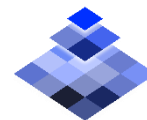


# Edge detection



Yoni Chechik

[www.AlisMath.com](http://www.AlisMath.com)



# References

- <http://szeliski.org/Book/>
- <http://www.cs.cornell.edu/courses/cs5670/2019sp/lectures/lectures.html>
- <http://www.cs.cmu.edu/~16385/>

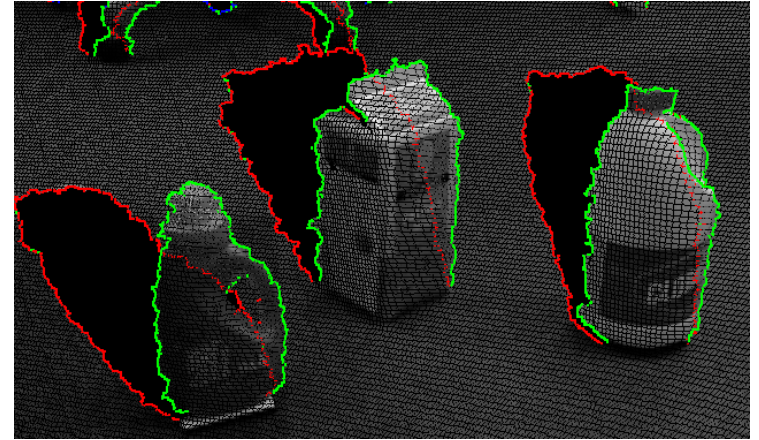
# Contents

- **Intro to edges**
- Basic edge image
- Edge thinning
  - LoG
  - NMS
- Edge mask
- Canny edge detector
- Other edge related topics
  - Frequency representation
  - Unsharp filter

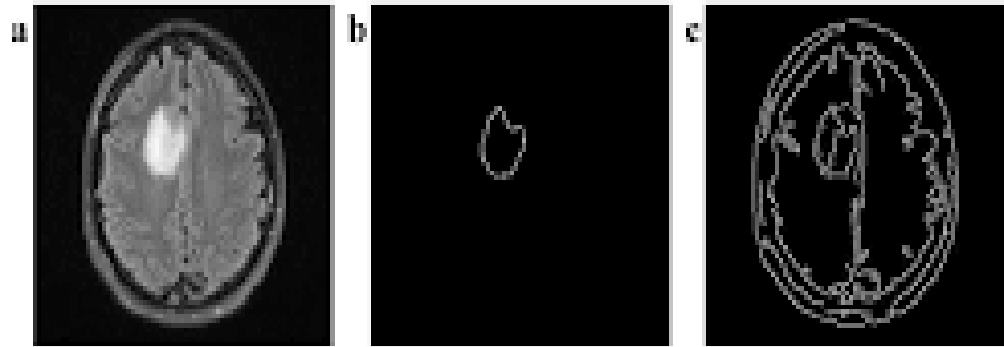
# Some motivation



Art  
(Instagram filters)



Robotics  
(scene understanding)



Medicine  
(tumor detection)



Autonomous vehicles  
(license plate detection)

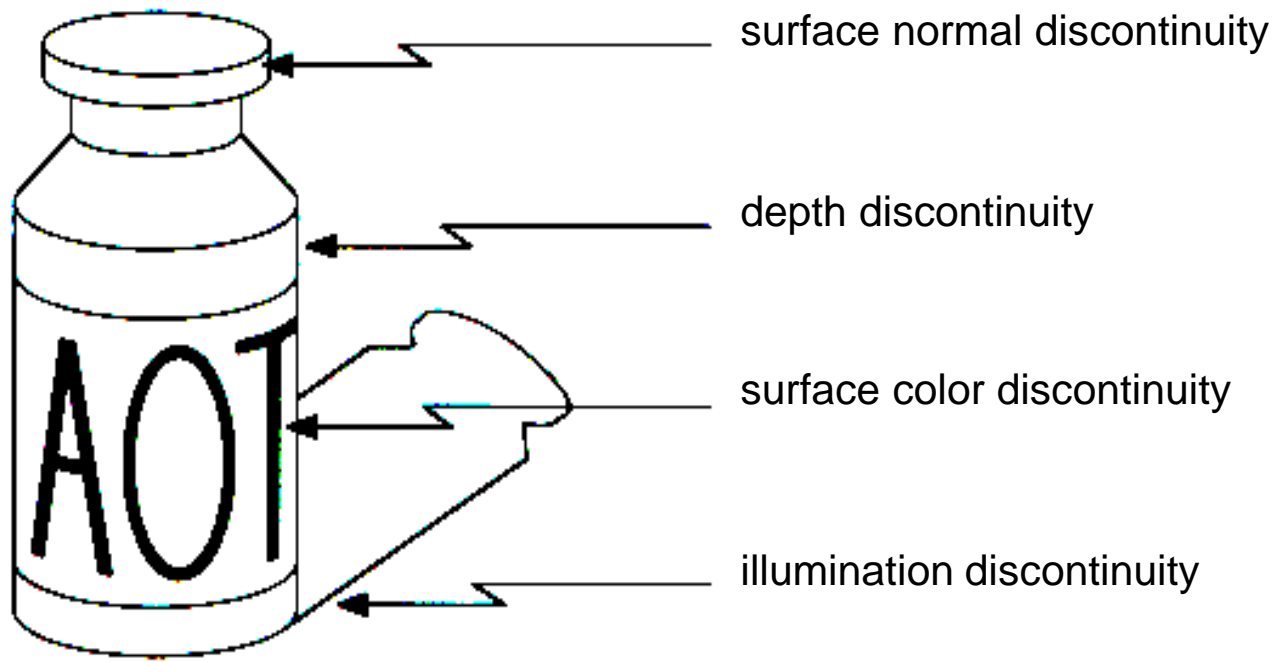
# Why edges?

- Representation of objects can be done without full image representation- more compact.
- Edges are salient features (salient- “most noticeable or important”).



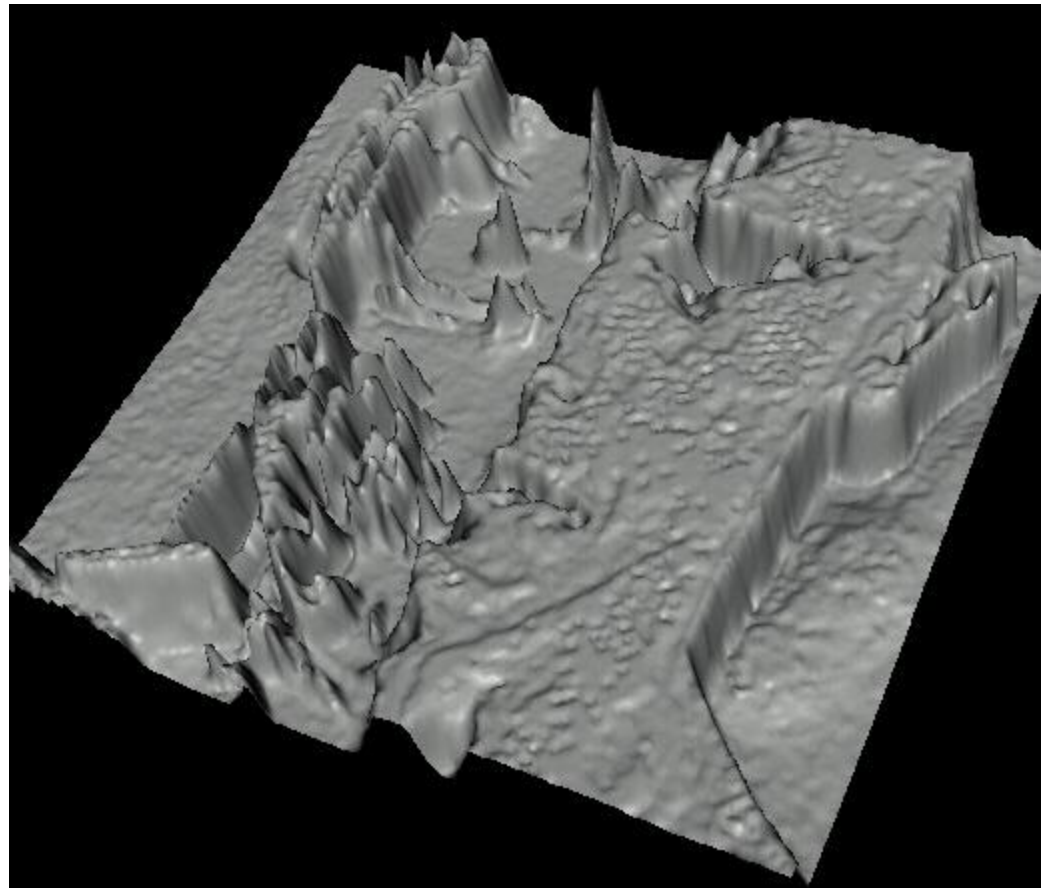
# What are edges?

