

# Edge Detection

Chul Min Yeum

Assistant Professor

Civil and Environmental Engineering

University of Waterloo, Canada

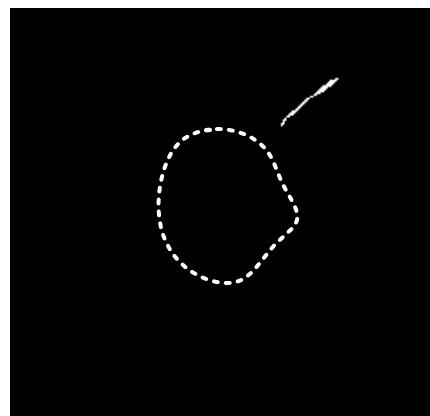
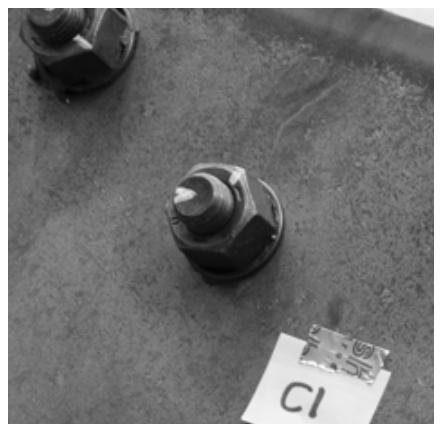
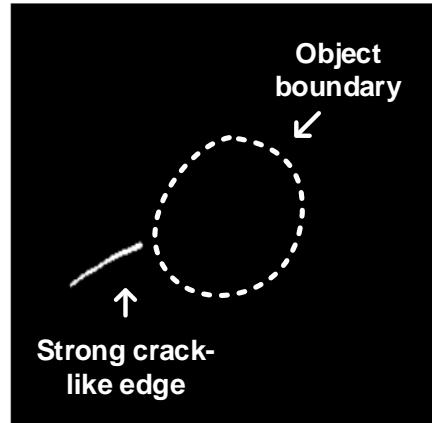


UNIVERSITY OF WATERLOO  
FACULTY OF ENGINEERING

CIVE 497 – CIVE 700: Smart Structure Technology

Last updated: 2019-02-26

# Edge Detection Applications

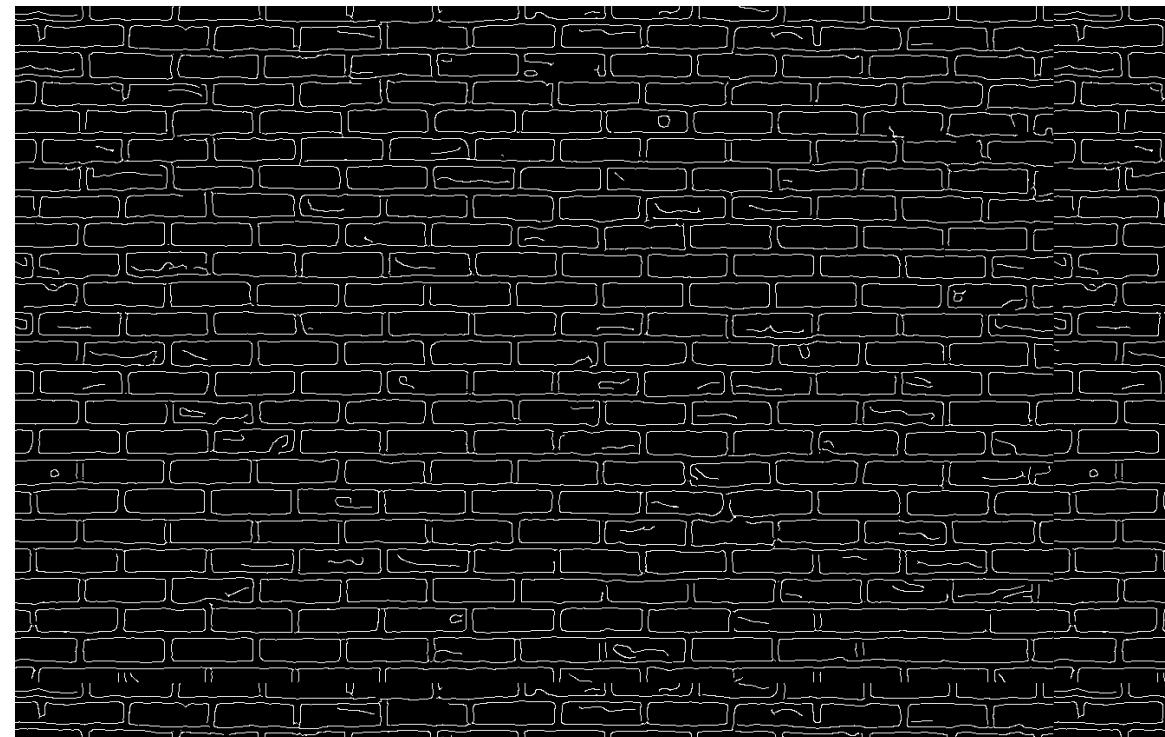
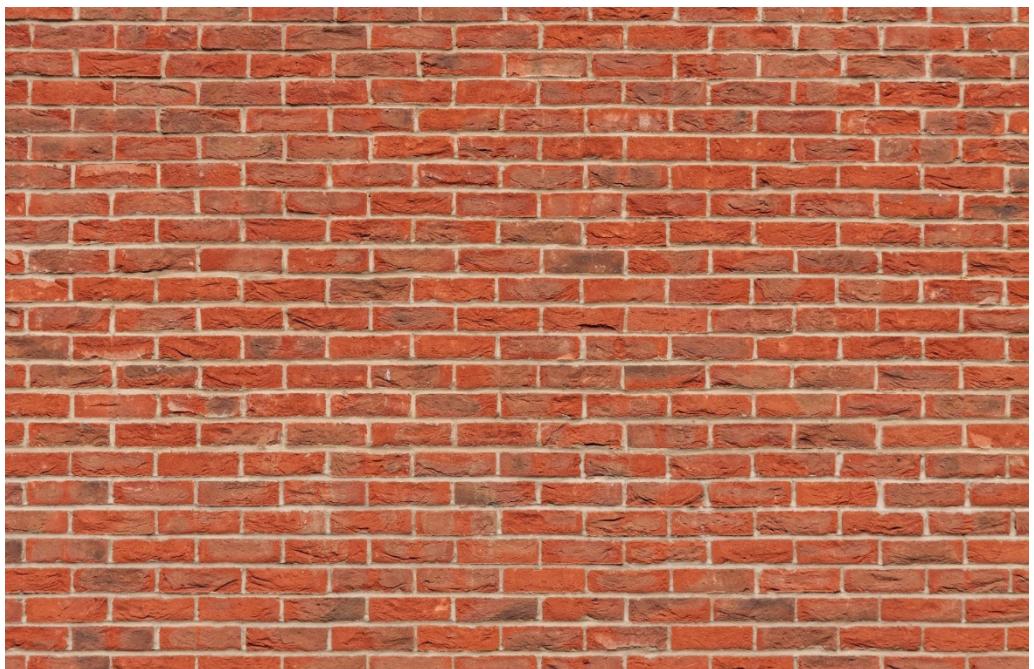


Crack detection



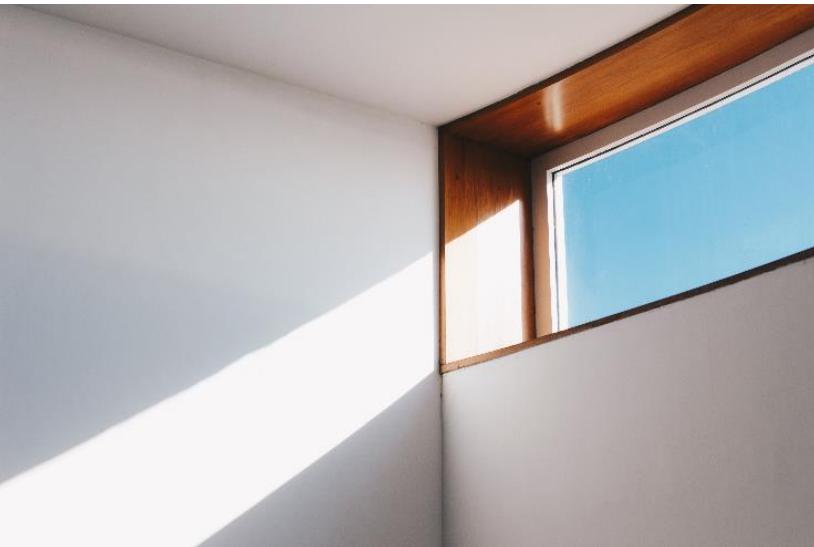
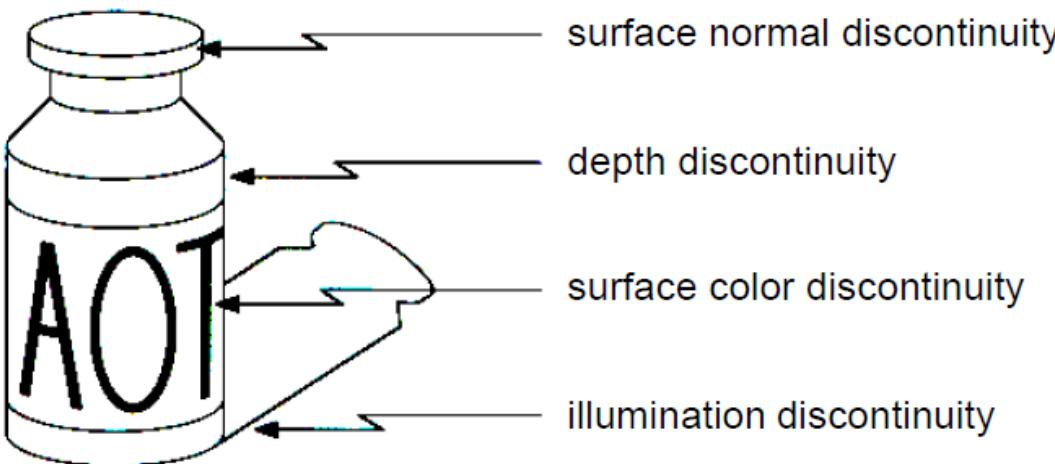
Augmented reality

# Edge Detection

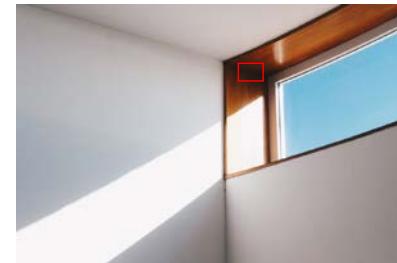


- Convert a 2D image into a set of curves
  - Semantic and shape information
  - More compact than pixels

