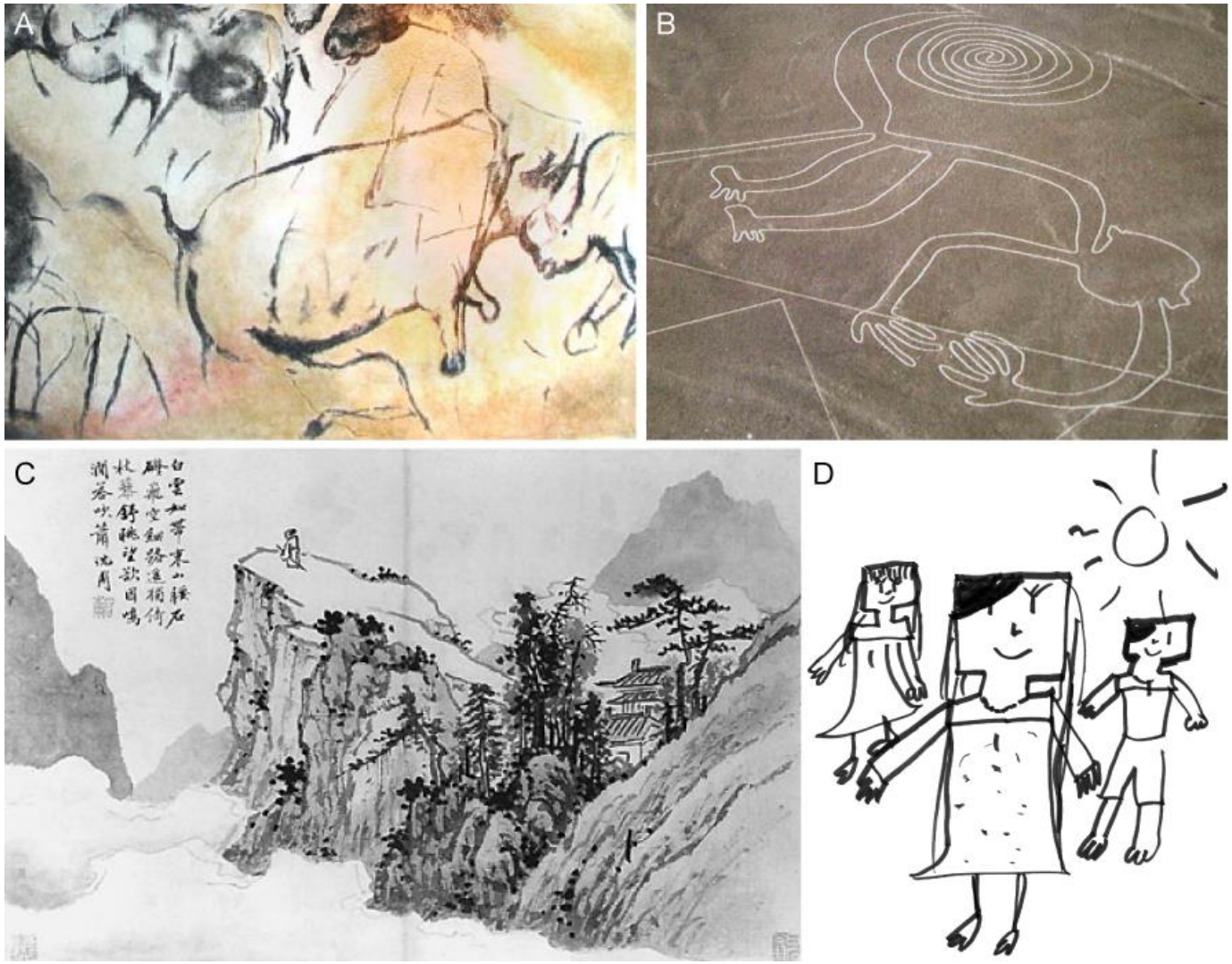
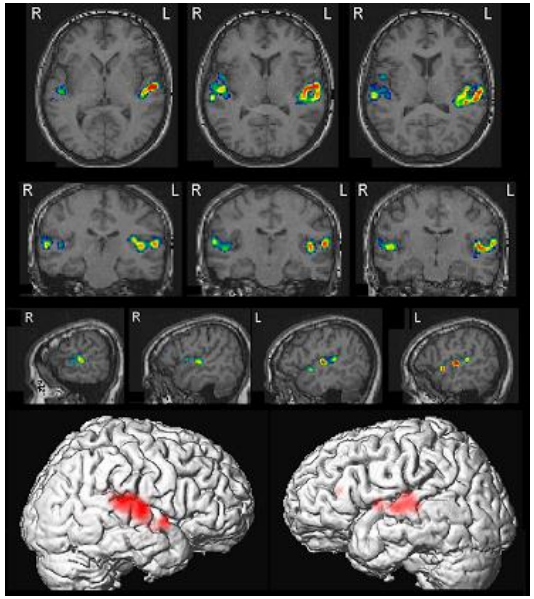
Edge Detection

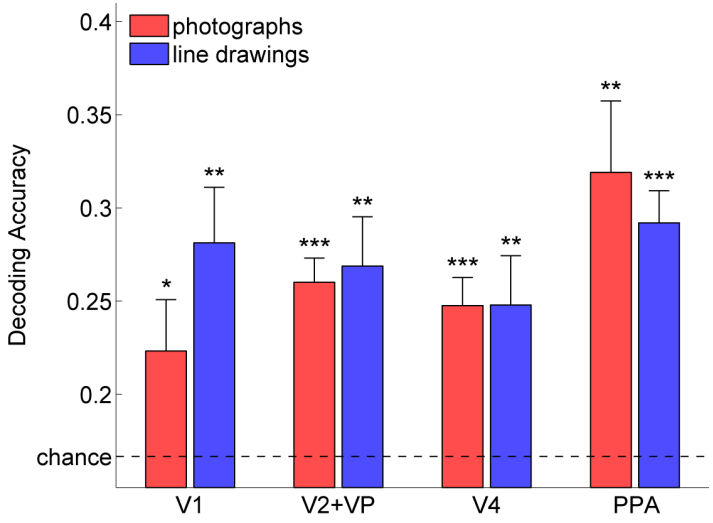
What we will learn today

- Edge detection
- Image Gradients
- A simple edge detector
- Sobel edge detector
- Canny edge detector
- Hough Transform



(A) Cave painting at Chauvet, France, about 30,000 B.C.;
(B) Aerial photograph of the picture of a monkey as part of the Nazca Lines geoglyphs, Peru, about 700 – 200 B.C.;
(C) Shen Zhou (1427-1509 A.D.): Poet on a mountain top, ink on paper, China;
(D) Line drawing by 7-year old I. Lleras (2010 A.D.).





Walther, Chai, Caddigan, Beck & Fei-Fei, *PNAS, 2011*

Edge detection

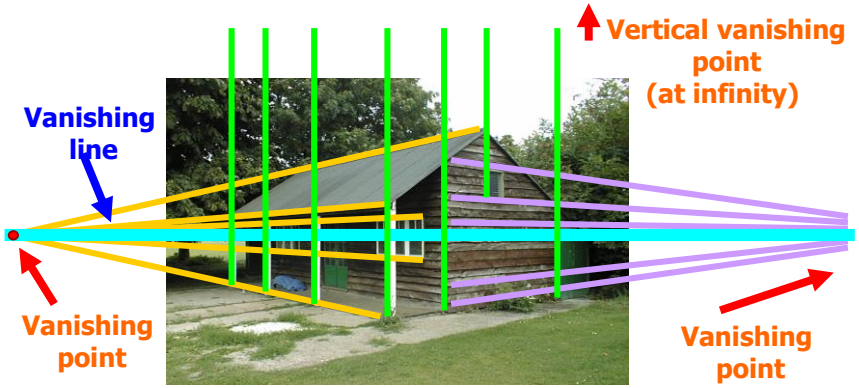
- **Goal:** Identify sudden changes (discontinuities) in an image
  - Intuitively, most semantic and shape information from the image can be encoded in the edges
  - More compact than pixels

Source: D. Lowe

Why do we care about edges?

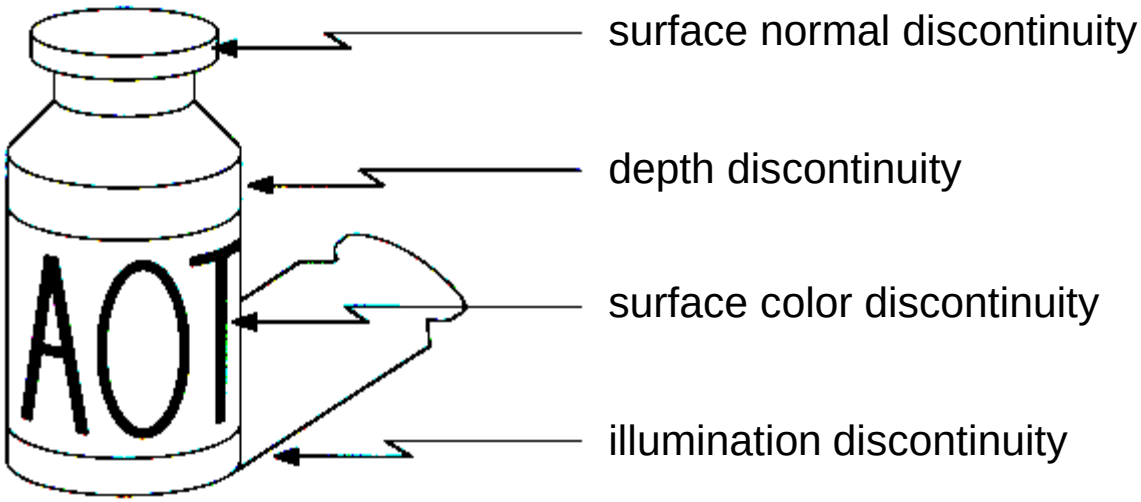
- Extract information, recognize objects

- Recover geometry and viewpoint



Source: J. Hayes

Origin of Edges

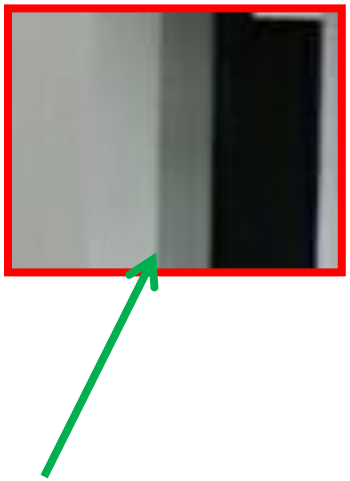


- Edges are caused by a variety of factors

Source: Steve Seitz

Closeup of edges

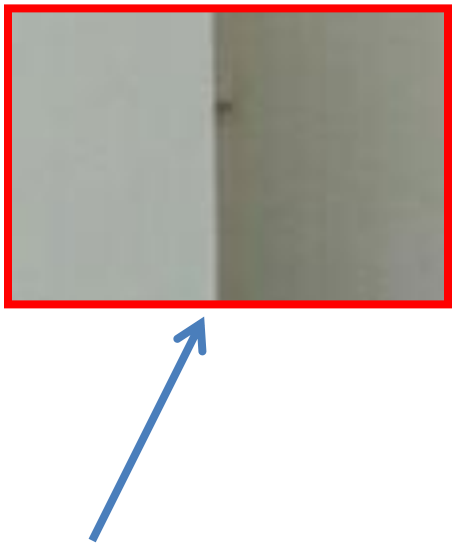
Surface normal discontinuity



Source: D. Hoiem

Closeup of edges

Depth discontinuity



Source: D. Hoiem

Closeup of edges

Surface color discontinuity

Source: D. Hoiem

What we will learn today

- Edge detection
- Image Gradients
- A simple edge detector
- Sobel edge detector
- Canny edge detector
- Hough Transform

Derivatives in 1D

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

Types of Discrete derivative in 1D

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

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

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

1D discrete derivative filters

- Backward filter: [0 1 -1]

$$f(x) - f(x-1) = f'(x)$$

1D discrete derivate filters

- Backward filter: [0 1 -1]

$$f(x) - f(x-1) = f'(x)$$

- Forward: [-1 1 0]

$$f(x) - f(x+1) = f'(x)$$

1D discrete derivate filters

- Backward filter: [0 1 -1]

$$f(x) - f(x-1) = f'(x)$$

- Forward: [-1 1 0]

$$f(x) - f(x+1) = f'(x)$$

- Central: [1 0 -1]

$$f(x+1) - f(x-1) = f'(x)$$

Discrete derivate in 2D

Given function $f(x, y)$

Discrete derivate in 2D

Given function $f(x, y)$

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

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

2D discrete derivative filters

What does this filter do?

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

2D discrete derivative filters

What about this filter?

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

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

2D discrete derivative - example

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

2D discrete derivative - example

What happens when we apply this filter?

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

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

2D discrete derivative - example

What happens when we apply this filter?

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

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

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

2D discrete derivative - example

Now let's try the other filter!

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

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

2D discrete derivative - example

What happens when we apply this filter?

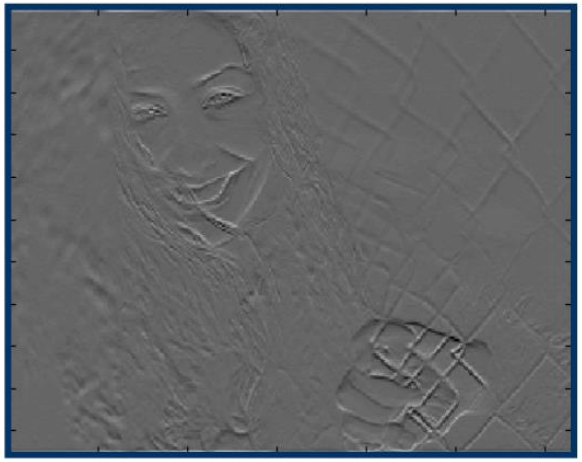
$$I = \begin{bmatrix} 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \\ 10 & 10 & 20 & 20 & 20 \end{bmatrix}$$

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

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

3x3 image gradient filters

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



What we will learn today

- Edge detection
- Image Gradients
- A simple edge detector
- Sobel edge detector
- Canny edge detector
- Hough Transform

Characterizing edges

- An edge is a place of rapid change in the image intensity function

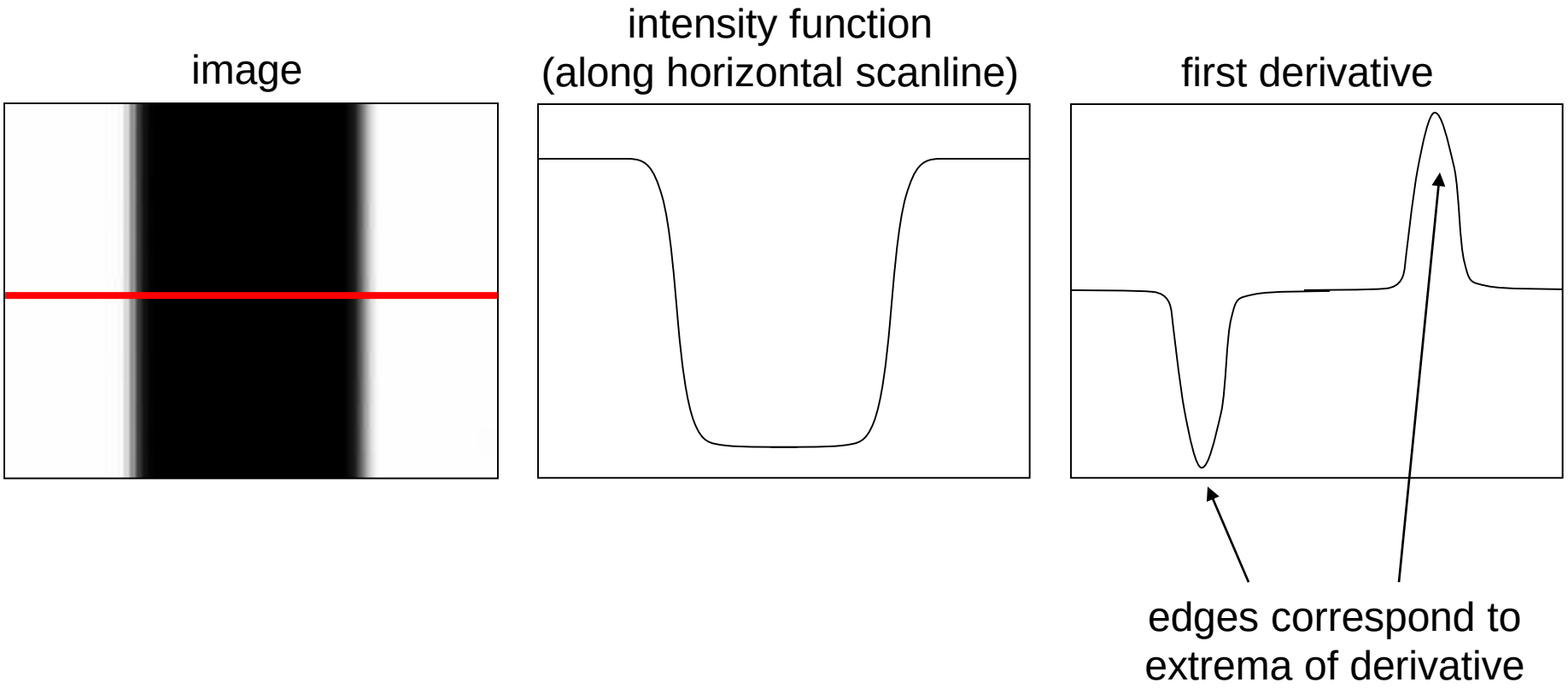
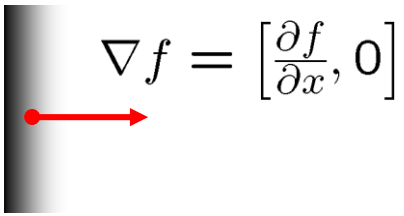
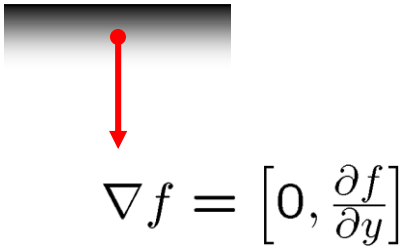
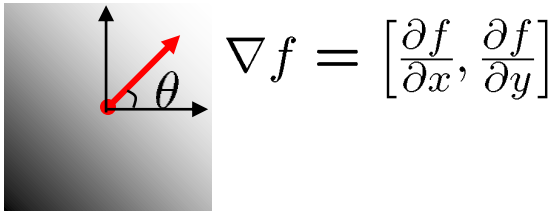


Image gradient

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







The gradient vector 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

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

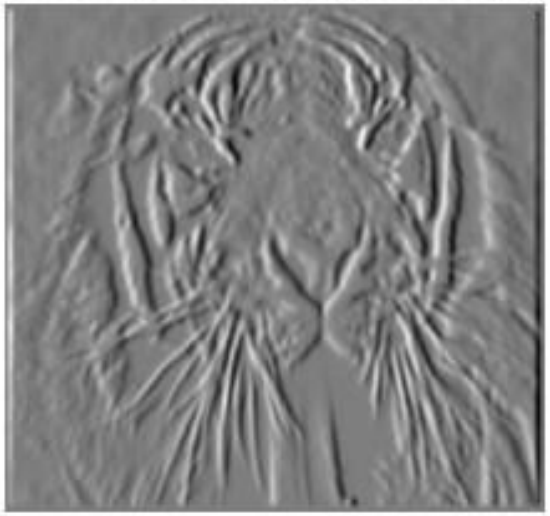
Source: Steve Seitz

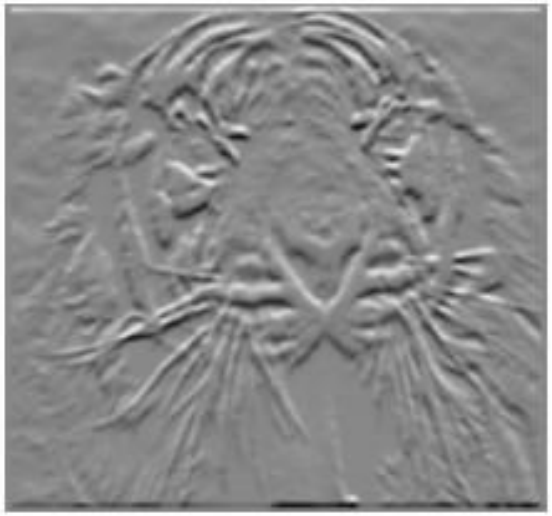
Finite differences: example

Original Image

Gradient magnitude

x-direction

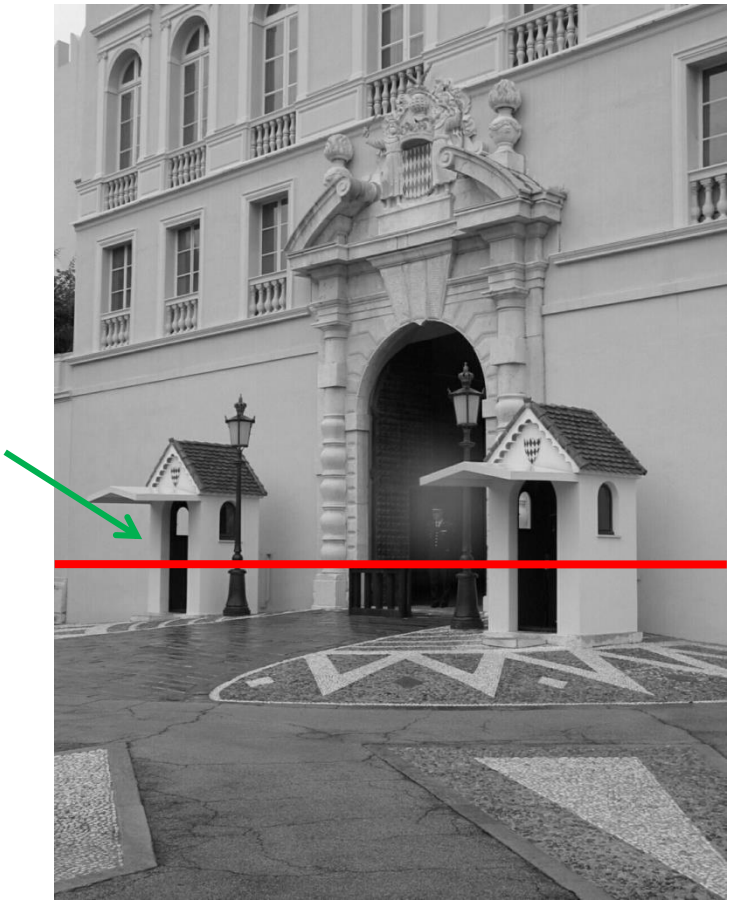


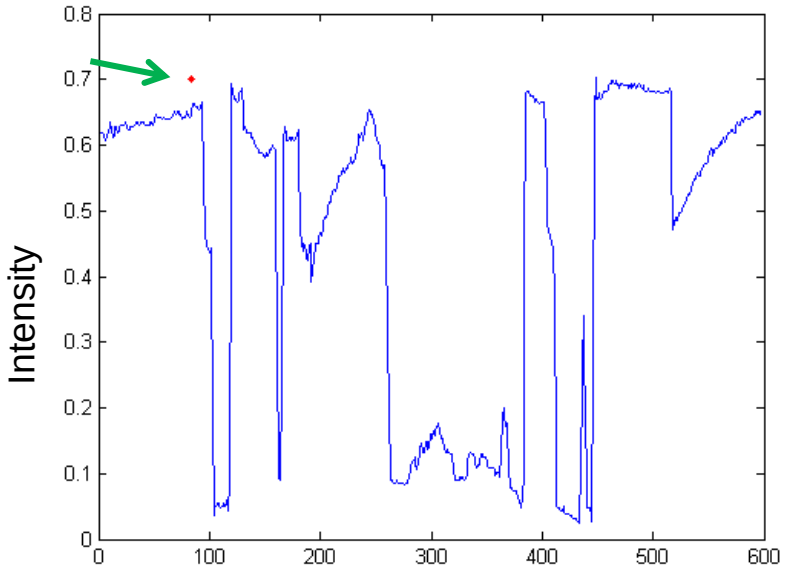


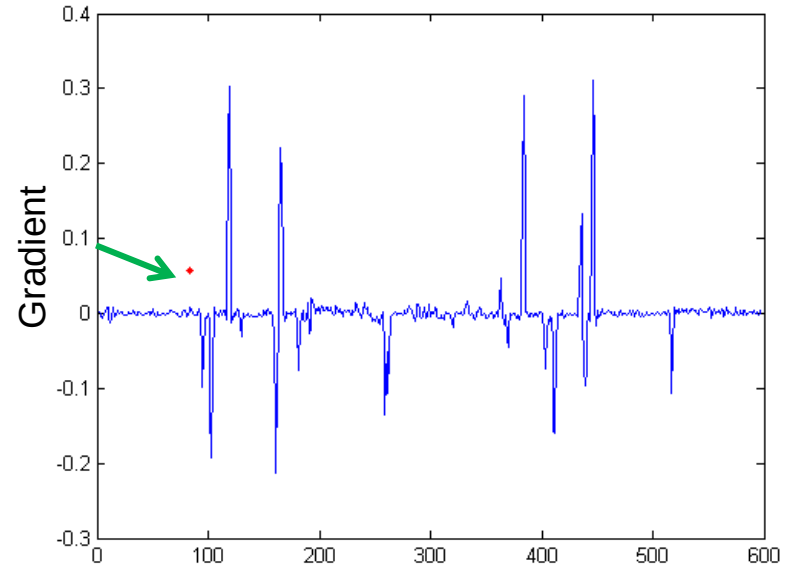
y-direction

- Which one is the gradient in the x-direction? How about y-direction?

