Image Stitching and Alignment

Multiple Images

- So far, algorithms deal with *a single*, *static* image
- In the real world, a static pattern is a rarity, continuous motion and change are the rule
- Human eyes are well-equipped to take advantage of motion or change in *an image sequence*
- *Stitching (Alignment) and Motion*
  - Stitching has a "global" model – all pixel movement can be explained by a simple mathematic model (far field, pure rotational, pure translation)
  - 2D motion field is a "local" model – pixels by themselves (similarity in a local neighborhood only)

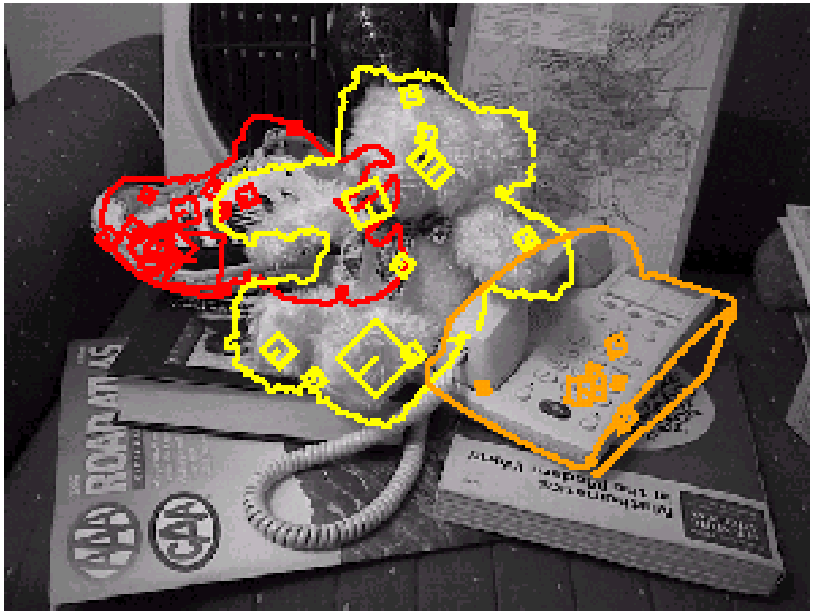
General Taxonomy

- Camera motion and the Scene is static
  - Driving, panorama
  - Near field (hard) vs. Far field (easy)
  - General camera motion (hard) vs. special camera motion (e.g., rotation only, easier)
  - General scene (hard) vs. special scene (planar, easier)
- Object motion and the camera is stationary
  - Surveillance
  - Background modeling and subtraction
- Both camera and object are moving
  - Sports video, driving, diving, etc.

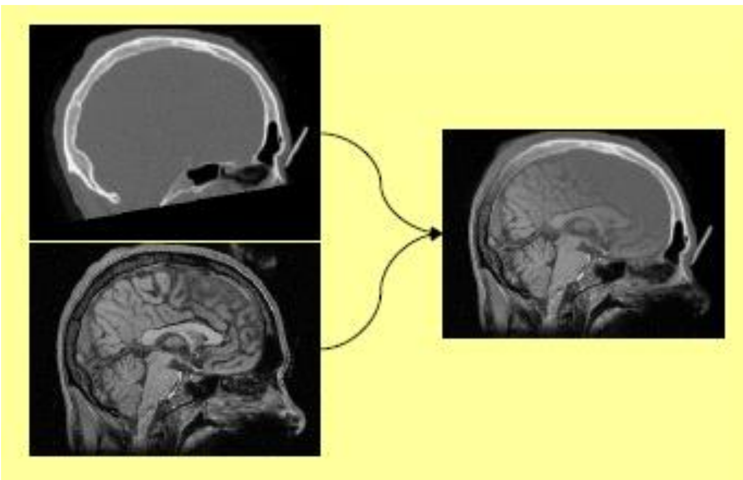
Alignment

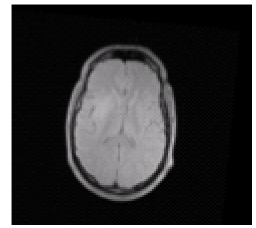
- Homographies
- Rotational Panoramas
- RANSAC
- Global alignment
- Warping
- Blending

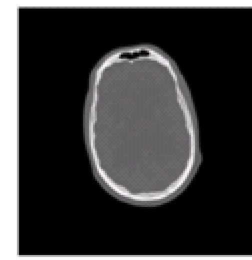
Motivation: Recognition

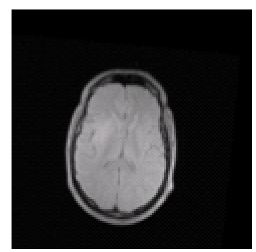


Motivation: medical image registration









Motivation: Mosaics

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

Motivation: Mosaics

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

Motivation: Mosaics

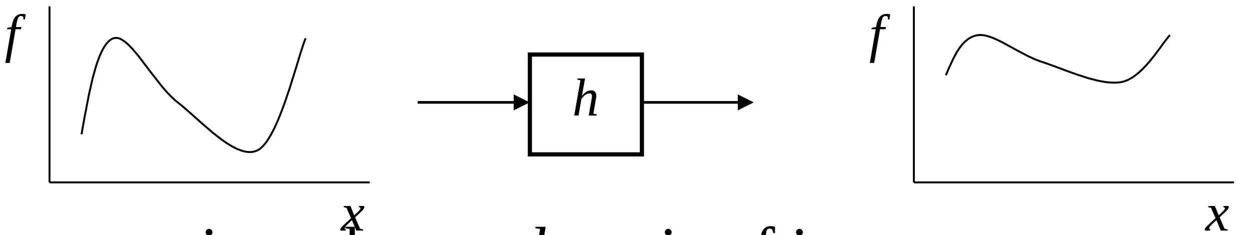
- Getting the whole picture
  - Consumer camera: 50° x 35°
  - Human Vision: 176° x 135°

Motion models

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

Image Warping

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



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

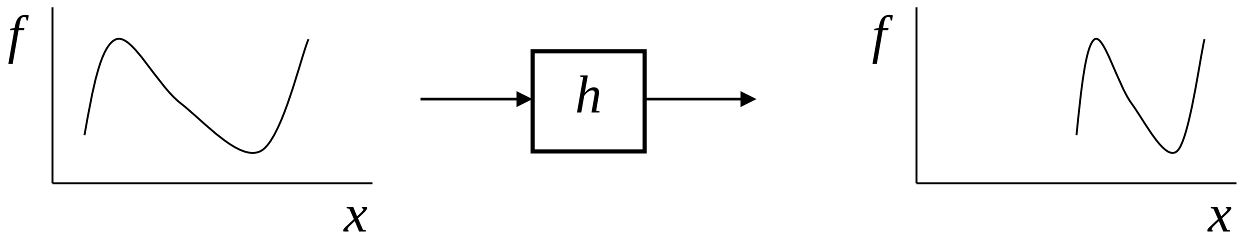
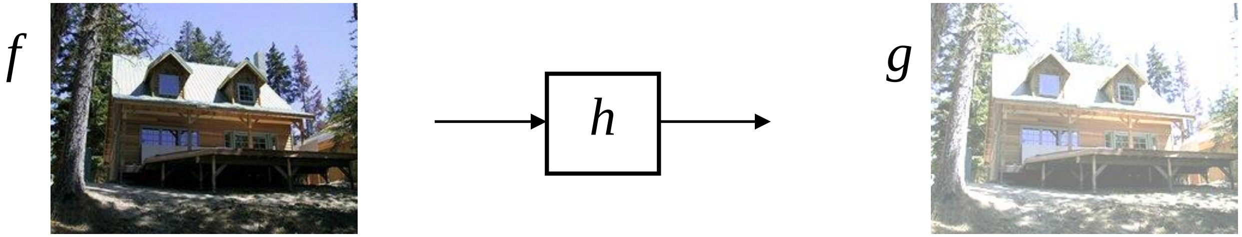
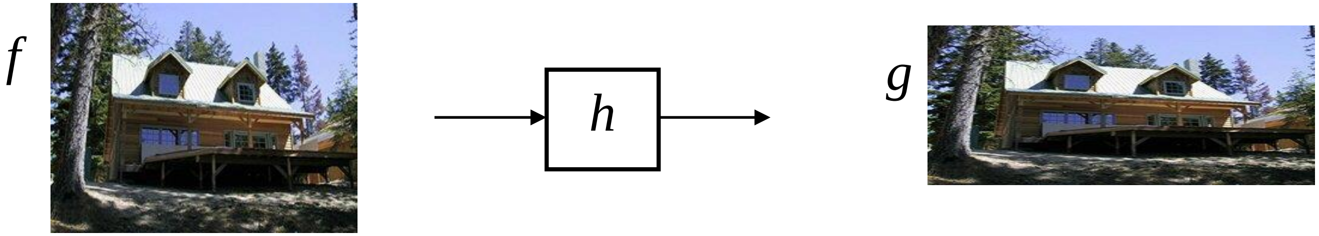


Image Warping

- 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

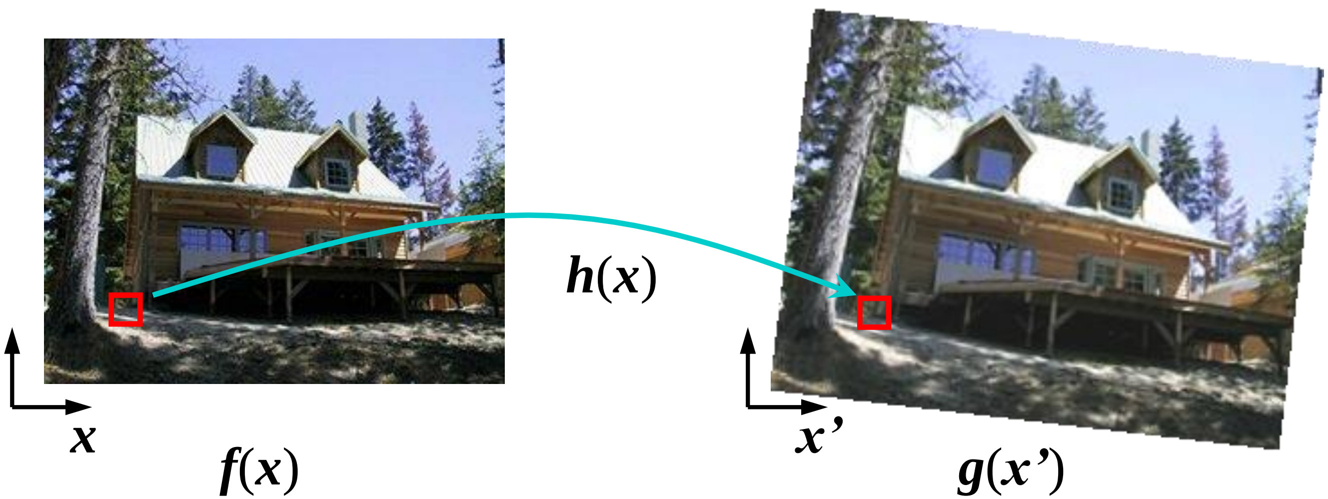

affine


perspective


cylindrical

Image Warping

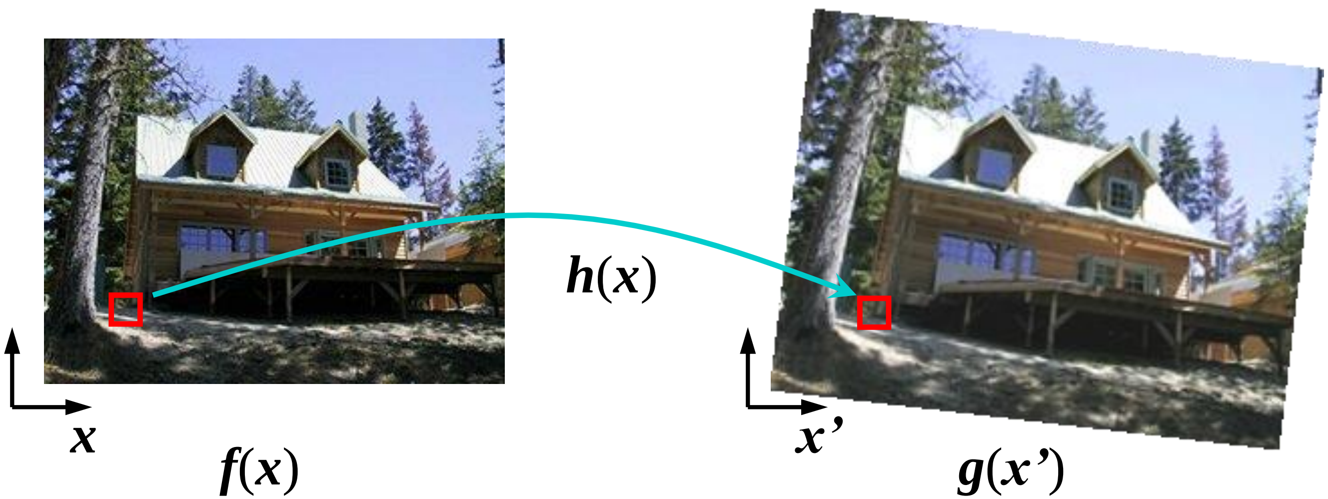
- Given a coordinate transform $\boldsymbol{x}' = \boldsymbol{h}(\boldsymbol{x})$ and a source image $\boldsymbol{f}(\boldsymbol{x})$, how do we compute a transformed image $\boldsymbol{g}(\boldsymbol{x}') = \boldsymbol{f}(\boldsymbol{h}(\boldsymbol{x}))$?



Forward Warping

- Send each pixel $f(x)$ to its corresponding location $x' = h(x)$ in $g(x')$

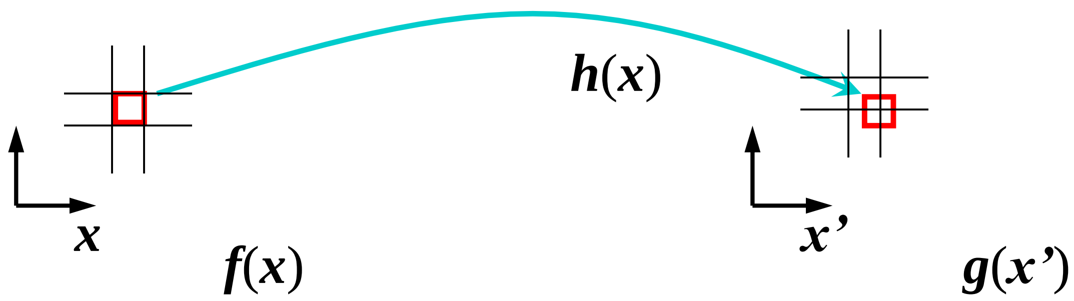
- What if pixel lands “between” two pixels?



Forward Warping

- Send each pixel $\boldsymbol{f}(\boldsymbol{x})$ to its corresponding location $\boldsymbol{x'} = \boldsymbol{h}(\boldsymbol{x})$ in $\boldsymbol{g}(\boldsymbol{x'})$

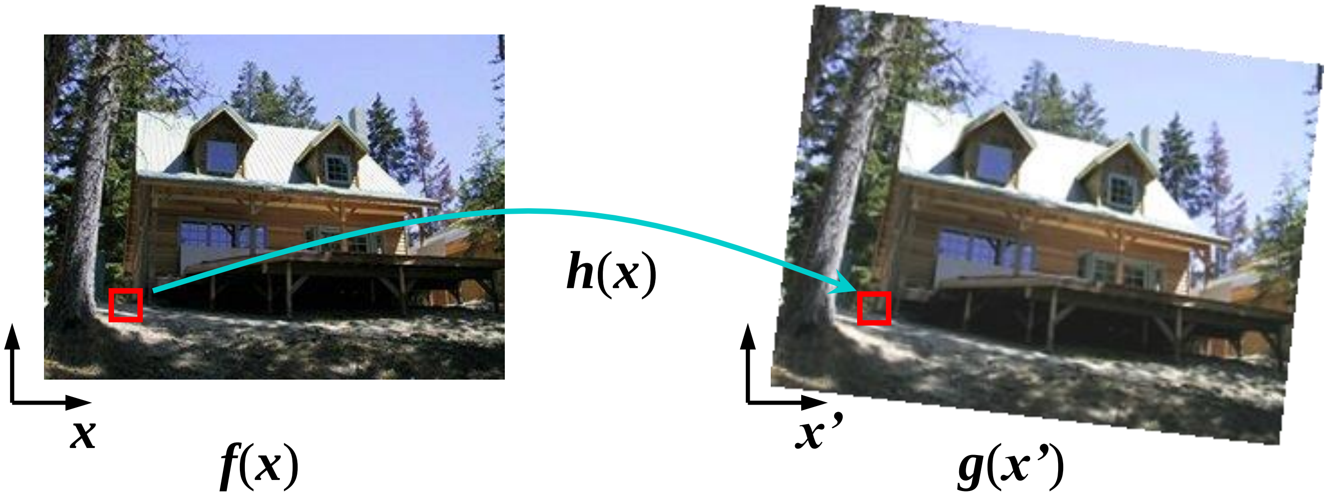
- What if pixel lands "between" two pixels?
- Answer: add "contribution" to several pixels, normalize later (*splatting*)



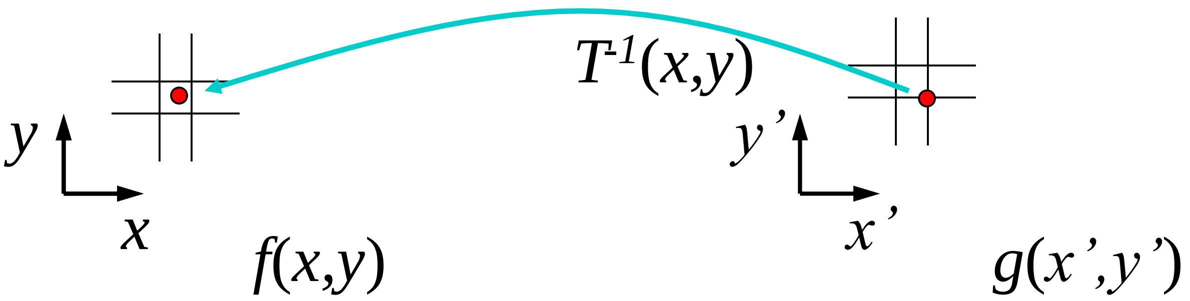
Inverse Warping

- Get each pixel $g(x')$ from its corresponding location $x' = h(x)$ in $f(x)$

- What if pixel comes from “between” two pixels?



