Image Processing

Segmentation and Edge Detection

Yeong Jun Koh
Department of Computer Science & Engineering
Chungnam National University

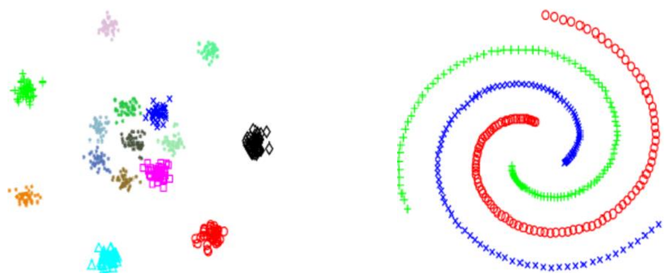
Objectives

- Image segmentation
  - Recent segmentation
    - Semantic segmentation
    - Instance segmentation
    - Video object segmentation
  - Basic concept about image segmentation
    - Thresholding

- Edge detection
  - First-order derivative, Second-order derivative
  - Canny edge
  - Understanding the Hough Transform

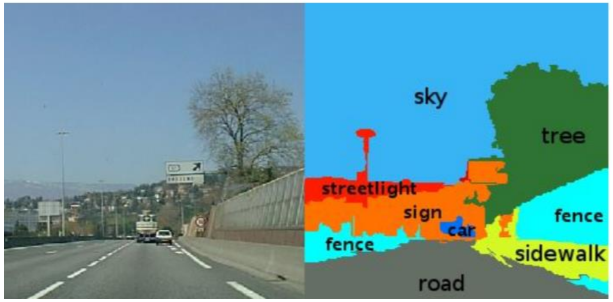
Segmentation

- Divide data into meaningful segments



Point clustering

Image segmentation



Semantic segmentation

Video object segmentation

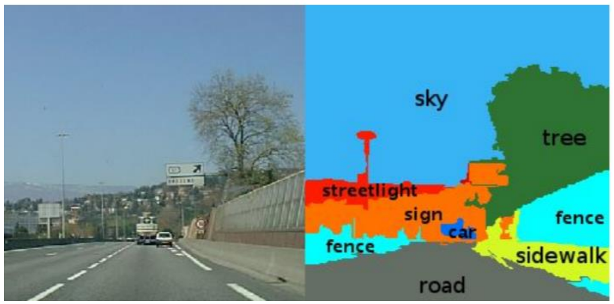
Image Segmentation

- Segmentation method (before deep learning)
  - K-means clustering
  - Mean shift
  - Normalized-cut
  - Graph cut
  - Random-walk
  - Markov random field (MRF) optimization

[Felzenszwalb and Huttenlocher 2004]

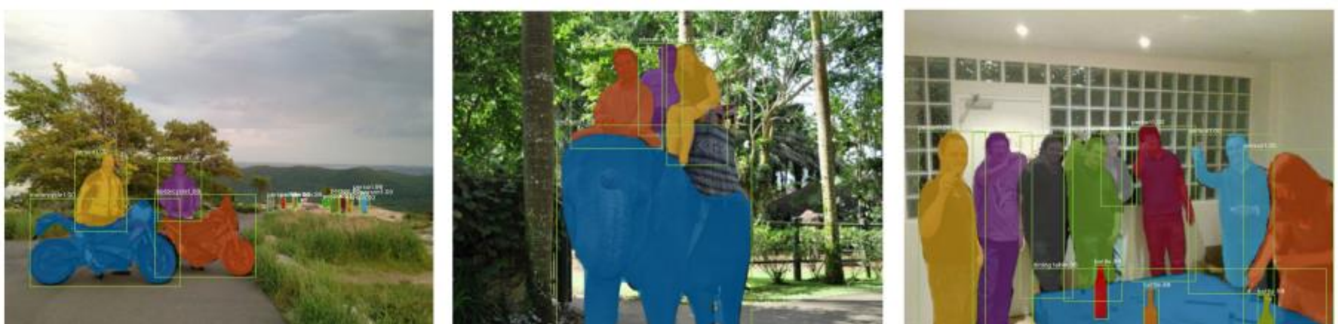
Image Segmentation

- Semantic segmentation



- Video object segmentation

- Instance segmentation



Semantic Segmentation

- Label each pixel in the image with a category label
- Don't differentiate instances, only care about pixels

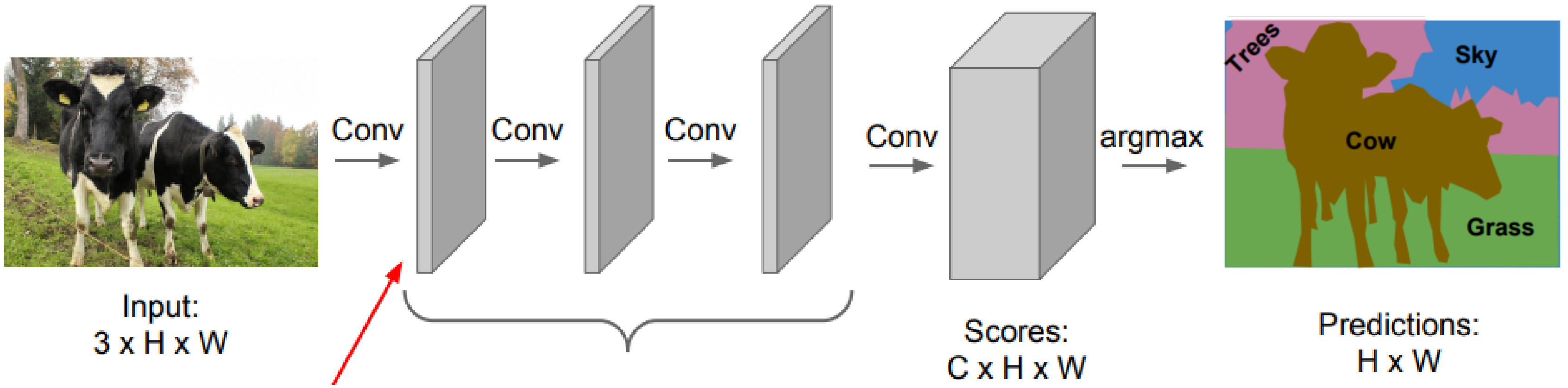
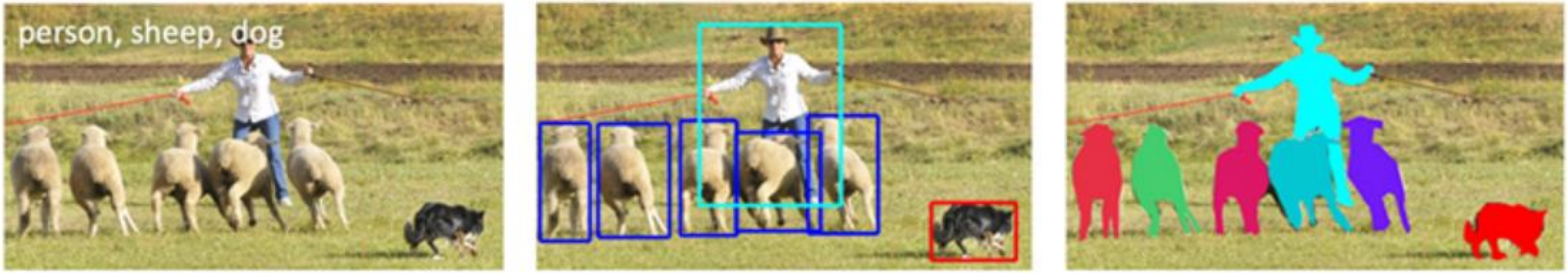


image from Stanford's cs231n course slides

Instance Segmentation

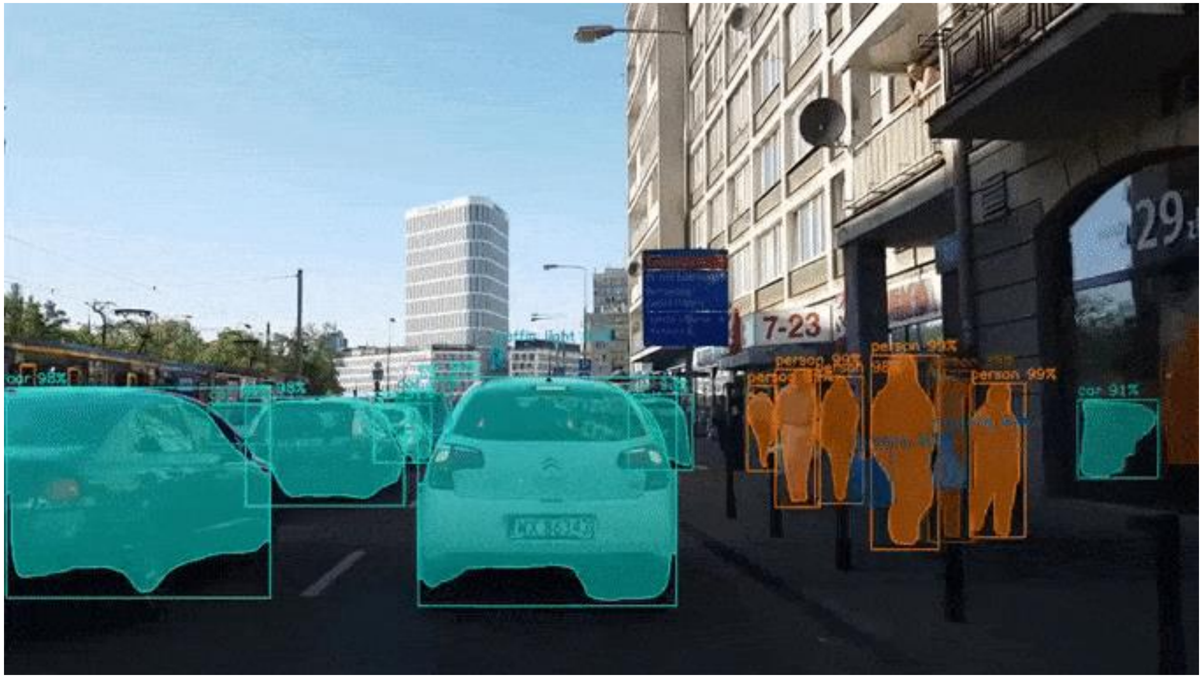
- Three classic tasks related to objects in computer vision
  - Classification
  - Object Detection
  - Instance Segmentation



(a) classification (b) detection (c) segmentation

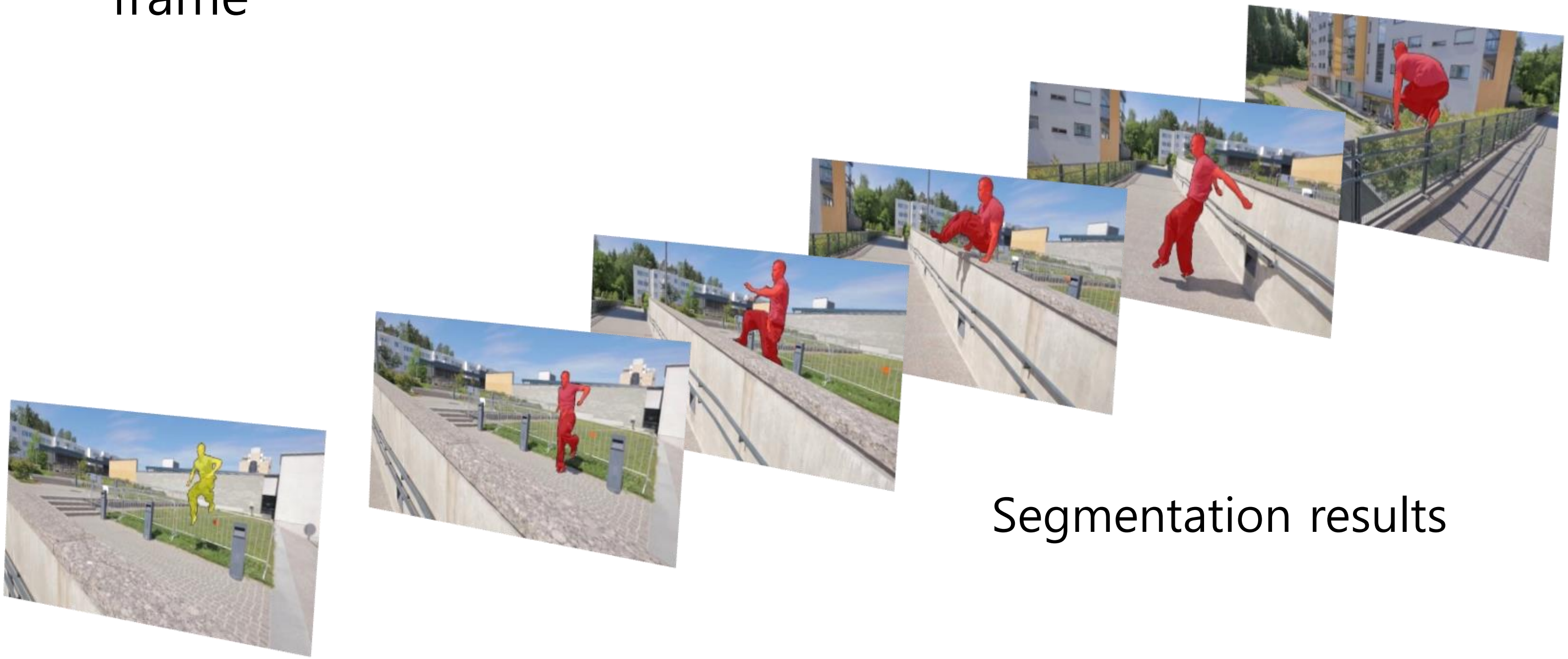
Instance Segmentation

- Three classic tasks related to objects in computer vision
    - Classification
    - Object Detection
    - Instance Segmentation



Video Object Segmentation

- Semi-supervised video object segmentation
  - Take user annotations for target object in the first frame
  - Track and segment the annotated objects in every consecutive frame



Segmentation results

Annotated target object
in the first frame

Image Segmentation

- Image segmentation
  - Divide an image into non-overlapping regions (objects) with similar information

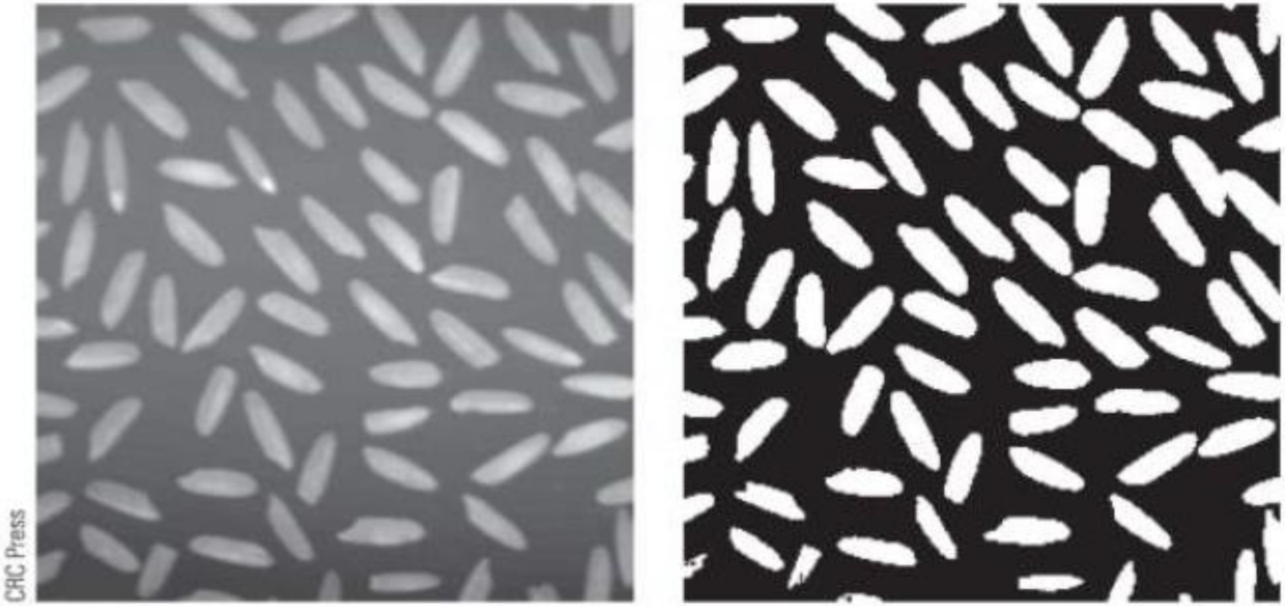
- The classic approaches
  - Image thresholding

