

The Ancient Secrets



Computer Vision

Logistics:

- Homework 1 is out!
 - Due Thursday
-

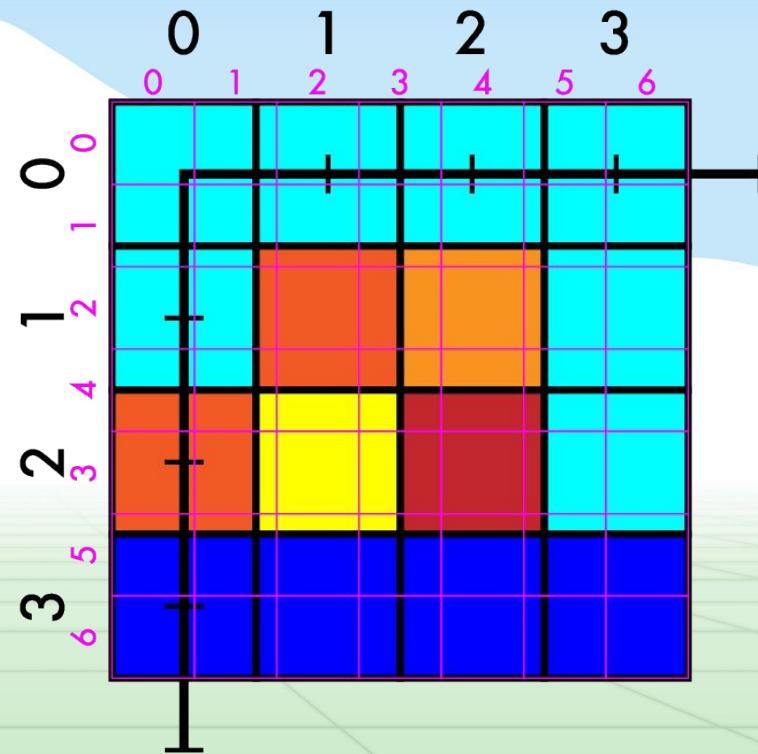
Previously
On



Ancient Secrets
of Computer Vision

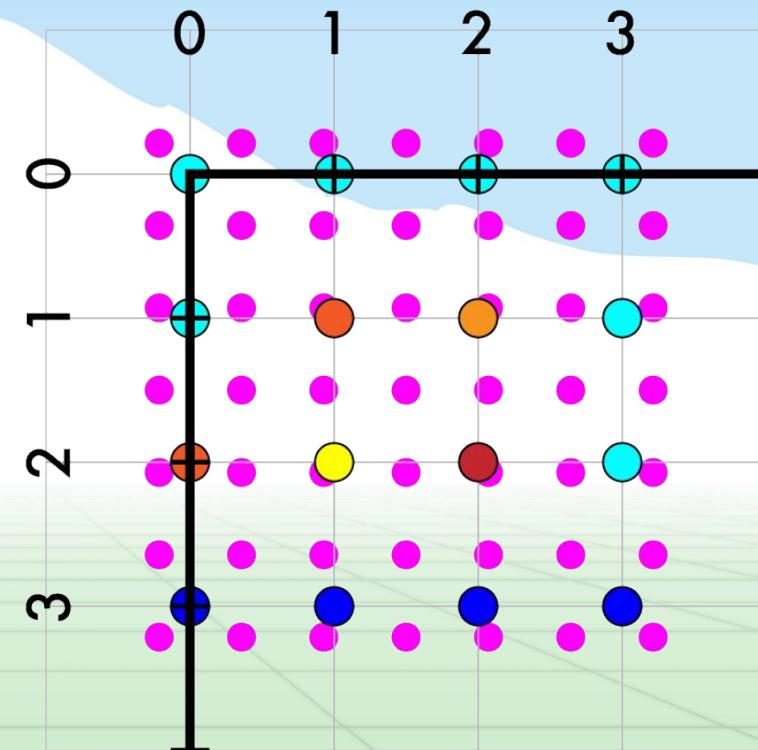
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates



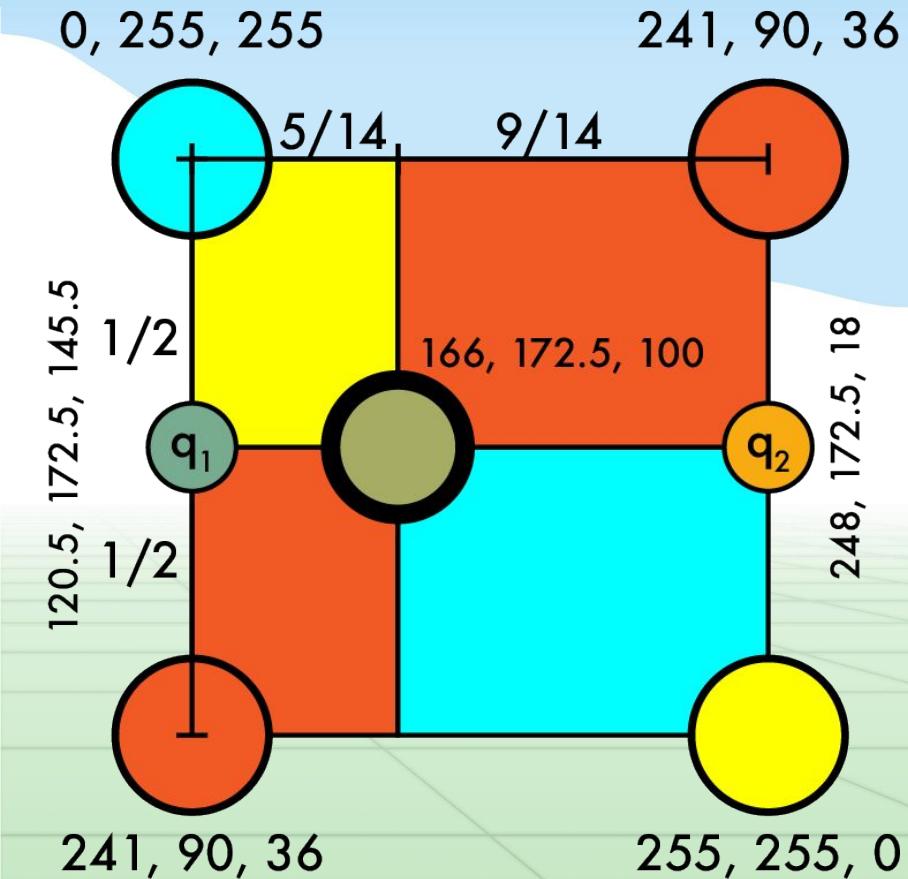
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
 - $4/7 \times - 3/14 = Y$
- Iterate over new pts
 - Map to old coords



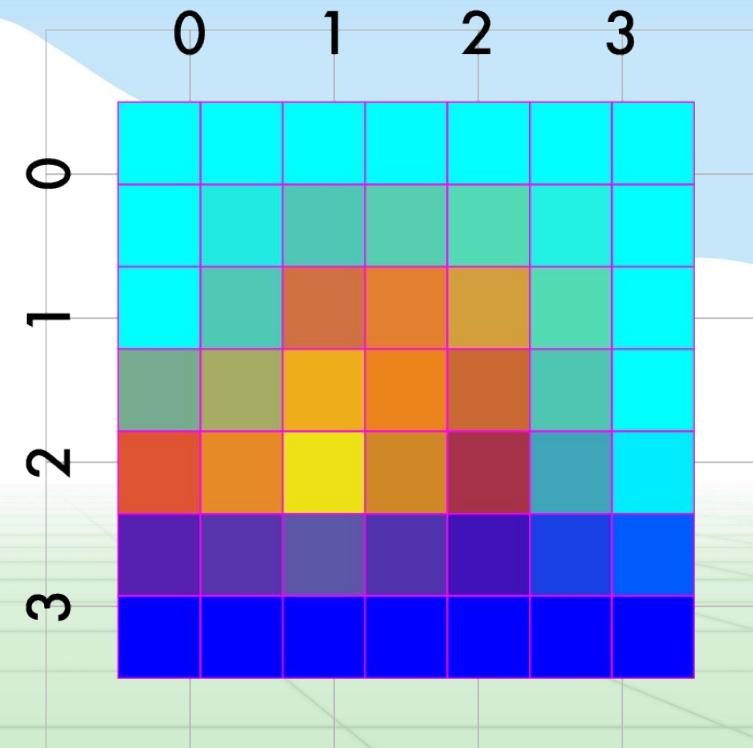
Resize 4x4 -> 7x7

- Create our new image
- Match up coordinates
 - $4/7 X - 3/14 = Y$
- Iterate over new pts
 - Map to old coords
 - $(1, 3) \rightarrow (5/14, 7/14)$
 - Interpolate old values
 - $q = (166, 172.5, 100)$

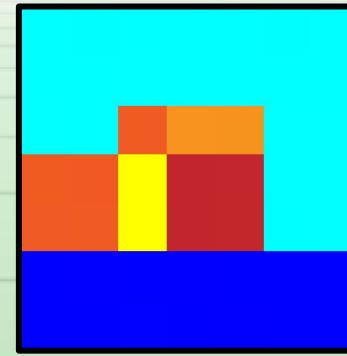
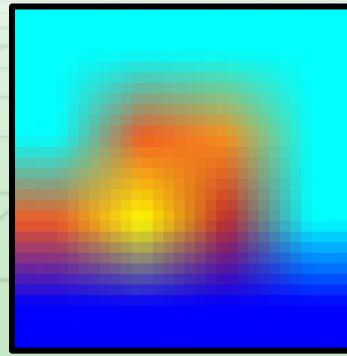
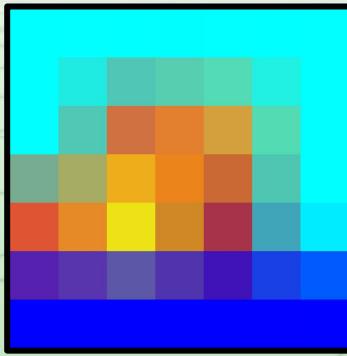
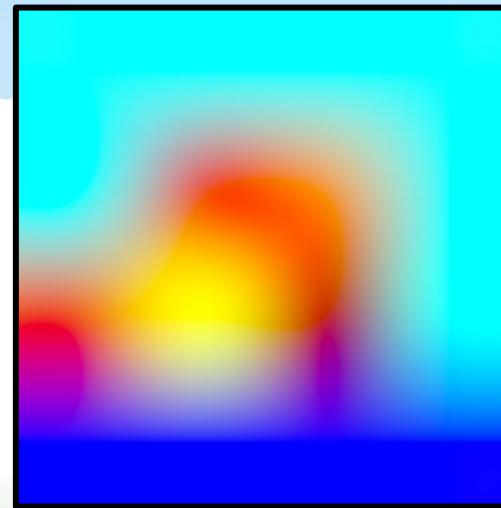
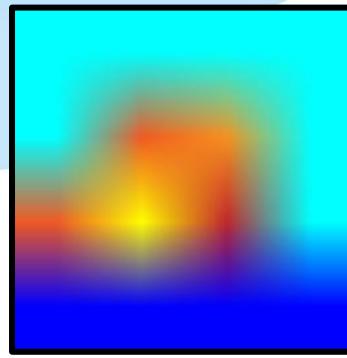
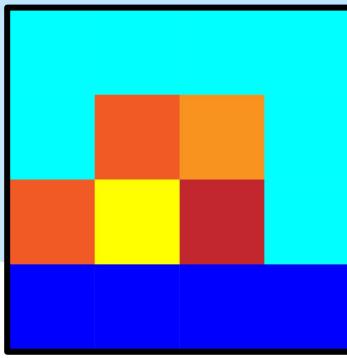


Resize 4x4 -> 7x7

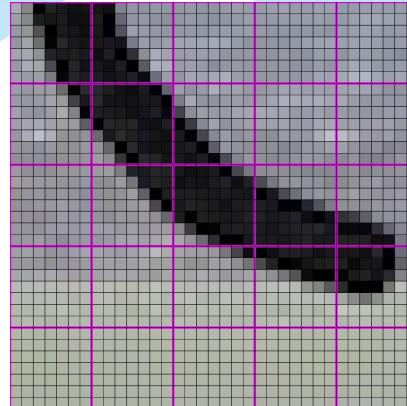
- Create our new image
- Match up coordinates
 - $4/7 X - 3/14 = Y$
- Iterate over new pts
 - Map to old coords
 - $(1, 3) \rightarrow (5/14, 7/14)$
 - Interpolate old values
 - $q = (166, 172.5, 100)$
- Fill in the rest



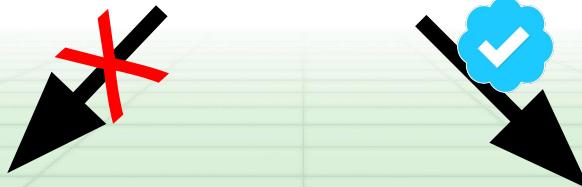
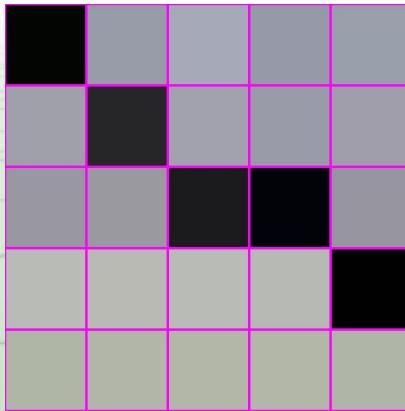
Different methods



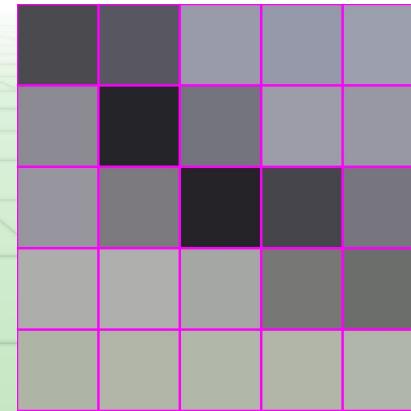
For shrinking interpolation bad, averaging good



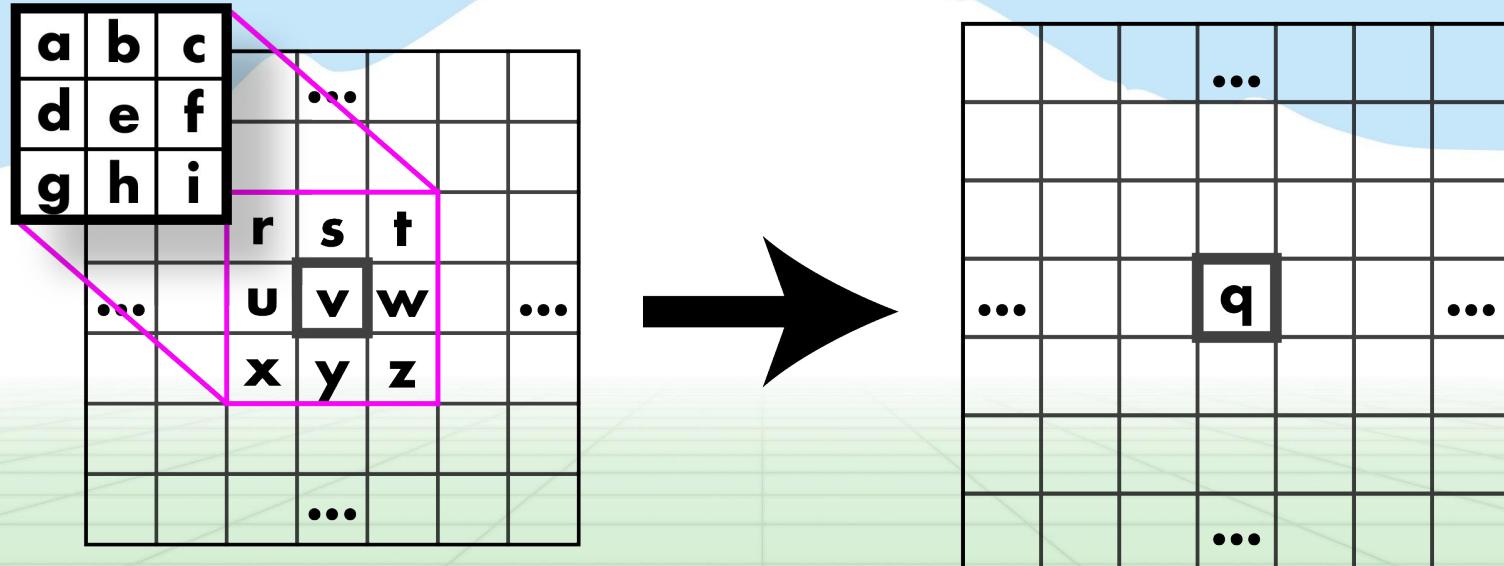
“interpolation”



averaging

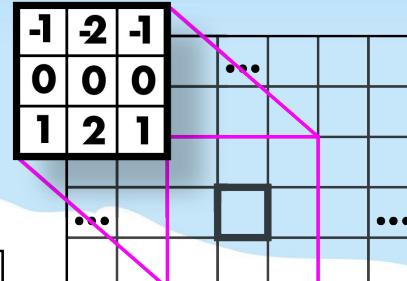
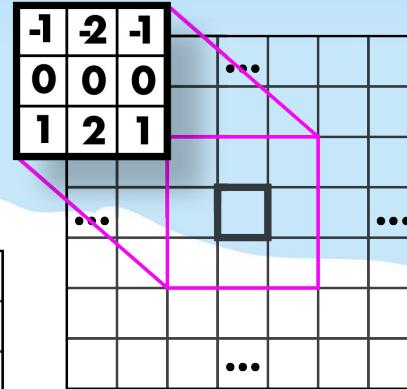
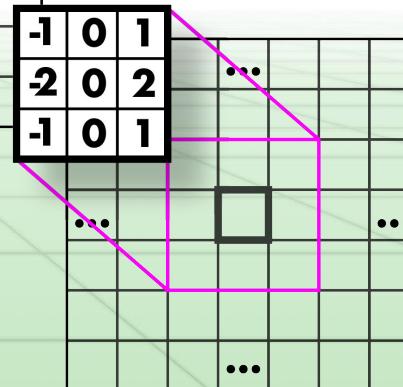
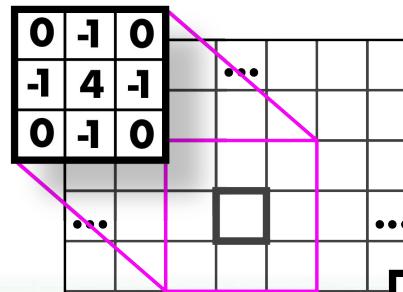
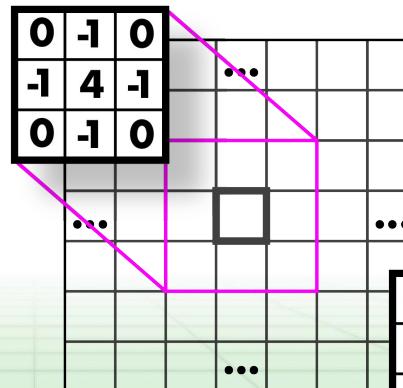
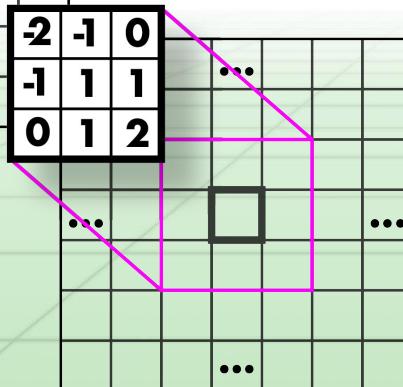
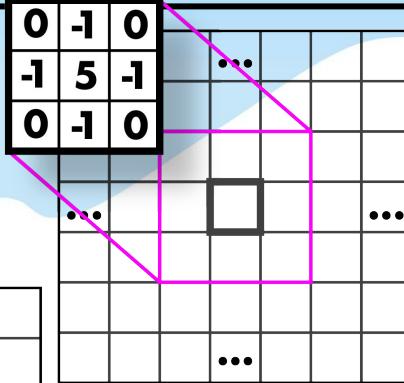
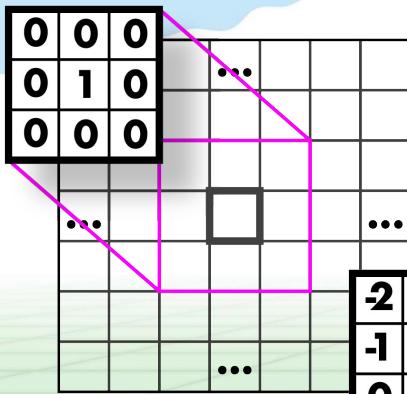


Convolution: Weighted sum over pixels



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

Filters



Chapter Five

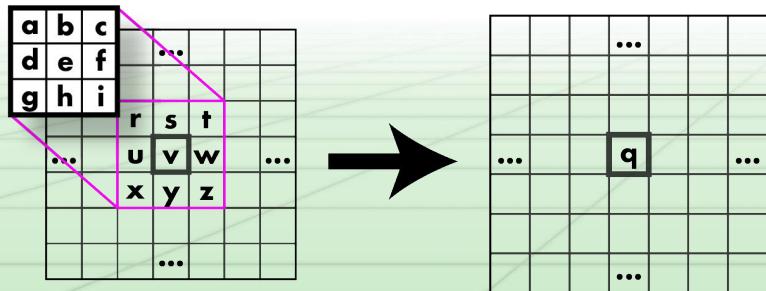


Edges and Features

Cross-Correlation vs Convolution

Cross-Correlation

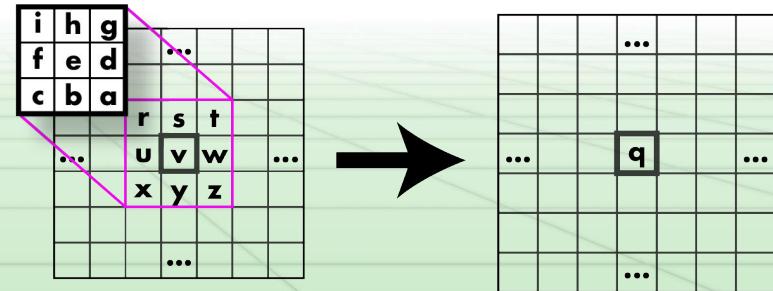
$$\left(\begin{array}{ccccc} a & b & c & & \\ d & e & f & & \\ g & h & i & & \end{array} \right) \star \left(\begin{array}{ccccccc} & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{array} \right)$$



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

Convolution

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



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

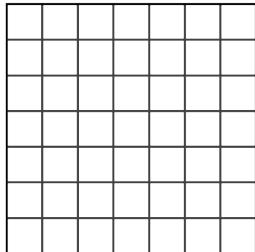
Cross-Correlation vs Convolution

Do this in HW!

