Lecture 10
Image segmentation-I: edge detection, thresholding, and region detection

Dr. Xiran Cai

Email: caixr@shanghaitech.edu.cn

Office: 3-438 SIST

Tel: 20684431

ShanghaiTech University

Segmentation

- Subdivide an image into its constituent regions or object
  - Determine the level of subdivision details by the problem to be solved
  - Stop the subdivision when the regions or objects have been detected
- Improve segmentation accuracy during acquisition
  - Environment control
  - Sensor selection
  - Imaging modality

Definition

- Image segmentation can be considered as a process that partitions region R into n subregions $R_1, R_2, \cdots, R_n$, such

a) $\bigcup_{i=1}^{n}(R_i) = R$

b) R_i is a connected set, $i = 1,2, \cdots, n$

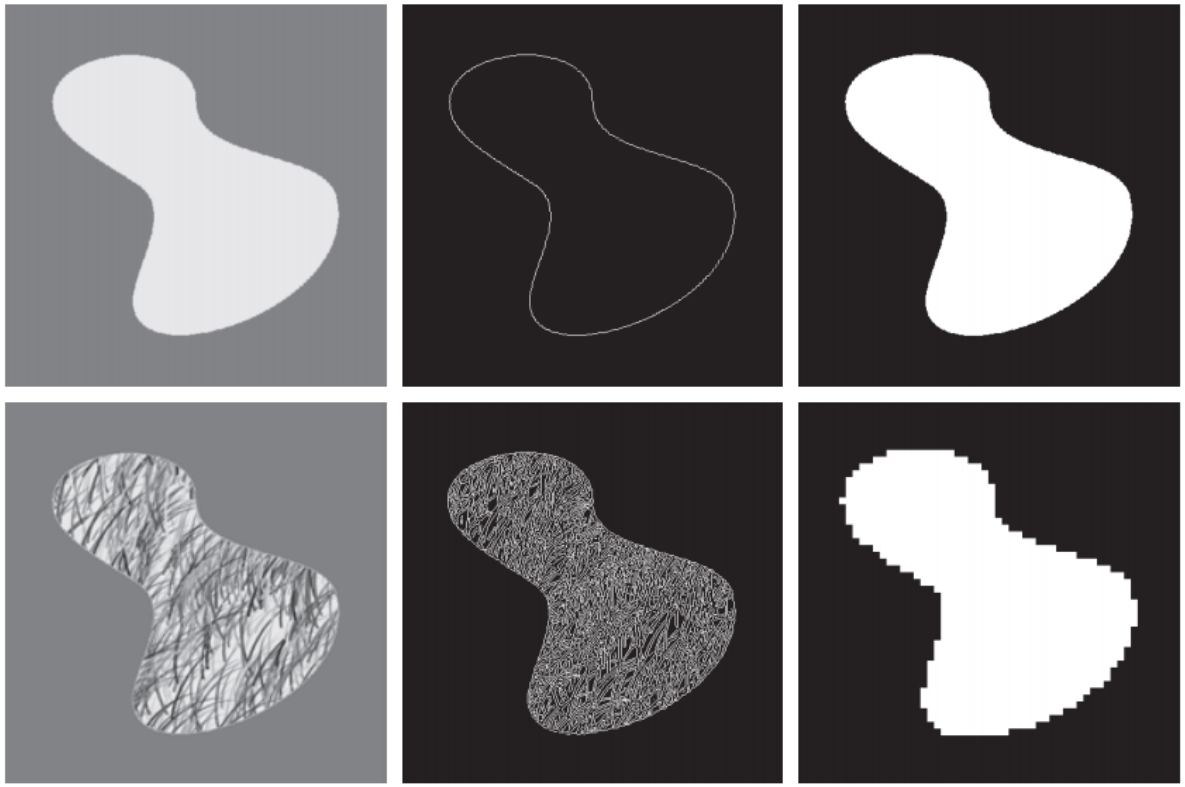
c) $R_i \cap R_j = \emptyset$ for all i and j, $i \neq j$

d) $Q(R_i) = True$ for $i = 1,2, \cdots, n$

e) $Q(R_i \cup R_j) = False$ for any adjacent region R_i and R_j

Properties of Intensity values

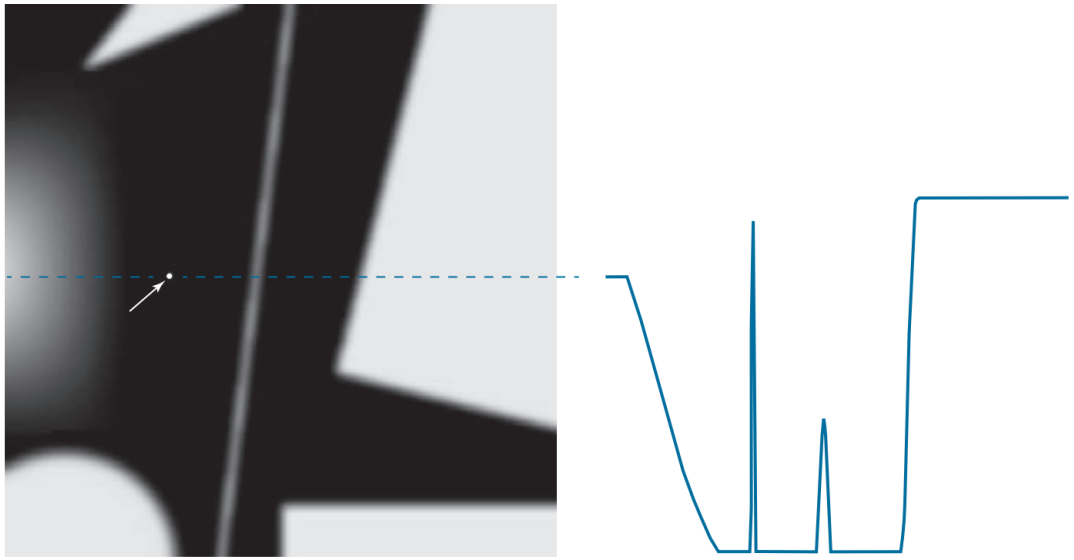
- Discontinuity: Edge-based segmentation
- Similarity: Region-based segmentation

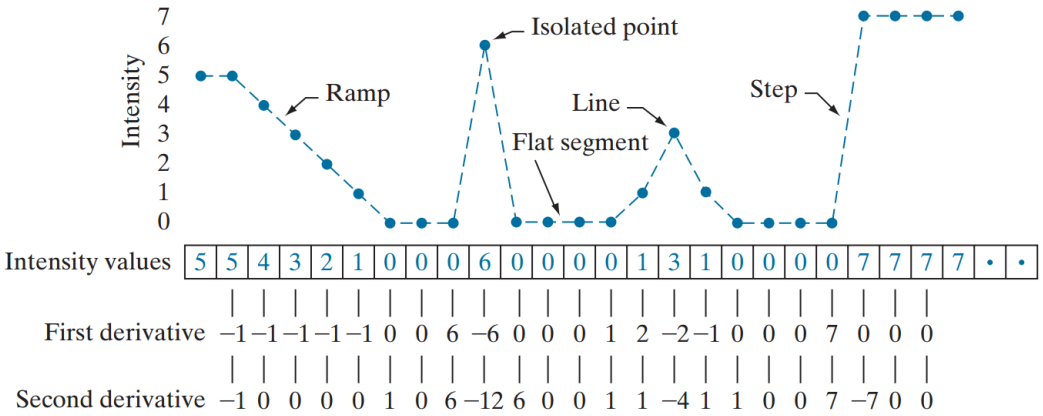


a b c
d e f

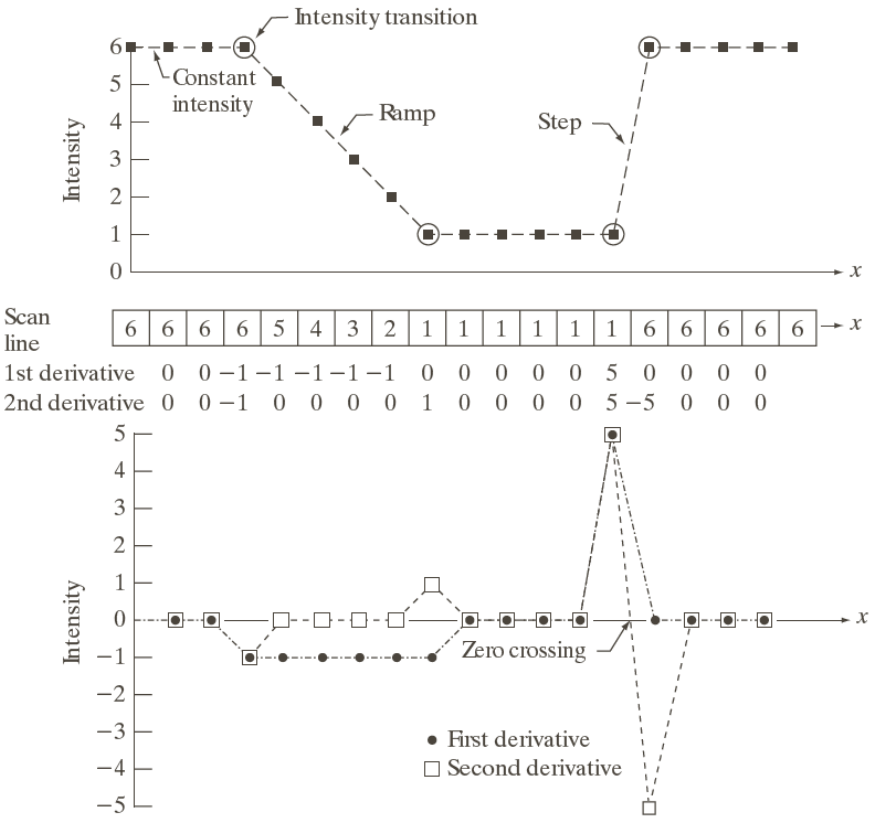
FIGURE 10.1
(a) Image of a constant intensity region.
(b) Boundary based on intensity discontinuities.
(c) Result of segmentation.
(d) Image of a texture region.
(e) Result of intensity discontinuity computations (note the large number of small edges).
(f) Result of segmentation based on region properties.

Point, Line and Edge Detection





Derivatives



(1) First-order derivatives generally produce thicker edges.

(2) Second-order derivatives have a stronger response to fine detail, such as thin lines, isolated points, and noise.

(3) Second-order derivatives produce a double-edge response at ramp and step transitions in intensity.

(4) The sign of the second derivative can be used to determine whether a transition into an edge is from light to dark or dark to light.

Spatial Filters

❑ Vector Operation

$$R = w_1 z_1 + w_2 z_2 + \cdots + w_{mn} z_{mn}$$

$$= \sum_{k=1}^{mn} w_k z_k = w^T z$$

$$R = w_1 z_1 + w_2 z_2 + \cdots + w_9 z_9 = \sum_{k=1}^{9} w_k z_k$$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

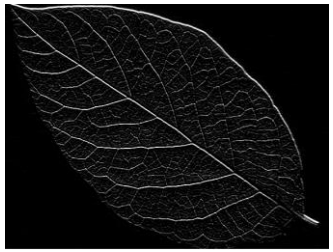
Spatial Filters

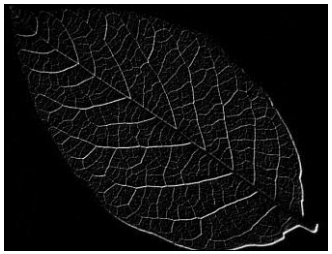
Sobel filter

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

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

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





Point Detection (点检测)

- **Output expression:**

$$g(x,y) = \begin{cases} 1, & |R(x,y)| \geq T \\ 0, & \text{otherwise} \end{cases}$$

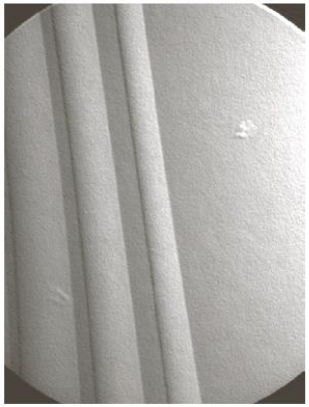
Where

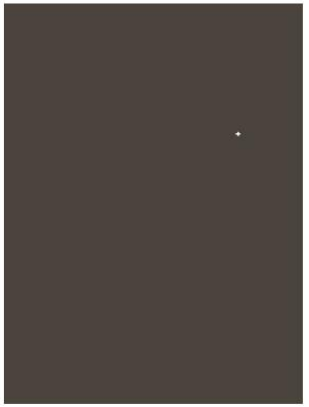
g: output image

T: nonnegative threshold

Laplacian filter

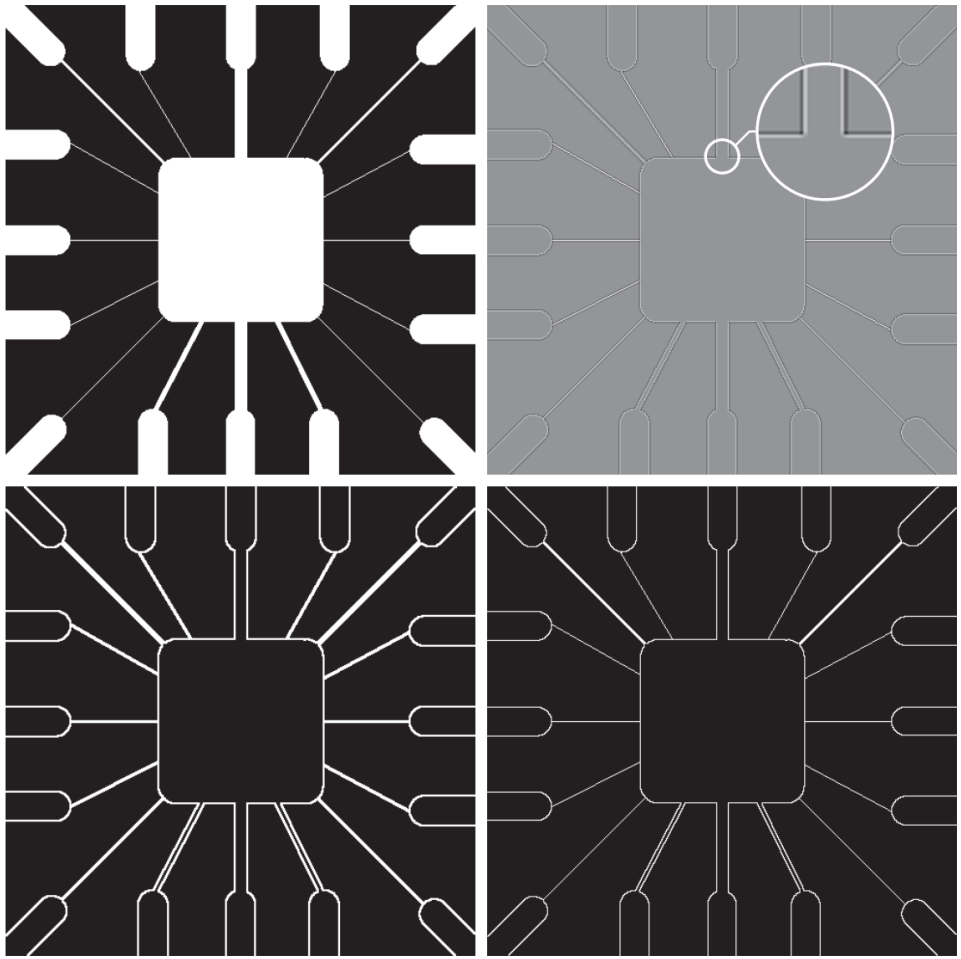
1	1	1
1	−8	1
1	1	1





Line Detection (线检测)

- Laplacian filter: notice double-line effect of the second derivative



a b
c d

FIGURE 10.5
(a) Original image.
(b) Laplacian image; the magnified section shows the positive/negative double-line effect characteristic of the Laplacian.
(c) Absolute value of the Laplacian.
(d) Positive values of the Laplacian.

