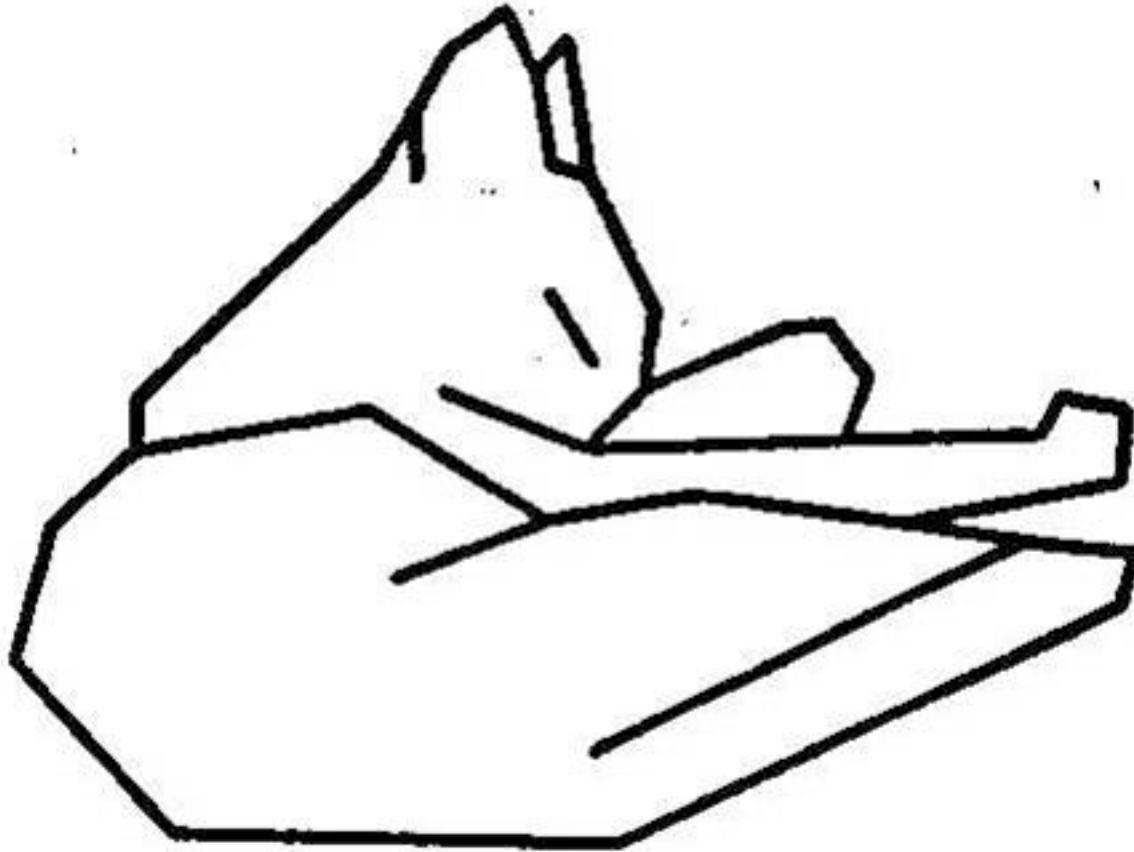


Edge Detection

CSE 576

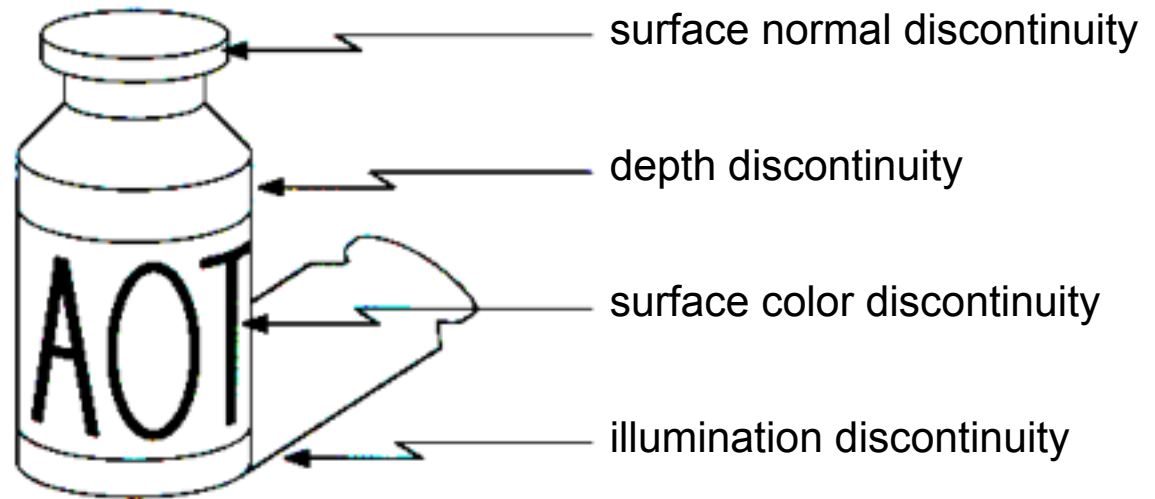
Ali Farhadi

Edge



Attneave's Cat (1954)