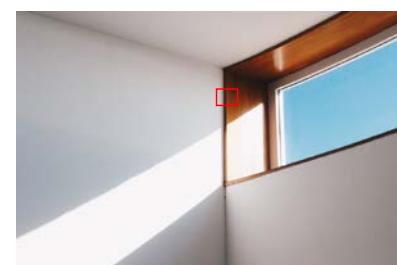
# Origin of Edges



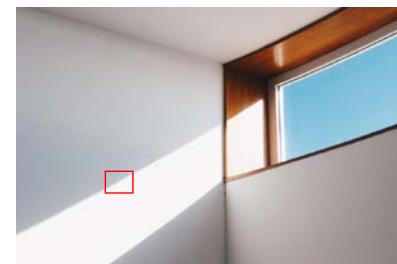
## Surface normal discontinuity



## Material discontinuity

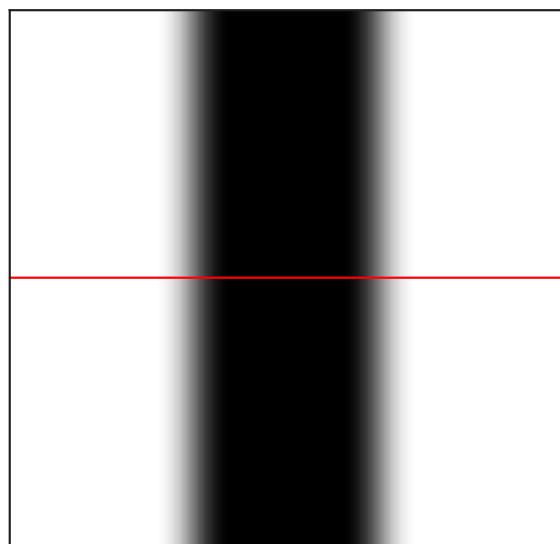


## Lighting discontinuity

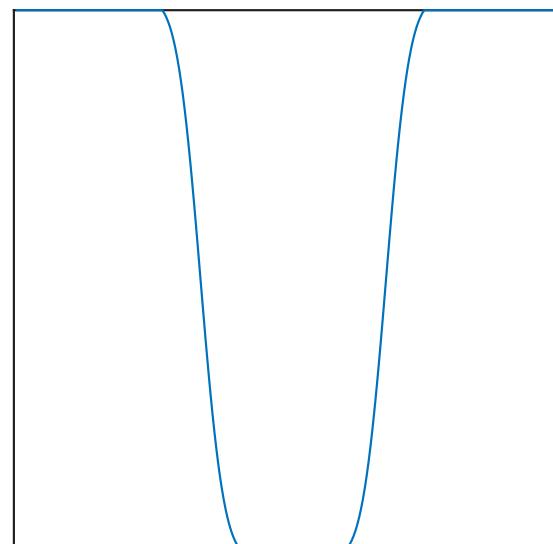


# Characterizing Edge by Differentiation

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

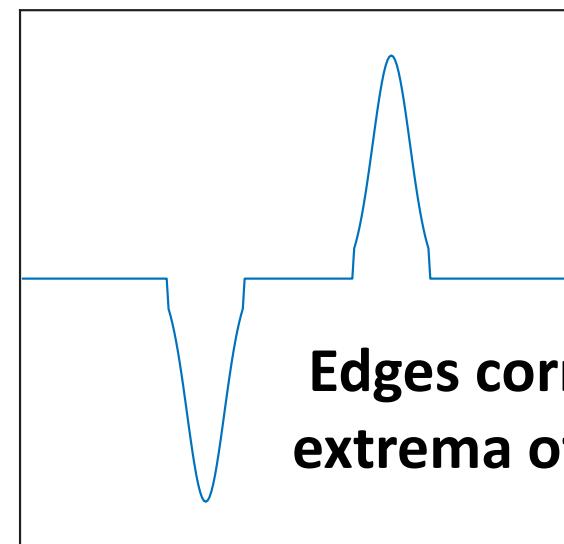


Image



Intensity function  
along a red line

$$\rightarrow \frac{d}{dx}$$



First derivative

Corner?

Edges correspond to  
extrema of derivative

# Image Derivatives

How can we differentiate a digital image  $F[x, y]$ ?

Option 1: Reconstruct a continuous image,  $f$ , then compute the derivative

Option 2: Take discrete derivative (finite difference)

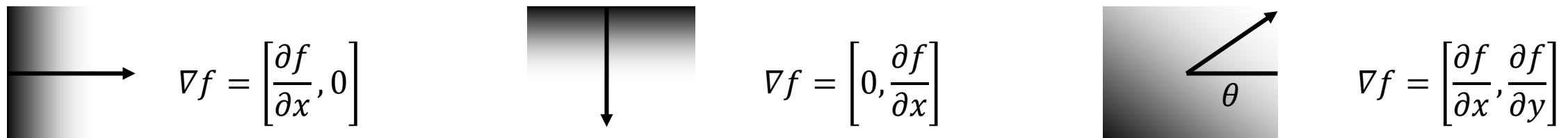
$$\frac{\partial}{\partial x} f[x, y] \approx F[x + 1, y] - F[x, y]$$

$$\Delta x = 1 \text{ pixel}$$


How would you implement this as a kernel?

# Image Gradient

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



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

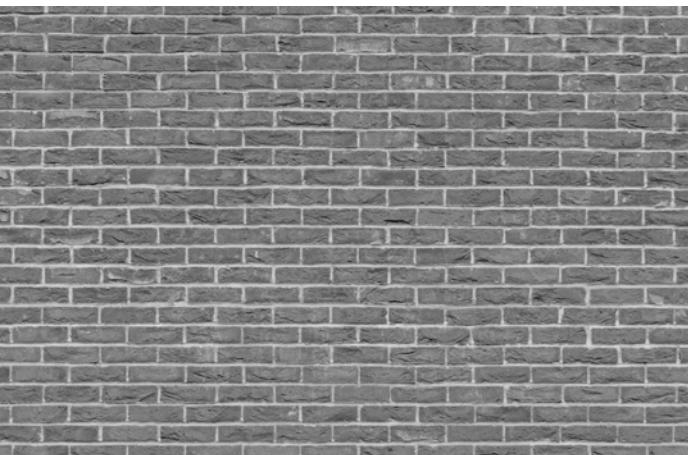
The gradient direction is given by

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

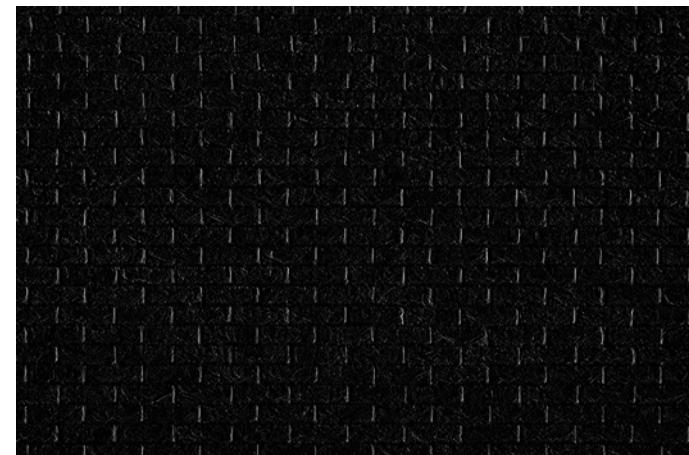
The edge strength is given by the gradient magnitude

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

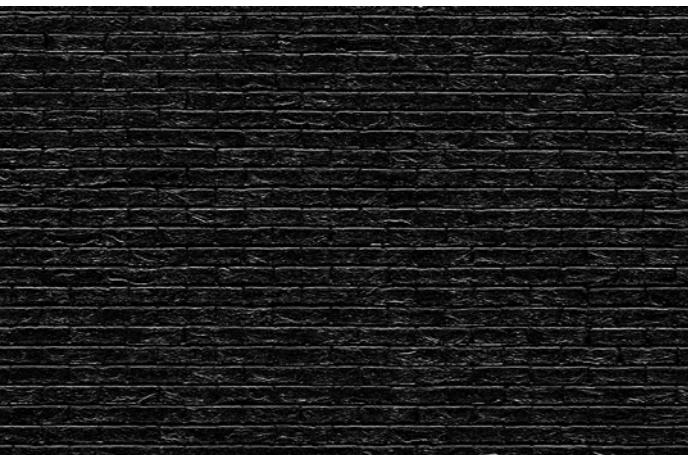
# Example: Image Gradient



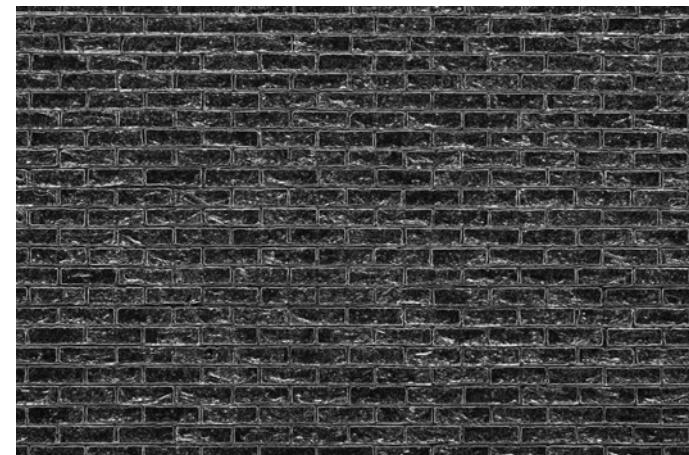
$f$



$\partial f / \partial x$

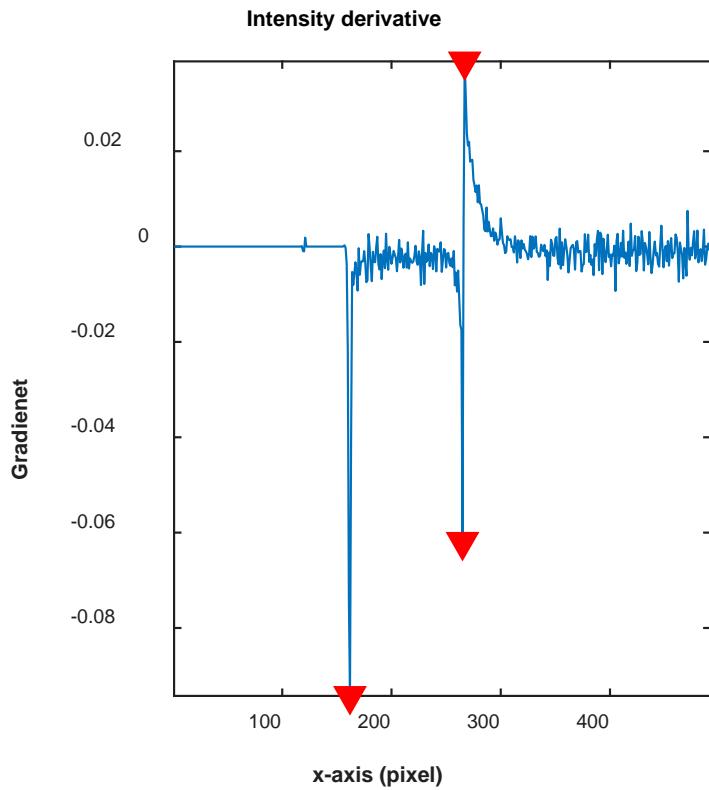
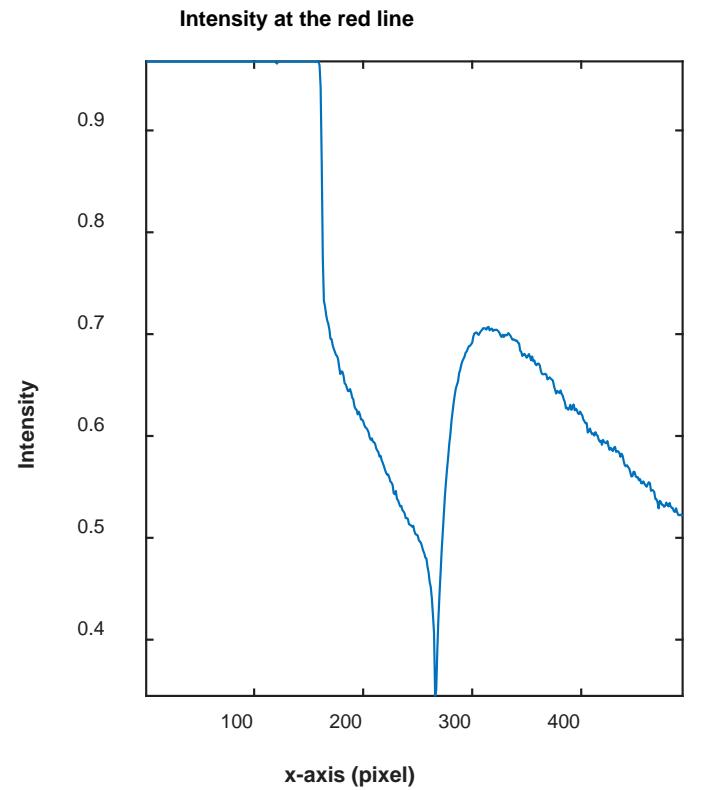
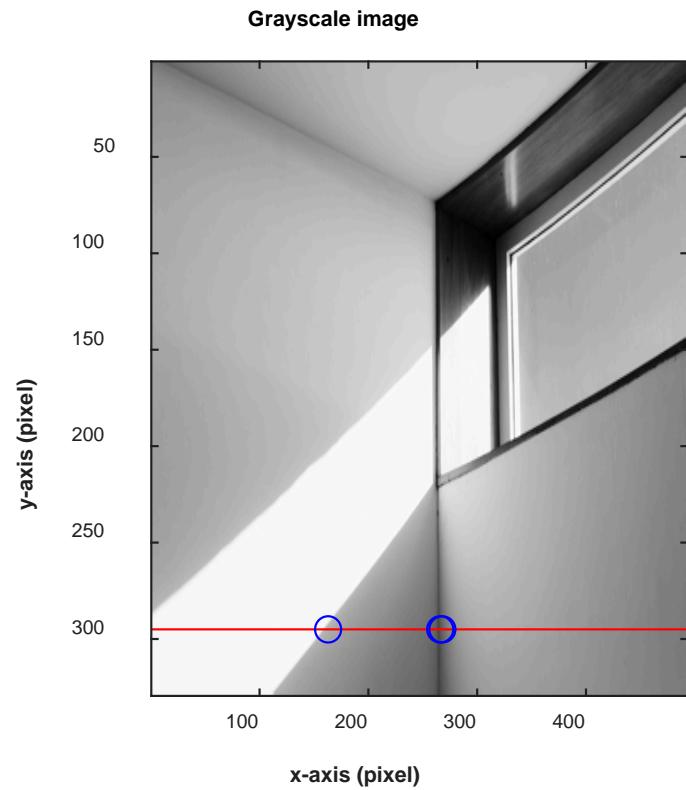


