CAP 4453
Robot Vision

Dr. Gonzalo Vaca-Castaño
gonzalo.vacacastano@ucf.edu

Administrative details

- REU low inscription rate
- Homework 1 graded
- Homework 2 questions?
- Any Doubts from last classes?

Robot Vision

5. Edge detection

Credits

- Some slides comes directly from:
  - Yogesh S Rawat (UCF)
  - Noah Snavely (Cornell)
  - Ioannis (Yannis) Gkioulekas (CMU)
  - Mubarak Shah (UCF)
  - S. Seitz
  - James Tompkin
  - Ulas Bagci
  - L. Lazebnik
  - D. Hoeim

Outline

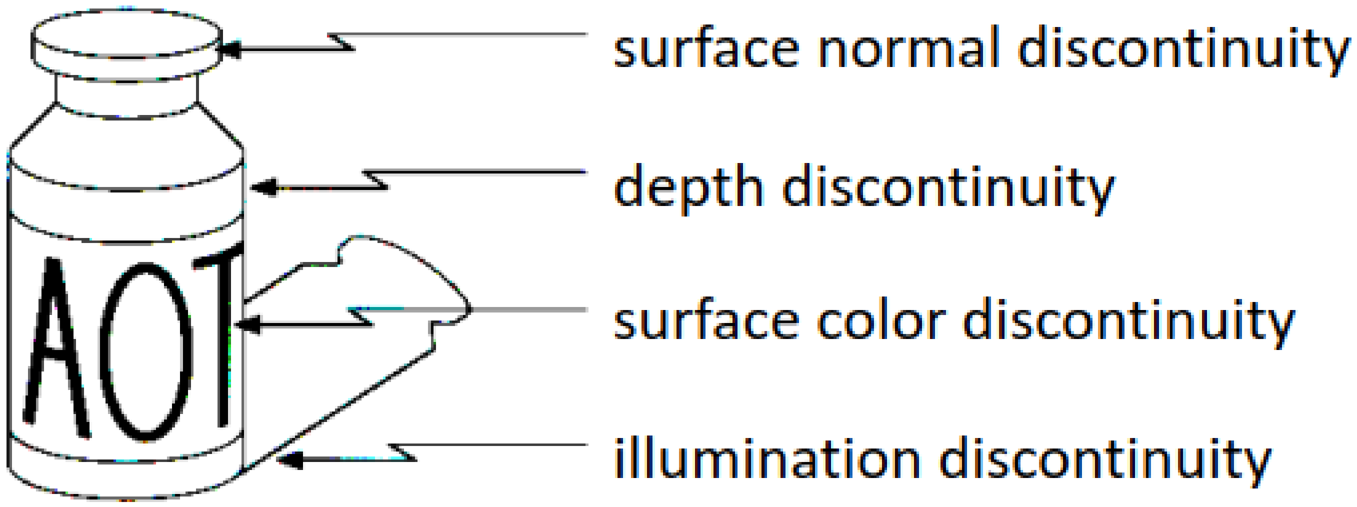
- Image as a function
- Extracting useful information from Images
  - ~~Histogram~~
  - ~~Filtering (linear)~~
  - ~~Smoothing/Removing noise~~
  - ~~Convolution/Correlation~~
  - ~~Image Derivatives/Gradient~~
  - Edges

Edge Detection

- Identify sudden changes in an image
  - Semantic and shape information
  - Mark the border of an object
  - More compact than pixels

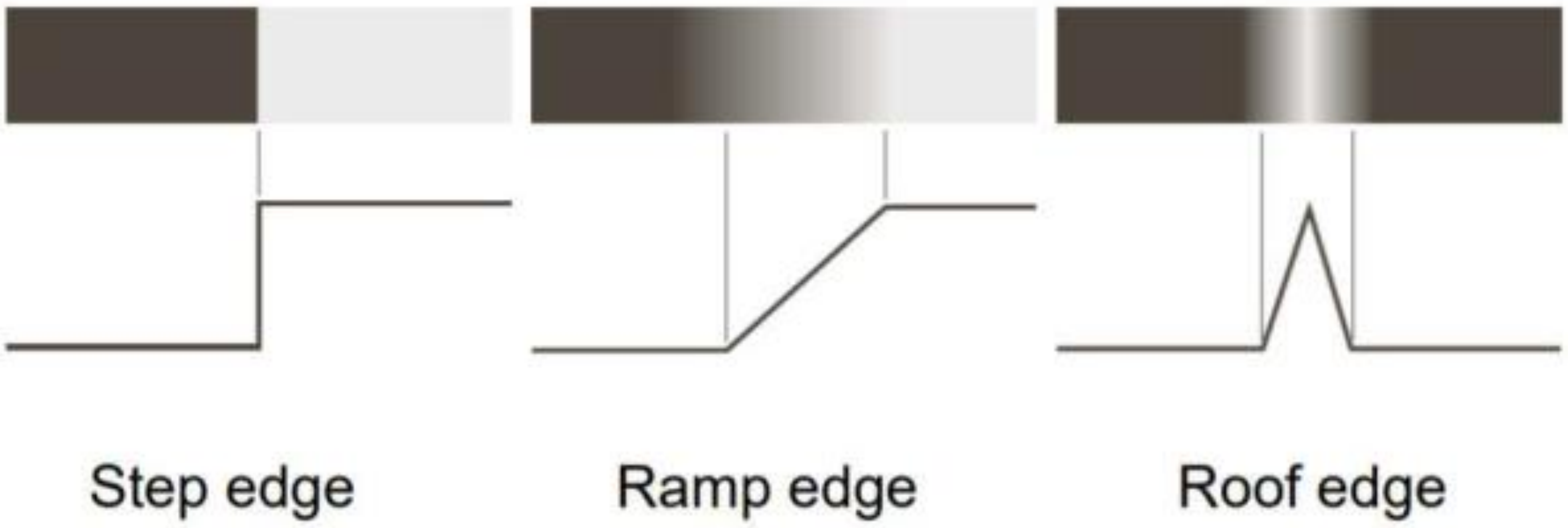
Origin of edges

- Edges are caused by a variety of factors



Type of edges

- Edge models



Why edge detection ?

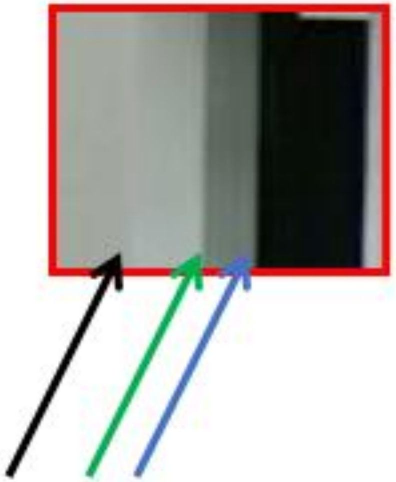
- Extract useful information from images
  - Recognizing objects
- Recover geometry



