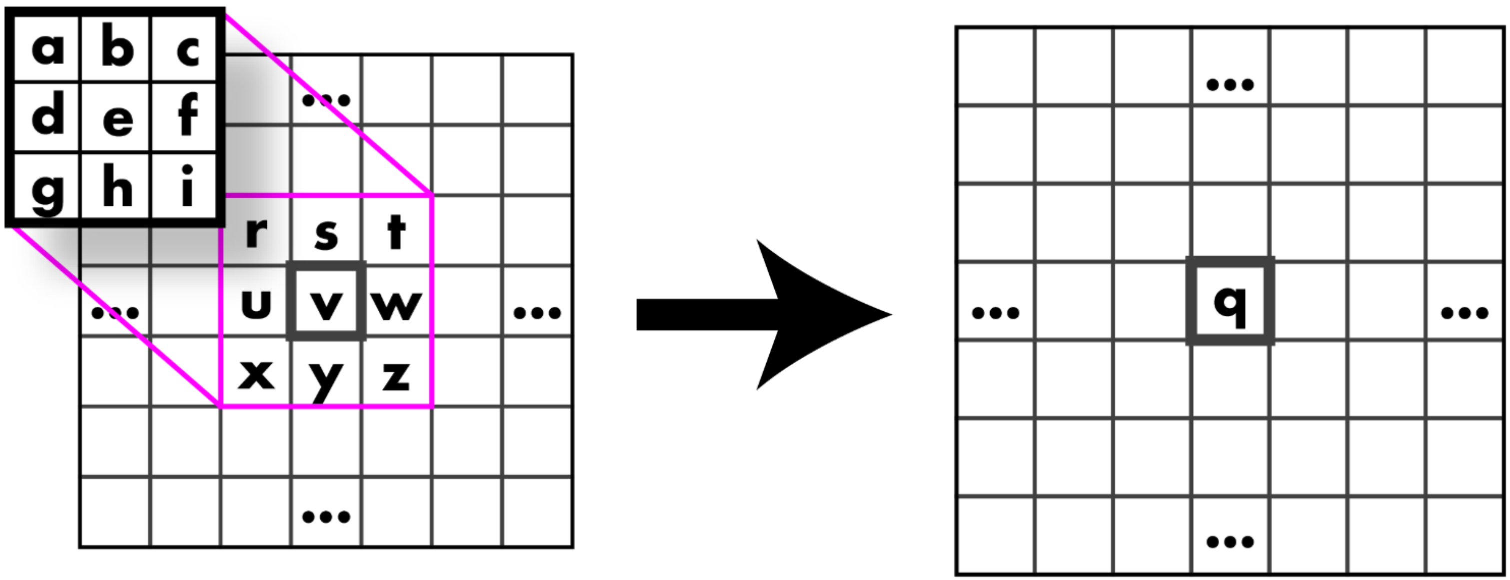
Computer Vision

CSE 455
Edges and Lines

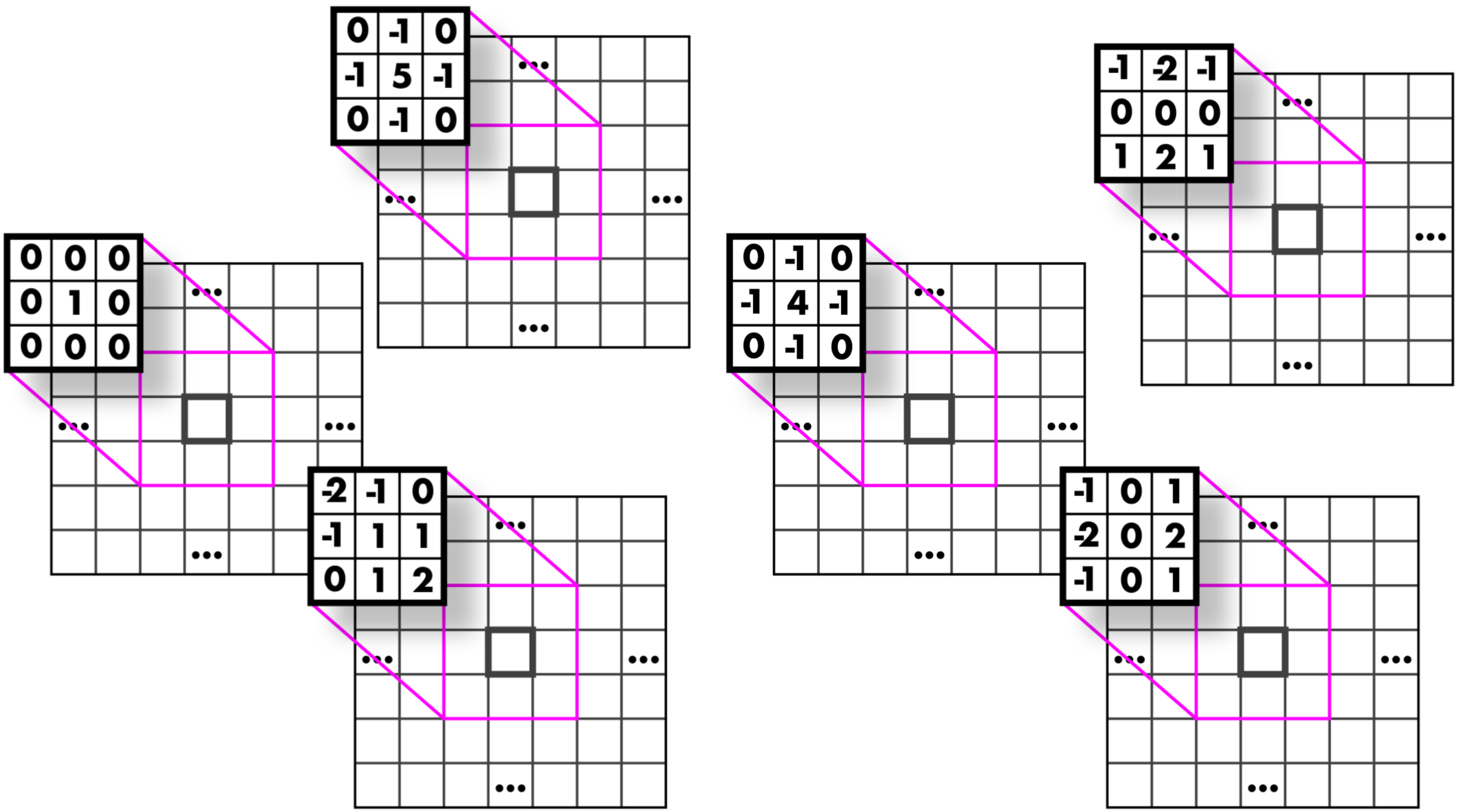
Linda Shapiro
Professor of Computer Science & Engineering
Professor of Electrical Engineering

Convolution: Weighted sum over pixels

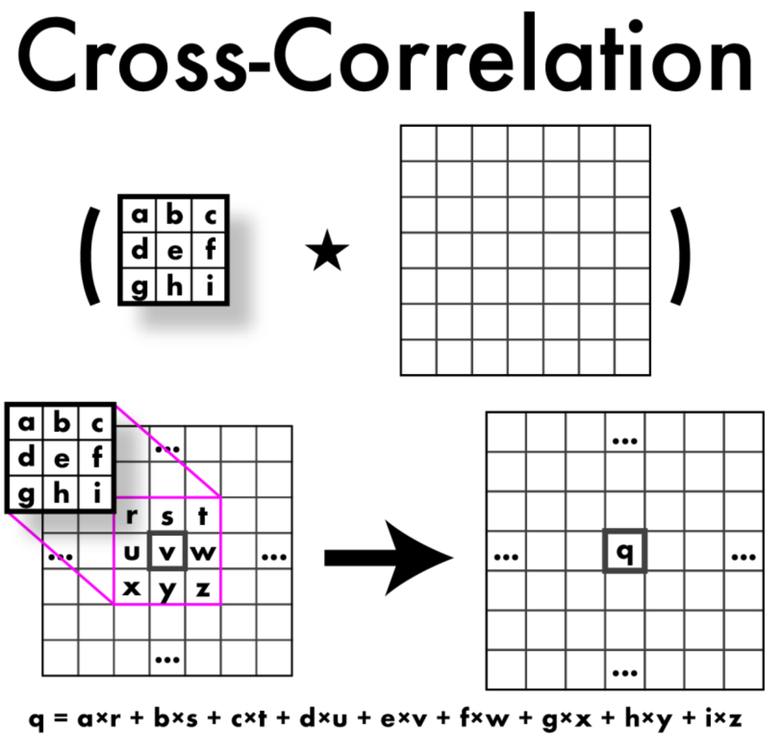


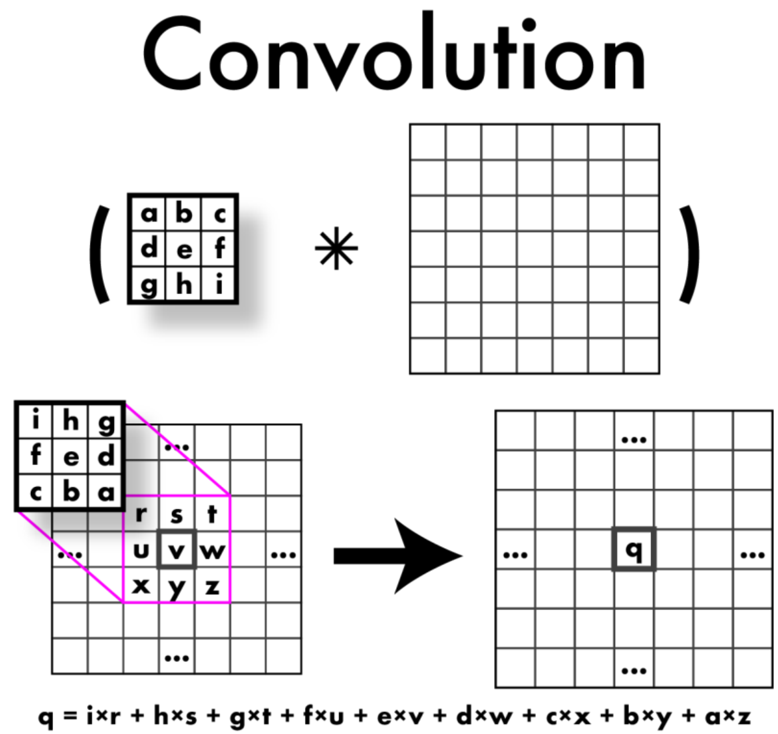
$$q = a \times r + b \times s + c \times t + d \times u + e \times v + f \times w + g \times x + h \times y + i \times z$$

Filters


0 -1 0
-1 5 -1
0 -1 0
0 0 0
0 1 0
0 0 0
-2 -1 0
-1 1 1
0 1 2
0 -1 0
-1 4 -1
0 -1 0
-1 -2 -1
0 0 0
1 2 1
-1 0 1
-2 0 2
-1 0 1

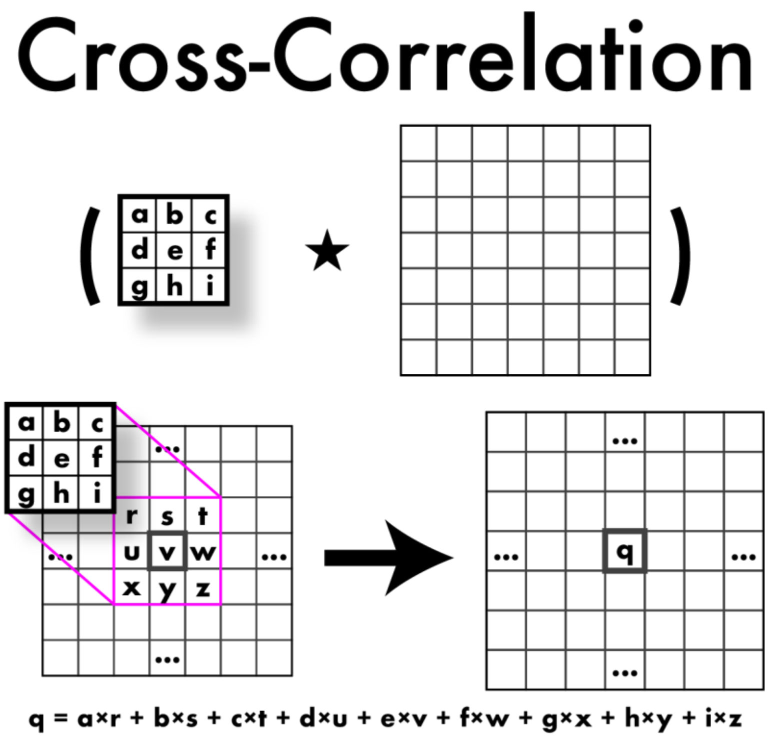
Cross-Correlation vs Convolution

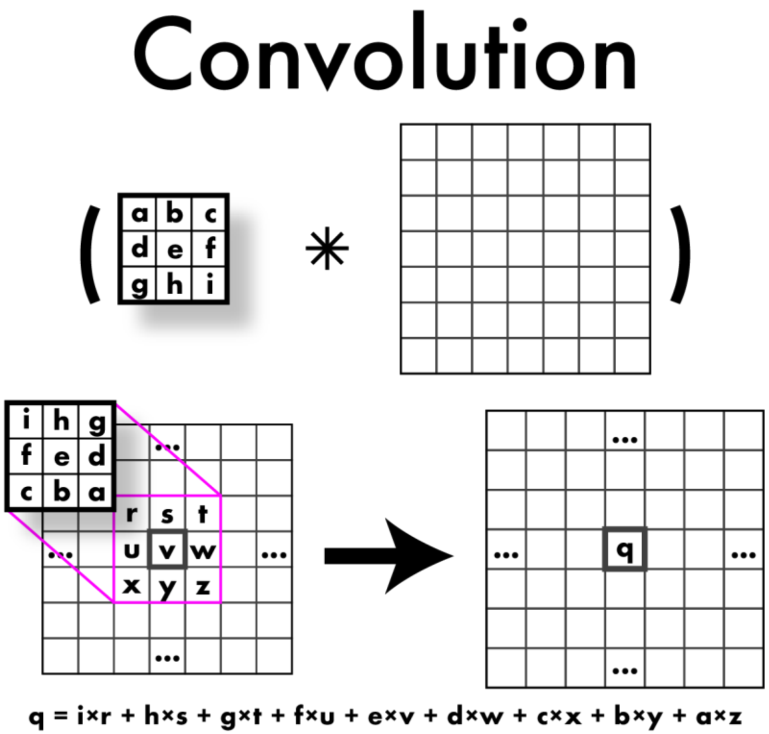




Cross-Correlation vs Convolution

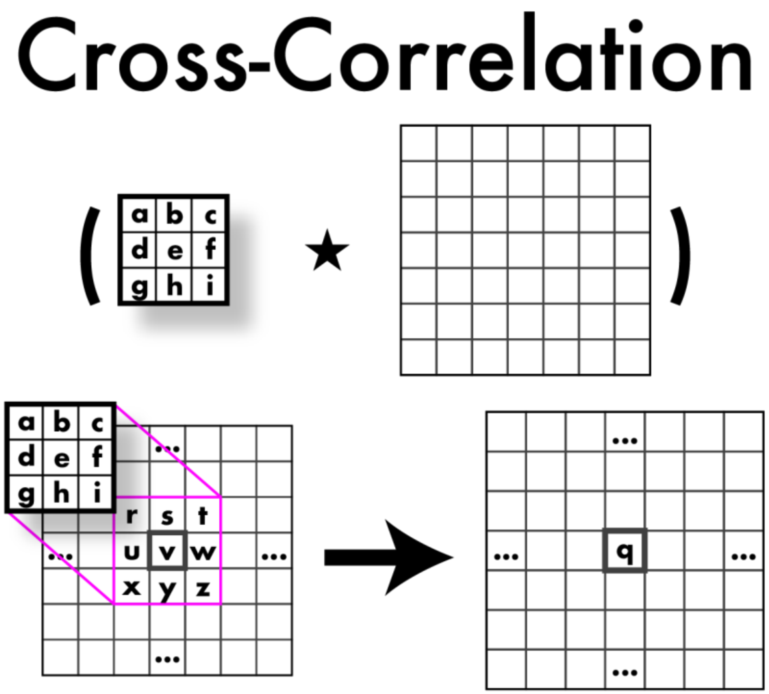
Do this in HW!



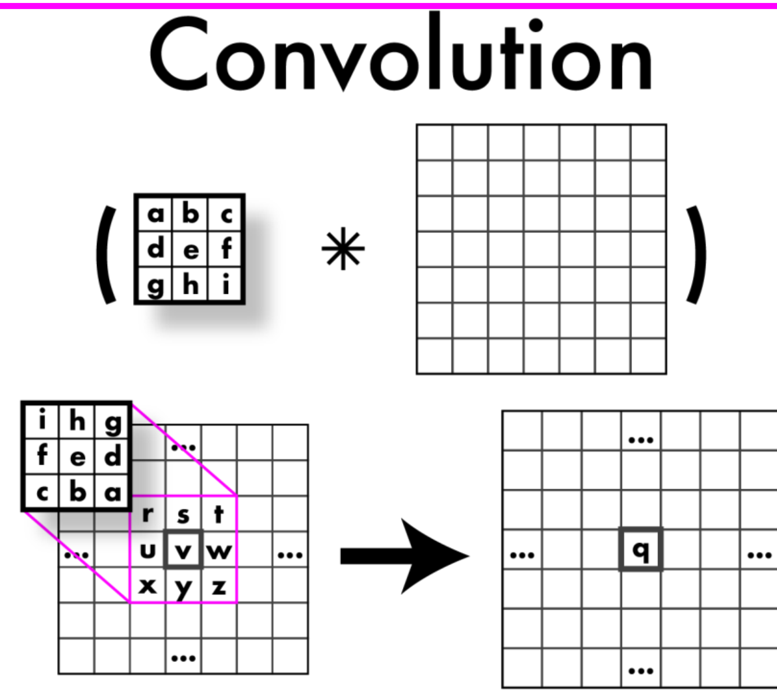


Cross-Correlation vs Convolution

These come from signal processing and have nice mathematical properties.



q = a×r + b×s + c×t + d×u + e×v + f×w + g×x + h×y + i×z



q = i×r + h×s + g×t + f×u + e×v + d×w + c×x + b×y + a×z

So what can we do with these convolutions anyway?

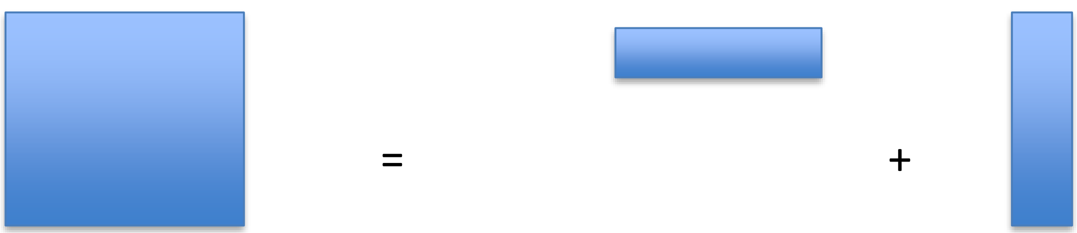
Mathematically: all the nice things

- Commutative
  - A*B = B*A
- Associative
  - A*(B*C) = (A*B)*C
- Distributes over addition
  - A*(B+C) = A*B + A*C
- Plays well with scalars
  - x(A*B) = (xA)*B = A*(xB)
- BUT WE TEND TO USE CORRELATION BECAUSE OUR FILTERS ARE SYMMETRIC, AND THEN WE JUST CALL IT CONVOLUTION!

So what can we do with these convolutions anyway?

This means some convolutions decompose:

- 2D Gaussian is just composition of 1D Gaussians
  - Faster to run 2 1D convolutions



So what can we do with these convolutions anyway?

- Blurring
- Sharpening
- Edges
- Features
- Derivatives
- Super-resolution
- Classification
- Detection
- Image captioning
- ...

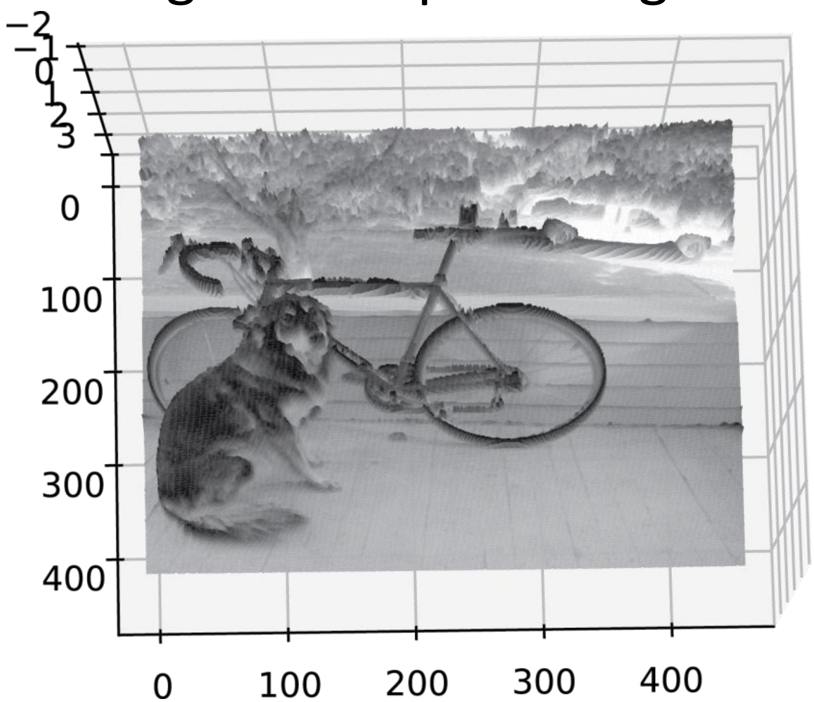
So what can we do with these convolutions anyway?

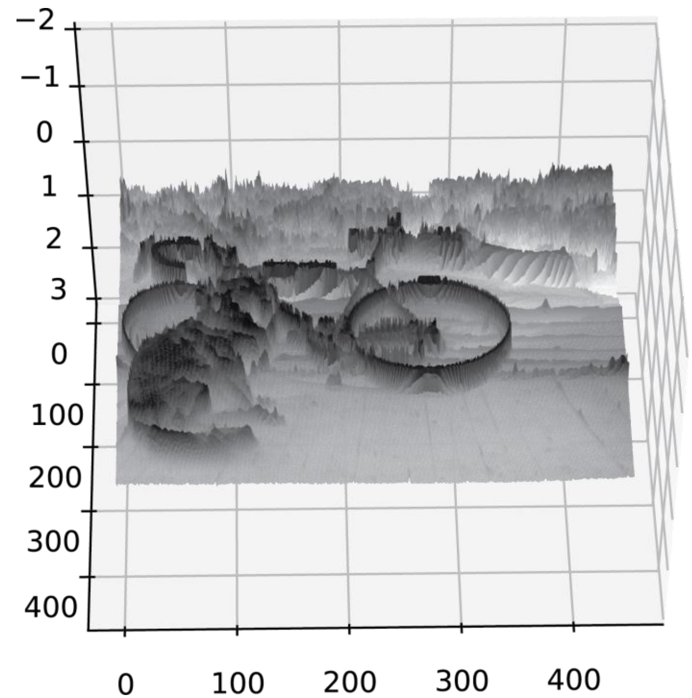
- Blurring
- Sharpening
- Edges
- Features
- Derivatives
- Super-resolution
- Classification
- Detection
- Image captioning
- ...

Much of low-level computer vision is **convolutions** (basically)

What's an edge?

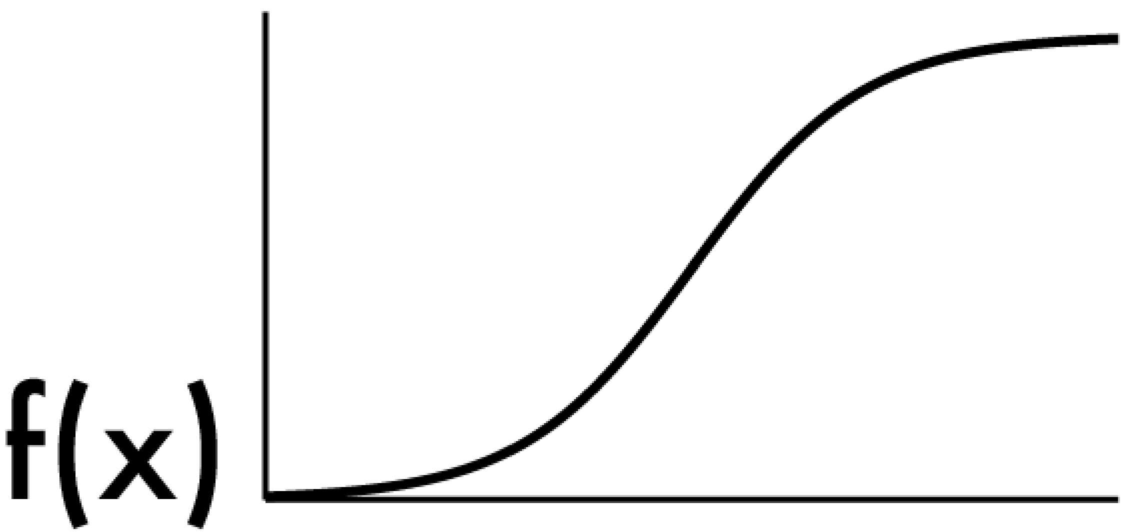
- Image is a function.
- Think of the gray tones as HEIGHTS.
- Edges are rapid changes in this function





What’s an edge?