Line Detection (线检测)

- **Output expression:**

$$|R_i| > |R_j|, \qquad j \neq i$$

Where R_i is the response to the masks

- **Matlab function:**

g = abs(imfilter(double(f)), w))

−1	−1	−1
2	2	2
−1	−1	−1

Horizontal

2	−1	−1
−1	2	−1
−1	−1	2

+45°

−1	2	−1
−1	2	−1
−1	2	−1

Vertical

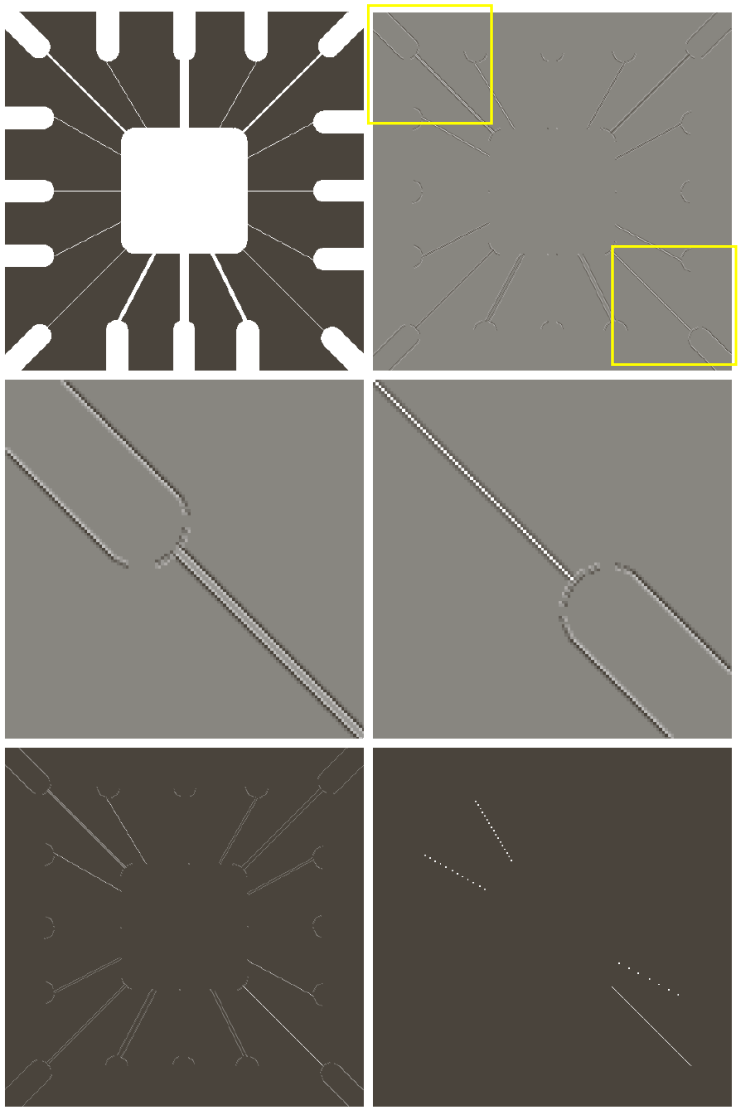
−1	−1	2
−1	2	−1
2	−1	−1

−45°

Line Detection (线检测)

2	−1	−1
−1	2	−1
−1	−1	2

+45°

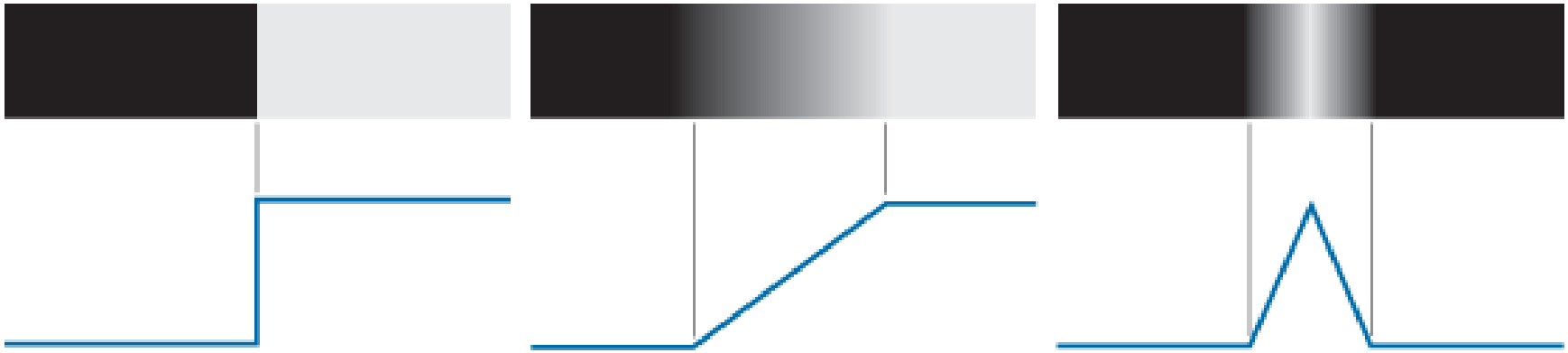


Take only positive

G>T, T = 254

Edge Detection(边缘检测)

- What is an edge?



step ramp roof

Edge Detection

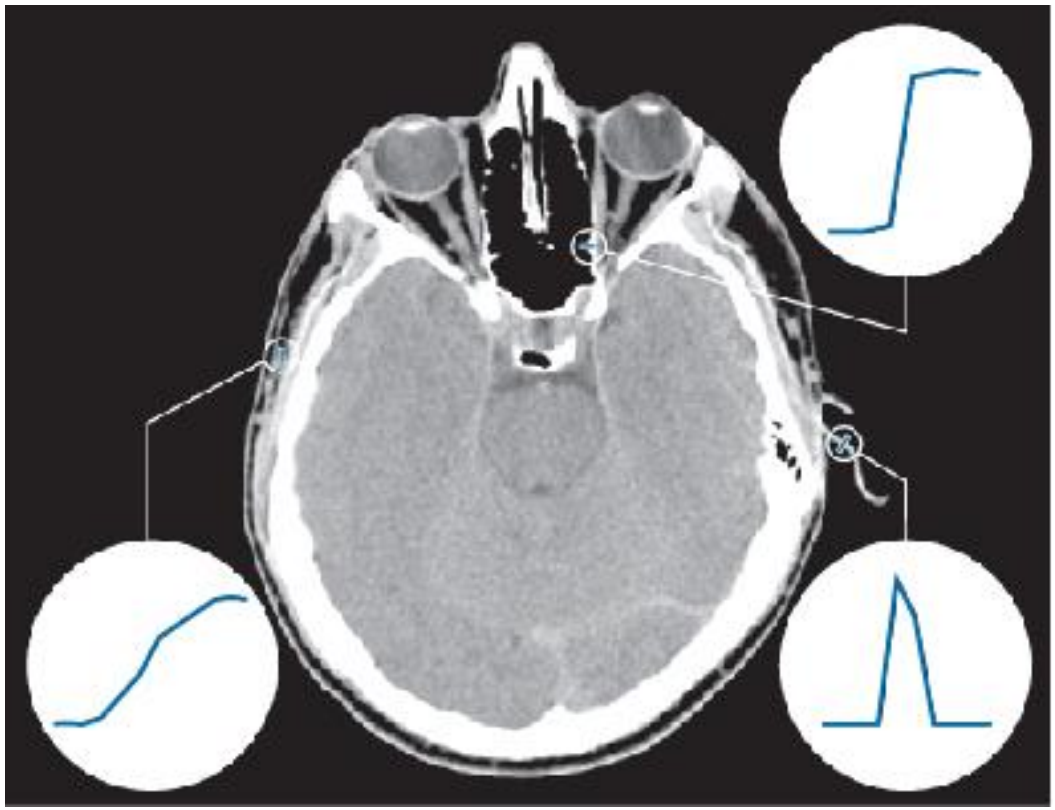
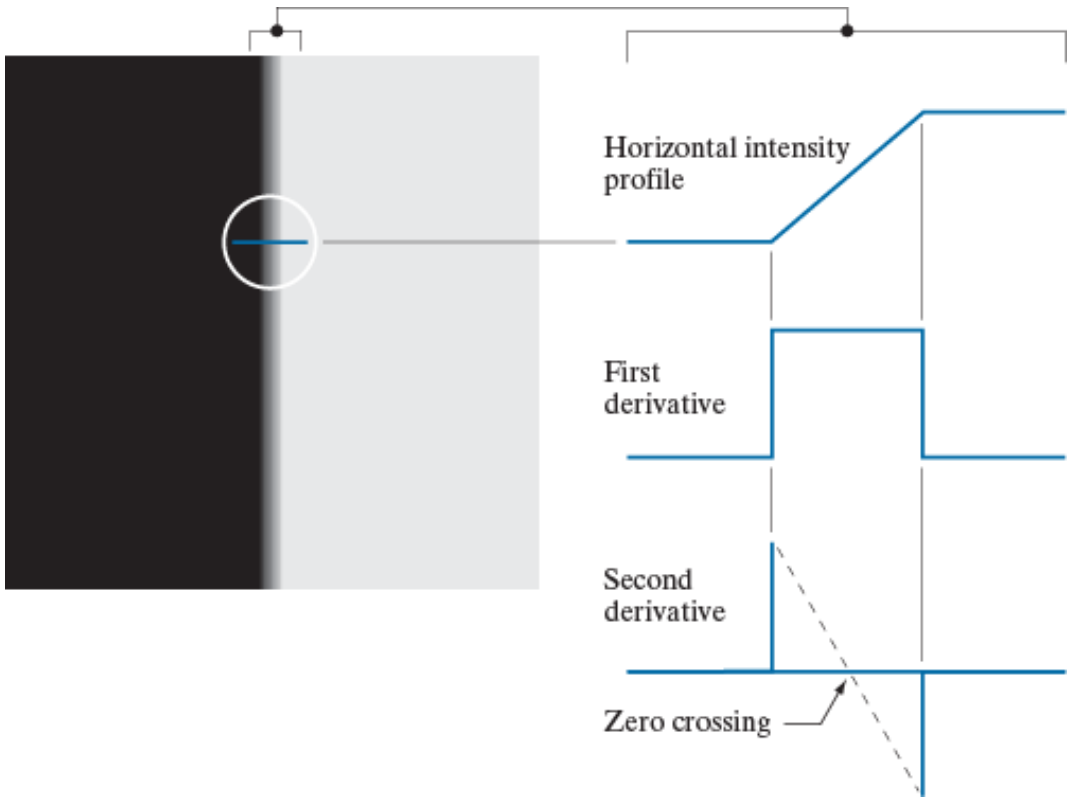


FIGURE 10.9 A 1508 × 1970 image showing (zoomed) actual ramp (bottom, left), step (top, right), and roof edge profiles. The profiles are from dark to light, in the areas enclosed by the small circles. The ramp and step profiles span 9 pixels and 2 pixels, respectively. The base of the roof edge is 3 pixels. (Original image courtesy of Dr. David R. Pickens, Vanderbilt University.)

Edge Detection

- 1st derivative v.s. 2nd derivative



Edge Detection

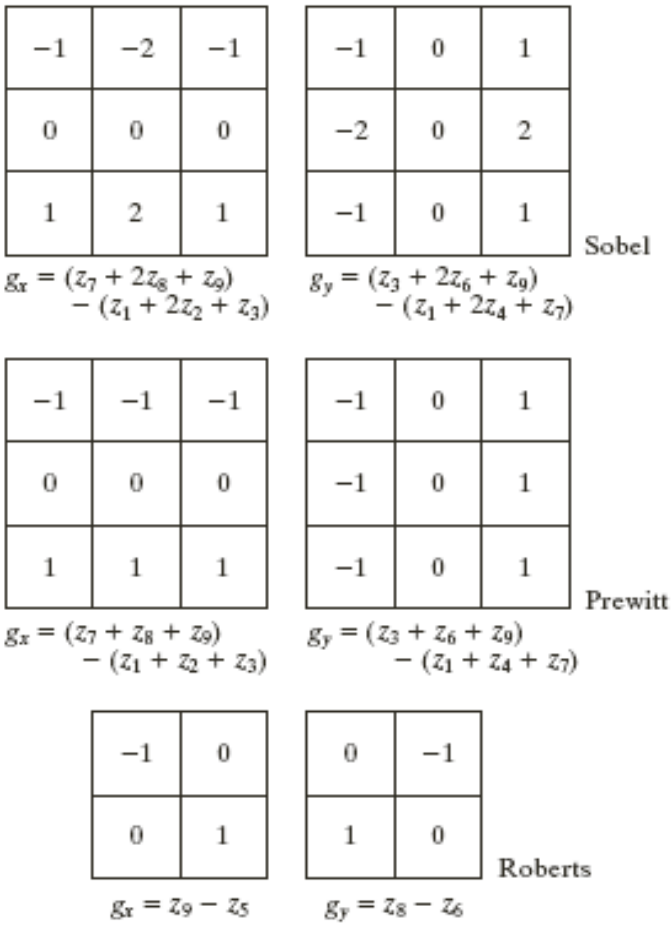
- **Rules:**
  - Seek the 1st derivative greater than a threshold
  - Seek the zero-crossing point on the 2nd derivative
- **General Steps:**
  1) Image smoothing for noise reduction
  2) Detection of edge points
  3) Edge localization

Edge Detection

Matlab function: [g, t] = edge(f, 'method', parameters)

Edge Detector	Description
Sobel	Finds edges using the Sobel approximation to the derivatives in Fig. 10.5(b)
Prewitt	Finds edges using the Prewitt approximation to the derivatives in Fig. 10.5(c).
Roberts	Finds edges using the Roberts approximation to the derivatives in Fig. 10.5(d).
Laplacian of a Gaussian (LoG)	Finds edges by looking for zero crossings after filtering $f(x, y)$ with a Laplacian of a Gaussian filter.
Zero crossings	Finds edges by looking for zero crossings after filtering $f(x, y)$ with a specified filter.
Canny	Finds edges by looking for local maxima of the gradient of $f(x, y)$. The gradient is calculated using the derivative of a Gaussian filter. The method uses two thresholds to detect strong and weak edges, and includes the weak edges in the output only if they are connected to strong edges. Therefore, this method is more likely to detect true weak edges.

Edge Detection



Matlab function:

- Sobel Edge Detector

  [g, t] = edge(f, 'sobel', T, dir)

- Prewitt Edge Detector

  [g, t] = edge(f, 'prewitt', T, dir)

- Roberts Edge Detector

  [g, t] = edge(f, 'Roberts', T, dir)

Gradients

$$\nabla F = grad\,(F\,) = \begin{bmatrix} \frac{\partial F}{\partial x} \\ \frac{\partial F}{\partial y} \end{bmatrix} = \begin{bmatrix} g_x \\ g_y \end{bmatrix} \approx \begin{bmatrix} F(x+1,y) - F(x,y) \\ F(x,y+1) - F(x,y) \end{bmatrix}$$

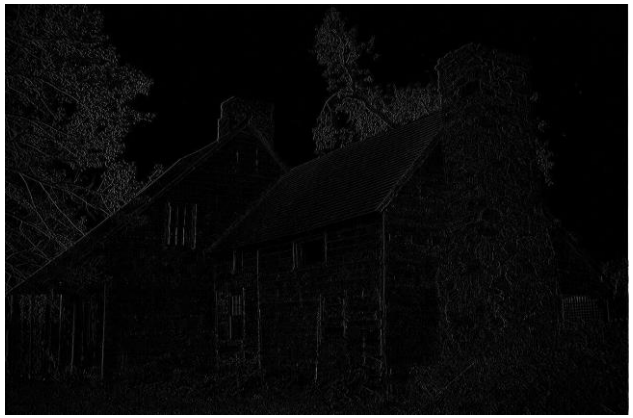
∇F is perpendicular to edge.

$$M(x,y) = \sqrt{(g_x^2 + g_y^2)}$$

$$\alpha(x,y) = tan^{-1}(g_y/g_x)$$

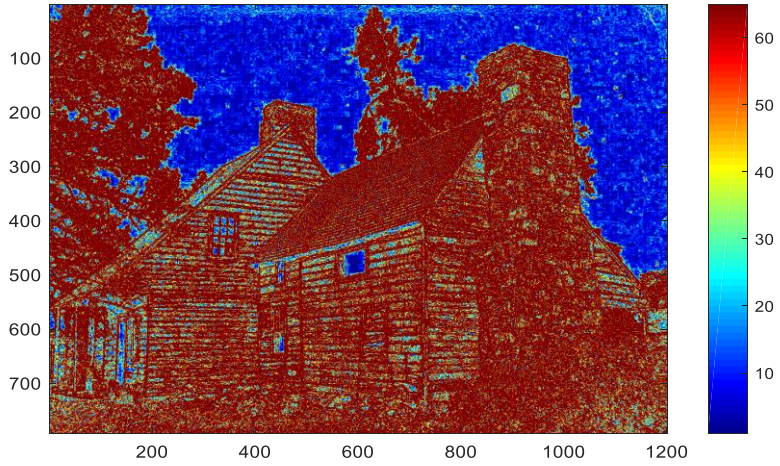
A simple real-world example

- Load image “Peter_Burr_House.jpg”.
- Generate a prewitt/sobel filter for x-direction and a prewitt/sobel for y-direction.
- Apply filters to image and show them.


A simple real-world example

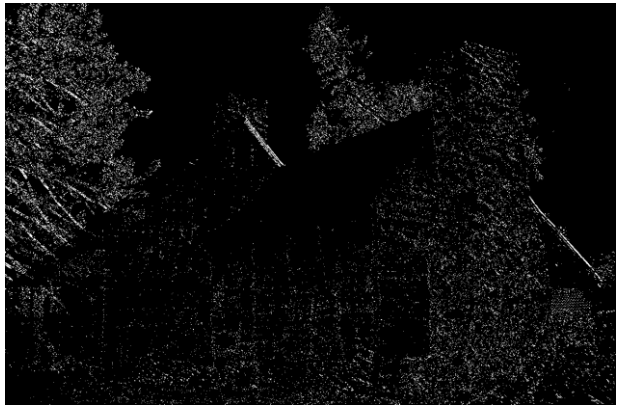
- Load image “Peter_Burr_House.jpg”.
- Generate a sobel/prewitt filter for x-direction and a sobel/prewitt for y-direction.
- Compute the Magnitude and angles of edge gradients.