Cross-Correlation

a	b	c
d	e	f
g	h	i

★

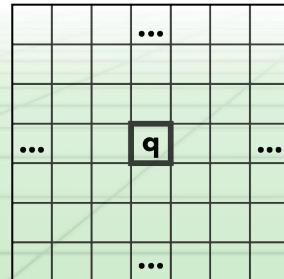


)

a	b	c
d	e	f
g	h	i

r	s	t
u	v	w
x	y	z

...
...
...

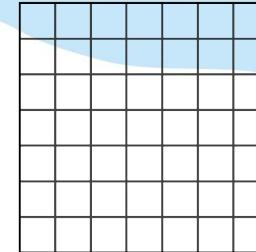


$$q = a \times r + b \times s + c \times t + d \times u + e \times v + f \times w + g \times x + h \times y + i \times z$$

Convolution

a	b	c
d	e	f
g	h	i

*

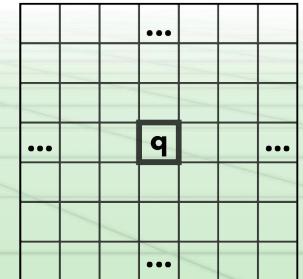


)

i	h	g
f	e	d
c	b	a

r	s	t
u	v	w
x	y	z

...
...
...

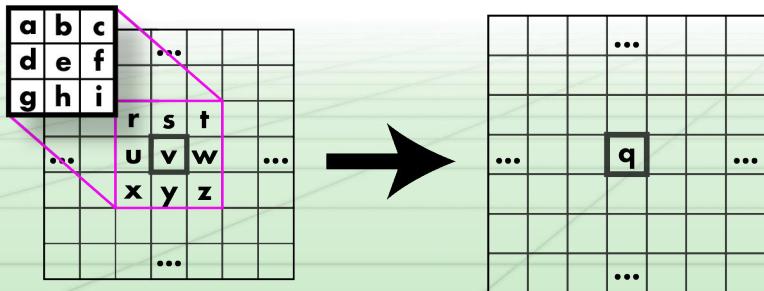


$$q = i \times r + h \times s + g \times t + f \times u + e \times v + d \times w + c \times x + b \times y + a \times z$$

Cross-Correlation vs Convolution

Cross-Correlation

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

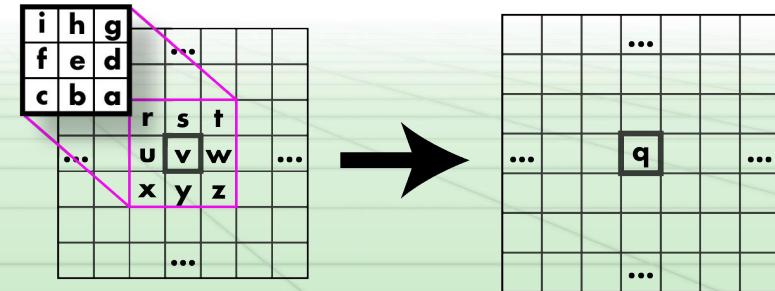


$$q = a \times r + b \times s + c \times t + d \times u + e \times v + f \times w + g \times x + h \times y + i \times z$$

These are nicer mathematically

Convolution

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



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

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
 - $x(A * B) = (xA) * B = A * (xB)$

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
- ...

All of computer vision
is **convolutions**
(basically)

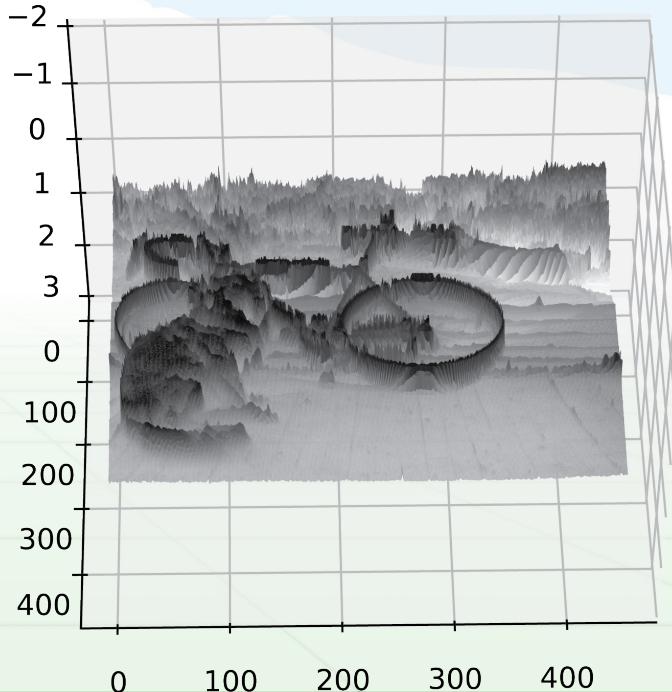
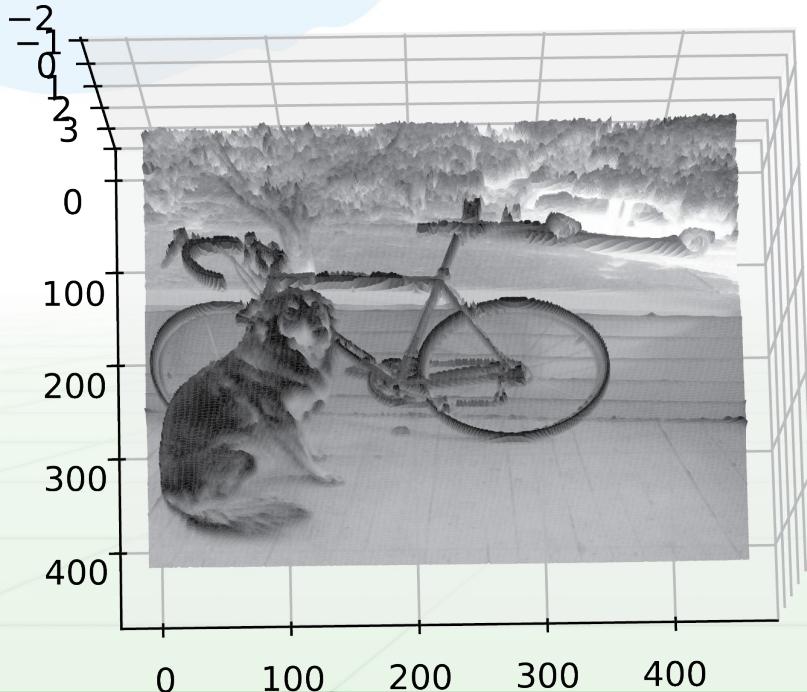
So what can we do with these convolutions anyway?

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

All of computer vision
is **convolutions**
(basically)

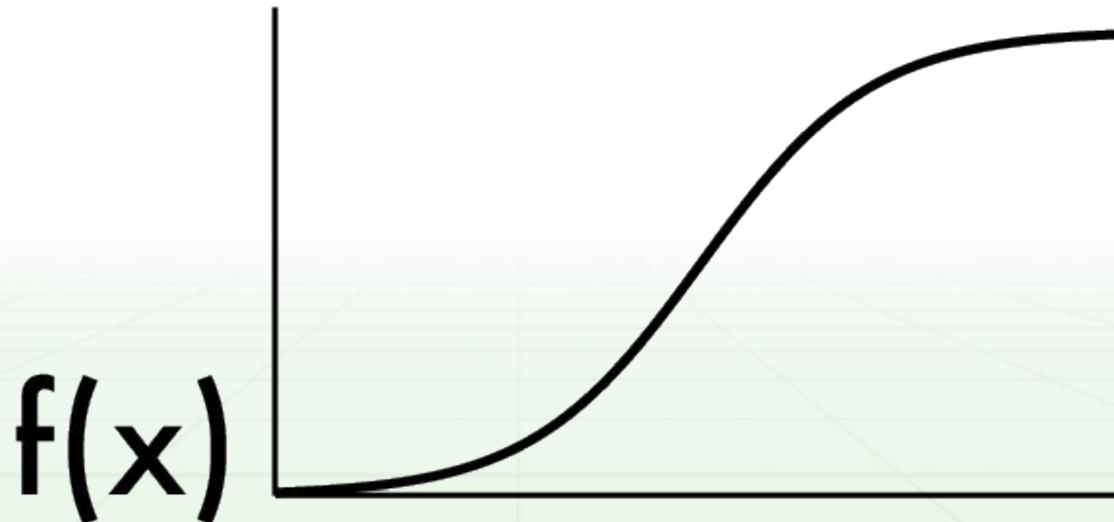
What's an edge?

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



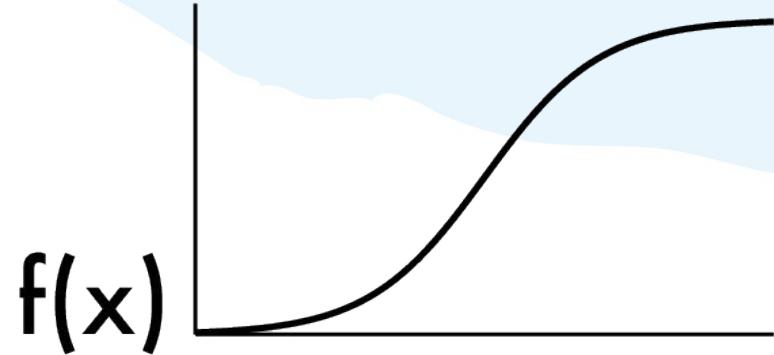
What's an edge?

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

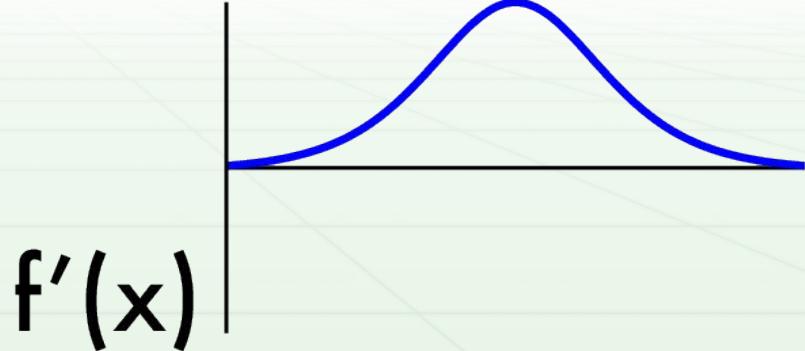


Finding edges

- Could take derivative
- Edges = high response



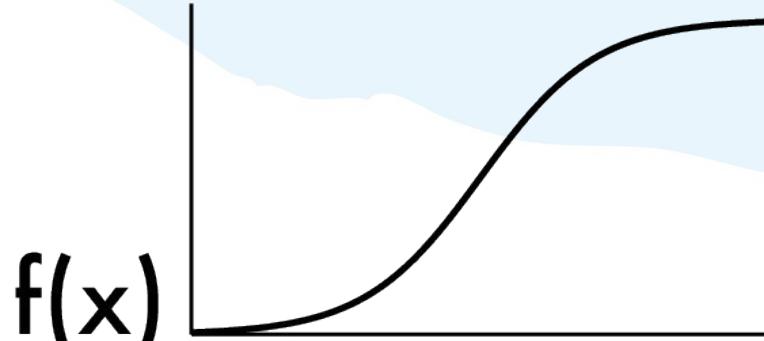
$f(x)$



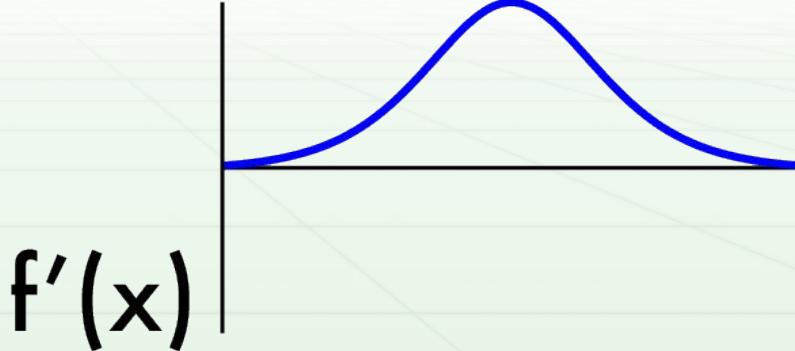
$f'(x)$

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?



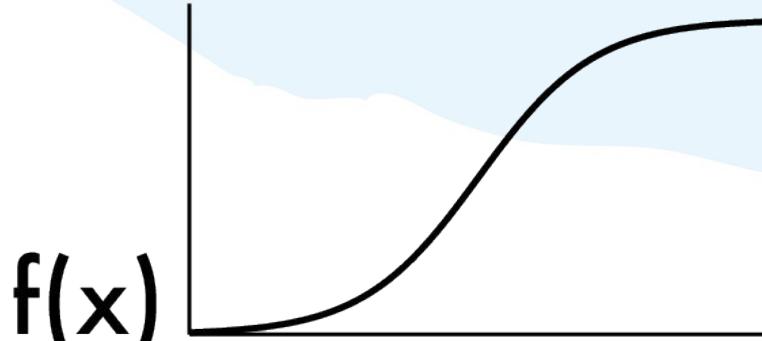
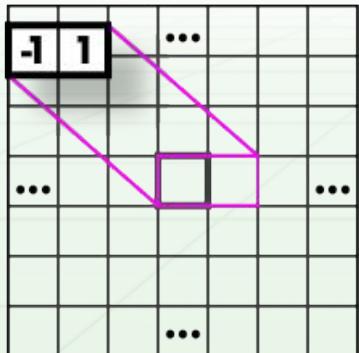
$f(x)$



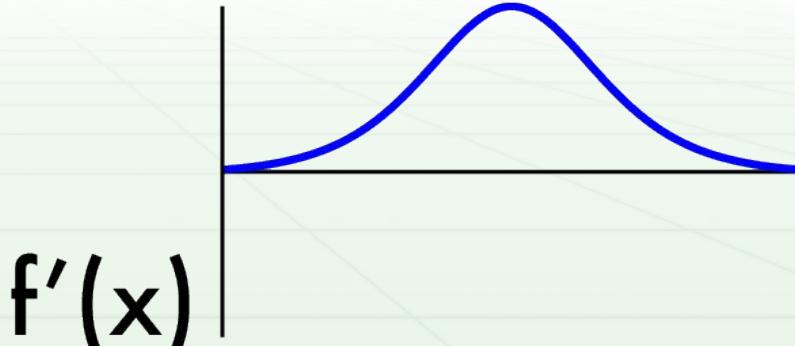
$f'(x)$

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?



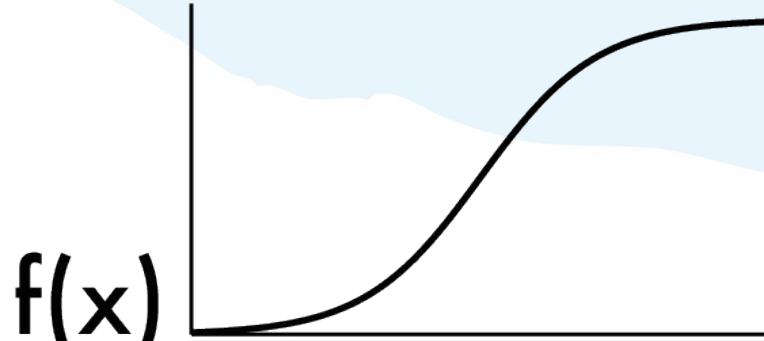
$f(x)$



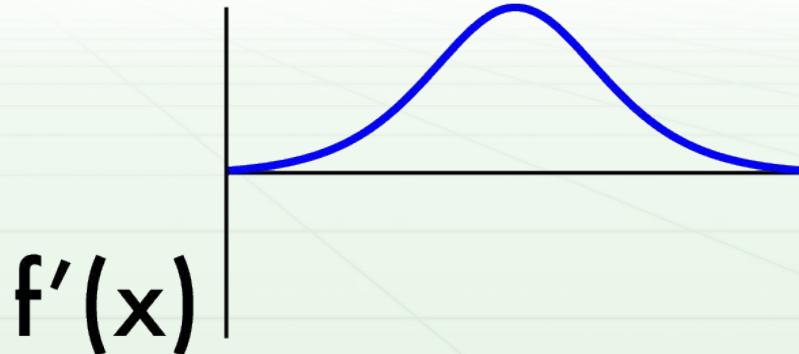
$f'(x)$

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?



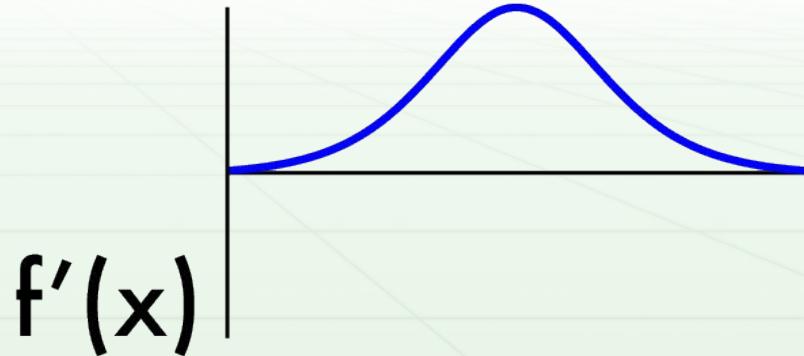
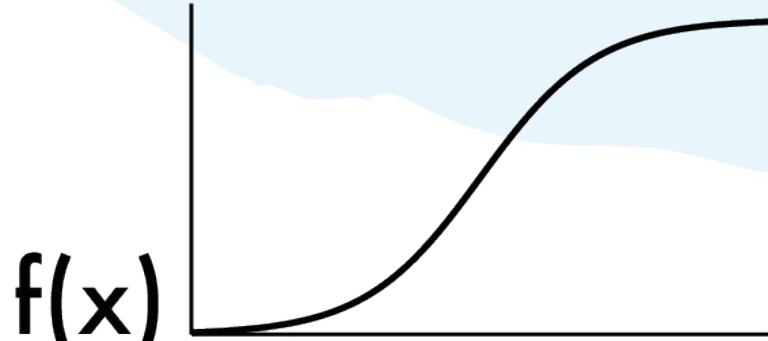
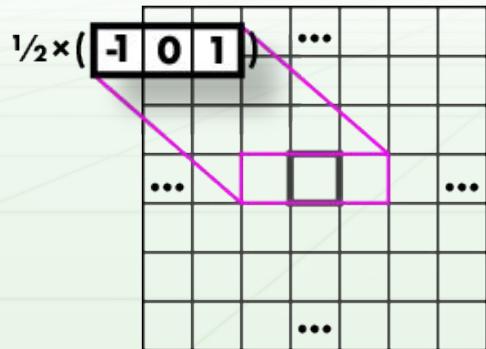
$f(x)$



