Segmentation

Previously:
Frequency domain

- Spatial Frequency and Fourier transform
- Convolution theorem
  - Filtering and ringing
  - Spatial sampling and aliasing

Fourier Transform and Convolution

Spatial Domain *(x)*		Frequency Domain *(u)*
$g = f * h$	$\longleftrightarrow$	$G = F \cdot H$
$g = f \cdot h$	$\longleftrightarrow$	$G = F * H$

Fourier Transform and Convolution

So, we can find *g(x)* by Fourier transform

$$g = f * h$$

IFT: g ← G

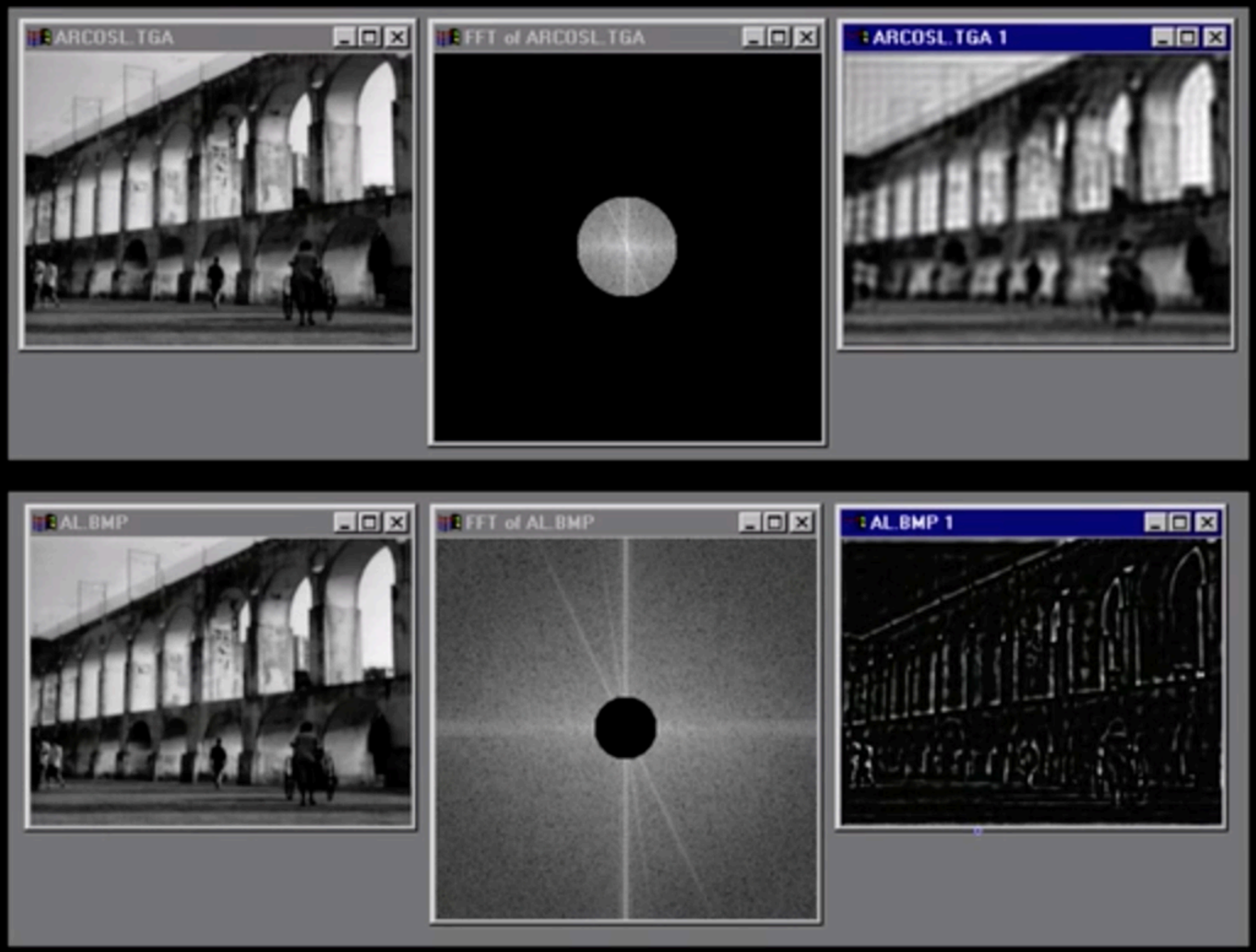
FT: f → F

FT: h → H

$$G = F \times H$$

Padding and Centering

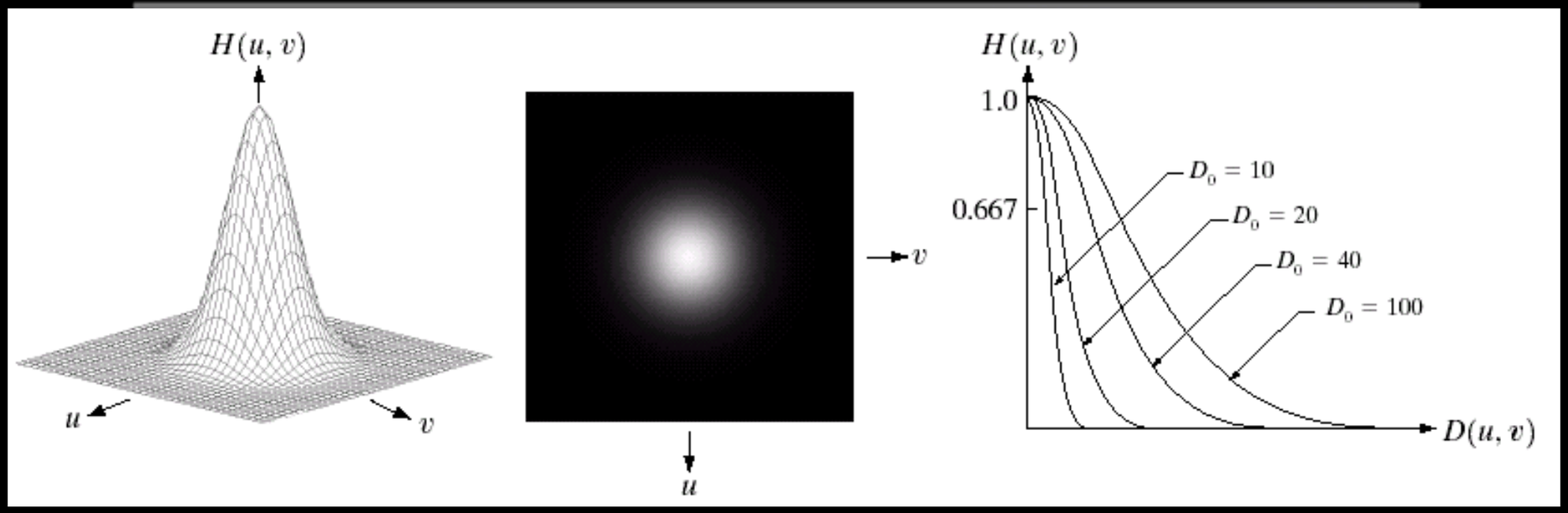
Low and High Pass filtering



Smoothening and Sharpening

Low and High Pass filtering

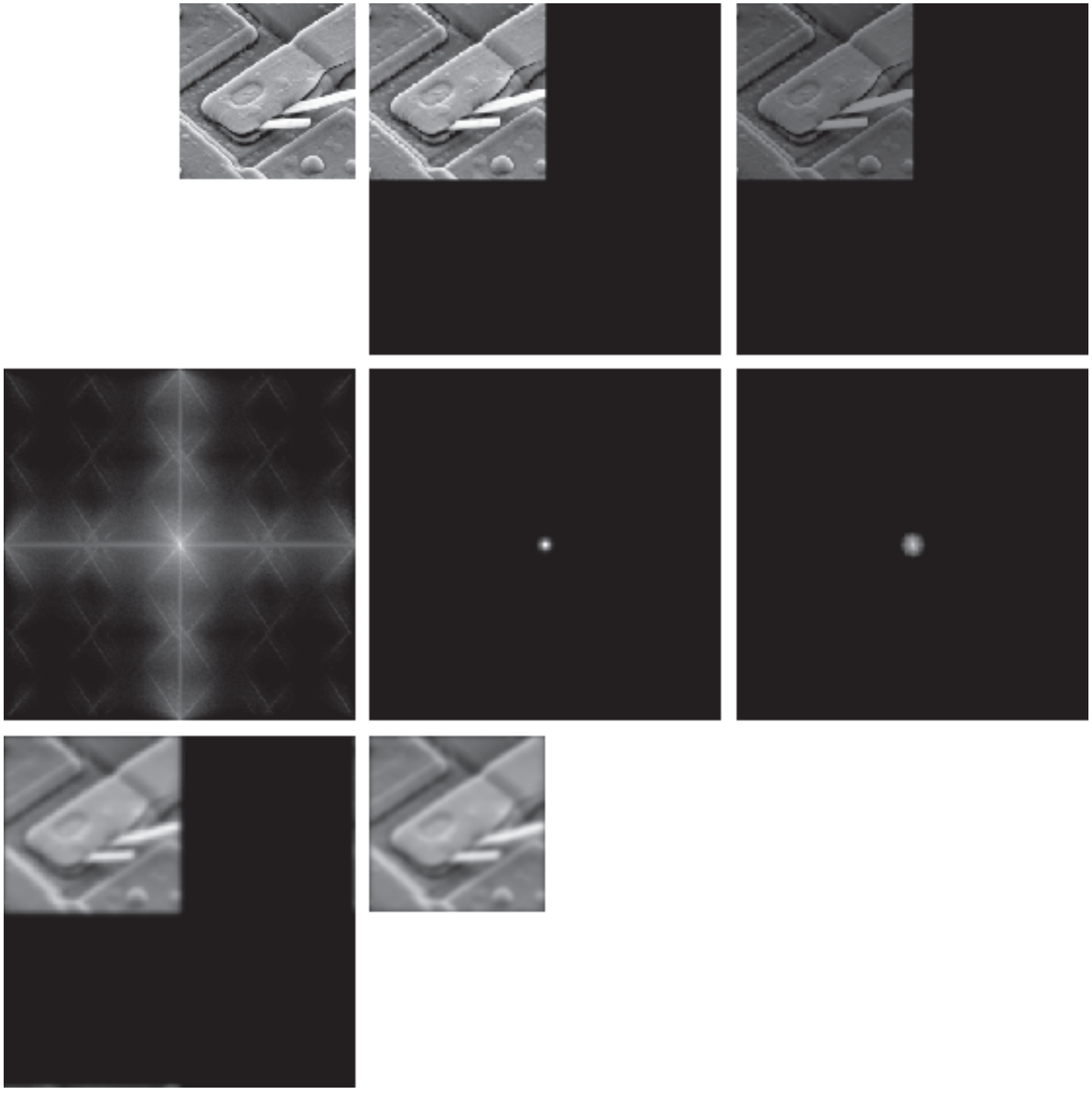
Ringing



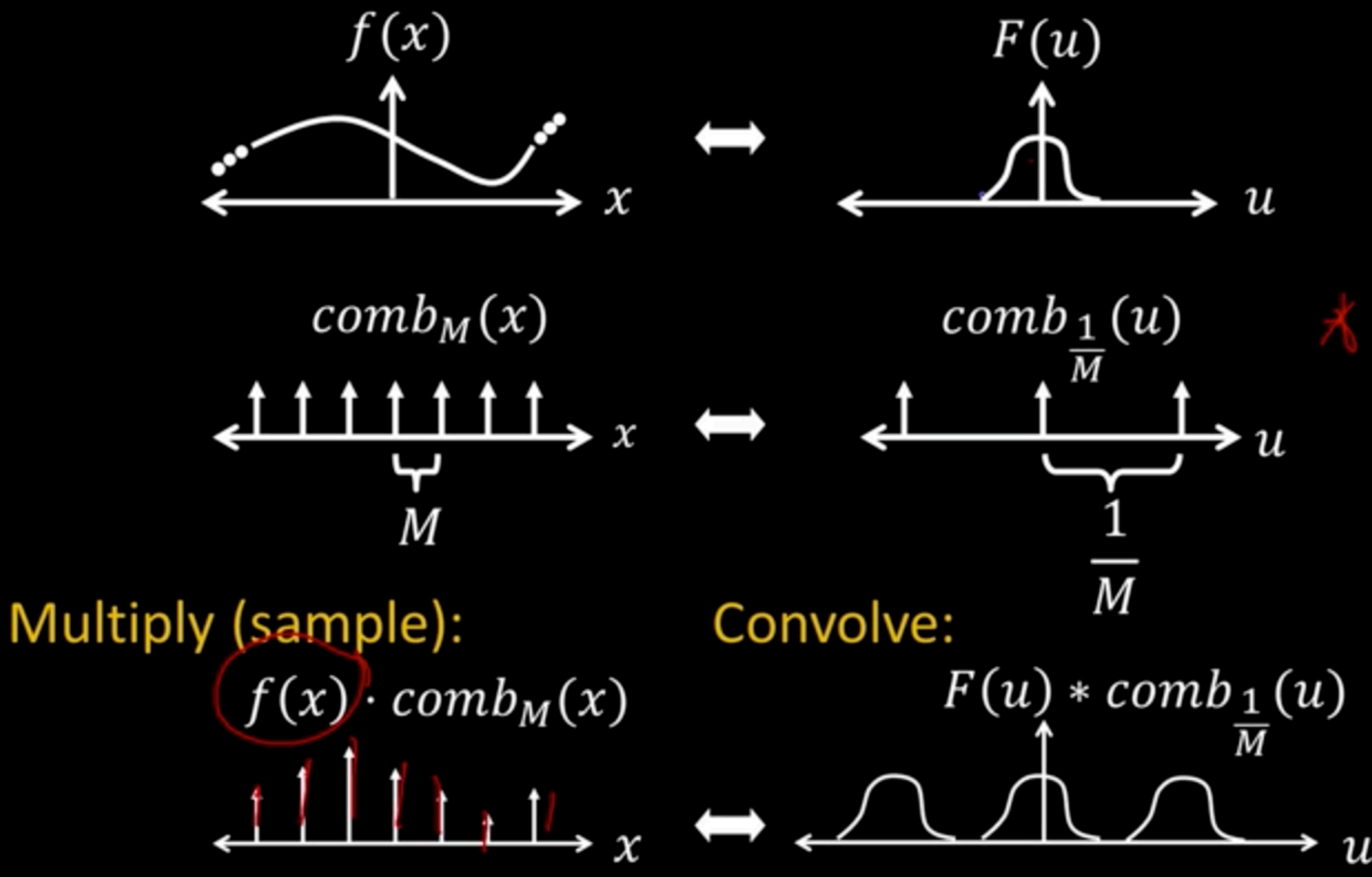
Use Gaussian or Butterworth instead of Ideal LP filter

a b c
d e f
g h

FIGURE 4.35
(a) An $M \times N$ image, f.
(b) Padded image, f_p of size $P \times Q$.
(c) Result of multiplying f_p by $(-1)^{x+y}$.
(d) Spectrum of F. (e) Centered Gaussian lowpass filter transfer function, H, of size $P \times Q$.
(f) Spectrum of the product HF.
(g) Image g_p, the real part of the IDFT of HF, multiplied by $(-1)^{x+y}$.
(h) Final result, g, obtained by extracting the first M rows and N columns of g_p.



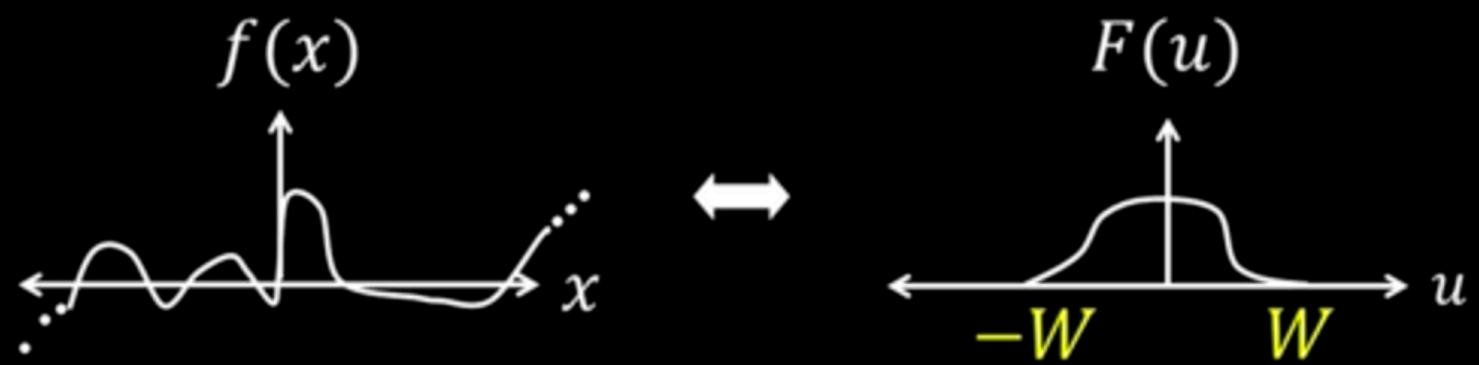
Sampling low frequency signal



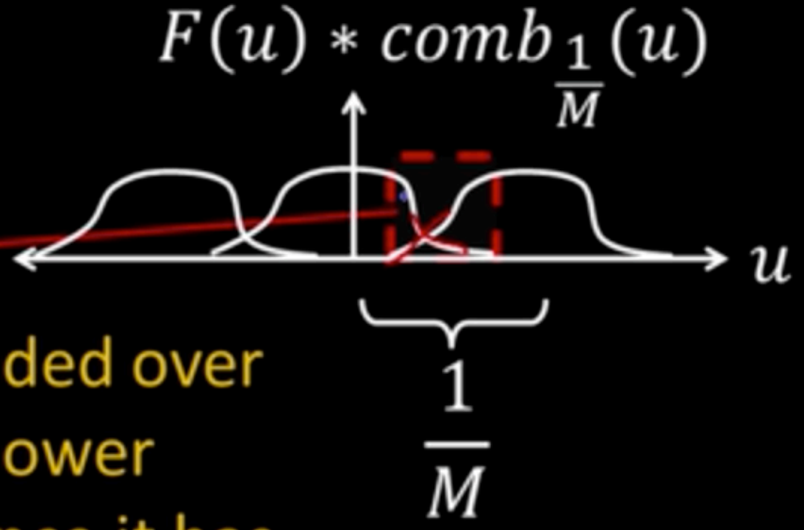
B.K. Gunturk

Sampling and Aliasing

Sampling high frequency signal

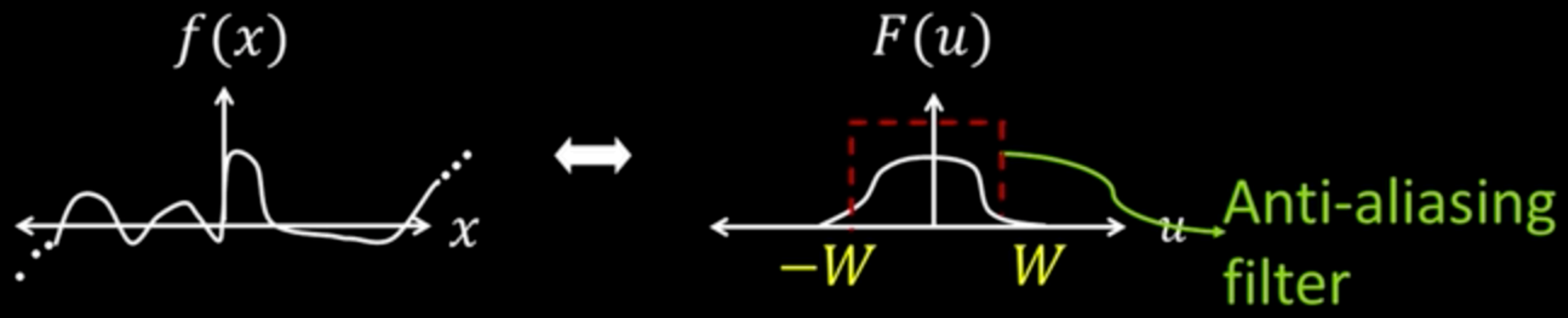


$< f(x) \cdot comb_M(x) >$

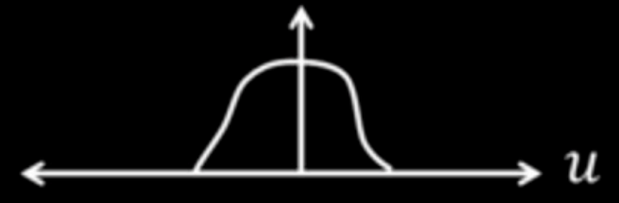


Overlap: The high frequency energy is folded over into low frequency. It is "aliasing" as lower frequency energy. And you cannot fix it once it has happened.

Sampling high frequency signal

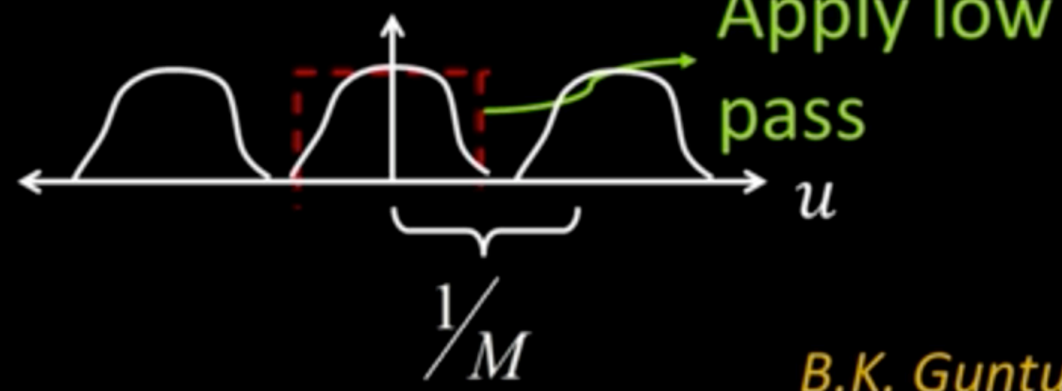


$$f(x) * h(x) \quad \Leftrightarrow$$



Anti-aliasing filter

$$[f(x) * h(x)]comb_M(x) \quad \Leftrightarrow$$



B.K. Gunturk

Today: Segmentation

- Partitioning and extracting (background vs. object), boundary vs. region, intensity changes vs. similarity
- Boundary-based methods (dissimilarity)
  - Edges vs. Boundary, Canny, Hough transform
- Region-based methods (similarity)
  - Thresholding, Region-growing ++

Segmentation

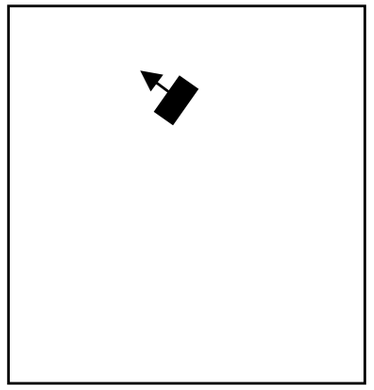
- Partitioning and image into regions, create a label image.
  - Object (foreground) vs. background.
- An object can be described by its..
  - Boundary (dissimilarity)
  - Region (similarity)

Boundary-based methods

Edges vs Boundaries

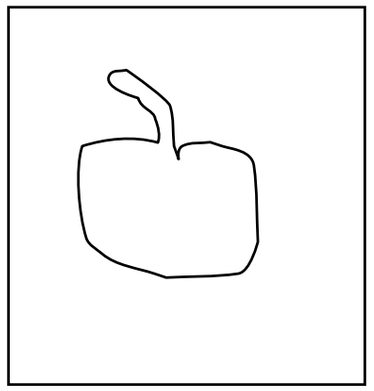
Magnitude (Gradient)

- Edges
  - Local intensity discontinuities
  - Points
  - Not dependent on models



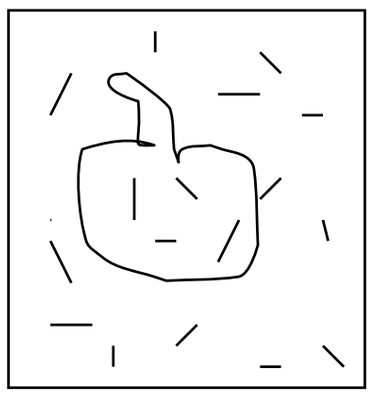
Local edge point or edge element

- Boundaries
  - Extensive
  - Composed of many points
  - May be dependent on models



Object boundary

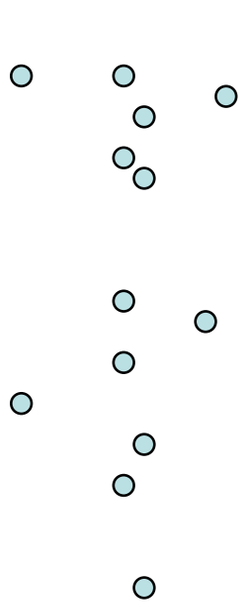
- Typically our goal is to reconstruct the boundary from local edge elements



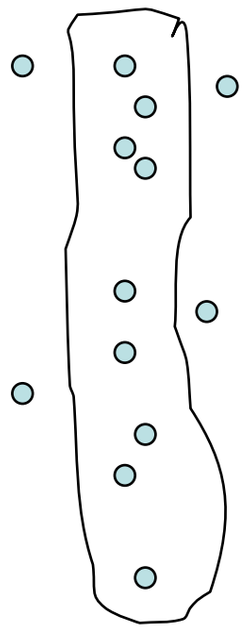
To group edge points, we may need a domain or object model

Boundaries

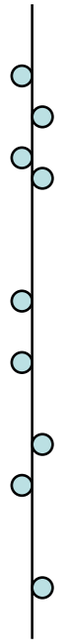
- If we can group individual local edge elements, we can fit a line or curve to them, to find the object boundary



Edge elements



Grouping

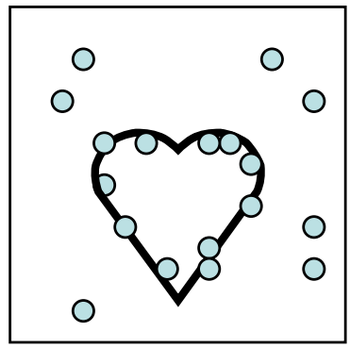


Best fit line

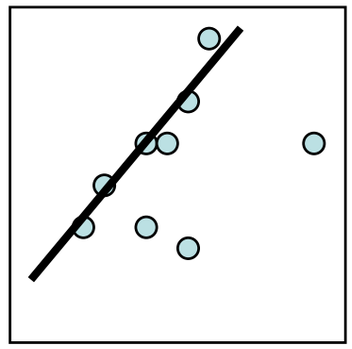
Knowledge about Boundary

More Knowledge Less

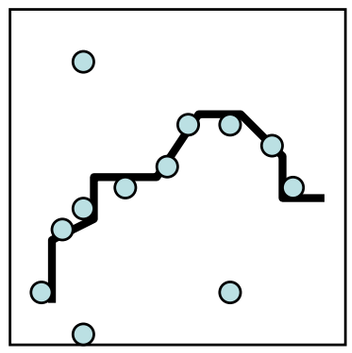
- Can extract the complete closed contour of the object if its shape is known



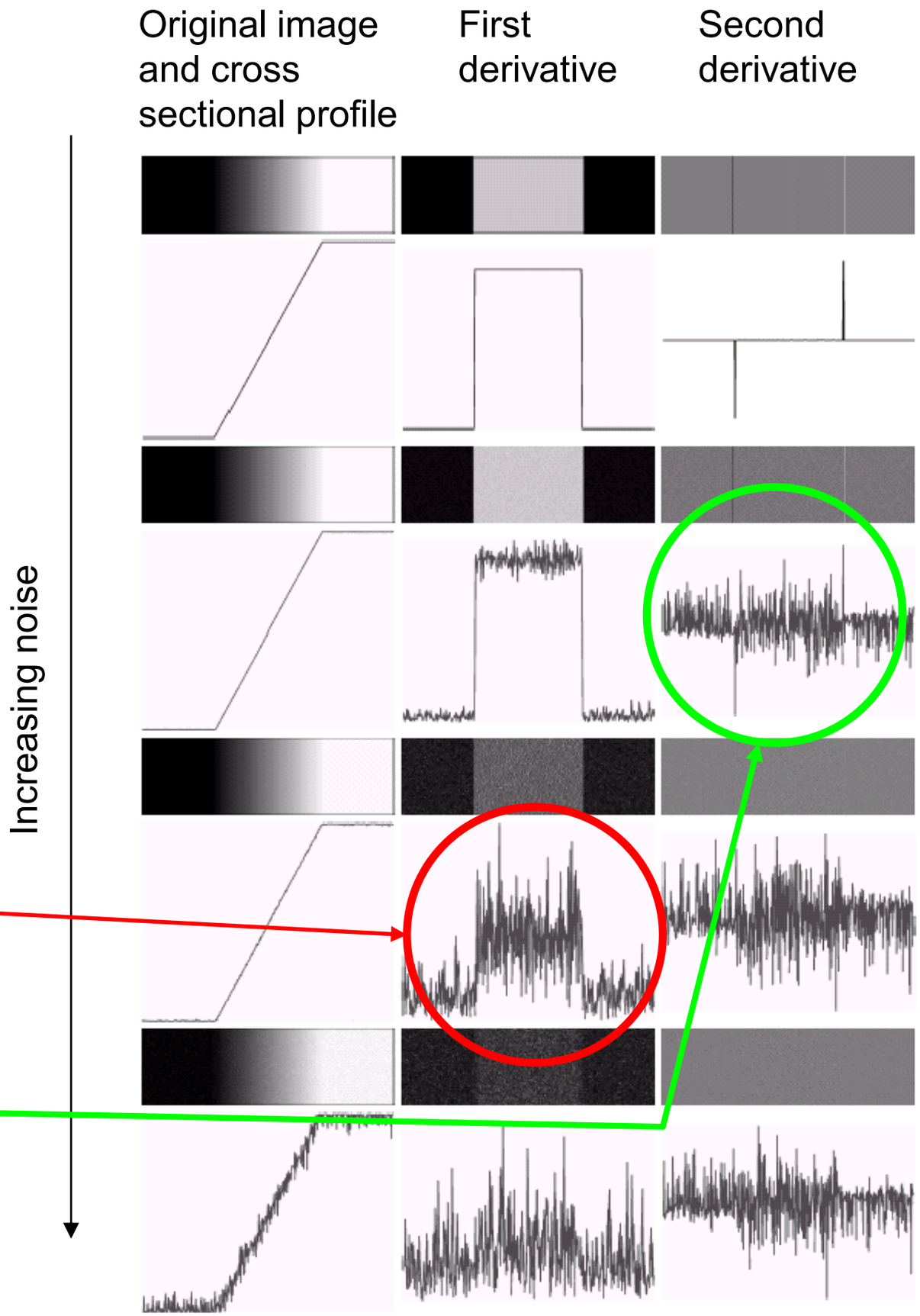
- Can extract pieces of the boundary using general line or curve models



- Can just try to find a connected series of edge elements, using only heuristics on say, edge curvature



- Edge operators also enhance noise
- Example – first and second derivatives of a noisy ramp edge
  - Even with very low noise, 1st derivative is useless at this noise level
  - Second derivative is useless for even less noise

