
The Ancient
Secrets
Computer Vision

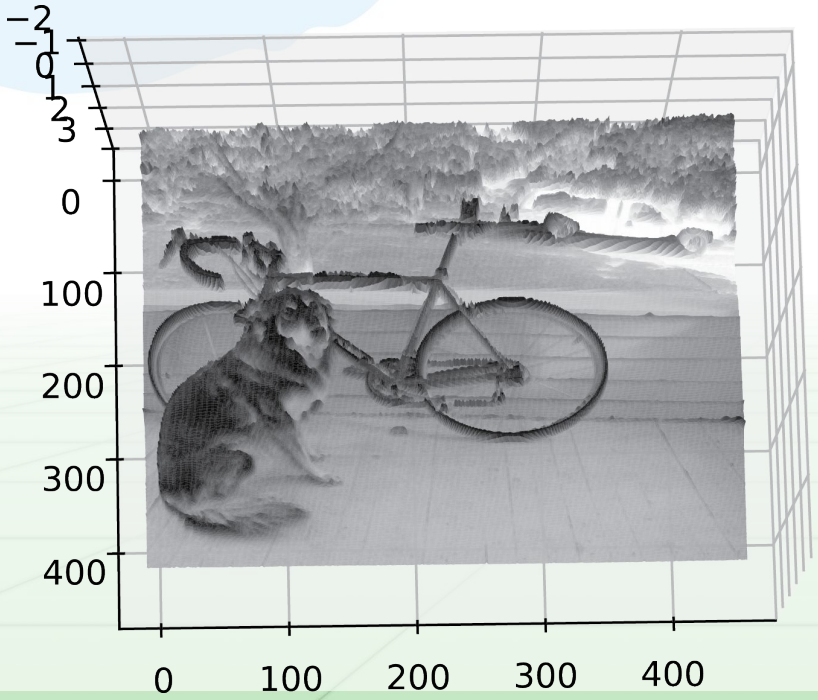
Logistics:

- Homework 1 due tonight at midnight!
  - If you are pretty sure your code is right don’t stress the tests. They are just to help us look at less code
  - You can still get full marks if you don’t pass all tests, iff the code is right!
- Two office hours today if you need help
- Homework 2 will be out tomorrow!
- Start thinking final projects
  - You’ll have the last 2 weeks to work on them
  - We’ll have a proposal phase to make sure everyone’s on the right track


Previously
On
the
Ancient Secrets
of Computer Vision

What’s an edge?

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



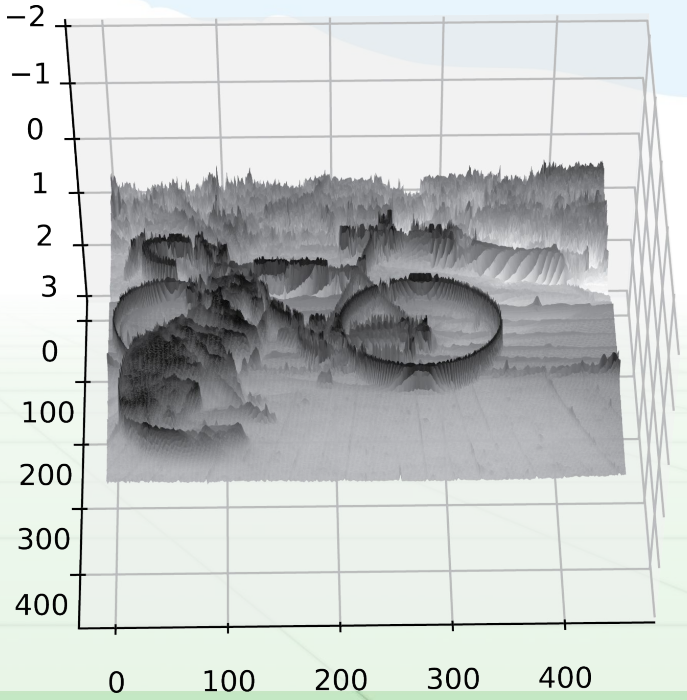
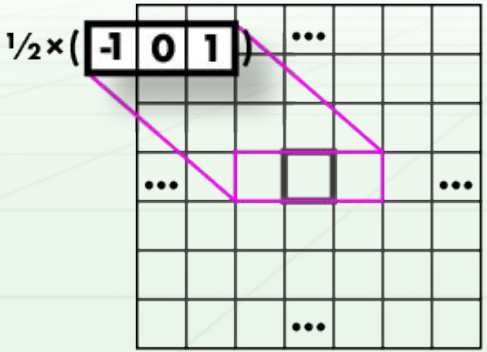


Image derivatives

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



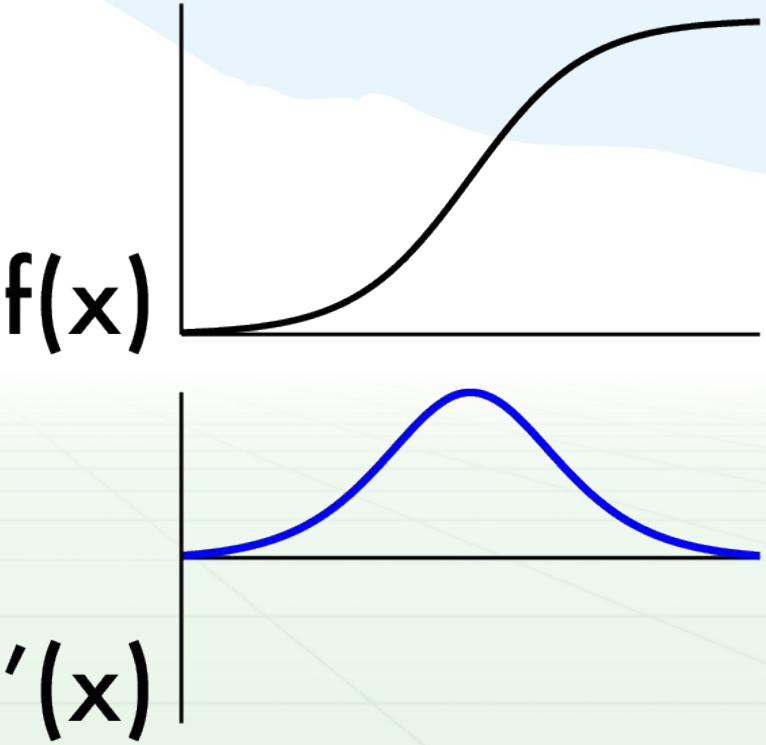
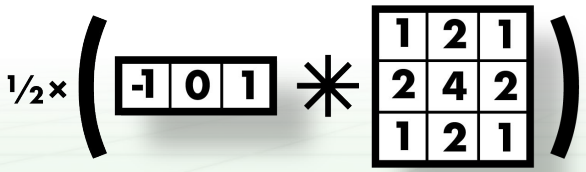
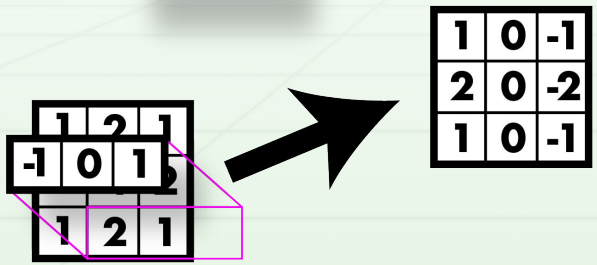
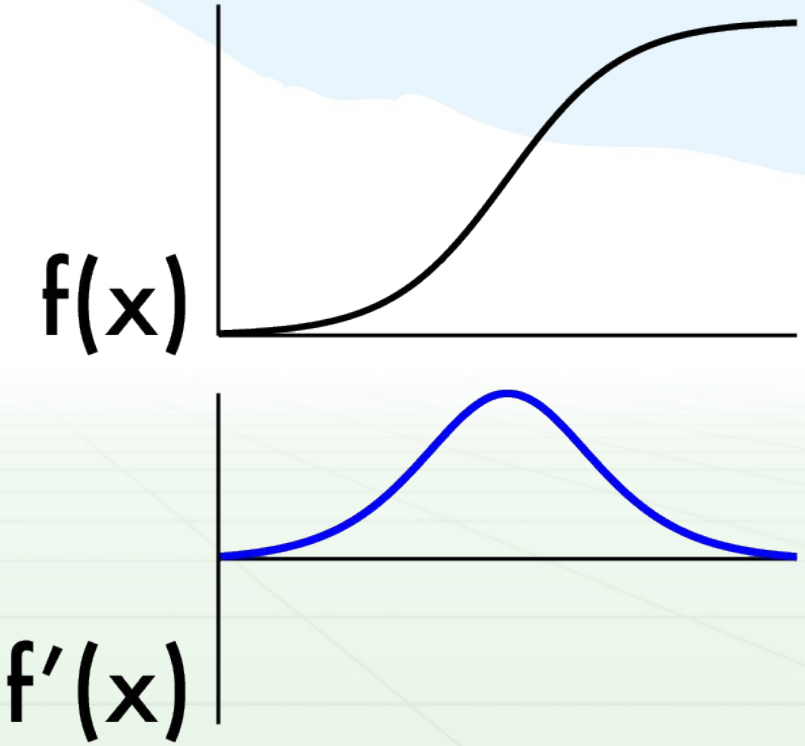


Image derivatives

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

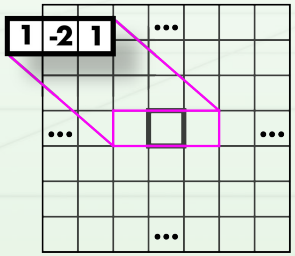


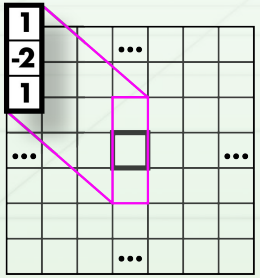


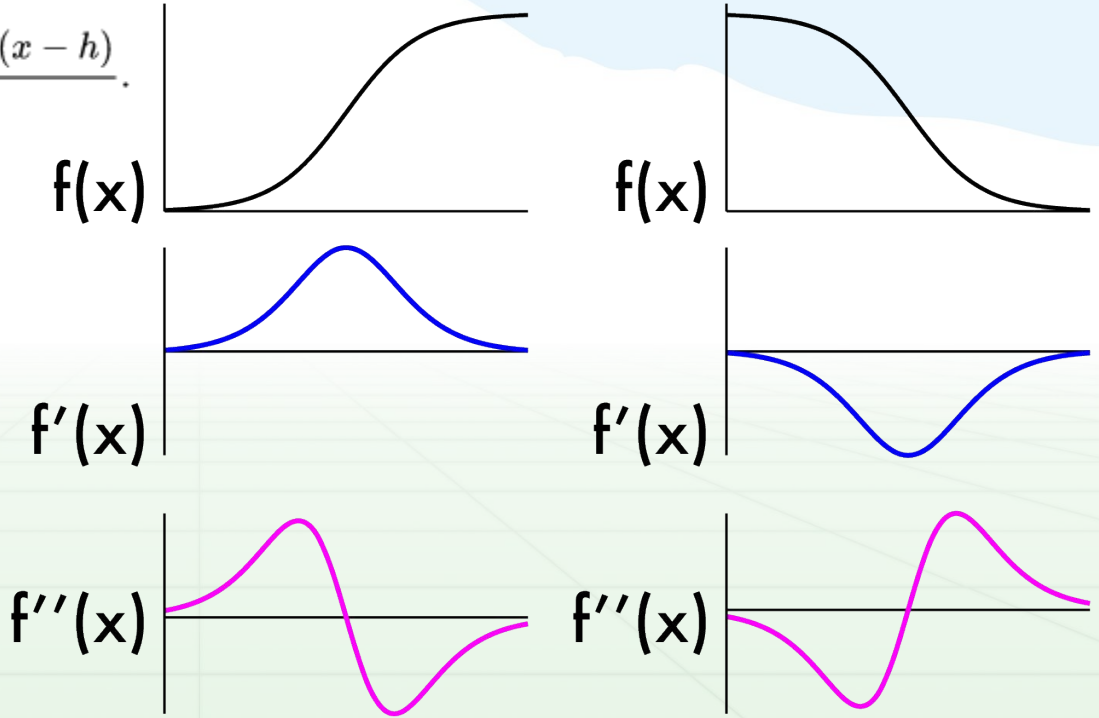


Laplacian (2nd derivative)!