Intensity profile



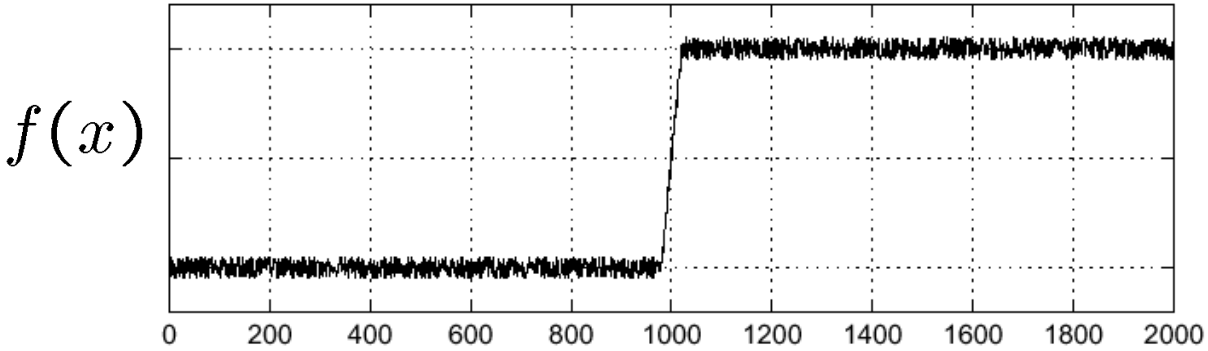


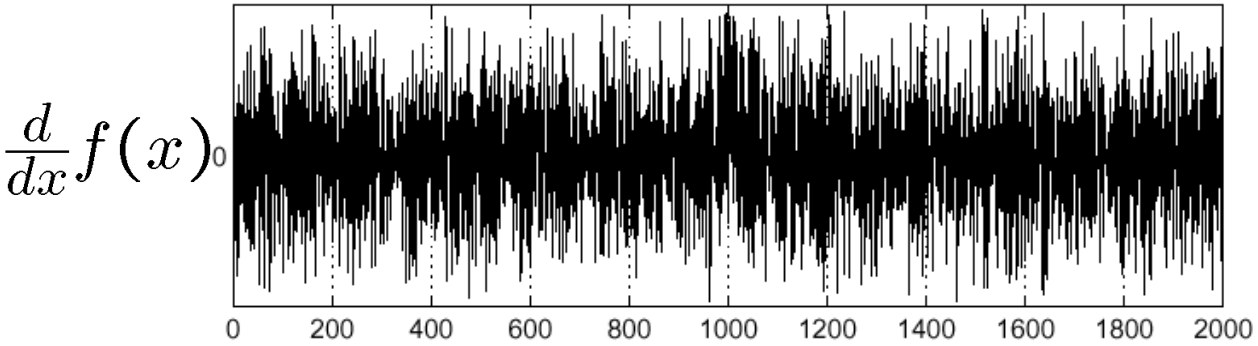


Source: D. Hoiem

Effects of noise

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

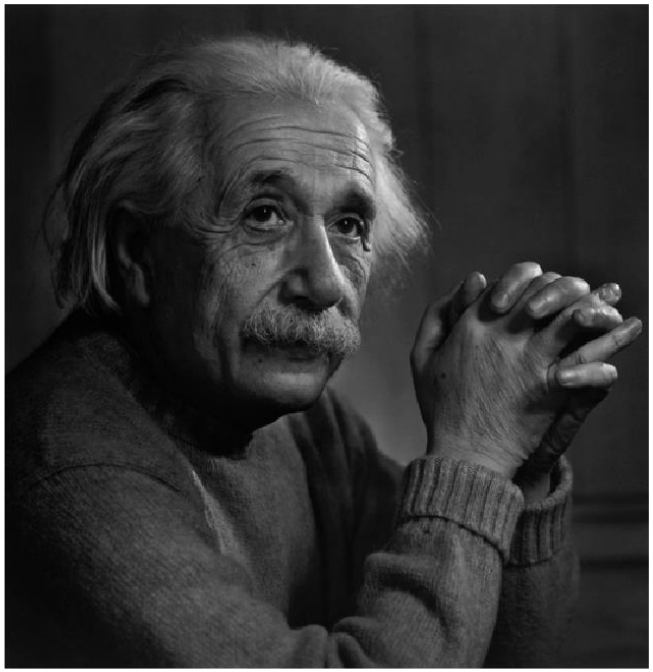


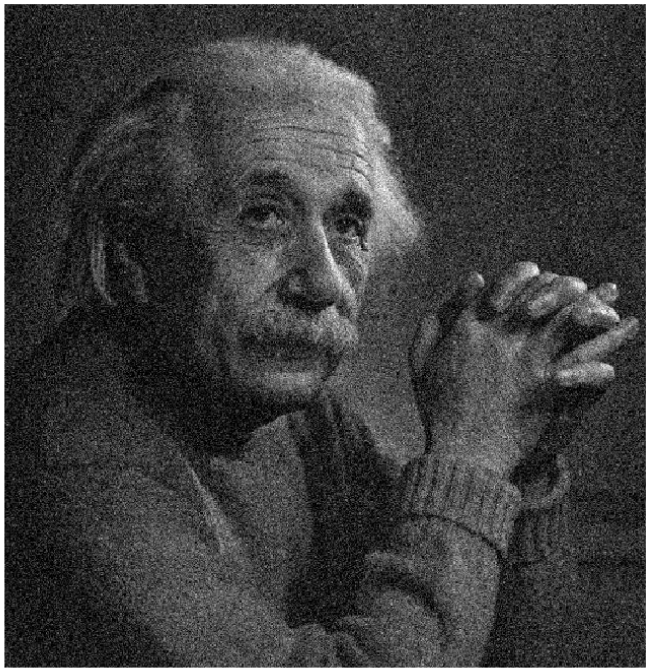


Where is the edge?

Source: S. Seitz

Effects of noise





Effects of noise

- Finite 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 is to be done?

Source: D. Forsyth

Effects of noise

- Finite 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 is to be done?
  - Smoothing the image should help, by forcing pixels different to their neighbors (=noise pixels?) to look more like neighbors

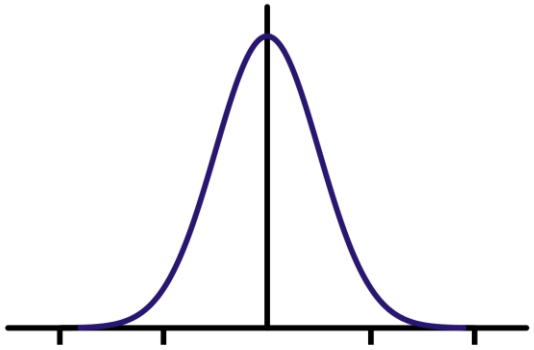
Source: D. Forsyth

Smoothing with different filters

- Mean smoothing

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

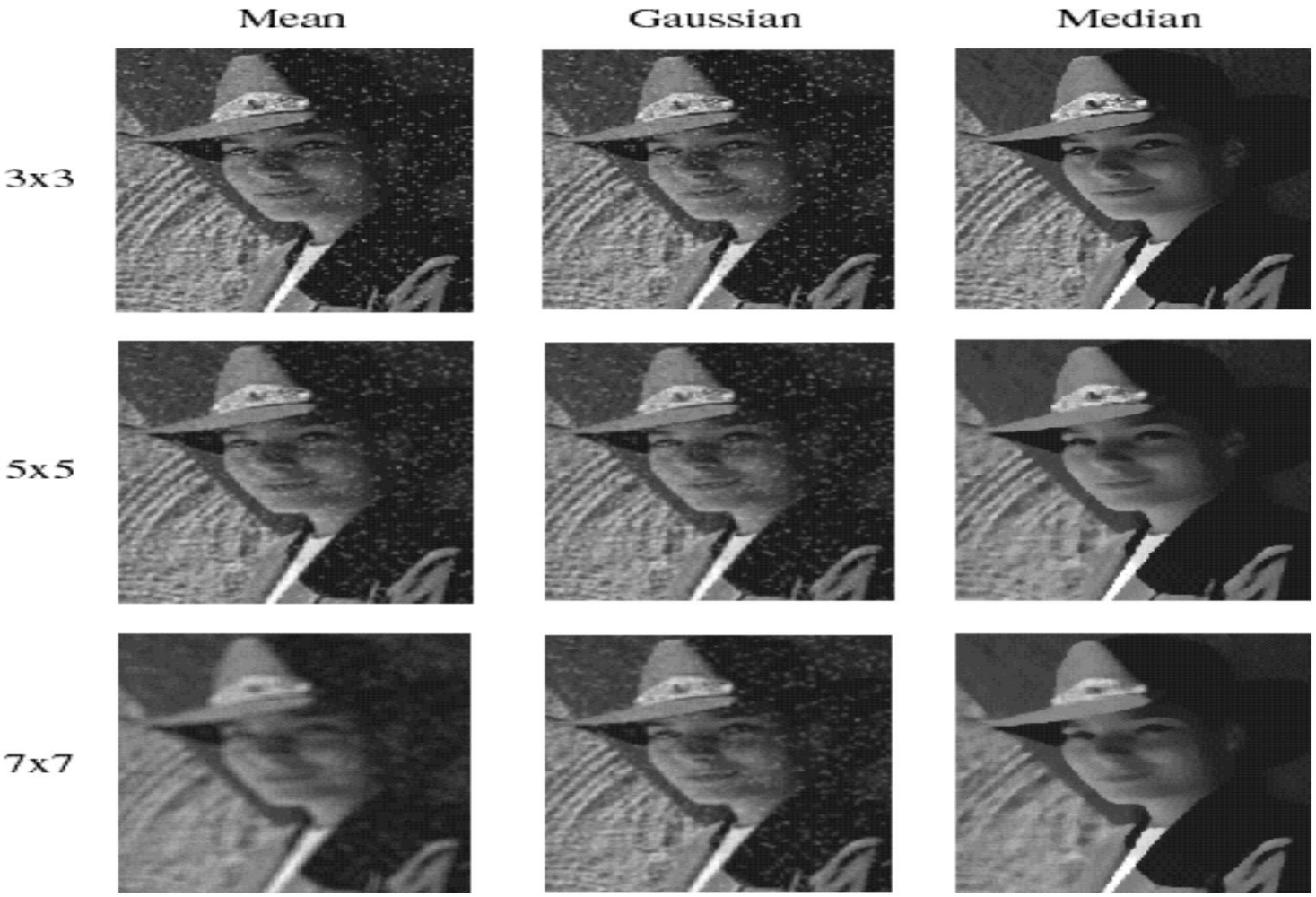
- Gaussian (smoothing * derivative)



$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

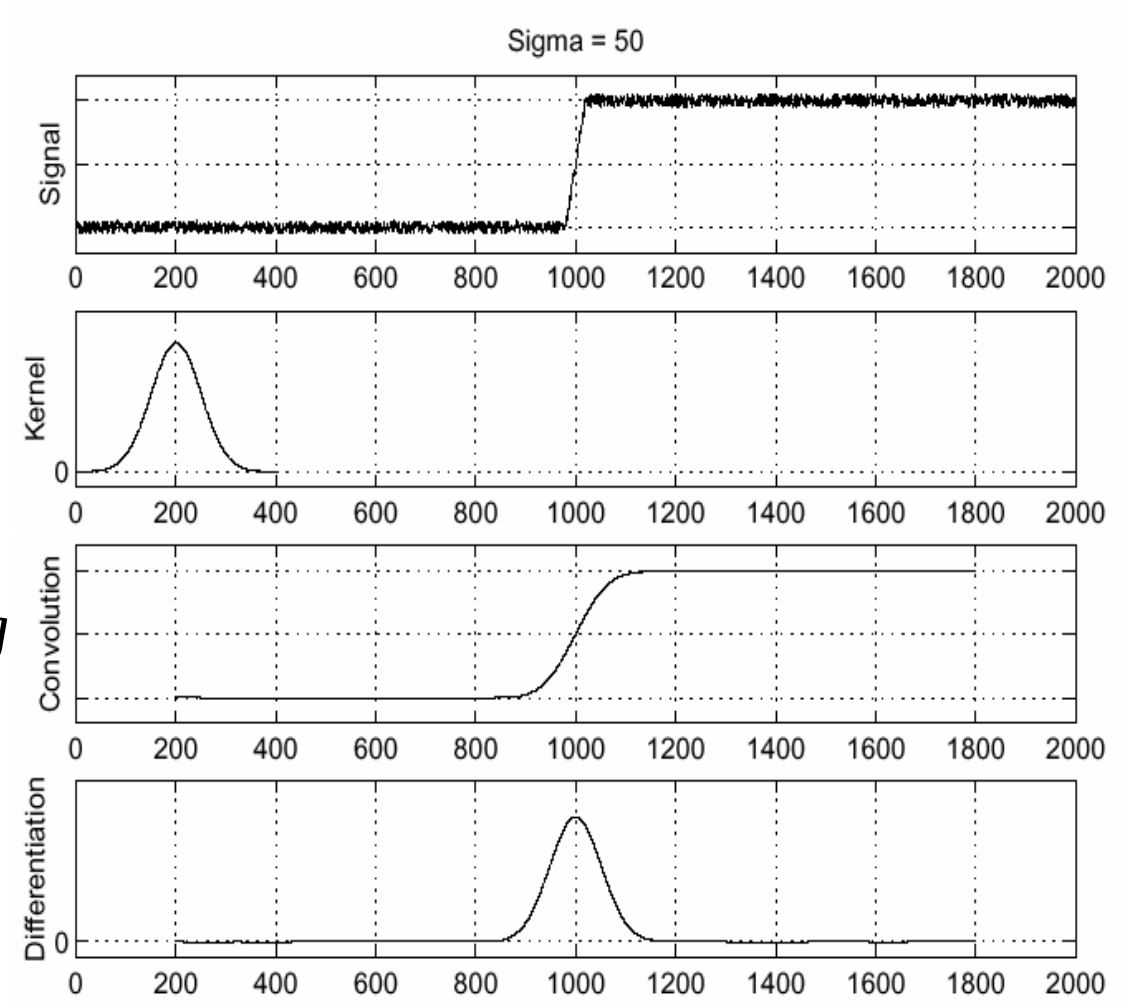
Slide credit: Steve Seitz

Smoothing with different filters



Slide credit: Steve Seitz

Solution: smooth first



f

g

$f * g$

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

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

Source: S. Seitz

Derivative theorem of convolution

- This theorem gives us a very useful property:

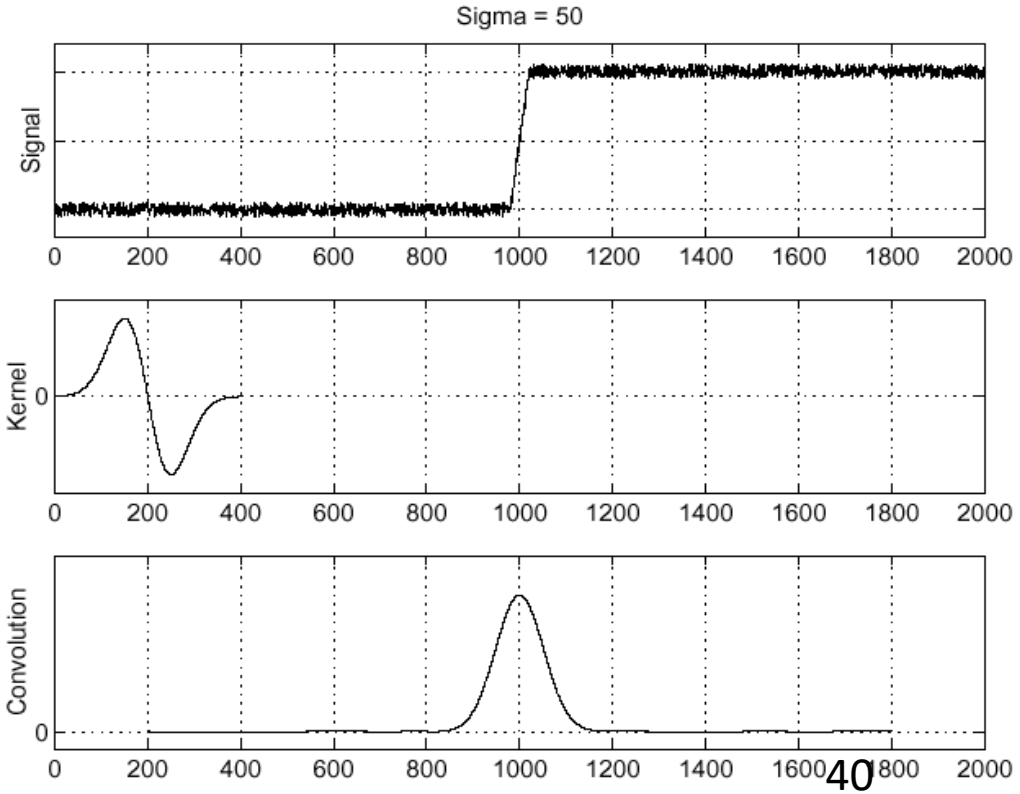
$$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$$

- This saves us one operation:

