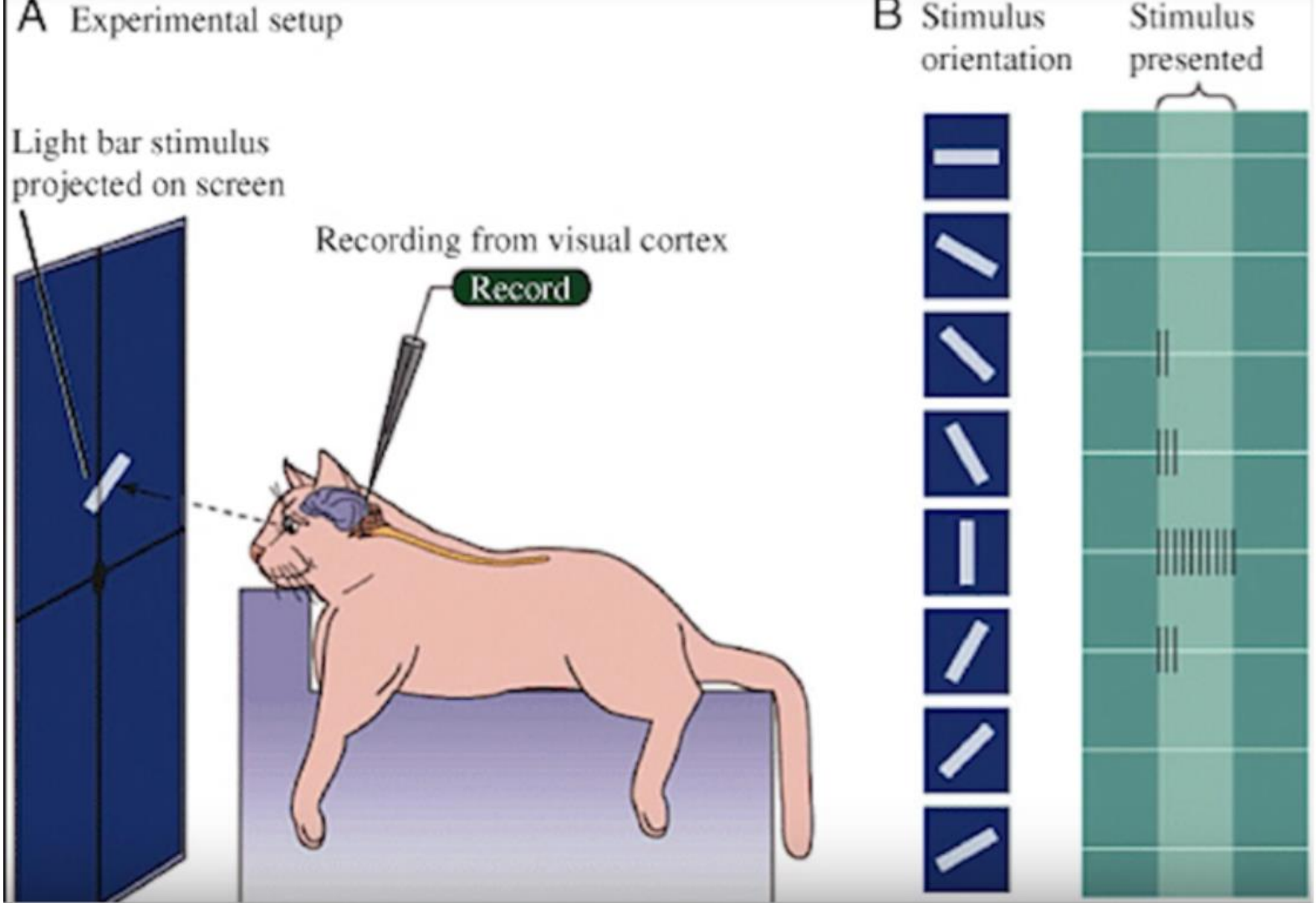
Edge detection

EE5178: Modern Computer Vision

What we will learn today

- Edge detection
- Image gradients
- A simple edge detection
- Sobel edge detector
- Canny edge detector



Hubel & Wiesel, 1960s

Definition of edges

Courtesy Trucco, Ch. 4 AND Jain et al., Ch. 5

- Edges are significant local changes of intensity in an image.
- Edges typically occur on the boundary between two different regions in an image.

Edge detection

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

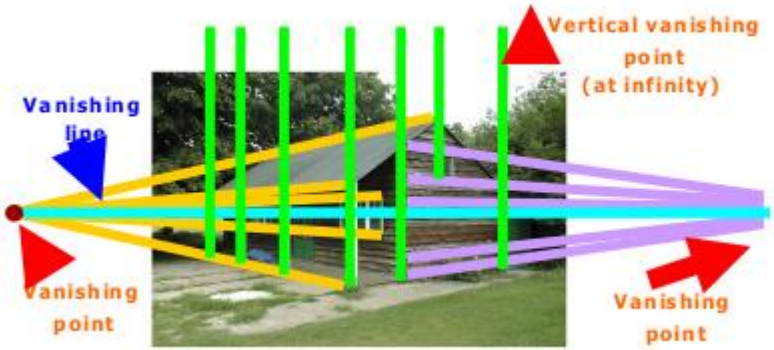
- **Ideal:** artist's line drawing (but artist is also using object-level knowledge)

Source: D. Lowe

Why do we care about edges?

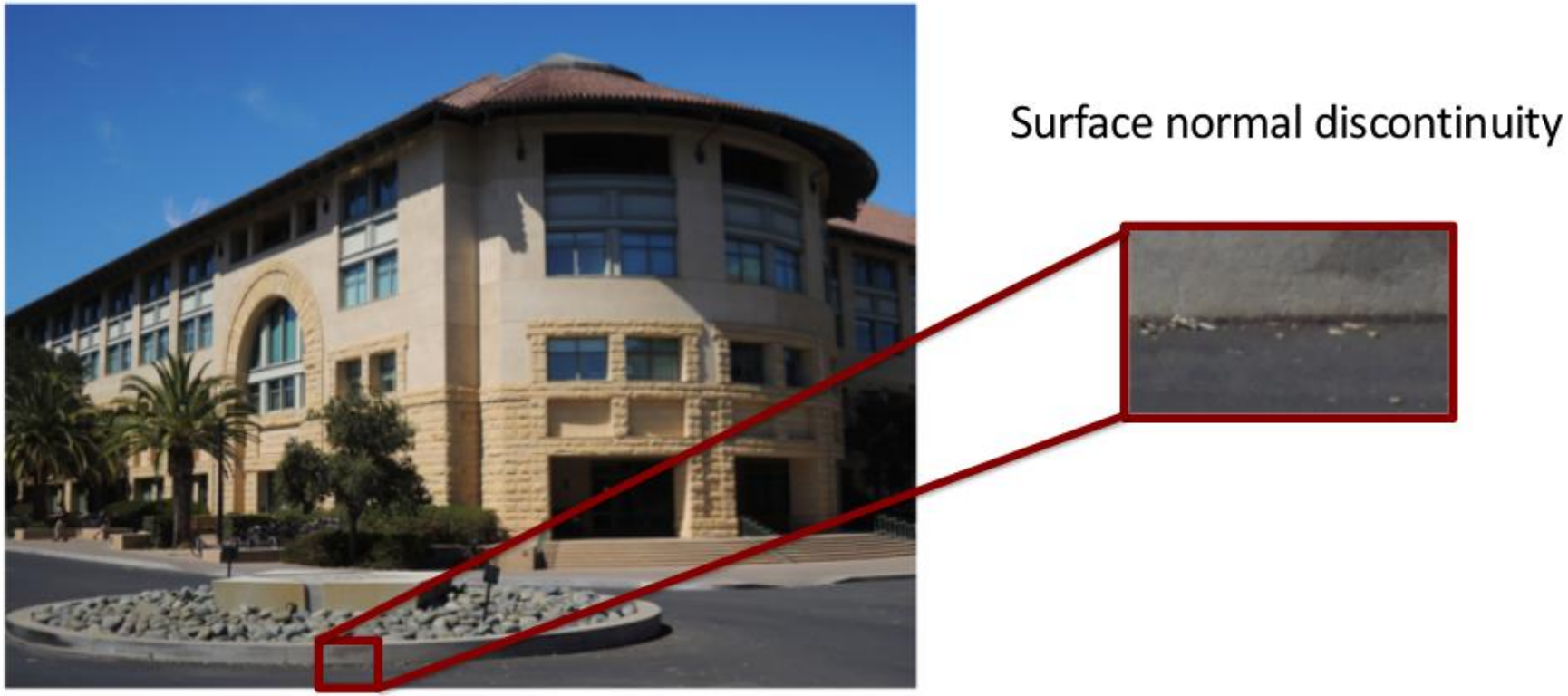
Courtesy Juan Carlos Niebles and Ranjay Krishna

- Extract information, recognize objects
- Recover geometry and viewpoint



Origins of edges

Courtesy Juan Carlos Niebles and Ranjay Krishna



Origins of edges

Courtesy Juan Carlos Niebles and Ranjay Krishna

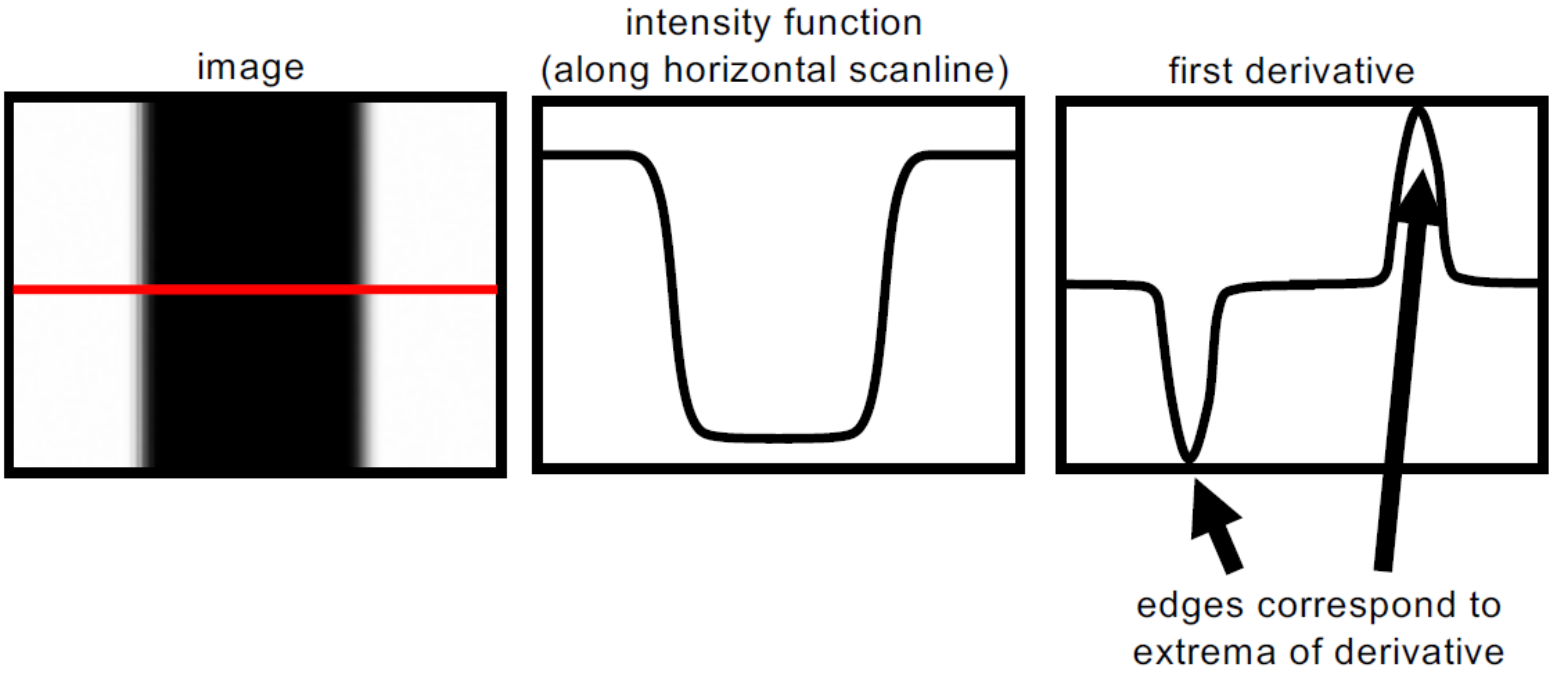
Origins of edges

Courtesy Juan Carlos Niebles and Ranjay Krishna

Surface color discontinuity

Characterizing edges

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



What we will learn today

- Edge detection
- Image gradients
- A simple edge detection
- Sobel edge detector
- Canny edge detector

Discrete Derivative in 1D

Courtesy Juan Carlos Niebles and Ranjay Krishna

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

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

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

Central

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

Computing the *first* derivative on 1D signals

Courtesy Trucco, Ch. 4 AND Jain et al., Ch. 5

mask M = [−1, 0, 1]

S_1			12	12	12	12	12	24	24	24	24	24
S_1	$\otimes$	M	0	0	0	0	12	12	0	0	0	0

(a) S_1 is an upward step edge

S_2			24	24	24	24	24	12	12	12	12	12
S_2	$\otimes$	M	0	0	0	0	-12	-12	0	0	0	0

(b) S_2 is a downward step edge

S_3			12	12	12	12	15	18	21	24	24	24
S_3	$\otimes$	M	0	0	0	3	6	6	6	3	0	0

(c) S_3 is an upward ramp

S_4			12	12	12	12	24	12	12	12	12	12
S_4	$\otimes$	M	0	0	0	12	0	-12	0	0	0	0

(d)

Discrete derivative in 2D

Courtesy Juan Carlos Niebles and Ranjay Krishna

Given function $f(x, y)$

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

Gradient magnitude $$\left|\nabla f(x,y)\right| = \sqrt{f_x^2 + f_y^2}$$

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

What does this filter do?

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

What about this filter?

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

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

Discrete derivative in 2D - Example

Courtesy Juan Carlos Niebles and Ranjay Krishna

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

Discrete derivative in 2D - Example

Courtesy Juan Carlos Niebles and Ranjay Krishna

What happens when we apply this filter?

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

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

Discrete derivative in 2D - Example

Courtesy Juan Carlos Niebles and Ranjay Krishna

What happens when we apply this filter?

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

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

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

Detects edges in Horizontal direction

Discrete derivative in 2D - Example

Courtesy Juan Carlos Niebles and Ranjay Krishna

Now let's try the other filter!

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

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

Discrete derivative in 2D - Example

Courtesy Juan Carlos Niebles and Ranjay Krishna

What happens when we apply this filter?

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

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

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

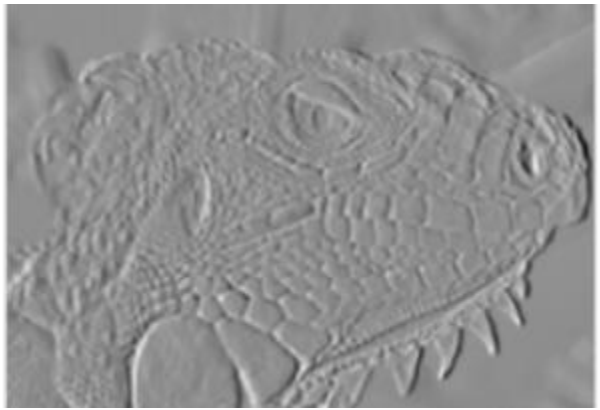
Detects edges in Vertical direction

Edge Detection using 1st Derivative (Gradients)

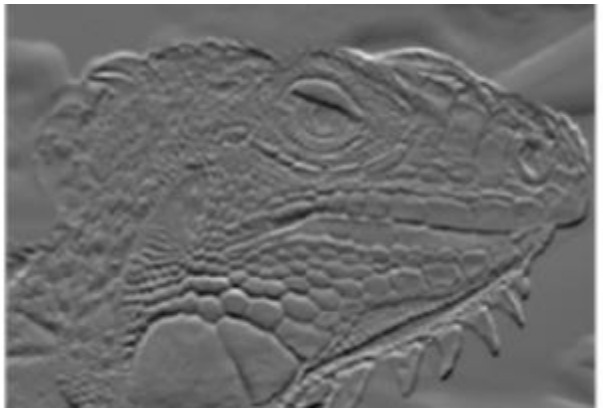
Courtesy Juan Carlos Niebles and Ranjay Krishna

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

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



Derivative in x-direction



Derivative in y-direction

What we will learn today

- Edge detection
- Image gradients
- A simple edge detection
- Sobel edge detector
- Canny edge detector