- “The outside limit of an object, area, or surface; a place or part farthest away from the center of something.”
- Edges can be caused from many reasons in images:



# Representation in images

- Rapid changes in colors.
- Looks like steep edges if represented as a surface:



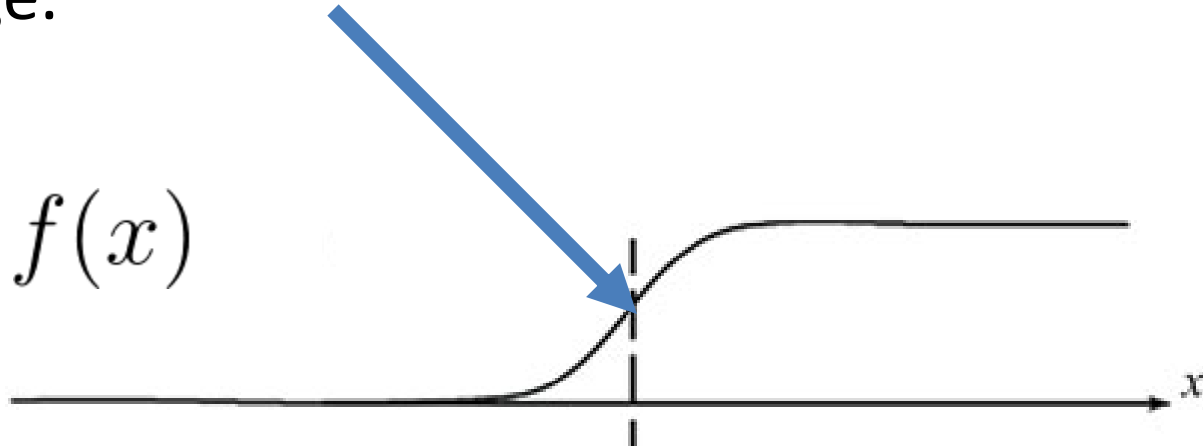
# Contents

- Intro to edges
- **Basic edge image**
- Edge thinning
  - LoG
  - NMS
- Edge mask
- Canny edge detector
- Other edge related topics
  - Frequency representation
  - Unsharp filter



# How to find edge image?

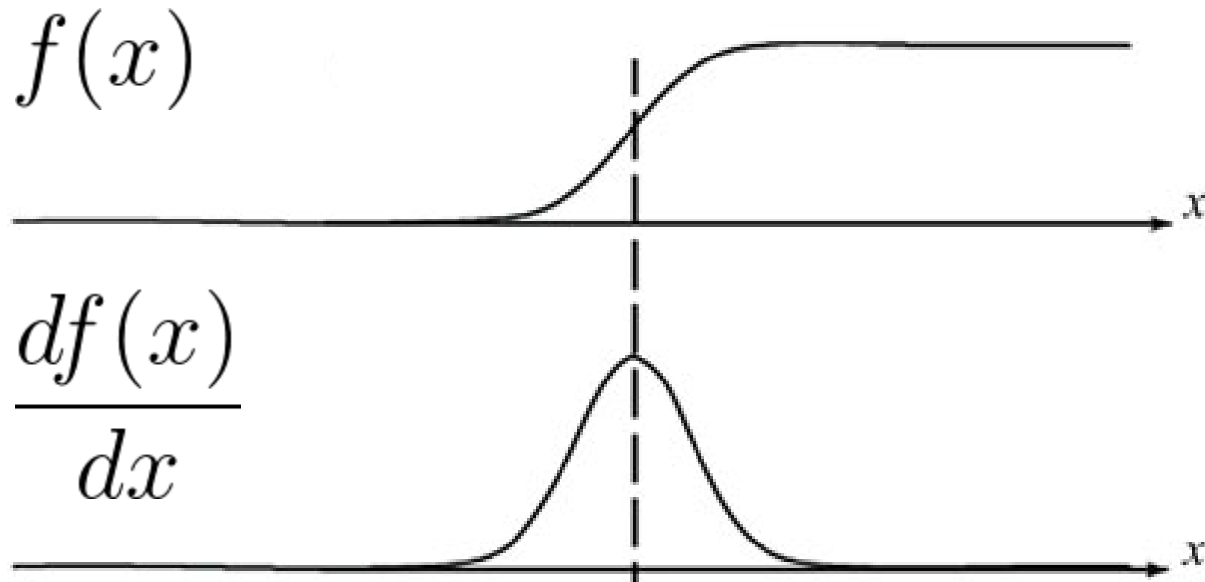
- Wanted result: image of a binary mask of where there is an edge.



- How to do so?

# First order derivative

- Derivative of an edge:



- Finding maximum points in the derivative of an image is a possible way to find edges!

# Deriving the derivative

- Definition of derivative in continuous functions:

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

# Deriving the derivative

- Definition of derivative in continuous functions:

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

- In discrete space we can set  $h = 1$ :

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

# Deriving the derivative

- Definition of derivative in continuous functions:

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

- In discrete space we can set  $h = 1$ :

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

- And in 2D space (derivative along x axis):

$$f'_x[x, y] = f[x+1, y] - f[x, y]$$

# 1<sup>st</sup> derivative filter

$$f'_x[x, y] = f[x + 1, y] - f[x, y]$$

- We can mimic this derivative as a convolution operator:

$$f'_x = f * \begin{array}{|c|c|} \hline +1 & -1 \\ \hline \end{array}$$

- Note 1: when a kernel size is even in some dimension, the center of the kernel needs to be specified (above the center is  $-1$ ).
- Note 2: remember that in the convolution operation **the kernel is flipped in both directions.**

# Symmetric 1<sup>st</sup> derivative

- A more common approach is using the symmetric 1<sup>st</sup> derivative:

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

- How it's written in a discrete form?

# Symmetric 1<sup>st</sup> derivative

- A more common approach is using the symmetric 1<sup>st</sup> derivative:

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

- Which translates to this kernel:

$$f'_x = f * \frac{1}{2} \begin{array}{|c|c|c|} \hline +1 & 0 & -1 \\ \hline \end{array}$$

- We'll use the kernel above without the  $\frac{1}{2}$  constant, since we only care about the ratio between gradients.



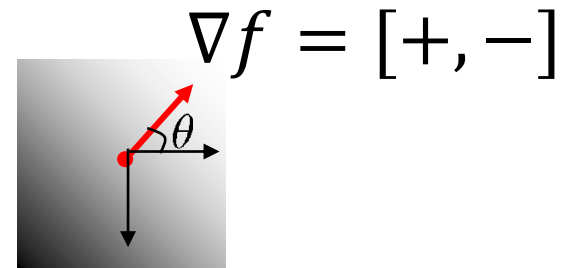
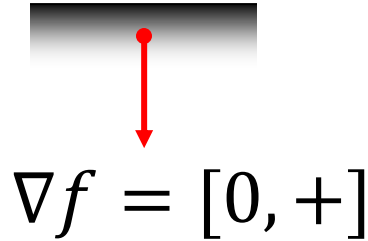
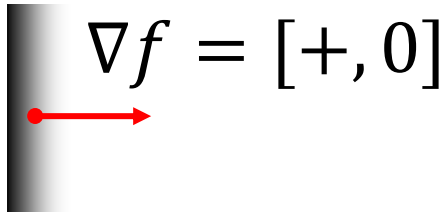
# Y direction

$$f'_y = f * \begin{array}{|c|} \hline +1 \\ \hline 0 \\ \hline -1 \\ \hline \end{array}$$

- The convolution kernel above is true for python/matlab/opencv image axis convention, where the positive y direction is down.
  - Let's use this convention in the below derivation.

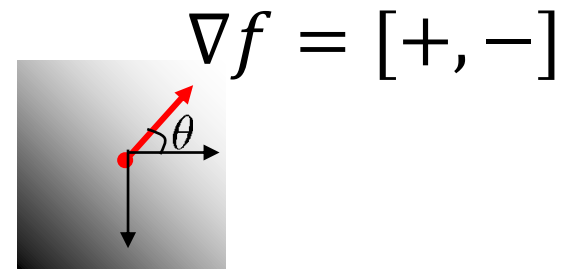
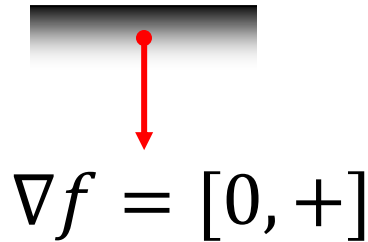
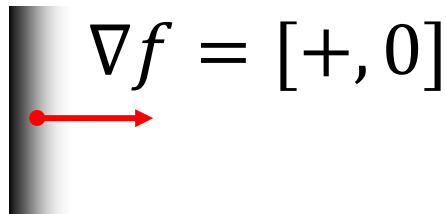
# Image gradient

- The **gradient** of an image:  $\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$
- The gradient points in the direction of most rapid increase in intensity:



# Image gradient

- The **gradient** of an image:  $\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$
- The gradient points in the direction of most rapid increase in intensity:



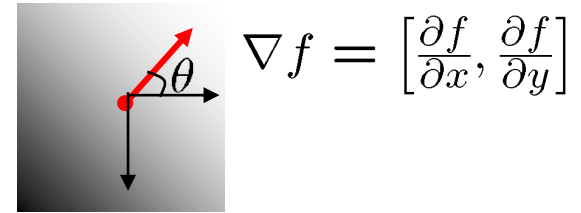
- The edge **strength** is given by the gradient magnitude:

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

# Gradient direction

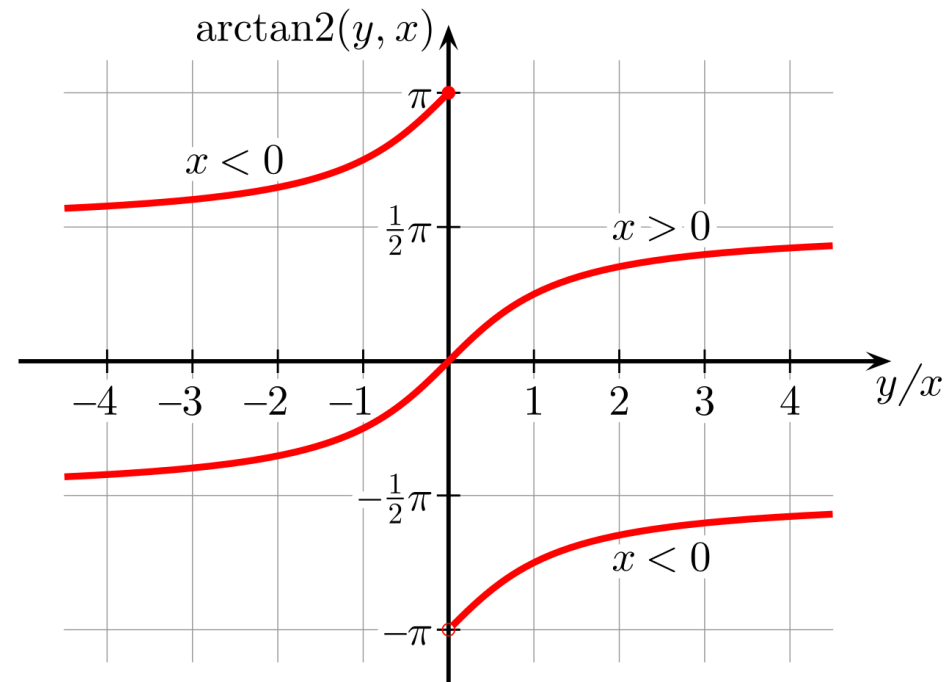
- The gradient direction is given by:

$$\theta = \text{atan2}(-f'_y, f'_x)$$



- $\theta \in (-\pi, \pi]$
- $-f'_y$  because of the inversed y direction.
- unlike regular  $\tilde{\theta} = \arctan(\frac{y}{x})$  in which  $-\frac{\pi}{2} < \tilde{\theta} < \frac{\pi}{2}$

$$\text{atan2}(y, x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0, \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0, \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0, \\ +\frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0, \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0, \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

