Lecture 5

Detecting Lines

Administrative

A1 is out

- It is graded
- Due **April 18th**

So far: discrete derivatives in 3 ways

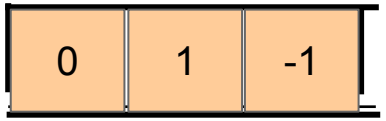
$$\frac{df}{dx} = f[x] - f[x-1]$$ Backward

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

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

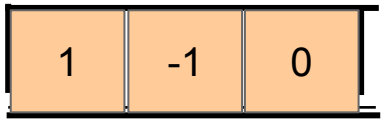
So far: Designing filters that perform differentiation

- Using Backward differentiation:



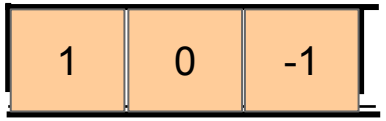
$$g[n,m] = f[n,m] - f[n,m-1]$$

- 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

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

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

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

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

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

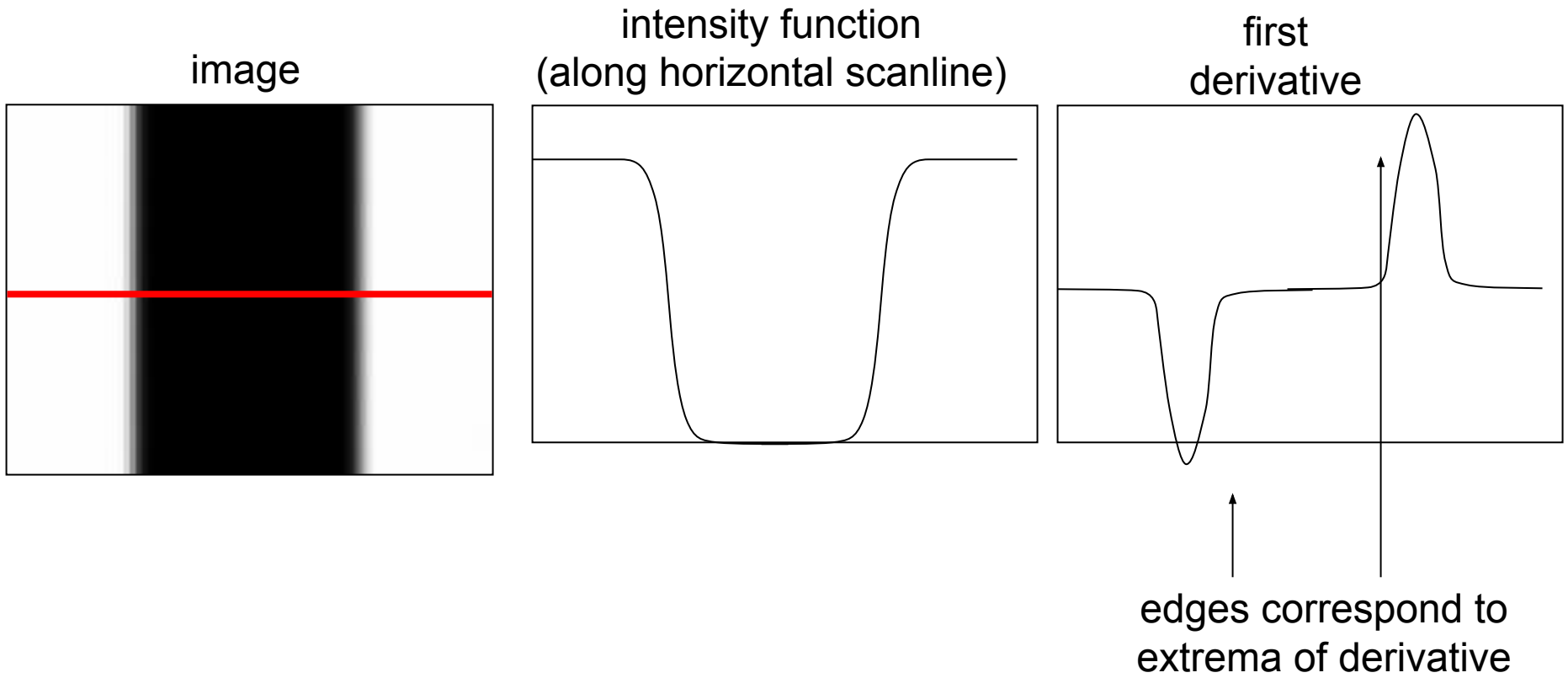
Today's agenda

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

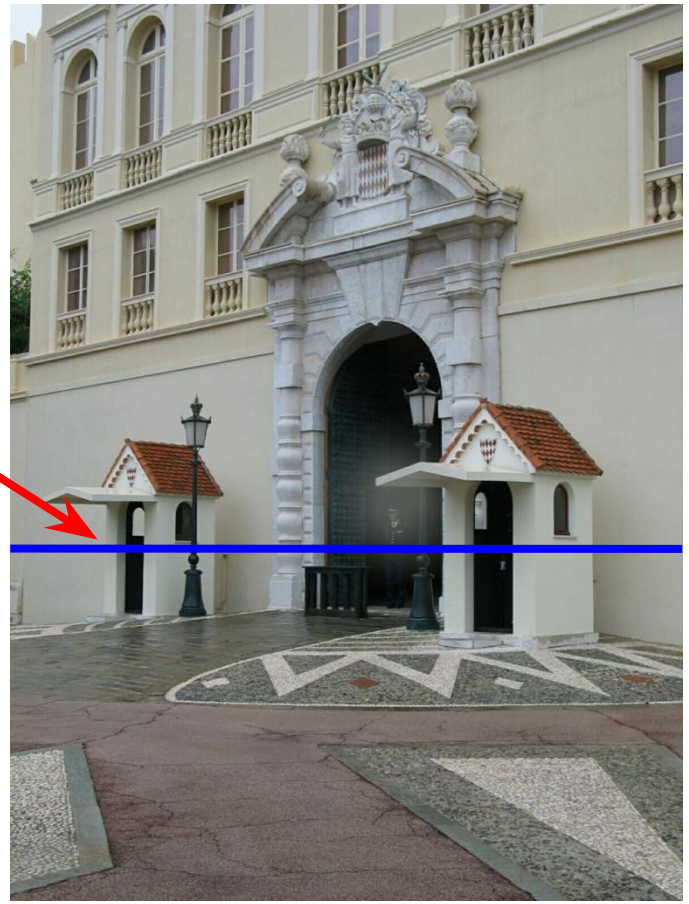
Optional reading:
Szeliski, Computer Vision: Algorithms and Applications, 2nd Edition
Sections 7.1, 8.1.4

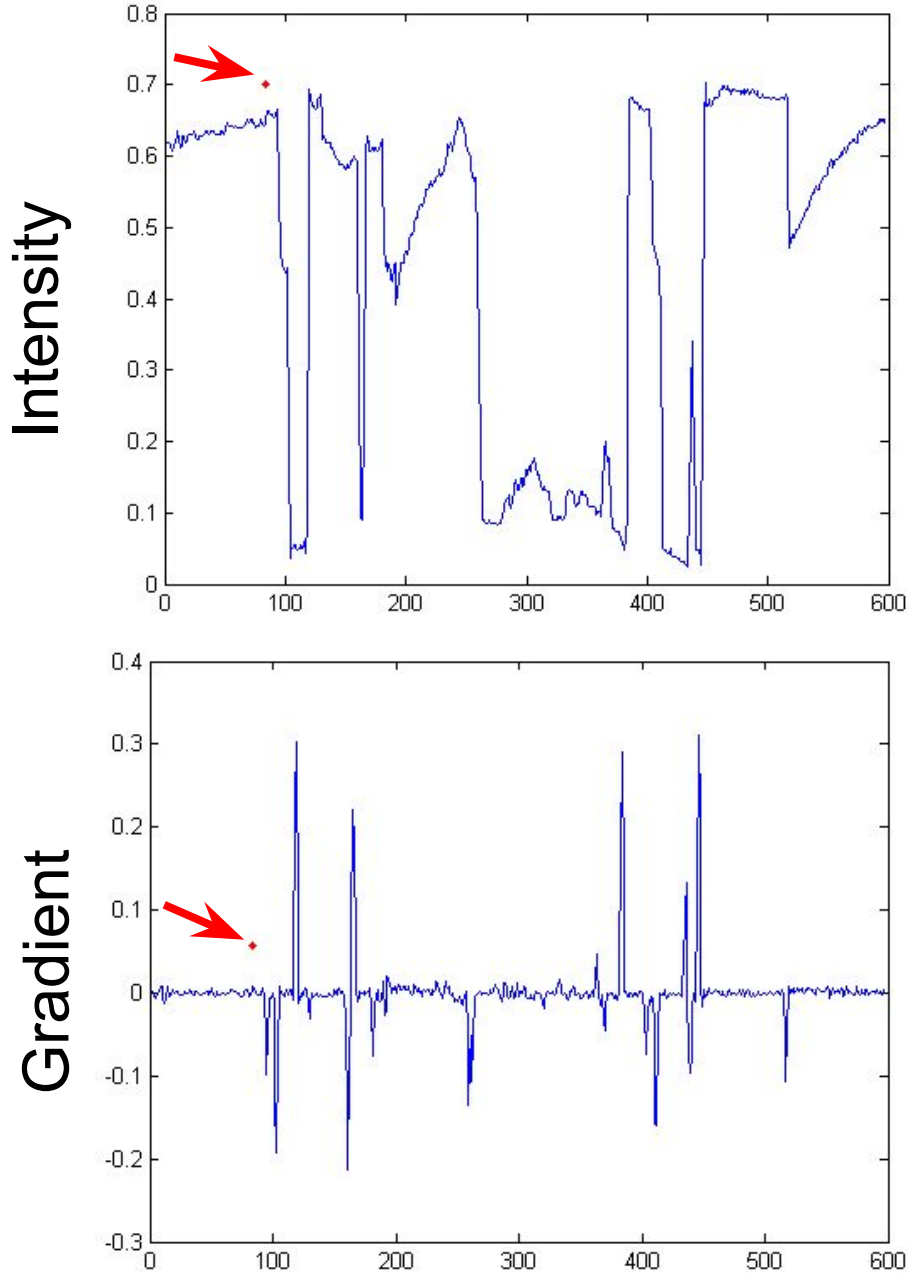
Characterizing edges

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

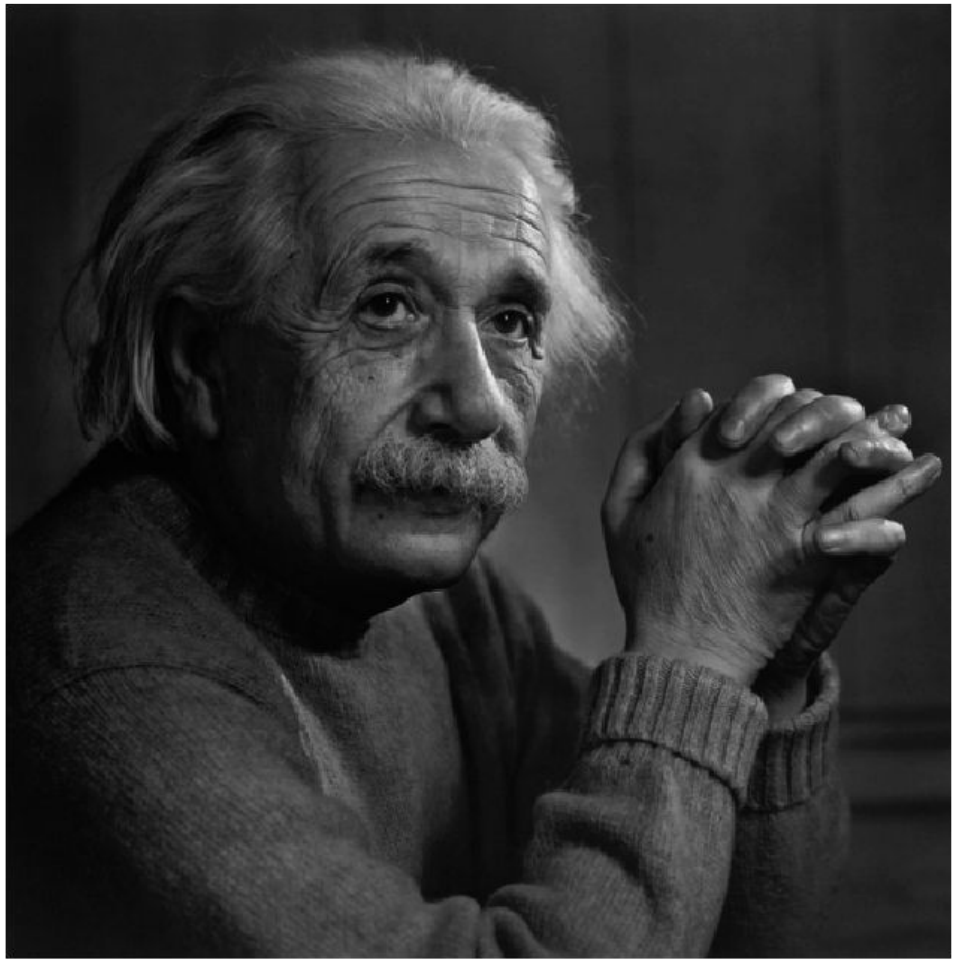


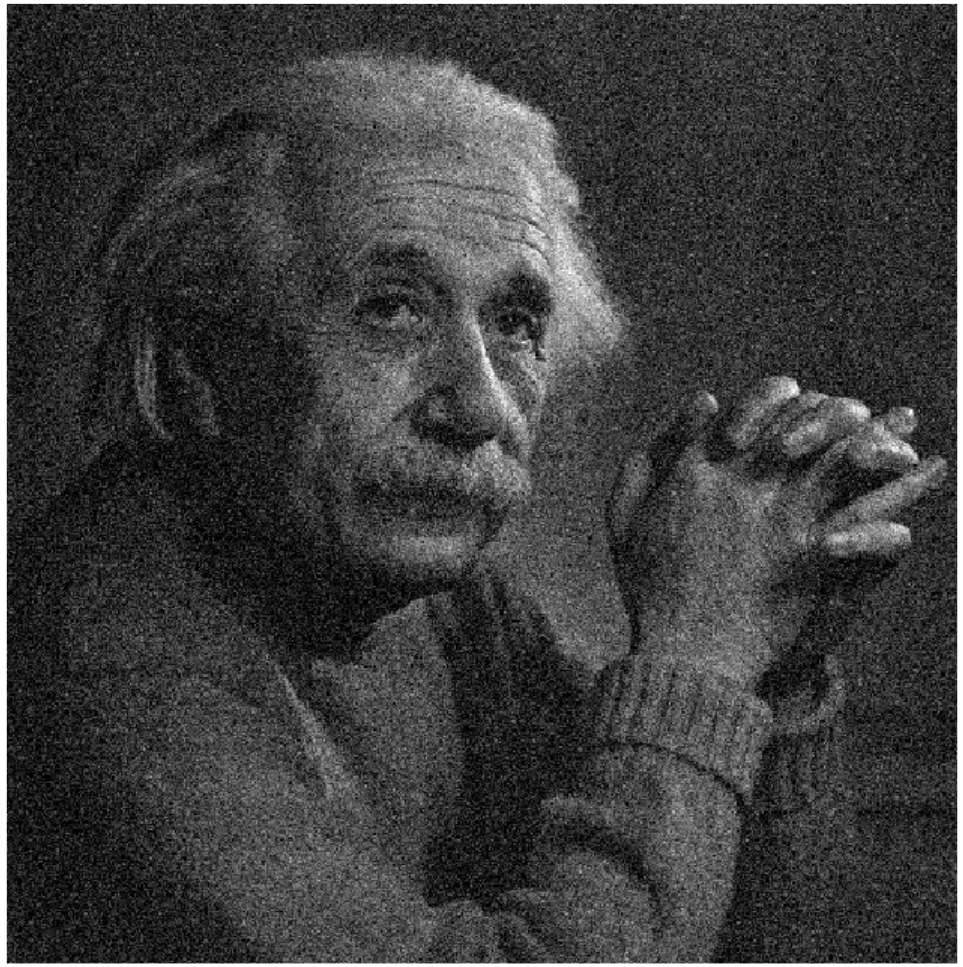
Intensity profile





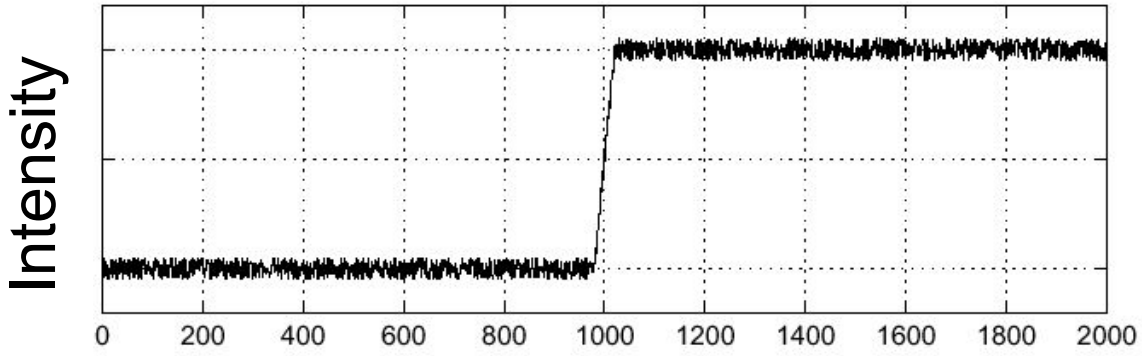
Q. What will happen if we use this edge detector on a noisy pixels?





