Computer Vision

CSE 455
Describing and Matching

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

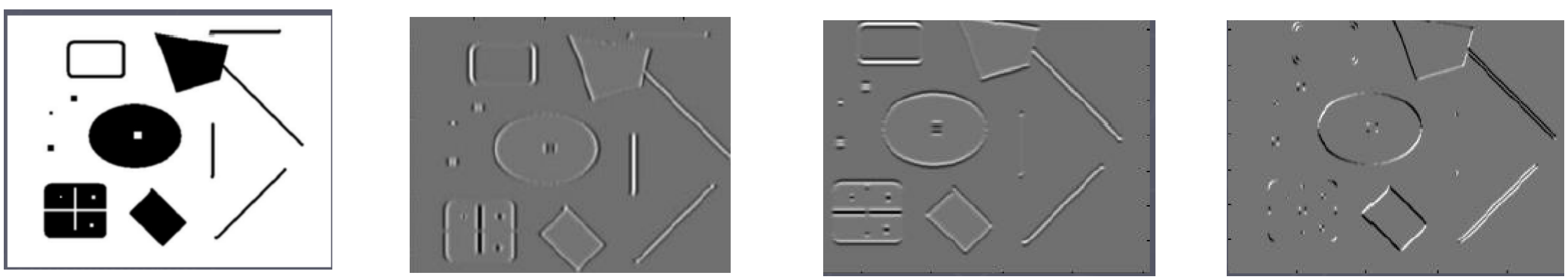
Review

- Harris Corner Dector
- SIFT Key Point Detector
- On to Descriptors

Second Moment Matrix or Harris Matrix

$$H = \begin{vmatrix} \Sigma_i w_i \, I_x \, I_x & \Sigma_i w_i \, I_x \, I_y \\ \Sigma_i w_i \, I_x \, I_y & \Sigma_i w_i \, I_y \, I_y \end{vmatrix}$$

2 x 2 matrix of image derivatives smoothed by Gaussian weights.



Notation: $I_x \Leftrightarrow \frac{\partial I}{\partial x}$ $\quad I_y \Leftrightarrow \frac{\partial I}{\partial y}$ $\quad I_x I_y \Leftrightarrow \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}$

- First compute I_x, I_y, and $I_x \, I_x$, $I_y \, I_y$, $I_x I_y$; then apply Gaussian to each.

From HW3: Structure matrix

- Weighted sum of gradient information
  - $| \Sigma_i w_i I_x(i) I_x(i) \quad \Sigma_i w_i I_x(i) I_y(i) |$
  - $| \Sigma_i w_i I_x(i) I_y(i) \quad \Sigma_i w_i I_y(i) I_y(i) |$

We'll tell you how to store this!

- Use Gaussian weighting
- 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

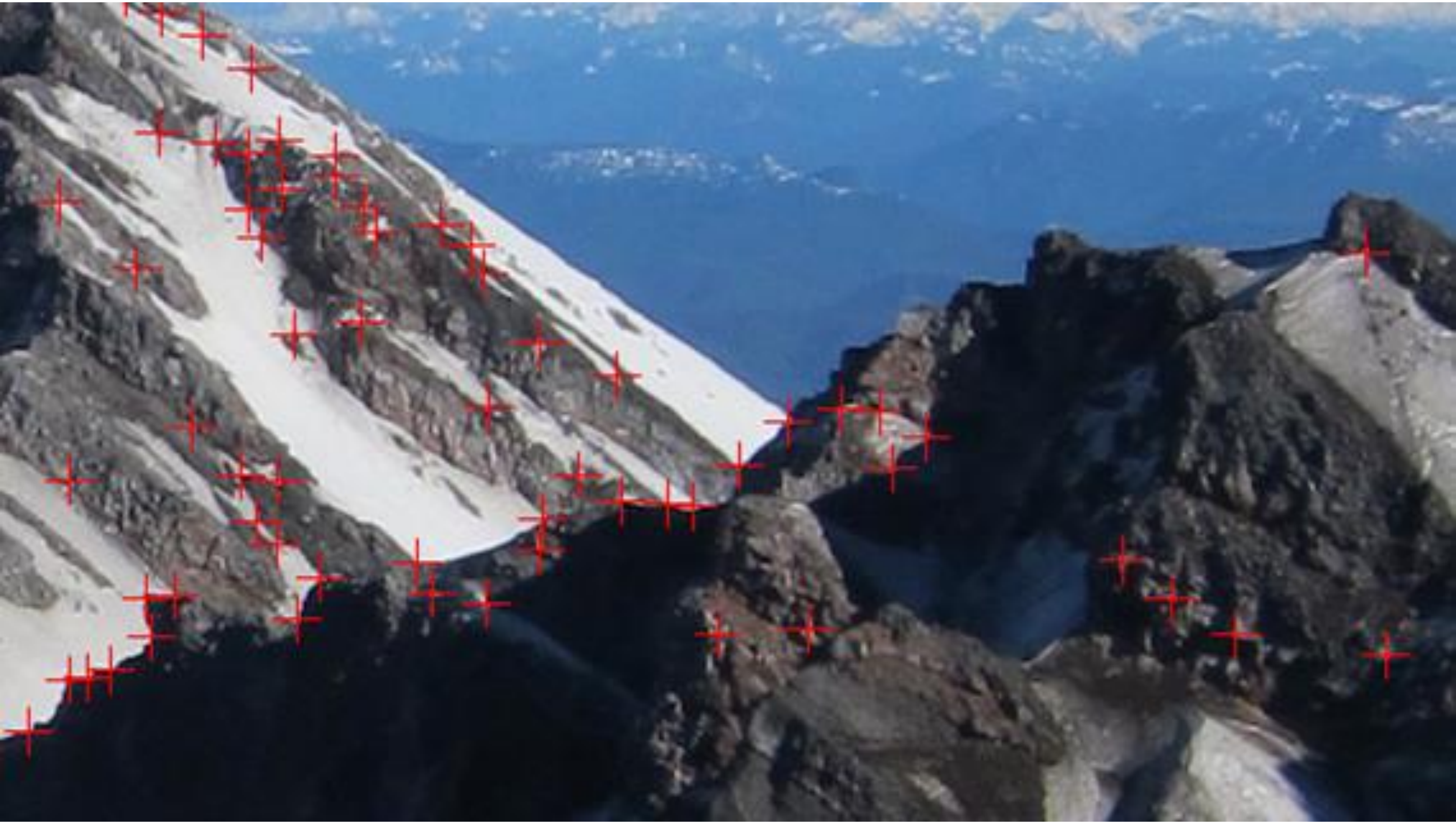
Estimating Response

$$H = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad \lambda_{\pm} = \frac{1}{2}\left((a+d) \pm \sqrt{4bc + (a-d)^2}\right)$$

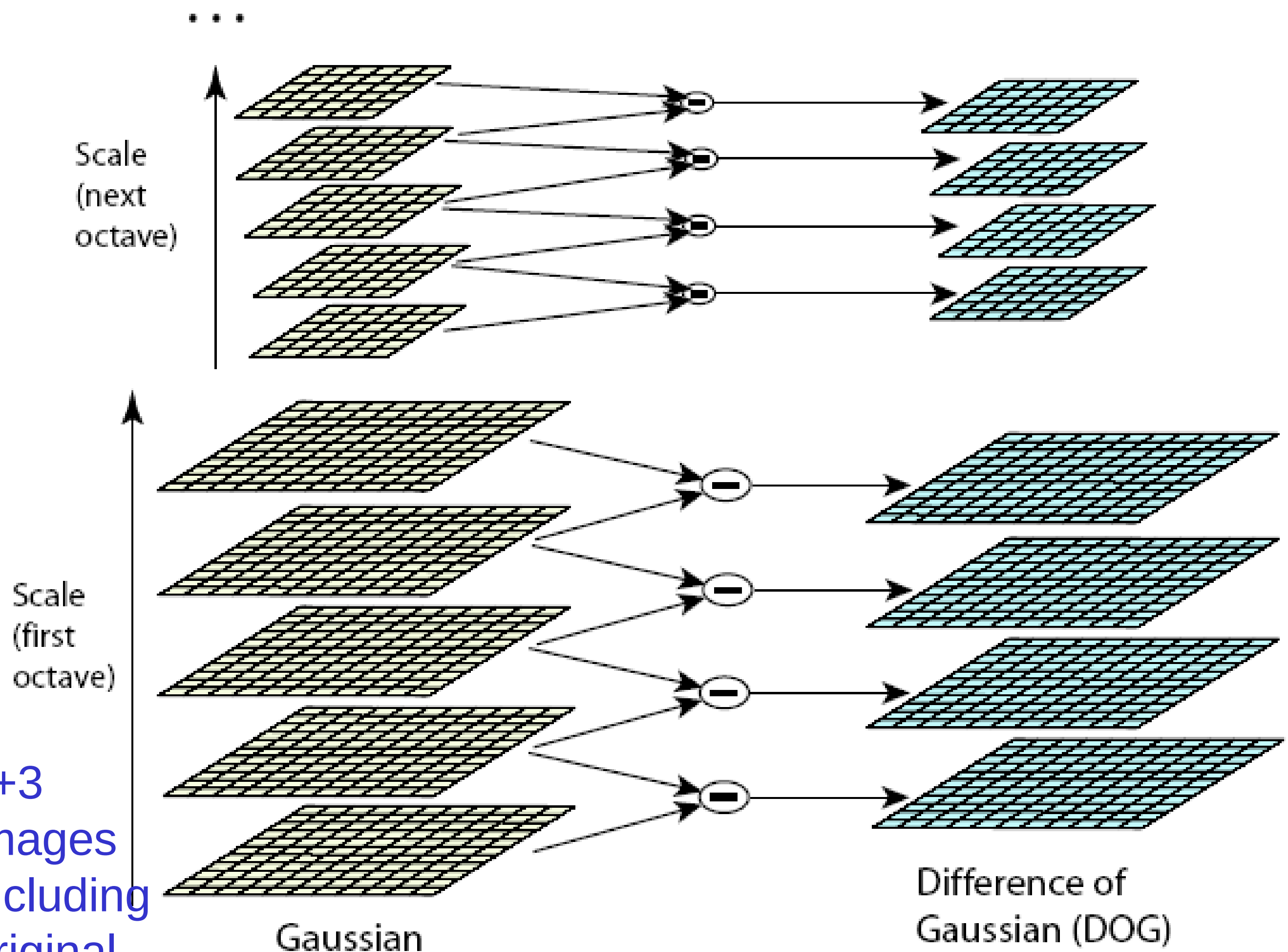
- A few methods we use to estimate:
- Calculate these directly from the 2x2 matrix
  - det(S) = ad – bc = $\lambda_1 * \lambda_2$
  - trace(S) = a + d = $\lambda_1 + \lambda_2$
- **Estimate formula 1: R = det(S) - α trace(S)^2 = $\lambda_1\lambda_2 - \alpha(\lambda_1+\lambda_2)^2$
- ---Estimate formula 2: R = det(S) / 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
- Non-max suppression (just like Canny)



Lowe's Pyramid Scheme



s+2 filters

$\sigma_{s+1}=2^{(s+1)/s}\sigma_0$

.

.

$\sigma_i=2^{i/s}\sigma_0$

.

.

$\sigma_2=2^{2/s}\sigma_0$

$\sigma_1=2^{1/s}\sigma_0$

σ_0

s+3 images including original

s+2 differ-ence images

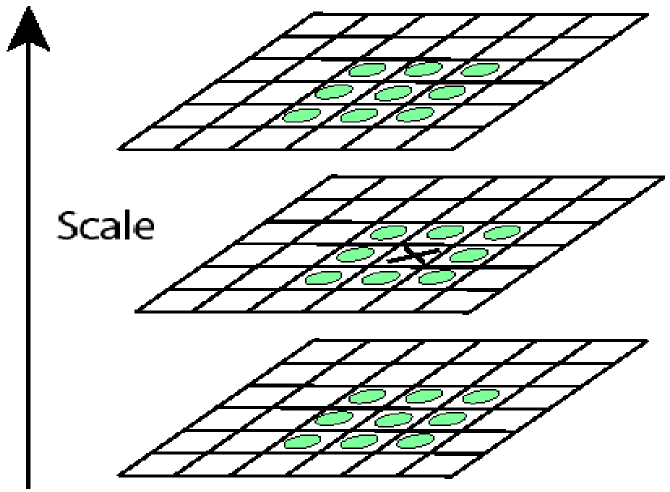
The parameter **s** determines the number of images per octave.

Key point localization

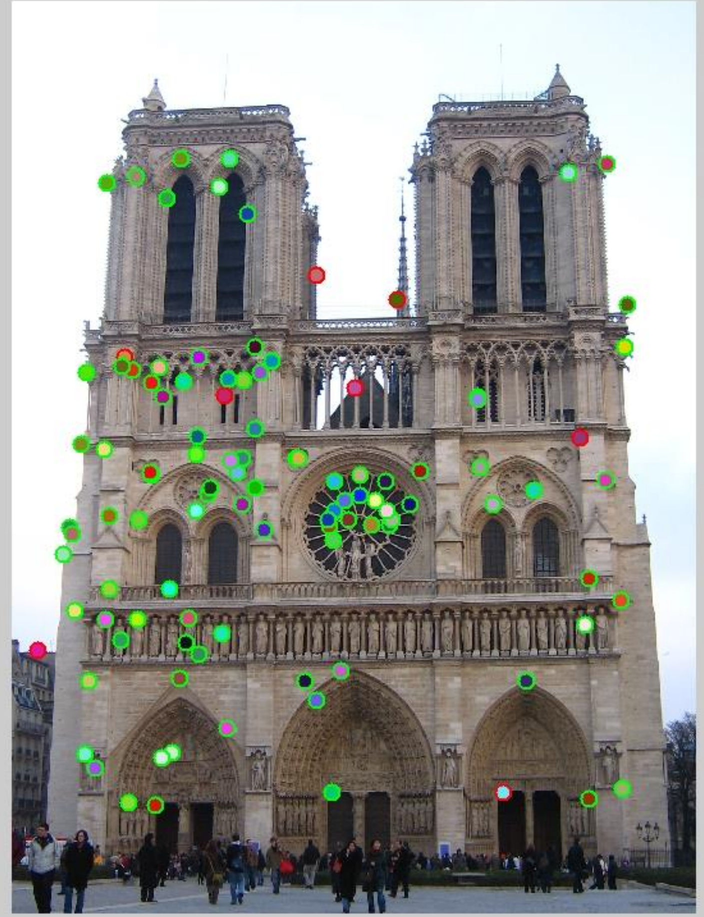
s+2 difference images.
top and bottom ignored.
s planes searched.

- Detect maxima and minima of difference-of-Gaussian in scale space

- Each point is compared to its 8 neighbors in the current image and 9 neighbors each in the scales above and below



For each max or min found, output is the **location** and the **scale**.



Important Point

- People just say “SIFT”.
- But there are TWO parts to SIFT.

1. an interest point detector
2. a region descriptor

- They are independent. Many people use the region descriptor without looking for the points.

Descriptors

- How are we going to describe the patch around each of the interest points we find in order to match them between images?
- We will use a very simple descriptor.
- Let’s look at that first.