Characterizing edges

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

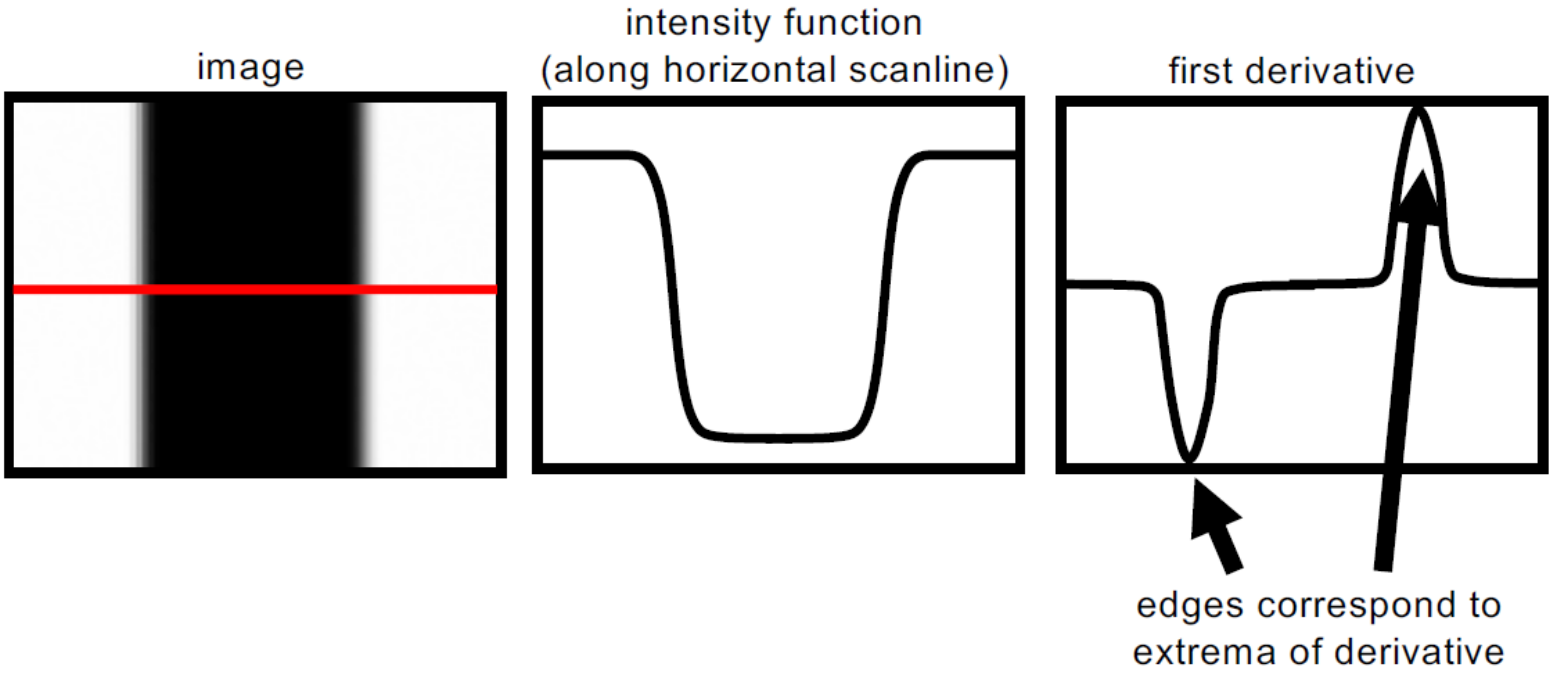
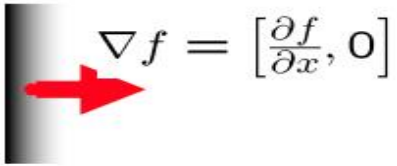
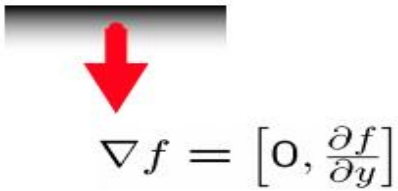
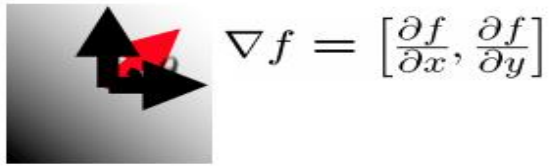


Image gradients

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







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

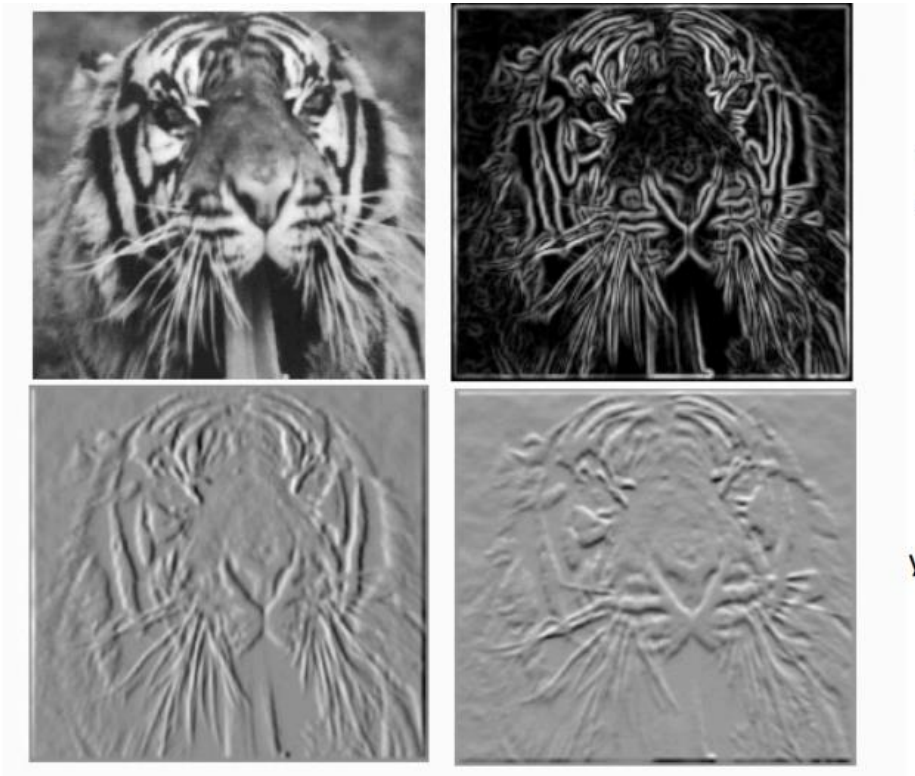
The gradient direction is given by $\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$

- how does this relate to the direction of the edge?

The *edge strength* is given by the gradient magnitude

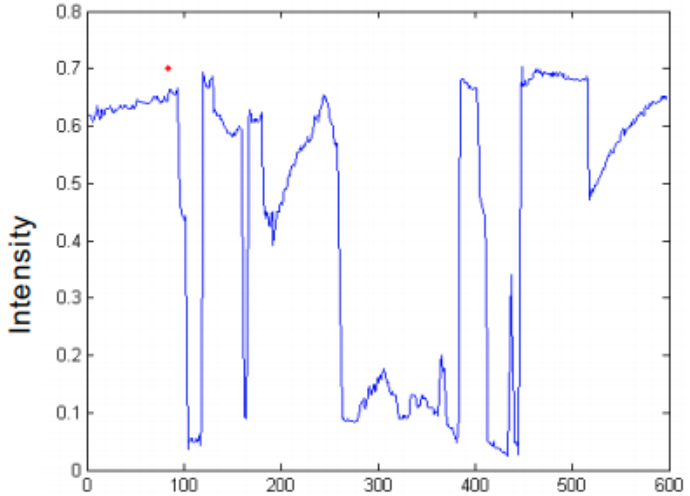
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

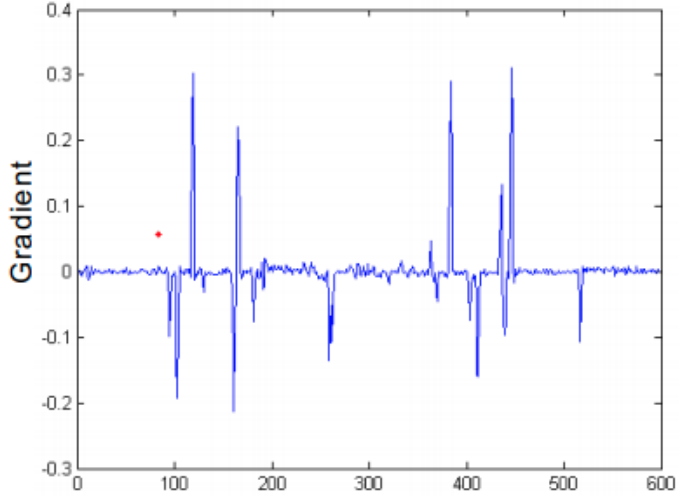
Finite differences: example



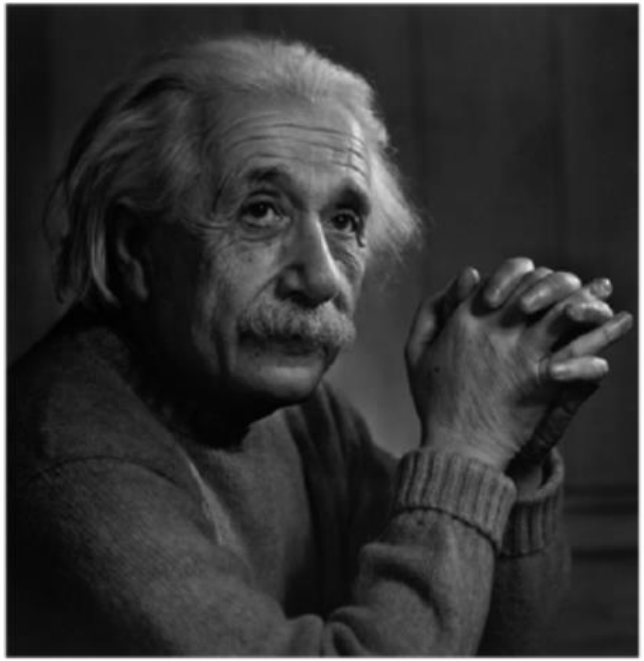
Which one is the gradient in the x-direction? How about y-direction?

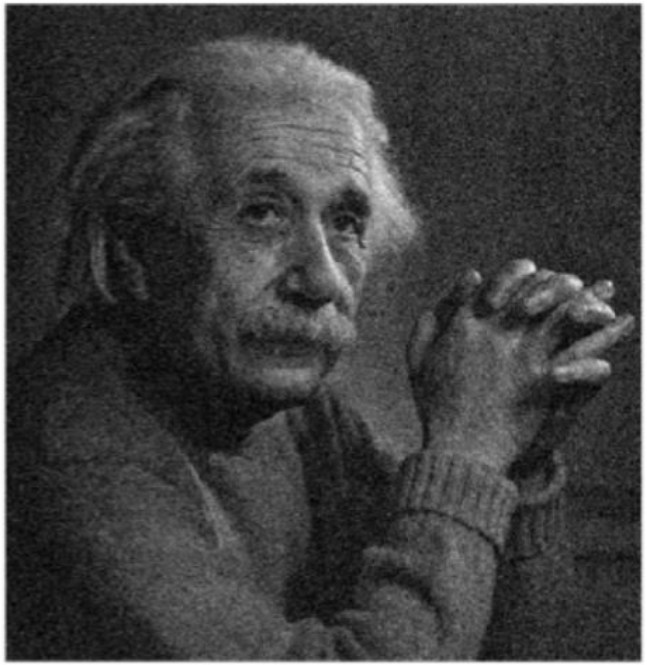
Intensity profile





Effects of noise

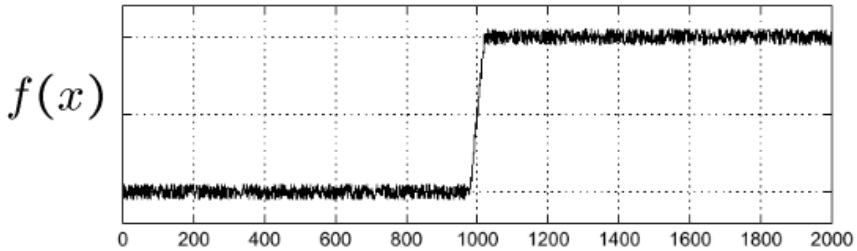




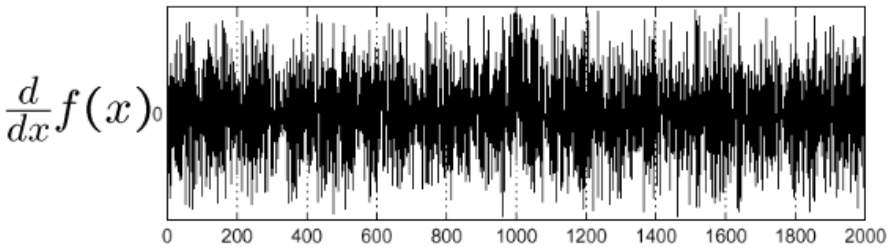
Some practical issues: Effect of NOISE

Courtesy Juan Carlos Niebles and Ranjay Krishna

Consider a single row or column of the image



– Plotting intensity as a function of position gives a signal



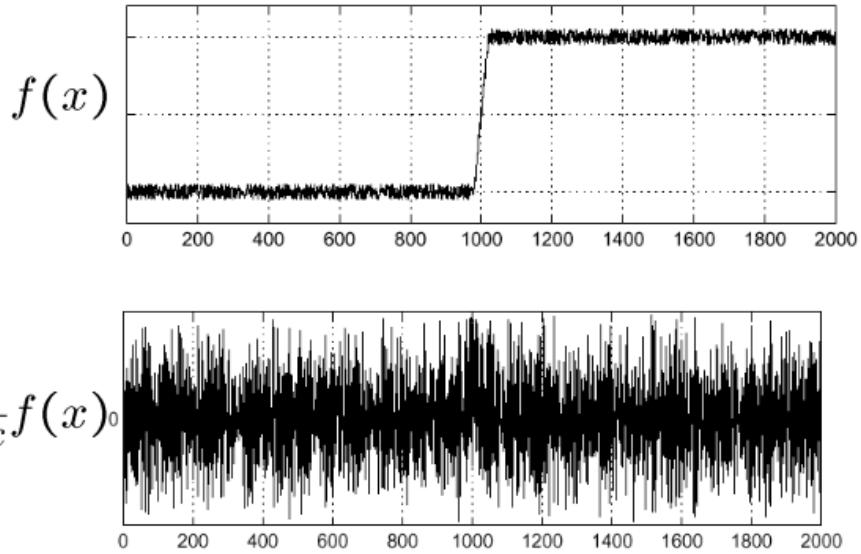
– Where is the edge?

Some practical issues: Effect of NOISE

Courtesy Juan Carlos Niebles and Ranjay Krishna

Finite difference filters respond strongly to noise

– Generally, the larger the noise the stronger the response



Derivatives Amplifies NOISE

Courtesy Saad J Bedros

$$I(x) = \hat{I}(x) + N(x) \qquad N(x) \sim N(0,\sigma) \text{ i.i.d}$$

Taking differences:

$$I'(x) \cong \hat{I}(x+1) + N(x+1) \; - (\hat{I}(x-1) + N(x-1)) \; =$$

$$= \underbrace{(\hat{I}(x+1) - \hat{I}(x-1))}_{\hat{I}'(x)} + \underbrace{(N(x+1) - N(x-1))}_{N_d(x)}$$

Output noise: $E(N_d(x)) = 0$

$$E(N_d{}^2(x)) = E(N^2(x+1) + N^2(x-1) + 2N(x+1)N(x-1)) =$$

$$= \sigma^2 + \sigma^2 + 0 = 2\sigma^2$$

Increases noise !!

Effect of NOISE: What is to be done?

Courtesy Juan Carlos Niebles and Ranjay Krishna

Smoothing the image should help, by forcing pixels different to their neighbors (=noise pixels?) to look more like neighbors.

Smoothing with different filters

Courtesy Juan Carlos Niebles and Ranjay Krishna

- Mean smoothing

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

- Gaussian (smoothing * derivative)

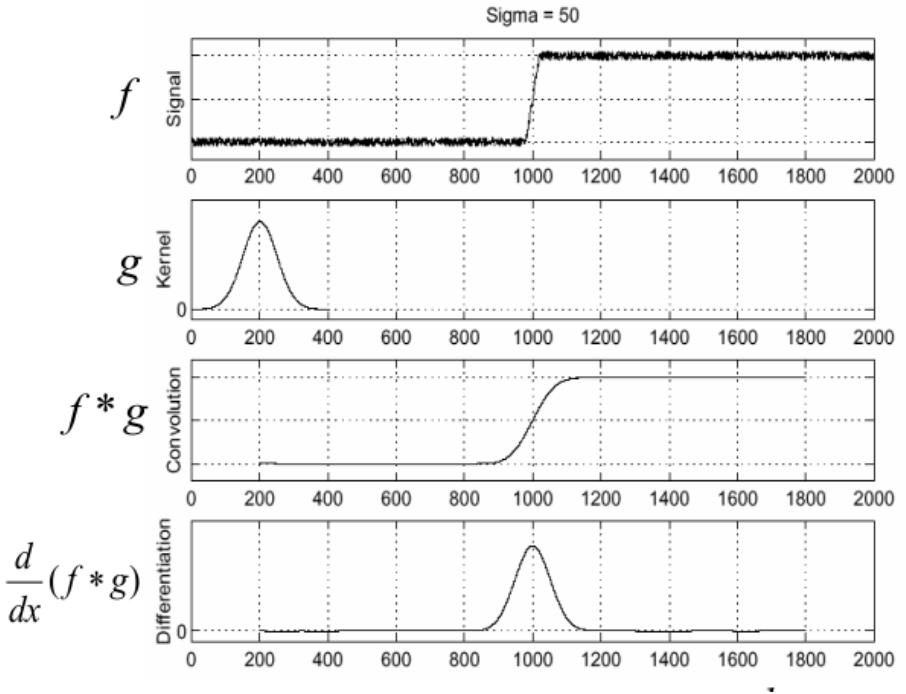
$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Solution: smooth first

Courtesy Juan Carlos Niebles and Ranjay Krishna

Image -> Smoothen the image ->

Take the derivative to detect Edges

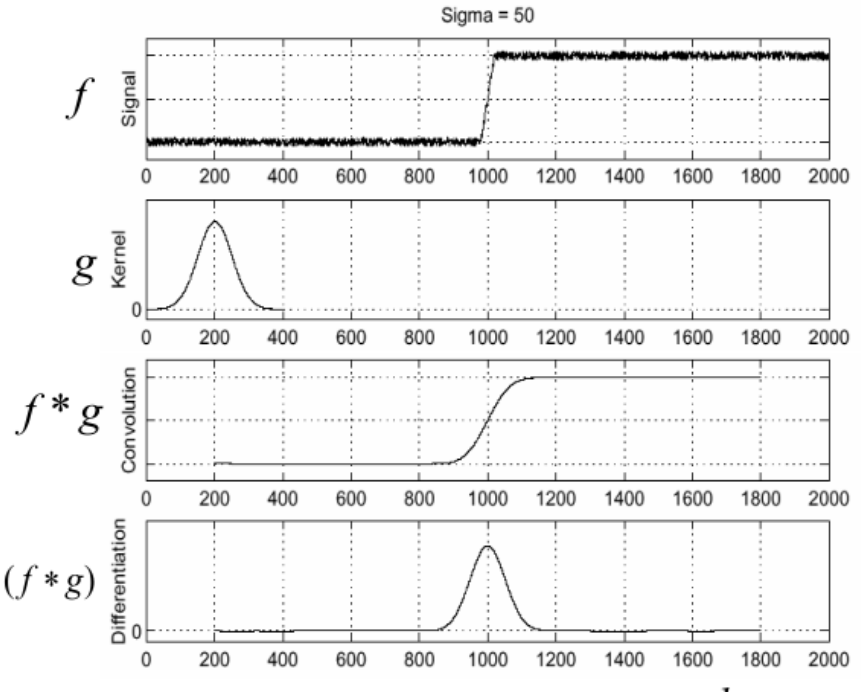


Derivative theorem of convolution - This saves us one operation

Courtesy Juan Carlos Niebles and Ranjay Krishna

Image -> Smoothen the image ->
Take the derivative to detect Edges

These two steps can be combined.