Effects of noise

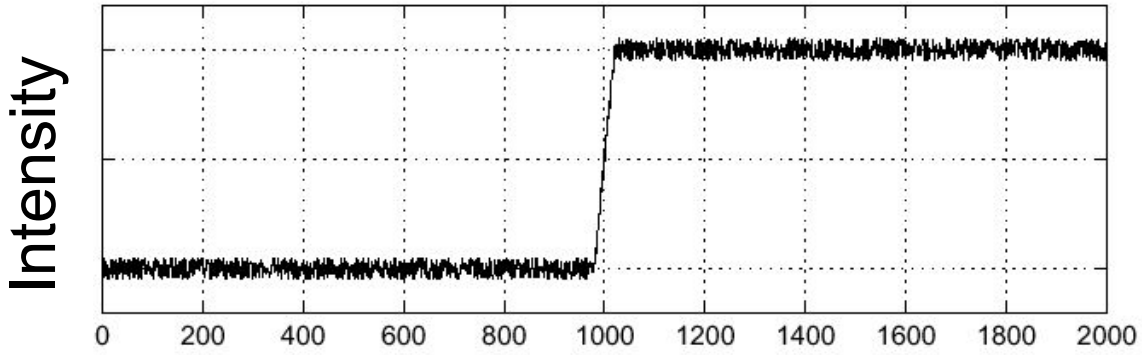
- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal

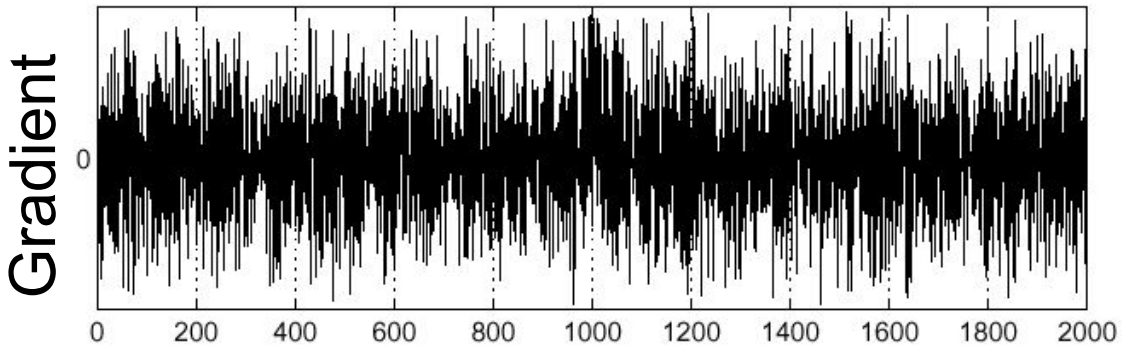


Source: S. Seitz

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

- 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?

Source: D. Forsyth

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

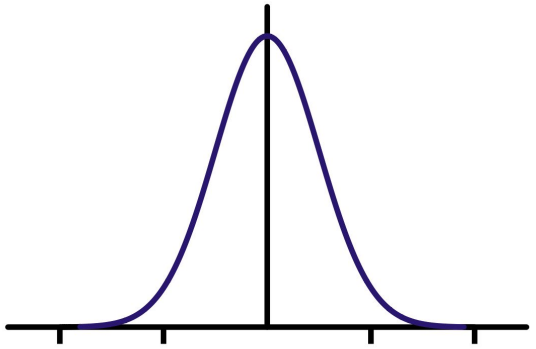
Source: D. Forsyth

Smoothing with different filters

- Mean smoothing

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

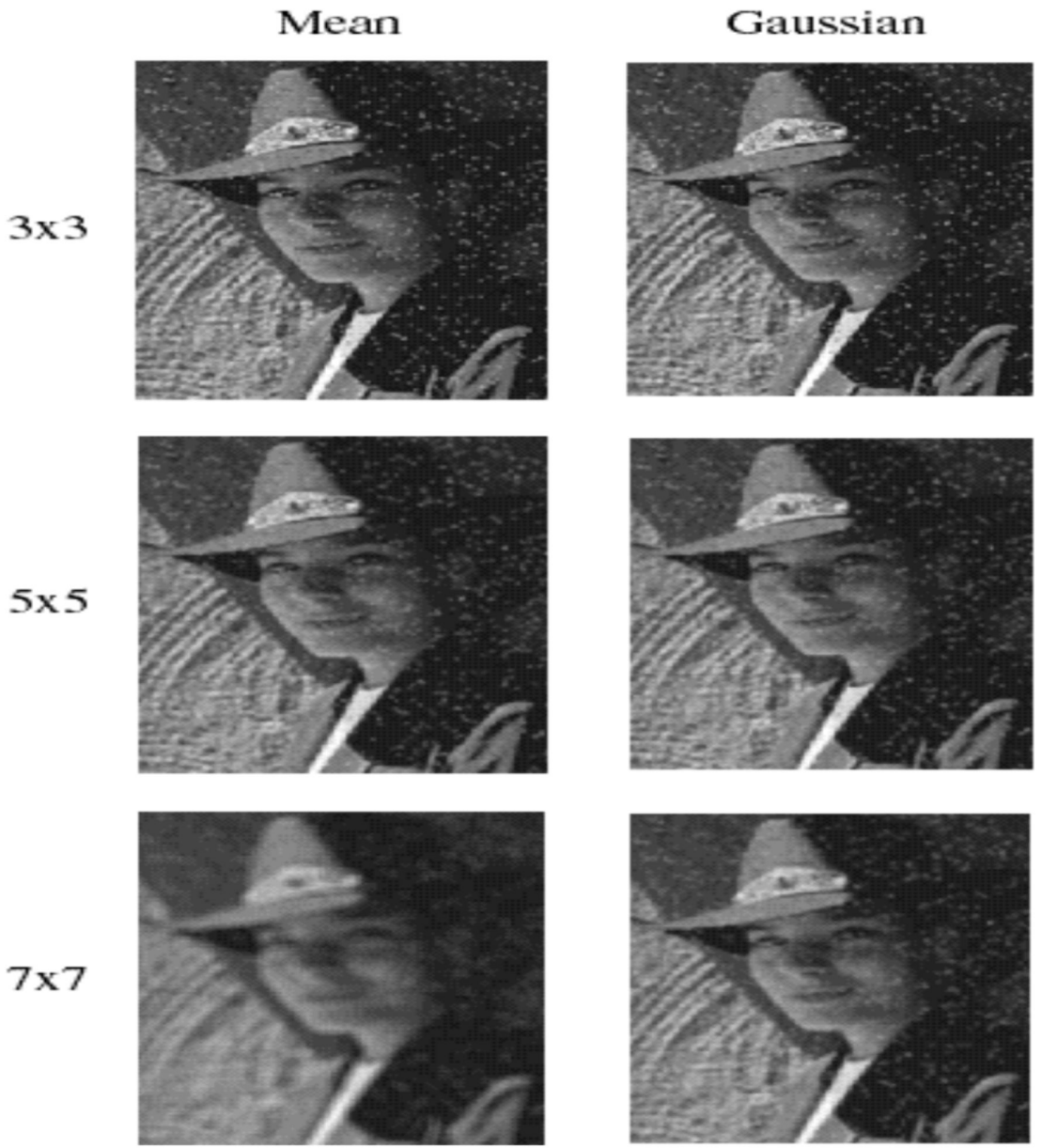
- Gaussian (smoothing * derivative)



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

Slide credit: Steve Seitz

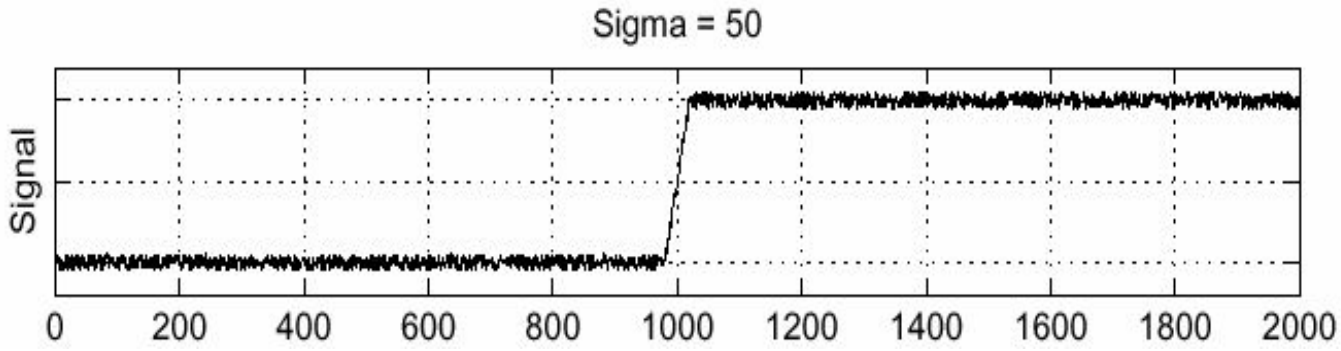
Smoothing with different filters



Solution: input function

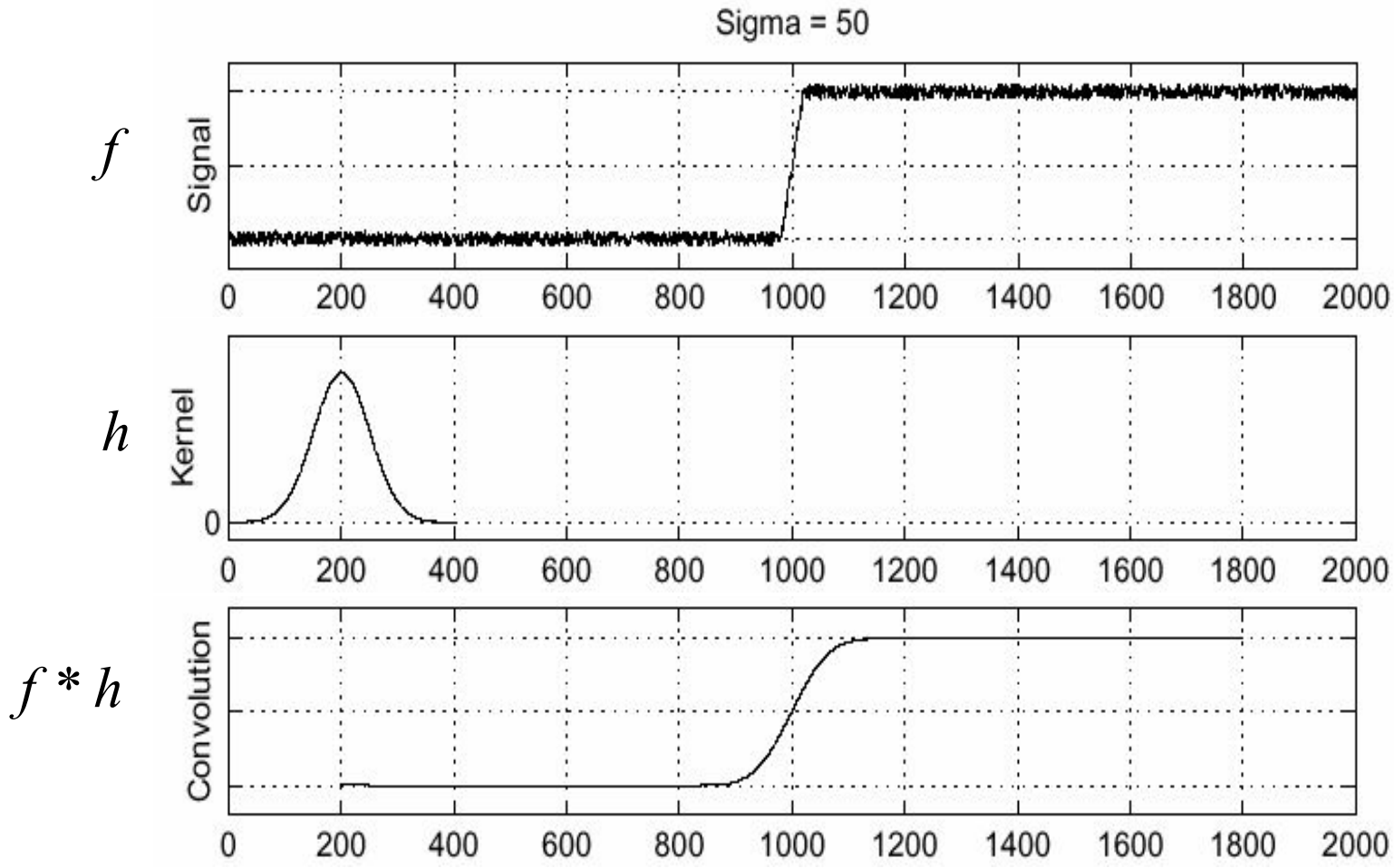
Let's look at a single image row:

f



Source: S. Seitz

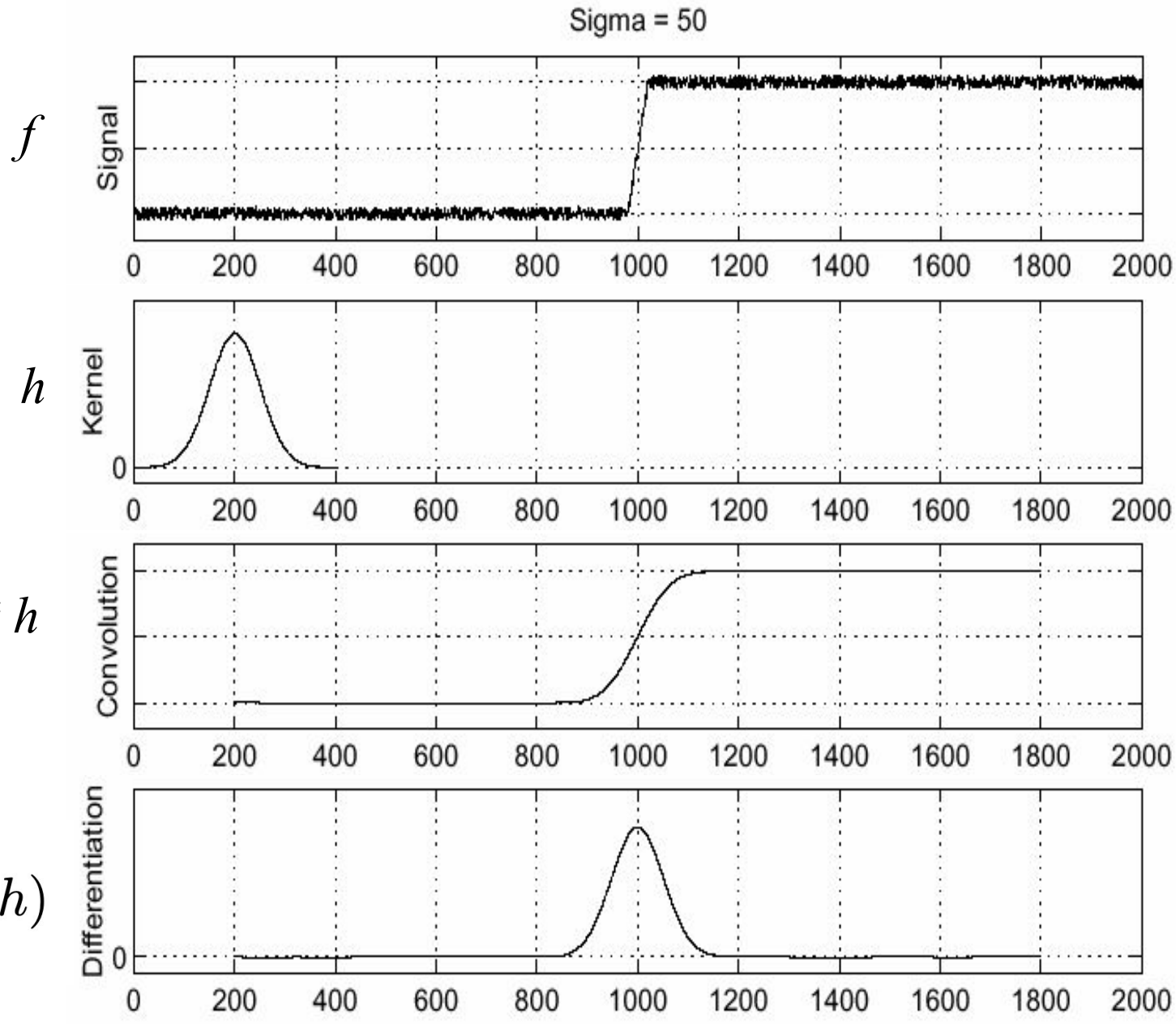
Solution: smooth first



Solution: smooth first

To find edges, look for peaks in

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

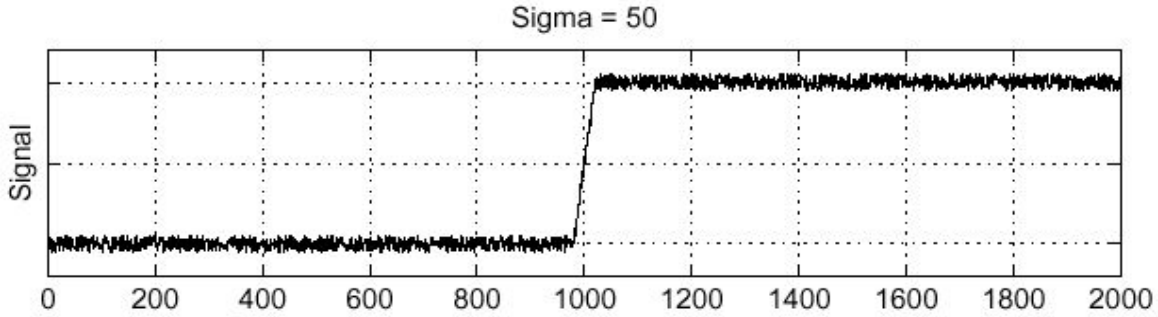


Derivative theorem of convolution

- This theorem gives us a very useful property:

