Image Alignment

A lot of slides from Darrell and Szeliski

Roadmap

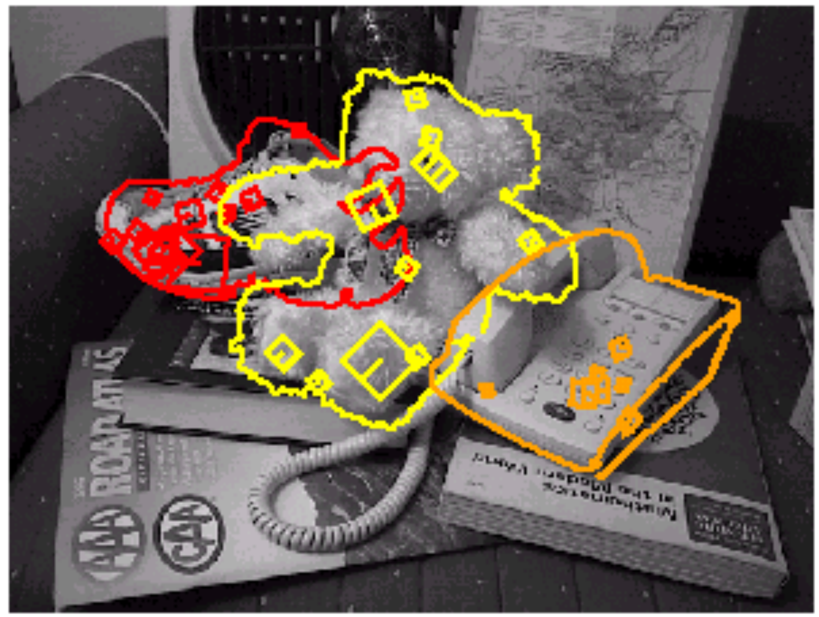
- Previous: Image formation, filtering, local features, connected components, object features, matching, …
- **Today: Feature-based Alignment**
  - **Stitching images together**
  - **Homographies, RANSAC, Warping, Blending**
  - **Global alignment of planar models**

Today: Alignment

- Homographies
- Rotational Panoramas
- RANSAC
- Global alignment
- Warping
- Blending

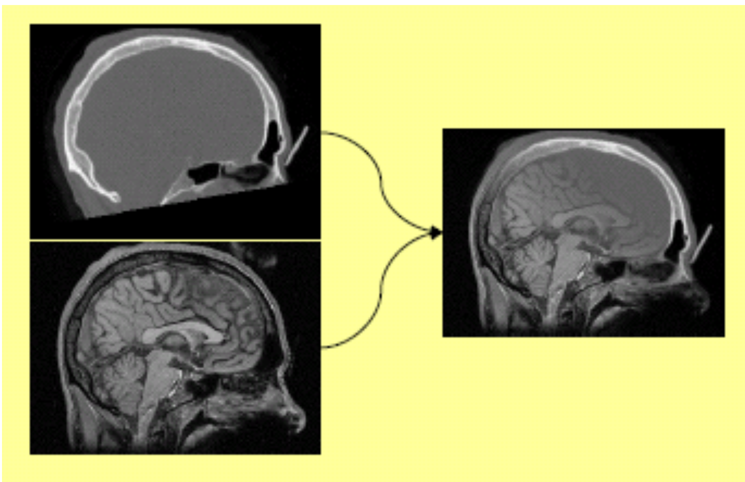
(a) (b) (c)

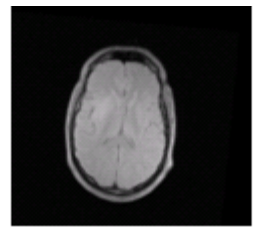
Motivation: Recognition

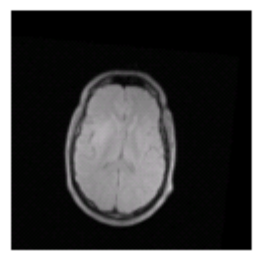


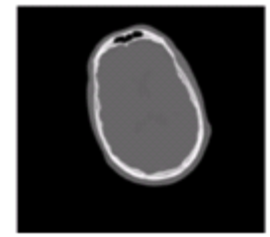
Figures from David Lowe

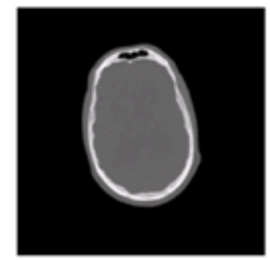
Motivation: medical image registration

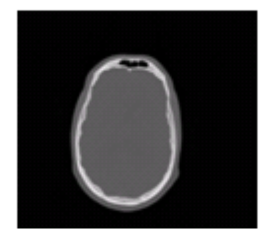












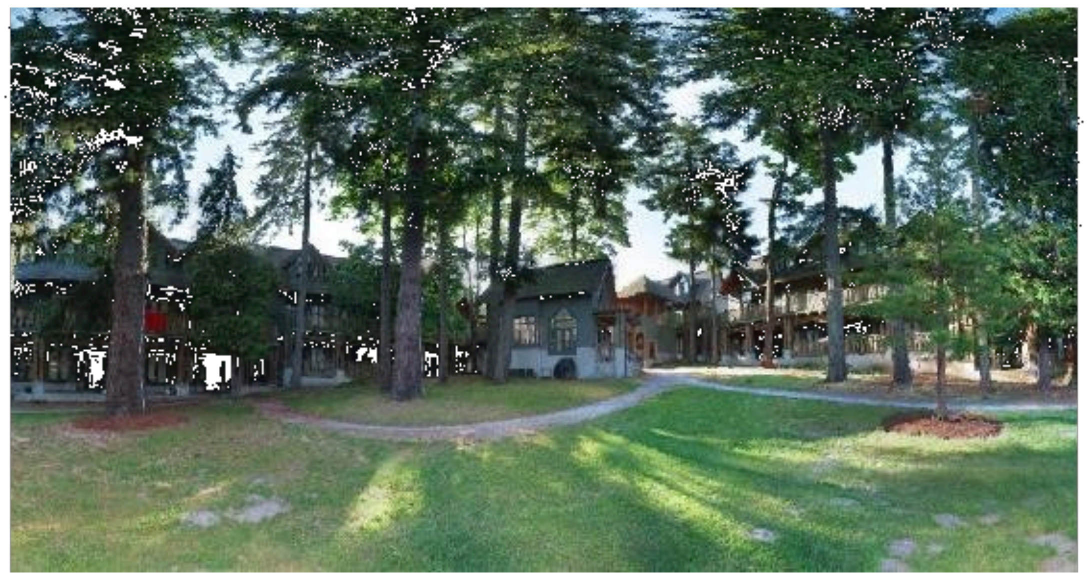
Motivation: Mosaics

- Getting the whole picture
  - Consumer camera: 50˚ x 35˚

Slide from Brown & Lowe 2003

Motivation: Mosaics

- Getting the whole picture
  - Consumer camera: 50˚ x 35˚
  - Human Vision: 176˚ x 135˚



Slide from Brown & Lowe 2003

Motivation: Mosaics

- Getting the whole picture
  - Consumer camera: 50˚ x 35˚
  - Human Vision: 176˚ x 135˚

- Panoramic Mosaic = up to 360 x 180°

Slide from Brown & Lowe 2003

Motion models

Motion models

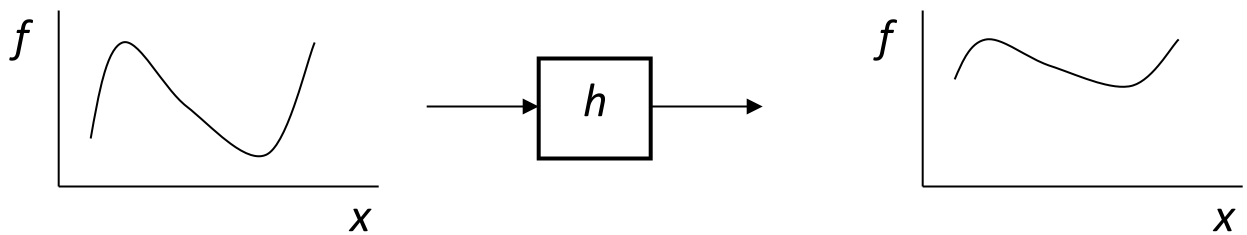
- What happens when we take two images with a camera and try to align them?
- translation?
- rotation?
- scale?
- affine?
- perspective?
- … see interactive demo (VideoMosaic)

Szeliski

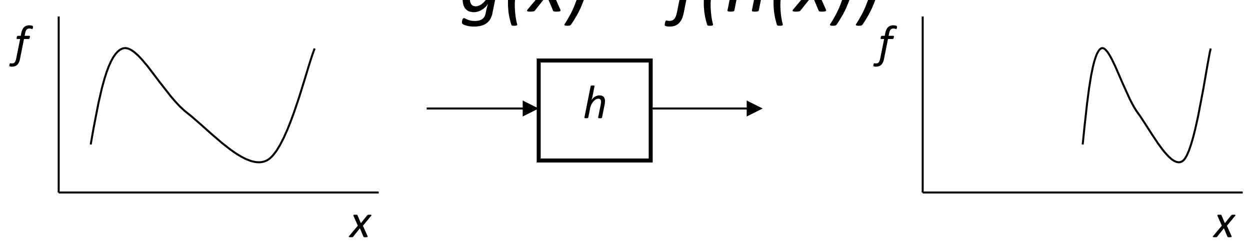
Image Warping

Image Warping

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



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

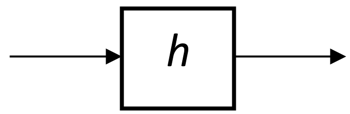


Szeliski

Image Warping

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

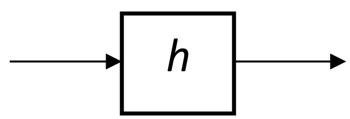
f



g

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

f



g

Szeliski

Parametric (global) warping

- Examples of parametric warps:

translation

rotation

aspect

affine

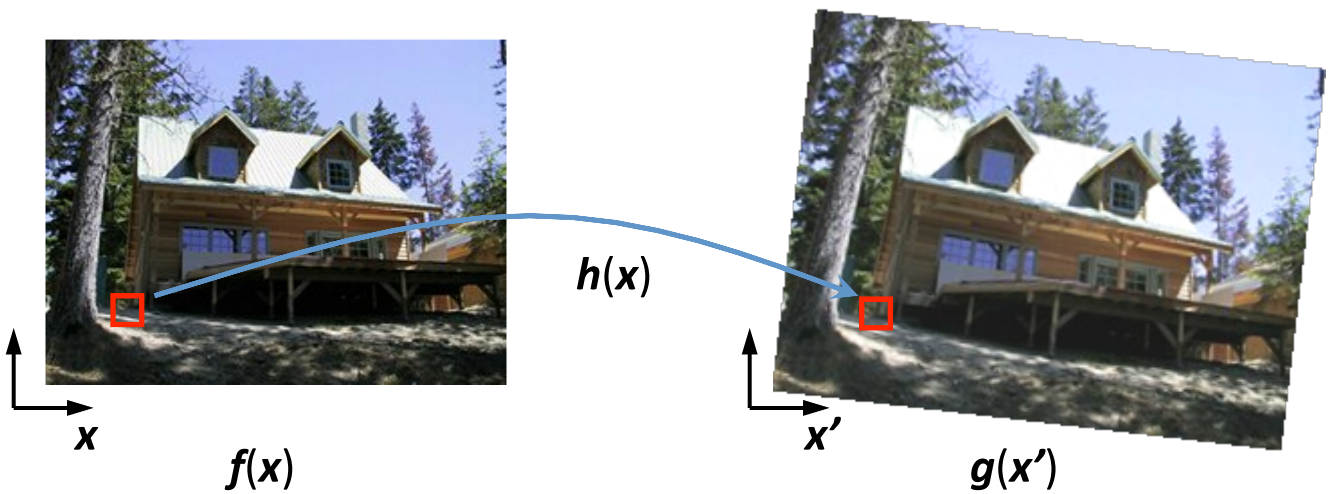
perspective

cylindrical

Szeliski

Image Warping

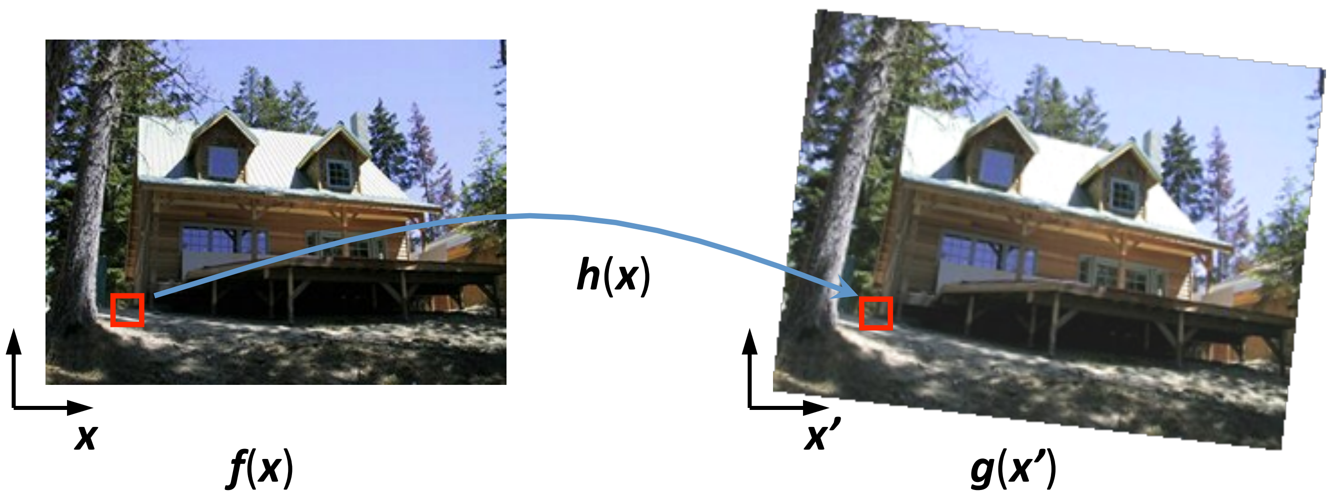
- Given a coordinate transform $\boldsymbol{x'} = \boldsymbol{h}(\boldsymbol{x})$ and a source image $\boldsymbol{f}(\boldsymbol{x})$, how do we compute a transformed image $\boldsymbol{g}(\boldsymbol{x'}) = \boldsymbol{f}(\boldsymbol{h}(\boldsymbol{x}))$?



Szeliski

Forward Warping

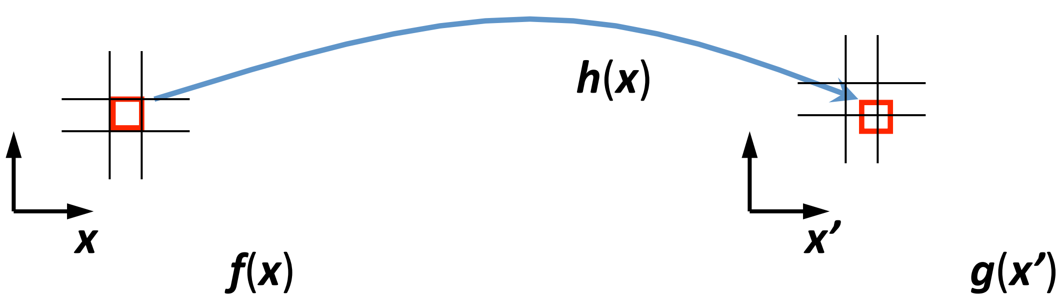
- Send each pixel $\boldsymbol{f}(\boldsymbol{x})$ to its corresponding location $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$ in $\boldsymbol{g}(\boldsymbol{x}')$
  - What if pixel lands "between" two pixels?



Szeliski

Forward Warping

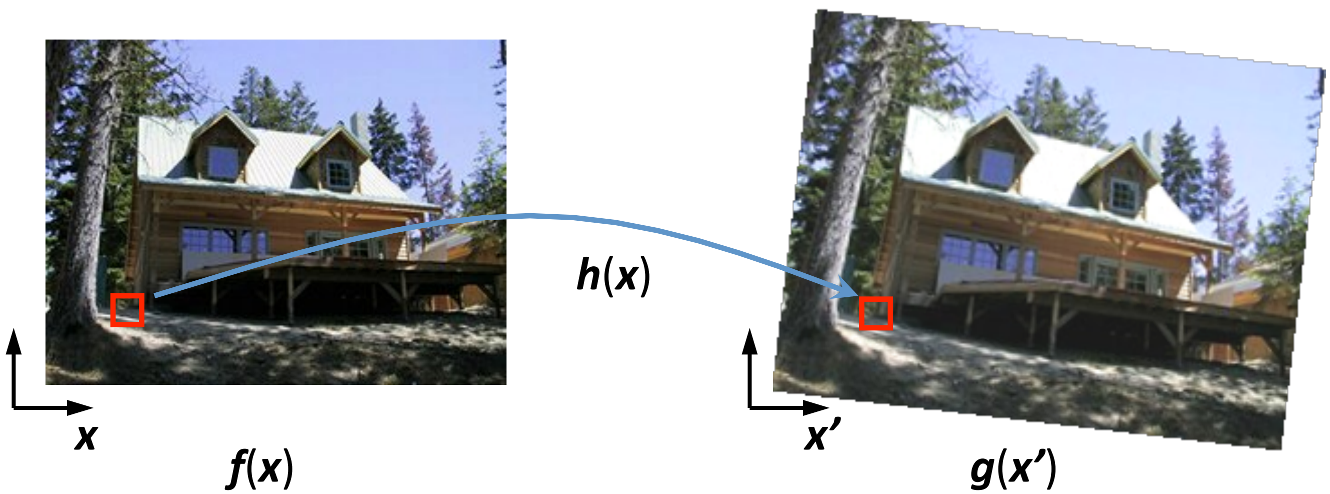
- Send each pixel ***f***(***x***) to its corresponding location ***x'*** = ***h***(***x***) in ***g***(***x'***)
  - What if pixel lands "between" two pixels?
  - Answer: add "contribution" to several pixels, normalize later (*splatting*)



Szeliski

Inverse Warping

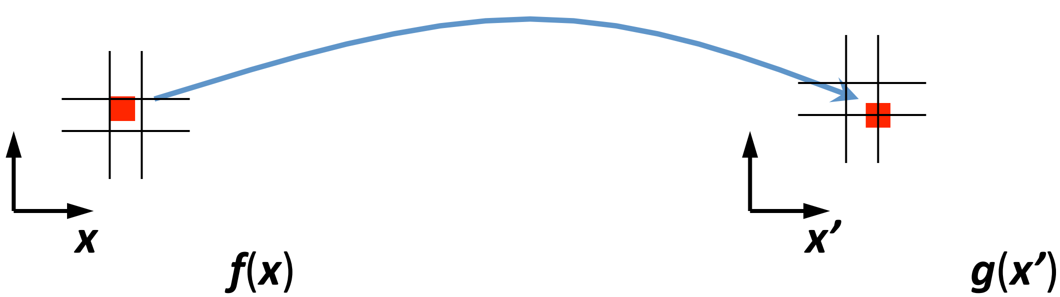
- Get each pixel ***g***(***x'***) from its corresponding location ***x'*** = ***h***(***x***) in ***f***(***x***)
  - What if pixel comes from "between" two pixels?



Szeliski

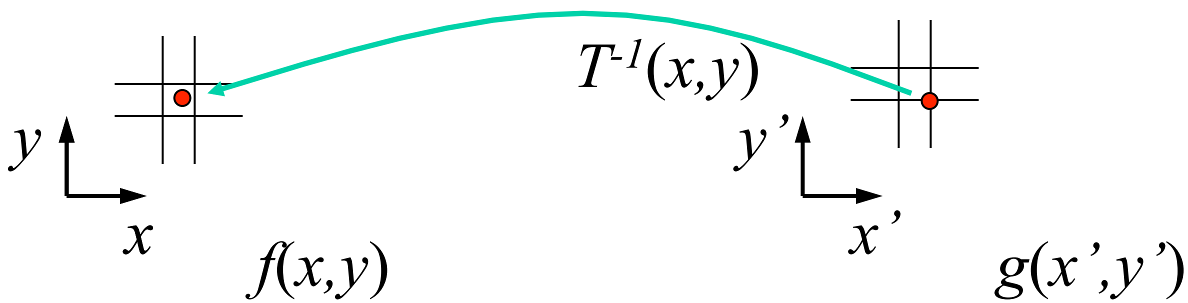
Inverse Warping

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



Szeliski

Inverse warping



Get each pixel $g(x',y')$ from its corresponding location

$(x,y) = T^{-1}(x',y')$ in the first image

Q: what if pixel comes from "between" two pixels?

A: *Interpolate* color value from neighbors

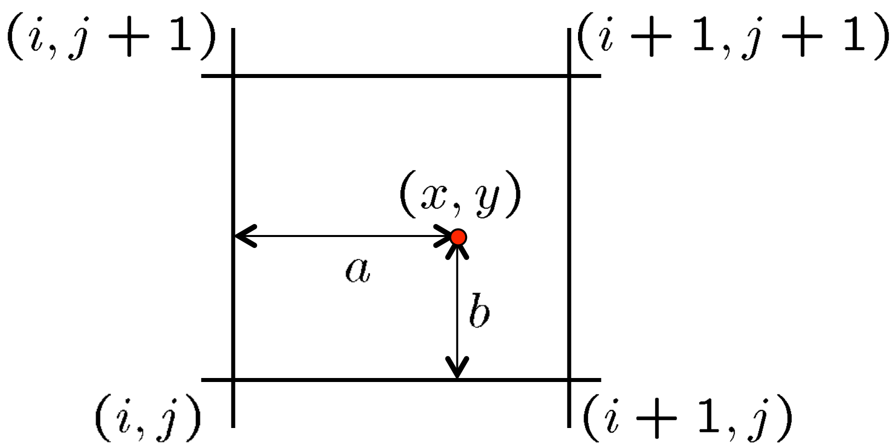
- nearest neighbor, bilinear…

`>> help interp2`

Slide from Alyosha Efros, CMU

Bilinear interpolation

Sampling at *f(x,y):*



$$
\begin{aligned}
f(x,y) = \quad & (1-a)(1-b) && f[i,j] \\
& +a(1-b) && f[i+1,j] \\
& +ab && f[i+1,j+1] \\
& +(1-a)b && f[i,j+1]
\end{aligned}
$$

Slide from Alyosha Efros, CMU

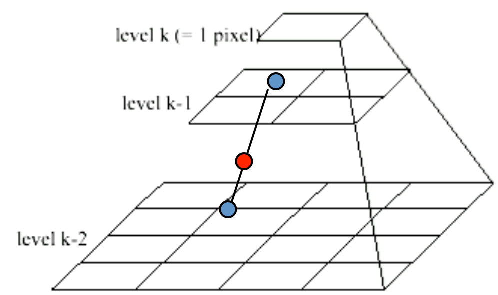
Interpolation

- Possible interpolation filters
  - nearest neighbor
  - bilinear
  - bicubic (interpolating)
  - sinc / FIR
- Needed to prevent "jaggies" and "texture crawl"

Szeliski

Prefiltering

- Essential for *downsampling* (*decimation*) to prevent *aliasing*
- MIP-mapping [Williams'83]:
  1. build pyramid (but what decimation filter?):
     - block averaging
     - Burt & Adelson (5-tap binomial)
     - 7-tap wavelet-based filter (better)
  2. *trilinear* interpolation
     - bilinear within each 2 adjacent levels
     - linear blend *between* levels (determined by pixel size)



2D coordinate transformations

- translation: $\boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{t}$ $\boldsymbol{x} = (x,y)$
- 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: $\underline{\boldsymbol{x}}' \cong \boldsymbol{H}\,\underline{\boldsymbol{x}}$ $\underline{\boldsymbol{x}} = (x,y,1)$
  ($\underline{\boldsymbol{x}}$ is a *homogeneous* coordinate)
- These all form a nested *group* (closed w/ inv.)

Szeliski

Basic 2D Transformations

Basic 2D transformations as 3x3 matrices

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

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

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

Rotate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh_x} & 0 \\ \boldsymbol{sh_y} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear

Source: Alyosha Efros

2D Affine Transformations

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

Affine transformations are combinations of …

- Linear transformations, and
- Translations

Parallel lines remain parallel

Grauman

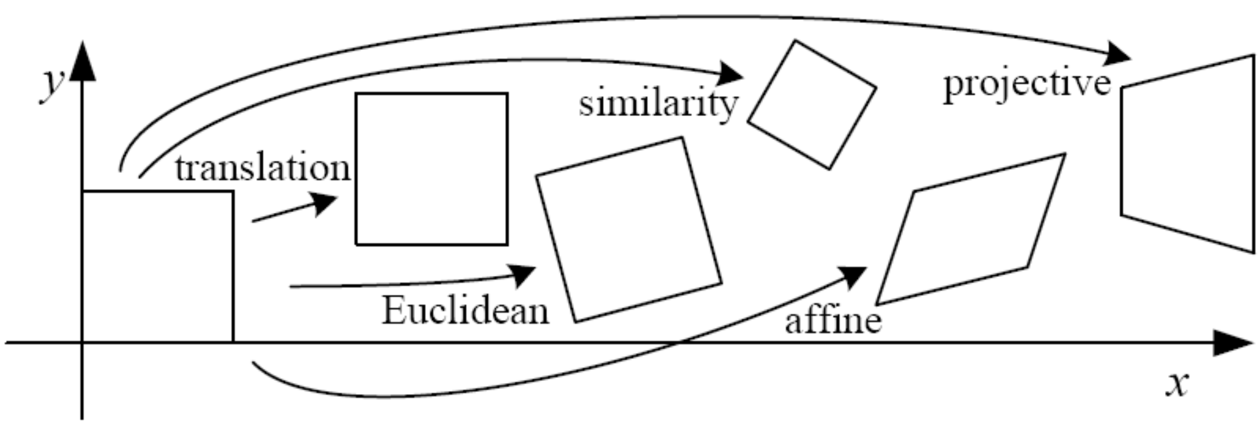
Projective Transformations

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

Projective transformations:

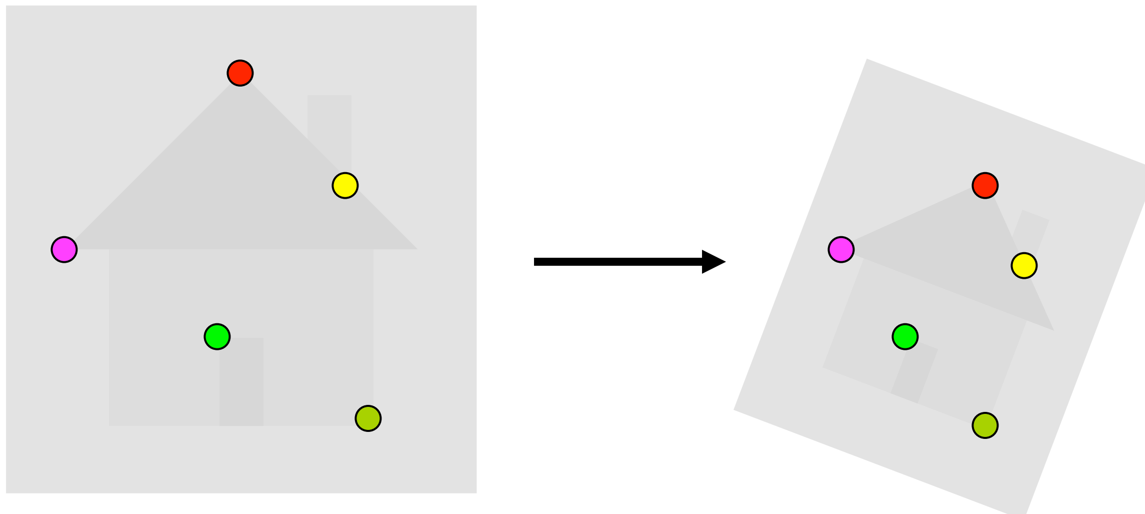
- Affine transformations, and
- Projective warps

Parallel lines do not necessarily remain parallel



Grauman

Image alignment



- Two broad approaches:
  - Direct (pixel-based) alignment
    - Search for alignment where most pixels agree
  - Feature-based alignment
    - Search for alignment where *extracted features* agree
    - Can be verified using pixel-based alignment

Source: L. Lazebnik

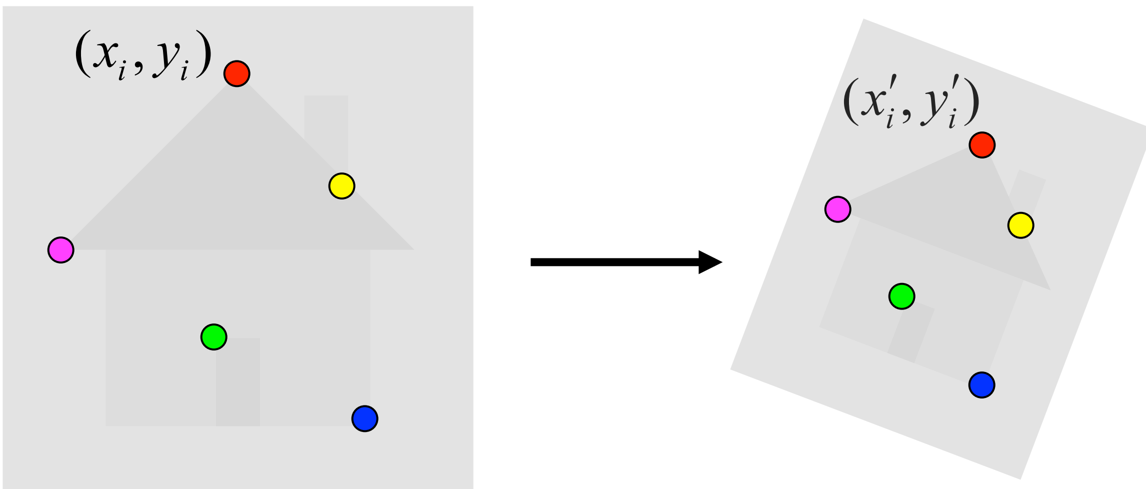
Fitting an affine transformation

Affine model approximates perspective projection of planar objects.

Figures from David Lowe, ICCV 1999

Fitting an affine transformation

- Assuming we know the correspondences, how do we get the transformation?