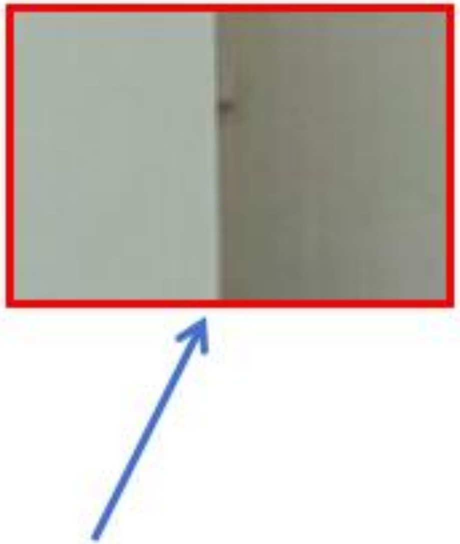


Close up of edges

Close up of edges

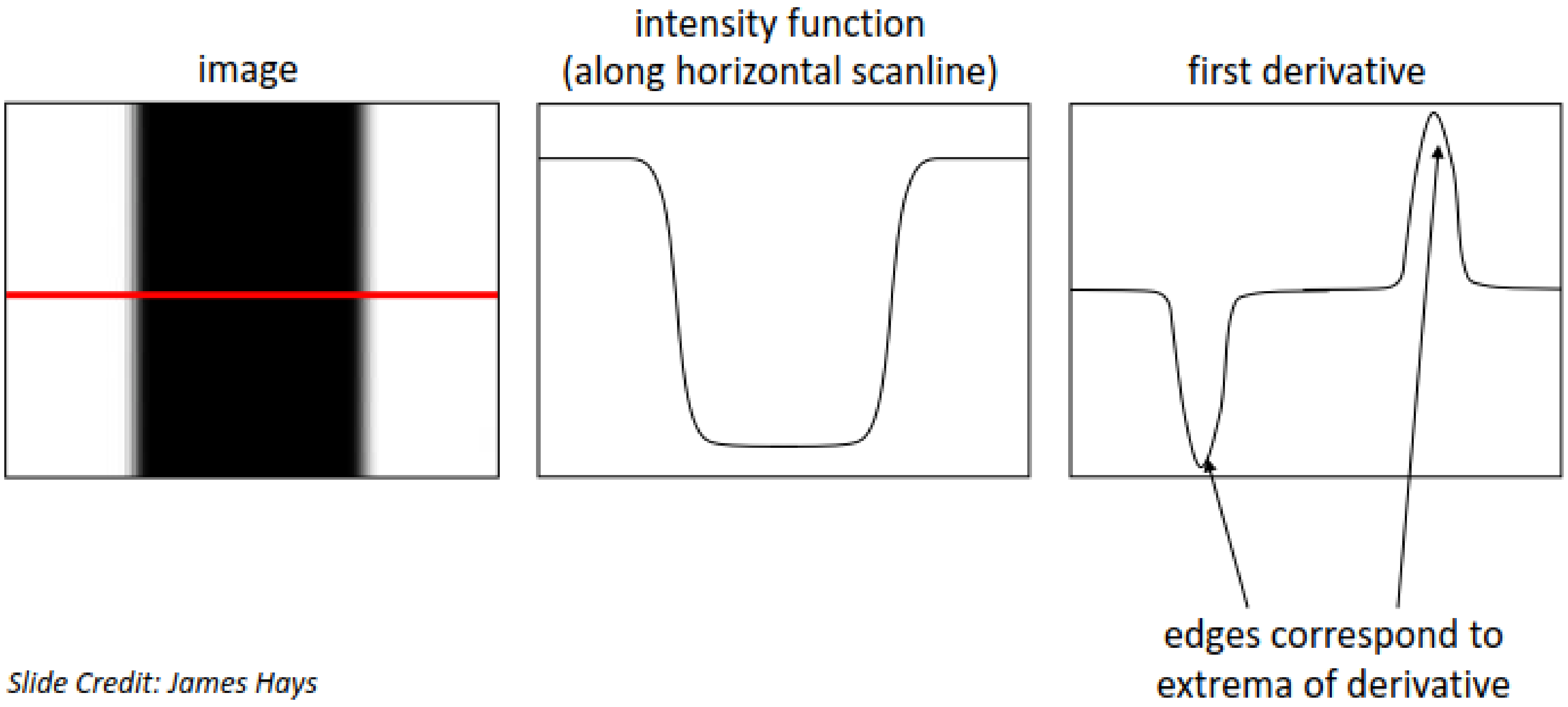


Close up of edges



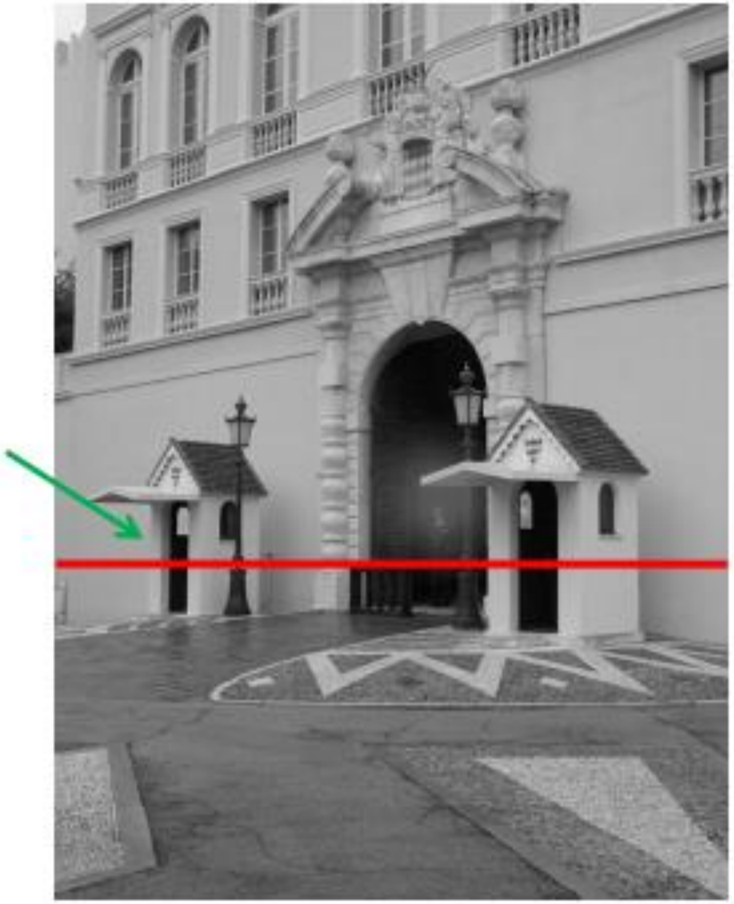
Close up of edges

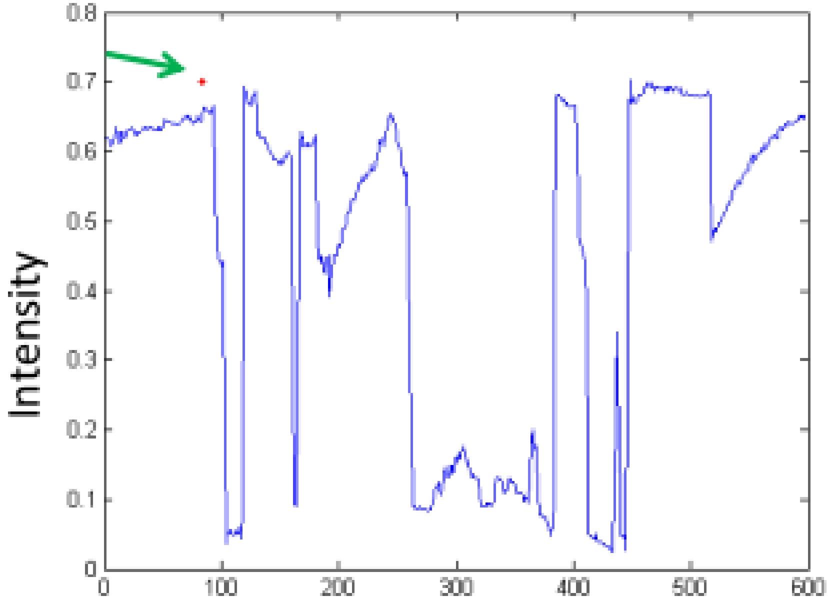
Characterizing edges

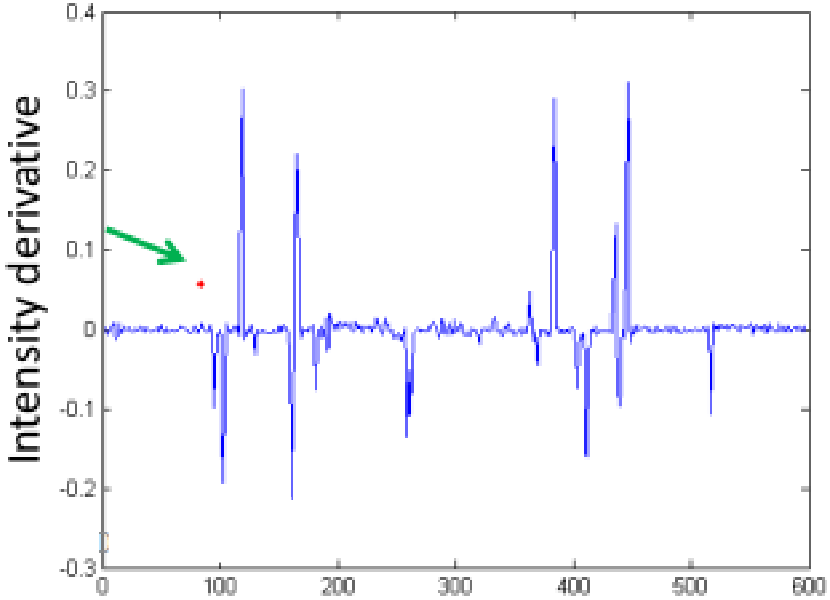


Slide Credit: James Hays

Intensity profile



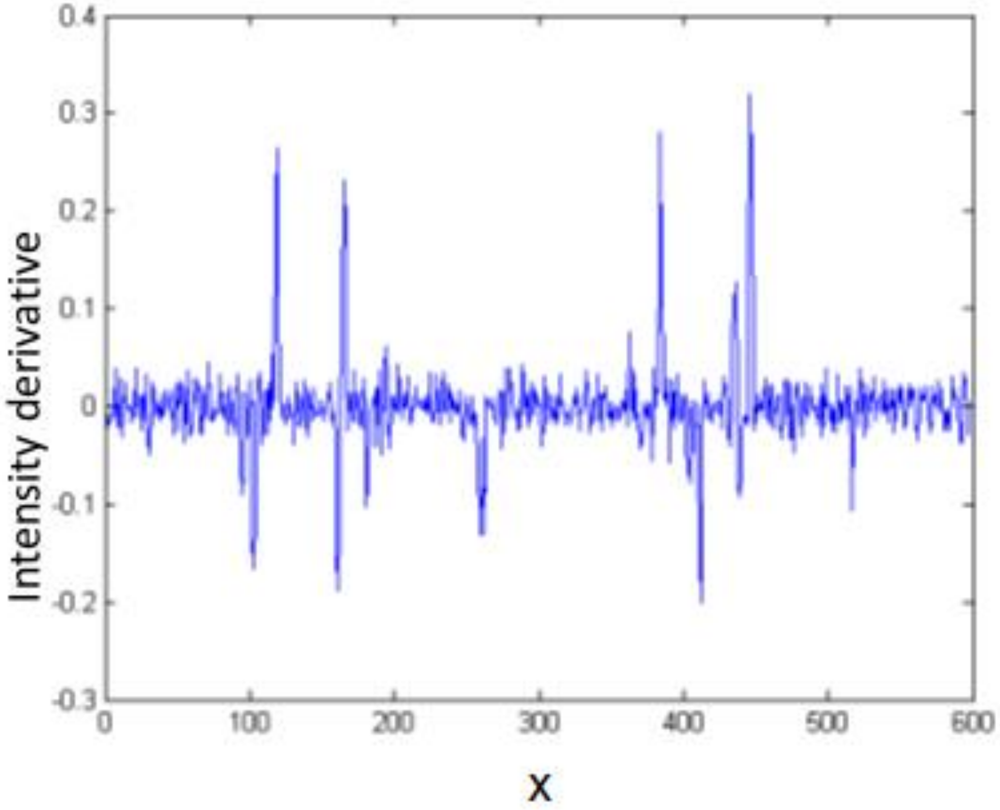




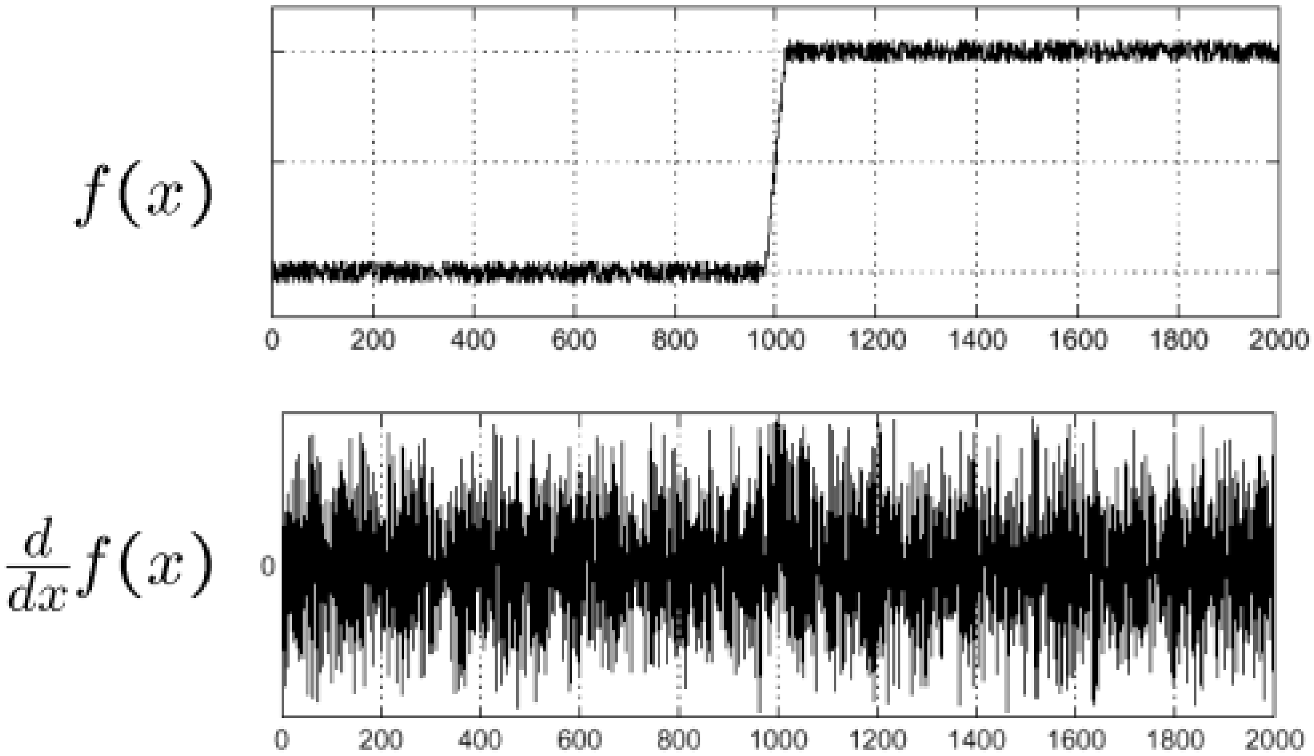
1D derivative filter

1	0	-1

With a little bit of gaussian noise

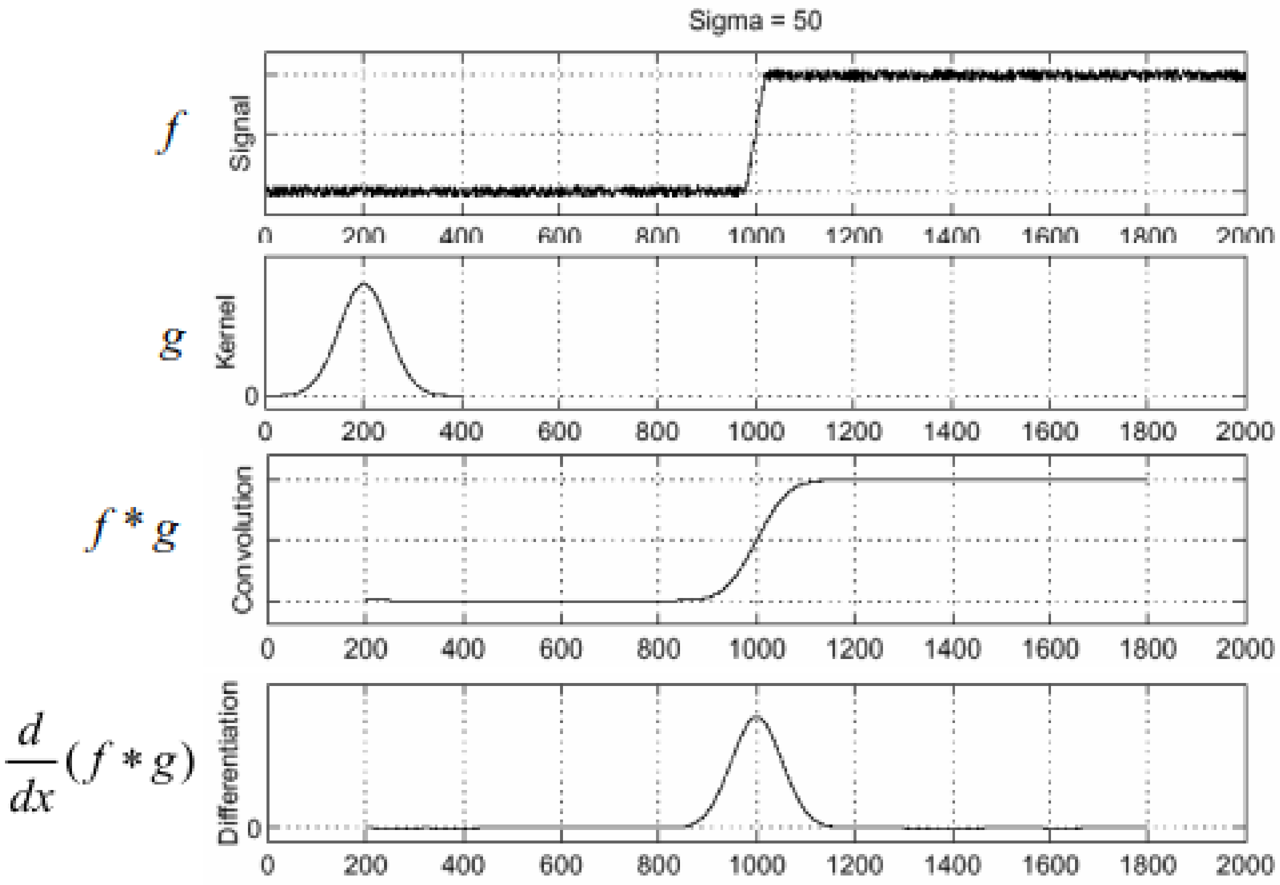


An extreme case



Where is the edge?

Solution: smooth and derivate



To find edges, look for peaks in $\frac{d}{dx}(f * g)$

The Sobel filter

$$\underbrace{\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}}_{\text{Sobel filter}} = \underbrace{\begin{bmatrix}1\\ 2\\ 1\end{bmatrix}}_{\text{Smoothing}} * \underbrace{\begin{bmatrix}1 & 0 & -1\end{bmatrix}}_{\text{1D derivative filter}}$$

$$\underbrace{\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix}}_{\text{Sobel filter}} = \underbrace{\begin{bmatrix}1 & 0 & -1\end{bmatrix}}_{\text{1D derivative filter}} * \underbrace{\begin{bmatrix}1\\ 2\\ 1\end{bmatrix}}_{\text{Smoothing}}$$

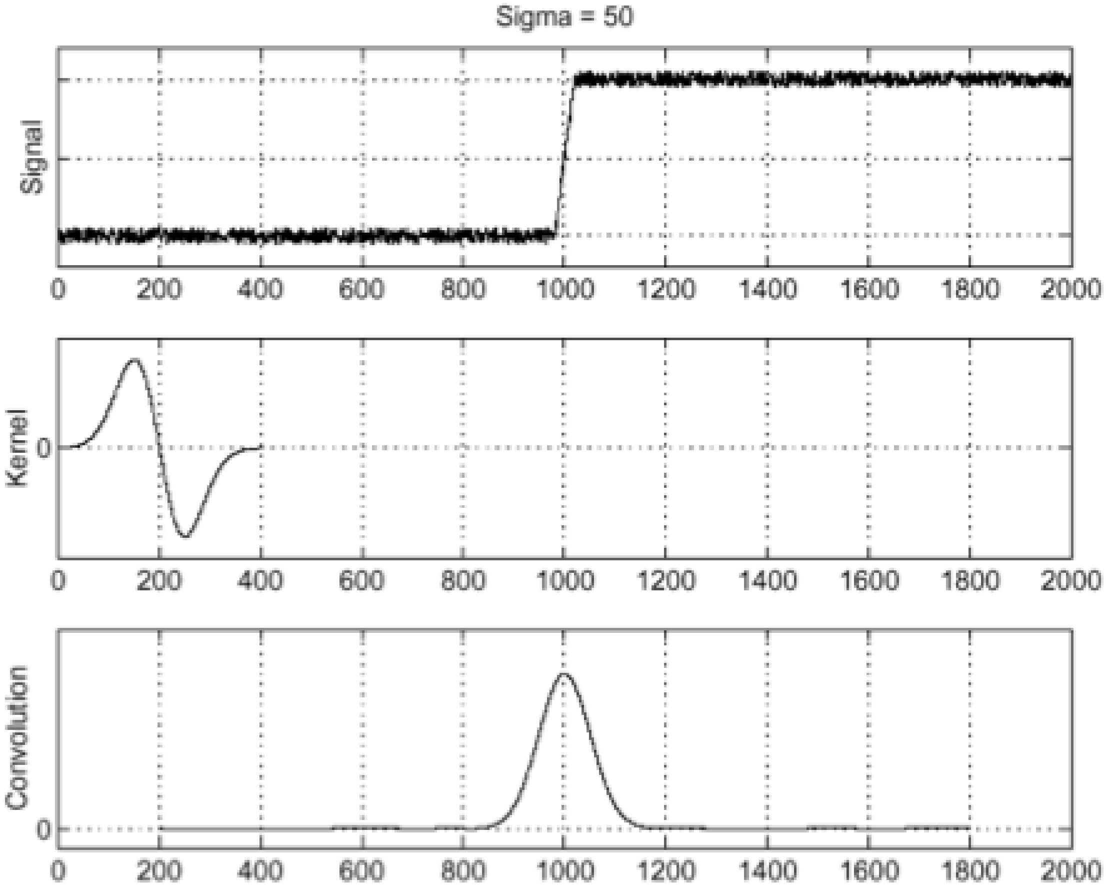
Derivative of Gaussian (DoG) filter

Derivative theorem of convolution: $\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$

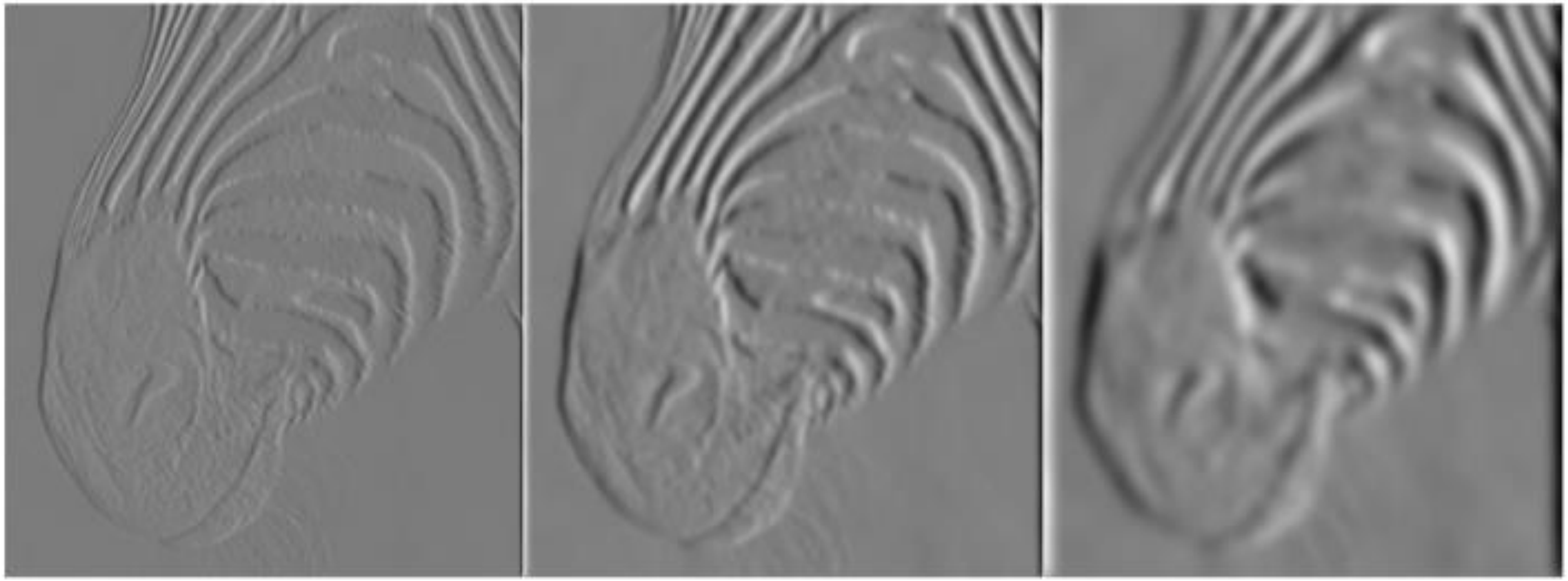
input

derivative of Gaussian

output (same as before)



Solution: smoothing



1 pixel 3 pixels 7 pixels

Smoothing remove noise, but also blur the edge

How to obtain the edges of an image?

Several derivative filters

Sobel

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

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

Scharr

3	0	-3
10	0	-10
3	0	-3

3	10	3
0	0	0
-3	-10	-3

Prewitt

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

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

Roberts

0	1
-1	0

1	0
0	-1

- How are the other filters derived and how do they relate to the Sobel filter?
- How would you derive a derivative filter that is larger than 3x3?

Edge detectors

- Gradient operators
  - Prewit
  - Sobel
- Marr-Hildreth (Laplacian of Gaussian)
- Canny (Gradient of Gaussian)

Gradient operators edge detector algorithm

1. Compute derivatives
   - In x and y directions
   - Use Sobel or Prewitt filters
2. Find gradient magnitude
3. Threshold gradient magnitude

Sobel

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

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

Prewitt

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

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

Computing image gradients

1. Select your favorite derivative filters.

$$\boldsymbol{S}_x = \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline\end{array} \qquad \boldsymbol{S}_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline\end{array}$$

2. Convolve with the image to compute derivatives.

$$\frac{\partial \boldsymbol{f}}{\partial x} = \boldsymbol{S}_x \otimes \boldsymbol{f} \qquad \frac{\partial \boldsymbol{f}}{\partial y} = \boldsymbol{S}_y \otimes \boldsymbol{f}$$

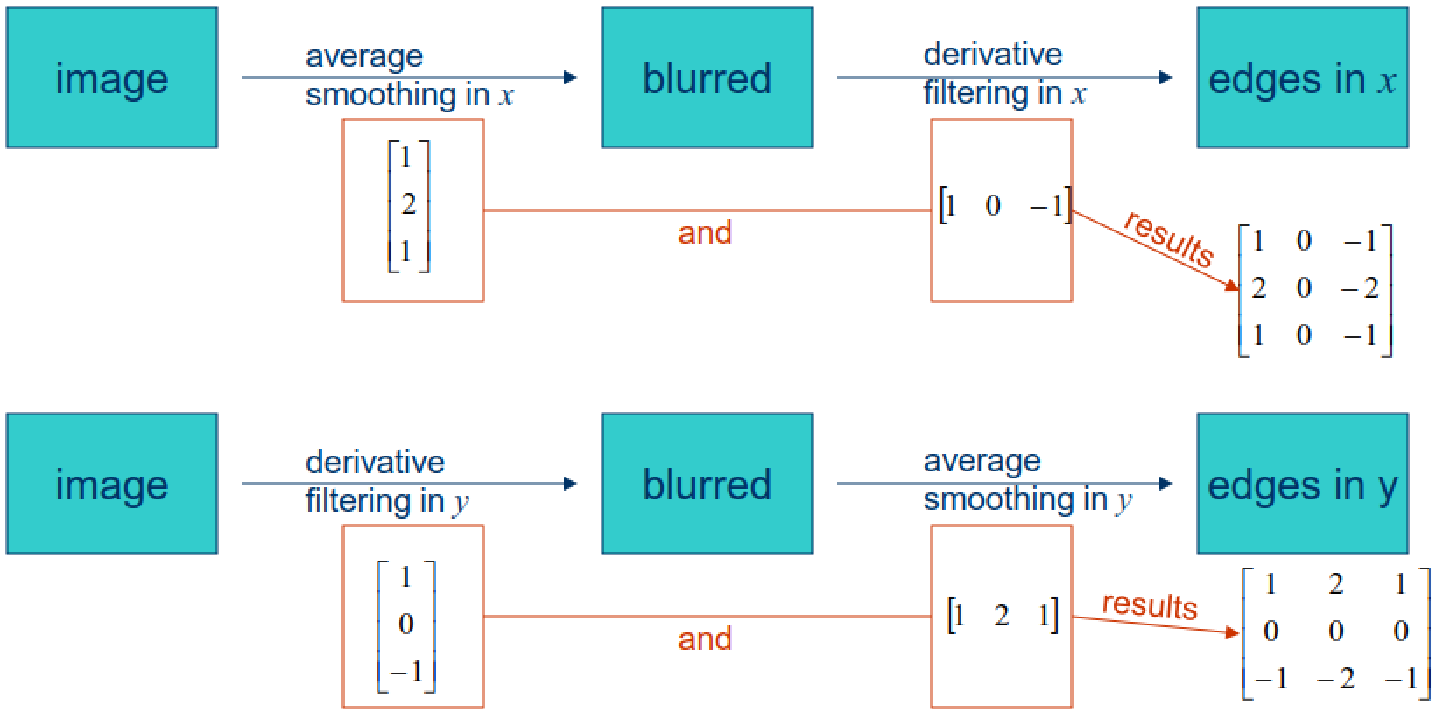
3. Form the image gradient, and compute its direction and amplitude.