Thresholding

- Single threshold

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

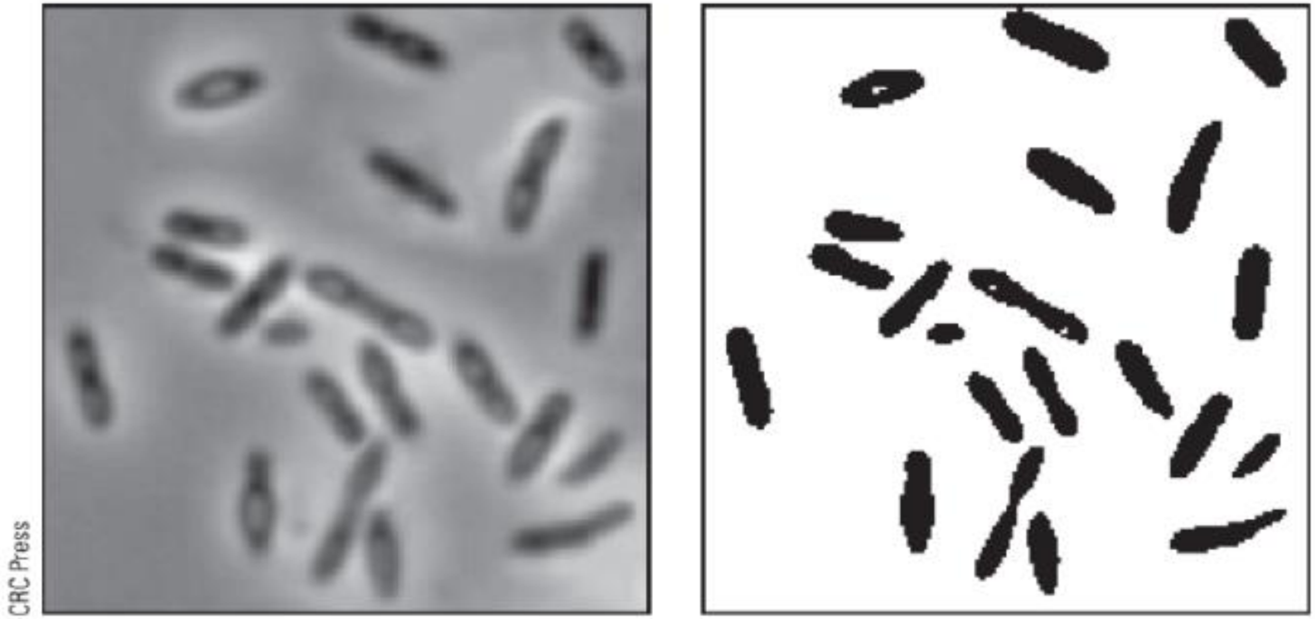
```
>> r=imread('rice.tif');
>> imshow(r),figure,imshow(r>110)
```



Thresholding

- Single threshold

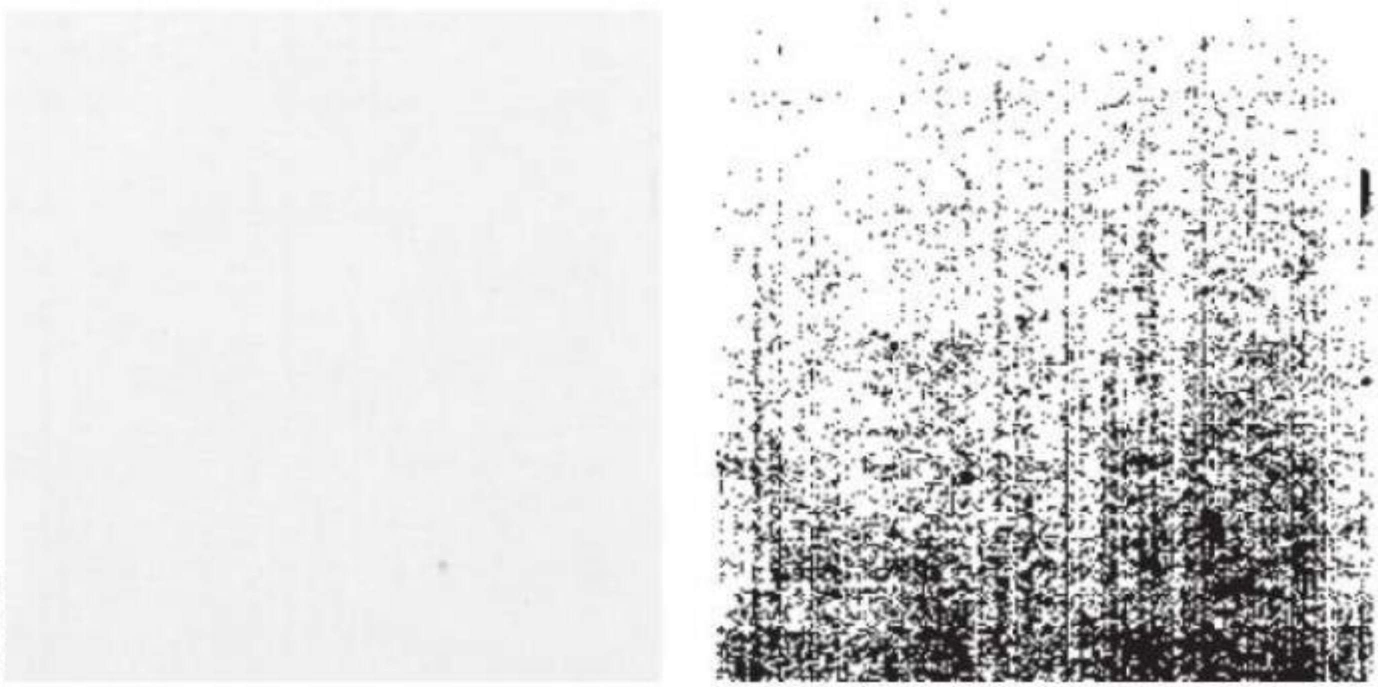
```
>> b=imread('bacteria.tif');
>> imshow(b),figure,imshow(b>100)
```



Thresholding

- Shows the hidden aspects of an image

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

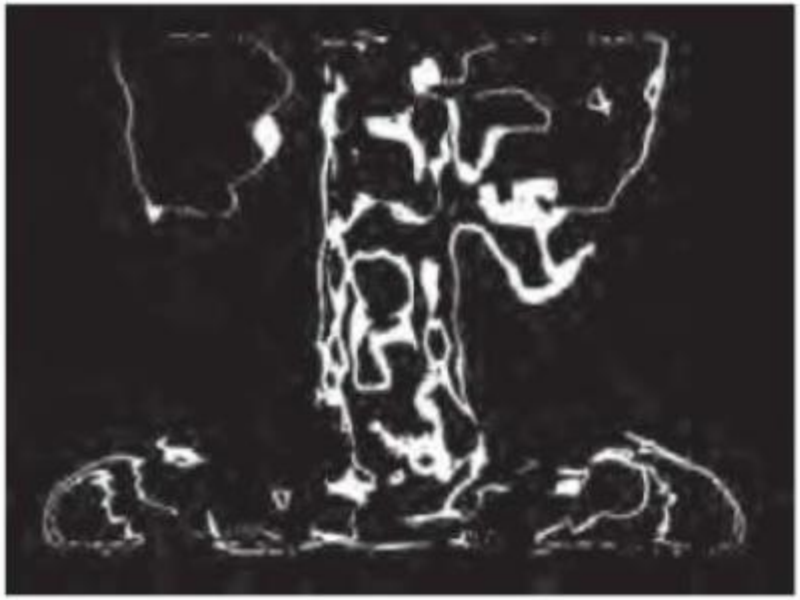


Mathworks

Double Thresholding

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

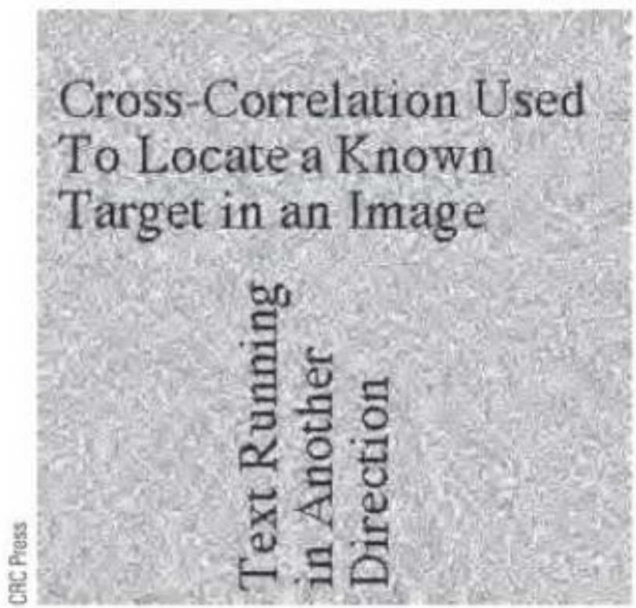
```
>> [x,map]=imread('spine.tif');
>> s=ind2gray(x,map);
>> imshow(s),figure,imshow(s>115 & s<125)
```



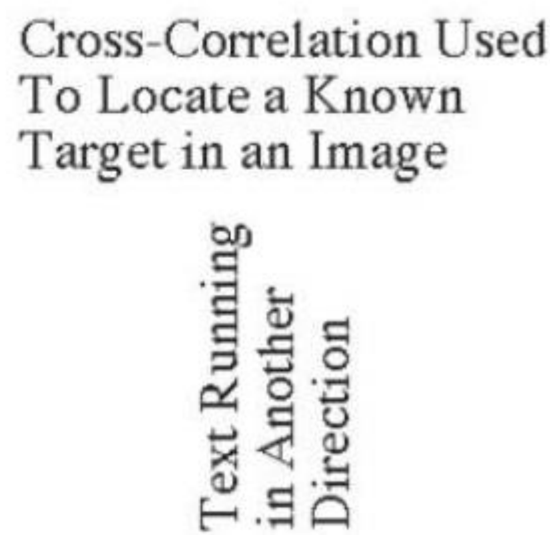
Simple Application using Thresholding

- Removing a background noise from text

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

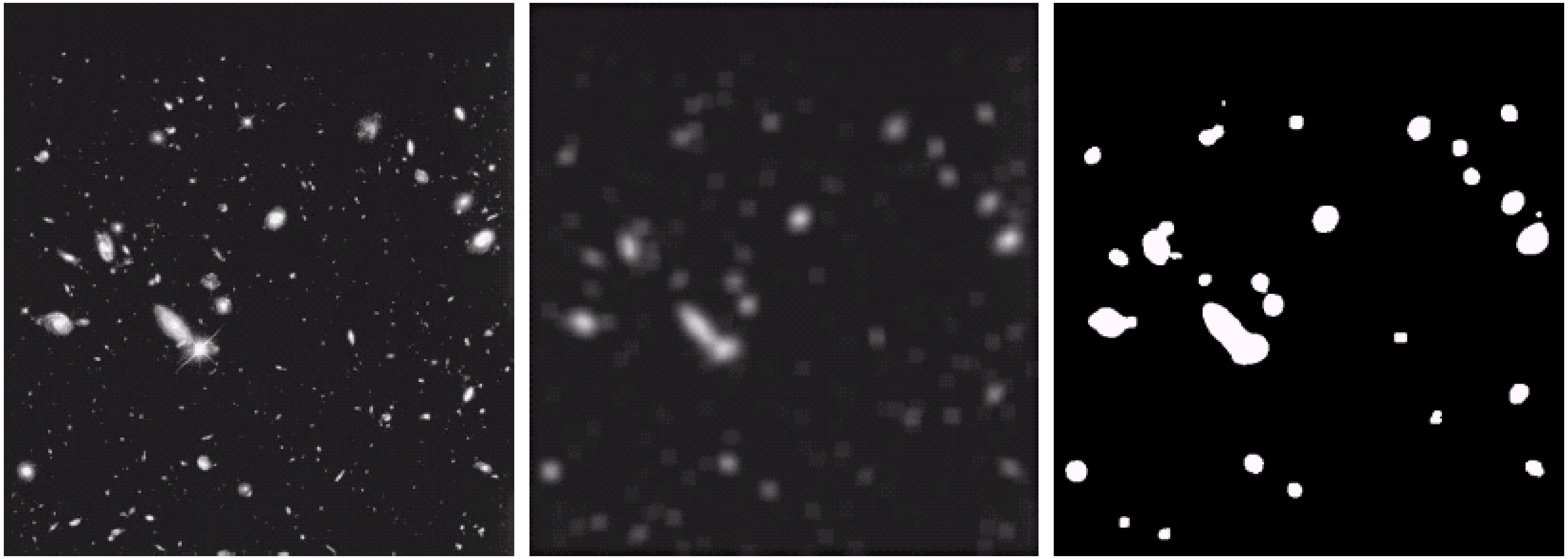



```
>> imshow(tr>100)
```



Simple Application using Thresholding

- Smoothing filter can be used before thresholding
  - Remove small objects and noise



a b c

FIGURE 3.36 (a) Image from the Hubble Space Telescope. (b) Image processed by a 15 × 15 averaging mask. (c) Result of thresholding (b). (Original image courtesy of NASA.)

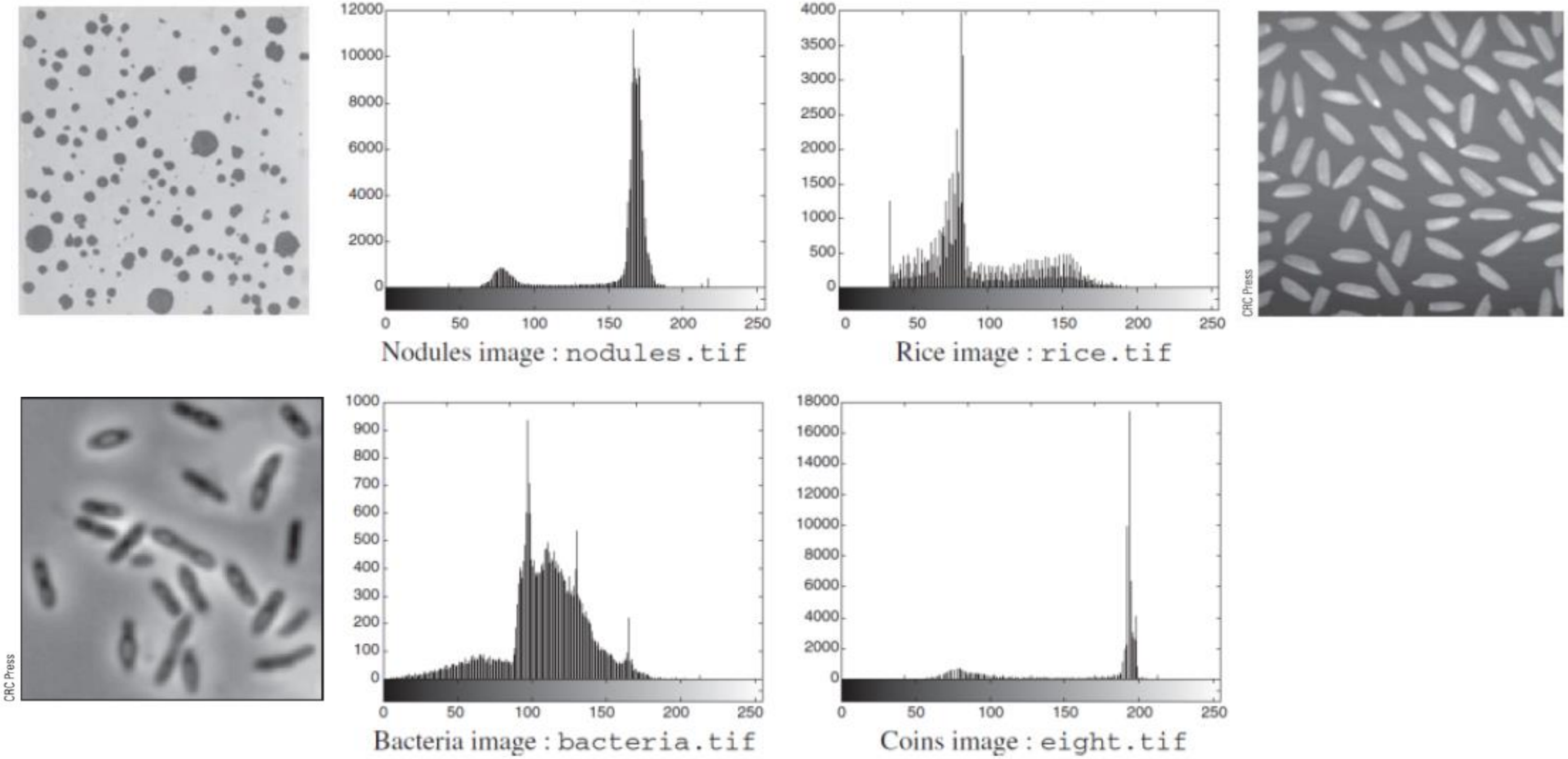
How to predict appropriate threshold?