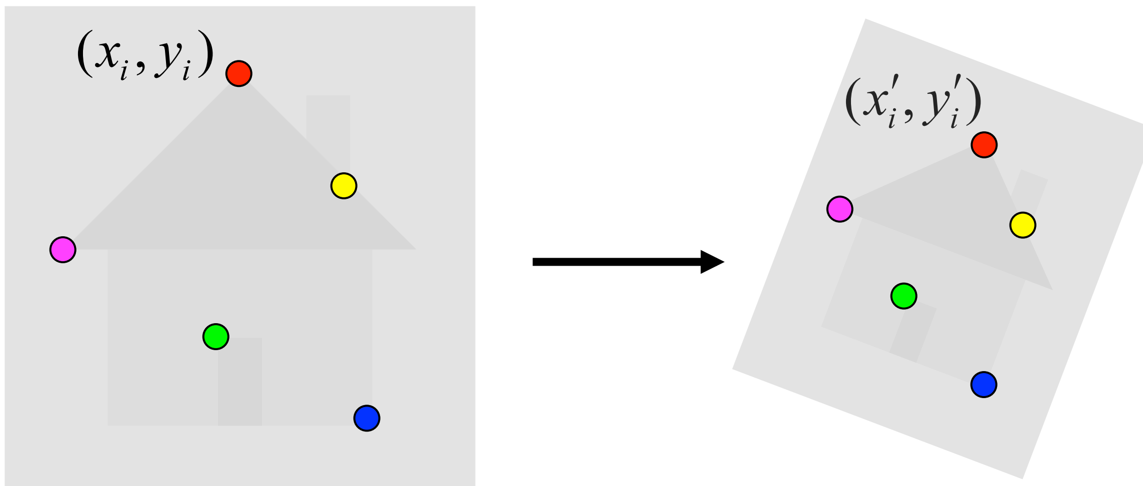


$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

Grauman

Fitting an affine transformation

- Assuming we know the correspondences, how do we get the transformation?



$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} m_1 & m_2 \\ m_3 & m_4 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

$$\begin{bmatrix} \ \\ \ \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \ \\ \ \end{bmatrix}$$

Fitting an affine transformation

$$\begin{bmatrix} & & \cdots & & & \\ x_i & y_i & 0 & 0 & 1 & 0 \\ 0 & 0 & x_i & y_i & 0 & 1 \\ & & \cdots & & & \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \cdots \\ x_i' \\ y_i' \\ \cdots \end{bmatrix}$$

- How many matches (correspondence pairs) do we need to solve for the transformation parameters?
- Once we have solved for the parameters, how do we compute the coordinates of the corresponding point for (x_{new}, y_{new}) ?

Grauman

Panoramas

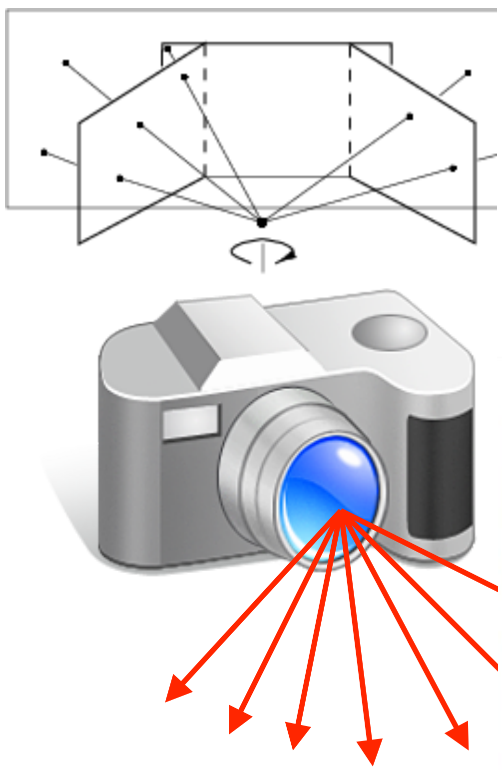


image from S. Seitz

Obtain a wider angle view by combining multiple images.

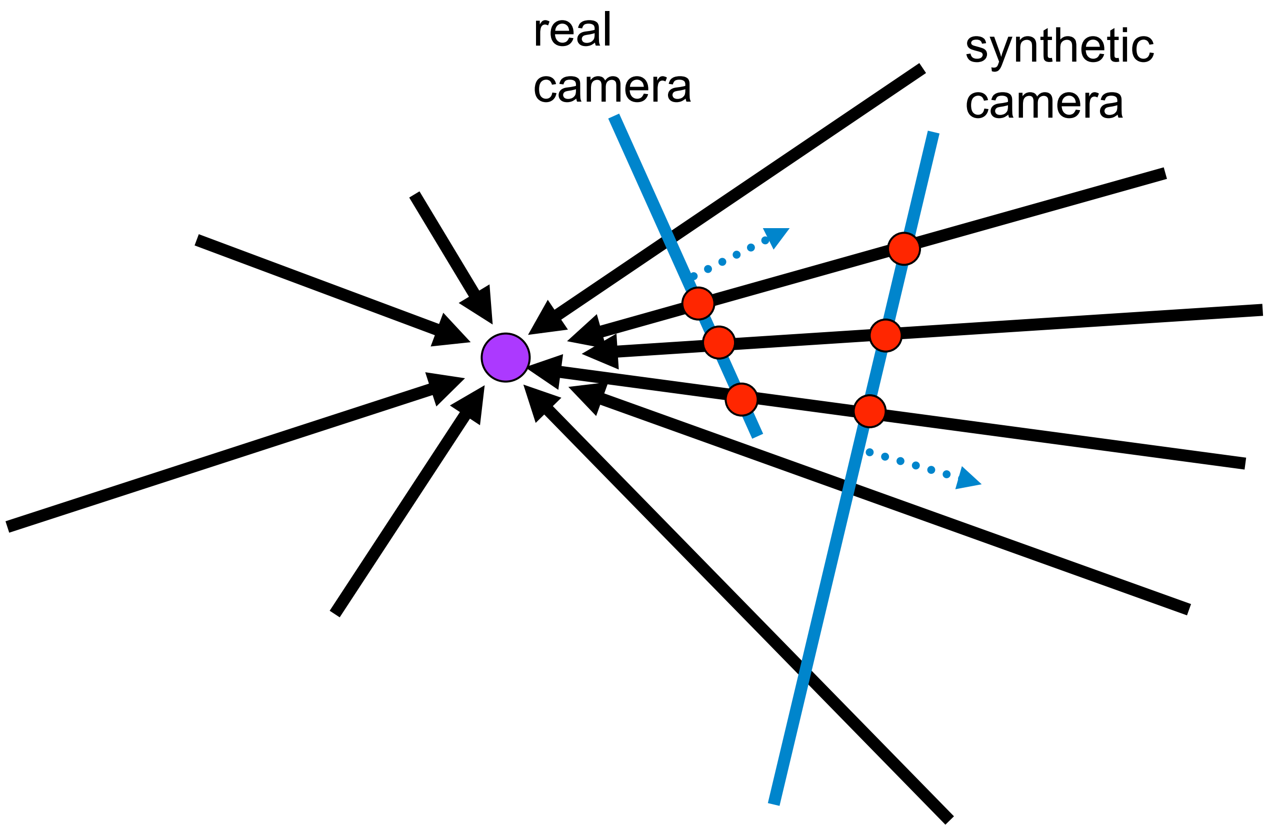
How to stitch together a panorama?

- Basic Procedure
  - Take a sequence of images from the same position
    - Rotate the camera about its optical center
  - Compute transformation between second image and first
  - Transform the second image to overlap with the first
  - Blend the two together to create a mosaic
  - (If there are more images, repeat)

- …but **wait**, why should this work at all?
  - What about the 3D geometry of the scene?
  - Why aren’t we using it?

Source: Steve Seitz

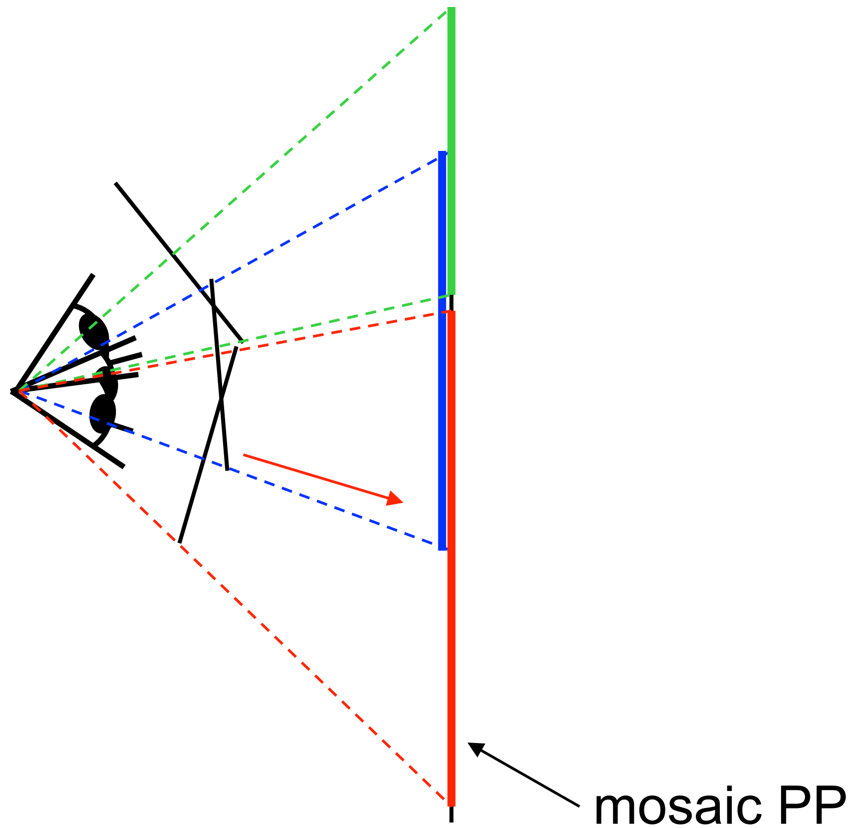
Panoramas: generating synthetic views



Can generate any synthetic camera view
as long as it has **the same center of projection**!

Source: Alyosha Efros

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
- Mosaic is a *synthetic wide-angle camera*

Source: Steve Seitz

Homography

How to relate two images from the same camera center?

– how to map a pixel from PP1 to PP2?

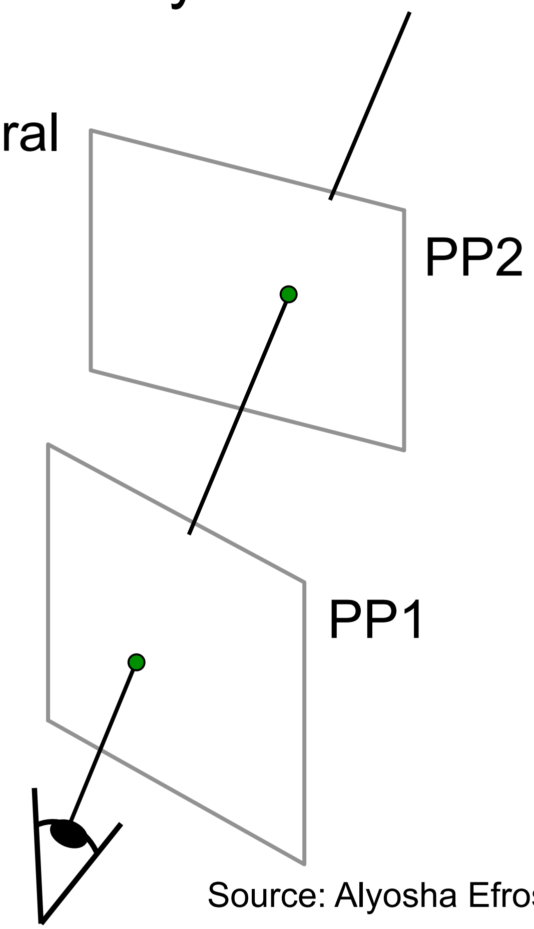
Think of it as a 2D **image warp** from one image to another.

A projective transform is a mapping between any two PPs with the same center of projection

- rectangle should map to arbitrary quadrilateral
- parallel lines aren't
- but must preserve straight lines

called **Homography**

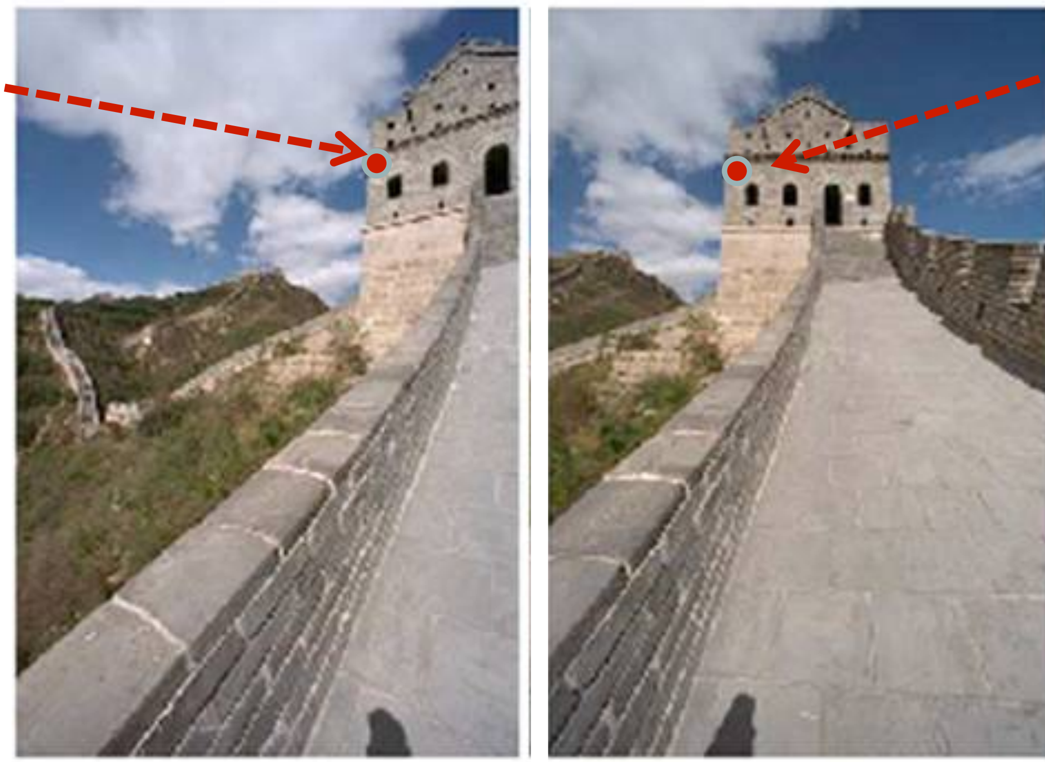
$$\underbrace{\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}}_{\mathbf{p'}} = \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\mathbf{p}}$$



Source: Alyosha Efros

Homography

$$(x, y)$$



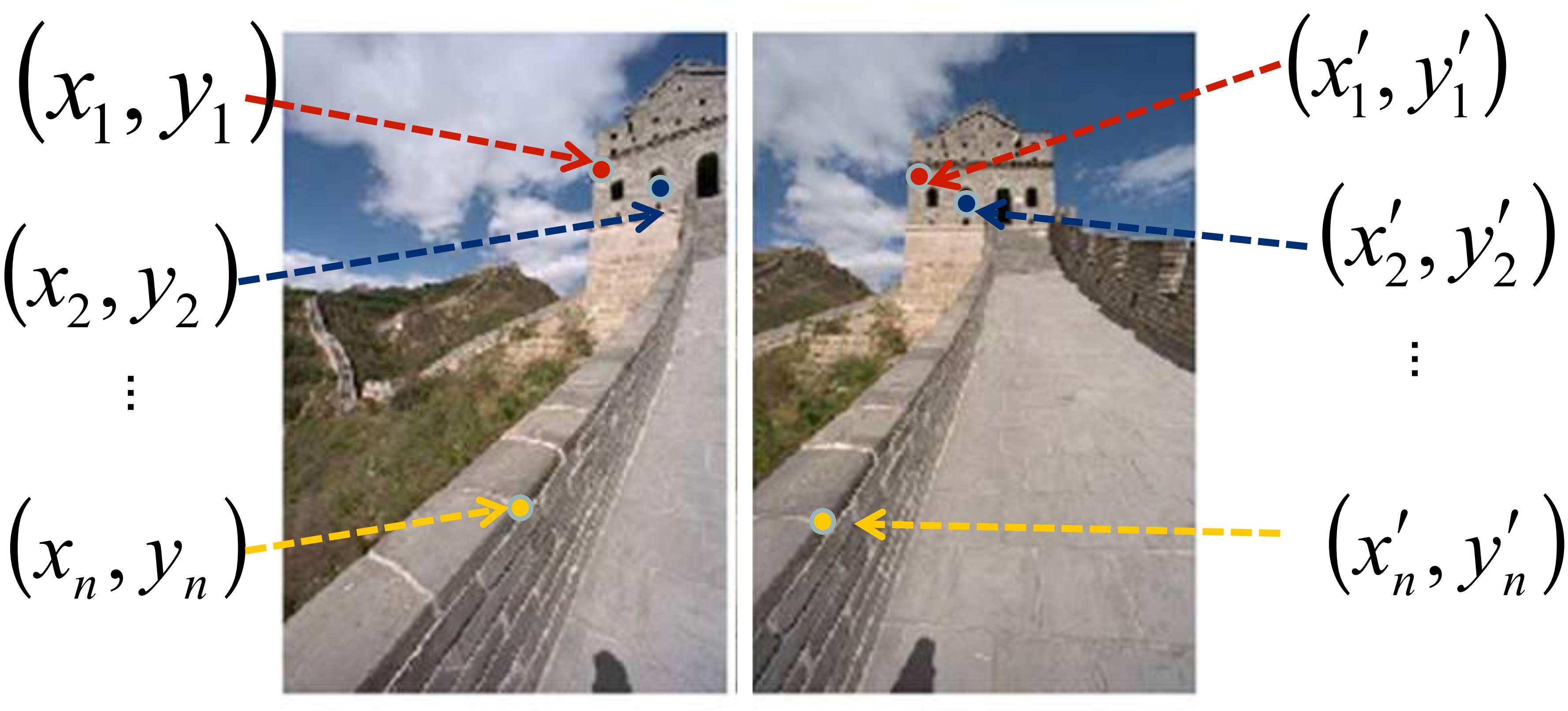
$$\left(\frac{wx'}{w}, \frac{wy'}{w} \right) = (x', y')$$

To **apply** a given homography **H**

- Compute **p'** = **Hp** (regular matrix multiply)
- Convert **p'** from homogeneous to image coordinates

$$\underbrace{\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix}}_{\mathbf{p'}} = \underbrace{\begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}}_{\mathbf{H}} \underbrace{\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}_{\mathbf{p}}$$

Homography



To **compute** the homography given pairs of corresponding points in the images, we need to set up an equation where the parameters of **H** are the unknowns…

Grauman

Solving for homographies

p' = Hp

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

Can set scale factor *i*=1. So, there are 8 unknowns.

Set up a system of linear equations:

Ah = b

where vector of unknowns h = $[a,b,c,d,e,f,g,h]^T$

Need at least 8 eqs, but the more the better…

Solve for h. If overconstrained, solve using least-squares:

$$\min \| Ah - b \|^2$$

>> **`help mldivide`**

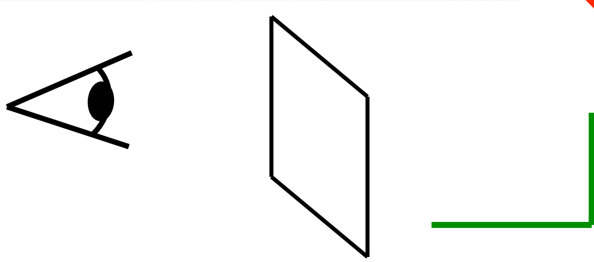
Grauman

Recap: How to stitch together a panorama?

- Basic Procedure
  - Take a sequence of images from the same position
    - Rotate the camera about its optical center
  - Compute transformation between second image and first
  - Transform the second image to overlap with the first
  - Blend the two together to create a mosaic
  - (If there are more images, repeat)

Source: Steve Seitz

Image warping with homographies



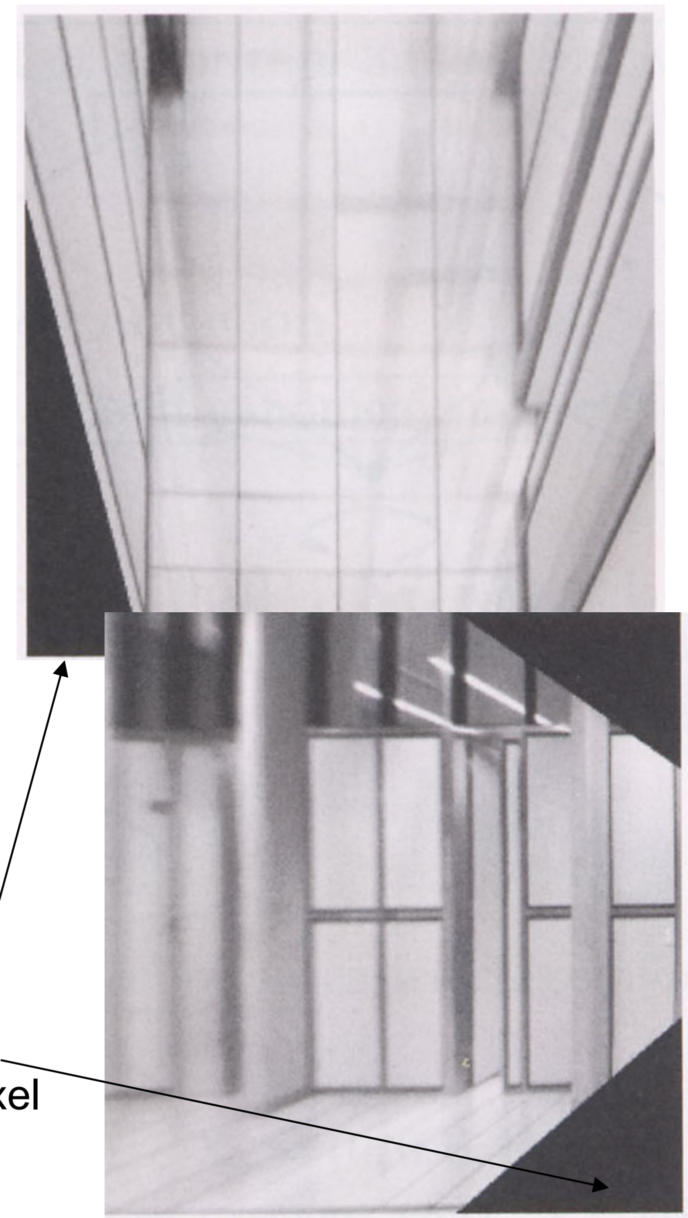
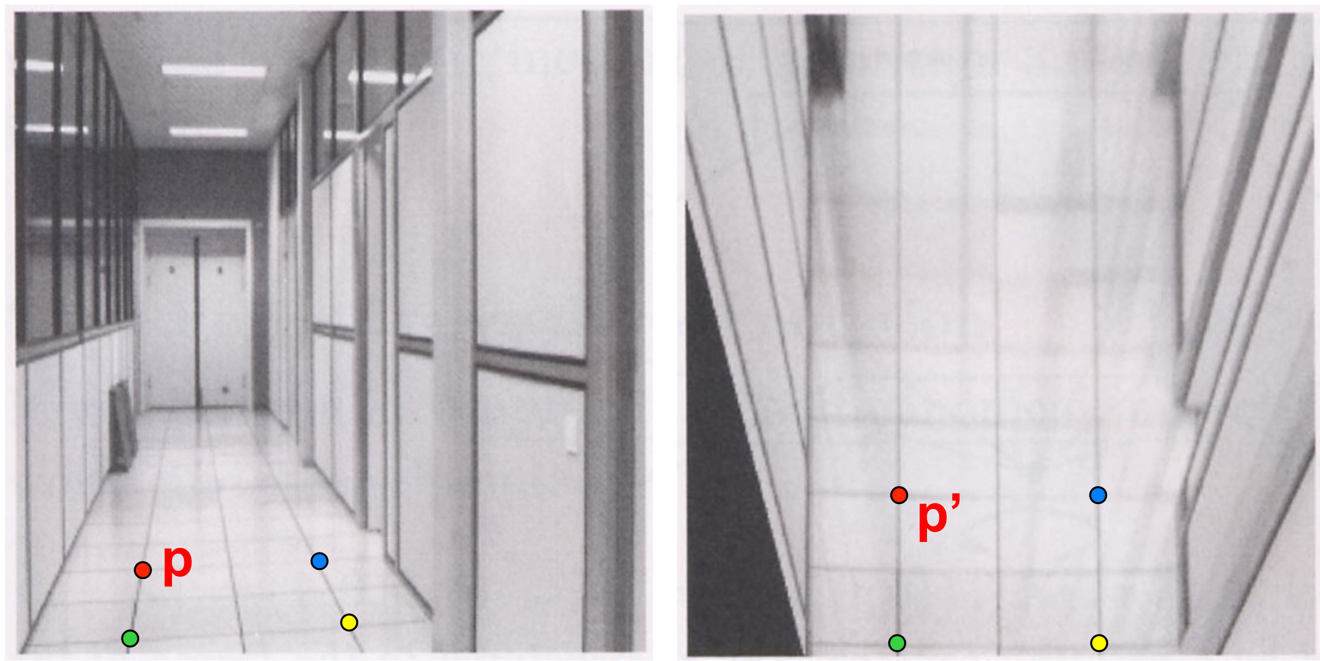


image plane in front

black area where no pixel maps to

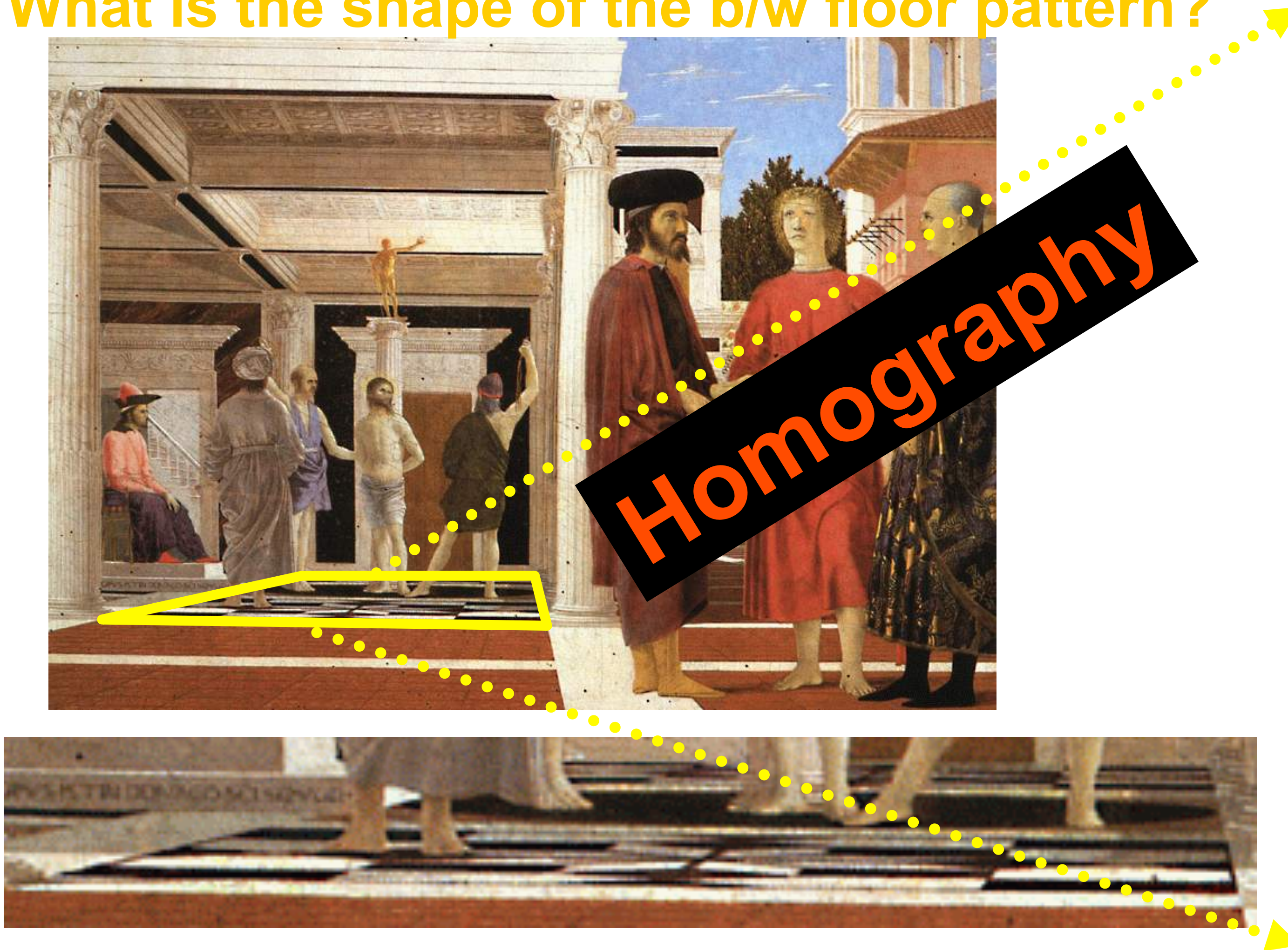
Source: Steve Seitz

Image rectification

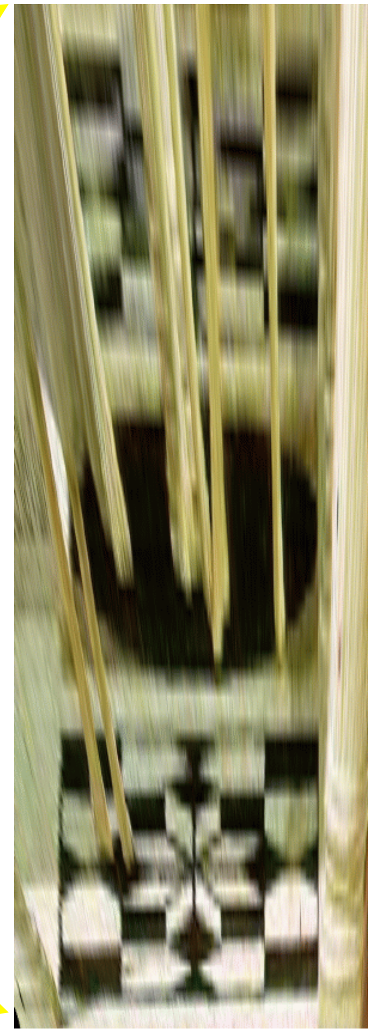


Analysing patterns and shapes

What is the shape of the b/w floor pattern?