- Image is a function
- Edges are rapid changes in this function



Finding edges

- Could take derivative
- Edges = high response

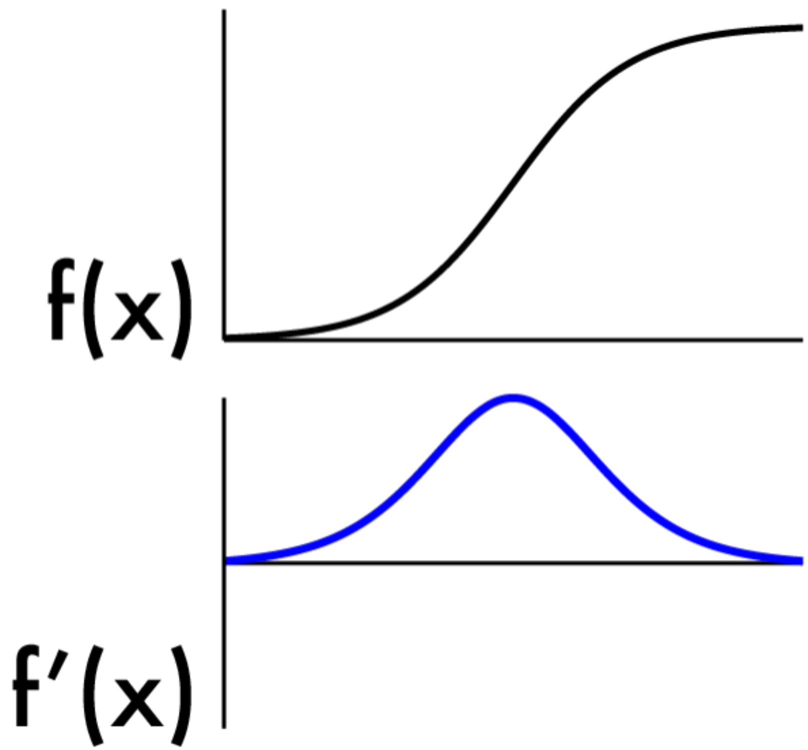


Image derivatives

- Recall:
  - $$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}.$$
- We don't have an "actual" function, must estimate
- Possibility: set h = 1
- What will that look like?

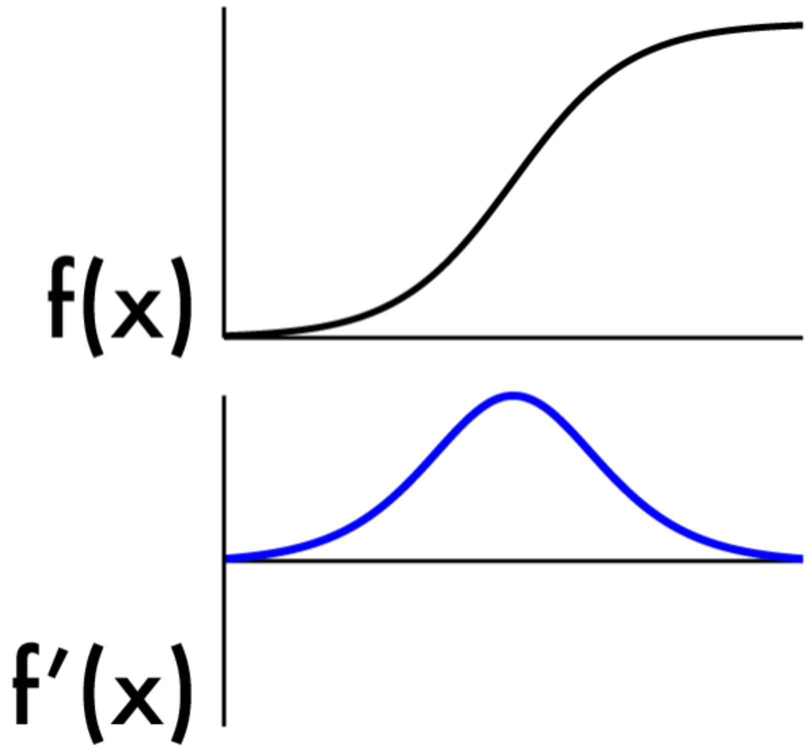
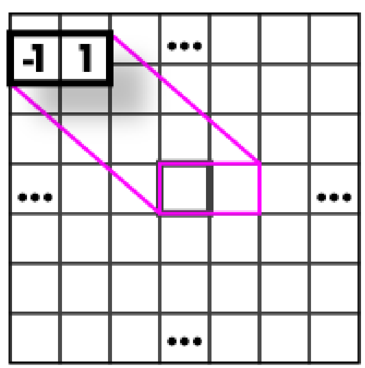


Image derivatives

- Recall:
  - $f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}.$
- We don't have an "actual" Function, must estimate
- Possibility: set h = 1
- What will that look like?



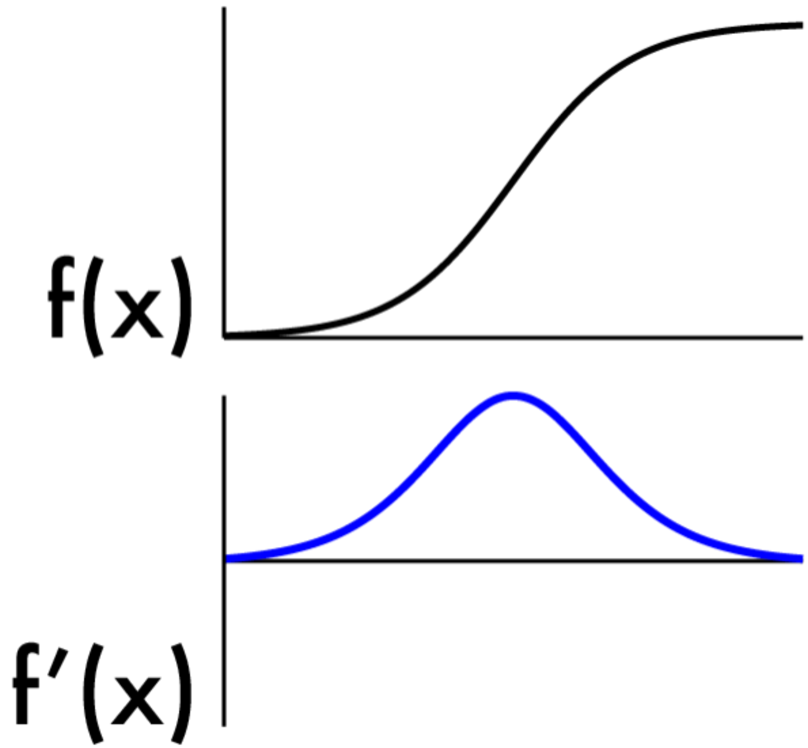


Image derivatives

- Recall:
  - $f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}.$
- We don't have an "actual" Function, must estimate
- Possibility: set h = 2
- What will that look like?

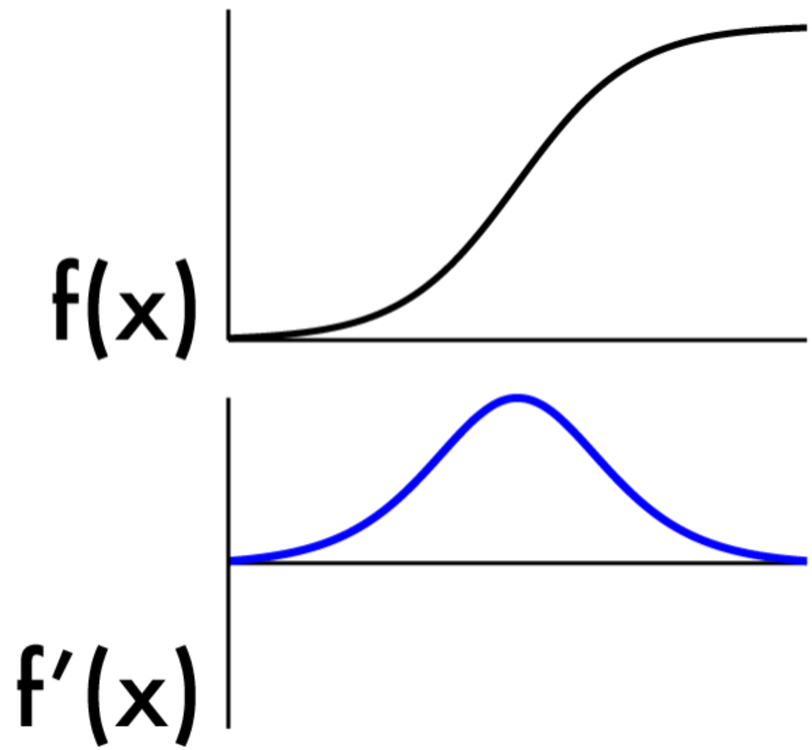
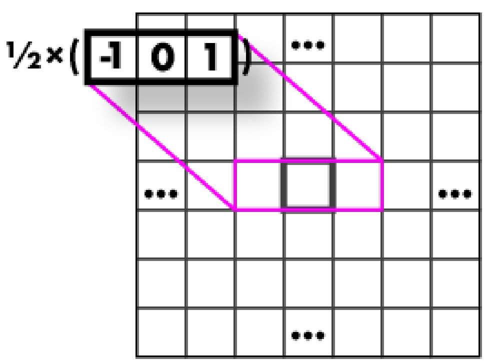
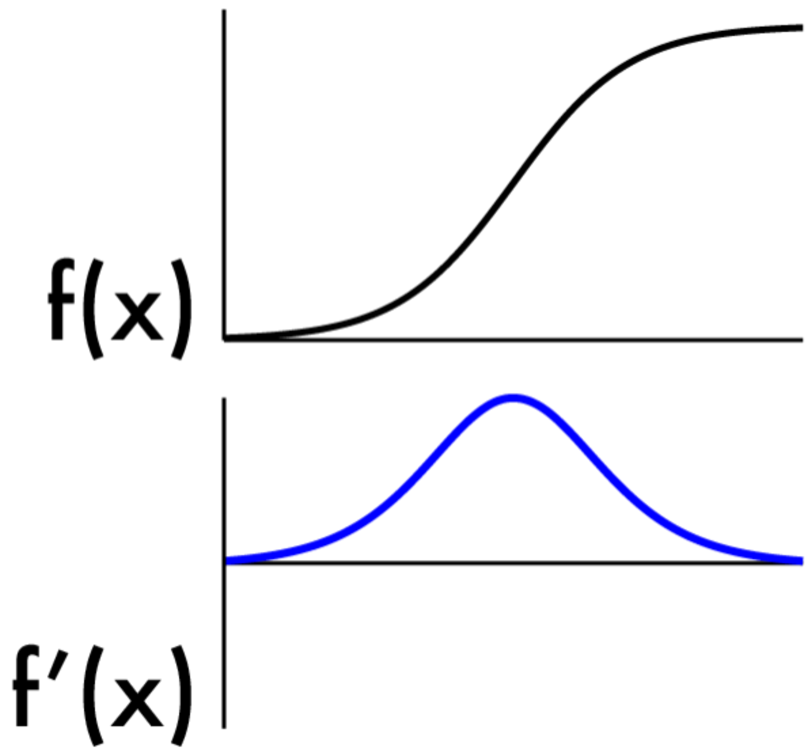


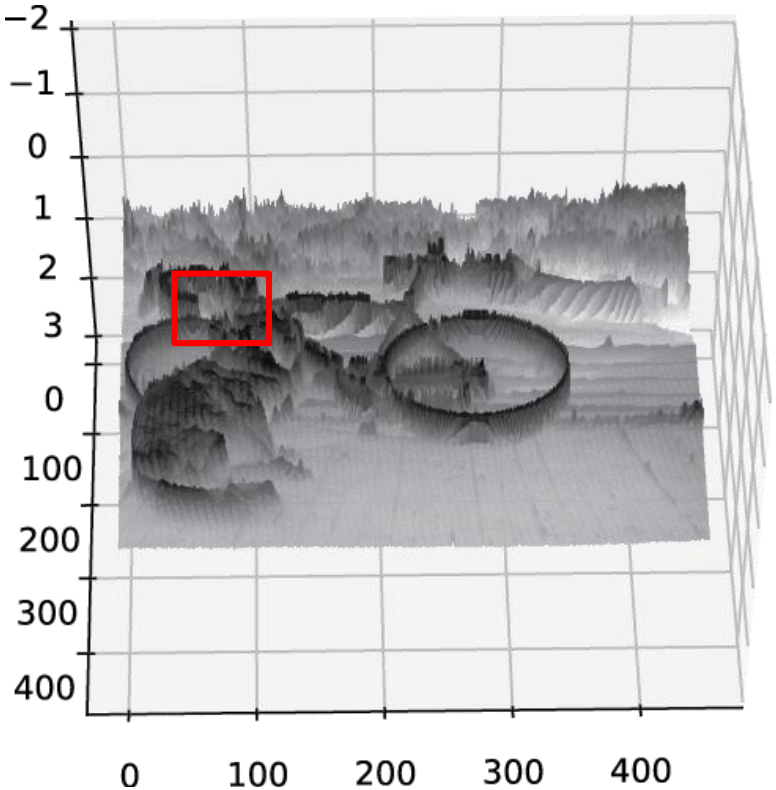
Image derivatives

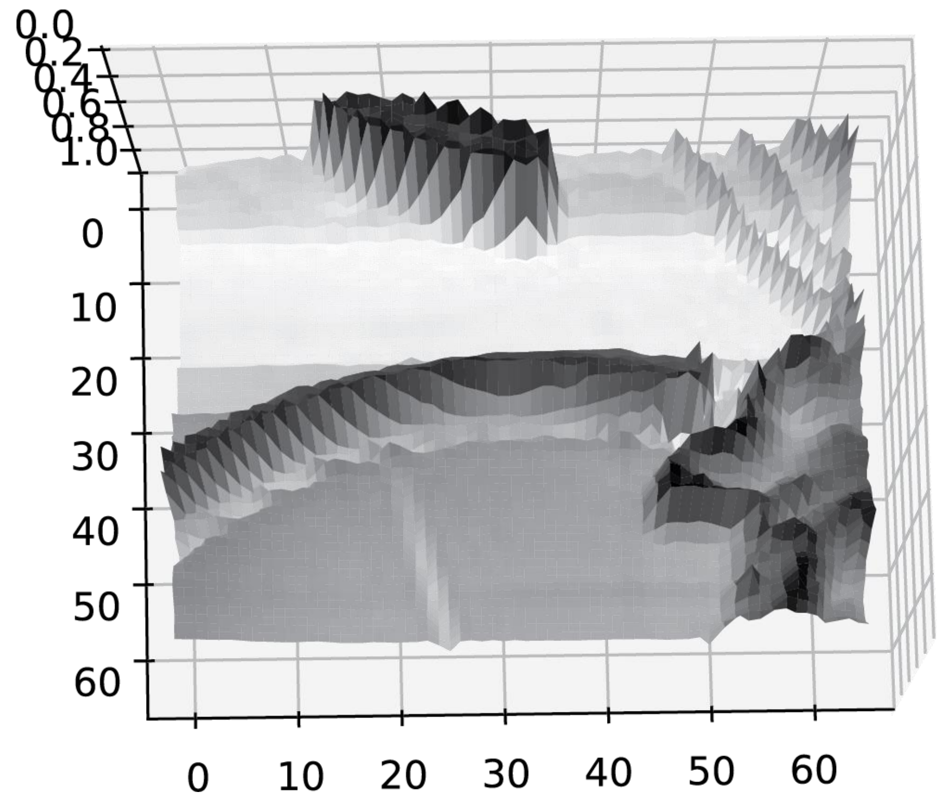
- Recall:
  - $f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}.$
- We don't have an "actual" Function, must estimate
- Possibility: set h = 2
- What will that look like?



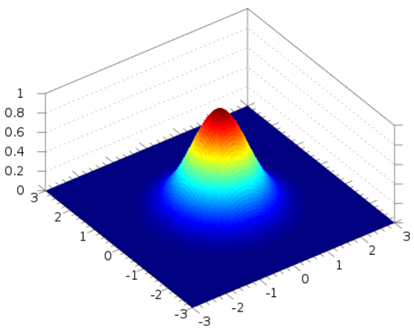


Images are noisy!