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

f



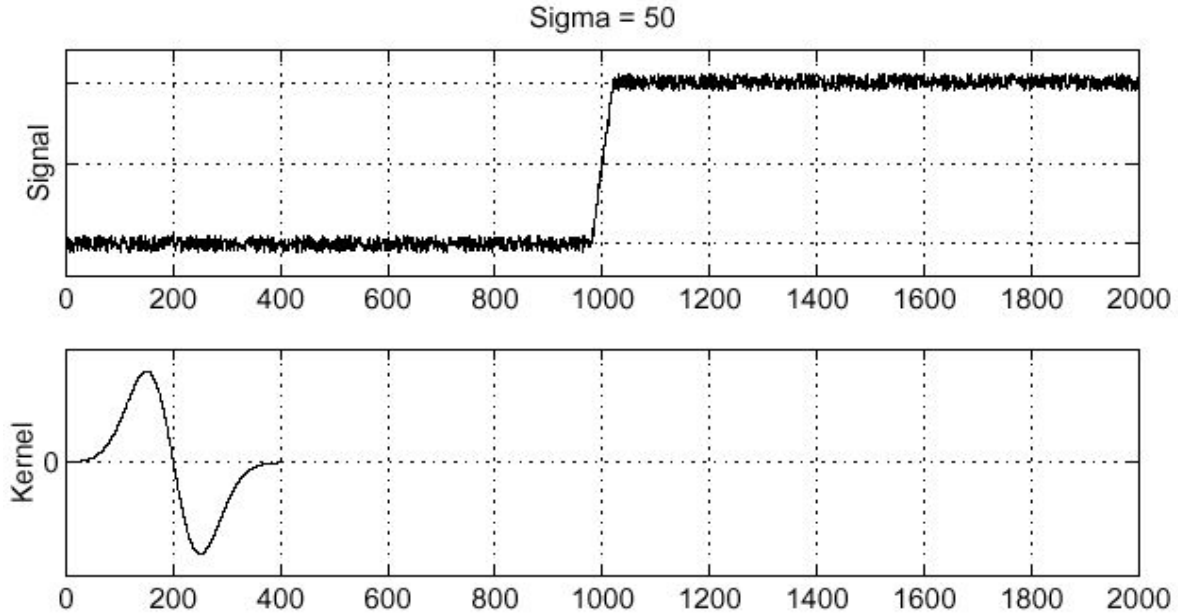
Derivative of a gaussian (DoG)

- This theorem gives us a very useful property:

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

f

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



Source: S. Seitz

Derivative of a gaussian (DoG)

- This theorem gives us a very useful property:

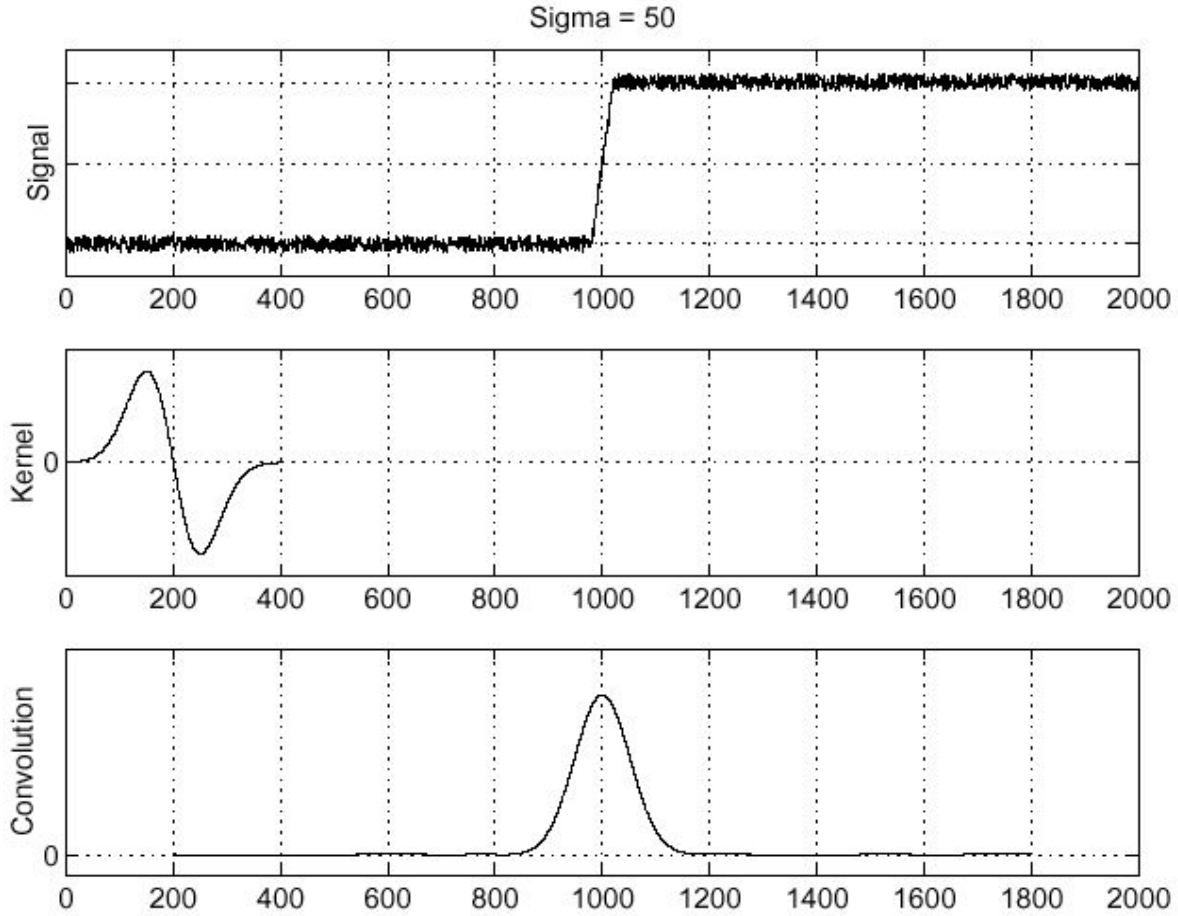
$$\frac{d}{dx}(f * h) = f * (\frac{d}{dx}h)$$

- This saves us one operation:

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

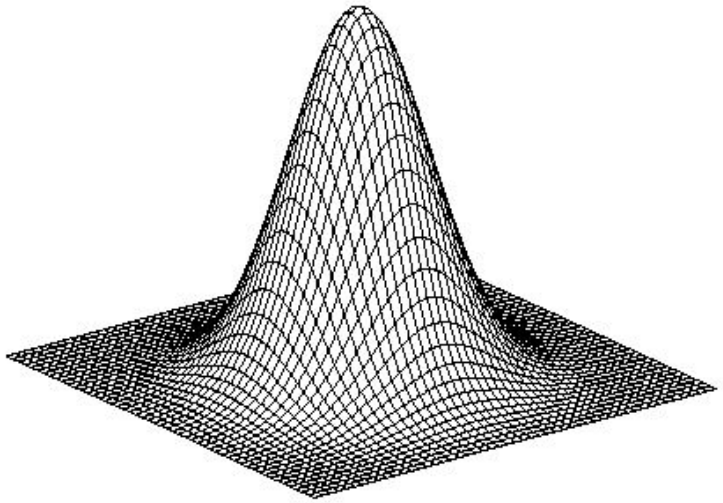
f

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

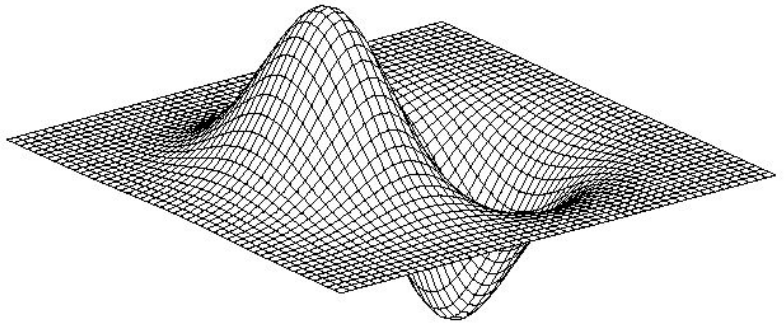


Source: S. Seitz

Derivative of Gaussian filter (central derivative)



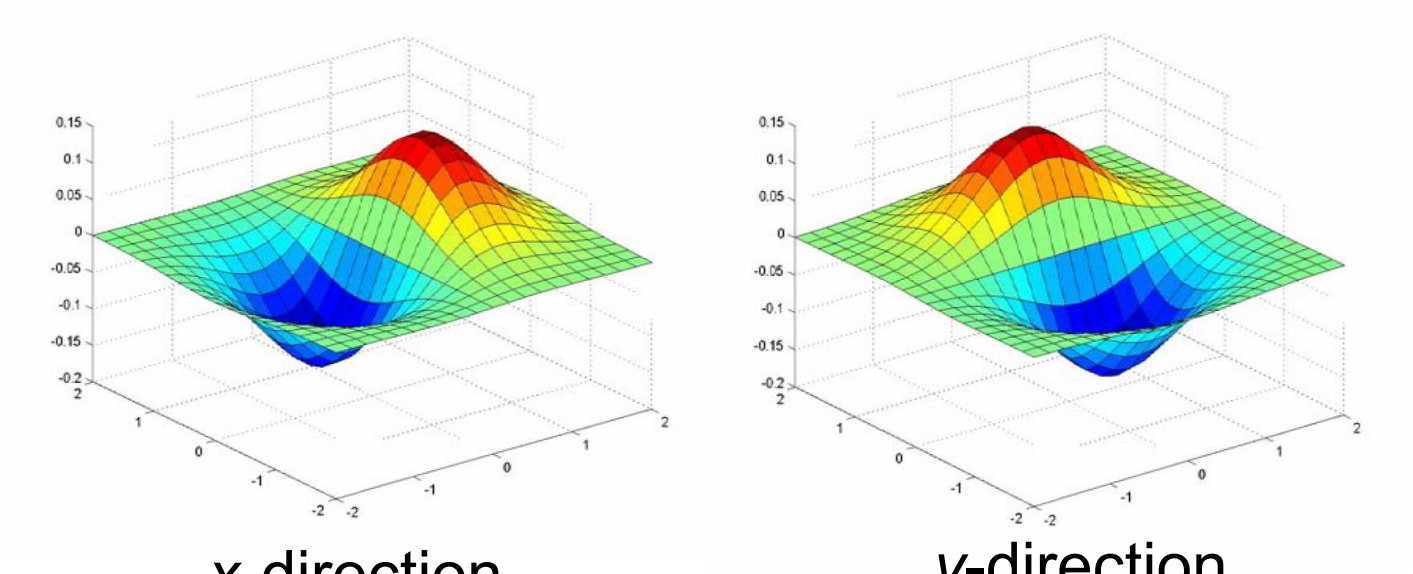
* [1 0 -1] =



2D-gaussian

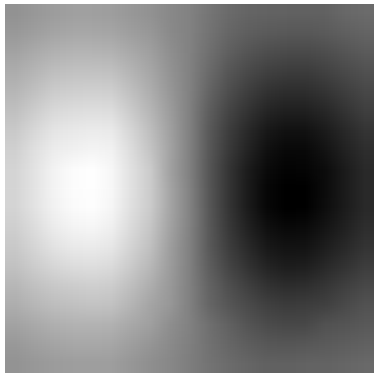
x - derivative

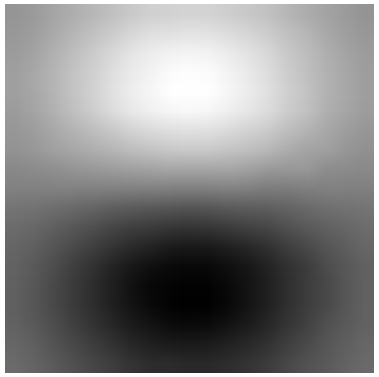
Derivative of Gaussian filter along x and y directions



x-direction

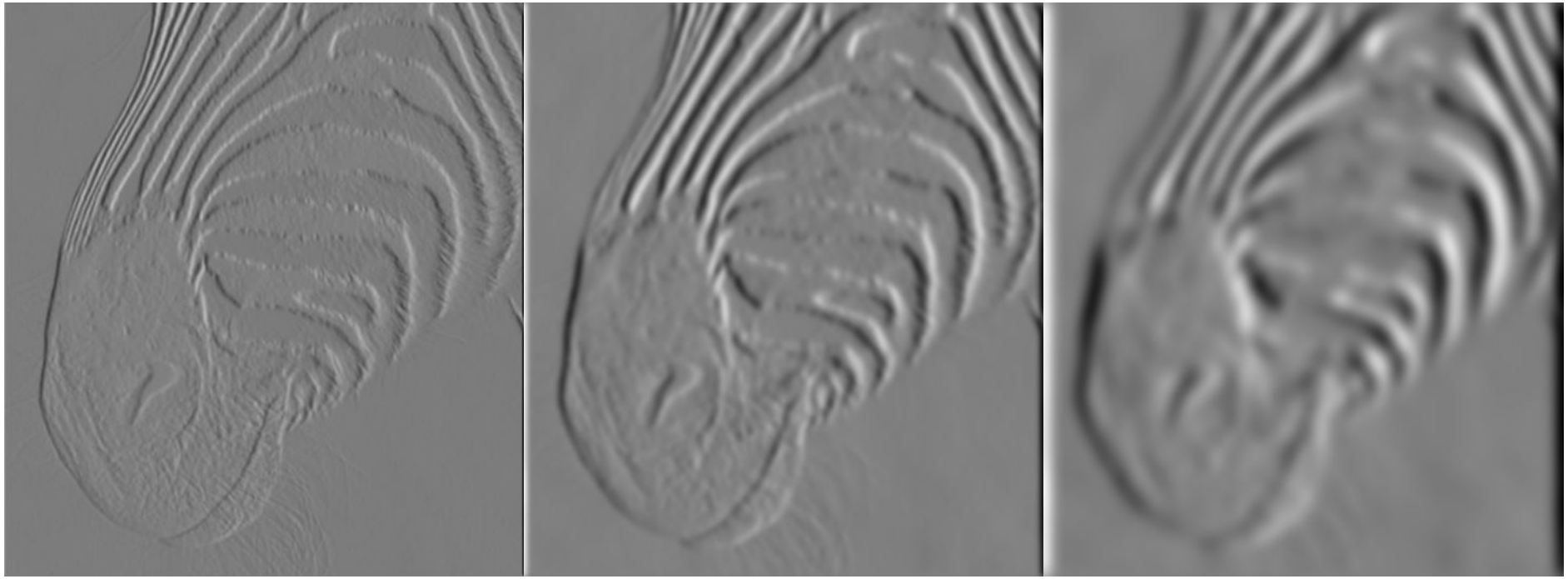
y-direction





Derivative of Gaussian filter

Tradeoff between smoothing at different scales

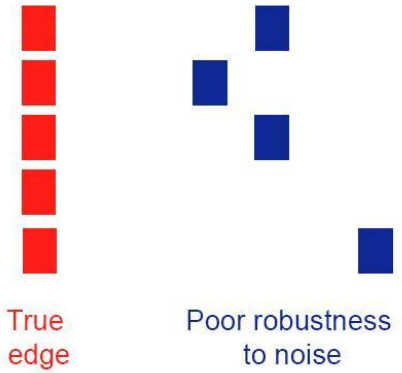


1 pixel 3 pixels 7 pixels

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

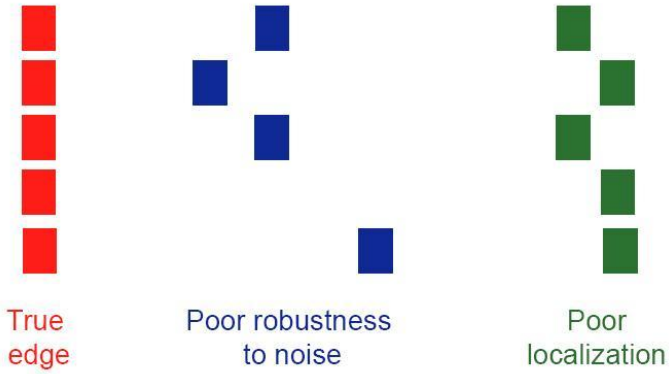
Designing an edge detector

- 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)



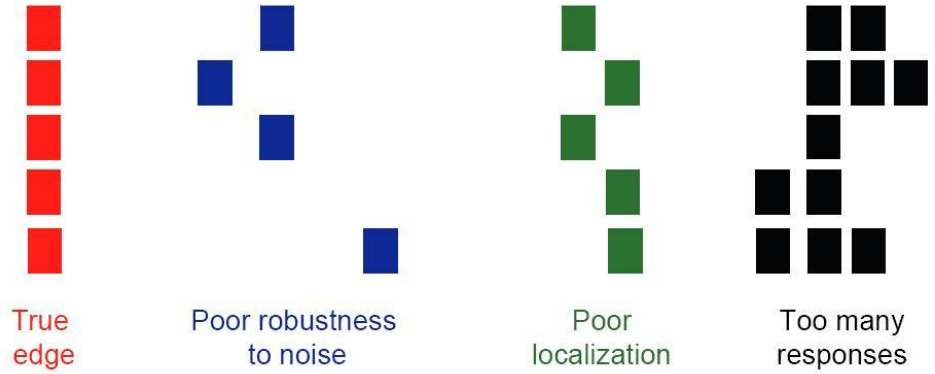
Designing an edge detector

- 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

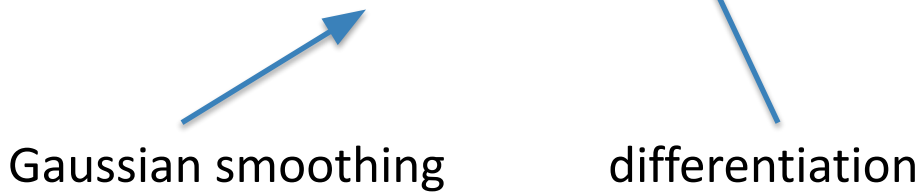
- uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives
- one for horizontal changes, and one for vertical

