Lecture 20: Fitting geometric models

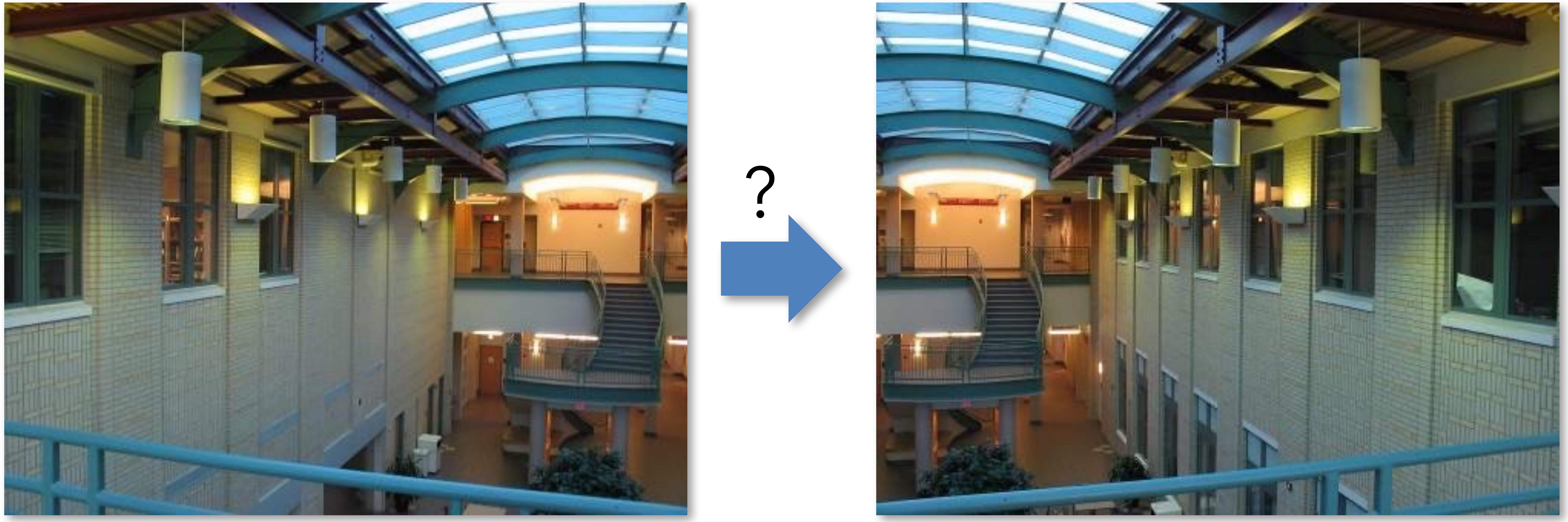
CS 5670: Introduction to Computer Vision

Most slides from Noah Snavely

Today

- Finish discussion of two-view geometry
- Finding correspondences
- Fitting a homography
- RANSAC

Recall: What is the geometric relationship between these two images?



Recall: Image alignment

Why don't these image line up exactly?

Source: N. Snavely

Recall: affine transformations

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

what happens when we change this row?

affine transformation

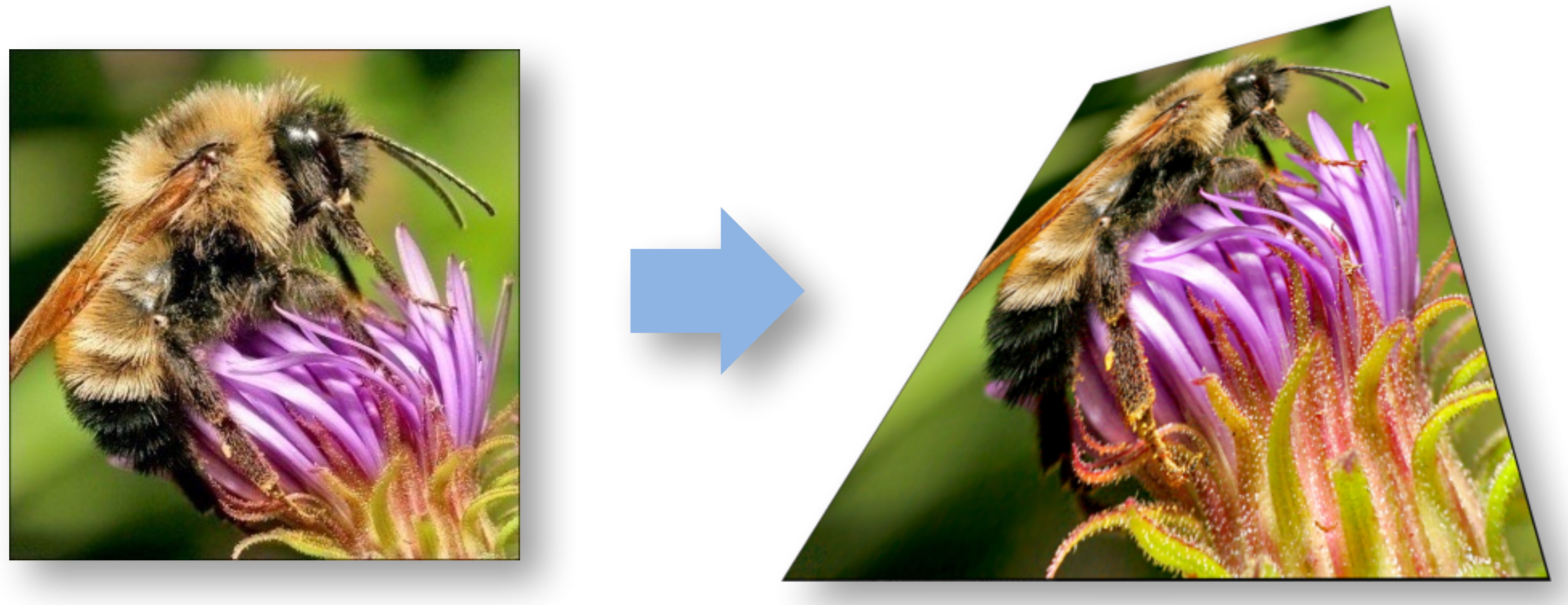
Source: N. Snavely

Recall: Projective Transformations aka Homographies aka Planar Perspective Maps

$$\mathbf{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

Called a **homography**

(or planar perspective map)



Source: N. Snavely

Homographies

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

Note that this can be 0!
A "point at infinity"

$$\sim \begin{bmatrix} \frac{ax+by+c}{gx+hy+1} \\ \frac{dx+ey+f}{gx+hy+1} \\ 1 \end{bmatrix}$$

Source: N. Snavely

Homography

Example: two pictures taken by rotating the camera:

If we try to build a panorama by overlapping them:

Source: Torralba, Isola, Freeman

Homography

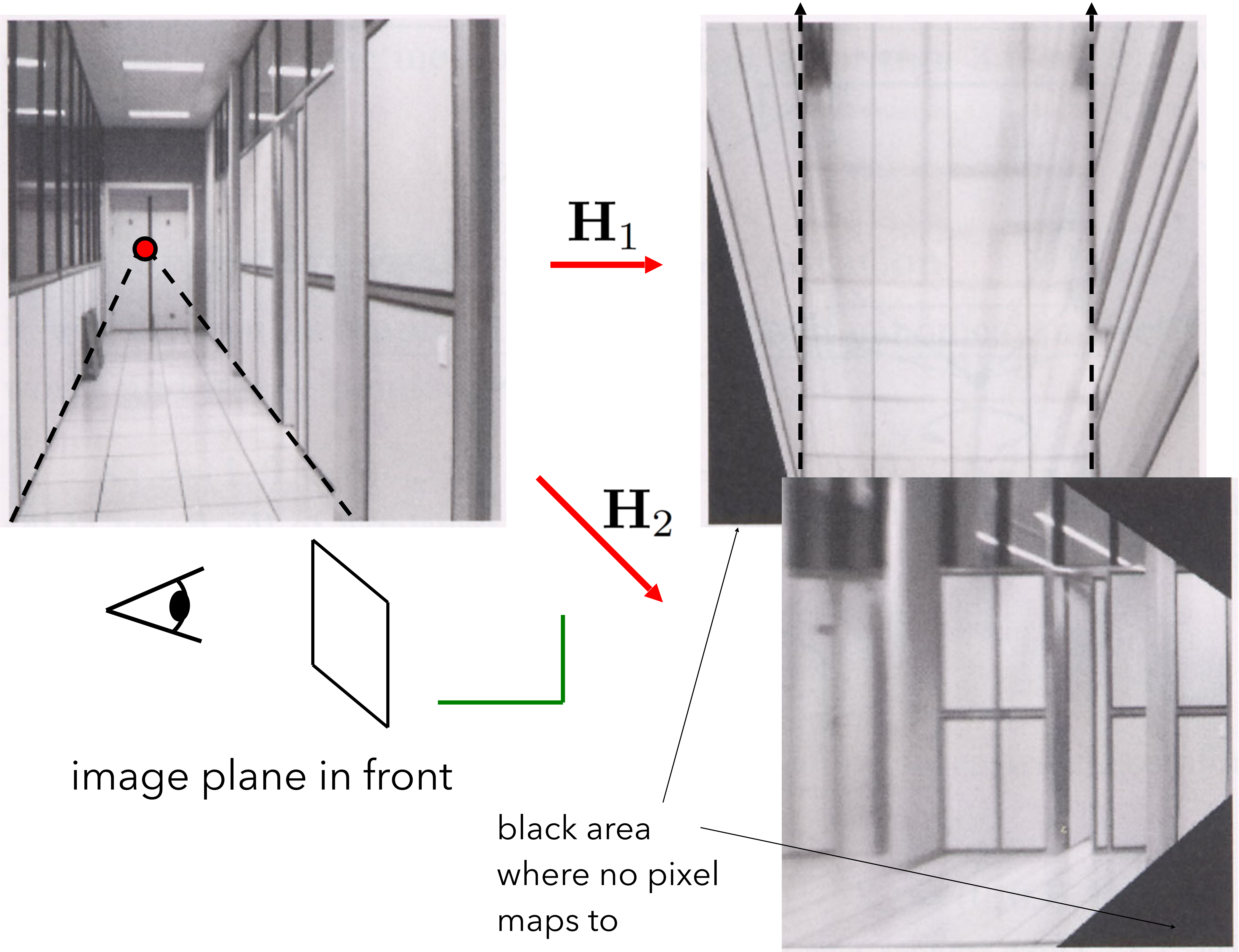
Example: two pictures taken by rotating the camera:

With a homography, you can map both images into a single camera:

We'll see why in PS5!

Source: Torralba, Isola, Freeman

Plane-to-plane homography



Source: N. Snavely

Homographies

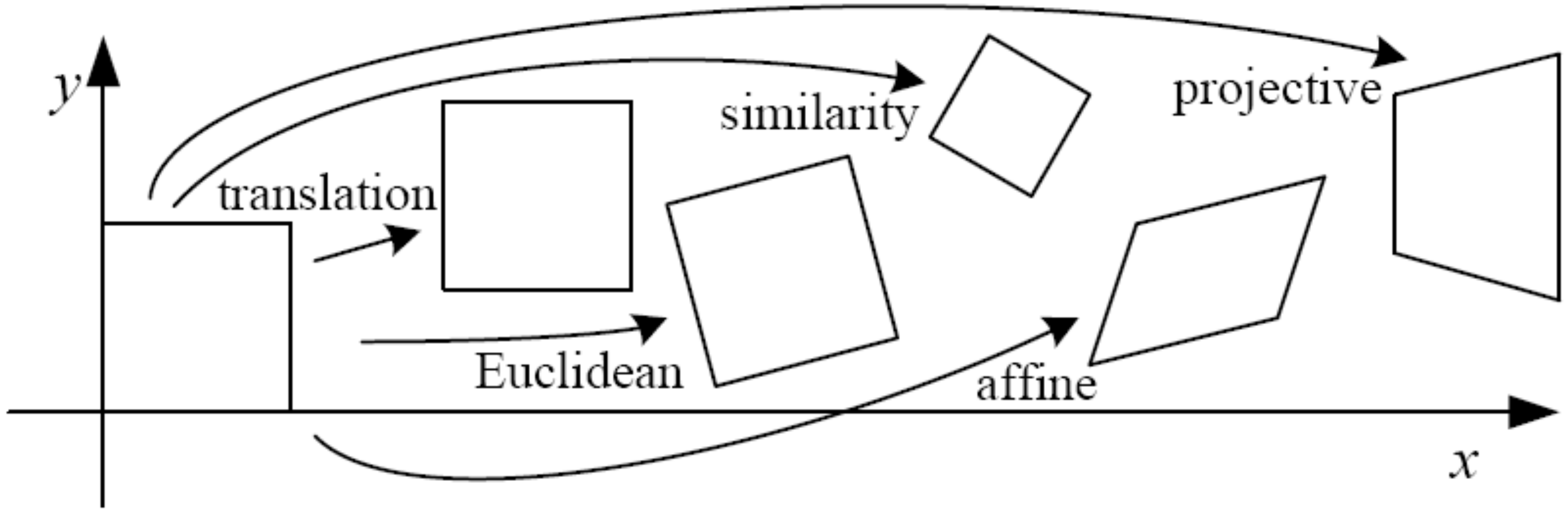
- Homographies …
  - Affine transformations, and
  - Projective warps

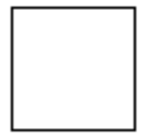
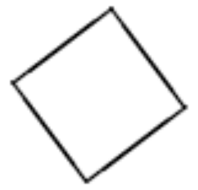
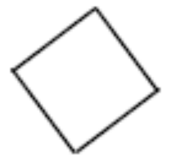
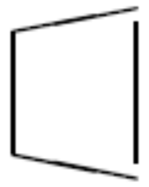
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Properties of projective transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Closed under composition

Source: N. Snavely

2D image transformations

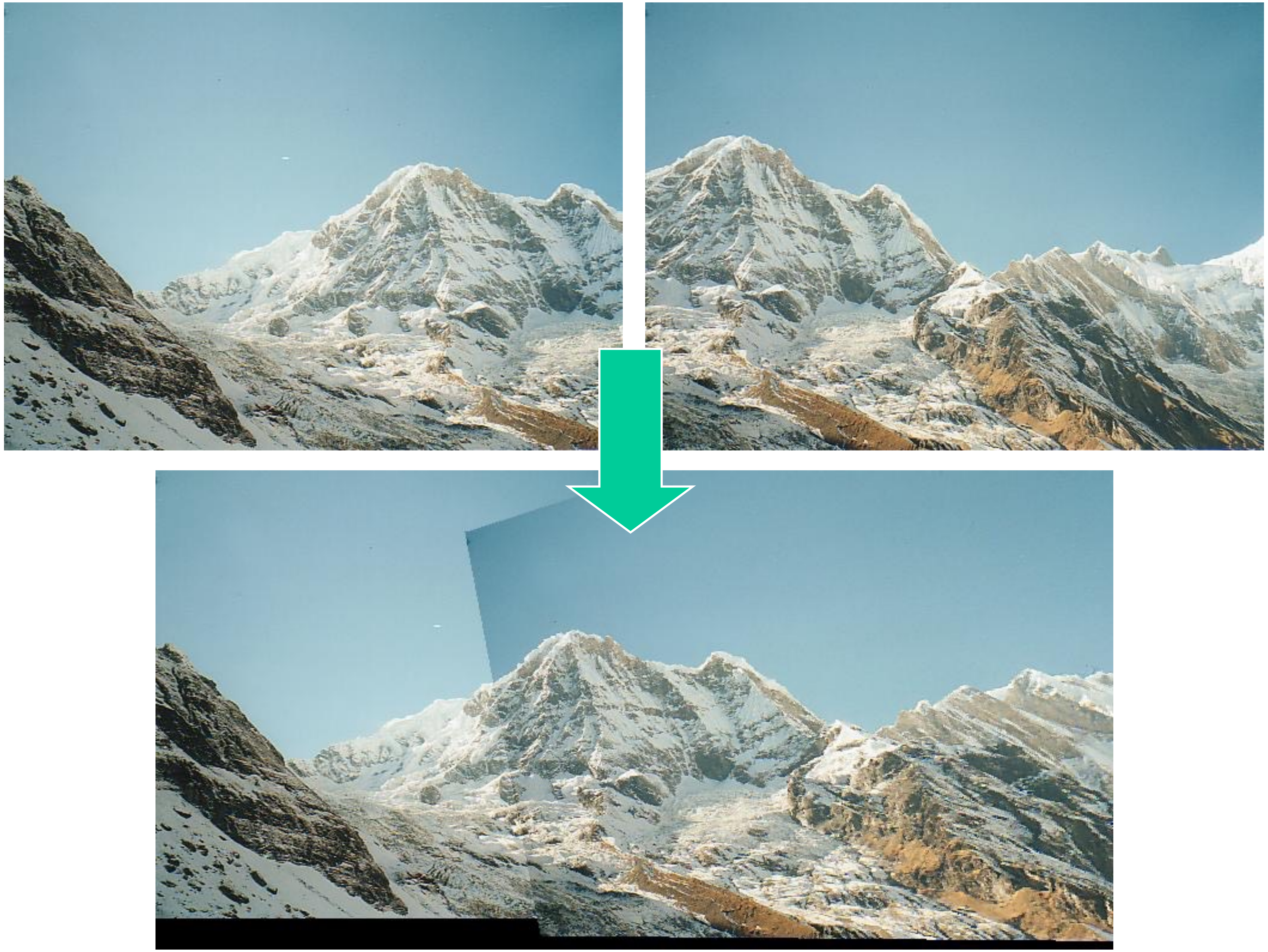


Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\left[\, \boldsymbol{I} \mid \boldsymbol{t} \,\right]_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\left[\, \boldsymbol{R} \mid \boldsymbol{t} \,\right]_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\left[\, s\boldsymbol{R} \mid \boldsymbol{t}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

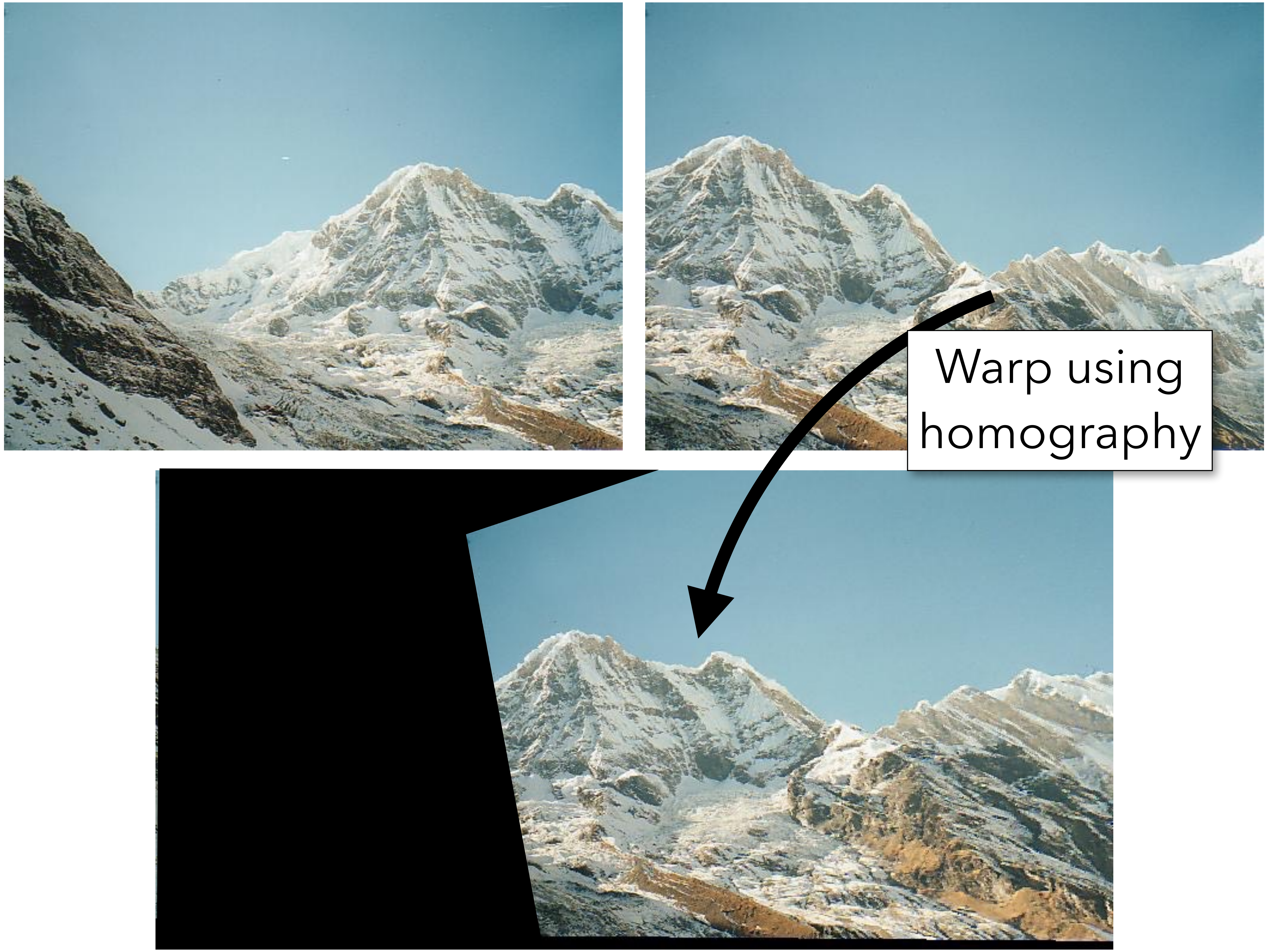
Source: N. Snavely

How do we perform this warp?

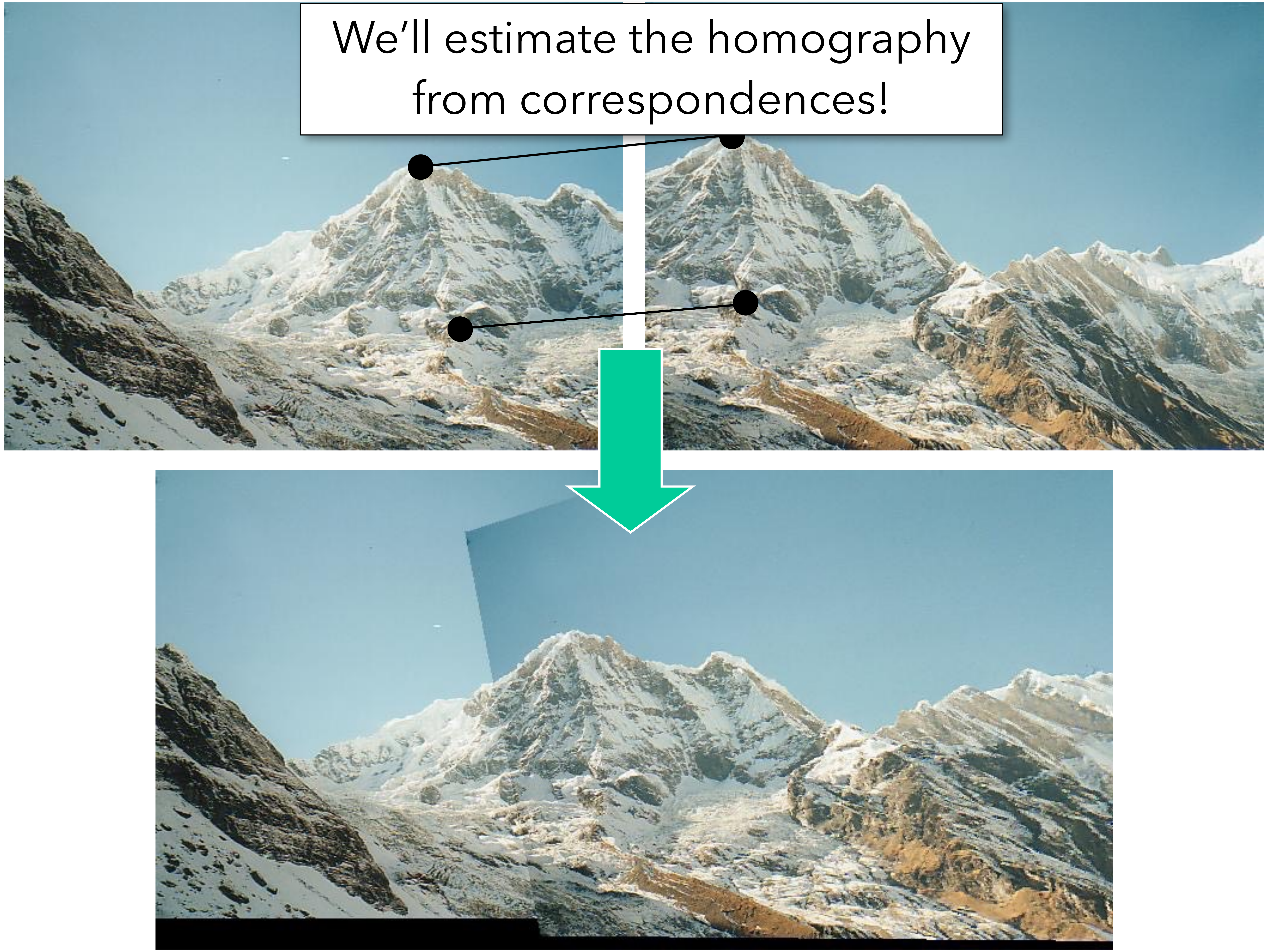
Panorama stitching (PS5)