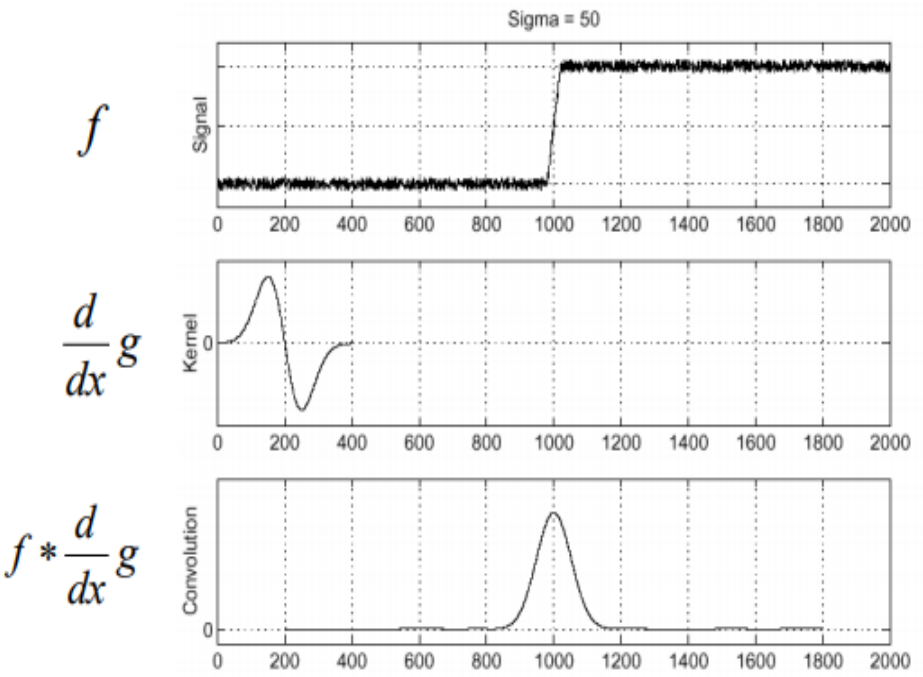
Derivative theorem of convolution - This saves us one operation

Courtesy Juan Carlos Niebles and Ranjay Krishna

Image -> CONVOLVE with APPROPRIATE KERNEL

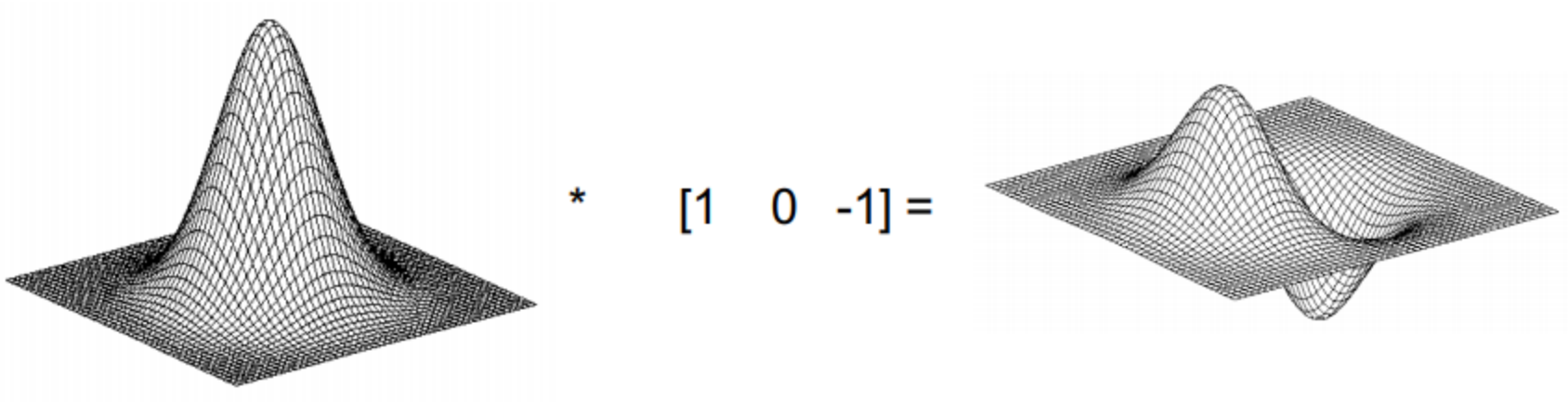
This theorem gives us a very useful property:

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



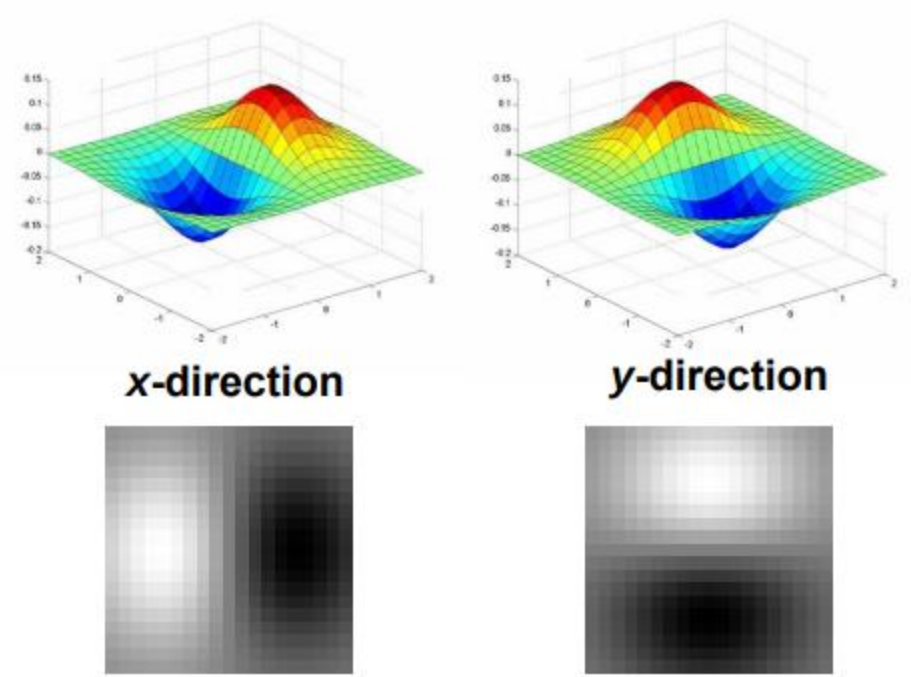
Derivative of Gaussian filter

Courtesy Juan Carlos Niebles and Ranjay Krishna



Derivative of Gaussian filter

Courtesy Juan Carlos Niebles and Ranjay Krishna



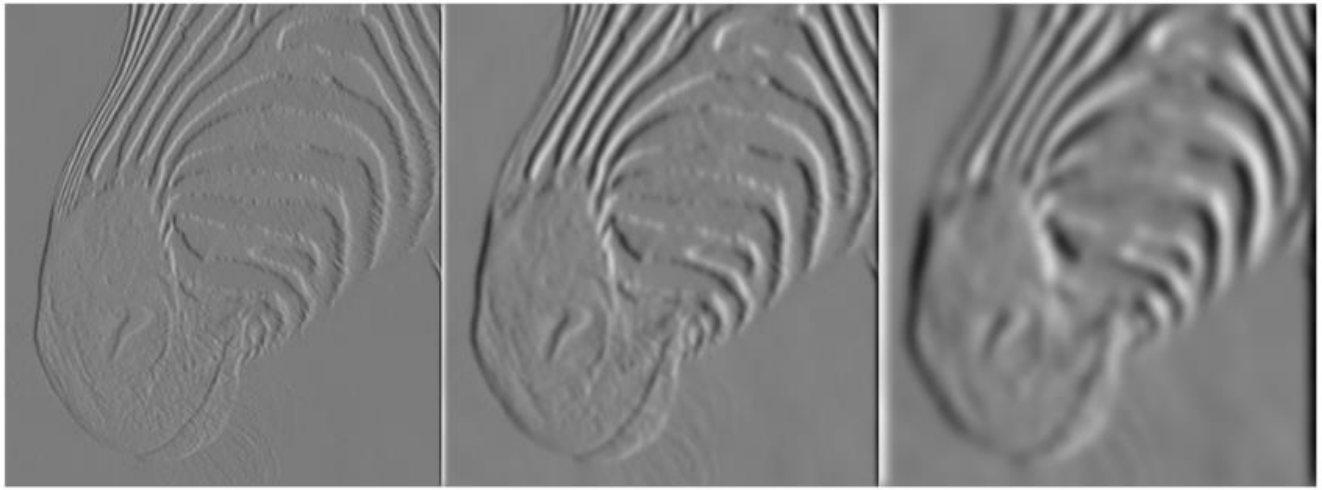
Derivative of Gaussian filter

Tradeoff between smoothing at different scales

Courtesy Juan Carlos Niebles and Ranjay Krishna

Smoothed derivative removes

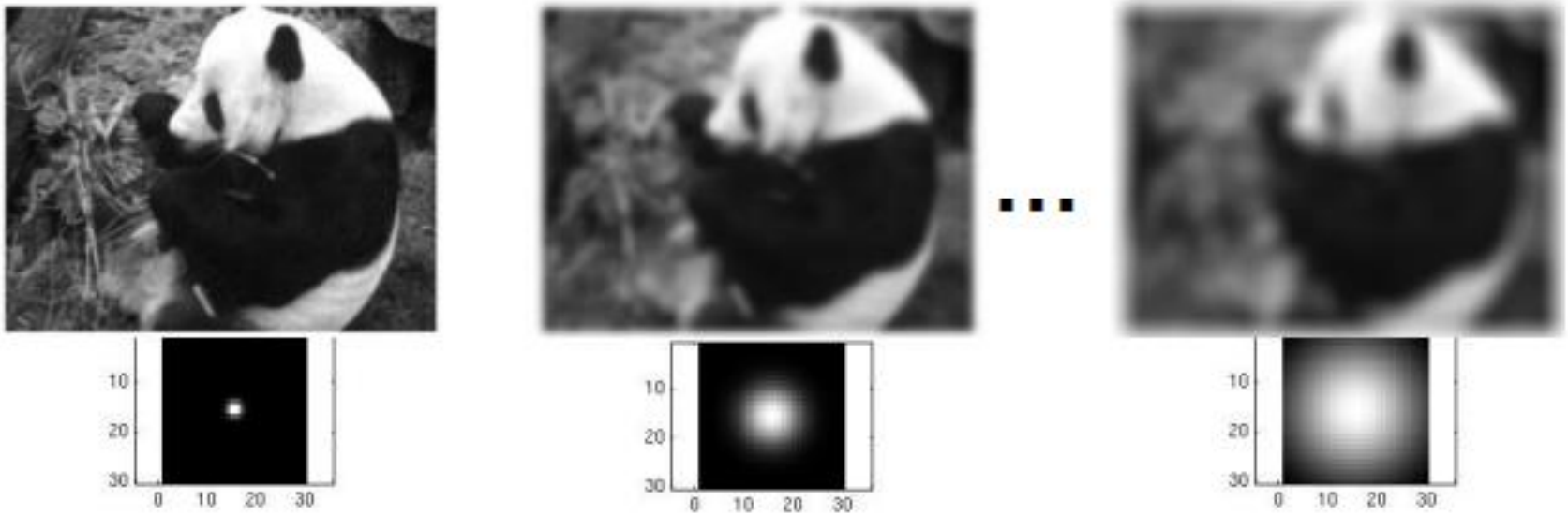
noise, but blurs edge.



Smoothing with a Gaussian

Courtesy Saad J Bedros

Recall: parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



Effect of σ on derivatives

Courtesy Saad J Bedros

σ = 1 pixel **σ = 3 pixels**

The apparent structures differ depending on Gaussian's scale parameter.

Larger values: larger scale edges detected
Smaller values: finer features detected

A basic edge detector

1) Filter image with x and y derivative of Gaussian

   That is, $I_x = I * \frac{dG}{dx}$ and $I_y = I * \frac{dG}{dy}$

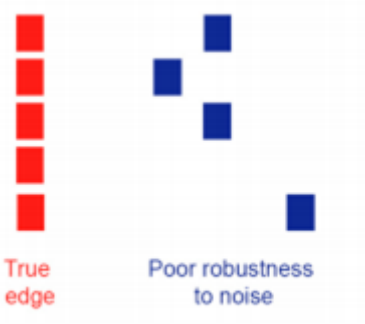
1) Find magnitude of gradient: $|\nabla I| = \sqrt{I_x^2 + I_y^2}$

2) Threshold to keep strong edges: if $|\nabla I| > T$ then strong edge.

Designing an edge detector

Courtesy Juan Carlos Niebles and Ranjay Krishna

- **Good detection:** the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of true negatives (missing real edges).



Designing an edge detector

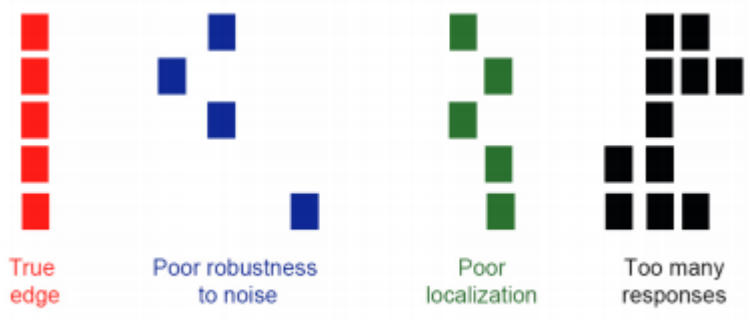
Courtesy Juan Carlos Niebles and Ranjay Krishna

- **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

Courtesy Juan Carlos Niebles and Ranjay Krishna

- **Good detection:** the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges).
- **Good localization**: the edges detected must be as close as possible to the true edges.
- **Single response**: the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge.



What we will learn today

- Edge detection
- Image gradients
- A simple edge detection
- Sobel edge detector
- Canny edge detector

Sobel Operator

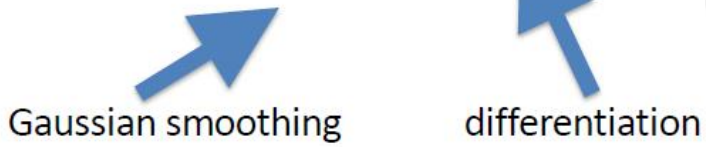
- uses two 3×3 kernels which are convolved with the original image to calculate approximations of the derivatives
- one for horizontal changes, and one for vertical

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

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

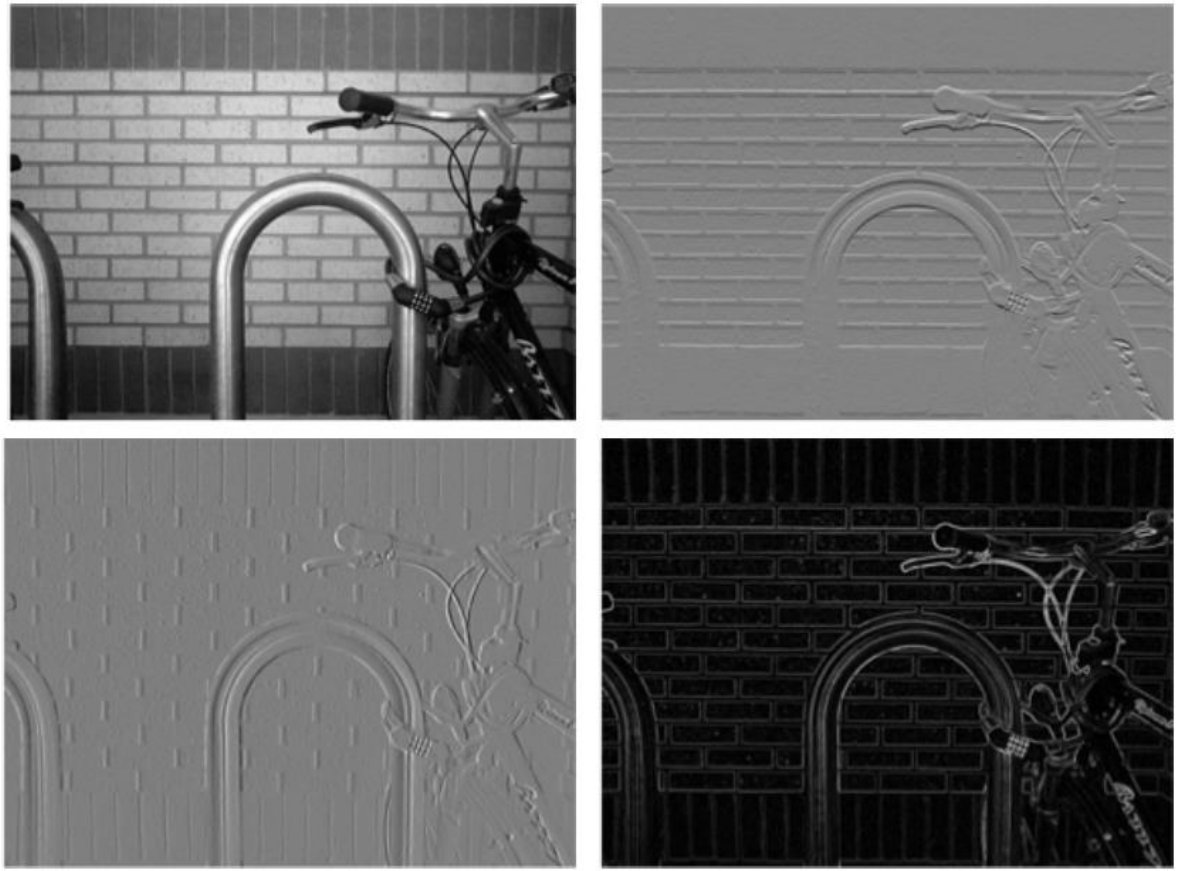
- Magnitude:

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

- Angle or direction of the gradient:

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

Sobel Filter example



Courtesy Digital Image Processing: Bernd Girod

Sum of squared horizontal and vertical gradients

threshold = 1600

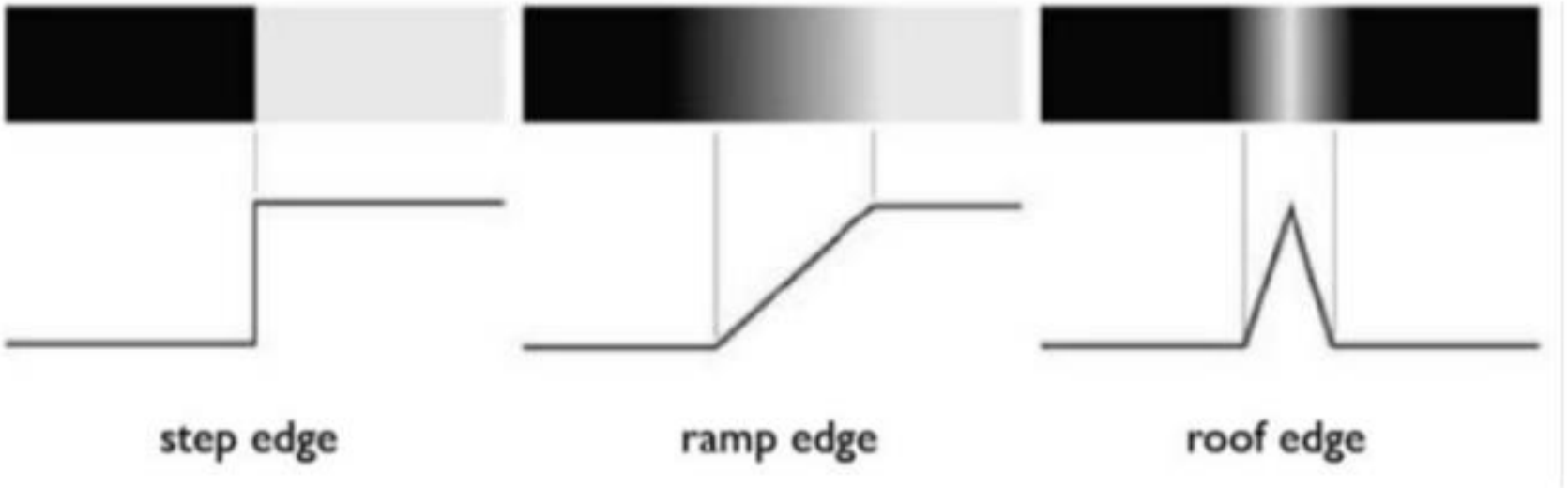
threshold = 8000

threshold = 12800

Sobel operator example

Sobel Filter Problems

Courtesy Juan Carlos Niebles and Ranjay Krishna



- Poor Localization (Trigger response in multiple adjacent pixels)

What we will learn today

- Edge detection
- Image gradients
- A simple edge detection
- Sobel edge detector
- Canny edge detector

Canny Edge Detector

Courtesy Juan Carlos Niebles and Ranjay Krishna

- This is probably the most widely used edge detector in computer vision.
- Theoretically model step-edges corrupted by additive Gaussian noise.
- Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of signal-to-noise ratio and localization.

Canny, John. "A computational approach to edge detection." ***IEEE Transactions on pattern analysis and machine intelligence*** **6 (1986): 679-698.**

Canny Edge Detector

Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna

- Suppress Noise
- Compute gradient magnitude and direction
- Apply Non-Maximum Suppression

  – Assures minimal response
- Use hysteresis and connectivity analysis to detect edges

Example

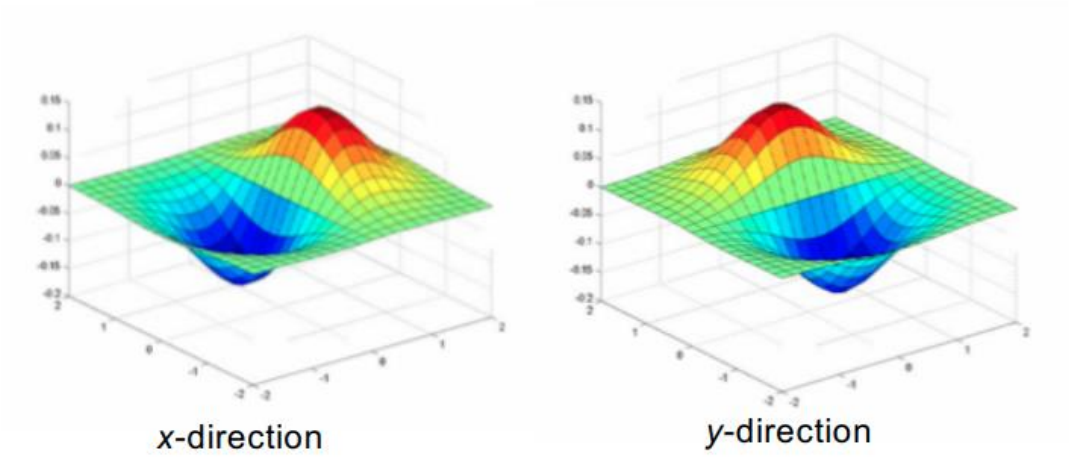
Canny Edge Detector

Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna

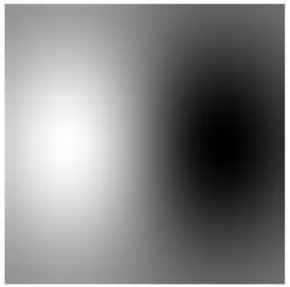
- Suppress Noise
- Compute gradient magnitude and direction
- Apply Non-Maximum Suppression
  – Assures minimal response
- Use hysteresis and connectivity analysis to detect edges

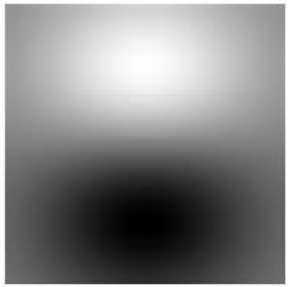
Derivative of Gaussian filter

Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna



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

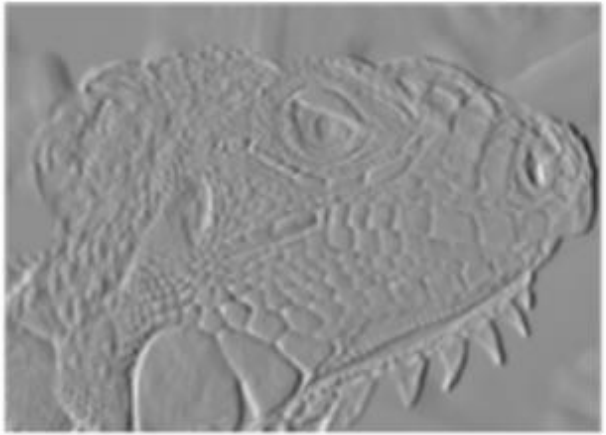




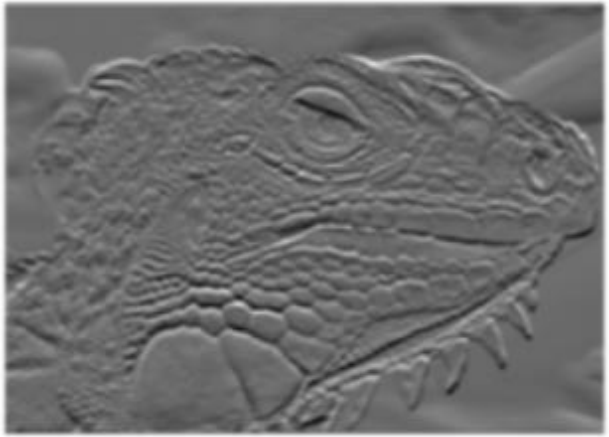
$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Compute gradients (DoG)

Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna



X-Derivative of Gaussian



Y-Derivative of Gaussian

Gradient Magnitude

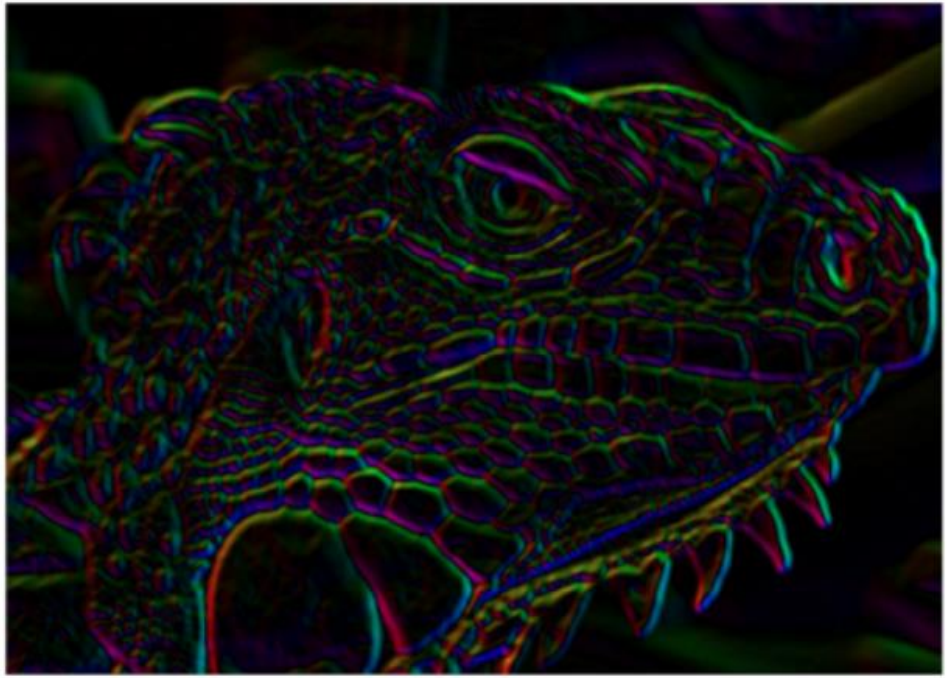
Canny Edge Detector

Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna

- Suppress Noise
- Compute gradient magnitude and direction
- Apply Non-Maximum Suppression
  – Assures minimal response
- Use hysteresis and connectivity analysis to detect edges

Get orientation at each pixel

Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna

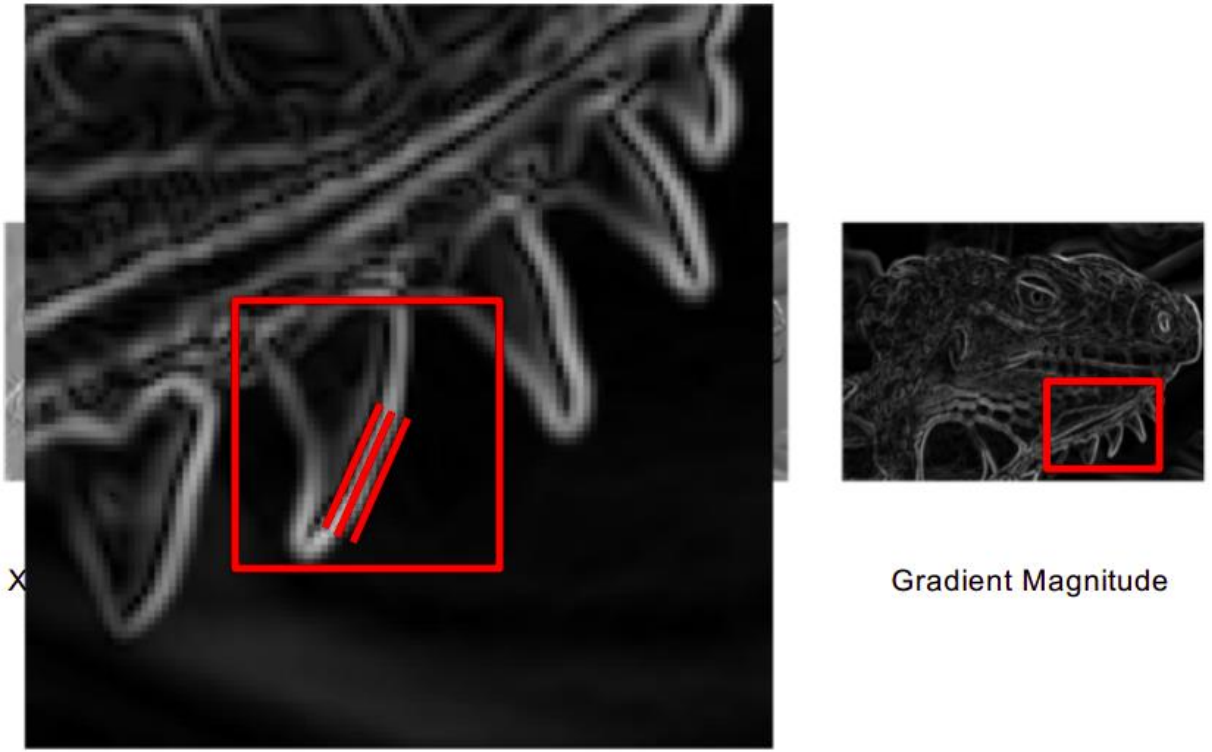


$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

Source: J. Hayes

Compute gradients (DoG)

Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna



Canny Edge Detector

Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna

- Suppress Noise
- Compute gradient magnitude and direction
- Apply Non-Maximum Suppression
  – Assures minimal response
- Use hysteresis and connectivity analysis to detect edges

Apply Non-Maximum Suppression

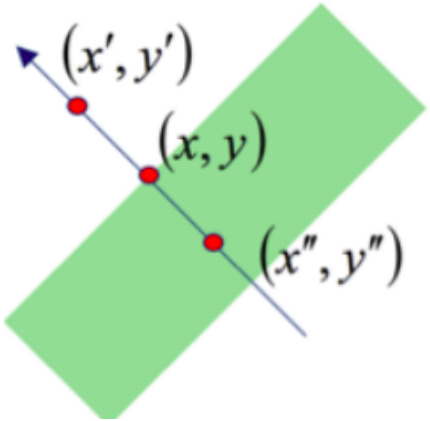
Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna

- Edge occurs where gradient reaches a maxima
- Suppress non-maxima gradient even if it passes threshold
- Only eight angle directions possible
  - Suppress all pixels in each direction which are not maxima
  - Do this in each marked pixel neighborhood

Remove spurious gradients

Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna

$|\nabla G|(x, y)$ is the gradient at pixel (x, y)

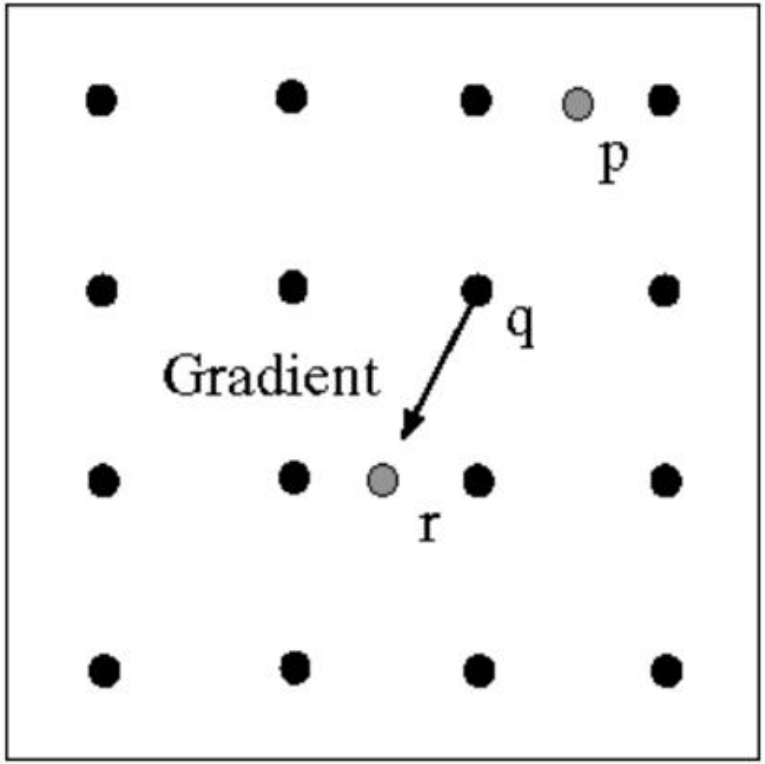


$$M(x,y) = \begin{cases} |\nabla G|(x,y) \text{ if } |\nabla G|(x,y) > |\nabla G|(x',y') \\ \quad\quad\quad\quad\quad\quad\quad\quad \&|\nabla G|(x,y) > |\nabla G|(x'',y'') \\ 0 \quad\quad\quad\quad \text{otherwise} \end{cases}$$

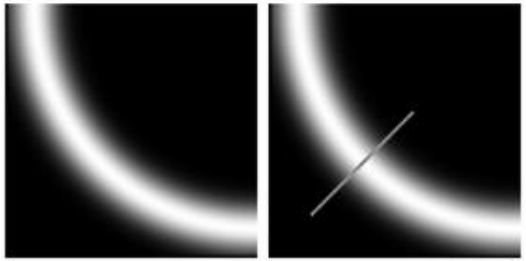
x' and **x"** are the neighbors of **x** along normal direction to an edge

