Lecture 5

Detecting Lines

Administrative

A1 is out

- It is a graded assignment
- Due **Oct 14**

So far: discrete derivatives in 3 ways

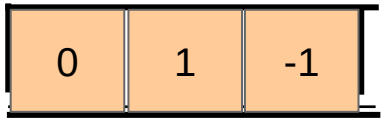
$$\frac{df}{dx} = f[x] - f[x-1]$$ Backward

$$= f[x+1] - f[x]$$ Forward

$$= \frac{1}{2}(f[x+1] - f[x-1])$$ Central but we can drop the 1/2

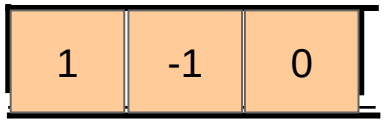
So far: Designing filters that perform differentiation

- Using Backward differentiation:



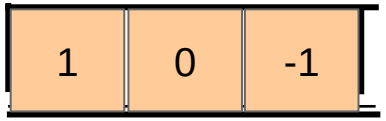
$$g[n,m] = f[n,m] - f[n,m-1]$$

- Using Forward differentiation:



$$g[n,m] = f[n,m+1] - f[n,m]$$

- Using Central differentiation:



$$g[n,m] = f[n,m+1] - f[n,m-1]$$

So far: Calculating gradient magnitude and direction

$$\text{Given function } f[n,m]$$

$$\text{Gradient filter } \nabla f[n,m] = \begin{bmatrix} \frac{df}{dn} \\ \frac{df}{dm} \end{bmatrix} = \begin{bmatrix} f_n \\ f_m \end{bmatrix}$$

$$\text{Gradient magnitude } |\nabla f[n,m]| = \sqrt{f_n^2 + f_m^2}$$

$$\text{Gradient direction } \theta = \tan^{-1}(\frac{f_m}{f_n})$$

Today’s agenda

- Edge detector with noisy images
- Sobel Edge detector
- Canny edge detector
- Hough Transform

Optional reading:
Szeliski, Computer Vision: Algorithms and Applications, 2nd Edition
Sections 7.1, 8.1.4

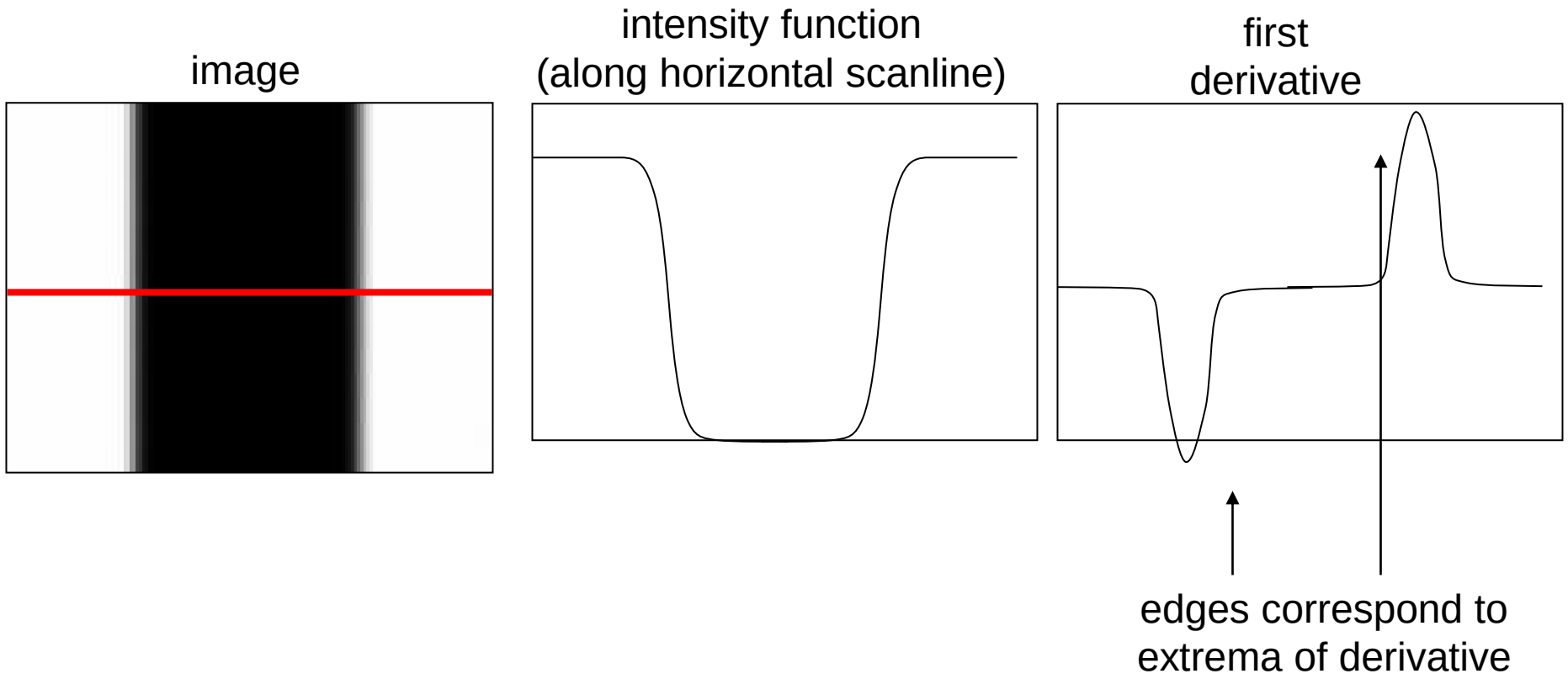
Today’s agenda

- Edge detector with noisy images
- Sobel Edge detector
- Canny edge detector
- Hough Transform

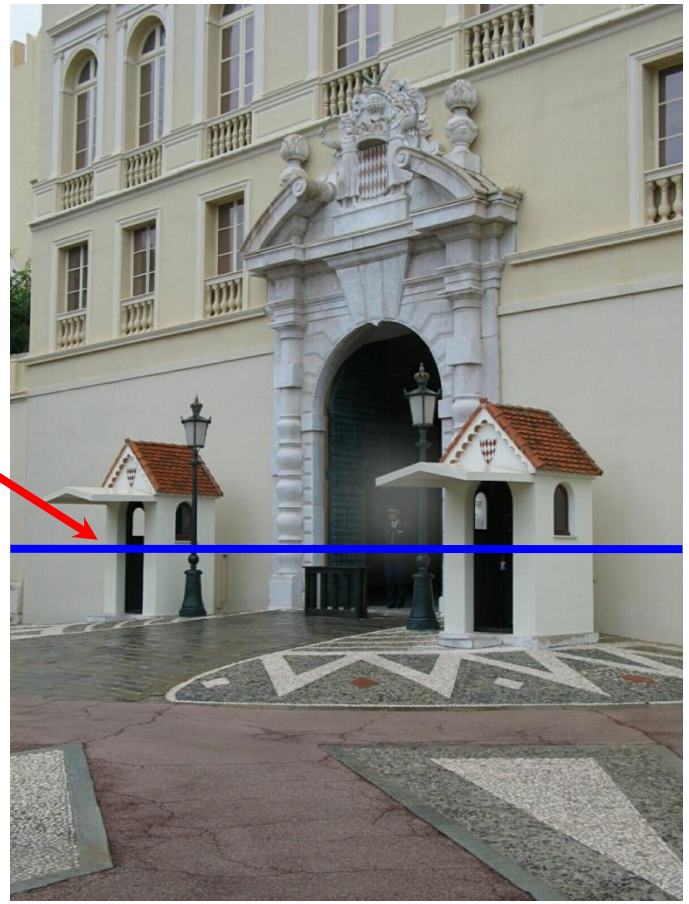
Optional reading:
Szeliski, Computer Vision: Algorithms and Applications, 2nd Edition
Sections 7.1, 8.1.4

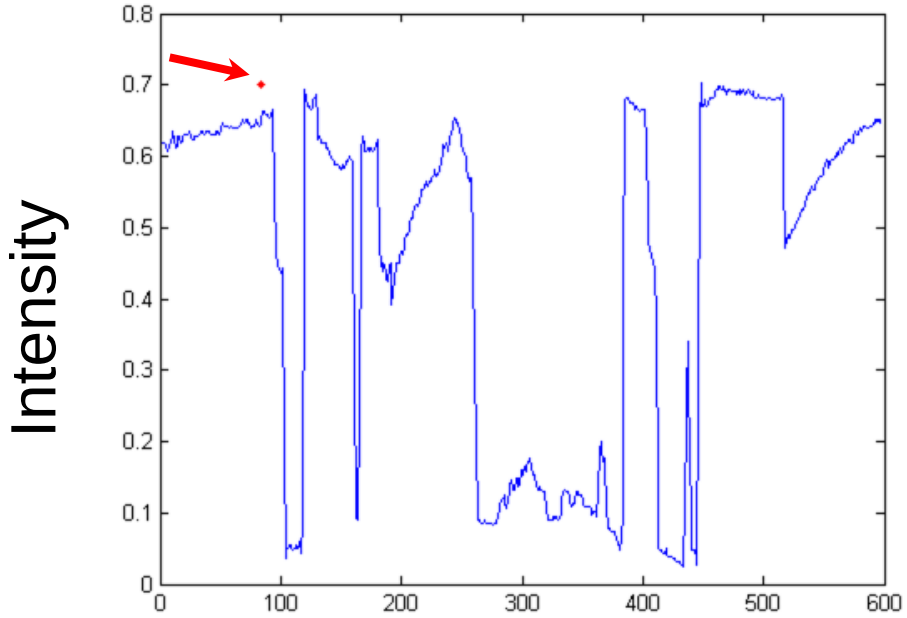
Characterizing edges

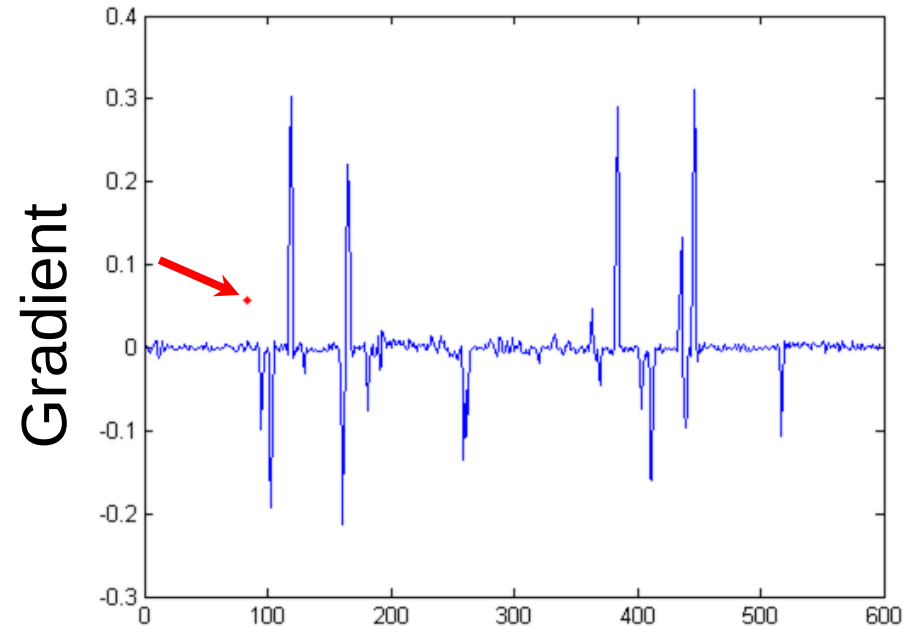
An edge is a place of rapid change in the image intensity function



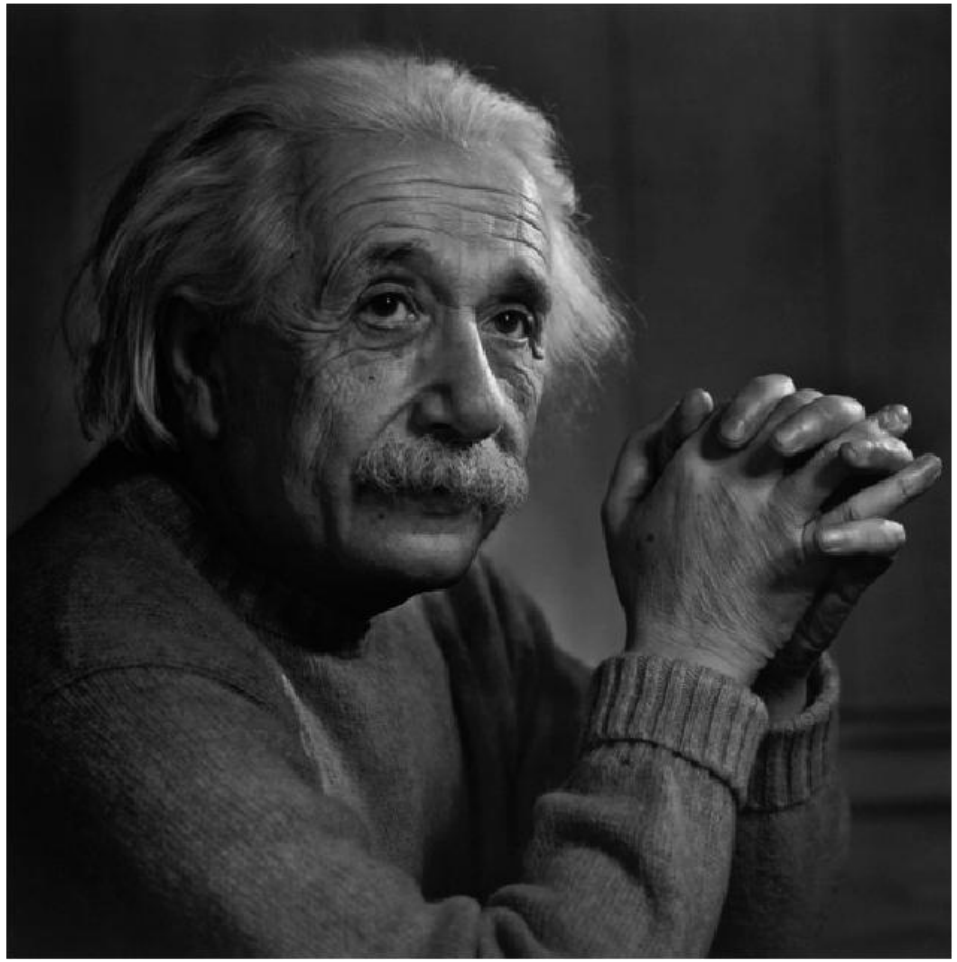
Intensity profile

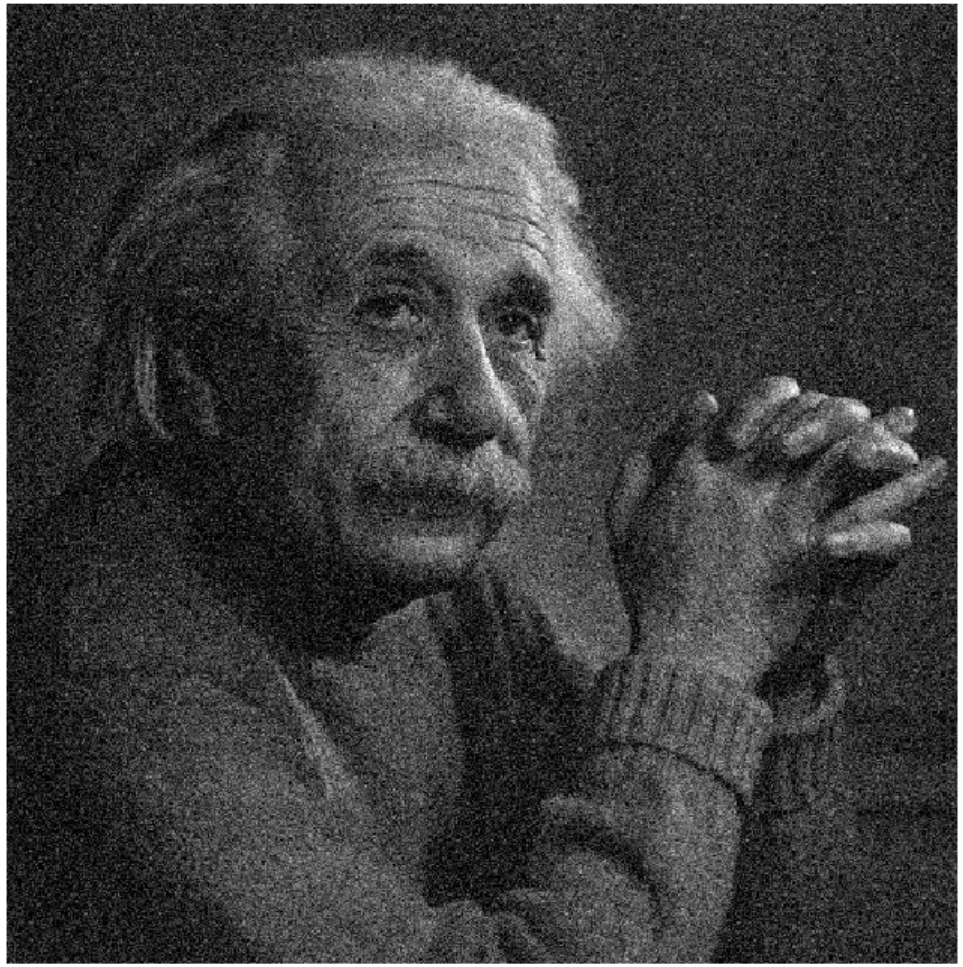






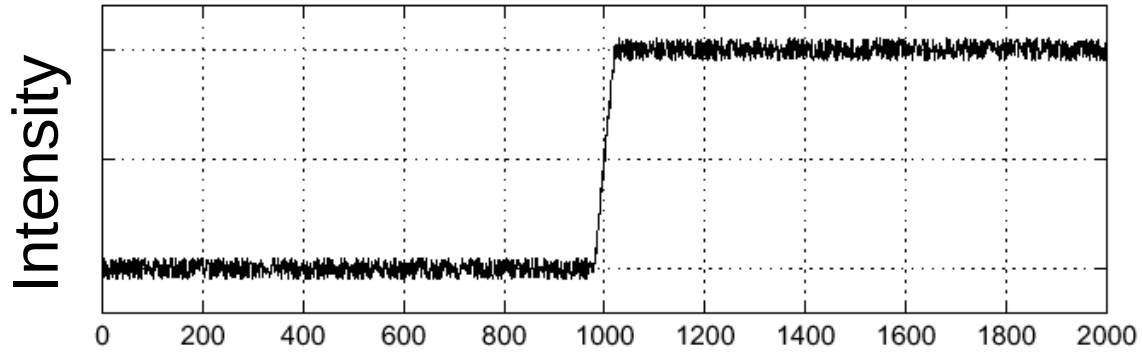
Q. What will happen if we use this edge detector on a noisy pixels?