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

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:

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

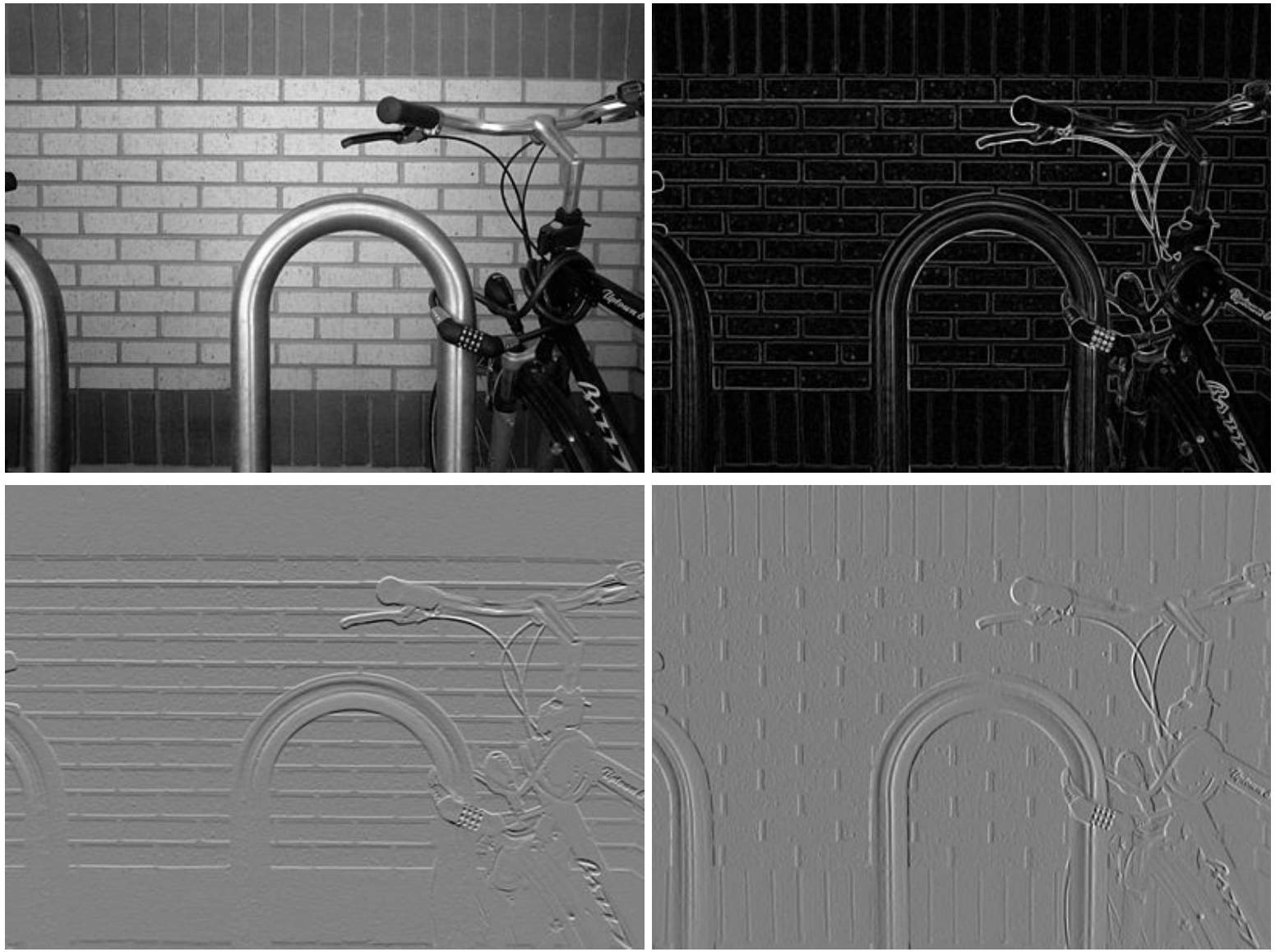
- Angle or direction of the gradient:

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

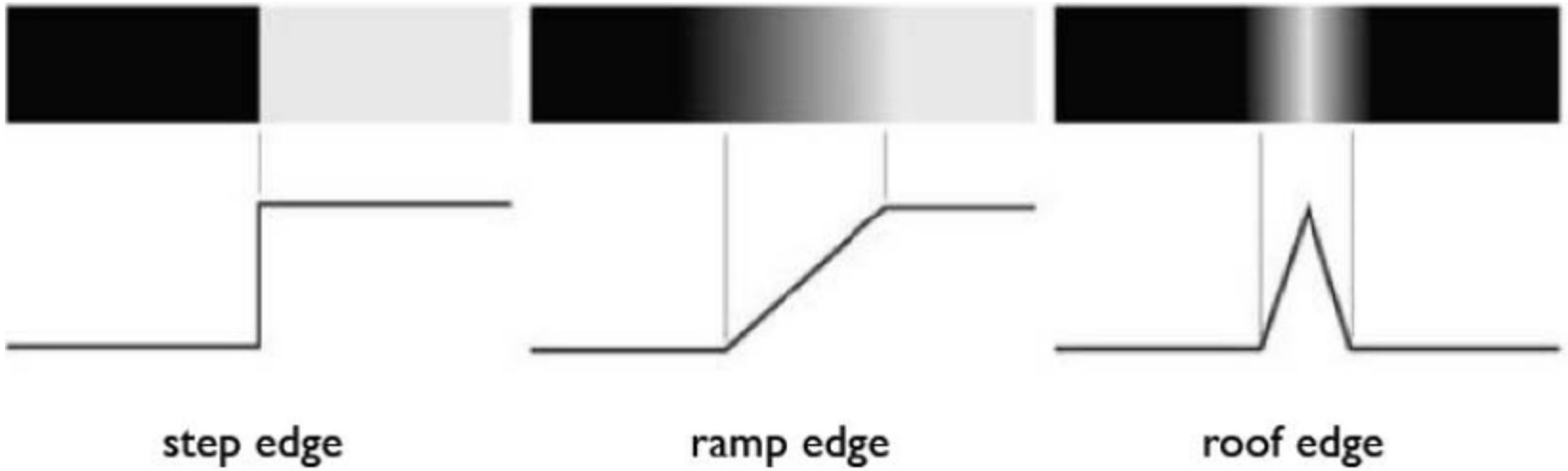
Sobel Filter example

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

Step 2: Threshold the values to generate a binary image



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:

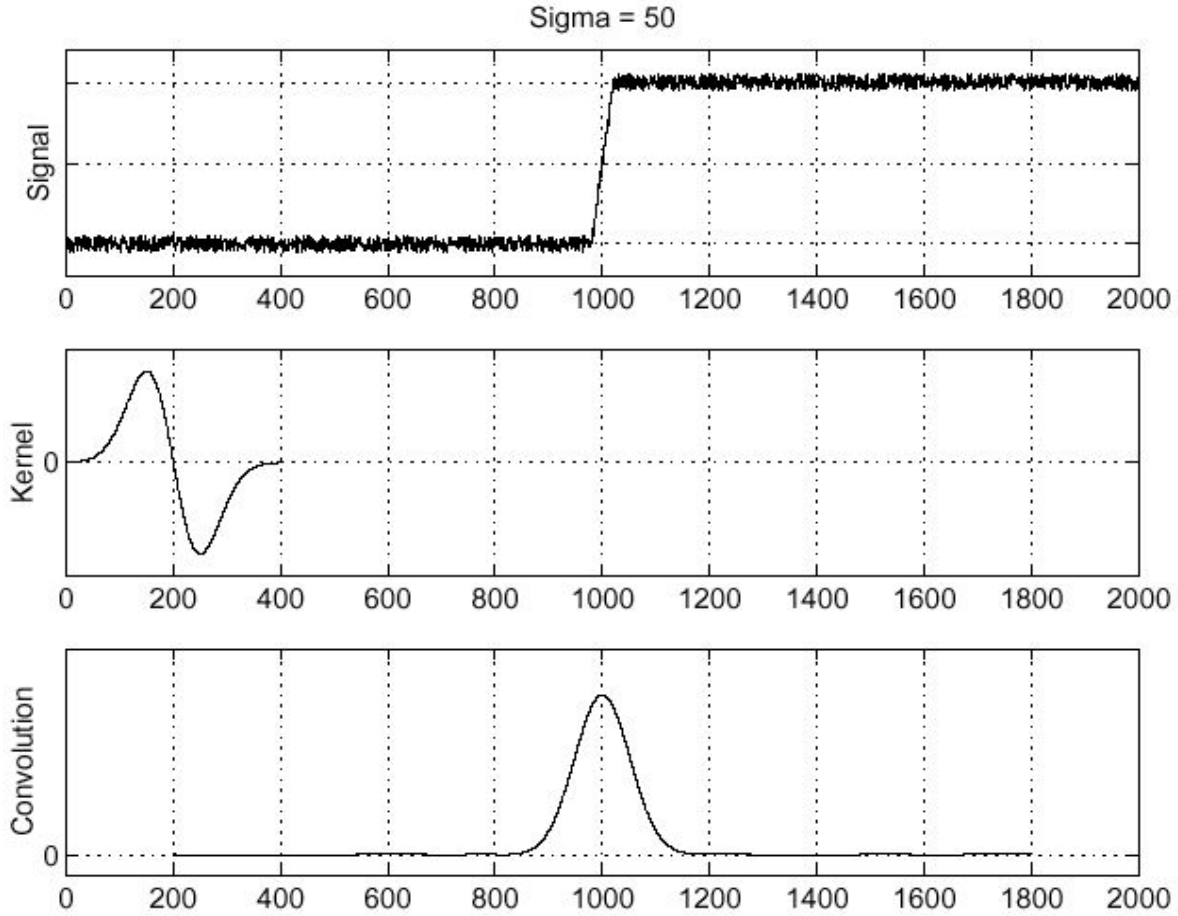
$$\frac{d}{dx}(f * h) = f * (\frac{d}{dx}h)$$

- This saves us one operation:

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

f

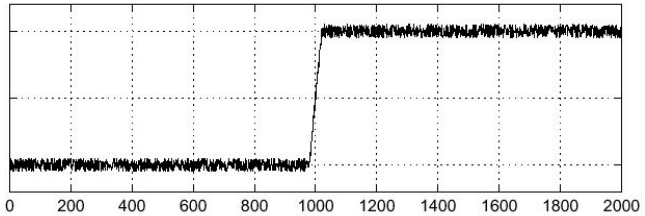
$f * (\frac{d}{dx}h)$



Source: S. Seitz

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

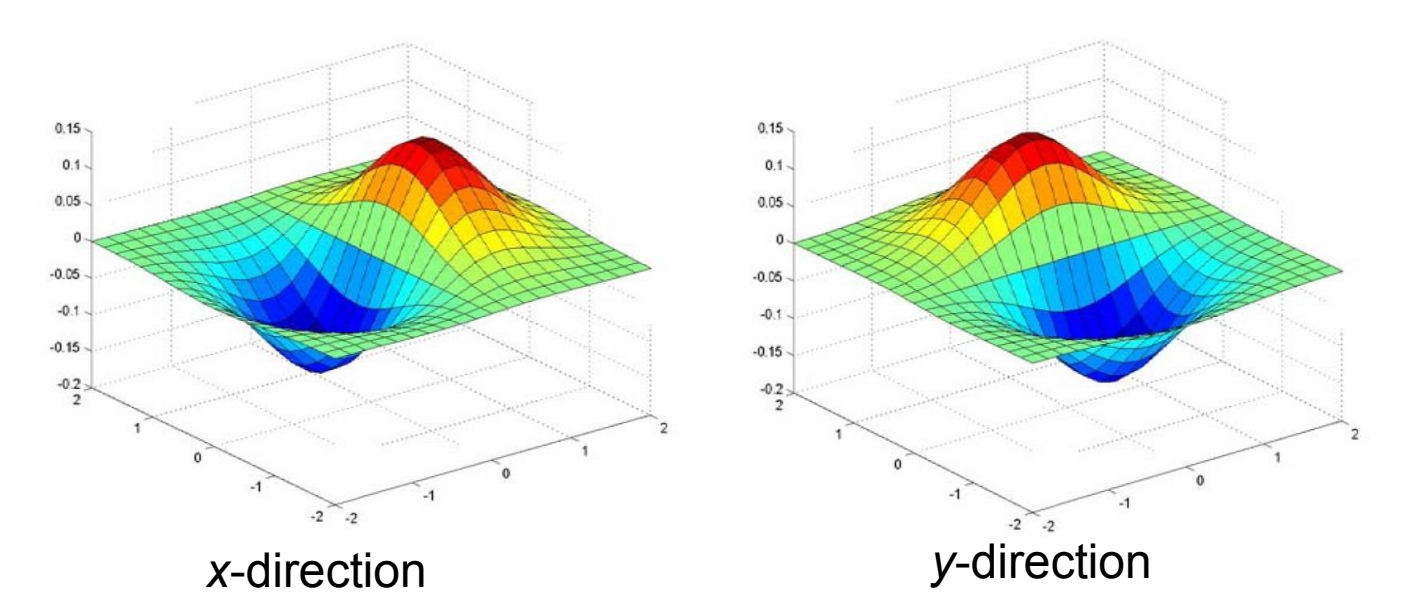
Example

- 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

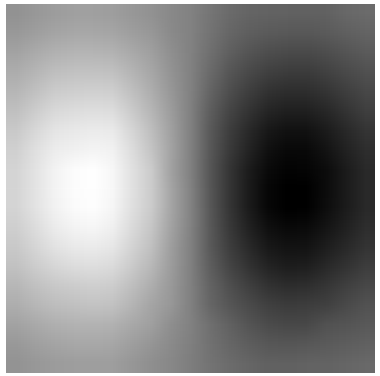
Derivative of Gaussian filter

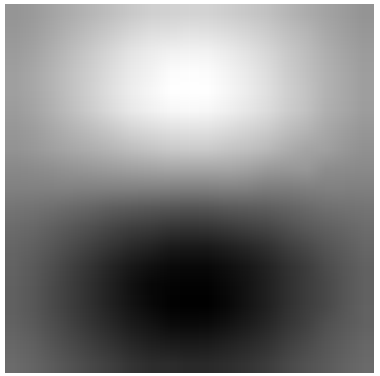


x-direction

y-direction

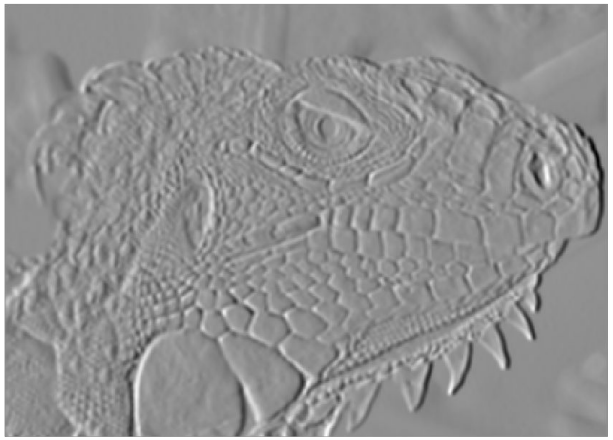
$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$



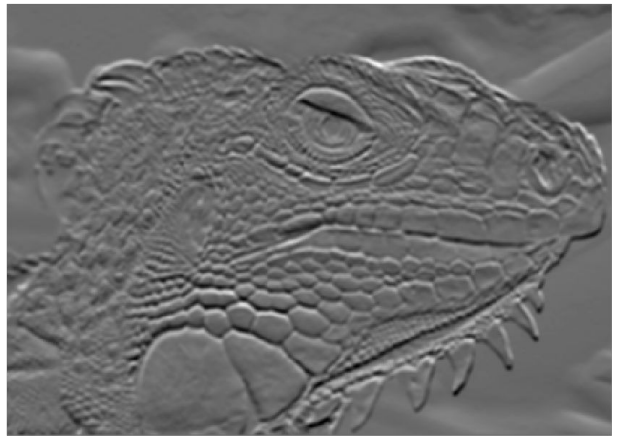


$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Compute gradients (DoG)