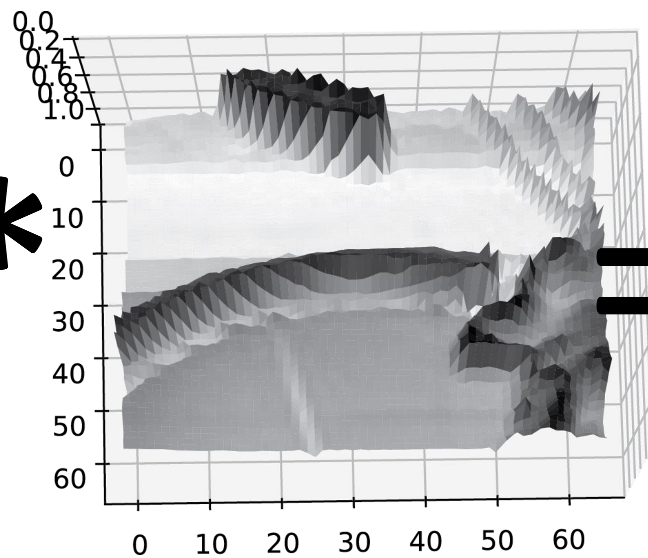




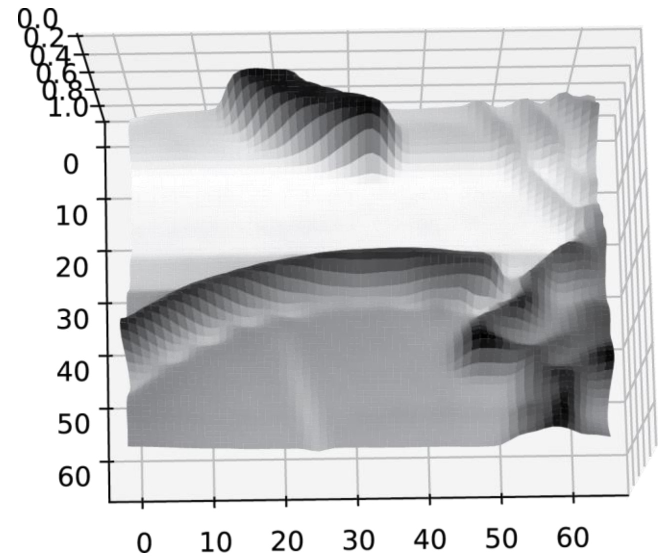
But we already know how to smooth



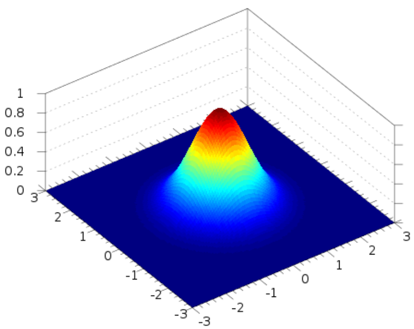
*



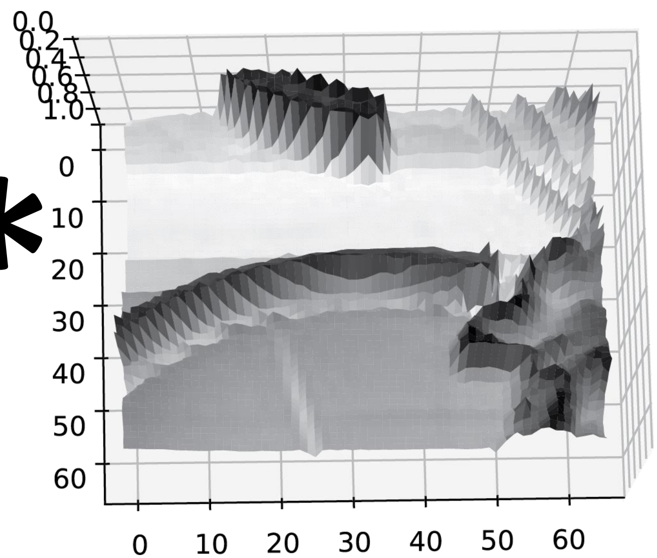
=



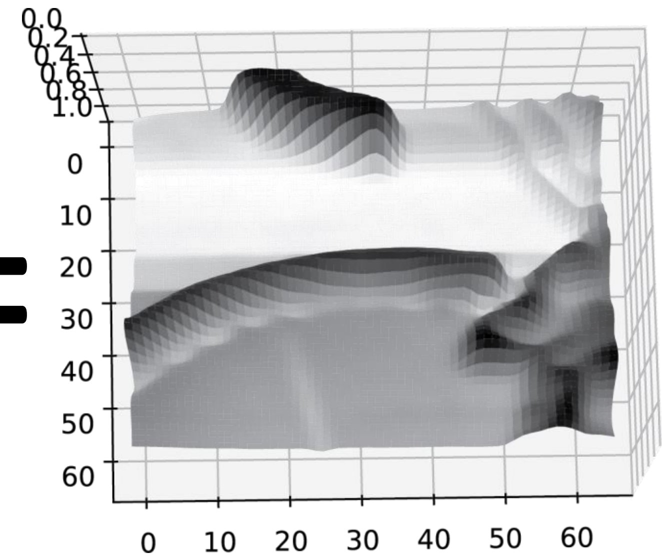
But we already know how to smooth



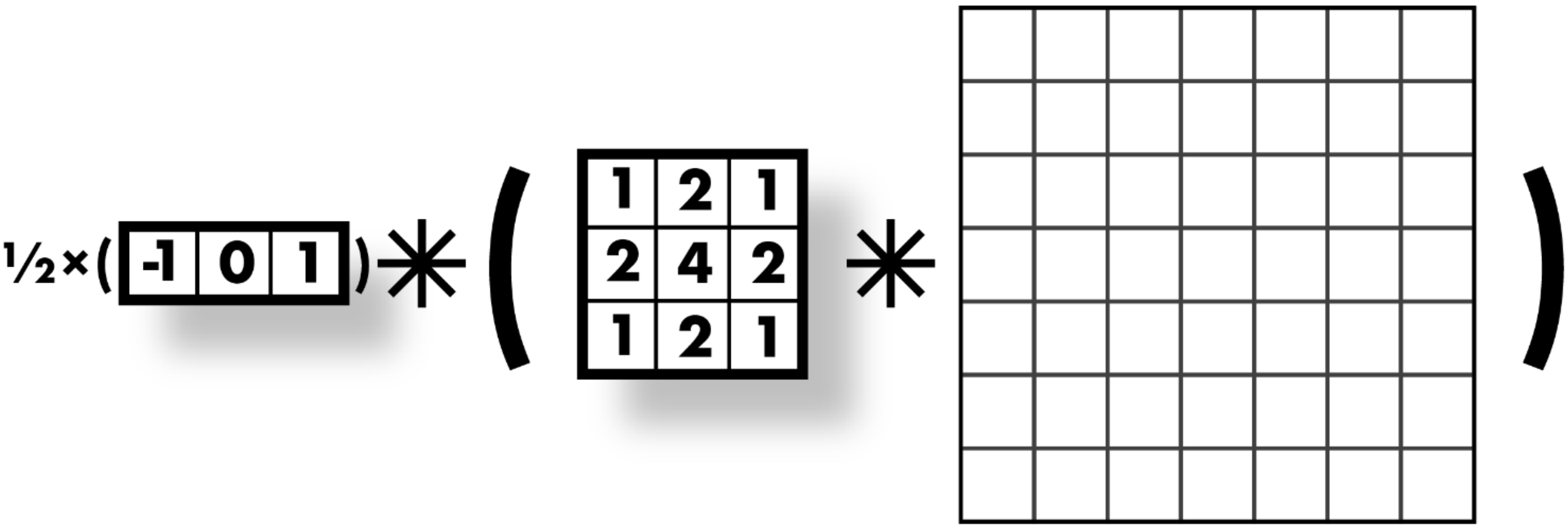
*



=

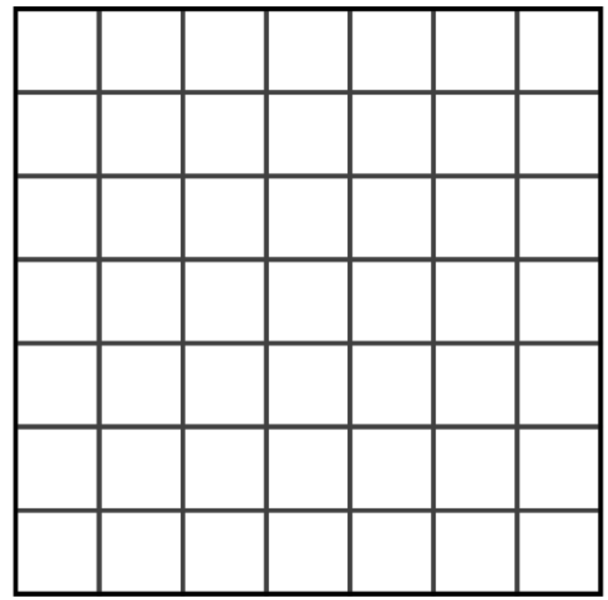


Smooth first, then derivative

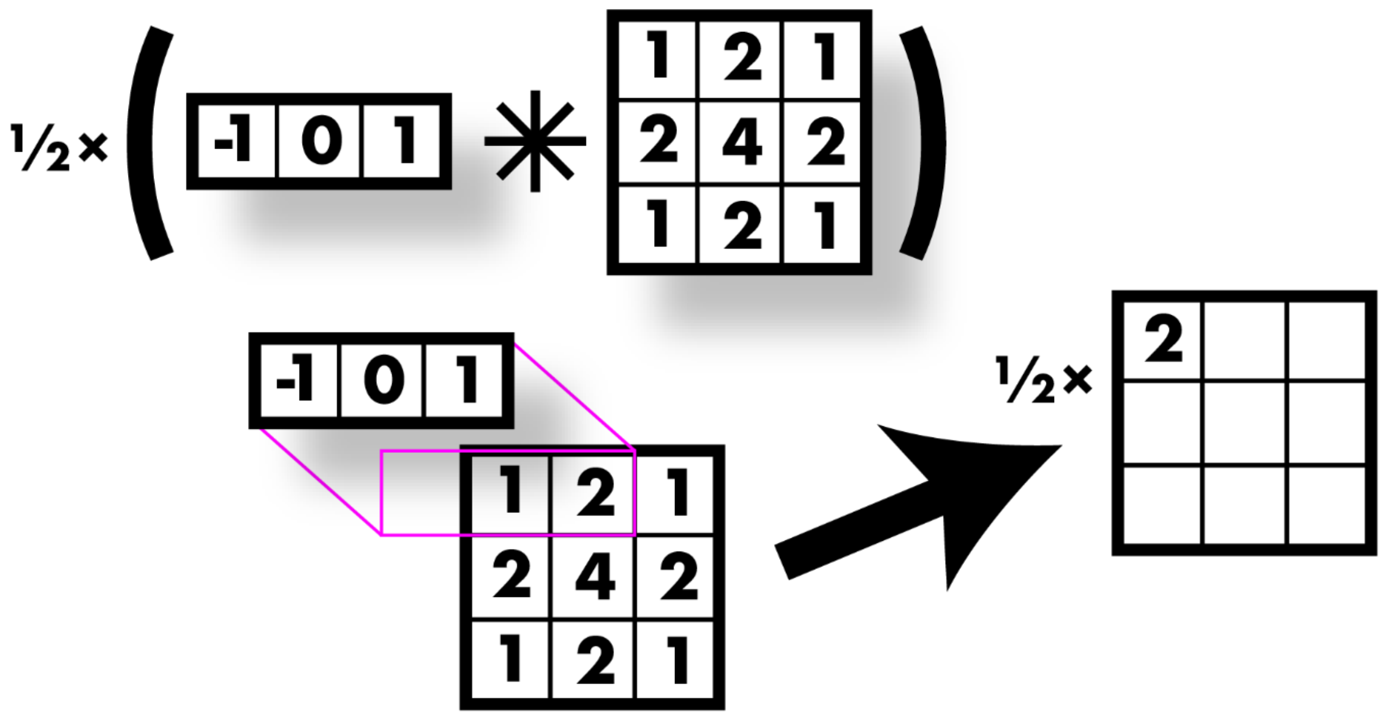


Smooth first, then derivative

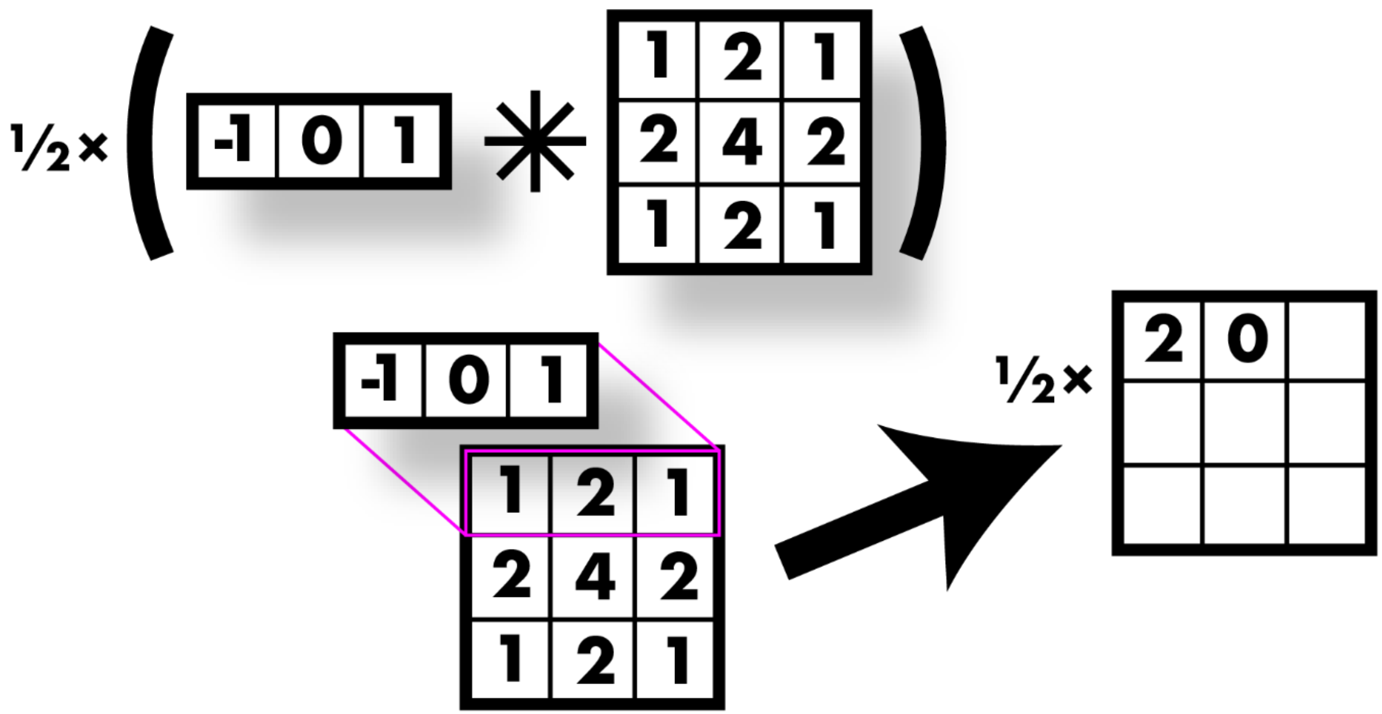
$$\frac{1}{2}\times\left(\begin{bmatrix}-1 & 0 & 1\end{bmatrix} * \begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}\right) *$$