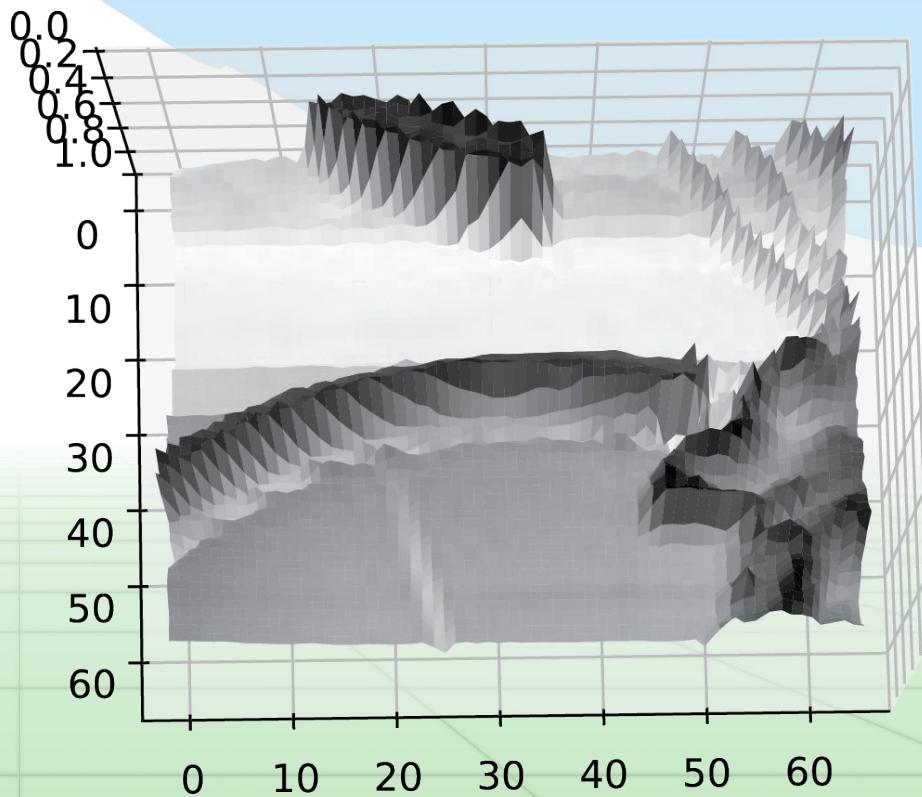
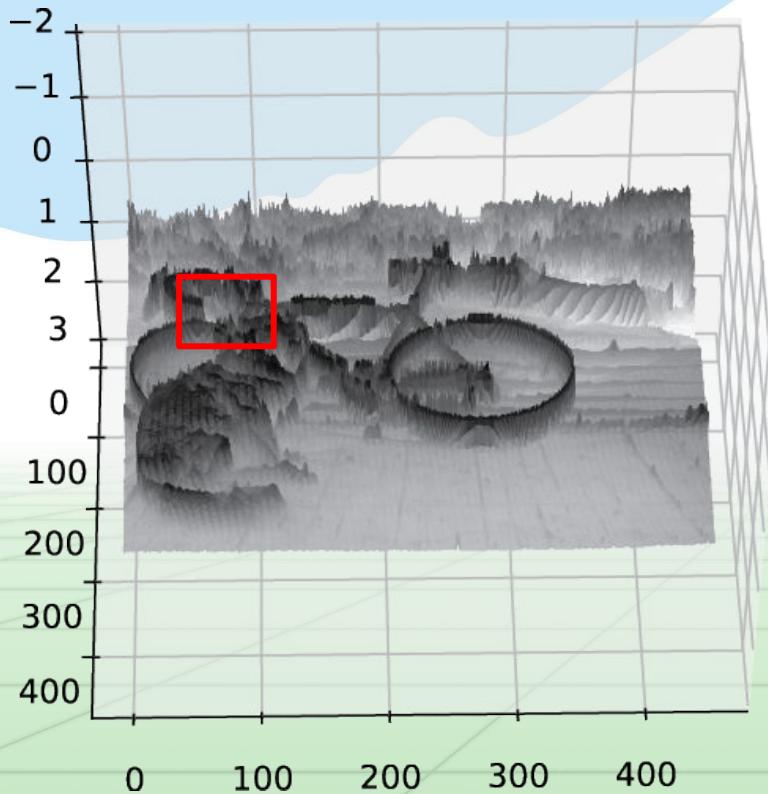
$f'(x)$

Image derivatives

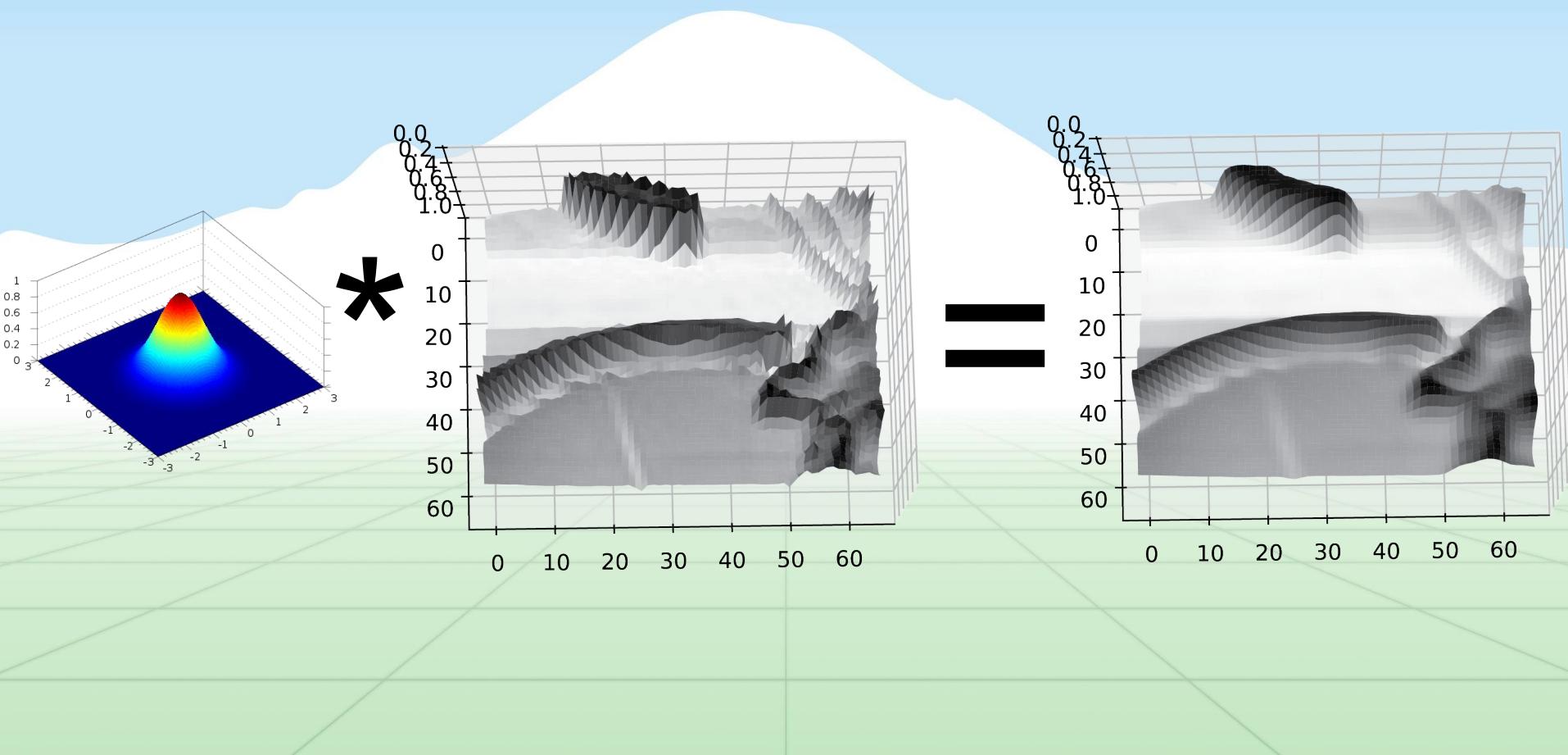
- Recall:
 - $f'(a) = \lim_{h \rightarrow 0} \frac{f(a + h) - f(a)}{h}$.
- We don't have an "actual" Function, must estimate
- Possibility: set $h = 2$
- What will that look like?



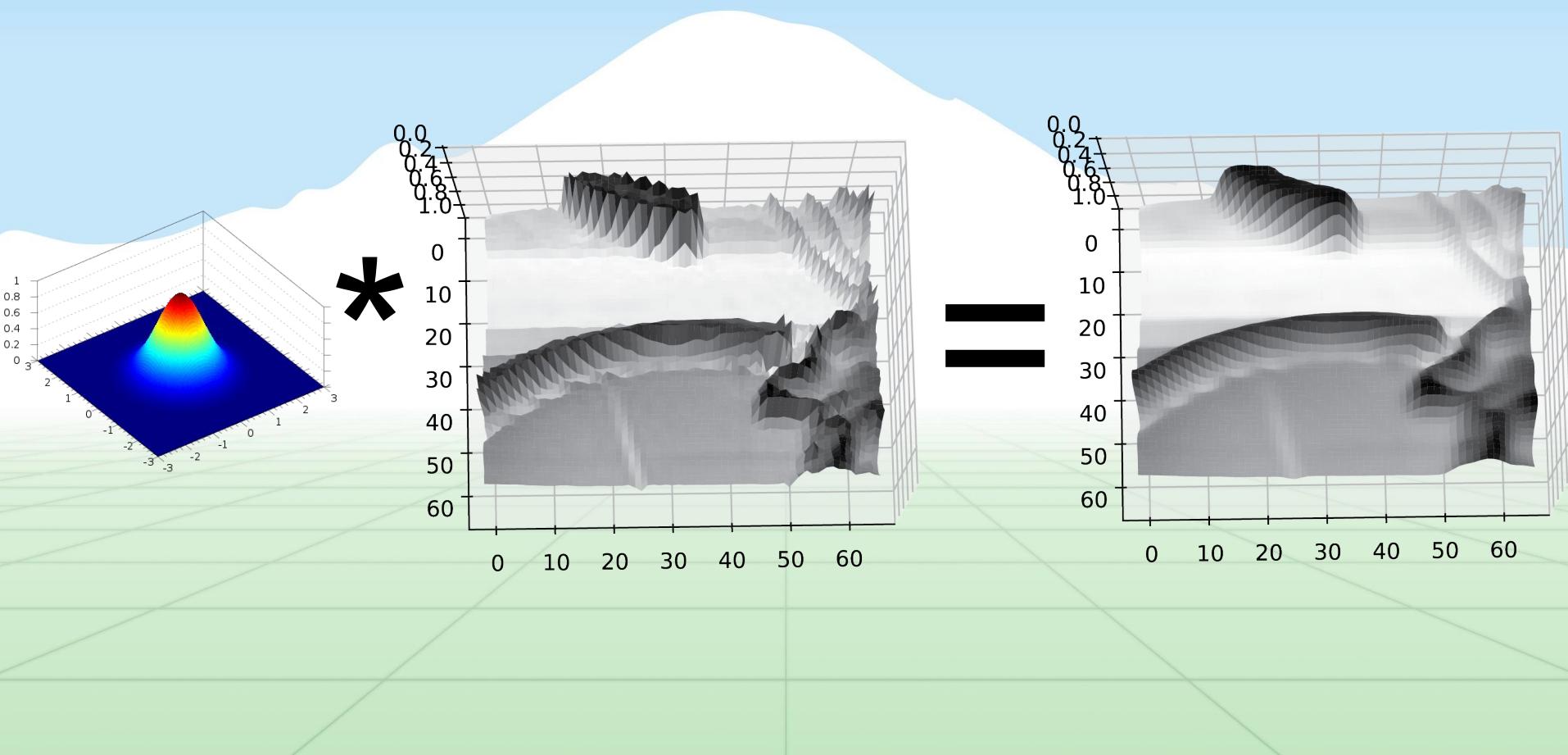
Images are noisy!



But we already know how to smooth

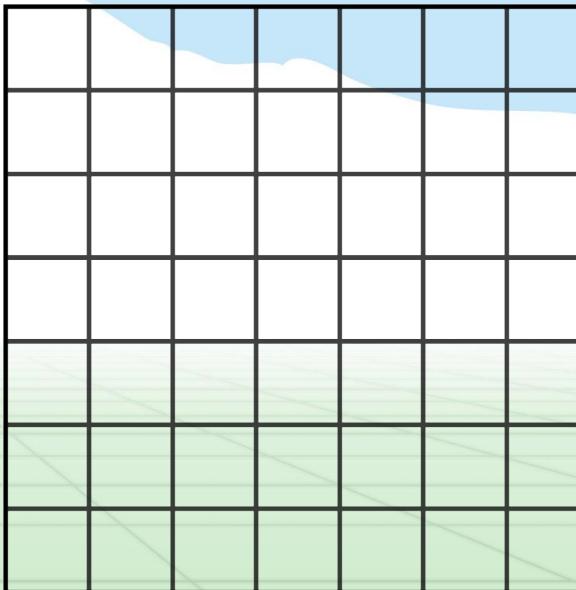


But we already know how to smooth



Smooth first, then derivative

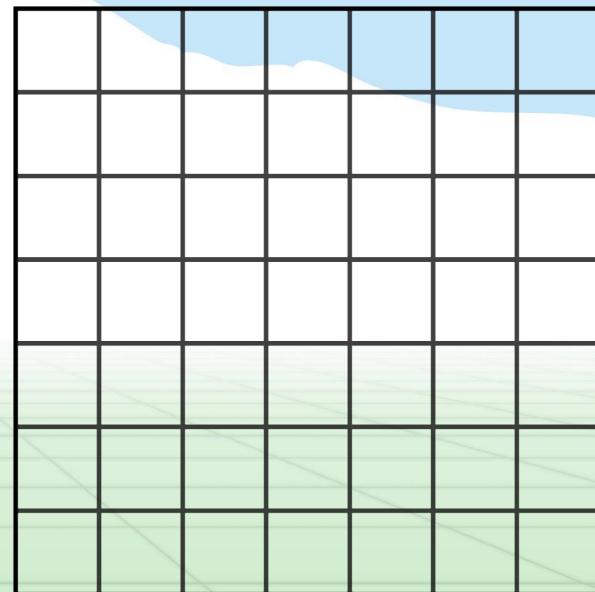
$$\frac{1}{2} \times (-1 \ 0 \ 1) * \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} *$$



)

Smooth first, then derivative

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



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 a convolution operation. On the left, a 3x3 input matrix is shown:

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

On the right, a 1x3 kernel is shown:

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

A pink arrow points from the top-left cell of the input matrix to the kernel, indicating the receptive field of that output unit. The result of the multiplication is a scalar value:

$$\frac{1}{2} \times 2 = 1$$

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

Smooth first, then derivative

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

A diagram illustrating a convolution operation. At the top, a 1x3 kernel $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ is shown next to a 3x3 input matrix:

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

The bottom part shows the result of applying the kernel to the input. A pink arrow points from the kernel to the center cell of the input matrix, which is highlighted with a pink border. A large black arrow points to the resulting 2x2 output matrix:

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

Smooth first, then derivative

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

The diagram illustrates a convolution operation. A 3x3 input matrix is multiplied by a 1x3 kernel. The result is scaled by $\frac{1}{2}$.

The input matrix is:

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

The kernel is:

$$(-1 \ 0 \ 1)$$

The result of the convolution is:

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

Smooth first, then derivative

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

A diagram illustrating a convolution operation. A 3x3 input matrix (gray) is multiplied by a 1x3 kernel (black). The result is a 2x3 output matrix (white). The input matrix has values 1, 2, 1 in the top row, 2, 4, 2 in the middle row, and 1, 2, 1 in the bottom row. The kernel has values -1, 0, 1 in the top row. The output matrix has values 2, 0, -2 in the top row, and 4 in the bottom row.

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

An arrow points from the input matrix to the output matrix, indicating the result of the convolution operation.

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

Smooth first, then derivative

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

A diagram illustrating a convolution operation. A 3x3 input matrix (bottom) is multiplied by a 1x3 kernel (top). The result is a 2x3 output matrix (right).

-1	0	1	
2	4	2	
1	2	1	

1	2	1
2	4	2
1	2	1

2	0	-2
4	0	

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

Smooth first, then derivative

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

The diagram illustrates a convolution operation. A 3x3 input matrix (gray background) is multiplied by a 1x3 kernel (black border). The result is scaled by $\frac{1}{2}$.

$\frac{1}{2} \times$

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

Smooth first, then derivative

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

A diagram illustrating a convolution operation. A 3x3 input matrix (with values 1, 2, 1; 2, 4, 2; 1, 2, 1) is multiplied by a 1x3 kernel (with values -1, 0, 1). The result is a 2x2 output matrix (with values 2, 0, -2; 4, 0, -4).

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

Smooth first, then derivative

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

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

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 (bottom) is multiplied by a 3x3 kernel (middle). The result is a 2x2 output matrix (top), which is then scaled by $\frac{1}{2}$.

$\frac{1}{2} \times$

1	2	1
2	4	2
1	2	1

-1	0	1
1	2	1
1	2	1

2	0	-2
4	0	-4
2	0	-2

Sobel filter! Smooth & derivative

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

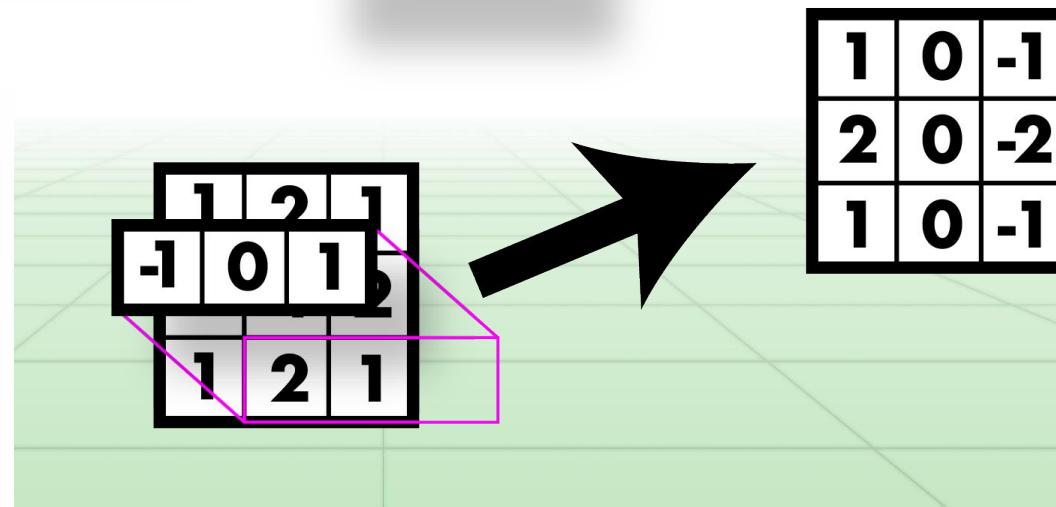
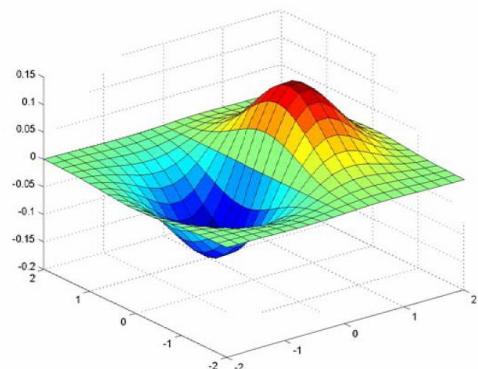
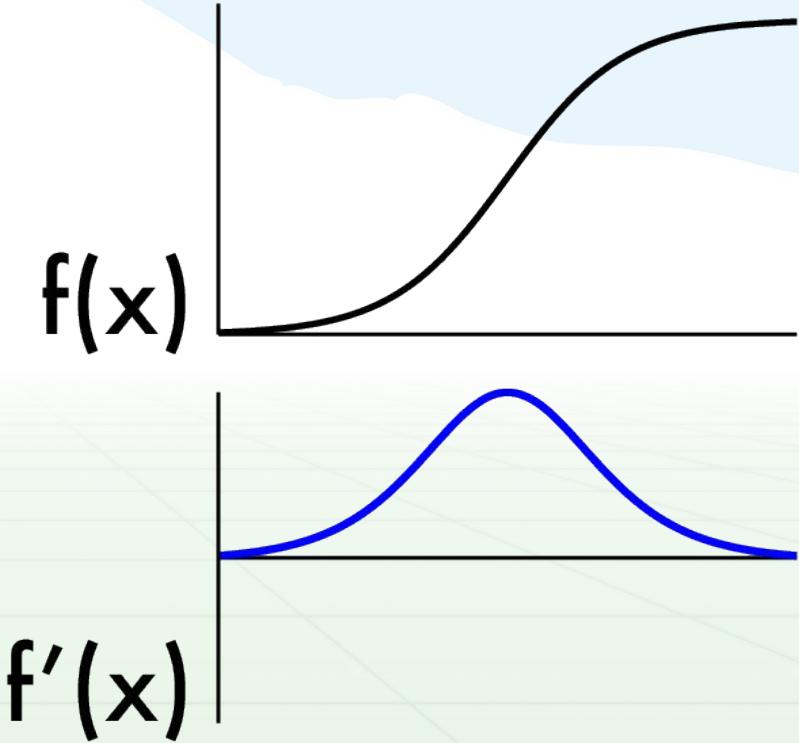