A simple real-world example

- Load image “Peter_Burr_House.jpg”.
- Generate a sobel/prewitt filter for x-direction and a sobel/prewitt for y-direction.
- Compute the Magnitude and angles of edge gradients.
- Pick out the edges that gradient magnitude is greater than $\tau 1$ and angle is around $\alpha 1$.





```
figure;
edge = (abs(angle+90)<20) & (mag>150);
imshow(edge);
```

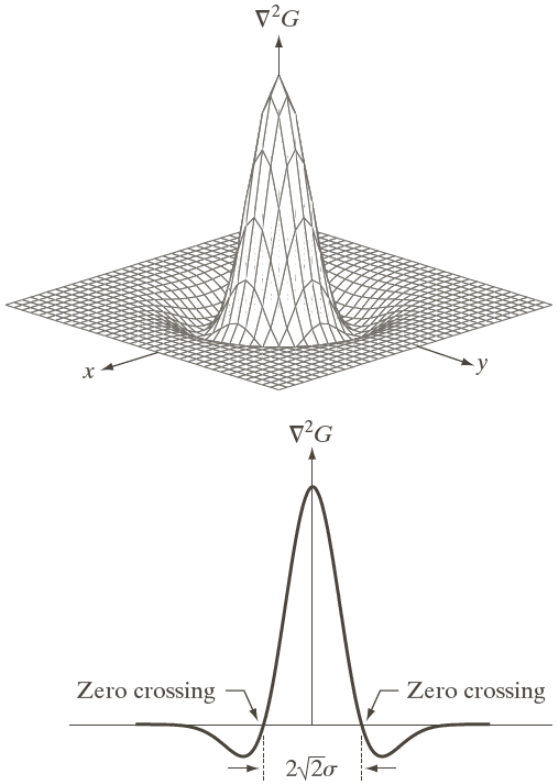
Edge Detectors

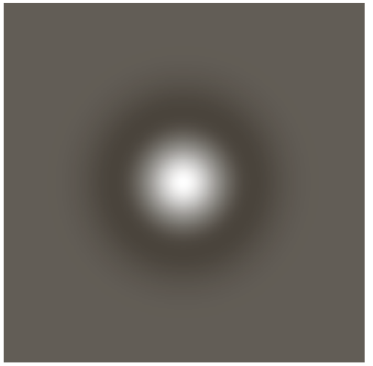
- **LoG (Laplacian of a Gaussian, 高斯拉普拉斯算子):**

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

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

$$= \left[\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$





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

- **Try to visualize a LoG:**
- Matlab code:

```
h1=fspecial('log',[101,101],3/10/30);
surf(h1,'edgecolor','none');
```

Canny Edge Detectors

- **Canny Detector (坎尼边缘检测器):**

  1) Smooth the input image with a Gaussian filter;

  2) Compute the gradient magnitude and angle images;

  3) Apply nonmaxima suppression (非最大值抑制) to the gradient magnitude image;

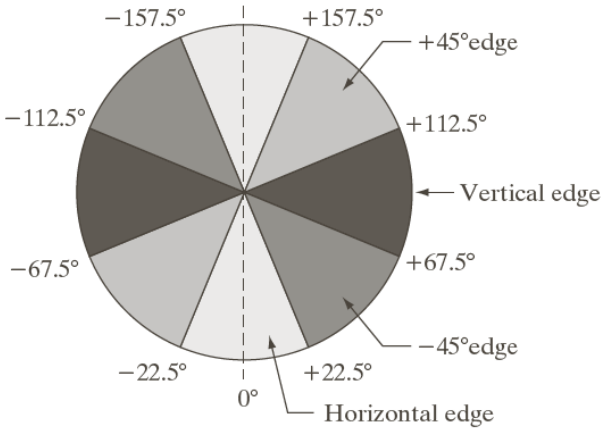
  4) Use double thresholding and connectivity analysis to detect and link edge.

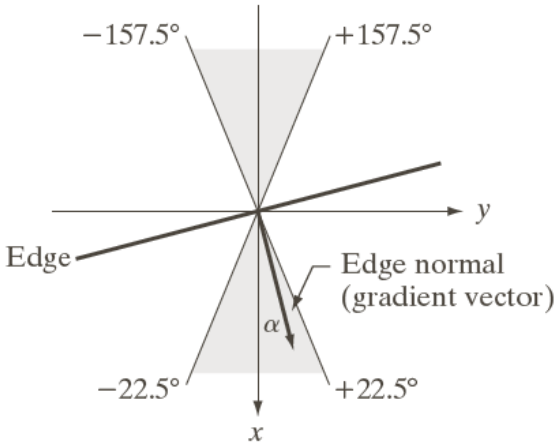
- **Matlab function:** [g, t] = edge(f, 'canny', T, sigma), where T=[T1, T2]

Non-maxima suppression （非最大值抑制）

1) Orientation quantize:

Input： image gradient magnitude and angle map;





2) Non-maxima suppression

If M(x,y) is greater than all its neighbors in the quantized edge direction, $G_N(x,y) = M(x,y)$, otherwise $G_N(x,y) = 0$.

Example

- **Detect and link edges.**

M(x,y)

17	15	30
10	25	20
17	15	10

A(x,y)

2	1	2
2	1	1
2	2	1

Double thresholding edge linking

- **Detect and link edges.**

$$G_H(x,y) = G_N(x,y) \geq T_2 \qquad \text{Strong edge}$$

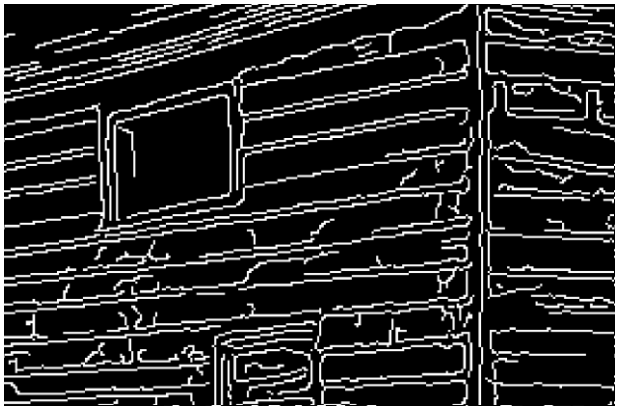
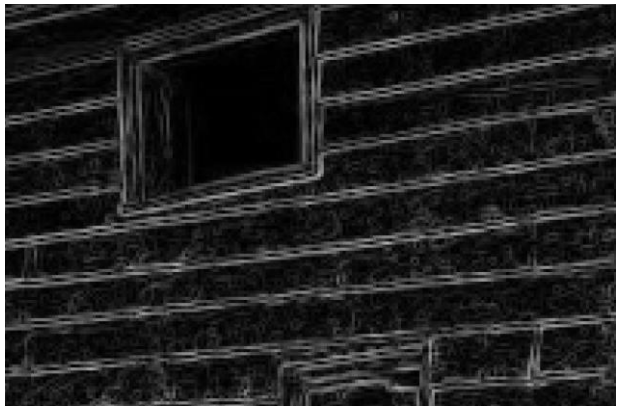
$$G_L(x,y) = T_1 \leq G_N(x,y) \leq T_2 \qquad \text{Weak edge}$$

Final map: $G_H(x,y)$ **and** all edges in $G_L(x,y)$ that are adjacent to at least one pixel of $G_H(x,y)$

- **Matlab function:** [g, t] = edge(f, 'canny', T, sigma), where T=[T1, T2]

Canny Edge Detectors




Canny Edge Detectors

Sigma = 1

Sigma = 2

Sigma = 4

Edge Linking

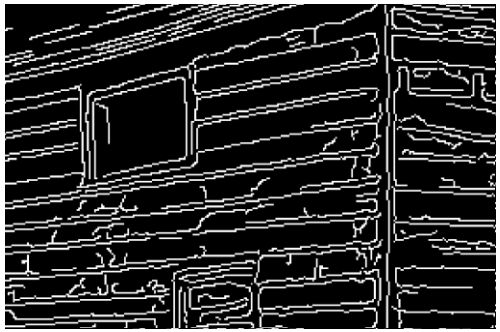
- Following the previous step: edge detector
- Edge Linking:

1. Start with edge pixels and corresponsding $M(x, y)$ and $\alpha(x, y)$;
2. Idea: for each edge pixel (x, y) make a window S_{xy} around that pixel for each $(s, t) \in S_{xy}$, "Link" (x, y) to (s, t) if

$$|M(x, y) - M(s, t)| \leq \tau_1$$

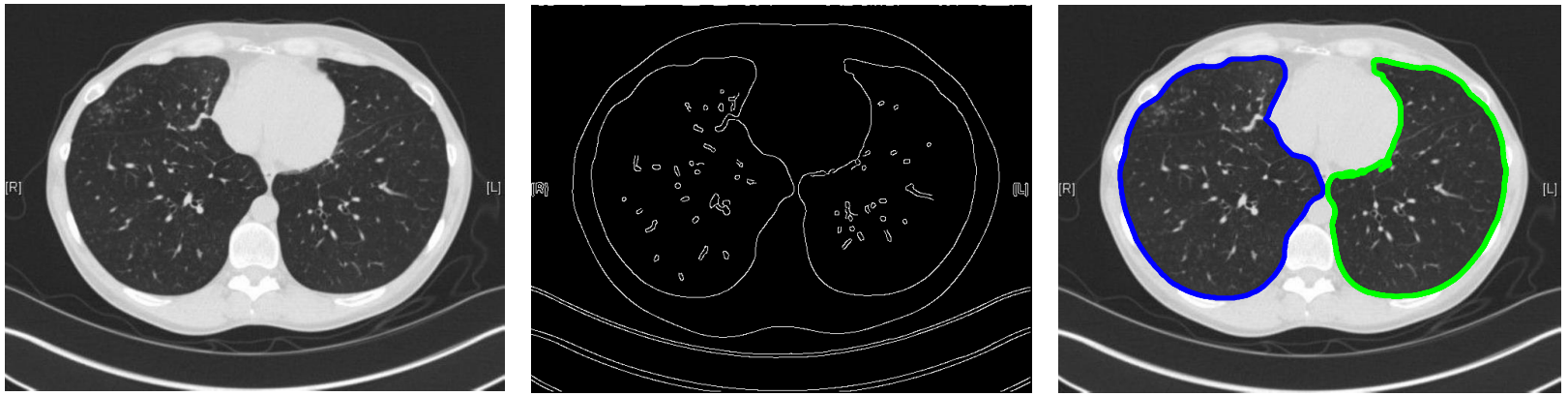
$$|\alpha(x, y) - \alpha(s, t)| \leq \tau_2$$

To take out longer edges

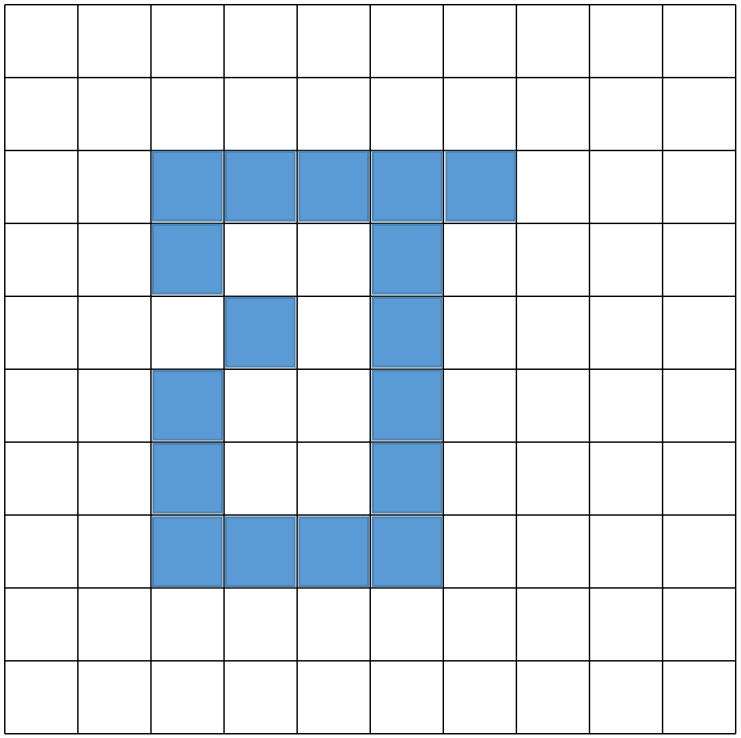


Boundary following

- We have edge point around a closed contour, we want to link/order them in a clock wise direction.



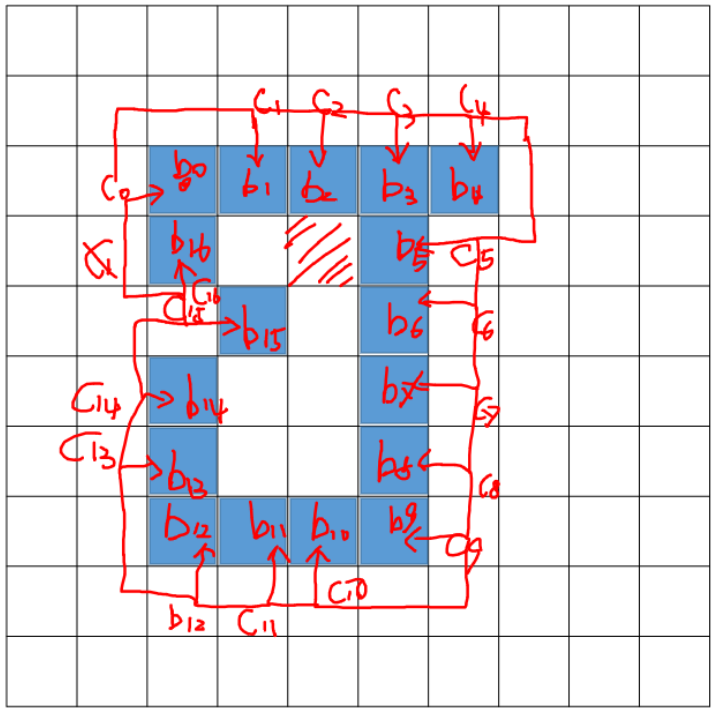
Boundary following



- **Moore's boundary following algorithms:**

1. Start with edge maps (binary).
2. Let starting point b_0 be the uppermost, leftmost point labelled "1". Let c_0 be the left neighbor of b_0.
3. Examine 8-neighbors of b_0, starting at c_0, and going clock-wise. Let b_1 be the first 1 pixel and c_1 be the preceding 0 pixel.
4. Let $b = b_1, c = c_1$.
5. Continue until $b = b_0$, and next bounding point found is b_1. Or until there is no edge point in the 8-neighbor of b.
6. The opened list of b is the boundary.

Boundary following

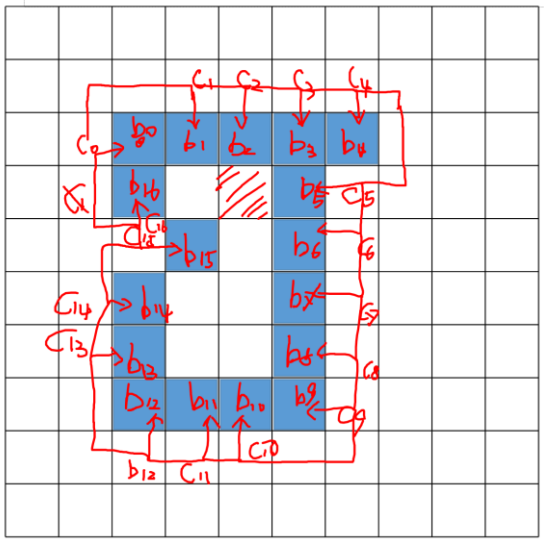


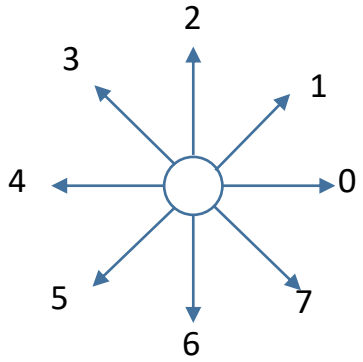
- **Moore's boundary following algorithms:**

1. Start with edge maps (binary).
2. Let starting point b_0 be the uppermost, leftmost point labelled "1". Let c_0 be the left neighbor of b_0.
3. Examine 8-neighbors of b_0, starting at c_0, and going clock-wise. Let b_1 be the first 1 pixel and c_1 be the preceding 0 pixel.
4. Let $b = b_1, c = c_1$.
5. Continue until $b = b_0$, and next bounding point found is b_1. Or until there is no edge point in the 8-neighbor of b.
6. The opened list of b is the boundary.

