图像变换、特征匹配与全景拼接

周晓巍

浙江大学CAD&CG实验室

Slides adapted from Noah Snavely, Jia-Bin Huang


DONE

How to combine two images?

How to combine two images?

First, we need to know what this transformation is.
Second, we need to know how to compute it using feature matches.

Part I
Image Warping
图像变形

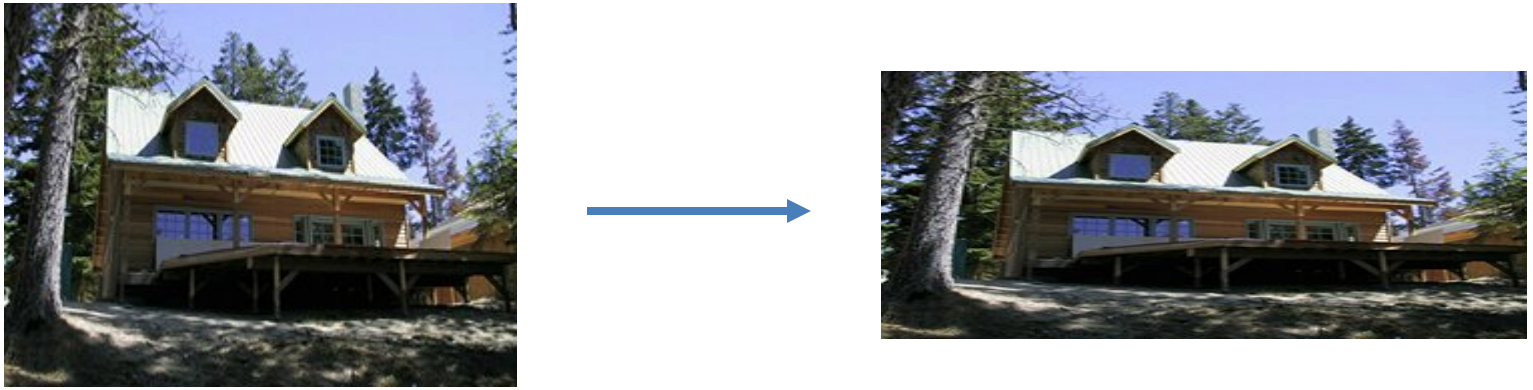
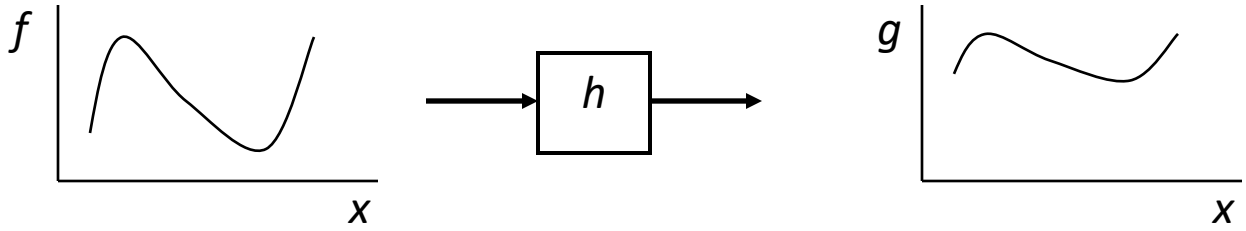


Image Warping

- filtering: change *range* of signal
  - $g(x) = h(f(x))$



- warping: change *domain* of signal
  - $g(x) = f(h(x))$

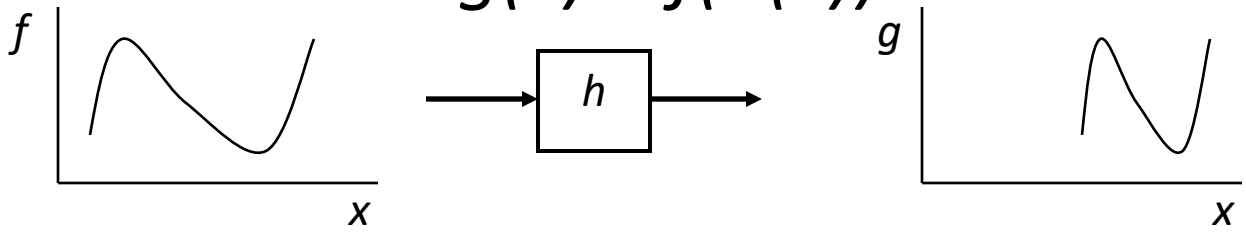
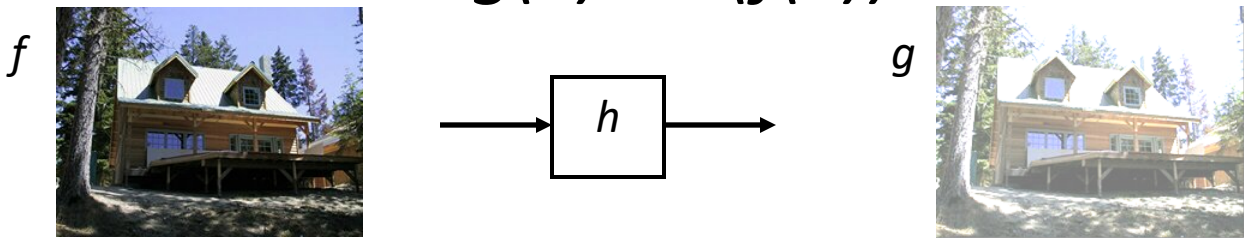
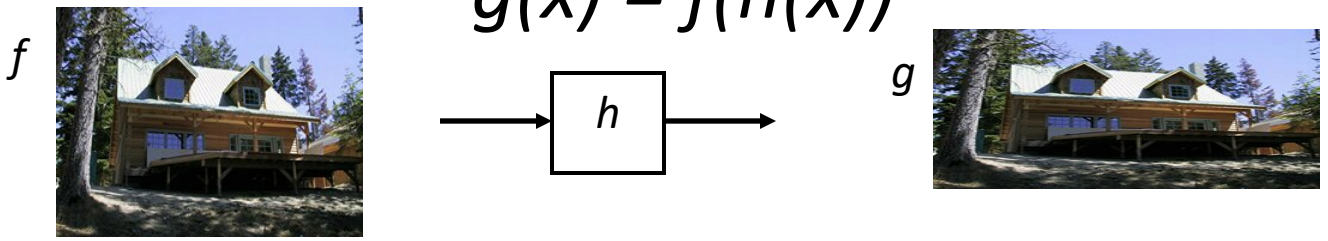


Image Warping

- image filtering: change *range* of image
  - $g(x) = h(f(x))$



- image warping: change *domain* of image
  - $g(x) = f(h(x))$



Parametric (global) warping

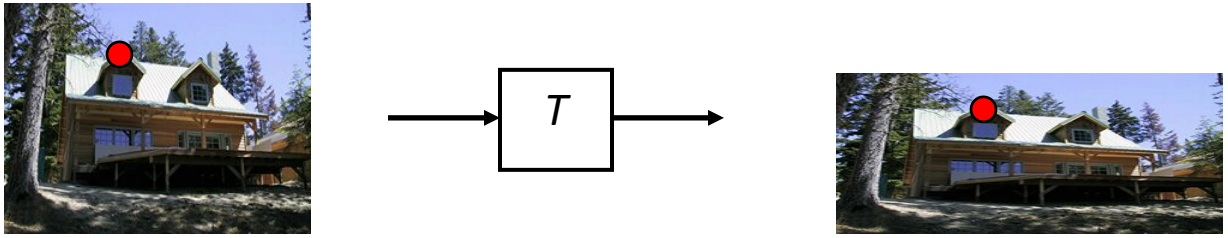
- Examples of parametric warps:

translation

rotation

aspect

Parametric (global) warping



p = (x,y)

p' = (x',y')

- Transformation T is a coordinate transformation:

$$p' = T(p)$$

- Let's consider *linear* forms (can be represented by a 2x2 matrix):

$$\mathbf{p}' = \mathbf{T}\mathbf{p} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

Common linear transformations

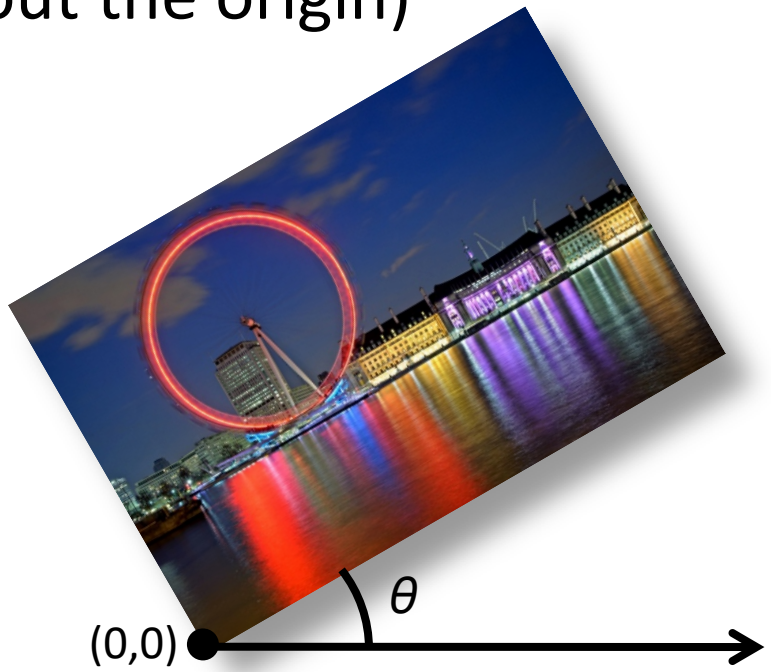
- Uniform scaling by *s*:

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

What is the inverse?

Common linear transformations

- Rotation by angle θ (about the origin)



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

What is the inverse?

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D mirror about Y axis?

$$\begin{aligned} x' &= -x \\ y' &= y \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

2D mirror across line y = x?

$$\begin{aligned} x' &= y \\ y' &= x \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Translation?

$$x' = x + t_x$$
$$y' = y + t_y$$

NO!

Translation is not a linear operation on 2D coordinates

How to make it a linear operation?

Make translation a linear operation

- Solution: homogeneous coordinates

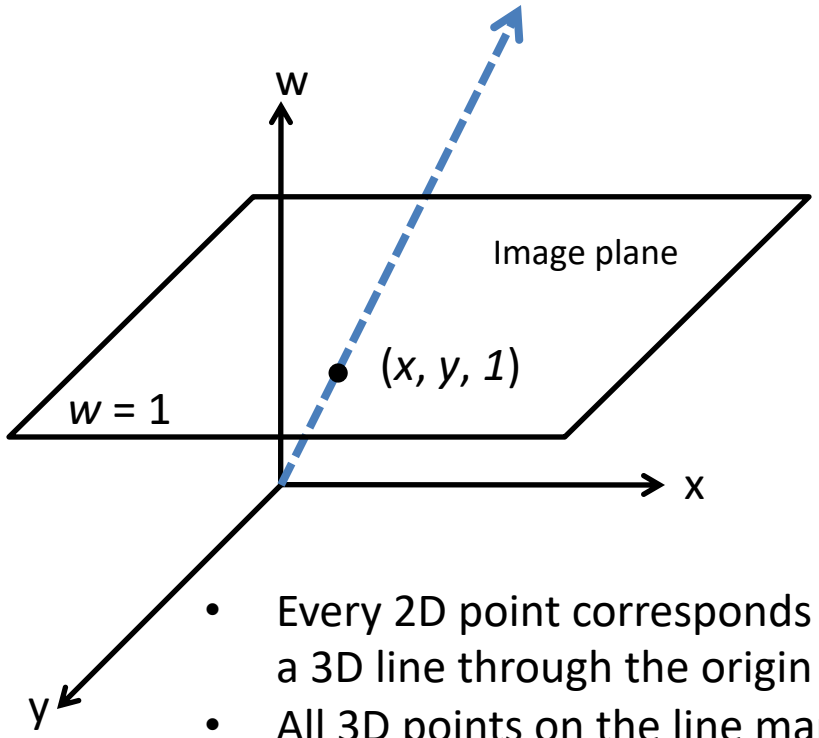
$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

Homogeneous coordinates

Trick: add one more coordinate:

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous coordinates



- Every 2D point corresponds to a 3D line through the origin
- All 3D points on the line map to the same 2D point

Transformations in homogeneous coordinates

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Translate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 & 0 \\ 0 & \boldsymbol{s}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Scale

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

2D *in-plane* rotation

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh}_x & 0 \\ \boldsymbol{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear

Affine transformations

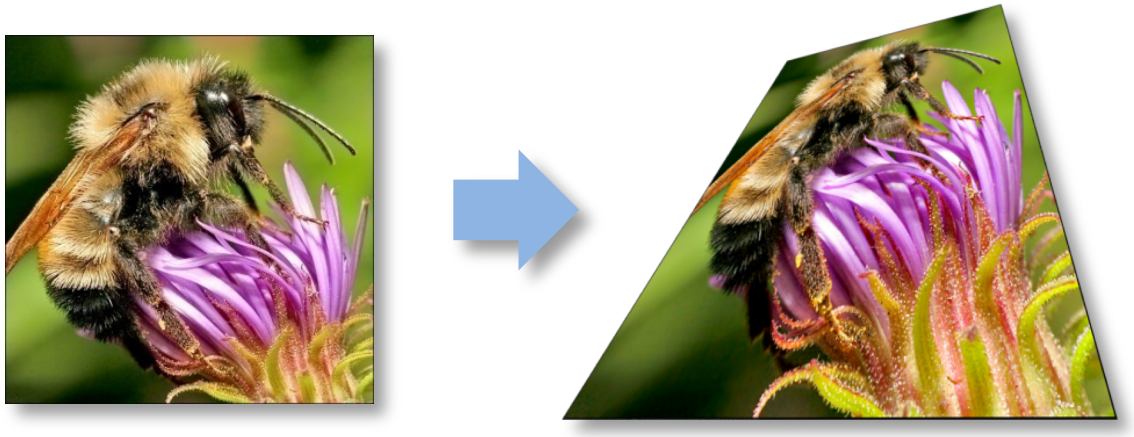
$$\mathbf{T} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix}$$

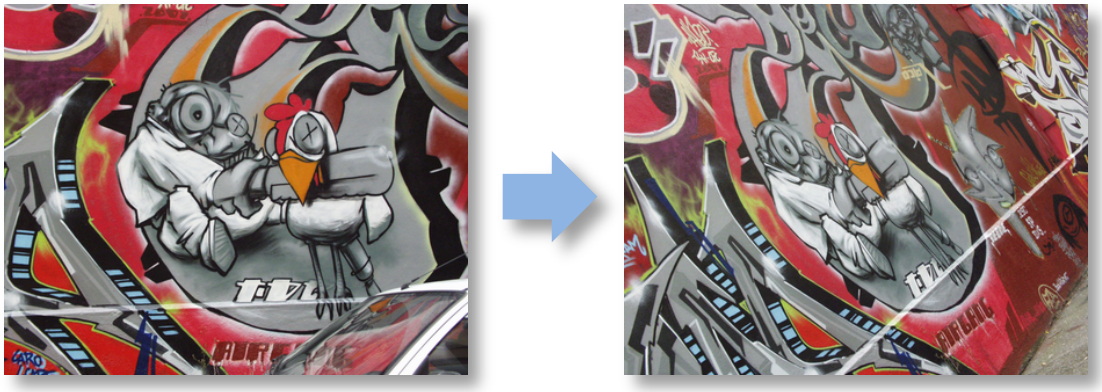
any transformation represented by a 3x3 matrix with last row [0 0 1] we call an *affine* transformation

What types of transformations are affine?

Is this an affine transformation?

Projective Transformations (Homographies)





Projective Transformations (Homographies)

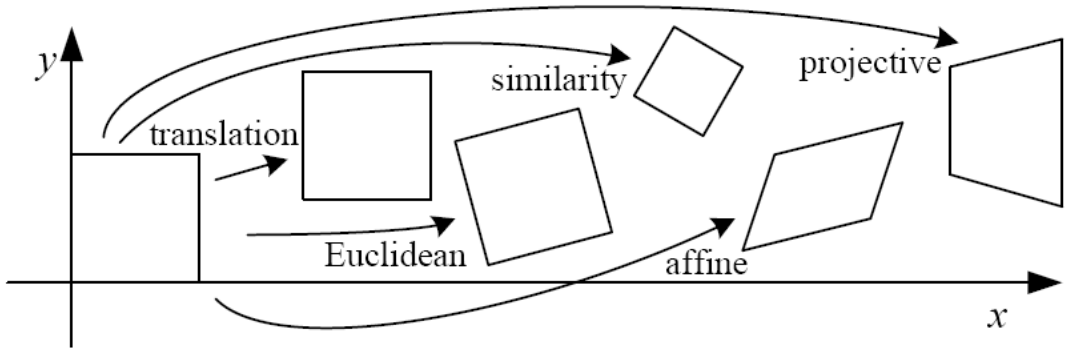
$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} \cong \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

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

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

- The matrix is called homography
- We usually constrain the length of the vector [h_{00} h_{01} … h_{22}] to be 1, which means the degree of freedom is 8

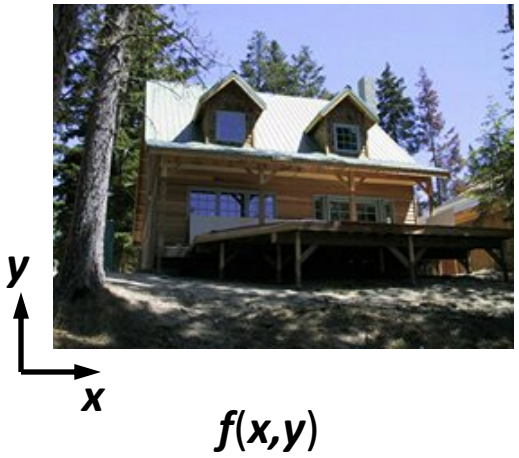
2D image transformations



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles $+\cdots$	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism $+\cdots$	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

Implementing image warping

- Given a coordinate transform $(x',y') = T(x,y)$ and a source image $f(x,y)$, how do we compute an transformed image $g(x',y') = f(T(x,y))$?

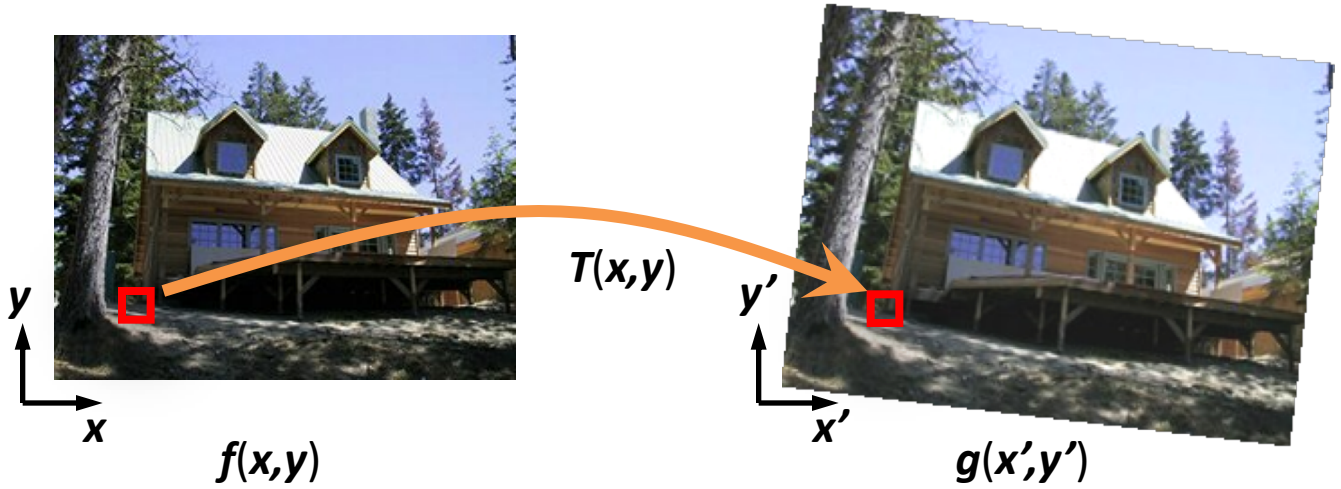


$f(x,y)$

$g(x',y')$

Forward Warping

- Send each pixel $f(x)$ to its corresponding location $(x',y') = T(x,y)$ in $g(x',y')$

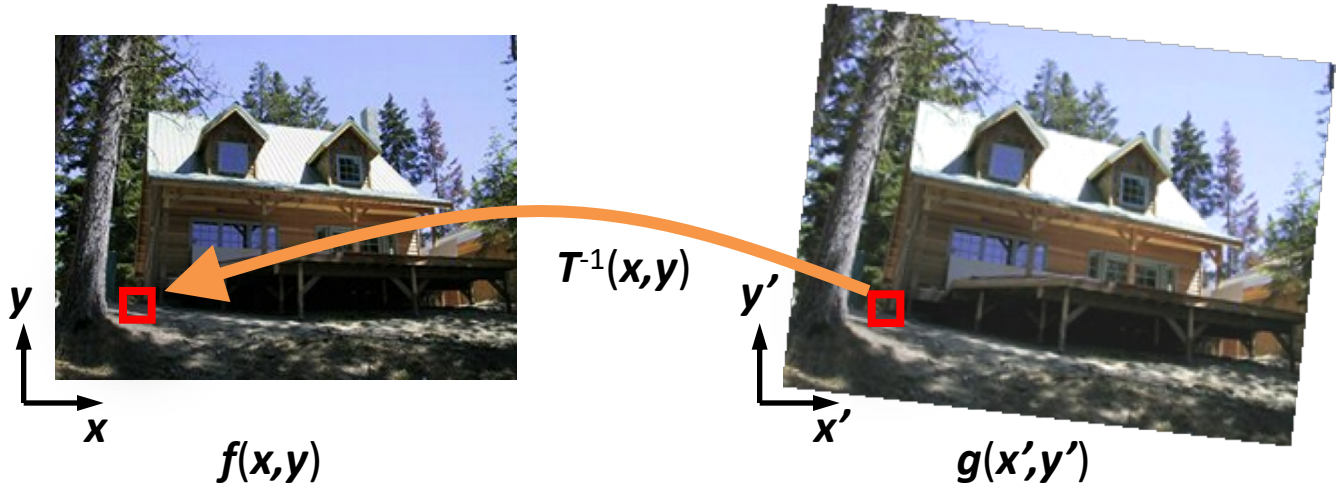


Forward Warping

- What if pixel lands “between” pixels?

Inverse Warping

- Get each pixel $g(x',y')$ from its corresponding location $(x,y) = T^{-1}(x,y)$ in $f(x,y)$



Inverse Warping

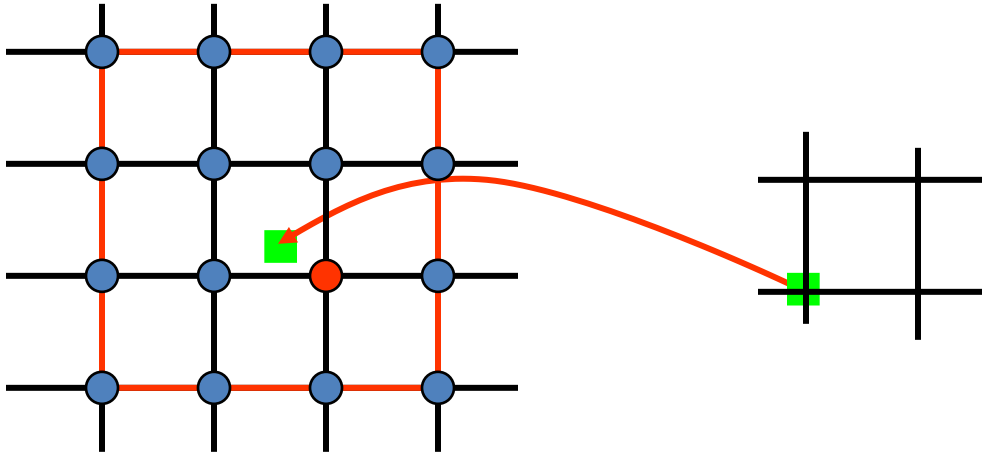
- What if pixel lands “between” pixels?