Panorama stitching

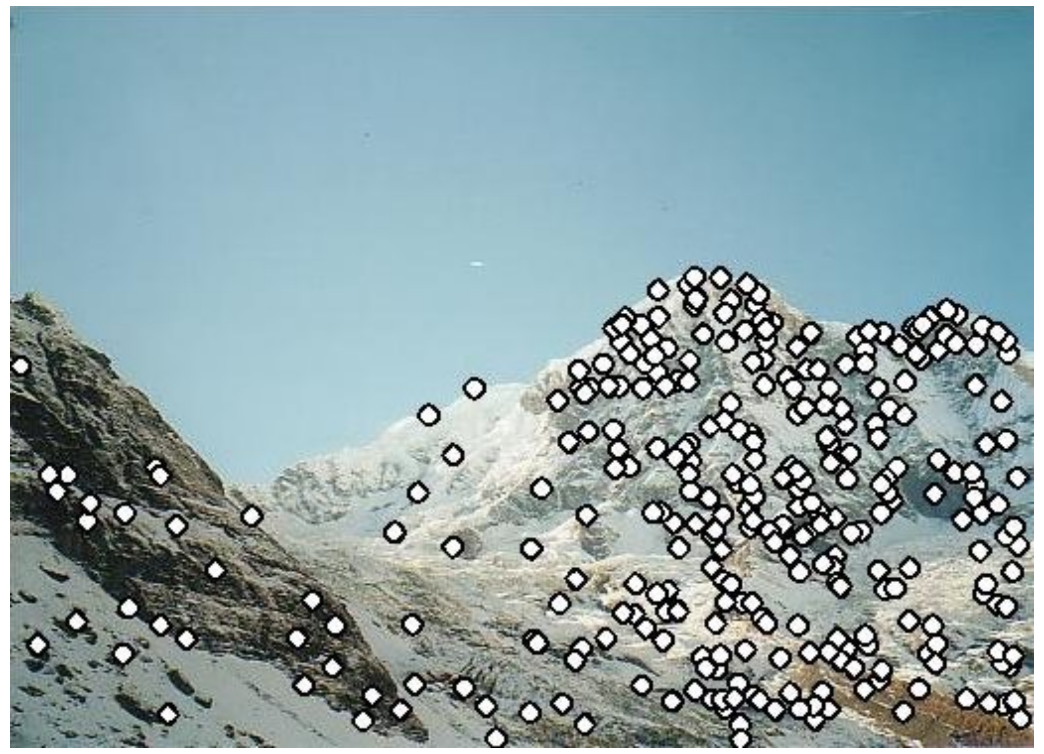


Panorama stitching



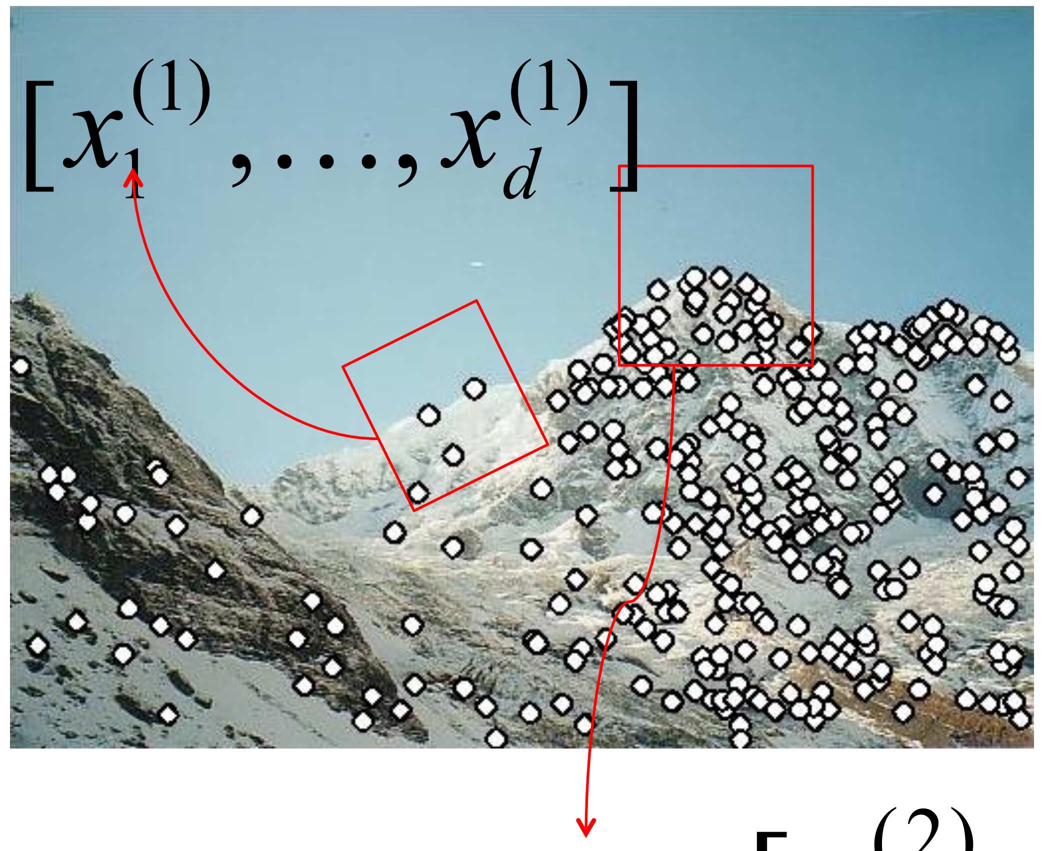
Finding correspondences with local features

1) **Detection:** Identify the interest points (a.k.a. keypoints), the candidate points to match.



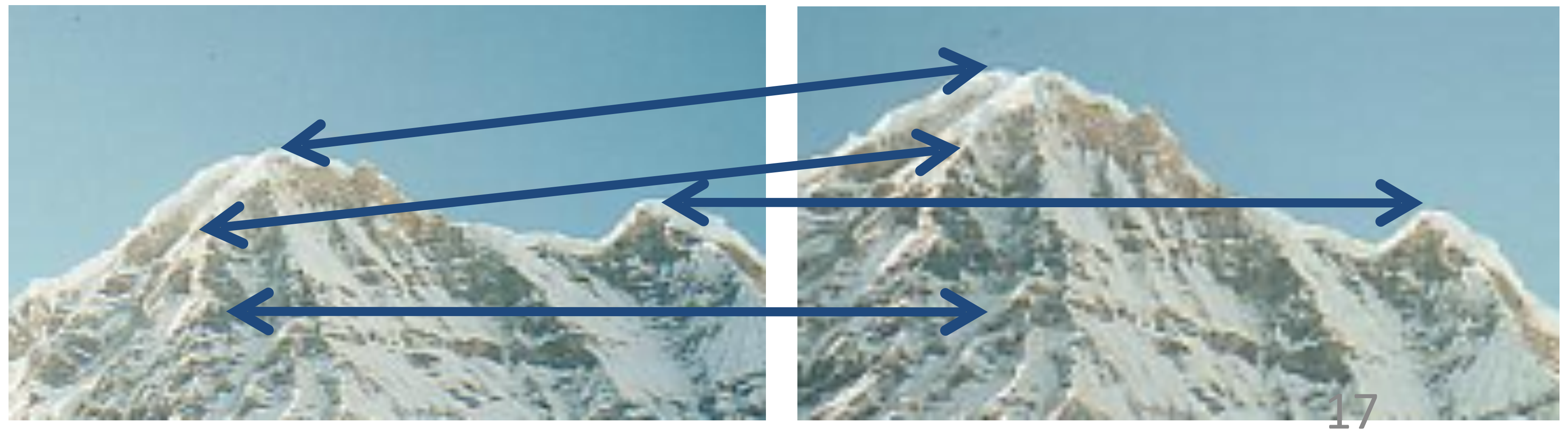
2) **Description:** Extract vector feature descriptor surrounding each interest point.

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



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

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



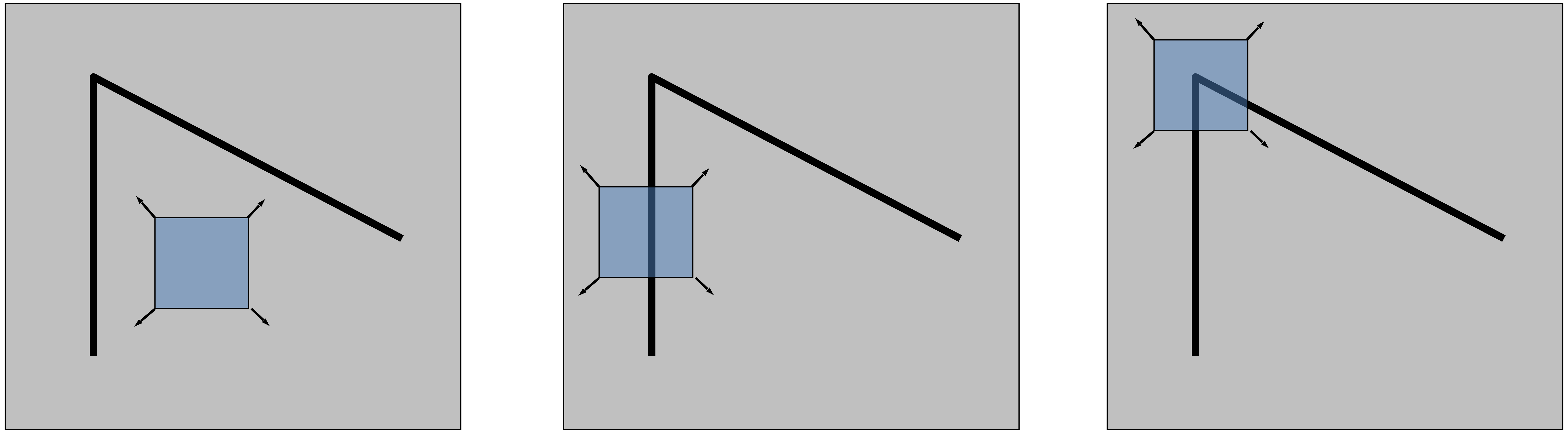
Source: K. Grauman

What are good regions to match?

Adapted from N. Snavely

What are good regions to match?

- How does the window change when you shift it?
- Shifting the window in any direction causes a big change



"flat" region:
no change in all
directions

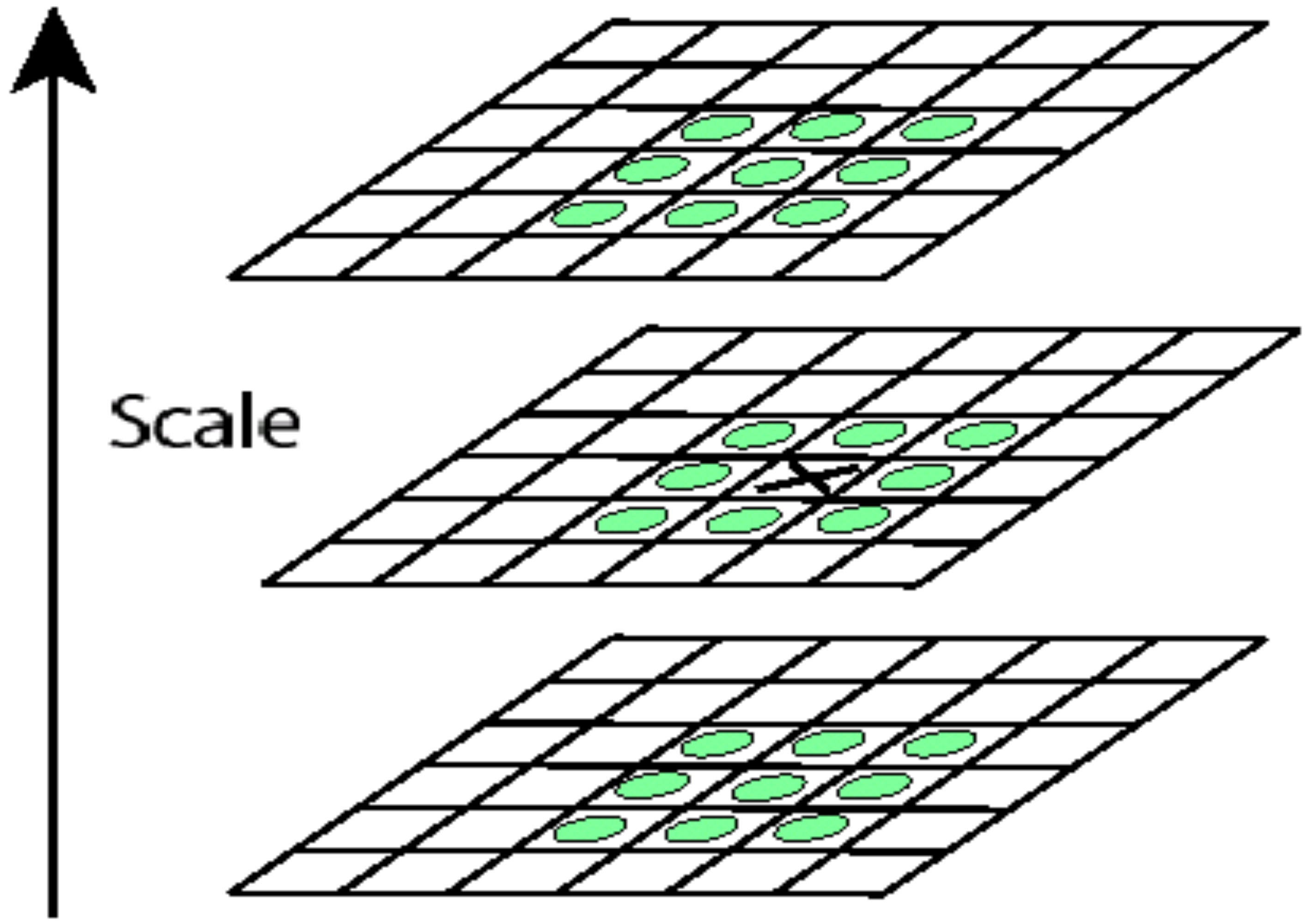
"edge":
no change along
the edge direction

"corner":
significant change
in all directions

Source: S. Seitz, D. Frolova, D. Simakov, N. Snavely

Finding good key points to match

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

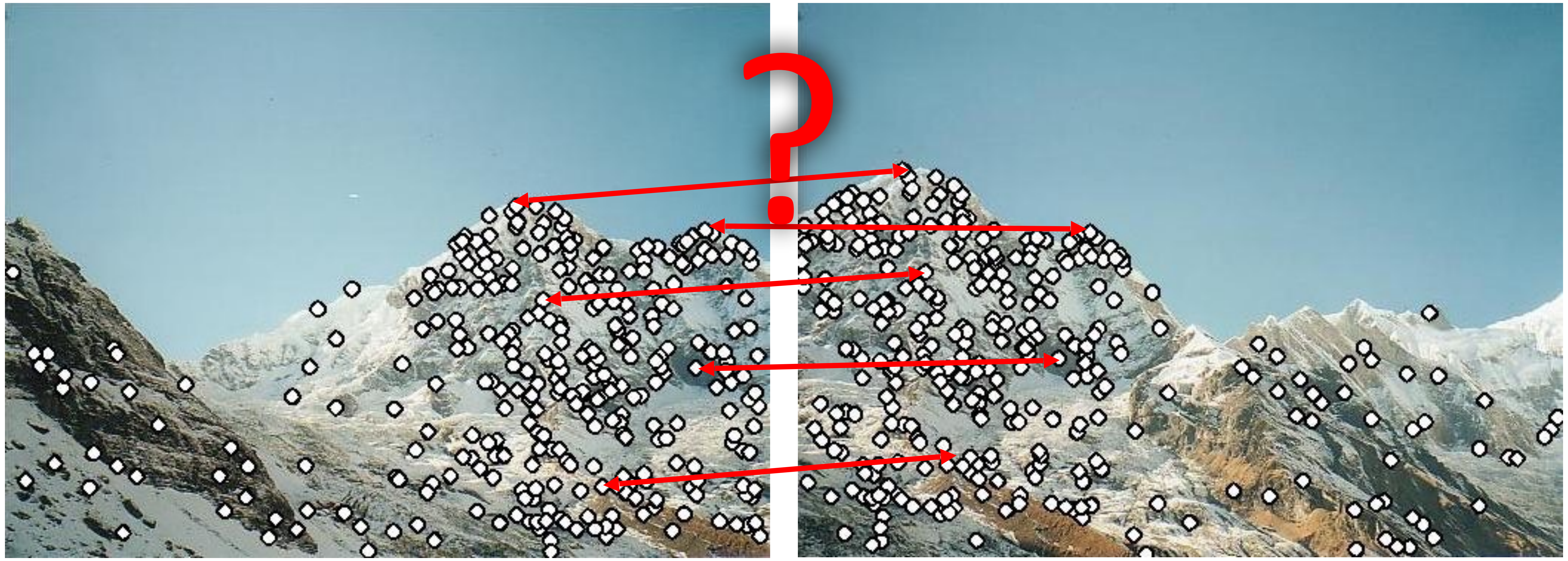


Compute difference-of-Gaussians filter (approx. to Laplacian).

Find local optima in space and scale using Laplacian pyramid.

Feature descriptors

We know how to detect good points
Next question: How do we match them?



Come up with a *descriptor* (feature vector) for each point, find similar descriptors between the two images

Source: N. Snavely

Scale Invariant Feature Transform (SIFT)

- Compute histograms of oriented gradients
- Take 16x16 square window around detected feature
- Compute edge orientation for each pixel
- Looks like a small, hand-crafted CNN

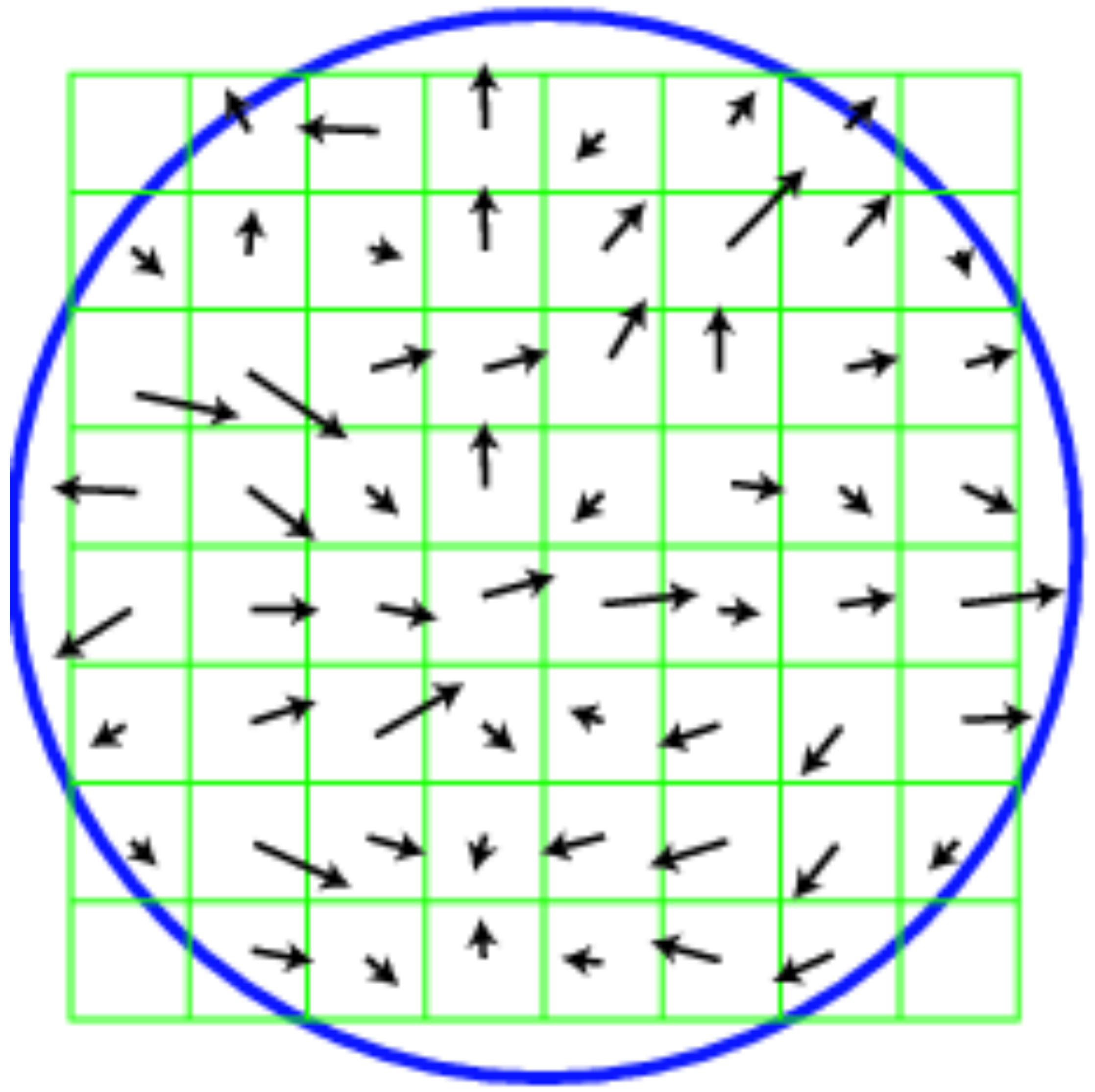
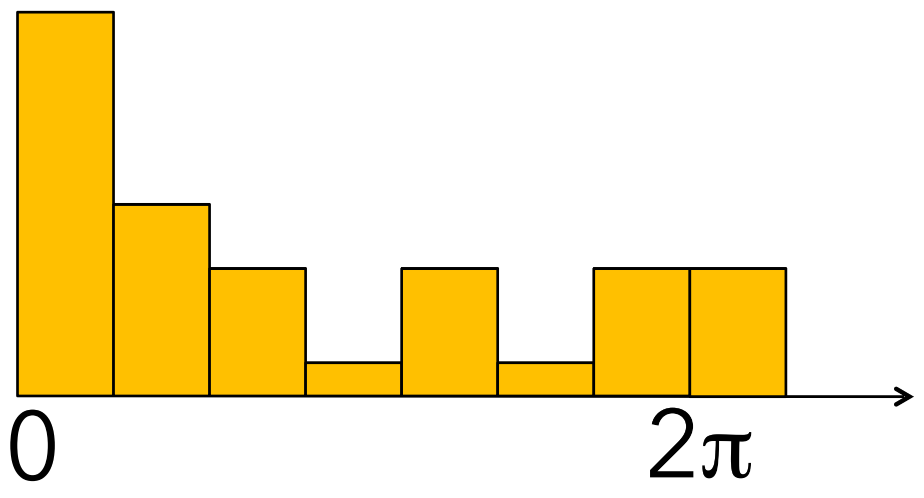
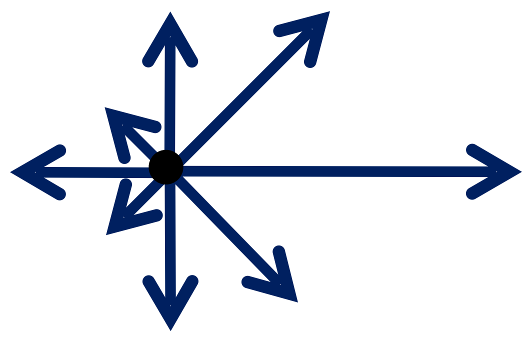


Image gradients



angle histogram



Source: N. Snavely, D. Lowe

Scale Invariant Feature Transform

Create the descriptor:

- Rotation invariance: rotate by “dominant” orientation
- Spatial invariance: spatial pool to 2x2
- Compute an orientation histogram for each cell
- (4 x 4) cells x 8 orientations = 128 dimensional descriptor