Image derivatives

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

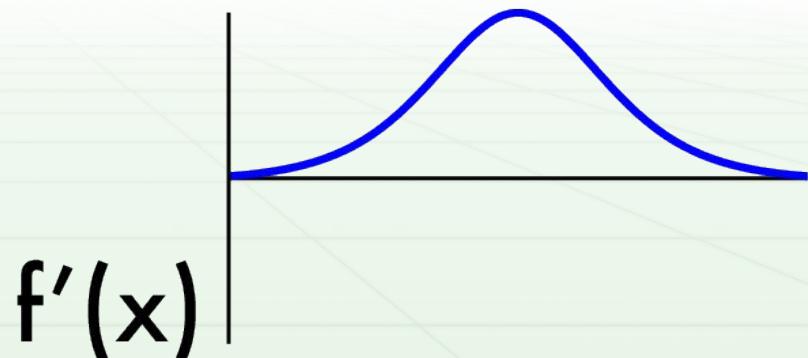
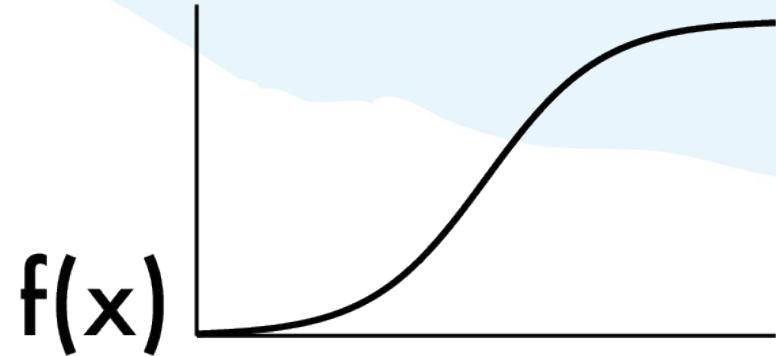
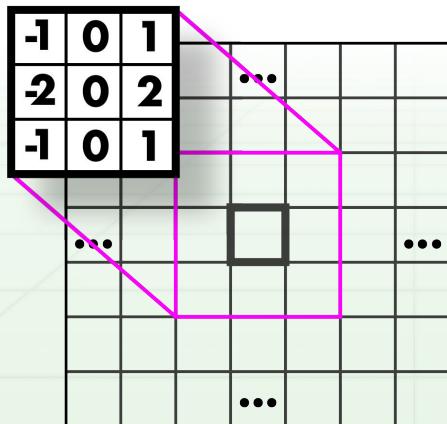
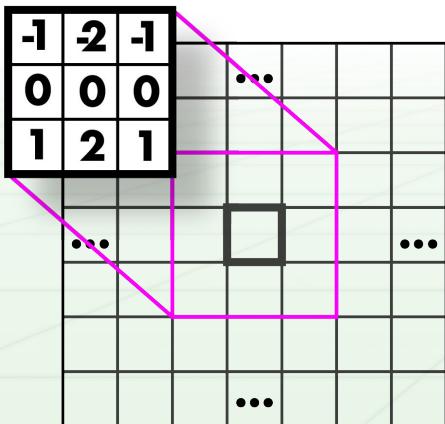
$$\frac{1}{2} \times \left(\begin{bmatrix} -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right)$$

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



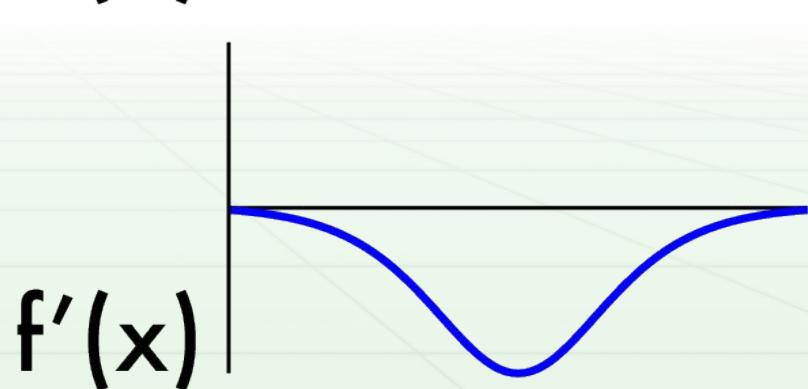
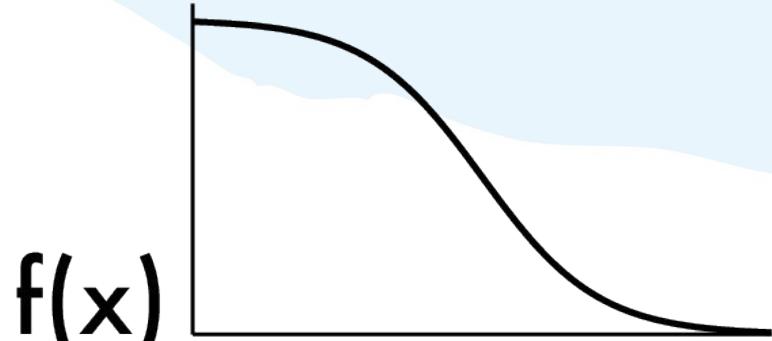
Finding edges

- Could take derivative
- Find high responses
- Sobel filters!
- But...



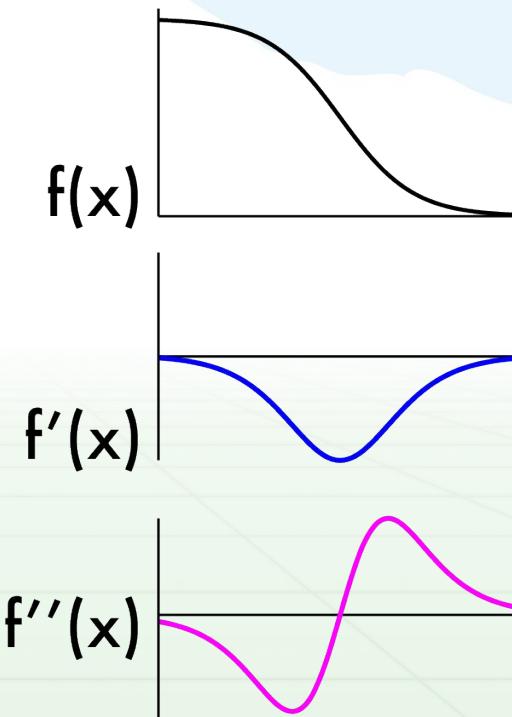
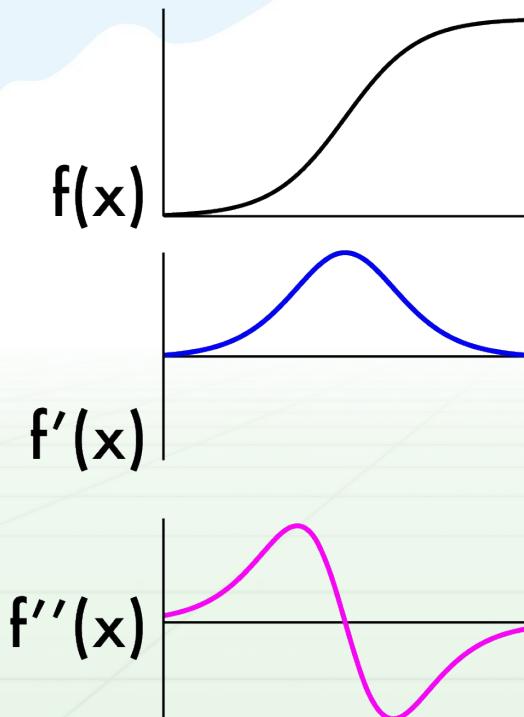
Finding edges

- 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

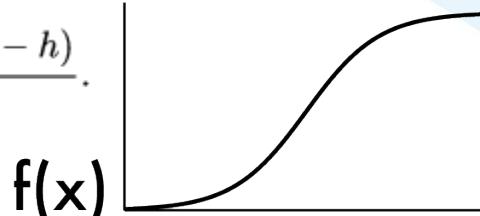
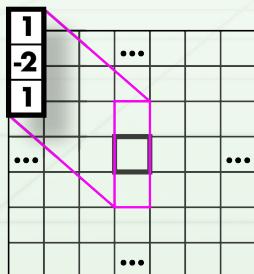
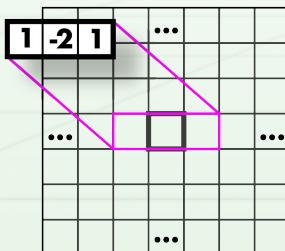
- Recall:

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

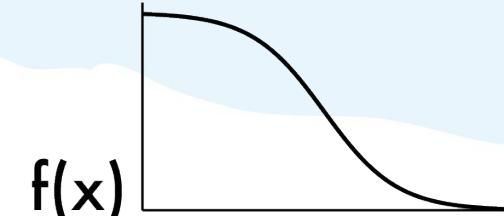
- Laplacian:

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

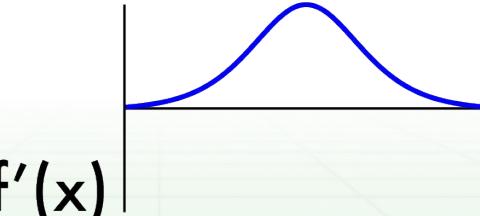
- Again, have to estimate $f''(x)$:



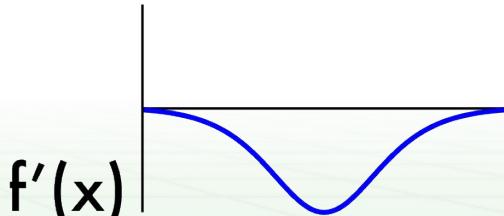
$f(x)$



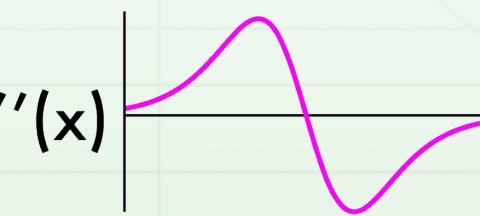
$f(x)$



$f'(x)$



$f'(x)$



$f''(x)$



$f''(x)$

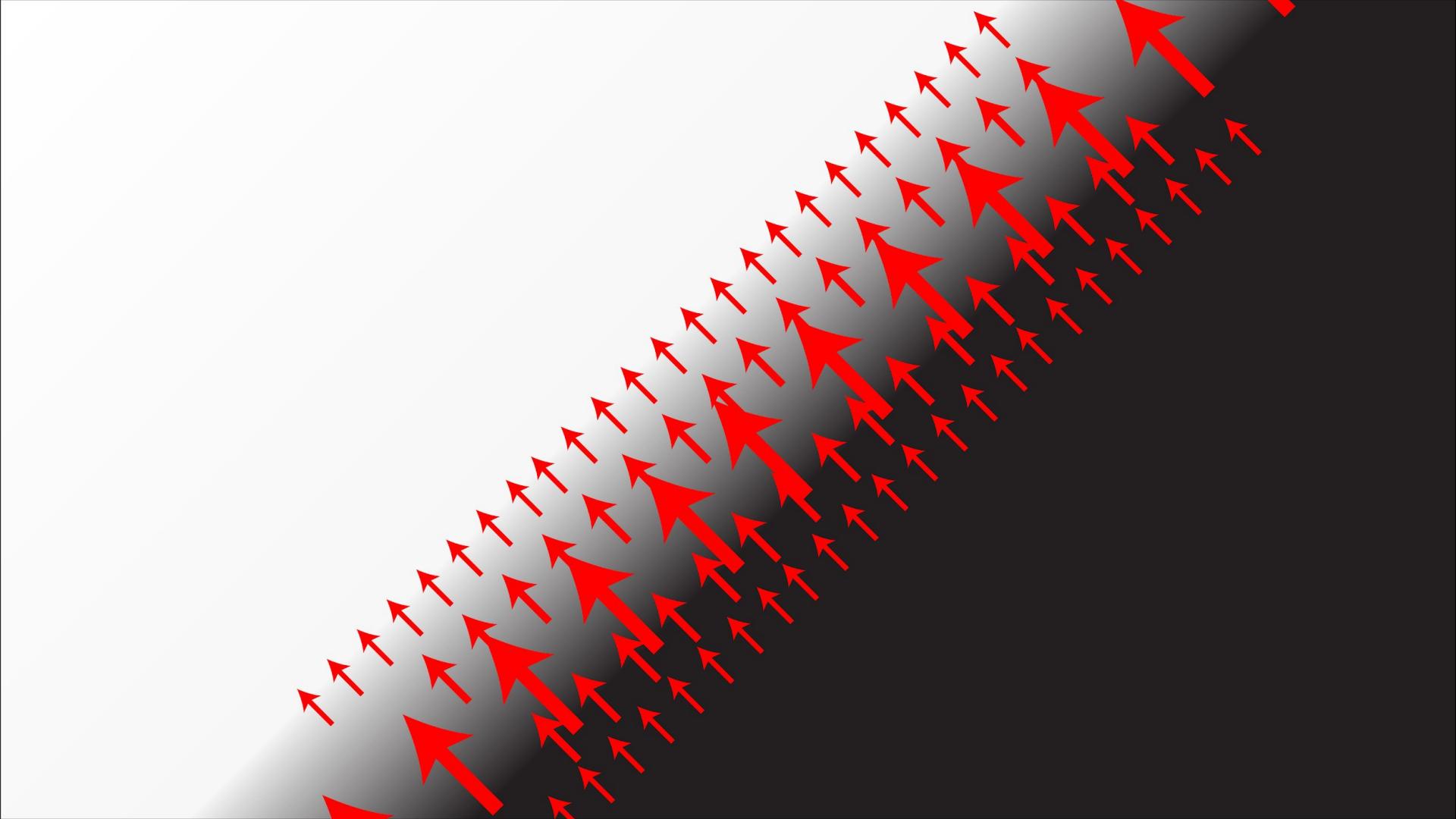
Laplacians

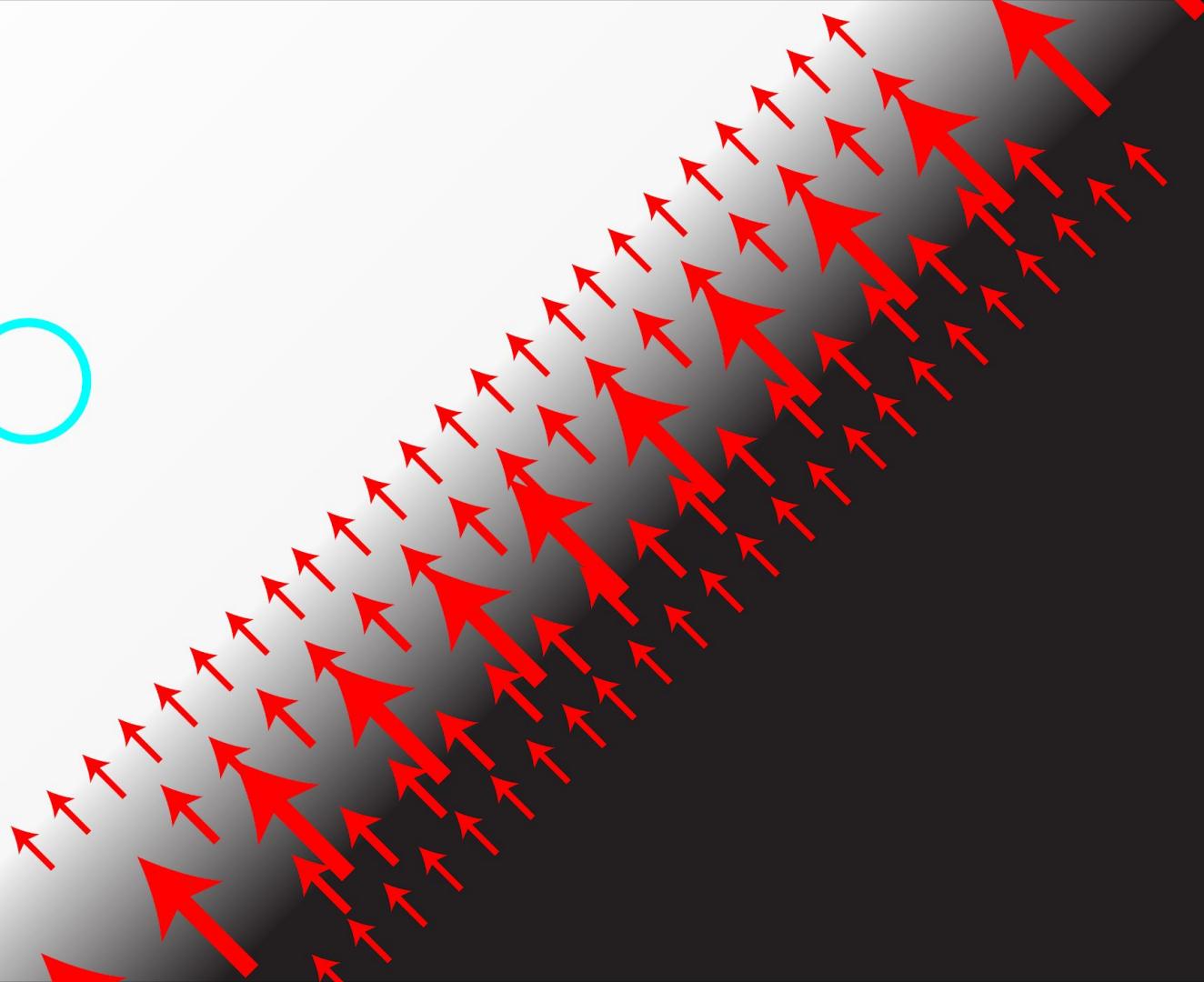
- Laplacian:

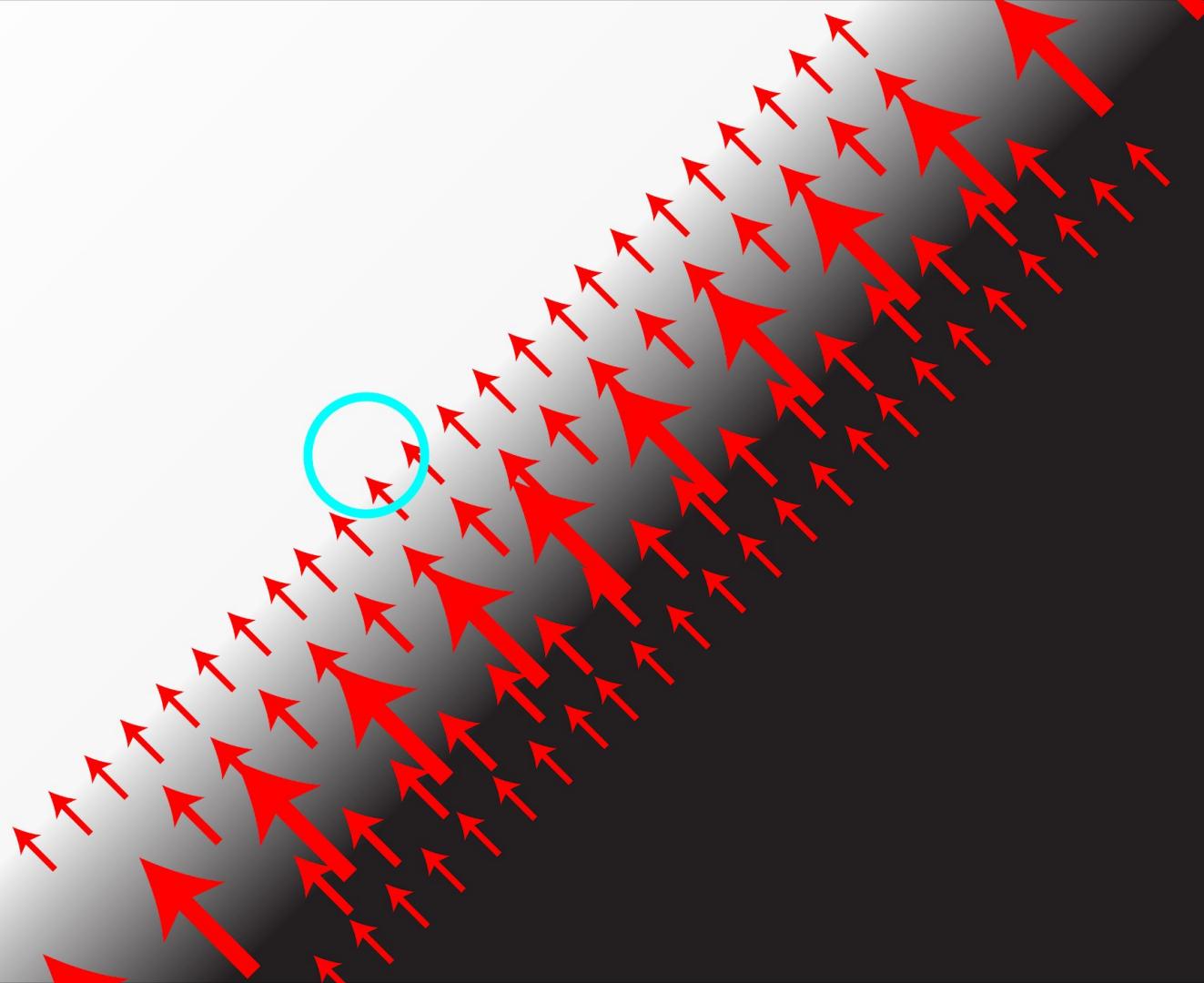
- $$- \Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