Non Maxima Suppression

Courtesy Juan Carlos Niebles and Ranjay Krishna

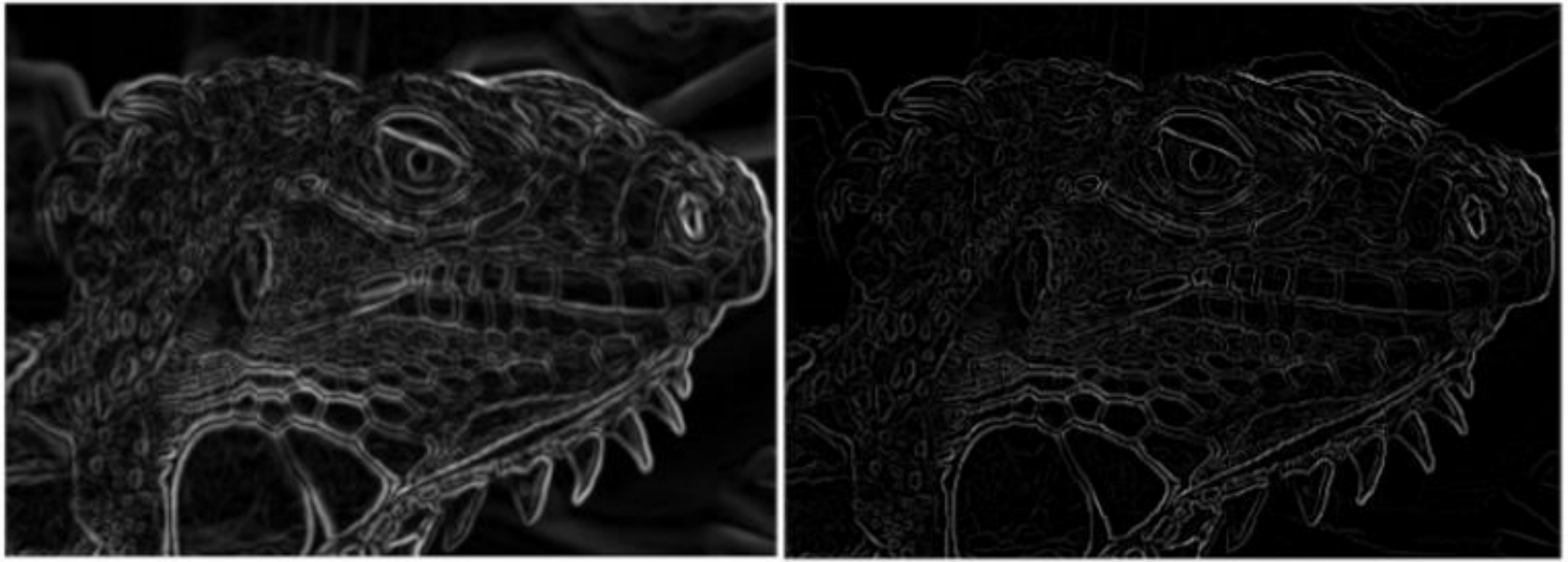


At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.



Canny Edge Detector: Non Maxima Suppression

Courtesy Juan Carlos Niebles and Ranjay Krishna



Before

After

Canny Edge Detector

Courtesy Courtesy Juan Carlos Niebles and Ranjay Krishna

- Suppress Noise
- Compute gradient magnitude and direction
- Apply Non-Maximum Suppression
  – Assures minimal response
- Use hysteresis and connectivity analysis to detect edges

Canny Edge Detector: Double thresholding using Hysteresis thresholding

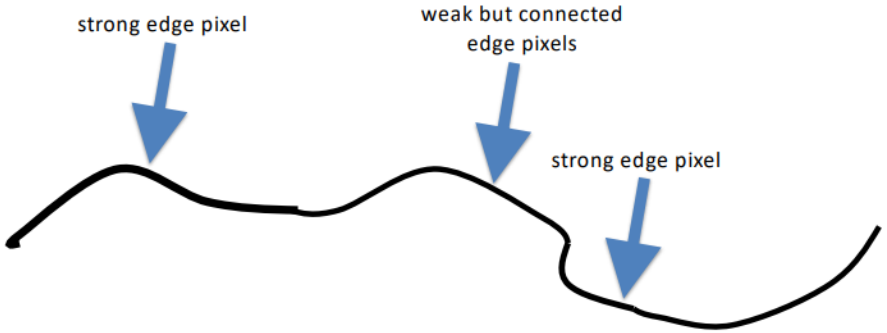
Courtesy : Bernd Girod; Juan Carlos Niebles and Ranjay Krishna

If the **gradient M[x,y]** at a pixel is

- above θ_{High}, declare it as an 'strong edge pixel'
- below θ_{Low}, declare it as a "non-edge-pixel"
- between θ_{Low} and θ_{High} – 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.

Strong edge: $M[x,y] \geq \theta_{high}$

Weak edge: $\theta_{high} > M[x,y] \geq \theta_{low}$



Source: S. Seitz

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:
   - Thin multi-pixel wide “ridges” down to single pixel width
4. Thresholding and linking (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

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

Source: S. Seitz

Canny Edge Detector - Example

Courtesy medium.com

Small σ Large σ

What we learnt:

- Edge detection
- Image gradients
- A simple edge detection
- Sobel edge detector
- Canny edge detector

What we will learn today

- Edge detection via Laplacian of Gaussian
- Deep Learning (DL) based edge detection
- Line Detection via Hough Transform

What we will learn today

- Edge detection via Laplacian of Gaussian
- Deep Learning (DL) based edge detection
- Line Detection via Hough Transform

Edge detection using derivatives

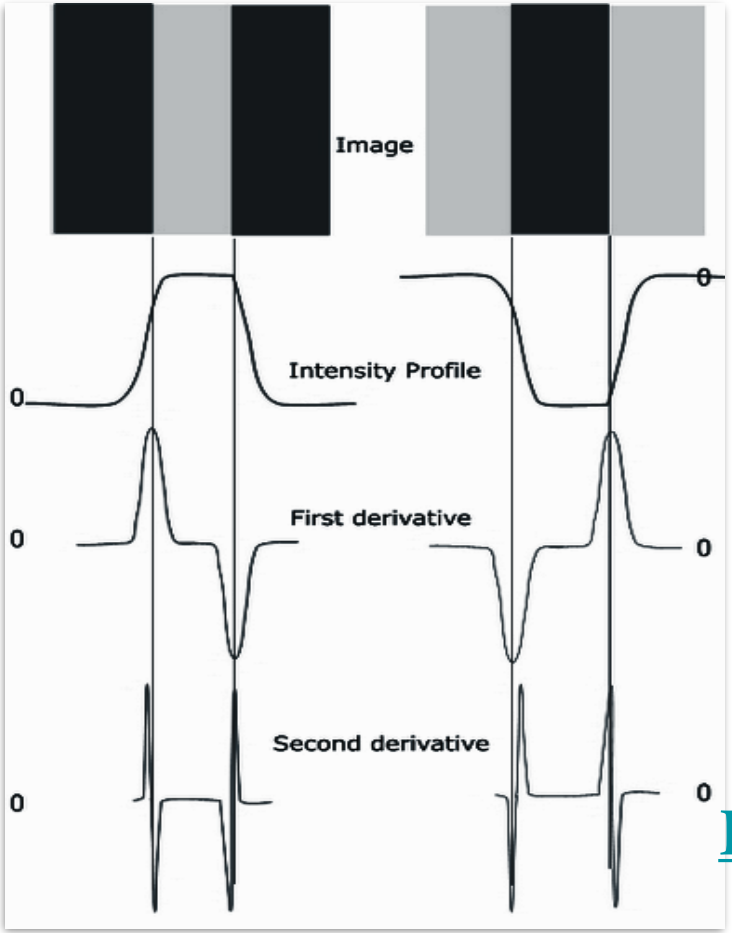
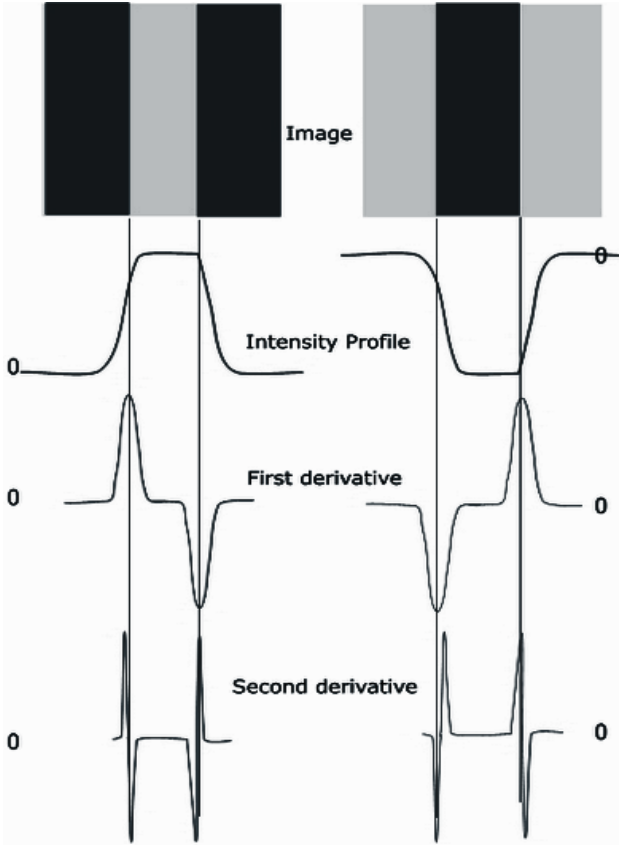


Image Source



Edge detection using derivatives

Courtesy Trucco, Ch. 4 AND Jain et al., Ch. 5

Points which lie on an edge can be detected by:

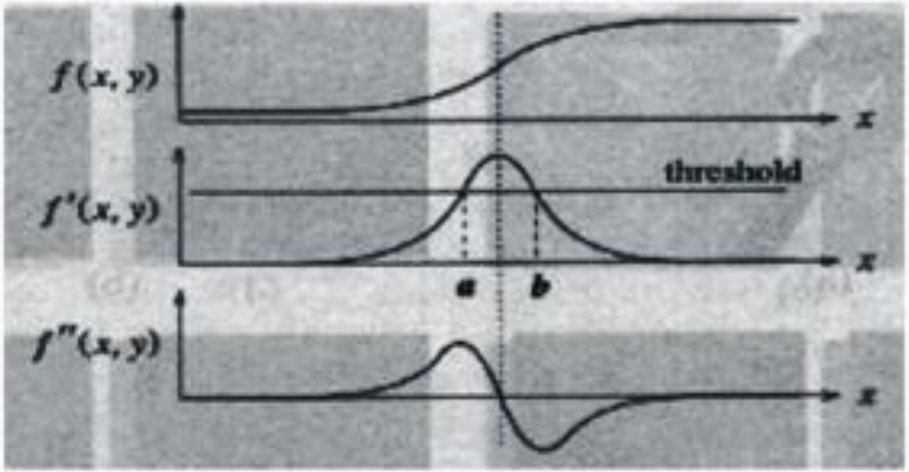
(1) detecting local maxima or minima of the **first derivative**

(2) detecting the zero-crossing of the **second derivative**

Edge detection using 2nd derivative

Courtesy Saad J Bedros

- Edge points can be detected by finding the zero-crossings of the second derivative.



Edge Detection using 2nd derivative

Courtesy Saad J Bedros

- Approximate finding maxima/minima of gradient magnitude by finding places where:

$$\frac{df^2}{dx^2}(x) = 0$$

- Can't always find discrete pixels where the second derivative is zero – look for zero-crossing instead.

Computing 2nd derivative for 1D signals

Courtesy Saad J Bedros

Computing the 2nd derivative:

$$f''(x) = \lim_{h \to 0} \frac{f'(x+h) - f'(x)}{h} \approx f'(x+1) - f'(x) =$$

$$f(x+2) - 2f(x+1) + f(x) \quad (h=1)$$

- This approximation is centered about $x + 1$
- By replacing $x + 1$ by x we obtain:

$$f''(x) \approx f(x+1) - 2f(x) + f(x-1)$$

mask: [1 −2 1]

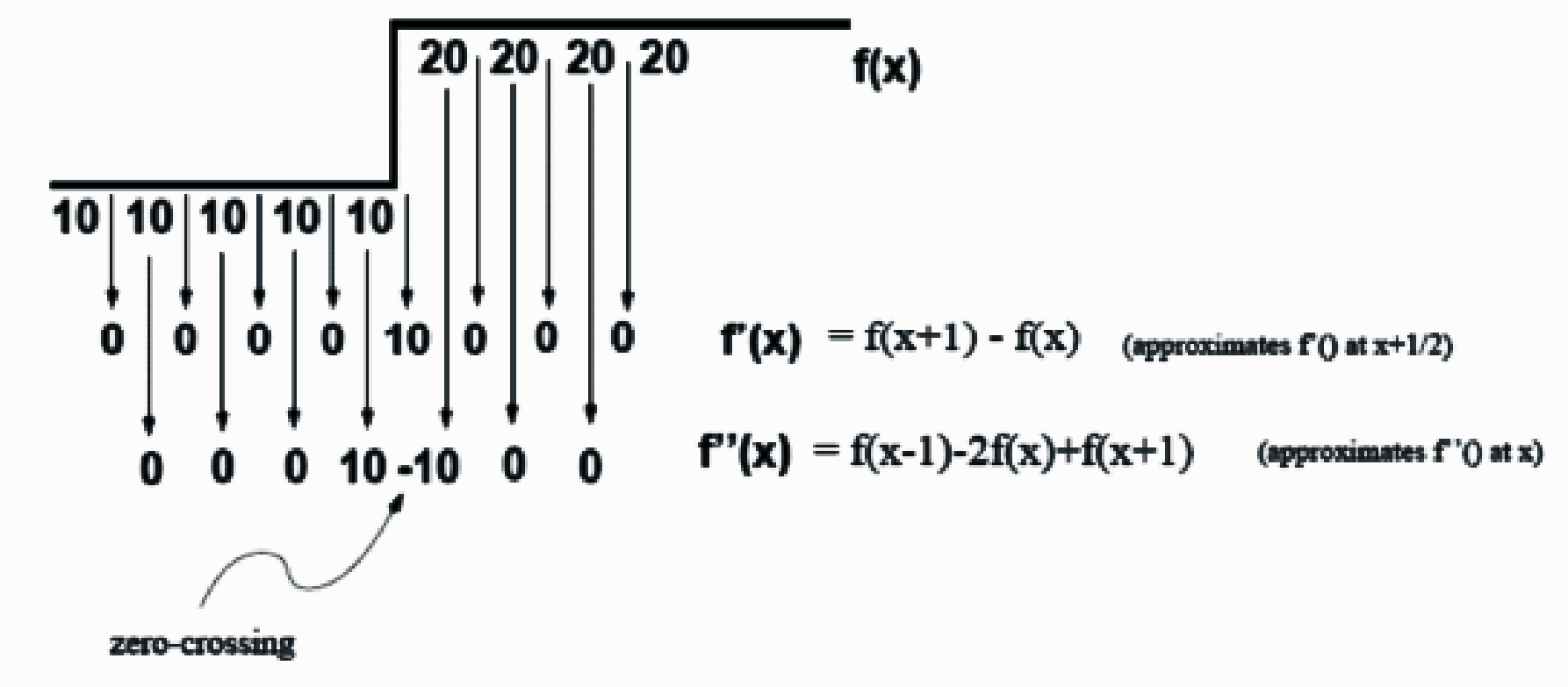
- 1st order

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

- 2nd order

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

Edge detection using zero crossing of 2nd derivative



Computing 2nd derivative for 1D signals - Example

Courtesy Saad J Bedros

mask M = [−1, 2, −1]

S_1			12	12	12	12	12	24	24	24	24	24
S_1	⊗	M	0	0	0	0	-12	12	0	0	0	0

(a) S_1 is an upward step edge

S_2			24	24	24	24	24	12	12	12	12	12
S_2	⊗	M	0	0	0	0	12	-12	0	0	0	0

(b) S_2 is a downward step edge

S_3			12	12	12	12	15	18	21	24	24	24
S_3	⊗	M	0	0	0	-3	0	0	0	3	0	0

(c) S_3 is an upward ramp

S_4			12	12	12	12	24	12	12	12	12	12
S_4	⊗	M	0	0	0	-12	24	-12	0	0	0	0

Laplacian Edge Detector

Courtesy Saad J Bedros

The *Laplacian* is defined mathematically as

$$\nabla^2 = \nabla \cdot \nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

When we apply it to an image, we get

$$\nabla^2 f = \left(\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \right) I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Laplacian Edge Detector

Courtesy Saad J Bedros

$$\frac{\partial^2 f}{\partial x^2} = f(i, j+1) - 2f(i, j) + f(i, j-1)$$

$$\frac{\partial^2 f}{\partial y^2} = f(i+1, j) - 2f(i, j) + f(i-1, j)$$

$$\nabla^2 f = -4f(i, j) + f(i, j+1) + f(i, j-1) + f(i+1, j) + f(i-1, j)$$

0	0	0
1	-2	1
0	0	0

+

0	1	0
0	-2	0
0	1	0

=

0	1	0
1	-4	1
0	1	0

Properties of Laplacian

Courtesy Saad J Bedros

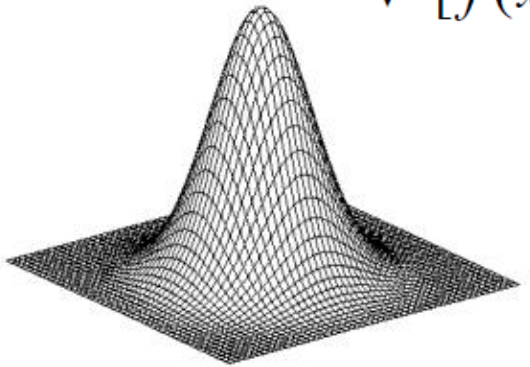
- It is cheaper to implement than the gradient (i.e., one mask only).
- It does not provide information about edge direction.
- It is more sensitive to noise (i.e., differentiates twice).
- It is an isotropic operator (equal weights in all directions)