The floor (enlarged)

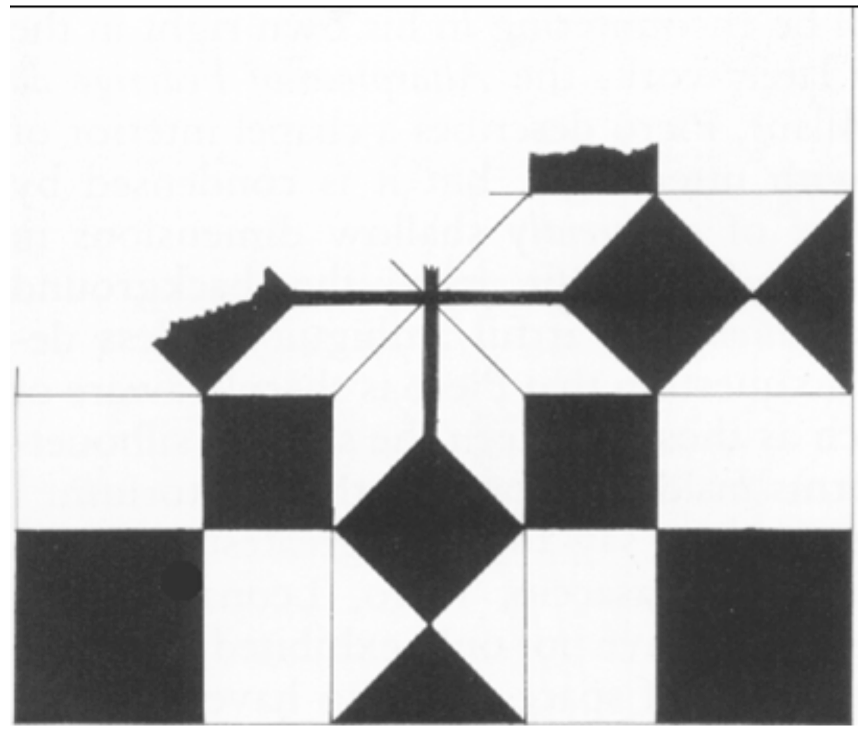


Automatically rectified floor

Slide from Criminisi

Analysing patterns and shapes

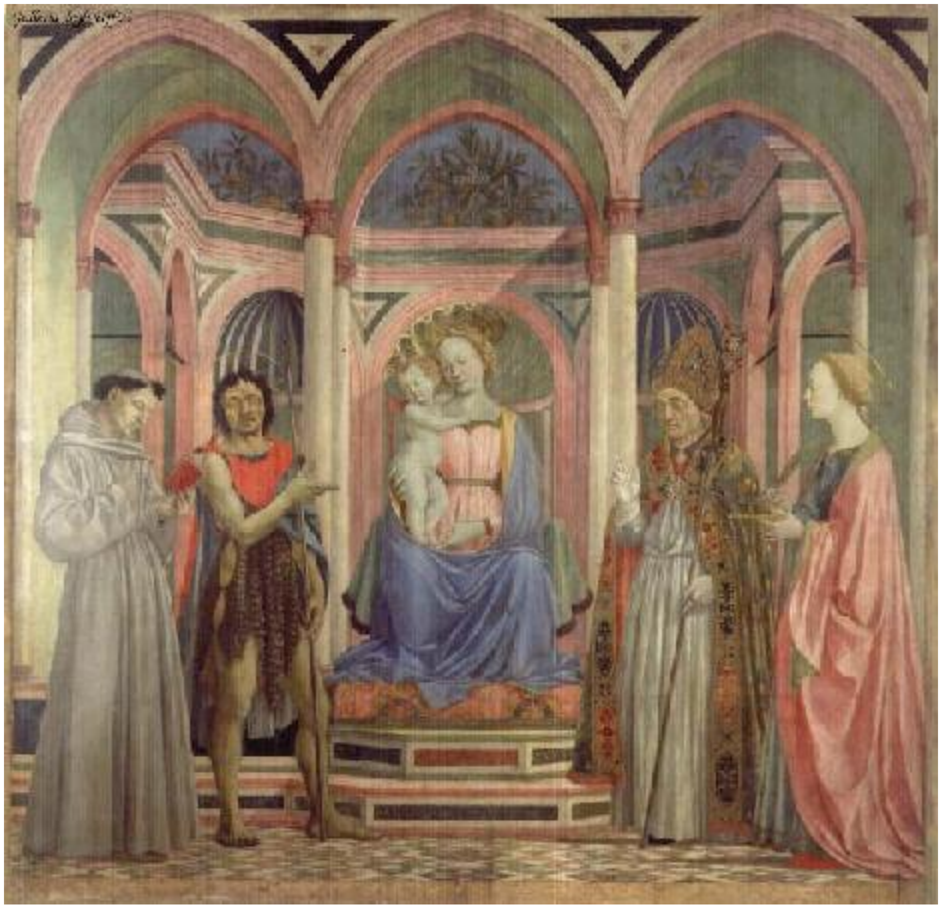
Automatic rectification



From Martin Kemp *The Science of Art*
(manual reconstruction)

Slide from Criminisi

Analysing patterns and shapes



***St. Lucy Altarpiece,* D. Veneziano**

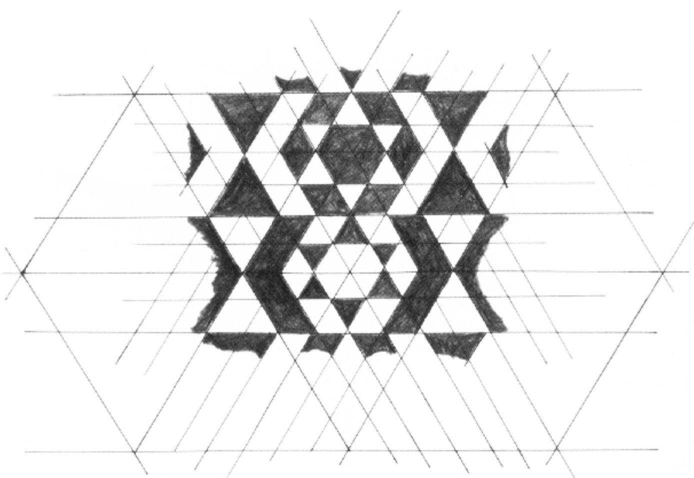
Slide from Criminisi

What is the (complicated) shape of the floor pattern?

Automatically rectified floor

Analysing patterns and shapes

Automatic rectification

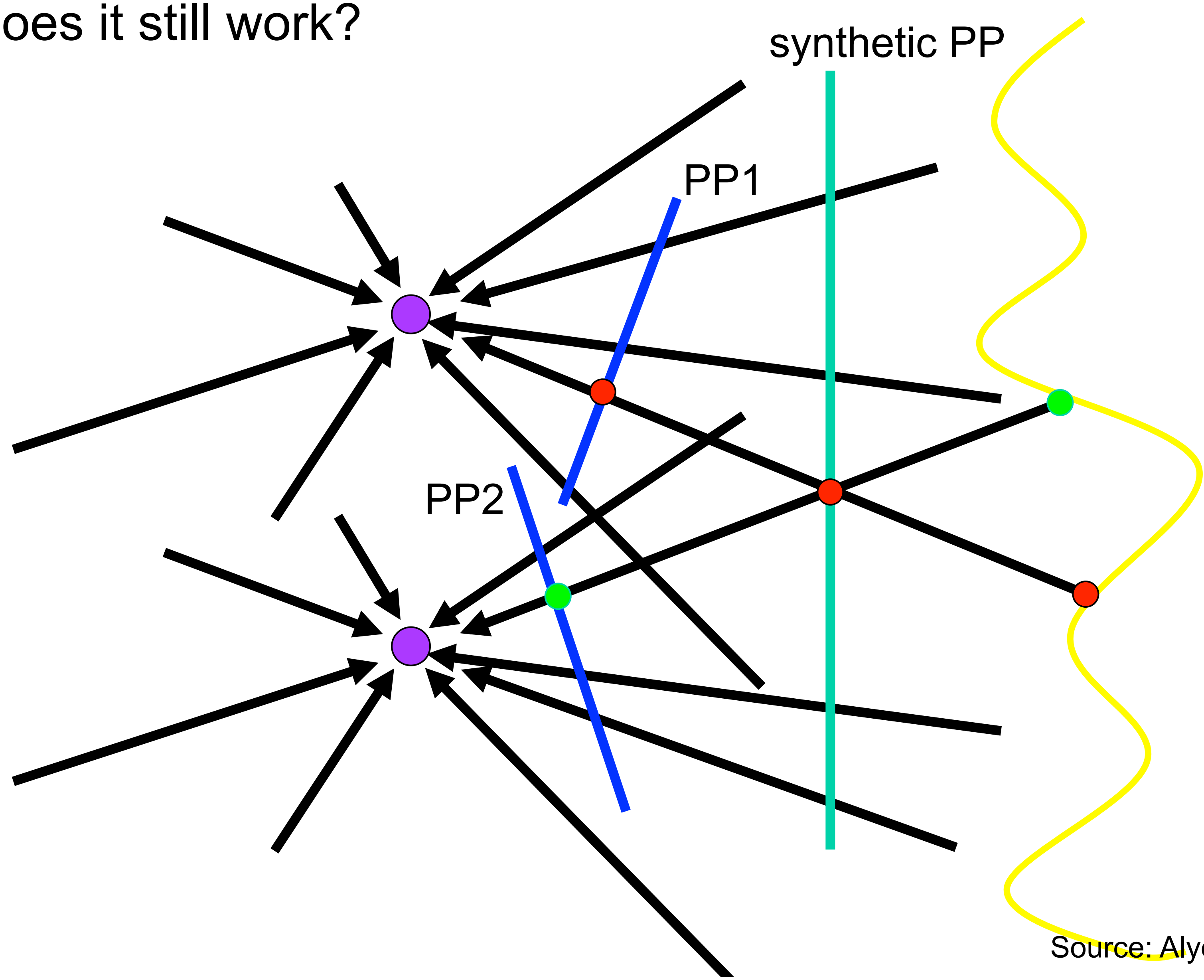


From Martin Kemp, *The Science of Art* *(manual reconstruction)*

Slide from Criminisi

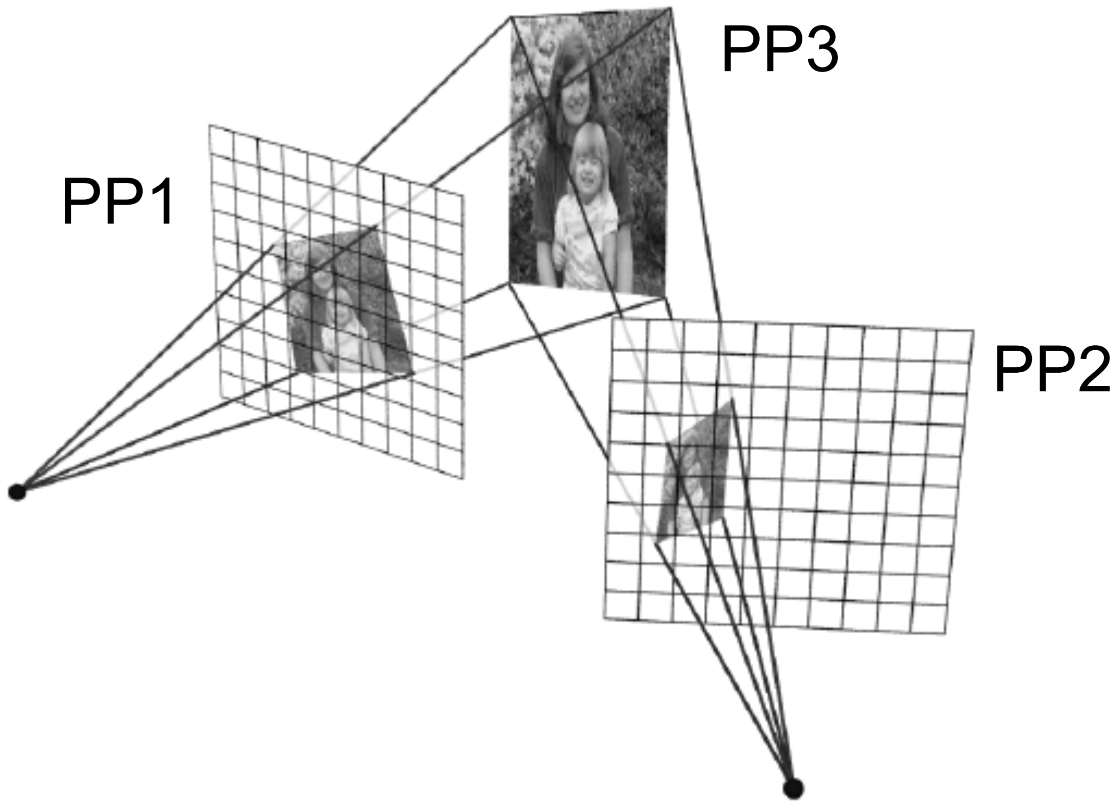
changing camera center

Does it still work?



Source: Alyosha Efros

Planar scene (or far away)



PP3 is a projection plane of both centers of projection, so we are OK!

This is how big aerial photographs are made

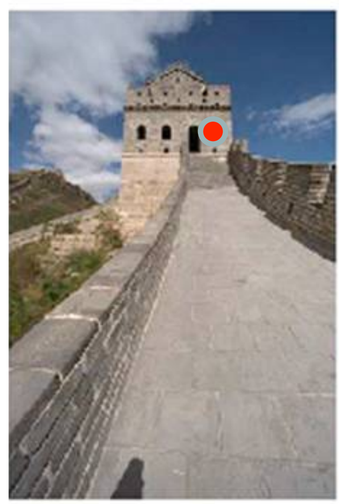
Source: Alyosha Efros

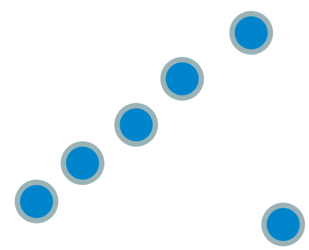
Grauman


Map
Satellite
Hybrid
200 ft
100 m
©2007 Google - Terms of Use

Outliers

- **Outliers** can hurt the quality of our parameter estimates, e.g.,
  - an erroneous pair of matching points from two images
  - an edge point that is noise, or doesn't belong to the line we are fitting.

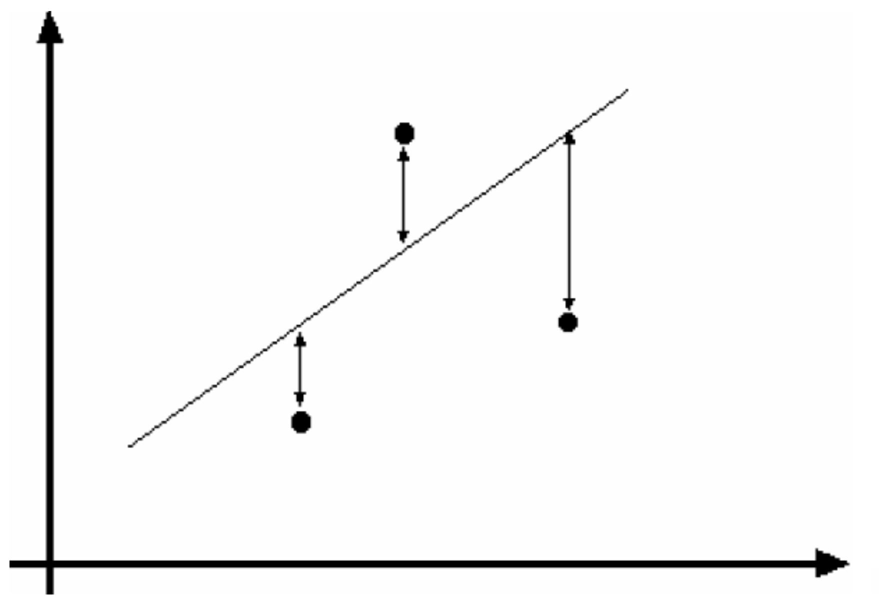




Grauman

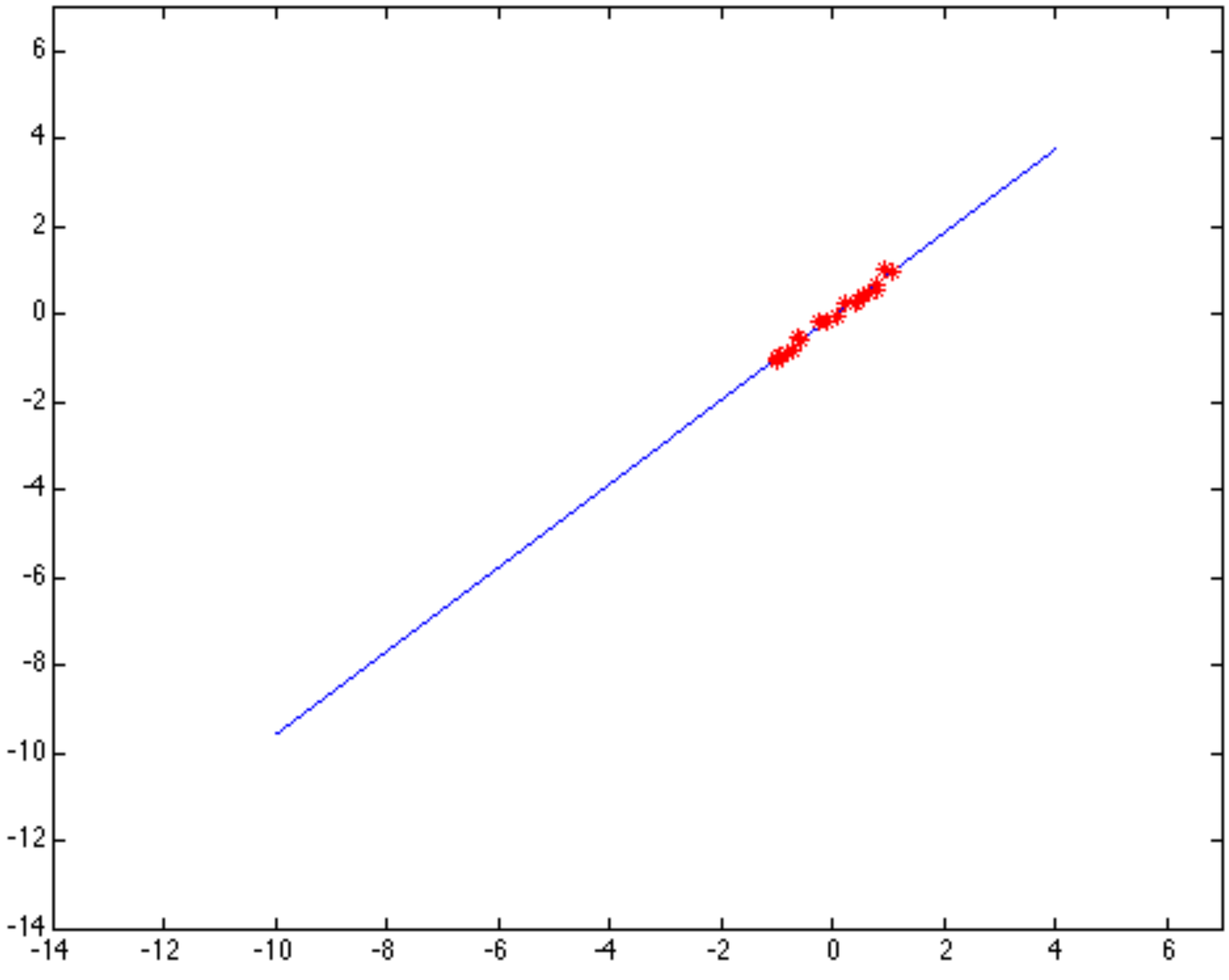
Example: least squares line fitting

- Assuming all the points that belong to a particular line are known

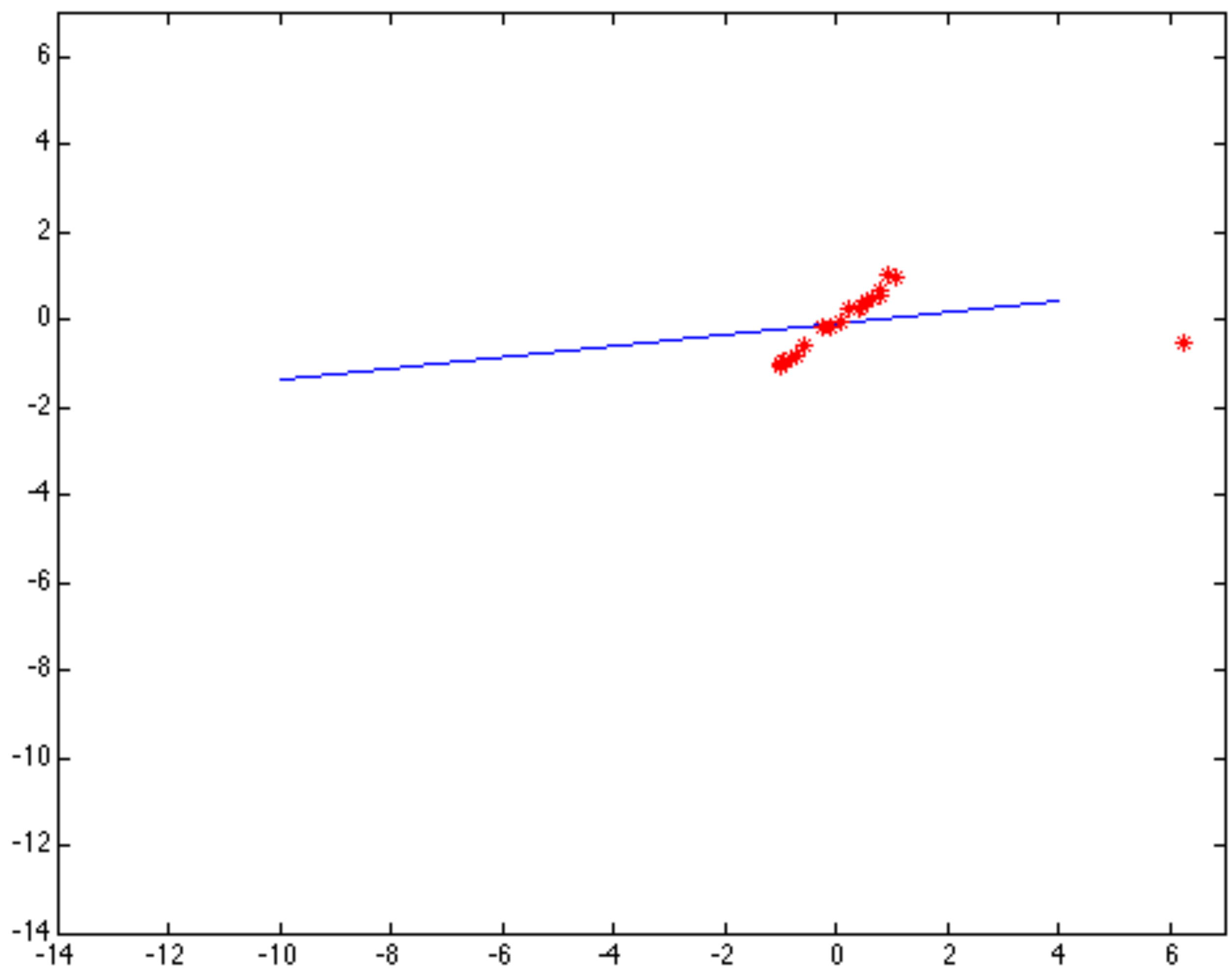


Grauman

Outliers affect least squares fit



Outliers affect least squares fit



RANSAC

- RANdom Sample Consensus

- Approach: we want to avoid the impact of outliers, so let’s look for “inliers”, and use those only.

- Intuition: if an outlier is chosen to compute the current fit, then the resulting line won’t have much support from rest of the points.

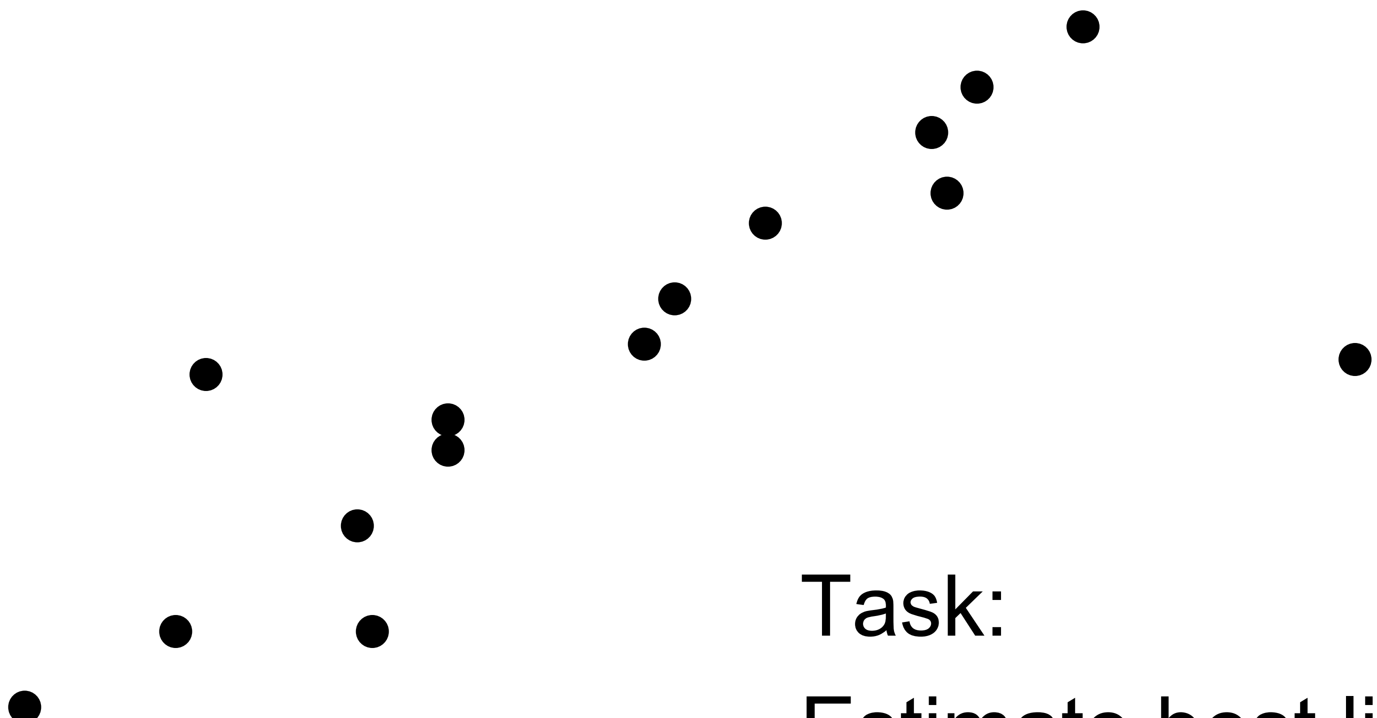
RANSAC

- <u>RANSAC loop</u>:

1. Randomly select a *seed group* of points on which to base transformation estimate (e.g., a group of matches)
2. Compute transformation from seed group
3. Find *inliers* to this transformation
4. If the number of inliers is sufficiently large, re-compute least-squares estimate of transformation on all of the inliers

- Keep the transformation with the largest number of inliers

RANSAC Line Fitting Example

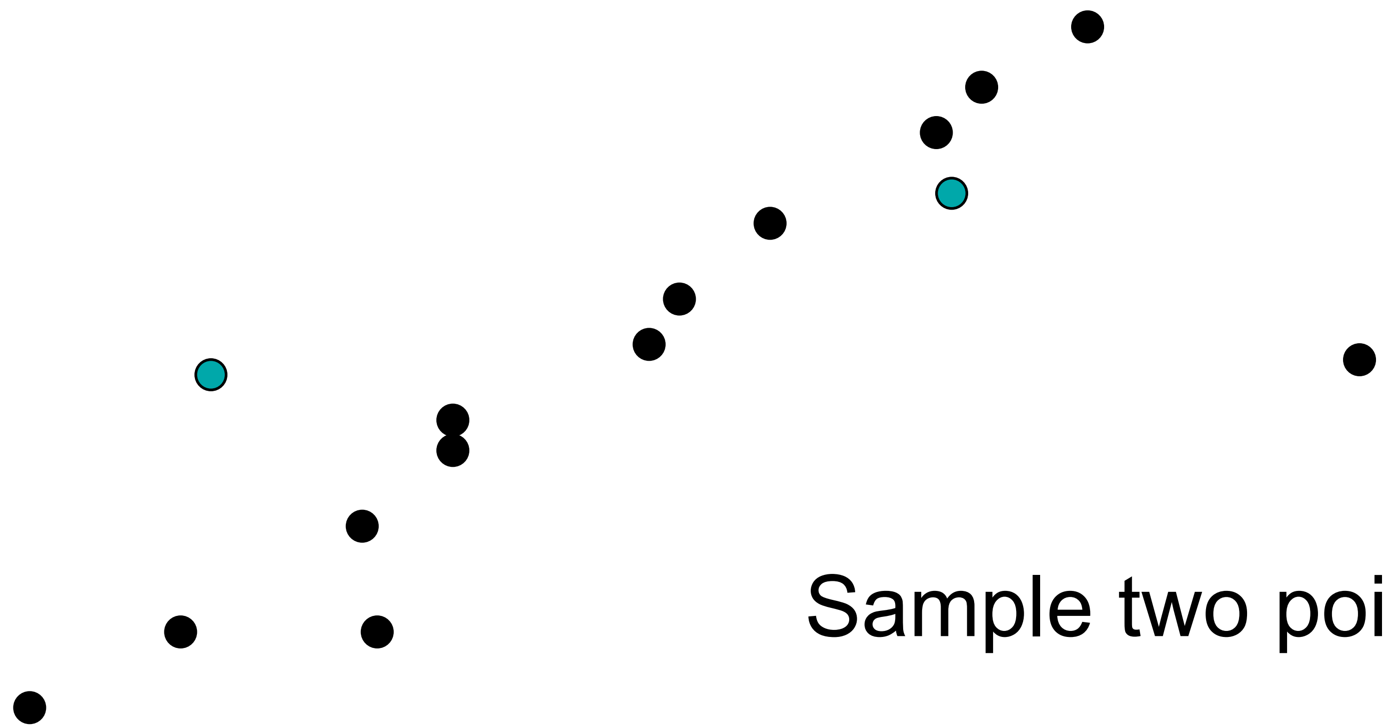


Task:

Estimate best line

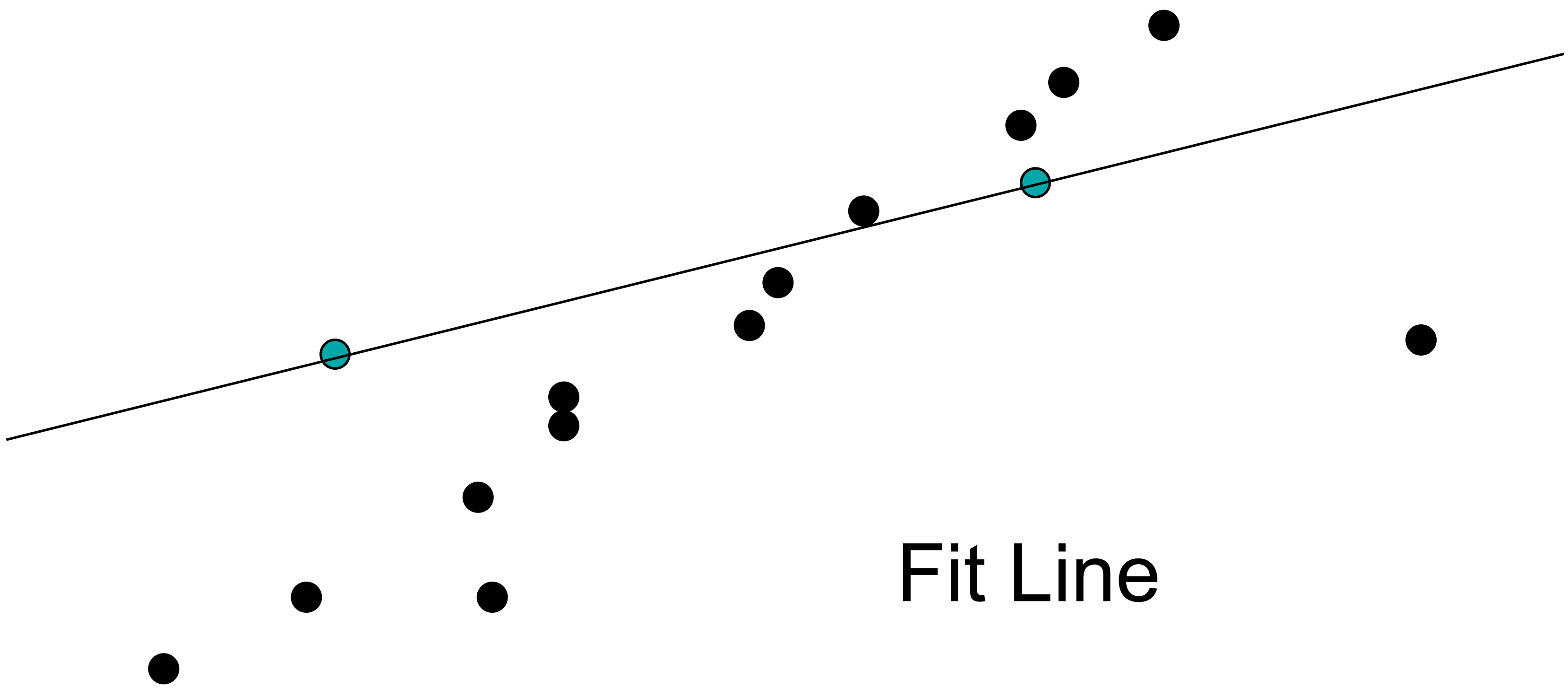
Slide credit: Jinxiang Chai, CMU

RANSAC Line Fitting Example

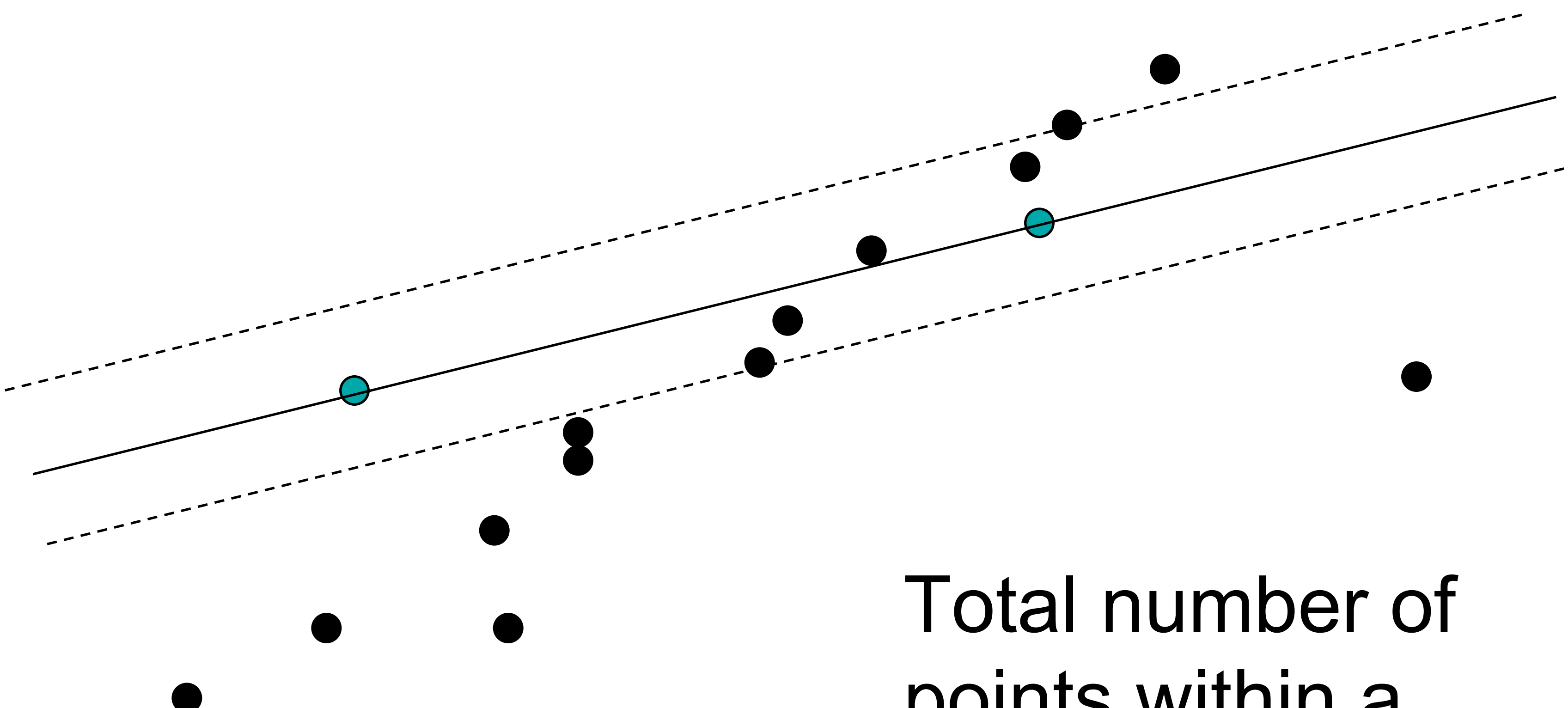


Sample two points

RANSAC Line Fitting Example



RANSAC Line Fitting Example

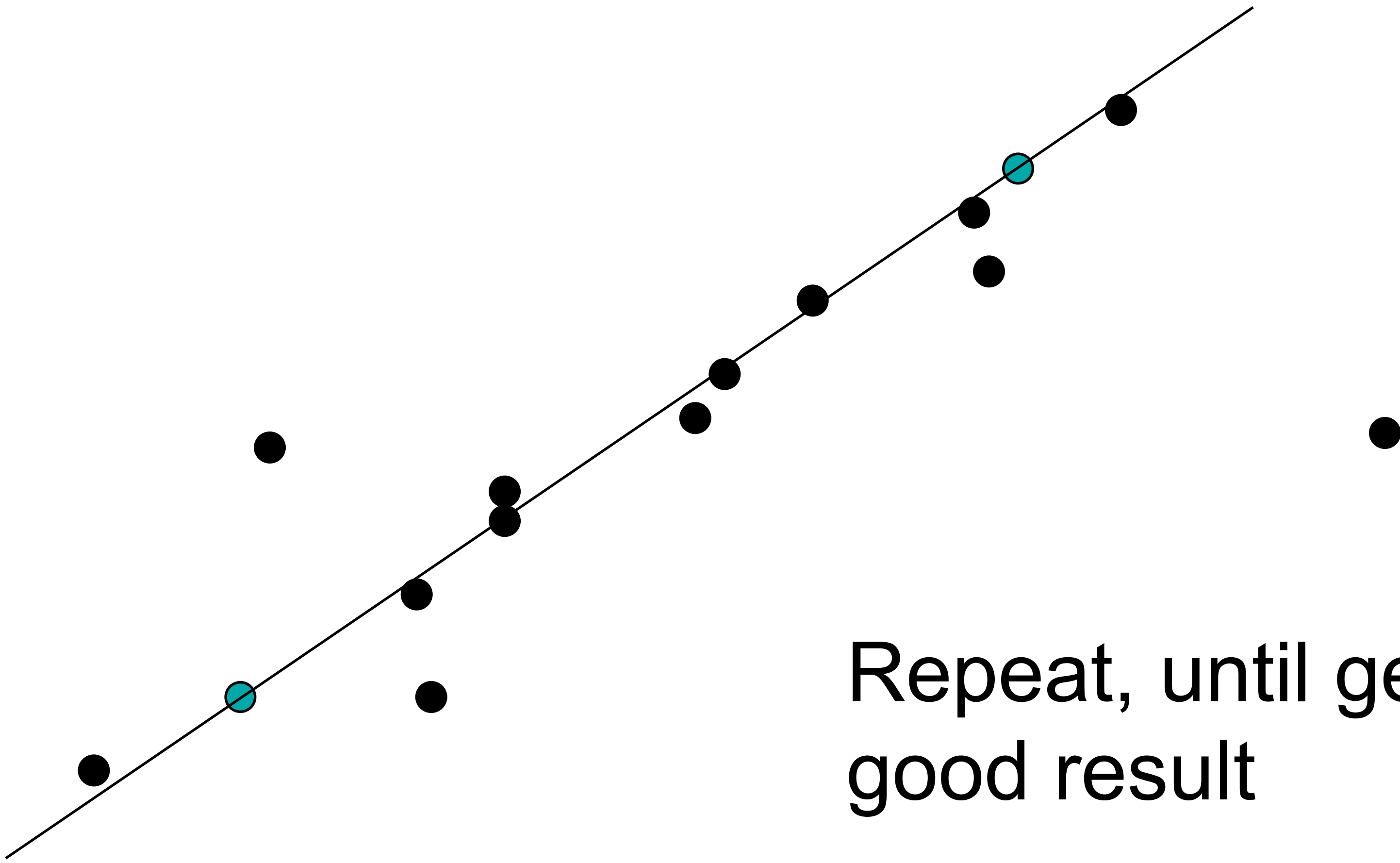


Total number of points within a threshold of line.