f

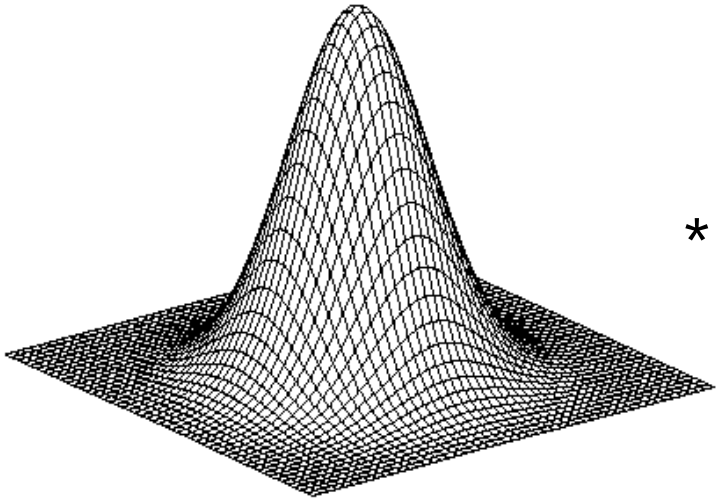
$\frac{d}{dx} g$

$f * \frac{d}{dx} g$



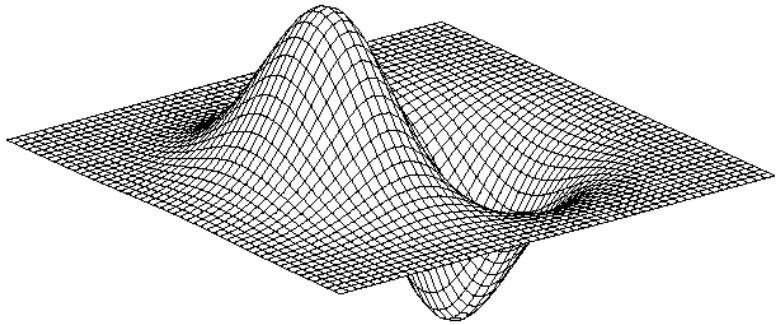
Source: S. Seitz

Derivative of Gaussian filter



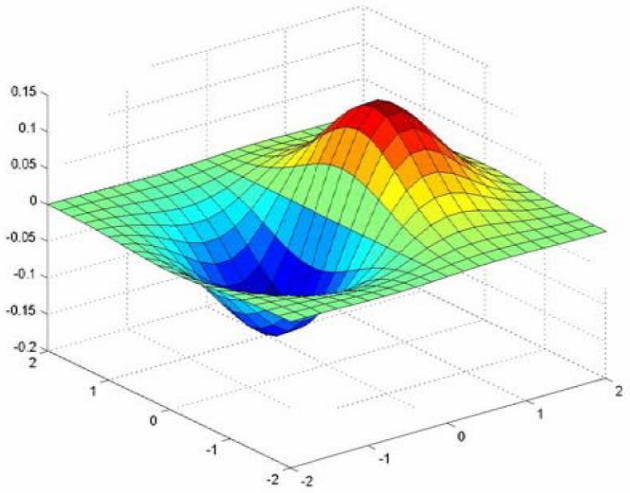
2D-gaussian

* [1 0 -1] =

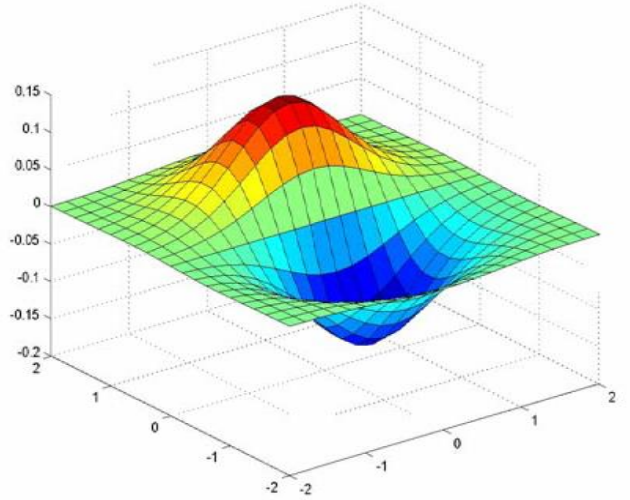


x - derivative

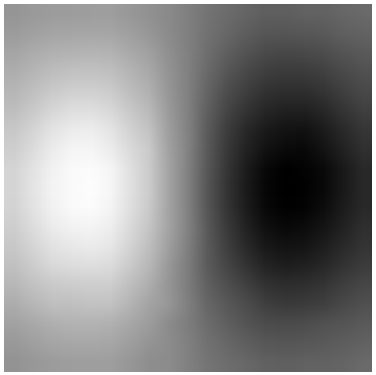
Derivative of Gaussian filter

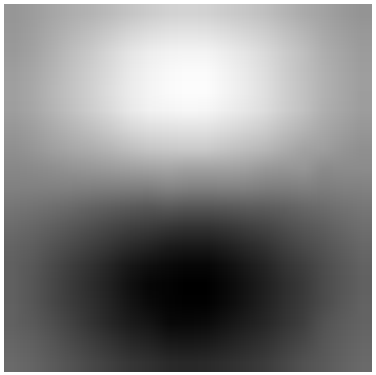


x-direction



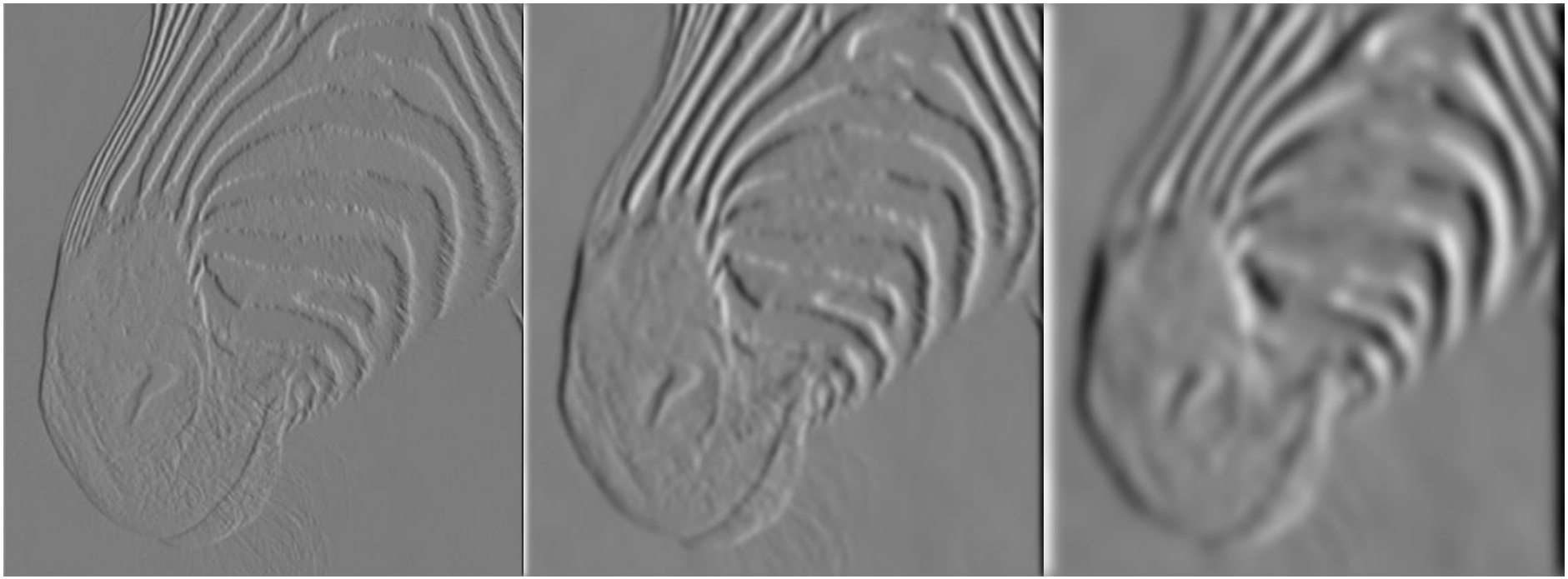
y-direction





Derivative of Gaussian filter

Tradeoff between smoothing at different scales



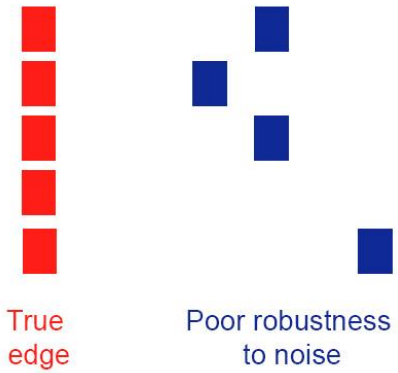
1 pixel 3 pixels 7 pixels

- Smoothed derivative removes noise, but blurs edge. Also finds edges at different “scales”.

Source: D. Forsyth

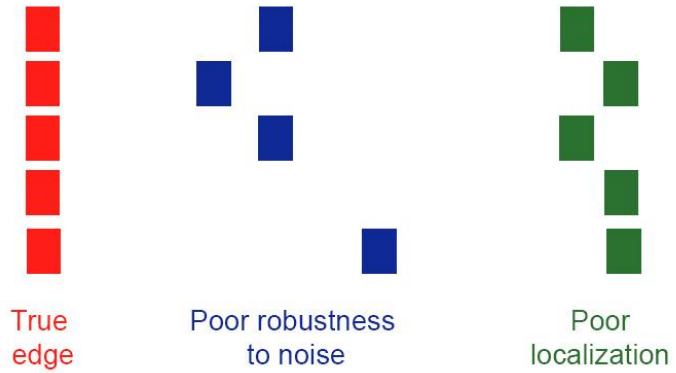
Designing an edge detector

- Criteria for an “optimal” edge detector:
  - **Good detection:** the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges)



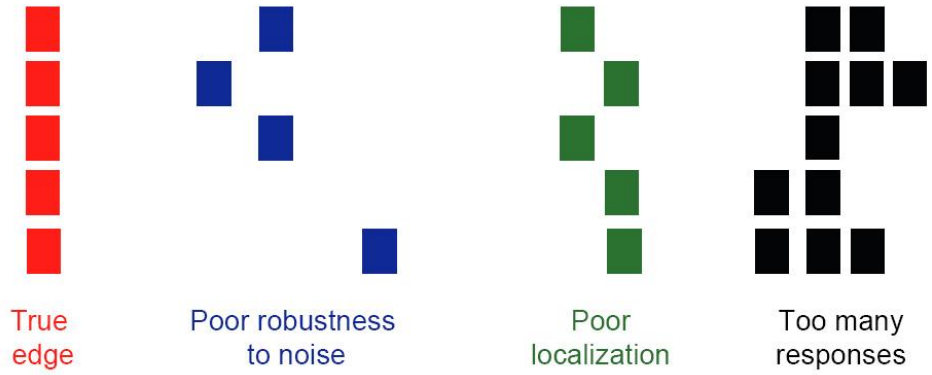
Designing an edge detector

- Criteria for an “optimal” edge detector:
  - **Good detection:** the optimal detector 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



Designing an edge detector

- Criteria for an “optimal” edge detector:
  - **Good detection:** the optimal detector 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:** the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge



What we will learn today

- Edge detection
- Image Gradients
- A simple edge detector
- Sobel Edge detector
- Canny edge detector
- Hough transform

Sobel Operator

- uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives
- one for horizontal changes, and one for vertical

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

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

Sobel Operation

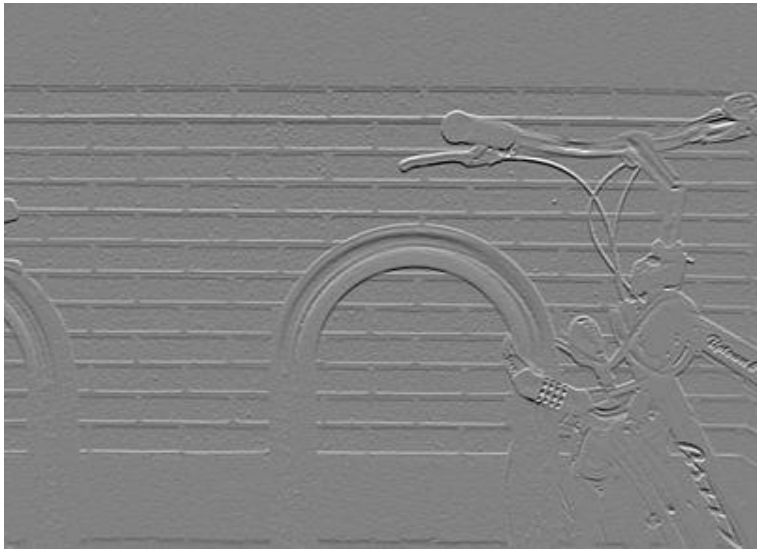
- Magnitude:

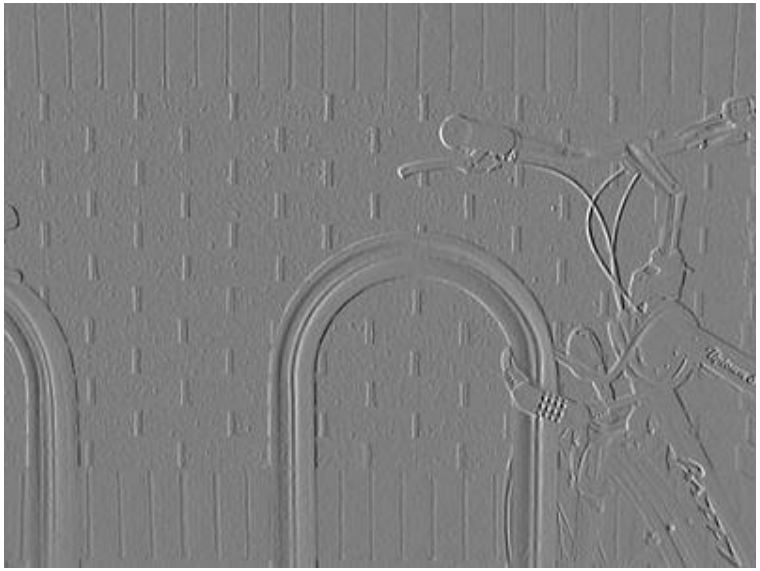
$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

- Angle or direction of the gradient:

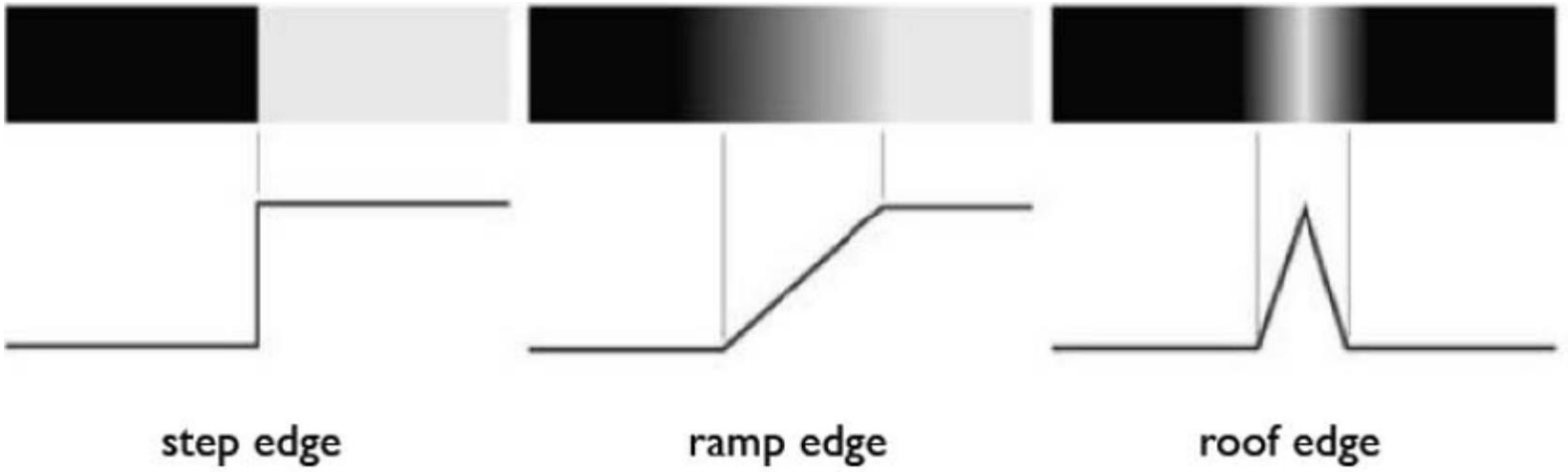
$$\mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

Sobel Filter example





Sobel Filter Problems



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

What we will learn today

- Edge detection
- Image Gradients
- A simple edge detector
- Sobel Edge detector
- Canny edge detector
- Hough Transform

Canny edge detector

- Probably the most widely used edge detector in computer vision.

J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

22,000 citations!

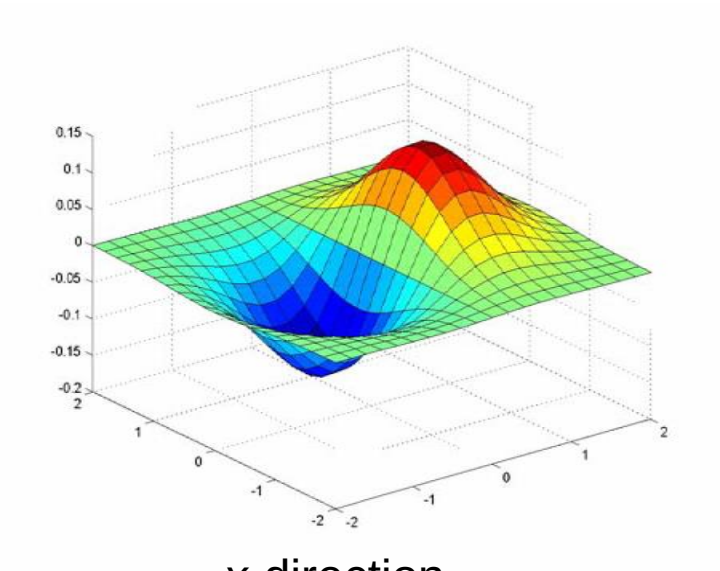
Demonstrator Image

Canny edge detector

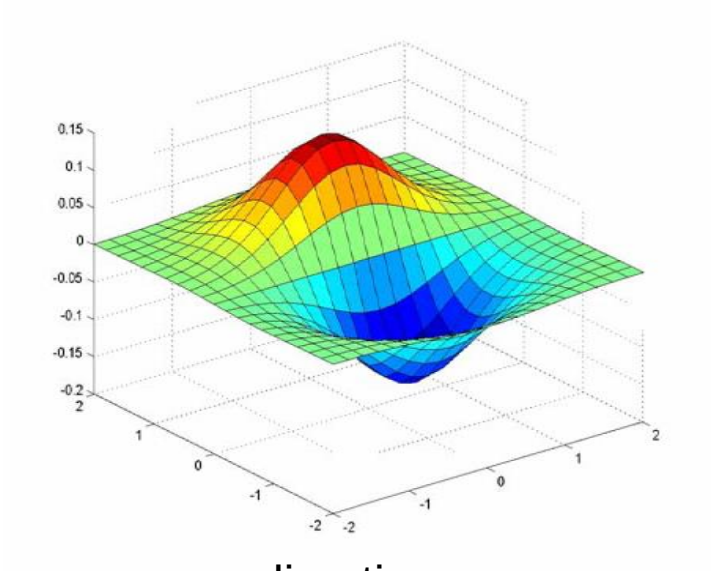
1. Filter image with x, y derivatives of Gaussian

Source: D. Lowe, L. Fei-Fei

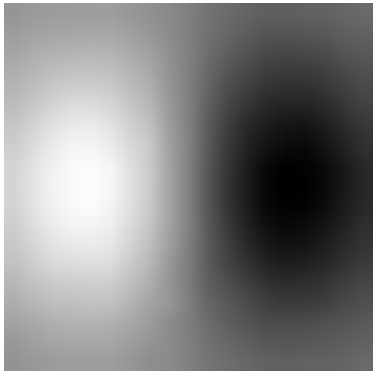
Derivative of Gaussian filter

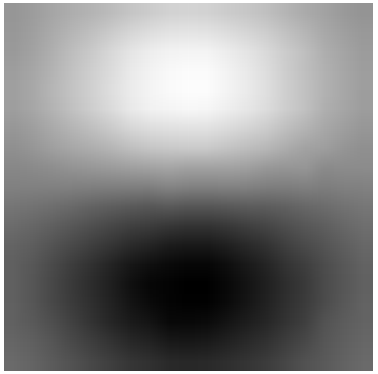


x-direction

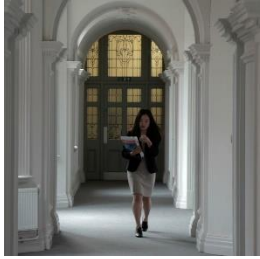


y-direction



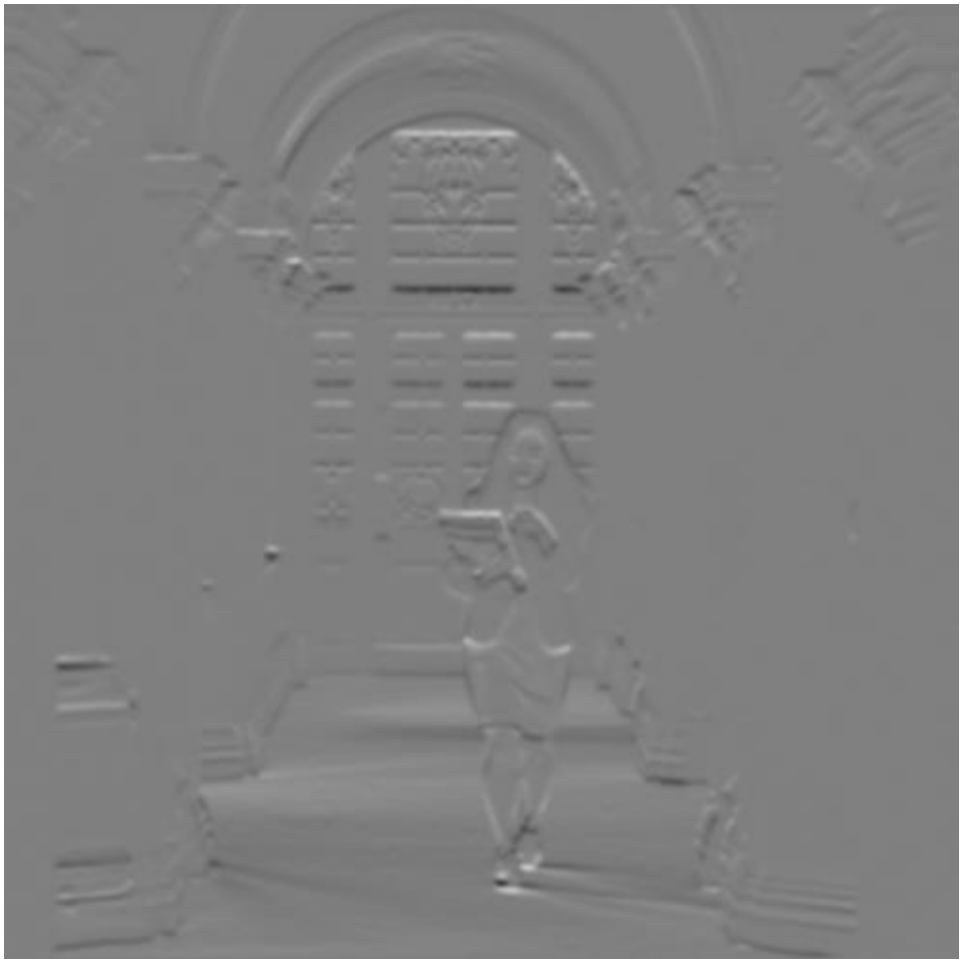


Compute Gradients



X Derivative of Gaussian

Y Derivative of Gaussian



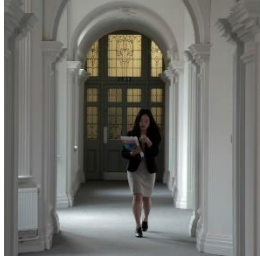
(x2 + 0.5 for visualization)

Canny edge detector

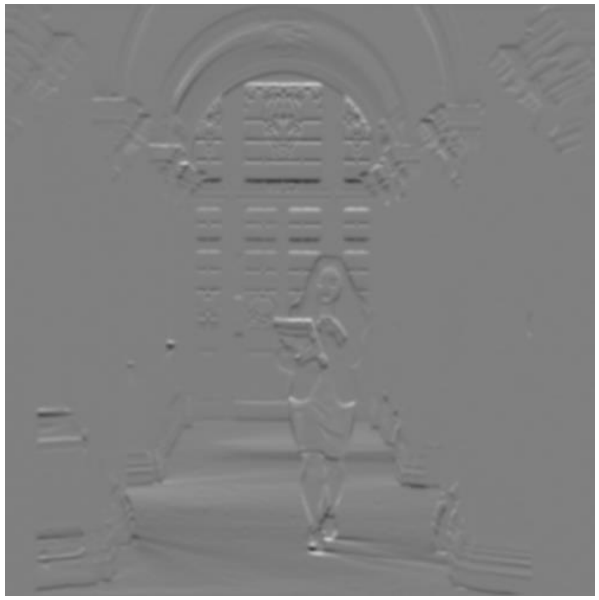
1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient

Source: D. Lowe, L. Fei-Fei

Compute Gradient Magnitude

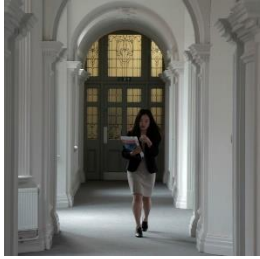


sqrt(XDerivOfGaussian .^2 + YDerivOfGaussian .^2) = gradient magnitude

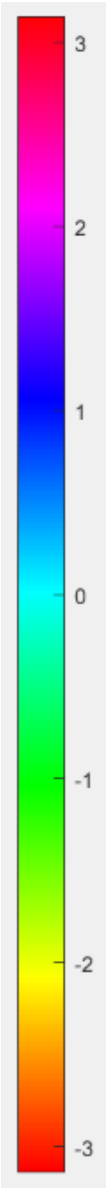


(x4 for visualization)

Compute Gradient Orientation



- Threshold magnitude at minimum level
- Get orientation via theta = atan2(gy, gx)

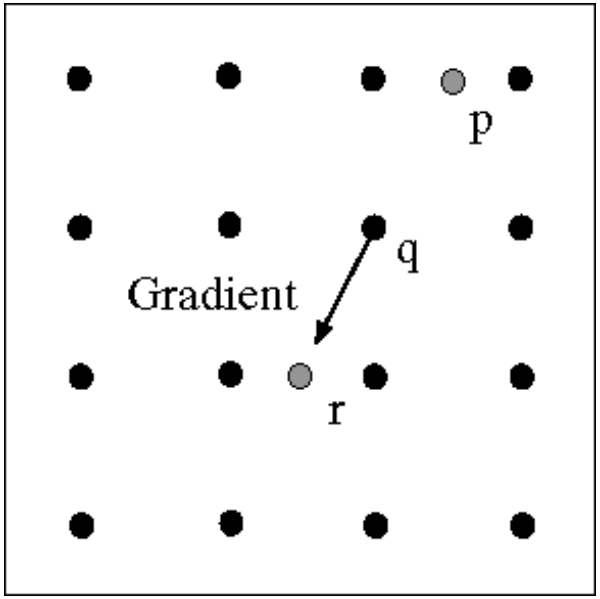


Canny edge detector

1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
   - Thin multi-pixel wide “ridges” to single pixel width

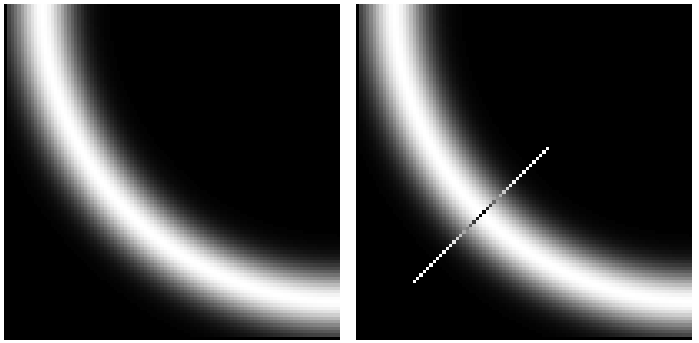
Source: D. Lowe, L. Fei-Fei

Non-maximum suppression for each orientation



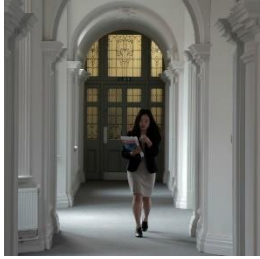
At pixel q:
We have a maximum if the value is larger than those at both p and at r.