$\partial f / \partial y$

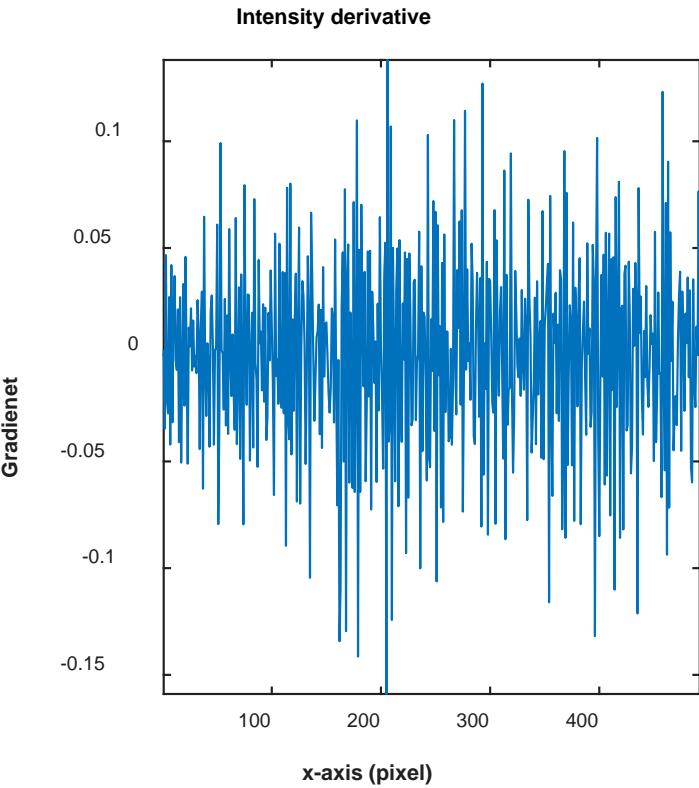
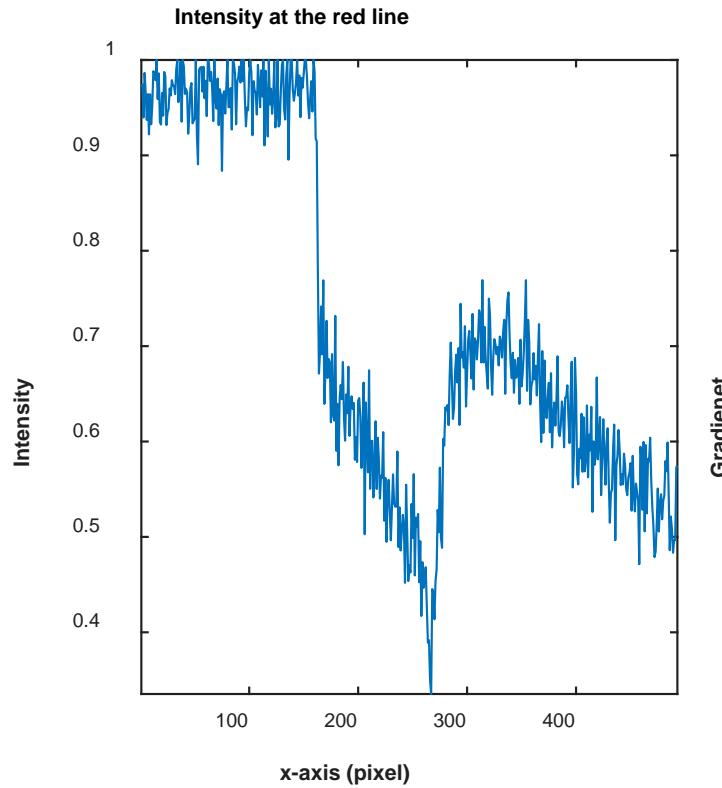
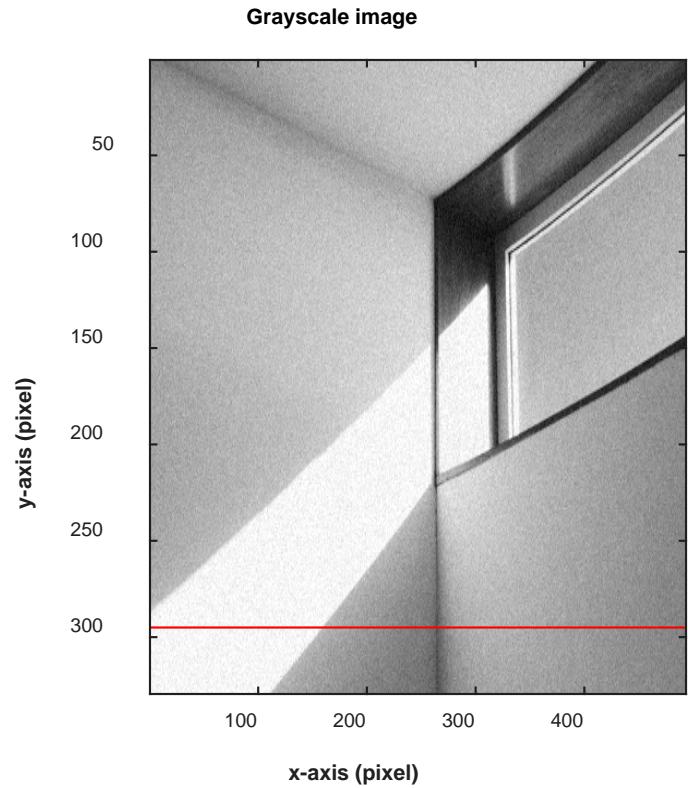


$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial Y}\right)^2}$$

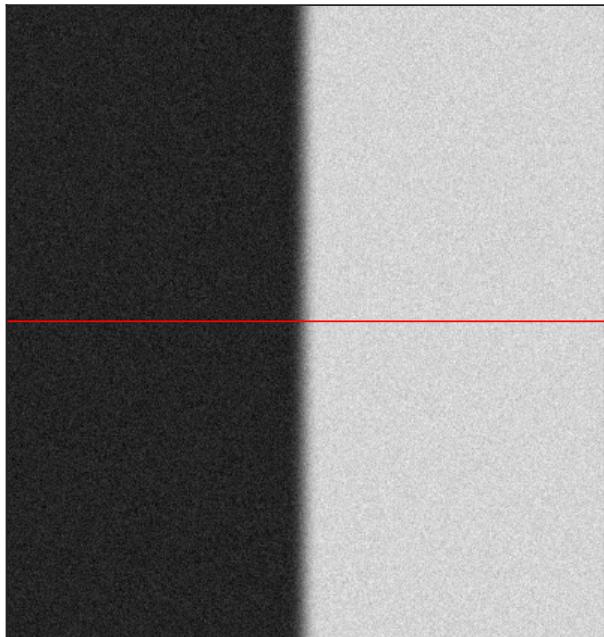
# Intensity Changes on an Image



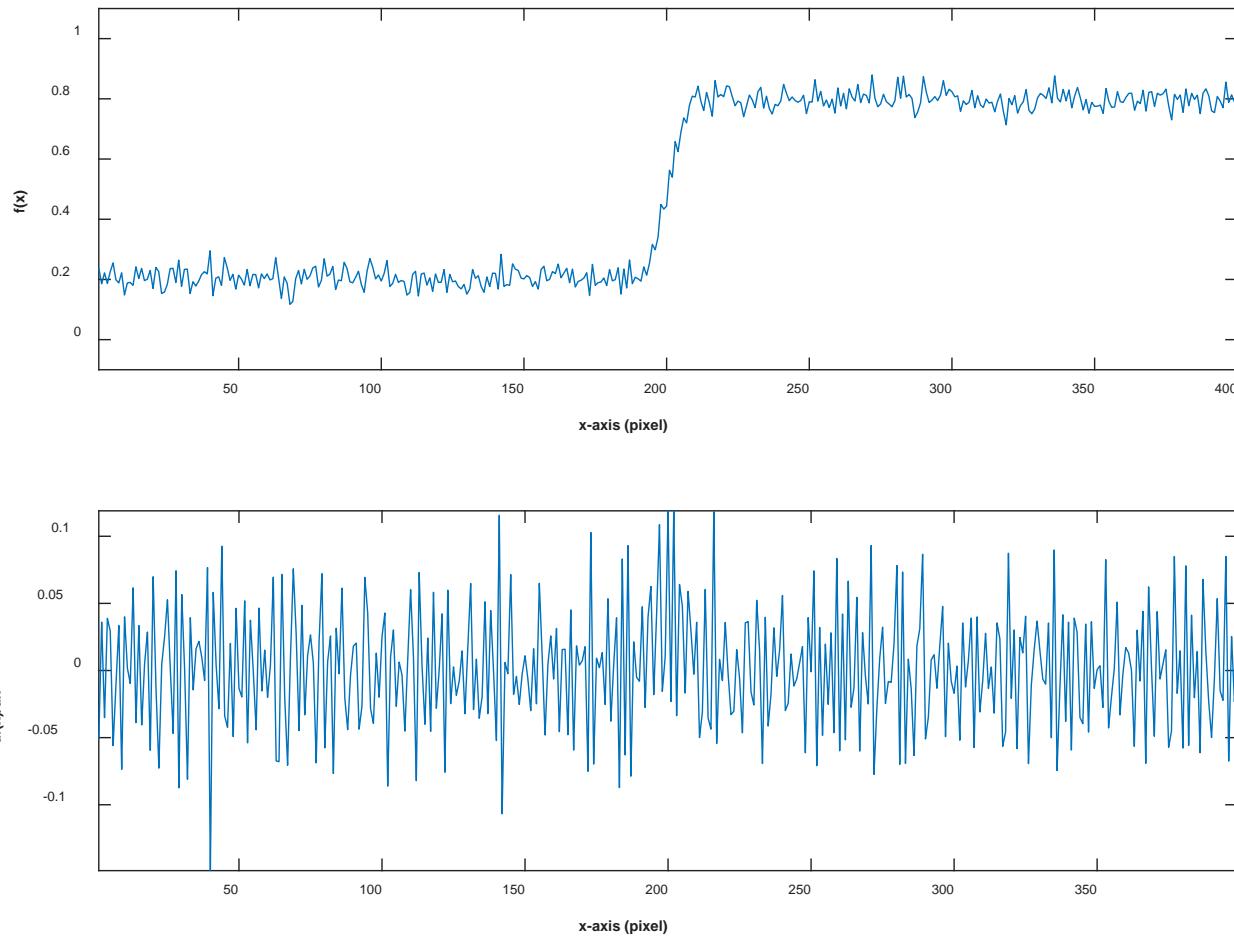
# Intensity Changes on an Image (with Noise)



# Effects of Noise

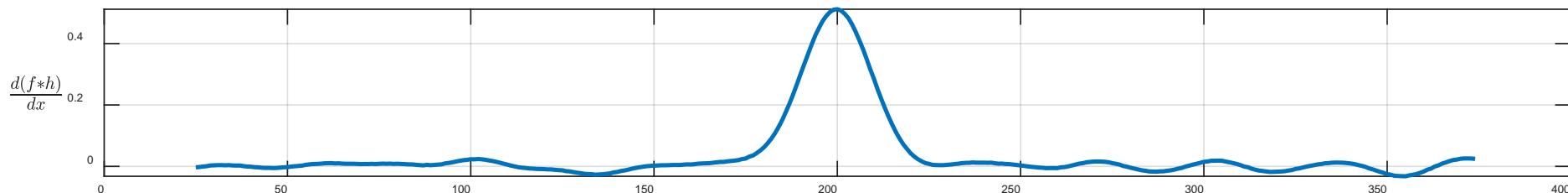
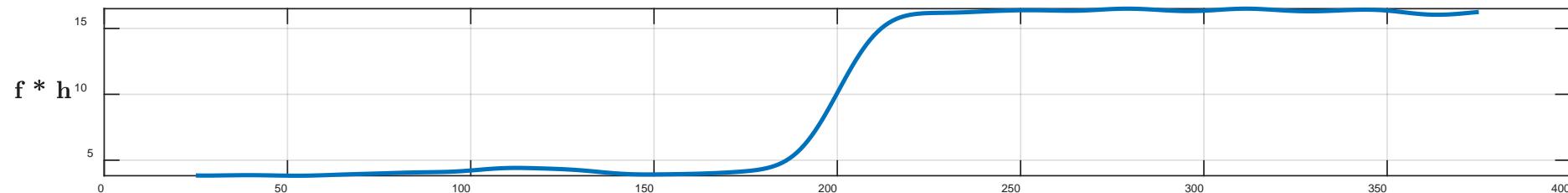
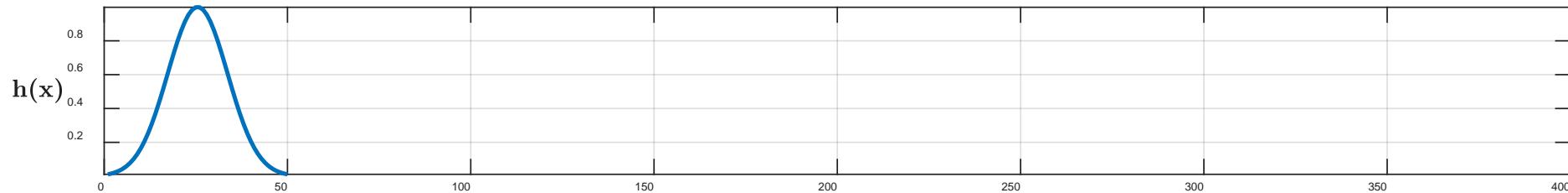
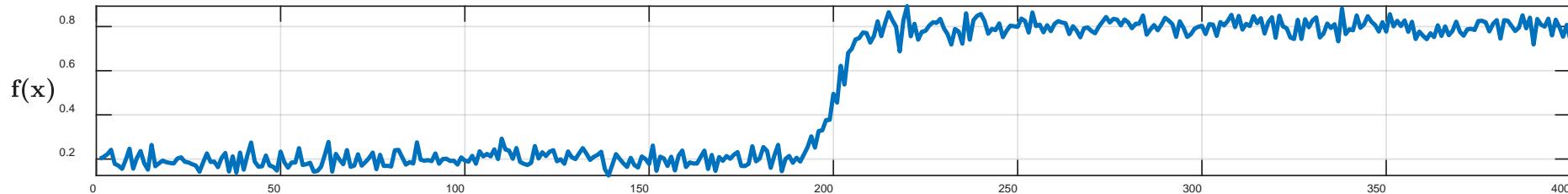