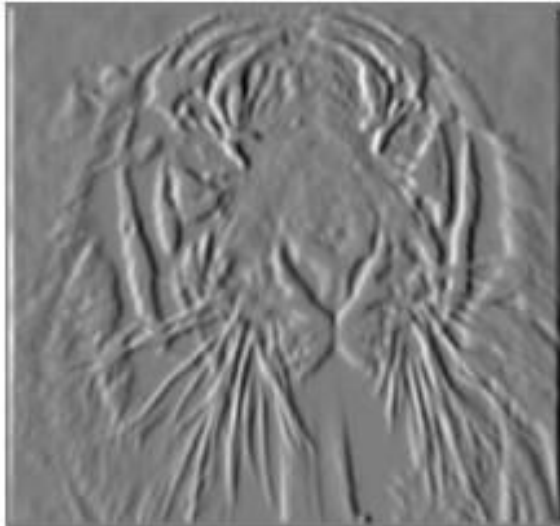


# Image gradient example

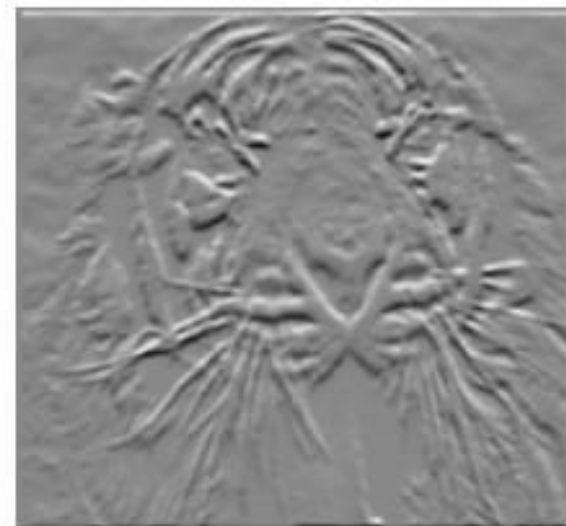
Original image



$\|\nabla f\|$



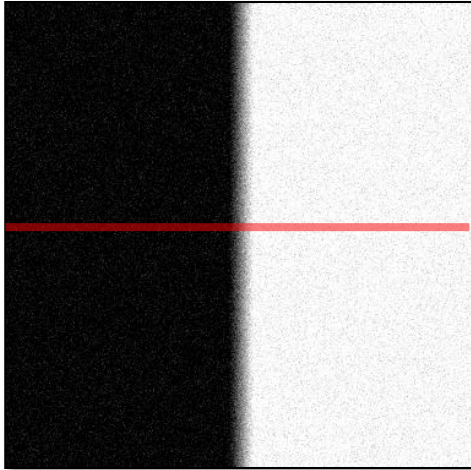
$\nabla f_x$



$\nabla f_y$

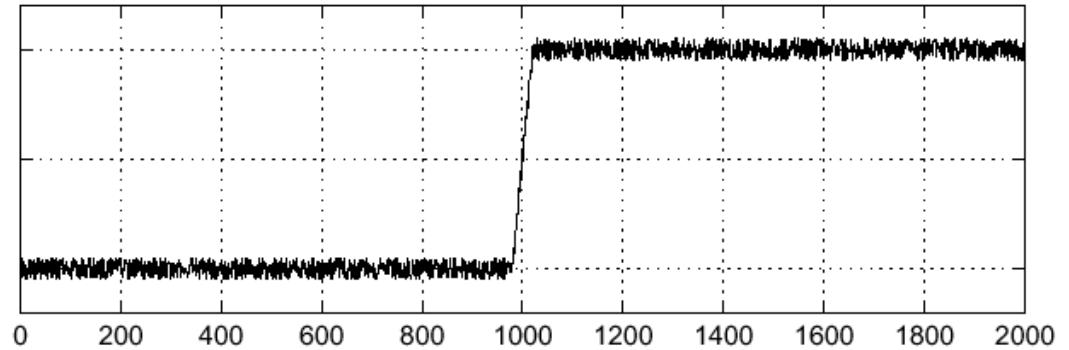
# Noise effects

- How to find maximum of derivative in noisy environment?

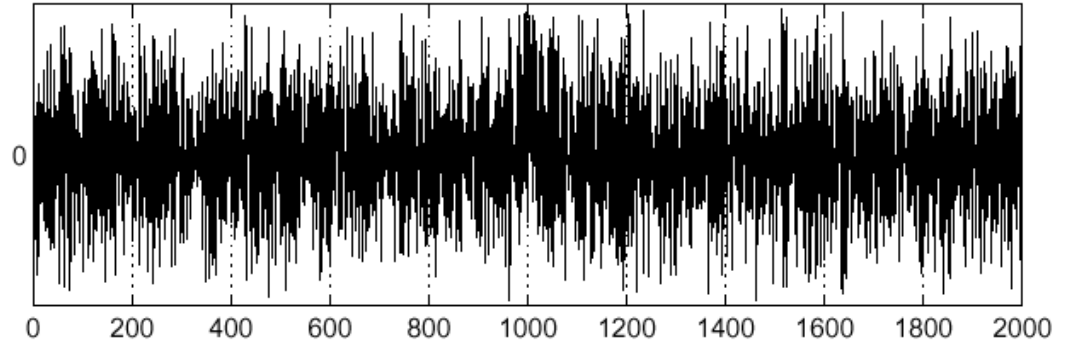


Noisy input image

$$f(x)$$

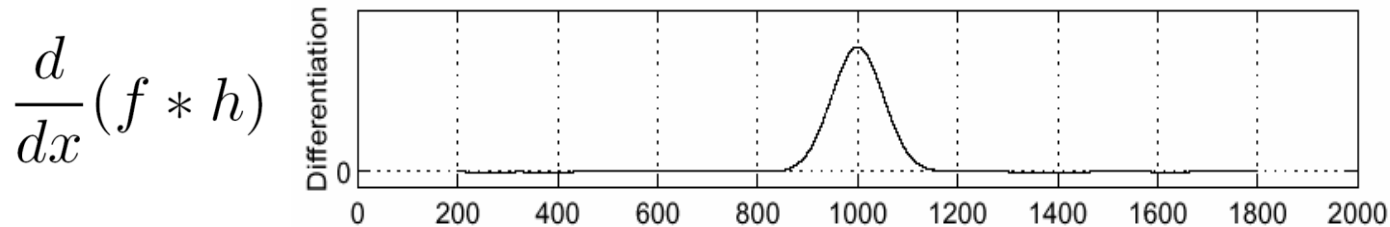
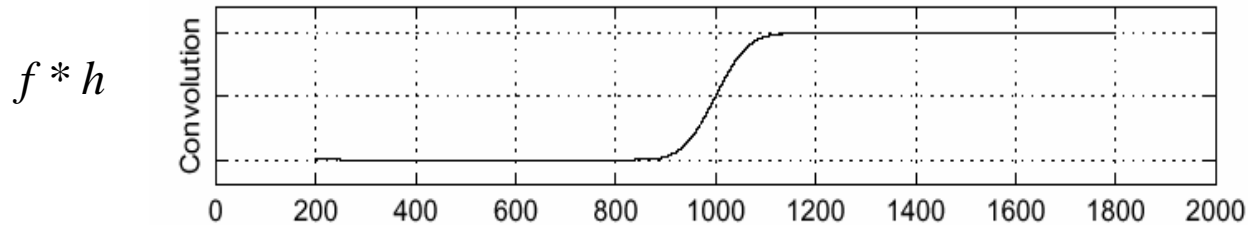
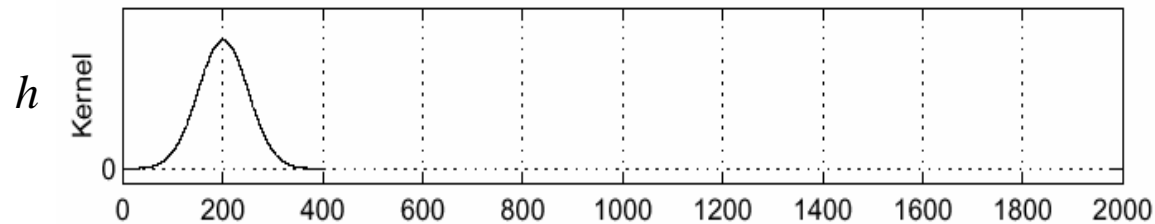
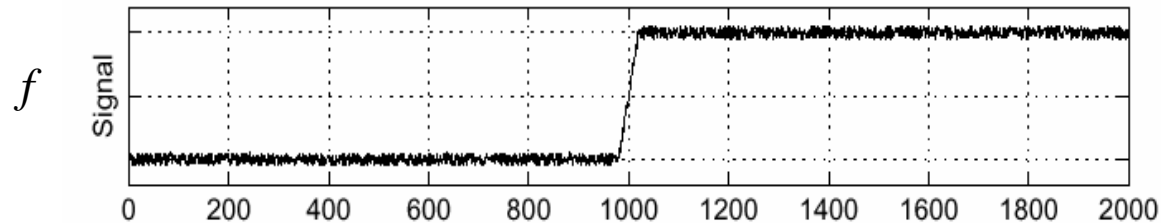


$$\frac{d}{dx} f(x)$$



# Solution 1: smoothing the noise

Sigma = 50



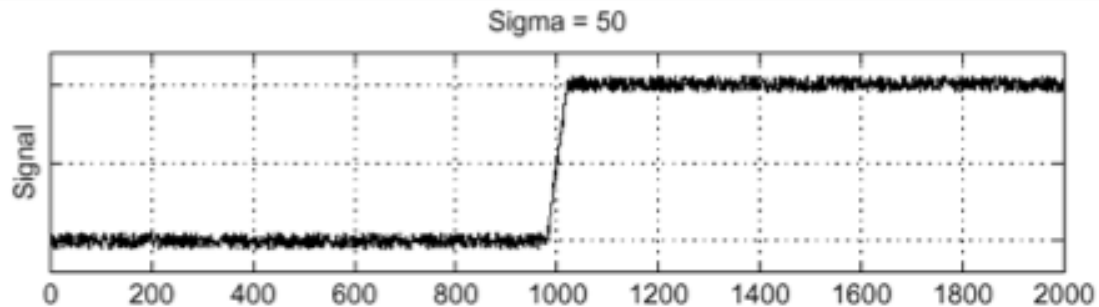
- Search for the maximum in the smooth image!