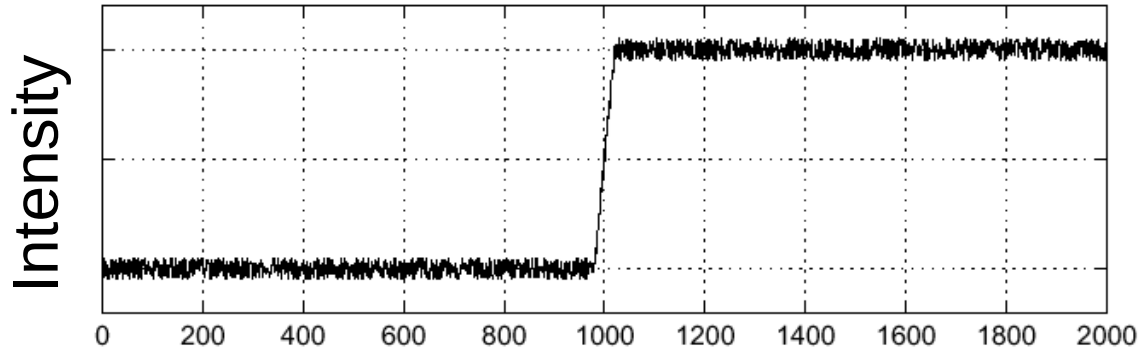
Effects of noise

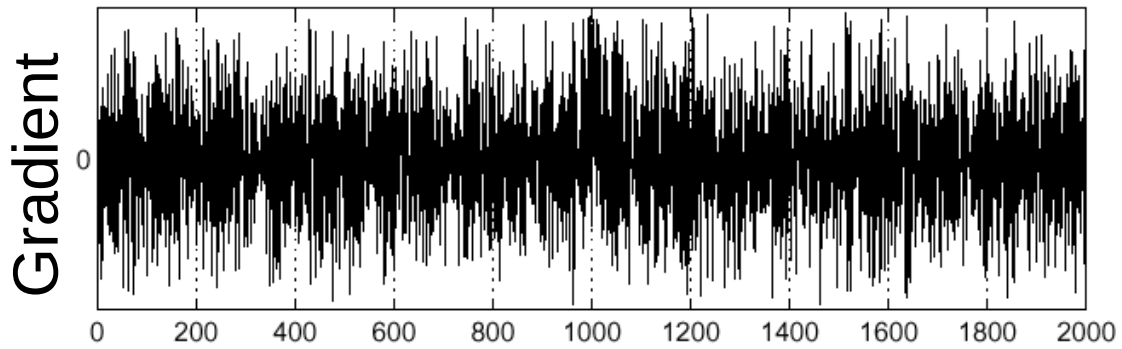
- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal



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?

Effects of noise

- Differentiation filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the larger the gradient
- Q. What is a potential quick fix for noisy images?

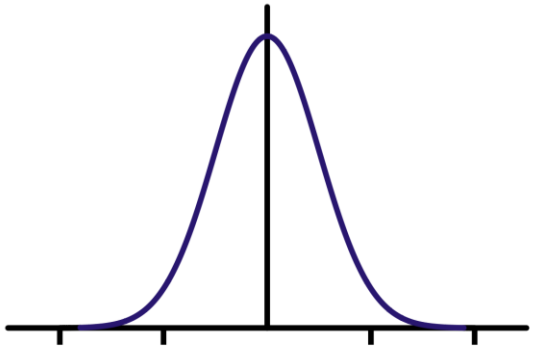
Effects of noise

- Differentiation 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
- Q. What is a potential quick fix for noisy images?
- Smoothing the image should help, by forcing pixels different to their neighbors (=noise pixels?) to look more like neighbors

Smoothing with different filters

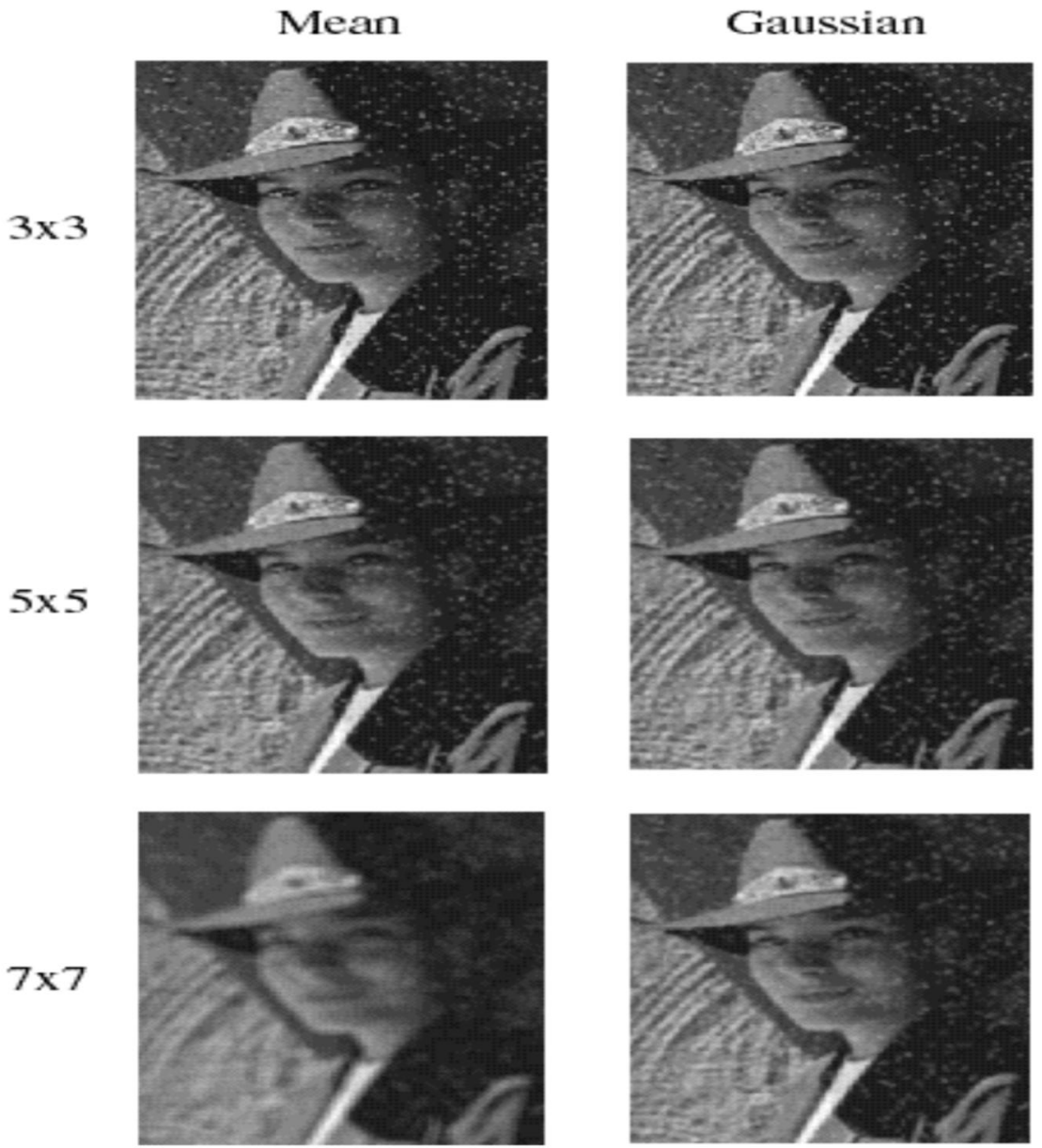
- Mean smoothing $\frac{1}{3}\begin{bmatrix}1\\1\\1\end{bmatrix}$ $\frac{1}{3}\begin{bmatrix}1 & 1 & 1\end{bmatrix}$

- Gaussian (smoothing * derivative)



$\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix}$ $\frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix}$

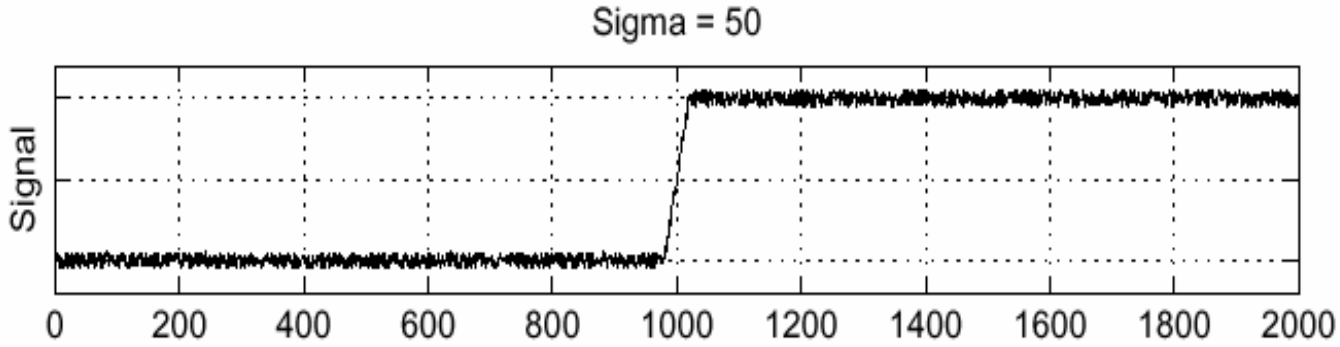
Smoothing with different filters



Solution: input function

Let’s look at a single image row:

f

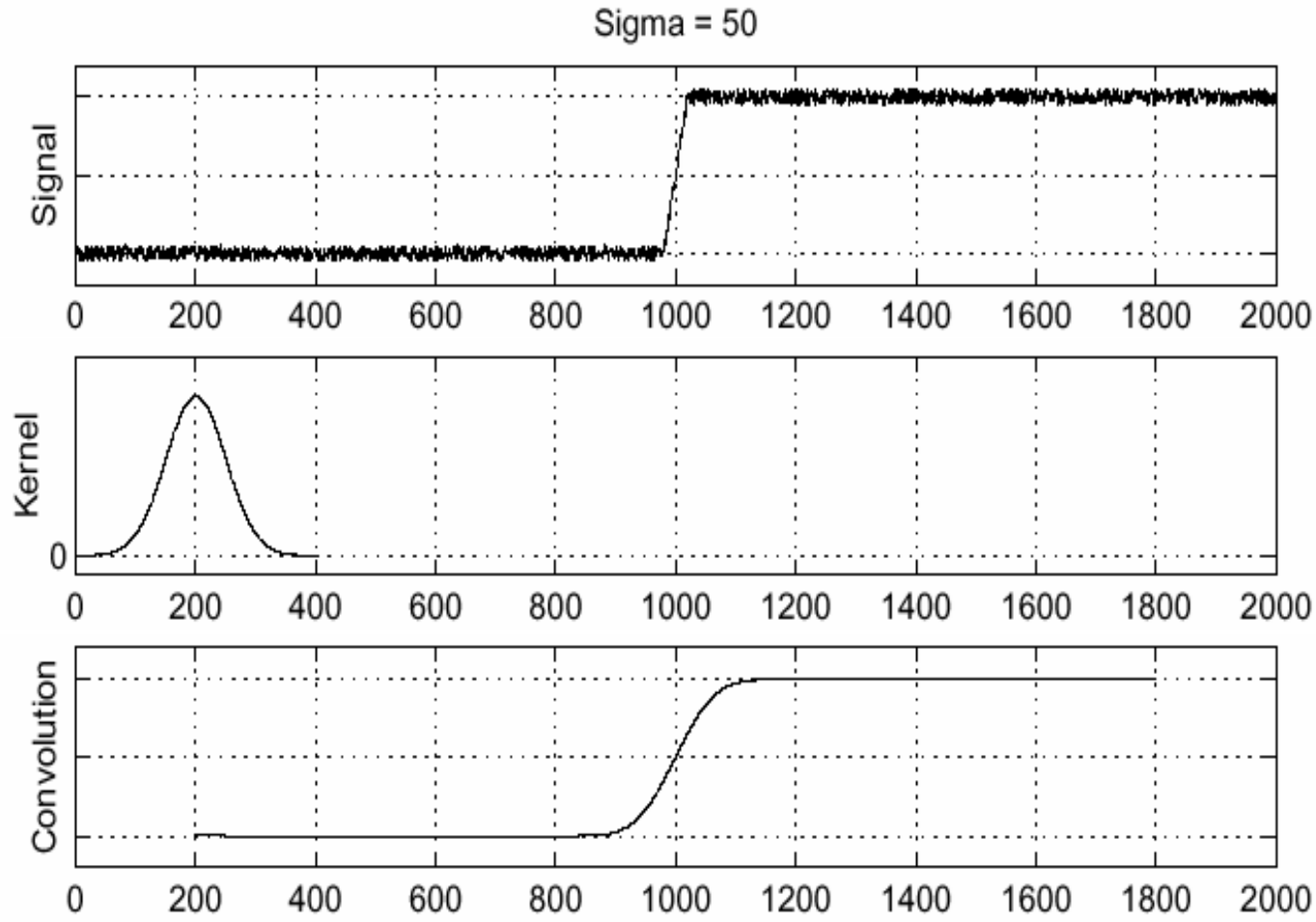


Solution: smooth first

f

h

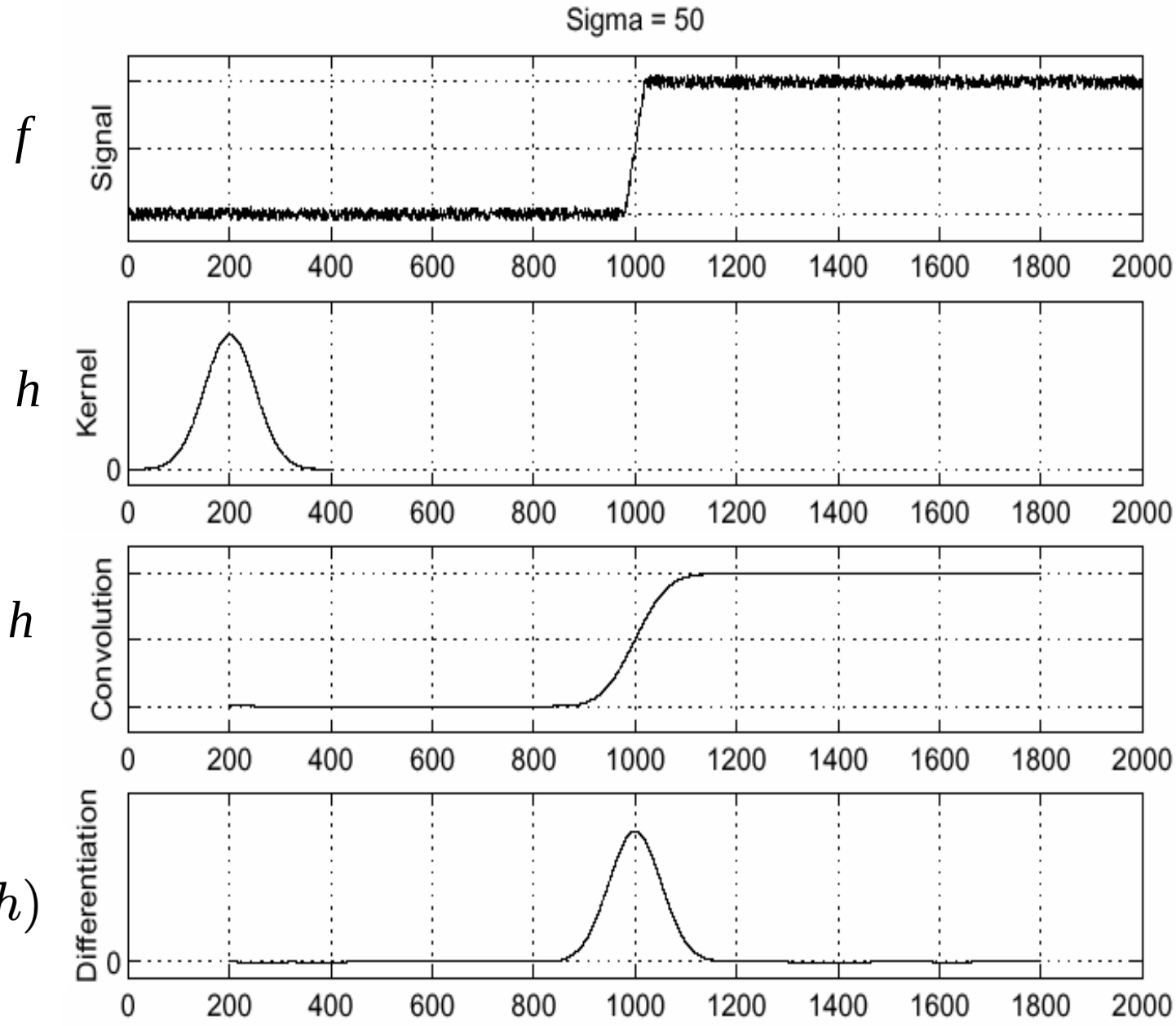
$f * h$



Solution: smooth first

To find edges, look for peaks in

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

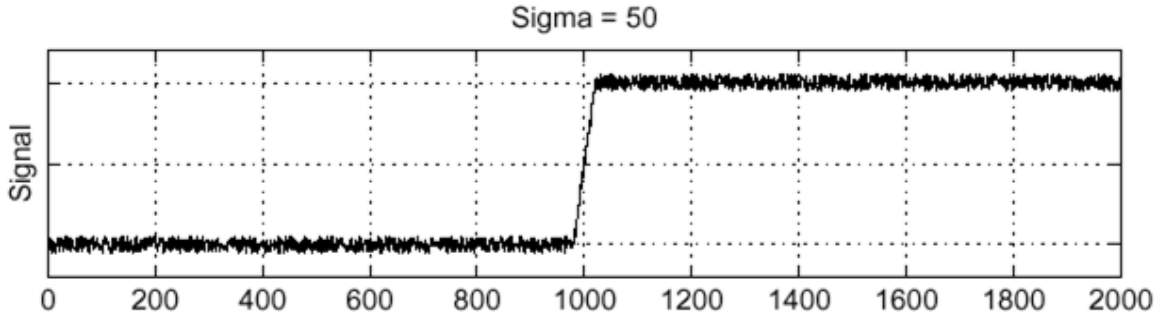


Derivative theorem of convolution

- This theorem gives us a very useful property:

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

f



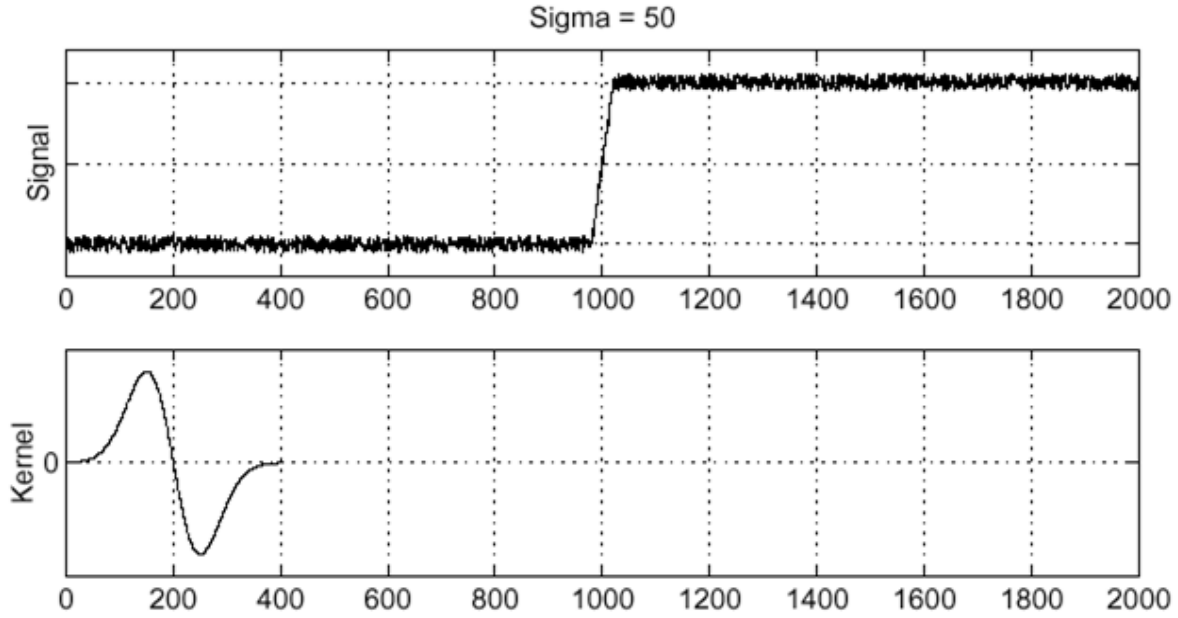
Derivative of a gaussian (DoG)

- This theorem gives us a very useful property:

$$\frac{d}{dx}(f * h) = f * \left(\frac{d}{dx} h\right)$$

f

$$\frac{d}{dx} h$$



Derivative of a gaussian (DoG)

- This theorem gives us a very useful property:

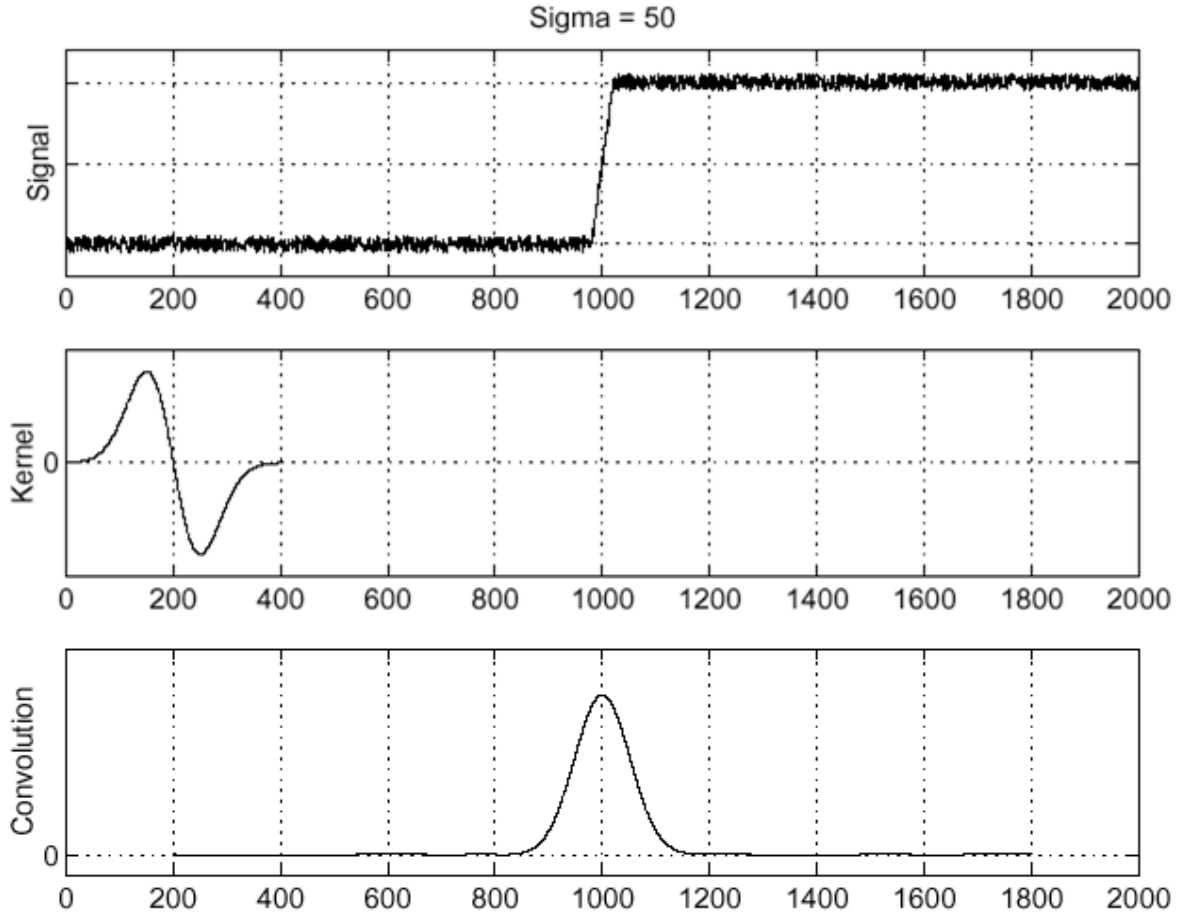
$$\frac{d}{dx}(f * h) = f * (\frac{d}{dx}h)$$

- This saves us one operation:

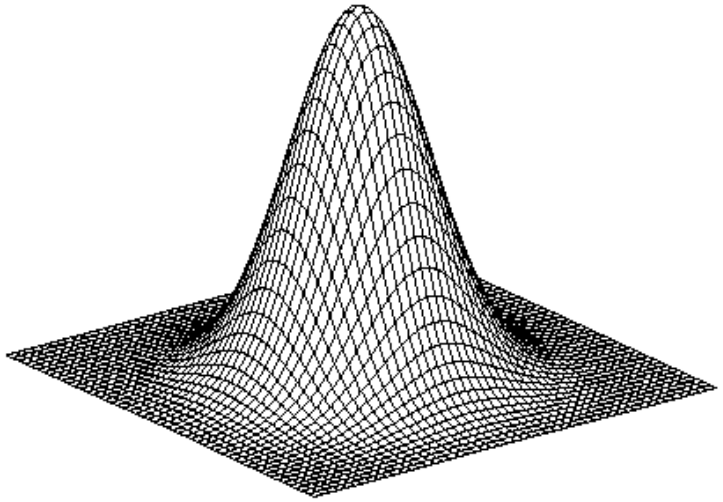
We can precompute: $\frac{d}{dx}h$

f

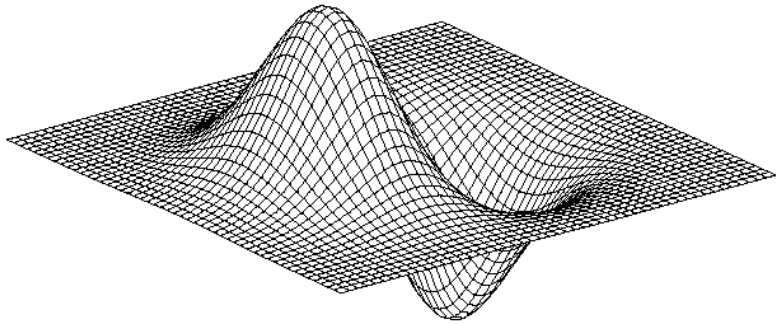
$f * (\frac{d}{dx}h)$



Derivative of Gaussian filter (central derivative)



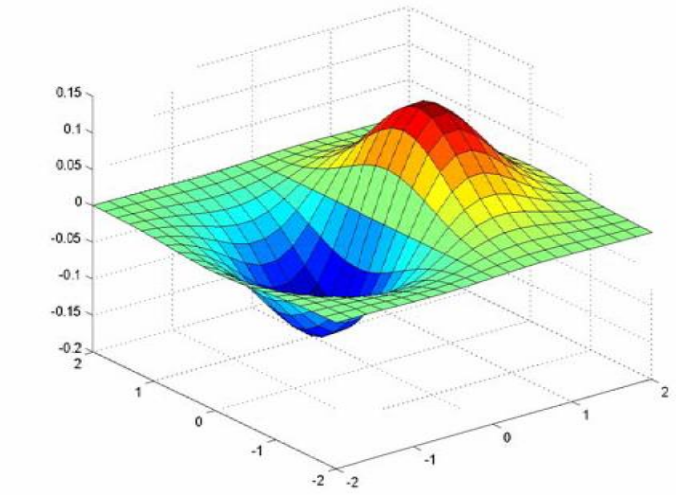
* [1 0 -1] =



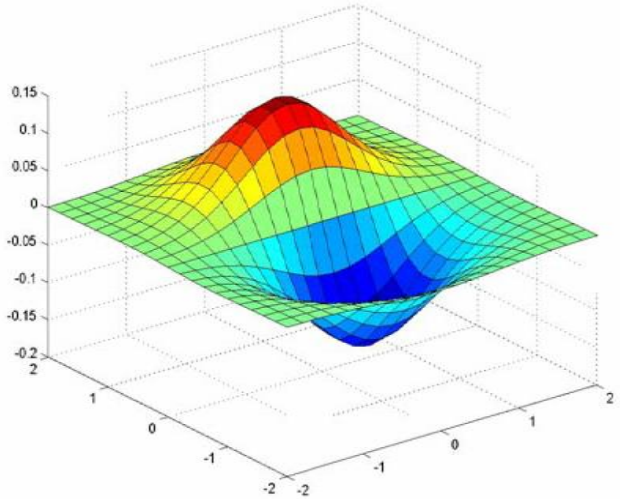
2D-gaussian

x - derivative

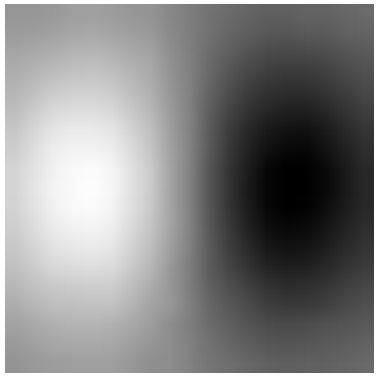
Derivative of Gaussian filter along x and y directions

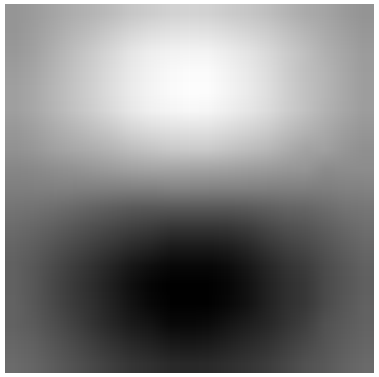


x-direction



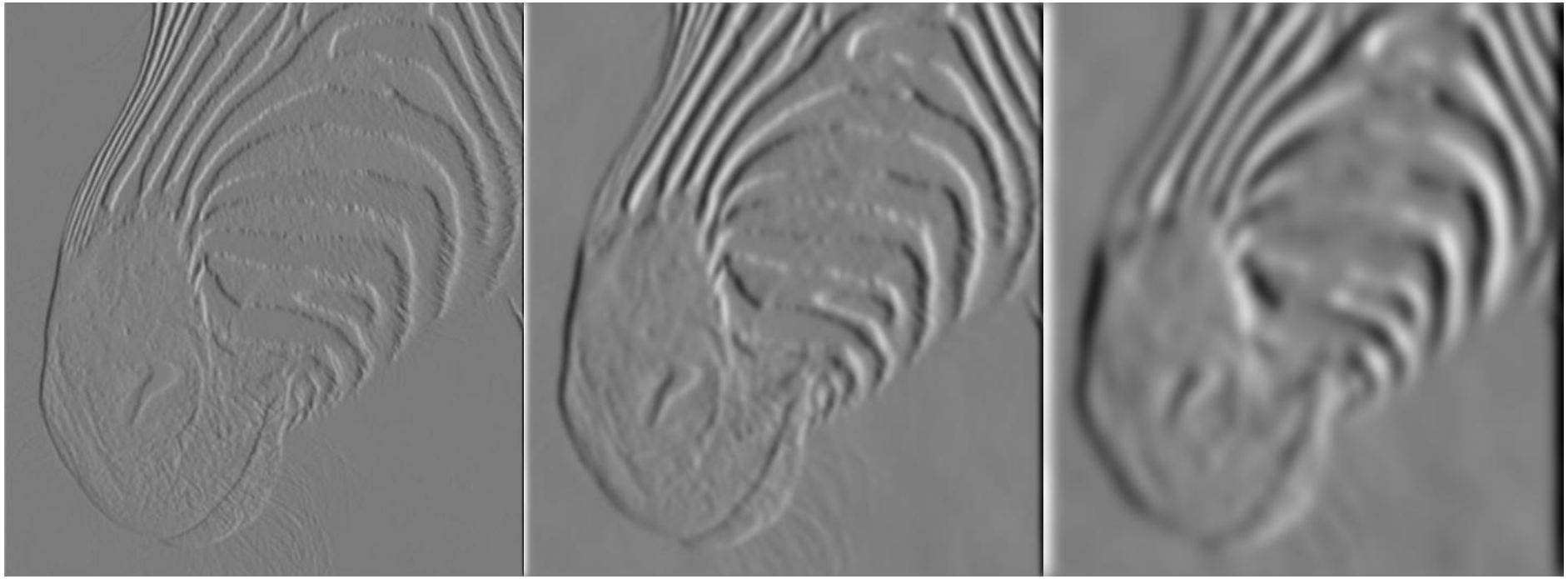
y-direction





Derivative of Gaussian filter

Tradeoff between smoothing at different scales

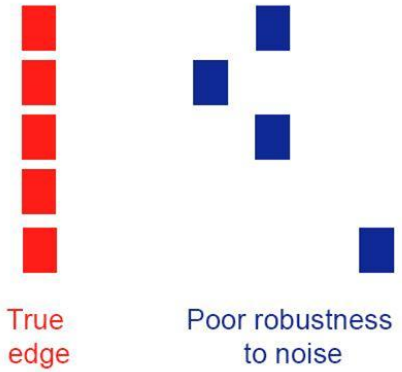


1 pixel 3 pixels 7 pixels

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

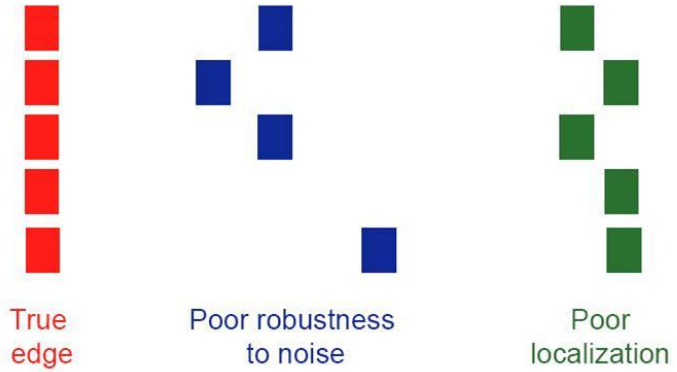
Designing an edge detector

- 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)