Simple Normalized Descriptor

interest point	neighborhood around interest point	normalized neighborhood around interest point
201	45 56 200 46 201 200 85 101 105	156 145 1 155 0 1 116 100 96

- The simple descriptor just subtracts each of the neighbors from the center value, including the center itself, to normalize for lighting and exposure.

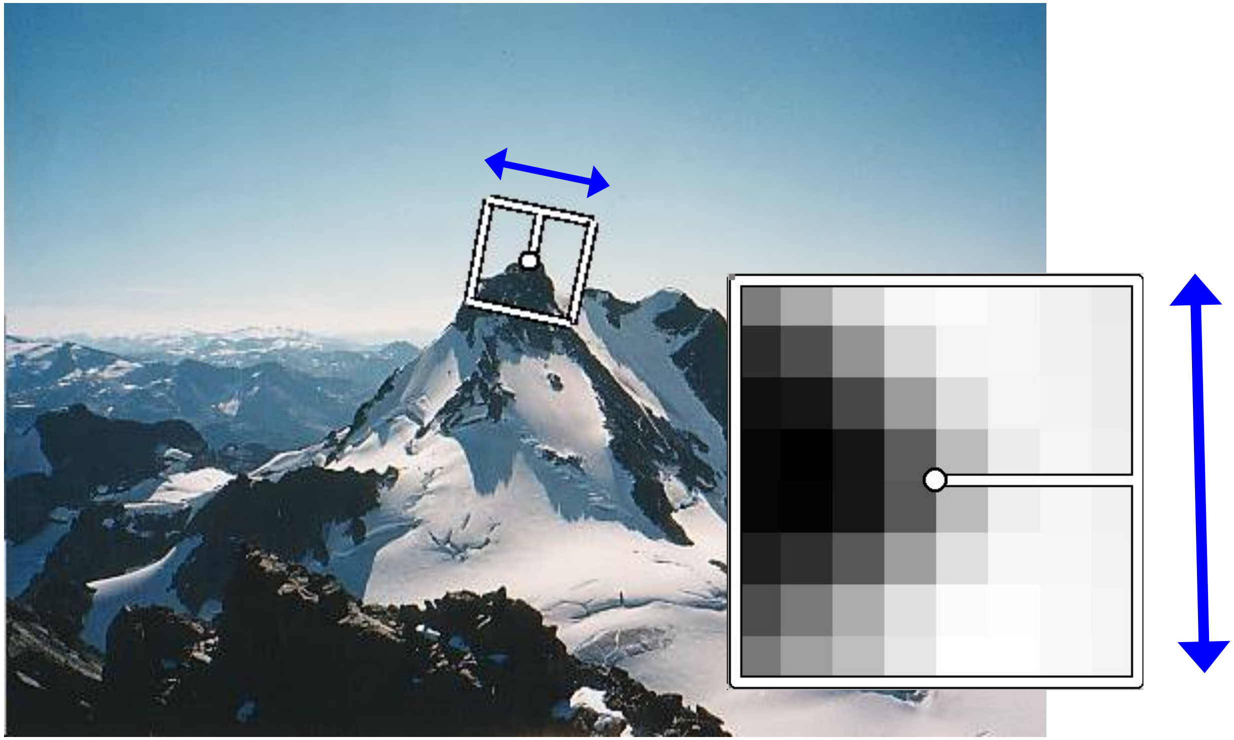
- We can store this as a 1D vector to be efficient:
  156 145 1 155 0 1 116 100 96

Properties of our Descriptor

- Translation Invariant
- Not scale invariant
- Not rotation invariant
- Somewhat invariant to lighting changes

- Let’s look at the SIFT descriptor, because it is heavily used, even without using the SIFT key point detector.
- It already solves the scale problem by computing at multiple scales and keeping track.

Rotation invariance



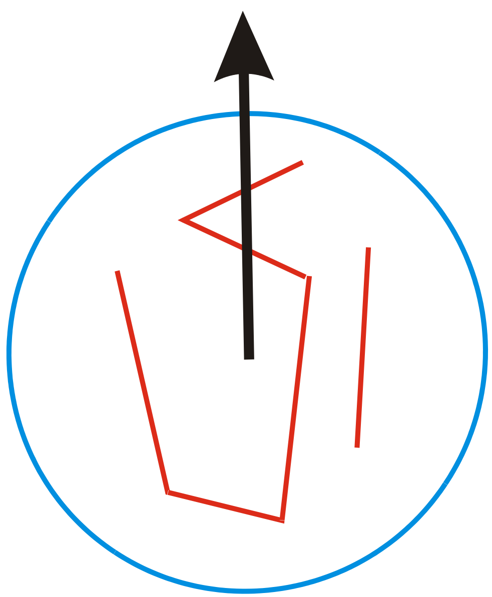
- Rotate patch according to its dominant gradient orientation
- This puts the patches into a canonical orientation.

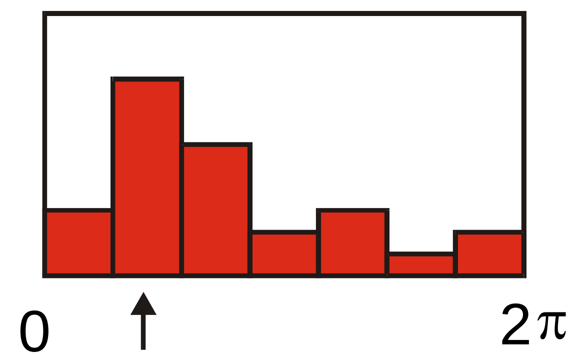
Image from Matthew Brown

Orientation Normalization

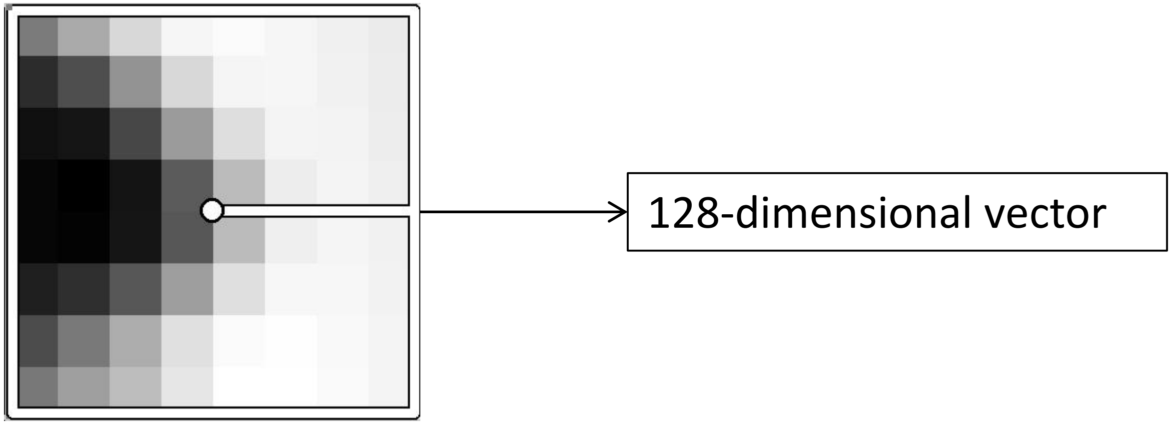
- Compute orientation histogram
- Select dominant orientation
- Normalize: rotate to fixed orientation

[Lowe, SIFT, 1999]





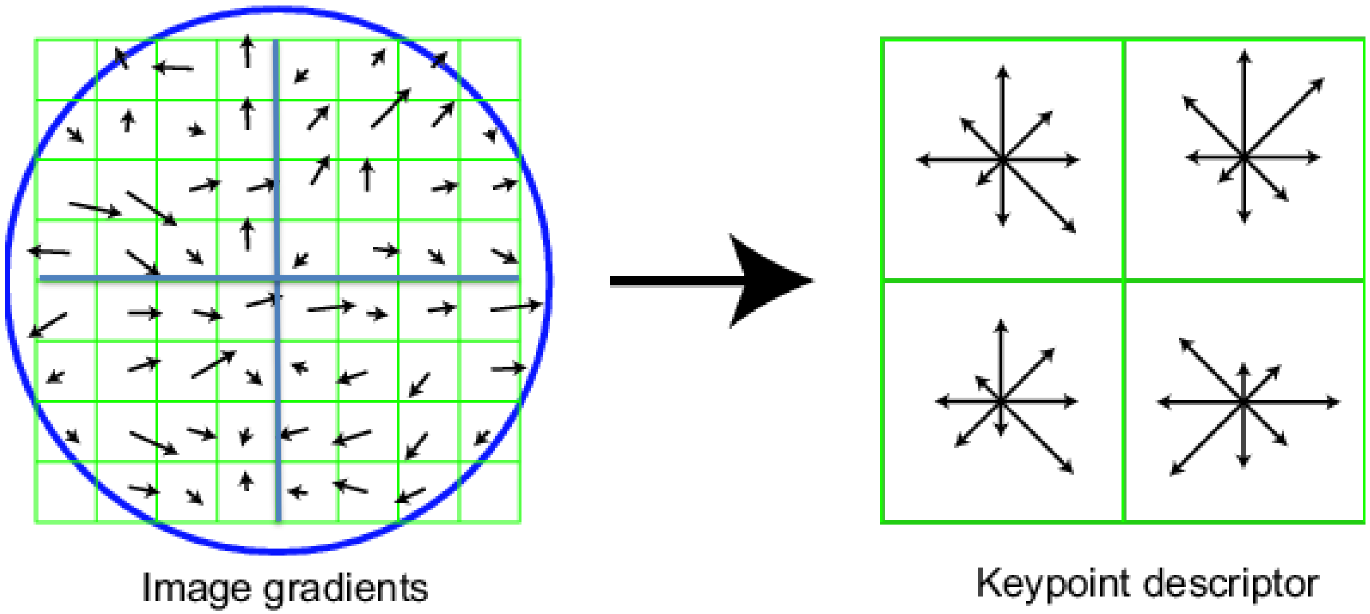
Once we have found the key points and a dominant orientation for each, we need to describe the (rotated and scaled) neighborhood about each.



SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor

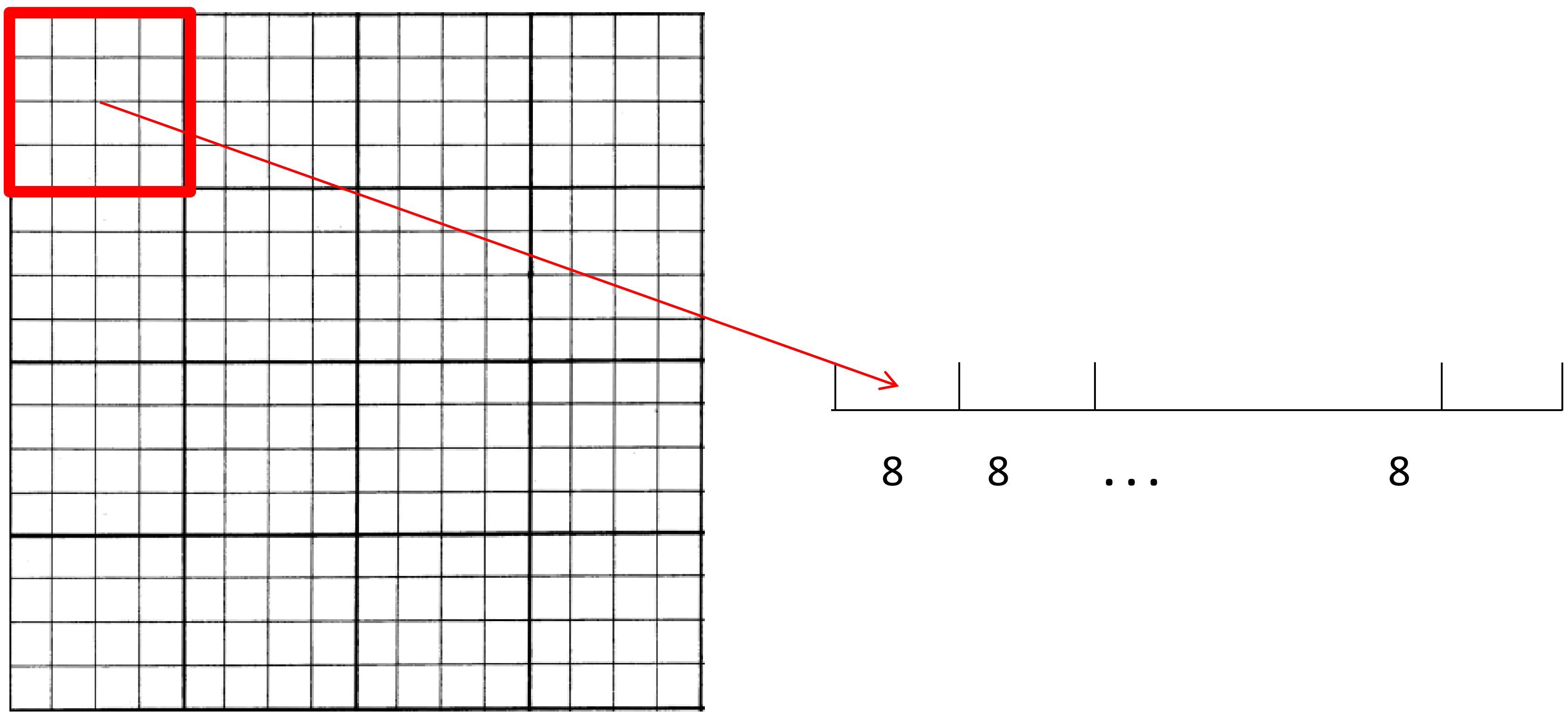


Adapted from slide by David Lowe

SIFT descriptor

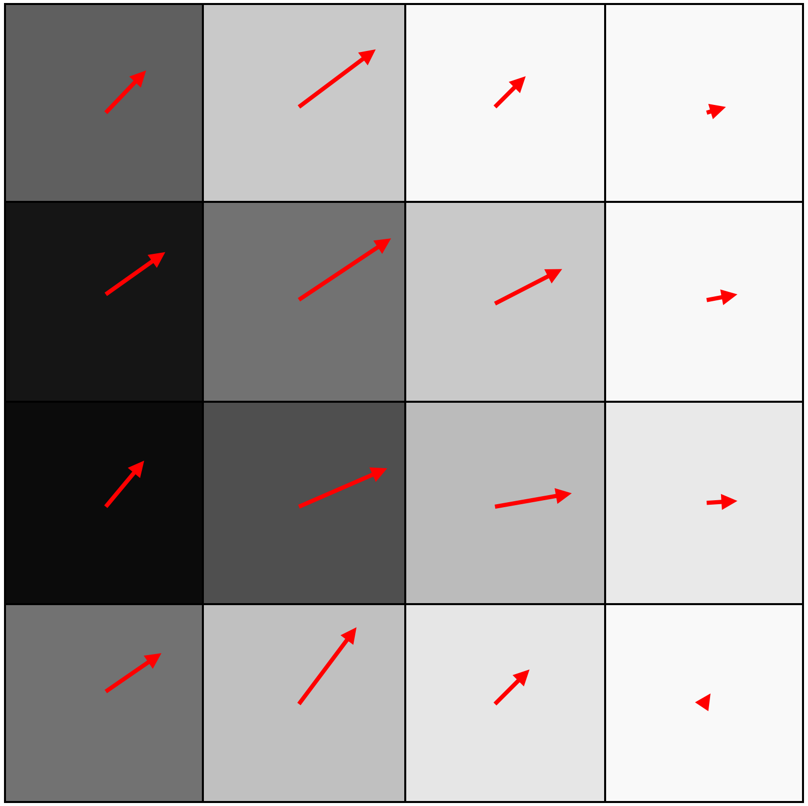
Full version

- Divide the 16x16 window into a 4x4 grid of cells
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor

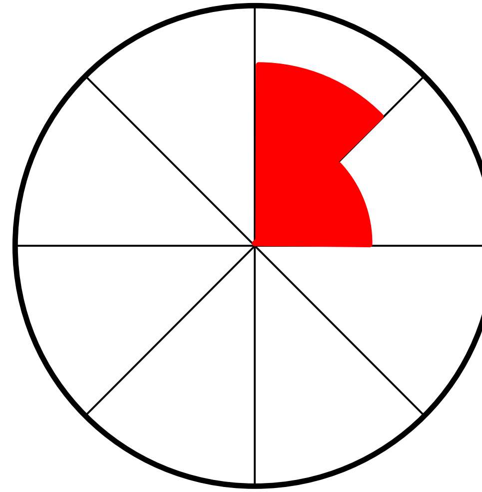


Numeric Example

0.37	0.79	0.97	0.98
0.08	0.45	0.79	0.97
0.04	0.31	0.73	0.91
0.45	0.75	0.90	0.98



Orientations in each of the 16 pixels of the cell



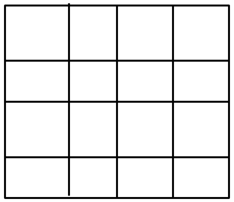
The orientations all ended up in two bins: 11 in one bin, 5 in the other. (rough count)

5 11 0 0 0 0 0 0

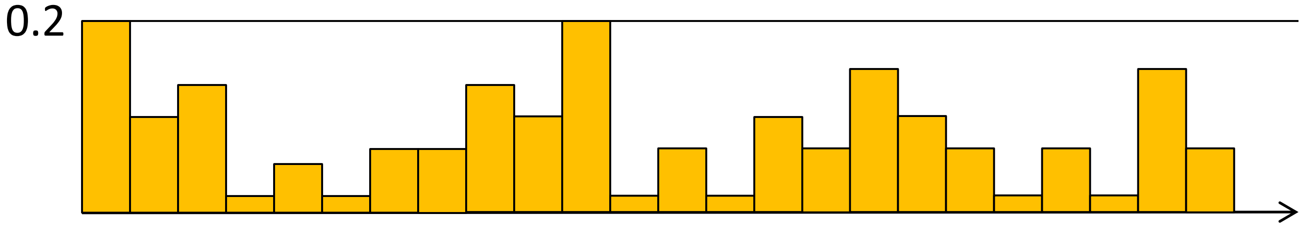
SIFT descriptor

Full version

- Start with a 16x16 window (256 pixels)
- Divide the 16x16 window into a 4x4 grid of cells (16 cells)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor
- Threshold normalize the descriptor:



$$\sum_i d_i^2 = 1 \quad \text{such that:} \quad d_i < 0.2$$

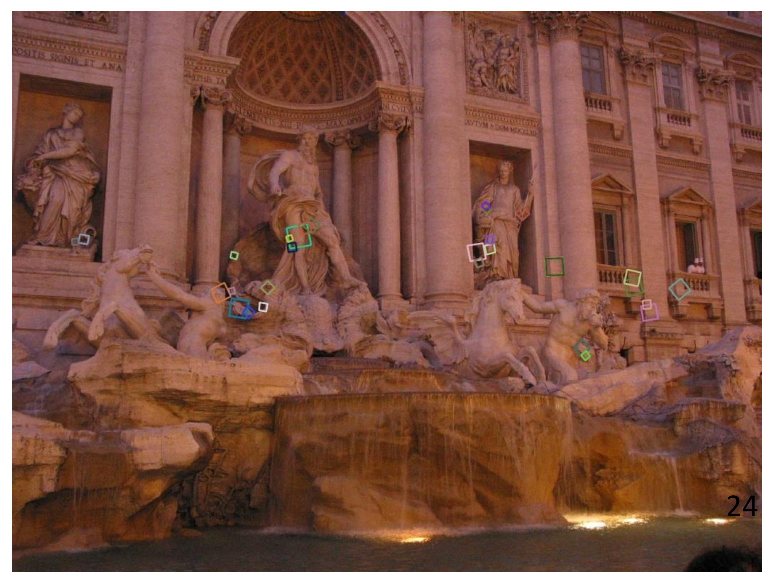


Adapted from slide by David Lowe

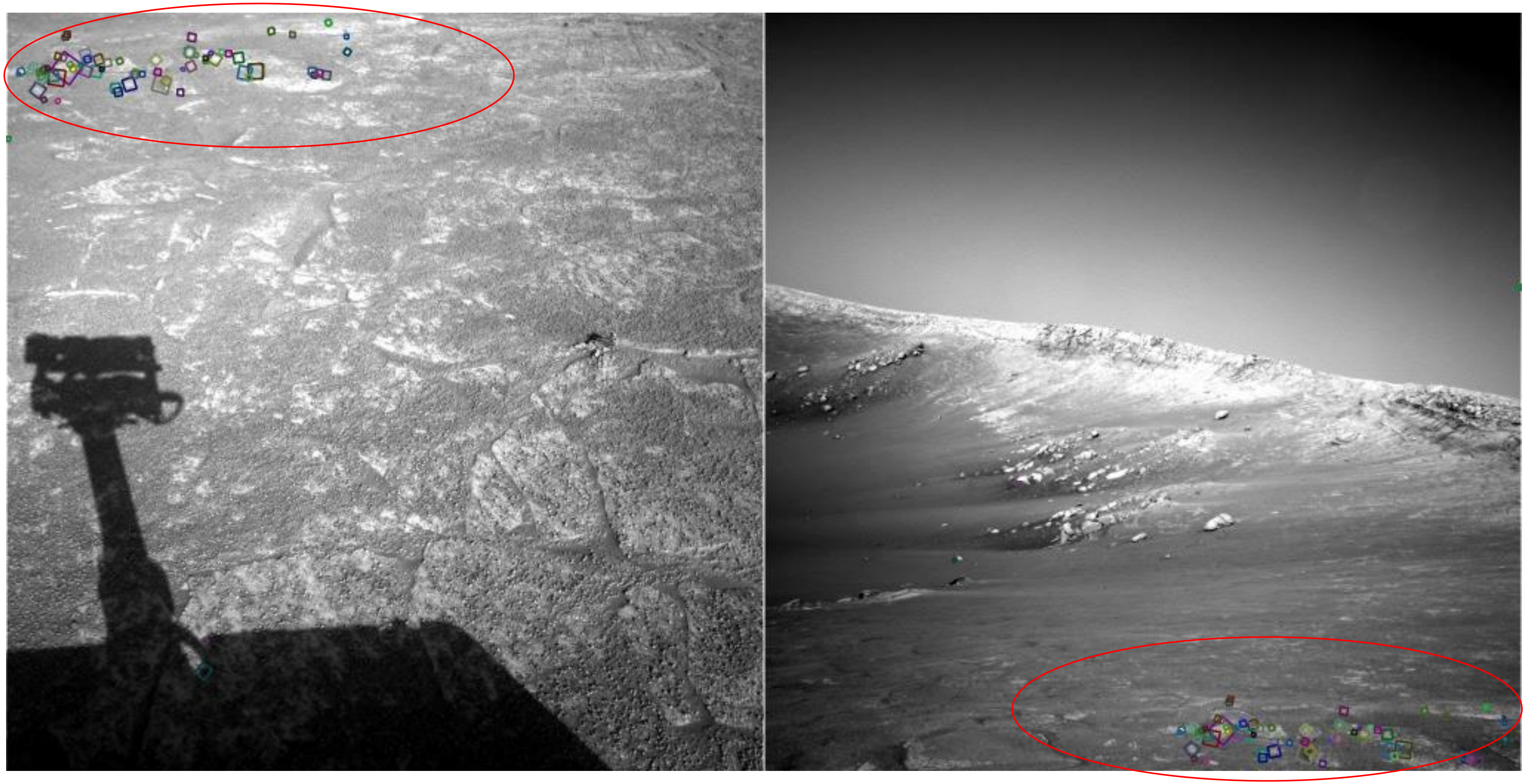
Properties of SIFT

Extraordinarily robust matching technique

- Can handle changes in viewpoint
  - Up to about 30 degree out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Various code available
  - http://www.cs.ubc.ca/~lowe/keypoints/



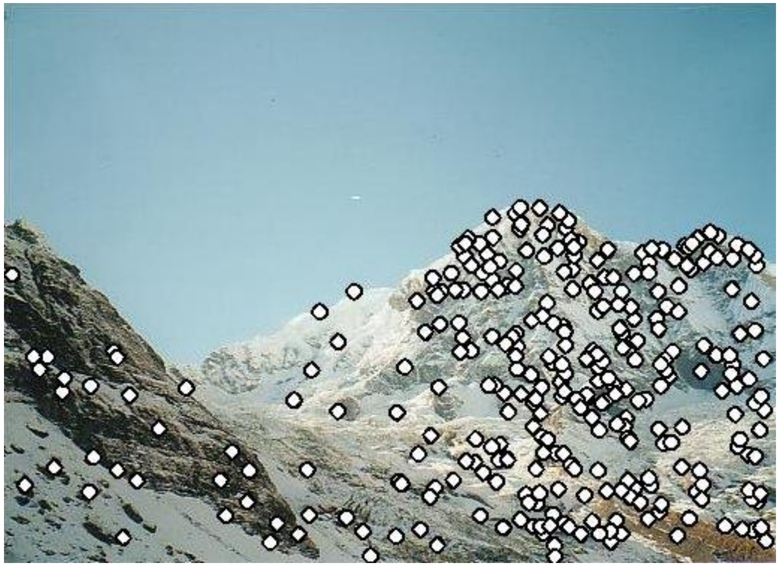
Example

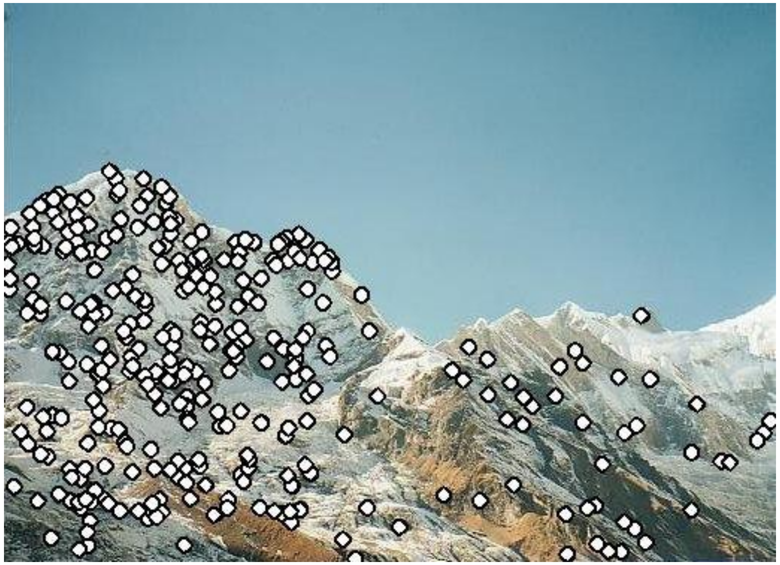


NASA Mars Rover images
with SIFT feature matches
Figure by Noah Snavely

Matching with Features

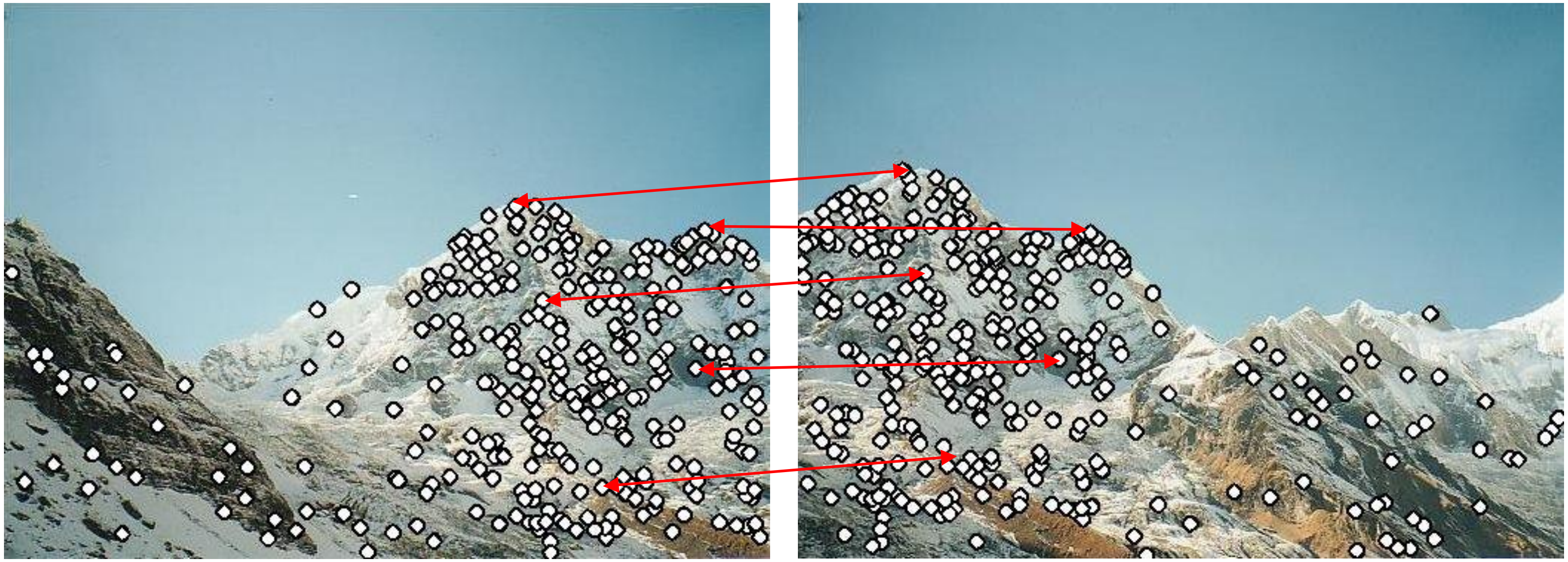
- Detect feature points in both images





Matching with Features

- Detect feature points in both images
- Find corresponding pairs

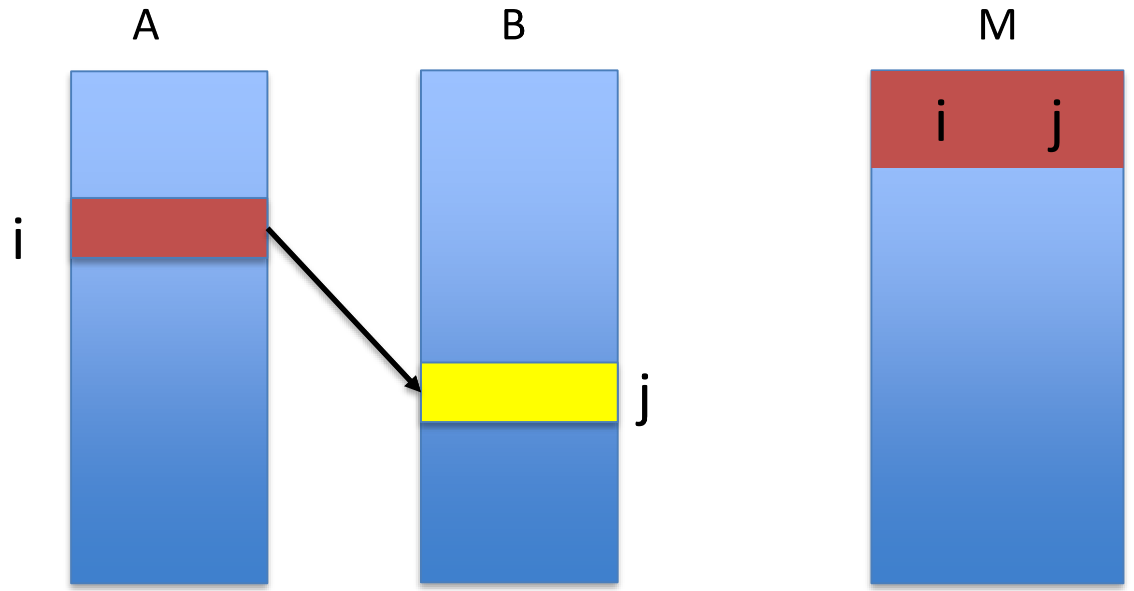


How do you find correspondences?