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

n: Original image   n1: Threshold too low   n2: Threshold too high

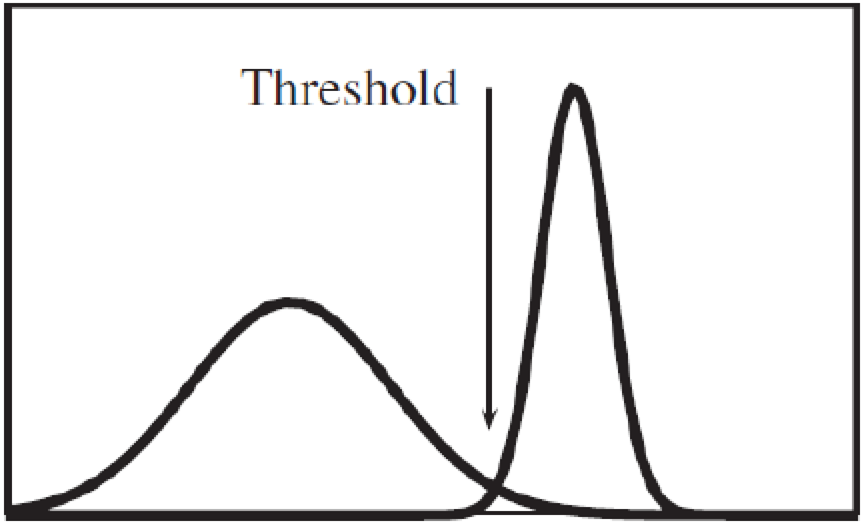
How to predict appropriate threshold?

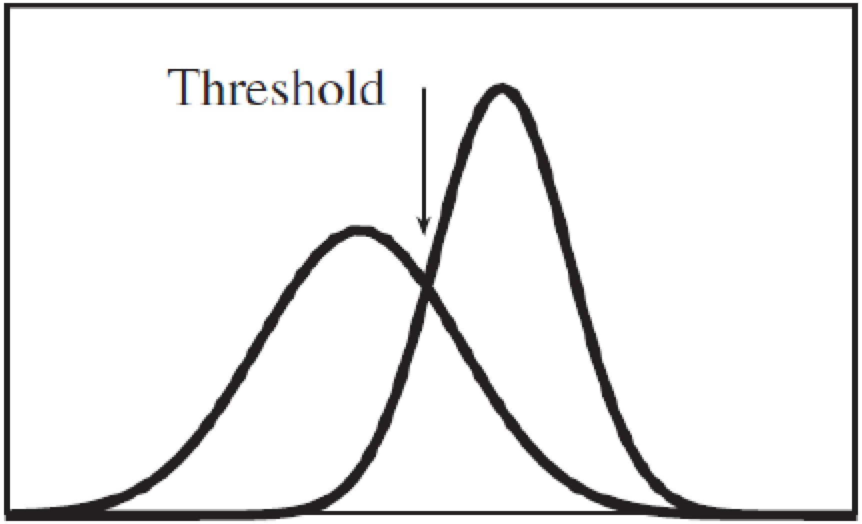
- Watch out the histogram
  - The threshold can be between two dominant distributions



How to predict appropriate threshold?

- Watch out the histogram
  - The threshold can be between two dominant distributions





Optimum Global Thresholding

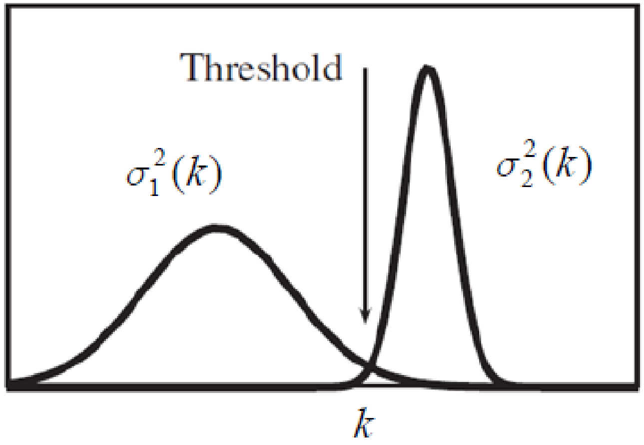
- Otsu's Method
  - Exhaustively search for the threshold that minimizes the intra-class variance (the variance within the class)
    - Minimize the intra-class (within-class) variance: a weighted sum of variances of the two classes

      ||
    - Maximize the inter-class (between-class) variance
  - Variance of bimodal signal

    = Within-class variance + Between-class variance

$m_1(k)$ or $m_2(k)$: mean at region 1 or region 2

$\sigma_1(k)$ or $\sigma_2(k)$: variance at region 1 or region 2



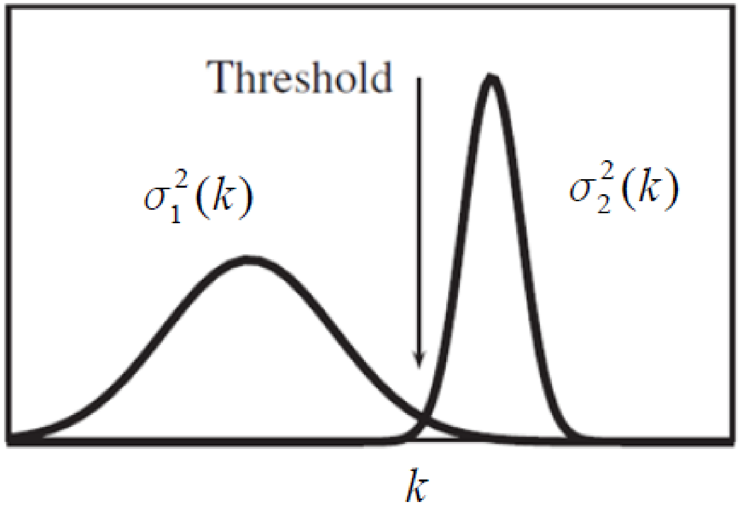
Optimum Global Thresholding

$M \times N$ image with L intensity levels
n_i: the number of pixels with intensity i

$$p_i = n_i / MN \quad \text{and} \quad \sum_{i=0}^{L-1} p_i = 1$$

Using threshold k, segment an image into two regions

$$C_1 \rightarrow [0, k],\ C_2 \rightarrow [k+1, L-1]$$



Optimum Global Thresholding

For each region, the following property holds.

$$q_1(k) = \sum_{i=0}^{k} p(i) \qquad q_2(k) = \sum_{i=k+1}^{L-1} p(i)$$

$$m_1(k) = \frac{\sum_{i=0}^{k} ip(i)}{\sum_{i=0}^{k} p(i)} = \frac{1}{q_1(k)} \sum_{i=0}^{k} ip(i) \qquad m_2(k) = \frac{\sum_{i=k}^{L-1} ip(i)}{\sum_{i=k+1}^{L-1} p(i)} = \frac{1}{q_2(k)} \sum_{i=k}^{L-1} ip(i)$$

$$\sigma_1^2(k) = \frac{1}{q_1(k)} \sum_{i=0}^{k} [i - m_1(k)]^2 p(i) = \frac{1}{q_1(k)} \sum_{i=0}^{k} i^2 p_i - m_1^2(k)$$

$$\sigma_2^2(k) = \frac{1}{q_2(k)} \sum_{i=k+1}^{L-1} [i - m_2(k)]^2 p(i) = \frac{1}{q_2(k)} \sum_{i=k+1}^{L-1} i^2 p(i) - m_2^2(k)$$

For an entire image,

$$m_G = \sum_{i=0}^{L-1} ip(i), \qquad \sigma_G^2 = \sum_{i=0}^{L-1} [i - m_G]^2 p(i)$$

$$\boxed{\begin{aligned} &q_1(k) + q_2(k) = 1 \\ &q_1(k)m_1(k) + q_2(k)m_2(k) = m_G \end{aligned}}$$

Optimum Global Thresholding

- Let's define the *within-class variance* as follows

$$\sigma_W^2(k) = q_1(k)\sigma_1^2(k) + q_2(k)\sigma_2^2(k)$$

- Then, we can simply seek the optimal threshold by minimizing the within-class variance

$$k_{opt} = \arg\min_k \sigma_W^2(k)$$

- We can also exploit the between-class variance $\sigma_B^2(k)$

$$k^* = \arg\max_k \sigma_B^2(k)$$

- Analyze relationship between the within-class variance and the between-class variance

Optimum Global Thresholding

After some algebra, the following equation is derived

$$\sigma_G^2 = \sigma_W^2(k) + \sigma_B^2(k)$$

$$\sigma_W^2(k) = q_1(k)\sigma_1^2(k) + q_2(k)\sigma_2^2(k)$$

$$\boxed{\sigma_B^2(k) = q_1(k)[m_1(k) - m_G]^2 + q_2(k)[m_2(k) - m_G]^2}$$

Proof)

$$\sigma_W^2(k) + \sigma_B^2(k)$$

$$= \sum_{i=0}^{k} i^2 p(i) - q_1 m_1^2 + \sum_{i=k+1}^{L-1} i^2 p(i) - q_2 m_2^2 + q_1(m_1^2 + m_G^2 - 2m_1 m_G) + q_2(m_2^2 + m_G^2 - 2m_2 m_G)$$

$$= \sum_{i=0}^{L-1} i^2 p(i) + q_1 m_G^2 - 2q_1 m_1 m_G + q_2 m_G^2 - 2q_2 m_2 m_G \longleftarrow \boxed{q_1 + q_2 = 1}$$

$$= \sum_{i=0}^{L-1} i^2 p(i) + m_G^2 - 2m_G(q_1 m_1 + q_2 m_2) = \sum_{i=0}^{L-1} i^2 p(i) + m_G^2 - 2m_G^2 \longleftarrow \boxed{q_1 m_1 + q_2 m_2 = m_G}$$

$$= \sum_{i=0}^{L-1} i^2 p(i) - m_G^2 = \sigma_G^2$$

Optimum Global Thresholding

- Maximizing the between-class variance

$$\sigma_B^2(k) = q_1(k)[m_1(k) - m_G]^2 + q_2(k)[m_2(k) - m_G]^2$$
$$= q_1(k)q_2(k)[m_1(k) - m_2(k)]^2$$

Proof)

$$q_1[m_1 - m_G]^2 + q_2[m_2 - m_G]^2$$
$$= q_1[m_1 - q_1 m_1 - q_2 m_2]^2 + q_2[m_2 - q_1 m_1 - q_2 m_2]^2$$
$$= q_1 q_2^2 (m_1 - m_2)^2 + q_2 q_1^2 (m_1 - m_2)^2$$
$$= q_1 q_2 (m_1 - m_2)^2 (q_2 + q_1) = \boxed{q_1 q_2 (m_1 - m_2)^2}$$

Don't need to compute variance for each class

Optimum Global Thresholding

- Compute $\sigma_B^2(k) = q_1(k)q_2(k)[m_1(k) - m_2(k)]^2$ recursively
  - Initialization $q_1(0) = p_0, m_1(0) = 0$
  - Iteratively compute the mean using moving average

$$q_1(k+1) = q_1(k) + p(k+1)$$

$$m_1(k+1) = \frac{q_1(k)m_1(k) + (k+1)p(k+1)}{q_1(k+1)}$$

$$m_2(k+1) = \frac{(1-q_1(k))m_2(k) - kp(k)}{1-q_1(k+1)}$$

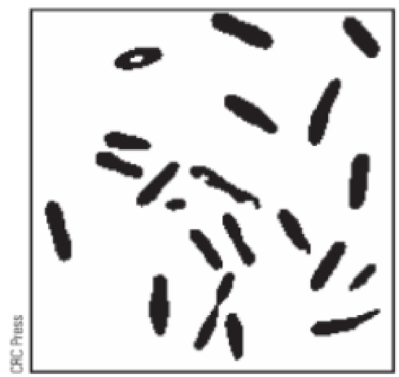
Otsu’s Method in MATLAB

```
>> 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))
```