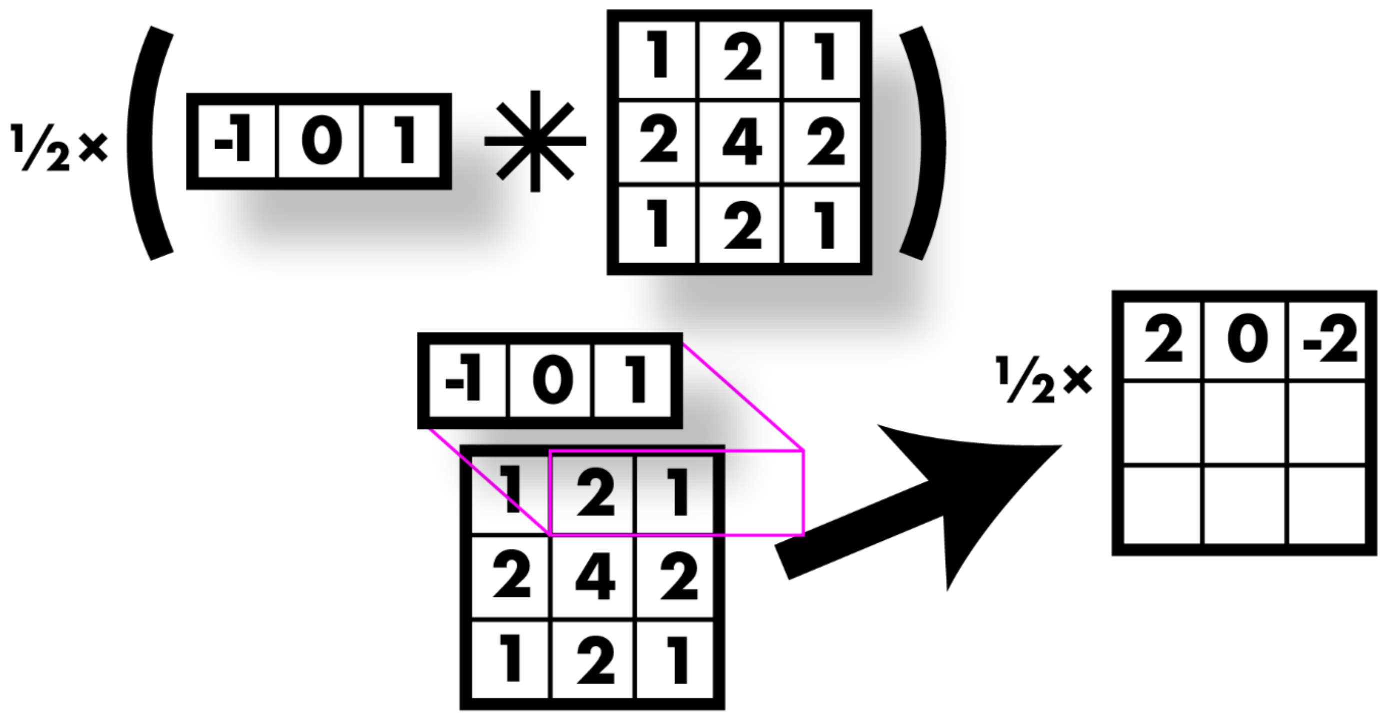
Smooth first, then derivative



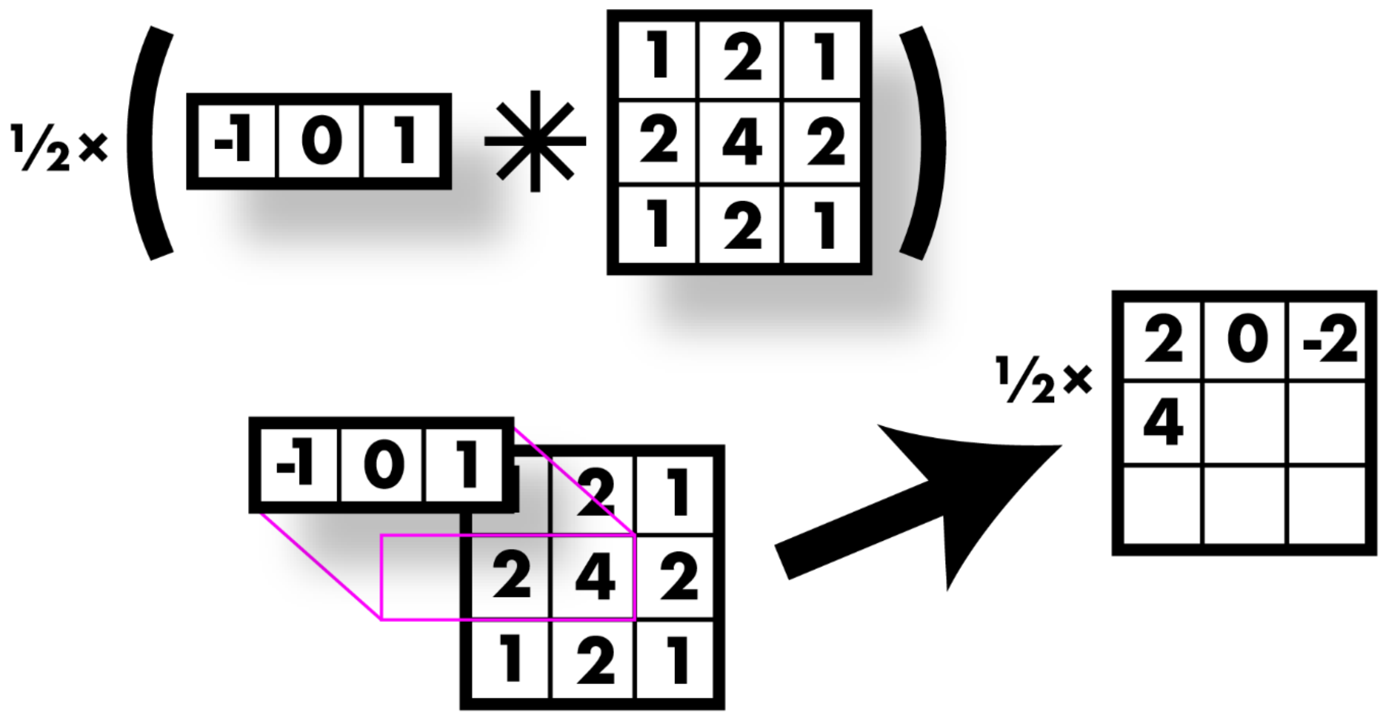
Smooth first, then derivative



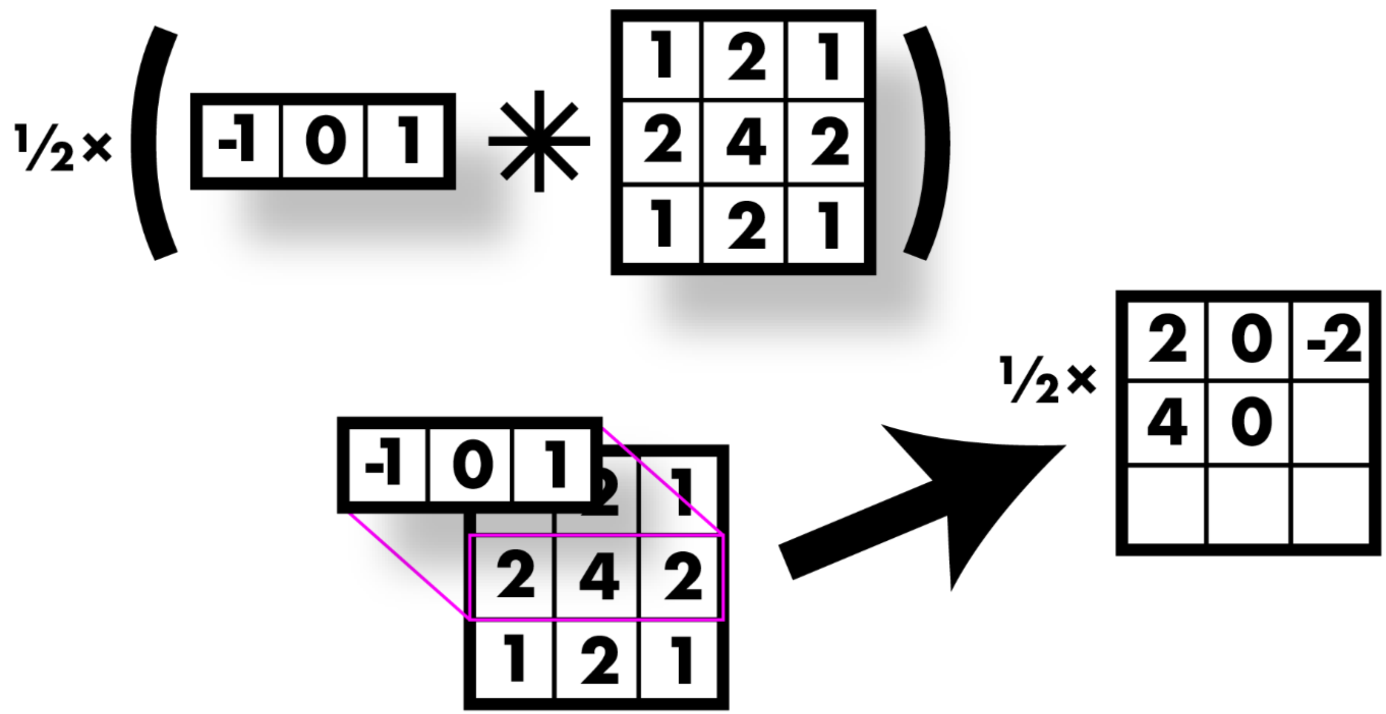
Smooth first, then derivative



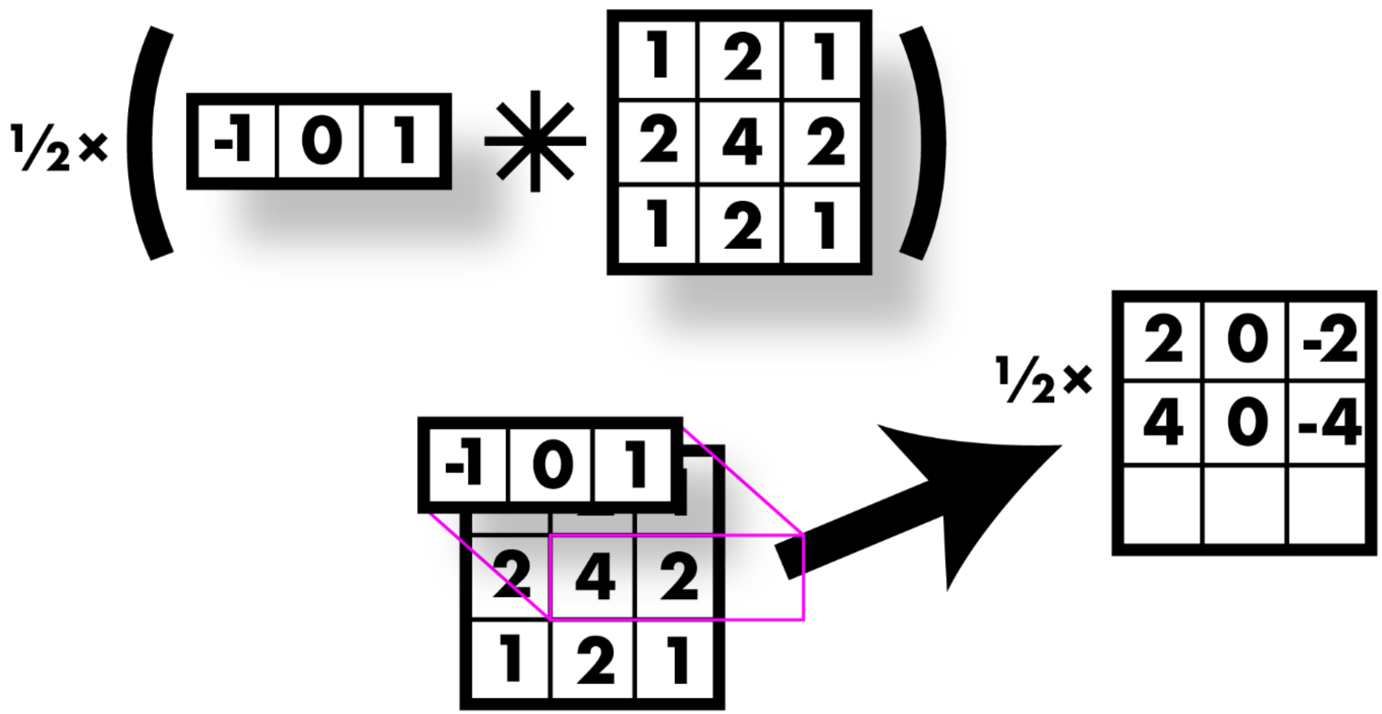
Smooth first, then derivative



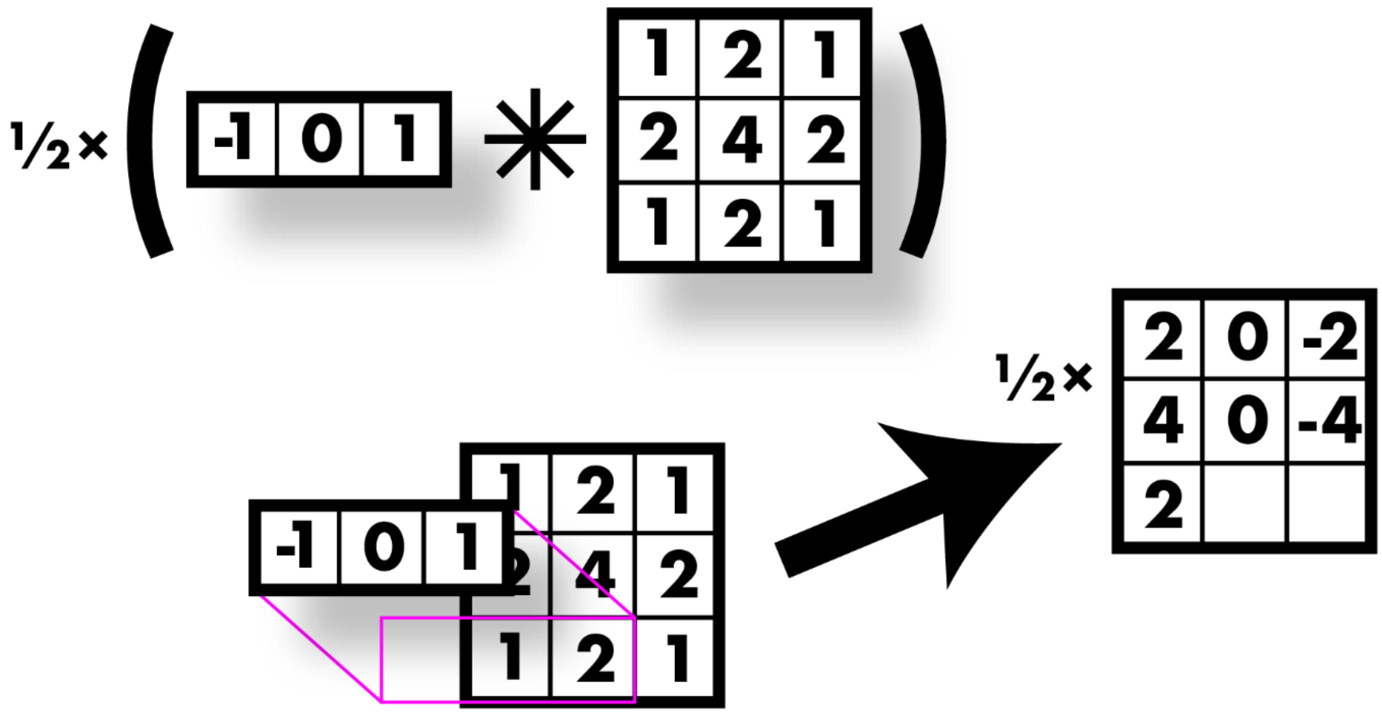
Smooth first, then derivative



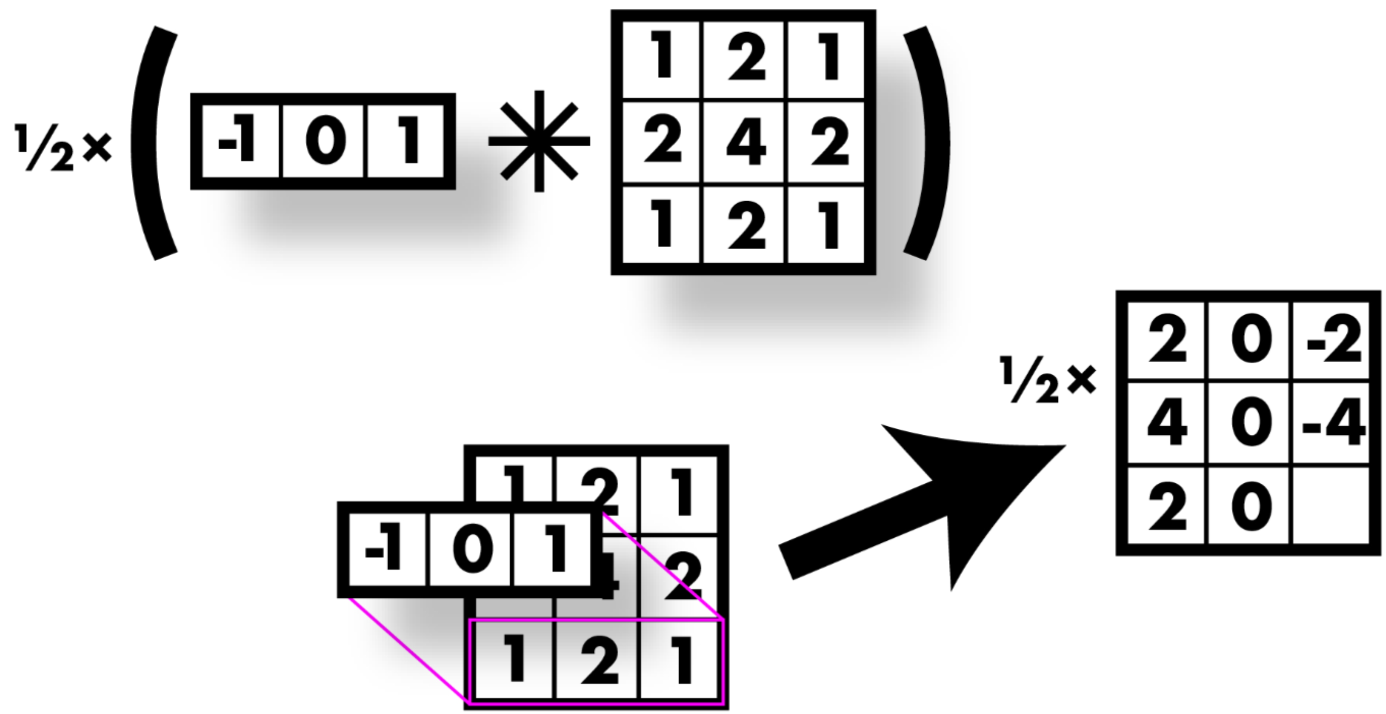
Smooth first, then derivative



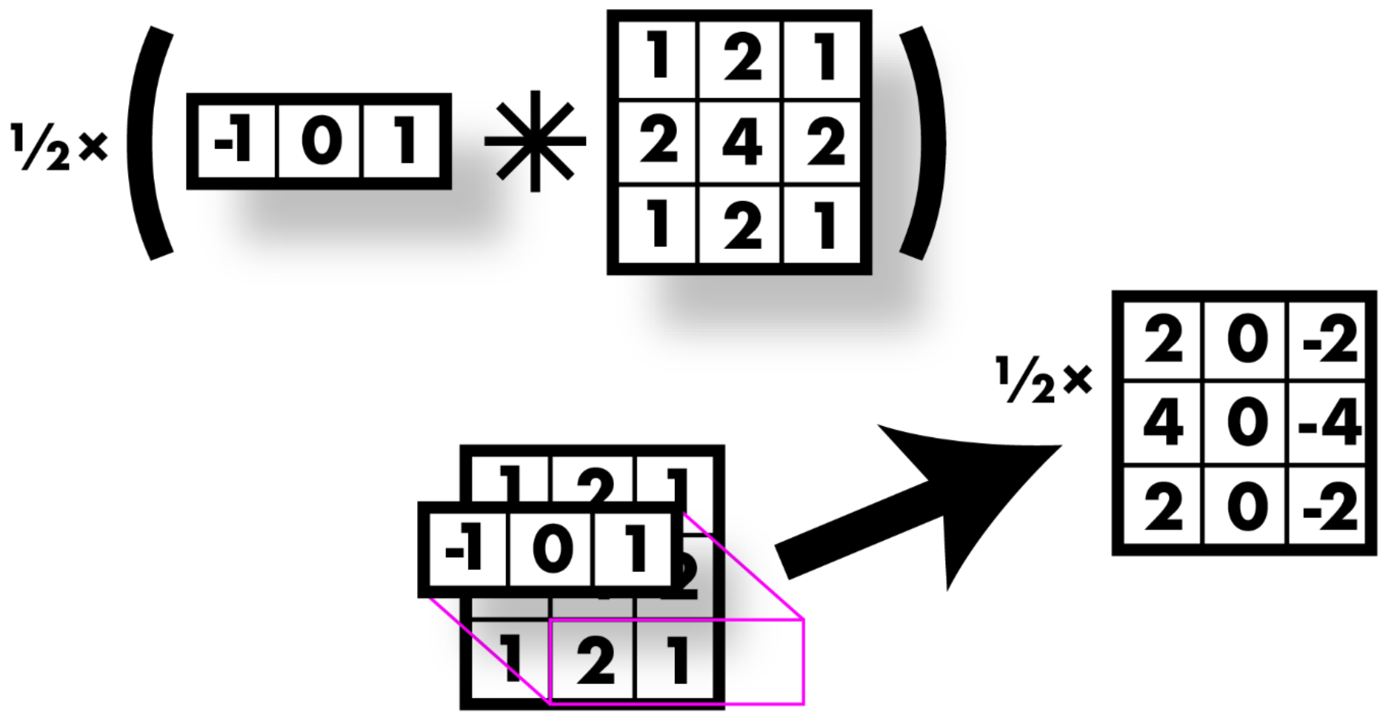
Smooth first, then derivative



Smooth first, then derivative

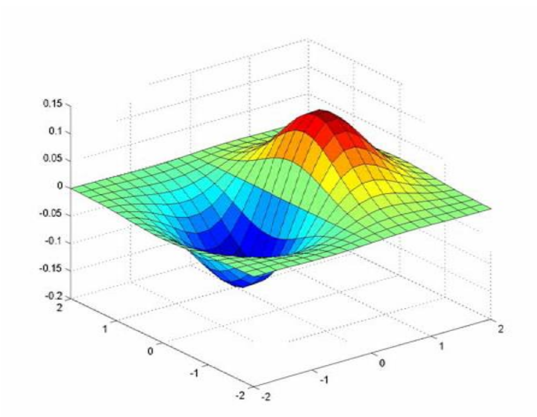


Smooth first, then derivative



Sobel filter! Smooth & derivative

$$\frac{1}{2}\times\left(\begin{bmatrix}-1 & 0 & 1\end{bmatrix} * \begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}\right)$$



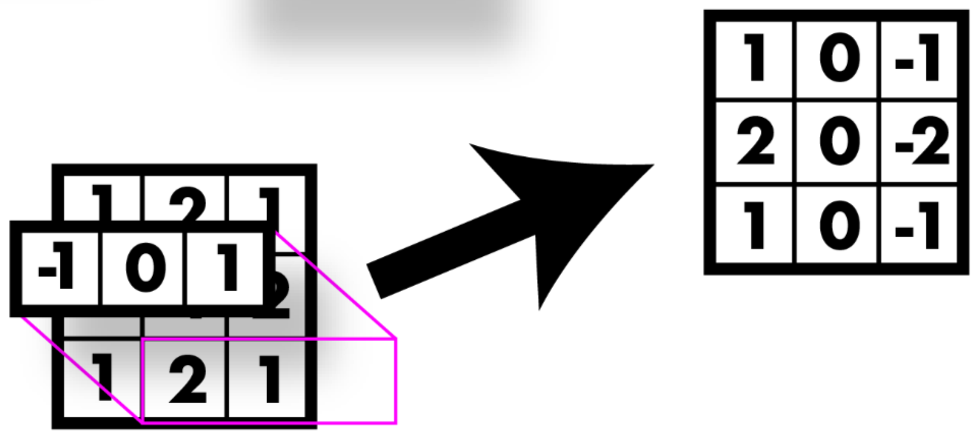
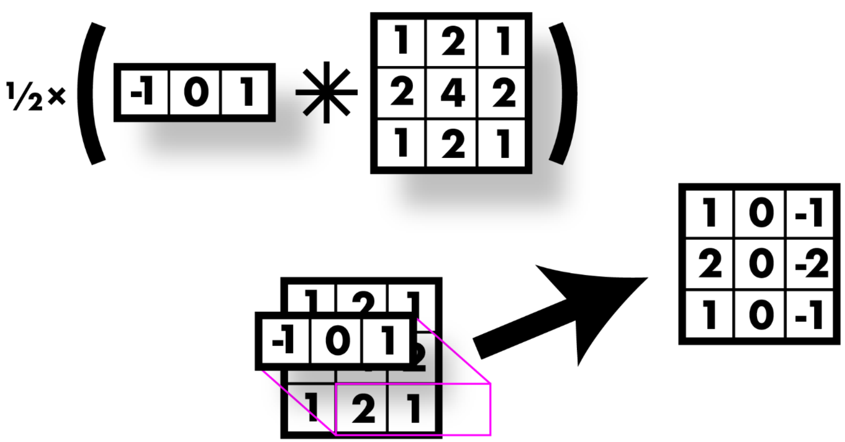
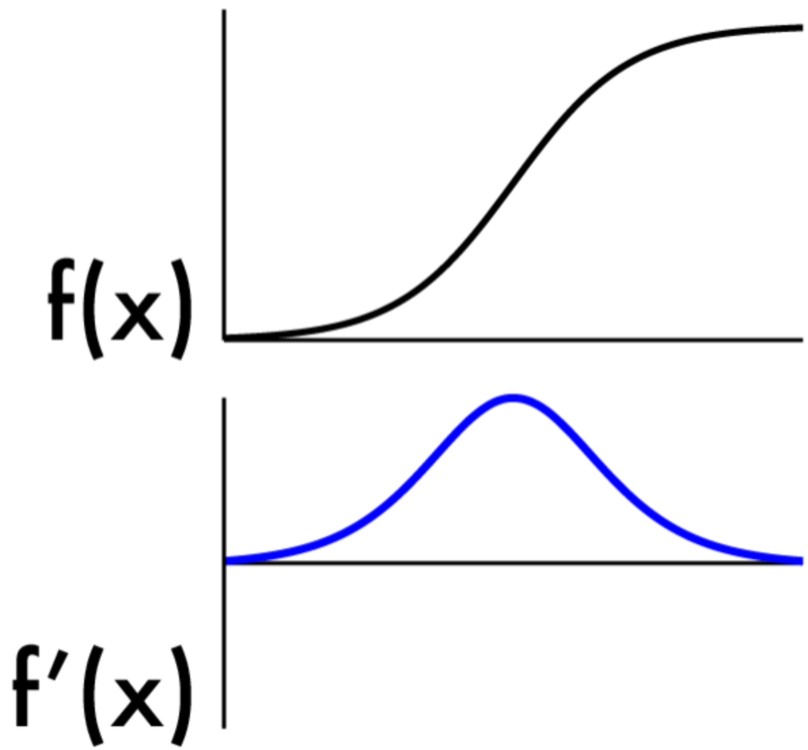


Image derivatives

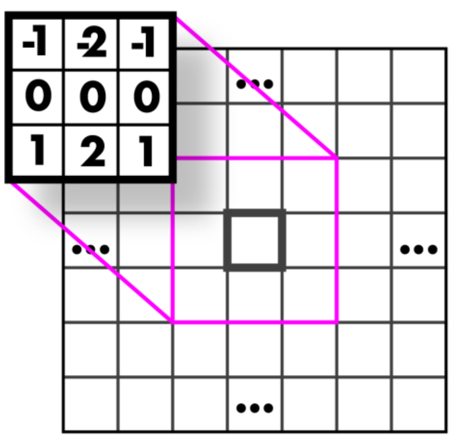
- Recall: $f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}.$
    -
- Want smoothing too!

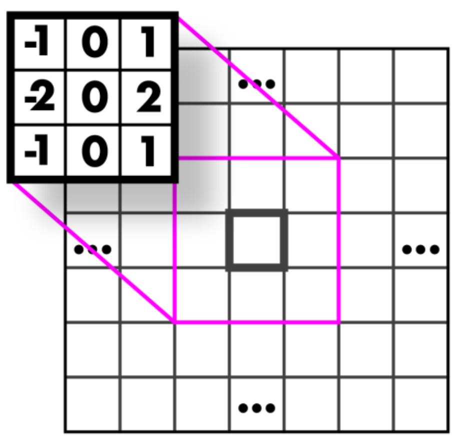


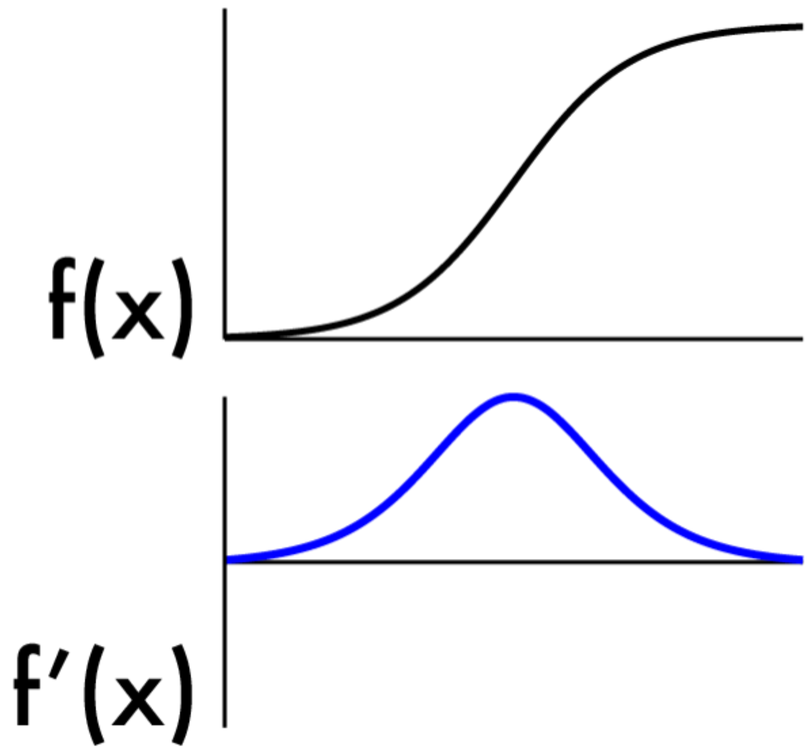


Finding edges

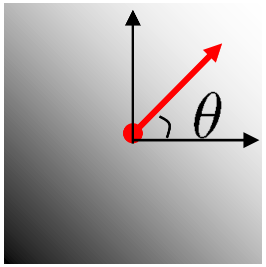
- Could take derivative
- Find high responses
- Sobel filters!
- But let’s stop a moment get some basics







Simplest image gradient



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

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

How would you implement this as a filter?

$\boxed{0 \quad -1 \quad 1}$

The gradient direction is give

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

How does this relate to the direction of the edge?

perpendicular

The *edge strength* is given by the gradient magnitude

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

Sobel operator

Who was Sobel?
Irwin Sobel (born 1940)
Consultant (HP Labs Retired – 8Mar13) · Computer Vision & Graphics

In practice, it is common to use:

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

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

Magnitude:

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

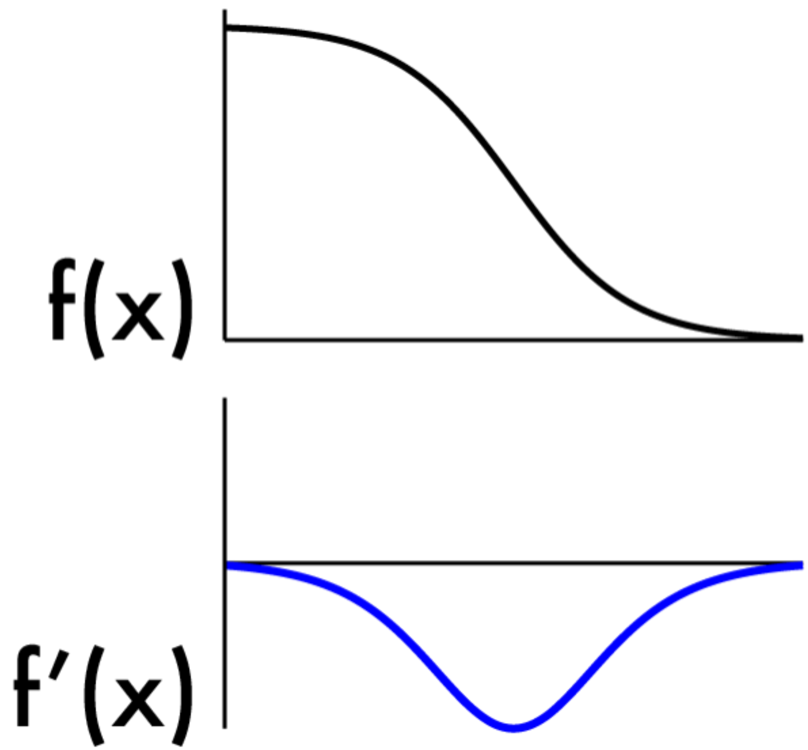
Orientation:

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

What's the C/C++ function?
Use atan2

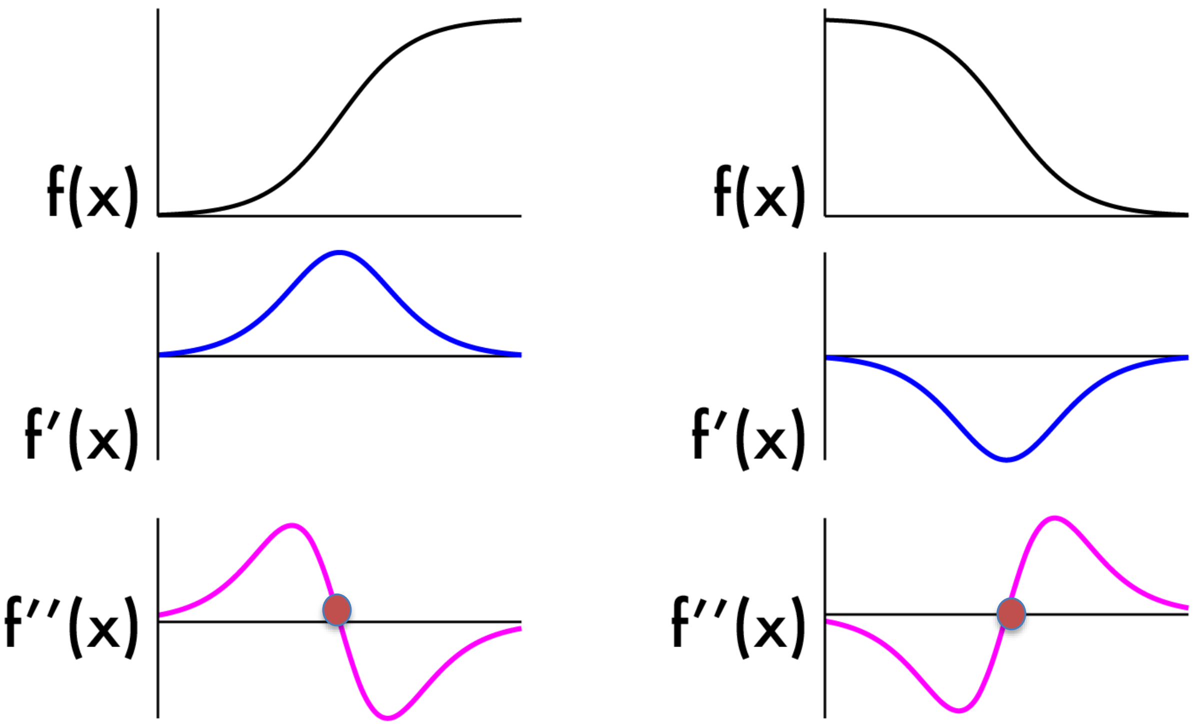
Finding edges

- Could take derivative
- Find high responses
- Sobel filters!
- But…
- Edges go both ways
- Want to find extrema



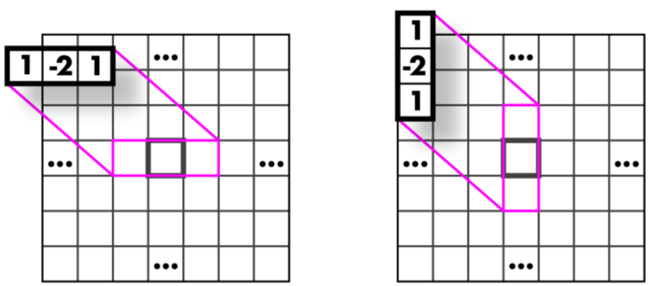
2nd derivative!

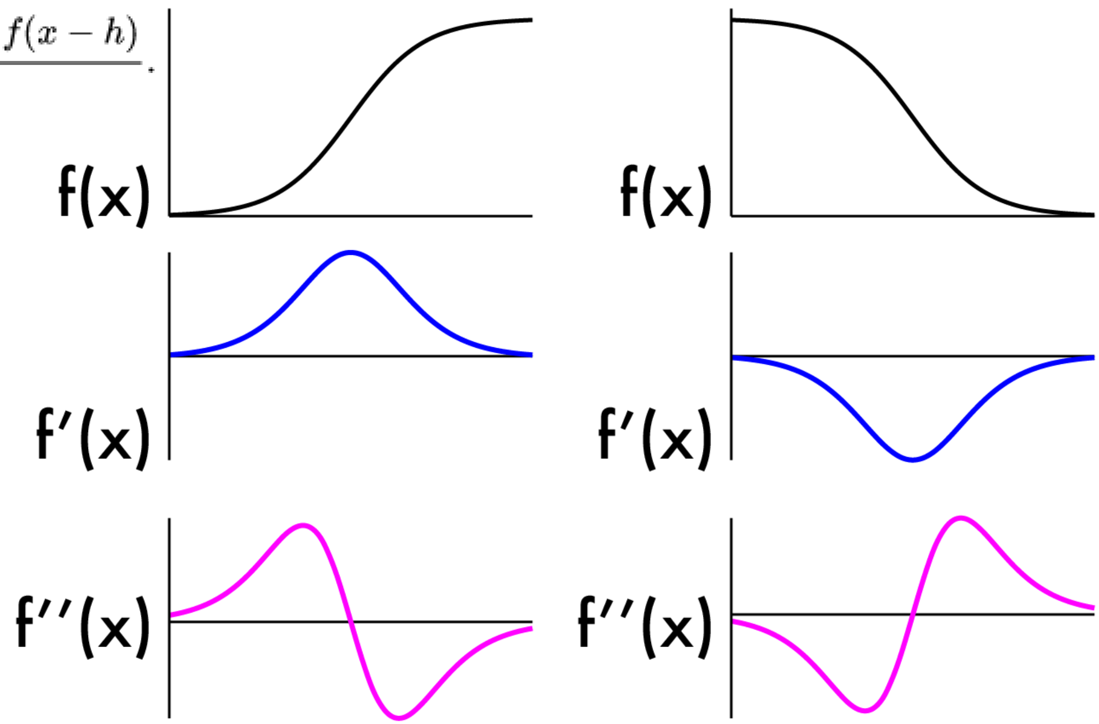
- Crosses zero at extrema



Laplacian (2nd derivative)!