$$\nabla \boldsymbol{f} = \left[\frac{\partial \boldsymbol{f}}{\partial x}, \frac{\partial \boldsymbol{f}}{\partial y}\right] \qquad \theta = \tan^{-1}\left(\frac{\partial f}{\partial y} \middle/ \frac{\partial f}{\partial x}\right) \qquad ||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

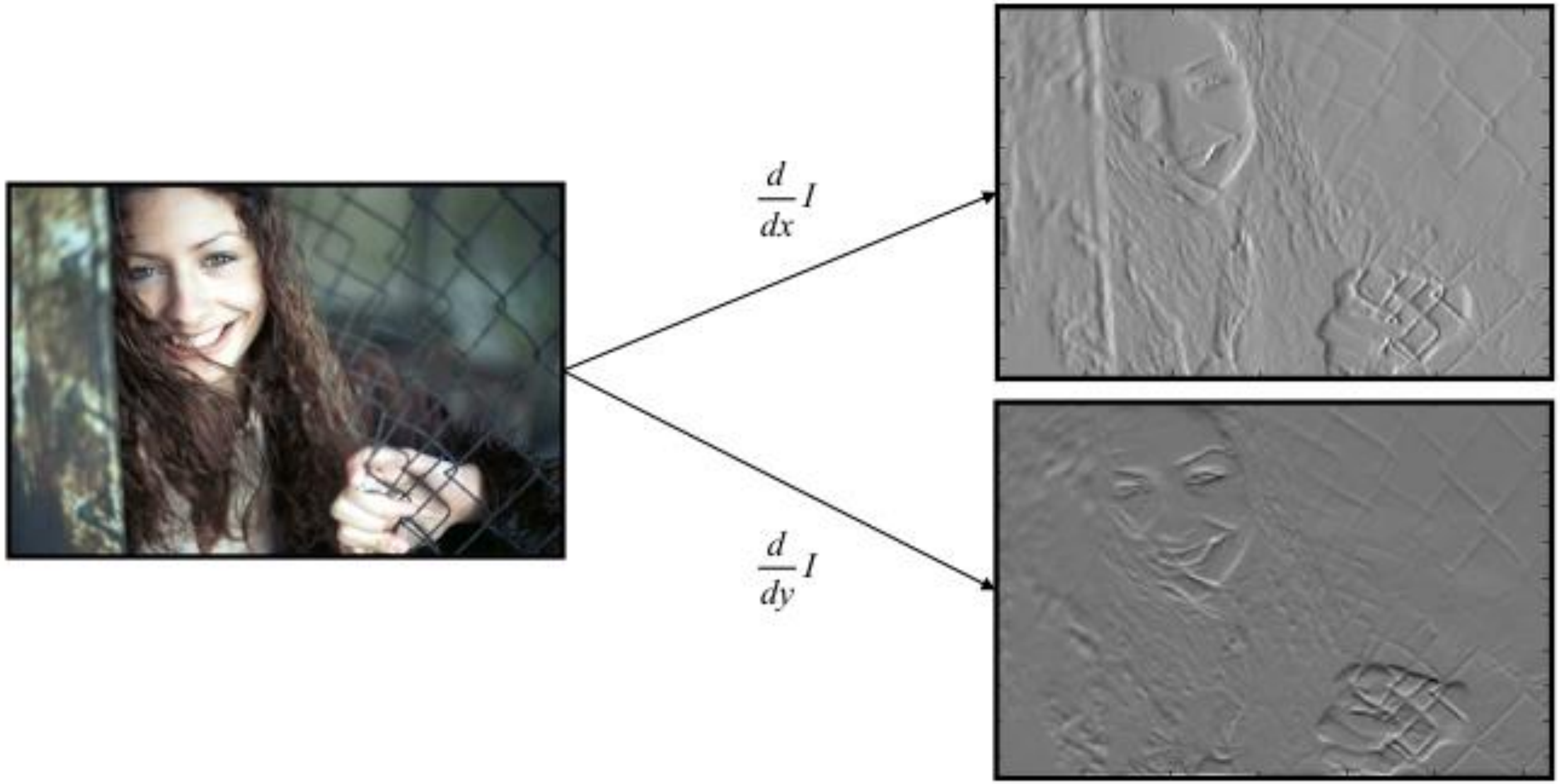
gradient direction amplitude

Sobel edge detector

1. Compute derivatives



Step 1



Sobel edge detector

2. Find gradient magnitude

Step 1

Image I

$$* \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \rightarrow \frac{d}{dx} I$$

$$* \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \rightarrow \frac{d}{dy} I$$

Step 2.

Take each pixel of the derivative of the image and square it

Add the two resulting images

Take a square root of each pixel

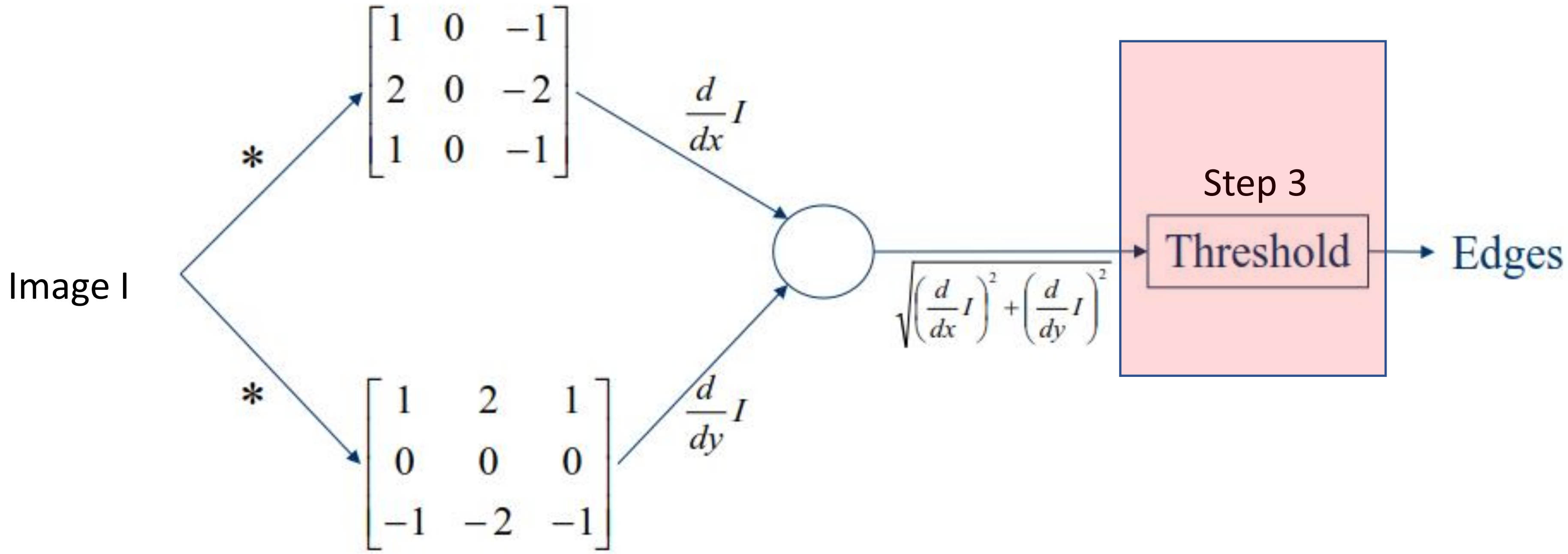
$$\sqrt{\left(\frac{d}{dx} I\right)^2 + \left(\frac{d}{dy} I\right)^2}$$

Step 2

$$\Delta = \sqrt{\left(\frac{d}{dx}I\right)^2 + \left(\frac{d}{dy}I\right)^2}$$

Sobel edge detector

3. Threshold

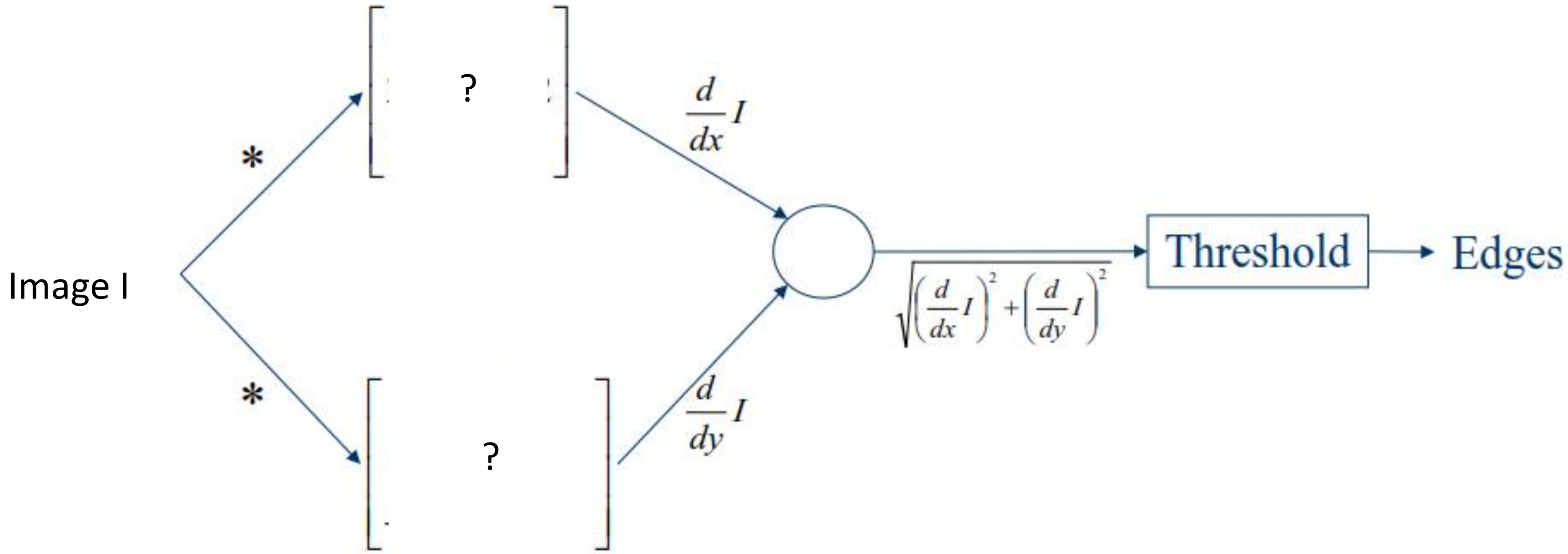


Sobel Edge Detector

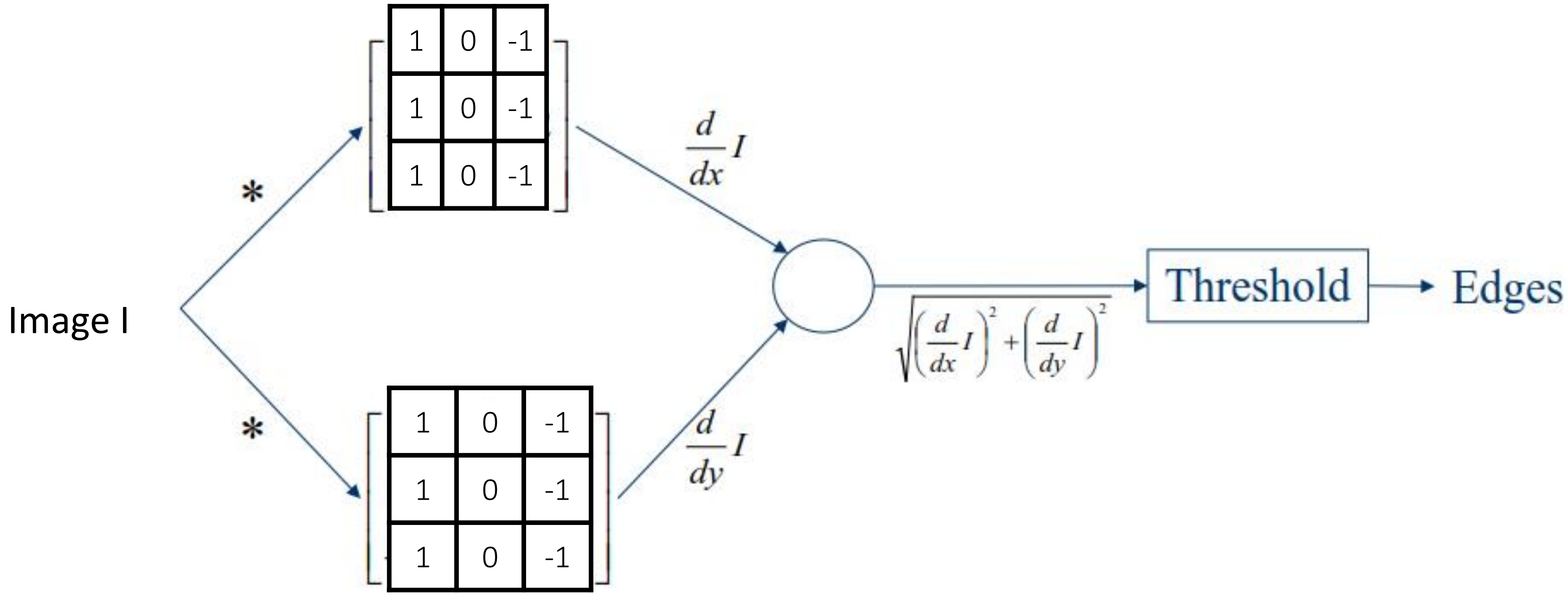
$$\Delta = \sqrt{\left(\frac{d}{dx}I\right)^2 + \left(\frac{d}{dy}I\right)^2}$$

$$\Delta \geq Threshold = 100$$

Prewitt edge detector



Prewitt edge detector

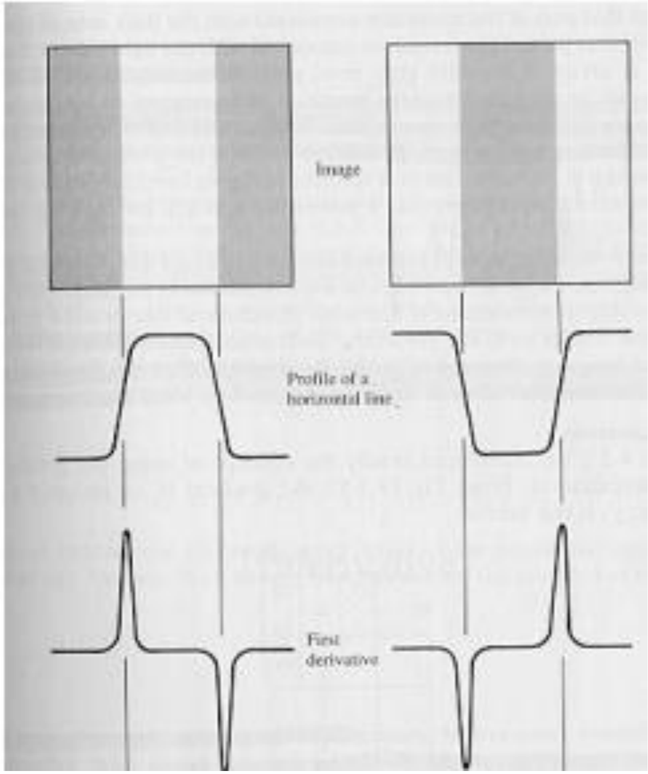


Edge detectors

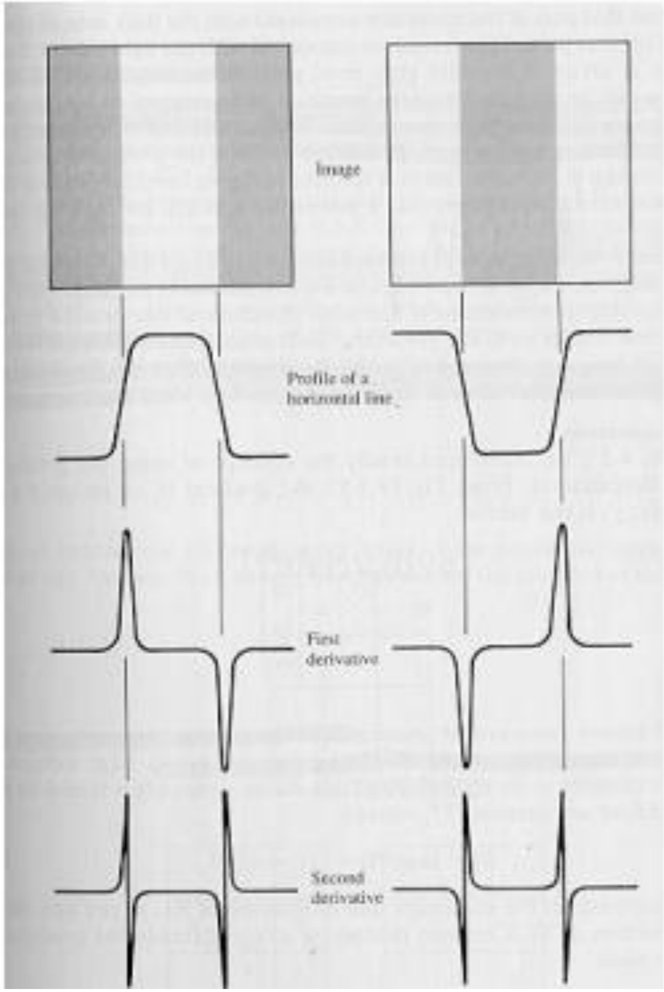
- Gradient operators
  - Prewit
  - Sobel
- **Marr-Hildreth (Laplacian of Gaussian)**
- Canny (Gradient of Gaussian)

Where are the edges ?