RANSAC Line Fitting Example

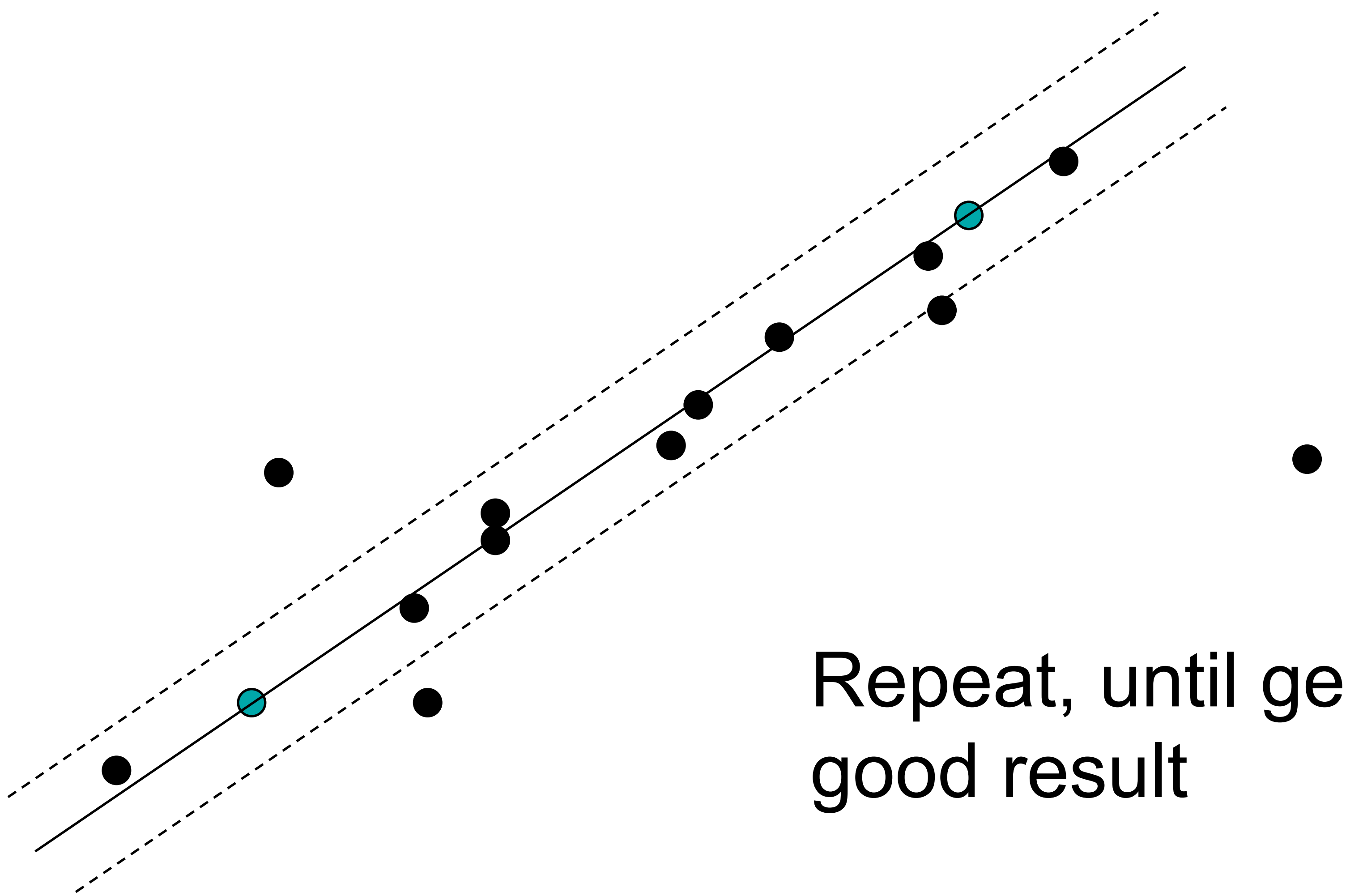
Repeat, until get a good result

RANSAC Line Fitting Example



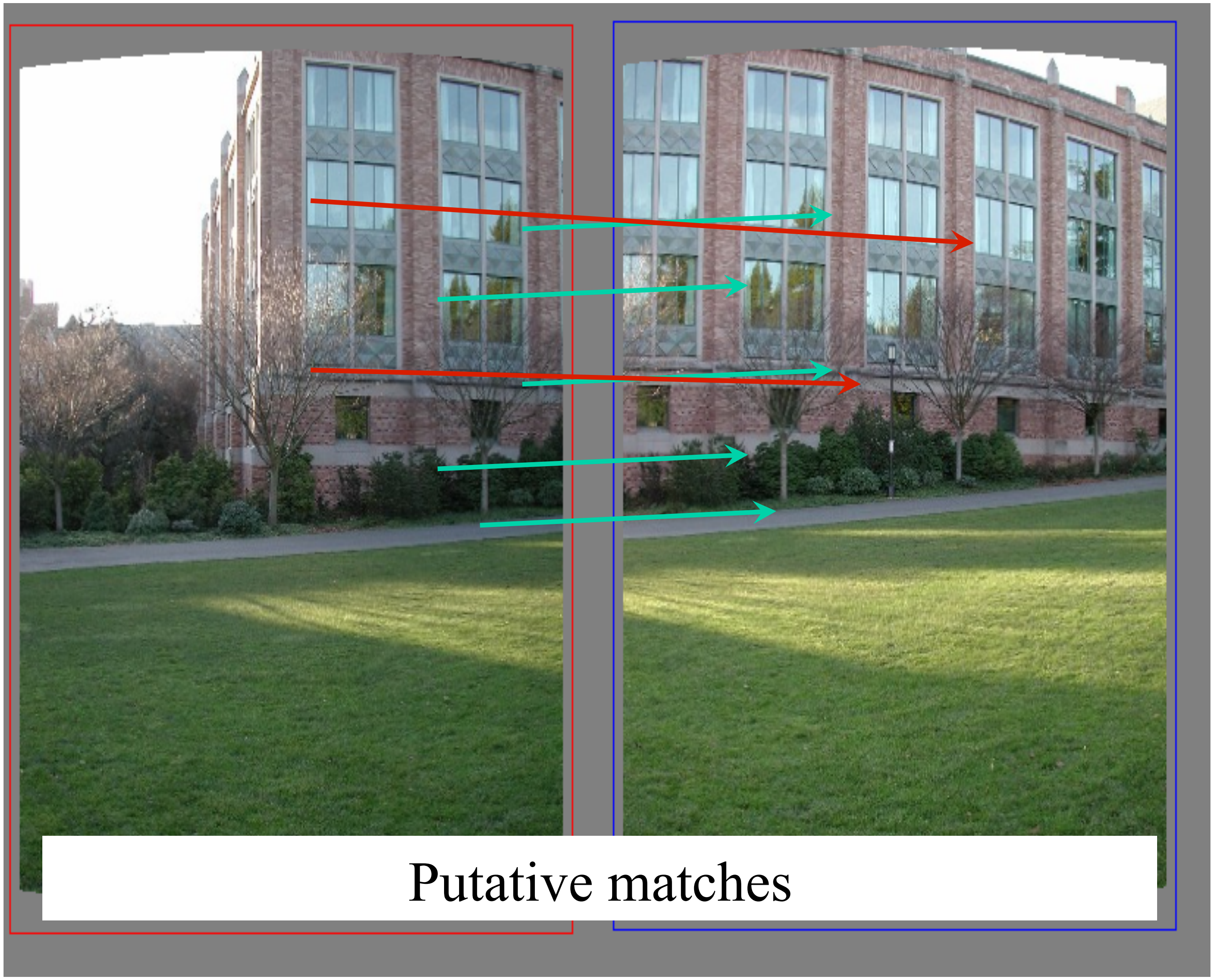
Repeat, until get a good result

RANSAC Line Fitting Example



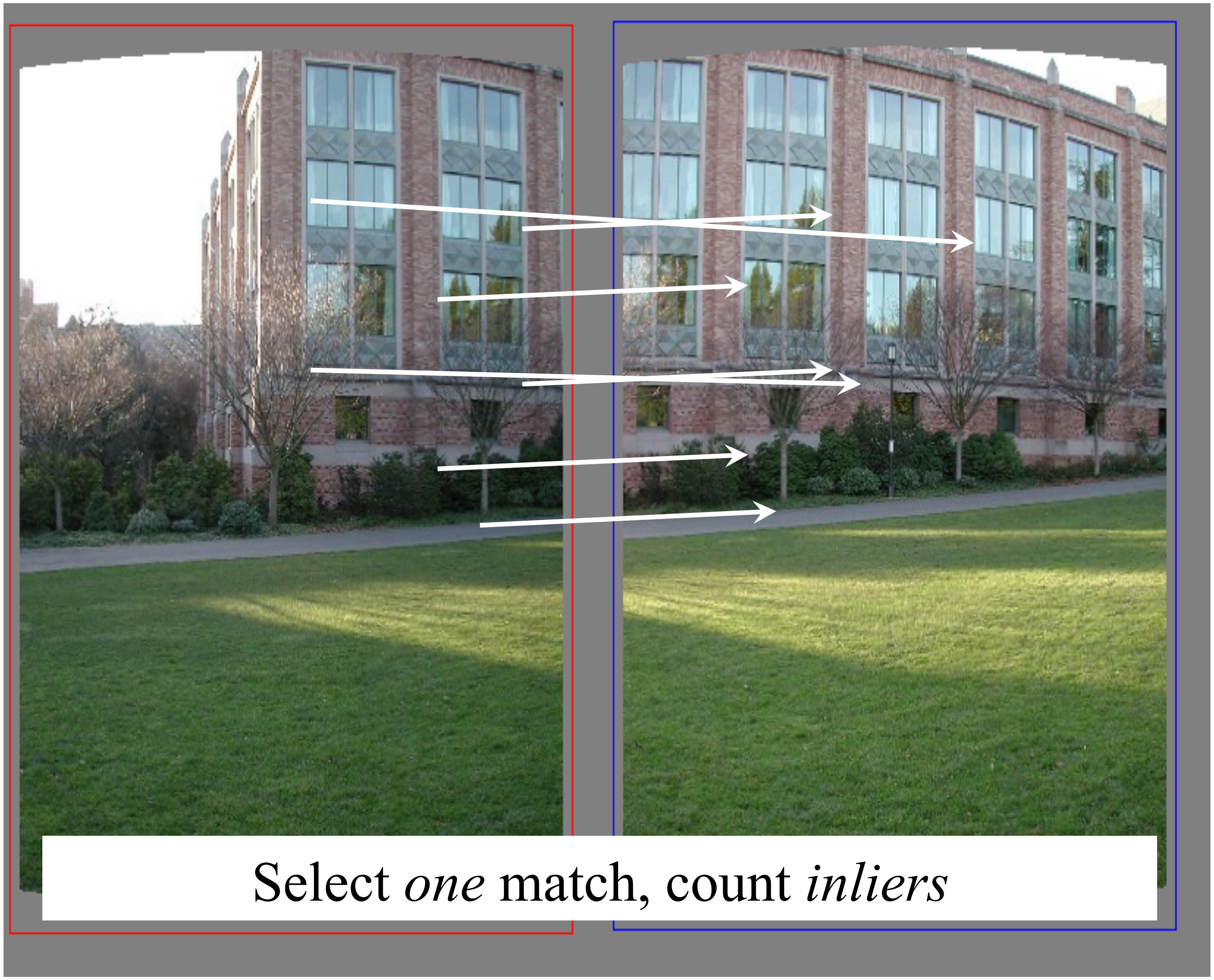
Repeat, until get a good result

RANSAC example: Translation

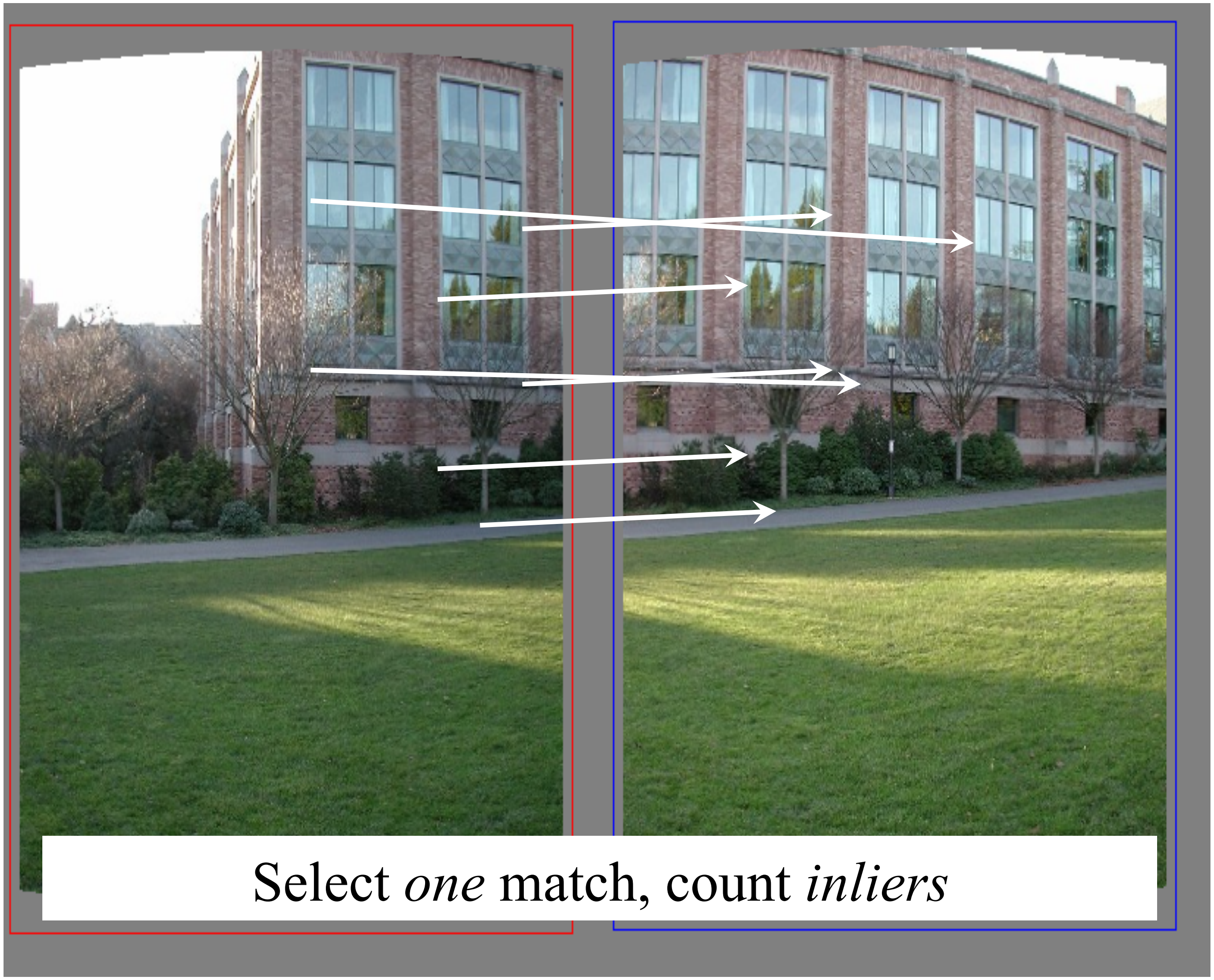


Source: Rick Szeliski

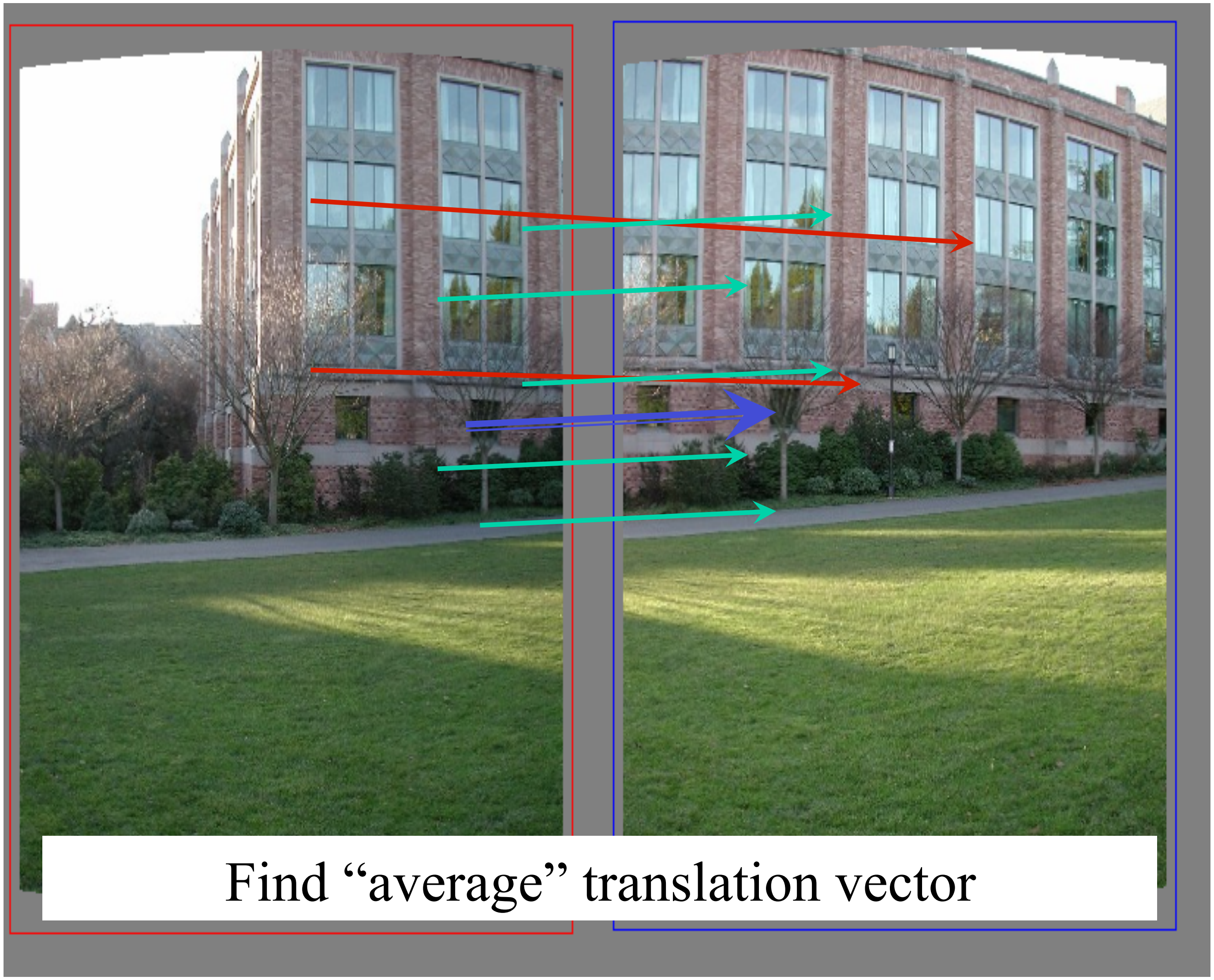
RANSAC example: Translation



RANSAC example: Translation



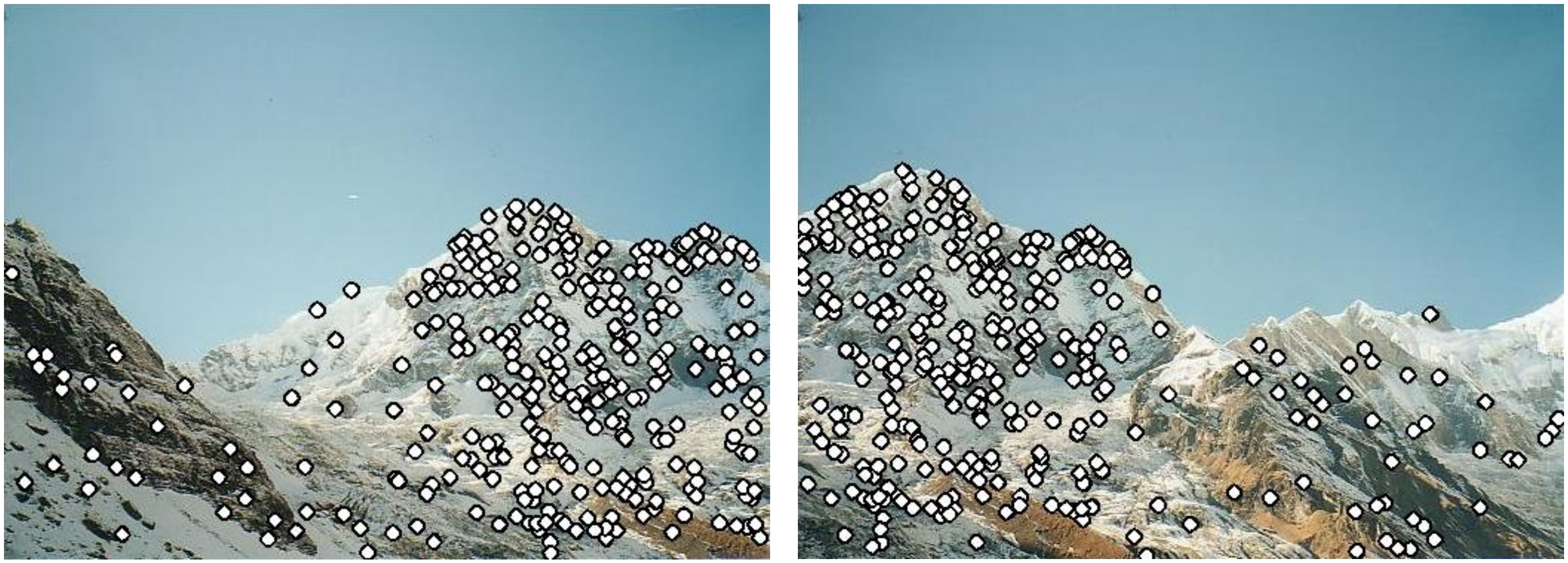
RANSAC example: Translation



Feature-based alignment outline

Source: L. Lazebnik

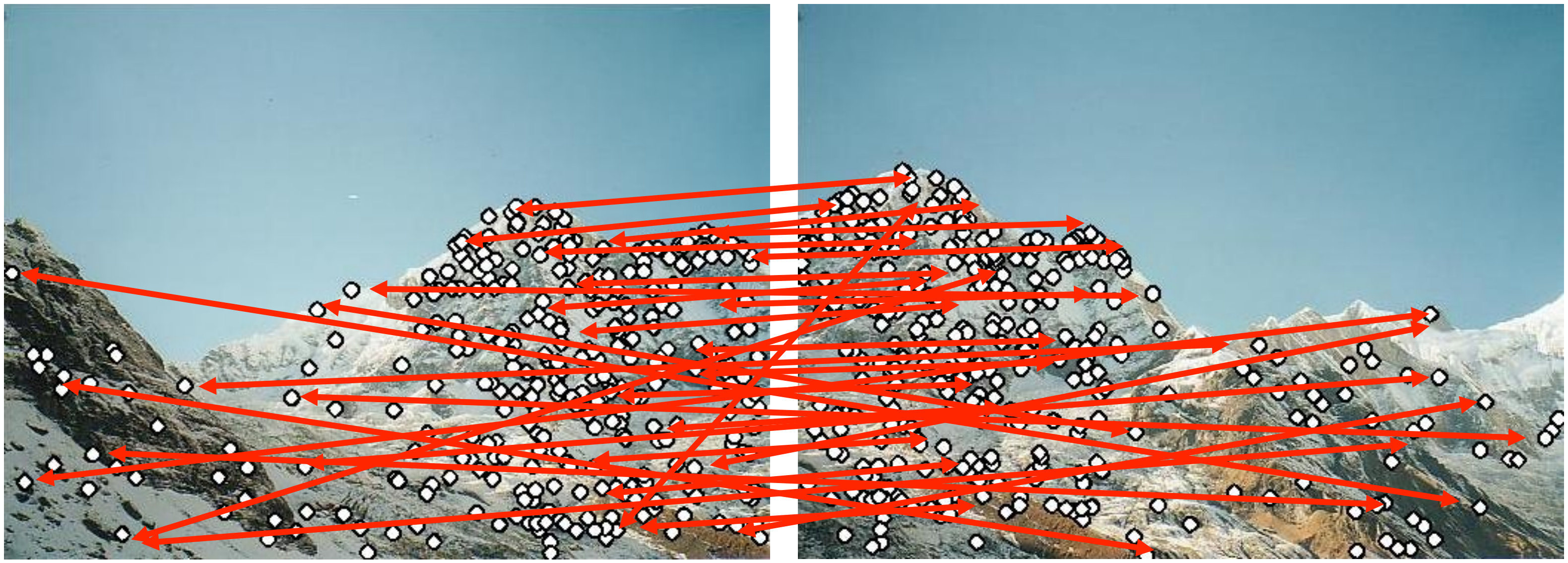
Feature-based alignment outline



- Extract features

Source: L. Lazebnik

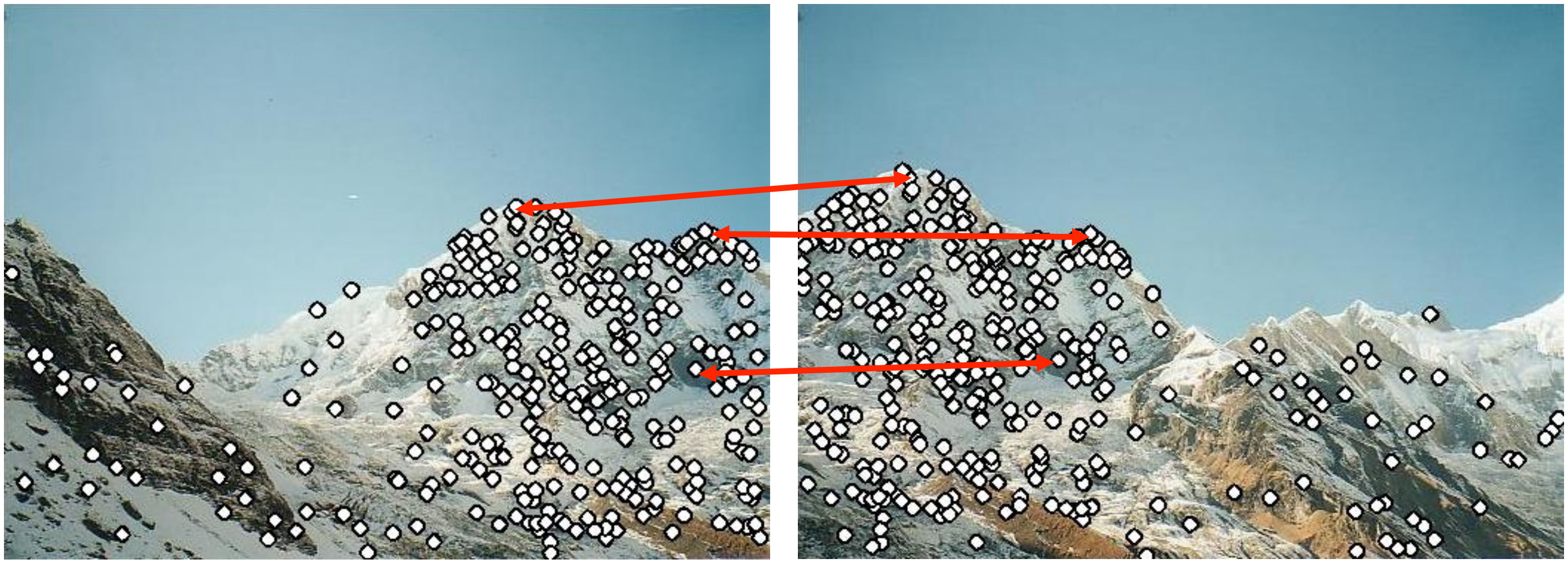
Feature-based alignment outline



- Extract features
- Compute *putative matches*

Source: L. Lazebnik

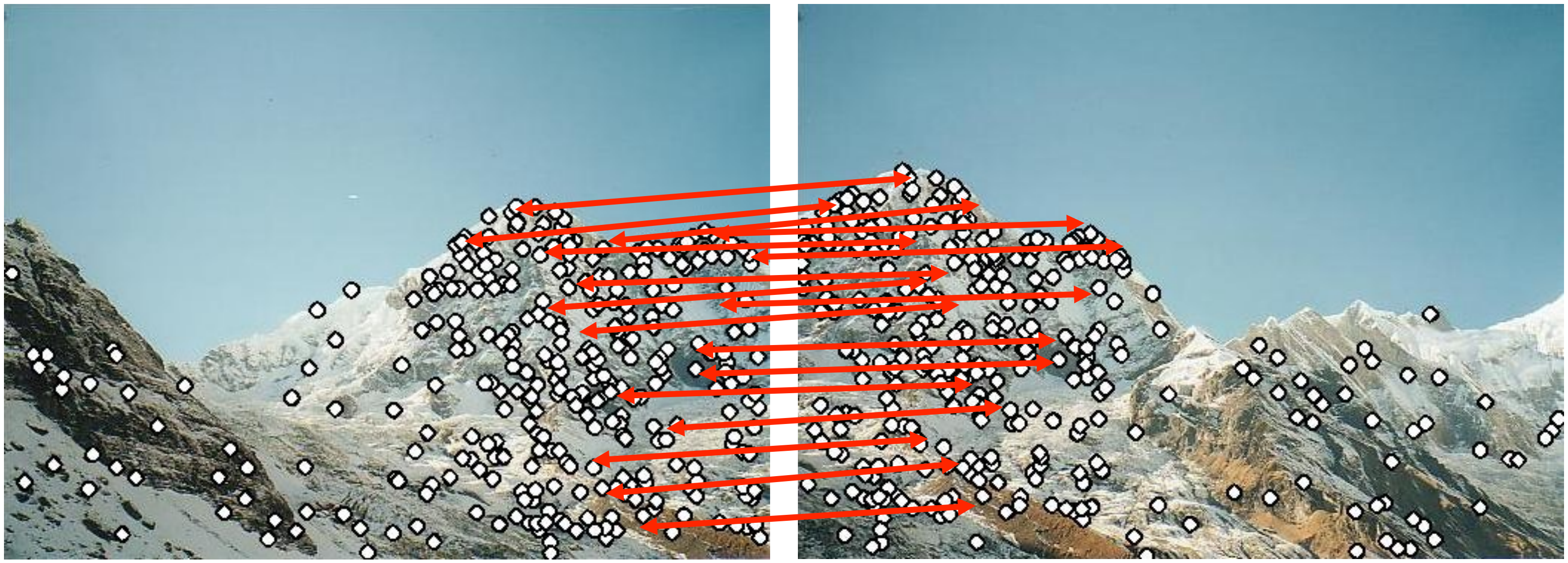
Feature-based alignment outline



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)

Source: L. Lazebnik

Feature-based alignment outline



- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)
  - *Verify* transformation (search for other matches consistent with *T*)

Source: L. Lazebnik

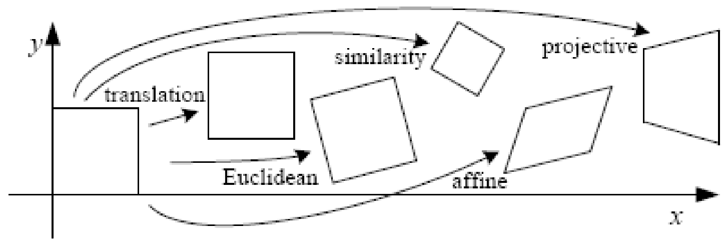
Feature-based alignment outline

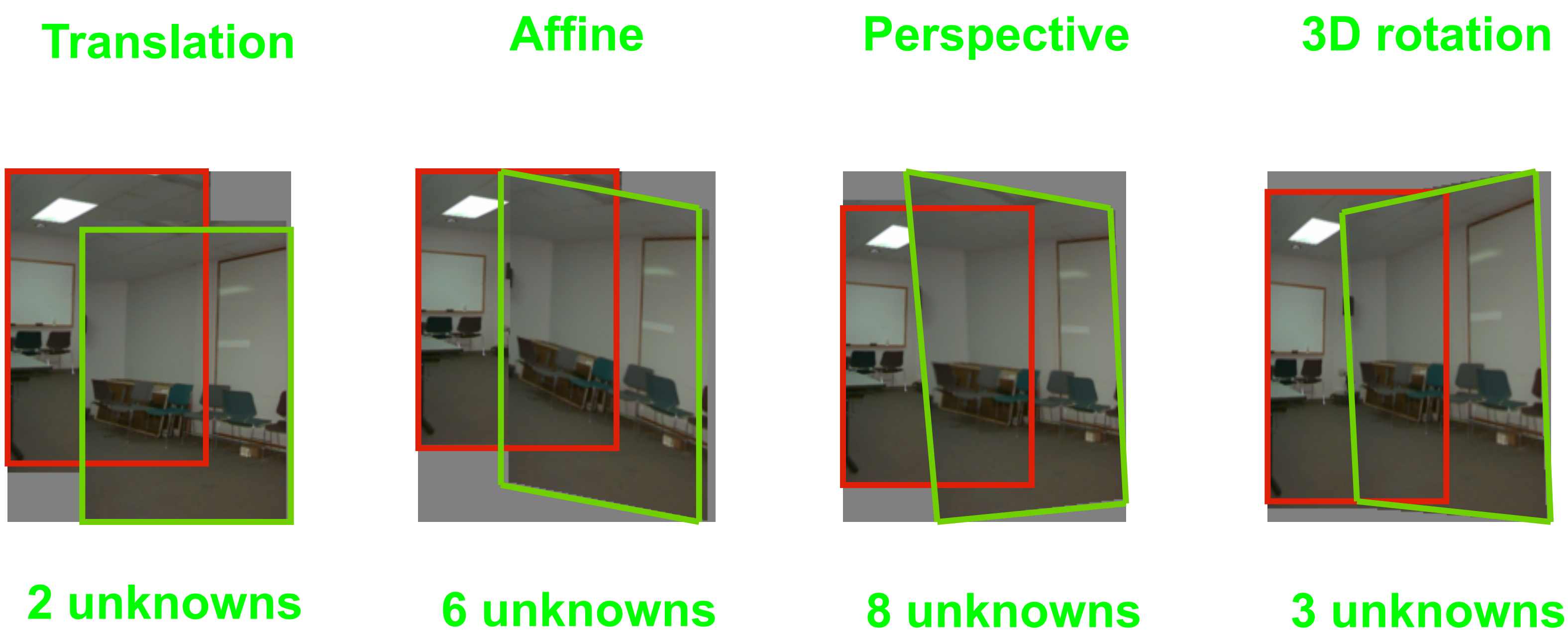
- Extract features
- Compute *putative matches*
- Loop:
  - *Hypothesize* transformation *T* (small group of putative matches that are related by *T*)
  - *Verify* transformation (search for other matches consistent with *T*)

Source: L. Lazebnik

Towards large-scale mosaics…

Motion models

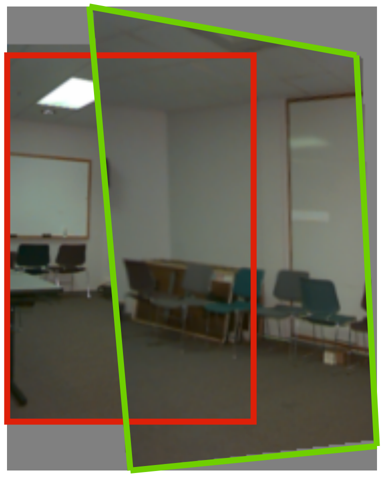




Szeliski

Plane perspective mosaics

- 8-parameter homographies
- Limitations:
  - local minima
  - slow convergence
  - difficult to control interactively



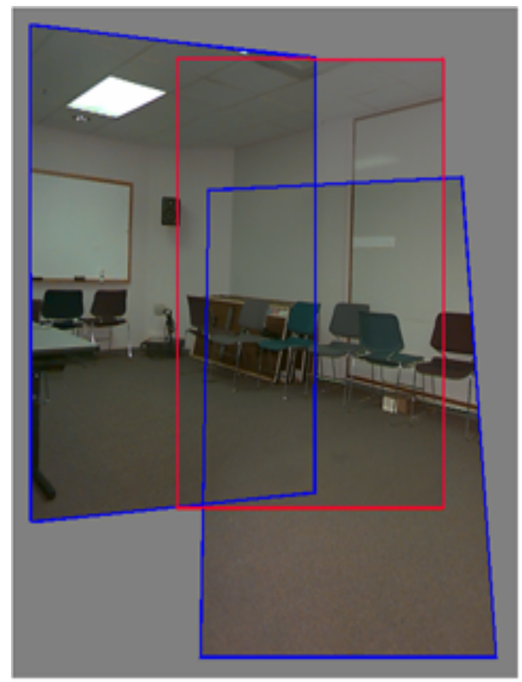
Szeliski

Rotational mosaics

– Directly optimize rotation and focal length

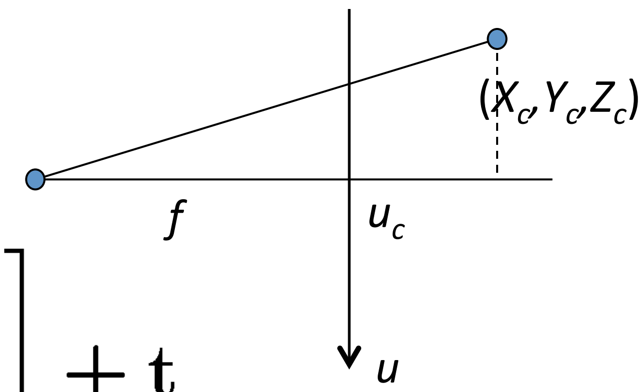
– Advantages:

- ability to build full-view panoramas
- easier to control interactively
- more stable and accurate estimates



Szeliski

3D → 2D Perspective Projection



$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} \mathbf{R} \end{bmatrix}_{3\times 3} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \mathbf{t}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} f & 0 & u_c \\ 0 & f & v_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

Szeliski

Rotational mosaic

- Projection equations

1. Project from image to 3D ray

- $(x_0,y_0,z_0) \quad = (u_0-u_c,v_0-v_c,f)$

2. Rotate the ray by camera motion

- $(x_1,y_1,z_1) \quad = \boldsymbol{R}_{01}\ (x_0,y_0,z_0)$

3. Project back into new (source) image

- $(u_1,v_1) \quad = (fx_1/z_1+u_c,fy_1/z_1+v_c)$

Szeliski

Establishing correspondences

1. 'Direct' method: *(more next week)*
   - Use generalization of affine motion model [Szeliski & Shum '97]
2. Feature-based method
   - Extract features, match, find consistent *inliers* [Lowe ICCV'99; Schmid ICCV'98, Brown&Lowe ICCV'2003]
   - Compute $\boldsymbol{R}$ from correspondences (absolute orientation)

Szeliski

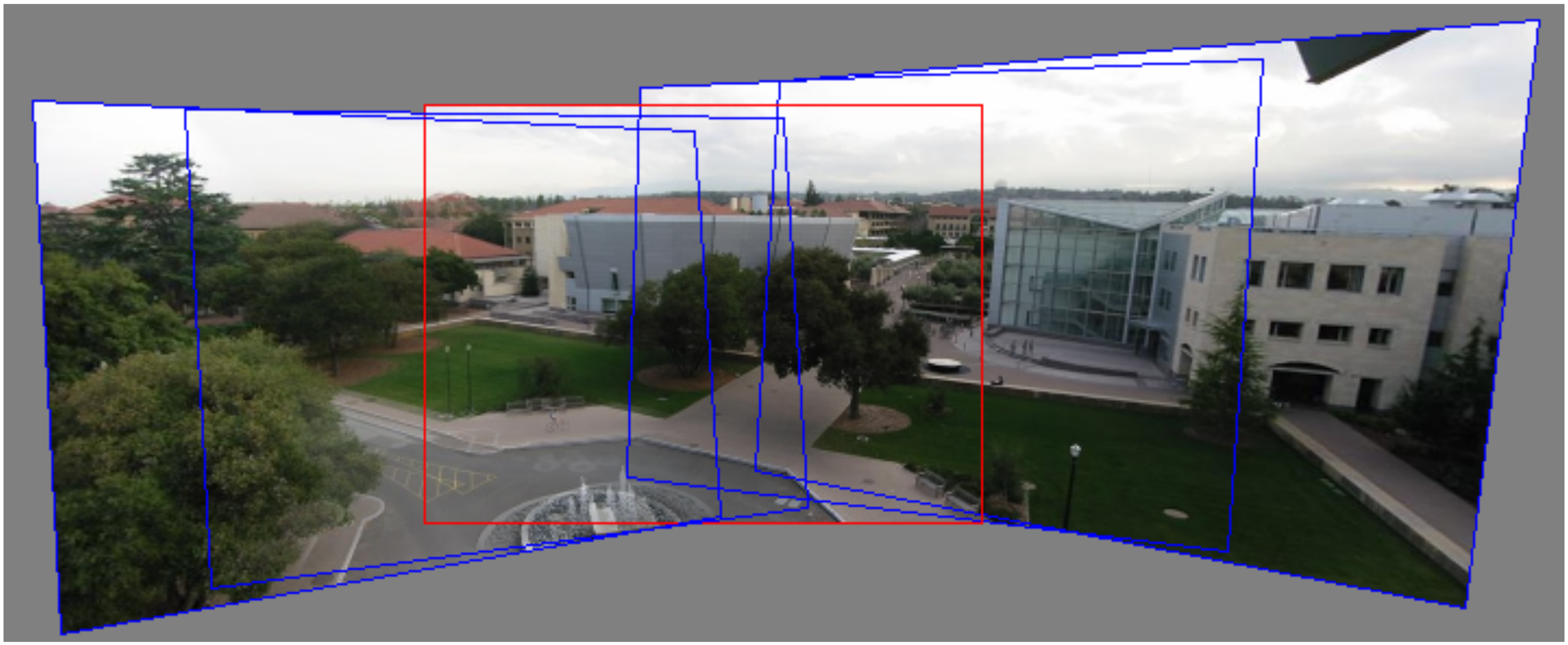
Absolute orientation

[Arun *et al.*, PAMI 1987] [Horn *et al.*, JOSA A 1988]
Procrustes Algorithm [Golub & VanLoan]

Given two sets of matching points, compute R

- $p_i' = \boldsymbol{R}\, p_i$ 3D rays
- $\boldsymbol{A} = \Sigma_{\mathbf{i}}\, p_i\, p_i'^T = \Sigma_{\mathbf{i}}\, p_i\, p_i^T\, \boldsymbol{R}^T = \boldsymbol{U}\, \boldsymbol{S}\, \boldsymbol{V}^T = (\boldsymbol{U}\, \boldsymbol{S}\, \boldsymbol{U}^T)\, \boldsymbol{R}^T$
- $\boldsymbol{V}^T = \boldsymbol{U}^T \boldsymbol{R}^T$
- $\boldsymbol{R} = \boldsymbol{V}\, \boldsymbol{U}^T$

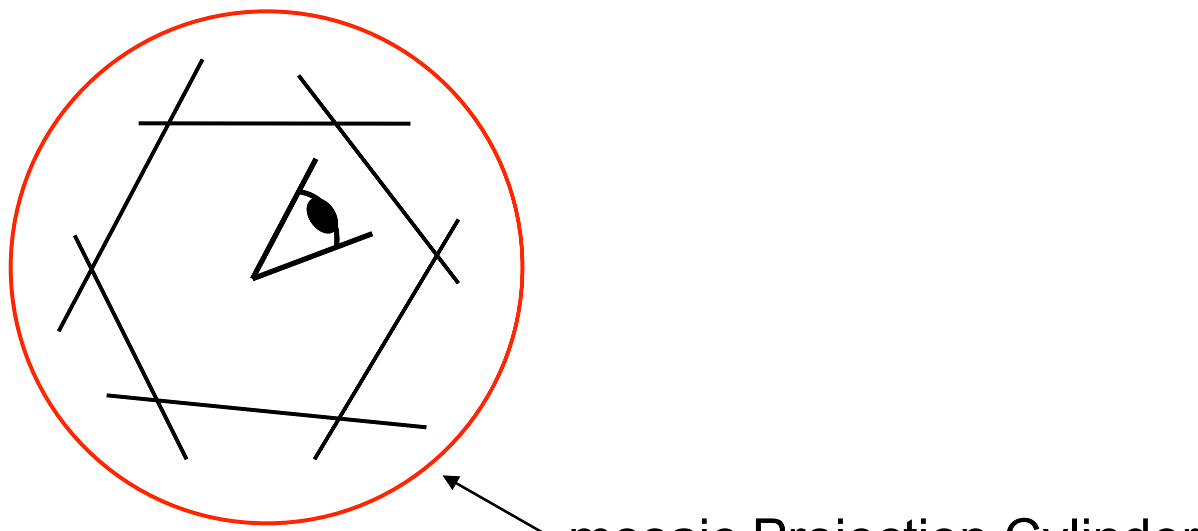
Stitching demo



Szeliski

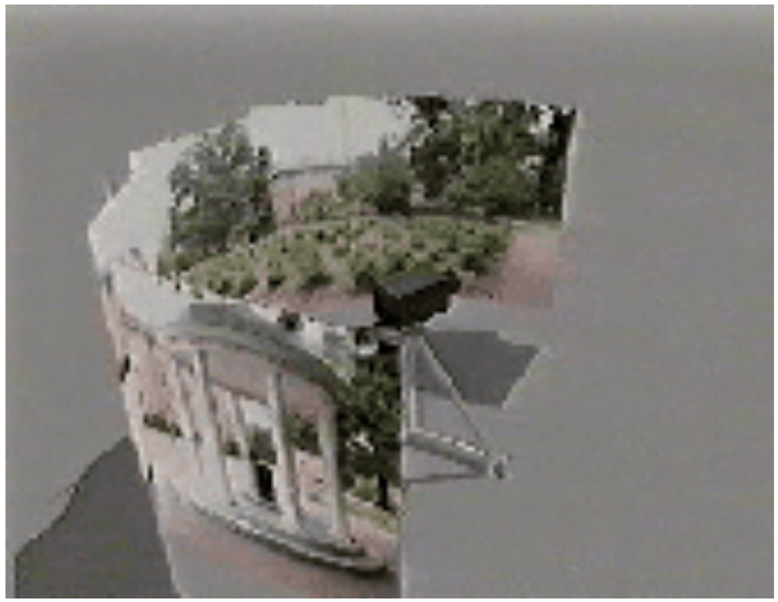
Panoramas

- What if you want a 360° field of view?