- Put the descriptors from each image into vectors and match them
- How do you compare two descriptors?
- Vector distance
  - L1-Distance(A,B) = $\boldsymbol{\Sigma}_i \ |a_i - b_i|$
  - L2-Distance(A,B) = $\boldsymbol{\Sigma}_i$ sqrt($a_i^2 - b_i^2$)
  - cosine distance (angle between 2 vectors)

Find the best matches

- For each descriptor a in A, find its best match b in B



- And store it in a vector of matches
- Note: this is abstract; see code for details.

Matching with Features

- Detect feature points in both images
- Find corresponding pairs
- Use these matching pairs to align images.

- Larger Goal: Combine two or more overlapping images to make one larger image

Slide credit: [Vaibhav Vaish]

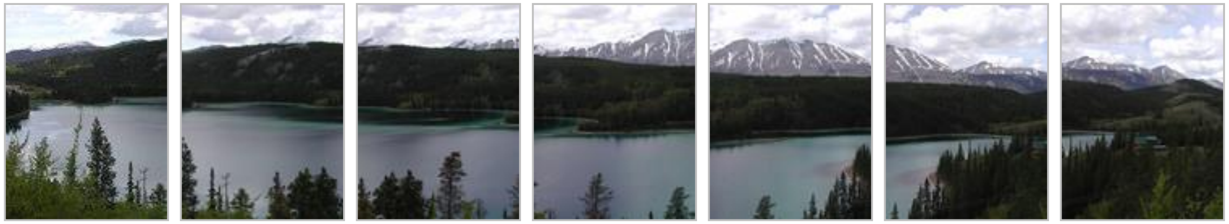
How to do it?

- Basic Procedure
  1. Take a sequence of images from the same position

     (Rotate the camera about its optical center)
  2. Compute transformation between second image and first
  3. Shift the second image to overlap with the first
  4. Blend the two together to create a mosaic
  5. If there are more images, repeat

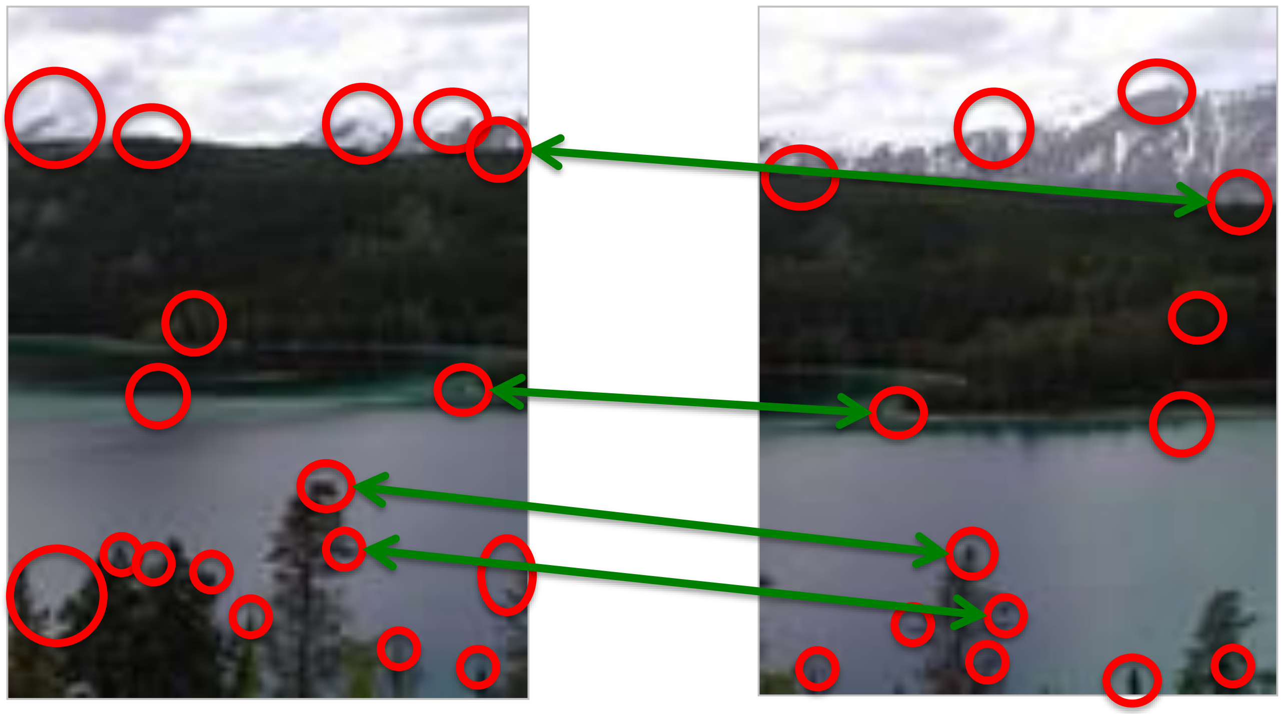
1. Take a sequence of images from the same position

- Rotate the camera about its optical center



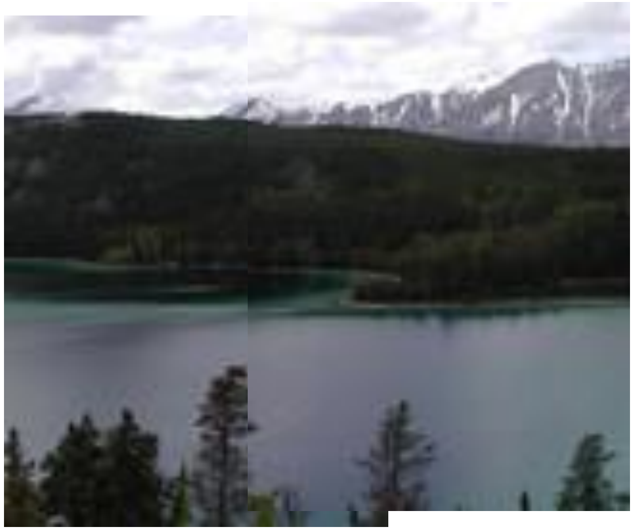
2. Compute transformation between images

- Extract interest points
- Find Matches
- Compute transformation ?

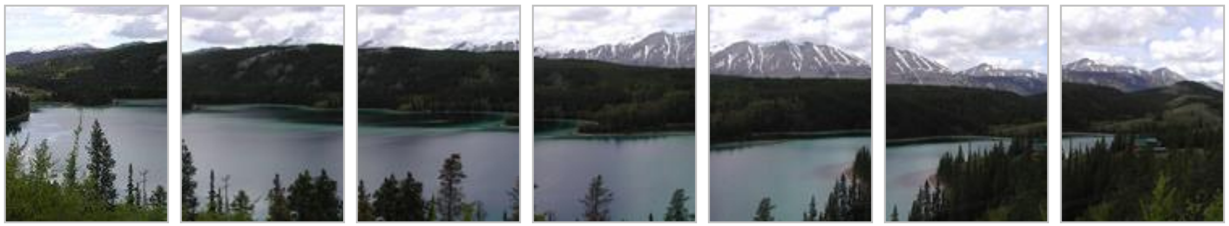


3. Shift the images to overlap

4. Blend the two together to create a mosaic



5. Repeat for all images



How to do it?

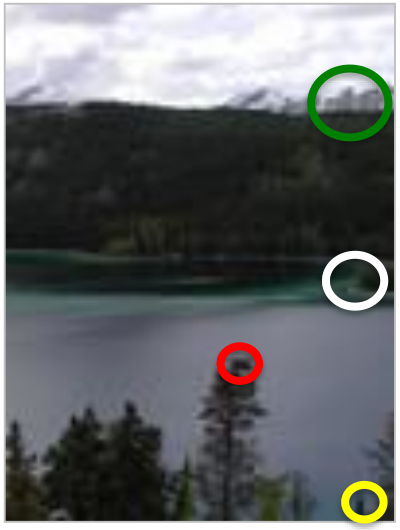
- Basic Procedure
  1. Take a sequence of images from the same position

     Rotate the camera about its optical center
  2. Compute transformation between second image and first
  3. Shift the second image to overlap with the first
  4. Blend the two together to create a mosaic
  5. If there are more images, repeat

Compute Transformations

✓ • Extract interest points
✓ • Find good matches
- Compute transformation

Let’s assume we are given a set of good matching interest points



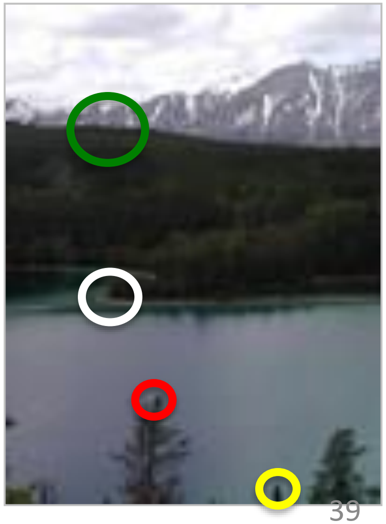
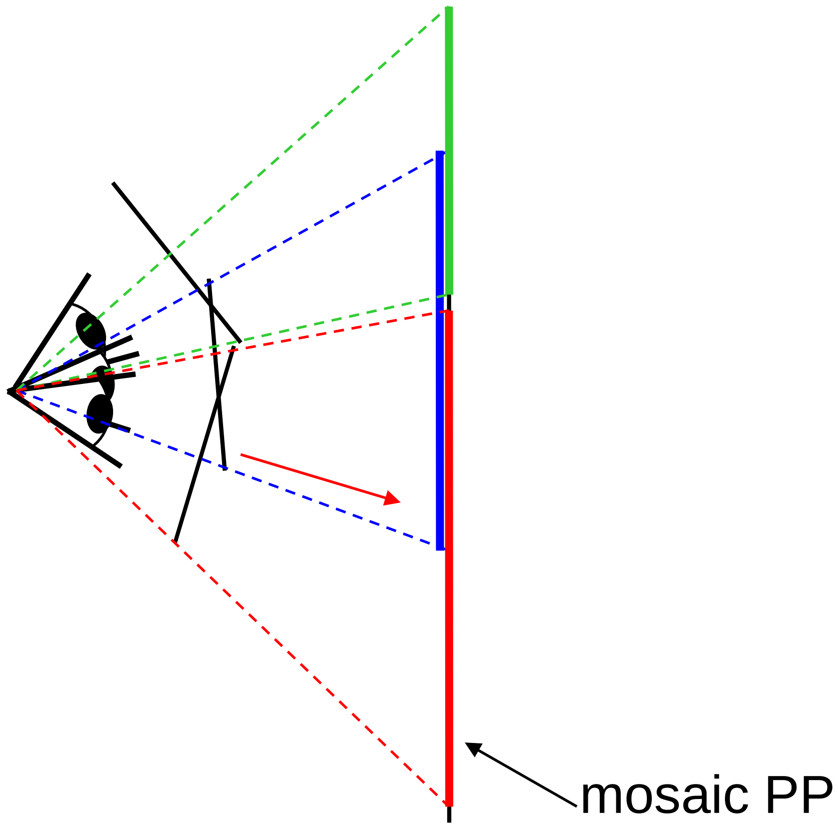
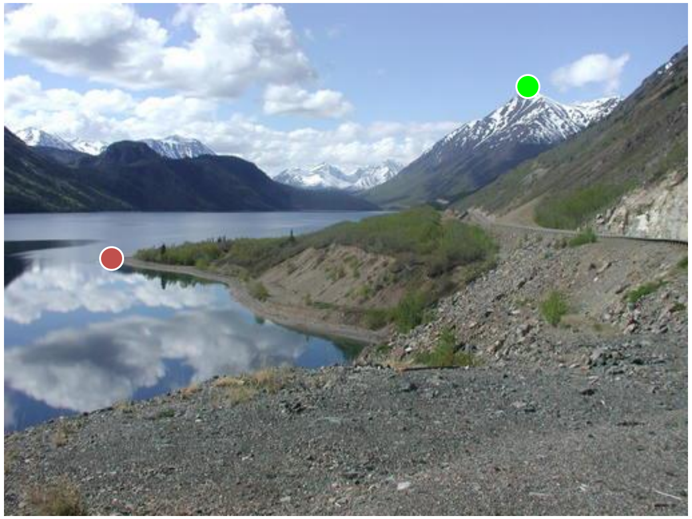


Image reprojection



- The mosaic has a natural interpretation in 3D
  - The images are reprojected onto a common plane
  - The mosaic is formed on this plane