Interpolate along gradient direction to get these values.



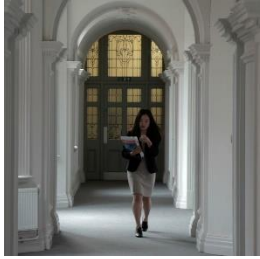
Source: D. Forsyth

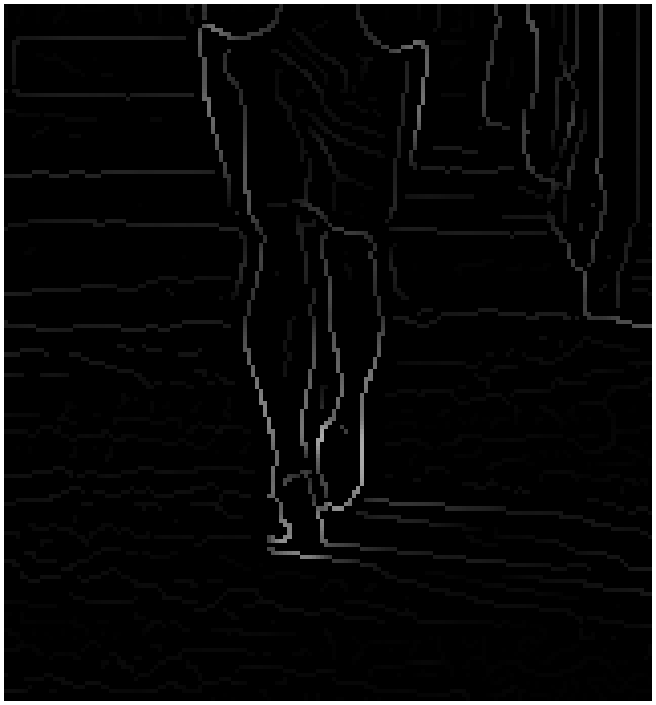
Before Non-max Suppression



Gradient magnitude (x4 for visualization)

After non-max suppression





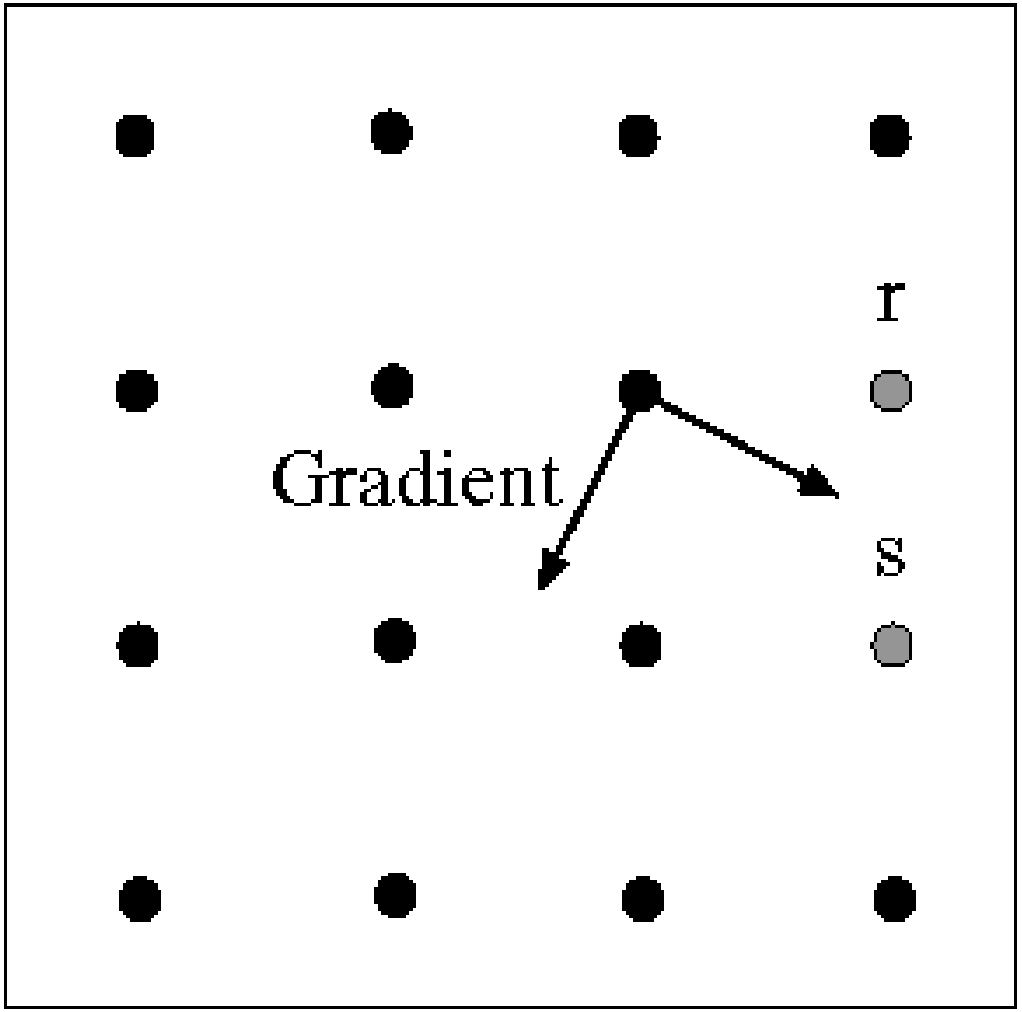
Gradient magnitude (x4 for visualization)

Canny edge detector

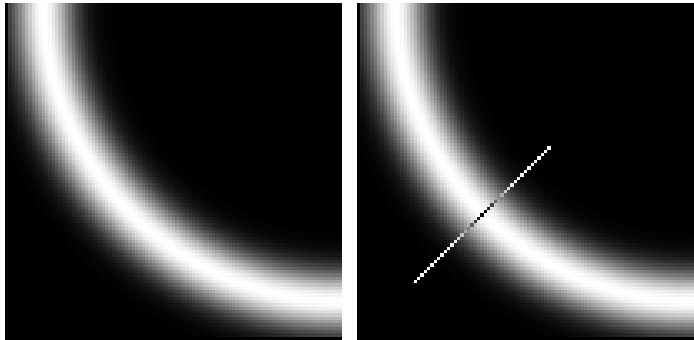
1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
   - Thin multi-pixel wide “ridges” to single pixel width
4. ‘Hysteresis’ Thresholding

Source: D. Lowe, L. Fei-Fei

Edge linking



Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).

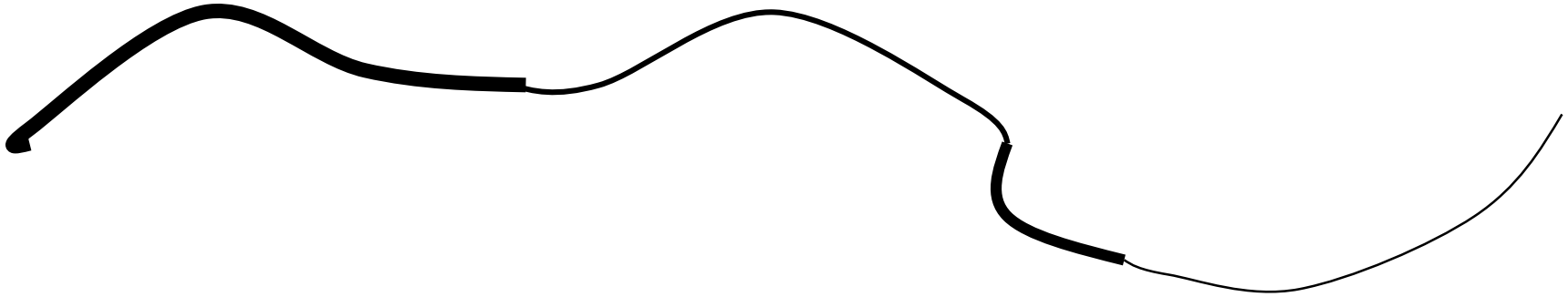


Source: D. Forsyth

‘Hysteresis’ thresholding

- Two thresholds – high and low
- Grad. mag. > high threshold? = strong edge
- Grad. mag. < low threshold? noise
- In between = weak edge

- ‘Follow’ edges starting from strong edge pixels
- Continue them into weak edges
    - Connected components (Szeliski 3.3.4)



Source: S. Seitz

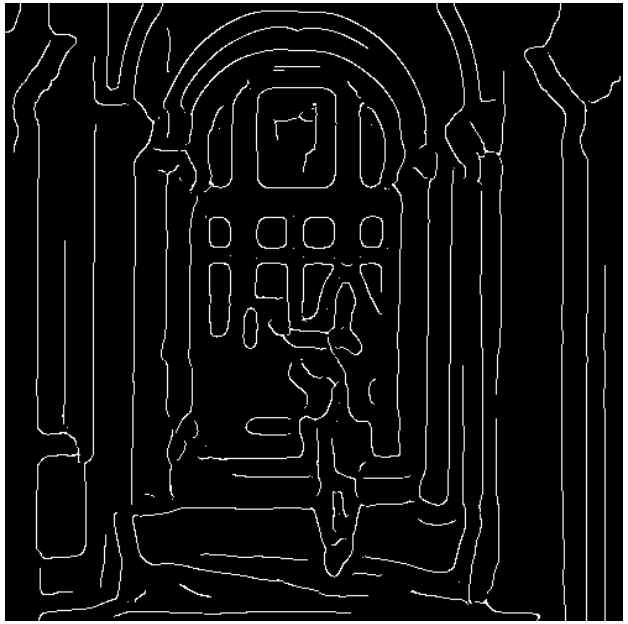
Final Canny Edges

$$\sigma = \sqrt{2}, t_{low} = 0.05, t_{high} = 0.1$$

Effect of σ (Gaussian kernel spread/size)

Original

$\sigma = \sqrt{2}$



$\sigma = 4\sqrt{2}$

The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

Source: S. Seitz

Gradients (e.g. Canny)

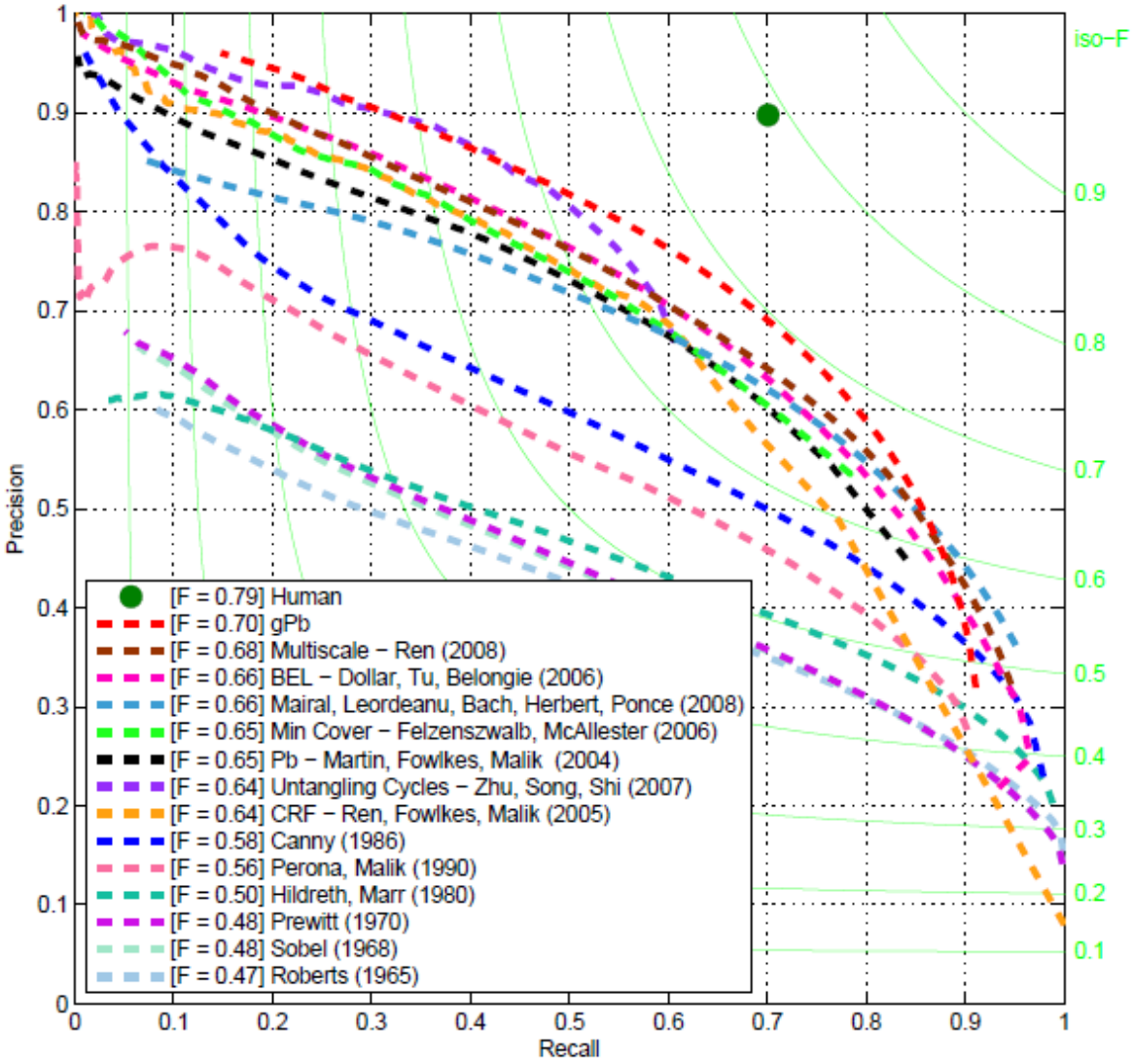
Color

Texture

Combined

Human

45 years of boundary detection



Source: Arbelaez, Maire, Fowlkes, and Malik. TPAMI 2011 (pdf)

What we will learn today

- Edge detection
- Image Gradients
- A simple edge detector
- Sobel Edge detector
- Canny edge detector
- Hough Transform

Intro to Hough transform

- The Hough transform (HT) can be used to detect lines.
- It was introduced in 1962 (Hough 1962) and first used to find lines in images a decade later (Duda 1972).
- The goal is to find the location of lines in images.
- **Caveat**: Hough transform can detect lines, circles and other structures ONLY if their parametric equation is known.
- It can give robust detection under noise and partial occlusion

Prior to Hough transform

- Assume that we have performed some edge detection, and a thresholding of the edge magnitude image.
- Thus, we have some pixels that may partially describe the boundary of some objects.



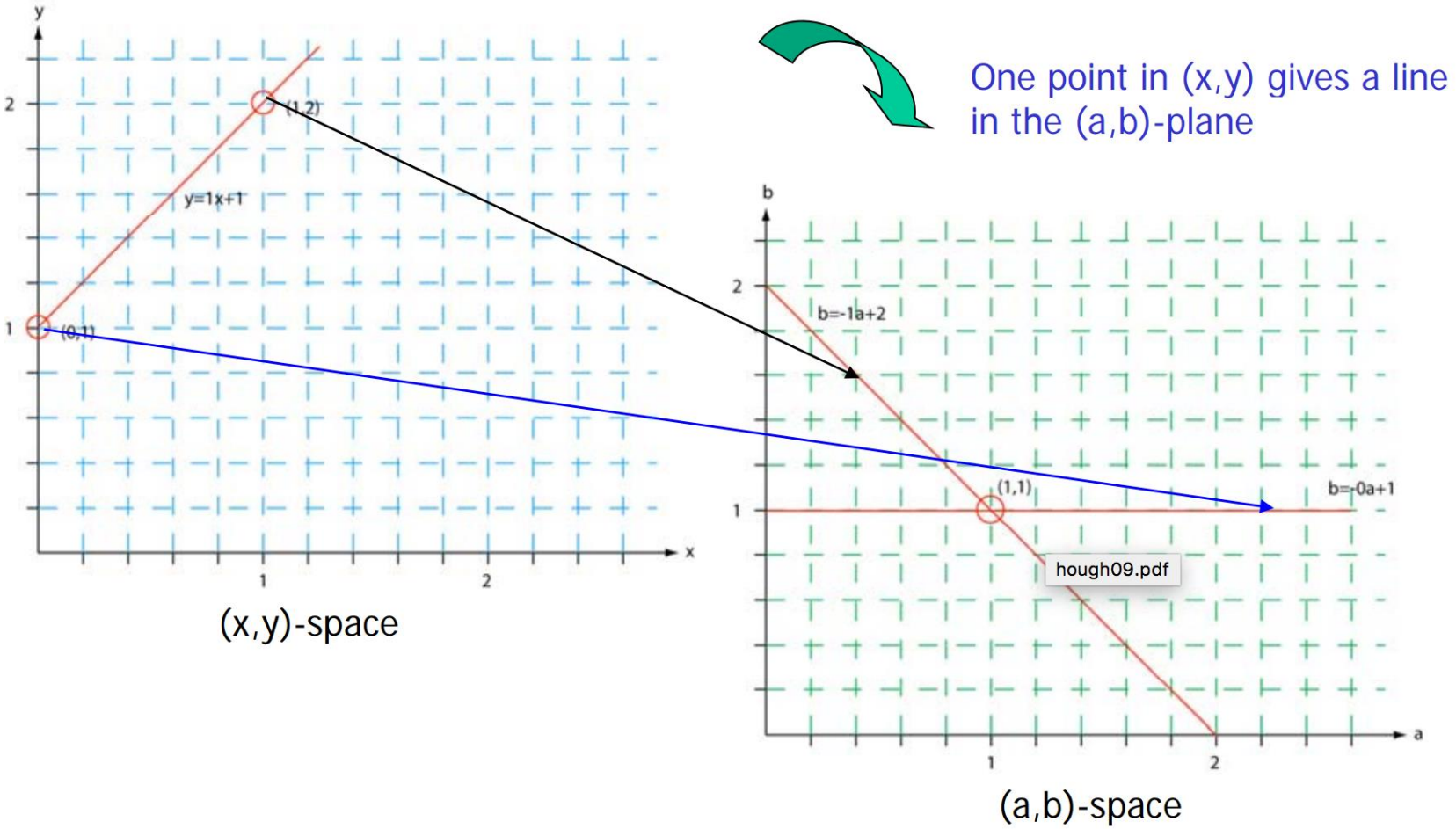
Detecting lines using Hough transform

- We wish to find sets of pixels that make up straight lines.
- Consider a point of known coordinates $(x_i;y_i)$
  - There are many lines passing through the point (x_i, y_i).
- Straight lines that pass that point have the form $y_i = a*x_i + b$
  - Common to them is that they satisfy the equation for some set of parameters (a, b)

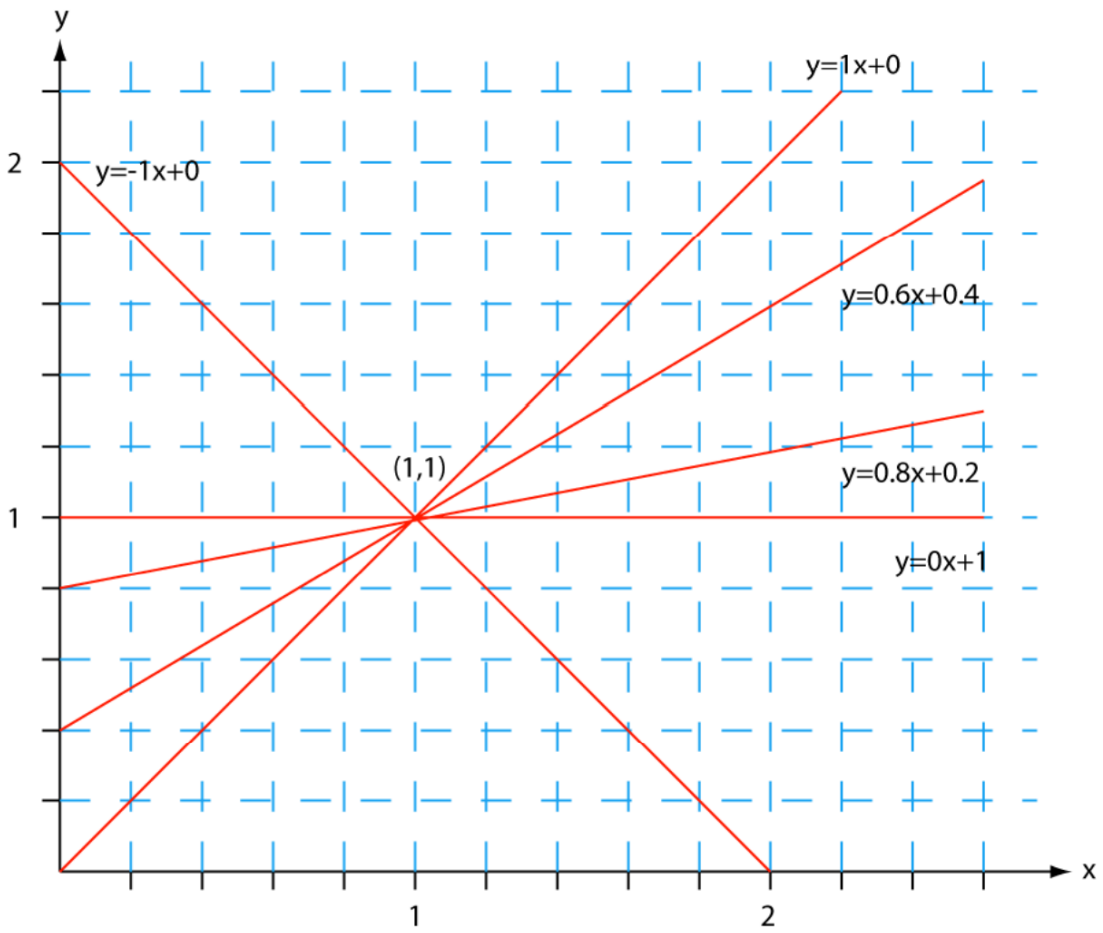
Detecting lines using Hough transform

- This equation can obviously be rewritten as follows:
  - $b = -a*x_i + y_i$
  - We can now consider x and y as parameters
  - a and b as variables.
- This is a line in (a, b) space parameterized by x and y.
  - So: a single point in x_1,y_1-space gives a line in (a,b) space.
  - Another point (x_2, y_2) will give rise to another line (a,b) space.