Zero crossing of Laplacian

- Sensitive to very fine detail and noise ➔ blur image first
- Responds equally to strong and weak edges
  ➔ suppress zero-crossings with low gradient magnitude

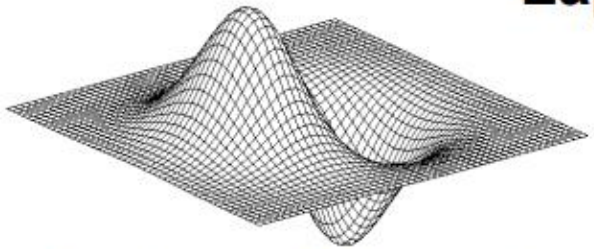
Derivative of Gaussian vs. Laplacian of Gaussian

$$\nabla^2[f(x,y) * G(x,y)] = \nabla^2 G(x,y) * f(x,y)$$



Gaussian

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

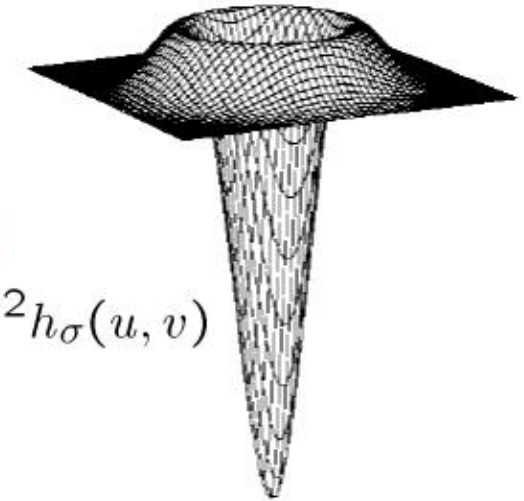


derivative of Gaussian

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

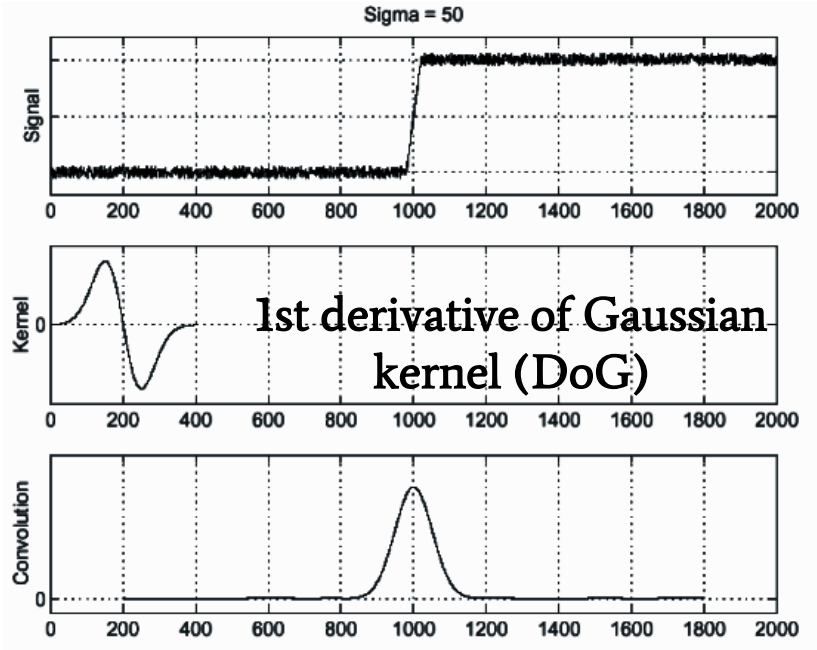
Laplacian of Gaussian

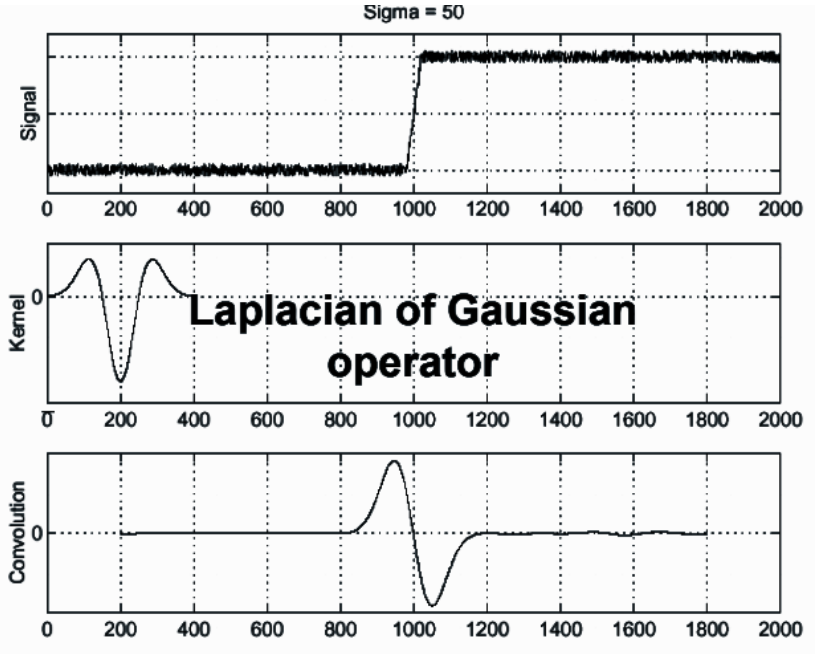
$$\nabla^2 h_\sigma(u,v)$$



Derivative of Gaussian vs. Laplacian of Gaussian

Courtesy Saad J Bedros

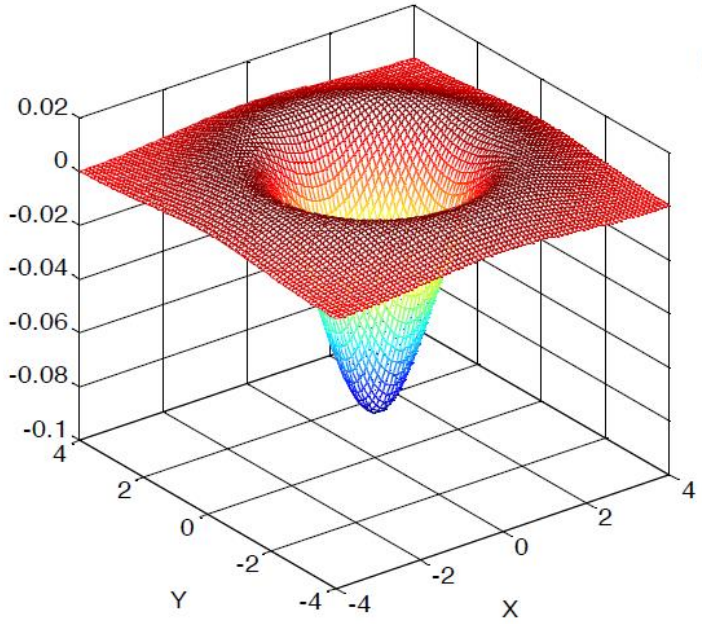




Laplacian of Gaussian

- Filtering of image with Gaussian and Laplacian operators can be combined into convolution with Laplacian of Gaussian (LoG) operator

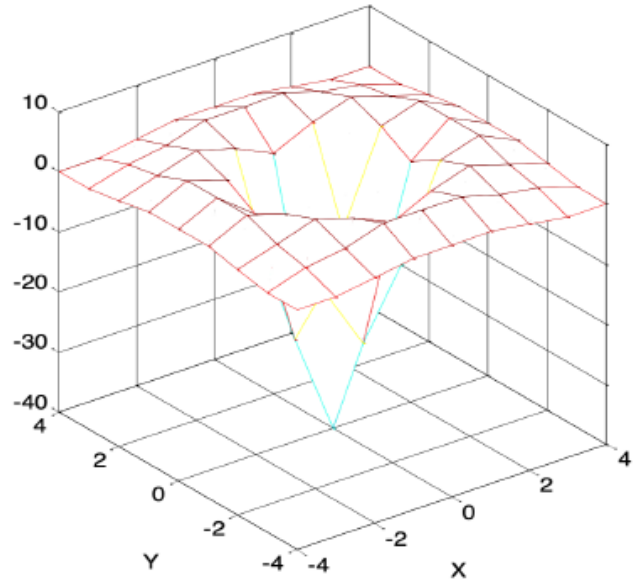
$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left(1-\frac{x^2+y^2}{2\sigma^2}\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$



Discrete approximation of Laplacian of Gaussian

$$\sigma = \sqrt{2}$$

0	0	1	2	2	2	1	0	0
0	2	3	5	5	5	3	2	0
1	3	5	3	0	3	5	3	1
2	5	3	-12	-23	-12	3	5	2
2	5	0	-23	-40	-23	0	5	2
2	5	3	-12	-23	-12	3	5	2
1	3	5	3	0	3	5	3	1
0	2	3	5	5	5	3	2	0
0	0	1	2	2	2	1	0	0

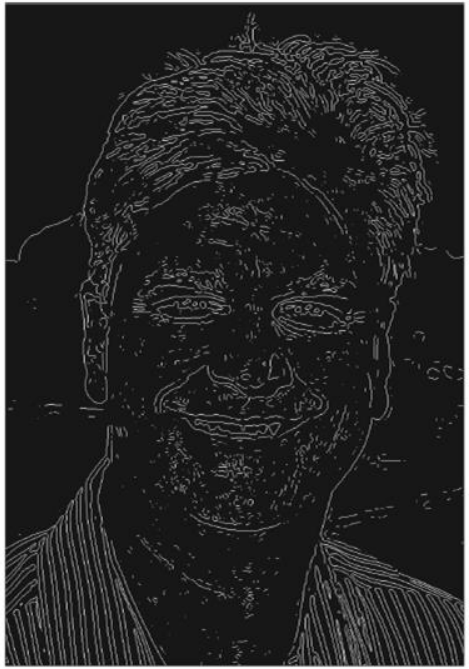


Zero-crossings of LoG

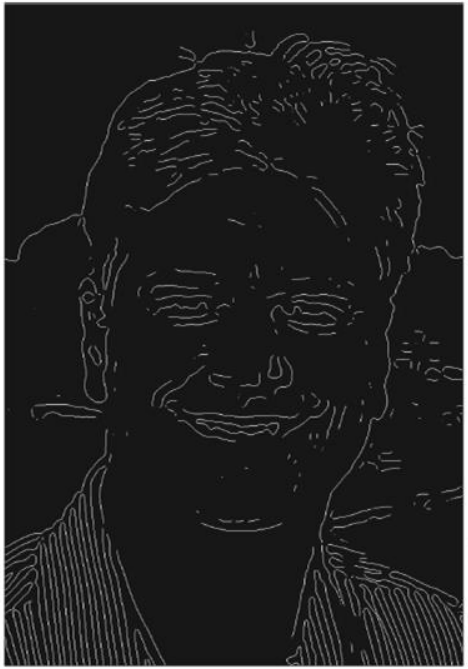
$\sigma = \sqrt{2}$ $\sigma = 2\sqrt{2}$ $\sigma = 4\sqrt{2}$ $\sigma = 8\sqrt{2}$

Zero crossings of LoG - gradient based thresholding

$\sigma = \sqrt{2}$



$\sigma = 2\sqrt{2}$



$\sigma = 4\sqrt{2}$

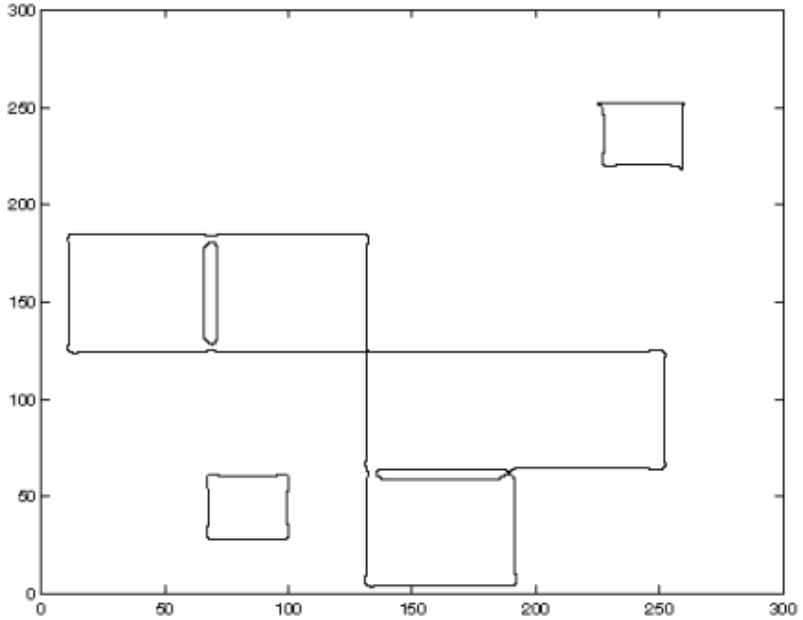
$\sigma = 8\sqrt{2}$

Gradient vs LoG

Courtesy Saad J Bedros

Disadvantage of LOG edge detection:

– Does not handle corners well



Gradient vs LoG

Courtesy Saad J Bedros

- Gradient works well when the image contains sharp intensity transitions and low noise.
- Zero-crossings of LOG offer better localization, especially when the edges are not very sharp.

step edge

2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8

0	0	0	6	−6	0	0	0
0	0	0	6	−6	0	0	0
0	0	0	6	−6	0	0	0
0	0	0	6	−6	0	0	0

ramp edge

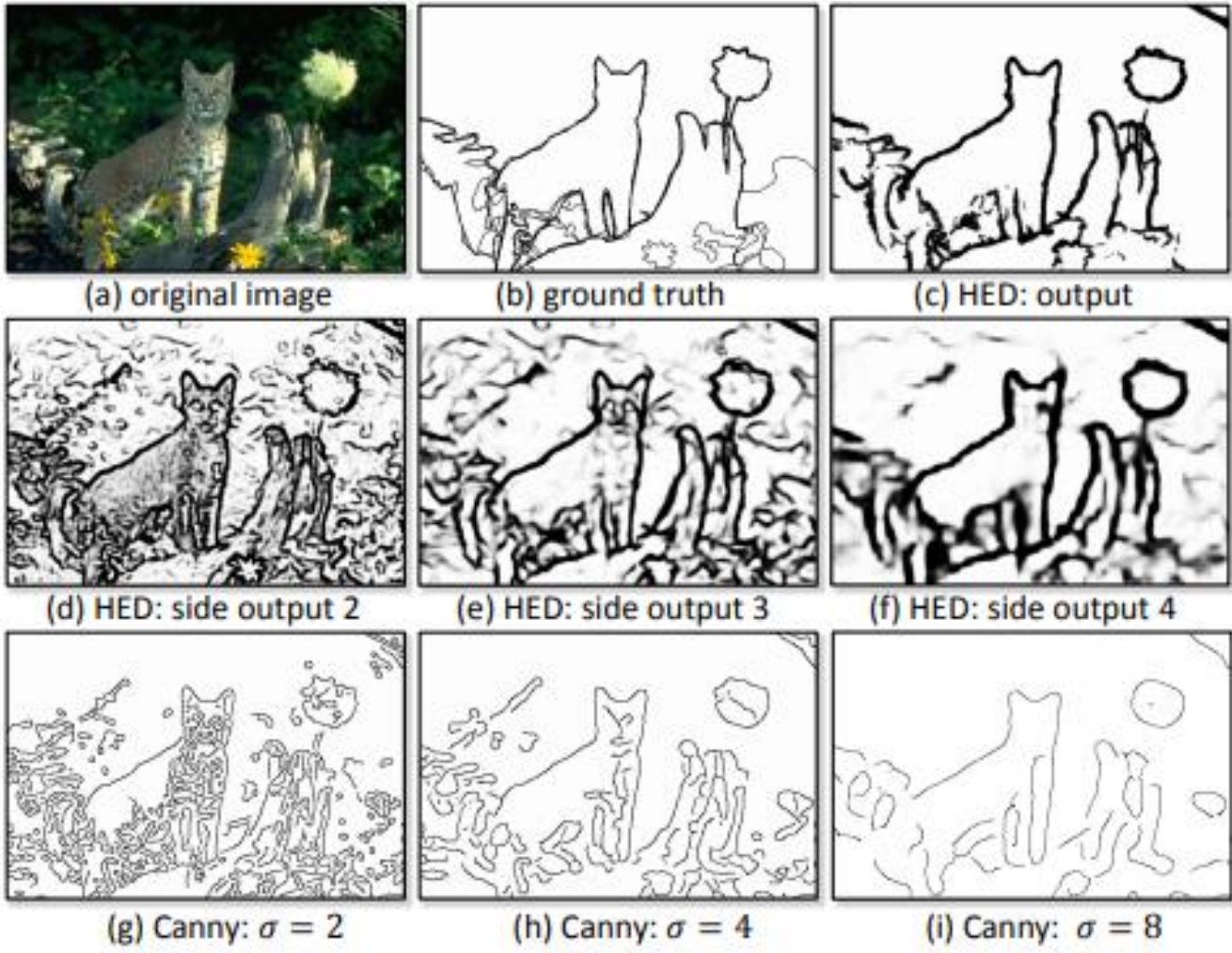
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8

0	0	0	3	0	−3	0	0
0	0	0	3	0	−3	0	0
0	0	0	3	0	−3	0	0
0	0	0	3	0	−3	0	0

What we will learn today

- Edge detection via Laplacian of Gaussian
- DL based edge detection
- Line Detection by Hough Transform

Holistically-Nested Edge Detection (HED), ICCV 2015



Holistically-Nested Edge Detection (HED)

- Holistic image training and prediction, inspired by fully convolutional neural networks, for *image-to-image classification* (the system takes an image as input, and directly produces the edge map image as output).

- Nested multi-scale feature learning, that performs *deep layer supervision* to “guide” early classification results.

- An improved speed (0.4s per image) that is orders of magnitude faster than some recent CNN-based edge detection algorithms.