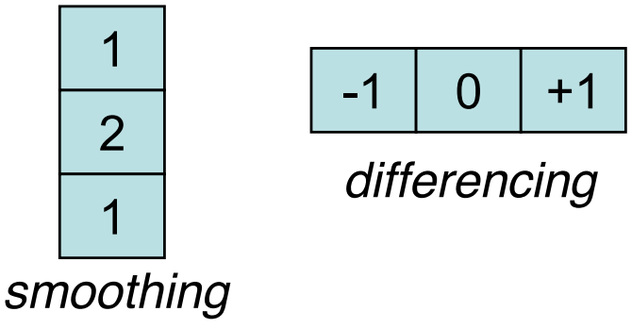


Edge Detection

- Practical edge detectors - first do image smoothing for noise reduction
- Example – Sobel filter

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

- Effectively does a smoothing followed by a differencing



- All edge detectors do
  - Smoothing for noise reduction
  - Detection of candidate edge points
  - Edge localization – keeping only the points that are closest to the true edge

Sobel Edge Detector Example

Original image “pout.tif”

Gradient magnitude, using Sobel masks

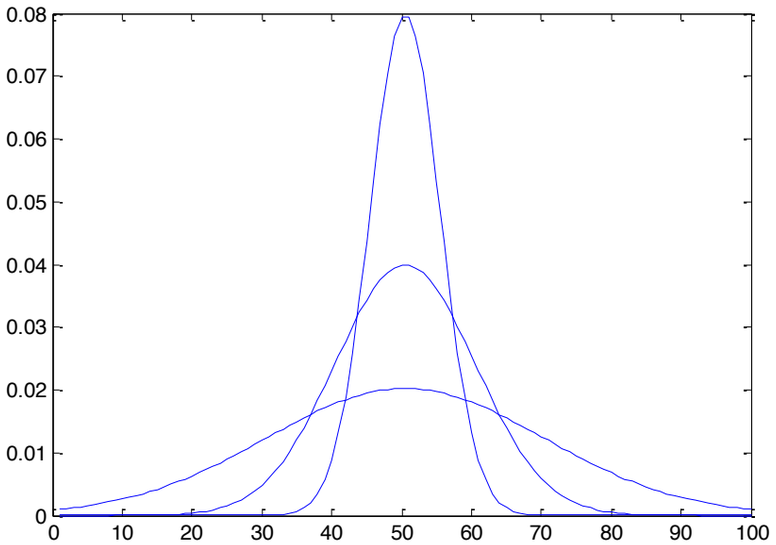
```
[fx,fy] = gradient(I);
fmag = (fx.^2 + fy.^2) .^ .5;
imtool(fmag,[]);
```

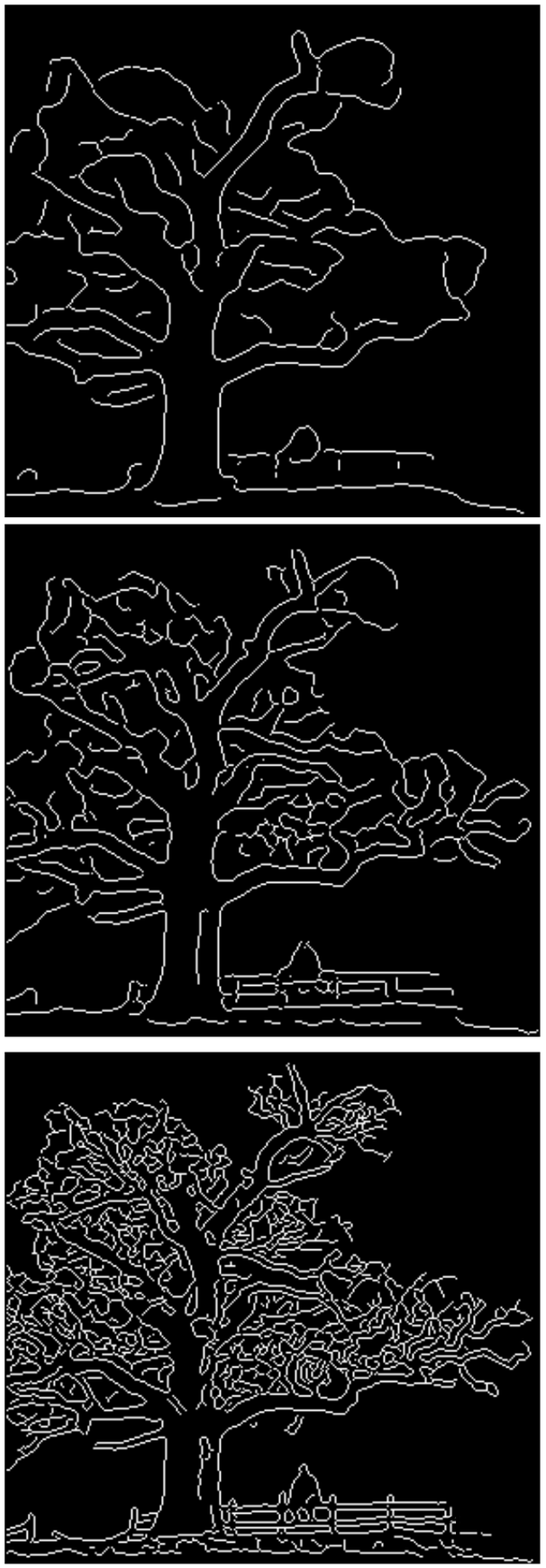
$$|\nabla \mathbf{f}| = \left[\left(\frac{\partial f}{\partial x} \right)^2 + \left(\frac{\partial f}{\partial y} \right)^2 \right]^{1/2}$$

Scale-Space

- The space of images created by applying a series of operators of different scales
- Gaussians of different sizes can be used to filter the image at different scales

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





- Convolving with a Gaussian blurs the image, wiping out all structure much smaller than σ

Laplacian of a Gaussian (LoG)

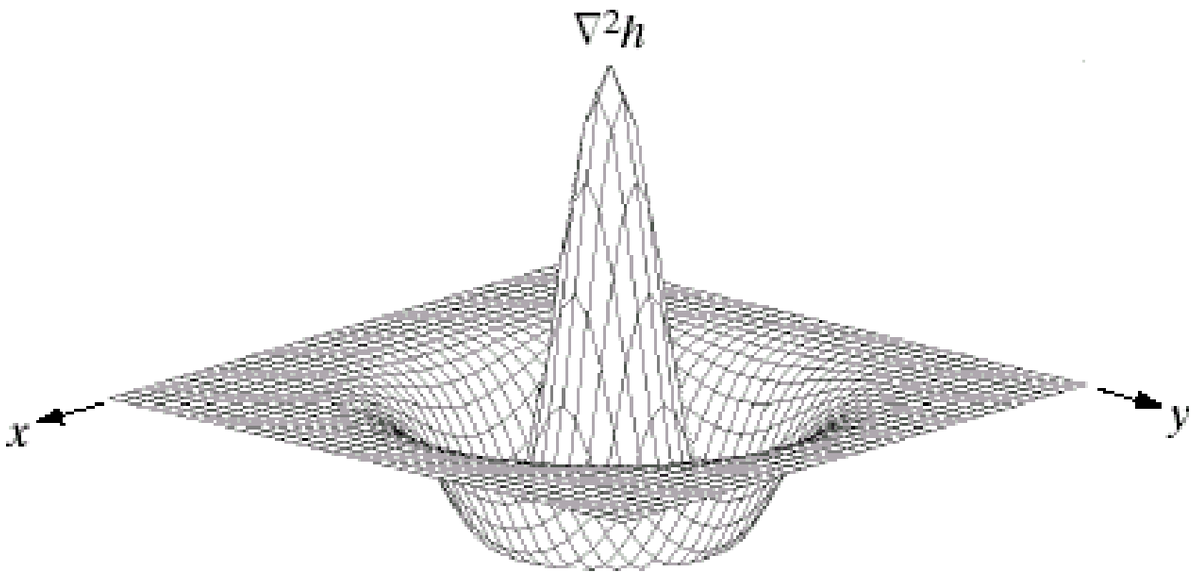
- Also called “Marr-Hildreth” operator

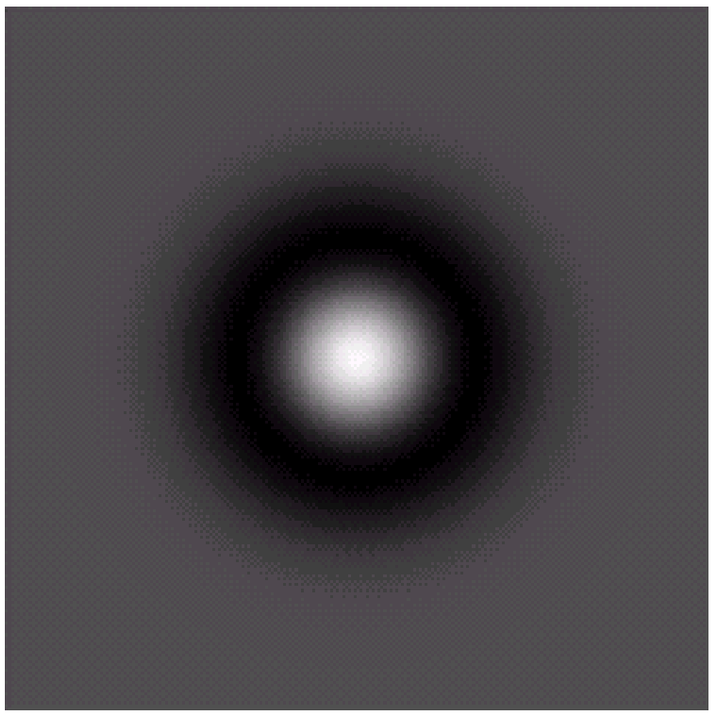
$$\nabla^2 g$$

$$\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}\right) \qquad \frac{1}{2\pi\sigma^2}e^{-(x^2+y^2)/(2\sigma^2)}$$

- Convolve with a Gaussian, then take the Laplacian; or do the equivalent operation

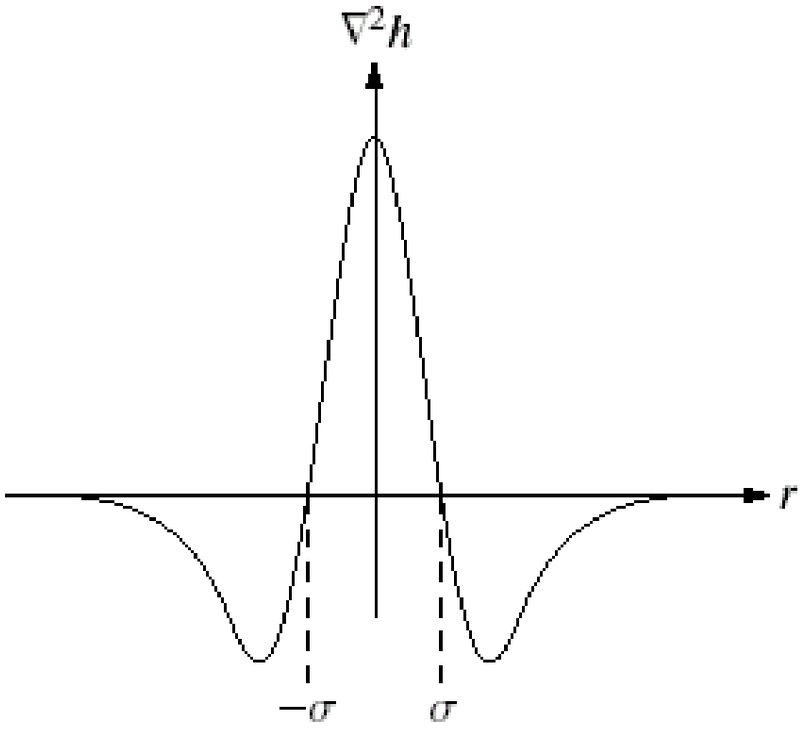
$$\nabla^2\left(g * I\right)=\left(\nabla^2 g\right)* I$$



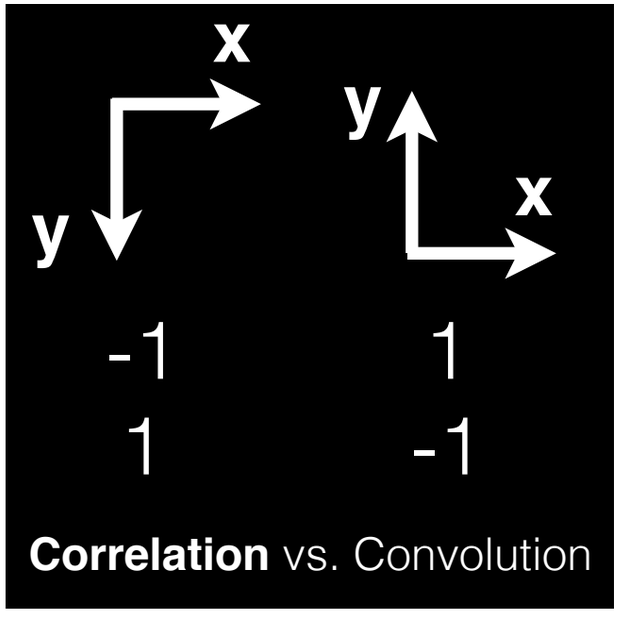


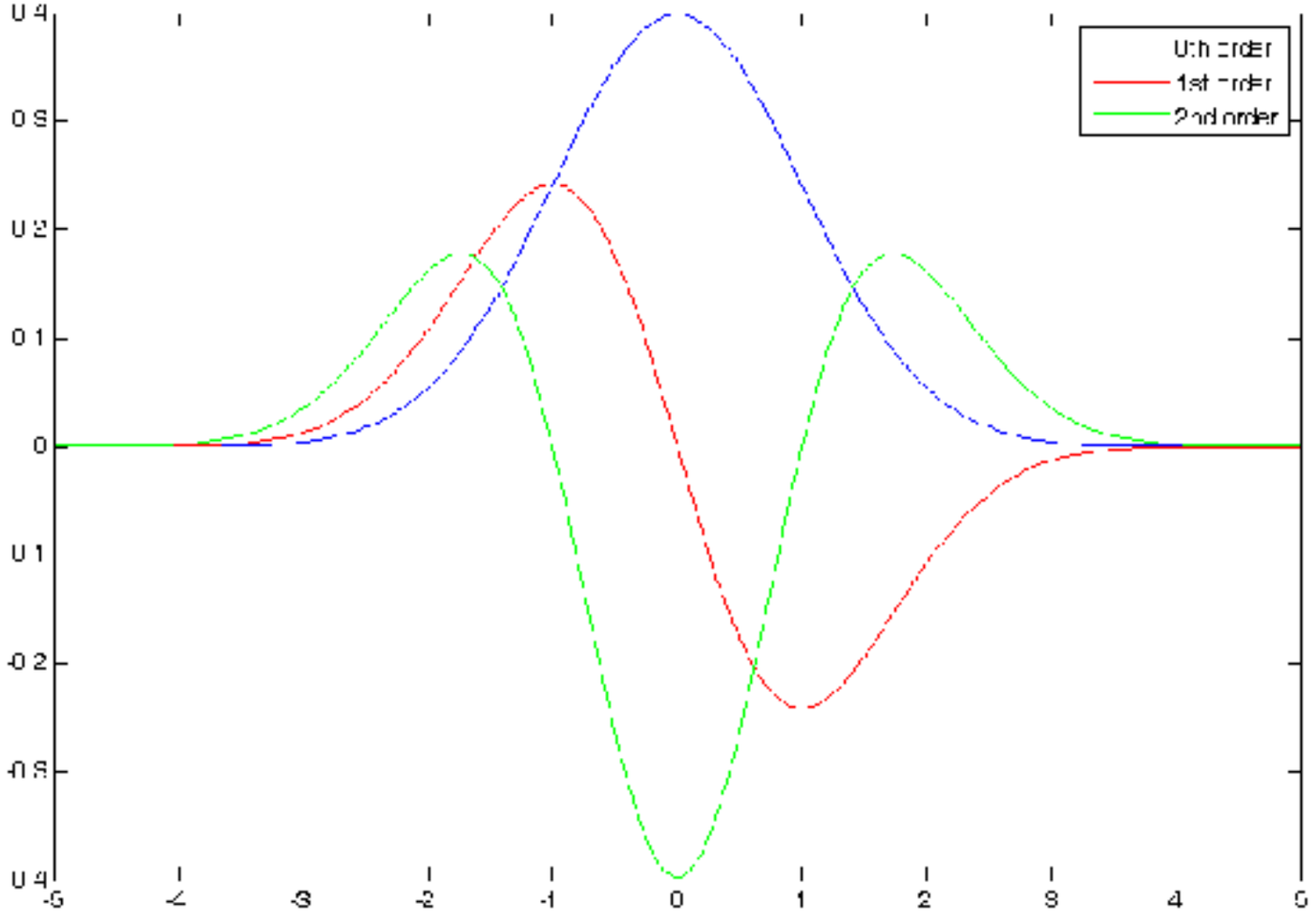
a b
c d

FIGURE 10.14
Laplacian of a Gaussian (LoG).
(a) 3-D plot.
(b) Image (black is negative, gray is the zero plane, and white is positive).
(c) Cross section showing zero crossings.
(d) 5×5 mask approximation to the shape of (a).

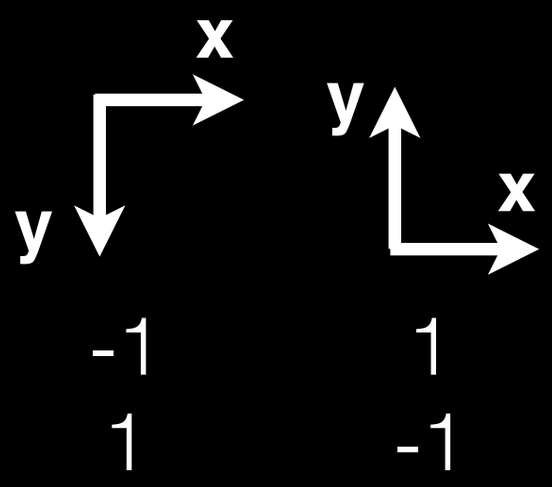


0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0



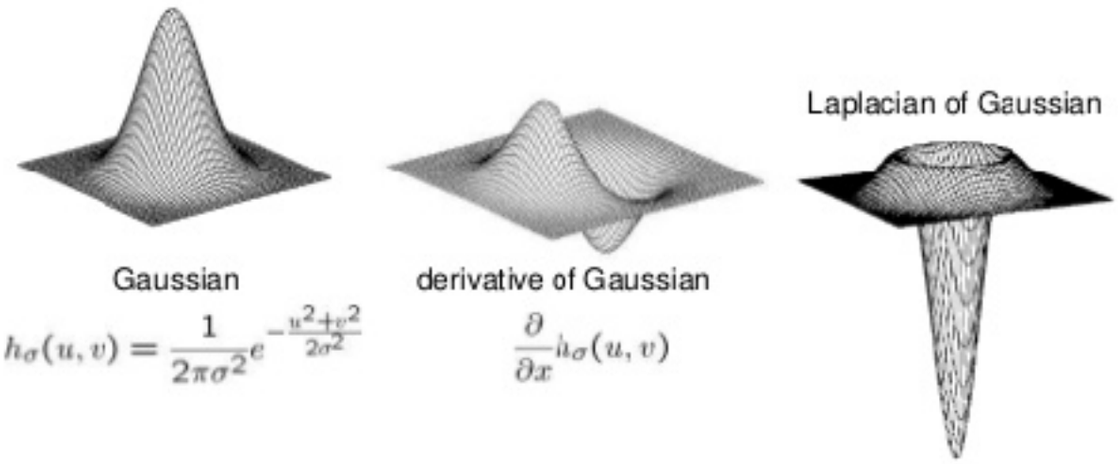


Correct!



Correlation vs. Convolution

2D edge detection filters



Gaussian

$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$

derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

$\nabla^2 h_\sigma(u,v)$

∇^2 is the **Laplacian** operator:

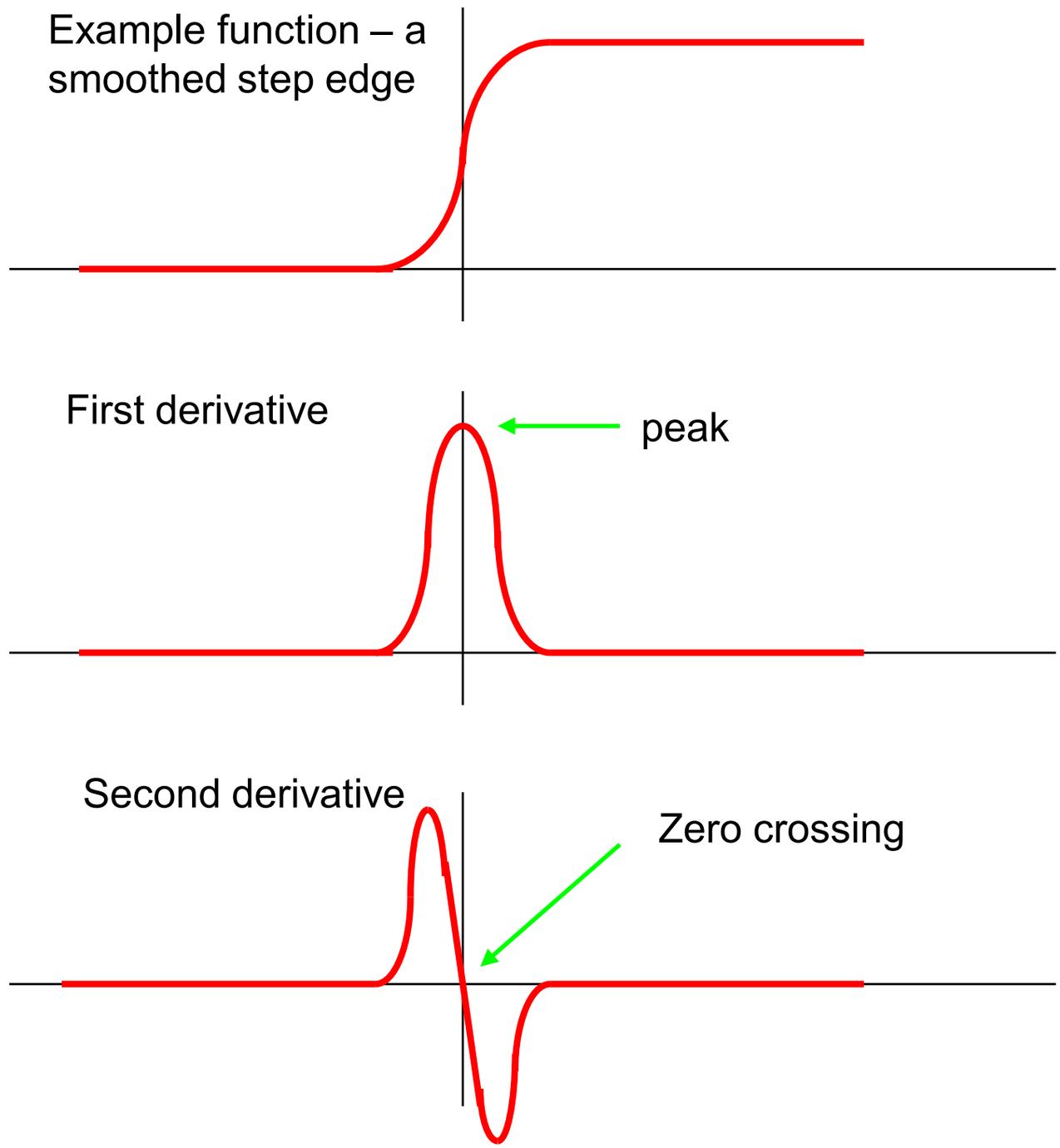
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Zero Crossings

- Zero crossings in the convolved result indicate presence of an edge

- Peaks in the first derivative are equivalent to zeros in the second derivative

- Magnitude of the edge is proportional to the slope at the zero crossing



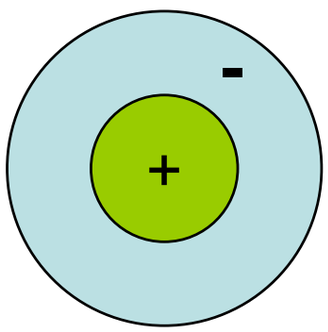
Other Properties of LoG

(DoG)

- LoG can be approximated by a difference of Gaussians

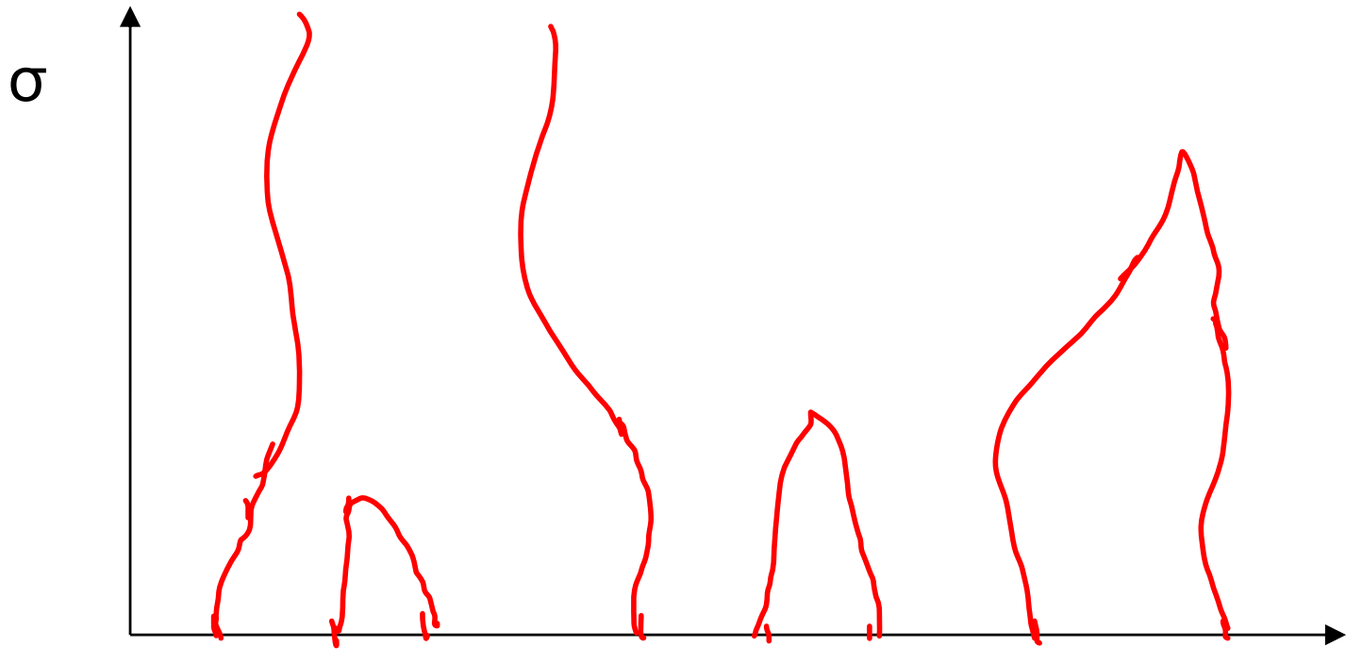
$$\nabla^2 g(\sigma) \approx g(\sigma_1) - g(\sigma_2), \quad \sigma_2 / \sigma_1 \approx 1.6$$

- There is evidence that the human visual system does edge detection similar to LoG
  - Excitatory center, inhibitory surround
  - A set of spatial frequency tuned channels



Properties of Zero Crossings

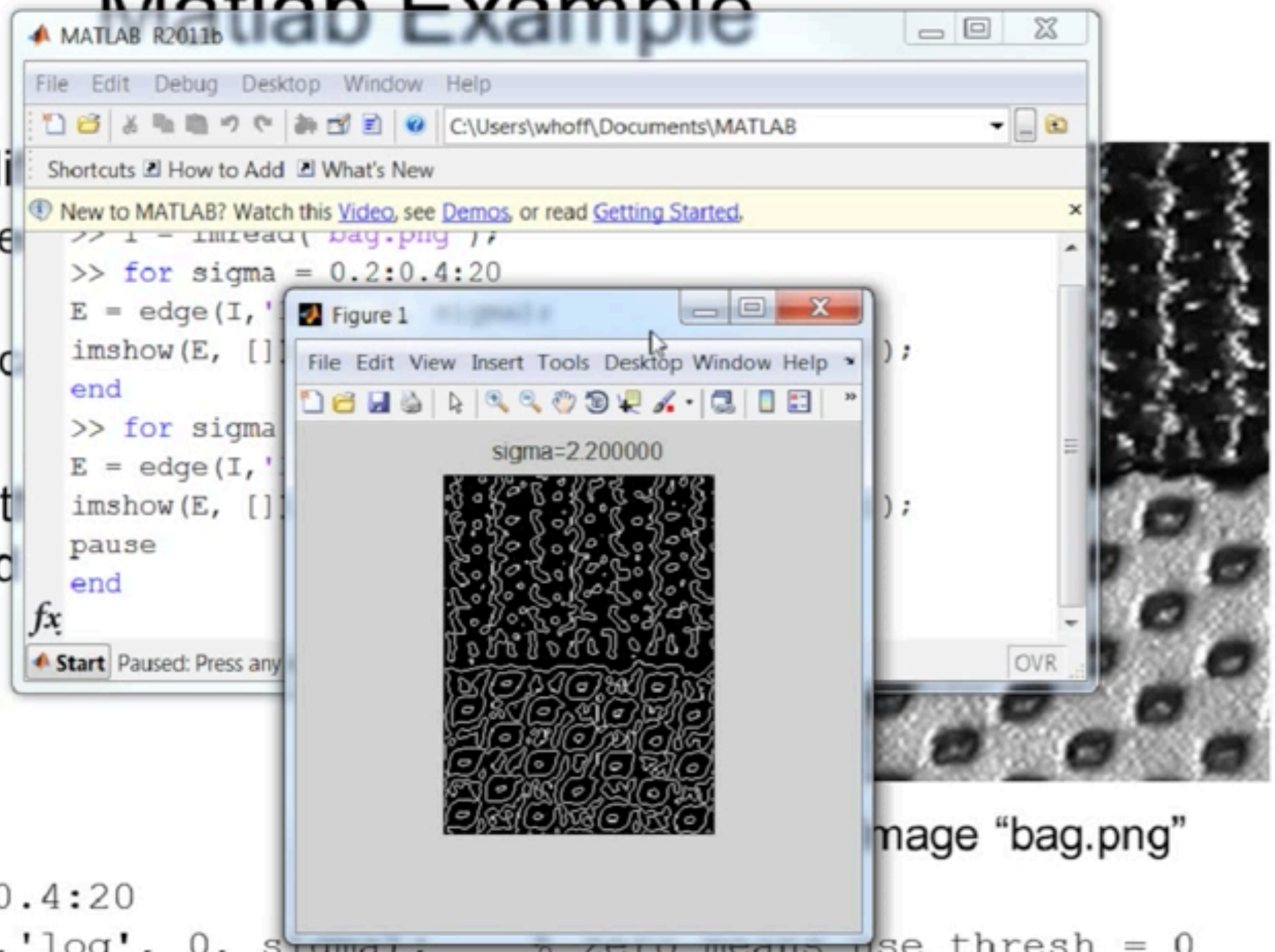
- Zero crossings form closed contours (they are a “level set”, separating positive from negative regions)
- Zero crossings are separated on the average by a distance of $2\sqrt{2}\,\sigma$
- As σ increases, zero crossings are never created; they only merge or eliminate in pairs



Cross section of an image showing locations of zero crossings

Matlab Example

- Would need di
  - Obtain close spots
  - Detect vertic
- Watch to see t only destroyed