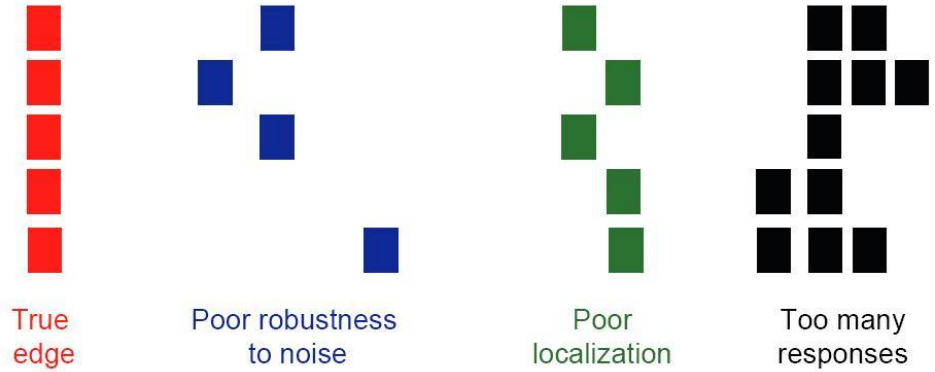
Designing an edge detector

- 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



Today’s agenda

- Edge detector with noisy images
- Sobel Edge detector
- Canny edge detector
- Hough Transform

Optional reading:
Szeliski, Computer Vision: Algorithms and Applications, 2nd Edition
Sections 7.1, 8.1.4

Sobel Operator

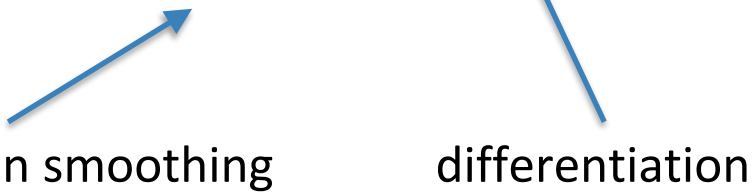
- uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives
- one for horizontal changes, and one for vertical

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

Sobel Operation

- Smoothing + differentiation

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



Sobel Operation

- Magnitude:

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

- Angle or direction of the gradient:

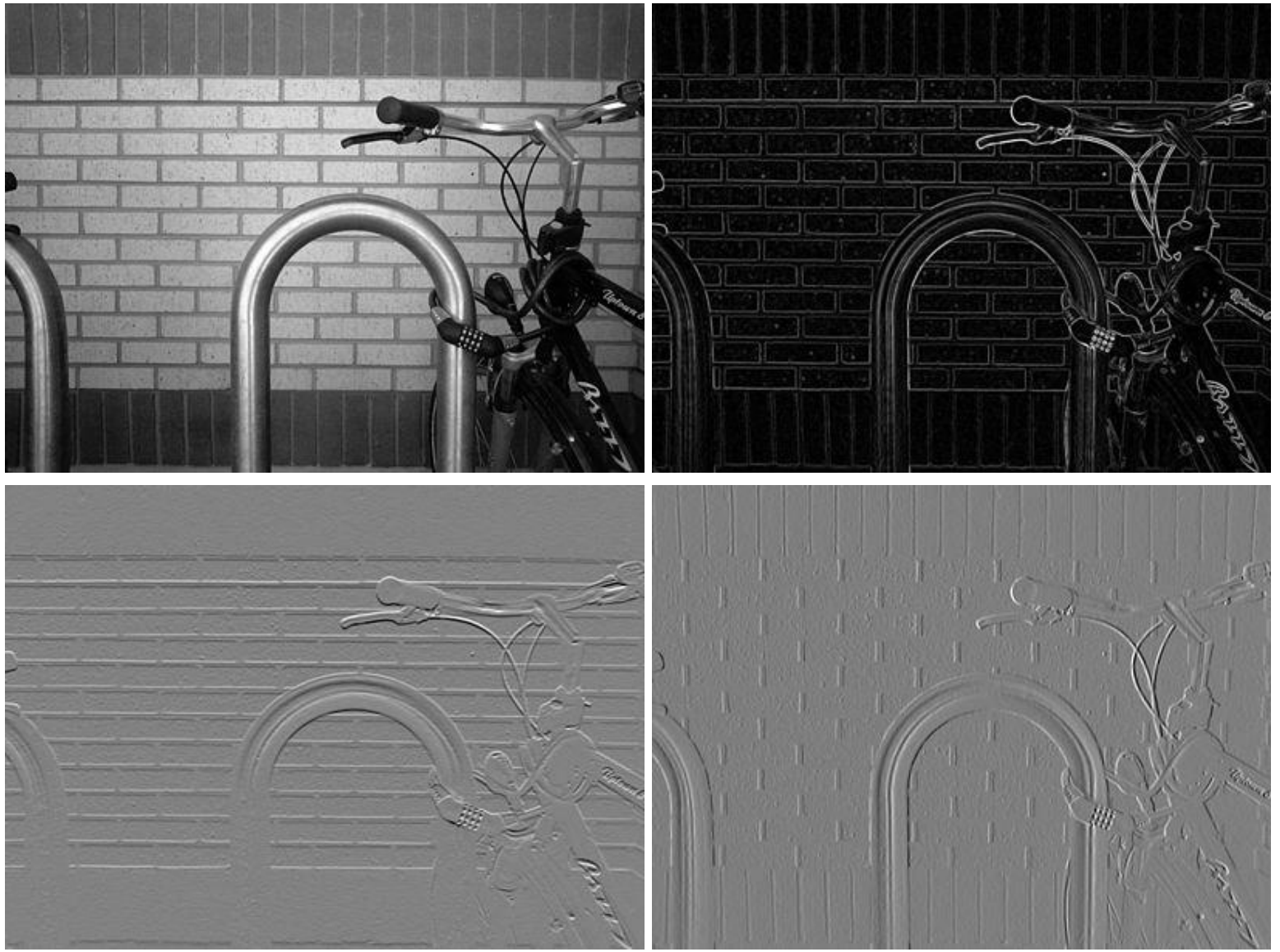
$$\mathbf{\Theta} = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

Use atan2 to avoid ambiguity

Sobel Filter example

Step 1: Calculate the gradient magnitude at every pixel location.

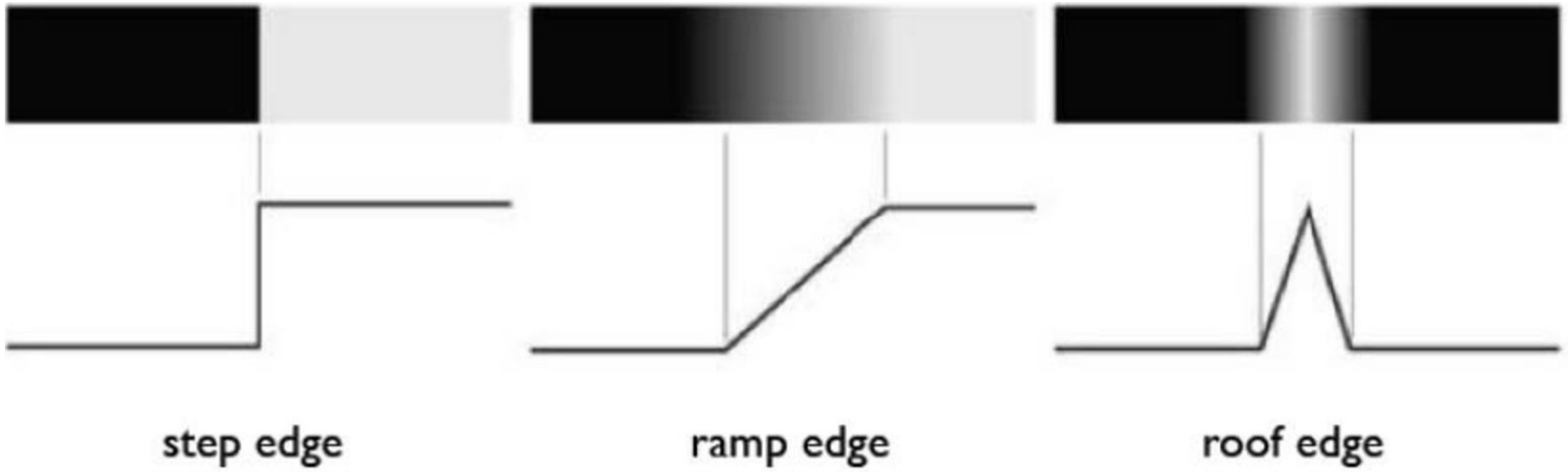
Step 2: Threshold the values to generate a binary image



- Irwin Sobel


Irwin Sobel · 2nd
IS_Consulting: Visualization, Vision & Graphics
Menlo Park, California, United States · Contact info

Sobel Filter Problems



- Poor Localization (Detects multiple adjacent edges)
- Thresholding value favors certain directions over others
  - Can miss diagonal edges more than horizontal or vertical edges
  - False negatives

What we will learn today

- Edge detector with noisy images
- Sobel Edge detector
- Canny edge detector
- Hough Transform

So far: A simple edge detector

- This theorem gives us a very useful property:

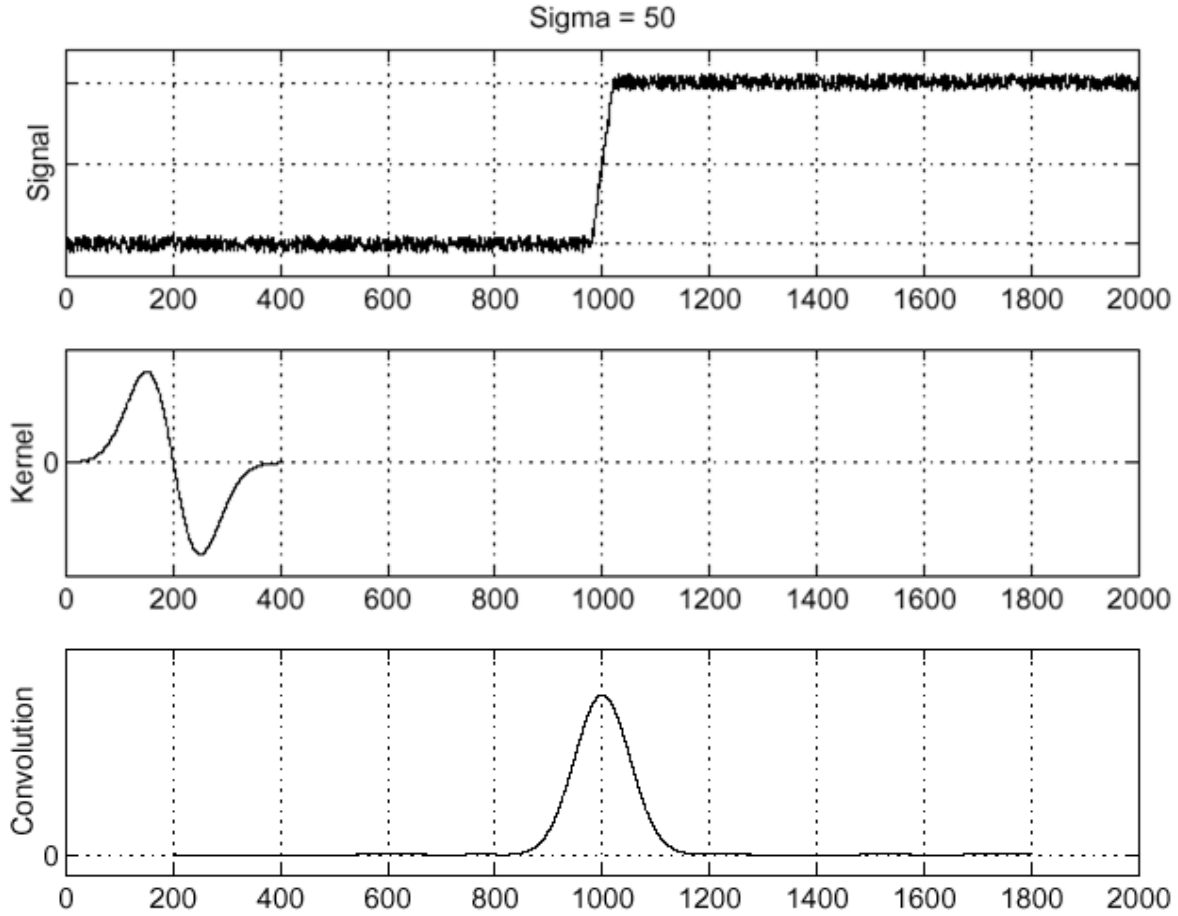
$$\frac{d}{dx}(f * h) = f * (\frac{d}{dx}h)$$

- This saves us one operation:

We can precompute: $\frac{d}{dx}h$

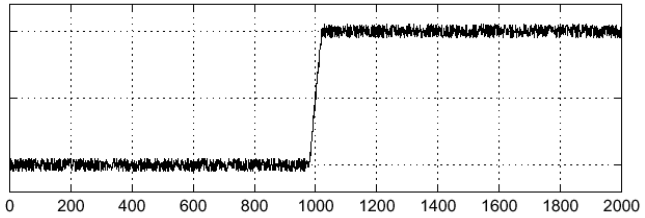
f

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



Canny edge detector

- This is probably the most widely used edge detector in computer vision
- **Theoretical model**: optimal edge detection when pixels are corrupted by additive Gaussian noise
- Theory shows that first derivative of the Gaussian closely approximates the operator that optimizes the product of ***signal-to-noise ratio***



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

Canny edge detector

1. Suppress Noise
2. Compute gradient magnitude and direction
3. Apply Non-Maximum Suppression
    - Assures minimal response
4. Use hysteresis and connectivity analysis to detect edges

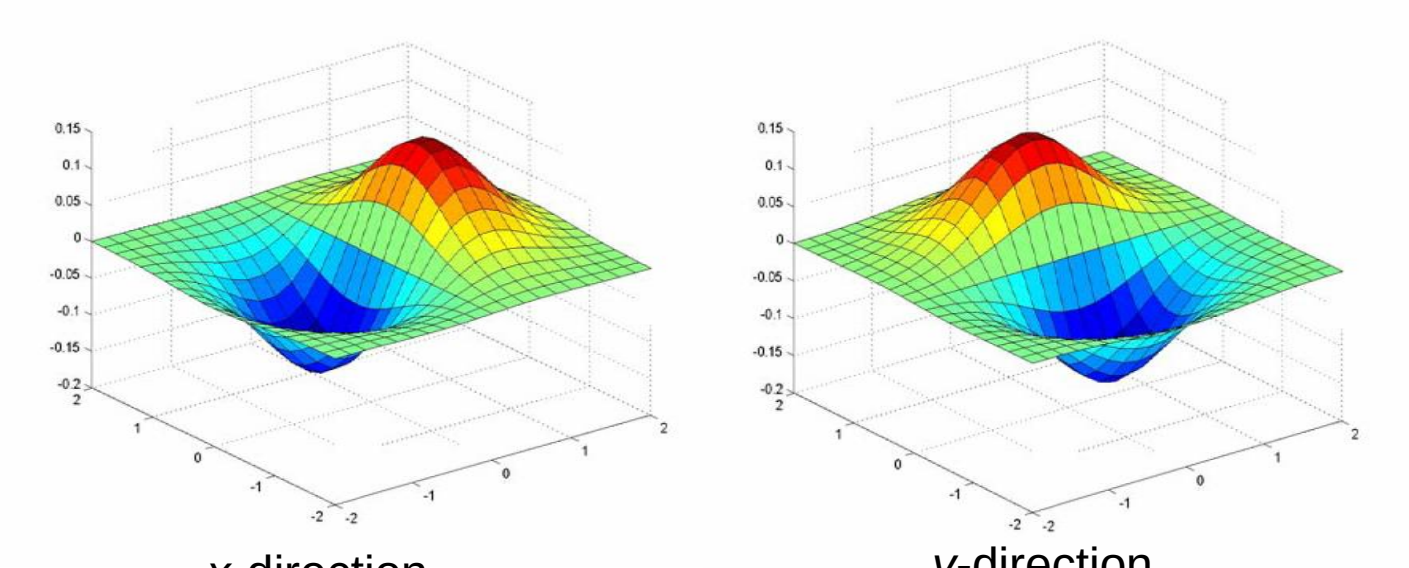
Example

- original image

Canny edge detector

1. Suppress Noise
2. Compute gradient magnitude and direction
3. Apply Non-Maximum Suppression
    - Assures minimal response
4. Use hysteresis and connectivity analysis to detect edges

