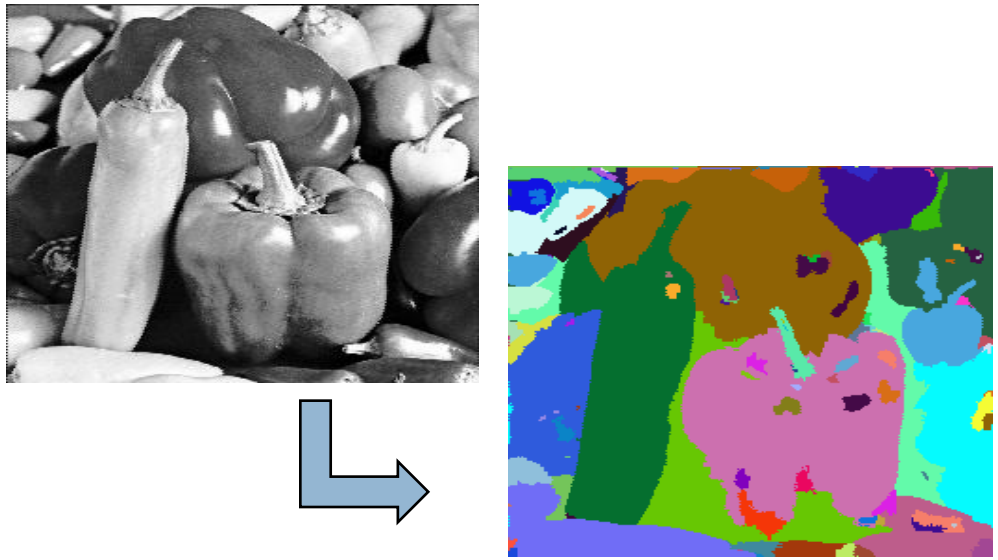
DIGITAL IMAGE PROCESSING

Image Segmentation:
Thresholding

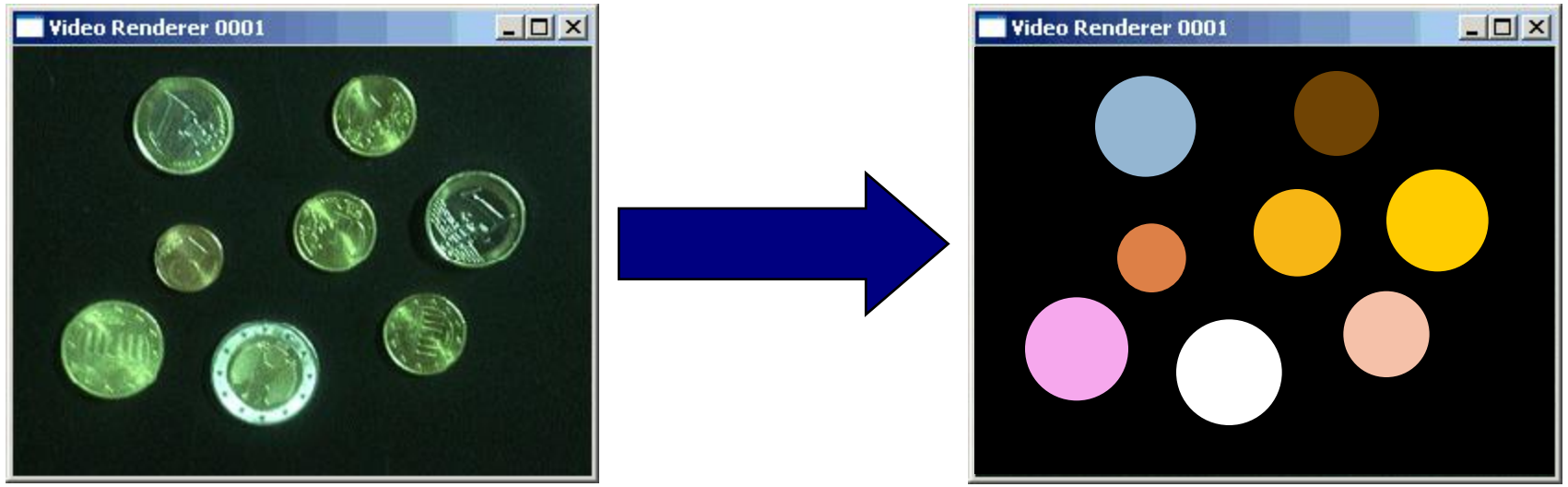
What is segmentation?

- Dividing the image into different regions.
  - Separating objects from background and giving them individual labels(ID numbers)
- The purpose of image segmentation is to partition an image into meaningful regions with respect to a particular application

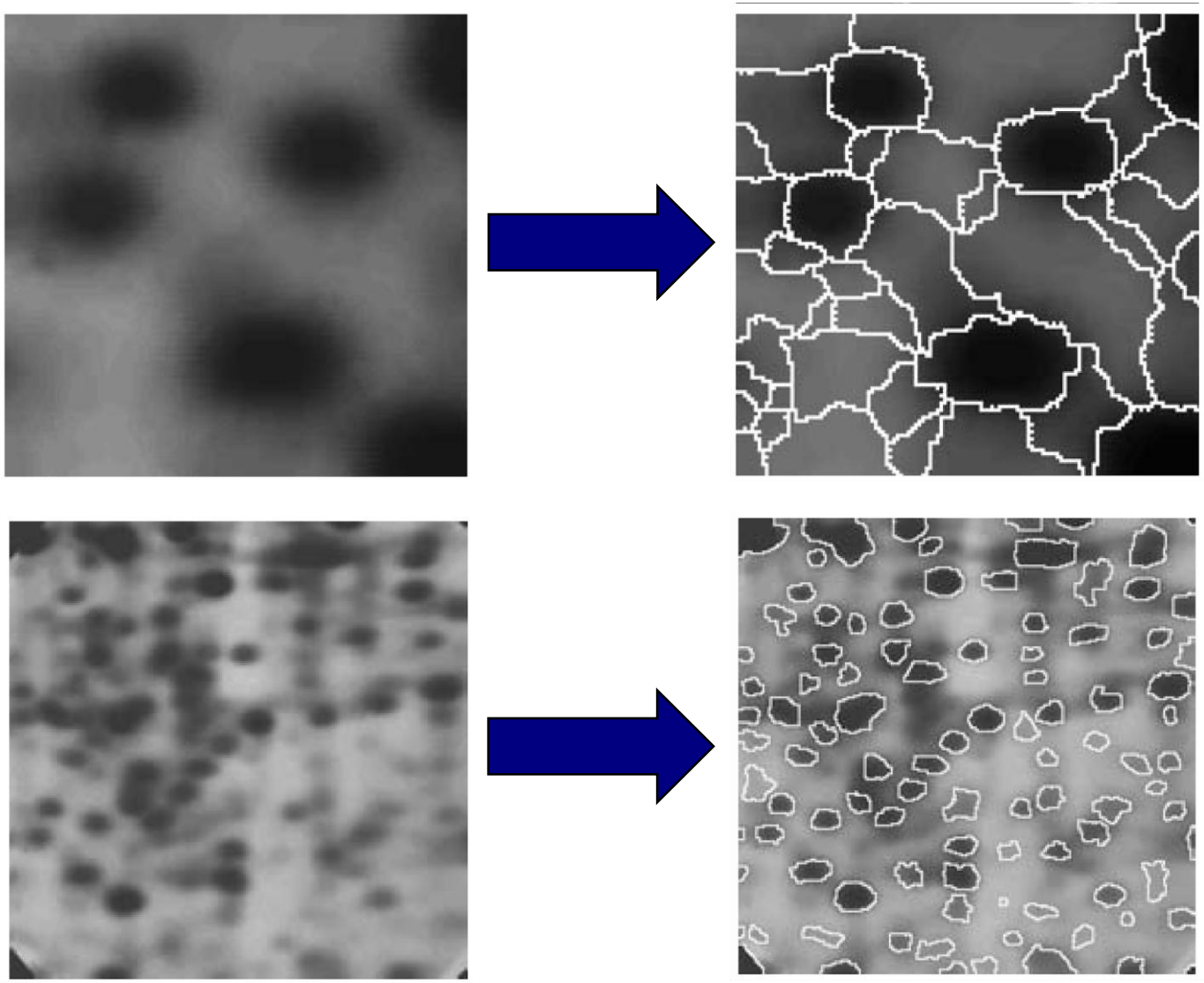


The Segmentation Problem

- Segmentation attempts to partition the pixels of an image into groups that strongly correlate with the objects in an image
- Typically the first step in any automated computer vision application



Segmentation Examples



Images taken from Gonzalez & Woods, Digital Image Processing (2002)

Wikipedia on segmentation

”In computer vision, Segmentation is the process of partitioning a digital image into multiple segments”

”More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics.”

”Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s).”

Why segmentation?

- Usually image segmentation is an initial and vital step in a series of processes aimed at overall image understanding
  - Segmentation is generally the first stage in any attempt to analyze or interpret an image automatically.

- Segmentation bridges the gap between low-level image processing and high-level image processing.

- Some kinds of segmentation technique will be found in any application involving the detection, recognition, and measurement of objects in images.

Why segmentation?

- The role of segmentation is crucial in most tasks requiring image analysis.
  - The success or failure of the task is often a direct consequence of the success or failure of segmentation.

- Accurate segmentation of objects of interest in an image greatly facilitates further analysis of these objects. For example, it allows us to:
  - Count the number of objects of a certain type.
  - Measure geometric properties (e.g., area, perimeter) of objects in the image.
  - Study properties of an individual object (intensity, texture, etc.)

Segmentation – difficulty

- Segmentation is often the most difficult problem to solve in image analysis.
  - There is no universal solution!

- A reliable and accurate segmentation of an image is, in general, very difficult to achieve by purely automatic means

”Since there is no general solution to the image segmentation problem, these [general purpose] techniques often have to be combined with domain knowledge in order to effectively solve an image segmentation problem for a problem domain.” - wikipedia

Targeted Segmentation

- Segmentation is an ill-posed problem...

What is a correct segmentation of this image?

Targeted Segmentation

- ...unless we specify a segmentation target.

"Segment the orange car from the background"

"Segment all road signs from the background"

Dilemma

input

result 1

result 2

What do we mean by “DIFFERENT” objects?

Another example: when we look at trees at a close distance, we consider each of them as a different object; but as we look at trees far away, they merge into one coherent object (woods)

Targeted Segmentation

- A segmentation can also be defined as a mapping from the set of pixels to some application dependent target set, e.g.
  - {Object, Background}
  - {Humans, Other objects}
  - {1,2,3,4,...}
  - {Healthy tissue, Tumors}

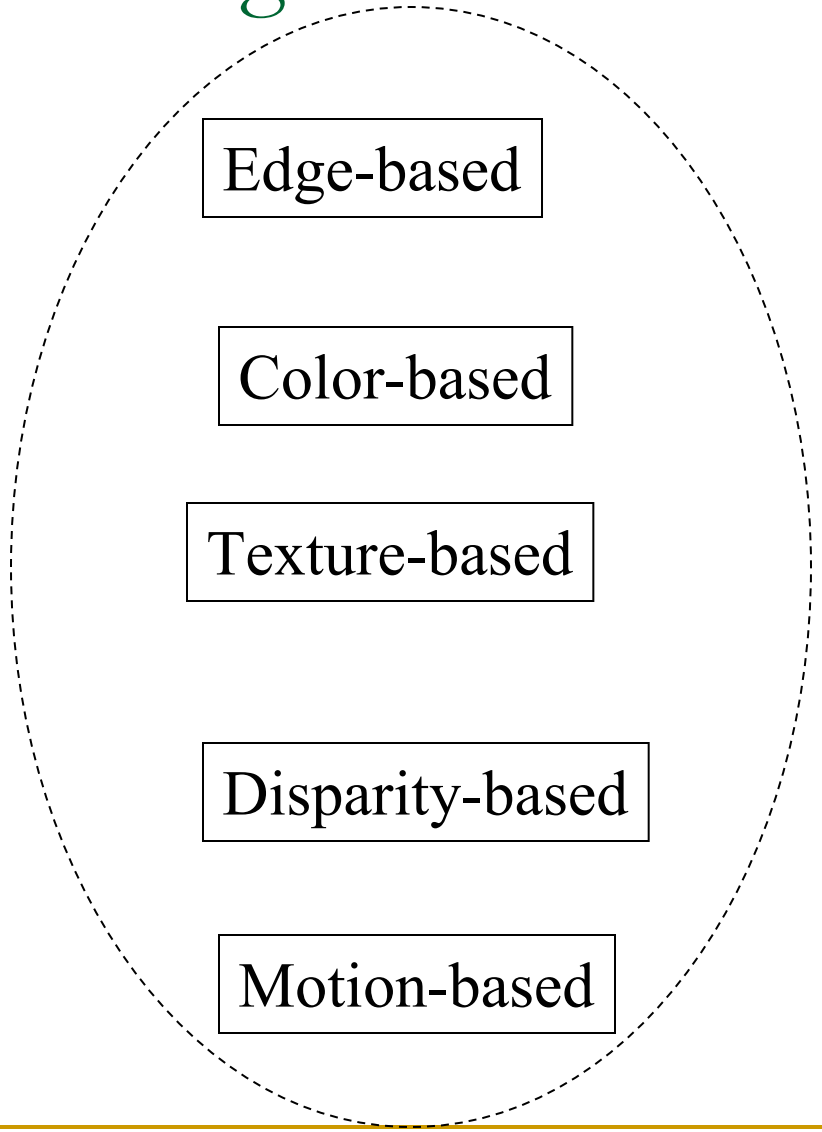
- To perform accurate segmentation, we (or our algorithms) need to somehow know how to differentiate between different elements of the target set.

Segmentation Algorithms

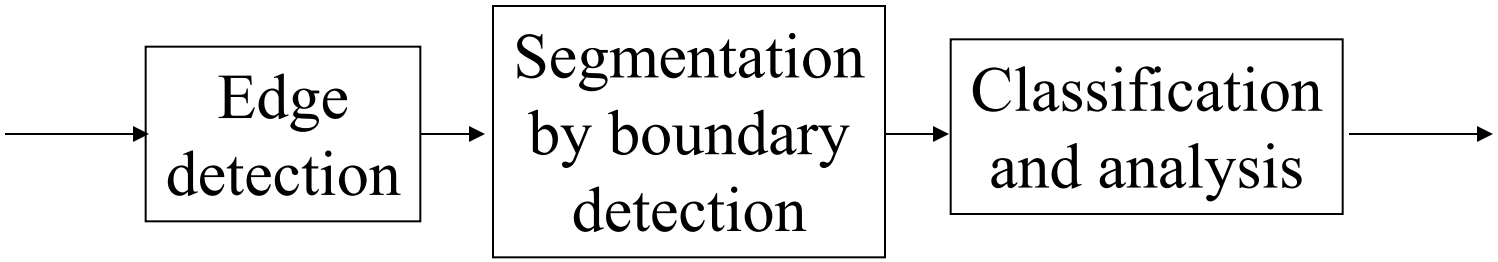
- Segmentation algorithms are based on one of two basic properties of color, gray values, or texture:

- **Similarity**
  - Partition an image into regions that are similar according to a predefined criteria.
- **Discontinuity**
  - Detecting boundaries of regions based on local discontinuity in intensity.

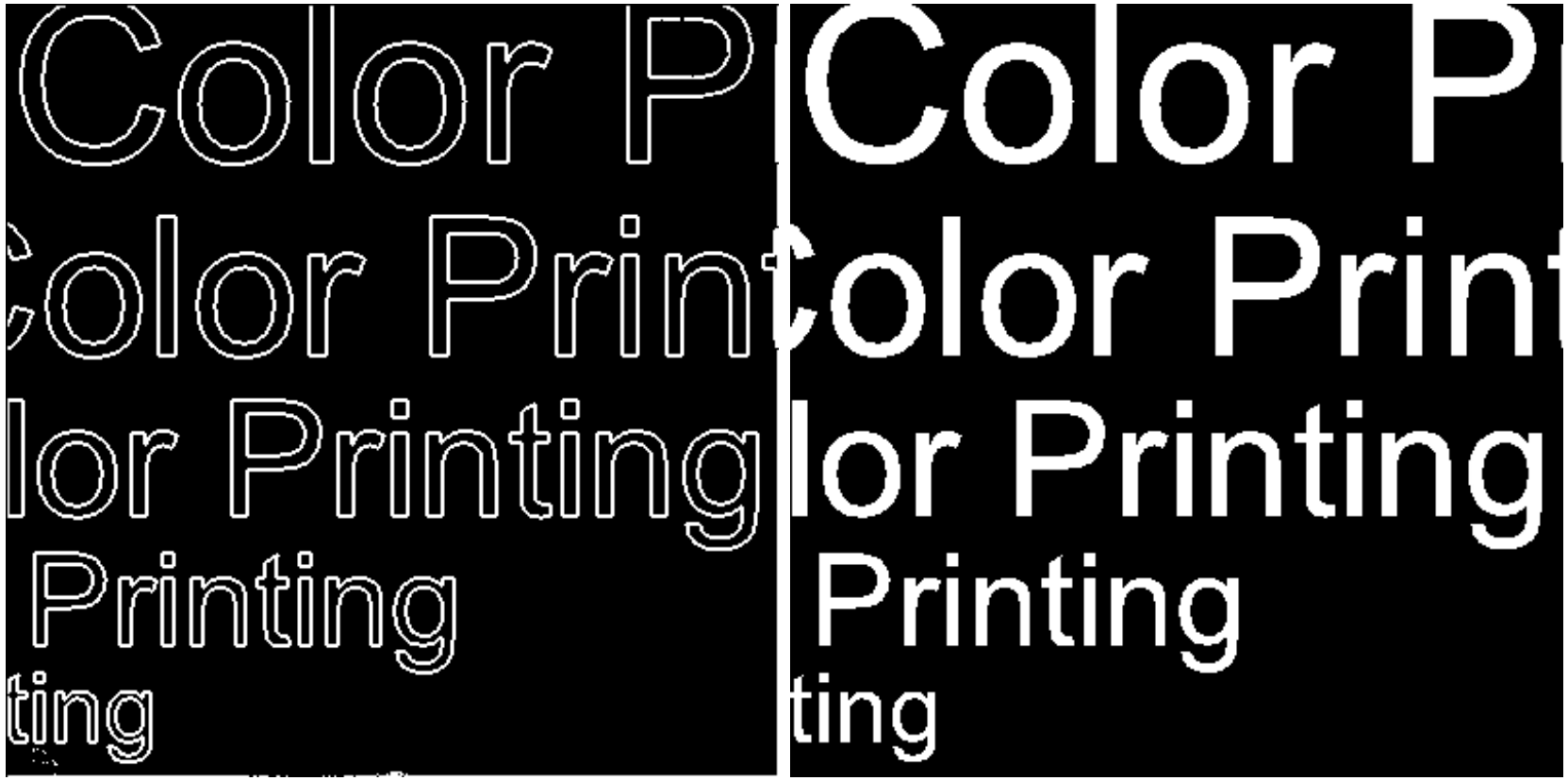
Overview of Segmentation Techniques



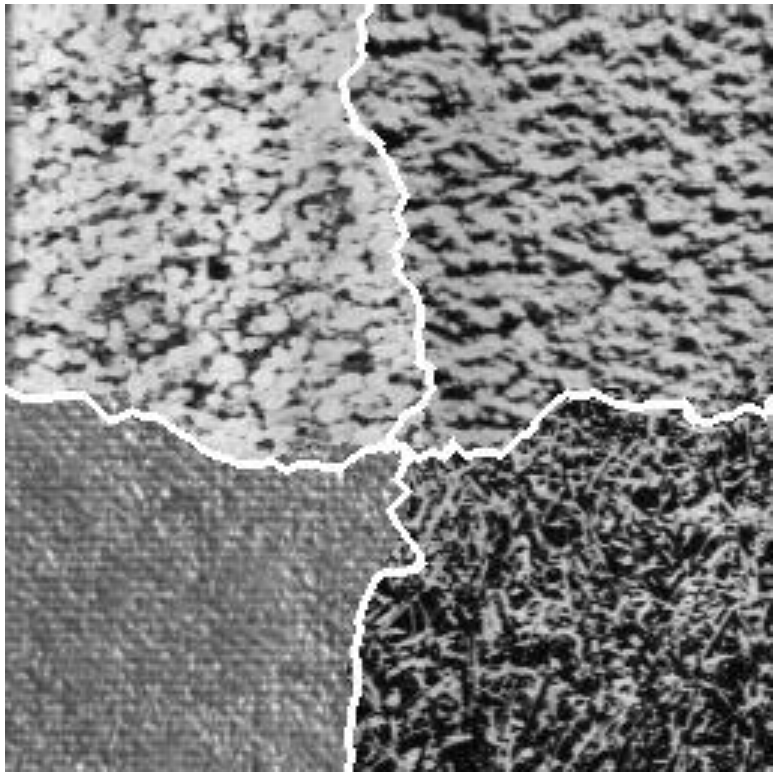
Edge-based Techniques



Region-Filling



Texture-based Techniques



What is Texture?

No one exactly knows.

In the visual arts, **texture** is the perceived surface quality of an artwork.

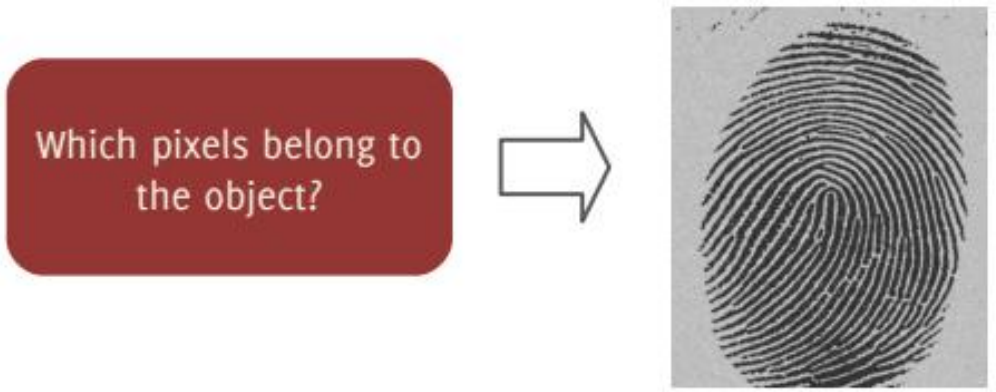
Segmentation Algorithms

- We will study Four types of algorithms
  - **Thresholding** → Similarity
    - Based on pixel intensities (shape of histogram is often used for automation).
  - **Edge-based** → Discontinuity
    - Detecting edges that separate regions from each other.
  - **Region-based** → Similarity
    - Grouping similar pixels (with e.g. region growing, split & merge).
  - **Watershed segmentation** → Discontinuity
    - Find regions corresponding to local minima in intensity.

Thresholding

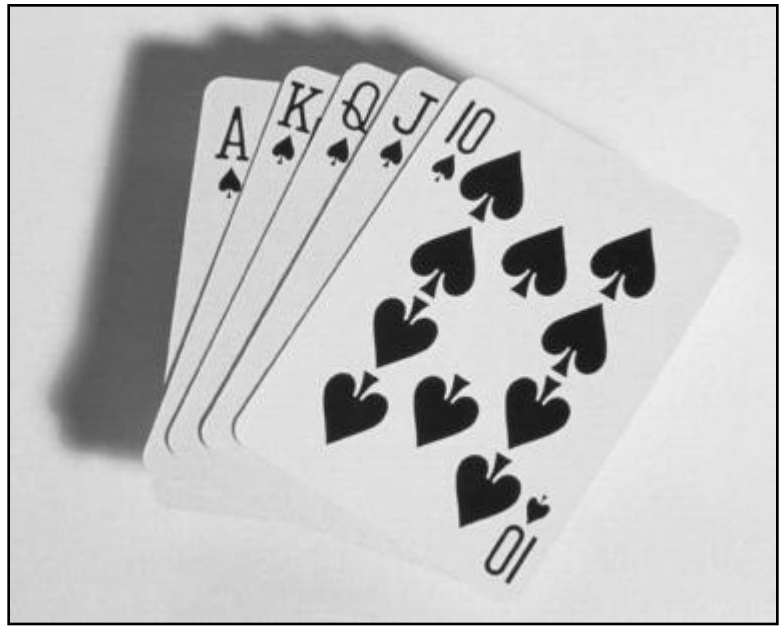
- **Simplest**, **widely used** for image segmentation.
- It is useful in discriminating foreground from the background.
- By selecting an adequate threshold T, the **grayscale** image can be converted to **binary image**.
- mathematically :

$$g(x,y) = \begin{cases} 1 \; if \;\; f(x,y) > T \\ 0 \; if \;\; f(x,y) \leq T \end{cases}$$

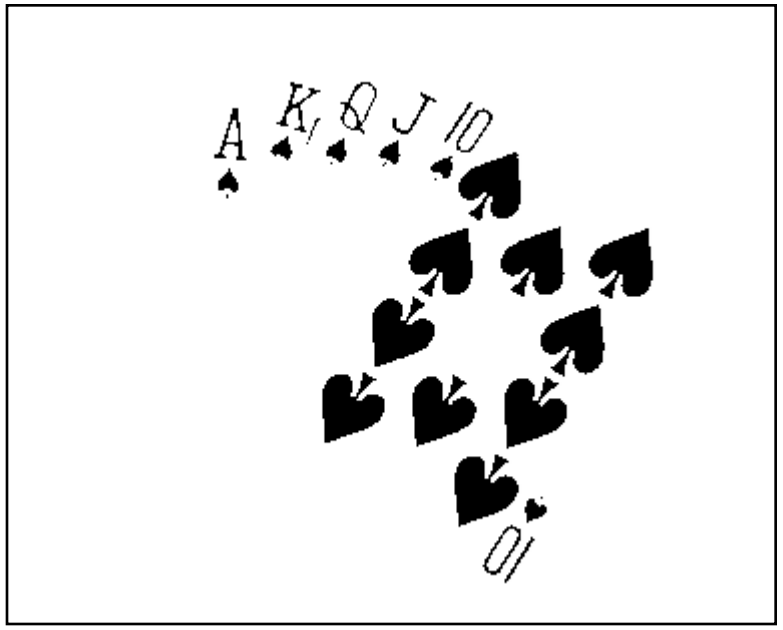


Thresholding Example

- Imagine a poker playing robot that needs to visually interpret the cards in its hand



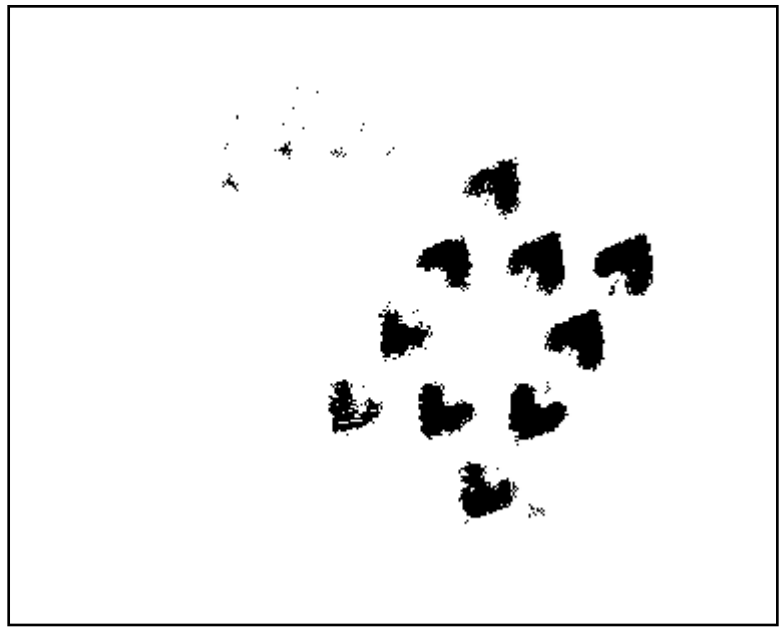
Original Image



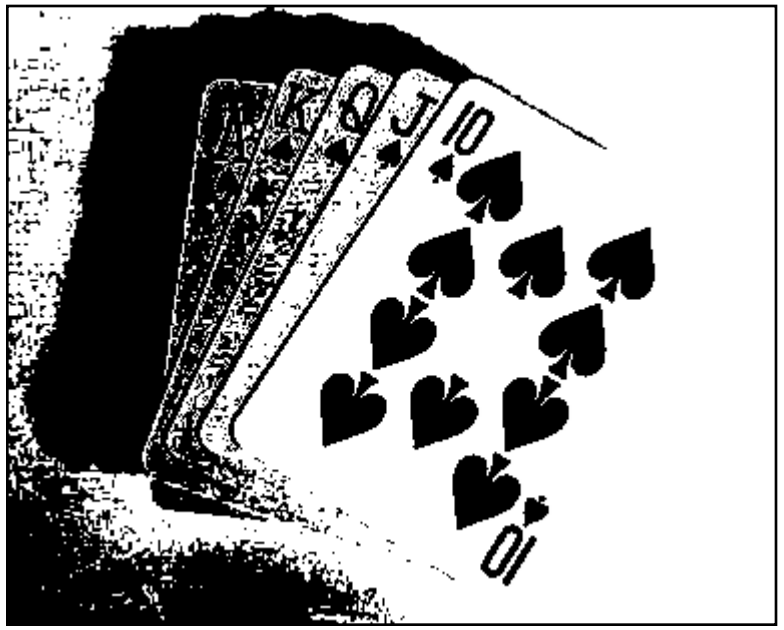
Thresholded Image

But Be Careful

- If you get the threshold wrong the results can be disastrous



Threshold Too Low



Threshold Too High

Thresholding - methods

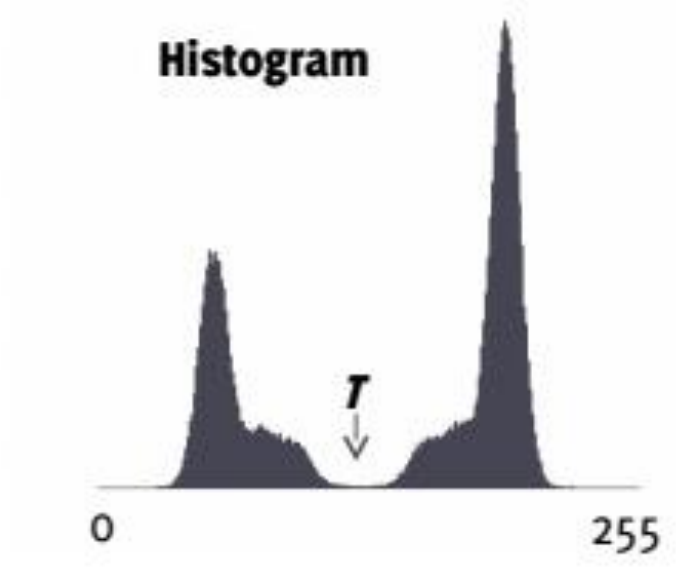
- **Global threshold**
  The same value is used for the whole image.

- **Optimal global threshold**
  Based on the shape of the current image histogram. Searching for valleys, Gaussian distribution etc.

- **Local (or dynamic) threshold**
  The image is divided into non-overlapping sections, which are thresholded one by one.

Global Thresholding

- Partition the image histogram using a single global threshold
- Success strongly depends on how well the histogram can be partitioned



- We chose a **threshold T** midway between the two gray value distributions.

How to find a global threshold

- The basic global threshold, T, is calculated
- as follows:

1. Select an initial estimate for T (typically the average grey level in the image)
2. Segment the image using T to produce two groups of pixels:

   G_1 : pixels with grey levels >T

   G_2 : pixels with grey levels ≤ T
3. Compute the average grey levels of pixels in G_1 to give μ_1 and G_2 to give μ_2

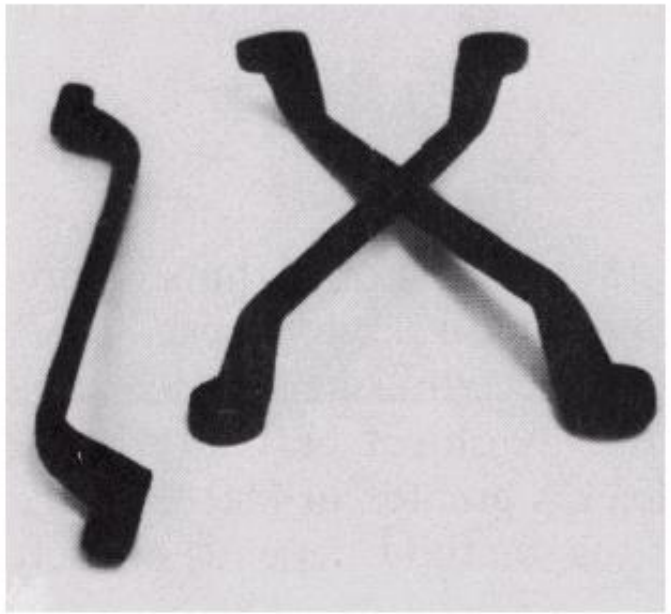
How to find a global threshold

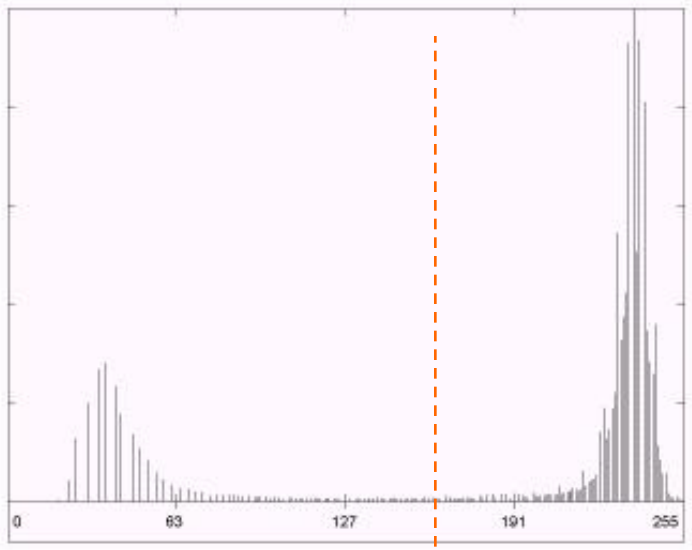
4. Compute a new threshold value:

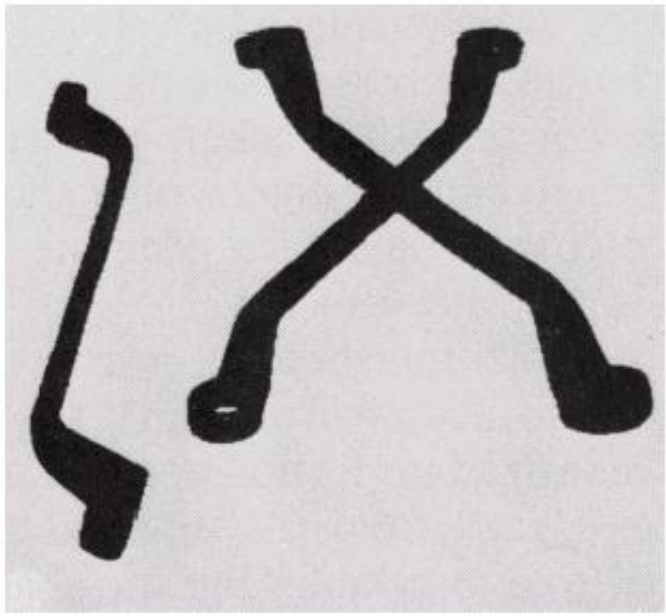
$$T = \frac{\mu_1 + \mu_2}{2}$$

5. Repeat steps 2 – 4 until the difference in T in successive iterations is less than a predefined limit T_∞

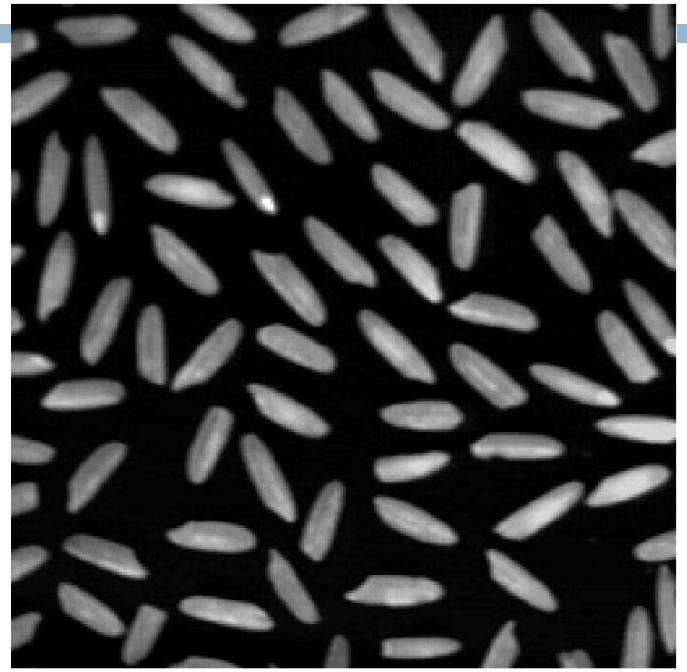
Global threshold - Example





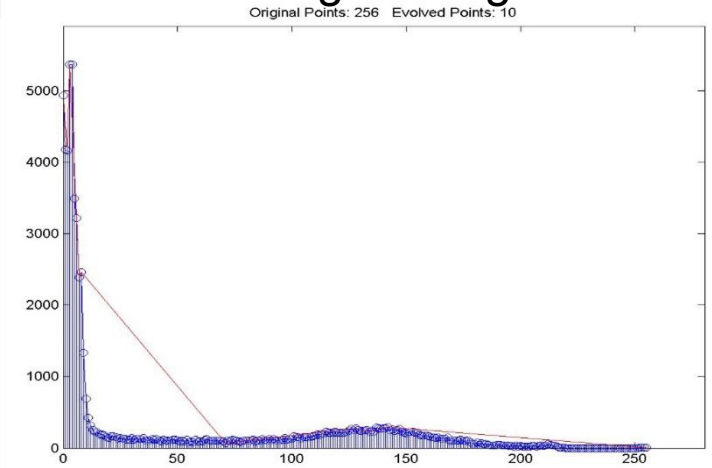


Global threshold - Example



Grayscale rice image

Image histogram



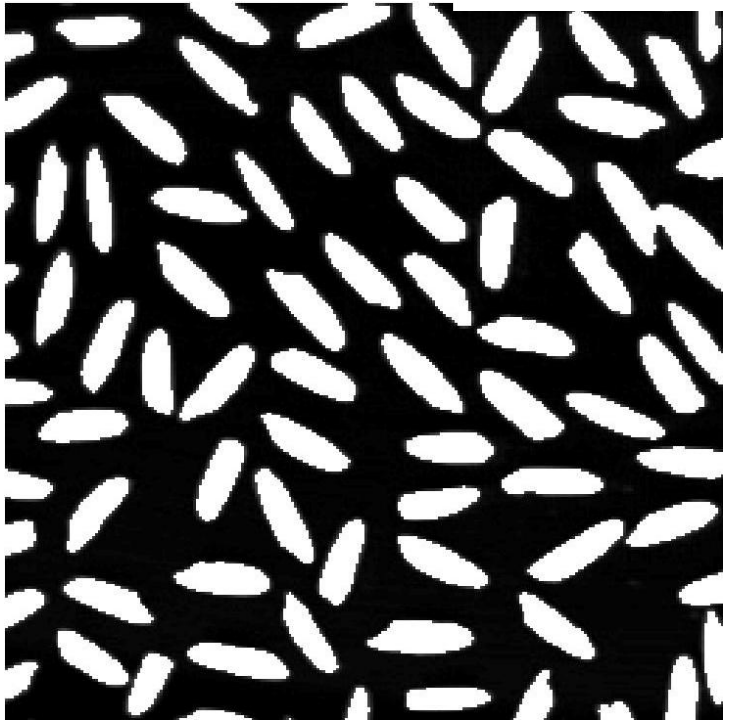
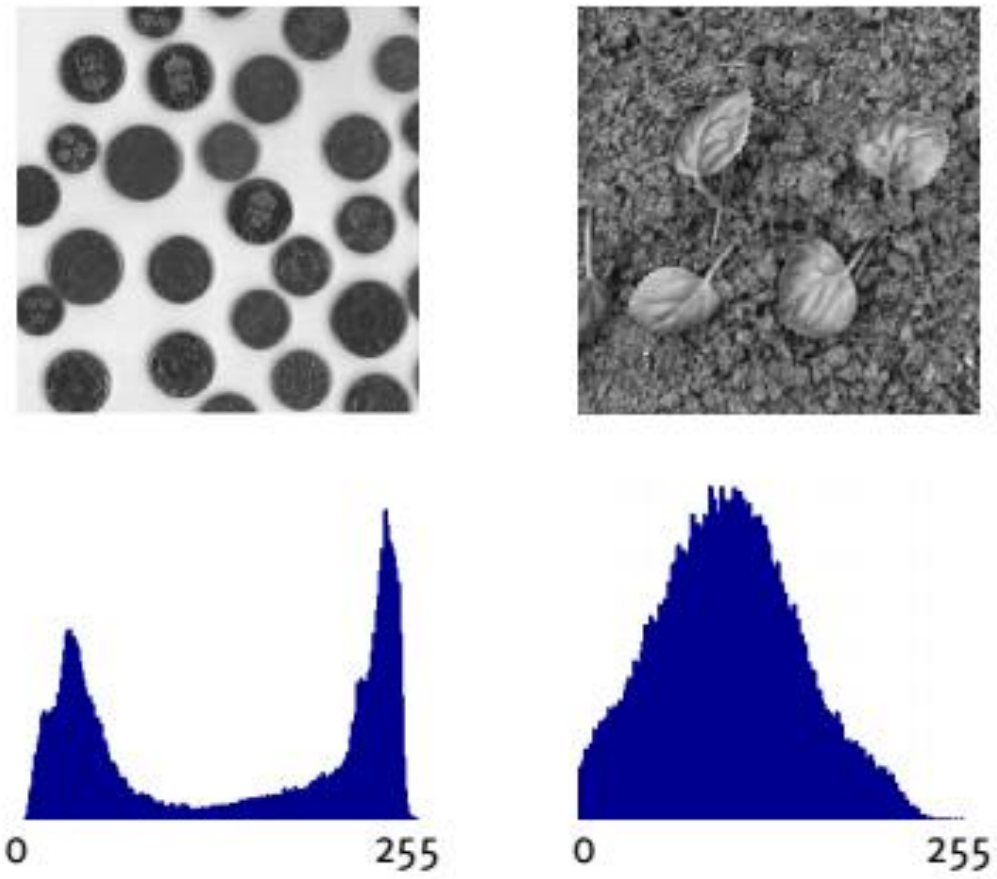


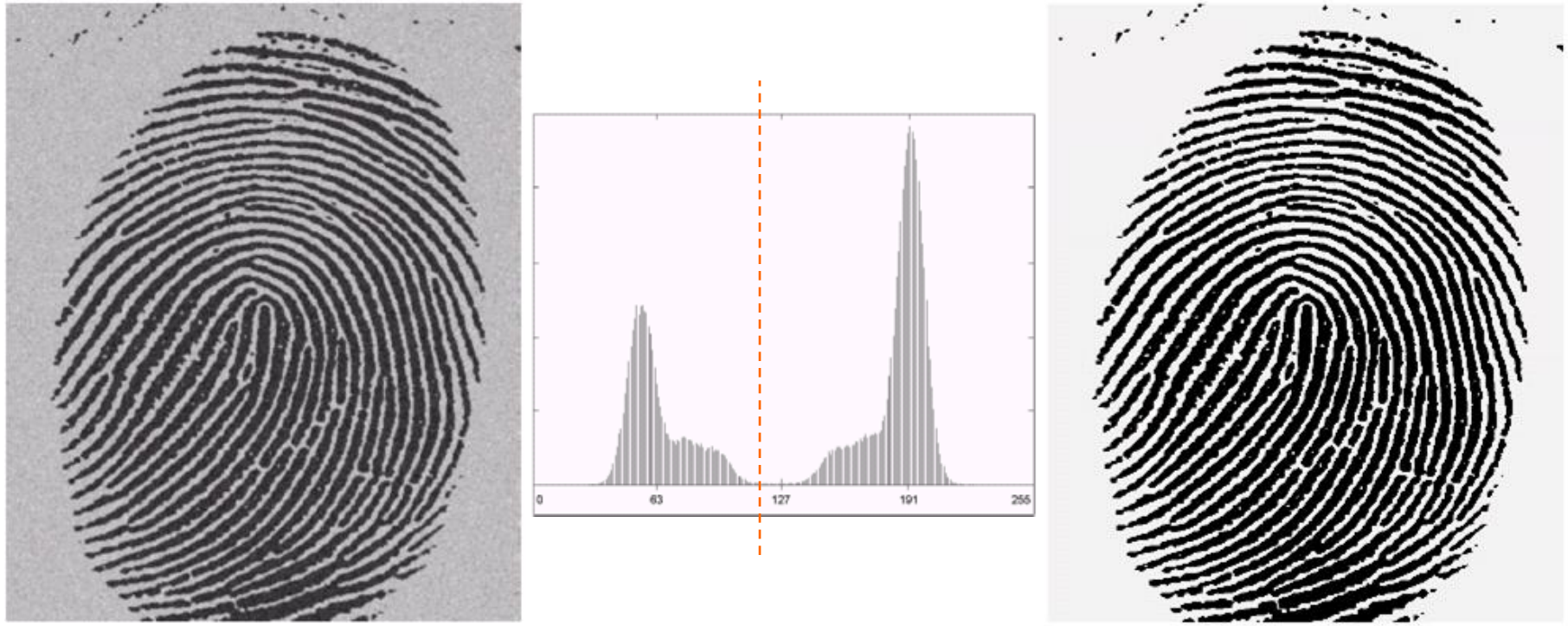
Image after segmentation

Global threshold

- This algorithm works very well for finding thresholds when the histogram is suitable (bi-modal)



Thresholding Example 2



Optimum Global threshold

- **Otsu's Method**
- Based on a very simple idea: Find the threshold that *minimizes the weighted within-class variance.*
- This turns out to be the same as *maximizing the between-class variance.*
- Operates directly on the gray level histogram [*e.g.* 256 numbers, P(i)], so it's fast (once the histogram is computed).

Otsu's Method

- {0,1,2,…,*L-1*} , *L* means gray level intensity

$$MN = n_0 + n_1 + n_2 + \ldots + n_{L-1}$$

$M*N$ = total number of pixel.

n_i = number of pixels with intensity i

$p_i = n_i / MN$, *probability of* intensity i

- Select a threshold $T(k) = k, 0 < k < L-1$, and use it to classify C_1: intensity in the range [0, *k*] and C_2: [k+1, *L-1*]

Class Probability $P_1(k) = \sum_{i=0}^{k} p_i$, $P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1(k)$

$$P_1 + P_2 = 1$$

Class mean $m_1(k) = \sum_{i=0}^{k} i \frac{p_i}{P_1} = \frac{1}{P_1} \sum_{i=0}^{k} i p_i$, $m_2(k) = \frac{1}{P_2} \sum_{i=k+1}^{L-1} i p_i$

Otsu's Method

Class variance $\sigma_1^2(k) = \frac{1}{P_1}\sum_{i=1}^{k}[i-m_1]^2 p_i$, $\sigma_2^2(k) = \frac{1}{P_2}\sum_{i=k+1}^{L-1}[i-m_2]^2 p_i$

Minimize: within Class variance $\sigma_w^2(k) = P_1\sigma_1^2 + P_2\sigma_2^2$

Between Class *variance* $\sigma_B^2(k) = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2$

Global mean $m_G = \sum_{i=0}^{L-1} ip_i$ $P_1 m_1 + P_2 m_2 = m_G$

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

Otsu's Method

since $\sigma_G^2 = \sigma_W^2 + \sigma_B^2$, minimizing σ_W^2 means maximizing σ_B^2

$$\begin{aligned}
\sigma_B^2 &= P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2 \\
&= P_1 m_1^2 + P_2 m_2^2 - 2m_G(P_1 m_1 + P_2 m_2) + m_G^2(P_1 + P_2) \\
&= P_1 m_1^2 + P_2 m_2^2 - (P_1 m_1 + P_2 m_2)^2 \\
&= P_1 m_1^2 (1 - P_1) + P_2 m_2^2 (1 - P_2) - 2P_1 P_2 m_1 m_2 \\
&= P_1 P_2 (m_1^2 + m_2^2 - 2m_1 m_2) \\
&= P_1 P_2 (m_1 - m_2)^2
\end{aligned}$$

Otsu's Method

$$\sigma_B^2(k) = P_1 P_2 (m_1 - m_2)^2 = \frac{(m_G P_1(k) - m(k))^2}{P_1(k)(1 - P_1(k))}$$

it is *between-class variance*

$$\eta = \frac{\sigma_B^2(k)}{\sigma_G^2} \qquad 0 \le \eta(k^*) \le 1$$

it is a measure of *separability between class*.

$$\sigma_B^2(k^*) = \max_{o \le k \le L-1} \sigma_B^2(k)$$

$$g(x,y) = \begin{cases} 1 & \text{if } f(x,y) \ge k^* \\ 0 & \text{if } f(x,y) \le k^* \end{cases}$$

For x = 0,1,2,…,*M*-1
and y = 0,1,2…,*N*-1.

Otsu's Method

since $\sigma_G^2 = \sigma_W^2 + \sigma_B^2$, minimizing σ_W^2 means maximizing σ_B^2

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

$$= P_1 P_2 \left(m_1 - \frac{m_G - P_1 m_1}{P_2} \right)^2$$

$$= P_1 P_2 \left(\frac{m_1 - m_G}{P_2} \right)^2 = P_1^2 \frac{(m_1 - m_G)^2}{P_1 P_2}$$

$$= \frac{(m - P_1 m_G)^2}{P_1 (1 - P_1)}$$

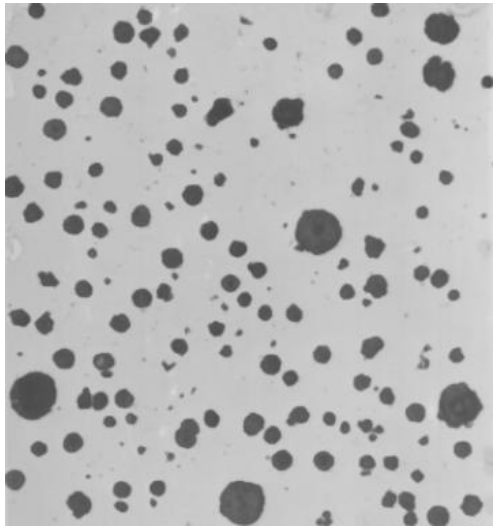
where $m = \sum_{i=0}^{k} i p_i = P_1 m_1$ Cumulative mean upto *k*

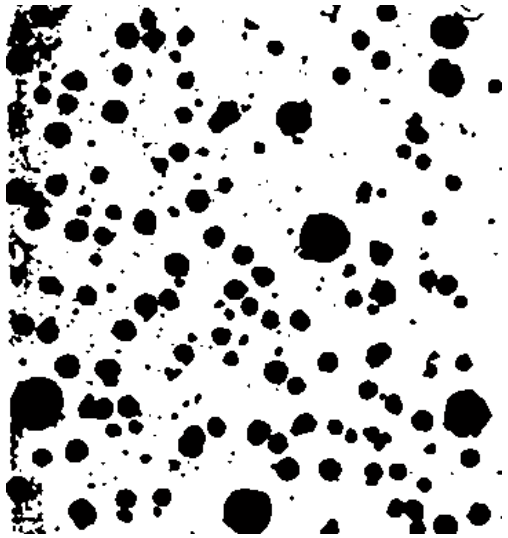
Otsu's Method

- The criterion function involves *between-classes* variance to the total variance is defined as:

$$\eta = \sigma_B^2 / \sigma_G^2$$

- All possible thresholds are evaluated in this way, and the one that maximizes **η** is chosen as the optimal threshold



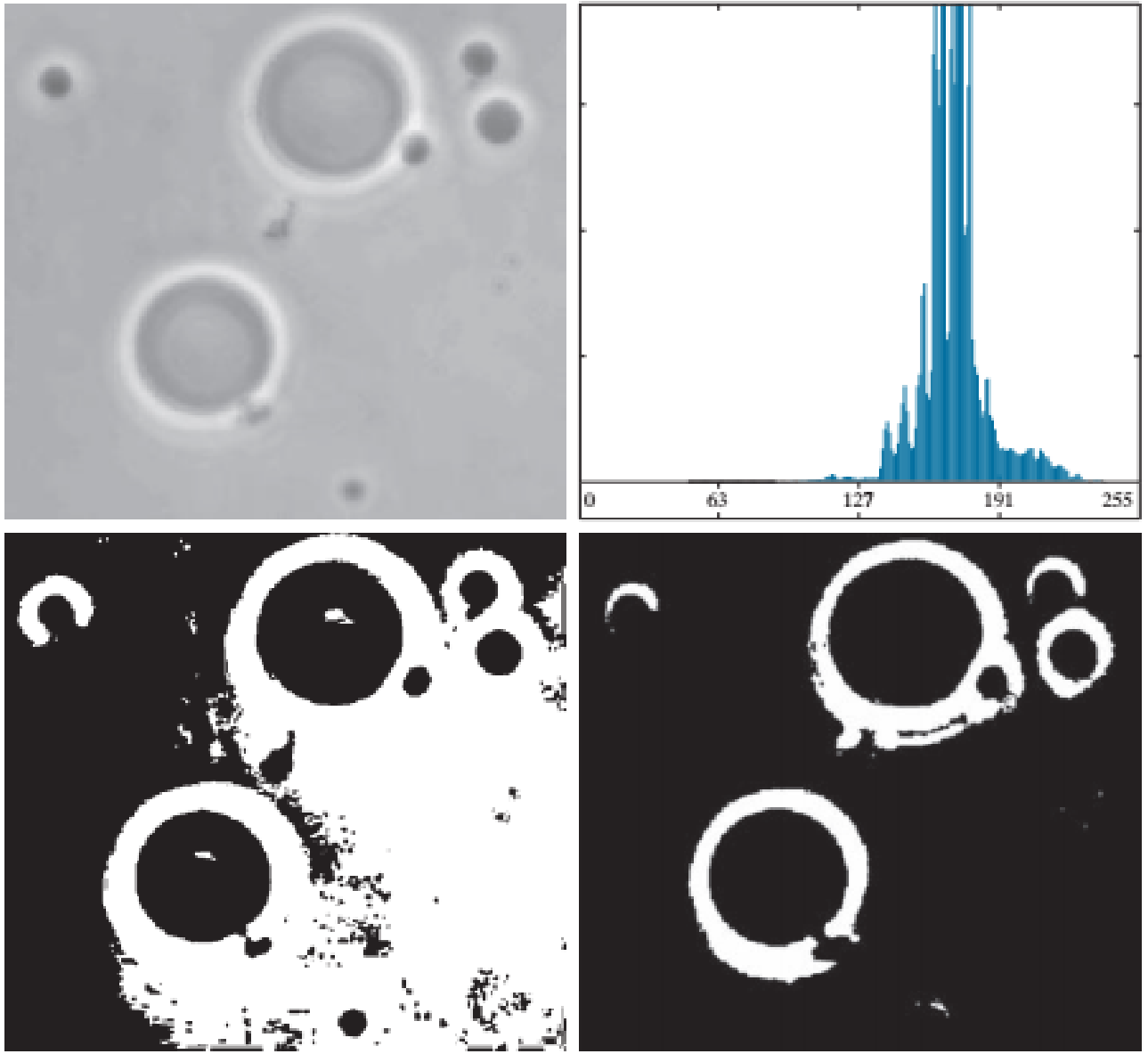


Otsu's Method

Otsu's algorithm may be summarized as follows:

1. Compute the normalized histogram of the input image. Denote the components of the histogram by p_i, $i = 0, 1, 2, \ldots, L - 1$.
2. Compute the cumulative sums, $P_1(k)$, for $k = 0, 1, 2, \ldots, L - 1$, using Eq. (10.3-4).
3. Compute the cumulative means, $m(k)$, for $k = 0, 1, 2, \ldots, L - 1$, using Eq. (10.3-8).
4. Compute the global intensity mean, m_G, using (10.3-9).
5. Compute the between-class variance, $\sigma_B^2(k)$, for $k = 0, 1, 2, \ldots, L - 1$, using Eq. (10.3-17).
6. Obtain the Otsu threshold, k^*, as the value of k for which $\sigma_B^2(k)$ is maximum. If the maximum is not unique, obtain k^* by averaging the values of k corresponding to the various maxima detected.
7. Obtain the separability measure, η^*, by evaluating Eq. (10.3-16) at $k = k^*$.