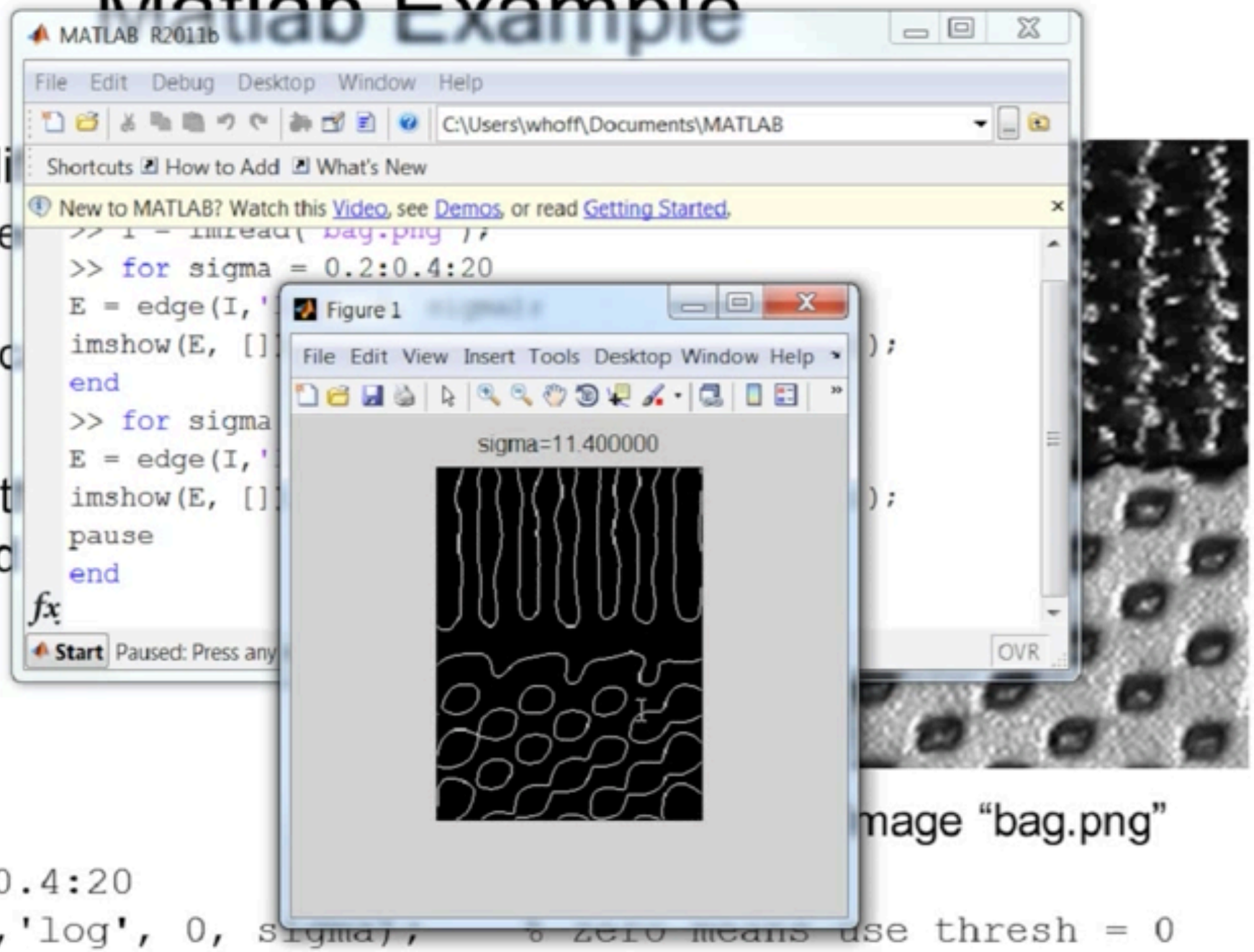


mage “bag.png”

```
for sigma=0.2:0.4:20
    E = edge(I,'log', 0, sigma);    % zero means use thresh = 0
    imshow(E, []), title(sprintf('sigma = %f', sigma));
    pause
end
```

Matlab Example

- Would need di
  - Obtain close spots
  - Detect vertic
- Watch to see t only destroyed



mage “bag.png”

```
for sigma=0.2:0.4:20
    E = edge(I,'log', 0, sigma);    % zero means use thresh = 0
    imshow(E, []), title(sprintf('sigma = %f', sigma));
    pause
end
```

Canny Edge Operator

- We can derive the optimal edge operator to find step edges in the presence of white noise, where “optimal” means
  - Good detection (minimize the probability of detecting false edges and missing real edges)
  - Good localization (detected edges must be close to the true edges)
  - Single response (return only one point for each true edge point)

- Canny’s analysis
  - He simulated step edges, with additive white Gaussian noise
  - Did numerical optimization to find the optimal edge operator
  - Found that a good approximation to the optimal operator is the first derivative of a Gaussian, in the direction of the gradient

Similar to LoG, but doesn’t use Laplacian

Derivative of Gaussian

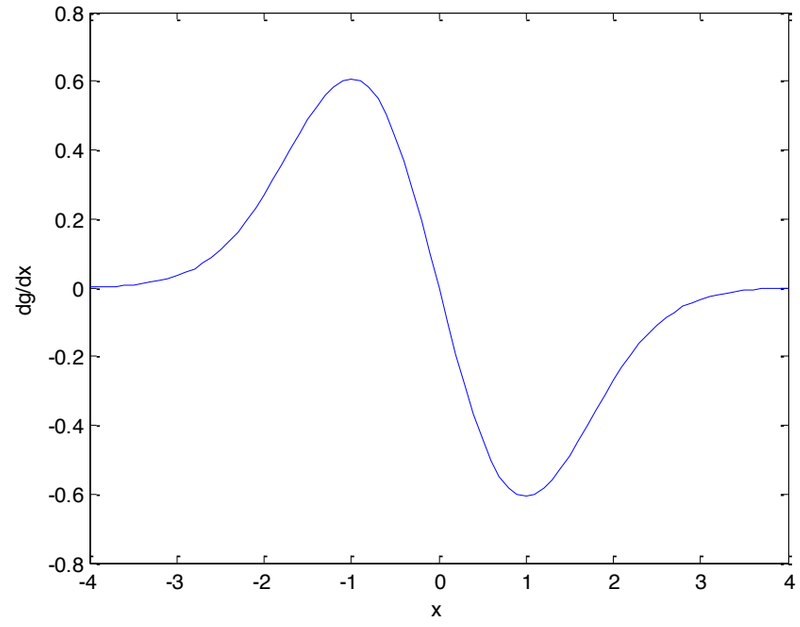
- The gradient of the smoothed image is

$$\nabla\left[G_\sigma * I\right] = \left[\nabla G_\sigma\right] * I$$

- The gradient operators are

$$\nabla G_\sigma = \begin{pmatrix} \frac{\partial G_\sigma}{\partial x} & \frac{\partial G_\sigma}{\partial y} \end{pmatrix}^T$$

$$= \begin{pmatrix} -x & -y \end{pmatrix} \frac{1}{\sigma^3} \exp\left(-\frac{x^2 + y^2}{2\sigma^2} \right)$$



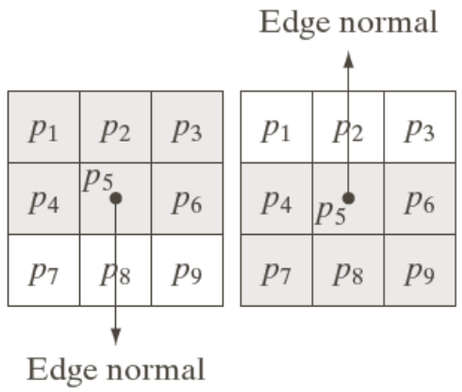
- An edge point is a peak in the magnitude of the gradient of the smoothed image, in the direction of the gradient
  - We then suppress nonmaxima along this direction

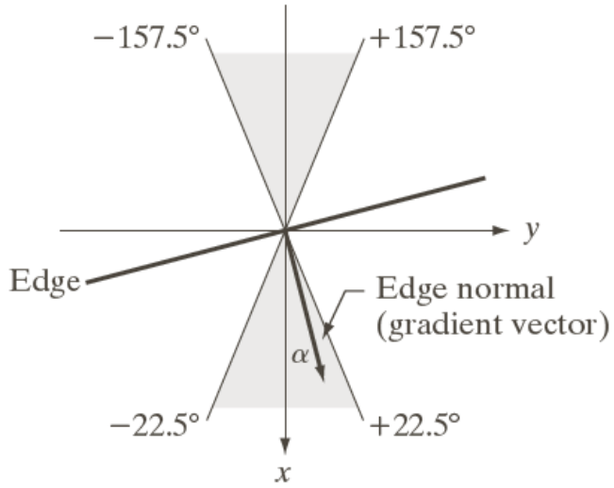
Canny Edge Operator

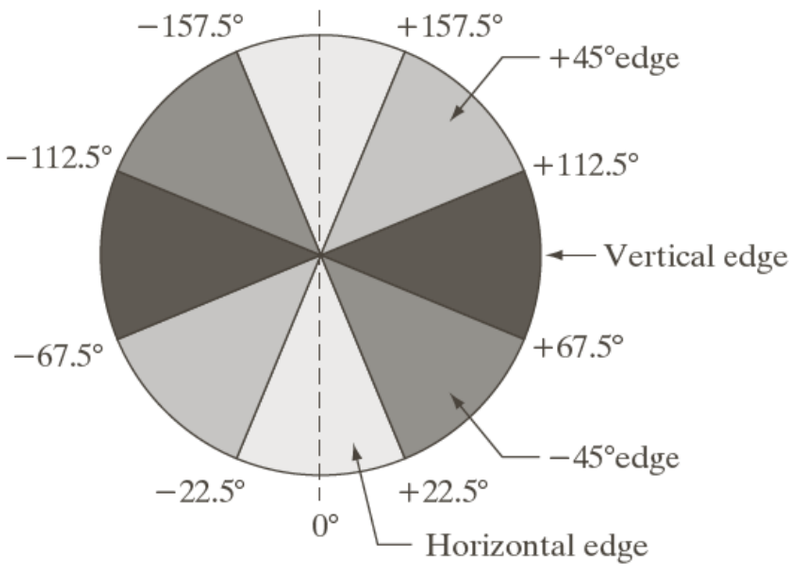
- Convolve image with derivative of Gaussian operators (dG/dx, (dG/dy)

- Find the gradient direction at each pixel; quantize into one of four directions (north-south, east-west, northeast-southwest, northwest-southeast)

- If magnitude of gradient is larger than the two neighbors along this direction, it is a candidate edge point





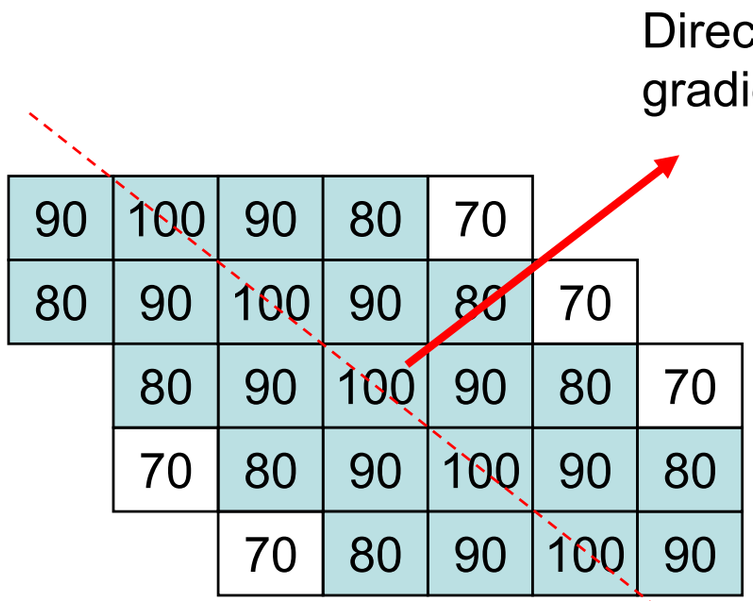


a b
c

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

Edge Thinning

- Take only the local maximum in the direction of the gradient
- Matlab's `edge` function does this



These points are local maxima in the direction of the gradient

```
ang = atan2(fy,fx);
ang2 = floor(8*(ang+pi)/(2*pi));
imtool(ang2,[]);
```

```
E = edge(uint8(I),'sobel');
```

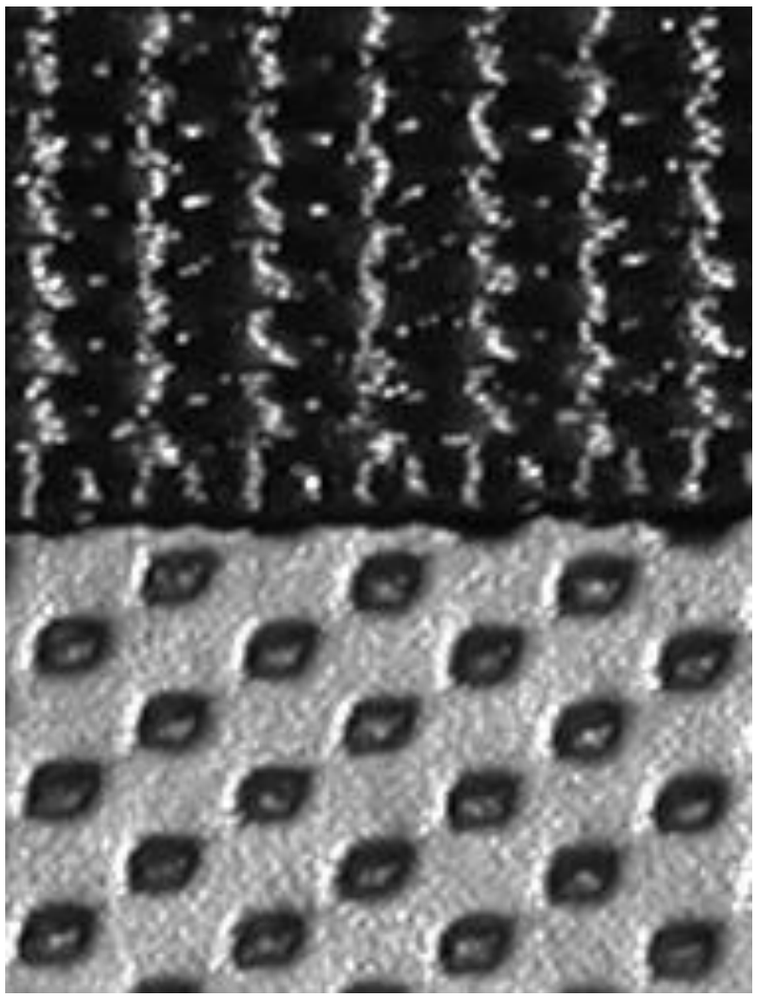
$$\theta = \tan^{-1}\left[\frac{\partial f / \partial y}{\partial f / \partial x}\right]$$

Edge Linking

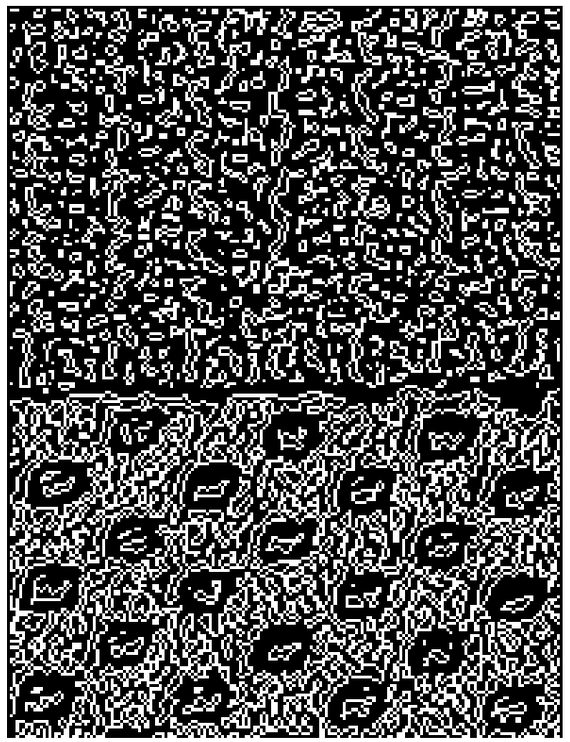
- We want to join edge points into connected curves or lines
  - This facilitates object recognition
- Problem:
  - Some edge points along the curve may be weak, causing us to miss them
  - This would result in a broken curve
- Solution:
  - We use a high threshold to make sure we capture true edge points
  - Given these detected points, link additional edge points into contours using a lower threshold (“hysteresis”)

- Algorithm
  - Find all edge points greater than t_{high}
  - From each strong edge point, follow the chains of connected edge points in both directions perpendicular to the edge normal
  - Mark all points greater than t_{low}

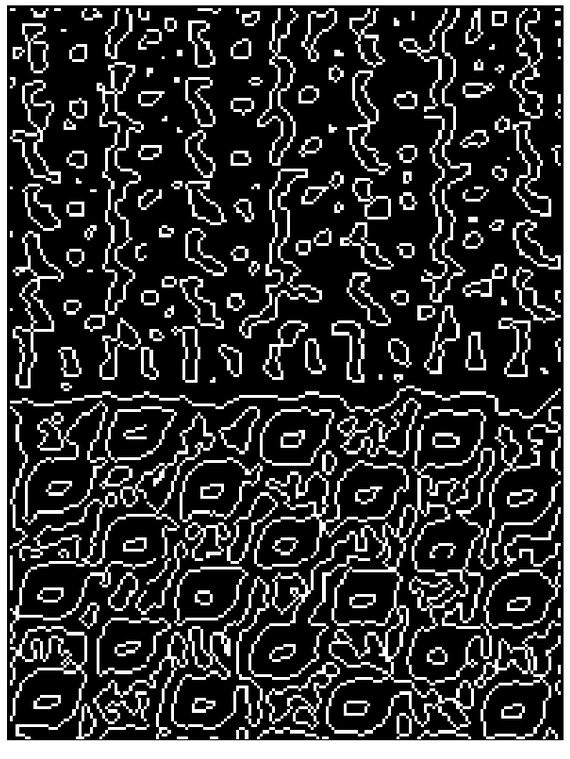
- Image bag.png



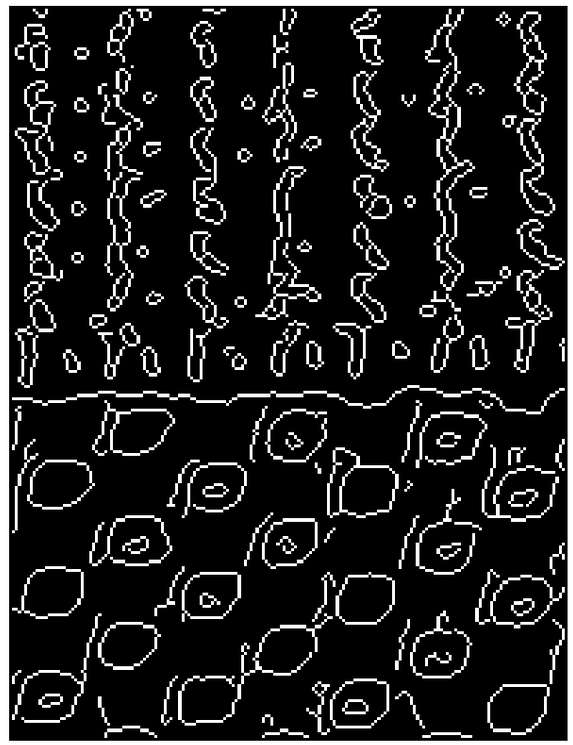
LoG s = 1.000000



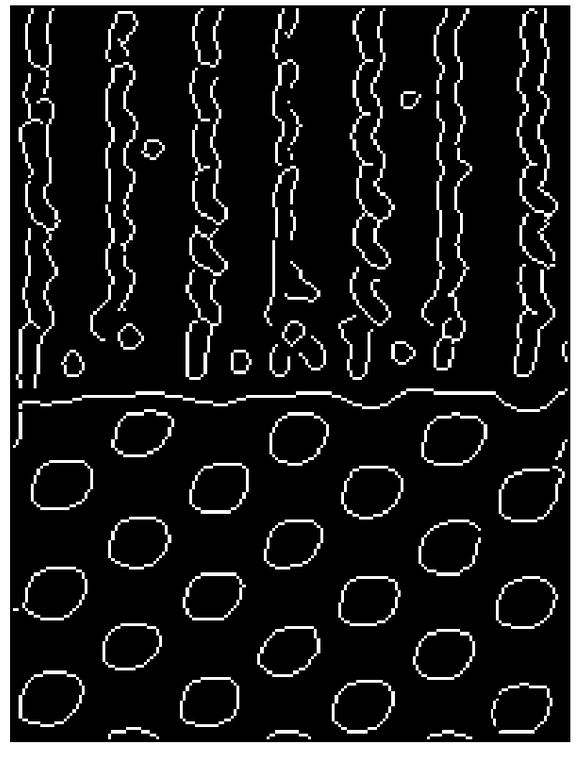
LoG s = 2.000000



Canny s = 1.000000



Canny s = 2.000000



Summary

- Compute the derivative of a gaussian (gradient of a smoothed image).
- Compute the gradient magnitude and gradient angle images (M(x,y) and alfa(x,y) respectively)
- Apply non-maxima suppression to the gradient magnitude image (in the direction of the gradient).
- Use double thresholding and connectivity analysis to detect and link edges (hysteresis).

Details:
Non-maxima suppression

- Find the direction d_k that is closest to the angle image alfa(x,y)

- If the value of the the magnitude image M(x,y) is less than at least one of its two neighbors along d_k, let g_N(x,y) = 0 (suppression), otherwise, let g_N(x,y) = M(x,y)

Details:
Hysteresis thresholding

1. Generate g_NH(x,y) and g_NL(x,y) using a **H**igh and **L**ow threshold.

2. Go through and find the next **unvisited strong edge pixel** p in g_NH(x,y)

3. Mark as **valid edge pixels** all the week pixels in g_NL(x,y) that are connected to p using, say 8-connectivity.

4. If all nonzero pixels in g_NH(x,y) have been visited go to Step 5. Else, return to step 2.

5. Set to zero all pixels in g_NL(x,y) that were not marked as **valid** edge pixels.

6. Append (valid week edge pixels) g_NL(x,y) to g_NH(x,y)

Hough transform

Principle: Voting

Hough transform:
Lines

(Also Circles and General structures)

Hough space

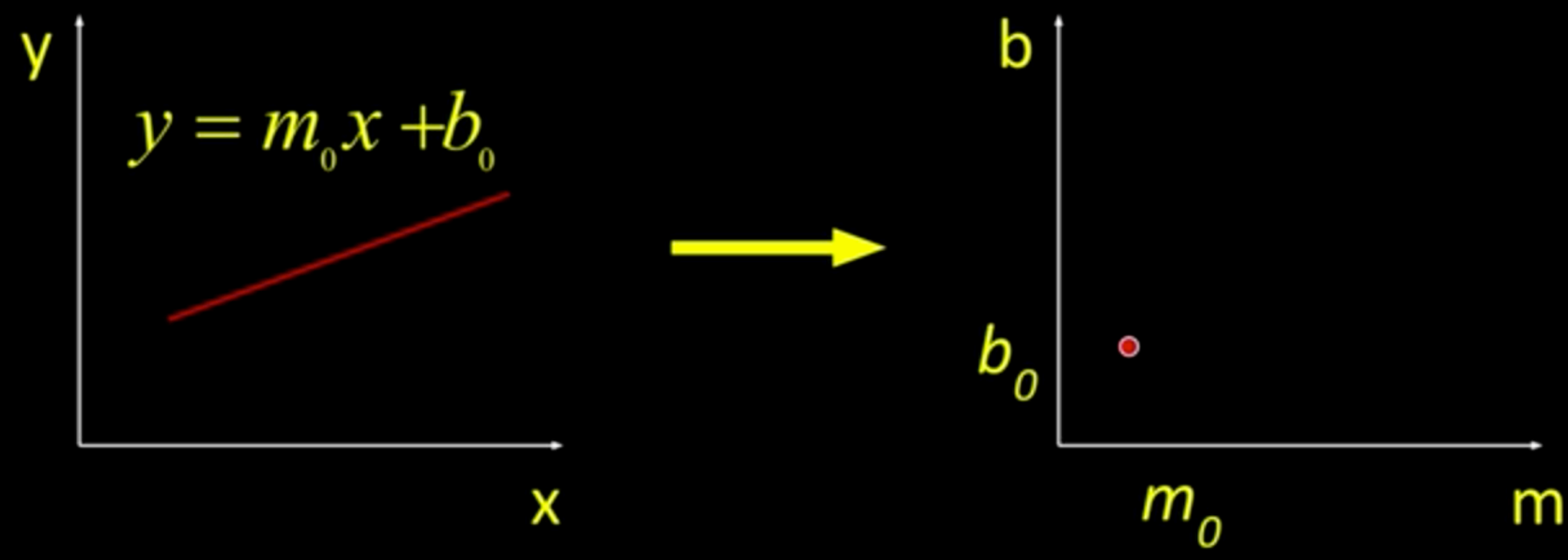
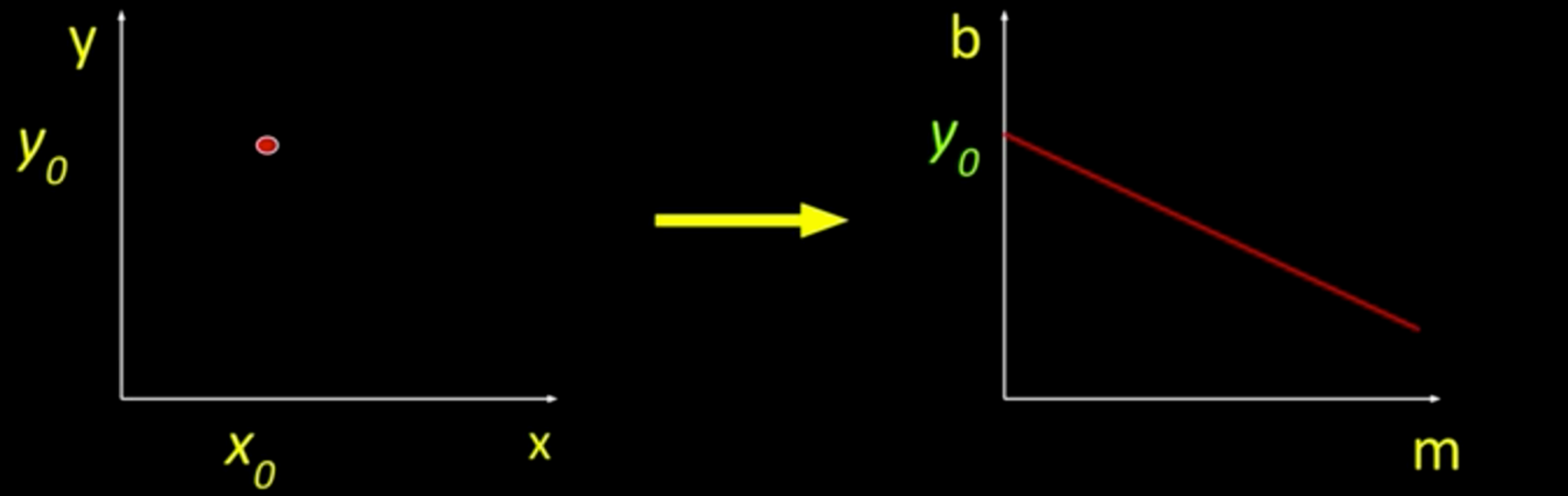


image space

Hough (parameter) space

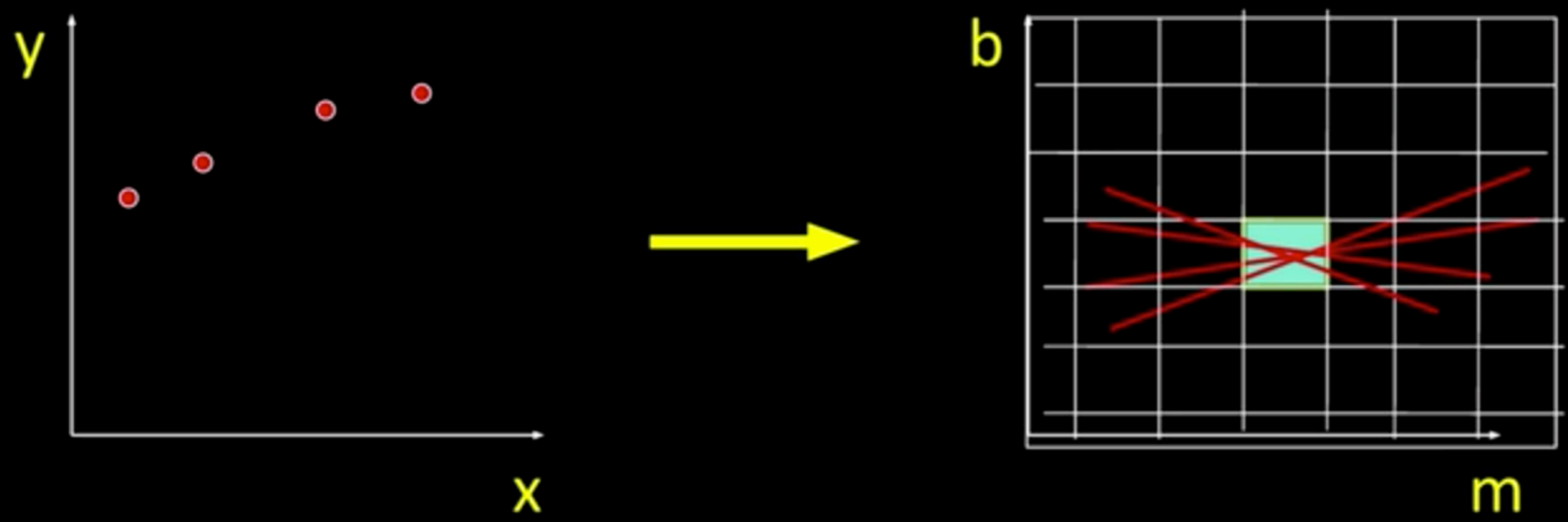
A line in the image corresponds to a point in Hough space

Hough space



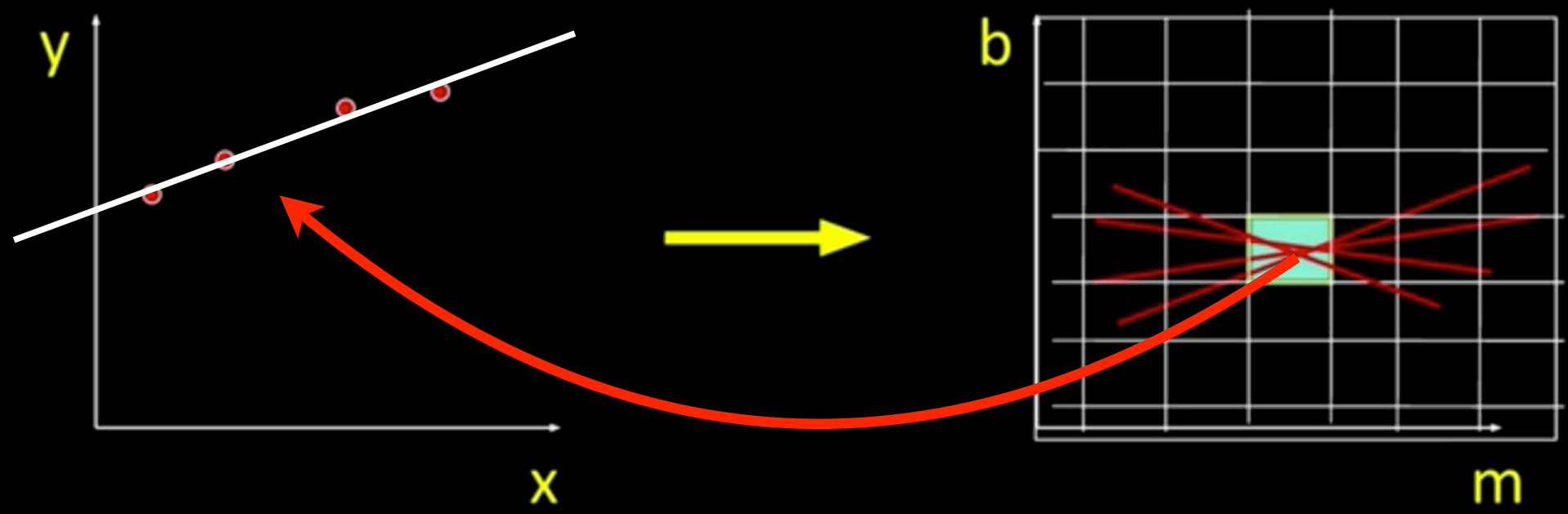
$$y_0 = mx_0 + b \longrightarrow b = -x_0 m + y_0$$

Hough algorithm



- Let each edge point in image space vote for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.