Detecting lines using Hough transform



Detecting lines using Hough transform



Detecting lines using Hough transform

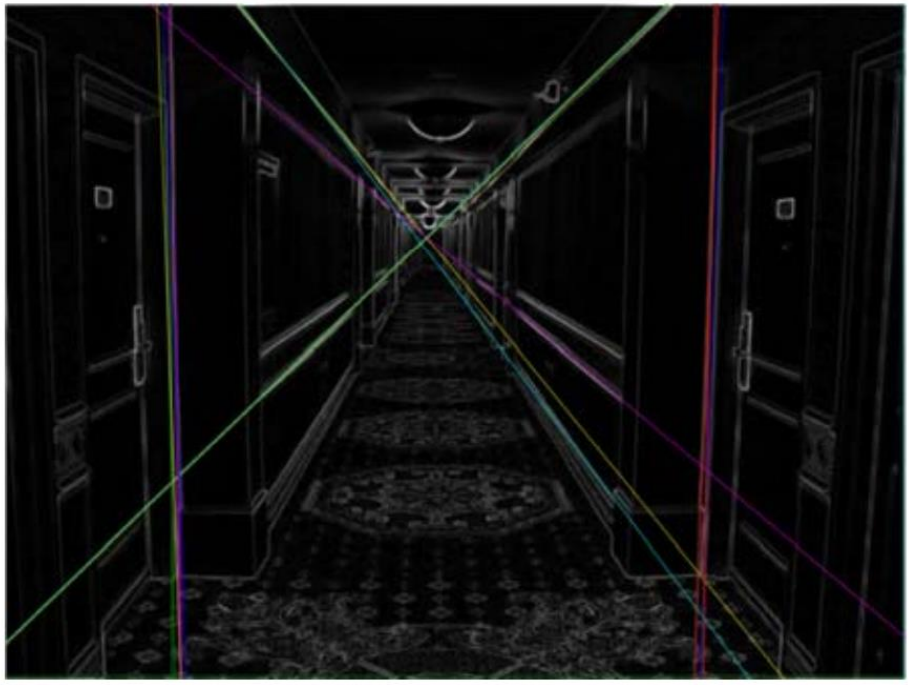
- Two points (x_1, y_1) and$(x_2\ y_2)$ define a line in the (x, y) plane.
- These two points give rise to two different lines in (a,b) space.
- In (a,b) space these lines will intersect in a point (a' b')
- All points on the line defined by (x_1, y_1) and $(x_2\ , y_2)$ in (x, y) space will parameterize lines that intersect in (a', b') in (a,b) space.

Algorithm for Hough transform

- Quantize the parameter space (a b) by dividing it into cells
- This quantized space is often referred to as the accumulator cells.
- Count the number of times a line intersects a given cell.
  - For each pair of points (x_1, y_1) and (x_2, y_2) detected as an edge, find the intersection (a',b') in (a, b)space.
  - Increase the value of a cell in the range $[[a_{min}, a_{max}],[b_{min},b_{max}]]$ that (a', b') belongs to.
  - Cells receiving more than a certain number of counts (also called 'votes') are assumed to correspond to lines in (x,y) space.

Output of Hough transform

- Here are the top 20 most voted lines in the image:

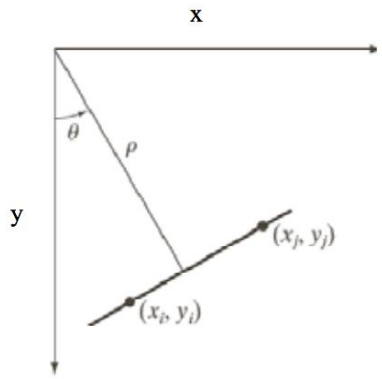


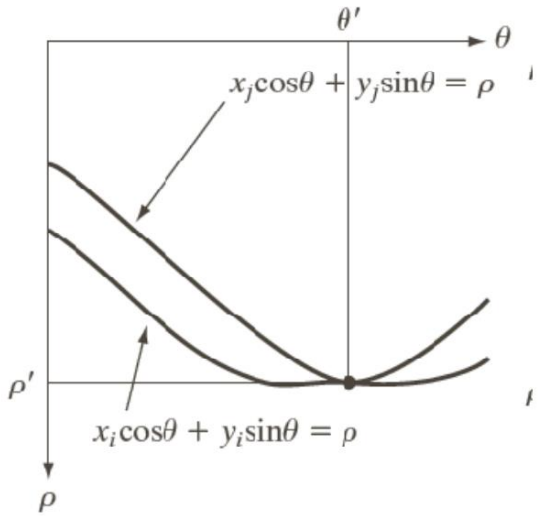
Other Hough transformations

- We can represent lines as polar coordinates instead of y = a*x + b
- Polar coordinate representation:
  - $x*\cos\theta + y*\sin\theta = \rho$
- Can you figure out the relationship between
  - (x y) and $(\rho\ \theta)$?

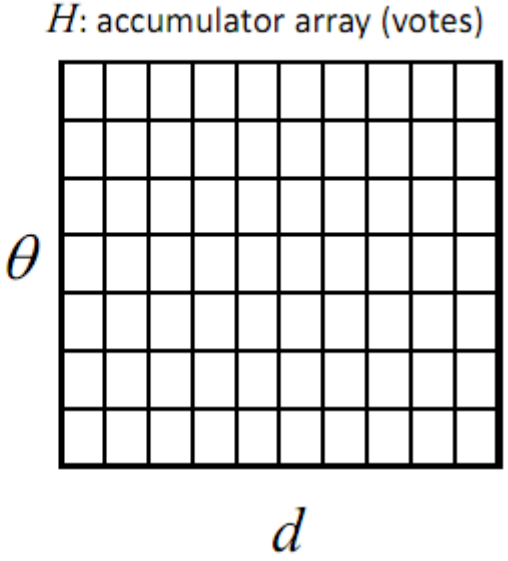
Other Hough transformations

- Note that lines in (x y) space are not lines in (ρ θ) space, unlike (a b) space.
- A vertical line will have θ=0 and ρ equal to the intercept with the x-axis.
- A horizontal line will have θ=90 and ρ equal to the intercept with the y-axis.





Other Hough transformations



Using the polar parameterization:

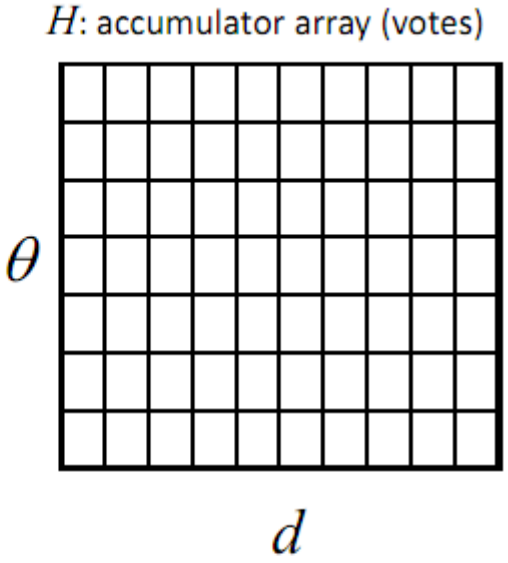
$$x\cos\theta + y\sin\theta = d$$

Basic Hough transform algorithm

1. Initialize $H[d,\theta] = 0$.
2. For each edge point (x,y) in the image
   for $\theta \in [0,\pi)$ // some quantization
   H[d, θ] += 1
3. Find the value(s) of (d,θ) where $H[d,\theta]$ is maximum.
4. The detected line in the image is given by

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

Other Hough transformations



Using the polar parameterization:

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

Basic Hough transform algorithm

1. Initialize $H[d,\theta] = 0$.
2. For each edge point (x,y) in the image
   for $\theta \in [0,\pi)$ // some quantization
   H[d, θ] += 1
3. Find the value(s) of (d,θ) where $H[d,\theta]$ is maximum.
4. The detected line in the image is given by

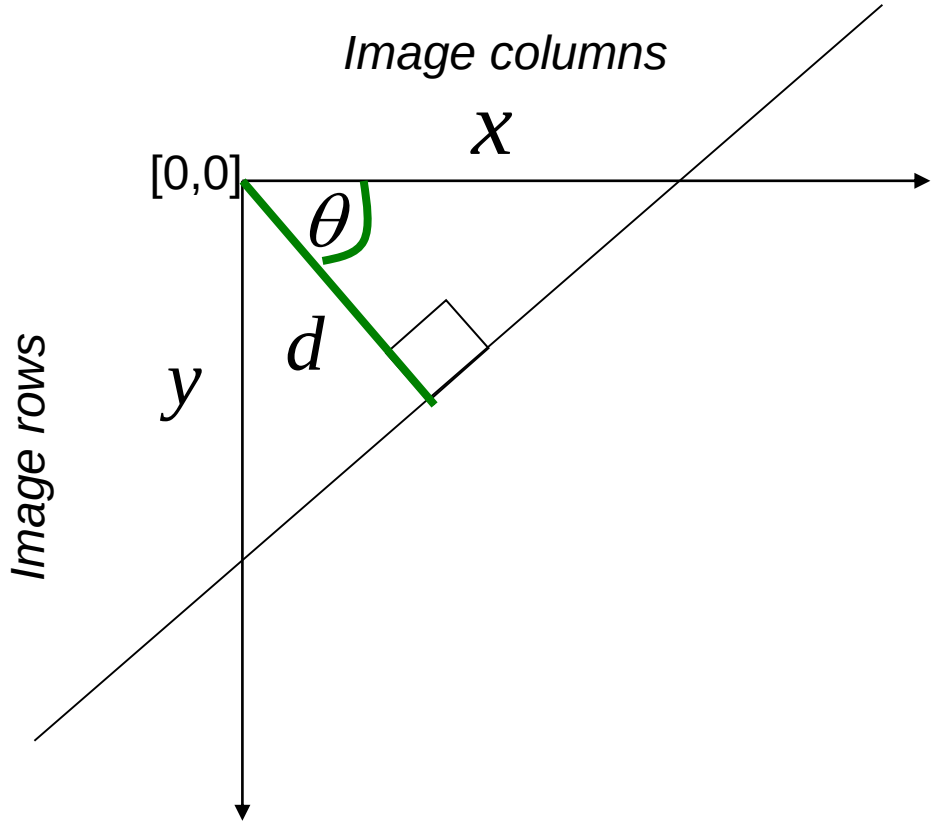
$$d = x\cos\theta + y\sin\theta$$

Example video

- https://youtu.be/4zHbI-fFIII?t=3m35s

Polar representation for lines

Issues with usual (*m,b*) parameter space: can take on infinite values, undefined for vertical lines.



d : perpendicular distance from line to origin

θ : angle the perpendicular makes with the x-axis

$$x\cos\theta - y\sin\theta = d$$

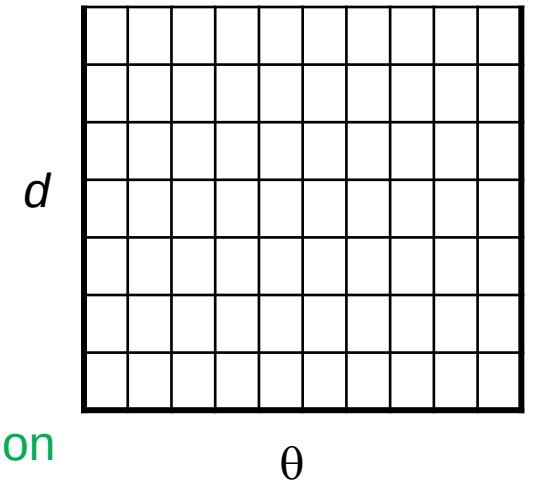
Point in image space → sinusoid segment in Hough space

Hough transform algorithm

Using the polar parameterization:

$$x\cos\theta - y\sin\theta = d$$

H: accumulator array (votes)



Basic Hough transform algorithm

1. Initialize H[d, θ]=0
2. for each edge point I[x,y] in the image

   for θ = [θ_{min} to θ_{max}] // some quantization

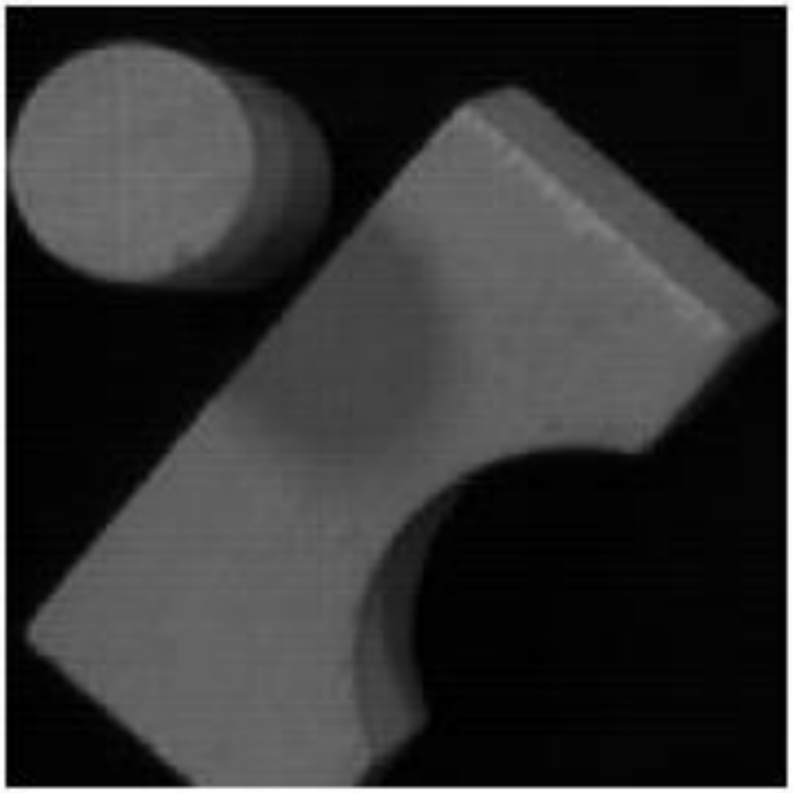
   $d = x\cos\theta - y\sin\theta$

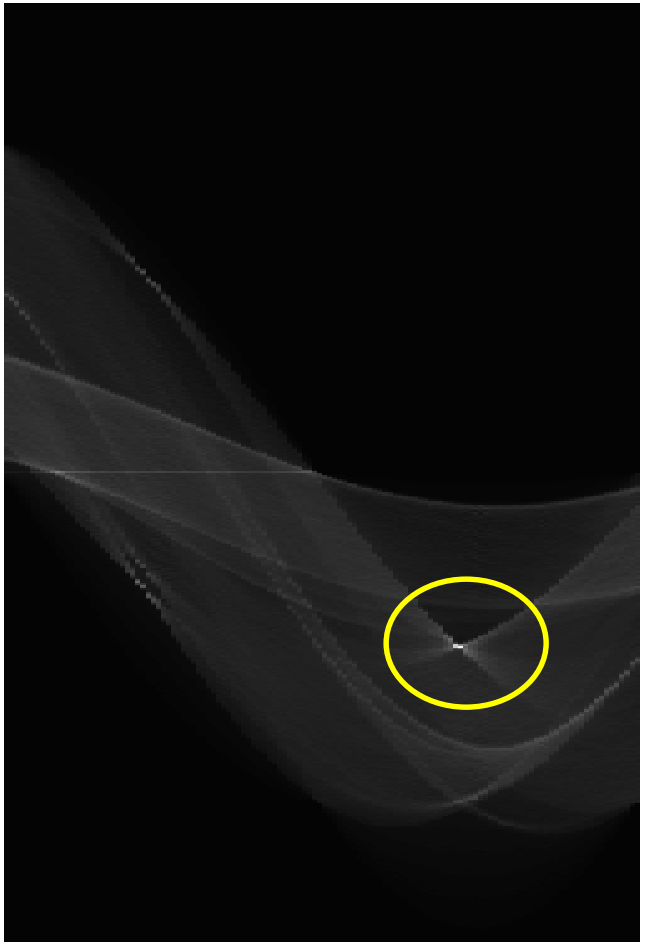
   H[d, θ] += 1
3. Find the value(s) of (d, θ) where H[d, θ] is maximum
4. The detected line in the image is given by $d = x\cos\theta - y\sin\theta$

Time complexity (in terms of number of votes per pt)?

Source: Steve Seitz

Example: Hough transform for straight lines





Which line generated this peak?

Impact of noise on Hough

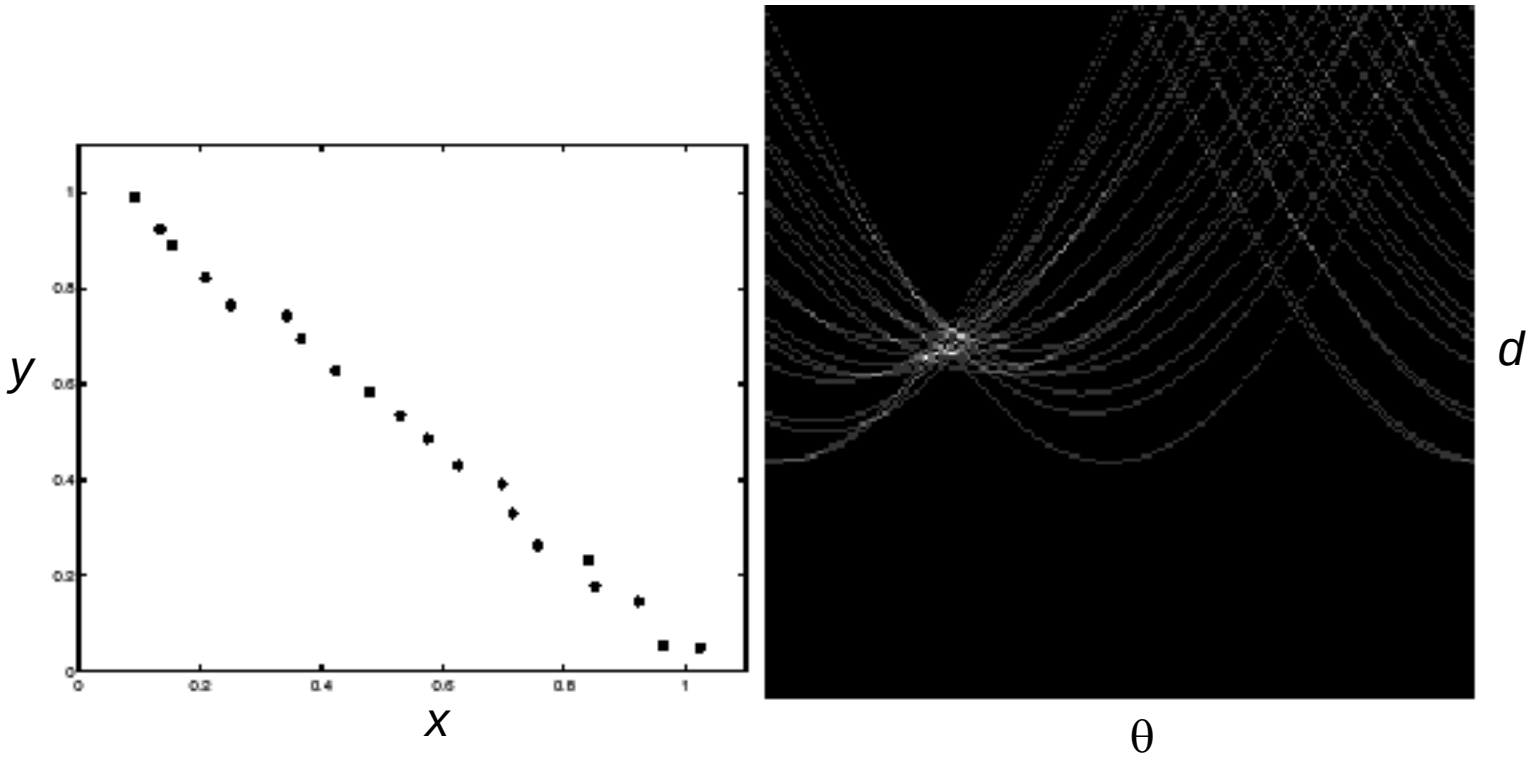


Image space edge coordinates

Votes

What difficulty does this present for an implementation?

Impact of noise on Hough

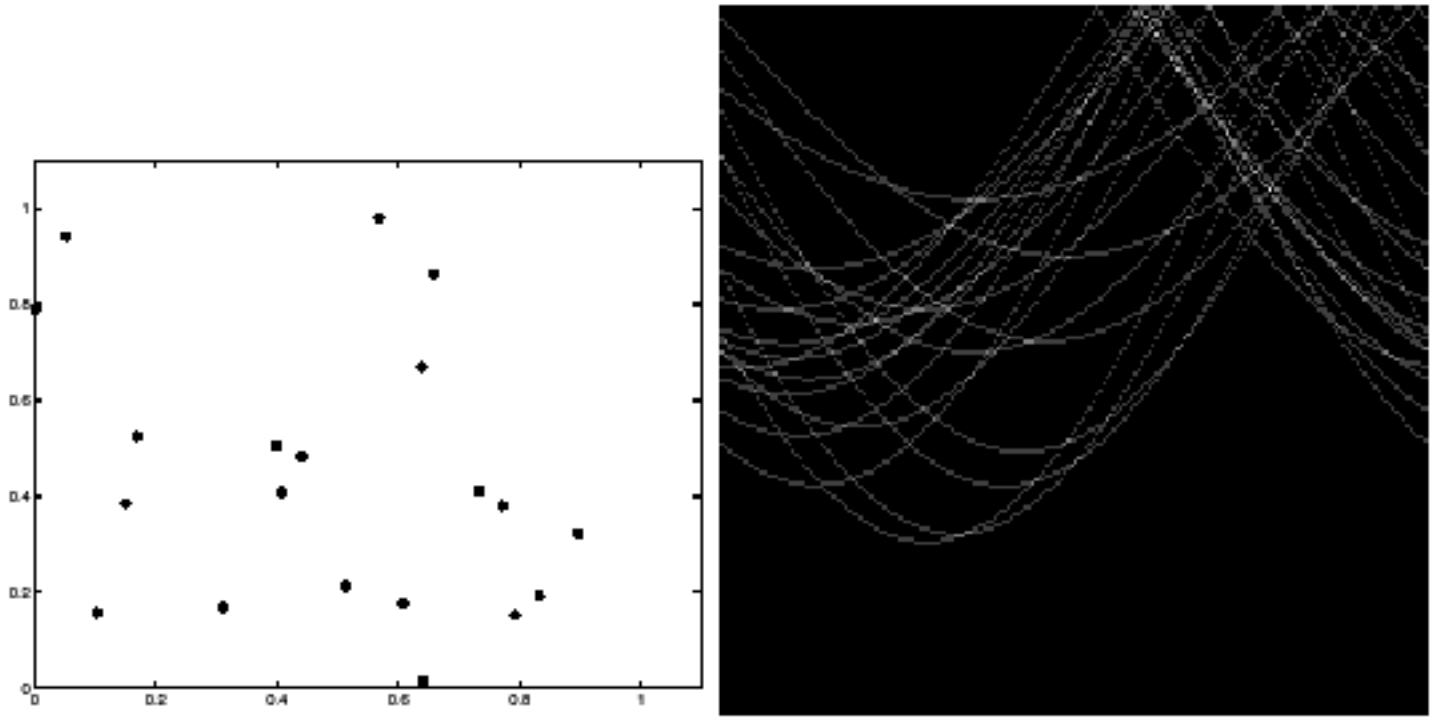


Image space edge coordinates

Votes

Here, everything appears to be “noise”, or random edge points, but we still see peaks in the vote space.

