



# Edge Detection

Dr. Mongkol Ekpanyapong



# Roadmap

- Introduction to image analysis (computer vision)
  - Its connection with psychology and neuroscience
  - Why is image analysis difficult?
- Theory of edge detection
  - Gradient operator
  - Advanced operators
- Applications
  - Road/sign detection in intelligent driving systems
  - Pupil detection in iris recognition systems

A decorative graphic in the top-left corner consisting of a blue square above a grid of smaller squares in various colors.

# Computer Vision: the Grand Challenge

- Teach a computer to see is nontrivial at all
- Unlike binary images, grayscale/color images acquired by the sensor are often easy to understand by human being but difficult for a machine or a robot
- There are lots of interesting problems in the field of computer vision (image analysis)
  - Image segmentation, image understanding, face detection/recognition, object tracking ...

# How does Human Vision System work?

## Top-down school

I see a human body



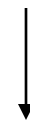
I expect to see a human face



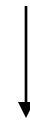
I expect to see two eyes and a nose

## Bottom-up school

pixels



components (such as edges, lines etc.)



objects

Two hypothesis and nobody knows the answer yet!

## An Amazing Image Example

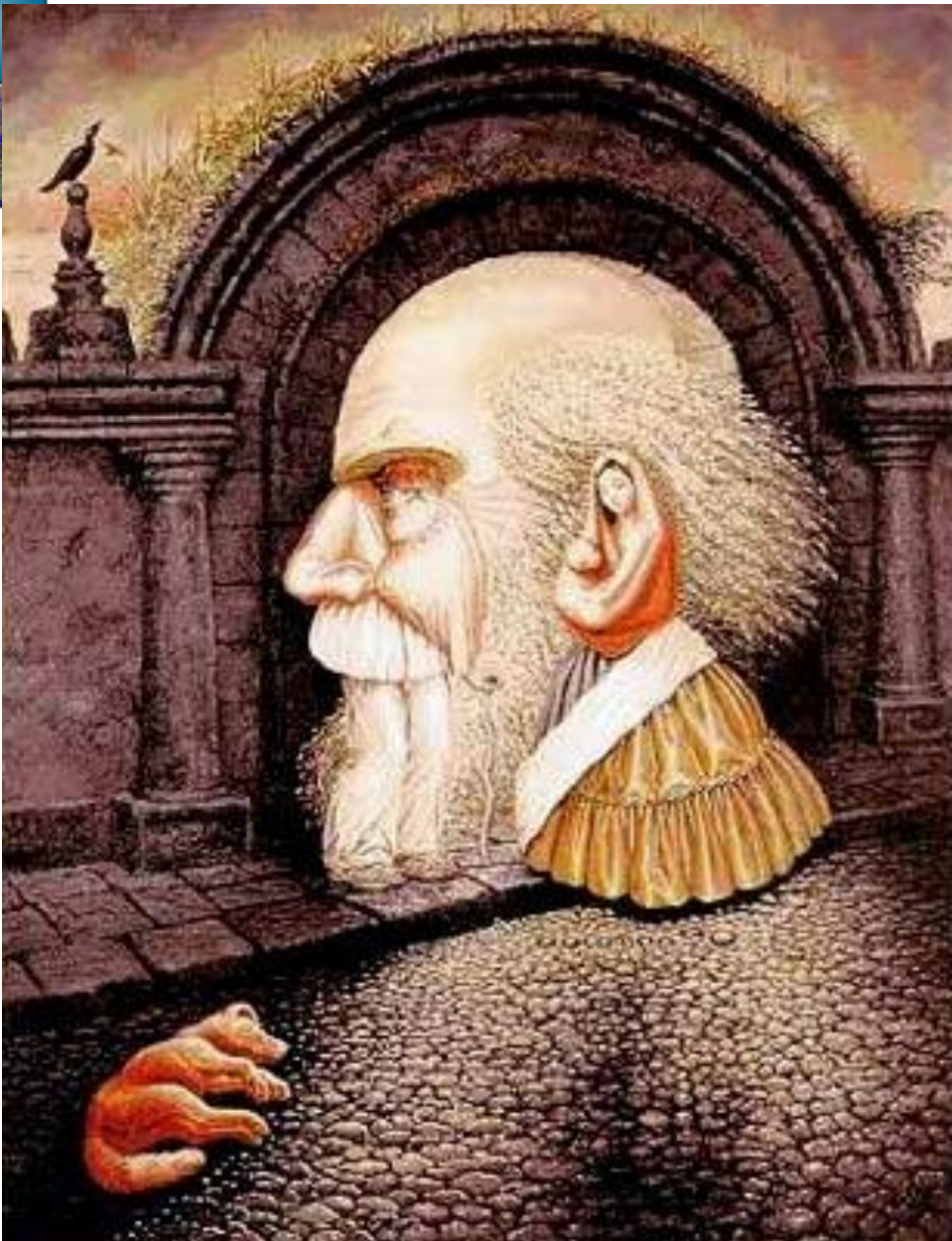
Person A:

I see an old man with a fancy earring and a strange hand

Person B:

I see two people on the street and a dog lying beside

If you try really hard, you will be able to locate at least eight different faces from this image

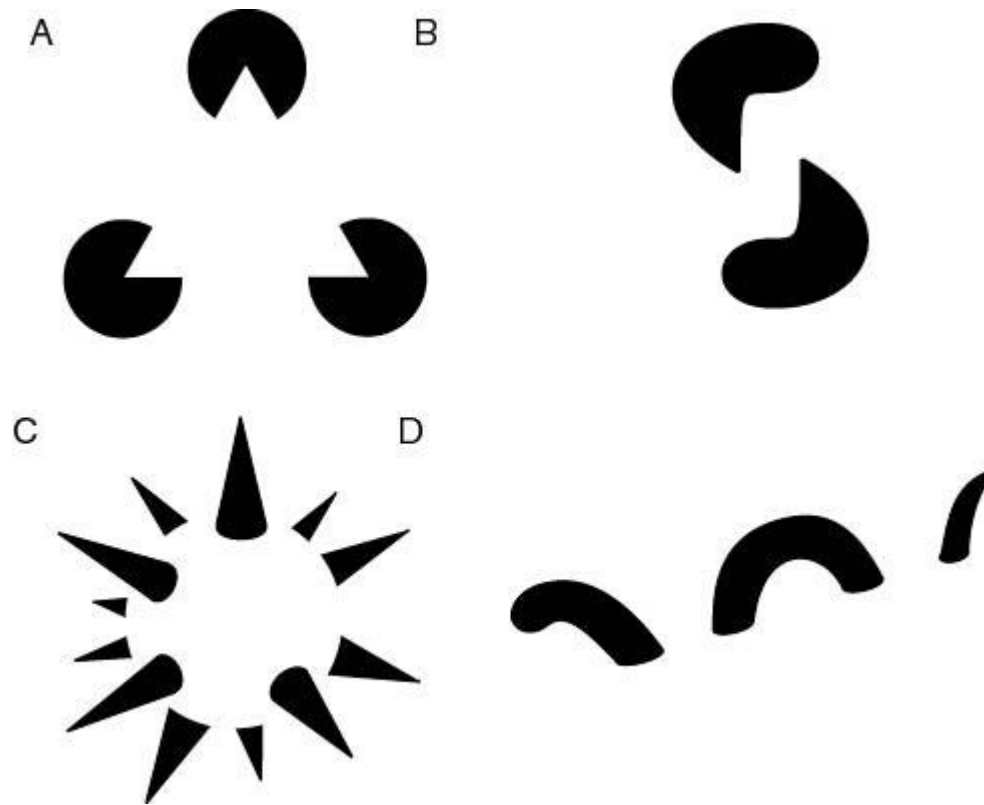


# Gestalt Theory (the Berlin School)

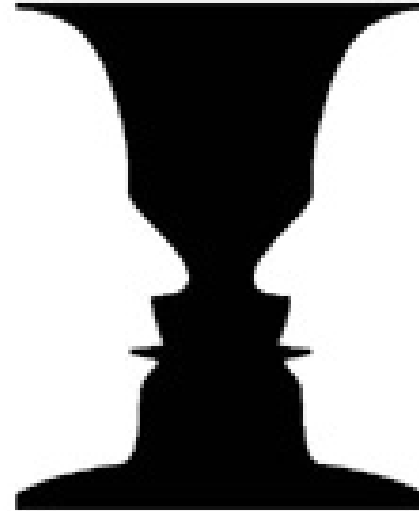
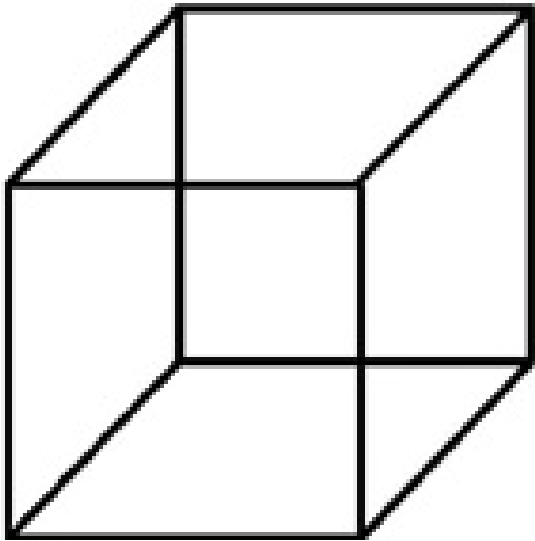


Emergence: the dog is perceived as a whole, all at once

# Reification

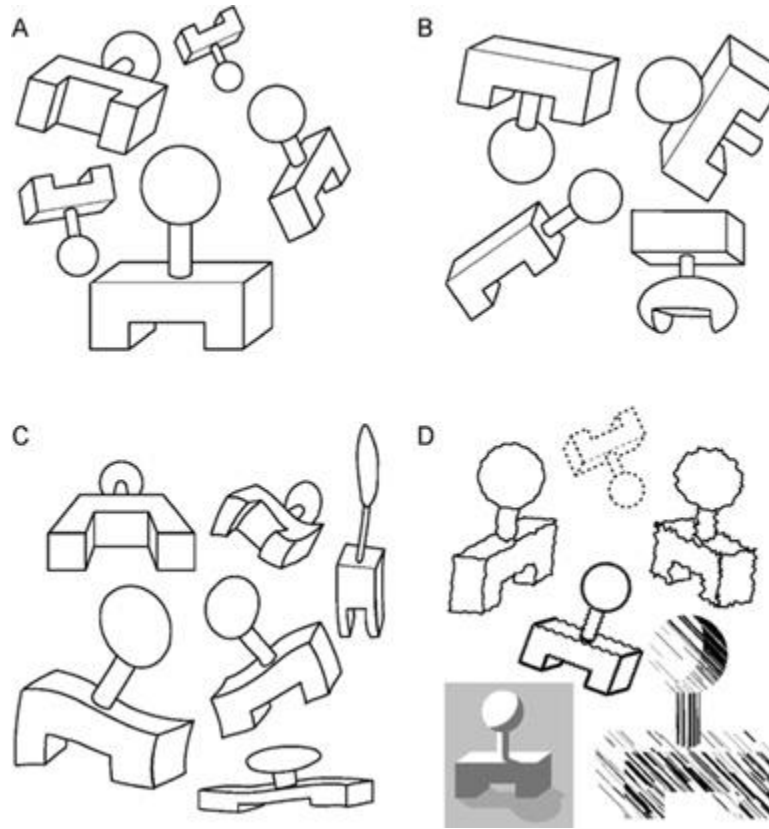


# Multistability (or Multistable Perception)





# Invariance



# Application: Face Detection



You are strongly encouraged to try the interactive demo out yourself

<http://vasc.ri.cmu.edu/demos/faceindex/>

# Image Segmentation

- Segmentation subdivides an image into its constituent regions or objects
- Some examples are point, line, and edge detection



# Point Detection

- Using the mask below, we say that an isolated point has been detected at the location on which the mark is centered if  $|R| \geq T$  when  $T$  is non negative threshold

-1	-1	-1
-1	8	-1
-1	-1	-1

**FIGURE 11.1**  
A mask for point  
detection.

---

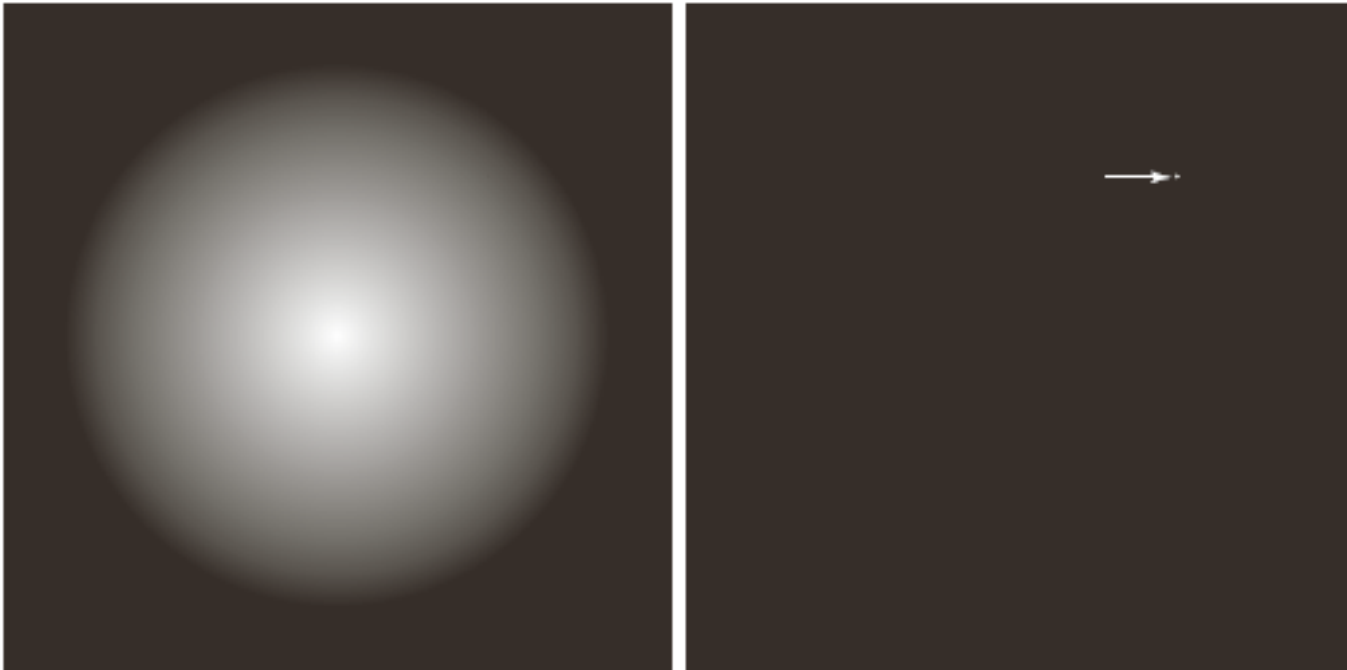


# Matlab Function

```
f=imread('sphere.tif');  
w = [ -1 -1 -1 ; -1 8 -1; -1 -1 -1];  
g = abs(imfilter(double(f),w)) ;  
g = g >= 15;  
imshow(f);  
figure, imshow(g)
```



# Result



a b

## FIGURE 11.2

(a) Gray-scale image with a nearly invisible isolated black point in the north-east quadrant of the sphere.

(b) Image showing the detected point. (The point was enlarged to make it easier to see.)



# Edge Detection

- Why detect edge?

Edges characterize object boundaries and are useful features for segmentation, registration and object identification in scenes.

- What is edge (to human vision system)?

**No rigorous definition exists**

Intuitively, edge corresponds to **singularities** in the image (i.e. where pixel value experiences abrupt change)

# Line Detection

- Let  $R_1$ ,  $R_2$ ,  $R_3$ , and  $R_4$  denote the responses of the masks below
- Suppose that the four masks are run individually through an image, if  $|R_i| > |R_j|$  then the point is more likely associated with a line in the direction  $i$

a b c d  
**FIGURE 11.3**  
 Line detector masks.

a	b	c	d
-1	-1	-1	2
2	2	2	-1
-1	-1	-1	2
2	-1	-1	-1
-1	2	-1	-1
-1	2	-1	2
-1	2	-1	-1
2	-1	-1	-1

Horizontal

+45°

Vertical

-45°



# Example

a	b
c	d
e	f

**FIGURE 11.4**

(a) Image of a wire-bond template.

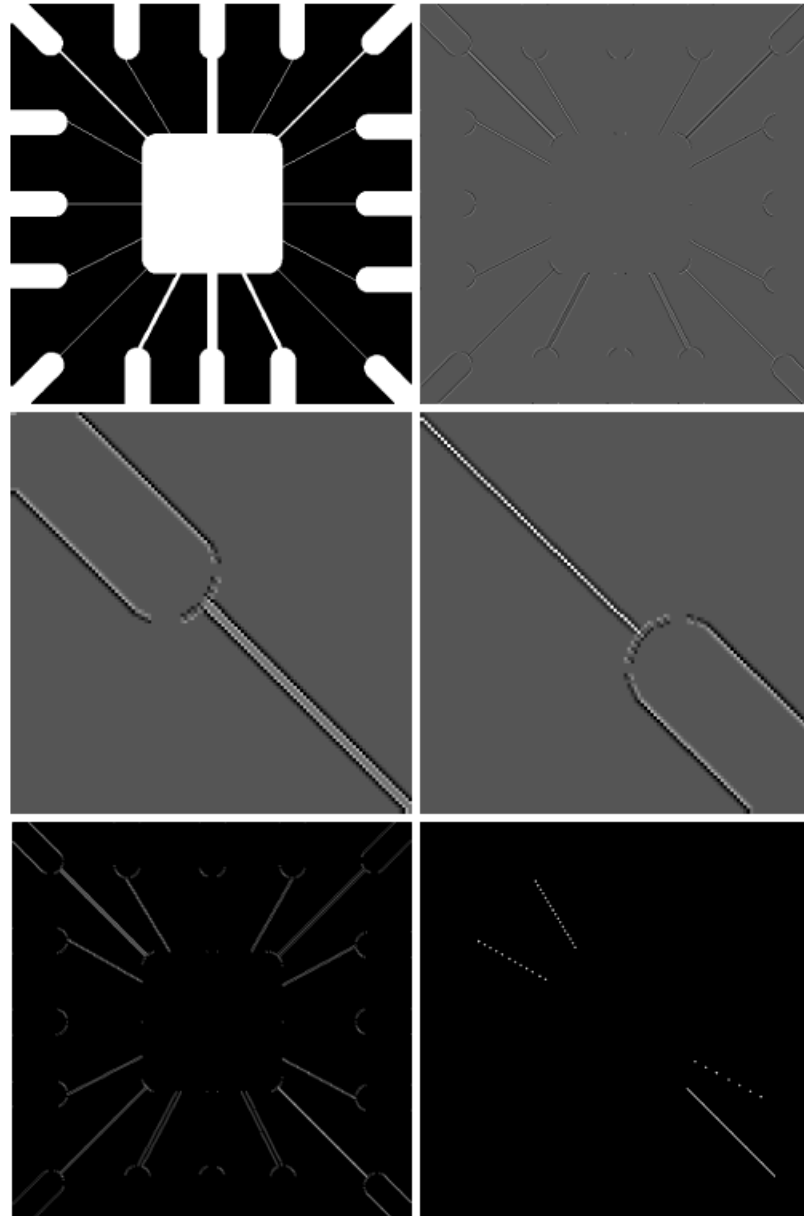
(b) Result of processing with the  $+45^\circ$  detector in Fig. 11.3.