Example



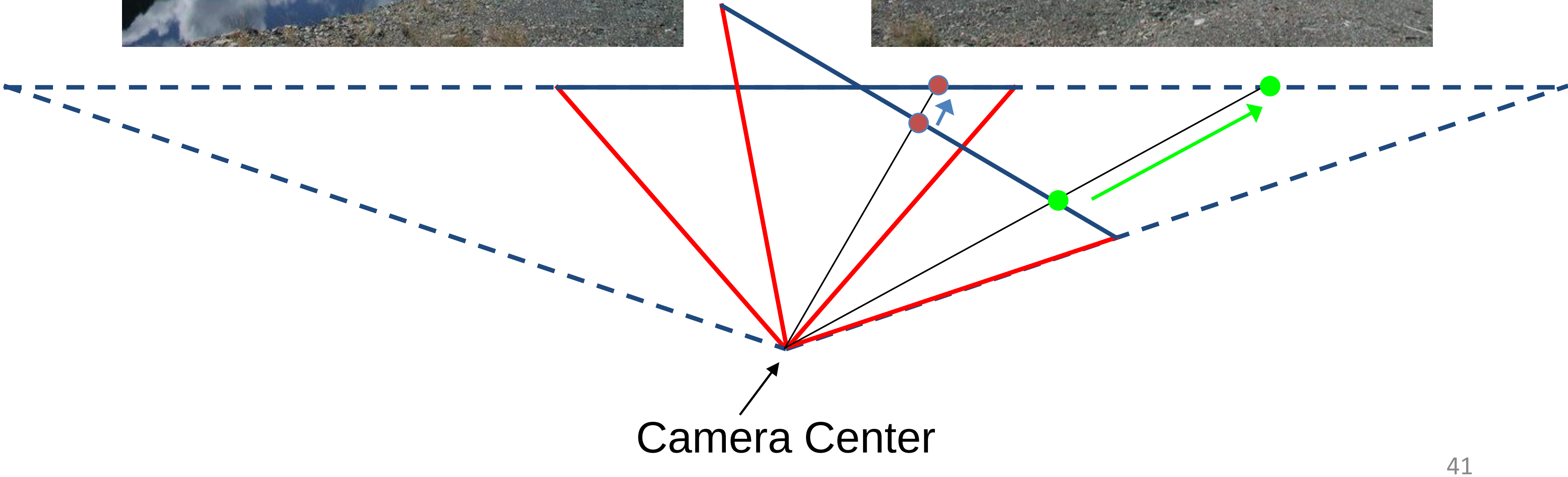
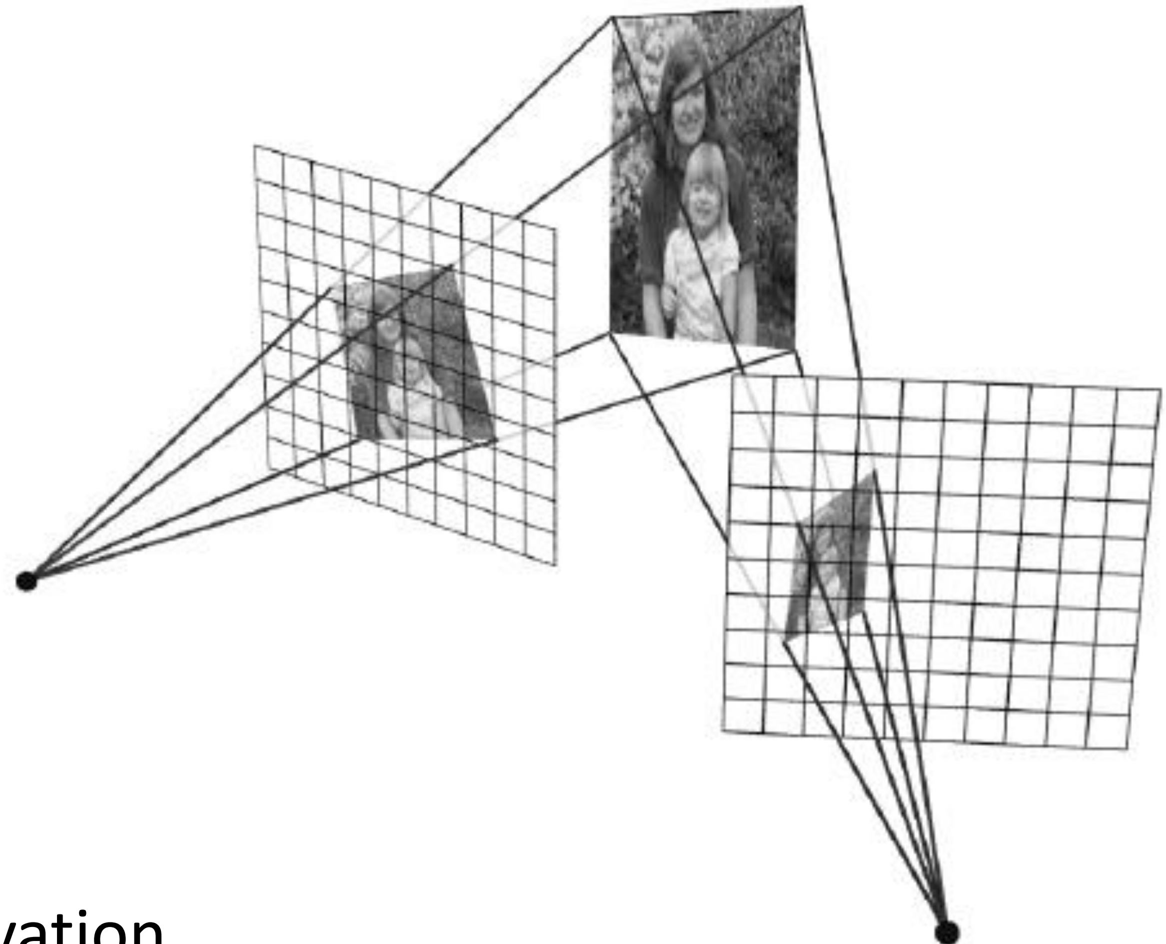


Image reprojection



- Observation
  - Rather than thinking of this as a 3D reprojection, think of it as a 2D image warp from one image to another

Motion models

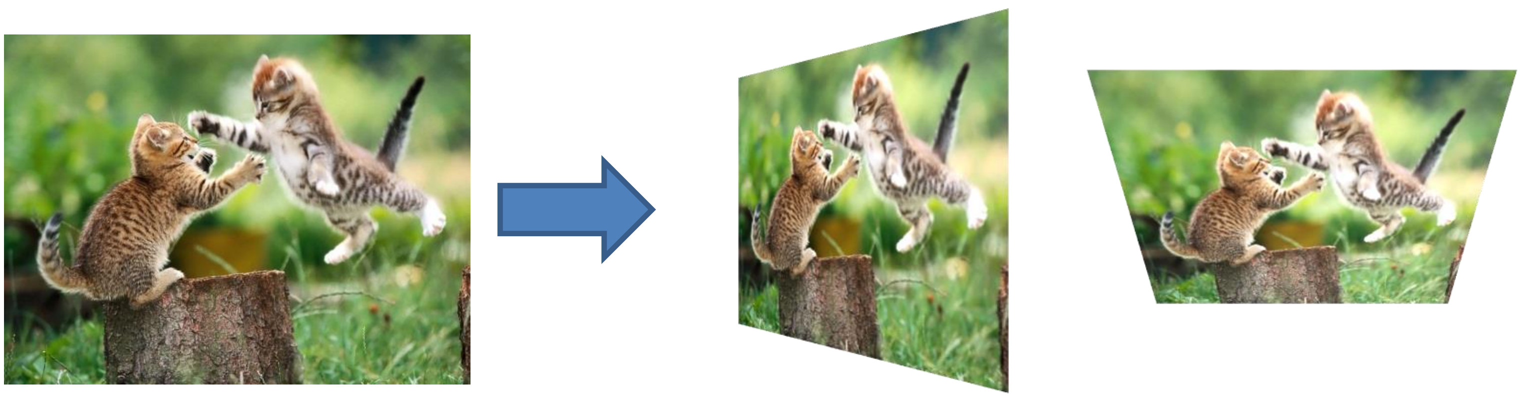
- What happens when we take two images with a camera and try to align them?
- translation?
- rotation?
- scale?
- affine?
- Perspective?

Projective transformations

- (aka *homographies*)

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

$$x' = u/w$$
$$y' = v/w$$



Parametric (global) warping

- Examples of parametric warps:

translation

rotation

aspect

affine

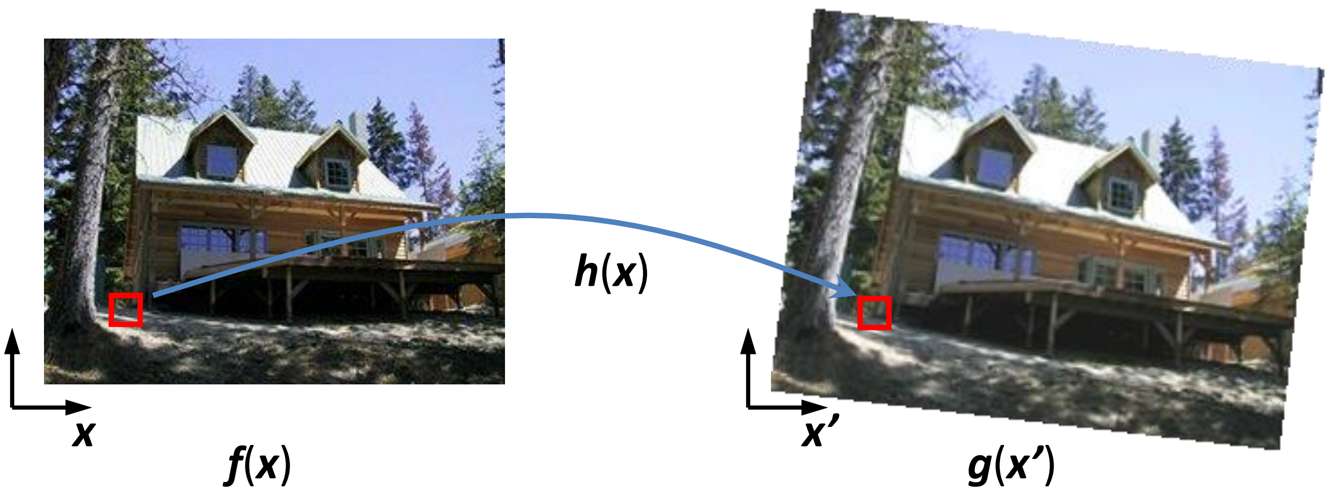
perspective

2D coordinate transformations

- translation: $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ $\qquad$ $\boldsymbol{x} = (x,y)$
- rotation: $\boldsymbol{x}' = \boldsymbol{R}\,\boldsymbol{x} + \boldsymbol{t}$
- similarity: $\boldsymbol{x}' = s\,\boldsymbol{R}\,\boldsymbol{x} + \boldsymbol{t}$
- affine: $\boldsymbol{x}' = \boldsymbol{A}\,\boldsymbol{x} + \boldsymbol{t}$
- perspective: $\underline{\boldsymbol{x}}' \cong \boldsymbol{H}\,\underline{\boldsymbol{x}}$ $\qquad$ $\underline{\boldsymbol{x}} = (x,y,1)$
  ($\underline{\boldsymbol{x}}$ is a *homogeneous* coordinate)

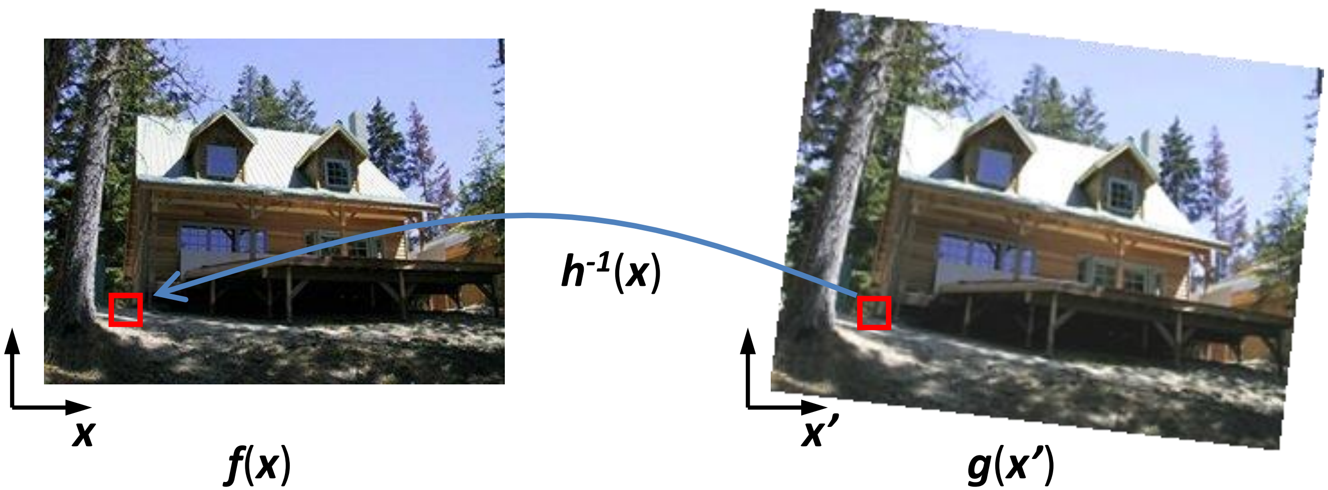
Image Warping

- Given a coordinate transform $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$ and a source image $\boldsymbol{f}(\boldsymbol{x})$, how do we compute a transformed image $\boldsymbol{g}(\boldsymbol{x}') = \boldsymbol{f}(\boldsymbol{h}(\boldsymbol{x}))$?



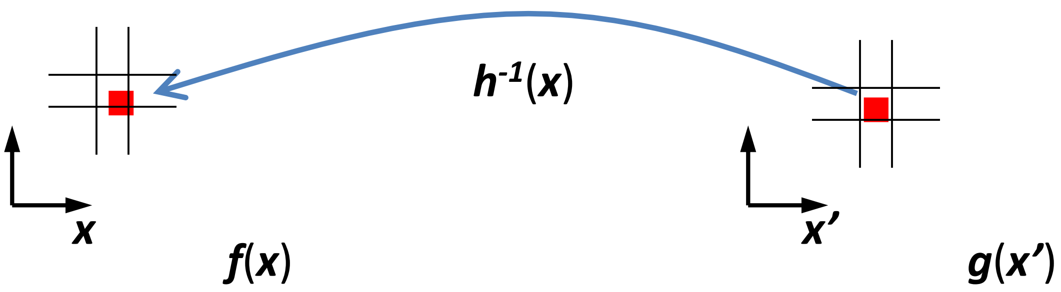
Inverse Warping

- Get each pixel $\boldsymbol{g}(\boldsymbol{x'})$ from its corresponding location $\boldsymbol{x'} = \boldsymbol{h}(\boldsymbol{x})$ in $\boldsymbol{f}(\boldsymbol{x})$
  - What if pixel comes from "between" two pixels?