Nodules

Rice



Bacteria

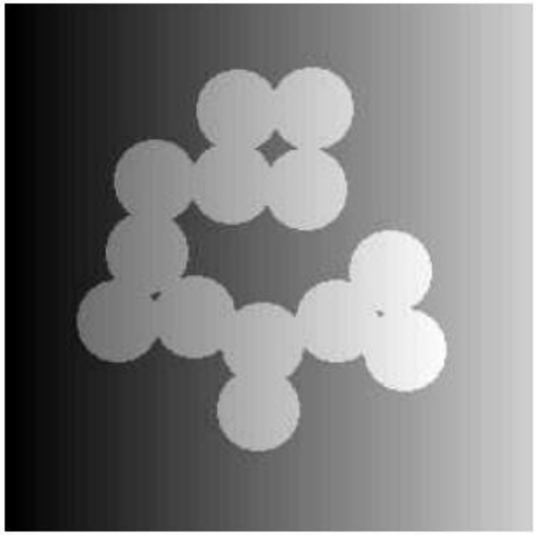
Coins

Adaptive Thresholding

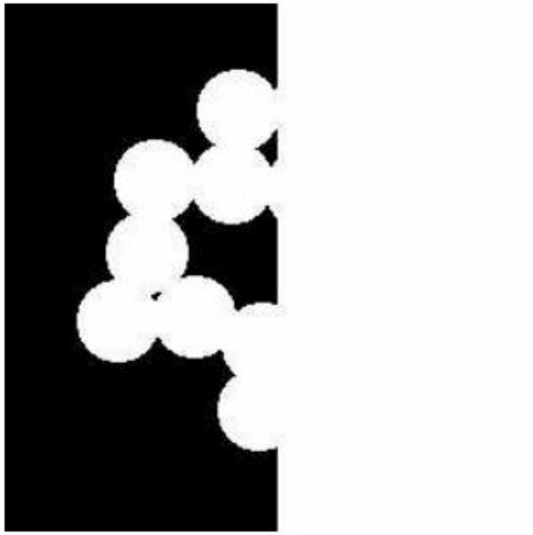
- Limitation of thresholding
  - The same threshold is used over an entire image

```
>> 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));
```

```
>> t=graythresh(c3)

t =

    0.4196

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



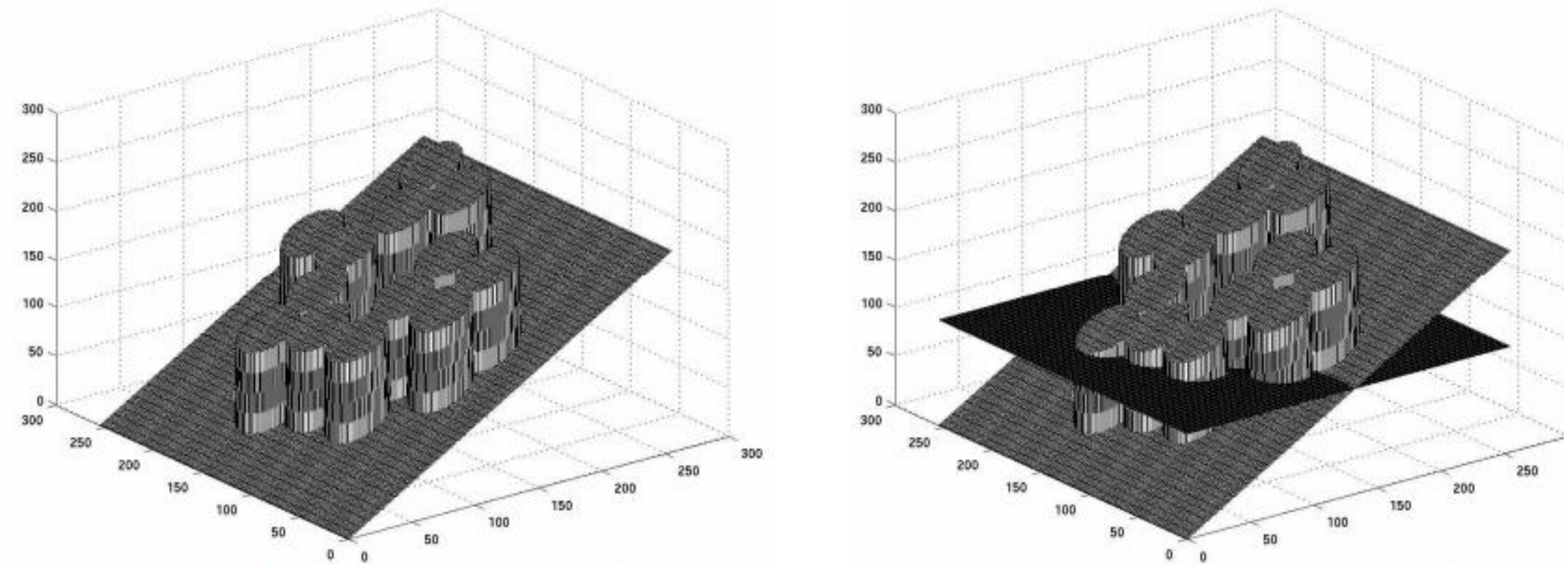
(a) Circles image: c3



(b) Thresholding attempt: ct

Adaptive Thresholding

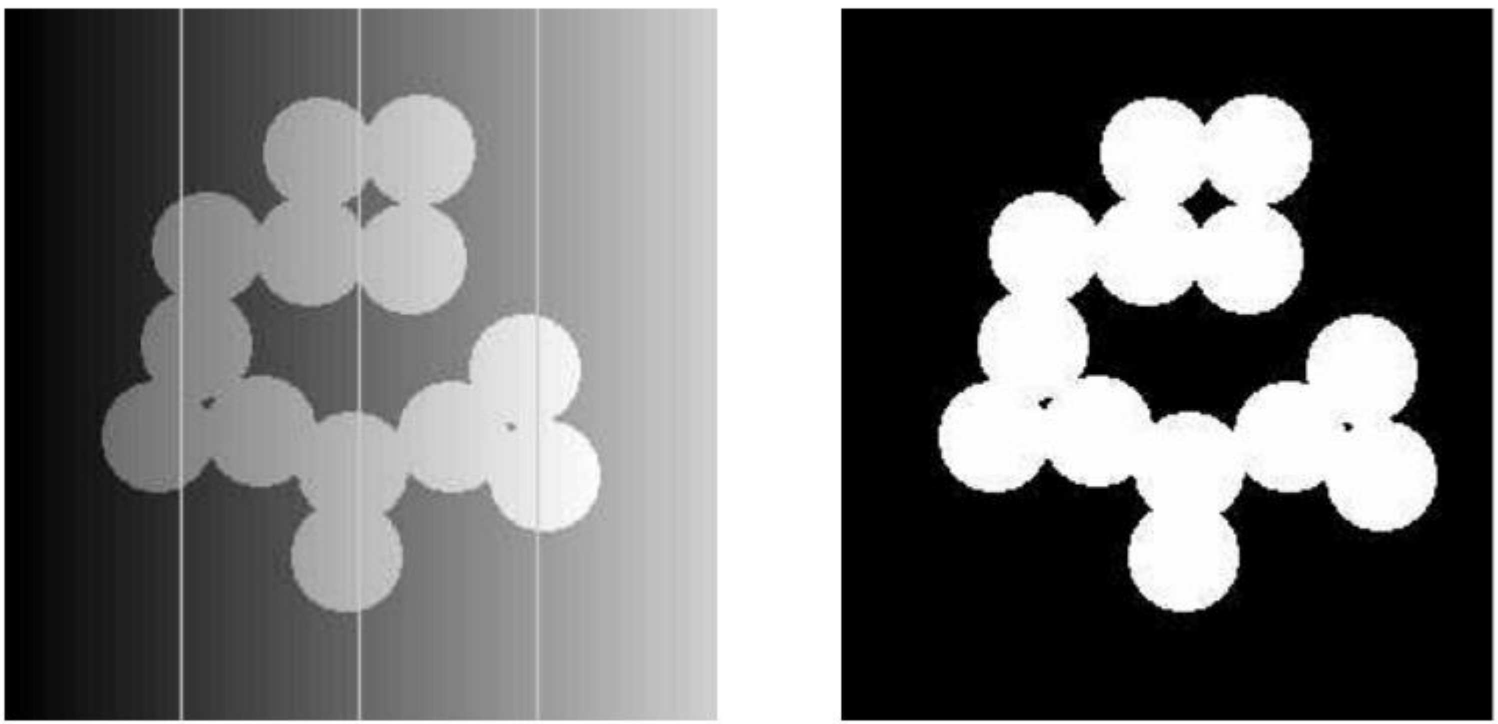
- Limitation of thresholding
  - The same threshold is used over an entire image



(a) The image as a function (b) Thresholding attempt

Adaptive Thresholding

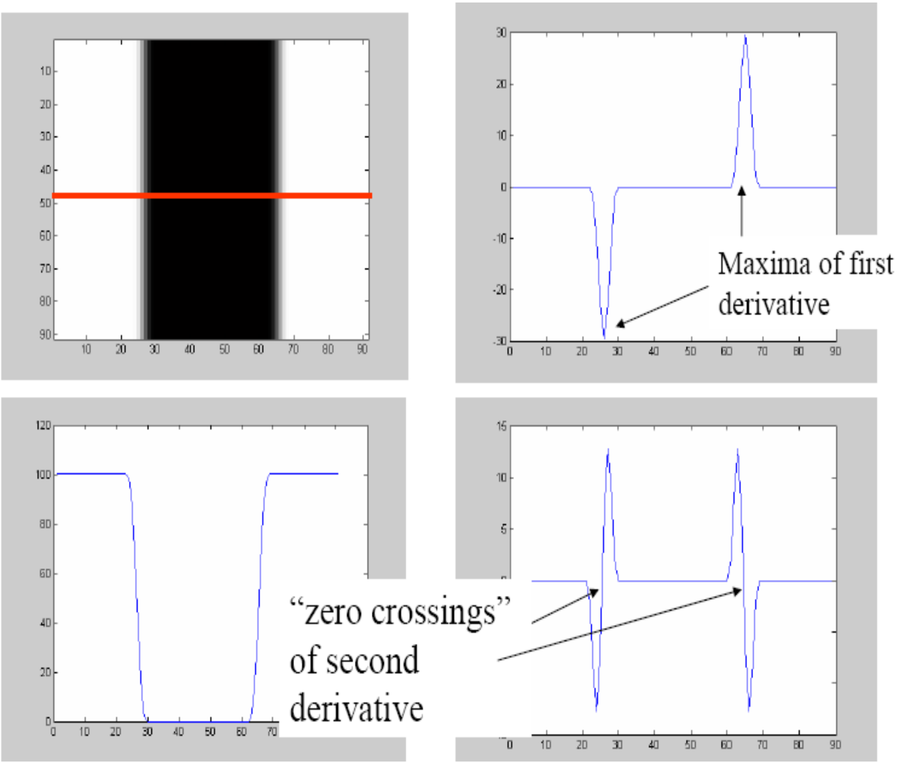
- Simple solution
  - Divide the image into a set of sub-images, and apply the threshold for each sub-image



(a) Cutting up the image (b) Thresholding each part separately

Edge Detection

Edges



- Where the image values exhibit sharp variations
- Edges can be measured by
  - 1st order derivatives
    - Determine the gradients
  - 2nd order derivatives
    - Find zero crossings in 2nd derivatives using Laplacian

First-order Derivative Filters (1D)

- Sharp changes correspond to peaks of the first-derivative of the input signal

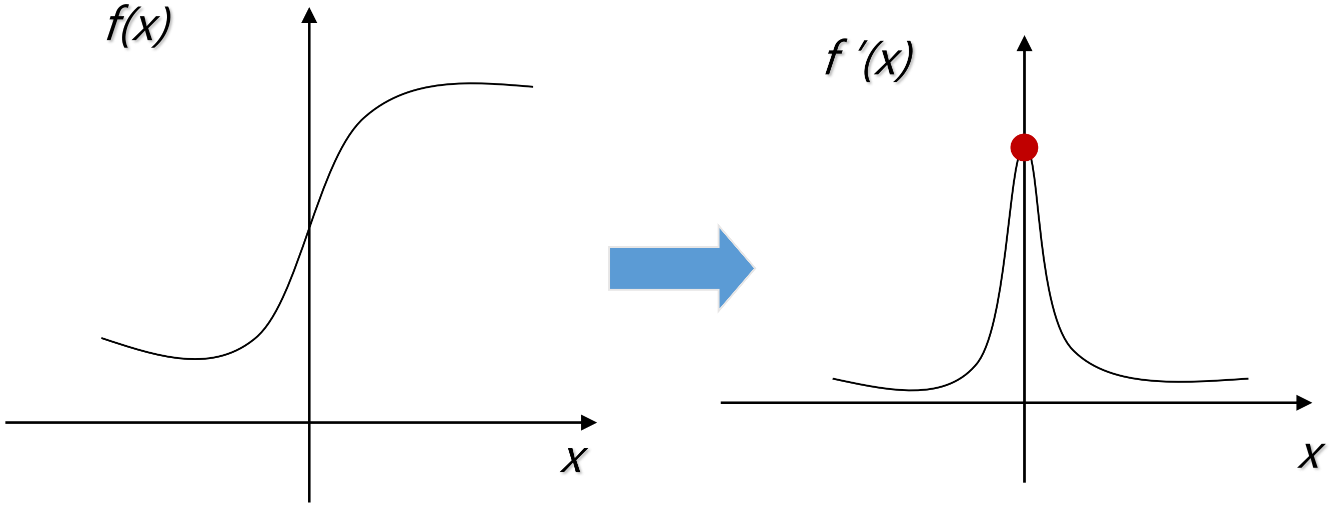


Image Gradient

- 2D gradient of an image:

$$\nabla I = (I_x, I_y) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right)$$

- The gradient magnitude (edge strength):

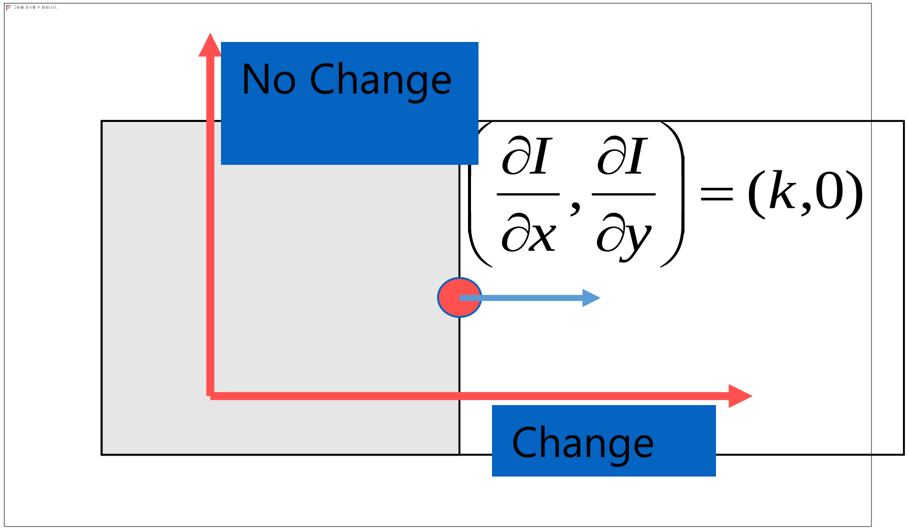
$$\|\nabla I\| = \sqrt{I_x^2 + I_y^2}$$

- The gradient direction:

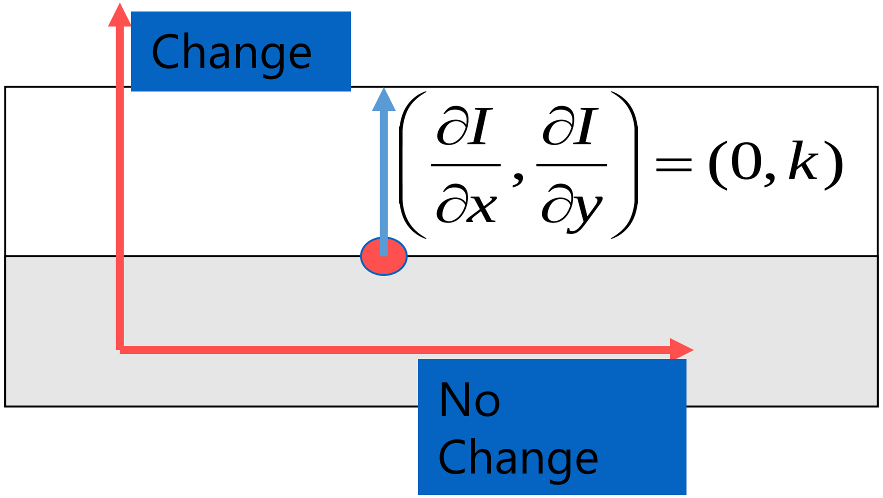
$$\theta = \tan^{-1}\left(\frac{I_y}{I_x} \right)$$

Image Gradient

- Horizontal change:



- Vertical change:



Discrete Approximation of Derivatives

- 1D derivative

$$\frac{df(x)}{dx} = \begin{cases} \lim\limits_{\Delta x \to 0} \dfrac{f(x+\Delta x) - f(x)}{\Delta x} & : \text{forward} \\ \lim\limits_{\Delta x \to 0} \dfrac{f(x) - f(x-\Delta x)}{\Delta x} & : \text{backward} \\ \lim\limits_{\Delta x \to 0} \dfrac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} & : \text{central} \end{cases}$$

- Discrete approximations

$$\frac{df(x)}{dx} \cong \begin{cases} f(x+1) - f(x) \\ f(x) - f(x-1) \\ \dfrac{f(x+1) - f(x-1)}{2} \end{cases}$$

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

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

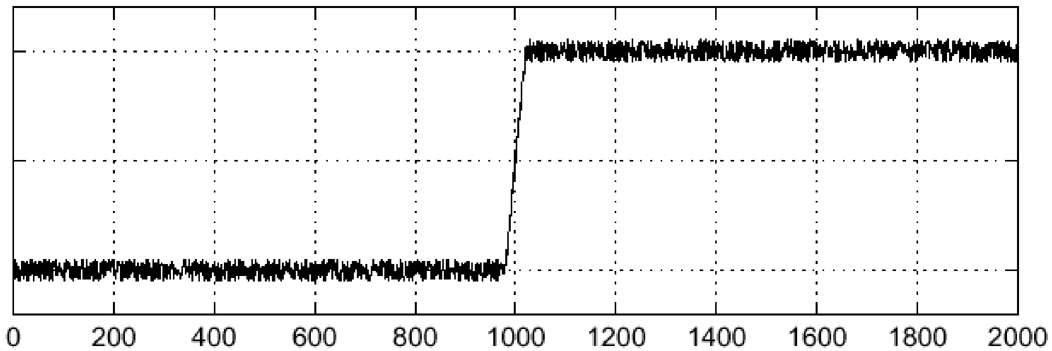
$$\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

symmetric

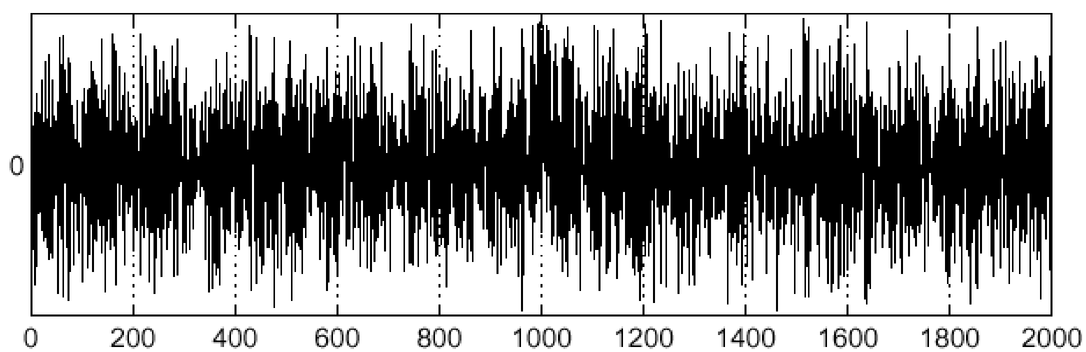
Effects of Noises

- Consider an 1-D signal

$$I(x)$$

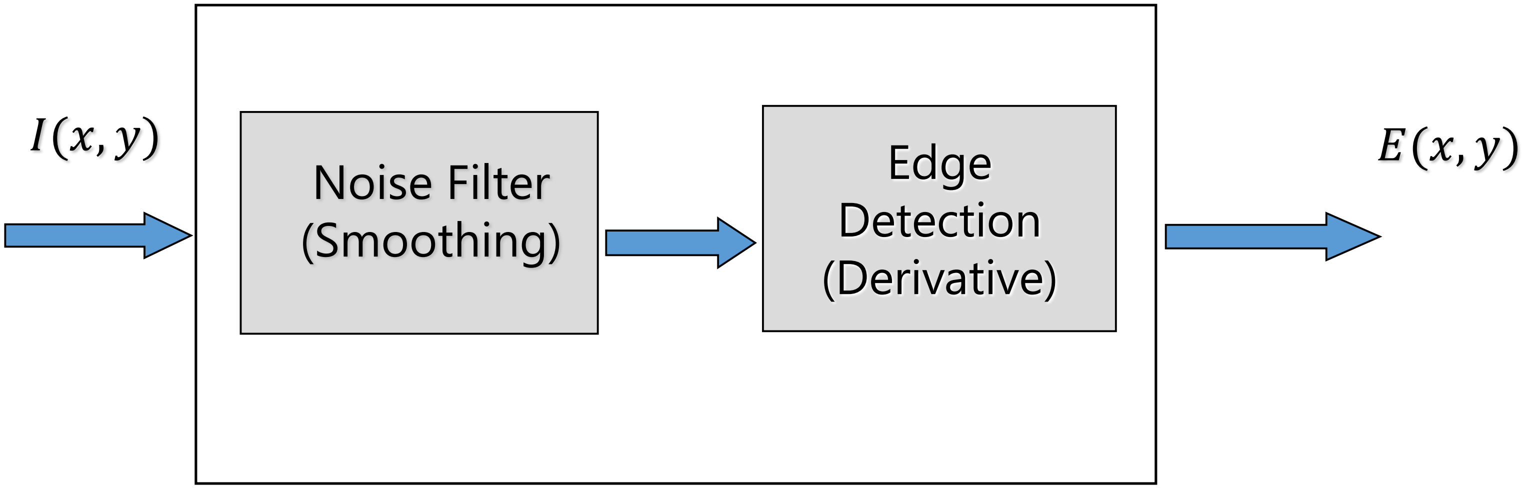


$$\frac{dI(x)}{dx}$$



- Can you detect the edge?

Noise Suppression: Pre-smoothing



$$
\begin{aligned}
E(x,y) &= D(x,y) * (S(x,y) * I(x,y)) \\
&= (D(x,y) * S(x,y)) * I(x,y)
\end{aligned}
$$

Noise smoothing and edge detection

- Prewitt edge detector:

Vertical mask

$$\frac{\partial}{\partial x}$$

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

Noise Smoothing

Vertical Edge Detection

Horizontal mask

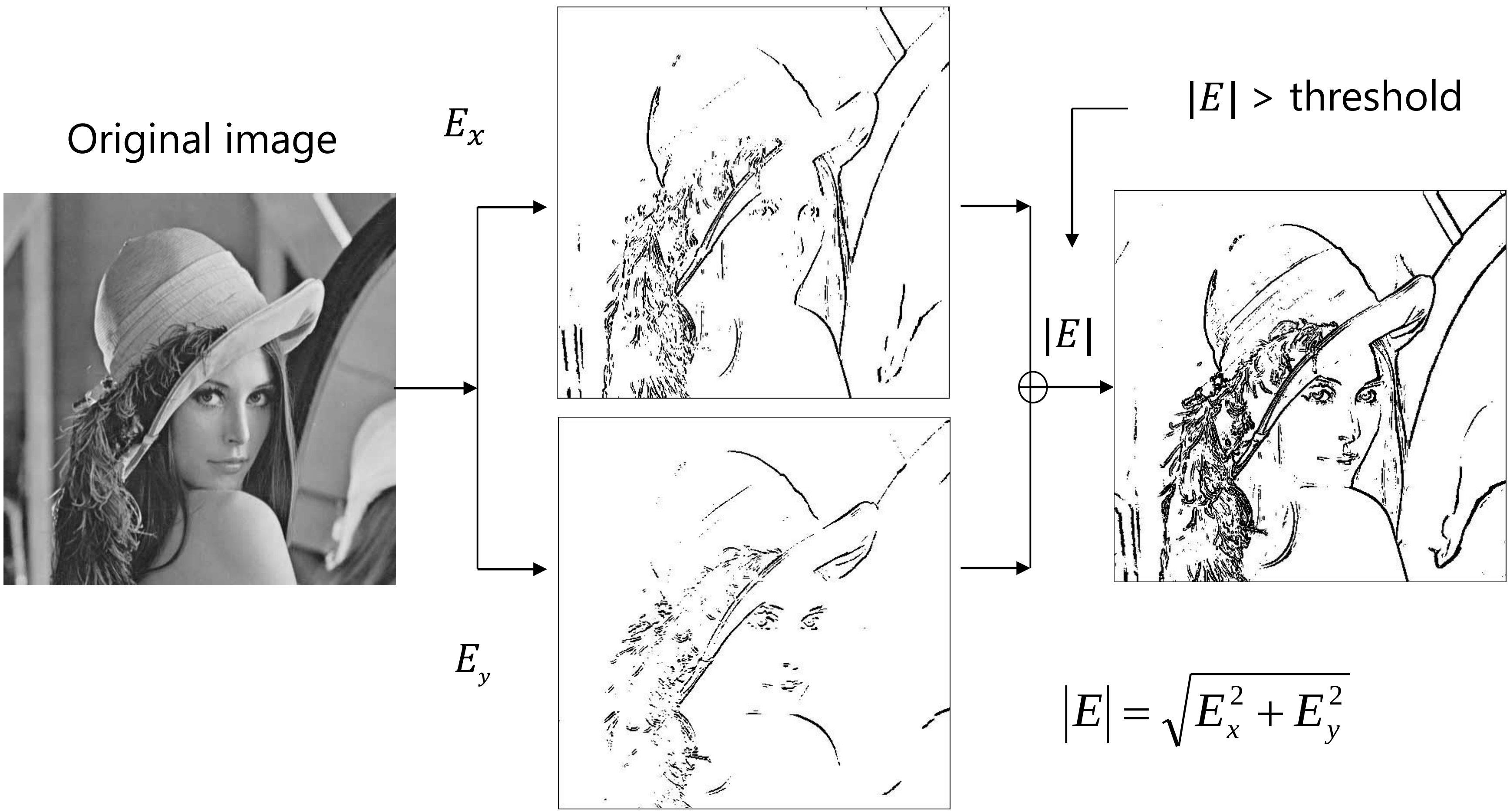
$$\frac{\partial}{\partial y}$$

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

Horizontal Edge Detection

Noise Smoothing

Prewitt Edge Detector



$$|E| = \sqrt{E_x^2 + E_y^2}$$

Result of Prewitt operator (threshold = 100)

Sobel Edge Detector

- Sobel Masks:
  - Give more weight to the 4-neighbors

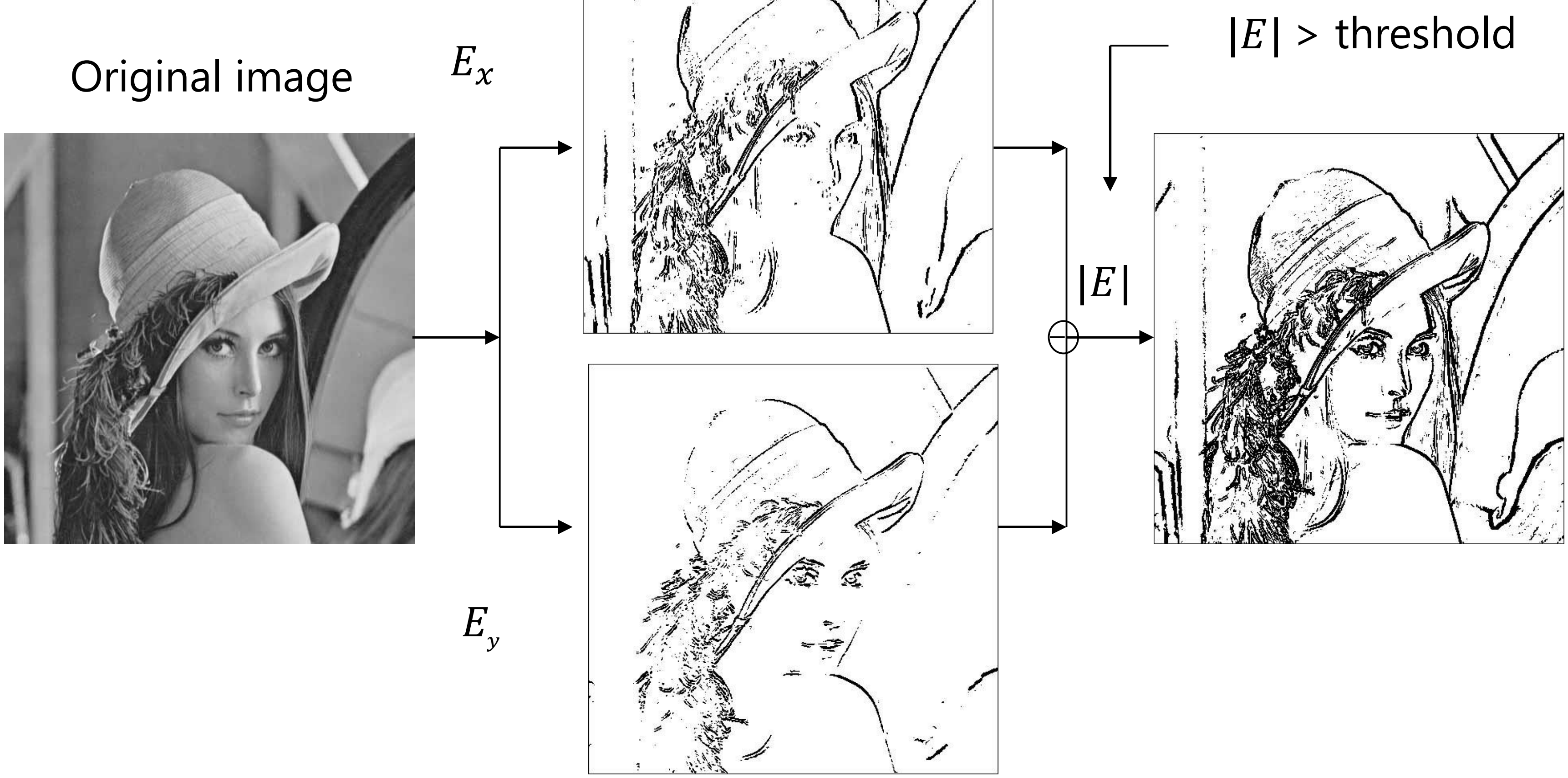
$$\frac{\partial}{\partial x}$$

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

$$\frac{\partial}{\partial y}$$

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

Sobel Edge Detector



Result of Sobel operator (threshold = 100)

Gaussian Smoothing

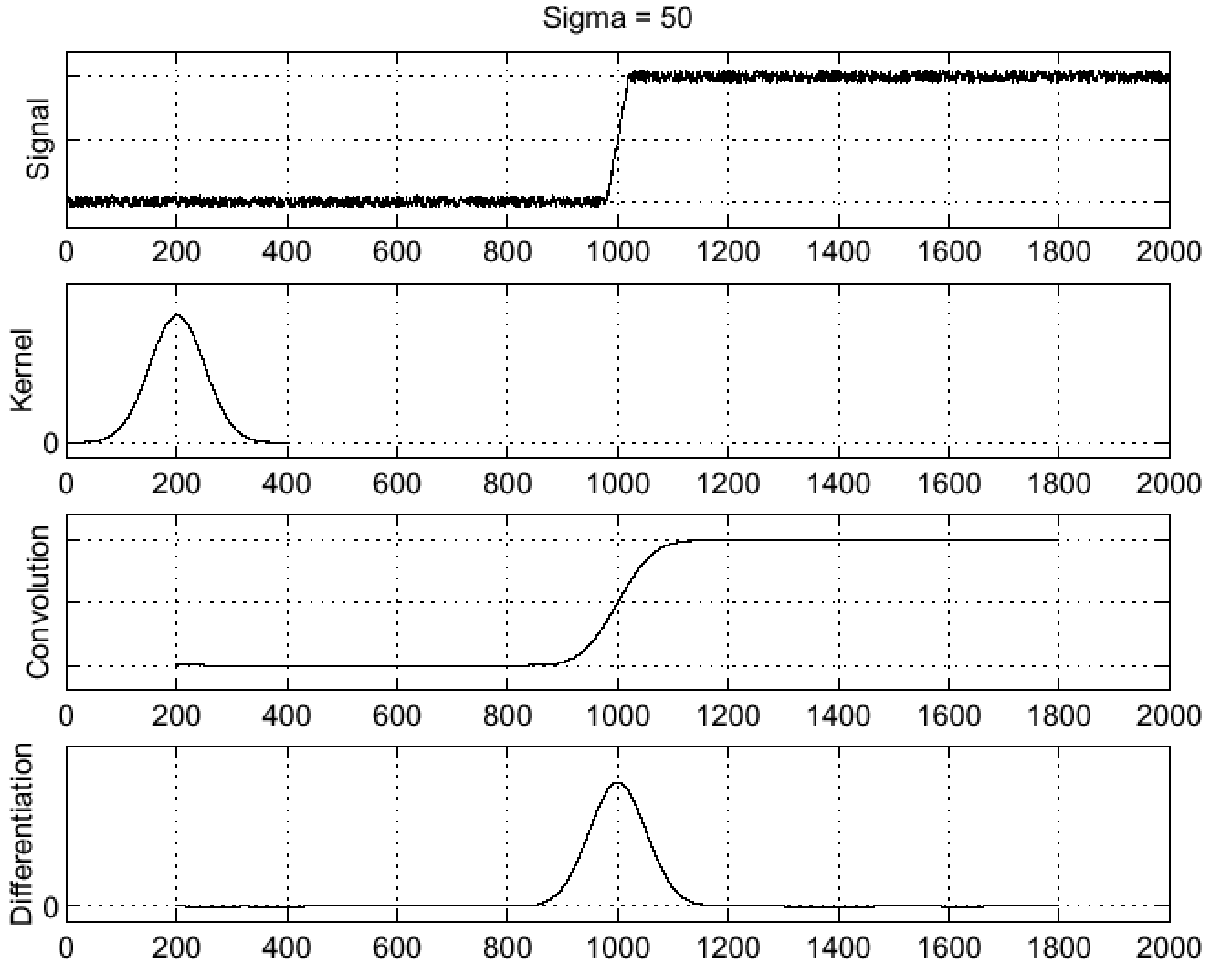
- Consider smoothing with Gaussian kernel

$$I$$

$$G(x) = e^{-\frac{x^2}{2\sigma^2}}$$

$$G * I$$

$$\frac{d}{dx}(G * I)$$



Derivative of Gaussian

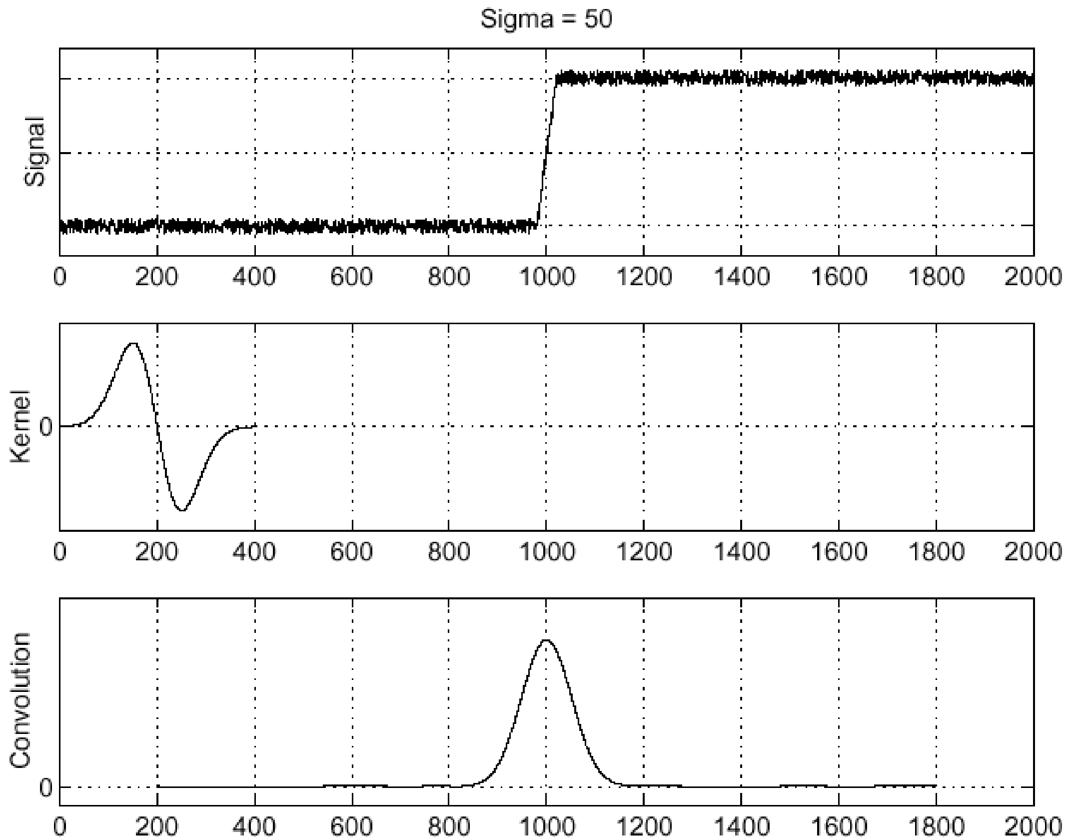
- Note that $\frac{d}{dx}(G * I) = \left(\frac{d}{dx}G\right) * I$ and $G'(x) = -\frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}}$
- This saves us one step

$$\boxed{G(x) = e^{-\frac{x^2}{2\sigma^2}}}$$

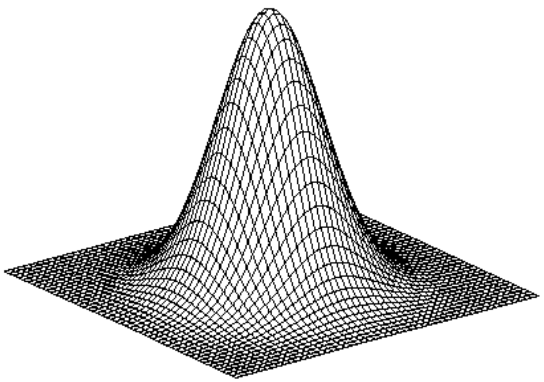
I

$\frac{d}{dx}G$

$\left(\frac{d}{dx}G\right) * I$

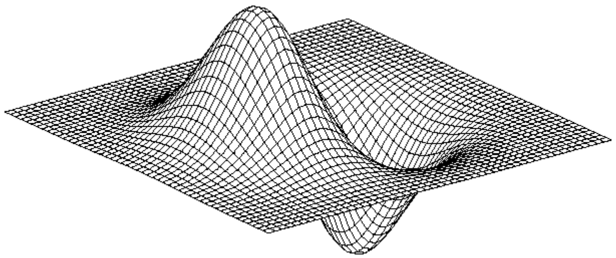


