Image Stitching

Ali Farhadi
CSE 576

Several slides from Rick Szeliski, Steve Seitz, Derek Hoiem, and Ira Kemelmacher

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

Slide credit: Vaibhav Vaish

How to do it?

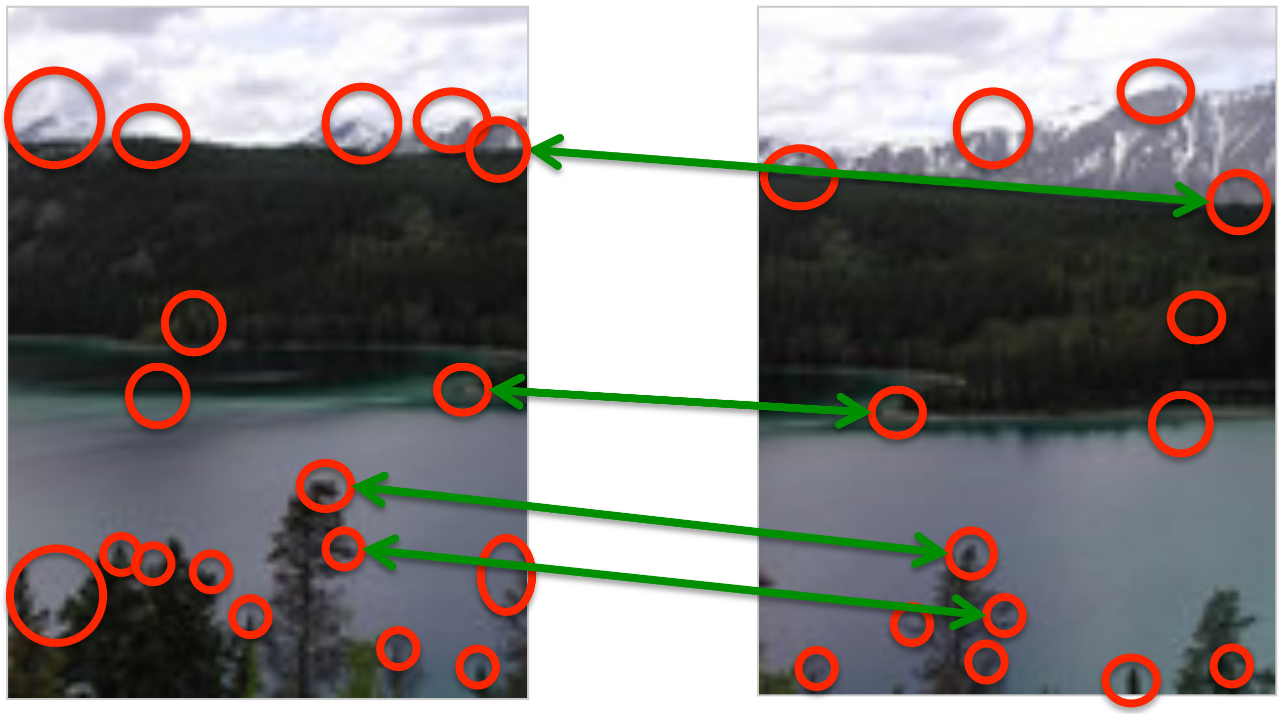
- Basic Procedure
  1. Take a sequence of images from the same position
     1. Rotate the camera about its optical center
  2. Compute transformation between second image and first
  3. Shift the second image to overlap with the first
  4. Blend the two together to create a mosaic
  5. If there are more images, repeat