Noisy input image



Where is the edge?

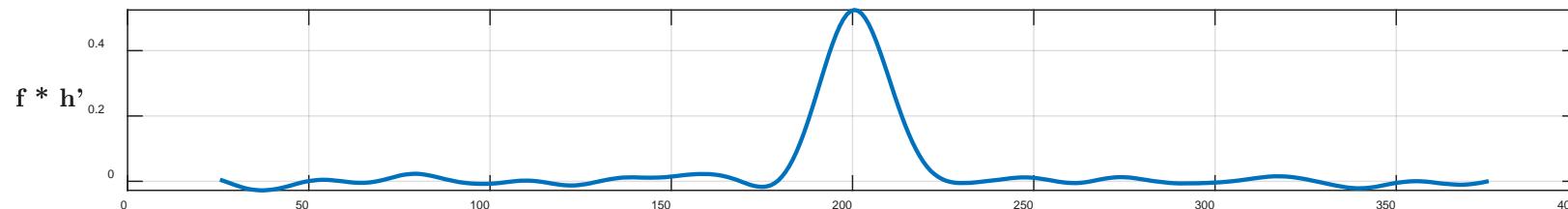
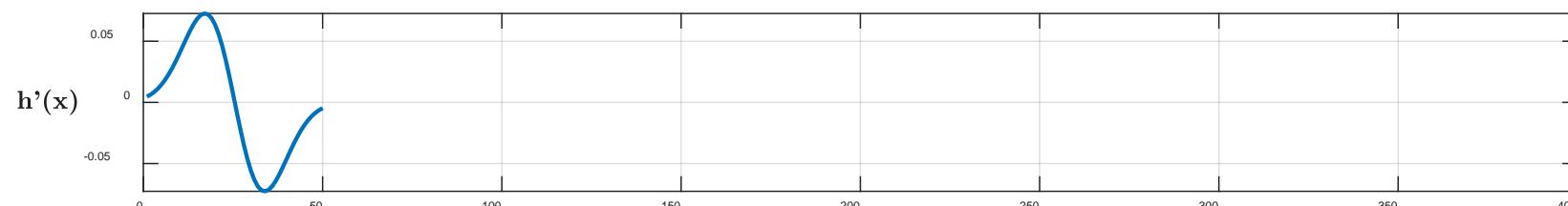
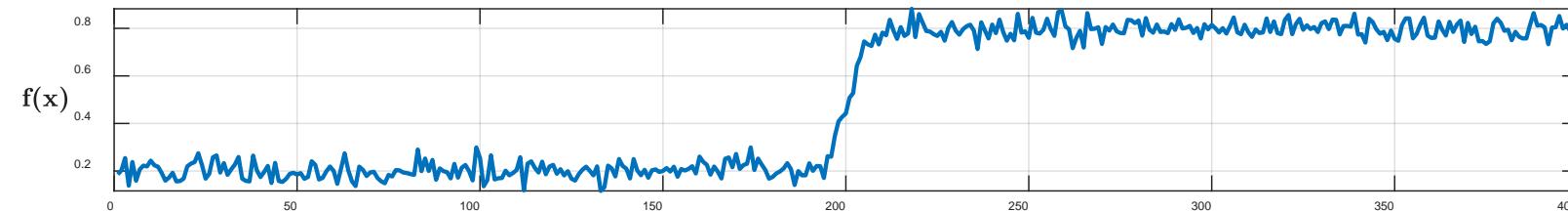
# Smoothing First and Applying a Derivative Operator



# Smoothing First and Applying a Derivative Operator (Continue)

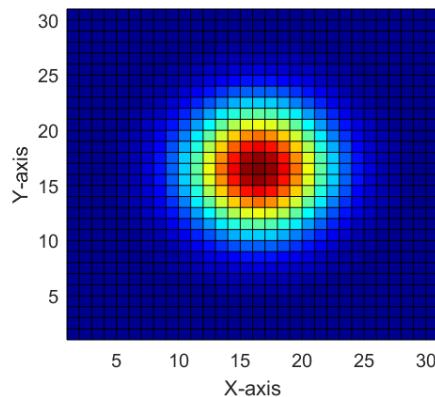
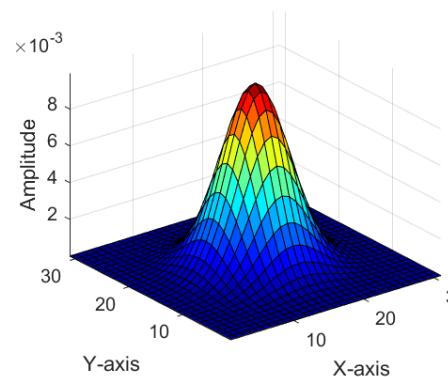
**Relationship with differentiation:**  $(F * H)' = F' * H = F * H'$

$$(f * h)' = f * h'$$



**What is the advantage?**

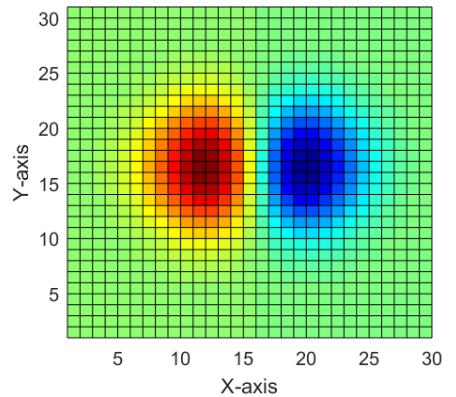
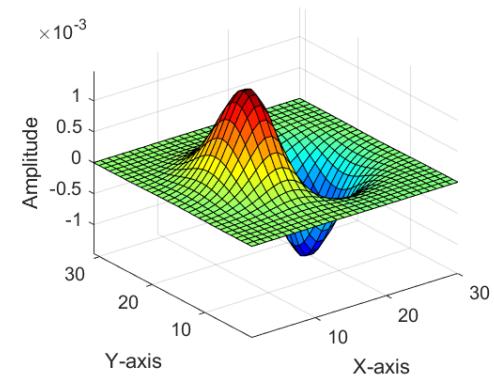
# 2D Edge Detection Filter



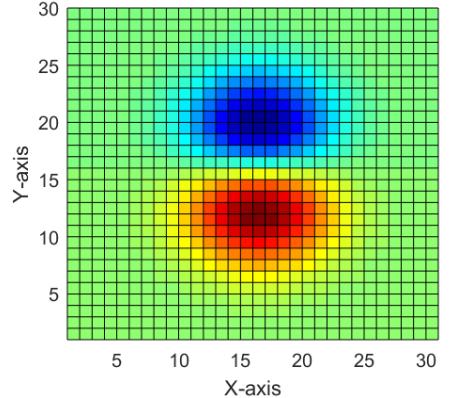
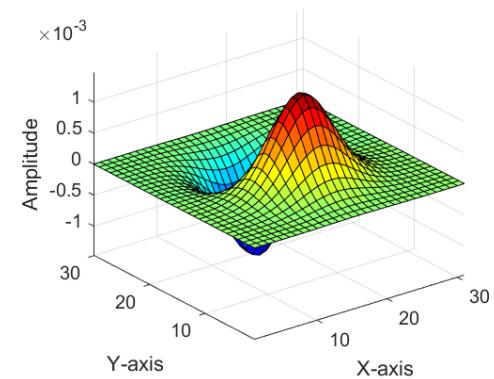
$$h_{\sigma}(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

**Gaussian kernel**

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

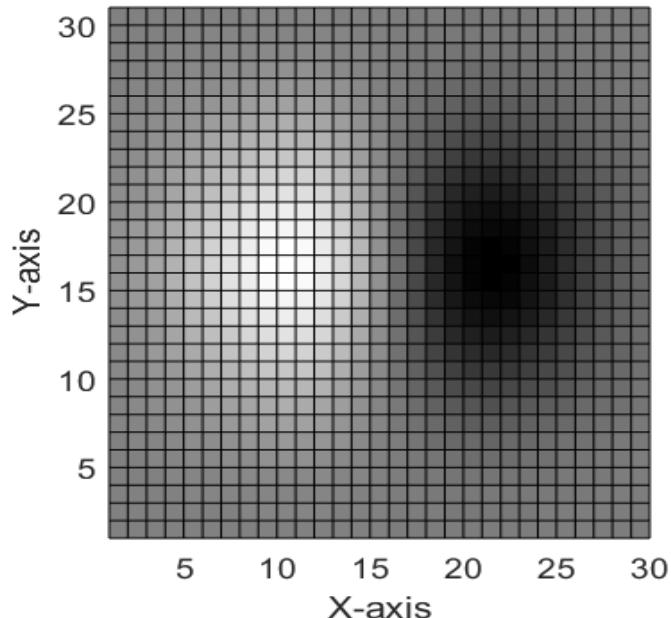
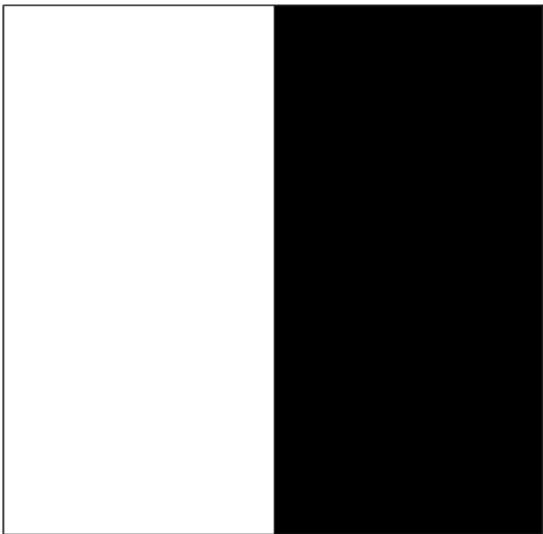


$$\frac{\partial h_{\sigma}(x, y)}{\partial y}$$



**Derivative of Gaussian filter**

# 2D Edge Detection Filter (Continue)



?