2D Gaussian Filters



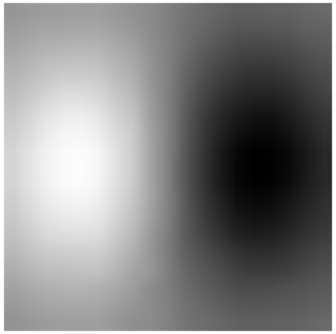
Gaussian

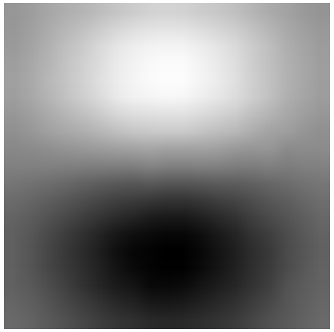
$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$



derivative of Gaussian (DOG)

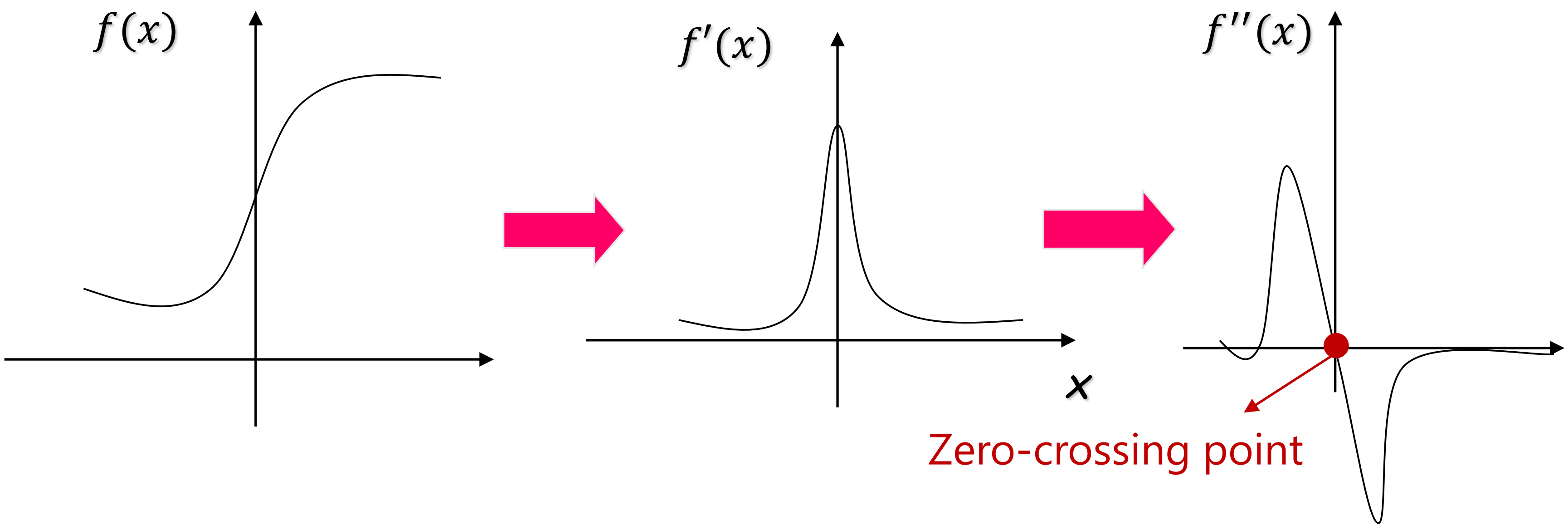
$$\nabla G(x,y)=\left(G_x,G_y\right)$$





Second-order derivative filters (1D)

- Peaks of the first-derivative of the input signal correspond to “zero-crossings” of the second-derivative.



Laplacian Operator

Laplacian Filtering output $L(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

$$
\begin{aligned}
\nabla^2 f &= \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}, \\
\frac{\partial^2 f}{\partial x^2} &= f(x+1,y) + f(x-1,y) - 2f(x,y), \\
\frac{\partial^2 f}{\partial y^2} &= f(x,y+1) + f(x,y-1) - 2f(x,y), \\
\nabla^2 f &= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y).
\end{aligned}
$$

0	1	0
1	-4	1
0	1	0

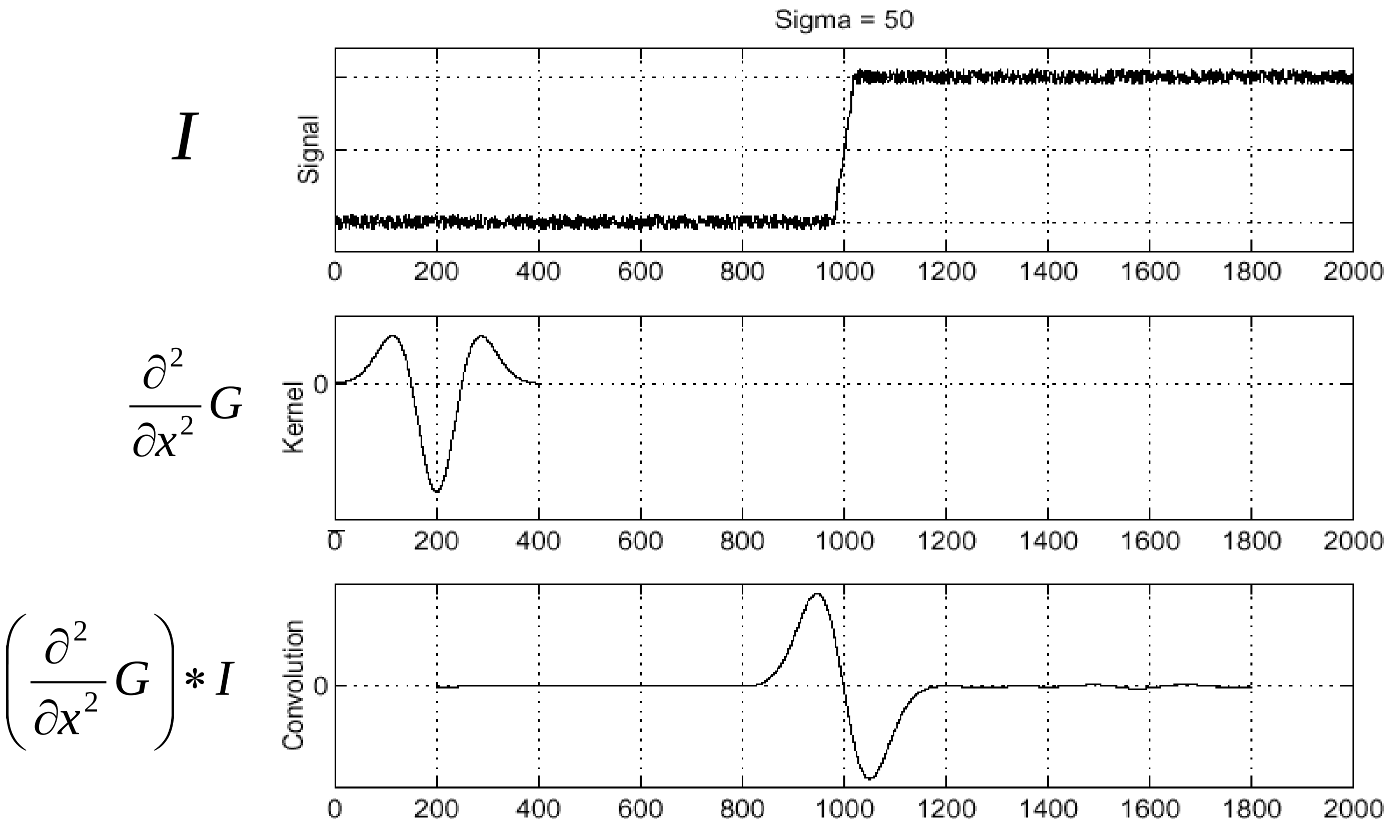
1	1	1
1	-8	1
1	1	1

Laplacian Operator

- $\nabla^2 I(x, y)$ is the sum of second-order derivatives
  - But taking derivatives increases noises
  - Very sensitive to noises
- It is always combined with a smoothing (Gaussian) operation

Laplacian of Gaussian (LOG)

- In 1D, consider $\dfrac{\partial^2}{\partial x^2}(G * I) = \left(\dfrac{\partial^2}{\partial x^2} G\right) * I$



- Edge is the zero-crossing of the bottom graph

Laplacian of Gaussian (LOG)

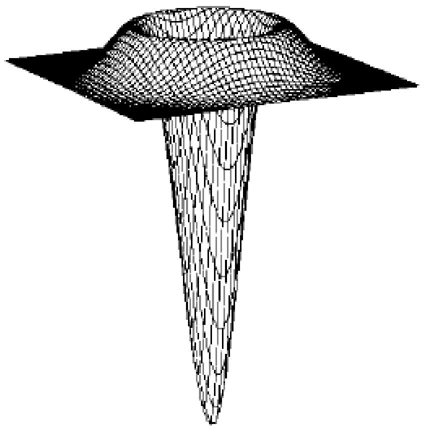
- $O(x, y) = \nabla^2(I(x, y) * G(x, y))$
  1. Smoothing with a Gaussian filter
  2. Finding zero-crossings with a Laplacian filter
- Using linearity:
  - $O(x, y) = \nabla^2 G(x, y) * I(x, y)$
  - The combined filter is called LOG

$$G(x,y) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

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

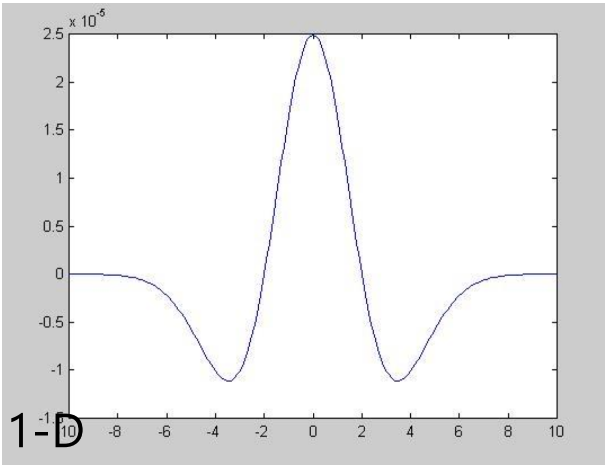
$$= \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2}\right)\exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

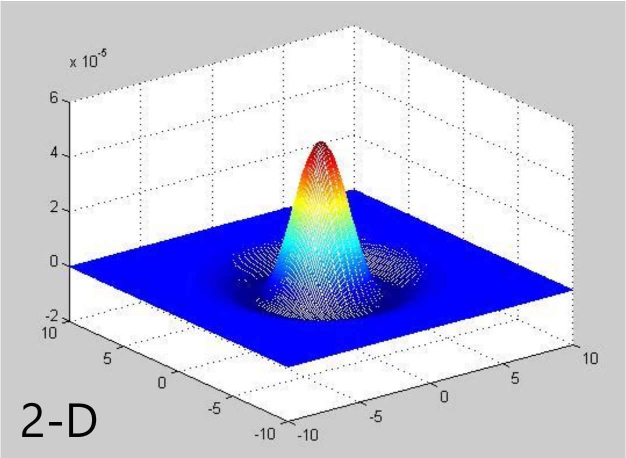
$$= \left(\frac{r^2}{\sigma^4} - \frac{2}{\sigma^2}\right)\exp\left(-\frac{r^2}{2\sigma^2}\right)$$



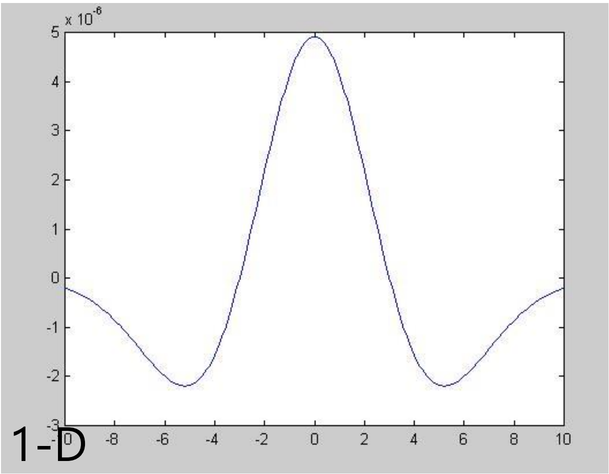
LOG Filter

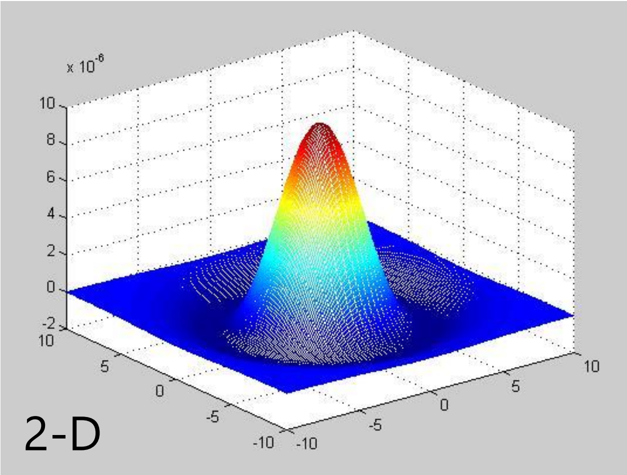
- Mexican hat operator (inverted LoG)





$\sigma = 2$



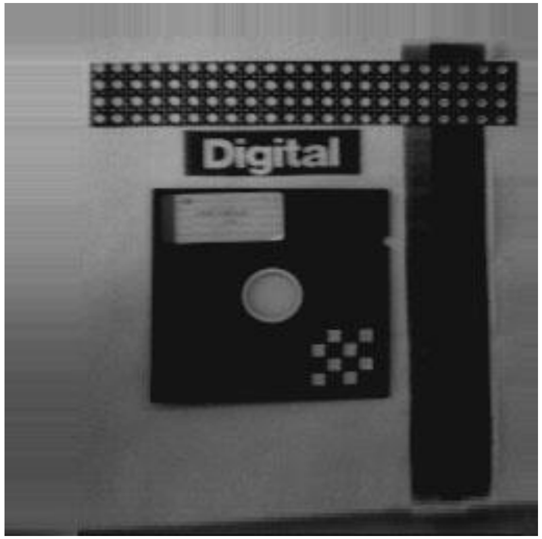


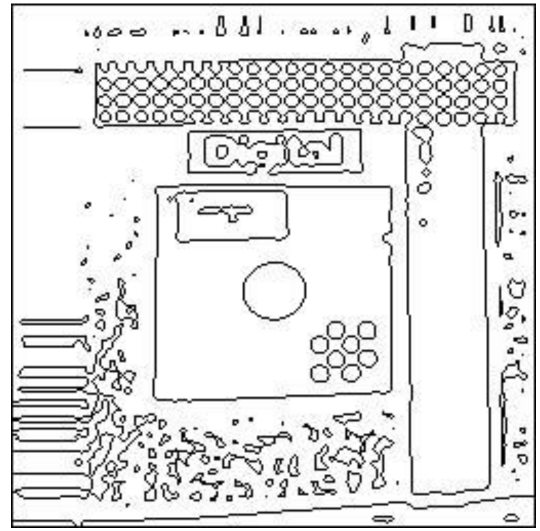
$\sigma = 3$

LOG Filter

LOG Filter

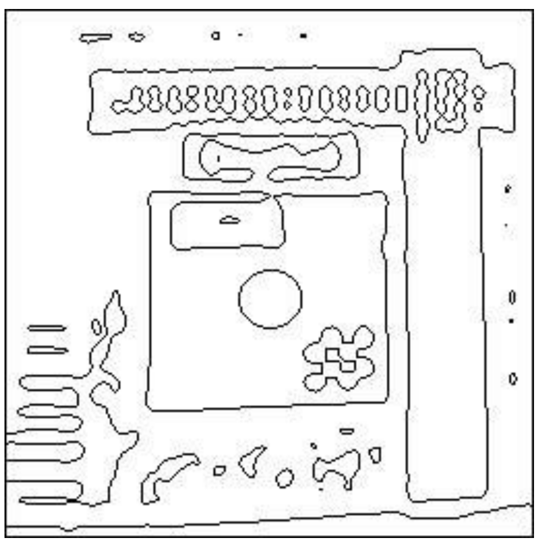
Original image

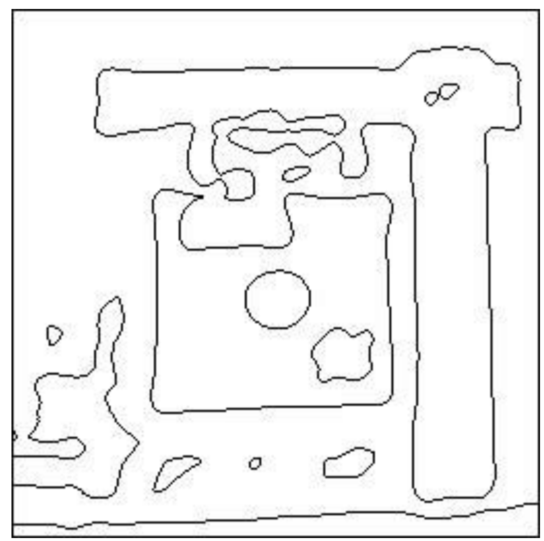




σ = 2.0

σ = 4.0





σ = 6.0

Second-Order Edge Detectors

- The Marr-Hildreth Method
  1. Laplacian of Gaussian (LoG)

$$O(x,y) = \nabla^2 G(x,y) * I(x,y)$$

$$G(x,y) = \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

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

$$= \left(\frac{x^2 + y^2}{\sigma^4} - \frac{2}{\sigma^2}\right)\exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

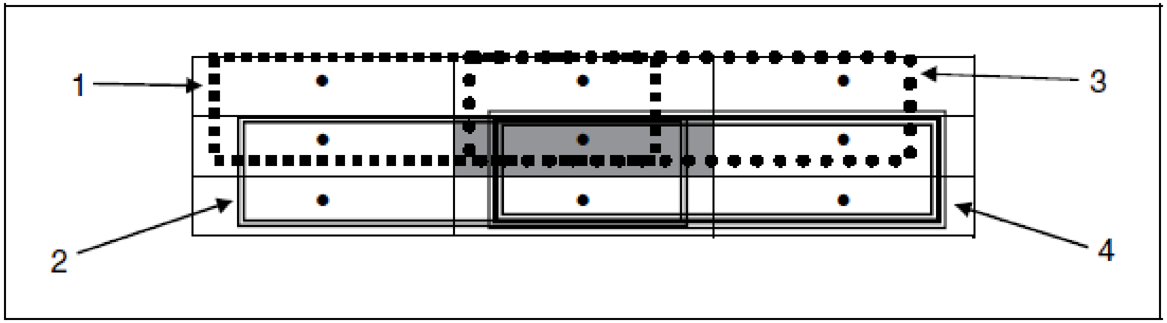
$$= \left(\frac{r^2}{\sigma^4} - \frac{2}{\sigma^2}\right)\exp\left(-\frac{r^2}{2\sigma^2}\right)$$

  2. Finding zero-crossing points

The location where the signs of filtered values change, after applying the Laplacian using the second derivative

Second-Order Edge Detectors

- Finding zero-crossing points

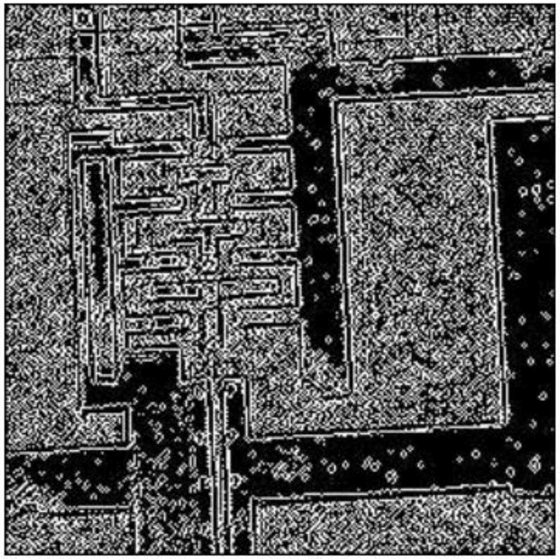


- Find the averages of the four quadrants
- If the max average is positive and the min average is negative, then the center point is detected

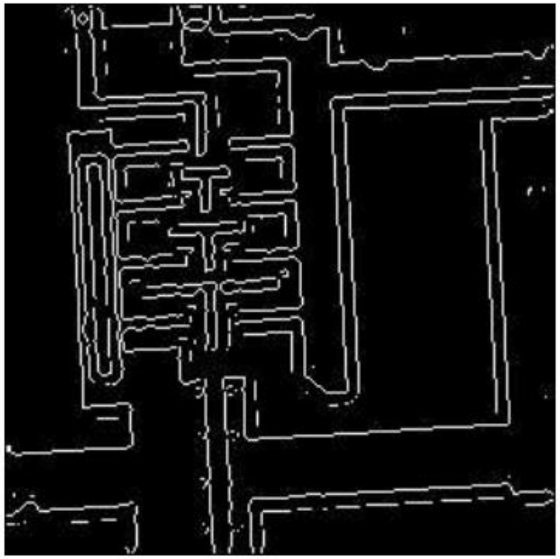
Second-Order Edge Detectors

- The Marr-Hildreth Method
  1. Apply the Gaussian filter for removing unnecessary noise.
  2. Apply the Laplacian filter
  3. Find the zero-crossing pixels

  LoG



(a) Zeros crossings

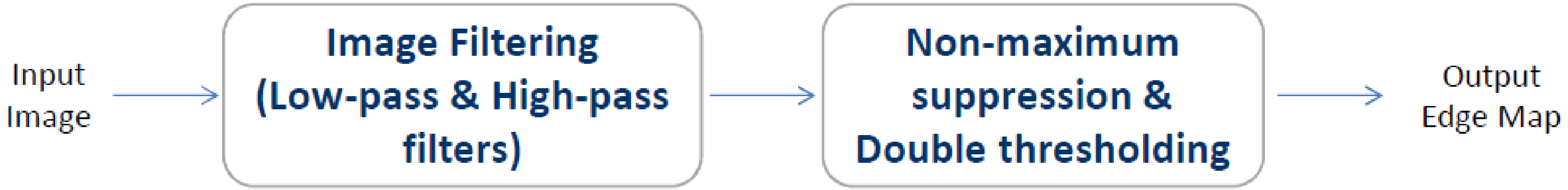


(b) Using an LoG filter first

Figure 8.13: Edge detection using zero crossings

Canny Edge Detection

- Three criteria for edge detection
  - Low error rate of detection: finding all edges
  - Localization of edges: computing precise locations of edges
  - Single response: returning a single pixel for a single edge



Canny Edge Detection

Canny Edge Detection

- Image Filtering using Low-pass & High-pass filters

1. Apply a low-pass filter, e.g. Gaussian filter $G = I(x,y) * f(x,y)$

$$f(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

2. Apply a high-pass filter, e.g. Sobel filter or 1D derivative of Gaussian filter