Dealing with noise

- 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)
- Try to get rid of irrelevant features
  - E.g., take only edge points with significant gradient magnitude

Extensions

Extension 1: Use the image gradient

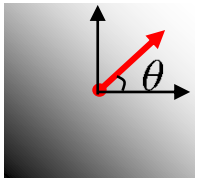
1. same
2. for each edge point I[x,y] in the image

   θ = gradient at (x,y)

   $$d = x\cos\theta - y\sin\theta$$

   H[d, θ] += 1
3. same
4. same

(Reduces degrees of freedom)



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

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

Extensions

Extension 1: Use the image gradient

1. same
2. for each edge point I[x,y] in the image
   compute unique (d, θ) based on image gradient at (x,y)
   H[d, θ] += 1
3. same
4. same

(Reduces degrees of freedom)

Extension 2

- give more votes for stronger edges (use magnitude of gradient)

Extension 3

- change the sampling of (d, θ) to give more/less resolution

Extension 4

- The same procedure can be used with circles, squares, or any other shape…

Source: Steve Seitz

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

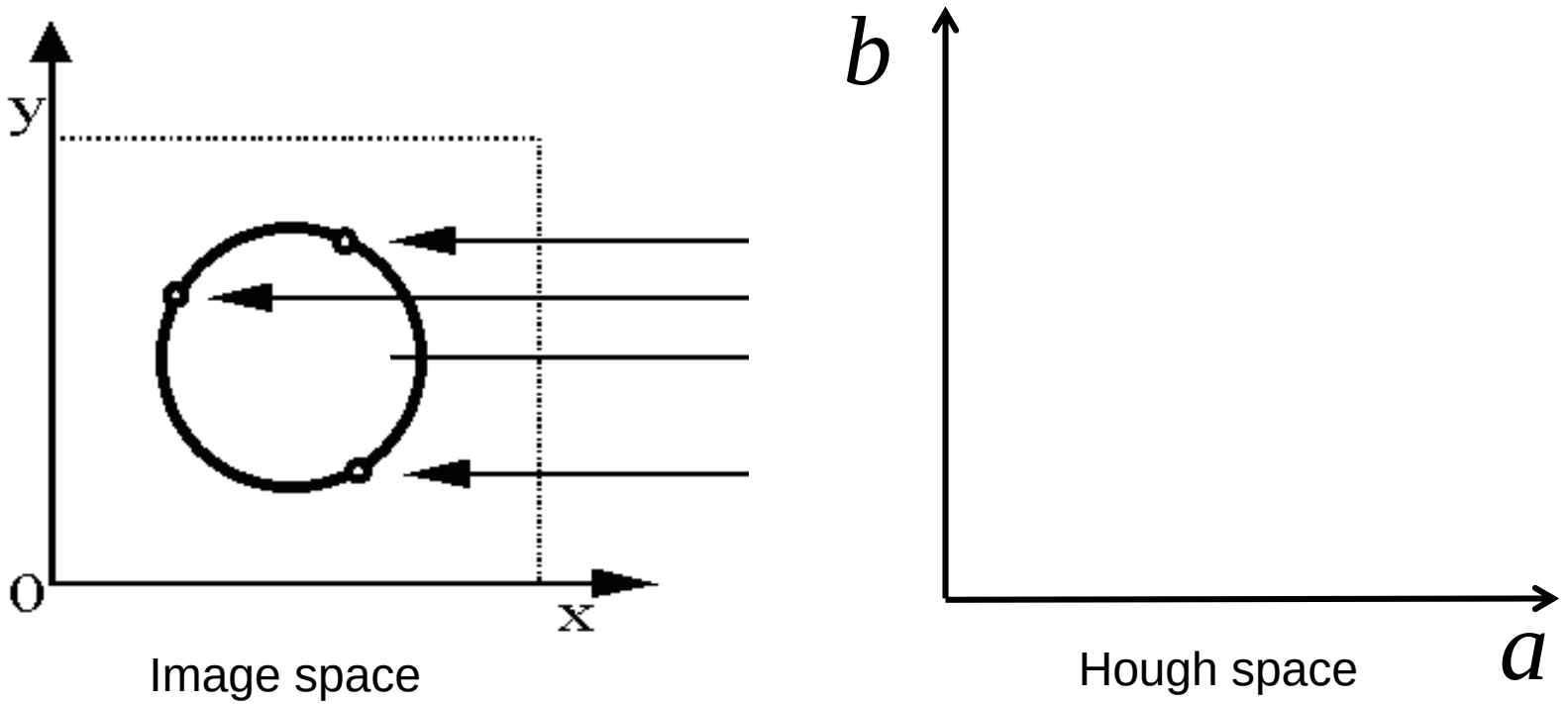


Image space

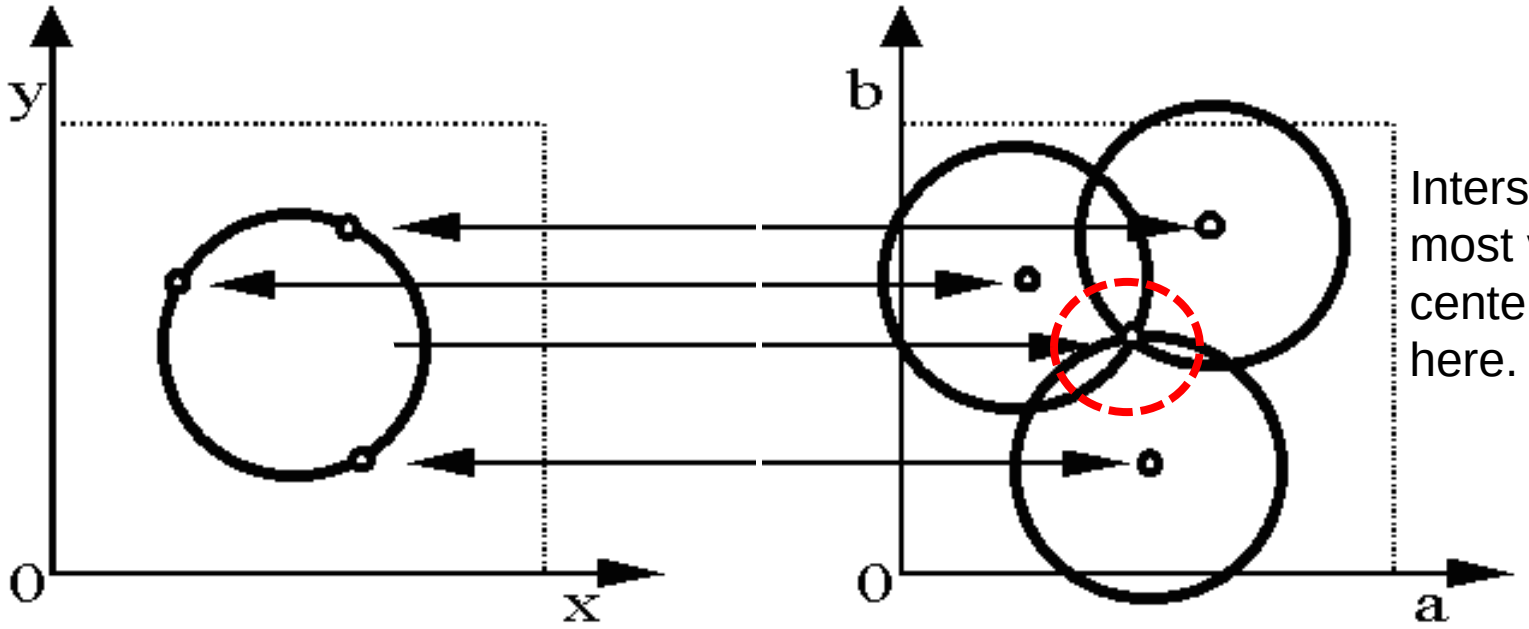
Hough space

Hough transform for circles

- Circle: center (a,b) and radius r

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

- For a fixed radius r, unknown gradient direction

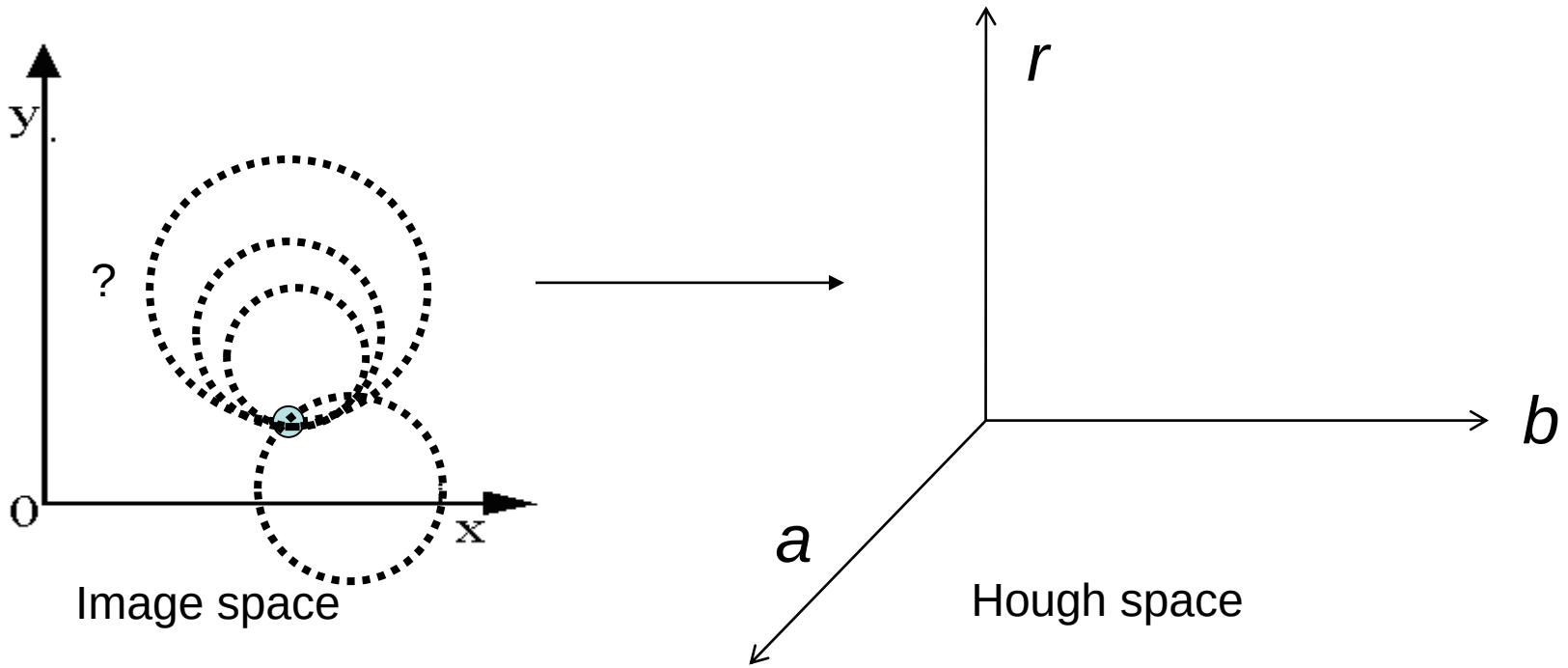


Hough transform for circles

- Circle: center (a,b) and radius r

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

- For an unknown 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 an unknown radius r, unknown gradient direction

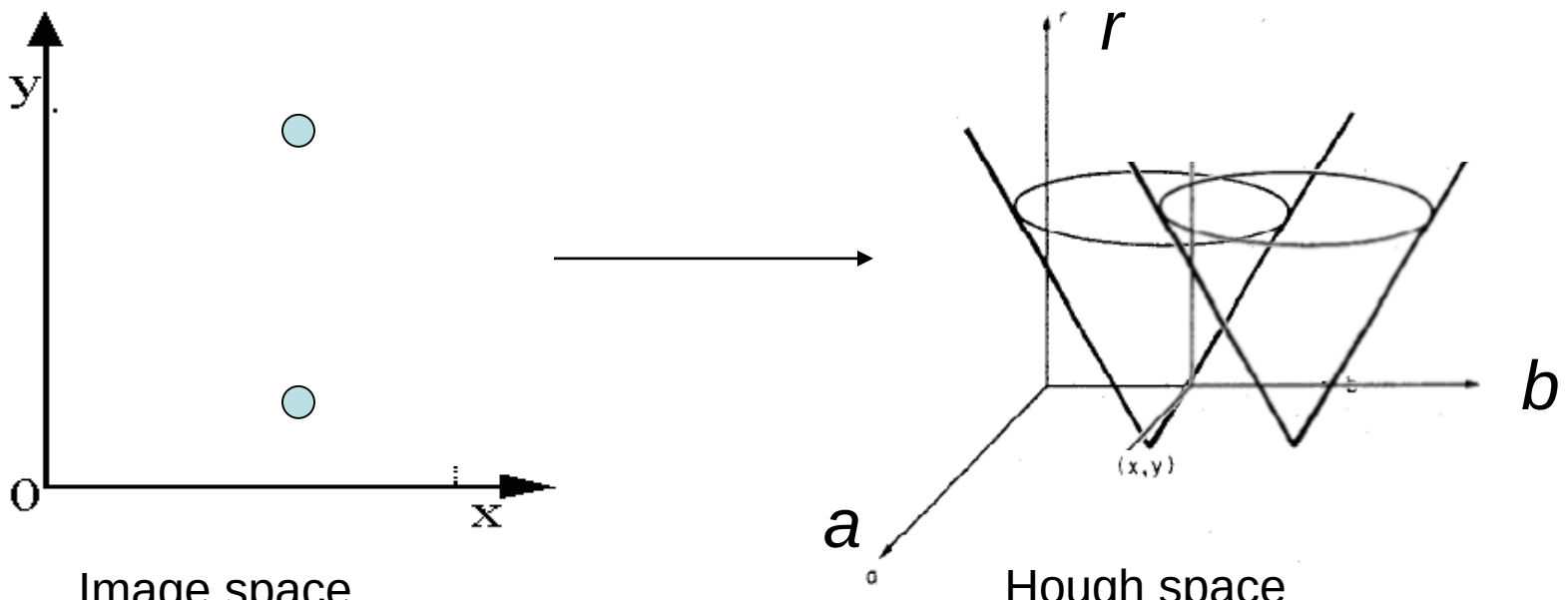


Image space

Hough space

Hough transform for circles

- Circle: center (a,b) and radius r

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

- For an unknown radius r, **known** gradient direction

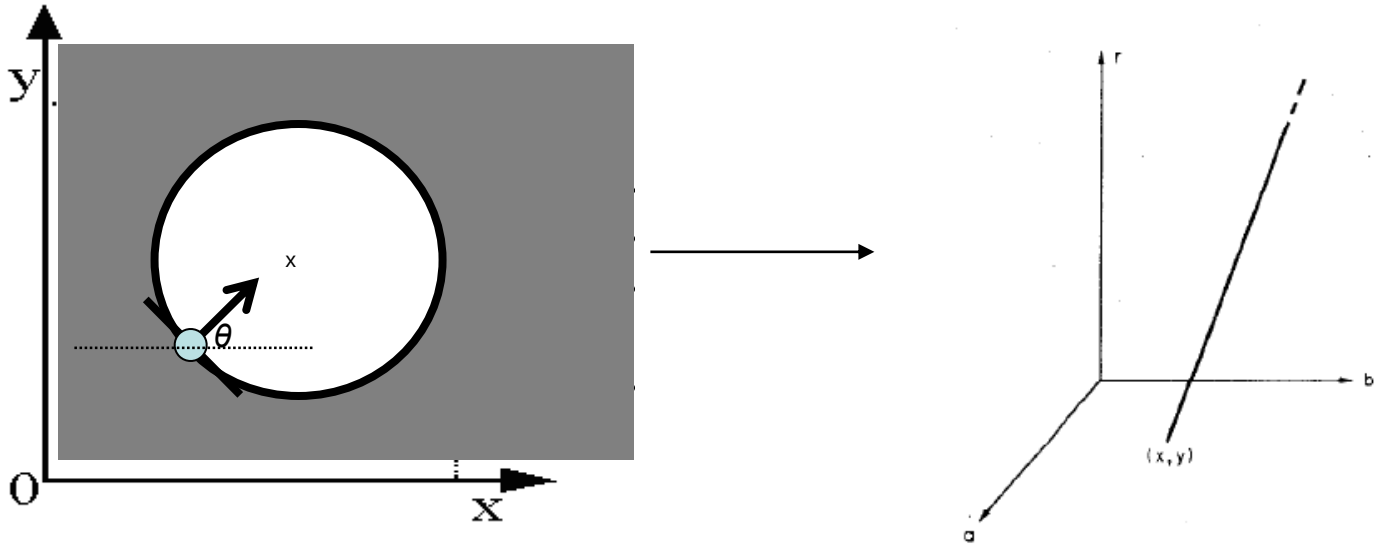


Image space

Hough space

Hough transform for circles

For every edge pixel (x,y) :

  For each possible radius value r:

    For each possible gradient direction θ:

      // or use estimated gradient at (x,y)

        $a = x + r \cos(\theta)$ // column

        $b = y - r \sin(\theta)$ // row

        $H[a,b,r] \mathrel{+}= 1$

  end

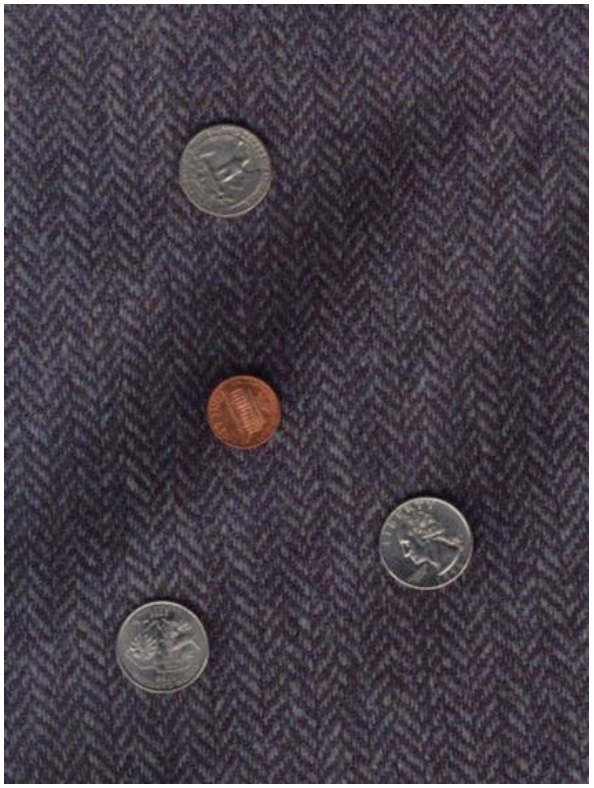
end

Time complexity per edge pixel?

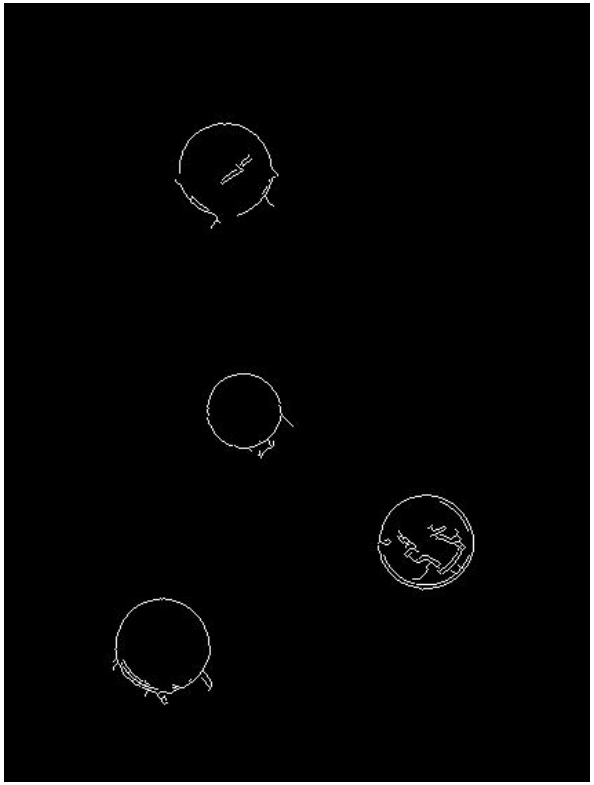
- Check out online demo : http://www.markschulze.net/java/hough/

Example: detecting circles with Hough

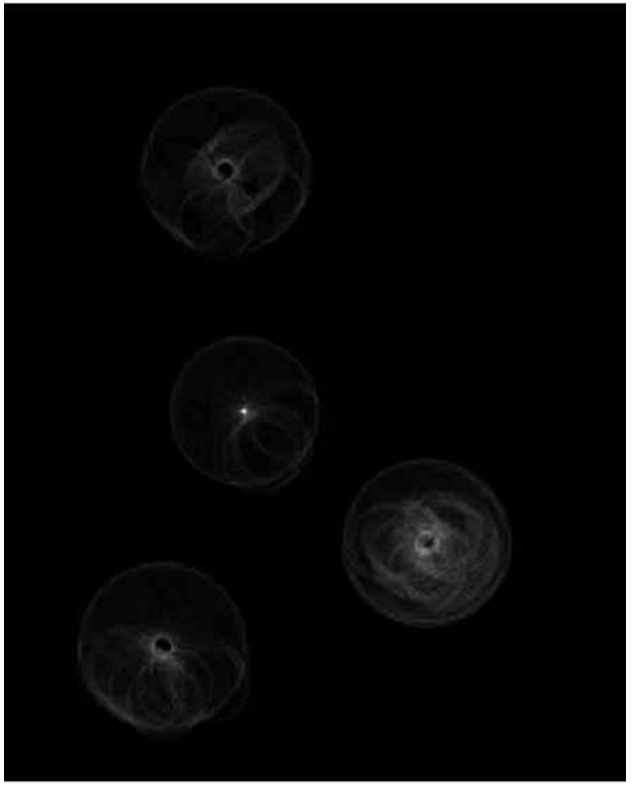
Original



Edges



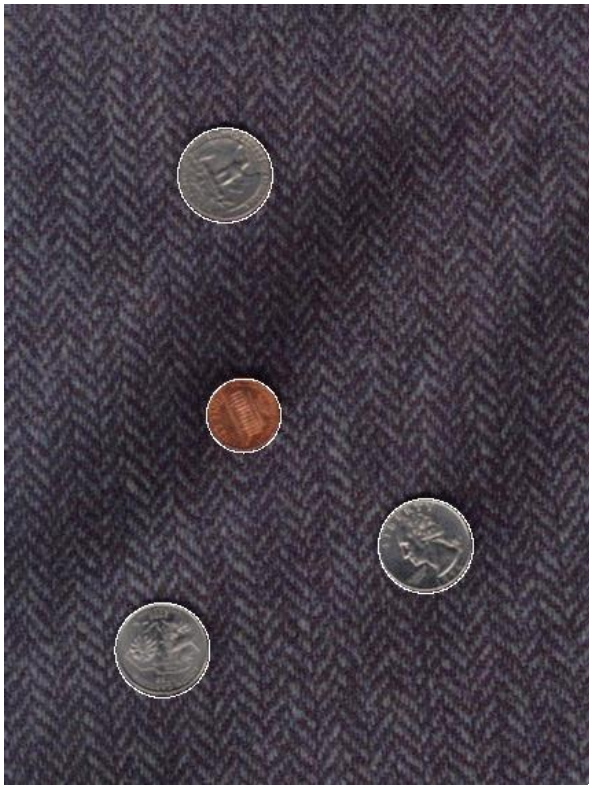
Votes: Penny



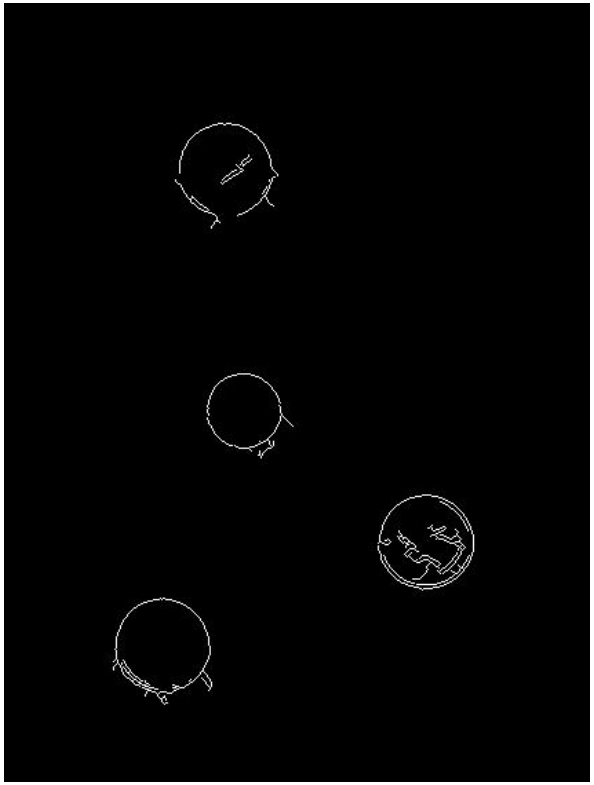
Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