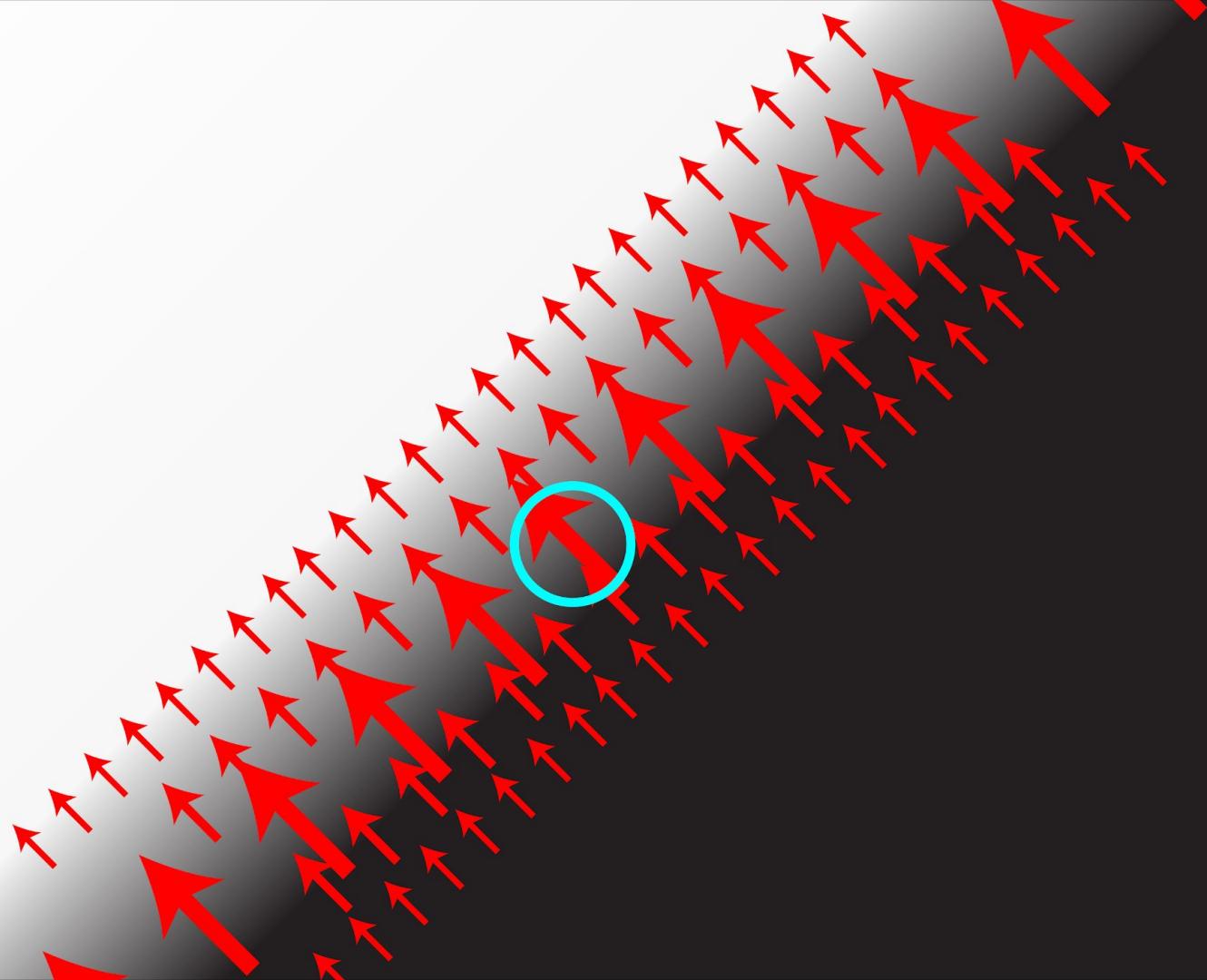
Laplacians

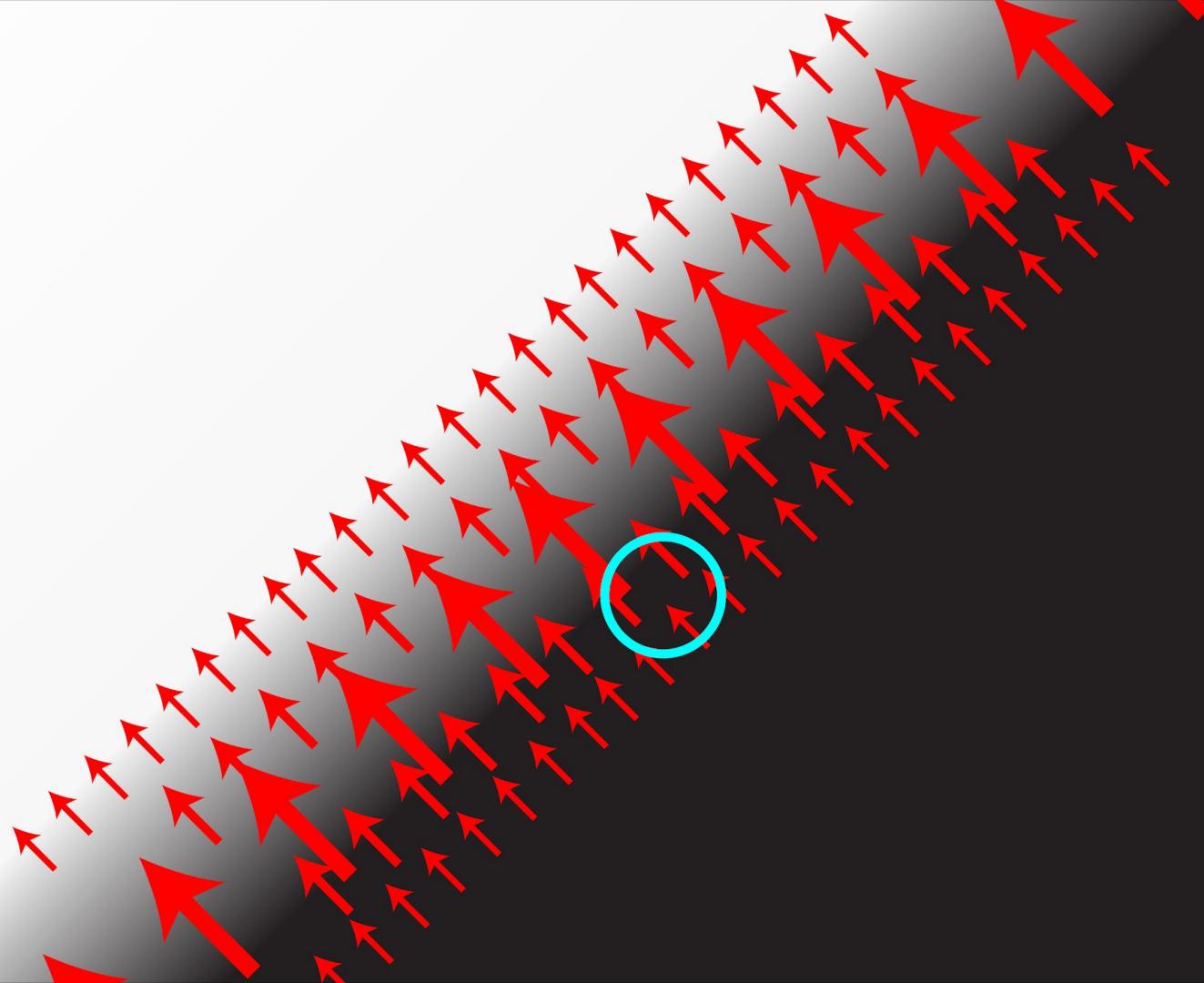
- Laplacian:
 - $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$
- Measures the divergence of the gradient
 - Flux of gradient vector field through small area







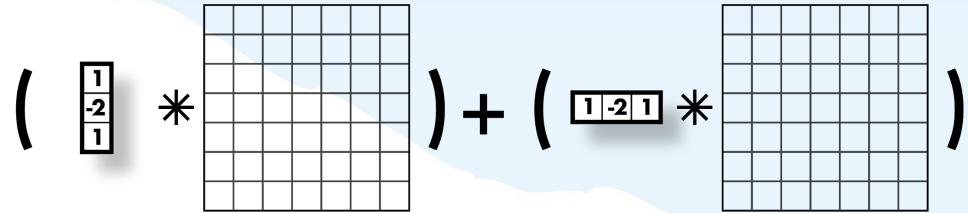




Laplacians

- Laplacian:

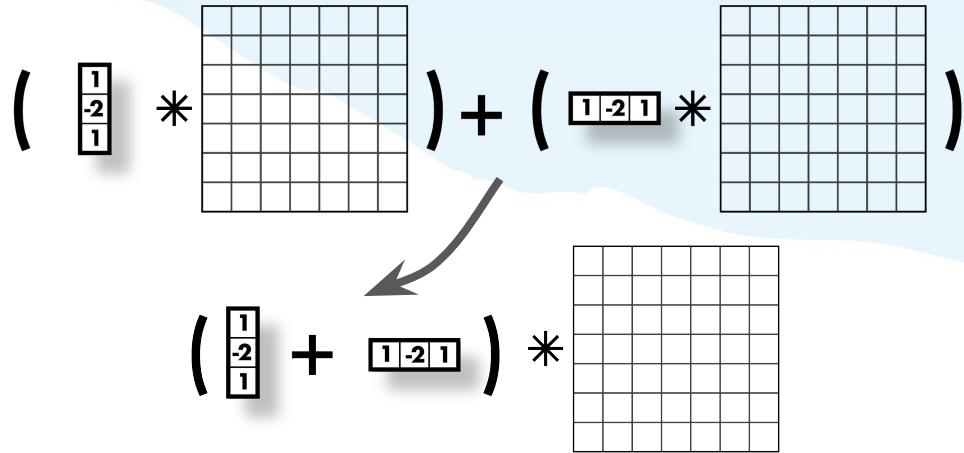
- $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$



Laplacians

- Laplacian:

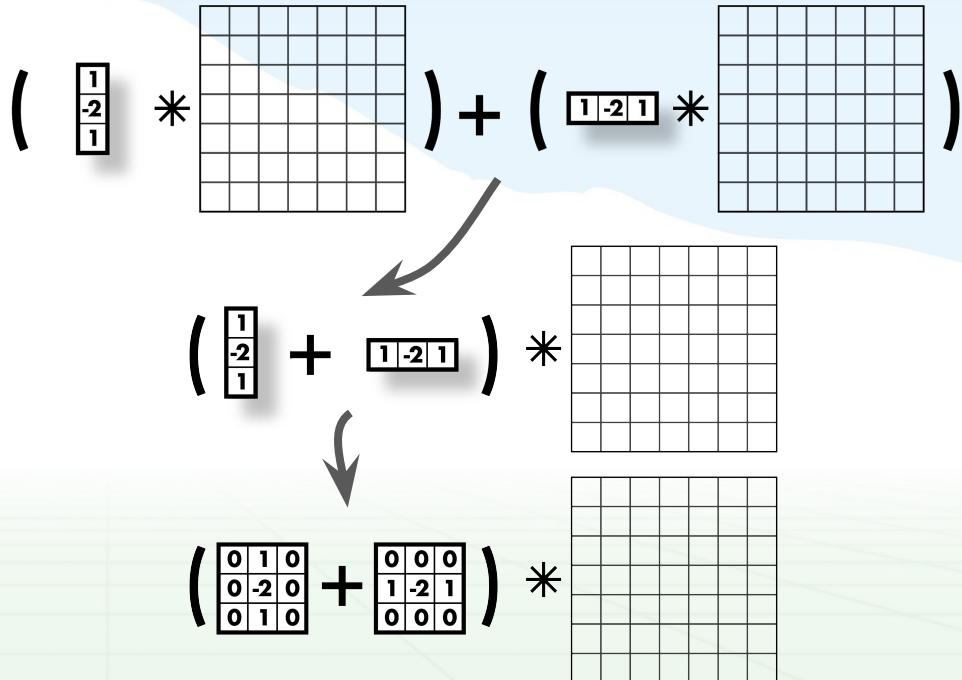
- $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$



Laplacians

- Laplacian:

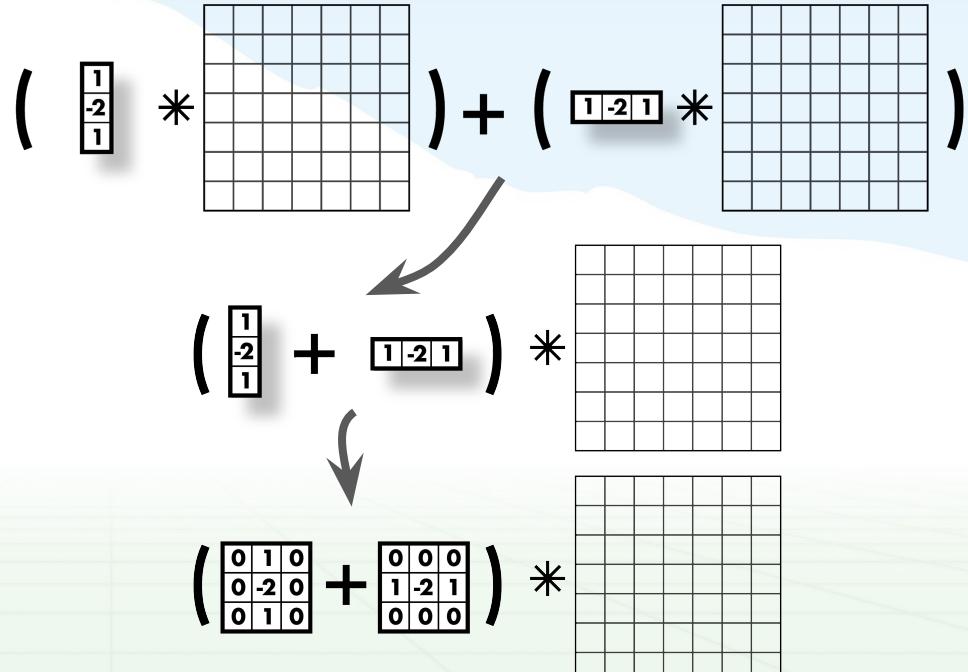
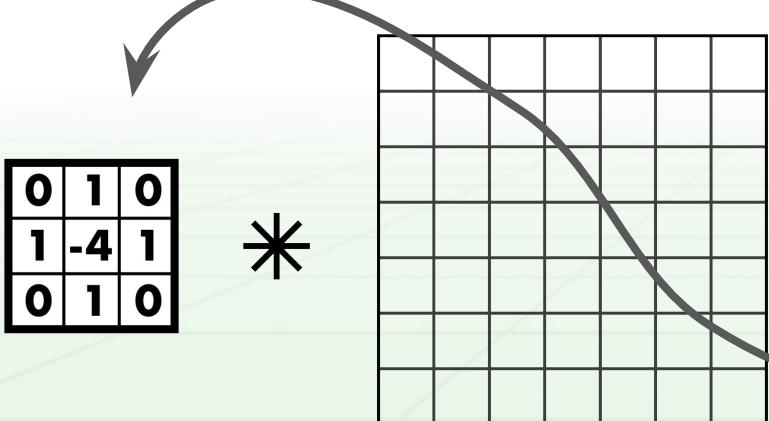
- $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$



Laplacians

- Laplacian:

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

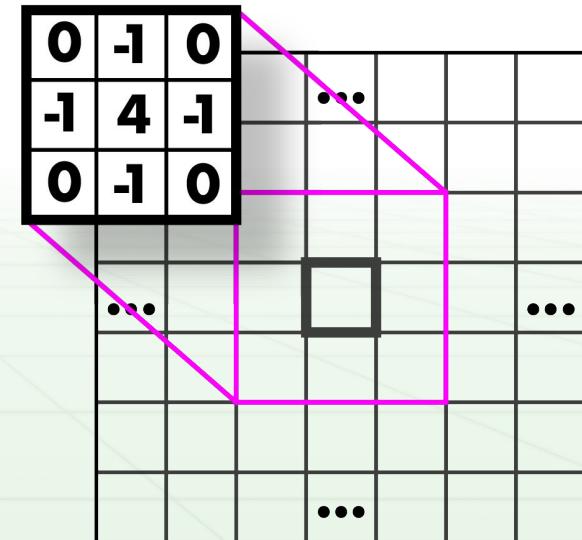
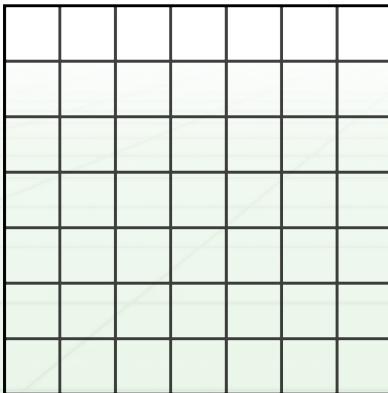


Laplacians

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

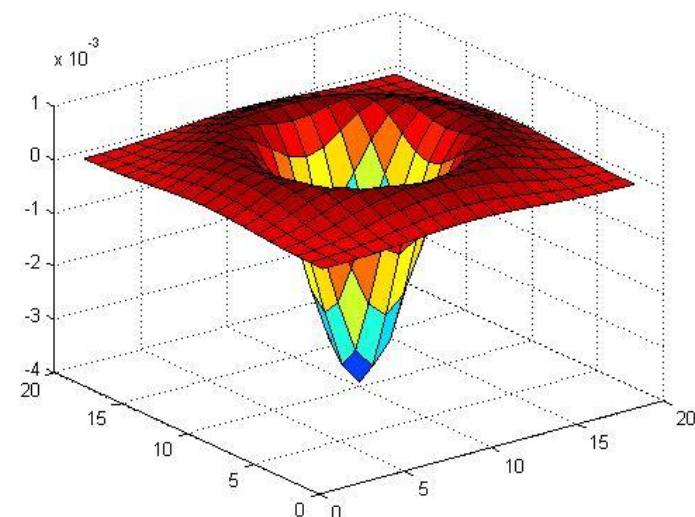
0	1	0
1	-4	1
0	1	0

*



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:

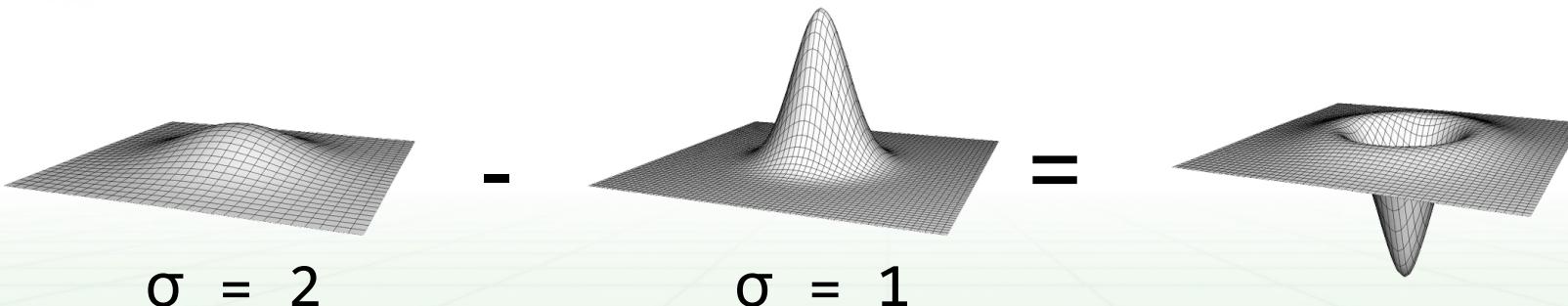
- Image is a function:
 - Has high frequency and low frequency components
 - Think in terms of fourier transform
- Edges are high frequency changes
- Maybe we want to find edges of a specific size
(i.e. specific frequency)

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

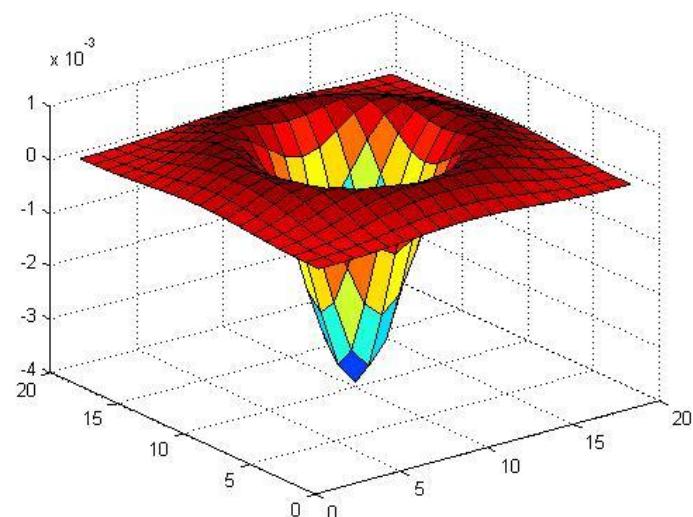
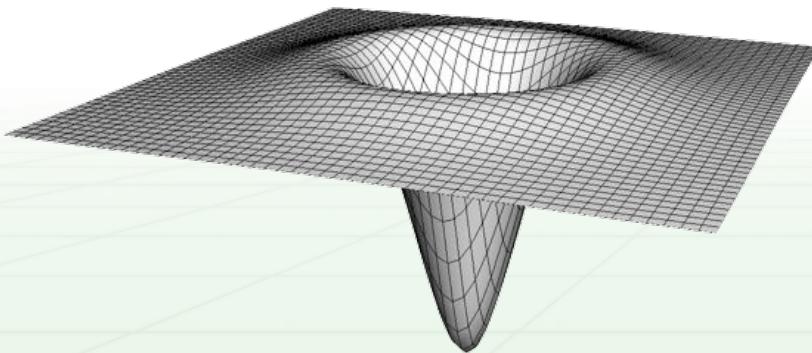
Difference of Gaussian (DoG)