(c) Zoomed view of the top, left region of (b).

(d) Zoomed view of the bottom, right section of (b).

(e) Absolute value of (b).

(f) All points (in white) whose values satisfied the condition  $g \geq T$ , where  $g$  is the image in (e). (The points in (f) were enlarged to make them easier to see.)



# Edge detection is a hard image processing problem

- Most edge detection solutions exhibit limited performance in the presence of images containing real-world scenes.
- It is common to precede the edge detection stage with pre-processing operations such as noise reduction and illumination correction



# Edge Detection

- Edge model
  - Step edge
  - Ramp edge
  - Roof edge



a b c

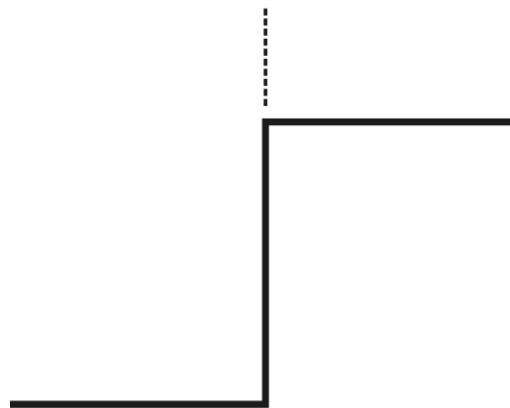
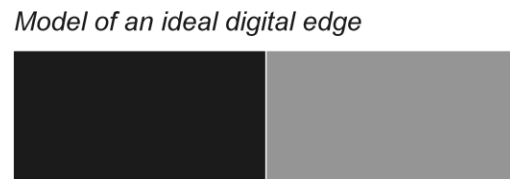
**FIGURE 10.8**

From left to right, models (ideal representations) of a step, a ramp, and a roof edge, and their corresponding intensity profiles.

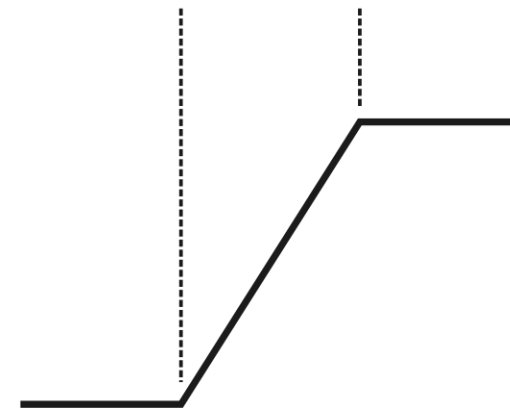
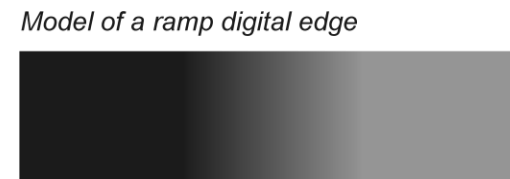


# Basic concepts

- Edge: a boundary between two image regions having distinct characteristics according to some feature (e.g., gray level, color, or texture).
- In grayscale 2D images: a sharp variation of the intensity function across a portion of the image.



Gray level profile of a horizontal line through the image



Gray level profile of a horizontal line through the image



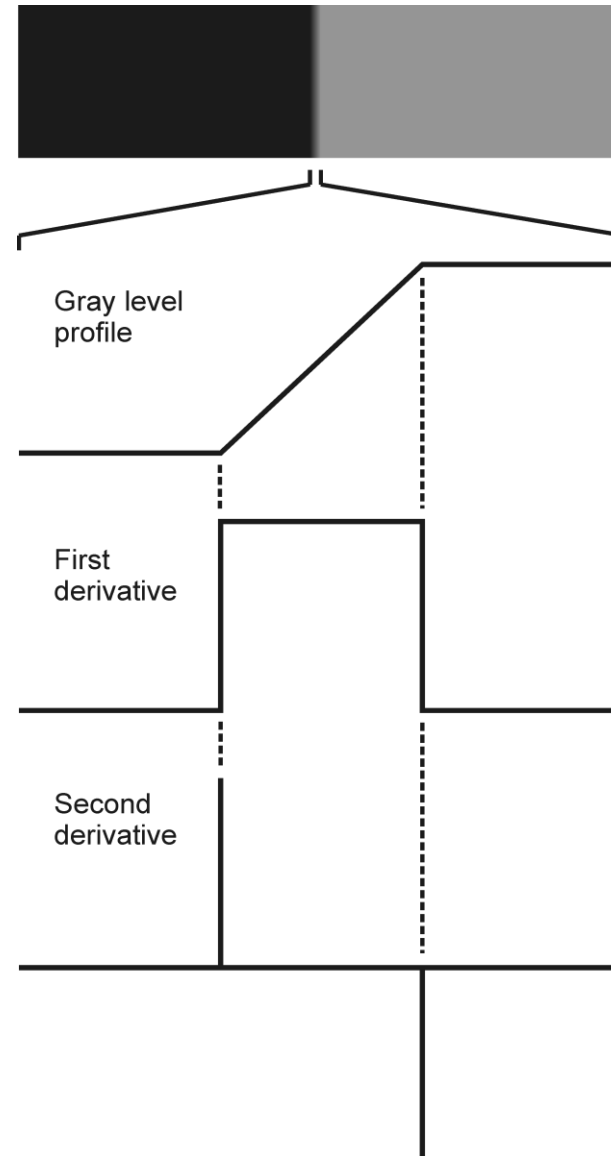
# Basic concepts

- Edge detection methods usually rely on calculations of the first or second derivative along the intensity profile.
  - The magnitude of the first derivative can be used to detect the presence of an edge at a certain point in the image.
  - The sign of second derivative can be used to determine whether a pixel lies on the dark or bright side of an edge.
  - Moreover, the zero crossing between its positive and negative peaks can be used to locate the center of thick edges.



# Basic concepts

- First and second derivative



# Basic concepts

- The influence of noise



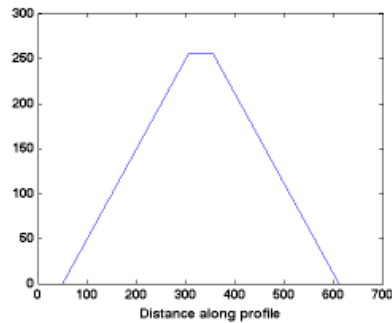
(a)



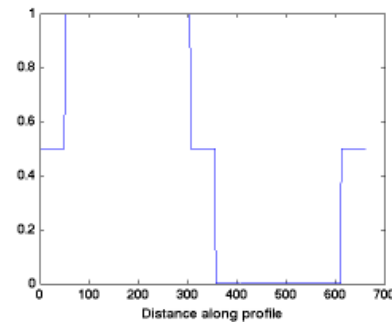
(b)



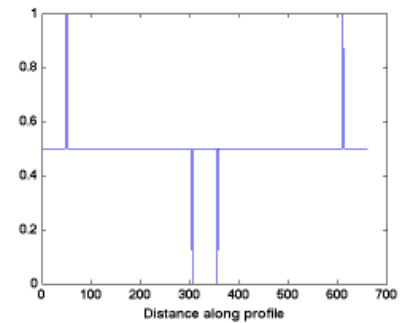
(c)



(d)



(e)



(f)



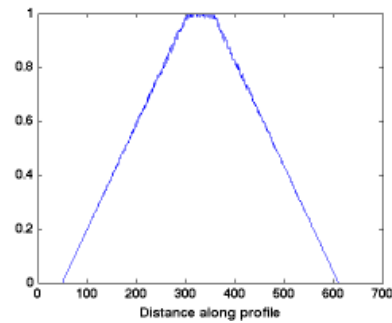
(g)



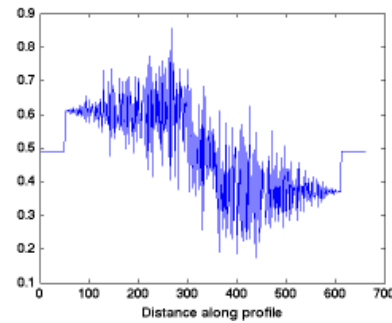
(h)



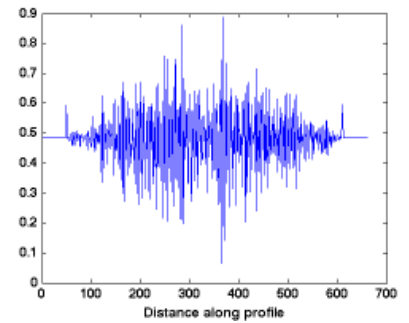
(i)



(j)



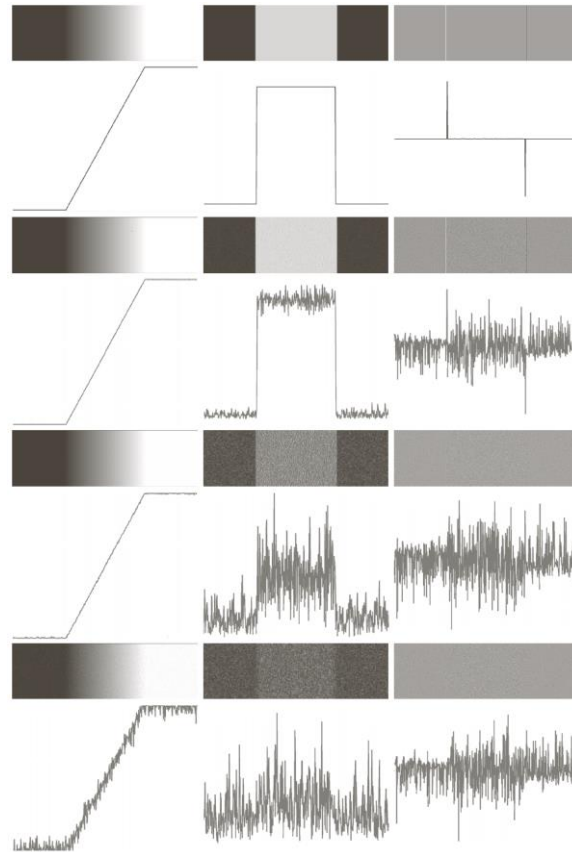
(k)



(l)



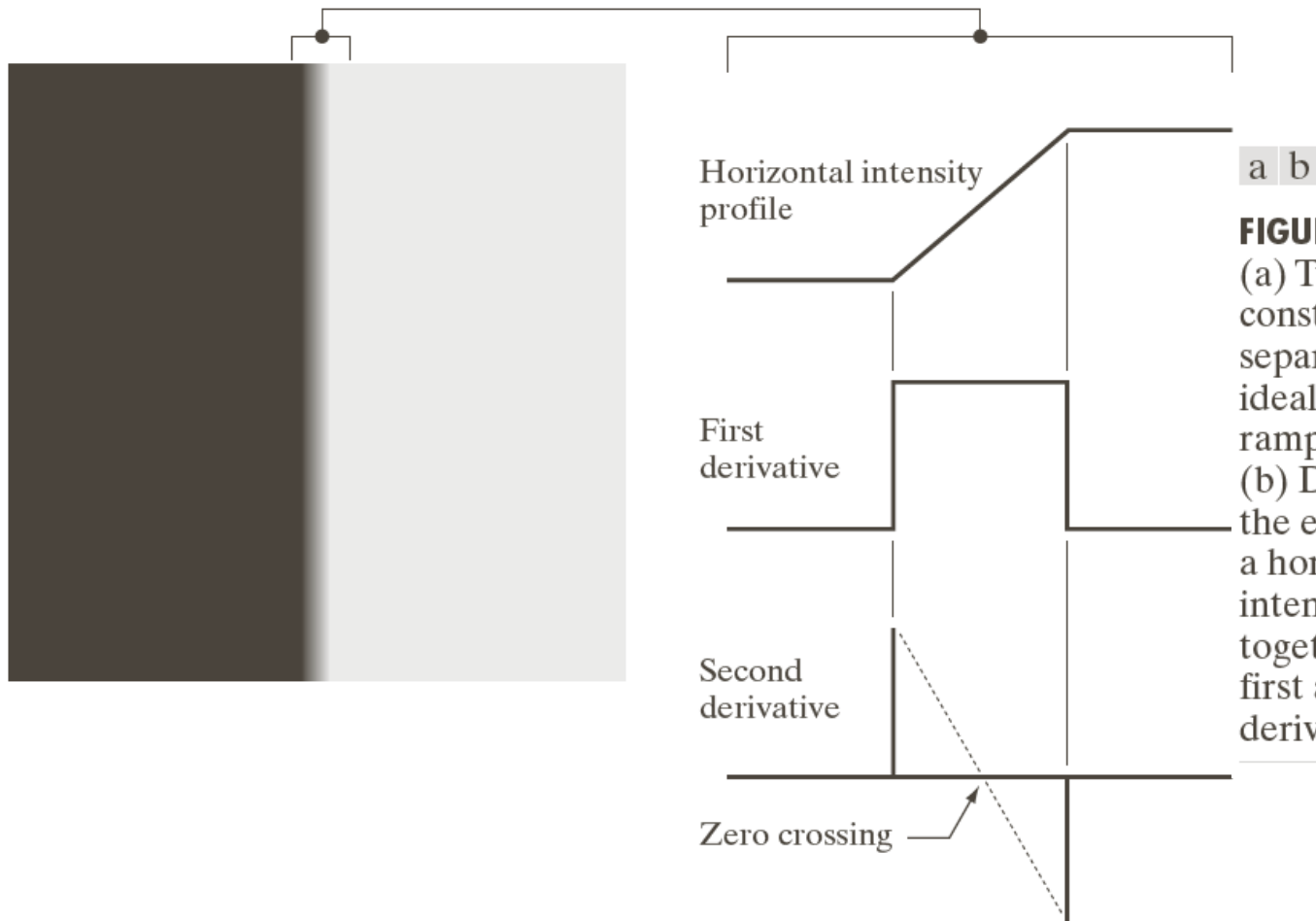
# Intensity profile of the edge with noise



**FIGURE 10.11** First column: Images and intensity profiles of a ramp edge corrupted by random Gaussian noise of zero mean and standard deviations of 0.0, 0.1, 1.0, and 10.0 intensity levels, respectively. Second column: First-derivative images and intensity profiles. Third column: Second-derivative images and intensity profiles.



# Intensity profile of the edge



**FIGURE 10.10**

(a) Two regions of constant intensity separated by an ideal vertical ramp edge.

(b) Detail near the edge, showing a horizontal intensity profile, together with its first and second derivatives.



# Basic concepts

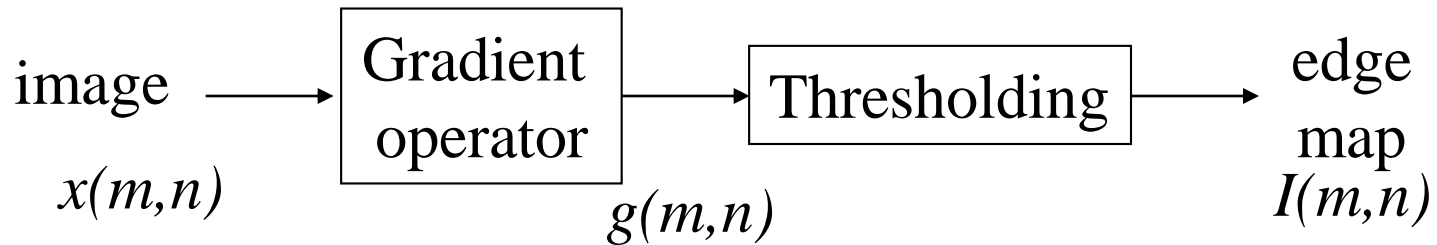
- The process of edge detection consists of three main steps:
  - Noise reduction
  - Detection of edge points
  - Edge localization
- In MATLAB: **edge**



# Gradient Operators

- Motivation: detect **changes**

change in the pixel value  $\longrightarrow$  large gradient



$$I(m,n) = \begin{cases} 1 & |g(m,n)| > th \\ 0 & otherwise \end{cases}$$

MATLAB function: `> help edge`

# Basic Edge Detection

- We know that we can find edge by using first and second derivative

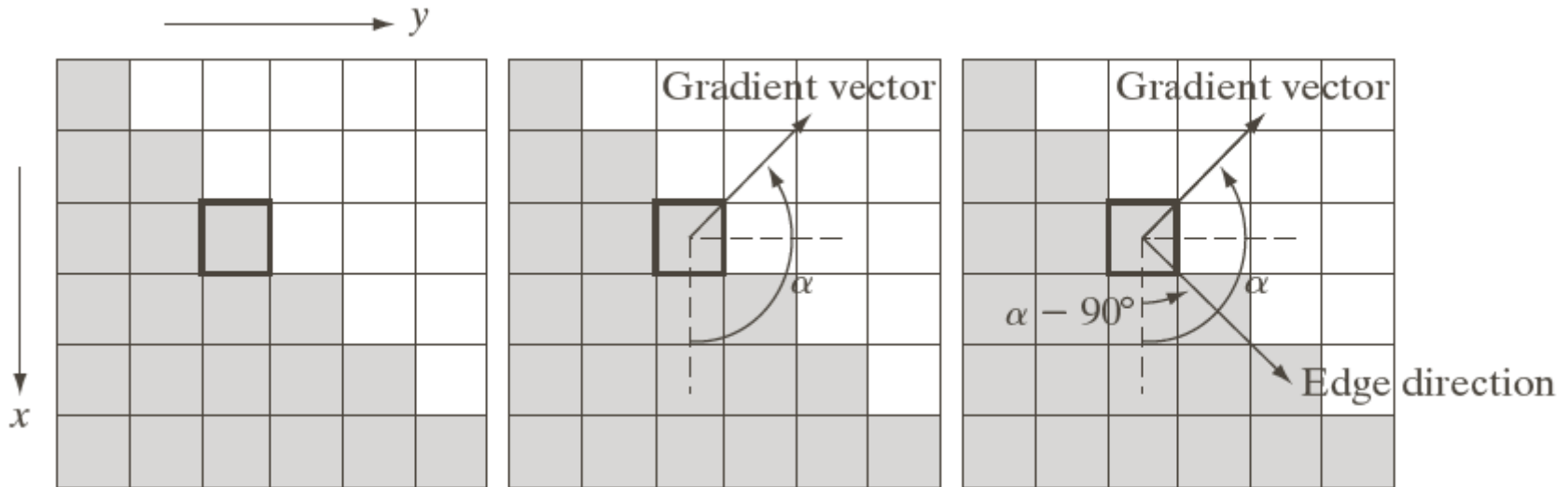
$$\nabla f = \text{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

- The magnitude is  $M(x, y) = \text{mag}(\nabla f) = \sqrt{g_x^2 + g_y^2}$

- The direction is  $\alpha(x, y) = \tan^{-1} \left[ \frac{g_y}{g_x} \right]$



# Gradient Vector



Grey pixel = 0, and white pixel = 1, using 3x3 neighborhood

From the picture  $g_x = -2$  (downward direction)

$$g_y = 2$$

Magnitude =  $2\sqrt{2}$  and angle = 145 degree

a b c

**FIGURE 10.12** Using the gradient to determine edge strength and direction at a point. Note that the edge is perpendicular to the direction of the gradient vector at the point where the gradient is computed. Each square in the figure represents one pixel.