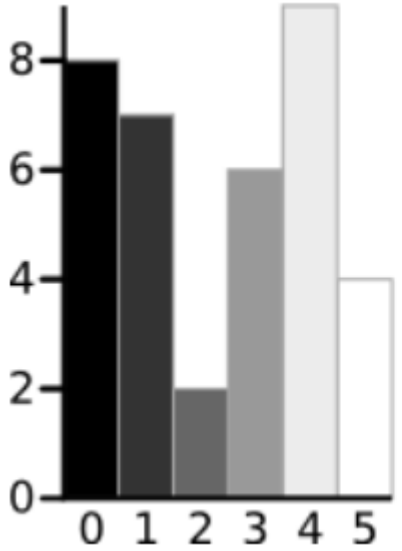
Otsu's Method



a b
c d

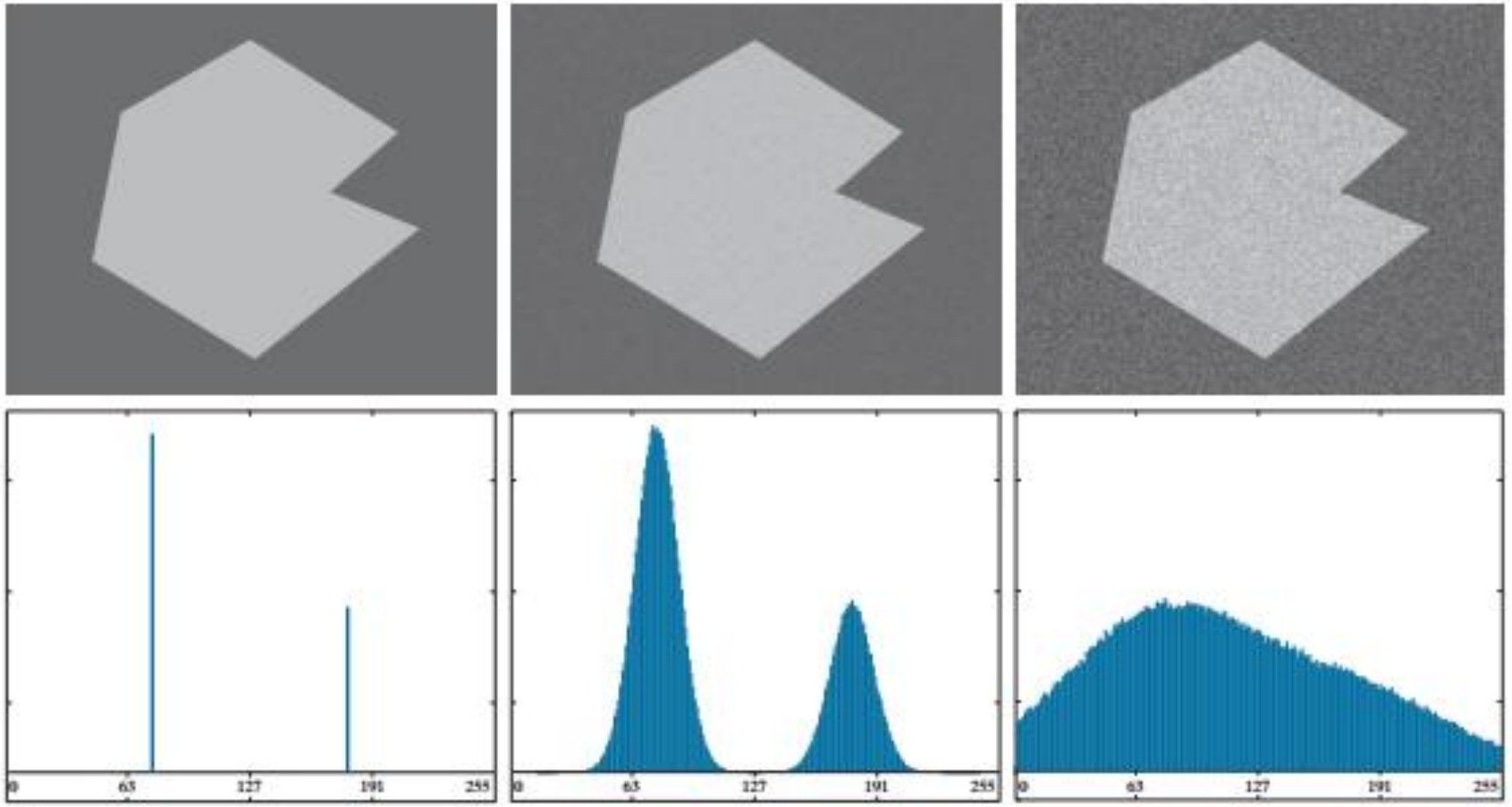
FIGURE 10.36
(a) Original image.
(b) Histogram (high peaks were clipped to highlight details in the lower values).
(c) Segmentation result using the basic global algorithm from Section 10.3.
(d) Result using Otsu's method.
(Original image courtesy of Professor Daniel A. Hammer, the University of Pennsylvania.)

Otsu - Example



		m_g	2.3611				
i	n_i	P_i	P1	i*p_i	m	(m - P1*m_g)^2	sigma2_B
0	8	0.2222	0.2222	0.0000	0.0000	0.2753	1.5928
1	7	0.1944	0.4167	0.1944	0.1944	0.6231	2.5635
2	2	0.0556	0.4722	0.1111	0.3056	0.6552	2.6287
3	6	0.1667	0.6389	0.5000	0.8056	0.4941	2.1417
4	9	0.2500	0.8889	1.0000	1.8056	0.0860	0.8705
5	4	0.1111	1.0000	0.5556	2.3611	0.0000	#DIV/0!
	36						

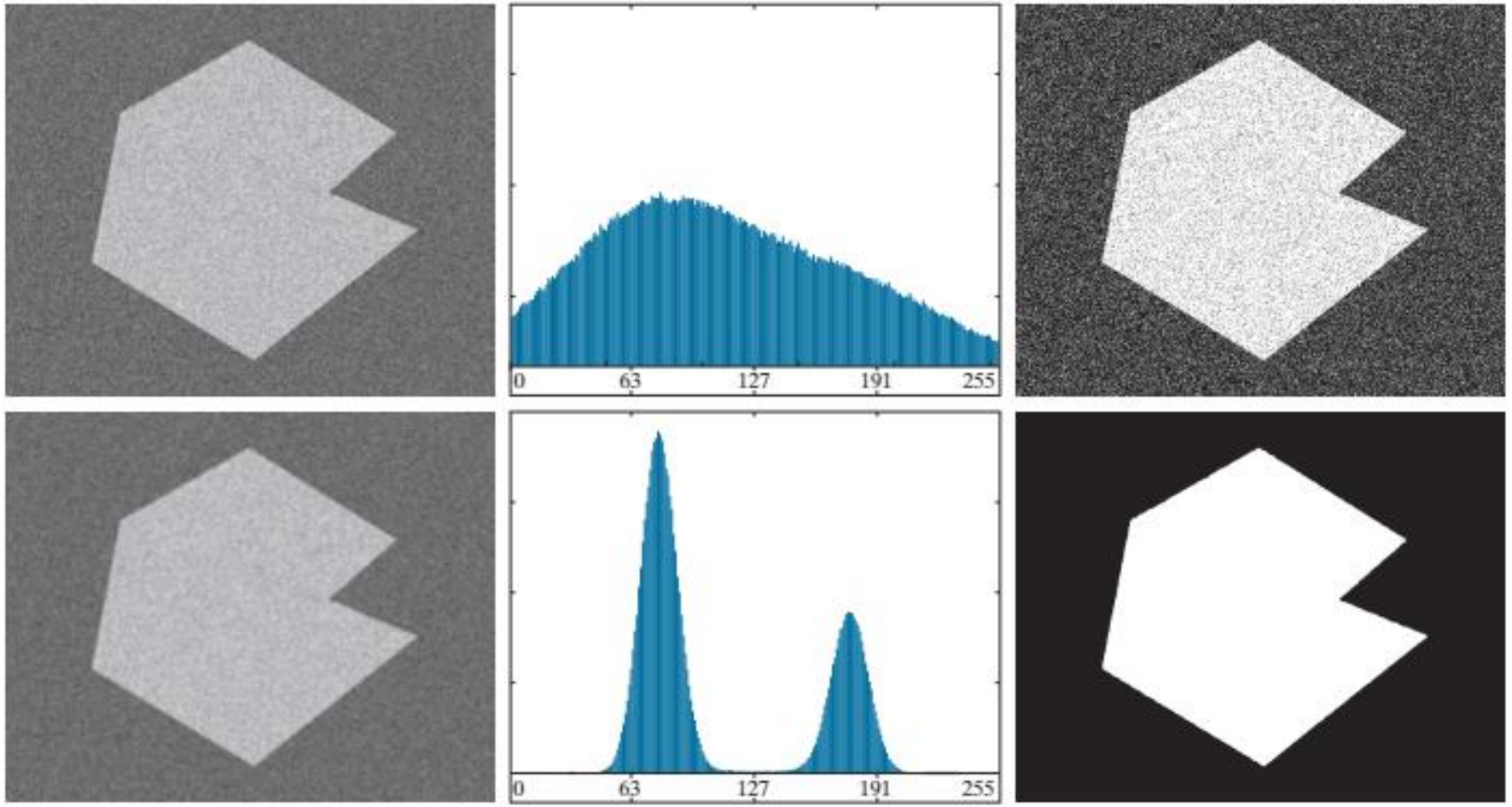
Gaussian noise added



a b c
d e f

FIGURE 10.33 (a) Noiseless 8-bit image. (b) Image with additive Gaussian noise of mean 0 and standard deviation of 10 intensity levels. (c) Image with additive Gaussian noise of mean 0 and standard deviation of 50 intensity levels. (d) through (f) Corresponding histograms.

Otsu's Method – effect of noise and smoothing

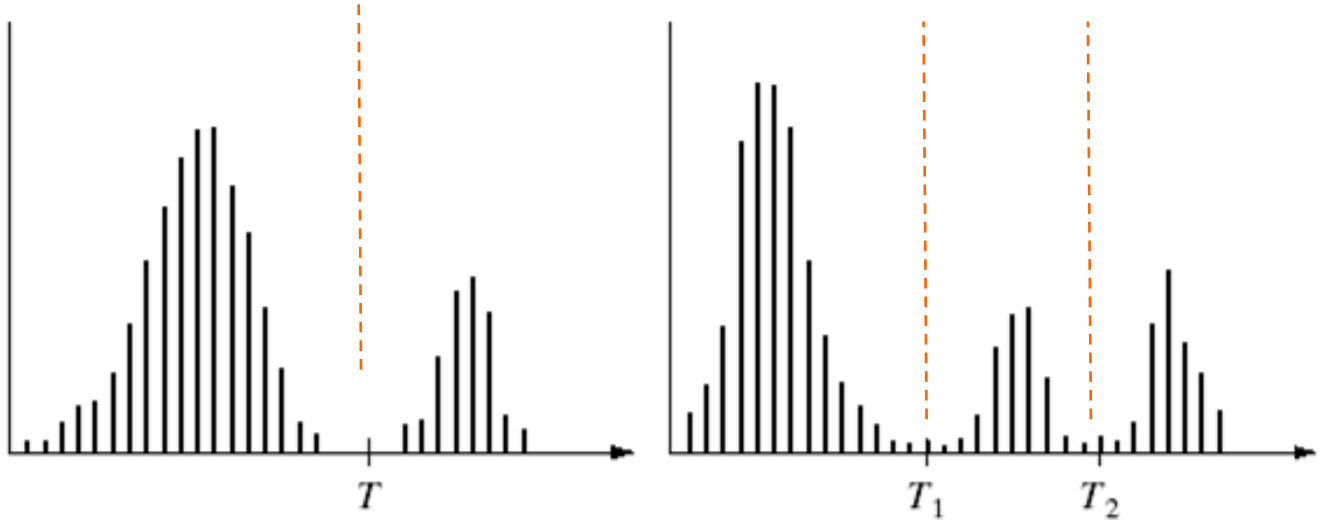


a b c
d e f

FIGURE 10.37 (a) Noisy image from Fig. 10.33(c) and (b) its histogram. (c) Result obtained using Otsu's method. (d) Noisy image smoothed using a 5×5 averaging kernel and (e) its histogram. (f) Result of thresholding using Otsu's method.

Multi-Valued Thresholding

- Single value thresholding only works for bimodal histograms
- Images with other kinds of histograms need more than a single threshold



Multi-Valued Thresholding

because the separability measure on which it is based also extends to an arbitrary number of classes (Fukunaga [1972]). In the case of K classes, c1, c2, c3 , …, K, the between-class variance generalizes to the expression

$$\sigma_B^2 = \sum_{k=1}^{K} P_k (m_k - m_G)^2 \qquad (10\text{-}66)$$

where

$$P_k = \sum_{i \in c_k} p_i \qquad (10\text{-}67)$$

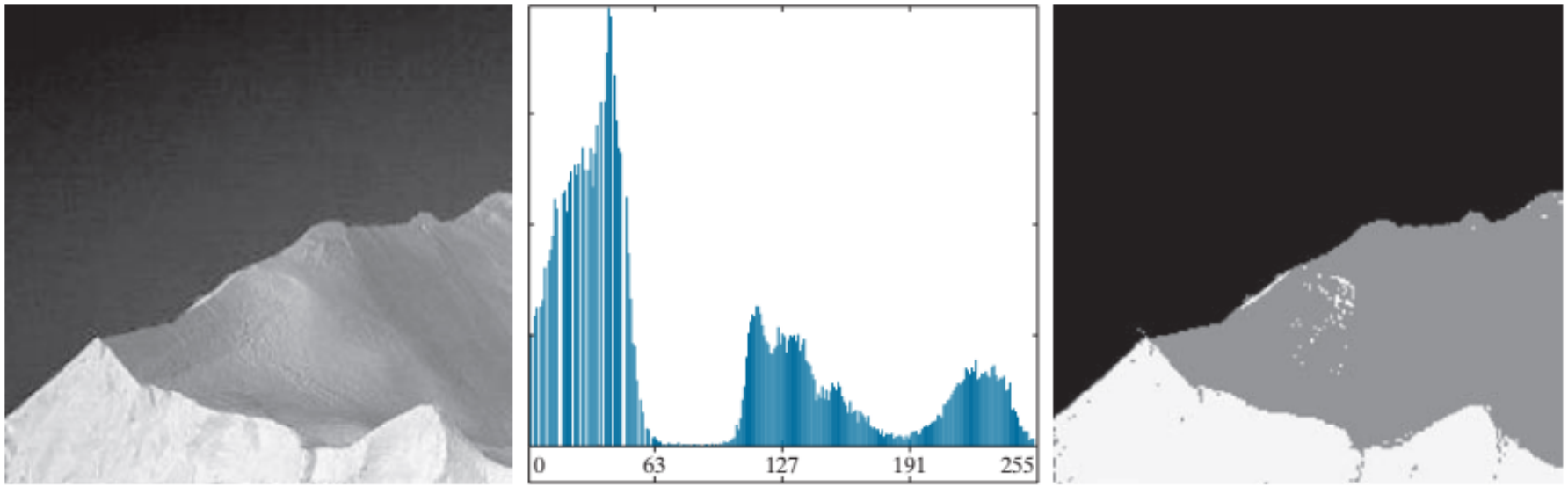
and

$$m_k = \frac{1}{P_k} \sum_{i \in c_k} i p_i \qquad (10\text{-}68)$$

As before, m_G is the global mean given in Eq. (10-54). The K classes are separated by $K-1$ thresholds whose values, $k_1^*, k_2^*, \ldots, k_{K-1}^*$, are the values that maximize Eq. (10-66):

$$\sigma_B^2 (k_1^*, k_2^*, \ldots, k_{K-1}^*) = \max_{0 < k_1 < k_2 < \ldots k_K < L-1} \sigma_B^2 (k_1, k_2, \ldots k_{K-1}) \qquad (10\text{-}69)$$

Multi-Valued Thresholding



a b c

FIGURE 10.42 (a) Image of an iceberg. (b) Histogram. (c) Image segmented into three regions using dual Otsu thresholds. (Original image courtesy of NOAA.)

MAXIMUM ENTROPY THRESHOLDING

Section 11.1.5

Digital Image Processing: An Algorithmic Introduction Using Java - Wilhelm Burger • Mark J. Burge

Maximum Entropy Thresholding

- The entropy statistic is high if a variable is well distributed over the available range, and low if it is well ordered and narrowly distributed: specifically, entropy is a measure of disorder, and is zero for a perfectly ordered system.
- The concept of entropy thresholding is to threshold at an intensity for which the sum of the entropies of the two intensity probability distributions thereby separated is maximized.
- The reason for this is to obtain the greatest reduction in entropy—i.e., the greatest increase in order—by applying the threshold: in other words, the most appropriate threshold level is the one that imposes the greatest order on the system, and thus leads to the most meaningful result

Maximum Entropy Thresholding

Z = {0, 1, . . . , K−1} ,

possible intensity values g = 0, . . . , K −1

$$p(g) \approx \mathsf{p}(g) = \frac{\mathsf{h}(g)}{MN} \qquad \sum_{g=0}^{K-1} \mathsf{p}(g) = 1$$

$$\mathsf{P}(g) = \sum_{i=0}^{g} \frac{\mathsf{h}(g)}{MN} = \sum_{i=0}^{g} \mathsf{p}(i)$$

P(0) = p(0) and P(K −1) = 1.

Entropy of image

$$H(Z) = \sum_{g \in Z} \mathsf{p}(g) \cdot \log_b\left(\frac{1}{\mathsf{p}(g)}\right) = -\sum_{g \in Z} \mathsf{p}(g) \cdot \log_b\left(\mathsf{p}(g)\right)$$

Maximum Entropy Thresholding

Given a particular threshold q (with $0 \leq q < K\text{-}1$), the estimated probability distributions for the resulting partitions C_0 and C_1

$$\mathcal{C}_0 : \left(\frac{p(0)}{P_0(q)} \;\; \frac{p(1)}{P_0(q)} \;\; \cdots \;\; \frac{p(q)}{P_0(q)} \;\; 0 \;\; 0 \;\; \cdots \;\; 0 \right)$$

$$\mathcal{C}_1 : \left(0 \;\; 0 \;\; \cdots \;\; 0 \;\; \frac{p(q+1)}{P_1(q)} \;\; \frac{p(q+2)}{P_1(q)} \;\; \cdots \;\; \frac{p(K-1)}{P_1(q)} \right)$$

$$P_0(q) = \sum_{i=0}^{q} p(i) = P(q) \qquad \text{and} \qquad P_1(q) = \sum_{i=q+1}^{K-1} p(i) = 1 - P(q)$$

$$P_0(q) + P_1(q) = 1$$

$$H_0(q) = -\sum_{i=0}^{q} \frac{p(i)}{P_0(q)} \cdot \log\left(\frac{p(i)}{P_0(q)}\right),$$

$$H_1(q) = -\sum_{i=q+1}^{K-1} \frac{p(i)}{P_1(q)} \cdot \log\left(\frac{p(i)}{P_1(q)}\right)$$

overall entropy $H_{01}(q) = H_0(q) + H_1(q)$ is to be maximized

Maximum Entropy Thresholding

Rearranging

$$
\begin{aligned}
H_0(q) &= -\sum_{i=0}^{q} \frac{\mathrm{p}(i)}{\mathrm{P}_0(q)} \cdot \big[\log\big(\mathrm{p}(i)\big) - \log\big(\mathrm{P}_0(q)\big)\big] \\
&= -\frac{1}{\mathrm{P}_0(q)} \cdot \sum_{i=0}^{q} \mathrm{p}(i) \cdot \big[\log\big(\mathrm{p}(i)\big) - \log\big(\mathrm{P}_0(q)\big)\big] \\
&= -\frac{1}{\mathrm{P}_0(q)} \cdot \underbrace{\sum_{i=0}^{q} \mathrm{p}(i) \cdot \log\big(\mathrm{p}(i)\big)}_{S_0(q)} + \frac{1}{\mathrm{P}_0(q)} \cdot \underbrace{\sum_{i=0}^{q} \mathrm{p}(i)}_{=\mathrm{P}_0(q)} \cdot \log\big(\mathrm{P}_0(q)\big) \\
&= -\frac{1}{\mathrm{P}_0(q)} \cdot S_0(q) + \log\big(\mathrm{P}_0(q)\big).
\end{aligned}
$$

Similarly $H_1(q)$ in Eqn. (11.34) becomes

$$
\begin{aligned}
H_1(q) &= -\sum_{i=q+1}^{K-1} \frac{\mathrm{p}(i)}{\mathrm{P}_1(q)} \cdot \big[\log\big(\mathrm{p}(i)\big) - \log\big(\mathrm{P}_1(q)\big)\big] \\
&= -\frac{1}{1-\mathrm{P}_0(q)} \cdot S_1(q) + \log\big(1-\mathrm{P}_0(q)\big).
\end{aligned}
$$

Maximum Entropy Thresholding

$$P_0(q) = \begin{cases} p(0) & \text{for } q = 0, \\ P_0(q-1) + p(q) & \text{for } 0 < q < K, \end{cases}$$

$$S_0(q) = \begin{cases} p(0) \cdot \log(p(0)) & \text{for } q = 0, \\ S_0(q-1) + p(q) \cdot \log(p(q)) & \text{for } 0 < q < K, \end{cases}$$

$$S_1(q) = \begin{cases} 0 & \text{for } q = K-1, \\ S_1(q+1) + p(q+1) \cdot \log(p(q+1)) & \text{for } 0 \leq q < K-1 \end{cases}$$

the values $S_0(q)$, $S_1(q)$ are obtained from precalculated tables S_0, S_1

Maximum Entropy Thresholding

MaximumEntropyThreshold(h)

Input: $\mathsf{h} : [0, K-1] \mapsto \mathbb{N}$, a grayscale histogram. Returns the optimal threshold value or -1 if no threshold is found.

$K \leftarrow \mathsf{Size}(\mathsf{h})$ ▷ number of intensity levels

$\mathsf{p} \leftarrow \mathsf{Normalize}(\mathsf{h})$ ▷ normalize histogram

$(\mathsf{S}_0, \mathsf{S}_1) \leftarrow \mathsf{MakeTables}(\mathsf{p}, K)$ ▷ tables for $S_0(q), S_1(q)$

$\mathsf{P}_0 \leftarrow 0$ ▷ $\mathsf{P}_0 \in [0, 1]$

$q_{\max} \leftarrow -1$

$H_{\max} \leftarrow -\infty$ ▷ maximum joint entropy

for $q \leftarrow 0, \cdots, K-2$ **do** ▷ check all possible threshold values q

$\mathsf{P}_0 \leftarrow \mathsf{P}_0 + \mathsf{p}(q)$

$\mathsf{P}_1 \leftarrow 1 - \mathsf{P}_0$ ▷ $\mathsf{P}_1 \in [0, 1]$

$$H_0 \leftarrow \begin{cases} -\frac{1}{\mathsf{P}_0} \cdot \mathsf{S}_0(q) + \log(\mathsf{P}_0) & \text{if } \mathsf{P}_0 > 0 \\ 0 & \text{otherwise} \end{cases}$$ ▷ BG entropy

$$H_1 \leftarrow \begin{cases} -\frac{1}{\mathsf{P}_1} \cdot \mathsf{S}_1(q) + \log(\mathsf{P}_1) & \text{if } \mathsf{P}_1 > 0 \\ 0 & \text{otherwise} \end{cases}$$ ▷ FG entropy

$H_{01} = H_0 + H_1$ ▷ *overall* entropy for q

if $H_{01} > H_{\max}$ **then** ▷ maximize $H_{01}(q)$

$H_{\max} \leftarrow H_{01}$

$q_{\max} \leftarrow q$

return $q_{\max}$

MakeTables(p, K)

Create maps $\mathsf{S}_0, \mathsf{S}_1 : [0, K-1] \mapsto \mathbb{R}$

$s_0 \leftarrow 0$

for $i \leftarrow 0, \cdots, K-1$ **do**

if $\mathsf{p}(i) > 0$ **then**

$s_0 \leftarrow s_0 + \mathsf{p}(i) \cdot \log\big(\mathsf{p}(i)\big)$

$\mathsf{S}_0(i) \leftarrow s_0$

$s_1 \leftarrow 0$

for $i \leftarrow K-1, \cdots, 0$ **do**

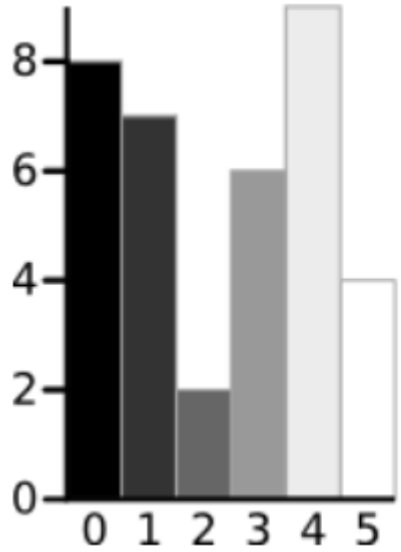
$\mathsf{S}_1(i) \leftarrow s_1$

if $\mathsf{p}(i) > 0$ **then**

$s_1 \leftarrow s_1 + \mathsf{p}(i) \cdot \log\big(\mathsf{p}(i)\big)$

return $(\mathsf{S}_0, \mathsf{S}_1)$

MET

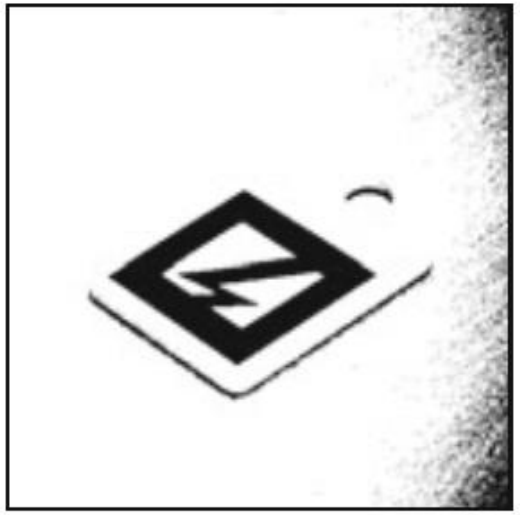


q	n_q	p(q)	P0	P1	S0	S1	H0	H1	H0+H1
0	8	0.222	0.222	0.7778	-0.145	-0.633	0	0.7052	0.7052
1	7	0.194	0.417	0.5833	-0.283	-0.488	0.3001	0.6029	0.903
2	2	0.056	0.472	0.5278	-0.353	-0.35	0.4221	0.3855	0.8076
3	6	0.167	0.639	0.3611	-0.483	-0.28	0.5612	0.3336	0.8948
4	9	0.25	0.889	0.1111	-0.633	-0.151	0.6614	0.4004	1.0618
5	4	0.111	1	0	-0.739	0	0.7394	#DIV/0!	#DIV/0!
	36								

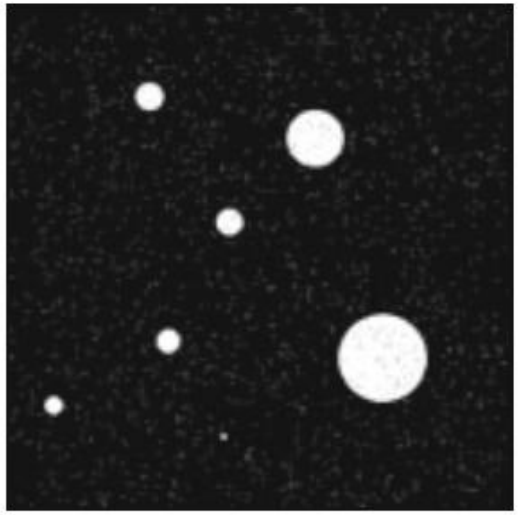
Maximum Entropy Thresholding

(a) $q = 133$

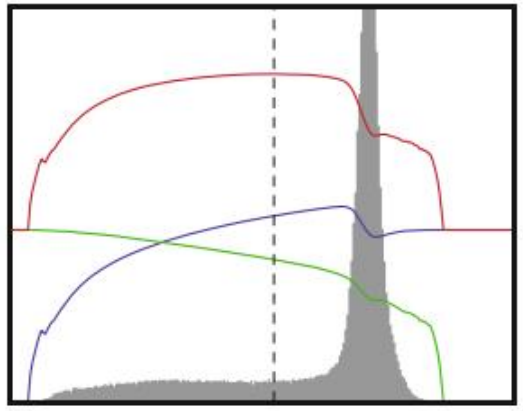
(b) $q = 139$



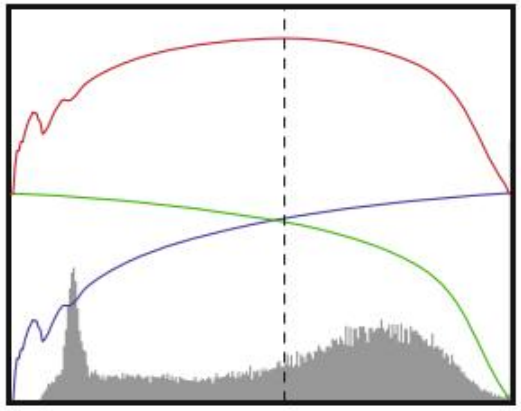
(c) $q = 118$



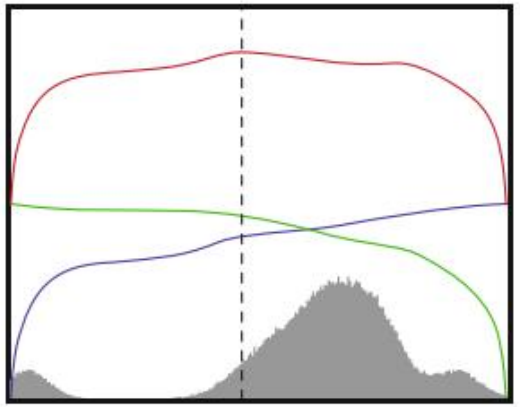
(d) $q = 126$



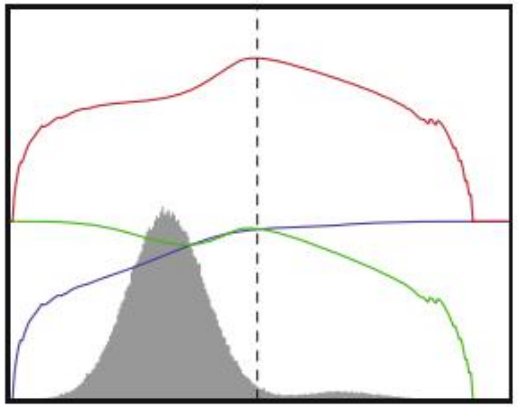
(e)



(f)



(g)

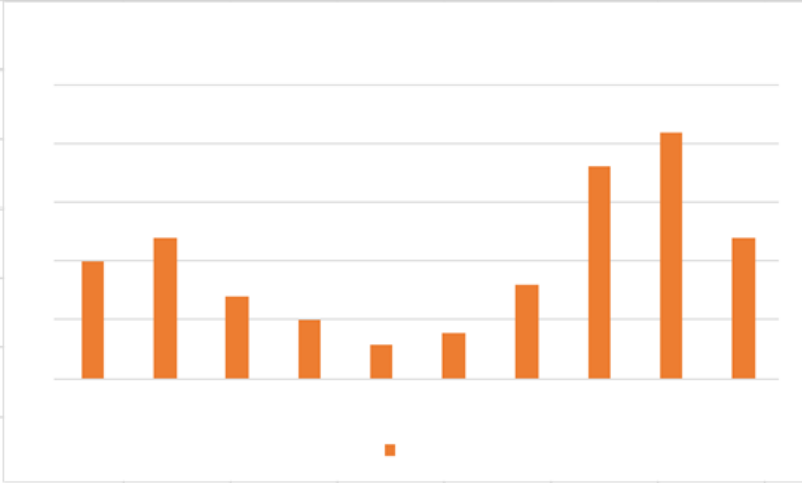


(h)

Graphs in (e–h) the background (green), foregro 1(q) (blue) and H01(q) = H0(q) (red)

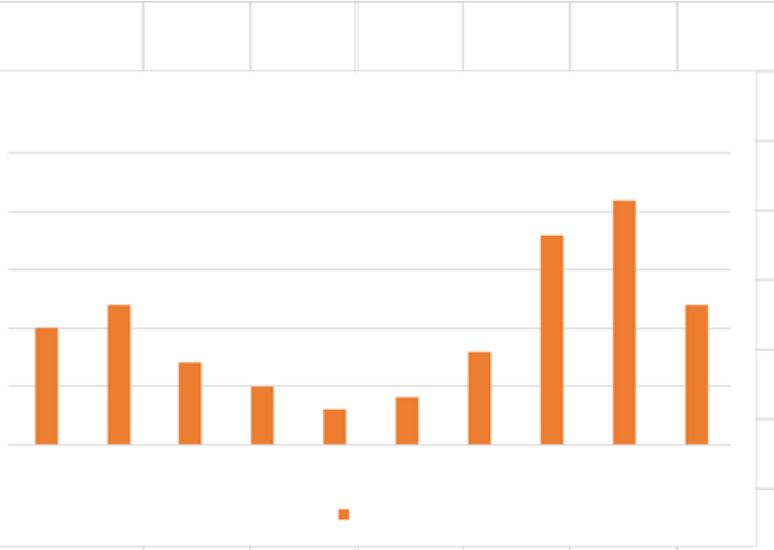
Exercise – Otsu

i	n_i	p_i	P1	i*p_i	m	(m - P1*m_g)^ 2	sigma2_B
0	10	0.1	0.1	0	0	0.27353	2.4618
1	12	0.12	0.22	0.12	0.12	1.06214	3.7658
2	7	0.07	0.29	0.14	0.26	1.57929	3.8665
3	5	0.05	0.34	0.15	0.41	1.87197	3.6338
4	3	0.03	0.37	0.12	0.53	1.97431	3.3617
5	4	0.04	0.41	0.2	0.73	2.00024	2.8784
6	8	0.08	0.49	0.48	1.21	1.8298	1.9045
7	18	0.18	0.67	1.26	2.47	1.06936	0.5267
8	21	0.21	0.88	1.68	4.15	0.20467	0.0279
9	12	0.12	1	1.08	5.23	7.9E-31	9E-47
	100			mg	5.23		

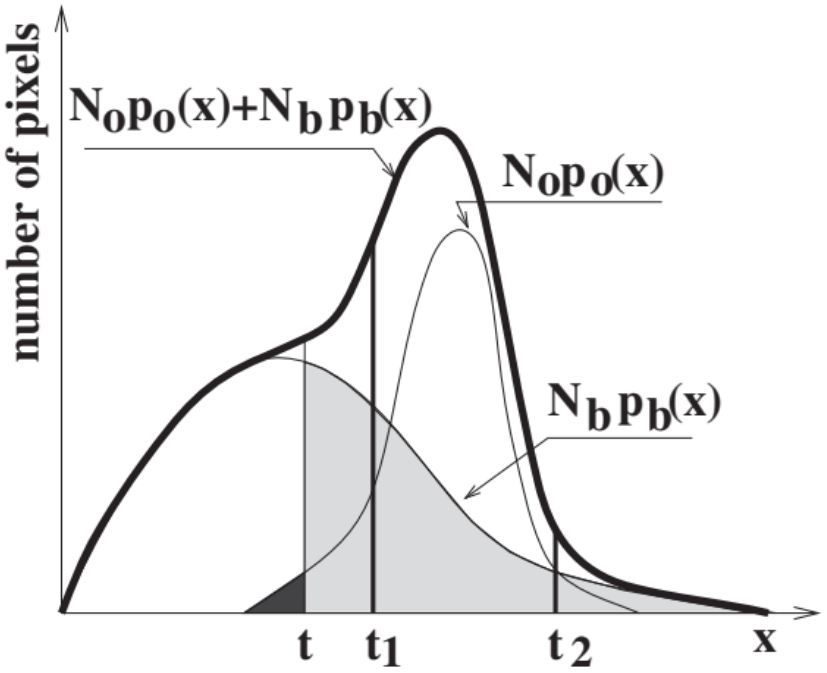


Exercise - MET

q	n_q	p(q)	P0	P1	S0	S1	H0	H1	H0+H1
0	10	0.1	0.1	0.9	-0.1	-0.822	0	0.8677	0.8677
1	12	0.12	0.22	0.78	-0.21	-0.722	0.2992	0.8179	1.1171
2	7	0.07	0.29	0.71	-0.291	-0.612	0.467	0.7127	1.1797
3	5	0.05	0.34	0.66	-0.356	-0.531	0.5797	0.6238	1.2035
4	3	0.03	0.37	0.63	-0.402	-0.466	0.6549	0.5386	1.1935
5	4	0.04	0.41	0.59	-0.458	-0.42	0.7298	0.4828	1.2127
6	8	0.08	0.49	0.51	-0.546	-0.364	0.804	0.4216	1.2255
7	18	0.18	0.67	0.33	-0.68	-0.276	0.8407	0.356	1.1967
8	21	0.21	0.88	0.12	-0.822	-0.142	0.8787	0.2653	1.144
9	12	0.12	1	0	-0.933	0	0.9326	#DIV/0!	#DIV/0!
	100								



How can we choose the minimum error threshold?



Fraction of the pixels that make up the object is θ, and, by inference, the fraction of the pixels that
make up the background is 1 − θ. Then, the total error is

$$E(t) = \theta \int_{-\infty}^{t} p_o(x)dx + (1-\theta)\int_{t}^{+\infty} p_b(x)dx$$

$$\frac{\partial E}{\partial t} = \theta p_o(t) - (1-\theta)p_b(t) = 0$$

$$\theta p_o(t) = (1-\theta)p_b(t)$$

error committed by misclassifying object pixels as background pixels $\int_{-\infty}^{t} p_o(x)dx$

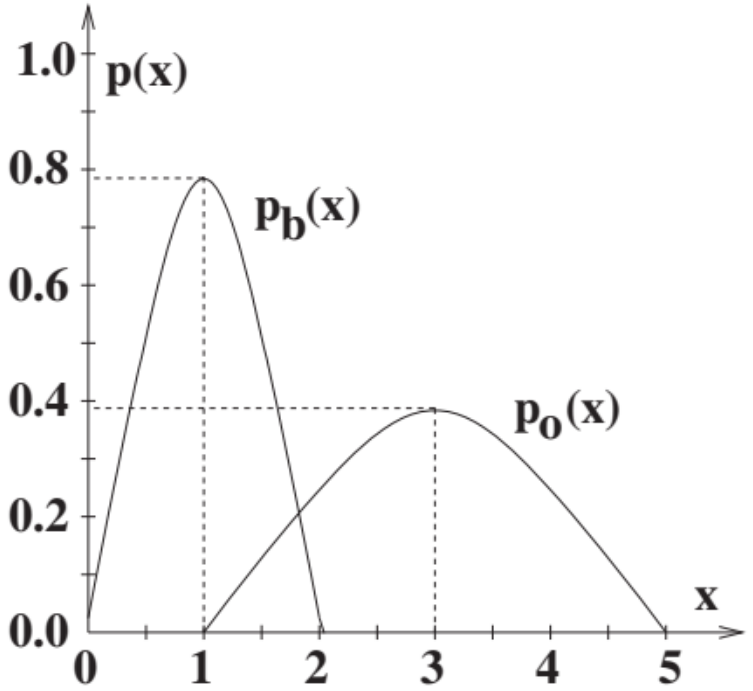
error committed by misclassifying background pixels as object pixels $\int_{t}^{+\infty} p_b(x)dx$

How can we choose the minimum error threshold?

- The grey values of the object and the background pixels are distributed according to the probability density function

$$p(x) = \begin{cases} \frac{\pi}{4a} \cos \frac{(x-x_0)\pi}{2a} & \textbf{for } x_0 - a \le x \le x_0 + a \\ 0 & \textbf{otherwise} \end{cases}$$

with $x_0 = 1$ and $a = 1$ for the objects, and $x_0 = 3$ and $a = 2$ for the background. Sketch the two probability density functions.



- If one-third of the total number of pixels are object pixels, determine the fraction of misclassified object pixels by optimal thresholding.

How can we choose the minimum error threshold?

$$\theta = \frac{1}{3} \Rightarrow 1 - \theta = \frac{2}{3}$$

$$p_0(x) = \frac{\pi}{4} \cos \frac{(x-1)\pi}{2}$$

$$p_b(x) = \frac{\pi}{8} \cos \frac{(x-3)\pi}{4}$$

$$\frac{1}{3} \times \frac{\pi}{4} \cos \frac{(t-1)\pi}{2} = \frac{2}{3} \times \frac{\pi}{8} \cos \frac{(t-3)\pi}{4}$$

$$\Rightarrow \quad \cos \frac{(t-1)\pi}{2} = \cos \frac{(t-3)\pi}{4}$$

$$\Rightarrow \quad \frac{(t-1)\pi}{2} = \pm \frac{(t-3)\pi}{4}$$

How can we choose the minimum error threshold?

Consider first $\frac{t-1}{2}\pi = \frac{t-3}{4}\pi \Rightarrow 2t-2 = t-3 \Rightarrow t = -1.$
This value is outside the acceptable range, so it is a meaningless solution.
Then:

$$\frac{(t-1)\pi}{2} = -\frac{(t-3)\pi}{4} \Rightarrow 2t-2 = -t+3 \Rightarrow 3t = 5 \Rightarrow t = \frac{5}{3}$$

$$\begin{aligned}
\int_{\frac{5}{3}}^{2} \frac{\pi}{4}\cos\frac{(x-1)\pi}{2}dx &= \frac{\pi}{4}\int_{\frac{2}{3}}^{1}\cos\frac{y\pi}{2}dy \\
&= \frac{\pi}{4}\left.\frac{\sin\frac{y\pi}{2}}{\frac{\pi}{2}}\right|_{\frac{2}{3}}^{1} \\
&= \frac{1}{2}\left(\sin\frac{\pi}{2} - \sin\frac{\pi}{3}\right) \\
&= \frac{1}{2}(1-\sin 60^o)
\end{aligned}
\qquad
\begin{aligned}
&= \frac{1}{2}\left(1-\frac{\sqrt{3}}{2}\right) \\
&= \frac{2-1.7}{4} \\
&= \frac{0.3}{4} \\
&= 0.075 = 7.5\%
\end{aligned}$$

What is the minimum error threshold when object and background pixels arenormally distributed?

$$p_o(x) = \frac{1}{\sqrt{2\pi}\sigma_o}\exp\left[-\frac{(x-\mu_o)^2}{2\sigma_o^2}\right] \qquad p_b(x) = \frac{1}{\sqrt{2\pi}\sigma_b}\exp\left[-\frac{(x-\mu_b)^2}{2\sigma_b^2}\right]$$

$$\theta\frac{1}{\sqrt{2\pi}\sigma_o}\exp\left[-\frac{(t-\mu_o)^2}{2\sigma_o^2}\right] = (1-\theta)\frac{1}{\sqrt{2\pi}\sigma_b}\exp\left[-\frac{(t-\mu_b)^2}{2\sigma_b^2}\right] \Rightarrow$$

$$\exp\left[-\frac{(t-\mu_o)^2}{2\sigma_o^2}+\frac{(t-\mu_b)^2}{2\sigma_b^2}\right] = \frac{1-\theta}{\theta}\frac{\sigma_o}{\sigma_b} \Rightarrow$$

$$-\frac{(t-\mu_o)^2}{2\sigma_o^2}+\frac{(t-\mu_b)^2}{2\sigma_b^2} = \ln\left[\frac{\sigma_o}{\sigma_b}\frac{1-\theta}{\theta}\right] \Rightarrow$$

$$(t^2+\mu_b^2-2t\mu_b)\sigma_o^2-(t^2+\mu_o^2-2\mu_o t)\sigma_b^2 = 2\sigma_o^2\sigma_b^2\ln\left[\frac{\sigma_o}{\sigma_b}\frac{1-\theta}{\theta}\right] \Rightarrow$$

$$(\sigma_o^2-\sigma_b^2)t^2+2(-\mu_b\sigma_o^2+\mu_o\sigma_b^2)t+\mu_b^2\sigma_o^2-\mu_o^2\sigma_b^2-2\sigma_o^2\sigma_b^2\ln\left[\frac{\sigma_o}{\sigma_b}\frac{1-\theta}{\theta}\right] = 0$$

What is the minimum error threshold when object and background pixels arenormally distributed?

This is a quadratic equation in t. It has two solutions in general, except when the two populations have the same standard deviation. If $\sigma_o = \sigma_b$, the above expression takes the form:

$$2(\mu_o - \mu_b)\sigma_o^2 t + (\mu_b^2 - \mu_o^2)\sigma_o^2 - 2\sigma_o^4 \ln\left(\frac{1-\theta}{\theta}\right) = 0 \Rightarrow$$

$$t = \frac{\sigma_o^2}{\mu_o - \mu_b} \ln\left(\frac{1-\theta}{\theta}\right) - \frac{\mu_o + \mu_b}{2} \tag{6.15}$$

This is the minimum error threshold.

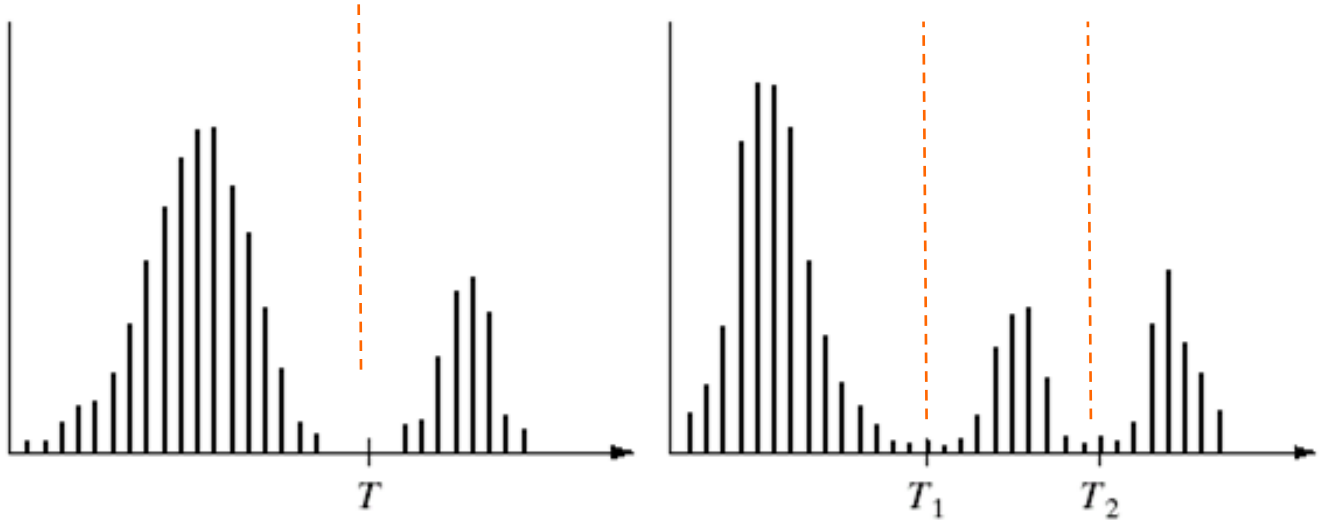
Are there any drawbacks in Otsu's method?

Yes, a few:

1. Although the method does not make any assumption about the probability density functions $p_o(x)$ and $p_b(x)$, it describes them by using only their means and variances. Thus, it tacitly assumes that these two statistics are sufficient to represent them. This may not be true.
2. The method breaks down when the two populations are very unequal. When the two populations become very different in size from each other, $\sigma_B^2(t)$ may have two maxima and actually the correct maximum is not necessarily the global maximum. That is why in practice the correct maximum is selected from among all maxima of $\sigma_B^2(t)$ by checking that the value of the histogram at the selected threshold, p_t, is actually a valley (i.e. $p_t < p_{\mu_o}$ and $p_t < p_{\mu_b}$). Only if this is true, t should be accepted as the best threshold.
3. The method, as presented above, assumes that the histogram of the image is bimodal, ie that the image contains two classes. For more than two classes present in the image, the method has to be modified so that multiple thresholds are defined which maximise the *interclass* variance and minimise the *intraclass* variance.
4. The method will divide the image into two classes, even if this division does not make sense. A case when the method should not be directly applied is that of variable illumination.