Model Architecture

- Based on VGGNet-16
- The last stage of VGGNet, including the 5th pooling layer and all the fully connected layers are removed.
- Side output layers are inserted after last convolutional layer in each stage, respectively conv1 2, conv2 2, conv3 3, conv4 3, conv5 3.
- One weighted-fusion layer is added to automatically learn how to combine outputs from multiple scales.

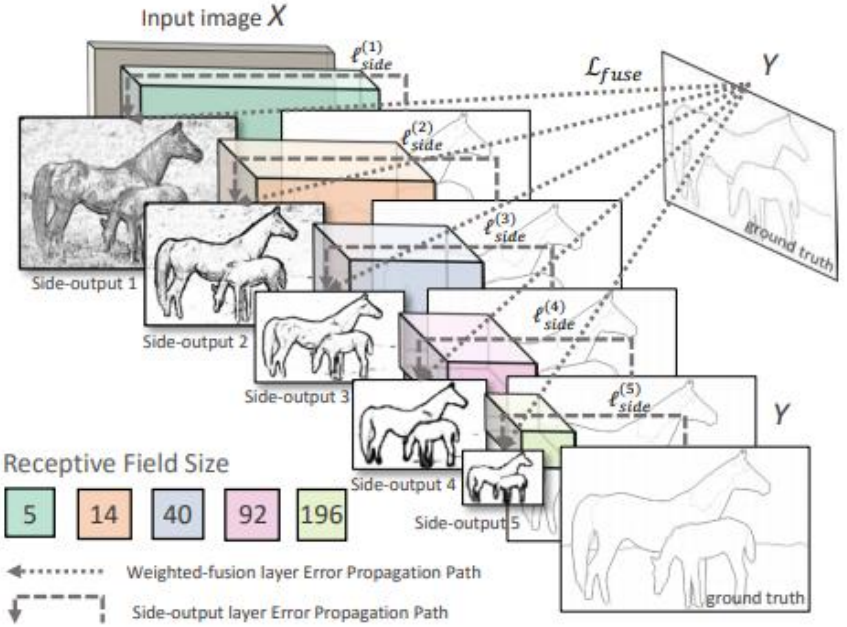
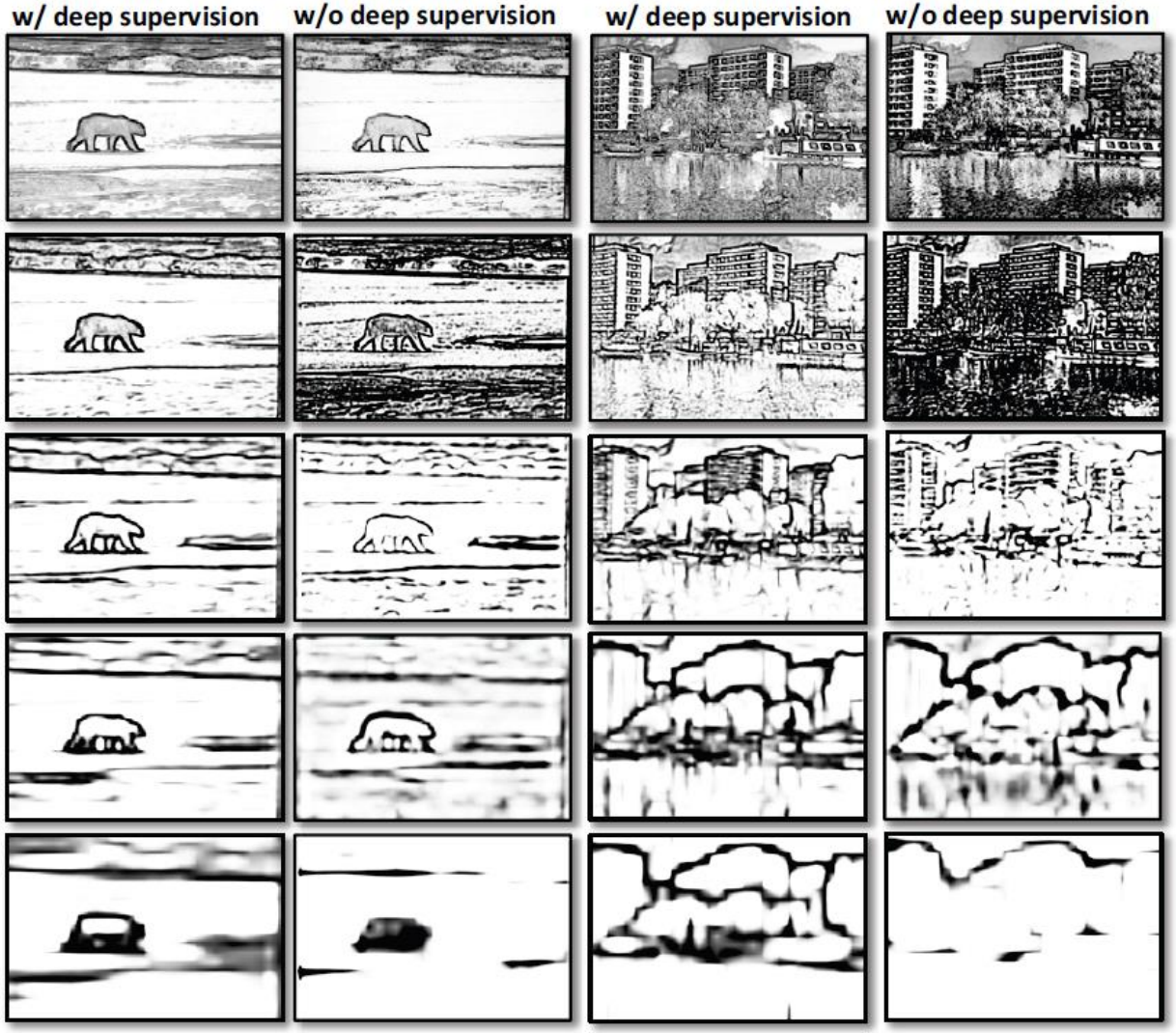


Figure 3. Illustration of our network architecture for edge detection, highlighting the error backpropagation paths. Side-output layers are inserted after convolutional layers. Deep supervision is imposed at each side-output layer, guiding the side-outputs towards edge predictions with the characteristics we desire. The outputs of HED are multi-scale and multi-level, with the side-output-plane size becoming smaller and the receptive field size becoming larger. One weighted-fusion layer is added to automatically learn how to combine outputs from multiple scales. The entire network is trained with multiple error propagation paths (dashed lines).

Loss function

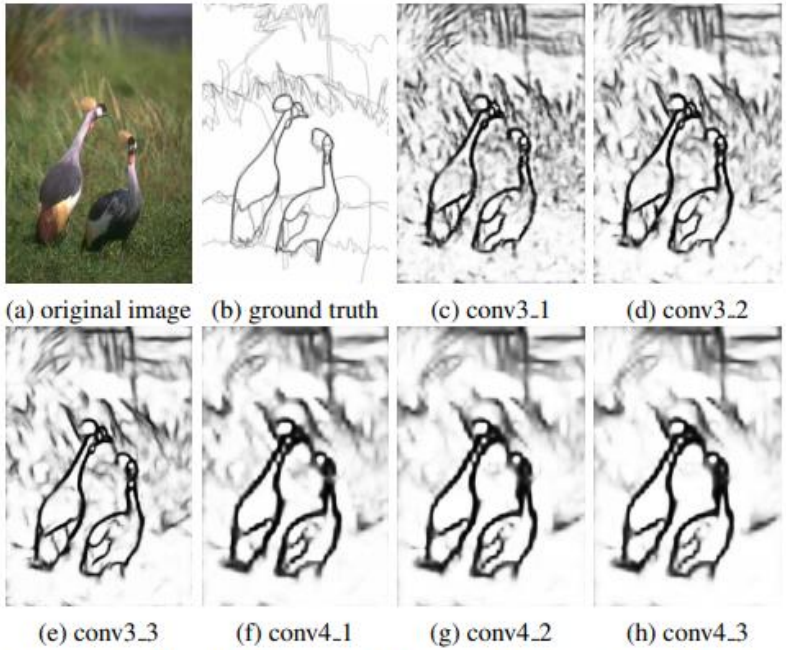
- The loss is computed not only over the final output of the network but all on all the M side outputs.

- The final HED output is weighted average of all the M side outputs and the final output of the network.



Deep supervision helps sideoutput layers to produce multi-scale dense predictions. Note that in the left column, the side outputs become *progressively coarser* and *more "global"*, while critical *object boundaries are preserved*. In the right column, the predictions tends to *lack any discernible order* (e.g. in layers 1 and 2), and *many boundaries are lost in later stages.*

Richer Convolutional Features for Edge Detection, CVPR 17 / TPAMI 19



(a) original image (b) ground truth (c) conv3_1 (d) conv3_2

(e) conv3_3 (f) conv4_1 (g) conv4_2 (h) conv4_3

Figure 1: We build a simple network based on VGG16 [50] to produce side outputs of *conv3_1*, *conv3_2*, *conv3_3*, *conv4_1*, *conv4_2* and *conv4_3*. One can clearly see that convolutional features become coarser gradually, and the intermediate layers *conv3_1*, *conv3_2*, *conv4_1*, and *conv4_2* contain lots of useful fine details that do not appear in other layers.

Richer Convolutional Features (RCF) for Edge Detection

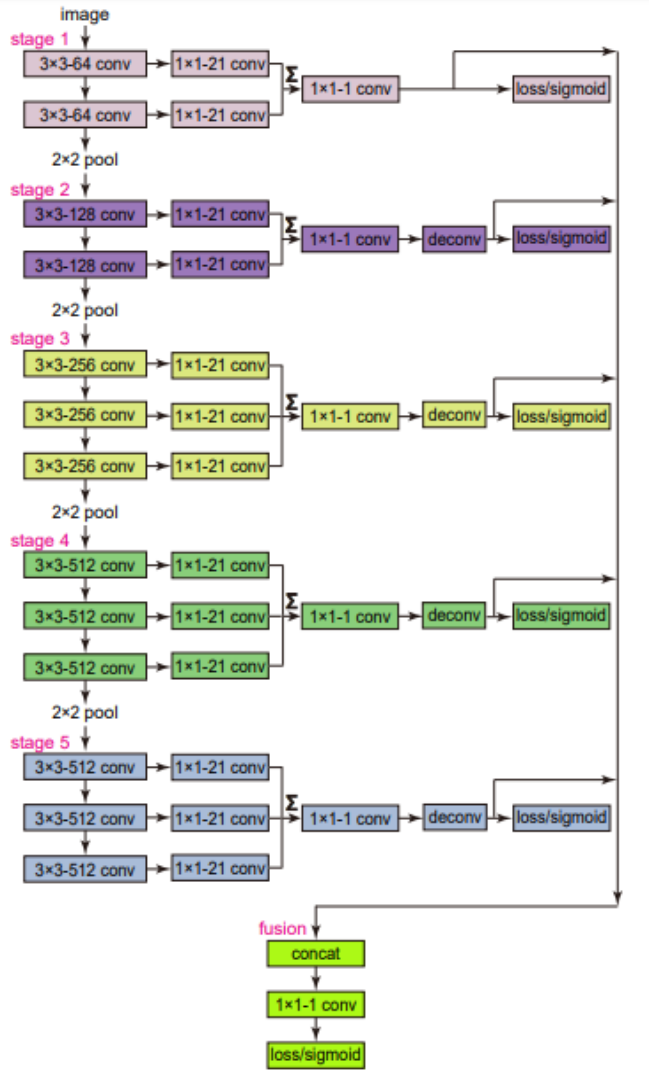
- Since objects in natural images possess *various scales and aspect ratios*, learning the rich hierarchical representations is very critical for edge detection.

- CNNs have been proved to be effective for this task. The convolutional features in CNNs gradually become coarser with the increase of the receptive fields.

- The proposed network fully exploits multiscale and multilevel information of objects to perform the image-to-image prediction by combining all the meaningful convolutional features in a holistic manner.

RCF Model Architecture

- Based on VGGNet-16
- All FC layers and the pool5 layer are removed. .
- Each conv layer in VGG16 is connected to a conv layer with kernel size 1 × 1 and channel depth 21. And the resulting layers in each stage are accumulated using an eltwise layer to attain hybrid features.
- An 1 × 1 − 1 conv layer follows each eltwise layer. Then, a deconv layer is used to up-sample this feature map.
- A cross-entropy loss / sigmoid layer is connected to the up-sampling layer in each stage.
- All the up-sampling layers are concatenated. Then an 1×1 conv layer is used to fuse feature maps from each stage. At last, a cross-entropy loss / sigmoid layer is followed to get the fusion loss / output.



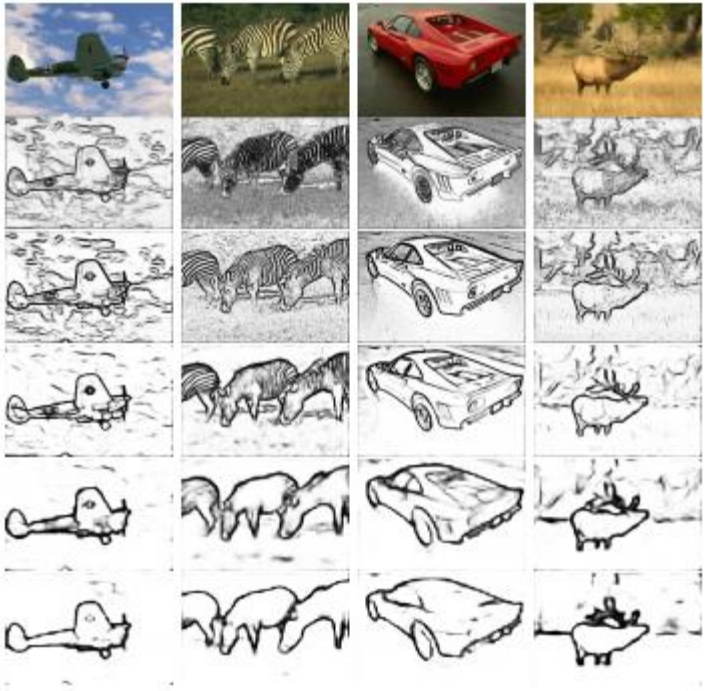


Figure 3: Several examples of the outputs in each stage of RCF. The top line is the original images from BSDS500 [2]. From second to sixth line is the output of stage 1, 2, 3, 4 and 5 respectively.

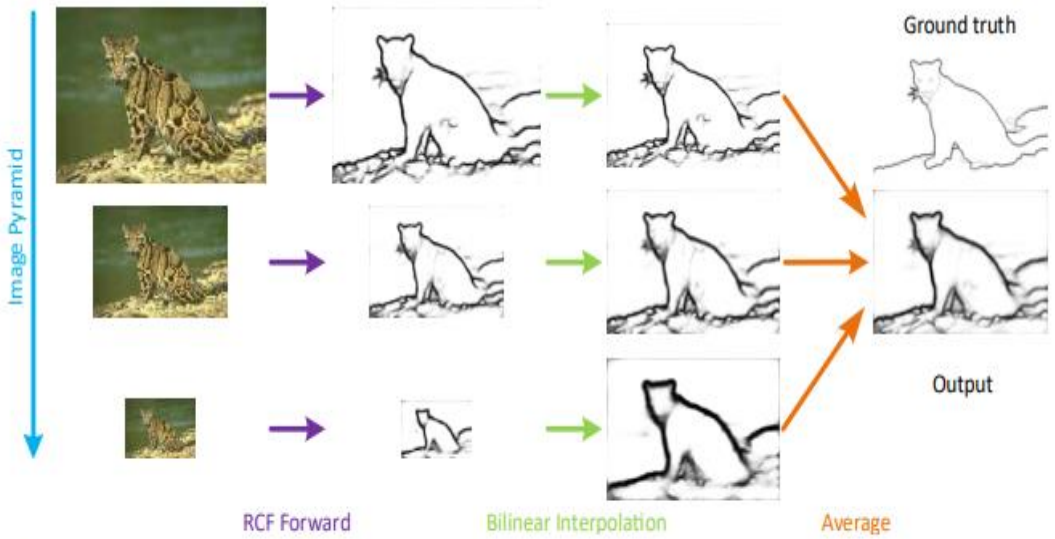


Figure 4: The pipeline of our multiscale algorithm. The original image is resized to construct an image pyramid. And these multiscale images are input to RCF network for a forward pass. Then, we use bilinear interpolation to restore resulting edge response maps to original sizes. A simple average of these edge maps will output high-quality edges.