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



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)

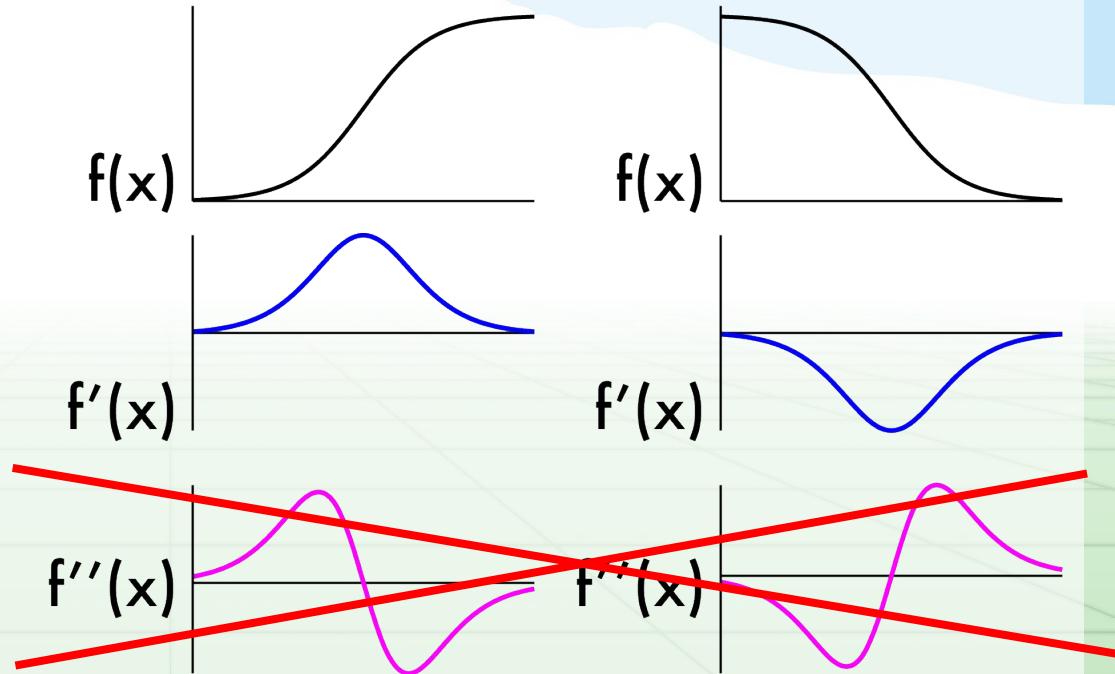


DoG (4 - 3)



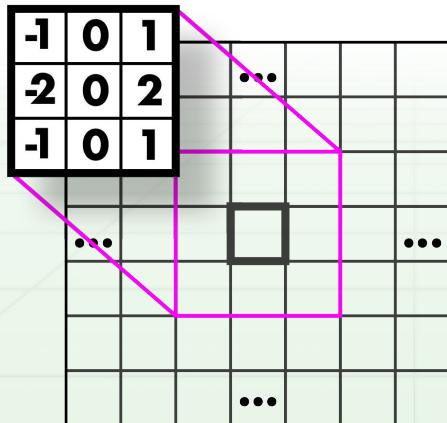
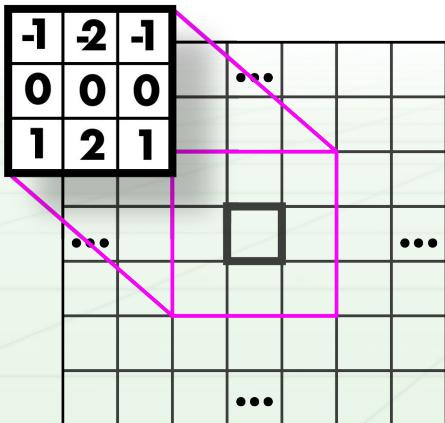
Another approach: gradient magnitude

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



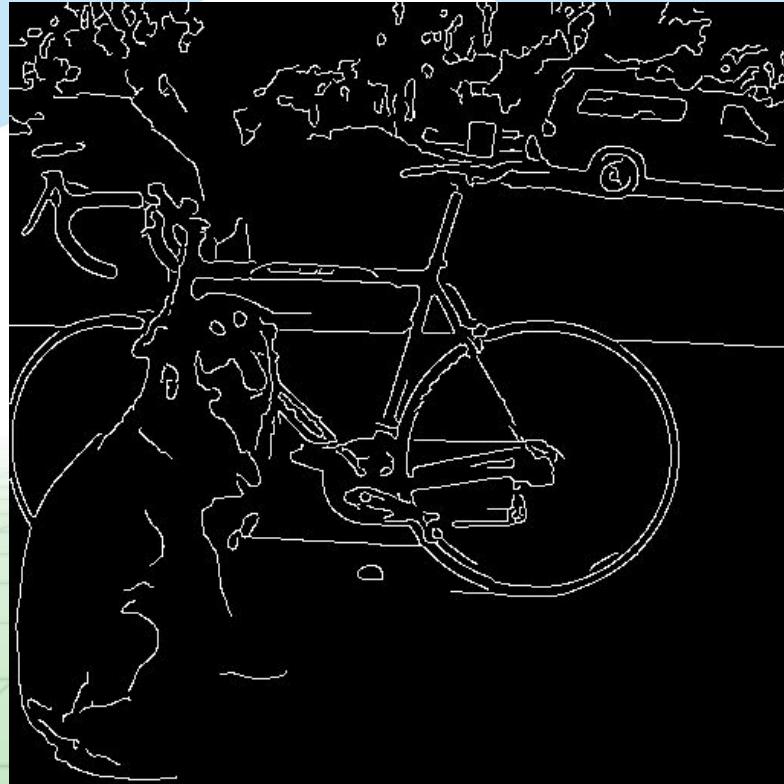
Another approach: gradient magnitude

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





What we really want: line drawing



Canny Edge Detection

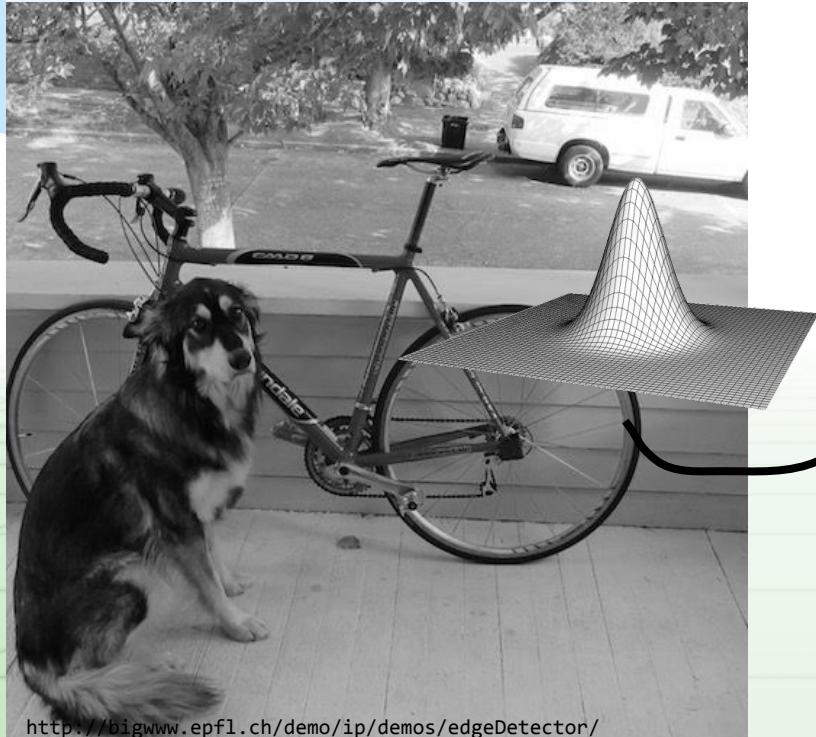
- Your first image processing pipeline!
 - Old-school CV is all about pipelines

Algorithm:

- 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

Smooth image

- You know how to do this, gaussians!

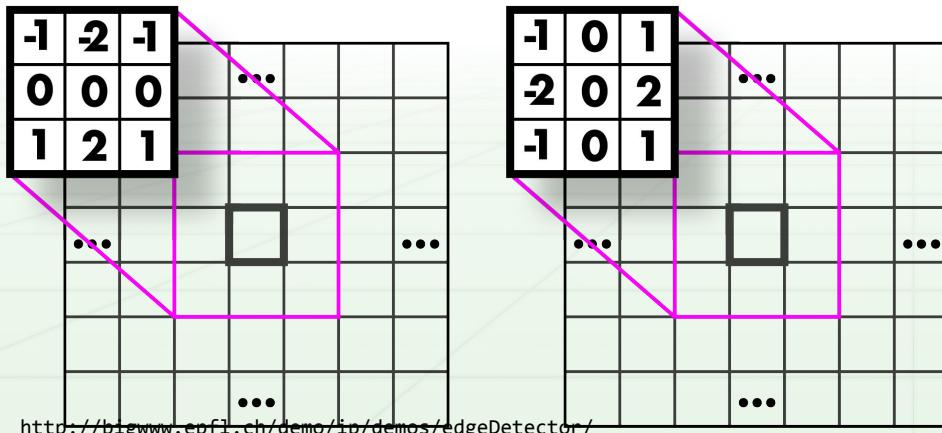


<http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/>



Gradient magnitude and direction

- Sobel filter

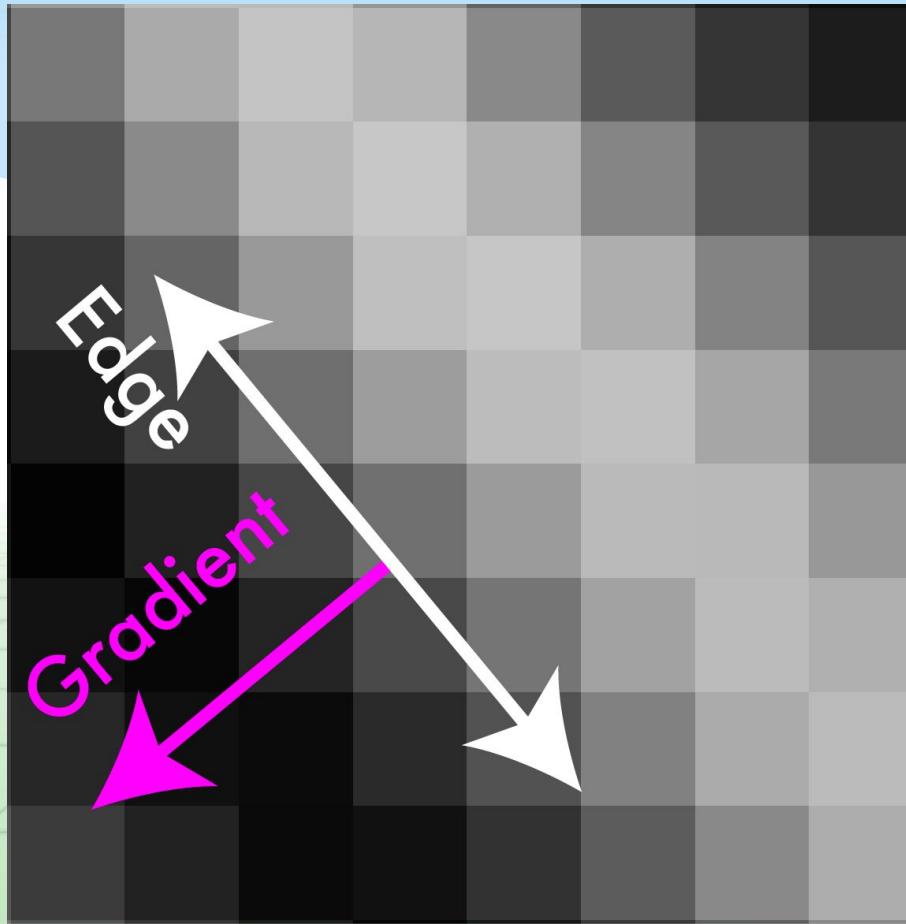


Non-maximum suppression

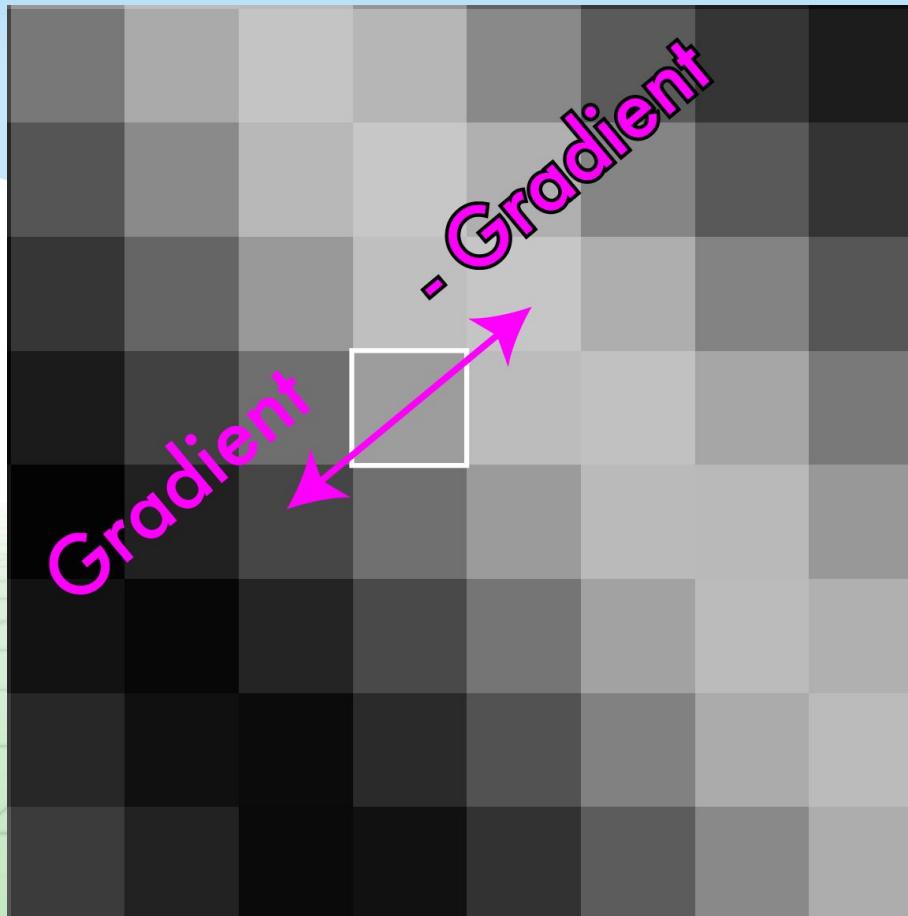
- Want single pixel edges, not thick blurry lines
- Need to check nearby pixels
- See if response is highest