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

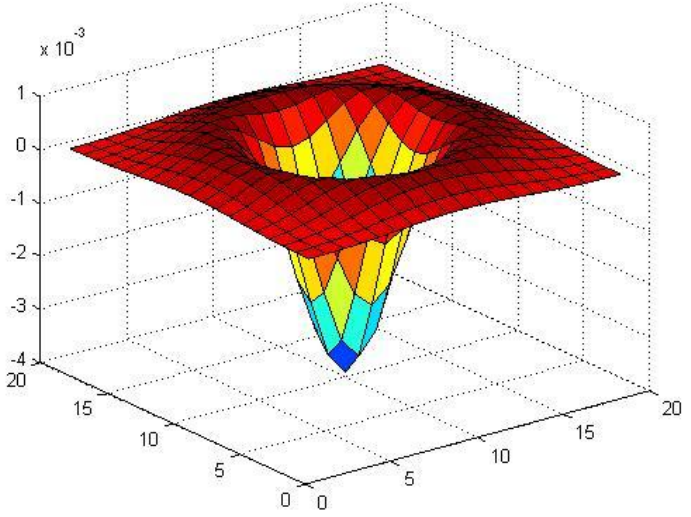






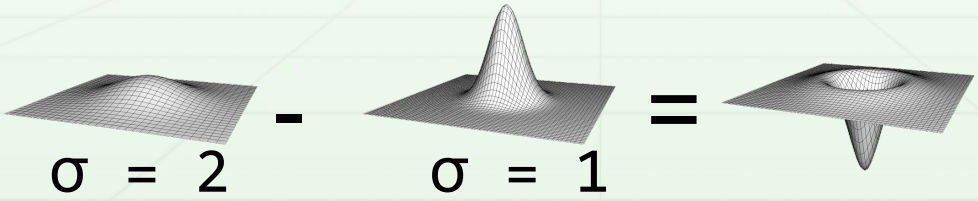
Laplacians also sensitive to noise

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

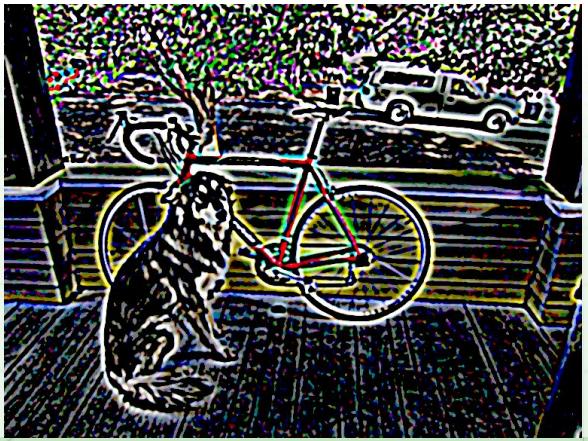


Difference of Gaussian (DoG)

- Gaussian is a low pass filter
- Strongly reduce components with frequency f < σ
- (g*I) low frequency components
- I - (g*I) high frequency components
- g(σ1)*I - g(σ2)*I
  - Components in between these frequencies
- g(σ1)*I - g(σ2)*I = [g(σ1) - g(σ2)]*I

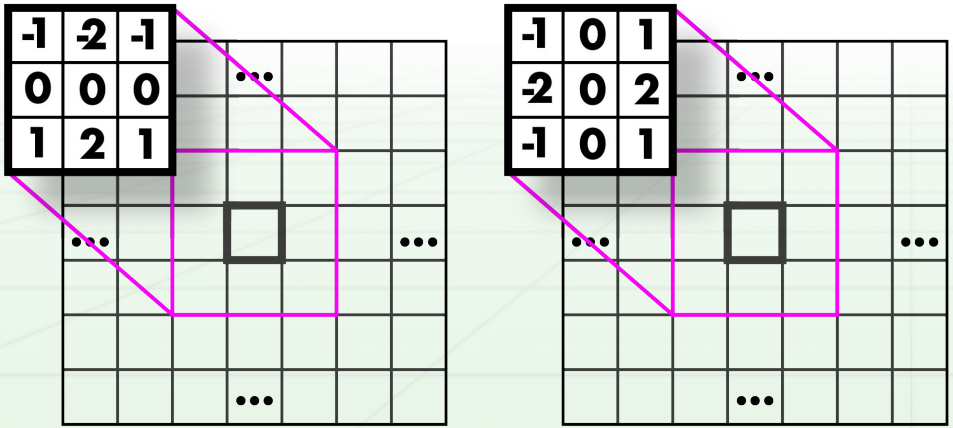


DoGs



Another approach: gradient magnitude

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



Canny Edge Detection

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

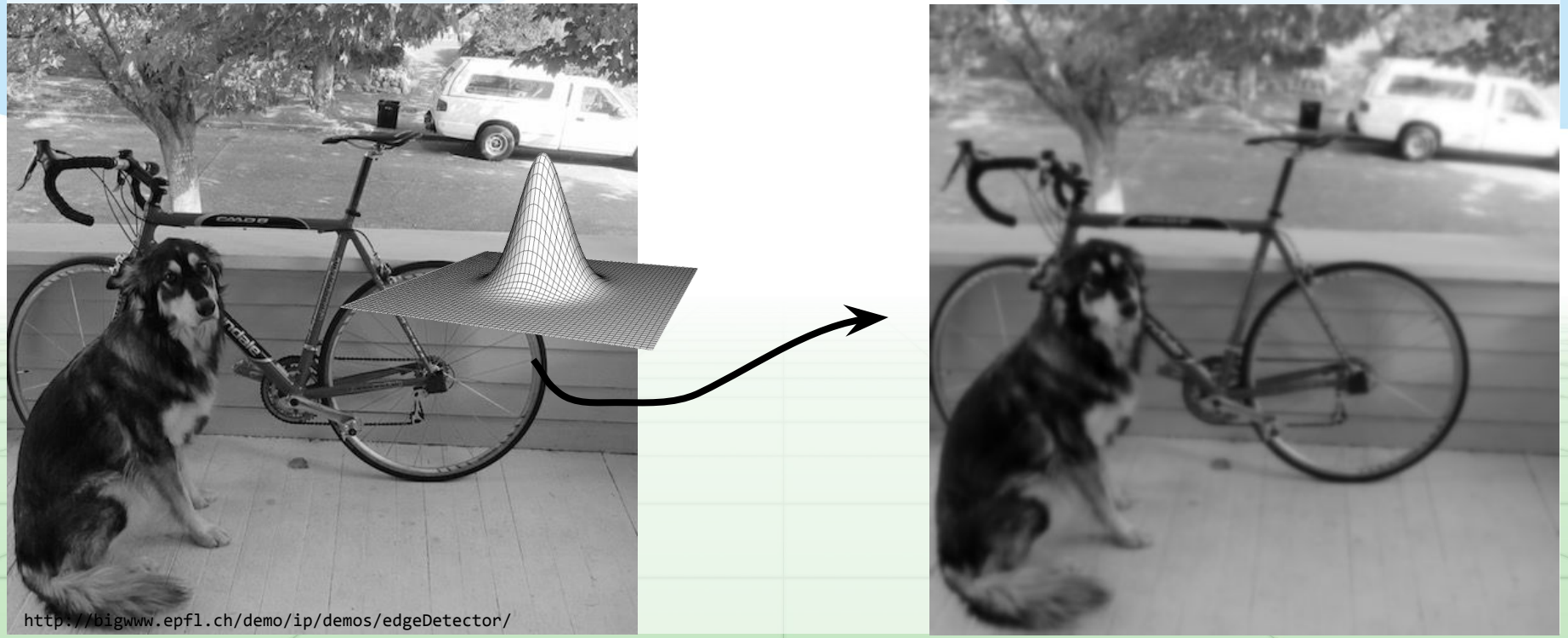
Algorithm:

- 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

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

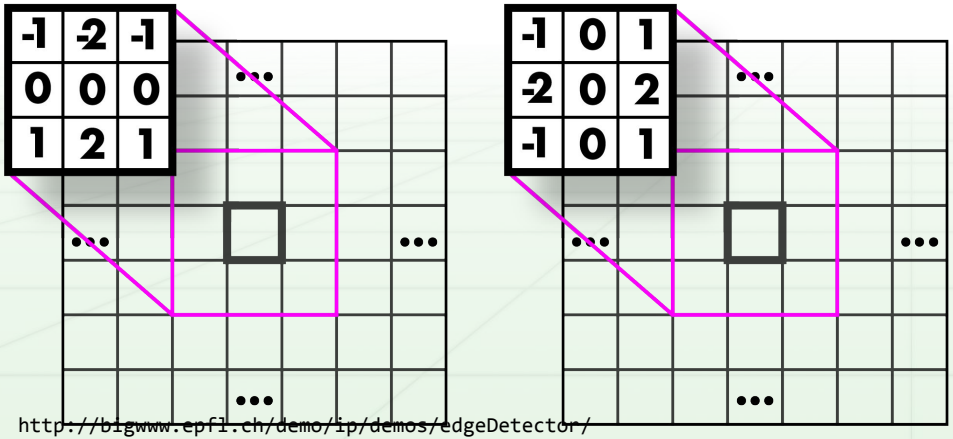
Smooth image

- You know how to do this, gaussians!