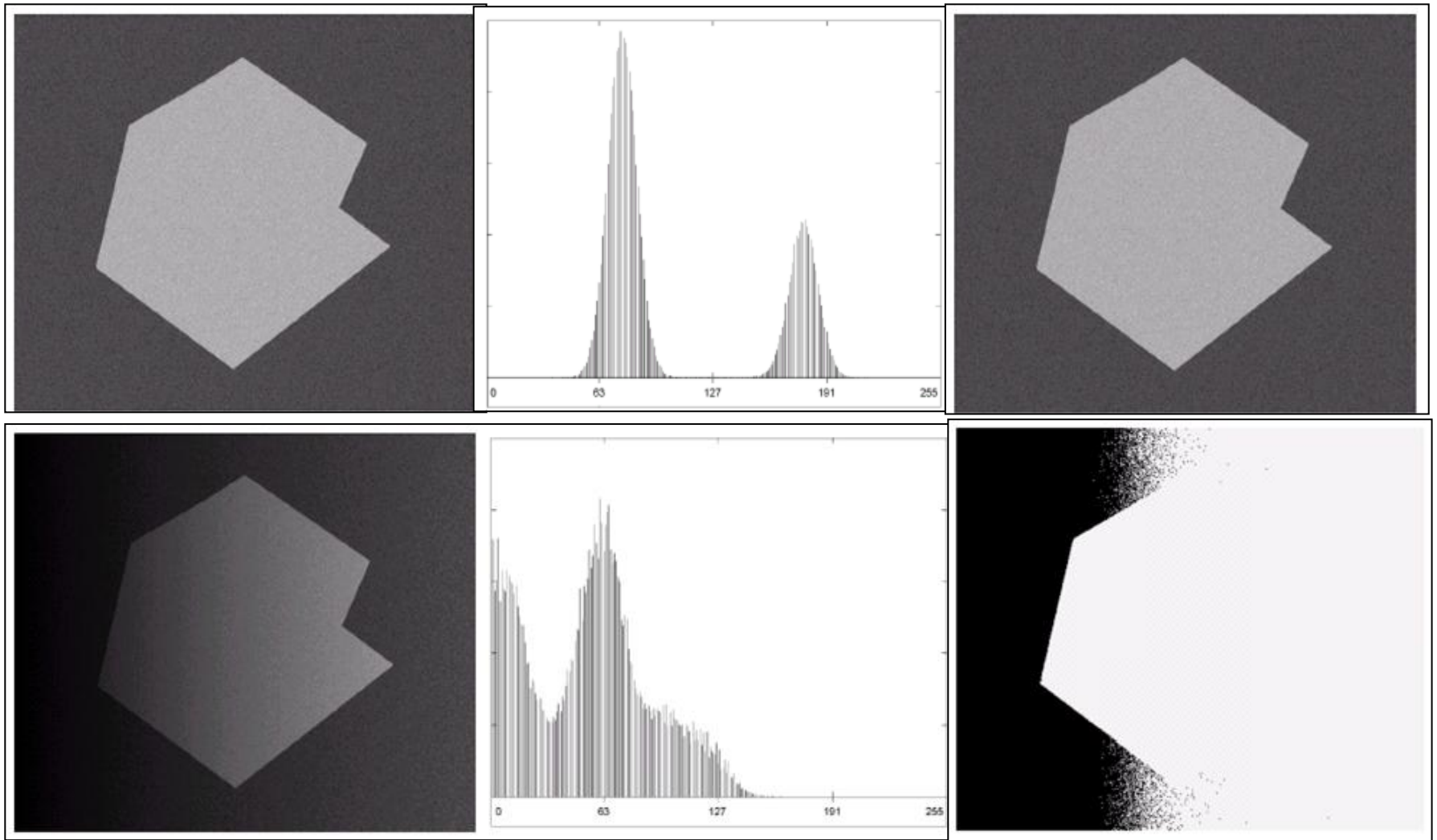
Problems With Single Value Thresholding

- Single value thresholding only works for bimodal histograms
- Images with other kinds of histograms need more than a single threshold



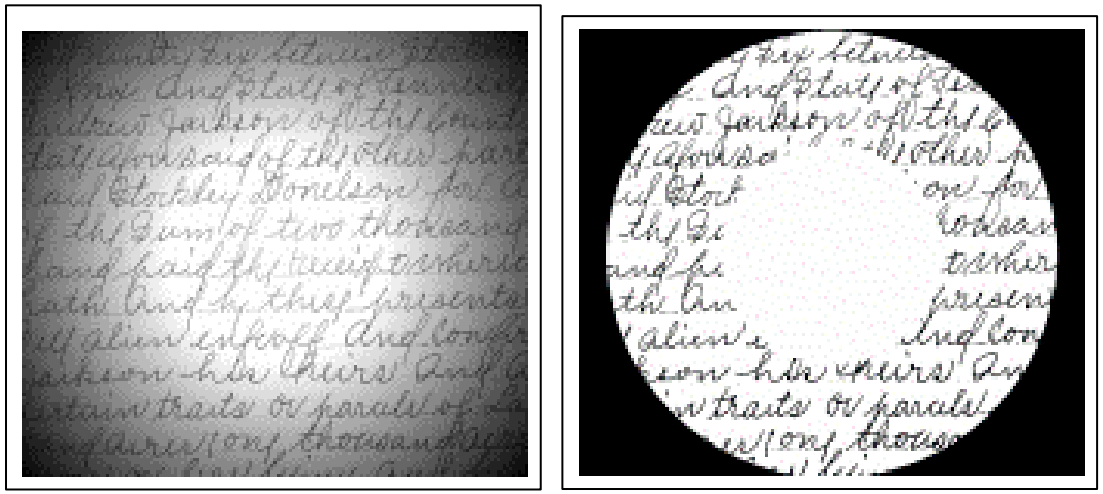
Single Value Thresholding - Illumination

- Uneven illumination can really upset a single valued thresholding scheme

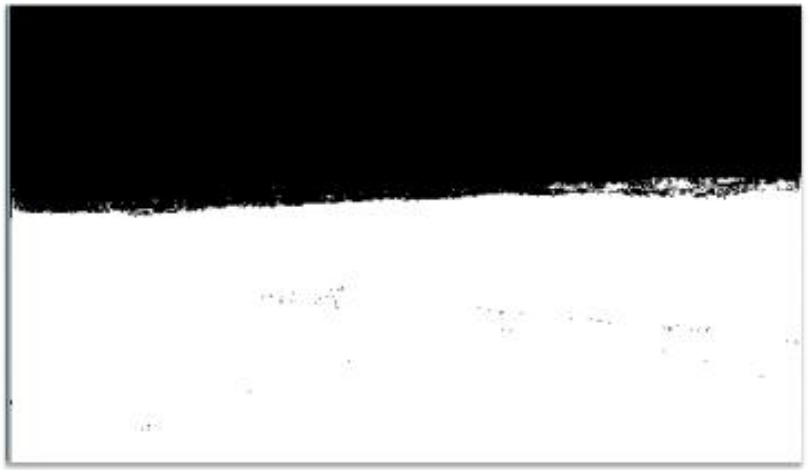


Single Value Thresholding - Illumination

- Uneven illumination or low light



Thermal image

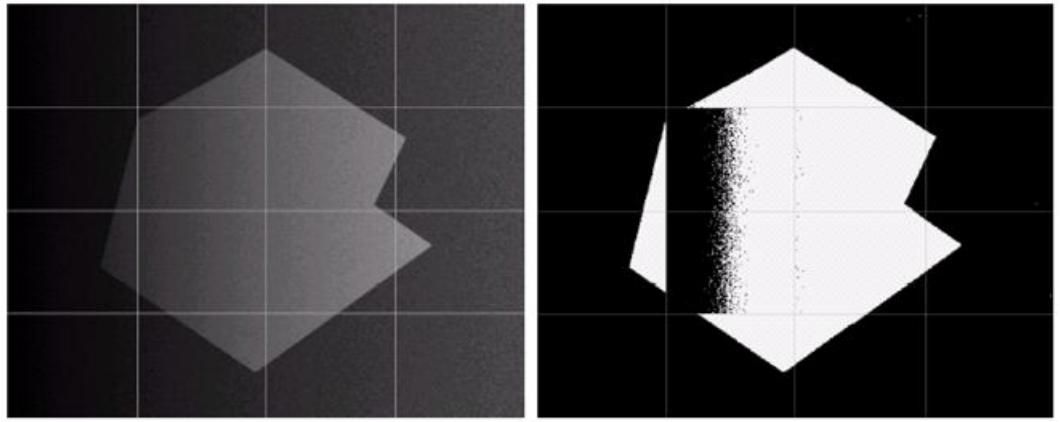


Thresholded image (Otsu's method)

Adaptive Thresholding

- **Partitioning**
- An approach to handling situations in which single value thresholding will not work is to divide an image into sub images and threshold these individually
- Since the threshold for each pixel depends on its location within an image this technique is said to *adaptive*

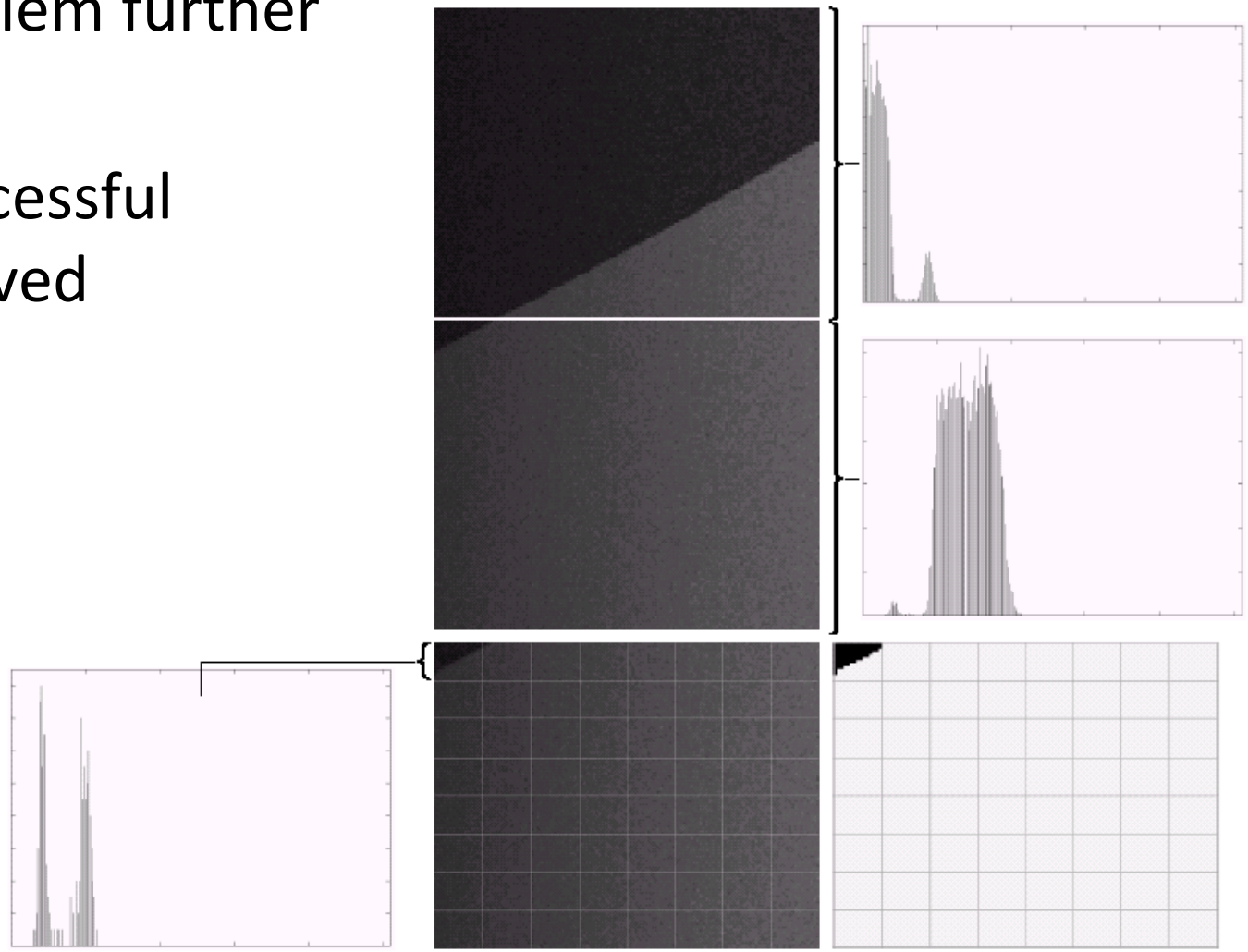
Adaptive Thresholding - partitioning



- As can be seen success is mixed
- It is work when the objects of interest and the background occupy regions of reasonably comparable size. If not , it will fail.
- But, we can further subdivide the troublesome sub images for more success

Adaptive Thresholding Example (cont…)

- These images show the troublesome parts of the previous problem further subdivided
- After this sub division successful thresholding can be achieved



Adaptive Thresholding

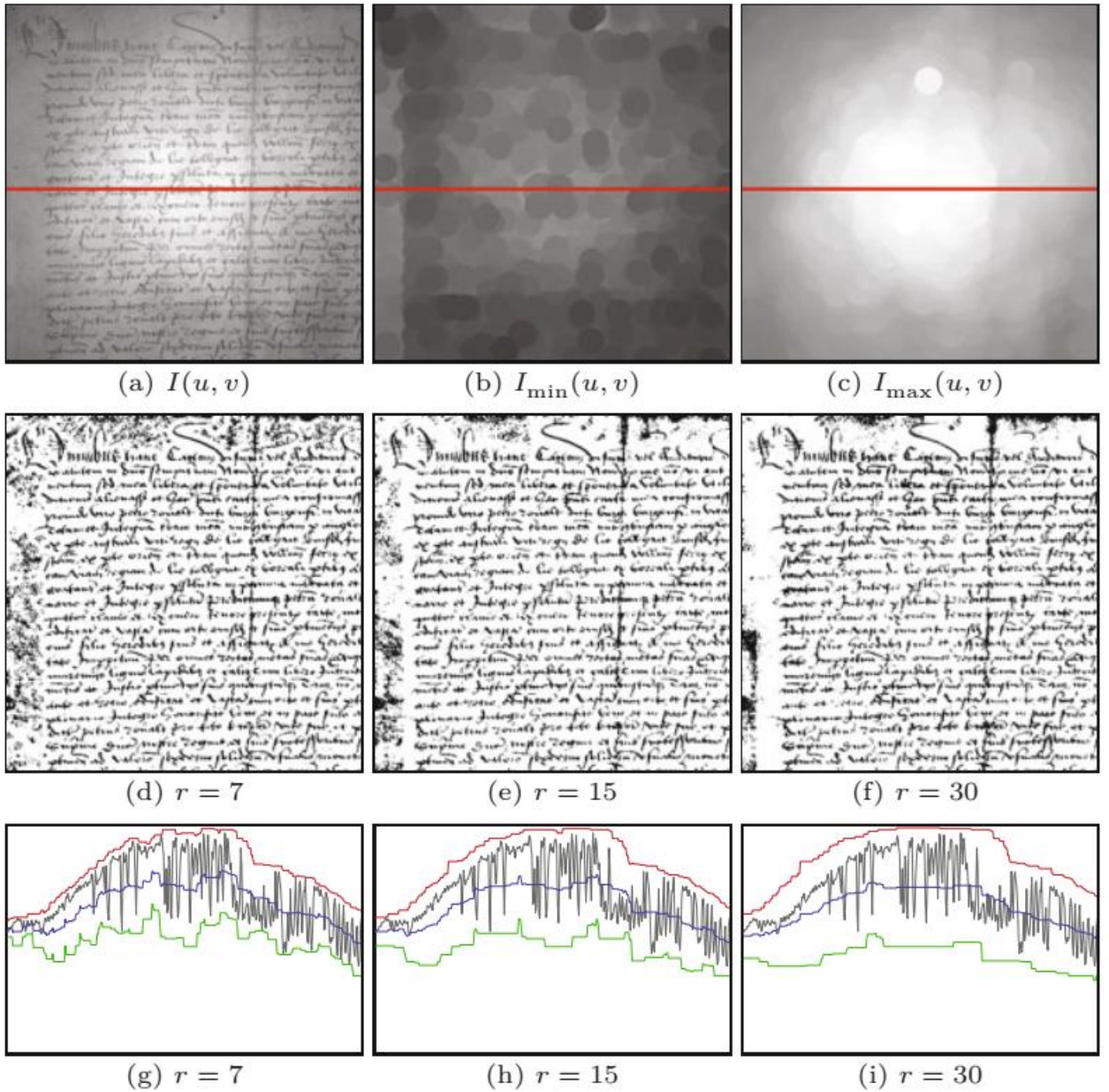
- Bernsen's Method

$$I_{\min}(u,v) = \min_{\substack{(i,j)\in \\ R(u,v)}} I(i,j),$$

$$I_{\max}(u,v) = \max_{\substack{(i,j)\in \\ R(u,v)}} I(i,j)$$

$$Q(u,v) = \frac{I_{\min}(u,v) + I_{\max}(u,v)}{2}$$

This is done as long as the local contrast $c(u, v) = I_{max}(u, v) - I_{min}(u, v)$ is above some predefined limit c_{min}



local minimum (green), maximum (red), and the actual threshold (blue)

Adaptive Thresholding

- Niblack's Method
  - based on local image properties
  - threshold $Q(u, v)$ is varied across the image as a function of the local intensity average $\mu_R(u, v)$ and standard deviation $\sigma_R(u, v)$

$$Q(u,v) := \mu_R(u,v) + \kappa \cdot \sigma_R(u,v) \;;\; \kappa \geq 0$$

- Or, to avoid low-amplitude noise ("ghosting") $Q(u,v) := \mu_R(u,v) + \kappa \cdot \sigma_R(u,v) + d$

- the structures of interest are darker than the background (as, e.g., in typical OCR applications), one could either work with inverted images or modify the calculation of the threshold

$$Q(u,v) := \begin{cases} \mu_R(u,v) + (\kappa \cdot \sigma_R(u,v) + d) & \text{for } \textit{dark} \text{ BG}, \\ \mu_R(u,v) - (\kappa \cdot \sigma_R(u,v) + d) & \text{for } \textit{bright} \text{ BG} \end{cases}$$

Adaptive Thresholding

NiblackThreshold(I, r, κ, d, bg)

Input: I, intensity image of size $M \times N$; r, radius of support region; κ, variance control parameter; d, minimum offset; $bg \in \{\text{dark}, \text{bright}\}$, background type. Returns a map with an individual threshold value for each image position.

$(M, N) \leftarrow \text{Size}(I)$

Create map $Q : M \times N \mapsto \mathbb{R}$

for all image coordinates $(u, v) \in M \times N$ **do**

Define a support region of radius r, centered at (u, v):

$(\mu, \sigma^2) \leftarrow \text{GetLocalMeanAndVariance}(I, u, v, r)$

$\sigma \leftarrow \sqrt{\sigma^2}$ ▷ local std. deviation σ_R

$$Q(u, v) \leftarrow \begin{cases} \mu + (\kappa \cdot \sigma + d) & \text{if } bg = \text{dark} \\ \mu - (\kappa \cdot \sigma + d) & \text{if } bg = \text{bright} \end{cases}$$ ▷ Eq. 11.72

return Q

GetLocalMeanAndVariance(I, u, v, r)

Returns the local mean and variance of the image pixels $I(i, j)$ within the disk-shaped region with radius r around position (u, v).

$R \leftarrow \text{MakeCircularRegion}(u, v, r)$

$n \leftarrow 0$

$A \leftarrow 0$

$B \leftarrow 0$

for all $(i, j) \in R$ **do**

$n \leftarrow n + 1$

$A \leftarrow A + I(i, j)$

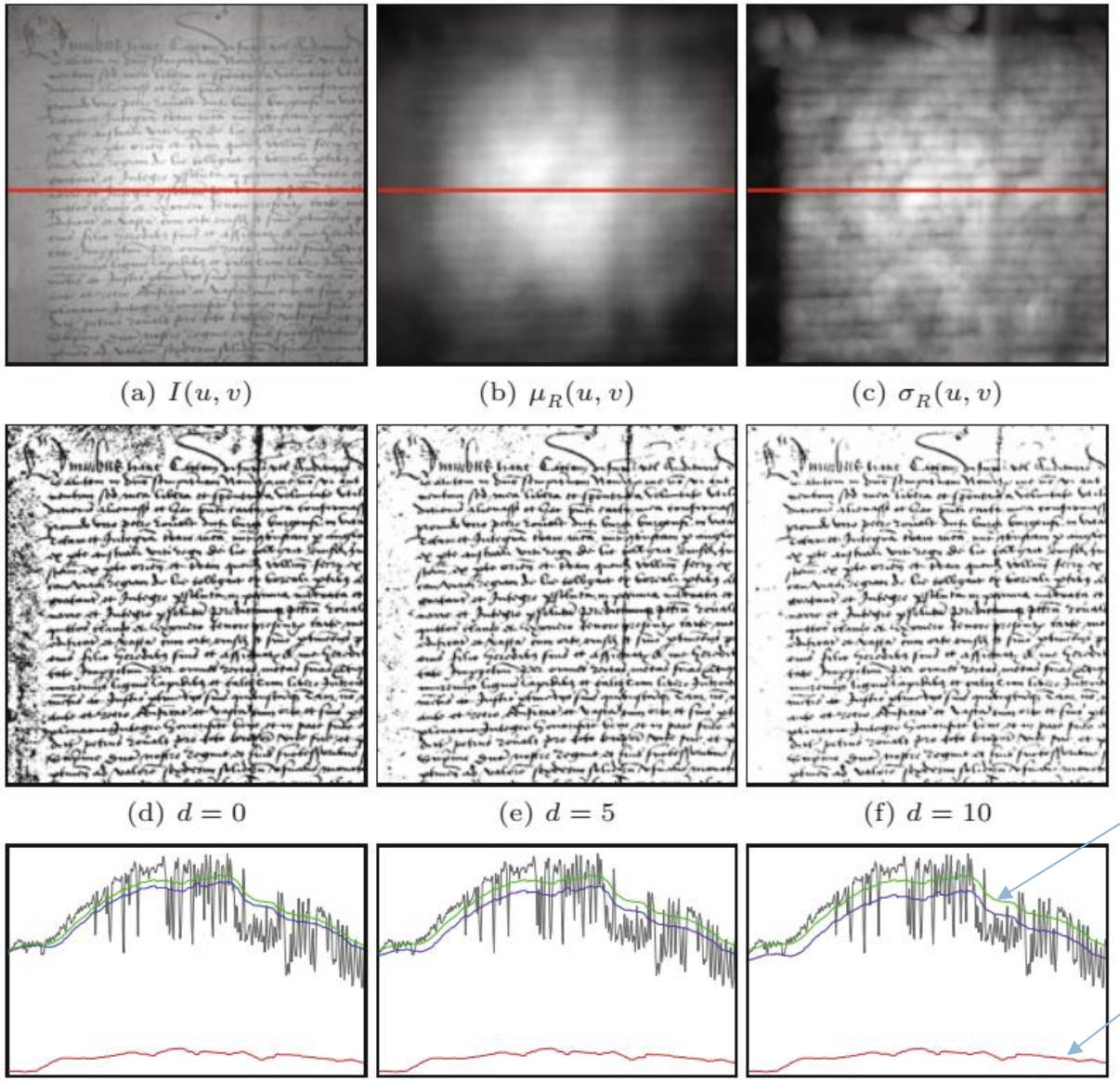
$B \leftarrow B + I^2(i, j)$

$\mu \leftarrow \frac{1}{n} \cdot A$

$\sigma^2 \leftarrow \frac{1}{n} \cdot (B - \frac{1}{n} \cdot A^2)$

return (μ, σ^2)

Adaptive Thresholding - Niblack's Method



(a) $I(u, v)$ (b) $\mu_R(u, v)$ (c) $\sigma_R(u, v)$

(d) $d = 0$ (e) $d = 5$ (f) $d = 10$

(g) $d = 0$ (h) $d = 5$ (i) $d = 10$

Adaptive thresholding using Niblack's method (with $r = 15$, $\kappa = 0.3$). Original image (a), local mean μ_R (b), and standard deviation σ_R (c). The result for $d = 0$ in (d) corresponds to Niblack's original formulation. Increasing the value of d reduces the amount of clutter in regions with low variance (e, f). The curves in (g–i) show the local intensity (gray), mean (green), variance (red), and the actual threshold (blue) along the horizontal line marked in (a–c).

Adaptive Thresholding

- Niblack's Method - modification

for thresholding deteriorated text images, Sauvola and Pietikäinen [207] proposed setting the threshold to

$$Q(u,v) := \begin{cases} \mu_R(u,v) \cdot \left[1 - \kappa \cdot \left(\frac{\sigma_R(u,v)}{\sigma_{\max}} - 1\right)\right] & \text{for } \textit{dark} \text{ BG}, \\ \mu_R(u,v) \cdot \left[1 + \kappa \cdot \left(\frac{\sigma_R(u,v)}{\sigma_{\max}} - 1\right)\right] & \text{for } \textit{bright} \text{ BG}, \end{cases} \tag{11.73}$$

with $\kappa = 0.5$ and $\sigma_{\max} = 128$ (the "dynamic range of the standard deviation" for 8-bit images) as suggested parameter values.

Adaptive Thresholding

Using moving average

- It is based on computing a moving average along scan lines of an image.

$$m(k+1) = \frac{1}{n}\sum_{i=k+2-n}^{k+1} z_i = m(k) + \frac{1}{n}(z_{k+1} - z_{k-n})$$

- z_{k+1} denote the intensity of the point at step k+1.

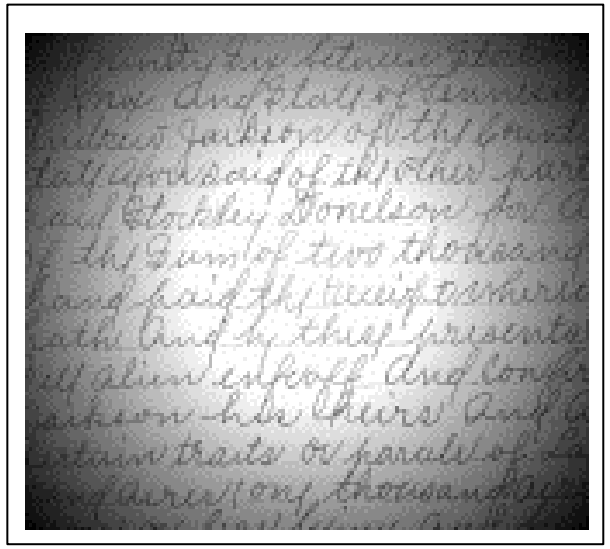
  n denote the number of point used in the average.

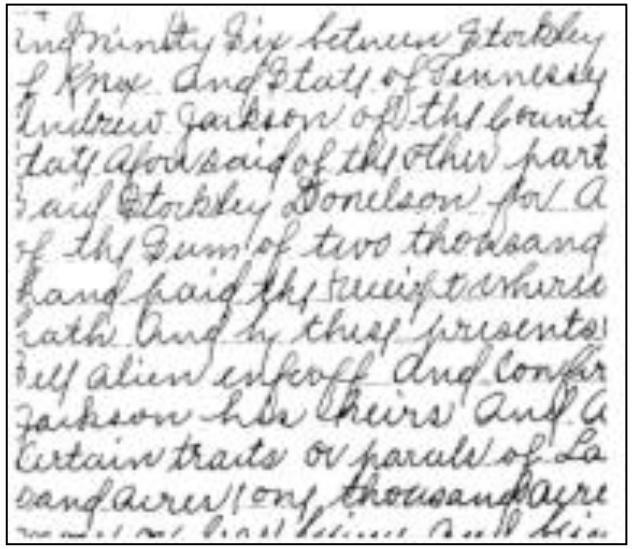
- $m(1) = z_1/n$ is the initial value.

- $T_{xy} = bm_{xy}$ where b is constant and m_{xy} is the moving average at point (x,y)

Adaptive Thresholding

- **moving average**



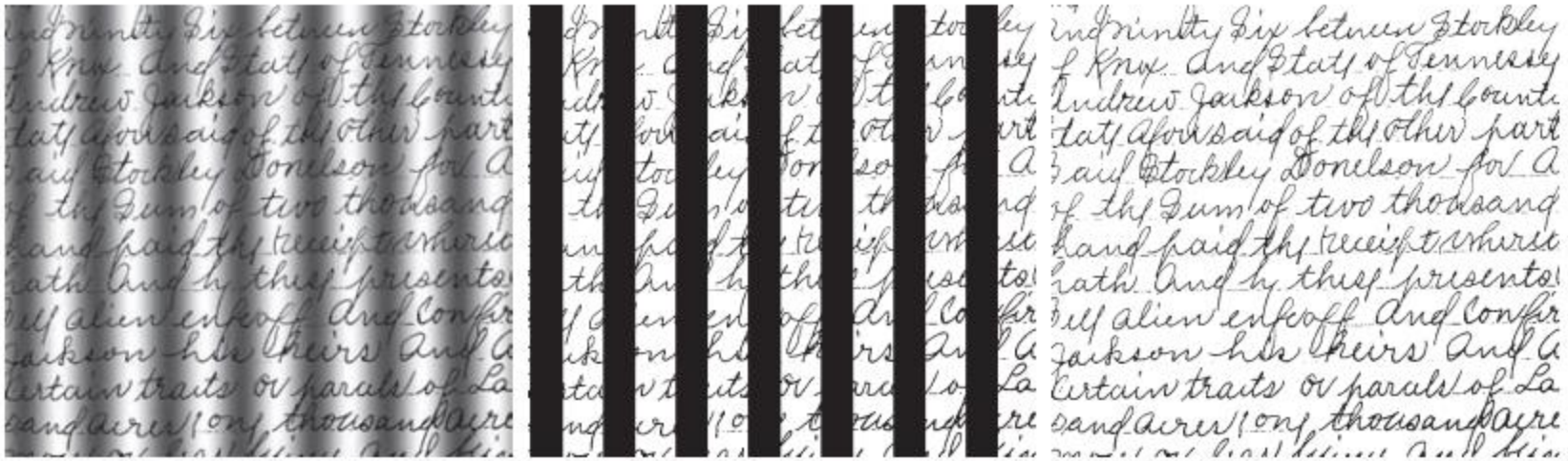


N=20, b=0.5

- Works well, when objects of interest are small (thin) with respect to the image size e.g. typed of hand written text

Adaptive Thresholding

- **moving average**



a b c

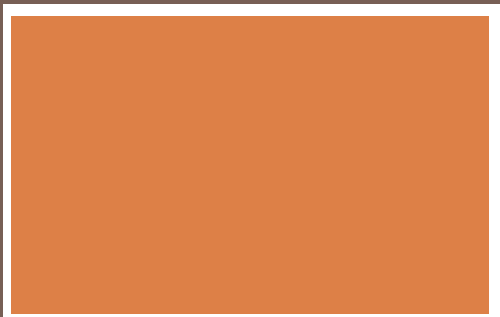
FIGURE 10.45 (a) Text image corrupted by sinusoidal shading. (b) Result of global thresholding using Otsu's method. (c) Result of local thresholding using moving averages..

- Works well, when objects of interest are small (thin) with respect to the image size e.g. typed of hand written text

Summary

- In this lecture we have begun looking at segmentation, and in particular thresholding
- We saw the basic global thresholding algorithm and its shortcomings
- We also saw a simple way to overcome some of these limitations using adaptive thresholding

REGION-BASED SEGMENTATION



Segmentation

- An image domain X must be segmented in N different regions R(1),…,R(N)
- The segmentation rule is a logical predicate of the form P(R)
- Image segmentation with respect to predicate P partitions the image X into subregions R_i, i =1,…,N such that

$X = \bigcup_{i=1,..N} R_i$

$R_i \cap R_j = 0$ for $i \neq j$

$P(R_i)$ = TRUE for i = 1,2,…,N

$P(R_i \cup R_j)$ = FALSE for $i \neq j$

Segmentation

- The segmentation property is a logical predicate of the form P(R,x,t)
- x is a feature vector associated with region R
- t is a set of parameters (usually thresholds). A simple segmentation rule has the form:

$$P(R) : I(r,c) < T \text{ for all } (r,c) \text{ in } R$$

Segmentation

- In the case of color images the feature vector x can be three RGB image components (R(r,c),G(r,c),B(r,c))
- A simple segmentation rule may have the form:

P(R) : (R(r,c) <T(R)) && (G(r,c)<T(G))&&

(B(r,c) < T(B))

Region Growing (Merge)

- A simple approach to image segmentation is to start from some pixels (**seeds**) representing distinct image regions and to grow them, until they cover the entire image
- For region growing we need a **rule** describing a growth mechanism and a rule checking the homogeneity of the regions after each growth step

Region Growing

- The growth mechanism – at each stage k and for each region $R_i(k)$, i = 1,…,N,
  - Check if there are unclassified pixels in the 8-neighbourhood of each pixel of the region border
- Before assigning such a pixel x to a region Ri(k),we check if the region homogeneity:

  $P(R_i(k) \cup \{x\})$ = TRUE , is valid
- Selection of similarity criteria: color, descriptors (gray level + moments / texture)

Region Growing

- Choosing a **seed** pixel:
  - Preferably provided by the user. A good seed can be drawn from the peak of the object histogram
- Minimum area thresholding:
  - No region will be smaller than this threshold in the segmented image
- Similarity threshold:
  - If a pixel and a region (or region A and region B) are considered similar enough a union is made, Otherwise a new region is formed
  - **High threshold value** – easy for new pixels to get accepted to the region
  - **Low threshold value** – hard for new pixels to get accepted

Region Growing Predicate

- **Similarity check example::**
- At each iteration, and for each region R_i Compute arithmetic mean m_i and standard deviation σ_i having $n = |R_i|$ pixels:

$$m_i = \frac{1}{n} \sum_{(x,y) \in R_i} f(x,y) \qquad \sigma_i = \sqrt{\frac{1}{n-1} \sum_{(x,y) \in R_i} (f(x,y) - m_i)^2}$$

- If the regions adhere the similarity (homogeneity) condition than we can unite them
- The predicate P can be used to decide if the merging of the two regions R_i, R_j is allowed

$$P: \quad |m_i - m_j| < k \times \min\{\sigma_i, \sigma_j\}$$

Region growing

- Other homogeneity criteria (with more features) can be considered

average intensity **variance**

color **texture**

Motion

shape **size**

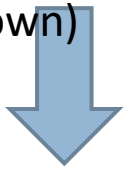
etc…

Split

- The opposite approach to region growing is region splitting.
- It is a top-down approach
- Briefly-

Split & merge

(top down)



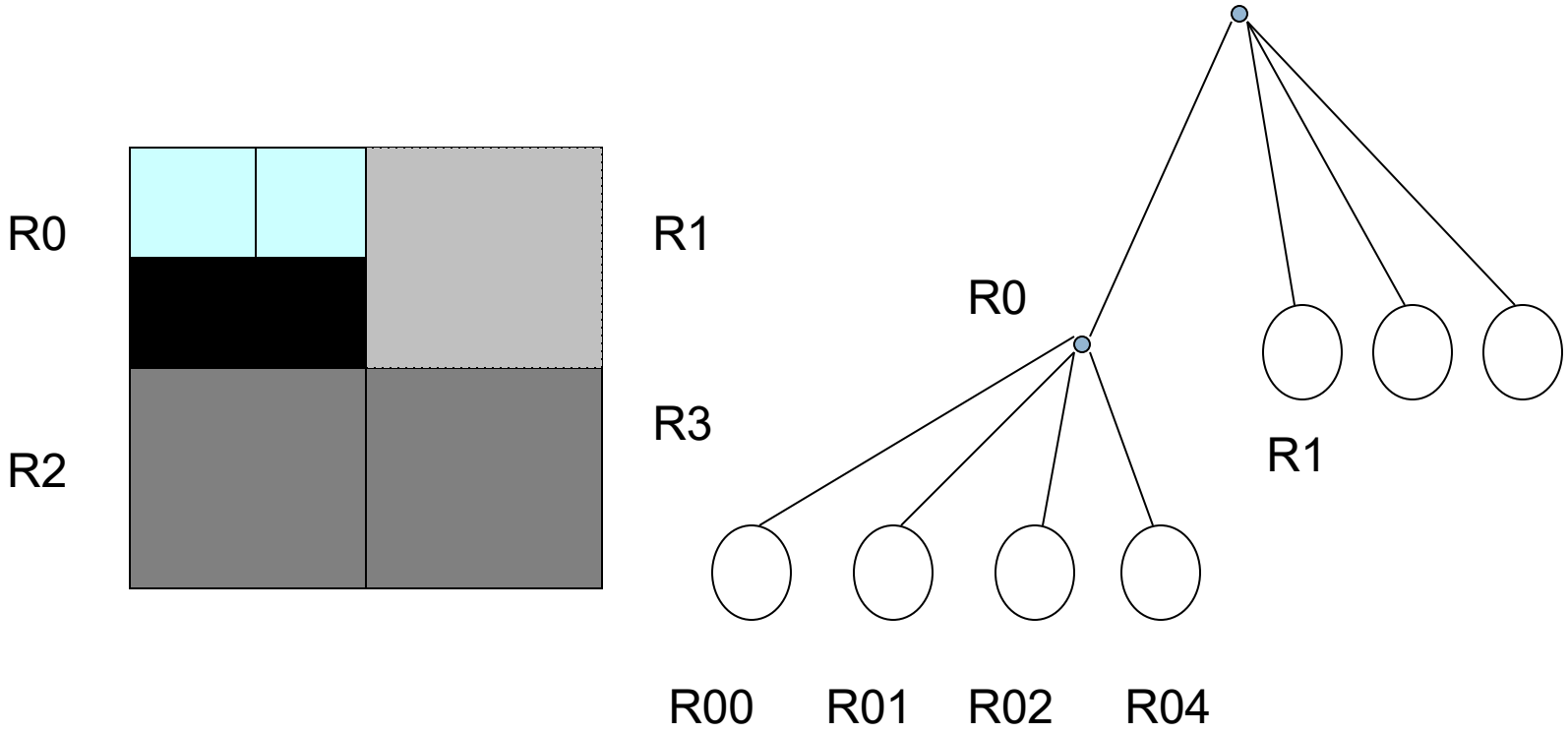
1. it starts with the assumption that the entire image is homogeneous
2. If this is not true (by the homogeneity criterion, P), the image is split into four sub images
3. This splitting procedure is repeated recursively until we split the image into homogeneous regions

Split

- If the original image is square N x N, having dimensions that are powers of 2($N = 2^n$):
- All regions produced but the splitting algorithm are squares having dimensions M x M , where M is a power of 2 as well.
- Since the procedure is recursive, it produces an image representation that can be described by a tree whose nodes have four sons each
- Such a tree is called a Quadtree.

Split

Quadtree



Split

- **Disadvantage**
  - They create regions that may be adjacent and homogeneous, but not merged.
- **Improvement** - Split and Merge
  - An iterative algorithm that includes both splitting and merging at each iteration:

Split / Merge

- Briefly-

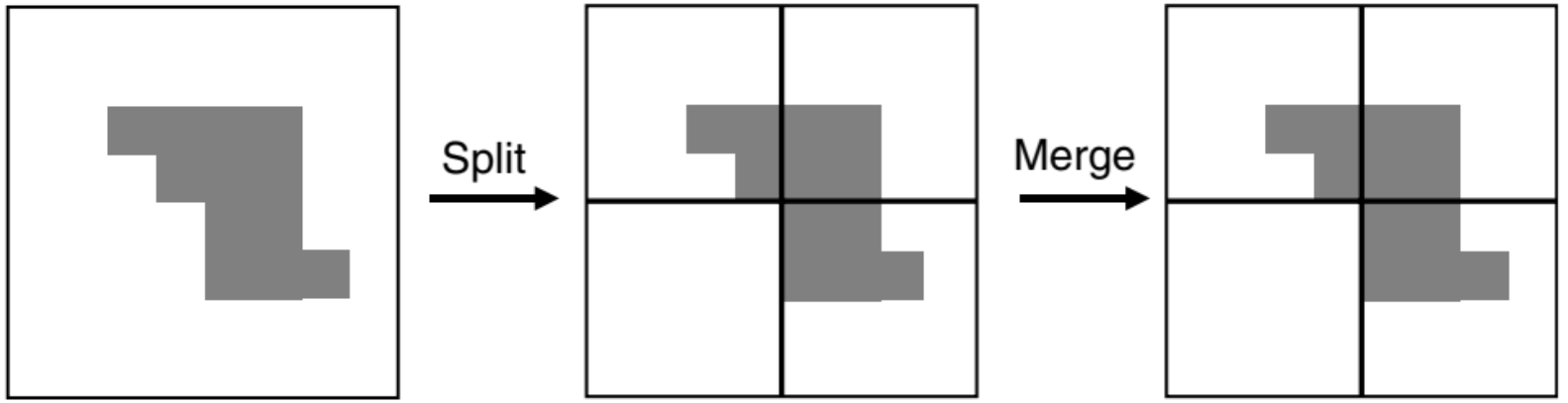
1. it starts with the assumption that the entire image is homogeneous
2. If this is not true (by the homogeneity criterion), the image is split into four sub images
3. This splitting procedure is repeated recursively until we split the image into homogeneous regions
4. **Merging phase:** If 2 adjacent regions are homogenous, they are merged
5. Repeat **step 4** until no further merging is possible

Split / Merge

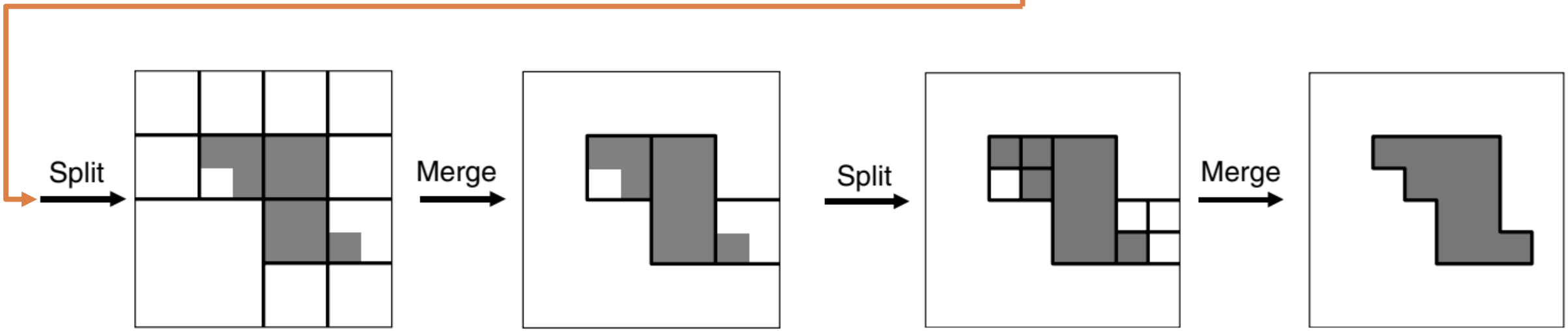
1. Split any region R_i into 4 disjoint quadrants for which $P(R_i)$ = FALSE
2. Merge any adjacent region R_i and R_j for which $P(R_i \cup R_j)$ = TRUE
3. Repeat step 1 and 2 until no further splitting or merging is possible.

- The split and merge algorithm produces more compact regions than the pure splitting algorithm

Split / Merge



Iteration 1



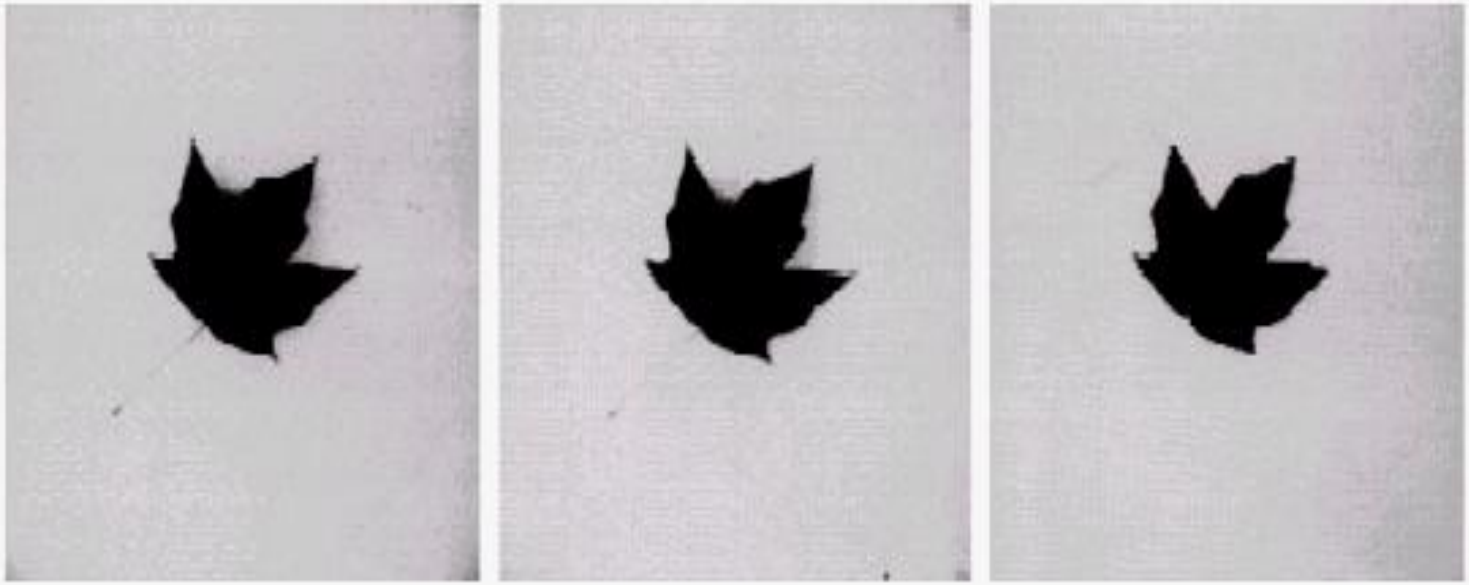
Iteration 2

Iteration 3

Split / Merge

a b c

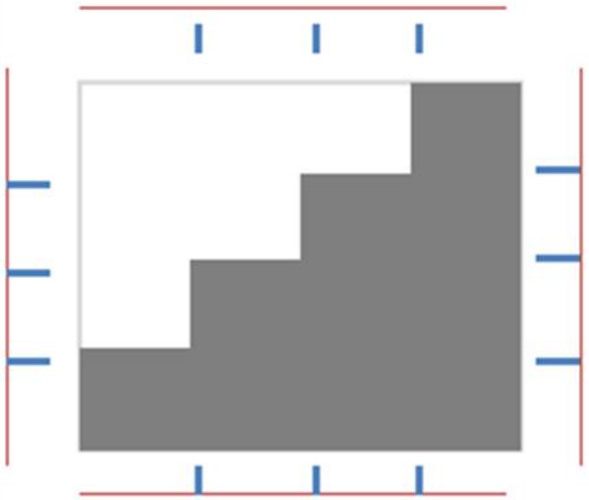
FIGURE 10.43
(a) Original image. (b) Result of split and merge procedure. (c) Result of thresholding (a).



$P(R_i)$ = TRUE if at least 80% of the pixels in R_i have the property $|z_j - m_i| \leq 2\sigma_i$, where

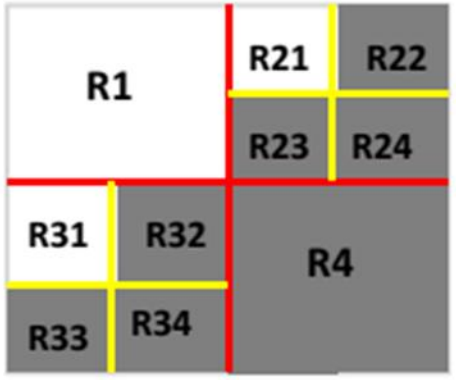
z_j is the gray level of the j^{th} pixel in R_i
m_i is the mean gray level of that region
σ_i is the standard deviation of the gray levels in R_i

Split / Merge



Figure(a): Image

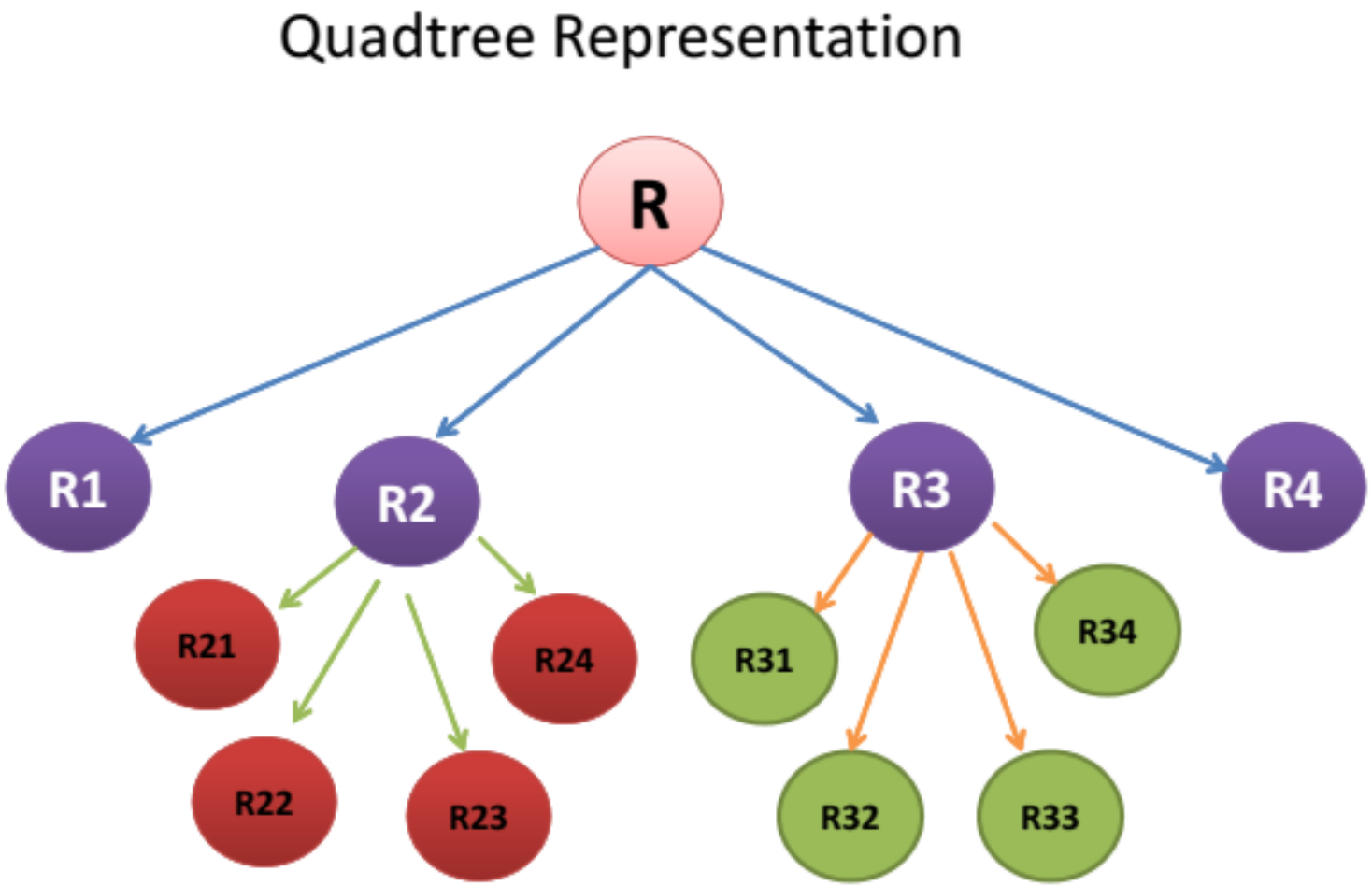
Figure(b): step-1



Figure(c): step-2

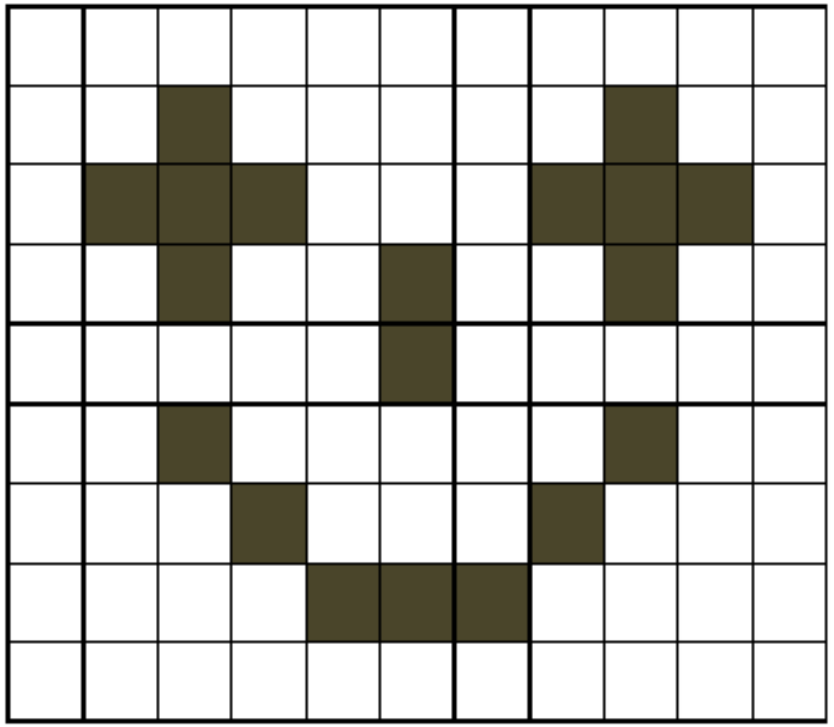
Figure(d): step-3

Split / Merge


Quadtree Representation
R
R1
R2
R3
R4
R21
R22
R23
R24
R31
R32
R33
R34

Split / Merge

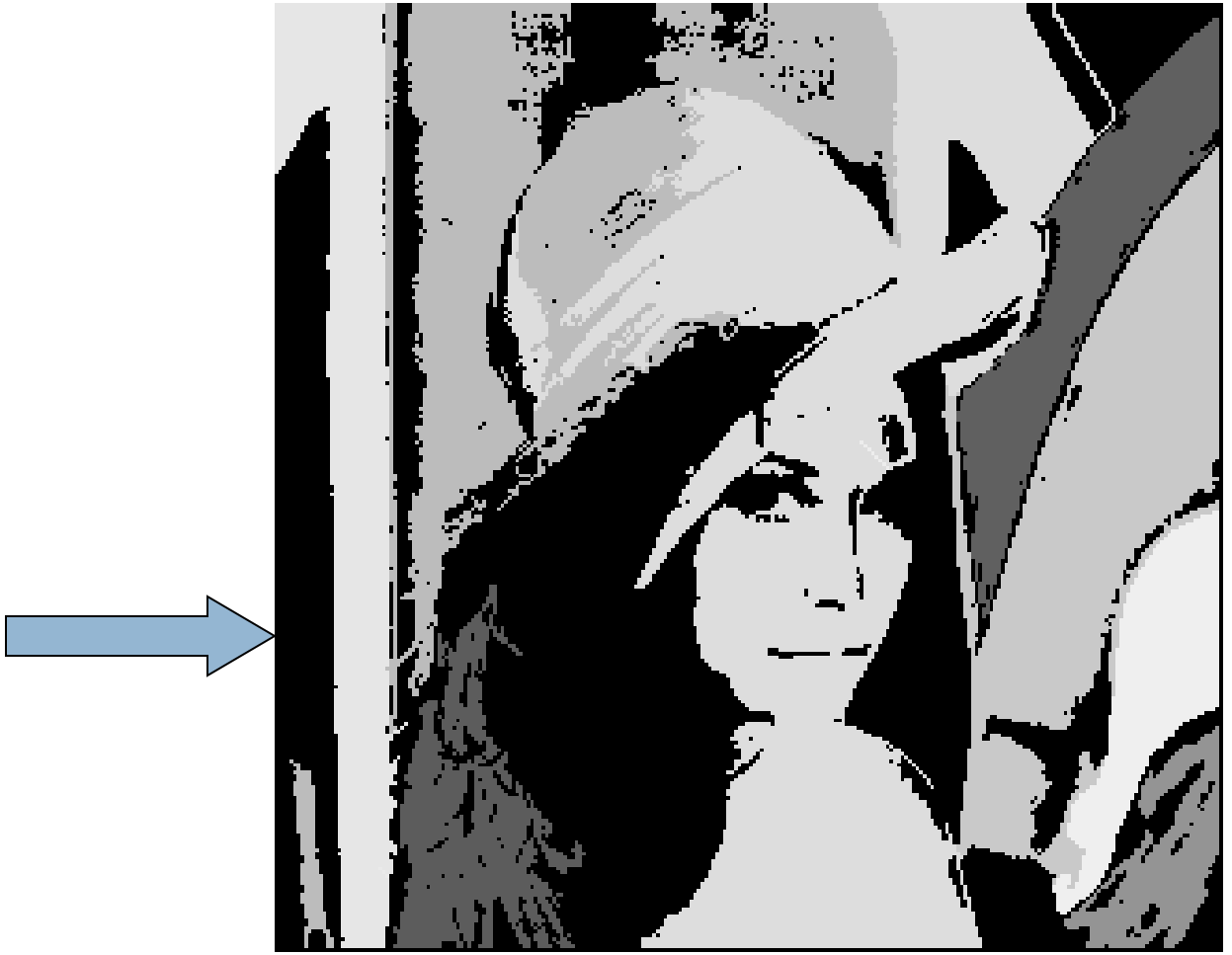
- Try this



Applications

- 3D – Imaging : A basic task in 3-D image processing is the segmentation of an image which classifies voxels/pixels into objects or groups.
- 3-D image segmentation makes it possible to create 3-D rendering for multiple objects and perform quantitative analysis for the size, density and other parameters of detected objects.
- Several applications in the field of Medicine like magnetic resonance imaging (MRI).