Inverse Warping

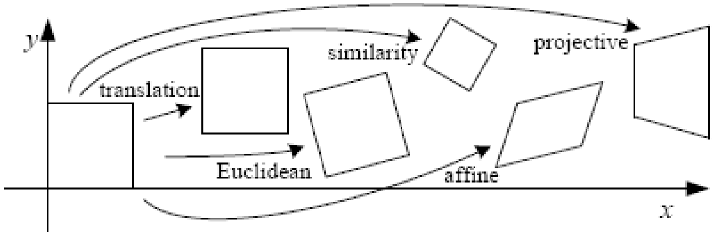
- Get each pixel ***g***(***x'***) from its corresponding location ***x'*** = ***h***(***x***) in ***f***(***x***)
  - What if pixel comes from "between" two pixels?
  - Answer: We already know this: *resample* color value from *interpolated* source image



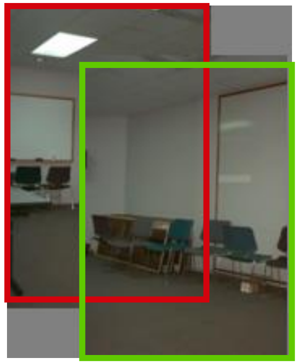
Interpolation

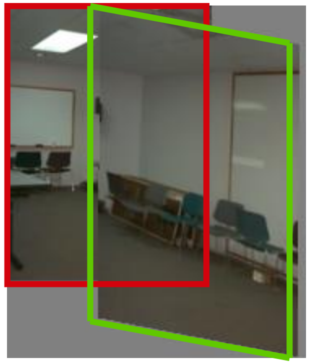
- Possible interpolation filters:
  - nearest neighbor
  - bilinear
  - bicubic (interpolating)

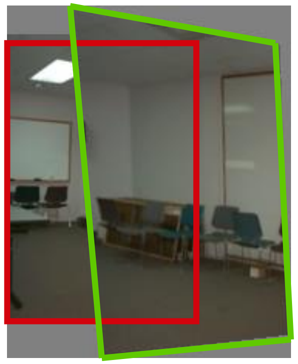
Motion models



Translation **Affine** **Perspective**







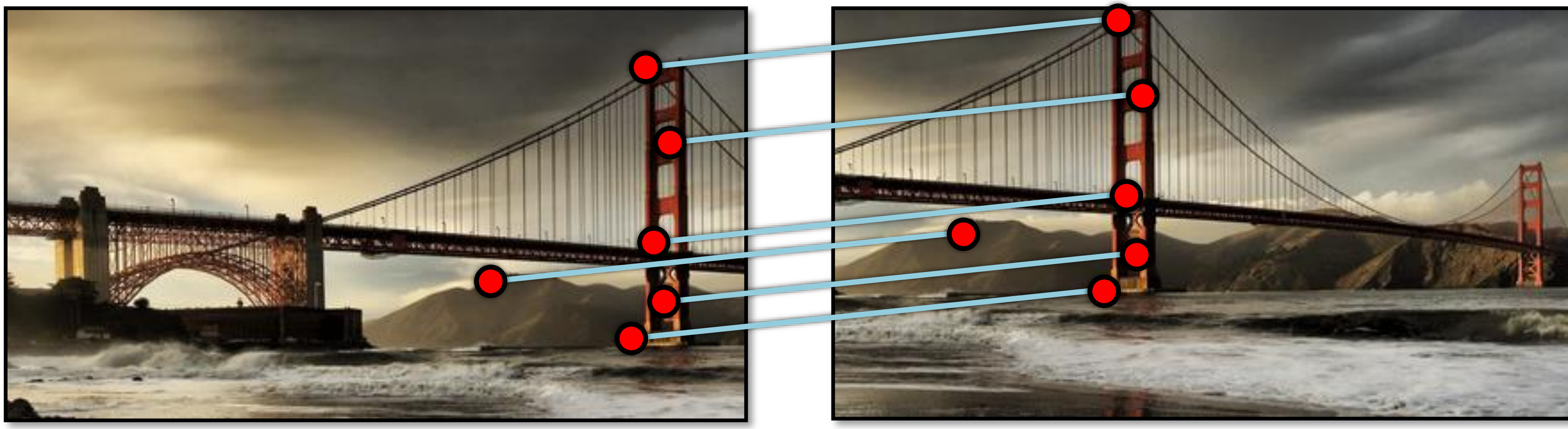
2 unknowns **6 unknowns** **8 unknowns**

Finding the transformation

- Translation = 2 degrees of freedom
- Similarity = 4 degrees of freedom
- Affine = 6 degrees of freedom
- Homography = 8 degrees of freedom

- How many corresponding points do we need to solve?

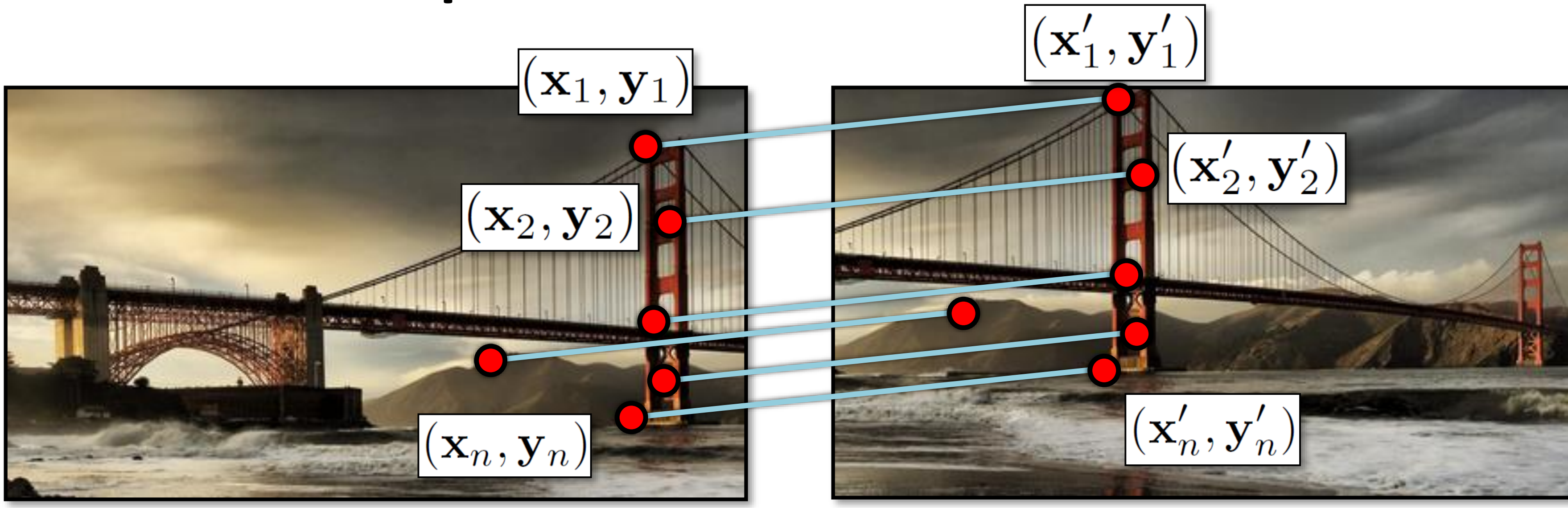
Simple case: translations



$(\mathbf{x}_t, \mathbf{y}_t)$

How do we solve for $(\mathbf{x}_t, \mathbf{y}_t)$?

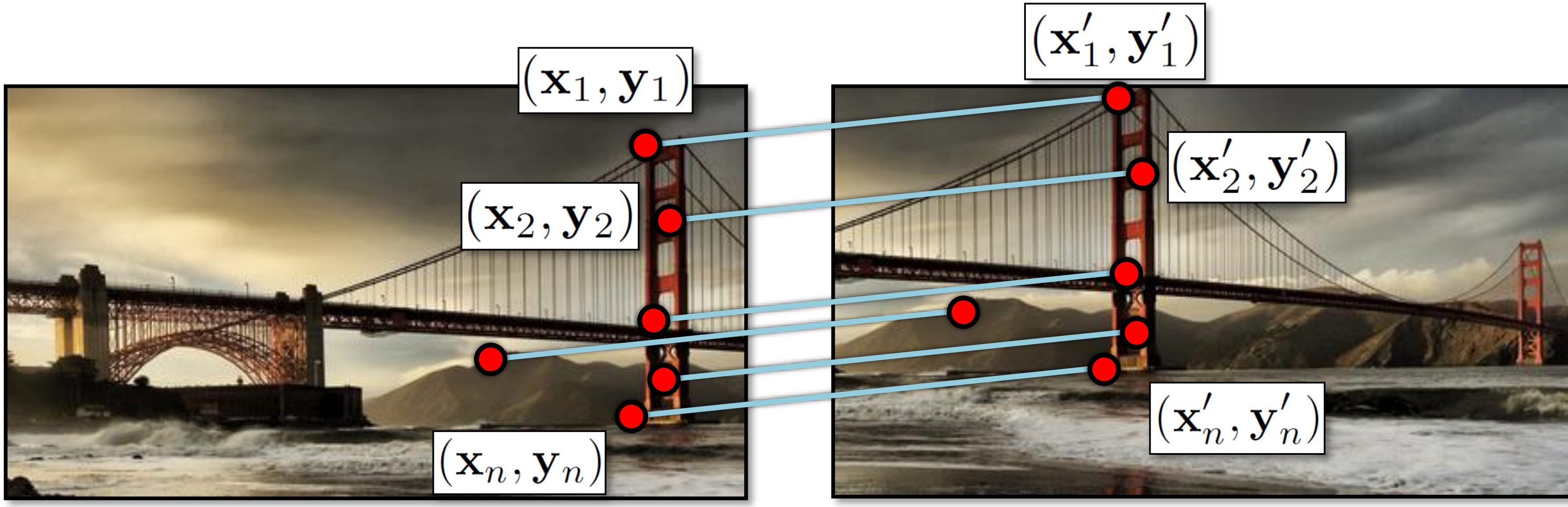
Simple case: translations



Displacement of match *i* = $(\mathbf{x}'_i - \mathbf{x}_i, \mathbf{y}'_i - \mathbf{y}_i)$

$$(\mathbf{x}_t, \mathbf{y}_t) = \left(\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}'_i - \mathbf{x}_i, \frac{1}{n} \sum_{i=1}^{n} \mathbf{y}'_i - \mathbf{y}_i \right)$$

Simple case: translations

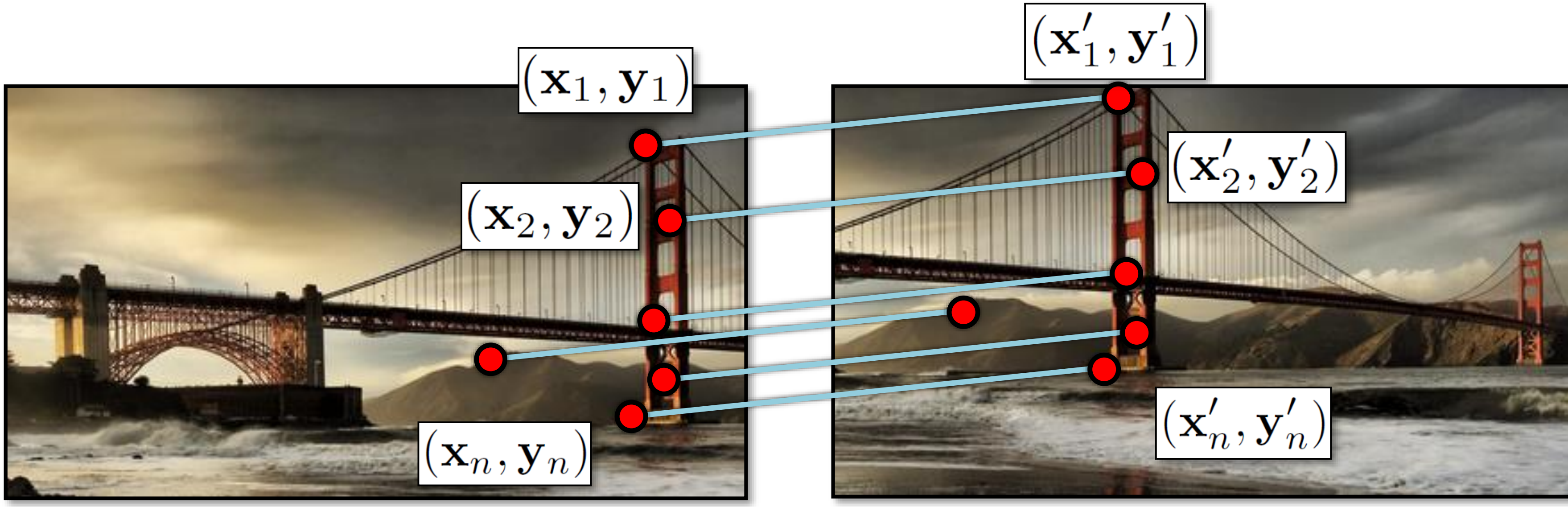


$$\mathbf{x}_i + \mathbf{x_t} = \mathbf{x}'_i$$

$$\mathbf{y}_i + \mathbf{y_t} = \mathbf{y}'_i$$

- System of linear equations
  - What are the knowns? Unknowns?
  - How many unknowns? How many equations (per match)?

Simple case: translations



$$\begin{aligned} \mathbf{x}_i + \mathbf{x_t} &= \mathbf{x}'_i \\ \mathbf{y}_i + \mathbf{y_t} &= \mathbf{y}'_i \end{aligned}$$

- Problem: more equations than unknowns
  - "Overdetermined" system of equations
  - We will find the *least squares* solution

Least squares formulation

- For each point $(\mathbf{x}_i, \mathbf{y}_i)$

$$\begin{aligned} \mathbf{x}_i + \mathbf{x_t} &= \mathbf{x}_i' \\ \mathbf{y}_i + \mathbf{y_t} &= \mathbf{y}_i' \end{aligned}$$

- we define the *residuals* as

$$\begin{aligned} r_{\mathbf{x}_i}(\mathbf{x}_t) &= (\mathbf{x}_i + \mathbf{x}_t) - \mathbf{x}_i' \\ r_{\mathbf{y}_i}(\mathbf{y}_t) &= (\mathbf{y}_i + \mathbf{y}_t) - \mathbf{y}_i' \end{aligned}$$

Least squares formulation

- Goal: minimize sum of squared residuals

$$C(\mathbf{x}_t, \mathbf{y}_t) = \sum_{i=1}^{n} \left(r_{\mathbf{x}_i}(\mathbf{x}_t)^2 + r_{\mathbf{y}_i}(\mathbf{y}_t)^2 \right)$$

- "Least squares" solution
- For translations, is equal to mean displacement

Solving for translations

- Using least squares

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ & \vdots \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x'_1 - x_1 \\ y'_1 - y_1 \\ x'_2 - x_2 \\ y'_2 - y_2 \\ \vdots \\ x'_n - x_n \\ y'_n - y_n \end{bmatrix}$$

$$\underset{2n \times 2}{\mathbf{A}} \quad \underset{2 \times 1}{\mathbf{t}} = \underset{2n \times 1}{\mathbf{b}}$$

Least squares

$$\mathbf{A}\mathbf{t} = \mathbf{b}$$

- Find **t** that minimizes

$$||\mathbf{A}\mathbf{t} - \mathbf{b}||^2$$

- To solve, form the *normal equations*

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{t} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$$

$$\mathbf{t} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$$

Affine transformations

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- How many unknowns?
- How many equations per match?
- $x' = ax + by + c$; $y' = dx + ey + f$
- How many matches do we need?

Affine transformations

- Residuals:

$$\begin{aligned} r_{x_i}(a,b,c,d,e,f) &= (ax_i + by_i + c) - x'_i \\ r_{y_i}(a,b,c,d,e,f) &= (dx_i + ey_i + f) - y'_i \end{aligned}$$

- Cost function:

$$C(a,b,c,d,e,f) = \sum_{i=1}^{n} \left(r_{x_i}(a,b,c,d,e,f)^2 + r_{y_i}(a,b,c,d,e,f)^2 \right)$$

Affine transformations

- Matrix form

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ & & & \vdots & & \\ x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix}}_{\substack{\mathbf{A} \\ 2n \text{ x } 6}} \underbrace{\begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}}_{\substack{\mathbf{t} \\ 6 \text{ x } 1}} = \underbrace{\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix}}_{\substack{\mathbf{b} \\ 2n \text{ x } 1}}$$

Solving for homographies

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Why is this now a variable and not just 1?

- A homography is a projective object, in that it has no scale. It is represented by the above matrix, up to scale.
- One way of fixing the scale is to set one of the coordinates to 1, though that choice is arbitrary.
- But that's what most people do and your assignment code does.