1. Take a sequence of images from the same position

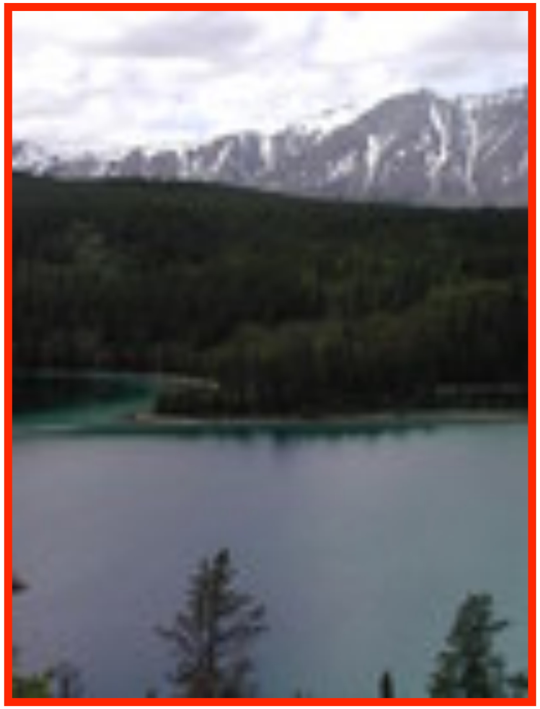
- Rotate the camera about its optical center

2. Compute transformation between images

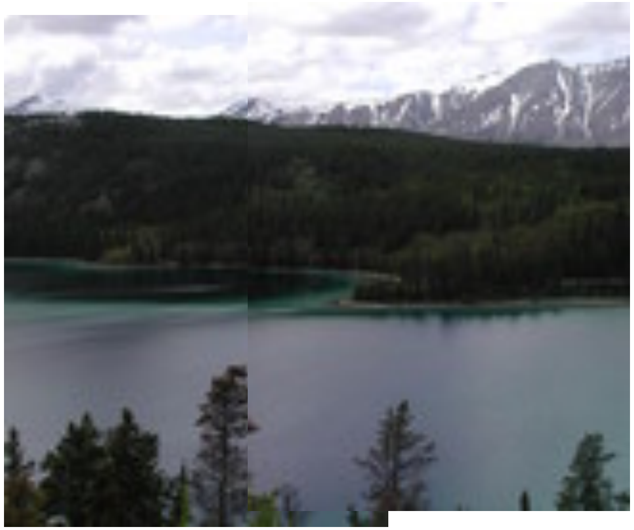
- Extract interest points
- Find Matches
- Compute transformation ?

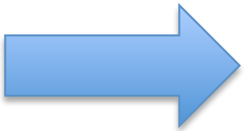


3. Shift the images to overlap



4. Blend the two together to create a mosaic





5. Repeat for all images

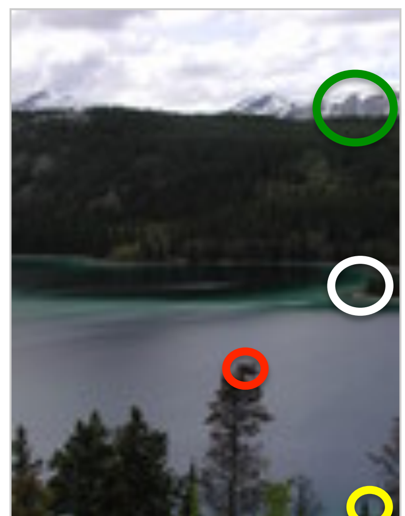
How to do it?

- Basic Procedure
  ✓ 1. Take a sequence of images from the same position
     1. Rotate the camera about its optical center
  2. Compute transformation between second image and first
  3. Shift the second image to overlap with the first
  4. Blend the two together to create a mosaic
  5. If there are more images, repeat

