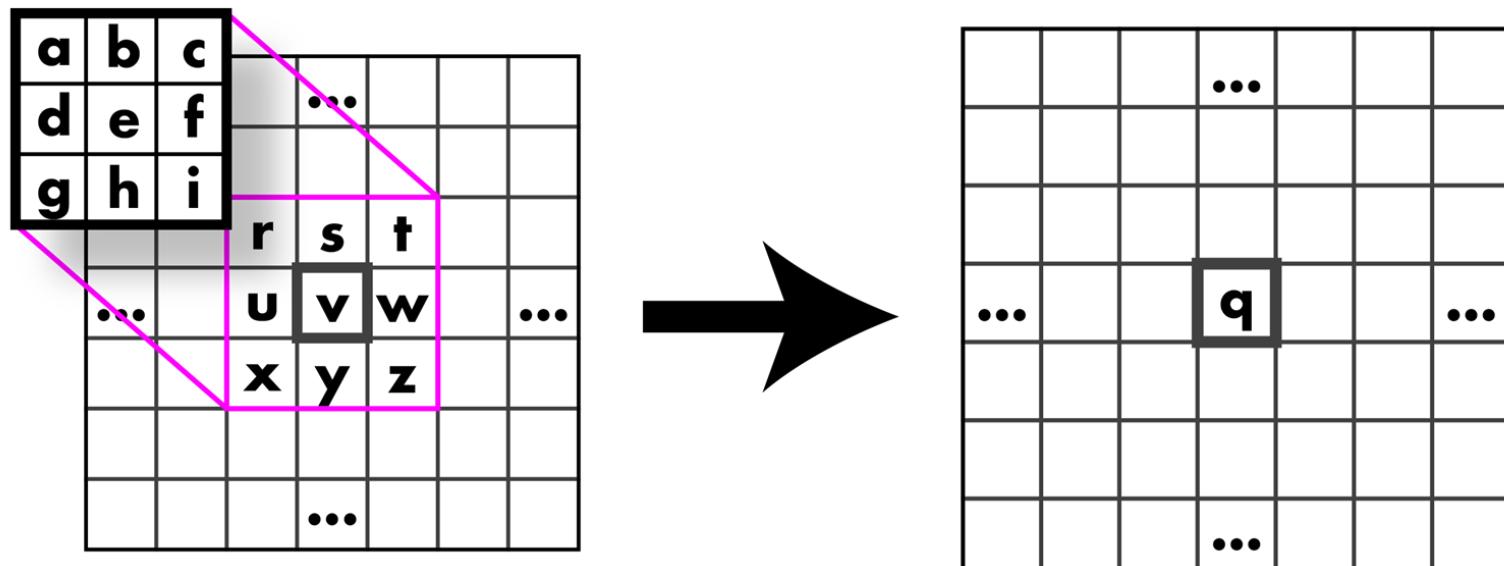


# Digital Image Processing Computer Vision

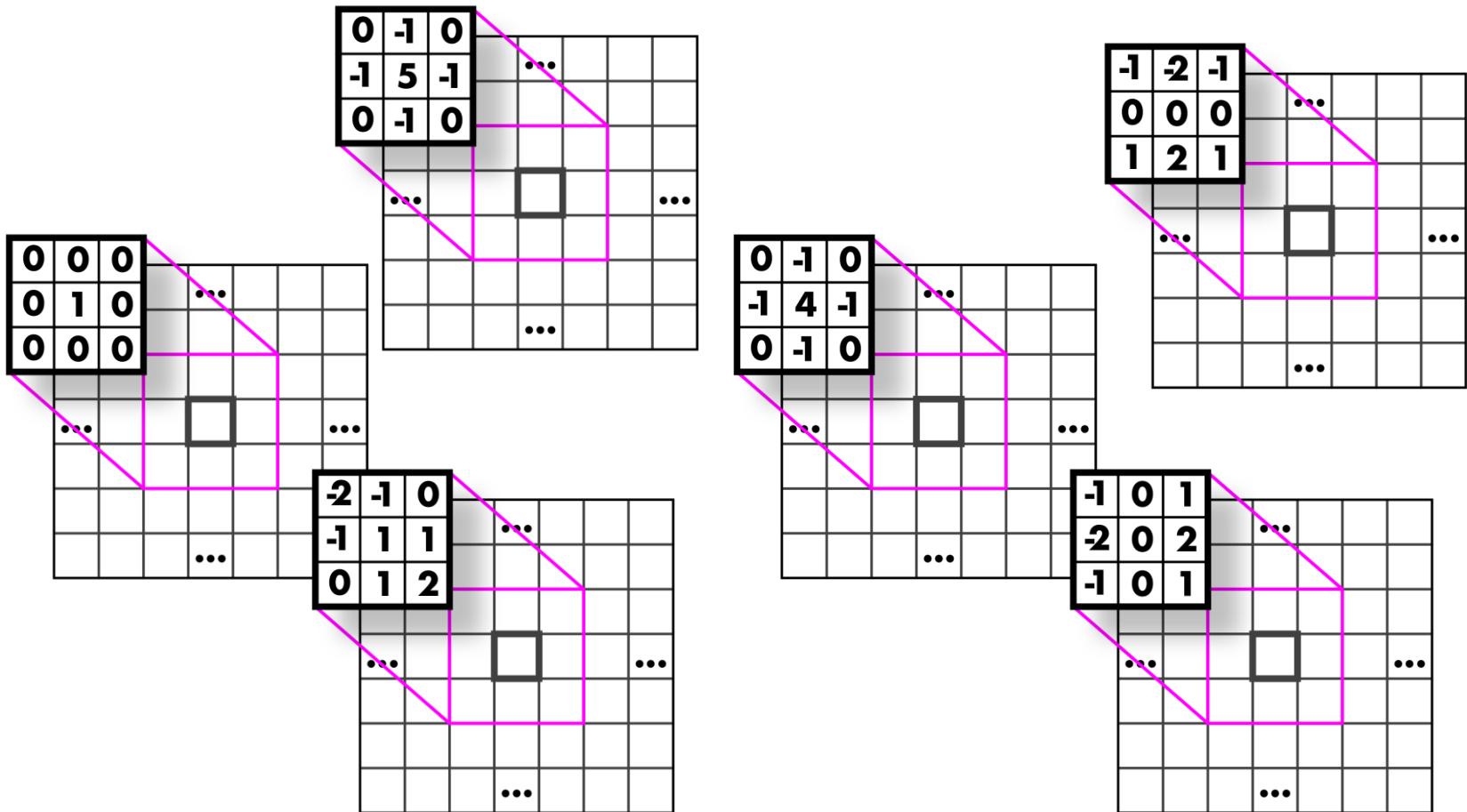
## Edges and Lines

# Convolution: Weighted sum over pixels



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

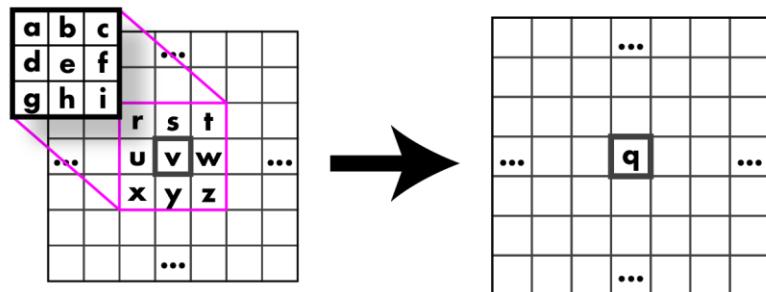
# Filters



# Cross-Correlation vs Convolution

## Cross-Correlation

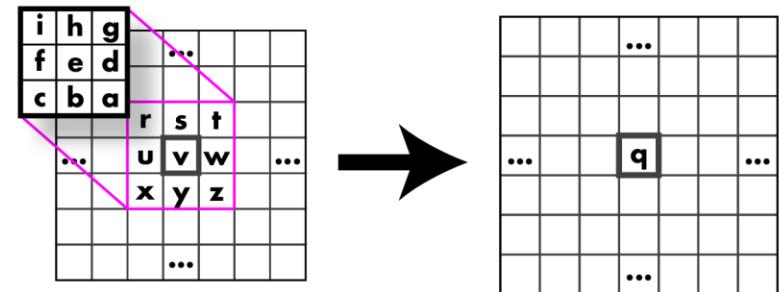
$$\left( \begin{array}{ccccc} a & b & c & & \\ d & e & f & & \\ g & h & i & & \end{array} \right) \star \left( \begin{array}{ccccccc} & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{array} \right)$$



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

## Convolution

$$\left( \begin{array}{ccccc} a & b & c & & \\ d & e & f & & \\ g & h & i & & \end{array} \right) * \left( \begin{array}{ccccccc} & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{array} \right)$$



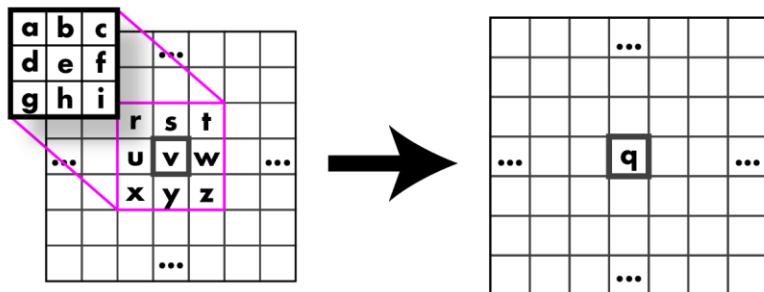
$$q = i \times r + h \times s + g \times t + f \times u + e \times v + d \times w + c \times x + b \times y + a \times z$$

# Cross-Correlation vs Convolution

Do this in HW!

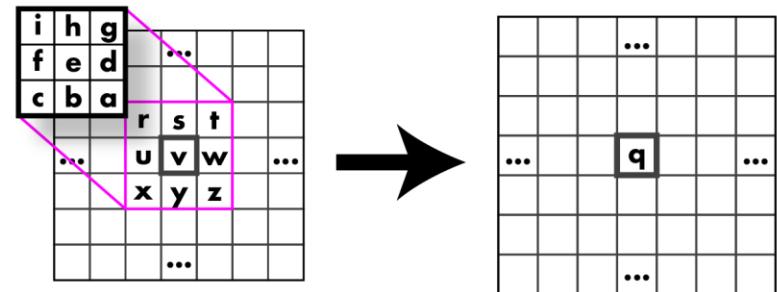
## Cross-Correlation

$$\left( \begin{array}{ccccc} a & b & c & & \\ d & e & f & & \\ g & h & i & & \end{array} \right) \star \left( \begin{array}{ccccccccc} & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{array} \right)$$



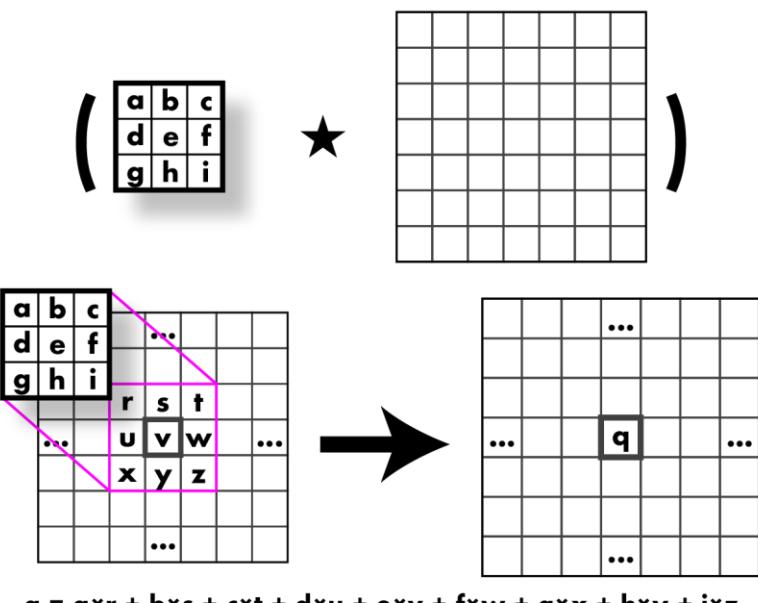
## Convolution

$$\left( \begin{array}{ccccc} a & b & c & & \\ d & e & f & & \\ g & h & i & & \end{array} \right) * \left( \begin{array}{ccccccccc} & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \\ & & & & & & & & \end{array} \right)$$



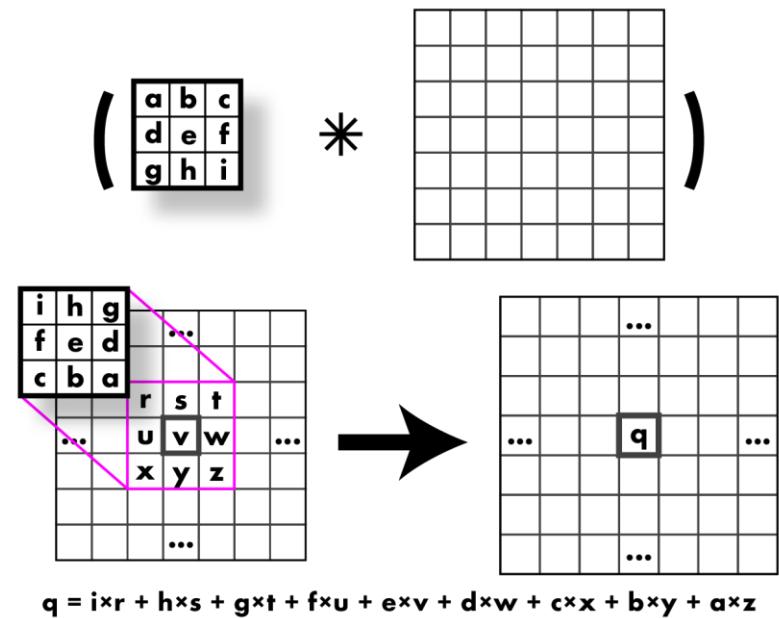
# Cross-Correlation vs Convolution

## Cross-Correlation



These come from signal processing and have nice mathematical properties.

## Convolution



---

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
- ...

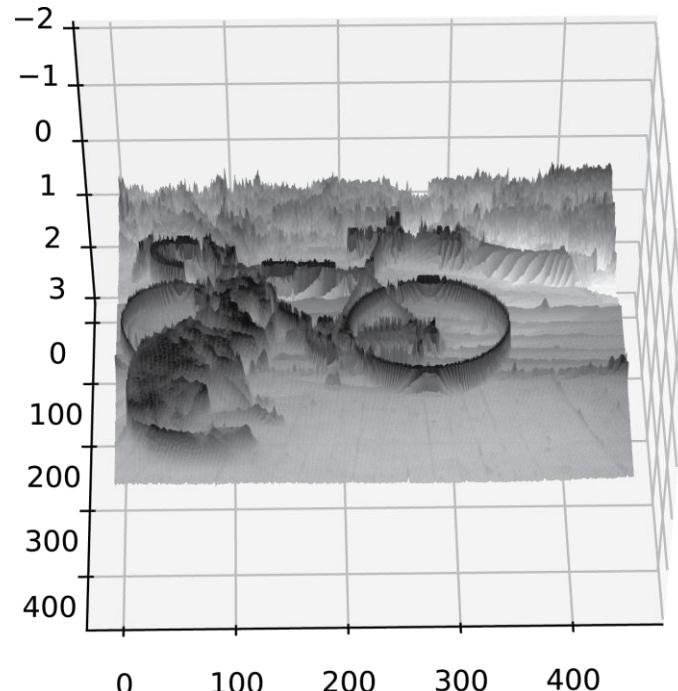
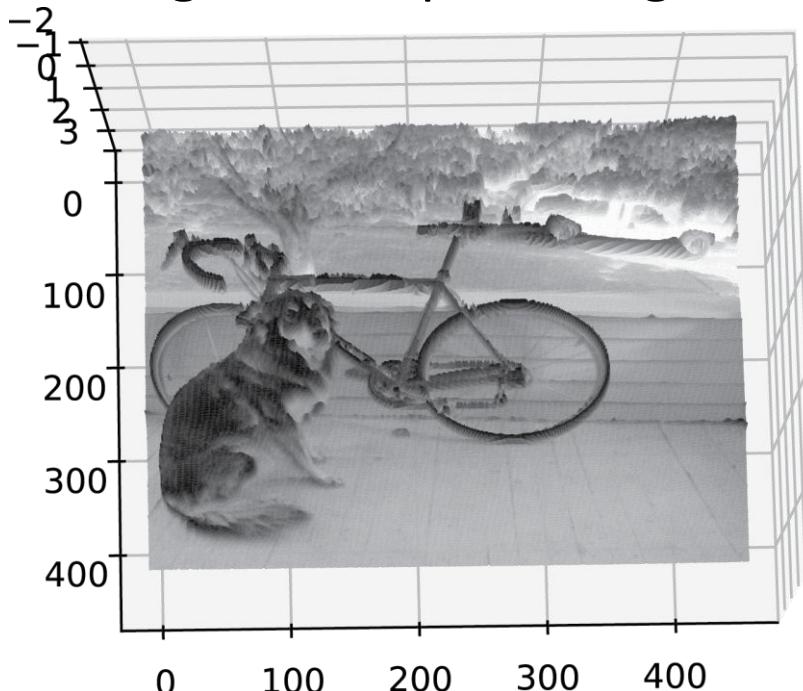
---

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?