Origin of edges

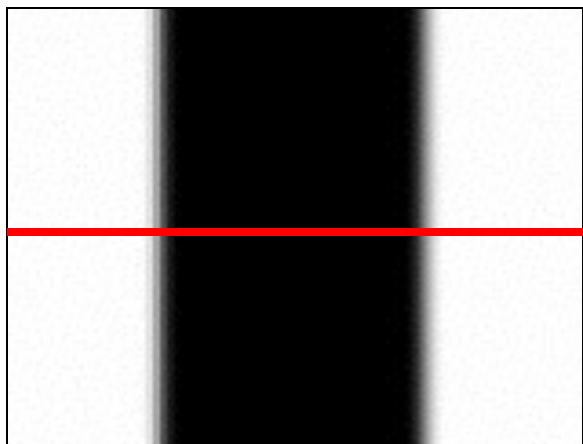


Edges are caused by a variety of factors

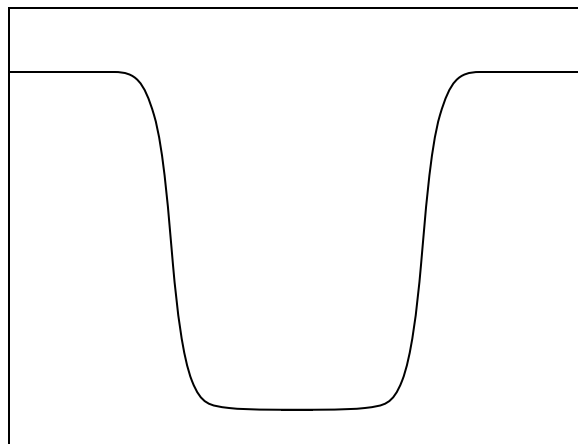
Characterizing edges

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

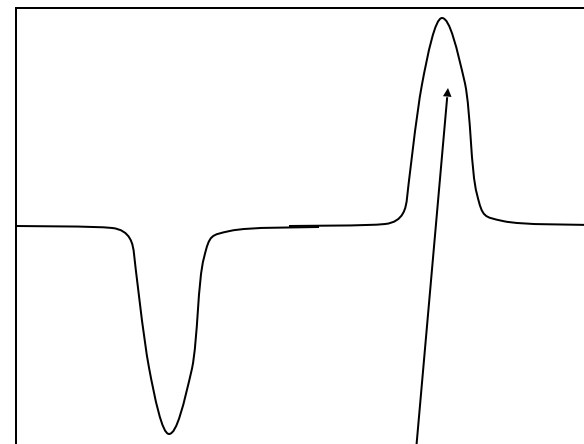
image



intensity function
(along horizontal scanline)



first derivative



edges correspond to
extrema of derivative

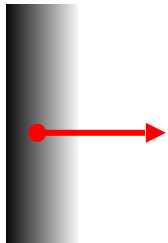
Image gradient



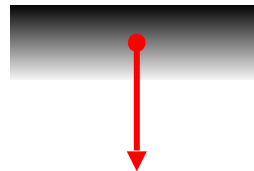
- The gradient of an image:

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

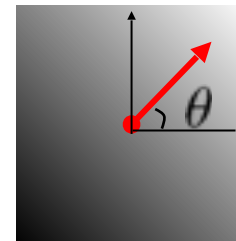
- The gradient points in the direction of most rapid change in intensity



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0 \right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y} \right]$$



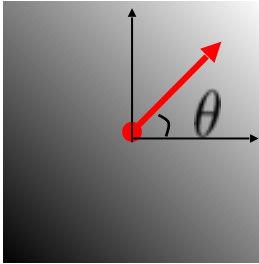
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

The discrete gradient

- How can we differentiate a *digital* image $F[x, y]$?
 - Option 1: reconstruct a continuous image, then take gradient
 - Option 2: take discrete derivative (“finite difference”)

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

Image gradient



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

$$\frac{\partial f}{\partial x} = f(x + 1, y) - f(x, y)$$

How would you implement this as a filter?

The gradient direction is given by:

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

How does this relate to the direction of the edge?

The *edge strength* is given by the gradient magnitude

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

Sobel operator

In practice, it is common to use:

$$g_x = \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$g_y = \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

Magnitude:

$$g = \sqrt{g_x^2 + g_y^2}$$

Orientation:

$$\Theta = \tan^{-1} \left(\frac{g_y}{g_x} \right)$$

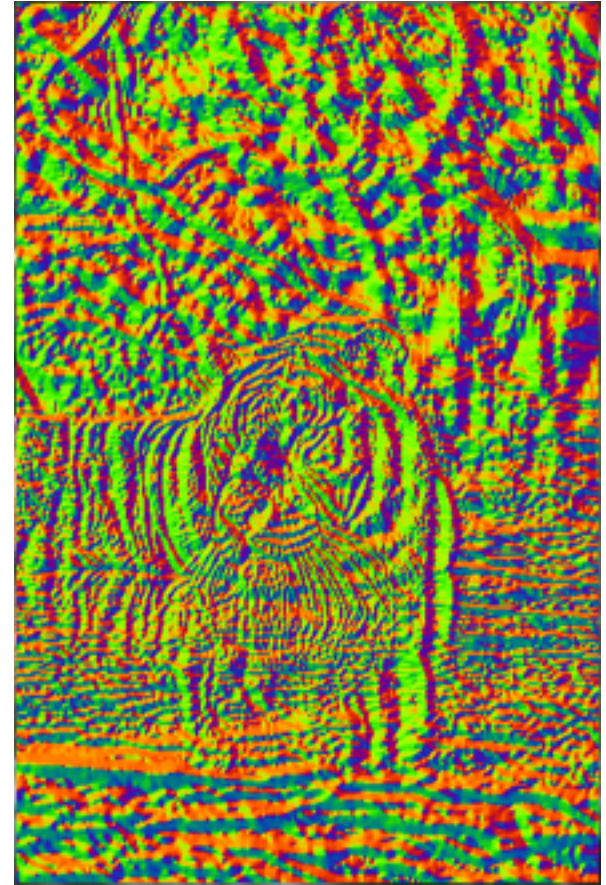
Sobel operator



Original



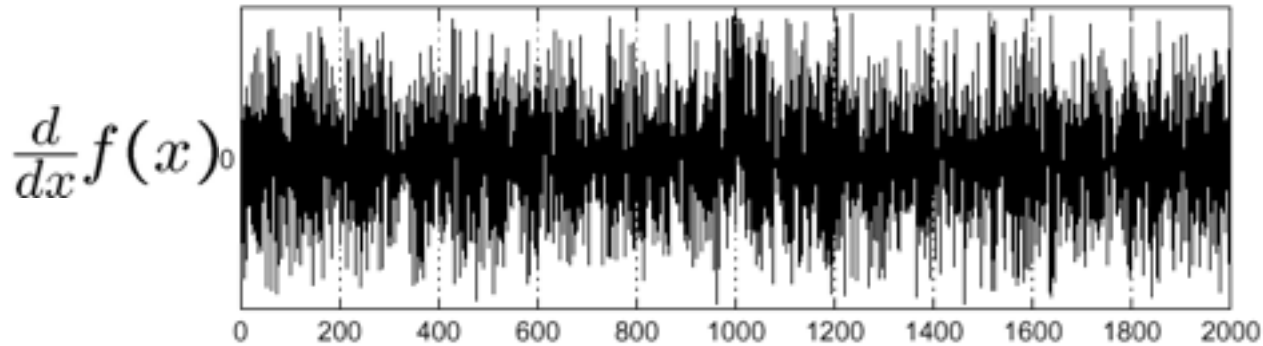
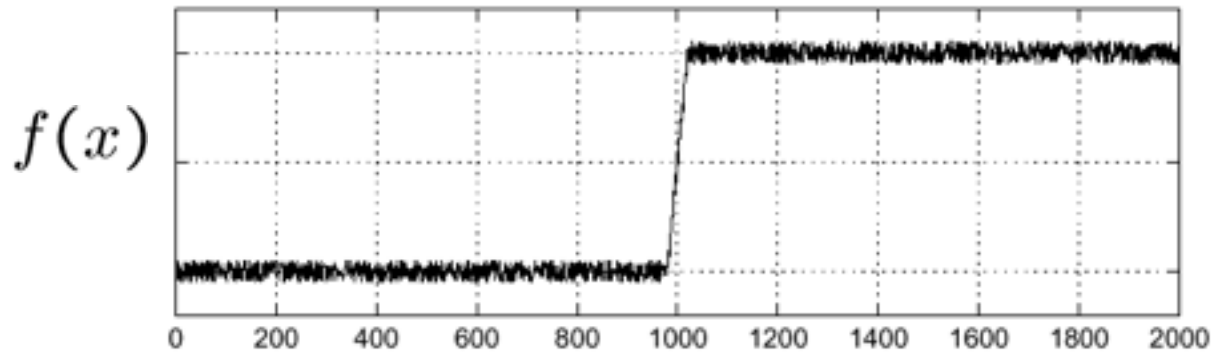
Magnitude