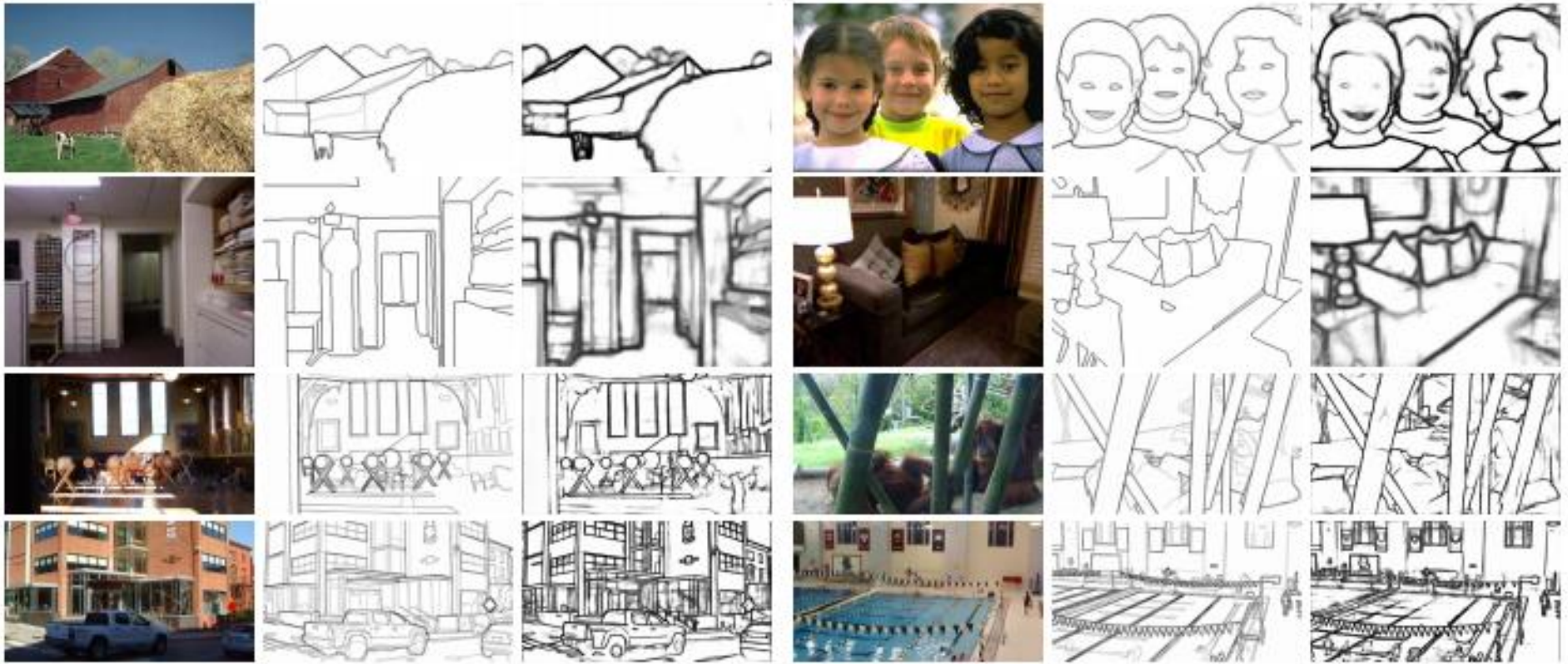


Figure 7: Some examples of RCF. From top to bottom: BSDS500 [2], NYUD [49], Multicue-Boundary [41], and Multicue-Edge [41]. From left to right: origin image, ground truth, RCF edge map, origin image, ground truth, and RCF edge map.

What we will learn today

- Edge detection via Laplacian of Gaussian
- DL based edge detection
- Line Detection by Hough Transform

Intro to Hough transform

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

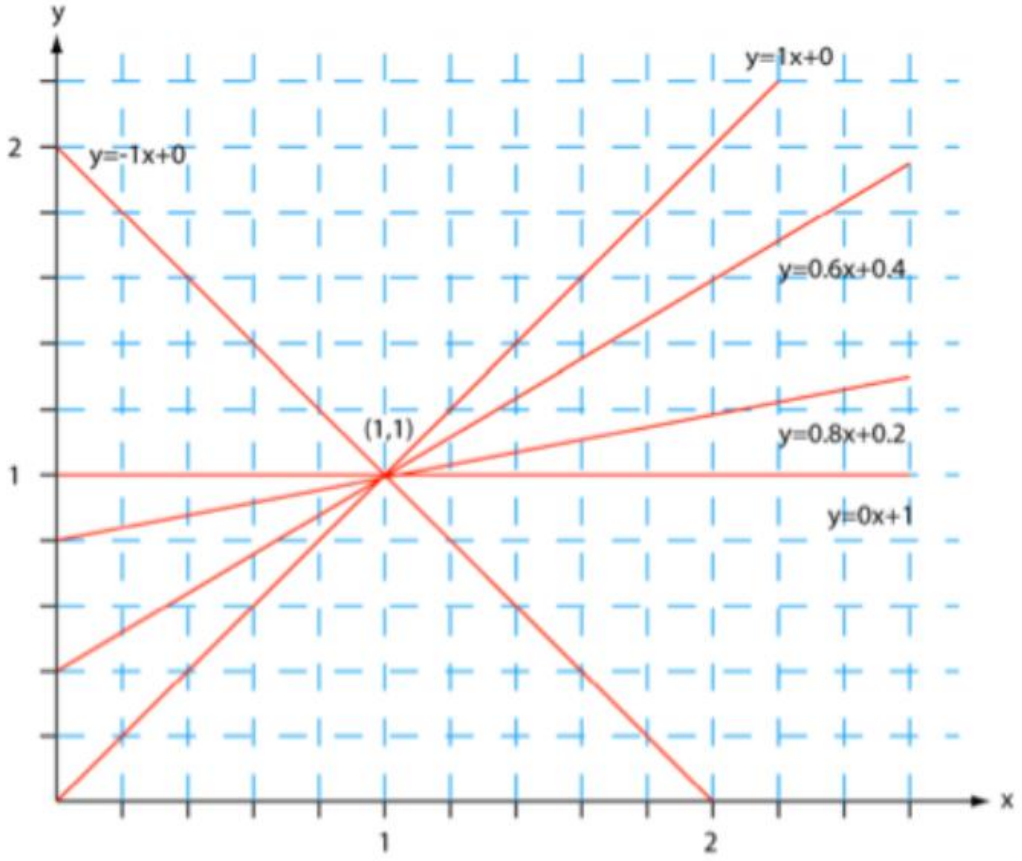
Prior to Hough transform

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

Detecting lines using Hough transform

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

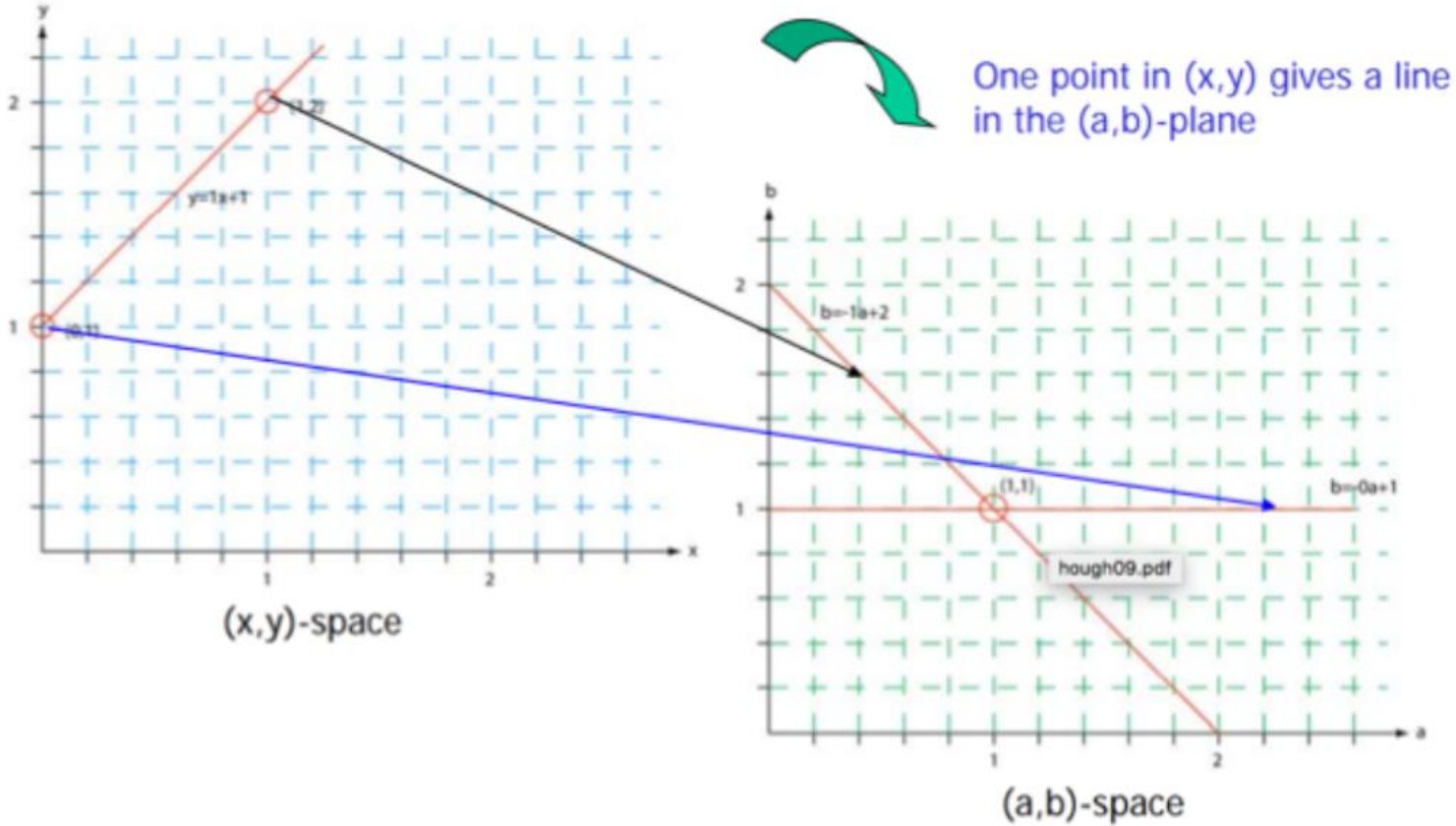
Detecting lines using Hough transform



Detecting lines using Hough transform

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

Detecting lines using Hough transform



Detecting lines using Hough transform

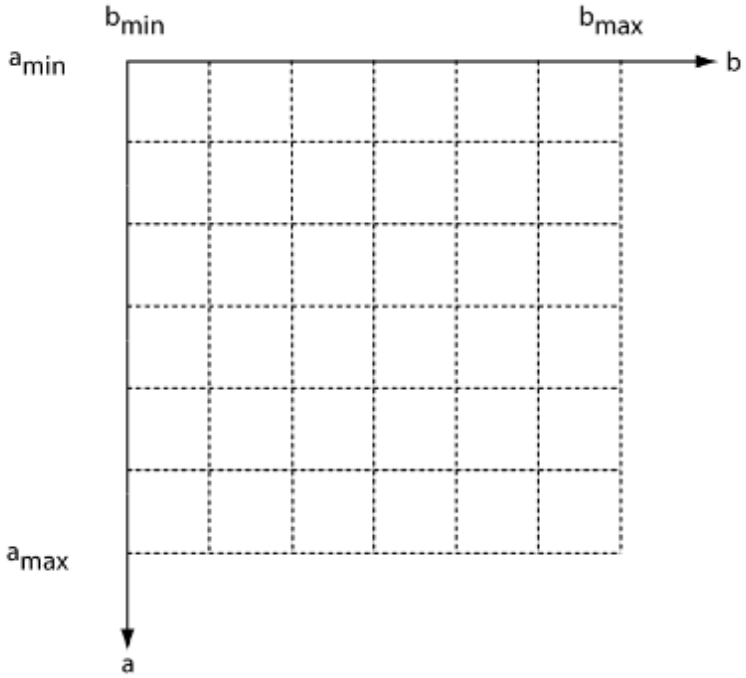
- Two points (x_1, y_1) and(x_2 y_2) define a line in the (x, y) plane.
- These two points give rise to two different lines in (a,b) space.
- In (a,b) space these lines will intersect in a point (a' b')
- All points on the line defined by (x_1, y_1) and (x_2 , y_2) in (x, y) space will parameterize lines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

Algorithm for Hough transform

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

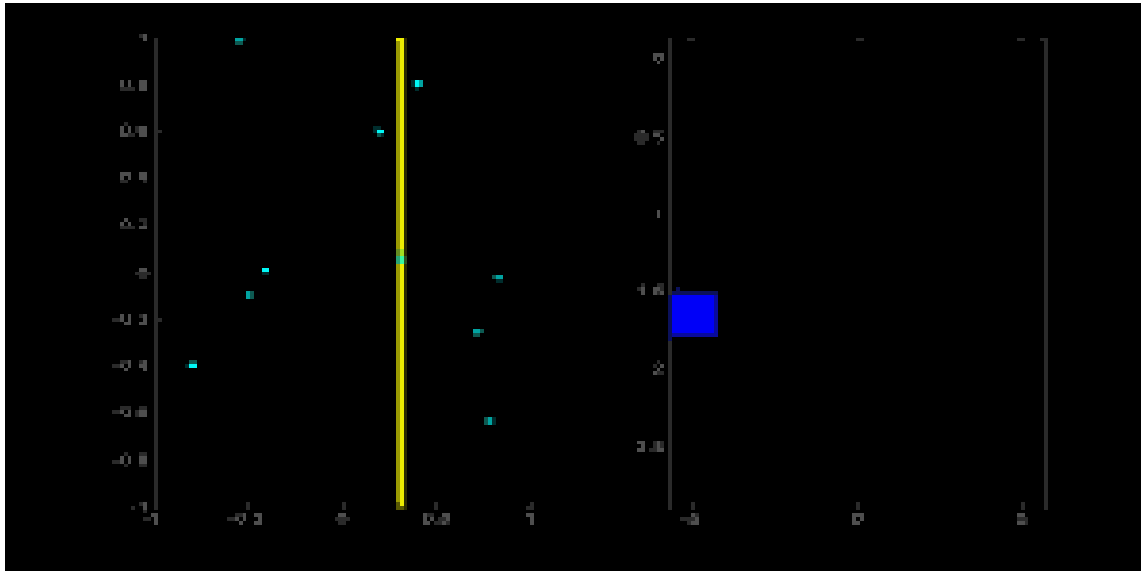
Detect Straight lines using Hough Transform

Courtesy: Anne Solberg



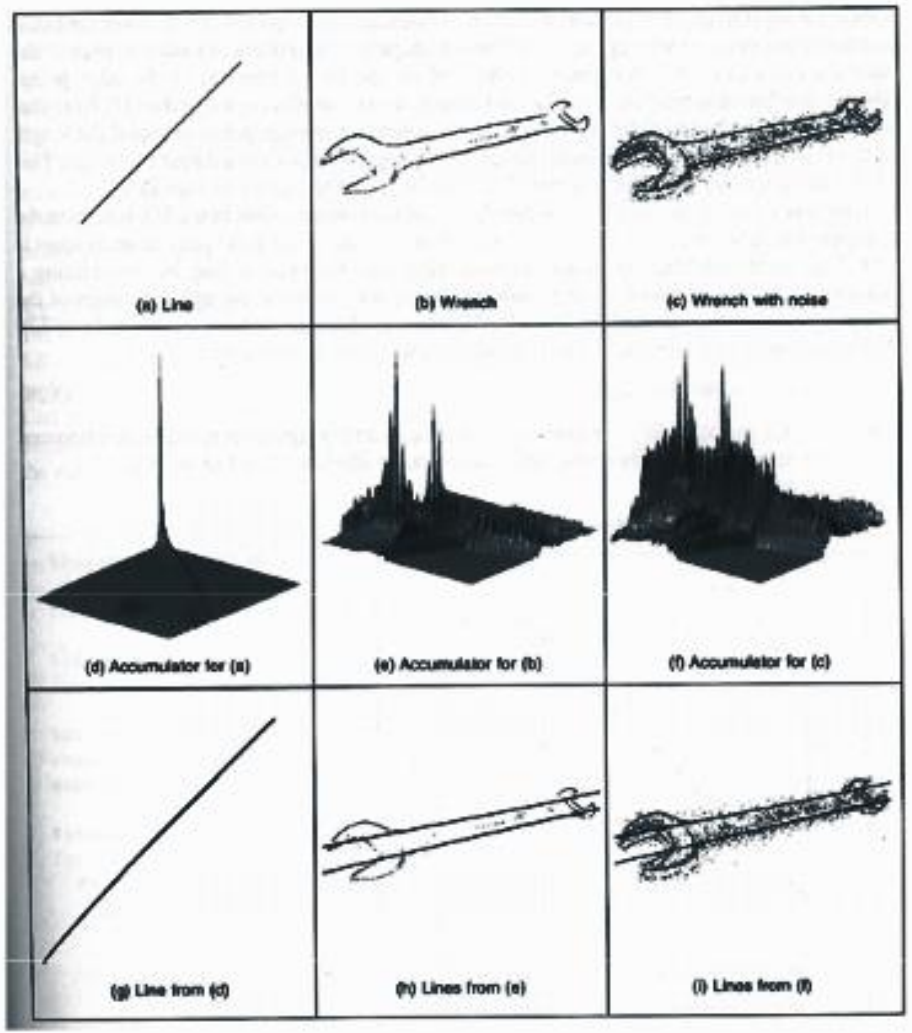
Hough accumulator cells

The accumulator then with the highest count are the true (a,b) for y-ax+b



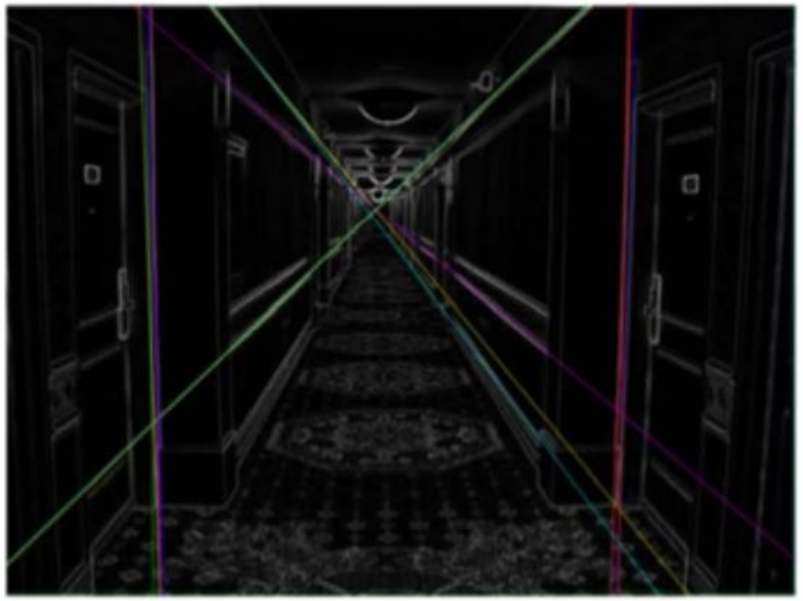
Detect Straight lines

Courtesy: Anne Solberg



Output of Hough transform

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



Concluding remarks

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

What we learnt today

- Edge detection via Laplacian of Gaussian
- DL based edge detection
- Line Detection by Hough Transform