# Gradient Operator

- Obtaining the gradient of an image requires the partial derivative

$$g_x = \frac{\partial f(x, y)}{\partial x} = f(x + 1, y) - f(x, y)$$

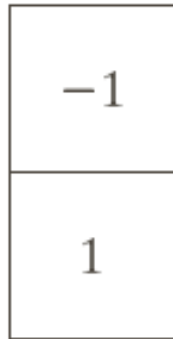
$$g_y = \frac{\partial f(x, y)}{\partial y} = f(x, y + 1) - f(x, y)$$

simplified version (no square root)



# One Dimension Mark

- Only use to present conceptually



$g_y$



$g_x$



# Image Neighborhood

$z_1$	$z_2$	$z_3$
$z_4$	$z_5$	$z_6$
$z_7$	$z_8$	$z_9$

Image neighborhood





# Simple 3x3 mask

$$g_x = \frac{\partial f}{\partial x} = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$

$$g_y = \frac{\partial f}{\partial y} = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$



# First-order derivative edge detection

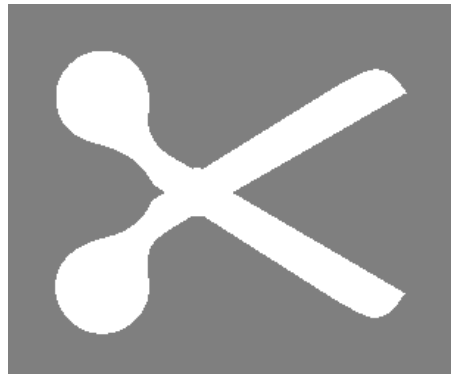
- Prewitt operators:

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

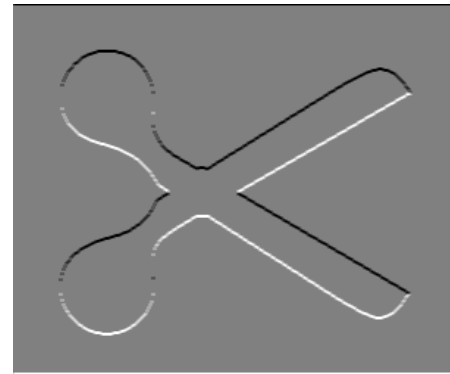
$$h_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

# First-order derivative edge detection

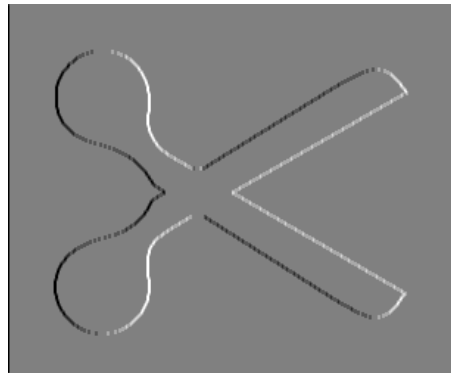
- Prewitt operator



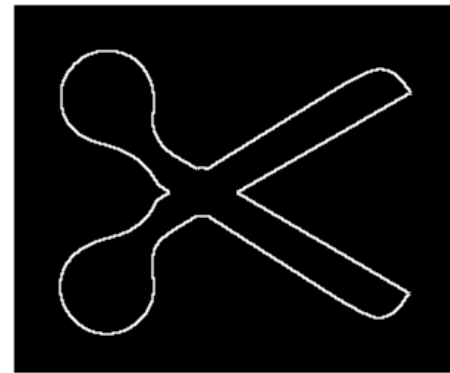
(a)



(b)



(c)



(d)



# Examples



original image



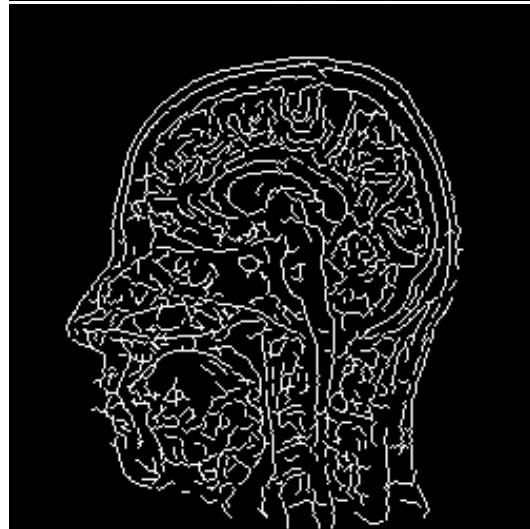
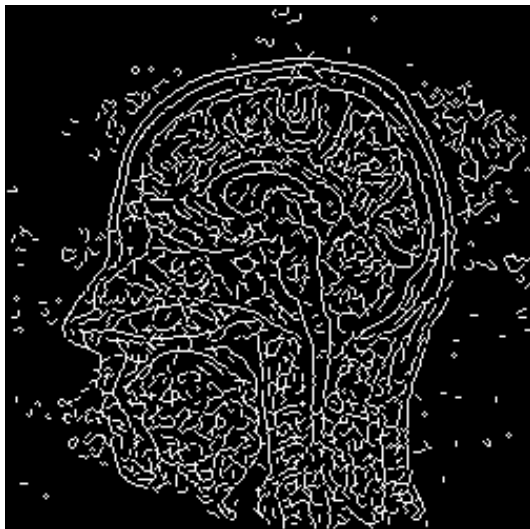
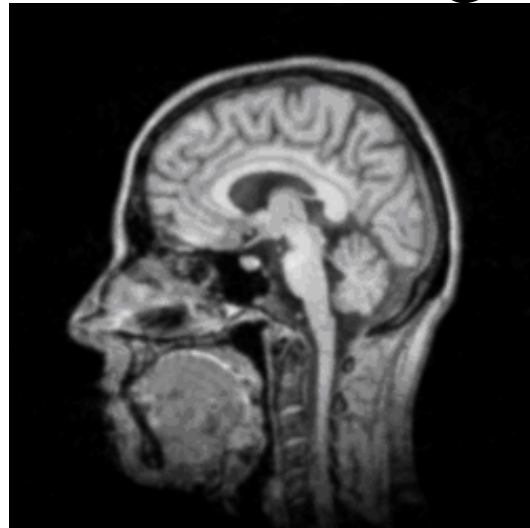
horizontal edge



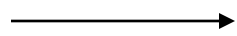
vertical edge

Prewitt operator

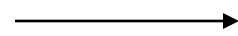
# Effect of Thresholding Parameters



small

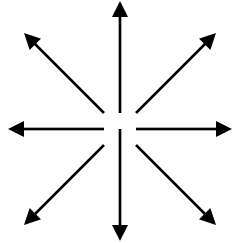


threshold



large

# Compass Operators

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

$$g(m, n) = \max_k \{ |g_k(m, n)| \}$$

# Examples



Compass operator

# Sobel Edge Detector

- The idea is to give more emphasis to on-axis pixels

$$\begin{aligned}g &= [G_x^2 + G_y^2]^{1/2} \\ &= \{[(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)]^2 \\ &\quad + [(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)]^2\}^{1/2}\end{aligned}$$





# First-order derivative edge detection

- Sobel operators:

$$h_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

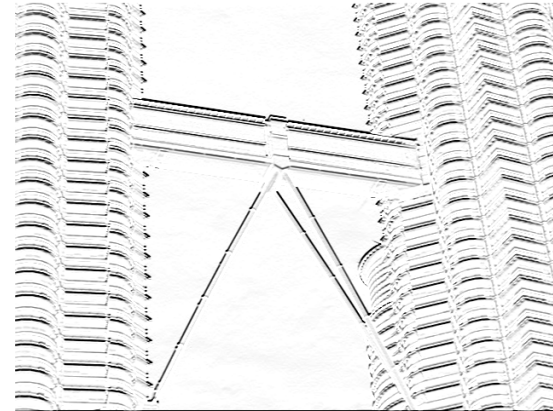
$$h_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

# First-order derivative edge detection

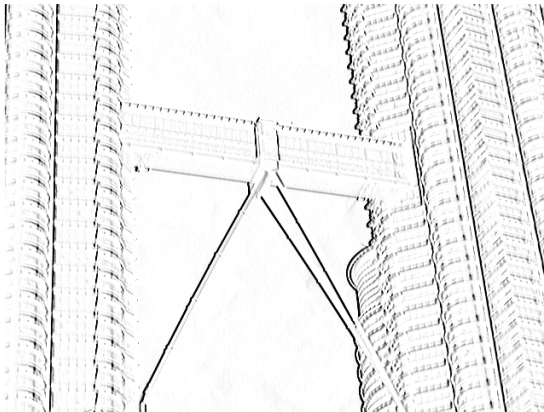
- Sobel operator



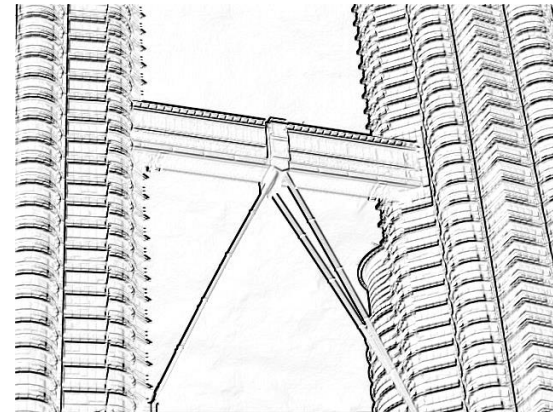
(a)



(b)



(c)



(d)



# 2-dimension mark

$z_1$	$z_2$	$z_3$
$z_4$	$z_5$	$z_6$
$z_7$	$z_8$	$z_9$

Image neighborhood

a
b c
d e
f g

-1	-2	-1
0	0	0
1	2	1

$$g_x = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

-1	0	1
-2	0	2
-1	0	1

$$g_y = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$

Sobel

-1	-1	-1
0	0	0
1	1	1

$$g_x = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$

-1	0	1
-1	0	1
-1	0	1

$$g_y = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)$$

Prewitt

-1	0
0	1

$$g_x = z_9 - z_5$$

0	-1
1	0

$$g_y = z_8 - z_6$$

Roberts

**FIGURE 10.14**

A  $3 \times 3$  region of an image (the  $z$ 's are intensity values) and various masks used to compute the gradient at the point labeled  $z_5$ .



# 2-dimension mark

0	1	1	-1	-1	0
-1	0	1	-1	0	1
-1	-1	0	0	1	1

Prewitt

0	1	2	-2	-1	0
-1	0	1	-1	0	1
-2	-1	0	0	1	2

Sobel

a	b
c	d

**FIGURE 10.15**  
Prewitt and Sobel  
masks for  
detecting diagonal  
edges.



# Original Image



# Different Edge Detection



a	b
c	d

**FIGURE 10.18**  
 Same sequence as in Fig. 10.16, but with the original image smoothed using a  $5 \times 5$  averaging filter prior to edge detection.



b)  $g_x$

c)  $g_y$

d)  $|g_x| + |g_y|$



a b  
c d  
e f

**FIGURE 11.6**

(a) Original image. (b) Result of function edge

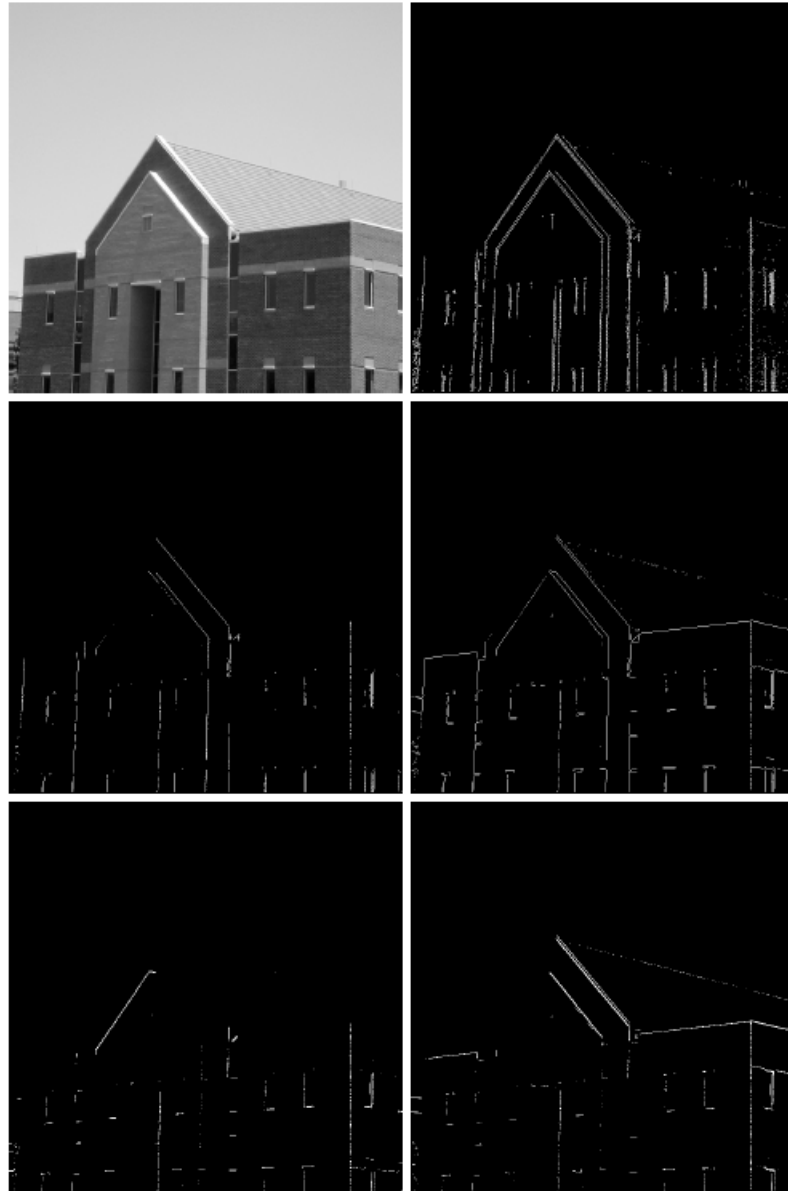
using a vertical Sobel mask with the threshold determined automatically.

(c) Result using a specified threshold.

(d) Result of determining both vertical and horizontal edges with a specified threshold.

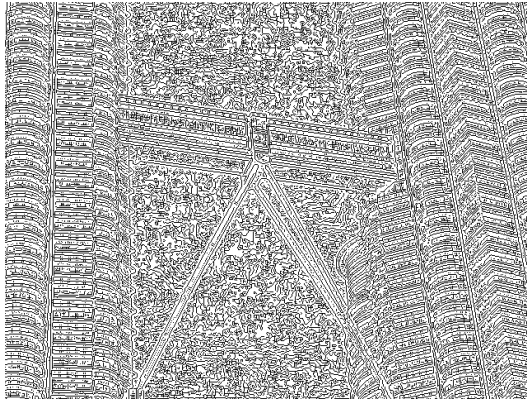
(e) Result of computing edges at  $-45^\circ$  with `imfilter` using a specified mask and a specified threshold.

(f) Result of computing edges at  $+45^\circ$  with `imfilter` using a specified mask and a specified threshold.