Orientation

Effects of noise

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

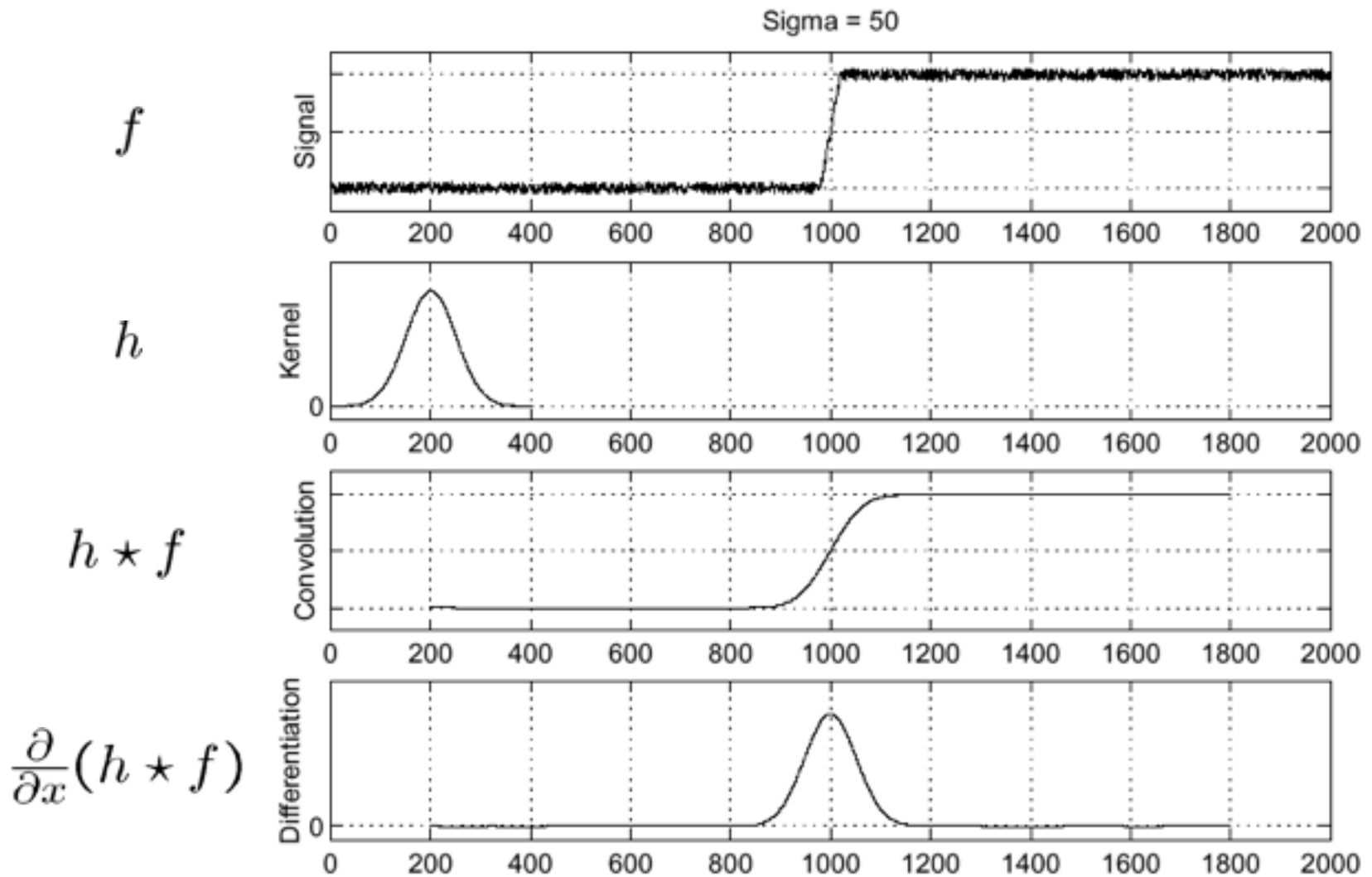


Where is the edge?

Effects of noise

- Difference filters respond strongly to noise
 - Image noise results in pixels that look very different from their neighbors
 - Generally, the larger the noise the stronger the response
- What can we do about it?