Szeliski

Cylindrical panoramas



- Steps
  - Reproject each image onto a cylinder
  - Blend
  - Output the resulting mosaic

Szeliski

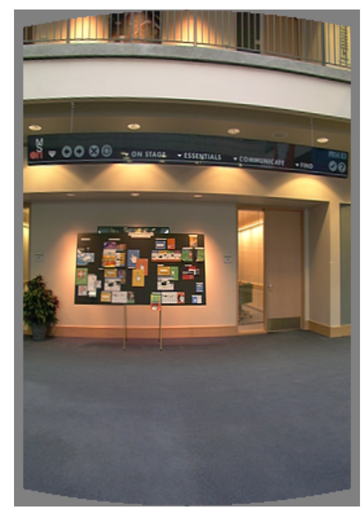
Cylindrical Panoramas

- Map image to cylindrical or spherical coordinates
  - need *known* focal length

Image 384x300

f = 180 (pixels)

f = 280

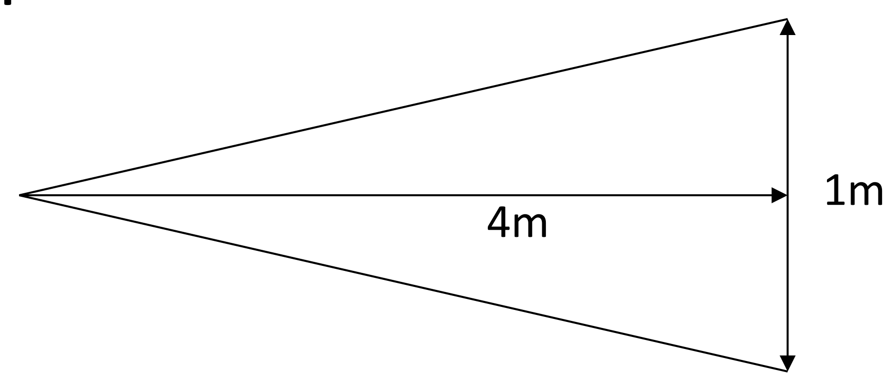


f = 380

Szeliski

Determining the focal length

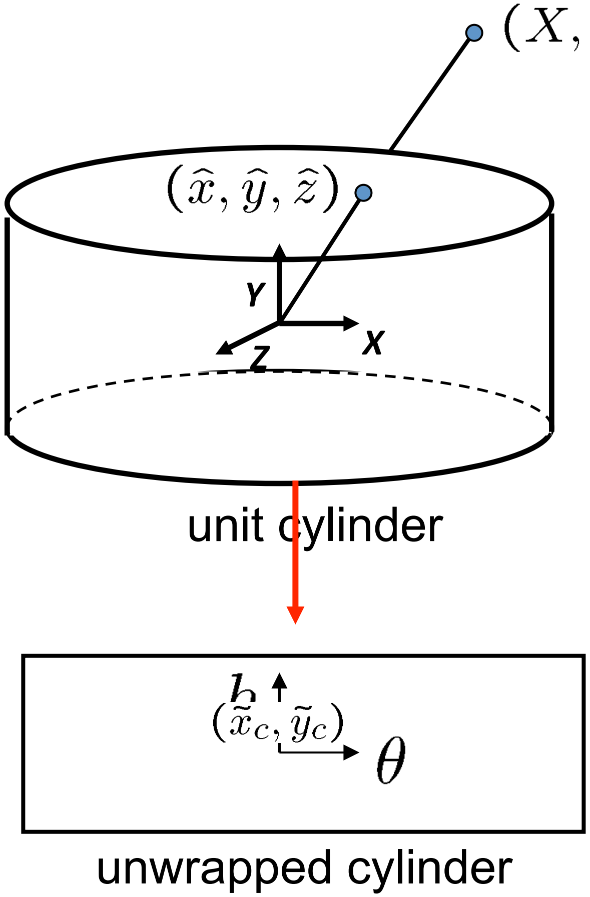
1. Initialize from homography $\boldsymbol{H}$ (see text or [SzSh'97])
2. Use camera's EXIF tags (approx.)
3. Use a tape measure



4. Ask your instructor

Szeliski

Cylindrical projection



(X, Y, Z) – Map 3D point (X,Y,Z) onto cylinder

$$(\hat{x}, \hat{y}, \hat{z}) = \frac{1}{\sqrt{X^2 + Z^2}}(X, Y, Z)$$

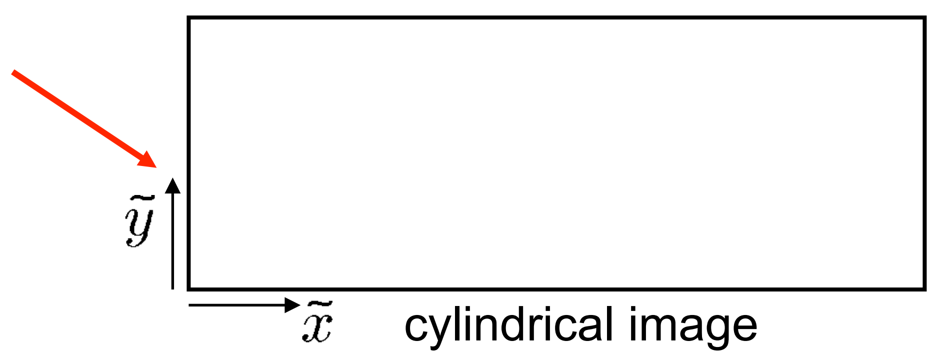
- Convert to cylindrical coordinates

$$(sin\theta, h, cos\theta) = (\hat{x}, \hat{y}, \hat{z})$$

- Convert to cylindrical image coordinates

$$(\tilde{x}, \tilde{y}) = (s\theta, sh) + (\tilde{x}_c, \tilde{y}_c)$$

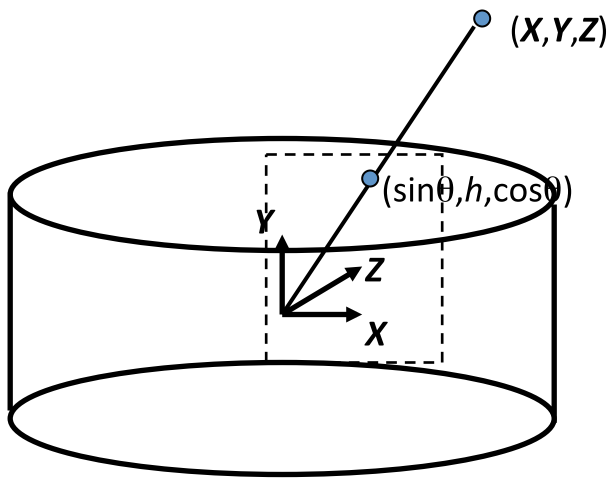
  – s defines size of the final image



Szeliski

Cylindrical warping

- Given focal length *f* and image center (x_c, y_c)

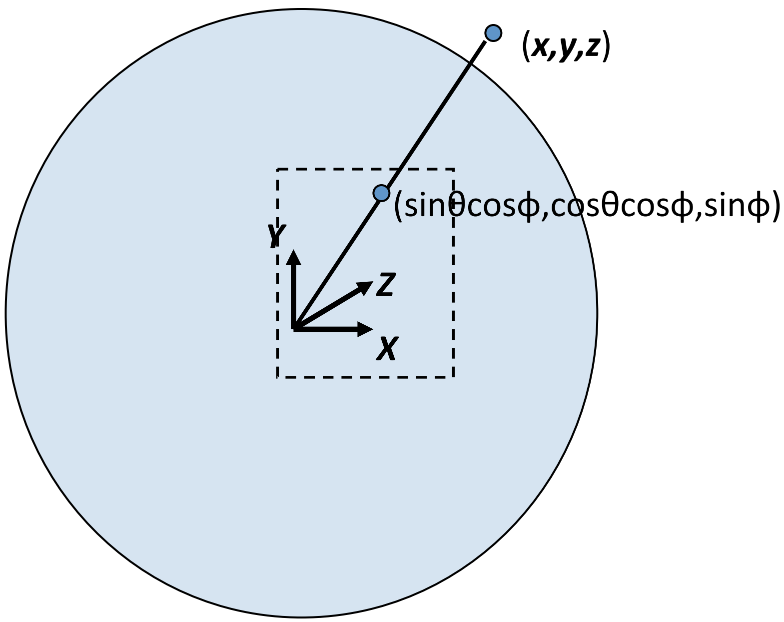


$$
\begin{aligned}
\theta &= (x_{cyl} - x_c)/f \\
h &= (y_{cyl} - y_c)/f \\
\hat{x} &= \sin\theta \\
\hat{y} &= h \\
\hat{z} &= \cos\theta \\
x &= f\hat{x}/\hat{z} + x_c \\
y &= f\hat{y}/\hat{z} + y_c
\end{aligned}
$$

Szeliski

Spherical warping

- Given focal length *f* and image center (x_c, y_c)

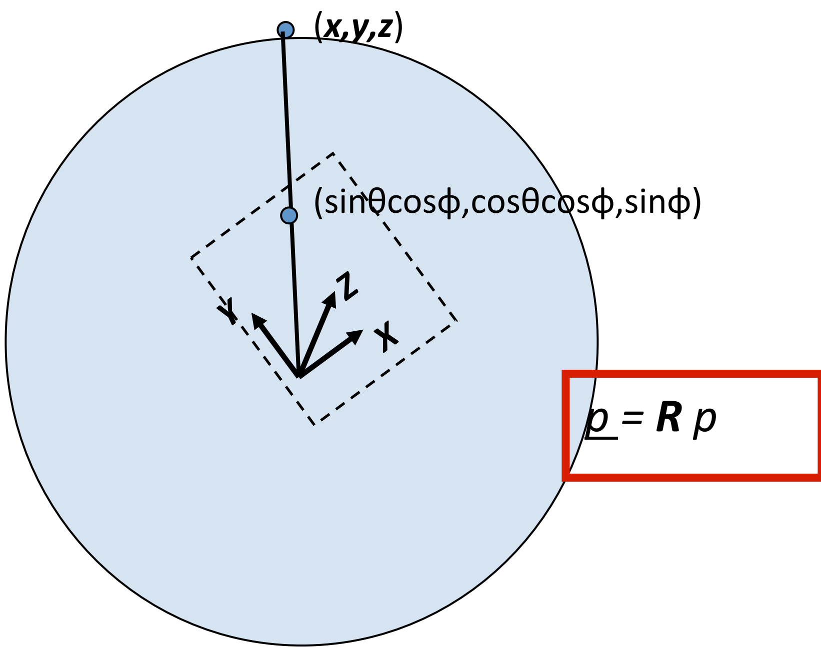


$$
\begin{aligned}
\theta &= (x_{cyl} - x_c)/f \\
\varphi &= (y_{cyl} - y_c)/f \\
\hat{x} &= \sin\theta \cos\varphi \\
\hat{y} &= \sin\varphi \\
\hat{z} &= \cos\vartheta \cos \\
x &= f\hat{x}/\hat{z} + x_c \\
y &= f\hat{y}/\hat{z} + y_c
\end{aligned}
$$

Szeliski

3D rotation

- Rotate image before placing on unrolled sphere

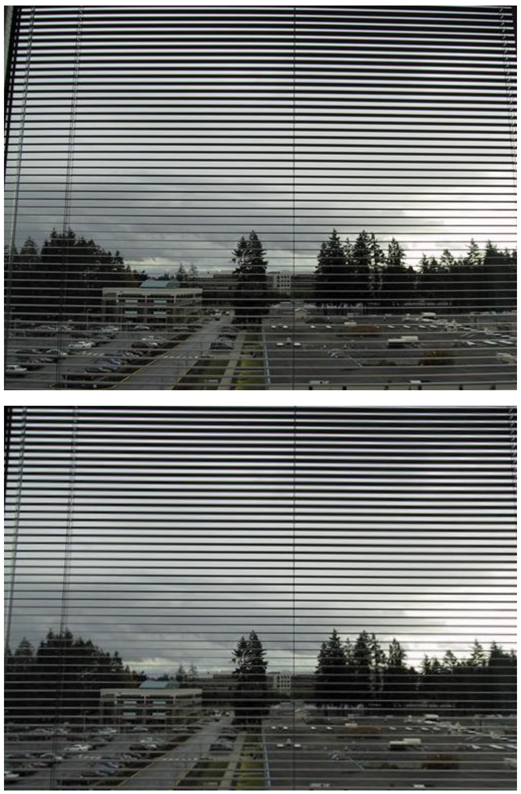


$$
\begin{aligned}
\theta &= (x_{cyl} - x_c)/f \\
\varphi &= (y_{cyl} - y_c)/f \\
\hat{x} &= \sin\theta \cos\varphi \\
\hat{y} &= \sin\varphi \\
\hat{z} &= \cos\vartheta \cos\varphi \\
x &= f\hat{x}/\hat{z} + x_c \\
y &= f\hat{y}/\hat{z} + y_c
\end{aligned}
$$

Szeliski

Radial distortion

- Correct for “bending” in wide field of view lenses



$$\begin{aligned}
\hat{r}^2 &= \hat{x}^2 + \hat{y}^2 \\
\hat{x}' &= \hat{x}/(1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4) \\
\hat{y}' &= \hat{y}/(1 + \kappa_1 \hat{r}^2 + \kappa_2 \hat{r}^4) \\
x &= f\hat{x}'/\hat{z} + x_c \\
y &= f\hat{y}'/\hat{z} + y_c
\end{aligned}$$

Szeliski

Fisheye lens

- Extreme "bending" in ultra-wide fields of view

$$\hat{r}^2 = \hat{x}^2 + \hat{y}^2$$

$$(\cos\theta \sin\phi, \sin\theta \sin\phi, \cos\phi) = s\,(x, y, z)$$

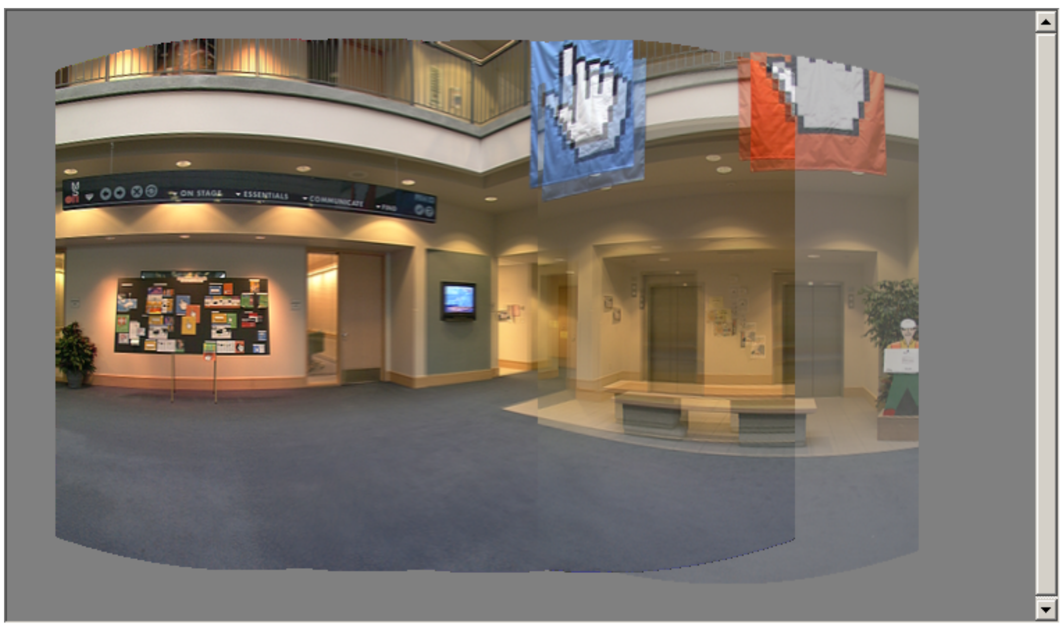
uations become

$$x' = s\phi\cos\theta = s\frac{x}{r}\tan^{-1}\frac{r}{z},$$
$$y' = s\phi\sin\theta = s\frac{y}{r}\tan^{-1}\frac{r}{z},$$

Szeliski

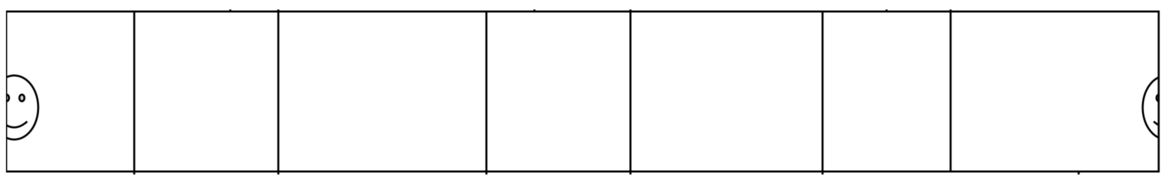
Image Stitching

1. Align the images over each other
   - camera pan ↔ translation on cylinder
2. Blend the images together



Szeliski

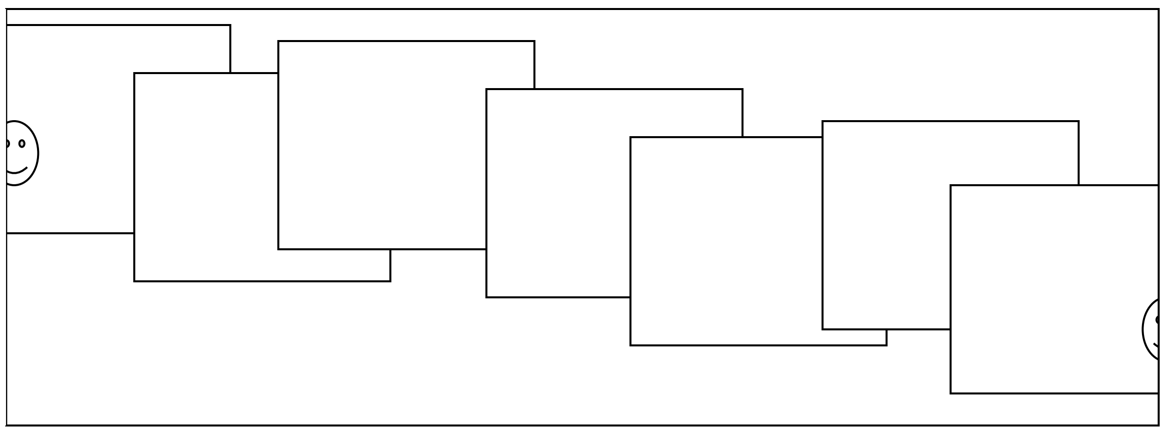
Assembling the panorama



- Stitch pairs together, blend, then crop

Szeliski

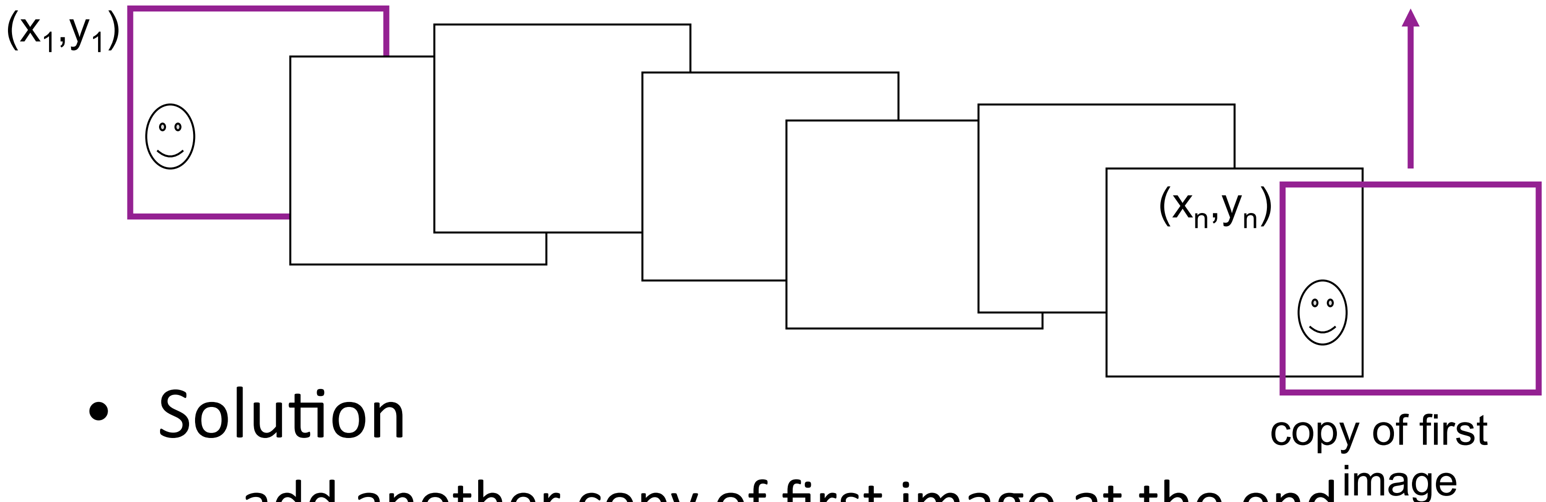
Problem: Drift



- Error accumulation
  - small (vertical) errors accumulate over time
  - apply correction so that sum = 0 (for 360° pan.)

Szeliski

Problem: Drift



- Solution
  - add another copy of first image at the end
  - this gives a constraint: $y_n = y_1$
  - there are a bunch of ways to solve this problem
    - add displacement of $(y_1 - y_n)/(n - 1)$ to each image after the first
    - compute a global warp: $y' = y + ax$
    - run a big optimization problem, incorporating this constraint
      - best solution, but more complicated
      - known as "bundle adjustment"

Szeliski

Full-view (360° spherical) panoramas

Szeliski

Full-view Panorama

Szeliski

Texture Mapped Model

Szeliski

Global alignment

- Register *all* pairwise overlapping images
- Use a 3D rotation model (one R per image)
- Use direct alignment (patch centers) or feature based
- *Infer* overlaps based on previous matches (incremental)
- Optionally *discover* which images overlap other images using feature selection (RANSAC)

Szeliski

Bundle adjustment formulations

Confidence / uncertainty of point i in image j

All pairs optimization:

$$E_{\text{all-pairs-2D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \| \tilde{\boldsymbol{x}}_{ik}(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j, \boldsymbol{R}_k, f_k) - \hat{\boldsymbol{x}}_{ik} \|^2, \quad (9.29)$$

Map 2D point i in image j to 2D point in image k

Full bundle adjustment, using 3-D point positions $\{\boldsymbol{x}_i\}$

$$E_{\text{BA-2D}} = \sum_i \sum_j c_{ij} \| \tilde{\boldsymbol{x}}_{ij}(\boldsymbol{x}_i; \boldsymbol{R}_j, f_j) - \hat{\boldsymbol{x}}_{ij} \|^2, \quad (9.30)$$

Map 3D point i in to 2D point in image i

Bundle adjustment using 3-D ray:

$$E_{\text{BA-3D}} = \sum_i \sum_j c_{ij} \| \tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j) - \boldsymbol{x}_i \|^2, \quad (9.31)$$

3-D ray from point i

All-pairs 3-D ray formulation:

$$E_{\text{all-pairs-3D}} = \sum_i \sum_{jk} c_{ij} c_{ik} \| \tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ij}; \boldsymbol{R}_j, f_j) - \tilde{\boldsymbol{x}}_i(\hat{\boldsymbol{x}}_{ik}; \boldsymbol{R}_k, f_k) \|^2. \quad (9.32)$$

3-D ray from points i and j

Projected point → $\tilde{\boldsymbol{x}}_{ij} \sim \boldsymbol{K}_j \boldsymbol{R}_j \boldsymbol{x}_i$ and $\boldsymbol{x}_i \sim \boldsymbol{R}_j^{-1} \boldsymbol{K}_j^{-1} \tilde{\boldsymbol{x}}_{ij}$, ← *3-D ray from point*

Recognizing Panoramas

Matthew Brown & David Lowe

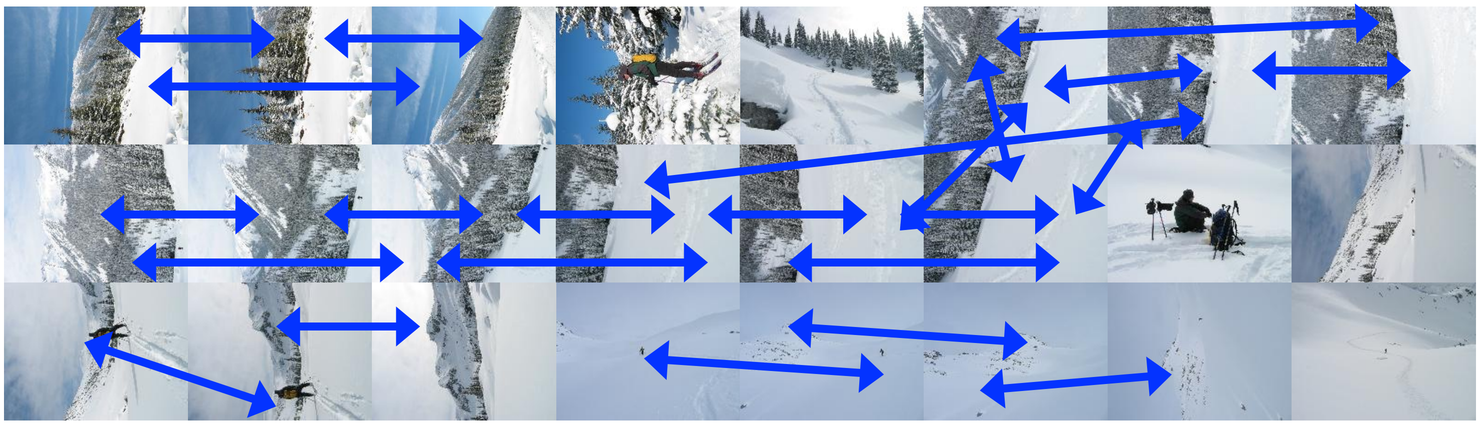
ICCV'2003

Recognizing Panoramas

[Brown & Lowe, ICCV'03]

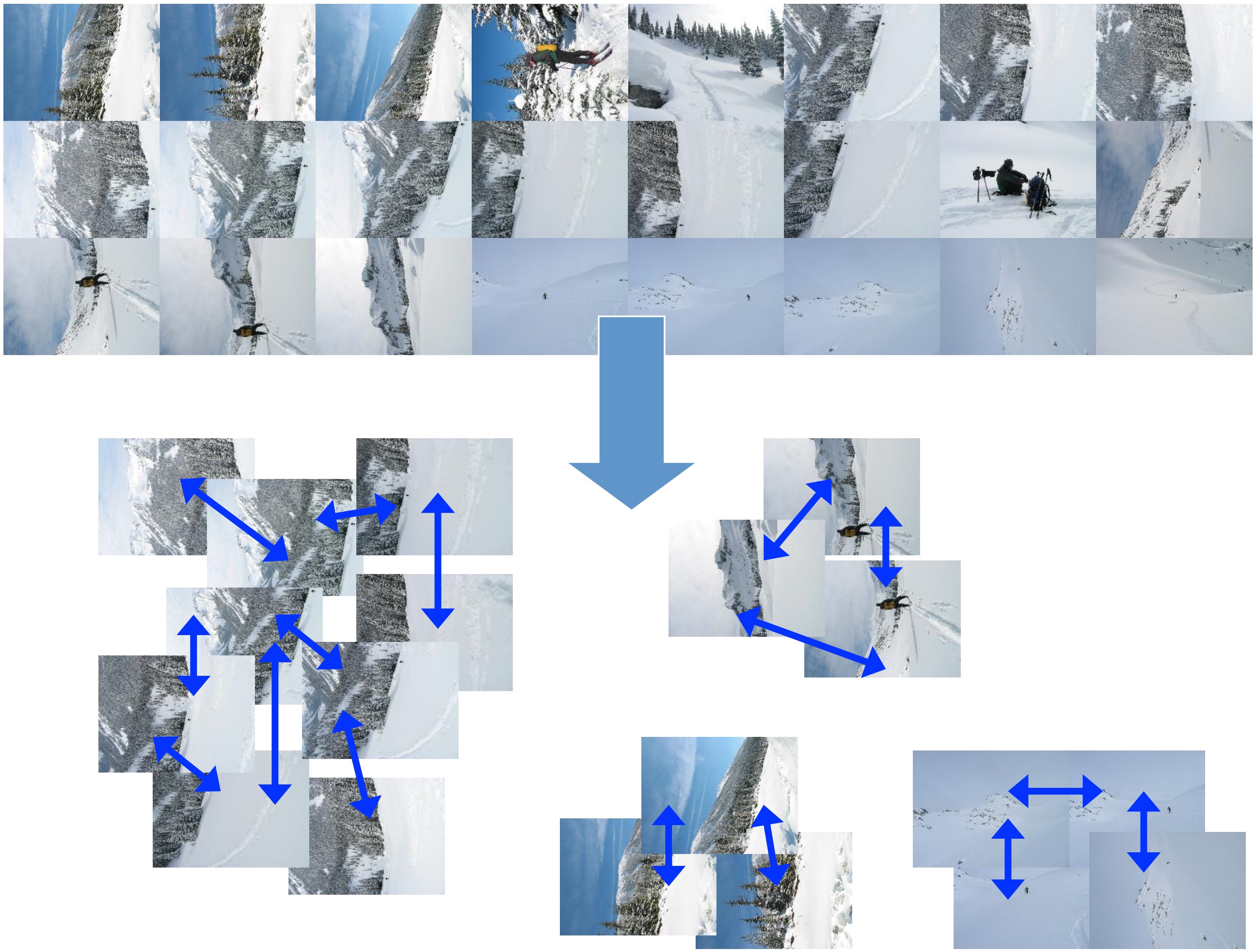
Szeliski

Finding the panoramas



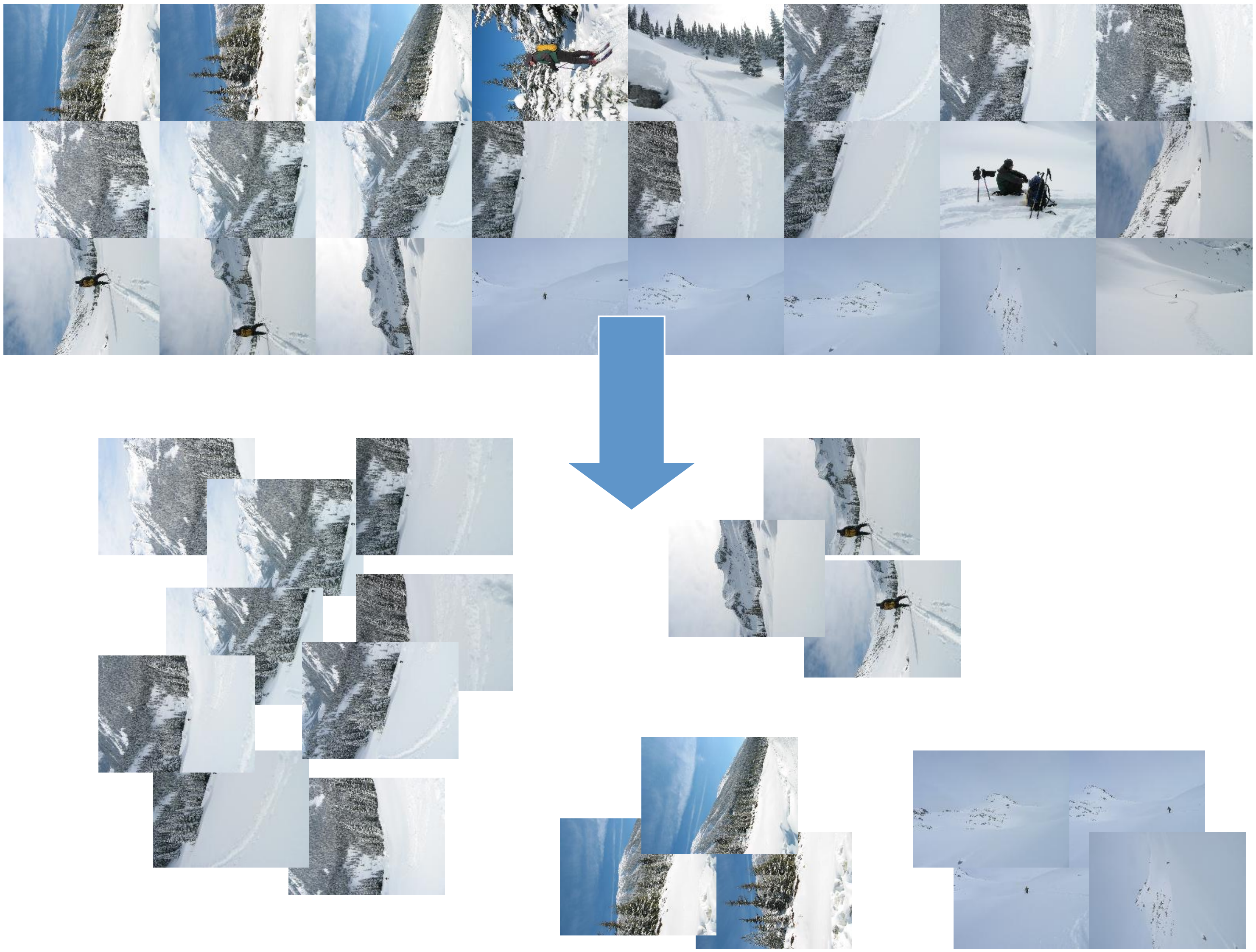
Szeliski

Finding the panoramas



Szeliski

Finding the panoramas

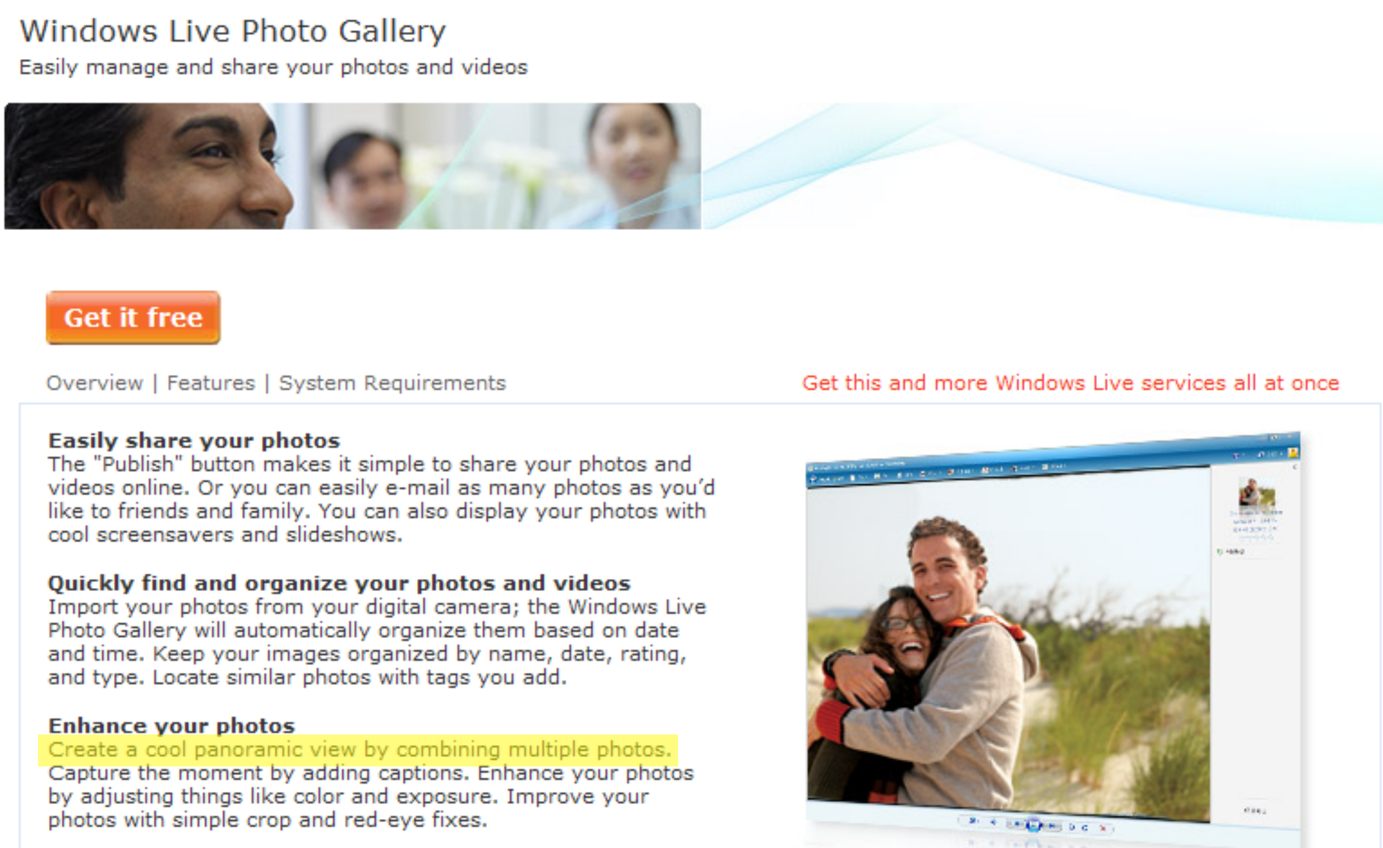


Szeliski

Finding the panoramas

Szeliski

Fully automated 2D stitching demo



http://get.live.com/photogallery/overview

Rec.pano.: system components

1. Feature detection and description
   – more uniform point density
2. Fast matching (hash table)
3. RANSAC filtering of matches
4. Intensity-based verification
5. Incremental bundle adjustment