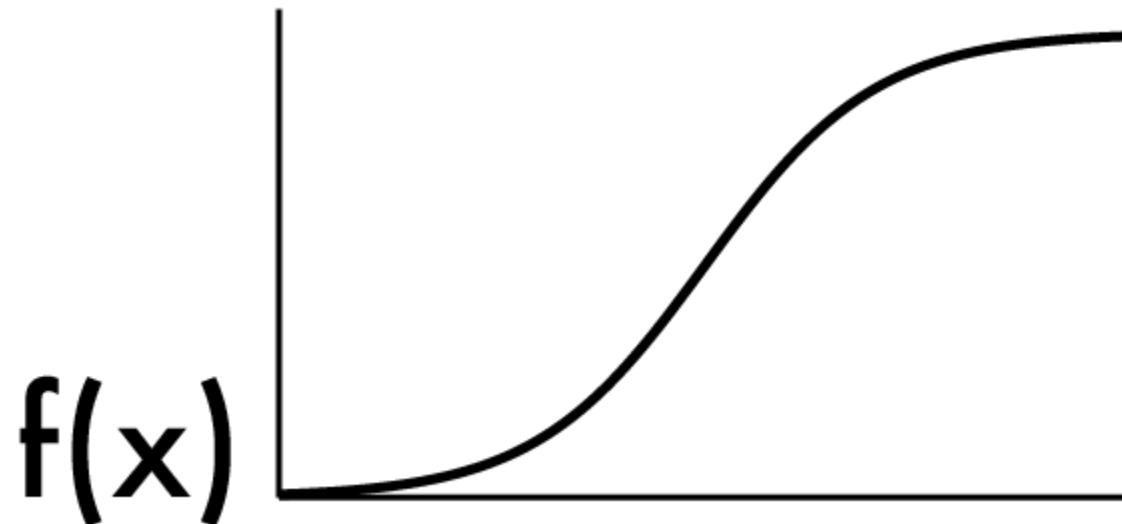
- Image is a function.
- Think of the gray tones as HEIGHTS.
- Edges are rapid changes in this function



---

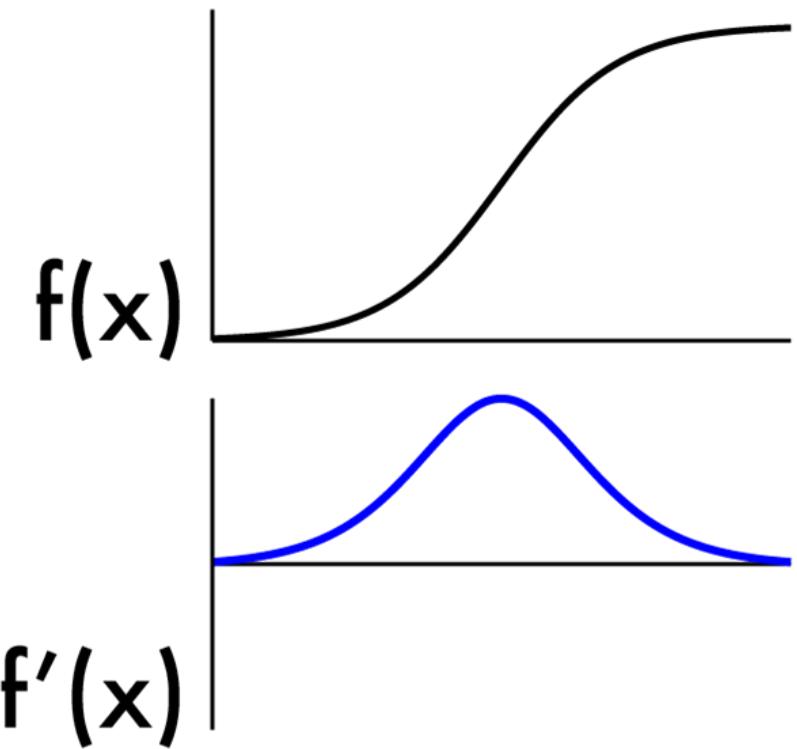
# What's an edge?

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



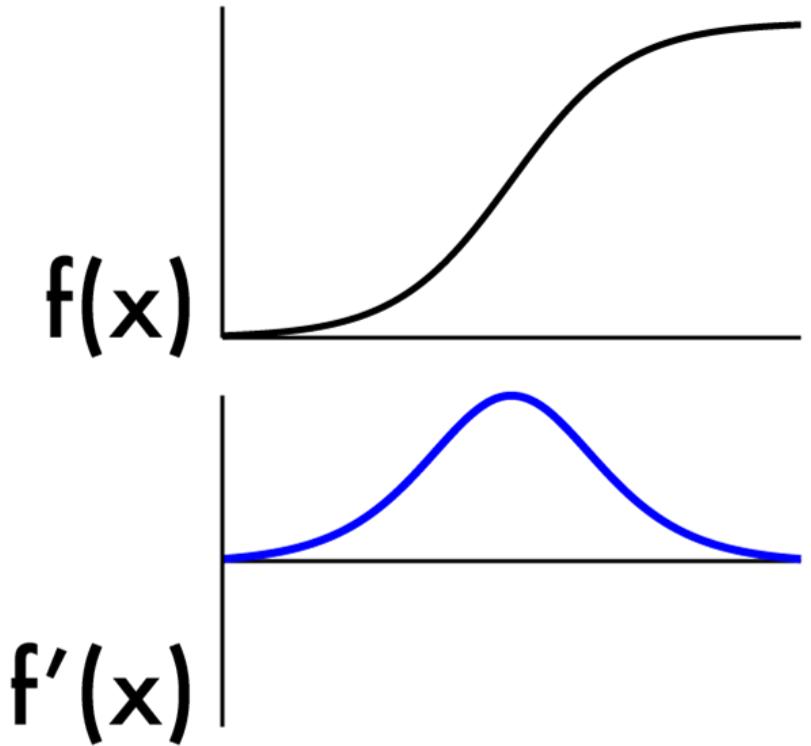
# Finding edges

- Could take derivative
- Edges = high response



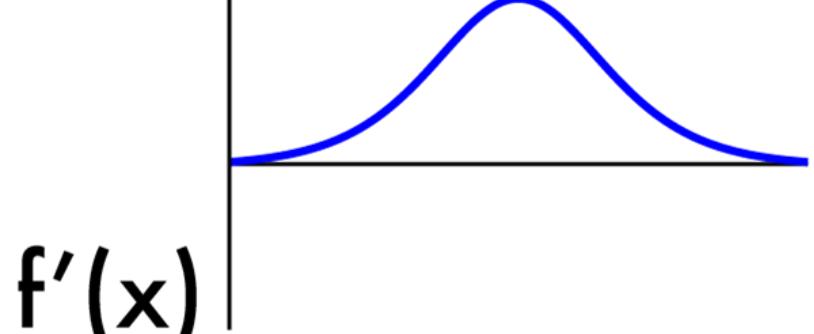
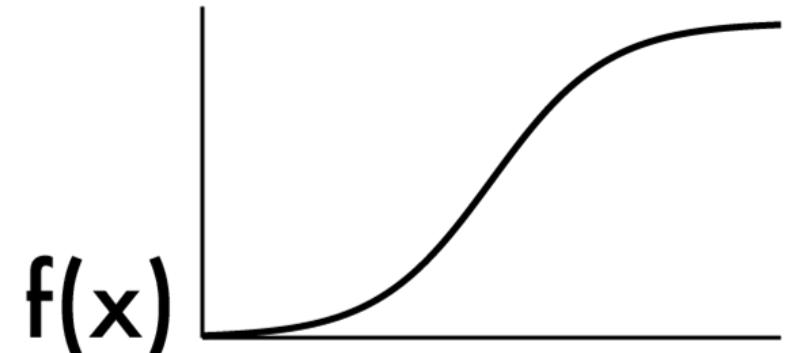
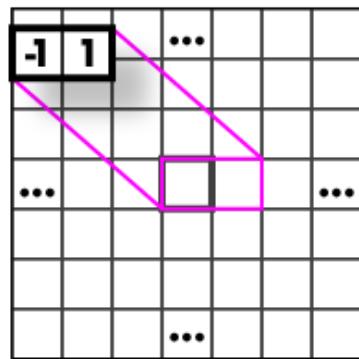
# Image derivatives

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



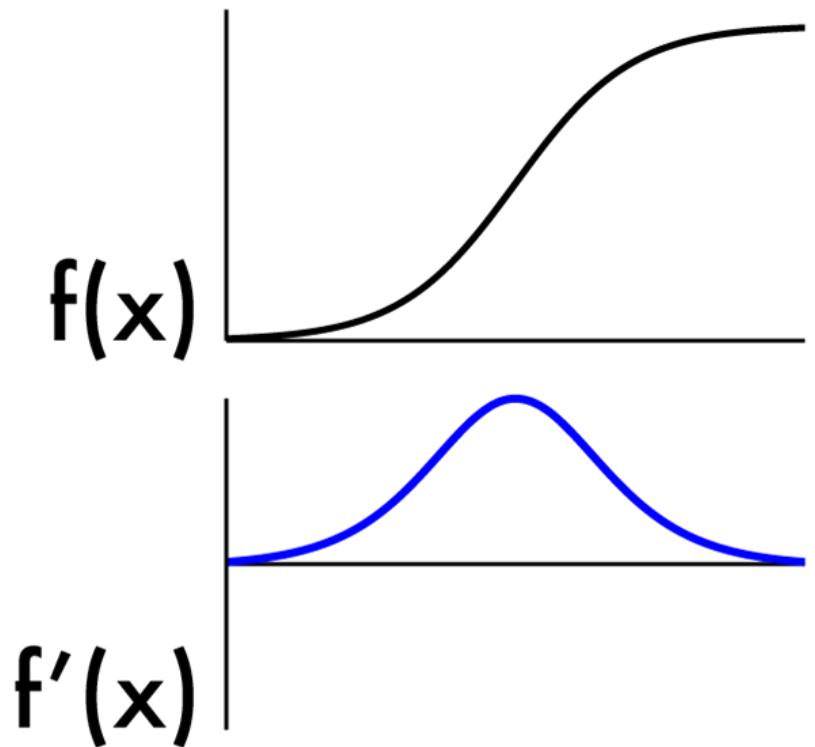
# Image derivatives

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



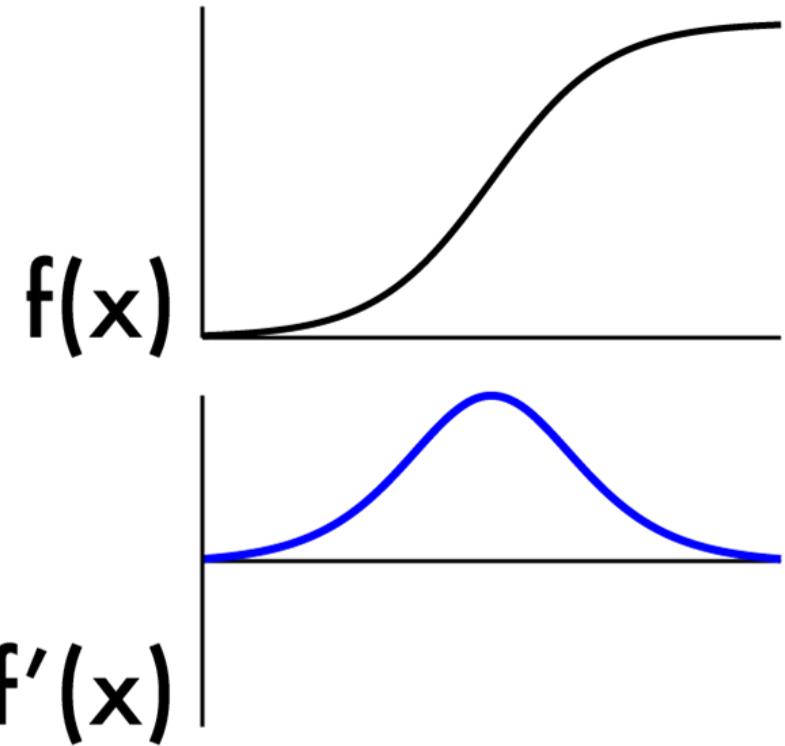
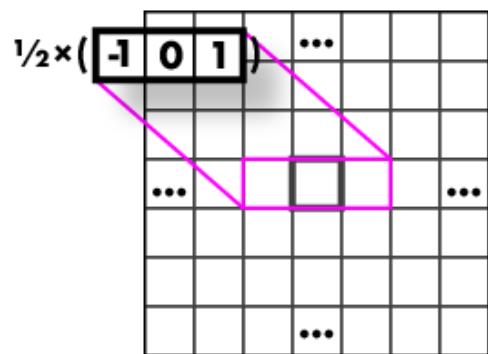
# Image derivatives

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



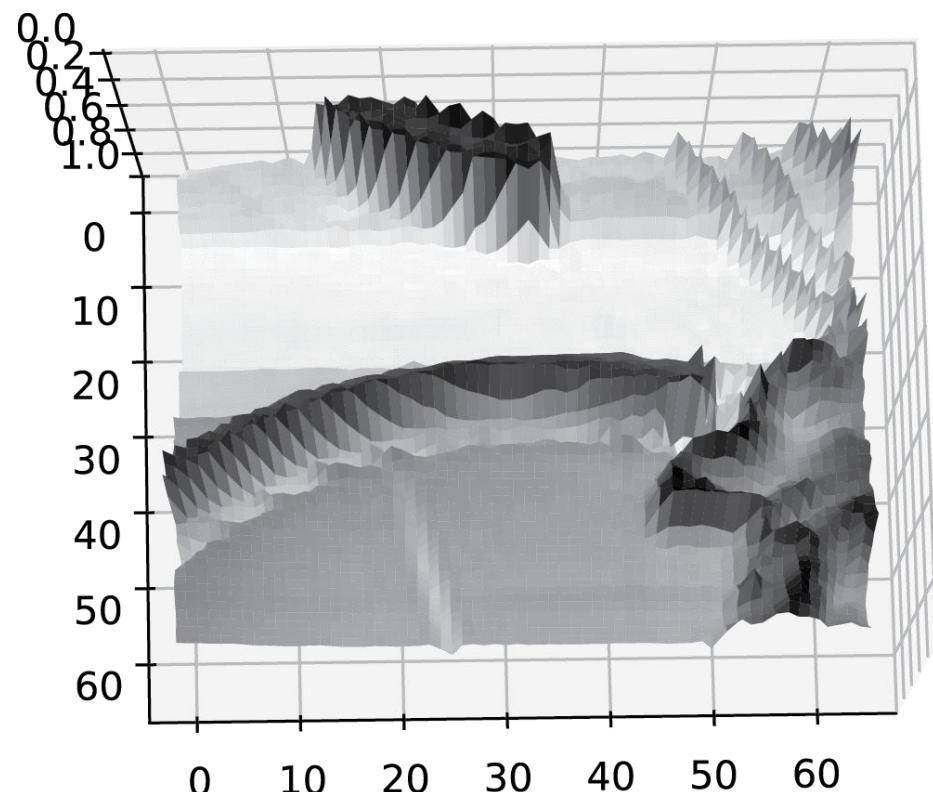
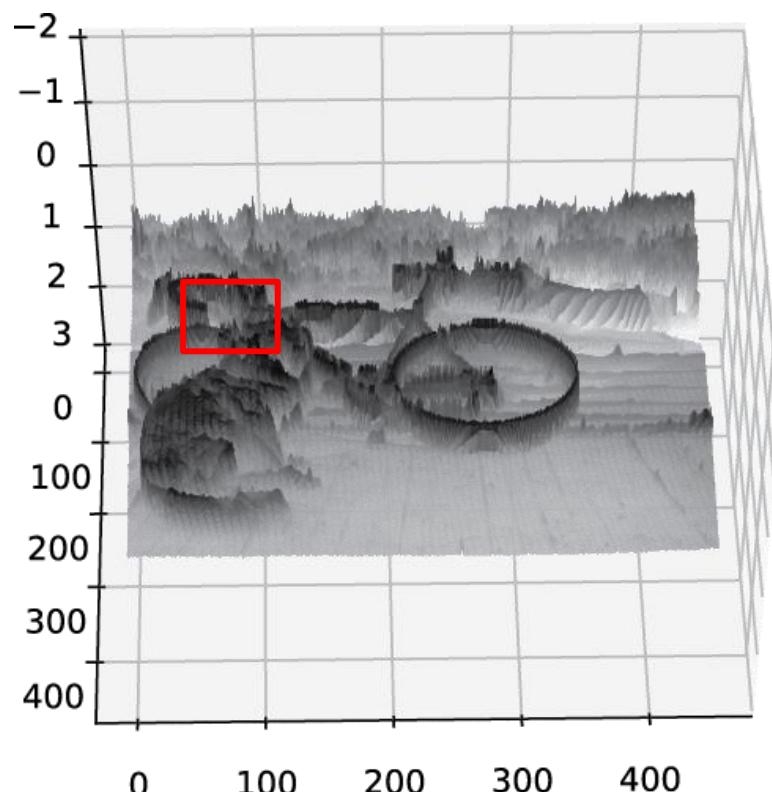
# Image derivatives