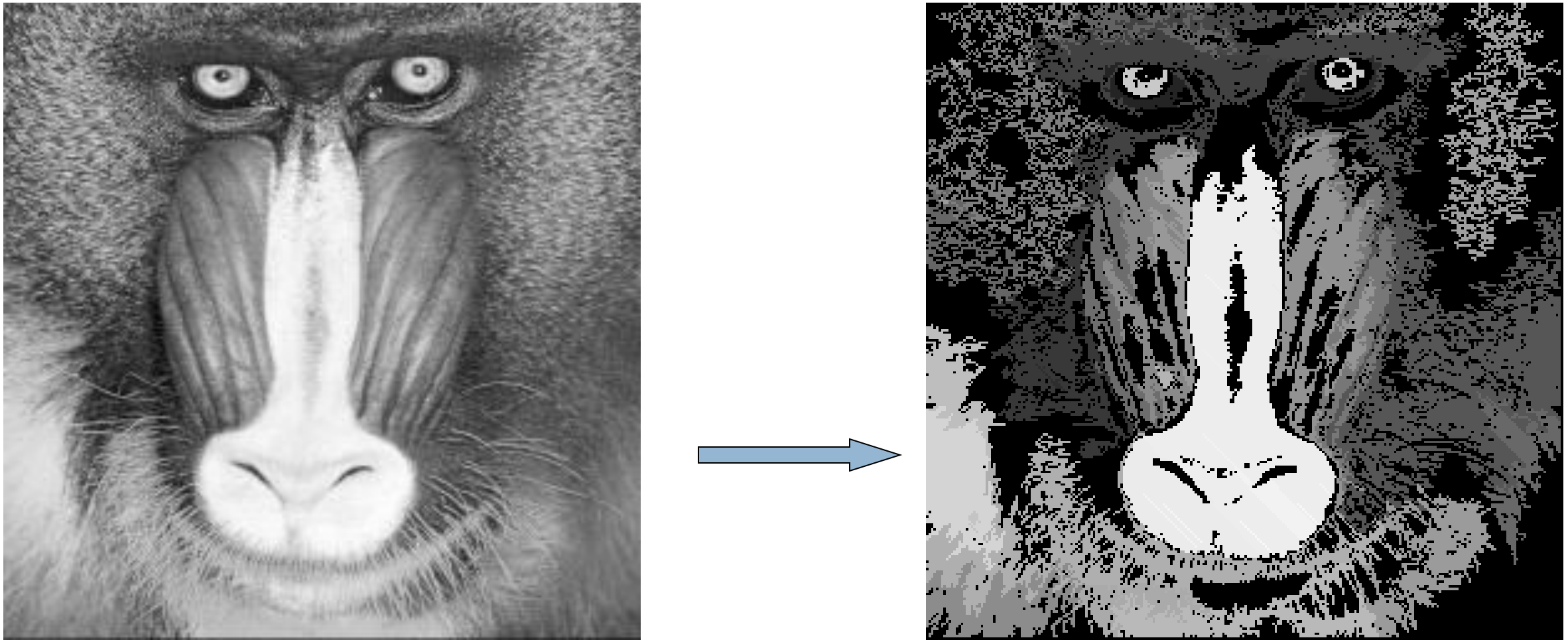
Results – Region grow



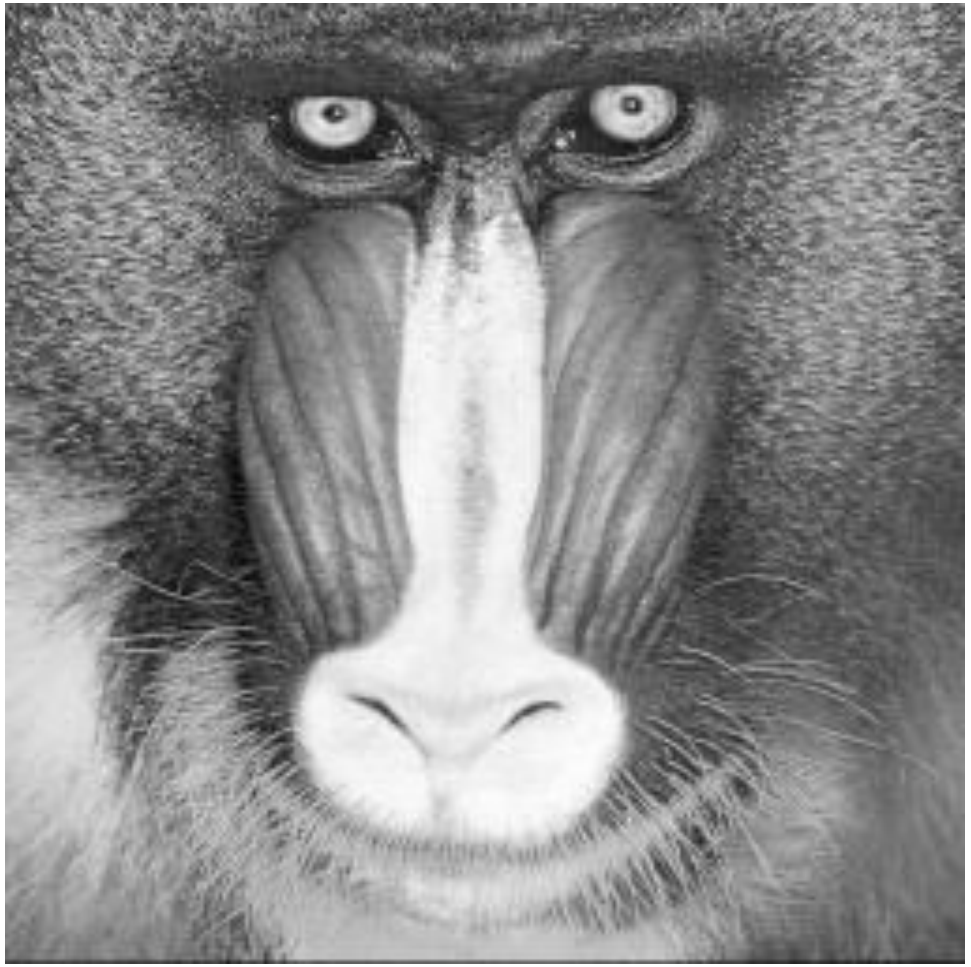
Results – Region Split

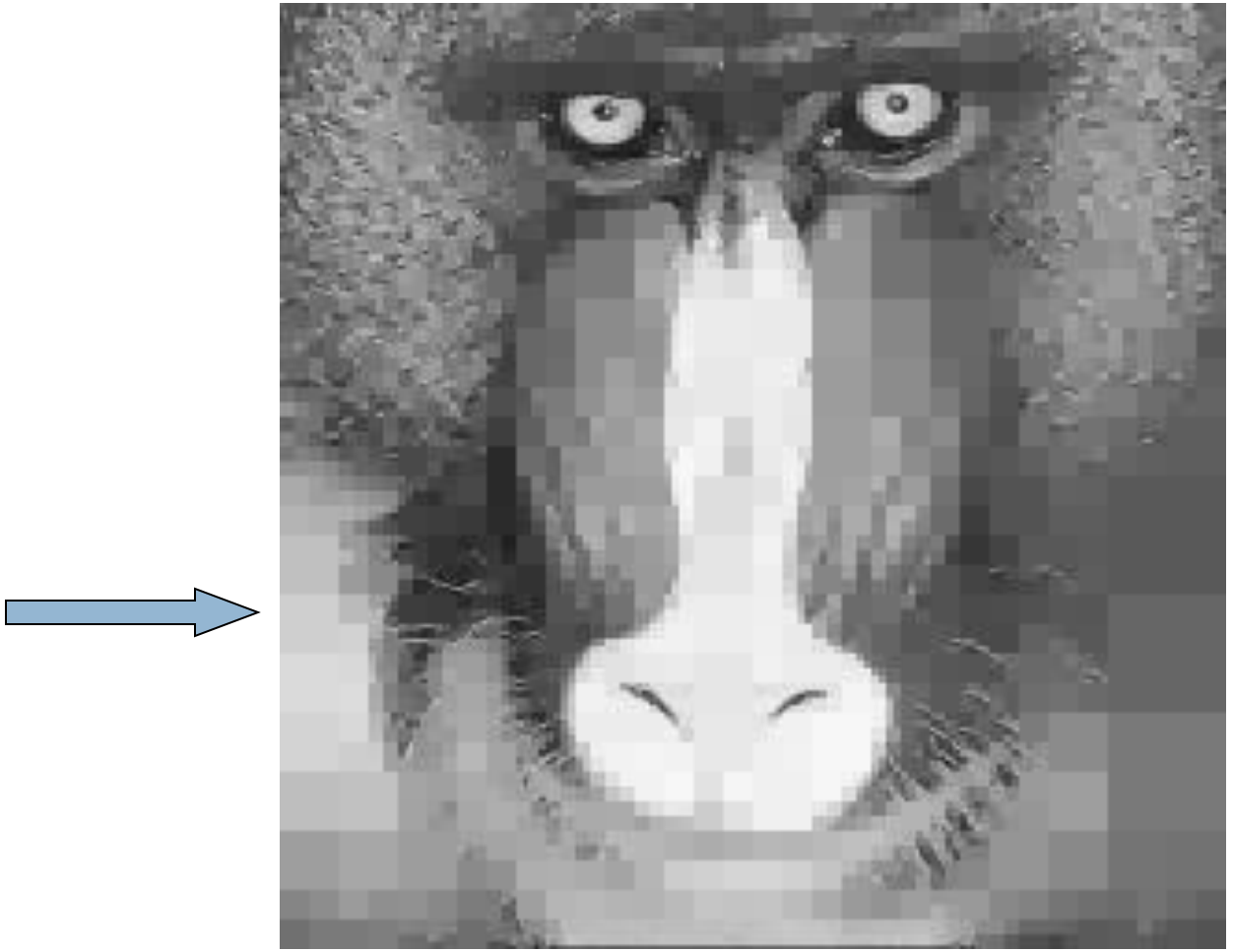
Results – Region Split and Merge

Results – Region growing

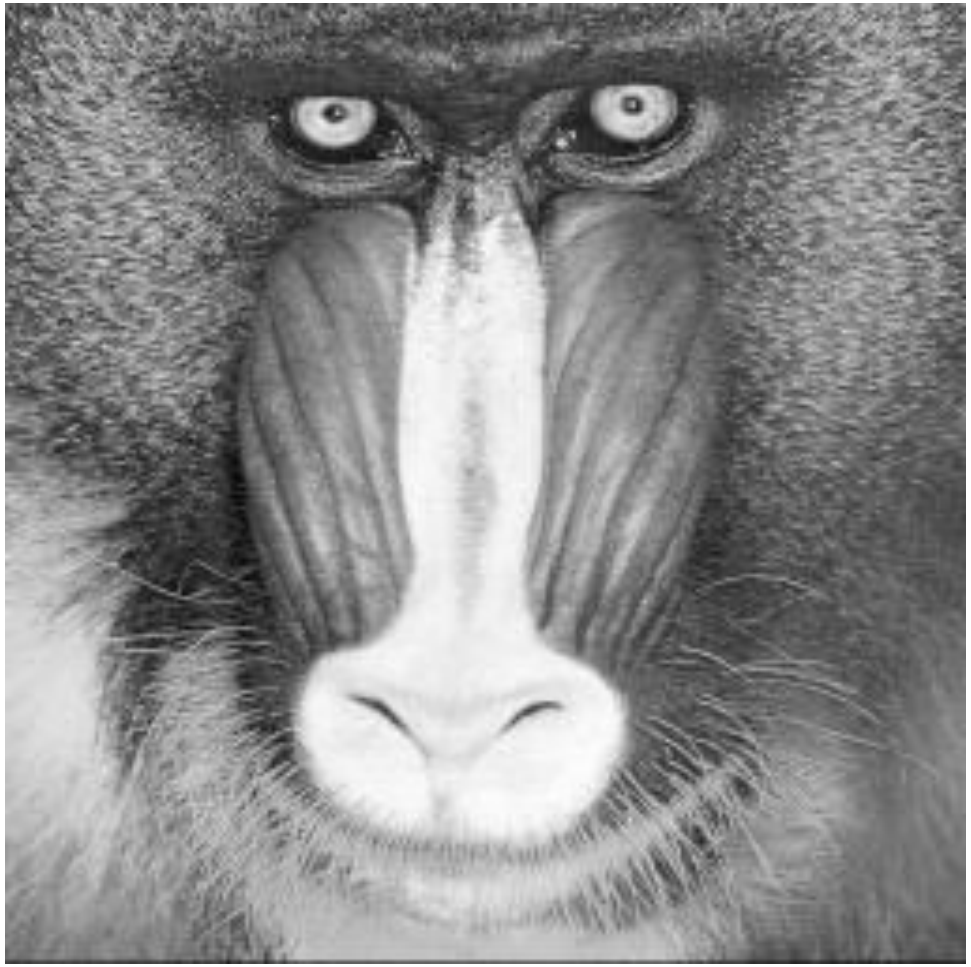


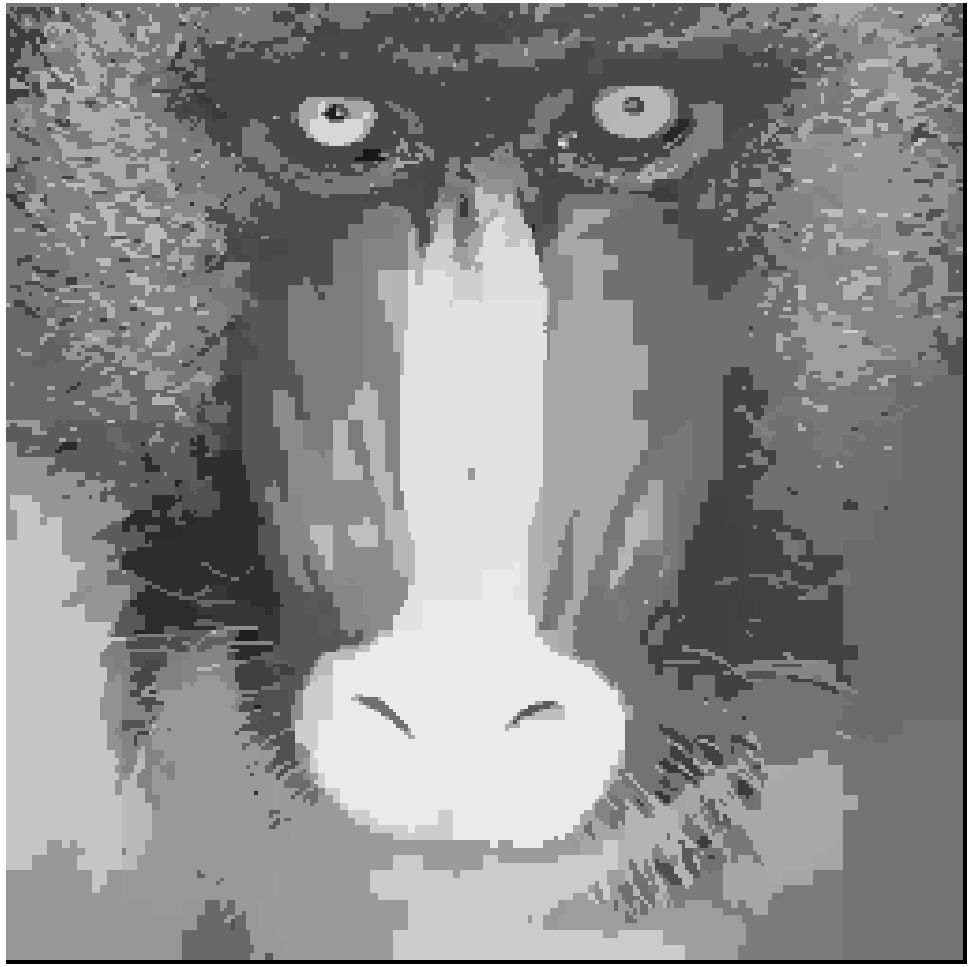
Results – Region Split





Results – Region Split and Merge





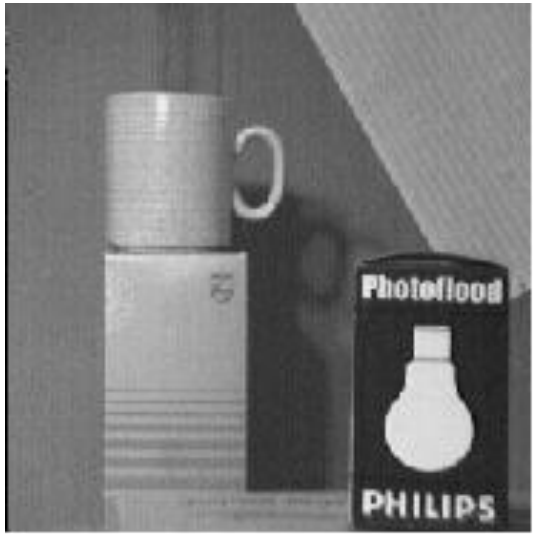
Problems with regional segmentation

- There are problems with regional segmentation of any form:
  - ``Meaningful'' regions may not be uniform: surface properties of a solid body will vary in brightness or colour dependent on the existence of slowly varying gradients due to lighting conditions.
    - Lighting effects or curvature affect the appearance, e.g. a sphere illuminated by a point light source may have intensities varying from pure white to black, yet is a single surface.
  - It is very unusual in practice for an image to be composed of uniform regions of similar intensity, or colour, or texture etc.
  - Regional segmentation works best with binary data as the limited range of values lead to more uniform regions.

Problems with regional segmentation

- In practice, boundary segmentation is much more widely applied than regional segmentation for several reasons
  - Algorithms are usually less complex: they tend to use local properties and software and hardware implementations are readily available.
  - Humans may use edge detection: there is evidence of links between edge detection and early human visual processing, which lead to the observation that contoured images are more easily identified than regional images, particularly when degraded in some form.
  - Edges are often more useful in matching: as finding regions or edges is often preliminary to identifying objects, it is important that edges have an easier model description (as lines).

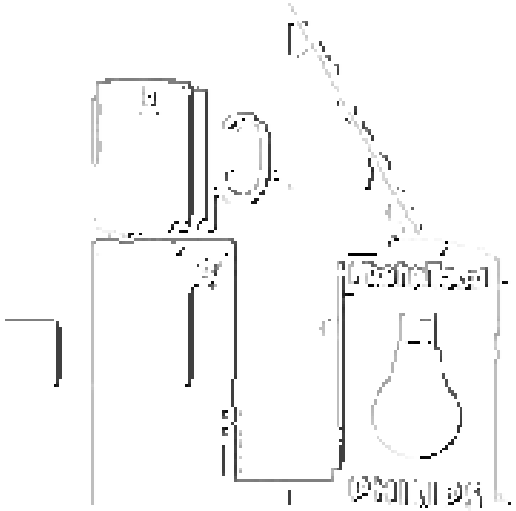
Problems with regional segmentation



Original image

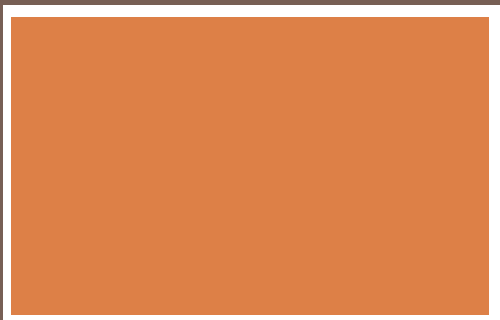
After quad splitting

After merging



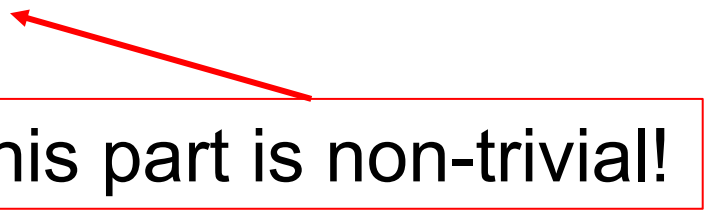
Edge delineation of regions

EDGE BASED SEGMENTATION



Edge-based segmentation

- Edge-based methods center around contour detection
- General workflow
  1. Detect edges, i.e., mark each pixel as ”edge” or ”not edge”.
  1. Divide the image into regions, based on the detected edges. (Edge linking, Hough transform)



- Weakness in connecting broken contour lines make them prone to failure in the presence of blurring.

Detection Of Discontinuities

- There are three basic types of grey level discontinuities that we tend to look for in digital images:
  - Points
  - Lines
  - Edges
- We typically find discontinuities using masks and correlation

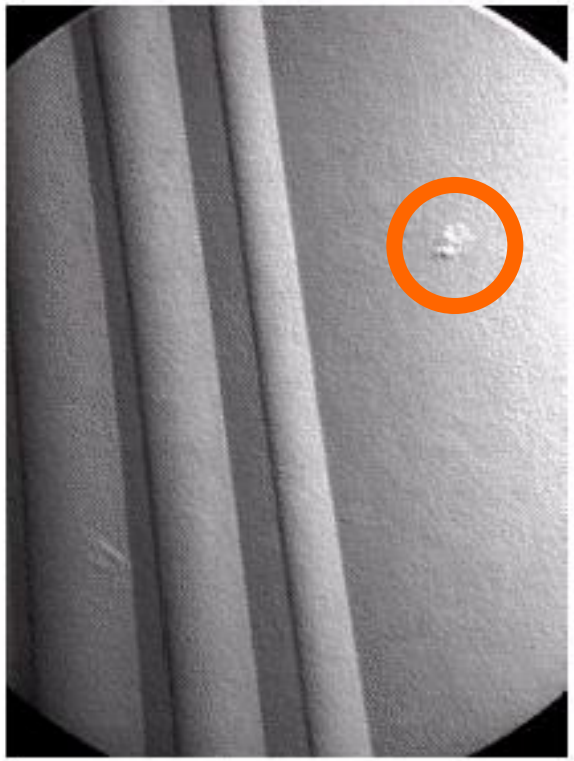
Point Detection

- Point detection can be achieved simply using the mask below:

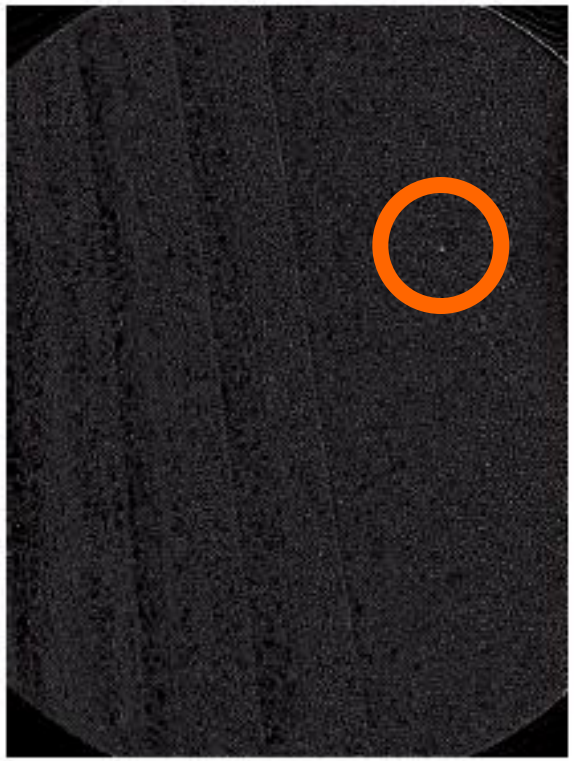
−1	−1	−1
−1	8	−1
−1	−1	−1

- Points are detected at those pixels in the subsequent filtered image that are above a set threshold

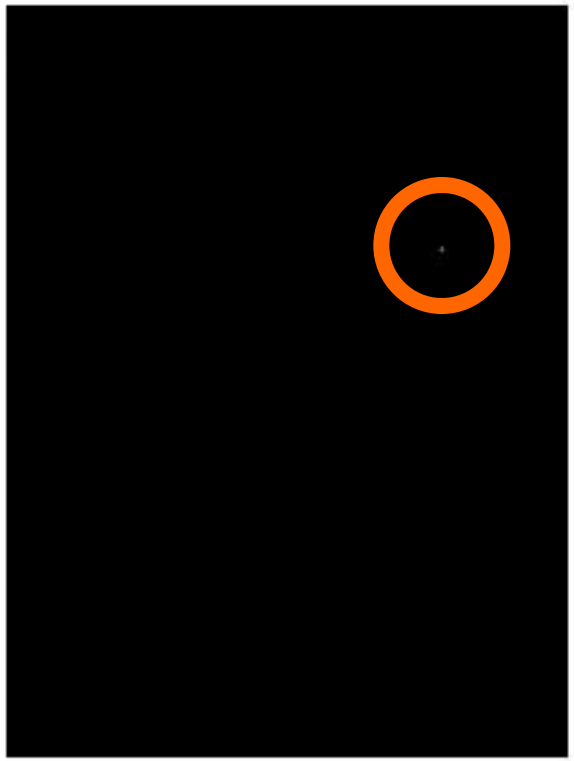
Point Detection (cont…)



X-ray image of
a turbine blade



Result of point
detection



Result of
thresholding

Line Detection

- The next level of complexity is to try to detect lines
- The masks below will extract lines that are one pixel thick and running in a particular direction

−1	−1	−1
2	2	2
−1	−1	−1

Horizontal

−1	−1	2
−1	2	−1
2	−1	−1

+45°

−1	2	−1
−1	2	−1
−1	2	−1

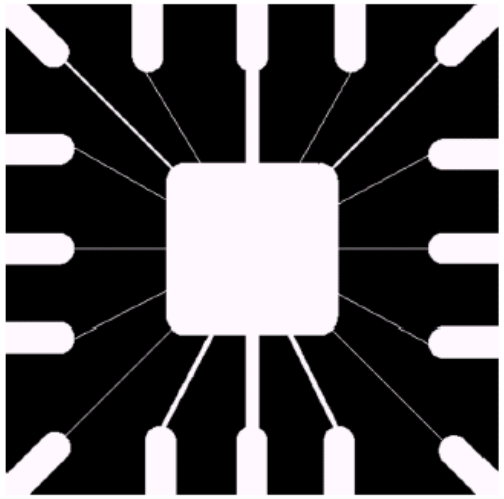
Vertical

2	−1	−1
−1	2	−1
−1	−1	2

−45°

Line Detection (cont…)

Binary image of a wire bond mask

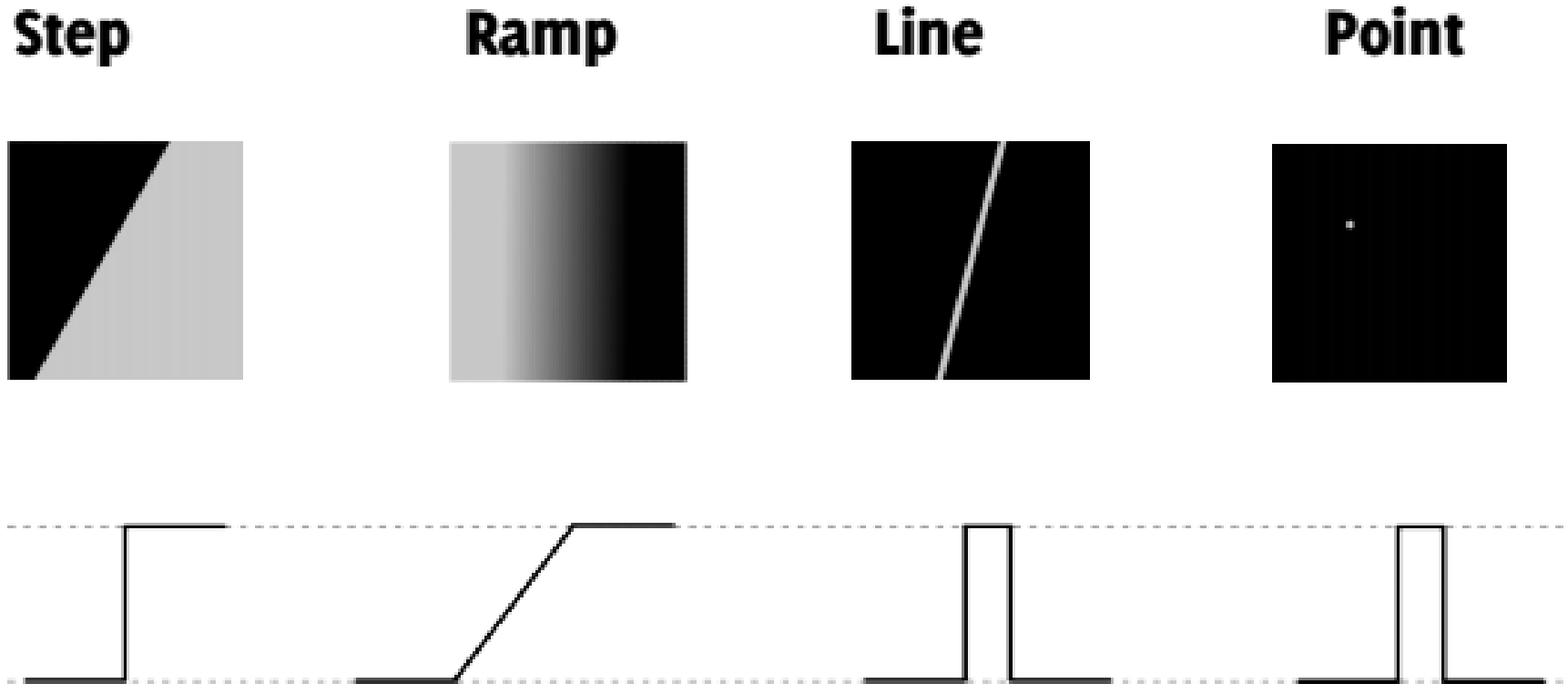


After processing with -45° line detector

Result of thresholding filtering result

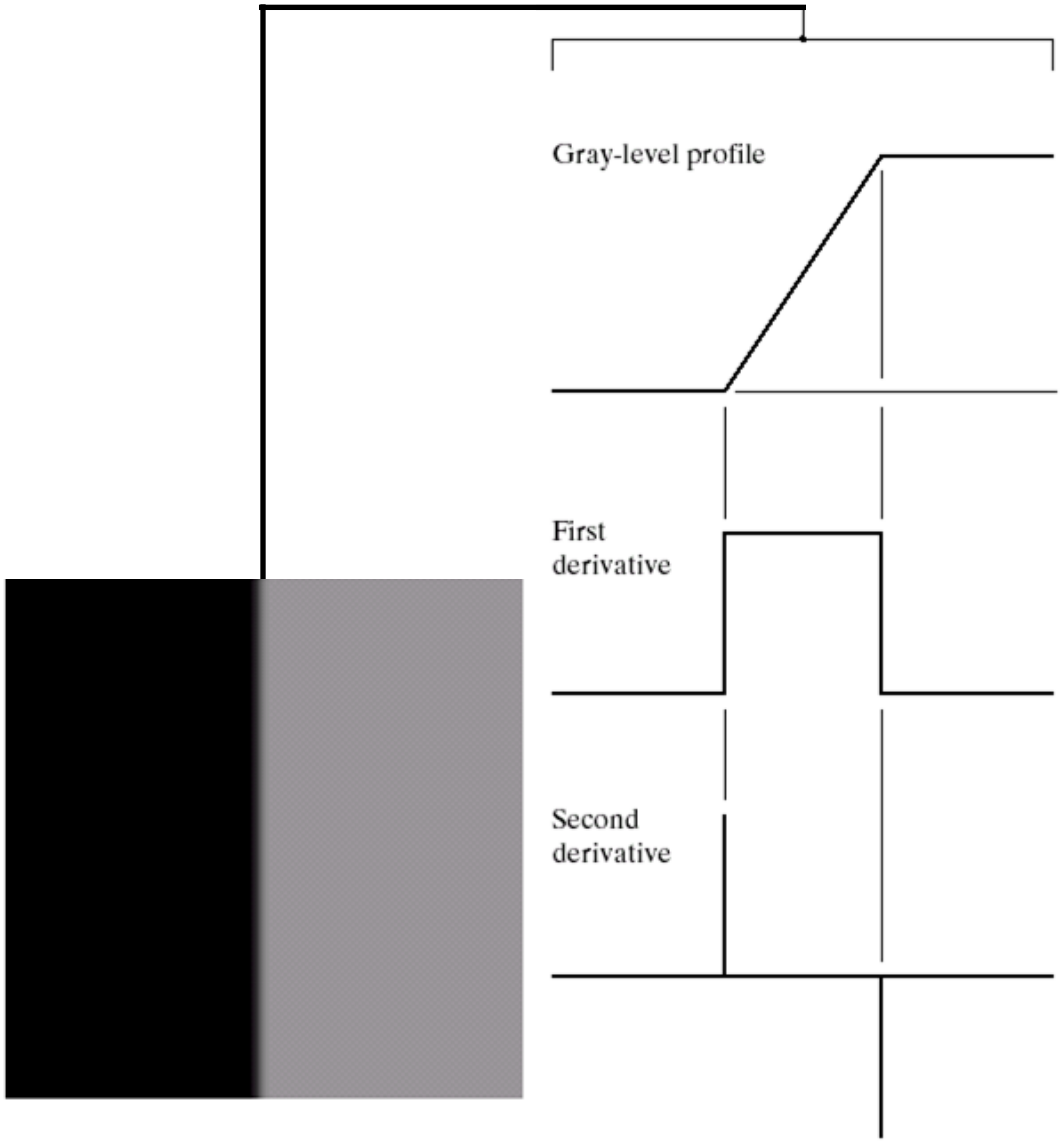
Edge Detection

- An edge is a set of connected pixels that lie on the boundary between two regions
- Edge models

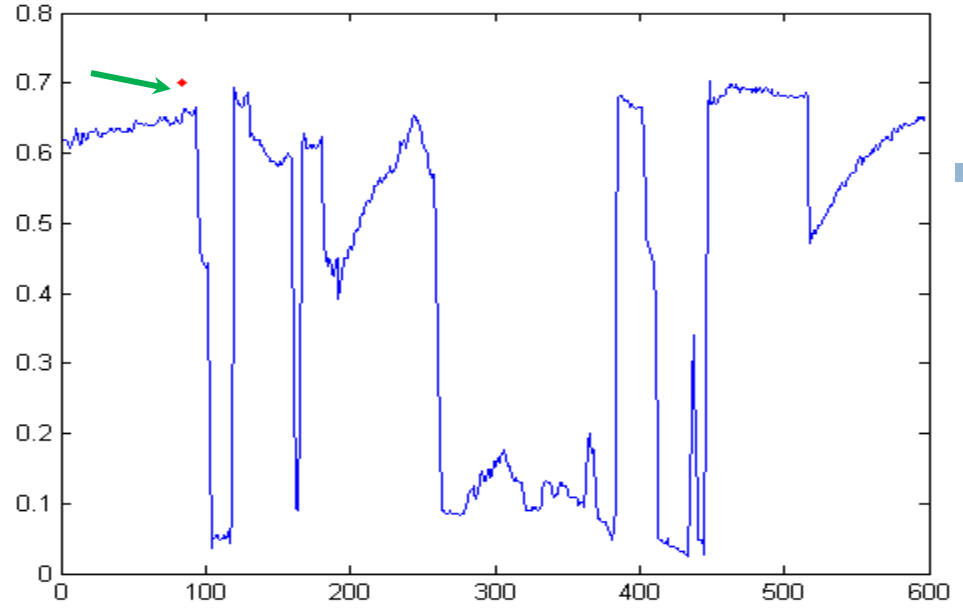


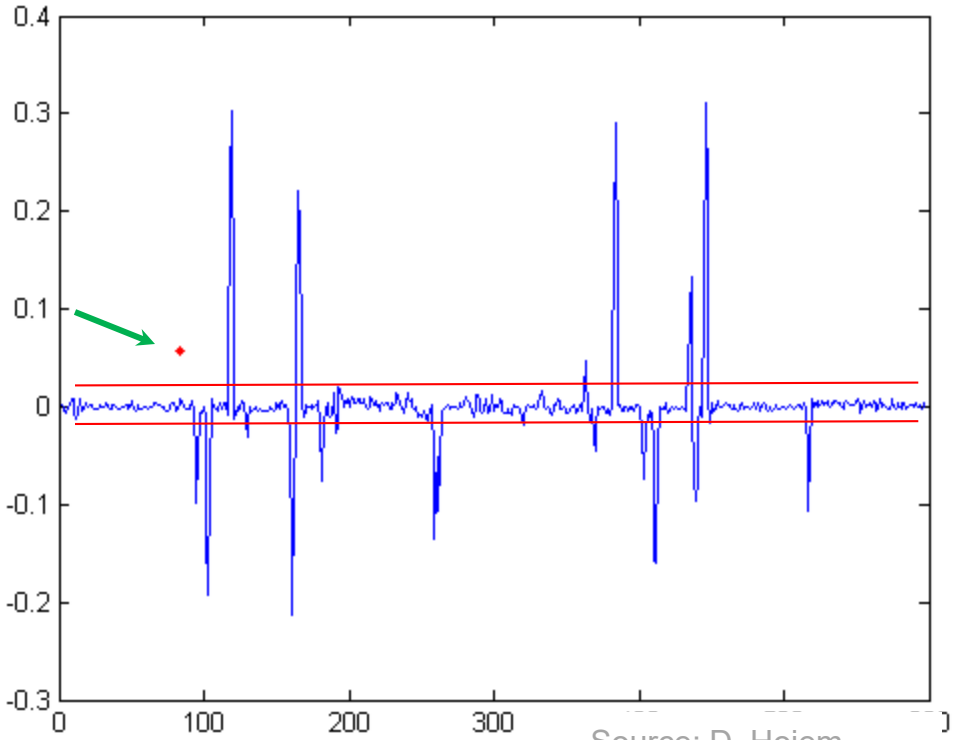
Edges & Derivatives

- We have already spoken about how derivatives are used to find discontinuities
- 1st derivative tells us where an edge is
- 2nd derivative can be used to show edge side



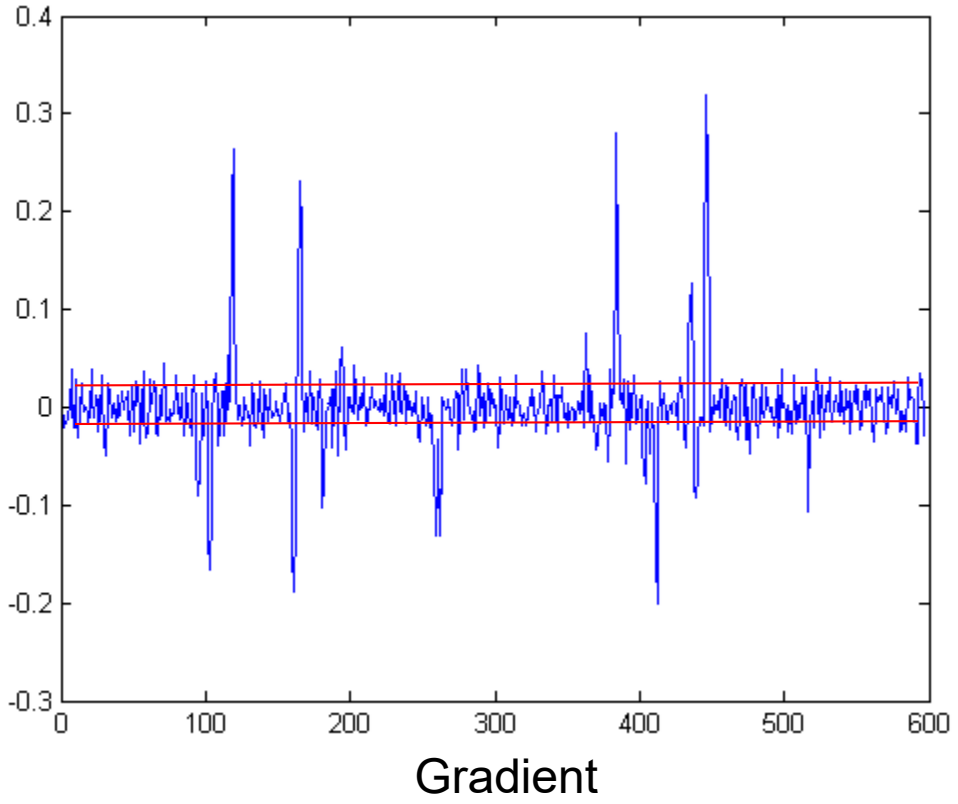
Intensity profile





Source: D. Hoiem

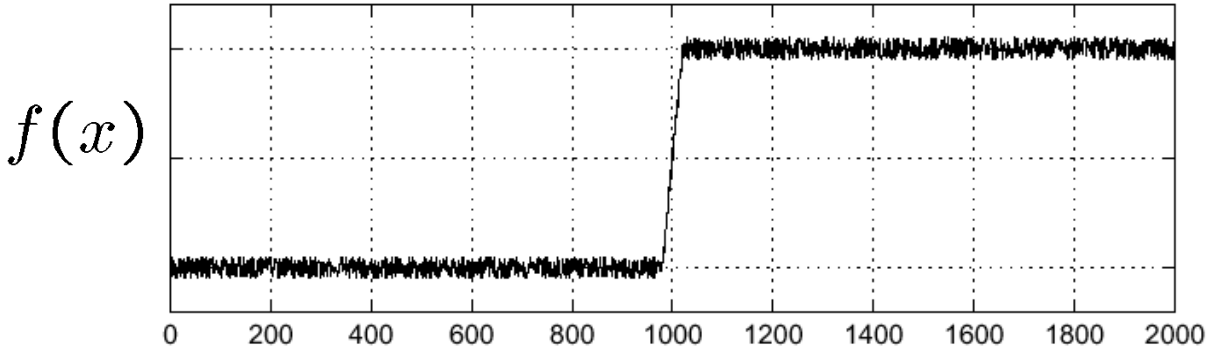
With a little Gaussian noise

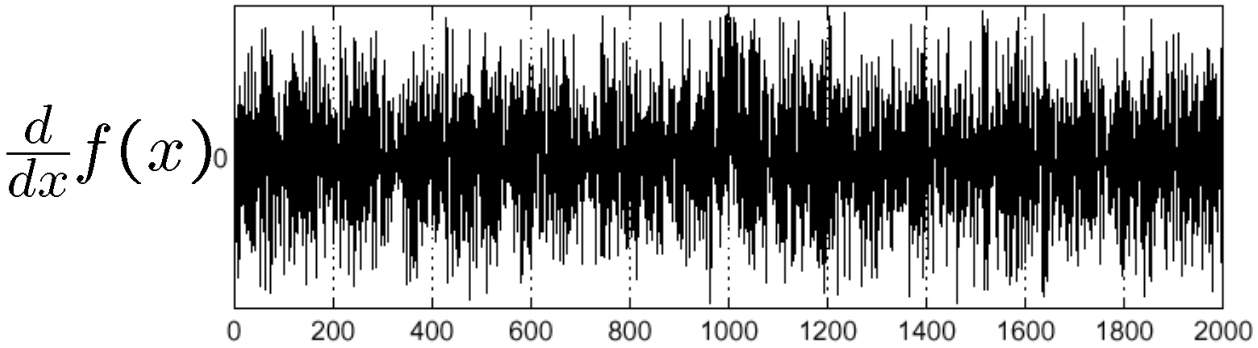


Source: D. Hoiem

Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal



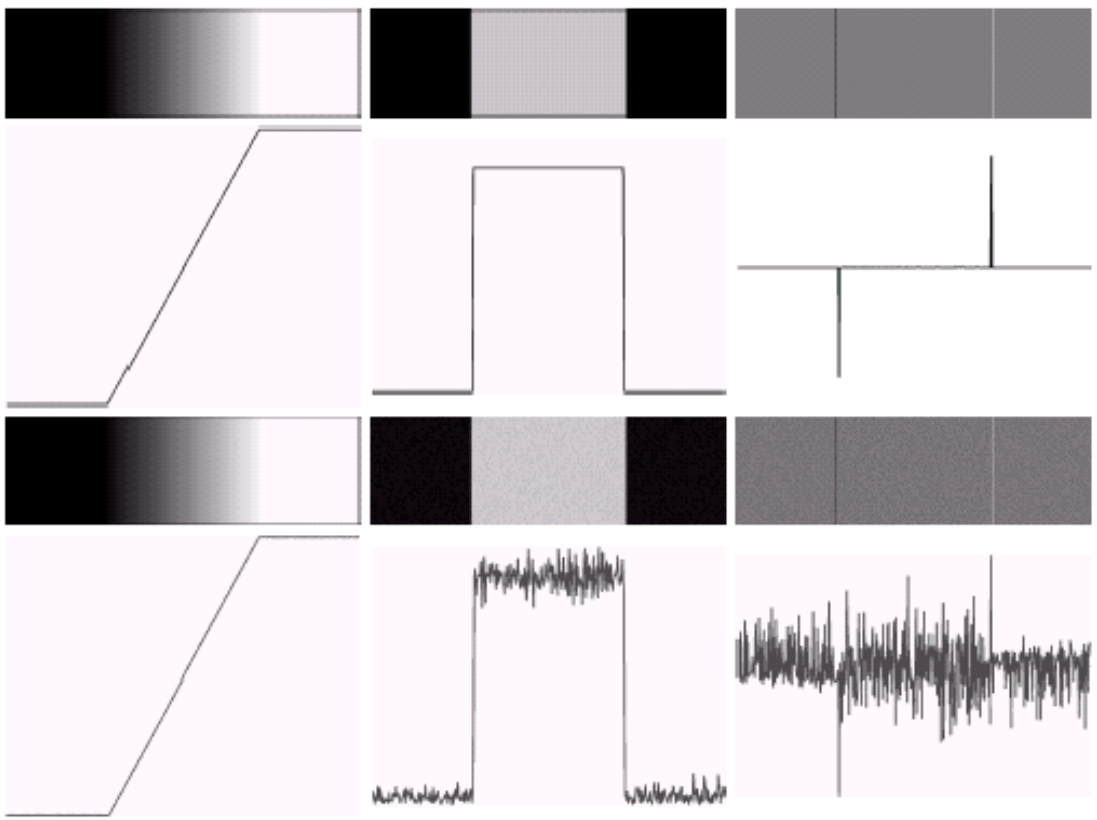


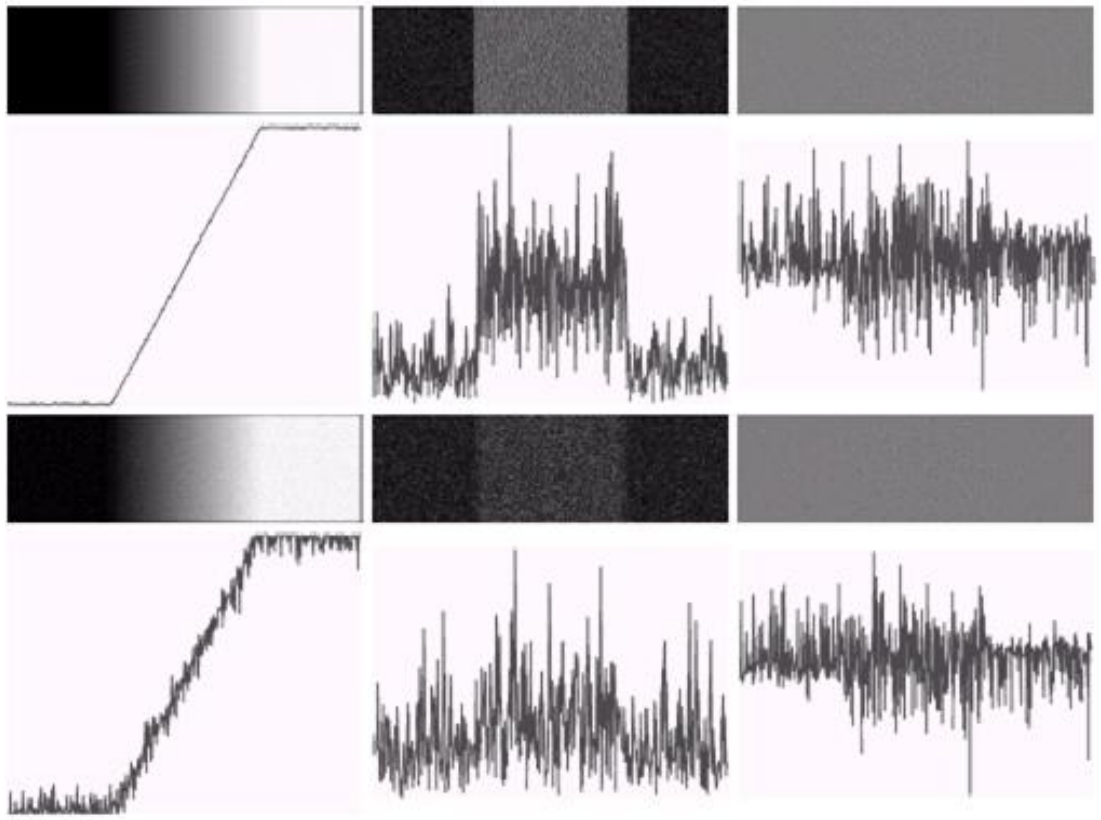
Where is the edge?

Source: S. Seitz

Derivatives & Noise

- Derivative based edge detectors are extremely sensitive to noise
- We need to keep this in mind



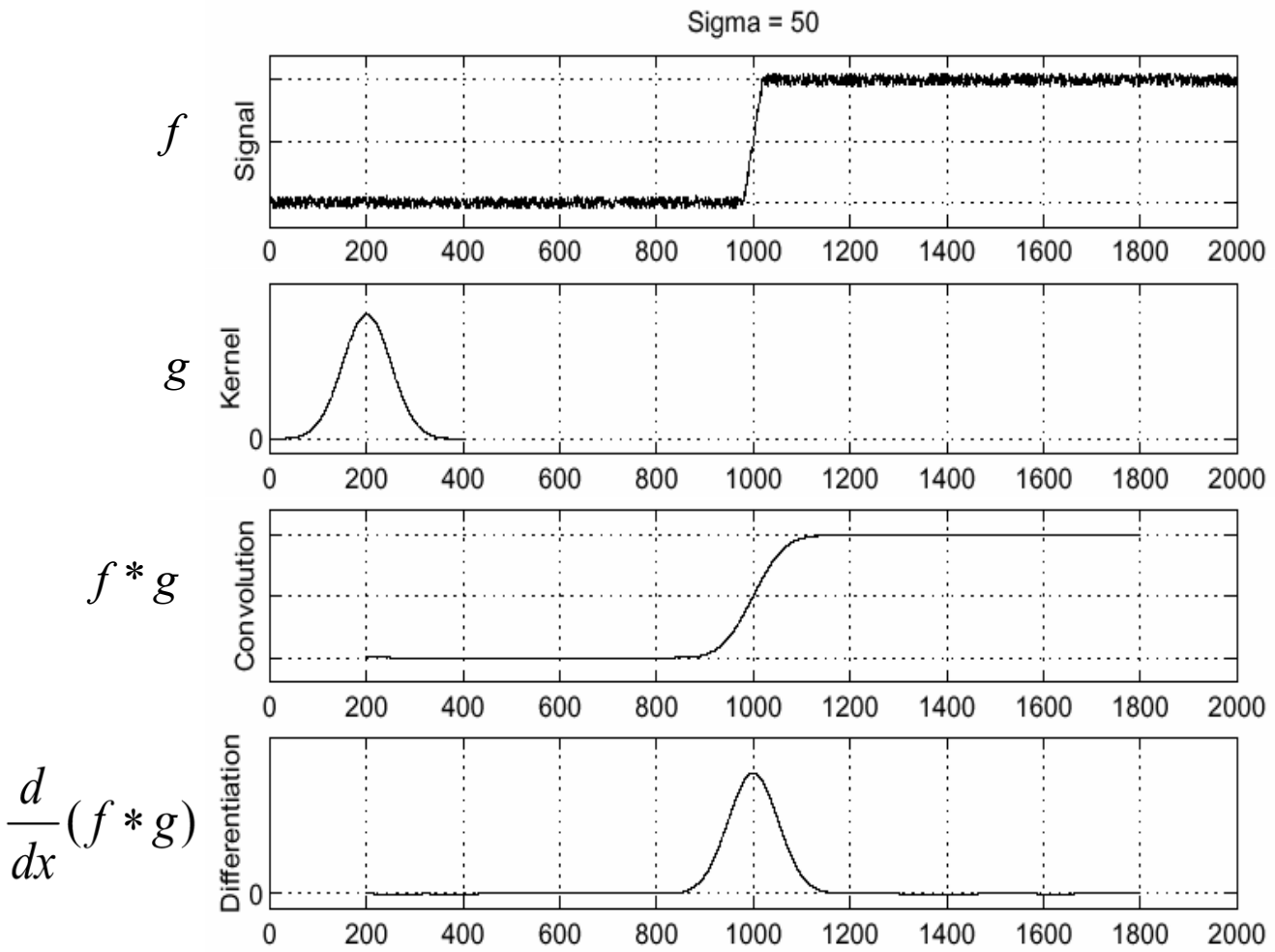


Effects of noise

- Difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the stronger the response
- What can we do about it?