$$\nabla G = (G_x, G_y) = \left(\frac{\partial G}{\partial x}, \frac{\partial G}{\partial y}\right)$$

S_x

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

S_y

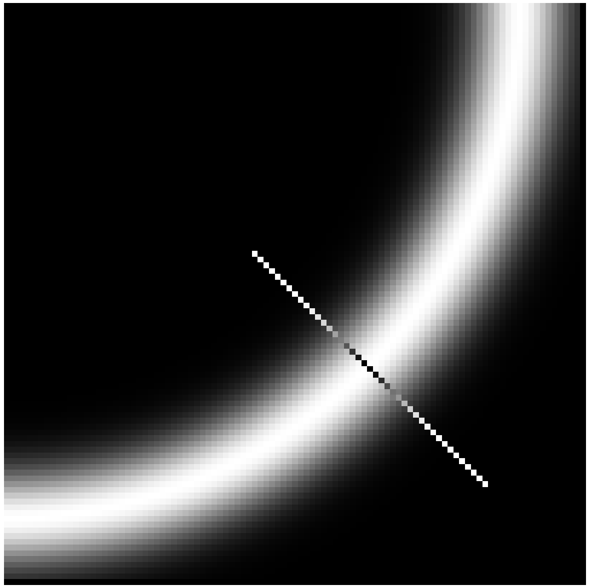
-1	-2	-1
0	0	0
1	2	1

3. Compute the magnitude and angle of ∇G

$$M(x,y) = \sqrt{{G_x}^2 + {G_y}^2} \qquad A(x,y) = tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Canny Edge Detection

- Non-maximum suppression
  - We wish to mark points along the curve where the magnitude is biggest
  - We can do this by looking for the maximum along a slice normal to the curve (nonmaximum suppression)



Canny Edge Detection

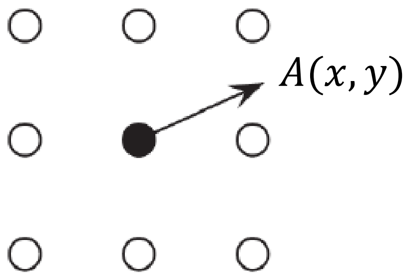
- Non-maximum suppression
  - Key idea: Survive only pixels with **a larger edge magnitude** $M(x, y)$ within a small window

$$M(x,y) = \sqrt{{G_x}^2 + {G_y}^2} \qquad A(x,y) = tan^{-1}\left(\frac{G_y}{G_x}\right)$$

  - **Procedure**

1. Within a small window (e.g. 3×3 window) centered at (x, y), find neighbor pixels in direction $A(x, y)$

2. Compare the edge magnitudes $M(x, y)$ of these two neighbor pixels



The edge direction at a pixel

Canny Edge Detection

- But, the image is a discrete signal
  - So, let's use an interpolation (linear interpolation or quantization (nearest neighbor)

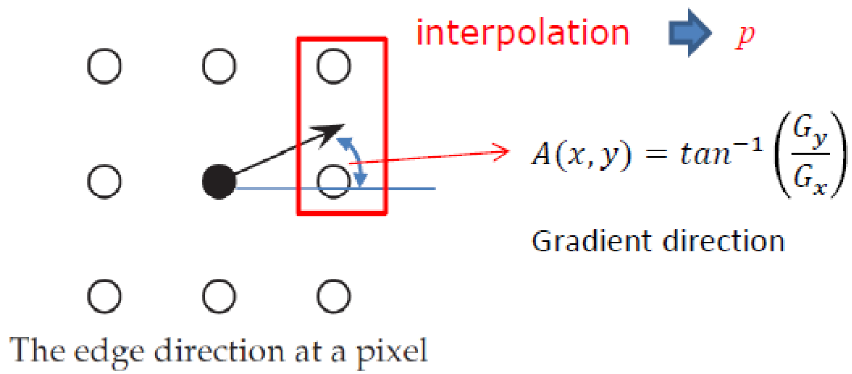


FIGURE 9.26 *Nonmaximum suppression in the Canny edge detector.*

Canny Edge Detection

- Double Thresholding
  - Hysteresis thresholding (T_L, T_H)
  - If $M(x,y) > T_H$, then (x,y) is an edge
  - If $M(x,y) < T_L$, then (x,y) is **NOT** an edge
  - If $T_L \leq M(x,y) \leq T_H$,
    - If the **neighboring** pixels of (x,y) is an edge, then (x,y) is an edge.
    - Otherwise, then (x,y) is **NOT** an edge.

Results

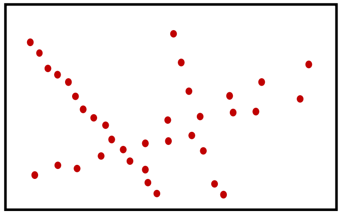
original image

Gradients

Nonmaximum suppression and thresholding

Hough Transform for Line Fitting

- Hough Transform is used for fitting lines, when a set of sparse points are given
  - In edge detection, the resulting edge image may often consist of individual edge points



- **Key idea**: transform $(x, y) \rightarrow (a, b)$

$$y = ax + b \Rightarrow 1 = a + b$$

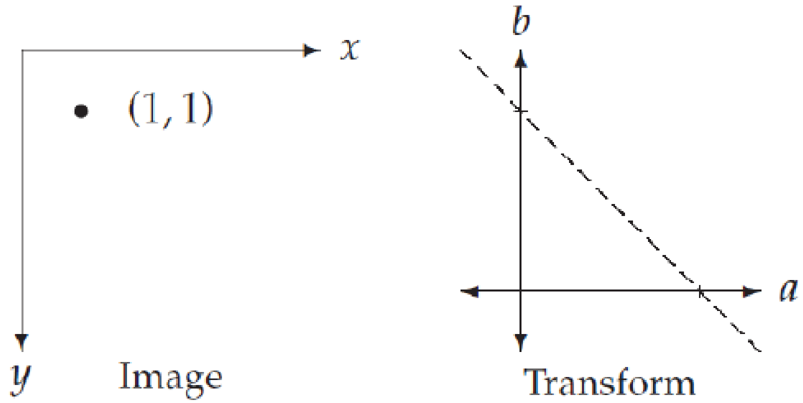


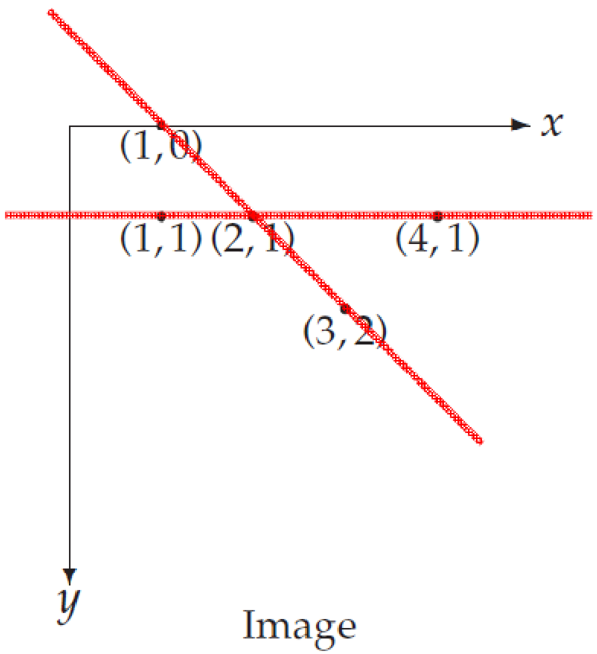
FIGURE 9.31 *A point in an image and its corresponding line in the transform.*

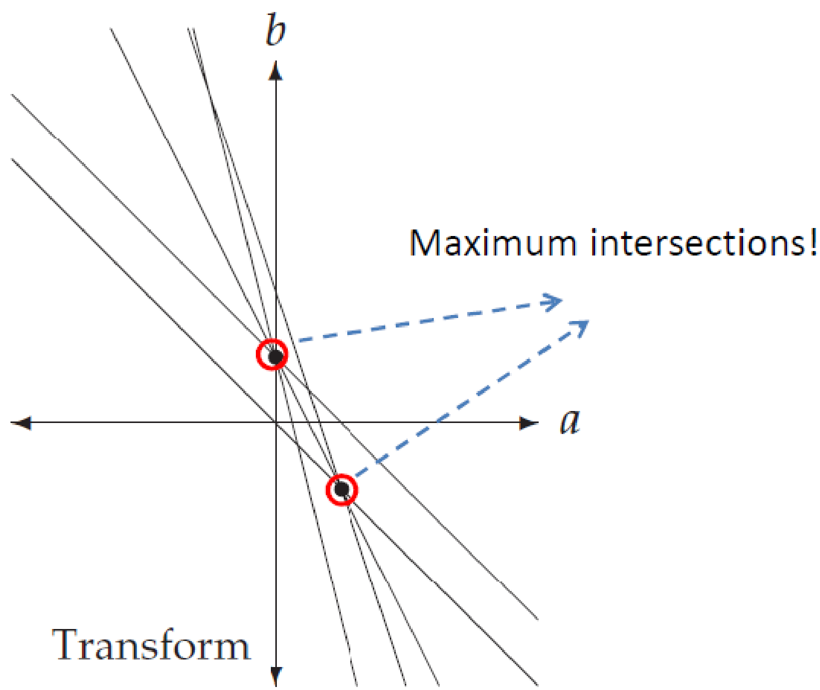
Hough Transform for Line Fitting

- Suppose we have five points

$$
\begin{aligned}
(1,0) &\rightarrow b = -a \\
(1,1) &\rightarrow b = -a + 1 \\
(2,1) &\rightarrow b = -2a + 1 \\
(4,1) &\rightarrow b = -4a + 1 \\
(3,2) &\rightarrow b = -3a + 2
\end{aligned}
$$

$$
\begin{aligned}
(a,b) &= (0,1) \\
(a,b) &= (1,-1)
\end{aligned}
$$





Hough Transform for Line Fitting

- But, $y = ax + b$ is not able to model a vertical line

  → Let's use different type of parameterization (r, θ)!

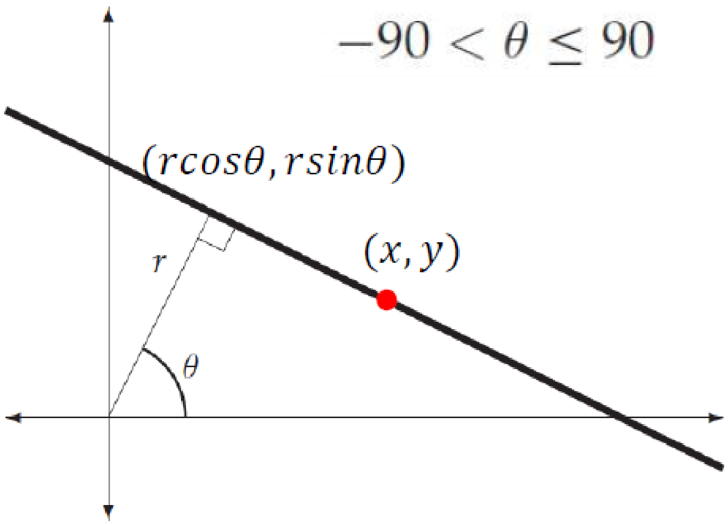


FIGURE 9.34 *A line and its parameters.*

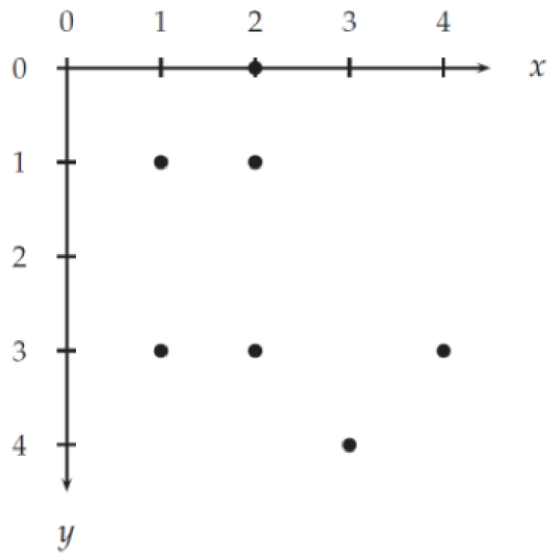
$$\frac{y - rsin\theta}{x - rcos\theta} = \tan(90 + \theta) = -\frac{cos\theta}{sin\theta}$$

$$\Rightarrow \boxed{xcos\theta + ysin\theta = r}$$

Hough Transform for Line Fitting

- One problem still happens in case of handling continuous parameters (r, θ) on the discrete domain
  - Too many possible (r, θ)
  - Let's just use a few number of quantized (r, θ)

- If we discretize θ to use only four values: -45°,0°,45°,90°



(x, y)	$-45°$	$0°$	$45°$	$90°$
$(2, 0)$	1.4	2	1.4	0
$(1, 1)$	0	1	1.4	1
$(2, 1)$	0.7	2	2.1	1
$(1, 3)$	-1.4	1	2.8	3
$(2, 3)$	-0.7	2	3.5	3
$(4, 3)$	0.7	4	4.9	3
$(3, 4)$	-0.7	3	4.9	4

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

Hough Transform for Line Fitting

- The accumulator array contains how many times each value of (r, θ) appears in the table

	−1.4	−0.7	0	0.7	1	1.4	2	2.1	2.8	3	3.5	4	4.9
−45°	1	2	1	2		1							
0°					2		3			1		1	
45°						2		1	1		1		2
90°			1		2					3		2	

accumulate

(x, y)	− 45°	0°	45°	90°
(2, 0)	1.4	2	1.4	0
(1, 1)	0	1	1.4	1
(2, 1)	0.7	2	2.1	1
(1, 3)	−1.4	1	2.8	3
(2, 3)	−0.7	2	3.5	3
(4, 3)	0.7	4	4.9	3
(3, 4)	−0.7	3	4.9	4

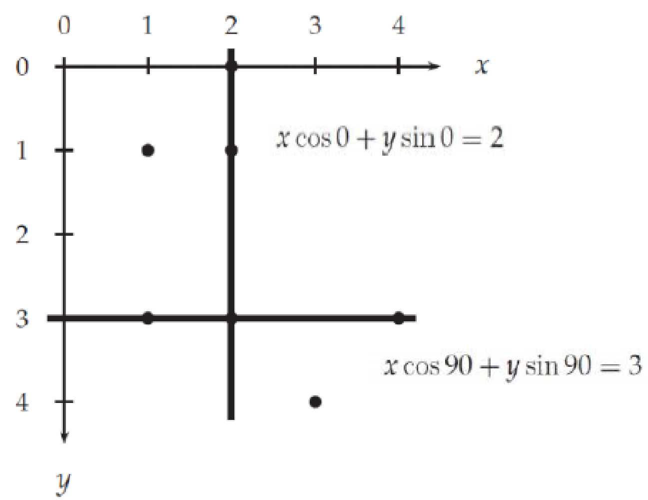
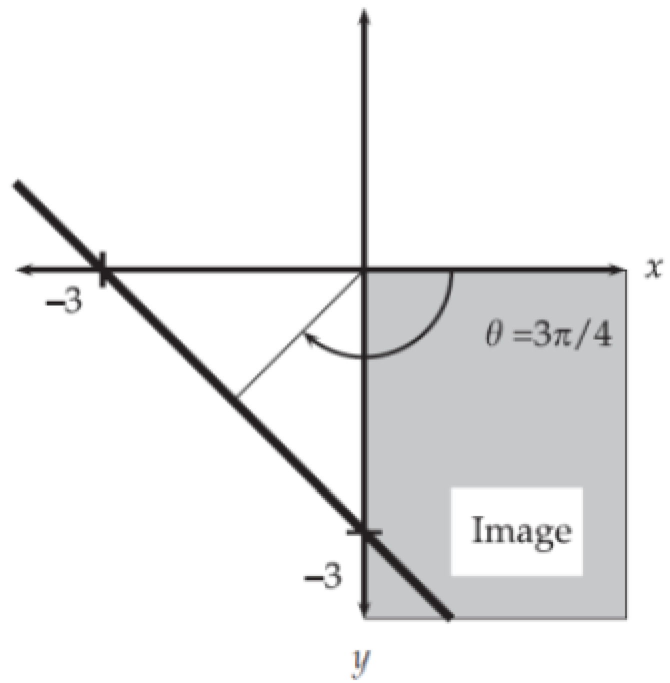


FIGURE 9.36 *Lines found by the Hough transform.*

Implementing Hough Transform in MATLAB

- Discretizing θ with sample rate 1°