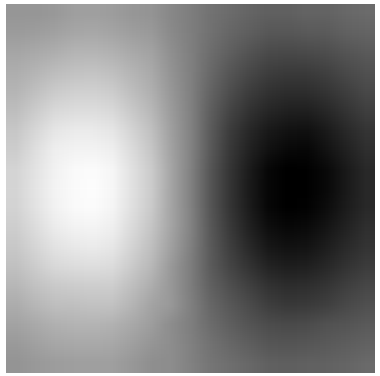
Derivative of Gaussian filter

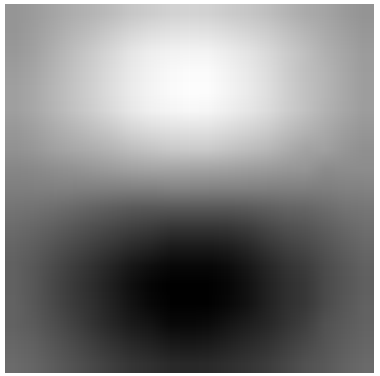


x-direction

y-direction

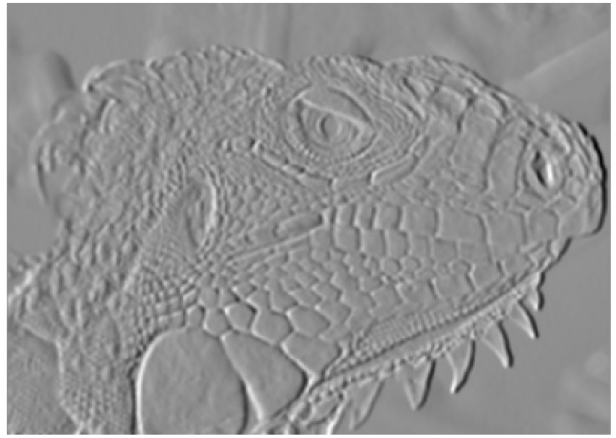
$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$



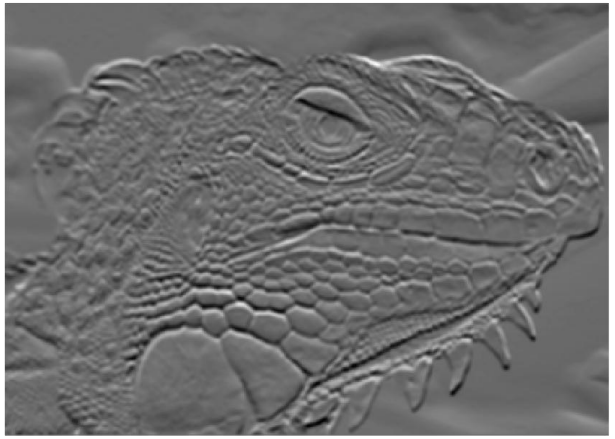


$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Compute gradients (DoG)



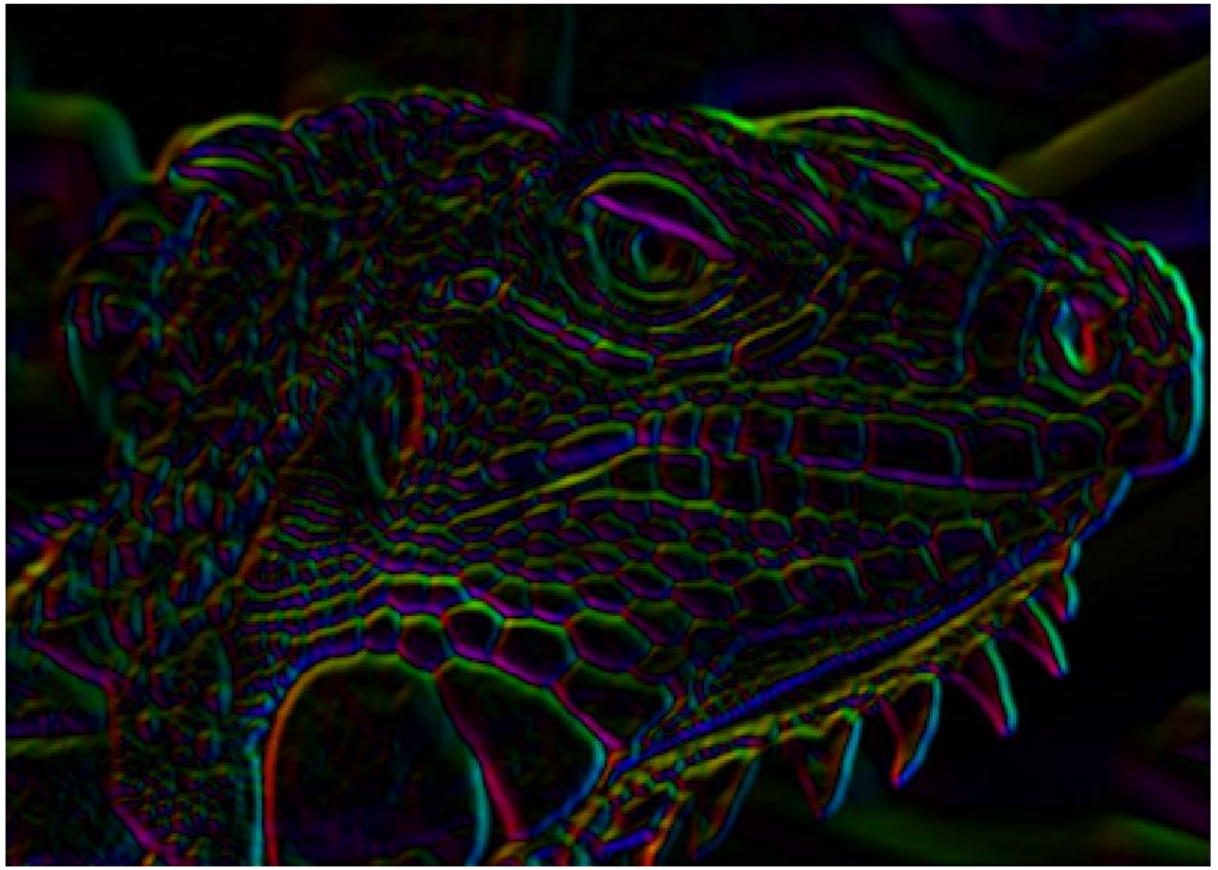
X-Derivative of Gaussian



Y-Derivative of Gaussian

Gradient Magnitude

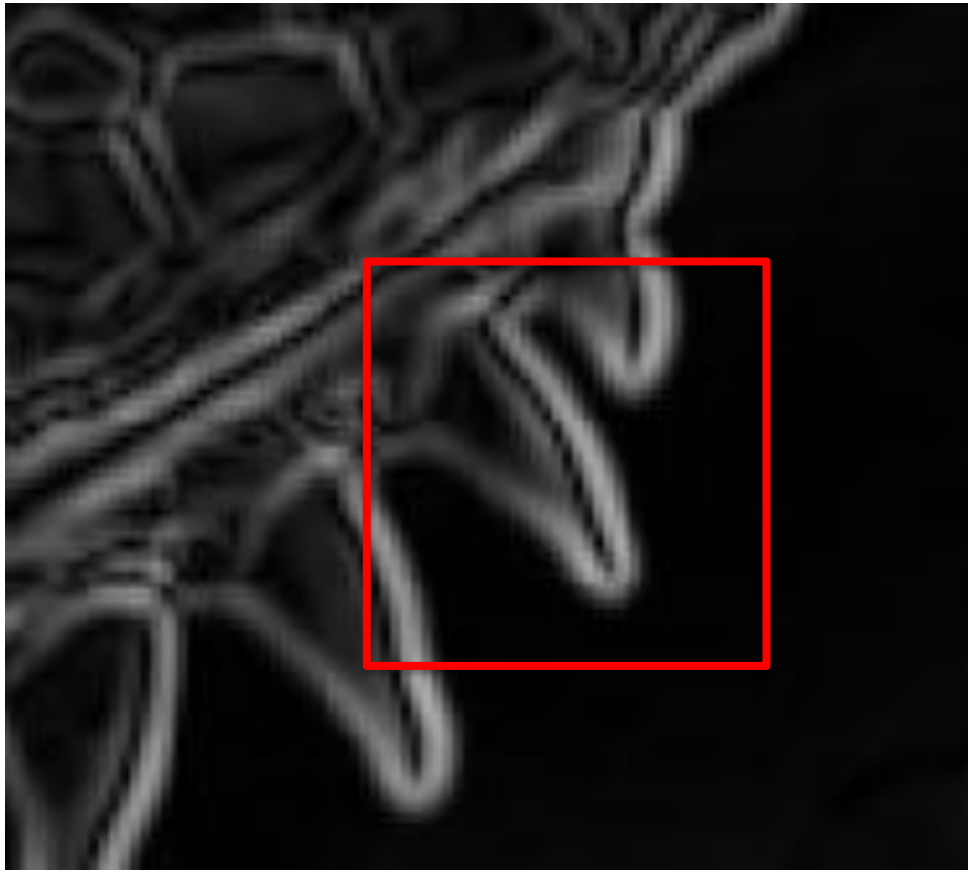
Get orientation at each pixel

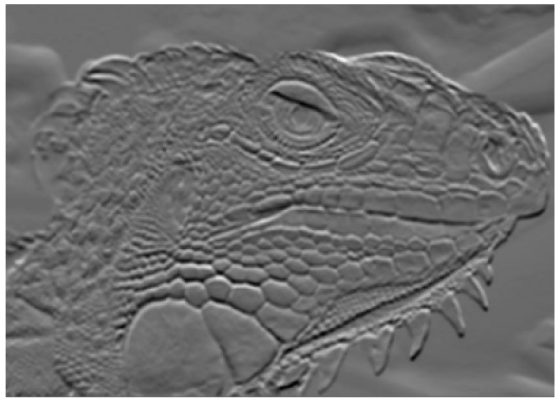


$$\Theta = \operatorname{atan}\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

= math.atan2(G_y, G_x)

Compute gradients (DoG)





Y-Derivative of Gaussian

Gradient Magnitude

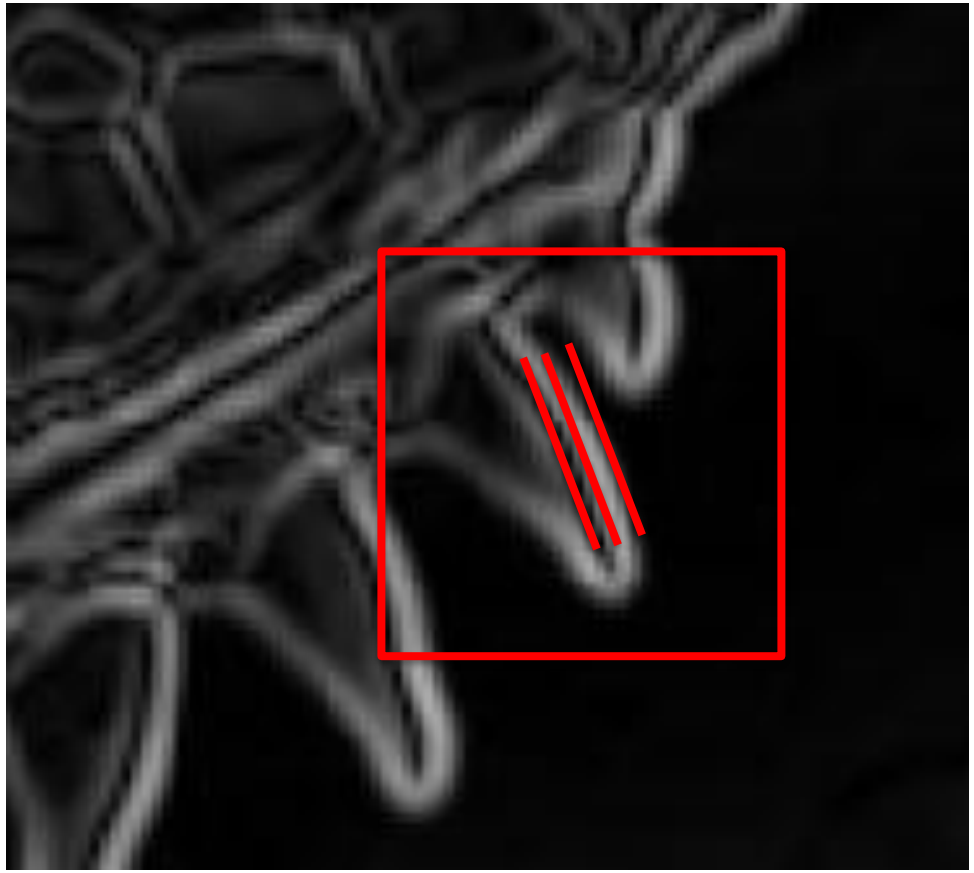
Canny edge detector

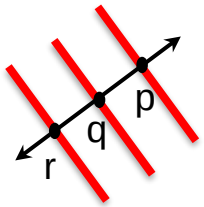
1. Suppress Noise
2. Compute gradient magnitude and direction
3. Apply Non-Maximum Suppression
    - Assures minimal response
4. Use hysteresis and connectivity analysis to detect edges

Non-maximum suppression

- **Assumption we make: An edge occurs where gradient is maximum**
  - Even if their magnitude is above the threshold
- **Suppress non-maxima neighboring** edges
  - Only suppress edges that are in the same direction nearby
  - Don’t suppress other edges

Intuition behind non-maximum suppression





Let's assume that out of the points:

p,

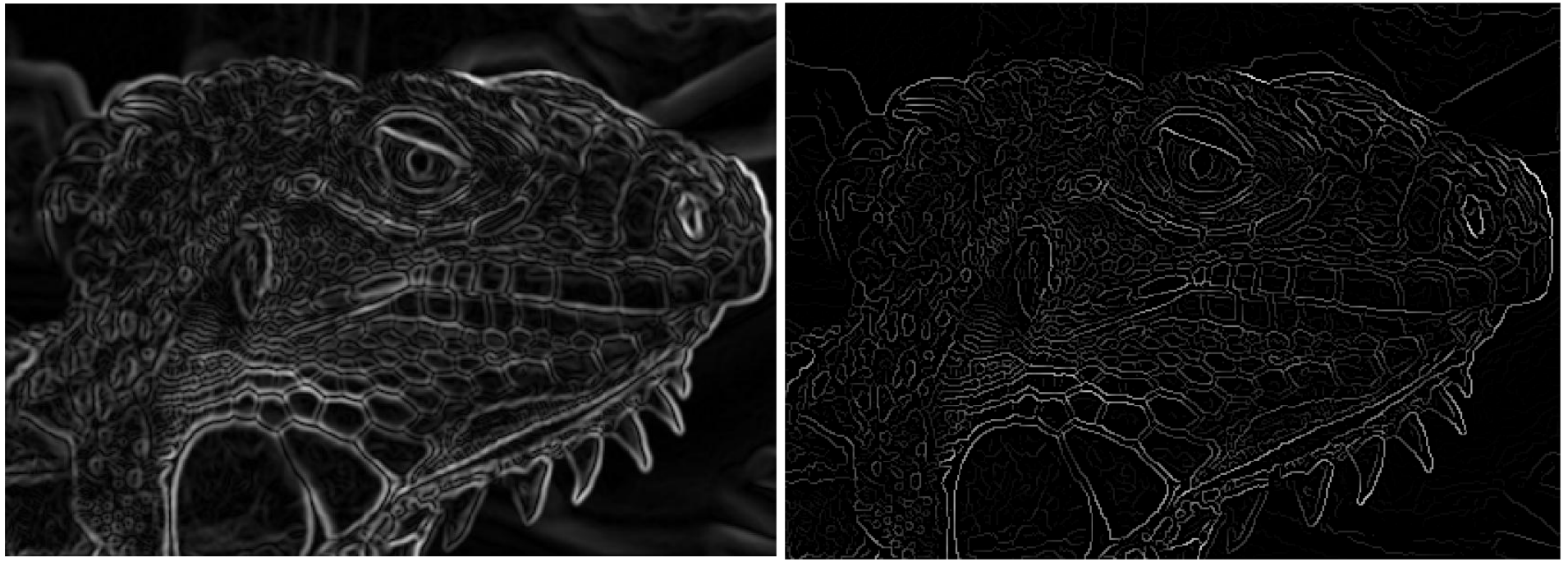
q,

r

q has the largest gradient.