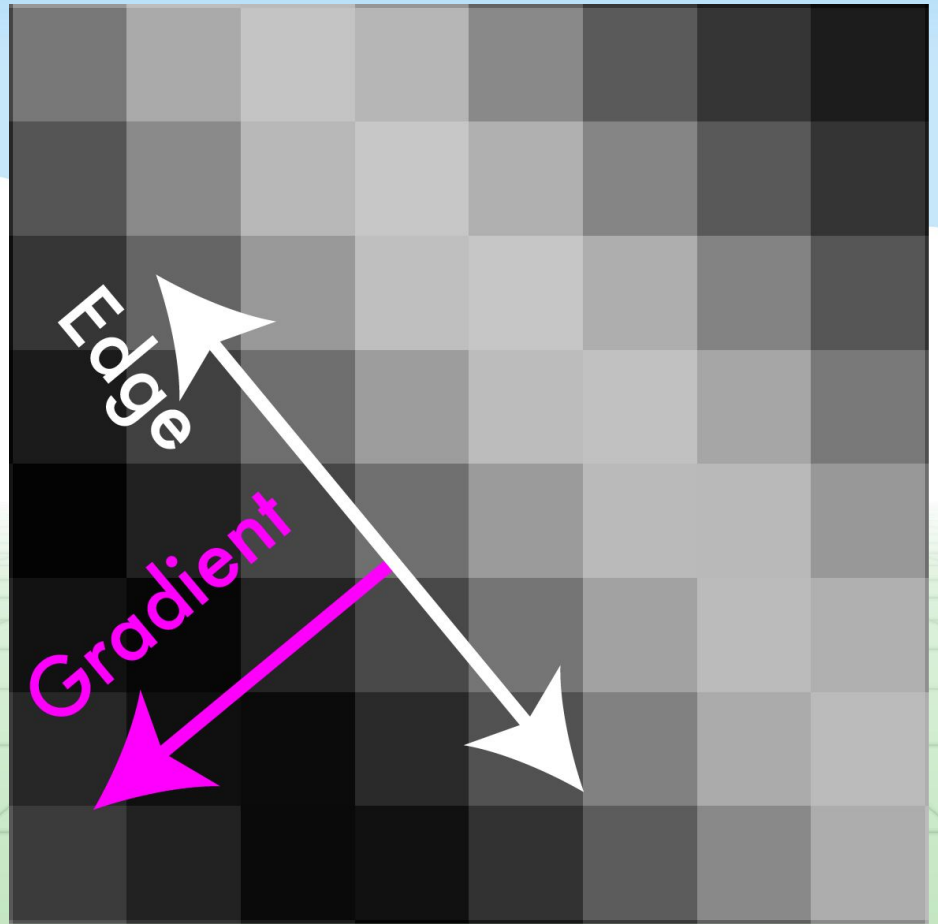
Gradient magnitude and direction

- Sobel filter

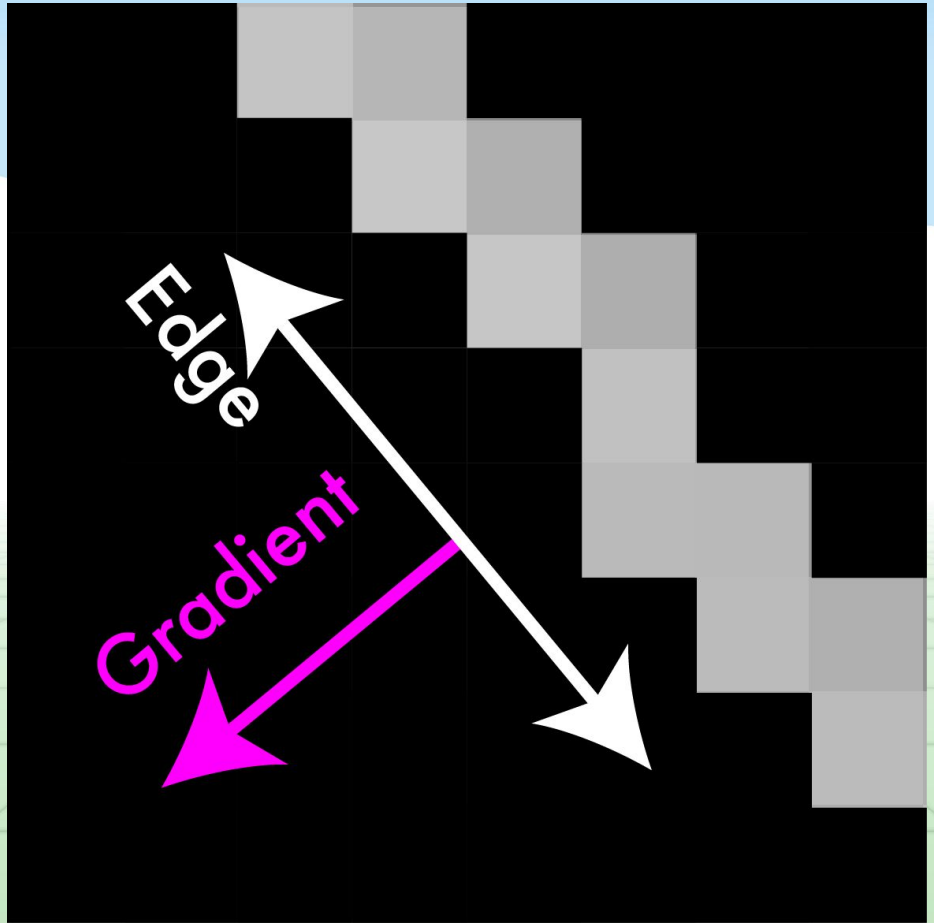


http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

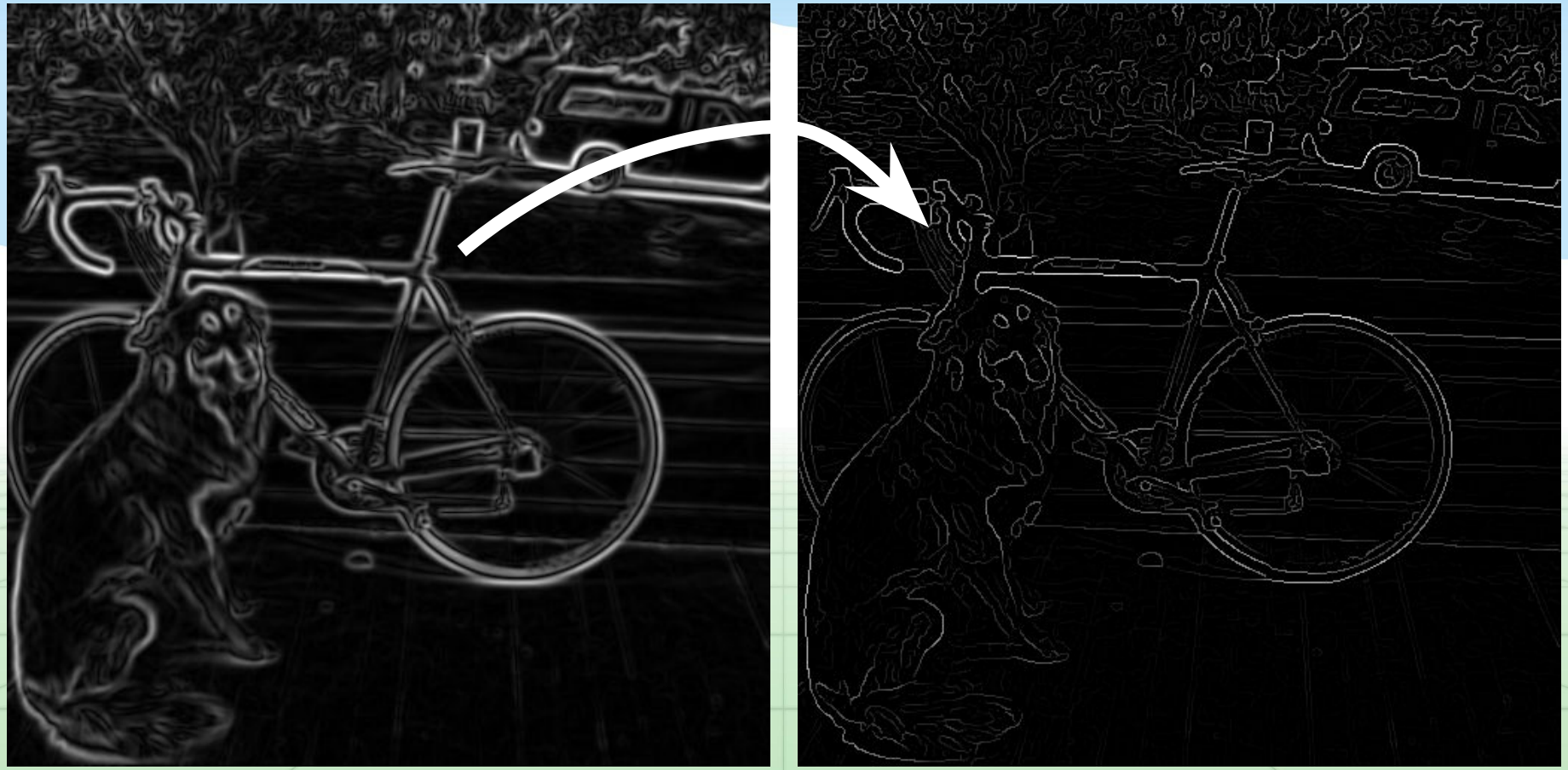
Non-maximum suppression



Non-maximum suppression



Non-maximum suppression



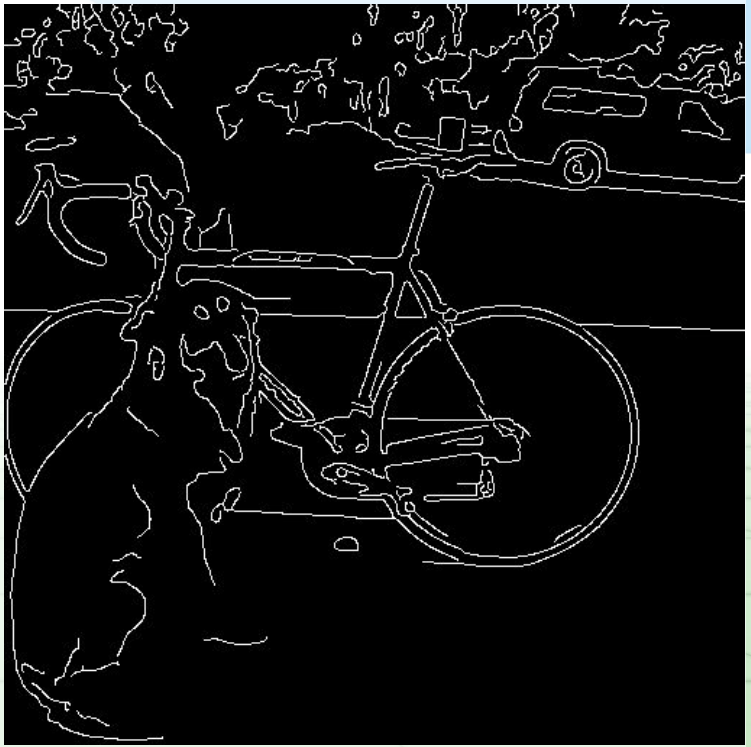
Threshold edges

- 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?

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

Connect ‘em up!

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



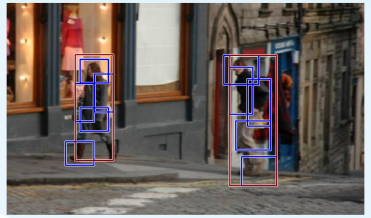
http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

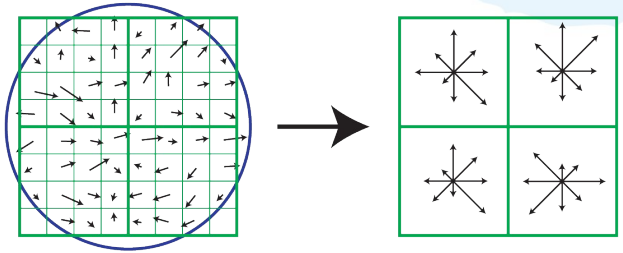
Canny Edge Detection

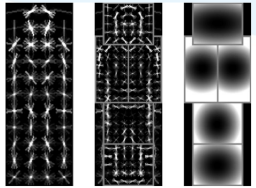

Chapter
Six
Features and RANSAC

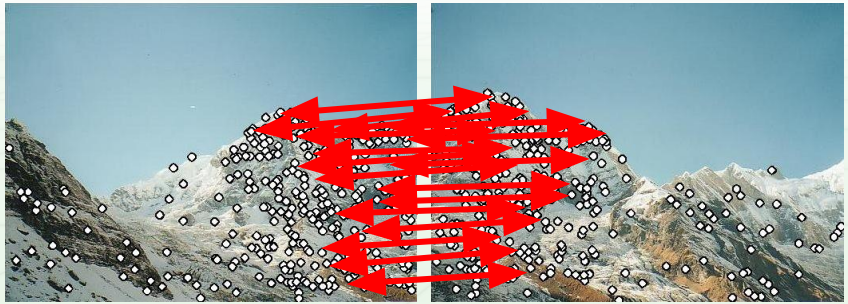
Features!

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









How to panorama

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

How close are two patches?

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

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

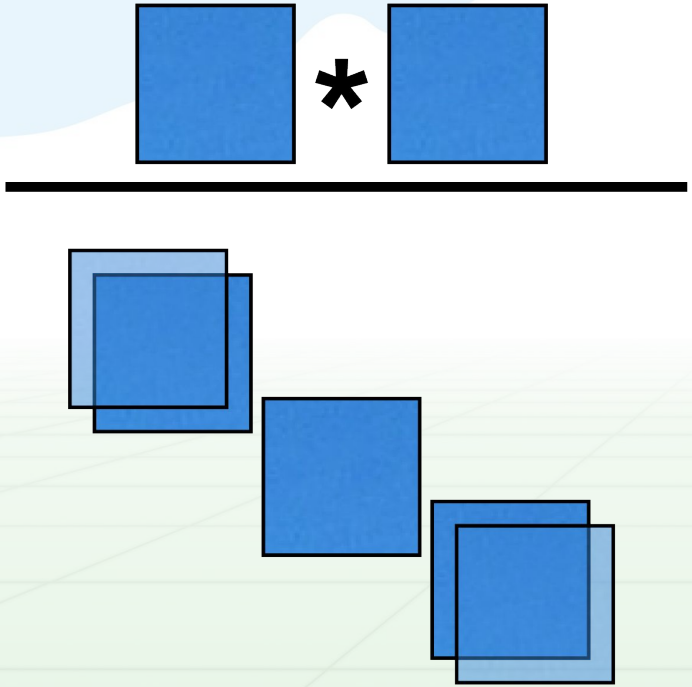
How can we find unique patches?