Answer: interpolate color values from neighboring pixels

Interpolation

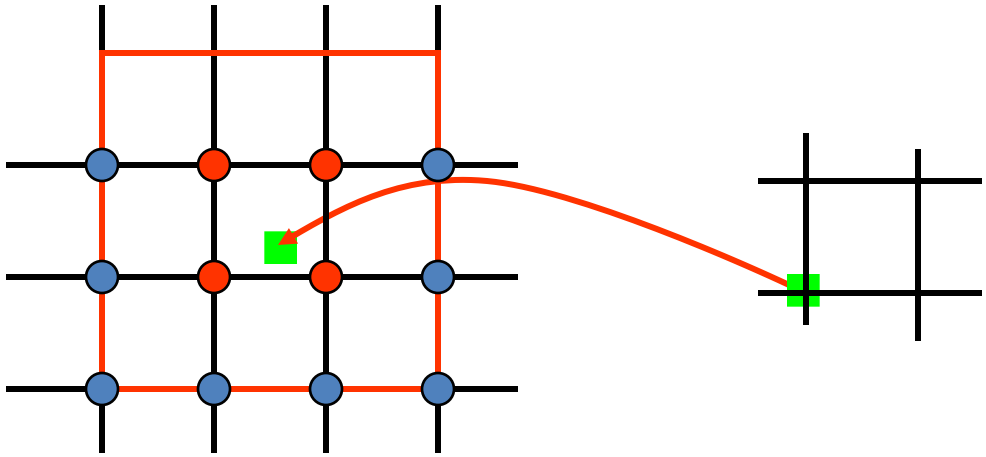
Nearest neighbor

- Copies the color of the pixel with the closest integer coordinate



Interpolation

Weighted sum of four neighboring pixels



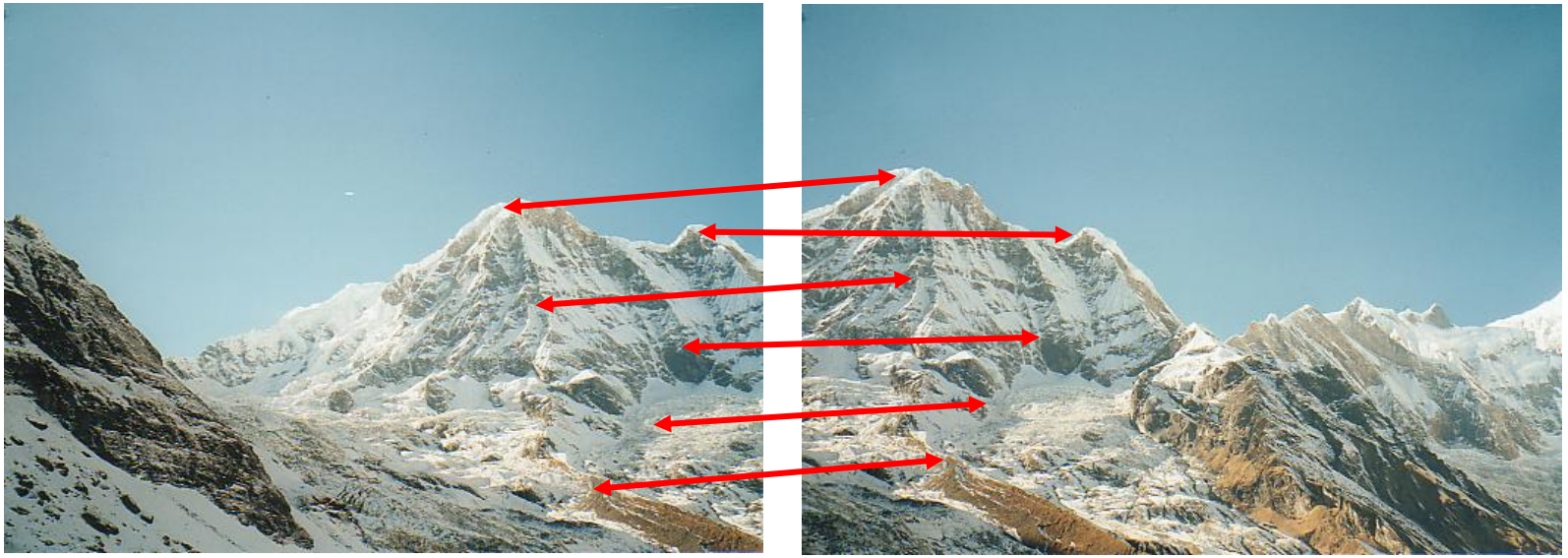
Interpolation

- Possible interpolation filters:
  - nearest neighbor
  - bilinear
  - bicubic
  - sinc

Questions?

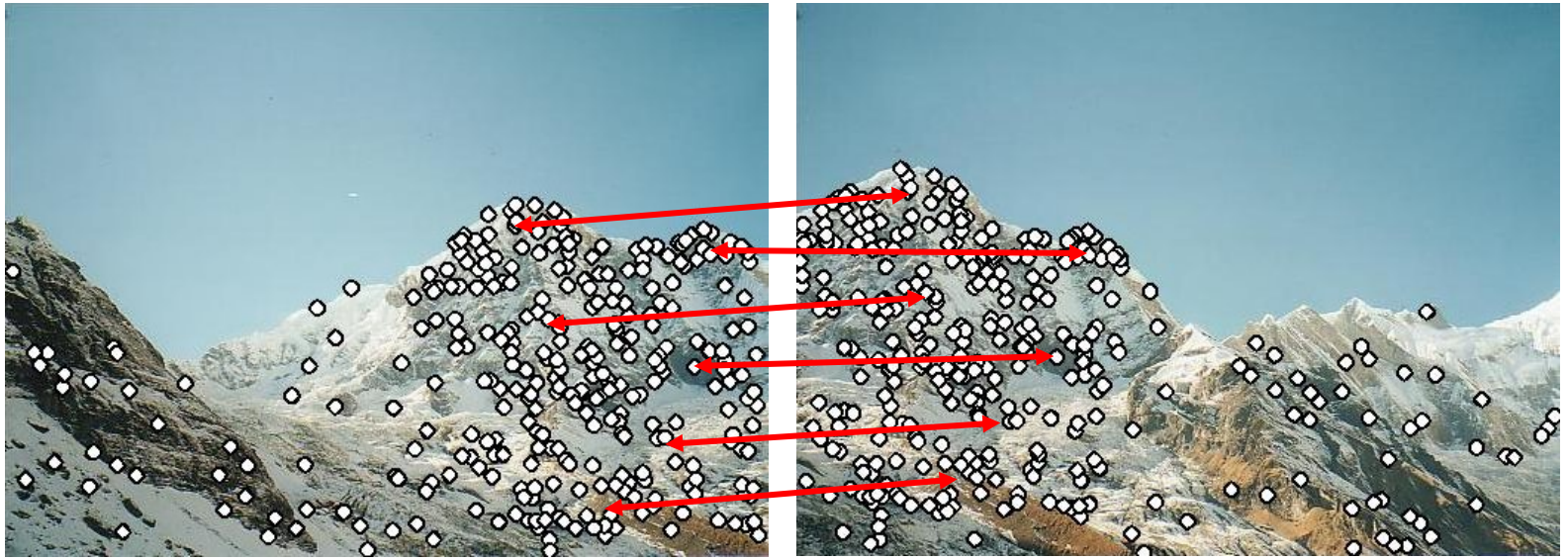
Part II
Feature Matching

How to compute transformation?



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \cong T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

How to do feature matching?

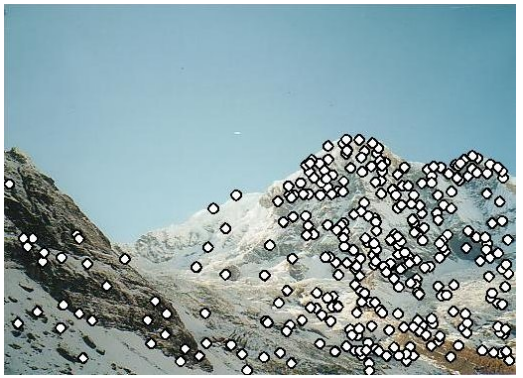


Step 1: extract features

Step 2: match features

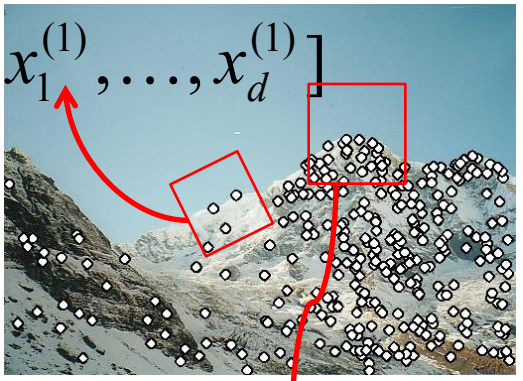
Main Components of Feature matching

1) **Detection:** Identify the interest points



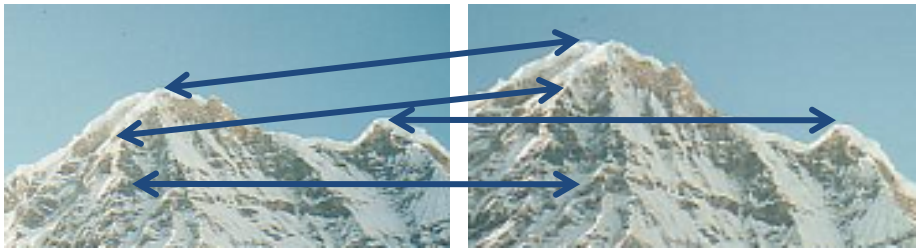
2) **Description:** Extract vector feature descriptor surrounding each interest point.

$$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$$



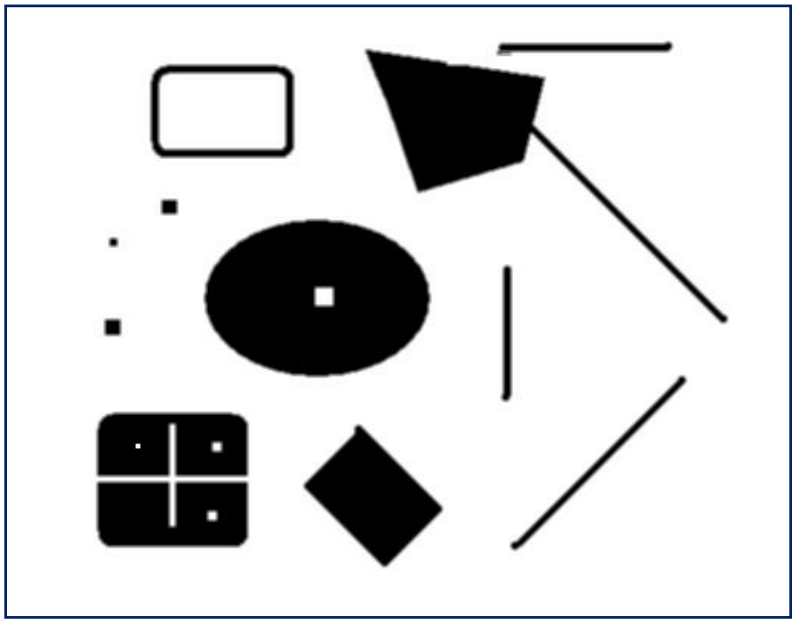
$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$

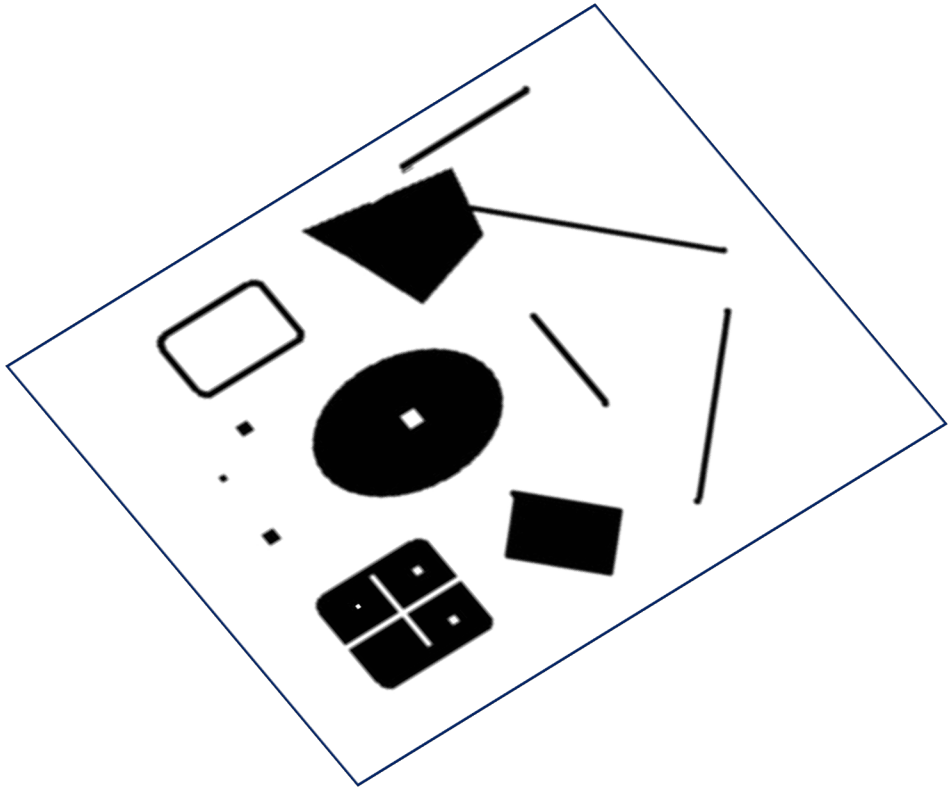
3) **Matching:** Determine correspondence between descriptors in two views



Interest points

Which points will you choose to match these two images?





Want uniqueness

Look for image regions that are unusual

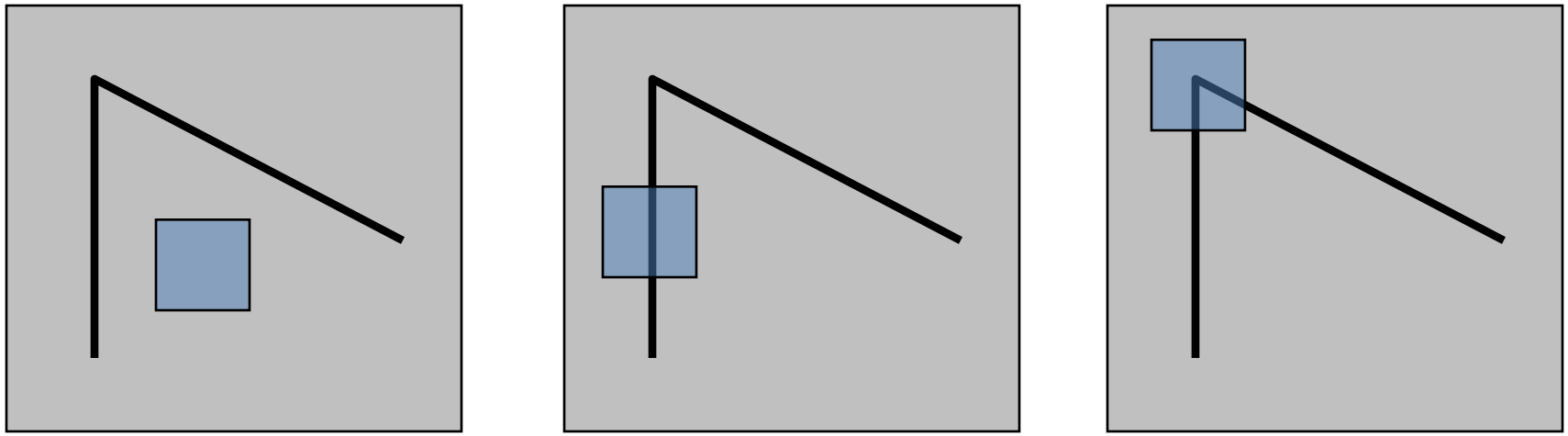
- Lead to unambiguous matches in other images

How to define “unusual”?

Local measures of uniqueness

Suppose we consider a small window of pixels (region)

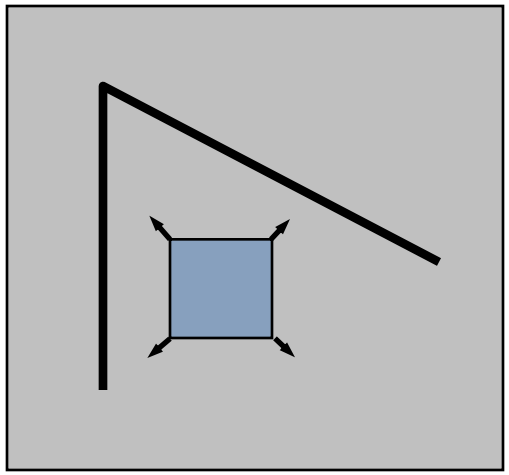
- What defines whether a region is unique?



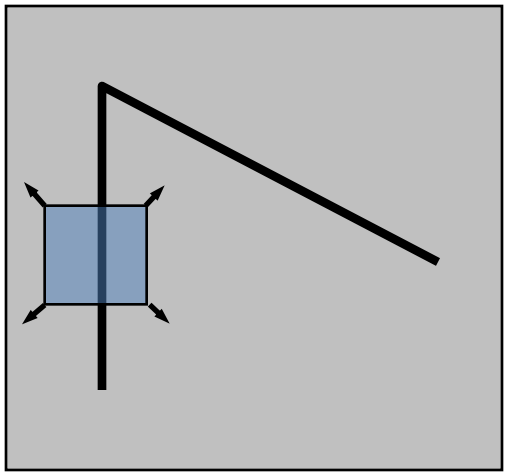
Credit: S. Seitz, D. Frolova, D. Simakov

Local measures of uniqueness

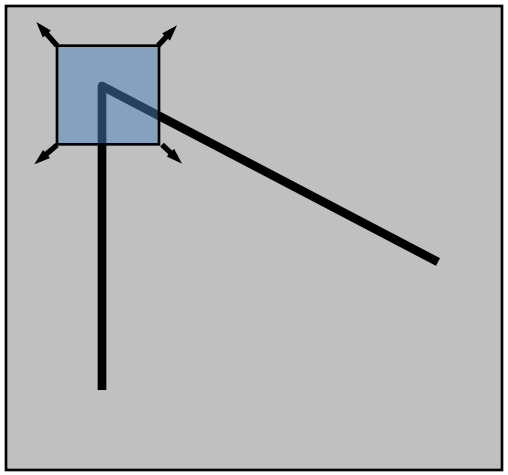
- Shifting the window in any direction causes a big change



"flat" region:
no change in all directions



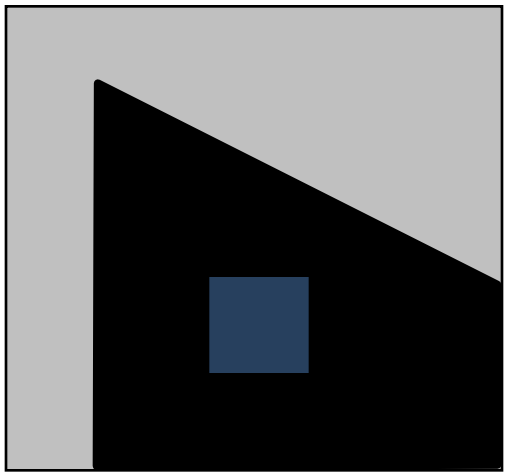
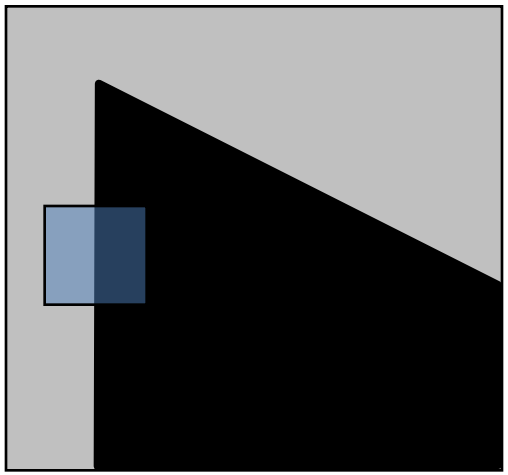
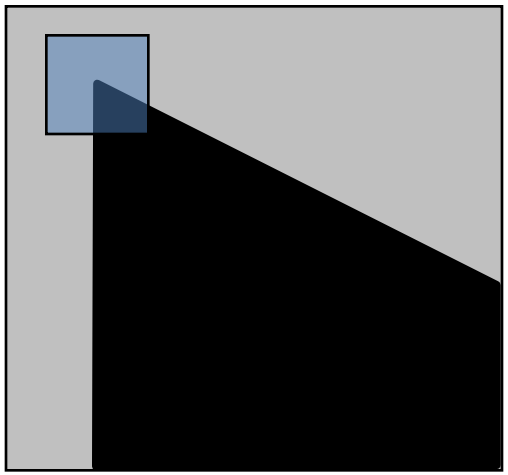
"edge":
no change along the edge direction



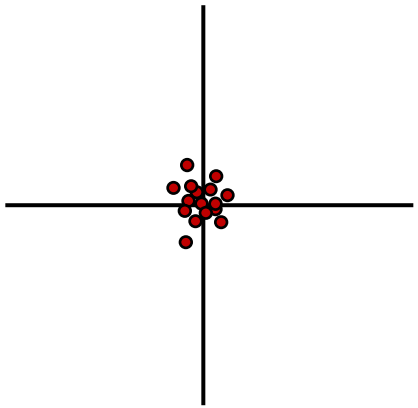
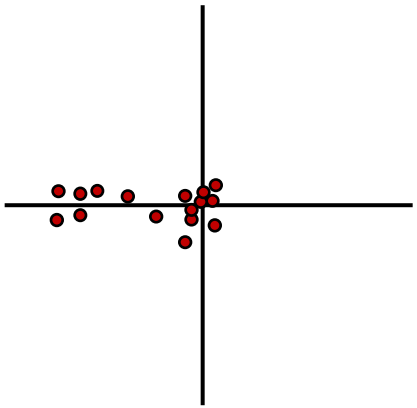
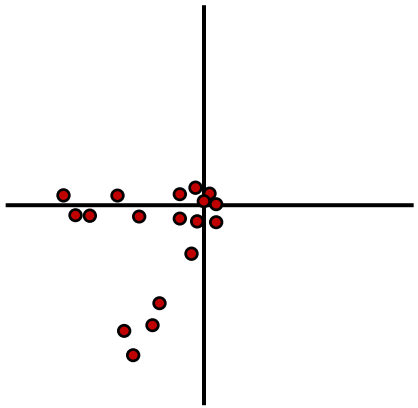
"corner":
significant change in all directions

Credit: S. Seitz, D. Frolova, D. Simakov

How to measure uniqueness mathematically?