- First derivative ?
  - Maxima or minima



Where are the edges ?



- First derivative ?
  - Maxima or minima
- Second derivative?
  - Zero-crossing

Laplace filter

Basically a second derivative filter.

- We can use finite differences to derive it, as with first derivative filter.

first-order finite difference

$$f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h} \longrightarrow$$

1D derivative filter

1	0	-1

second-order finite difference

$$f''(x) = \lim_{h \to 0} \frac{f(x + h) - 2f(x) + f(x - h)}{h^2} \longrightarrow$$

Laplace filter

1	-2	1

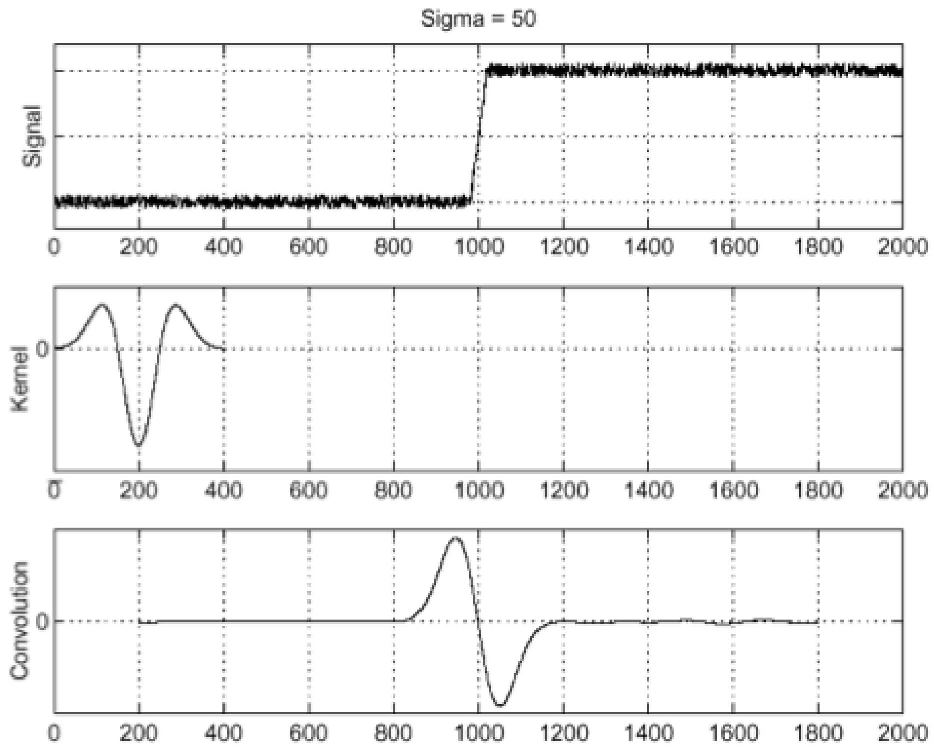
Laplacian of Gaussian (LoG) filter

As with derivative, we can combine Laplace filtering with Gaussian filtering

input

Laplacian of Gaussian

output



"zero crossings" at edges

Laplace and LoG filtering examples

Laplacian of Gaussian filtering

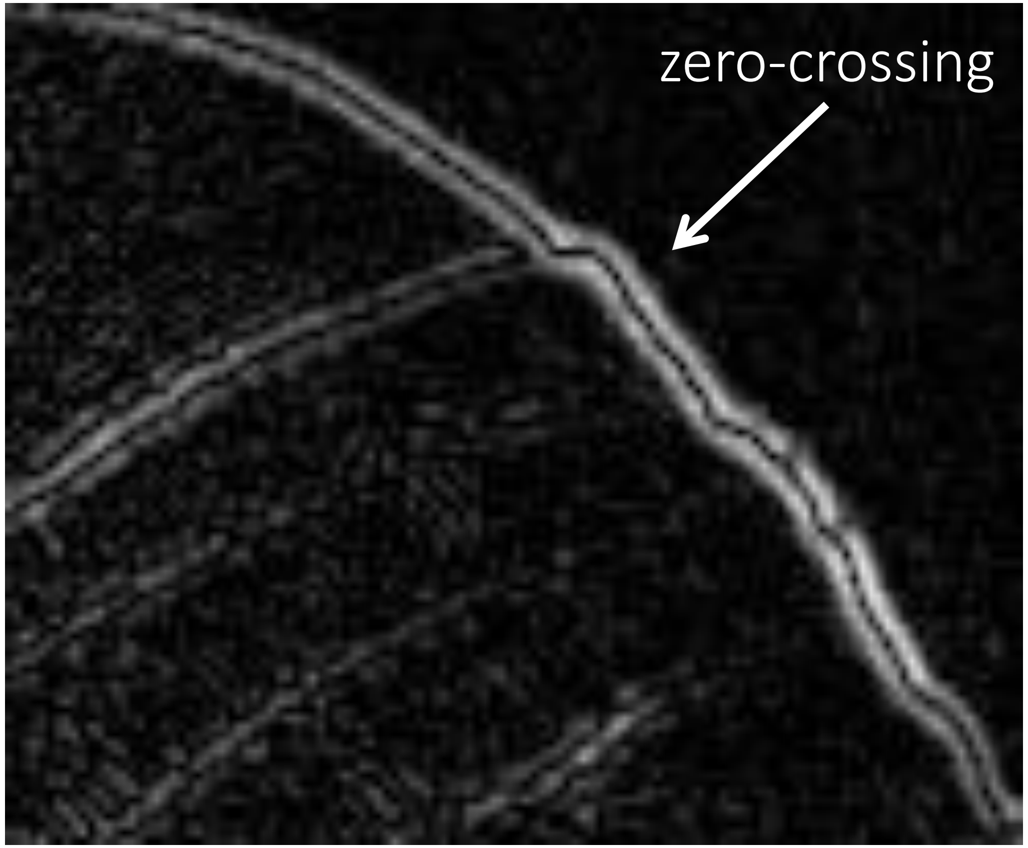
Laplace filtering

Laplacian of Gaussian vs Derivative of Gaussian

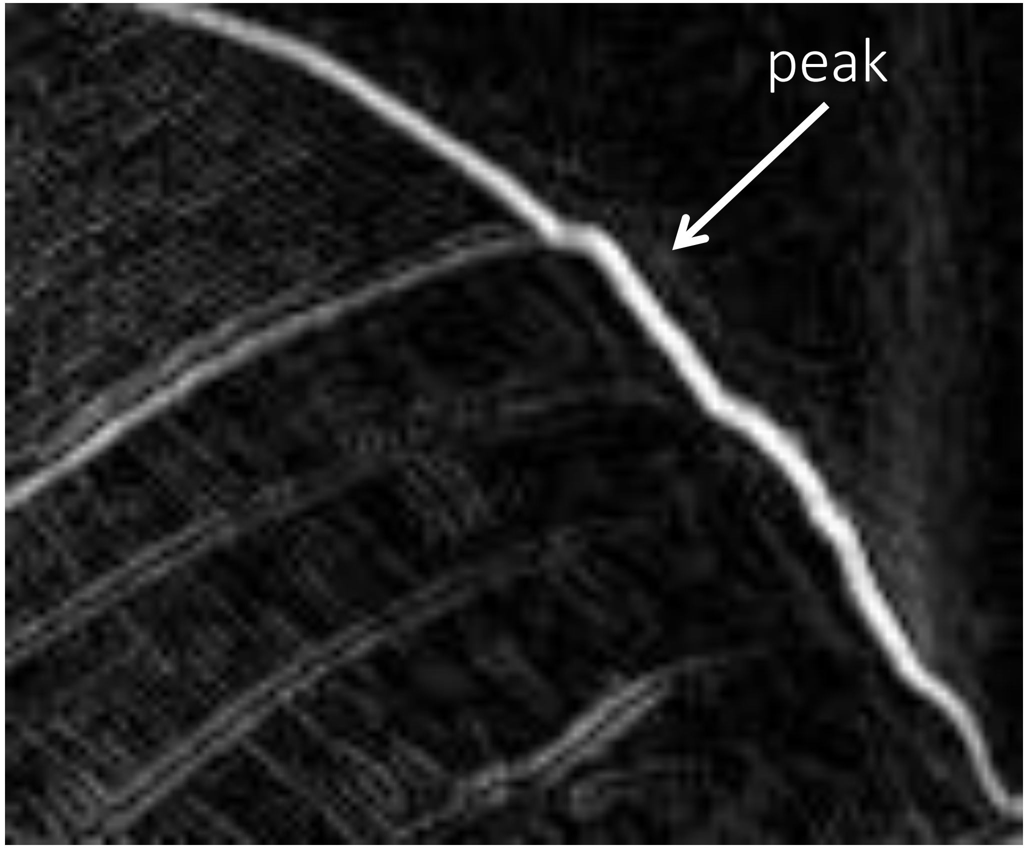
Laplacian of Gaussian filtering

Derivative of Gaussian filtering

Laplacian of Gaussian vs Derivative of Gaussian



Laplacian of Gaussian filtering



Derivative of Gaussian filtering

Zero crossings are more accurate at localizing edges

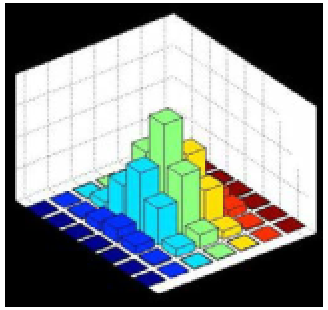
Marr-Hildreth edge detector algorithm

1. Smooth image by Gaussian filtering
2. Apply Laplacian to smoothed image
   - Used in mechanics, electromagnetics, wave theory, quantum mechanics
3. Find Zero crossings

Marr-Hildreth edge detector algorithm

1. Smooth image by Gaussian filtering
   - Gaussian smoothing

$$\overbrace{S}^{\text{smoothed image}} = \overbrace{g}^{\text{Gaussian filter}} * \overbrace{I}^{\text{image}} \qquad g = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



2. Apply Laplacian to smoothed image
   - Find Laplacian

$$\Delta^2 S = \overbrace{\frac{\partial^2}{\partial x^2}}^{\text{second order derivative in } x} S + \overbrace{\frac{\partial^2}{\partial y^2}}^{\text{second order derivative in } y} S$$

- ∇ is used for gradient (first derivative)
- Δ^2 is used for Laplacian (Second derivative)

Marr-Hildreth edge detector algorithm

$$\Delta^2 S = \overbrace{\frac{\partial^2}{\partial x^2} S}^{\text{second order derivative in } x} + \overbrace{\frac{\partial^2}{\partial y^2} S}^{\text{second order derivative in } y}$$

- ∇ is used for gradient (first derivative)
- Δ^2 is used for Laplacian (Second derivative)

$$\overbrace{S}^{\text{smoothed image}} = \overbrace{g}^{\text{Gaussian filter}} * \overbrace{I}^{\text{image}} \qquad g = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$\Delta^2 S = \Delta^2 (g * I) = (\Delta^2 g) * I$$

This is more efficient computationally

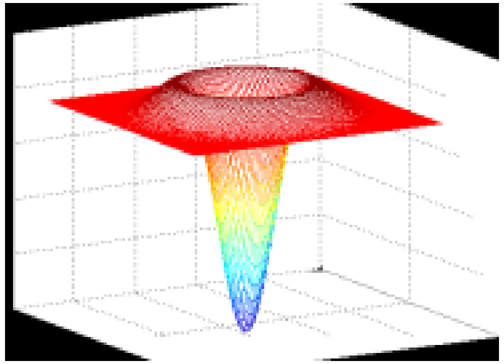
Marr-Hildreth edge detector algorithm

$$\Delta^2 S = \Delta^2 (g * I) = (\Delta^2 g) * I$$

$$g = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The second derivative of a gaussian

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



Marr-Hildreth edge detector algorithm

$$\Delta^2 S = \Delta^2 (g * I) = (\Delta^2 g) * I$$

$$g = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The second derivative of a gaussian

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

Marr-Hildreth edge detector algorithm

$$\Delta^2 S = \Delta^2 (g * I) = (\Delta^2 g) * I$$

$$g = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The second derivative of a gaussian

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

Given a σ, Compute LoG for each x,y to obtain a Kernel

Marr-Hildreth edge detector algorithm

$$\Delta^2 S = \Delta^2 (g * I) = (\Delta^2 g) * I$$

$$g = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The second derivative of a gaussian

For $\sigma = 1.4$

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

$$LoG(0,0) \approx -0.1624$$

Given a σ, Compute LoG for each x,y to obtain a Kernel

Marr-Hildreth edge detector algorithm

$$\Delta^2 S = \Delta^2 (g * I) = (\Delta^2 g) * I$$

$$g = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The second derivative of a gaussian

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

Given a σ, Compute LoG for each x,y to obtain a Kernel

For $\sigma = 1.4$

$$\begin{pmatrix} 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 2 & 5 & 0 & -23 & -40 & -23 & 0 & 5 & 2 \\ 2 & 5 & 3 & -12 & -23 & -12 & 3 & 5 & 2 \\ 3 & 3 & 5 & 3 & 0 & 3 & 5 & 3 & 3 \\ 0 & 2 & 3 & 5 & 5 & 5 & 3 & 2 & 0 \\ 0 & 0 & 3 & 2 & 2 & 2 & 3 & 0 & 0 \end{pmatrix}$$

Marr-Hildreth edge detector algorithm

1. Smooth image by Gaussian filtering
2. Apply Laplacian to smoothed image
    - Used in mechanics, electromagnetics, wave theory, quantum mechanics
3. **Find Zero crossings**
    - Scan along each row, record an edge point at the location of zero-crossing.
    - Repeat above step along each column

Marr-Hildreth edge detector algorithm

3. Find Zero crossings

- Scan along each row, record an edge point at the location of zero-crossing.
- Repeat above step along each column

```
from skimage.filters import laplace
import numpy as np

lap = np.sign(laplace(image))
lap = np.pad(lap, ((0, 1), (0, 1)))
diff_x = lap[:-1, :-1] - lap[:-1, 1:] < 0
diff_y = lap[:-1, :-1] - lap[1:, :-1] < 0

edges =  np.logical_or(diff_x, diff_y).astype(float)
```

Marr-Hildreth edge detector algorithm

3. Find Zero crossings

- Scan along each row, record an edge point at the location of zero-crossing.
- Repeat above step along each column

```
from skimage.filters import laplace
import numpy as np

lap = np.sign(laplace(image))
lap = np.pad(lap, ((0, 1), (0, 1)))
diff_x = lap[:-1, :-1] - lap[:-1, 1:] < 0
diff_y = lap[:-1, :-1] - lap[1:, :-1] < 0

edges =  np.logical_or(diff_x, diff_y).astype(float)
```

returns -1 if x < 0, 0 if x==0, 1 if x > 0.

Marr-Hildreth edge detector algorithm

3. Find Zero crossings

- Scan along each row, record an edge point at the location of zero-crossing.
- Repeat above step along each column

```
from skimage.filters import laplace
import numpy as np

lap = np.sign(laplace(image))
lap = np.pad(lap, ((0, 1), (0, 1)))
diff_x = lap[:-1, :-1] - lap[:-1, 1:] < 0
diff_y = lap[:-1, :-1] - lap[1:, :-1] < 0

edges =  np.logical_or(diff_x, diff_y).astype(float)
```

returns -1 if x < 0, 0 if x==0, 1 if x > 0.

Add extra row, and extra column

Marr-Hildreth edge detector algorithm

3. Find Zero crossings

- Scan along each row, record an edge point at the location of zero-crossing.
- Repeat above step along each column

```
from skimage.filters import laplace
import numpy as np

lap = np.sign(laplace(image))
lap = np.pad(lap, ((0, 1), (0, 1)))
diff_x = lap[:-1, :-1] - lap[:-1, 1:] < 0
diff_y = lap[:-1, :-1] - lap[1:, :-1] < 0

edges =  np.logical_or(diff_x, diff_y).astype(float)
```

returns -1 if x < 0, 0 if x==0, 1 if x > 0.