Compute Transformations

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

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



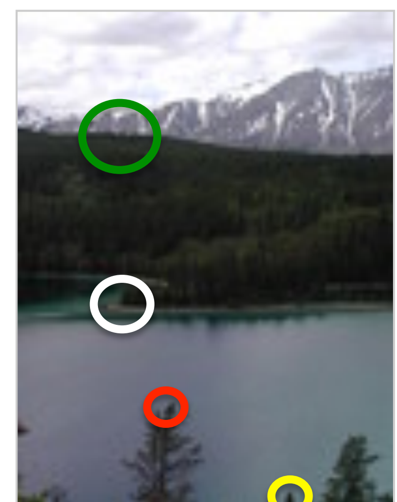
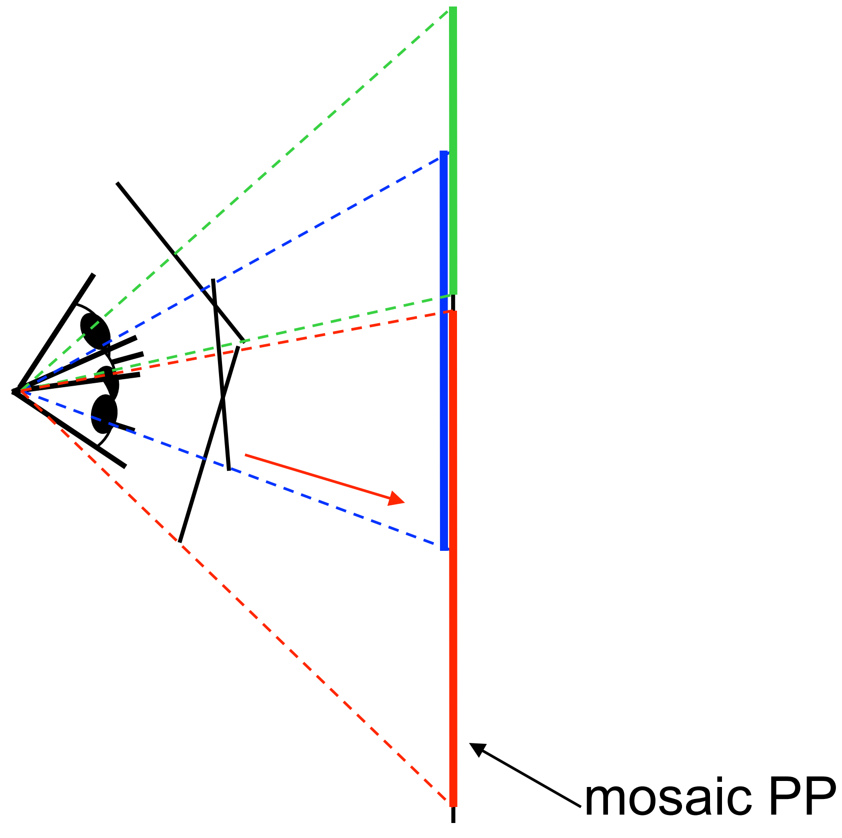
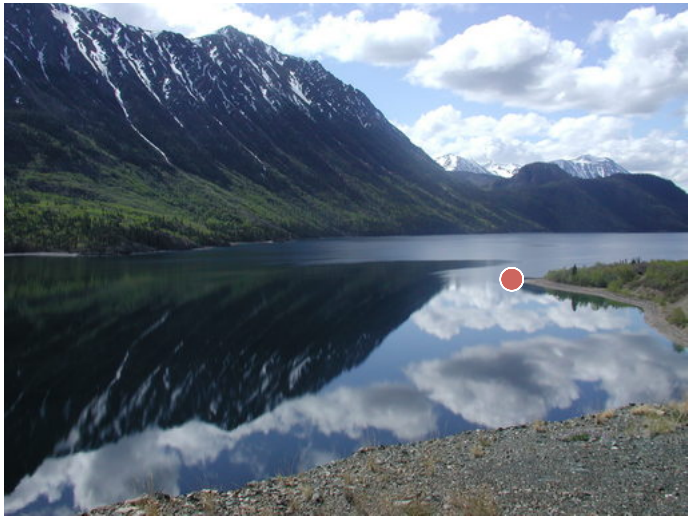