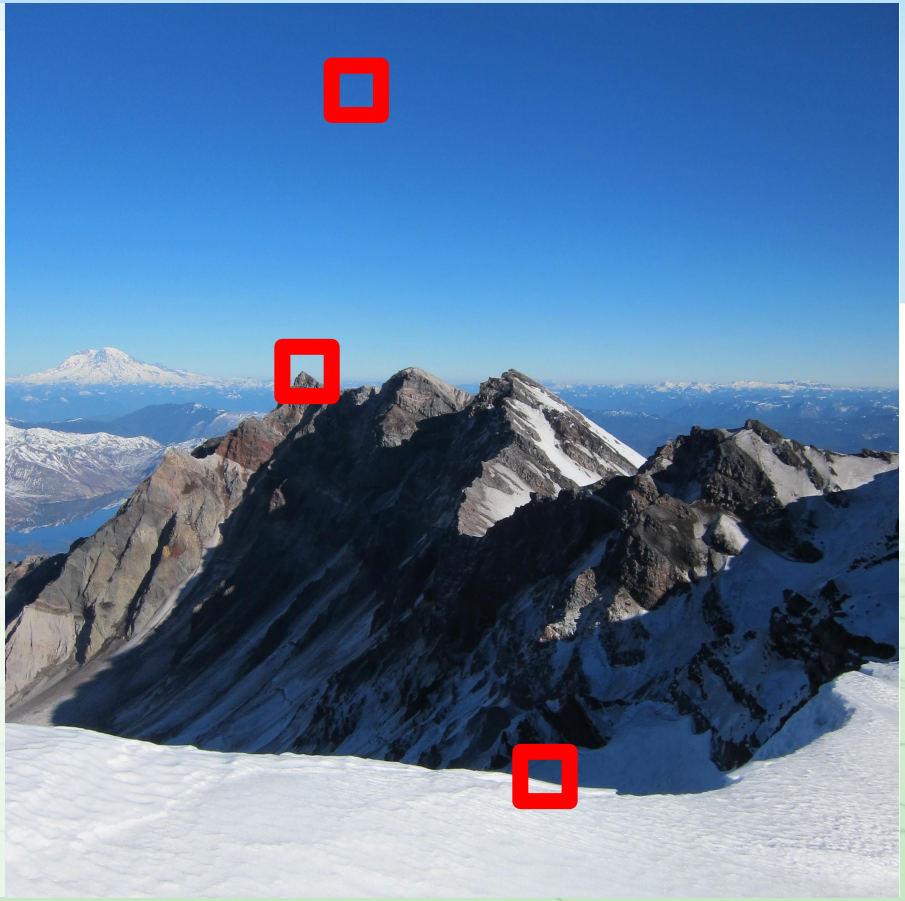
- Want a patch that is unique in the image
- Can calculate distance between patch and every other patch, lot of computation
- Instead, we could think about auto-correlation:
  - How well does image match shifted version of itself?
- $\Sigma_{\mathbf{d}}\Sigma_{x,y}\ (I(x,y) - I(x+\mathbf{d}_x, y+\mathbf{d}_y))^2$

- Measure of self-difference (how am I not myself?)

Self-difference

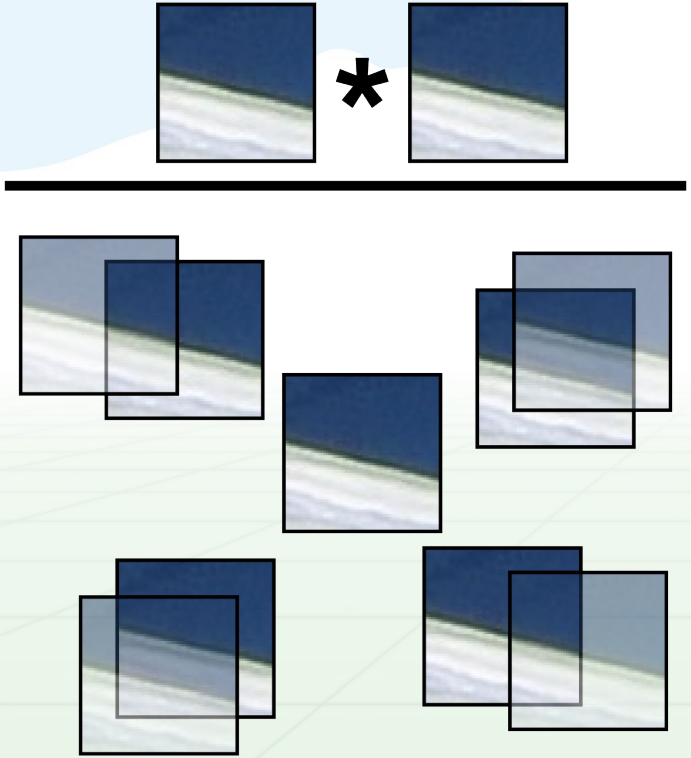
Sky: low everywhere

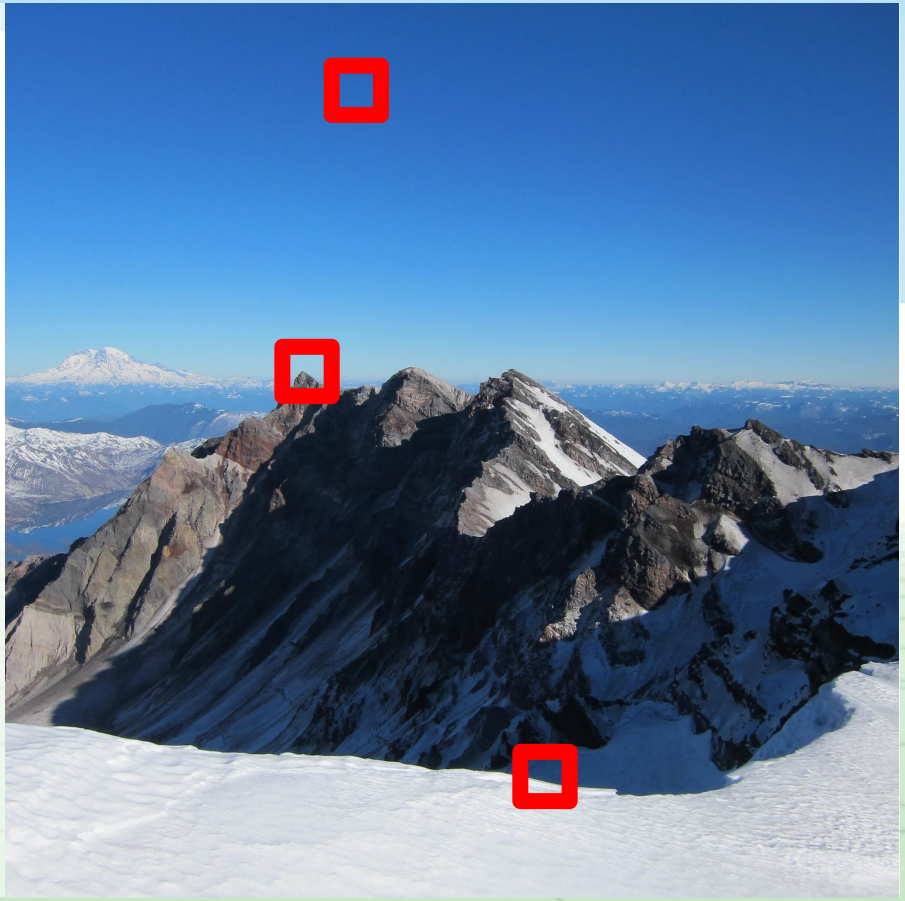




Self-difference

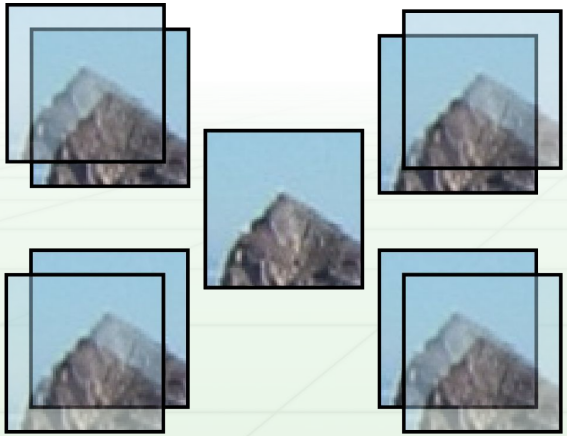
Edge: low along edge

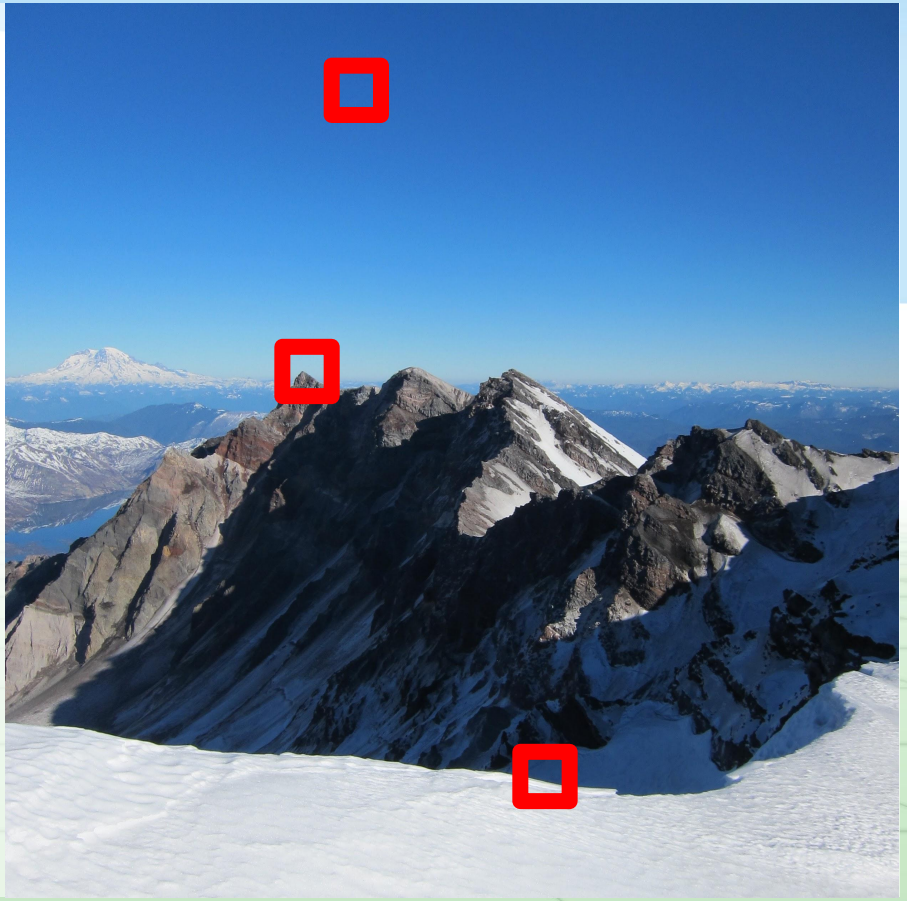




Self-difference

Corner: mostly high



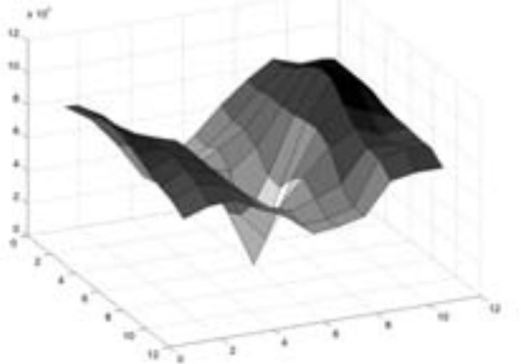


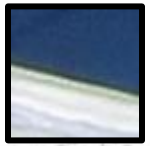
Self-difference

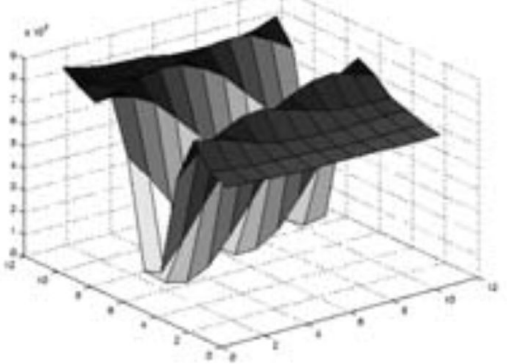
Sky: low everywhere

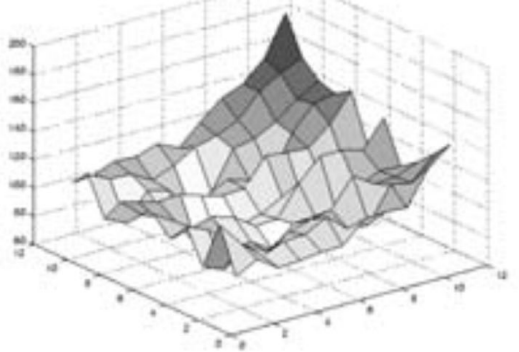
Edge: low along edge

Corner: mostly high









Self-difference is still expensive

- $\Sigma_{\mathbf{d}}\Sigma_{x,y}\ (I(x,y) - I(x+\mathbf{d}_x, y+\mathbf{d}_y))^2$
- 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