Add extra row, and extra column

Zero-crossing X direction

Zero-crossing Y direction

Marr-Hildreth edge detector algorithm

3. Find Zero crossings (Another implementation)

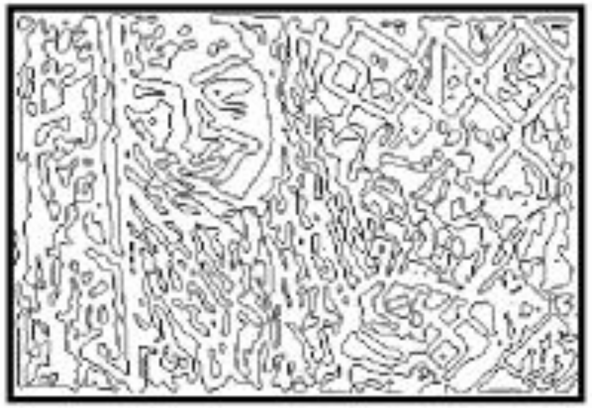
- Four cases of zero-crossings :
  - {+,-}
  - {+,0,-}
  - {-,+}
  - {-,0,+}
- Slope of zero-crossing {a, -b} is |a+b|.
- To mark an edge
  - compute slope of zero-crossing
  - Apply a threshold to slope

Example

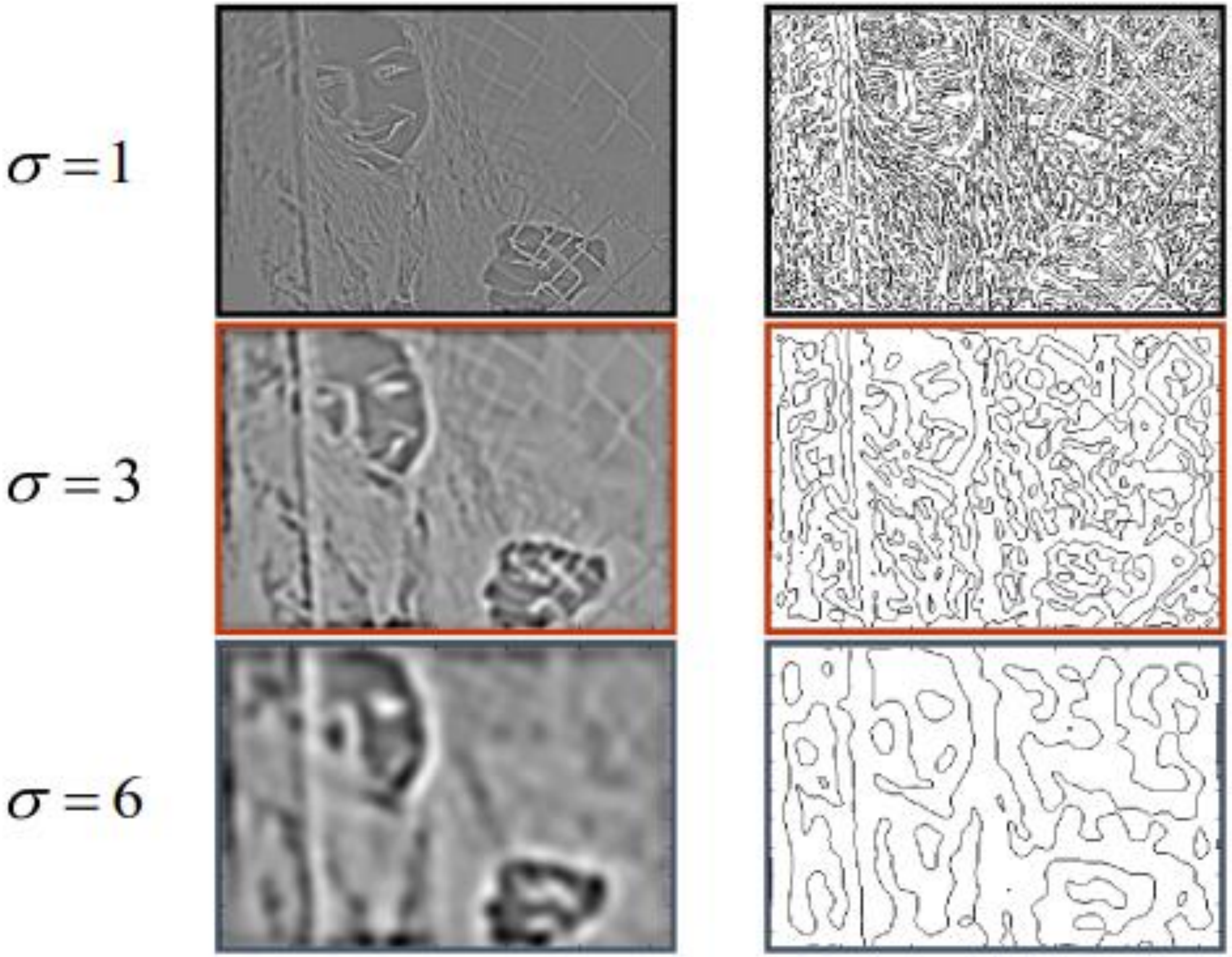
I

$I * \left(\Delta^2 g\right)$

Zero crossings of $\Delta^2 S$



Example

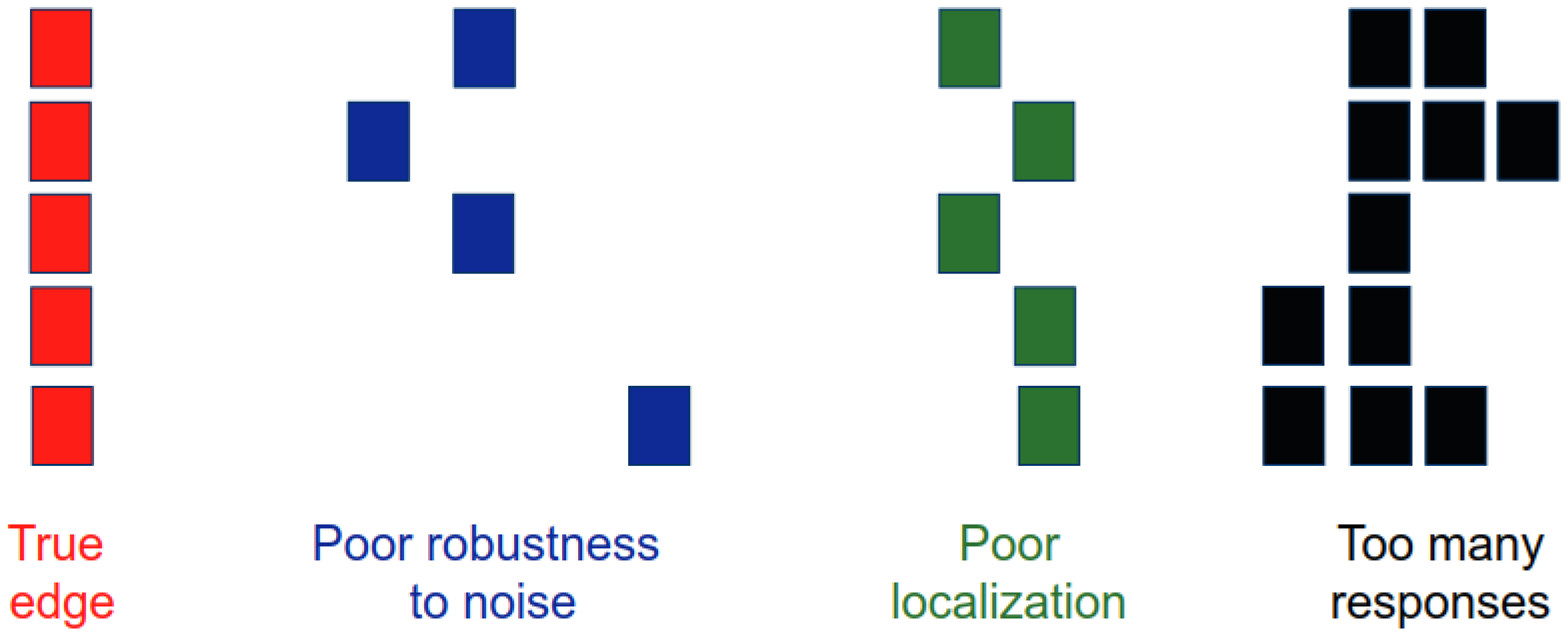


Edge detectors

- Gradient operators
  - Prewit
  - Sobel
- Marr-Hildreth (Laplacian of Gaussian)
- **Canny (Gradient of Gaussian)**

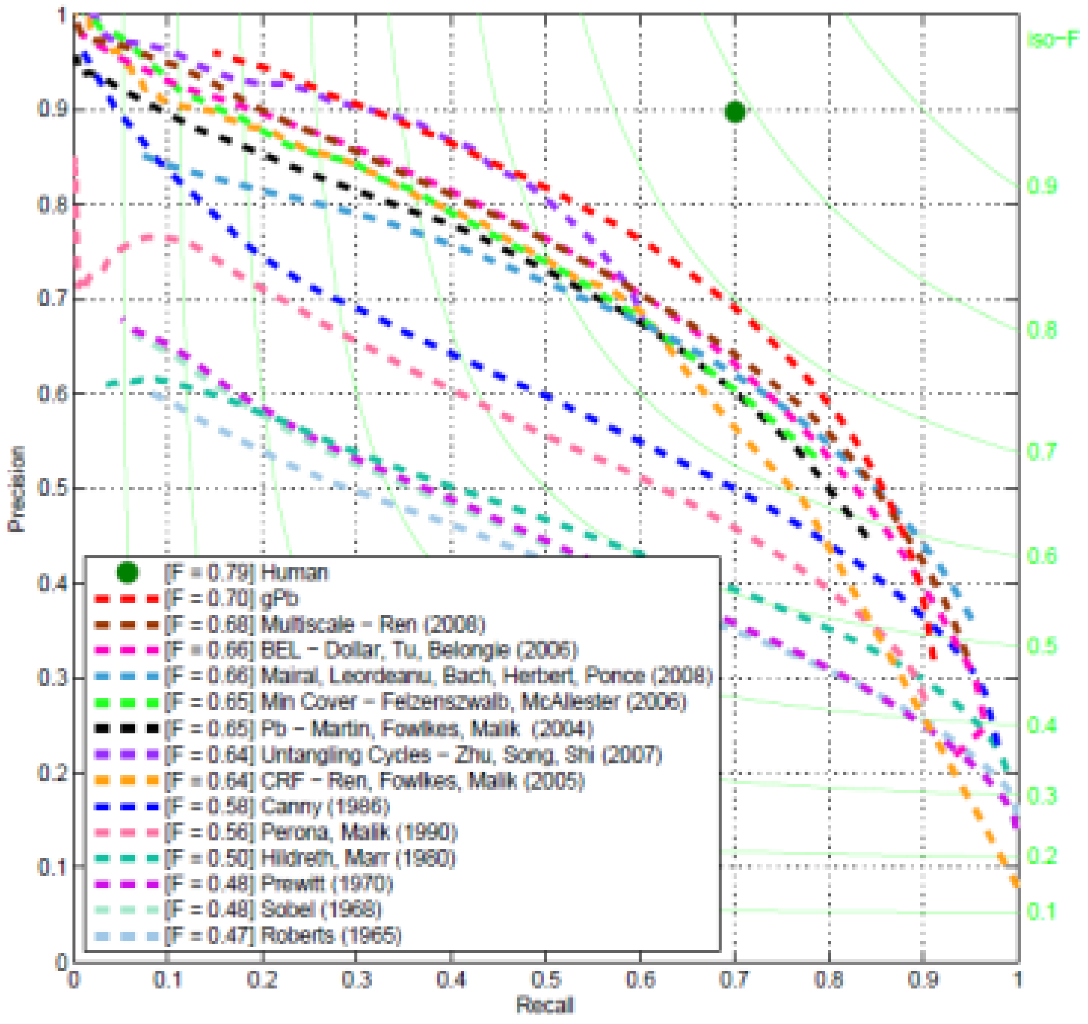
Design Criteria for Edge Detection

- Good detection: find all real edges, ignoring noise or other artifacts
- Good localization
  - as close as possible to the true edges
  - one point only for each true edge point

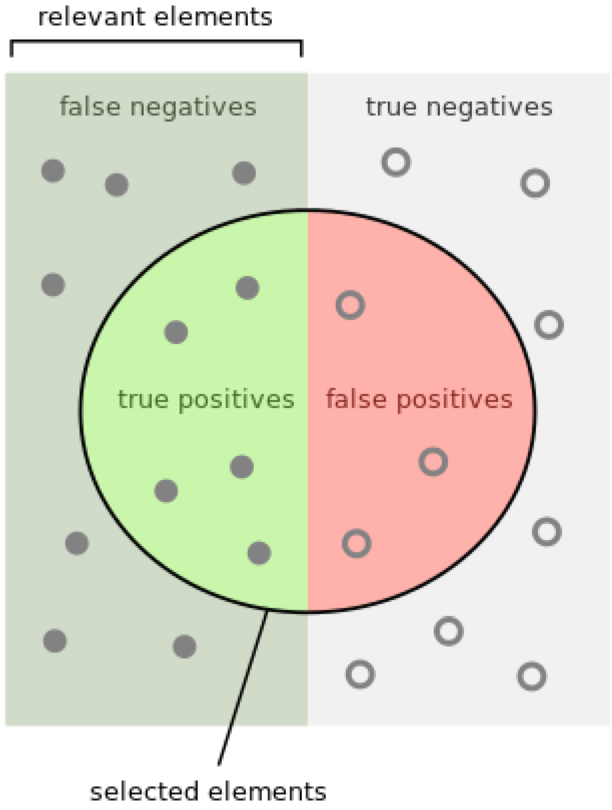


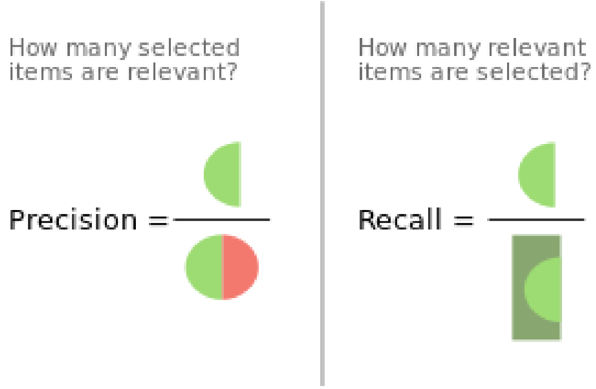
45 years of boundary detection

[Pre deep learning]