Inverse warping



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

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

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

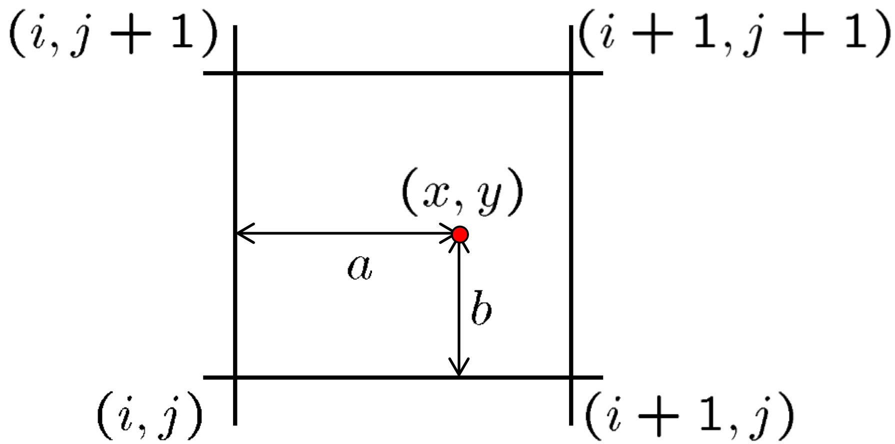
A: *Interpolate* color value from neighbors

- nearest neighbor, bilinear…

Slide from Alyosha Efros, CMU

Bilinear interpolation

Sampling at *f(x,y):*



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

Interpolation

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

2D coordinate transformations

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

  ($\underline{\boldsymbol{x}}$ is a *homogeneous* coordinate)
- These all form a nested *group* (closed w/ inv.)

Homogeneous Coordinates

- consistent representation for all linear transform (including translation)
- can be concatenated & pre-computed

$$(x, y) \rightarrow \qquad (wx, wy, w), w \neq 0$$

$$(wx, wy, w) \rightarrow \qquad (wx / w, wy / w)$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = (TRS) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

2D Affine Transformations

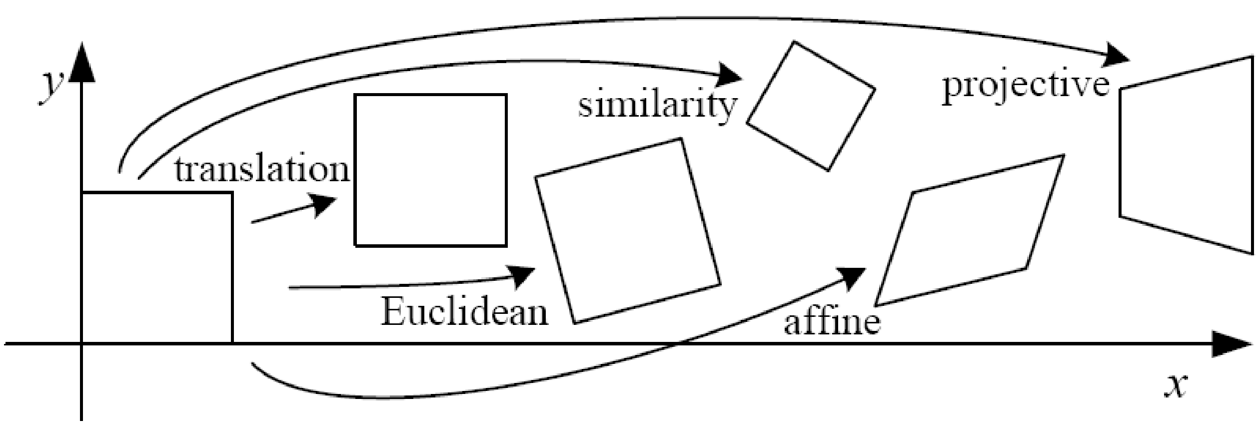
$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Affine transformations are combinations of …
  - Linear transformations, and
  - Translations
- Parallel lines remain parallel

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

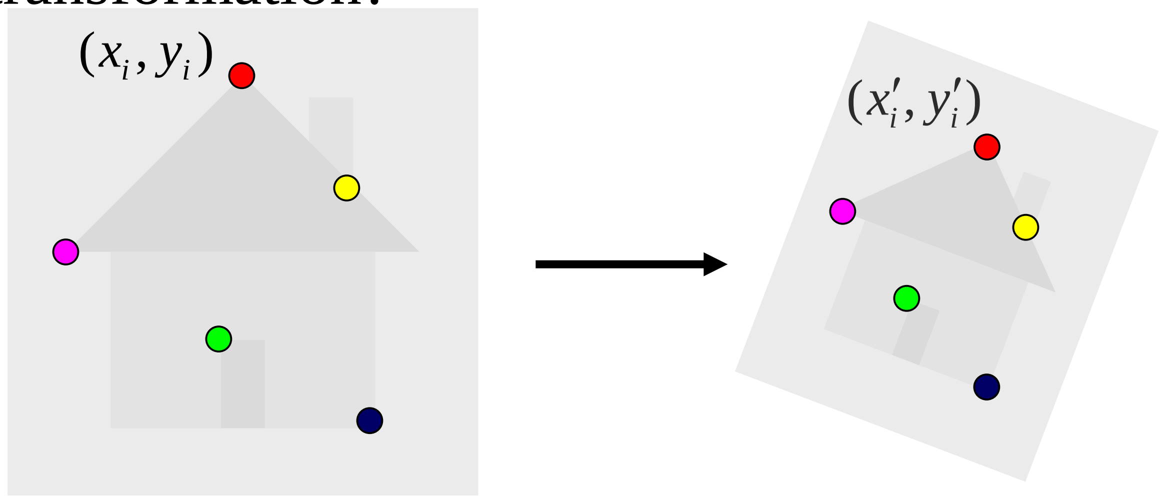


Fitting an affine transformation

Affine model approximates perspective projection of planar objects.

Fitting an affine transformation

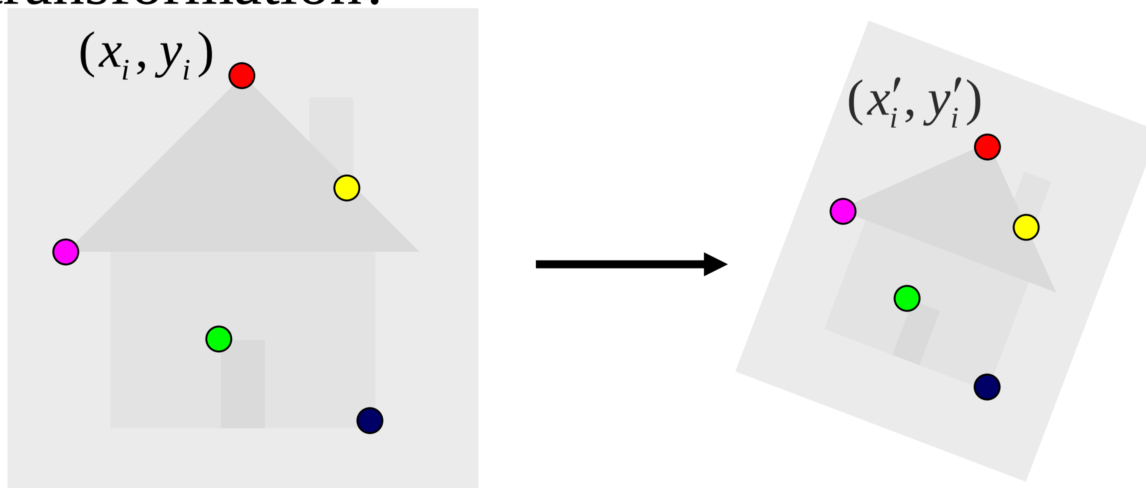
- Assuming we know the correspondences, how do we get the transformation?



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

Fitting an affine transformation

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



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

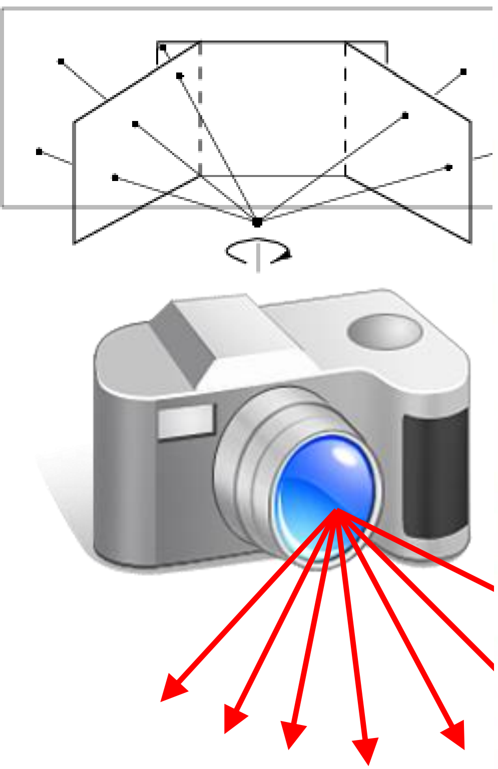
$$\begin{bmatrix} \\ \\ \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} \\ \\ \end{bmatrix}$$

Fitting an affine transformation

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

- 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})

Panoramas



. . .

image from S. Seitz

Obtain a wider angle view by combining multiple images.

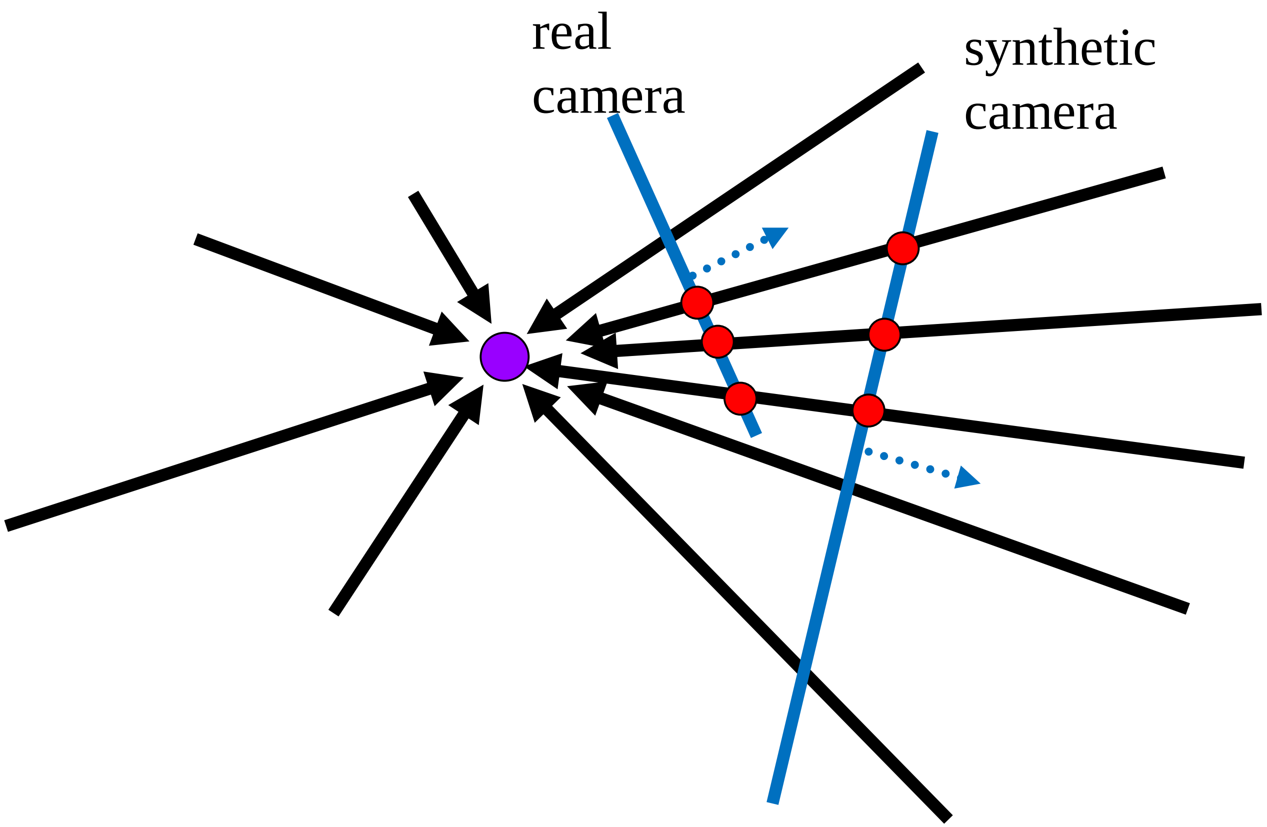
How to stitch together a panorama?

- 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?

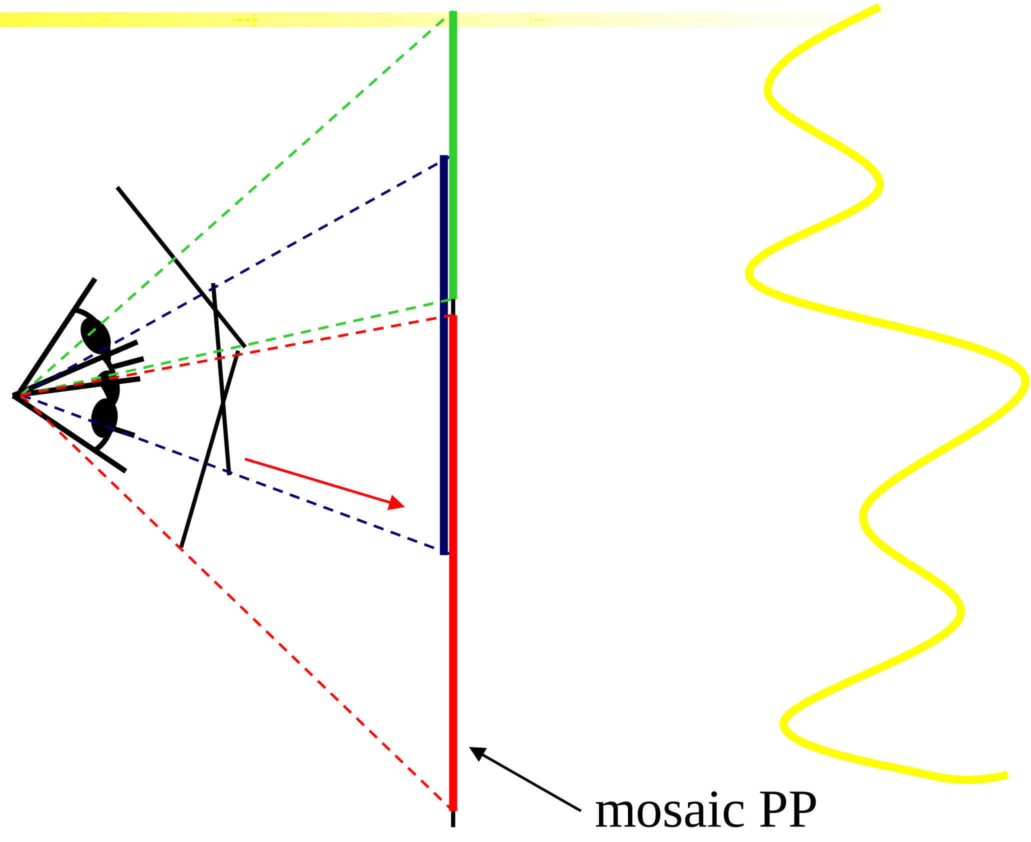
Source: Steve Seitz

Panoramas: generating synthetic views



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

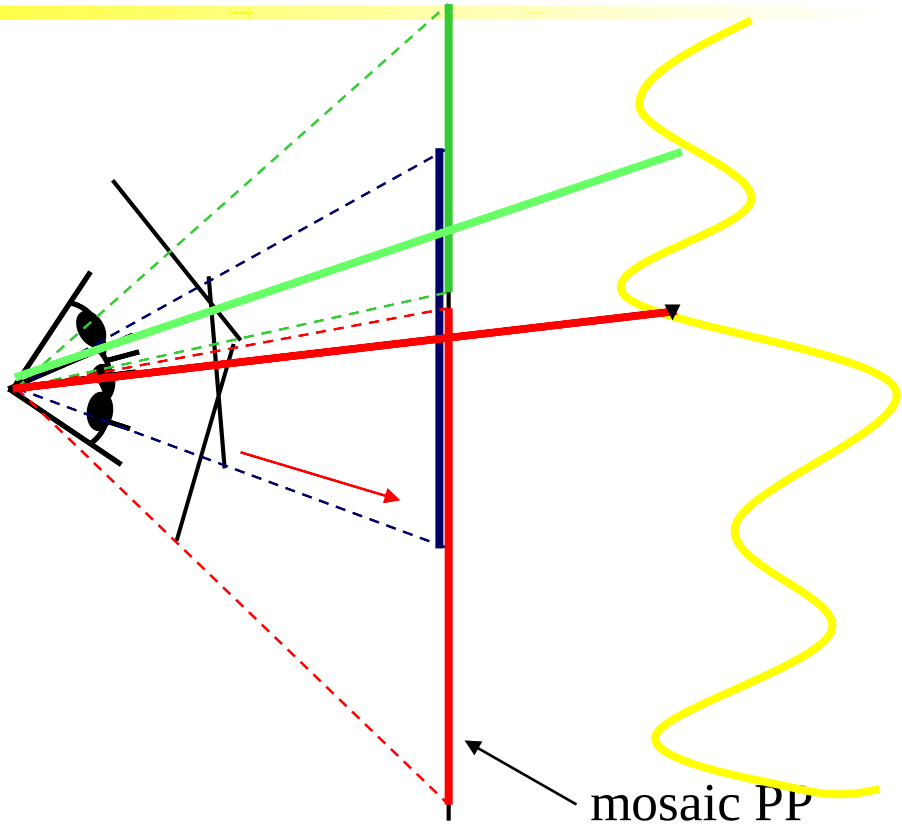
Image reprojection



- 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

Image reprojection



- The mosaic has a natural interpretation in 3D as a plane
- This is true even if the real scene is not planar as long as you have *the same focal point*