Let‘s look at the distribution of gradients in the region:

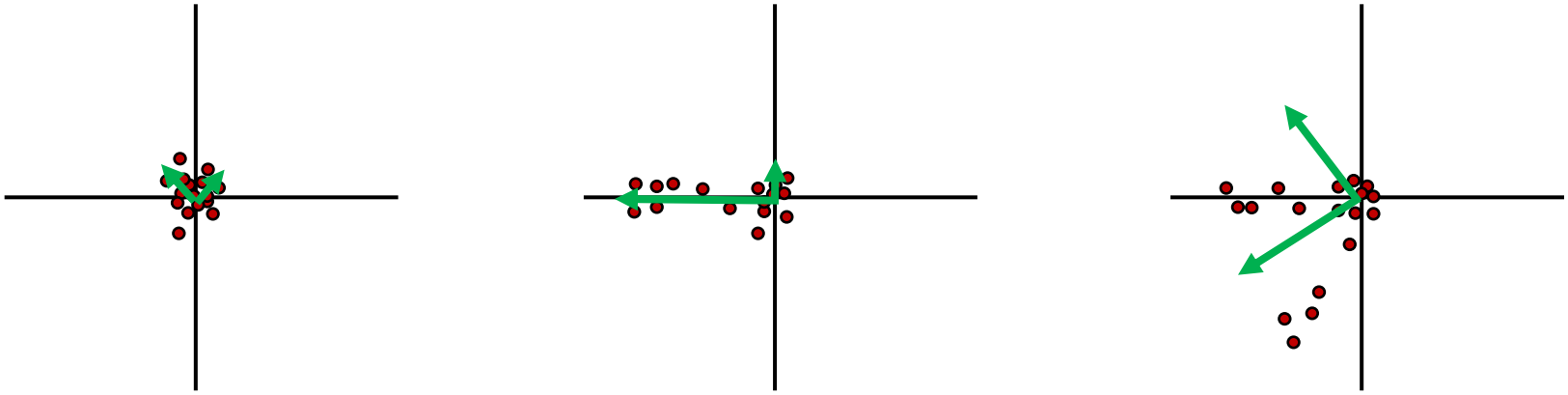
Principle Component Analysis

Principal component is the direction of highest variance.

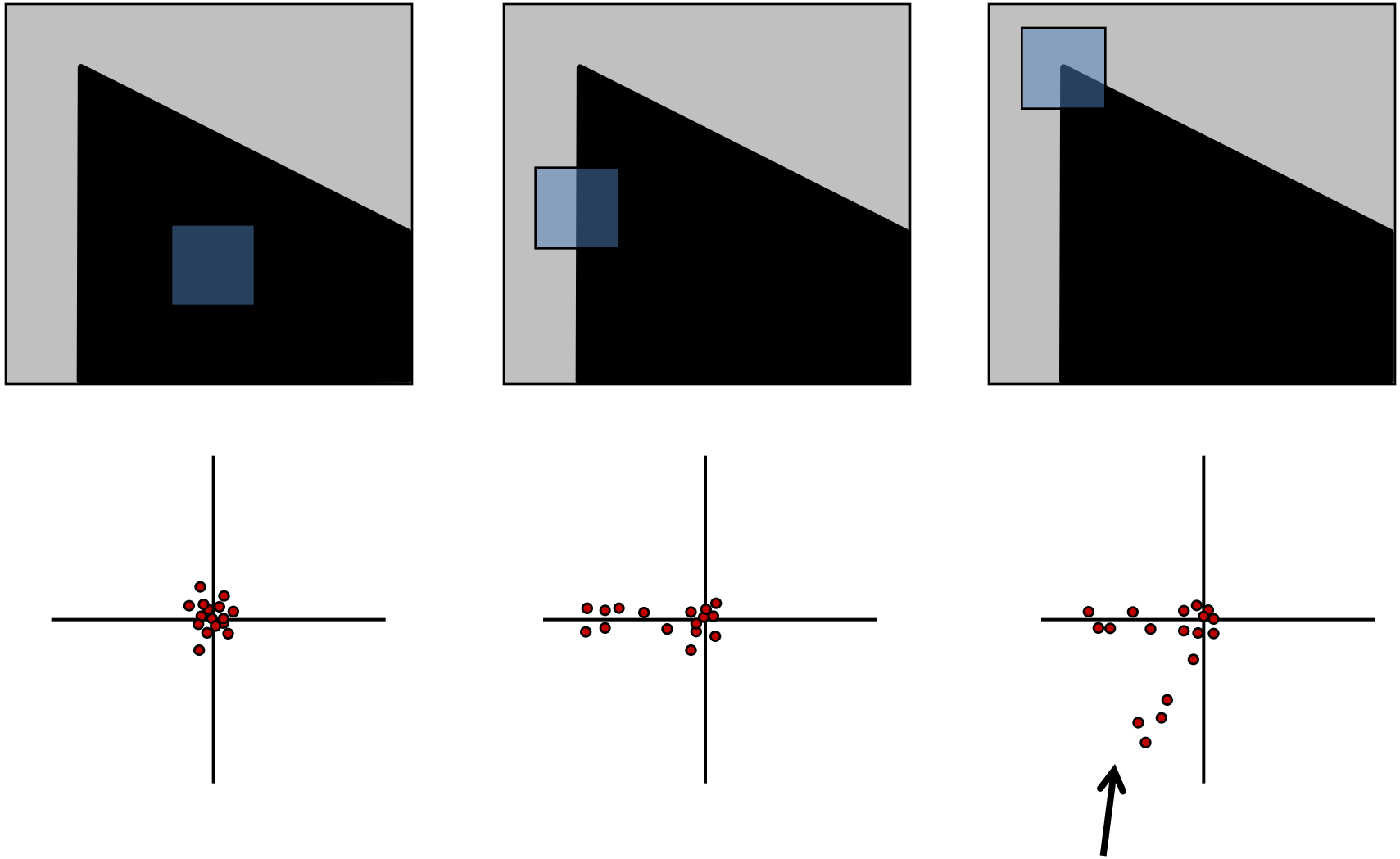
Next, highest component is the direction with highest variance *orthogonal* to the previous components.

How to compute PCA components:
1.Subtract off the mean for each data point.
2.Compute the covariance matrix.
3.Compute eigenvectors and eigenvalues.
4.The components are the eigenvectors ranked by the eigenvalues.

$$Hx = \lambda x$$



Corners have …



The math

To compute the eigenvalues:

1. Compute the covariance matrix.

$$H = \sum_{(u,v)} w(u,v) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \qquad I_x = \frac{\partial f}{\partial x}, I_y = \frac{\partial f}{\partial y}$$

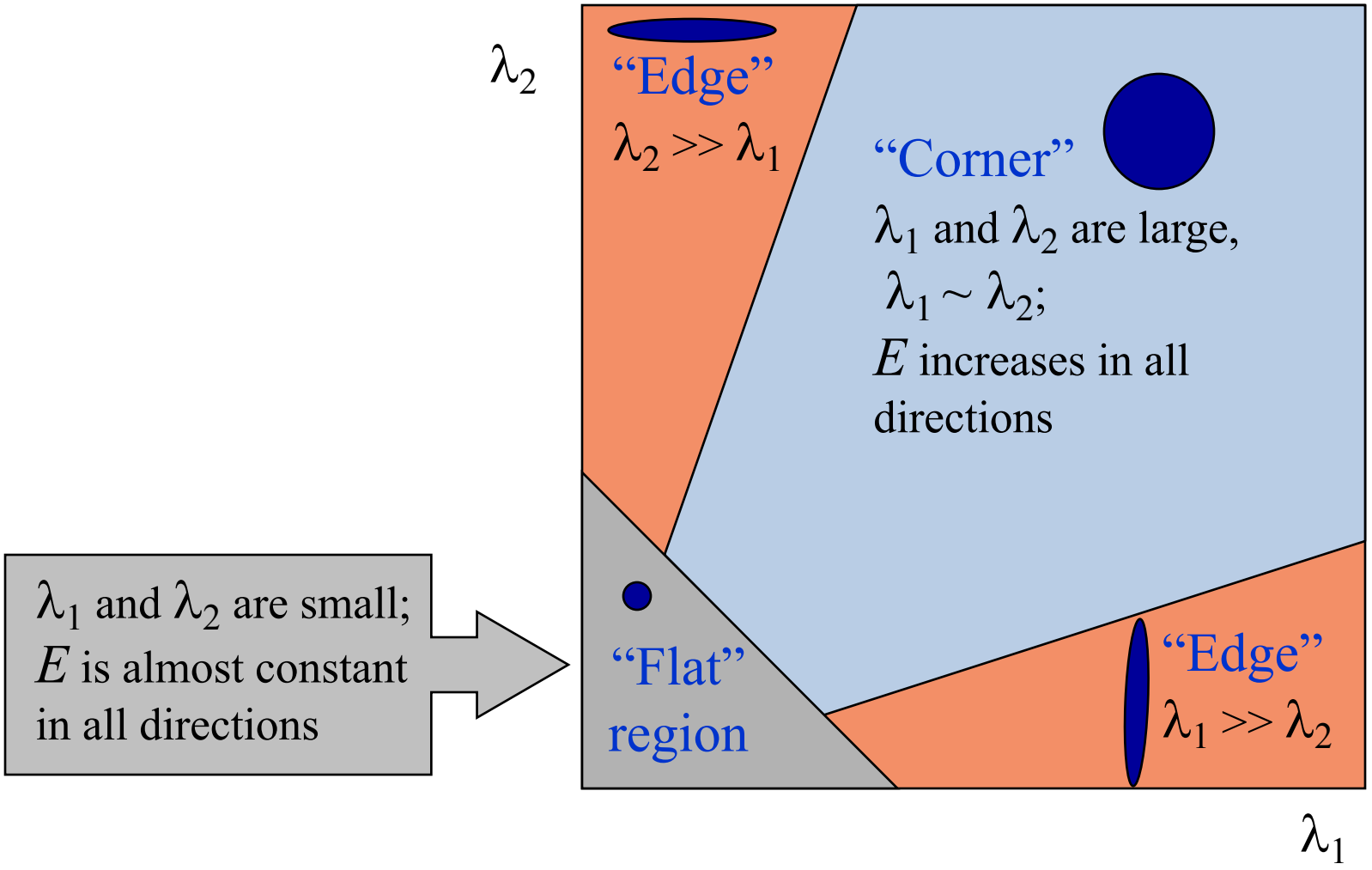
Typically Gaussian weights

2. Compute eigenvalues.

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

Interpreting the eigenvalues

Classification of image points using eigenvalues of *M*:



The Harris operator

Computing eigenvalues are expensive
Harris corner detector uses the following alternative

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{determinant(H)}{trace(H)}$$

Reminder:

$$det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc \qquad trace\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = a + d$$

Harris detector: Steps

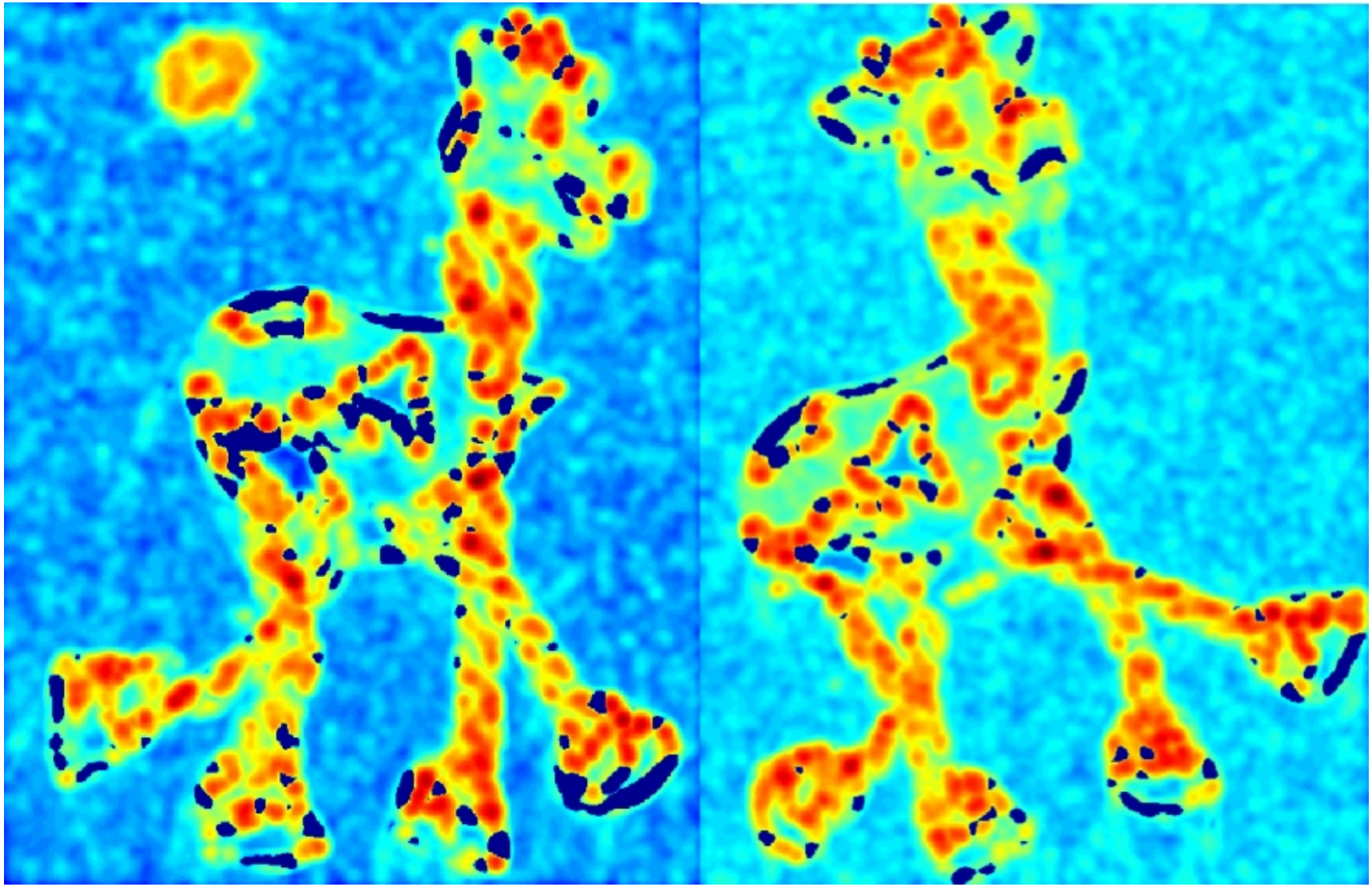
1. Compute derivatives at each pixel
2. Compute second moment matrix *M* in a Gaussian window around each pixel
3. Compute corner response function *f*
4. Threshold *f*
5. Find local maxima of response function (nonmaximum suppression)

C.Harris and M.Stephens. "A Combined Corner and Edge Detector." *Proceedings of the 4th Alvey Vision Conference*: pages 147—151, 1988.