Then p and r are not real edges and should be *suppressed*

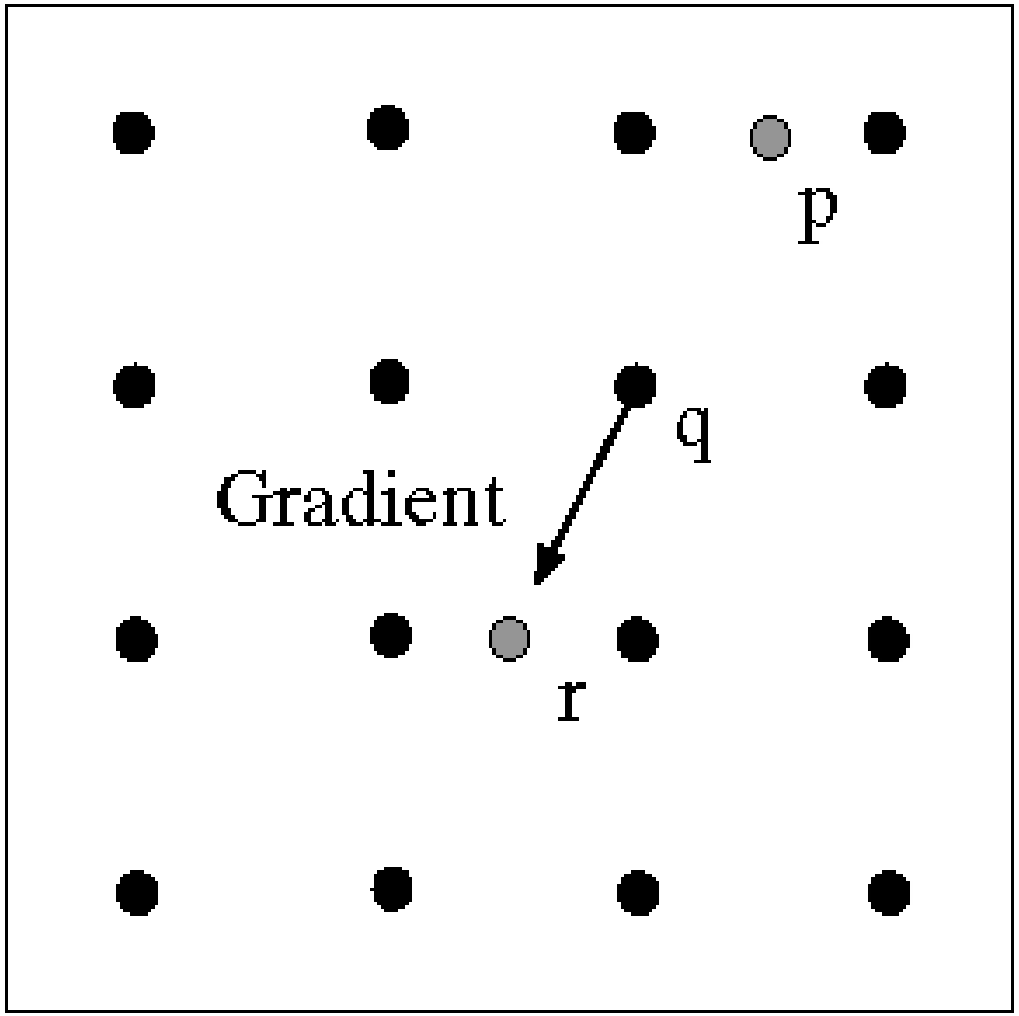
What the output looks like after Non-max Suppression



Before

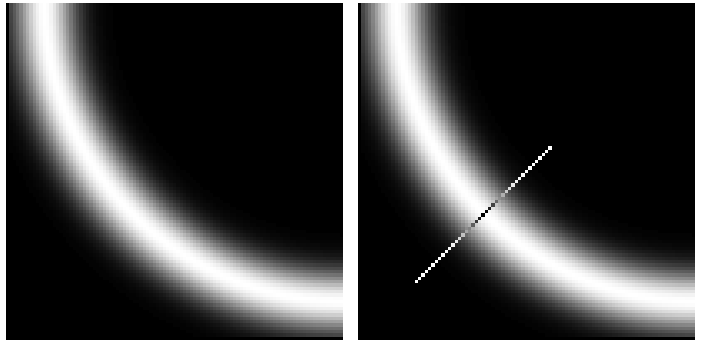
After

What if $p = [n_1, m_1]$ **or** $r = [n_2, m_2]$**,** is not a pixel location

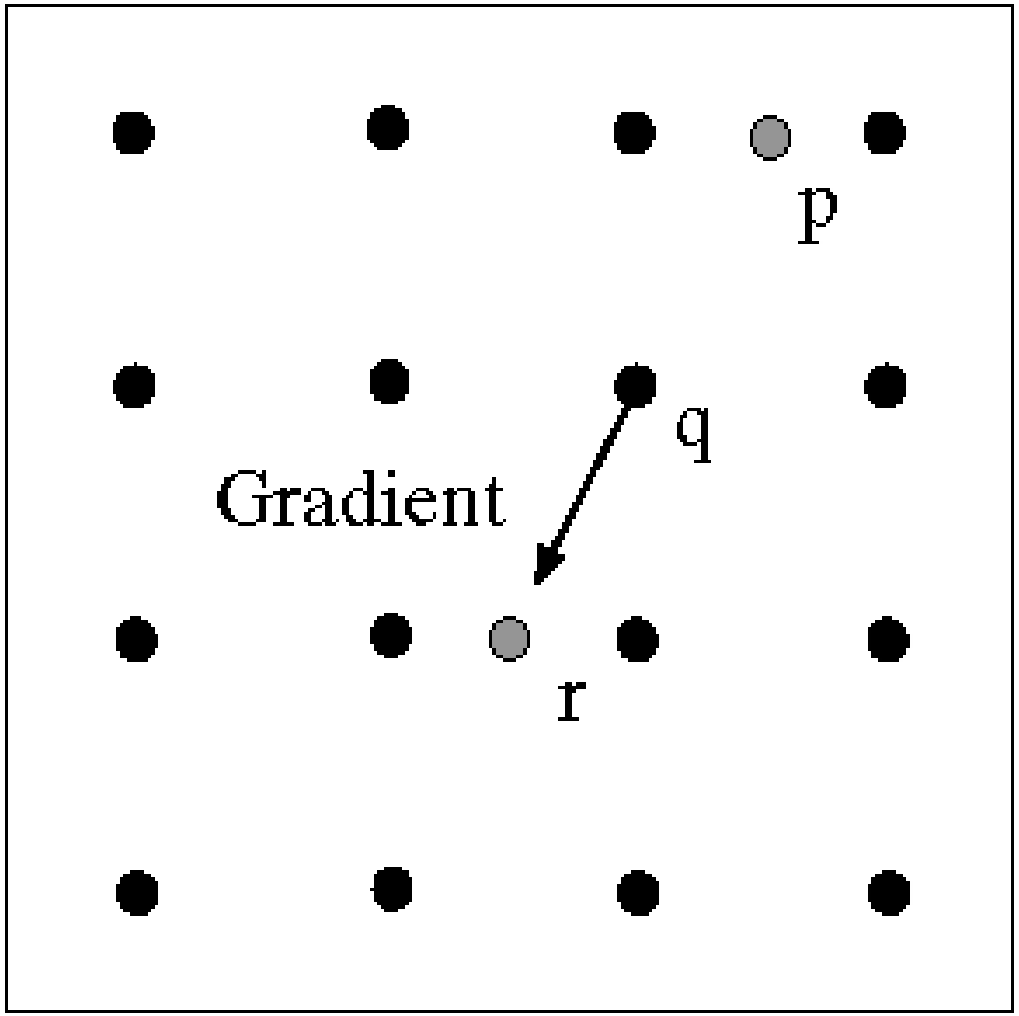


q is a maximum if the value is larger than those at both p and at r.

How should we calculate magnitude at p and r?



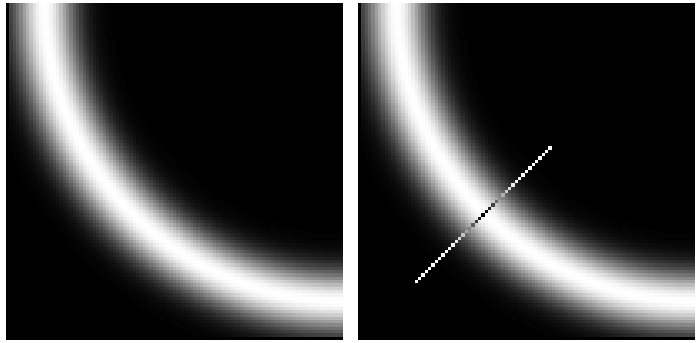
What if $p = [n_1, m_1]$ **or** $r = [n_2, m_2]$**,** is not a pixel location



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

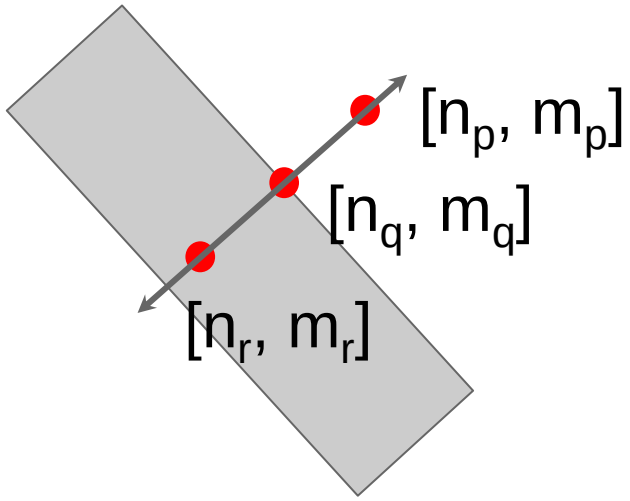
How should we calculate magnitude at p and r?

Calculate p and r as averaged values of top k=8 closest pixel locations*



Some people use Bilinear interpolation.

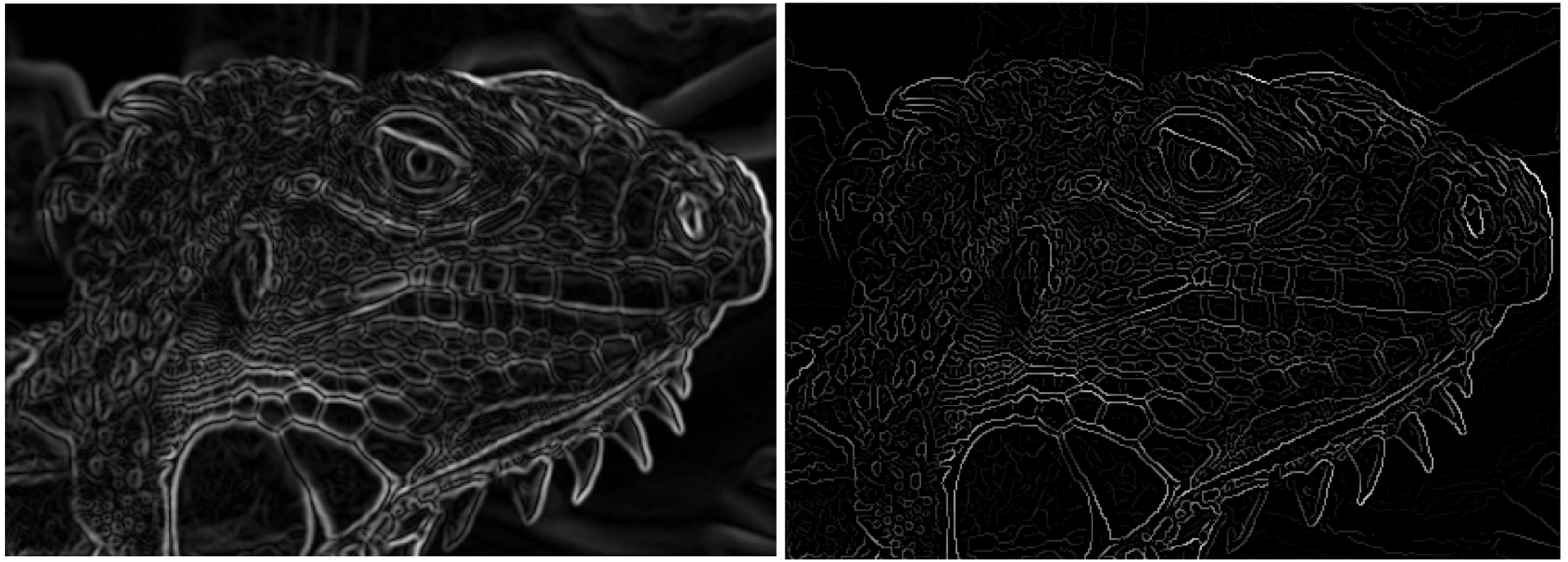
In code, you will calculate gradient magnitudes at every q and set it to zero if it is not the local max



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

$$G[n_q, m_q] = \begin{cases} G[n_q, m_q] & \text{if } G[n_q, m_q] > G[n_p, m_p] \text{ and } G[n_q, m_q] > G[n_r, m_r] \\ 0 & \text{otherwise} \end{cases}$$

What the output looks like after Non-max Suppression



Before

After

Canny edge detector

1. Suppress Noise
2. Compute gradient magnitude and direction
3. Apply Non-Maximum Suppression
    - Assures minimal response
4. Use hysteresis and connectivity analysis to detect edges

Problem: if your threshold is too high (left) or too low (right), you have too many or too few edges

Problem: Also, you have too many disconnected edges

What the output of hysteresis looks like:

Before

After

Hysteresis thresholding connects edges to create long edges

- How does it work?

- Define two thresholds: **Low** and **High**
  - If less than Low => **not an edge**
  - If greater than High => **strong edge**

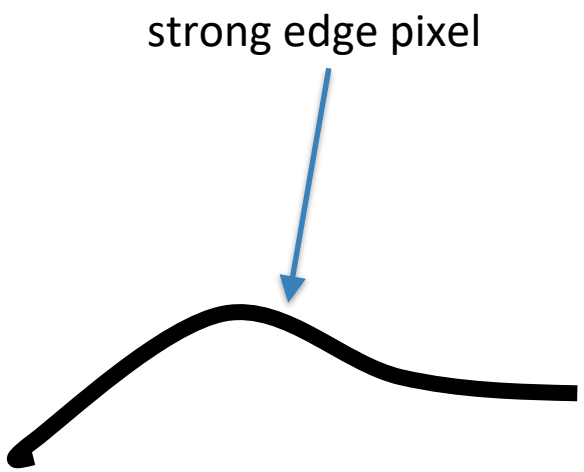
  - If between Low and High => **weak edge**

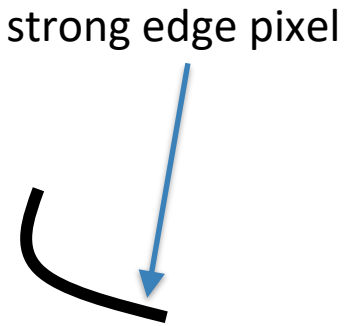
Hysteresis thresholding

If the gradient at a pixel is between Low and High thresholds,

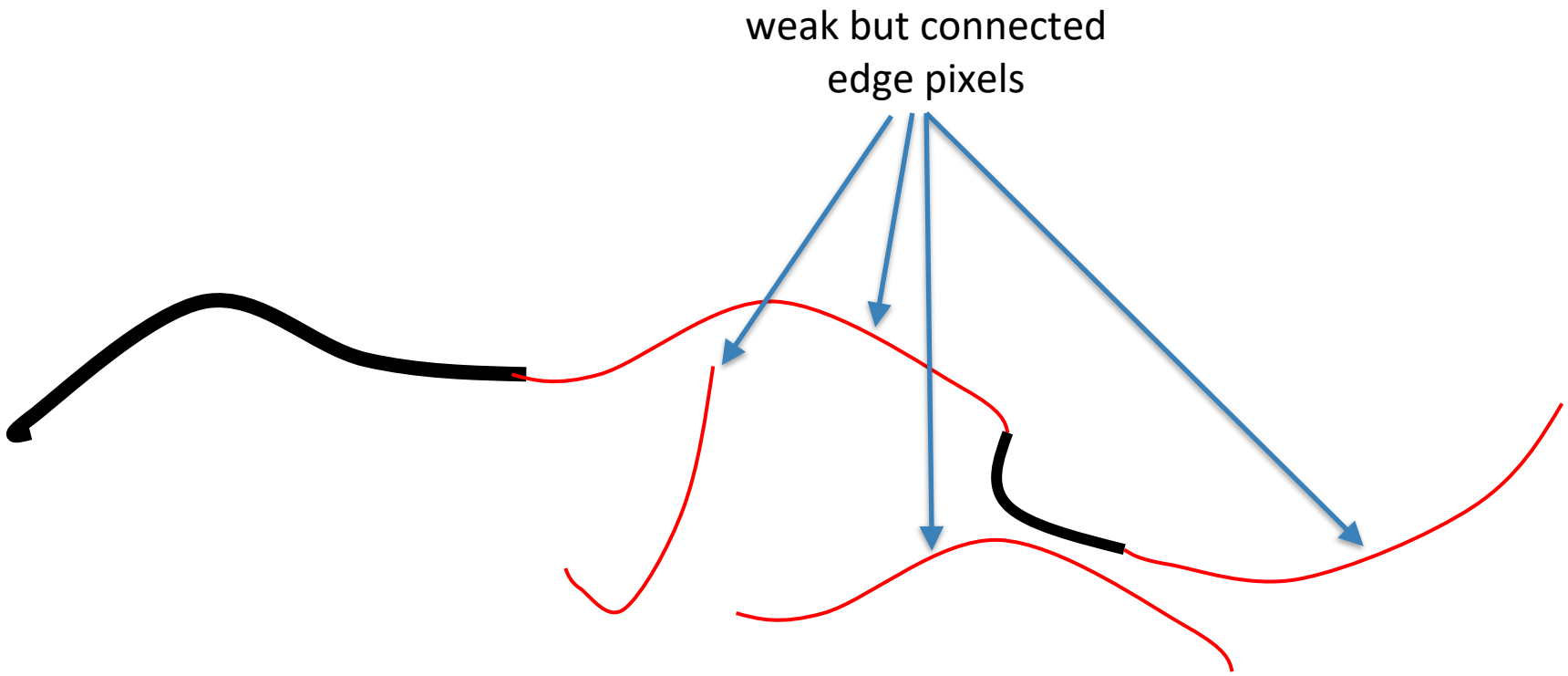
- Consider its neighbors iteratively then declare it an “edge pixel” if it is connected to a ‘strong edge pixel’ directly or via pixels between Low and High