Solving for homographies

$$\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$x'_i = \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

$$y'_i = \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}}$$

Why the division?

$$x'_i(h_{20}x_i + h_{21}y_i + h_{22}) = h_{00}x_i + h_{01}y_i + h_{02}$$
$$y'_i(h_{20}x_i + h_{21}y_i + h_{22}) = h_{10}x_i + h_{11}y_i + h_{12}$$

Solving for homographies

$$
\begin{aligned}
x'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\
y'_i(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12}
\end{aligned}
$$

$$
\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 & -x'_i x_i & -x'_i y_i & -x'_i \\ 0 & 0 & 0 & x_i & y_i & 1 & -y'_i x_i & -y'_i y_i & -y'_i \end{bmatrix}
\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}
= \begin{bmatrix} 0 \\ 0 \end{bmatrix}
$$

This is just for one pair of points.

Direct Linear Transforms (n points)

$$\underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n \end{bmatrix}}_{\substack{\mathbf{A} \\ 2n \times 9}} \underbrace{\begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix}}_{\substack{\mathbf{h} \\ 9}} = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}}_{\substack{\mathbf{0} \\ 2n}}$$

Defines a least squares problem: minimize $\|\mathbf{A}\mathbf{h} - \mathbf{0}\|^2$

- Since $\mathbf{h}$ is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
- Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue
- Works with 4 or more points

Direct Linear Transforms

- Why could we not solve for the homography in exactly the same way we did for the affine transform, ie.

$$\mathbf{t} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}$$

Answer from Sameer Agarwal (Dr. Rome in a Day)

- For an affine transform, we have equations of the form $Ax_i + b = y_i$, solvable by linear regression.
- For the homography, the equation is of the form

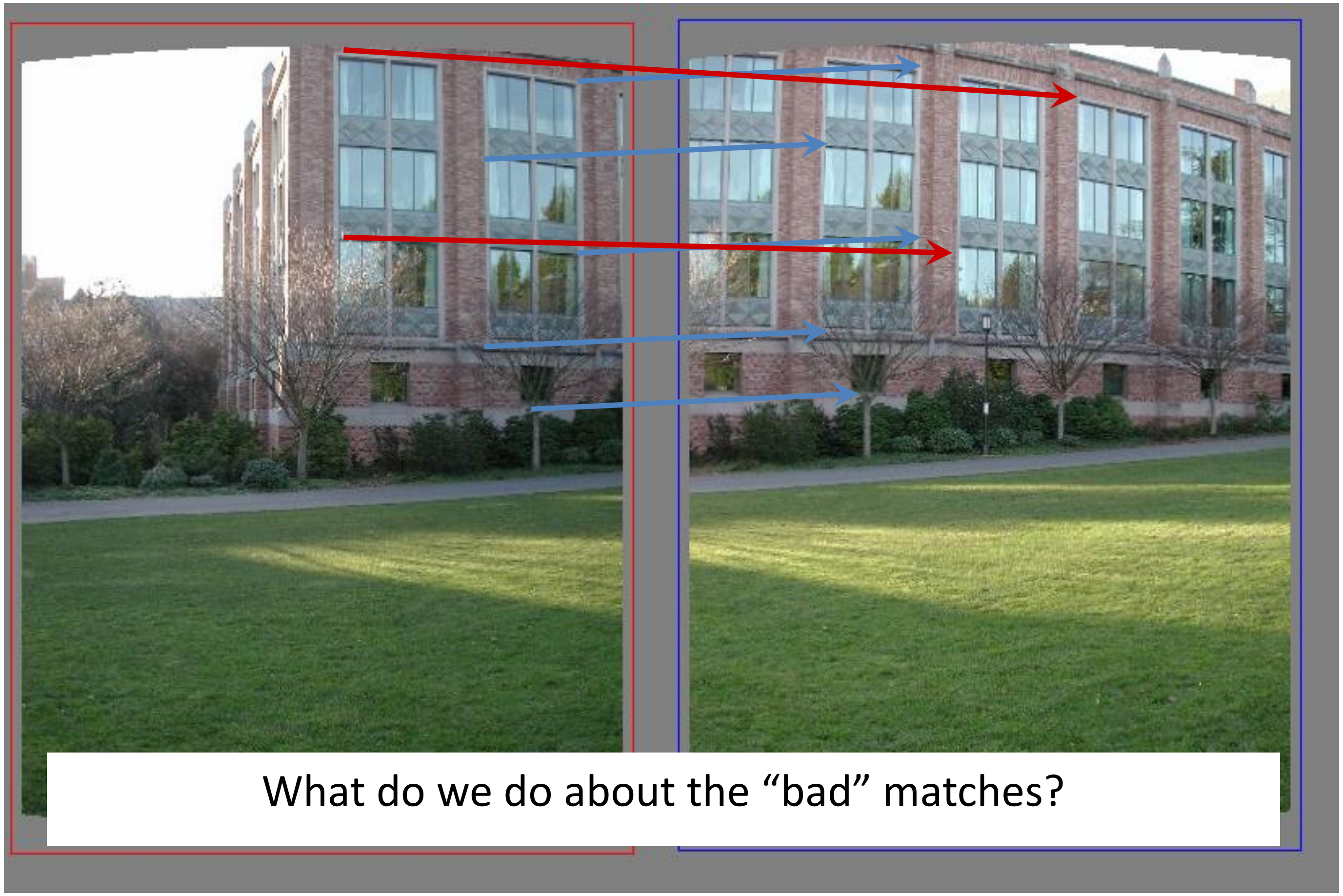
  $H\tilde{x}_i \sim \tilde{y}_i$ (homogeneous coordinates)

and the ~ means it holds only up to scale. The affine solution does not hold.

Colosseum: 2,097 images, 819,242 points

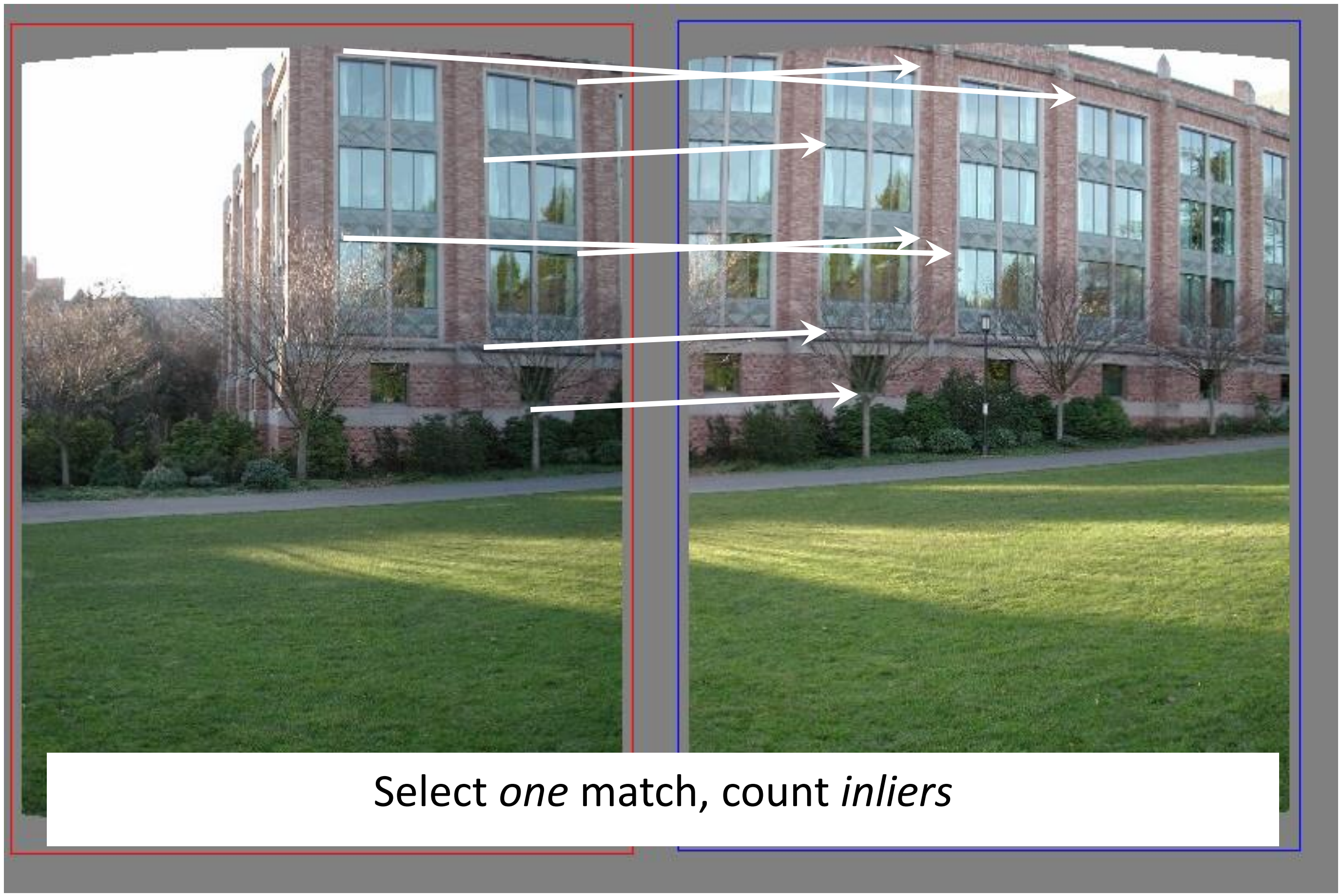
Trevi Fountain: 1,935 images, 1,055,153 points

Matching features



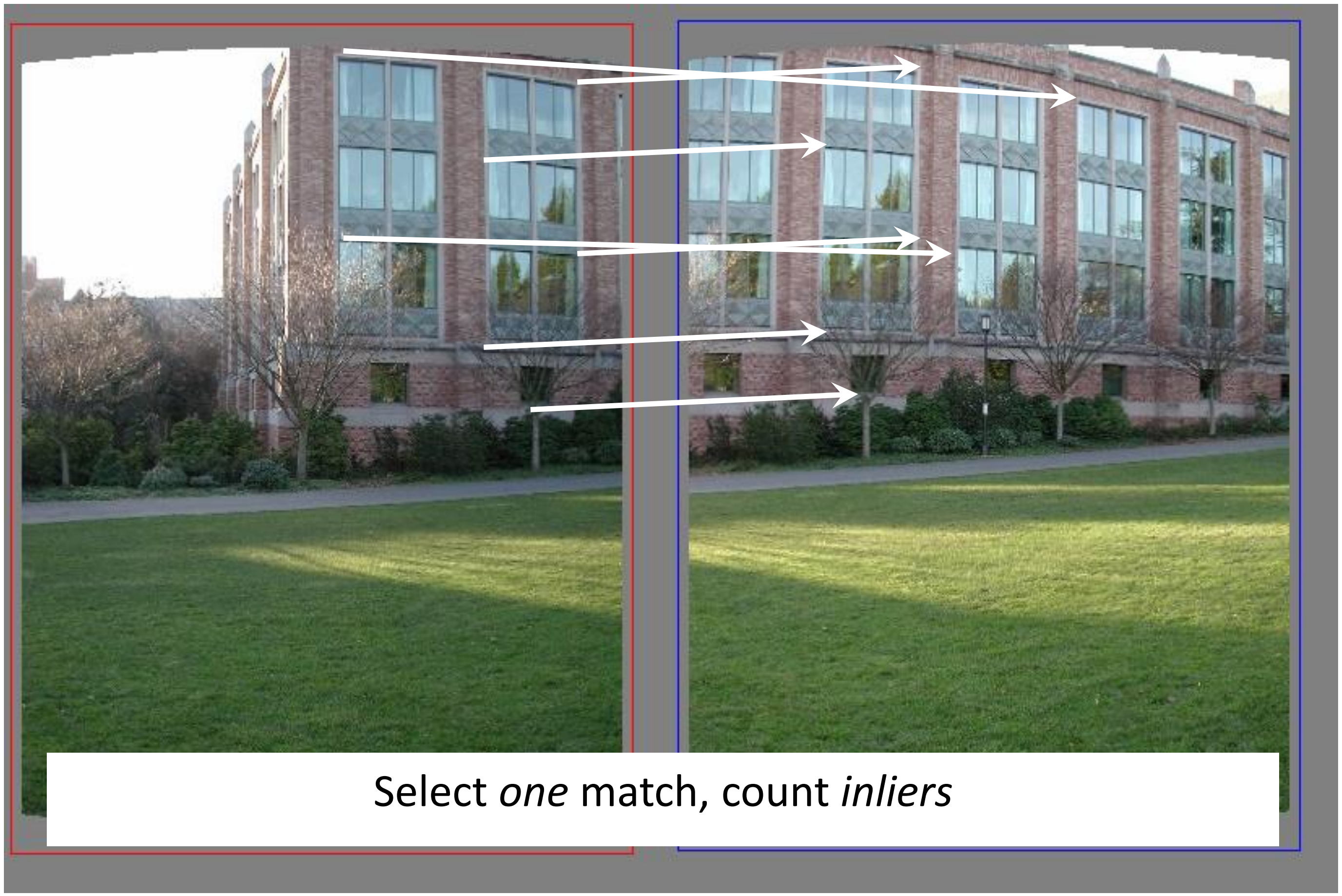
What do we do about the “bad” matches?