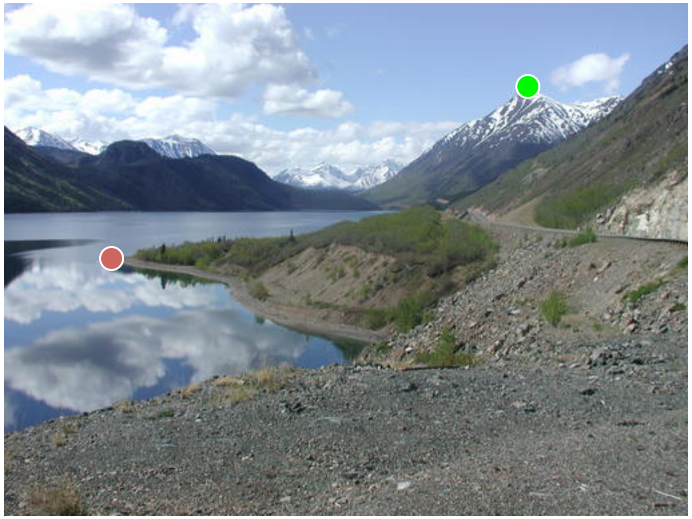
Image reprojection



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

Example





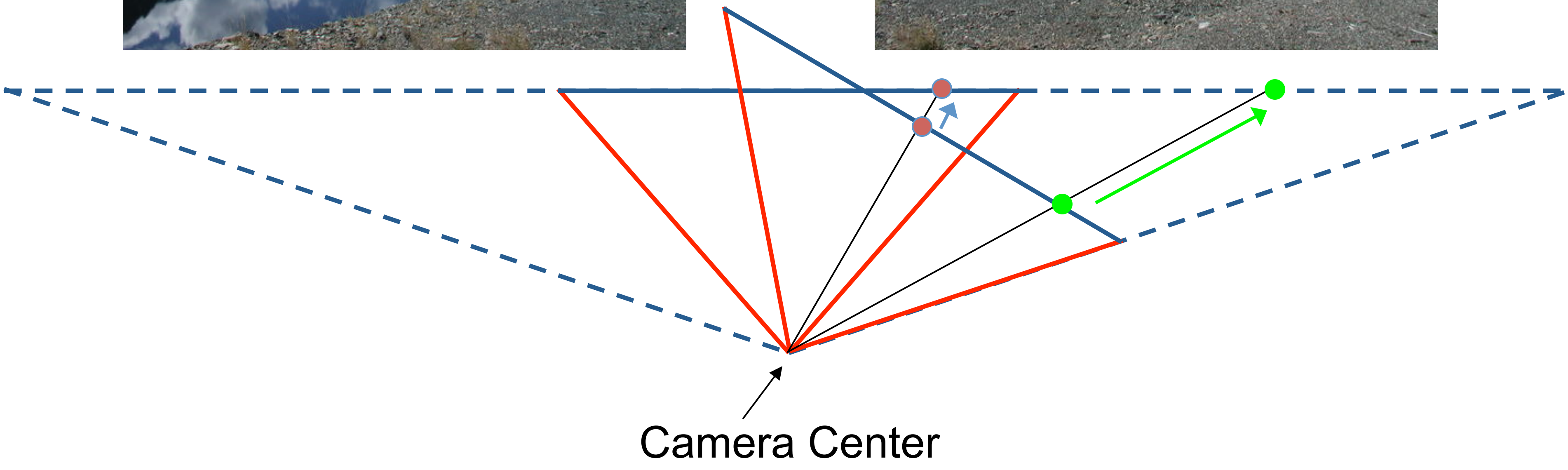
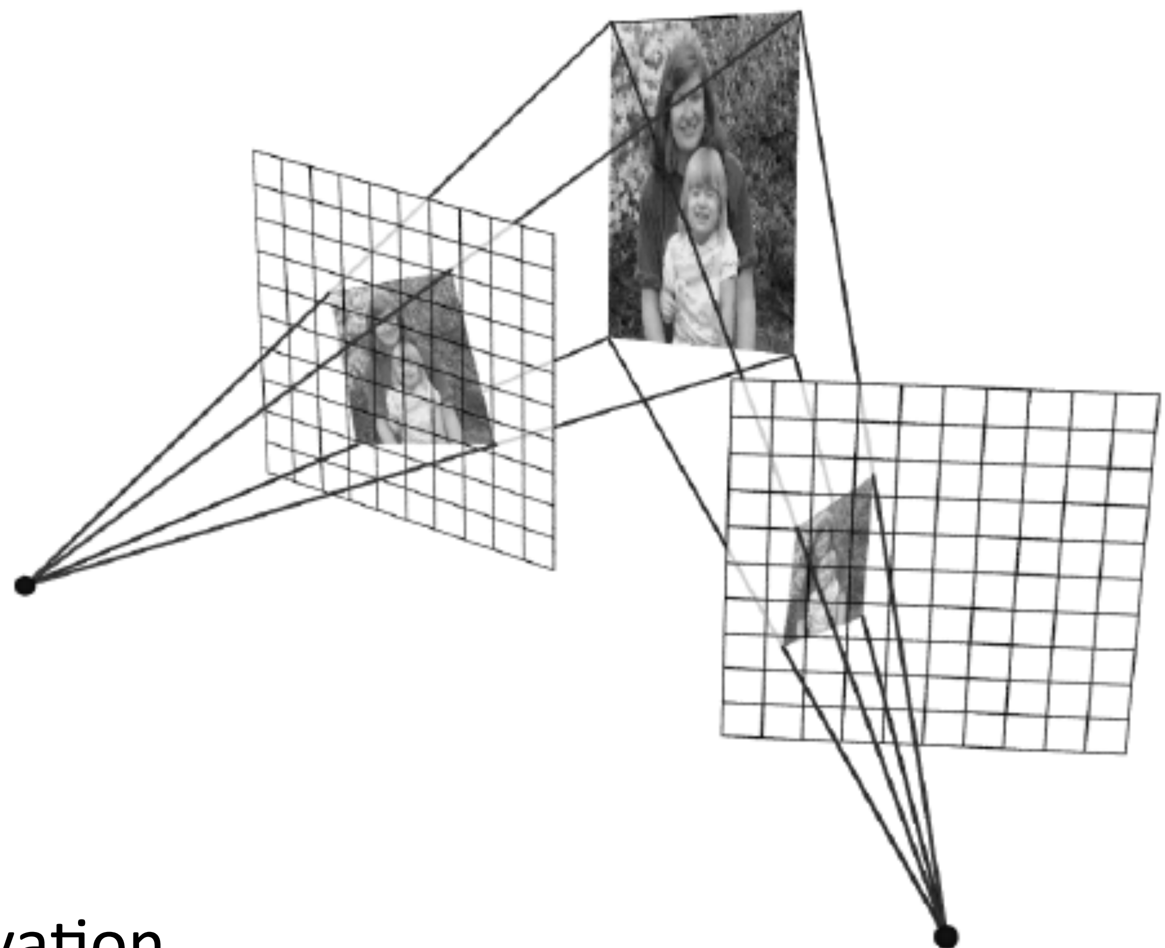


