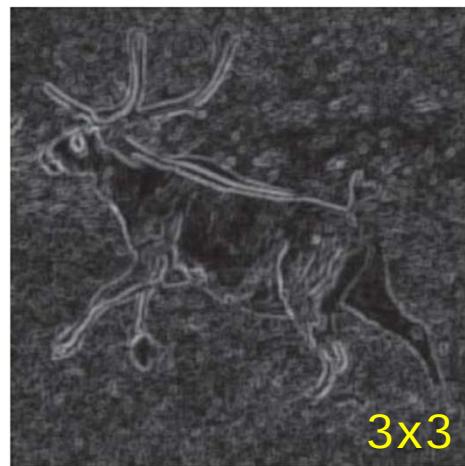
(b)

FIGURE 10.35 Grayscale opening and closing. (a) Opening. (b) Closing.

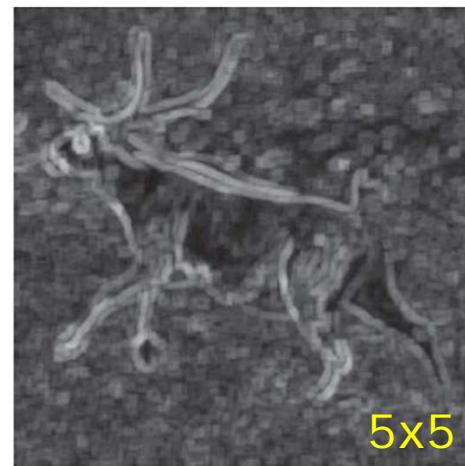
# Applications of Grayscale Morphology: Edge Detection

- morphological gradient in grayscale  $(A \oplus B) - (A \ominus B)$
- 3x3 and 5x5 square SE

```
>> str1=strel('square',3);
>> str2=strel('square',5);
>> ce1=imerode(c,str1);
>> ce2=imerode(c,str2);
>> cd1=imdilate(c,str1);
>> cd2=imdilate(c,str2);
>> cg1=imsubtract(cd1,ce1);
>> cg2=imsubtract(cd2,ce2);
>> imshow(cg1),figure,imshow(cg2)
```



3x3



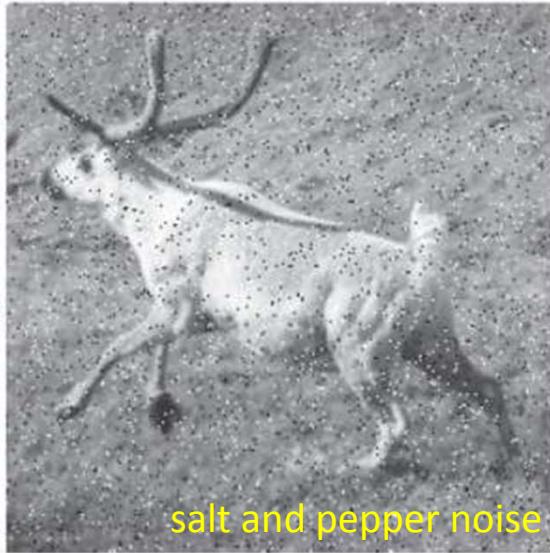
5x5

# Applications of Grayscale Morphology: Noise Removal

- opening followed by a closing in grayscale
- In general, it does not perform well on Gaussian noise

```
>> cn=imnoise(c,'salt & pepper');
>> cf=imclose(imopen(cn,str),str);
>> imshow(cn),figure,imshow(cf)
```

Capt. Budd Christman/NOAA Corps.



salt and pepper noise

(a)



grayscale opening and closing

(b)

FIGURE 10.37 Use of morphological filtering to remove noise.

# Summary

- **Chap 9. Image Segmentation**
  - Single & double thresholding
  - How to determine the threshold value
  - Adaptive thresholding
  - Edge detection: 1st and 2nd derivatives
  - Canny edge detector
  - Hough Transform
- **Chap 10. Mathematical Morphology**
  - Erosion & dilation -> boundary detection
  - Opening & closing -> noise removal
  - Hit-or-miss transform -> shape detection
  - Binary applications: region filling, connected components, skeletons
  - Grayscale morphology -> edge detection, noise removal