-1	0	1
-1	0	1
-1	0	1

Prewitt operator

# Sobel Operator

Common approximation of derivative of Gaussian

$$\frac{1}{8}$$

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

$S_x$

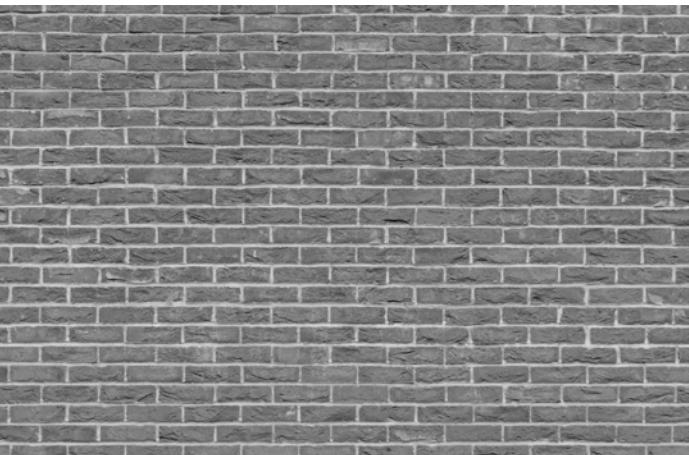
$$\frac{1}{8}$$

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

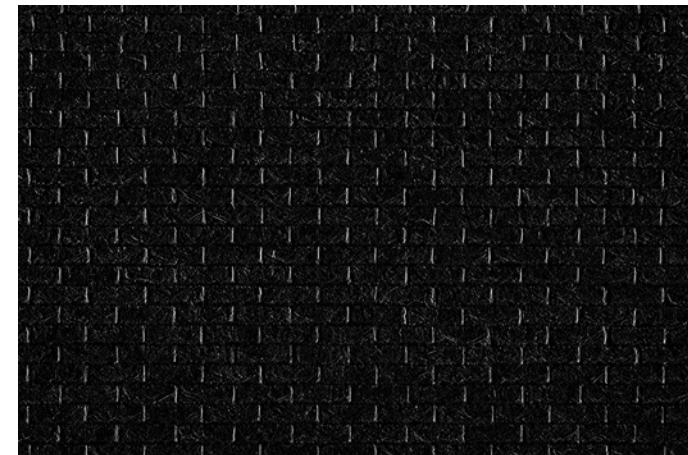
$S_y$

The Sobel operator omits the 1/8 term because it does not make a difference for edge detection

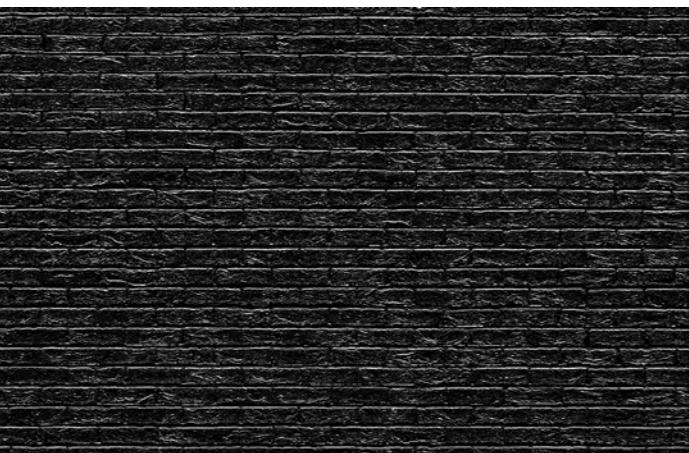
# Sobel Operator (Example)



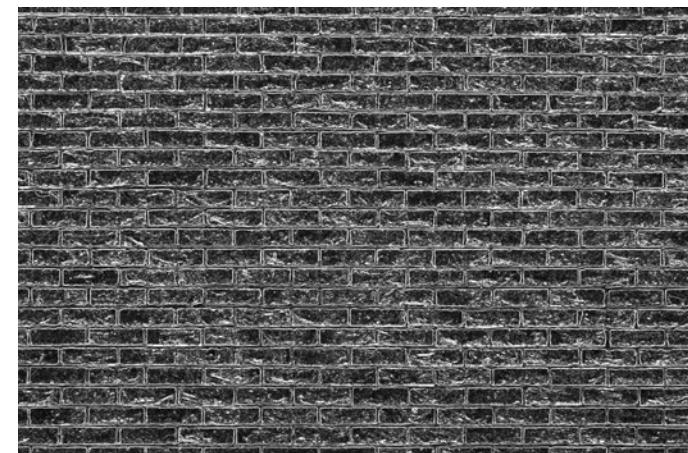
$f$



$s_x$

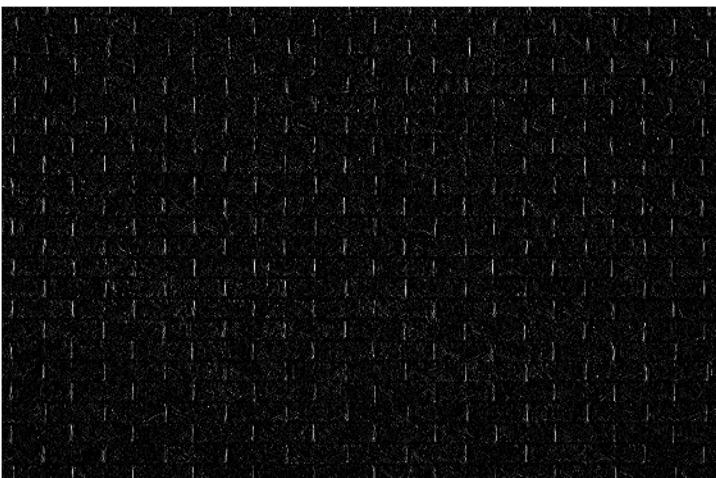


$s_y$

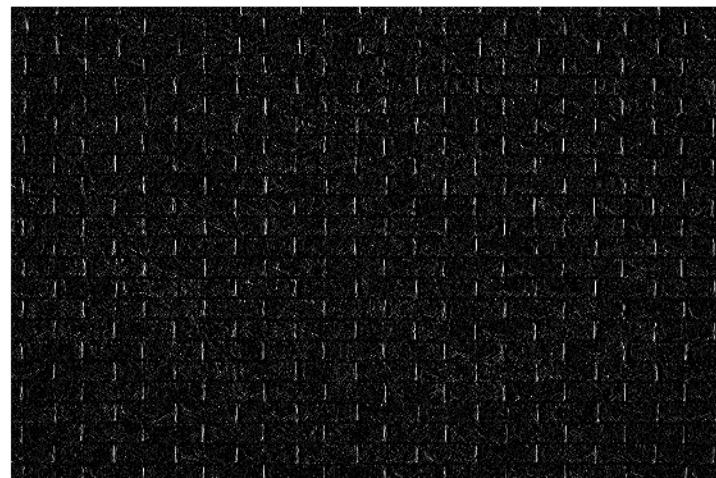


$$\|\nabla f\| = \sqrt{(s_x)^2 + (s_y)^2}$$

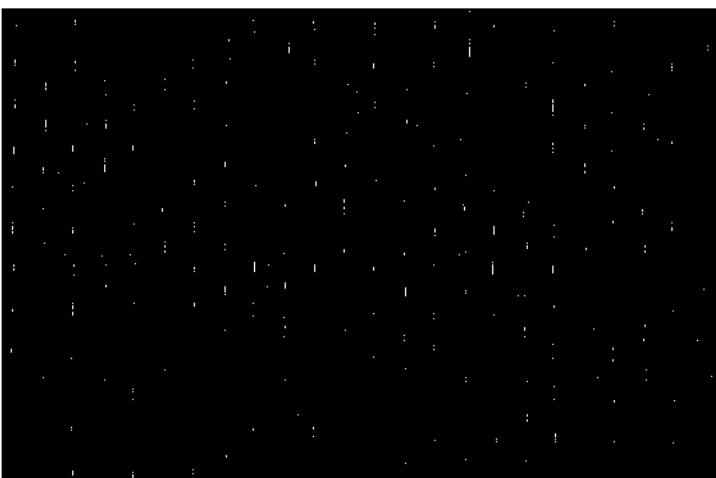
# Comparison of Derivative Operators (Prewitt and Sobel)



$f_x$



$S_x$

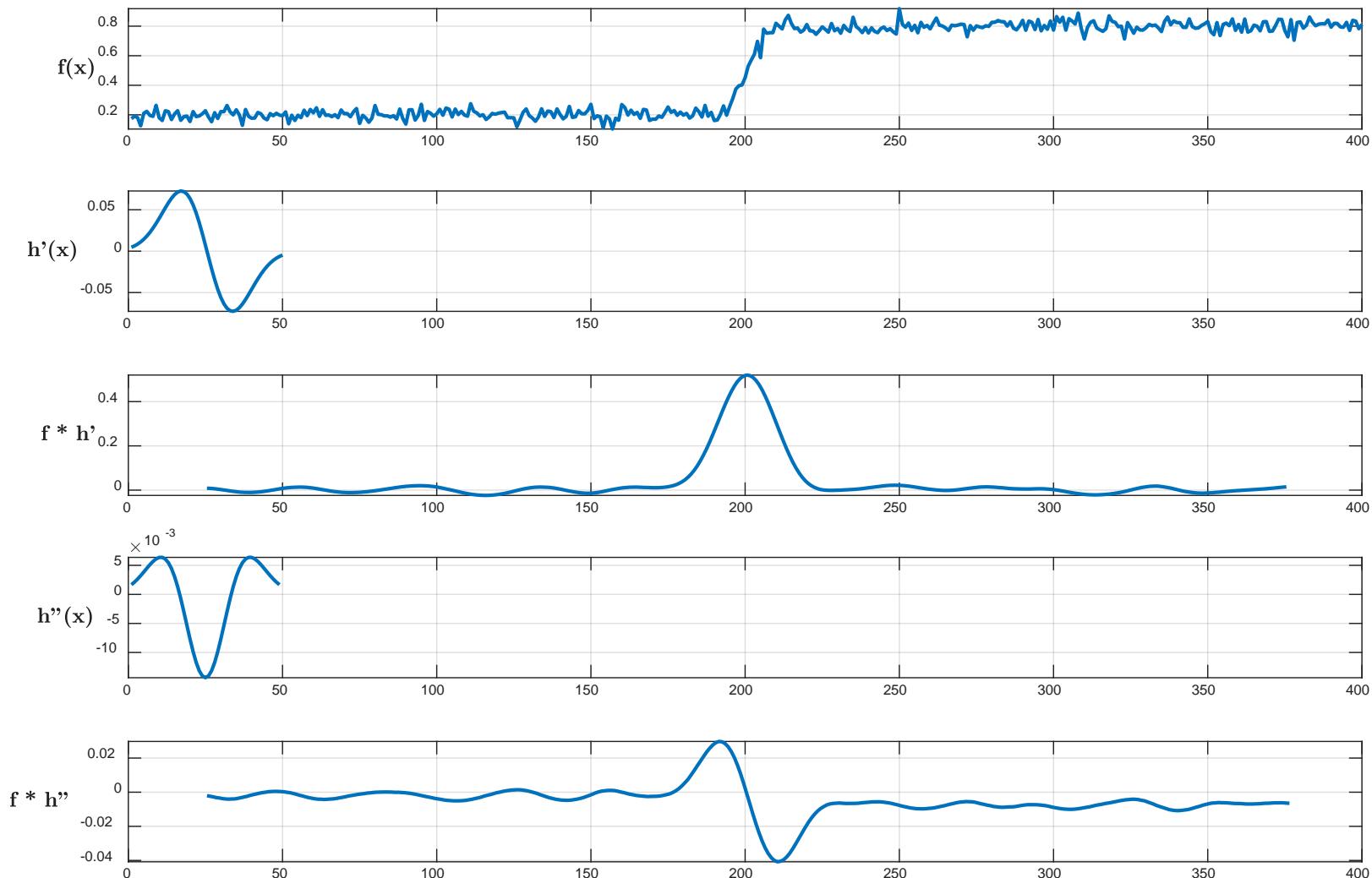


$f_x > 0.75$



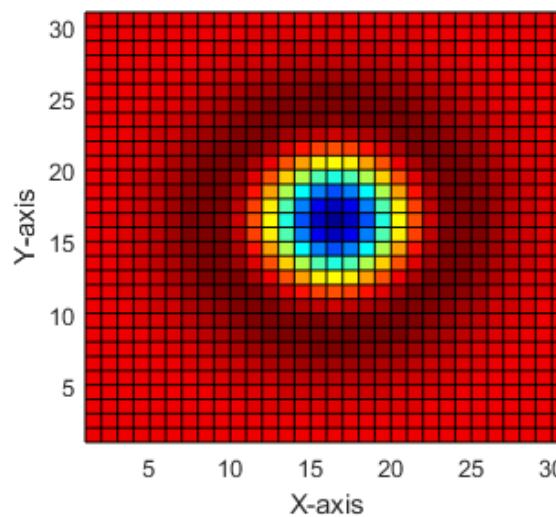
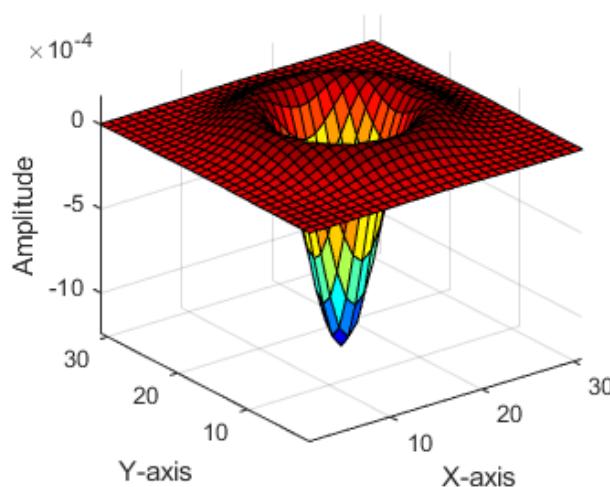
$s_x > 0.75$

# Laplacian of Gaussian (1D Example)



# Laplacian of Gaussian

$$\nabla^2 h_\sigma(x, y)$$



<b>0</b>	<b>-1</b>	<b>0</b>
<b>-1</b>	<b>4</b>	<b>-1</b>
<b>0</b>	<b>-1</b>	<b>0</b>

$$\nabla^2 h_\sigma(x, y) = \frac{\partial^2 h_\sigma}{\partial x^2} + \frac{\partial^2 h_\sigma}{\partial y^2}$$

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

# Implementation Issues for Gradient-based Edge Detection

- The gradient magnitude is large along a thick “trail” or “ridge,” so how do we identify the actual edge points?
- How do we link the edge points to form curves?