Source: D. Forsyth

Solution: smooth first



- To find edges, look for peaks in $\frac{d}{dx}(f * g)$

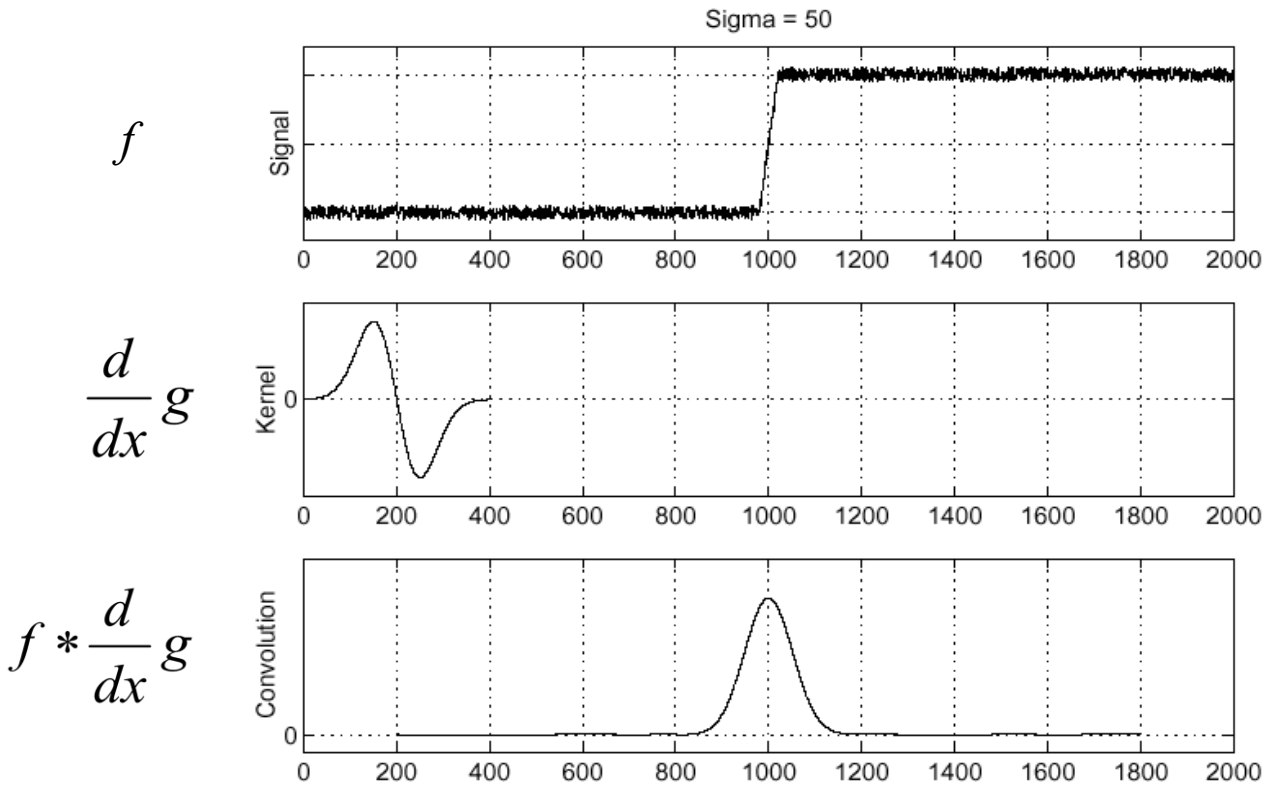
Source: S. Seitz

Derivative theorem of convolution

- Differentiation is convolution, and convolution is associative:

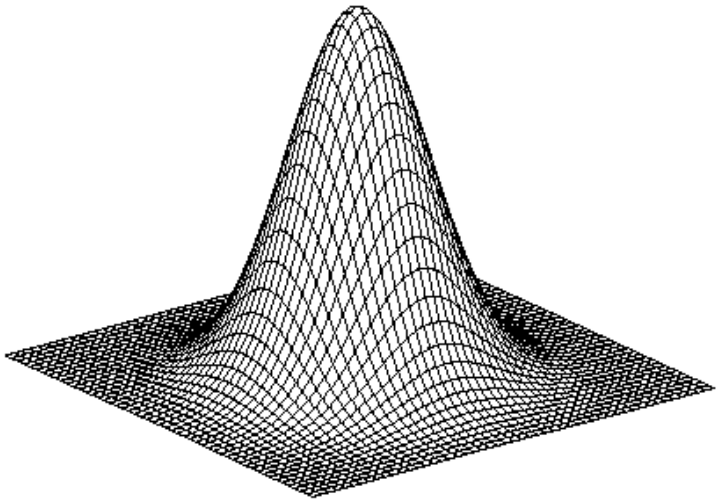
$$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$$

- This saves us one operation:

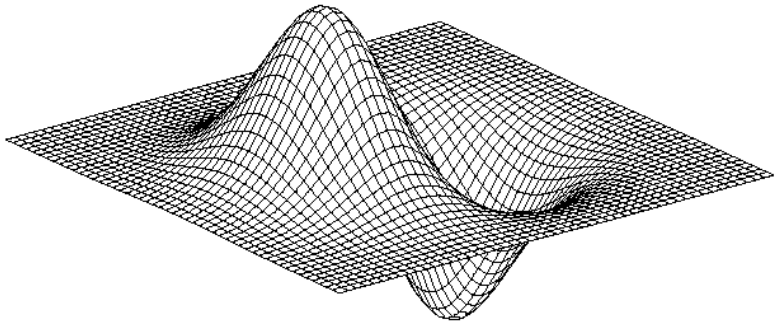


Source: S. Seitz

Derivative of Gaussian filter



* [1 -1] =



Discrete Derivative in 1D

$$\frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x) - f(x - \Delta x)}{\Delta x} = f'(x)$$

$$\frac{df}{dx} = \frac{f(x) - f(x-1)}{1} = f'(x)$$

$$\frac{df}{dx} = f(x) - f(x-1) = f'(x)$$

1D Discrete derivatives & Filters

derivate filters

Backward $\frac{df}{dx} = f(x) - f(x-1) = f'(x)$ [0 1 -1]

Forward $\frac{df}{dx} = f(x) - f(x+1) = f'(x)$ [-1 1 0]

Central $\frac{df}{dx} = f(x+1) - f(x-1) = f'(x)$ [1 0 -1]

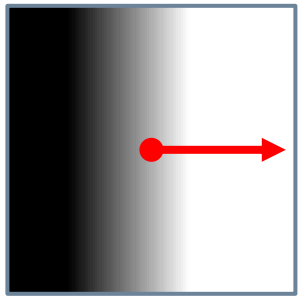
Discrete derivate in 2D

Given function $f(x, y)$

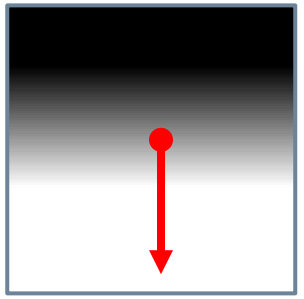
Gradient vector $$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x, y)}{\partial x} \\ \frac{\partial f(x, y)}{\partial y} \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

Gradient magnitude $$\left|\nabla f(x, y)\right| = \sqrt{f_x^2 + f_y^2}$$

Gradient direction $$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$



$\nabla f = [\frac{\partial f}{\partial x}, 0]$



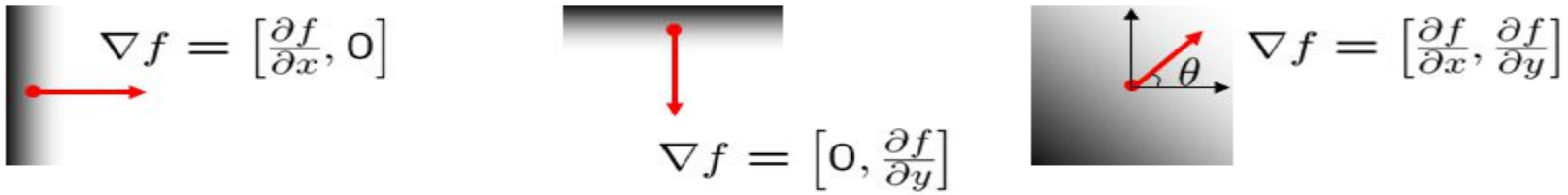
$\nabla f = [0, \frac{\partial f}{\partial y}]$

$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}]$

Discrete derivate in 2D

Image gradient

The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$



The gradient points in the direction of most rapid increase in intensity

The gradient direction is given by $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$

- how does this relate to the direction of the edge?

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Source: Steve Seitz

2D discrete derivative filters

$$\frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\frac{1}{3}\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Normally, don't write

$$I = \begin{bmatrix} 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \end{bmatrix}$$

$$I_x = \begin{bmatrix} 0 & 10 & 10 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \\ 0 & 10 & 10 & 0 & 0 \end{bmatrix}$$

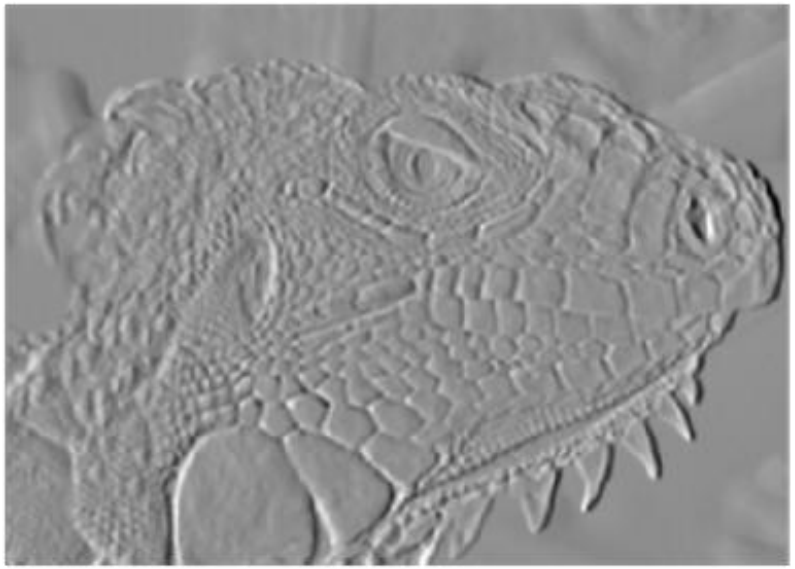
$$I_y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

3x3 image gradient filters

Prewitt operator

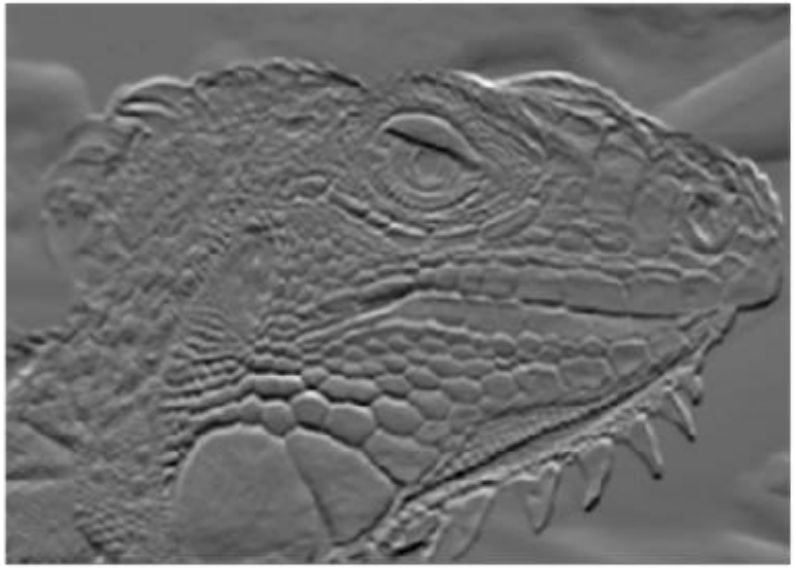
$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Derivative in x direction



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

Derivative in y direction



Sobel Operator

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \qquad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Sobel operation = Smoothing + differentiation

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} +1 & 0 & -1 \end{bmatrix}$$

Gaussian smoothing

differentiation

Other Edge operators

−1	0	0	−1
0	1	1	0

Roberts

$$\begin{bmatrix} +3 & 0 & -3 \\ +10 & 0 & -10 \\ +3 & 0 & -3 \end{bmatrix}$$

Scharr-x

$$\begin{bmatrix} +3 & +10 & +3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

Scharr-y

−1	−1	−1	−1	0	1	1	1	1	1	0	−1
0	0	0	−1	0	1	0	0	0	1	0	−1
1	1	1	−1	0	1	−1	−1	−1	1	0	−1
W			S			E			*N*		
0	1	1	−1	−1	0	0	−1	−1	1	1	0
−1	0	1	−1	0	1	1	0	−1	1	0	−1
−1	−1	0	0	1	1	1	1	0	0	−1	−1
SE			SW			NW			NE		

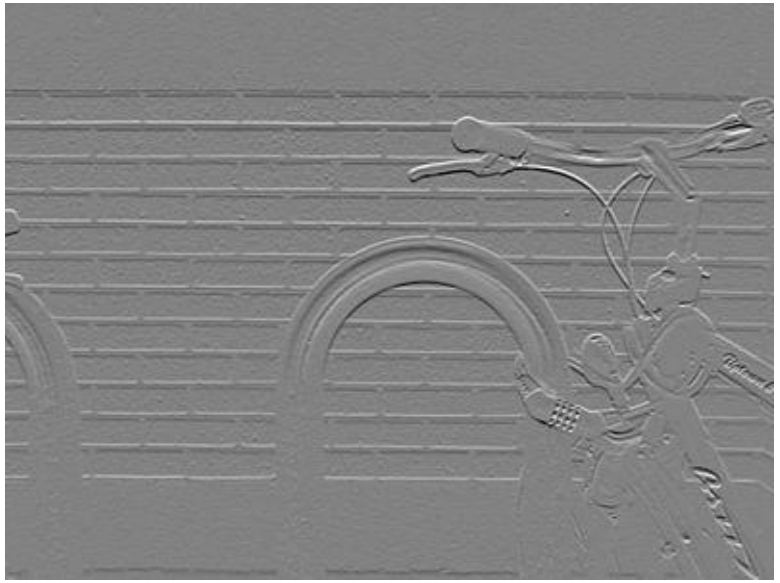
Kirsch compass operator

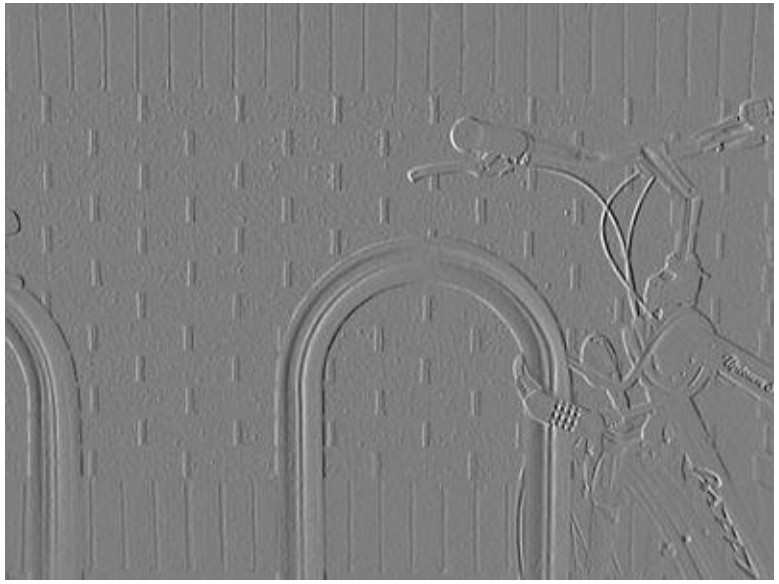
Simple Edge Detector

- Gradient based
  - Compute gradient magnitude
    - Use Sobel or any other edge operator
    - Optionally smooth before computing gradient
  - Apply thresholding

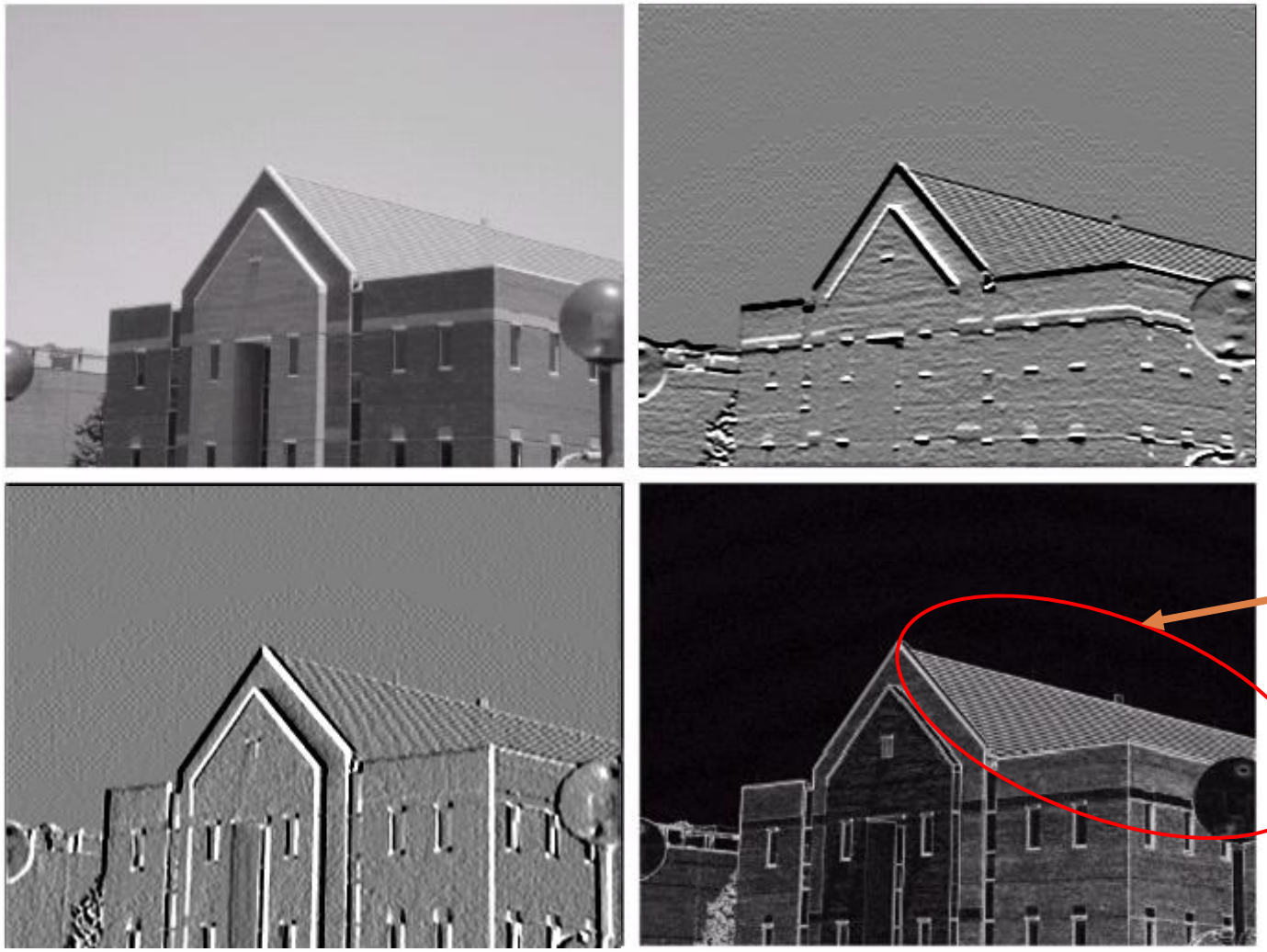
$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

Simple Edge Detector – Sobel based





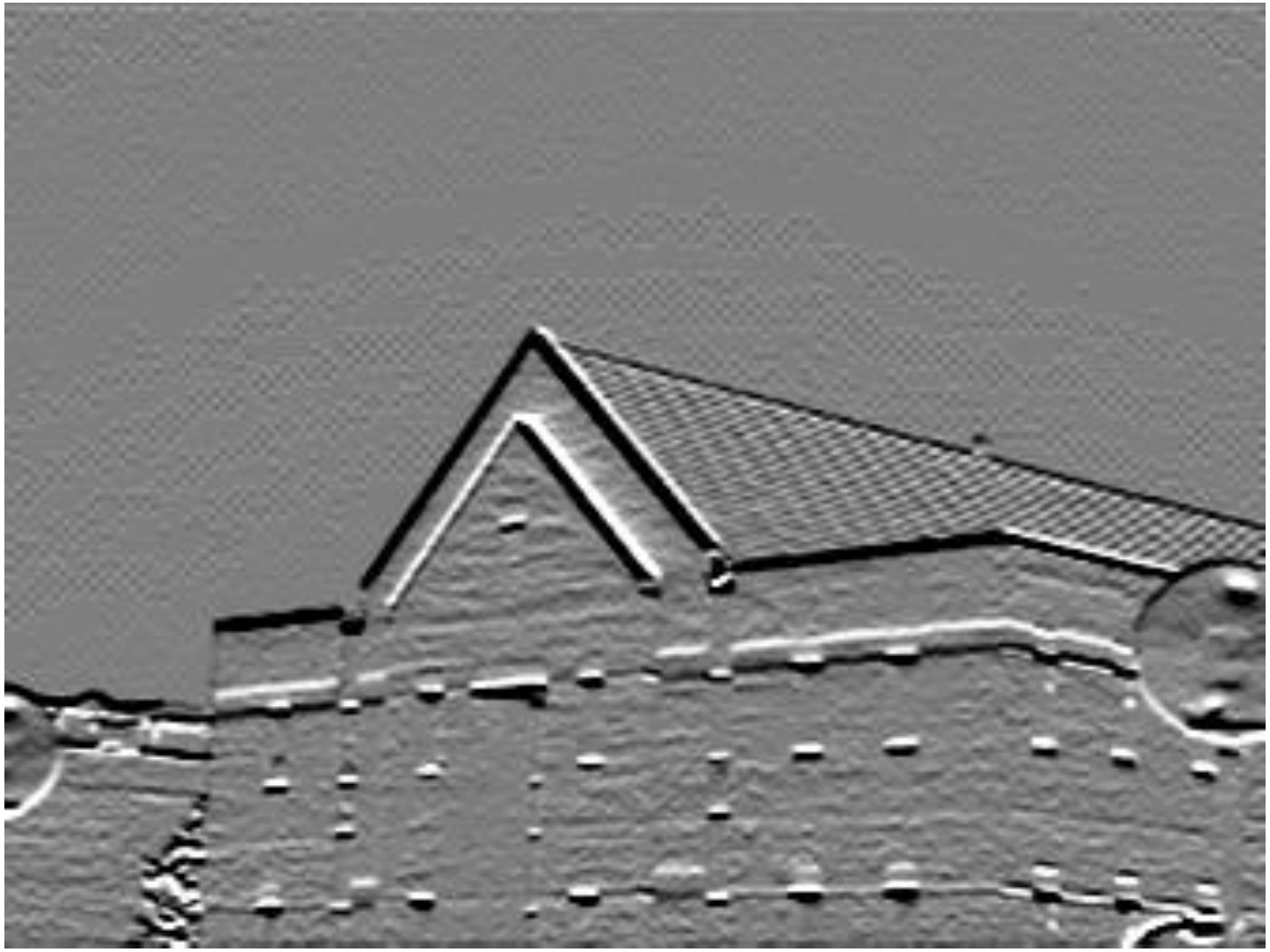
Simple Edge Detector – Sobel based



Edge Detection Example

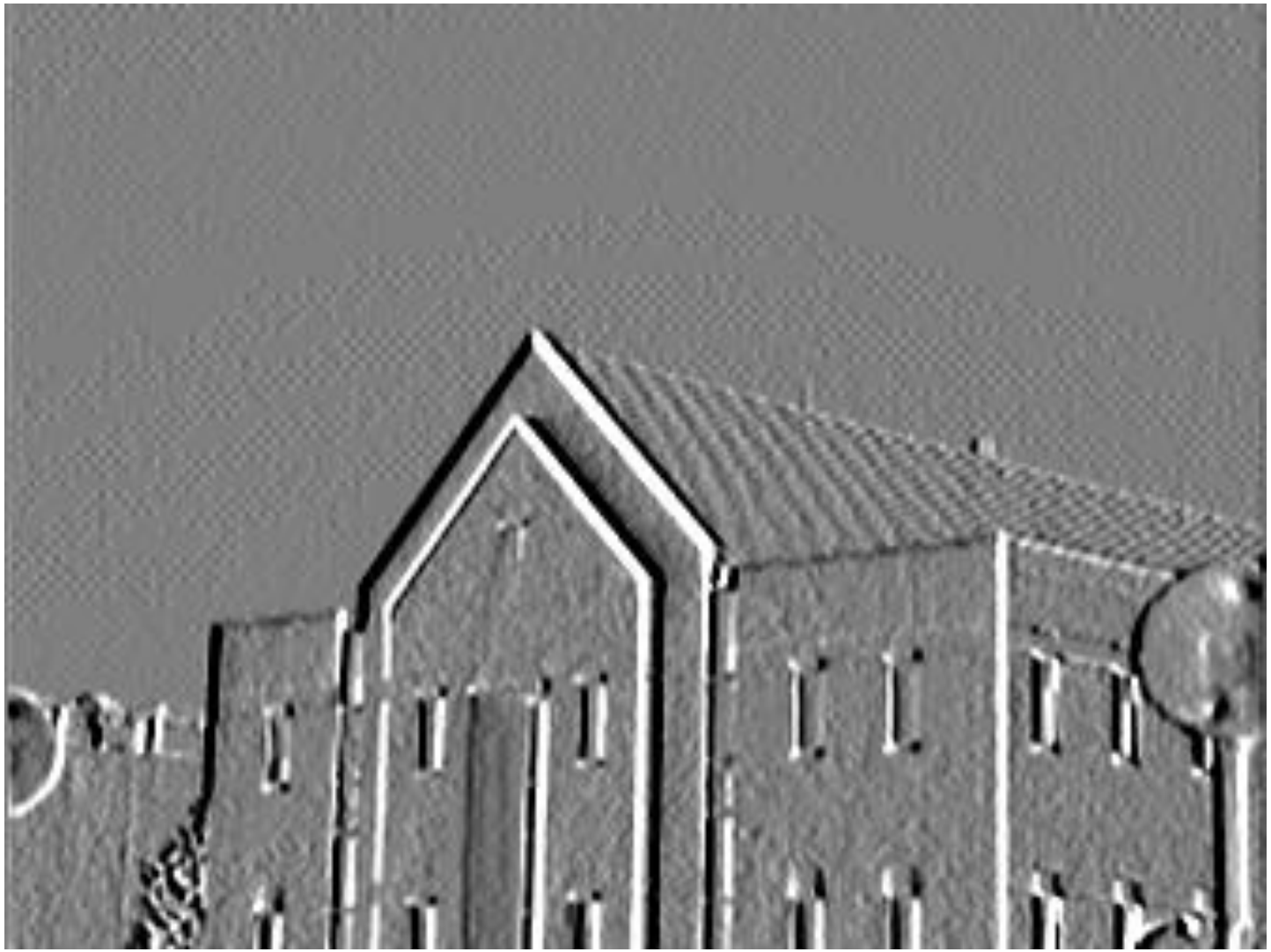
Edge Detection Example

Horizontal Gradient Component



Edge Detection Example

Vertical Gradient Component



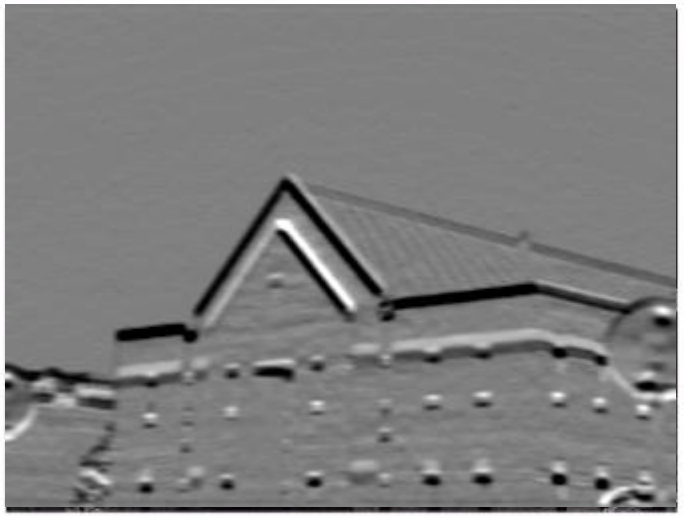
Edge Detection Example

Edge Detection Problems

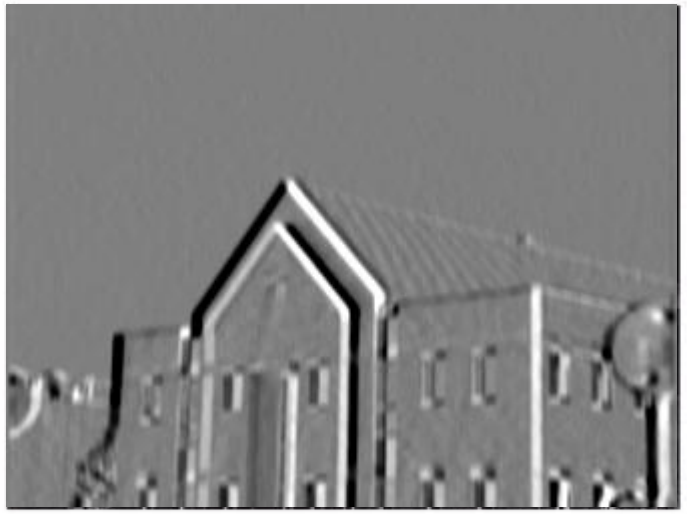
- Often, problems arise in edge detection in that there are is too much detail
- For example, the brickwork in the previous example
- One way to overcome this is to smooth images prior to edge detection

Simple Edge Detector – Sobel based

- With Smoothing - 5x5 average filter



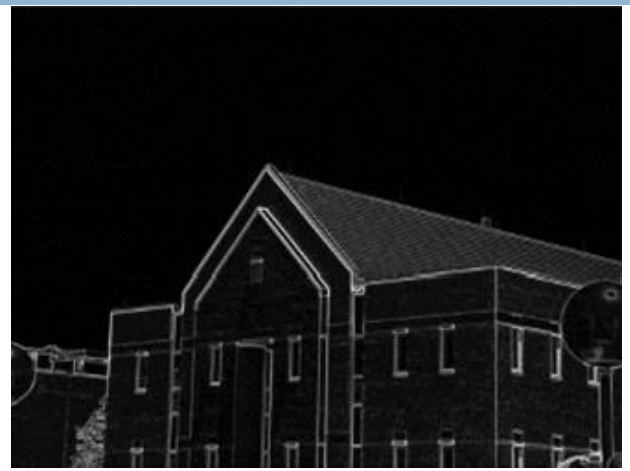
Gy



Gx

Magnitude

Simple Edge Detector - threshold



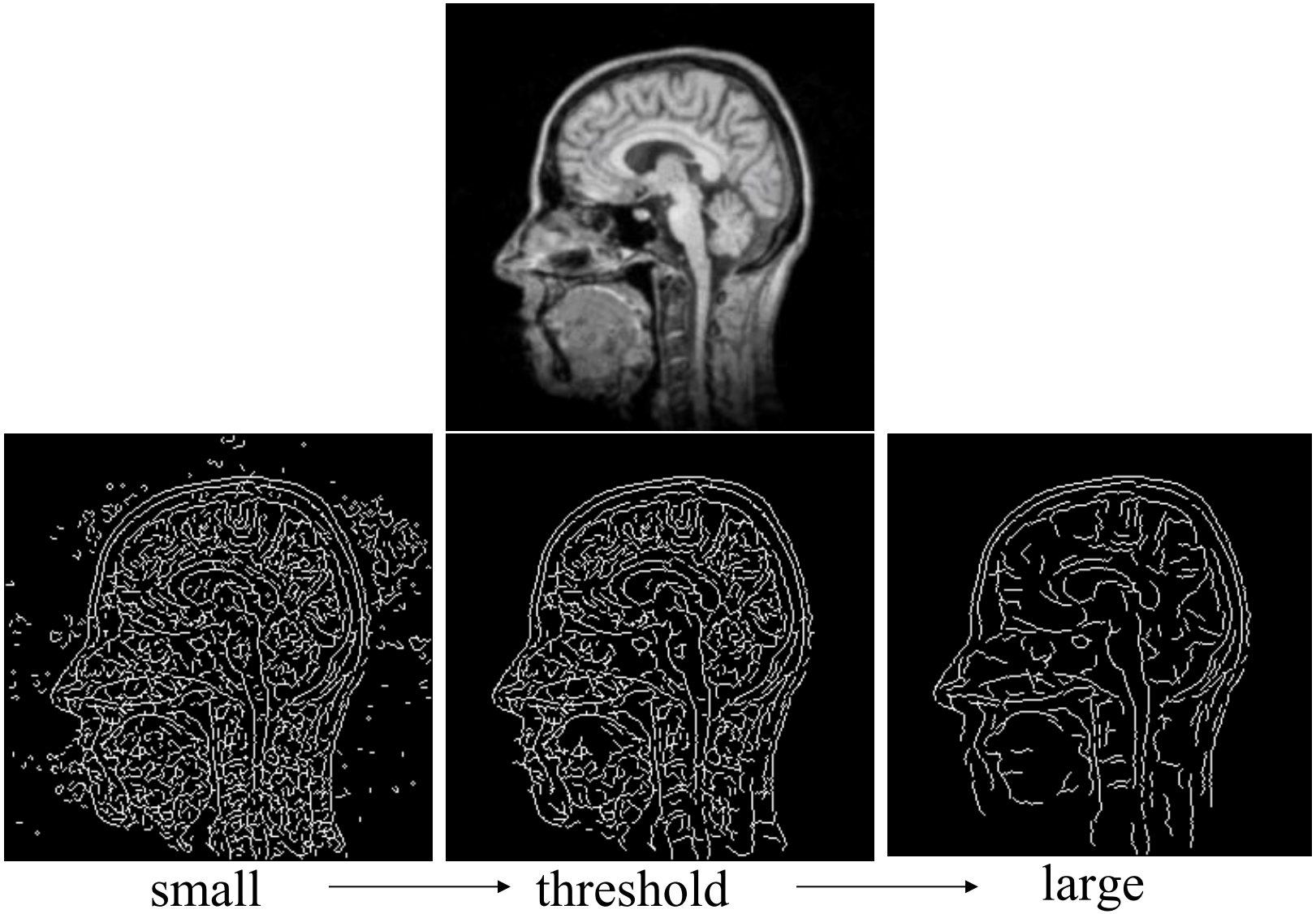
50 80 100

small threshold large

Simple Edge Detector

- **Issues**
  - Poor Localization (Trigger response in multiple adjacent pixels)
  - Thresholding value favors certain directions over others
  - Sobel can miss oblique edges more than horizontal or vertical edges
  - False negatives

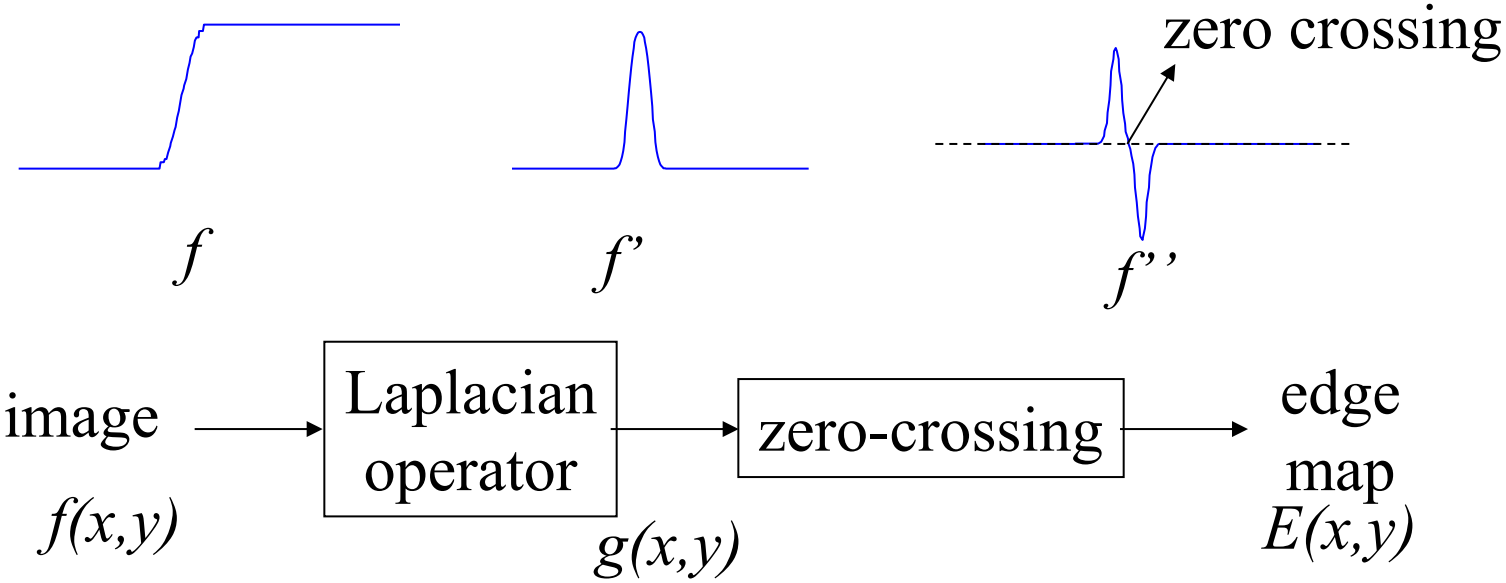
Combining Gradient with thresholds



Laplacian Edge Detection

- We encountered the 2nd-order derivative based Laplacian filter already

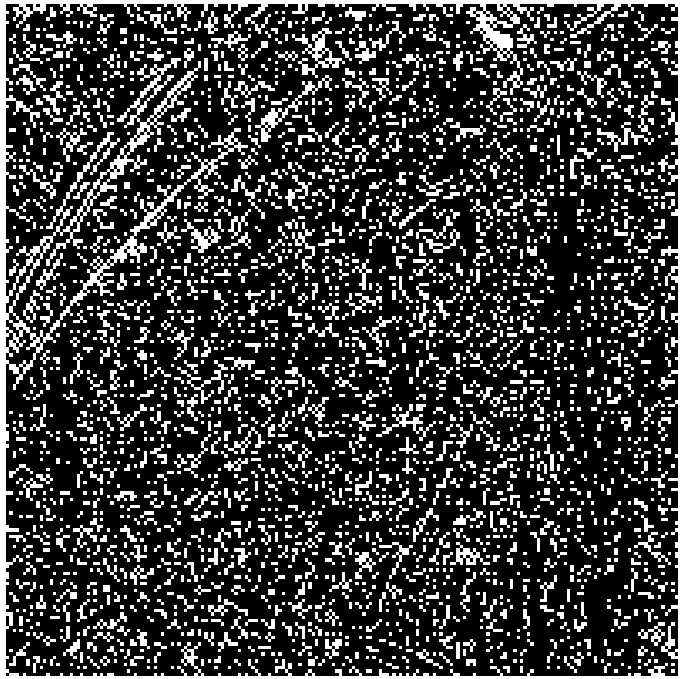
0	−1	0	−1	−1	−1
−1	4	−1	−1	8	−1
0	−1	0	−1	−1	−1



Laplacian Edge Detection

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

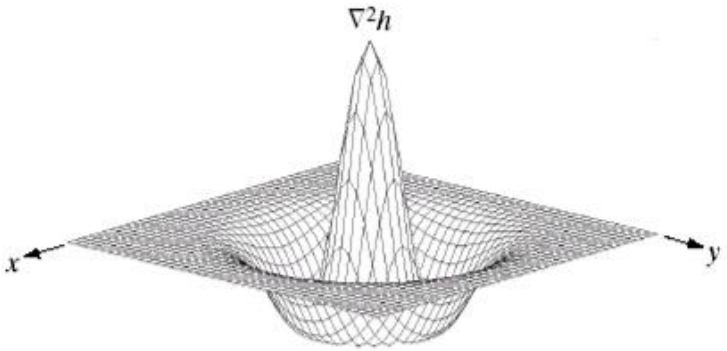
Ideas to Improve Robustness (SS)

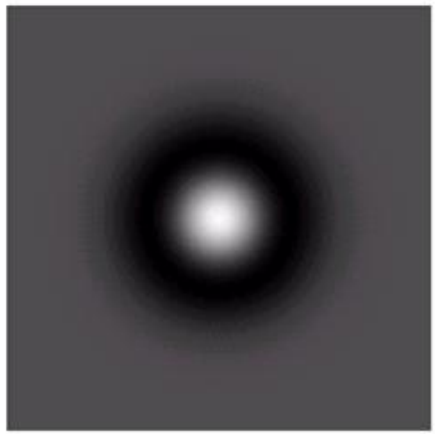
- The Laplacian is typically not used by itself as it is too sensitive to noise
- Linear filtering
  - Use a Gaussian filter to smooth out noise component → **Laplacian of Gaussian (LoG) filter (Marr – Hildreth operator)**
- Spatially-adaptive (Nonlinear) processing
  - Apply different detection strategies 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

Laplacian Of Gaussian

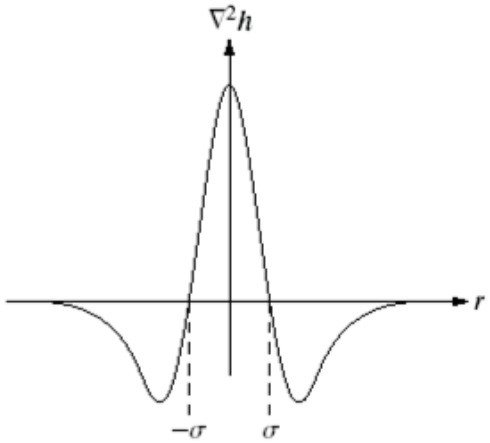
- The Laplacian of Gaussian (or Mexican hat) filter uses the Gaussian for noise removal and the Laplacian for edge detection

$$\nabla^2 G(x,y) = \left[\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right] e^{-\frac{x^2+y^2}{2\sigma^2}}$$

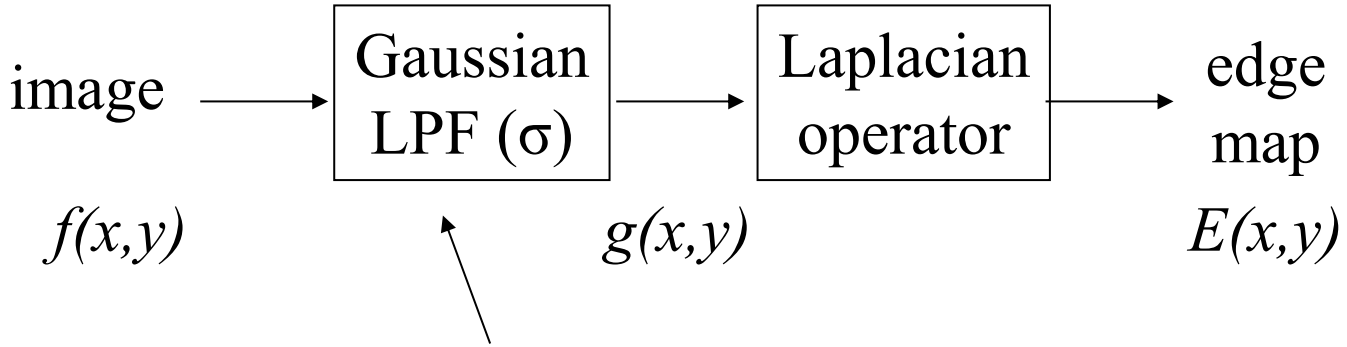




0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

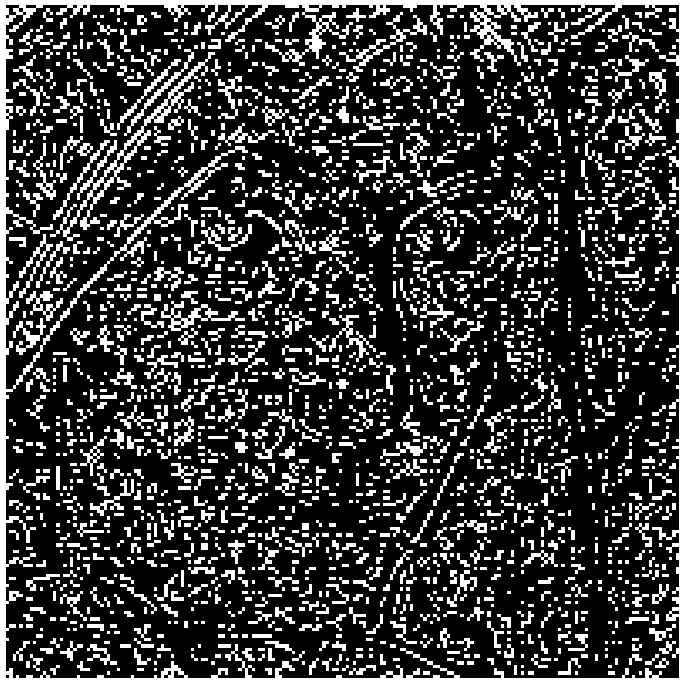


Laplacian Of Gaussian

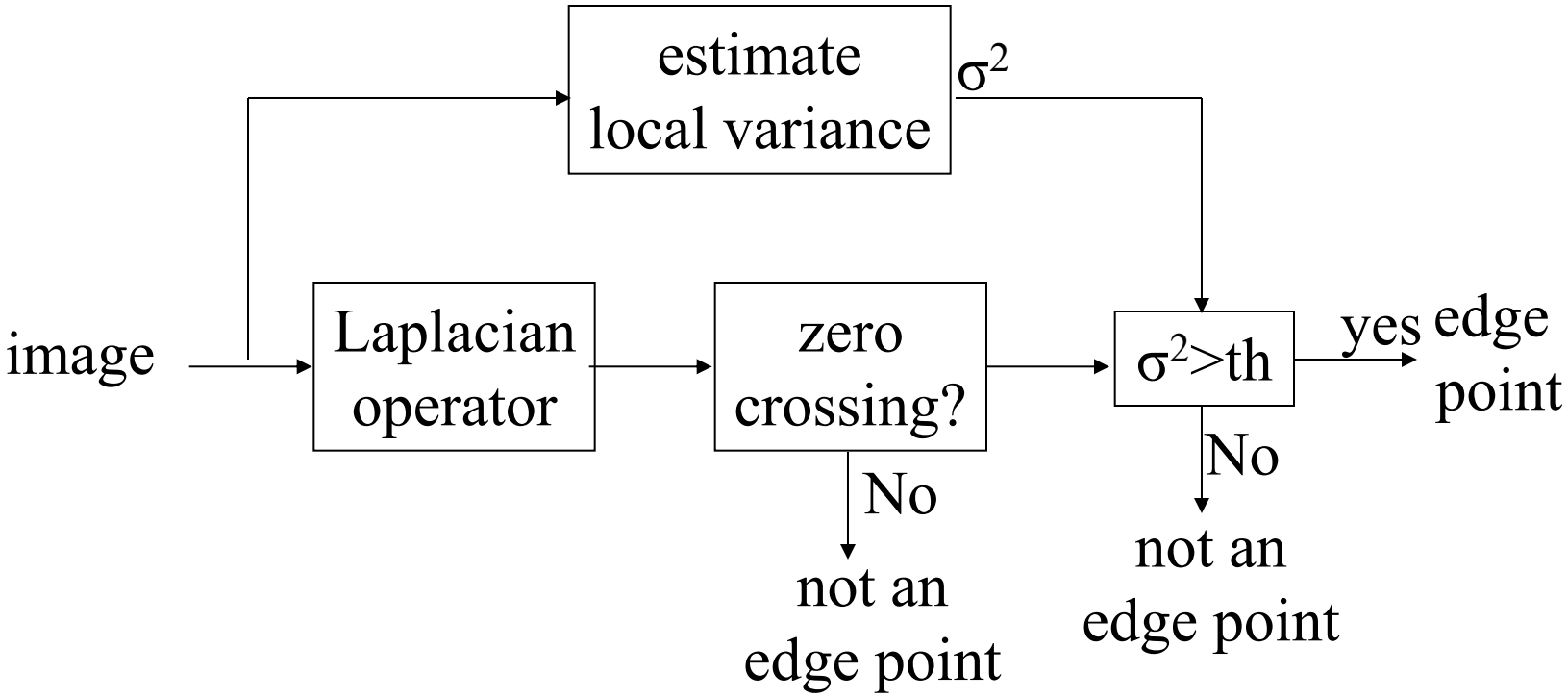


Pre-filtering: attenuate the noise sensitivity of the Laplacian

Better than Laplacian alone but still sensitive due to zero crossing



Robust Laplacian-based Edge Detector

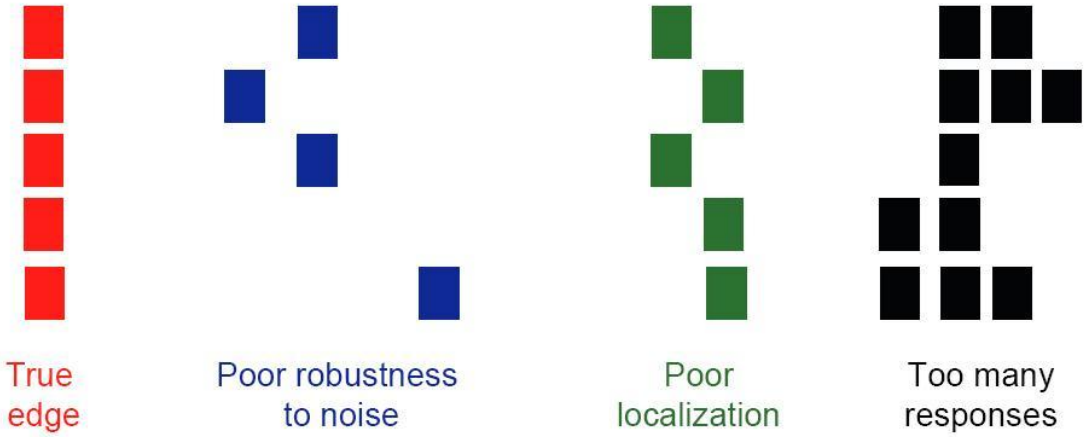


Robust Laplacian-based Edge Detector

More robust but return multiple edge pixels (poor localization)

Designing an edge detector

- Criteria for an “optimal” edge detector:
  - **Good detection:** must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges)
  - **Good localization:** the edges detected must be as close as possible to the true edges
  - **Single response:** must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge

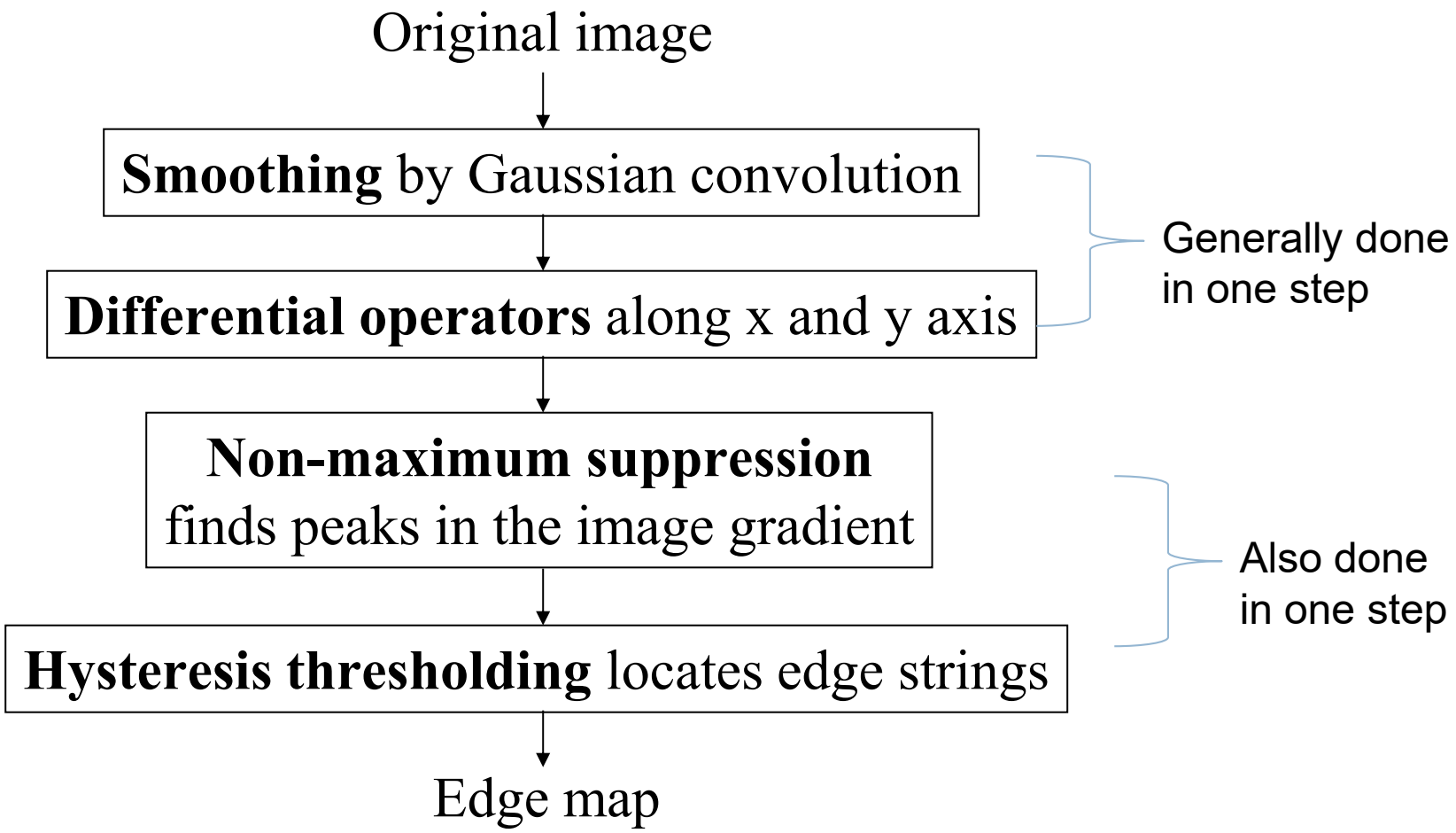


Canny Edge Detector

- most widely used edge detector in computer vision
- Low error rate of detection
  - Well match human perception results
- Good localization of edges
  - The distance between actual edges in an image and the edges found by a computational algorithm should be minimized
- Single response
  - The algorithm should not return multiple edges pixels when only a single one exists

Flow-chart of Canny Edge Detector*

(J. Canny'1986)



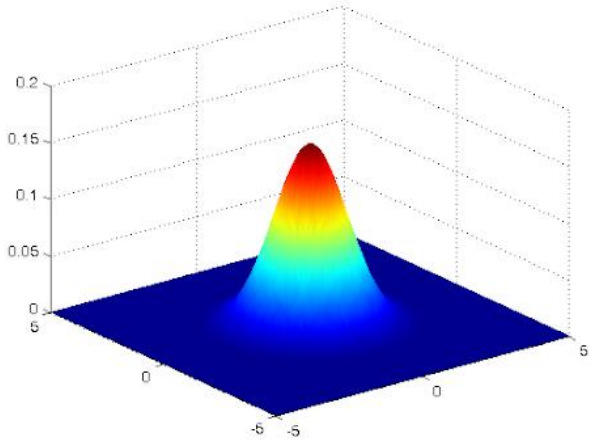
Example

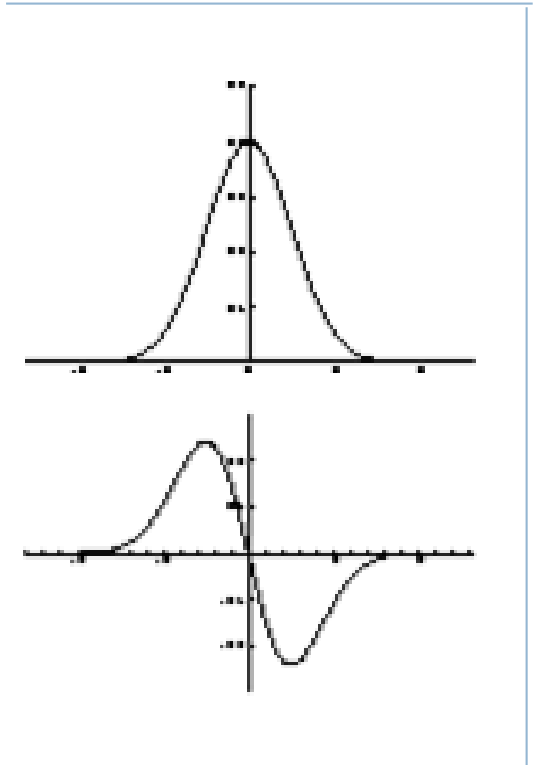
original image (Lena)