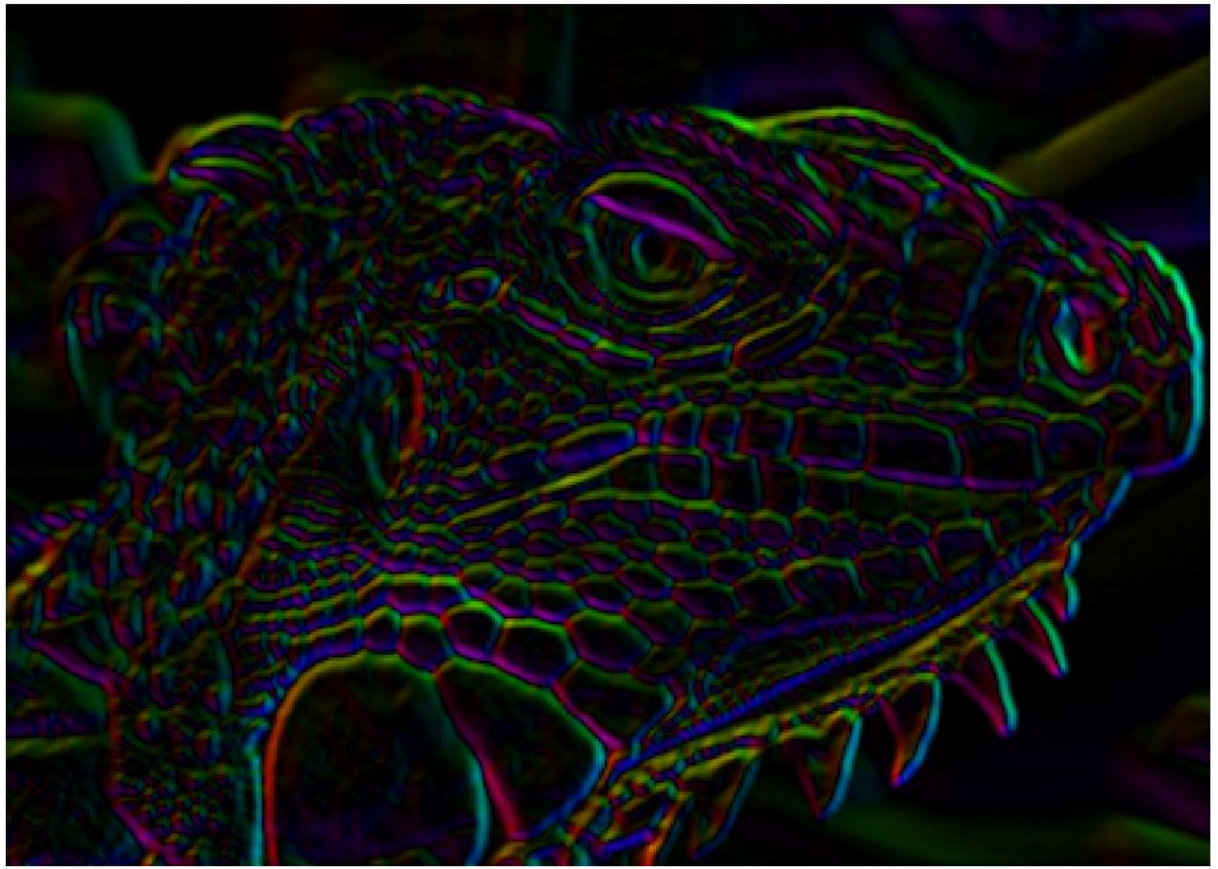
X-Derivative of Gaussian



Y-Derivative of Gaussian

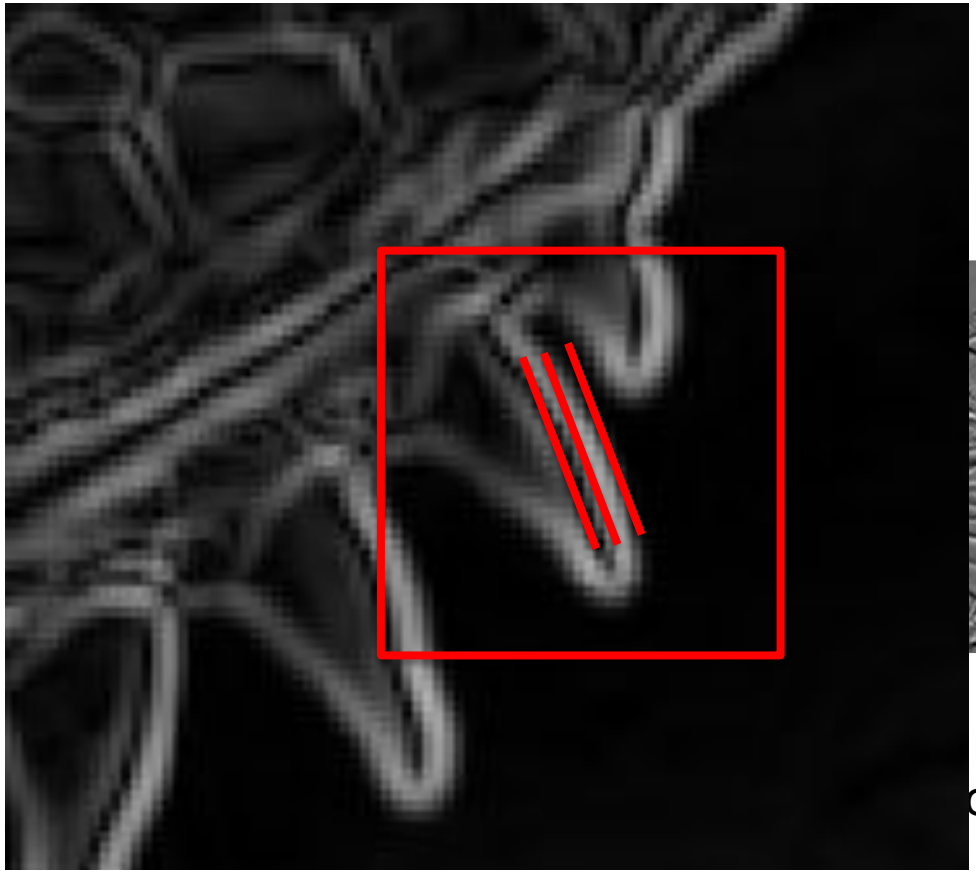
Gradient Magnitude

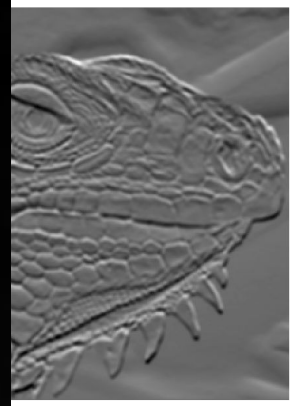
Get orientation at each pixel



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

Compute gradients (DoG)





of Gaussian

Gradient Magnitude

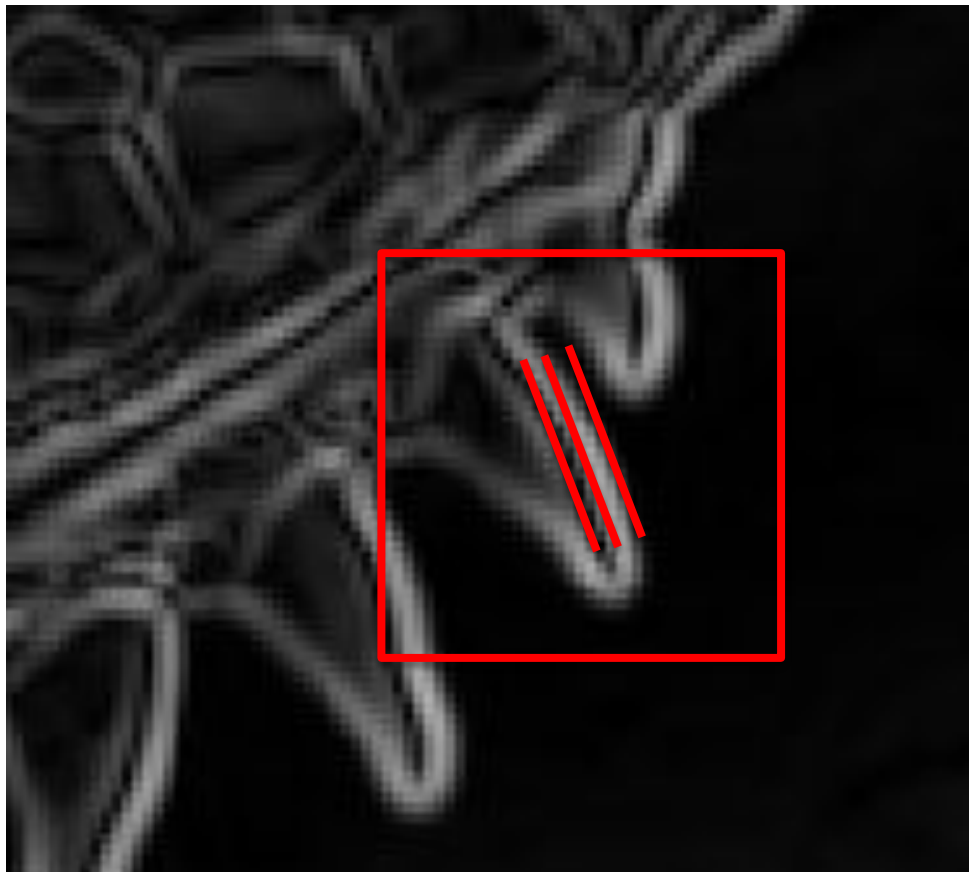
Canny edge detector

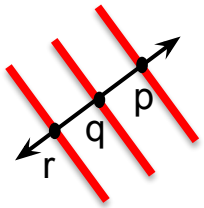
1. Suppress Noise
2. Compute gradient magnitude and direction
3. Apply Non-Maximum Suppression
    - Assures minimal response
4. Use hysteresis and connectivity analysis to detect edges

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,

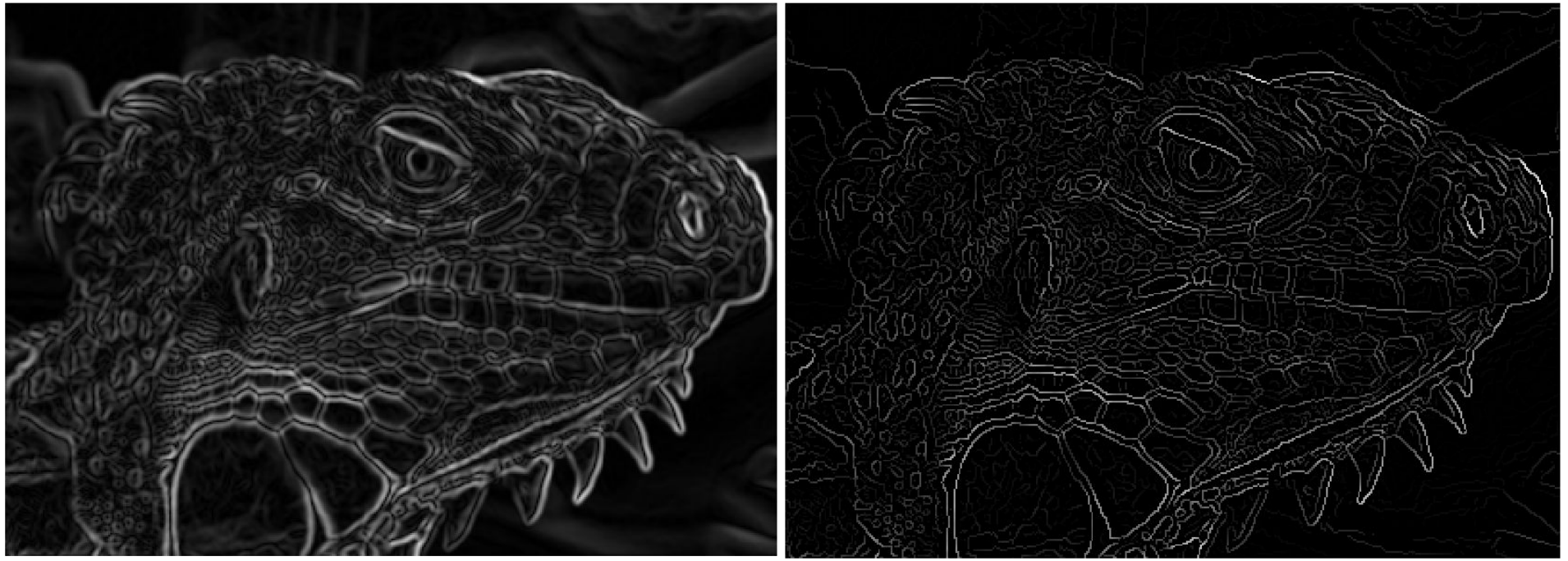
q,

r

q has the largest gradient.

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

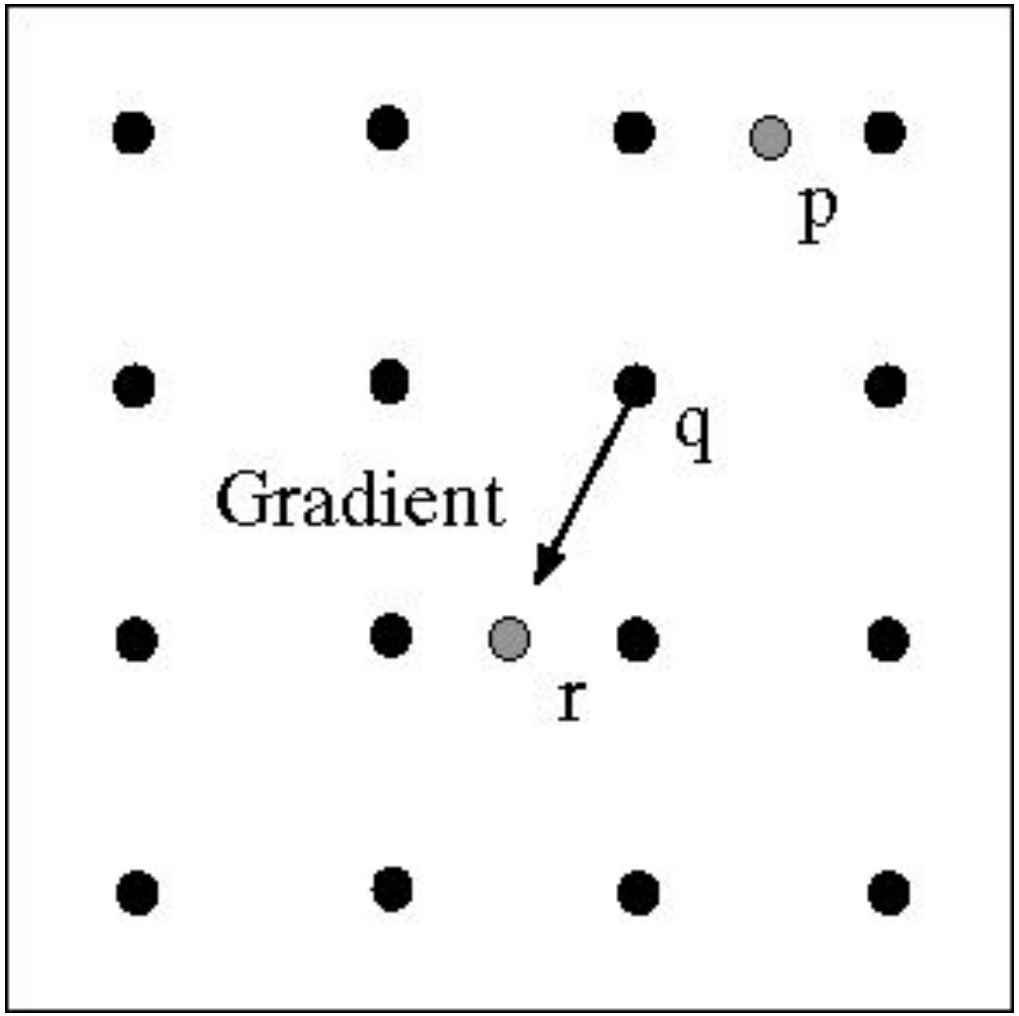
What the output looks like after Non-max Suppression



Before

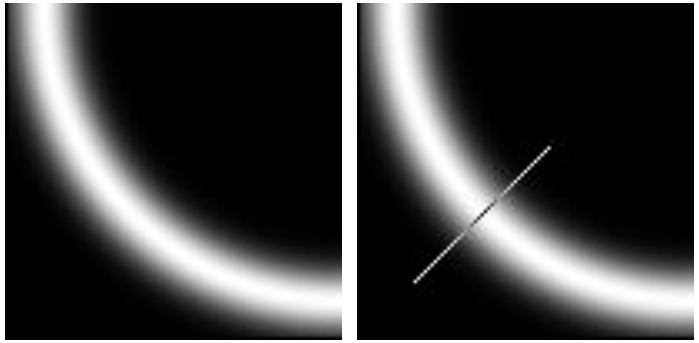
After

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

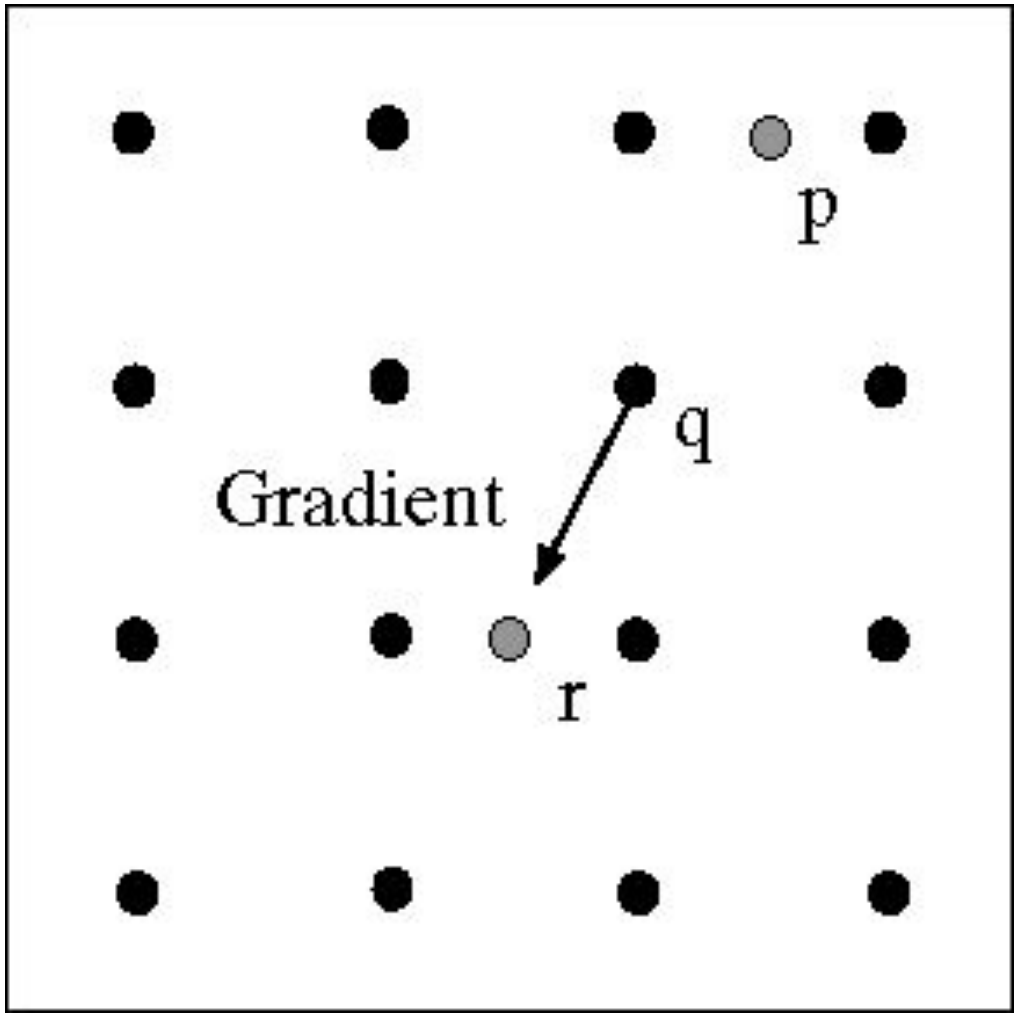


q is a maximum if the value is larger than those at both p and at r.