# Canny Edge Detector

- This is probably the most widely used edge detector in computer vision
- Theoretical model: step-edges corrupted by additive Gaussian noise
- J. Canny, [\*A Computational Approach To Edge Detection\*](#), IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

## Steps

- Filter image with x, y derivatives of Gaussian
- Find magnitude and orientation of gradient
- Non-maximum suppression: Thin multi-pixel wide “ridges” down to single pixel width
- Thresholding and linking (hysteresis): Define two thresholds: **low and high** and use the high threshold to start edge curves and the low threshold to continue them
- MATLAB: `edge(image, 'canny')`

# Original Image and Its Gradient

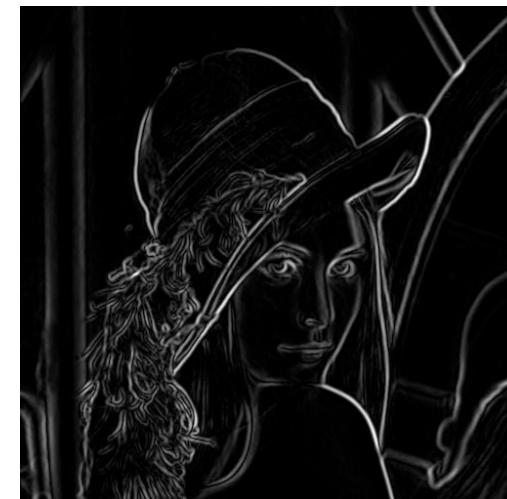
original image (Lena)



X-Derivative of Gaussian

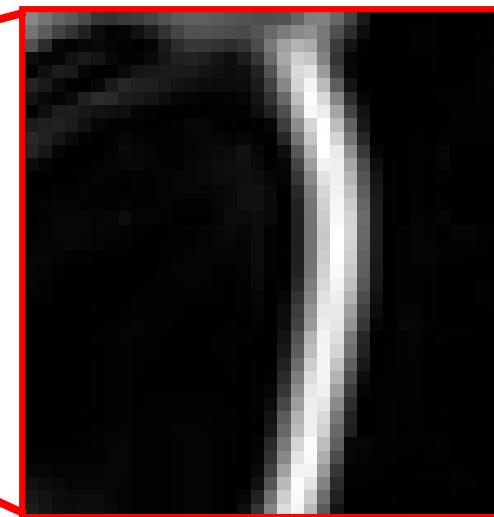


Y-Derivative of Gaussian

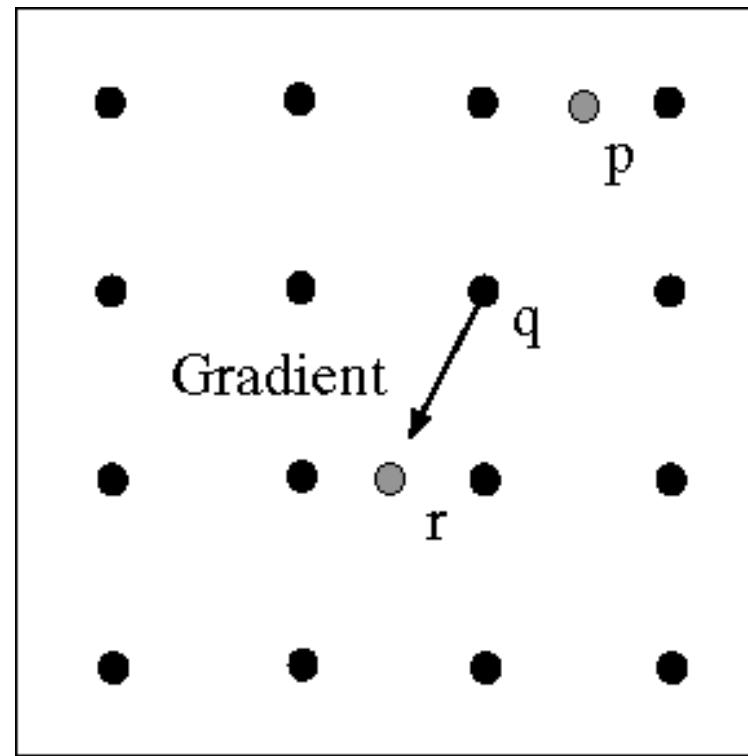
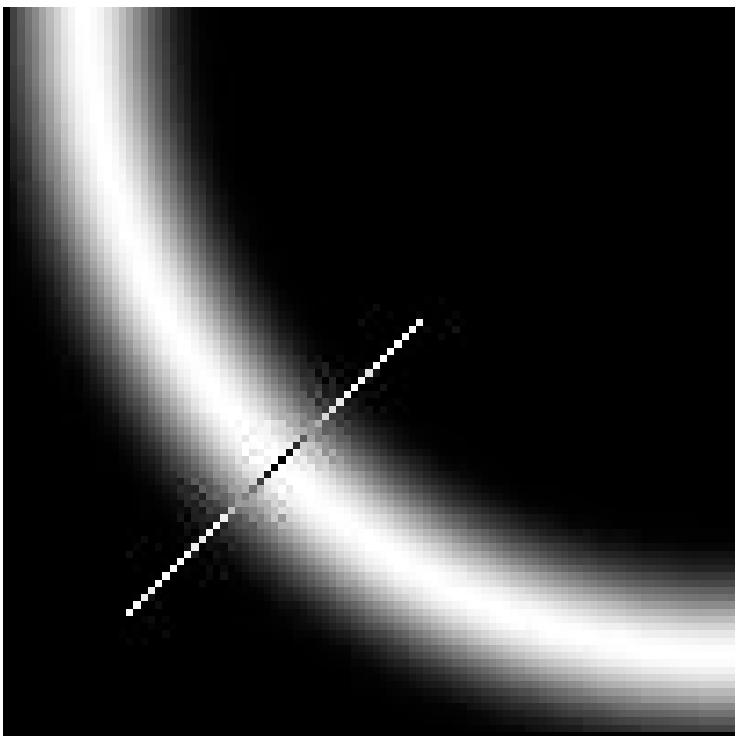


Gradient Magnitude

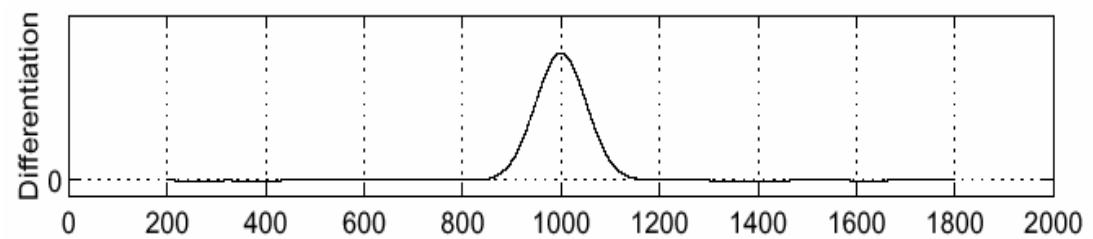
# Finding Edges



# Non-maximum Suppression



- Check if pixel is local maximum along gradient direction
  - Requires interpolating pixels  $p$  and  $r$



# After Non-maximum Suppression



# Hysteresis Thresholding

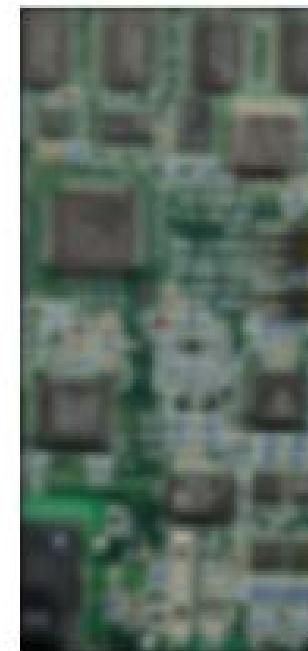
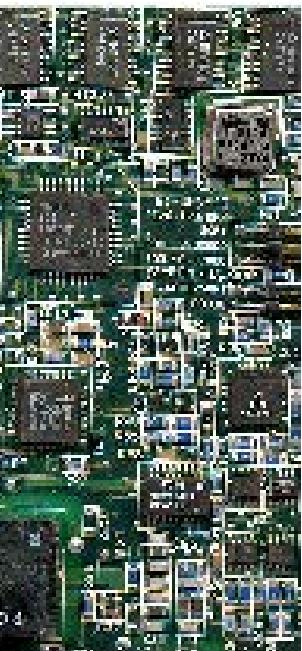
- Threshold at low/high levels to get weak/strong edge pixels
- Do connected components, starting from strong edge pixels



# Final Canny Edges

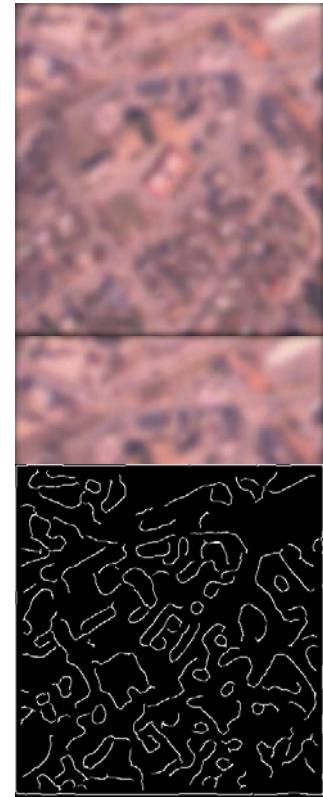


# (Revisit) Effect of Gaussian Window Sizes



```
f1 = fspecial('gaussian', 101,1);
f2 = fspecial('gaussian', 101,5);
f3 = fspecial('gaussian', 101,10);
f4 = fspecial('gaussian', 101,30);
```

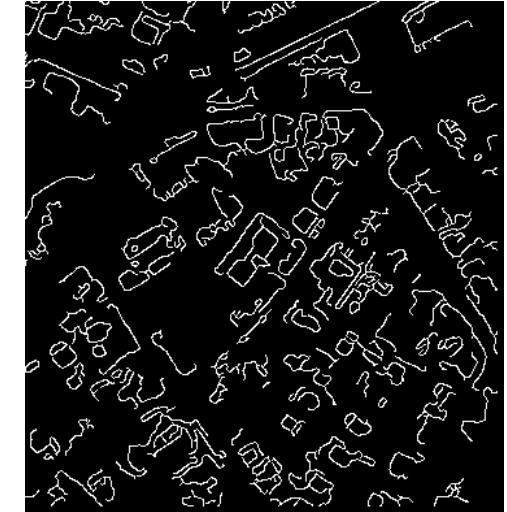
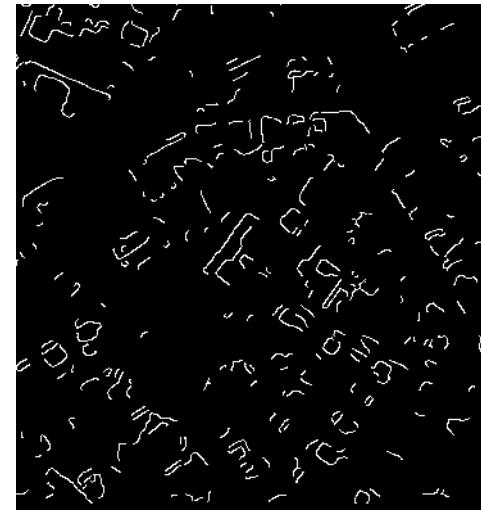
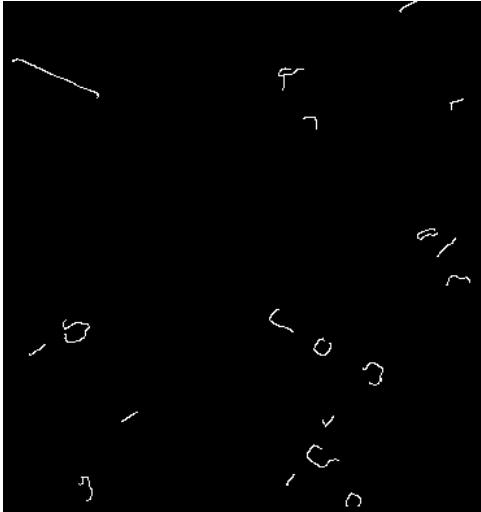
# Effect of $\sigma$ (Gaussian Kernel Size)