Source: Steve Seitz

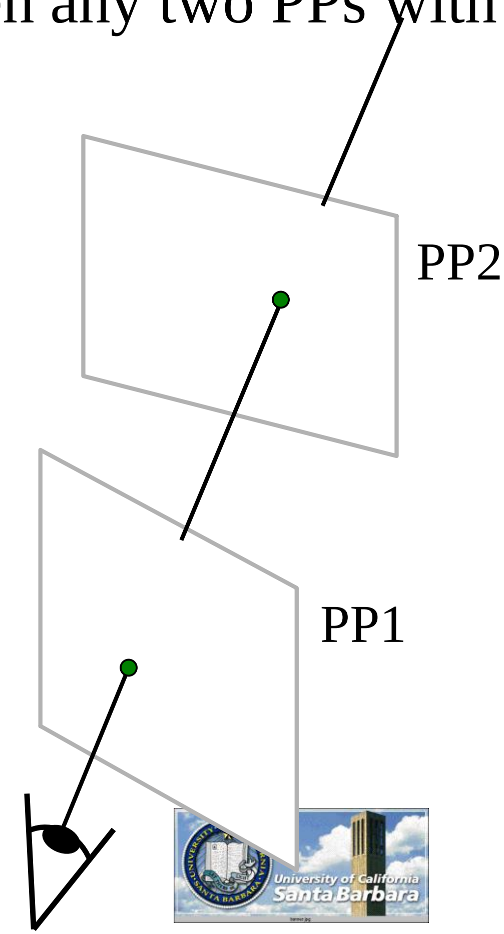
In reality

- The scene is not planar
  - But if you are shooting panorama against far-away objects (e.g., from the south rim of the Grand Canyon against the north rim), the distance variation can be ignored
  - Panorama works best for far-field scene
- The rotation is about the person holding the camera, not the camera's focal center
  - If the scene is far away, such small deviation does not matter
- In fact, image stitching works well if you exercise some caution
- Why all phones these days have the panorama mode

Homography

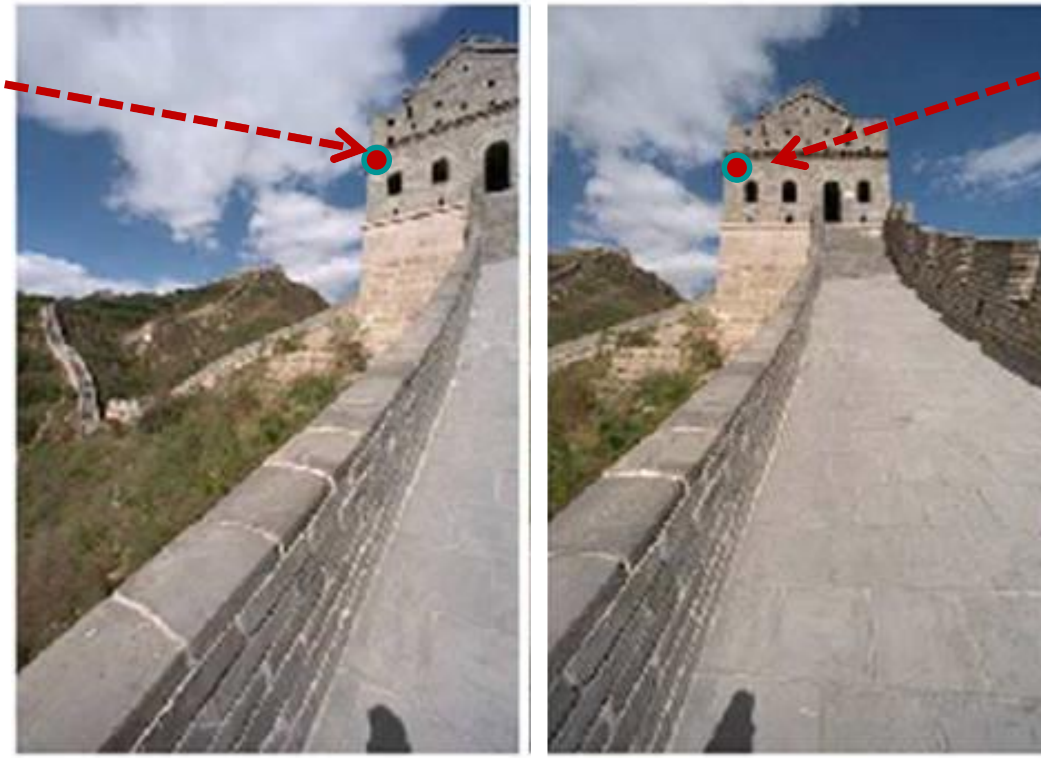
- How to relate two images from the same camera center?
  - how to map a pixel from PP1 to PP2?
- Think of it as a 2D **image warp** from one image to another.
- A projective transform is a mapping between any two PPs with the same center of projection
  - rectangle should map to arbitrary quadrilateral
  - parallel lines aren't
  - but must preserve straight lines
- called **Homography**

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



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

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

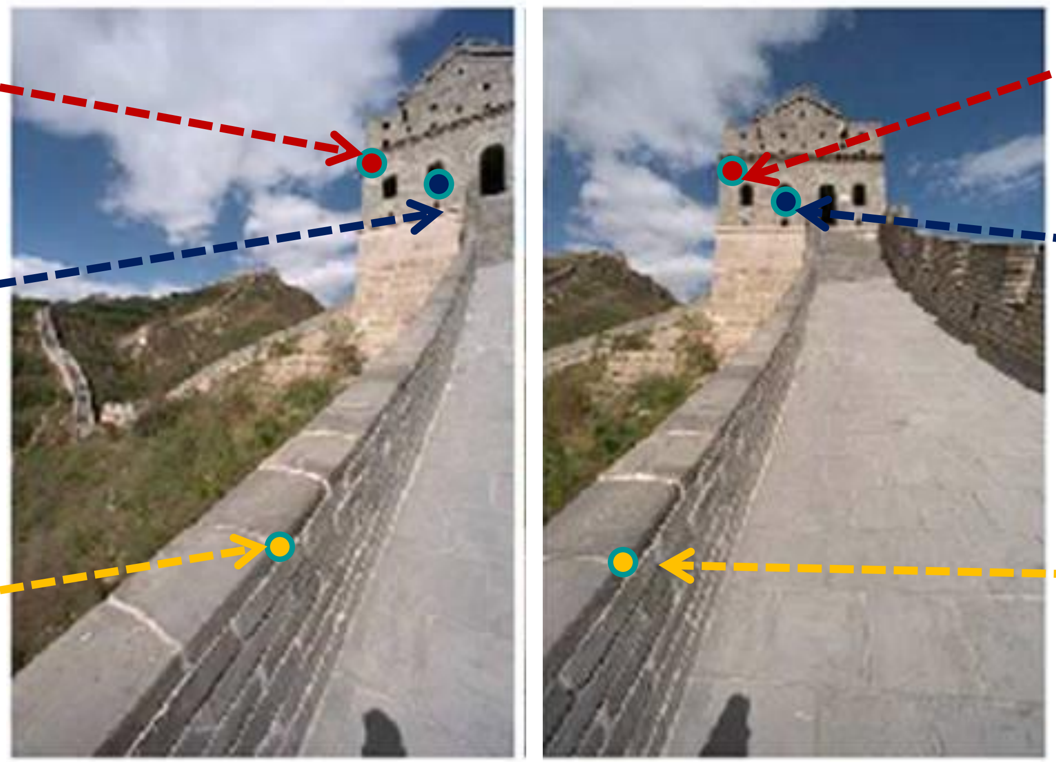
Homography

(x_1, y_1) (x'_1, y'_1)

(x_2, y_2) (x'_2, y'_2)

$\vdots$ $\vdots$

(x_n, y_n) (x'_n, y'_n)



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…

Number of measurements required

- At least as many independent equations as degrees of freedom required
- Example:

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

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2 independent equations / point
8 degrees of freedom

$4 \times 2 \geq 8$

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:

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

- Work well if i is not close to 0 (not recommended!)

Direct Linear Transformation (DLT)

$$H = \begin{bmatrix} h^{1T} \\ h^{2T} \\ h^{3T} \end{bmatrix}$$

$$\mathrm{x}'_i \times \mathrm{H}\mathrm{x}_i = 0 \qquad \mathrm{x}'_i = (x'_i, y'_i, w'_i)^{\mathsf{T}} \qquad \mathrm{H}\mathrm{x}_i = \begin{pmatrix} \mathrm{h}^{1^{\mathsf{T}}}\mathrm{x}_i \\ \mathrm{h}^{2^{\mathsf{T}}}\mathrm{x}_i \\ \mathrm{h}^{3^{\mathsf{T}}}\mathrm{x}_i \end{pmatrix}$$

$$\mathrm{x}'_i \times \mathrm{H}\mathrm{x}_i = \begin{pmatrix} y'_i\mathrm{h}^{3^{\mathsf{T}}}\mathrm{x}_i - w'_i\mathrm{h}^{2^{\mathsf{T}}}\mathrm{x}_i \\ w'_i\mathrm{h}^{1^{\mathsf{T}}}\mathrm{x}_i - x'_i\mathrm{h}^{3^{\mathsf{T}}}\mathrm{x}_i \\ x'_i\mathrm{h}^{2^{\mathsf{T}}}\mathrm{x}_i - y'_i\mathrm{h}^{1^{\mathsf{T}}}\mathrm{x}_i \end{pmatrix}$$

$$\begin{bmatrix} 0^{\mathsf{T}} & -w'_i\mathrm{x}_i^{\mathsf{T}} & y'_i\mathrm{x}_i^{\mathsf{T}} \\ w'_i\mathrm{x}_i^{\mathsf{T}} & 0^{\mathsf{T}} & -x'_i\mathrm{x}_i^{\mathsf{T}} \\ -y'_i\mathrm{x}_i^{\mathsf{T}} & x'_i\mathrm{x}_i^{\mathsf{T}} & 0^{\mathsf{T}} \end{bmatrix} \begin{pmatrix} \mathrm{h}^1 \\ \mathrm{h}^2 \\ \mathrm{h}^3 \end{pmatrix} = 0$$

$$\mathrm{A}_i\mathrm{h} = 0$$

Direct Linear Transformation (DLT)

- Equations are linear in h

$$A_i \mathrm{h} = 0$$

- Only 2 out of 3 are linearly independent (indeed, 2 eq/pt)

$$\begin{bmatrix} 0^\top & -w_i'\mathrm{x}_i^\top & y_i'\mathrm{x}_i^\top \\ w_i'\mathrm{x}_i^\top & 0^\top & -x_i'\mathrm{x}_i^\top \\ -y_i'\mathrm{x}_i^\top & x_i'\mathrm{x}_i^\top & 0^\top \end{bmatrix} \begin{pmatrix} \mathrm{h}^1 \\ \mathrm{h}^2 \\ \mathrm{h}^3 \end{pmatrix} = 0$$

$$\begin{bmatrix} 0^\top & -w_i'\mathrm{x}_i^\top & y_i'\mathrm{x}_i^\top \\ w_i'\mathrm{x}_i^\top & 0^\top & -x_i'\mathrm{x}_i^\top \end{bmatrix} \begin{pmatrix} \mathrm{h}^1 \\ \mathrm{h}^2 \\ \mathrm{h}^3 \end{pmatrix} = 0$$

$$x_i' A_i^1 + y_i' A_i^2 + w_i' A_i^3 = 0$$

(only drop third row if $w_i' \neq 0$)

- Holds for any homogeneous representation, e.g. $(x_i', y_i', 1)$

Direct Linear Transformation (DLT)

- Solving for H

$$\begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix} \mathbf{h} = 0$$

size A is 8x9 or 12x9, but rank 8

Trivial solution is $h=0_9^T$ is not interesting

1-D null-space yields solution of interest

pick for example the one with $\|h\| = 1$

Direct Linear Transformation (DLT)

- Over-determined solution

$$\begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_n \end{bmatrix} \mathbf{h} = \mathbf{0}$$

No exact solution because of inexact measurement i.e. "noise"

Find approximate solution

- Additional constraint needed to avoid 0, e.g. $\|\mathrm{h}\| = 1$
- $\mathrm{Ah} = 0$ not possible, so minimize $\|\mathrm{Ah}\|$

DLT algorithm

Objective

Given $n \geq 4$ 2D to 2D point correspondences $\{x_i \leftrightarrow x_i'\}$, determine the 2D homography matrix H such that $x_i' = Hx_i$

Algorithm

(i) For each correspondence $x_i \leftrightarrow x_i'$ compute A_i. Usually only two first rows needed.

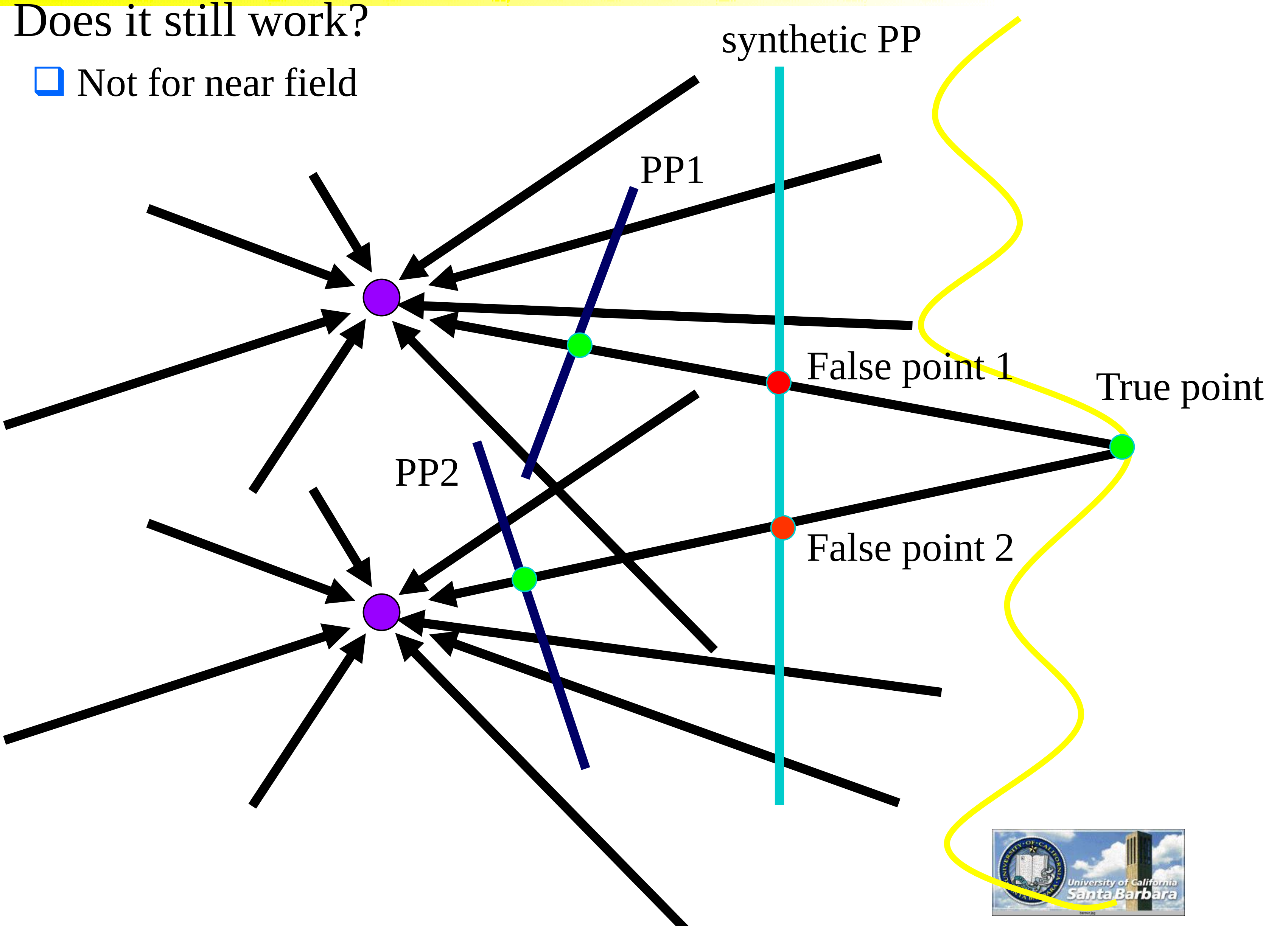
(ii) Assemble n 2x9 matrices A_i into a single $2n$x9 matrix A