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

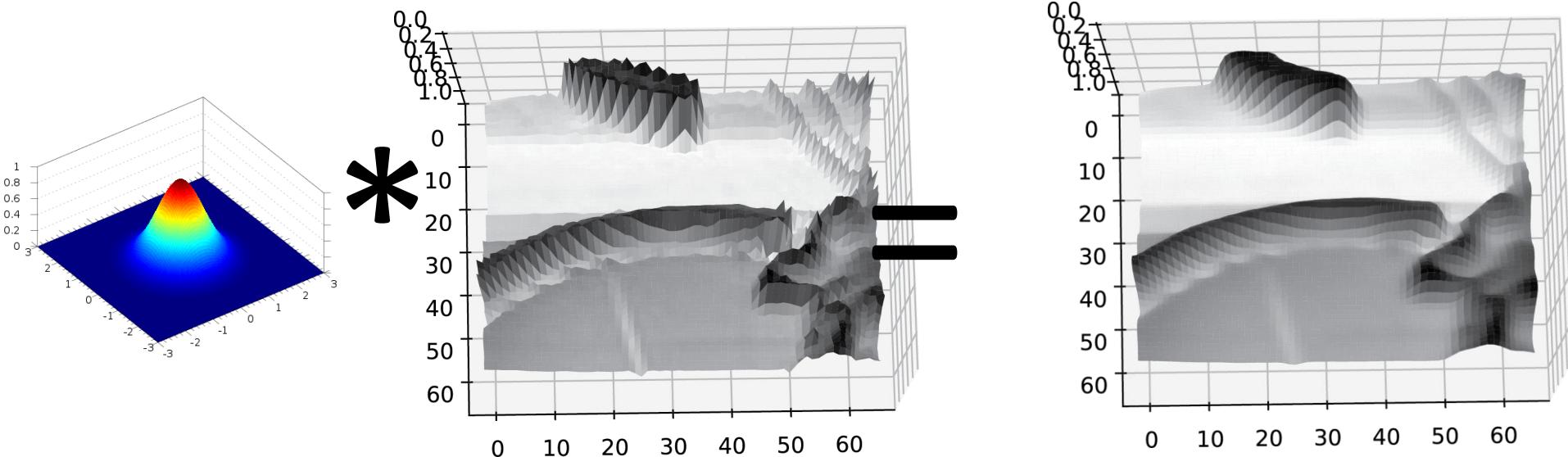


---

# Images are noisy!

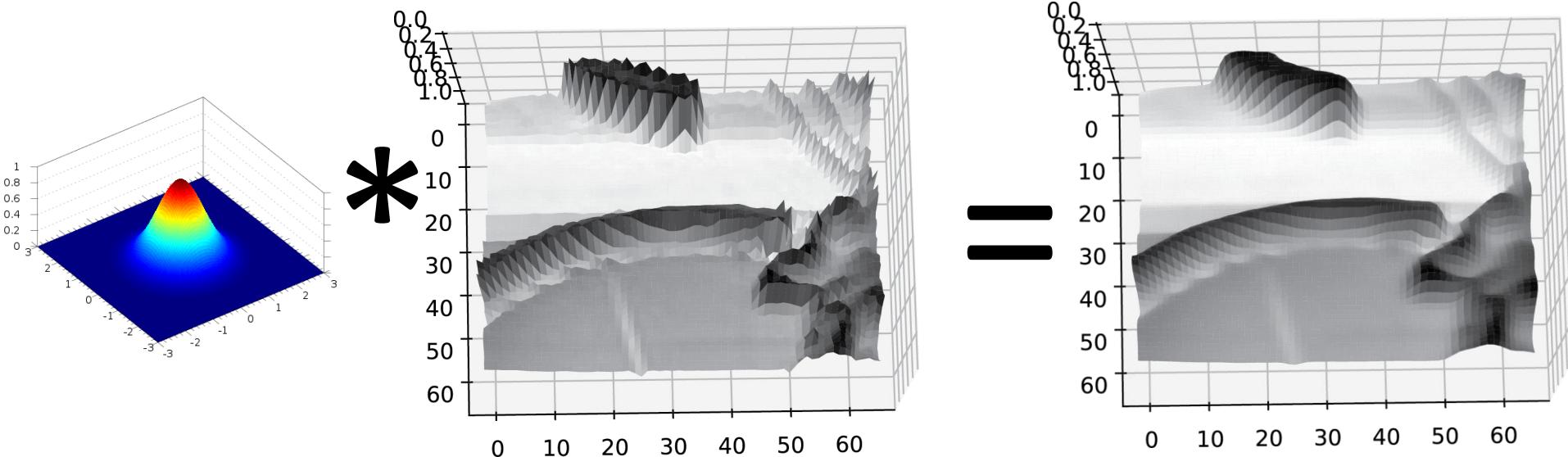


# But we already know how to smooth



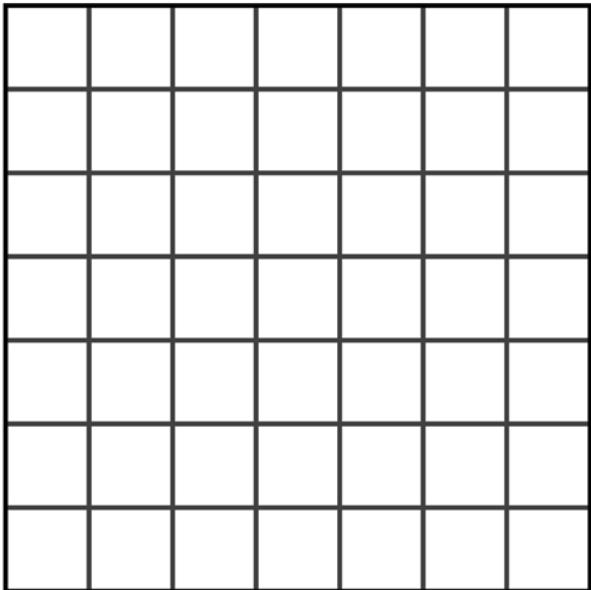
---

# But we already know how to smooth



---

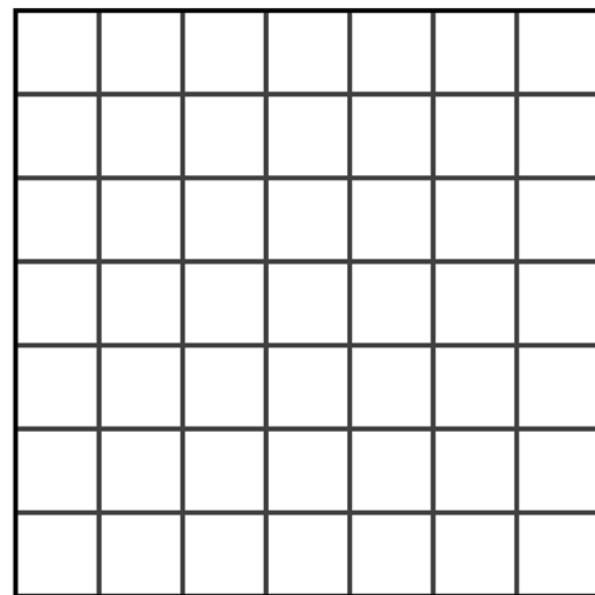
# Smooth first, then derivative

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

$$)$$

---

# Smooth first, then derivative

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



# Smooth first, then derivative

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

A diagram illustrating the convolution operation. A 1x3 kernel  $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$  is applied to a 3x3 input matrix  $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ . The result is a 1x1 output  $\begin{bmatrix} 2 \end{bmatrix}$ , scaled by  $\frac{1}{2}$ .

The diagram shows the convolution process. A pink arrow points from the top-left element of the kernel to the top-left element of the input matrix. A large black arrow points from the input matrix to the output scalar. The output scalar is multiplied by  $\frac{1}{2}$ .

# Smooth first, then derivative

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

The diagram illustrates the convolution operation. A 1x3 kernel  $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$  is applied to a 3x3 input matrix  $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ . The result is a 1x1 output  $\begin{bmatrix} 2 & 0 \end{bmatrix}$ .

A pink arrow points from the top-left cell of the input matrix to the top-left cell of the output matrix, indicating the receptive field of that output unit.

# Smooth first, then derivative

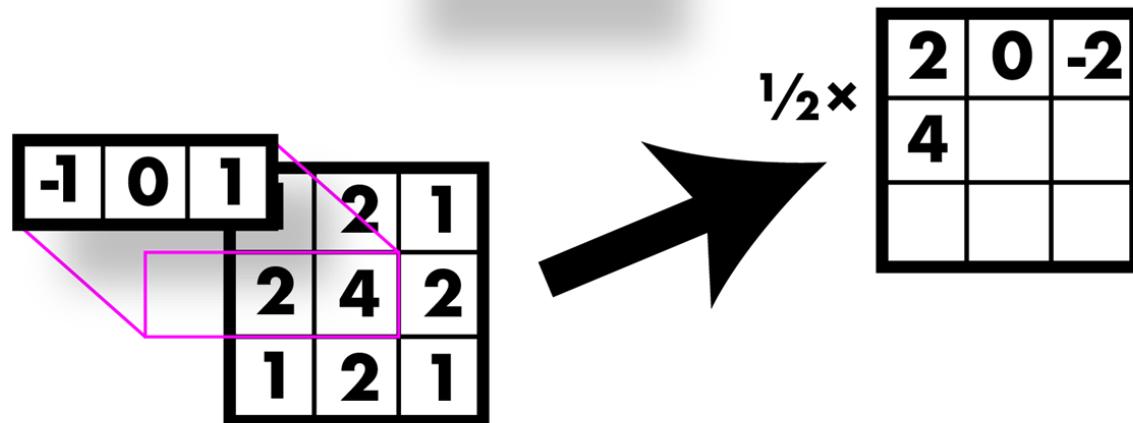
$$\frac{1}{2} \times \left( \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right)$$

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

The diagram illustrates a convolution operation. At the top, a horizontal vector  $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$  is multiplied by a vertical  $3 \times 3$  input matrix. The result is then scaled by  $\frac{1}{2}$ . Below this, a large black arrow points from the input matrix to a resulting vector  $\begin{bmatrix} 2 & 0 & -2 \end{bmatrix}$ . The input matrix has a highlighted central element (2) with a pink border, indicating it is the receptive field for the output value 2.

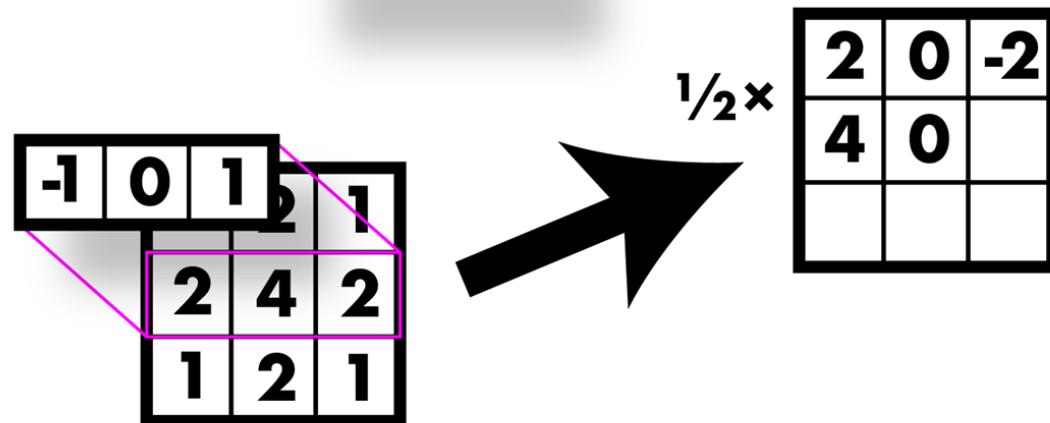
# Smooth first, then derivative

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



# Smooth first, then derivative

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



# Smooth first, then derivative

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

The diagram illustrates a convolution operation. A 3x3 input matrix is multiplied by a 3x3 kernel matrix, with the result being scaled by  $\frac{1}{2}$ . The input matrix has its top-right element highlighted in pink. The result is a 2x2 output matrix.

Input Matrix:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Kernel Matrix:

$$\begin{bmatrix} -1 & 0 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Output Matrix (Result of  $\frac{1}{2} \times$ ):

$$\begin{bmatrix} 2 & 0 & -2 \\ 4 & 0 & -4 \end{bmatrix}$$

# Smooth first, then derivative

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

The diagram illustrates a convolution operation. At the top, a horizontal vector  $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$  is multiplied by a vertical  $3 \times 3$  input matrix. The result is then multiplied by  $\frac{1}{2}$ . Below this, a large black arrow points from the input matrix to the resulting output matrix. The input matrix has its second row highlighted with a black bar, and the third column of the kernel is highlighted with a pink box. The output matrix has its first row highlighted with a black bar.

$$\frac{1}{2} \times \begin{bmatrix} 2 & 0 & -2 \\ 4 & 0 & -4 \\ 2 & & \end{bmatrix}$$

# Smooth first, then derivative

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

The diagram illustrates a convolution operation. A 3x3 input matrix (smoothed) is multiplied by a 1x3 kernel (derivative). The result is a 3x1 output vector.

Input Matrix (Smoothed):

1	2	1
2	4	2
1	2	1

Kernel (Derivative):

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

Output Vector:

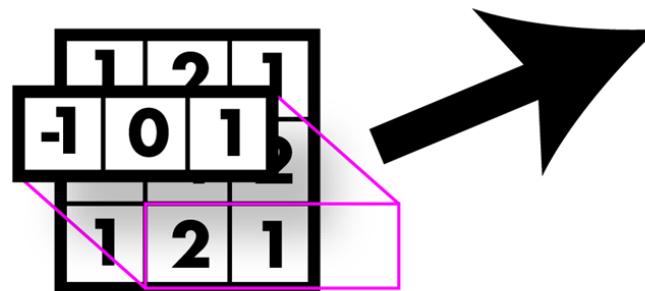
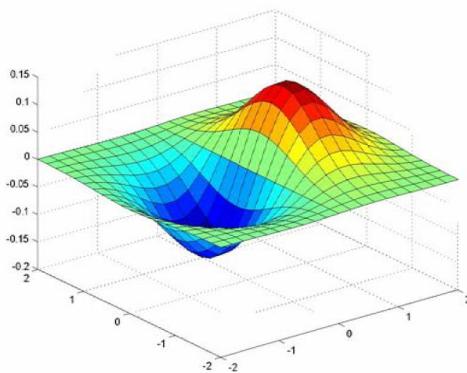
2	0	-2
4	0	-4
2	0	

# Smooth first, then derivative

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

# Sobel filter! Smooth & derivative

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



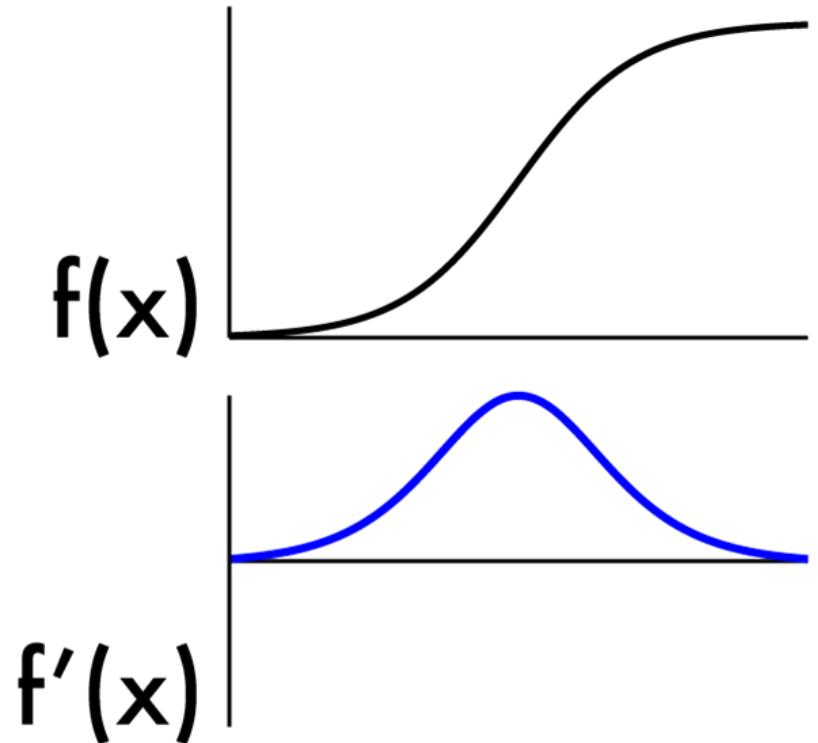
$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

# Image derivatives

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

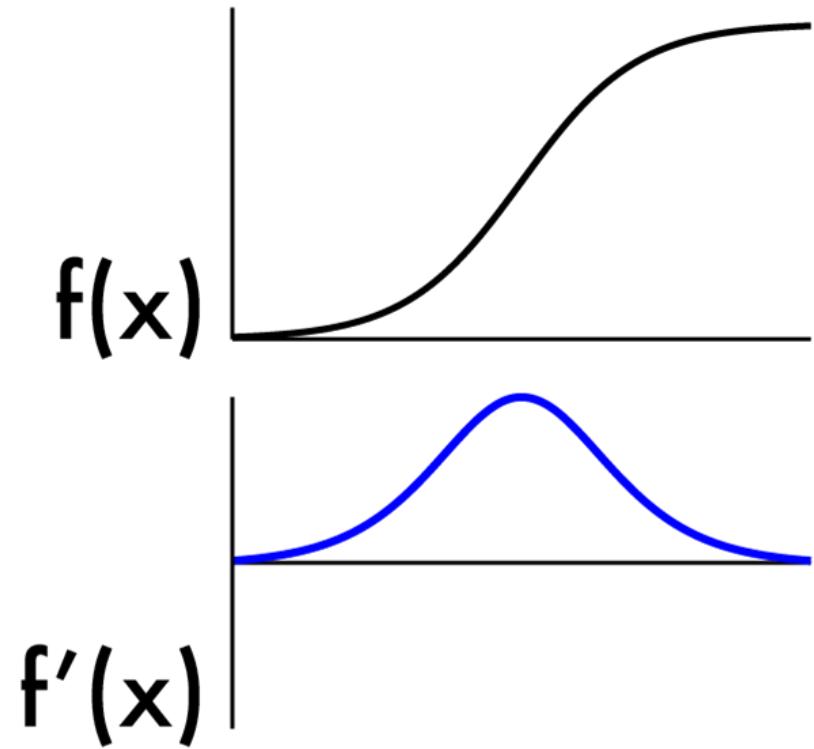
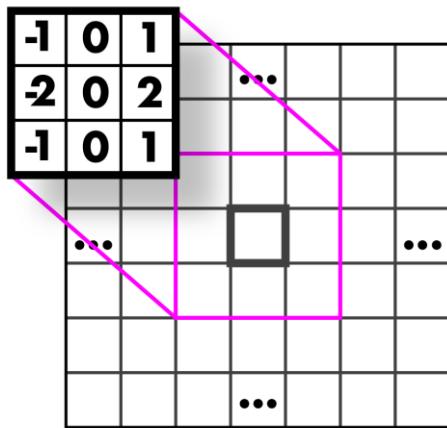
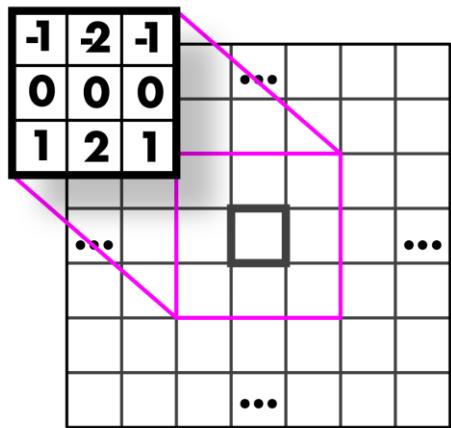
$$\frac{1}{2} \times \left( \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right)$$

The diagram illustrates the convolution operation. It shows a 3x3 input kernel  $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$  being multiplied by a 3x3 weight kernel  $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$ . The result is a 3x3 output kernel  $\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$ . A pink box highlights the bottom-right element of the input kernel, and a pink arrow points from it to the bottom-right element of the output kernel.