- [M. Brown, R. Szeliski, and S. Winder. Multi-image matching using multi-scale oriented patches, CVPR'2005]

Szeliski

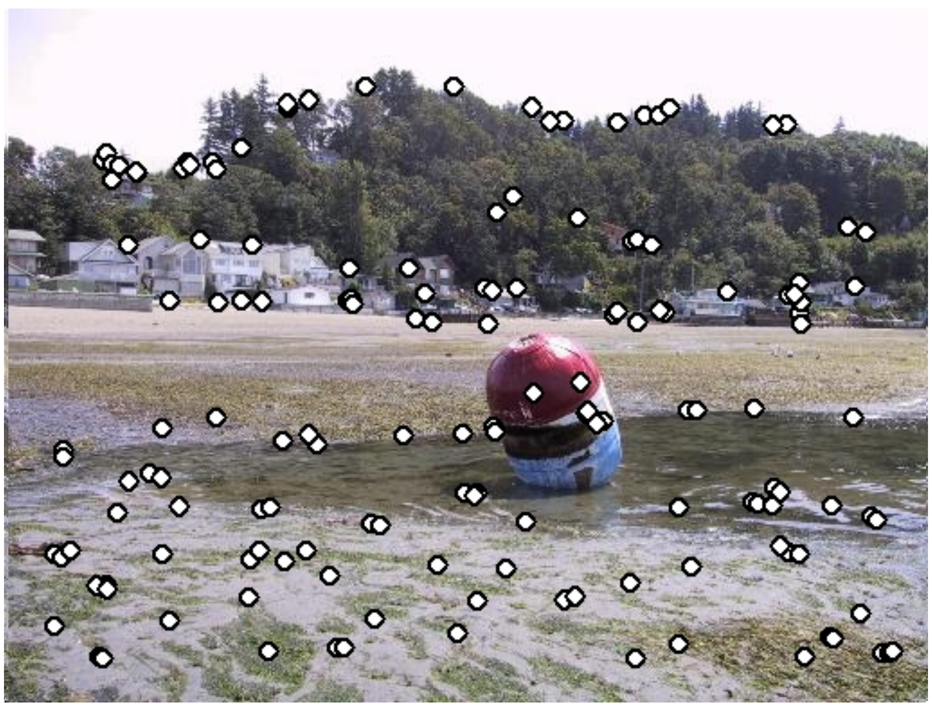
Multi-Scale Oriented Patches

- Interest points
  - Multi-scale Harris corners
  - Orientation from blurred gradient
  - Geometrically invariant to similarity transforms
- Descriptor vector
  - Bias/gain normalized sampling of local patch (8x8)
  - Photometrically invariant to affine changes in intensity

Szeliski

Features

- Distribute points evenly over the image



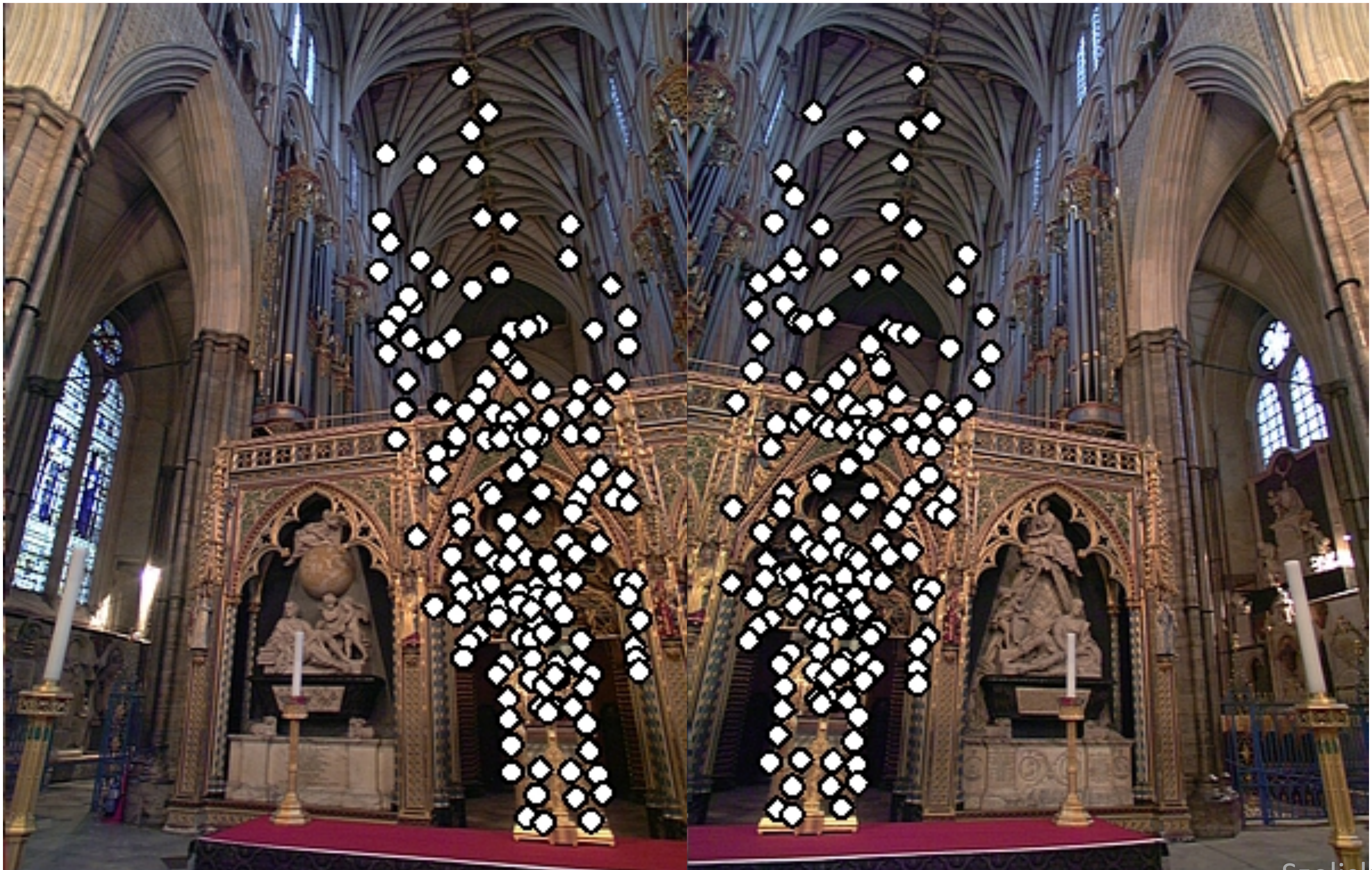
Szeliski

Descriptor Vector

- Orientation = blurred gradient
- Similarity Invariant Frame
  - Scale-space position (x, y, s) + orientation (θ)

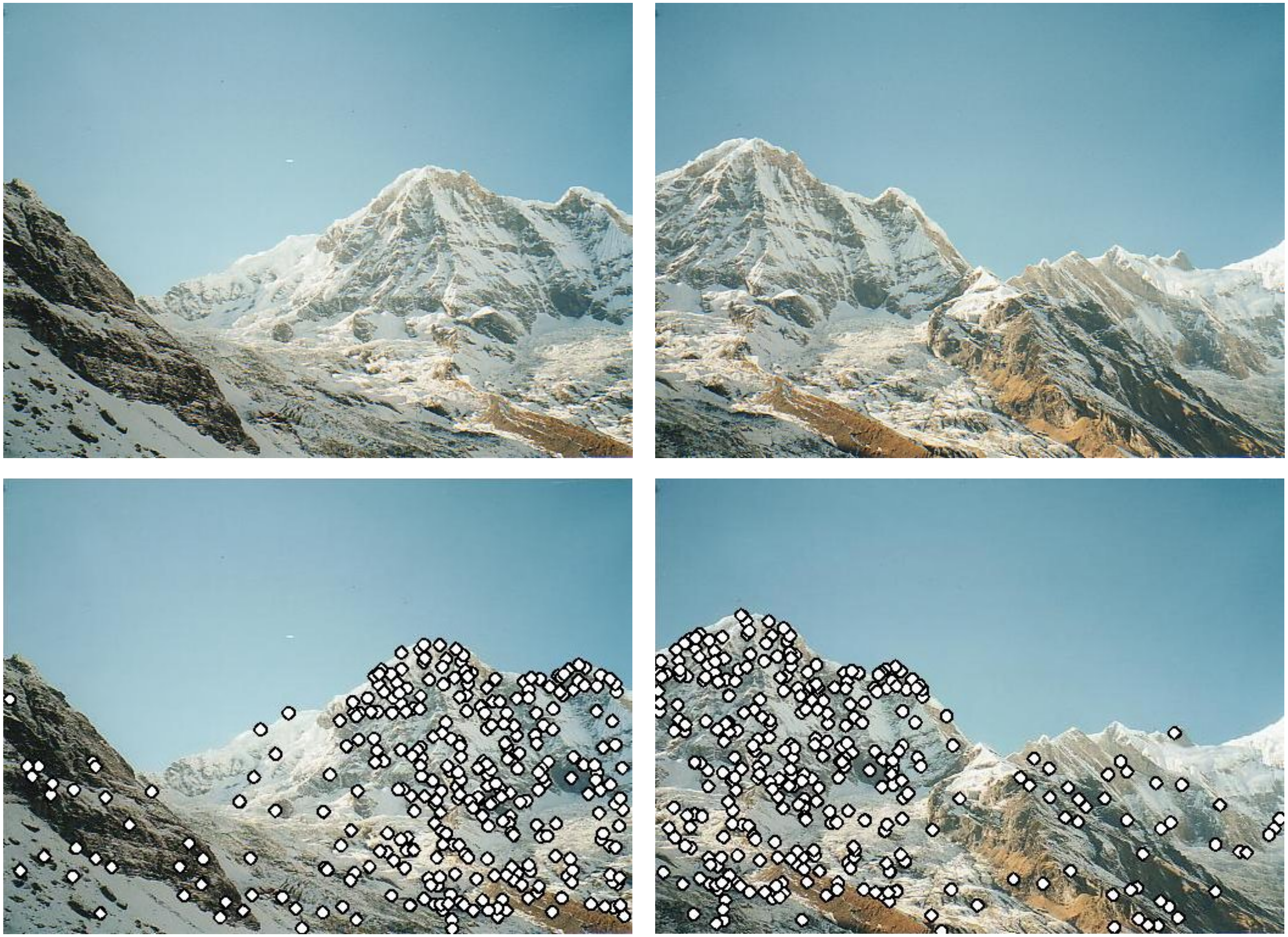
Szeliski

Probabilistic Feature Matching



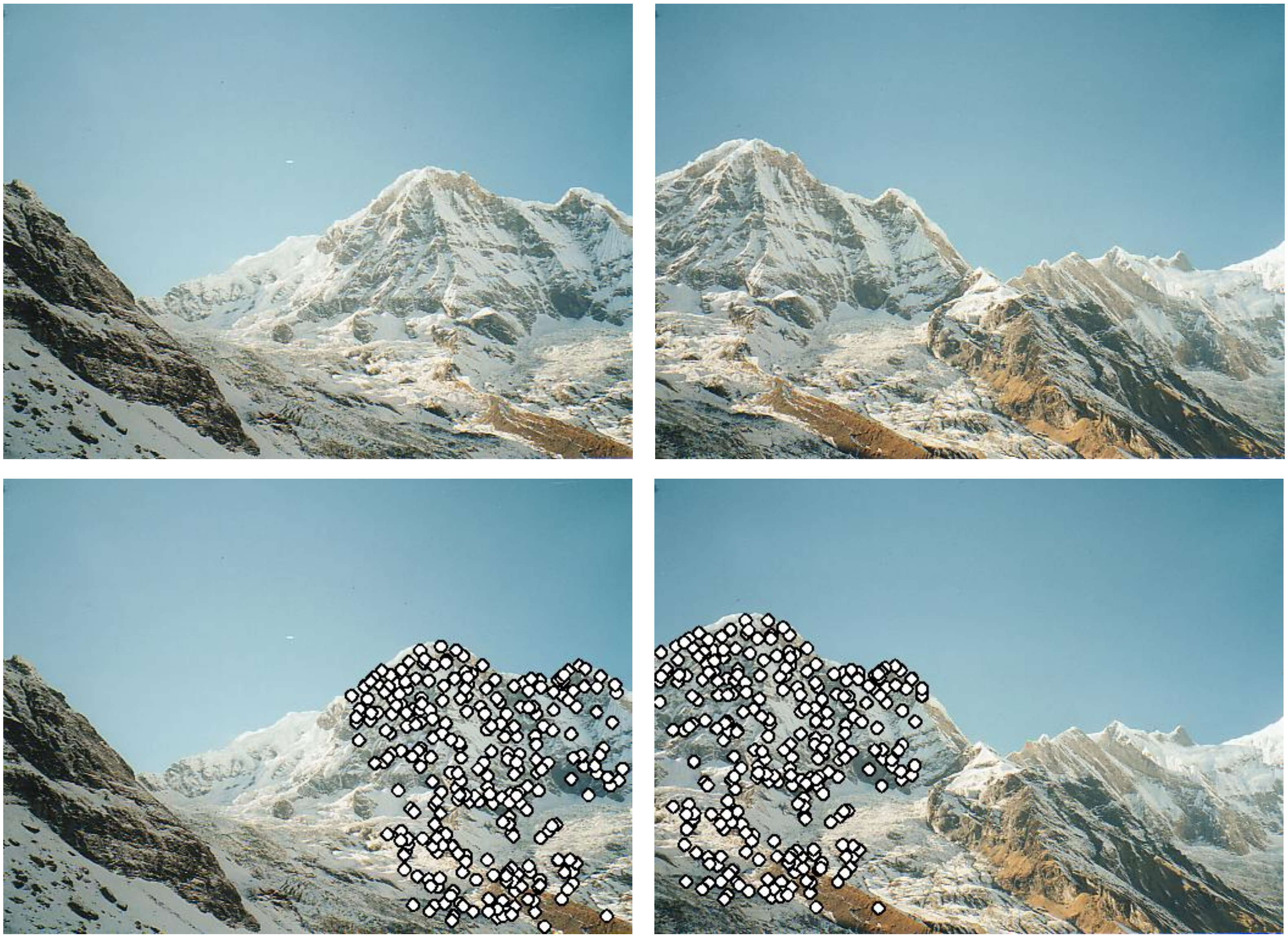
Richard Szeliski

RANSAC motion model



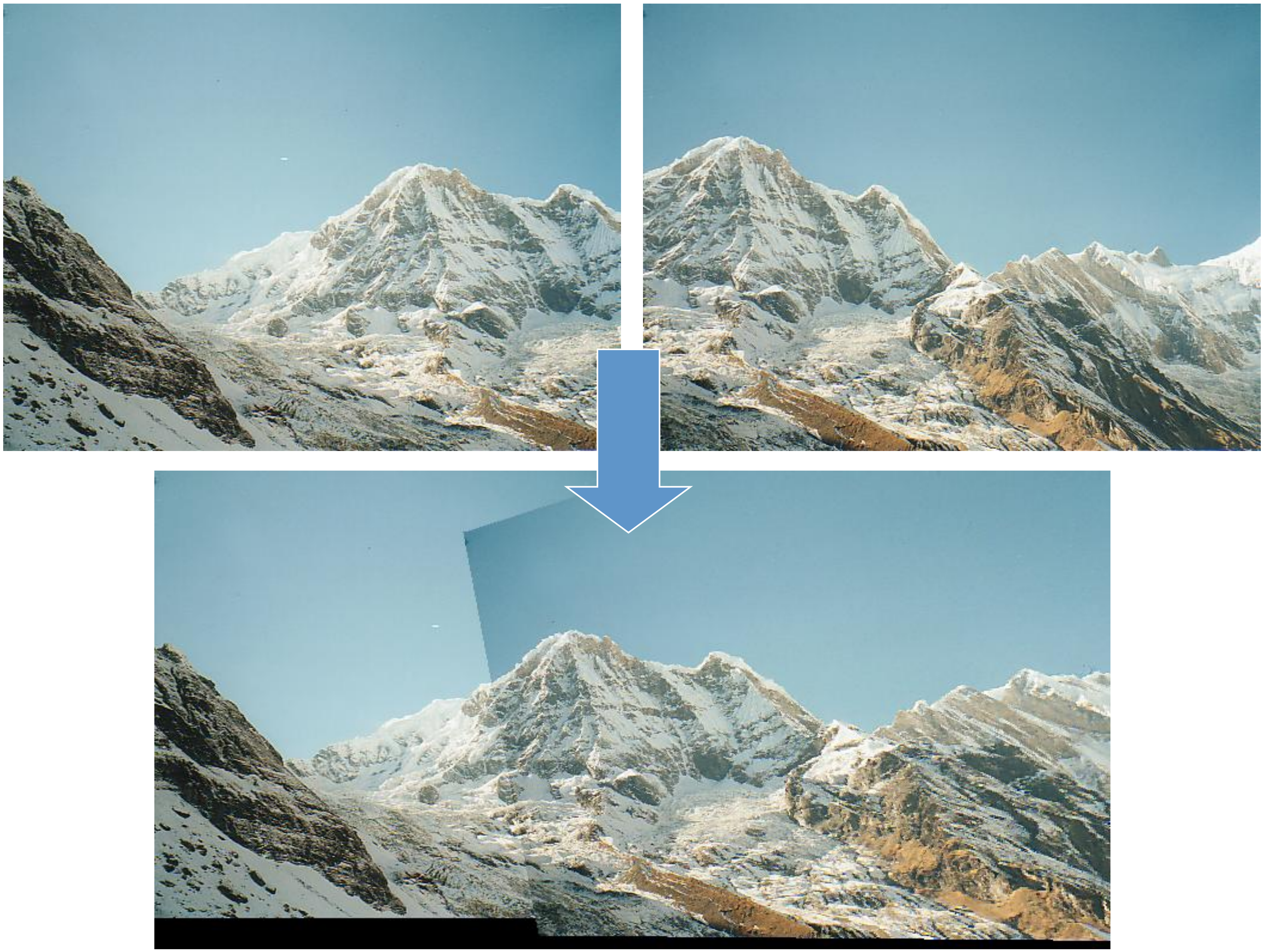
Szeliski

RANSAC motion model



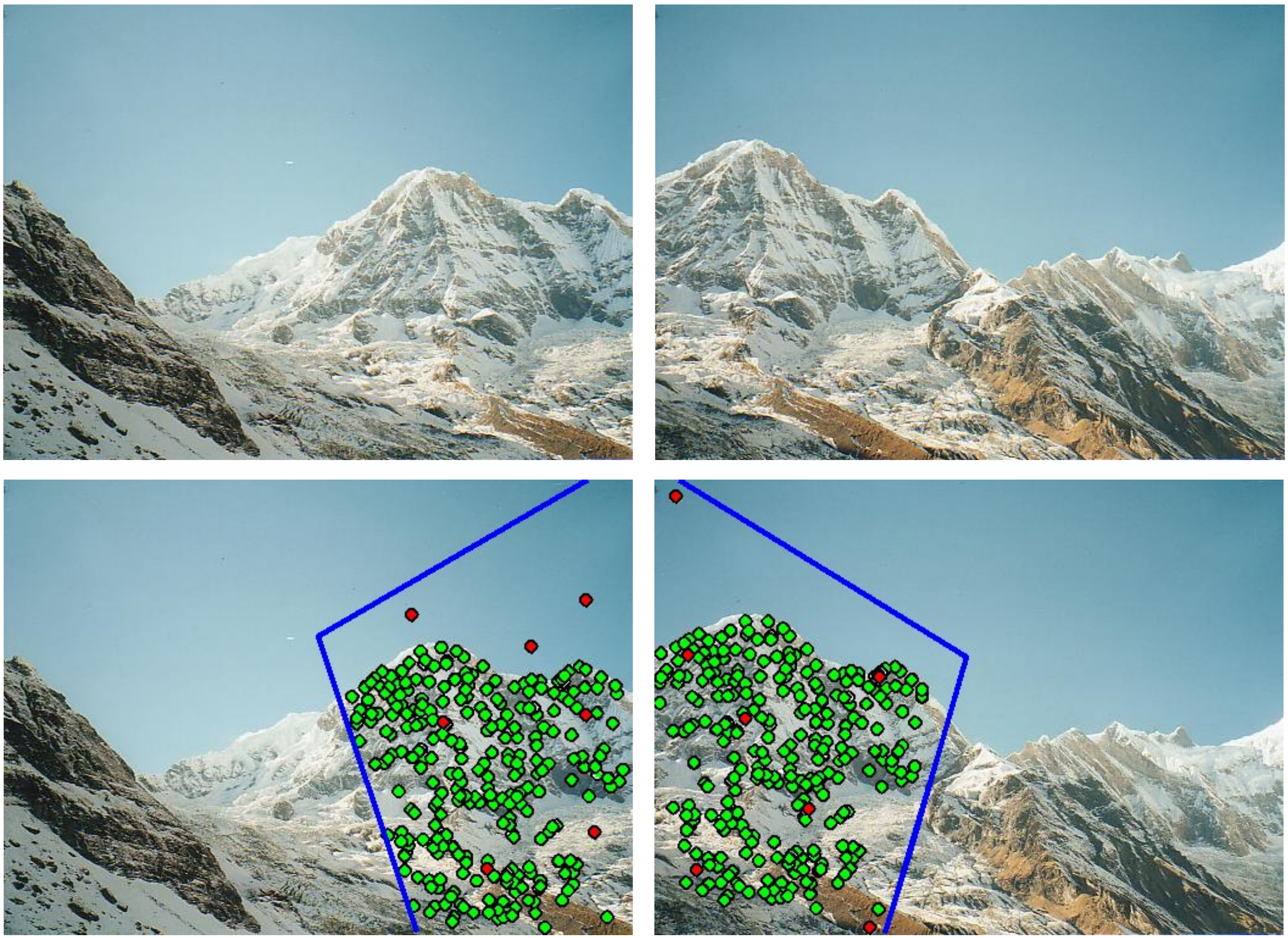
Szeliski

RANSAC motion model



Szeliski

Probabilistic model for verification



Szeliski

How well does this work?

Test on 100s of examples…

How well does this work?

Test on 100s of examples…

…still too many failures (5-10%)
for consumer application

Szeliski

Matching Mistakes: False Positive

Szeliski

Matching Mistakes: False Positive

Szeliski

Matching Mistake: False Negative

- Moving objects: large areas of disagreement

Szeliski

Matching Mistakes

- Accidental alignment
  - repeated / similar regions
- Failed alignments
  - moving objects / parallax
  - low overlap
  - “feature-less” regions (more variety?)
- No 100% reliable algorithm?

How can we fix these?