RAndom SAmple Consensus



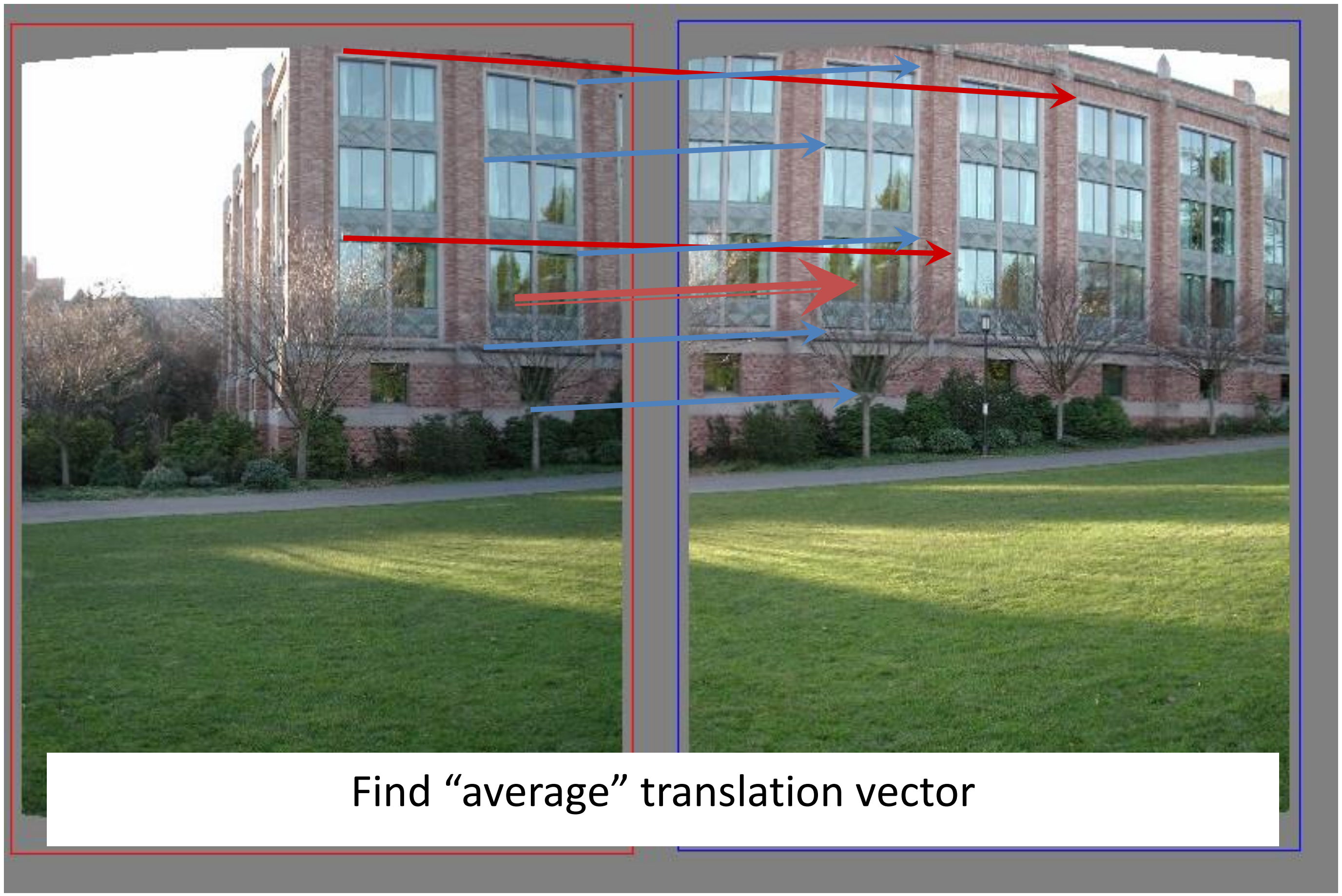
Select *one* match, count *inliers*

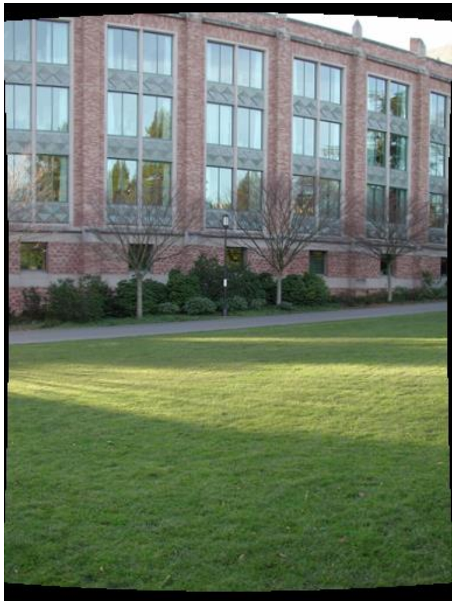
<u>RA</u>ndom <u>SA</u>mple <u>C</u>onsensus



Select *one* match, count *inliers*

Least squares fit (from inliers)





RANSAC for estimating homography

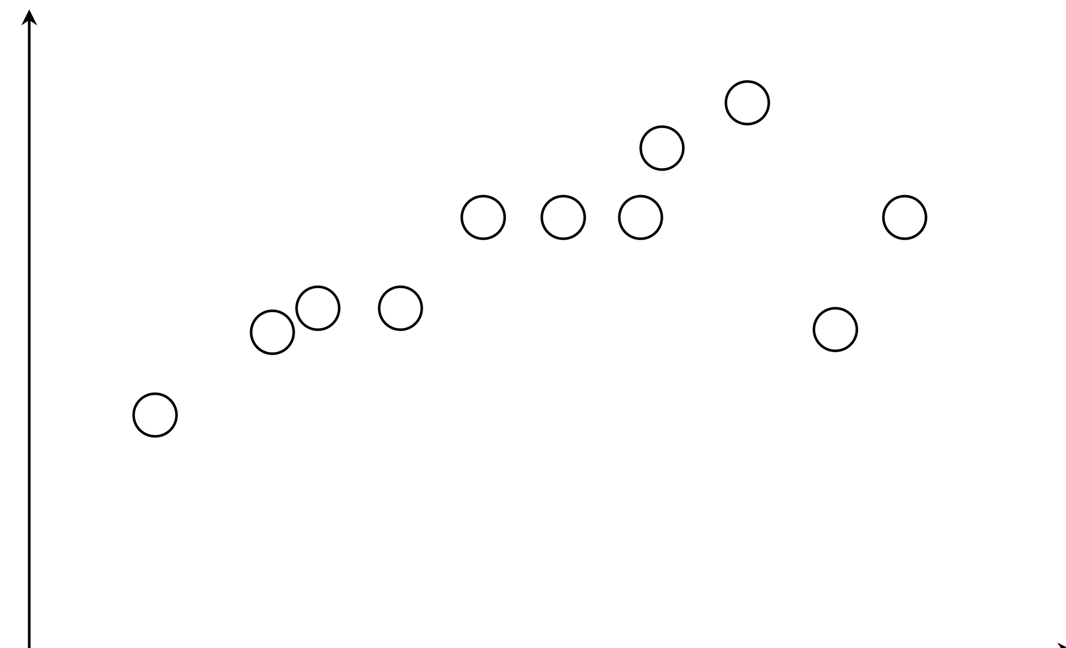
- RANSAC loop:

1. Select four feature pairs (at random)
2. Compute homography $\boldsymbol{H}$ (exact)
3. Compute inliers where $\|p_i', \boldsymbol{H}\, p_i\| < \varepsilon$

- Keep largest set of inliers
- Re-compute least-squares $\boldsymbol{H}$ estimate using all of the inliers

Simple example: fit a line

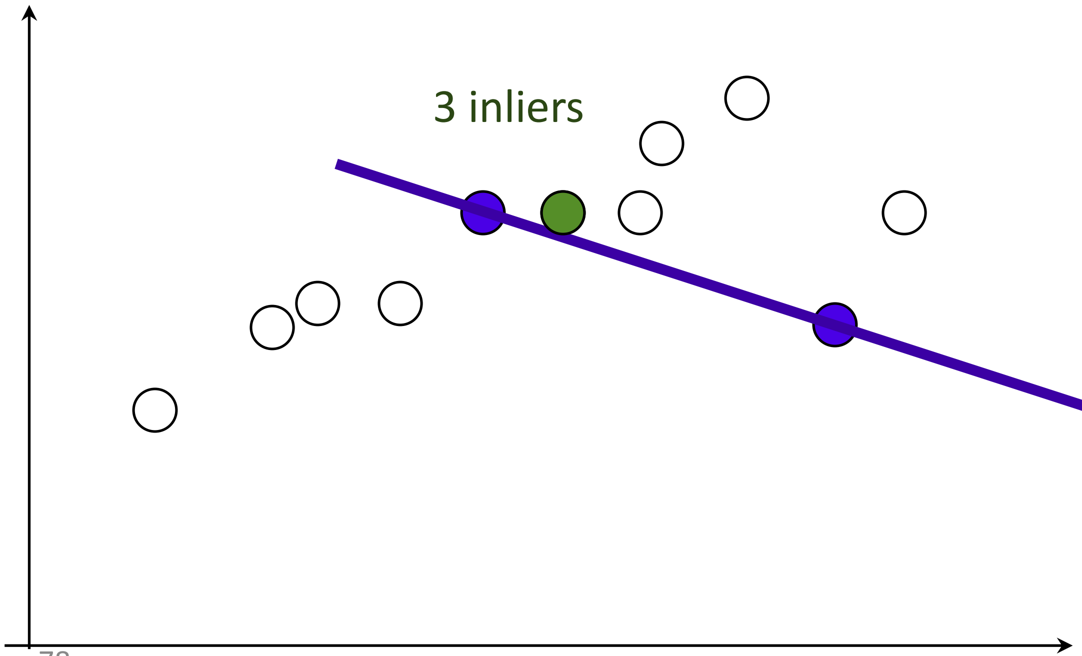
- Rather than homography H (8 numbers) fit y=ax+b (2 numbers a, b) to 2D pairs



77

Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers

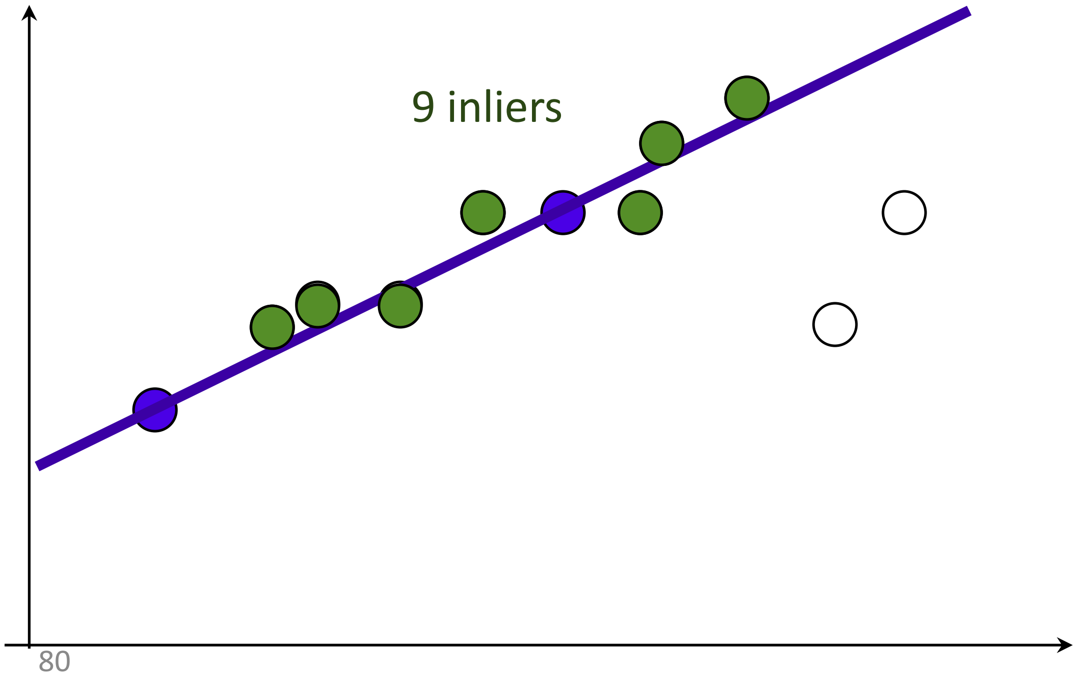


Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers

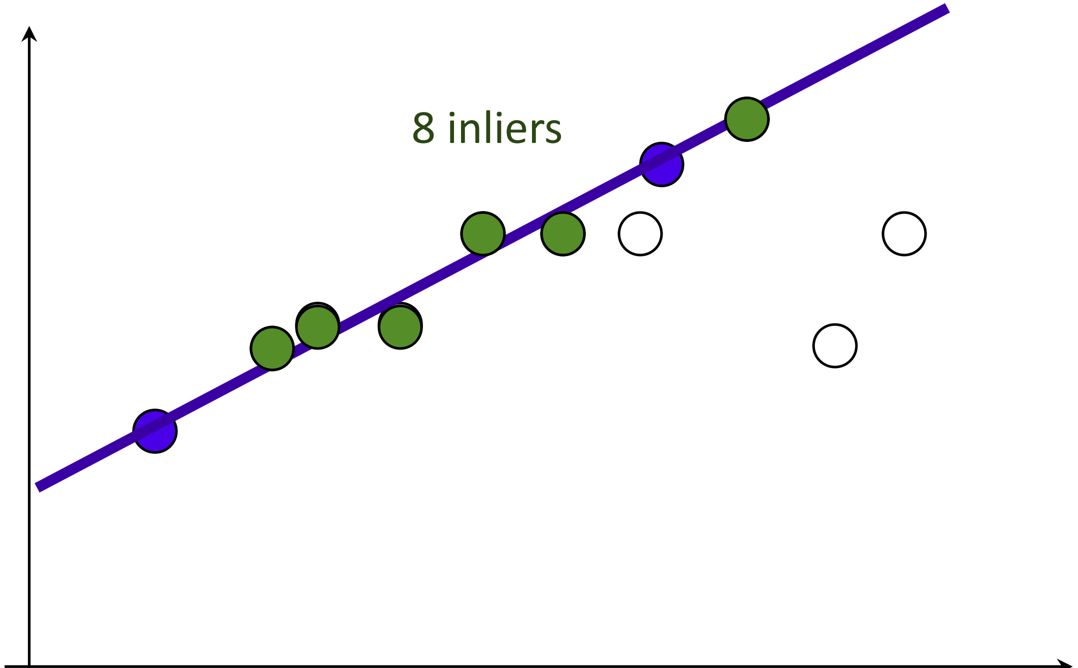
Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers



Simple example: fit a line

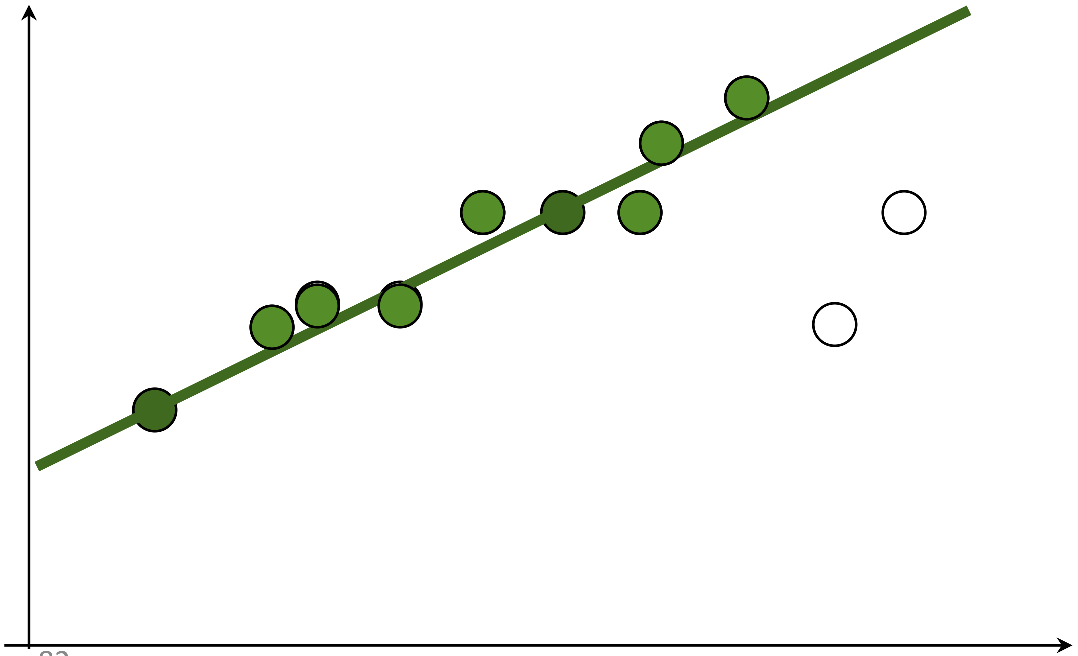
- Pick 2 points
- Fit line
- Count inliers



81

Simple example: fit a line

- Use biggest set of inliers
- Do least-square fit



82

What still needs to be fixed?

- The planar projections may not work so well
- Your homework uses cylindrical projections instead.

Panorama algorithm:

Find corners in both images

Calculate descriptors

Match descriptors

RANSAC to find homography

Stitch together images with homography

NEXT TIME.