Solution: smooth first

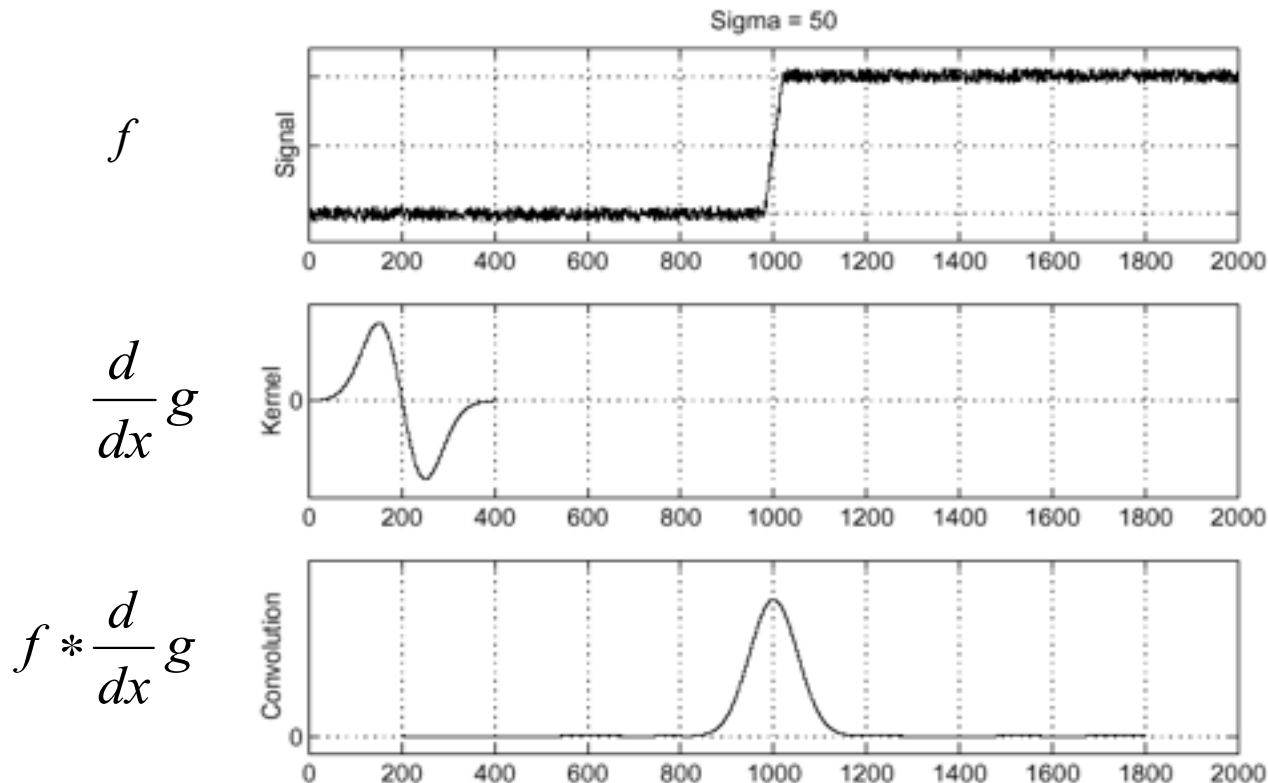


Where is the edge?

Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

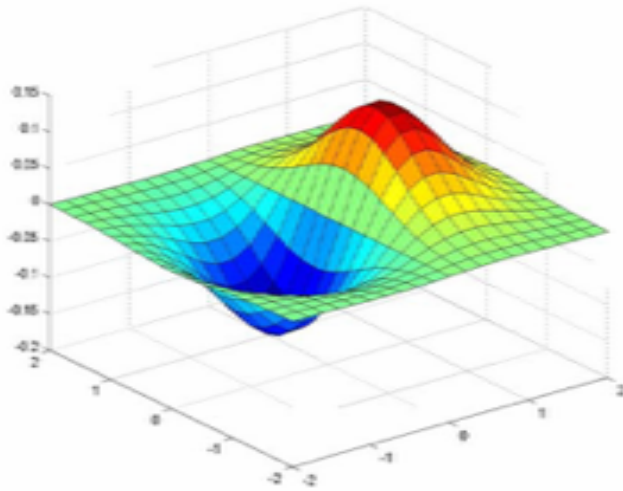
Derivative theorem of convolution

- Differentiation is convolution, and convolution is associative: $\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$
- This saves us one operation:

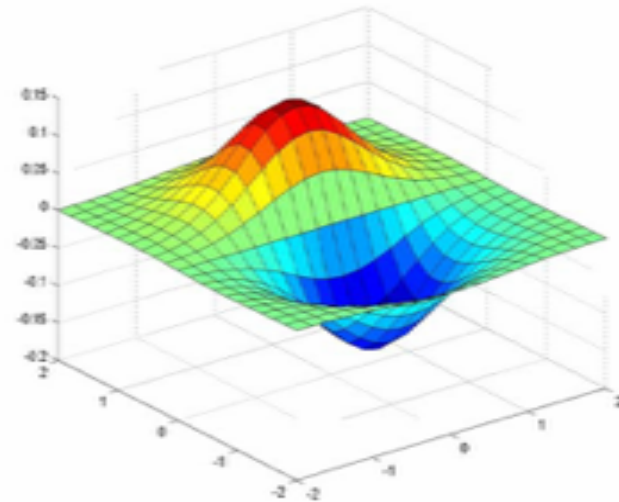


How can we find (local) maxima of a function?

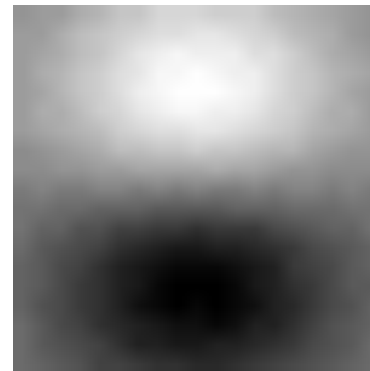
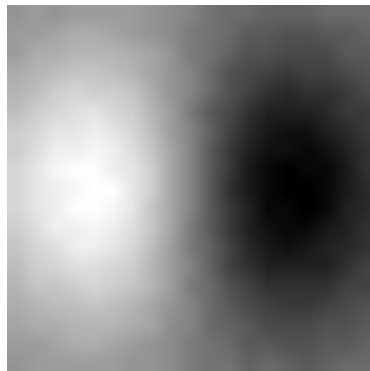
Remember: Derivative of Gaussian filter