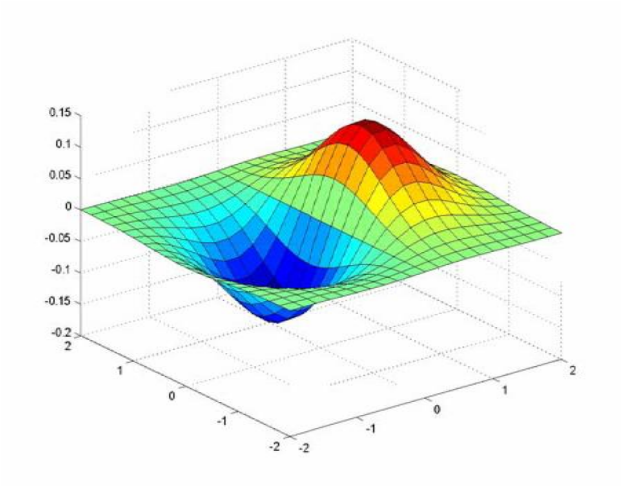
Derivative of Gaussian filter

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

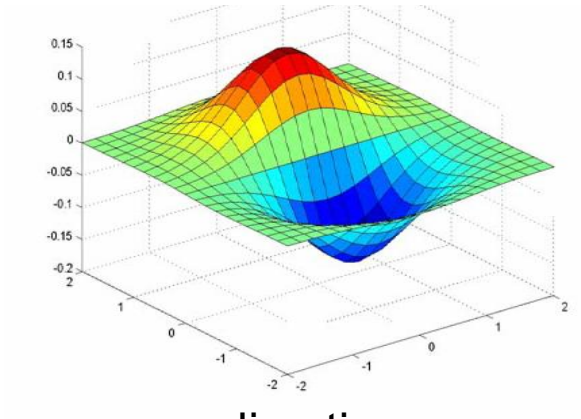
$$G'(r) = \frac{-r}{\sigma^2} G(r)$$



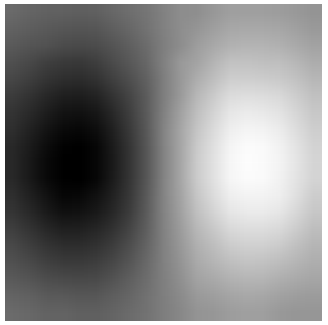


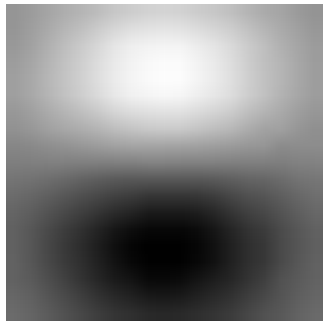


x-direction



y-direction





Compute Gradients

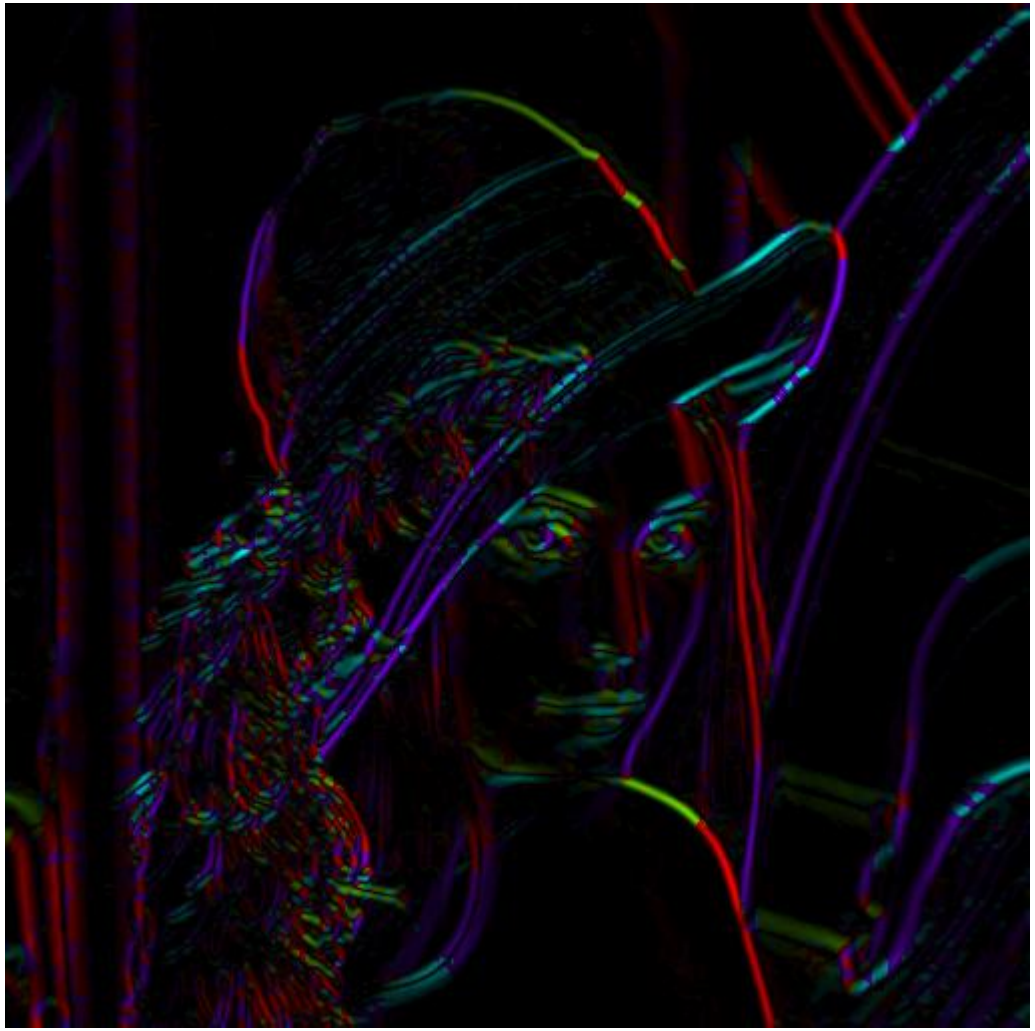
X-Derivative of Gaussian

Y-Derivative of Gaussian

Gradient Magnitude

Get Orientation at Each Pixel

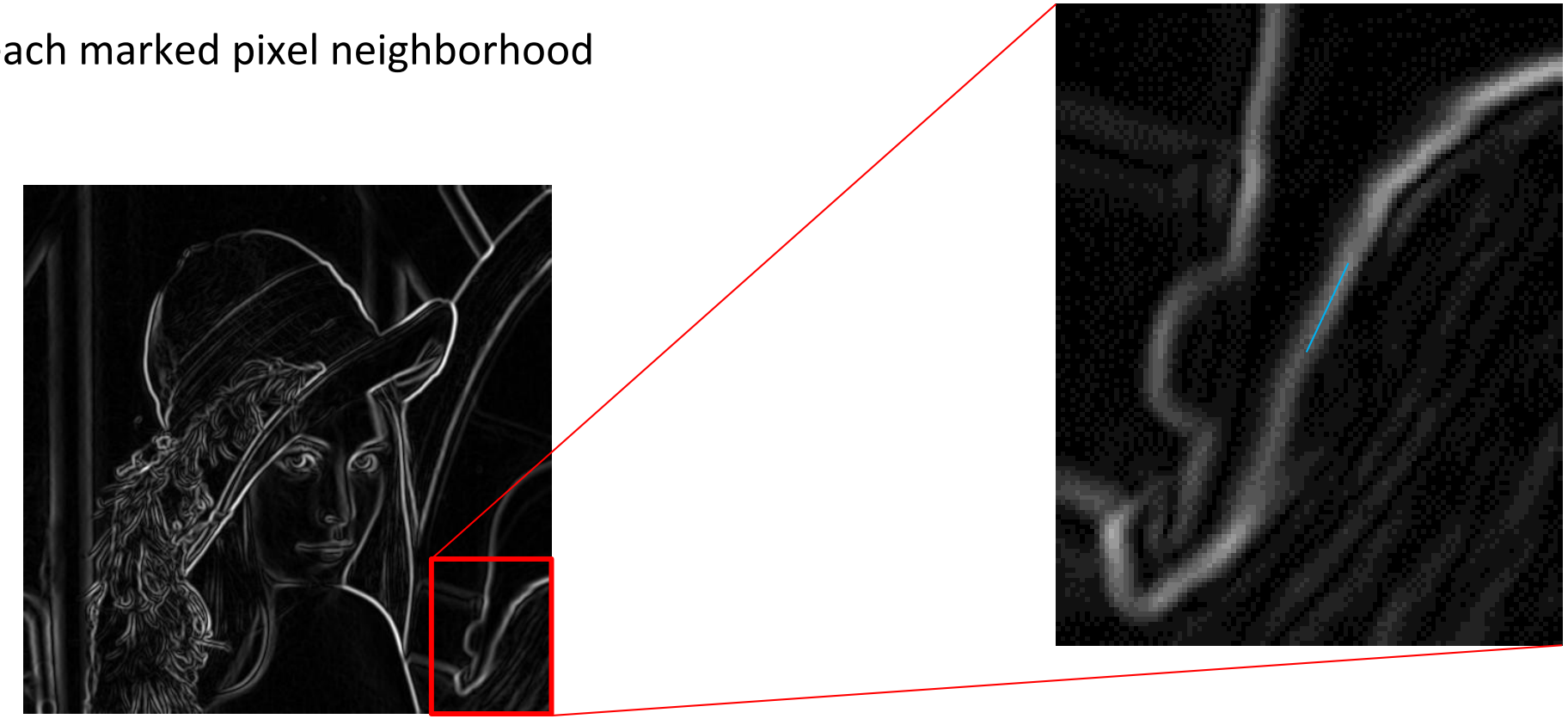
- Get orientation



theta = atan2(Gy, Gx)

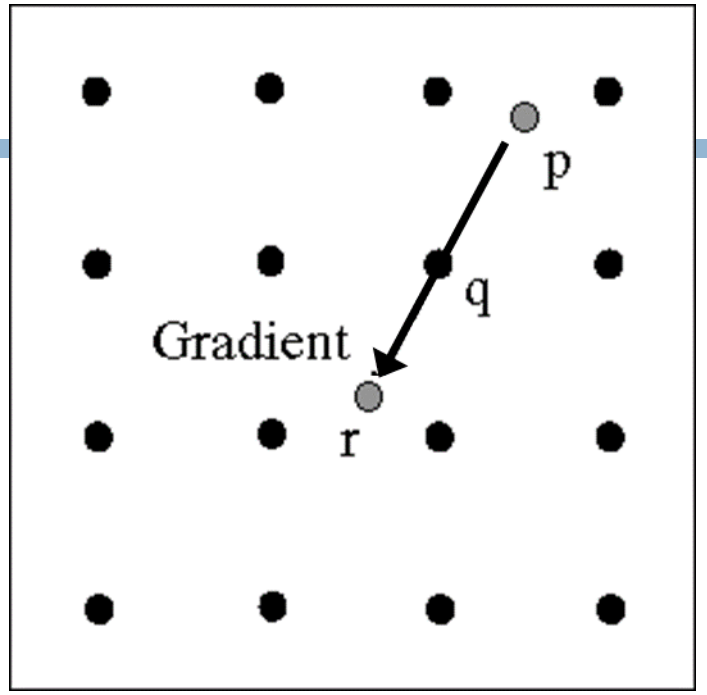
Non-maximum suppression

- Edge occurs where gradient reaches a maxima
- Suppress non-maxima gradient even if it passes threshold
- Only eight angle directions possible
  - Suppress all pixels in each direction which are not maxima
  - Do this in each marked pixel neighborhood



Non-maximum suppression

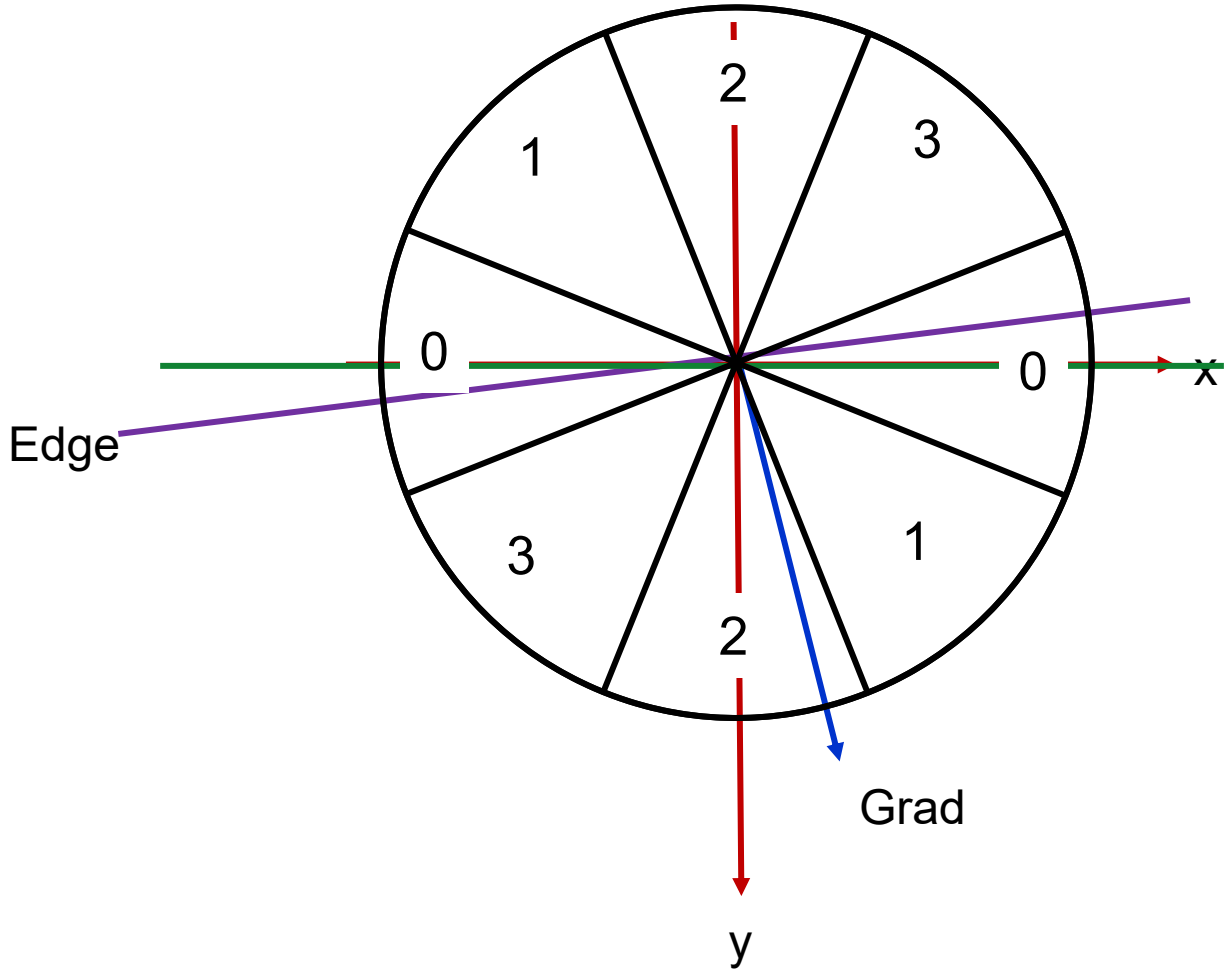
At q, we have a maximum if the value is larger than those at both p and at r.



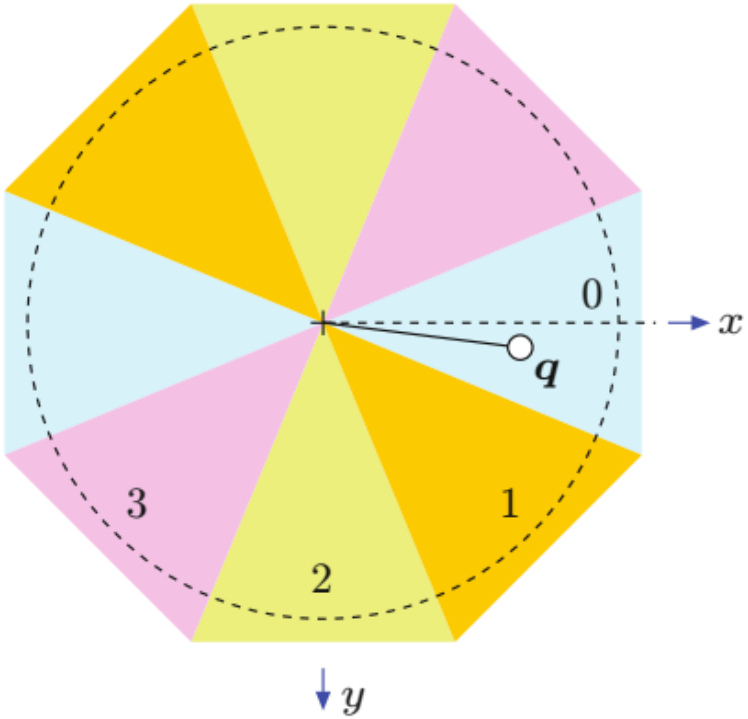
$$M(q) = \begin{cases} |\nabla f(q)| & \text{if } |\nabla f(q)| > |\nabla f(p)|\,| \\ & \text{and } |\nabla f(q)| > |\nabla f(r)|\,| \\ 0 & \text{otherwise} \end{cases}$$

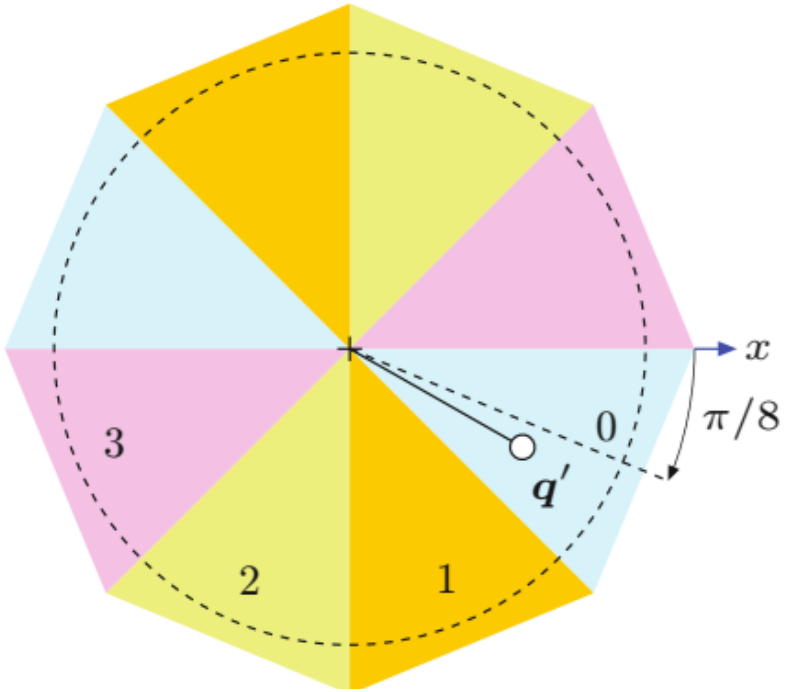
q= current pixel (x,y),
p, r = neighbor positions along gradient direction
Interpolate to get magnitude of these values.

Non-maximum suppression – avoid tan function



Non-maximum suppression – avoid tan function



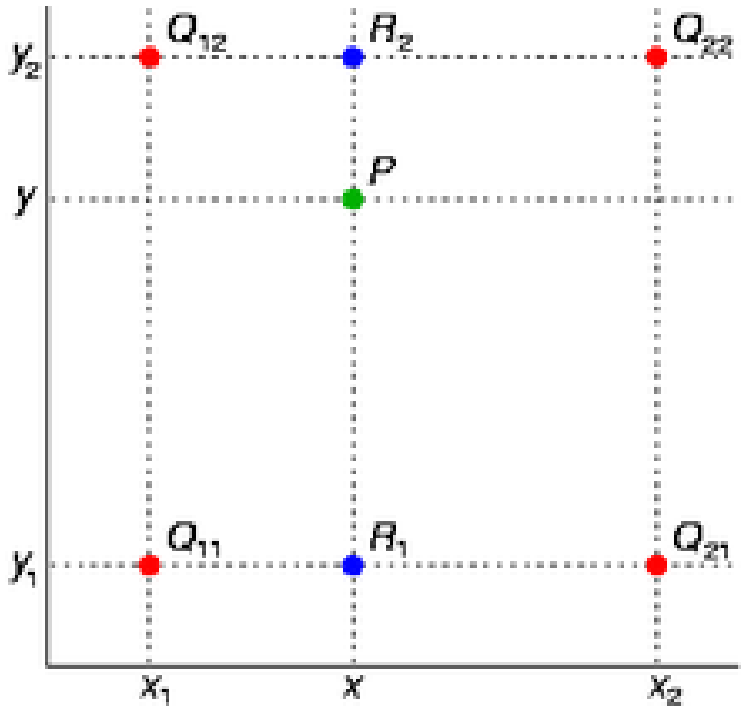


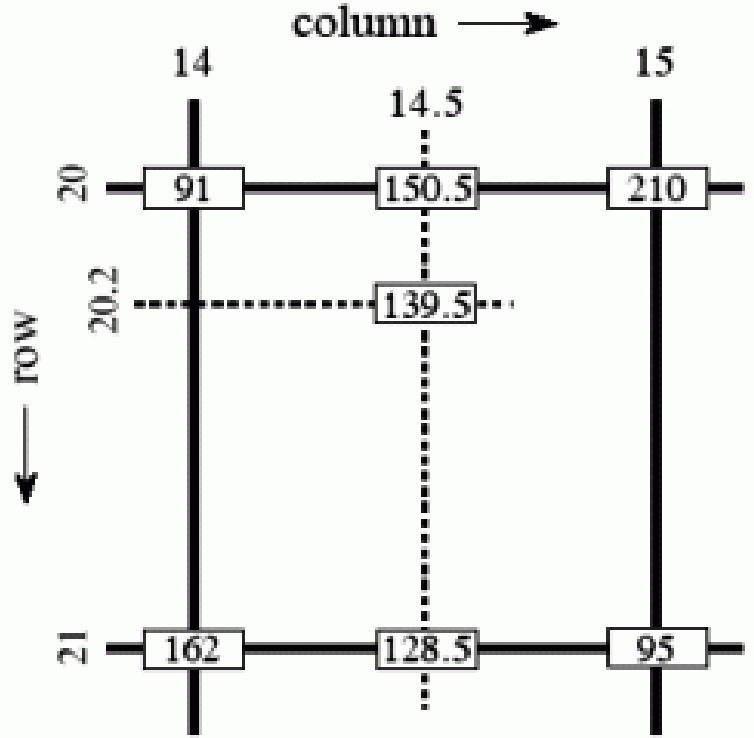
$$\begin{pmatrix} d'_x \\ d'_y \end{pmatrix} \leftarrow \begin{pmatrix} \cos(\pi/8) & -\sin(\pi/8) \\ \sin(\pi/8) & \cos(\pi/8) \end{pmatrix} \cdot \begin{pmatrix} d_x \\ d_y \end{pmatrix}$$

$$s_\theta \leftarrow \begin{cases} 0 & \text{if } (d'_x \geq 0) \wedge (d'_x \geq d'_y) \\ 1 & \text{if } (d'_x \geq 0) \wedge (d'_x < d'_y) \\ 2 & \text{if } (d'_x < 0) \wedge (-d'_x < d'_y) \\ 3 & \text{if } (d'_x < 0) \wedge (-d'_x \geq d'_y) \end{cases}$$

Sidebar: Bilinear Interpolation

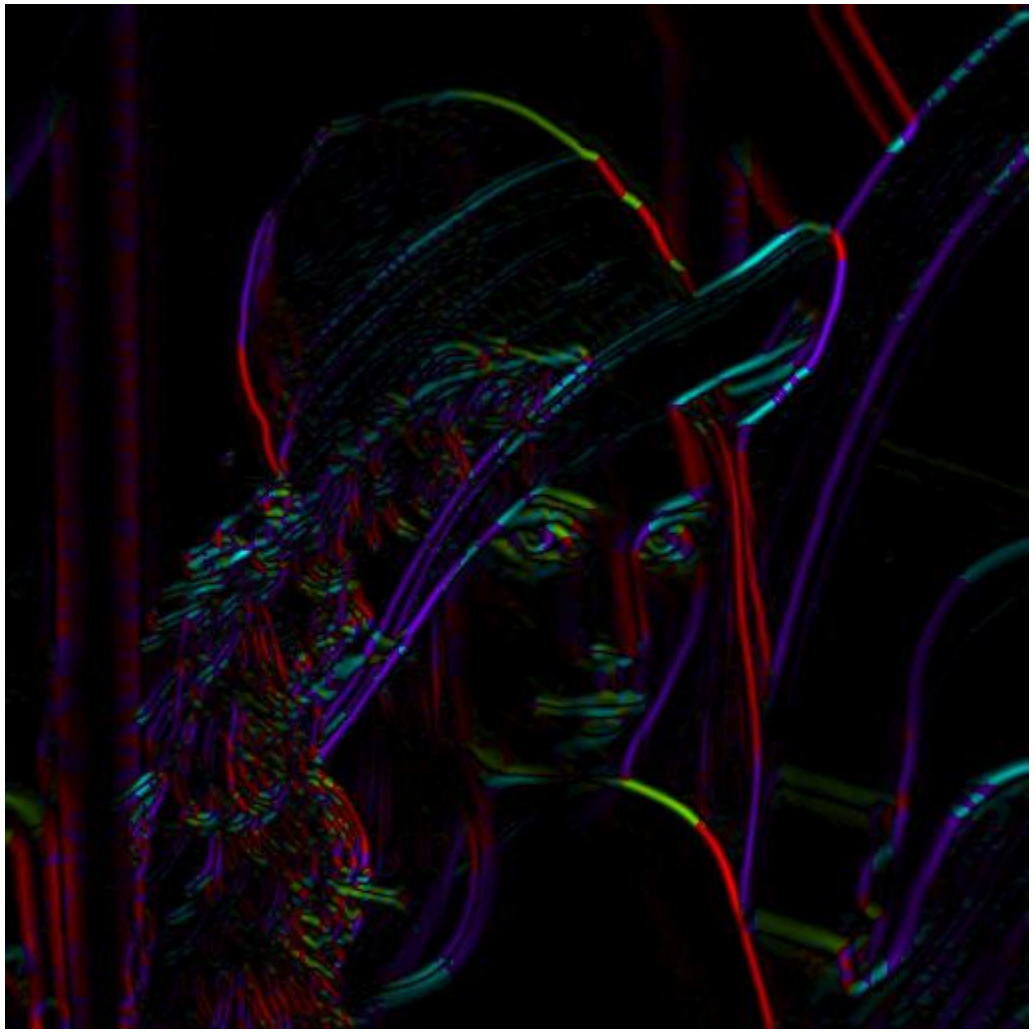
$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}.$$



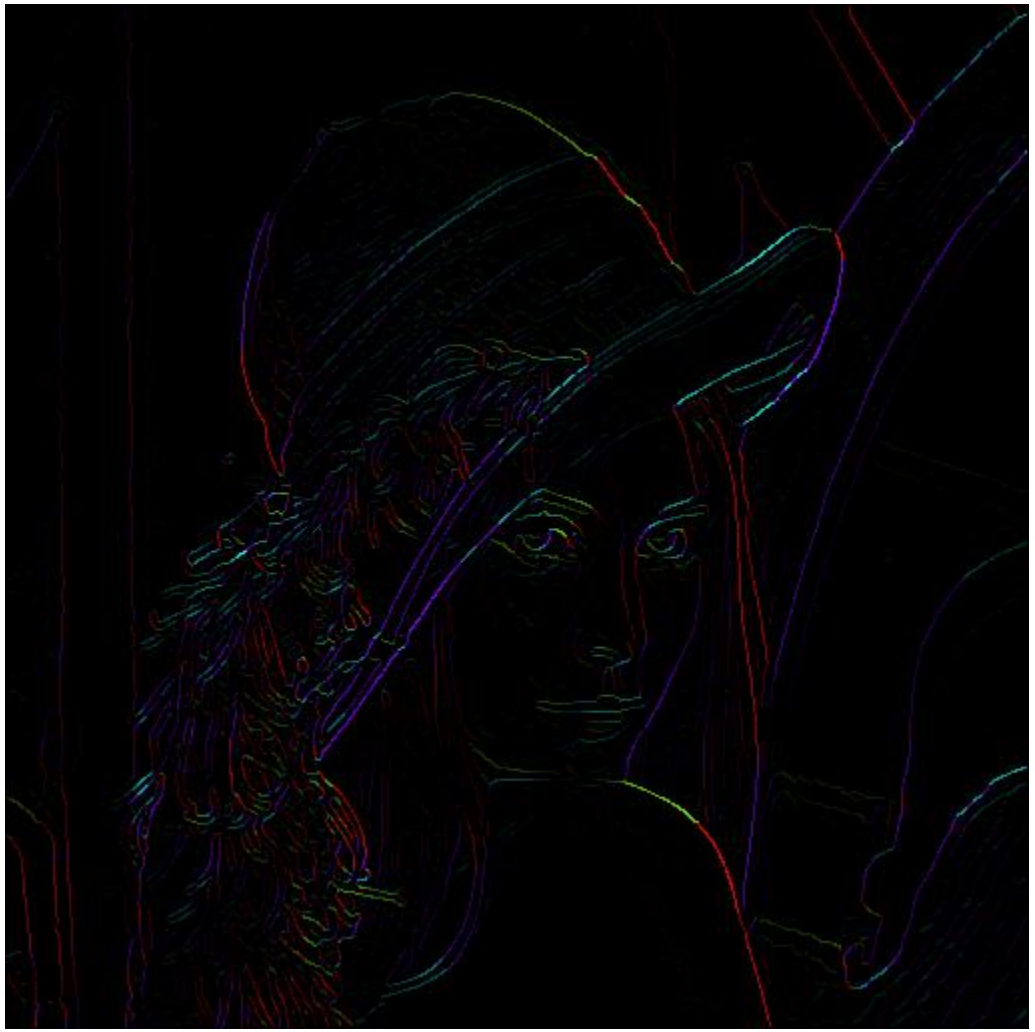


http://en.wikipedia.org/wiki/Bilinear_interpolation

Before Non-max Suppression



After non-max suppression



Before Non-max Suppression

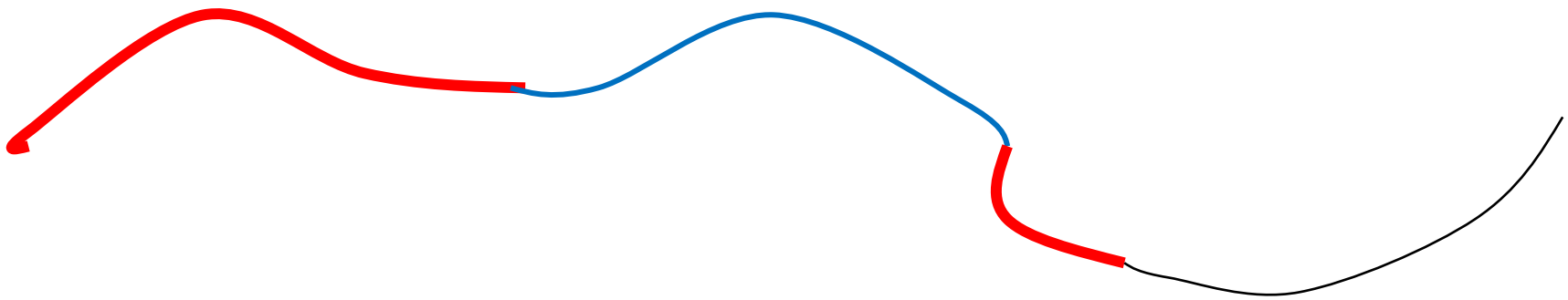
After non-max suppression

Hysteresis thresholding

- Threshold at low/high levels to get weak/strong edge pixels
- Do connected components, starting from strong edge pixels

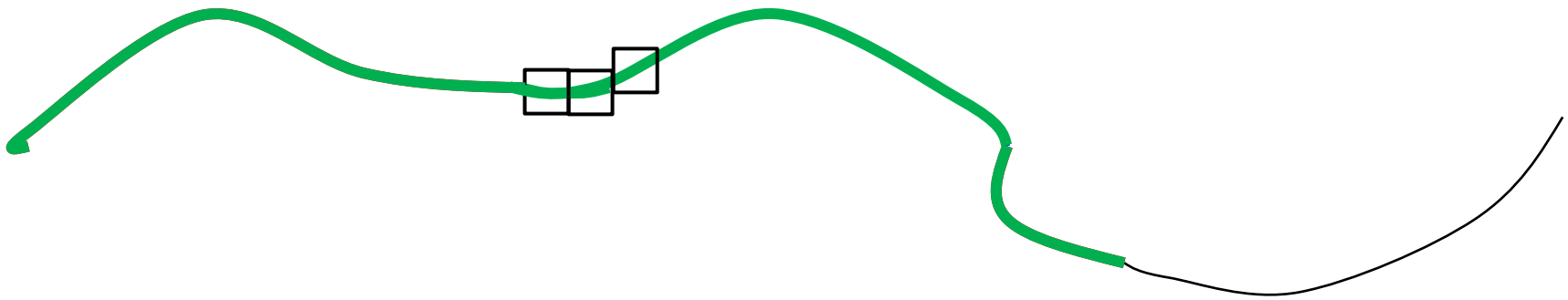
Hysteresis Threshold

- Canny uses hysteresis threshold to fill up the narrow gap along the edge.
- Uses two threshold - high threshold T_h and low threshold T_l.
- Pixels with gray values greater than T_h is set as the initial edge pixels
- Pixels are set as background if their values are lower than T_l.



Hyteresis Threshold

- A following recursive process takes charge of pixels lying between T_h and T_l.
  - Let, call them as vibrating pixels.
  - The recurs checks 8-adjancent neighbors of each initial edge pixel to see whether there are vibrating pixels.
  - Vibrating pixels will be added to the initial edge pixel set.
  - This process will go on until all the initial edge pixels have been recursed, and vibrating pixels that have not been visited will be set as background pixel.



Hysteresis thresholding

Hysteresis threshold

Canny Algorithm

1: **CannyEdgeDetector**$(I, \sigma, \mathrm{t_{hi}}, \mathrm{t_{lo}})$

Input: I, a grayscale image of size $M \times N$; σ, scale (radius of Gaussian filter $H^{\mathrm{G},\sigma}$); $\mathrm{t_{hi}}$, $\mathrm{t_{lo}}$, hysteresis thresholds ($\mathrm{t_{hi}} > \mathrm{t_{lo}}$).
Returns a binary edge map of size $M \times N$.

2: $\bar{I} \leftarrow I * H^{\mathrm{G},\sigma}$ ▷ blur with Gaussian of width σ

3: $\bar{I}_x \leftarrow \bar{I} * [-0.5 \;\; \mathbf{0} \;\; 0.5]$ ▷ x-gradient

4: $\bar{I}_y \leftarrow \bar{I} * [-0.5 \;\; \mathbf{0} \;\; 0.5]^\mathsf{T}$ ▷ y-gradient

5: $(M, N) \leftarrow \mathsf{Size}(\boldsymbol{I})$

6: Create maps:

7: $E_{\mathrm{mag}} : M \times N \mapsto \mathbb{R}$ ▷ gradient magnitude

8: $E_{\mathrm{nms}} : M \times N \mapsto \mathbb{R}$ ▷ maximum magnitude

9: $E_{\mathrm{bin}} : M \times N \mapsto \{0, 1\}$ ▷ binary edge pixels

10: **for all** image coordinates $(u, v) \in M \times N$ **do**

11: $E_{\mathrm{mag}}(u, v) \leftarrow \left[\bar{I}_x^2(u, v) + \bar{I}_y^2(u, v)\right]^{1/2}$

12: $E_{\mathrm{nms}}(u, v) \leftarrow 0$

13: $E_{\mathrm{bin}}(u, v) \leftarrow 0$

Canny Algorithm

```
14:   for u ← 1, ..., M−2 do
15:       for v ← 1, ..., N−2 do
16:           d_x ← Ī_x(u, v),    d_y ← Ī_y(u, v)
17:           s_θ ← GetOrientationSector(d_x, d_y)                ▷ Alg. 6.2
18:           if IsLocalMax(E_mag, u, v, s_θ, t_lo) then          ▷ Alg. 6.2
19:               E_nms(u, v) ← E_mag(u, v)    ▷ only keep local maxima
20:   for u ← 1, ..., M−2 do
21:       for v ← 1, ..., N−2 do
22:           if (E_nms(u, v) ≥ t_hi) ∧ (E_bin(u, v) = 0) then
23:               TraceAndThreshold(E_nms, E_bin, u, v, t_lo)
                                                                   ▷ Alg. 6.2
24:   return E_bin.
```

Canny Algorithm

GetOrientationSector(d_x, d_y)

Returns the discrete octant s_θ for the orientation vector $(d_x, d_y)^\mathsf{T}$. See Fig. 6.13 for an illustration.

$$\begin{pmatrix} d'_x \\ d'_y \end{pmatrix} \leftarrow \begin{pmatrix} \cos(\pi/8) & -\sin(\pi/8) \\ \sin(\pi/8) & \cos(\pi/8) \end{pmatrix} \cdot \begin{pmatrix} d_x \\ d_y \end{pmatrix} \quad \triangleright \text{ rotate } \begin{pmatrix} d_x \\ d_y \end{pmatrix} \text{ by } \pi/8$$

if $d'_y < 0$ **then**

$$d'_x \leftarrow -d'_x, \qquad d'_y \leftarrow -d'_y \quad \triangleright \text{ mirror to octants } 0, \ldots, 3$$

$$s_\theta \leftarrow \begin{cases} 0 & \text{if } (d'_x \geq 0) \wedge (d'_x \geq d'_y) \\ 1 & \text{if } (d'_x \geq 0) \wedge (d'_x < d'_y) \\ 2 & \text{if } (d'_x < 0) \wedge (-d'_x < d'_y) \\ 3 & \text{if } (d'_x < 0) \wedge (-d'_x \geq d'_y) \end{cases}$$

return s_θ. $\quad \triangleright$ sector index $s_\theta \in \{0, 1, 2, 3\}$

Canny Algorithm

7: **IsLocalMax**$(E_{\text{mag}}, u, v, s_\theta, \mathsf{t}_{\text{lo}})$

Determines if the gradient magnitude E_{mag} is a local maximum at position (u, v) in direction $s_\theta \in \{0, 1, 2, 3\}$.

8: $m_{\text{C}} \leftarrow E_{\text{mag}}(u, v)$

9: **if** $m_{\text{C}} < \mathsf{t}_{\text{lo}}$ **then**

10: **return** false

11: **else**

12: $$m_{\text{L}} \leftarrow \begin{cases} E_{\text{mag}}(u-1, v) & \text{if } s_\theta = 0 \\ E_{\text{mag}}(u-1, v-1) & \text{if } s_\theta = 1 \\ E_{\text{mag}}(u, v-1) & \text{if } s_\theta = 2 \\ E_{\text{mag}}(u-1, v+1) & \text{if } s_\theta = 3 \end{cases}$$

13: $$m_{\text{R}} \leftarrow \begin{cases} E_{\text{mag}}(u+1, v) & \text{if } s_\theta = 0 \\ E_{\text{mag}}(u+1, v+1) & \text{if } s_\theta = 1 \\ E_{\text{mag}}(u, v+1) & \text{if } s_\theta = 2 \\ E_{\text{mag}}(u+1, v-1) & \text{if } s_\theta = 3 \end{cases}$$

14: **return** $(m_{\text{L}} \leq m_{\text{C}}) \wedge (m_{\text{C}} \geq m_{\text{R}})$.

Canny Algorithm

15: **TraceAndThreshold**$(E_{\text{nms}}, E_{\text{bin}}, u_0, v_0, \mathsf{t}_{\text{lo}})$

Recursively collects and marks all pixels of an edge that are 8-connected to (u_0, v_0) and have a gradient magnitude above t_{lo}.

16: $E_{\text{bin}}(u_0, v_0) \leftarrow 1$ ▷ mark (u_0, v_0) as an edge pixel

17: $u_{\text{L}} \leftarrow \max(u_0 - 1, 0)$ ▷ limit to image bounds

18: $u_{\text{R}} \leftarrow \min(u_0 + 1, M - 1)$

19: $v_{\text{T}} \leftarrow \max(v_0 - 1, 0)$

20: $v_{\text{B}} \leftarrow \min(v_0 + 1, N - 1)$

21: **for** $u \leftarrow u_{\text{L}}, \ldots, u_{\text{R}}$ **do**

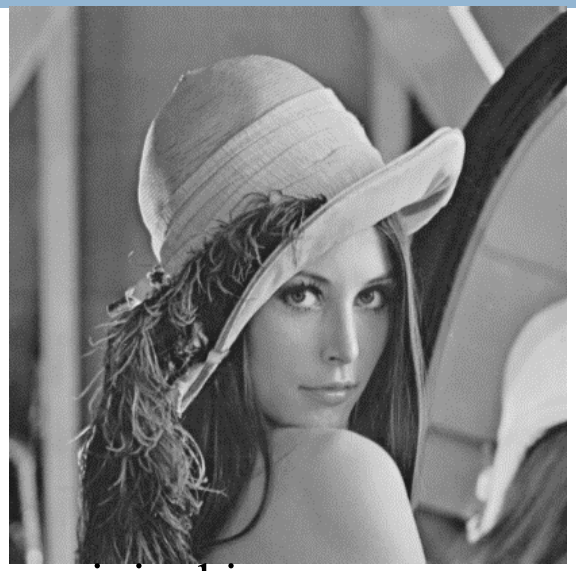
22: **for** $v \leftarrow v_{\text{T}}, \ldots, v_{\text{B}}$ **do**

23: **if** $(E_{\text{nms}}(u, v) \geq \mathsf{t}_{\text{lo}}) \wedge (E_{\text{bin}}(u, v) = 0)$ **then**

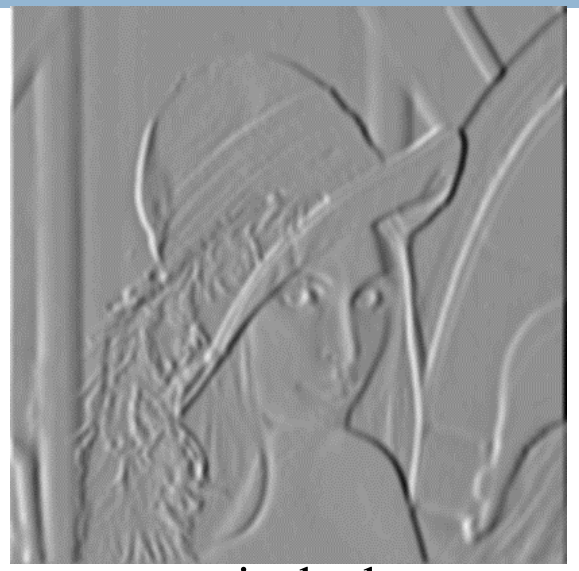
24: TraceAndThreshold$(E_{\text{nms}}, E_{\text{bin}}, u, v, \mathsf{t}_{\text{lo}})$

25: **return**

Canny Edge Detector Example



original image



vertical edges

horizontal edges

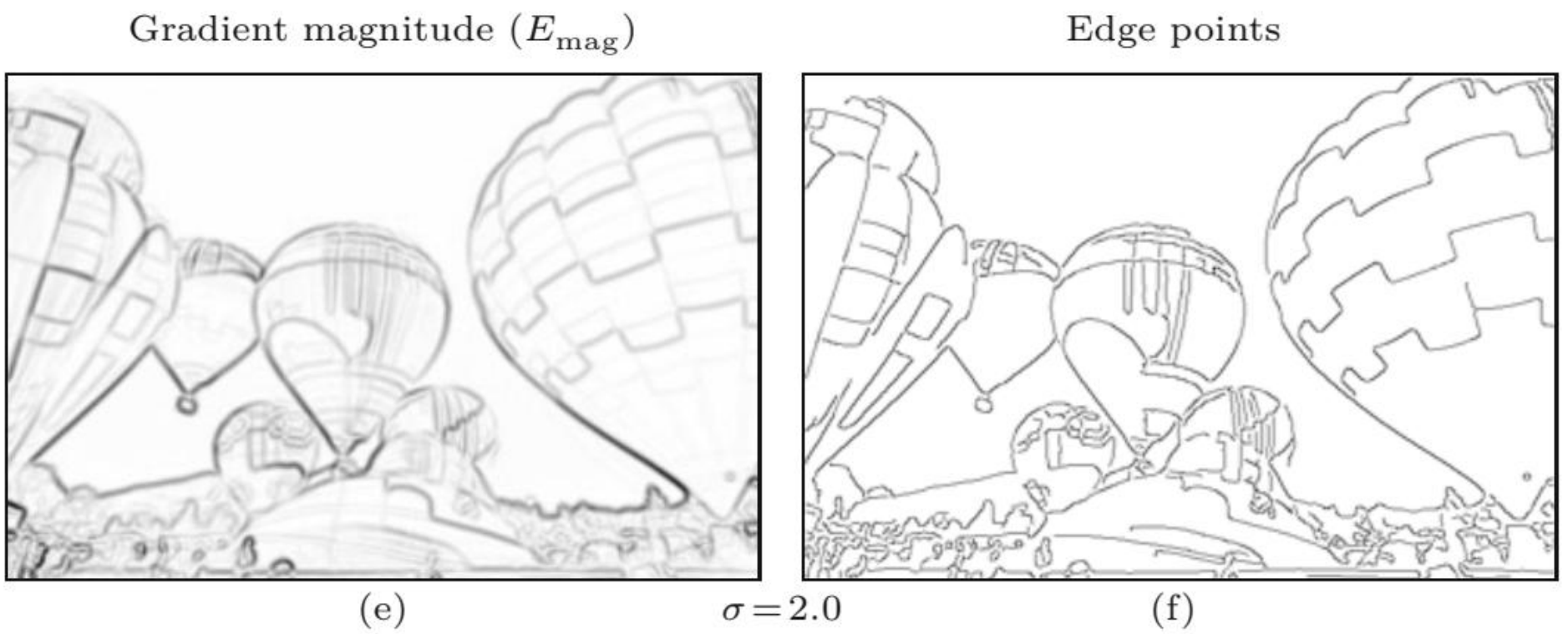
norm of the gradient

after NMS

after H. Thresholding

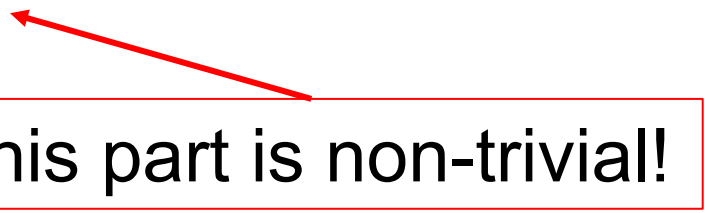
Canny – effect of sigma

Canny – effect of sigma



Edge-based segmentation

- Edge-based methods center around contour detection
- General workflow
  1. Detect edges, i.e., mark each pixel as ”edge” or ”not edge”.
  1. Divide the image into regions, based on the detected edges. (Edge linking, Hough transform)



- Weakness in connecting broken contour lines make them prone to failure in the presence of blurring.

An edge is not a line...

How can we detect ***lines*** ?

Finding lines in an image

- Option 1:
  - Search for the line at every possible position/orientation
  - What is the cost of this operation?

- Option 2:
  - Use a voting scheme: **Hough transform**

Hough Transform

- Performed after Edge Detection
- It is a technique to isolate the curves of a given shape / shapes in a given image
- Classical Hough Transform can locate regular curves like straight lines, circles, parabolas, ellipses, etc.
  - Requires that the curve be specified in some parametric form
- Generalized Hough Transform can be used where a simple analytic description of feature is not possible

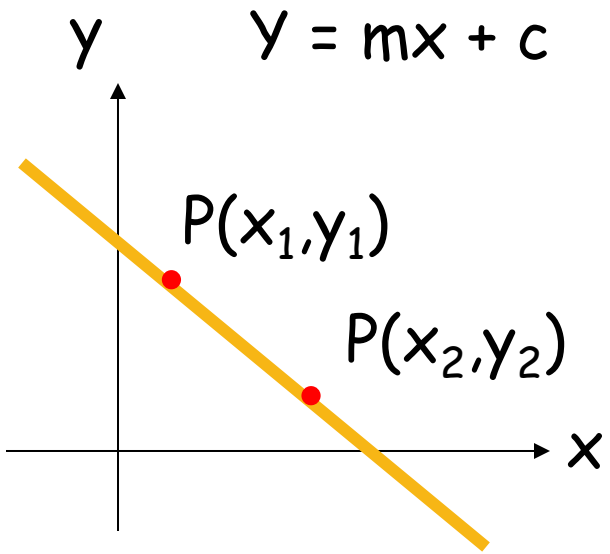
Hough Transform

Advantages

- The Hough Transform is tolerant of gaps in the edges
- It is relatively unaffected by noise
- It is also unaffected by occlusion in the image

Hough Transform

- Mathematical model of a line:



$Y_1 = m\,x_1 + c$

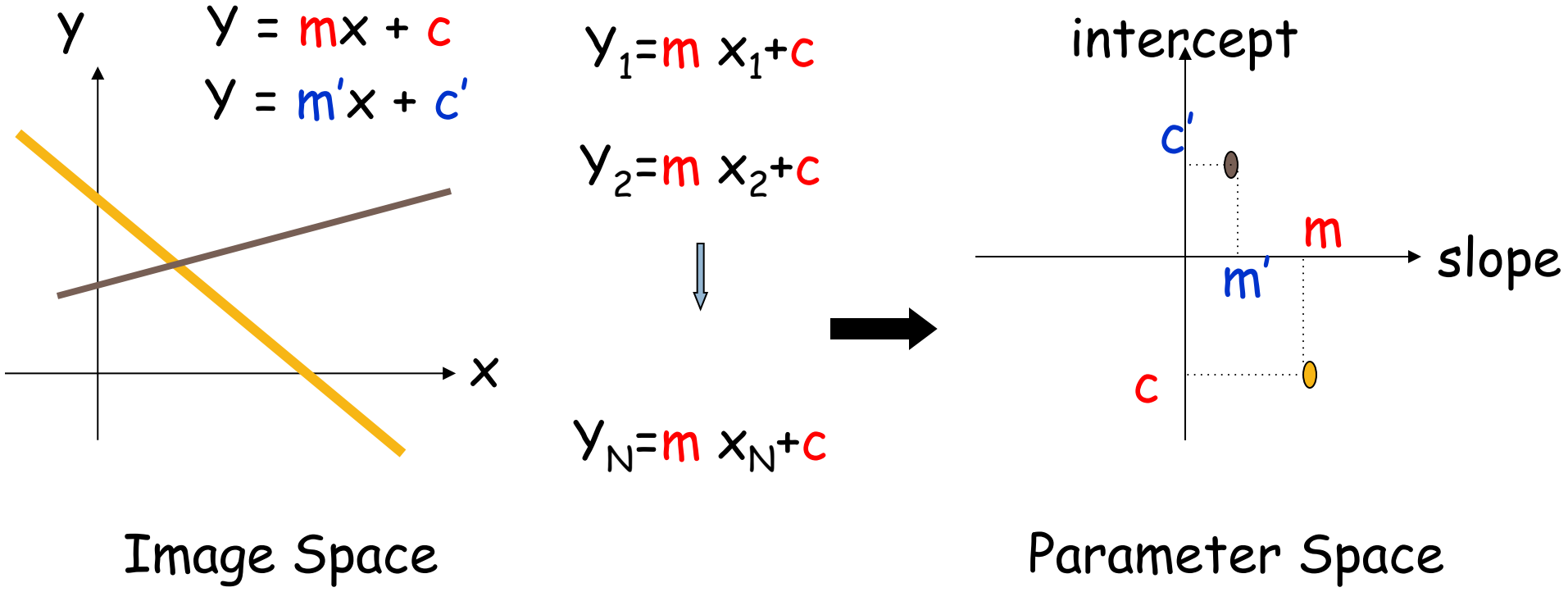
$Y_2 = m\,x_2 + c$

$\Downarrow$

$Y_N = m\,x_N + c$

Hough Transform

- **Image and Parameter Spaces**



Line in Img. Space ~ Point in Param. Space

Hough Transform

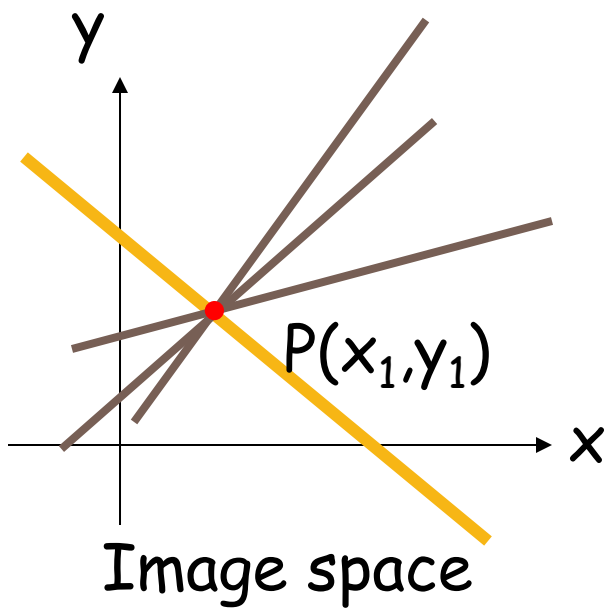
- **Image and Parameter Spaces**

Image space

$y = mx + c$

Fix (m,c), Vary (x,y) - Line

Fix (x_1,y_1), Vary (m,c) – Lines thru a Point



$y_1 = m_1 x_1 + c_1$

$y_1 = m_2 x_1 + c_2$

:

$y_1 = m_n x_1 + c_n$

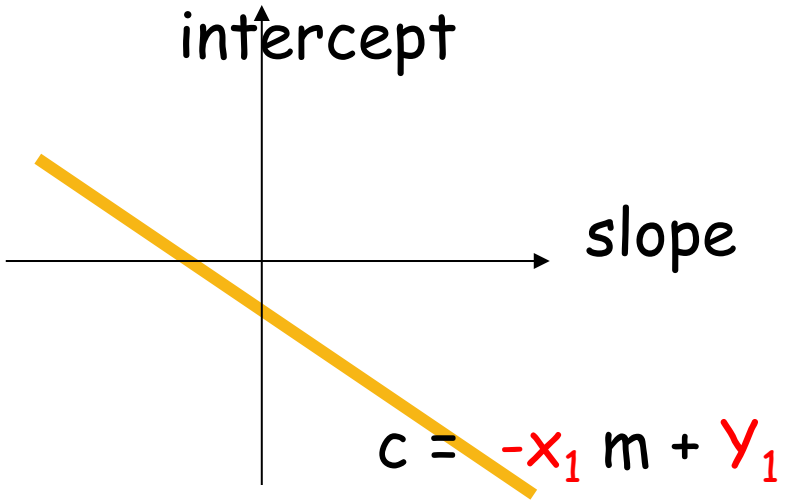
Hough Transform

- **Image and Parameter Spaces**

  Parameter space

  $y_1 = m\,x_1 + c$ Can be re-written as: $c = -x_1\,m + y_1$

Fix $(-x_1, y_1)$, Vary (m,c) - Line

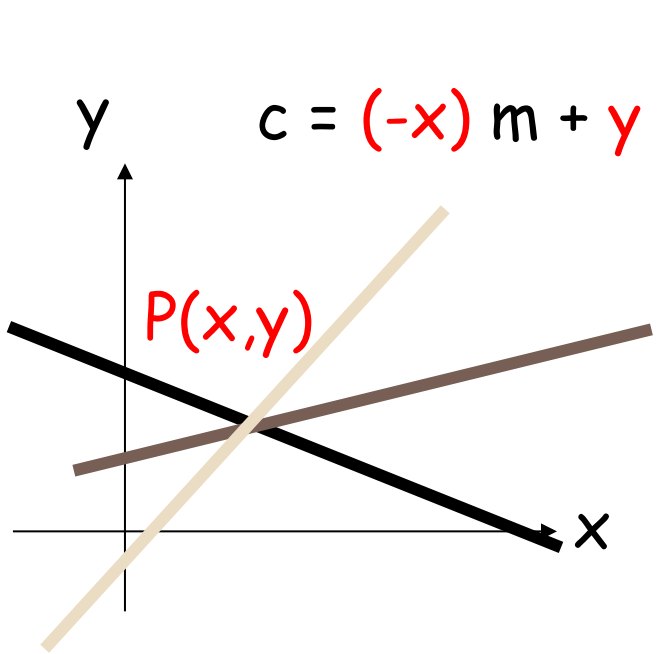


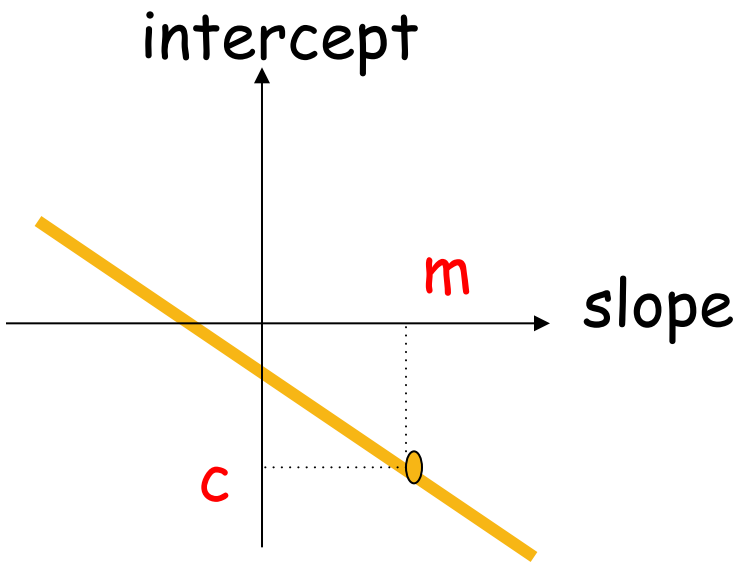
Parameter Space

Hough Transform

Image and Parameter Spaces

- Given an edge point / pixel, there is an infinite number of lines passing through it (Vary m and c).
  - These lines can be represented as a single line in parameter space.