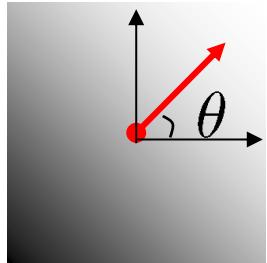


# Finding edges

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



# Simplest image gradient



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

How would you implement this as a filter?

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

$$\begin{array}{c} 0 \end{array} \begin{array}{c} -1 \end{array} \begin{array}{c} 1 \end{array}$$

The **gradient direction** is given

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

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:

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

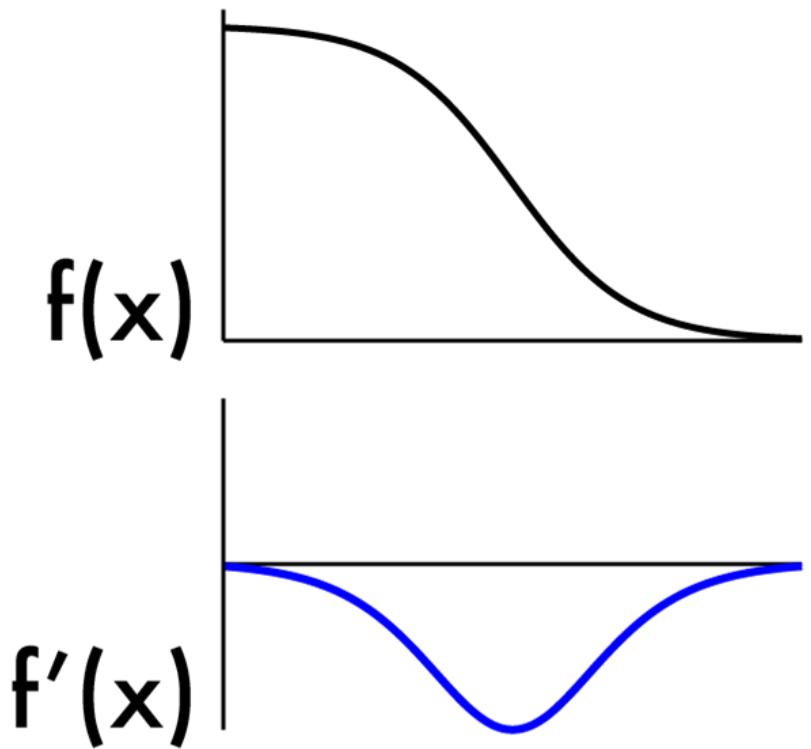
Orientation:

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

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

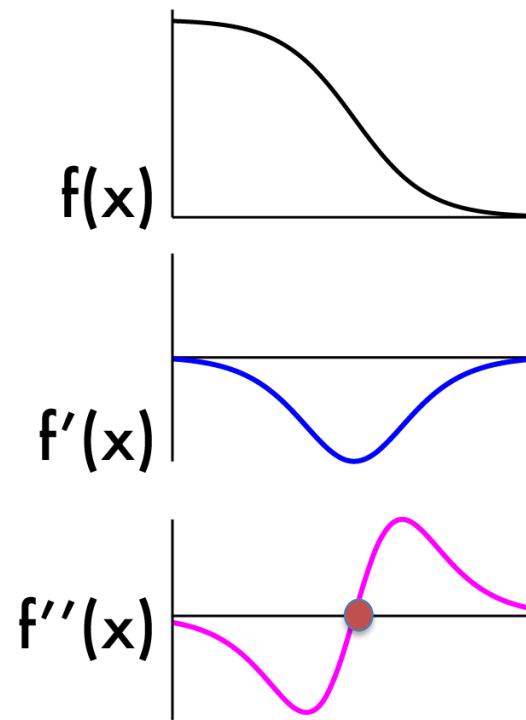
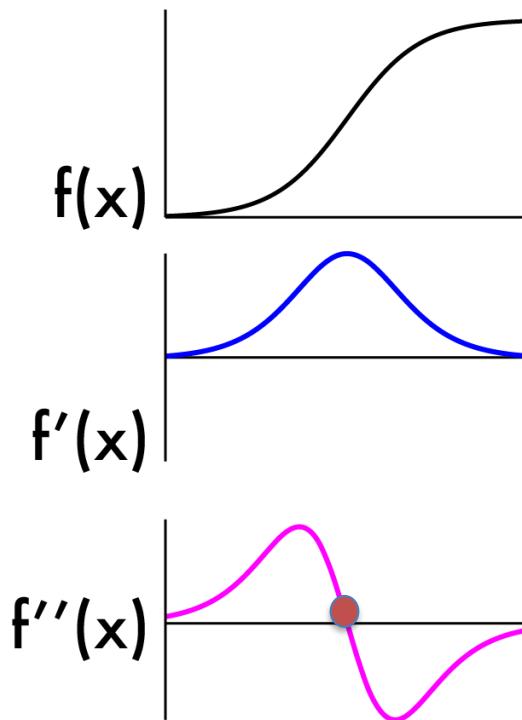
# Finding edges

- 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

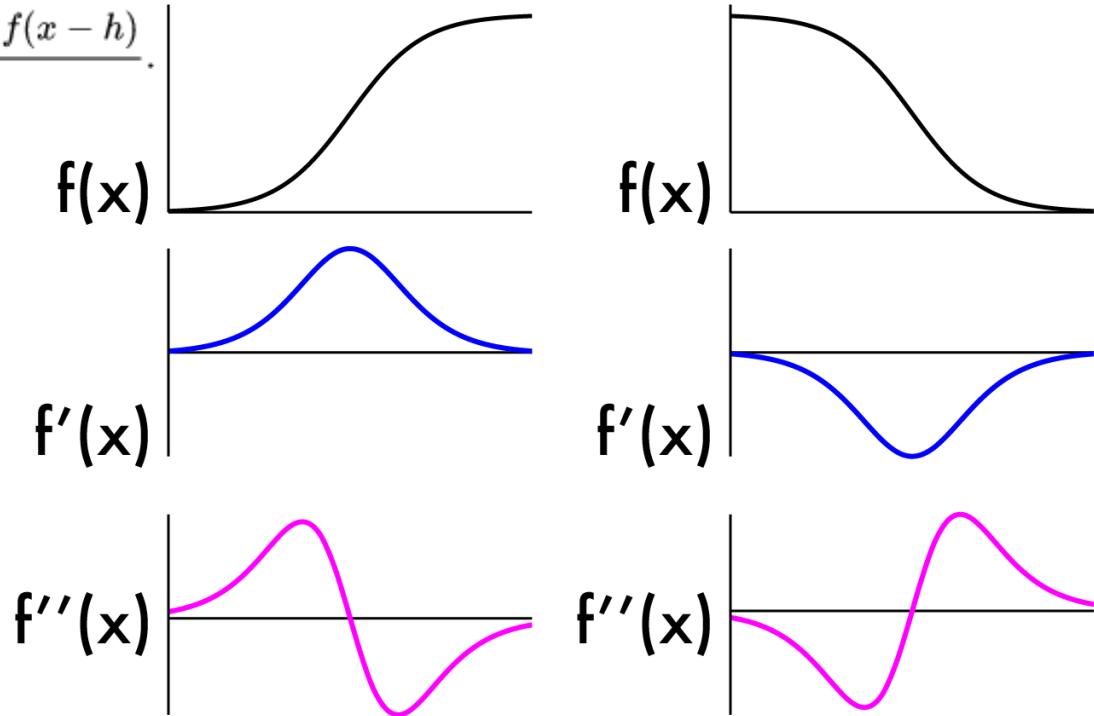
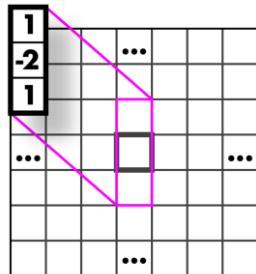
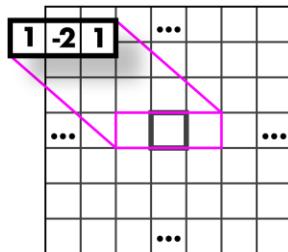
- Recall:

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

- 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

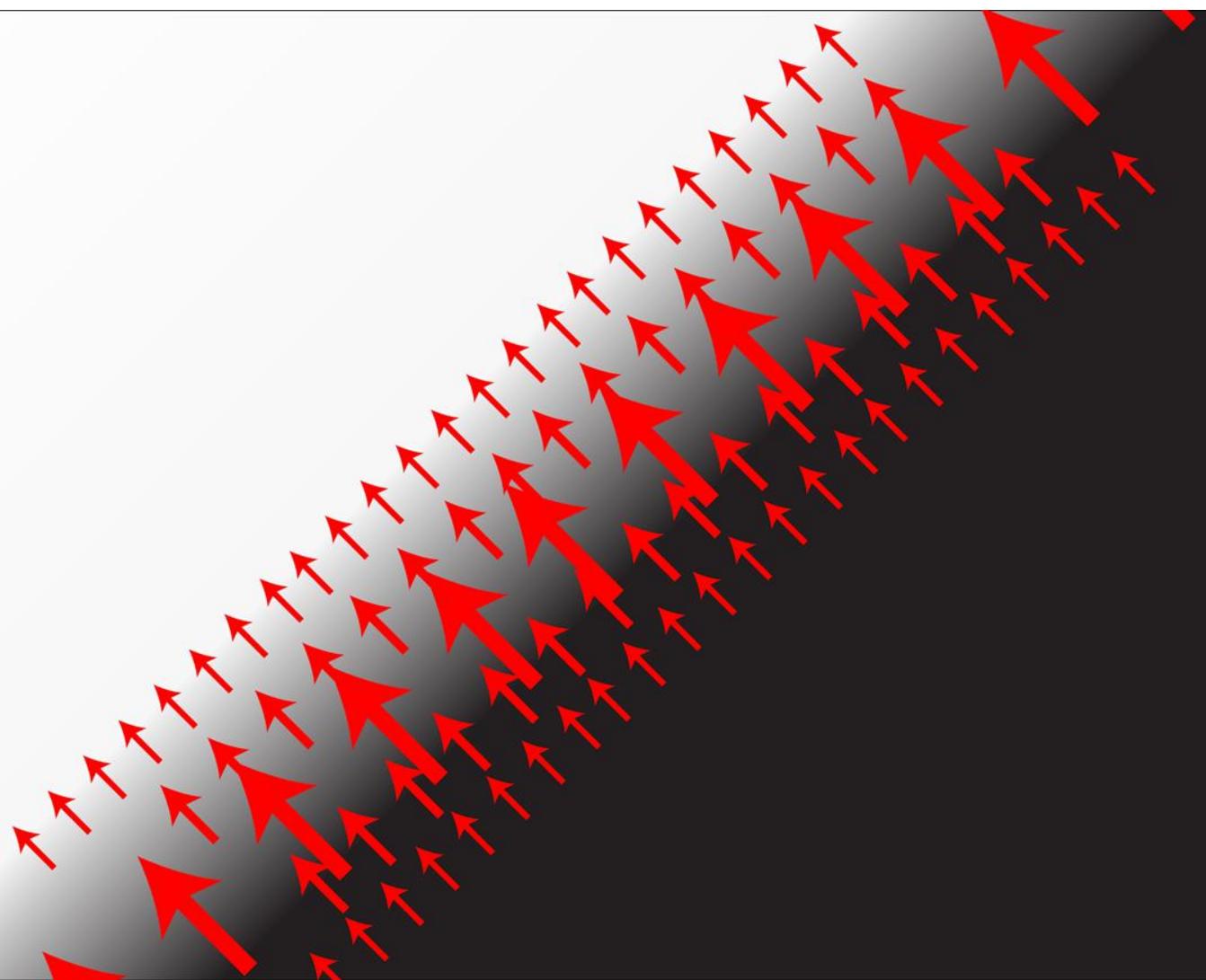
- Laplacian:

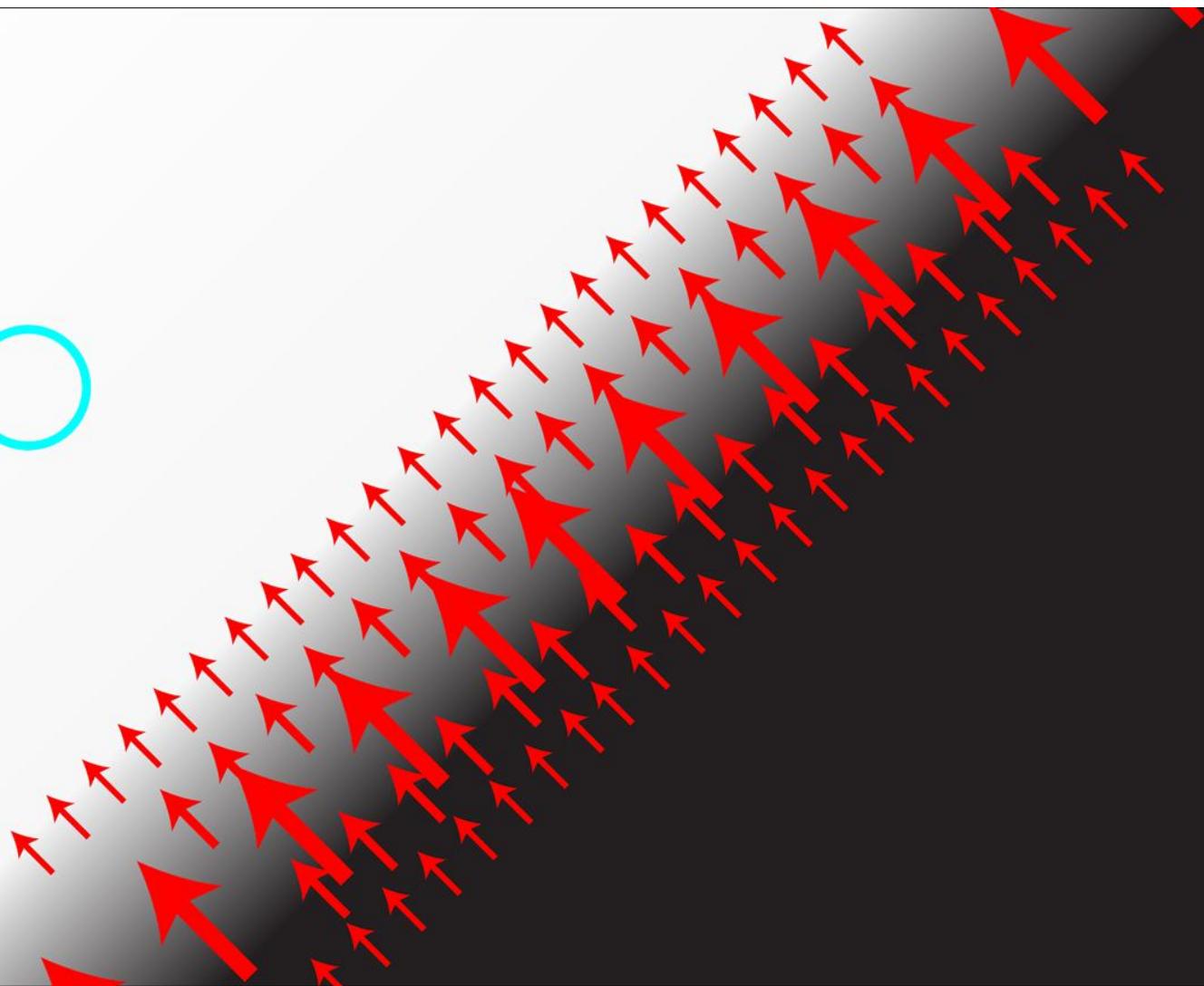
$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

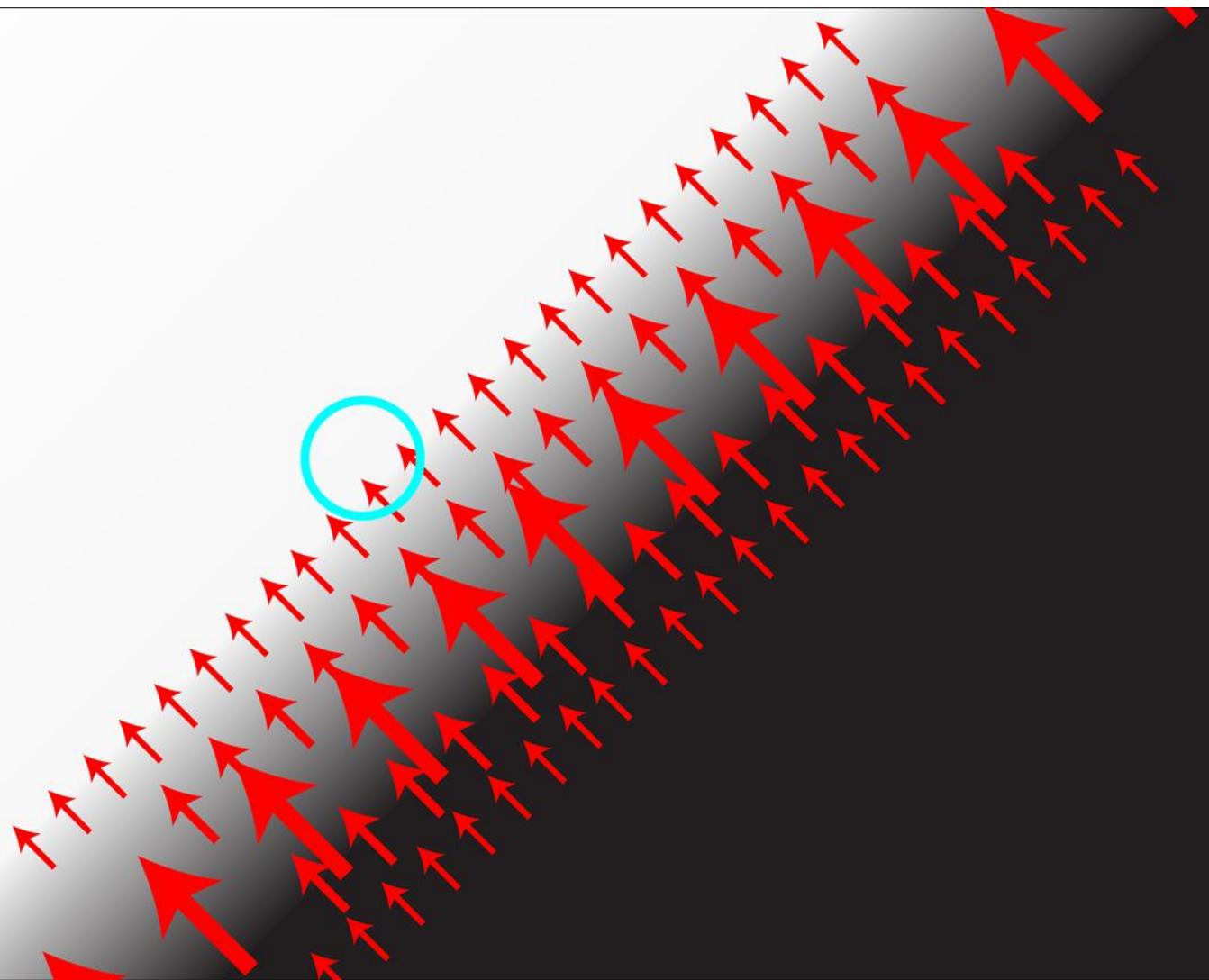
- Measures the divergence of the gradient
  - Flux of gradient vector field through small area