x-direction



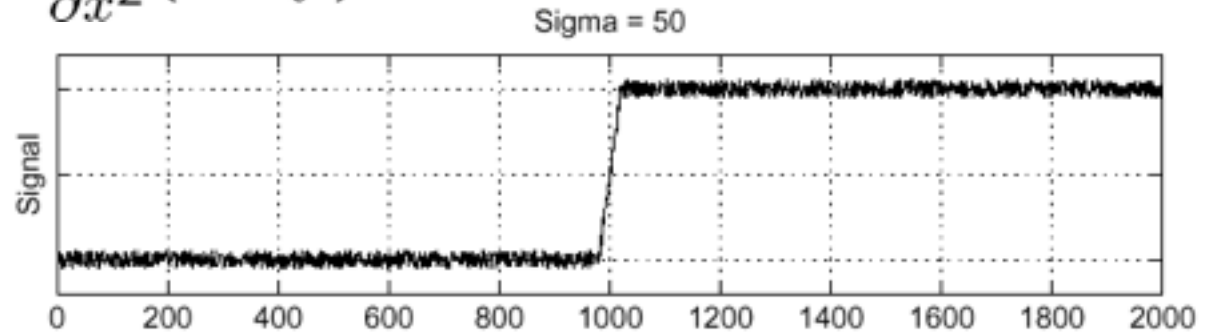
y-direction



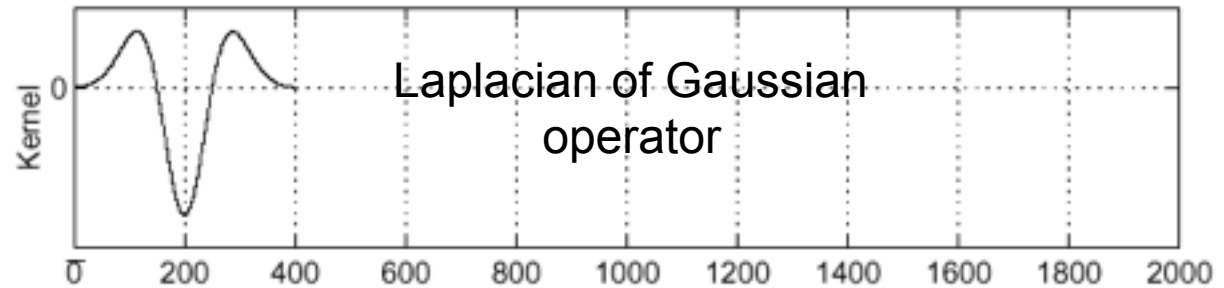
Laplacian of Gaussian

- Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

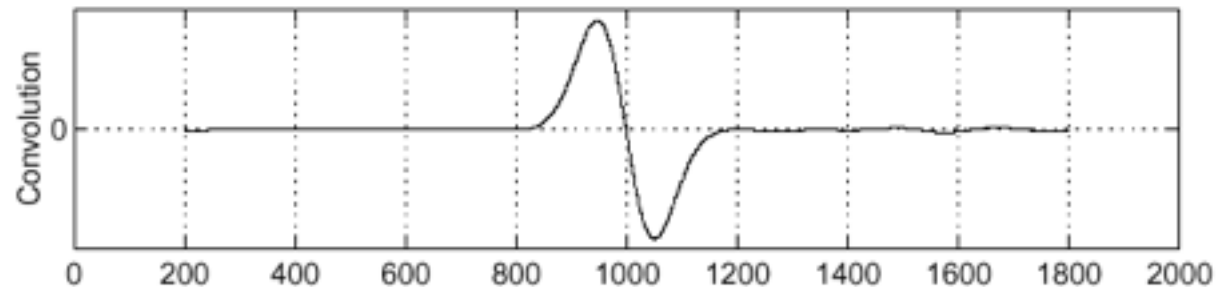
f



$\frac{\partial^2}{\partial x^2}h$



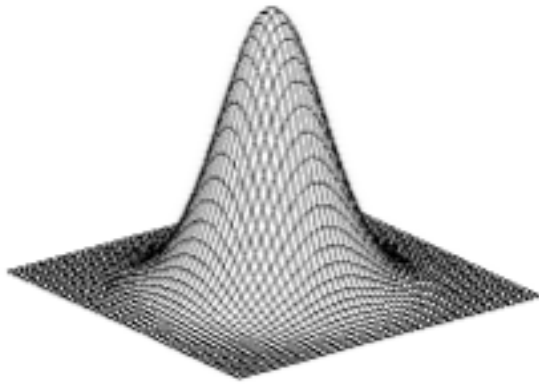
$(\frac{\partial^2}{\partial x^2}h) \star f$



Where is the edge?

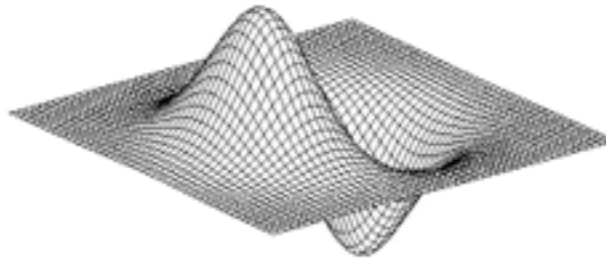
Zero-crossings of bottom graph

2D edge detection filters



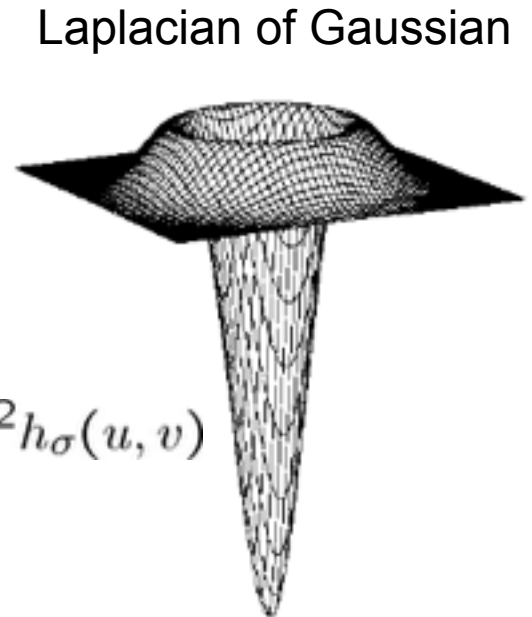
Gaussian

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



derivative of Gaussian

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



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

∇^2 is the **Laplacian** operator:

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

Edge detection by subtraction



original

Edge detection by subtraction



smoothed (5x5 Gaussian)