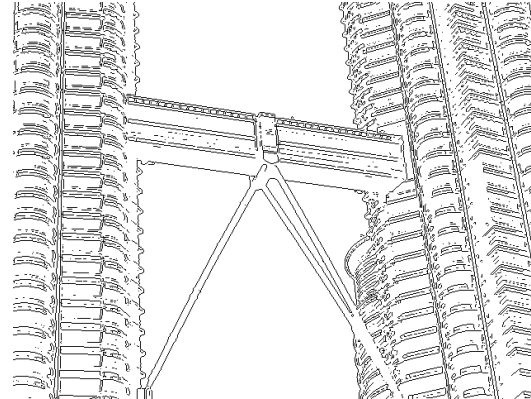


# First-order derivative edge detection

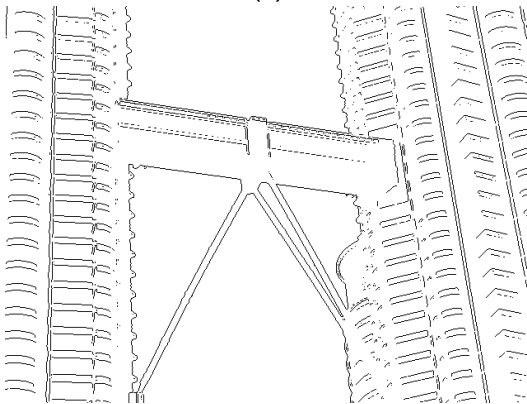
- Sobel operator with threshold (example)



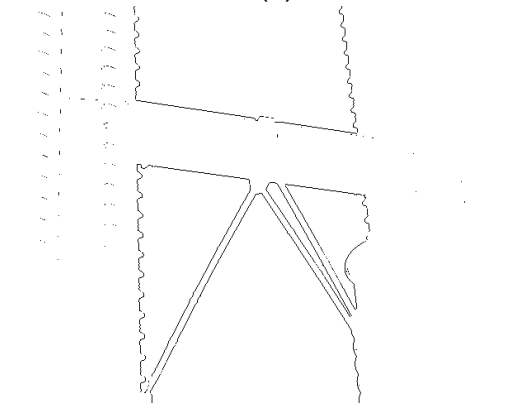
(a)



(b)



(c)



(d)







# Second-order derivative edge detection

- Second-order derivative  
(the Laplacian of an image):

$$\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x + 1, y) + f(x - 1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y + 1) + f(x, y - 1) - 2f(x, y)$$

$$\nabla^2 f = [f(x + 1, y) + f(x - 1, y) + f(x, y + 1) + f(x, y - 1)] - 4f(x, y)$$





# Second-order derivative edge detection

- Laplacian of an image:

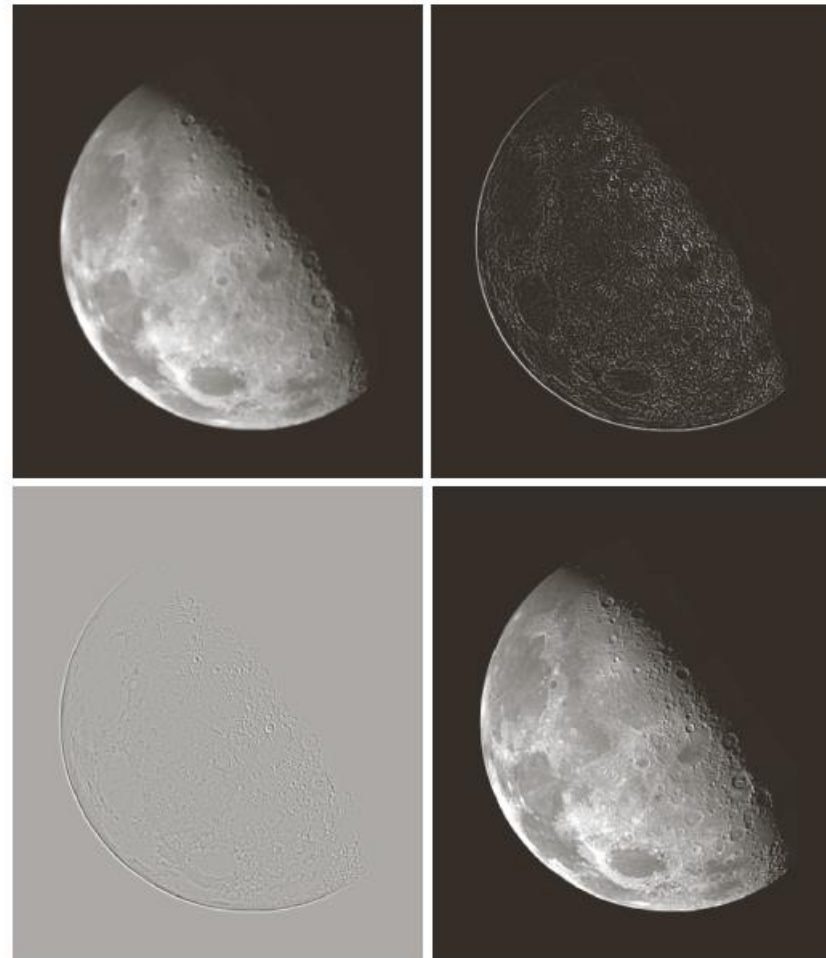
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- The general version:

$$\begin{bmatrix} \frac{\alpha}{1 + \alpha} & \frac{1 - \alpha}{1 + \alpha} & \frac{\alpha}{1 + \alpha} \\ \frac{1 - \alpha}{1 + \alpha} & -4 & \frac{1 - \alpha}{1 + \alpha} \\ \frac{\alpha}{1 + \alpha} & \frac{1 - \alpha}{1 + \alpha} & \frac{\alpha}{1 + \alpha} \end{bmatrix}$$



# Example



a b  
c d

**FIGURE 3.17**

(a) Image of the North Pole of the moon.

(b) Laplacian filtered image, using `uint8` format. (Because `uint8` is an unsigned type, negative values in the output were clipped to 0.)

(c) Laplacian filtered image obtained using floating point.

(d) Enhanced result, obtained by subtracting (c) from (a).

(Original image courtesy of NASA.)



# Laplacian Operators

- Gradient operator: first-order derivative  
sensitive to abrupt change, but not **slow change**

↓

second-order derivative:  $\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$   
(Laplacian operator)

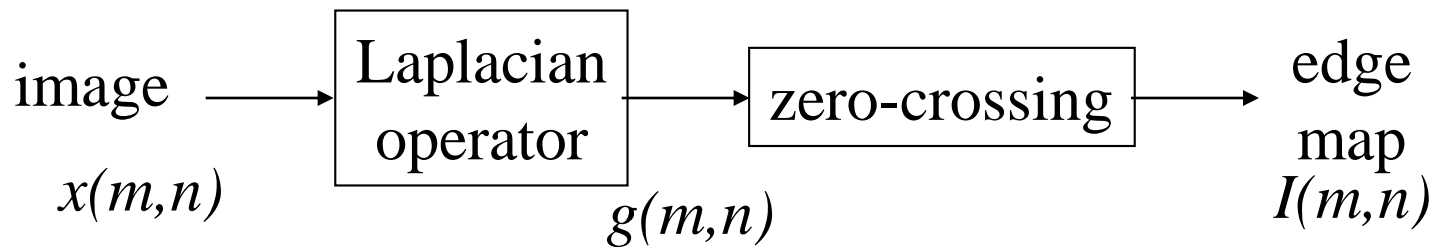
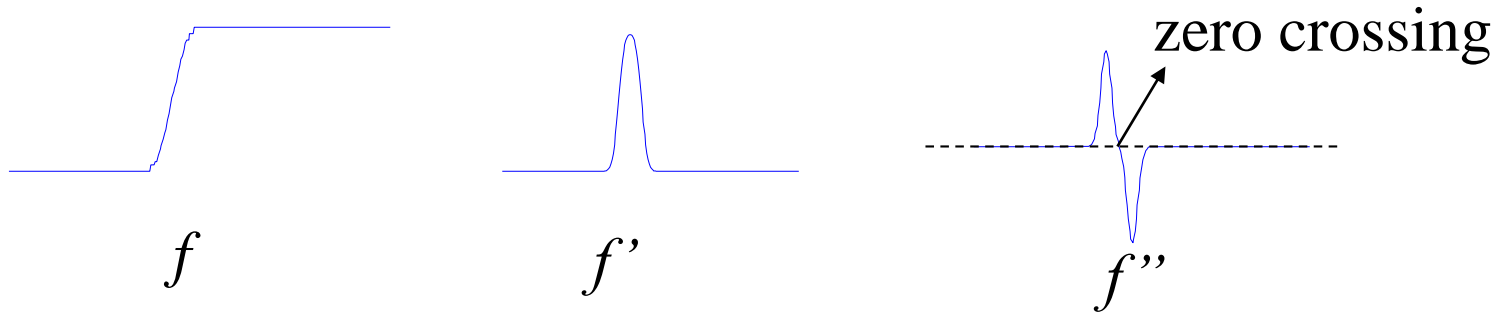
$$\frac{\partial^2 f}{\partial x^2} = 0 \longrightarrow \text{local extreme in } f'$$

- Discrete Laplacian operator

$$\frac{1}{1+a} \begin{bmatrix} a & 1-a & a \\ 1-a & -4 & 1-a \\ a & 1-a & a \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

a=0 a=0.5

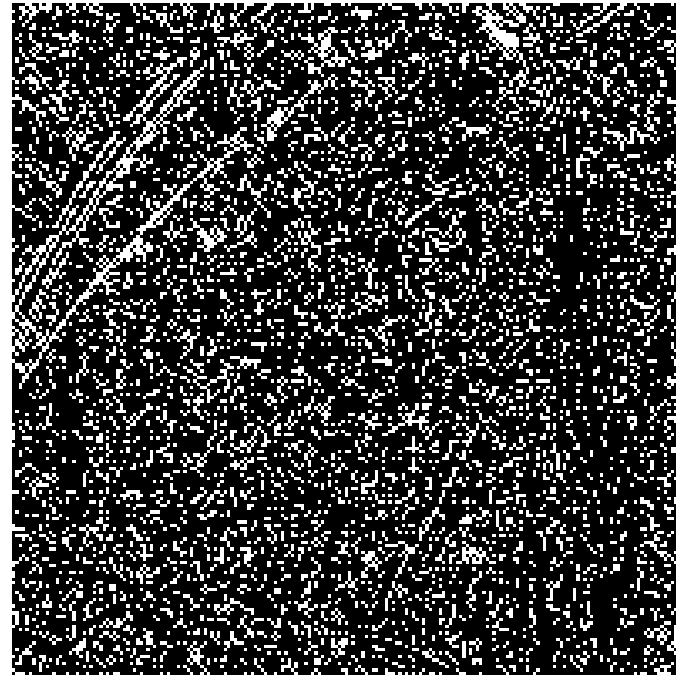
# Zero Crossings



# Examples



original image



zero-crossings

Question: why is it so sensitive to noise (many false alarms)?

Answer: a sign flip from 0.01 to -0.01 is treated the same as from 100 to -100

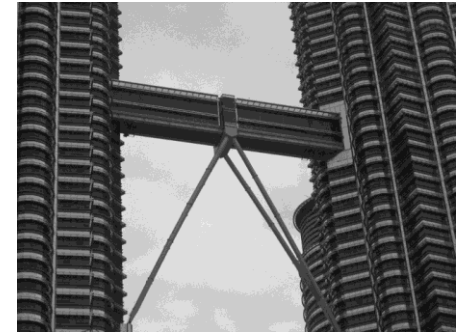
# Second-order derivative edge detection

- Laplacian and zero-cross

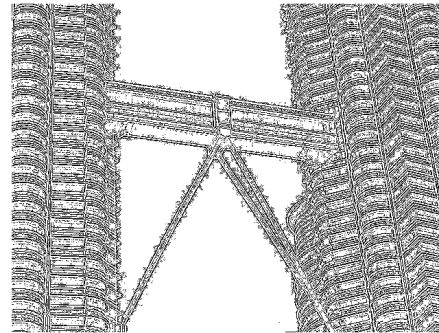
- a) Input
- b) Laplacian
- c) Zero crossing
- d) Input with noise
- e) Laplacian
- f) Zero crossing



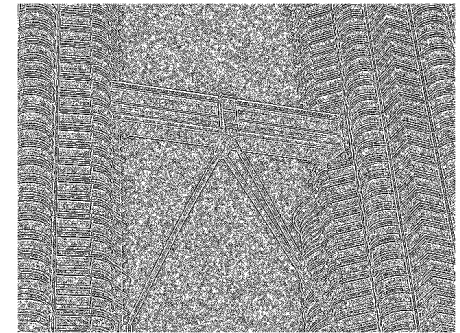
(a)



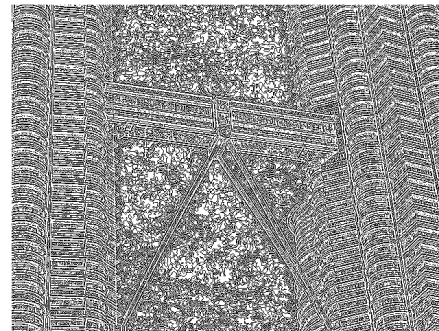
(d)



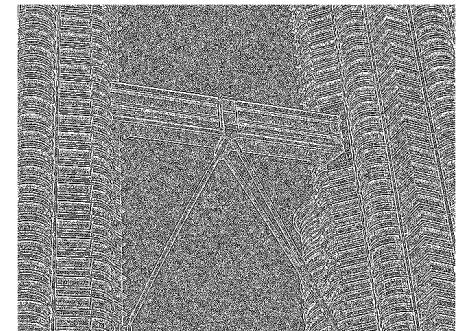
(b)



(e)



(c)



(f)





```
int main(int argc, char** argv)
{
    Mat src = imread("d:/image/cameraman2.tif", IMREAD_GRAYSCALE);
    Mat dst, dst2;
    Sobel(src, dst, -1, 1, 1);
    Laplacian(src, dst2, -1);
    // Show the results
    namedWindow(" ORIGINAL ", WINDOW_AUTOSIZE);
    imshow(" ORIGINAL ", src);
    namedWindow(" SOBEL ", WINDOW_AUTOSIZE);
    imshow(" SOBEL ", dst);
    namedWindow(" LAPLACIAN ", WINDOW_AUTOSIZE);
    imshow(" LAPLACIAN ", dst2);
    waitKey();
    return 0;
}
```





# Ideas to Improve Robustness

- Linear filtering
  - Use a Gaussian filter to smooth out noise component
    - Laplacian of Gaussian
- Spatially-adaptive (Nonlinear) processing
  - Apply different detection strategies to smooth areas (low-variance) and non-smooth areas (high-variance)
    - Robust Laplacian edge detector
- Return single response to edges (not multiple edge pixels)
  - Hysteresis thresholding → Canny's edge detector

# Second-order derivative edge detection

- Problems with Laplacian:
  - It generates “double edges”, i.e., positive and negative values for each edge.
  - It is extremely sensitive to noise.

- In MATLAB:

```
h = fspecial('laplacian',0);  
J = edge(I,'zerocross',t,h);
```



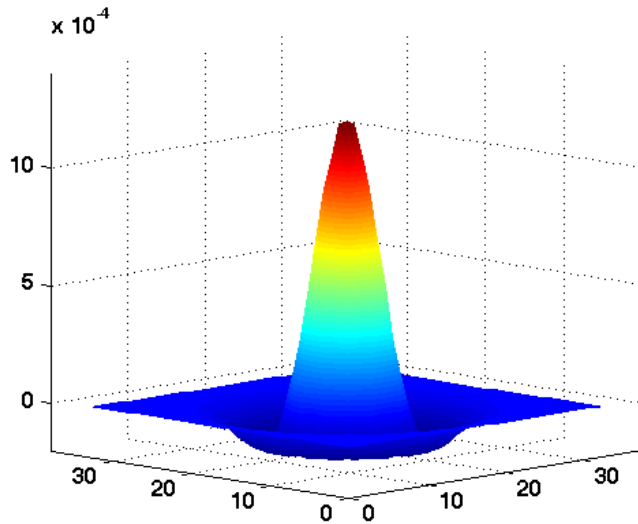
# Second-order derivative edge detection

- Laplacian of Gaussian (LoG):
  - Works by smoothing the image with a Gaussian low-pass filter, and then applying a Laplacian edge detector to the result.
  - The LoG filter can sometimes be approximated by taking the differences of two Gaussians of different widths, in a method known as *Difference of Gaussians* (DoG).

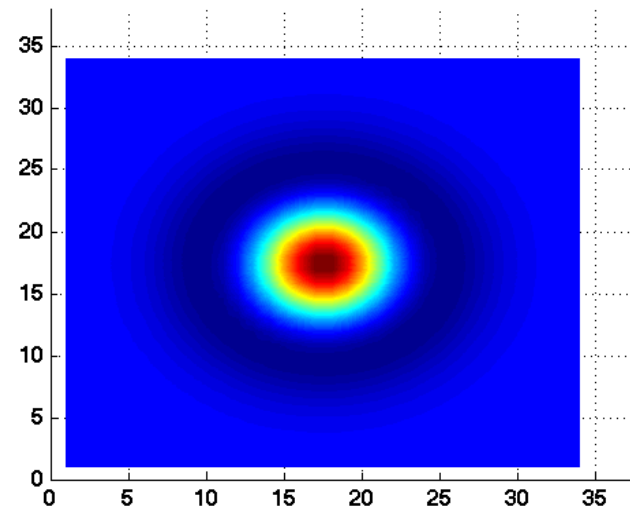


# Second-order derivative edge detection

- Laplacian of Gaussian (LoG) transfer function (Mexican hat)



(a)

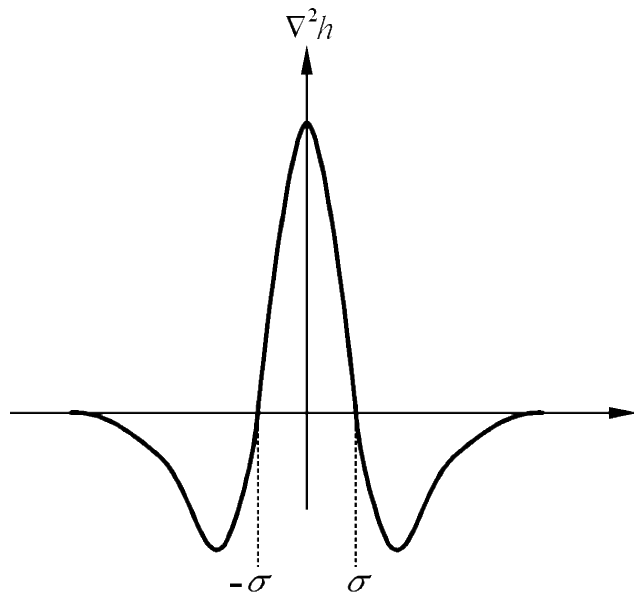


(b)



# Second-order derivative edge detection

- Laplacian of Gaussian (LoG) transfer function (2D cross-section of 3D plot) and mask (kernel)

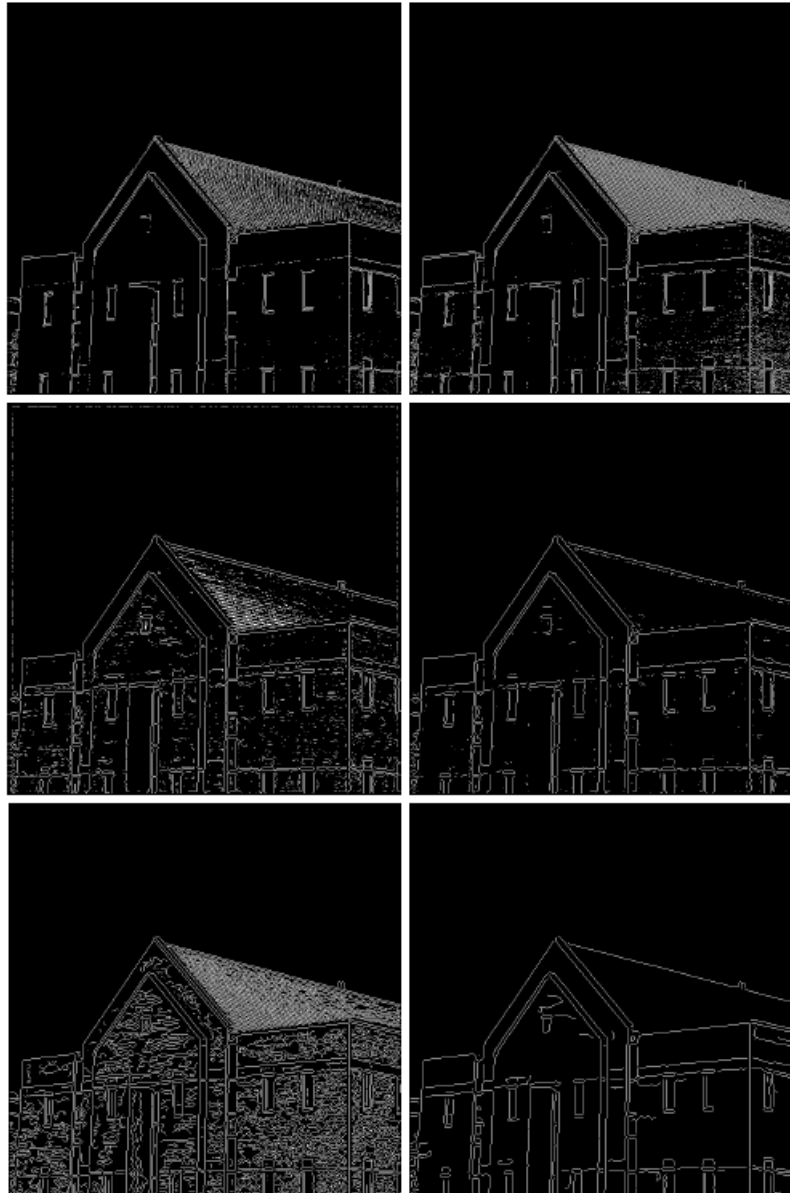


0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0



a	b
c	d
e	f

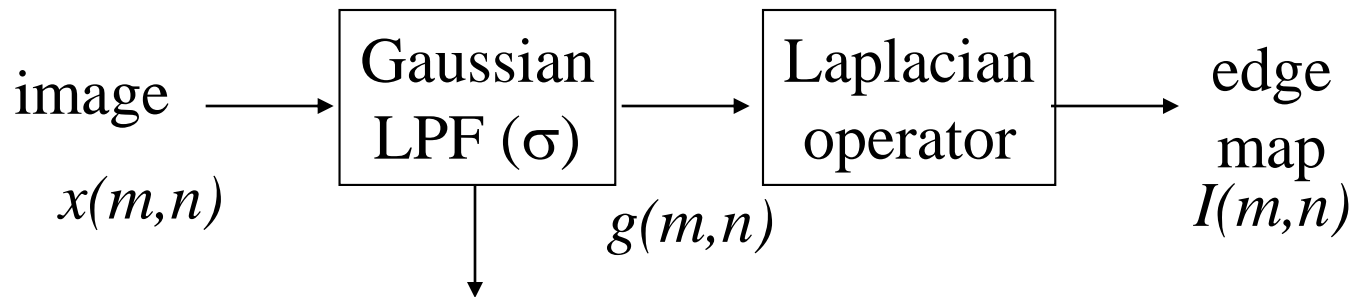
**FIGURE 11.7**  
Left column:  
Default results for  
the Sobel, LoG,  
and Canny edge  
detectors. Right  
column: Results  
obtained  
interactively to  
bring out the  
principal features  
in the original  
image of  
Fig. 11.6(a), while  
reducing  
irrelevant detail.  
The Canny edge  
detector produced  
the best result.