- Crosses zero at extrema
- Recall:
  - $$f''(x) = \lim_{h \to 0} \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$
- Laplacian:
  - $$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$
- Again, have to estimate f''(x):





Laplacians

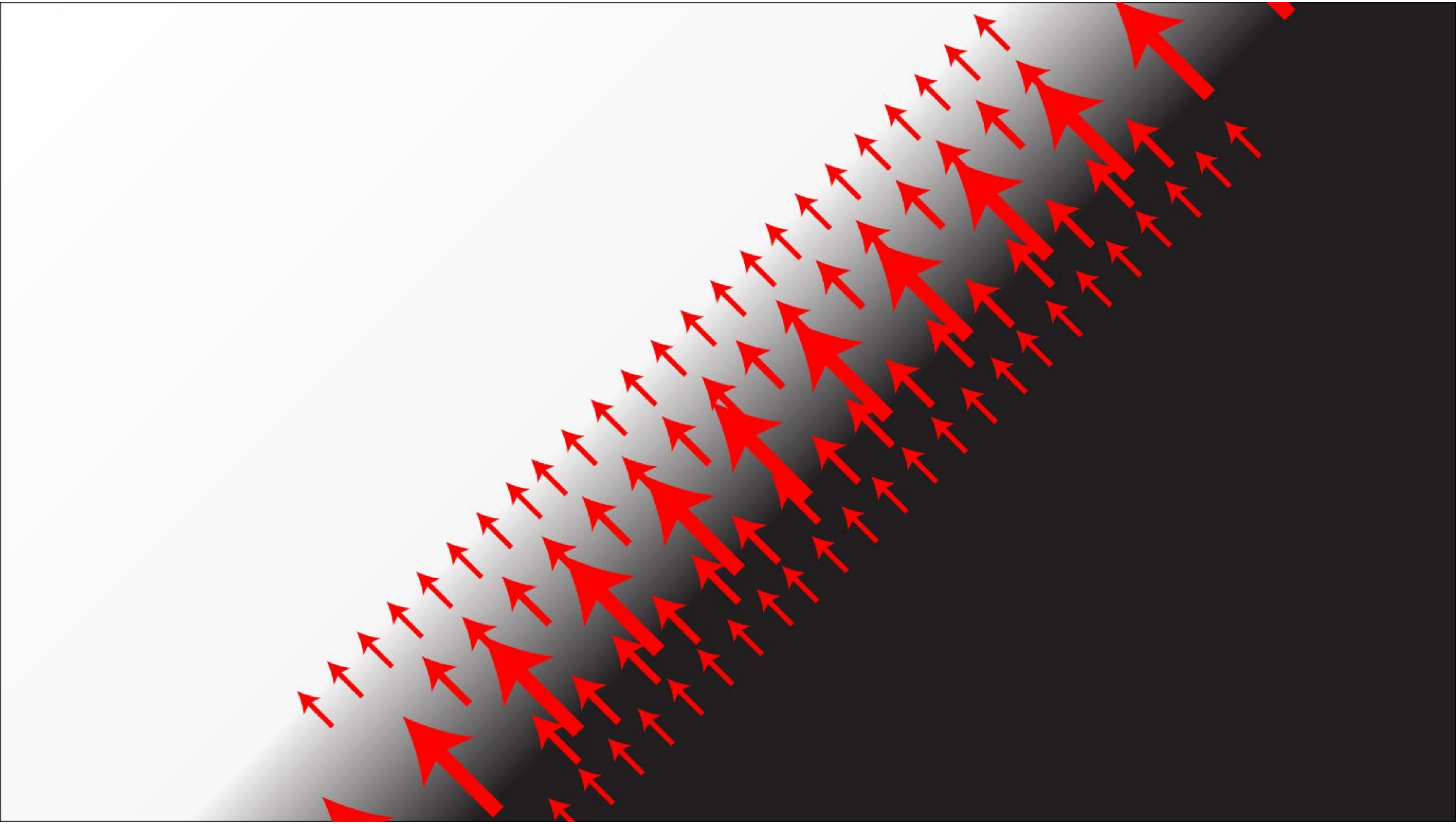
- Laplacian:
  - $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$

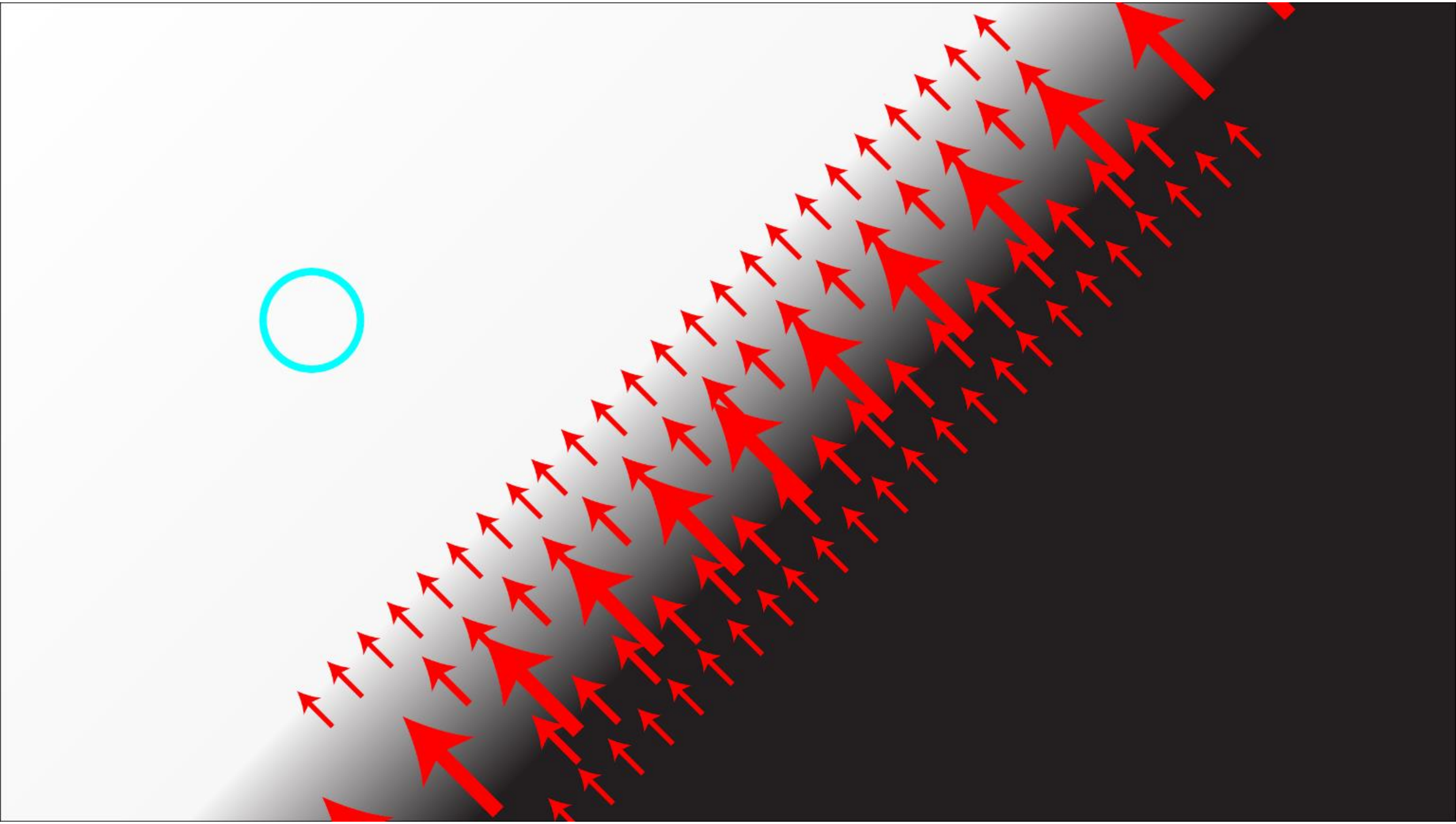
Laplacians

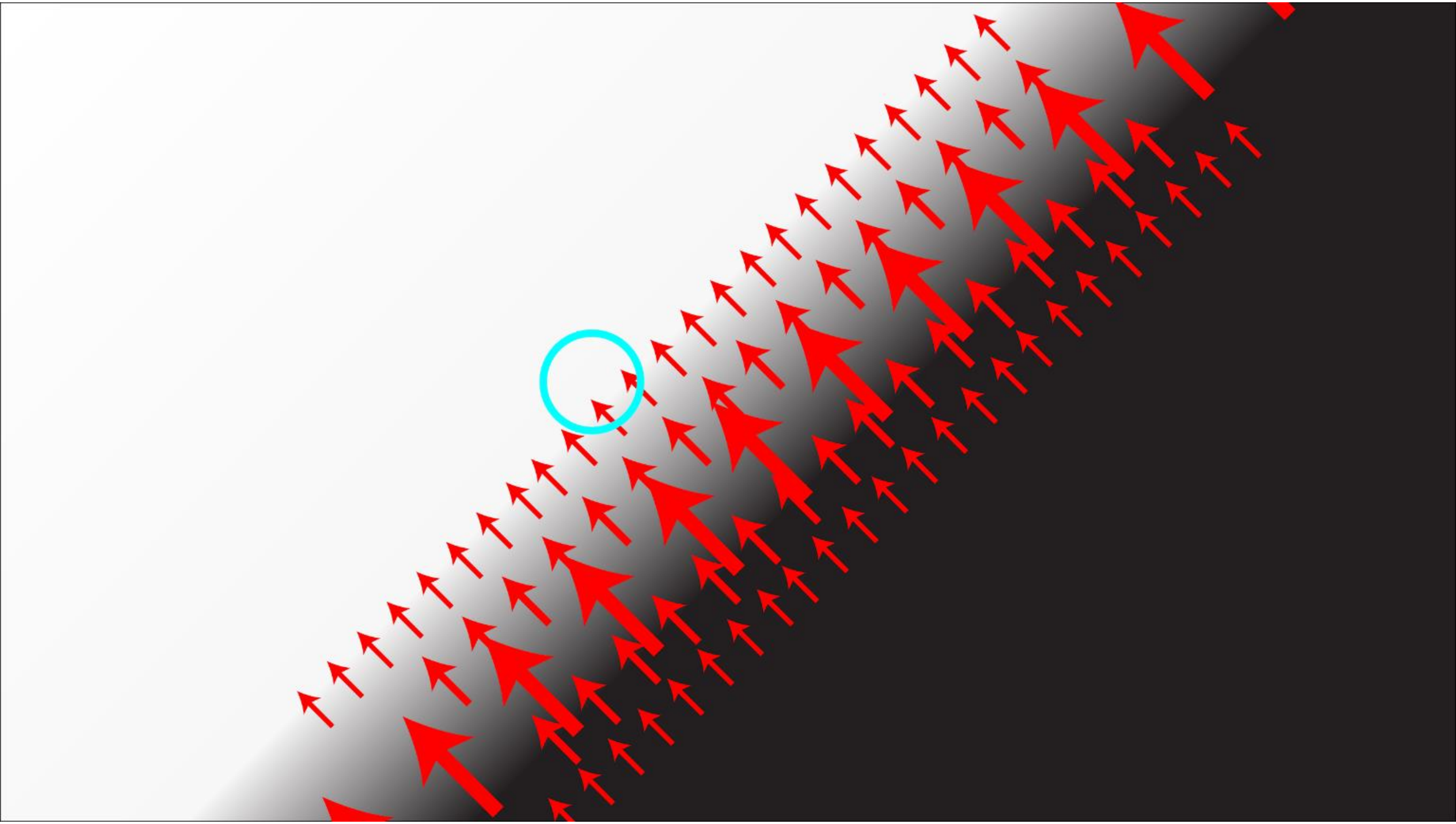
- Laplacian:

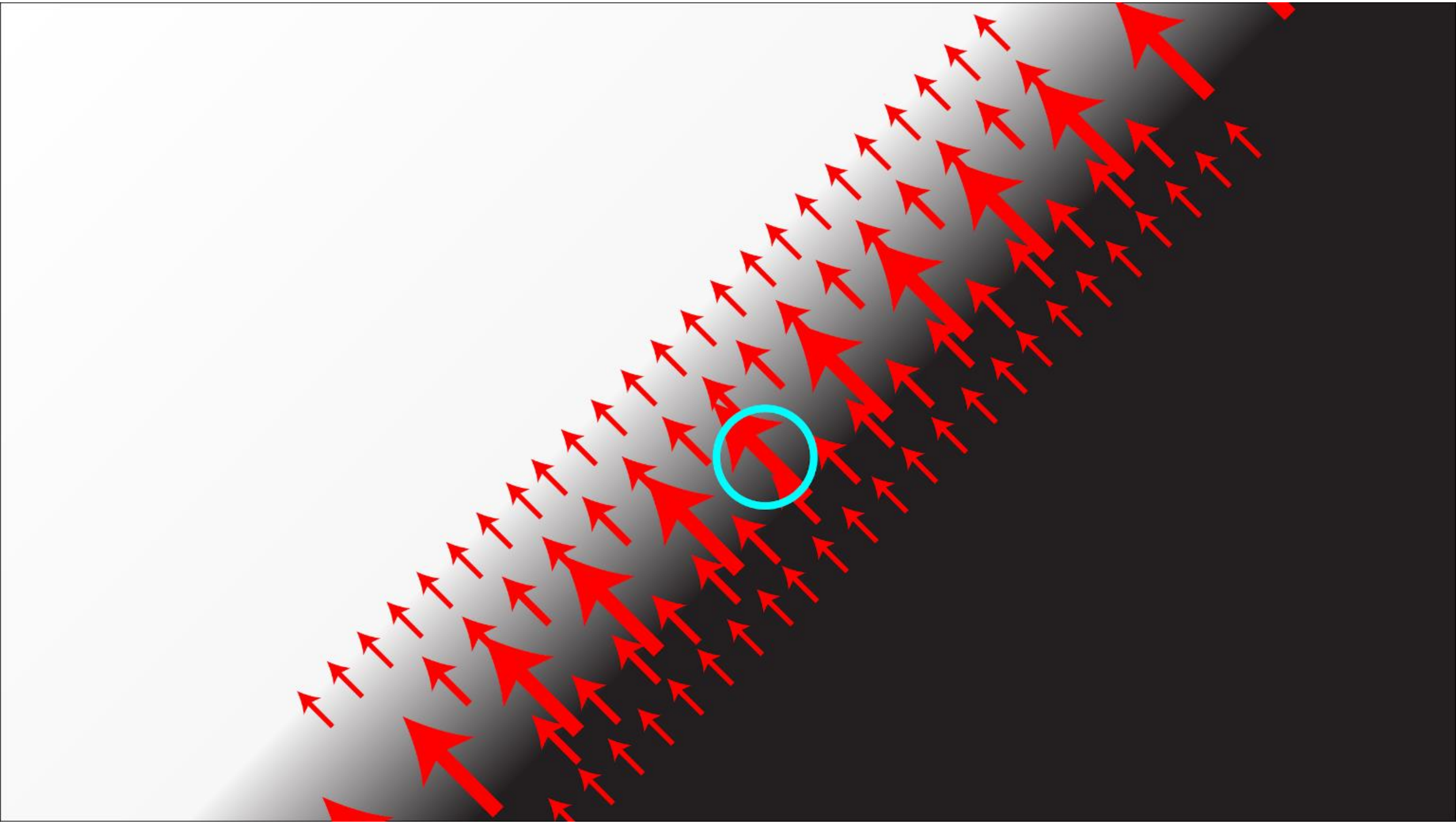
$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

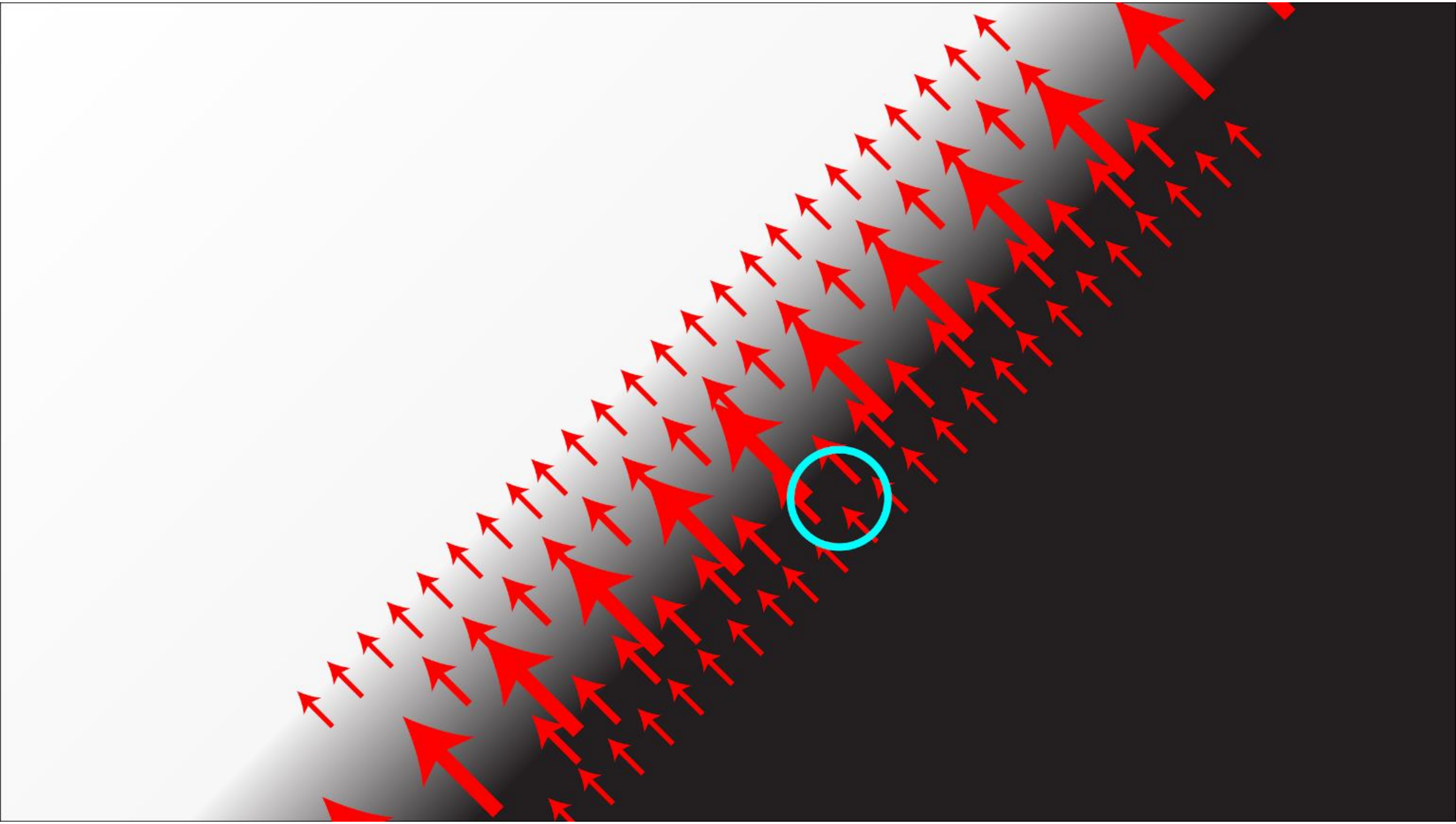
- Measures the divergence of the gradient
    - Flux of gradient vector field through small area





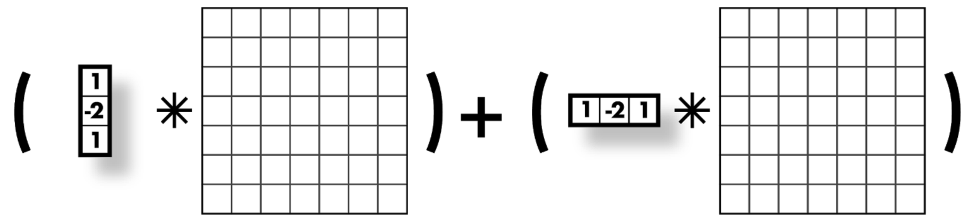






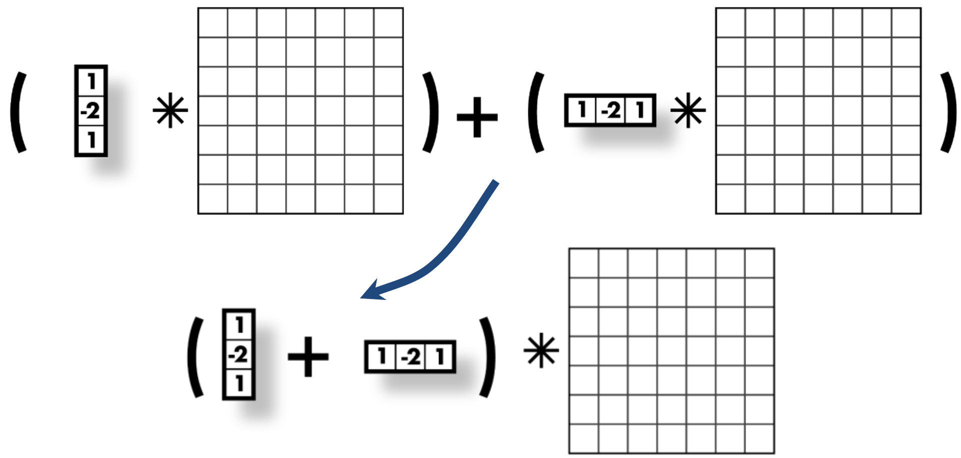
Laplacians

- Laplacian:
  - $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$



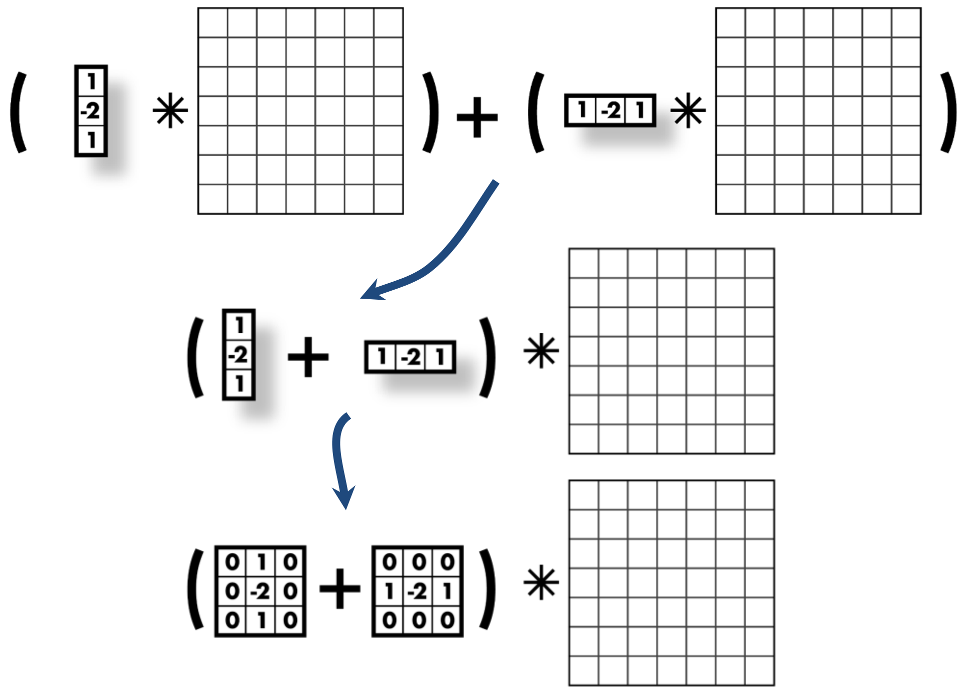
Laplacians

- Laplacian:
  - $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$



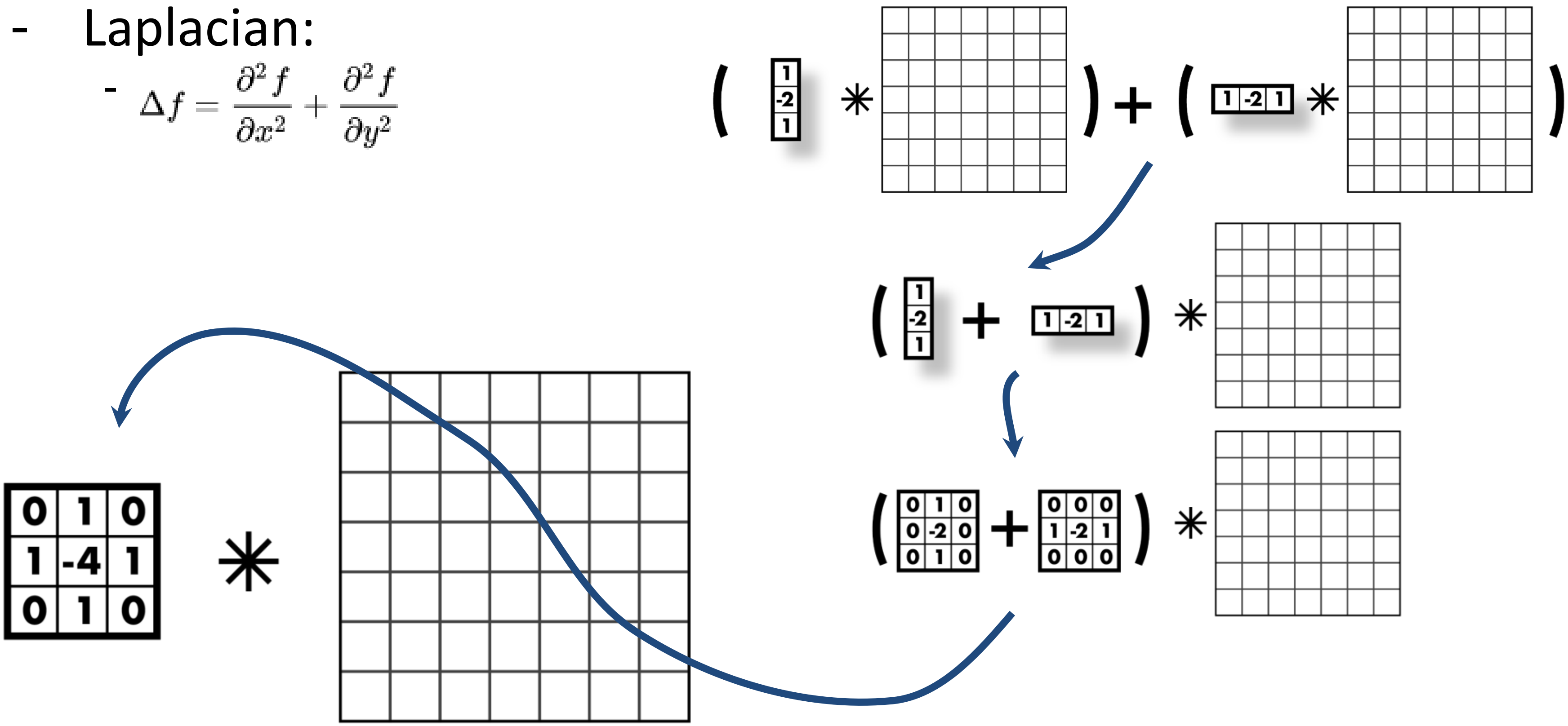
Laplacians

- Laplacian:
  - $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$



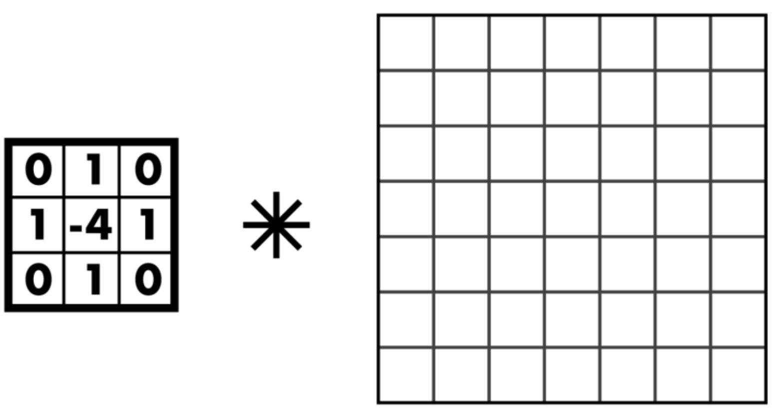
Laplacians

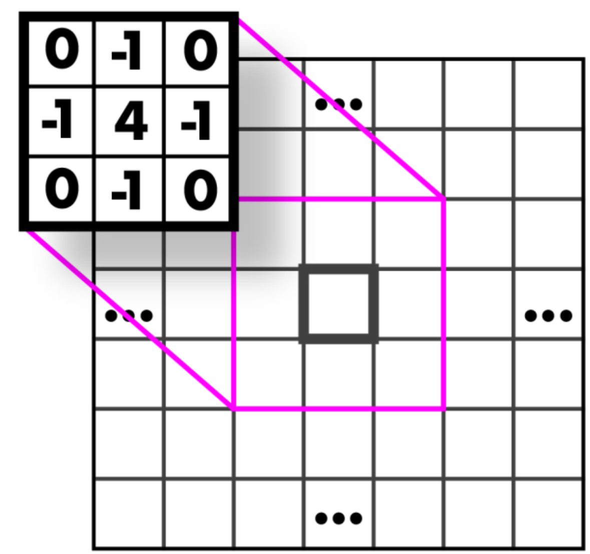
- Laplacian:
  - $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$