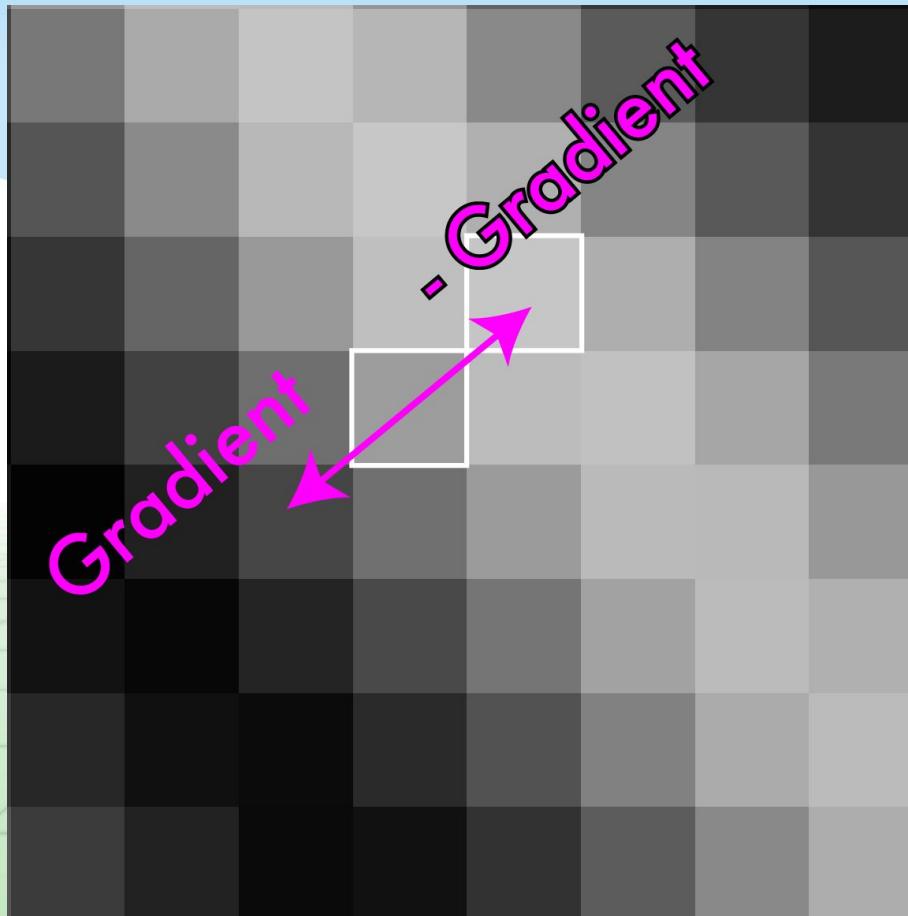
Non-maximum suppression



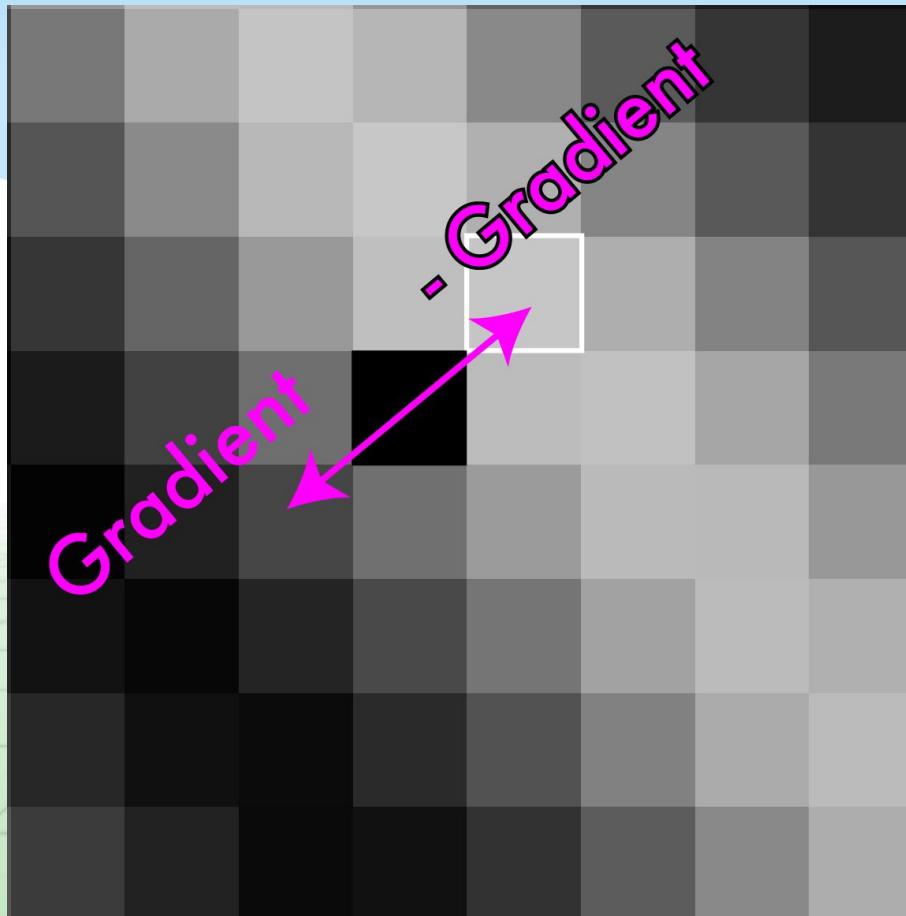
Non-maximum suppression



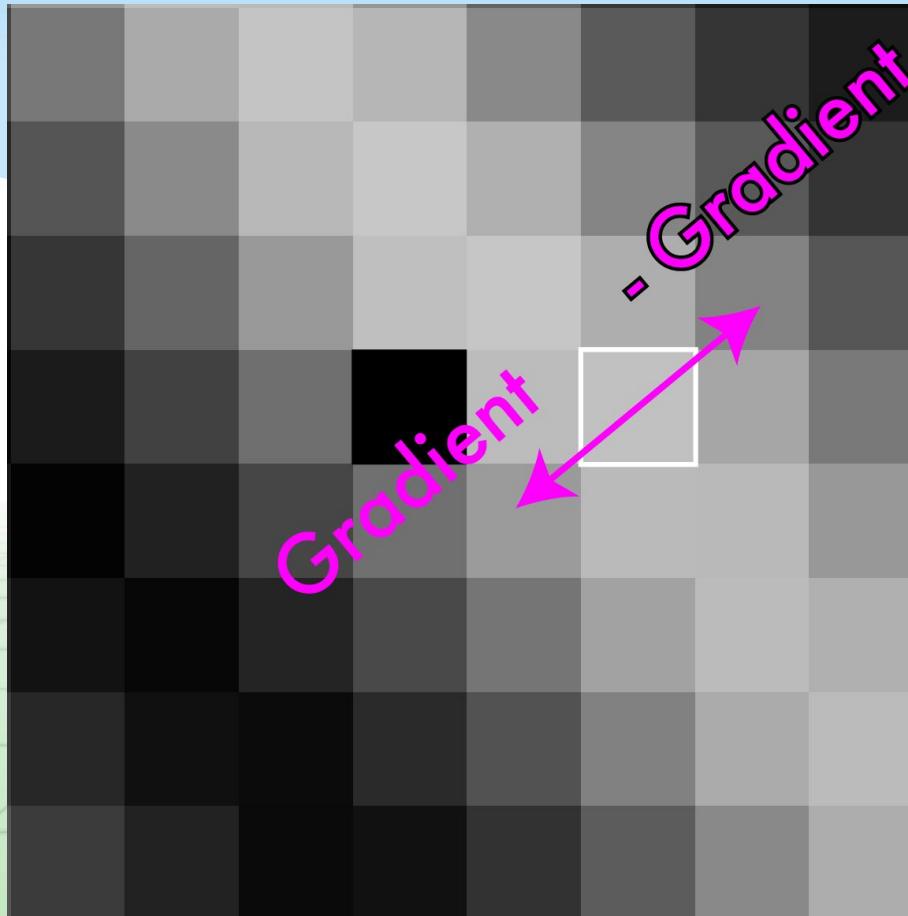
Non-maximum suppression



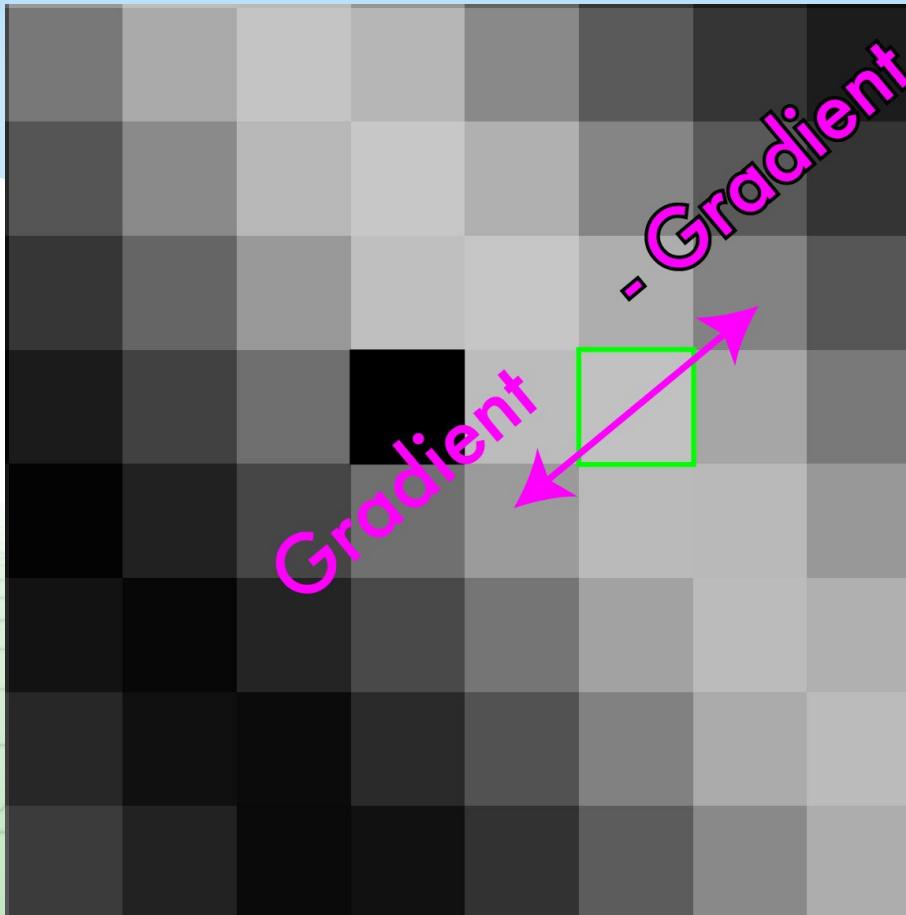
Non-maximum suppression



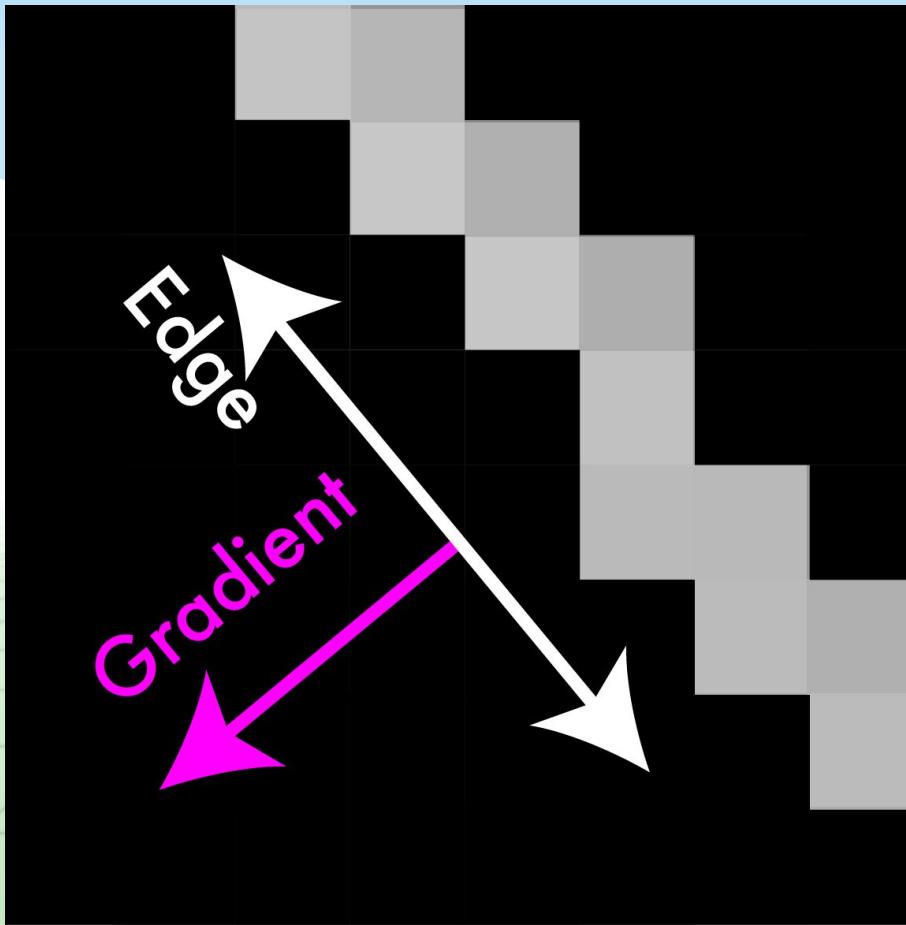
Non-maximum suppression



Non-maximum suppression



Non-maximum suppression

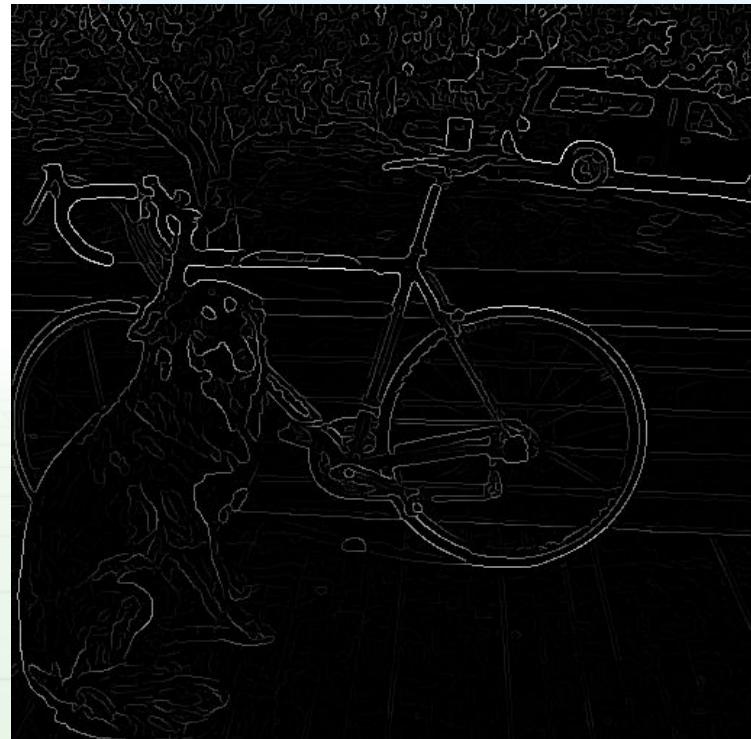


Non-maximum suppression



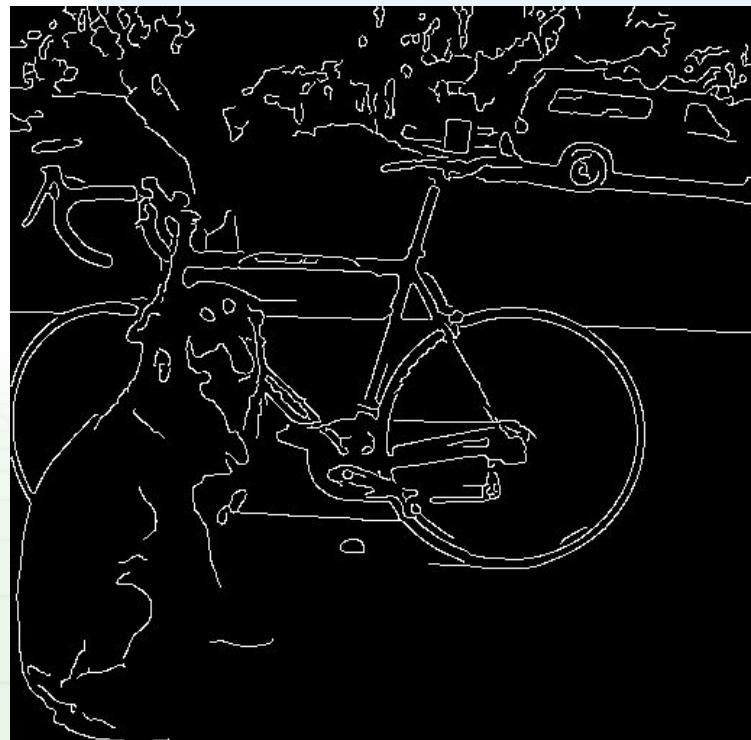
Threshold edges

- Still some noise
- Only want strong edges
- 2 thresholds, 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)



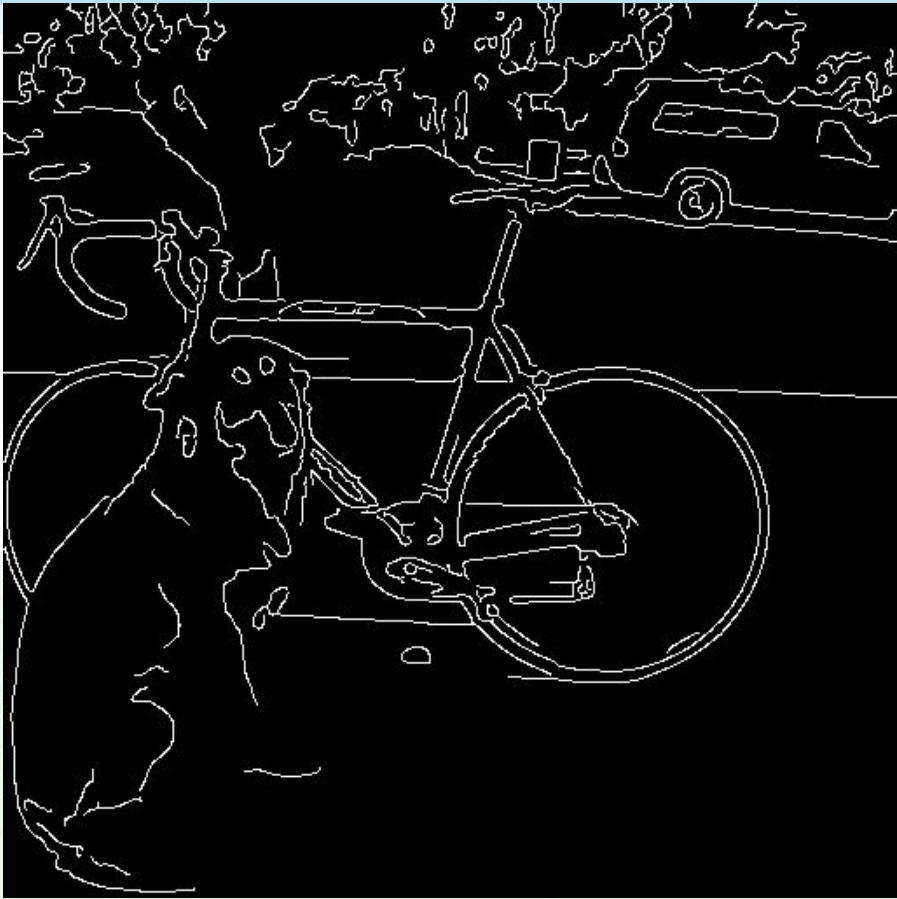
Canny Edge Detection

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

Algorithm:

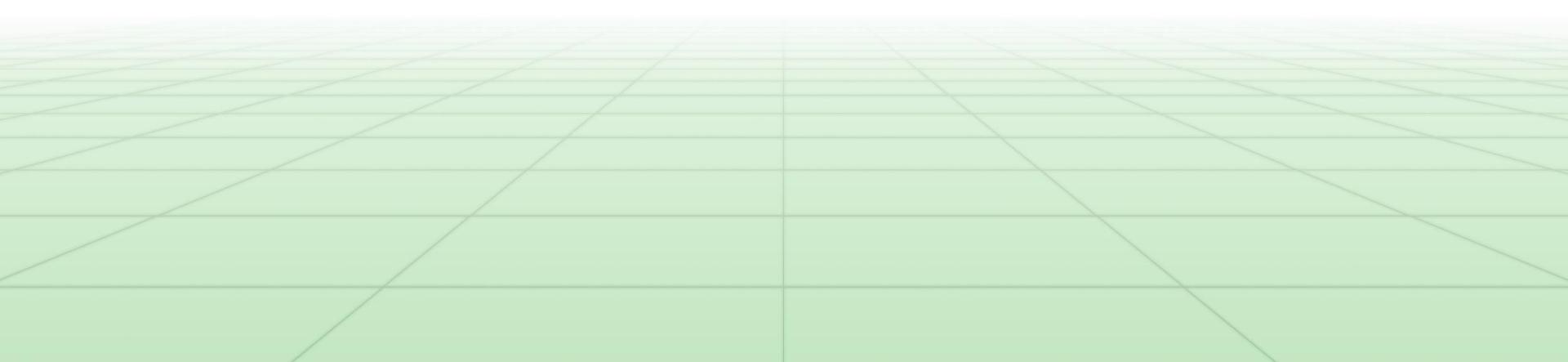
- Smooth image (only want “real” edges, not noise)
- Calculate gradient direction and magnitude
- Non-maximum suppression perpendicular to edge
- Threshold into strong, weak, no edge
- Connect together components
- Tunable: Sigma, thresholds

Canny Edge Detection





What else is there?

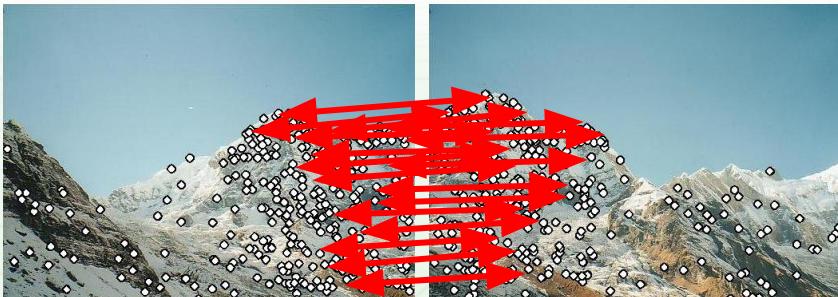
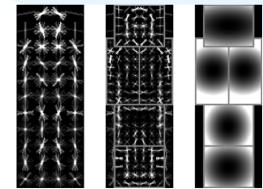
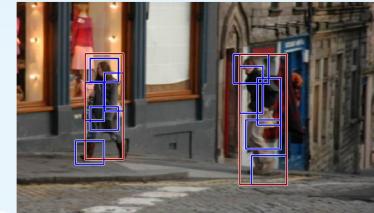
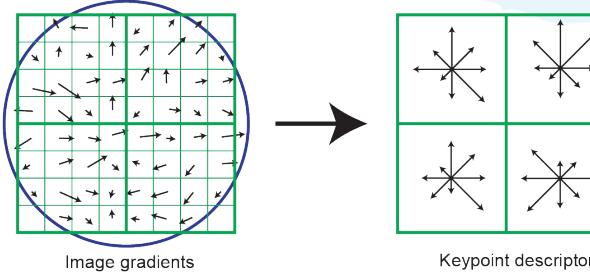


So what can we do with these convolutions anyway?

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

Features!

- Highly descriptive local regions
- Ways to describe those regions
- Useful for:
 - Matching
 - Recognition
 - Detection



What makes a good feature?

- Want to find patches in image that are useful or have some meaning
- For objects, want a patch that is common to that object but not in general
- For panorama stitching, want patches that we can find easily in another image of same place
- Good features are unique!
 - Can find the “same” feature easily
 - Not mistaken for “different” features

How close are two patches?

- Sum squared difference
- Images I, J
- $\sum_{x,y} (I(x,y) - J(x,y))^2$

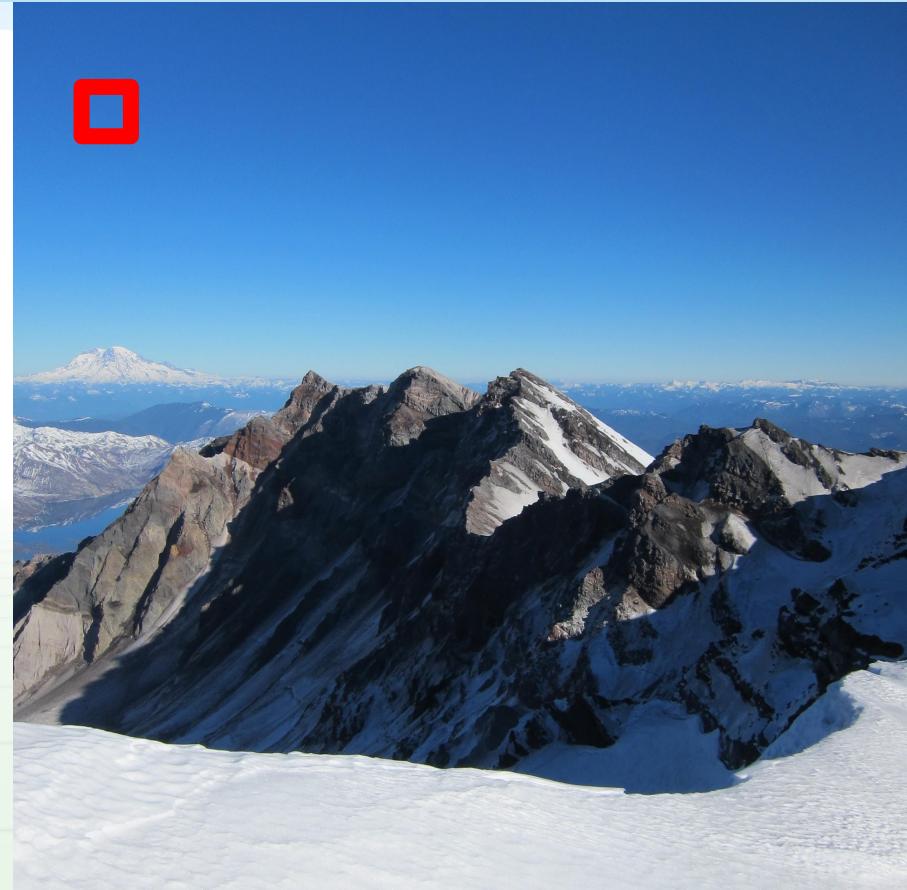
How can we find unique patches?