# Laplacian of Gaussian

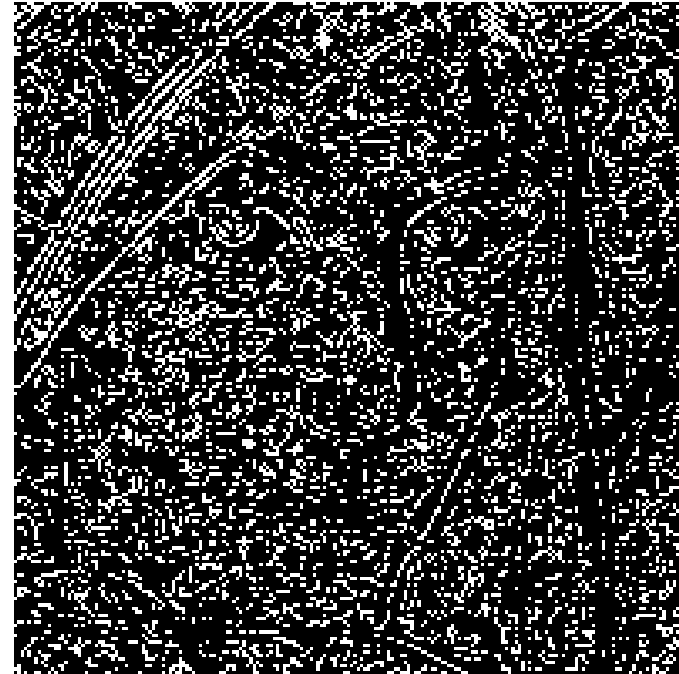
- Generalized Laplacian operator

$$h(m, n) = c \left[ 1 - \frac{(m^2 + n^2)}{\sigma^2} \right] \exp\left(-\frac{m^2 + n^2}{2\sigma^2}\right)$$



Pre-filtering: attenuate the noise sensitivity of the Laplacian

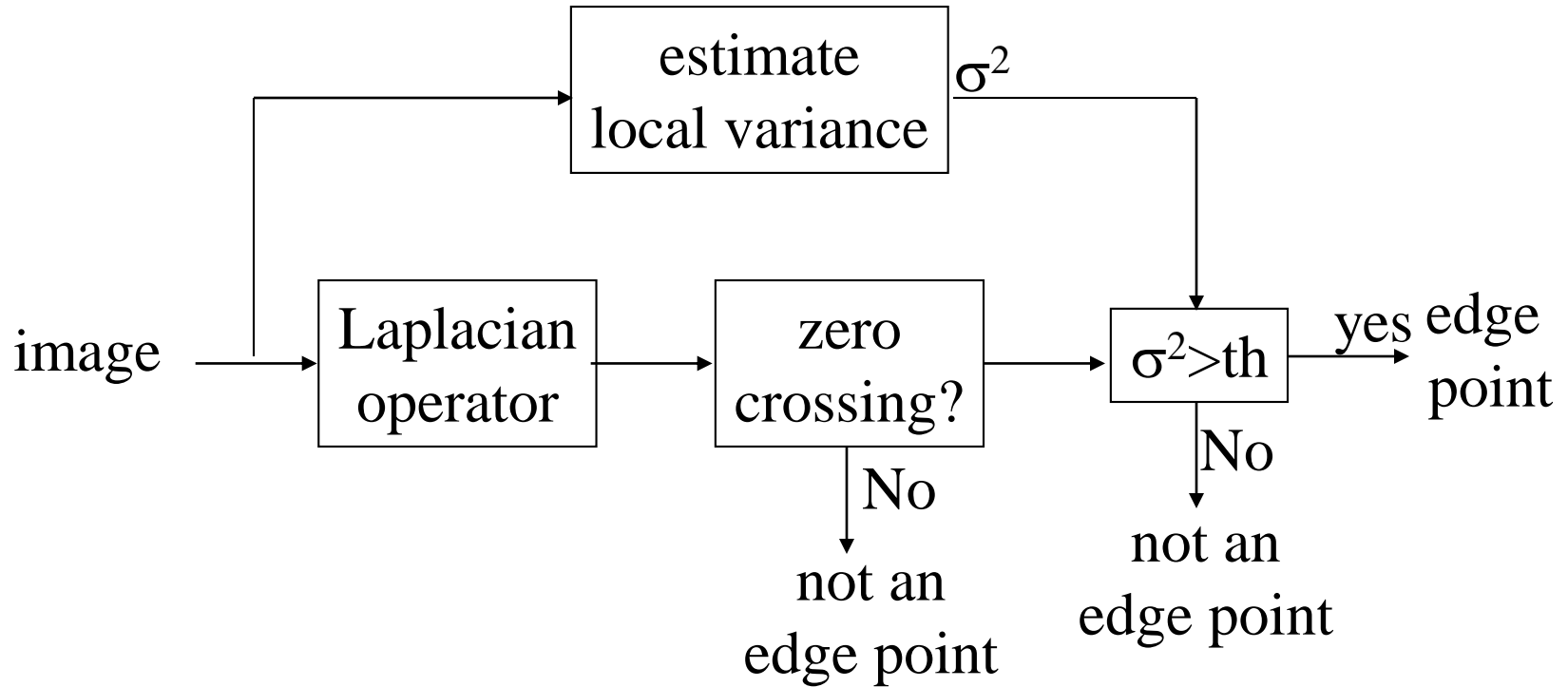
# Examples



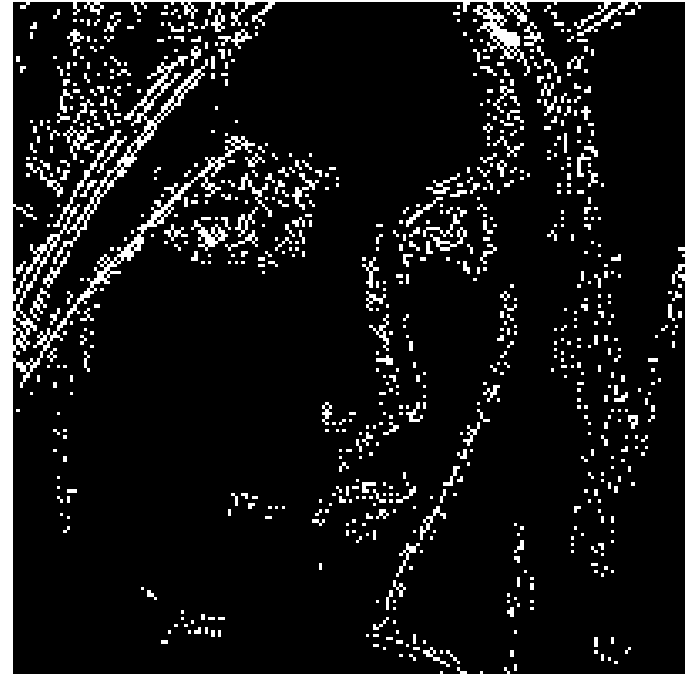
Better than Laplacian alone but still sensitive due to zero crossing



# Robust Laplacian-based Edge Detector



# Examples



More robust but return multiple edge pixels (poor localization)



# Canny Edge Detector

Canny edge approach is based on 3 objectives:

- Low error rate: all edges should be found
- Edge points should be well localized: the edges located must be as close as possible to true edges
- Single edge point response: should return only one point for each true edge point



# Flow-chart of Canny Edge Detector\*

*(J. Canny'1986)*

Original image

**Smoothing** by Gaussian convolution

**Differential operators** along x and y axis

**Non-maximum suppression**  
finds peaks in the image gradient

**Hysteresis thresholding** locates edge strings

Edge map

# The Canny edge detector

1. The input image is smoothed using a Gaussian low pass filter (LPF), with a specified value of  $\sigma$ .
2. The local gradient (intensity and direction) is computed for each point in the smoothed image.
3. The edge points at the output of step 2 result in wide ridges. The algorithm thins those ridges, leaving only the pixels at the top of each ridge (*non-maximal suppression*).
4. The ridge pixels are then thresholded using two thresholds,  $T_{low}$  and  $T_{high}$ : ridge pixels with value greater than  $T_{high}$  are considered **strong** edge pixels; ridge pixels with values between  $T_{low}$  and  $T_{high}$  are said to be **weak** pixels. This process is known as *hysteresis thresholding*.
5. The algorithm performs edge linking, aggregating weak pixels that are 8-connected to the strong pixels.

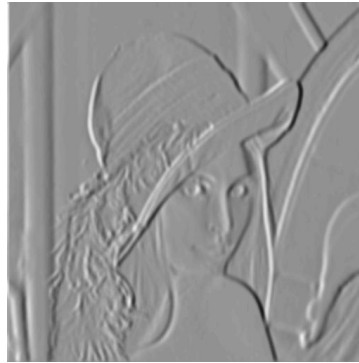
In MATLAB: `J = edge(I, 'canny', T, sigma);`



# Canny Edge Detector Example



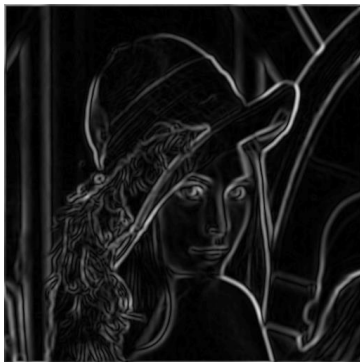
original image



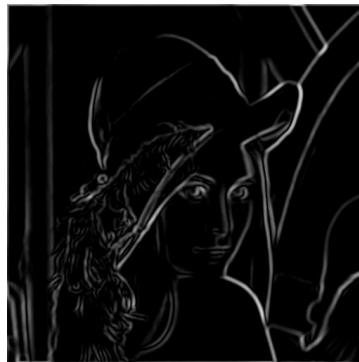
vertical edges



horizontal edges



norm of the gradient

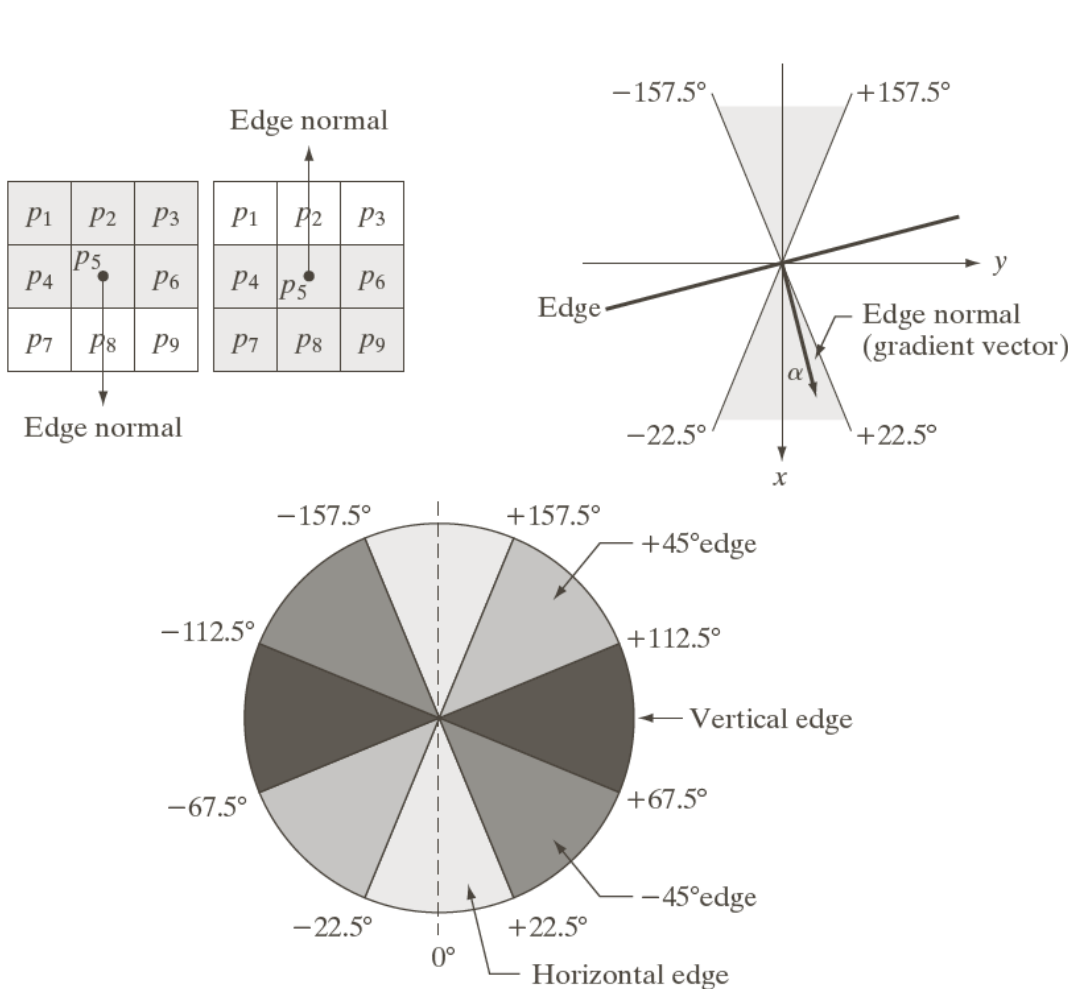


after thresholding



after thinning

# Compute Image gradient



a b  
c

**FIGURE 10.24**  
 (a) Two possible orientations of a horizontal edge (in gray) in a  $3 \times 3$  neighborhood.  
 (b) Range of values (in gray) of  $\alpha$ , the direction angle of the *edge normal*, for a horizontal edge.  
 (c) The angle ranges of the edge normals for the four types of edge directions in a  $3 \times 3$  neighborhood. Each edge direction has two ranges, shown in corresponding shades of gray.



# Example



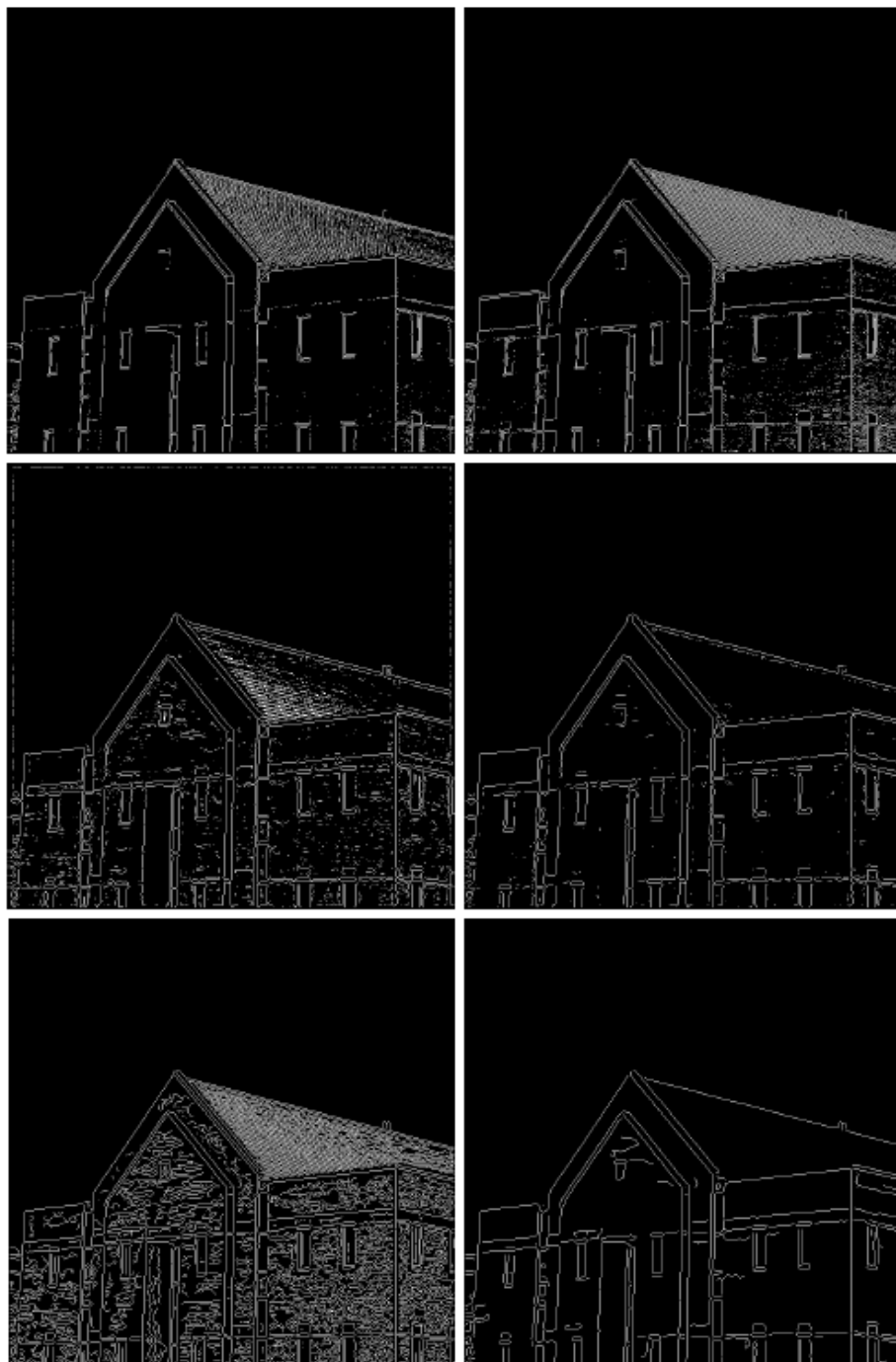
a	b
c	d


**FIGURE 10.25**  
 (a) Original image of size  $834 \times 1114$  pixels, with intensity values scaled to the range  $[0, 1]$ .  
 (b) Thresholded gradient of smoothed image.  
 (c) Image obtained using the Marr-Hildreth algorithm.  
 (d) Image obtained using the Canny algorithm. Note the significant improvement of the Canny image compared to the other two.



a b  
c d  
e f

**FIGURE 11.7**  
Left column:  
Default results for  
the Sobel, LoG,  
and Canny edge  
detectors. Right  
column: Results  
obtained  
interactively to  
bring out the  
principal features  
in the original  
image of  
Fig. 11.6(a), while  
reducing  
irrelevant detail.  
The Canny edge  
detector produced  
the best result.