How should we calculate magnitude at p and r?



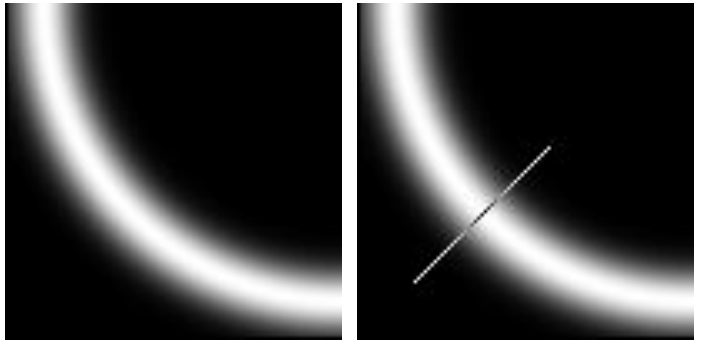
What if $p = [n_1, m_1]$ **or** $r = [n_2, m_2]$, is not a pixel location



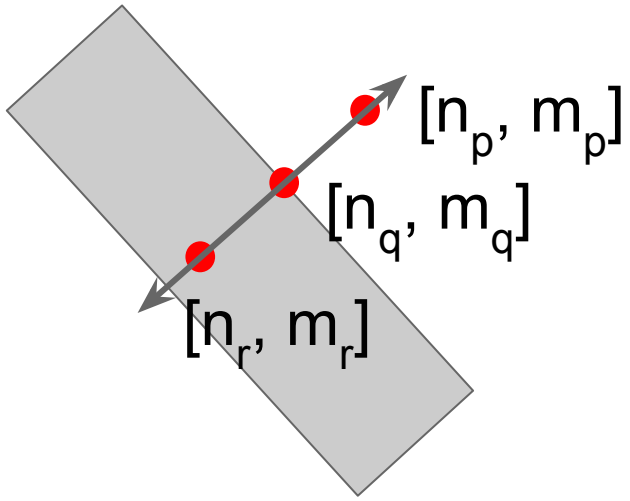
q is a maximum if the value is larger than those at both p and at r.

How should we calculate magnitude at p and r?

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



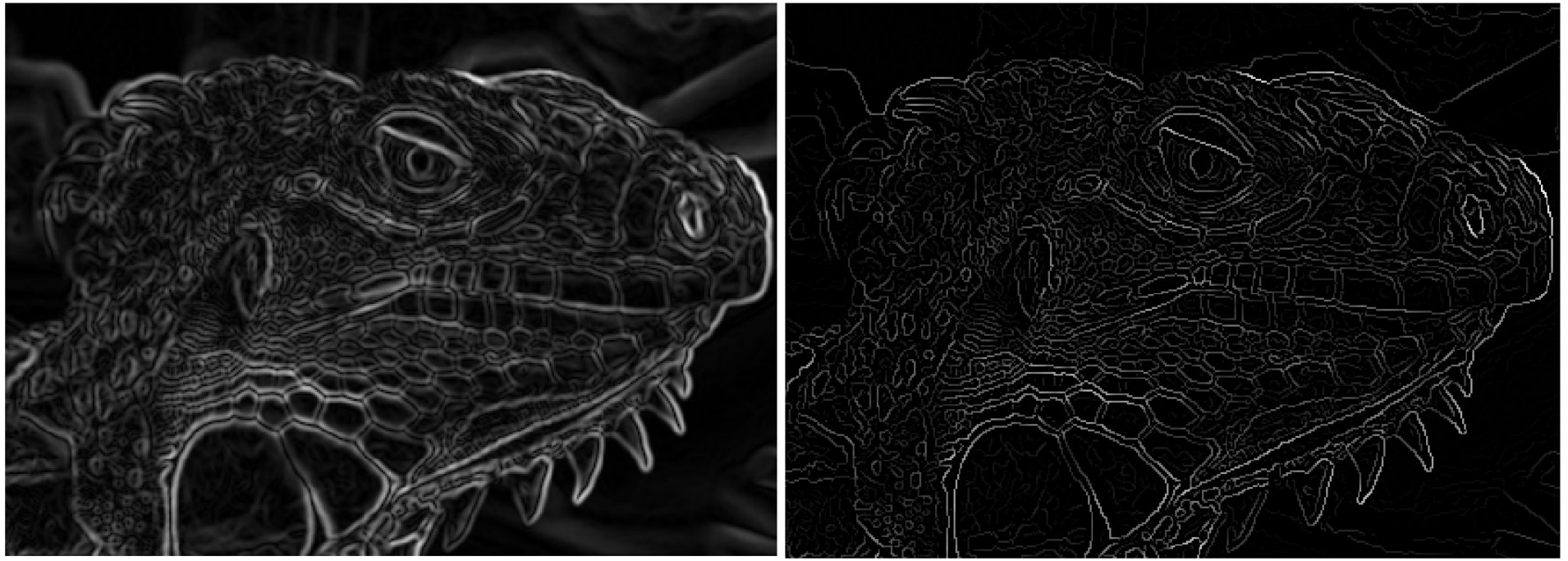
In code, you will calculate gradient magnitudes at every q and set it to zero if it is not the local max



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

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

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**

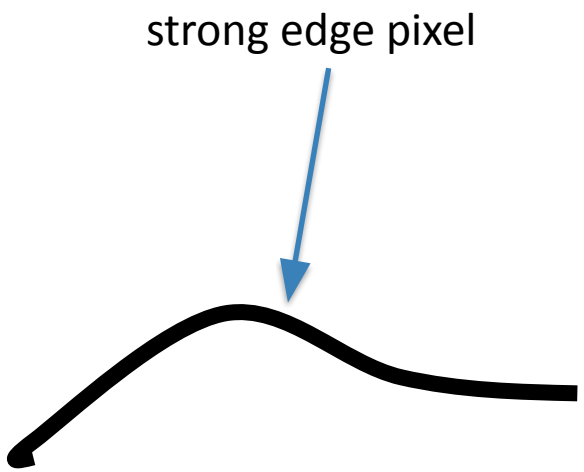
  - If between Low and High => **weak edge**

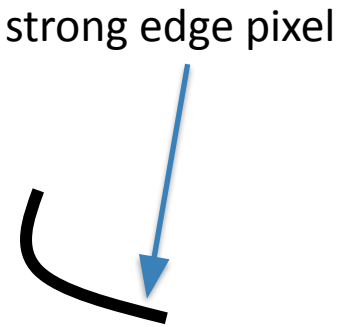
Hysteresis thresholding

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

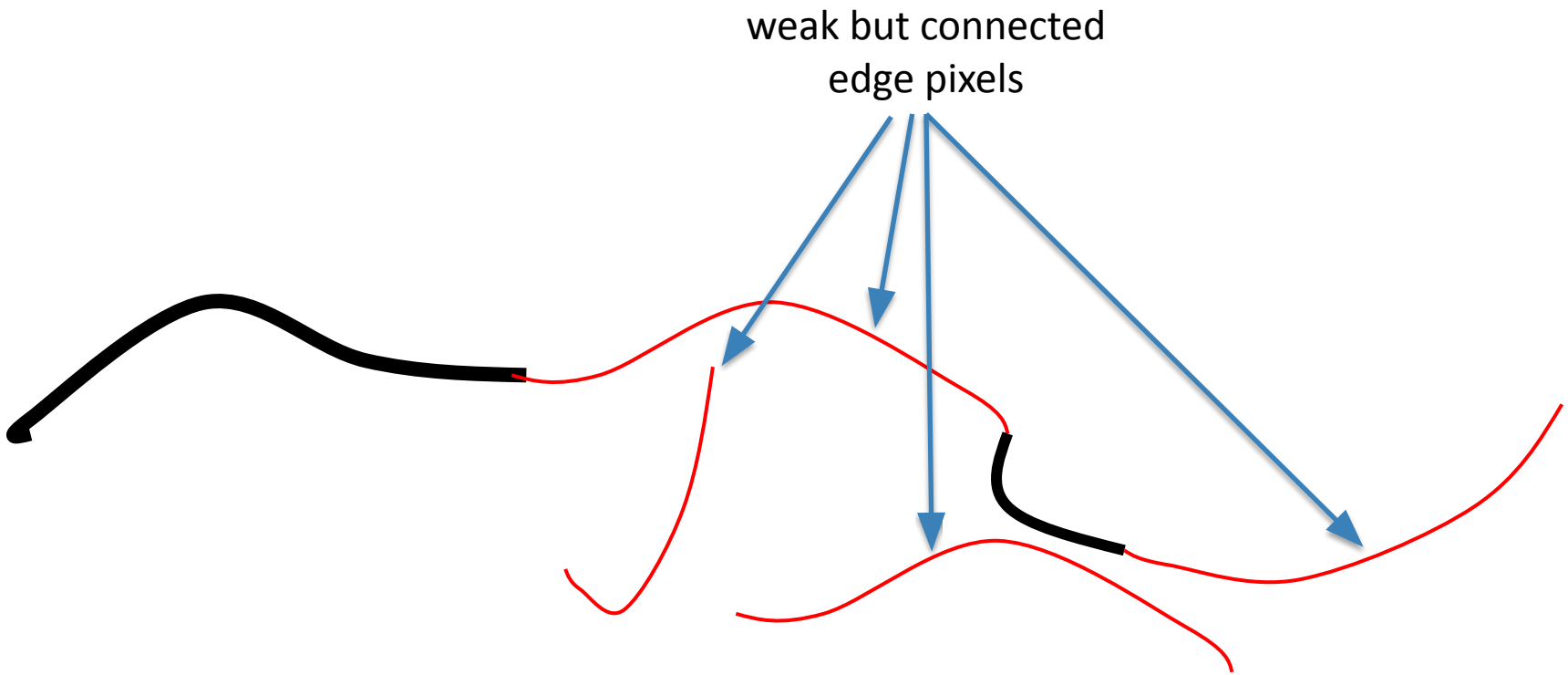
- Consider its neighbors iteratively then declare it an “edge pixel” if it is connected to an ‘strong edge pixel’ directly or via pixels between Low and High

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

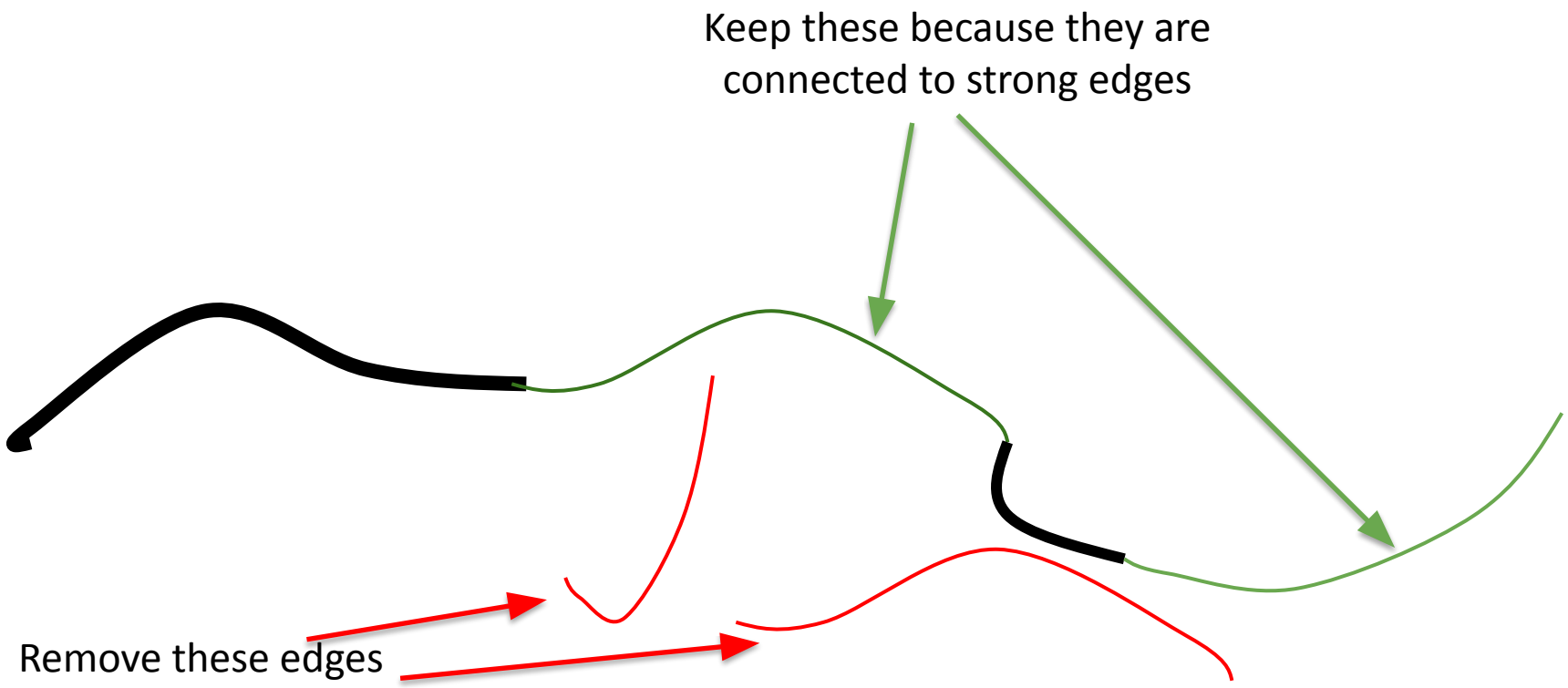




Now, let's assume all the red pixels are weak edges (between low and high thresholds)



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 $\sigma = 2$

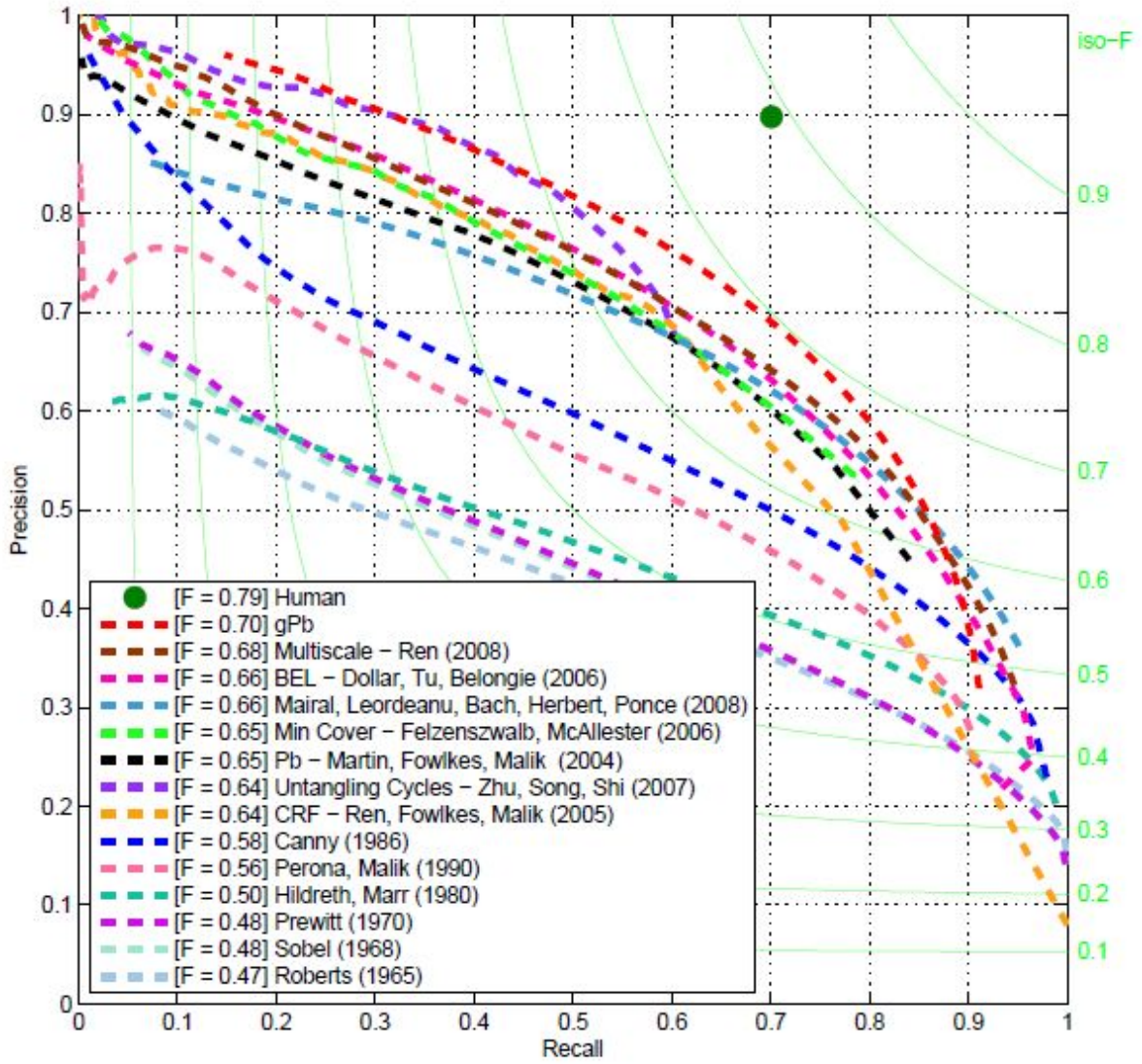
The choice of σ depends on desired behavior

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

Gradients (e.g. Canny)

Human

45 years of edge detection



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 Transform edge detections into lines

Original image

Edge image

Hough transform

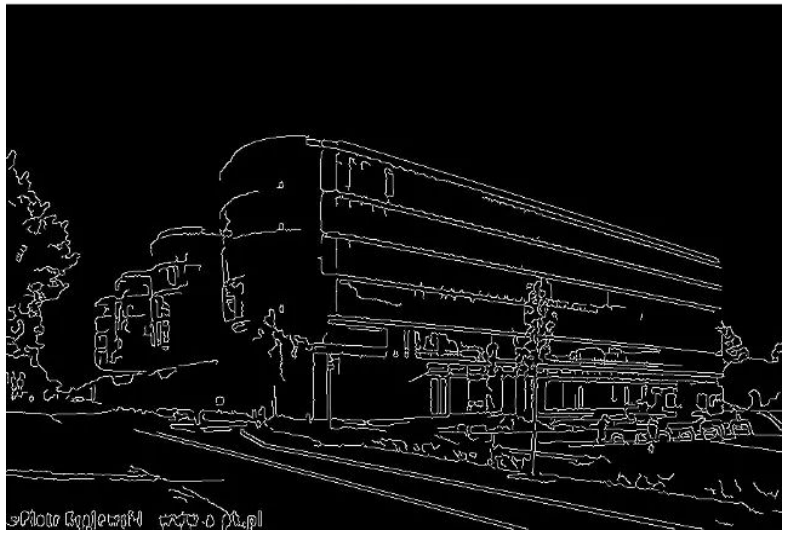
- It was introduced in 1962 (Hough 1962) and first used to find lines in images a decade later (Duda 1972).