- Tune the feature detector
- Tune the feature matcher (cost metric)
- Tune the RANSAC stage (motion model)
- Tune the verification stage
- Use "higher-level" knowledge
  - e.g., typical camera motions

- → Sounds like a big "learning" problem
  - Need a large training/test data set (panoramas)

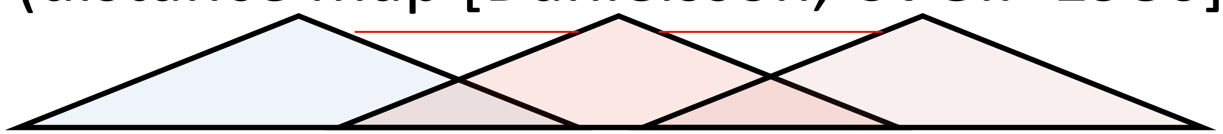
Szeliski

Image Blending

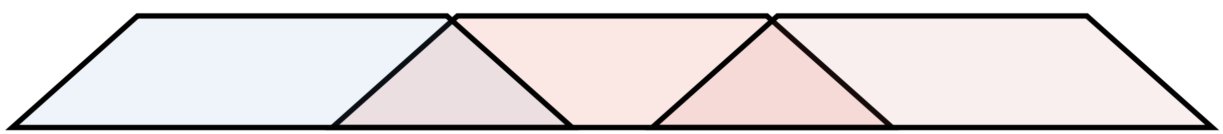
Szeliski

Image feathering

- Weight each image proportional to its distance from the edge
  (distance map [Danielsson, CVGIP 1980]

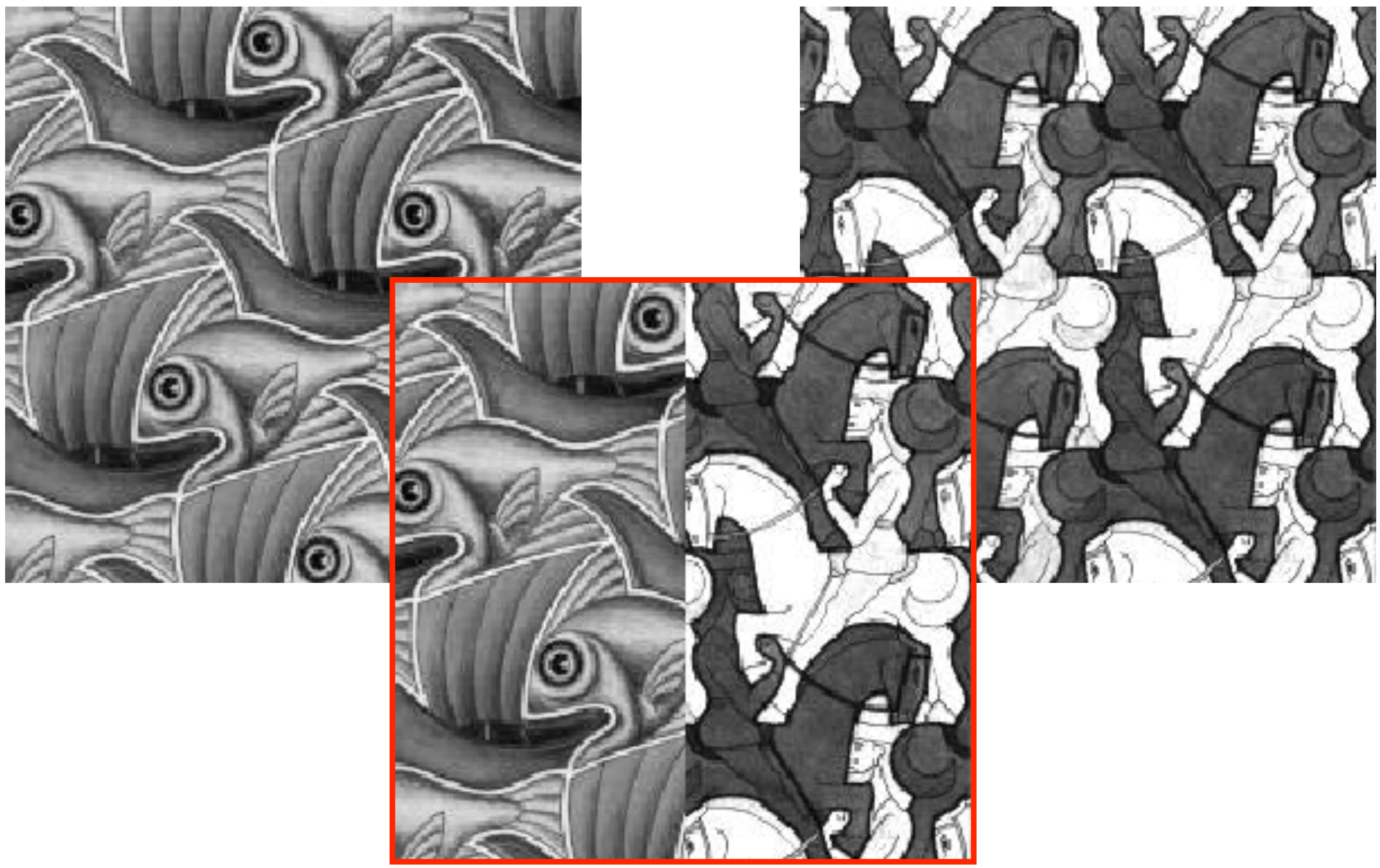


- 1. Generate *weight map* for each image
- 2. Sum up all of the weights and divide by sum: weights sum up to 1: $w_i' = w_i / (\sum_i w_i)$



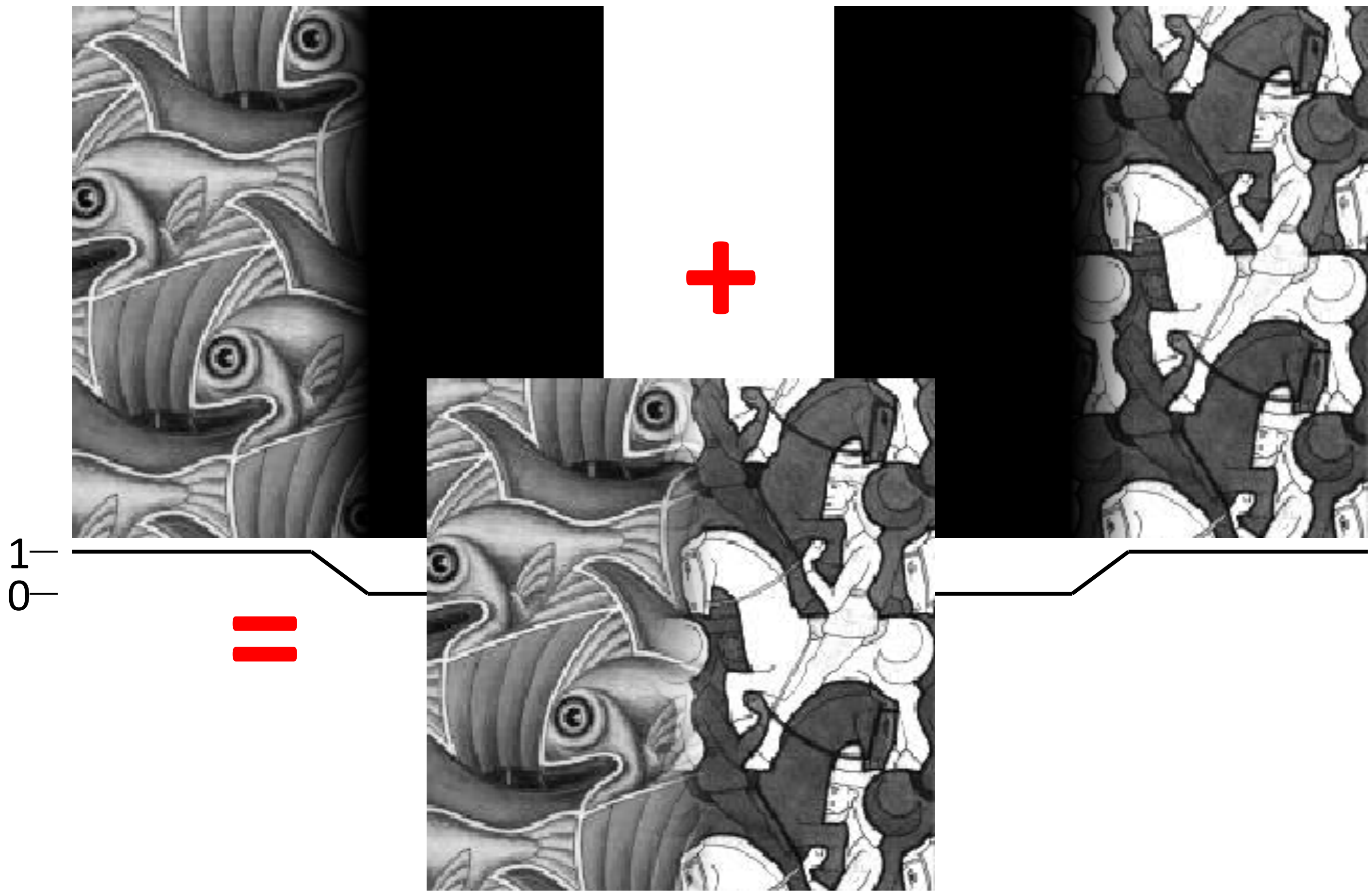
Szeliski

Image Feathering



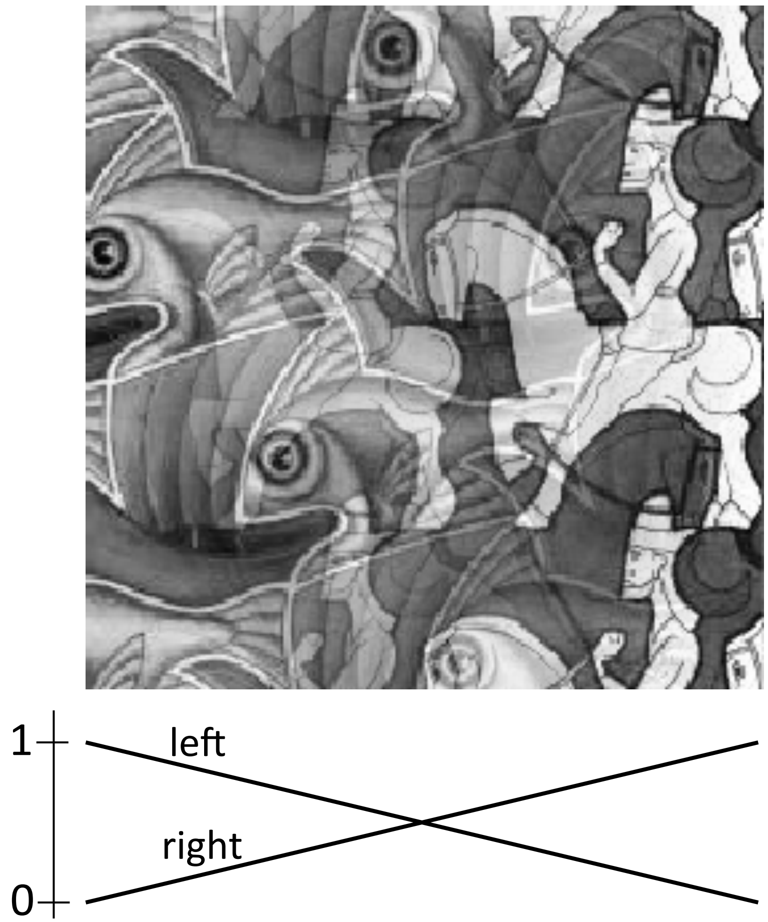
Szeliski

Feathering



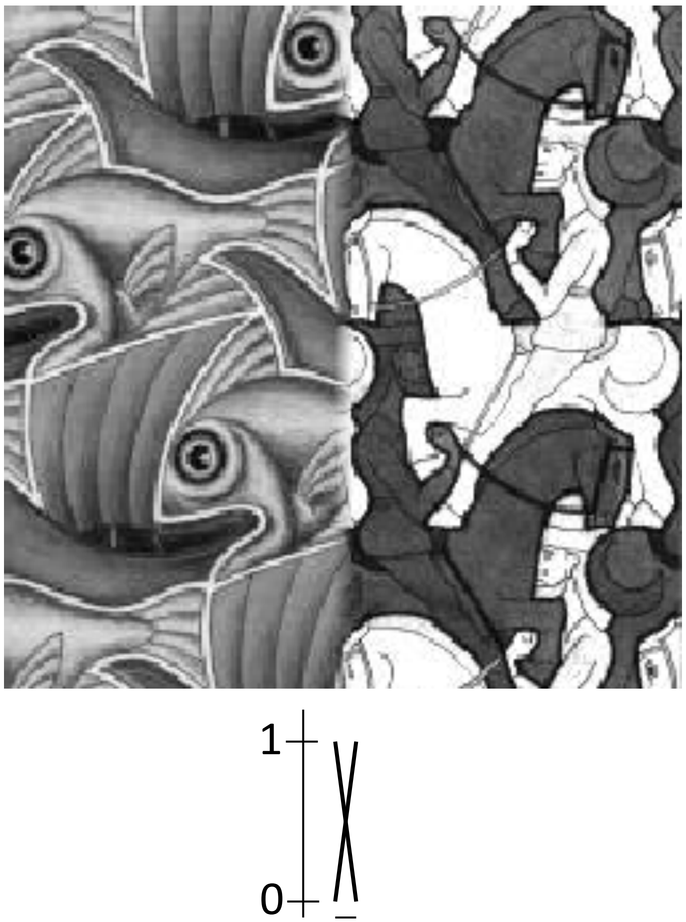
Szeliski

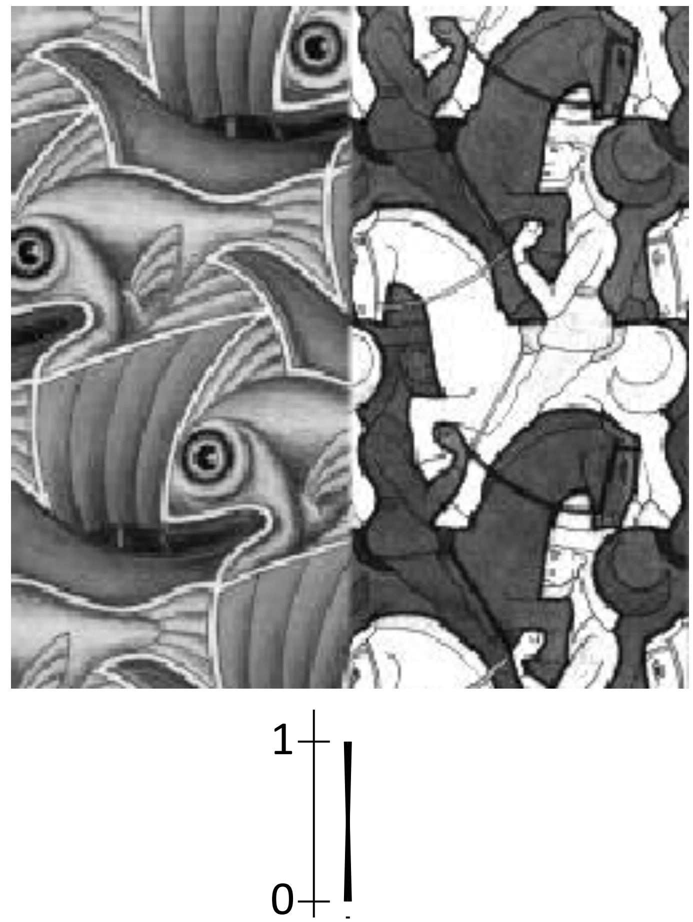
Effect of window size



Szeliski

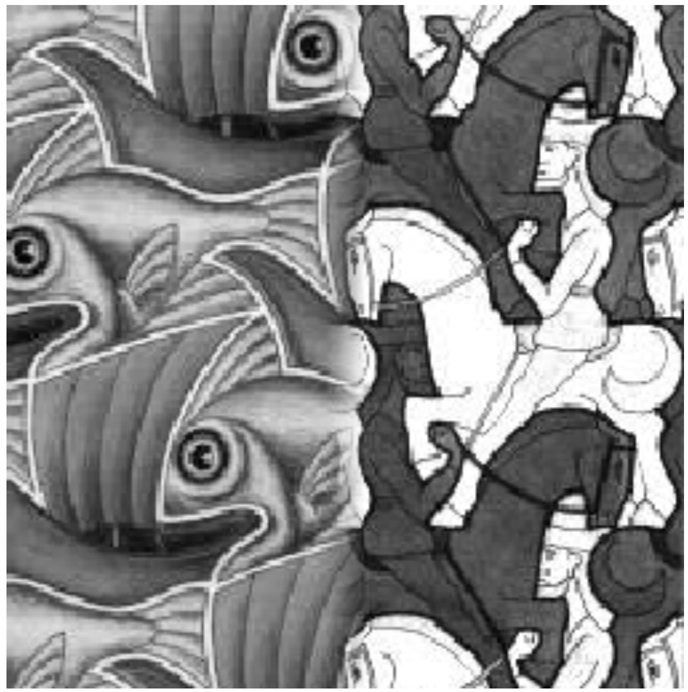
Effect of window size

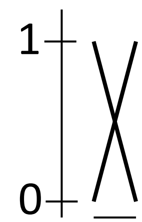




Szeliski

Good window size



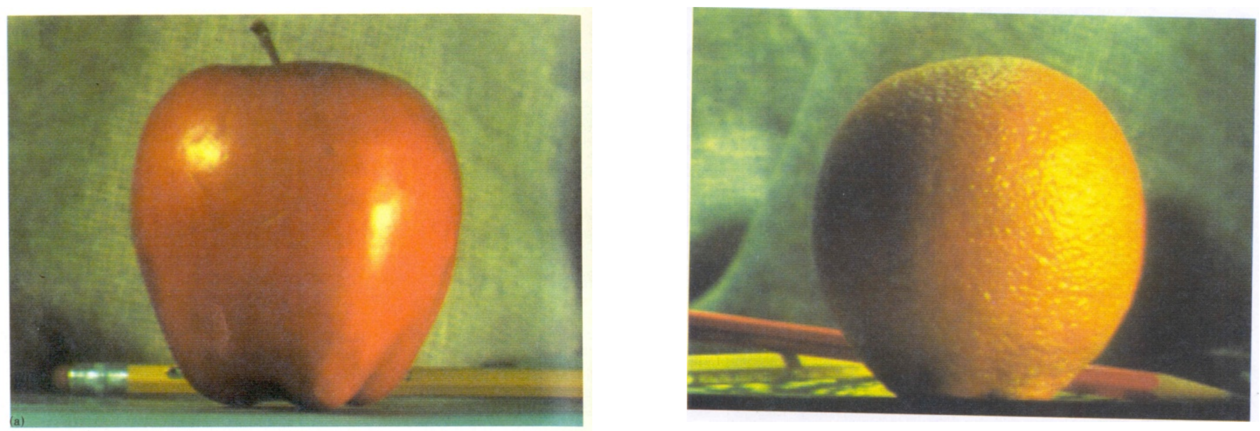


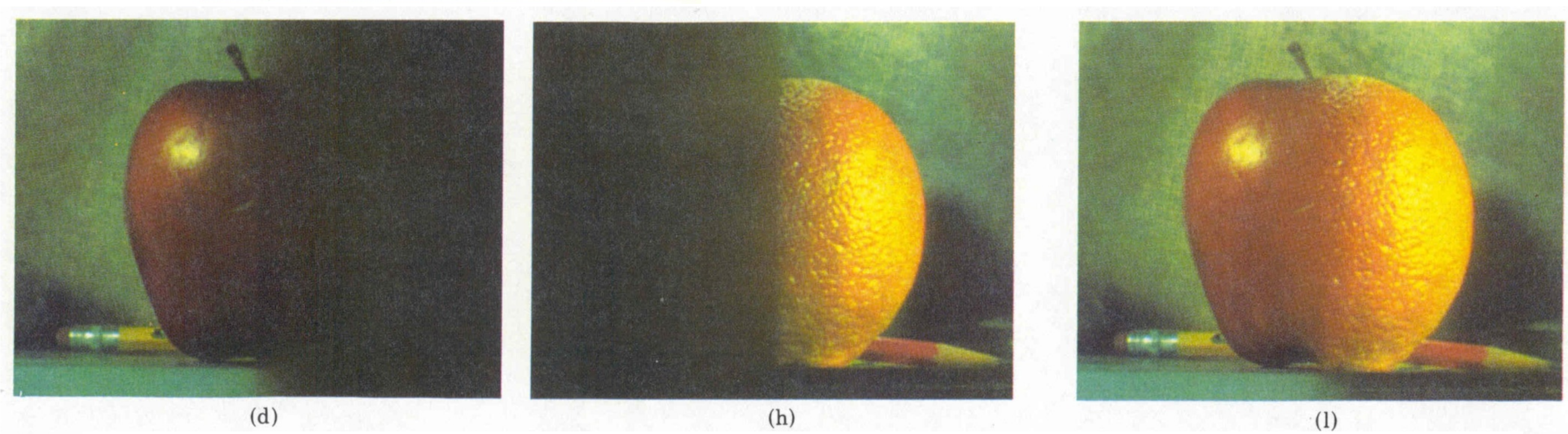
"Optimal" window: smooth but not ghosted

- Doesn't always work...

Szeliski

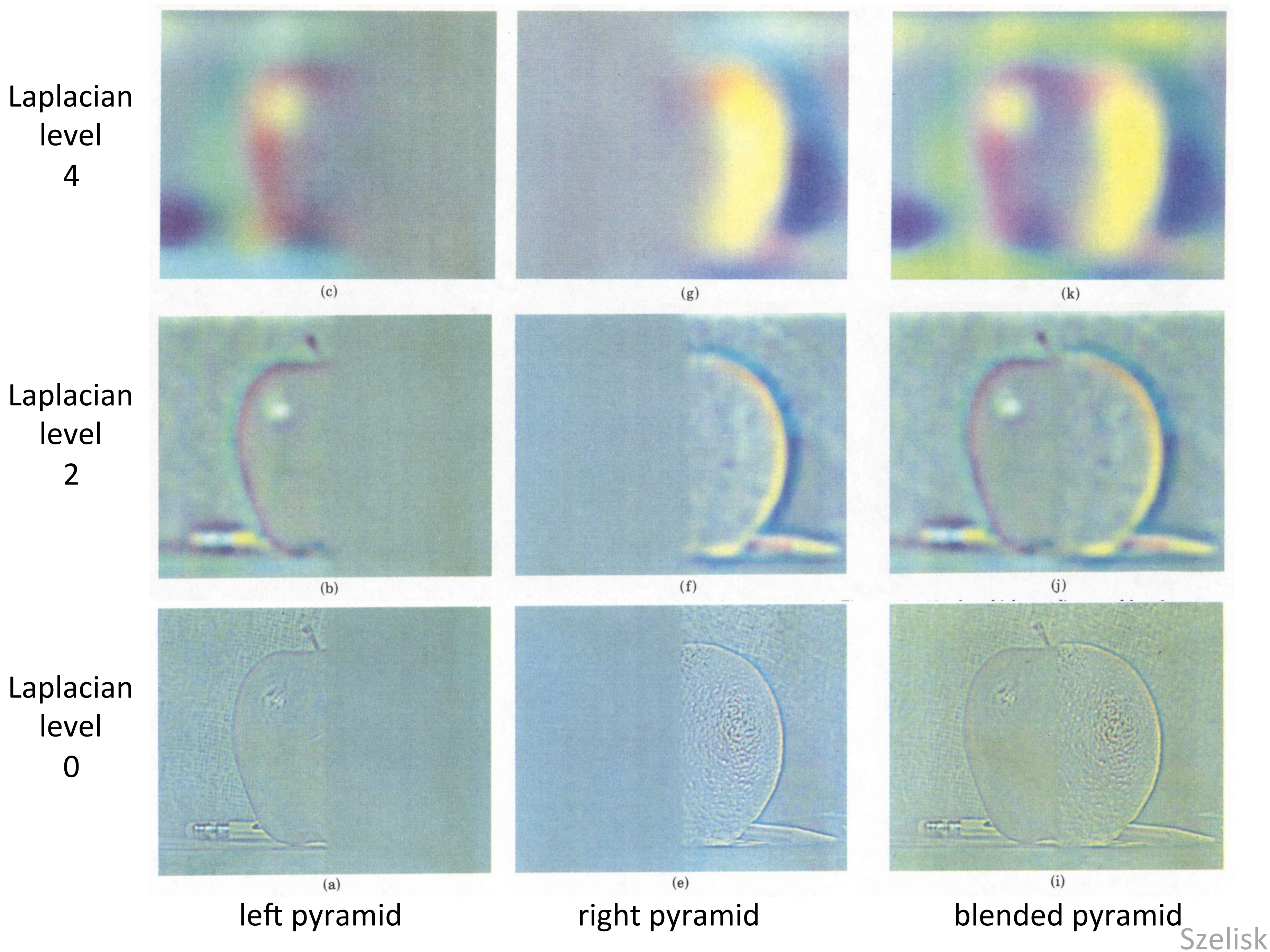
Pyramid Blending





Burt, P. J. and Adelson, E. H.,
A multiresolution spline with applications to image mosaics, ACM Transactions on Graphics, 42(4), October 1983, 217-236.

Szeliski


Laplacian level 4
Laplacian level 2
Laplacian level 0
(c)
(g)
(k)
(b)
(f)
(j)
(a)
(e)
(i)
left pyramid
right pyramid
blended pyramid
Szeliski

Laplacian image blend

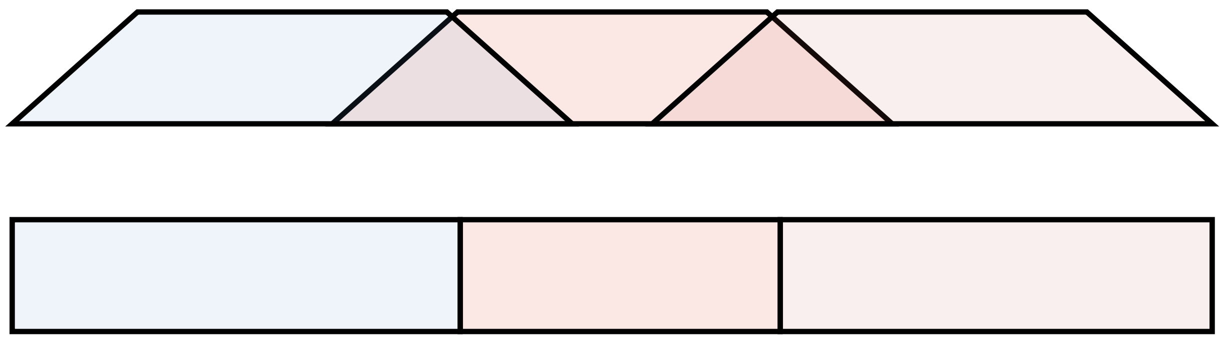
1. Compute Laplacian pyramid
2. Compute Gaussian pyramid on *weight* image (can put this in A channel)
3. Blend Laplacians using Gaussian blurred weights
4. Reconstruct the final image

- Q: How do we compute the original weights?
- A: For horizontal panorama, use *mid-lines*
- Q: How about for a general "3D" panorama?

Szeliski

Weight selection (3D panorama)

- Idea: use original feather weights to select *strongest* contributing image



- Can be implemented using L-∞ norm: ($p = 10$)
- $$w_i' = [w_i^p / (\sum_i w_i^p)]^{1/p}$$

Szeliski

Poisson Image Editing

- Blend the gradients of the two images, then integrate
- For more info: Perez et al, SIGGRAPH 2003

Szeliski

De-Ghosting

Szeliski

Local alignment (deghosting)

- Use local optic flow to compensate for small motions [Shum & Szeliski, ICCV'98]

Figure 3: Deghosting a mosaic with motion parallax: (a) with parallax; (b) after single deghosting step (patch size 32); (c) multiple steps (sizes 32, 16 and 8).

Szeliski

Local alignment (deghosting)

- Use local optic flow to compensate for radial distortion [Shum & Szeliski, ICCV'98]

Figure 4: Deghosting a mosaic with optical distortion: (a) with distortion; (b) after multiple steps.

Szeliski

Region-based de-ghosting

- Select only one image in *regions-of-difference* using weighted vertex cover [Uyttendaele *et al.*, CVPR'01]

(A) (B)

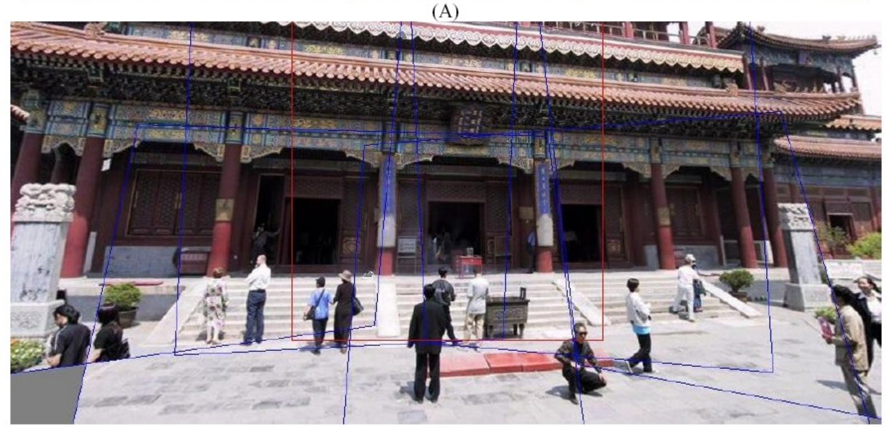
Figure 5 – (A) Ghosted mosaic. (B) Result of de-ghosting algorithm.

Szeliski

Region-based de-ghosting

- Select only one image in *regions-of-difference* using weighted vertex cover [Uyttendaele *et al.*, CVPR'01]

(A)



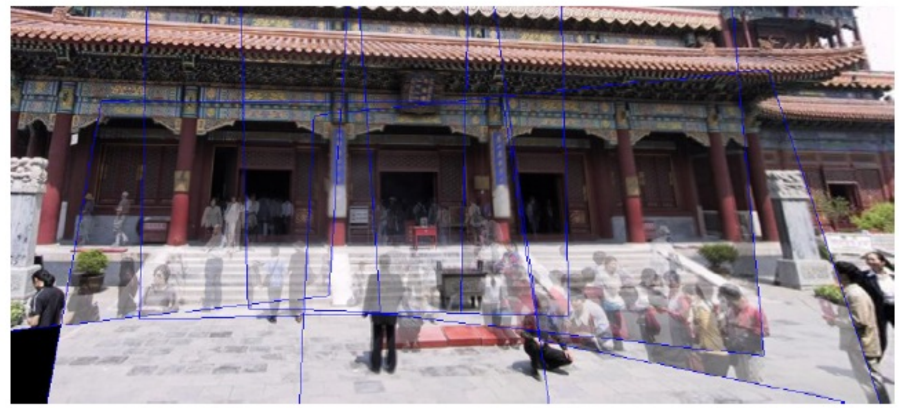
(B)

Figure 6 – (A) Ghosted mosaic. (B) Result of de-ghosting algorithm.

Szeliski

Cutout-based de-ghosting

- Select only one image per output pixel, using spatial continuity
- Blend across seams using gradient continuity ("Poisson blending")



[Agarwala *et al.*, SG'2004]

Cutout-based compositing

- Photomontage [Agarwala *et al.*, SG'2004]
- Interactively blend *different* images:
  group portraits

Figure 1 From a set of five source images (of which four are shown on the left), we quickly create a composite family portrait in which everyone is smiling and looking at the camera (right). We simply flip through the stack and coarsely draw strokes using the *designated source* image objective over the people we wish to add to the composite. The user-applied strokes and computed regions are color-coded by the borders of the source images on the left (middle).

Cutout-based compositing

- Photomontage [Agarwala *et al.*, SG'2004]
- Interactively blend *different* images:
  focus settings

Figure 2 A set of macro photographs of an ant (three of eleven used shown on the left) taken at different focal lengths. We use a global *maximum contrast* image objective to compute the graph-cut composite automatically (top left, with an inset to show detail, and the labeling shown directly below). A small number of remaining artifacts disappear after gradient-domain fusion (top, middle). For comparison we show composites made by Auto-Montage (top, right), by Haeberli's method (bottom, middle), and by Laplacian pyramids (bottom, right). All of these other approaches have artifacts; Haeberli's method creates excessive noise, Auto-Montage fails to attach some hairs to the body, and Laplacian pyramids create halos around some of the hairs.

Cutout-based compositing

- Photomontage [Agarwala *et al.*, SG'2004]
- Interactively blend *different* images:
  people's faces

Figure 6 We use a set of portraits (first row) to mix and match facial features, to either improve a portrait, or create entirely new people. The faces are first hand-aligned, for example, to place all the noses in the same location. In the first two images in the second row, we replace the closed eyes of a portrait with the open eyes of another. The user paints strokes with the *designated source* objective to specify desired features. Next, we create a fictional person by combining three source portraits. Gradient-domain fusion is used to smooth out skin tone differences. Finally, we show two additional mixed portraits.

Szeliski

Other types of mosaics

- Can mosaic onto *any* surface if you know the geometry
  - See NASA's Visible Earth project for some stunning earth mosaics
    - http://earthobservatory.nasa.gov/Newsroom/BlueMarble/

Szeliski

Final thought: What is a “panorama”?

- Tracking a subject
- Repeated (best) shots

- Multiple exposures

- “Infer” what photographer wants?

Szeliski

Slide Credits

- Steve Seitz
- Kristen Grauman
- Alyosha Efros

Next time: Parametric Motion and Optic Flow

- The ‘Direct Motion’ analogue to today…