A small image showing the result of Canny edge detection on a blue and white image, with edges highlighted in red and green.

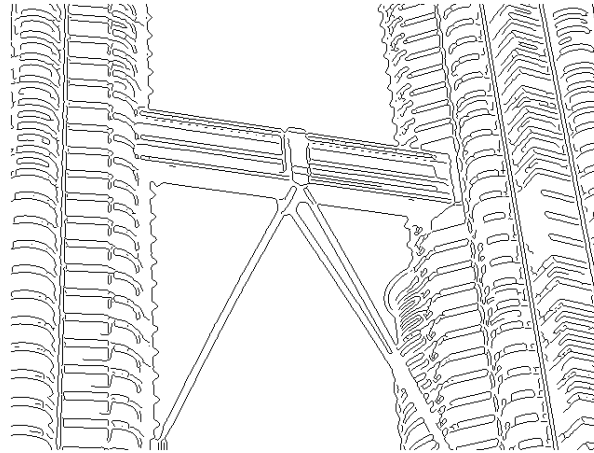
```
void CannyThreshold(int, void*){
    blur(src, detected_edges, Size(3, 3));
    Canny(detected_edges, detected_edges, lowThreshold, lowThreshold*ratio,
kernel_size);
    dst = Scalar::all(0);
    src.copyTo(dst, detected_edges);
    imshow(window_name, dst);
}
```

```
int main(int argc, char** argv){
    src = imread("d:/image/cameraman2.tif", IMREAD_GRAYSCALE);
    dst.create(src.size(), src.type());
    namedWindow(window_name, CV_WINDOW_AUTOSIZE);
    createTrackbar("Min Threshold:", window_name, &lowThreshold,
max_lowThreshold, CannyThreshold);
    CannyThreshold(0, 0);
    waitKey(0);
    return 0;
}
```

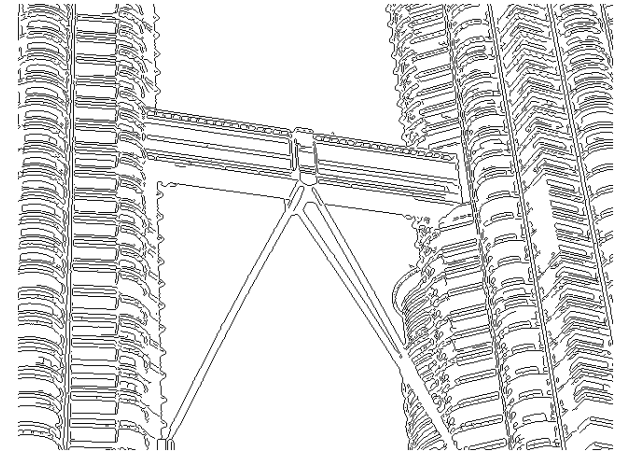


# The Canny edge detector

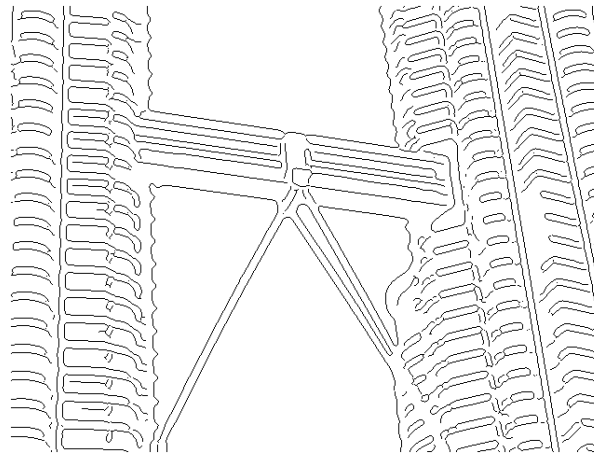
- (a) default values ( $\sigma = 1$ ,  
 $T_{\text{flow}} = 0.0625$ ,  
 $T_{\text{high}} = 0.1563$ );
- (b)  $\sigma = 0.5$ ;
- (c)  $\sigma = 2$ ;
- (d)  $\sigma = 1$ ,  
 $T_{\text{flow}} = 0.01$ ,  
 $T_{\text{high}} = 0.1$ .



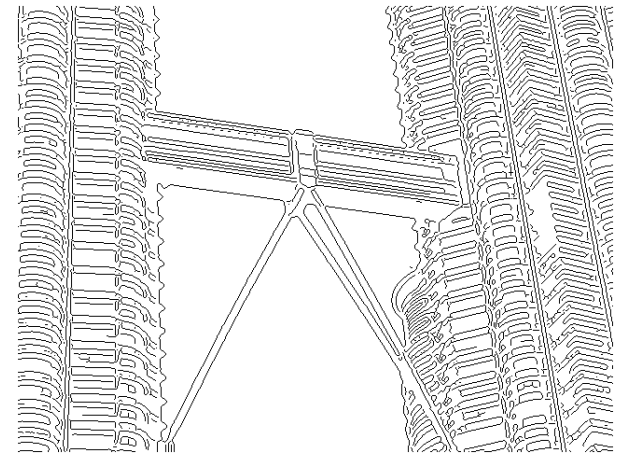
(a)



(b)



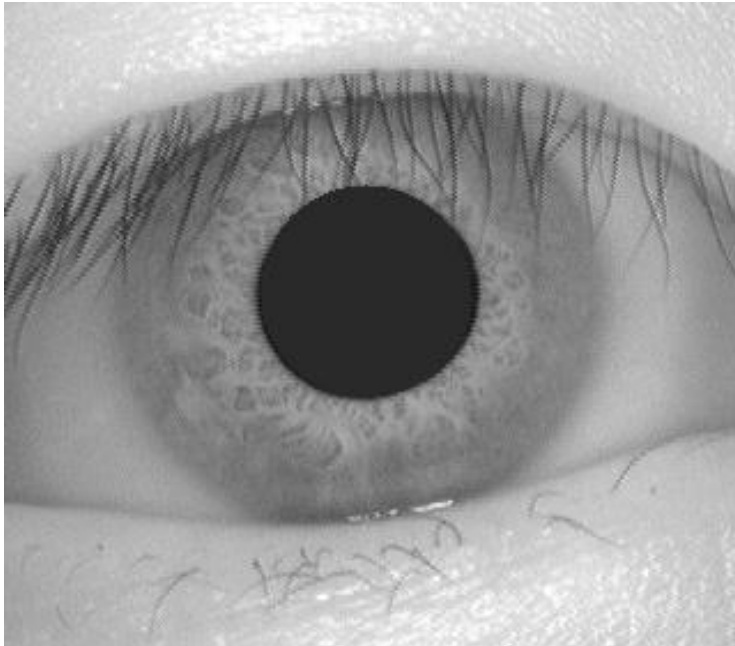
(c)



(d)



# Edge Detection in Iris Image



```
>l=imread("ris.bmp");  
>c=edge(l, 'canny');  
>[x,y]=find(c==1);
```

# Edge linking and boundary detection

- Goal of edge detection: to produce an image containing *only the edges* of the original image.
- However, due to the many technical challenges (noise, shadows, occlusion, etc.), most edge detection algorithms will output an image containing fragmented edges.
- Additional processing is needed to turn fragmented edge segments into useful lines and object boundaries.
- Hough transform: a global method for edge linking and boundary detection.



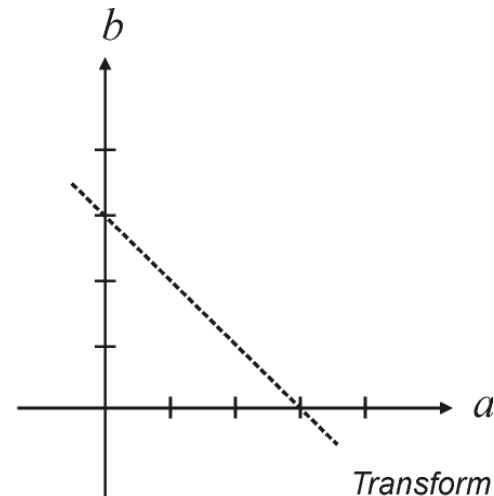
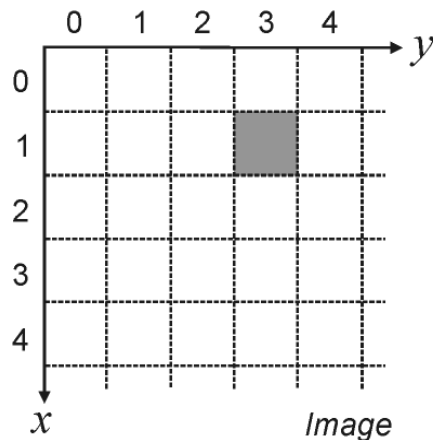
# The Hough transform

- A mathematical method designed to find lines in images.
  - It can be used for linking the results of edge detection, turning potentially sparse, broken, or isolated edges into useful lines that correspond to the actual edges in the image.



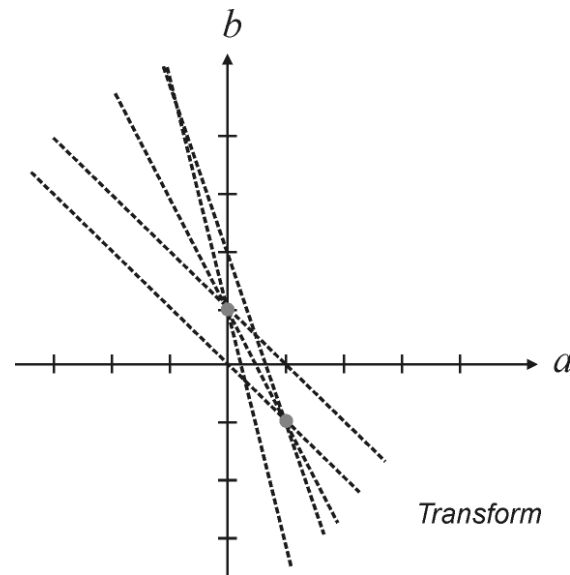
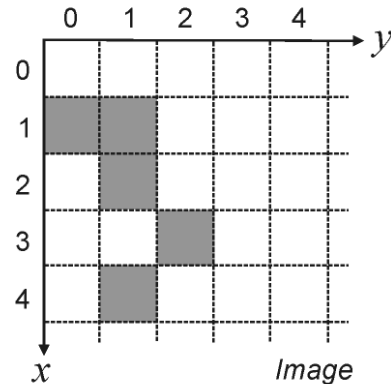
# The Hough transform

- Let  $(x,y)$  be the coordinates of a point in a binary image (containing thresholded edge detection results).
- The Hough transform stores in an *accumulator array* all pairs  $(a,b)$  that satisfy the equation  $y = ax + b$ . The  $(a,b)$  array is called the *transform array*.
  - Example:, the point  $(x,y) = (1,3)$  in the input image will result in the equation  $b = -a + 3$ , which can be plotted as a line that represents all pairs  $(a,b)$  that satisfy this equation.



# The Hough transform

- Since each point in the image will map to a line in the transform domain, repeating the process for other points will result in many intersecting lines, one per point.
- The meaning of two or more lines intersecting in the transform domain is that the points to which they correspond are aligned in the image.
- The points with the greatest number of intersections in the transform domain correspond to the longest lines in the image.





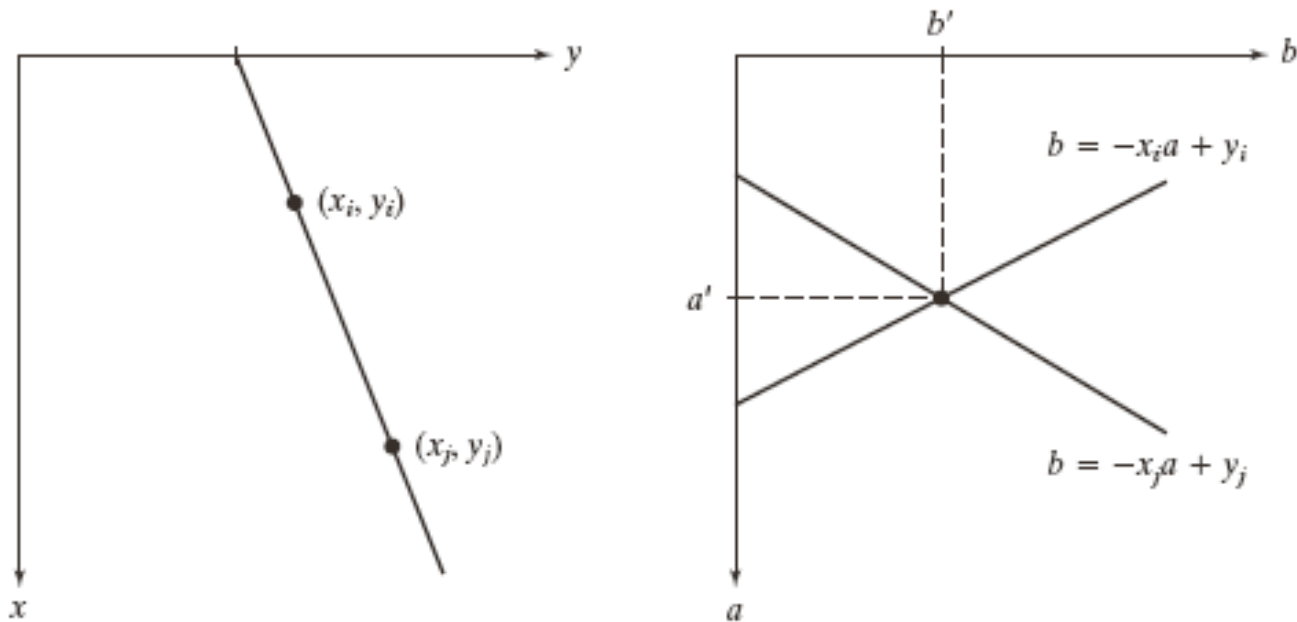


# Line Detection using Hough Transform

- Consider a line that passes through points  $(x_i, y_i)$  and  $(x_j, y_j)$
- The line equation can be written as:  
$$y_i = a x_i + b \text{ and } y_j = a x_j + b$$
- We can also write  
$$b = -x_i a + y_i \text{ and } b = -x_j a + y_j$$



# Example



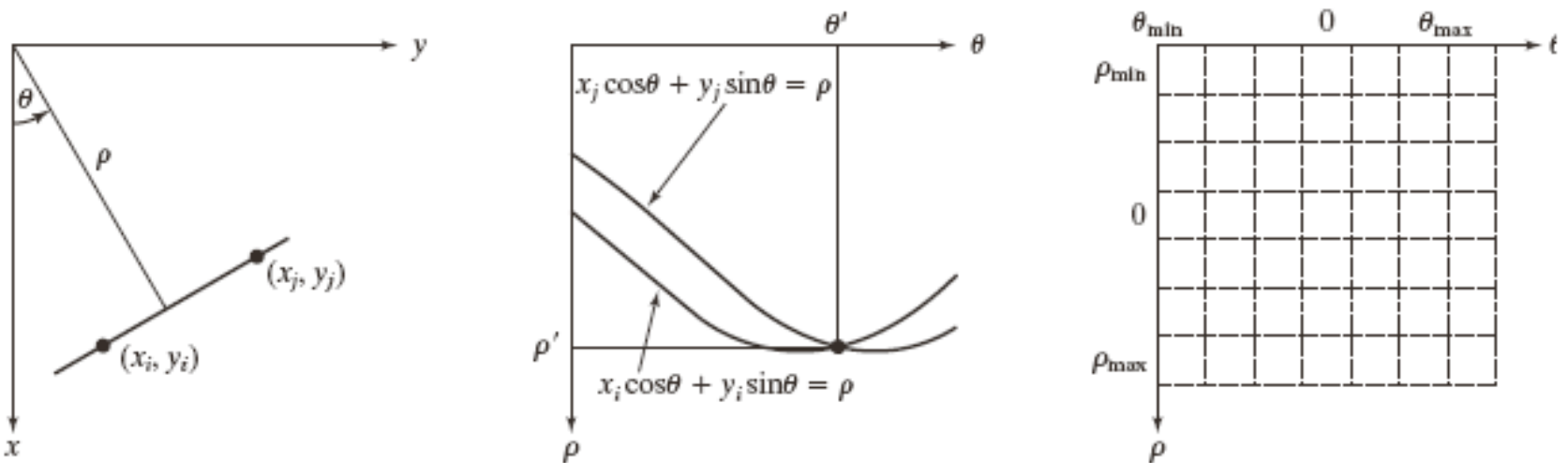
a b

**FIGURE 11.8**  
(a)  $xy$ -plane.  
(b) Parameter space.

# Concept

- The parameter-space lines corresponds to all point  $(x_k, y_k)$  in the  $xy$  plane could be plotted.
- Lines in the plane could be found by identifying points in parameter space where large number of lines intersect (  $b = -x_i a + y_i$  )
- However,  $a$  (slope of the line) can approach infinity when we approach the vertical direction
- One way around is to use:  $x \cos \theta + y \sin \theta = \rho$

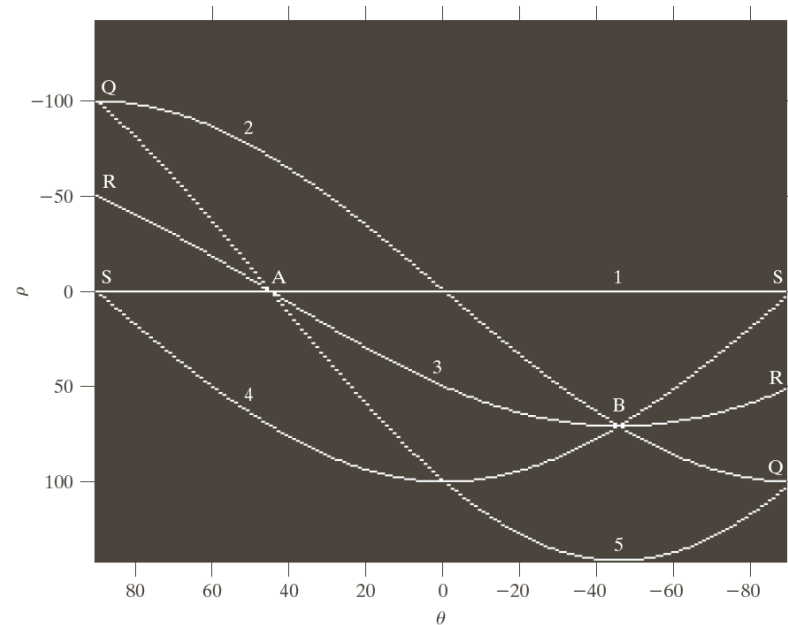
# Sinusoidal curves



a b c

**FIGURE 11.9** (a) Parameterization of lines in the  $xy$ -plane. (b) Sinusoidal curves in the  $\rho\theta$ -plane; the point of intersection,  $(\rho', \theta')$ , corresponds to the parameters of the line joining  $(x_i, y_i)$  and  $(x_j, y_j)$ . (c) Division of the  $\rho\theta$ -plane into accumulator cells.