- **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]

Detecting lines using Hough transform

- Consider a line that passes through two points in the image
  - (x_1, y_1) and (x_2, y_2)
- Straight lines that pass that point have the form:

$$y = a*x + b$$

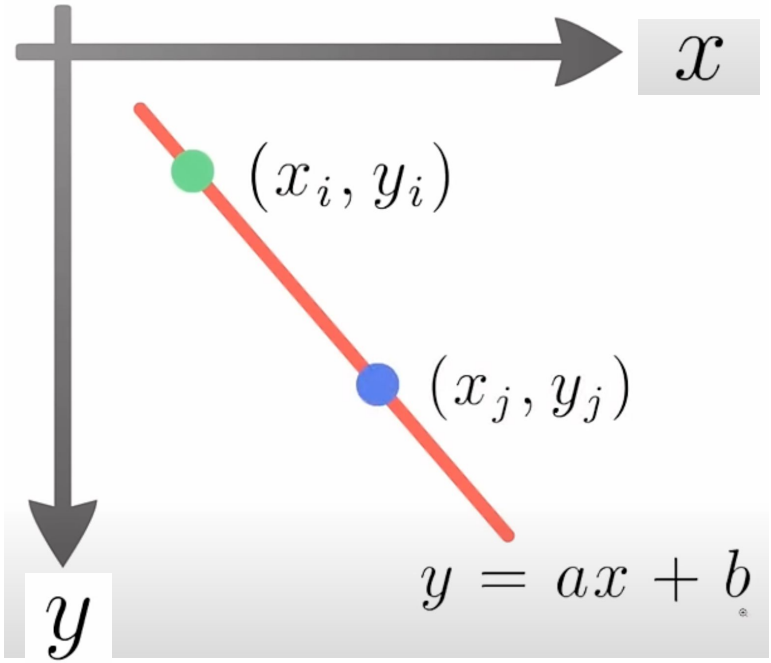
- How do we calculate the parameters (a, b)?

$$a = (y_2 - y_1) / (x_2 - x_1)$$
$$b = y_1 - a \times x_1$$

Detecting lines using Hough transform

- Consider a line that passes through two points in the image
  - (x_1, y_1) and (x_2, y_2)
- Straight lines that pass that point have the form:

$$y = a*x + b$$



- How do we calculate the parameters (a, b)?

$$a = (y_2 - y_1) / (x_2 - x_1)$$
$$b = y_1 - a \times x_1$$

Detecting lines using Hough transform

- Problem: We don’t know which pairs of edge points belong to the same line.

- That’s where Hough transform comes in!

The Hough transform

- Consider a line that passes through a single point in the image
  - (x_i, y_i)

- All straight lines that pass that point have the form:

$$y_i = a*x_i + b$$

The Hough transform

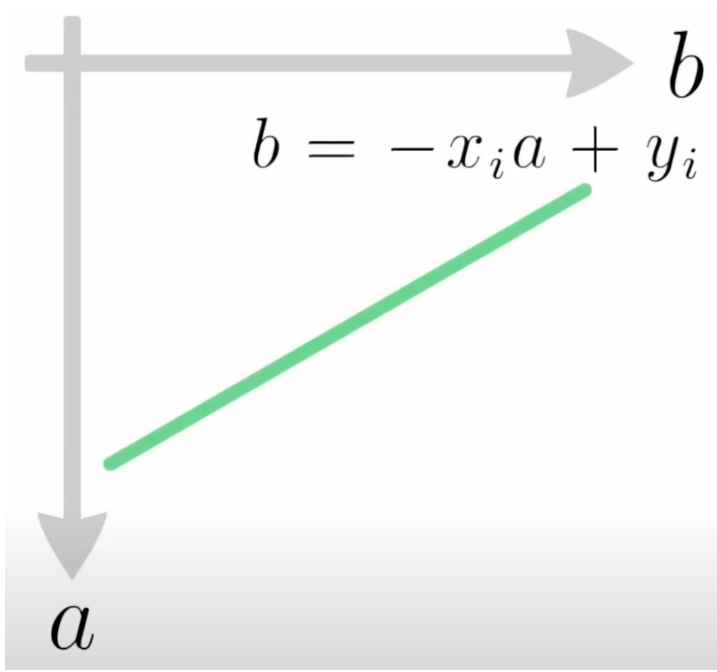
$$y_i = a*x_i + b$$

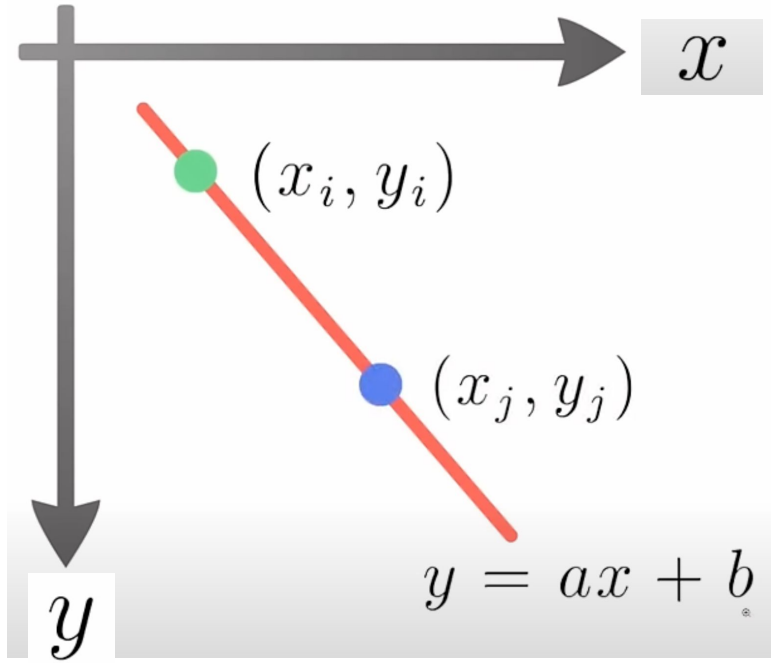
- This equation can be rewritten as follows:
  - $b = -a*x_i + y_i$

The Hough transform

$y_i = a*x_i + b$

- This equation can be rewritten as follows:
  - $b = -a*x_i + y_i$
  - We can now consider x and y as parameters
  - a and b as coordinates.

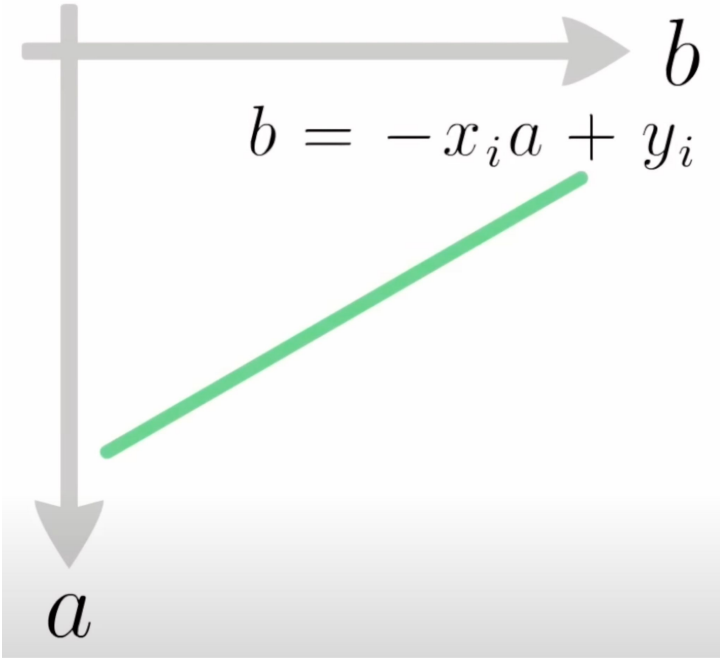




The Hough transform

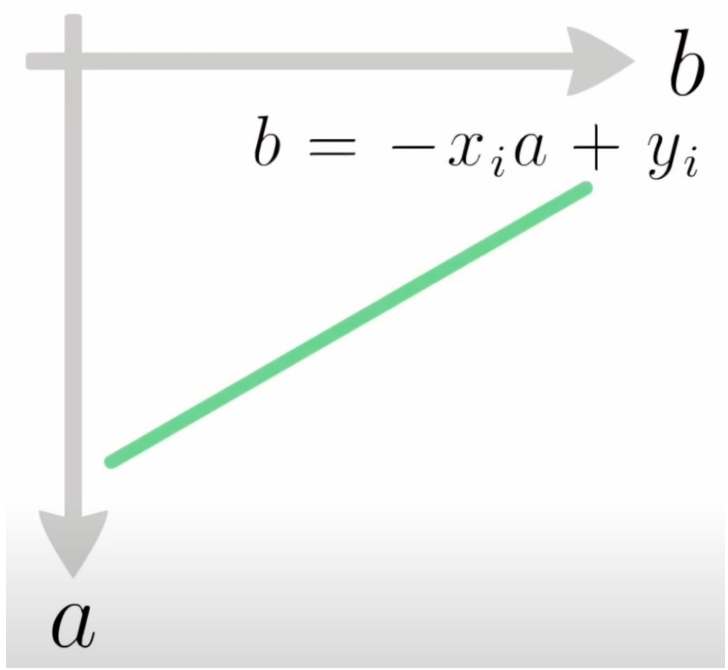
$y_i = a^*x_i + b$

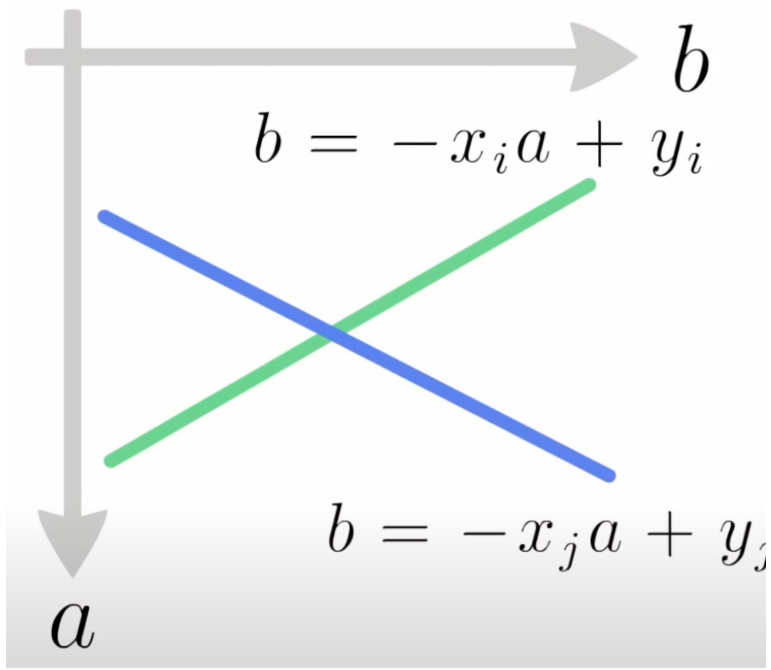
- $b = -a^*x_i + y_i$
- If our coordinates were (a,b) instead of (x, y):
  - We could say the above equation is a line in (a,b)-space
  - parameterized by x and y.
  - So: one point (x_i,y_i) gives a line in (a,b) space.



The Hough transform

- So: one point (x_i,y_i) gives a line in (a,b) space.
- Another point (x_j,y_j) will give rise to another line in (a,b)-space.



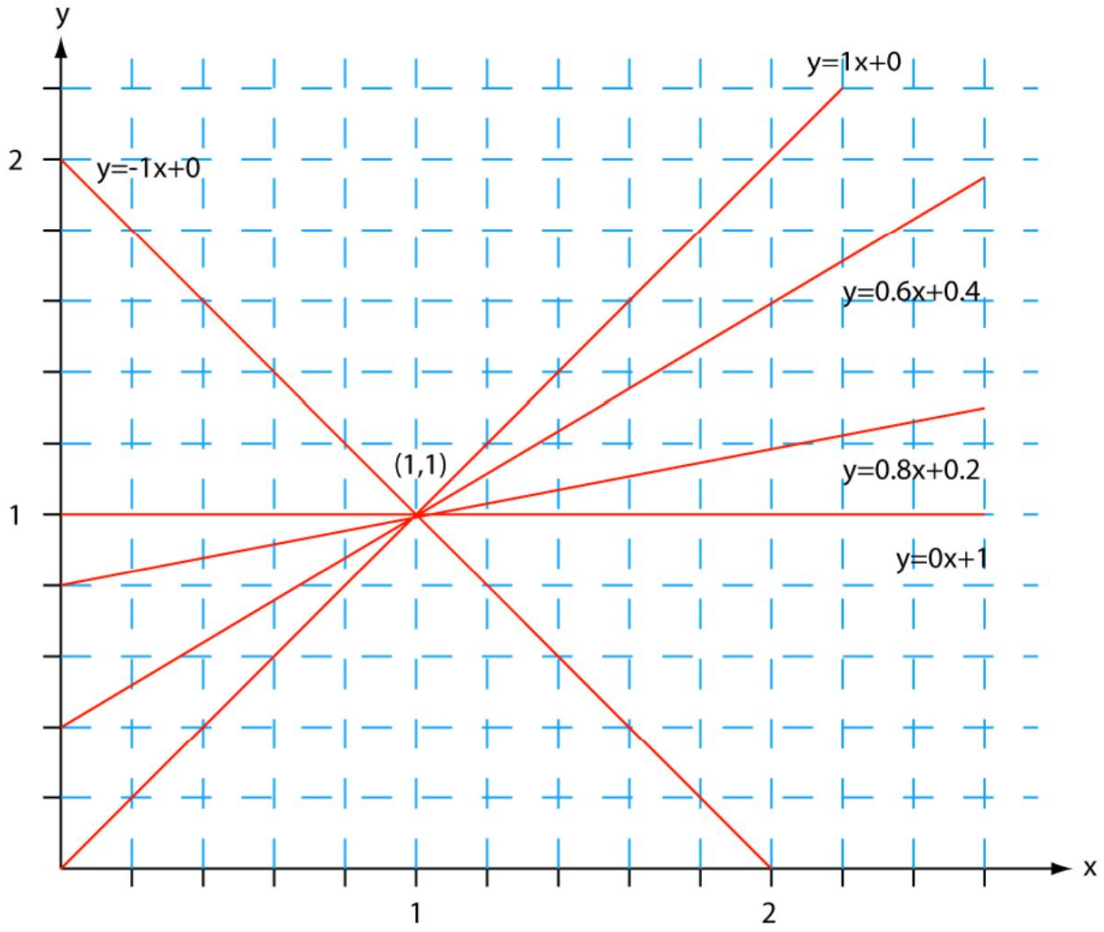


The Hough transform

- Doing this for 6 edge points will result in an graph like the one on the right.

- In (a,b) space these lines will intersect in a point **(a', b')**
  - On the right, a' = 1, b' = 1

- All points on the line defined by (x_i, y_i) and (x_j , y_j) in (x, y)-space will parameterize lines that intersect in (a', b') in (a,b) space.