Image reprojection



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

Motion models

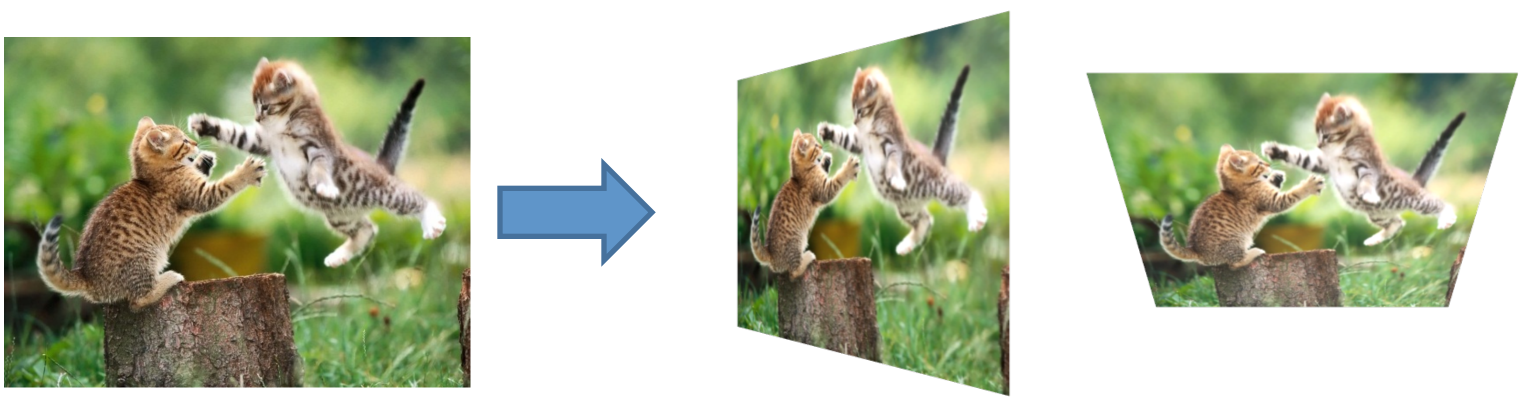
- What happens when we take two images with a camera and try to align them?
- translation?
- rotation?
- scale?
- affine?
- Perspective?