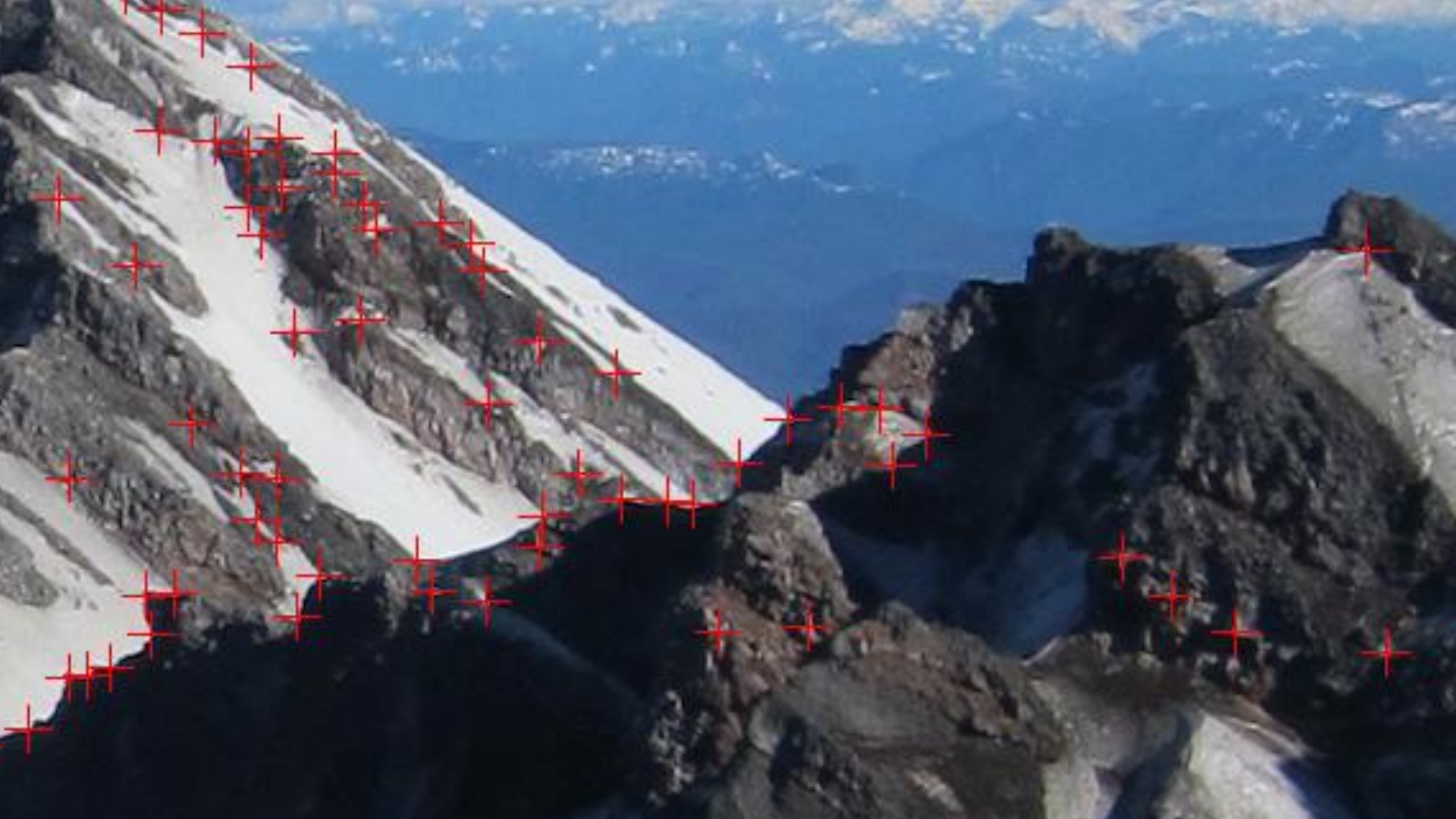
- Weighted sum of gradient information
  - $| \Sigma_i w_i I_x(i) I_x(i) \quad \Sigma_i w_i I_x(i) I_y(i) |$
  - $| \Sigma_i w_i I_x(i) I_y(i) \quad \Sigma_i w_i I_y(i) I_y(i) |$
- Can use Gaussian weighting (so many gaussians)
- Eigen vectors/values of this matrix summarize the distribution of the gradients nearby
- λ_1 and λ_2 are eigenvalues
  - λ_1 and λ_2 both small: no gradient
  - $\lambda_1 >> \lambda_2$: 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

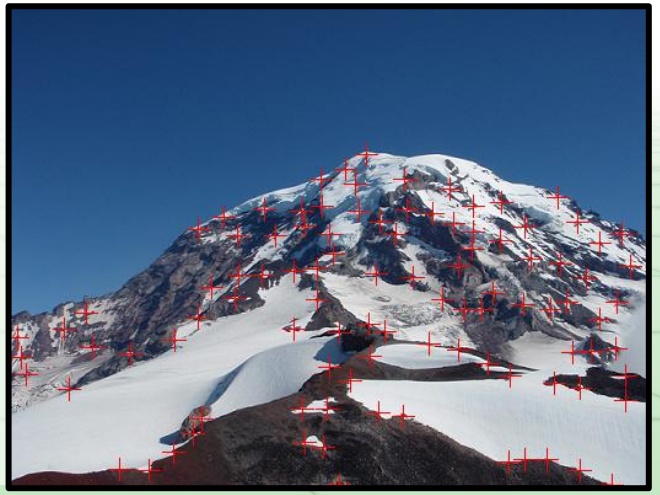
Harris Corner Detector

- Calculate derivatives I_x and I_y
- Calculate 3 measures I_xI_x, I_yI_y, I_xI_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)



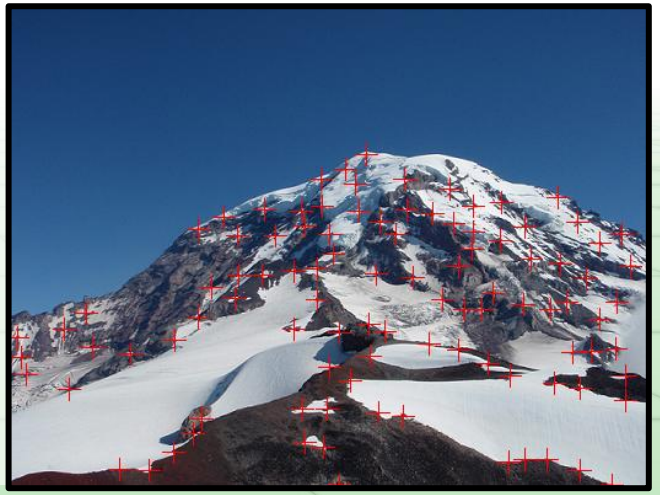
Ok, we found corners, now what?

- Need to match image patches to each other
- Need to figure out transform between images



Ok, we found corners, now what?

- Need to match image patches to each other
  - What is a patch? How do we look for matches? Pixels?
- Need to figure out transform between images
  - How can we transform images?
  - How do we solve for this transformation given matches?

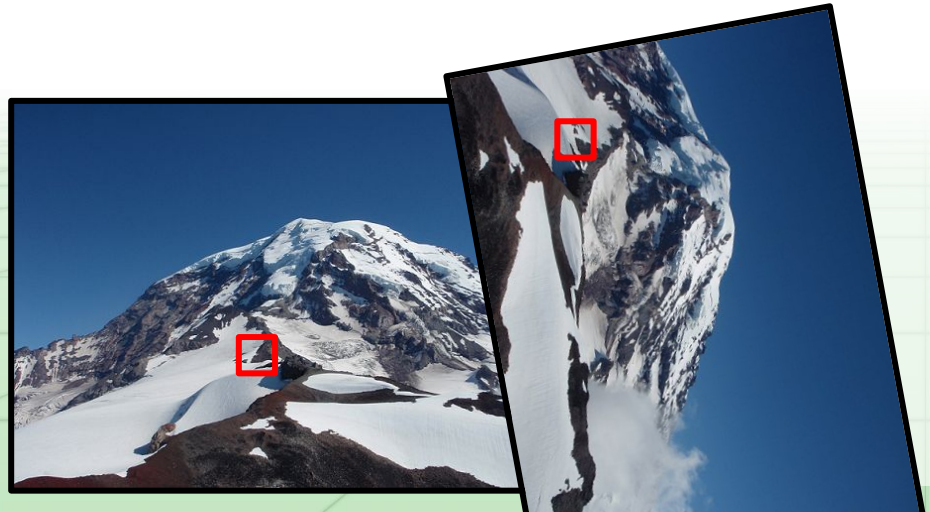


Matching patches: descriptors!

- We want a way to represent an image patch
- Can be very simple, just pixels!
- Finding matching patch is easy, distance metric:
  - $\Sigma_{x,y}\ (I(x,y) - J(x,y))^2$
  - What problems are there with just using pixels?

Matching patches: descriptors!

- We want a way to represent an image patch
- Can be very simple, just pixels!
- Finding matching patch is easy, distance metric:
  - $\Sigma_{x,y} (I(x,y) - J(x,y))^2$
  - What problems are there with just using pixels?

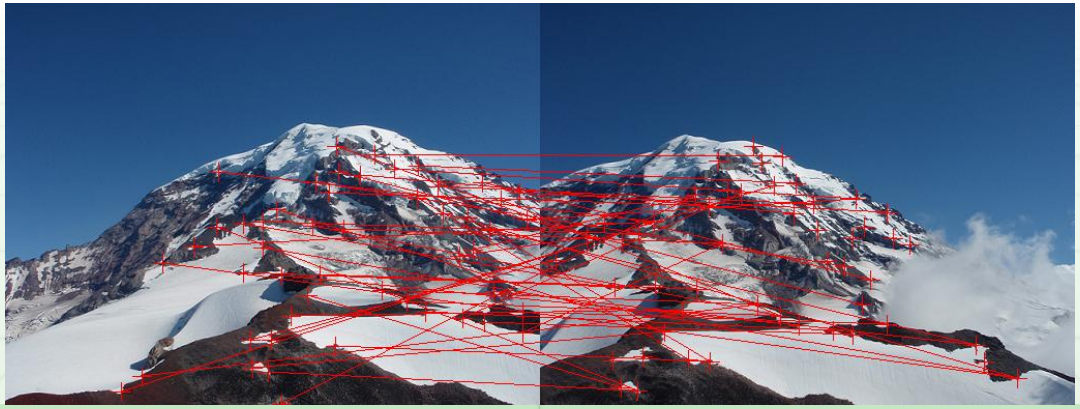


Matching patches: descriptors!