Harris Detector: Steps

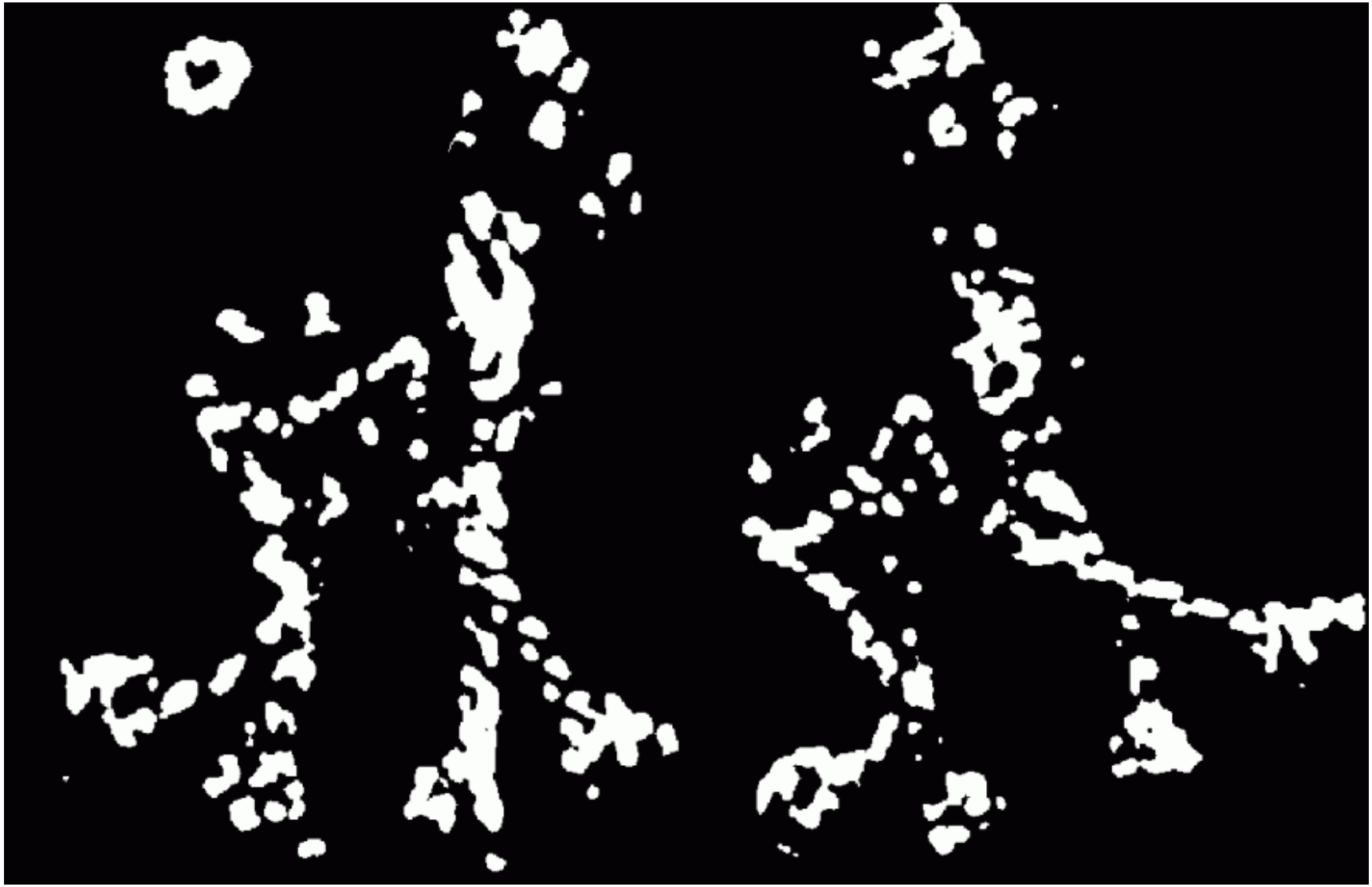
Harris Detector: Steps

Compute corner response *f*



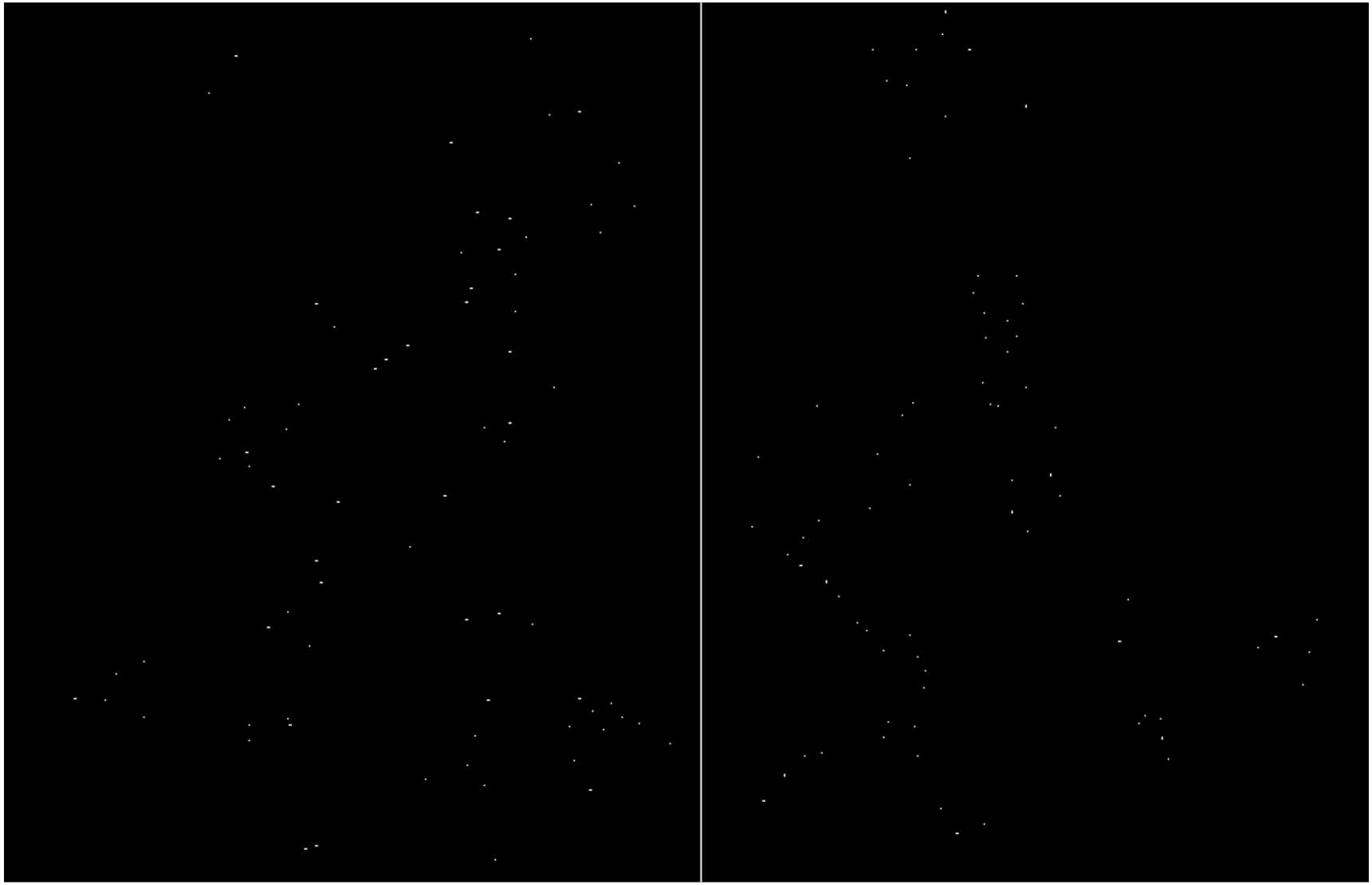
Harris Detector: Steps

Find points with large corner response: f>threshold



Harris Detector: Steps

Take only the points of local maxima of *f*

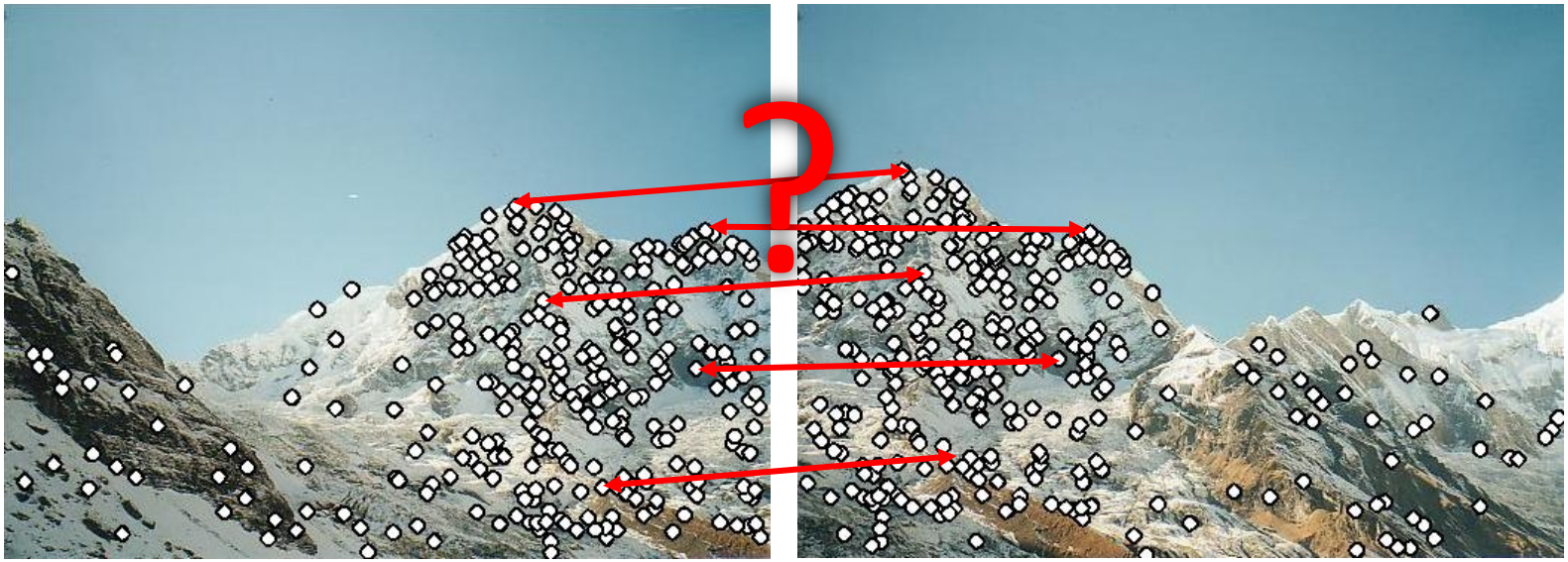


Harris Detector: Steps

Feature descriptors

We know how to detect good points

Next question: **How to match them?**

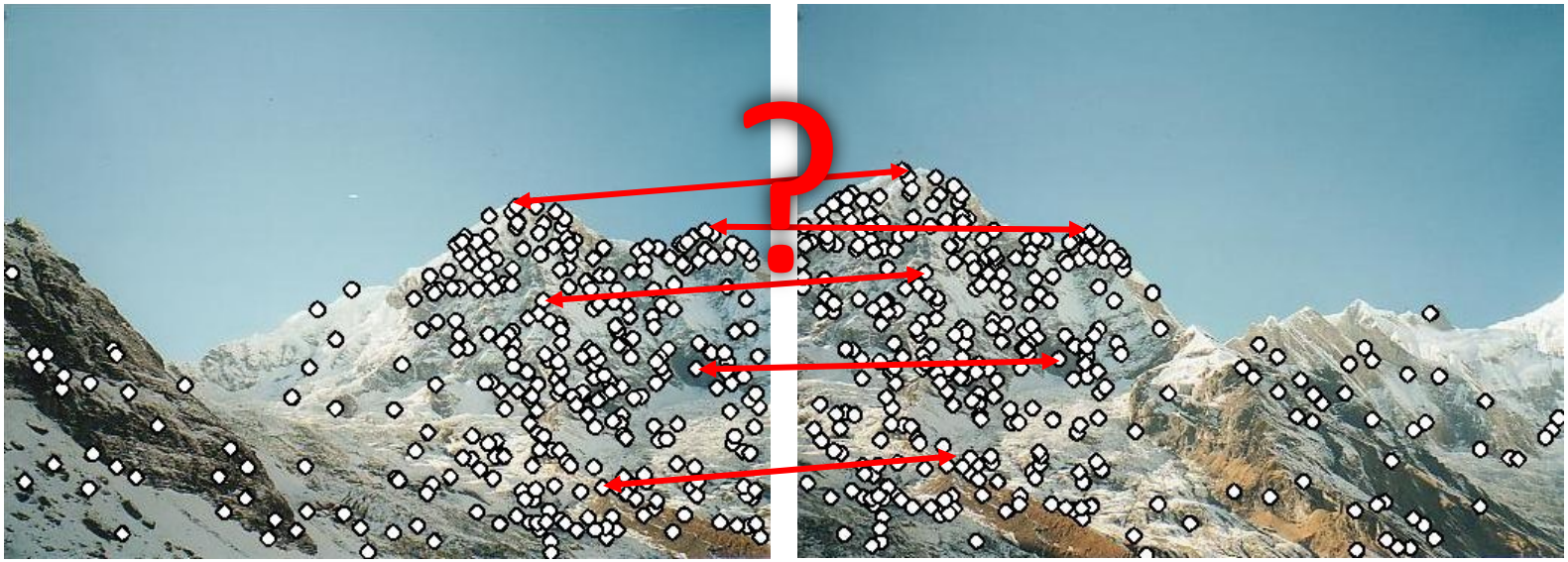


Answer: Come up with a *descriptor* for each point, find similar descriptors between the two images

Feature descriptors

We know how to detect good points

Next question: **How to match them?**



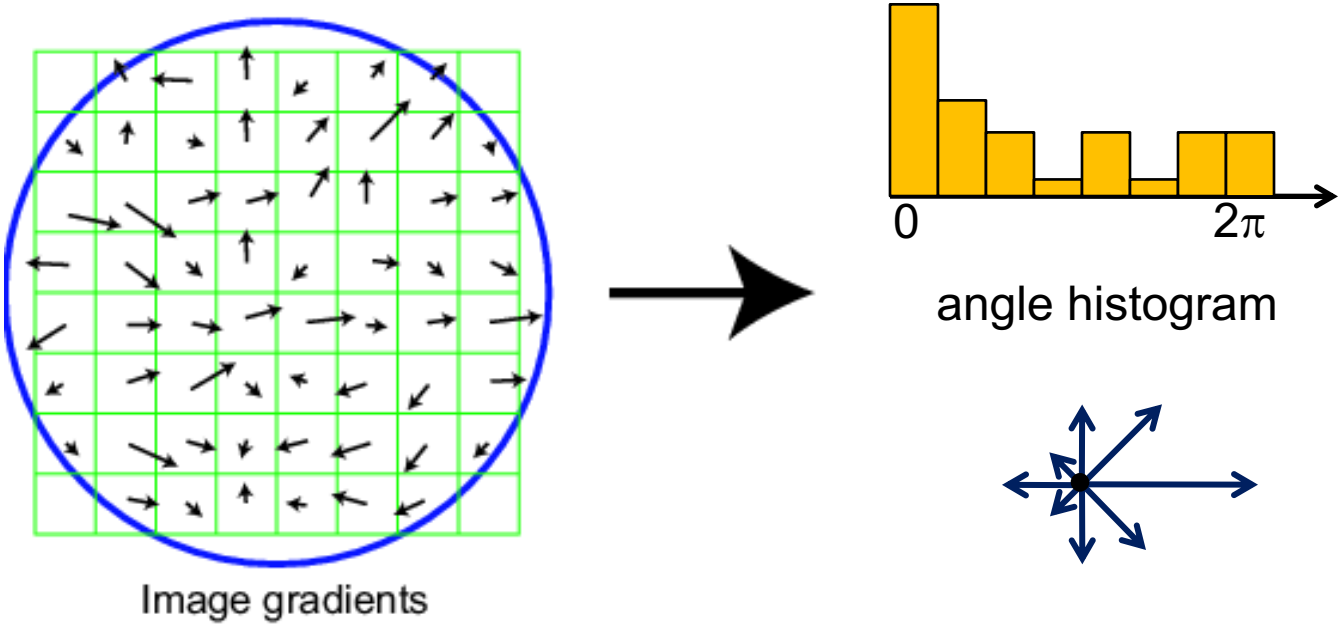
Lots of possibilities

- Simple option: match square windows around the point
- State of the art approach: SIFT
  - David Lowe, UBC http://www.cs.ubc.ca/~lowe/keypoints/

Scale Invariant Feature Transform

Basic idea:

- Take NxN square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations

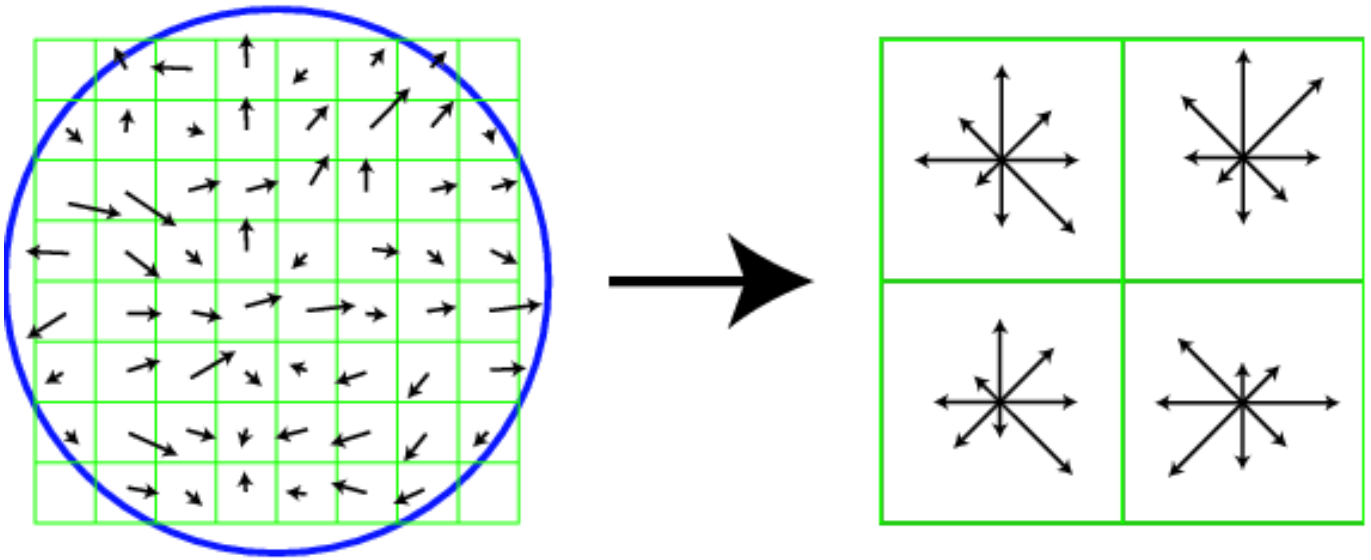


Adapted from slide by David Lowe

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



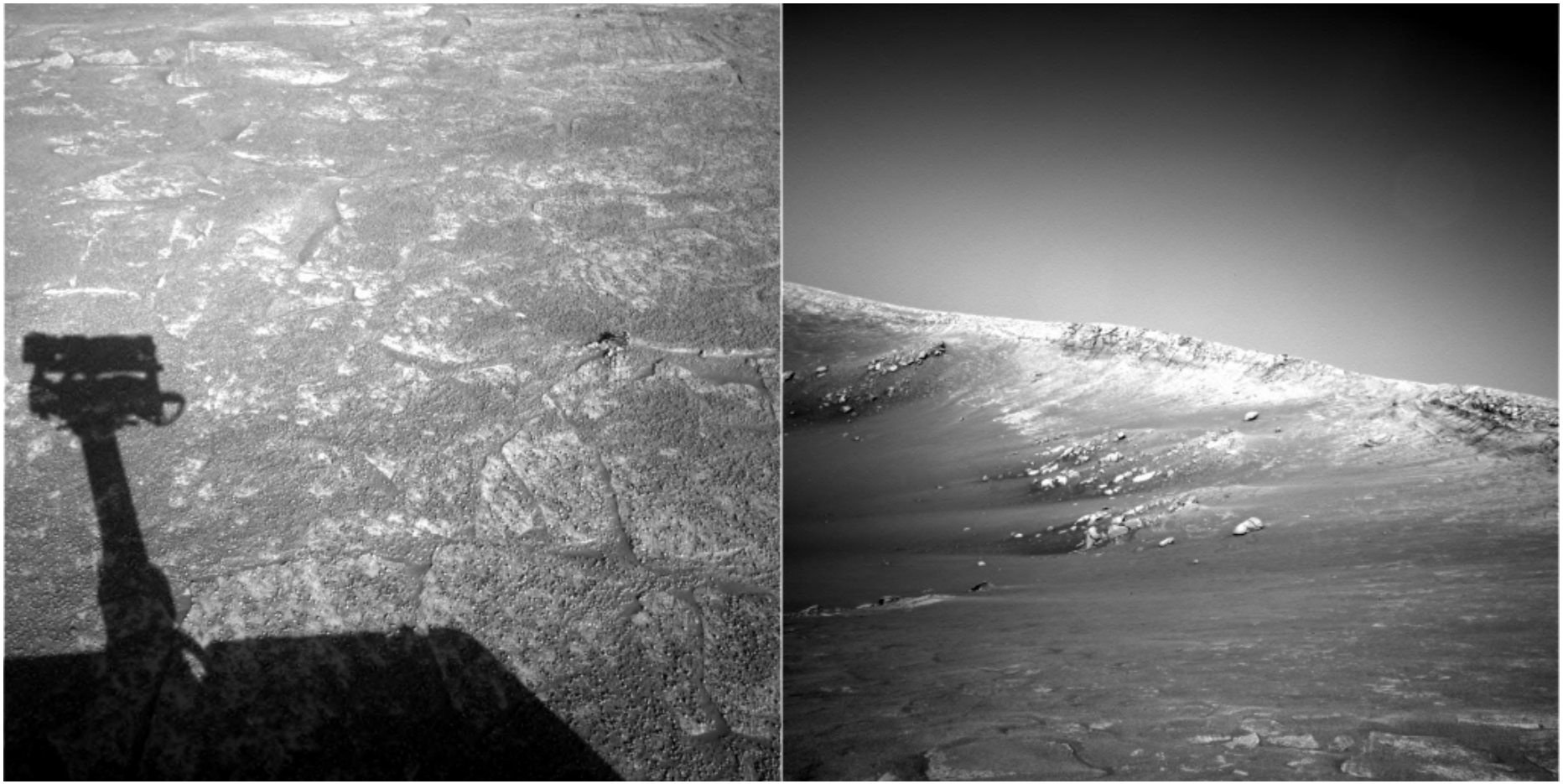
Adapted from slide by David Lowe

Properties of SIFT

Extraordinarily robust matching technique

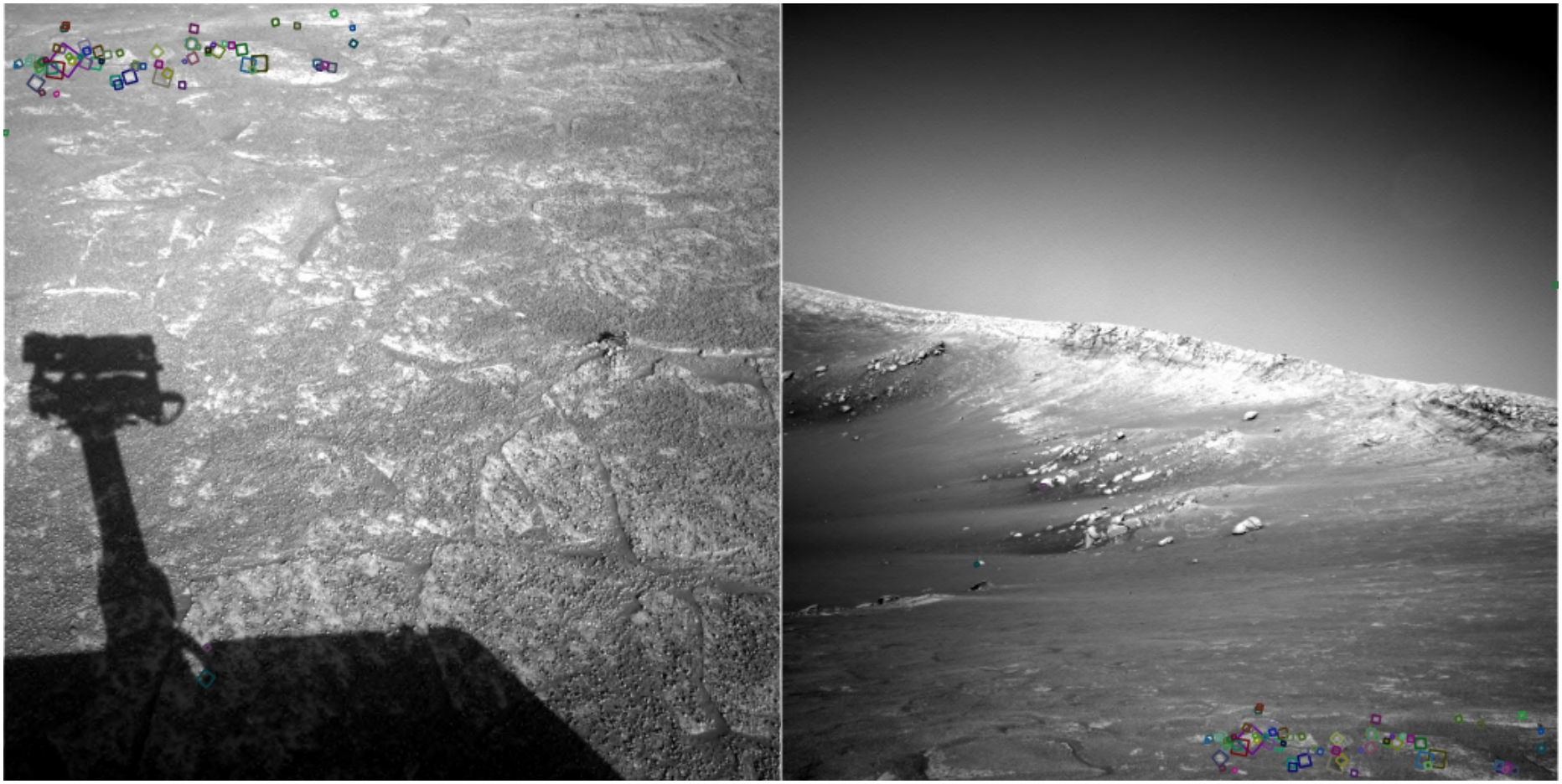
- Can handle changes in viewpoint
  - Up to about 60 degree out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available
  - http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT

Example



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

Example

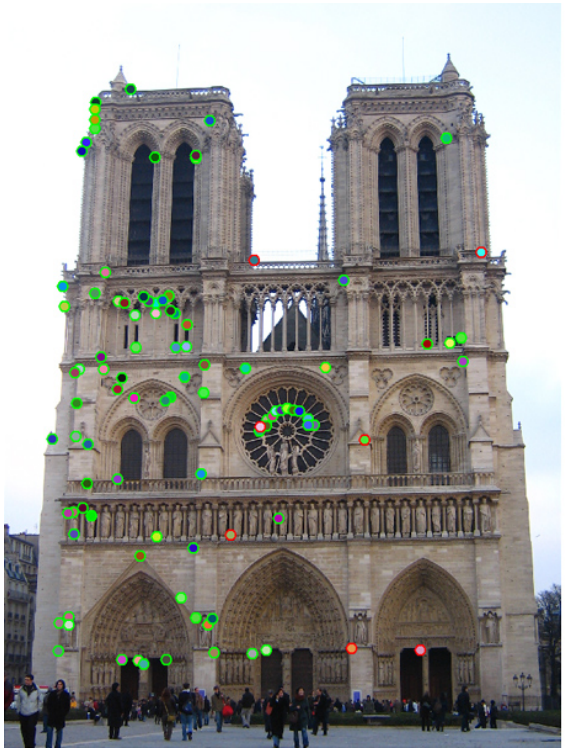


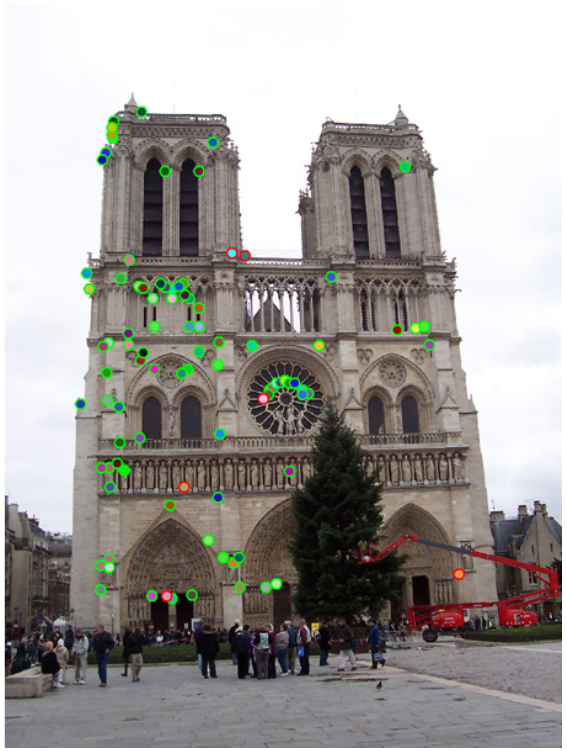
NASA Mars Rover images
with SIFT feature matches
Figure by Noah Snavely

Feature matching

Given a feature in I_1, how to find the best match in I_2?

1. Define distance function that compares two descriptors
2. Test all the features in I_2, find the one with min distance

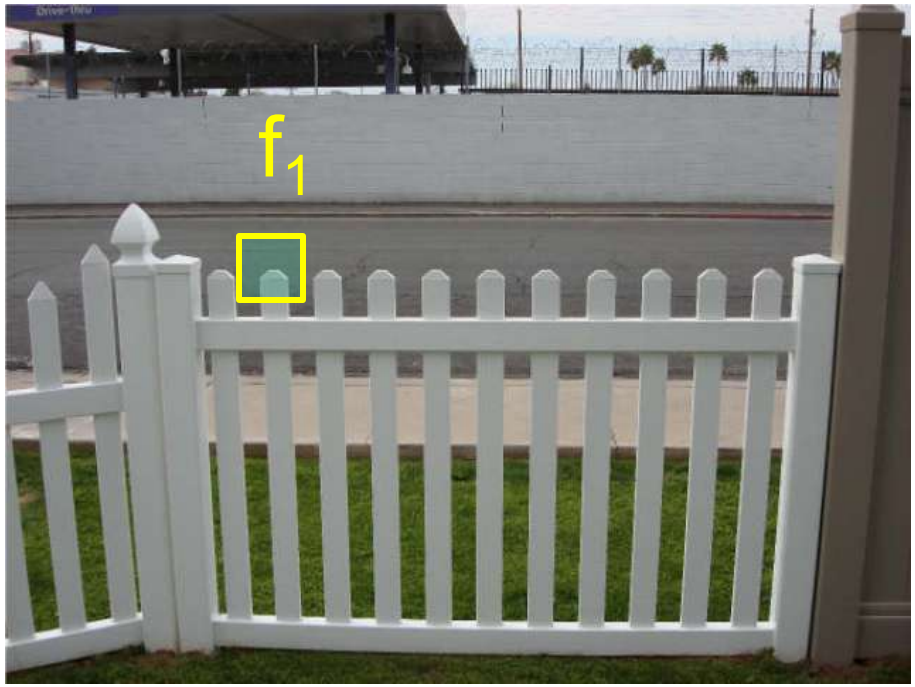




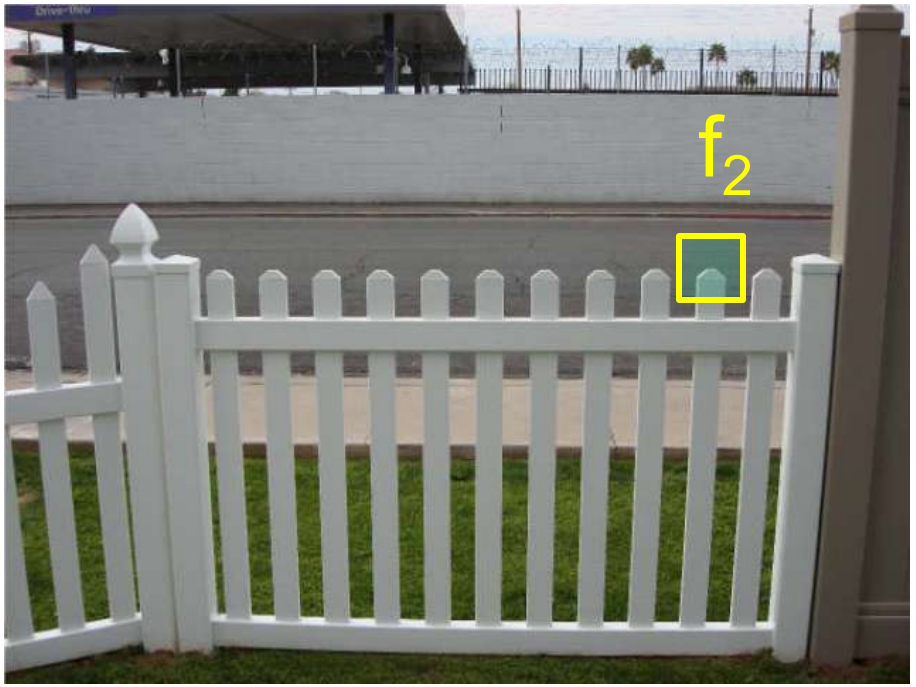
Feature distance

How to define the difference between two features f_1, f_2?