The choice of  $\sigma$  depends on desired behavior

- large  $\sigma$  detects large scale edges
- small  $\sigma$  detects fine features

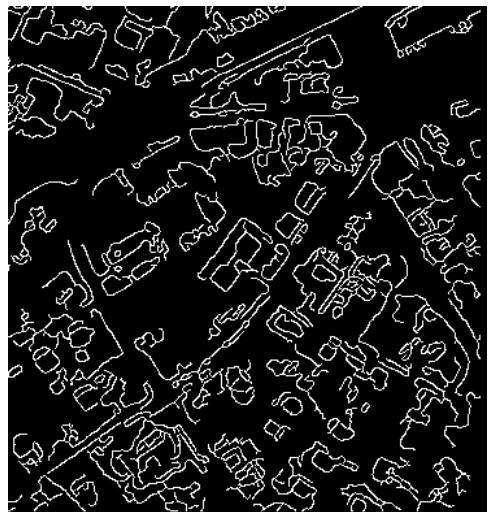
# Effect of High and Low Thresholds



**[0.4 0.8]**

**[0.4 0.5]**

**[0.2 0.5]**



**[0.1 0.5]**

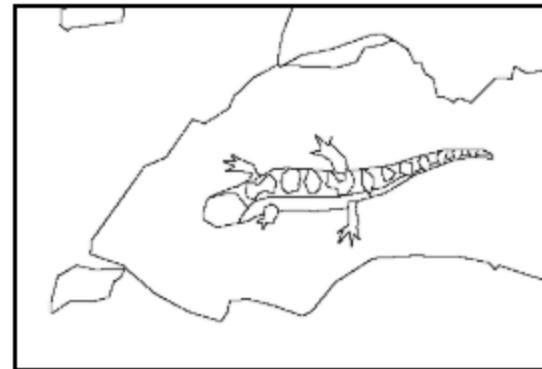
# Contour Detection and Image Segmentation Resources (Computer Vision Group in Berkeley)



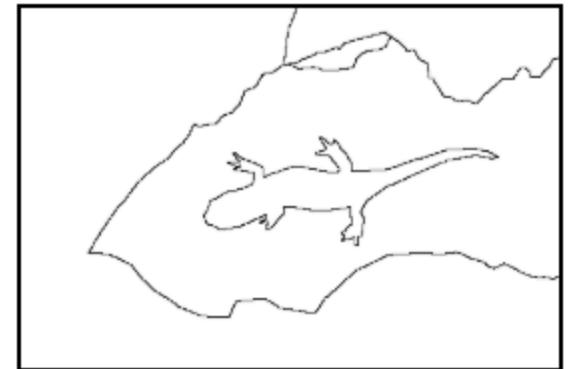
Original Image



Subject 1

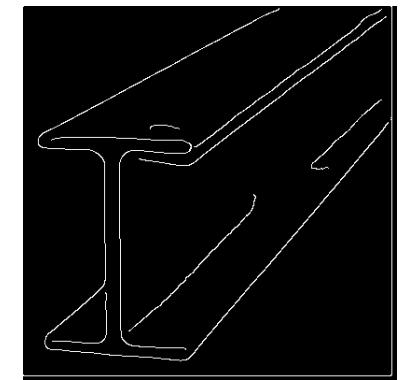
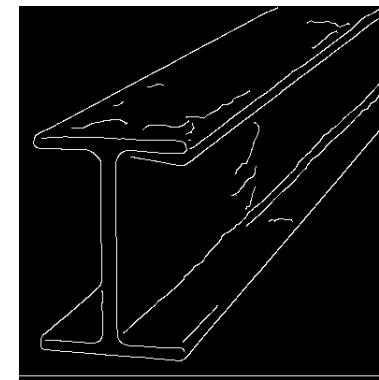
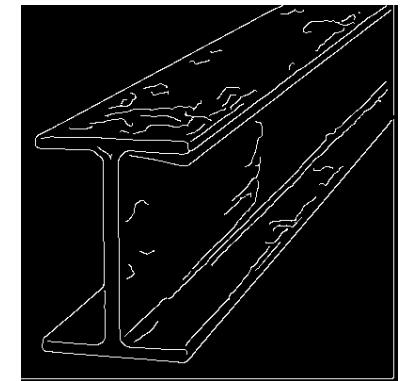
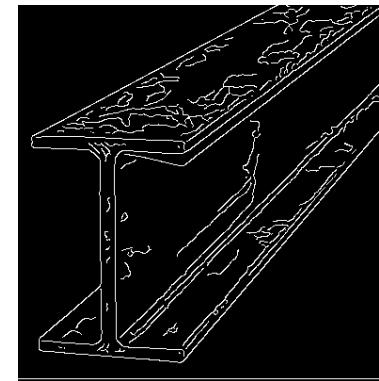


Subject 2



Subject 3

# Hough Transform



The basic idea is to examine the parameter space for lines, rather than the image spaces.

# Image and Parameter Spaces

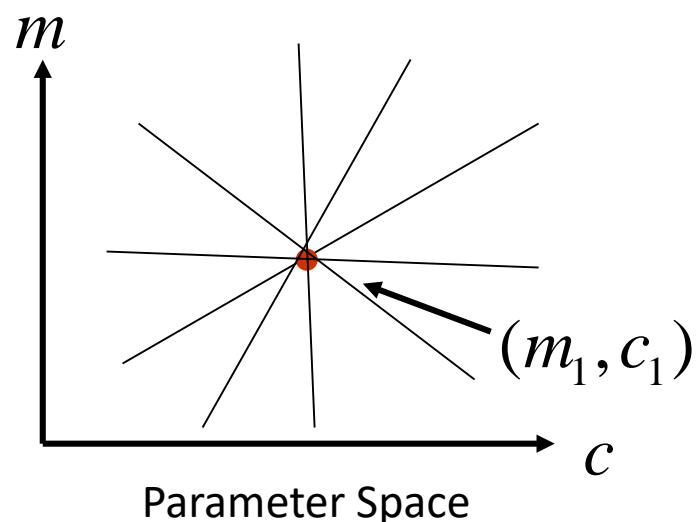
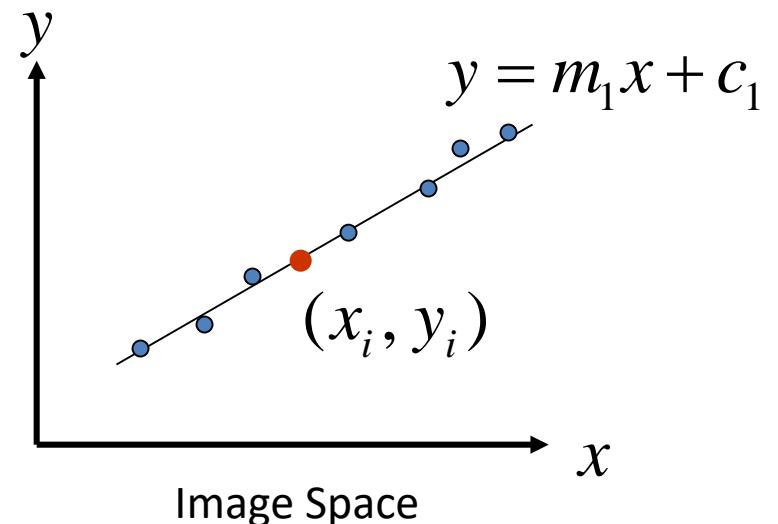
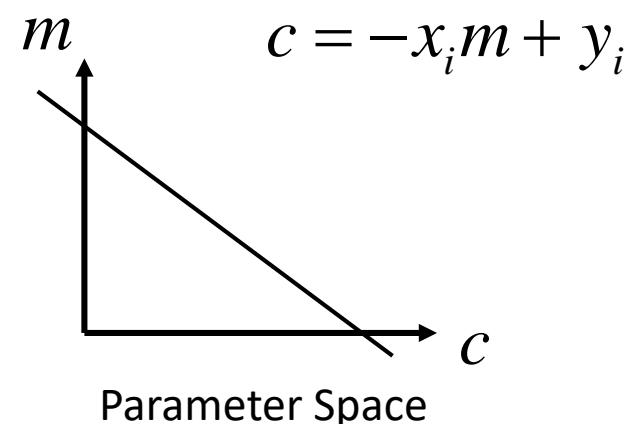
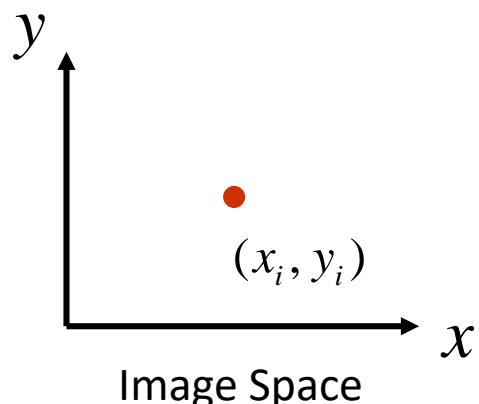
Equation of Line:  $y = mx + c$

Find:  $(m, c)$

Consider point:  $(x_i, y_i)$

$$y_i = mx_i + c \quad \text{or} \quad c = -x_i m + y_i$$

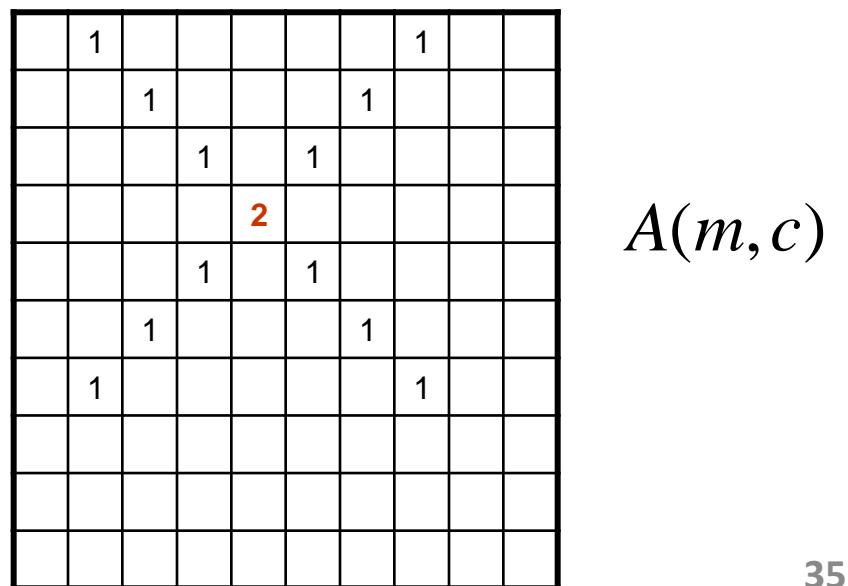
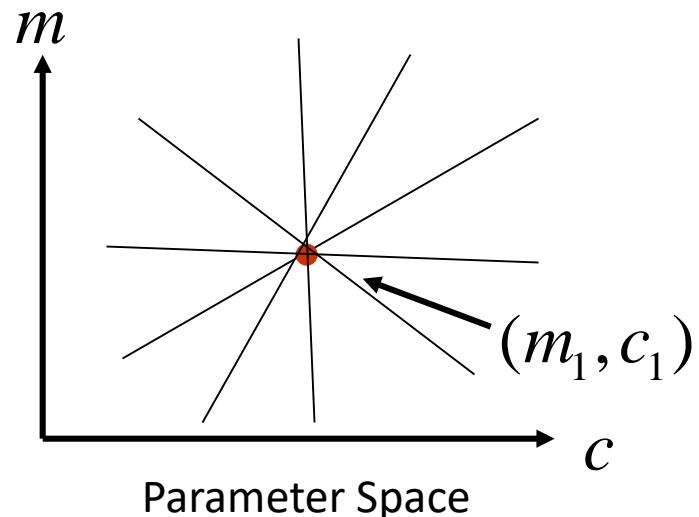
Parameter space also called Hough Space



# Line Detection by Hough Transform

Algorithm:

- Quantize Parameter Space  $(m, c)$
- Create Accumulator Array  $A(m, c)$
- Set  $A(m, c) = 0 \quad \forall m, c$
- For each image edge  $(x_i, y_i)$  increment:  
$$A(m, c) = A(m, c) + 1$$
- If  $(m, c)$  lies on the line:  
$$c = -x_i m + y_i$$
- Find local maxima in  $A(m, c)$