Boundary following

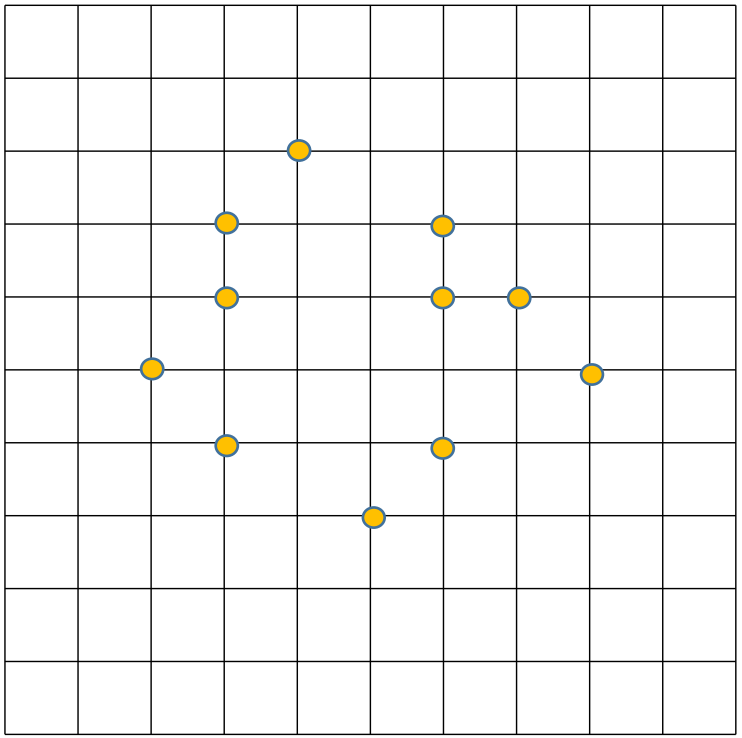
- Describe the boundary with a chain code:
  - Define 3-bit direction, corresponding to previous boundary point.





b0	b1	b2	b3	b4	b5	b6	b7	b8	b9
Direction for next P	0	0	0	0	5	6	6	6	6
ΔD	--	0	0	0	5	-1	0	0	0

Polygonal fitting

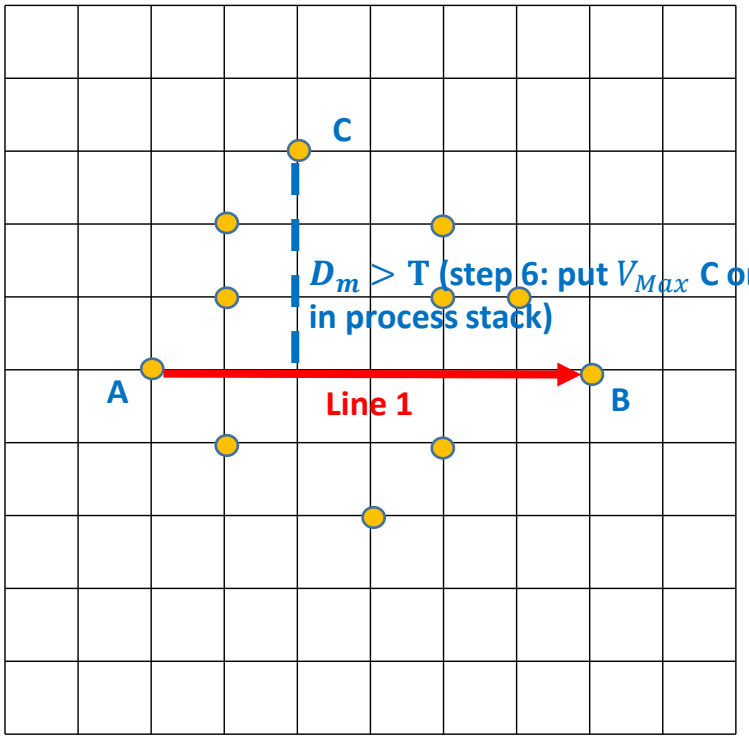


Polygonal fitting

- **Fitting a set of ordered points (find windows/doors)**

1. Let P be a sequence of ordered, distinct points. (e.g. ordered edges after boundary following).
2. Specify two starting points A, B.
3. Specify a threshould T (pixel distance).
4. Creating the stacks: [final] and [in process].
5. Compute the distance from this line to all the points between these vertices. Select vertex V_{max} with the max distance D_m.

Polygonal fitting



1. Let P be a sequence of ordered, distinct points. (e.g. ordered edges after boundary following).

2.Specify two starting points A,B with largest distance among all points.

3.Specify a threshold T (pixel distance).

4.Creating the stacks using the two starting points: for example [final] as A and [in process] A,B. Then connect the vertices on top of each stack with a **directed line**.

5. Compute the distance from this line to all the points in the **clock-wise or anti-clock-wise** side of the directed line between these vertices. Select vertex V_{Max} with the max distance D_m.

6. If $D_m > \mathrm{T}$ (a threshold set), put V_{Max} at the end of [in process], and go to step 4.

7. Otherwise, remove the last vertex from [in process] and make it the last vertex in [final].

8. If [in process] is not empty, go to step 4.

9. other wise, done. The vertices in are the ordered vertices of a polygonal.

Polygonal fitting

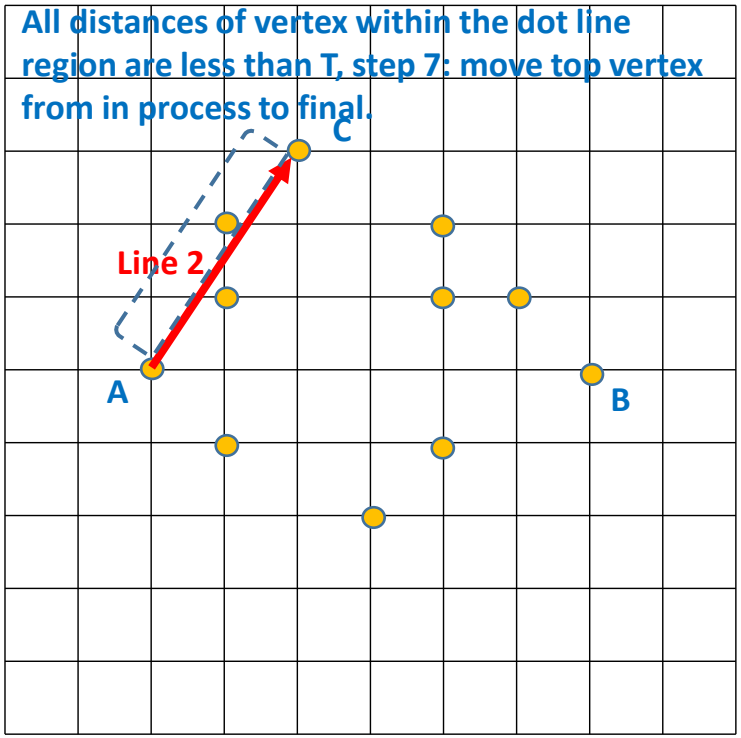
final

A

A

AC

iter2



In process

AB

ABC

AB

1. Let P be a sequence of ordered, distinct points. (e.g. ordered edges after boundary following).

2.Specify two starting points A,B with largest distance among all points.

3.Specify a threshold T (pixel distance).

4.Creating the stacks using the two starting points: for example [final] as A and [in process] A,B. Then connect the vertices on top of each stack with a **directed line**.

5. Compute the distance from this line to all the points in the **clock-wise or anti-clock-wise** side of the directed line between these vertices. Select vertex V_{Max} with the max distance D_m.

6. If $D_m > \mathrm{T}$ (a threshold set), put V_{Max} at the end of [in process], and go to step 4.

7. Otherwise, remove the last vertex from [in process] and make it the last vertex in [final].

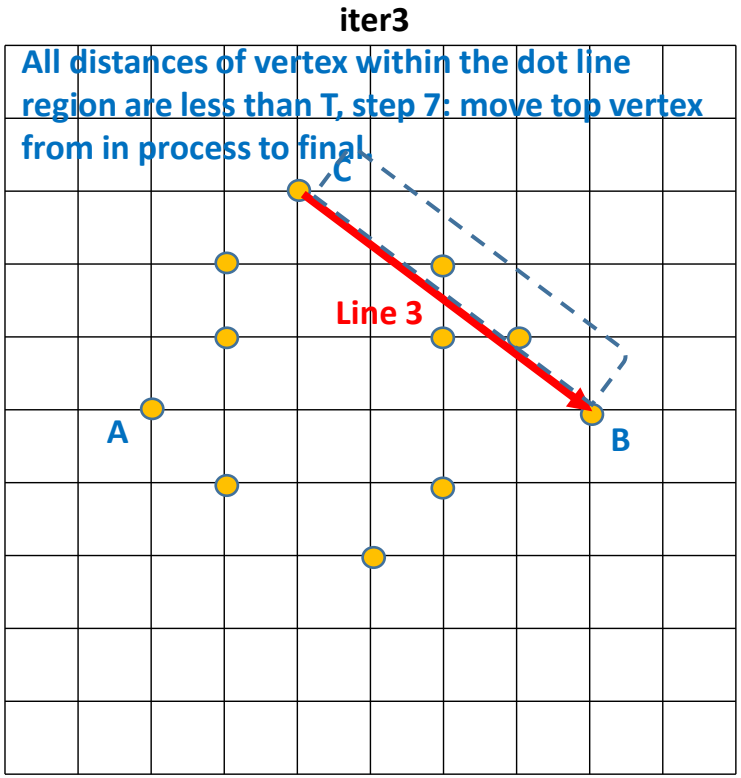
8. If [in process] is not empty, go to step 4.

9. other wise, done. The vertices in are the ordered vertices of a polygonal.

Polygonal fitting

final

A
A
AC
ACB



In process

AB
ABC
AB
A

1. Let P be a sequence of ordered, distinct points. (e.g. ordered edges after boundary following).

2.Specify two starting points A,B with largest distance among all points.

3.Specify a threshold T (pixel distance).

4.Creating the stacks using the two starting points: for example [final] as A and [in process] A,B. Then connect the vertices on top of each stack with a **directed line**.

5. Compute the distance from this line to all the points in the **clock-wise or anti-clock-wise** side of the directed line between these vertices. Select vertex V_{Max} with the max distance D_m.

6. If $D_m > \mathrm{T}$ (a threshold set), put V_{Max} at the end of [in process], and go to step 4.

7. Otherwise, remove the last vertex from [in process] and make it the last vertex in [final].

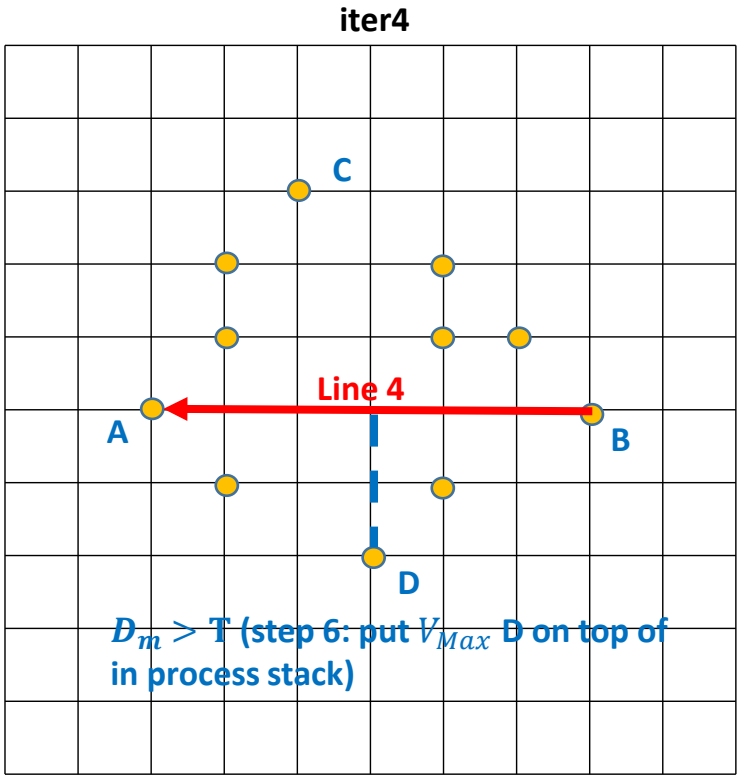
8. If [in process] is not empty, go to step 4.

9. other wise, done. The vertices in are the ordered vertices of a polygonal.

Polygonal fitting

final

A
A
AC
ACB
ACB



In process

AB
ABC
AB
A
AD

1. Let P be a sequence of ordered, distinct points. (e.g. ordered edges after boundary following).

2.Specify two starting points A,B with largest distance among all points.

3.Specify a threshold T (pixel distance).

4.Creating the stacks using the two starting points: for example [final] as A and [in process] A,B. Then connect the vertices on top of each stack with a **directed line**.

5. Compute the distance from this line to all the points in the **clock-wise or anti-clock-wise** side of the directed line between these vertices. Select vertex V_{Max} with the max distance D_m.

6. If $D_m > T$ (a threshold set), put V_{Max} at the end of [in process], and go to step 4.

7. Otherwise, remove the last vertex from [in process] and make it the last vertex in [final].

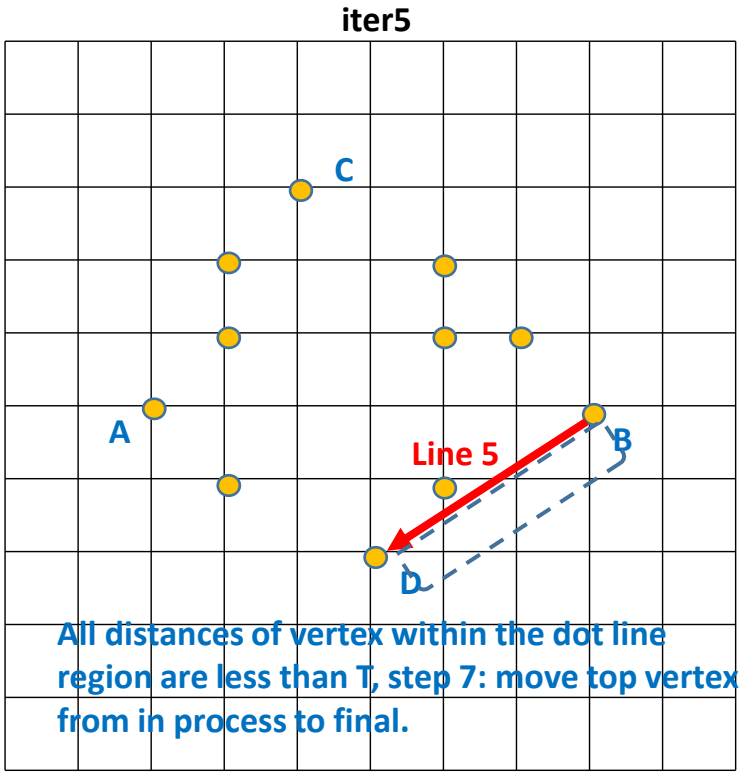
8. If [in process] is not empty, go to step 4.

9. other wise, done. The vertices in are the ordered vertices of a polygonal.

Polygonal fitting

final

A
A
AC
ACB
ACB
ACBD



In process

AB
ABC
AB
A
AD
A

1. Let P be a sequence of ordered, distinct points. (e.g. ordered edges after boundary following).

2.Specify two starting points A,B with largest distance among all points.

3.Specify a threshold T (pixel distance).

4.Creating the stacks using the two starting points: for example [final] as A and [in process] A,B. Then connect the vertices on top of each stack with a **directed line**.

5. Compute the distance from this line to all the points in the **clock-wise or anti-clock-wise** side of the directed line between these vertices. Select vertex V_{Max} with the max distance D_m.

6. If $D_m > \mathrm{T}$ (a threshold set), put V_{Max} at the end of [in process], and go to step 4.

7. Otherwise, remove the last vertex from [in process] and make it the last vertex in [final].

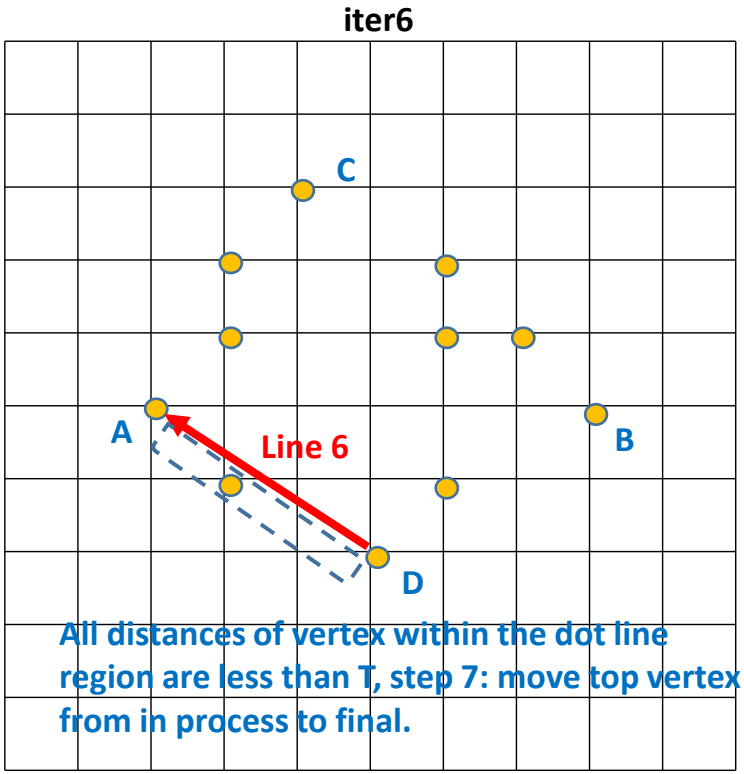
8. If [in process] is not empty, go to step 4.

9. other wise, done. The vertices in are the ordered vertices of a polygonal.