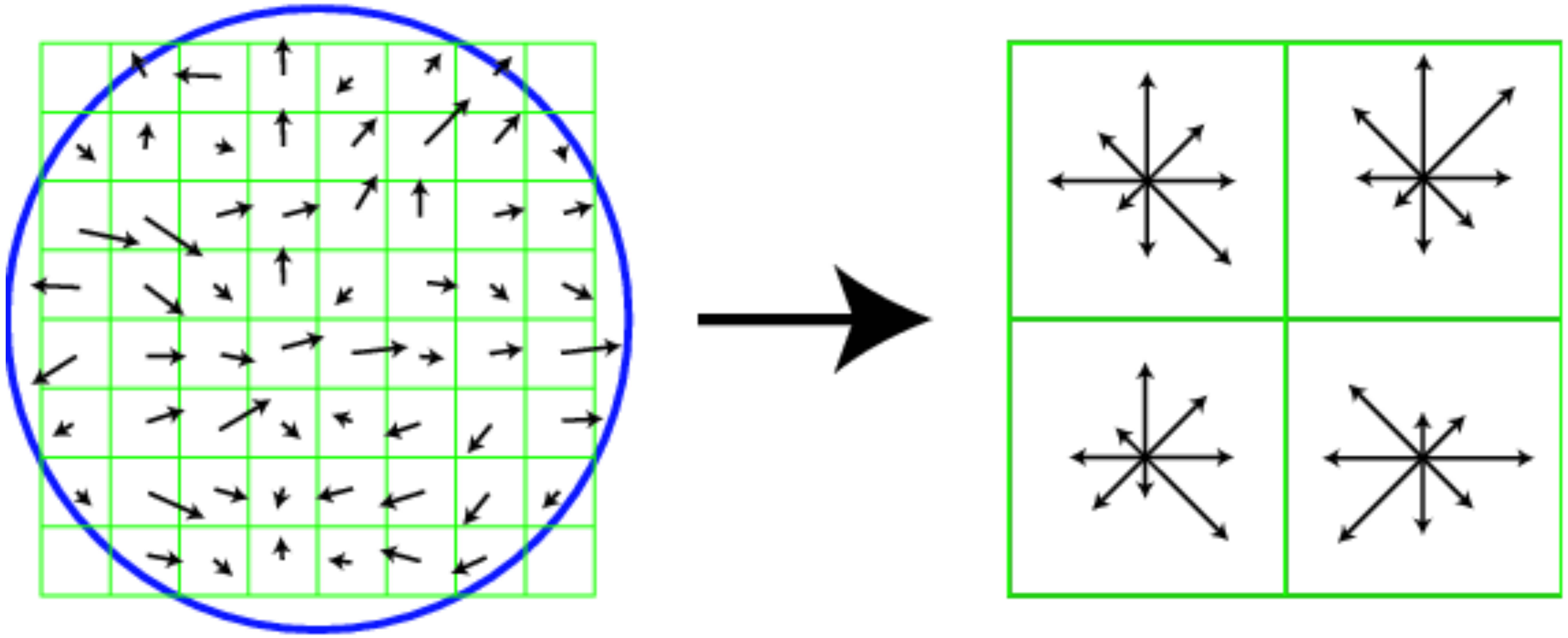


Image gradients   Keypoint descriptor

Source: N. Snavely, D. Lowe

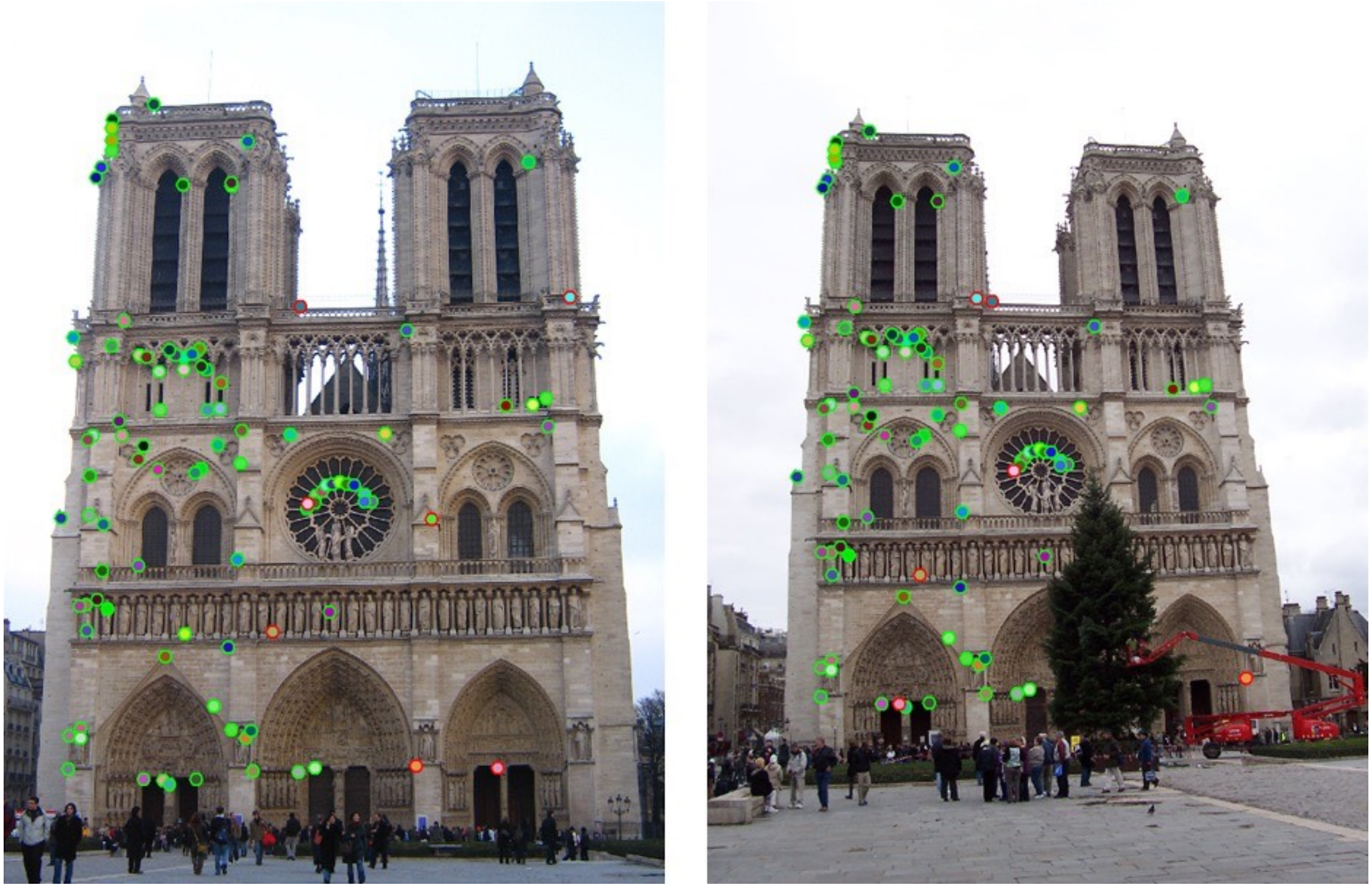
SIFT invariances

Source: N. Snavely

Today

- Finding correspondences
  - Computing local features
  - **Matching**
- Fitting a homography
- RANSAC

How can we tell whether two features match?



Source: N. Snavely

Feature matching

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

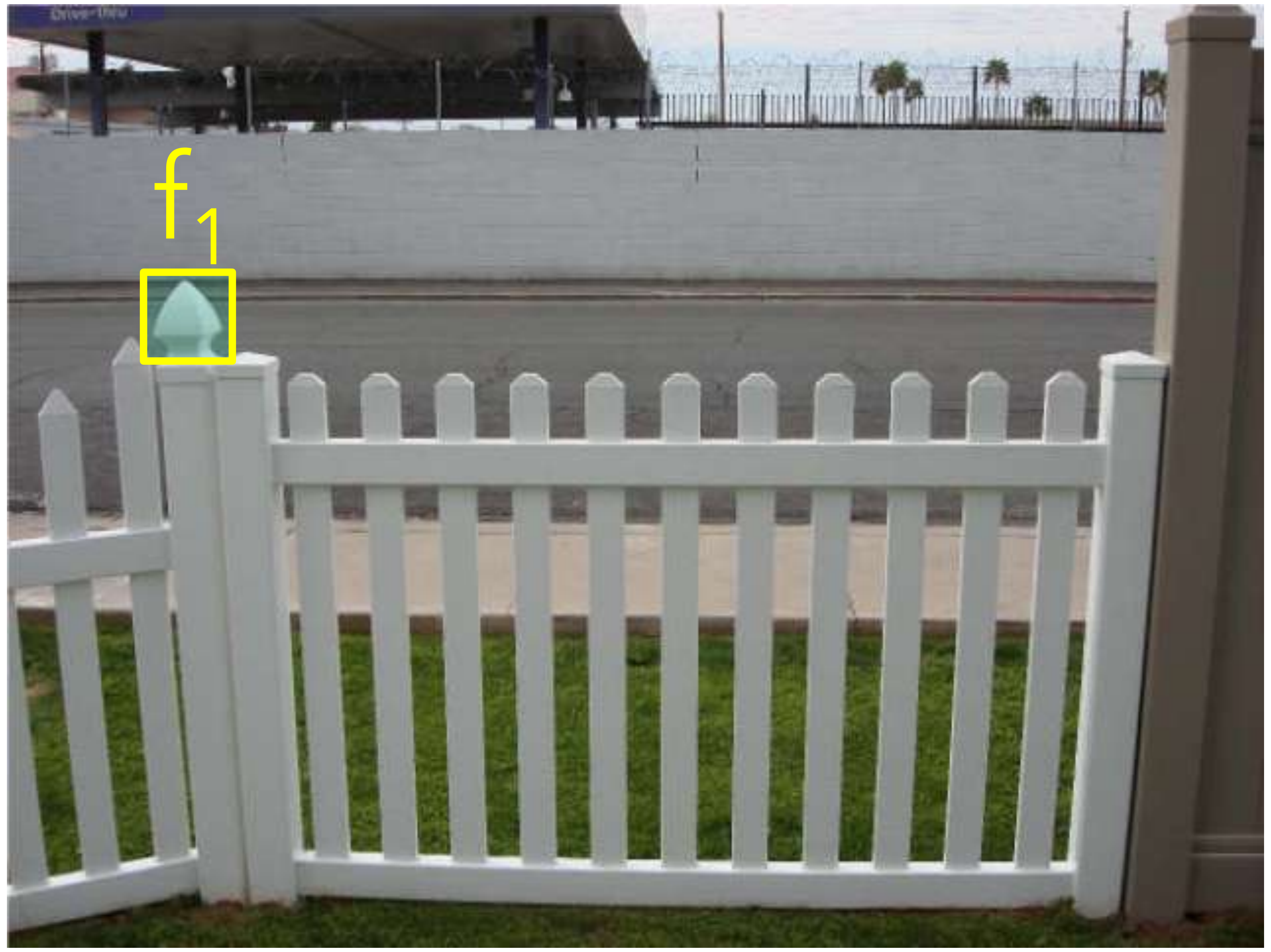
1. Define distance function that compares two descriptors
2. Test all the features in I_2, find the closest one.

Source: N. Snavely

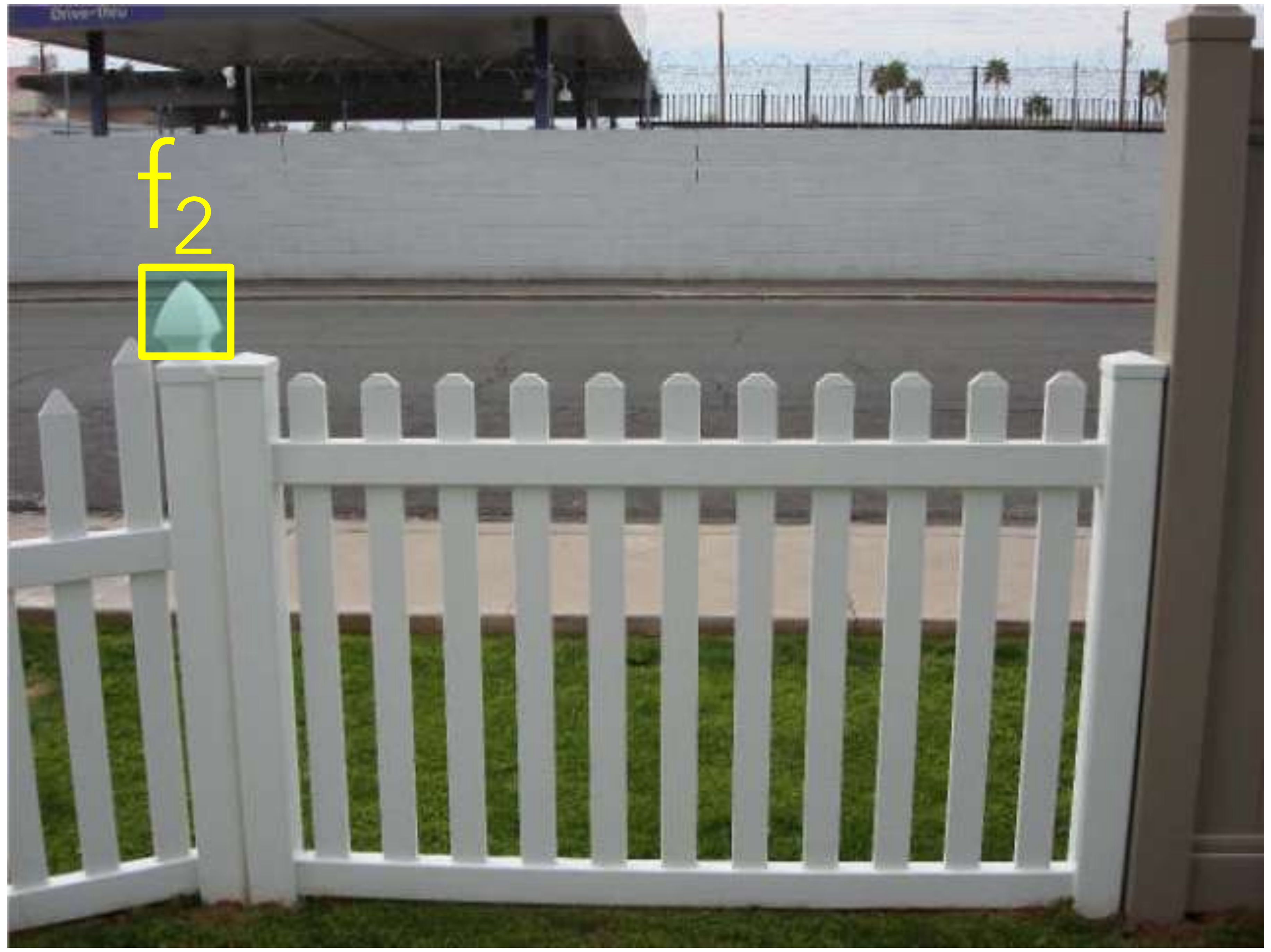
Finding matches

How do we know if two features match?

- Simple approach: are they the nearest neighbor in L_2 distance, $||f_1 - f_2||$?



I_1



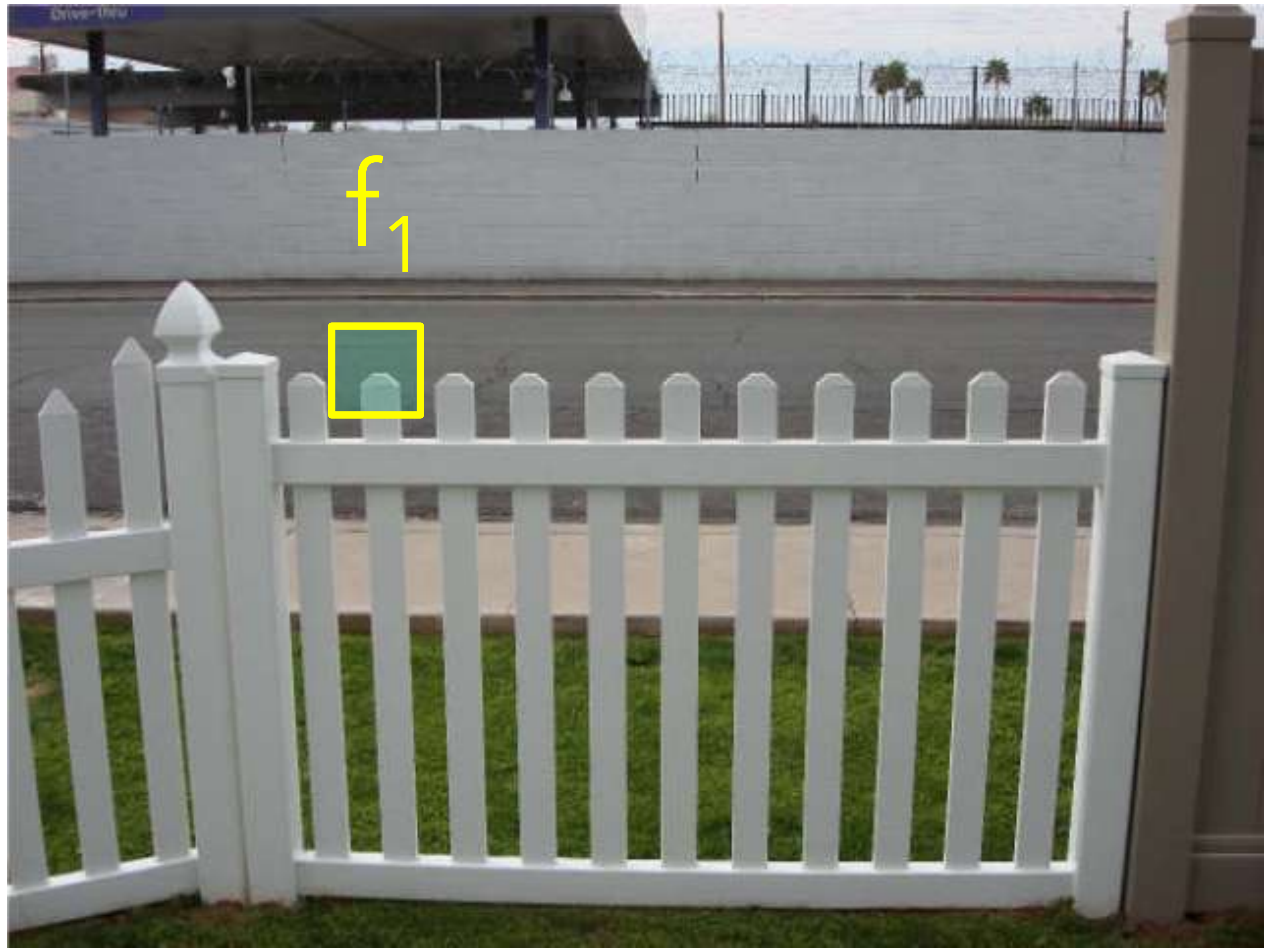
I_2

Source: N. Snavely

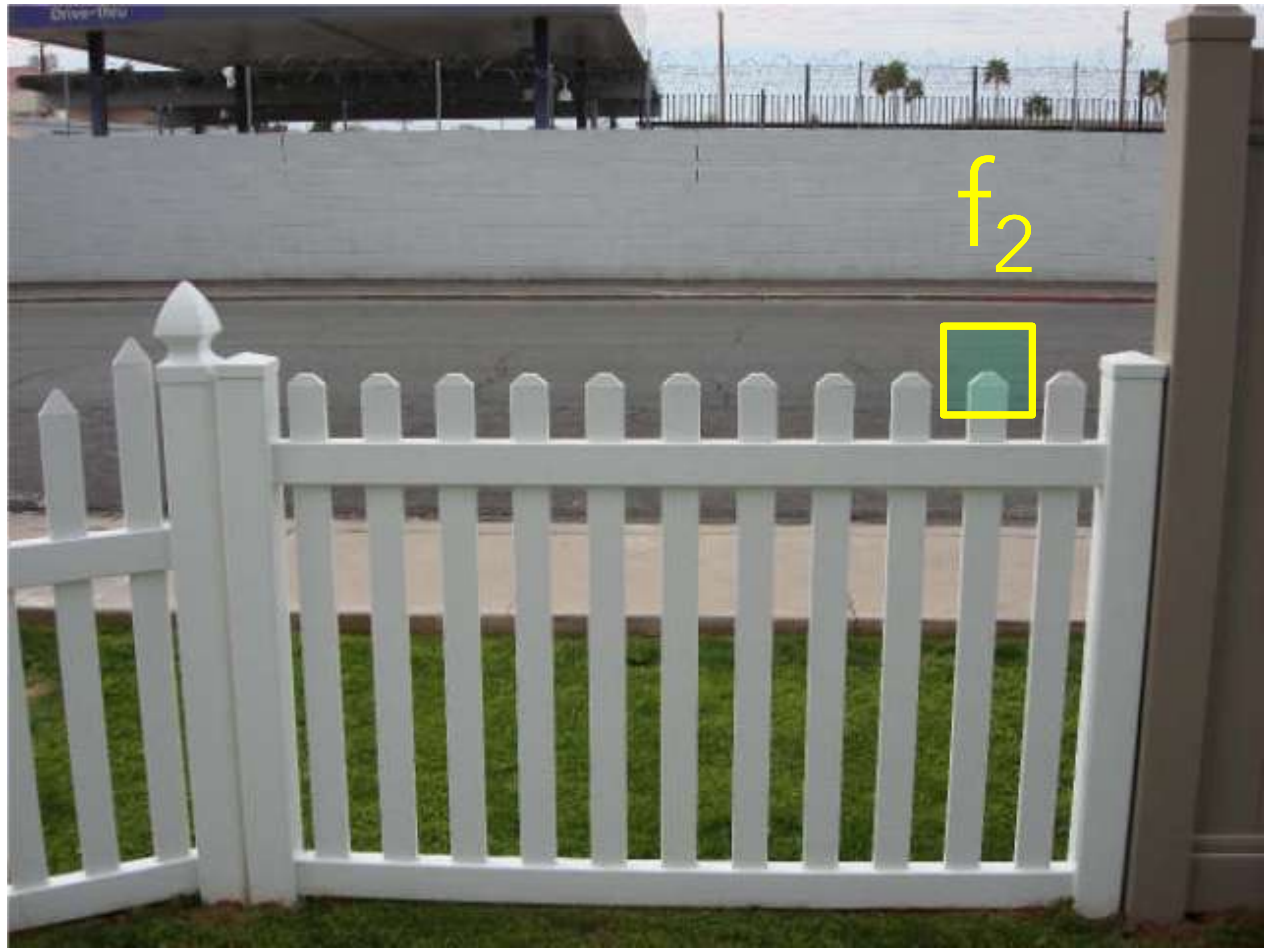
Finding matches

How do we know if two features match?

- Simple approach: are they the nearest neighbor in L_2 distance, $||f_1 - f_2||$?
- Can give good scores to ambiguous (incorrect) matches.



I_1



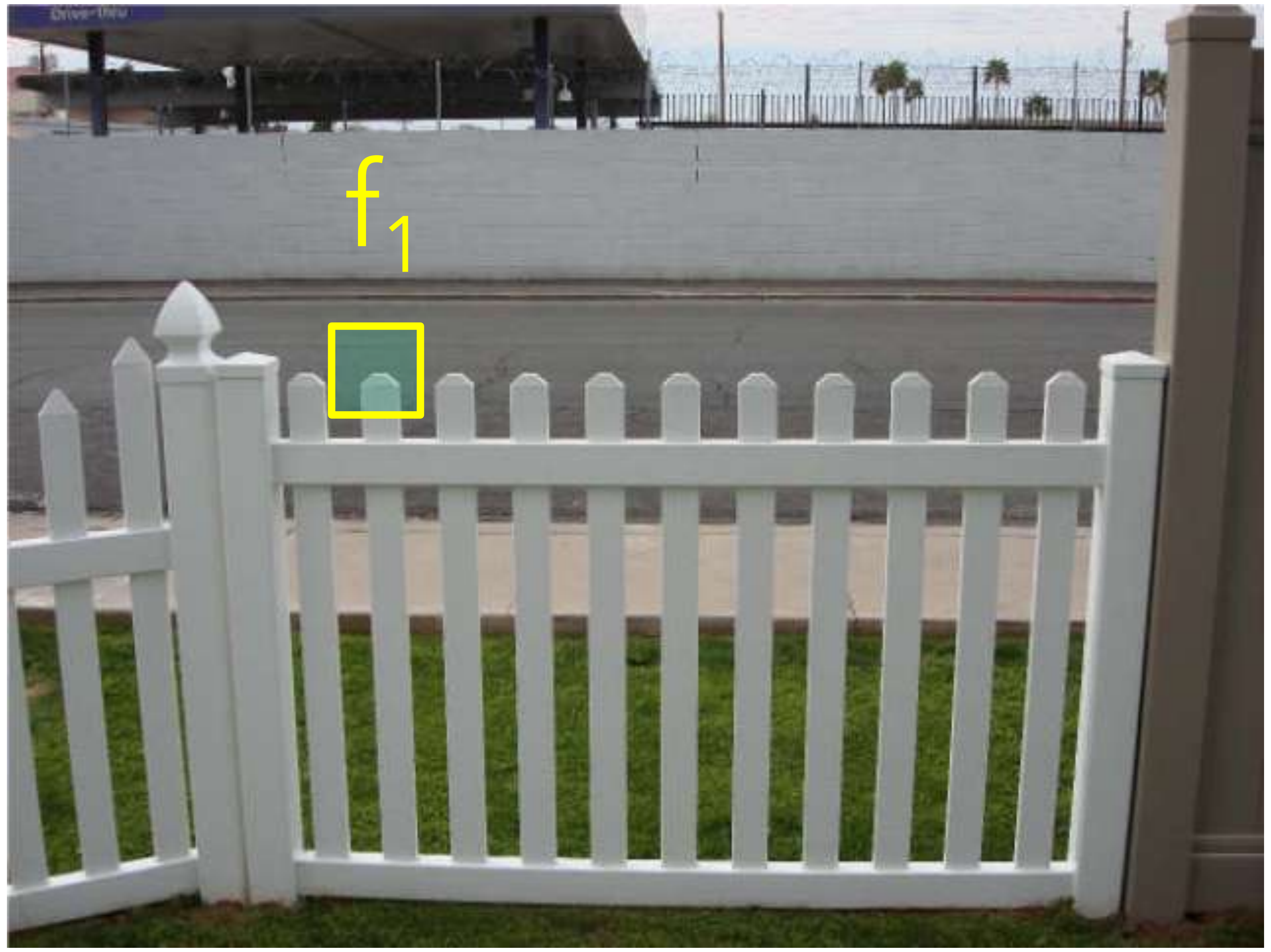
I_2

Source: N. Snavely

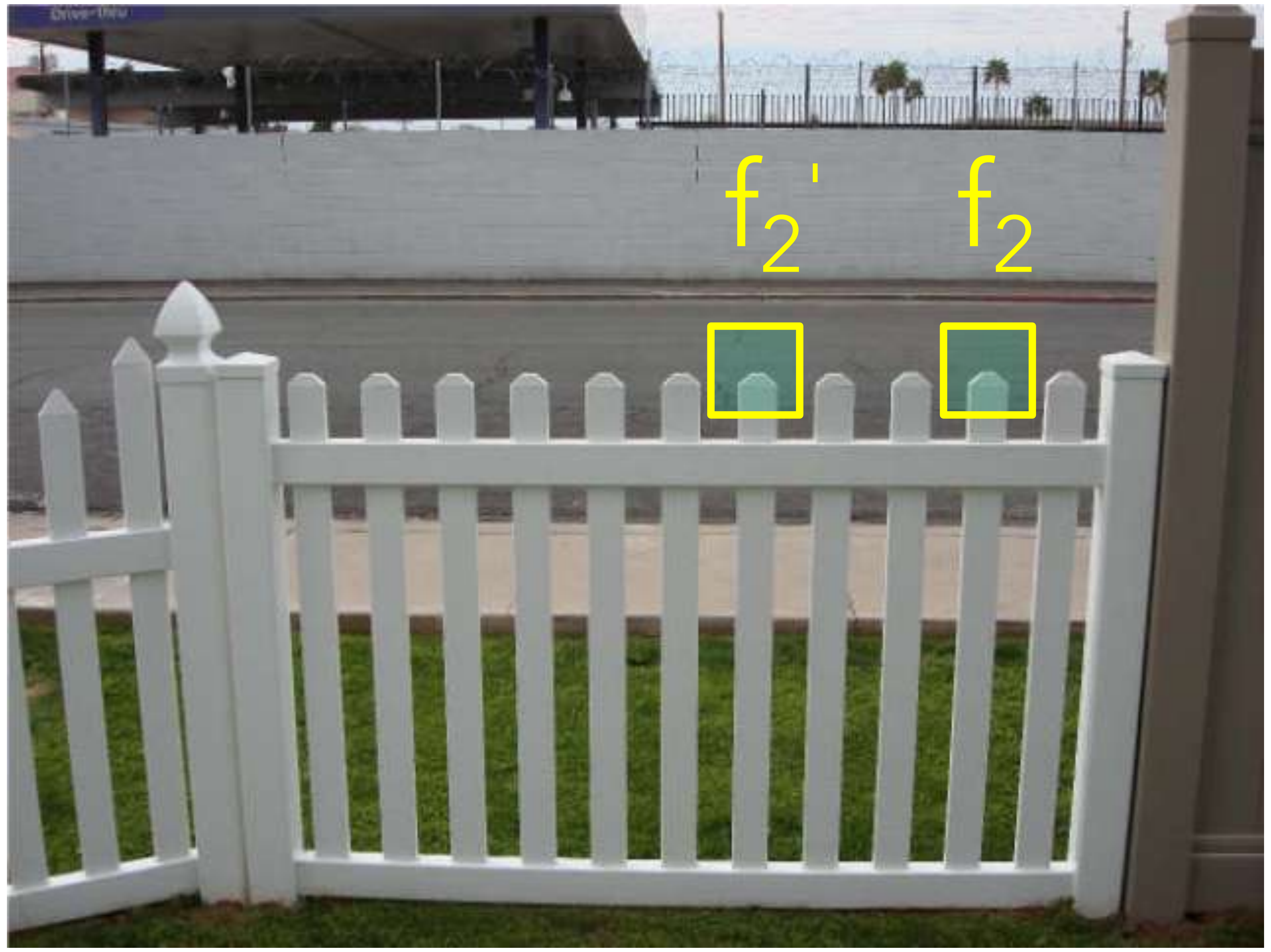
Finding matches

Throw away matches that fail tests:

- **Ratio test**: this *by far* the best match? Compare best and 2nd-best matches.
  - 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
- **Forward-backward consistency**: f_1 should also be nearest neighbor of f_2



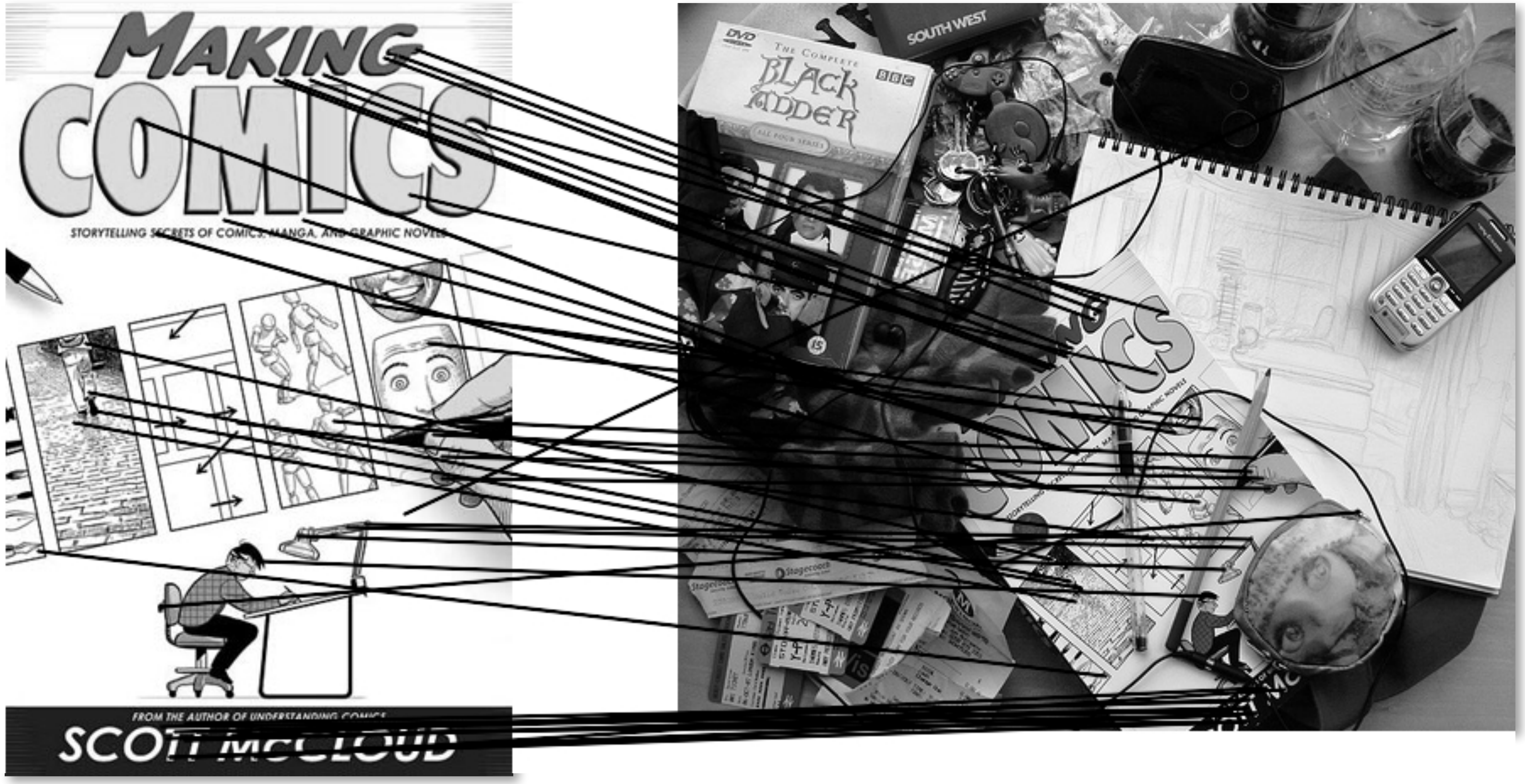
I_1



I_2

Source: N. Snavely

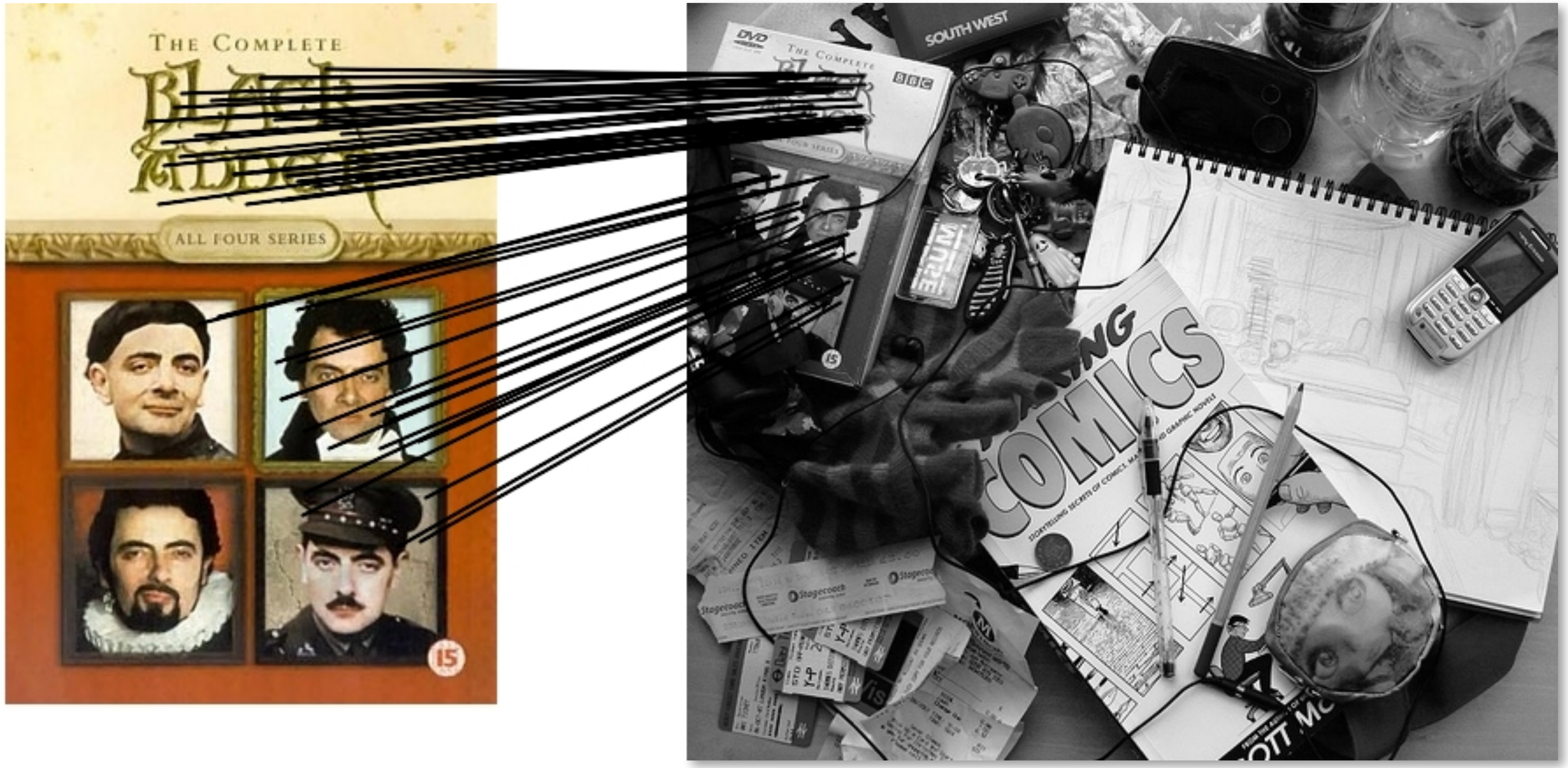
Feature matching example



51 feature matches after ratio test

Source: N. Snavely

Feature matching example



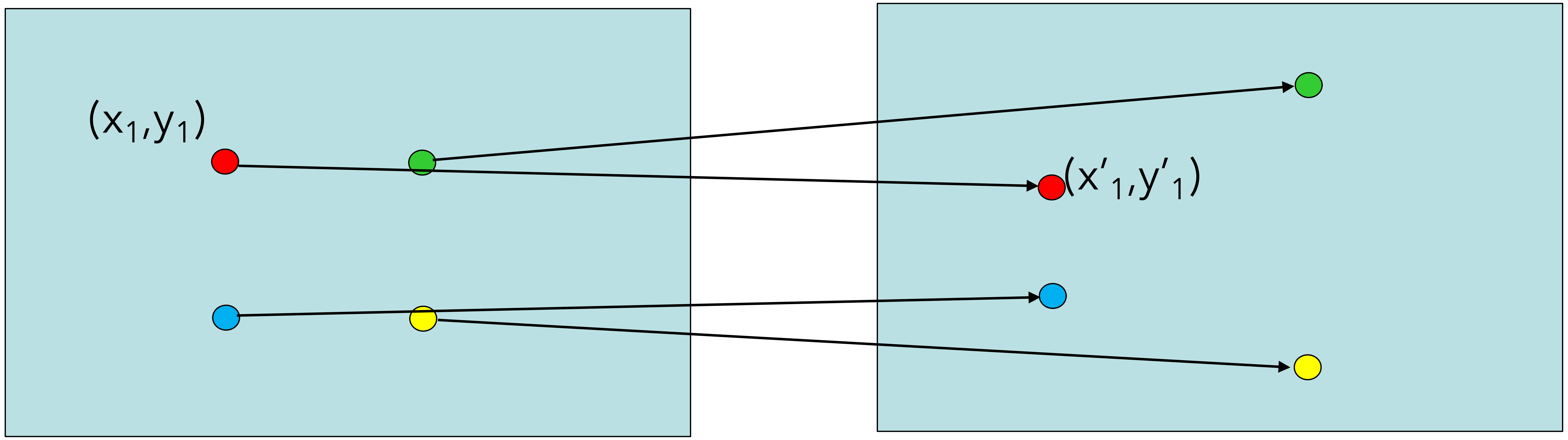
58 feature matches after ratio test

Source: N. Snavely

Today

- Finding correspondences
  - Computing local features
  - Matching
- **Fitting a homography**
- RANSAC

From matches to a homography



$$\begin{bmatrix} x_1' \\ y_1' \\ w_1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Source: Torralba, Isola, Freeman

From matches to a homography

Point in 1st image

Matched point in 2nd

minimize $J(H) = \sum_i ||f_H(p_i) - p_i'||^2$

where $f_H(p_i) = Hp_i/(H_3^T p_i)$ applies homography

Remember: homogenous coordinates.

H_3 is the third row of H

Option #1: Direct linear transform

$$\begin{bmatrix} x_1' \\ y_1' \\ w_1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Leaving homogeneous coordinates:

$$x_1' = \frac{ax_1 + by_1 + c}{gx_1 + hy_1 + i}$$

$$y_1' = \frac{dx_1 + ey_1 + f}{gx_1 + hy_1 + i}$$

Re-arranging the terms:

$$gx_1x'_1 + hy_1x'_1 + ix_1' = ax_1 + by_1 + c$$

$$gx_1y'_1 + hy_1y'_1 + ix_1' = dx_1 + ey_1 + f$$

Source: Torralba, Freeman, Isola

Option #1: Direct linear transform

$$g x_1 x'_1 + h y_1 x'_1 + i x_1' = a x_1 + b y_1 + c$$

$$g x_1 y'_1 + h y_1 y'_1 + i x_1' = d x_1 + e y_1 + f$$

More rearranging:

$$g x_1 x'_1 + h y_1 x'_1 + i x'_1 - a x_1 - b y_1 - c = 0$$

$$g x_1 y'_1 + h y_1 y'_1 + i y'_1 - d x_1 - e y_1 - f = 0$$

In matrix form:

$$\begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1 x'_1 & y_1 x'_1 & x'_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1 y'_1 & y_1 y'_1 & y'_1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

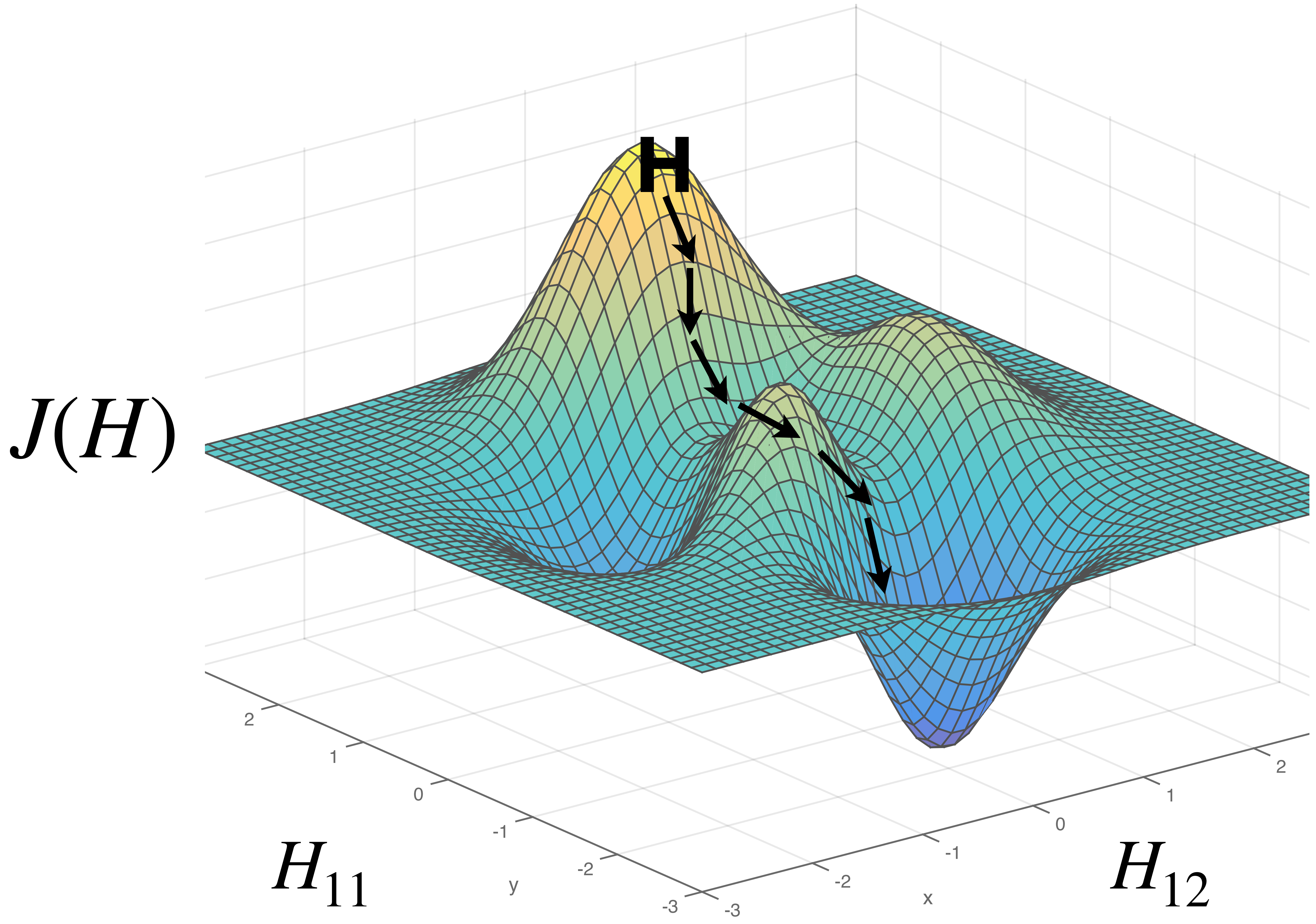
Can solve using Singular Value Decomposition (SVD). Avoids trivial solution (all 0).

Fast to solve (but not using “right” loss function). Uses an algebraic trick.

Often used in practice for initial solutions!

Source: Torralba, Freeman, Isola

Option #2: Optimization



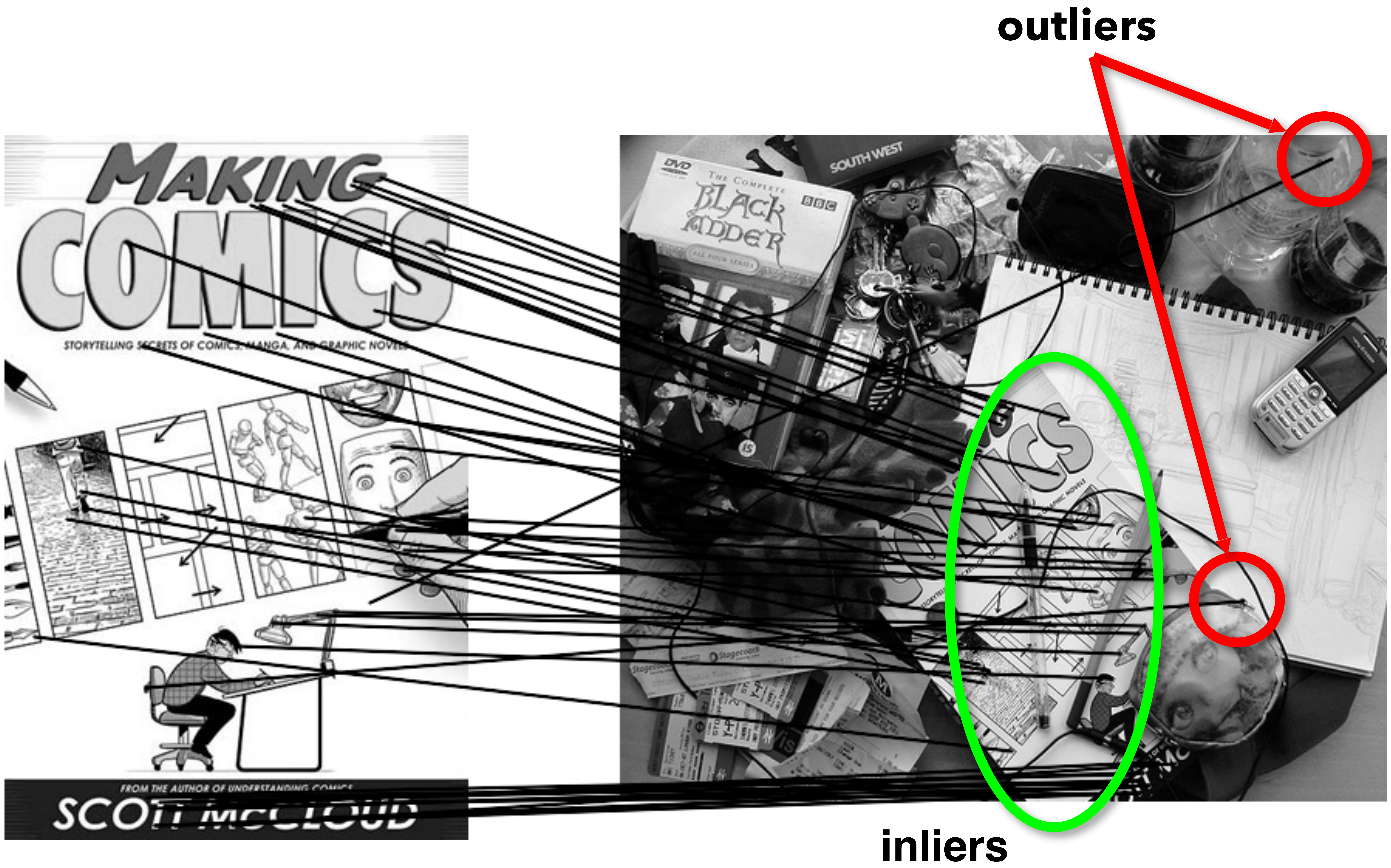
$$\text{minimize} \quad J(H) = \sum_i ||f_H(p_i) - p_i'||^2$$

Optimization

$$\text{minimize} \quad J(H) = \sum_i ||f_H(p_i) - p'_i||^2$$

- Can use gradient descent, just like when learning neural nets
- But while these problems are **smaller scale** than deep learning problems, they have **more local optima**:
  - Use 2nd derivatives to improve optimization
  - Can use finite differences or autodiff
- Can use special-purpose **nonlinear least squares** methods.
  - Exploits structure in the problem for a sum-of-squares loss.

Problem: outliers



Source: N. Snavely

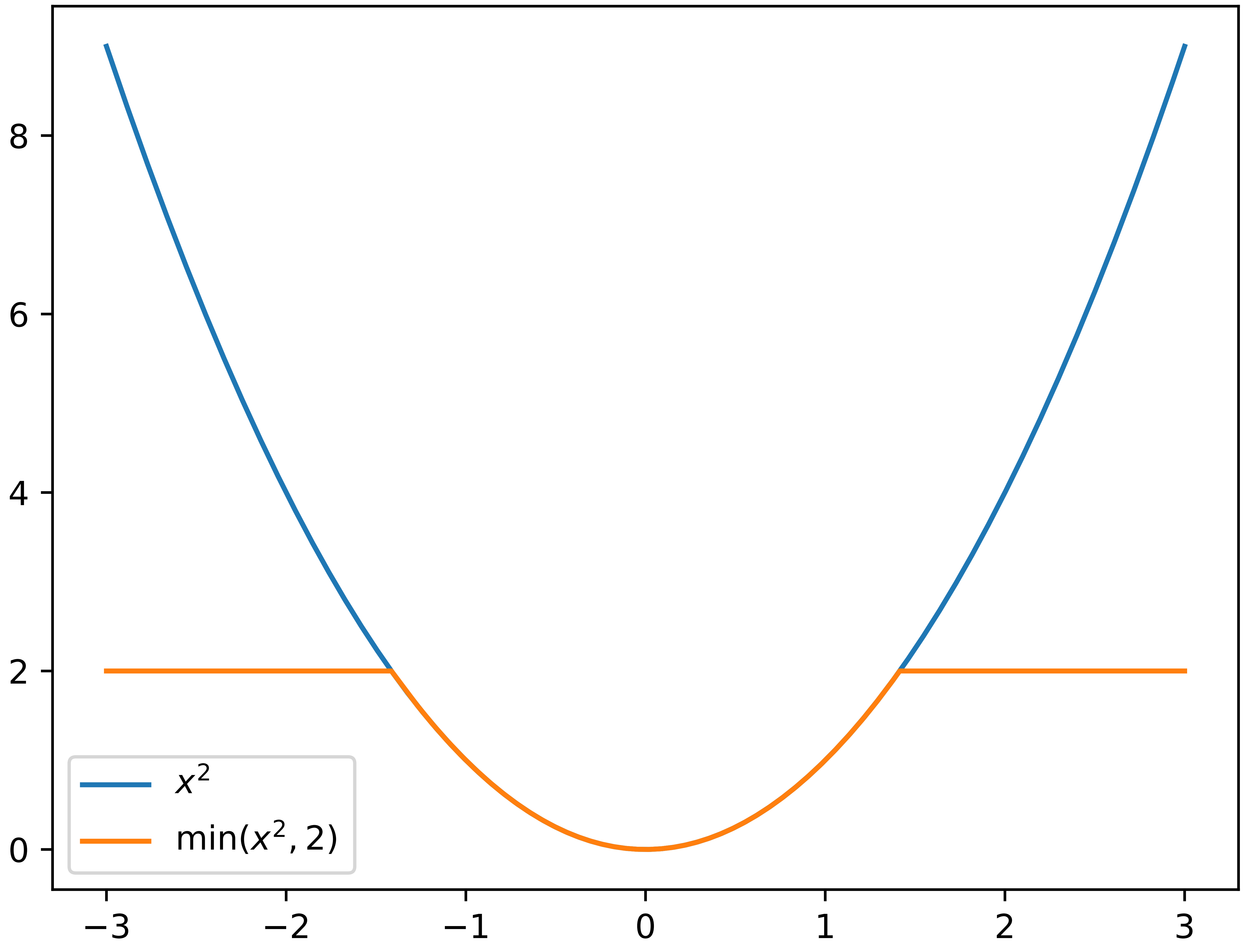
One idea: robust loss functions

minimize $J(H) = \sum_{i=1}^{N} \sum_{j=1}^{2} \rho(f_H(p_{ij}) - p'_{ij})$

where $\rho(x)$ is a **robust** loss.