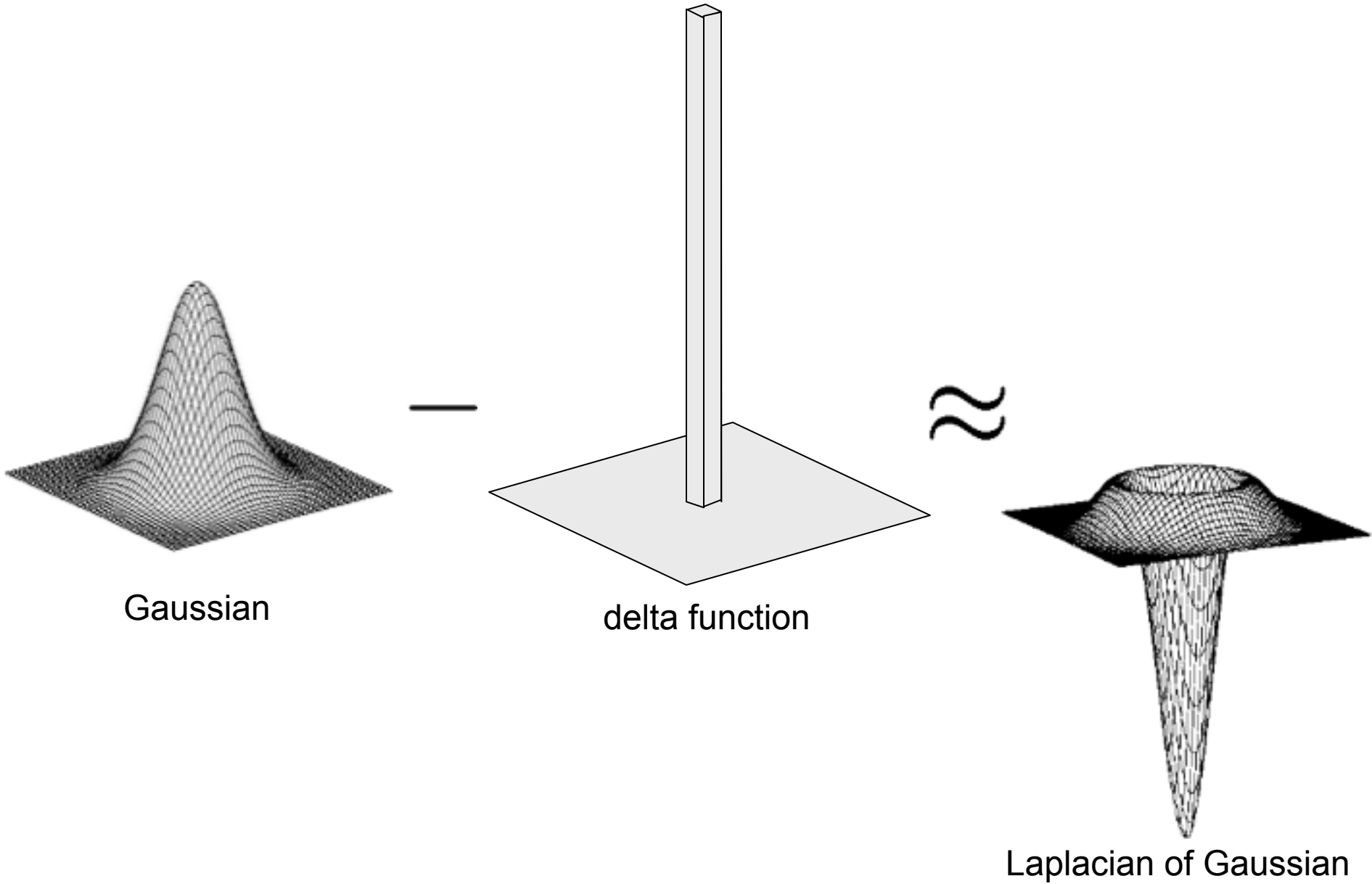
Edge detection by subtraction



Why does
this work?

smoothed – original
(scaled by 4, offset +128)

Gaussian - image filter



Canny edge detector

- This is probably the most widely used edge detector in computer vision

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

The Canny edge detector



- original image (Lena)

The Canny edge detector



norm of the gradient

The Canny edge detector



thresholding

Get Orientation at Each Pixel



$$\text{theta} = \text{atan2}(-g_y, g_x)$$

The Canny edge detector

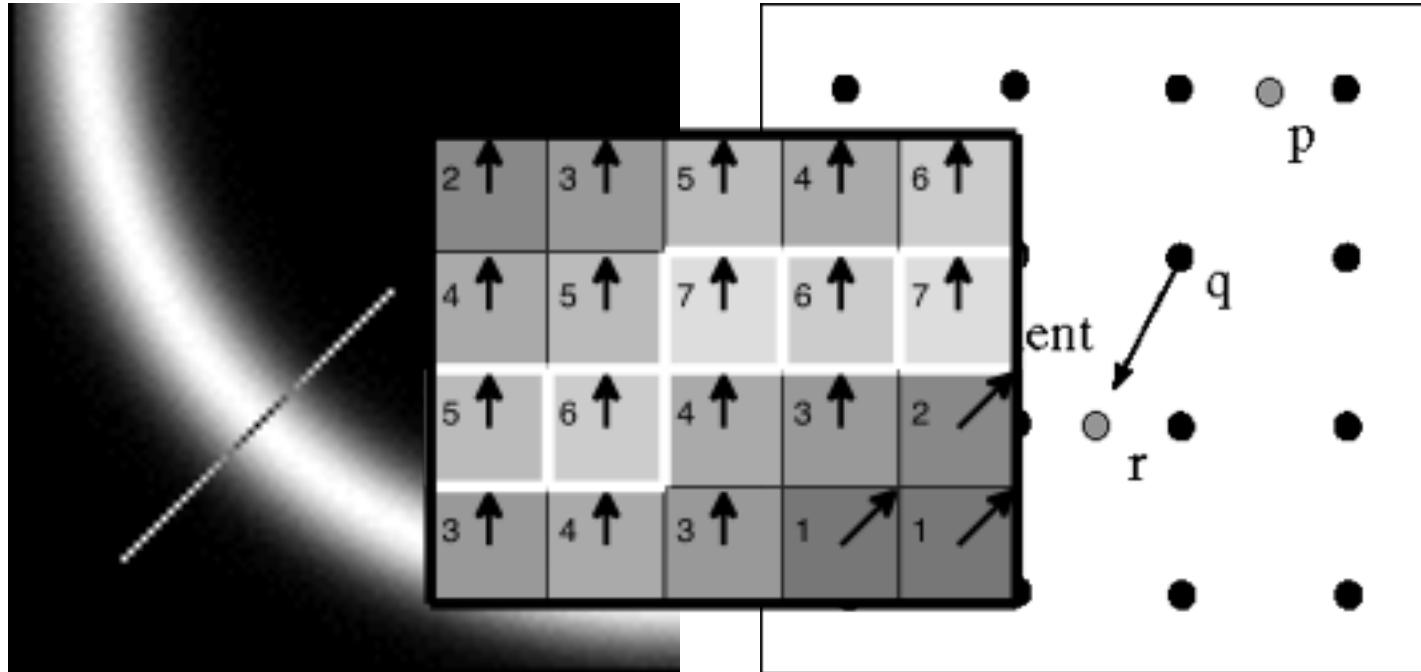


The Canny edge detector



thinning
(non-maximum suppression)