Laplacians

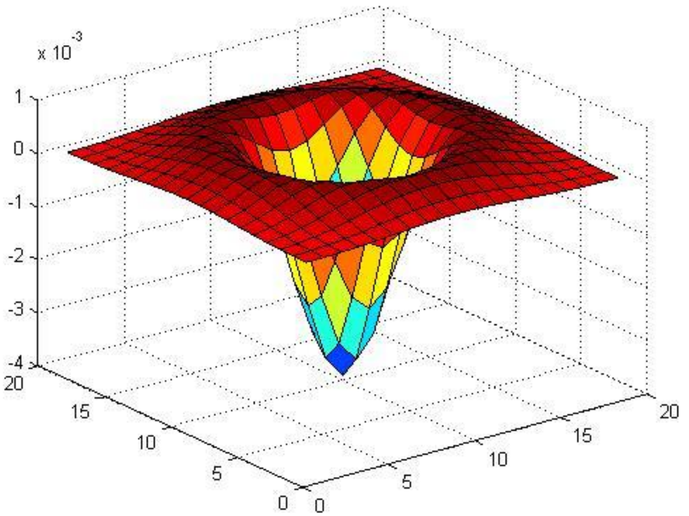
- Laplacian: $$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$
  -
- Negative Laplacian, -4 in middle
- Positive Laplacian --->





Laplacians also sensitive to noise

- Again, use gaussian smoothing
- Can just use one kernel since convs commute
- Laplacian of Gaussian, **LoG**
- Can get good approx. with 5x5 - 9x9 kernels



Another edge detector:

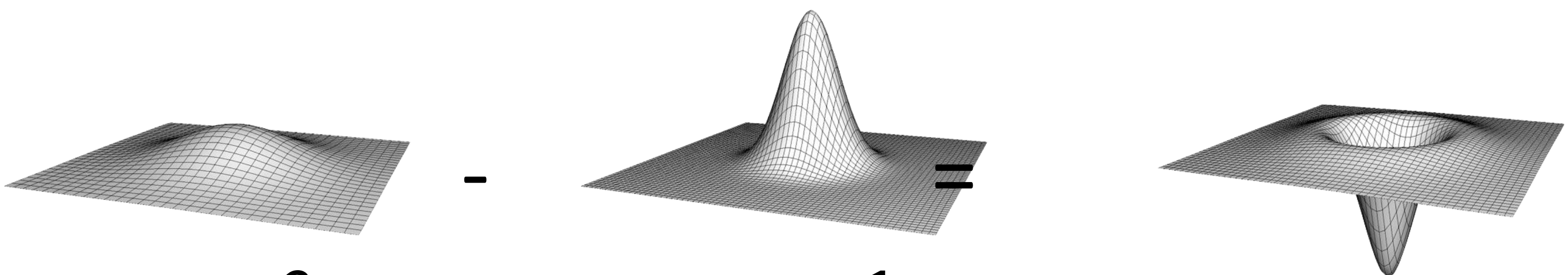
- Image is a function:
  - Has high frequency and low frequency components
  - Think in terms of fourier transform
- Edges are high frequency changes
- Maybe we want to find edges of a specific size (i.e. specific frequency)

Difference of Gaussian (DoG)

- Gaussian is a low pass filter
- Strongly reduce components with frequency $f < \sigma$
- (g*I) low frequency components
- I - (g*I) high frequency components
- $g(\sigma 1)*I - g(\sigma 2)*I$
  - Components in between these frequencies
- $g(\sigma 1)*I - g(\sigma 2)*I =$ **$[g(\sigma 1) - g(\sigma 2)]*I$**

Difference of Gaussian (DoG)

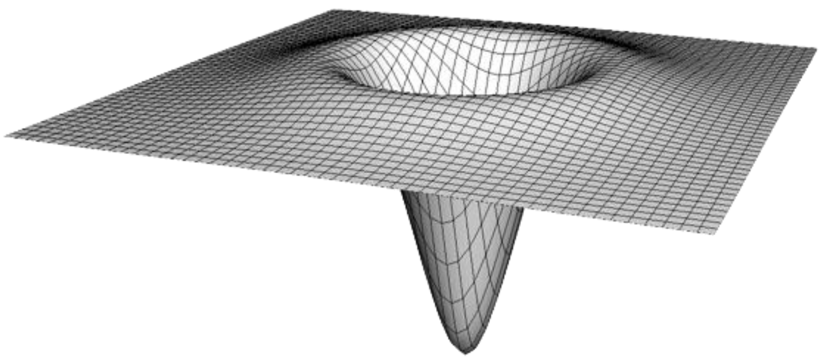
- $g(\sigma 1)*I - g(\sigma 2)*I = [g(\sigma 1) - g(\sigma 2)]*I$

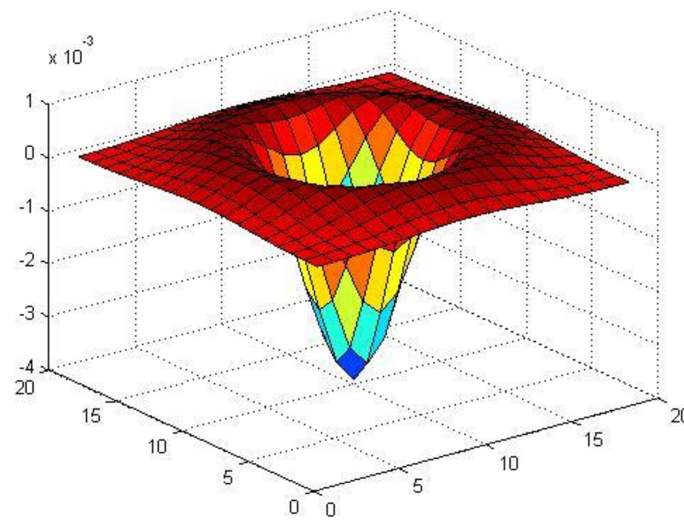


$\sigma = 2$ $\sigma = 1$

Difference of Gaussian (DoG)

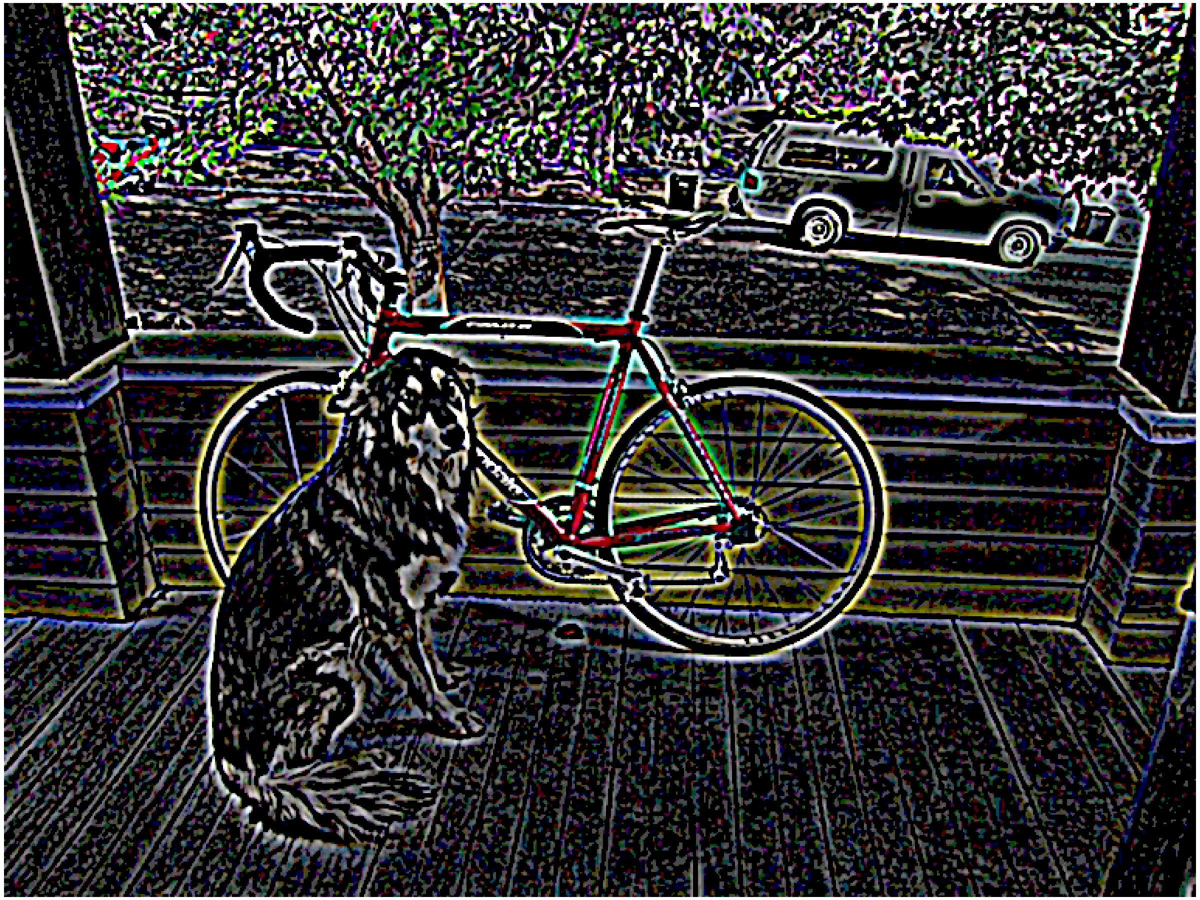
- $g(\sigma 1)*I - g(\sigma 2)*I = [g(\sigma 1) - g(\sigma 2)]*I$
- This looks a lot like our LoG!
- (not actually the same but similar)



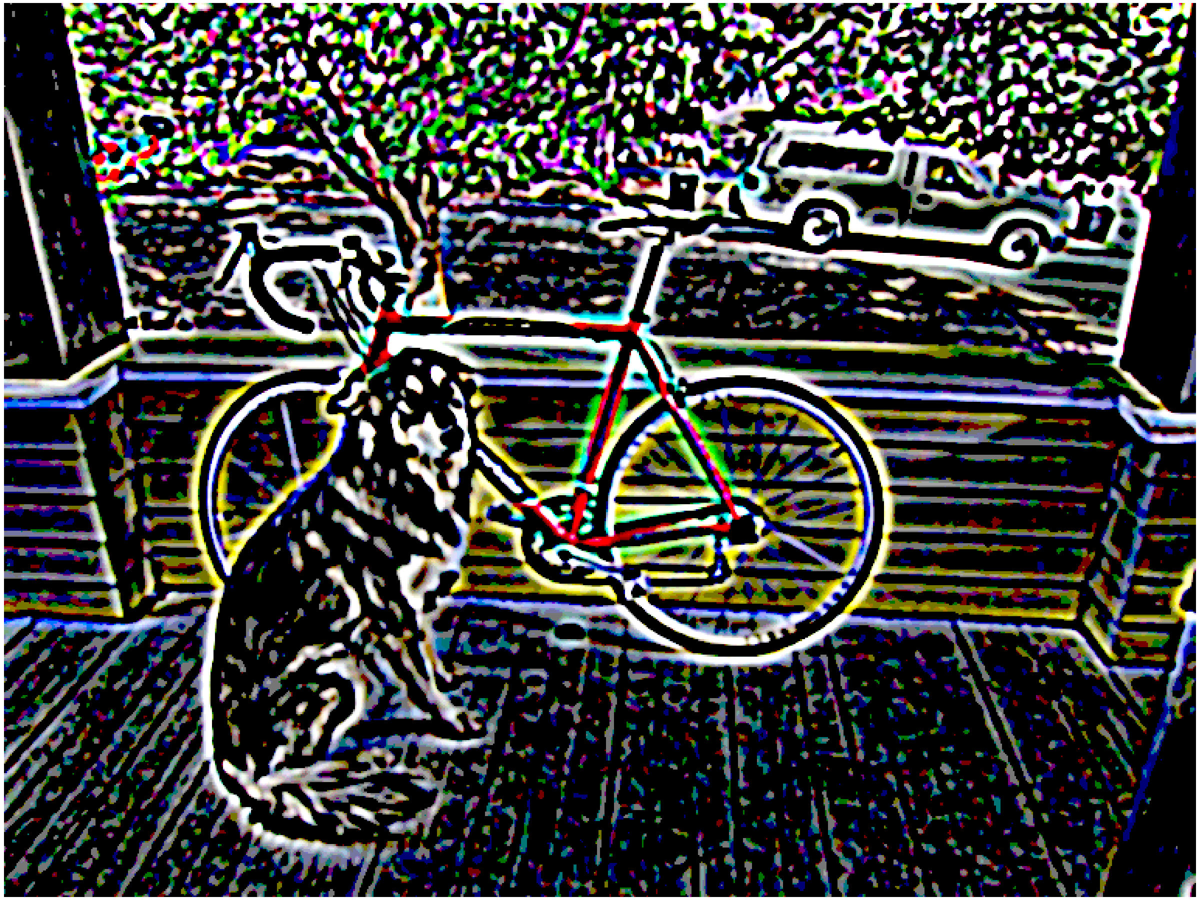


DoG (1 - 0)

DoG (2 - 1)



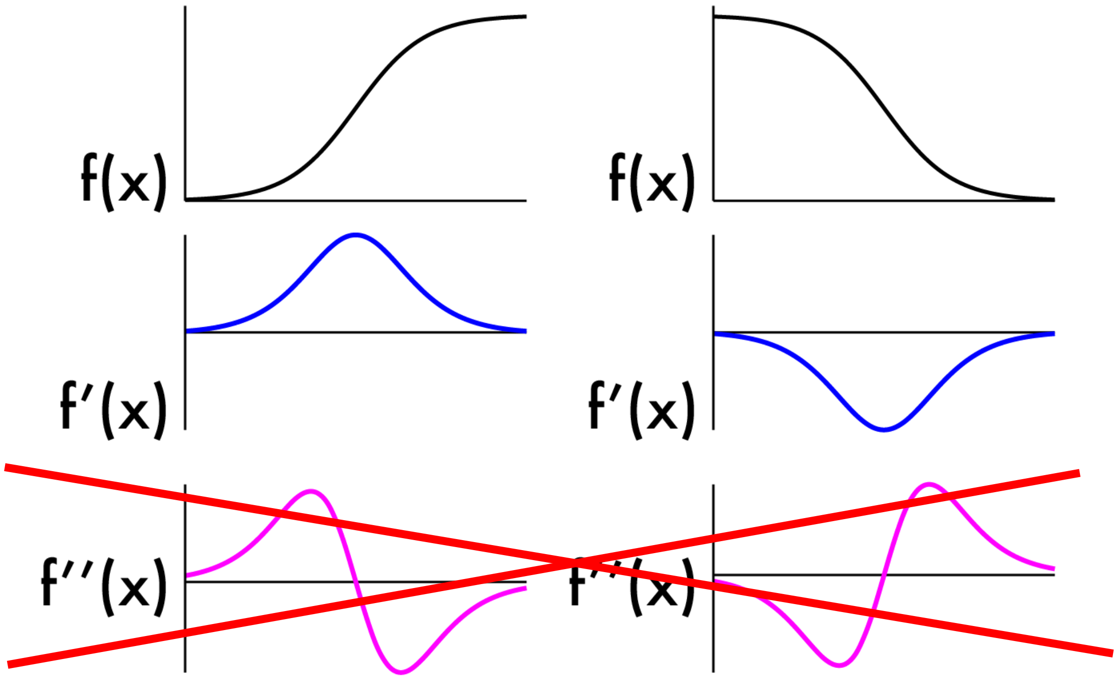
DoG (3 - 2)



DoG (4 - 3)

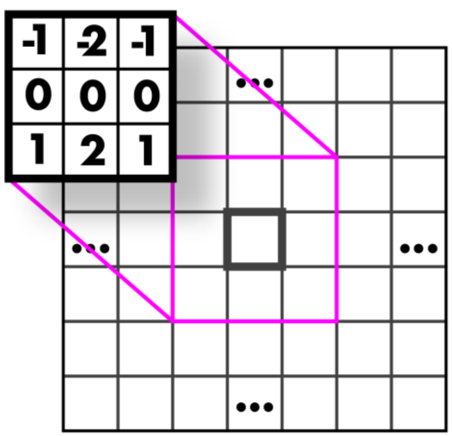
Another approach: gradient magnitude

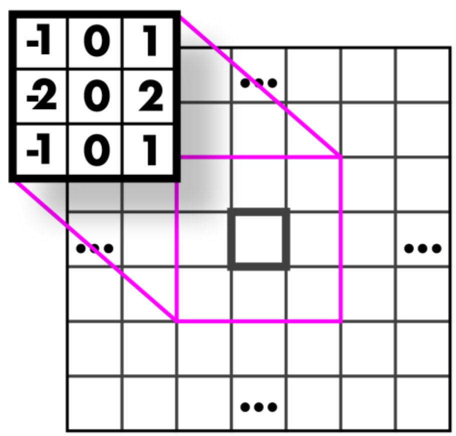
- Don’t need 2nd derivatives
- Just use magnitude of gradient



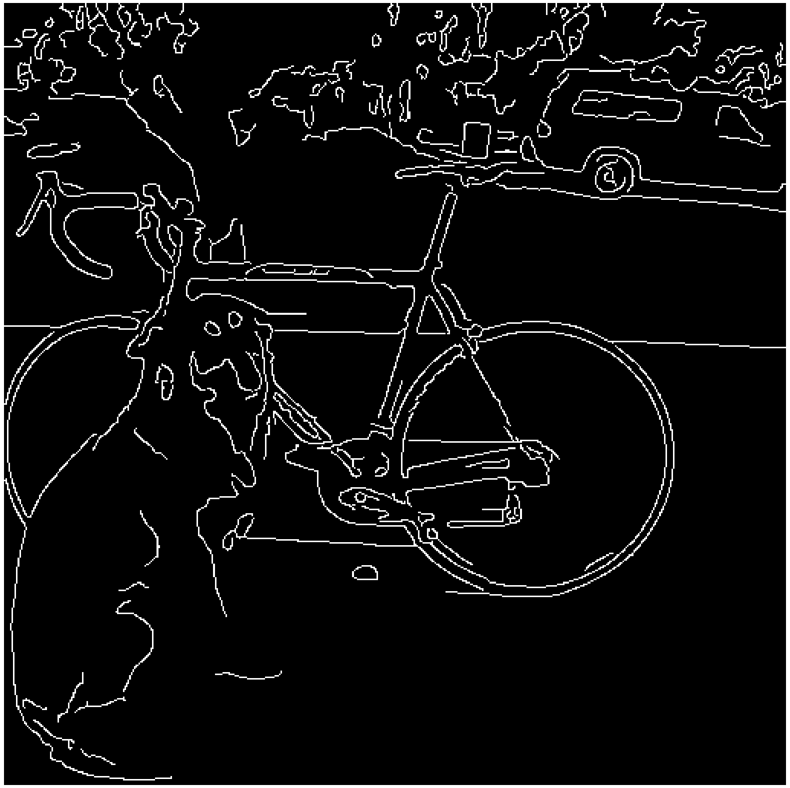
Another approach: gradient magnitude

- Don’t need 2nd derivatives
- Just use magnitude of gradient
- Are we done? No!





What we really want: line drawing



Canny Edge Detection

- Your first image processing pipeline!
  - Old-school CV is all about pipelines

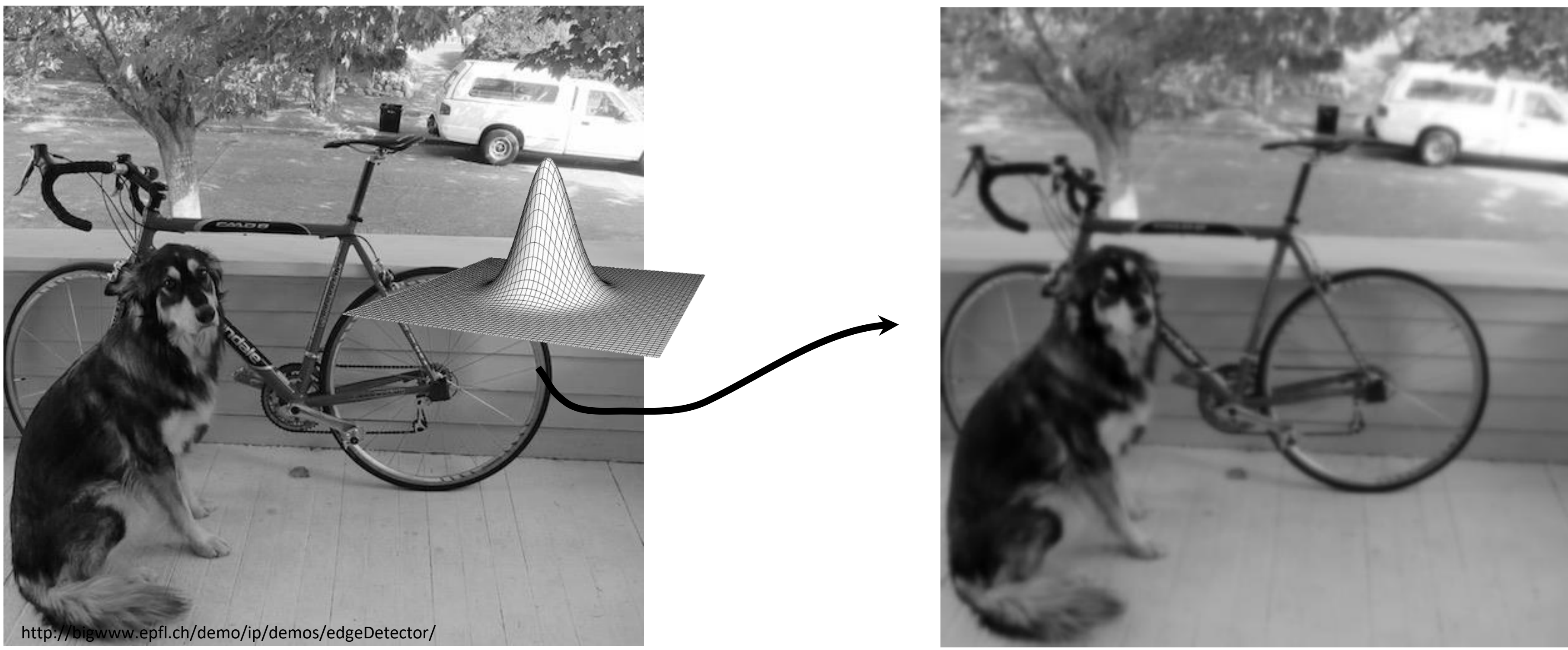
Algorithm:

- 1. Smooth image (only want “real” edges, not noise)
- 2. Calculate gradient direction and magnitude
- 3. Non-maximum suppression perpendicular to edge
- 4. Threshold into strong, weak, no edge
- 5. Connect together components

http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

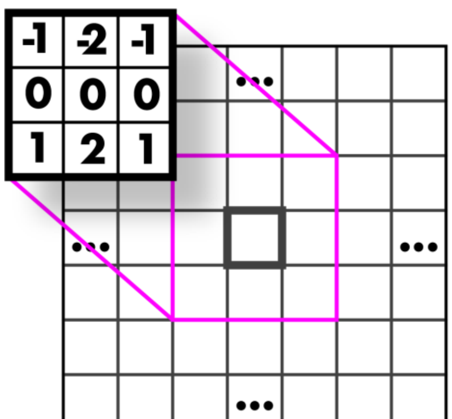
Smooth image

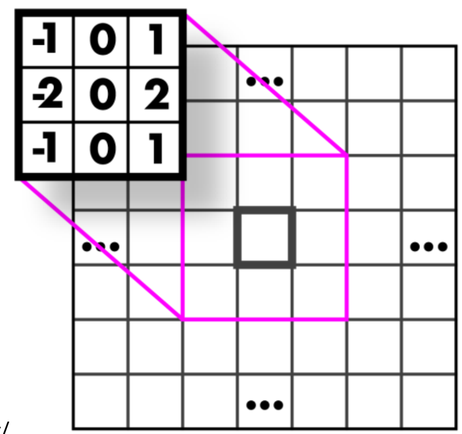
- You know how to do this, gaussians!