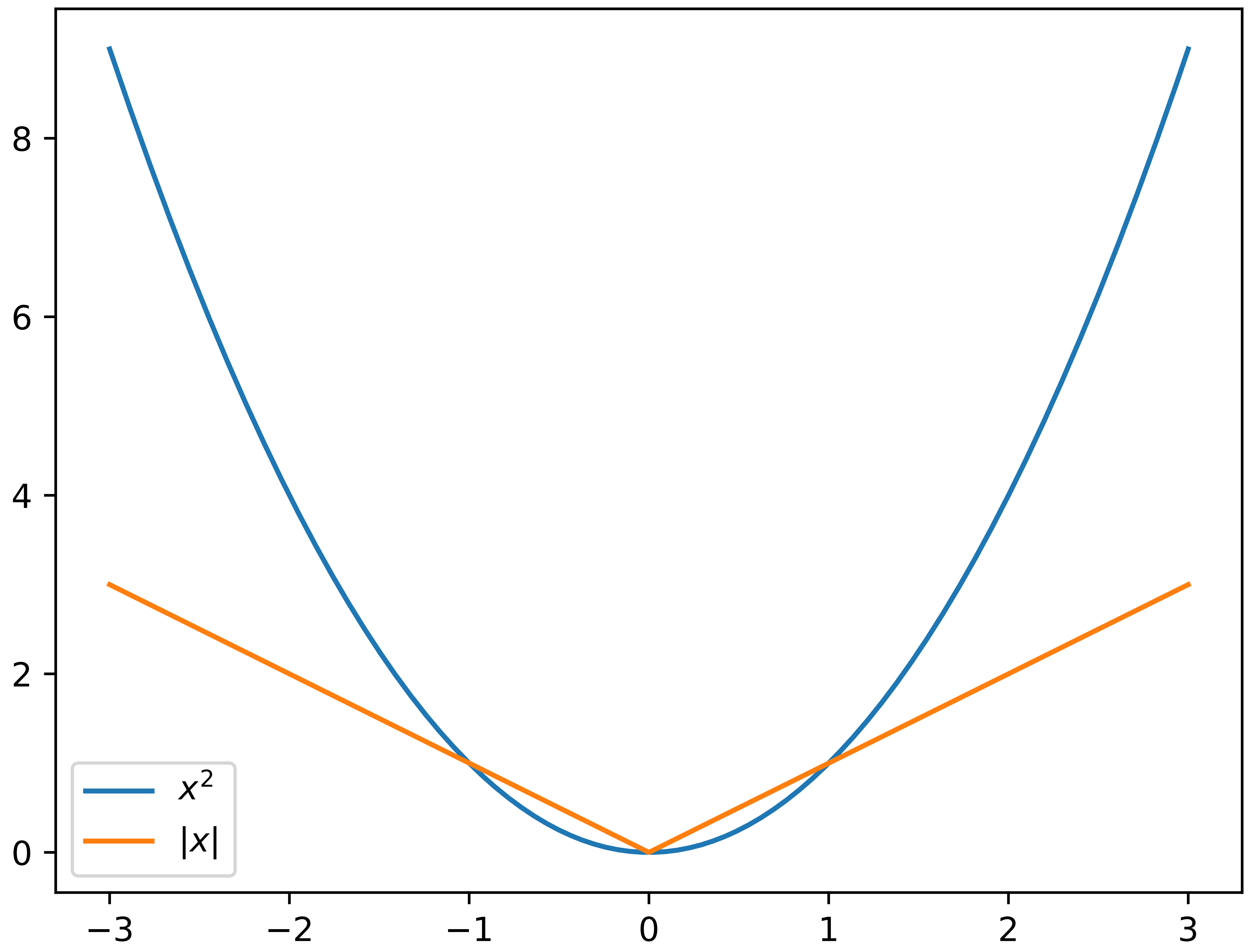
Special case: $\rho(x) = x^2$ is L2 loss (same as before)

Robust loss functions



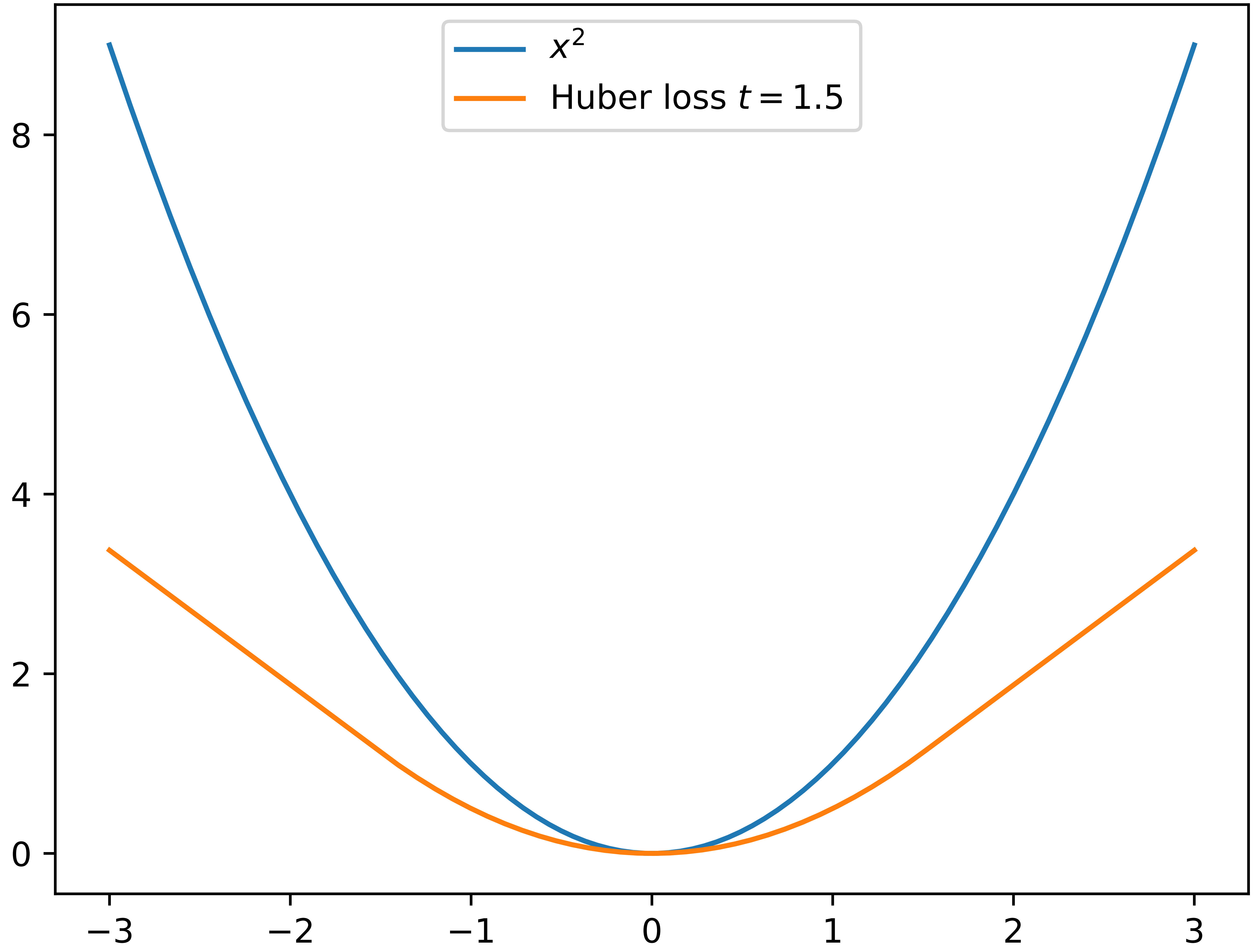
Truncated quadratic: $\rho(x) = \min(x^2, \tau)$

Robust loss functions



L1 loss: $\rho(x) = |x|$

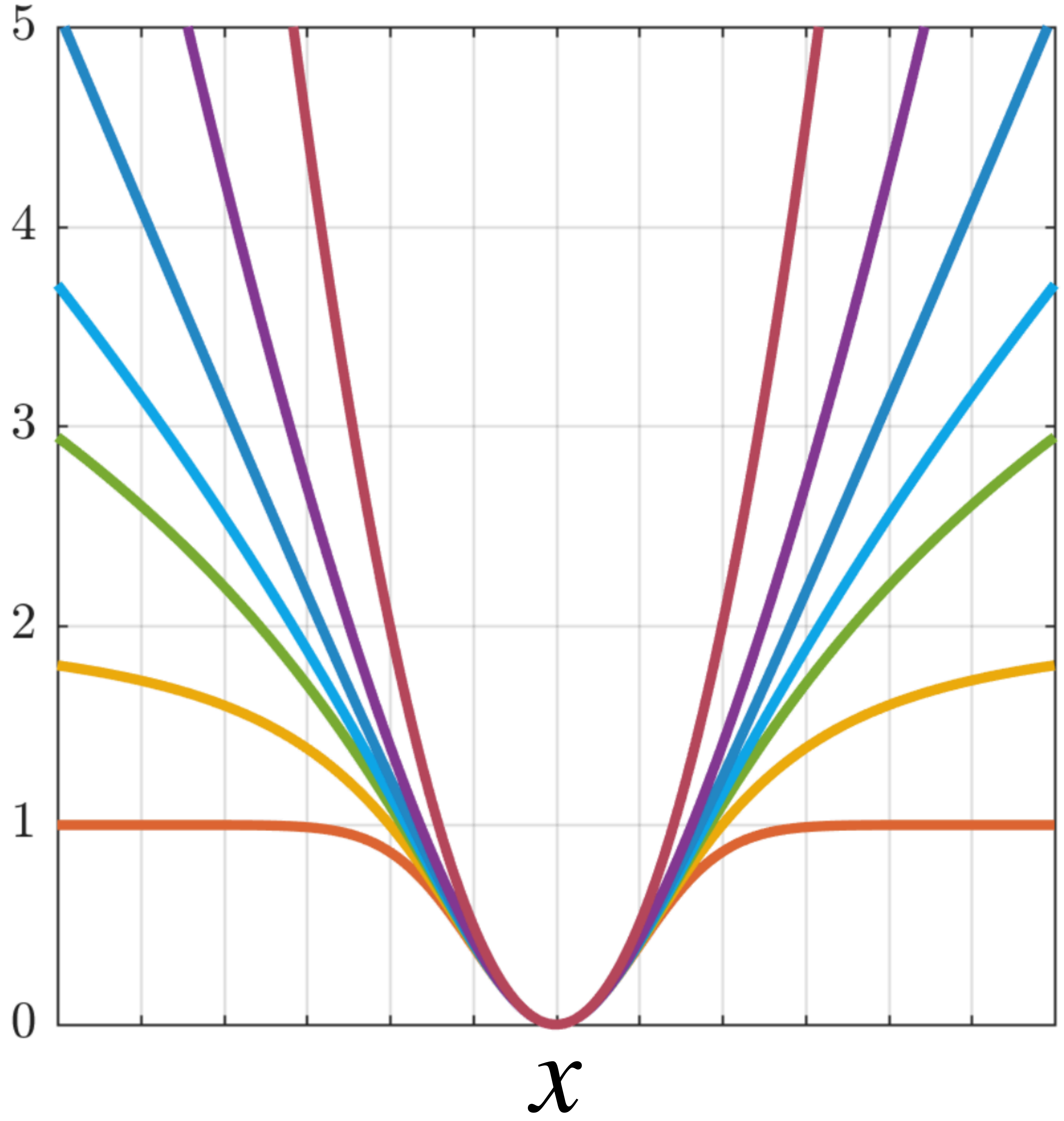
Robust loss functions



Huber loss:

$$\rho(x) = \begin{cases} \frac{1}{2}x^2 & \textbf{if } |x| \leq \tau, \\ \tau(|x| - \frac{1}{2}\tau), & \textbf{else} \end{cases}$$

Robust loss functions

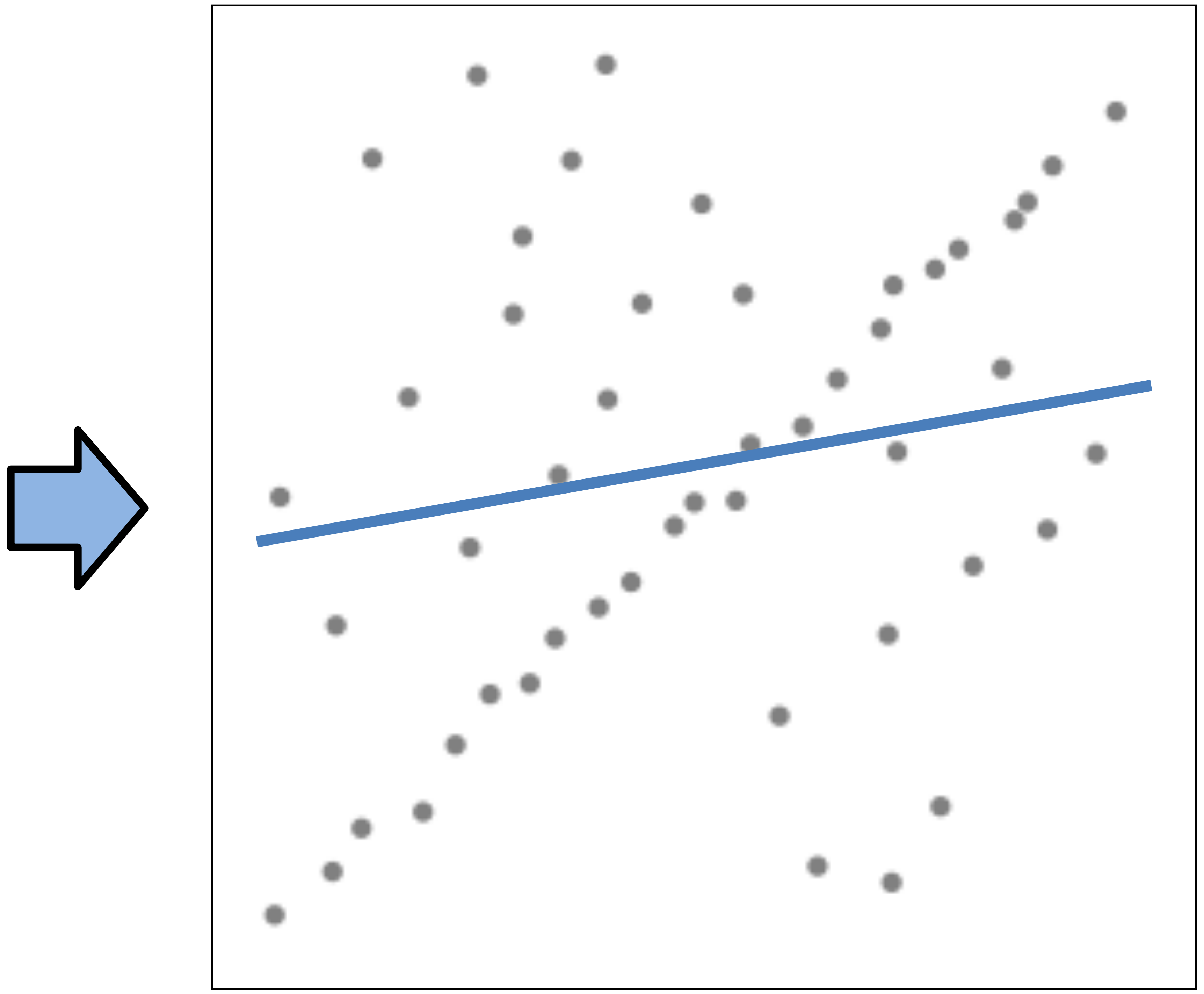


Source: [Barron 2019, "A General and Adaptive Robust Loss Function"]

Handling outliers

- Can be hard to fit a robust loss, e.g., due to local minima
- Another idea: trial and error!
- Let's consider the problem of linear regression

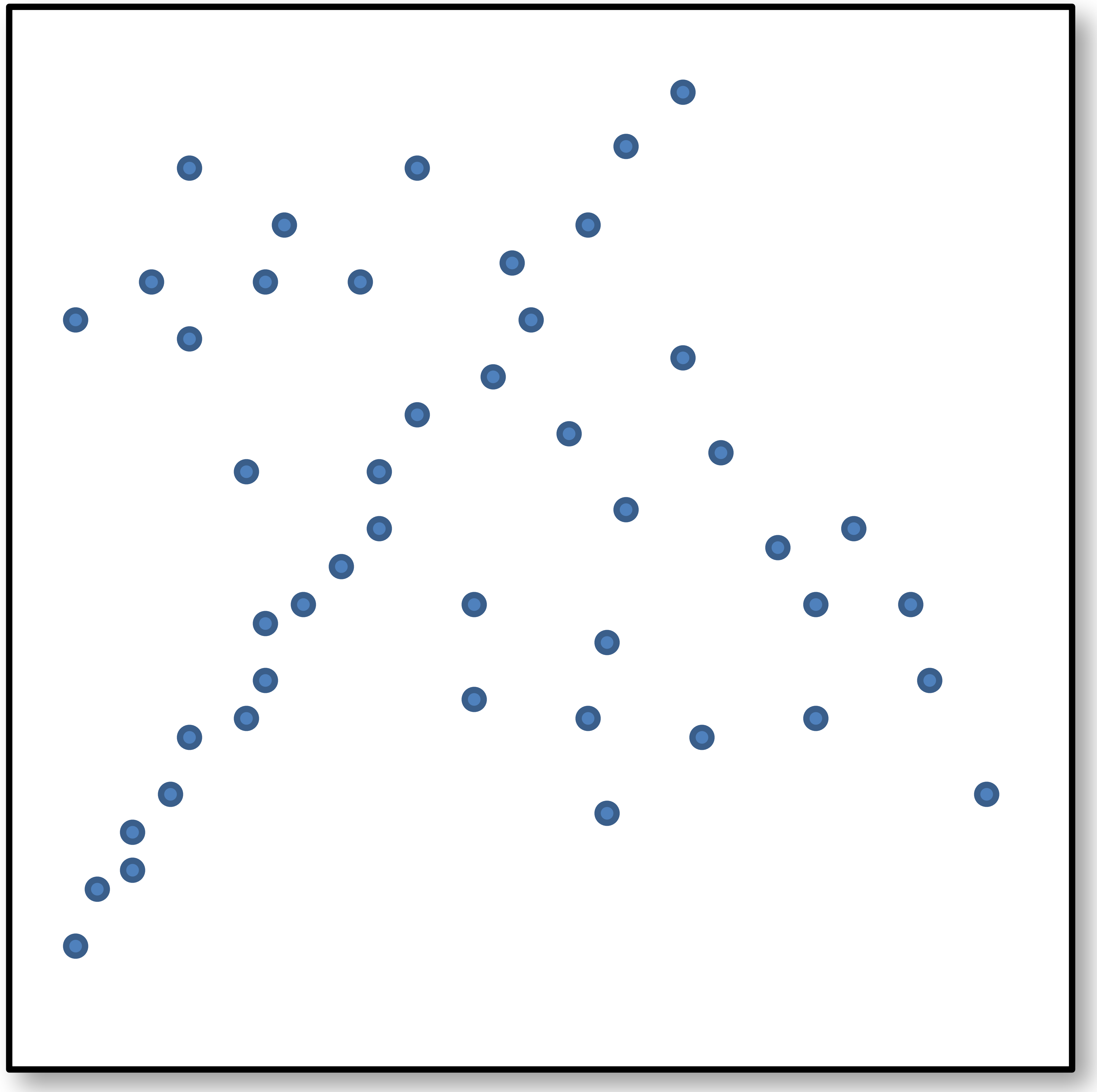
Problem: Fit a line to these data points



Least squares fit

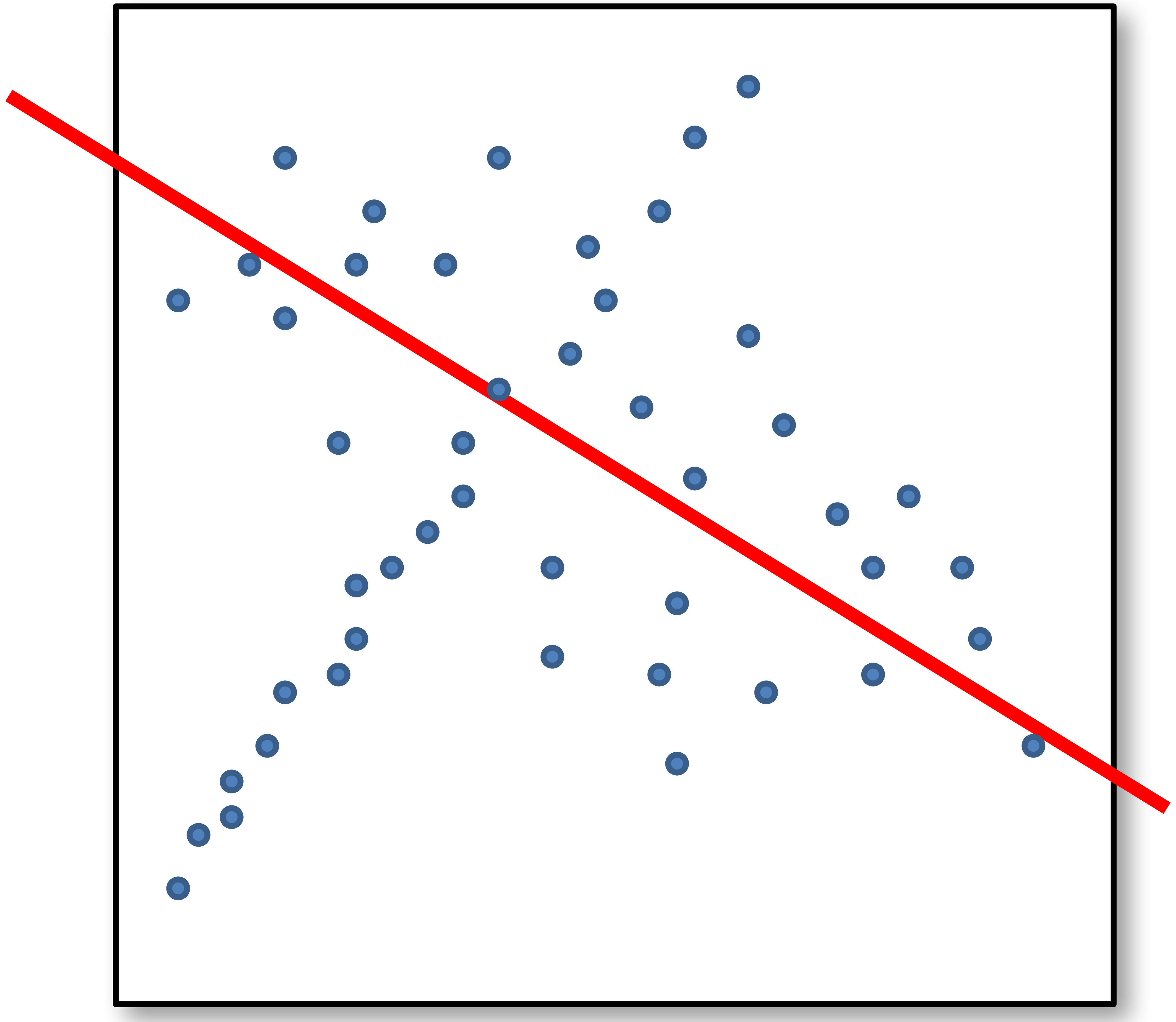
Source: N. Snavely

Counting inliers



Source: N. Snavely

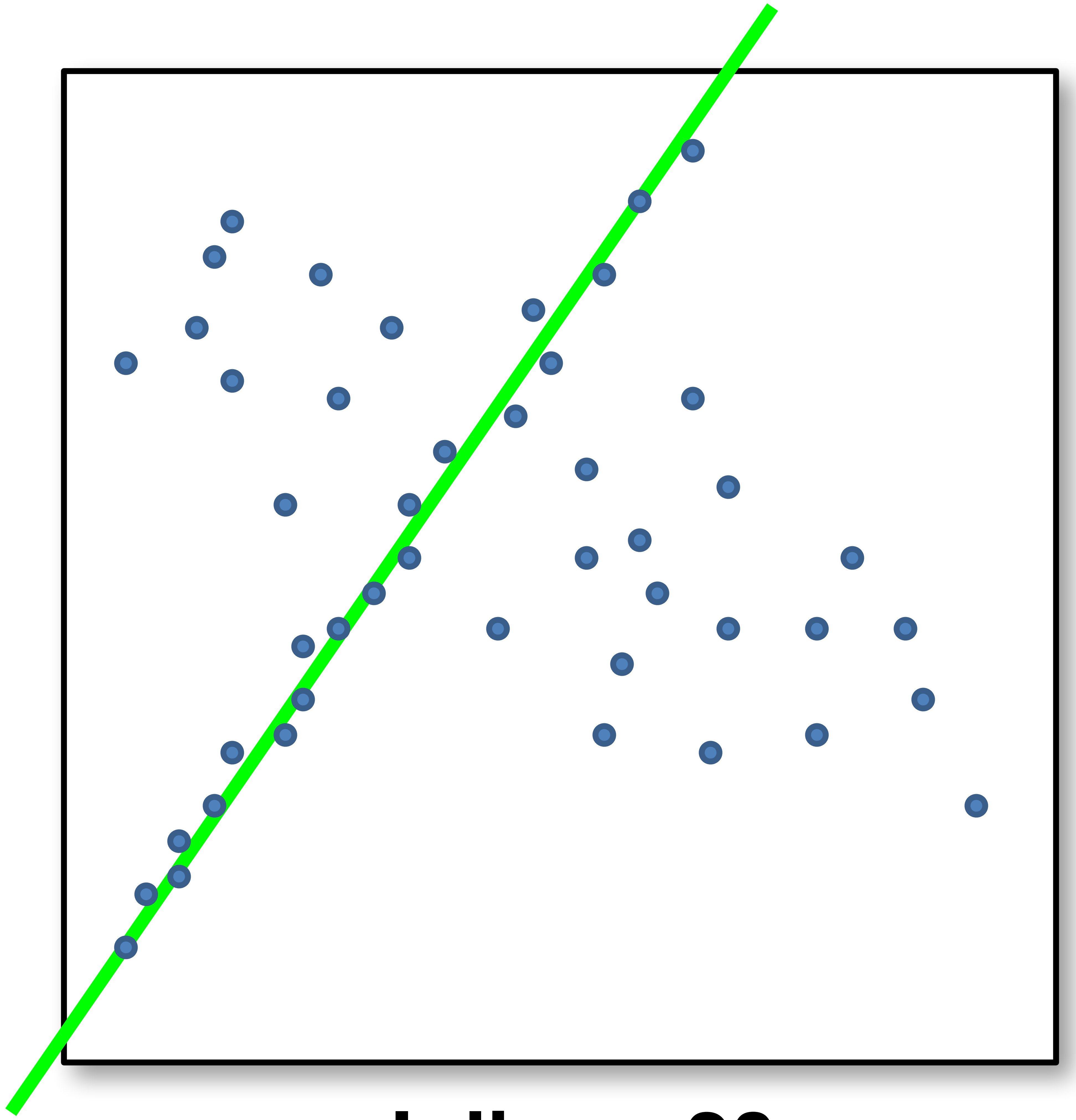
Counting inliers



Inliers: 3

Source: N. Snavely

Counting inliers



Inliers: 20

Source: N. Snavely

RANSAC

- Idea:
  - All the inliers will agree with each other on the solution; 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