- We want a way to represent an image patch
- Can be very simple, just pixels!
- Finding matching patch is easy, distance metric:
  - $\Sigma_{x,y}\ (I(x,y) - J(x,y))^2$
  - What problems are there with just using pixels?
- Descriptors can be more complex
  - Gradient information
  - How much context?
  - Edges, etc. we'll talk more about descriptors later!

Matching patches: descriptors!

- Already have our patches that are likely “unique”-ish
- Loop over good patches in one image
  - Find best match in other image
- Do something with them?



How can we transform images?

- Need to warp one image into the other
- Many different image transforms
  - Nested hierarchy of transformations
- Need to learn some new notation to make math easier!

How can we transform images?

- **x** is a point in our image where:
  - **x** = (x, y) or in matrix terms

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

Say we want new coordinate system

- Map points from one image into another
- Often we can use matrix operations
- Given a point x, map to new point x’ using M

$$\mathbf{x}' = \mathbf{M}\,\mathbf{x}$$

Scaling is just a matrix operation

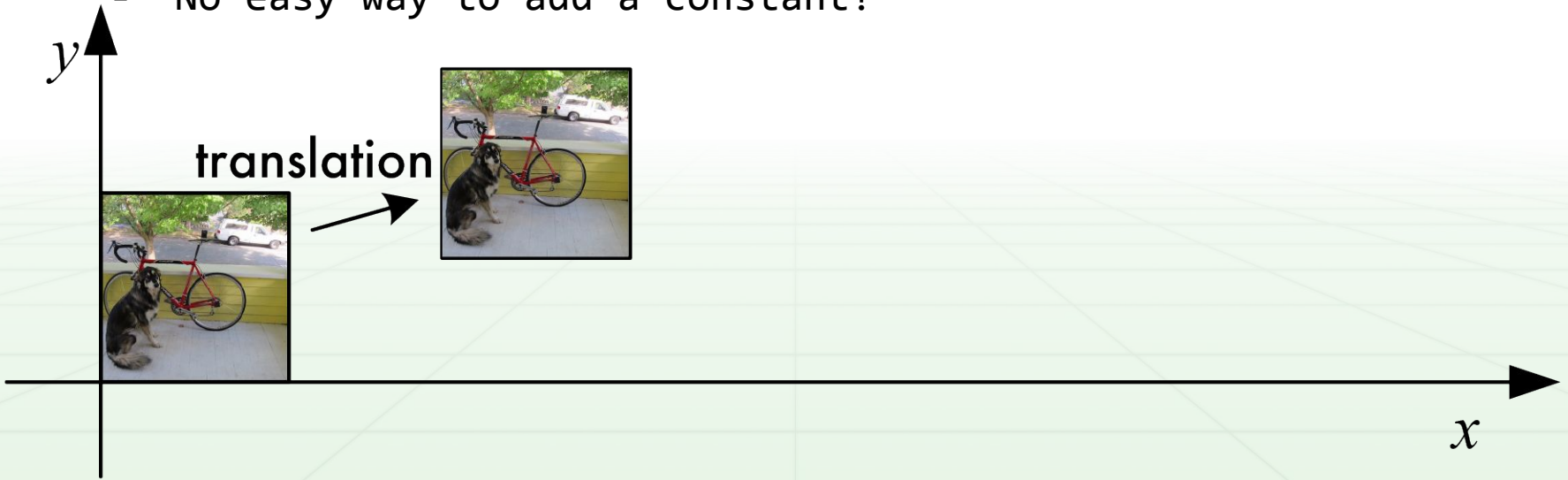
- Map points from one image into another
- Often we can use matrix operations
- Given a point x, map to new point x’ using M

$$\mathbf{x}' = \mathbf{S}\,\mathbf{x}$$

$$\mathbf{x}' = \begin{bmatrix} S & 0 \\ 0 & S \end{bmatrix} \mathbf{x}$$

Translation is harder...

- **x’ = M x**
  - Want to move x’ by dx and y’ by dy
  - How do we pick M?
  - Can only add up multiples of x or y
  - No easy way to add a constant!



Translation: add another row

- $\bar{\mathbf{x}}$ is $\mathbf{x}$ but with an added 1
- *Augmented vector*

$$\bar{\mathbf{x}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translation: add another row

- $\bar{\mathbf{x}}$ is $\mathbf{x}$ but with an added 1
- *Augmented vector*
- Now translation is easy

$$\bar{\mathbf{x}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = \begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix} \bar{\mathbf{x}}$$

Reminder, I = Identity

Common to just use **I** as a generic, whatever size identity fits here.

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ & & & \dots & \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

$$\mathbf{I}_{2\times2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{I}_{3\times3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Translation: add another row

- $\bar{\mathbf{x}}$ is $\mathbf{x}$ but with an added 1
- *Augmented vector*
- Now translation is easy

$$\bar{\mathbf{x}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = [\, \mathbf{I} \;\; \mathbf{t} \,]\, \bar{\mathbf{x}}$$

Translation: add another row

- $\bar{\mathbf{x}}$ is $\mathbf{x}$ but with an added 1
- *Augmented vector*
- Now translation is easy
- x’ = 1*x + 0*y + dx*1
- y’ = 0*x + 1*y + dy*1

$$\bar{\mathbf{x}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = [\, \mathbf{I} \;\; \mathbf{t} \,] \, \bar{\mathbf{x}}$$

Translation: add another row

- $\bar{\mathbf{x}}$ is $\mathbf{x}$ but with an added 1
- *Augmented vector*
- Now translation is easy
- x’ = 1*x + 0*y + dx*1
- y’ = 0*x + 1*y + dy*1

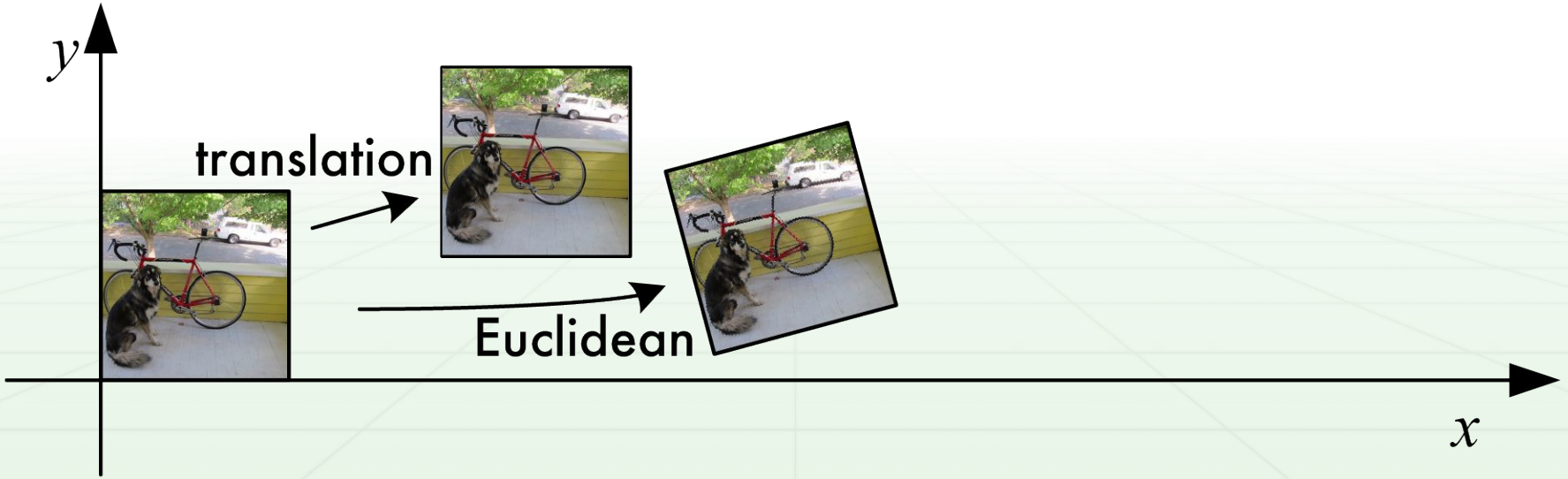
$$\bar{\mathbf{x}} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = [\, \mathbf{I} \;\; \mathbf{t} \,] \, \bar{\mathbf{x}}$$

Euclidean: rotation + translation

- Want to translate and rotate at same time
- Still just matrix operation



Euclidean: rotation + translation

- Want to translate and rotate at same time
- Still just matrix operation

$$\mathbf{x}' = [\, \mathbf{R} \;\; \mathbf{t} \,] \, \bar{\mathbf{x}}$$

Euclidean: rotation + translation

- Want to translate and rotate at same time
- Still just matrix operation
- **R** is rotation matrix, **t** is translation

$$\mathbf{x}' = [\,\mathbf{R}\ \ \mathbf{t}\,]\,\bar{\mathbf{x}}$$

$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Euclidean: rotation + translation

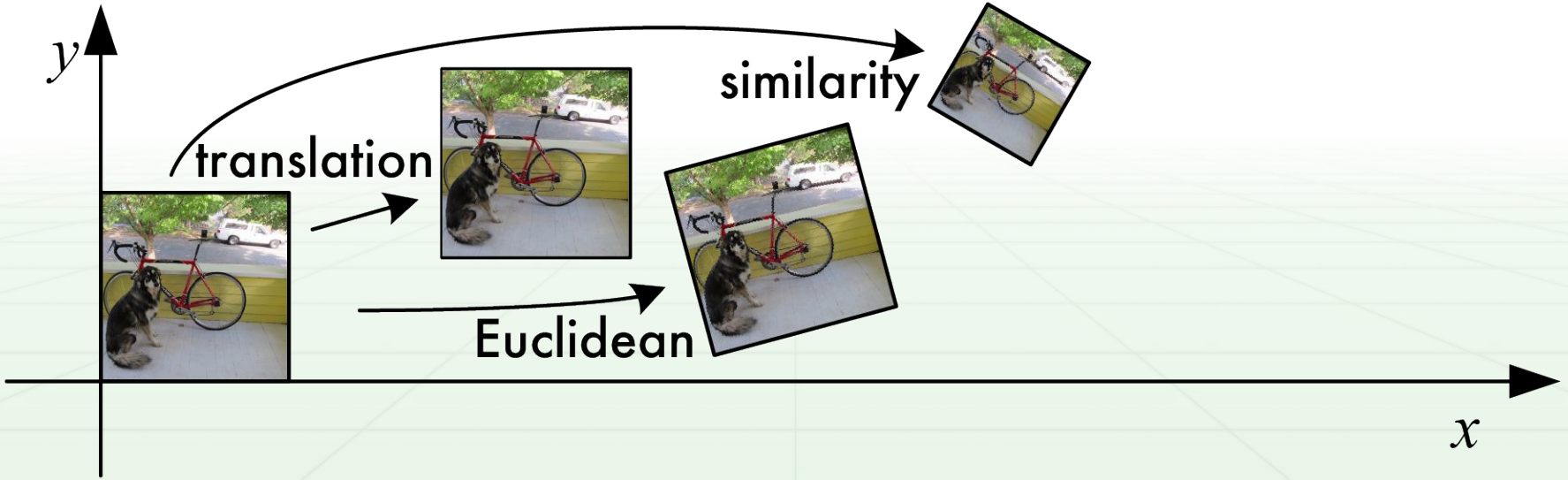
- Want to translate and rotate at same time
- Still just matrix operation
- **R** is rotation matrix, **t** is translation

$$\mathbf{x}' = \begin{bmatrix} \cos\theta & -\sin\theta & dx \\ \sin\theta & \cos\theta & dy \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{x}' = [\,\mathbf{R} \;\; \mathbf{t}\,]\,\bar{\mathbf{x}}$$