(iii) Obtain SVD of A. Solution for h is last column of V

(iv) Determine H from h

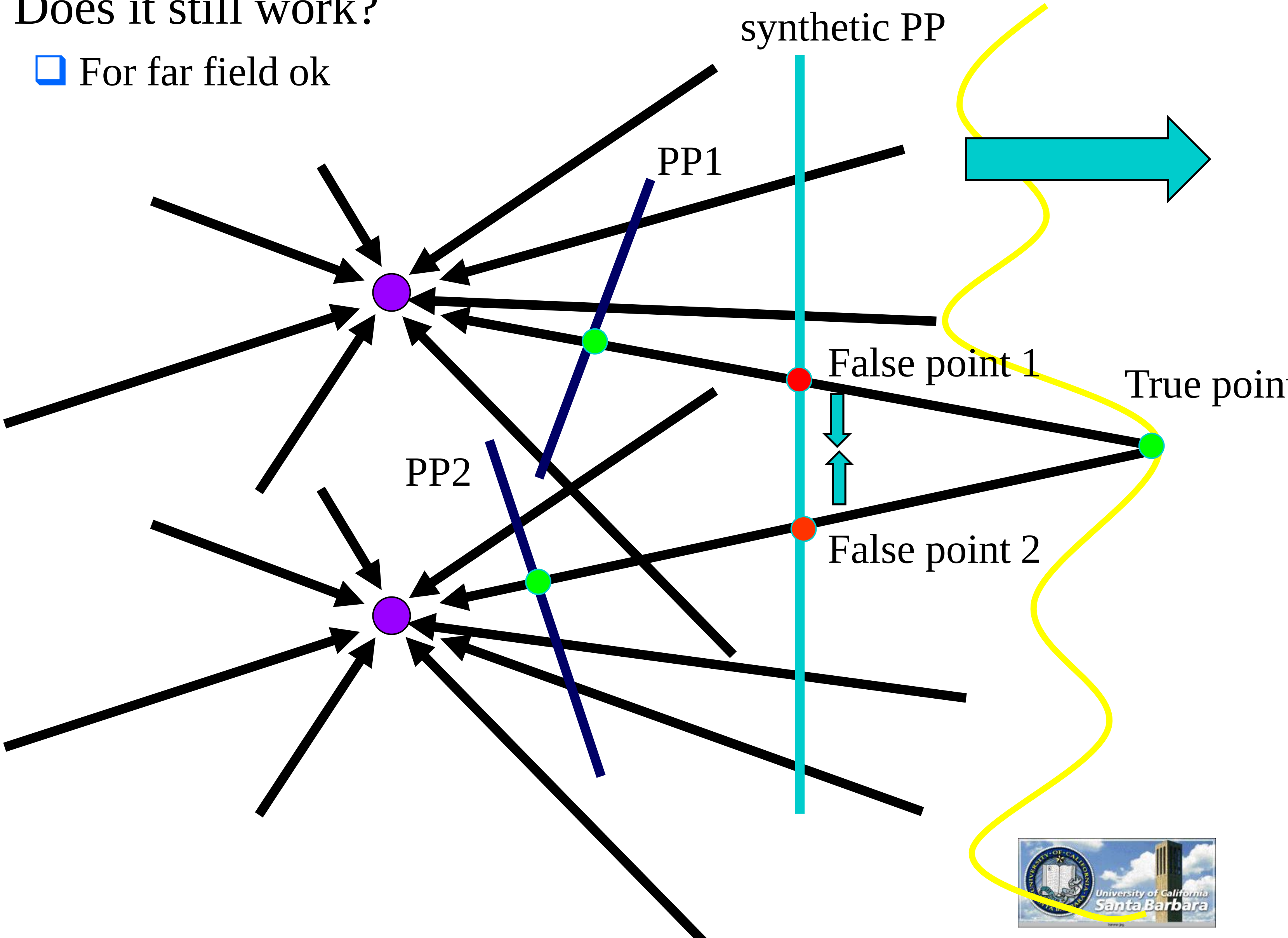
changing camera center

- Does it still work?
  - Not for near field

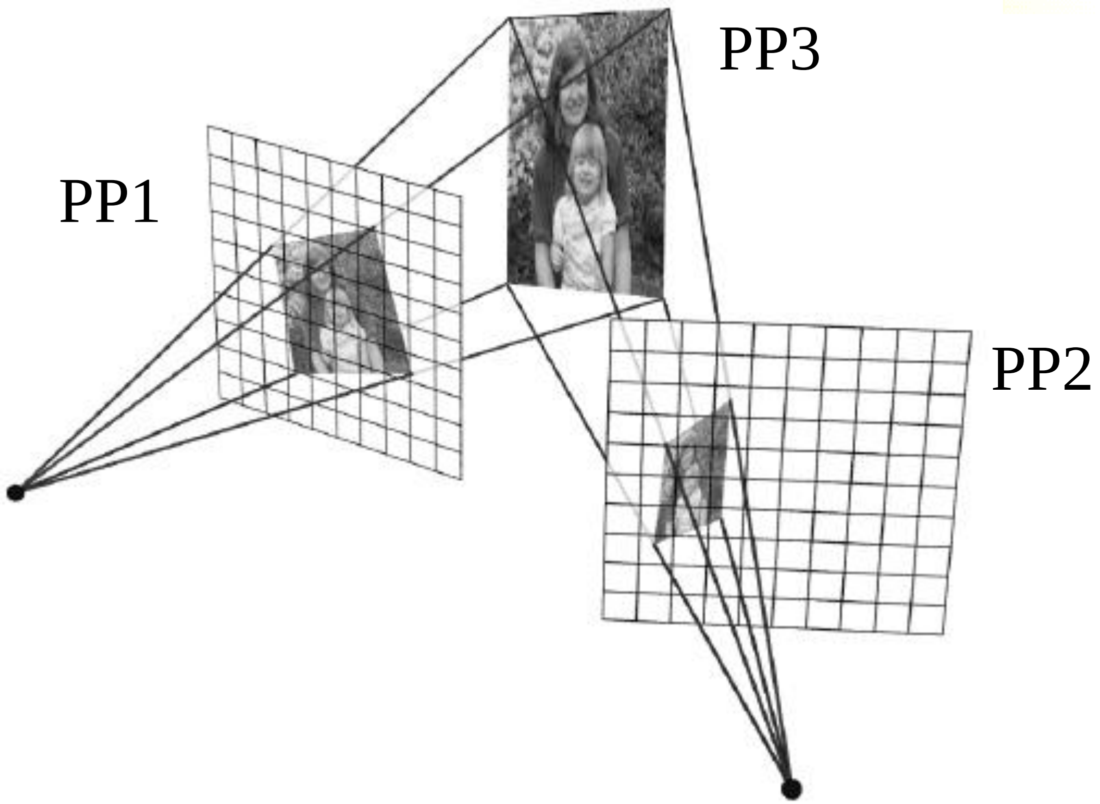


changing camera center

- Does it still work?
  - For far field ok



Planar scene (or far away)



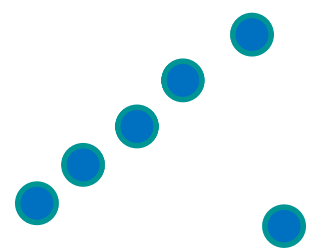
- PP3 is a projection plane of both centers of projection, so we are OK!
- This is how big aerial photographs are made

Source: Alyosha Efros


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

Outliers

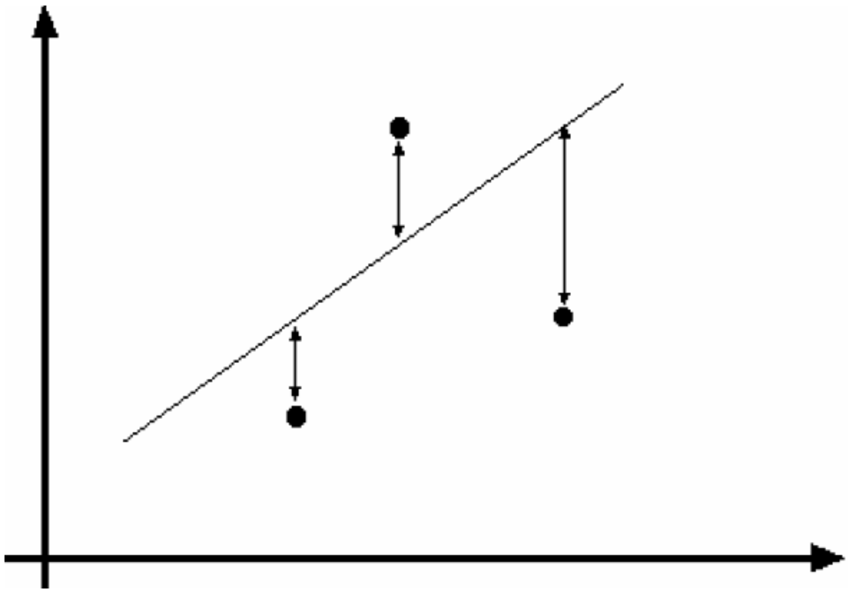
- **Outliers** can hurt the quality of our parameter estimates, e.g.,
  - an erroneous pair of matching points from two images
  - an edge point that is noise, or doesn't belong to the line we are fitting.



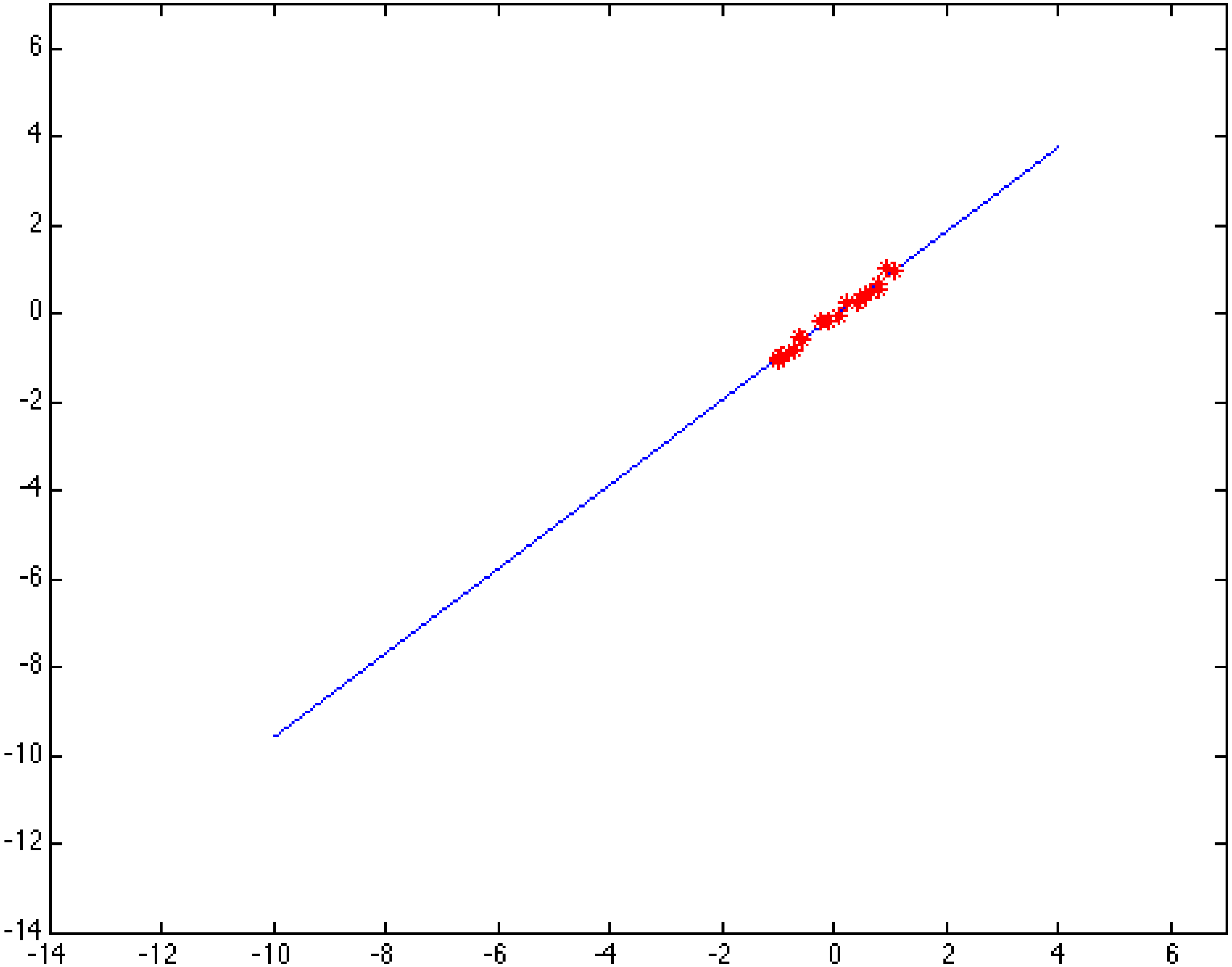
Grauman

Example: least squares line fitting

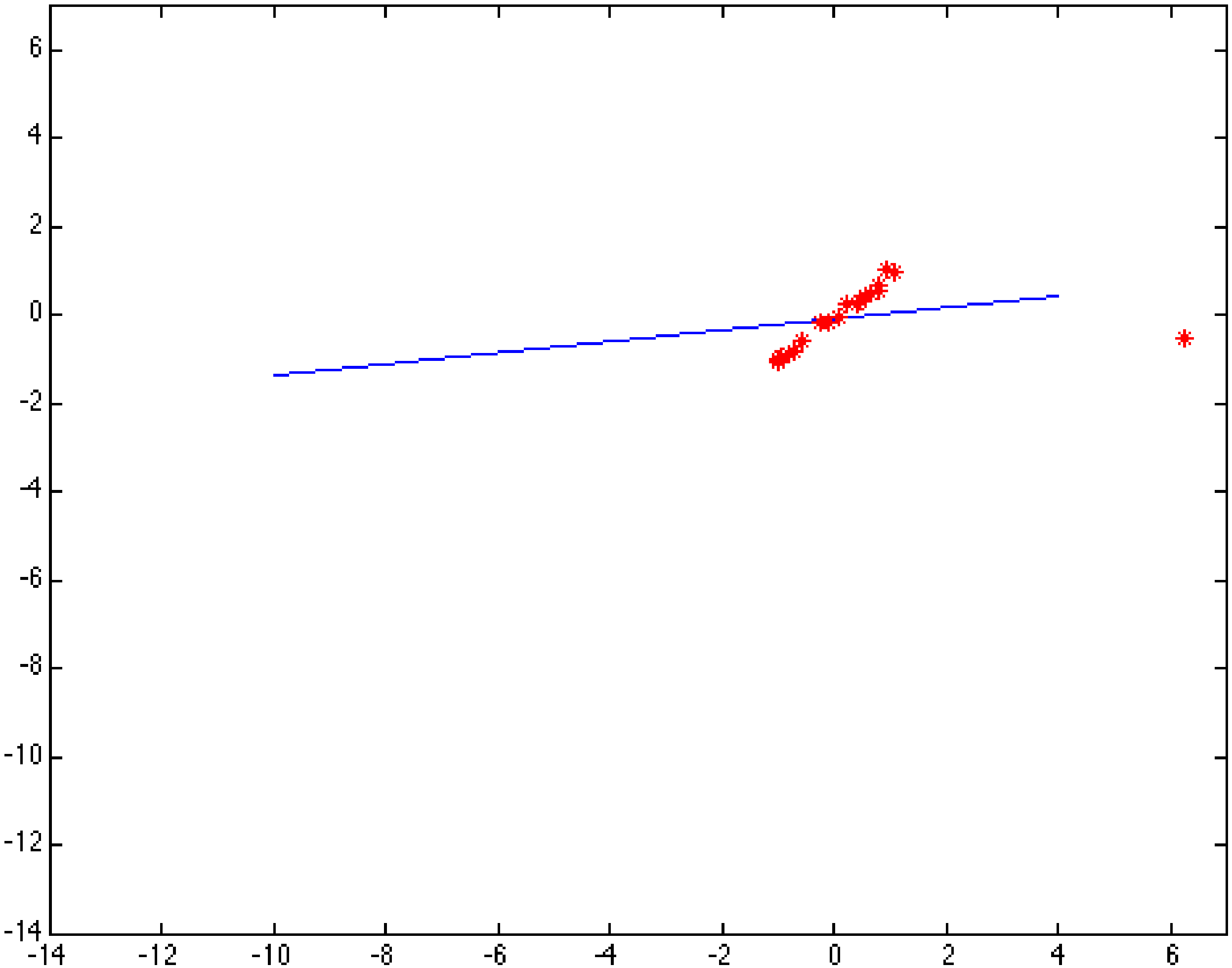
- Assuming all the points that belong to a particular line are known



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.