Recall: Projective transformations

- (aka *homographies*)

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

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



Parametric (global) warping

- Examples of parametric warps:

translation

rotation

aspect

affine

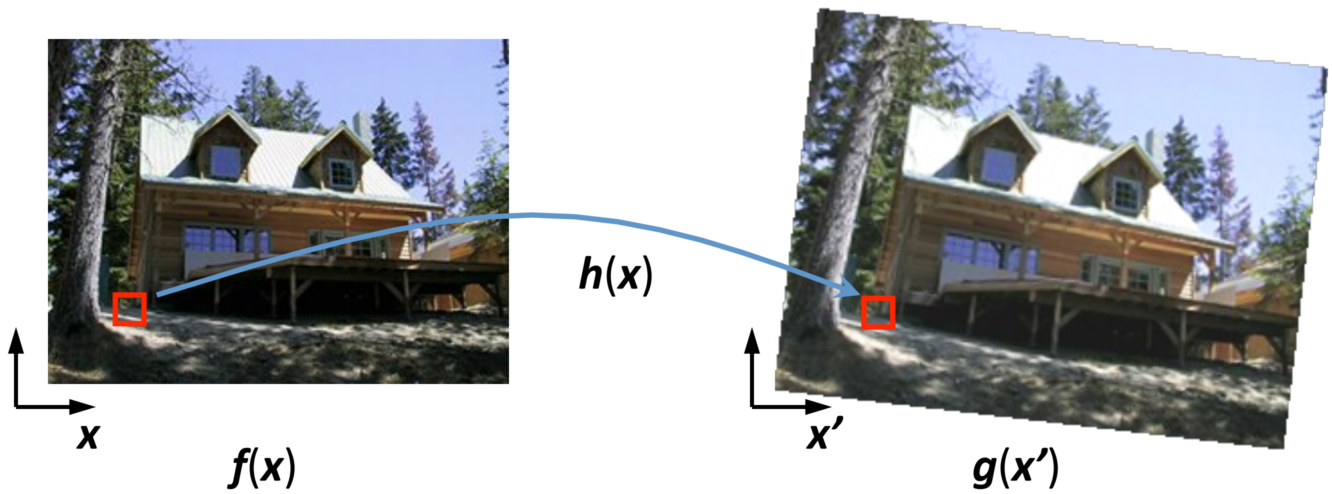
perspective

2D coordinate transformations

- translation: $\boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{t}$ $\boldsymbol{x} = (x,y)$
- 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)