Hough algorithm



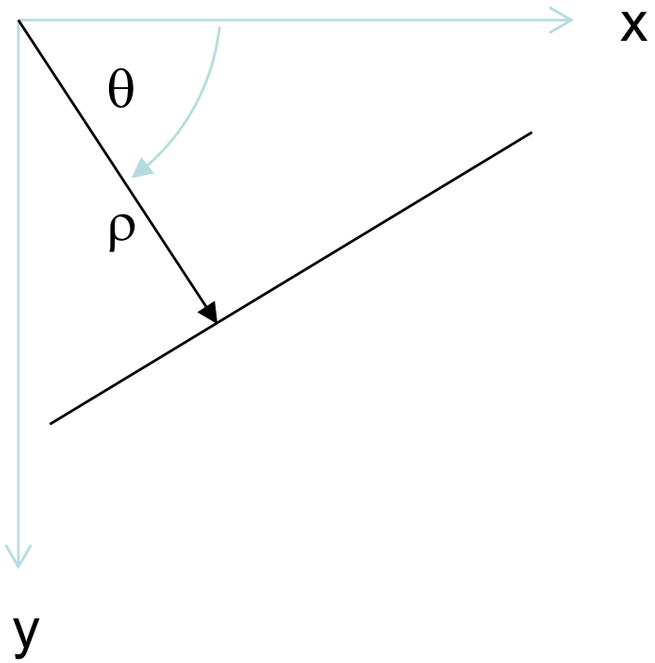
- Let each edge point in image space vote for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.

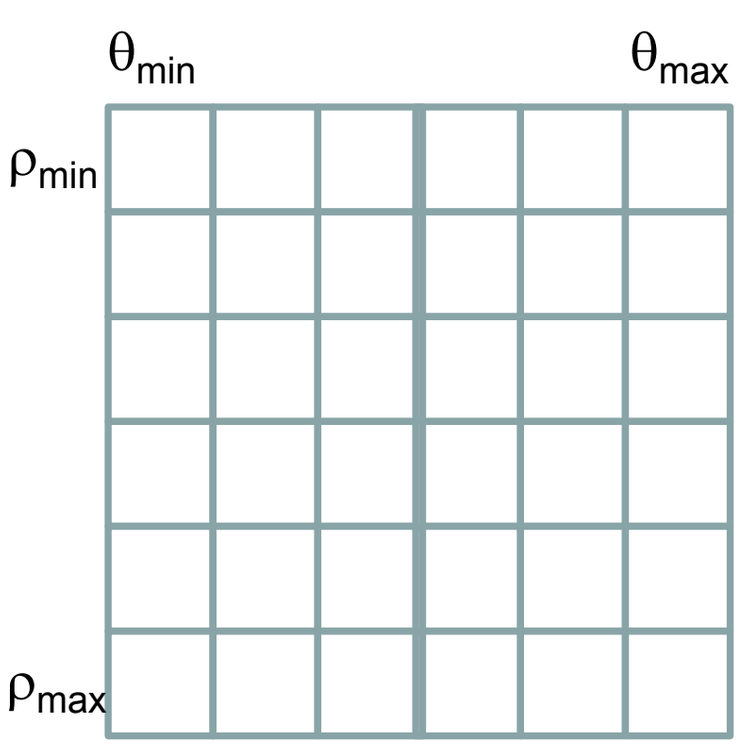
What about vertical lines?

Polar Coordinate Representation of Line

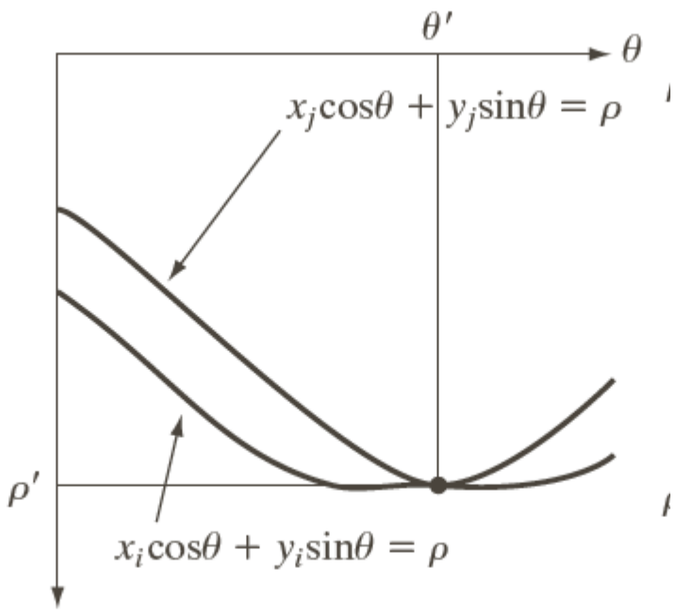
- $\rho = x \cos\theta + y \sin\theta$
  - Avoids infinite slope
  - Constant resolution

The parameter space transform of a point is a sinusoidal curve





A(ρ,θ)



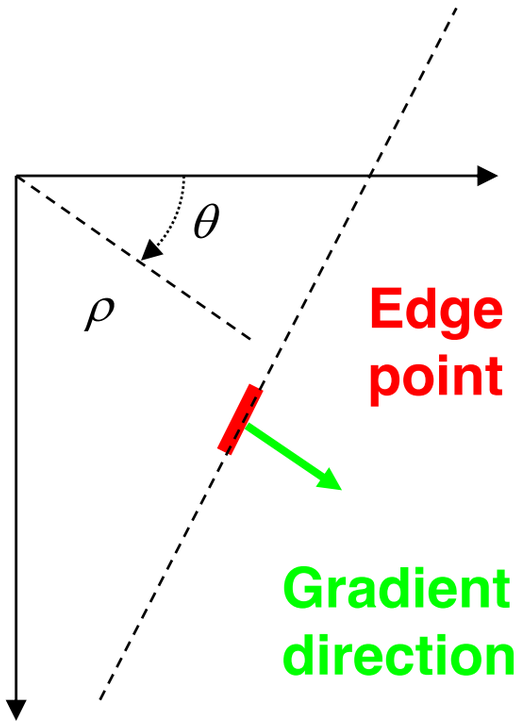
Pseudo Code

```
for all x
    for all y
        if edge point at (x,y)
            for all theta
                rho = x*cos(theta) + y*sin(theta)
                increment (add 1 to) the cell in H corresponding to (theta,rho)
            end
        end
    end
end
```

Extension to use Gradient Direction

- If you know the gradient direction at each edge point, it reduces the complexity of the Hough transform

- Namely, the angle of the gradient already gives the angle of the line



- This eliminates one “for” loop – the loop over the angles

- Potential problem
  - Image noise can cause uncertainty in the gradient angle
  - You can increment multiple cells corresponding to several angles near the estimated angle





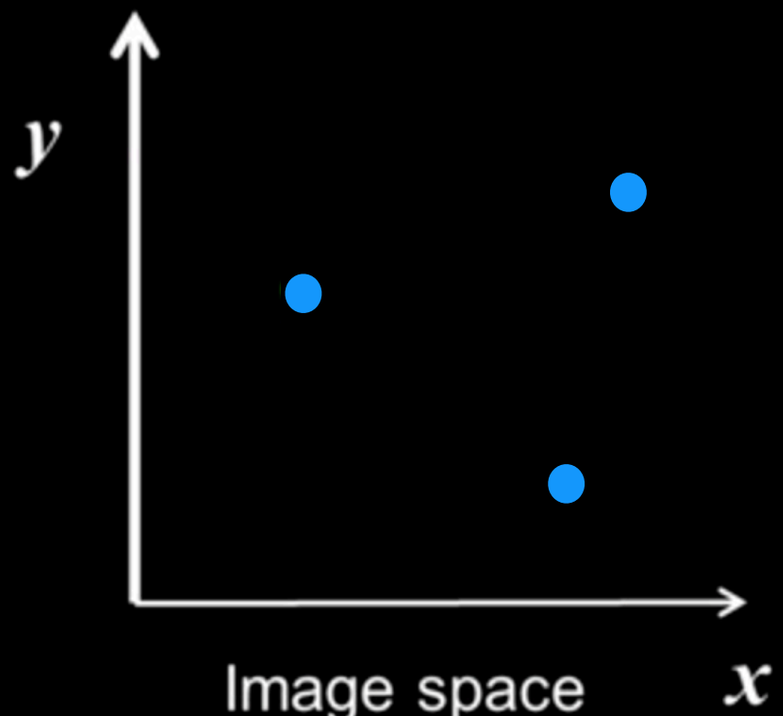
a b
c d e

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

Hough transform:
Circles

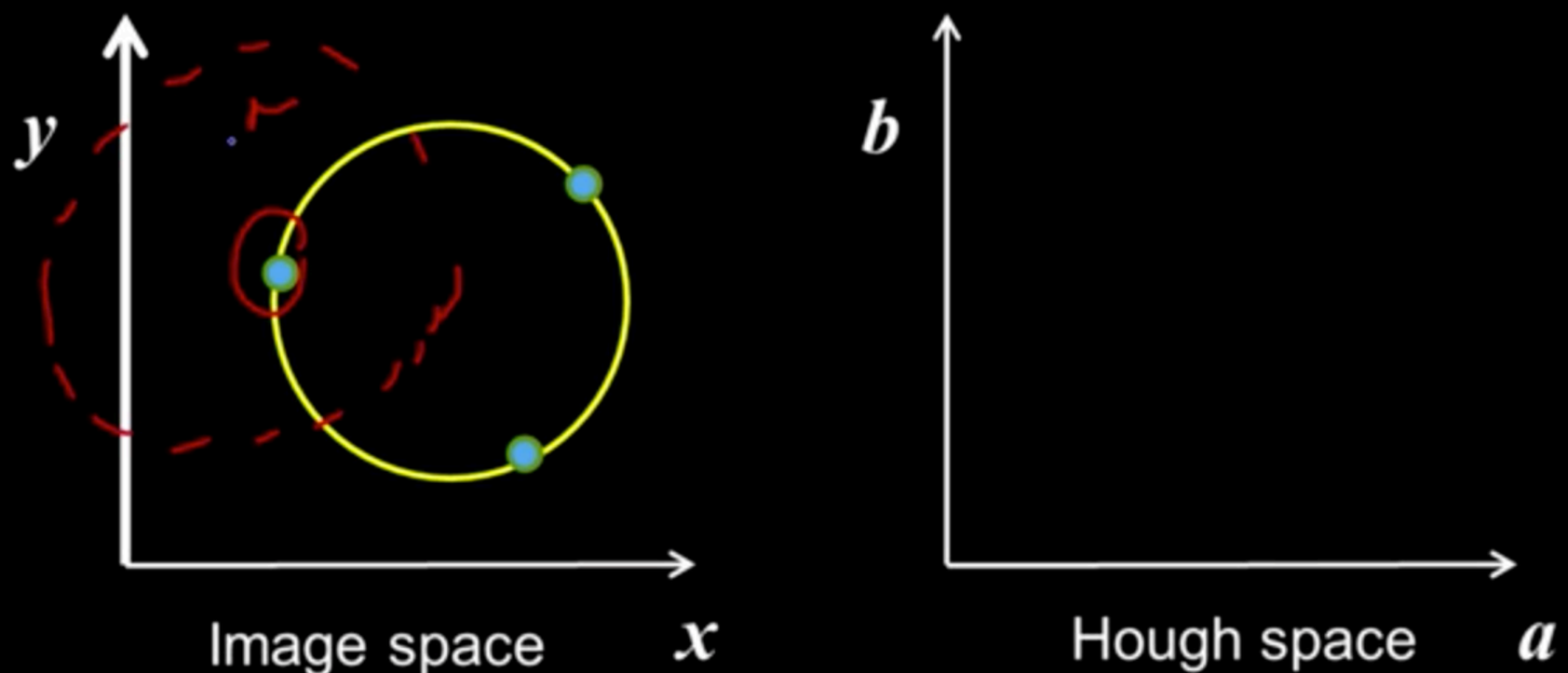
Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For a fixed radius r, unknown gradient direction:



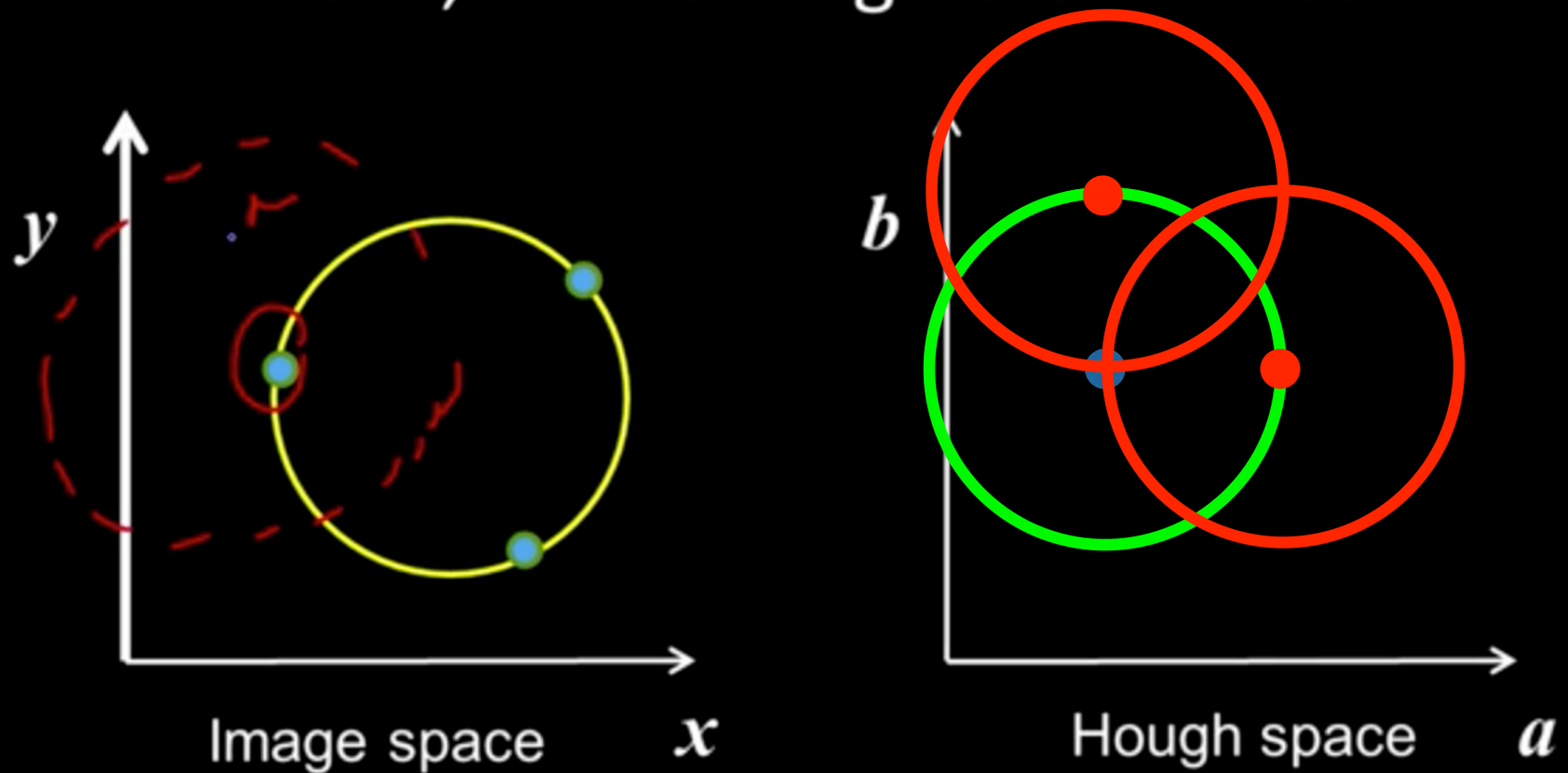
Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For a fixed radius r, unknown gradient direction:



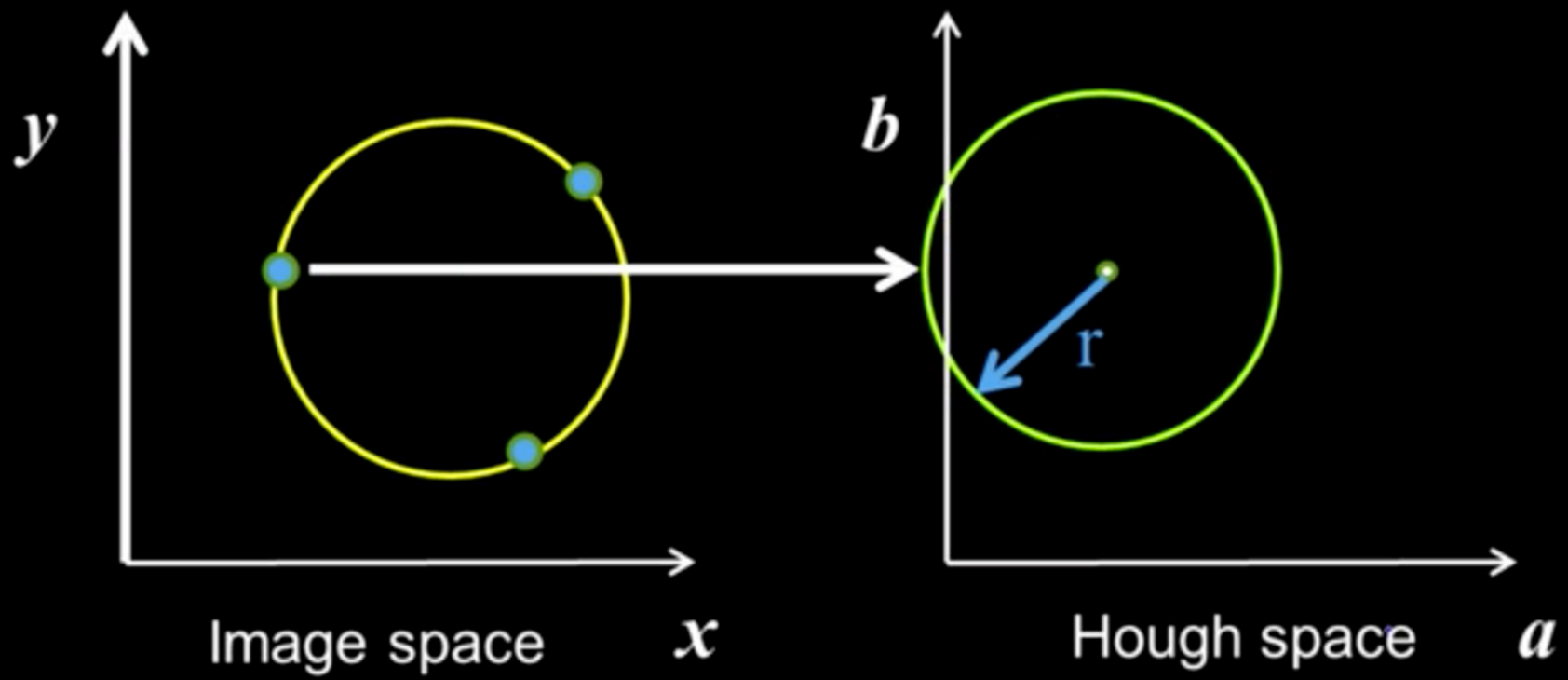
Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For a fixed radius r, unknown gradient direction:



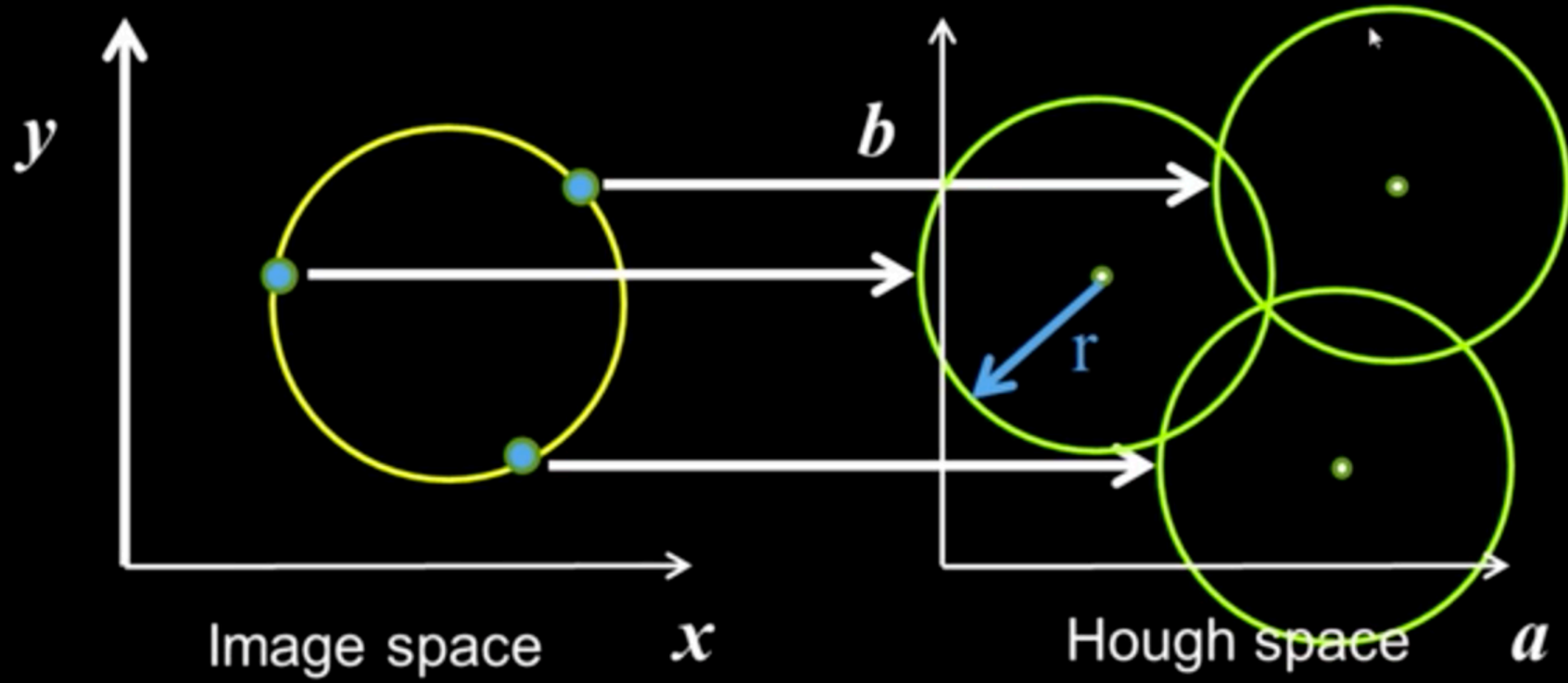
Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For a fixed radius r, unknown gradient direction:



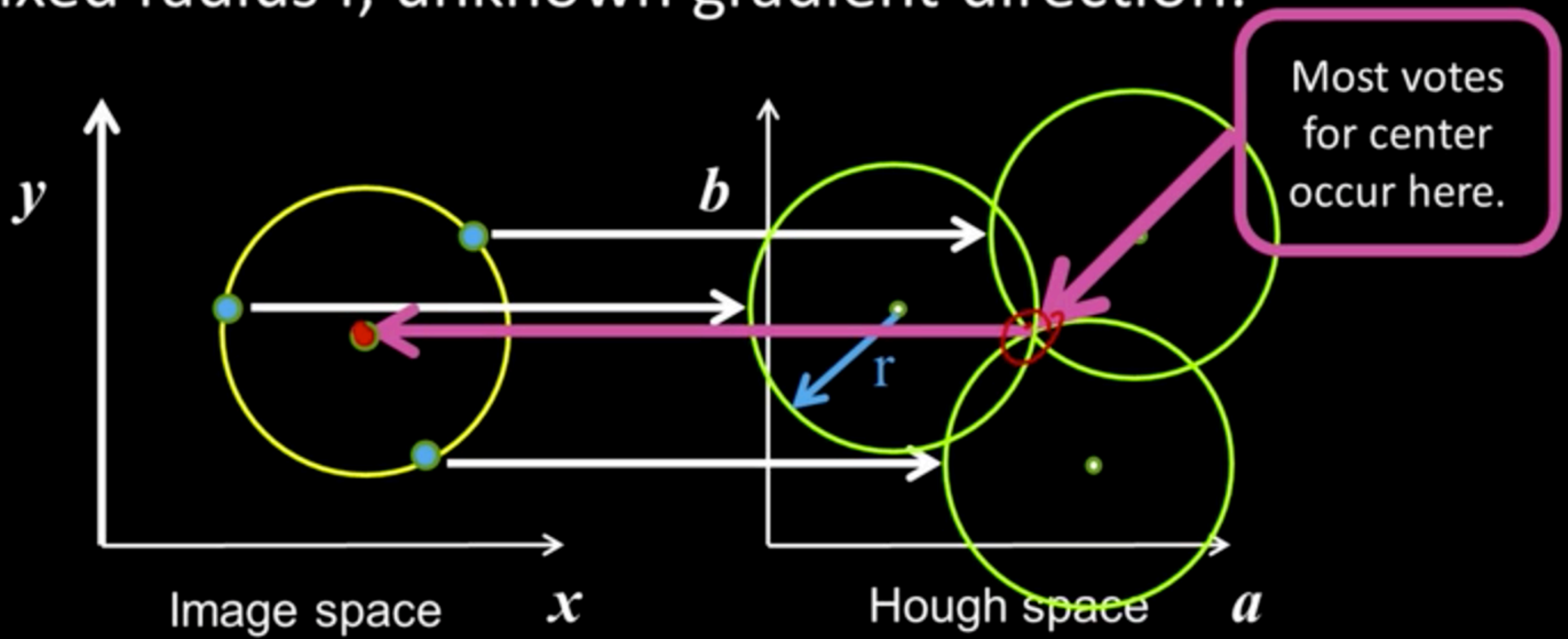
Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For a fixed radius r, unknown gradient direction:



Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For a fixed radius r, unknown gradient direction:

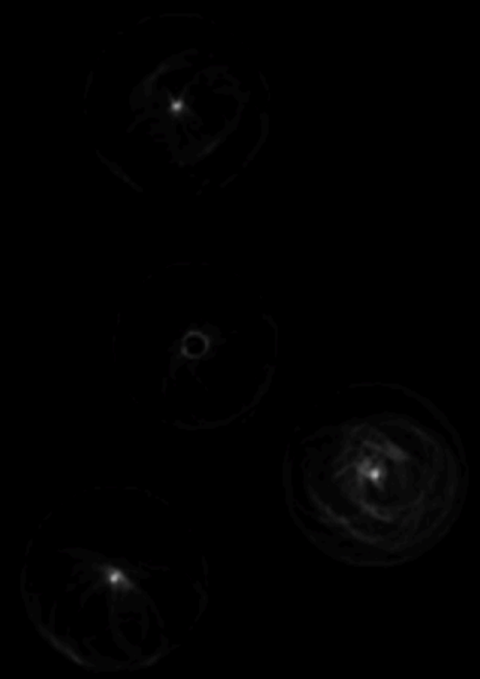


Example: detecting circles with Hough

Original

Edges

Votes: Quarter



Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For ***unknown*** radius r, no gradient:

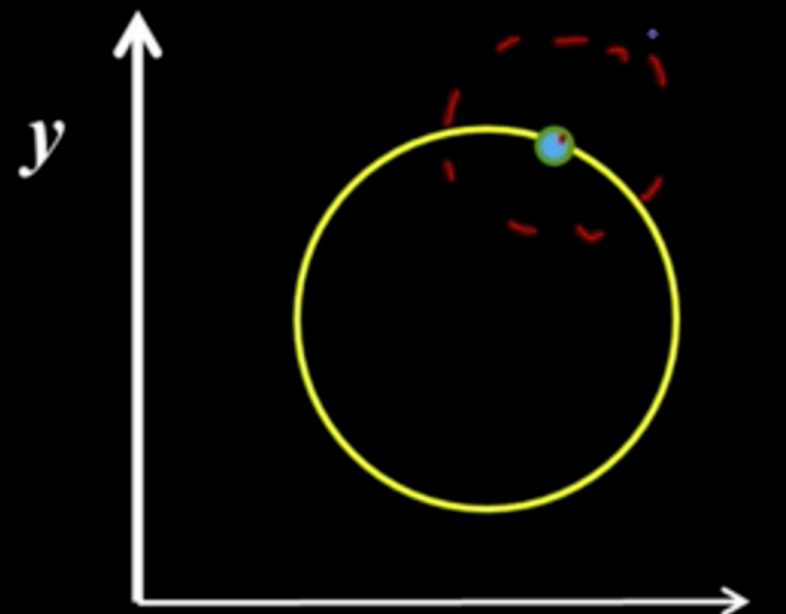
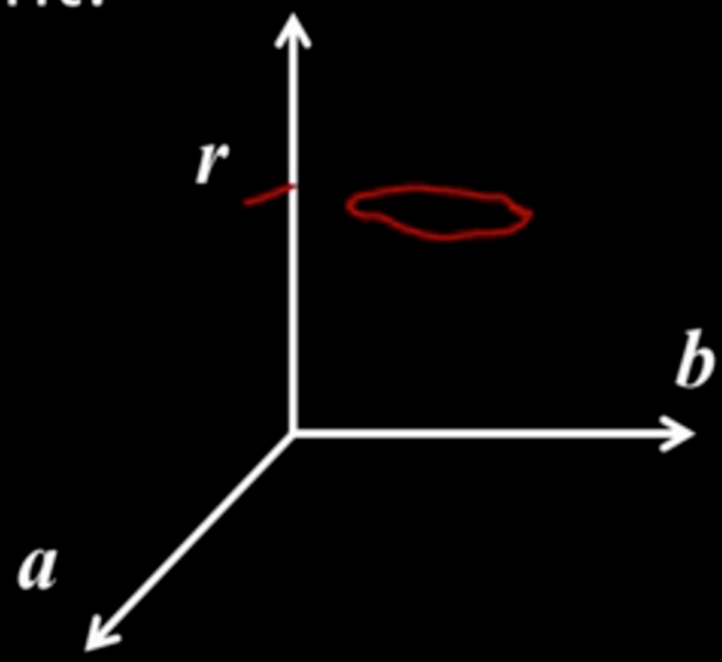


Image space



Hough space

Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For ***unknown*** radius r, no gradient:

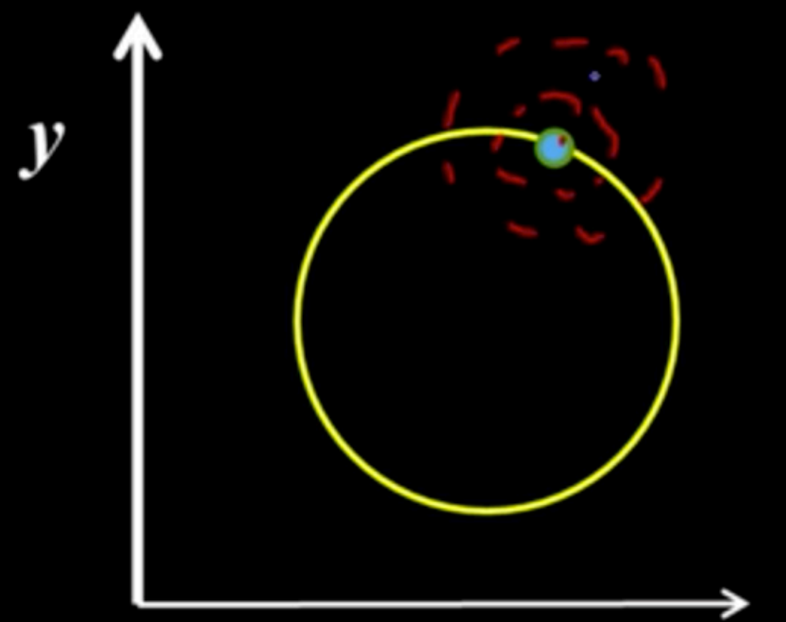
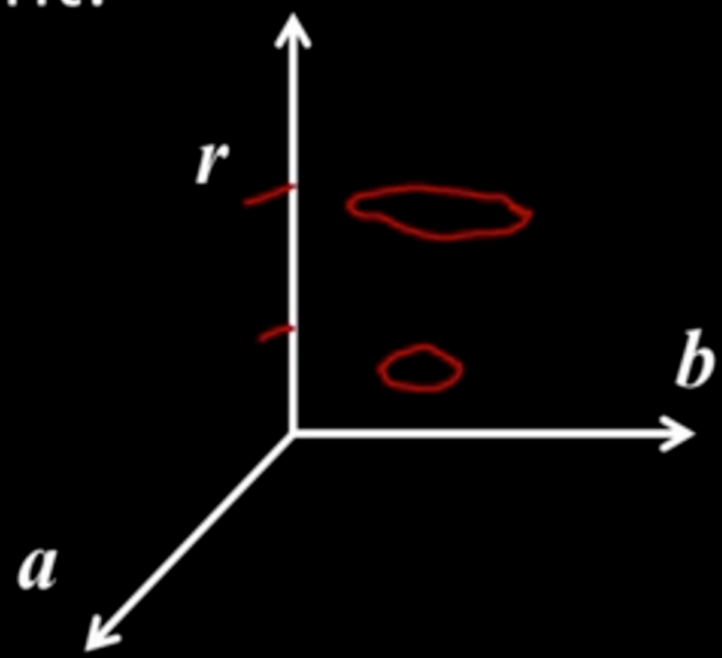


Image space



Hough space

Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For ***unknown*** radius r, no gradient:

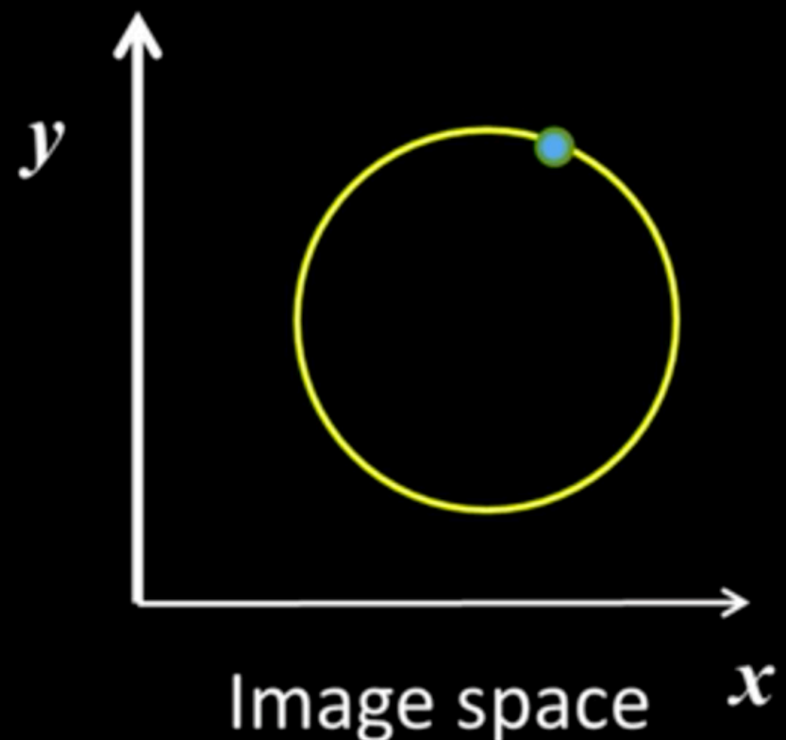
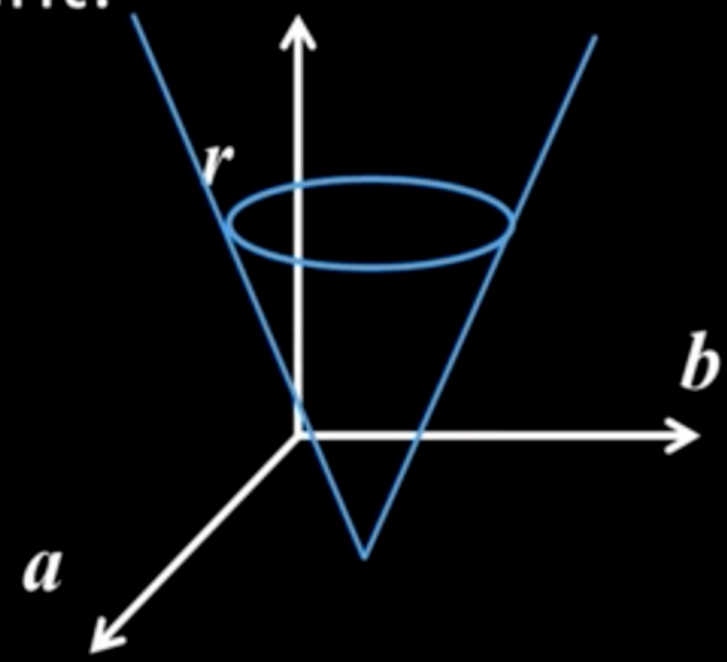


Image space



Hough space

Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For ***unknown*** radius r, no gradient:

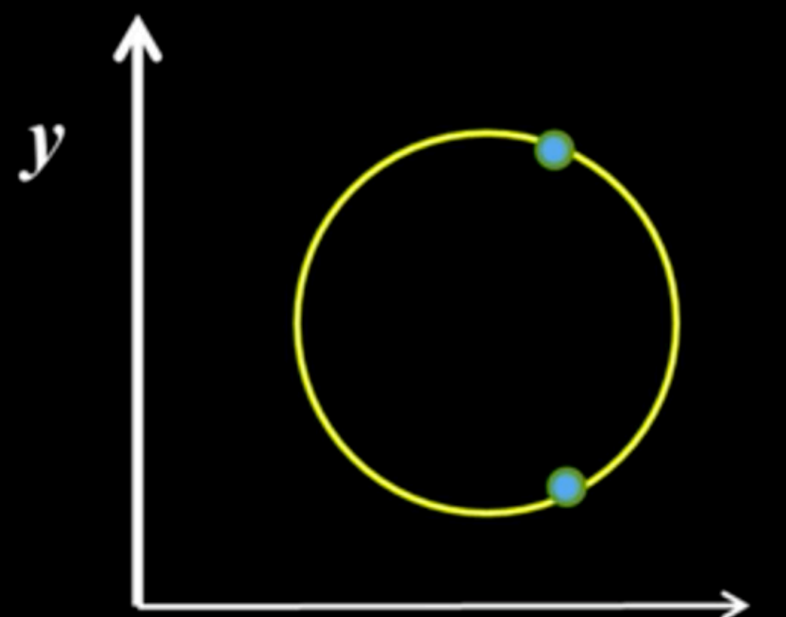
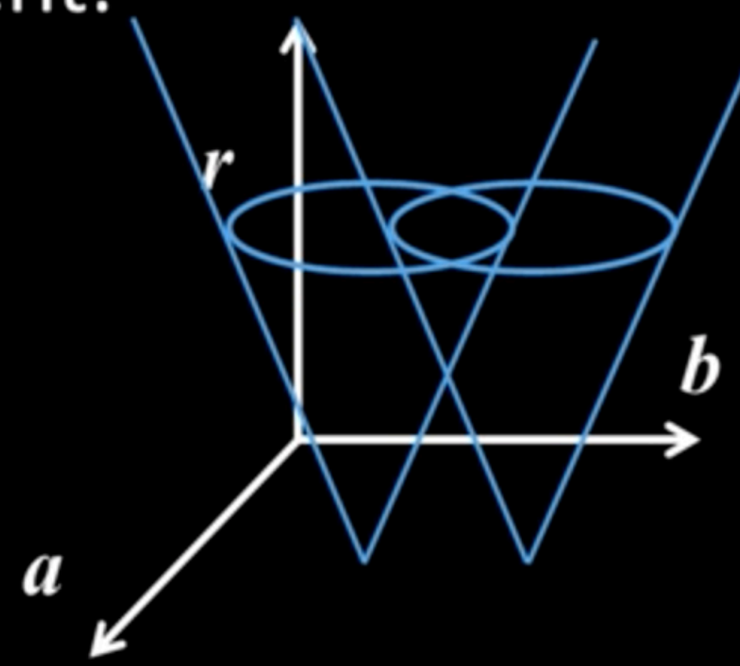


Image space



Hough space

Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For ***unknown*** radius r, with ***gradient***:

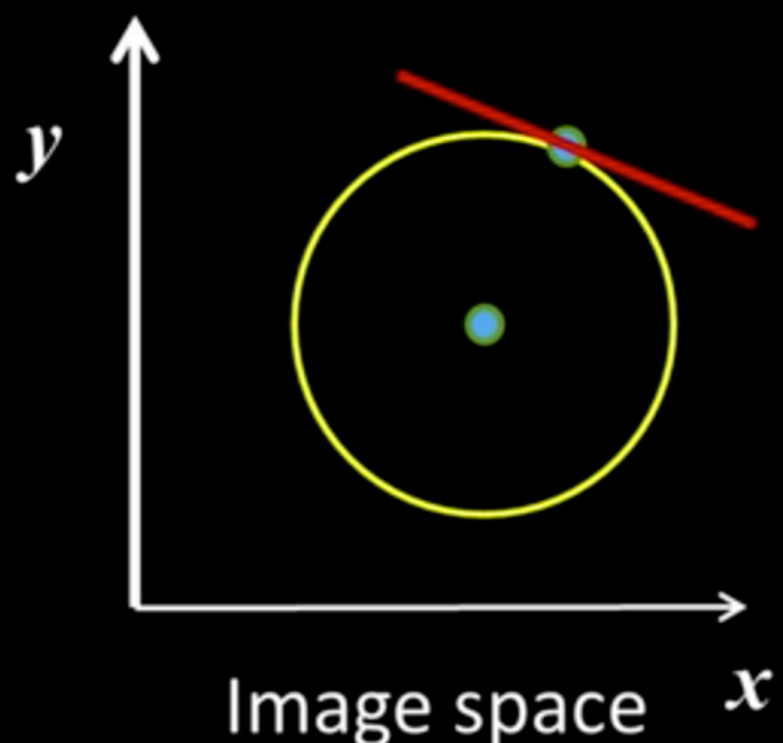
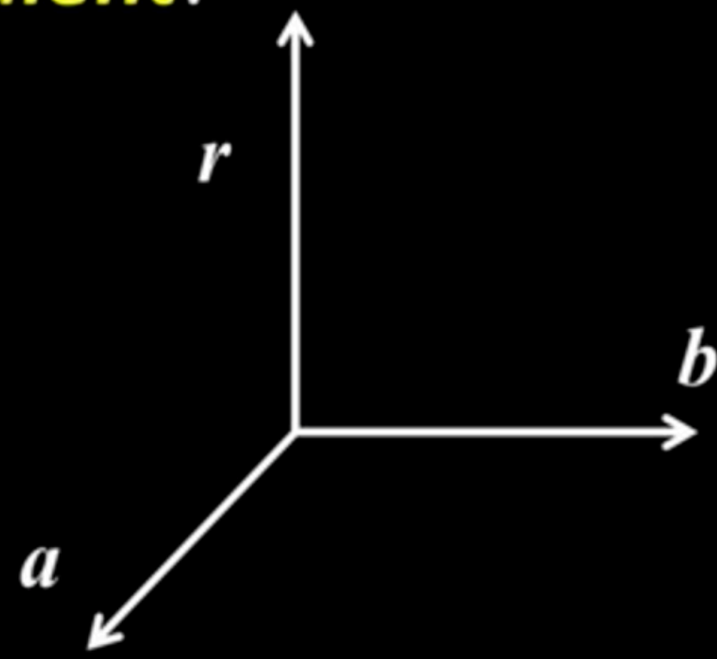


Image space



Hough space

Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For ***unknown*** radius r, with ***gradient***:

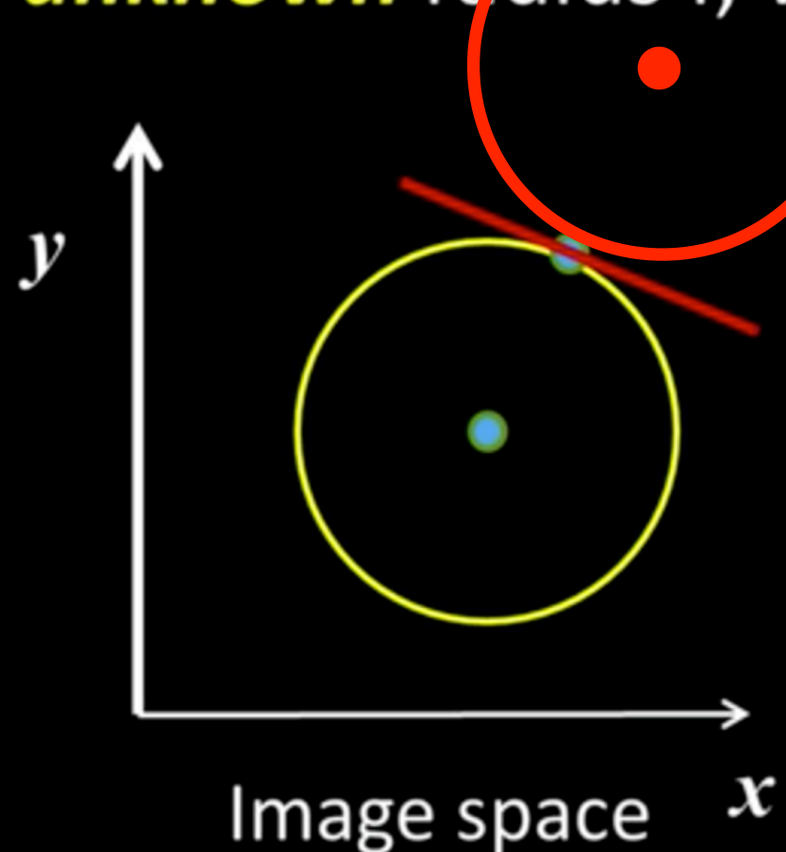
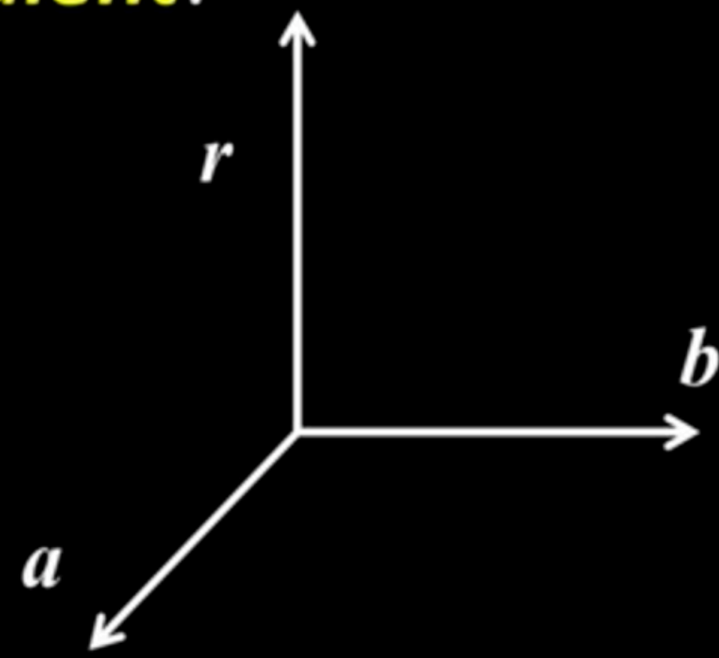


Image space



Hough space

Hough transform for circles

- Circle: center (a,b) and radius r: $(x_i - a)^2 + (y_i - b)^2 = r^2$
- For ***unknown*** radius r, with ***gradient***:

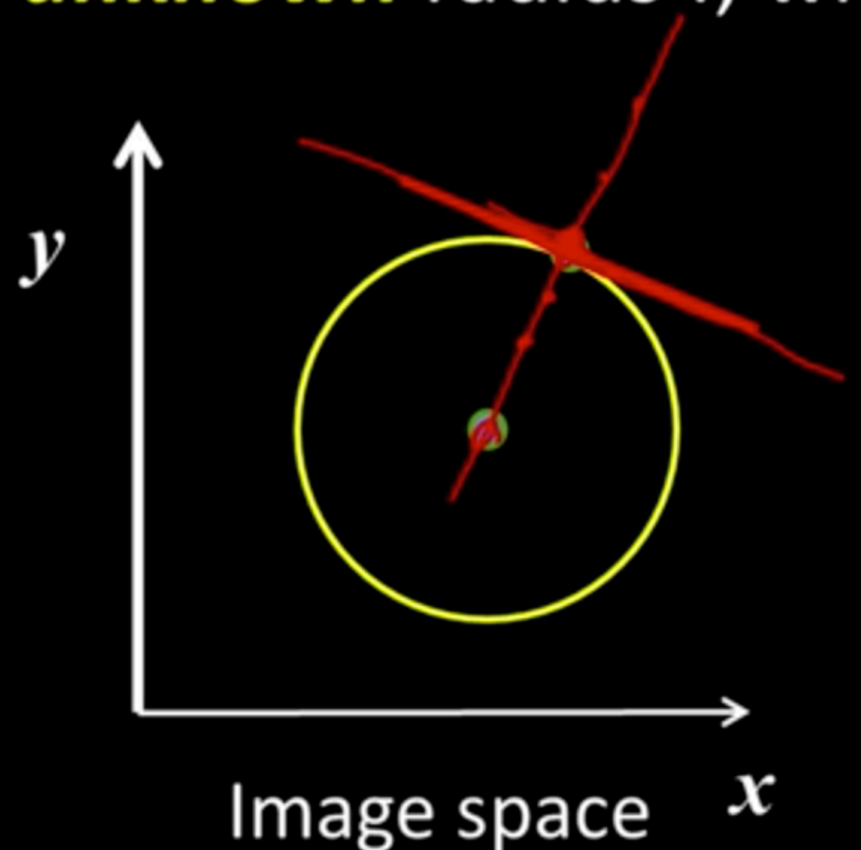
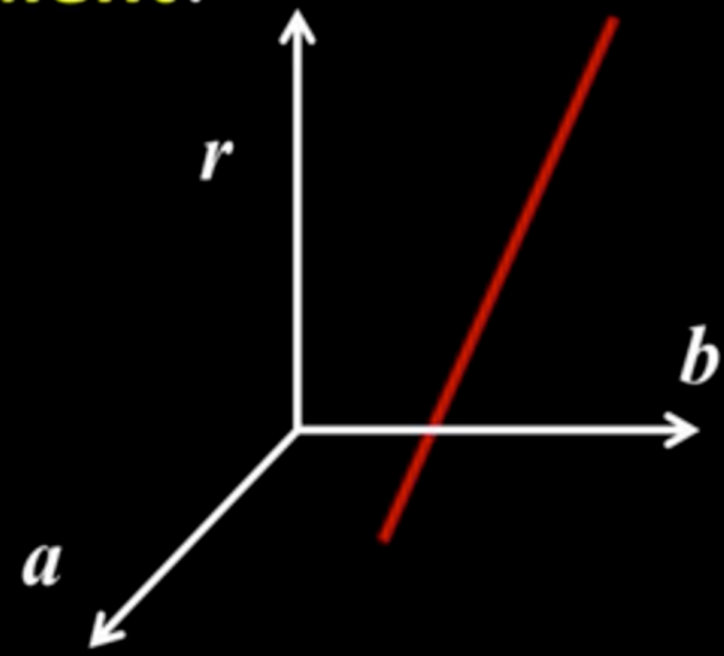


Image space



Hough space

Hough transform:

?

Hough transform:

?

Can be used to find any parameterized curve

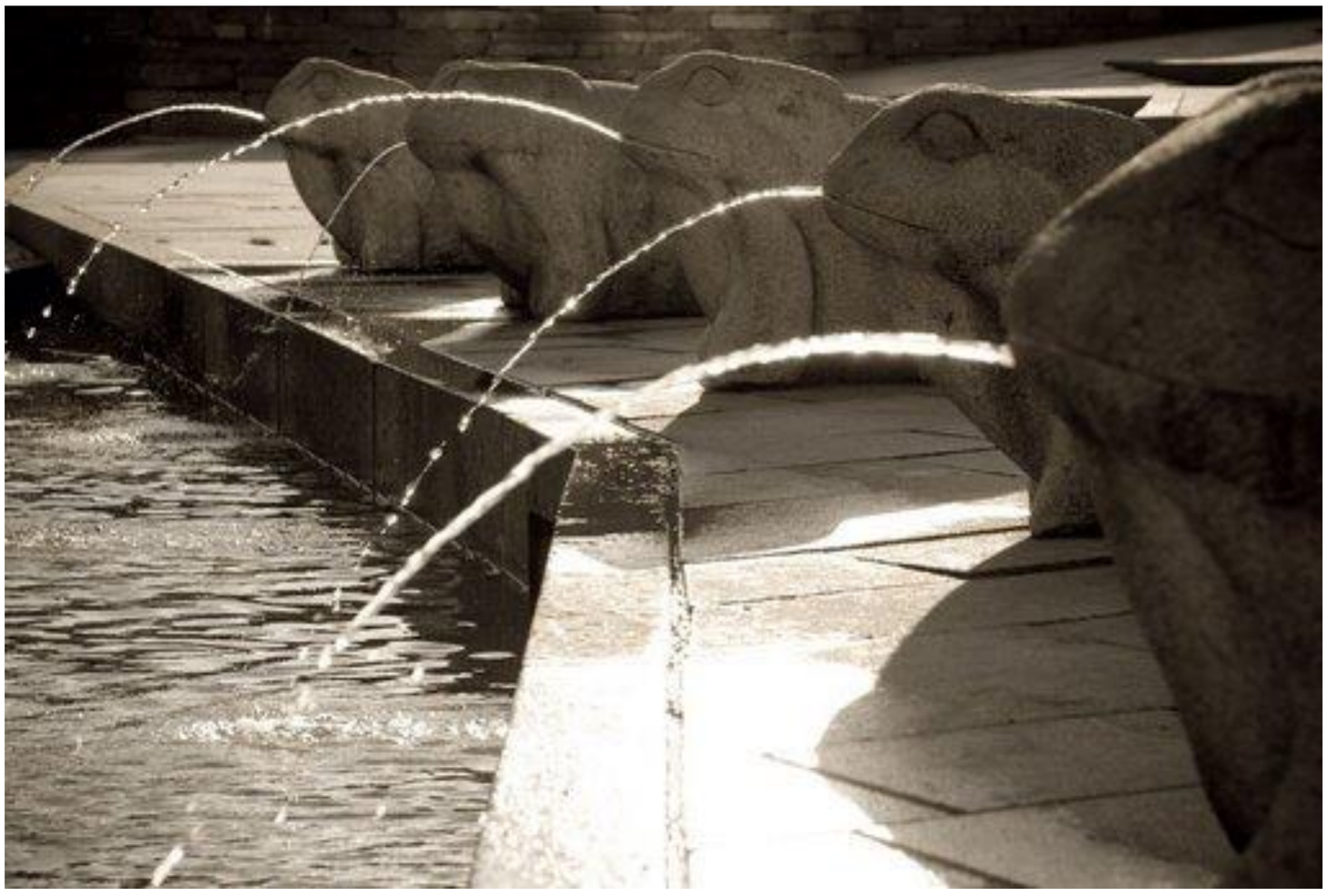
Example – detecting parabolas

- A parabola centered at (x0,y0) has the equation

$$y - y_0 = a\left(x - x_0\right)^2$$

- So we need to search for three parameters: x0,y0,a

- Can limit range of "a" to search over



Grenouilles.jpg

Figure 1

File Edit View Insert Tools Desktop Window Help

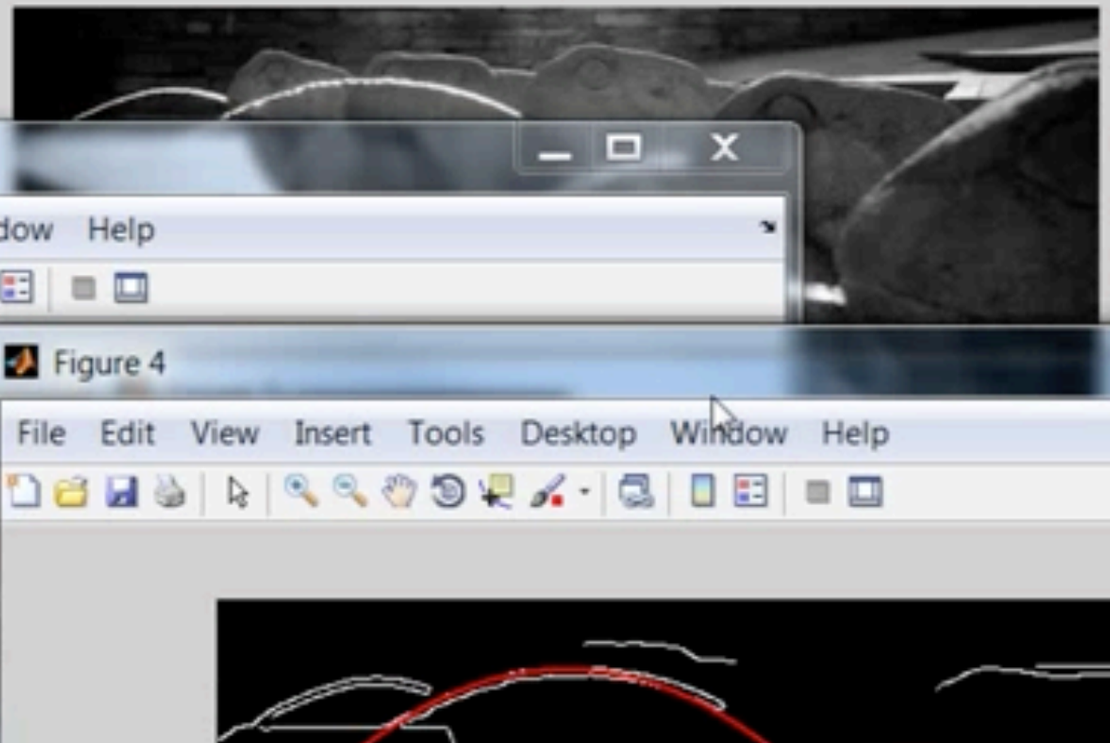


Figure 2

File Edit View Insert Tools Desktop Window Help

Figure 4

File Edit View Insert Tools Desktop Window Help

Start Slide Show

Outline

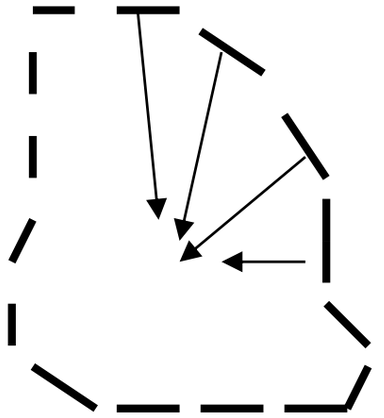
end

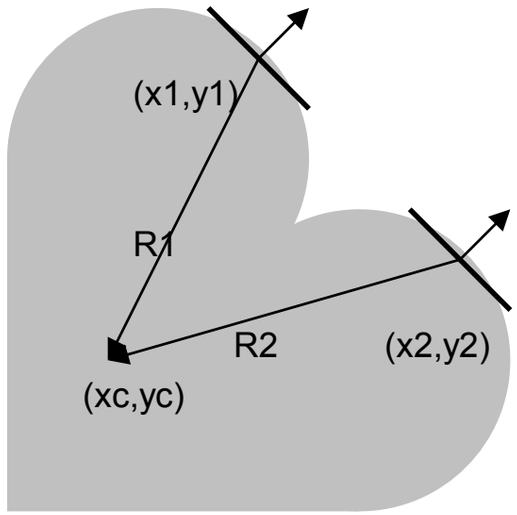
35

Generalized Hough transform

Generalized Hough Transform

- Can find arbitrary shapes
  - The shape is represented by a sequence of boundary points
  - For each point, we specify the distance and angle to the center





The two boundary points (x1,y1) and (x2,y2) have the same gradient direction. The vector from each boundary point to the object's center (xc,yc) is R1 and R2, respectively.