Non-maximum suppression



- Check if pixel is local maximum along gradient direction

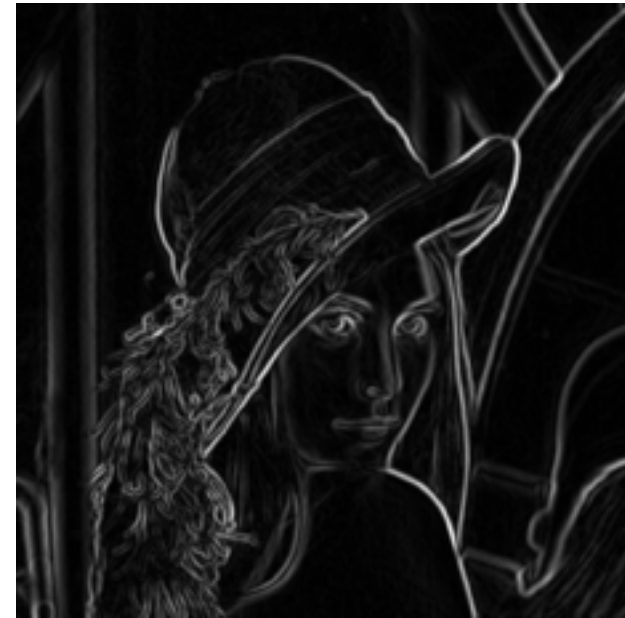
Compute Gradients (DoG)



X-Derivative of Gaussian



Y-Derivative of Gaussian



Gradient Magnitude

Canny Edges



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

An edge is not a line...

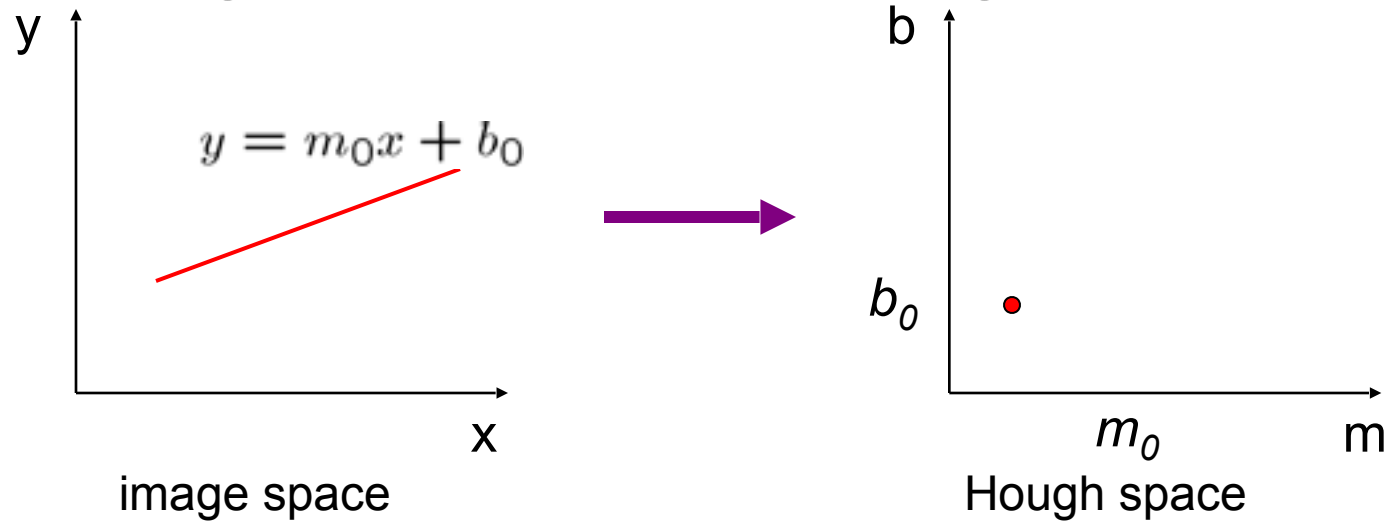


How can we detect *lines* ?

Finding lines in an image

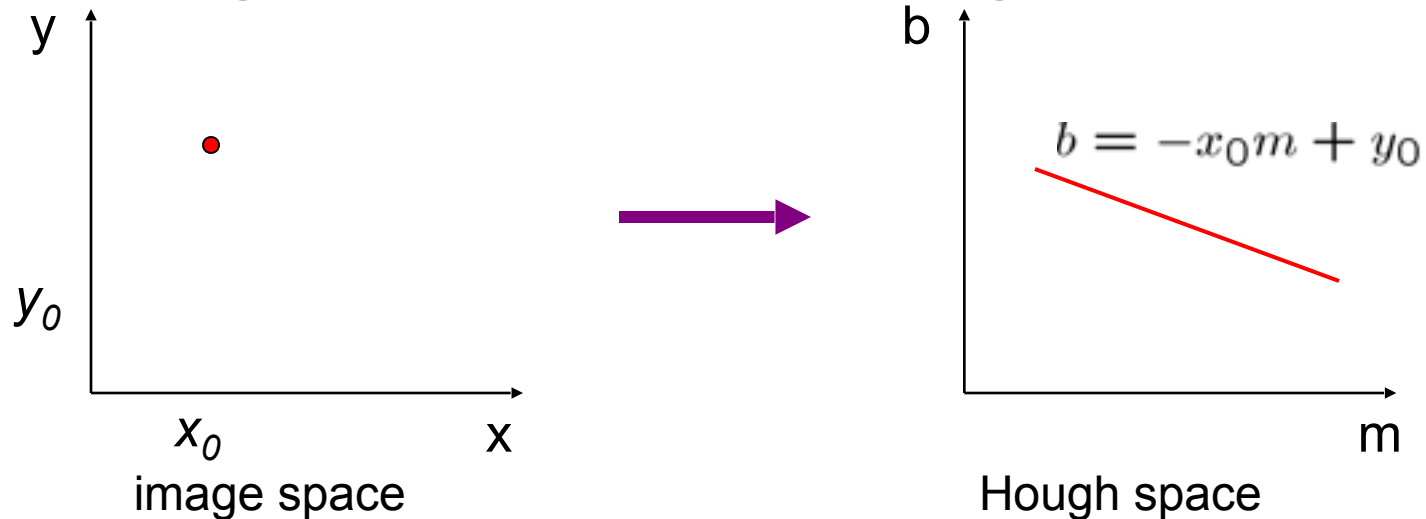
- Option 1:
 - Search for the line at every possible position/ orientation
 - What is the cost of this operation?
- Option 2:
 - Use a voting scheme: Hough transform