Polygonal fitting

final

A
A
AC
ACB
ACB
ACBD
ACBDA



In process

AB
ABC
AB
A
AD
A
null

1. Let P be a sequence of ordered, distinct points. (e.g. ordered edges after boundary following).

2.Specify two starting points A,B with largest distance among all points.

3.Specify a threshold T (pixel distance).

4.Creating the stacks using the two starting points: for example [final] as A and [in process] A,B. Then connect the vertices on top of each stack with a **directed line**.

5. Compute the distance from this line to all the points in the **clock-wise or anti-clock-wise** side of the directed line between these vertices. Select vertex V_{Max} with the max distance D_m.

6. If $D_m > \mathrm{T}$ (a threshold set), put V_{Max} at the end of [in process], and go to step 4.

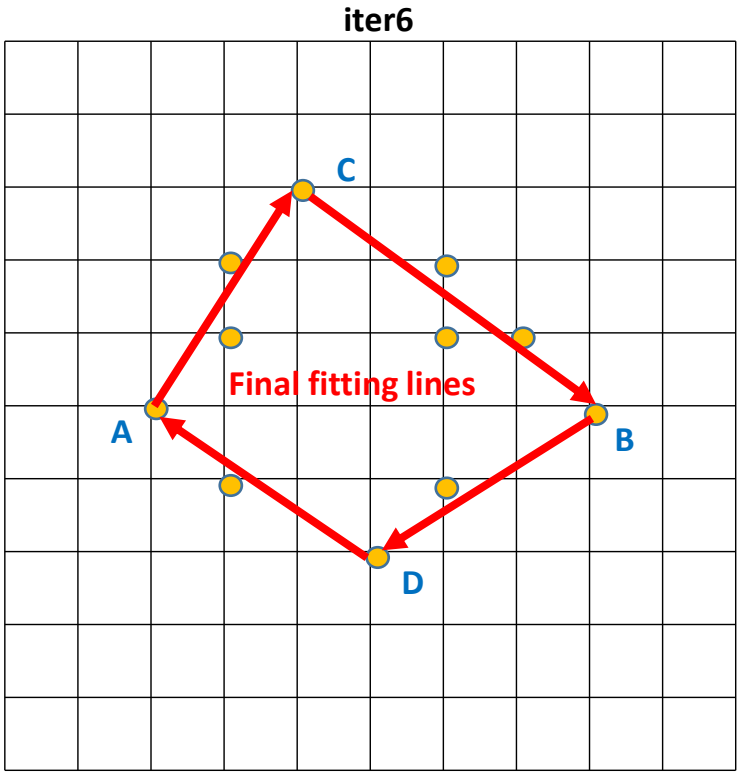
7. Otherwise, remove the last vertex from [in process] and make it the last vertex in [final].

8. If [in process] is not empty, go to step 4.

9. other wise, done. The vertices in are the ordered vertices of a polygonal.

Polygonal fitting

final	In process
A	AB
A	ABC
AC	AB
ACB	A
ACB	AD
ACBD	A
ACBDA	null



1. Let P be a sequence of ordered, distinct points. (e.g. ordered edges after boundary following).

2.Specify two starting points A,B with largest distance among all points.

3.Specify a threshold T (pixel distance).

4.Creating the stacks using the two starting points: for example [final] as A and [in process] A,B. Then connect the vertices on top of each stack with a **directed line**.

5. Compute the distance from this line to all the points in the **clock-wise or anti-clock-wise** side of the directed line between these vertices. Select vertex V_{Max} with the max distance D_m.

6. If $D_m > \mathrm{T}$ (a threshold set), put V_{Max} at the end of [in process], and go to step 4.

7. Otherwise, remove the last vertex from [in process] and make it the last vertex in [final].

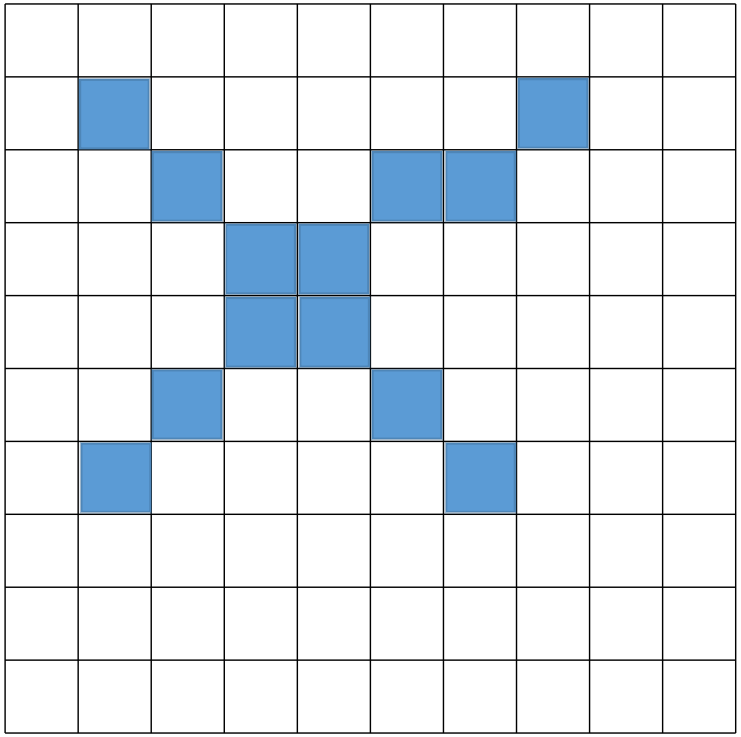
8. If [in process] is not empty, go to step 4.

9. other wise, done. The vertices in are the ordered vertices of a polygonal.

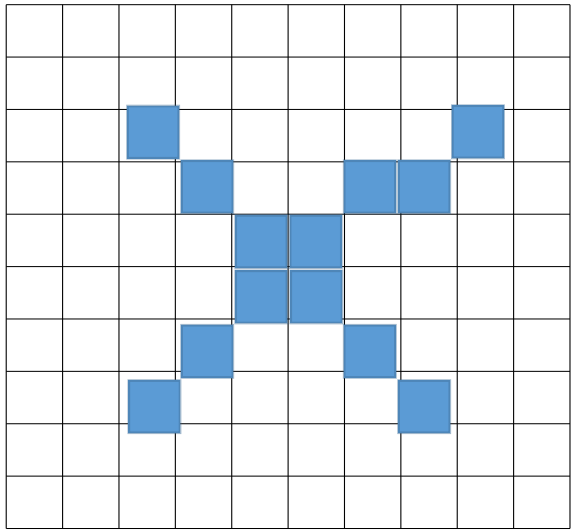
Polygonal fitting

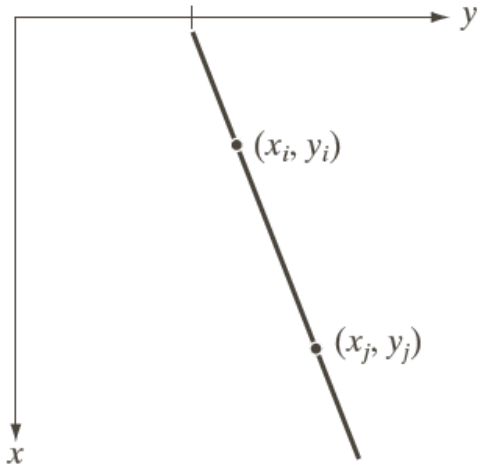
- If the threshold T is small, we will have a polygon with many vertices and smooth fitting.
- Otherwise, a polygon fitting with simple structure and large error.

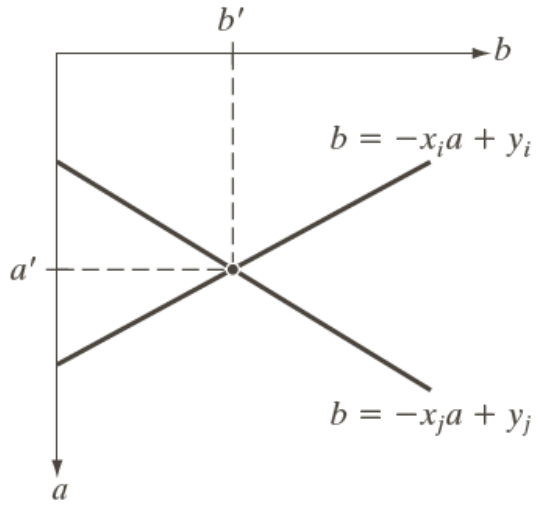
Another challenge

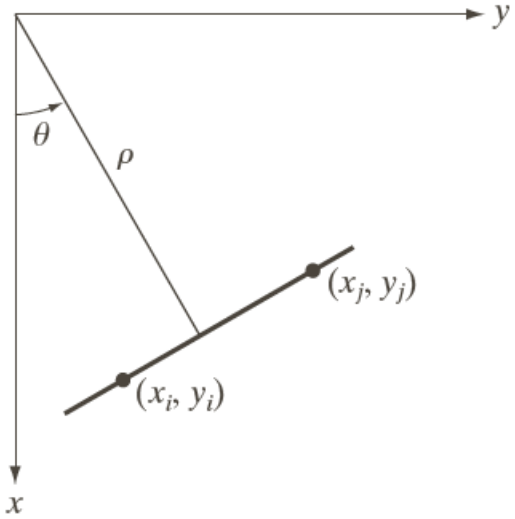


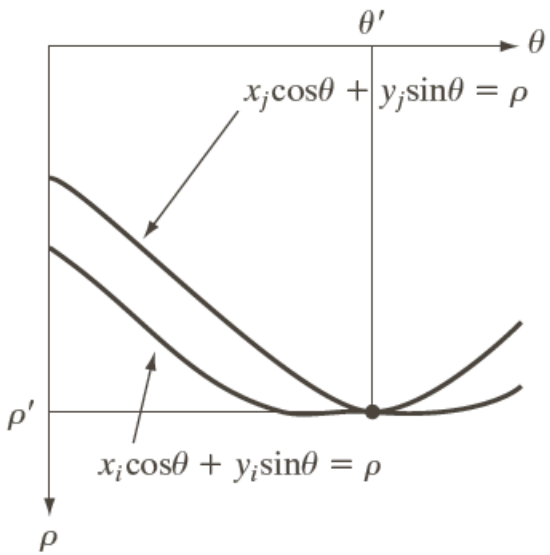
Hough Transform (霍夫变换)

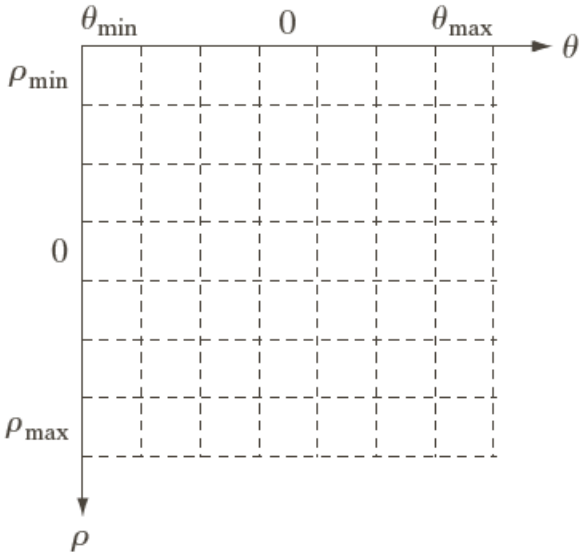












Hough Transform (霍夫变换)

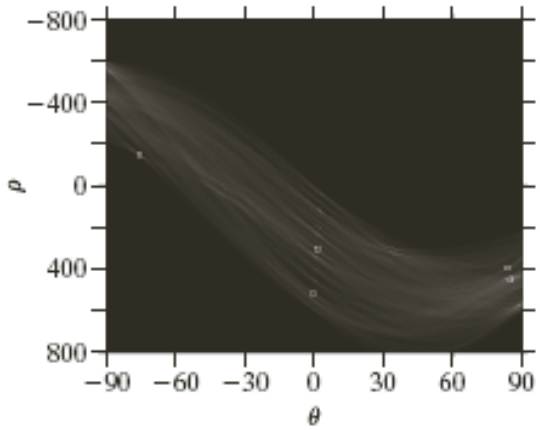
- **An approach based on Hough Transform**

  1. Obtain a binary edge image using any edge detector;
  2. Specify subdivisions in the $\rho\theta$-plane;
  3. Examine the counts of the accumulator cells (累加器单元) for high pixel concentrations;
  4. Examine the relationship between pixels in a chosen cell.

- **Matlab function:**

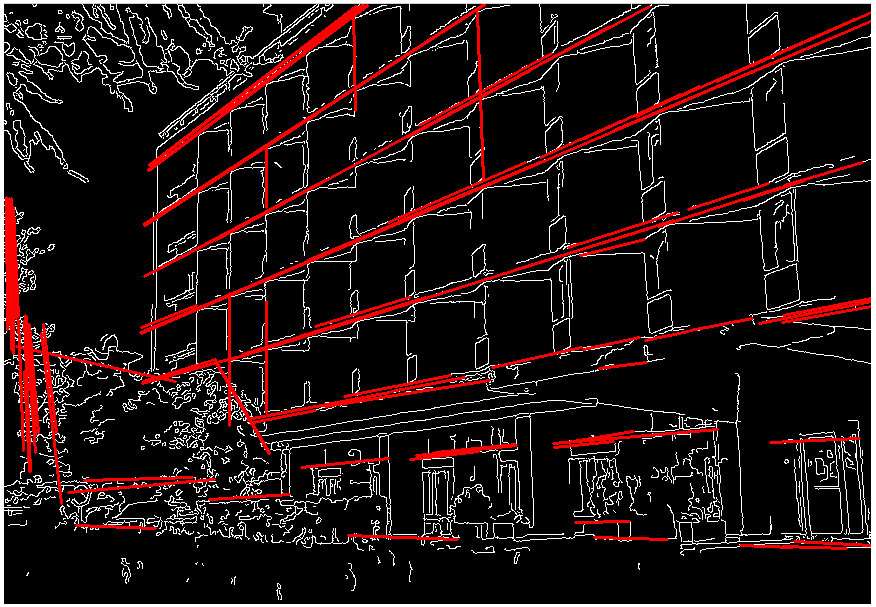
```
[H, theta, rho] = hough(f);

peaks = houghpeaks(H, NumPeaks)

lines = houghlines(f, theta, rho, peaks)
```

Hough Transform (霍夫变换)



Take home message

- Boundary detection & Global structure detection:



Threshold

- Global thresholding (全局阈值处理)
  - Basic global thresholding
  - Optimal global thresholding using Otsu's method
  - Improve global thresholding by using edges
  - Multiple thresholding
- Variable thresholding （可变阈值处理）
  - Image partitioning （图像分块）
  - Variable thresholding based on local image properties

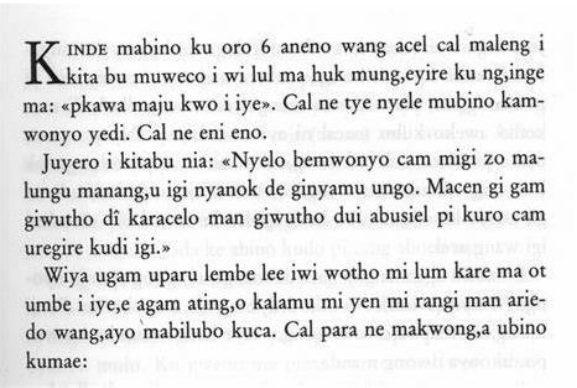
Simple thresholding

- **Definition**

$$g(x,y) = \begin{cases} 1, & f(x,y) > T \quad \text{(object points)} \\ 0, & f(x,y) \le T \quad \text{(background points)} \end{cases}$$

```
foreground = rgb2gray(im)<150;
background = rgb2gray(im)>150;
```

Input image

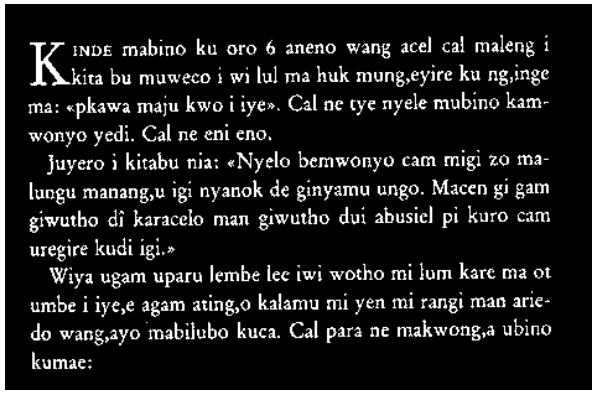


KINDE mabino ku oro 6 aneno wang acel cal maleng i kita bu muweco i wi lul ma huk mung,eyire ku ng,inge ma: «pkawa maju kwo i iye». Cal ne tye nyele mubino kamwonyo yedi. Cal ne eni eno.

Juyero i kitabu nia: «Nyelo bemwonyo cam migi zo malungu manang,u igi nyanok de ginyamu ungo. Macen gi gam giwutho dî karacelo man giwutho dui abusiel pi kuro cam uregire kudi igi.»