# Gaussian derivative kernel

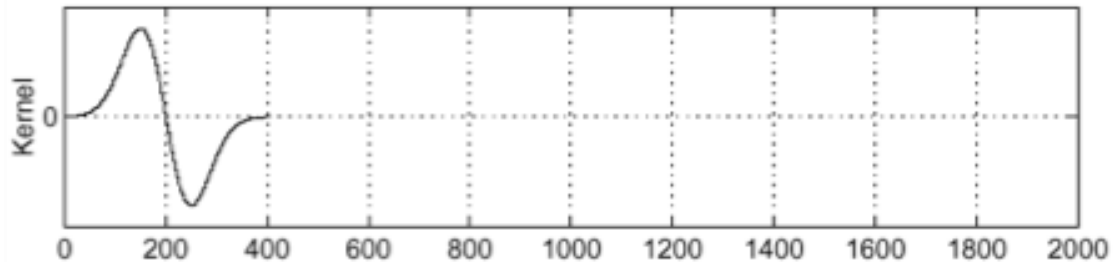
- Using this convolution trick:

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

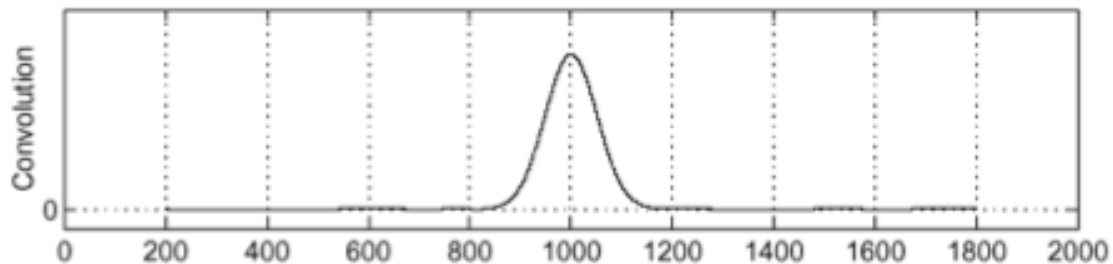
input



derivative of  
Gaussian

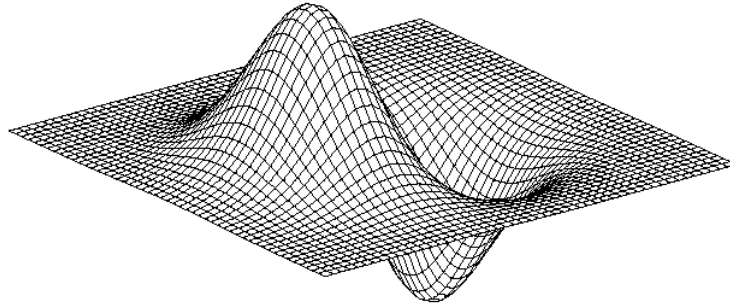


output (same  
as before)



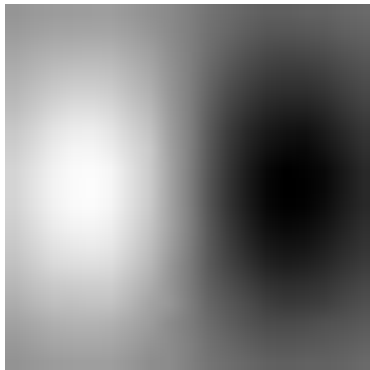


# Gaussian derivative kernel 2D

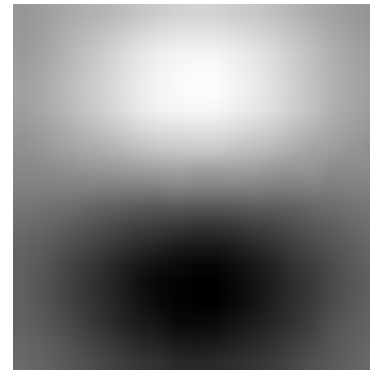


Derivative of Gaussian

x-direction

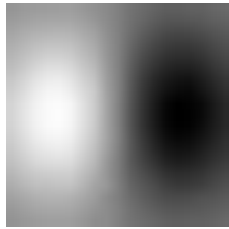


y-direction



# Sobel filter

- Common approximation of derivative of Gaussian

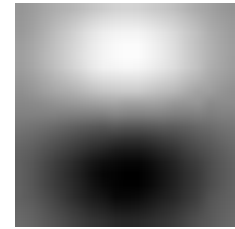


1	0	-1
2	0	-2
1	0	-1

$s_x$

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

$s_y$



- Can also be thought of as a kernel with higher weighting for closer neighbors.

## Solution 2: Prewitt filter

$$f'_x = f * \begin{array}{|c|c|c|} \hline +1 & 0 & -1 \\ \hline +1 & 0 & -1 \\ \hline +1 & 0 & -1 \\ \hline \end{array}$$

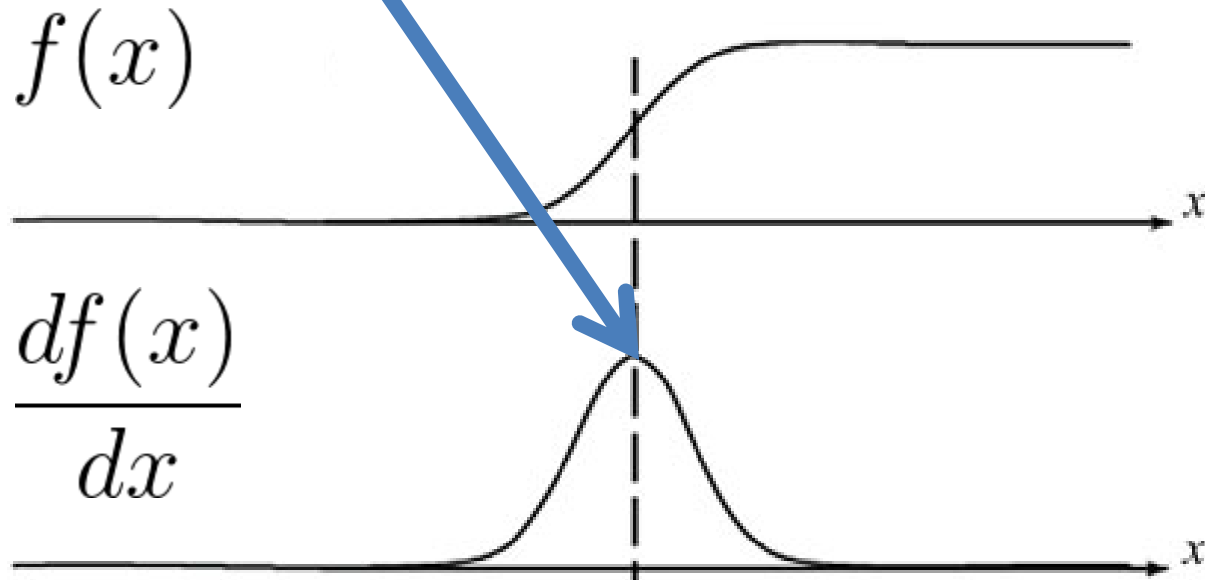
- Like Sobel but with different weighting for the neighbors.
- **In practice-** Non definitive superiority between Sobel and Prewitt.

# Contents

- Intro to edges
- Basic edge image
- Edge thinning
  - **LoG**
  - NMS
- Edge mask
- Canny edge detector
- Other edge related topics
  - Frequency representation
  - Unsharp filter