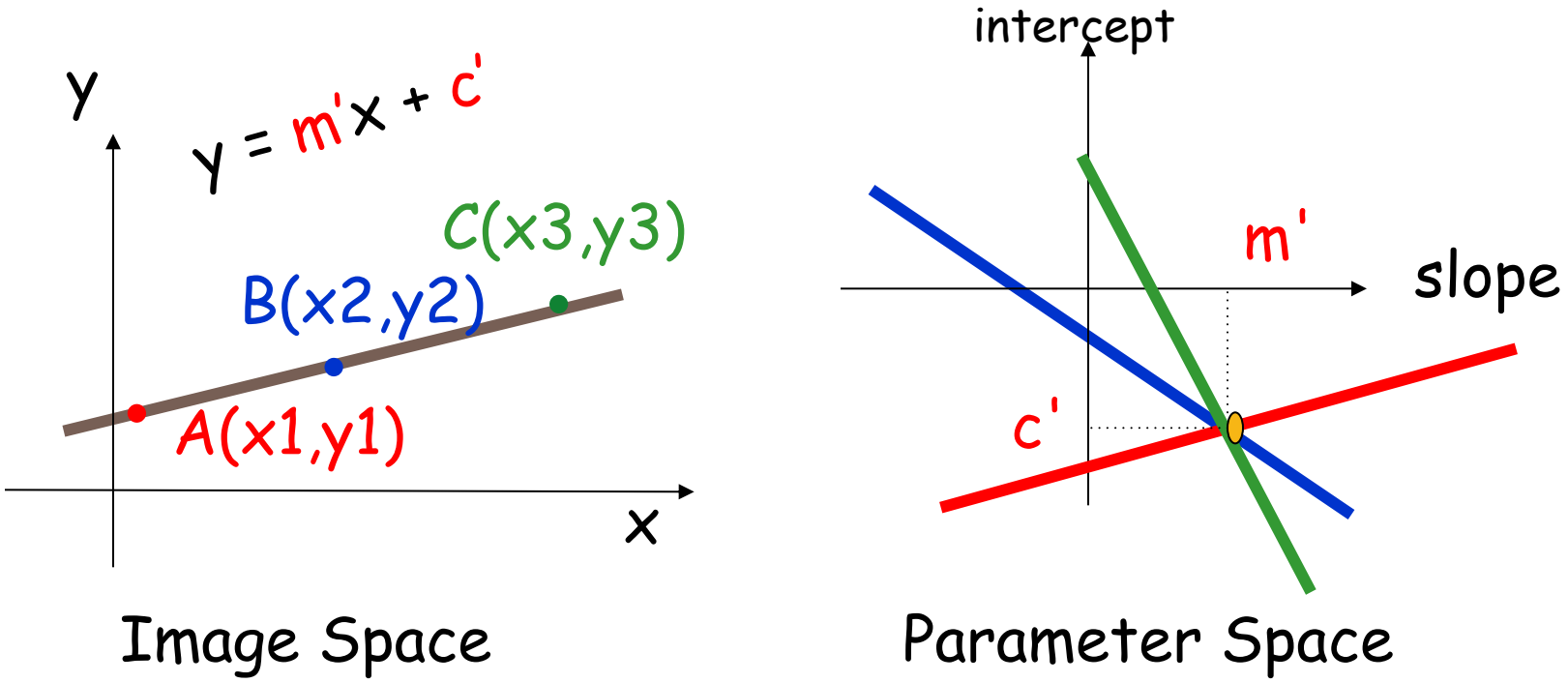




Parameter Space

Hough Transform

- **Image and Parameter Spaces**
  - Given a set of collinear edge points, each of them have associated a line in parameter space.
    - These lines intersect at the point (m',c') corresponding to the parameters of the line in the image space.



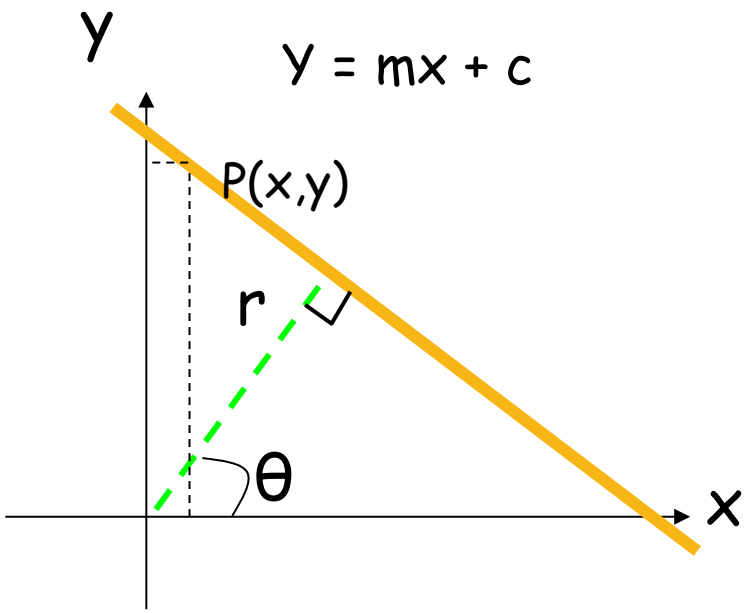
Hough Transform

Image and Parameter Spaces

- Image Space
  - Lines
  - Points
  - Collinear points
- Parameter Space
  - Points
  - Lines
  - Intersecting lines

Hough Parameterization

- **Practical Issues**
  - The slope of the line is $-\infty<m<\infty$, parameter space is INFINITE
  - The representation $y = mx + c$ does not express lines of the form $x = k$
- **Solution:** Use the "Normal"/ polar equation of a line:

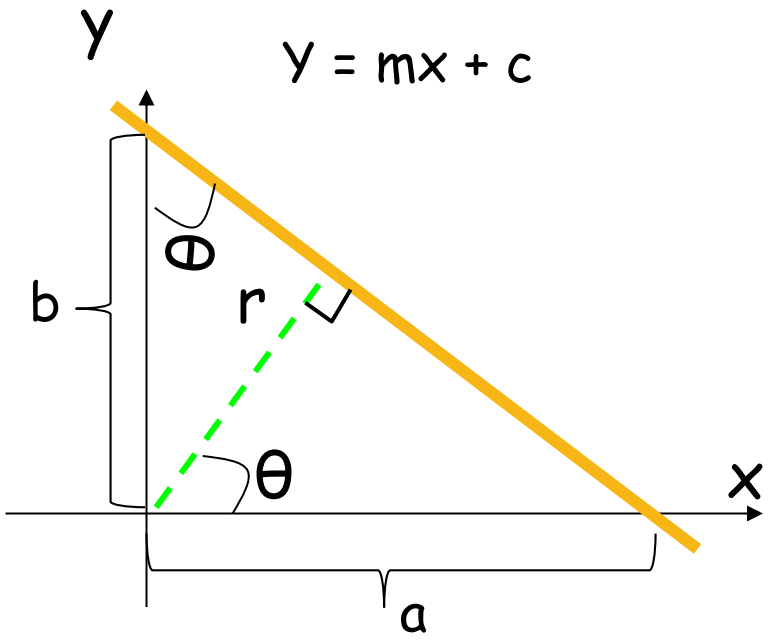


$r = x \cos\theta + y \sin\theta$

θ Is the line orientation

r Is the distance between the origin and the line

Hough Parameterization



$$\frac{x}{a} + \frac{y}{b} = 1$$

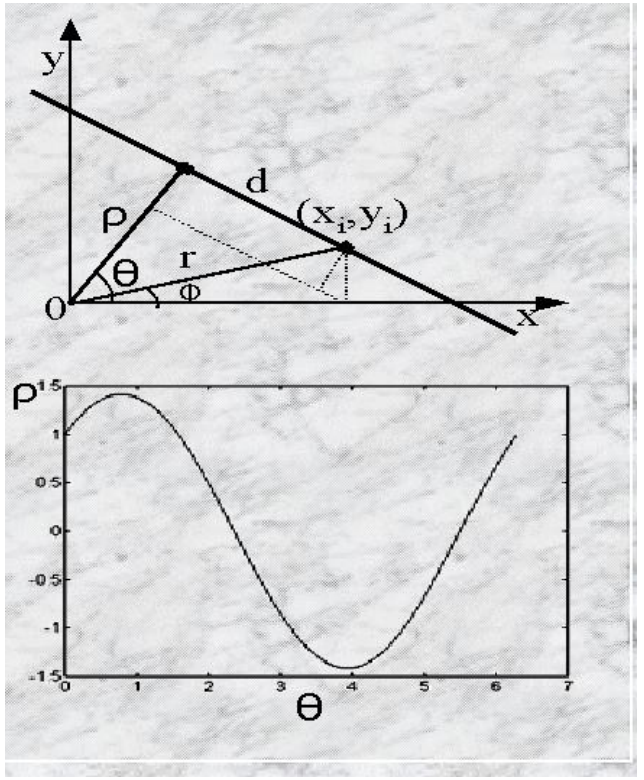
r = x cosθ+y sinθ

$$\frac{r}{a} = \cos\theta; \quad a = \cos\theta / r$$

$$\frac{r}{b} = \sin\theta; \quad b = \sin\theta / r$$

Consequence:

- A Point P(x, y) in Image Space is now represented as a SINUSOID in the parameter space
  - $r(\theta) = x \cos\theta + y \sin\theta$
- Use the parameter space (r, θ)
- The new space is FINITE
  - $0 < r < D$, where D is the image diagonal.
  - $0 < \theta < \pi$
- The new space can represent all lines
  - Y = k is represented with r = k, θ=90
  - X = k is represented with r = k, θ=0



r,θ space

- In (slope, intercept) space
  - point in image space == line in (m,c) space

- In (r,θ) space
  - point in image space == sinusoid in (r,θ) space
  - where sinusoids overlap, accumulator = max
  - maxima still = lines in image space

- Practically, finding maxima in accumulator is non-trivial
  - often smooth the accumulator for better results

Hough transform

- An early type of voting scheme
- General outline:
  - Discretize parameter space into bins
  - For each feature point in the image, put a vote in every bin in the parameter space that could have generated this point
  - Find bins that have the most votes

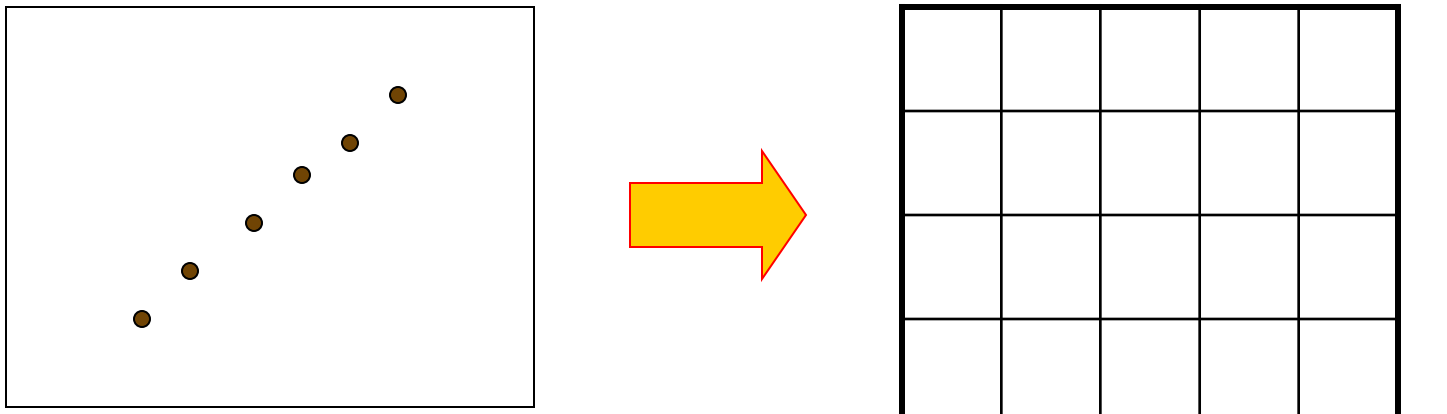
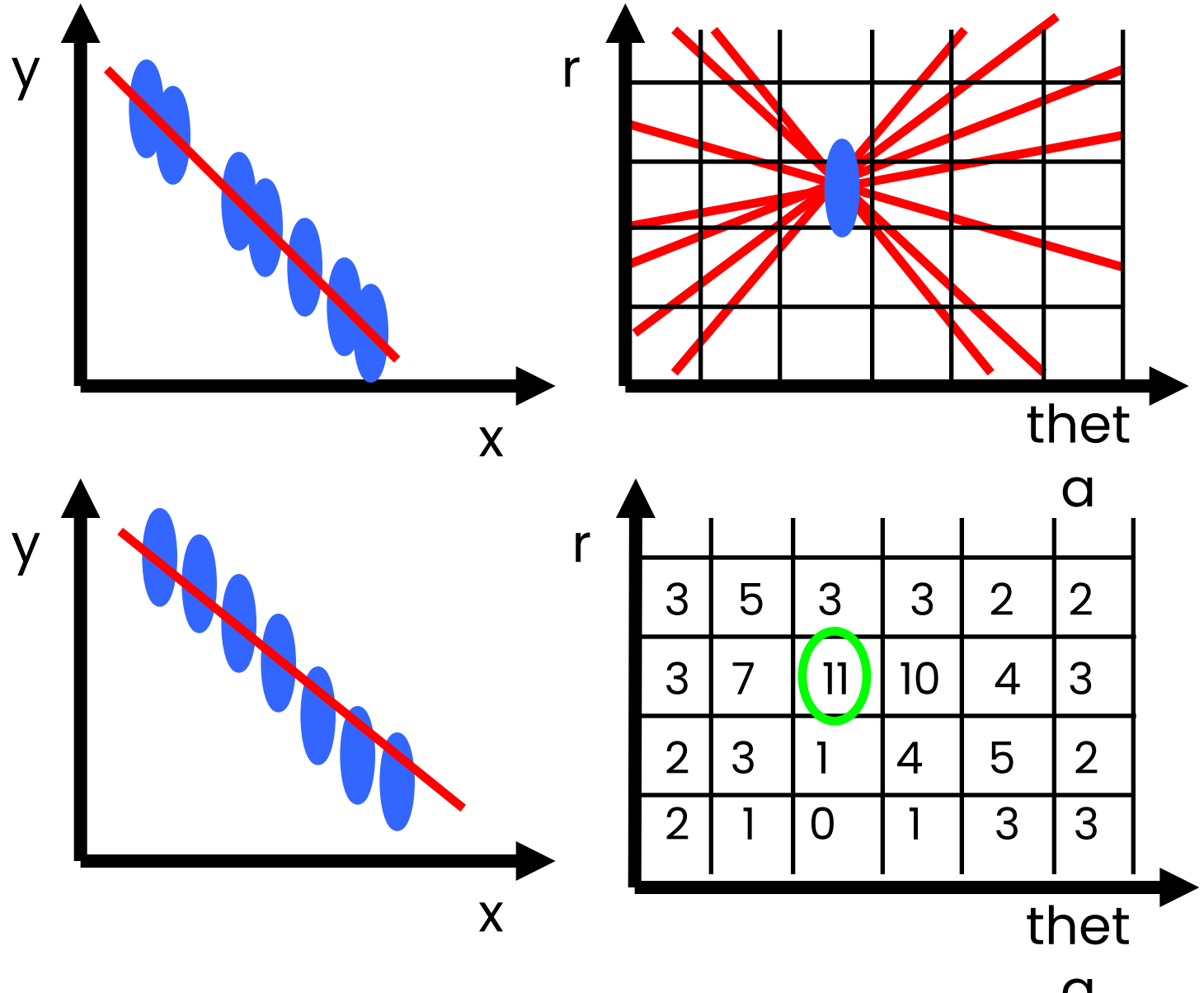


Image space

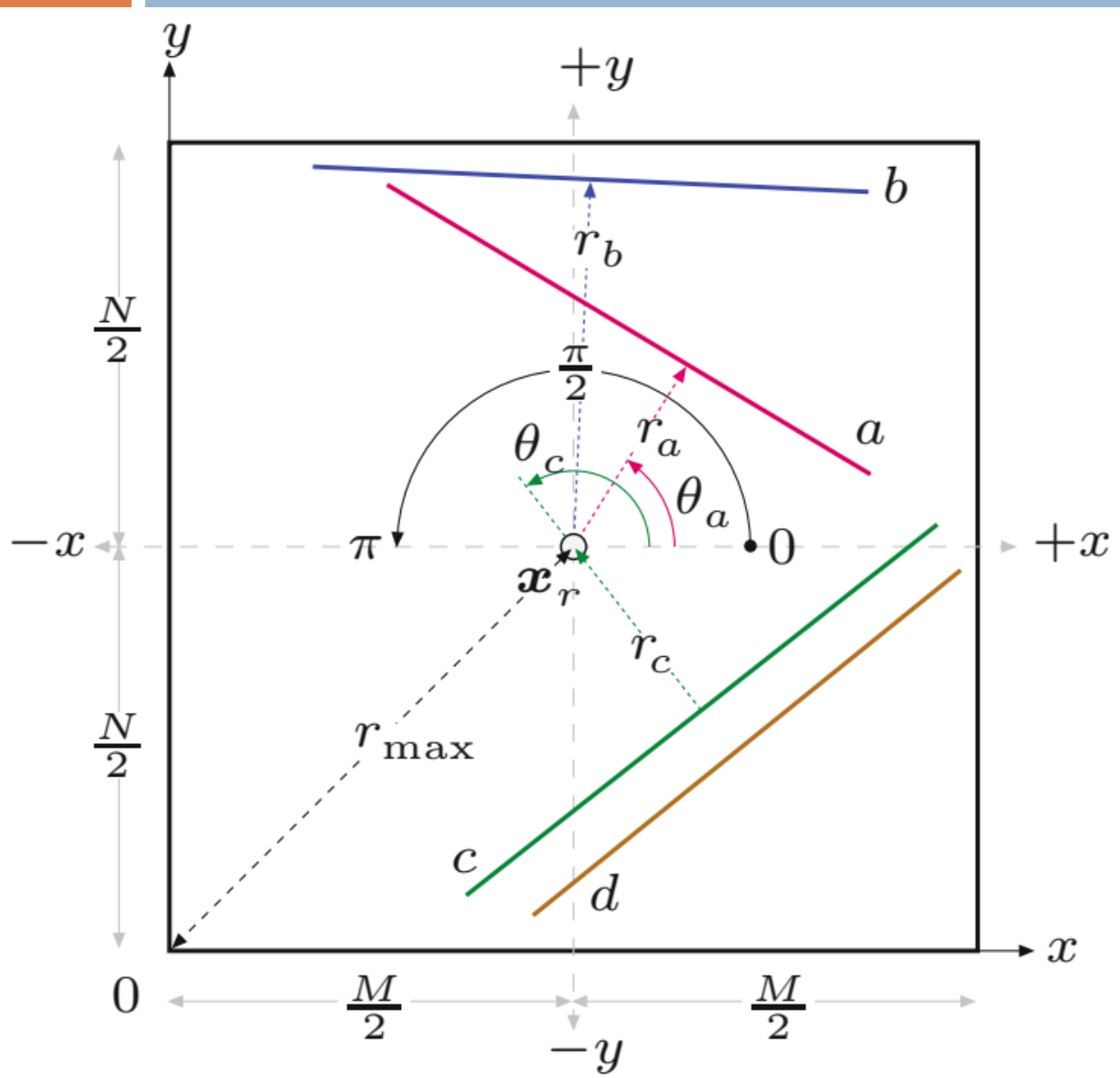
Hough parameter space

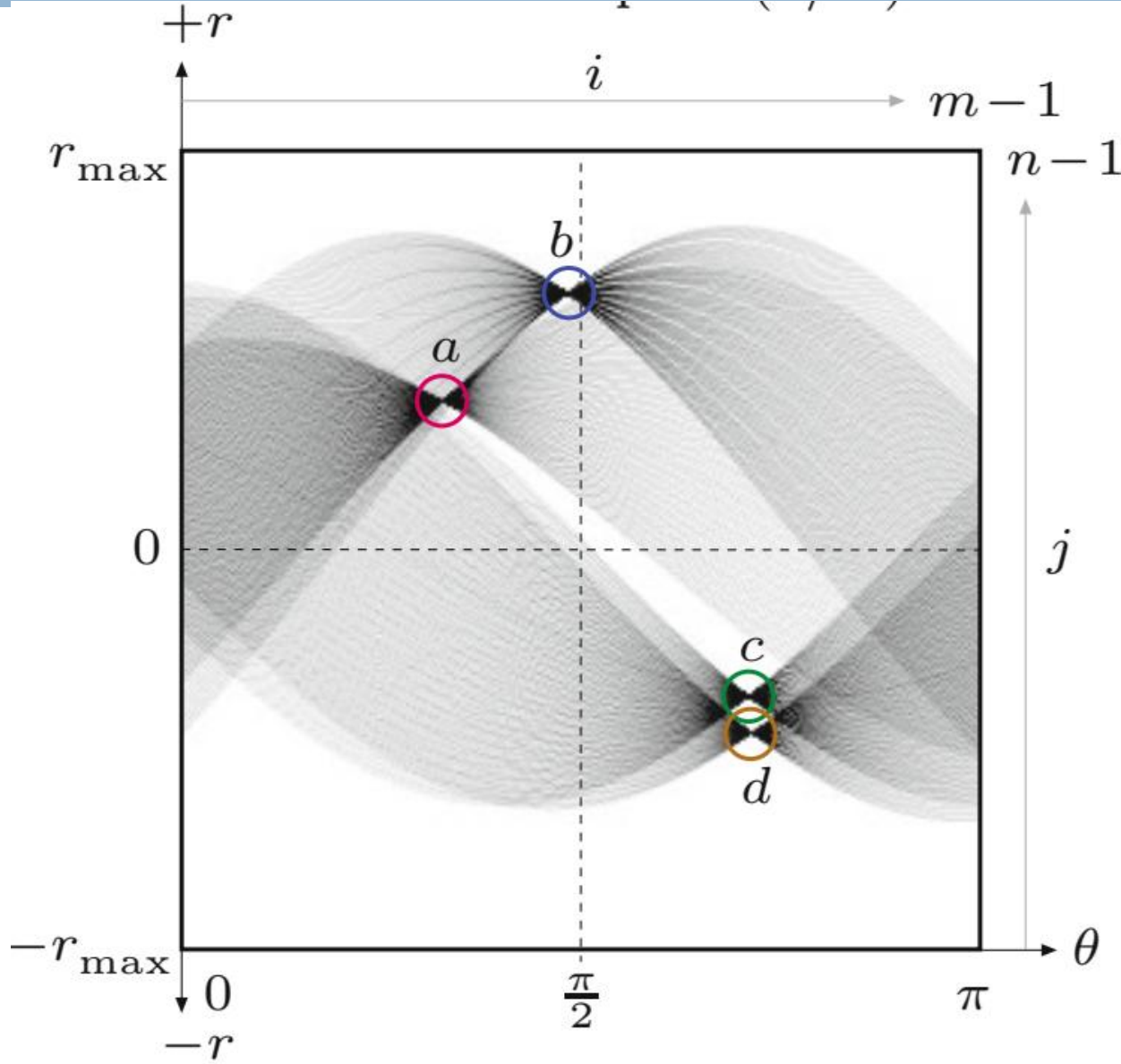
P.V.C. Hough, *Machine Analysis of Bubble Chamber Pictures,* Proc. Int. Conf. High Energy Accelerators and Instrumentation, 1959

Hough transform



Hough Transform Algorithm





Hough Transform Algorithm

$$\boldsymbol{x}_r = \begin{pmatrix} x_r \\ y_r \end{pmatrix} = \frac{1}{2} \cdot \begin{pmatrix} M \\ N \end{pmatrix}$$

$$-r_{\max} \le r(\theta) \le r_{\max}, \qquad \text{with} \qquad r_{\max} = \tfrac{1}{2}\sqrt{M^2 + N^2}$$

$$d_\theta = \pi / m \qquad \text{and} \qquad d_r = \sqrt{M^2 + N^2}/n$$

$$j_0 = n \div 2$$

For each relevant image point (u, v)

$$r(\theta_i) = (u - x_r) \cdot \cos(\theta_i) + (v - y_r) \cdot \sin(\theta_i)$$

$$\theta_i = \theta_0, \ldots, \theta_{m-1} = i \cdot d_\theta$$

$$j = j_0 + \text{round}\left(\frac{r(\theta_i)}{d_r}\right)$$

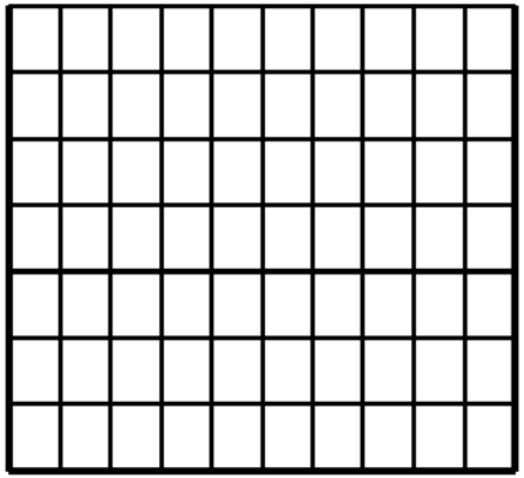
Increment accumulator cell A(i,j) by one

The line parameters θ_i and r_j for a given accumulator position (*i, j*) can be calculated as

$$\theta_i = i \cdot d_\theta \qquad \text{and} \qquad r_j = (j - j_0) \cdot d_r$$

Hough Transform Algorithm

A : accumulator array (votes)



m×
n

HoughTransformLines$(I, m, n, a_{\min})$

Input: I, a binary image of size $M \times N$; m, angular accumulator steps; n, radial accumulator steps; $a_{\min}$, minimum accumulator count per line. Returns a sorted sequence $\mathcal{L} = (L_1, L_2, \ldots)$ of the most dominant lines found.

$(M, N) \leftarrow \mathsf{Size}(I)$

$(x_r, y_r) \leftarrow \frac{1}{2} \cdot (M, N)$ ▷ reference point $\boldsymbol{x}_r$ (image center)

$d_\theta \leftarrow \pi / m$ ▷ angular step size

$d_r \leftarrow \sqrt{M^2 + N^2}/n$ ▷ radial step size

$j_0 \leftarrow n \div 2$ ▷ map index for $r = 0$

Step 1 – set up and fill the Hough accumulator:

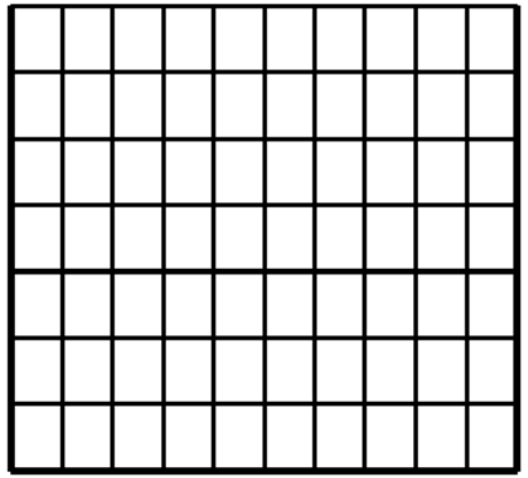
Create map $\mathsf{A}: [0, m-1] \times [0, n-1] \mapsto \mathbb{Z}$ ▷ accumulator

for all accumulator cells (i, j) **do**

$\mathsf{A}(i, j) \leftarrow 0$ ▷ initialize accumulator

Hough Transform Algorithm

A : accumulator array (votes)

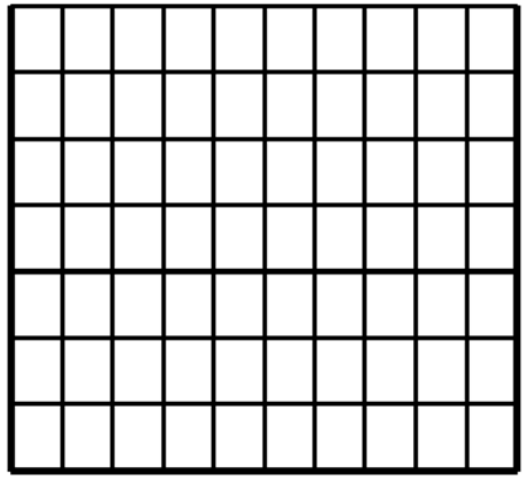


m×
n

```
10: for all (u,v) ∈ M×N do                          ▷ scan the ima
11:     if I(u,v) > 0 then                          ▷ I(u,v) is a foreground pi
12:         (x,y) ← (u−x_r, v−y_r)                  ▷ shift to referen
13:         for i ← 0,...,m−1 do                    ▷ angular coordinat
14:             θ ← d_θ · i                         ▷ angle, 0 ≤ θ <
15:             r ← x·cos(θ) + y·sin(θ)             ▷ see Eqn. 8
16:             j ← j_0 + round(r/d_r)              ▷ radial coordinate
17:             A(i,j) ← A(i,j) + 1                 ▷ increment A(i,
```

Hough Transform Algorithm

A : accumulator array (votes)



m×
n

Step 2 – extract the most dominant lines:

18: $\mathcal{L} \leftarrow ()$ ▷ start with empty sequence of li
19: **for all** accumulator cells (i, j) **do** ▷ collect local maxi
20: **if** $(\mathsf{A}(i, j) \geq a_{\min}) \wedge \mathsf{IsLocalMax}(\mathsf{A}, i, j)$ **then**
21: $\theta \leftarrow i \cdot d_\theta$ ▷ angl
22: $r \leftarrow (j - j_0) \cdot d_r$ ▷ radiu
23: $a \leftarrow \mathsf{A}(i, j)$ ▷ accumulated valu
24: $L \leftarrow \langle \theta, r, a \rangle$ ▷ create a new line
25: $\mathcal{L} \leftarrow \mathcal{L} \frown (L)$ ▷ add line L to sequence
26: $\mathsf{Sort}(\mathcal{L})$ ▷ sort $\mathcal{L}$ by descending accumulator coun
27: **return** $\mathcal{L}$

Hough Transform Algorithm

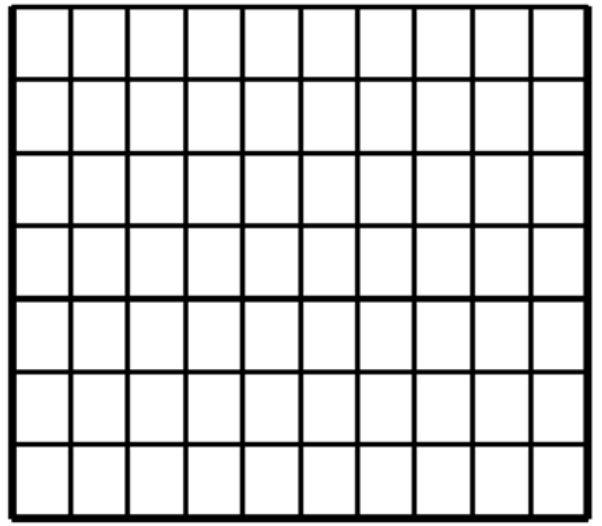
Input: edge image (E(x,y)=1 for edgels)

1. Discretize r (<D) in increments of dρ

   Discretize θ (<π) in increments of dθ (total# T) , and

   Let A(R,T) be an accumulator array, initialized to 0.

A : accumulator array (votes)



2. For each pixel E(x,y)=1 and For t=1,2,…T do
   i. θ =t * dθ
   ii. ρ = x cos(θ) + y sin(θ)
   iii. Find closest integer r corresponding to ρ
   iv. Increment counter A(r,t) by one i.e. A(r,t) = A(r,t) +1

Hough Transform Algorithm

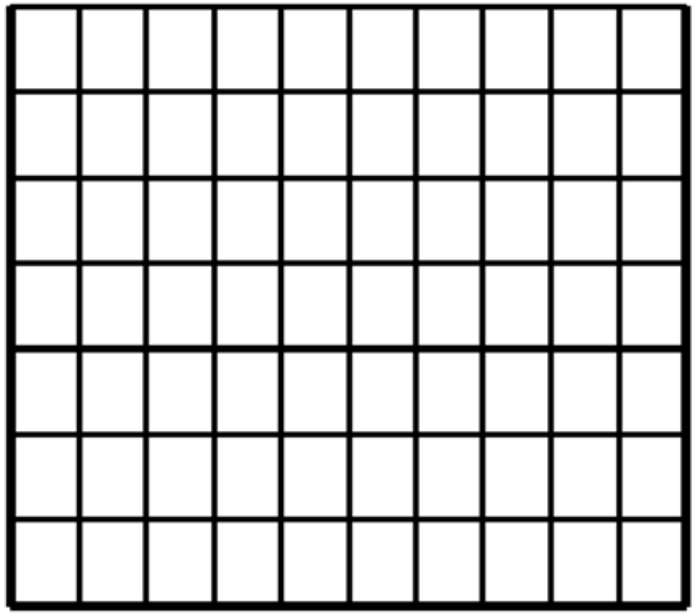
3. Compute the value(s) of (r, θ) by finding A[r, t] is maximum, where θ = t * dθ

4. The detected line in the image space is given by

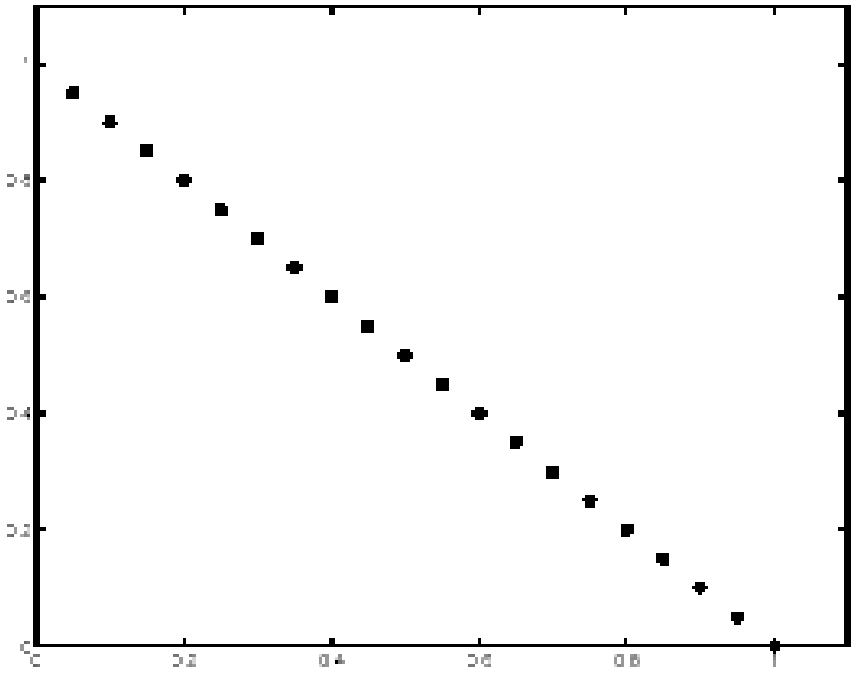
$$r = x \cos\theta + y \sin\theta$$

They are now constant

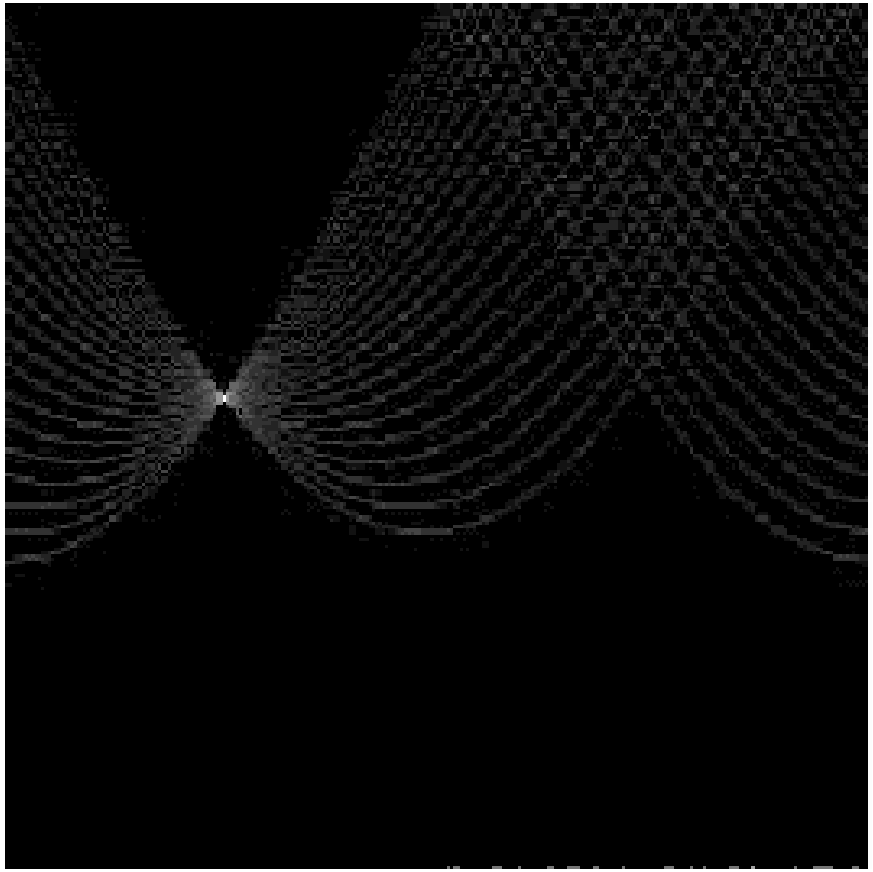
A : accumulator array (votes)