- Simple approach: L_2 distance, $||f_1 - f_2||$
- can give small distances for ambiguous (incorrect) matches



I_1

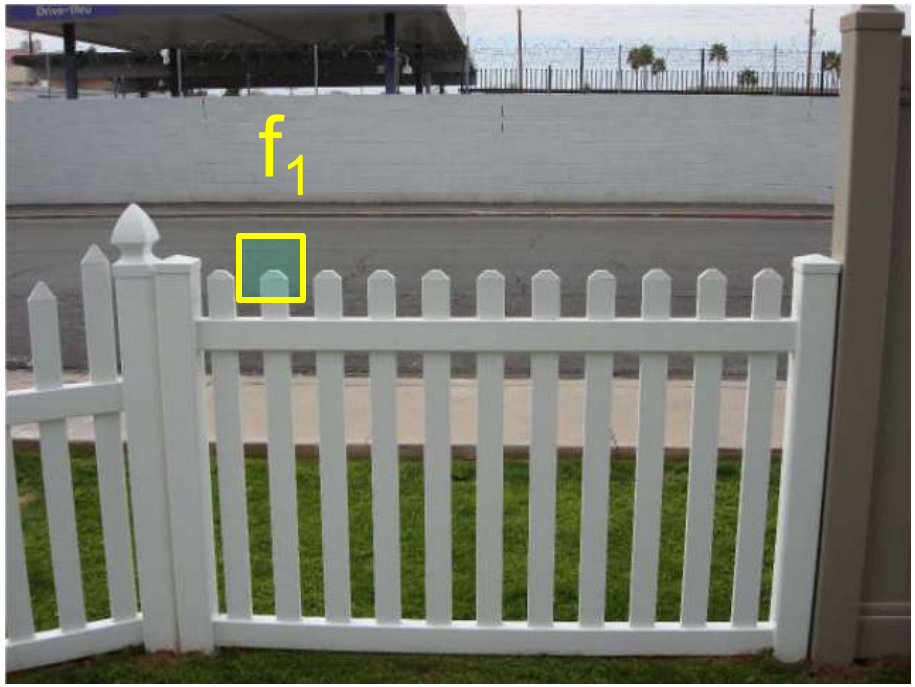


I_2

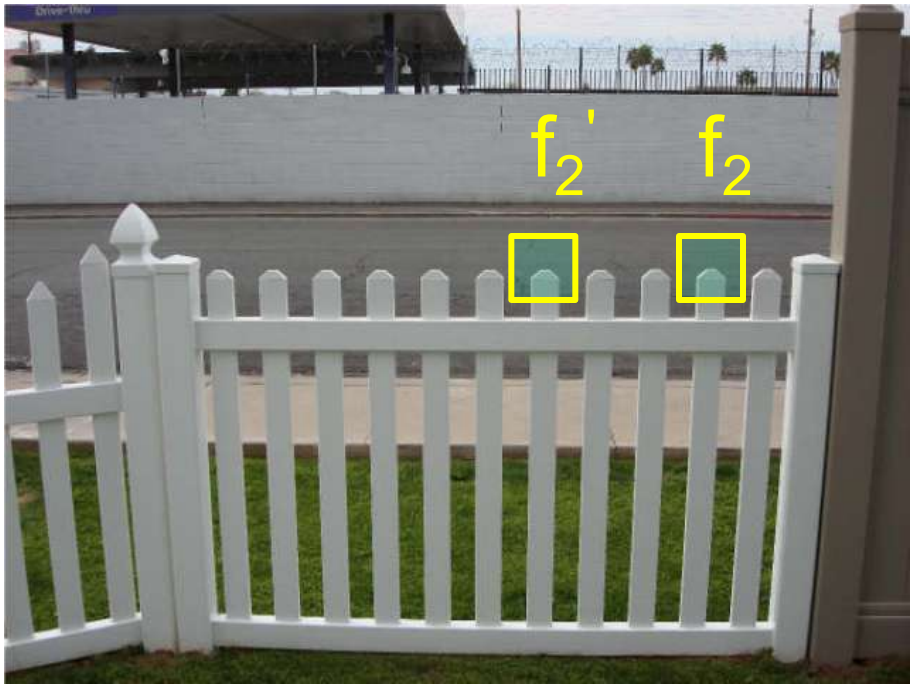
Feature distance

How to define the difference between two features f_1, f_2?

- Better approach: ratio distance = $||f_1 - f_2|| / ||f_1 - f_2'||$
  - f_2 is best SSD match to f_1 in I_2
  - f_2' is 2nd best SSD match to f_1 in I_2
  - gives large values for ambiguous matches

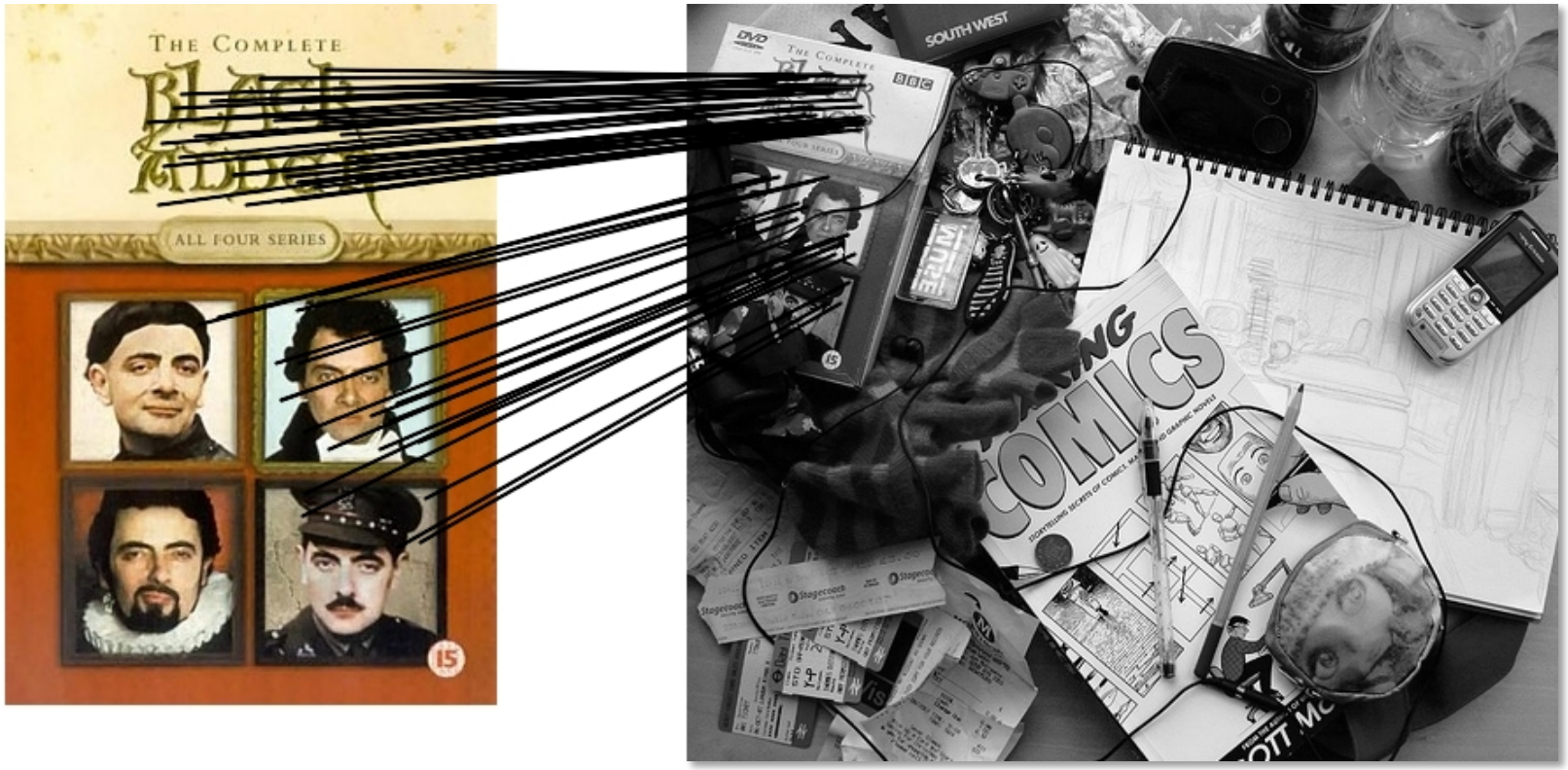


I_1



I_2

Feature matching example



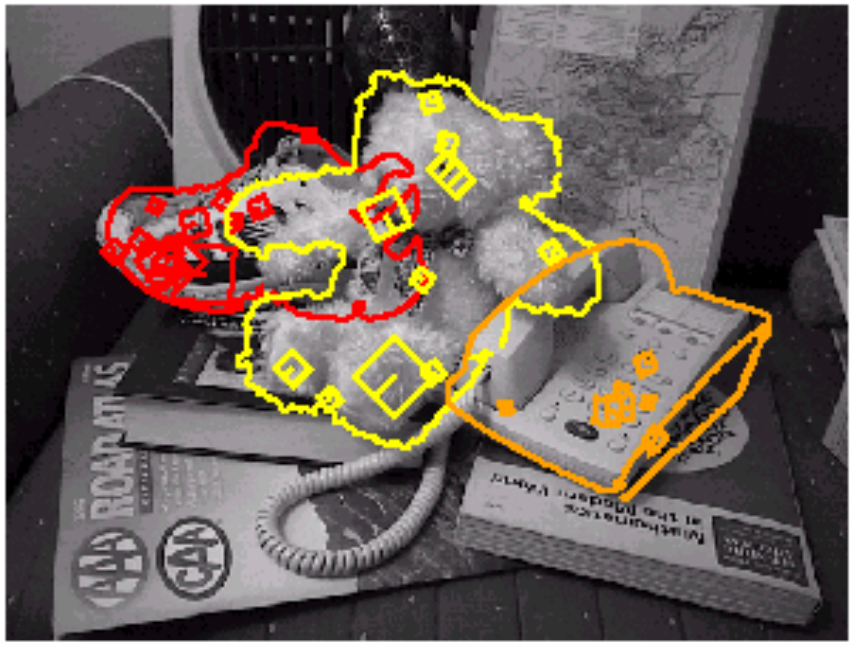
58 matches (thresholded by ratio score)

Lots of applications

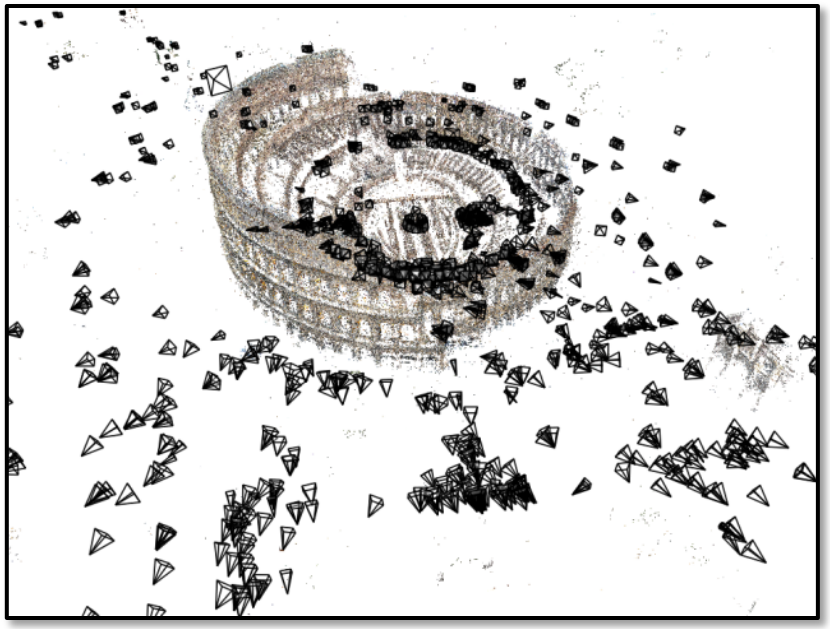
Features are used for:

- Image alignment (e.g., mosaics)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- … other

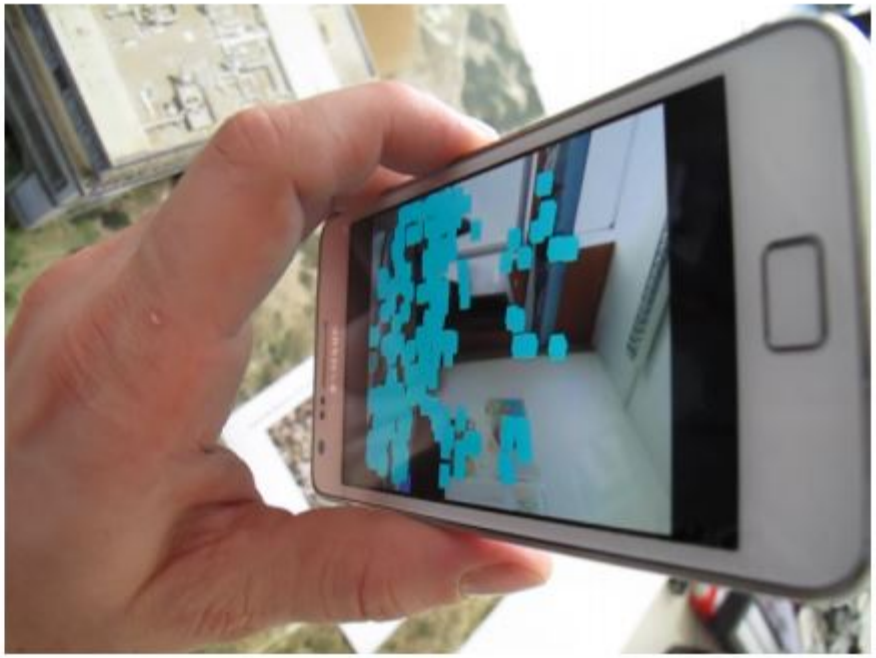
Object recognition



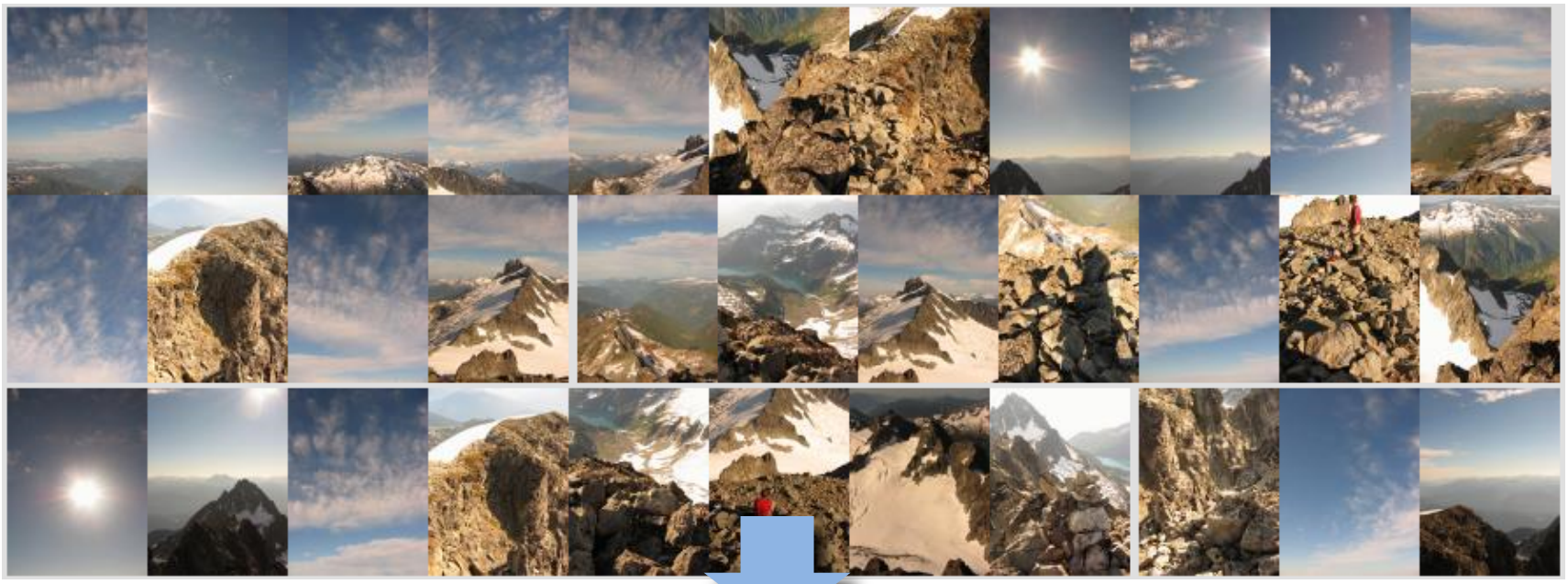
3D Reconstruction

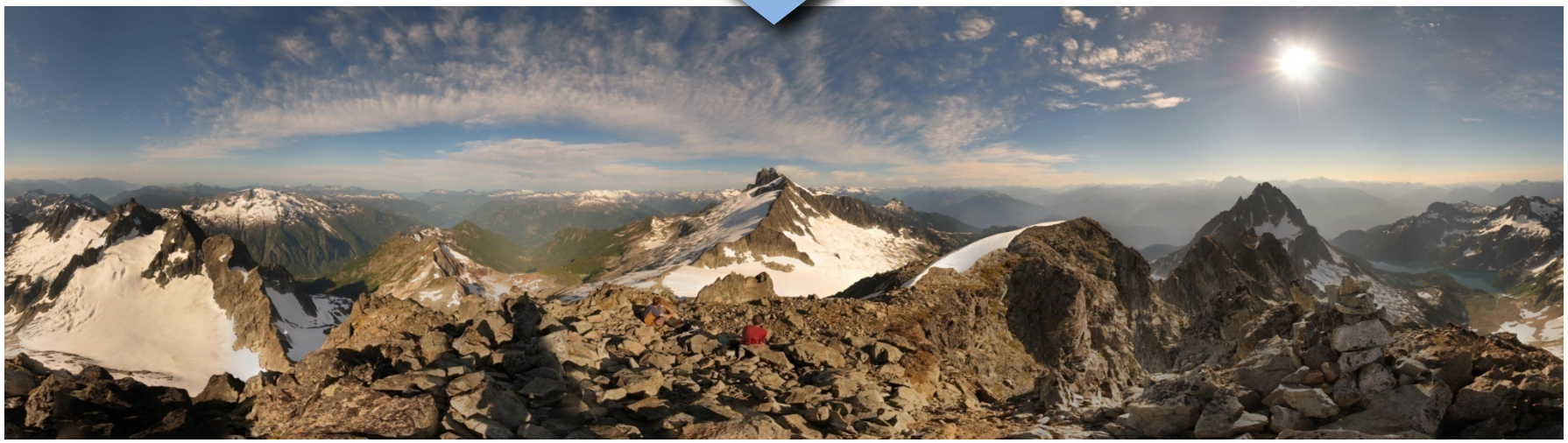


Visual SLAM



Automatic panoramas

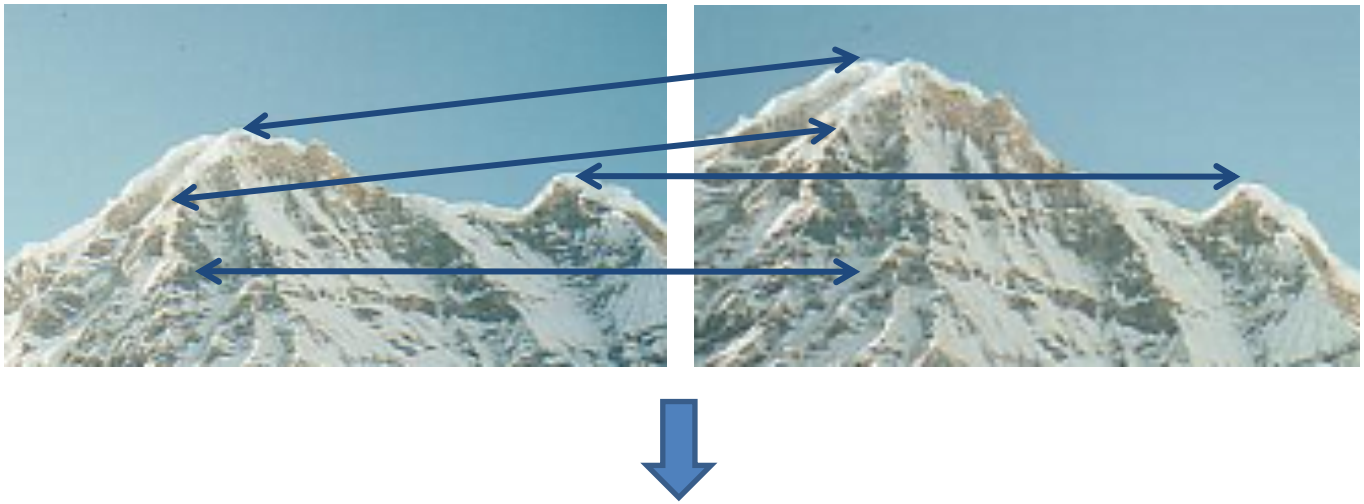




Credit: Matt Brown

Part III
Image Stitching

Image stitching



Computing transformations

Given a set of matches between images A and B

How can we compute the transform T from A to B?