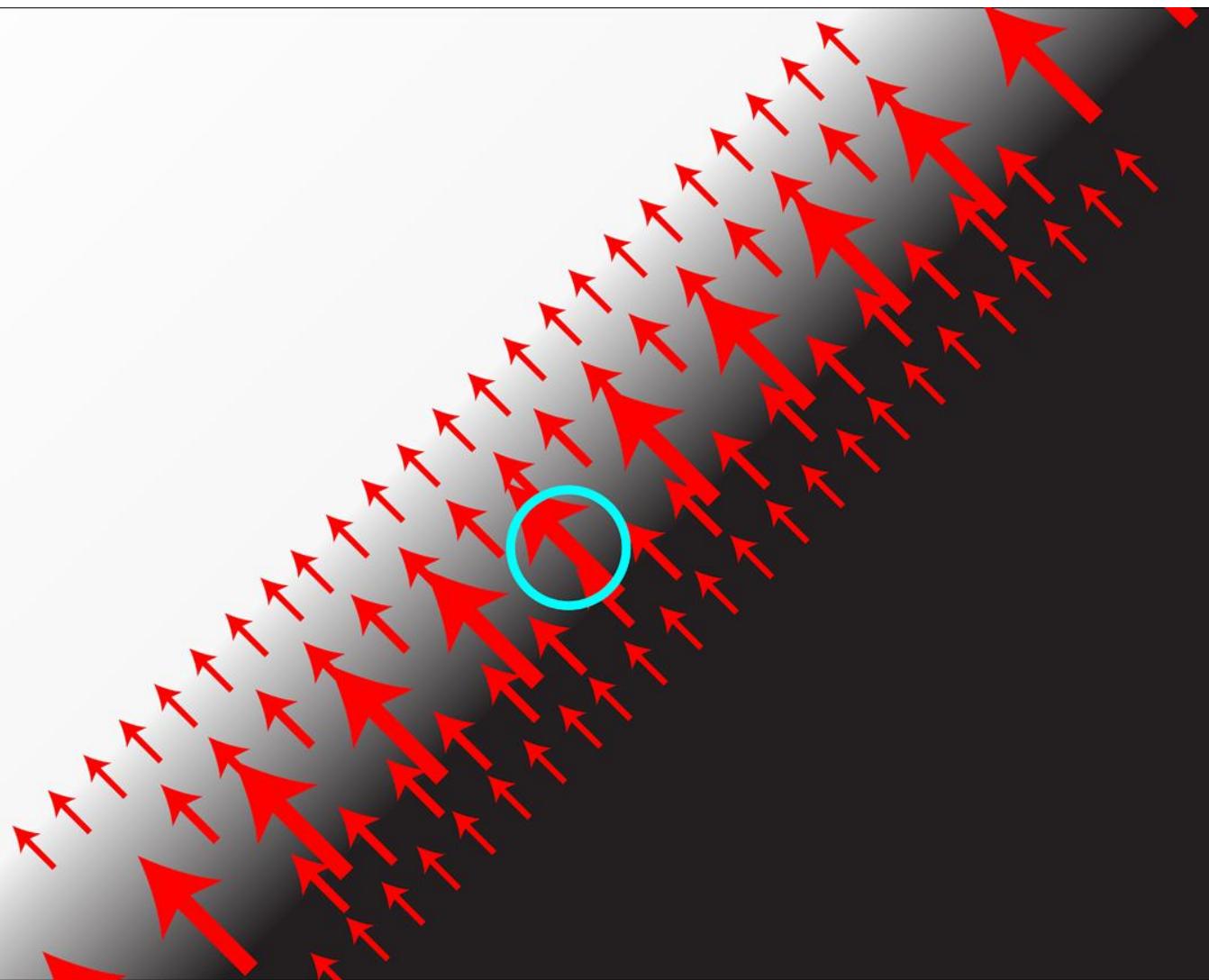


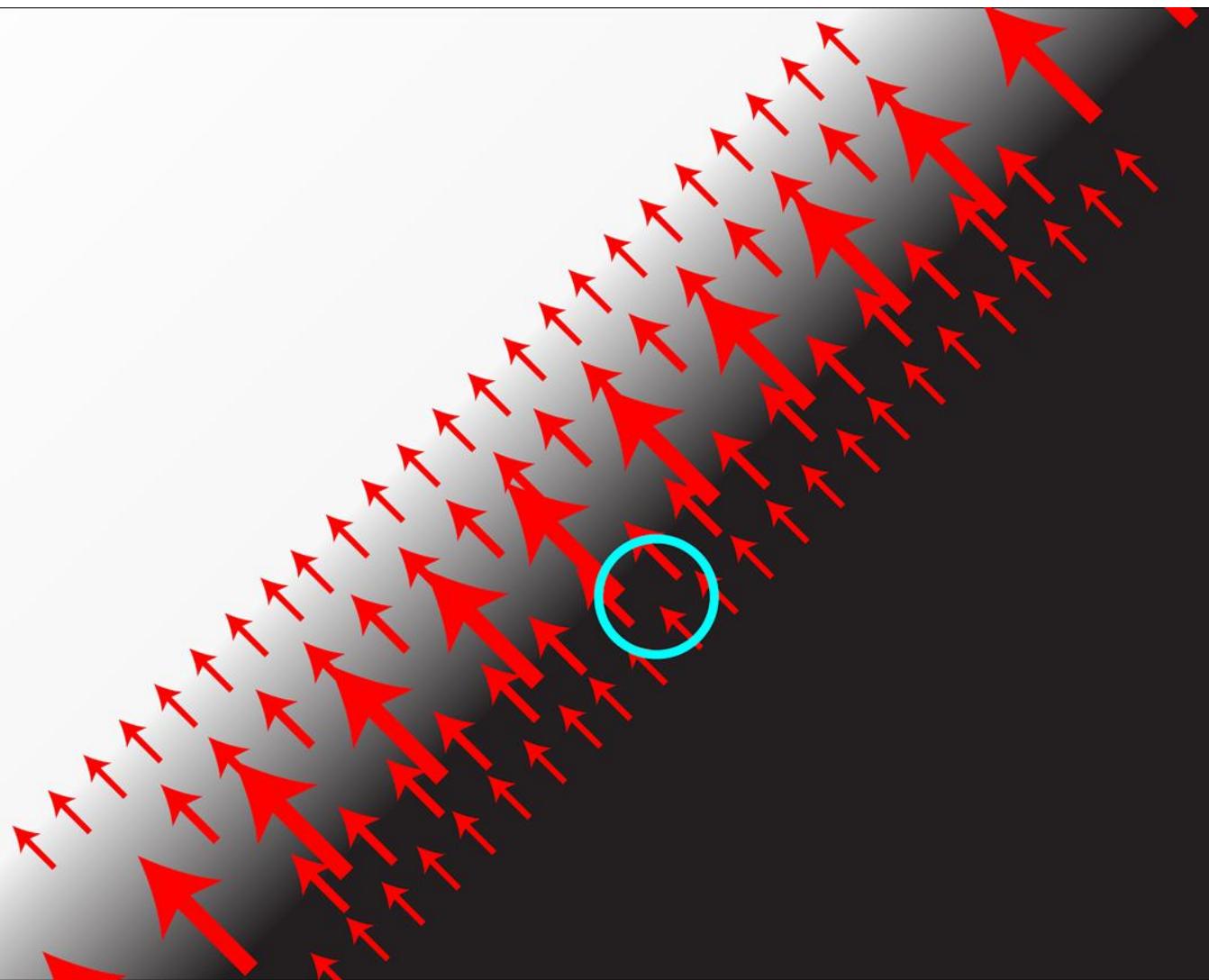








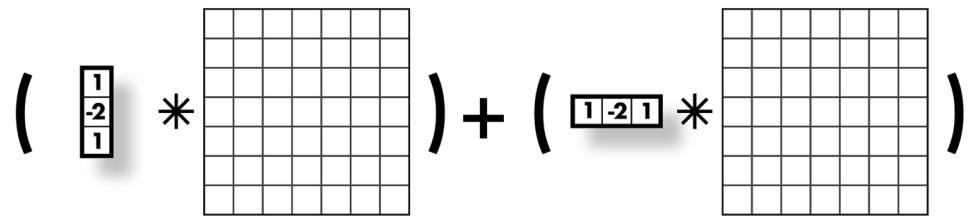




# Laplacians

- ### - Laplacian:

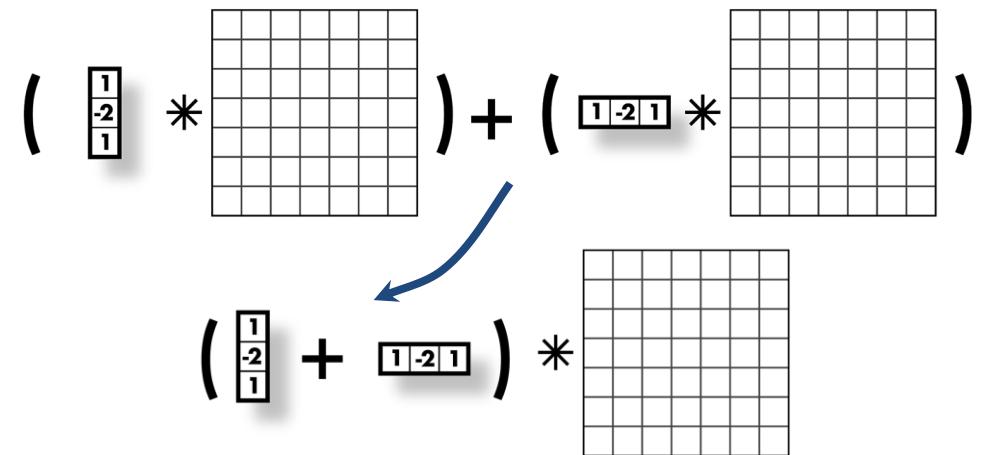
$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$



# Laplacians

- Laplacian:

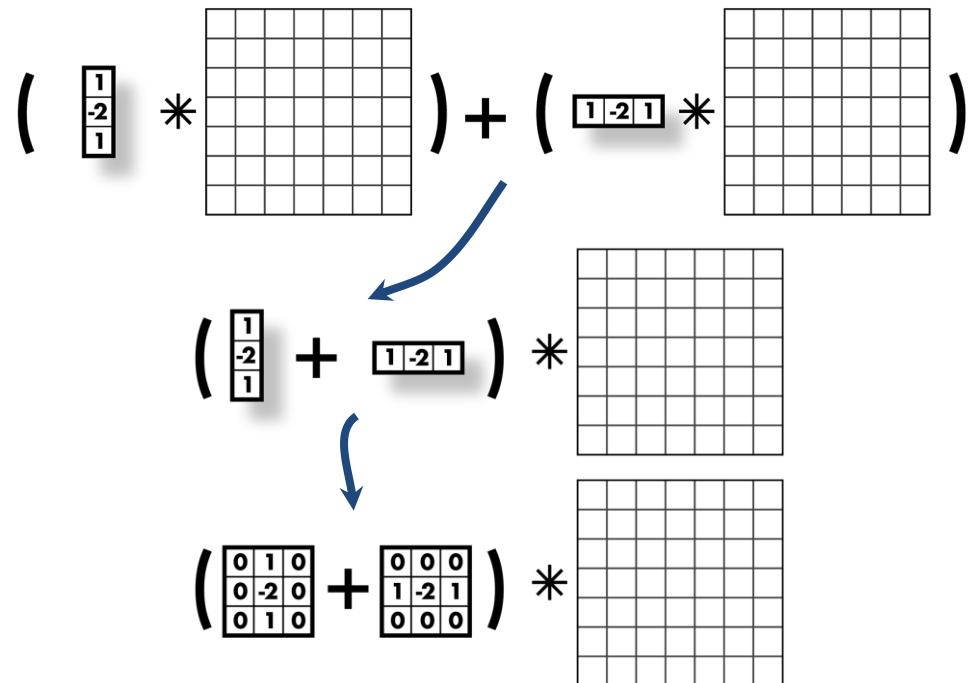
- $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$



# Laplacians

- Laplacian:

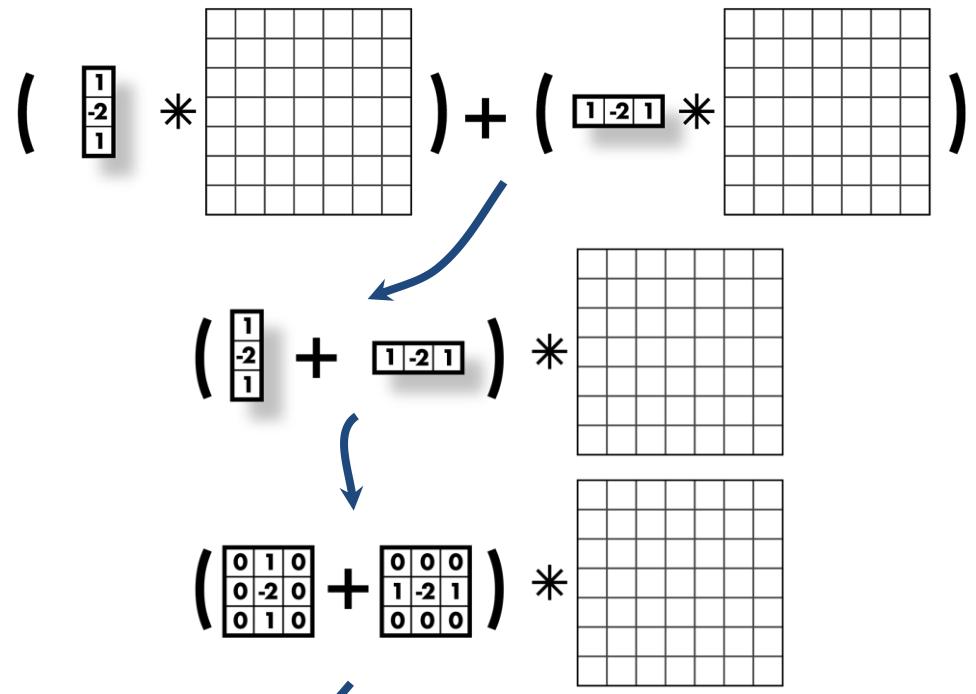
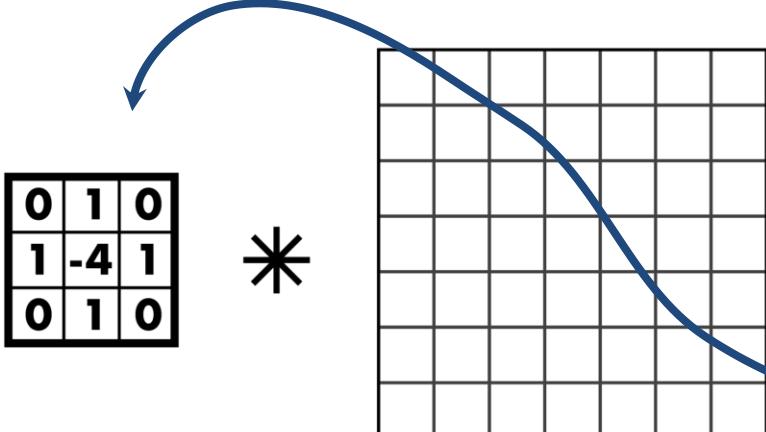
- $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$



# Laplacians

- Laplacian:

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

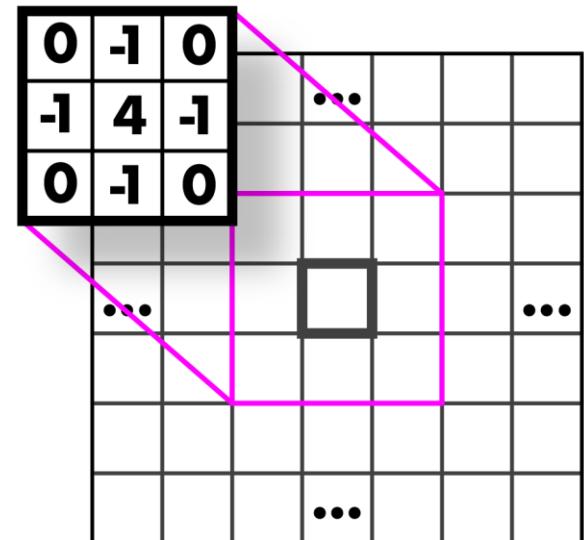
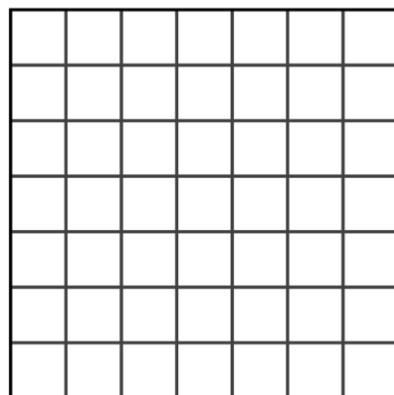