Example: detecting circles with Hough

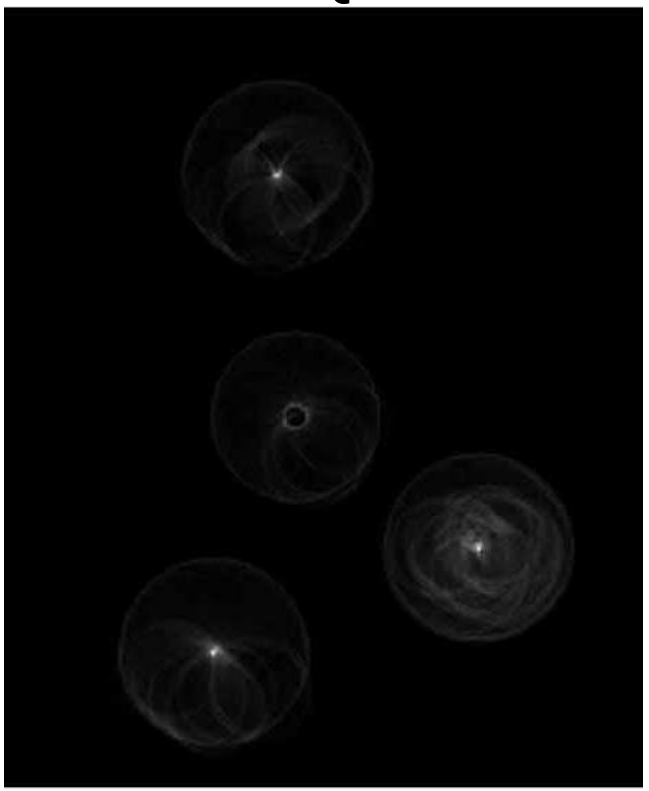
Original



Edges



Votes: Quarter

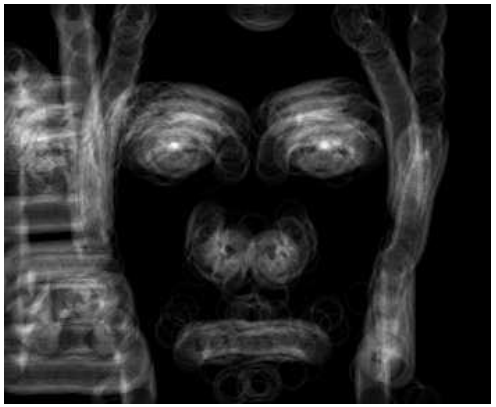


Combined detections

Coin finding sample images from: Vivek Kwatra

Example: iris detection

Gradient+threshold



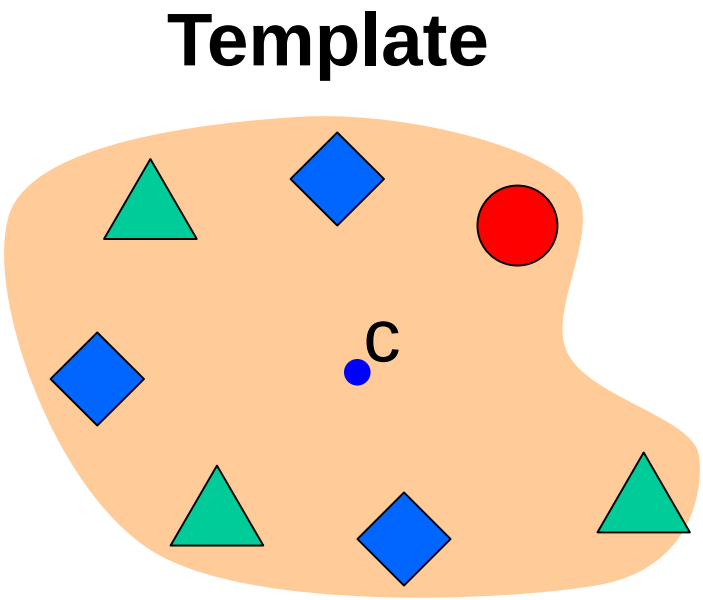
Hough space
(fixed radius)

Max detections

- Hemerson Pistori and Eduardo Rocha Costa
  http://rsbweb.nih.gov/ij/plugins/hough-circles.html

Generalized Hough transform

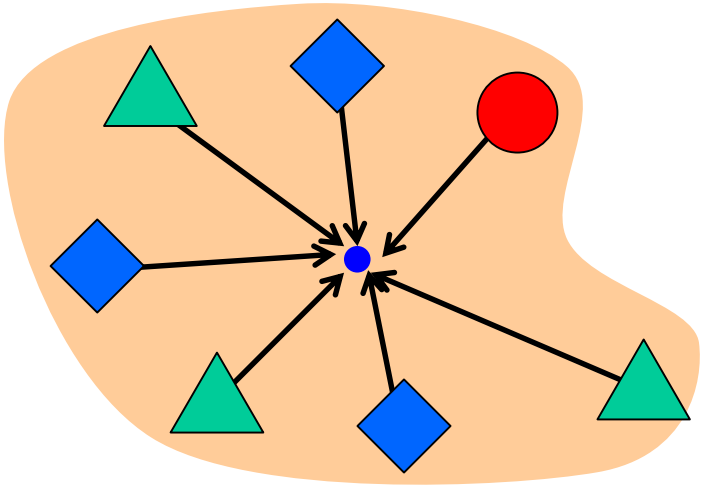
- We want to find a template defined by its reference point (center) and several distinct types of landmark points in stable spatial configuration



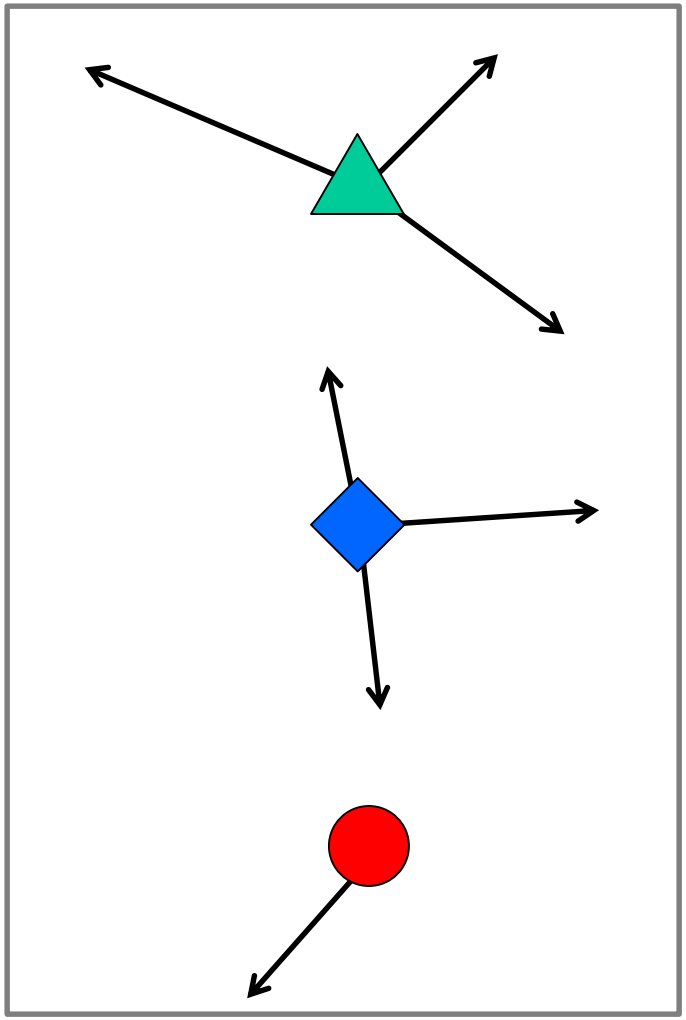
Generalized Hough transform

- Template representation: for each type of landmark point, store all possible displacement vectors towards the center

Template



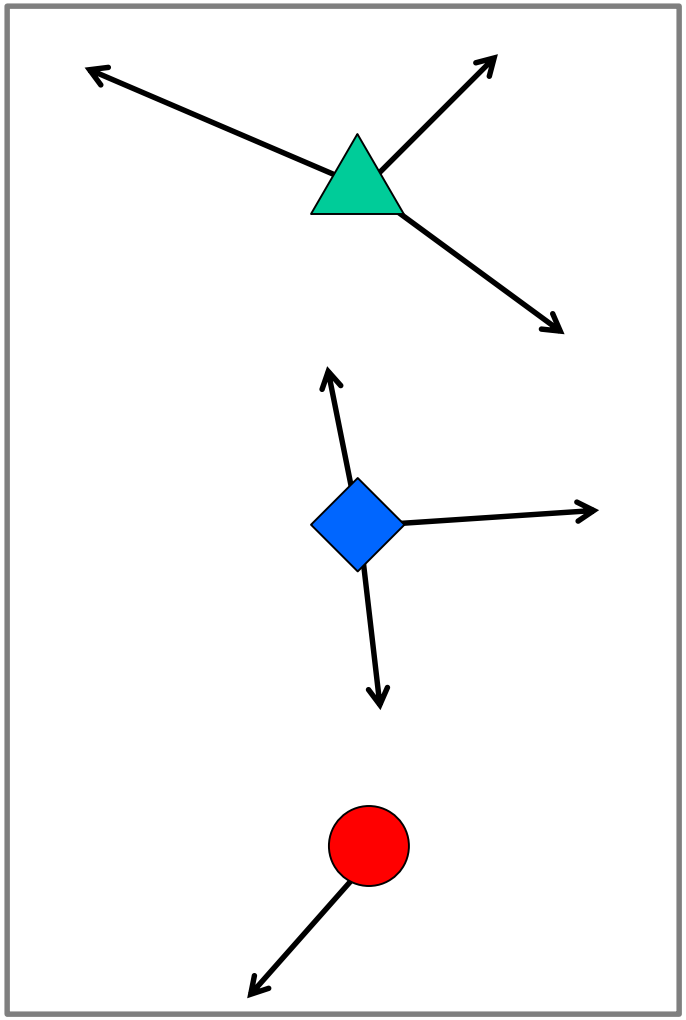
Model



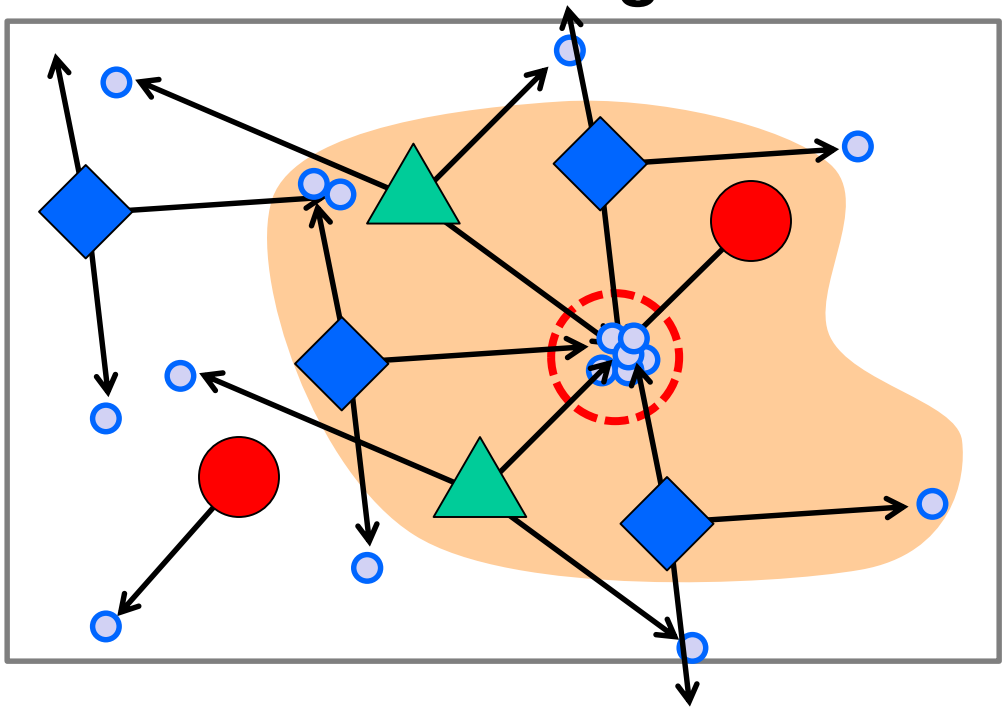
Generalized Hough transform

- Detecting the template:
  - For each feature in a new image, look up that feature type in the model and vote for the possible center locations associated with that type in the model

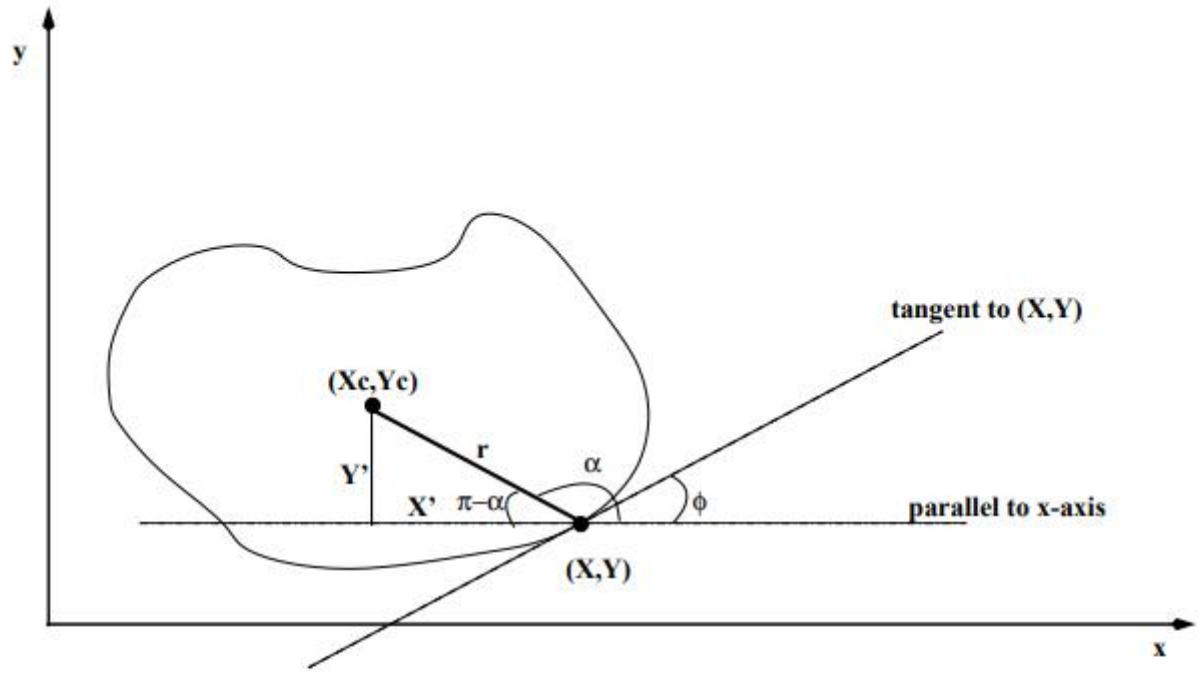
Model



Test image



Generalized Hough transform



$$x = x_c + x' \quad \text{or} \quad x_c = x - x'$$
$$y = y_c + y' \quad \text{or} \quad y_c = y - y'$$

$$cos(\pi - \alpha) = \frac{y'}{r} \quad \text{or} \quad y' = rcos(\pi - \alpha) = -rsin(\alpha)$$
$$sin(\pi - \alpha) = \frac{x'}{r} \quad \text{or} \quad x' = rsin(\pi - \alpha) = -rcos(\alpha)$$

Generalized Hough transform

- Combining the above equations we have:

$$x_c = x + r\cos(\alpha)$$
$$y_c = y + r\sin(\alpha)$$

Preprocessing step

(1) Pick a reference point (e.g., (x_c, y_c))

(2) Draw a line from the reference point to the boundary.

(3) Compute ϕ (i.e., perpendicular to gradient's direction).

(4) Store the reference point (x_c, y_c) as a function of ϕ (i.e., build the *R-table*)

ϕ_1:	(r_1^1, α_1^1), (r_2^1, α_2^1), ...
ϕ_2:	(r_1^2, α_1^2), (r_2^2, α_2^2), ...
ϕ_n:	(r_1^n, α_1^n), (r_2^n, α_2^n), ...

- The *R-table* allows us to use the contour edge points and gradient angle to recompute the location of the reference point.

Note: we need to build a separate *R-table* for each different object.

Generalized Hough transform

Detection

(1) Quantize the parameter space:

$$P[x_{c_{\min}} \cdots x_{c_{\max}}][y_{c_{\min}} \cdots y_{c_{\max}}]$$

(2) for each edge point (x, y)

(2.1) Using the gradient angle ϕ, retrieve from the *R-table* all the (α,r) values indexed under ϕ.

(2.2) For each (α,r), compute the candidate reference points:

$$x_c = x + r\cos(\alpha)$$
$$y_c = y + r\sin(\alpha)$$

(2.3) Increase counters (voting):

$$++(P[x_c][y_c])$$

(3) Possible locations of the object contour are given by local maxima in $P[x_c][y_c]$

- If $P[x_c][y_c] > T$, then the object contour is located at $x_c, y_c)$

Concluding remarks

- Advantages:
  - Conceptually simple.
  - Easy implementation
  - Handles missing and occluded data very gracefully.
  - Can be adapted to many types of forms, not just lines
- Disadvantages:
  - Computationally complex for objects with many parameters.
  - Looks for only one single type of object
  - The length and the position of a line segment cannot be determined.
  - Co-linear line segments cannot be separated.

RANSAC [Fischler & Bolles 1981]

- RANdom SAmple Consensus
- Approach: we want to avoid the impact of outliers, so let's look for "inliers", and use only those.
- Intuition: if an outlier is chosen to compute the current fit, then the resulting line won't have much support from rest of the points.

Slide credit: Kristen Grauman

RANSAC [Fischler & Bolles 1981]

- The RANSAC algorithm is used for estimating the parameters of models in images (i.e., model fitting). The basic idea behind RANSAC is to solve the fitting problem many times using randomly selected minimal subsets of the data and choosing the best performing fit.
- The RANSAC algorithm can be used to estimate parameters of different models; this is proven beneficial in image stitching, outlier detection, lane detection (linear model estimation), and stereo camera calculations.