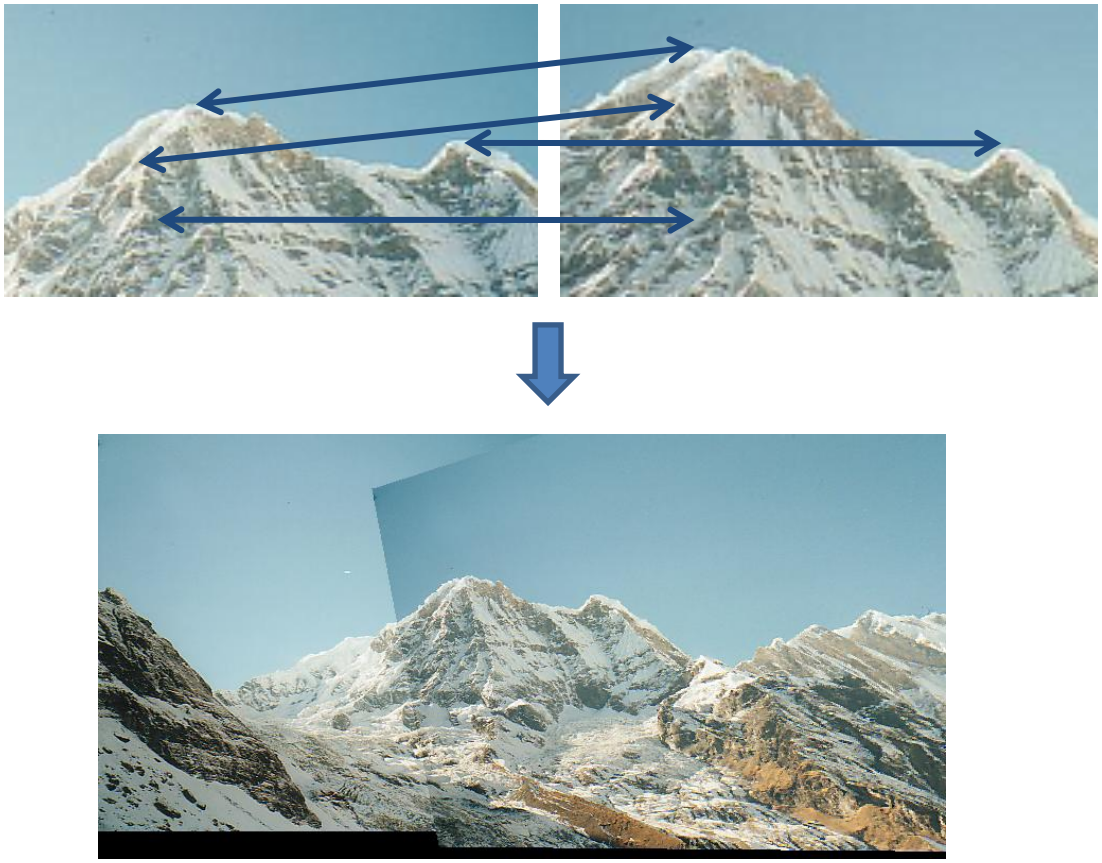
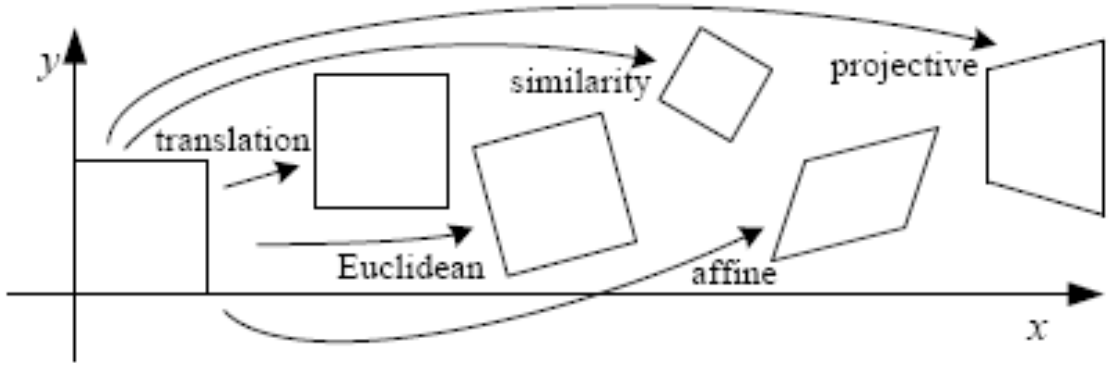
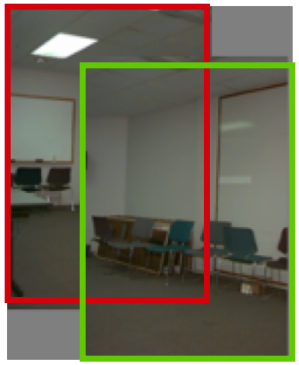


Image transformations

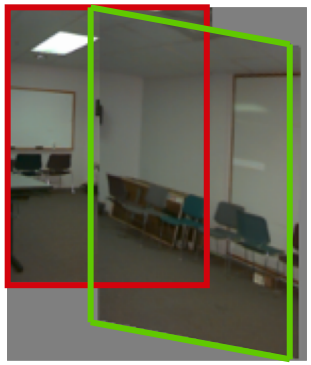


平移变换
Translation



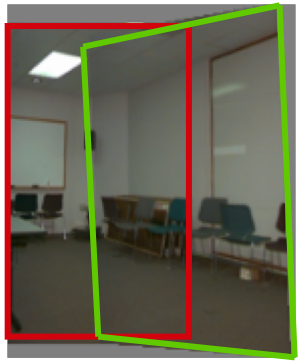
2自由度

仿射变换
Affine



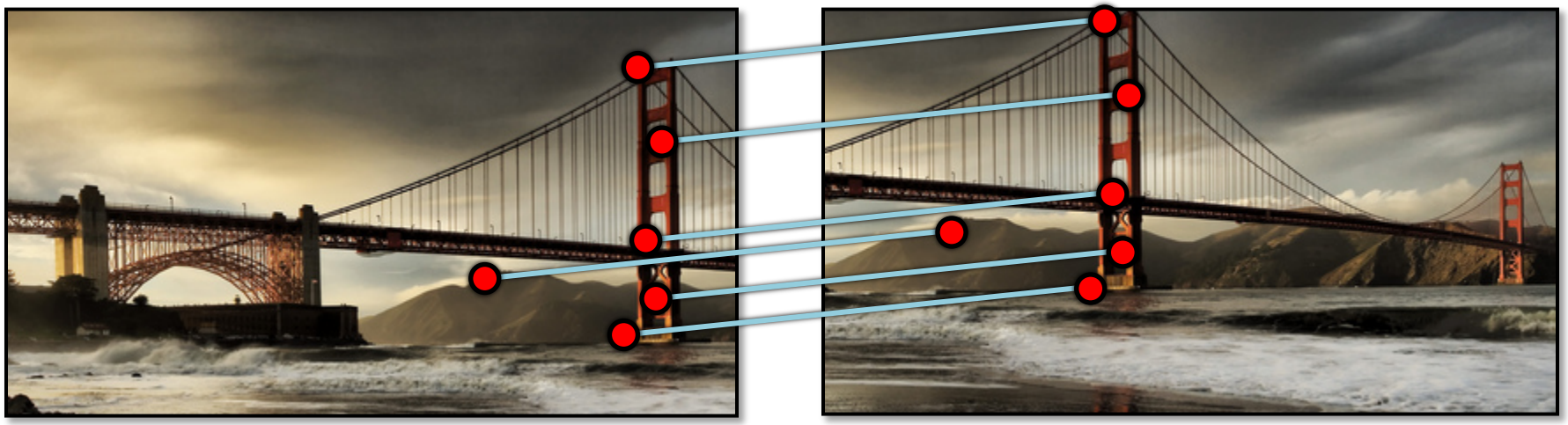
6自由度

透视变换
Projective



8自由度

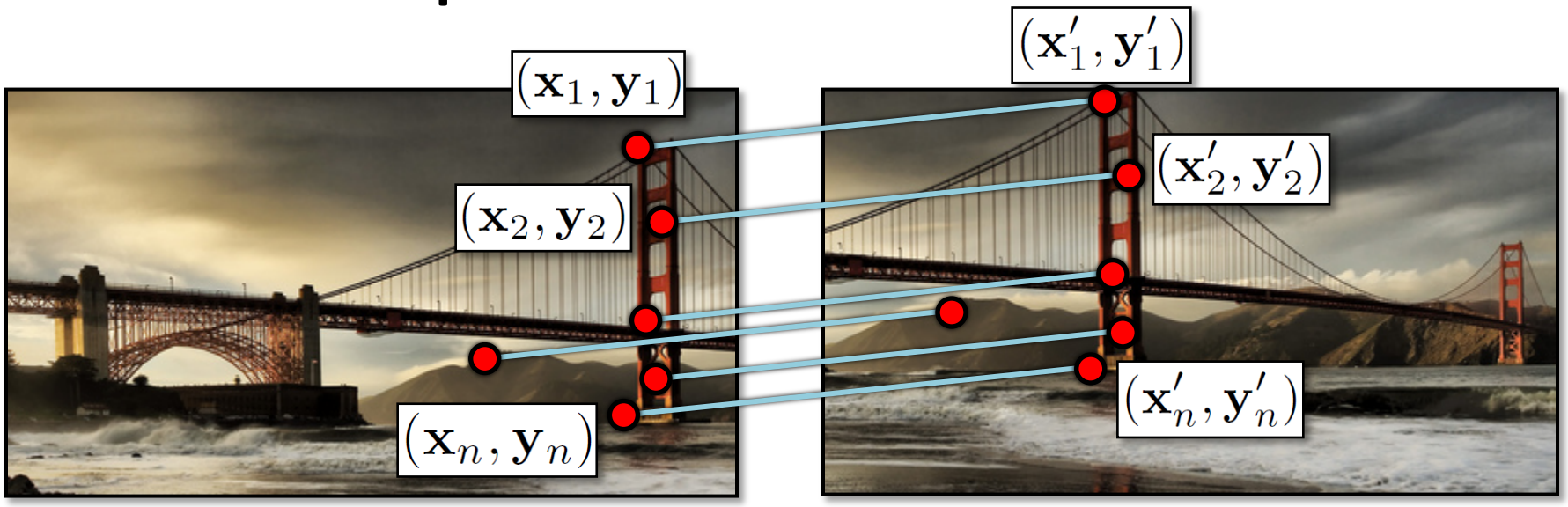
Simple case: translations



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

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

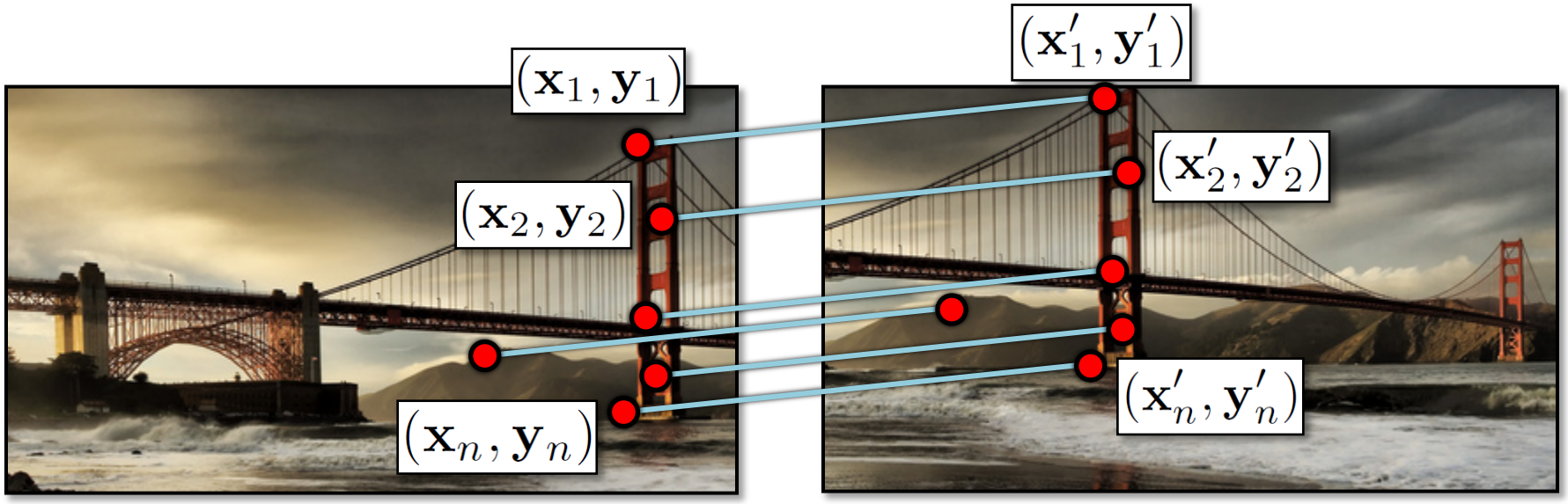
Simple case: translations



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

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

Another view



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

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

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

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 (average) displacement

Least squares formulation

- Can also write as a matrix equation

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

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

Least squares

$$\mathbf{At} = \mathbf{b}$$

- Find **t** that minimizes

$$||\mathbf{At} - \mathbf{b}||^2$$

- To solve, form the *normal equations*

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

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

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?
- How many matches do we need?

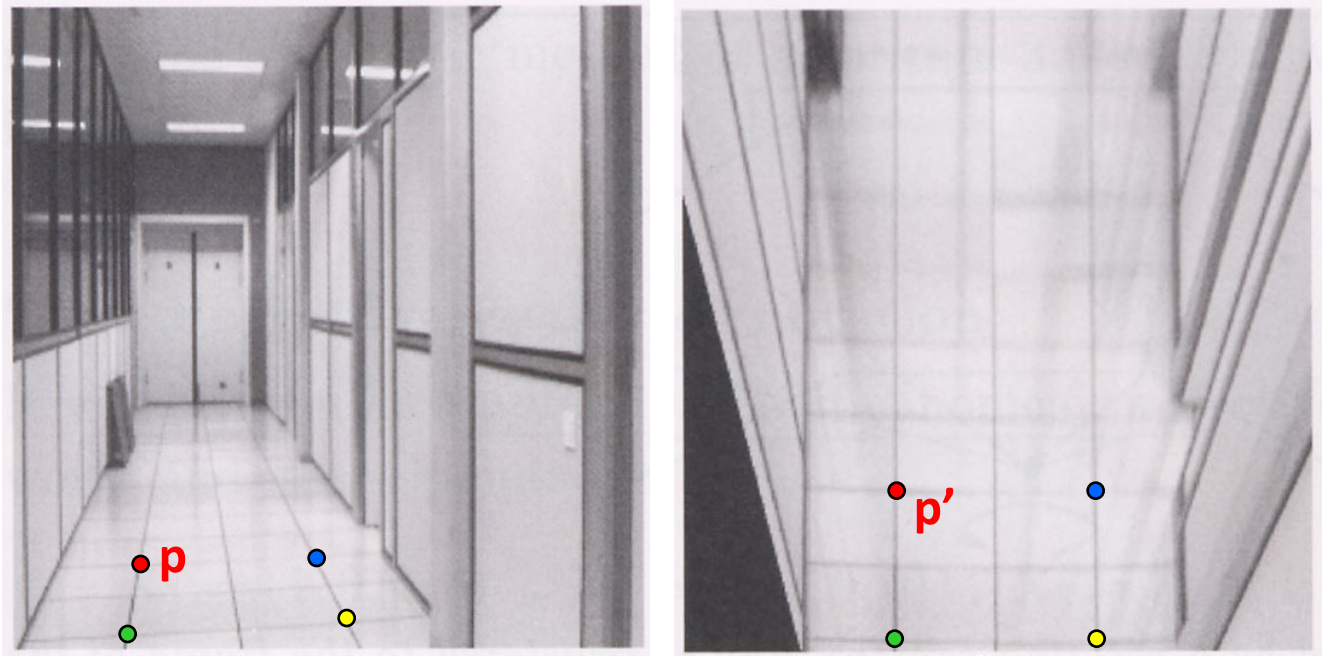
Affine transformations

- Matrix form

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_n & y_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ \vdots \\ x'_n \\ y'_n \end{bmatrix}$$

$$\underset{2n \times 6}{\mathbf{A}} \qquad \underset{6 \times 1}{\mathbf{t}} = \underset{2n \times 1}{\mathbf{b}}$$

Projective transformations



P’ Homography P

$$\begin{pmatrix} x_2 \\ y_2 \\ 1 \end{pmatrix} \sim H_{10} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix} = \begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ 1 \end{pmatrix}$$

Solving for homographies

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

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

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

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

Solving for homographies

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

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

Solving for homographies

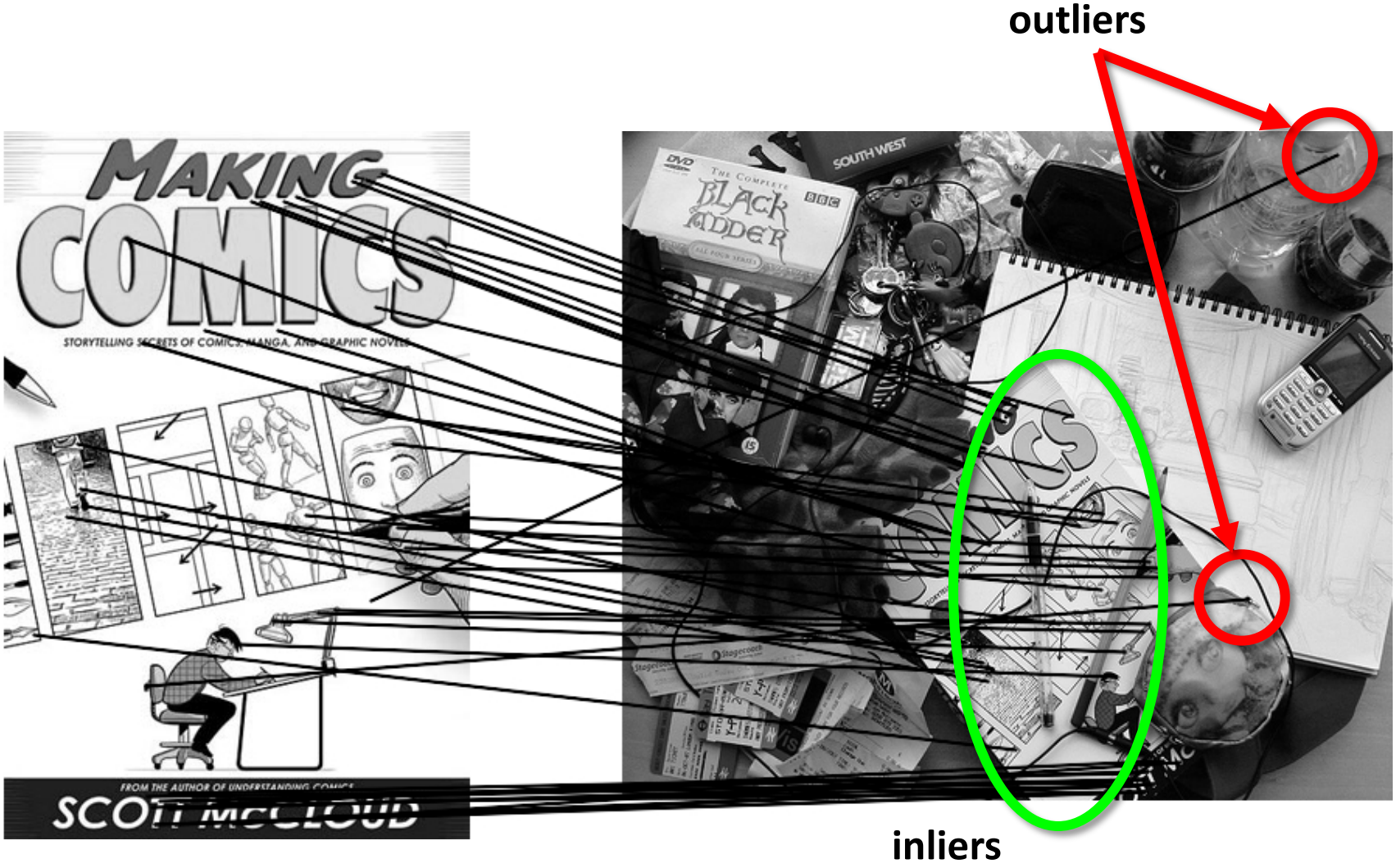
$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ & & & & \vdots & & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_nx_n & -x'_ny_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_nx_n & -y'_ny_n & -y'_n \end{bmatrix} \begin{bmatrix} h_{00} \\ h_{01} \\ h_{02} \\ h_{10} \\ h_{11} \\ h_{12} \\ h_{20} \\ h_{21} \\ h_{22} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$$\underset{2n \times 9}{\mathbf{A}} \qquad \underset{9}{\mathbf{h}} \qquad \underset{2n}{\mathbf{0}}$$

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

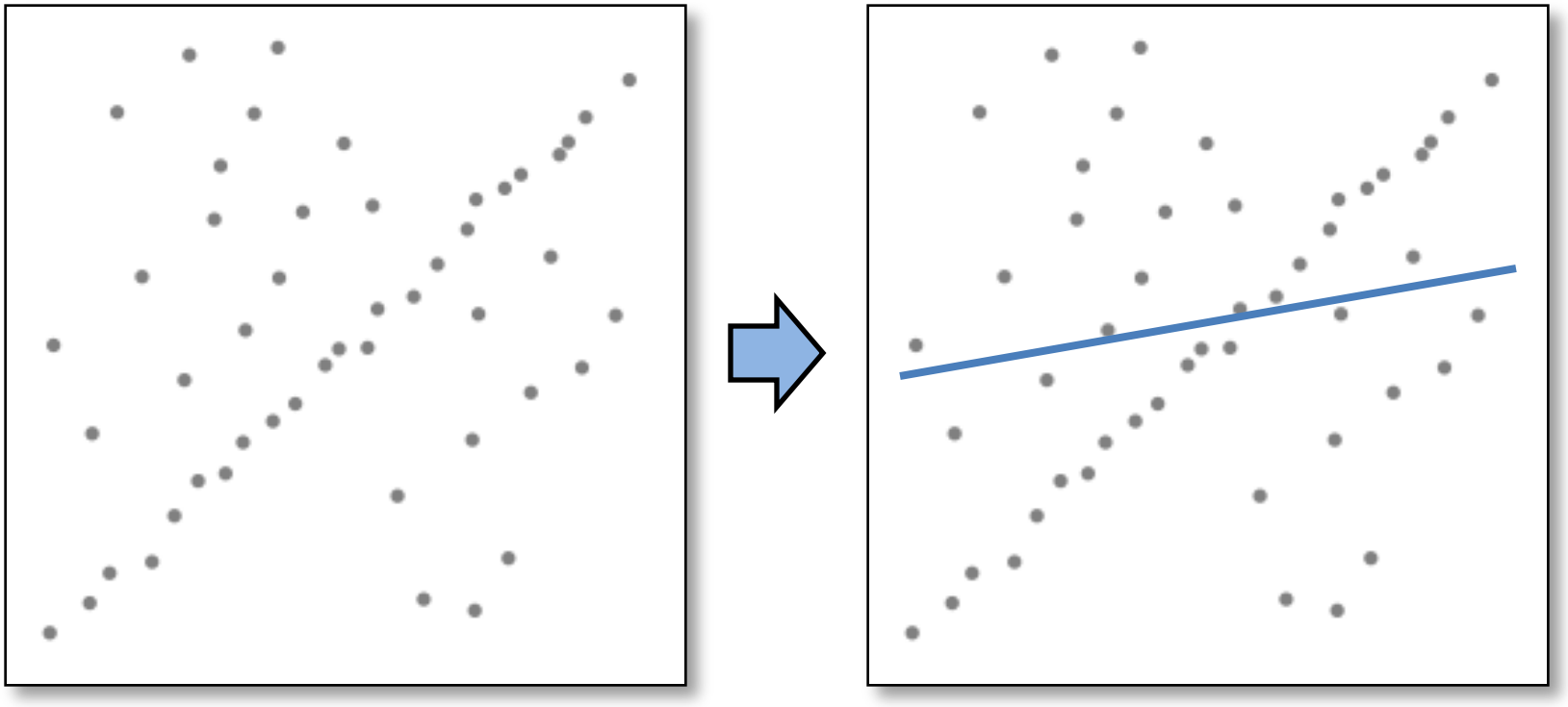
- Since $\mathbf{h}$ is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
- Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue
- Works with 4 or more points

Outliers


outliers
MAKING COMICS
STORYTELLING SECRETS OF COMICS, MANGA, AND GRAPHIC NOVELS
FROM THE AUTHOR OF UNDERSTANDING COMICS
SCOTT McCLOUD
THE COMPLETE BLACK ADDER
BBC
SOUTH WEST
inliers

Robustness

- Let's consider a simpler example… linear regression



Problem: Fit a line to these datapoints

Least squares fit

- How can we fix this?