Hough Transform - Basic illustration

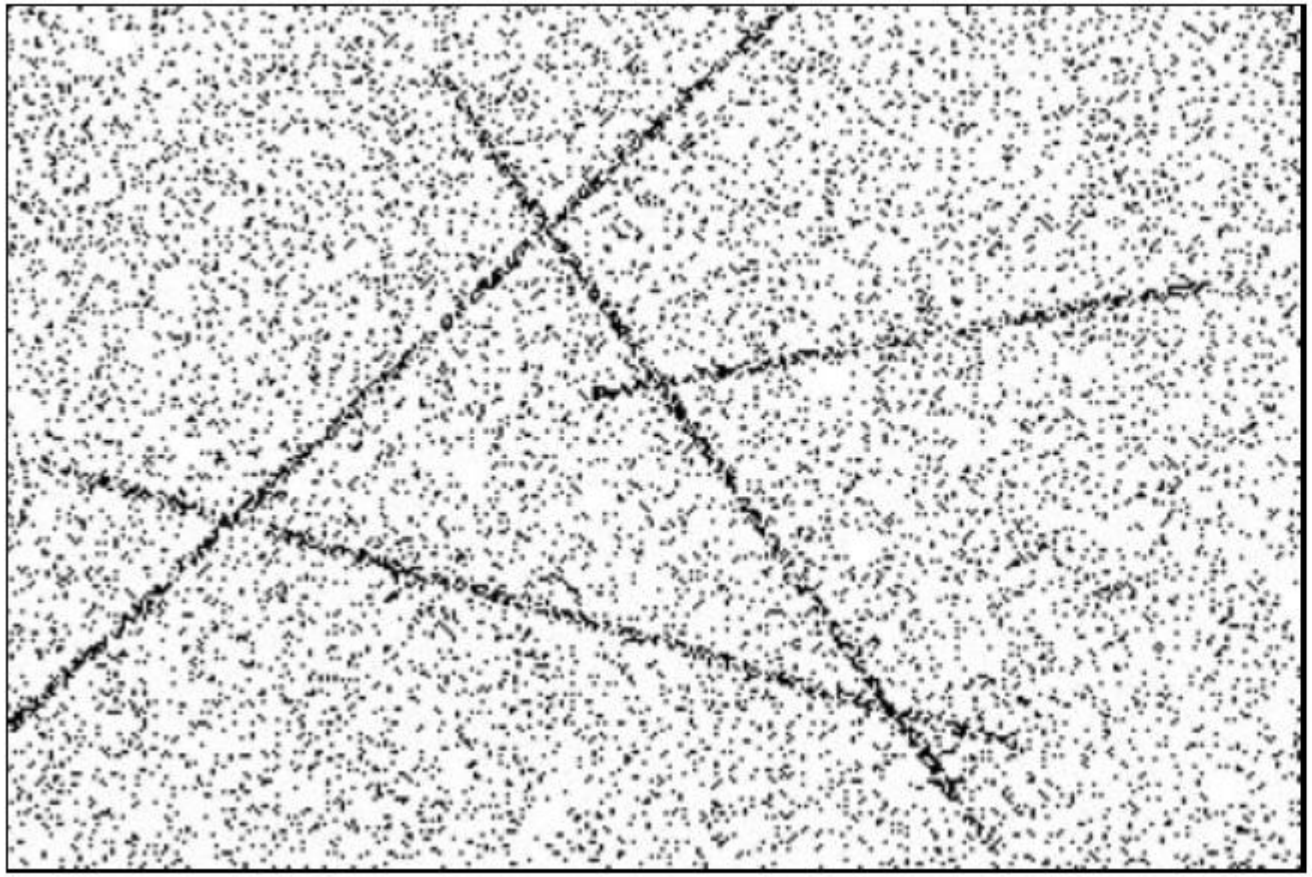


features

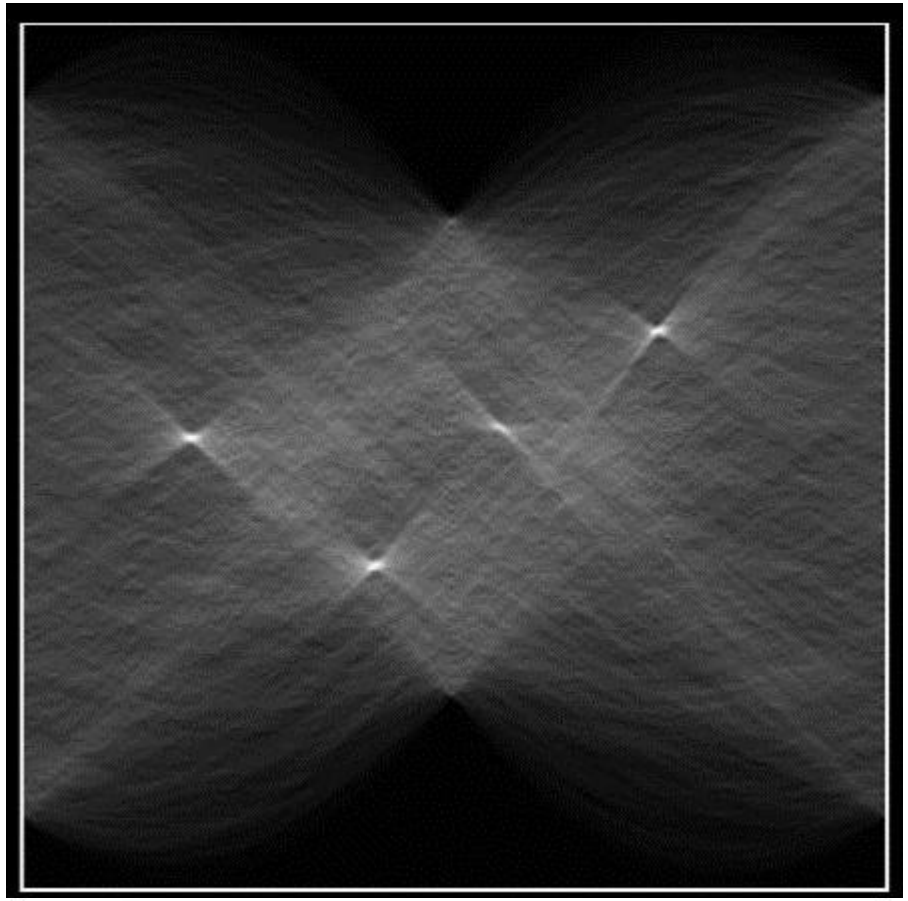


votes

Hough Transform - Basic illustration

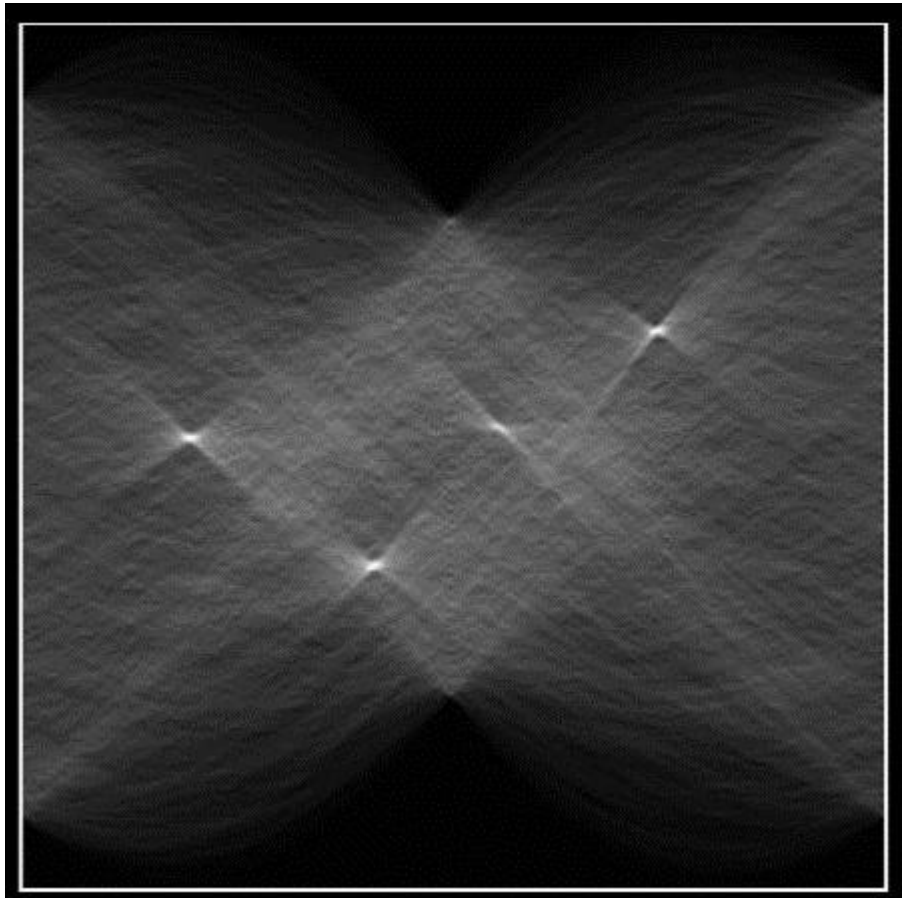


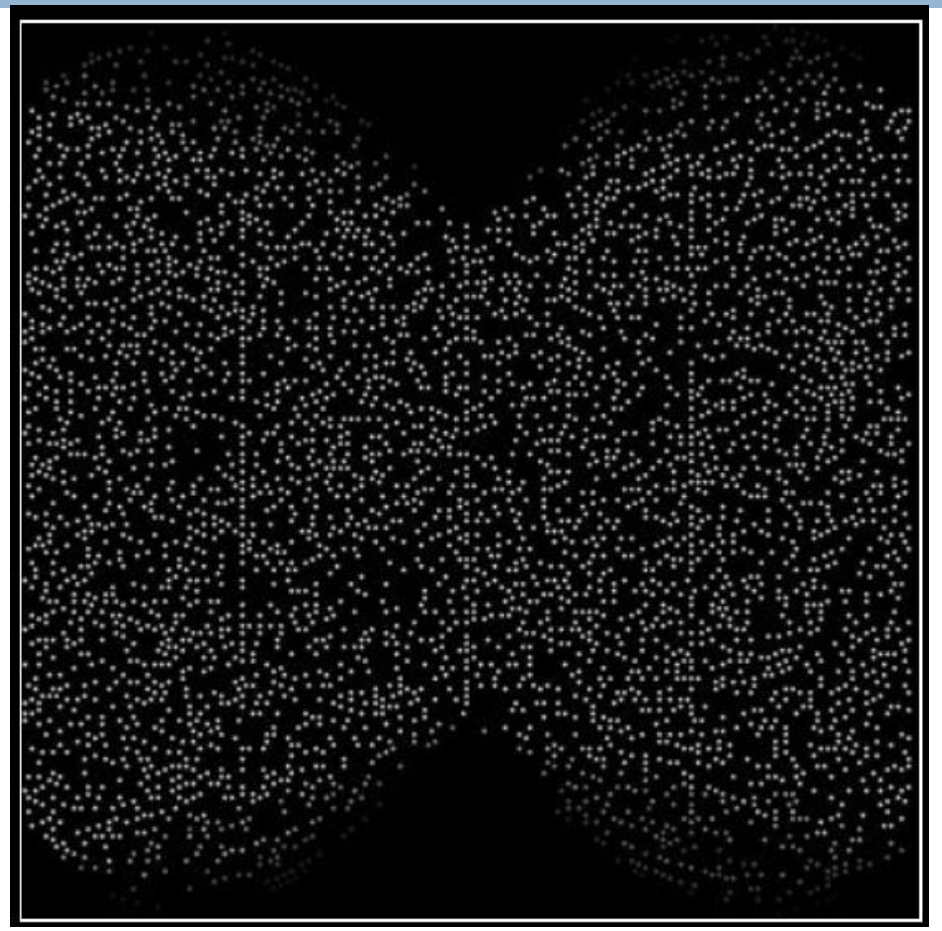
features



votes

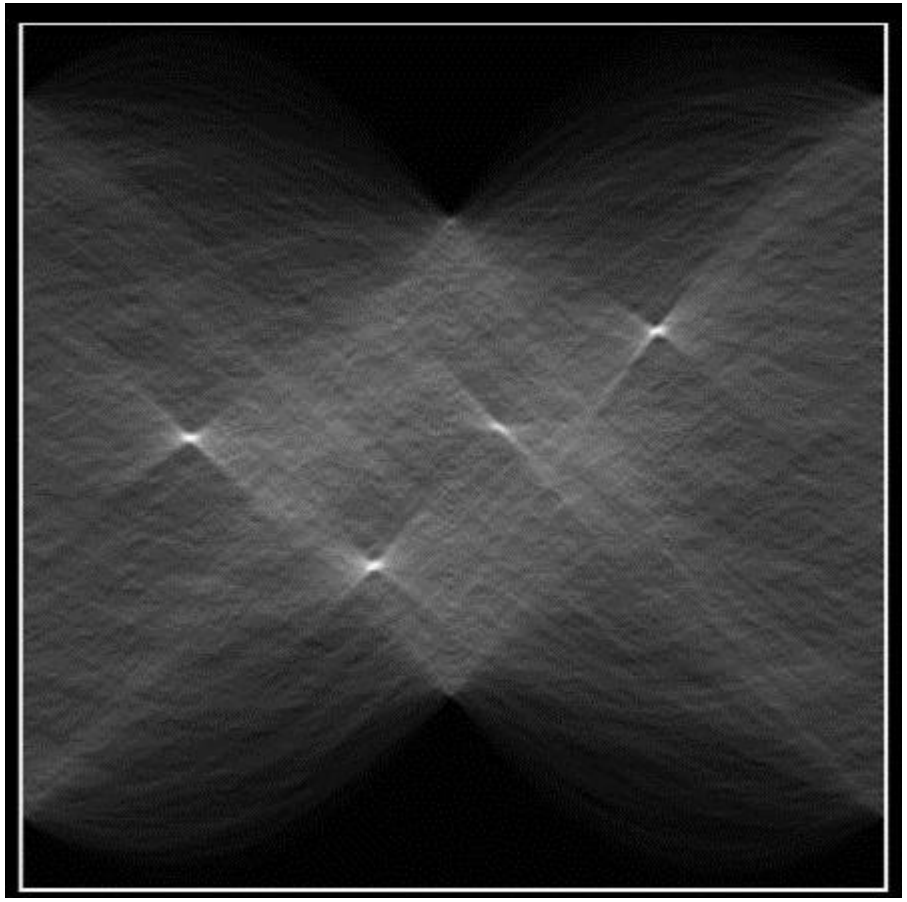
Hough Transform - issues

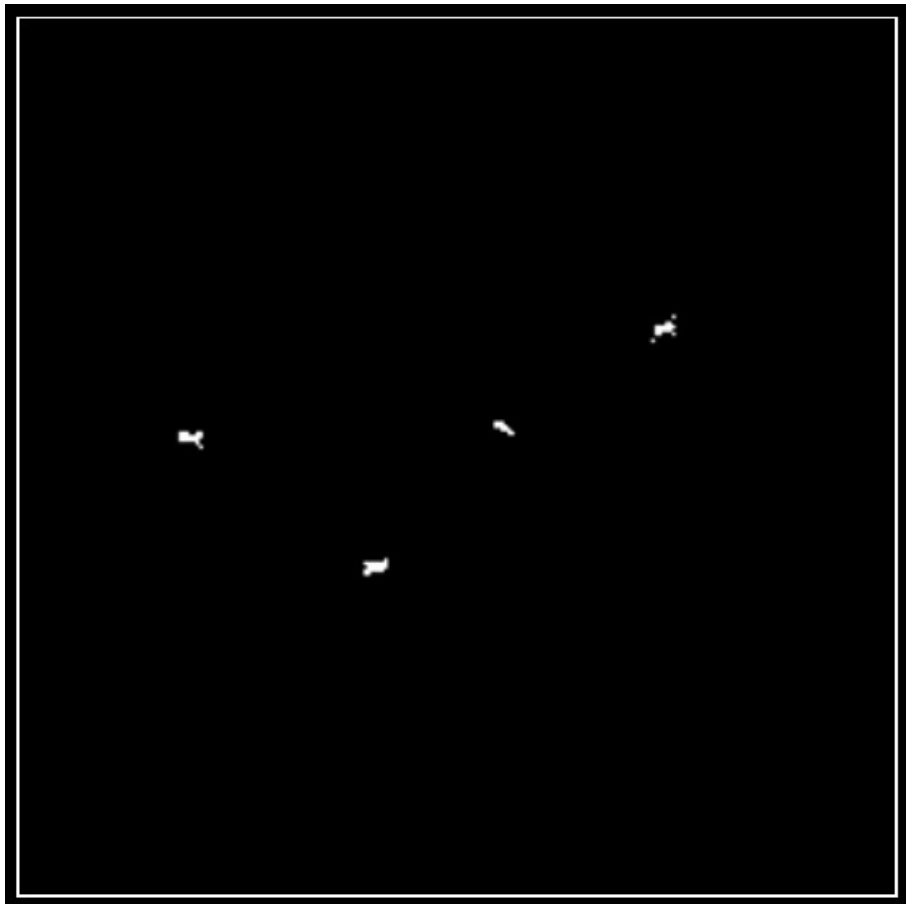




Using only non max suppression

Hough Transform - issues

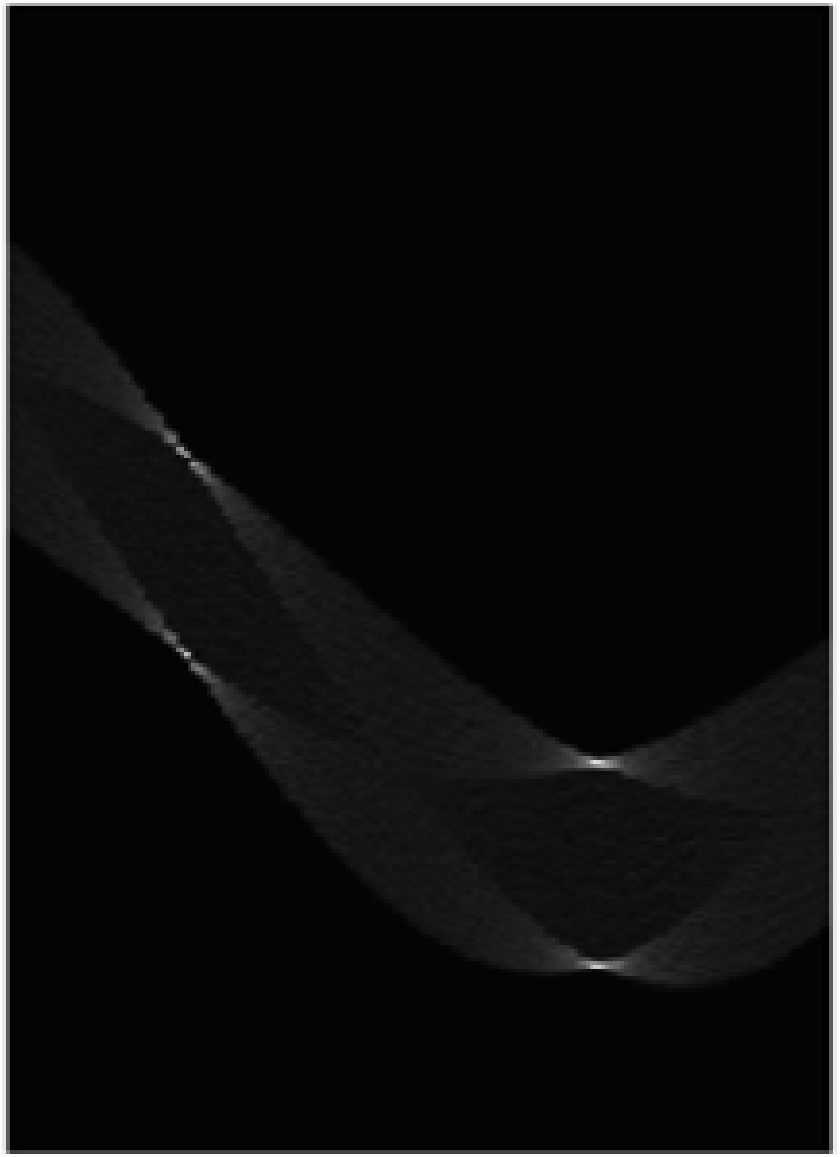




Threshold operation using 50% of the maximum value

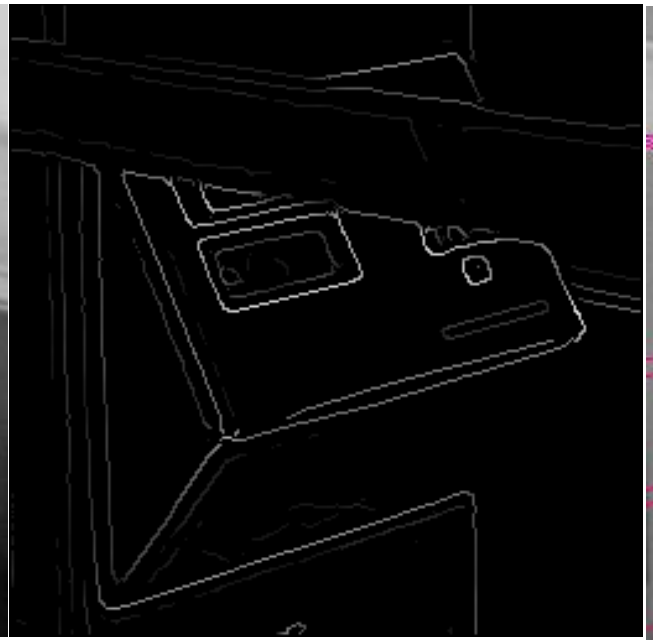
Hough Transform- Other shapes

Square

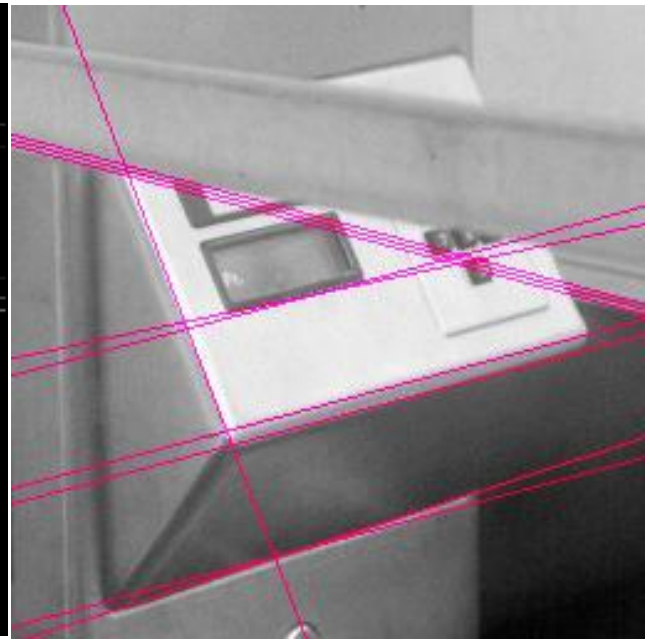


Hough Transform- Real World Example

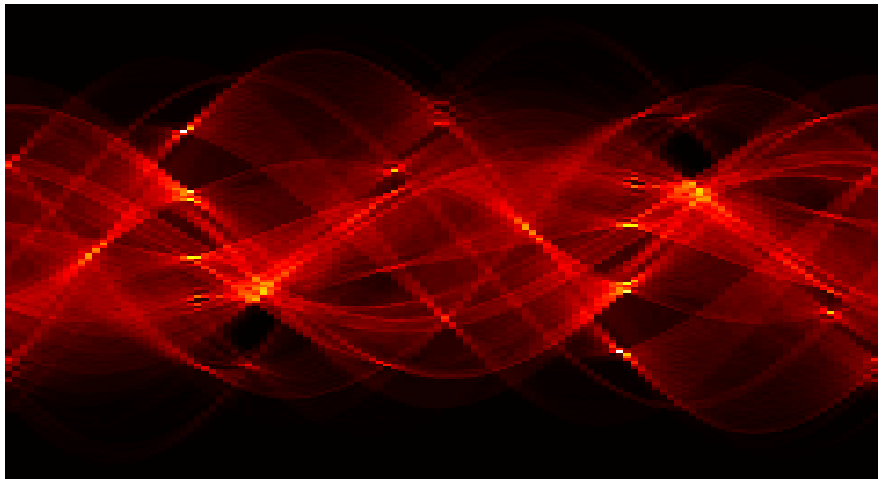
Original



Edge Detection



Found Lines



Parameter Space

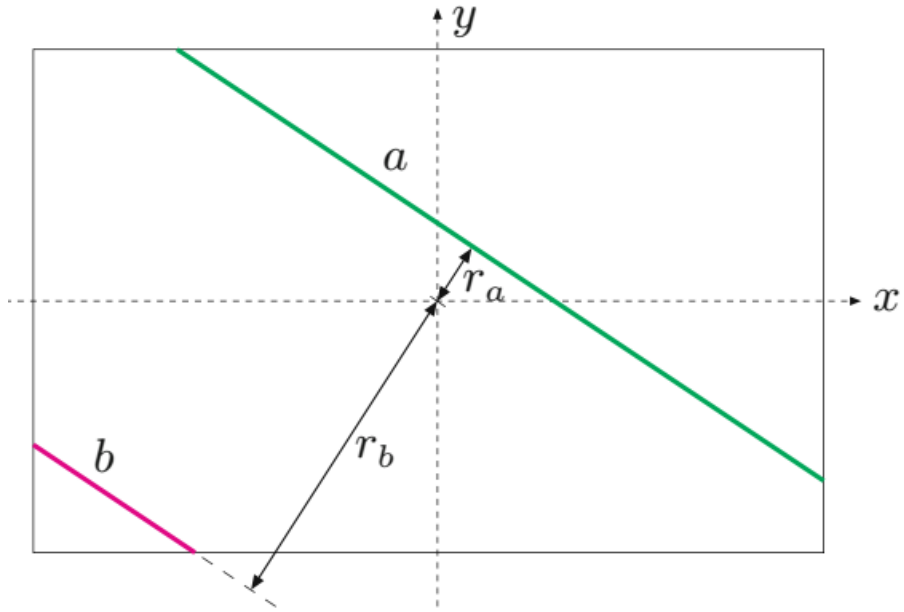
Hough Transform - Speed Up

- If we know the orientation of the edge – usually available from the edge detection step
  - We fix theta in the parameter space and increment **only one** counter!
  - We can allow for orientation uncertainty by incrementing a *few* counters around the "nominal" counter.

Hough Transform - Bias compensation

- if we only search the accumulator array for maximal values, it is likely that we will completely miss short line segments

$$\mathsf{A}(i,j) \leftarrow \frac{\mathsf{A}(i,j)}{\max(1, \mathsf{A}_{\max}(i,j))}$$



$A_{max}(i, j)$ = maximum number of image points possible for a line with the corresponding parameters

Hough Transform - Line endpoints

- For this, every cell of the accumulator array is supplemented with two additional coordinate pairs

$$\boldsymbol{x}_s = (x_s, y_s),\ \boldsymbol{x}_e = (x_e, y_e)$$

$$\mathsf{A}(i, j) = \langle count, \boldsymbol{x}_s, \boldsymbol{x}_e \rangle$$

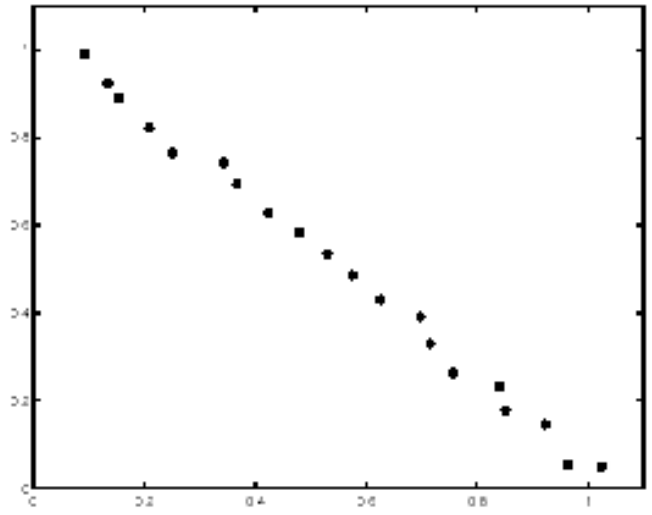
Hough Transform – Modified accumulation

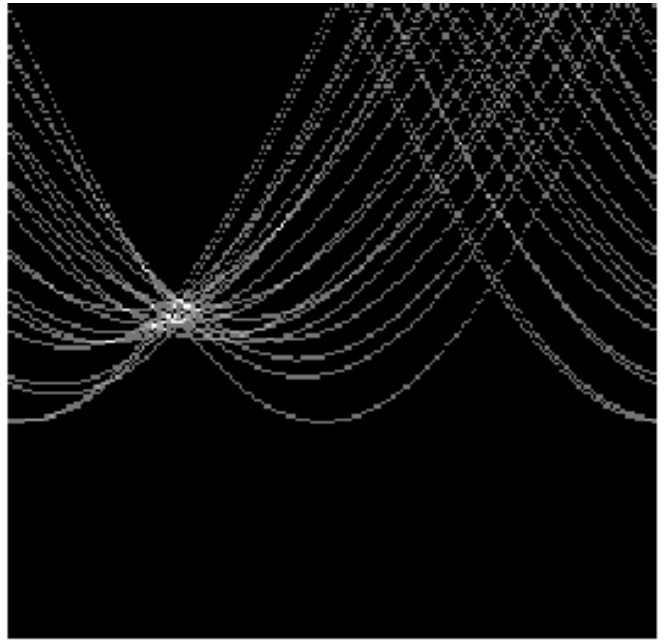
- Due to the discrete nature of the image and accumulator coordinates,
  - rounding errors usually cause the parameter curves not to intersect in a single accumulator cell, even when the associated image lines are exactly straight

- for a given angle $\theta = i \cdot d\theta$, increment
  - not only the main accumulator cell A(i, j)
  - but also the neighboring cells A(i, j−1) and A(i, j + 1), with different weights.

Hough Transform- Effect of noise

- Peak gets fuzzy and hard to locate

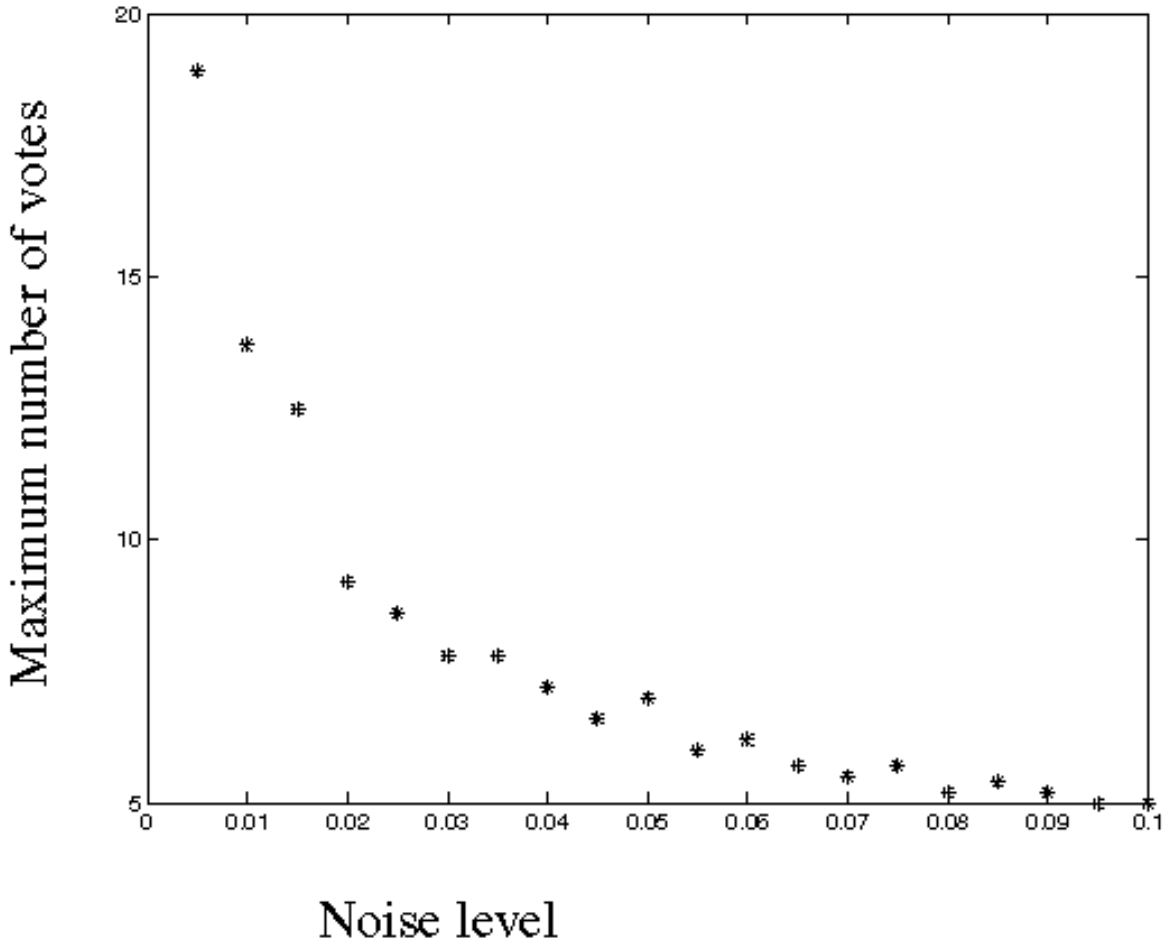




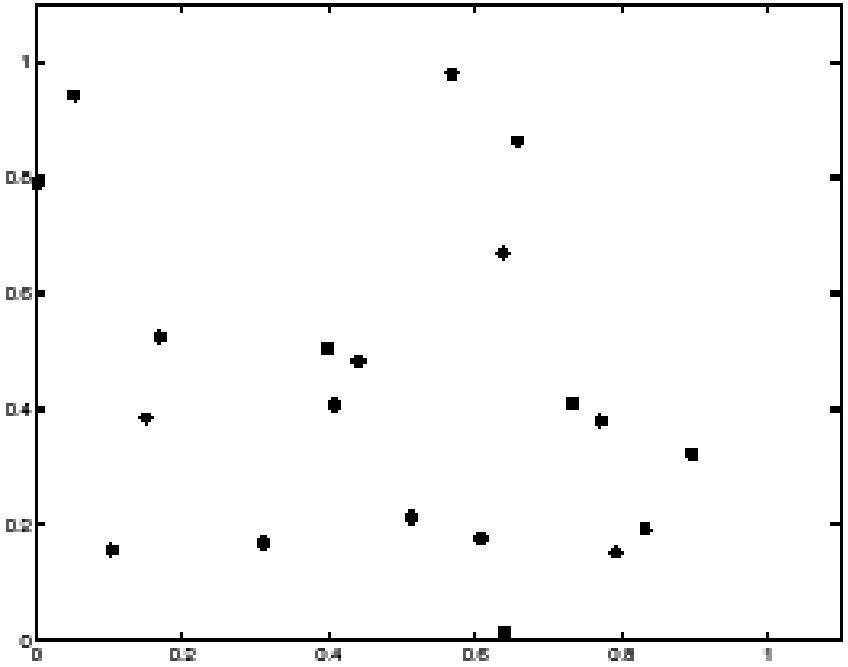
- The resulting scattering of points, or point clouds, are first coalesced into regions using a technique such as a morphological **closing** operation
- Next the remaining regions must be localized, for instance using the region-finding technique,
- and then each region's centroid can be utilized as the (noninteger) coordinates for the potential image space line.

Effect of noise

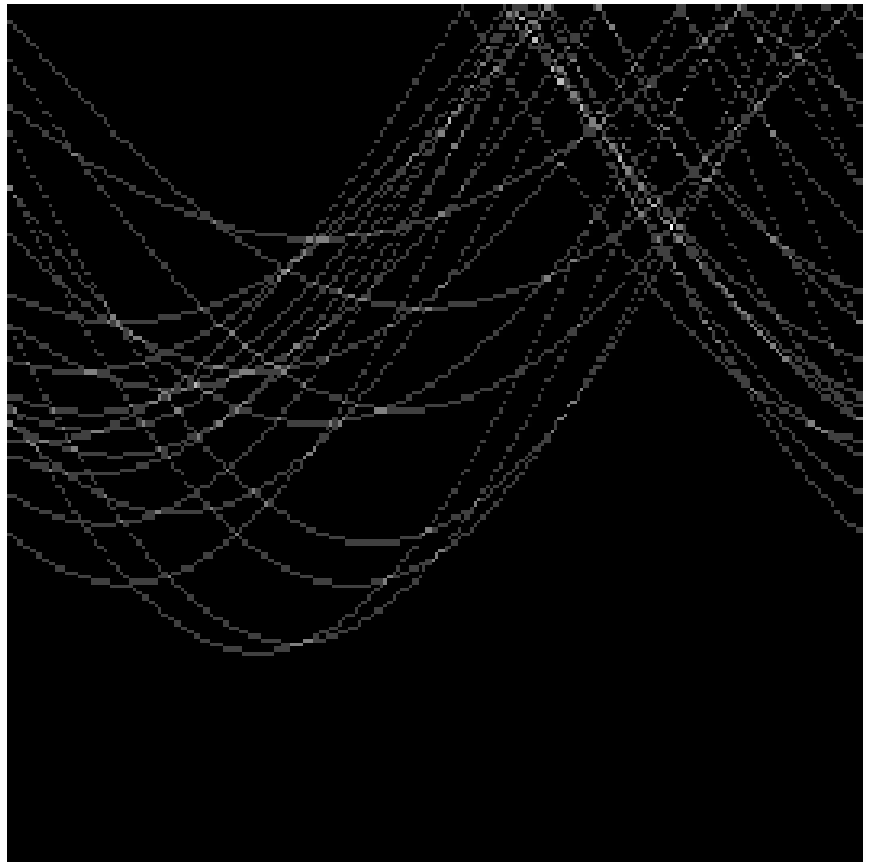
- Number of votes for a line of 20 points with increasing noise:



Random points



features

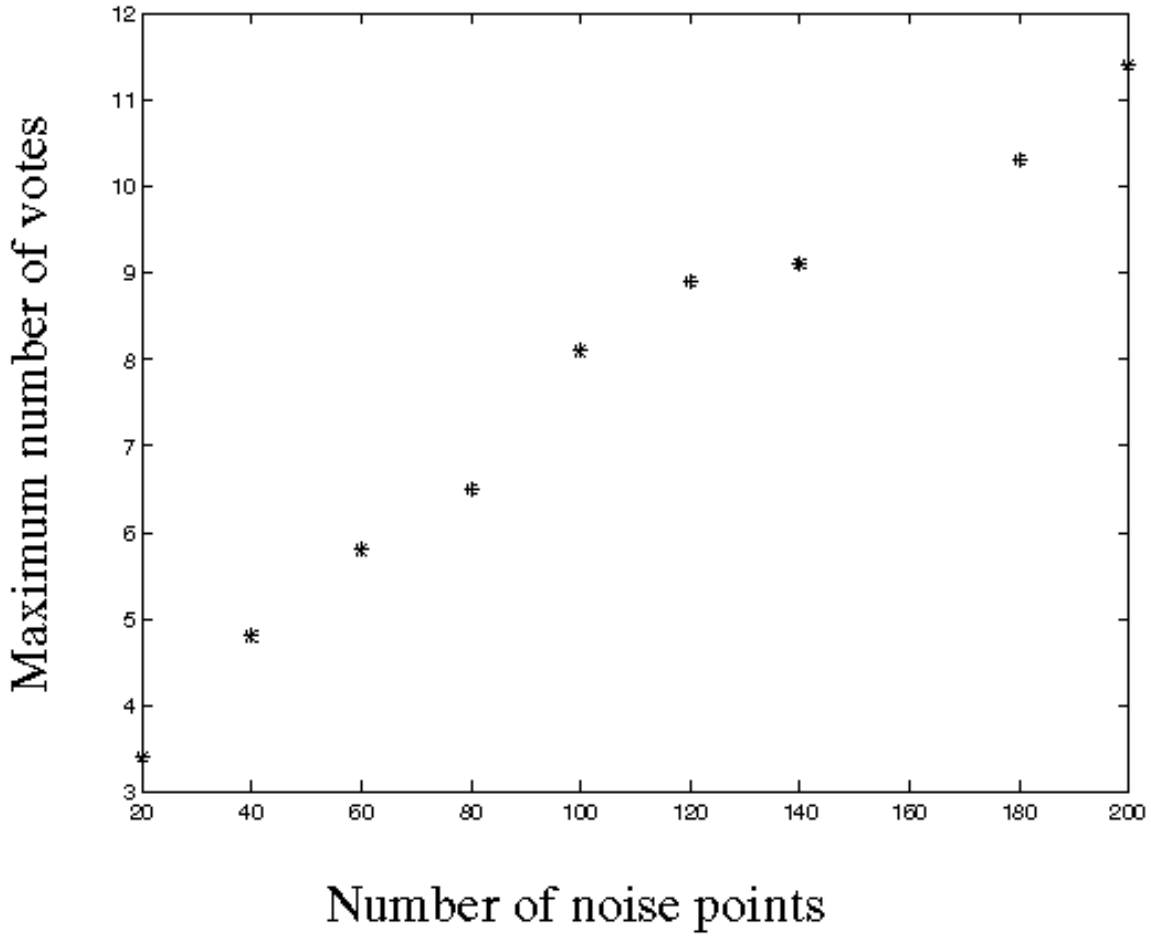


votes

- Uniform noise can lead to spurious peaks in the array

Random points

- As the level of uniform noise increases, the maximum number of votes increases too:



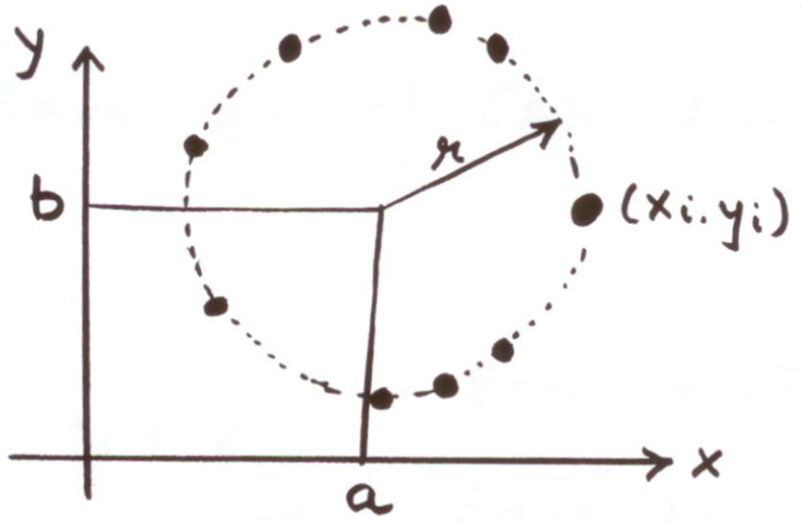
Finding Circles by Hough Transform

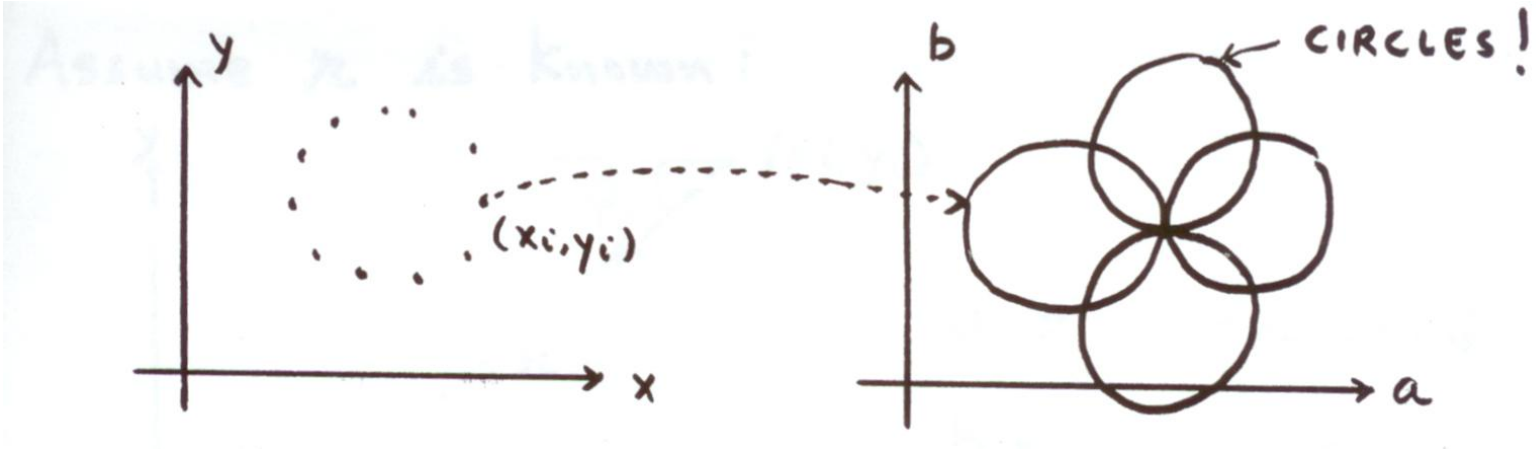
Equation of Circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

If radius is known: (2D Hough Space)

Accumulator Array $A(a,b)$

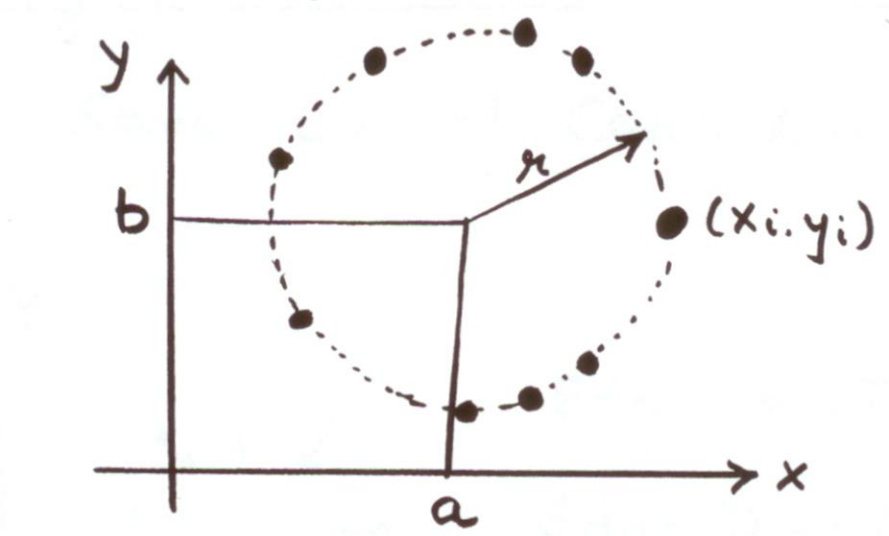




Finding Circles by Hough Transform

Equation of Circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$



If radius is not known: 3D Hough Space!

Use Accumulator array $A(a, b, r)$

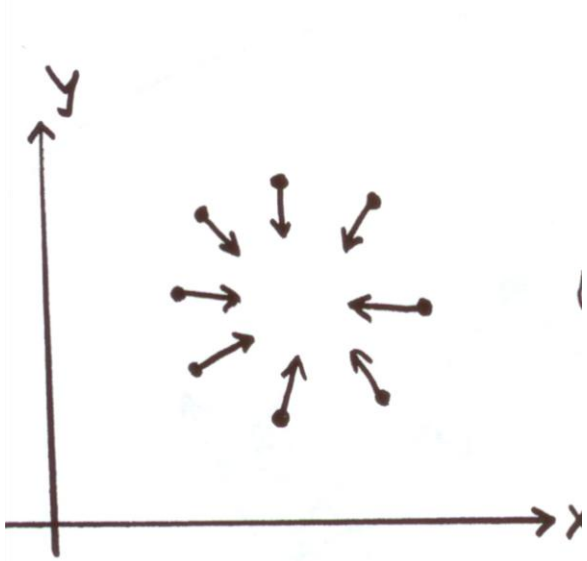
What is the surface in the hough space?

Using Gradient Information

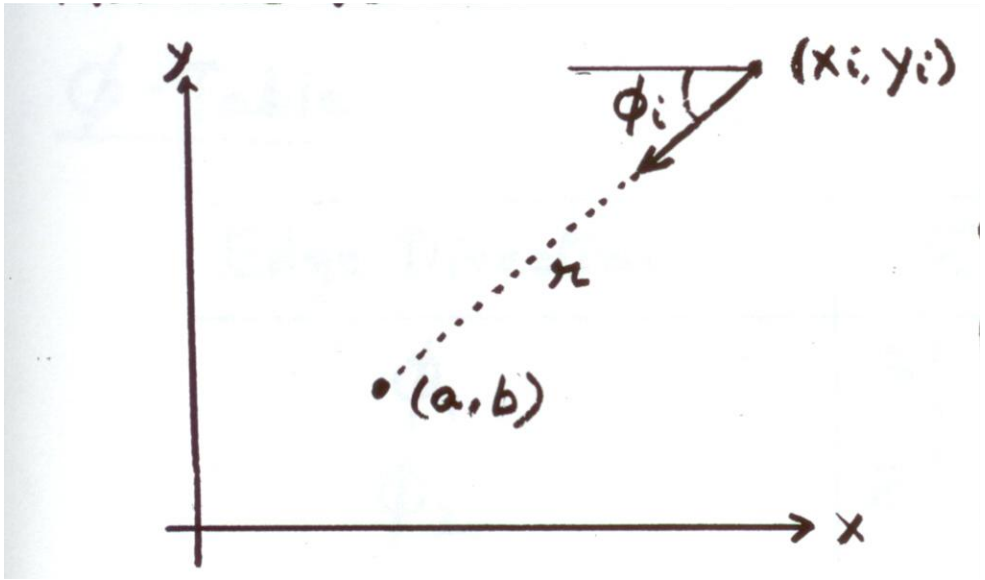
- Gradient information can save lot of computation:

Edge Location (x_i, y_i)

Edge Direction ϕ_i



Assume radius is known:



$$a = x - r\cos\phi$$

$$b = y - r\sin\phi$$

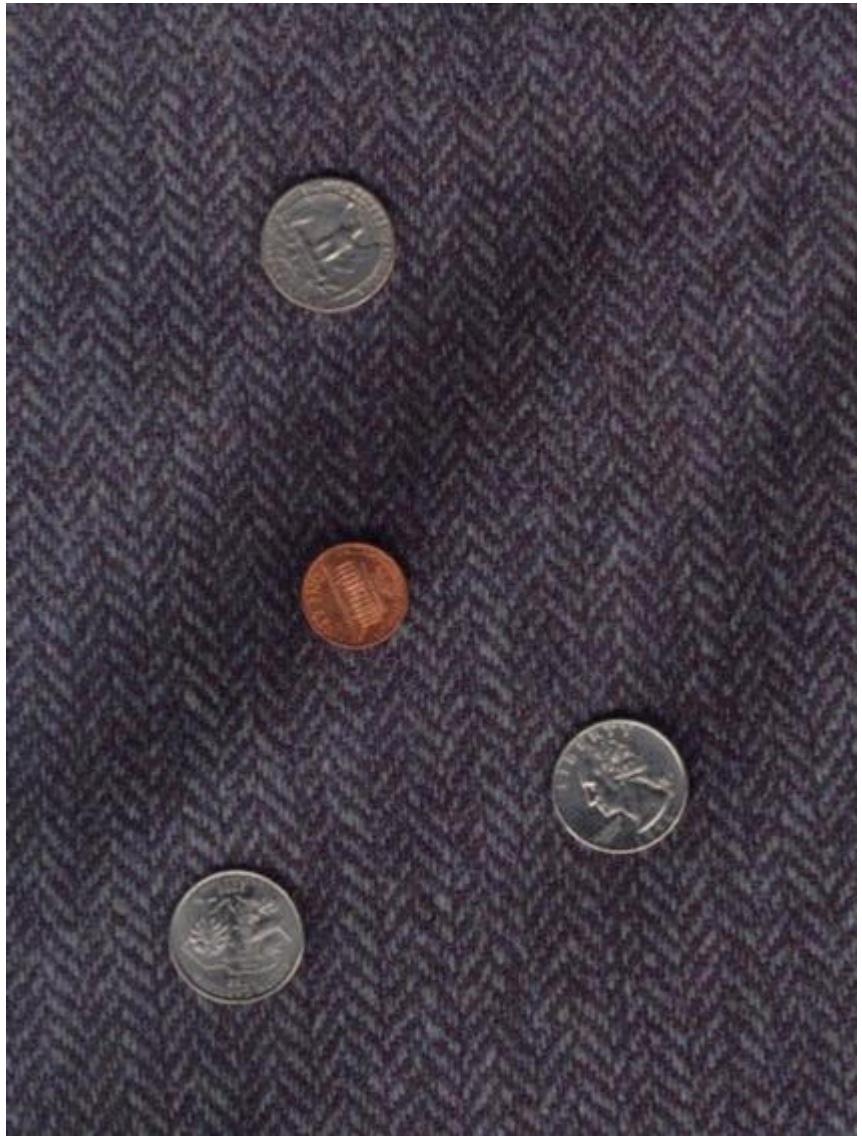
Need to increment only one point in Accumulator!!

Real World Circle Examples

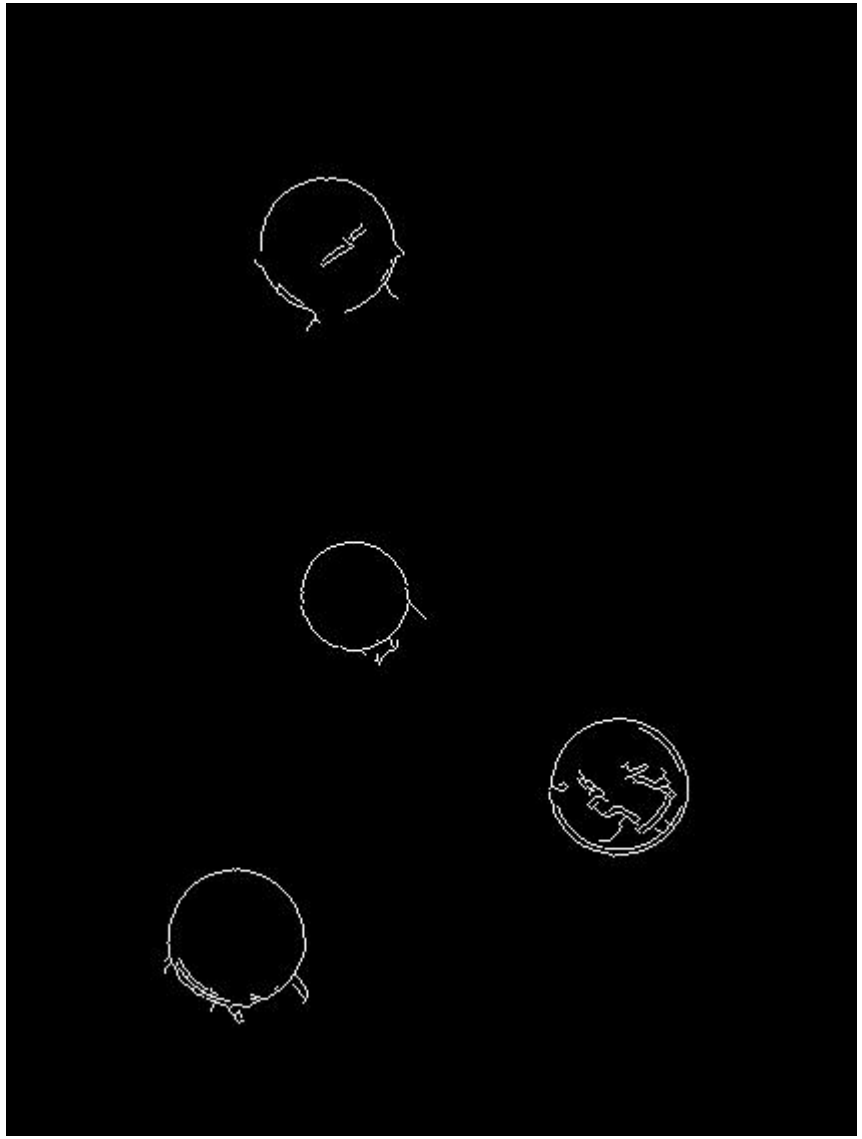
Crosshair indicates results of Hough transform, bounding box found via motion differencing.

Finding Coins

Original



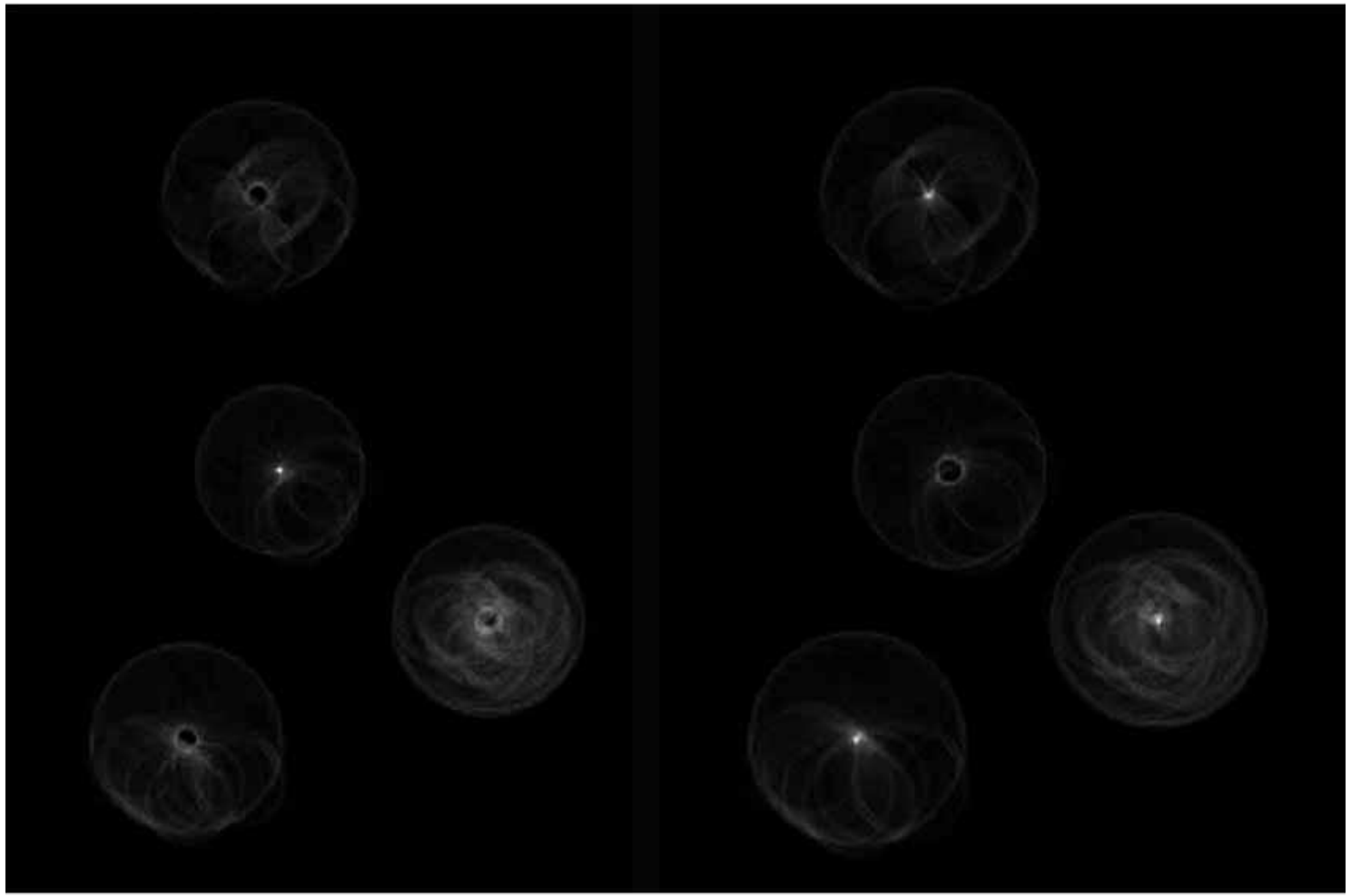
Edges (note noise)



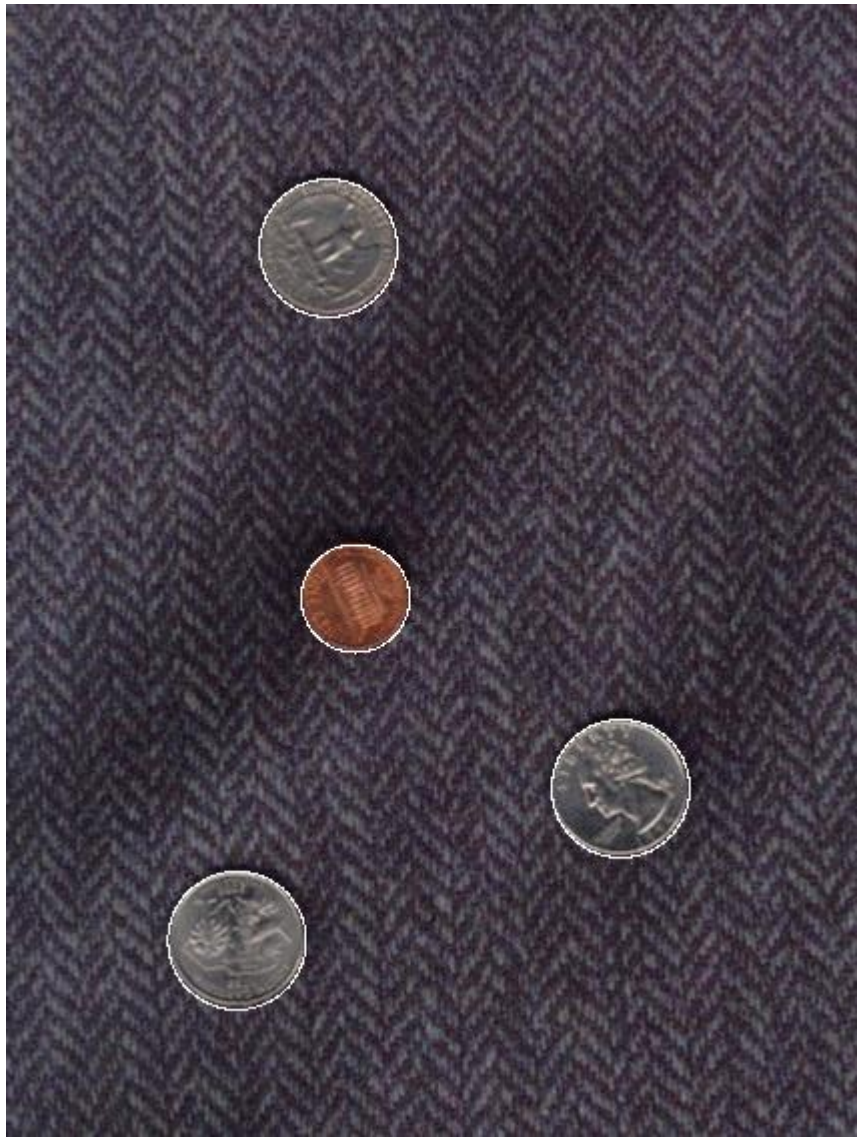
Finding Coins (Continued)

Penn

Quarters



Finding Coins (Continued)

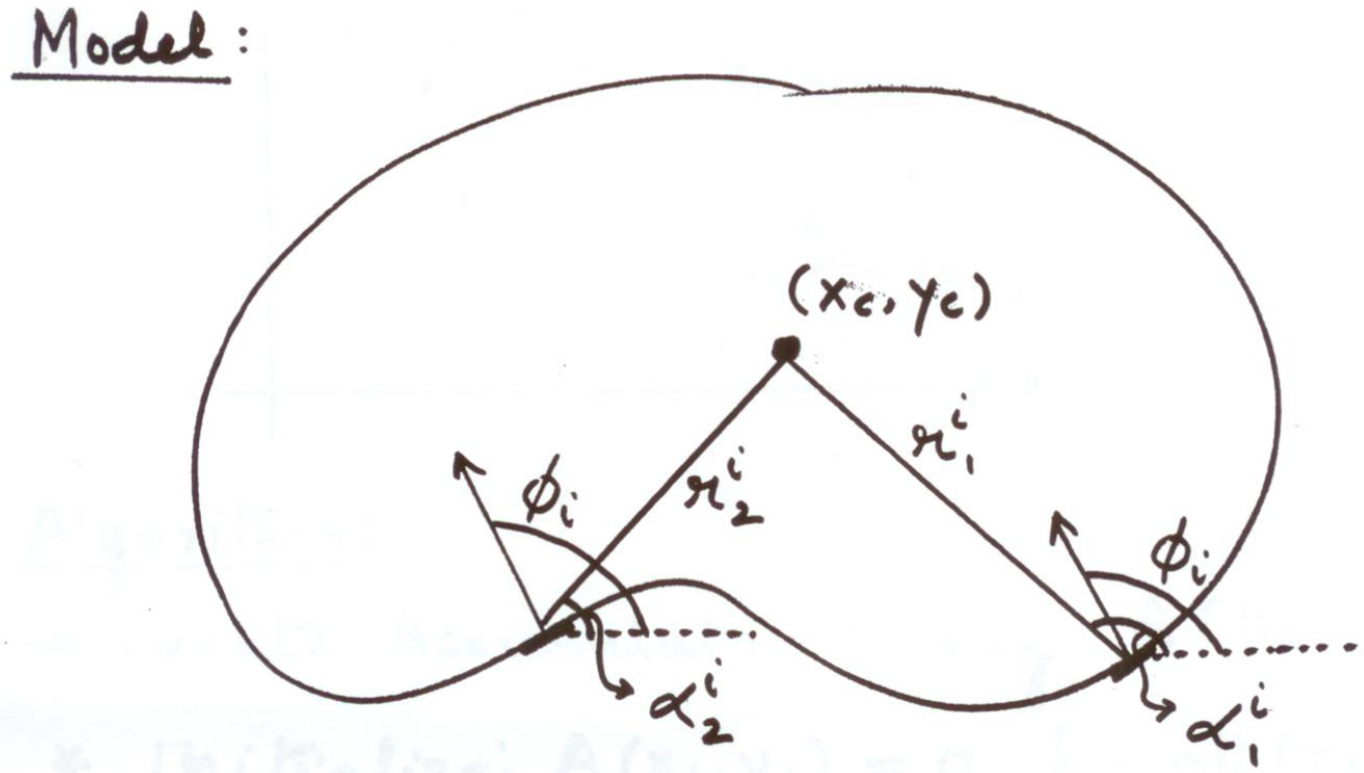


Note that because the quarters and penny are different sizes, a different Hough transform (with separate accumulators) was used for each circle size.

Coin finding sample images from: Vivek Kwatra

Generalized Hough Transform

- Model Shape NOT described by equation



Generalized Hough Transform

- Model Shape NOT described by equation

ϕ-Table

Edge Direction	$\bar{r} = (r, \alpha)$
ϕ_1	$\bar{r}_1^1, \bar{r}_2^1, \bar{r}_3^1$
ϕ_2	$\bar{r}_1^2, \bar{r}_2^2$
ϕ_i	$\bar{r}_1^i \vdots \bar{r}_2^i$
ϕ_n	$\bar{r}_1^n, \bar{r}_2^n$

Generalized Hough Transform

Find Object Center (x_c, y_c) given edges (x_i, y_i, ϕ_i)

Create Accumulator Array $A(x_c, y_c)$

Initialize: $A(x_c, y_c) = 0 \quad \forall (x_c, y_c)$

For each edge point (x_i, y_i, ϕ_i)

For each entry $\overline{r}_k^i$ in table, compute:

$$x_c = x_i + r_k^i \cos \alpha_k^i$$

$$y_c = y_i + r_k^i \sin \alpha_k^i$$

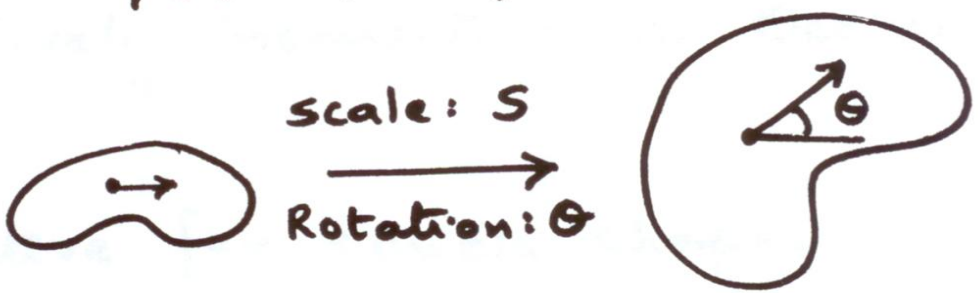
Increment Accumulator: $A(x_c, y_c) = A(x_c, y_c) + 1$

Find Local Maxima in $A(x_c, y_c)$

Scale & Rotation:

Use Accumulator Array:

$$A[x_c, y_c, S, \theta]$$



Use:

$$x_c = x_i + r_k^i S \cos(\alpha_k^i + \theta)$$

$$y_c = y_i + r_k^i S \sin(\alpha_k^i + \theta)$$

$$A(x_c, y_c, S, \theta) = A(x_c, y_c, S, \theta) + 1 .$$

Hough Transform: Comments

- Works on Disconnected Edges
- Relatively insensitive to occlusion
- Effective for simple shapes (lines, circles, etc)
- Trade-off between work in Image Space and Parameter Space
- Handling inaccurate edge locations:
    - Increment Patch in Accumulator rather than a single point

Practical details

- Try to get rid of irrelevant features
  - Take only edge points with significant gradient magnitude
- Choose a good grid / discretization
  - Too coarse: large votes obtained when too many different lines correspond to a single bucket
  - Too fine: miss lines because some points that are not exactly collinear cast votes for different buckets
- Increment neighboring bins (smoothing in accumulator array)
- Who belongs to which line?
  - Tag the votes

Hough transform: Pros

- Can deal with non-locality and occlusion
- Can detect multiple instances of a model in a single pass
- Some robustness to noise: noise points unlikely to contribute consistently to any single bin

Hough transform: Cons

- Complexity of search time increases exponentially with the number of model parameters
- Non-target shapes can produce spurious peaks in parameter space
- It’s hard to pick a good grid size

Summary

- In this lecture we have begun looking at segmentation, and in particular edge detection
- Edge detection is massively important as it is in many cases the first step to object recognition