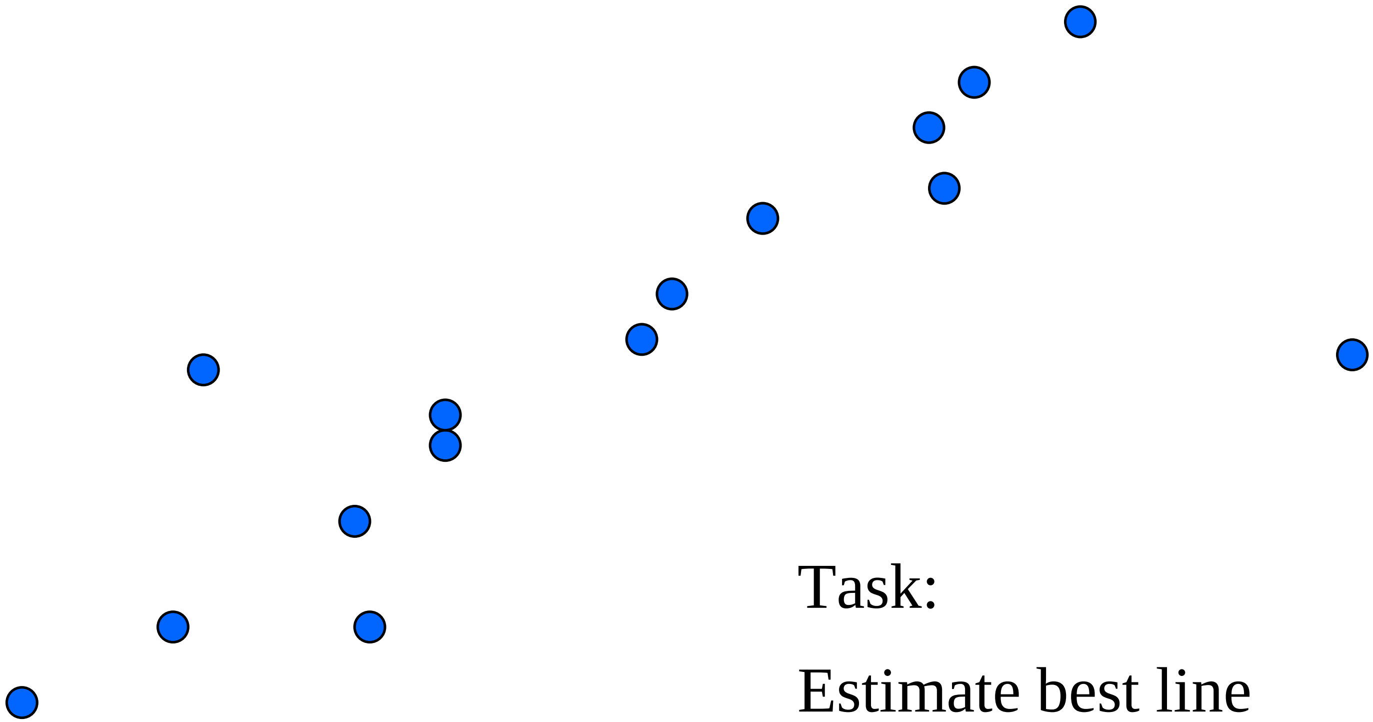
RANSAC

- <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

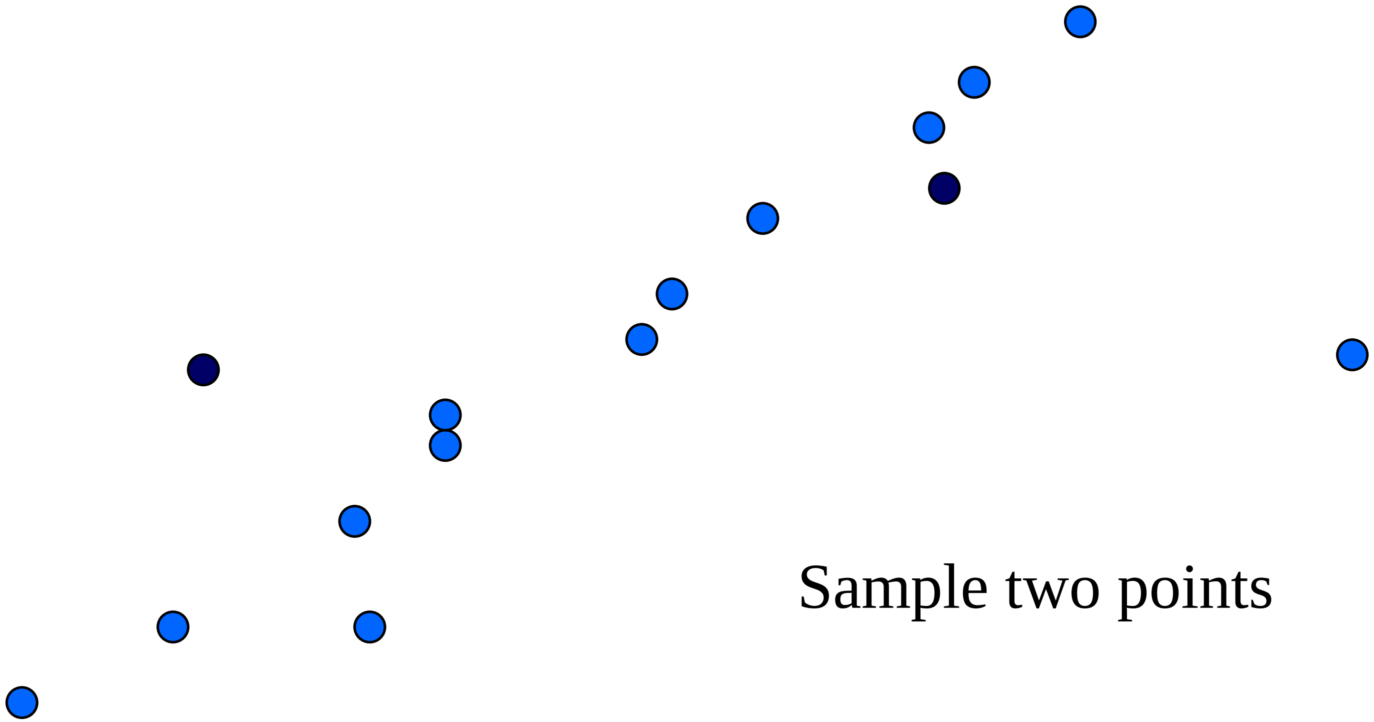


Task:

Estimate best line

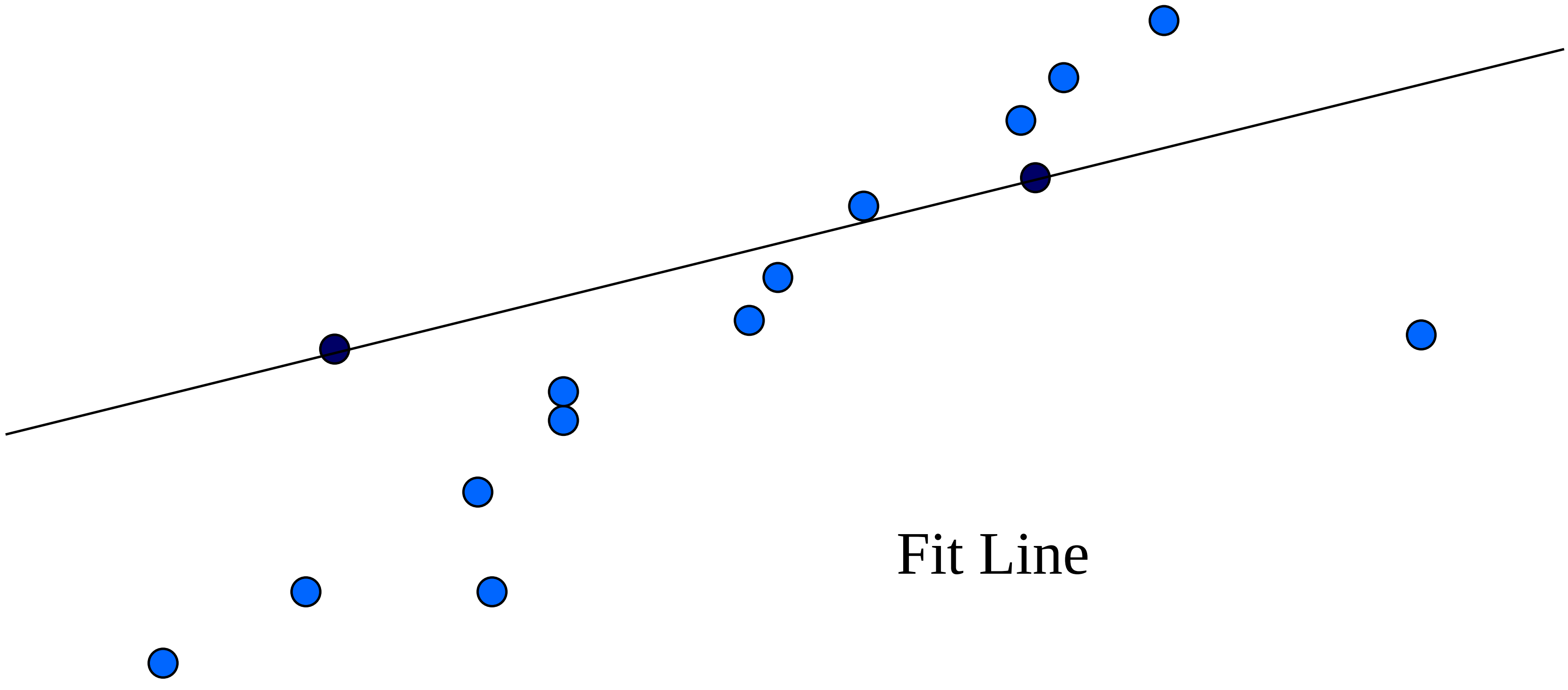
Slide credit: Jinxiang Chai, CMU

RANSAC Line Fitting Example

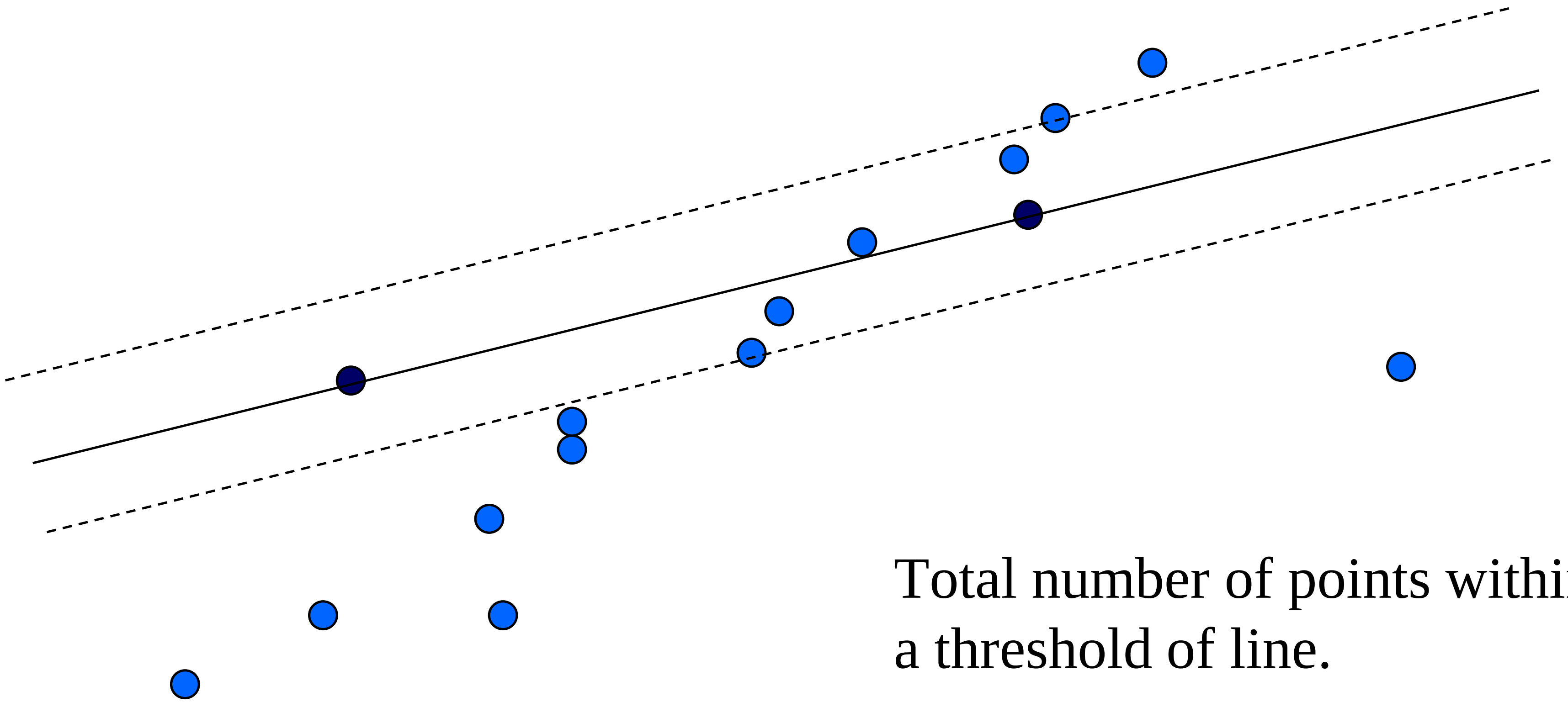


Sample two points

RANSAC Line Fitting Example

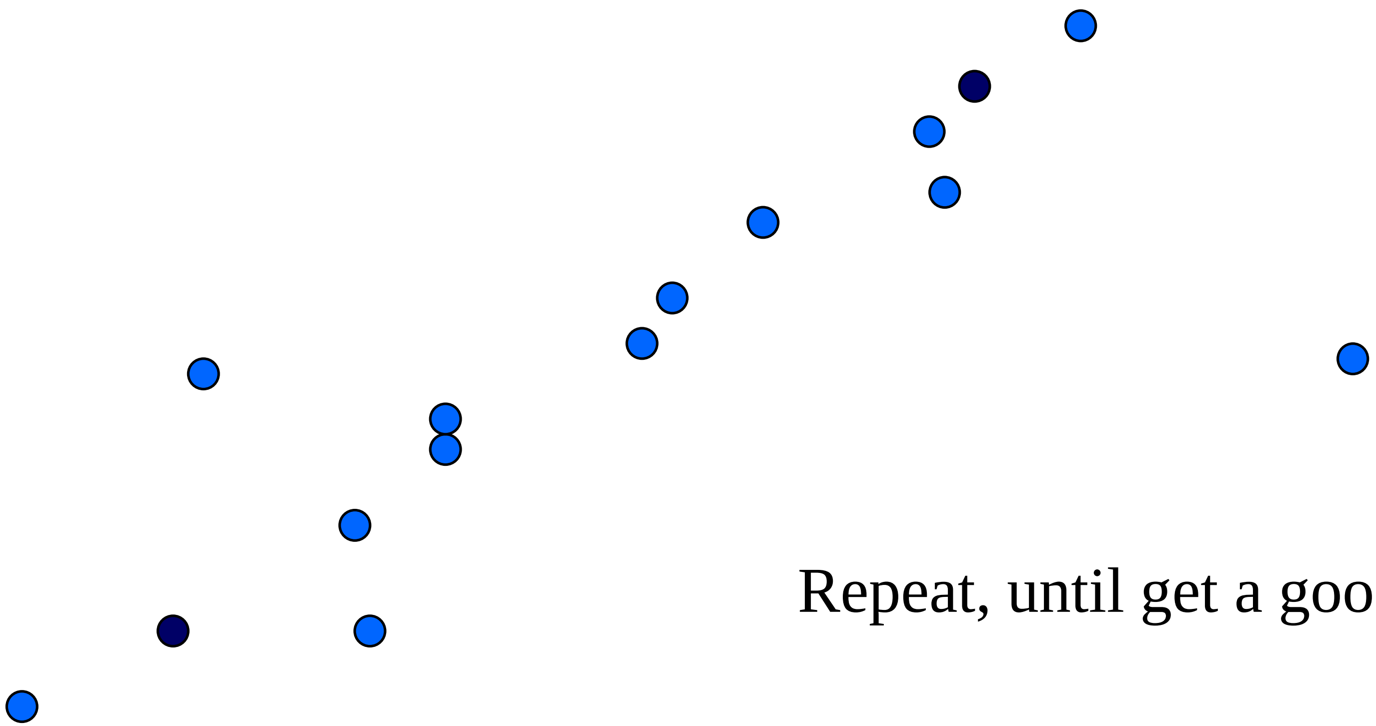


RANSAC Line Fitting Example



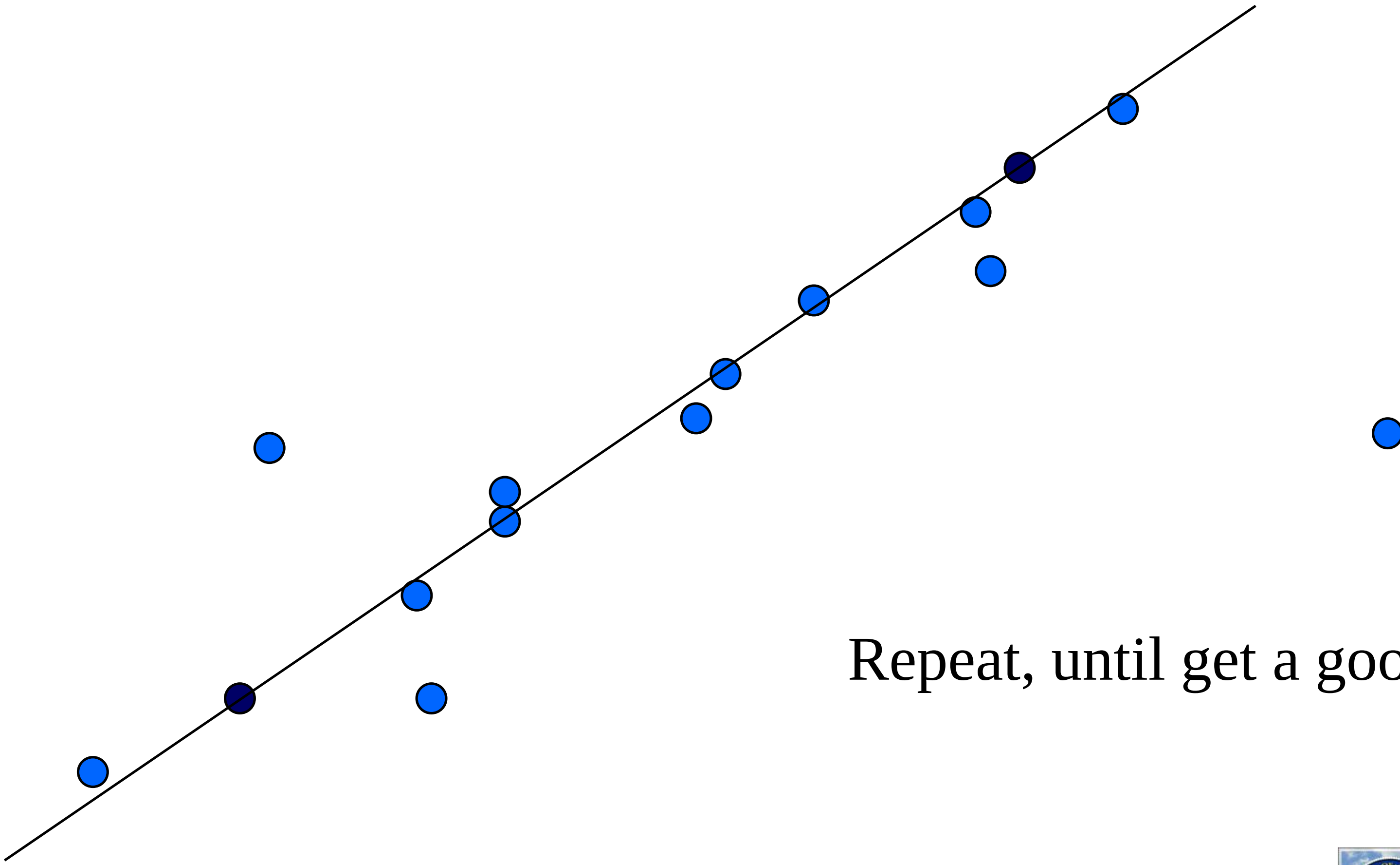
Total number of points within a threshold of line.