We need a better cost function…

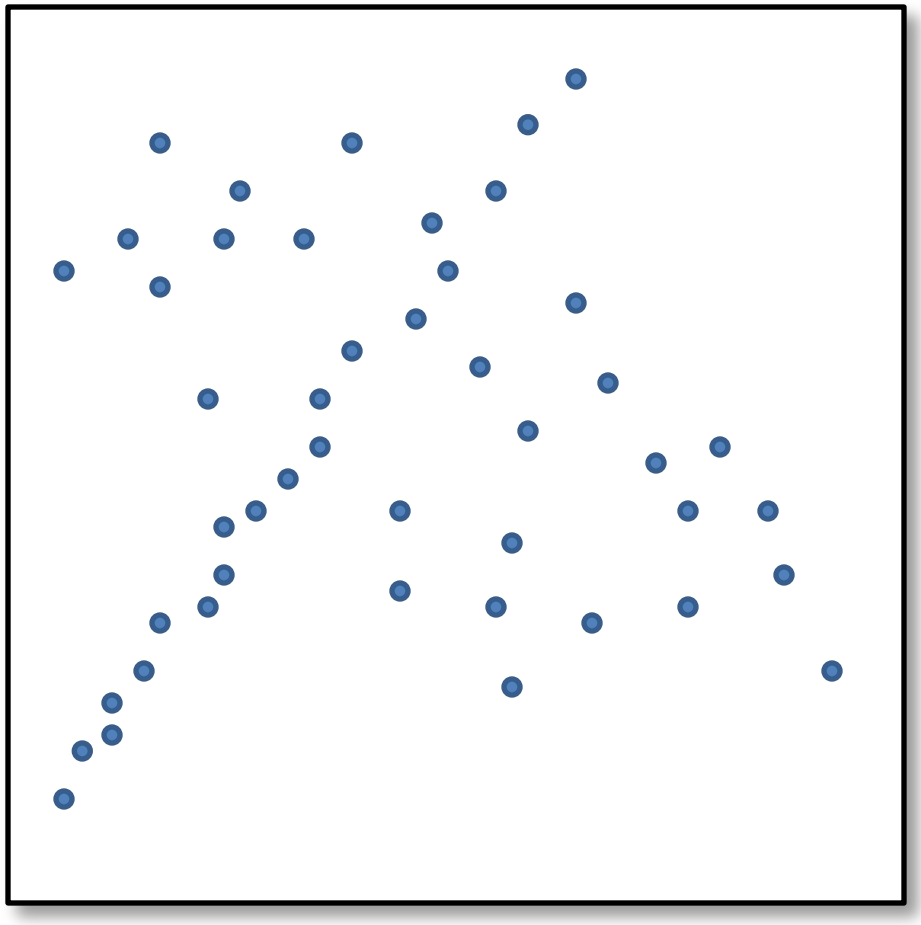
- Suggestions?

Idea

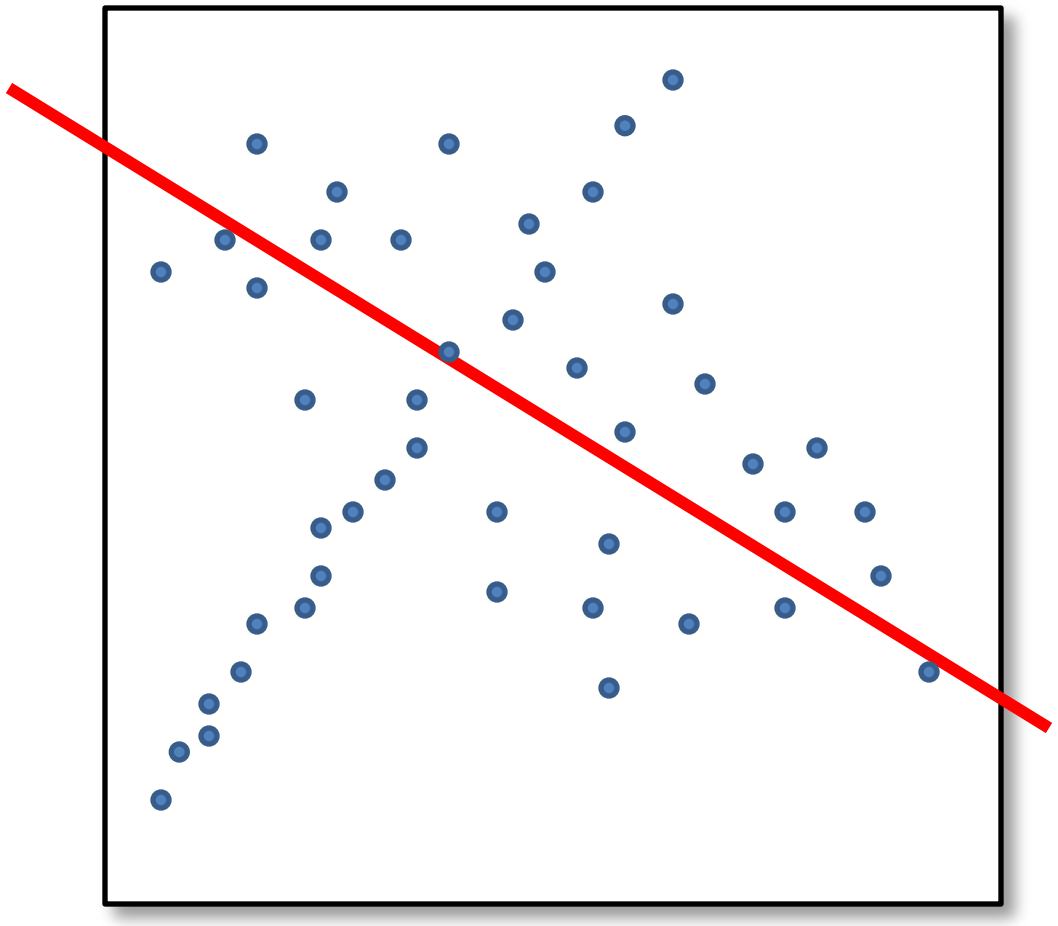
- Given a hypothesized line
- Count the number of points that “agree” with the line
  - “Agree” = within a small distance of the line
  - I.e., the **inliers** to that line

- For all possible lines, select the one with the largest number of inliers

Counting inliers

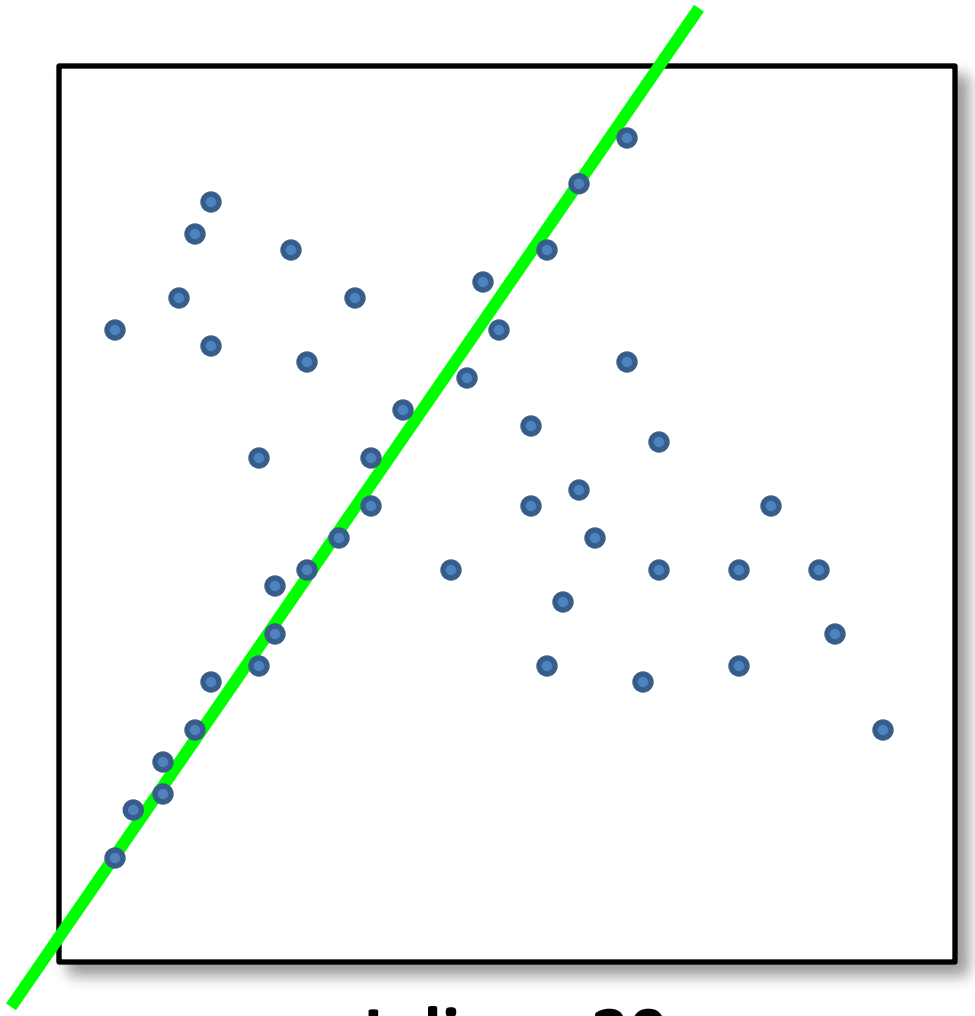


Counting inliers



Inliers: 3

Counting inliers



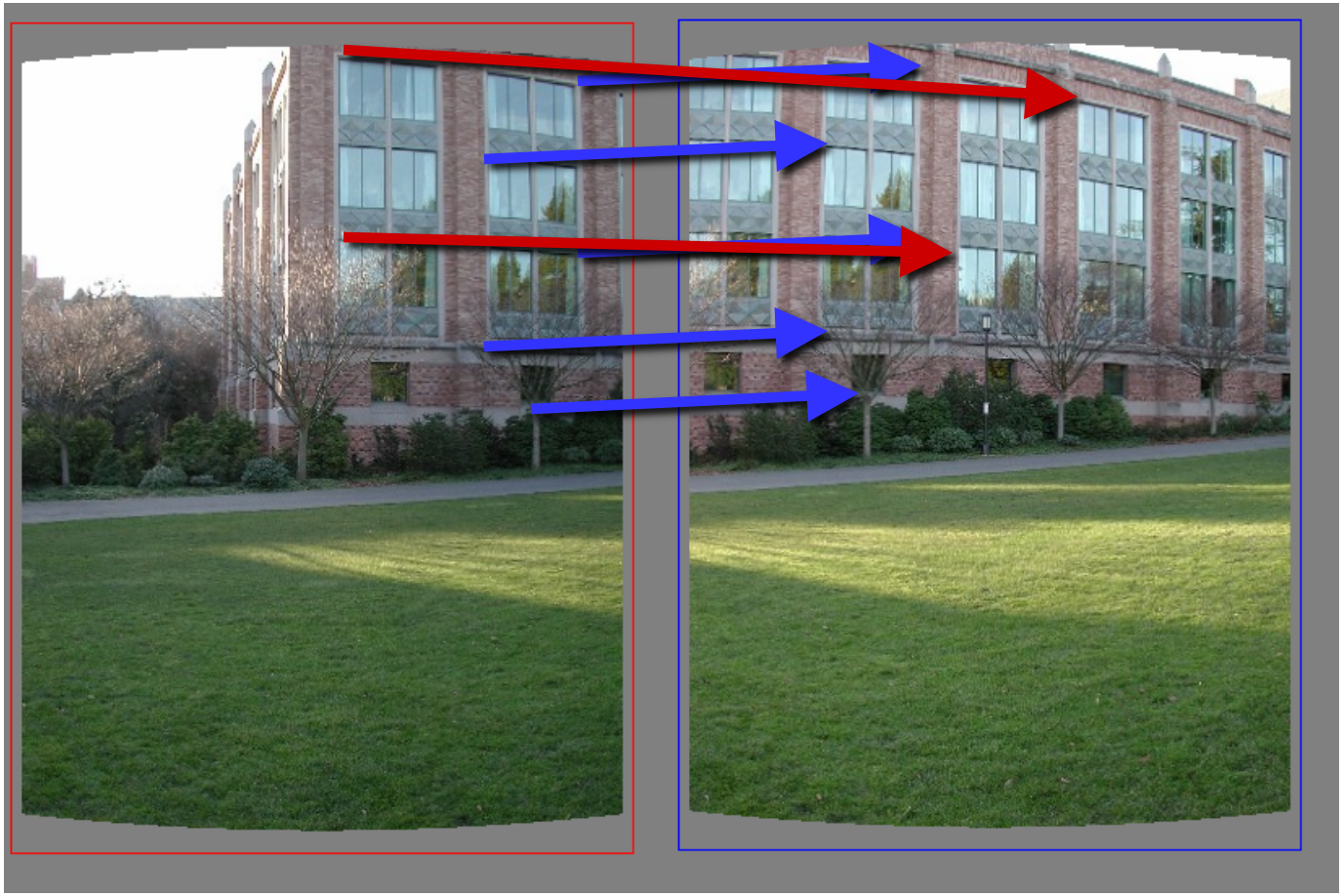
Inliers: 20

How do we find the best line?

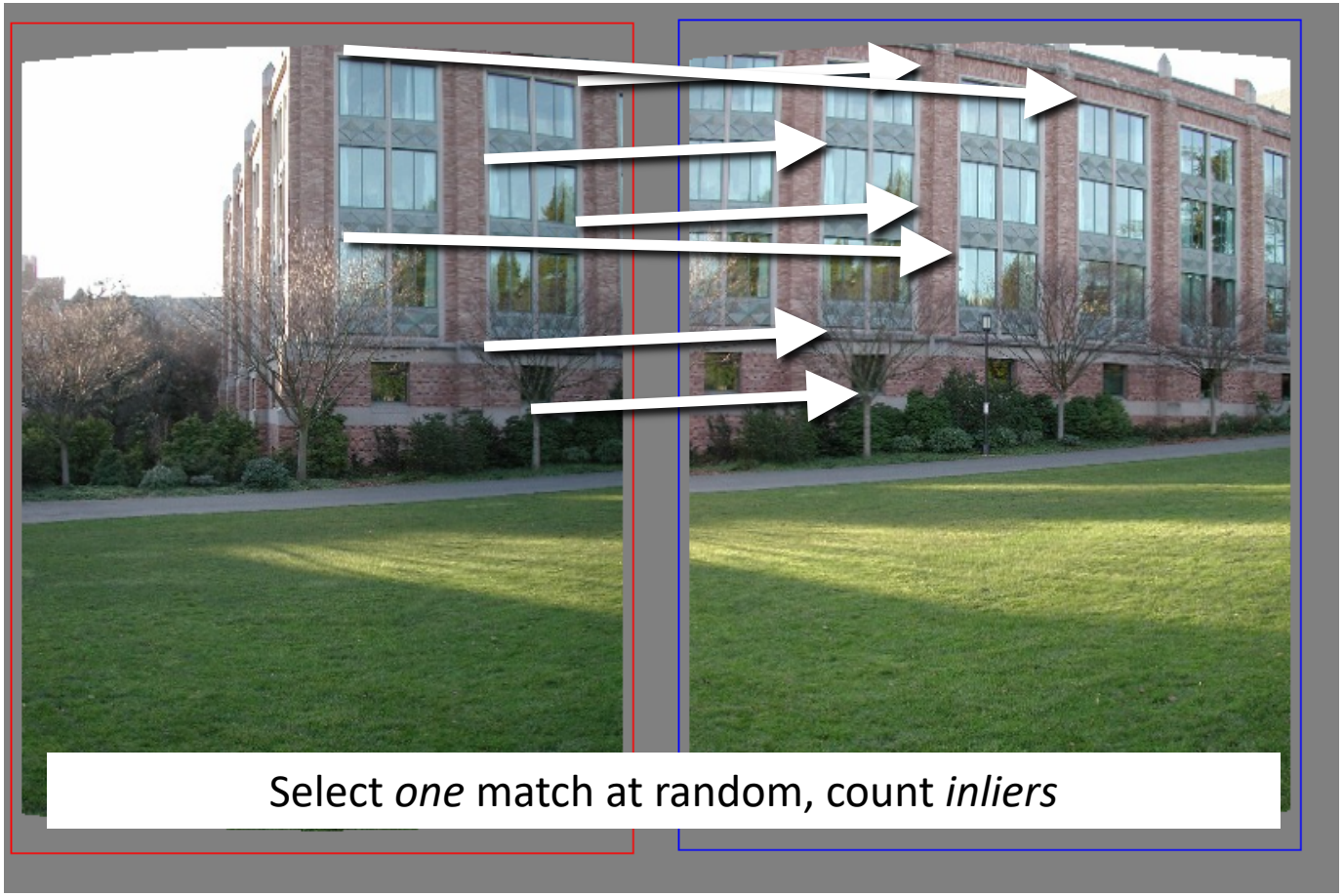
- Unlike least-squares, no simple closed-form solution

- Hypothesize-and-test
  - Try out many lines, keep the best one
  - Which lines?

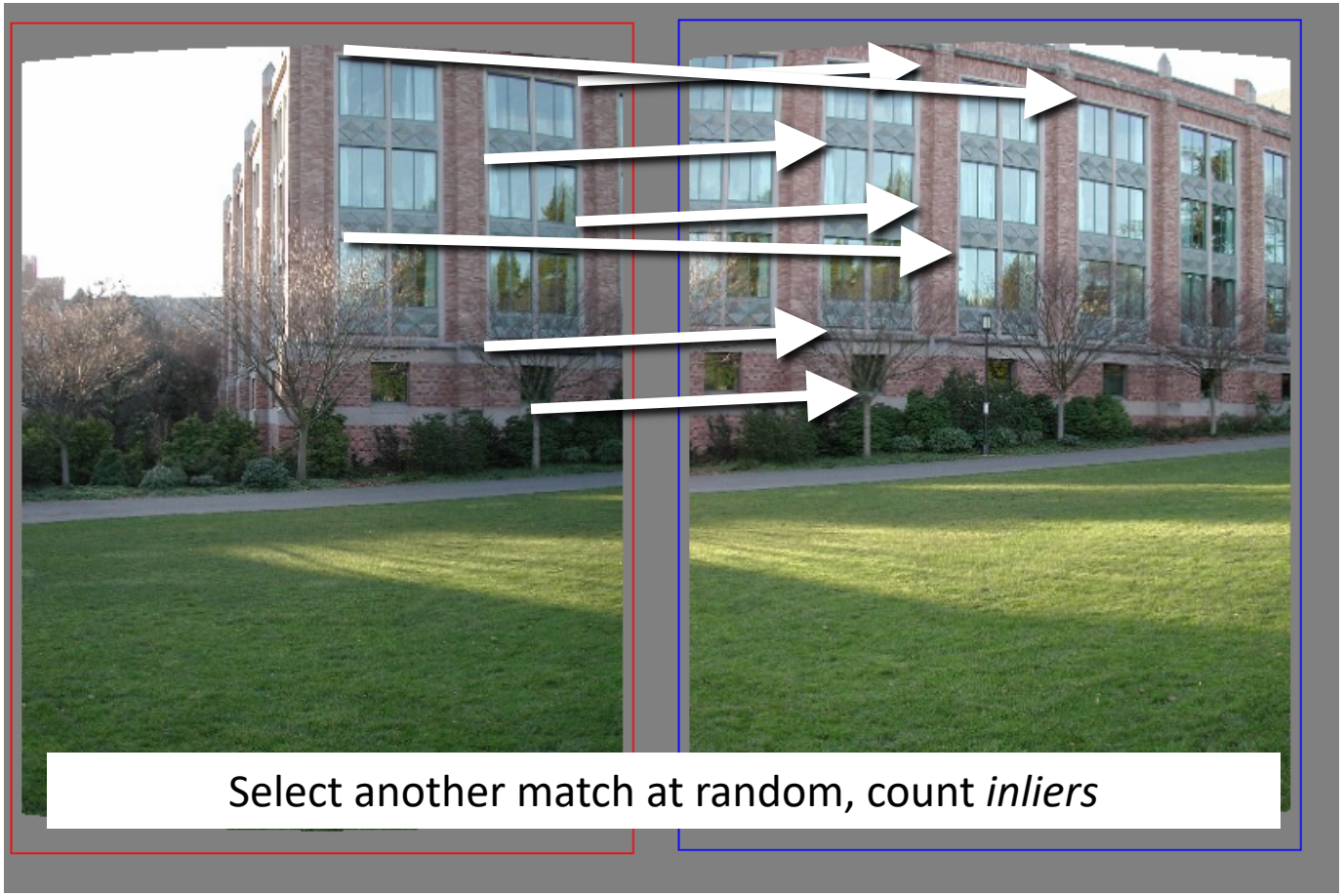
Translations



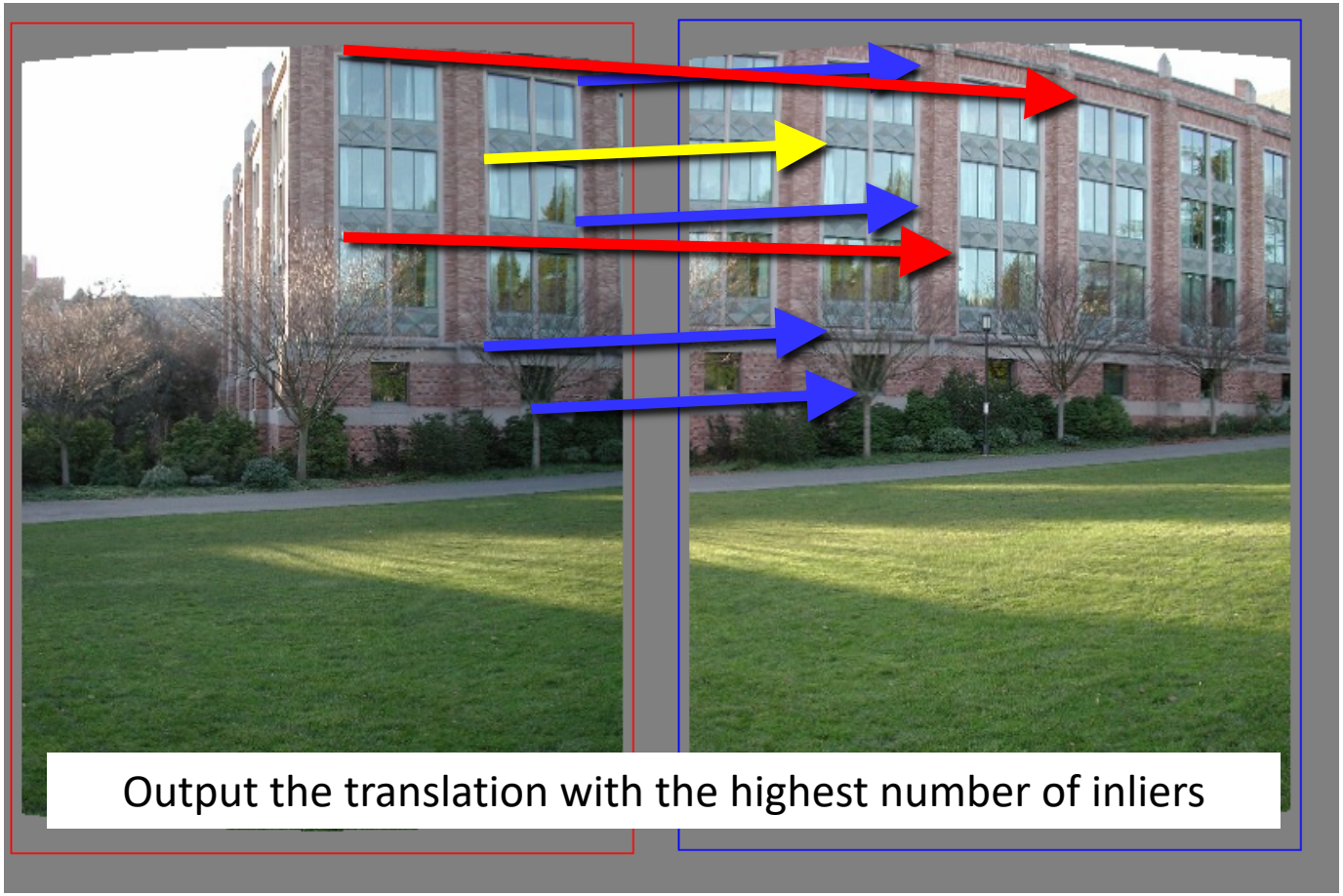
RAndom SAmple Consensus



RAndom SAmple Consensus



RAndom SAmple Consensus



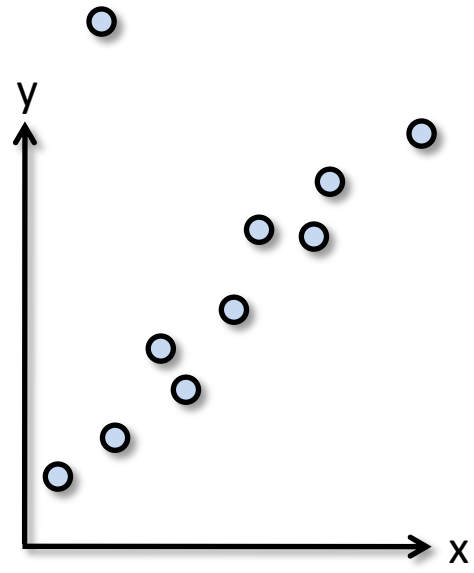
RANSAC

- Idea:
  - All the inliers will agree with each other on the translation vector; the (hopefully small) number of outliers will (hopefully) disagree with each other
    - RANSAC only has guarantees if there are < 50% outliers
  - "All good matches are alike; every bad match is bad in its own way."