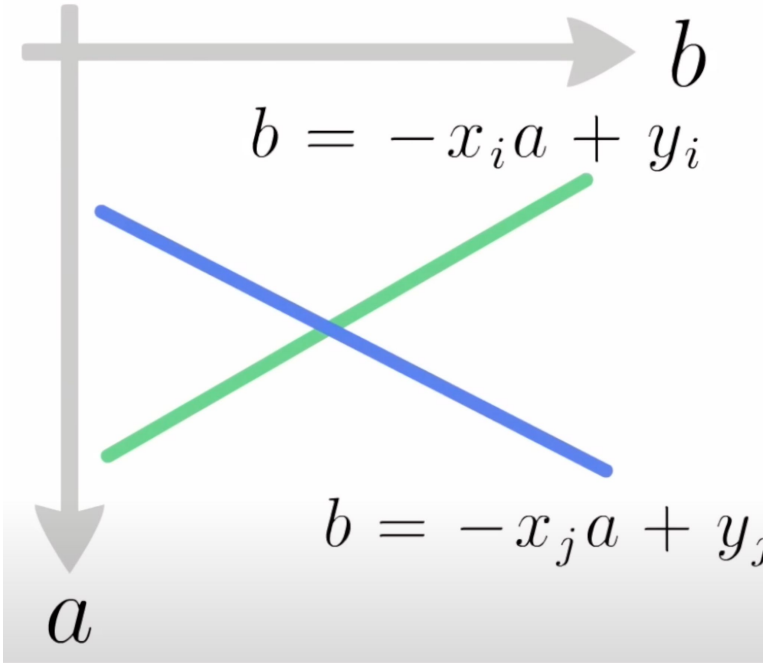
The need to quantize and “vote”

Not all intersections will be valid lines.

Consider two edge points that are not part of a real edge:

- They might still intersect in (a, b) space.

Problem: How do we identify intersections that are belong to the same edge versus random points?



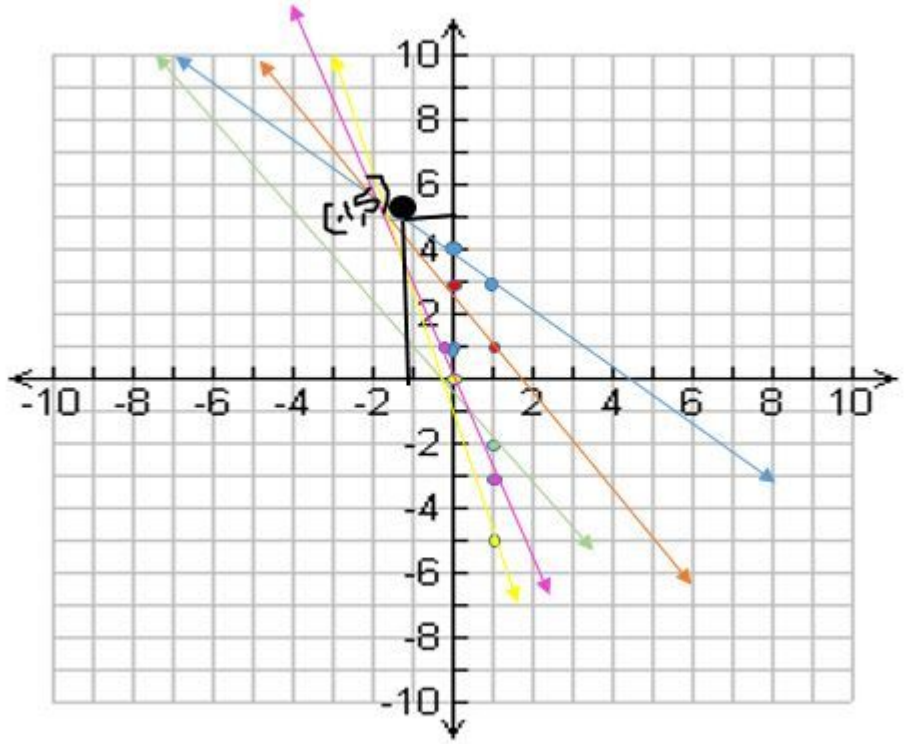
Intuition behind voting

The more lines intersect at the same (a', b') point, the more likely y=a'x + b' is a real edge in the image.

So, we need to count how many lines intersect at a point and keep the ones with high count

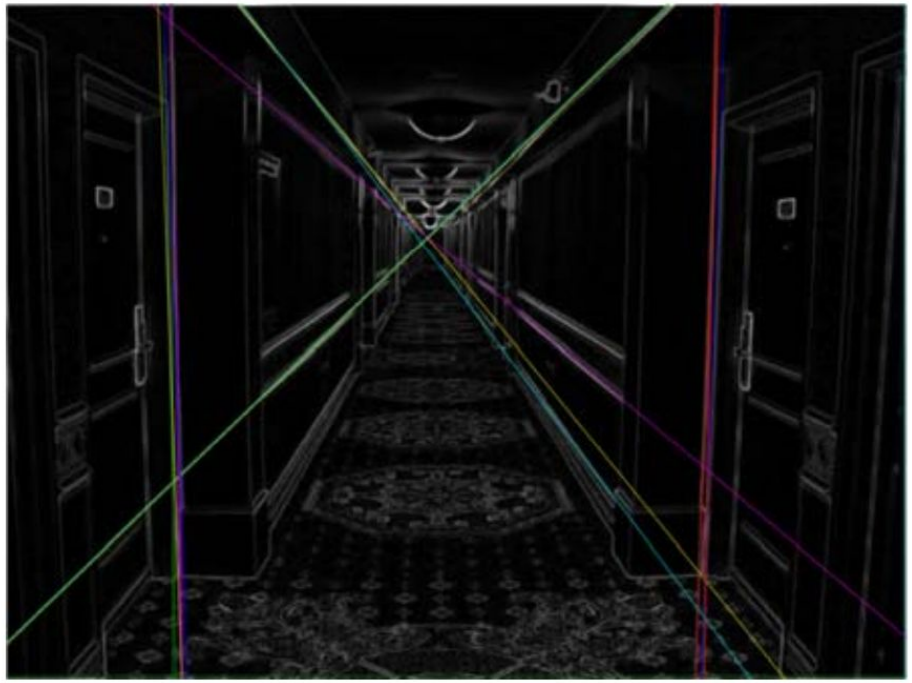
Counting in quantized (a, b)-space

1. Quantize the parameter space (a b) by dividing it into cells
   a. $[[a_{min}, a_{max}],[b_{min},b_{max}]]$
2. For each pair of points (x_i, y_i) and (x_j, y_j), find the intersection (a',b') in (a,b)-space.
3. Increase the value of a cell that (a', b') belongs to by 1.
4. 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:

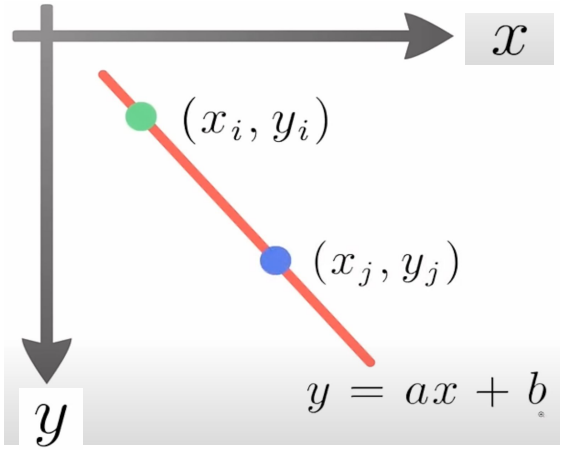


Other Hough transformations

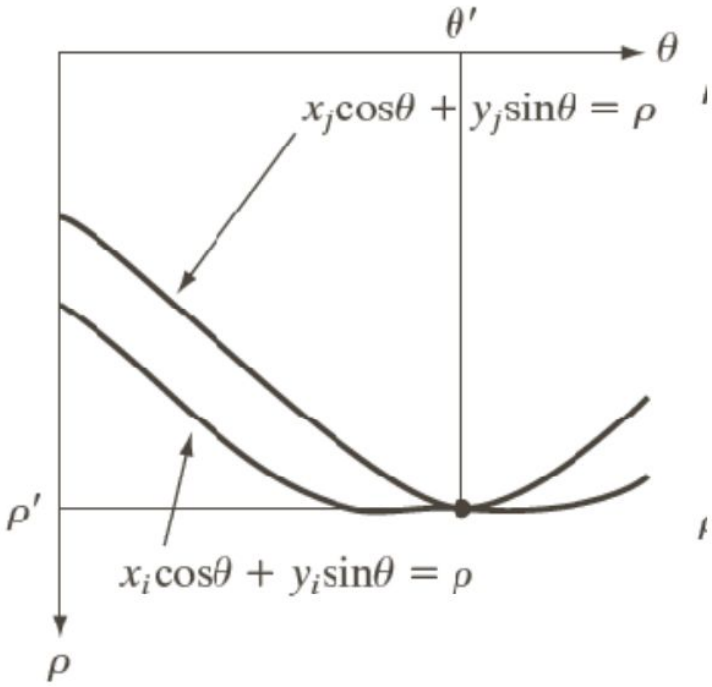
- We can represent lines as polar coordinates instead of y = a*x + b
- Polar coordinate representation:
  - x*cosθ + y*sinθ = ρ
- We can transform points in (x, y) space to curves in (ρ θ)-space
  - (x y) and (ρ θ)?

Other Hough transformations

- Note that lines in (x, y)-space are not lines in (ρ, θ)-space



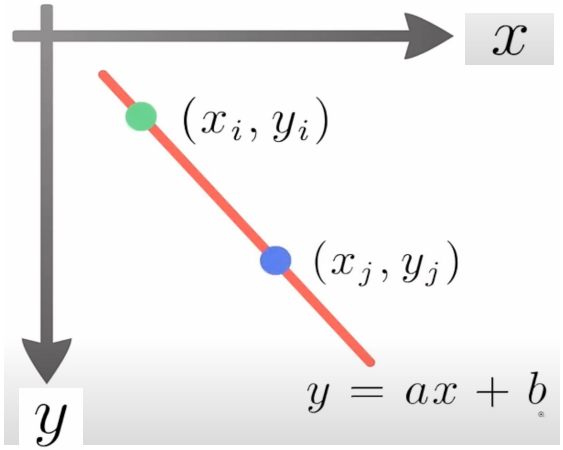
- Curves in (ρ, θ)-space intersect similarly like in (a, b)-space.

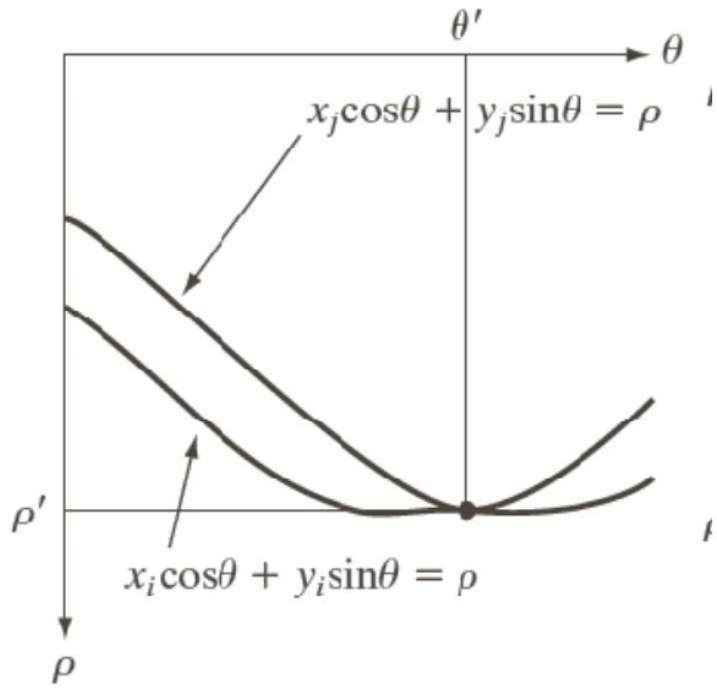


Other Hough transformations

- x*cosθ + y*sinθ = ρ

- Q. For a vertical line in (x, y)-space, what are the θ and ρ values?



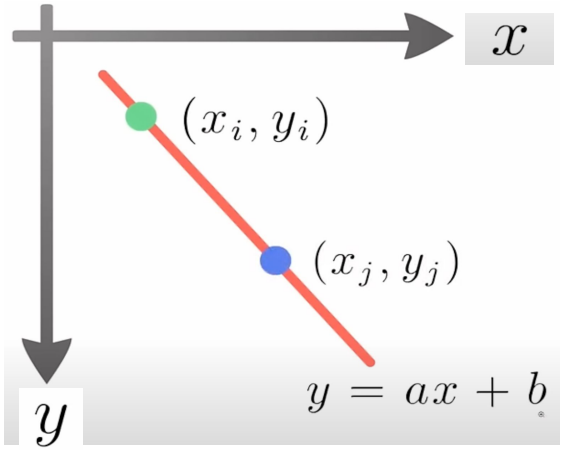


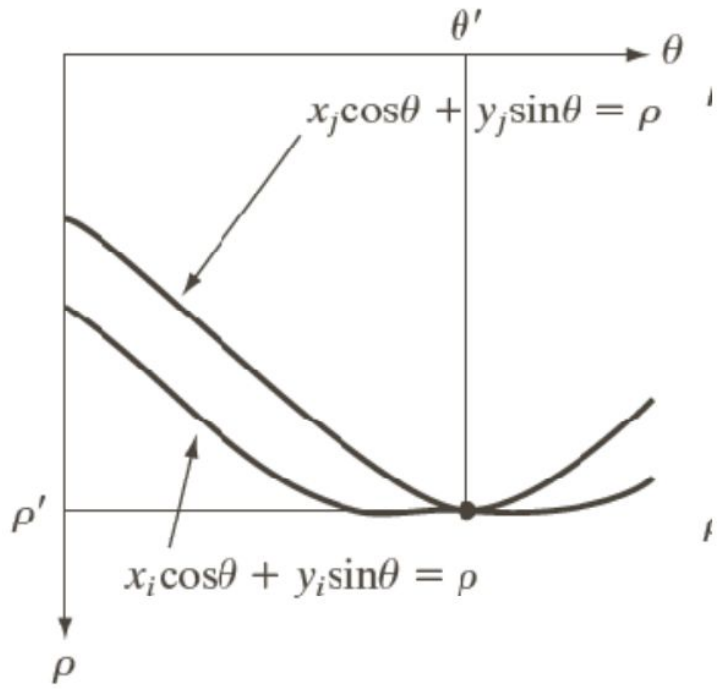
Other Hough transformations

- x*cosθ + y*sinθ = ρ

- Q. For a vertical line in (x, y)-space, what are the θ and ρ values?
  - θ=0, ρ=x

- Q. For a horizontal line in (x, y)-space, what are the θ and ρ values?





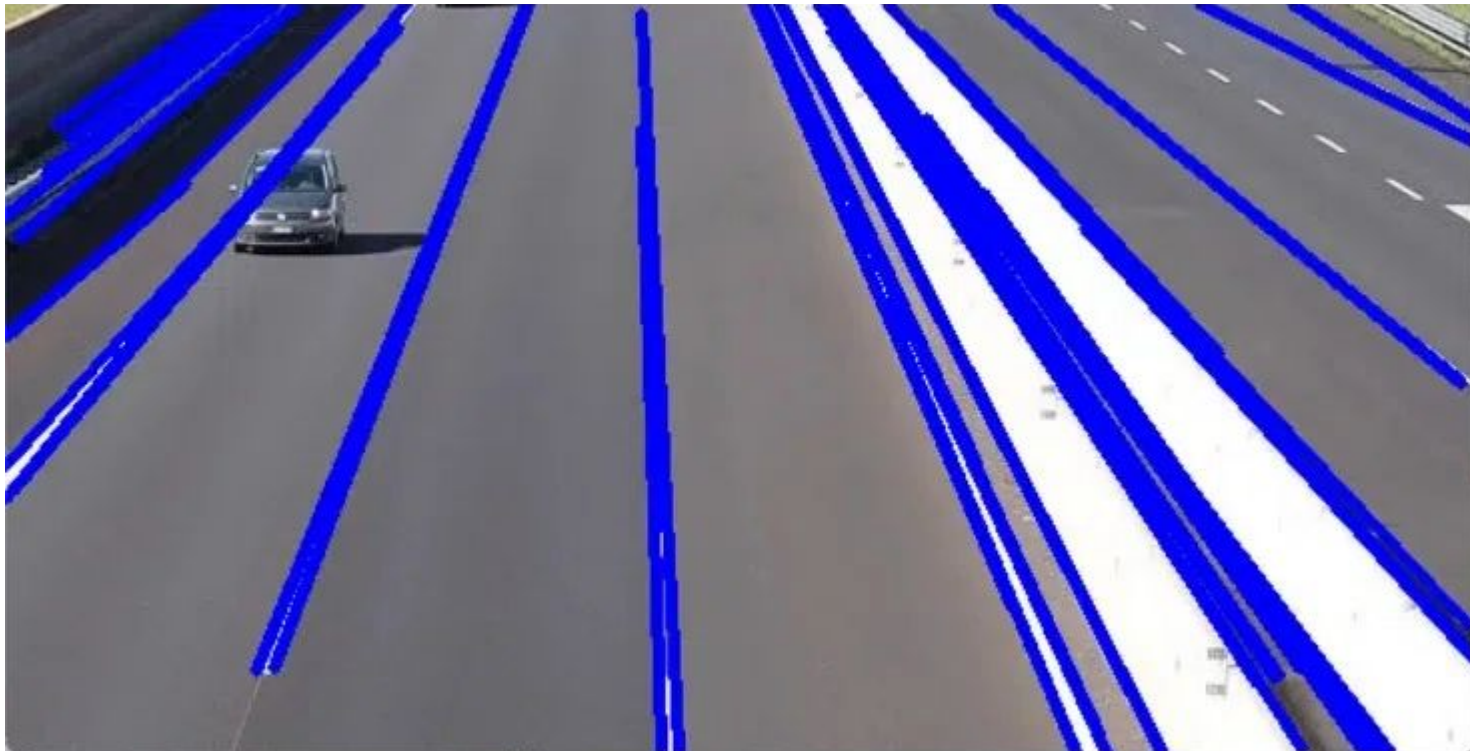
Hough transform remarks

- **Advantages**:
  - Conceptually simple.
  - Easy implementation
  - Handles missing and occluded data very gracefully.
  - Can be adapted to many types of forms, not just lines

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
- **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.
  - Runs in $O(N^2)$ since all pairs of points should be considered

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

Lines and Corners