- We create an "R-table"
  - For each gradient direction, store the vector(s) R to the object

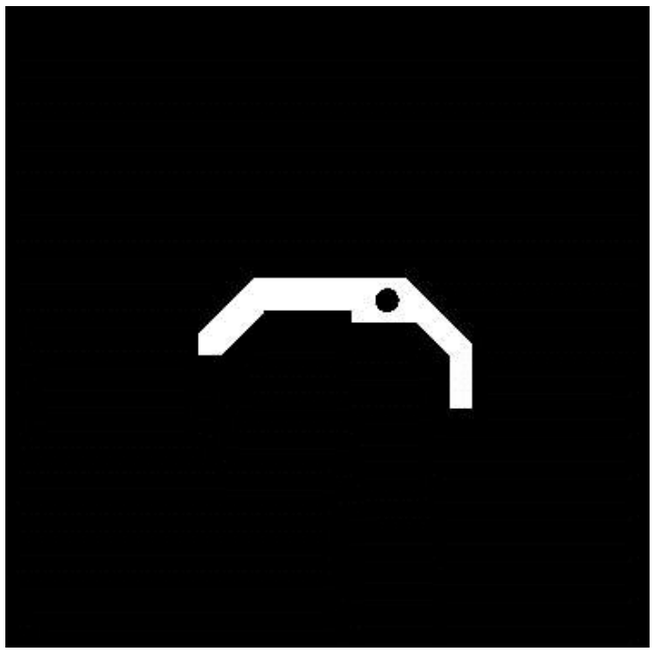
Gradient angle	Vectors pointing to object origin
θ1	R1,R2
θ2	R3,R4,R5,…
:	
θN	…

Generalized Hough Transform

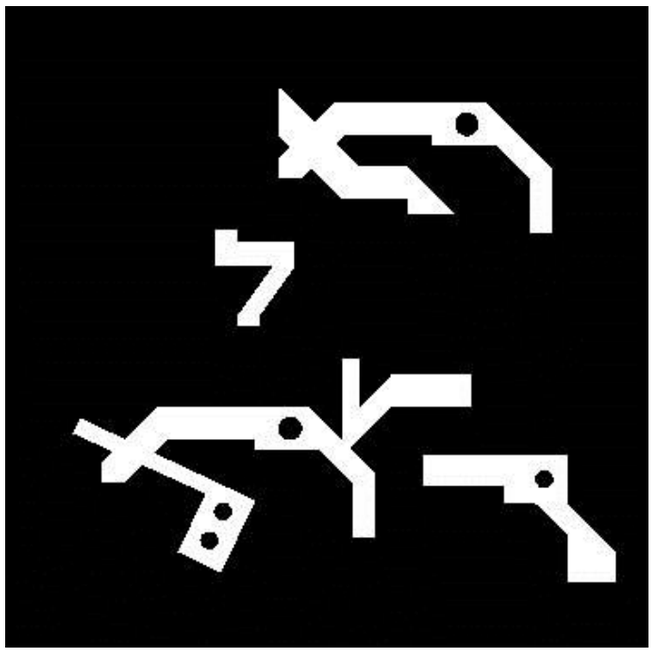
- To find the shape
  - Compute the gradient direction at each edge point
  - Look up the vectors R from the R-table ... these vectors point to the possible location of the object center
  - Increment an accumulator array at those point(s)

- Example

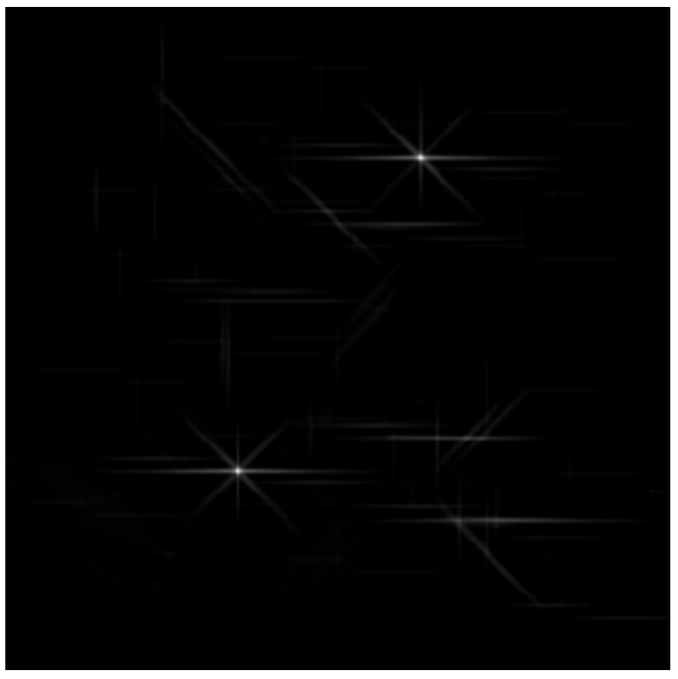
Template object



Test image

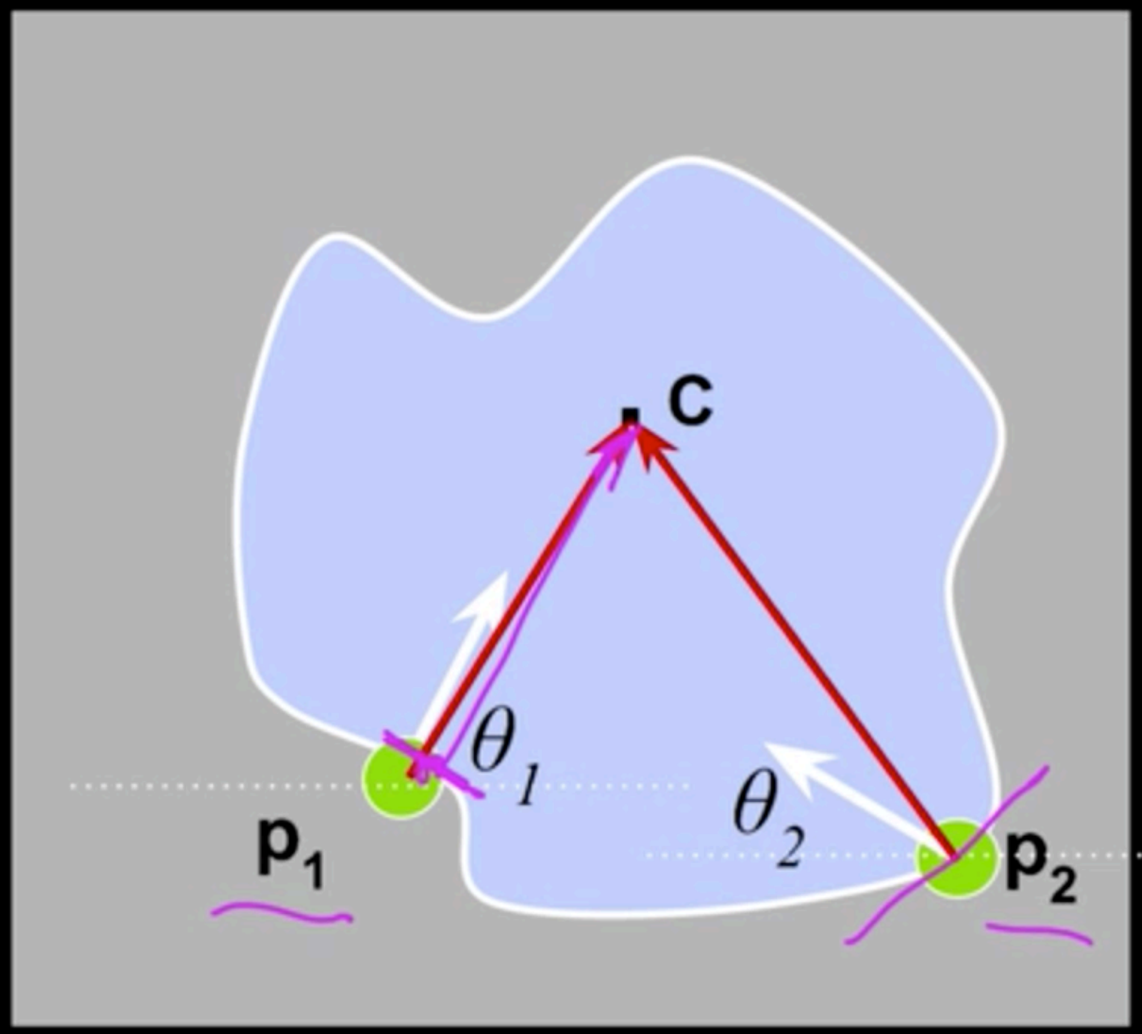


Hough accumulator array



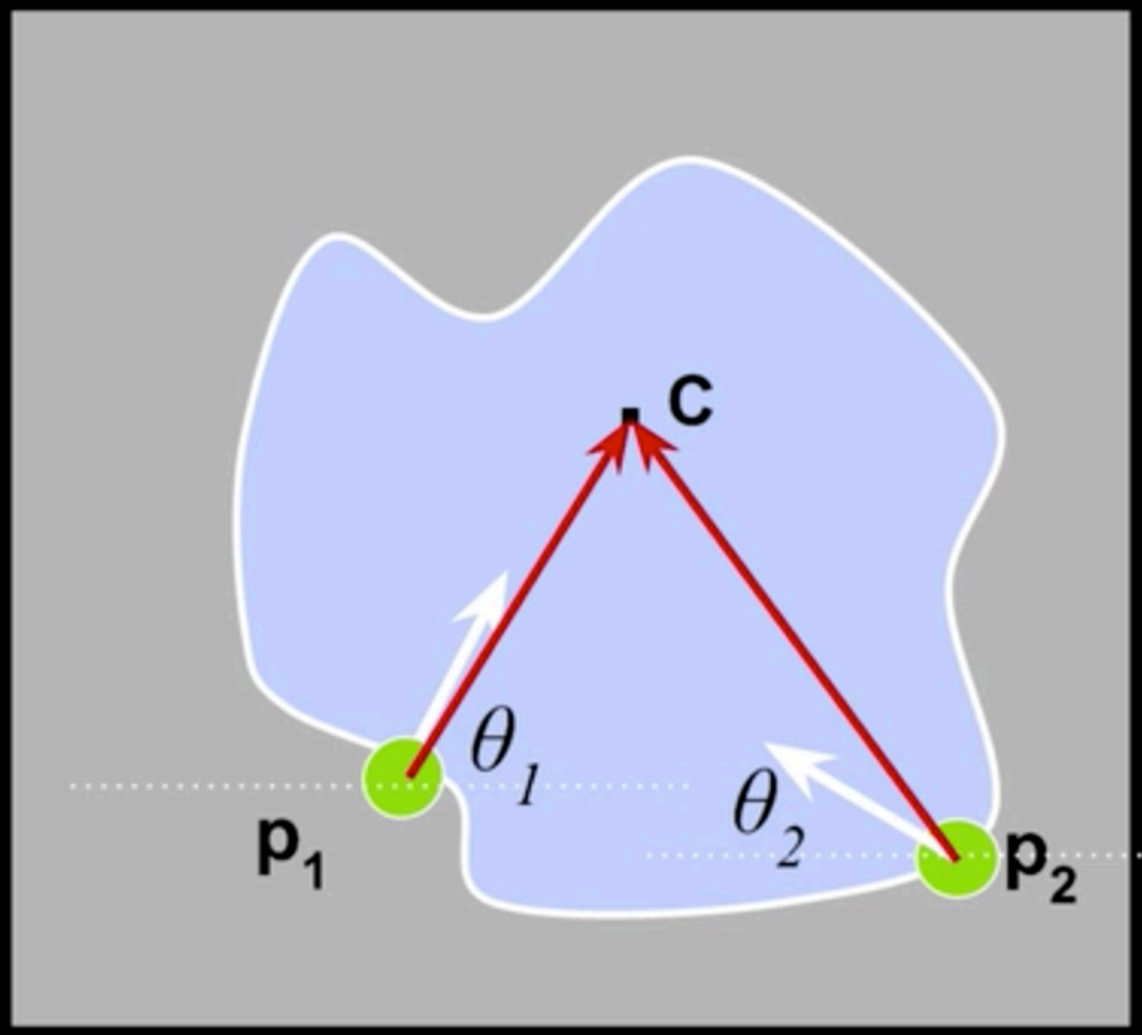
Training: build a Hough table

1. At each boundary point, compute displacement vector: $\mathbf{r} = \mathbf{c} - \mathbf{p}_i$.
2. Measure the gradient angle θ at the boundary point.
3. Store that displacement in a table indexed by θ.



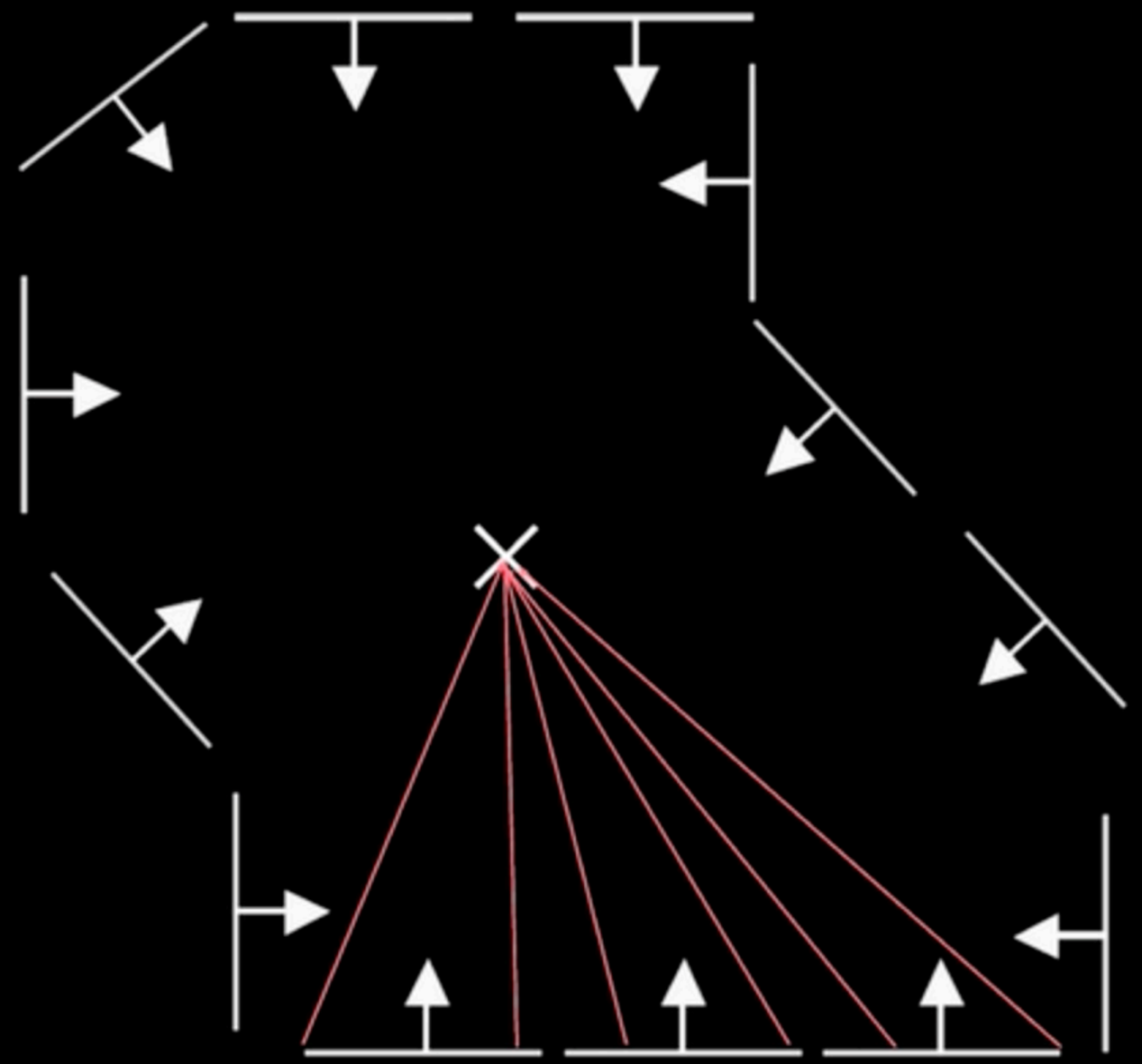
Recognition:

1. At each boundary point, measure the gradient angle θ
2. Look up all displacements in θ displacement table.
3. Vote for a center at each displacement.

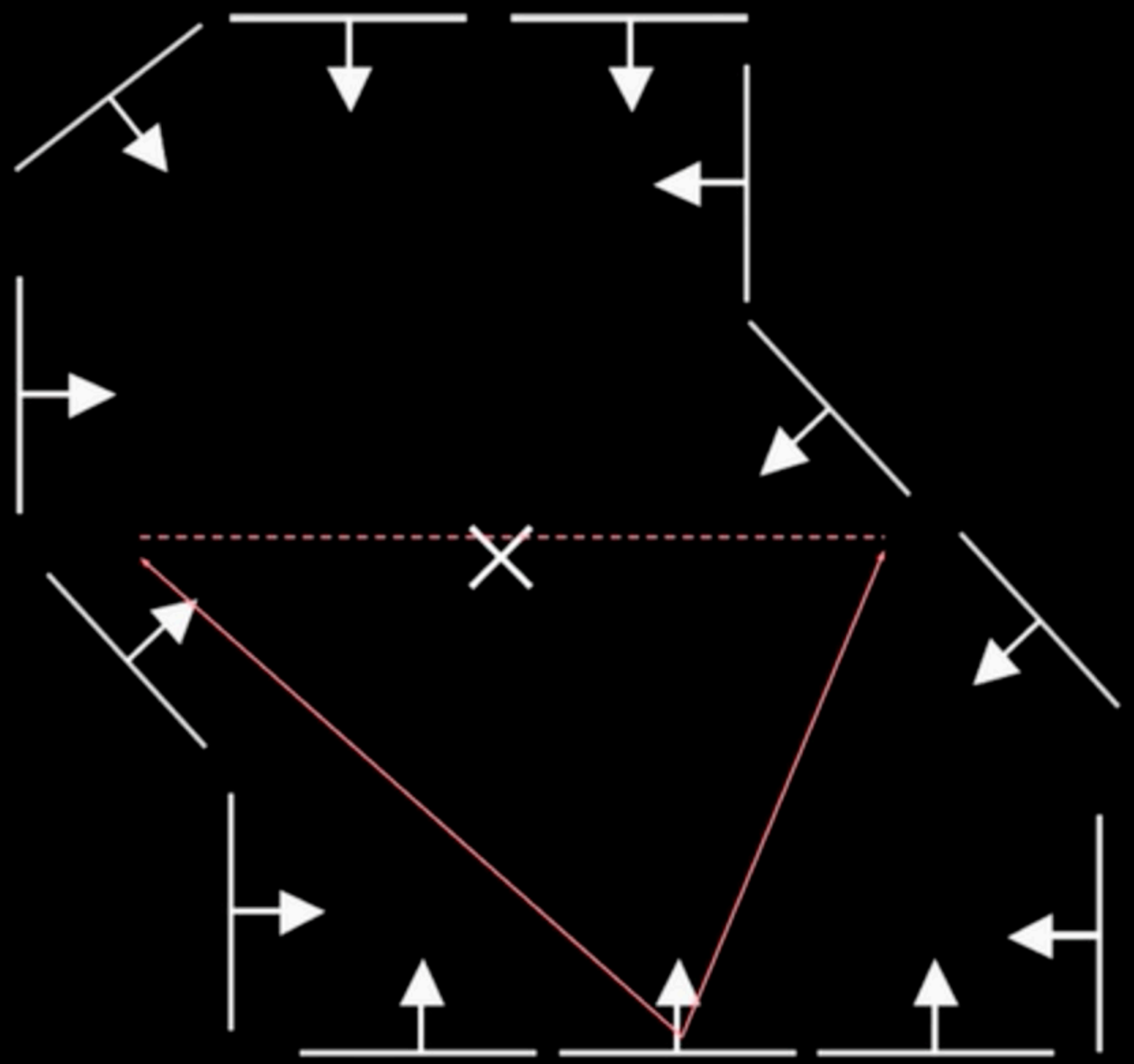


[Dana H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, 1980]

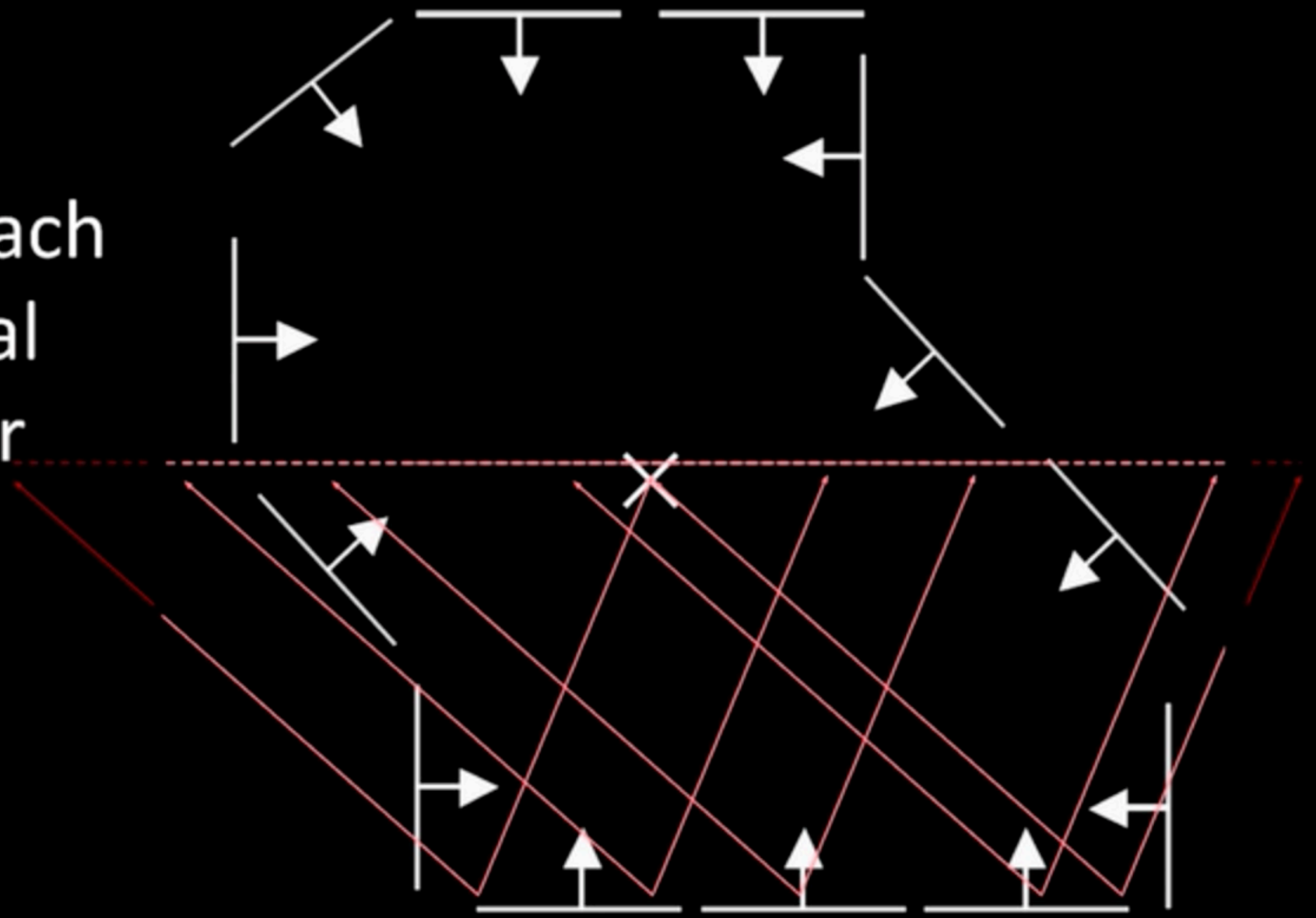
Looking at the bottom horizontal boundary points (all the same θ), the set of displacements ranges over all the red vectors.



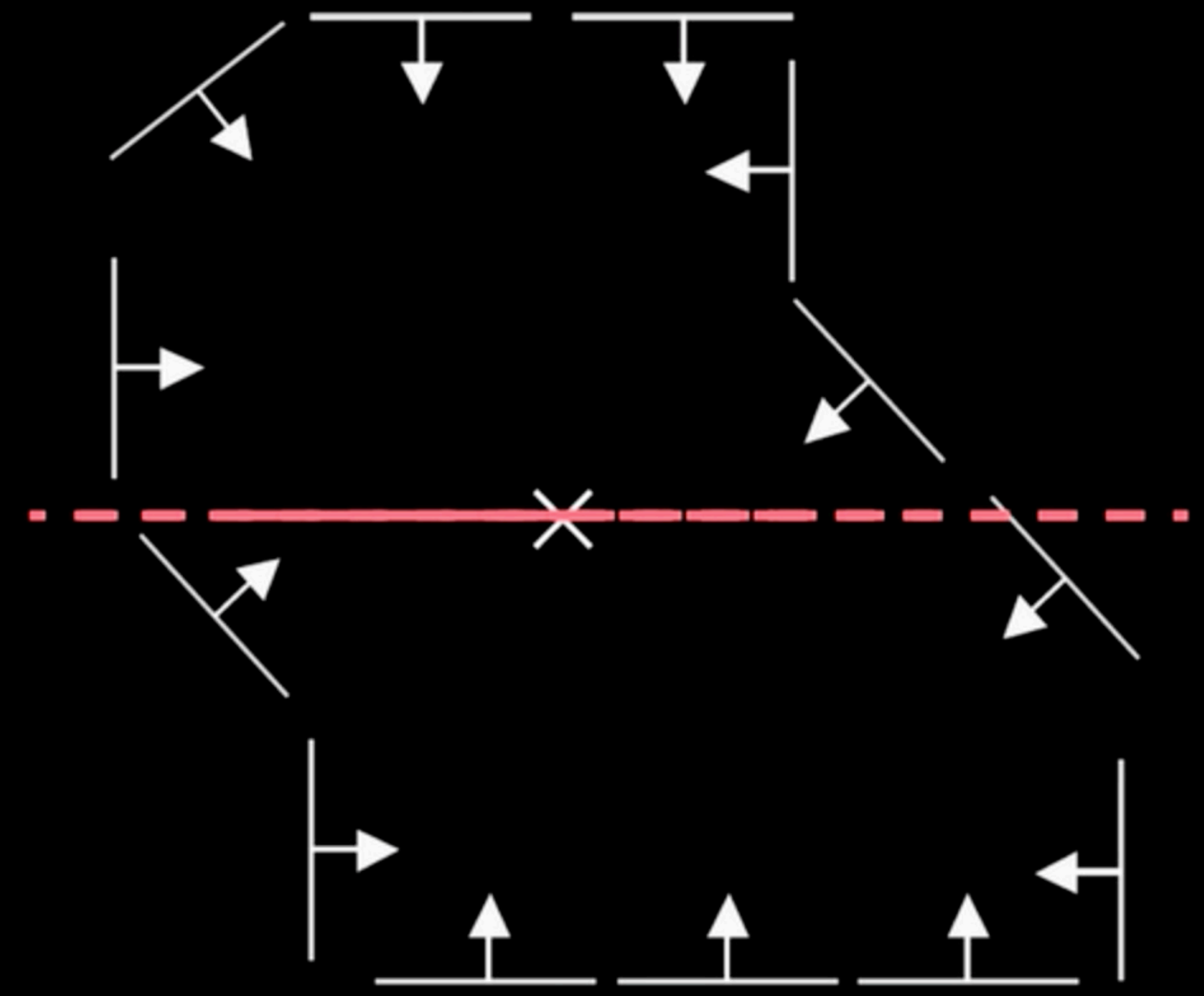
At recognition, each bottom horizontal element votes for all those displacements.



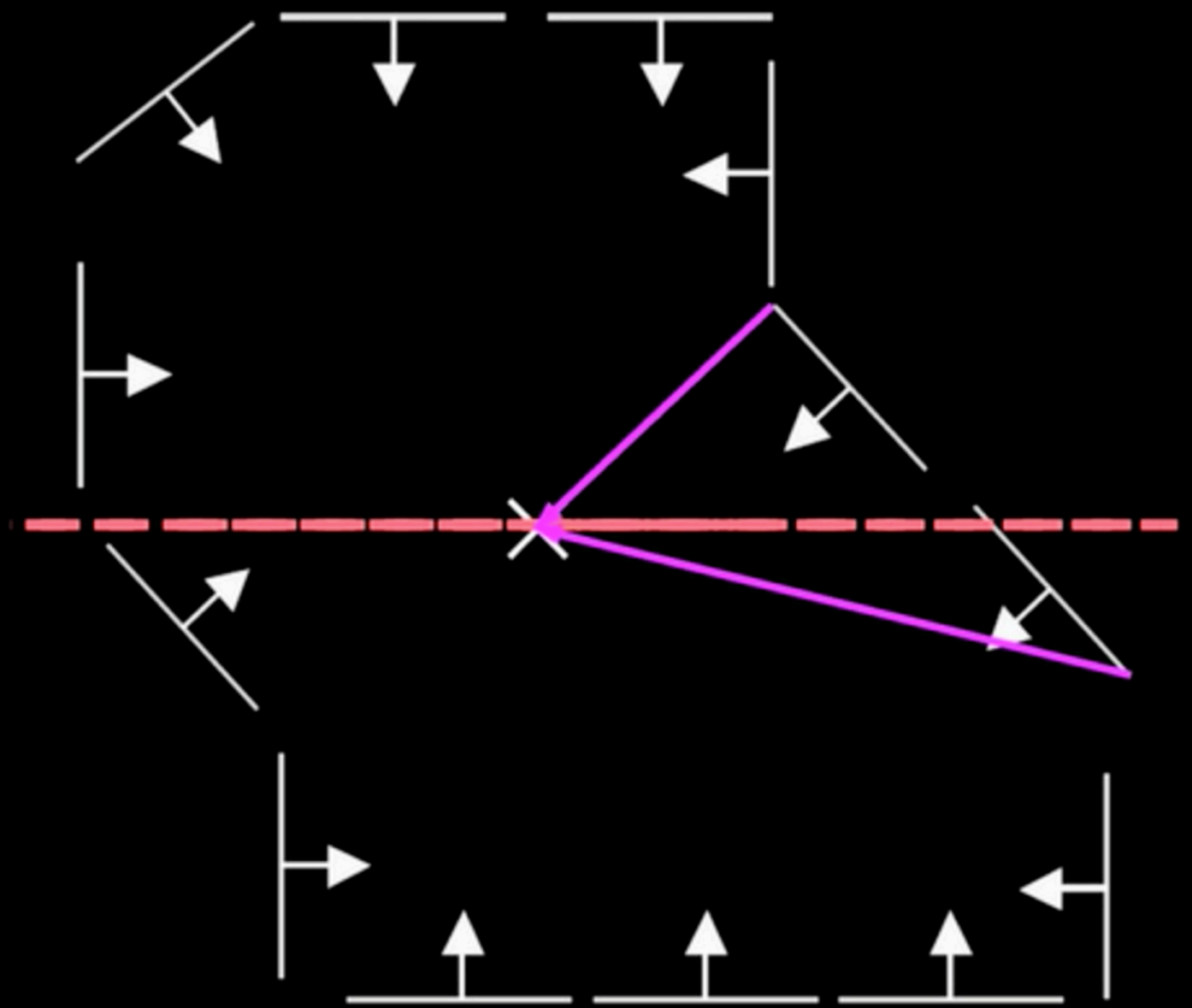
At recognition, each bottom horizontal element votes for all those displacements.