Wiya ugam uparu lembe lee iwi wotho mi lum kare ma ot umbe i iye,e agam ating,o kalamu mi yen mi rangi man ariedo wang,ayo mabilubo kuca. Cal para ne makwong,a ubino kumae:

foreground

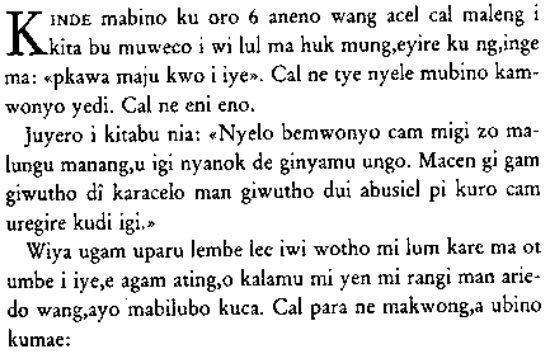


KINDE mabino ku oro 6 aneno wang acel cal maleng i kita bu muweco i wi lul ma huk mung,eyire ku ng,inge ma: «pkawa maju kwo i iye». Cal ne tye nyele mubino kamwonyo yedi. Cal ne eni eno.

Juyero i kitabu nia: «Nyelo bemwonyo cam migi zo malungu manang,u igi nyanok de ginyamu ungo. Macen gi gam giwutho dî karacelo man giwutho dui abusiel pi kuro cam uregire kudi igi.»

Wiya ugam uparu lembe lee iwi wotho mi lum kare ma ot umbe i iye,e agam ating,o kalamu mi yen mi rangi man ariedo wang,ayo mabilubo kuca. Cal para ne makwong,a ubino kumae:

background

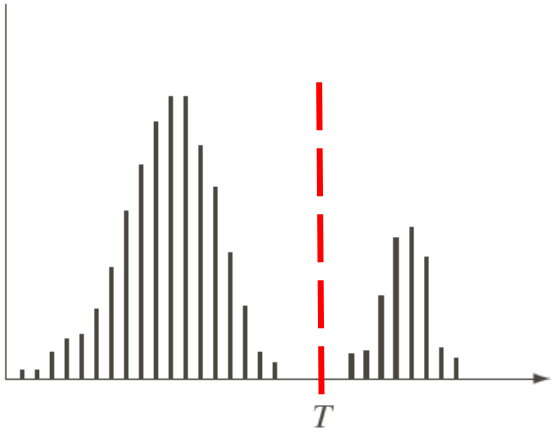


KINDE mabino ku oro 6 aneno wang acel cal maleng i kita bu muweco i wi lul ma huk mung,eyire ku ng,inge ma: «pkawa maju kwo i iye». Cal ne tye nyele mubino kamwonyo yedi. Cal ne eni eno.

Juyero i kitabu nia: «Nyelo bemwonyo cam migi zo malungu manang,u igi nyanok de ginyamu ungo. Macen gi gam giwutho dî karacelo man giwutho dui abusiel pi kuro cam uregire kudi igi.»

Wiya ugam uparu lembe lee iwi wotho mi lum kare ma ot umbe i iye,e agam ating,o kalamu mi yen mi rangi man ariedo wang,ayo mabilubo kuca. Cal para ne makwong,a ubino kumae:

Intensity Valley

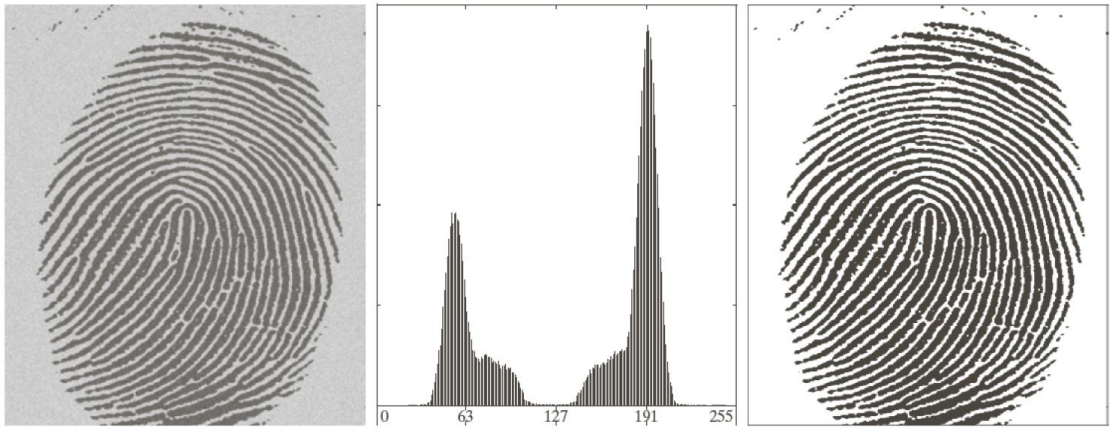


$$g(x,y) = \begin{cases} 1, & f(x,y) > T \quad \text{(object points)} \\ 0, & f(x,y) \le T \quad \text{(background points)} \end{cases}$$

Basic global thresholding

- **Steps:**

1. Select an initial estimate of the global threshold **T**;
2. Segment the image using **T** to two groups G_1(>T) and G_2(⩽T) ;
3. Compute average intensity m_1 and m_2 for G_1 and G_2 respectively;
4. Compute new threshold **T**=(m_1 + m_2)/2;
5. Repeat 2-4 until the difference between **T** in successive iteration is smaller than requirement.



Otsu's method (大津二值化)

- Maximize the between-class variance.
- A good threshold should separate pixels into tight cluster.

[1] Otsu N. A threshold selection method from gray-level histogram. IEEE Trans,1979;SMC-9;62-66

Otsu's method

Image PMF:

p_i = probability that $I(x, y) = i, i = 1,2, \cdots, L-1$

$m_G = \sum_{i=1}^{L-1} i * p_i$

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

where

m_G: average intensity of entire image (global mean).

σ_G: global variance.

Otsu's method

- Suppose we select a threshold T.

$$C_1 = \{(x, y) | I(x, y) < T\}$$

$$C_2 = \{(x, y) | I(x, y) > T\}$$

Then

$$P_1 = \sum_{i=1}^{T} p_i \qquad P_2 = \sum_{i=T+1}^{L-1} p_i = 1 - P_1$$

Class conditional

$$m_1 = \sum_{i=1}^{T} i * p_i \quad m_2 = \sum_{i=T+1}^{L-1} i * p_i$$

Between-class variance is defined as:

$$\sigma_B = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2$$

In practice, we just consider all possible T, and choose the T that maximizes σ_B

Try this

Input image

Output binary map

Fig 3:Histogram with T labeled

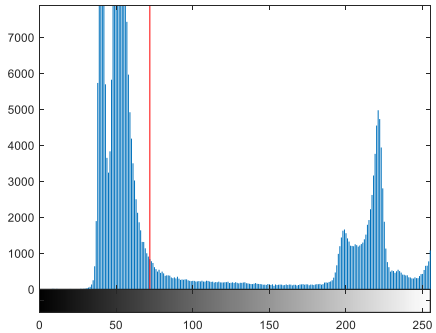
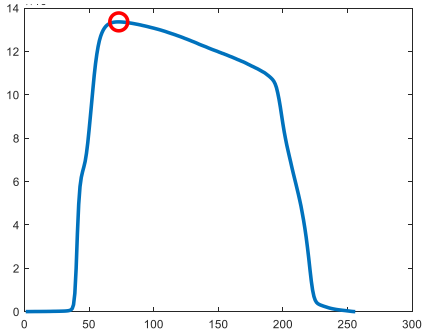


Fig 4:Between-class variance

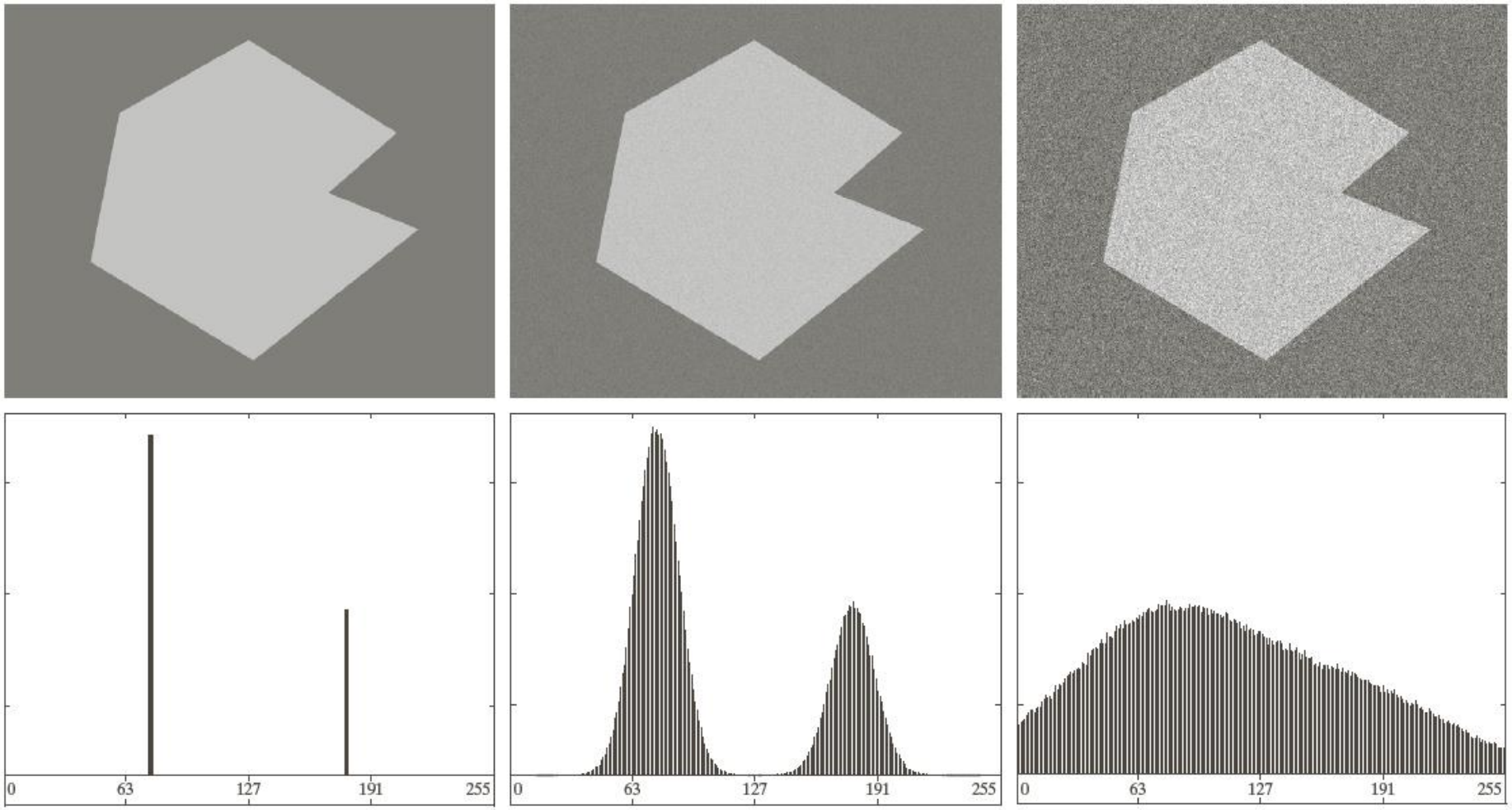


- Find the matlab function [level,EM] = graythresh(I) . Read it and try to modify it to output fig3 and fig4.
- Otsu can fail when:

-no strong peaks in the histogram

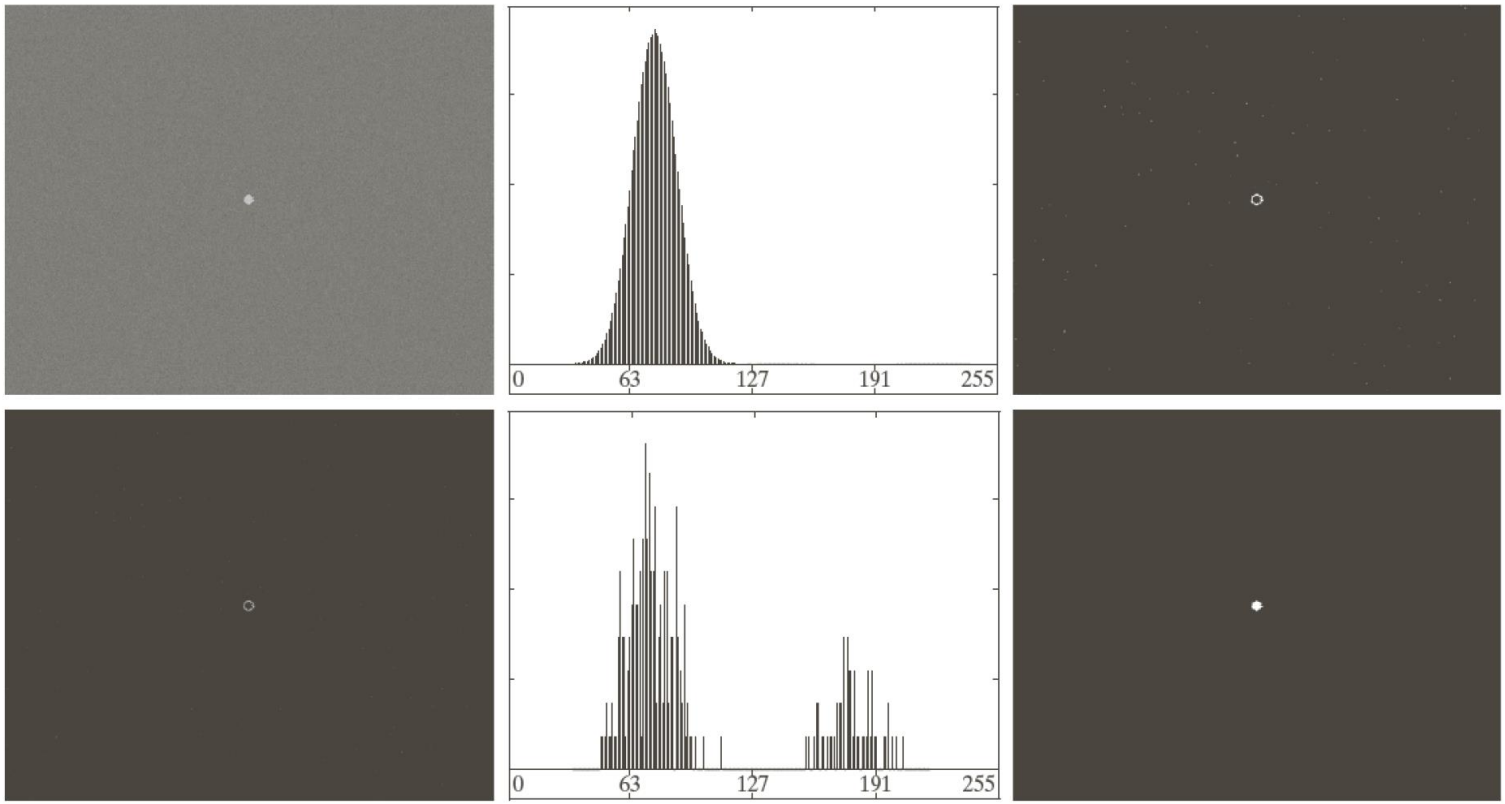
-object is small (with respect to) background

Influence of Noise



Improve Global thresholding

- Using edges:

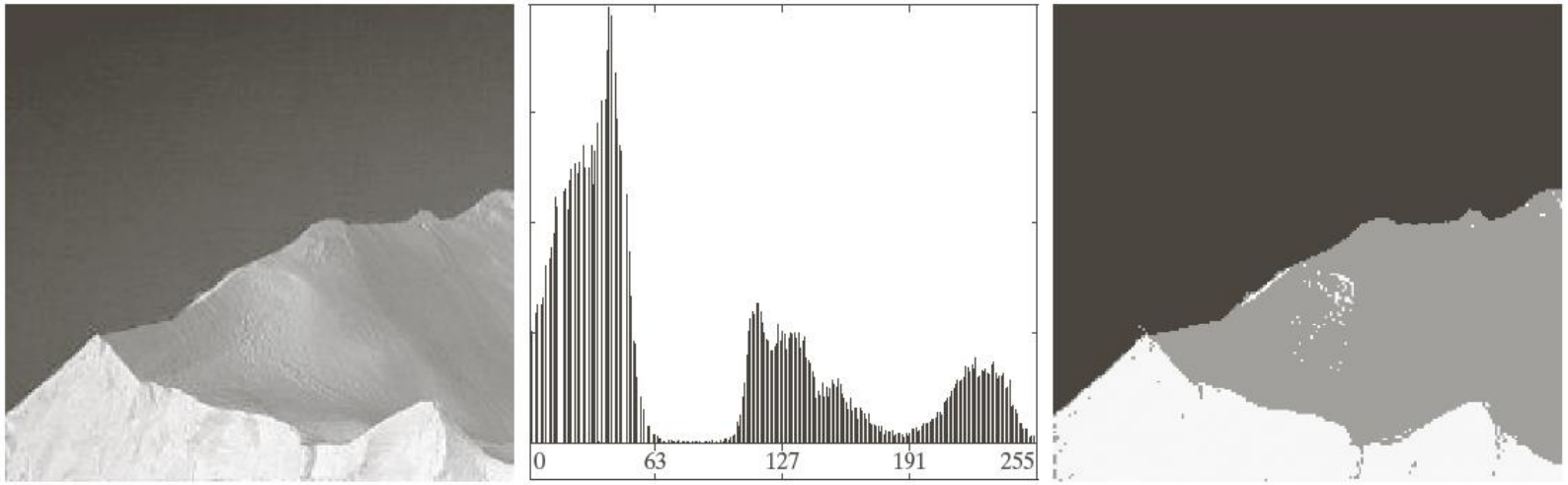


Improve Global thresholding

- **Using edges:**

1. compute an edge image from the input image $f(x, y)$ using any edge detector;
2. specify a threshold value T;
3. Threshold the edge image using T to produce a binary image $g_T(x, y)$
4. compute a histogram using only the pixels in $f(x, y)$ that correspond to the locations of the 1-valued pixels in $g_T(x, y)$
5. use the histogram to segment $f(x, y)$;