RANSAC Line Fitting Example



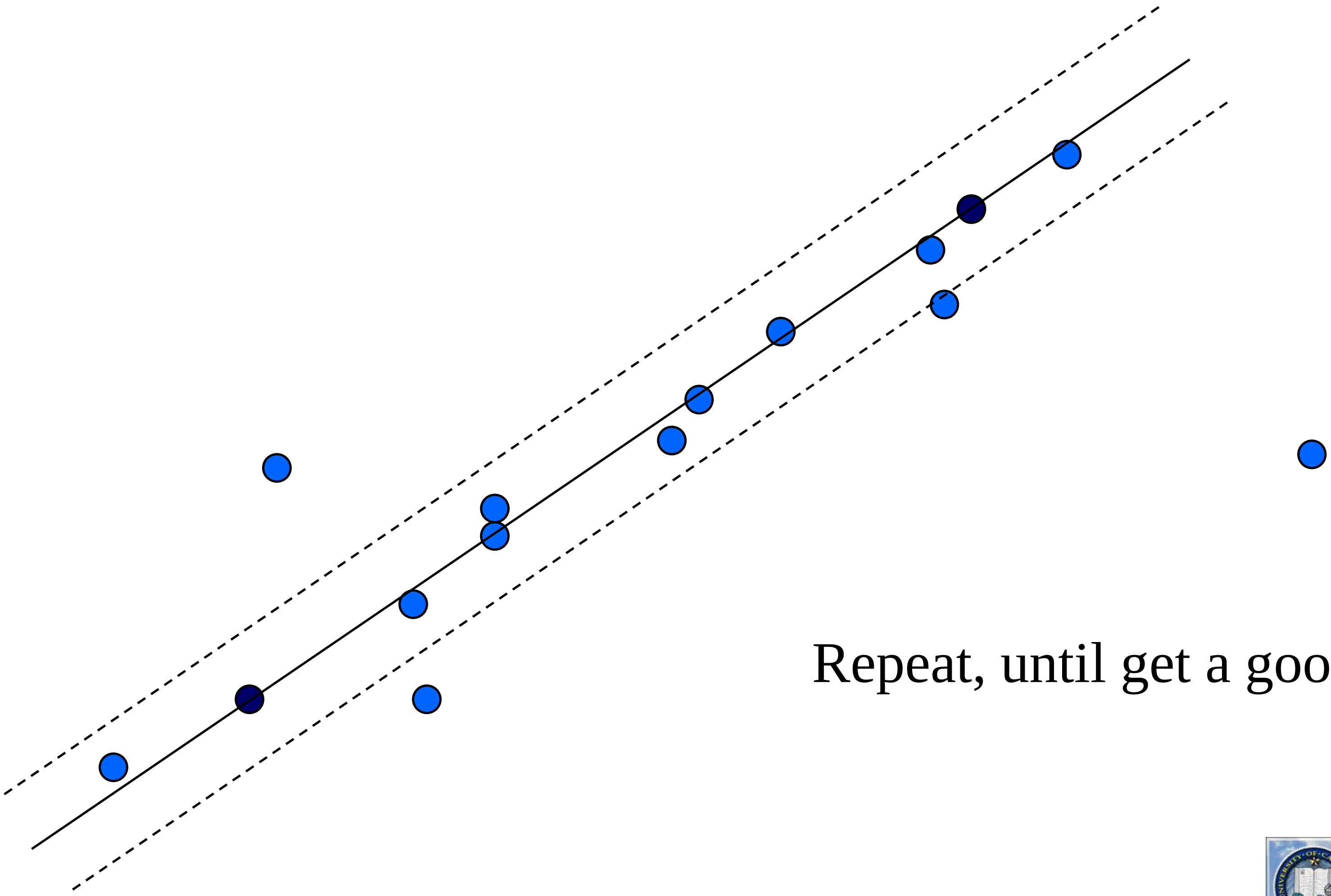
Repeat, until get a good result

RANSAC Line Fitting Example



Repeat, until get a good result

RANSAC Line Fitting Example



Repeat, until get a good result

How Many Trials?

- Well, theoretically it is *C(n,p)* to find all possible *p*-tuples
- Very expensive

$$1 - (1 - (1 - \varepsilon)^p)^m$$

ε : fraction of bad data

$(1 - \varepsilon)$: fraction of good data

$(1 - \varepsilon)^p$: all p samples are good

$1 - (1 - \varepsilon)^p$: at least one sample is bad

$(1 - (1 - \varepsilon)^p)^m$: got bad data in all m tries

$1 - (1 - (1 - \varepsilon)^p)^m$: got at least one good p set in m tries

How Many Trials (cont.)

- Make sure the probability is high (e.g. >95%)
- given p and epsilon, calculate m

p	5%	10 %	20 %	25 %	30 %	40 %	50 %
1	1	2	2	3	3	4	5
2	2	2	3	4	5	7	11
3	2	3	5	6	8	13	23
4	2	3	6	8	11	22	47
5	3	4	8	12	17	38	95

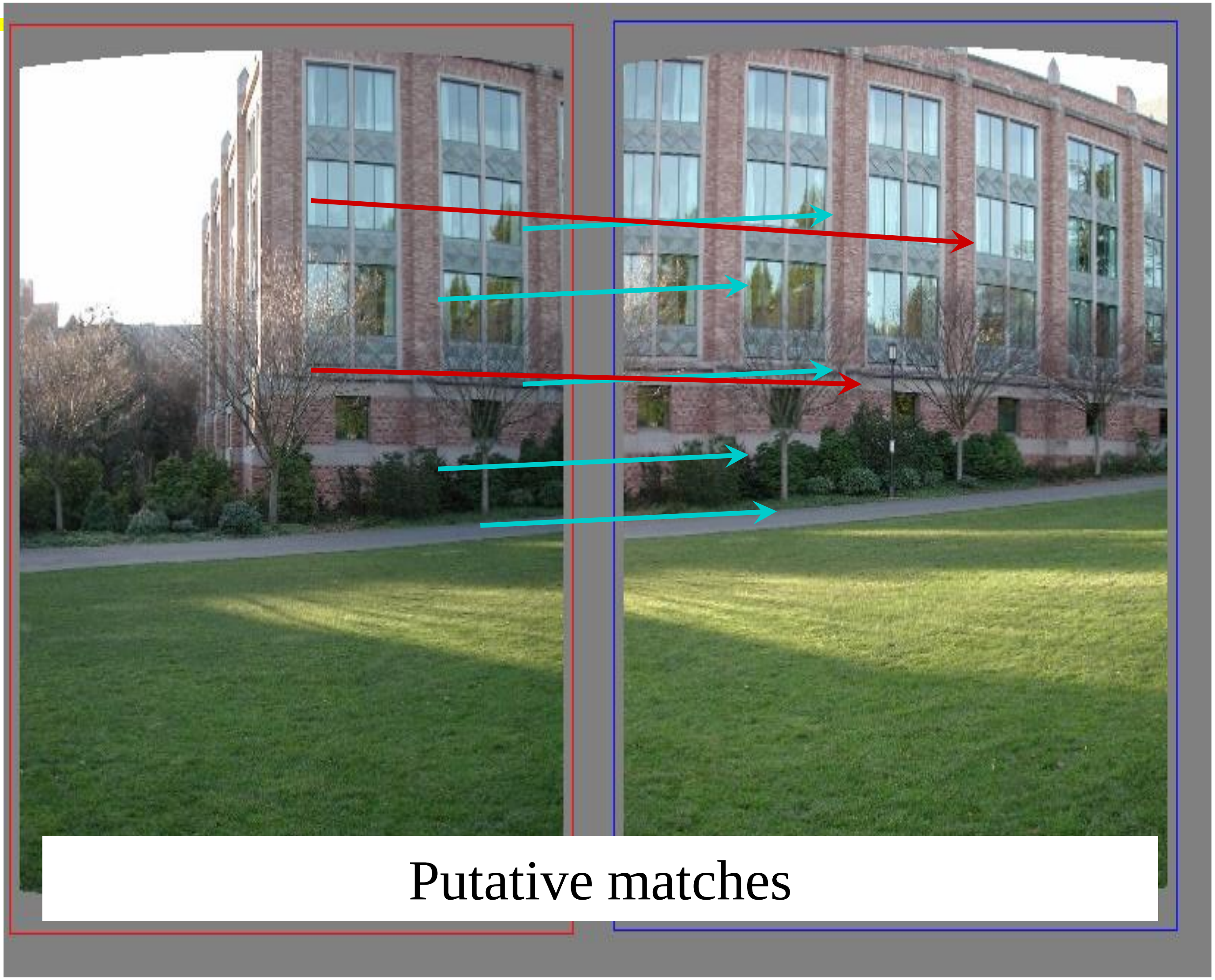
Best Practice

- Randomized selection can completely remove outliers
- "plutocratic"
- Results are based on a small set of features

- LS is most fair, everyone get an equal say
- "democratic"
- But can be seriously influenced by bad data

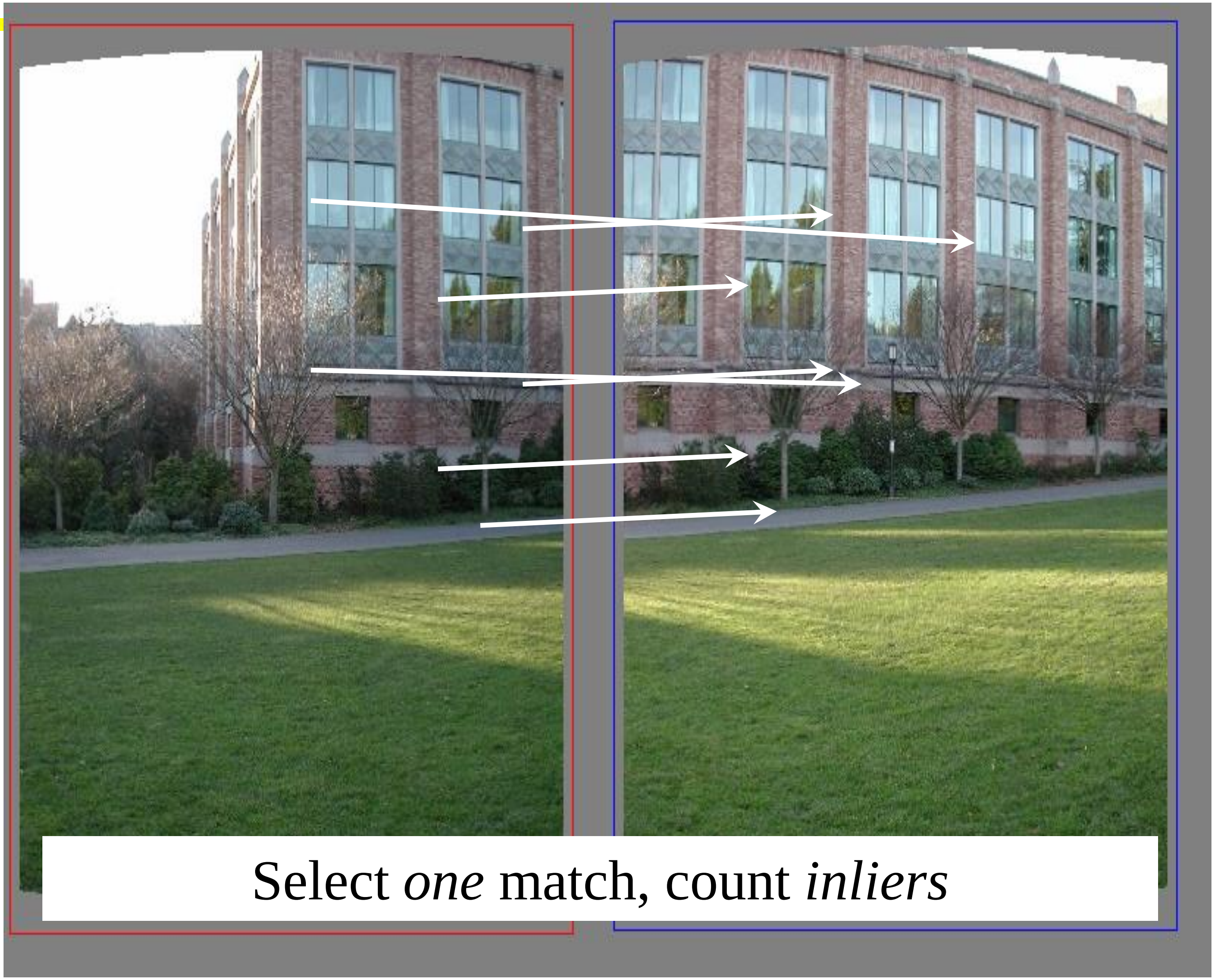
- Use randomized algorithm to remove outliers
- Use LS for final "polishing" of results (using all "good" data)
- Allow up to 50% outliers theoretically