# Laplacians

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

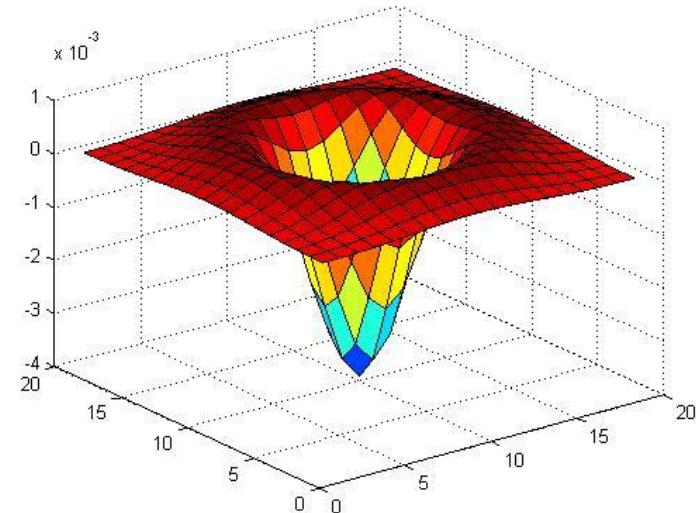
0	1	0
1	-4	1
0	1	0

\*



# 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)

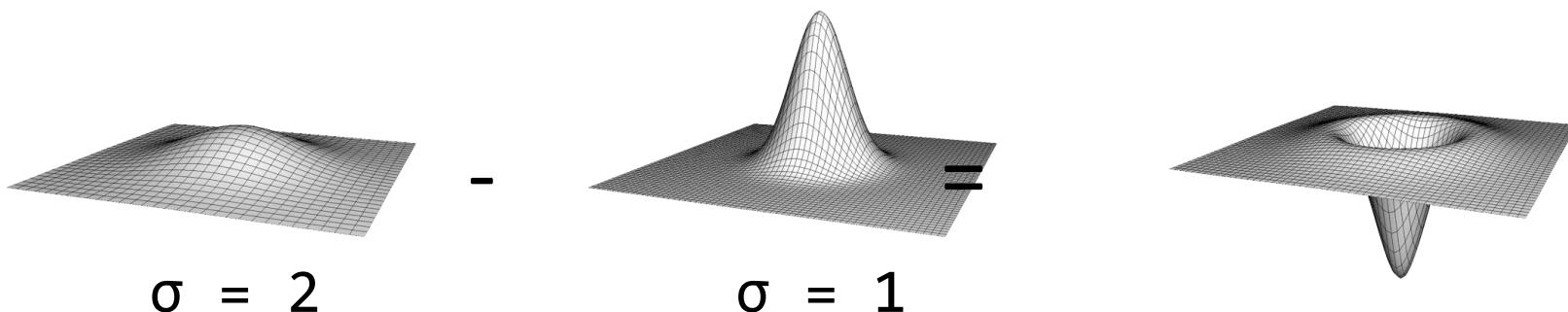
---

# 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)

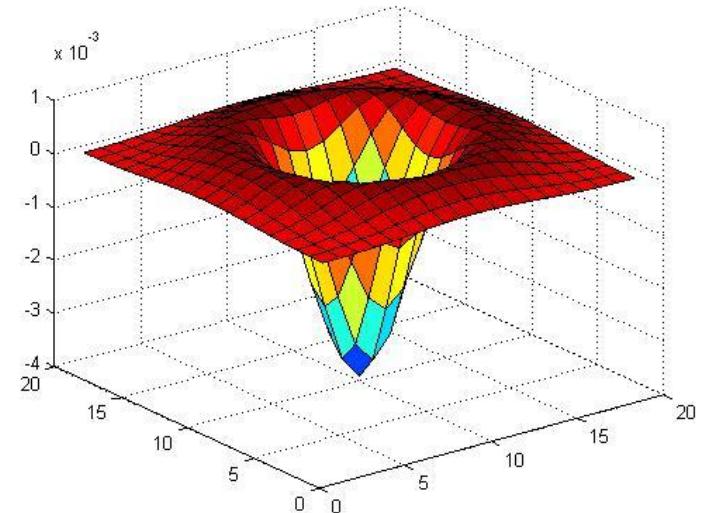
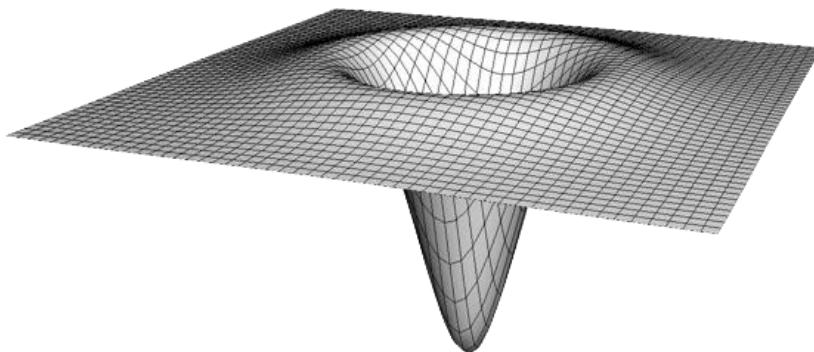
$$- g(\sigma_1) * I - g(\sigma_2) * I = [g(\sigma_1) - g(\sigma_2)] * I$$



---

# 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)



---

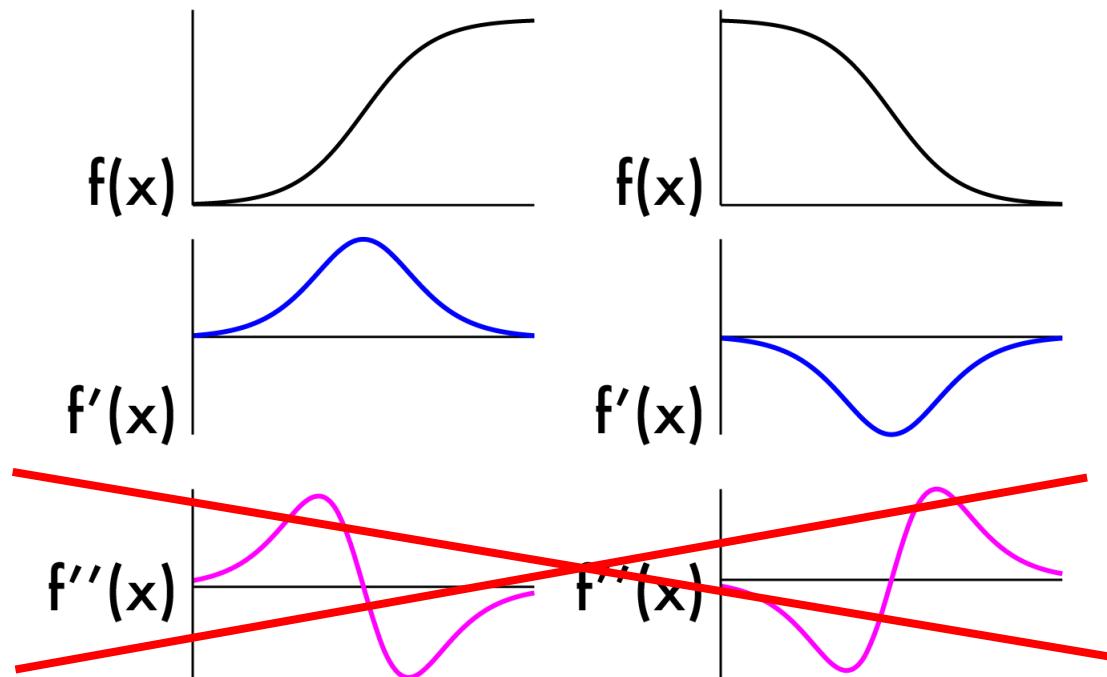
# DoG (4 - 3)



---

## Another approach: gradient magnitude

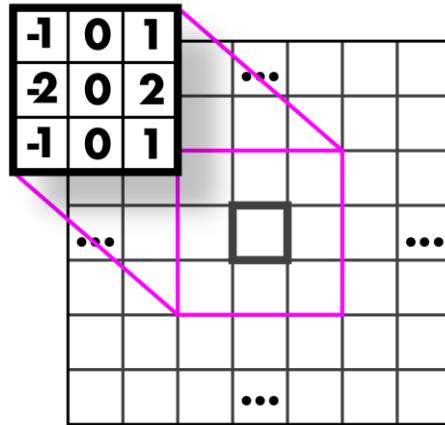
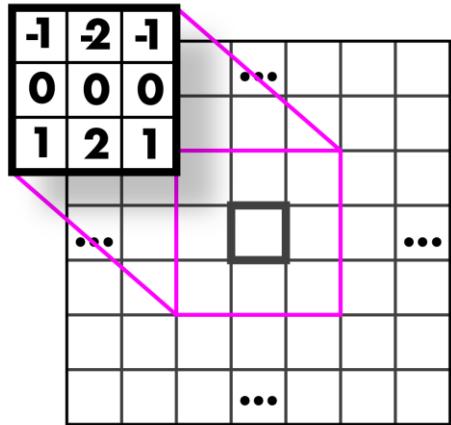
- Don't need 2nd derivatives
- Just use magnitude of gradient



---

## Another approach: gradient magnitude

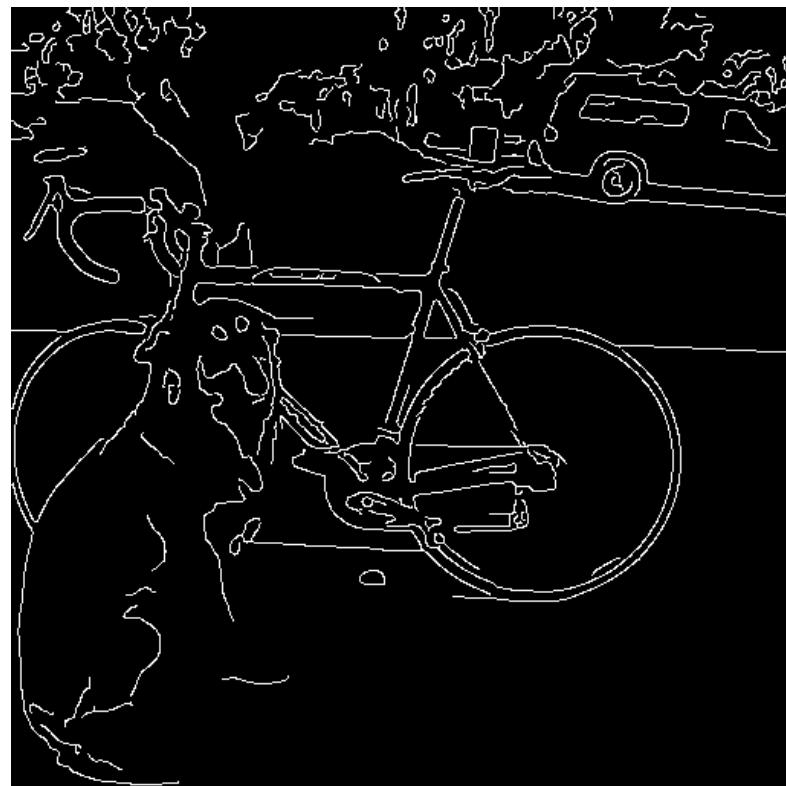
- Don't need 2nd derivatives
- Just use magnitude of gradient
- Are we done? No!





---

# 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

---

# Smooth image

- You know how to do this, gaussians!

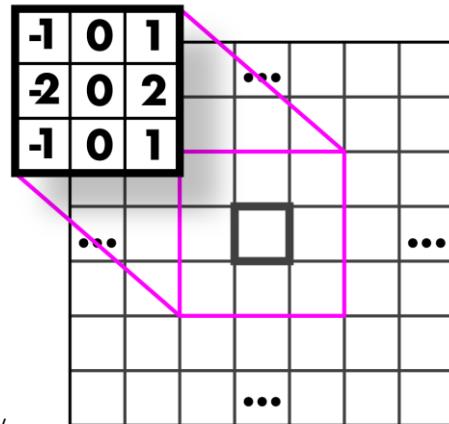
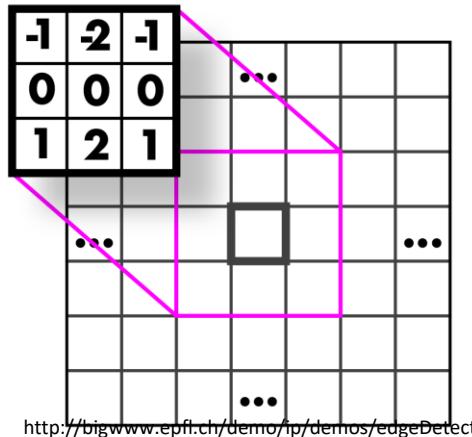