Source: N. Snavely

RANSAC: random sample consensus

RANSAC loop (for N iterations):

- Select four feature pairs (at random)
- Compute homography H
- Count **inliers** where $||p_i' - f_H(p_i)|| < \varepsilon$

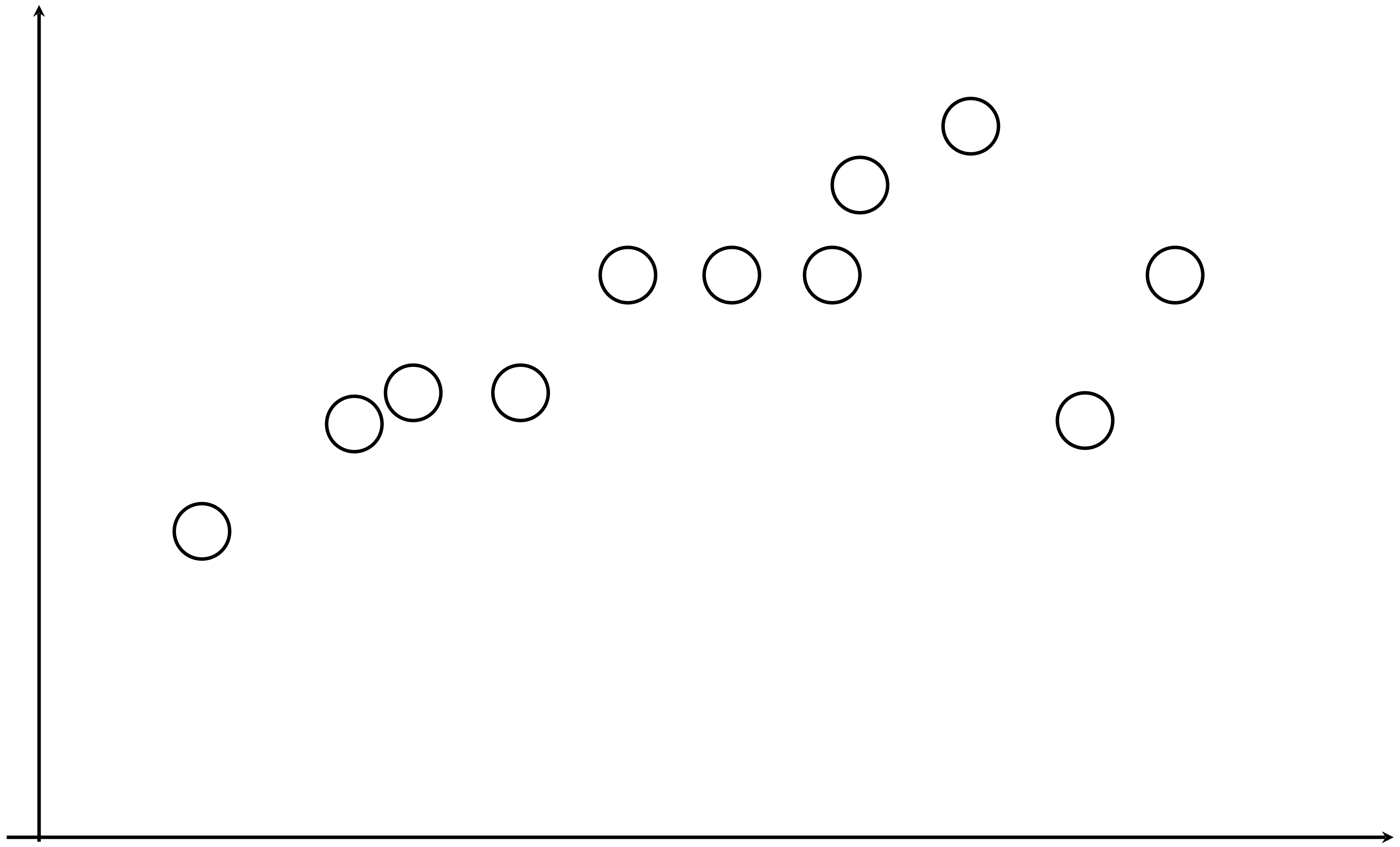
Afterwards:

- Choose H with largest set of inliers
- Recompute H using only those inliers (often using high-quality nonlinear least squares)

Source: Torralba, Freeman, Isola

Simple example: fit a line

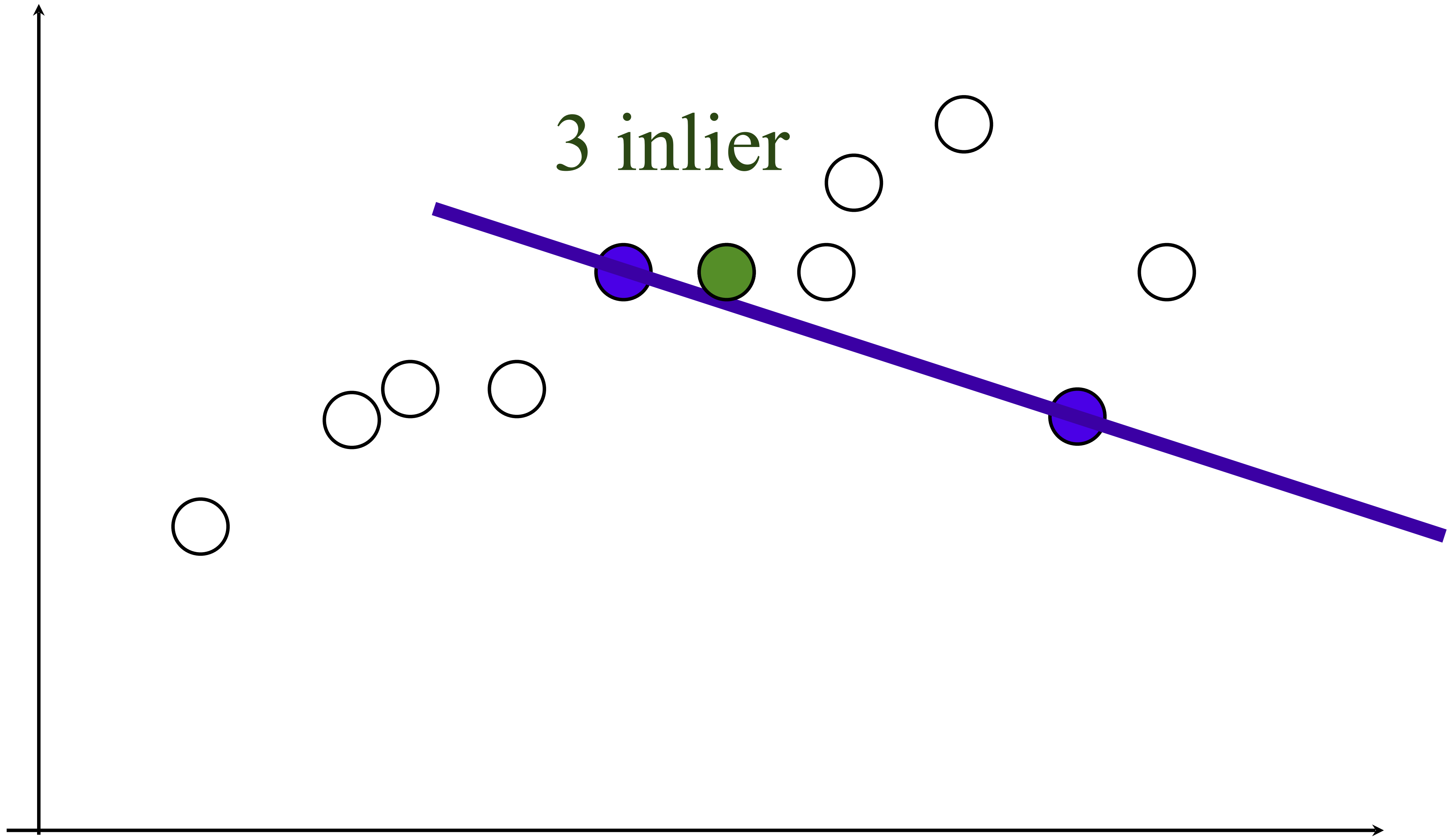
- Rather than homography H (8 numbers) fit y=ax+b (2 numbers a, b) to 2D pairs



Source: Torralba, Freeman, Isola

Simple example: fit a line

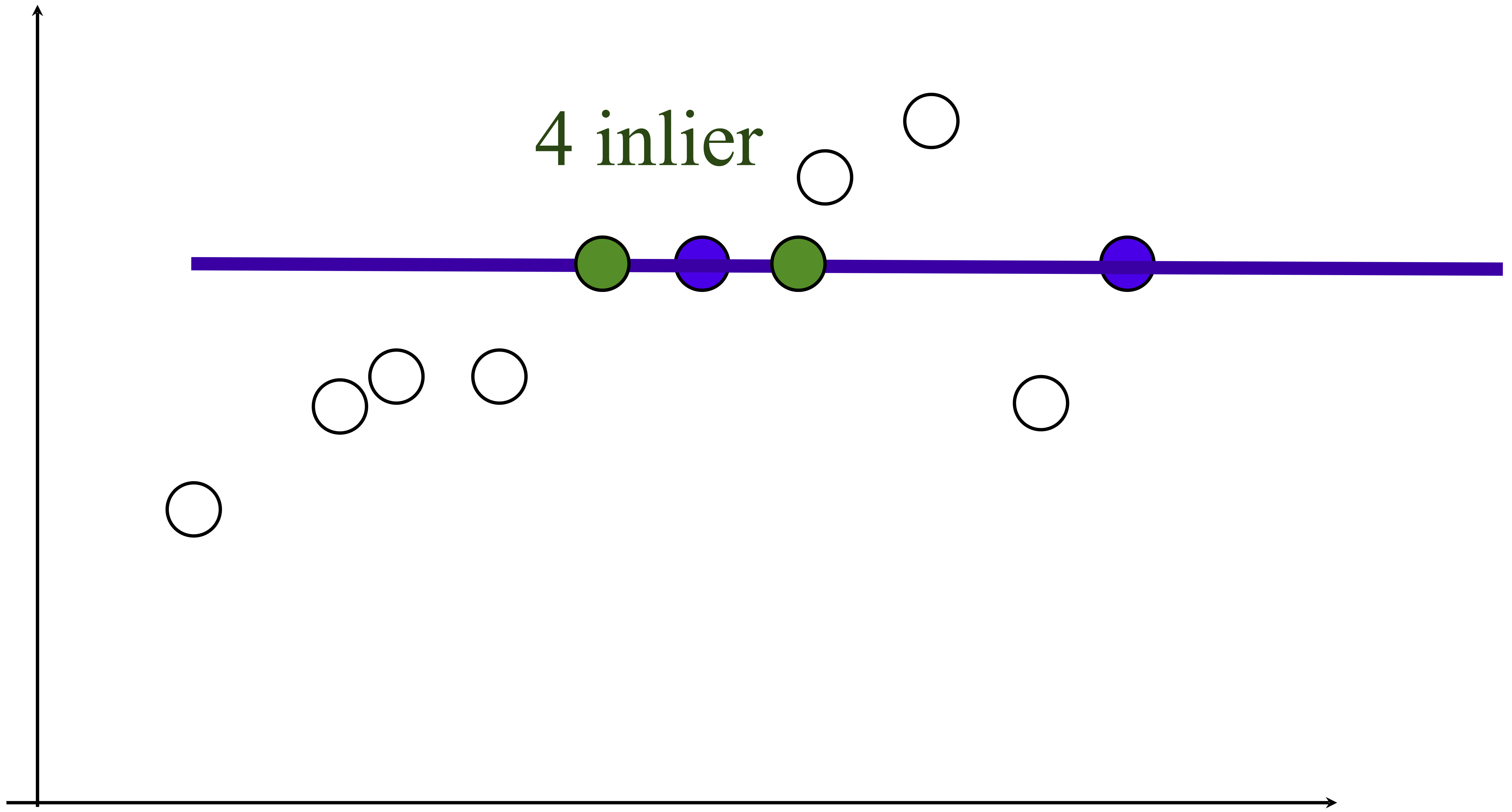
- Pick 2 points
- Fit line
- Count inliers



Source: Torralba, Freeman, Isola

Simple example: fit a line

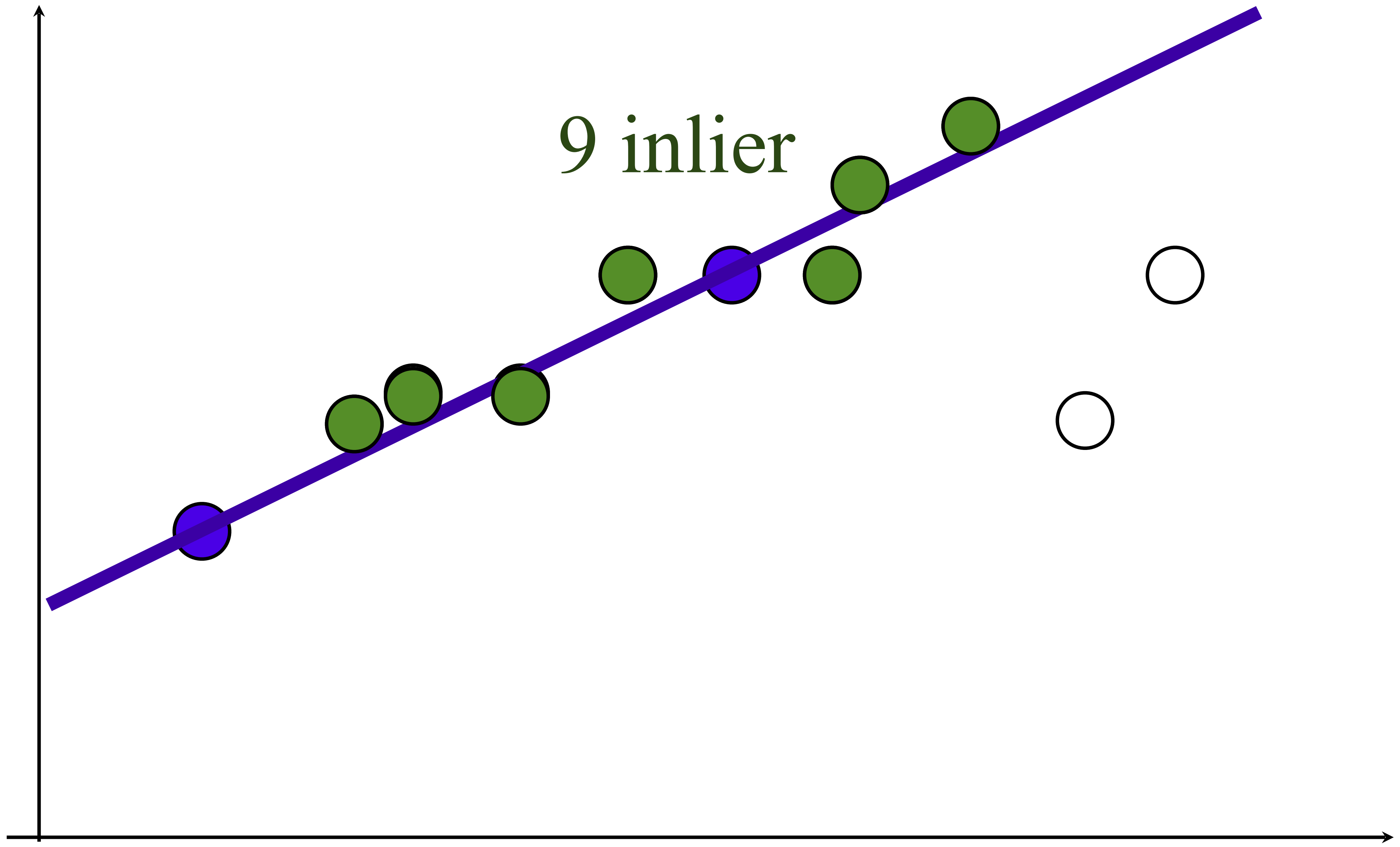
- Pick 2 points
- Fit line
- Count inliers



Source: Torralba, Freeman, Isola

Simple example: fit a line

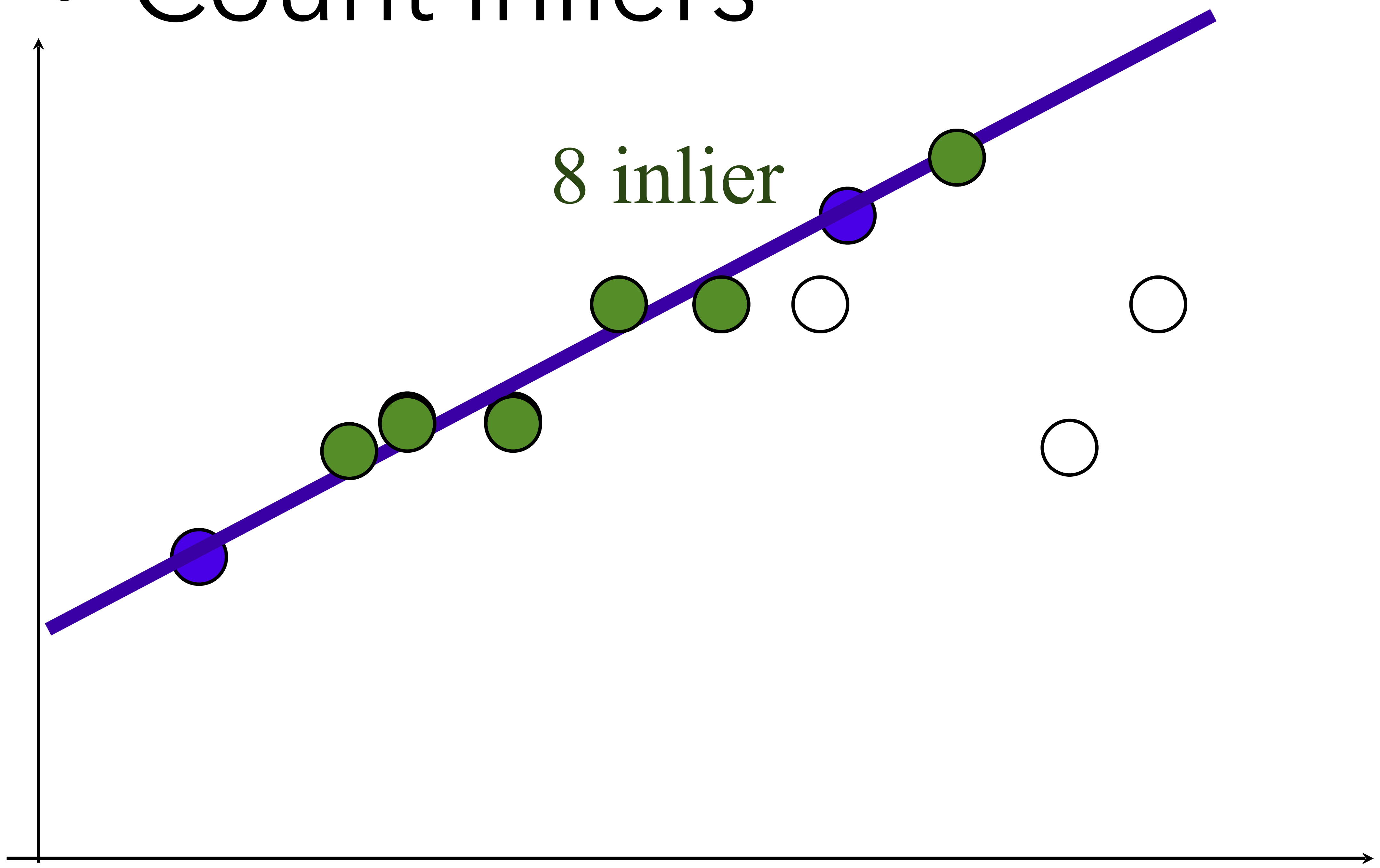
- Pick 2 points
- Fit line
- Count inliers



Source: Torralba, Freeman, Isola

Simple example: fit a line

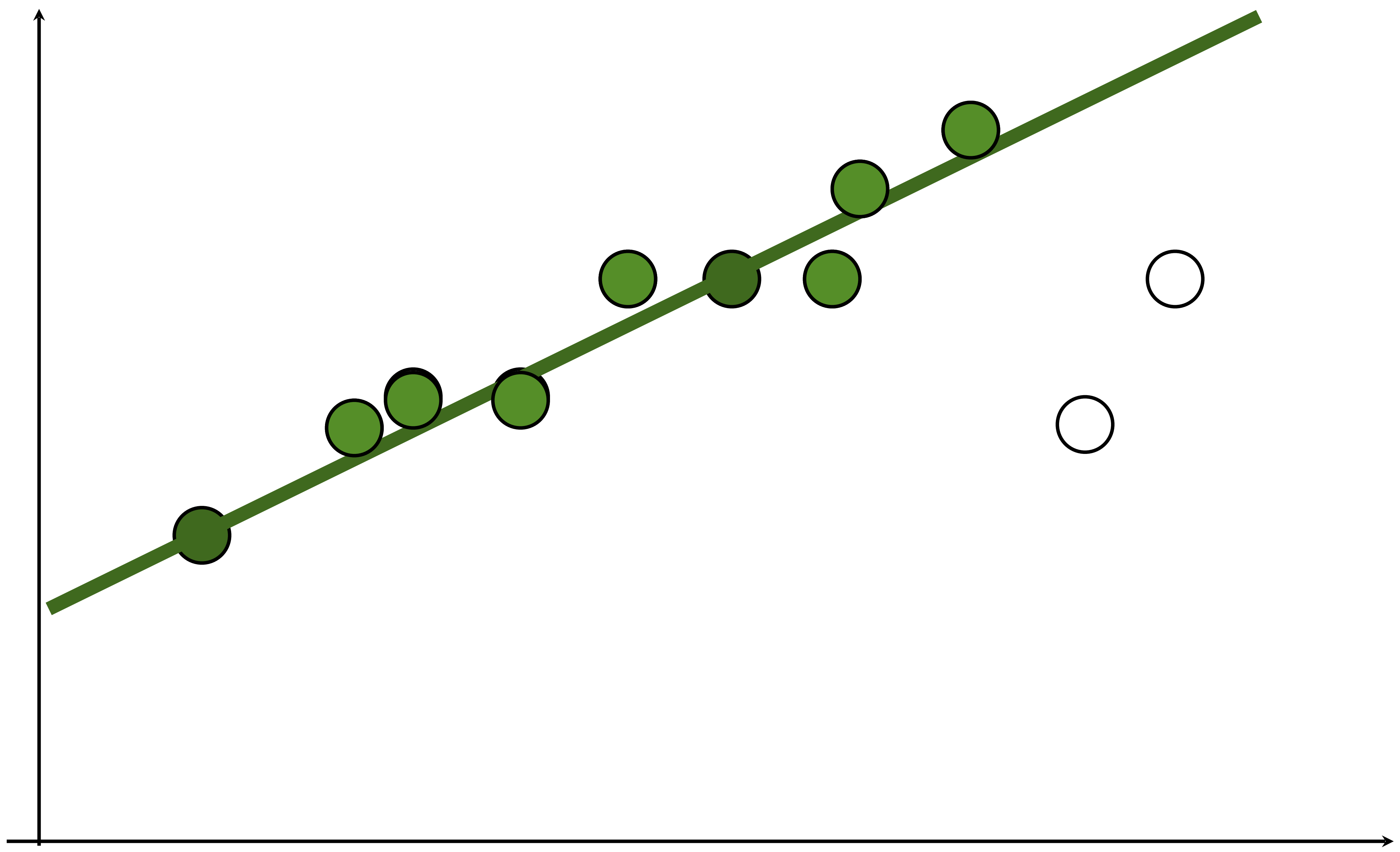
- Pick 2 points
- Fit line
- Count inliers