RANSAC example: Translation



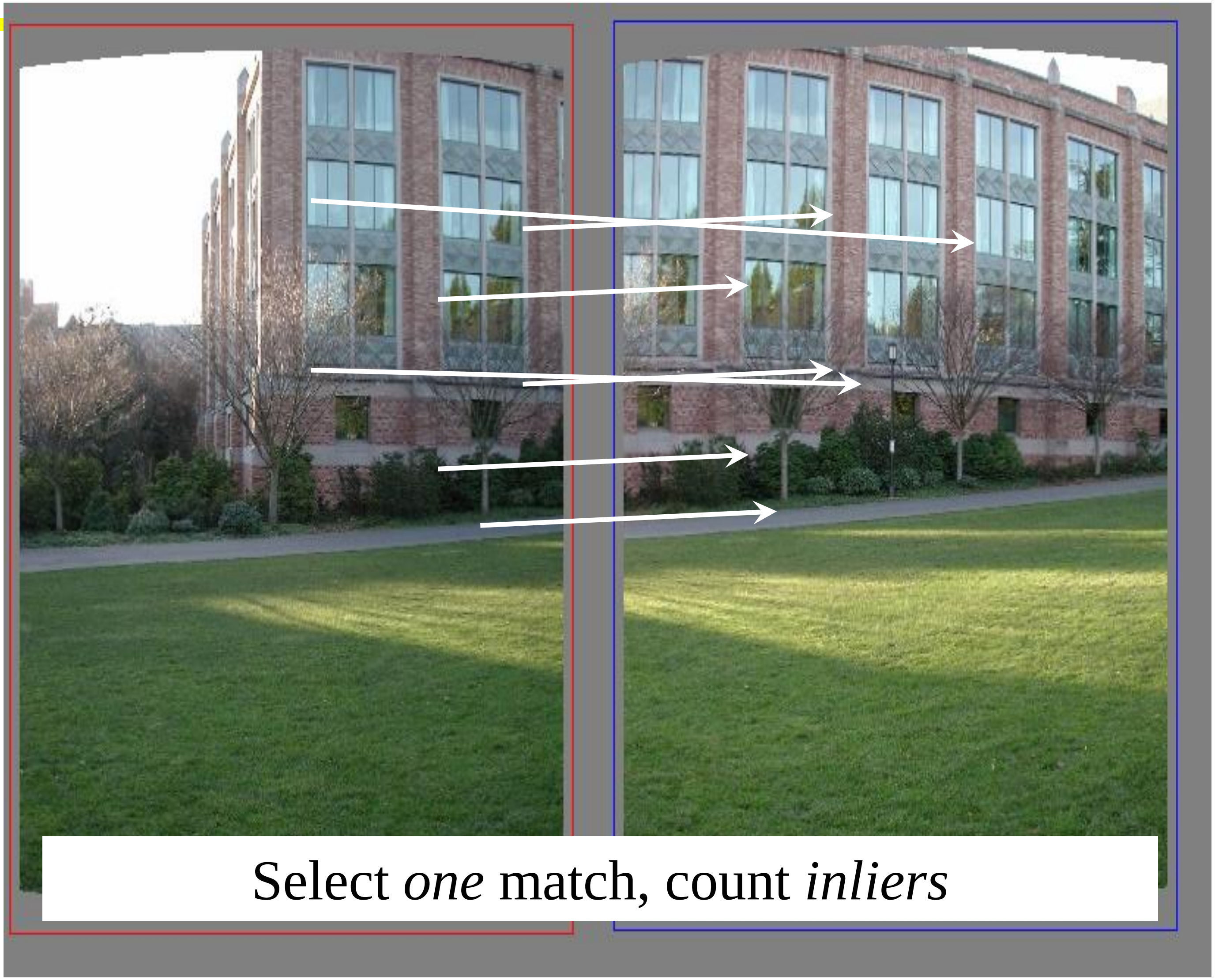
Source: Rick Szeliski

RANSAC example: Translation



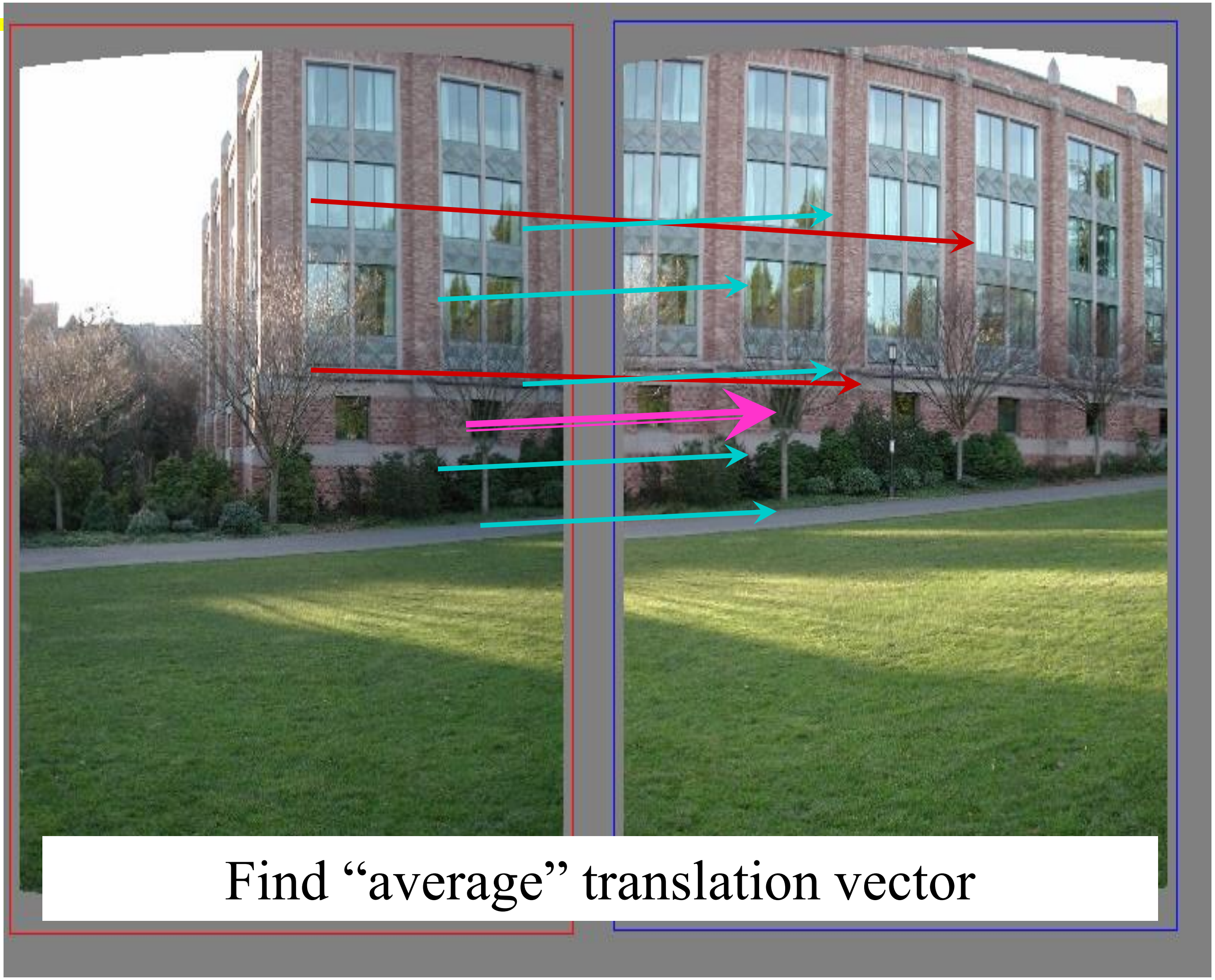
Select *one* match, count *inliers*

RANSAC example: Translation



Select *one* match, count *inliers*

RANSAC example: Translation

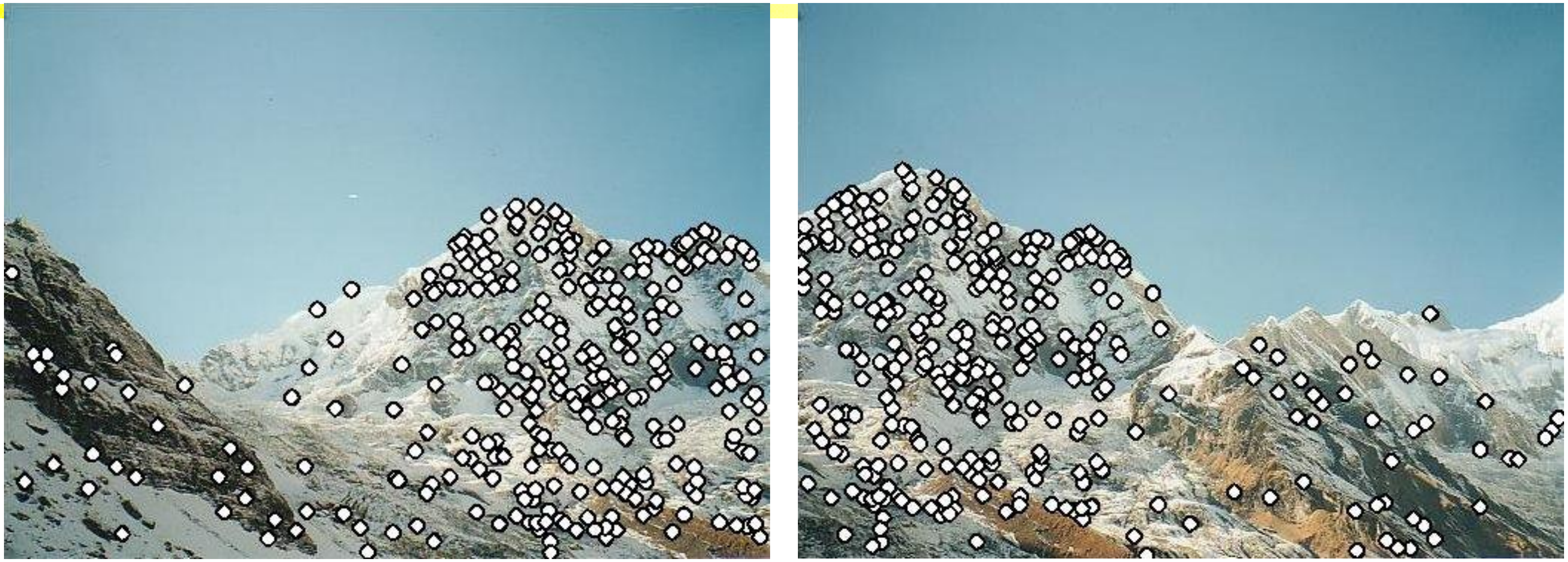


Find “average” translation vector

Feature-based alignment outline


Source: L. Lazebnik

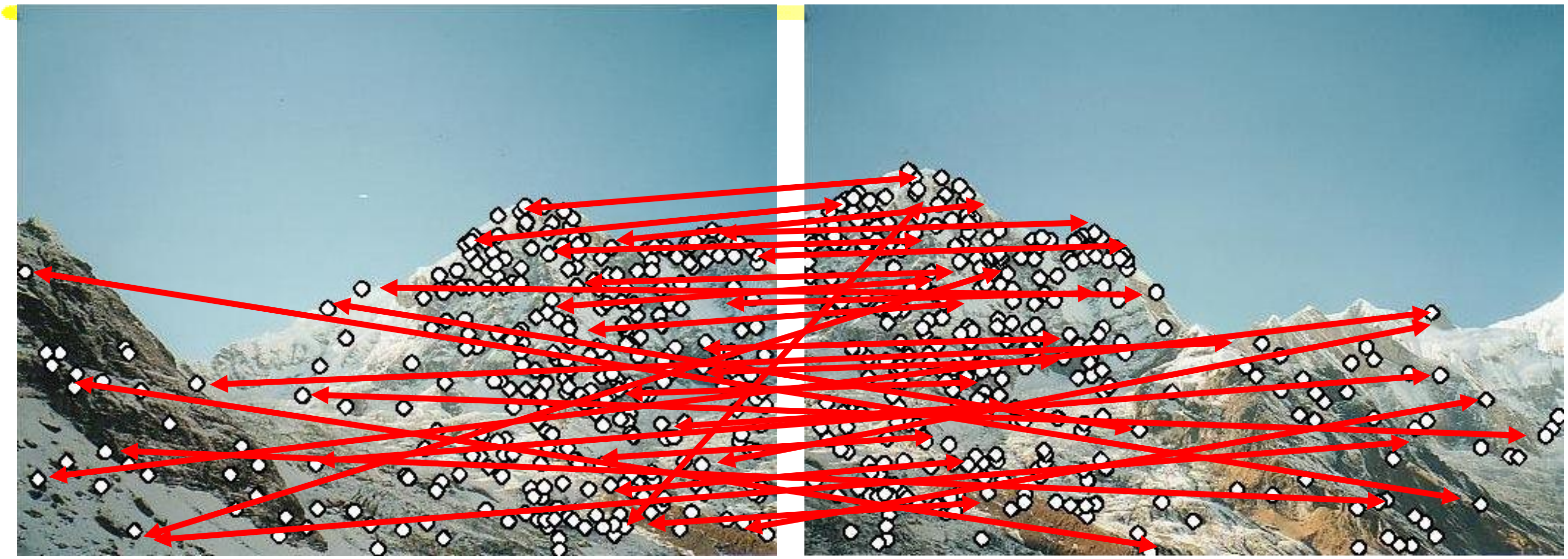
Feature-based alignment outline



- Extract features

Source: L. Lazebnik

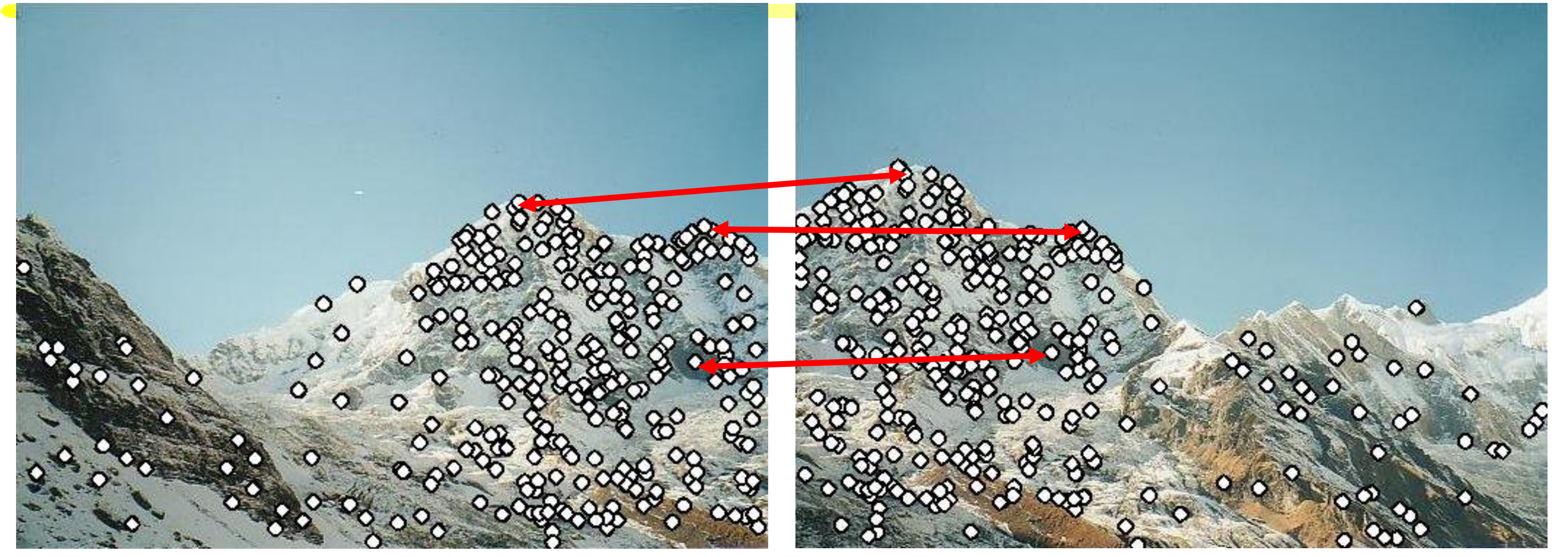
Feature-based alignment outline



- Extract features
- Compute *putative matches*

Source: L. Lazebnik

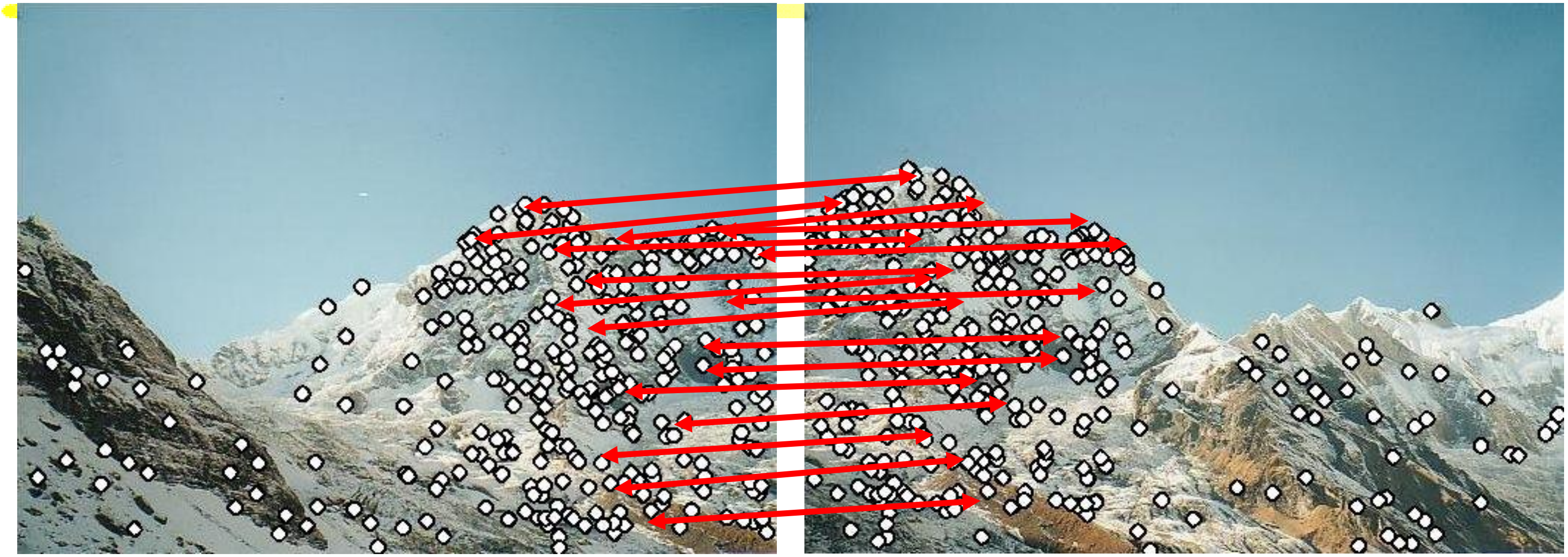
Feature-based alignment outline



- 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

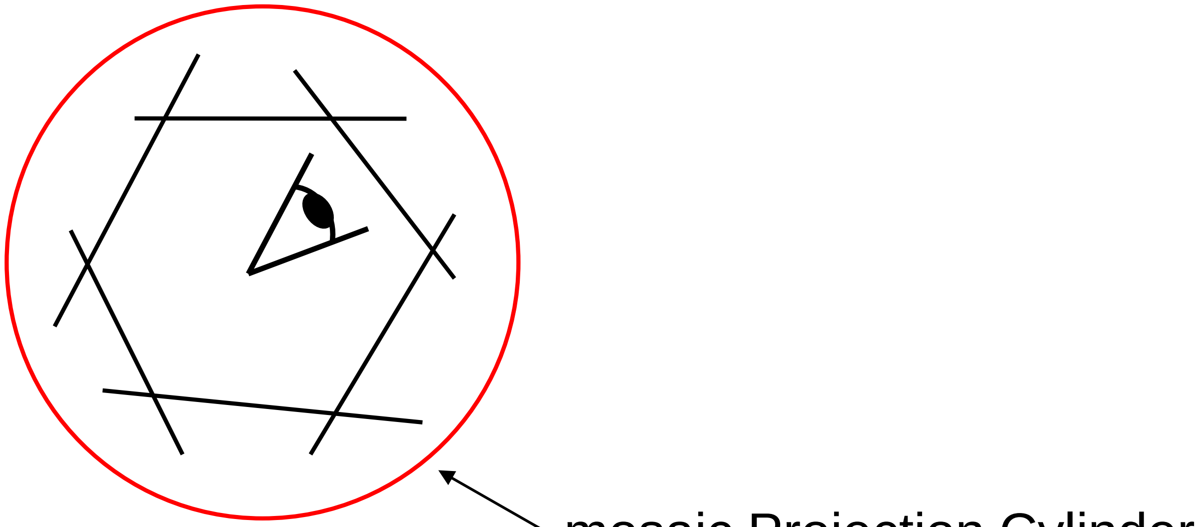
Feature-based alignment outline

- 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*)

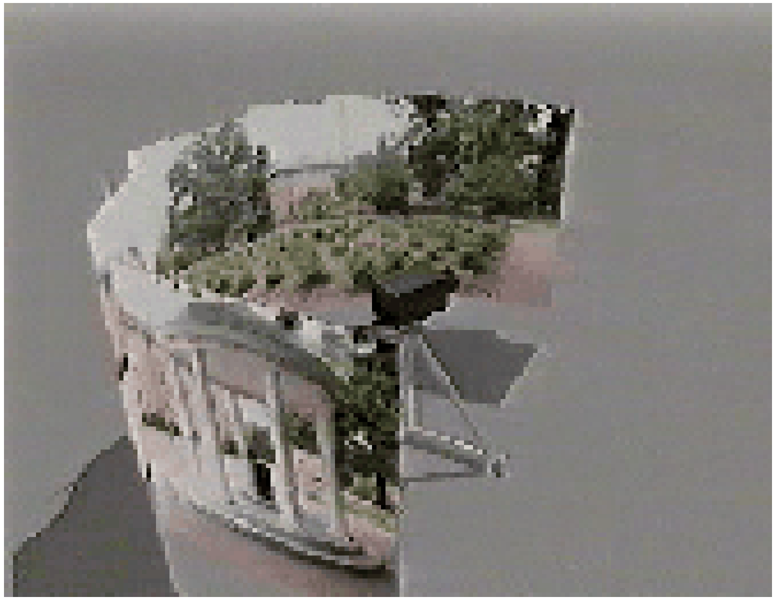
Source: L. Lazebnik

Panoramas

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



Cylindrical panoramas



- Steps
  - Project each image onto a cylinder (warp)
  - Estimate motion (a pure translation now)
  - Blend
  - Optional: project it back (unwarp)
  - Output the resulting mosaic