Slide credit: Kristen Grauman

RANSAC [Fischler & Bolles 1981]

RANSAC loop:

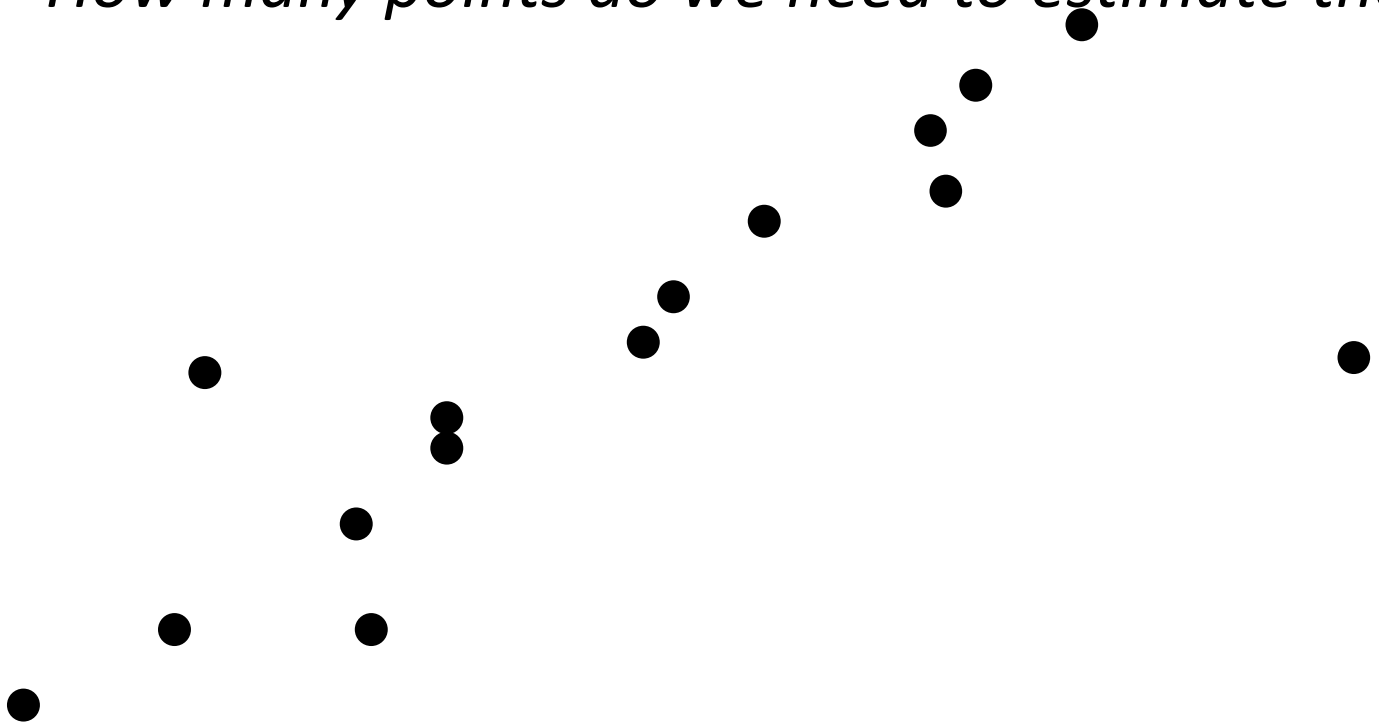
1. Randomly select a *seed group* of points on which to base transformation estimate
2. Compute transformation from seed group
3. Find *inliers* to this transformation
4. If the number of inliers is sufficiently large, re-compute least-squares estimate of transformation on all of the inliers

- Keep the transformation with the largest number of inliers

Slide credit: Kristen Grauman

RANSAC Line Fitting Example

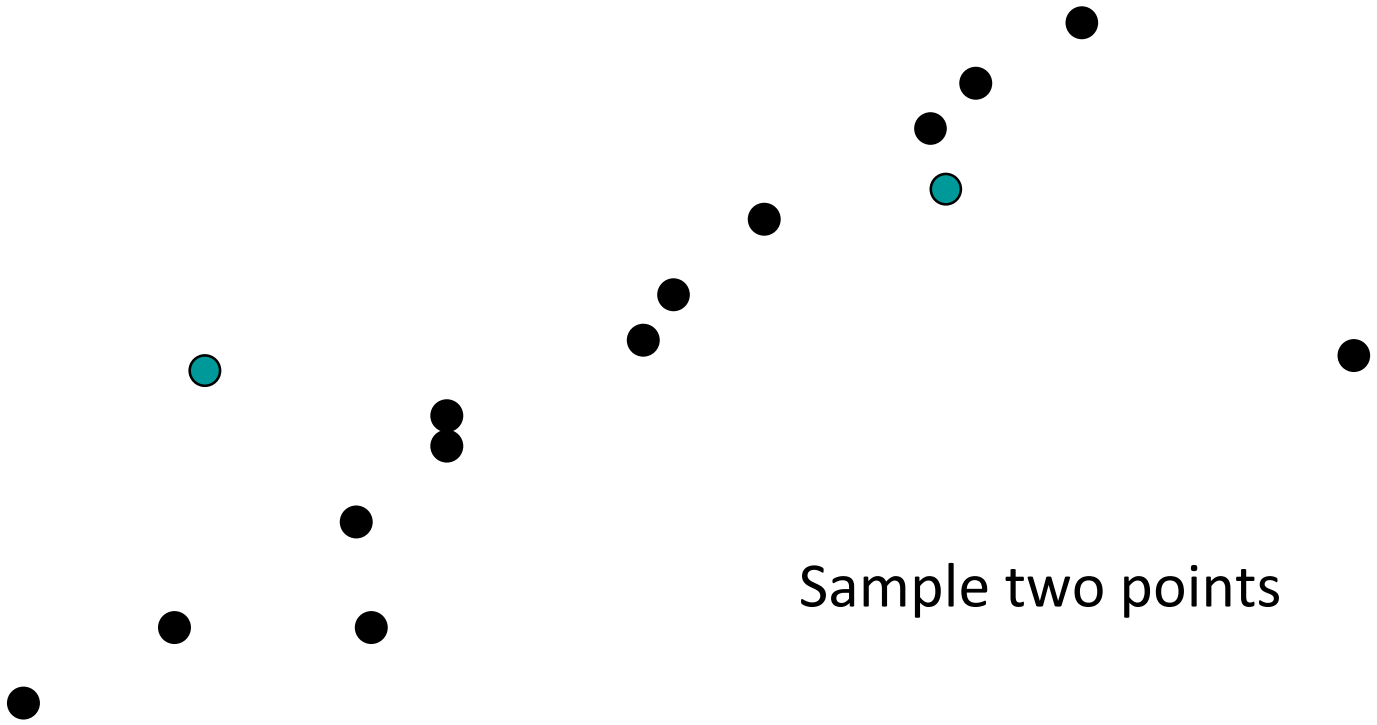
- Task: Estimate the best line
  - *How many points do we need to estimate the line?*



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

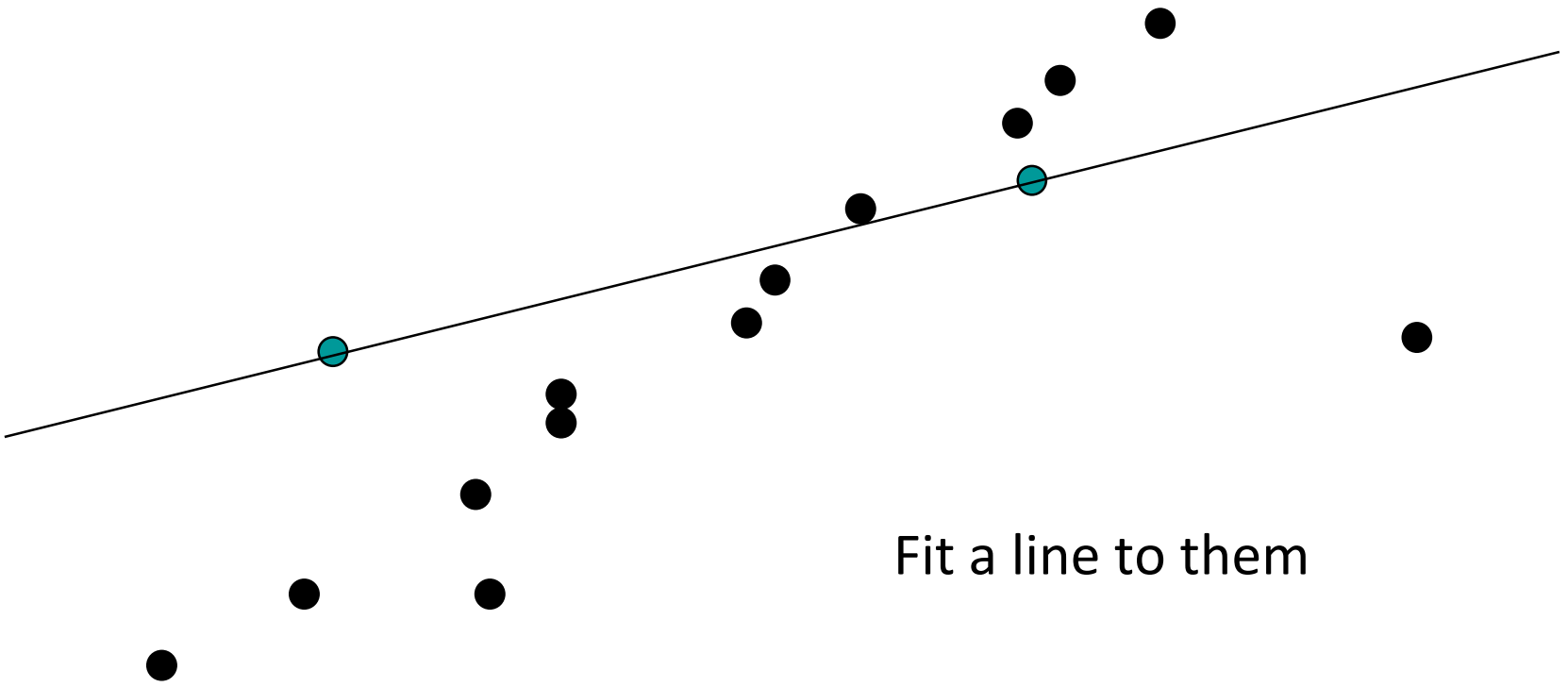
- Task: Estimate the best line



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

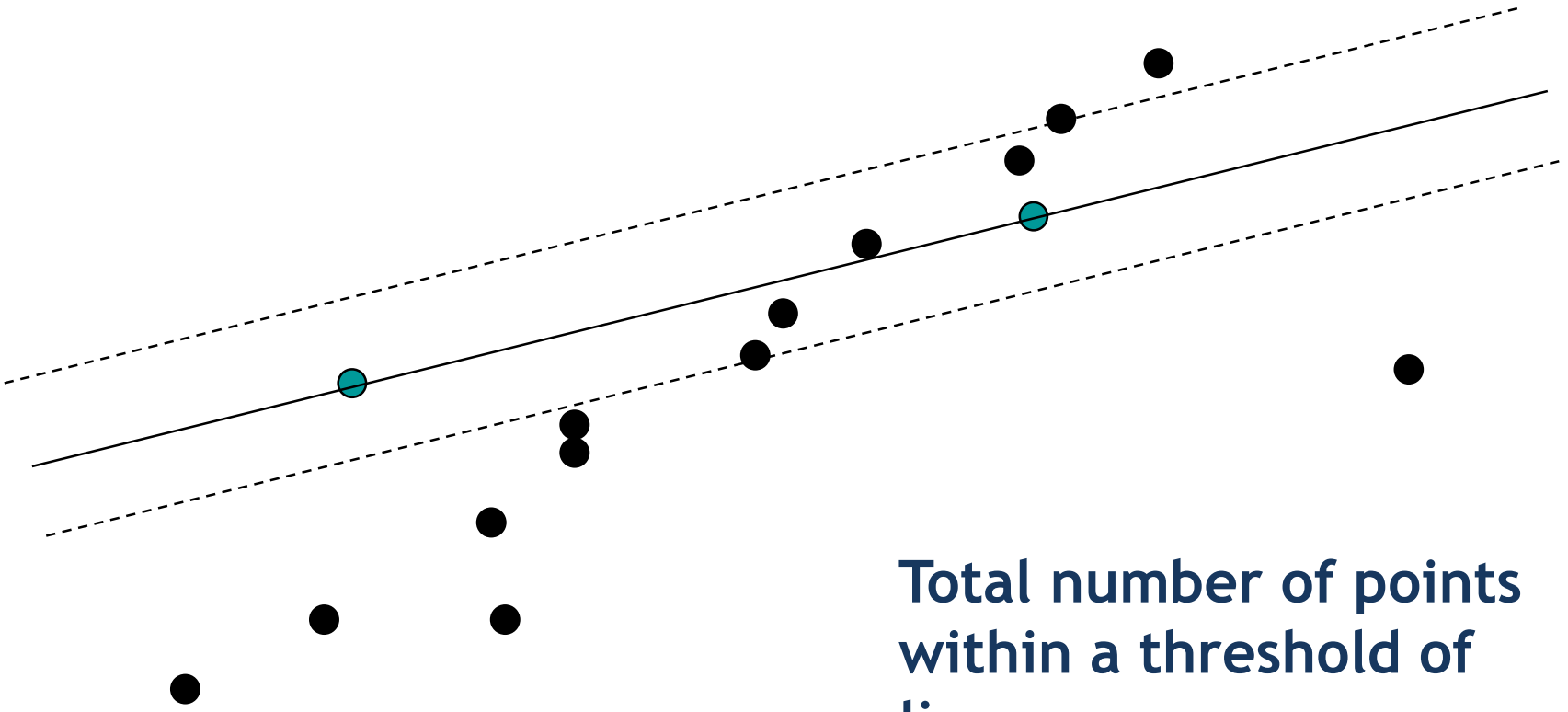
- Task: Estimate the best line



Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

- Task: Estimate the best line

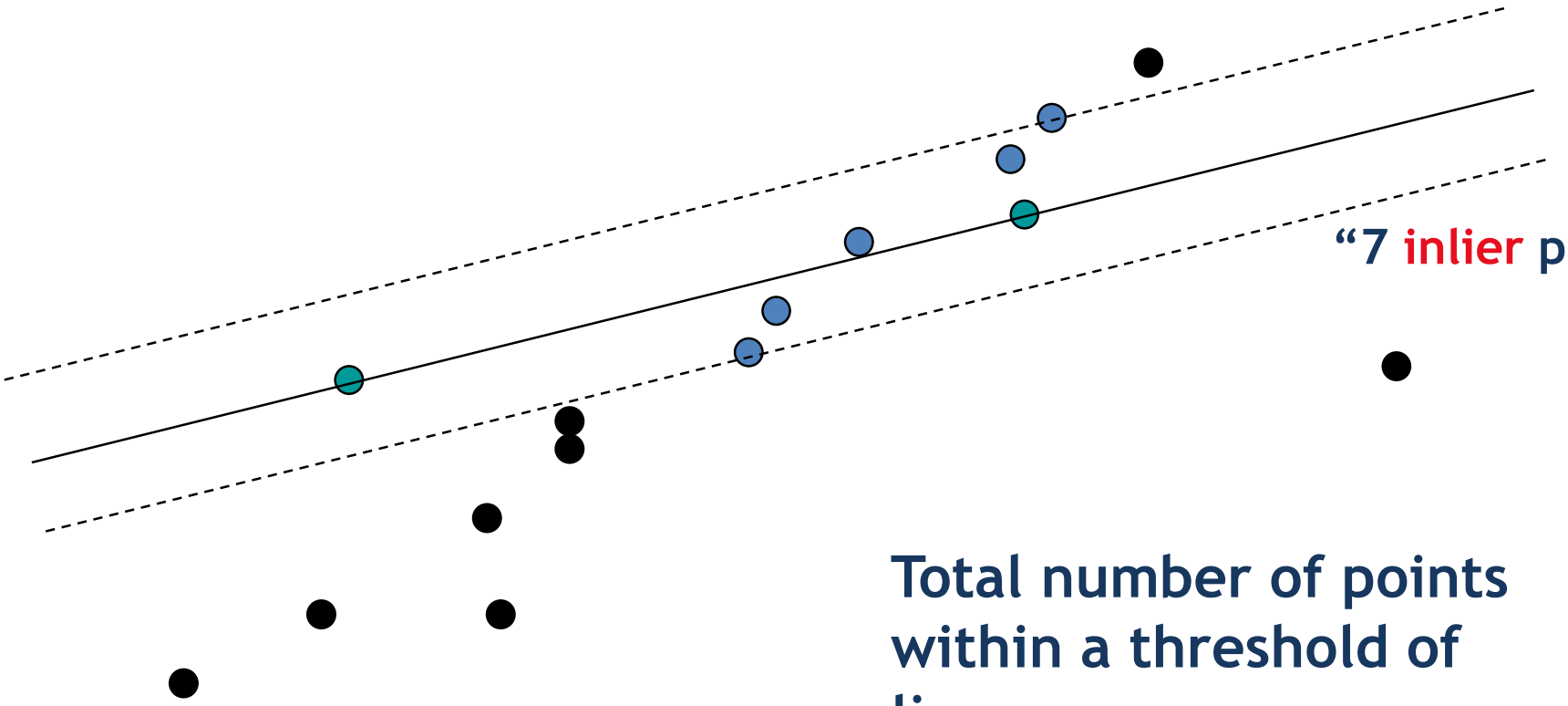


Total number of points within a threshold of line.

Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

- Task: Estimate the best line

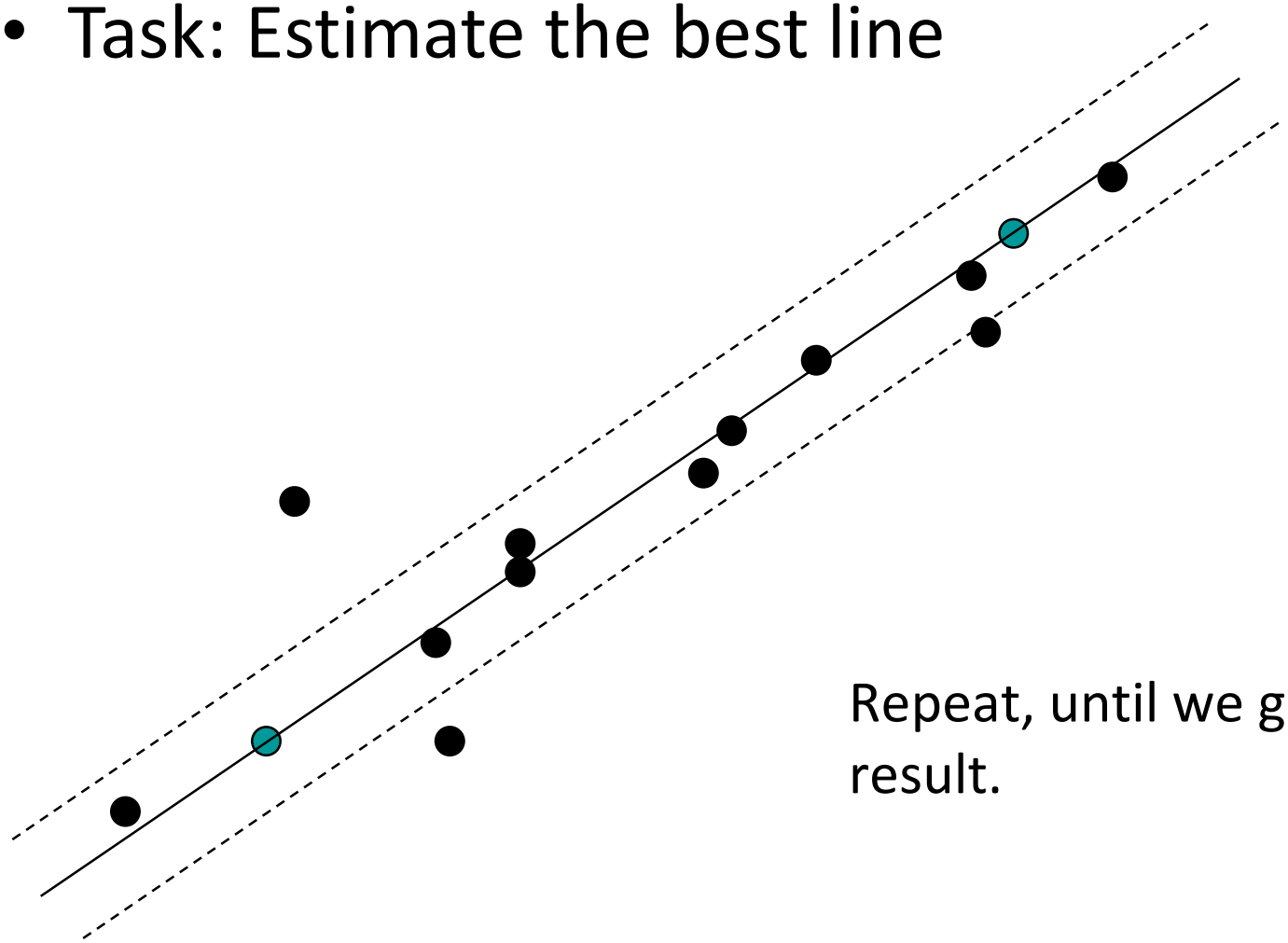


Total number of points within a threshold of line.

Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

- Task: Estimate the best line

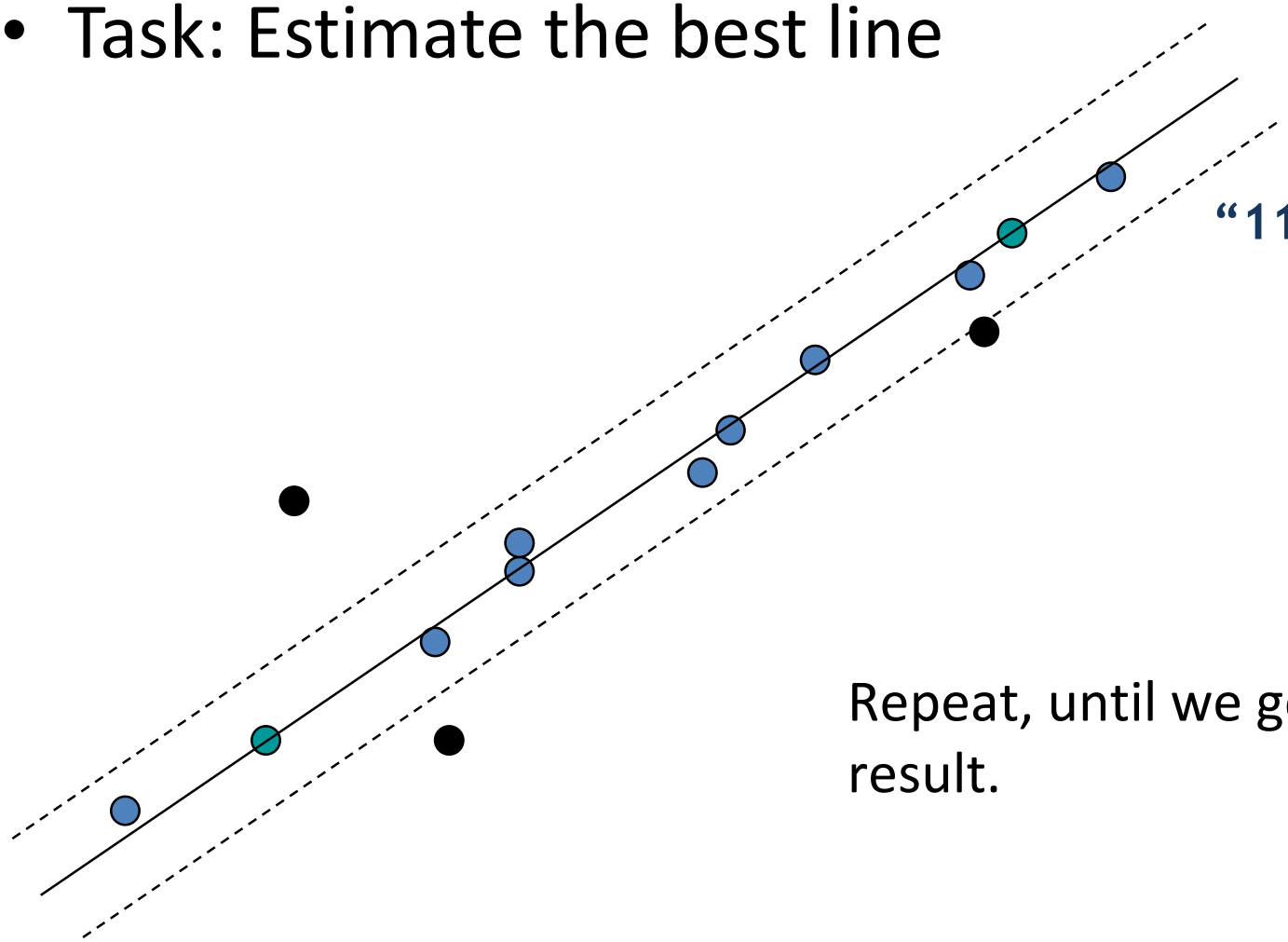


Repeat, until we get a good result.

Slide credit: Jinxiang Chai

RANSAC Line Fitting Example

- Task: Estimate the best line



"11 inlier points"

Repeat, until we get a good result.

Slide credit: Jinxiang Chai

Algorithm 15.4: RANSAC: fitting lines using random sample consensus

Determine:
    n — the smallest number of points required
    k — the number of iterations required
    t — the threshold used to identify a point that fits well
    d — the number of nearby points required
      to assert a model fits well
Until k iterations have occurred
    Draw a sample of n points from the data
      uniformly and at random
    Fit to that set of n points
    For each data point outside the sample
      Test the distance from the point to the line
        against t; if the distance from the point to the line
        is less than t, the point is close
    end
    If there are d or more points close to the line
      then there is a good fit. Refit the line using all
      these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion

RANSAC: How many samples?

- How many samples are needed?
  - Suppose w is fraction of inliers (points from line).
  - n points needed to define hypothesis (2 for lines)
  - k samples chosen.
- Prob. that a single sample of n points is correct: w^n
- Prob. that all k samples fail is: $(1-w^n)^k$

$\Rightarrow$ Choose k high enough to keep this below desired failure rate.

Slide credit: David Lowe

RANSAC: Computed k (p=0.99)

Sample size n	Proportion of outliers						
	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Slide credit: David Lowe

RANSAC: Pros and Cons

- **Pros**:
  - General method suited for a wide range of model fitting problems
  - Easy to implement and easy to calculate its failure rate
- **Cons**:
  - Only handles a moderate percentage of outliers without cost blowing up
  - Many real problems have high rate of outliers (but sometimes selective choice of random subsets can help)
- The Hough transform can handle high percentage of outliers