Source: Torralba, Freeman, Isola

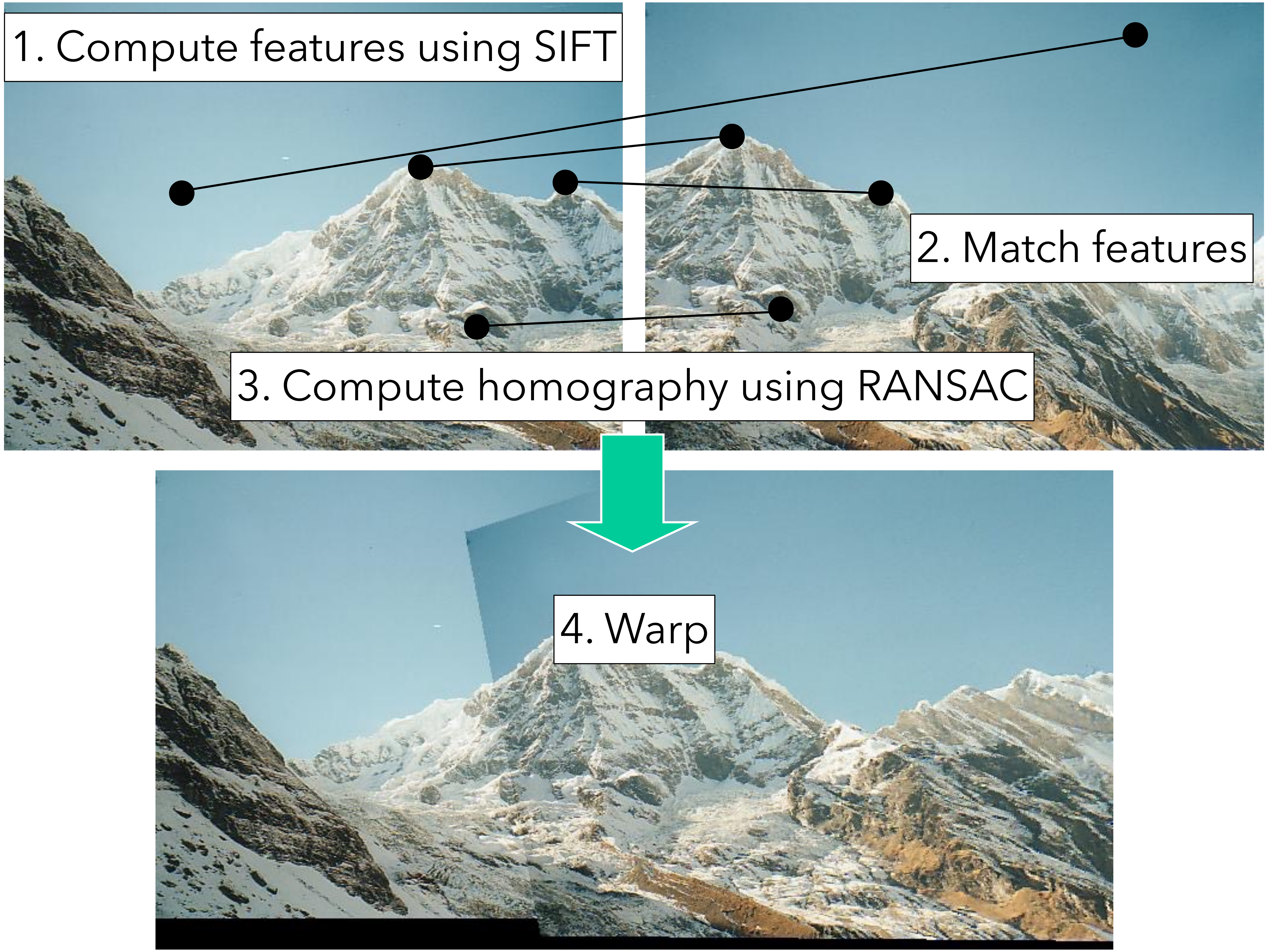
Simple example: fit a line

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



Source: Torralba, Freeman, Isola

Warping with a homography (PS5)

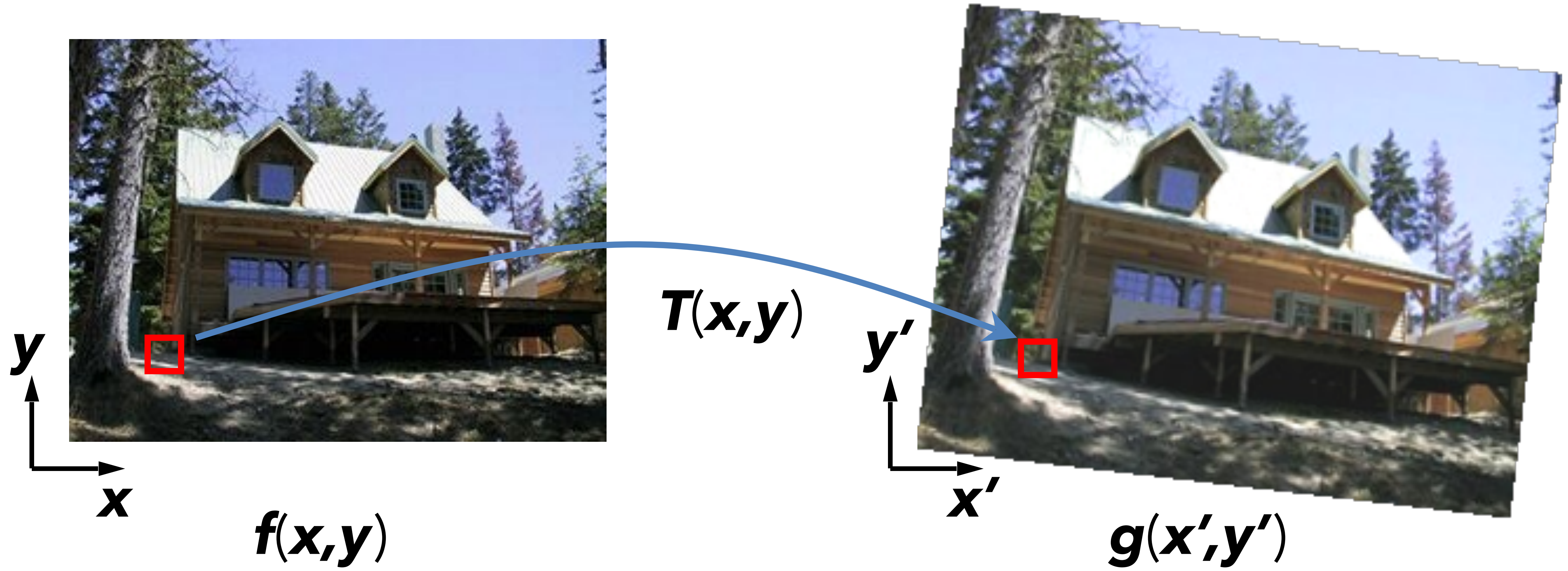


Source: N. Snavely

How do we perform this warp?

Image warping

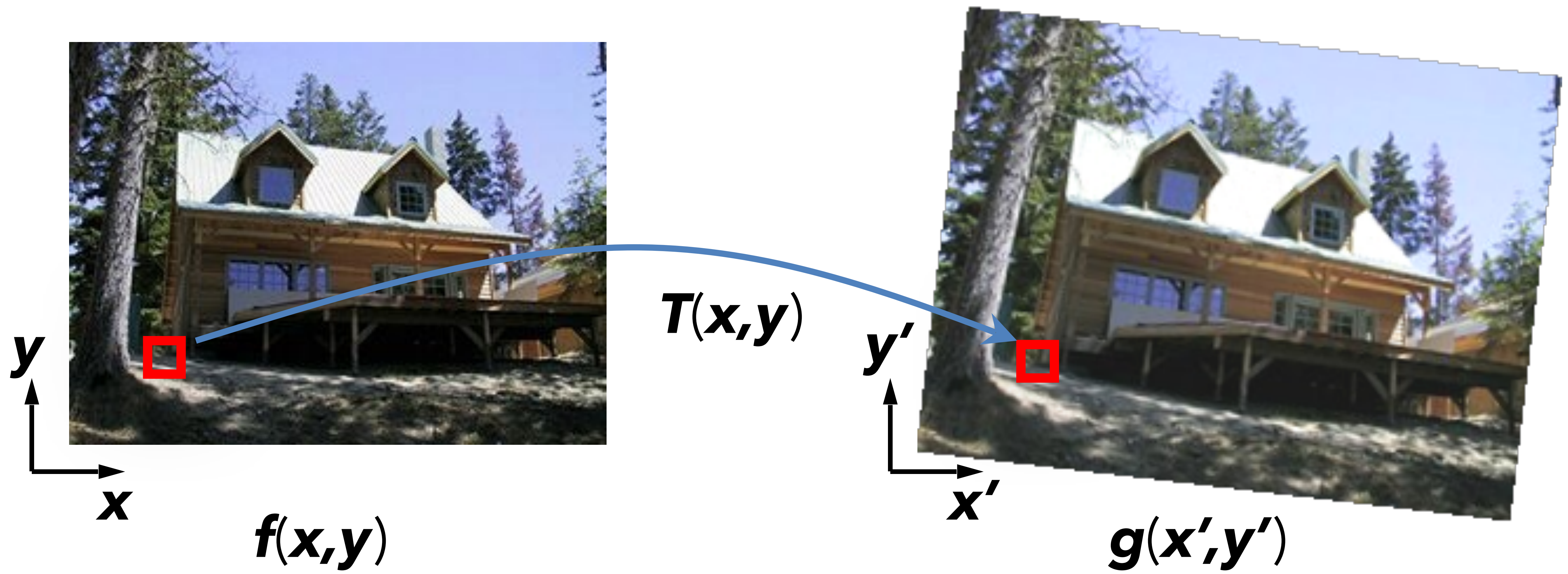
Given a coordinate transformation $(\boldsymbol{x}',\boldsymbol{y}') = \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y})$ and a source image $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$, how do we compute a transformed image $\boldsymbol{g}(\boldsymbol{x}',\boldsymbol{y}') = \boldsymbol{f}(\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y}))$?



Source: N. Snavely

Forward warping

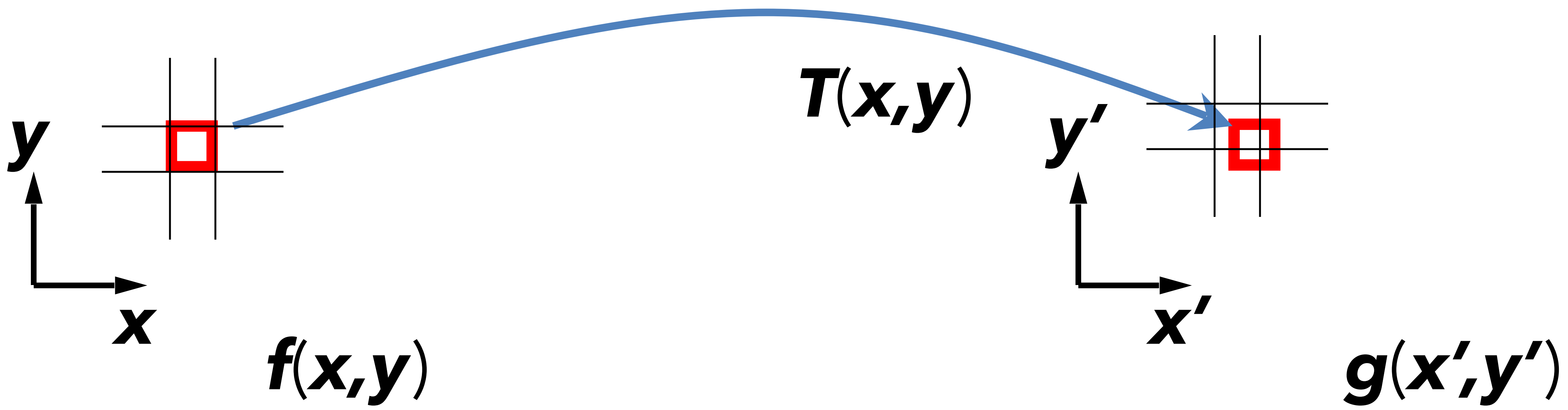
- Send each pixel $\boldsymbol{f}(\boldsymbol{x})$ to its corresponding location $(\boldsymbol{x'},\boldsymbol{y'}) = \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y})$ in $\boldsymbol{g}(\boldsymbol{x'},\boldsymbol{y'})$
- What if a pixel lands "between" two pixels?



Source: N. Snavely

Forward warping

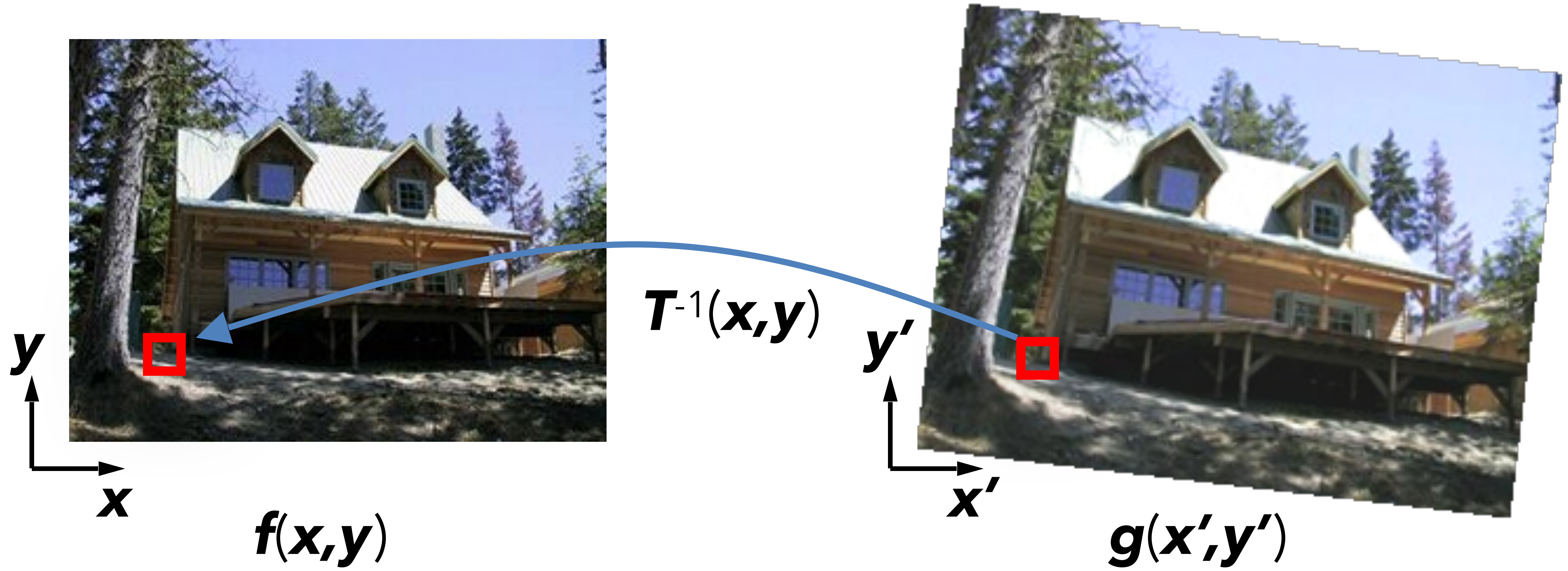
- Send each pixel $\boldsymbol{f}(\boldsymbol{x})$ to its corresponding location $(\boldsymbol{x'},\boldsymbol{y'}) = \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y})$ in $\boldsymbol{g}(\boldsymbol{x'},\boldsymbol{y'})$
- What if a pixel lands "between" two pixels?
  - Answer: add "contribution" to several pixels, normalize later (*splatting*)
  - Can still result in holes



Source: N. Snavely

Backward (inverse) warping

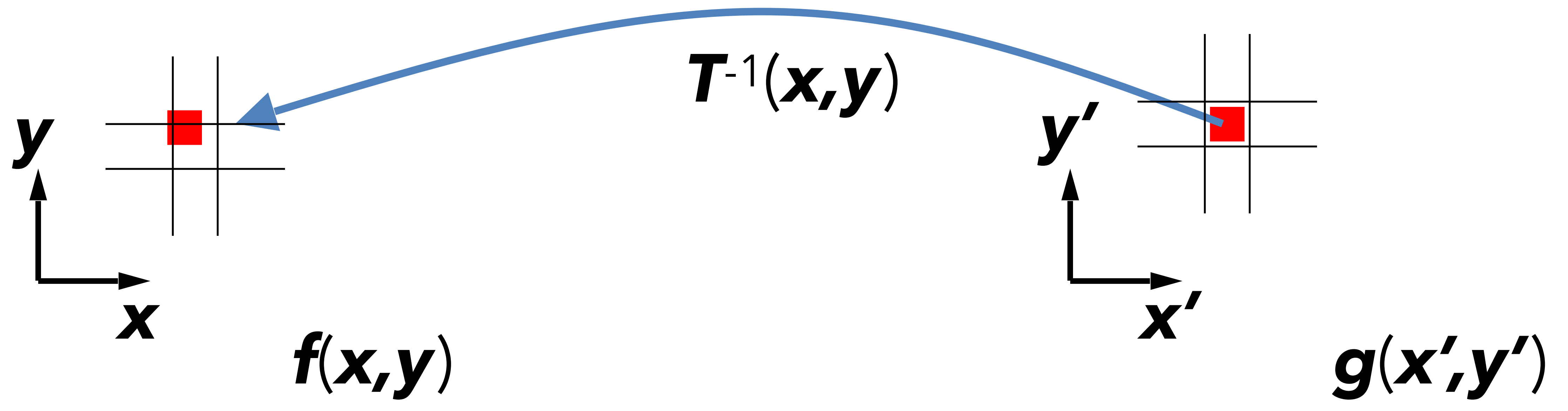
- Get each pixel $\boldsymbol{g}(\boldsymbol{x'},\boldsymbol{y'})$ from its corresponding location $(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{T}^{-1}(\boldsymbol{x},\boldsymbol{y})$ in $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$
  - Requires taking the inverse of the transform
  - What if pixel comes from "between" two pixels?



Source: N. Snavely

Backward (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?
  - Answer: *resample* color value from *interpolated* source image

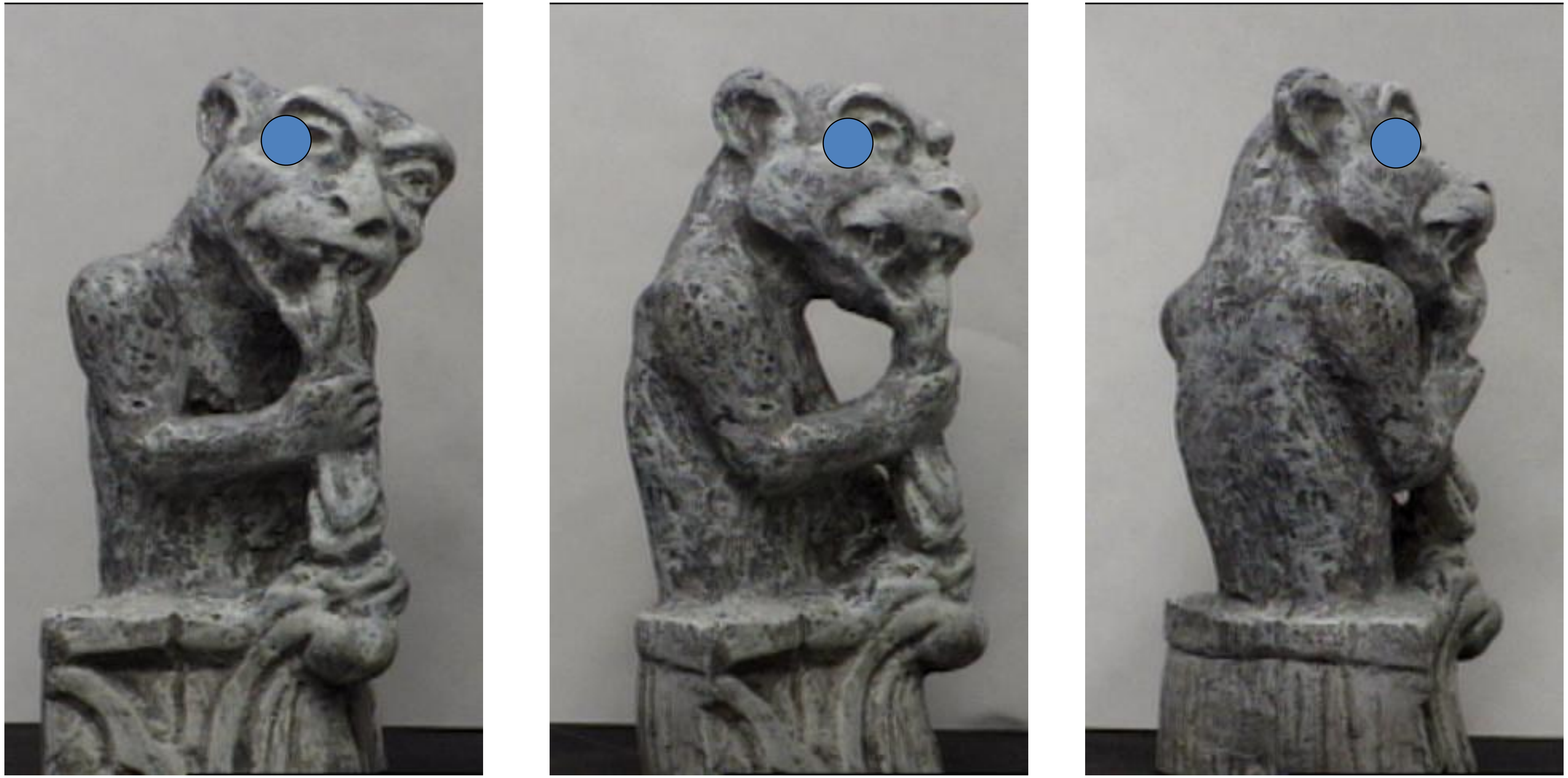


Source: N. Snavely

Other geometric problems

A similar geometric problem: triangulation

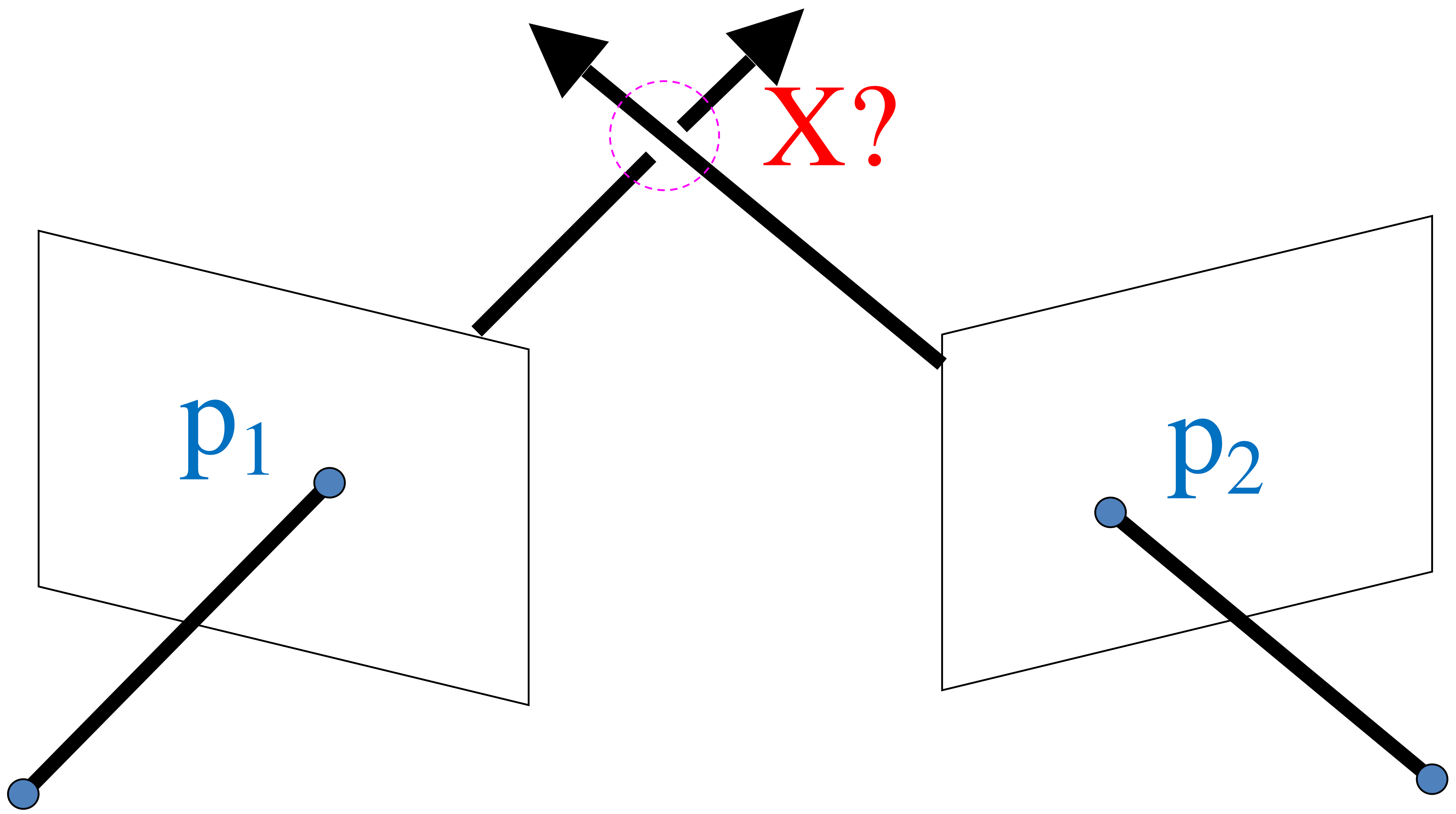
Given projection $\mathbf{p}_i$ of unknown 3D point $\mathbf{X}$ in two or more images (with known cameras $\mathbf{P}_i$), find $\mathbf{X}$