# Example



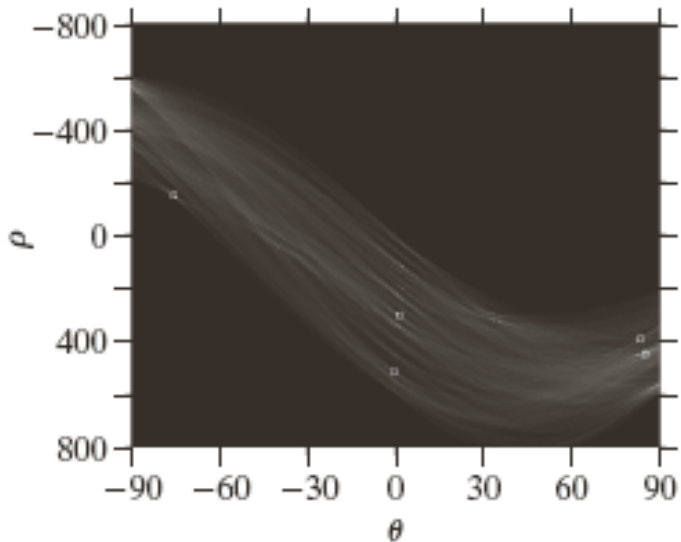
a  
b

**FIGURE 10.33**

(a) Image of size  $101 \times 101$  pixels, containing five points.  
 (b) Corresponding parameter space. (The points in (a) were enlarged to make them easier to see.)



# Hough Transform Peak Detection

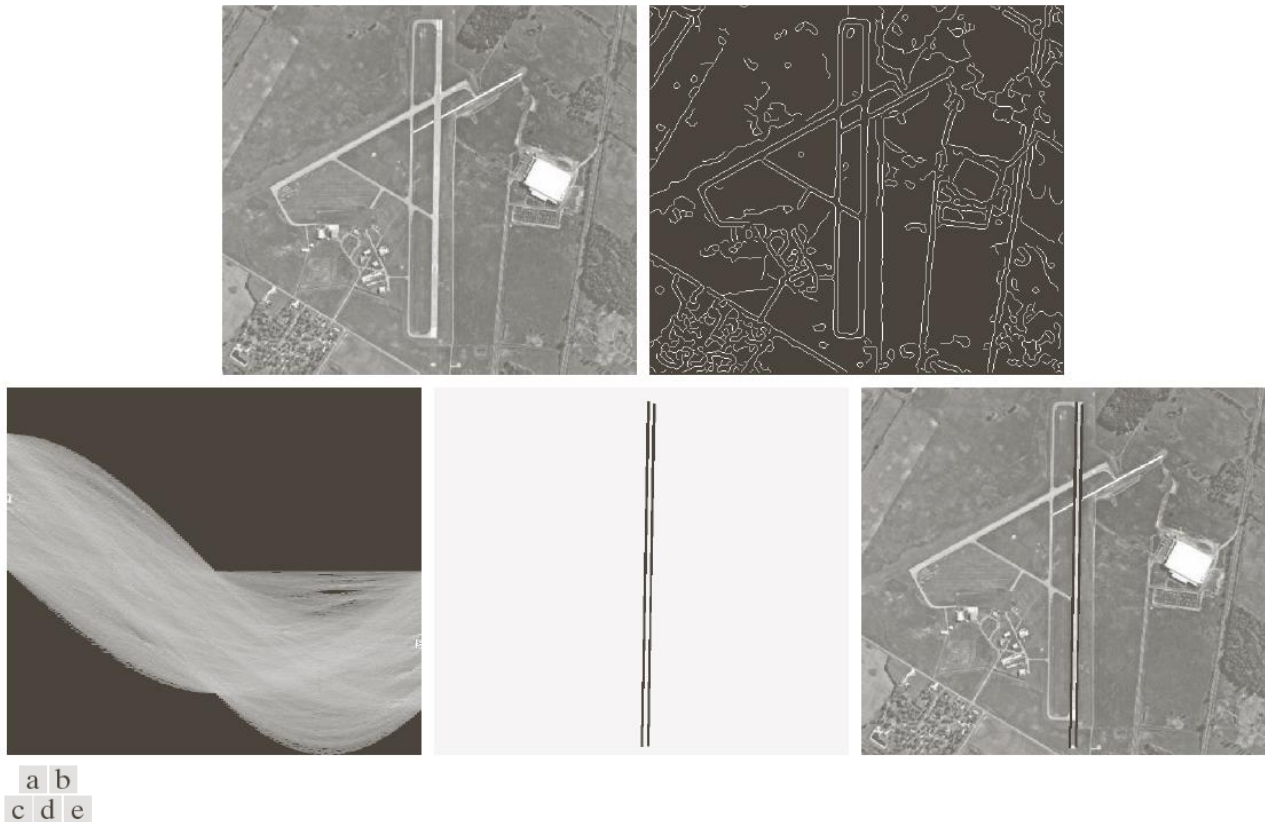


a b

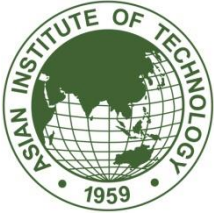
**FIGURE 11.11**  
(a) Hough transform with five peak locations selected.  
(b) Line segments (in bold) corresponding to the Hough transform peaks.



# Example



**FIGURE 10.34** (a) A  $502 \times 564$  aerial image of an airport. (b) Edge image obtained using Canny's algorithm. (c) Hough parameter space (the boxes highlight the points associated with long vertical lines). (d) Lines in the image plane corresponding to the points highlighted by the boxes). (e) Lines superimposed on the original image.



# OpenCV

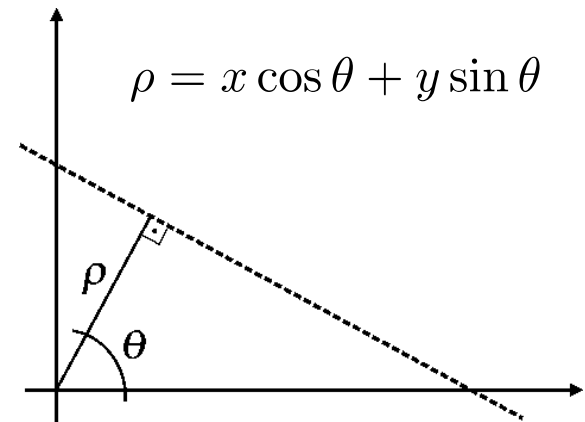
```
void cv::HoughLines(  
    cv::InputArray  image,           // Input single channel image  
    cv::OutputArray lines,          // N-by-1 two-channel array  
    double          rho,             // rho resolution (pixels)  
    double          theta,          // theta resolution (radians)  
    int             threshold,       // Unnormalized accumulator threshold  
    double          srn              // rho refinement (for MHT)  
    double          stn              // theta refinement (for MHT)  
);
```





# The Hough transform

- Coordinate conversion
  - Describing lines using the equation  $y = ax + b$  (where  $a$  represents the gradient) poses a problem, though, since vertical lines have infinite gradient.
    - This limitation can be circumvented by using the *normal representation* of a line, which consists of two parameters:  $\rho$  and  $\theta$ .
    - In this new representation, vertical lines will have  $\theta = 0$ .
    - It is common to allow  $\rho$  to have negative values, therefore restricting  $\theta$  to the range  $-90^\circ < \theta \leq 90^\circ$ .



# The Hough transform

- Under the new set of coordinates, the Hough transform can be implemented as follows:
  1. Create a 2D array corresponding to a discrete set of values for  $\rho$  and  $\theta$ . Each element in this array is often referred to as an *accumulator cell*.
  2. For each pixel  $(x,y)$  in the image and for each chosen value of  $\theta$ , compute  $x \cos \theta + y \sin \theta$  and write the result in the corresponding position  $(\rho, \theta)$  in the accumulator array.
  3. The highest values in the  $(\rho, \theta)$  array will correspond to the most relevant lines in the image.



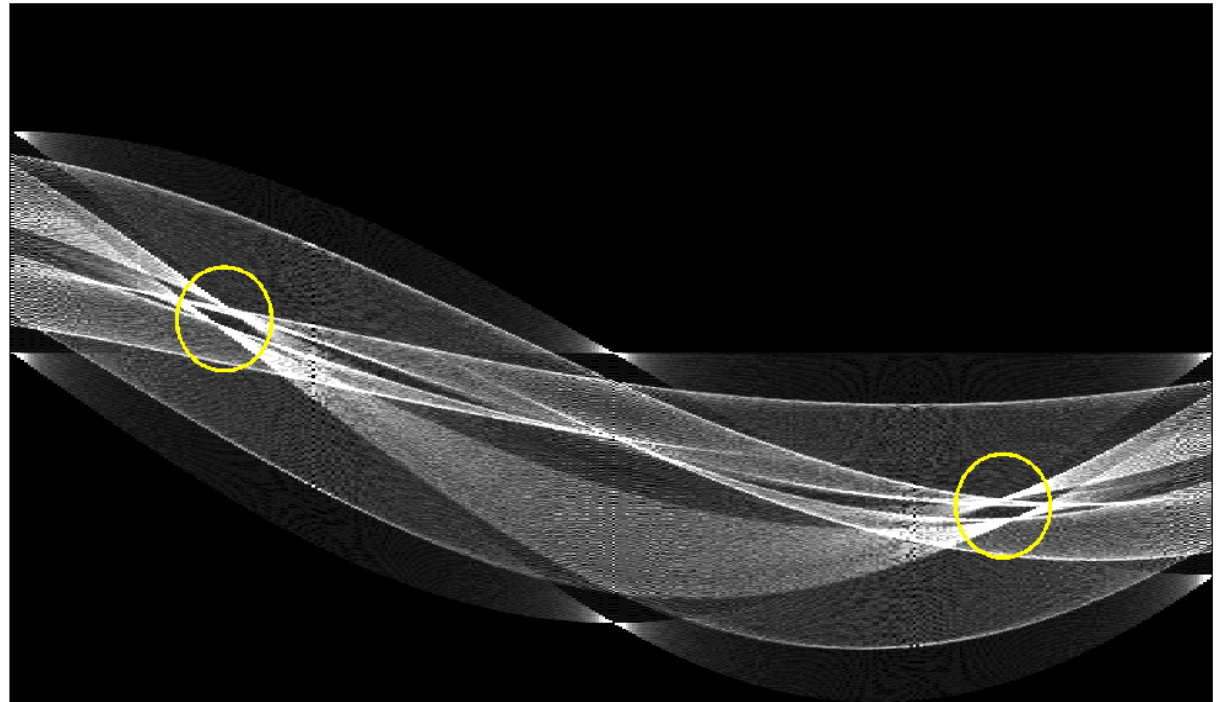
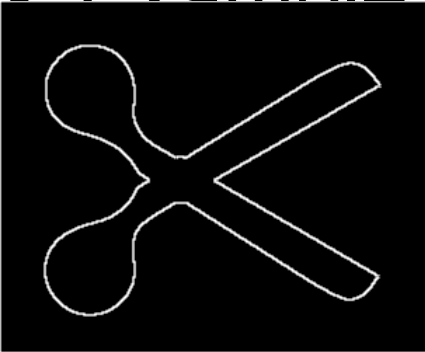


```
int main(int argc, char** argv){
    Mat src = imread("d:/image/checkerboard.png",
IMREAD_GRAYSCALE);
    Mat dst, cdst;
    Canny(src, dst, 50, 200, 3);
    cvtColor(dst, cdst, CV_GRAY2BGR);
    vector<Vec4i> lines;
    HoughLinesP(dst, lines, 1, CV_PI / 180, 50, 50, 10);
    for (size_t i = 0; i < lines.size(); i++){
        Vec4i l = lines[i];
        line(cdst, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 0, 255), 3,
CV_AA);
    }
    imshow("source", src);
    imshow("detected lines", cdst);
    waitKey(0);
}
```



# The Hough transform

- Example 14.6

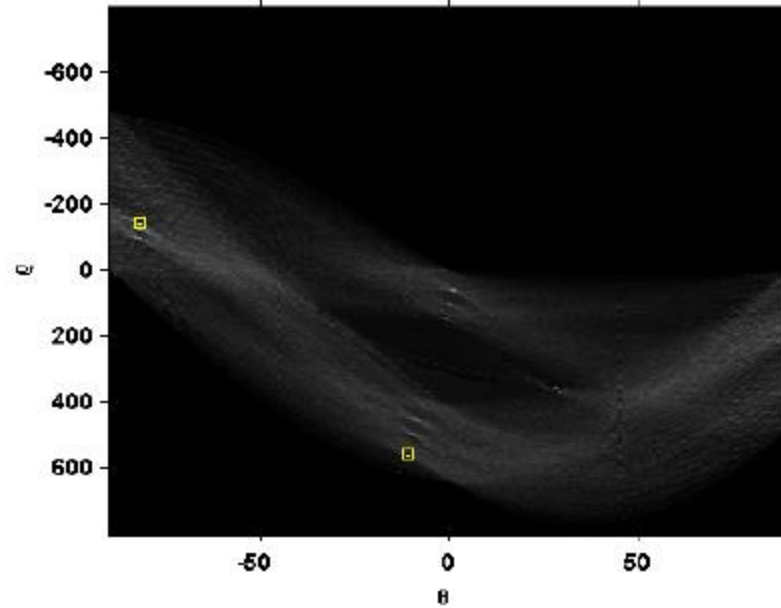


# The Hough transform

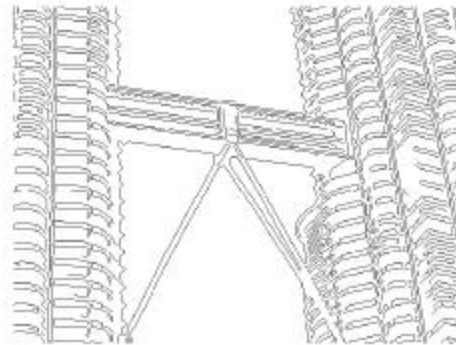
- In MATLAB:
  - The IPT also includes two useful companion functions for exploring and plotting the results of Hough Transform calculations:
    - **houghpeaks**: identifies the  $k$  most salient peaks in the Hough transform results, where  $k$  is passed as a parameter.
    - **houghlines**: draws the lines associated with the highest peaks on top of the original image.