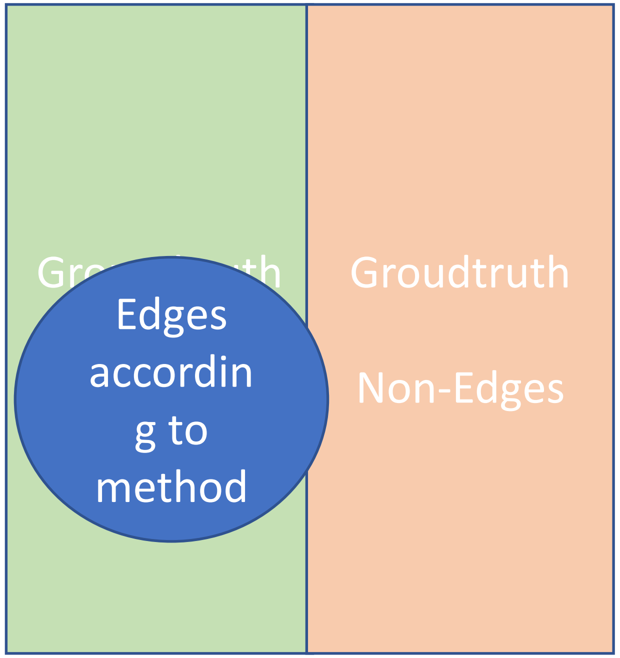


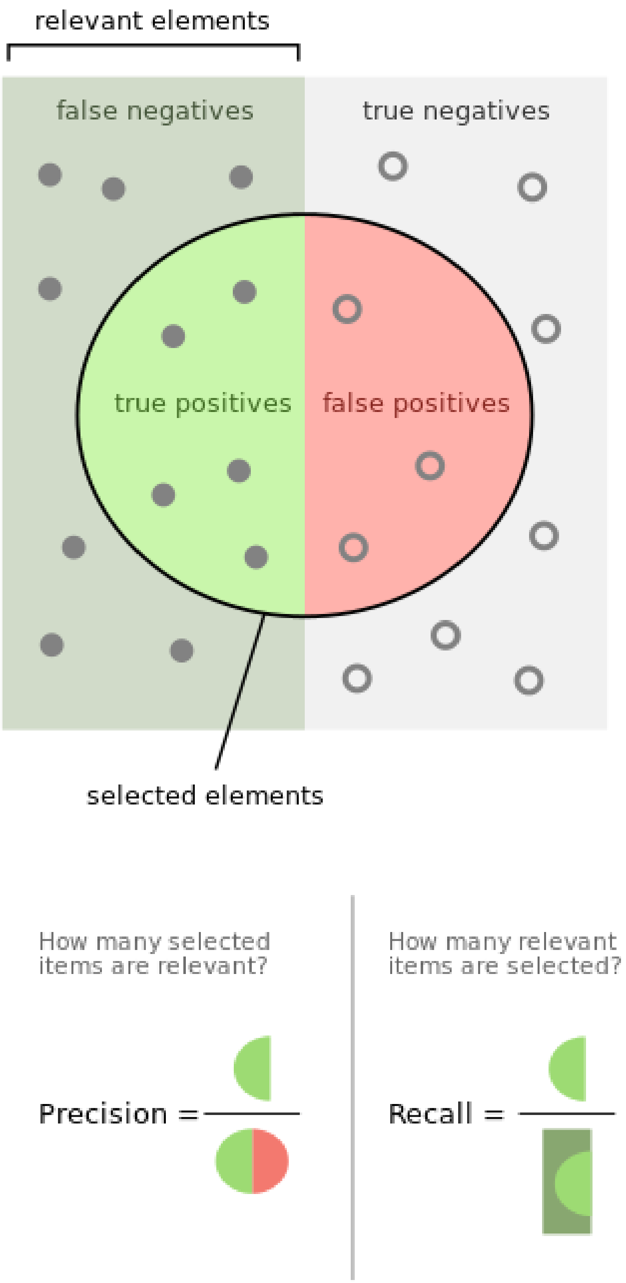
Precision Recall



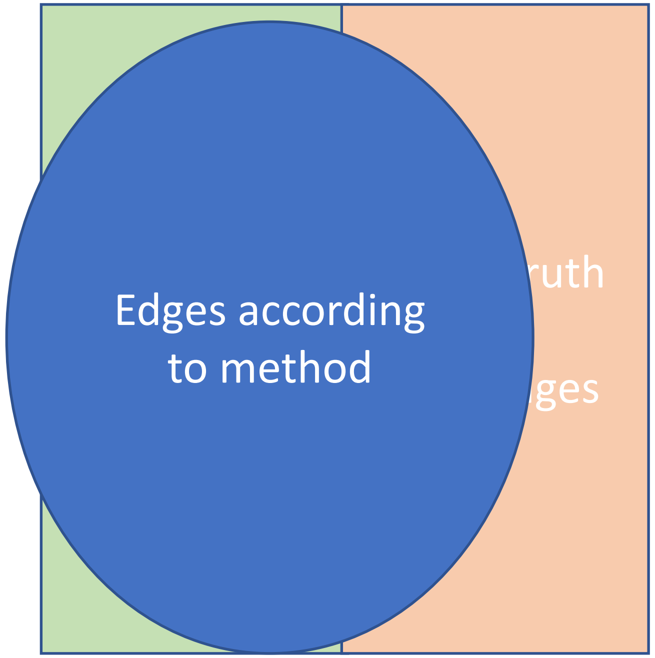


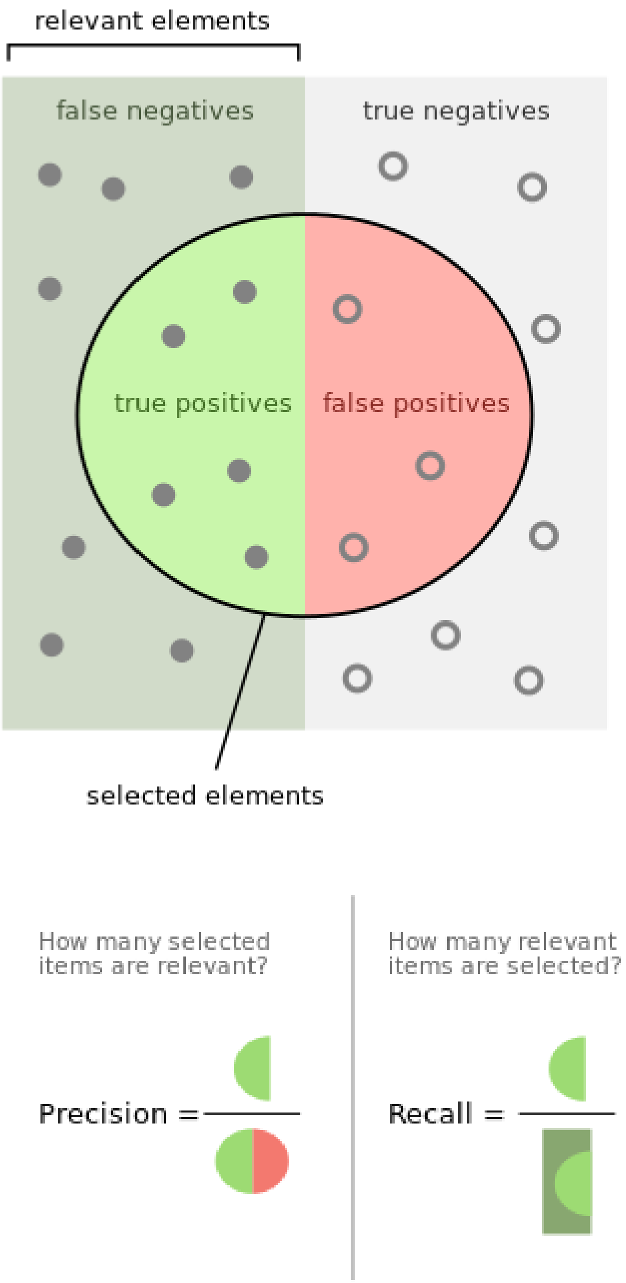
Precision Recall



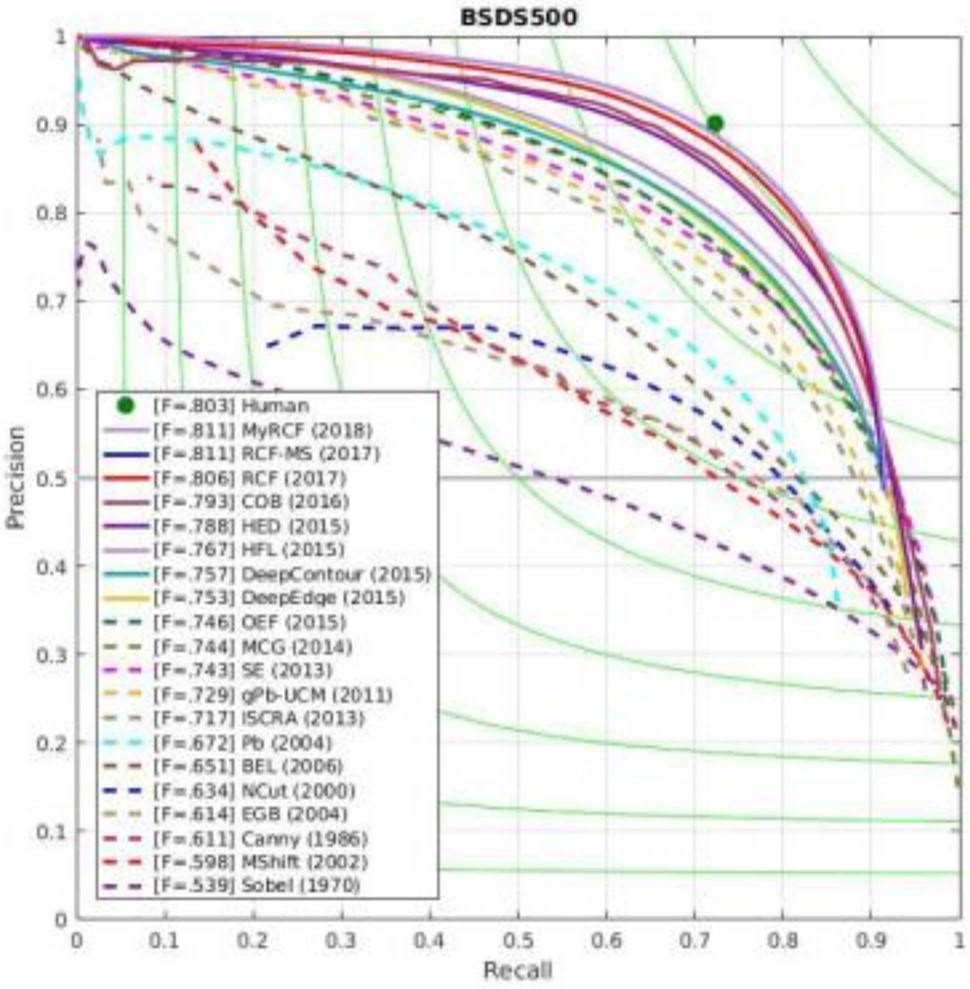


Precision Recall





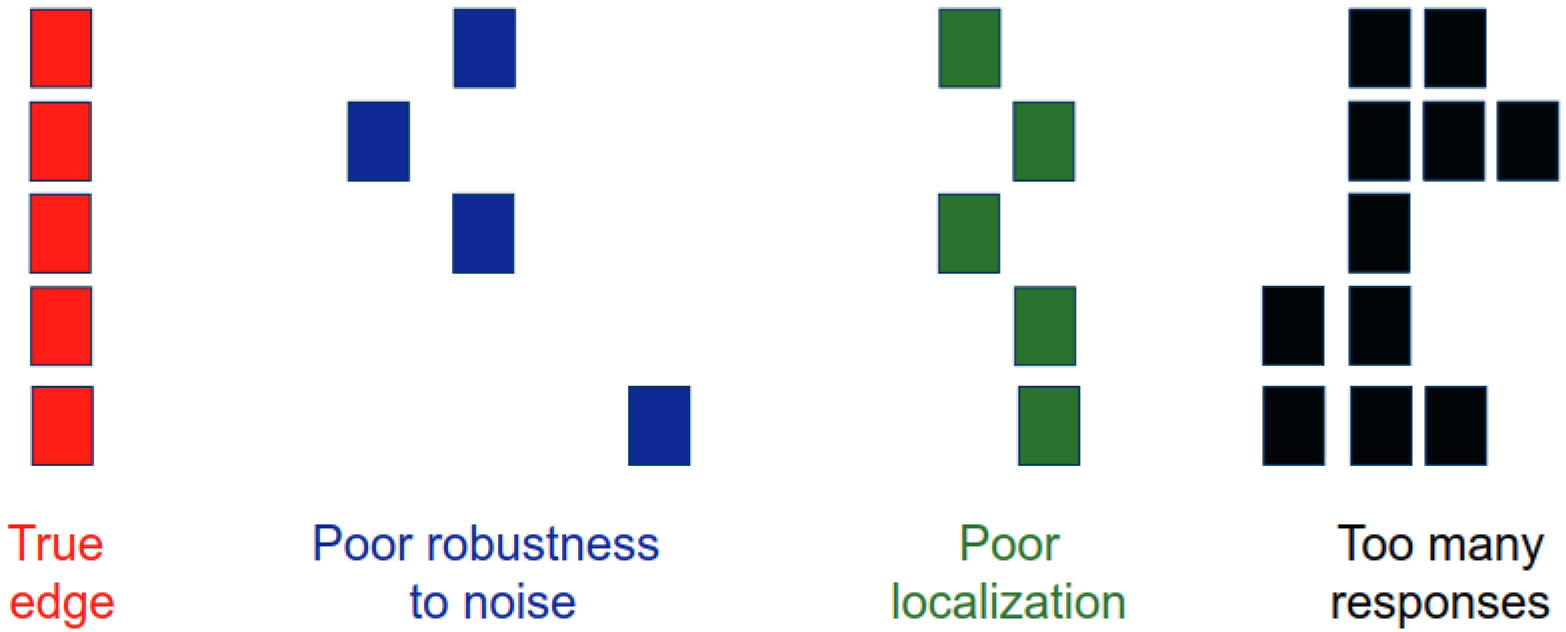
Edge Detection with Deep Learning



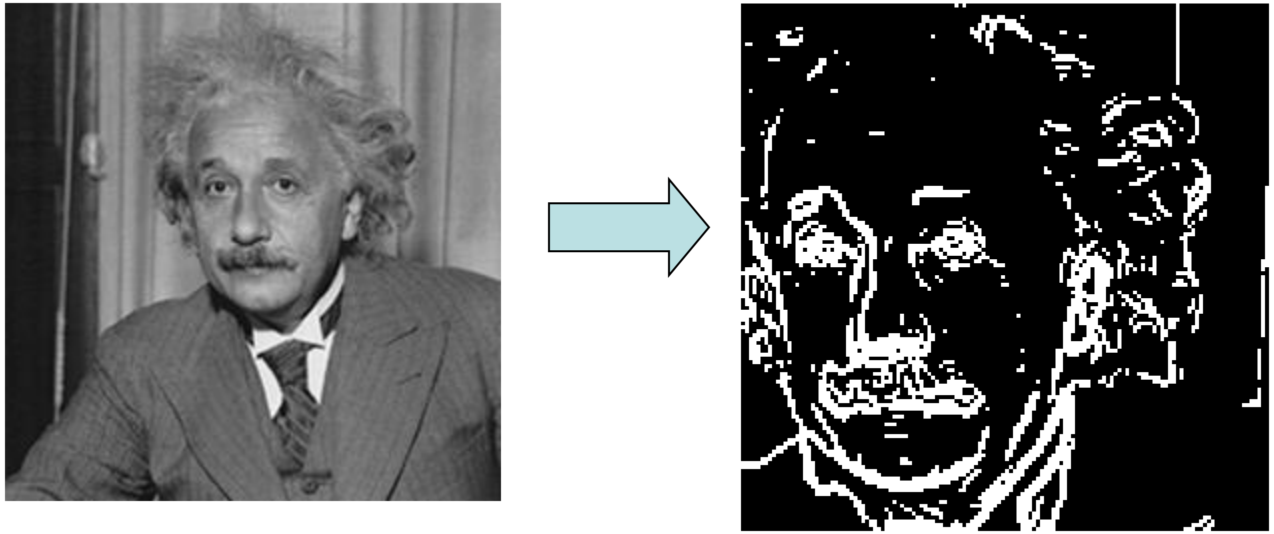
Canny edge detector

Design Criteria for Edge Detection

- Good detection: find all real edges, ignoring noise or other artifacts
- Good localization
  - as close as possible to the true edges
  - one point only for each true edge point



Problems



- We get thick edges
- Redundant, especially if we going to be searching in places where edges are found

Solution

- Identify the local maximums
- Called “non-maximal suppression”

Canny Edge detector algorithm

1. Smooth image with Gaussian filter
2. Compute derivative of filtered image
3. Find magnitude and orientation of gradient
4. Apply “Non-maximum Suppression”
5. Apply “Hysteresis Threshold”

Canny Edge detector algorithm

1. **Smooth image with Gaussian filter**

$$S = I * g(x, y) = g(x, y) * I$$

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

2. **Compute derivative of filtered** ·

$$\nabla S = \nabla(g * I) = (\nabla g) * I$$

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$

$$\nabla S = \begin{bmatrix} g_x \\ g_y \end{bmatrix} * I = \begin{bmatrix} g_x * I \\ g_y * I \end{bmatrix}$$

Canny Edge detector algorithm

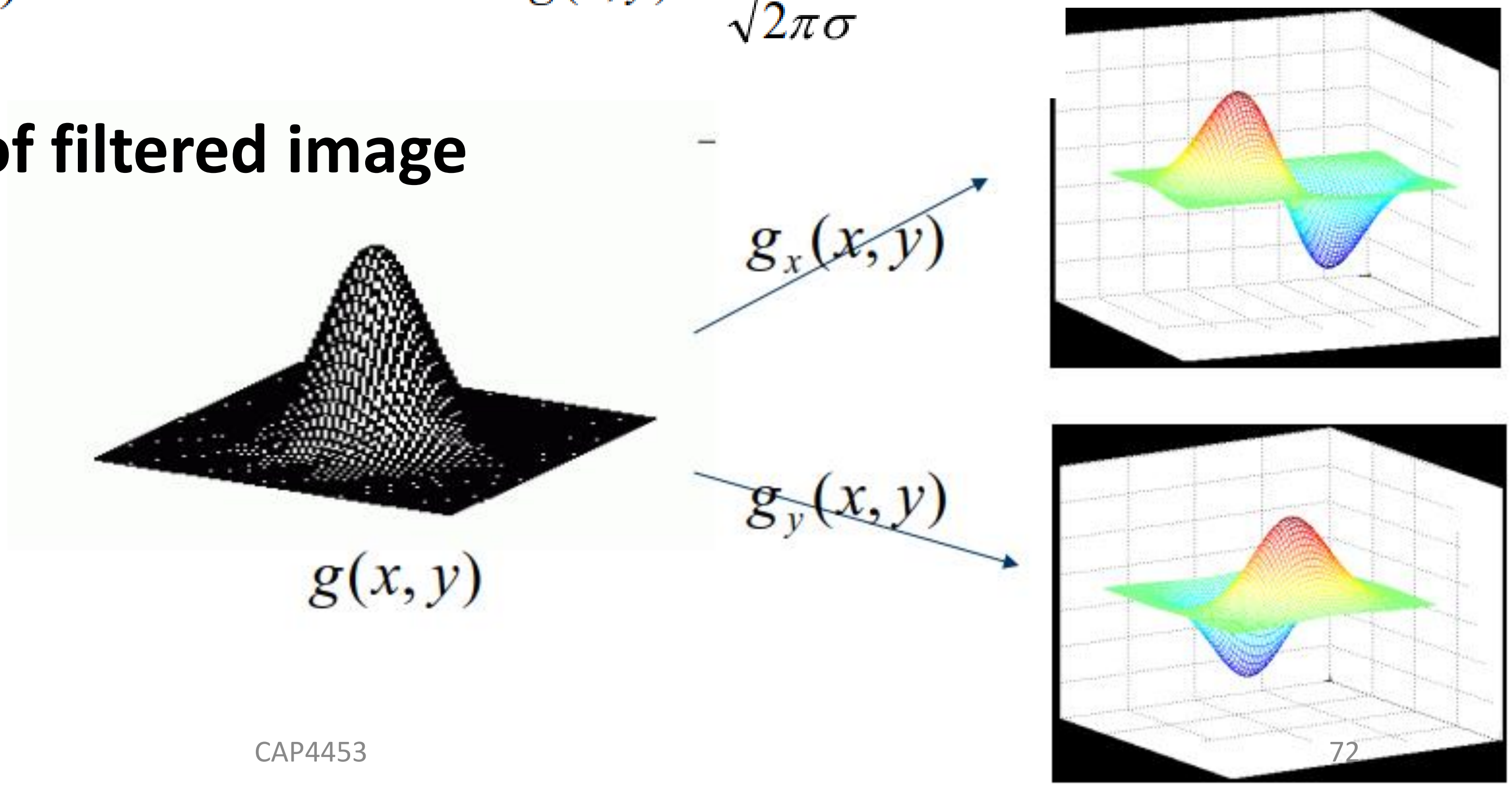
1. **Smooth image with Gaussian filter**

$$S = I * g(x, y) = g(x, y) * I \qquad g(x, y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

2. **Compute derivative of filtered image**

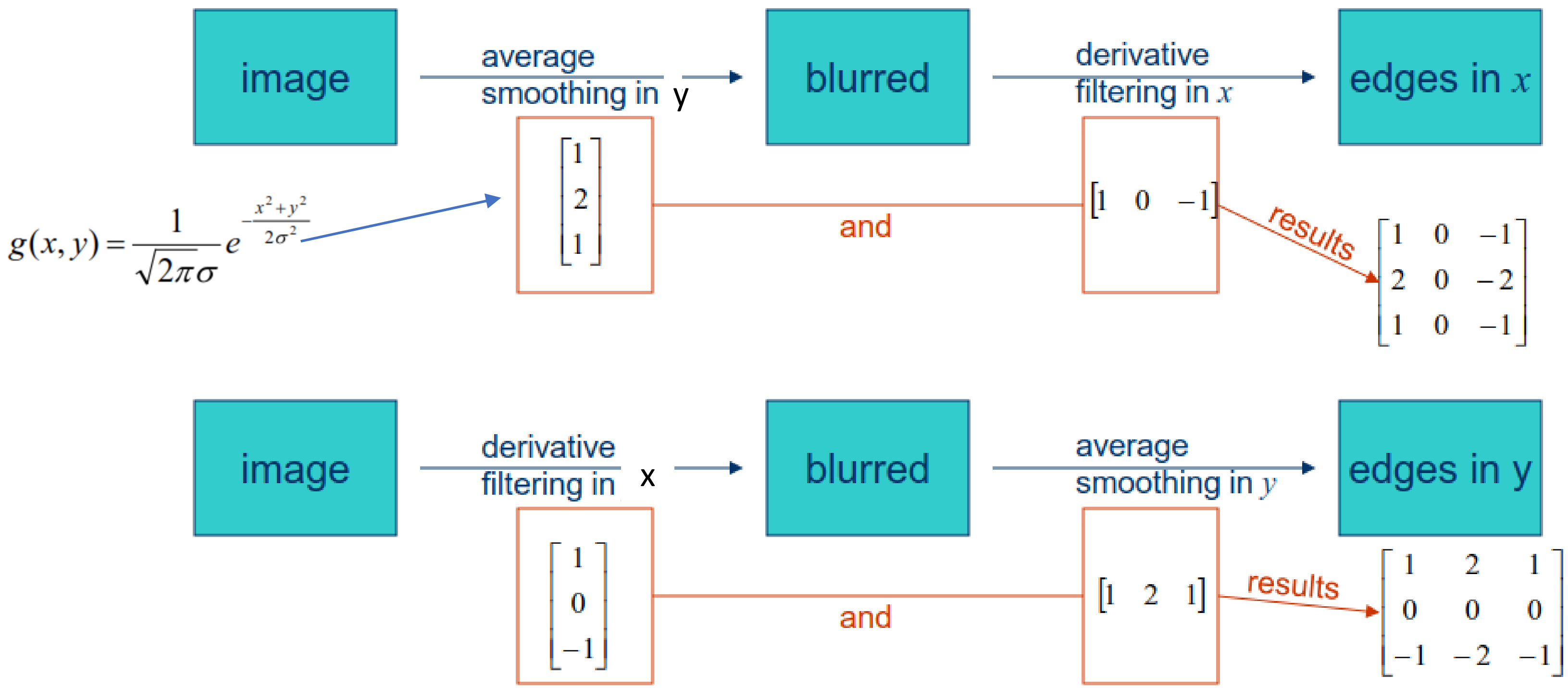
$$\nabla S = \begin{bmatrix} g_x \\ g_y \end{bmatrix} * I = \begin{bmatrix} g_x * I \\ g_y * I \end{bmatrix}$$