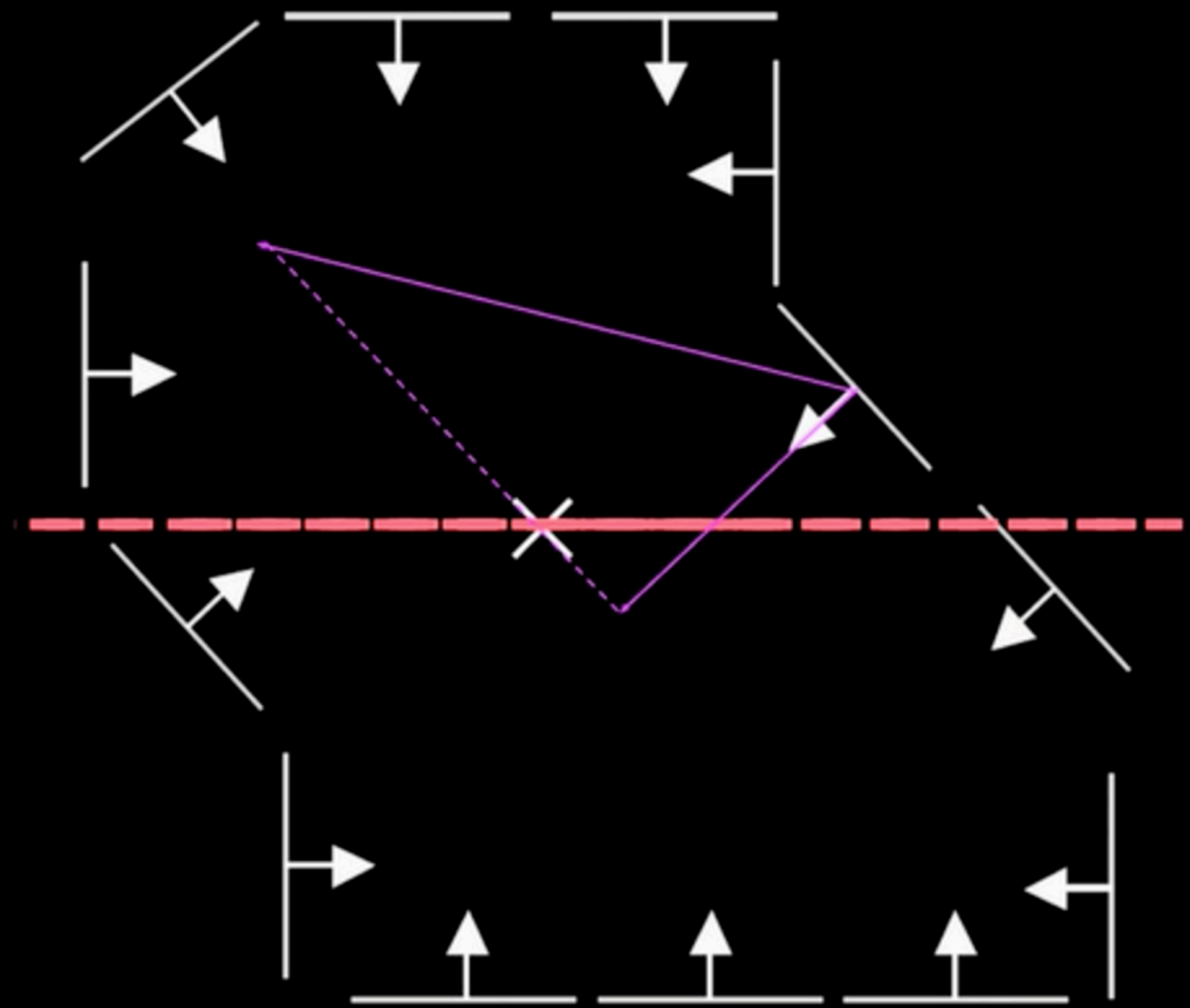
At recognition, each bottom horizontal element votes for all those displacements.



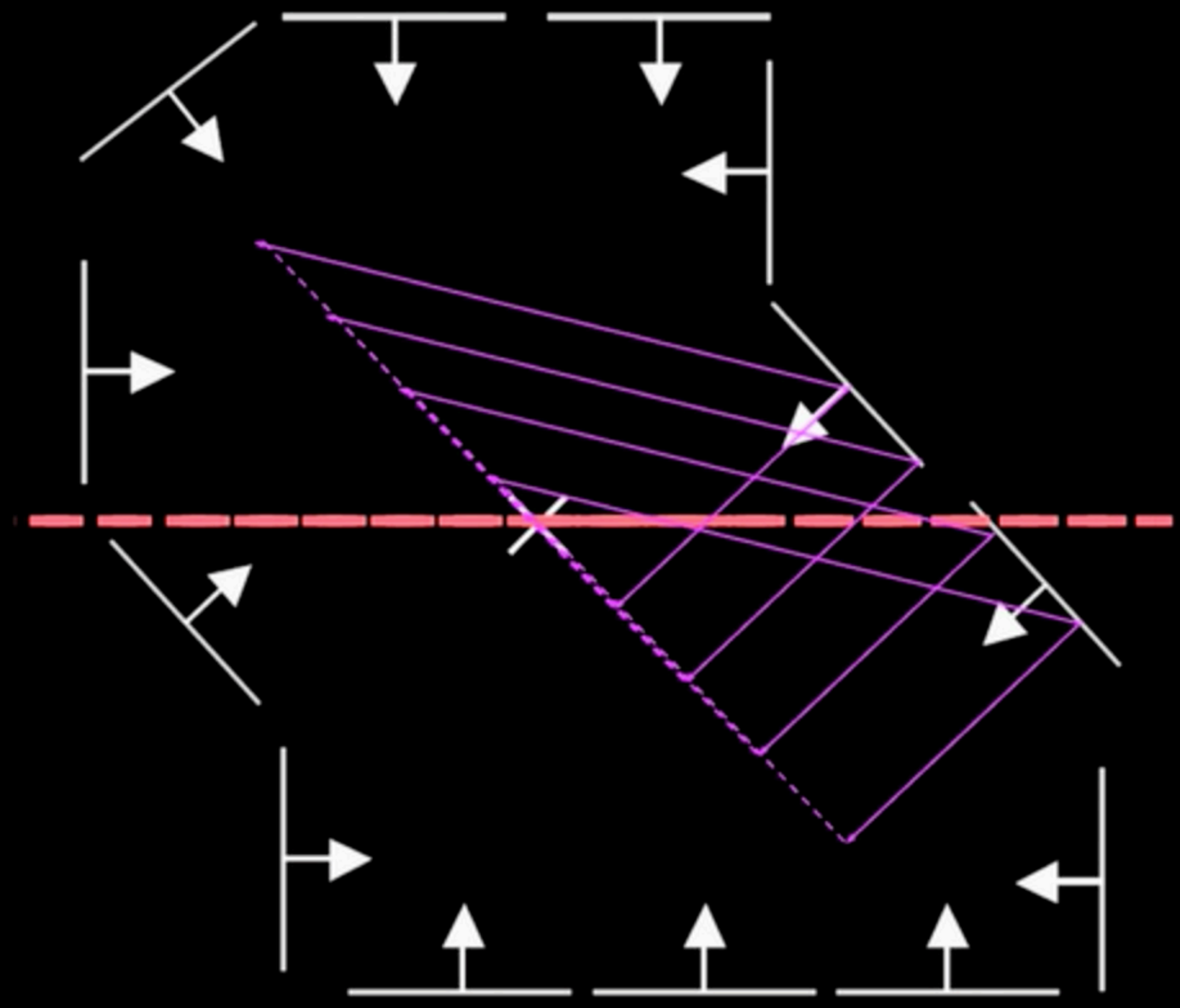
Now do for the leftward pointing diagonals.



Now do for the leftward pointing diagonals.

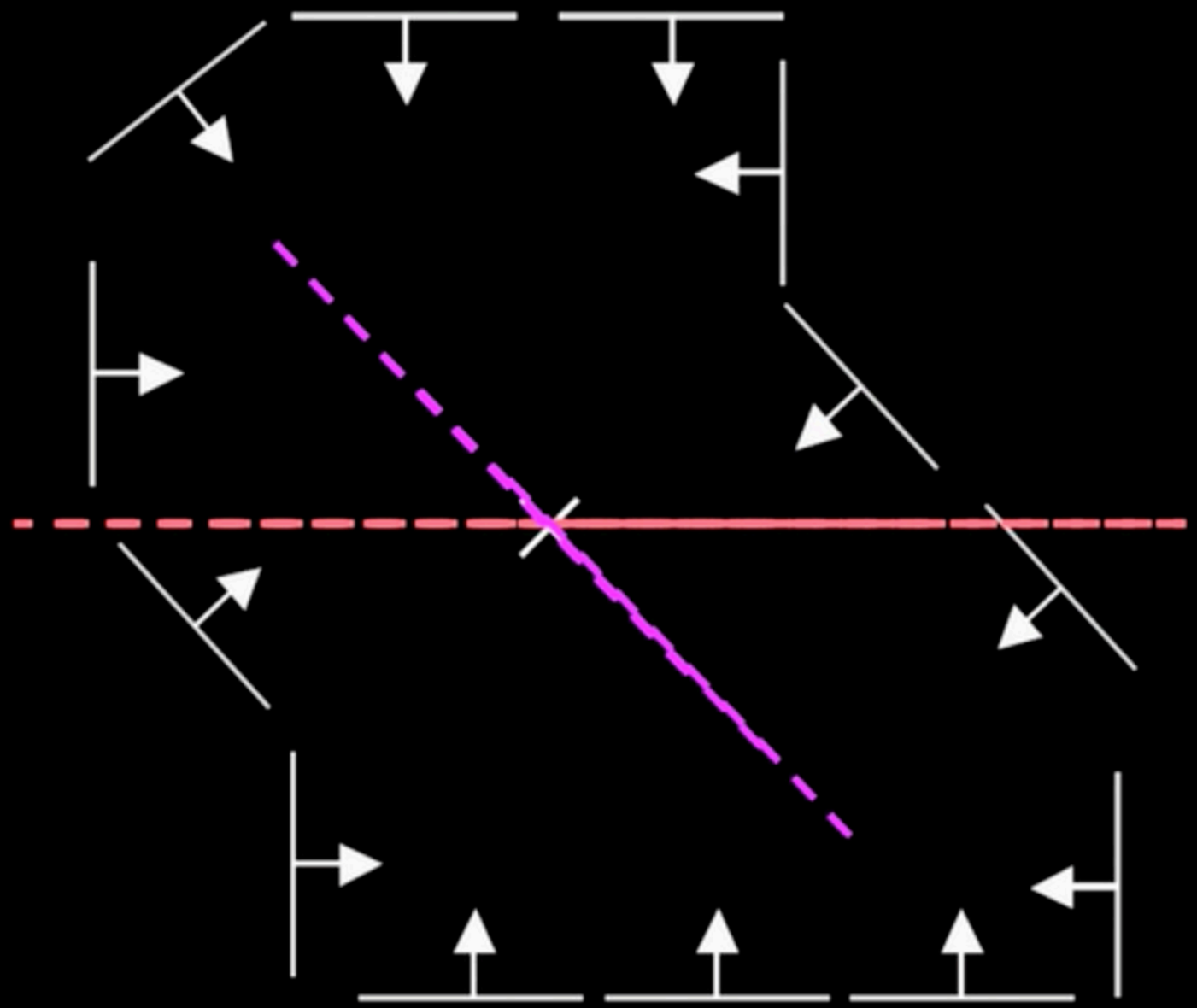


Now do for the leftward pointing diagonals.

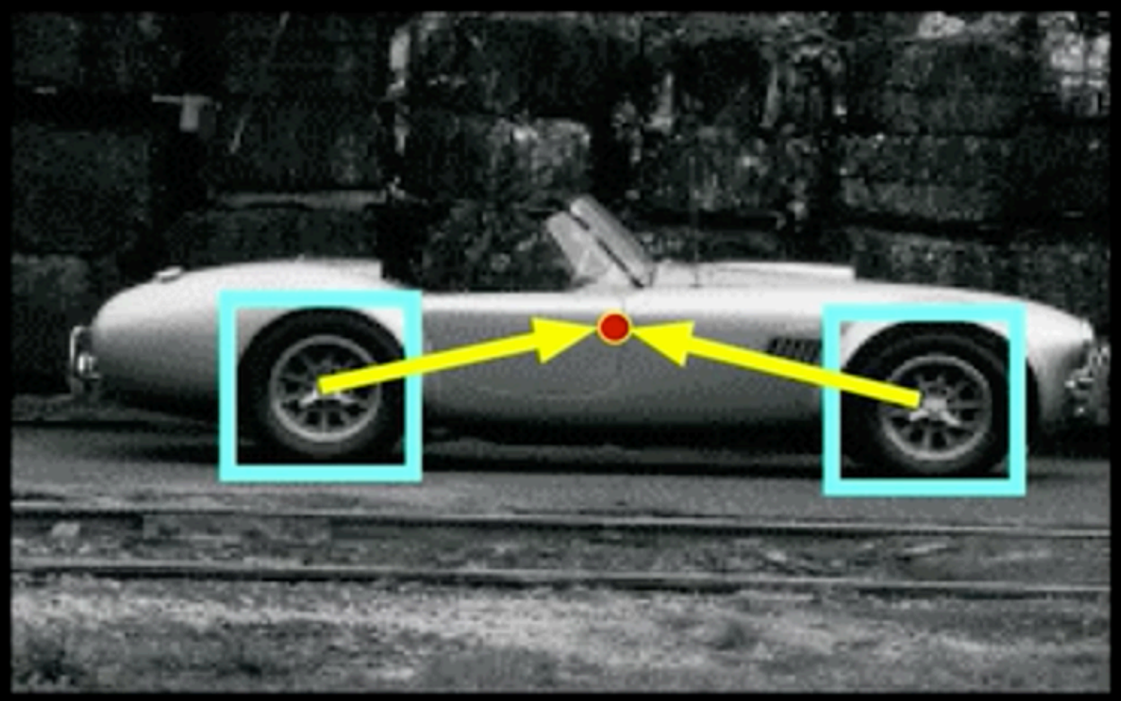


Now do for the leftward pointing diagonals.

And the center is found



- Instead of indexing displacements by gradient orientation, index by “visual codeword”

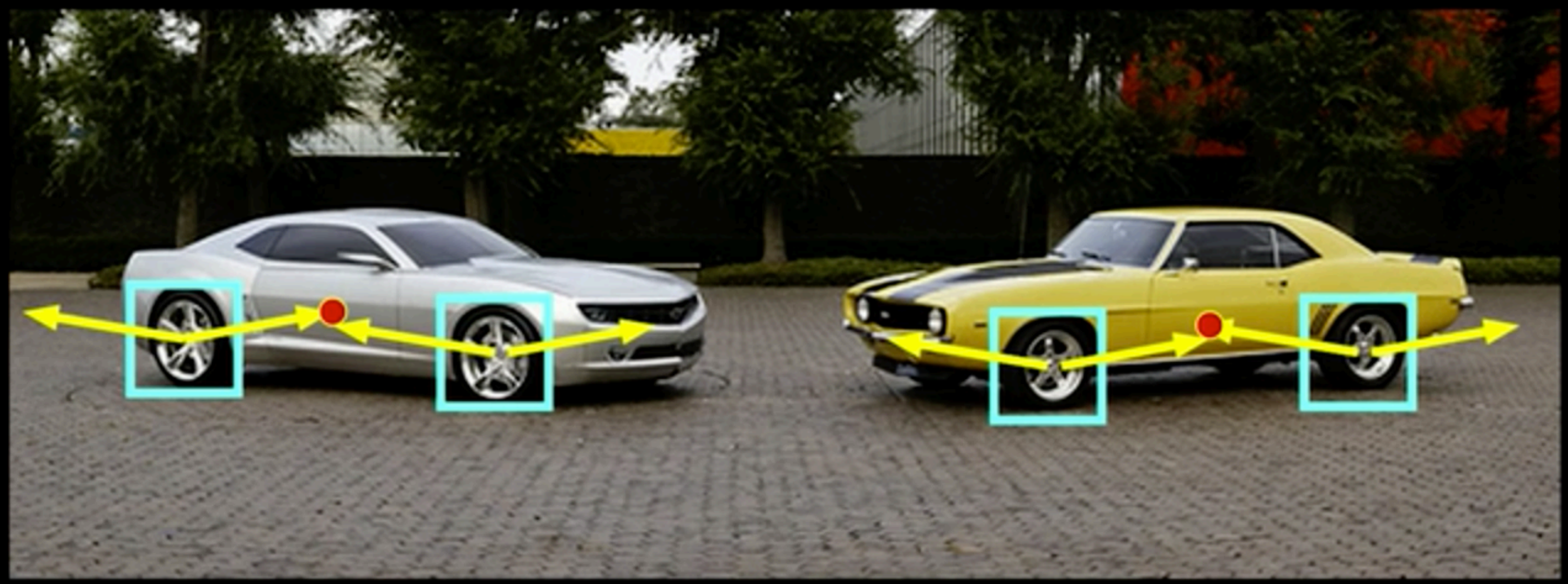


training image

visual codeword with displacement vectors

B. Leibe, A. Leonardis, and B. Schiele, Combined Object Categorization and Segmentation with an Implicit Shape Model, ECCV Workshop 2004

Application in recognition

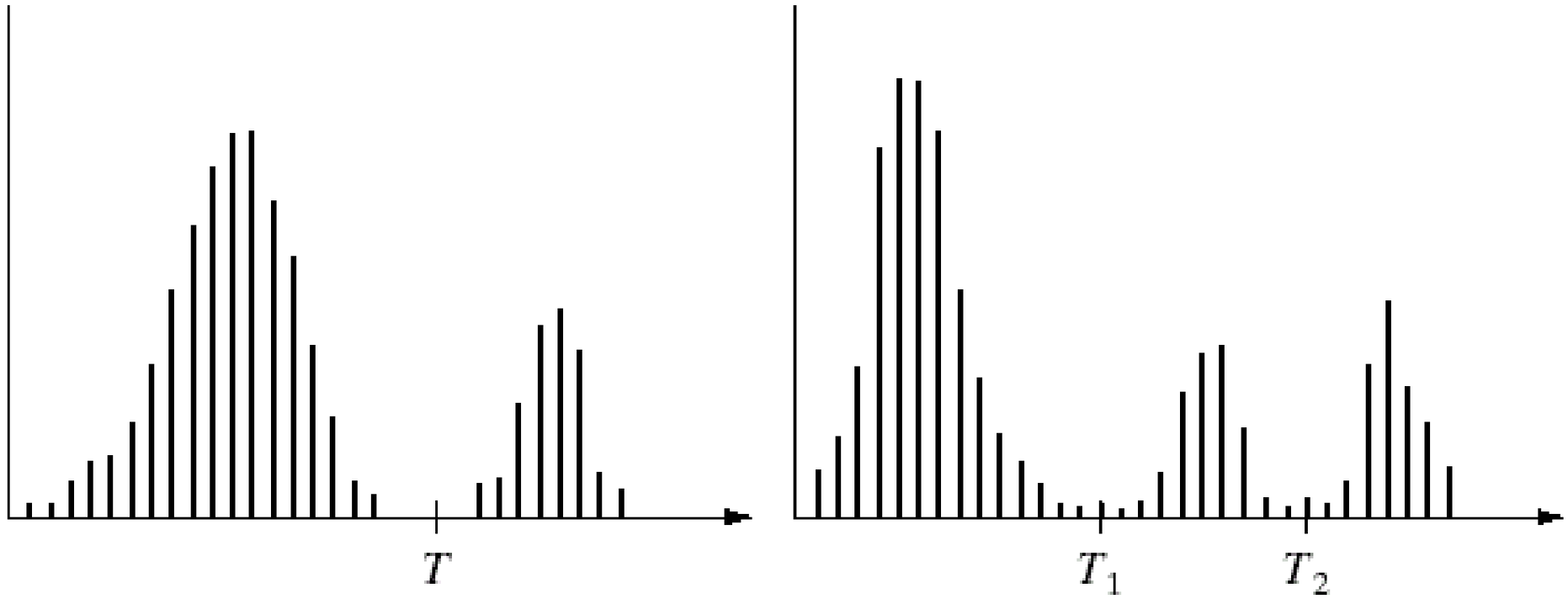


Region-based methods

Thresholding

Thresholding

- Label pixel as belonging to one of two (or more) classes



a b

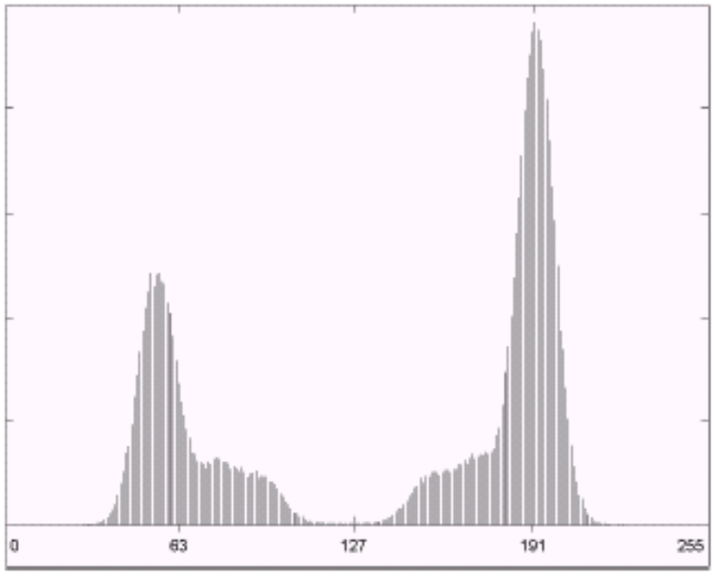
FIGURE 10.26 (a) Gray-level histograms that can be partitioned by (a) a single threshold, and (b) multiple thresholds.

Basic Global Thresholding

- A heuristic algorithm

1. Select an initial estimate for threshold T
2. Segment the image using T
   - G1 is all pixels with intensities > T
   - G2 is all pixels with intensities <= T
3. Compute averages m1 and m2 for the pixels in G1 and G2
4. Let T = (m1+m2)/2
5. Repeat steps 1-4 until no more change

- Note – you can use the histogram to compute the averages (very fast)



a b
c

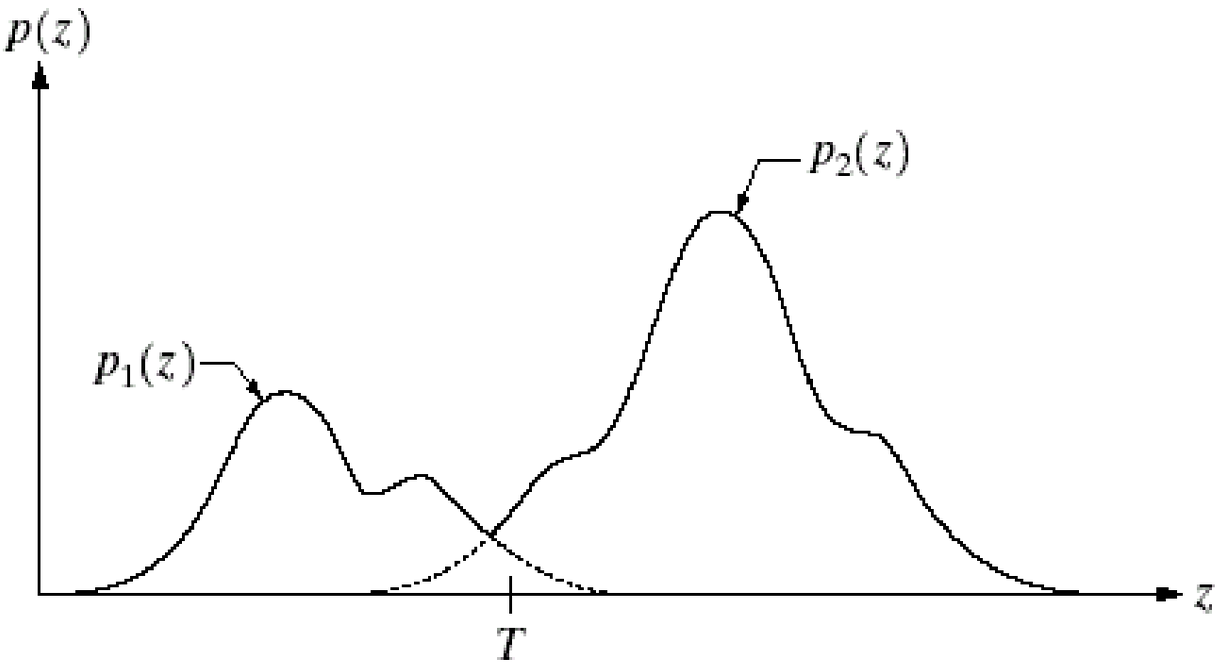
FIGURE 10.29
(a) Original image. (b) Image histogram. (c) Result of segmentation with the threshold estimated by iteration. (Original courtesy of the National Institute of Standards and Technology.)

Optimal Global Thresholding

- Want to pick a threshold that minimizes the error of miss-classifying a point

$$E_1(T) = \int_{-\infty}^{T} p_2(z)\,dz \quad and \quad E_2(T) = \int_{T}^{\infty} p_1(z)\,dz$$

FIGURE 10.32 Gray-level probability density functions of two regions in an image.



Unfortunately, we usually don't know the pdf's $p_1(z)$, $p_2(z)$ in advance

Otsu's Method for Global Thresholding

Used in Matlab's "graythresh" function

- Choose threshold to minimize the variance within groups

$$\sigma_W^2 = P_1\,\sigma_1^2 + P_2\,\sigma_2^2$$

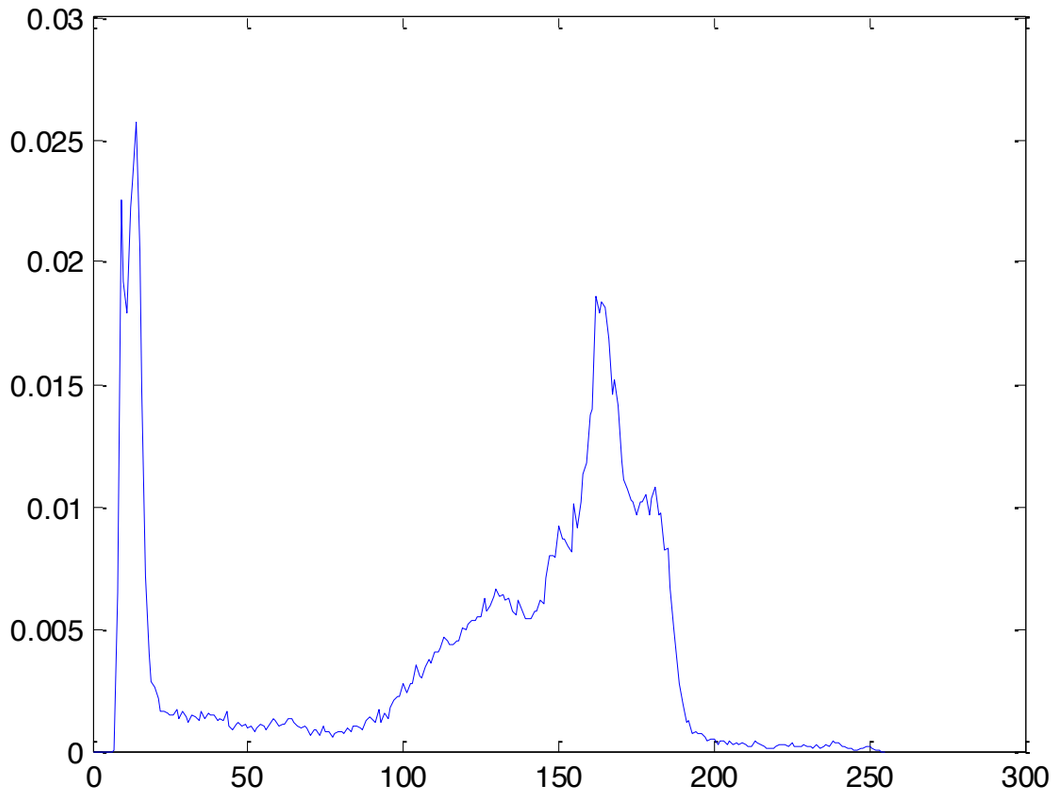
- Or equivalently, maximize the variance between groups

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

- where

$$P_1 = \sum_{i=0}^{k} p_i, \quad P_2 = \sum_{i=k+1}^{L-1} p_i$$

- m_G is the global mean; m_k is the mean of class k



Statistical decision theory
Bayes decision rule
Otsu: maximize the between class variance

Otsu's algorithm

1. Compute the normalized histogram: $p_i, i = 0,1,..,L-1$

2. Compute the cumulative sums: $$P_1(k) = \sum_{i=1}^{k} p_i$$

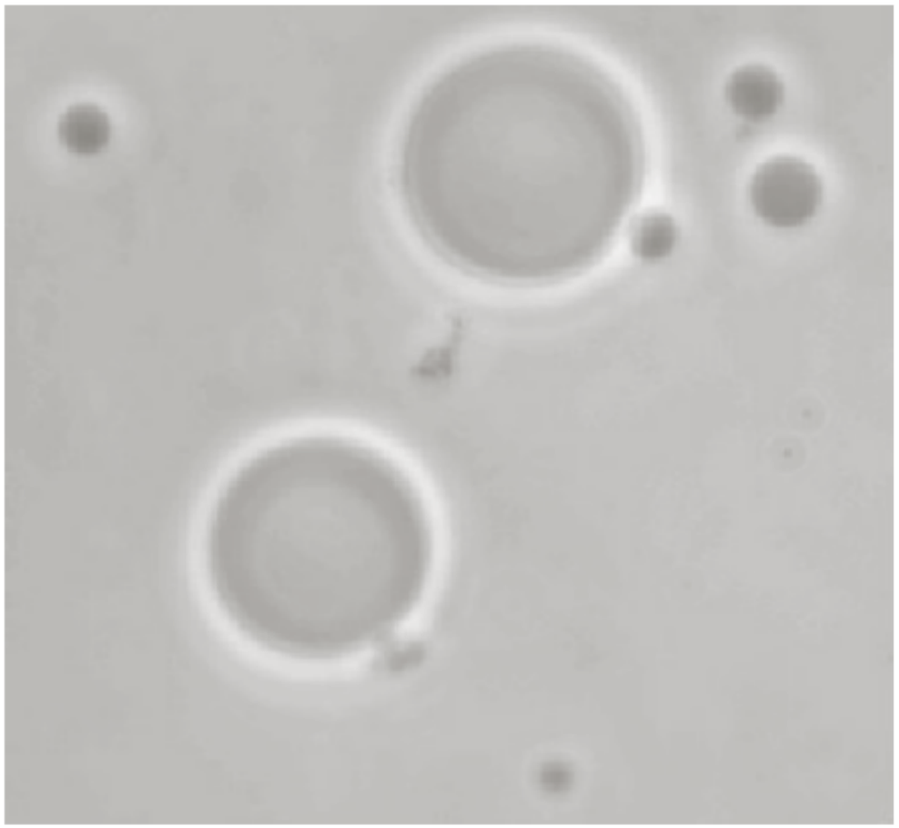
3. Compute the cumulative means: $$m(k) = \sum_{i=1}^{k} ip_i$$

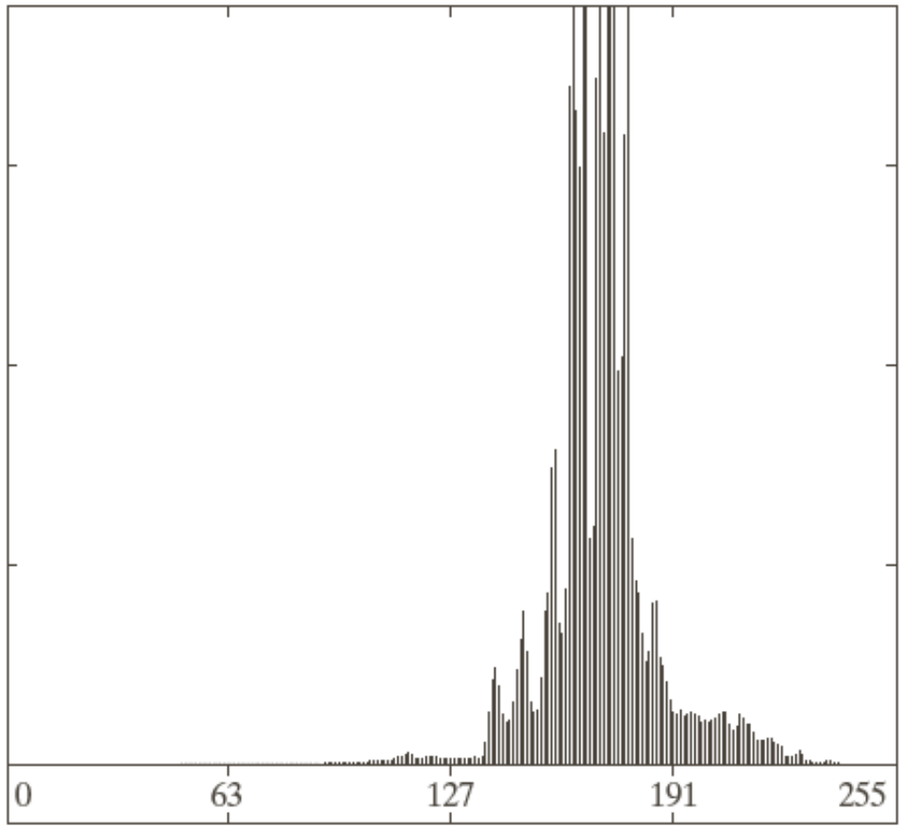
4. Compute the global mean: $$m_G = \sum_{i=1}^{L-1} ip_i$$

5. Compute the between-class variance: $$\sigma_B^2(k) = \frac{(m_G P_1(k) - m(k))^2}{P_1(k)(1 - P_1(k))}$$

6. Obtain the Otsu's threshold that maximizes: $$\sigma_B^2(k^*) = \max_k \sigma_B^2(k)$$

7. Obtain the the separability measure: $$\mu(k) = \frac{\sigma_B^2(k)}{\sigma_G^2}$$





a b
c d

FIGURE 10.39
(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 10.3.2.
(d) Result obtained using Otsu's method.
(Original image courtesy of Professor Daniel A. Hammer, the University of Pennsylvania.)