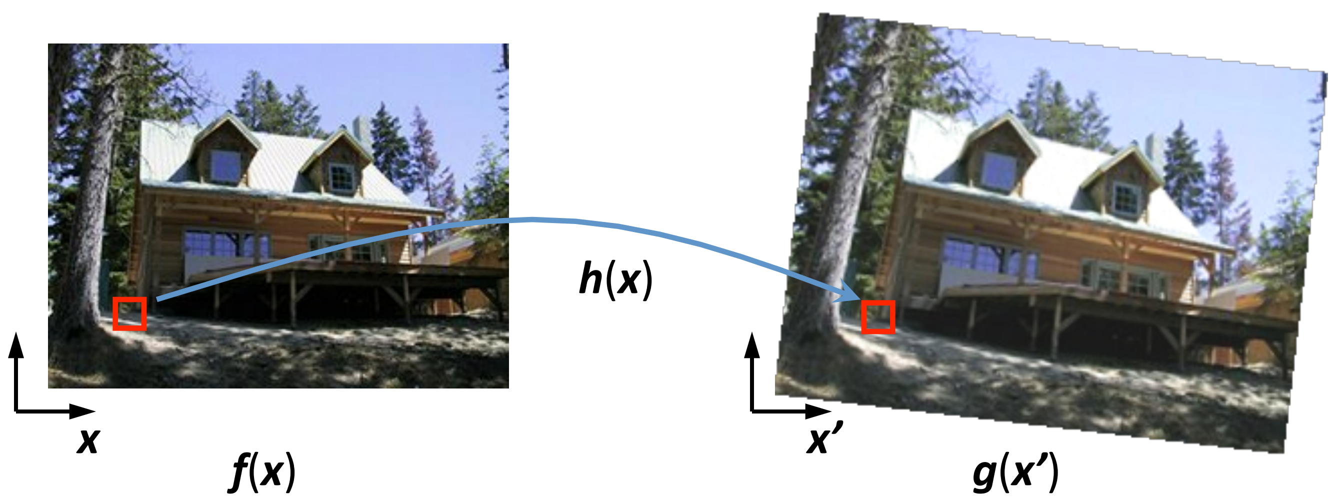
Image Warping

- 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 $\boldsymbol{g}(\boldsymbol{x'}) = \boldsymbol{f}(\boldsymbol{h}(\boldsymbol{x}))$?



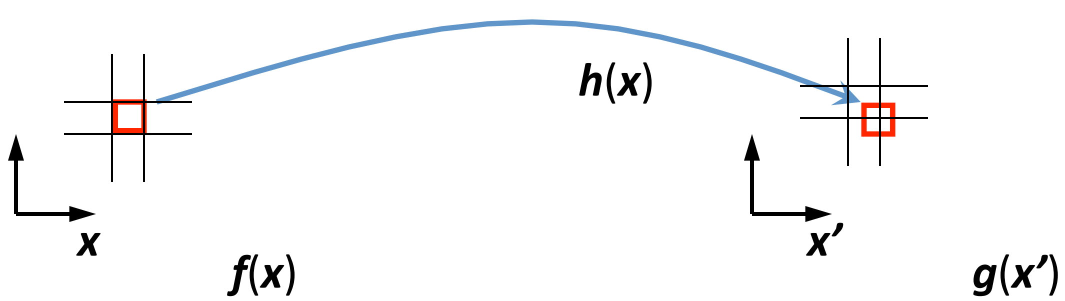
Forward Warping

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



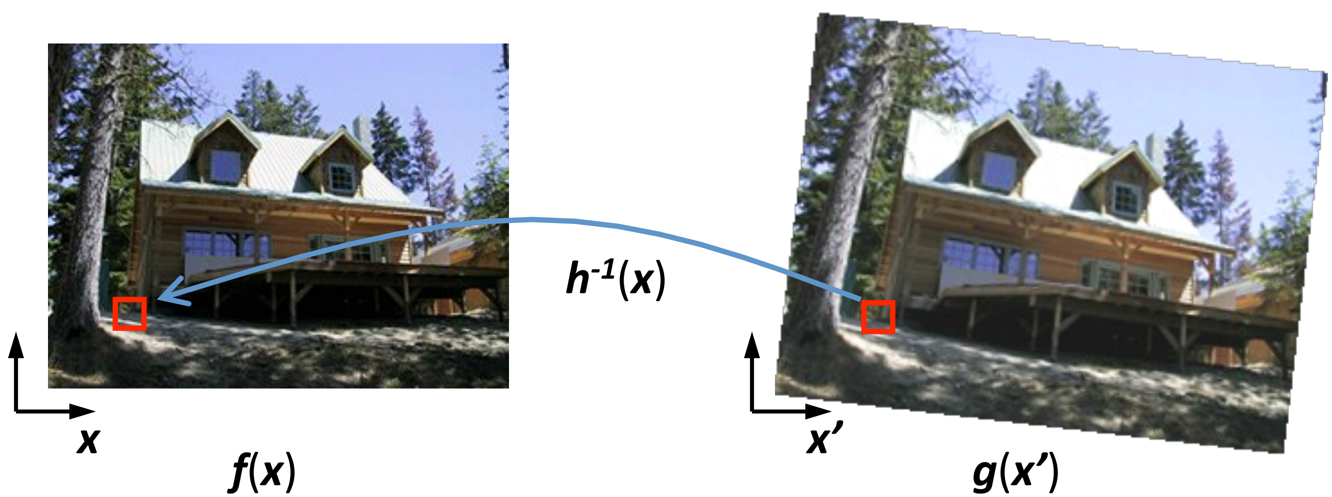
Forward Warping

- Send each pixel ***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*)