Multiple thresholds



Multiple thresholds

- **Between-class variance (类间方差):**

$$\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 + P_3(m_3 - m_G)^2$$

Where

$$P_1 = \sum_{i=0}^{k_1} p_i \qquad P_2 = \sum_{i=k_1+1}^{k_2} p_i \qquad P_3 = \sum_{i=k_2+1}^{L-1} p_i$$

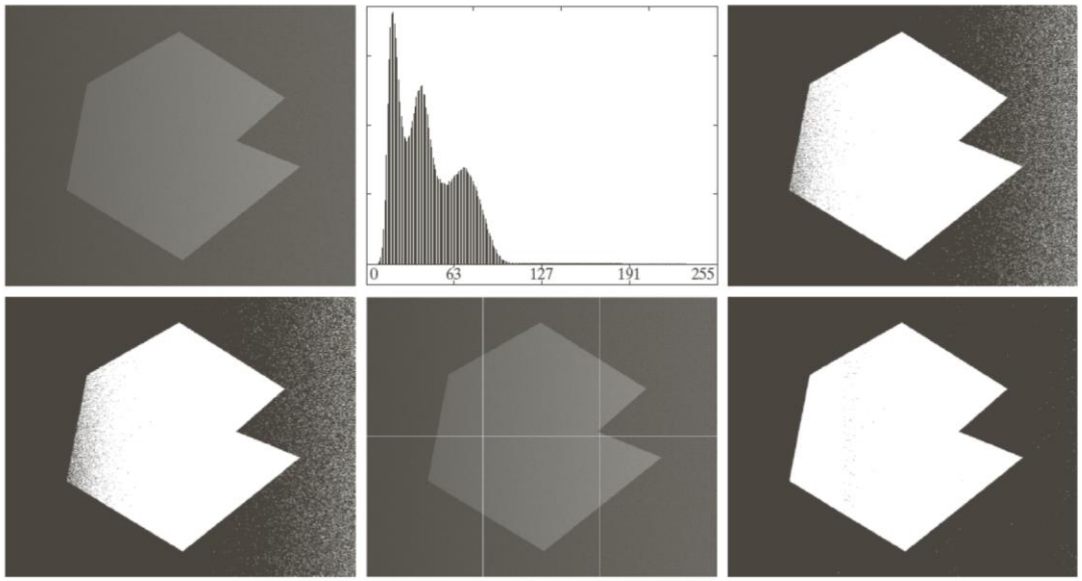
$$m_1 = \sum_{i=0}^{k_1} ip_i \qquad m_2 = \sum_{i=k_1+1}^{k_2} ip_i \qquad m_3 = \sum_{i=k_2+1}^{L-1} ip_i$$

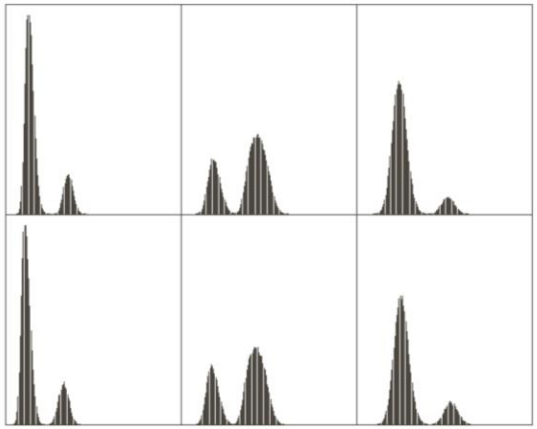
$$P_1 m_1 + P_2 m_2 + P_3 m_3 = m_G \qquad P_1 + P_2 + P_3 = 1$$

The two optimum thresholds k_1^* and k_2^* are the values that maximize $\sigma_B^2(k_1, k_2)$, then

$$g(x,y) = \begin{cases} a, & f(x,y) \le k_1^* \\ b, & k_1^* < f(x,y) \le k_2^* \\ c, & f(x,y) > k_2^* \end{cases} \quad \text{and} \quad \eta(k_1^*, k_2^*) = \frac{\sigma_B^2(k_1^*, k_2^*)}{\sigma_G^2}$$

Image partitioning





Variable thresholding based on local image properties

- We can make the rules like this:

$$g(x,y) = \begin{cases} 1 & I(x,y) > \mu_{xy} + 2\sigma_{xy} \\ 0 & else \end{cases}$$

- Turn this pixel on. If locally brighter than others or

$$g(x,y) = \begin{cases} 1 & I(x,y) > \mu_{xy} \\ 0 & else \end{cases}$$

$$g(x,y) = \begin{cases} 1 & I(x,y) > \left|I(x,y) - \mu_{xy}\right| > 2\sigma_{xy} \\ 0 & else \end{cases}$$

$$g(x,y) = \begin{cases} 1 & I(x,y) > \mu_{xy} + 2\sigma_{xy} \; and \; I(x,y) > \tau_{min} \\ 0 & else \end{cases}$$

Variable thresholding based on local image properties

- **Algorithm:**

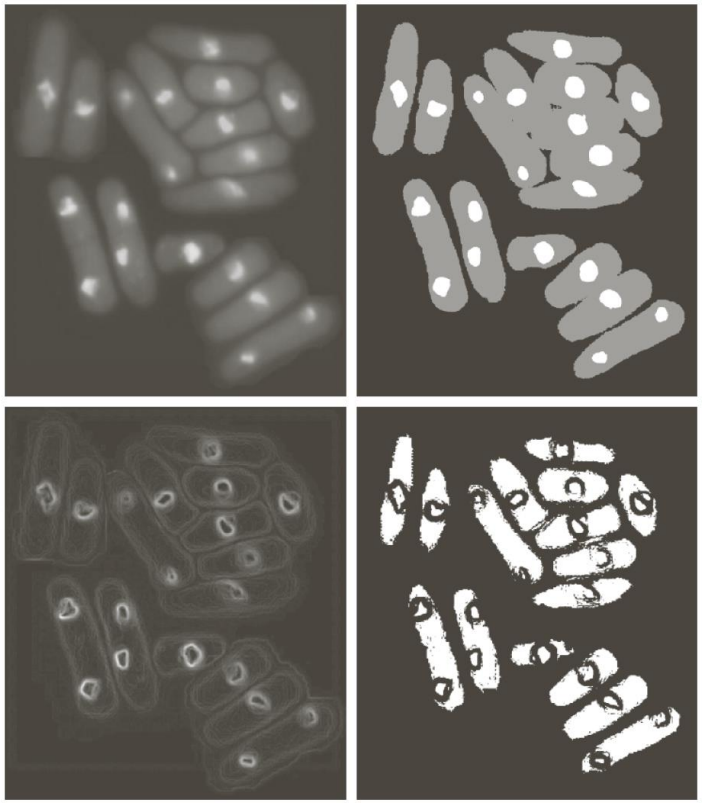
$$T_{xy} = a\sigma_{xy} + bm_{xy}$$

or

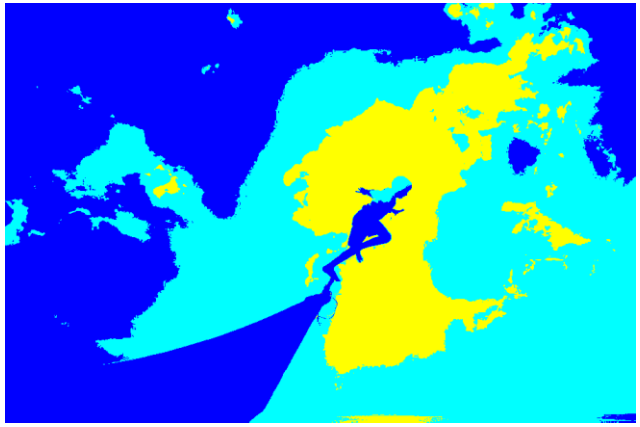
$$T_{xy} = a\sigma_{xy} + bm_G$$

- **Matlab function:**

```
g = stdfilt(f, nhood);
```



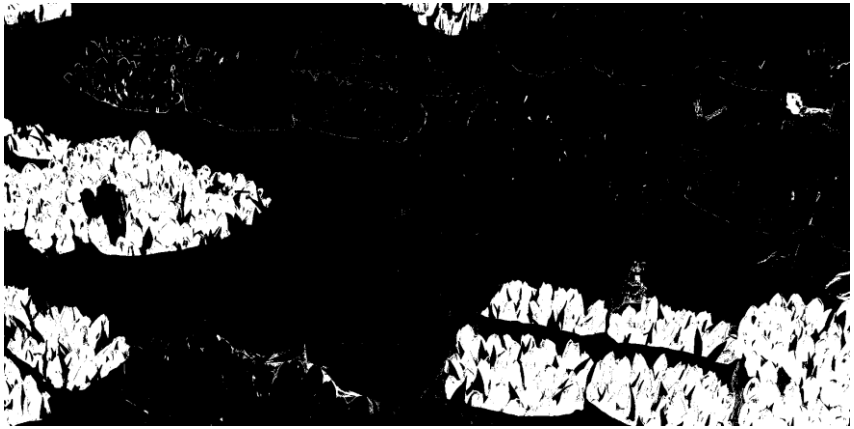
Different thresholding methods



Thresholding for color images

- Thresholding independently on RGB channels
- Combine channels

$$\|I(x,y) - c\| < \tau$$



Region-based Segmentation: outline

- Region growing
- Region spilt and merge
- Clustering and Super-pixel

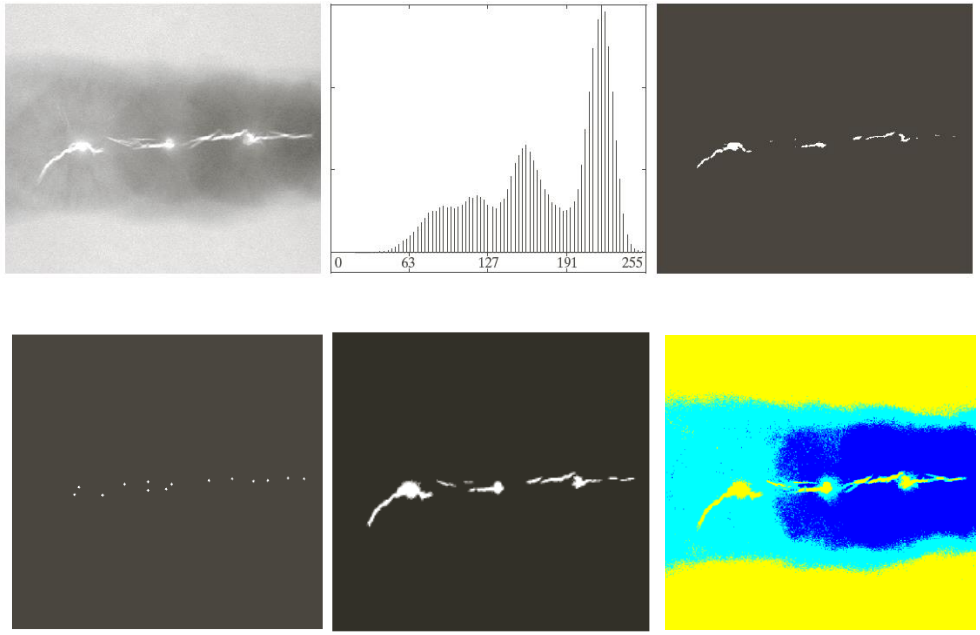
Basic region growing

- If we only want "common" pixels near one point.

  1) from input image $I(x, y)$ get a binary "seed image" $S(x, y)$ for locations of interest. (e.g. by thresholding).

  2) reduce seed connected components down to single point each.

  3) let $T(x, y) = 1$ if $I(x, y)$ satisfies some predicate/condition and 0 else. (e.g. (x, y) is 8-connected to seed point (x_i, y_i) and $|I(x, y) - I(x_i, y_i)| < T$).



Basic region growing

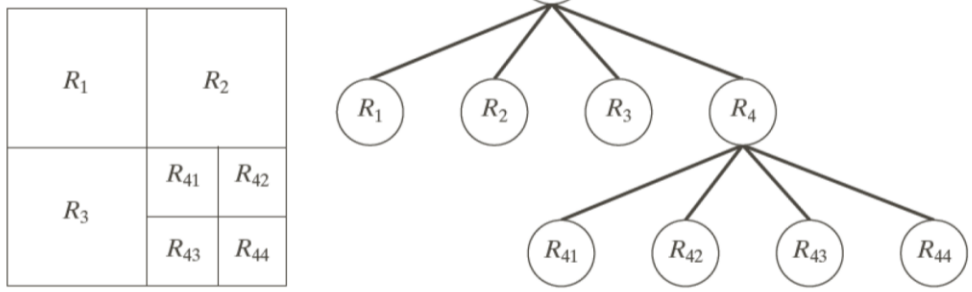
❑ Matlab code

```
function regiongrow(im,t)
while 1
 …
   g = round(ginput(1));
   if nargin<2
      bw = grayconnected(im,g(2),g(1));
   else
      bw = grayconnected(im,g(2),g(1),t);
   end
 …
end
```

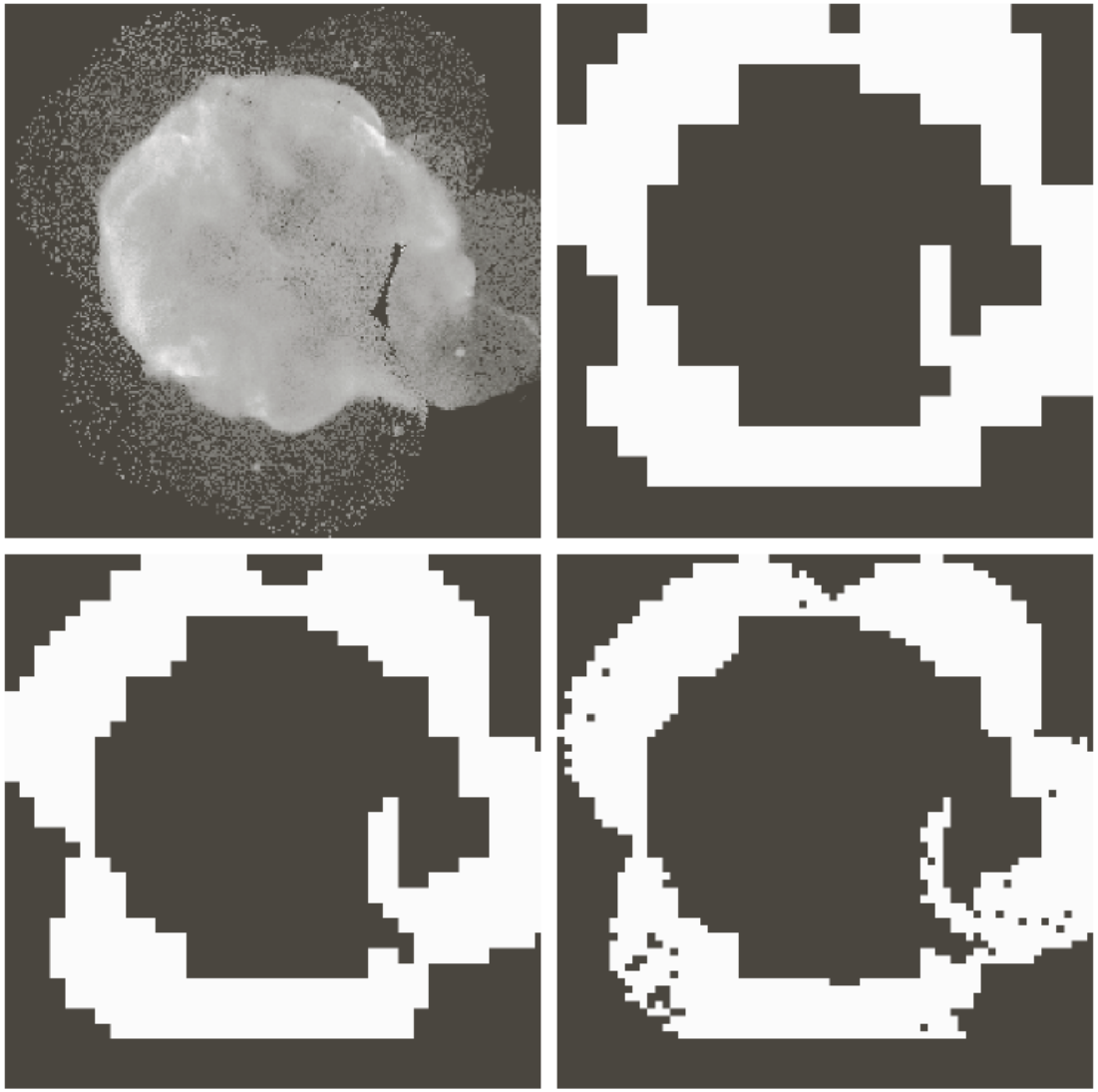
Region Splitting and Merging

➢ Steps

1. Split into four disjoint quadrants any region R_i for which $Q(R_i) = False$ (need to specify a minimum quadregion size beyond which no further splitting is carried out;
2. Merge any adjacent regions R_j and R_k for which $Q(R_j \cup R_k) = True$;
3. Stop when no further merging is possible.