    – Tolstoy via Alyosha Efros

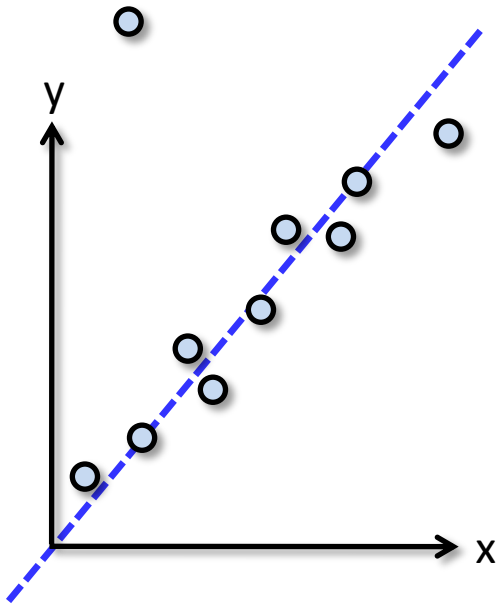
RANSAC

- Back to linear regression
- How do we generate a hypothesis?



RANSAC

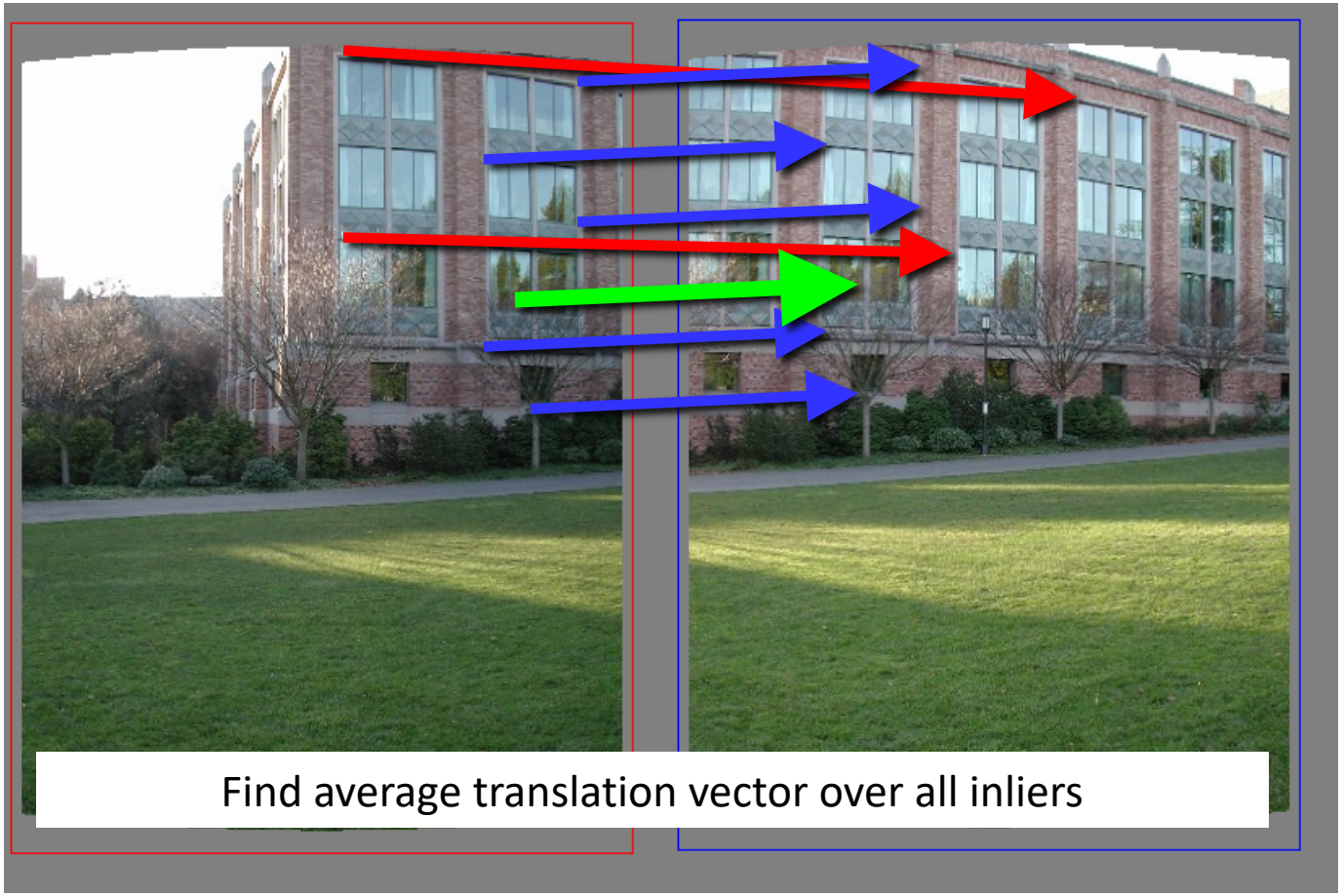
- Back to linear regression
- How do we generate a hypothesis?



RANSAC

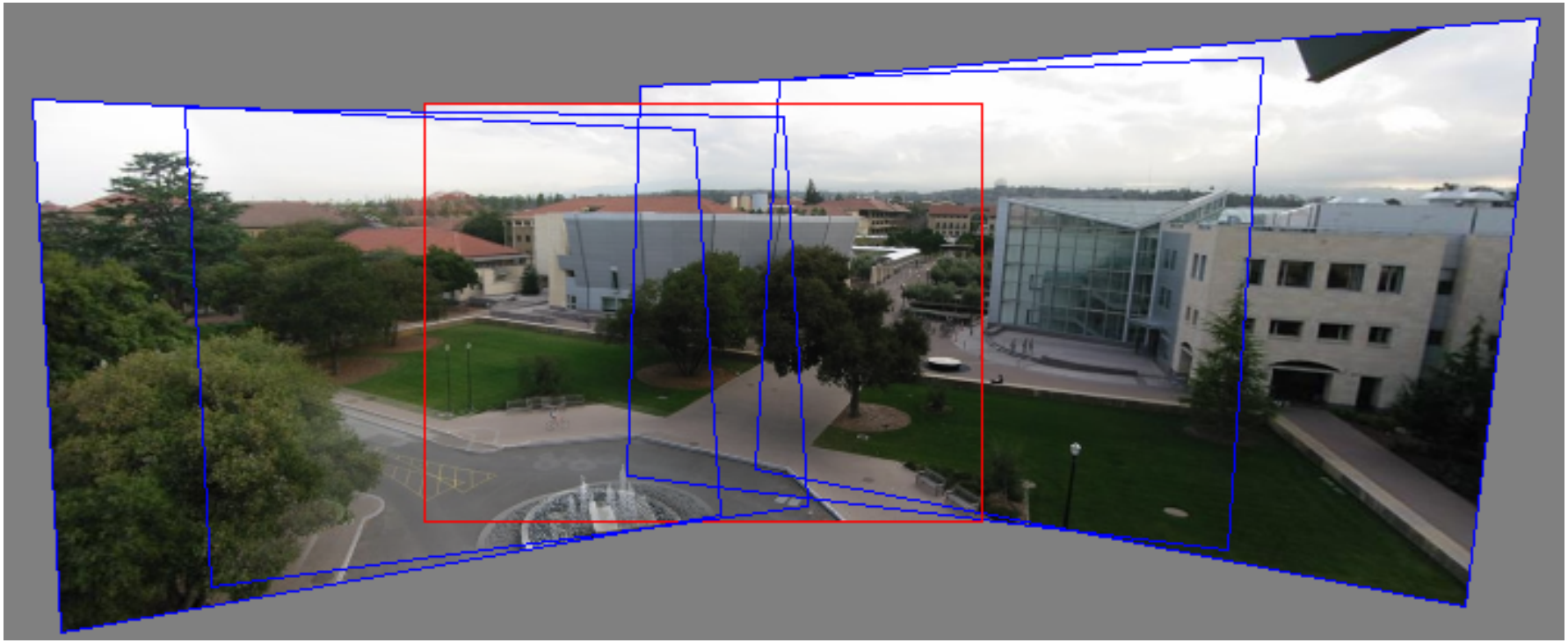
- General version:
  1. Randomly choose *s* samples
     - Typically *s* = minimum sample size that lets you fit a model
  2. Fit a model (e.g., line) to those samples
  3. Count the number of inliers that approximately fit the model
  4. Repeat *N* times
  5. Choose the model that has the largest set of inliers

Final step: least squares fit



Panoramas

- Now we know how to create panoramas!



Panoramas

- Now we know how to create panoramas!

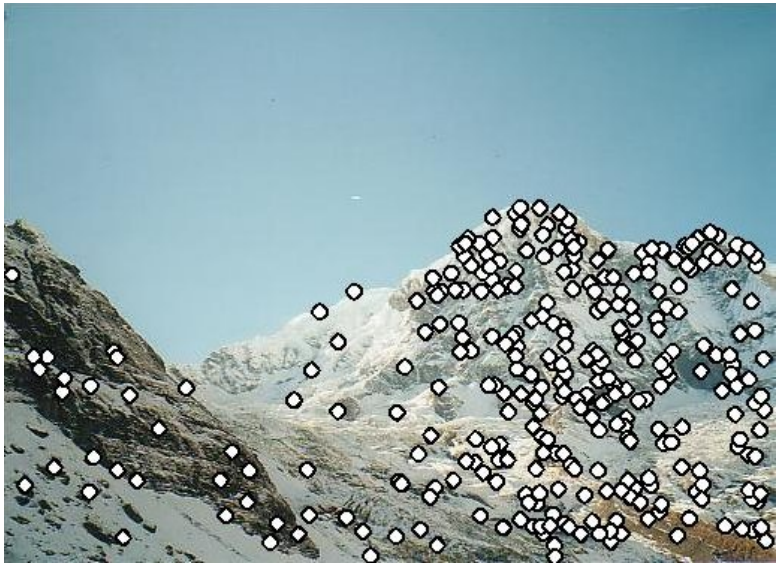
- Given two images:
  - Step 1: Detect features
  - Step 2: Match features
  - Step 3: Compute a homography using RANSAC
  - Step 4: Warp and combine the images together

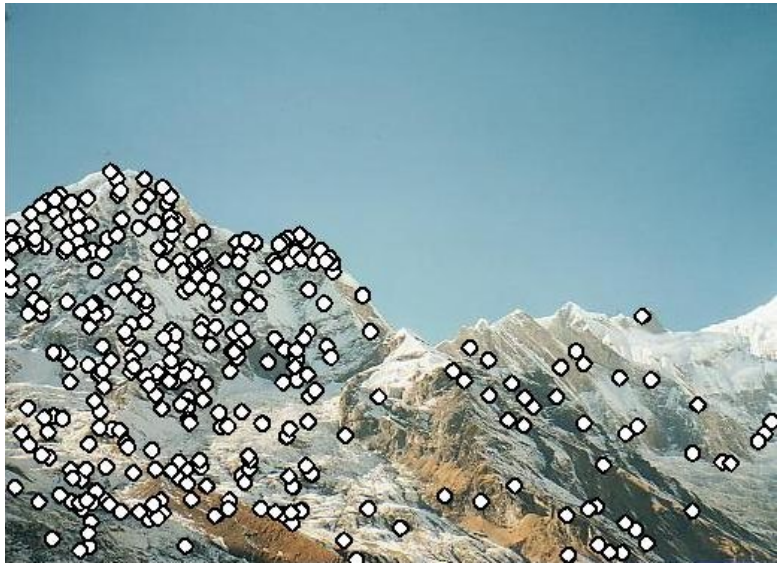
- Repeat if there are more images

图像拼接

输入图像

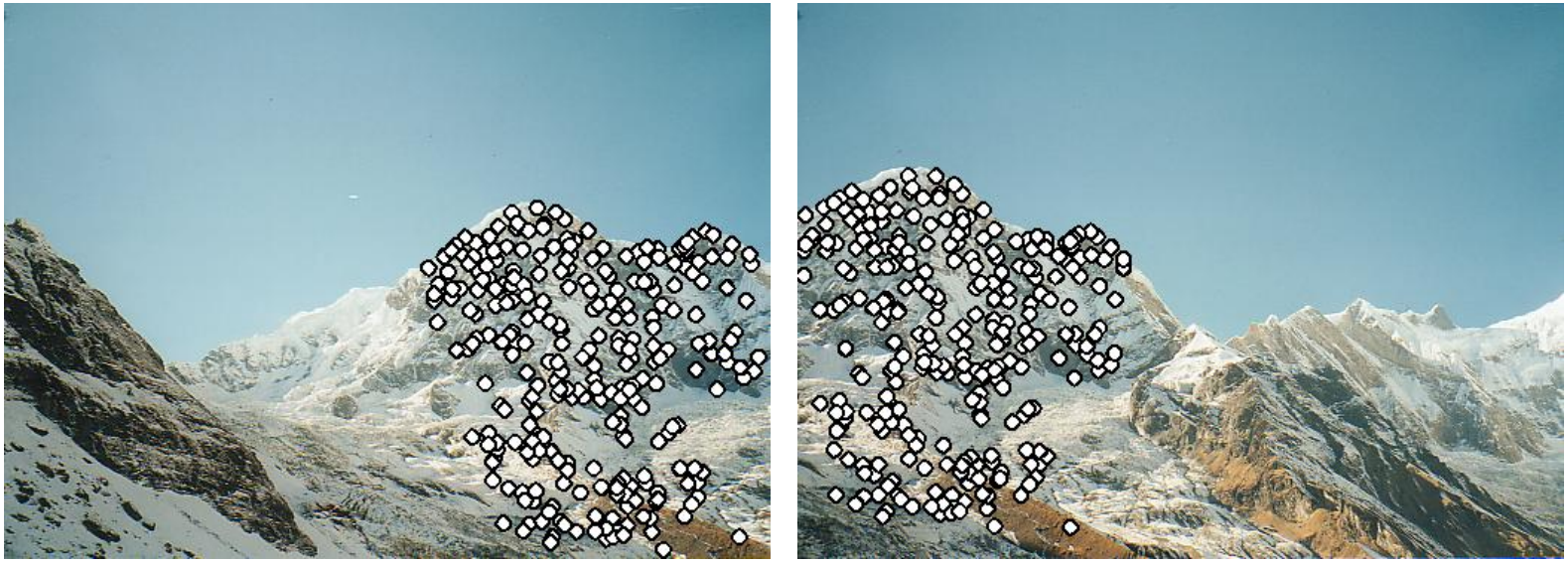
图像拼接





特征匹配

图像拼接



RANSAC剔除outlier

图像拼接

Homography进行变换，固定第一幅图

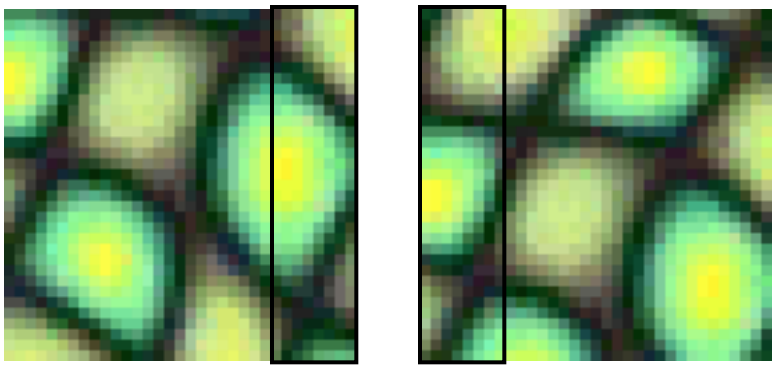
无缝融合

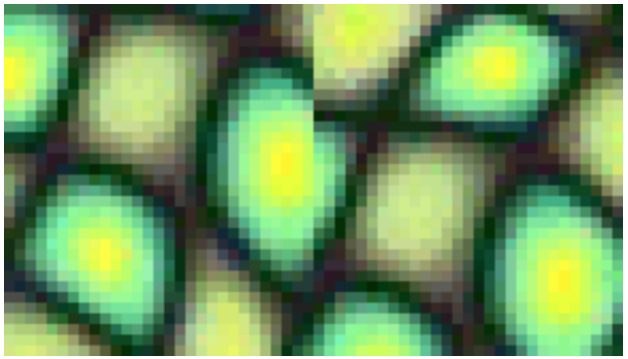
- Graphcut
- Poisson Image Editing

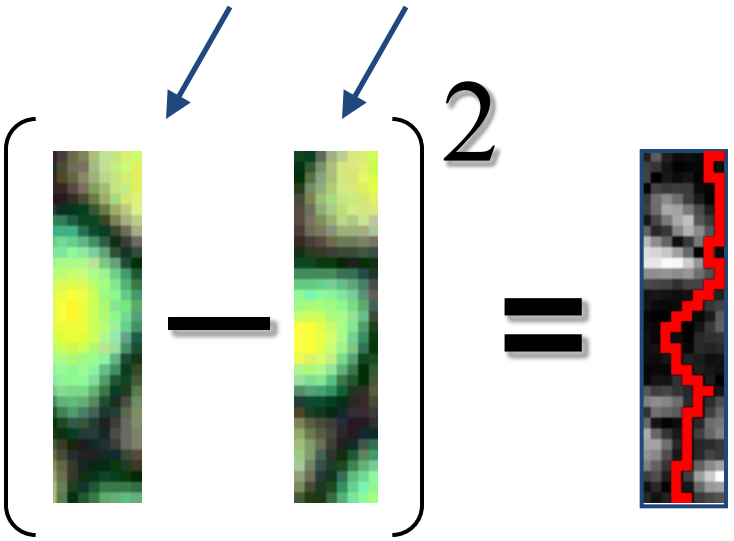
无缝融合

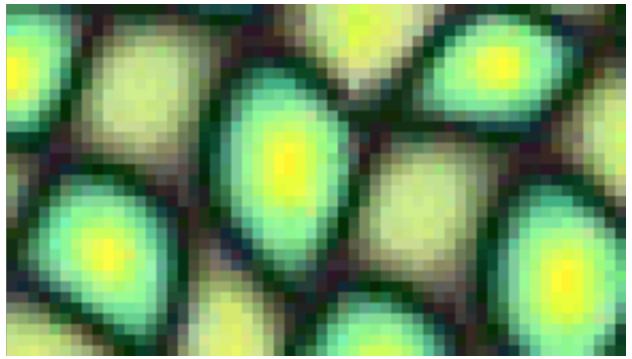
重叠的图像

简单的接缝





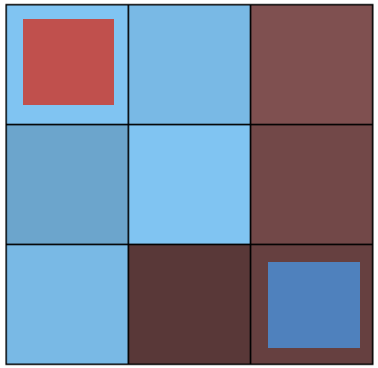


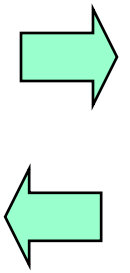


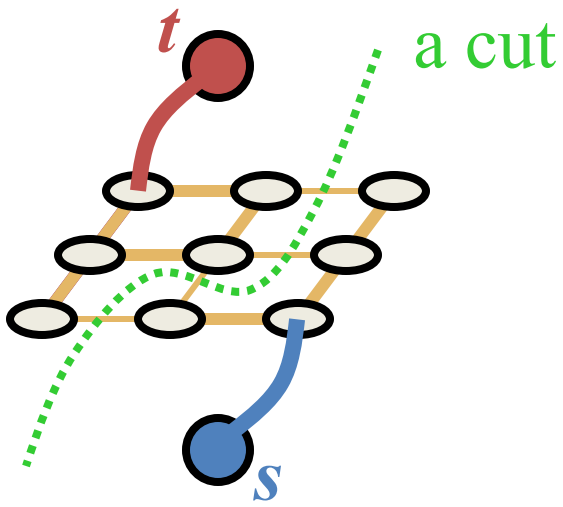
颜色差

最小化颜色差的接缝

无缝融合





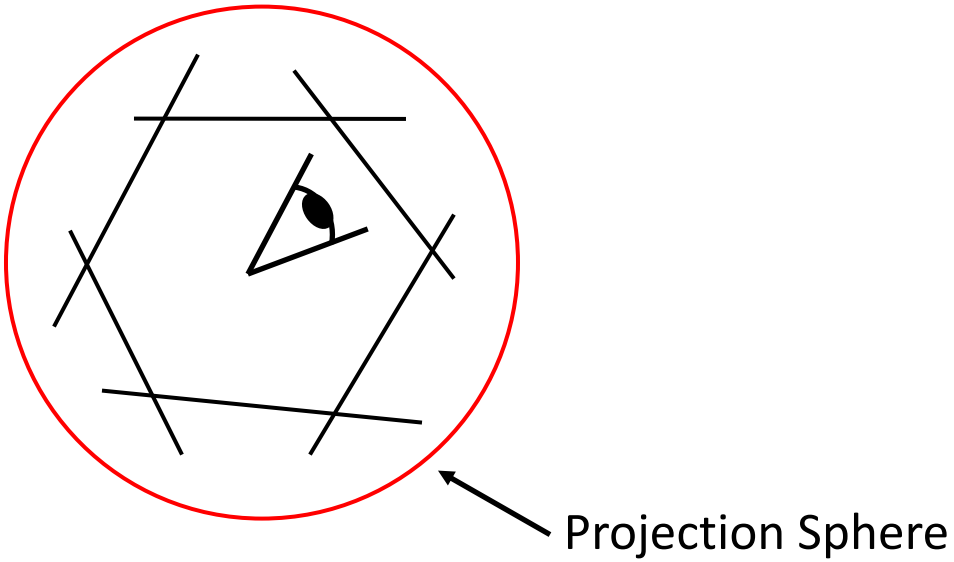


最大流最小割算法

多项式时间

Spherical panoramas

Spherical panoramas



- Map image to spherical coordinates (spherical images)
- Camera rotation = image translation on sphere
- We can align spherical images by translation

Questions?