# Better Parameterization

NOTE:  $-\infty \leq m \leq \infty$

Large Accumulator

More memory and computations

Improvement: (Finite Accumulator Array Size)

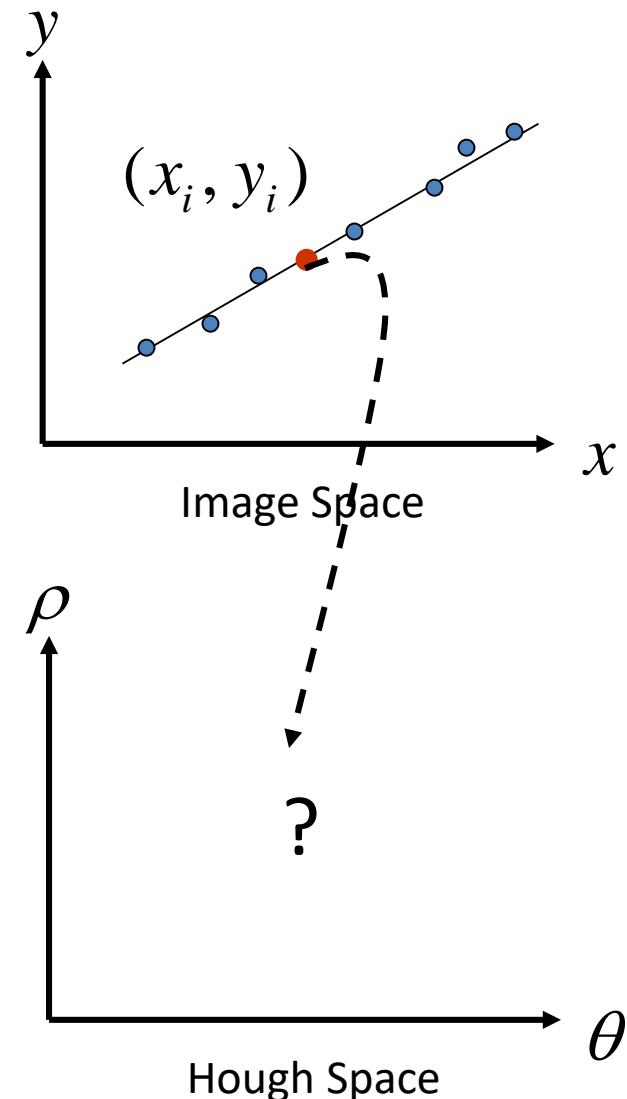
$$\text{Line equation: } \rho = -x \cos \theta + y \sin \theta$$

Here  $0 \leq \theta \leq 2\pi$

$$0 \leq \rho \leq \rho_{\max}$$

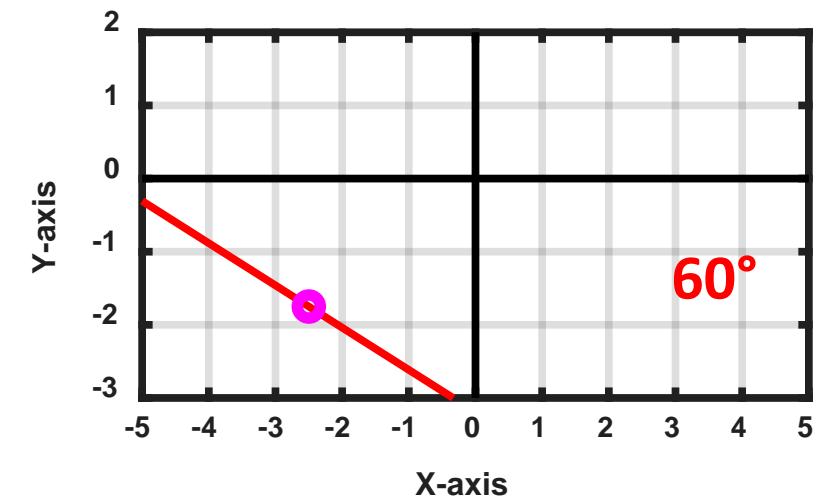
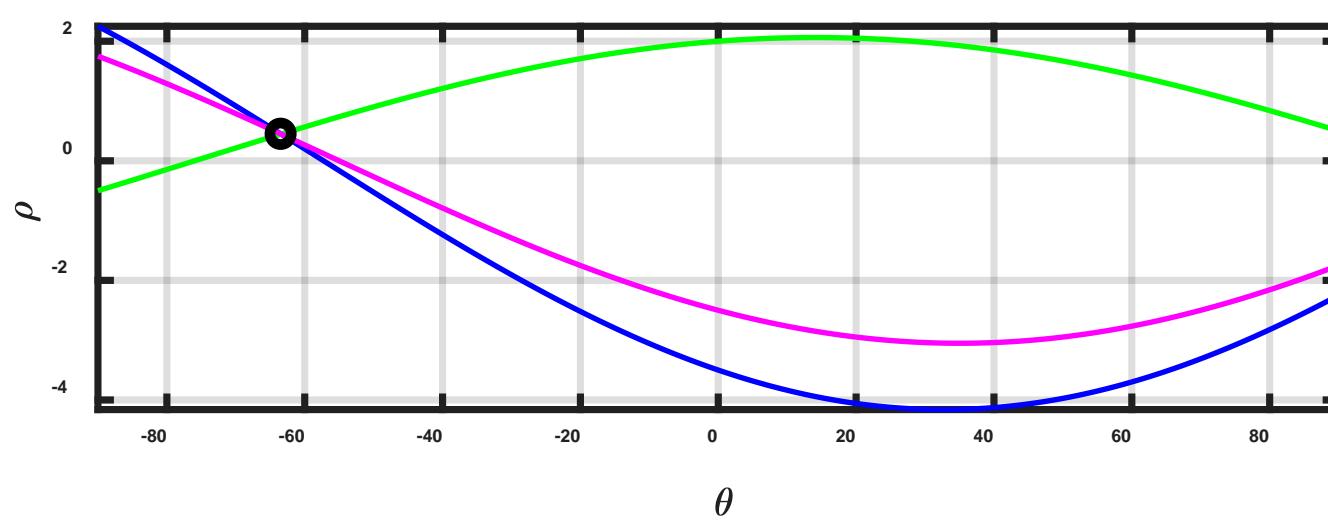
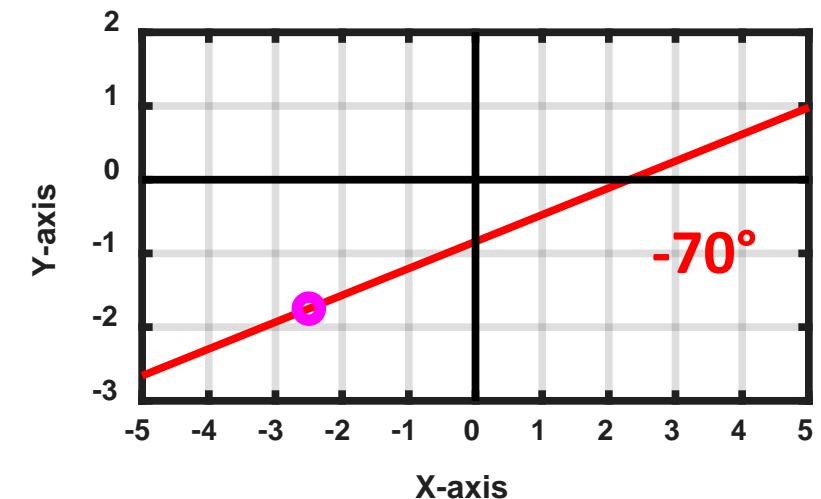
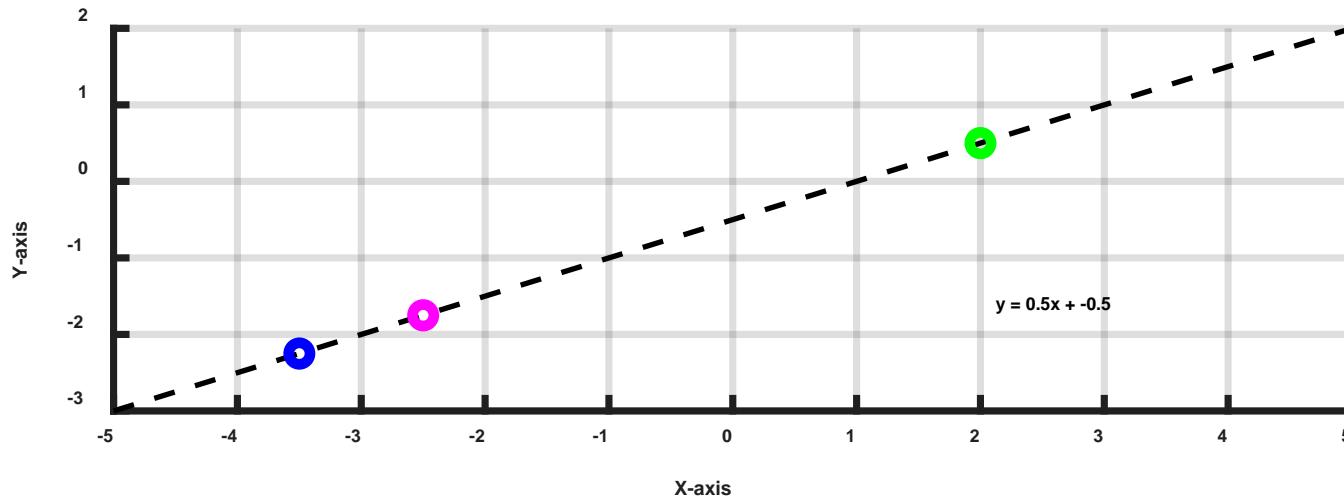
Given points  $(x_i, y_i)$  find  $(\rho, \theta)$

Hough Space Sinusoid

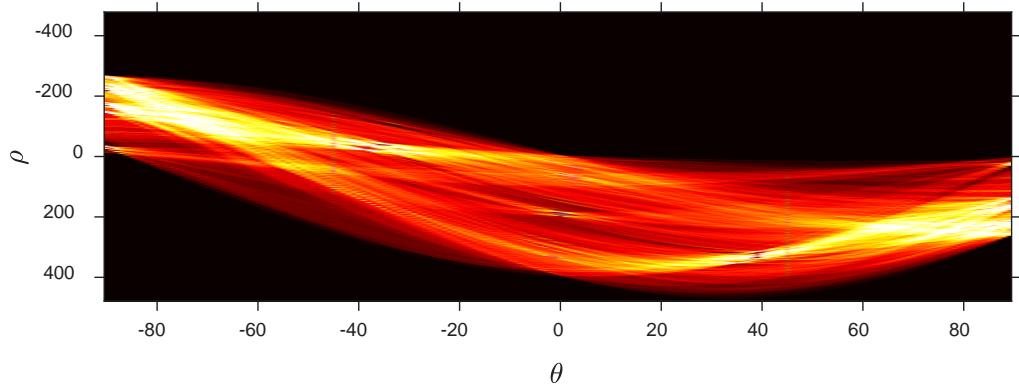


# Hough Transform Demo

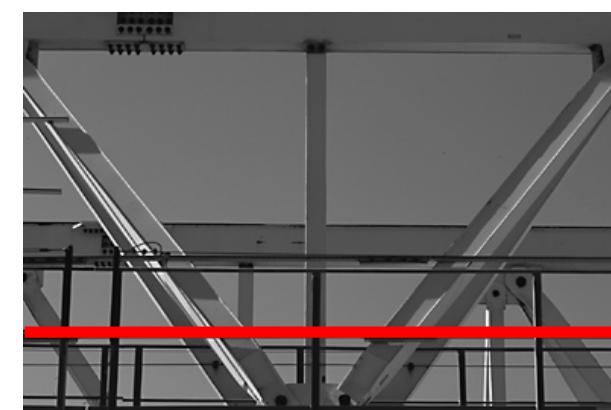
# Example: Line Passing Through Three Dots



# Example: Line Detection



1<sup>st</sup> Peak



# Mechanics of the Hough Transform

- Difficulties
  - how big should the cells be? (too big, and we merge quite different lines; too small, and noise causes lines to be missed)
  - How many lines?
    - Count the peaks in the Hough array
    - Treat adjacent peaks as a single peak
  - Which points belong to each line?
    - Search for points close to the line
    - Solve again for line and iterate

# Slide Credits and References

- Lecture notes: S. Narasimhan
- Lecture notes: Gordon Wetzstein
- Lecture notes: Noah Snavely
- Lecture notes: L. Fei-Fei
- Lecture notes: D. Frosyth
- Lecture notes: James Hayes
- Lecture notes: Yacov Hel-Or