Summary / Questions

- Thresholding is a way to segment an image, such that each pixel is labeled as belonging to one of (typically) two classes.

- A global thresholding algorithm chooses a single threshold, based on the image histogram across the entire image.

- What property does the Otsu thresholding algorithm seek to minimize (or maximize)?

Split & Merge

“Split-and-merge” Segmentation

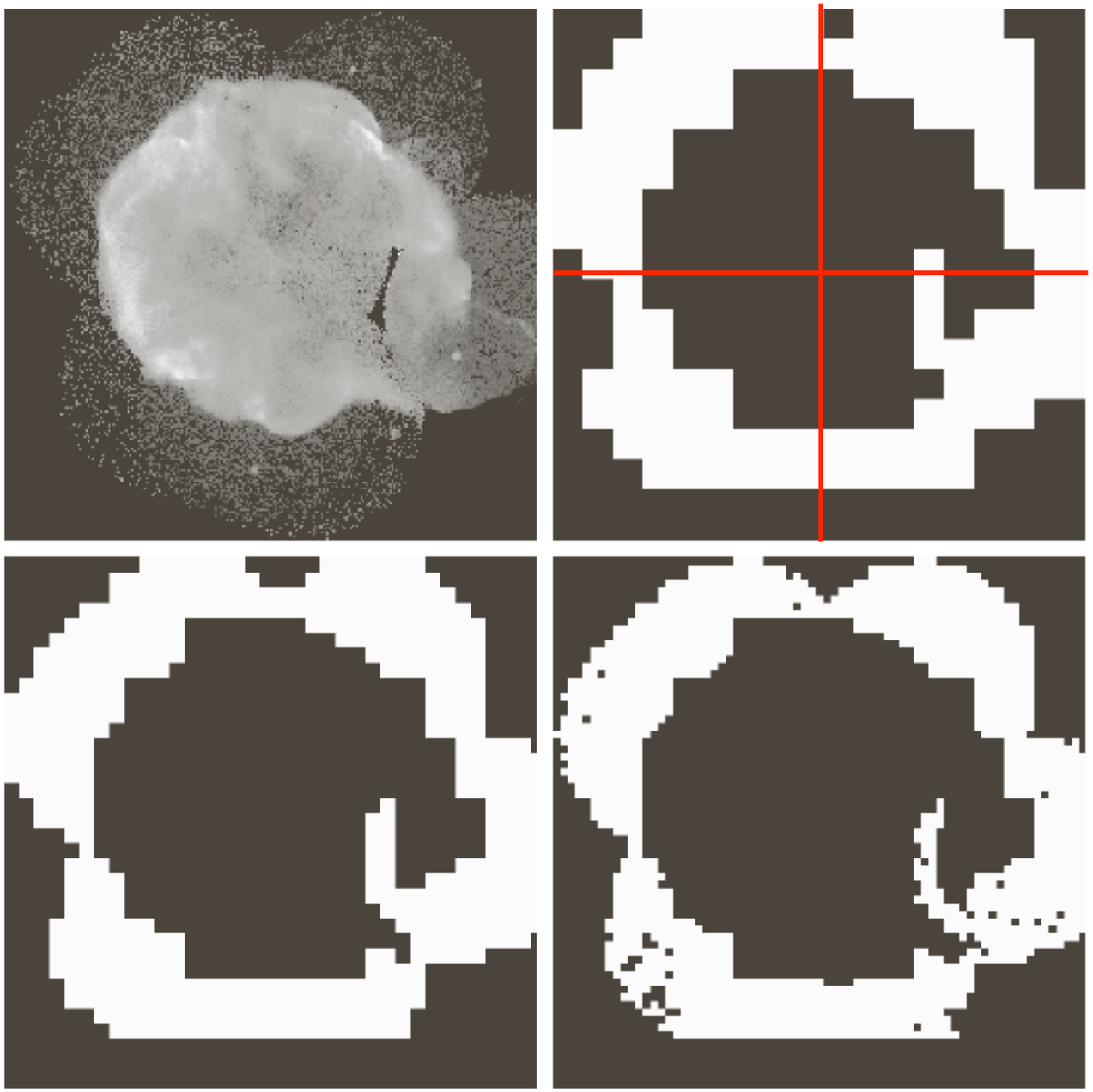
- Find a partitioning of connected regions such that each region has a consistent property (eg, color or texture) within it

- “Split and merge”:
  - Split regions that are inconsistent
  - Merge adjoining regions that are mutually consistent
  - Repeat until no further change is possible

Example

Consistency property:

Q = TRUE if σ > a and μ < b



a b
c d

FIGURE 10.53
(a) Image of the Cygnus Loop supernova, taken in the X-ray band by NASA's Hubble Telescope. (b)–(d) Results of limiting the smallest allowed quadregion to sizes of 32×32, 16×16, and 8×8 pixels, respectively. (Original image courtesy of NASA.)

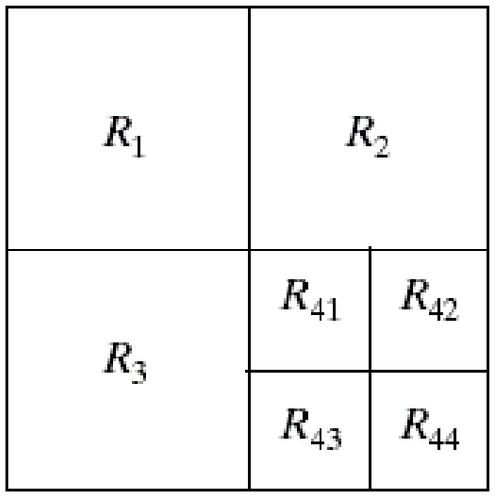
How can we remove holes?

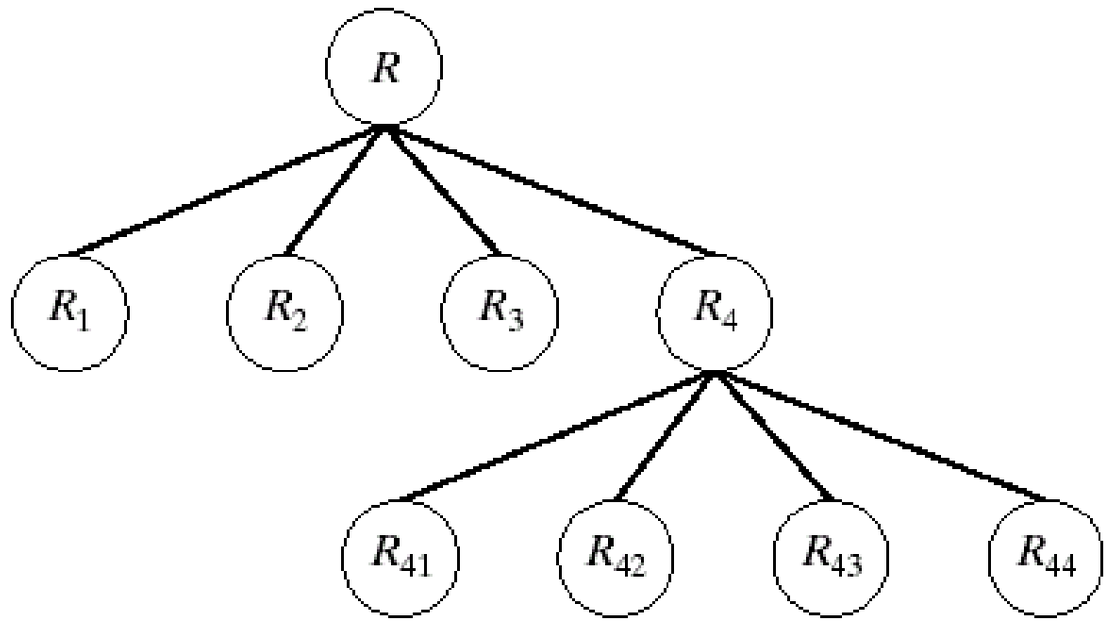
Quadtree Representation

- Recursively divide image into four quadrants
- Stop dividing when each quadrant is consistent
- A very concise and efficient representation

a b

FIGURE 10.42
(a) Partitioned image.
(b) Corresponding quadtree.





Split and Merge Algorithm

- Unlike RG, region splitting starts with the whole image as a single region and subdivides it into subsidiary regions recursively while a condition of homogeneity is not satisfied.
- Region merging is the opposite of splitting, and works as a way of avoiding over-segmentation

Split and Merge Algorithm

0	1	0	0	7	7	7	7
1	0	2	2	7	7	7	7
0	2	2	2	7	7	7	7
4	4	2	2	7	7	7	7
0	0	1	1	3	3	7	7
1	1	2	2	3	7	7	7
2	4	3	0	5	7	7	7
2	3	3	5	5	0	7	7

original image

0	1	0	0	7	7	7	7
1	0	2	2	7	7	7	7
0	2	2	2	7	7	7	7
4	4	2	2	7	7	7	7
0	0	1	1	3	3	7	7
1	1	2	2	3	7	7	7
2	4	3	0	5	7	7	7
2	3	3	5	5	0	7	7

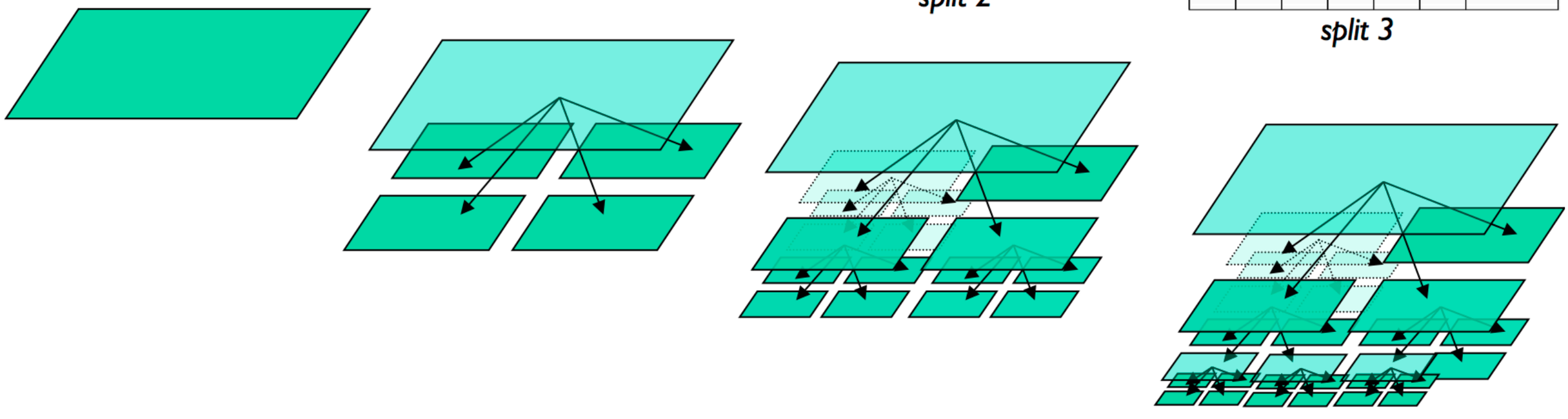
split 1

0	1	0	0	7	7	7	7
1	0	2	2	7	7	7	7
0	2	2	2	7	7	7	7
4	4	2	2	7	7	7	7
0	0	1	1	3	3	7	7
1	1	2	2	3	7	7	7
2	4	3	0	5	7	7	7
2	3	3	5	5	0	7	7

split 2

0	1	0	0	7	7	7	7
1	0	2	2	7	7	7	7
0	2	2	2	7	7	7	7
4	4	2	2	7	7	7	7
0	0	1	1	3	3	7	7
1	1	2	2	3	7	7	7
2	4	3	0	5	7	7	7
2	3	3	5	5	0	7	7

split 3



http://

Split and Merge Algorithm

- When a homogeneous region is created, its neighboring regions are checked and the newly created region is merged with an existing one if they have identical properties.
- If the similarity criteria are met by more than one adjacent region, the new region is merged with the most similar one.

Split and Merge Algorithm

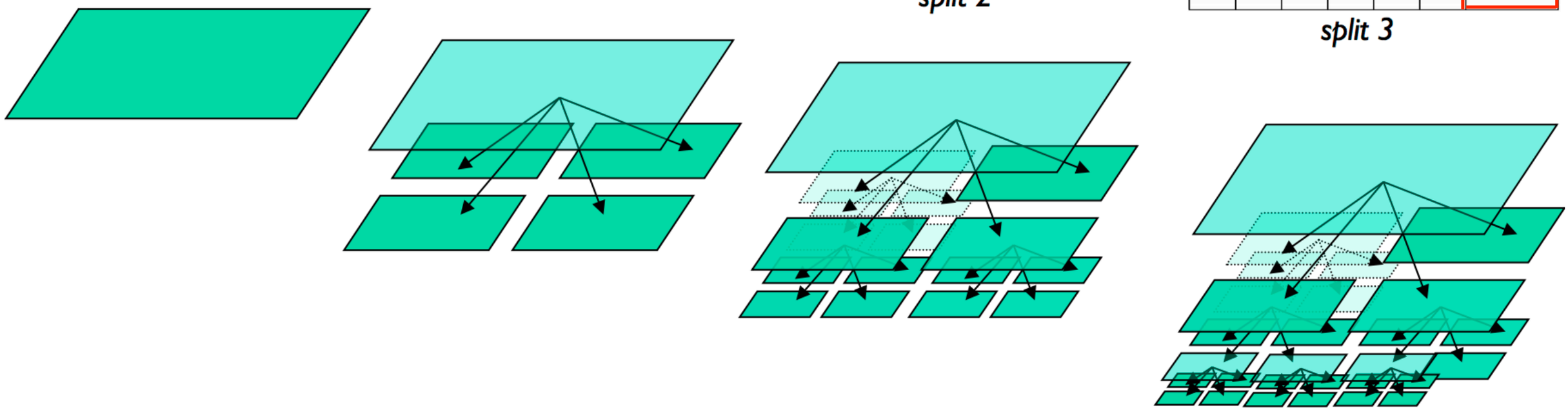
0	1	0	0	7	7	7	7
1	0	2	2	7	7	7	7
0	2	2	2	7	7	7	7
4	4	2	2	7	7	7	7
0	0	1	1	3	3	7	7
1	1	2	2	3	7	7	7
2	4	3	0	5	7	7	7
2	3	3	5	5	0	7	7

original image

split 1

split 2

split 3

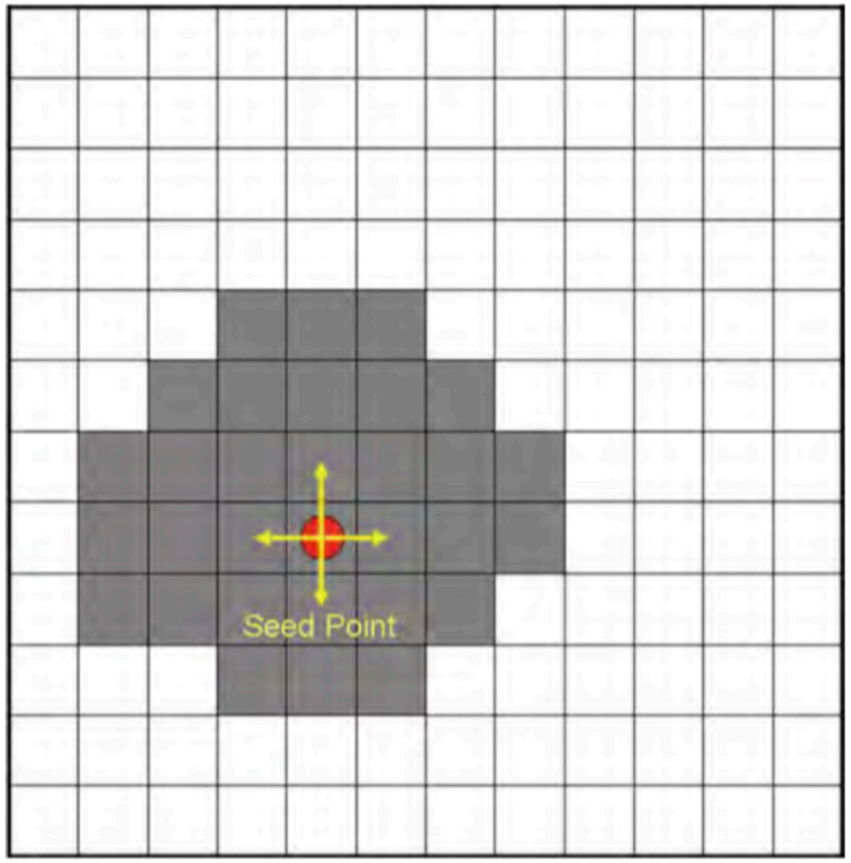


http://

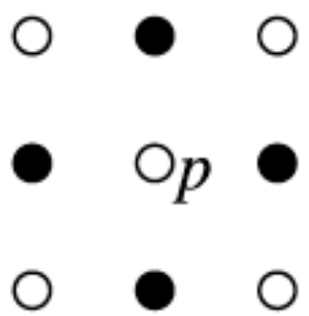
Region Growing

Region Growing/Merging

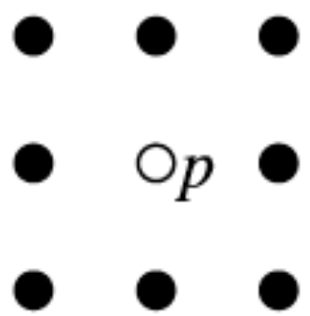
- Perhaps, one of the simplest approaches among region based methods.



1. Select a seed point/points
2. Define a growth criteria **(e.g. similar intensities)**
3. Joint all voxels connected to the seed that follow the growth criteria
4. Stop when no adjacent voxel agree with the growth criteria



4-connectivity



8-connectivity

Algorithm

1. Choose the seed pixel
2. Check the neighboring pixels and add them to the region if they are similar to the seed
3. Repeat step 2 for each of the newly added pixels; stop if no more pixels can be added

$$|\text{neighboring pixels} - \text{seed}| < Threshold$$

1. We select a seed point p_{seed} in a grey-level or colour image having brightness $B(p_{seed})$. At the beginning, the current segment only contains p_{seed}.
2. In an iteration process, we merge any pixel q to the current segment that is 8-adjacent to a pixel in the current segment, not yet labelled, and has an intensity $B(q)$ satisfying $|B(q) - B(p_{seed})| \leq \tau$.
3. We stop if there is no further not yet labelled pixel q, 8-adjacent to the current segment, which still can be merged.

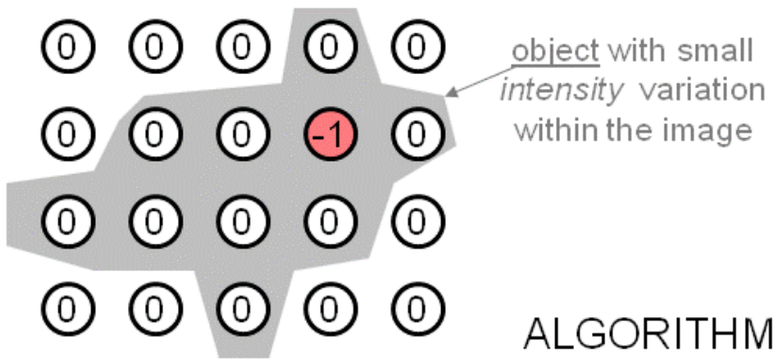
$$\tau > 0$$

Implementation

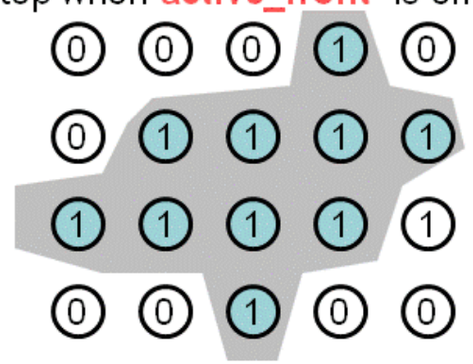
region(all pixels) = 0
region(seed) = -1

red nodes are the "active_front" (queue or stack) **= ()**

add seed into active_front **= (seed)**

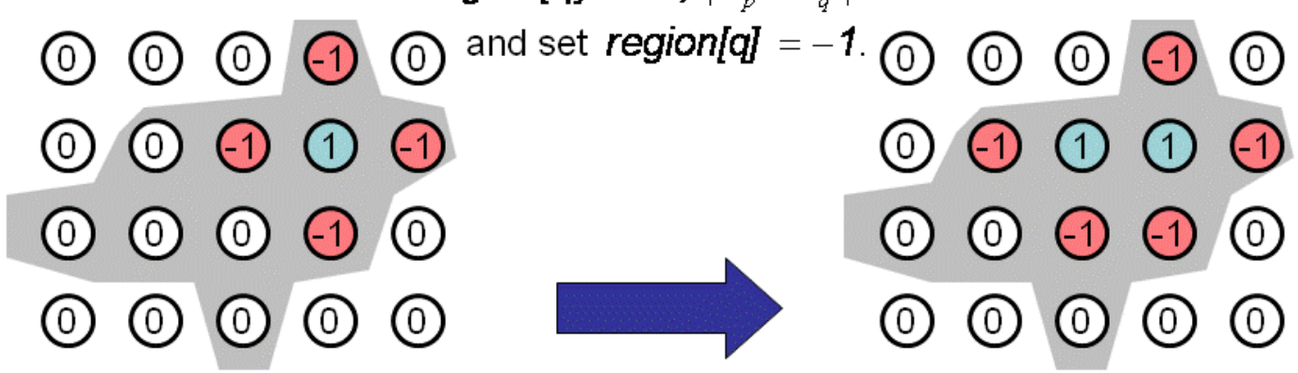


stop when active_front is empty



ALGORITHM:

Remove pixel *p* from active_front
and mark it as $region[p] = 1$.
Add all neighbors *q* such that:
$region[q] == 0,\ |I_p - I_q| < T$
and set $region[q] = -1$.

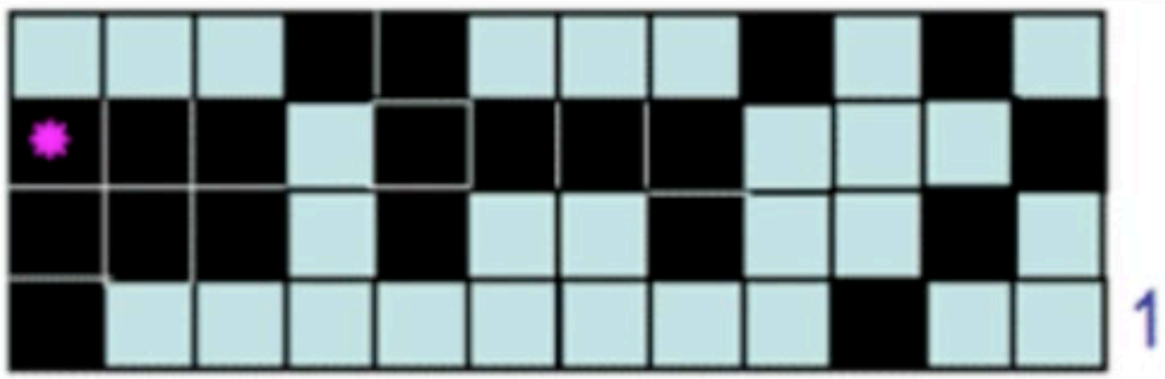


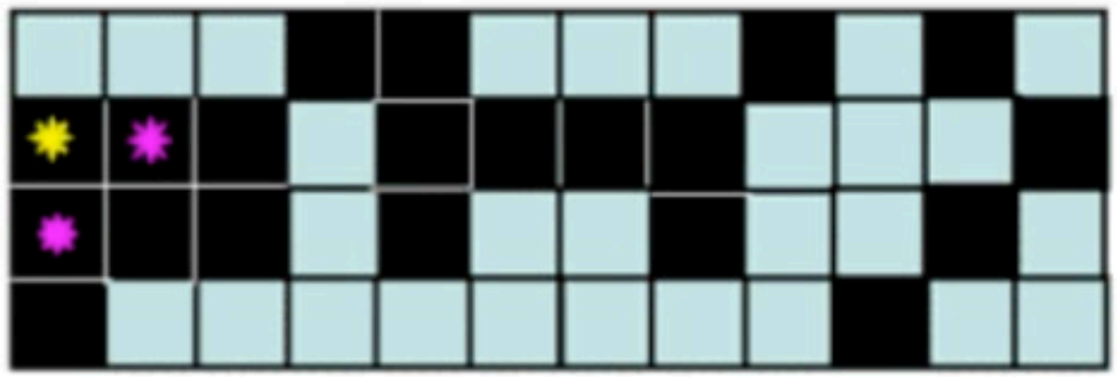
active_front = stack(q_left, q_bottom, q_right, q_top(start)

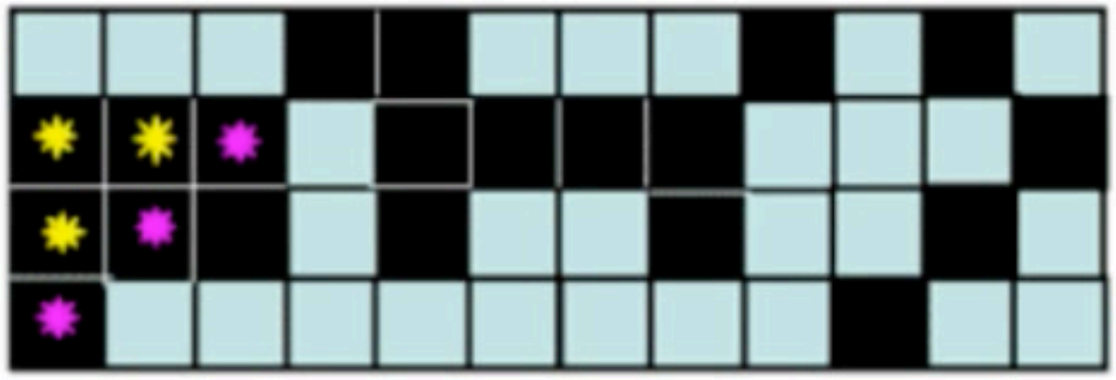
active_front = stack(q_bottom, q_right, q_top(start)

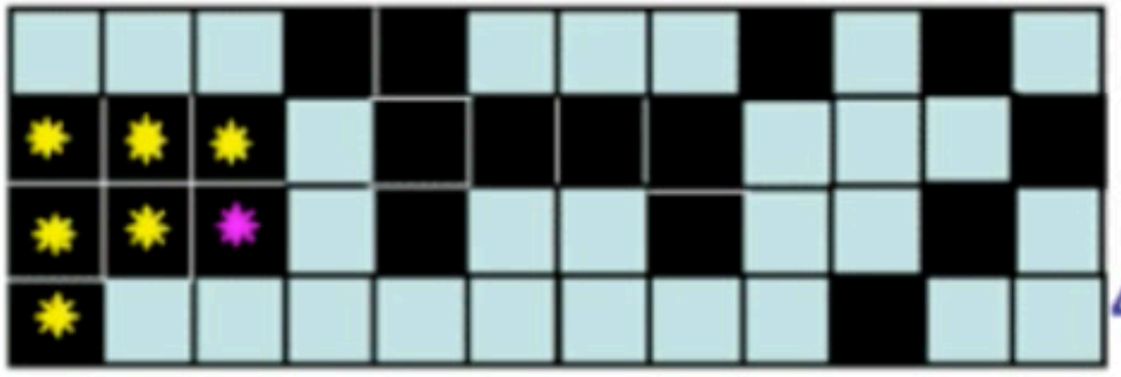
active_front = stack(q2_left, q2_bottom, q_bottom, q_right, q_top(start)

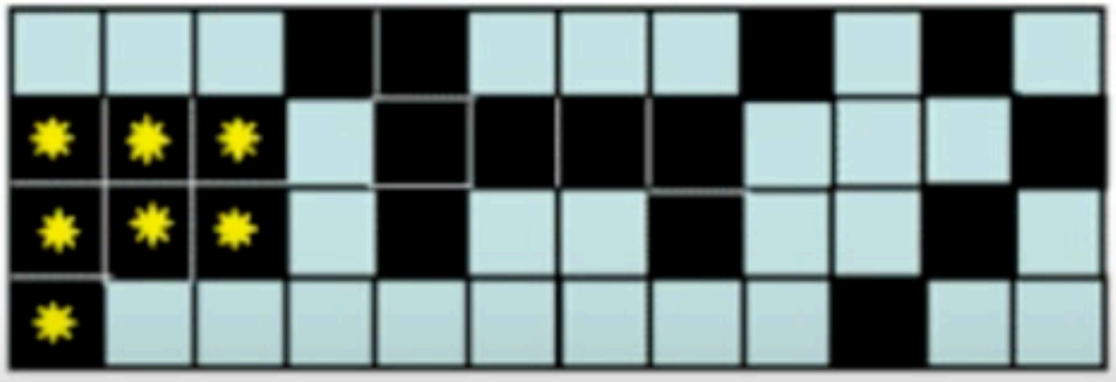
Example

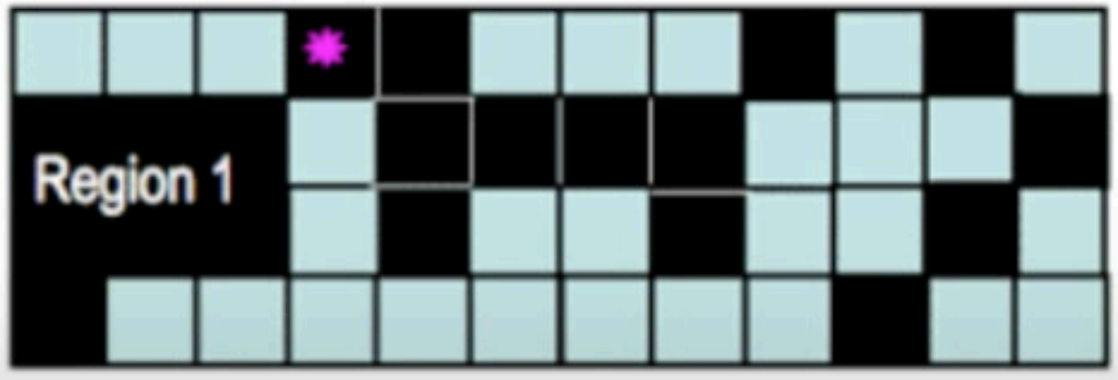

1


2


3


4


5



6