All the white pixels are not edges (below the low threshold)
The black pixels below are strong edges (above the high threshold)

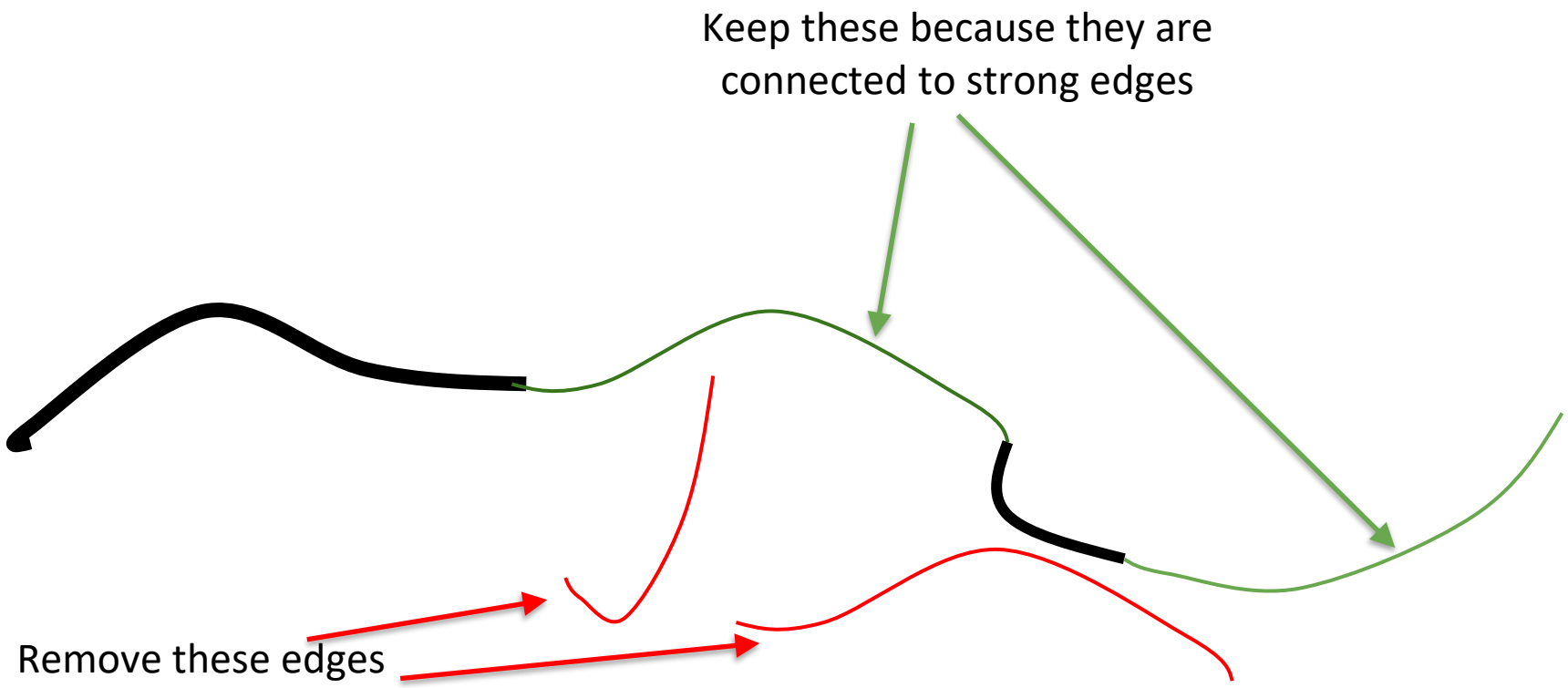




Now, let's assume all the red pixels are weak edges (between low and high thresholds)



Now, let's assume all the red pixels are weak edges (between low and high thresholds)



Final Canny Edges

Canny edge detector

1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
   - Reduce multi-pixel wide edges down to single pixel edge
4. Thresholding and linking (hysteresis):
   - Define two thresholds: low and high
   - Connect edges together and remove everything else

Effect of σ (Gaussian kernel spread/size)

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

The choice of σ depends on desired behavior

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

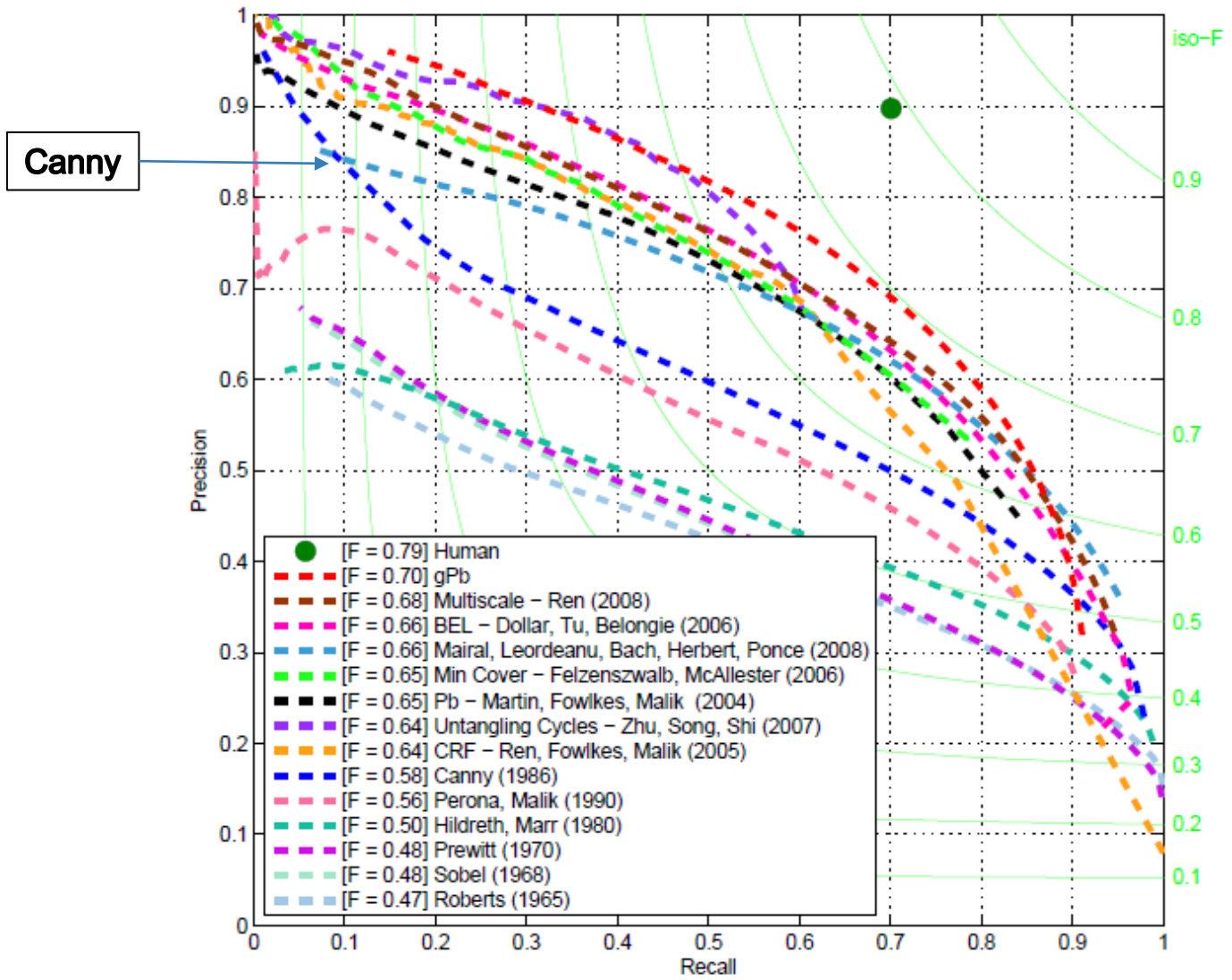
Gradients
(e.g. Canny)

Human

45 years of edge detection

Recall is how many of the actual edges did it find.

Precision is how good were the ones it called edges (were they edges?).



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

What we will learn today

- Edge detector with noisy images
- Sobel Edge detector
- Canny edge detector
- Hough Transform

Hough transform

How to transform edge detections into lines

Original image

Edge image

Hough transform

- It was introduced in 1962 (Hough 1962) and first used to find lines in images a decade later (Duda 1972).

- **Caveat**: Hough transform can detect lines, circles and other shapes
  - but only for shapes that can be expressed as a math equation.
- It gives us good detections even when the image is noisy and even if the shape is partially hidden.

Input to Hough transform algorithm

- We have performed some edge detection (Sobel filter, Canny Edge detector, etc.), including a thresholding of the edge magnitude image.
- Thus, we have some pixels that may partially describe the boundary of some objects.

Edge image

Detecting lines using Hough transform

- We wish to find sets of pixels that make up straight lines.
- Instead of using [n, m], this might be easier to do with (x, y)

How do we transform [n, m] to (x, y)?

- Simple: We assume
  - n = y,
  - m = x.

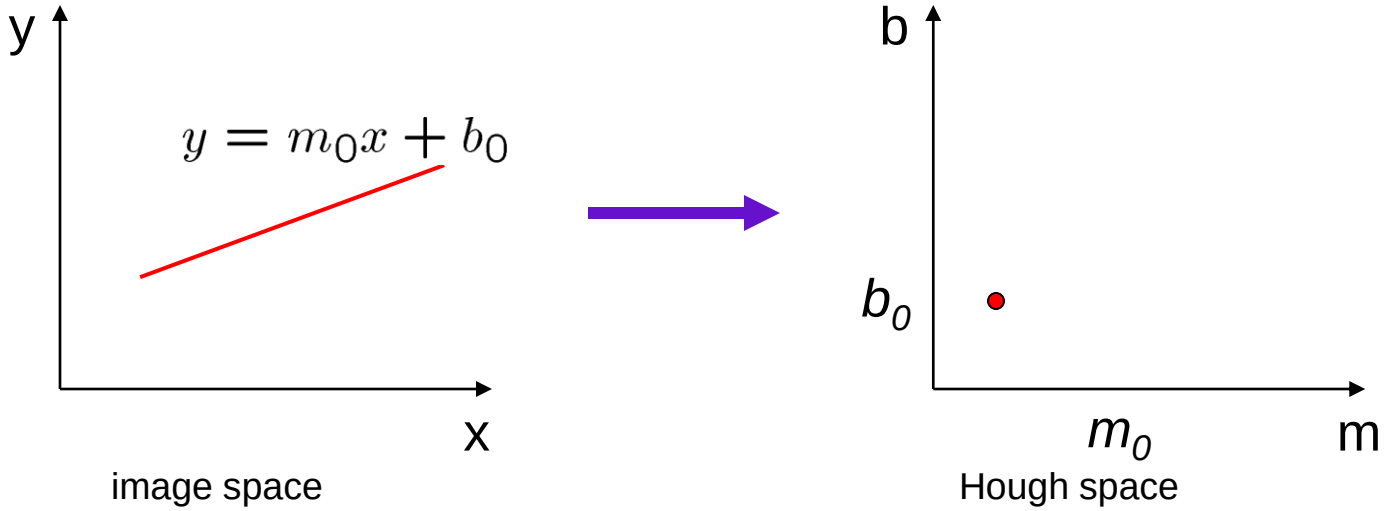
- So, f[n, m] = f[y, x]

Finding lines in an image

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

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

Finding lines in an image



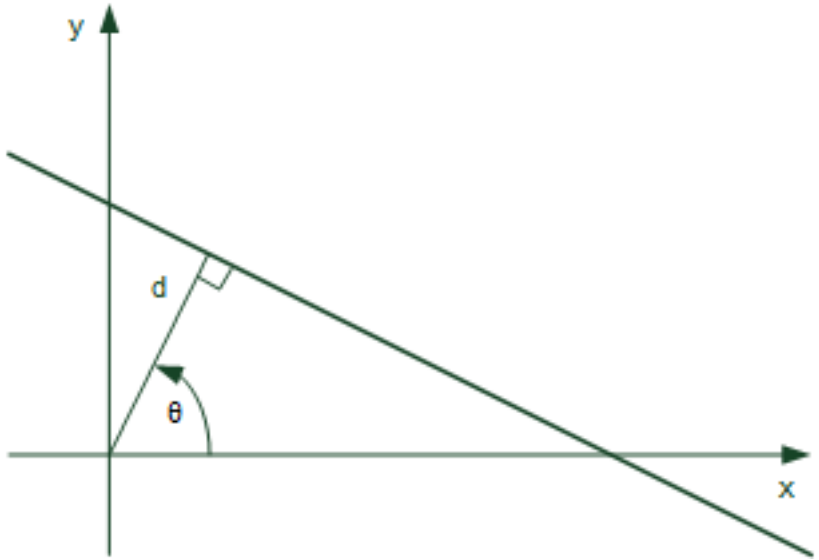
- Connection between image (x,y) and Hough (m,b) spaces
  - A line in the image corresponds to a point in Hough space
  - To go from image space to Hough space:
    - given a set of points (x,y), find all (m,b) such that y = mx + b

●Hough transform algorithm

- Typically use a different parameterization

$$d = xcos\theta + ysin\theta$$

  - d is the perpendicular distance from the line to the origin
  - θ is the angle of this perpendicular with the horizontal.



Hough transform algorithm

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

     compute gradient magnitude m and angle θ

     $d = xcos\theta + ysin\theta$

     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

     $d = xcos\theta + ysin\theta$

Array H

Complexity?

●How do you extract the line segments from the accumulators? (this is nonstandard)

pick the bin of H with highest value V
while V > value_threshold {

- order the corresponding pointlist from PTLIST
- merge in high gradient neighbors within 10 degrees
- create line segment from final point list
- zero out that bin of H
- pick the bin of H with highest value V }

●Example

gray-tone image

0	0	0	100	100
0	0	0	100	100
0	0	0	100	100
100	100	100	100	100
100	100	100	100	100

DQ

-	-	3	3	-
-	-	3	3	-
3	3	3	3	-
3	3	3	3	-
-	-	-	-	-

THETAQ

-	-	0	0	-
-	-	0	0	-
90	90	40	20	-
90	90	90	40	-
-	-	-	-	-

Accumulator H

360	-	-	-	-	-	-	-
.	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-
3	4	-	1	-	2	-	5
0	-	-	-	-	-	-	-
distance							
angle	0	10	20	30	40	…	90

PTLIST

360	-	-	-	-	-	-	-
.	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-
3	*	-	*	-	*	-	*
0	-	-	-	-	-	-	-

(3,1)
(3,2)
(4,1)
(4,2)
(4,3)

(1,3)(1,4)(2,3)(2,4)

- Line segments from Hough Transform

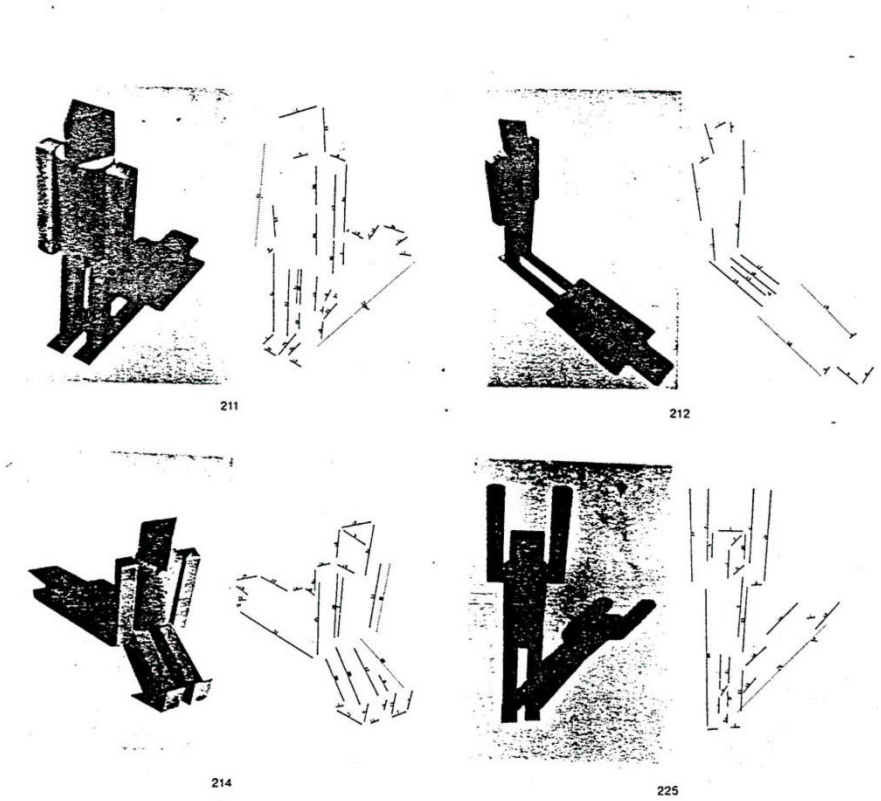


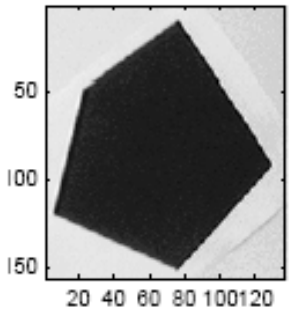
Fig.7. Puppet scenes 211, 212, 214, 225 and the edges recovered by the algorithm.