# The Hough transform



(a)



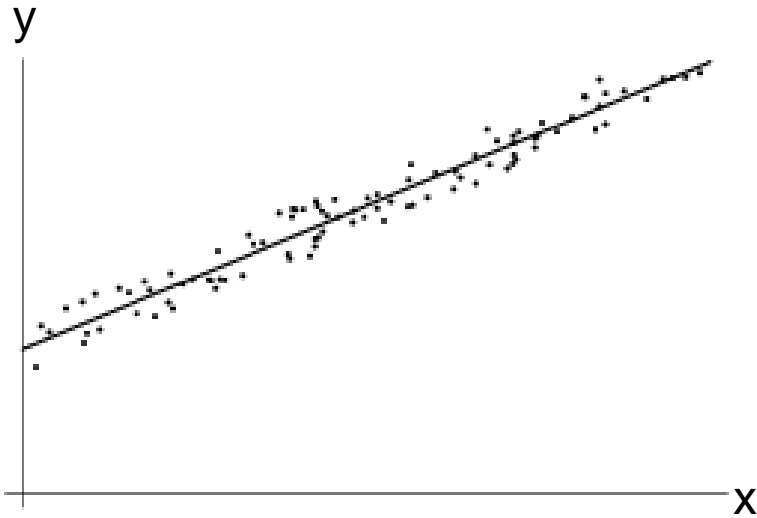
(b)



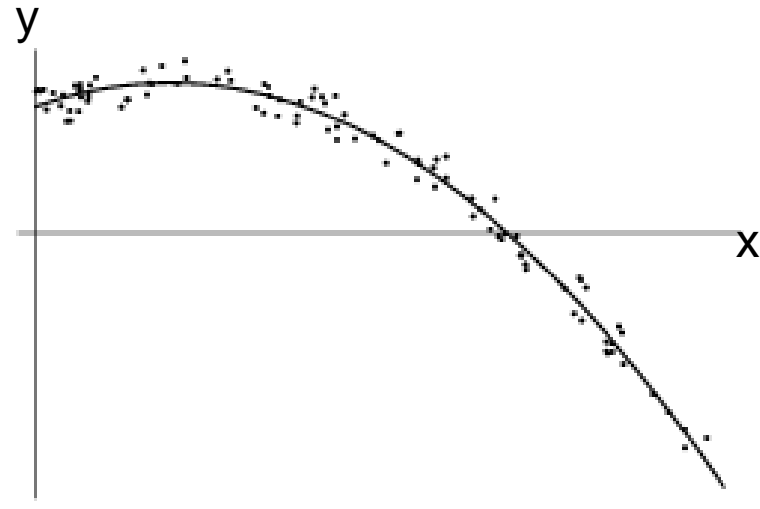
(c)



# Line/Polynomial Fitting



$$y=ax+b$$



$$y=ax^2+bx+c$$

MATLAB function:  $P = \text{POLYFIT}(X, Y, N)$

↓  
Polynomial of degree N

# Analytical vs. Numerical Solutions

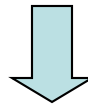
$$f(x) = x^2 - 3x + 2$$



Analytical

Two roots:  $x_1 = 1, x_2 = 2$

Numerical  $f(x) = x^2 - 3x + 2 = 0 \Rightarrow x = \frac{x^2 + 2}{3} = g(x)$

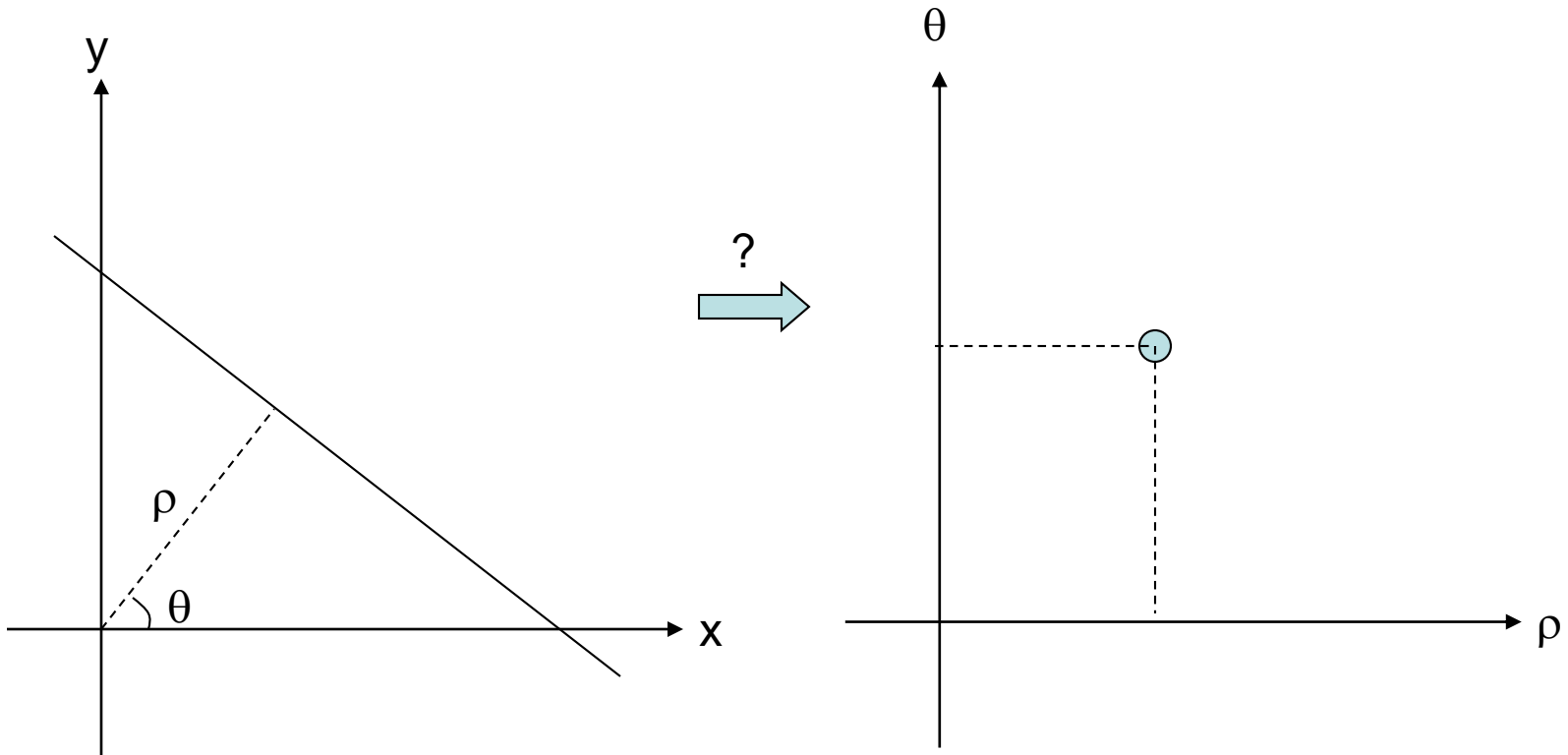


successive substitution:

$$x_{i+1} = \frac{x_i^2 + 2}{3}$$

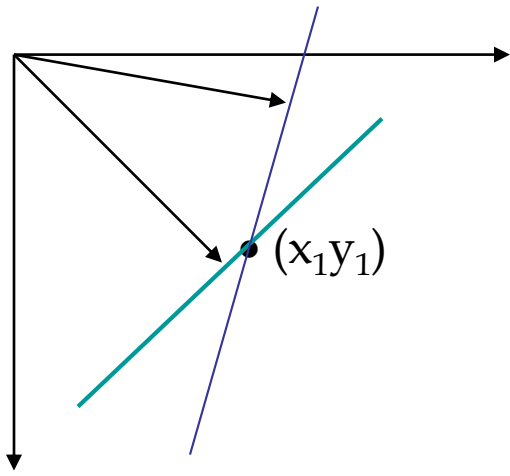


# Basic Idea Behind

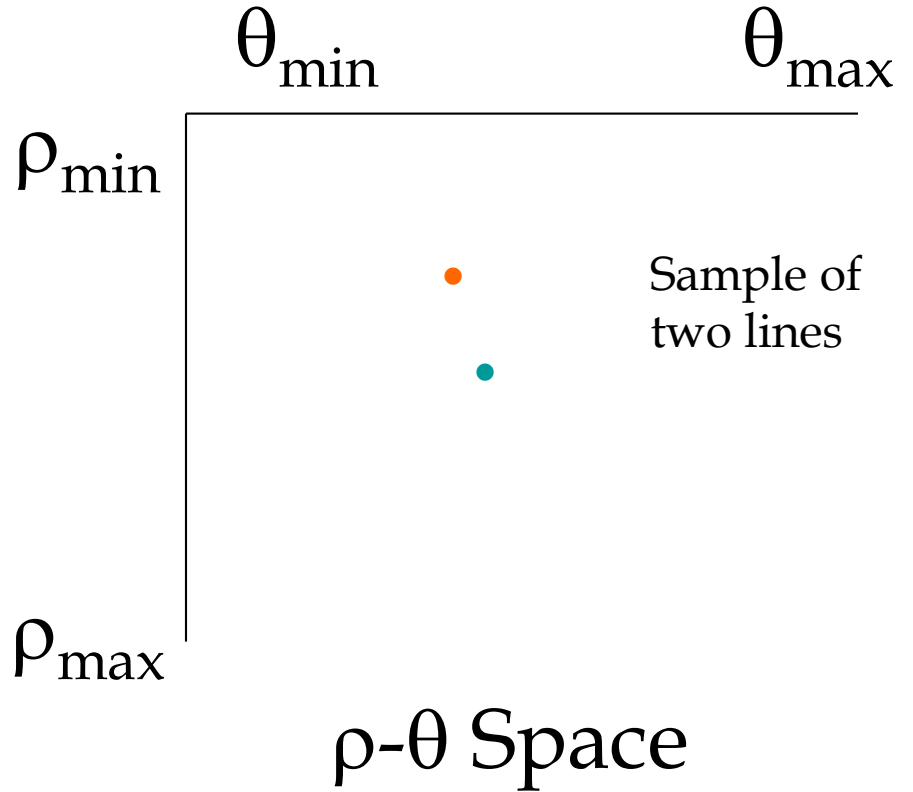


Question: Can we find a transform that maps a line to a point?

# Radon/Hough Transform: Mathematics\*



Two of an infinite number  
of lines through point  $(x_1, y_1)$



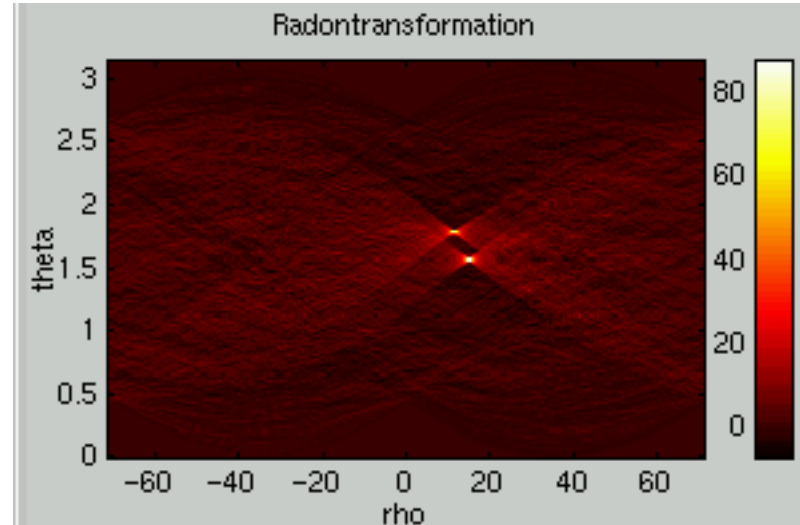
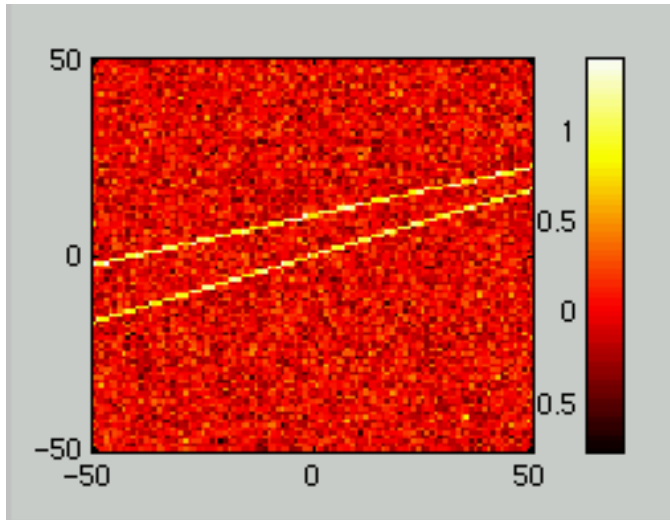
$$\check{g}(\rho, \theta) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) \delta(\rho - x \cos \theta - y \sin \theta) dx dy$$

# Generalizations of Hough Transform (from 2D to N-D)



Analytic Form	Parameters	Equation
Line	$\rho, \theta$	$x\cos\theta + y\sin\theta = \rho$
Circle	$x_0, y_0, \rho$	$(x-x_0)^2 + (y-y_0)^2 = r^2$
Parabola	$x_0, y_0, \rho, \theta$	$(y-y_0)^2 = 4\rho(x-x_0)$
Ellipse	$x_0, y_0, a, b, \theta$	$(x-x_0)^2/a^2 + (y-y_0)^2/b^2 = 1$

# Radon/Hough Transform: Image Examples



Conclusion: Line detection can be implemented by point (peak) detection in the Radom/Hough transform domain

# Line Detection from Real-world Images



Gray Scale Image

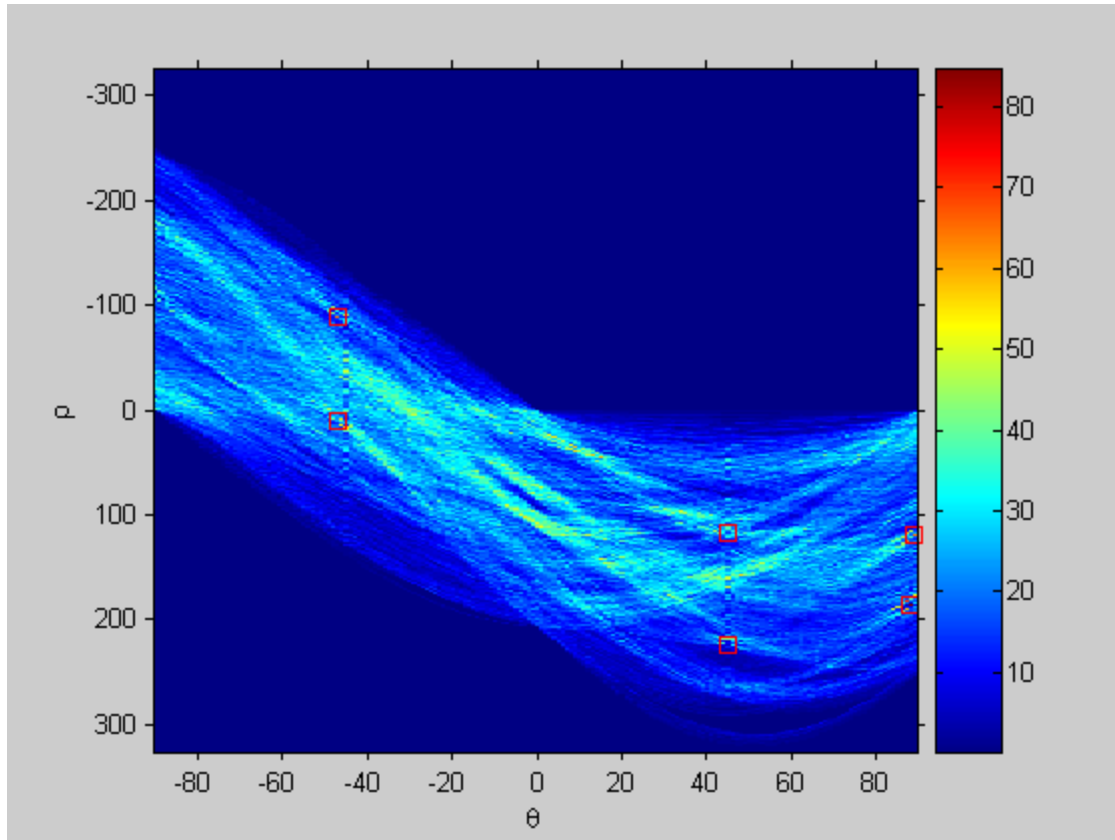
Canny Edge



Edge Image (Canny)



# Peak Finding



# Application in Computer

(From Jain's Fig 10.1)

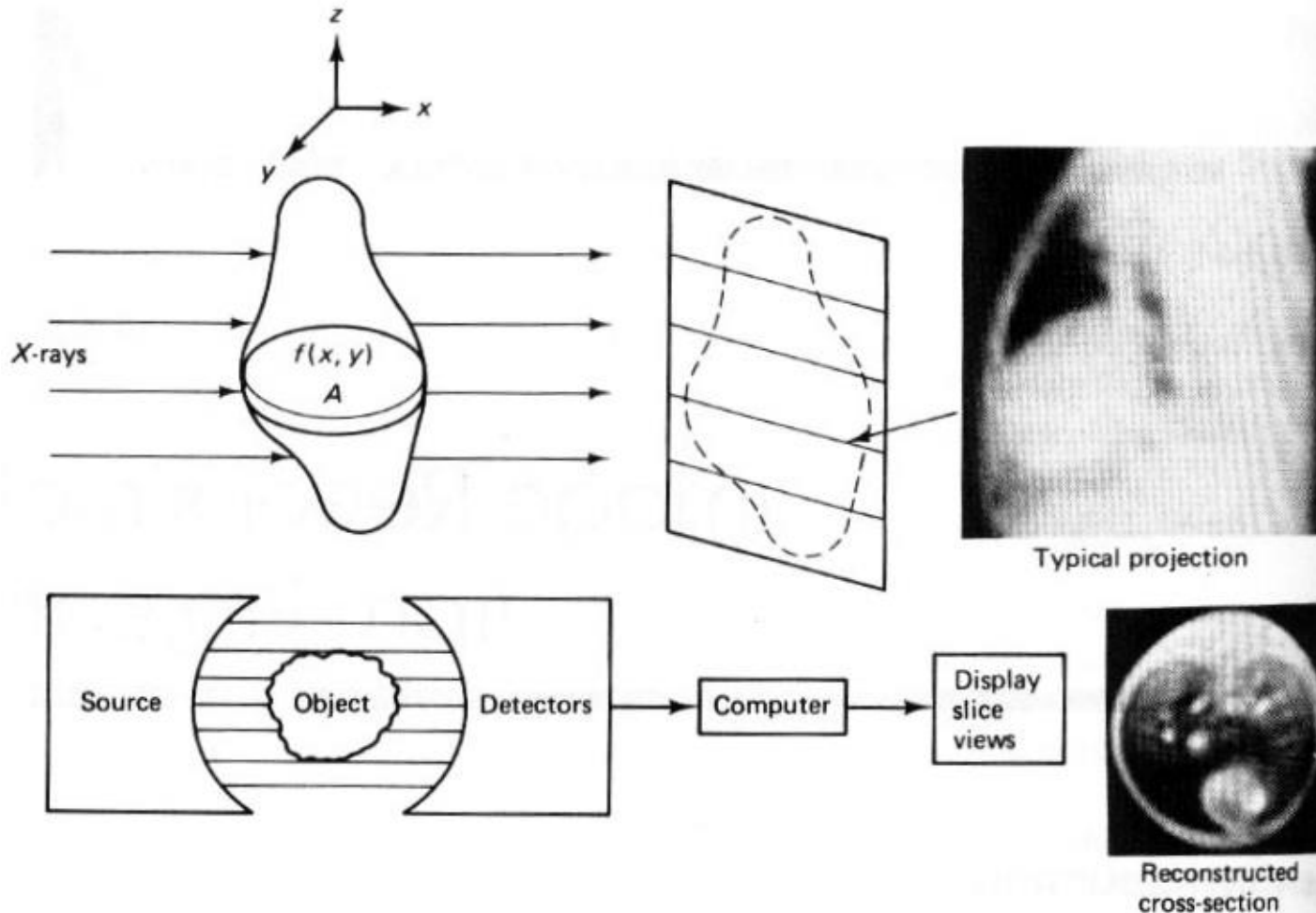


Figure 10.1 An X-ray CT scanning system.

# Beyond Edge Detection

- Edges are among the earliest primitives that have been studied in computer vision, but their definition remains fuzzy
- There are many other primitives that can be more rigorously defined or at least conceptually easier to define
  - **Corner, line, circle, ellipse**, square, triangle, ...
- The ultimate vision tasks involve the detection of general objects
  - **Face, pedestrian, vehicle**, mouth, eyes, door



A decorative graphic in the top-left corner consisting of a blue square above a grid of smaller squares in various colors.

# Two Paradigms

- **Model-based** approaches
  - Build an explicit model to characterize the objects we want to detect
  - Suitable for the class of simple objects for which good models are relatively easy to find
  - Examples: corner/line/circle detection
- **Data-driven** (machine learning) approaches
  - Provide a training set (e.g., manually detected results) and detector works like a black box
  - Suitable for the class of complex objects for which good models are NOT easy to find
  - Examples: neural network, support vector machine



# Questions?