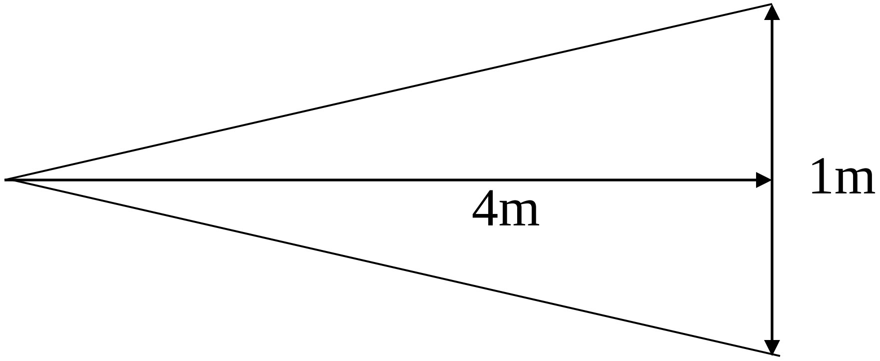
Cylindrical Panoramas

- Map image to cylindrical or spherical coordinates
  - need *known* focal length
  - Work only if a single tilt (e.g., camera on tripod)

Image 384x300 **f = 180 (pixels)** **f = 280** **f = 380**

Determining the focal length

1. Initialize from homography ***H***
   (see text or [SzSh'97])
2. Use camera's EXIF tags (approx.)
3. Use a tape measure
4. Try and error ☺



Practical Methods for F

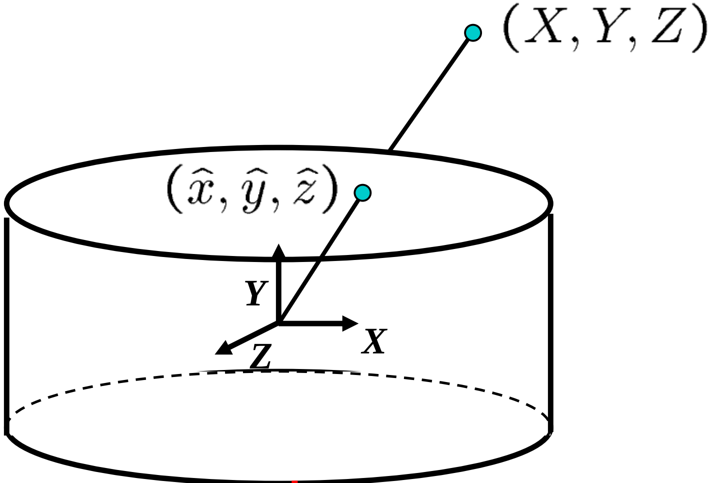
- Use program jhead (http://www.sentex.net/~mwandel/jhead/)
- Mac, Windows, and Linux
- Sample outputs

```
File name    : 0805-153933.jpg
File size    : 463023 bytes
File date    : 2001:08:12 21:02:04
Camera make  : Canon
Camera model : Canon PowerShot S100
Date/Time    : 2001:08:05 15:39:33
Resolution   : 1600 x 1200
Flash used   : No
Focal length :  5.4mm  (35mm equivalent: 36mm)
CCD Width    : 5.23mm
Exposure time: 0.100 s  (1/10)
Aperture     : f/2.8
Focus Dist.  : 1.18m
Metering Mode: center weight
Jpeg process : Baseline
```

Calculating F

- With image resolution (width x height), CCD width and f
  - f*(width/CCD width) or 5.4*(1600/5.23) = 1652 (pixels)
- With equivalent f (35mm film is 36mmx24mm)
  - (equivalent f)*(width/36) or 36*(1600/36) = 1600 (pixels)
- If you don't have the above (more often than not), guess!
  - No zoom f ~ (picture width in pixels)
  - 2x zoom f ~ 2 * (picture width in pixels)

Cylindrical projection



unit cylinder

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

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

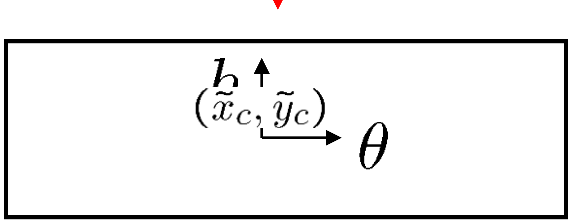
- Convert to cylindrical coordinates

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

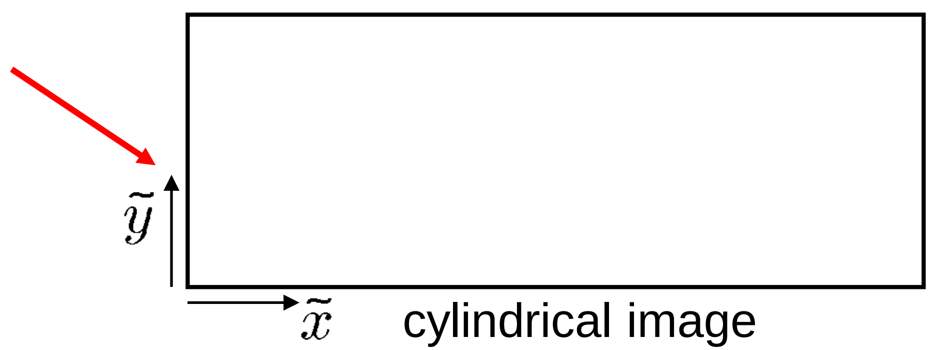
- Convert to cylindrical image coordinates

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

  – s defines size of the final image



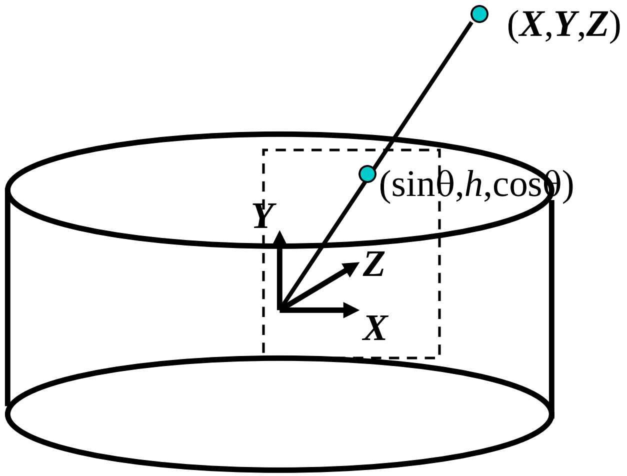
unwrapped cylinder



cylindrical image

Cylindrical warping

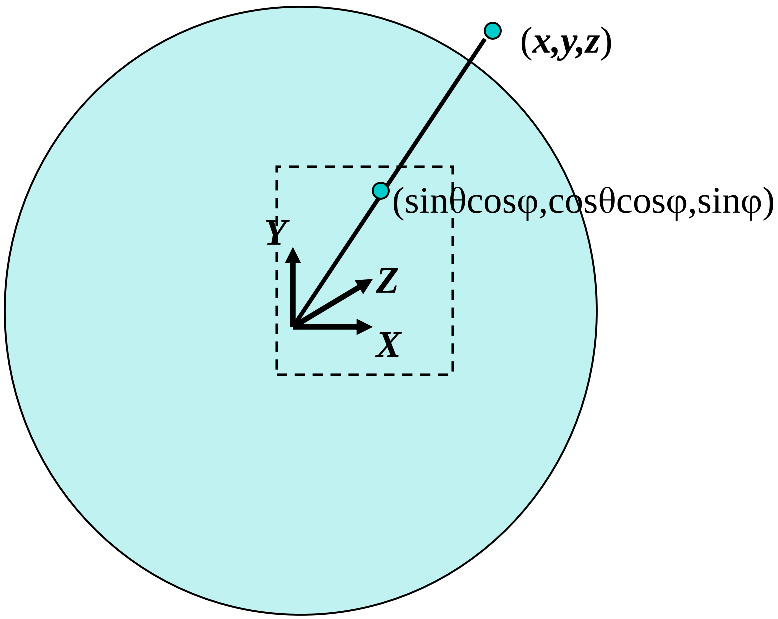
- Given focal length *f* and image center (x_c, y_c)



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

Spherical warping

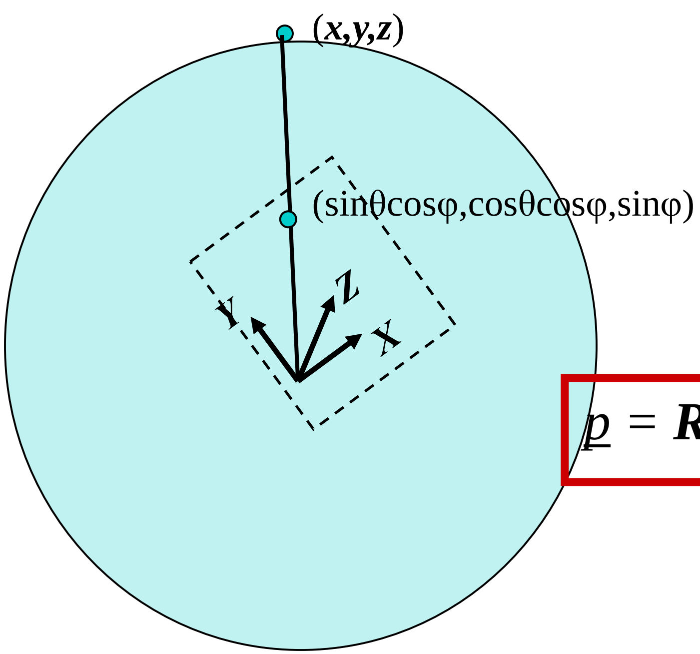
- Given focal length *f* and image center (x_c, y_c)



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

3D rotation

- Rotate image before placing on unrolled sphere



$\underline{p} = \boldsymbol{R}\, p$

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

Radial distortion

- Correct for "bending" in wide field of view lenses

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

Fisheye lens

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

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

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

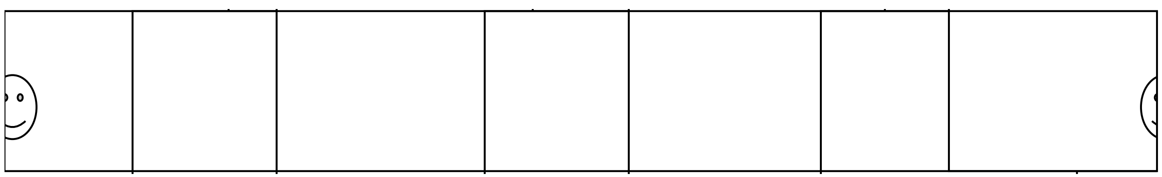
uations become

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

Image Stitching

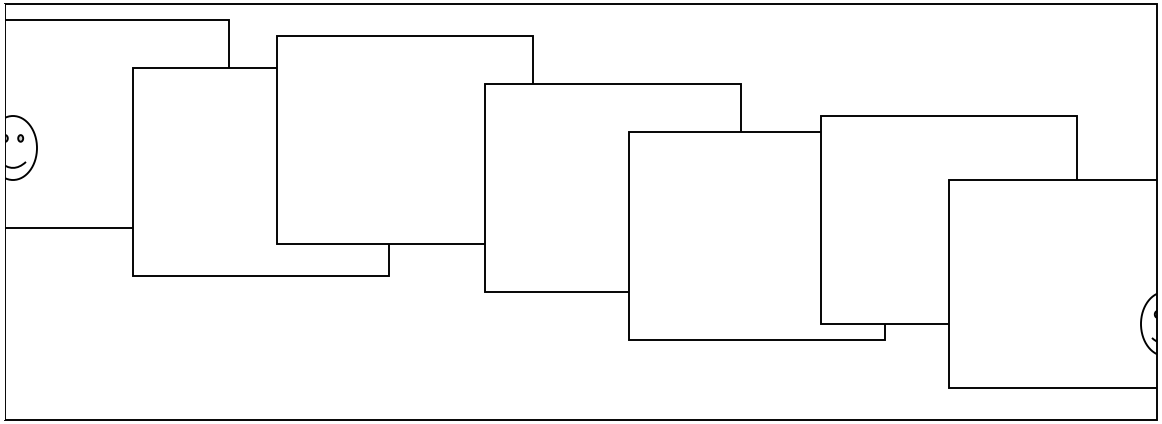
1. Align the images over each other
   - camera pan ↔ translation on cylinder
2. Blend the images together

Assembling the panorama



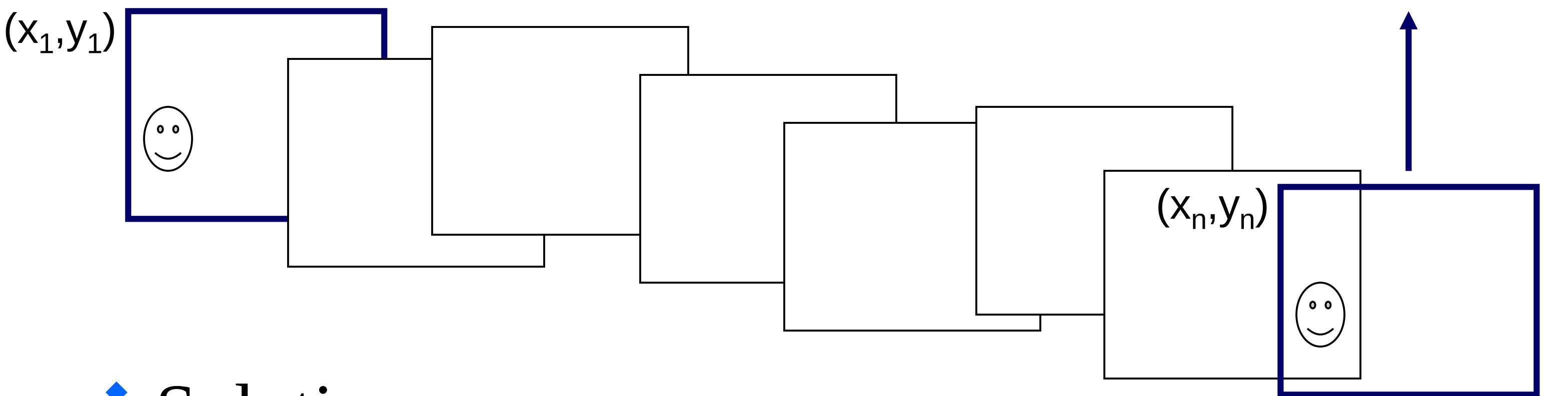
- Stitch pairs together, blend, then crop

Problem: Drift



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

Problem: Drift

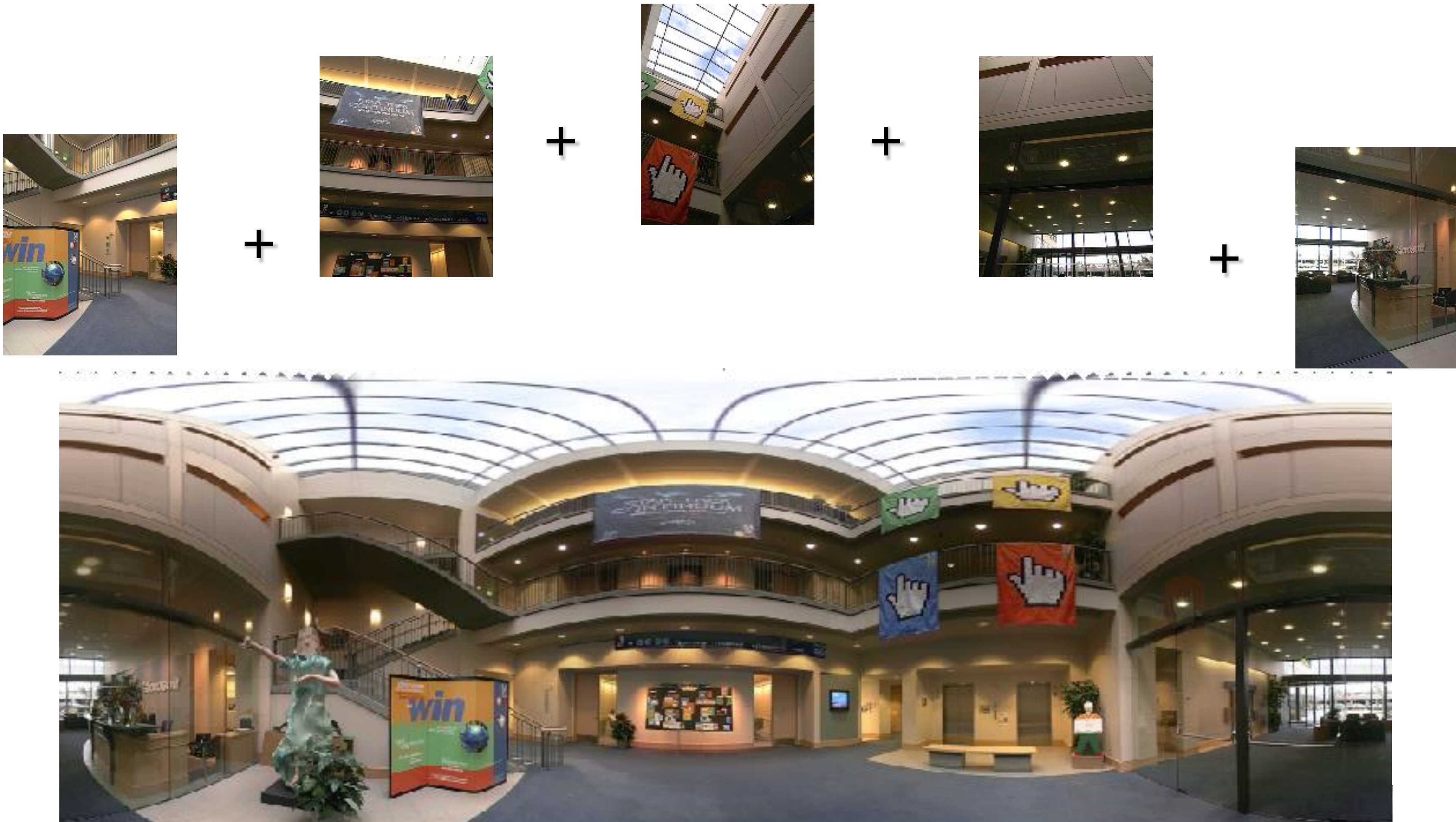


copy of first image

- 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”

Full-view (360° spherical) panoramas

Full-view Panorama



Texture Mapped Model

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)

Bundle adjustment formulations

All pairs optimization:

Confidence / uncertainty of point i in image j

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

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

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

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

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

Bundle adjustment using 3-D ray:

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

3-D ray from point i

All-pairs 3-D ray formulation:

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

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*