Extensions

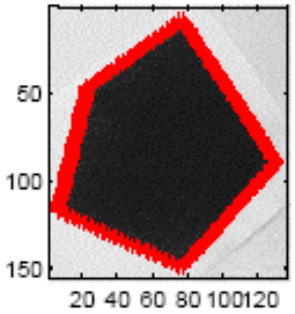
- Extension 1: Use the image gradient (we just did that)

- Extension 2
  - give more votes for stronger edges
- 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, How?
- Extension 5; the Burns procedure. Uses only angle, two different quantifications, and connected components with votes for larger one.
- Extension 6 (in your homework code). Use a small range of angles at each edge point in case there is more than one line through the point.

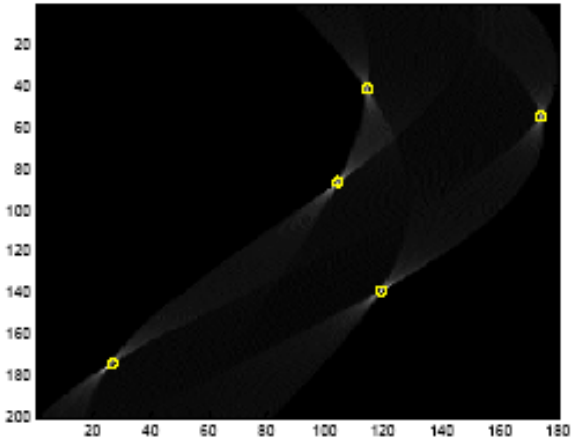
Examples



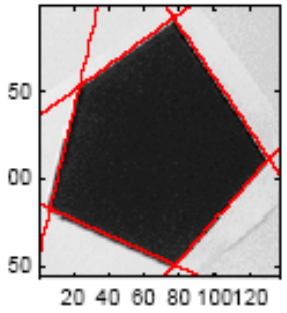
Original Image



Detected Edges



The vote histogram with the detected lines marked with 'o'



Detected lines

Examples cont

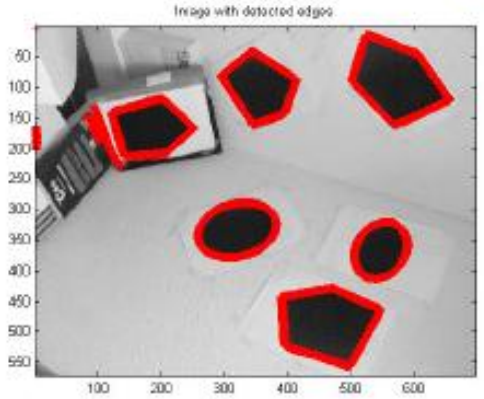
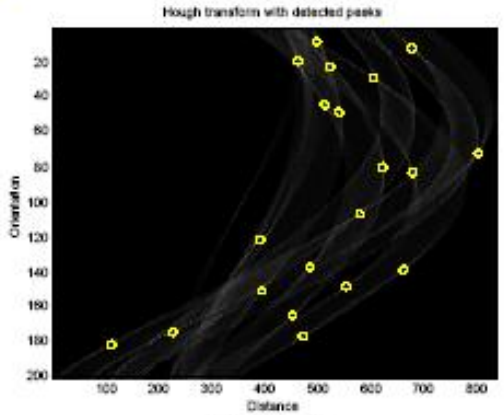


Image with detected edges



The vote histogram with the selected lines

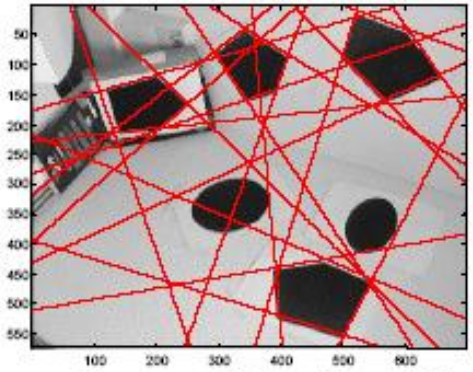


Image with the detected lines

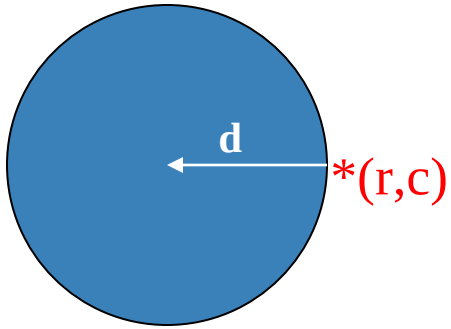
●Hough Transform for Finding Circles

Equations:

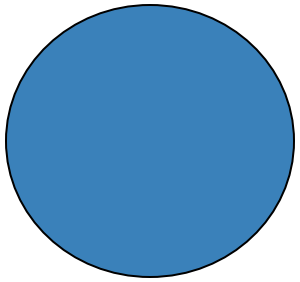
$$r = r0 + d \sin \theta$$
$$c = c0 - d \cos \theta$$

r, c, d are parameters

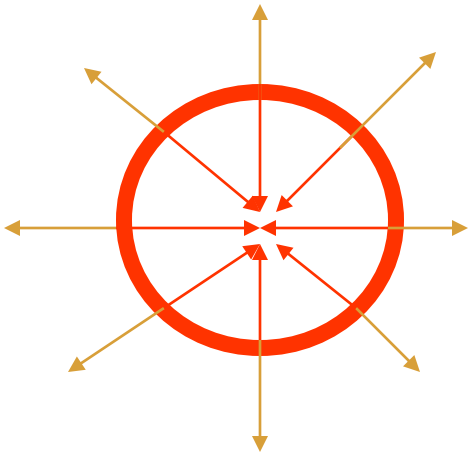
Main idea: The gradient vector at an edge pixel points to the center of the circle.



●Why it works

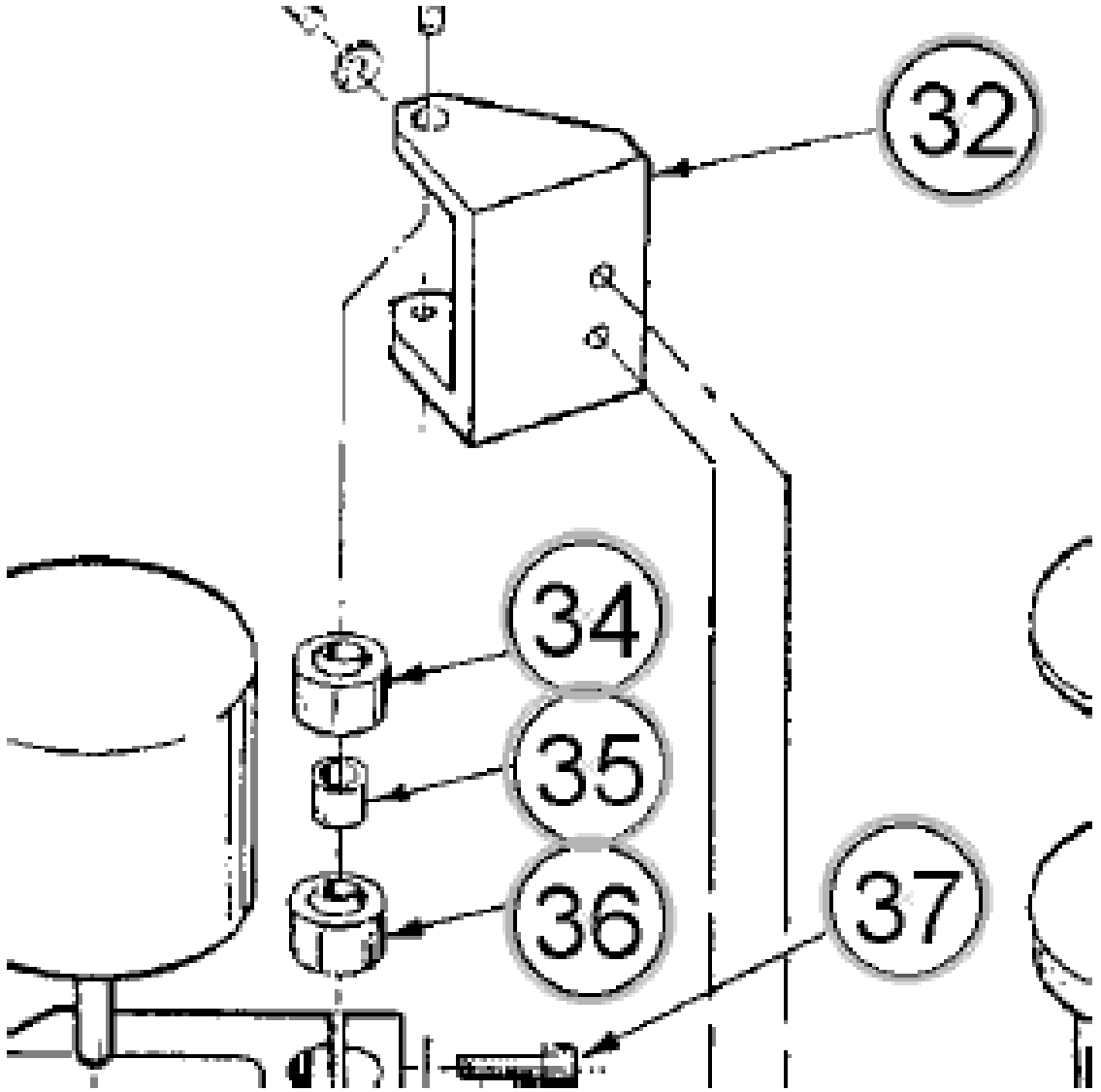


Filled Circle:
Outer points of circle have gradient direction pointing to center.



Circular Ring:
Outer points gradient towards center.
Inner points gradient away from center.

The points in the away direction don't accumulate in one bin!


32
34
35
36
37

- Finding lung nodules (Kimme & Ballard)

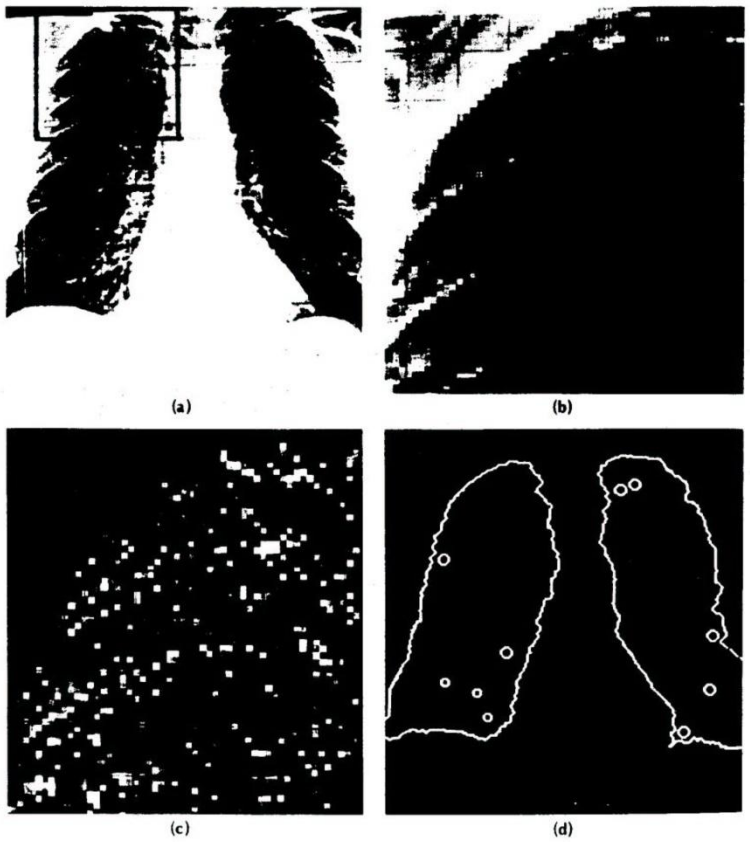


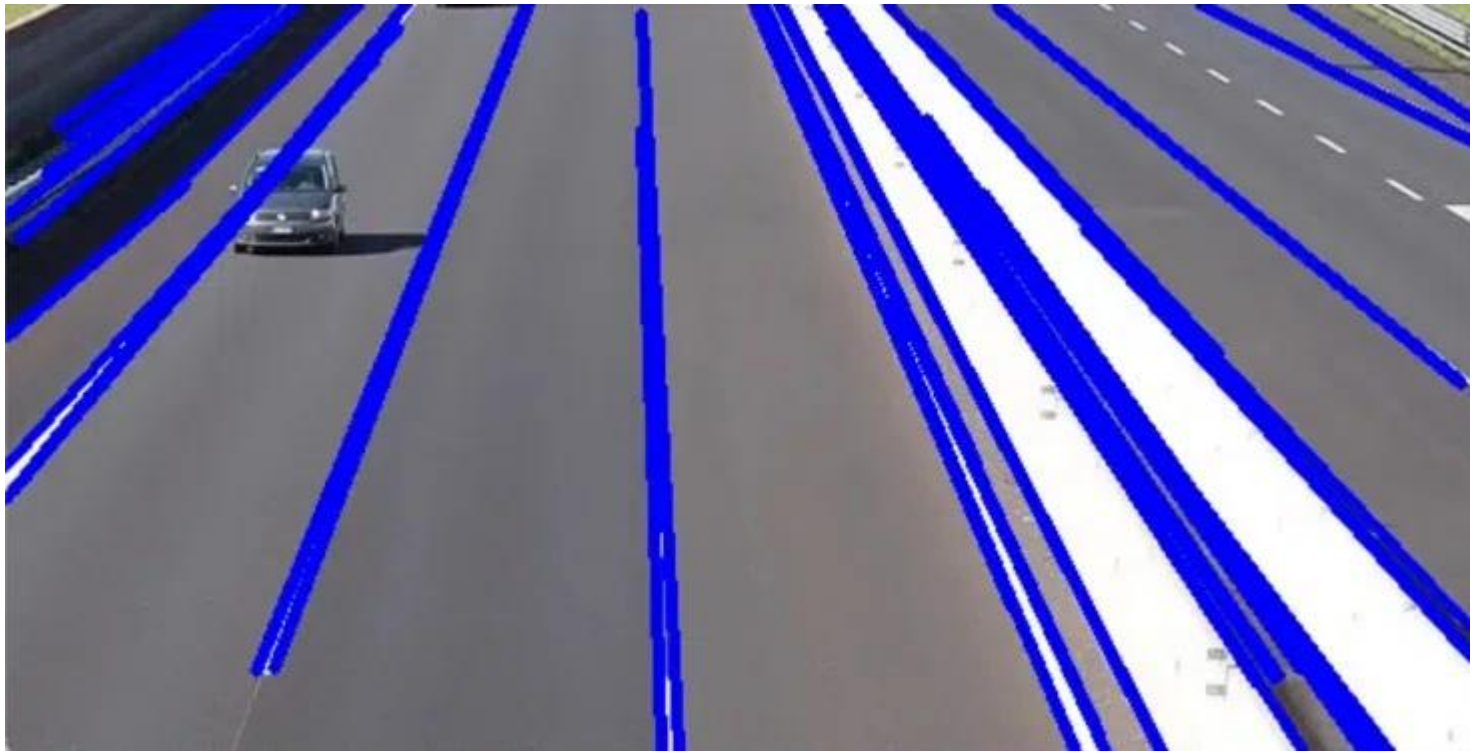
Fig. 4.7 Using the Hough technique for circular shapes. (a) Radiograph. (b) Window. (c) Accumulator array for $r = 3$. (d) Results of maxima detection.

Hough transform remarks

- **Advantages**:
  - Conceptually simple.
  - Easy implementation
  - Handles missing and occluded data very gracefully.
  - Can be adapted to many types of forms, not just lines
  - Runs in O(N*num_ang_per_point) where N is the number of edge pixels

- **Disadvantages**:
  - Computationally complex for shapes with many parameters.
  - Looks for only one single shape of object
  - Can be “fooled” by “apparent lines”.
  - The length and the position of a line segment cannot be determined.
  - Co-linear line segments cannot be separated.

Applications



Summary

- Edge detector with noisy images
- Sobel Edge detector
- Canny edge detector
- Hough Transform

Optional reading:
Szeliski, Computer Vision: Algorithms and Applications, 2nd Edition
Sections 7.1, 8.1.4

Next time

Key Points and Corners