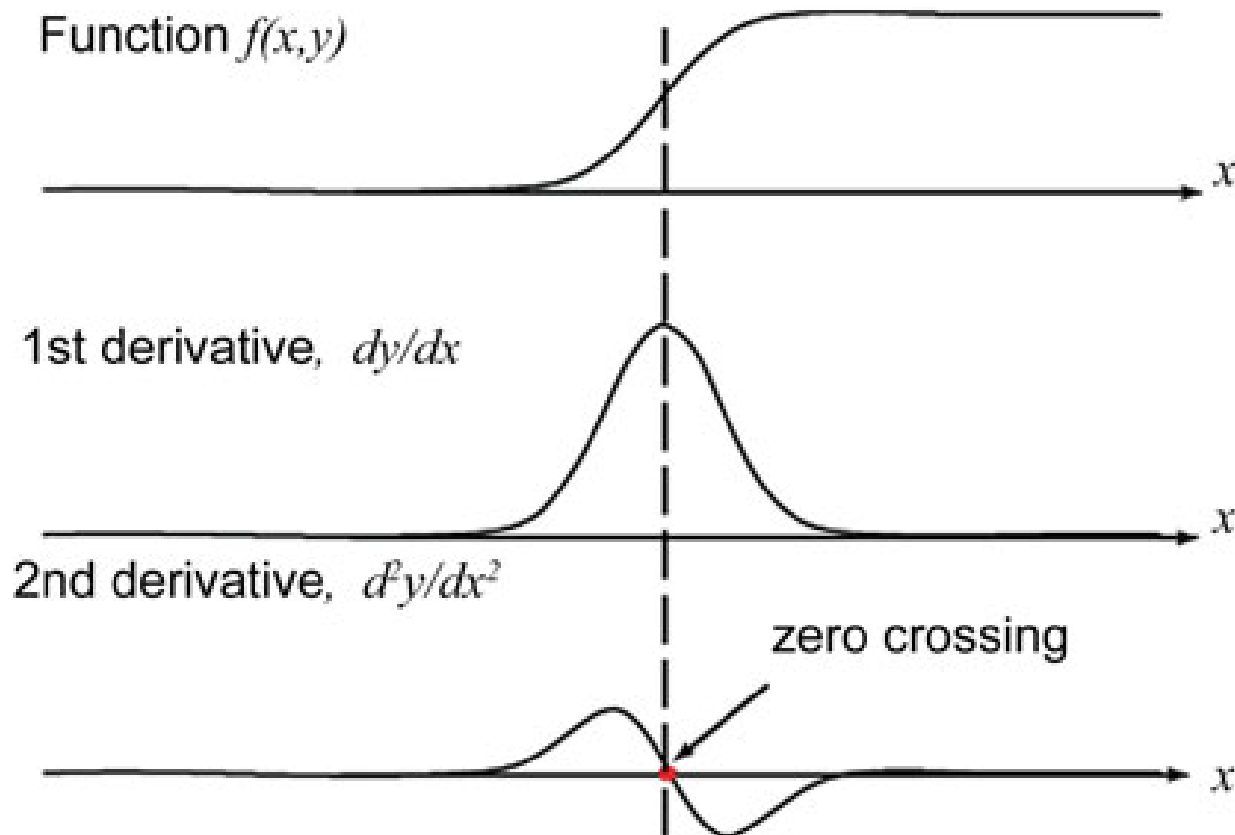
# Edge thinning

- I have the edge filter result, but I want only one pixel to represent the edge in a binary mask.
- How do I find this?



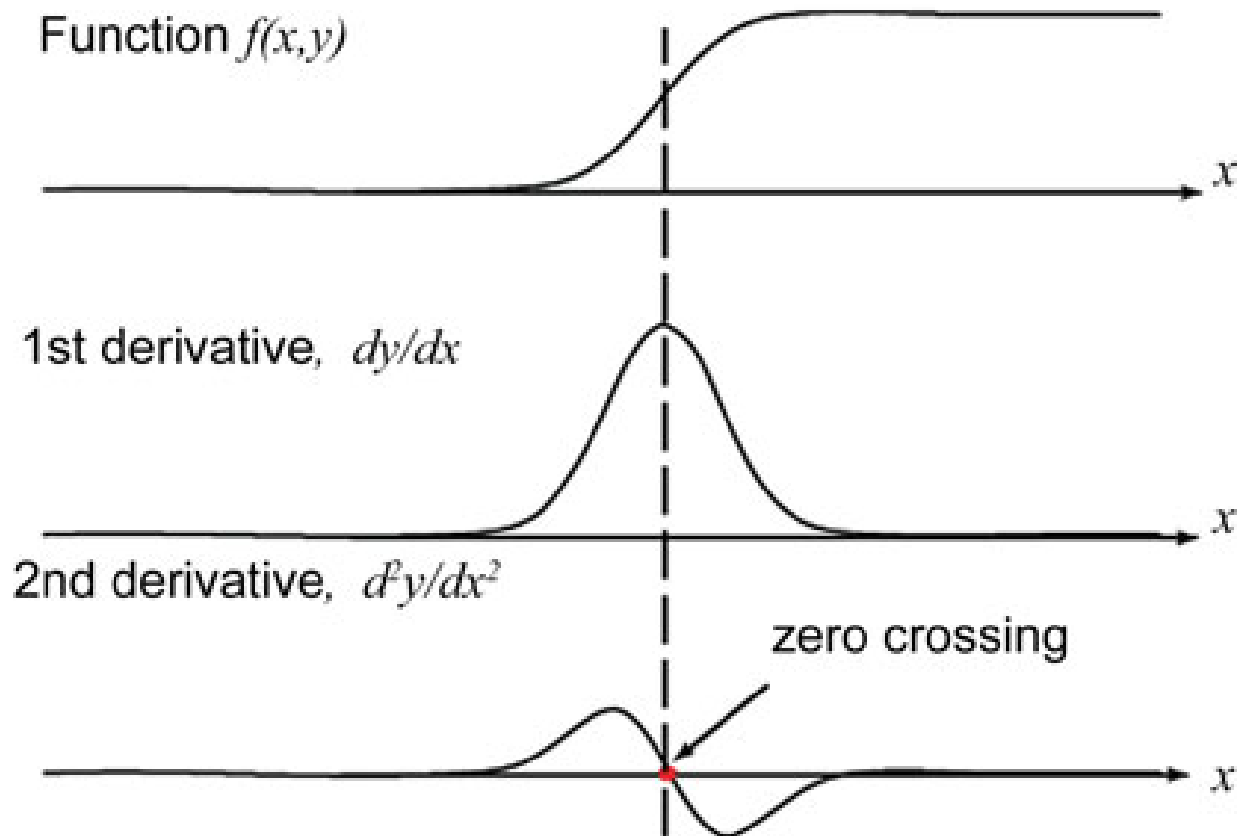
# Naïve approach: 2<sup>nd</sup> derivative

- Let's try to find the zero crossing of the 2<sup>nd</sup> derivative.
- Only single zero crossing- should produce thinner edge



# Naïve approach: 2<sup>nd</sup> derivative

- **In practice:** this approach is very susceptible to noise!



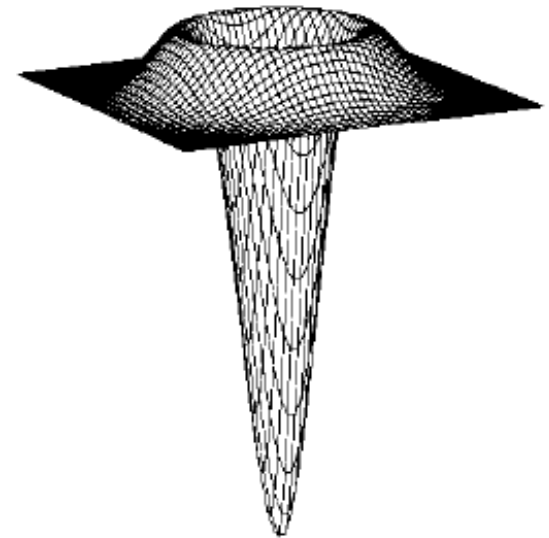
# A better approach: LoG

- Let's take the 2<sup>nd</sup> derivative of the Gaussian (Laplacian of Gaussian: LoG) kernel so smoothing will help with noise reduction:

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f = \nabla \cdot \left[ \frac{df}{dx}, \frac{df}{dy} \right] = \frac{d^2 f}{dx^2} + \frac{d^2 f}{dy^2}$$

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

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



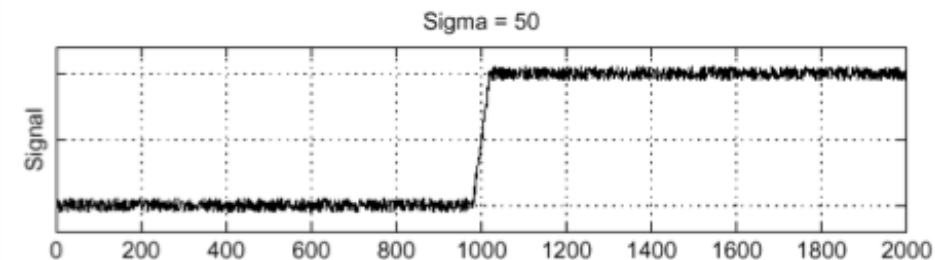
Laplacian of Gaussian



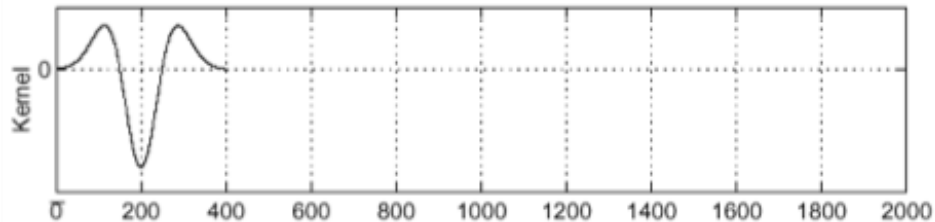
# Find edge in noisy signal: LoG

- Input is noisy step signal; output is zero crossing at the step.

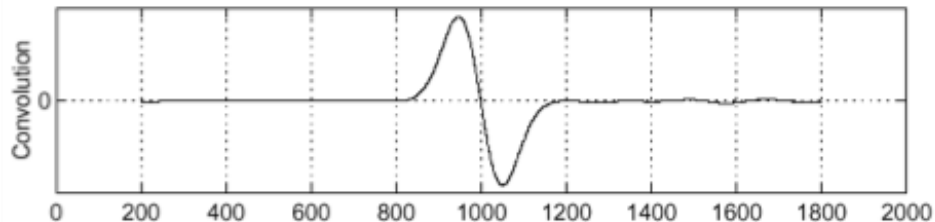
input



Laplacian of  
Gaussian



output



# LoG quantization

- Can be filter of different sizes:

– 3X3:

0	1	0
1	-4	1
0	1	0

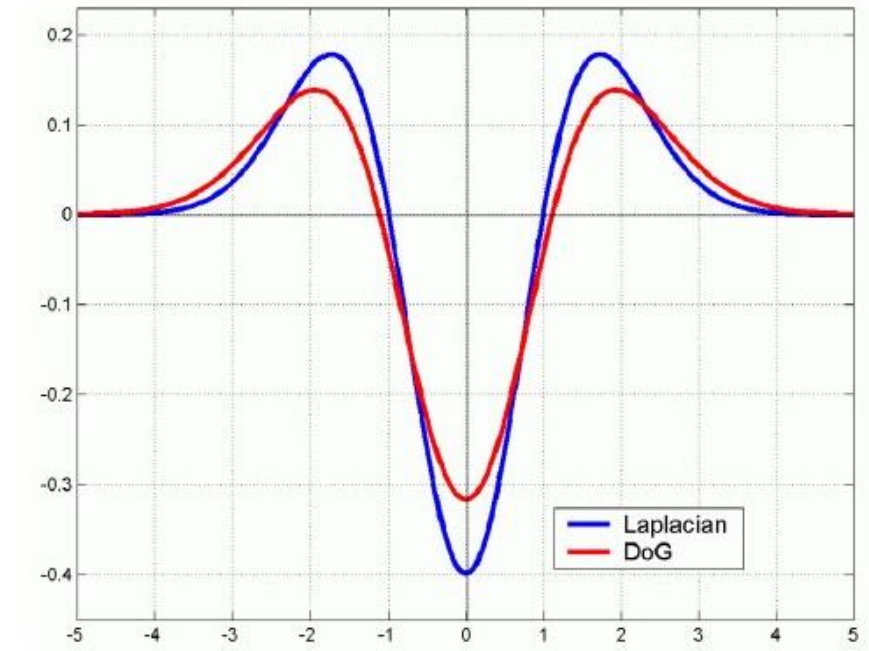
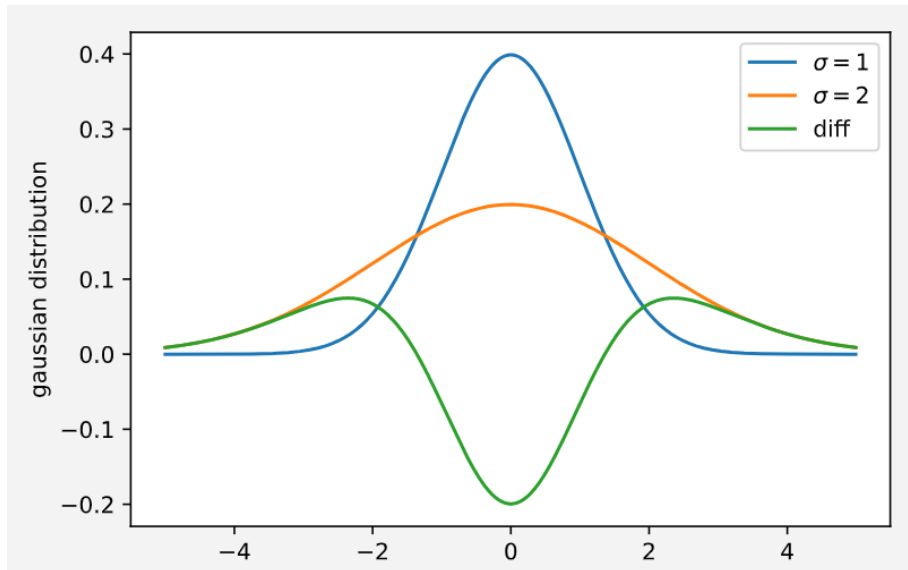
1	1	1
1	-8	1
1	1	1

– 9X9:

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

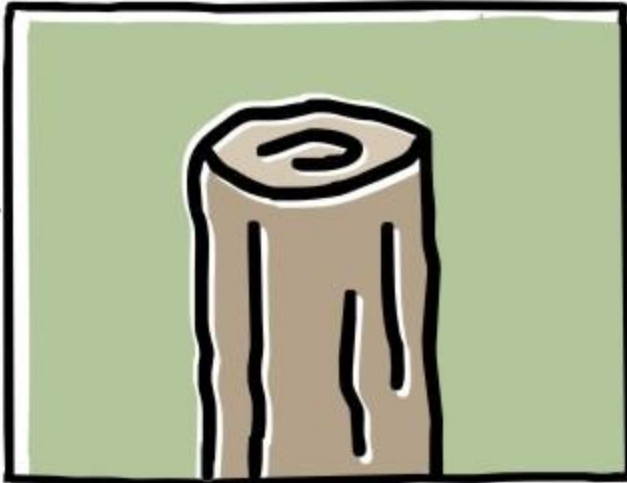
# DoG

- Can also use difference of Gaussians (DoG) to mimic LoG.
- Why do we want to do this? Faster computationally (explained here: <https://dsp.stackexchange.com/a/37675>)



# log vs. dog

Dist. by Universal Uclick



likes to be outside  
found near the fireplace  
gives warmth and comfort  
plays dead  
bark  
doesn't have a tail

© John Atkinson, Wrong Hands



likes to be outside  
found near the fireplace  
gives warmth and comfort  
plays dead  
bark  
can't be made into shelves

# Example: LoG



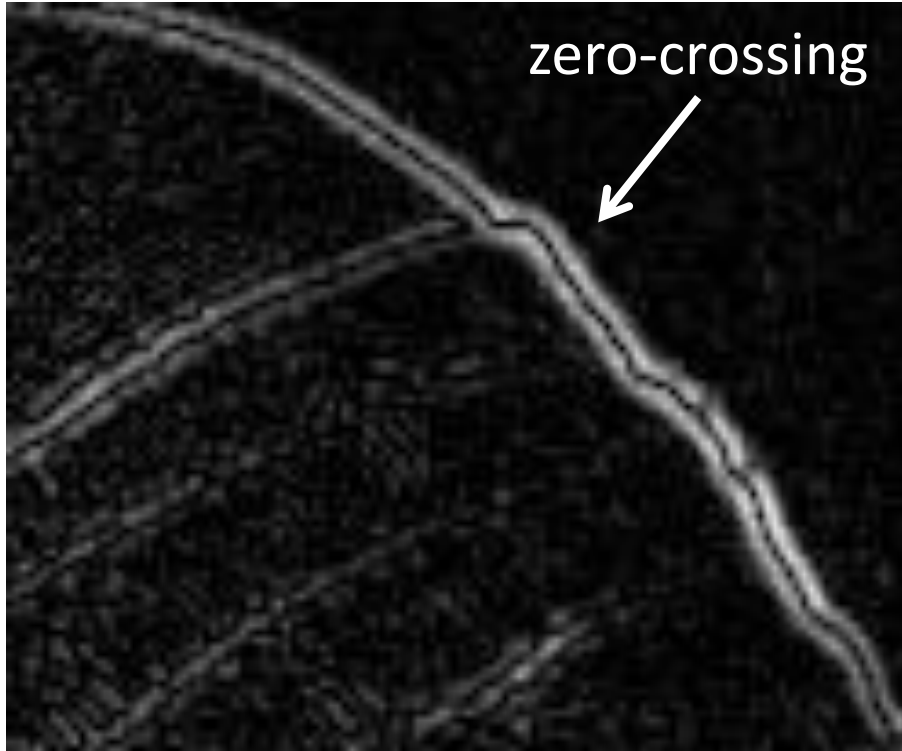
$|LoG(x)|$



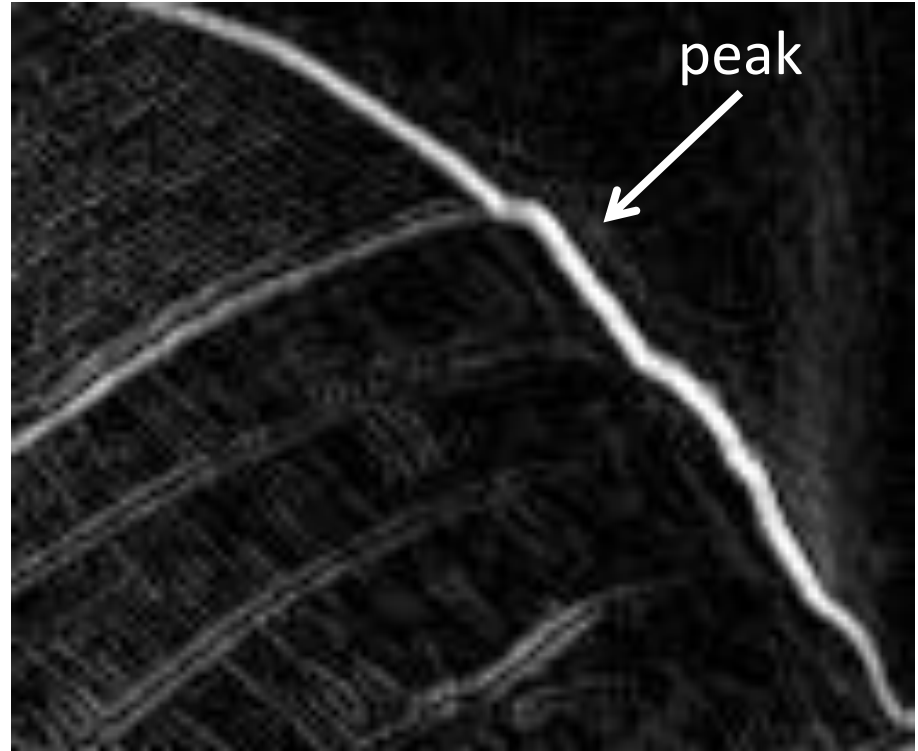
Absolute magnitude 1<sup>st</sup> derivative  
gaussian filter

# Example: LoG

- Note: both images are after absolute value.



$|LoG(x)|$



Absolute magnitude 1<sup>st</sup> derivative  
gaussian filter

# Zero crossing

- The new problem arising from the LoG filter is: how to mark the zero crossings?
- Answer: no easy algorithm to detect zero crossings.
  - E.g.: planes with minor noise will also produce zero crossing artifacts.