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 $y = mx + b$

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 $y = mx + b$
 - What does a point (x_0, y_0) in the image space map to?
 - A: the solutions of $b = -x_0m + y_0$
 - this is a line in Hough space

Hough transform algorithm

- Typically use a different

parameterization

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

- d is the perpendicular distance from the line to the origin
- θ is the angle

Hough transform algorithm

- Basic Hough transform algorithm

1. Initialize $H[d, \theta] = 0$

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

for $\theta = 0$ to 180

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

$$H[d, \theta] += 1$$

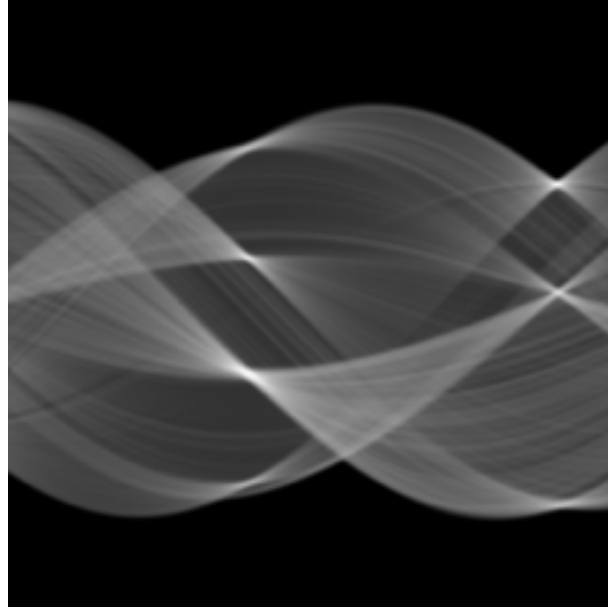
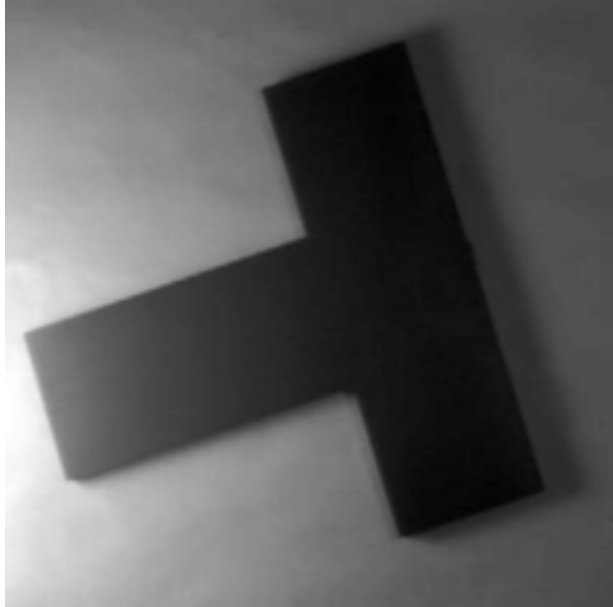
3. Find the value(s) of (d, θ) where $H[d, \theta]$ is maximum

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

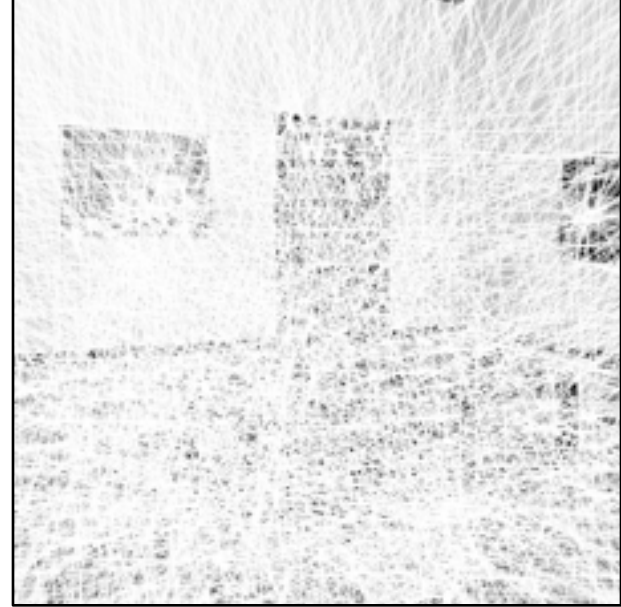
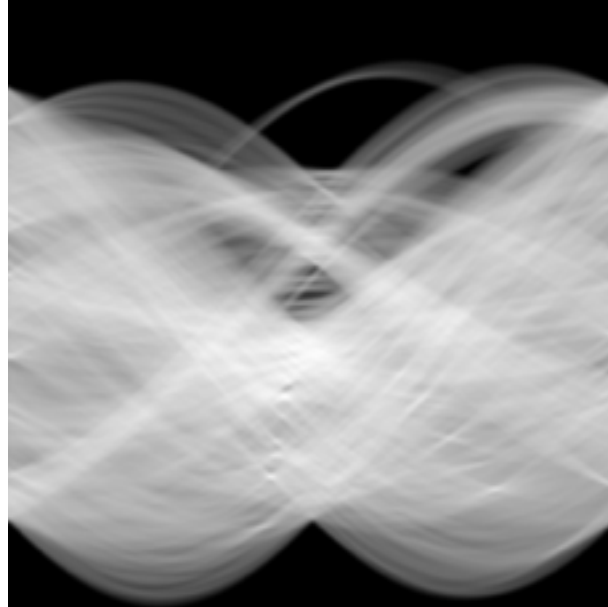
4. The detected line in the image is given by

- What's the running time (measured in # votes)?

Hough transform algorithm



Hough transform algorithm



Extensions

- Extension 1: Use the image gradient
 1. same
 2. for each edge point $I[x,y]$ in the image
 - compute unique (d, θ) based on image gradient at (x,y)
 - $H[d, \theta] += 1$
 3. same
 4. same
- What's the running time measured in votes?
- 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?