Region Splitting and Merging



K-means Clustering

- **Algorithm:**

1) Specify an initial set of clusters centers $m_1, m_2, \cdots, m_k \in \mathcal{R}^n$.

2) For each $x_i \in \mathcal{R}^n$ in dataset, assign it to closest cluster

$$x_i \in cluster_j \text{ if } \|x_i - m_j\| < \|x_i - m_k\| \quad (k \neq j)$$

3) Update the mean $m_j =$ average value of all x in cluster j.

$$m_j = \frac{1}{c_j} \cdot \sum_{x \in c_j} x \quad j = 1,2, \cdots, k$$

4) Keeps altering 2) and 3) until stop changing.

e.g., Image histogram

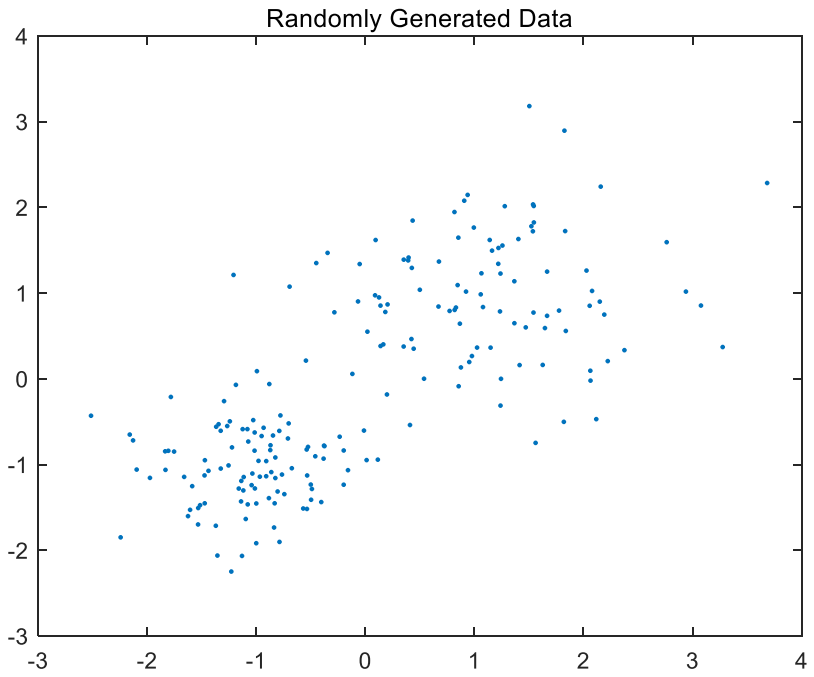
e.g., Color space k-means cluster.

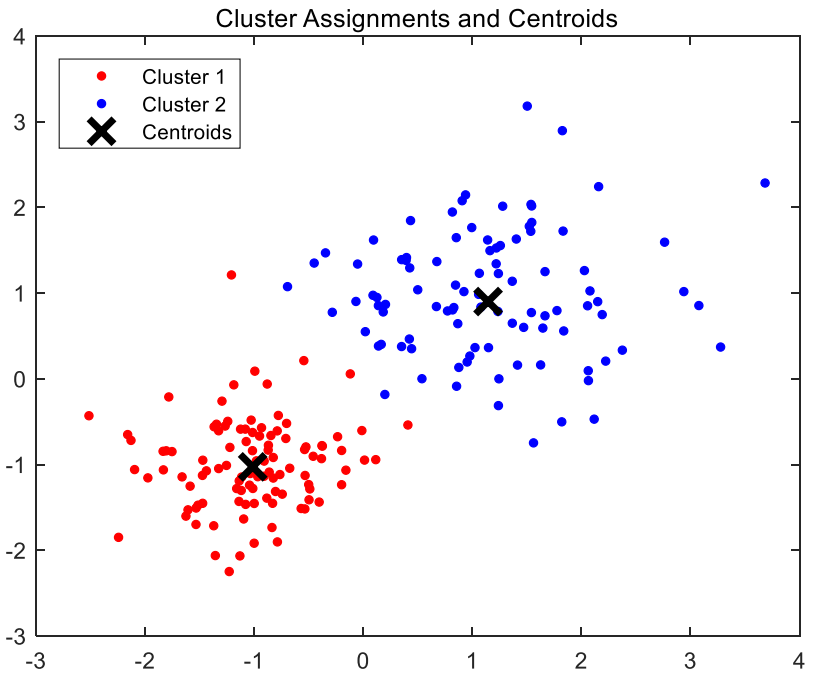
Loss function:

$$J = \sum_{i=1}^{C} \sum_{i=1}^{N} r_{ij} \cdot \nu(x_j, \mu_i)$$

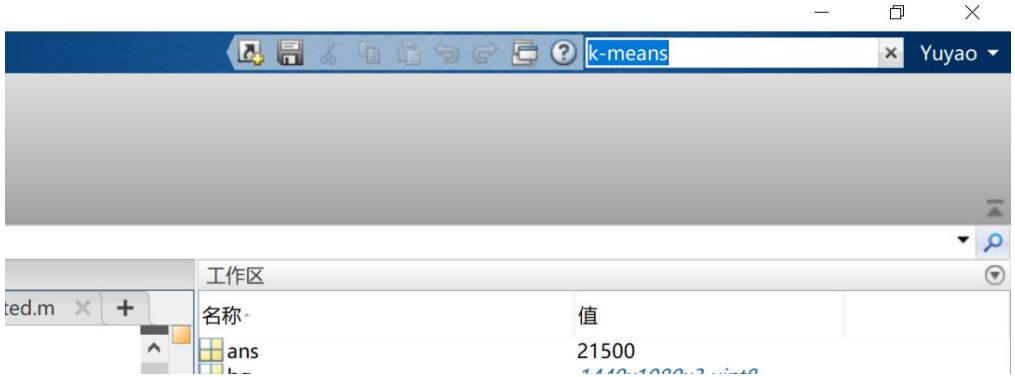
$$\nu(x_j, \mu_i) = ||x_j - \mu_i||^2, \quad r_{nk} = \begin{cases} 1 & if\ x_n \in k \\ 0 & else \end{cases}$$

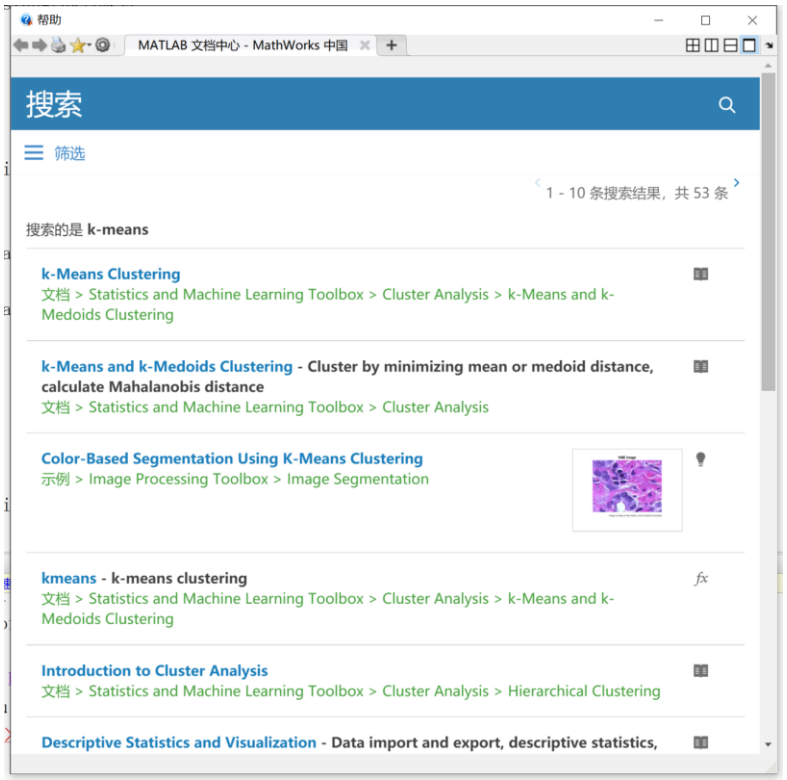
K-means Clustering





K-means Clustering





K-means Clustering

H&E image

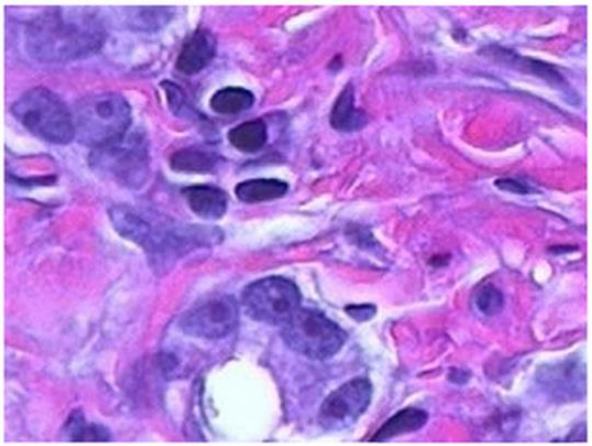
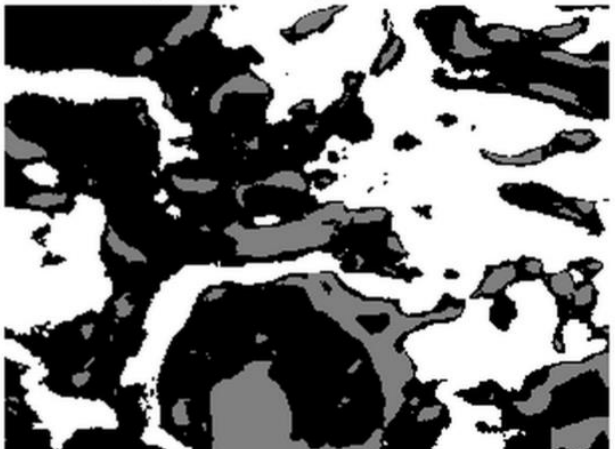
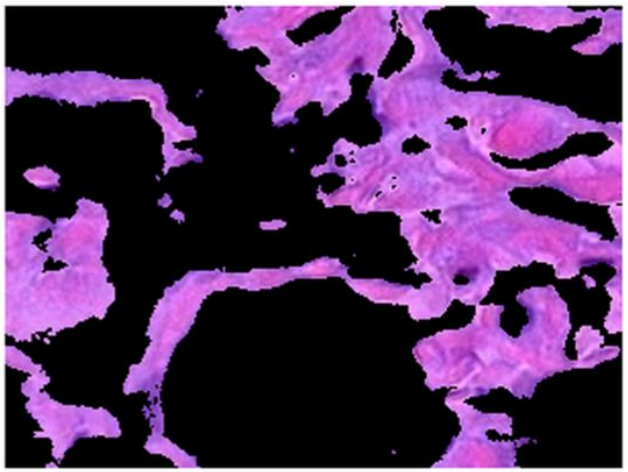


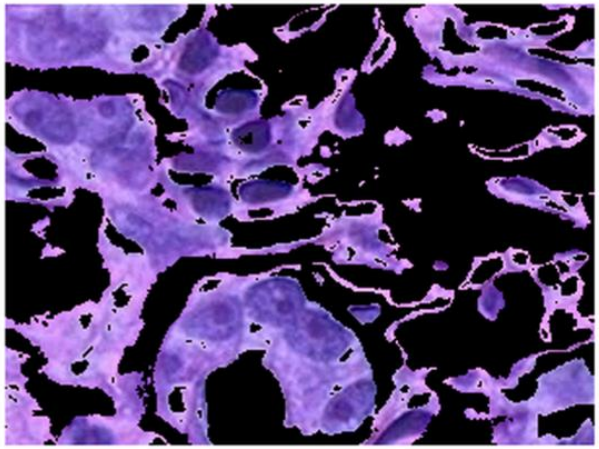
image labeled by cluster index



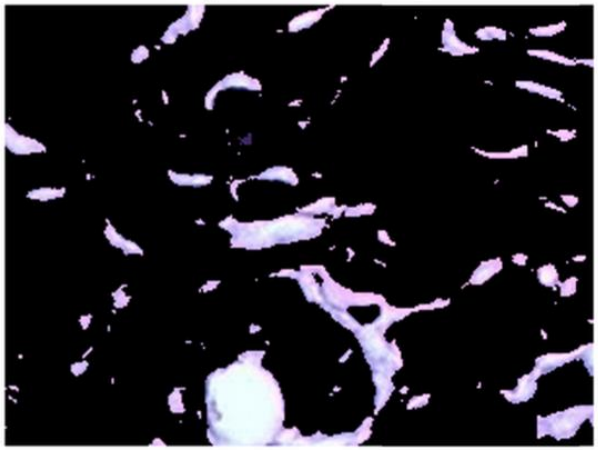
objects in cluster 3



objects in cluster 1



objects in cluster 2



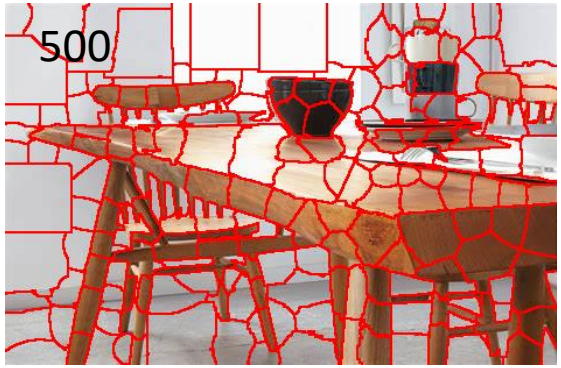
Super-pixel

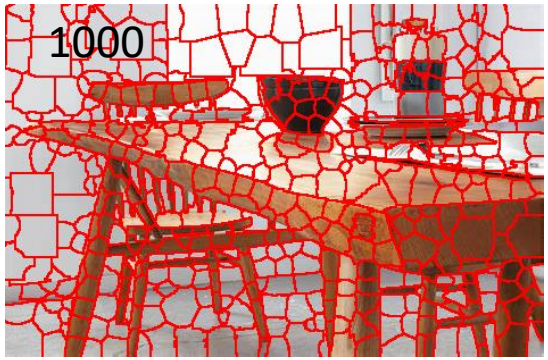
- Modification of K-means used in image processing;
- Regions of image that are contiguous and have similar intensity/color.
- Why doing this?
  - More compact (e.g. thousands of super-pixels could represent millions of pixels).
  - “Keeps things together” better for subsequent segmentation; computationally efficient.

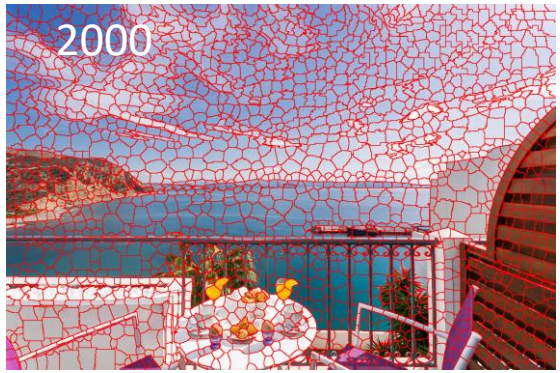
Super-pixel

- Idea: Clustering 5-D vectors $[r,g,b,x,y]$
  - 1) initialize super-pixels center by sampling N locations on a region grid in image plane.
    - Move slightly within 3x3 neighborhood to lie on lowest gradient position (Don't want to start on an edge).
  - 2) For each cluster center m_i, compute distance bwt m_i and each pixel in a neighborhood of m_i.
    - Only search in neighborhood, the region size depends on # of N.
    - Assign pixel to cluster i if its distance is better than its current value.

Super-pixel







Super-pixel

Take home message

- **Pixel-based segmentation:**

  each pixel is segmented based on gray-level values, no contextual information, only histogram.

  -Example: Hough transform

- **Edge-based segmentation:**

  Detects and links edge pixels to form contours.

- **Region-based segmentation:**

  considers gray-levels from neighboring pixels by

  – including similar neighboring pixels (region growing),

  – split-and-merge,

  – super-pixel segmentation (k-means).

Take home message

- Region based methods are robust because:
  - Regions cover more pixels than edges and thus you have more information available in order to characterize your region
  - When detecting a region you could for instance use texture which is not easy when dealing with edges
  - Region growing techniques are generally better in noisy images where edges are difficult to detect
- The edge based method can be preferable because:
  - Algorithms are usually less complex
  - Edges are important features in an image to separate regions
- The edge of a region can often be hard to find because of noise or occlusions
- Combination of results may often be a good idea