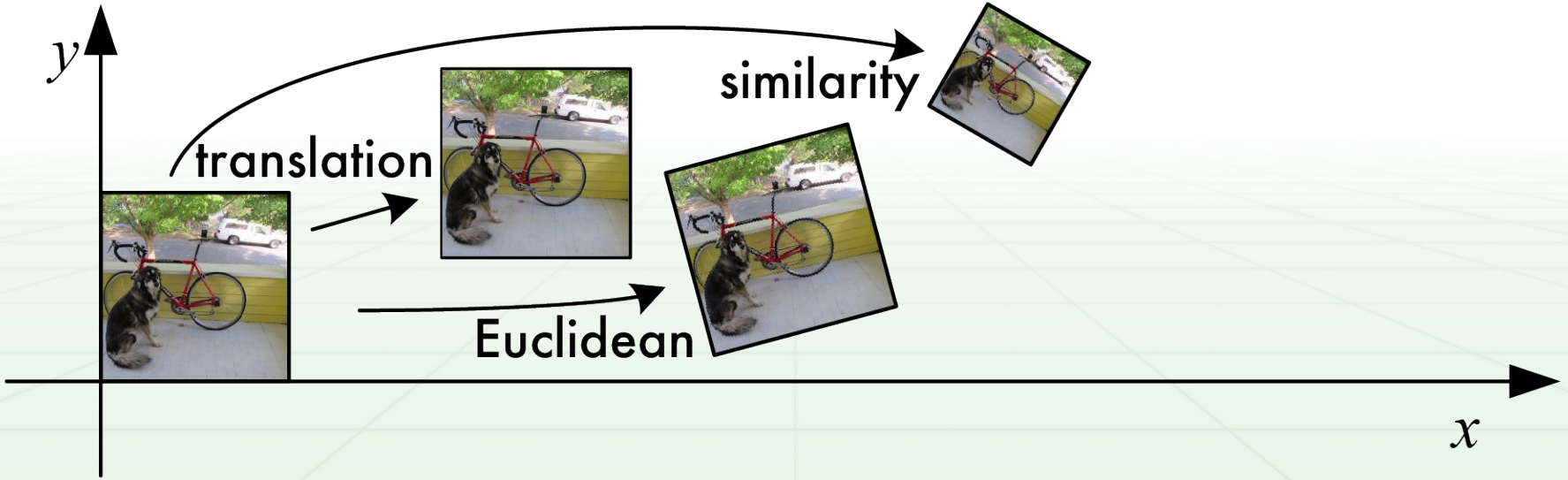
$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Similarity: scale, rotate, translate



Similarity: scale, rotate, translate

$$\mathbf{x}' = [\, s\mathbf{R} \;\; \mathbf{t} \,]\, \bar{\mathbf{x}}$$

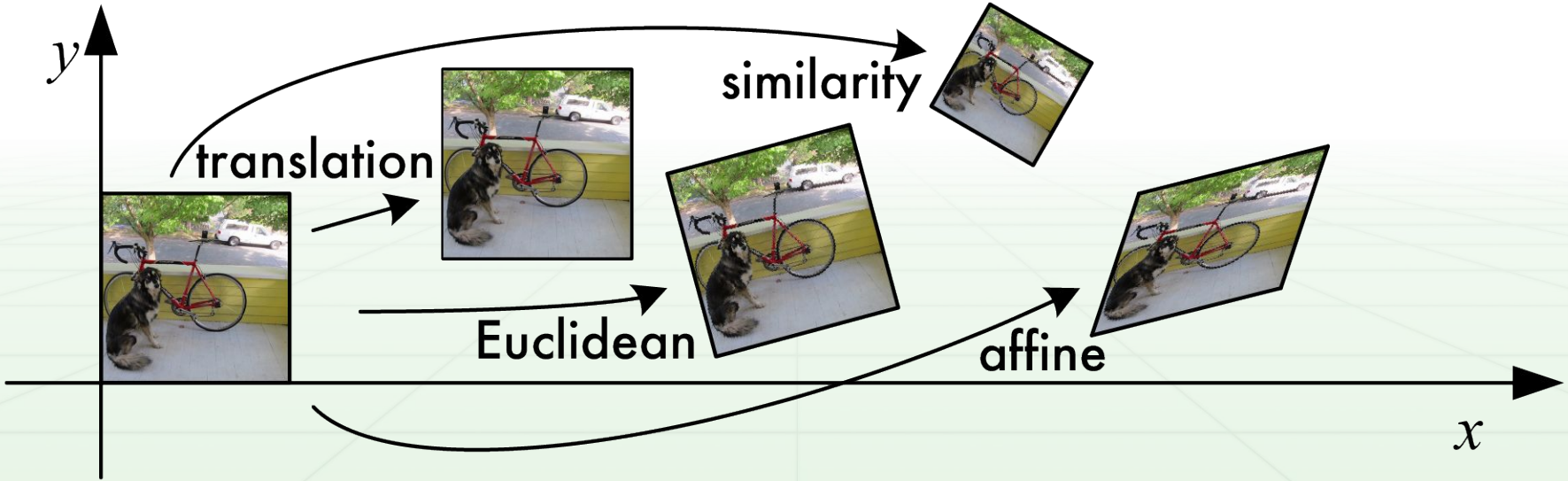


Similarity: scale, rotate, translate

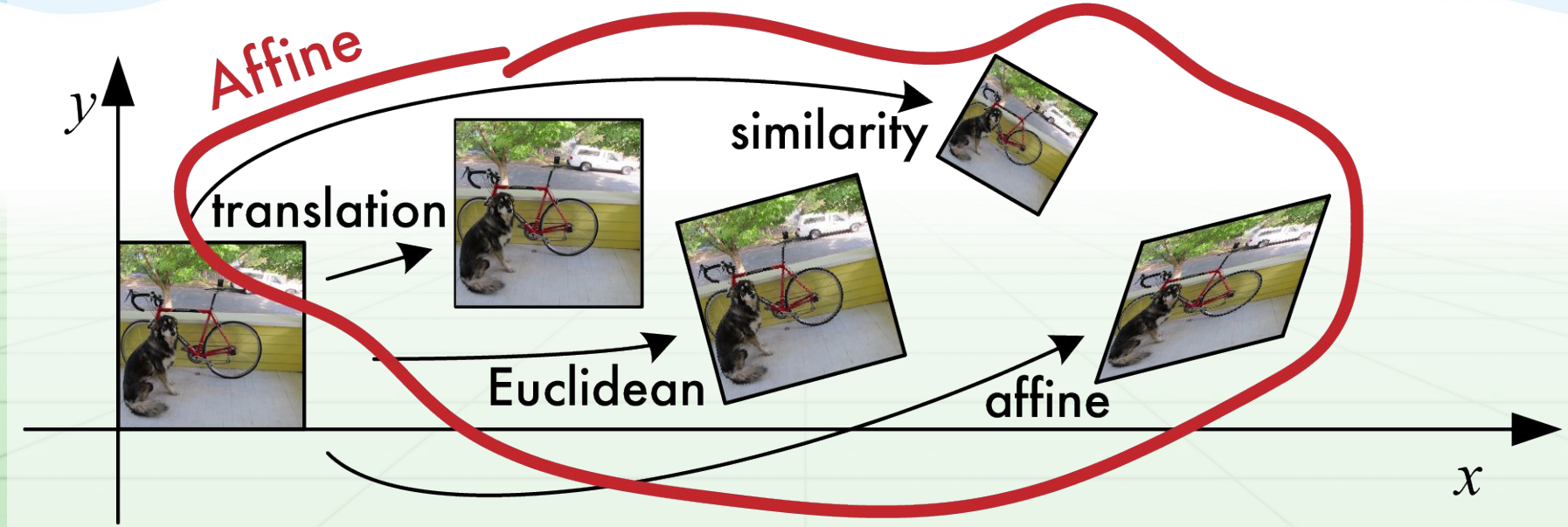
$$\mathbf{x}' = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix} \bar{\mathbf{x}}$$

$$\mathbf{x}' = \begin{bmatrix} a & -b & dx \\ b & a & dy \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine: scale, rotate, translate, shear



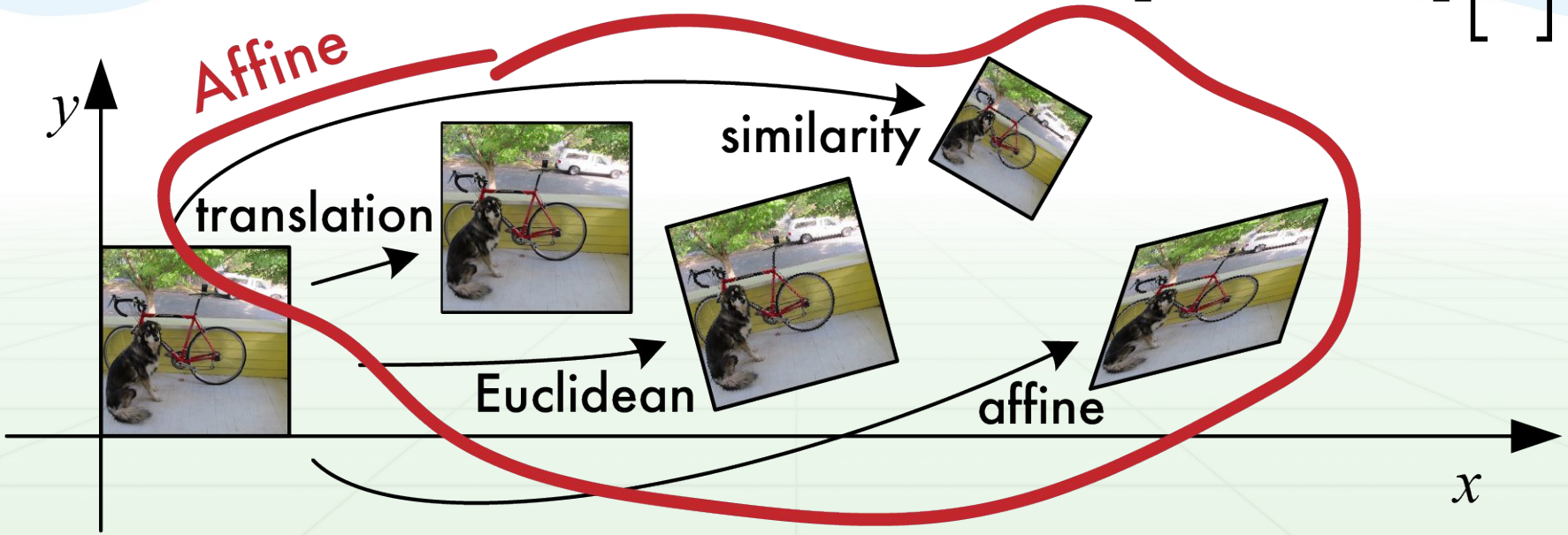
Affine: scale, rotate, translate, shear



Affine: scale, rotate, translate, shear

General case of 2x3 matrix

$$\mathbf{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



Combinations are still affine

Say you want to translate, then rotate, then translate back, then scale.

$\mathbf{x'} = \mathbf{S}\ \mathbf{t}\ \mathbf{R}\ \mathbf{t}\ \bar{\mathbf{x}} = \mathbf{M}\ \bar{\mathbf{x}}$,

If $\mathbf{M} = (\mathbf{S}\ \mathbf{t}\ \mathbf{R}\ \mathbf{t})$

$\mathbf{M}$ is still affine transformation

Wait, but these are all 2x3, how to we multiply them together?