$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$



Canny Edge detector algorithm

1. **Smooth image with Gaussian filter**
2. **Compute derivative of filtered image**



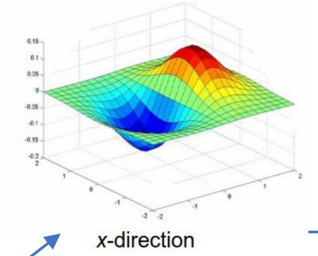
Canny Edge detector algorithm

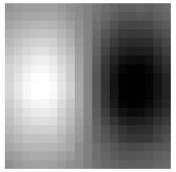
1. Smooth image with Gaussian filter
2. Compute derivative of filtered image
3. **Find magnitude and orientation of gradient**
4. Apply “Non-maximum Suppression”
5. Apply “Hysteresis Threshold”

Canny Edge detector algorithm

3. Find magnitude and orientation of gradient

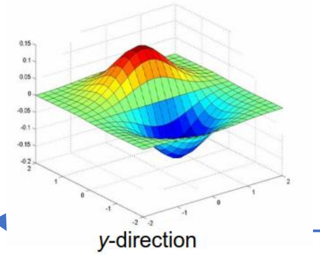
original image (Lena)

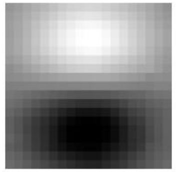




S_x

X-Derivative of Gaussian



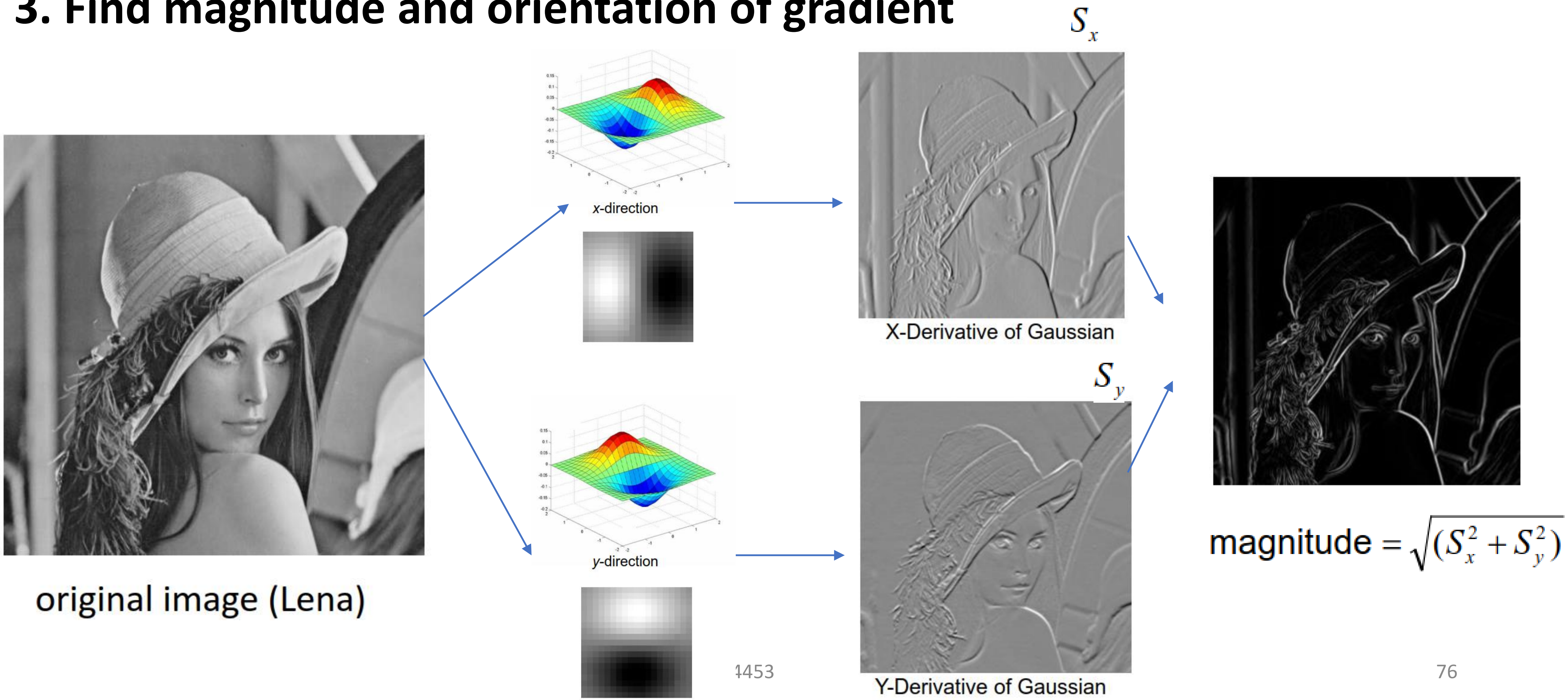


S_y

Y-Derivative of Gaussian

Canny Edge detector algorithm

3. Find magnitude and orientation of gradient

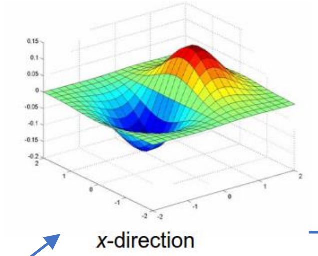


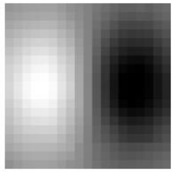
original image (Lena)

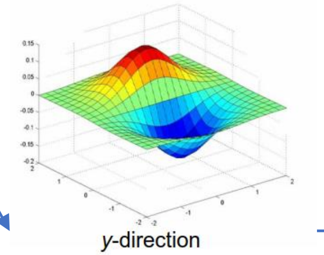
Canny Edge detector algorithm

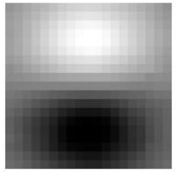
3. Find magnitude and orientation of gradient

original image (Lena)







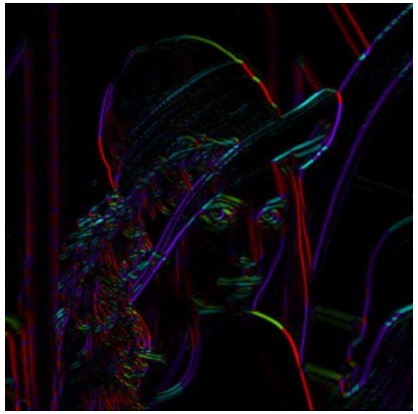


S_x

X-Derivative of Gaussian

S_y

Y-Derivative of Gaussian



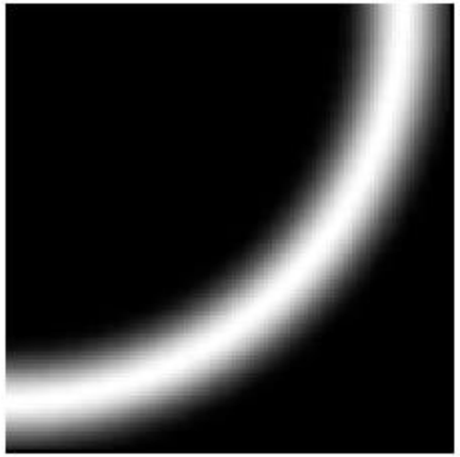
$$Theta = \mathrm{atan2}(S_y, S_x)$$

Canny Edge detector algorithm

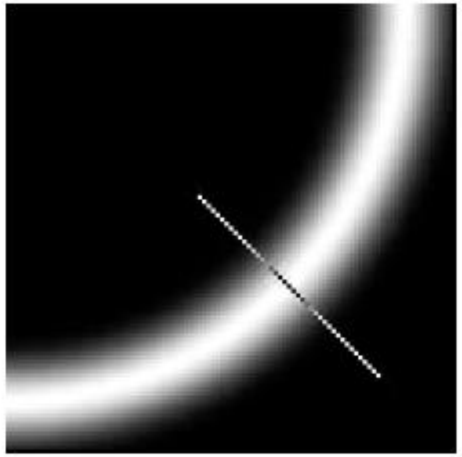
1. Smooth image with Gaussian filter
2. Compute derivative of filtered image
3. Find magnitude and orientation of gradient
4. **Apply "Non-maximum Suppression"**
5. Apply "Hysteresis Threshold"

Canny Edge detector algorithm

4. **Apply “Non-maximum Suppression”**



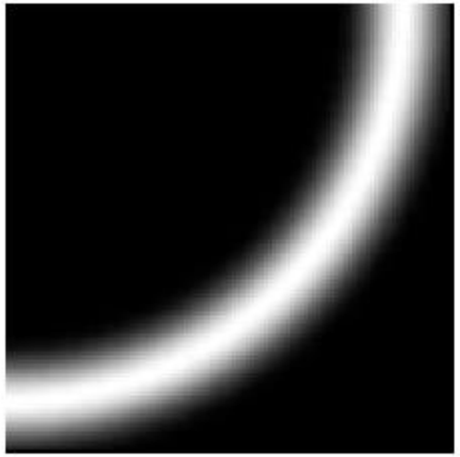
Goal: keep pixels along the curve where magnitude is largest



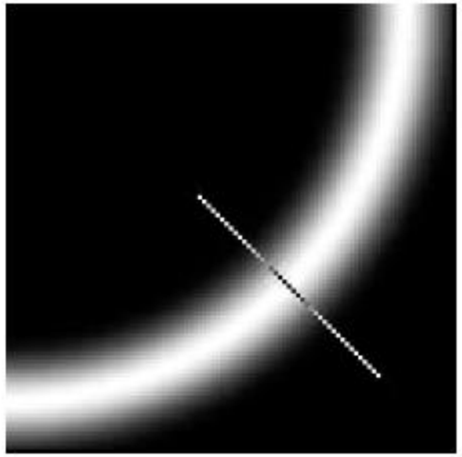
How to: looking for a maximum along a slice normal to the curve

Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**



Goal: keep pixels along the curve where magnitude is largest



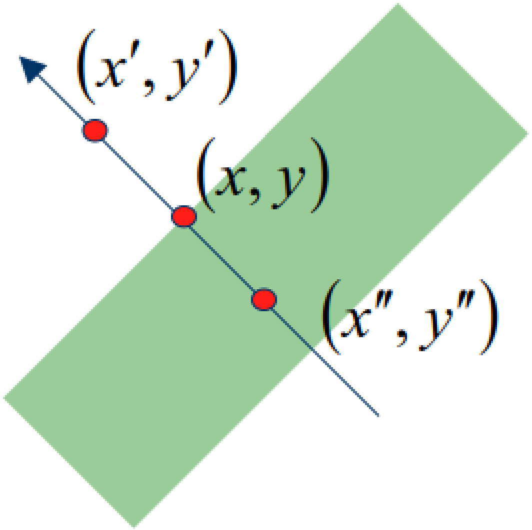
That is the direction of the gradient !

How to: looking for a maximum along a slice normal to the curve

Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**

- Suppress the pixels in $|\nabla S|$ which are not local maximum



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

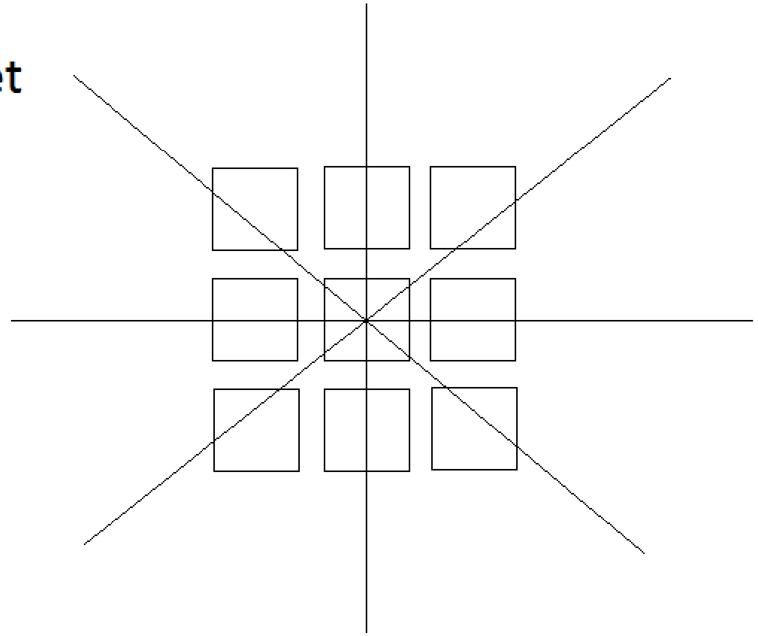
$\mathbf{x}'$ and $\mathbf{x}''$ are the neighbors of $\mathbf{x}$ along normal direction to an edge

Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**

Consider a 3x3 set

of pixels.



We need to examine triples

along the directions shown

to see if the center pixel is a

peak (in magnitude) compared

to its two neighbors.

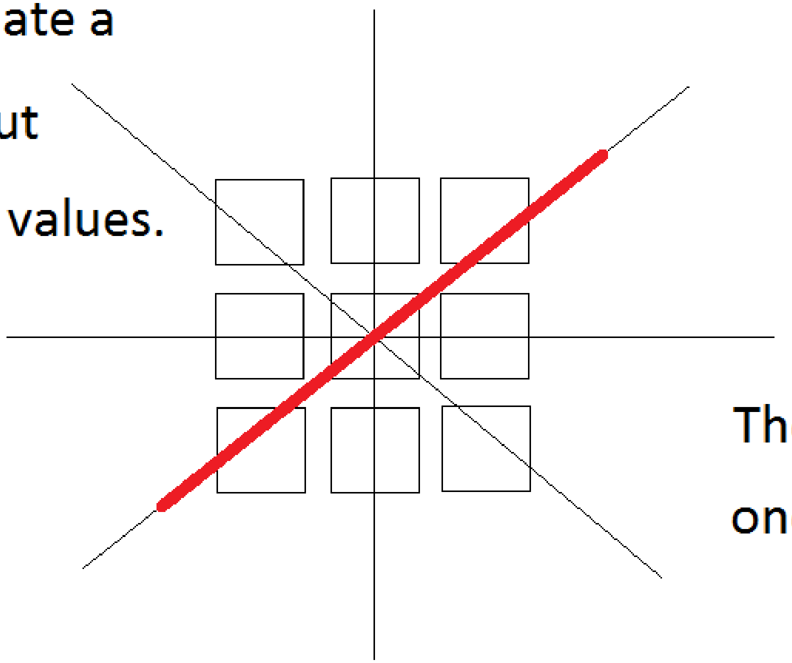
Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**

So, we need to isolate a

triple, and ask about

the three adjacent values.



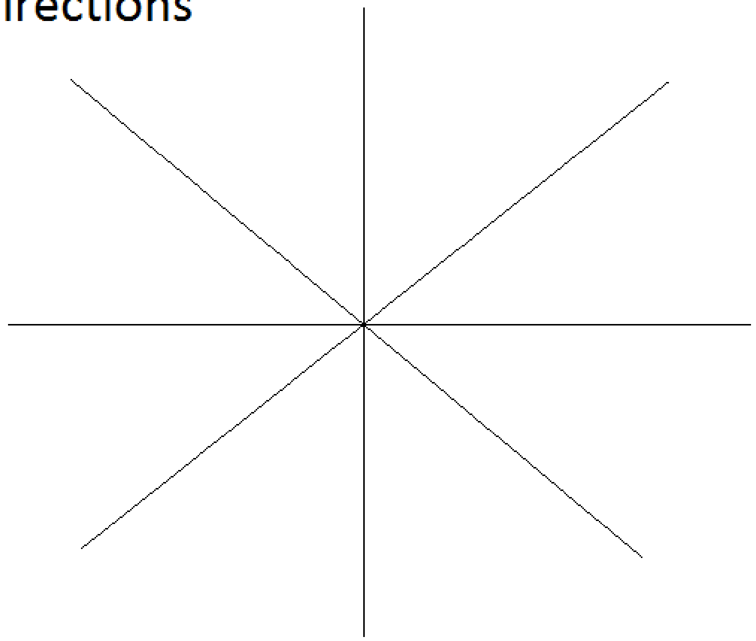
There are 4 such directions,

one triple per direction.

Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**

So, consider the 4 directions



For each direction, we will ask whether the center mag value exceeds the mag value of neighbor on one side, and neighbor on other side; if yes, mark as peak.