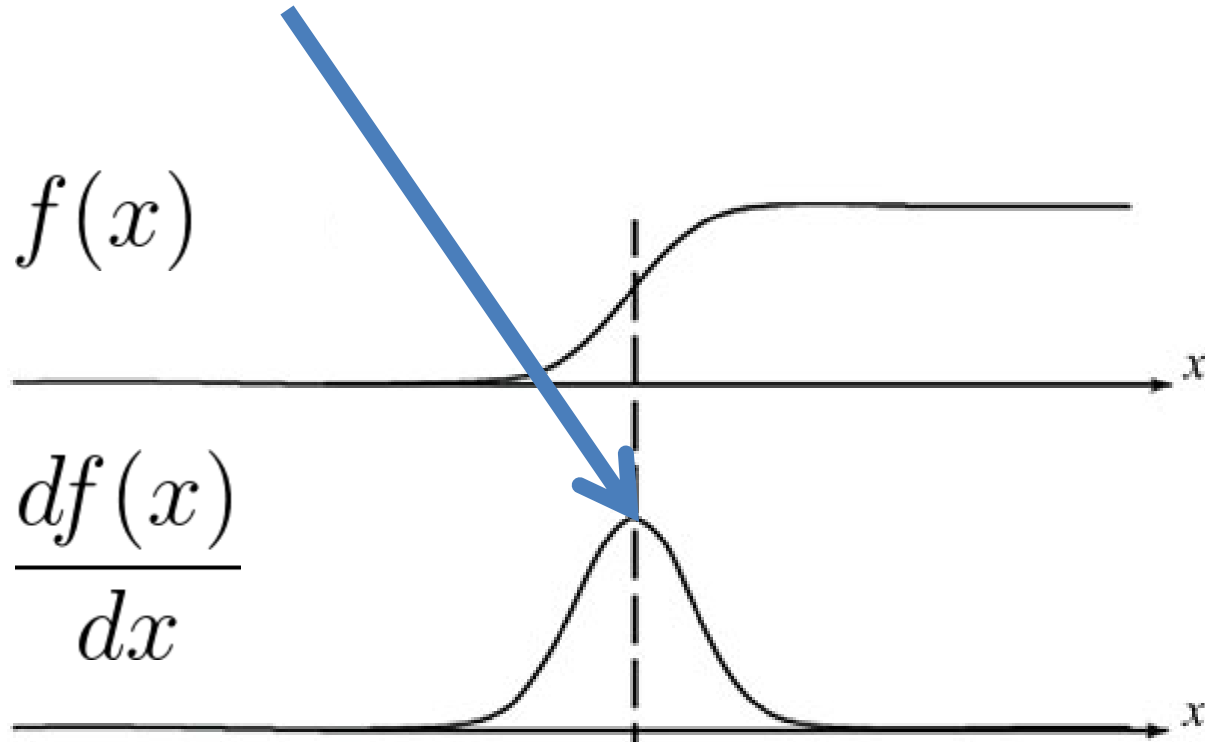
# Contents

- Intro to edges
- Basic edge image
- Edge thinning
  - LoG
  - **NMS**
- Edge mask
- Canny edge detector
- Other edge related topics
  - Frequency representation
  - Unsharp filter



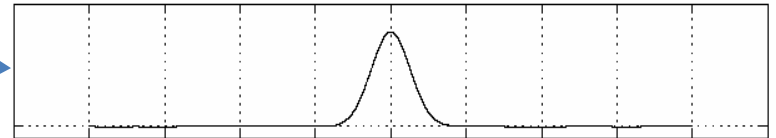
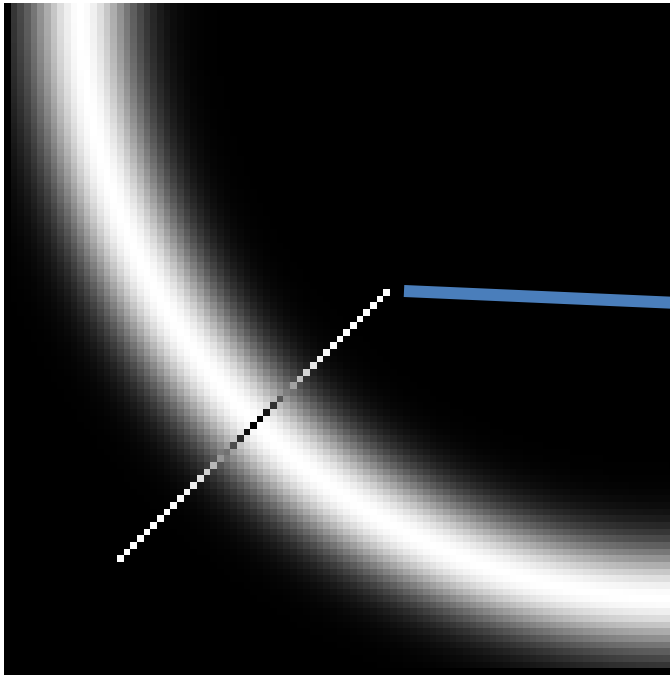
# Edge thinning

- I have the edge filter result, but I want only one pixel to represent the edge in a binary mask.
- How do I find this?



# Non maximum suppression

- NMS
- Find the gradient magnitude + orientation of each pixel and search on this 1D line for maximum point.



# NMS algorithm

**get** image gradient magnitude + orientation using 1D 3X3 gradient filter (e.g.: Sobel).

**for** each pixel  $p_0$ :

**Quantize**  $\angle p_0$  to one of four possibilities:  
[0°, 45°, 90°, 135°].

**In** 3X3 neighborhood of  $p_0$ , find two neighbors in quantized gradient orientation  $\{p_1, p_2\}$ .

**If**  $\|p_0\| < \|p_1\|$  or  $\|p_0\| < \|p_2\|$ :

$\|p_0\| \leftarrow 0$

# NMS results

Before NMS

After NMS



# Contents

- Intro to edges
- Basic edge image
- Edge thinning
  - LoG
  - NMS
- **Edge mask**
- Canny edge detector
- Other edge related topics
  - Frequency representation
  - Unsharp filter

# Edge mask

- How do we transform this integer image to a binary mask of where there is/ isn't an edge?

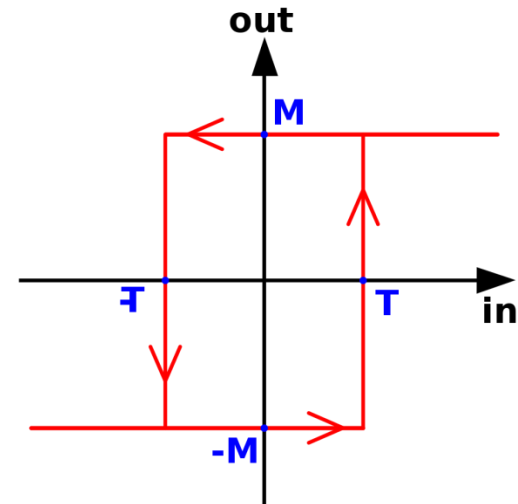
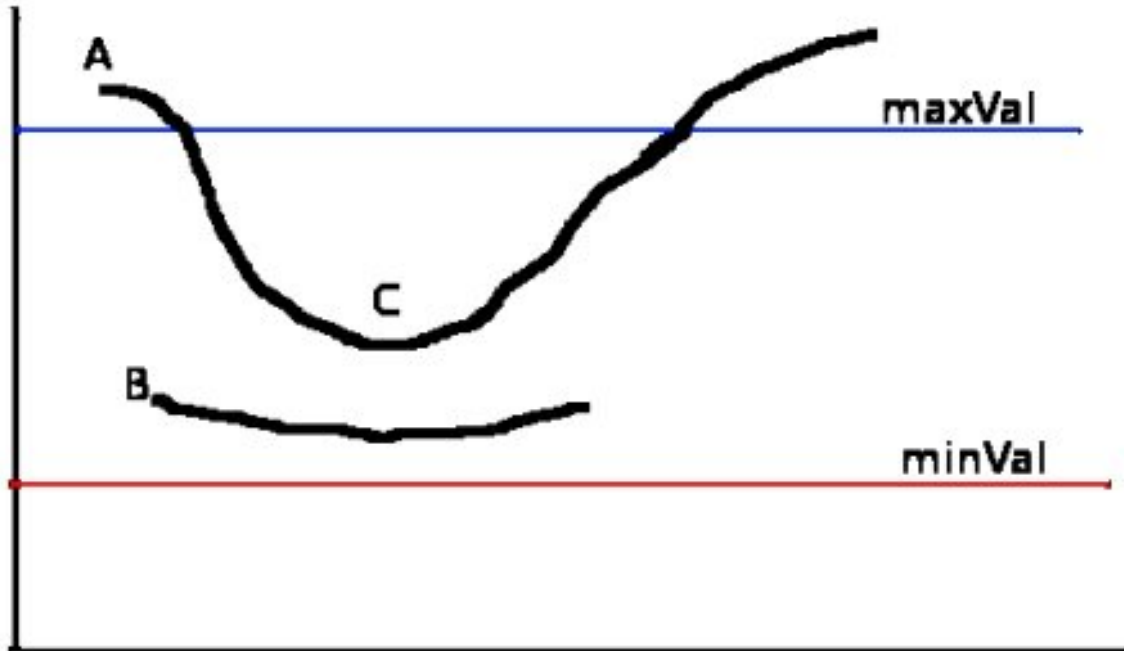


# First try: single threshold edge mask

- Mask == binary image.
- Possible 1<sup>st</sup> solution- thresholding:
  - Choose an TH edge value, above which the pixel mask is 1, 0 otherwise.
  - The value can be a constant or percentile of the maximum edge value exists in the image.
  - Low TH: will get extra edges, but also input noise.
  - High TH: can miss lower valued edge pixels, less noise.
- How can we difference between low value edge pixels and noise?

# Hysteresis motivation

- Weak edges are usually neighbors of strong edges, while noise can be at any pixel.
  - Usually “neighbors” means 3X3 square of adjacent pixels.
- If we know that a neighbor of a weak edge is a strong edge, then **the weak edge is a strong edge!**



Schmitt trigger  
hysteresis example plot



# hysteresis

Choose two thresholds:  $\{TH_h, TH_l | TH_h > TH_l\}$

For each pixel  $p_i$ :

If  $p_i \geq TH_h$ :

$p_i \leftarrow 1$

elif  $TH_l \leq p_i < TH_h$ :

$p_i \leftarrow \text{weak\_edge\_pixel}$

Else: *//  $p_i < TH_l$*

$p_i \leftarrow 0$

While *weak\_edge\_pixels* that are neighbors of 1 exists:

for each *weak\_edge\_pixel*  $p_i$ :

If *weak\_edge\_pixel*  $p_i$  neighbor of 1:

*weak\_edge\_pixel*  $p_i \leftarrow 1$

All remaining *weak\_edge\_pixels*  $\leftarrow 0$

# Contents

- Intro to edges
- Basic edge image
- Edge thinning
  - LoG
  - NMS
- Edge mask
- **Canny edge detector**
- Other edge related topics
  - Frequency representation
  - Unsharp filter

# Canny edge detector

- Canny edge detector is one of the most known and used CV algorithms, still highly used even today (developed in 1986, cited 33000 times until 2019):

**Gaussian filter**

**Find image gradient magnitude and orientation**

**NMS**

**Hysteresis**

# Example output



# Important note: tradeoffs

- It's a common **misconception** in CV to think that one algorithm is **always** better than another.
- In CV, algorithms are highly dependent in the given environment in which they are executed. Each environment can vary in:
  - Noise.
  - Needed computation efficiency.
  - Overall problem variance.
  - Etc...