Added row to transforms

$$\bar{\mathbf{x}}' = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

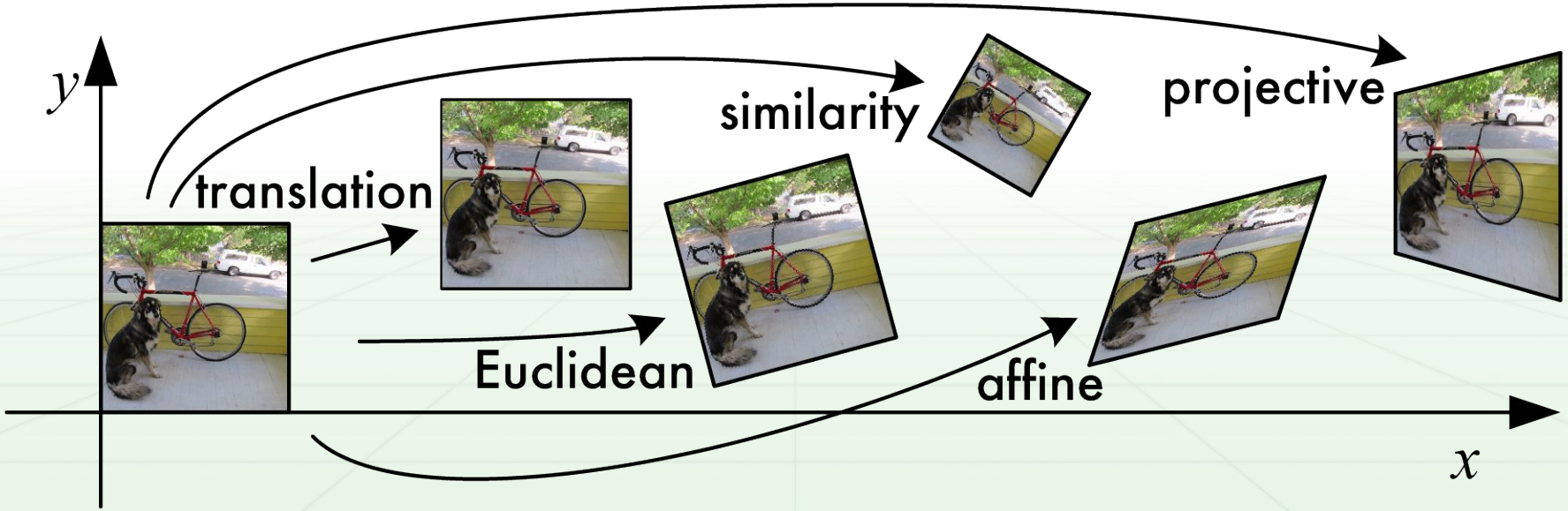
$$\bar{\mathbf{x}}' = \begin{bmatrix} \cos\theta & -\sin\theta & dx \\ \sin\theta & \cos\theta & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\bar{\mathbf{x}}' = \begin{bmatrix} a & -b & dx \\ b & a & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\bar{\mathbf{x}}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Projective transform

- AKA perspective transform or homography
- Wait but affine was any 2x3 matrix...



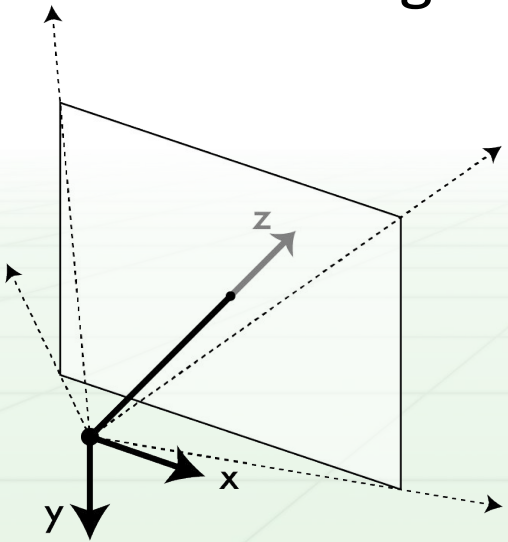
Need some new coordinates!

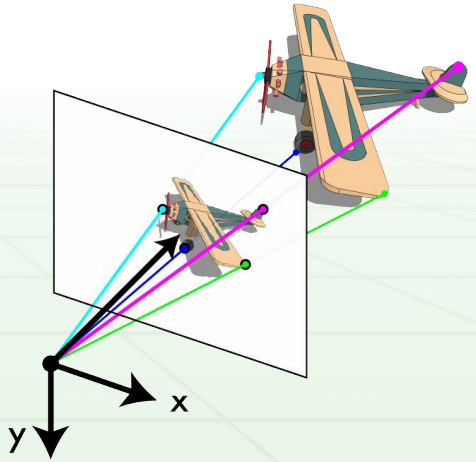
- Homogeneous coordinate system
  - Useful because we can do this kind of transform
- Each point in 2d is actually a vector in 3d
- Equivalent up to scaling factor
- Have to normalize to get back to 2d

$$\tilde{\mathbf{x}} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix} \qquad \bar{\mathbf{x}} = \tilde{\mathbf{x}} / \tilde{w}$$

Why does this make sense?

- Remember our pinhole camera model
- Every point in 3d projects onto our viewing plane through our aperture
- Points along a vector are indistinguishable





Projective transform

- AKA perspective transform or homography
- Wait but affine was any 2x3 matrix...
- Homography is general 3x3 matrix
- Multiplication by scalar is equivalent

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\,\tilde{\mathbf{x}}$$

Projective transform

- AKA perspective transform or homography
- Wait but affine was any 2x3 matrix...
- Homography is general 3x3 matrix
- Multiplication by scalar: equivalent projection
  - 3*H ~ H

$$\tilde{\mathbf{x}}' = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix}$$

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\,\tilde{\mathbf{x}}$$

Using homography to project point

- Multiply **x~** by **H~** to get **x'~**
- Convert to **x'** by dividing by **w'~**

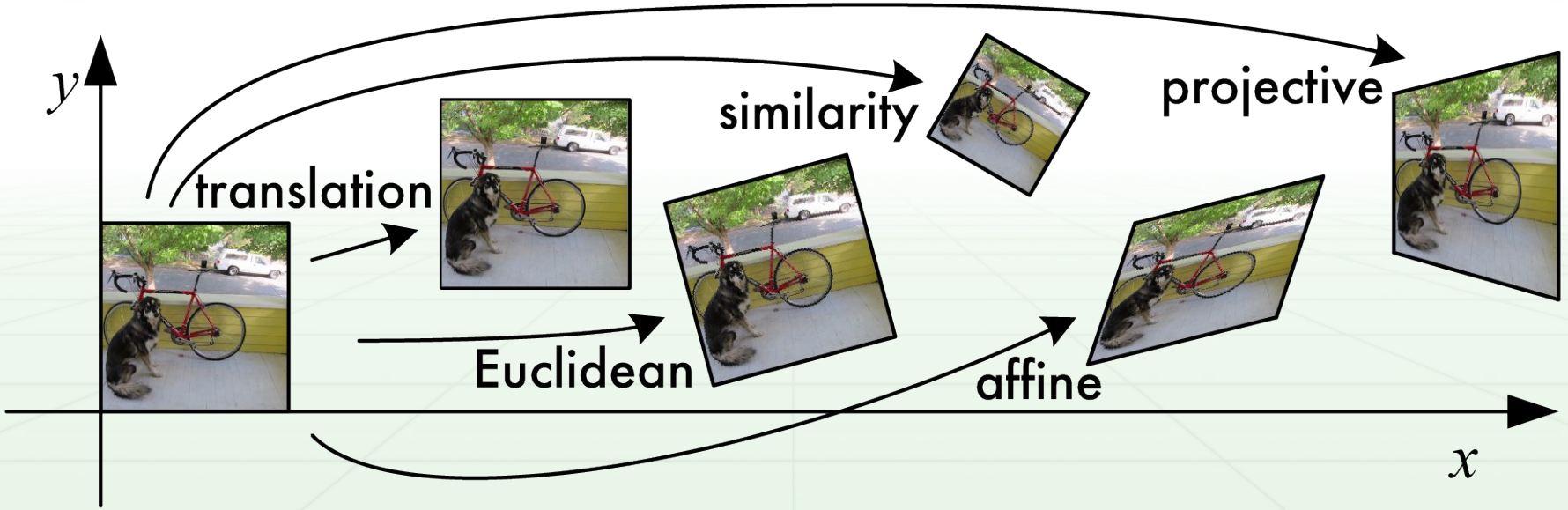
$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\,\tilde{\mathbf{x}}$$

$$\begin{bmatrix} \tilde{x}' \\ \tilde{y}' \\ \tilde{w}' \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix}$$

$$\bar{\mathbf{x}} = \tilde{\mathbf{x}} / \tilde{w}$$

Lots to choose from

- What do each of them do?
- Which is right for panorama stitching?



How hard are they to recover?

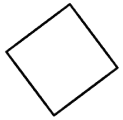
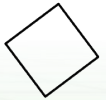
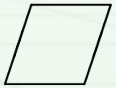
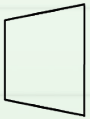
$$\bar{\mathbf{x}}' = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \bar{\mathbf{x}}' = \begin{bmatrix} \cos\theta & -\sin\theta & dx \\ \sin\theta & \cos\theta & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\bar{\mathbf{x}}' = \begin{bmatrix} a & -b & dx \\ b & a & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\bar{\mathbf{x}}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

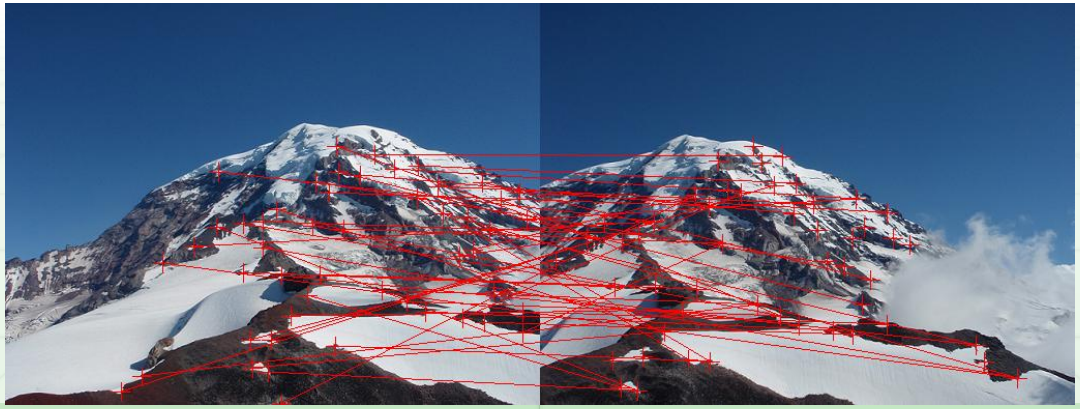
$$\tilde{\mathbf{x}}' = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{bmatrix}$$

Lots to choose from

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} \mid \mathbf{t} \end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} \mid \mathbf{t} \end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} \mid \mathbf{t} \end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

Say we want affine transformation

- Have our matched points
- Want to estimate **A** that maps from **x** to **x’**
- **xA = x’**



Say we want affine transformation

- Have our matched points
- Want to estimate **A** that maps from **x** to **x’**
- **xA = x’**
- How many degrees of freedom?

Say we want affine transformation

- Have our matched points
- Want to estimate **A** that maps from **x** to **x’**
- **xA = x’**
- How many degrees of freedom?
  - 6
- How many knowns do we get with one match? **mX = n**

$$\mathbf{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Say we want affine transformation

- Have our matched points
- Want to estimate **A** that maps from **x** to **x’**
- **xA = x’**
- How many degrees of freedom?
  - 6
- How many knowns do we get with one match? **mA = n**
  - 2
  - $n_x = a_{00}*m_x + a_{01}*m_y + a_{02}*1$
  - $n_y = a_{10}*m_x + a_{11}*m_y + a_{12}*1$

$$\mathbf{x'} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Say we want affine transformation

- How many knowns do we get with one match? **mA = n**
  - $n_x = a_{00} * m_x + a_{01} * m_y + a_{02} * 1$
  - $n_y = a_{10} * m_x + a_{11} * m_y + a_{12} * 1$
  - Solve **M a = b**
    - $M^T M a = M^T b$
    - $a = (M^T M)^{-1} M^T b$
  - Still works if overdetermined
    - WHY???

$$\underset{\mathbf{M}}{\begin{bmatrix} m_{x1} & m_{y1} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_{x1} & m_{y1} & 1 \\ m_{x2} & m_{y2} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_{x2} & m_{y2} & 1 \\ m_{x3} & m_{y3} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & m_{x3} & m_{y3} & 1 \\ & & & \cdots & & \\ & & & \cdots & & \end{bmatrix}} \underset{\mathbf{a}}{\begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \end{bmatrix}} = \underset{\mathbf{b}}{\begin{bmatrix} n_{x1} \\ n_{y1} \\ n_{x2} \\ n_{y2} \\ n_{x3} \\ n_{y3} \end{bmatrix}}$$