Gradient magnitude and direction

- Sobel filter





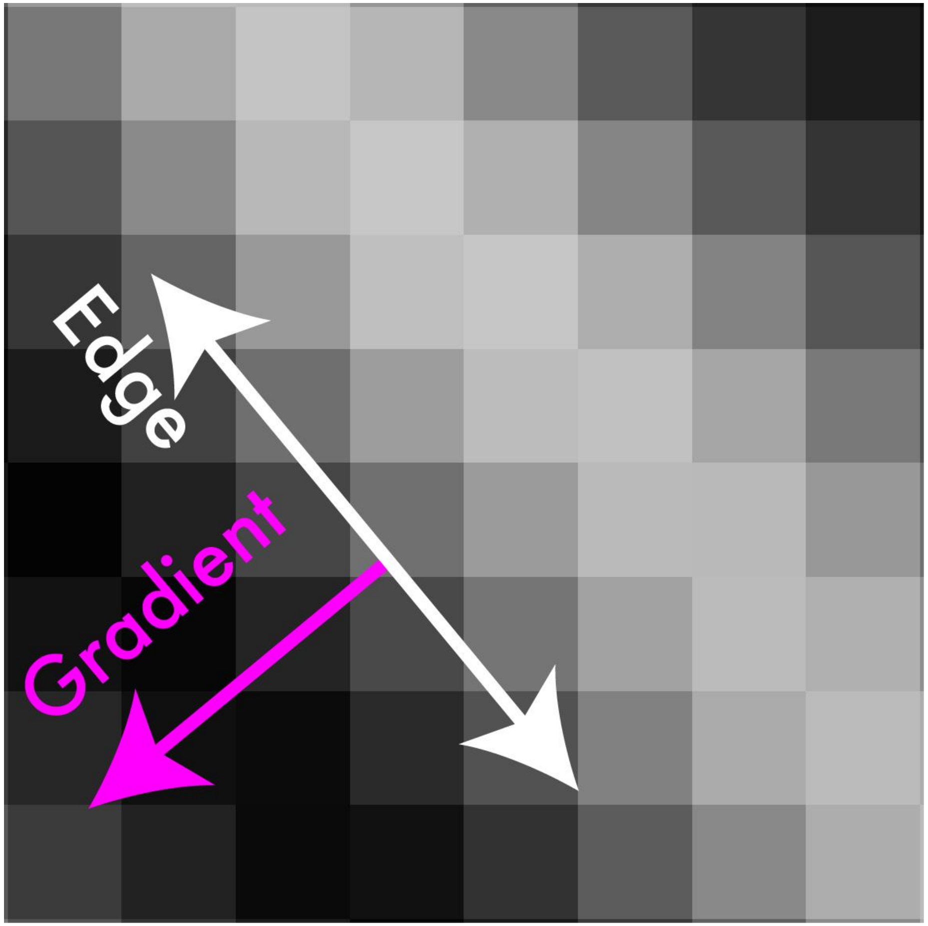
http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

Non-maximum suppression

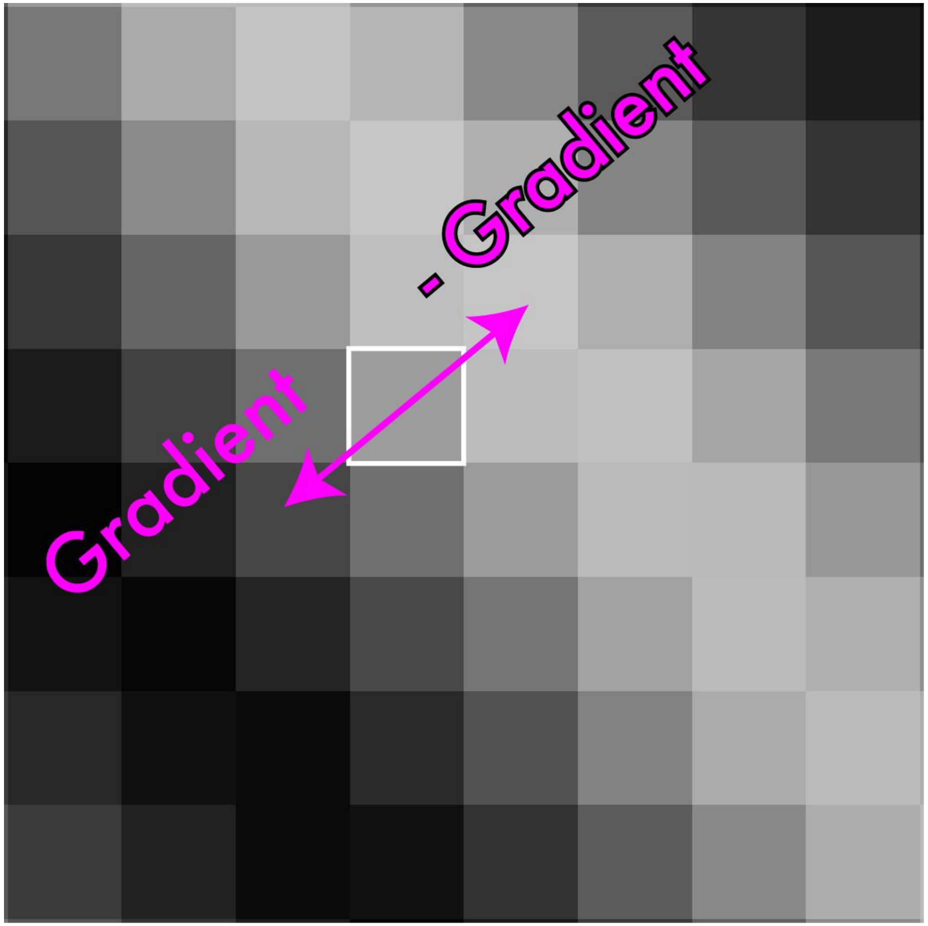
- Want single pixel edges, not thick blurry lines
- Need to check nearby pixels
- See if response is highest

http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

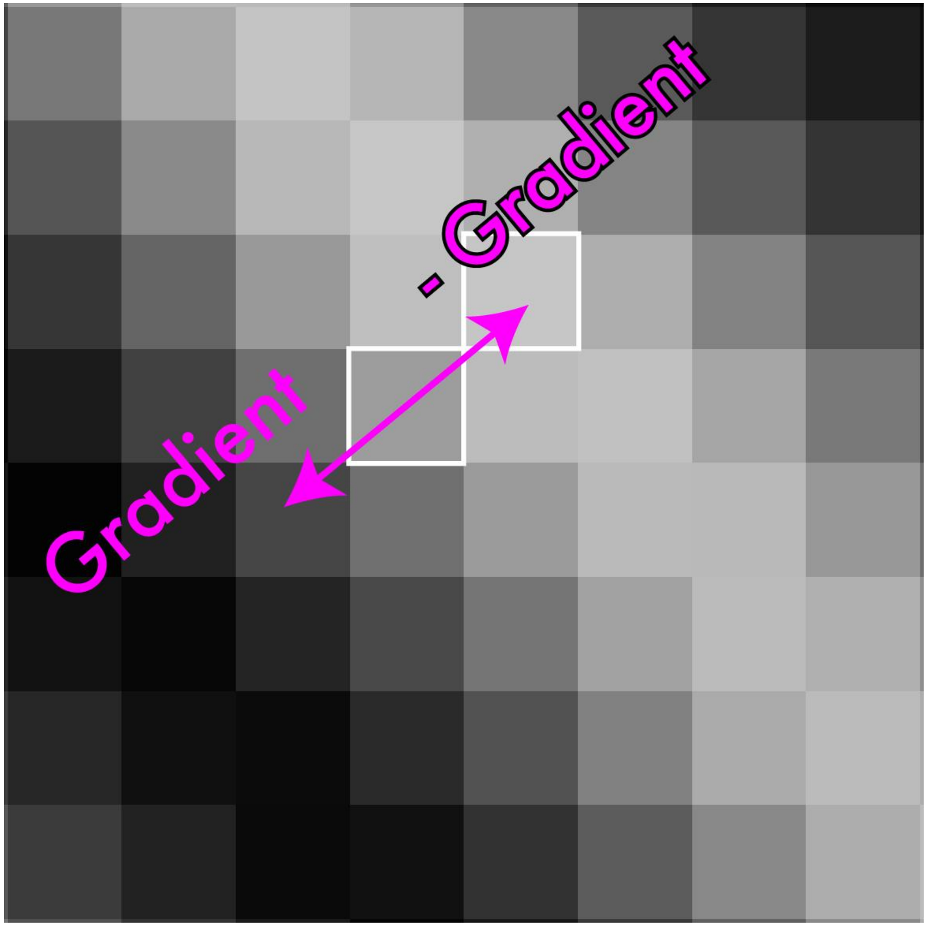
Non-maximum suppression



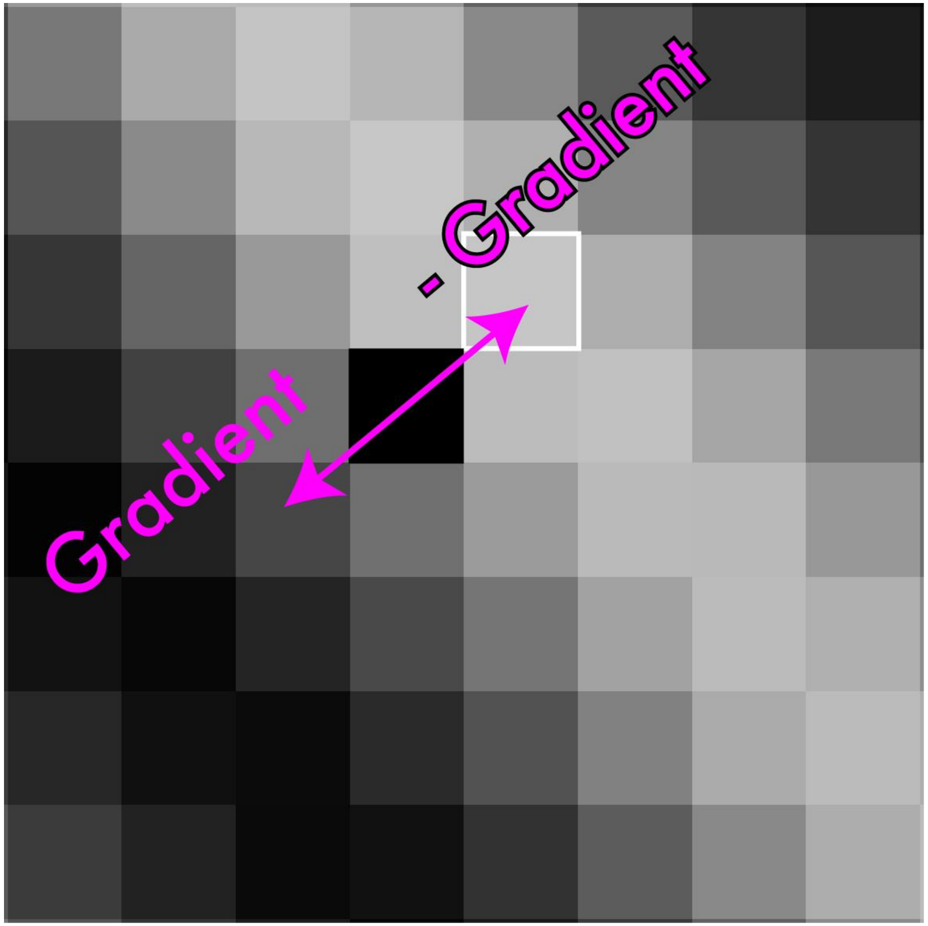
Non-maximum suppression



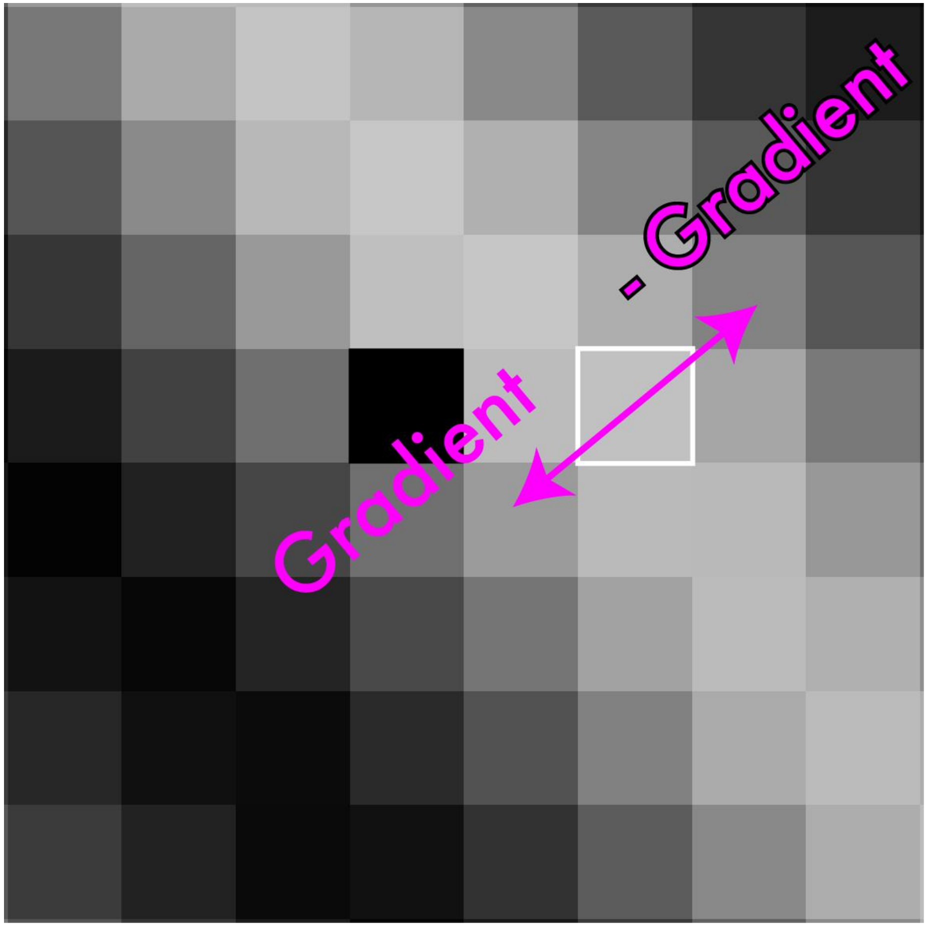
Non-maximum suppression



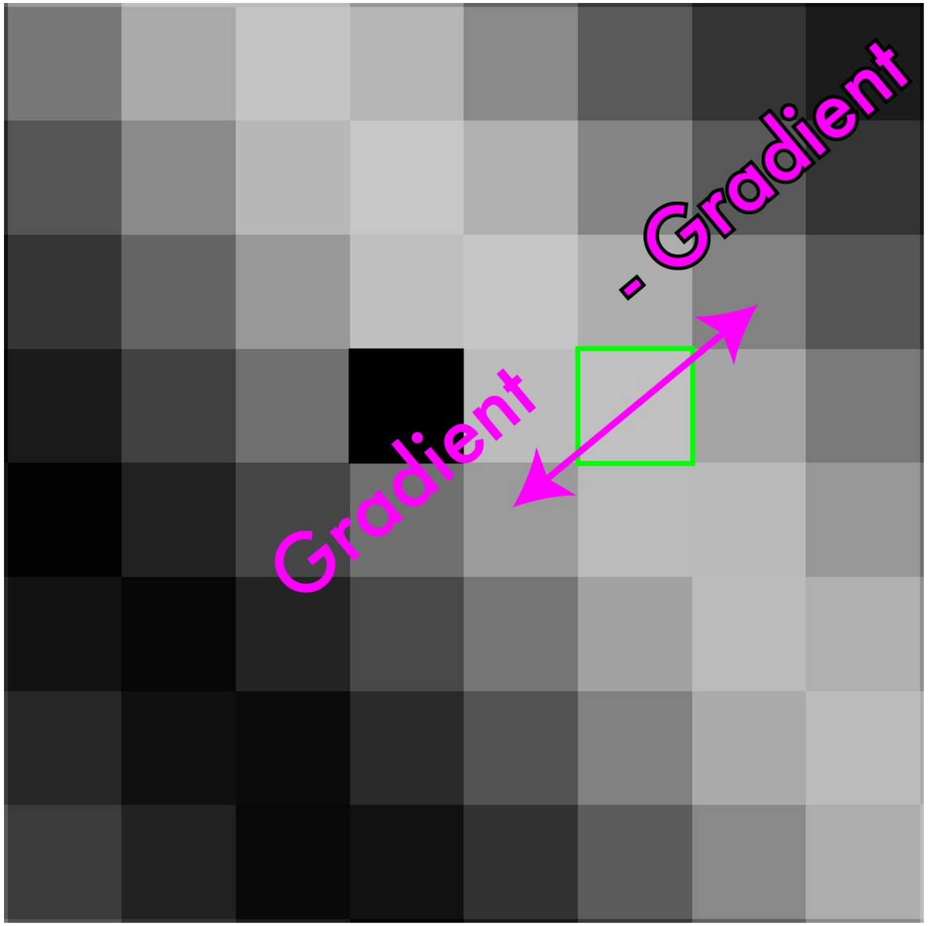
Non-maximum suppression



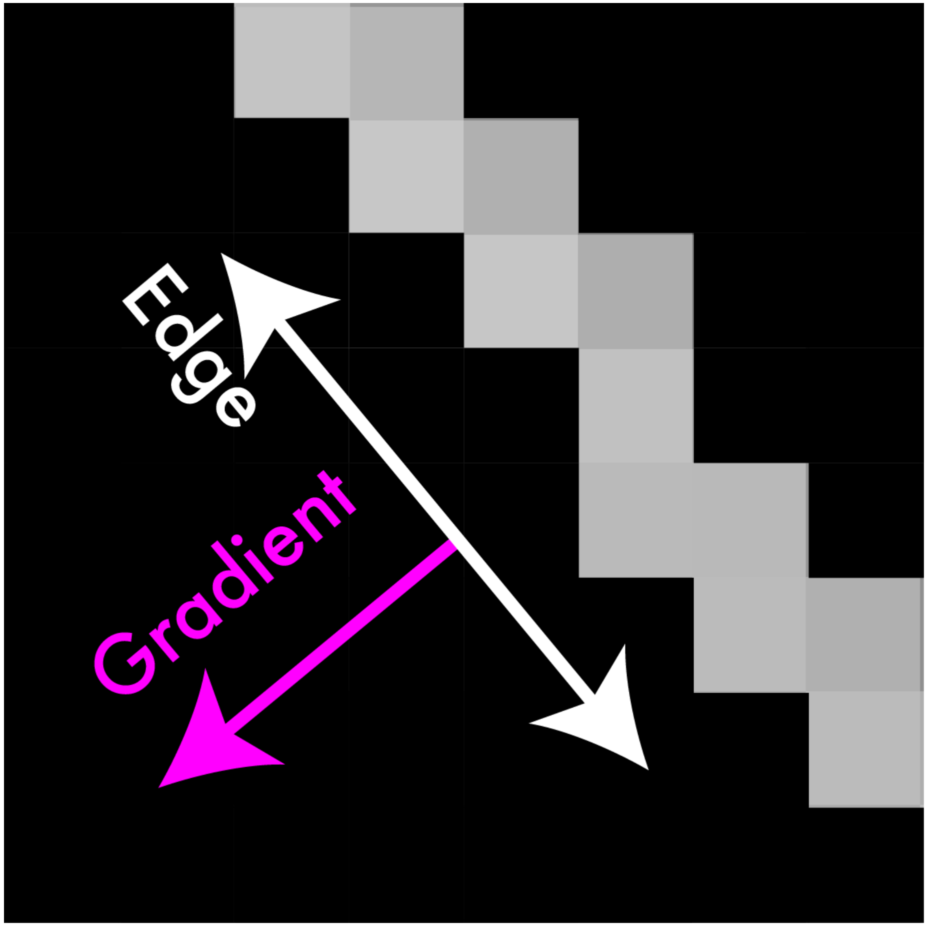
Non-maximum suppression



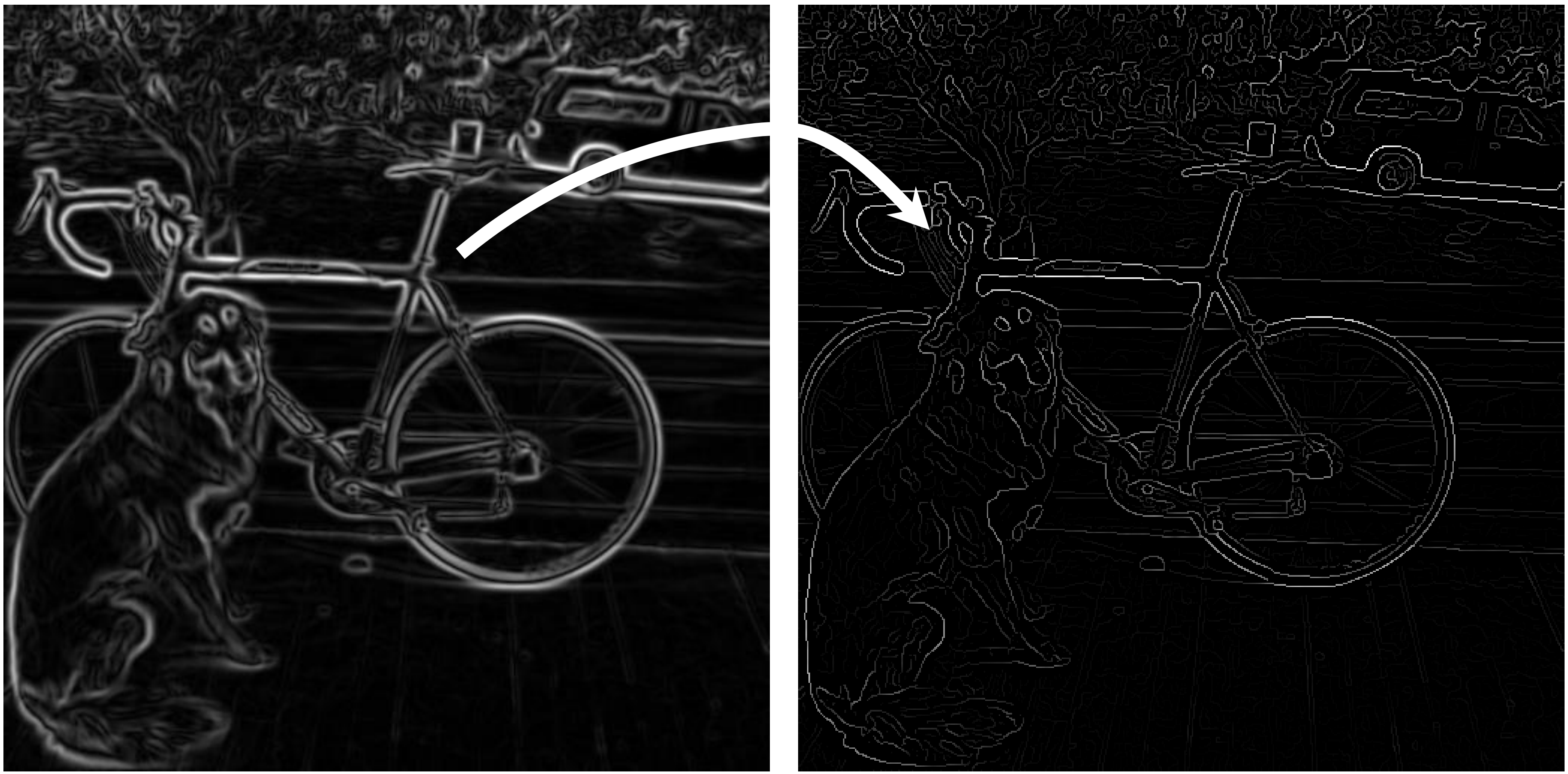
Non-maximum suppression



Non-maximum suppression



Non-maximum suppression



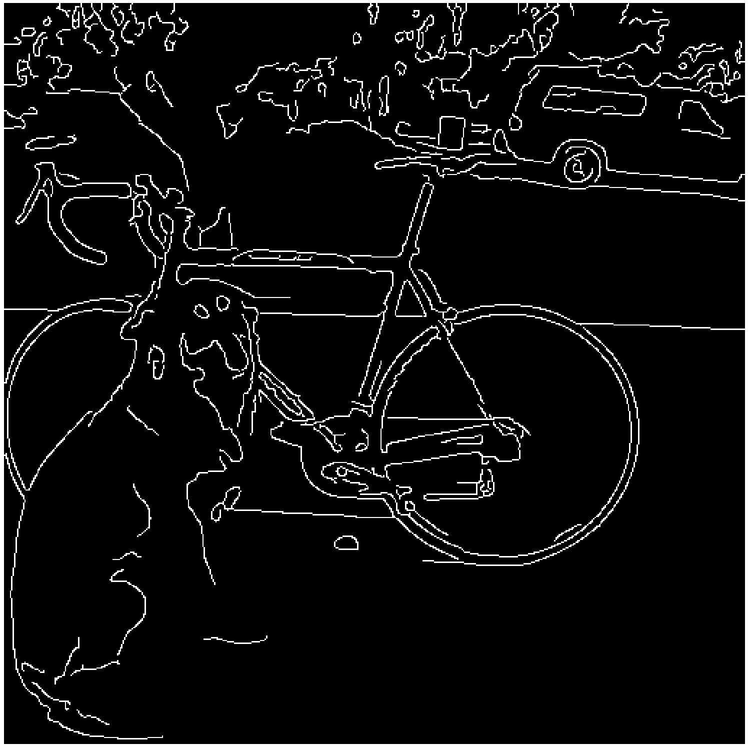
Threshold edges

- Still some noise
- Only want strong edges
- 2 thresholds T and t, 3 cases
  - R > T: strong edge
  - R < T but R > t: weak edge
  - R < t: no edge
- Why two thresholds?

http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

Connect ‘em up!