<http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/>



# Gradient magnitude and direction

- Sobel filter



---

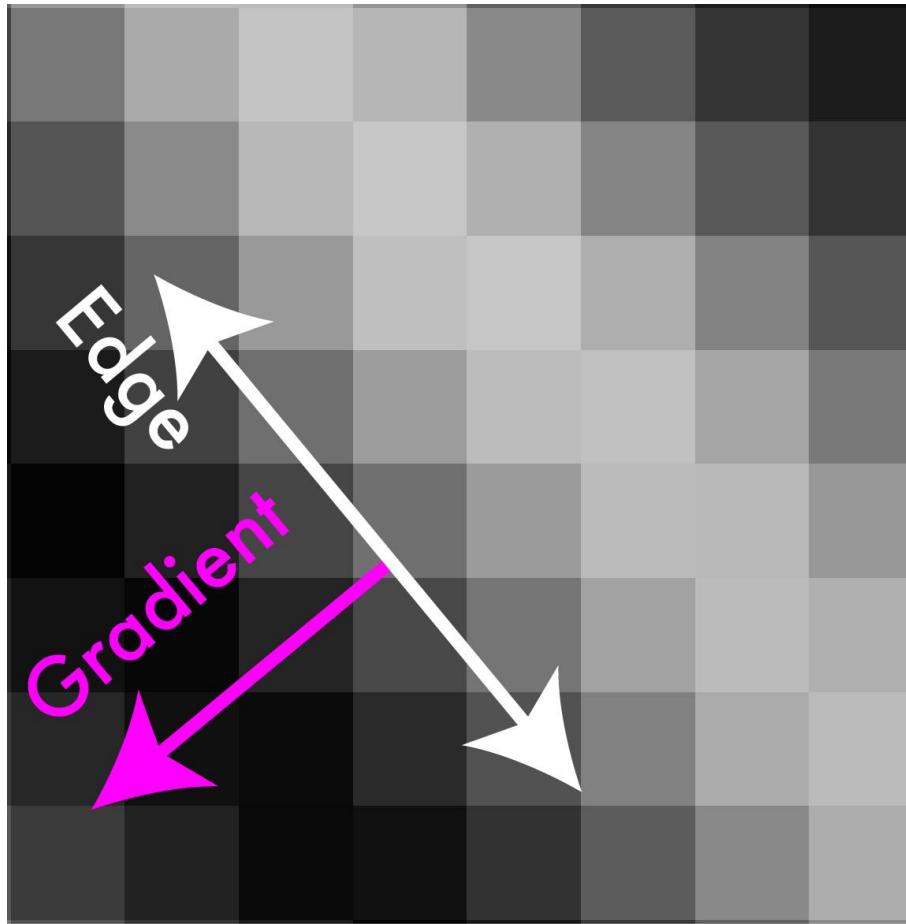
# Non-maximum suppression

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



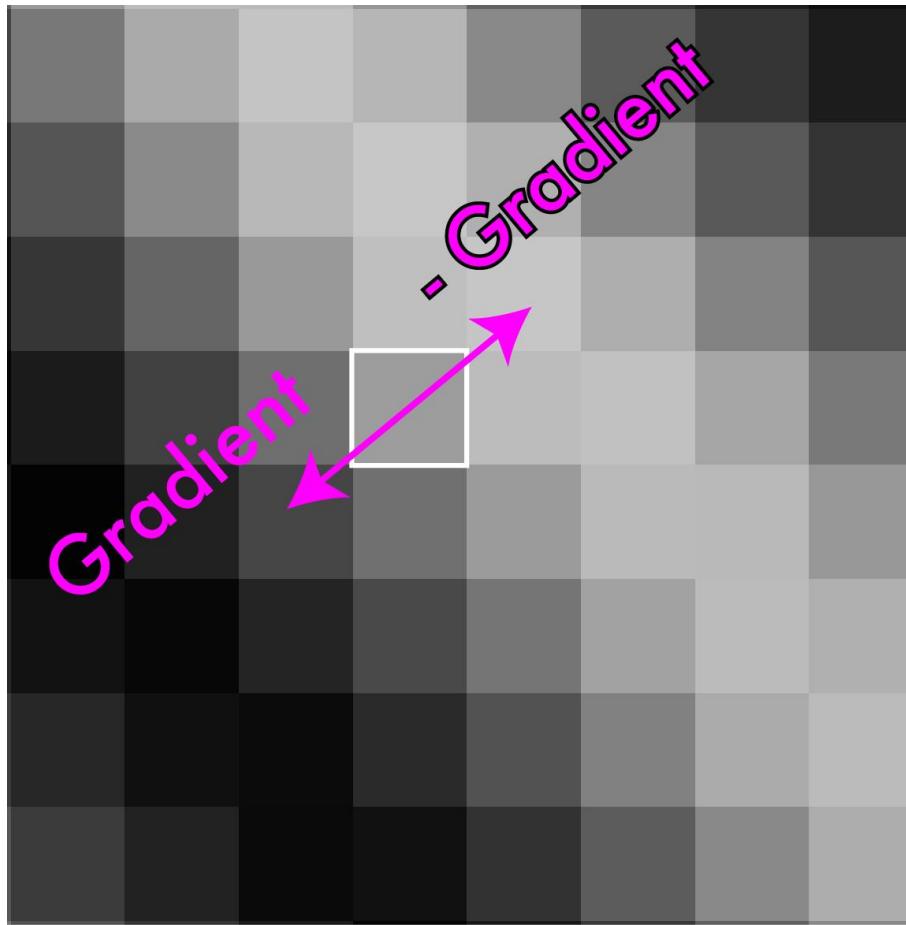
---

# Non-maximum suppression



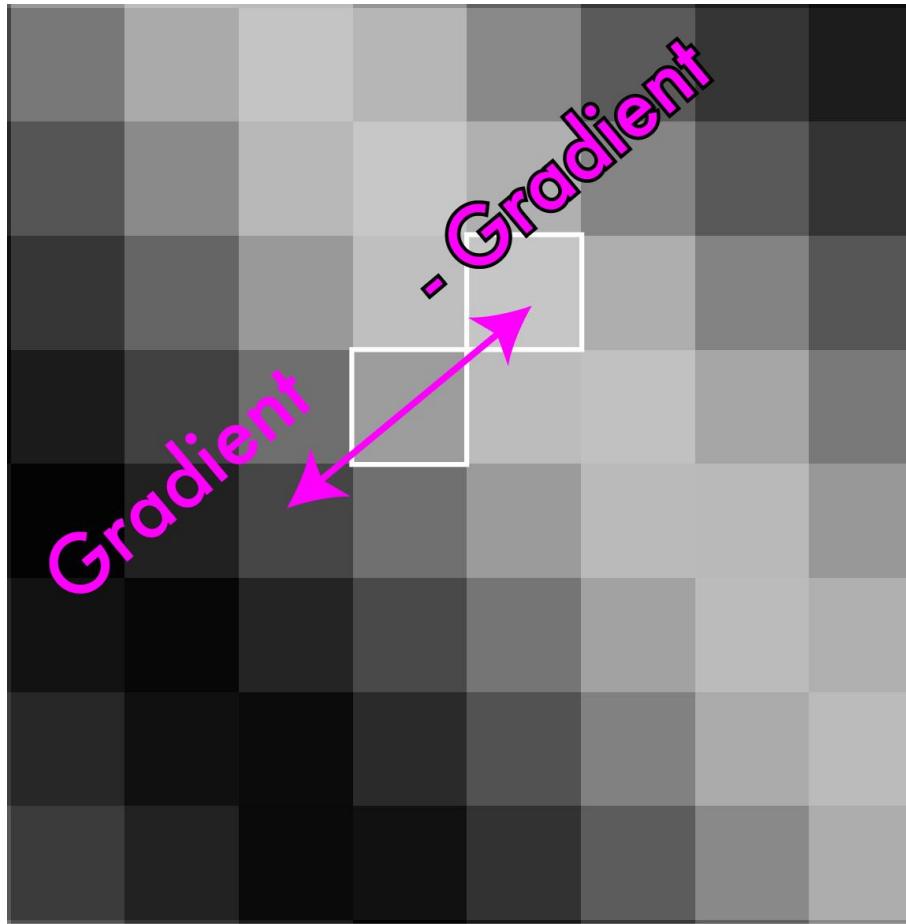
---

# Non-maximum suppression



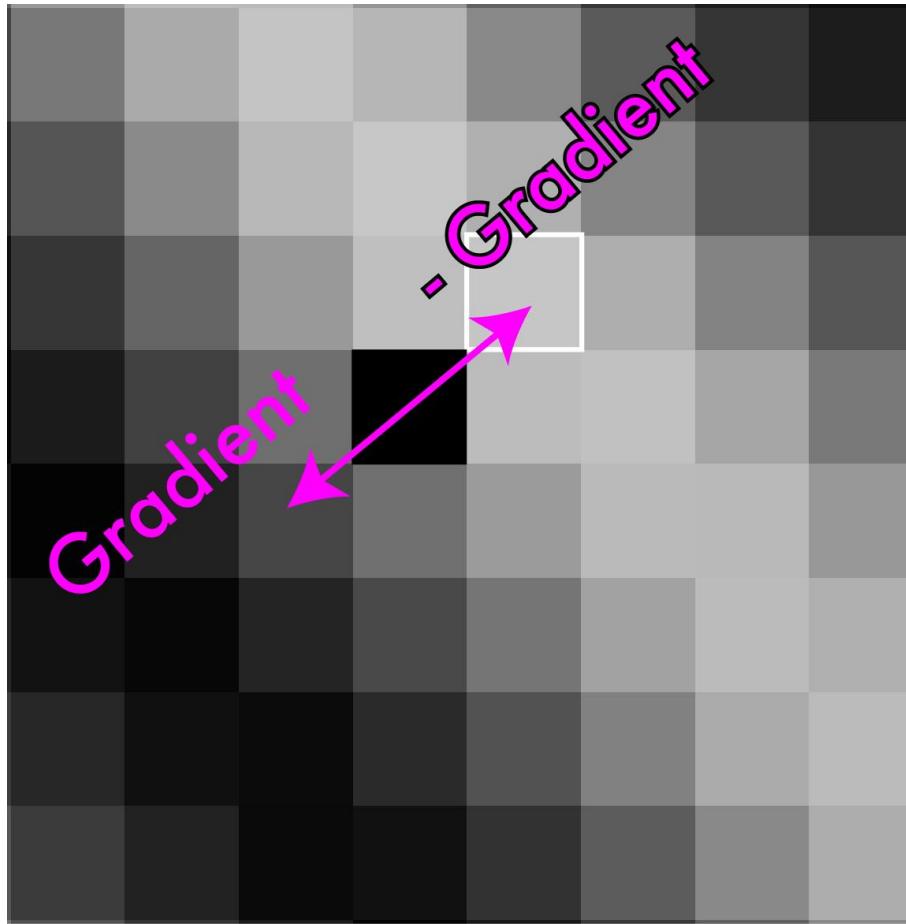
---

# Non-maximum suppression



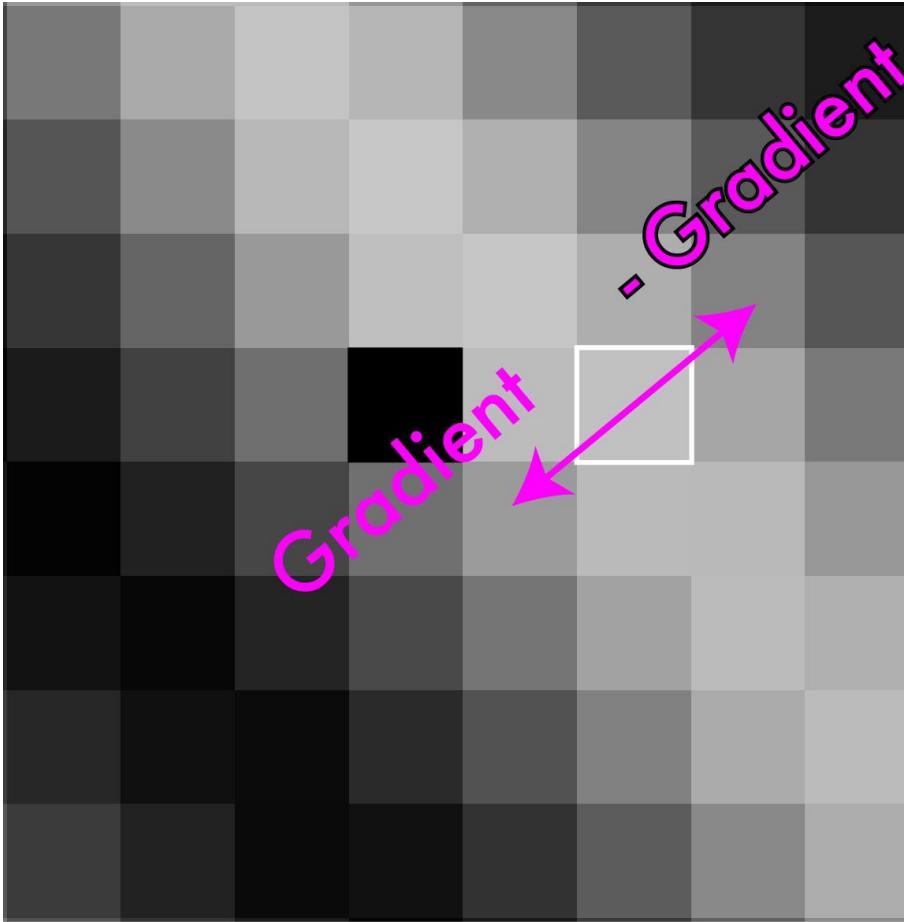
---

# Non-maximum suppression



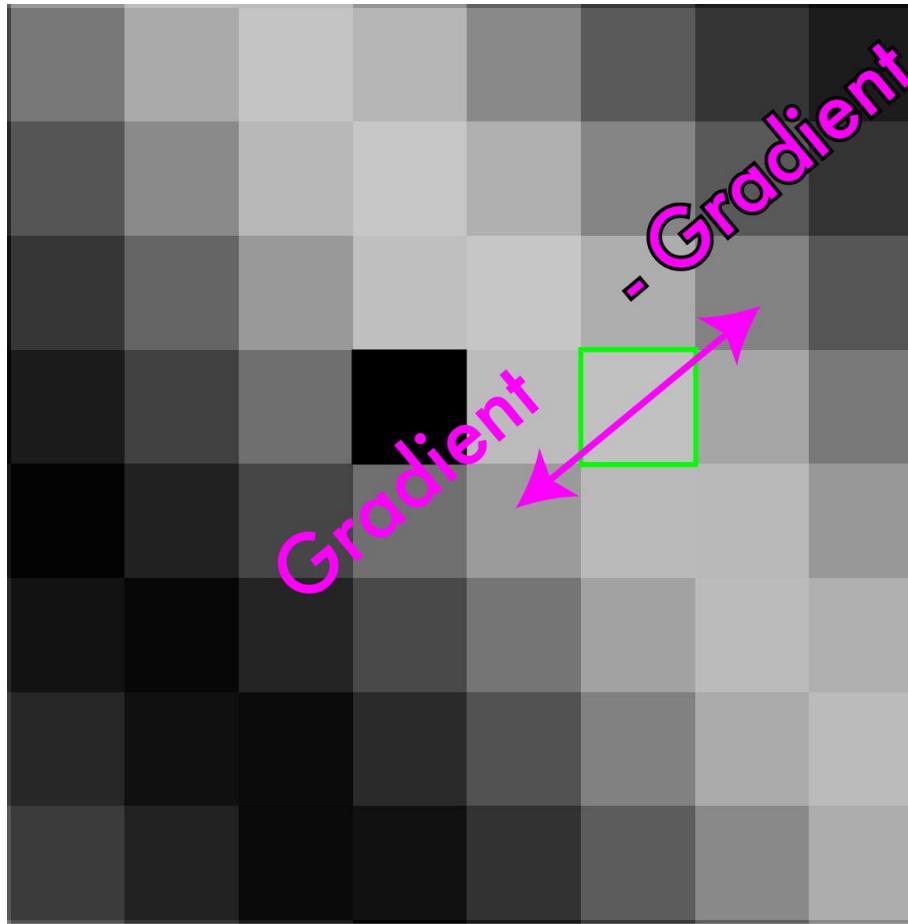
---

# Non-maximum suppression



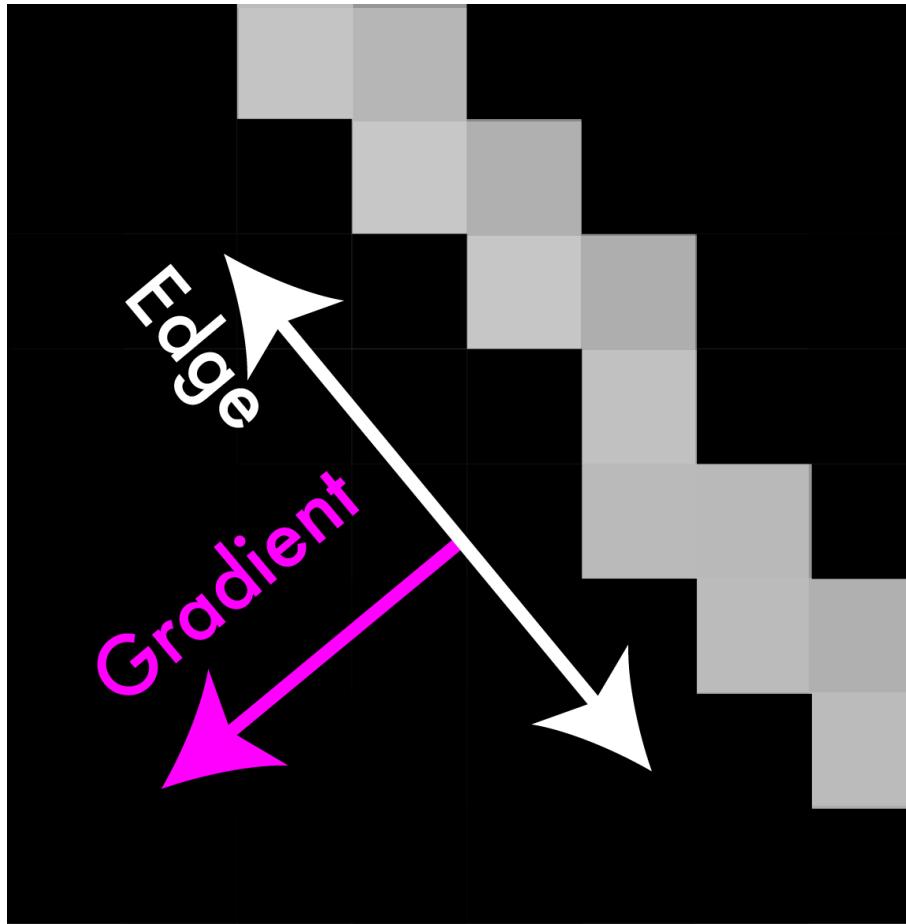
---

# Non-maximum suppression



---

# Non-maximum suppression

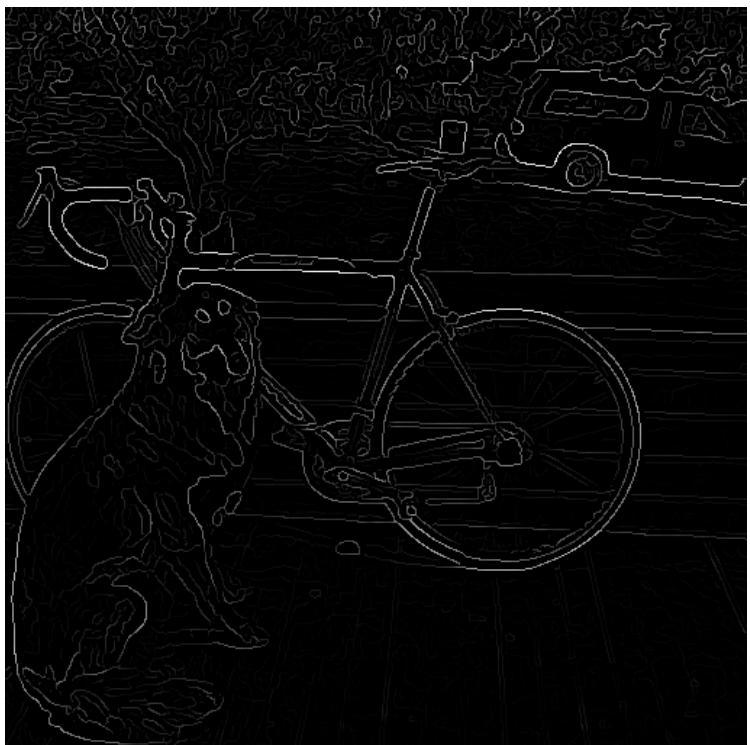


# Non-maximum suppression



# 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?



---

# Connect 'em up!

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



---

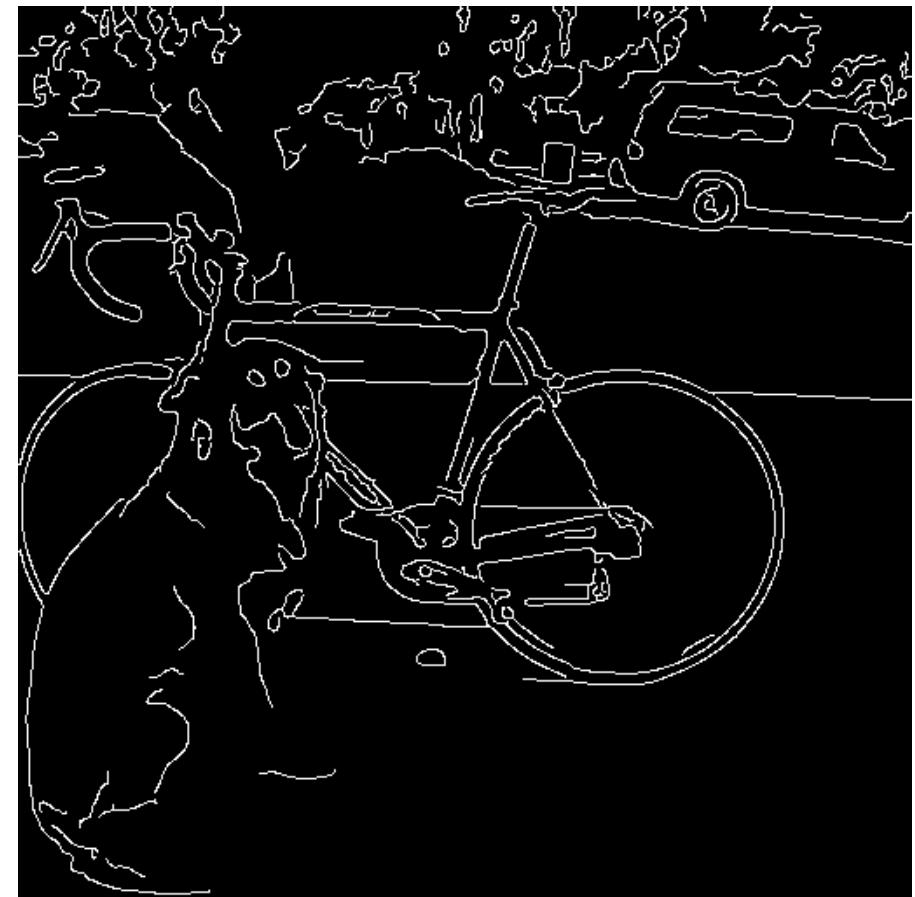
# Canny Edge Detection

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

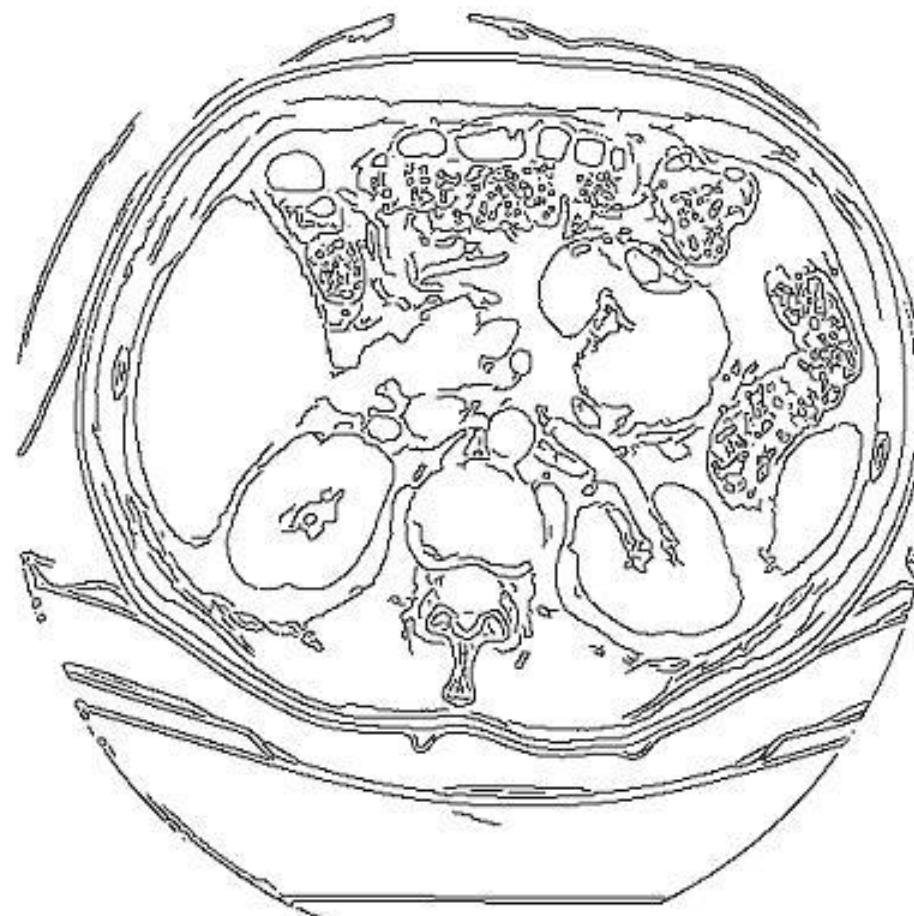
Algorithm:

- 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

# 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...

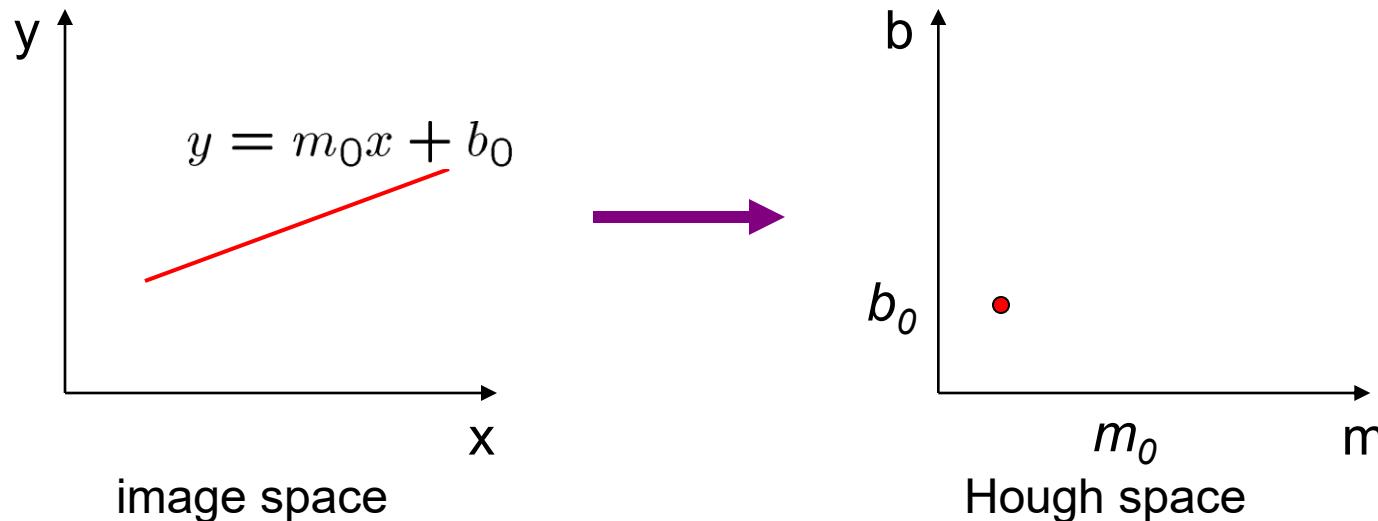


How can we detect *lines* ?

# 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

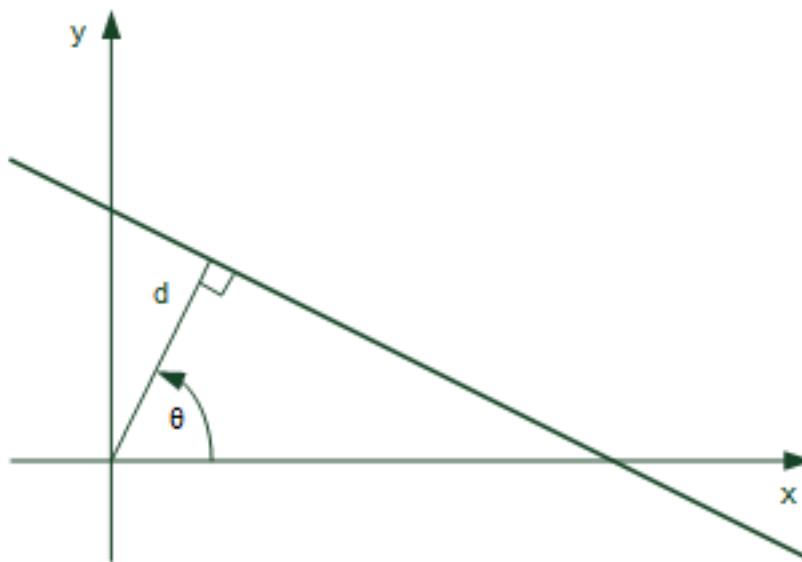


- Connection between image  $(x,y)$  and Hough  $(m,b)$  spaces
  - A line in the image corresponds to a point in Hough space
  - To go from image space to Hough space:
    - given a set of points  $(x,y)$ , find all  $(m,b)$  such that  $y = mx + b$

# 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
- $\theta$  is the angle of this perpendicular with the horizontal.