```
>> angles = [-90:180]*pi/180;
```



Implementing Hough Transform in MATLAB

- Calculating the r values
  - If im is a **binary image**

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

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

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

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

- Forming the accumulator array

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

Implementing Hough Transform in MATLAB

- Updating the accumulator array

	−1.4	−0.7	0	0.7	1	1.4	2	2.1	2.8	3	3.5	4	4.9
−45°	1	2	1	2		1							
0°					2		3			1		1	
45°						2		1	1		1		2
90°			1		2					3		2	

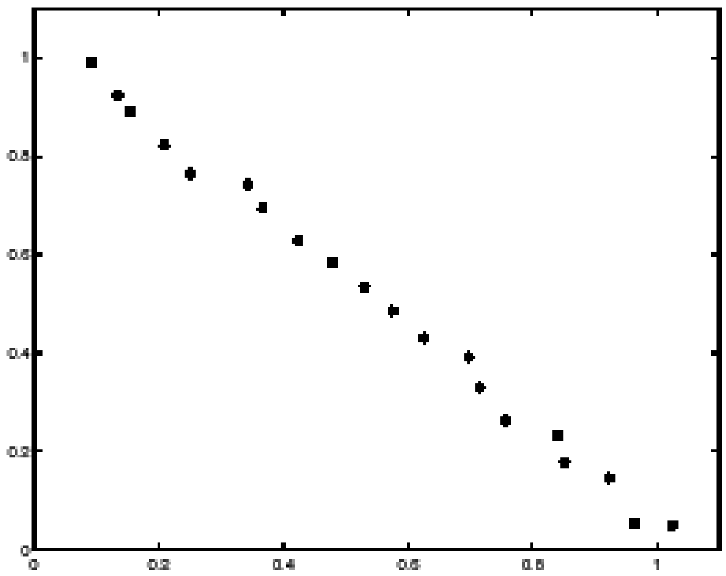
```
function res=hough2(image)