- Say we are stitching a panorama
- Want patches in image to match to other image
- Need to only match one spot



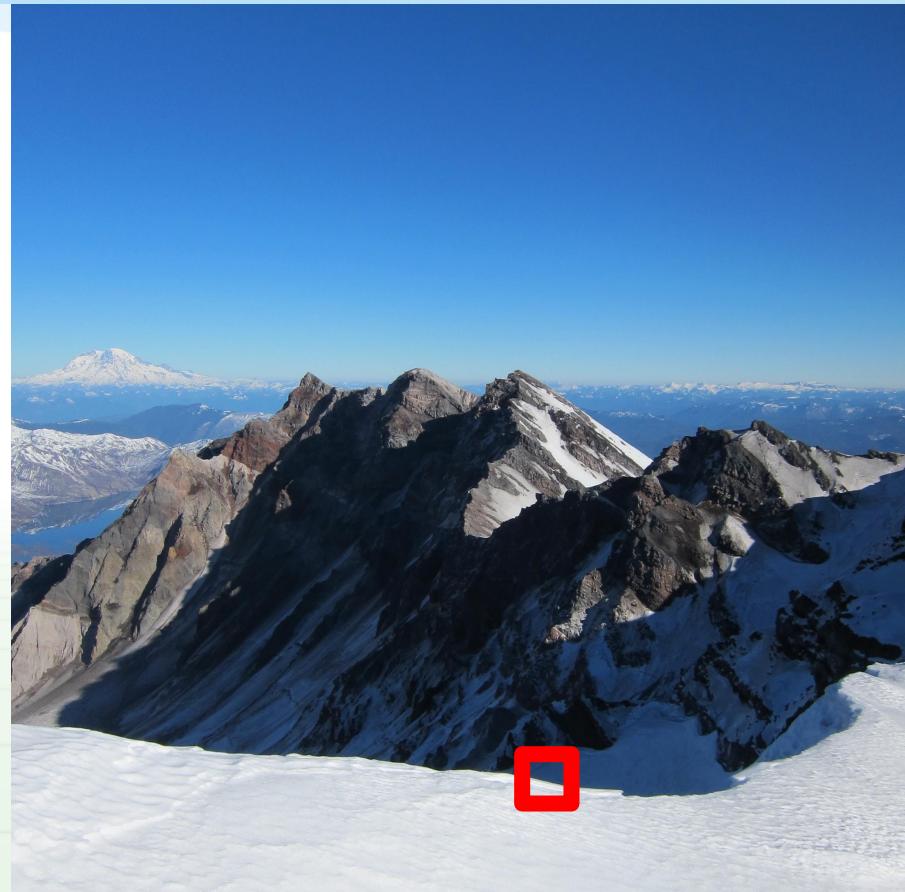
How can we find unique patches?

- Sky: bad
 - Very little variation
 - Could match any other sky



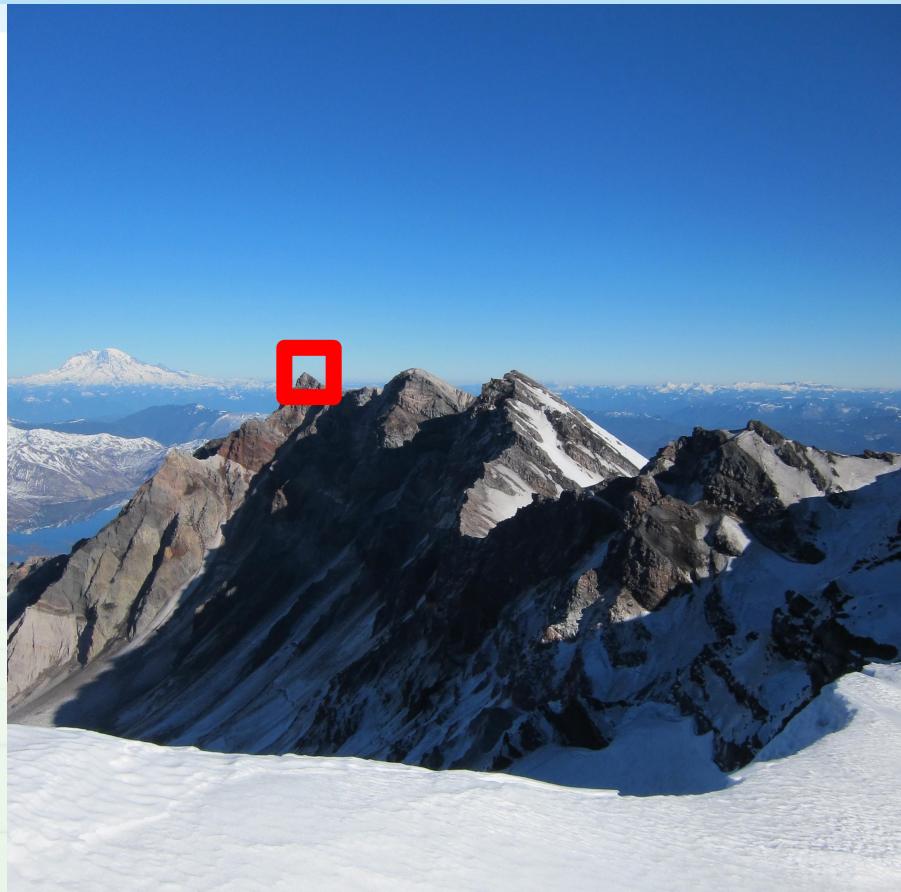
How can we find unique patches?

- Sky: bad
 - Very little variation
 - Could match any other sky
- Edge: ok
 - Variation in one direction
 - Could match other patches along same edge



How can we find unique patches?

- Sky: bad
 - Very little variation
 - Could match any other sky
- Edge: ok
 - Variation in one direction
 - Could match other patches along same edge
- Corners: good!
 - Only one alignment matches



How can we find unique patches?

- Want a patch that is unique in the image
- Can calculate distance between patch and every other patch, lot of computation

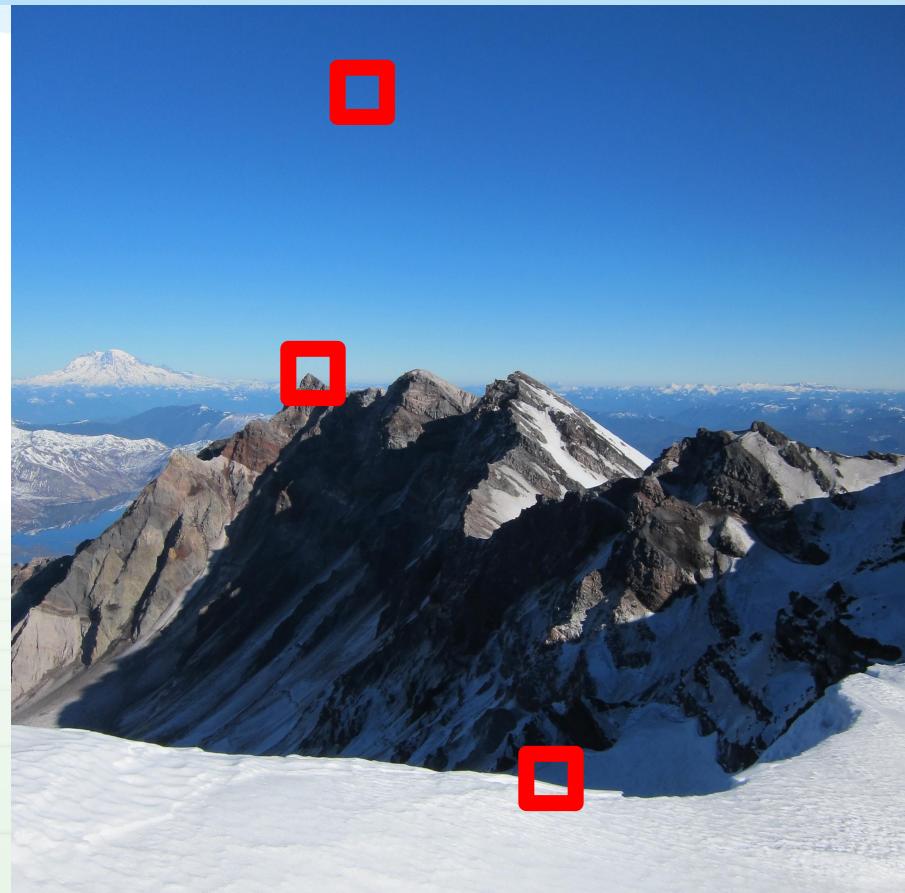
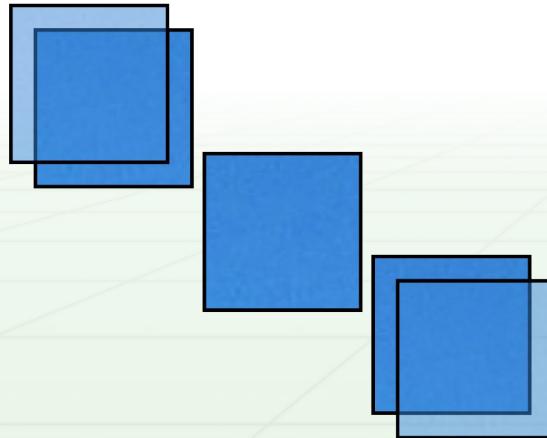
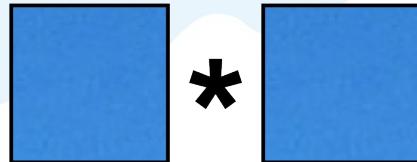


How can we find unique patches?

- Want a patch that is unique in the image
- Can calculate distance between patch and every other patch, lot of computation
- Instead, we could think about auto-correlation:
 - How well does image match shifted version of itself?
- $\sum_d \sum_{x,y} (I(x,y) - I(x+d_x, y+d_y))^2$
- Measure of self-difference (how am I not myself?)

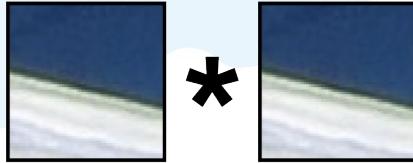
Self-difference

Sky: low everywhere

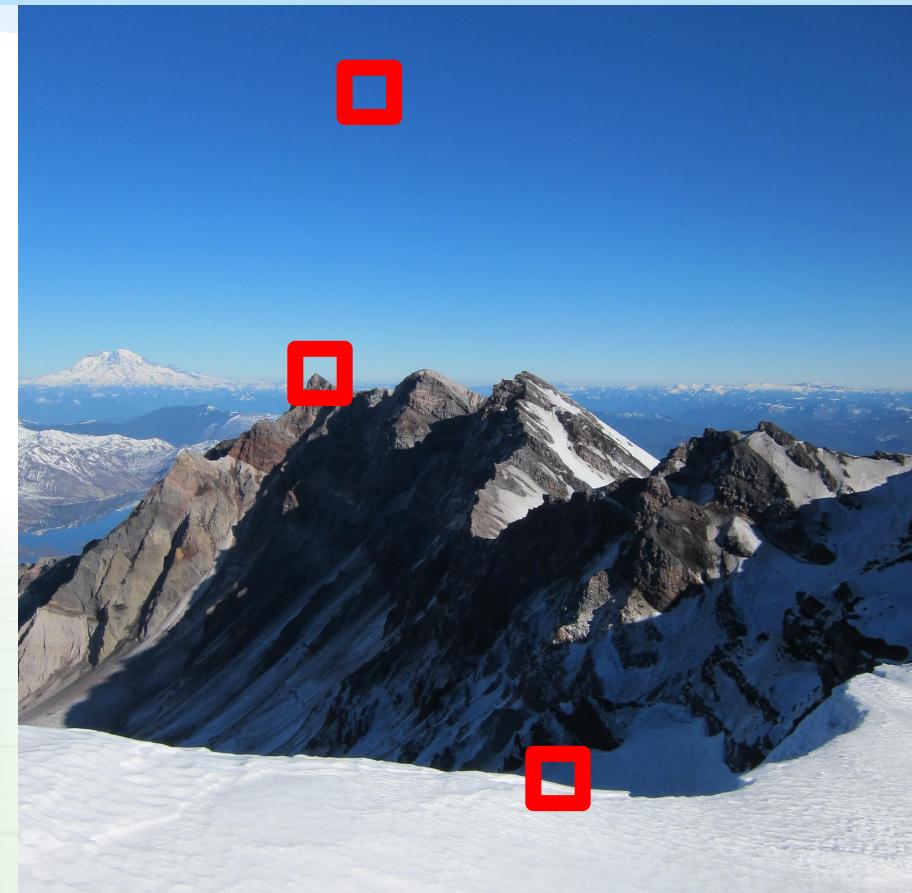
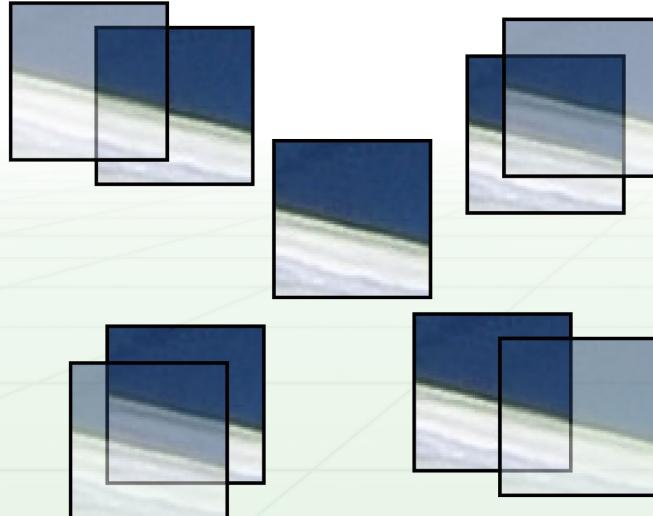


Self-difference

Edge: low along edge



*

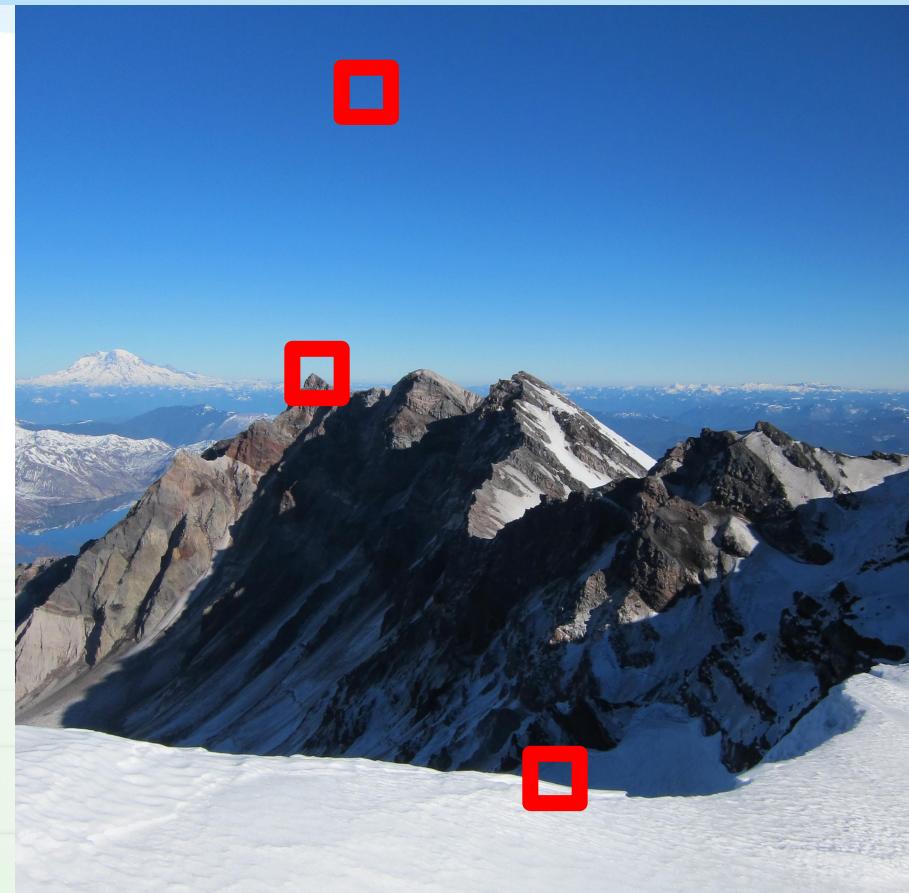
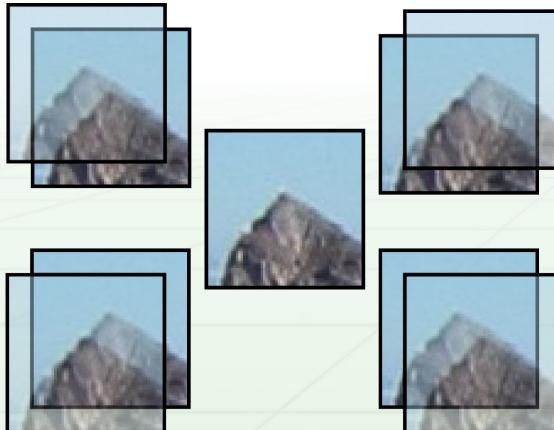


Self-difference

Corner: mostly high



*

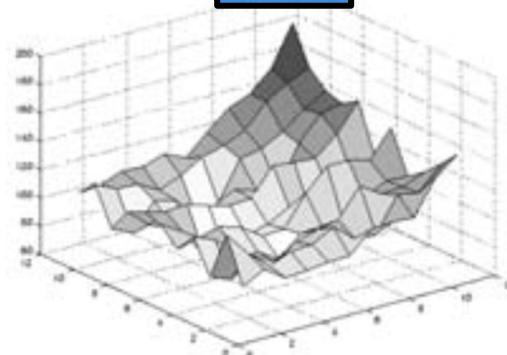
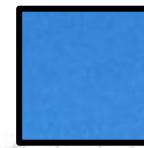
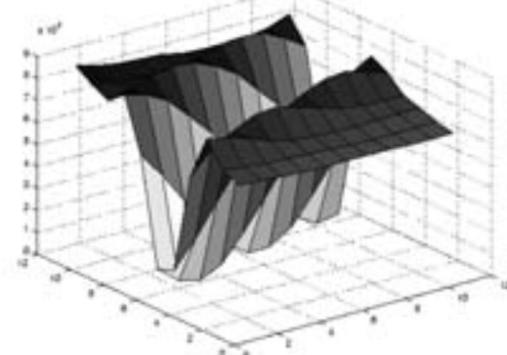
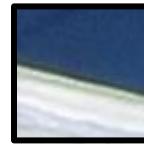
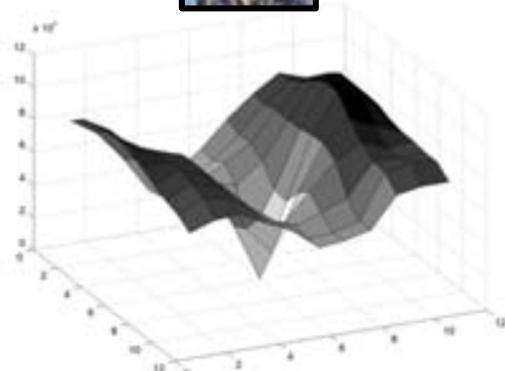


Self-difference

Sky: low everywhere

Edge: low along edge

Corner: mostly high



Self-difference is still expensive

- $\sum_d \sum_{x,y} (I(x,y) - I(x+d_x, y+d_y))^2$
- Lots of summing
- Need an approximation
 - If you want the mathy details, Szeliski pg 212

Approximate self-difference

- Look at nearby gradients I_x and I_y
- If gradients are mostly zero, not a lot going on
 - Low self-difference
- If gradients are mostly in one direction, edge
 - Still low self-difference
- If gradients are in twoish directions, corner!
 - High self-difference, good patch!

Approximate self-difference

- How do we tell what's going on with gradients?
- Eigen vectors/values!
- Need structure matrix for patch, just a weighted sum of nearby gradient information

$$S_w[p] = \begin{bmatrix} \sum_r w[r](I_x[p-r])^2 & \sum_r w[r]I_x[p-r]I_y[p-r] \\ \sum_r w[r]I_x[p-r]I_y[p-r] & \sum_r w[r](I_y[p-r])^2 \end{bmatrix}$$

- Not as complex as it looks, weighted sum of gradients near pixel

Structure matrix

- Weighted sum of gradient information
 - $\begin{vmatrix} \sum_i w_i I_x(i) I_x(i) & \sum_i w_i I_x(i) I_y(i) \\ \sum_i w_i I_y(i) I_x(i) & \sum_i w_i I_y(i) I_y(i) \end{vmatrix}$
- Can use Gaussian weighting (so many gaussians)
- Eigen vectors/values of this matrix summarize the distribution of the gradients nearby
- λ_1 and λ_2 are eigenvalues
 - λ_1 and λ_2 both small: no gradient
 - $\lambda_1 \gg \lambda_2$: gradient in one direction
 - λ_1 and λ_2 similar: multiple gradient directions, corner

Estimating smallest eigen value

- A few methods:
 - $\det(S) = \lambda_1 * \lambda_2$
 - $\text{trace}(S) = \lambda_1 + \lambda_2$
- $\det(S) - \alpha \text{trace}(S)^2 = \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$
- $\det(S) / \text{trace}(S) = \lambda_1 \lambda_2 / (\lambda_1 + \lambda_2)$
- If these estimates are large, λ_2 is large

Harris Corner Detector

- Calculate derivatives I_x and I_y
- Calculate 3 measures $I_x I_x$, $I_y I_y$, $I_x I_y$
- Calculate weighted sums
 - Want a weighted sum of nearby pixels, guess what this is?
 - Gaussian!
- Estimate response based on smallest eigen value
- Non-max suppression (just like canny)