# Hough transform algorithm

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

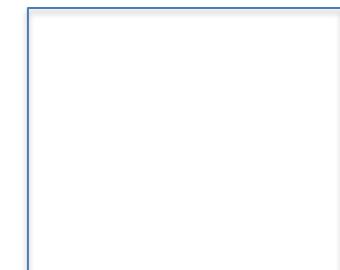
compute gradient magnitude  $m$  and angle  $\theta$

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

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

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

360	-	-	-	-	-	-
.	-	-	-	-	-	-
6	-	-	-	-	-	-
3	4	-	1	-	2	-
0	-	-	-	-	-	-

distance angle 0 10 20 30 40 ...90

PTLIST

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

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

# Line segments from Hough Transform

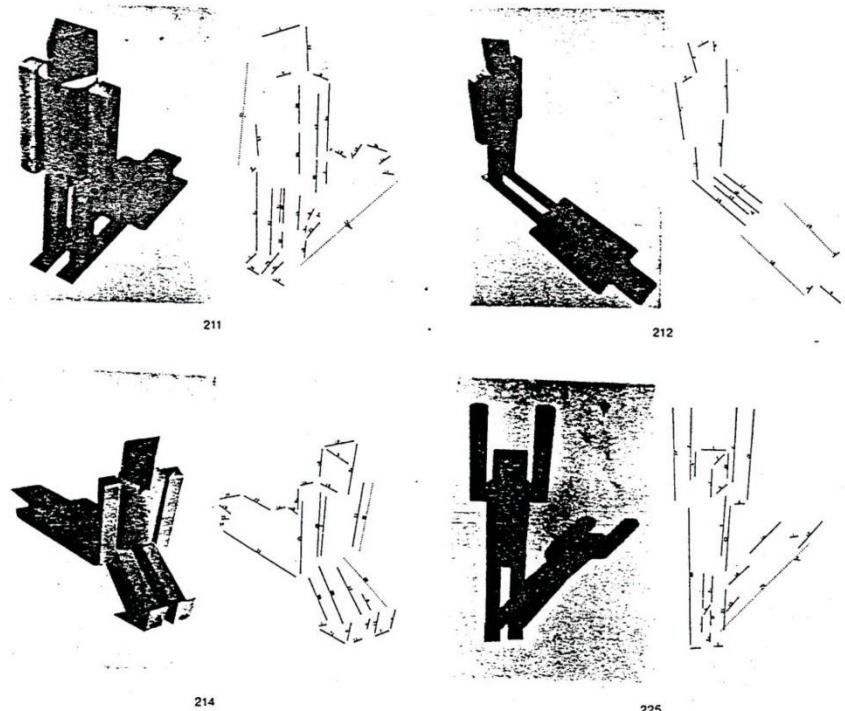


Fig.7. Puppet scenes 211, 212, 214, 225 and the edges recovered by the algorithm.

# 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  $(d, \theta)$  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.

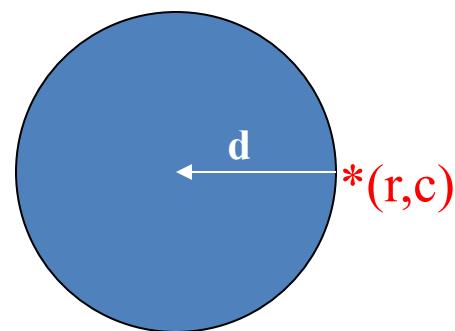
# Hough Transform for Finding Circles

Equations:

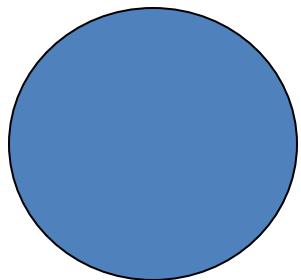
$$\begin{aligned} r &= r_0 + d \sin \theta \\ c &= c_0 - d \cos \theta \end{aligned}$$

r, c, d are parameters

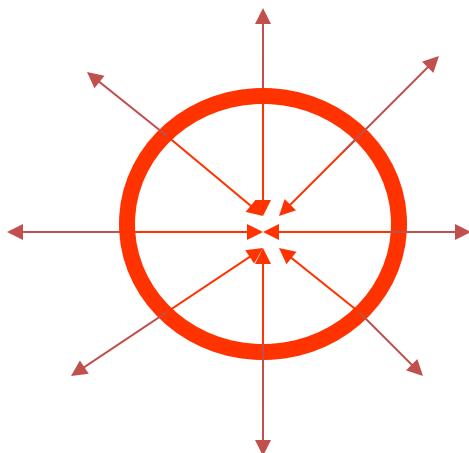
Main idea: The gradient vector at an edge pixel points to the center of the circle.



# Why it works

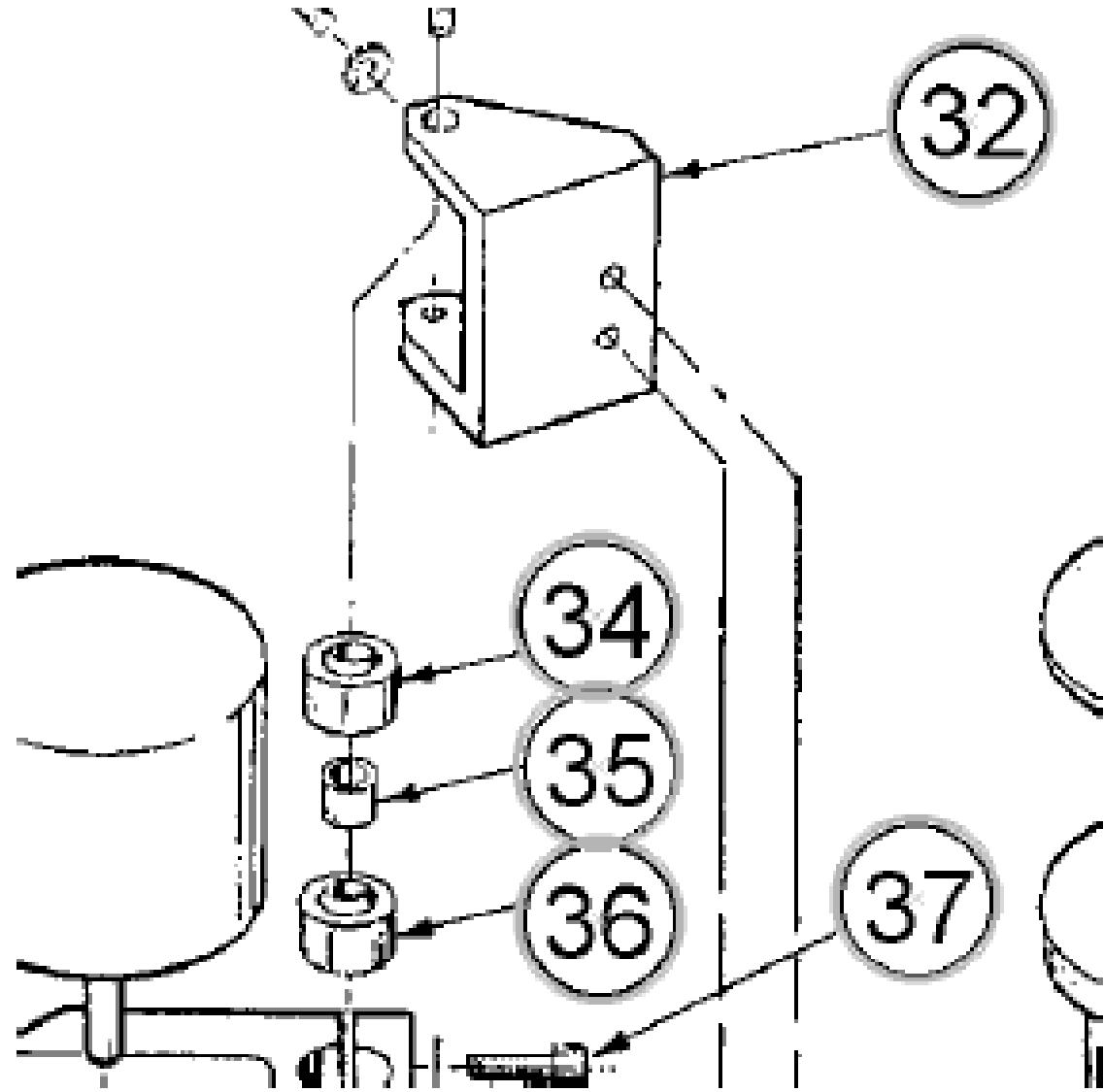


Filled Circle:  
Outer points of circle have gradient direction pointing to center.



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!



# Finding lung nodules (Kimme & Ballard)

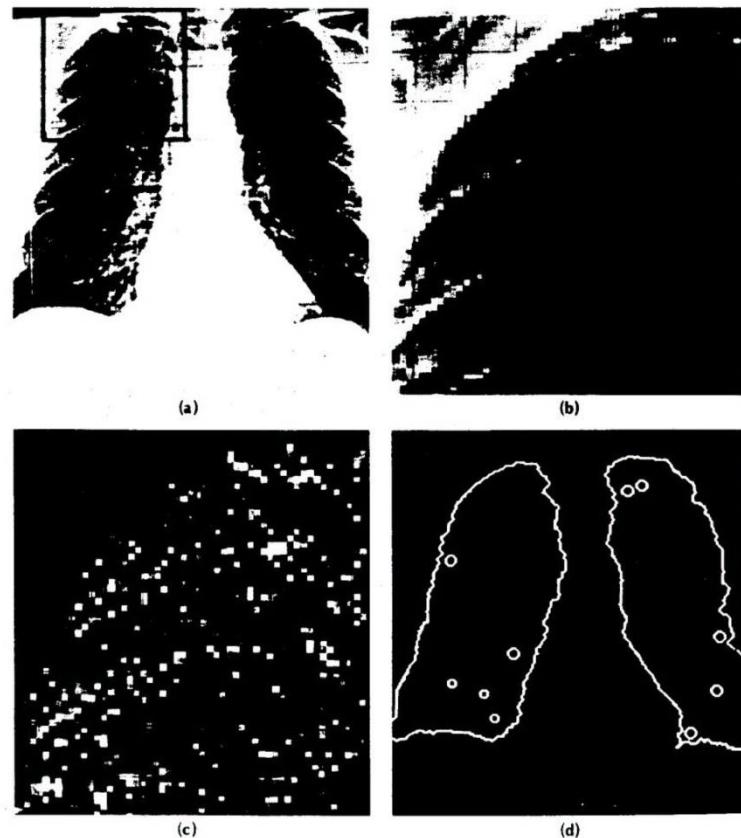


Fig. 4.7 Using the Hough technique for circular shapes. (a) Radiograph. (b) Window. (c) Accumulator array for  $r = 3$ . (d) Results of maxima detection.

# 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.