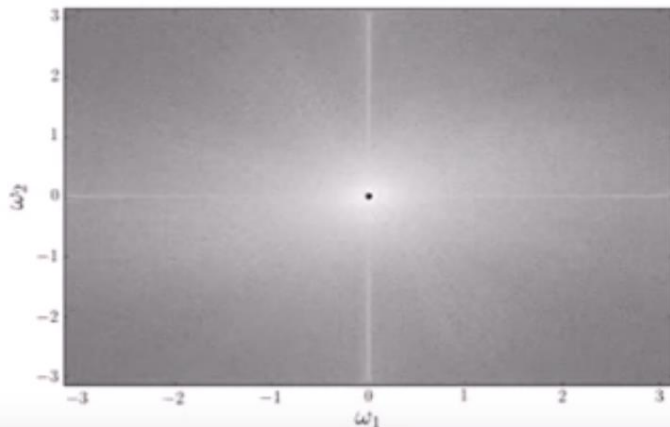
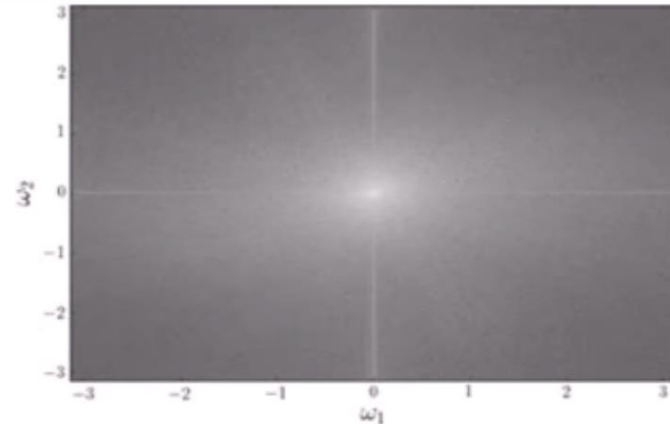
CV is the  
land Of  
tradeoffs

# Contents

- Intro to edges
- Basic edge image
- Edge thinning
  - LoG
  - NMS
- Edge mask
- Canny edge detector
- Other edge related topics
  - **Frequency representation**
  - Unsharp filter

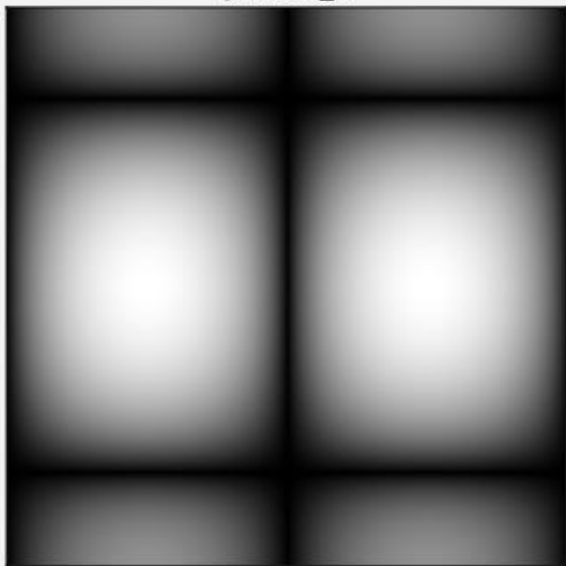
# HP filter

- Higher frequencies represents the edges of images.
- Removing the lower frequencies of an image will result in edge image!

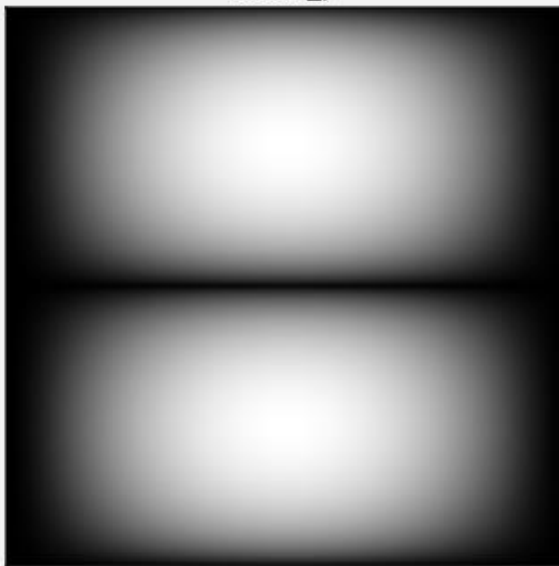


# Edge filters- frequency representation

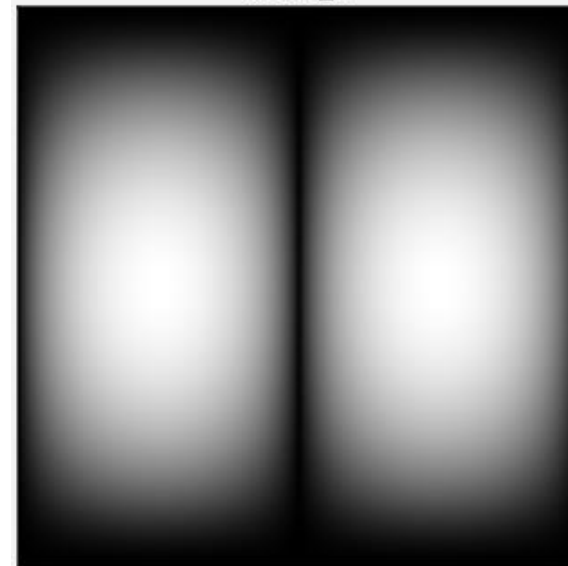
prewitt\_x



sobel\_y



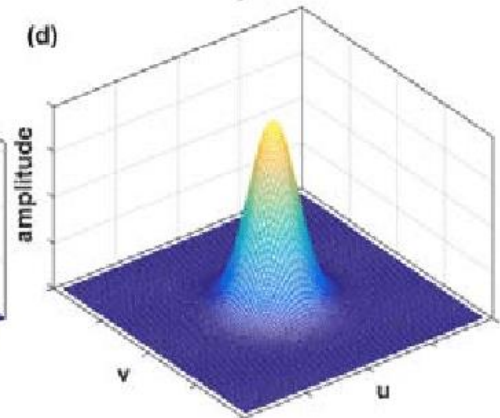
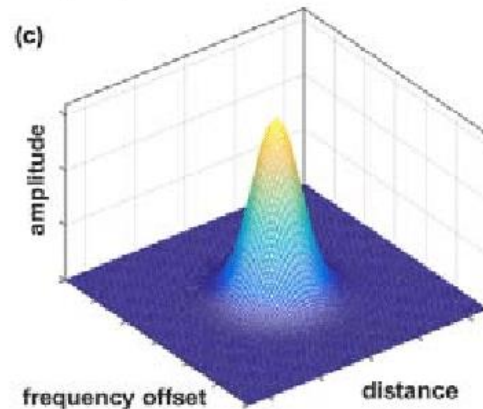
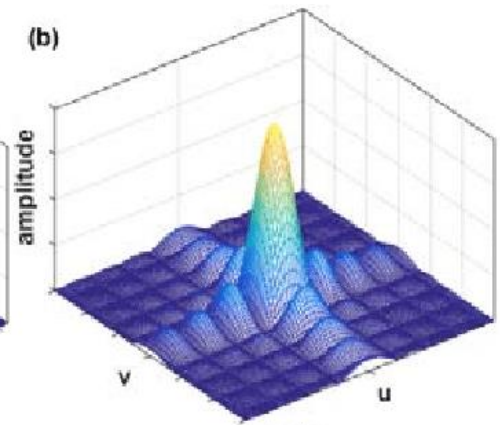
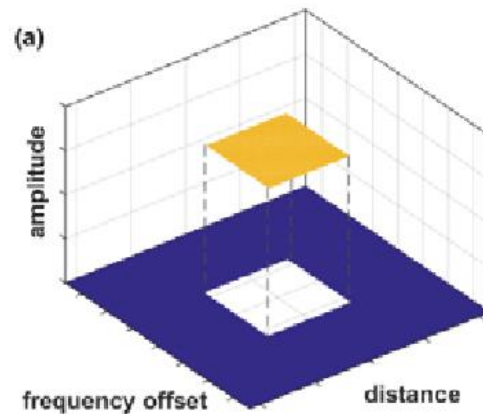
sobel\_x





# Why prewitt has waves?

- Recalling the mean filter – we can say that prewitt is like two side by side rectangles.
- Sobel is like two gaussians side by side!



# Contents

- Intro to edges
- Basic edge image
- Edge thinning
  - LoG
  - NMS
- Edge mask
- Canny edge detector
- Other edge related topics
  - Frequency representation
  - **Unsharp filter**

# Image sharpening

**Obtain** the high frequencies magnitude image.

**Enhance** the edges (e.g. by multiplying with a constant  $> 1$ ).

**Add** the enhanced edges back to the original image.

- Or- one liner:

$$f_{sharpen} = f + \gamma \cdot \|\nabla f\|$$



# Unsharp filter

- The former can also be done with only low pass filtering!

$$f_{unsharp} = f + \gamma(f - h_{blur} * f)$$

- This was also the way that photographers enhanced edges before CV (dates to the 1930s). More on this topic here:

[https://en.wikipedia.org/wiki/Unsharp\\_masking#Photographic darkroom unsharp masking](https://en.wikipedia.org/wiki/Unsharp_masking#Photographic_darkroom_unsharp_masking)