%
% 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;
```

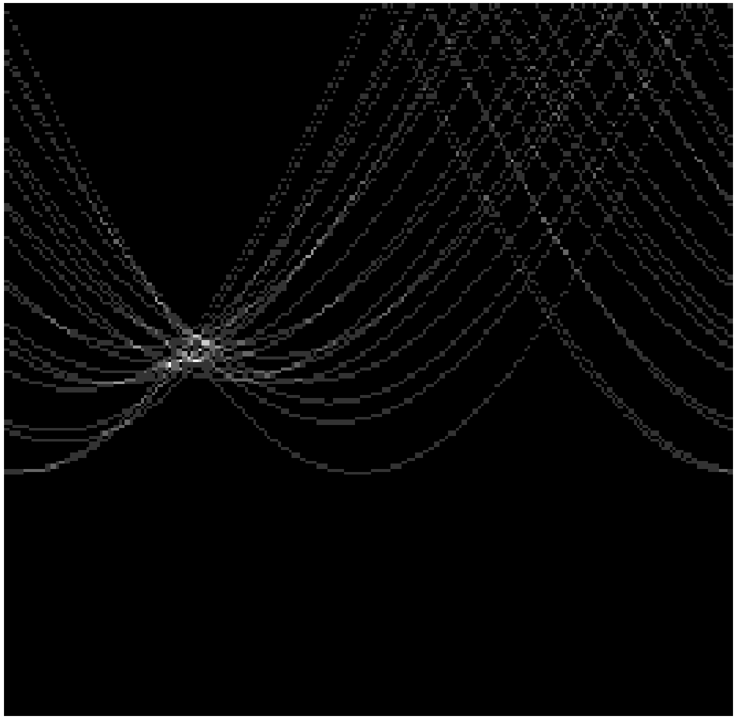
Hough Transform for Line Fitting

- If the set of points are given, we can plot the graph from the equation $x\cos\theta + y\sin\theta = r$

Hough Transform for Line Fitting

- Example : The Hough transform array
  - for a line with noises in the range [0, 0.05]->regular





Hough Transform for Line Fitting

- Example : The Hough transform array
  - for random points->irregular