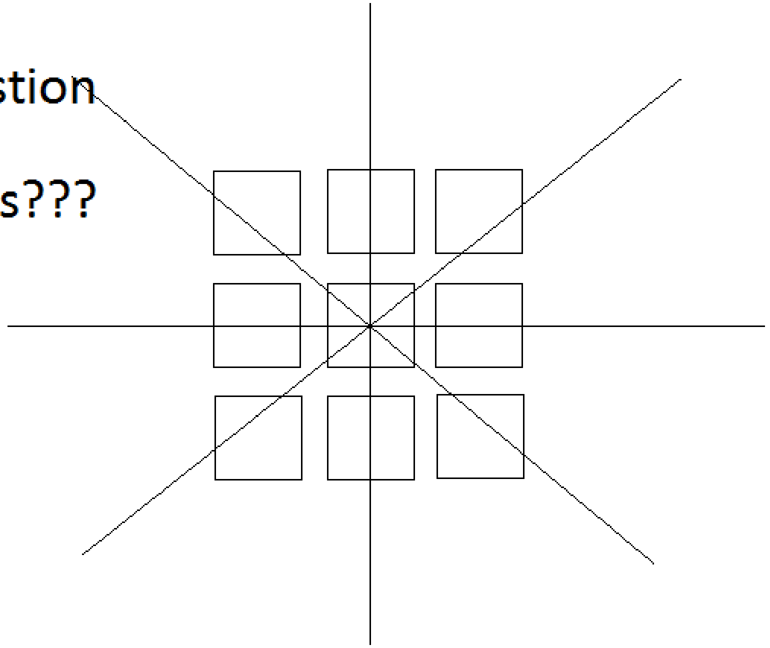
Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**

For a center pixel, do we

ask the maxing question

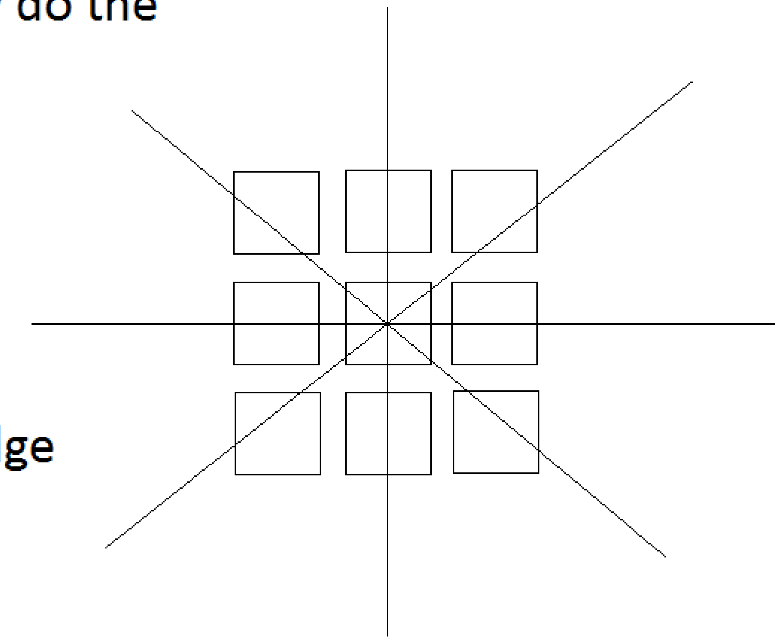
for all four directions???



Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**

No, we should only do the maxing test for the direction that is dictated by the direction of the edge



We always want to max-test in the direction perpendicular to the edge, i.e., across the edge. This means in the direction of the gradient vector.

Canny Edge detector algorithm

4. **Apply “Non-maximum Suppression”**

So, given these 4 possible

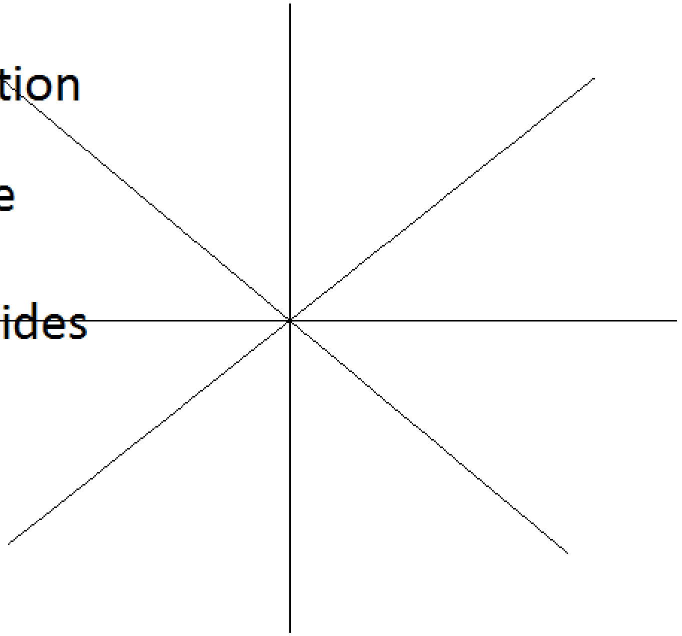
directions, each direction

will be called up, if the

gradient vector coincides

with the max-test

direction.



Canny Edge detector algorithm

4. **Apply “Non-maximum Suppression”**

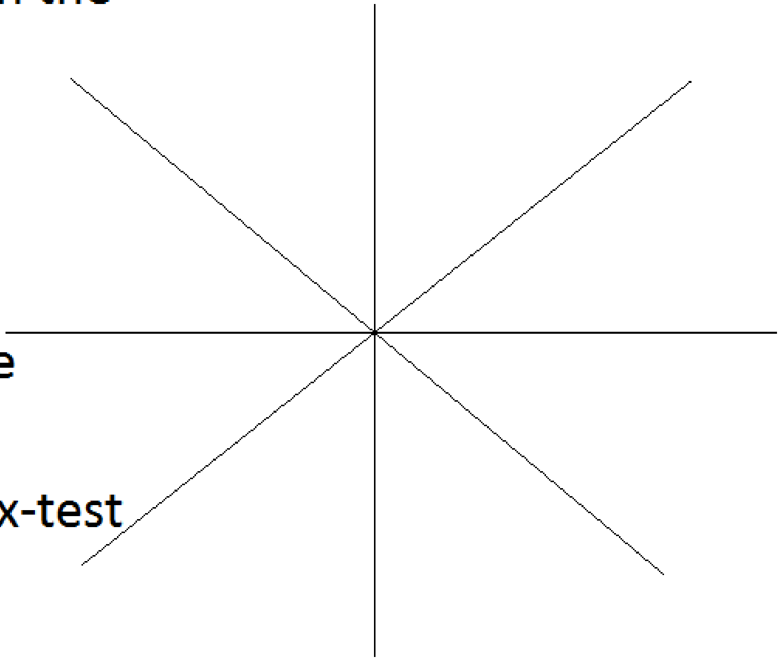
Hence, we ascertain the

gradient vector's

direction, and then

depending on it, we

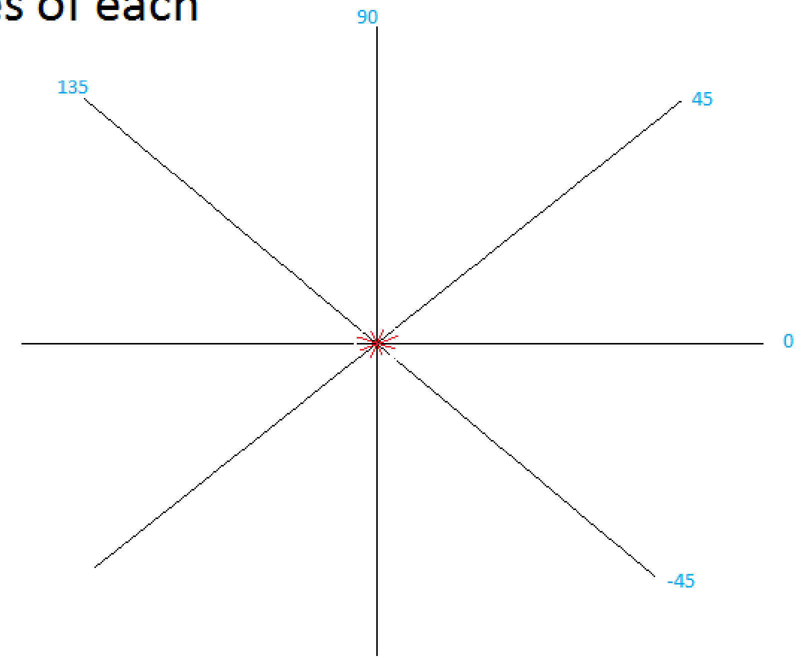
proceed to the max-test



Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**

We know the angles of each direction.



Canny Edge detector algorithm

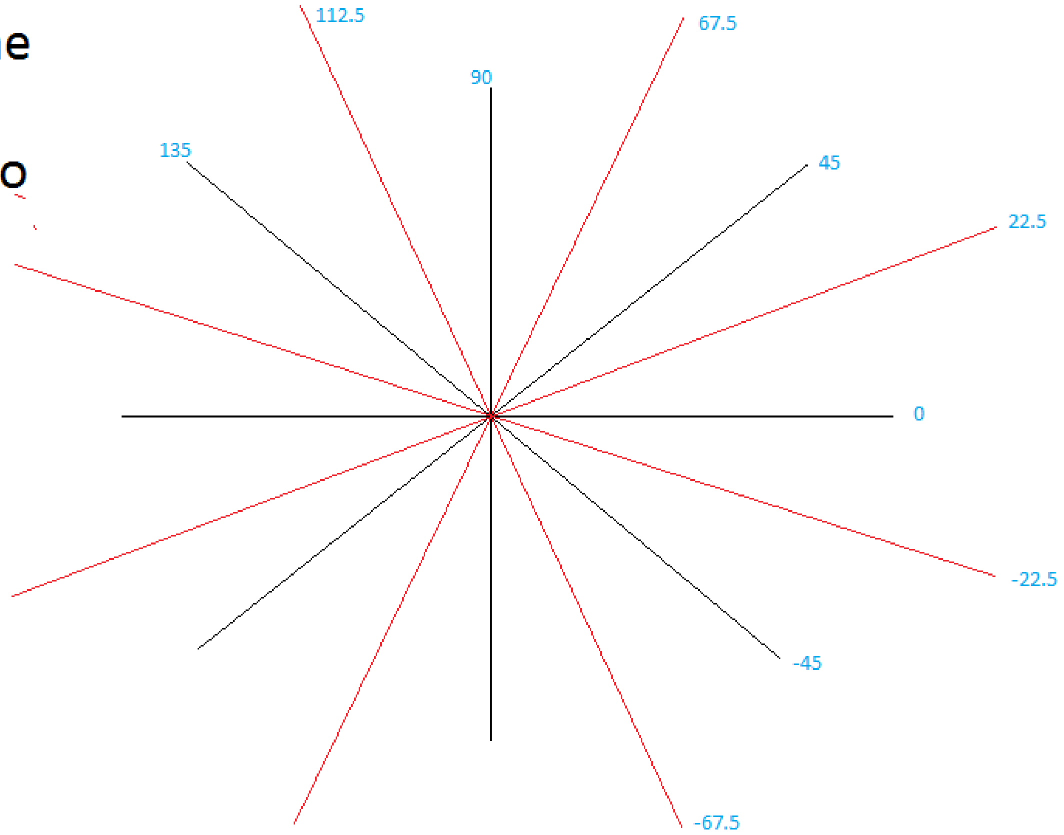
4. **Apply “Non-maximum Suppression”**

We break up the

angle space into

4 cones of

directions



Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**

For example, if it is found

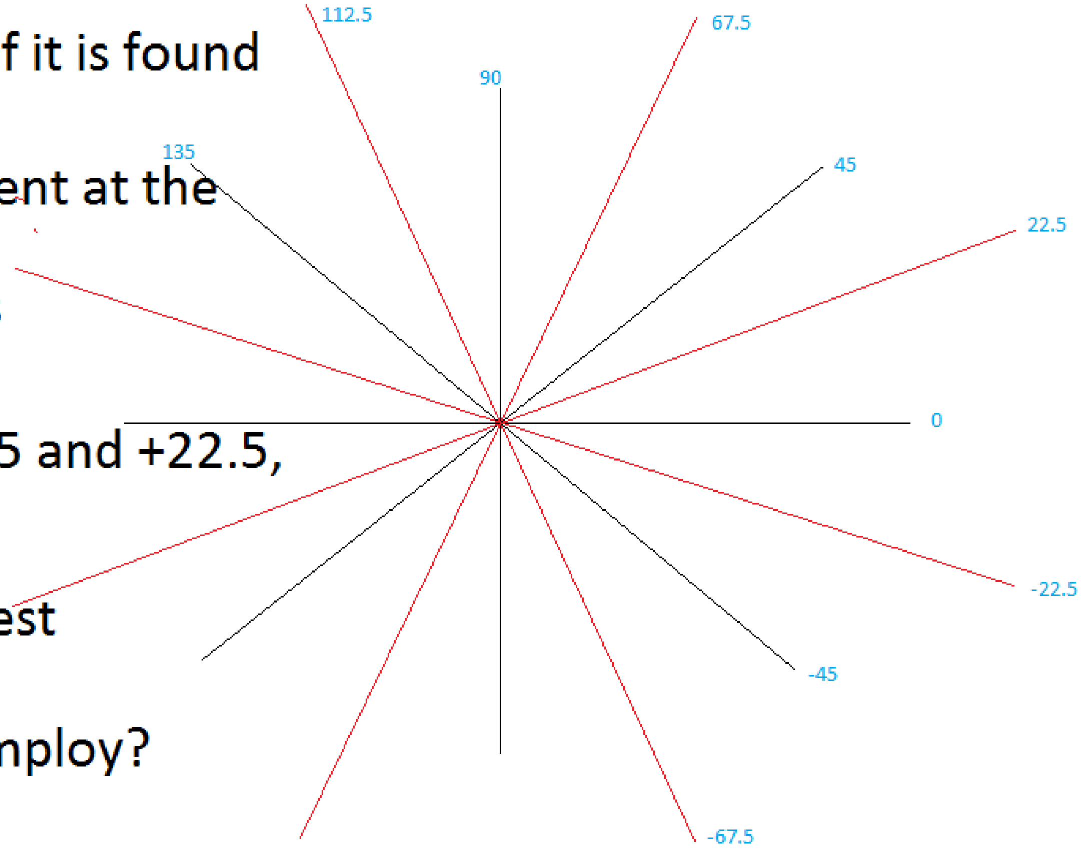
that the gradient at the

center pixel is

between -22.5 and +22.5,

Which max-test

should we employ?



Canny Edge detector algorithm

4. **Apply “Non-maximum Suppression”**

So, put = sign in

For each pixel (i.e., double-for loop)

Get pixel's gradient direction , Dir

If - 22.5 < Dir <=22.5

Employ Horizontal Max-Test

else if +22.5 < Dir <=+67.5

And, remove Vertical cone test, use "otherwise"

Employ Test involving SW and NE pixels

else if -67.5 < Dir <=-22.5

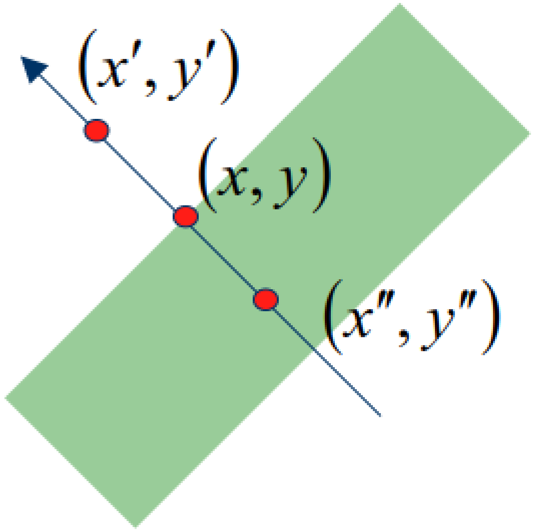
Employ Test involving SE and NW pixels

else Employ Vertical test

Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**

- Suppress the pixels in $|\nabla S|$ which are not local maximum



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

$\mathbf{x}'$ and $\mathbf{x}''$ are the neighbors of $\mathbf{x}$ along normal direction to an edge

Canny Edge detector algorithm

4. **Apply "Non-maximum Suppression"**

$$|\Delta S| = \sqrt{S_x^2 + S_y^2}$$

M

For visualization

$M \geq Threshold = 25$

Comparison

Gradient Thresholding

With non-maximal suppression

Canny Edge detector algorithm

1. Smooth image with Gaussian filter
2. Compute derivative of filtered image
3. Find magnitude and orientation of gradient
4. Apply “Non-maximum Suppression”
5. **Apply “Hysteresis Threshold”**

Hysteresis Thresholding

- Edges tend to be continuous
- Still threshold the gradient
- Use a lower threshold if a neighboring point is an edge
- The "Canny Edge Detector" uses all of these heuristics

Canny Edge detector algorithm

5. Apply "Hysteresis Threshold"

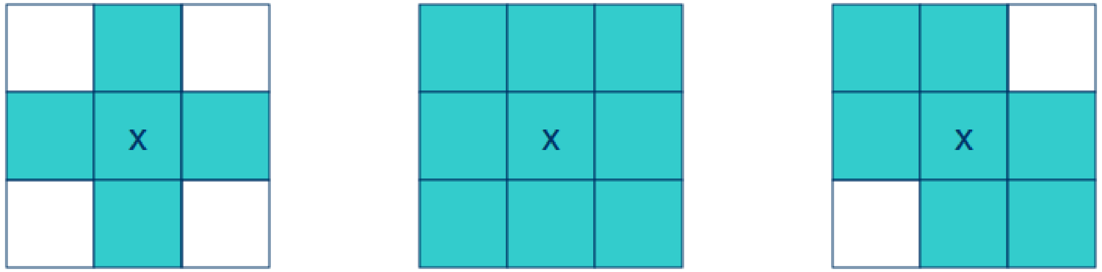
- If the gradient at a pixel is
  - above "**High**", declare it as an '**edge pixel**'
  - below "**Low**", declare it as a "**non-edge-pixel**"
  - **between** "low" and "high"
    - Consider its neighbors iteratively then declare it an "edge pixel" if it is **connected** to an 'edge pixel' **directly** or via pixels **between** "low" and "high".

Canny Edge detector algorithm

5. Apply “Hysteresis Threshold”

- If the gradient at a pixel is
  - above “**High**”, declare it as an ‘**edge pixel**’
  - below “**Low**”, declare it as a “**non-edge-pixel**”
  - **between** “low” and “high”
    - Consider its neighbors iteratively then declare it an “edge pixel” if it is **connected** to an ‘edge pixel’ **directly** or via pixels **between** “low” and “high”.

- Connectedness



4 connected | 8 connected | 6 connected

Canny Edge detector algorithm

5. Apply "Hysteresis Threshold"

- Scan the image from left to right, top-bottom.
  - The gradient magnitude at a pixel is above a high threshold declare that as an edge point
  - Then recursively consider the *neighbors* of this pixel.
    - If the gradient magnitude is above the low threshold declare that as an edge pixel.

Canny Edge detector algorithm

5. Apply “Hysteresis Threshold”

M

regular

$M \geq 25$

Hysteresis

$High = 35$

$Low = 15$

Canny Edge detector algorithm

Before non-max suppression

After non-max suppression

Canny Edge detector algorithm

Threshold at low/high levels to get weak/strong edge pixels

- Do connected components, starting from strong edge pixels

Final Canny Edge

Canny Edge detector algorithm

Final Canny Edge

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

Questions?