Source: D. Fouhey

Triangulation

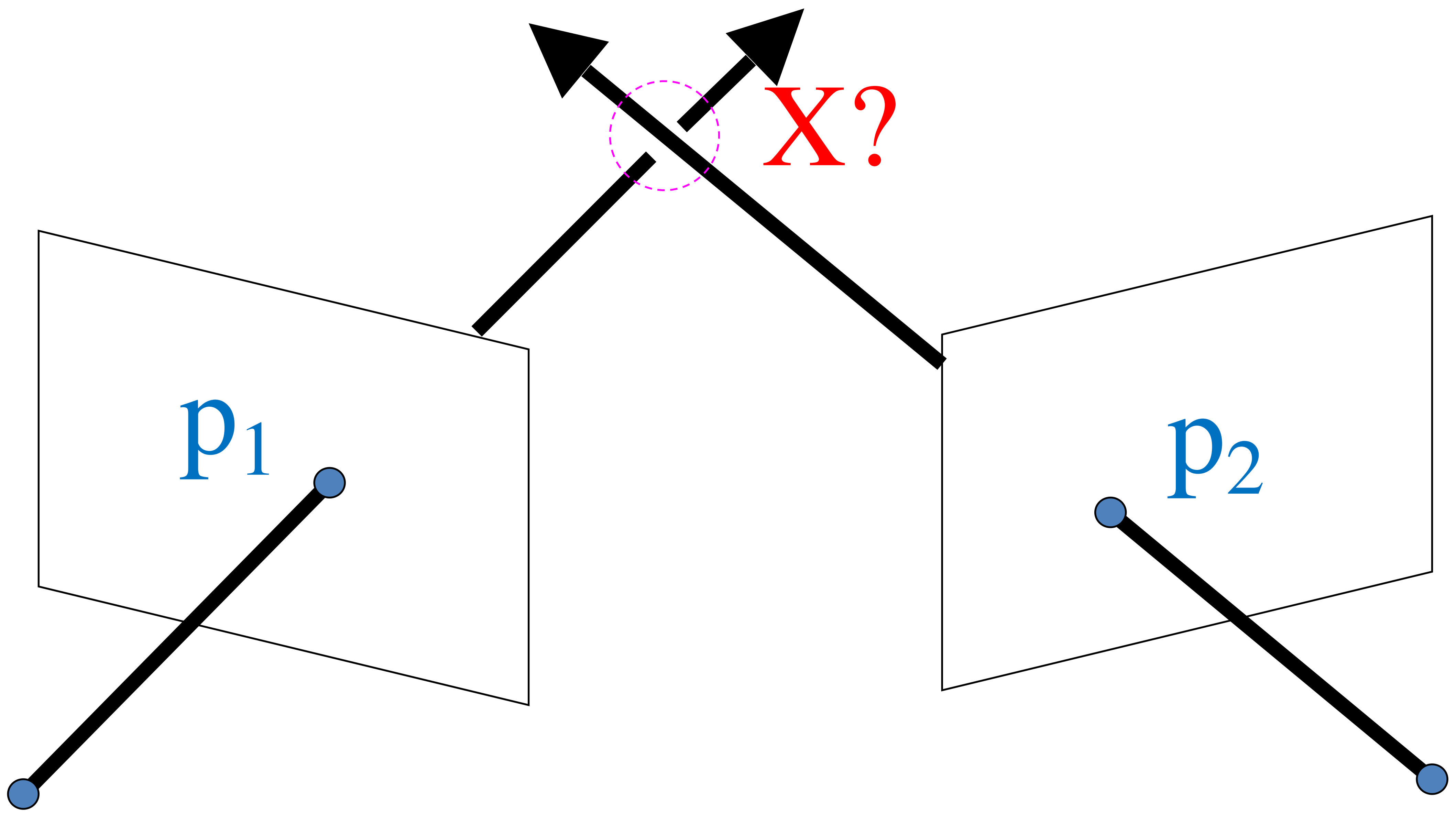
Given projection $\mathbf{p}_i$ of unknown 3D point $\mathbf{X}$ in two or more images (with known camera projection matrices $\mathbf{P}_i$), find $\mathbf{X}$



Source: D. Fouhey

Triangulation

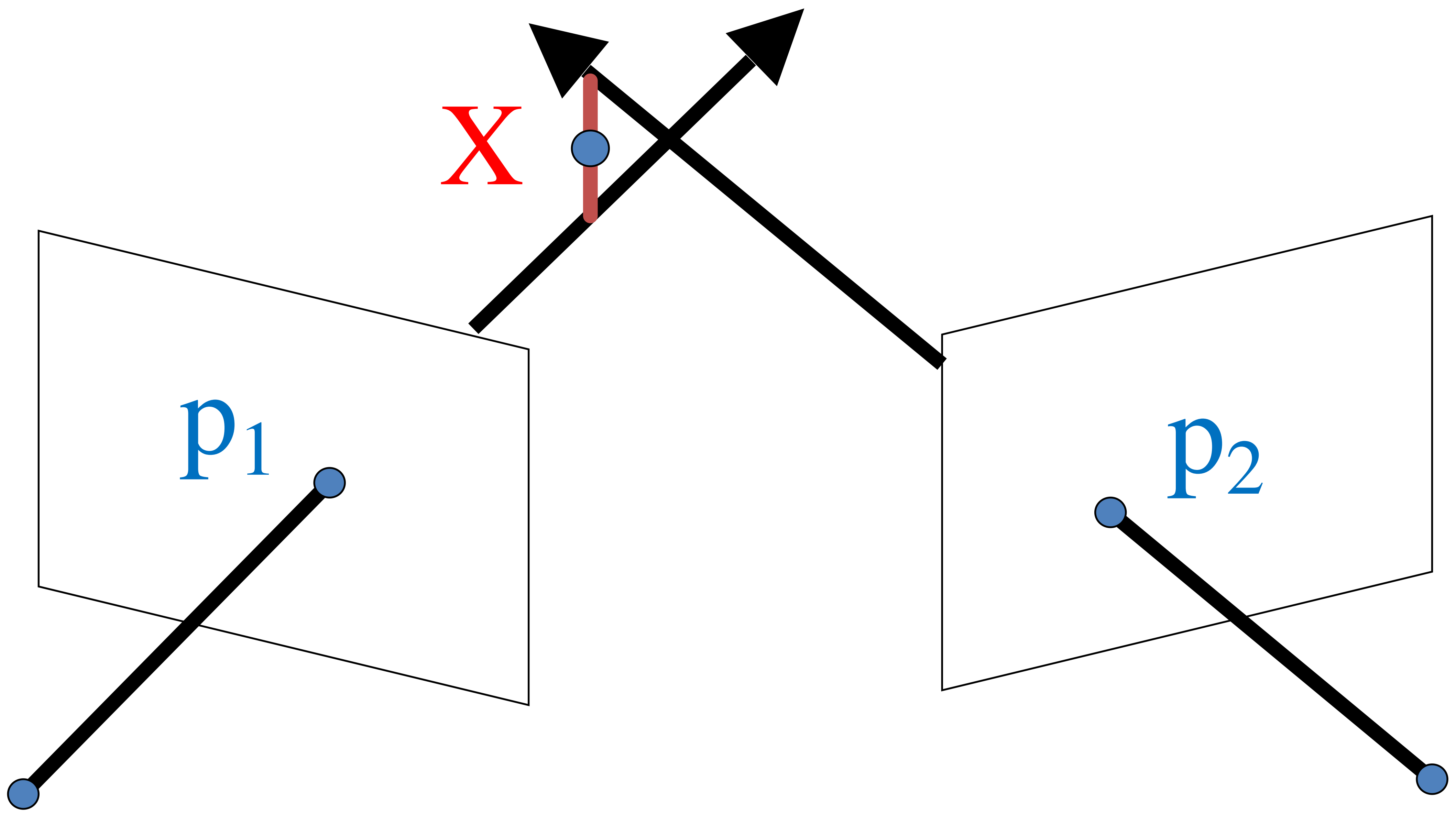
Rays in principle should intersect, but in practice usually don't exactly due to noise, numerical errors.



Source: D. Fouhey

Triangulation – Geometry

Find shortest segment between viewing rays, set **X** to be the midpoint of the segment.

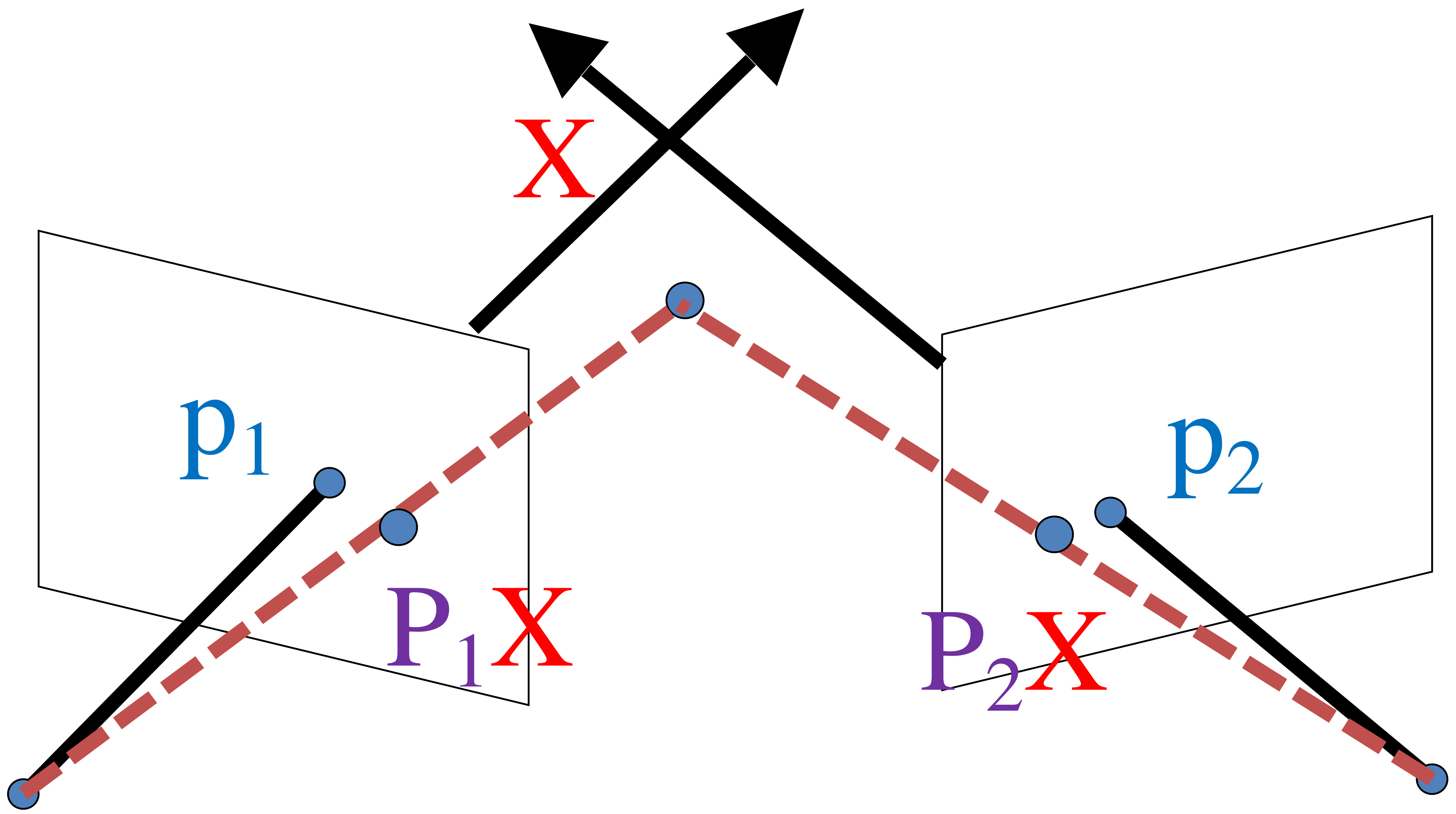


Source: D. Fouhey

Triangulation – Non-linear Optim.

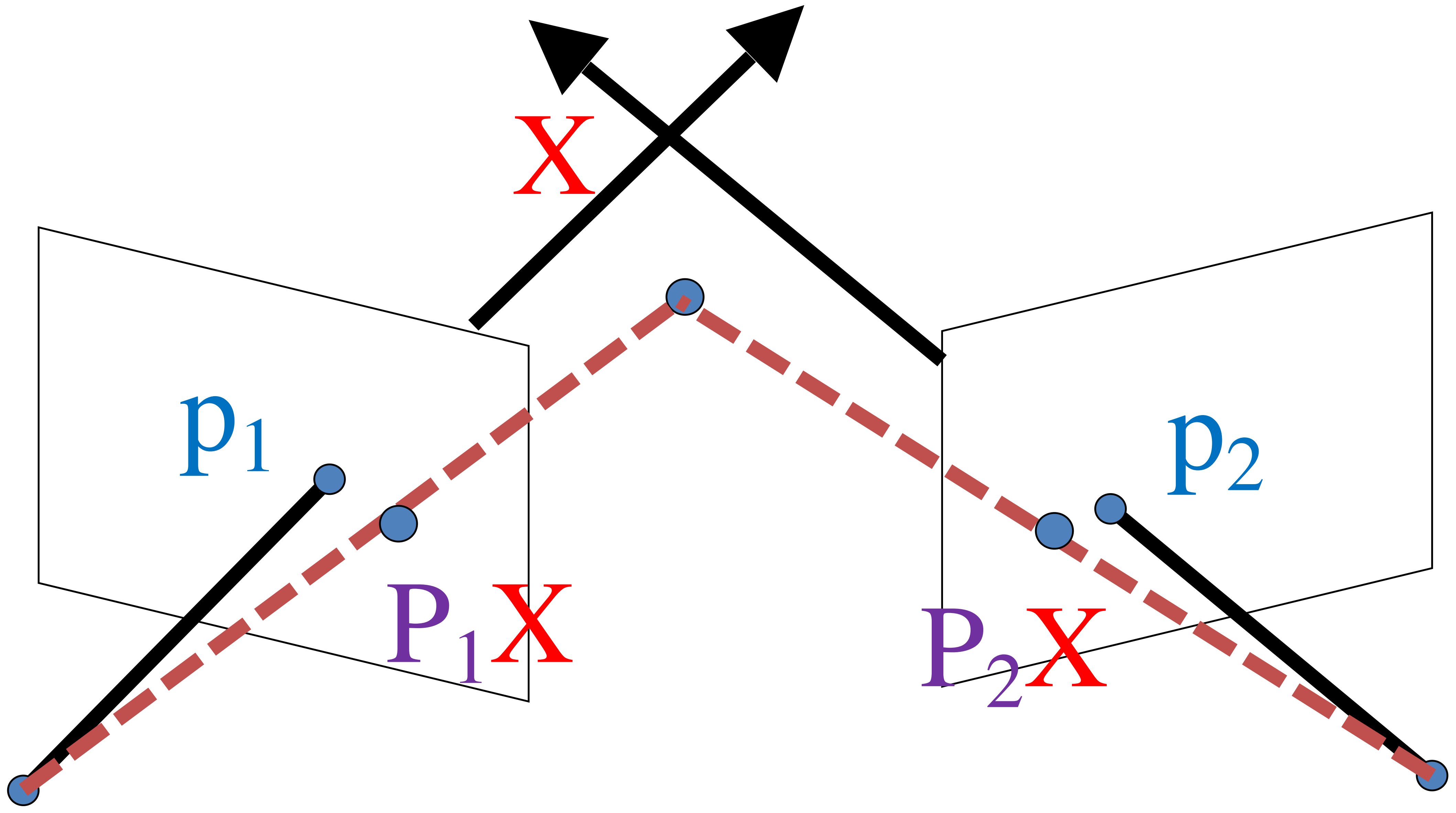
Find X minimizing $d(\boldsymbol{p}_1, P_1\boldsymbol{X})^2 + d(\boldsymbol{p}_2, P_2\boldsymbol{X})^2$

where d is distance in image space



Source: D. Fouhey

Triangulation – Linear Optimization



Source: D. Fouhey

First: A better way to handle homogeneous coordinates in linear optimization

Projection in homogeneous coordinates.

$$\boldsymbol{p}_i \equiv P\boldsymbol{X}_i$$

i.e., **PX$_i$ & p$_i$** are proportional/scaled copies of each other

$$\boldsymbol{p}_i = \lambda P\boldsymbol{X}_i, \ \lambda \neq 0$$

This implies their cross product is **0**, since $a \times b = \|a\| \|b\| \sin(\theta)$.

$$\boldsymbol{p}_i \times P\boldsymbol{X}_i = \mathbf{0}$$

Handles the "divide by 0" issue when solving.

Source: D. Fouhey

Triangulation – Linear Optimization

$$p_1 \equiv P_1\mathbf{X} \qquad p_2 \equiv P_2\mathbf{X}$$

$$\rightarrow \quad p_1 \times P_1\mathbf{X} = \mathbf{0} \qquad p_2 \times P_2\mathbf{X} = \mathbf{0}$$

$$\rightarrow \quad [p_{1x}]P_1\mathbf{X} = \mathbf{0} \qquad [p_{2x}]P_2\mathbf{X} = \mathbf{0}$$

Cross product as matrix

$$\boldsymbol{a} \times \boldsymbol{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = [\boldsymbol{a}_x]\boldsymbol{b}$$

$$[p_{1x}]P_1\mathbf{X} = \mathbf{0} \qquad [p_{2x}]P_2\mathbf{X} = \mathbf{0}$$

$$\rightarrow \quad ([p_{1x}]P_1)\mathbf{X} = \mathbf{0} \qquad ([p_{2x}]P_2)\mathbf{X} = \mathbf{0}$$

$\rightarrow$ Two equations per camera for 3 unknown in X

Source: D. Fouhey

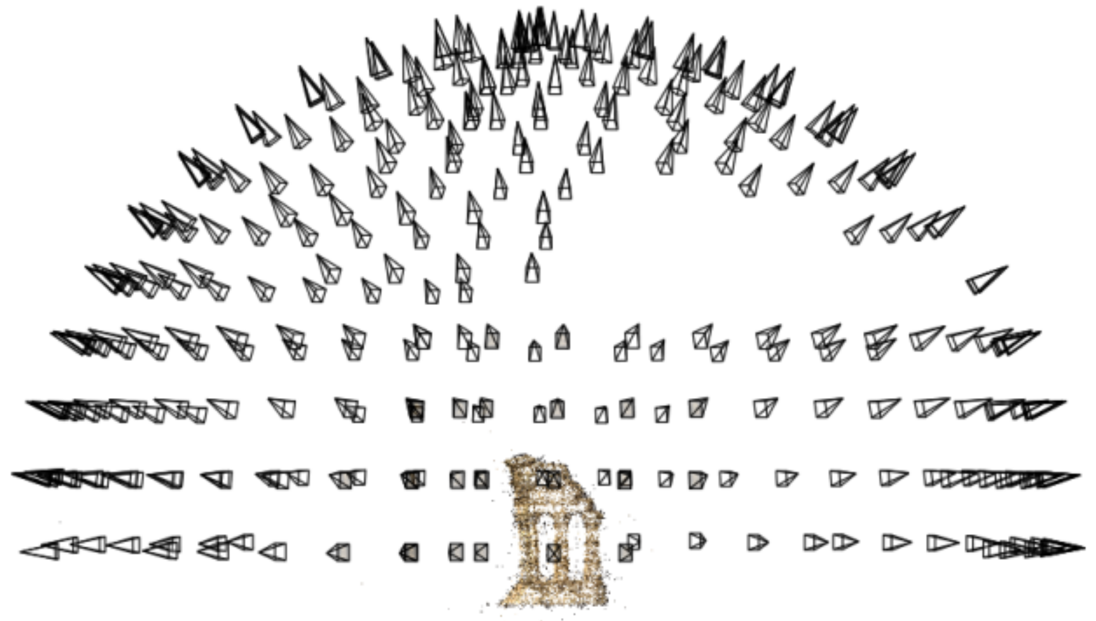
Next time: Estimating 3D structure

- Given many images, how can we…
  1. Figure out where they were all taken from?
  2. Build a 3D model of the scene?

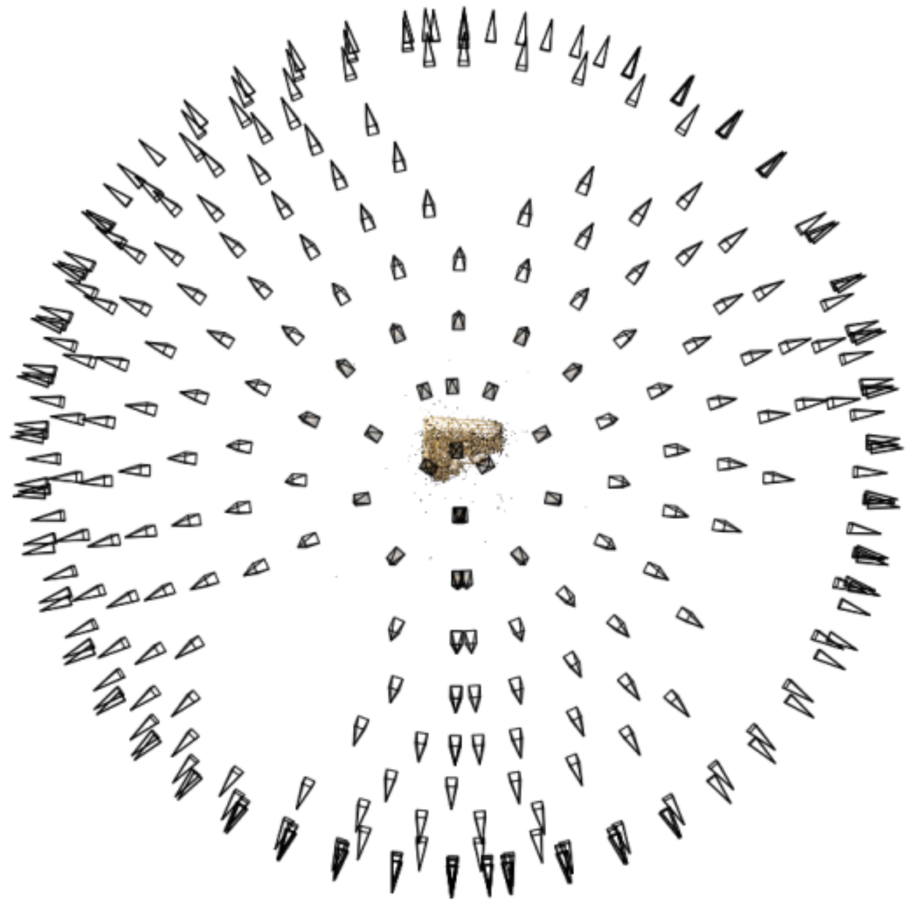
This is the **structure from motion** problem.

Source: N. Snavely

Structure from motion



Reconstruction (side)



(top)

- Input: images with pixels in correspondence $p_{i,j} = (u_{i,j}, v_{i,j})$

- Output
  - **Structure:** 3D location $\mathbf{x}_i$ for each point p_i
  - **Motion:** camera parameters $\mathbf{R}_j$, $\mathbf{t}_j$ possibly $\mathbf{K}_j$

- Objective function: minimize reprojection error

Source: N. Snavely

Camera calibration & triangulation

- Suppose we know 3D points
  - And have matches between these points and an image
  - Computing camera parameters similar to homography estimation

- Suppose we have know camera parameters, each of which observes a point
  - We can solve for the 3D location

- Seems like a chicken-and-egg problem, but in SfM we can solve both at once!

Source: N. Snavely

Next class: more 3D