- Strong edges are edges!
- Weak edges are edges
  iff they connect to strong
- Look in some neighborhood
  (usually 8 closest)



http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

Canny Edge Detection

- Your first image processing pipeline!
  - Old-school CV is all about pipelines

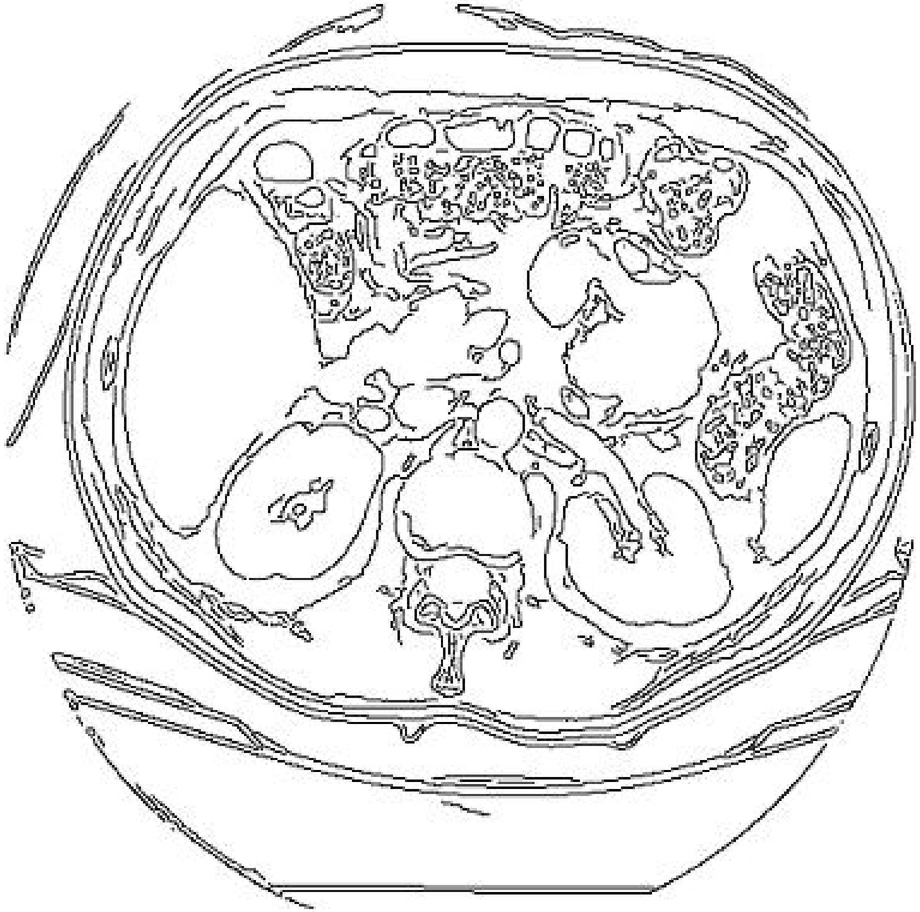
Algorithm:

- Smooth image (only want “real” edges, not noise)
- Calculate gradient direction and magnitude
- Non-maximum suppression perpendicular to edge
- Threshold into strong, weak, no edge
- Connect together components
- Tunable: Sigma, thresholds

http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

Canny Edge Detection

Canny on Kidney



Canny Characteristics

- The Canny operator gives single-pixel-wide images with good continuation between adjacent pixels
- It is the most widely used edge operator today; no one has done better since it came out in the late 80s. Many implementations are available.
- It is very sensitive to its parameters, which need to be adjusted for different application domains.

An edge is not a line...

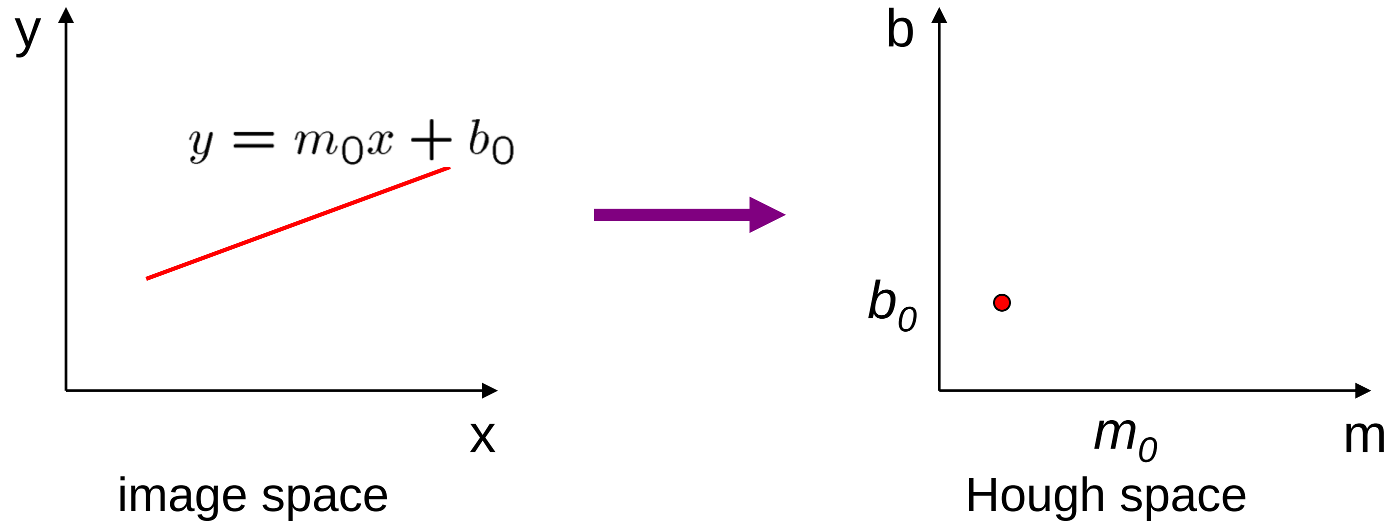
How can we detect ***lines*** ?

Finding lines in an image

- Option 1:
  - Search for the line at every possible position/orientation
  - What is the cost of this operation?

- Option 2:
  - Use a voting scheme: Hough transform

Finding lines in an image



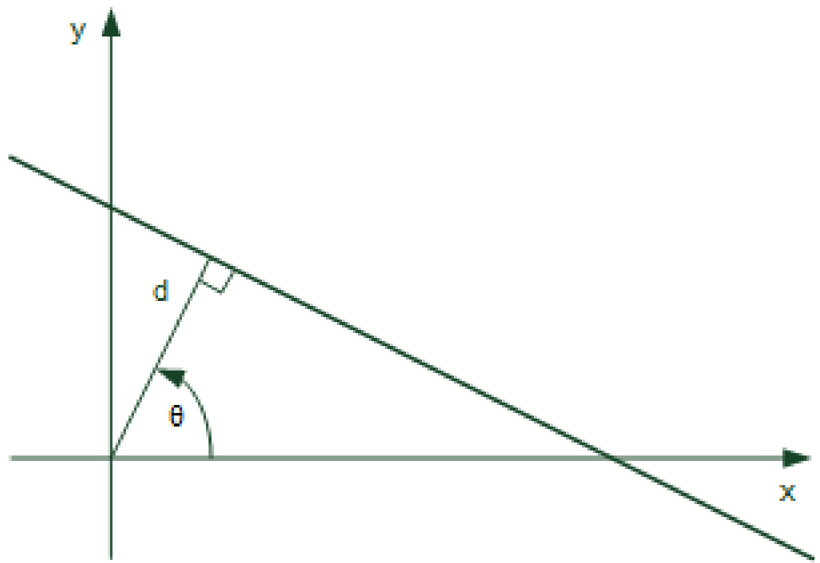
- Connection between image (x,y) and Hough (m,b) spaces
  - A line in the image corresponds to a point in Hough space
  - To go from image space to Hough space:
    - given a set of points (x,y), find all (m,b) such that y = mx + b

Hough transform algorithm

- Typically use a different parameterization

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

  - d is the perpendicular distance from the line to the origin
  - θ is the angle of this perpendicular with the horizontal.



Hough transform algorithm

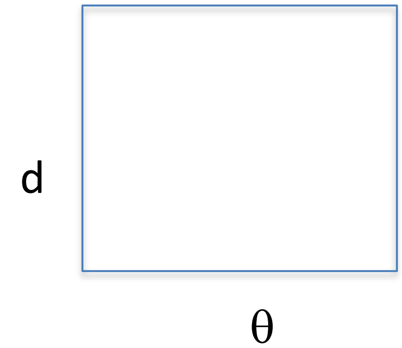
- Basic Hough transform algorithm
  1. Initialize H[d, θ]=0
  2. for each edge point I[x,y] in the image

     compute gradient magnitude m and angle θ

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

     H[d, θ] += 1
  3. Find the value(s) of (d, θ) where H[d, θ] is maximum
  4. The detected line in the image is given by $d = x\cos\theta + y\sin\theta$

Array H



Complexity?

How do you extract the line segments from the accumulators?

pick the bin of H with highest value V
while V > value_threshold {

- order the corresponding pointlist from PTLIST
- merge in high gradient neighbors within 10 degrees
- create line segment from final point list
- zero out that bin of H
- pick the bin of H with highest value V }

Example

gray-tone image

0	0	0	100	100
0	0	0	100	100
0	0	0	100	100
100	100	100	100	100
100	100	100	100	100

DQ

-	-	3	3	-
-	-	3	3	-
3	3	3	3	-
3	3	3	3	-
-	-	-	-	-

THETAQ

-	-	0	0	-
-	-	0	0	-
90	90	40	20	-
90	90	90	40	-
-	-	-	-	-

Accumulator H

distance \ angle	0	10	20	30	40	…	90
360	-	-	-	-	-	-	-
.	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-
3	4	-	1	-	2	-	5
0	-	-	-	-	-	-	-

PTLIST

360	-	-	-	-	-	-	-
.	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-
3	*	-	*	-	*	-	*
0	-	-	-	-	-	-	-

(3,1)
(3,2)
(4,1)
(4,2)
(4,3)

(1,3)(1,4)(2,3)(2,4)

Line segments from Hough Transform

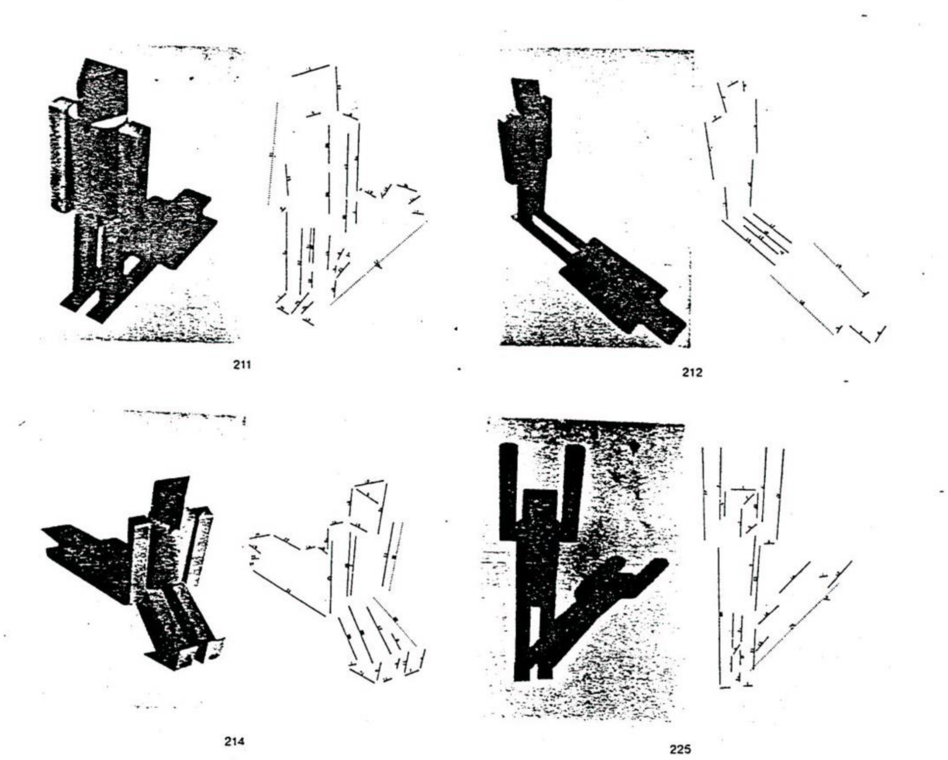


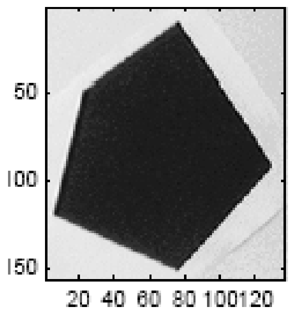
Fig.7. Puppet scenes 211, 212, 214, 225 and the edges recovered by the algorithm.

Extensions

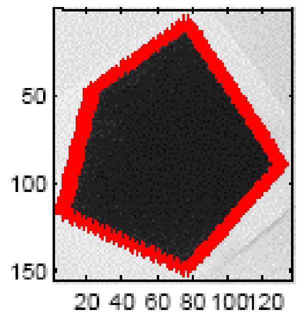
- Extension 1: Use the image gradient (we just did that)

- Extension 2
  - give more votes for stronger edges
- Extension 3
  - change the sampling of (d, θ) to give more/less resolution
- Extension 4
  - The same procedure can be used with circles, squares, or any other shape, How?
- Extension 5; the Burns procedure. Uses only angle, two different quantifications, and connected components with votes for larger one.

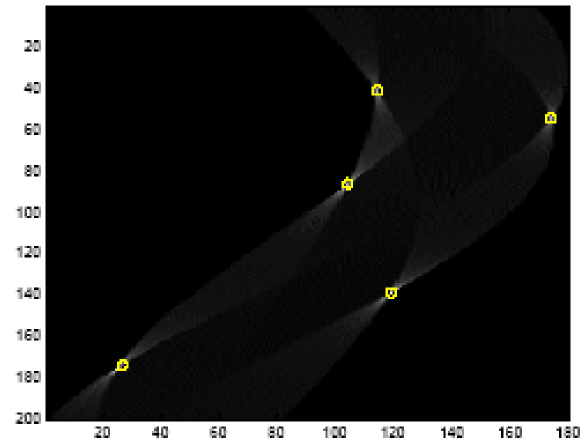
Examples



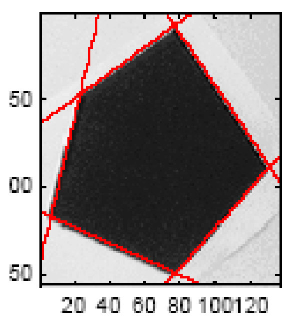
Original Image



Detected Edges



The vote histogram with the detected lines marked with 'o'



Detected lines

Examples cont

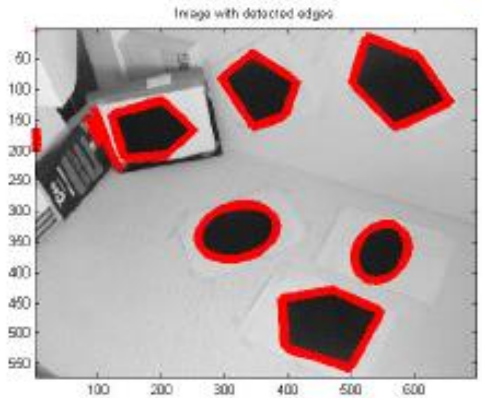
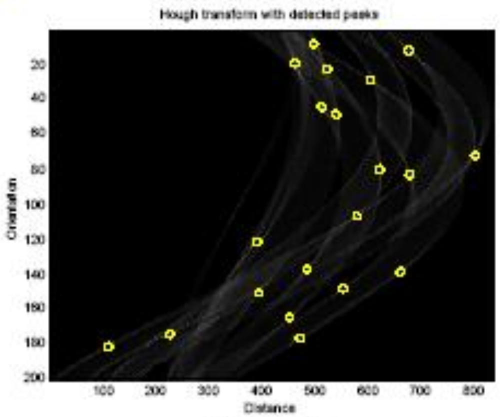


Image with detected edges



The vote histogram with the selected lines

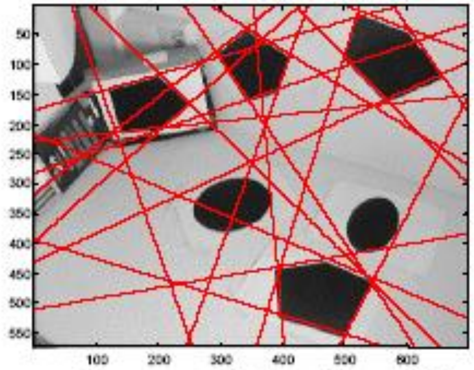
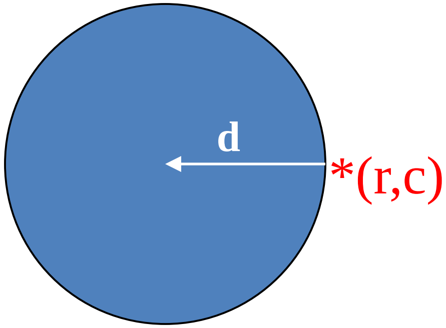


Image with the detected lines

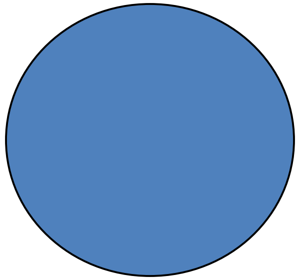
Hough Transform for Finding Circles

Equations: $\boxed{\begin{array}{l} r = r0 + \ d \sin \theta \\ c = c0 - \ d \cos \theta \end{array}}$ r, c, d are parameters

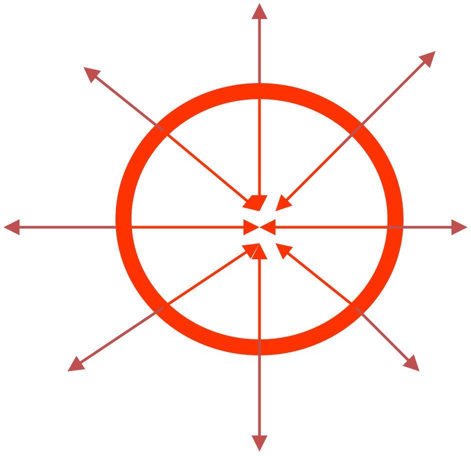
Main idea: The gradient vector at an edge pixel points to the center of the circle.



Why it works

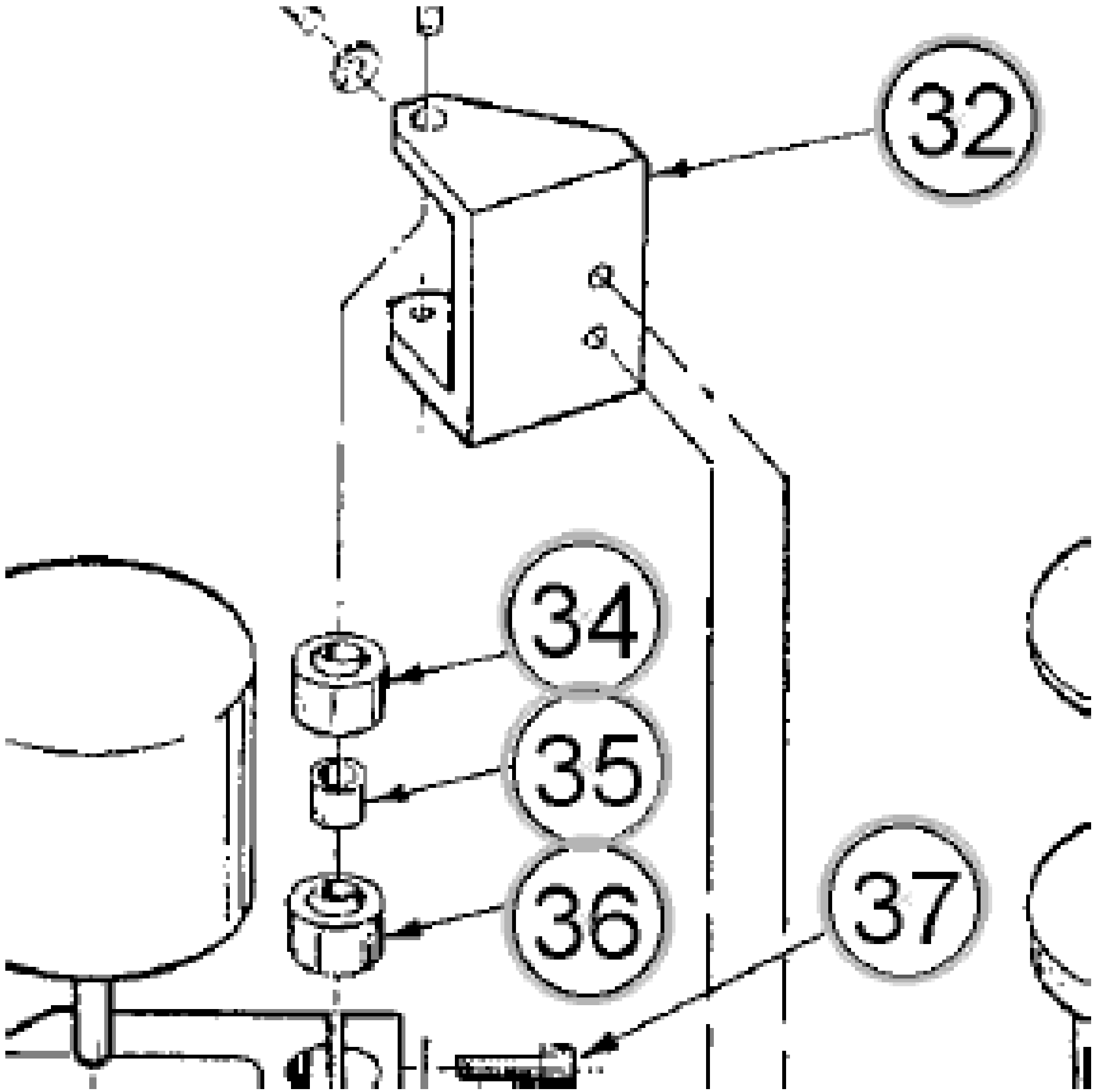


Filled Circle:
Outer points of circle have gradient direction pointing to center.



Circular Ring:
Outer points gradient towards center.
Inner points gradient away from center.

The points in the away direction don't accumulate in one bin!


32
34
35
36
37

Finding lung nodules (Kimme & Ballard)

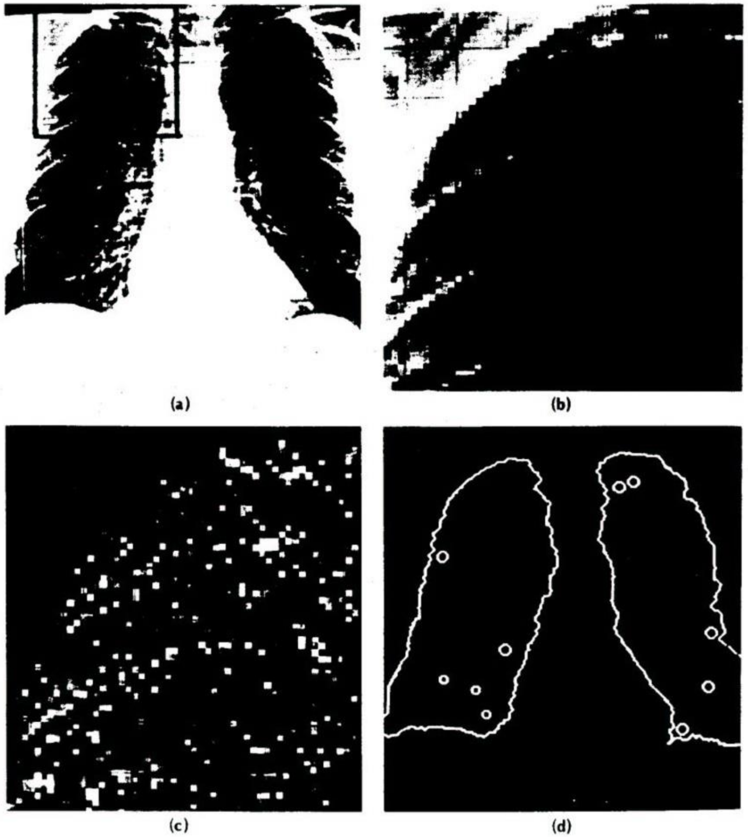


Fig. 4.7 Using the Hough technique for circular shapes. (a) Radiograph. (b) Window. (c) Accumulator array for $r = 3$. (d) Results of maxima detection.

Finale

- Edge operators are based on estimating derivatives.
- While first derivatives show approximately where the edges are, zero crossings of second derivatives were shown to be better.
- Ignoring that entirely, Canny developed his own edge detector that everyone uses now.
- After finding good edges, we have to group them into lines, circles, curves, etc. to use further.
- The Hough transform for circles works well, but for lines the performance can be poor. The Burns operator or some tracking operators (old ORT pkg) work better.