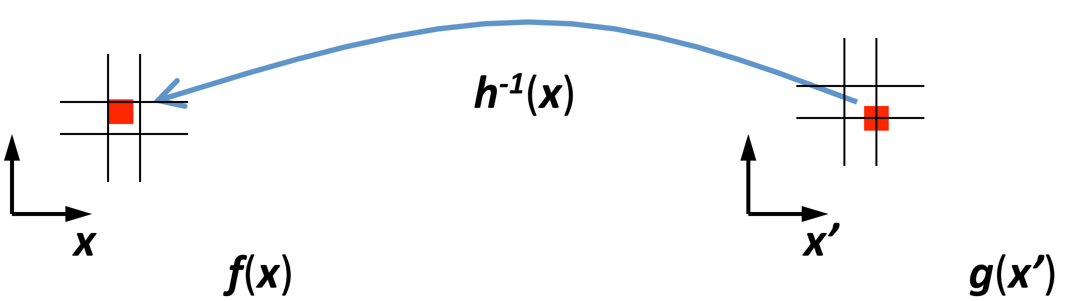
Inverse Warping

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



Inverse Warping

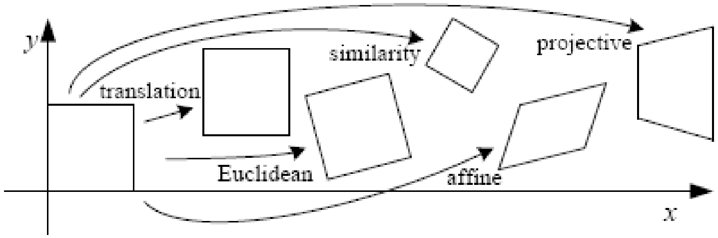
- Get each pixel ***g***(***x'***) from its corresponding location ***x'*** = ***h***(***x***) in ***f***(***x***)
  - What if pixel comes from "between" two pixels?
  - Answer: *resample* color value from *interpolated* source image



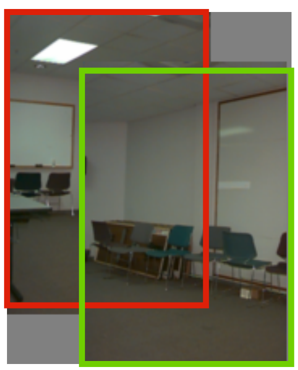
Interpolation

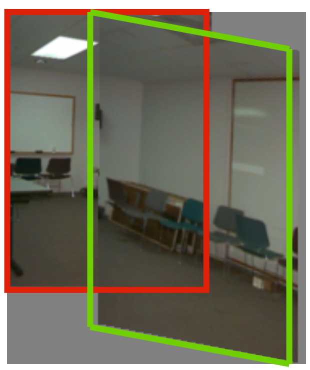
- Possible interpolation filters:
  - nearest neighbor
  - bilinear
  - bicubic (interpolating)

Motion models



Translation | **Affine** | **Perspective**





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

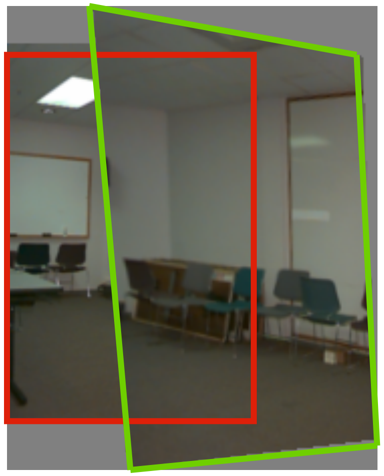
Finding the transformation

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

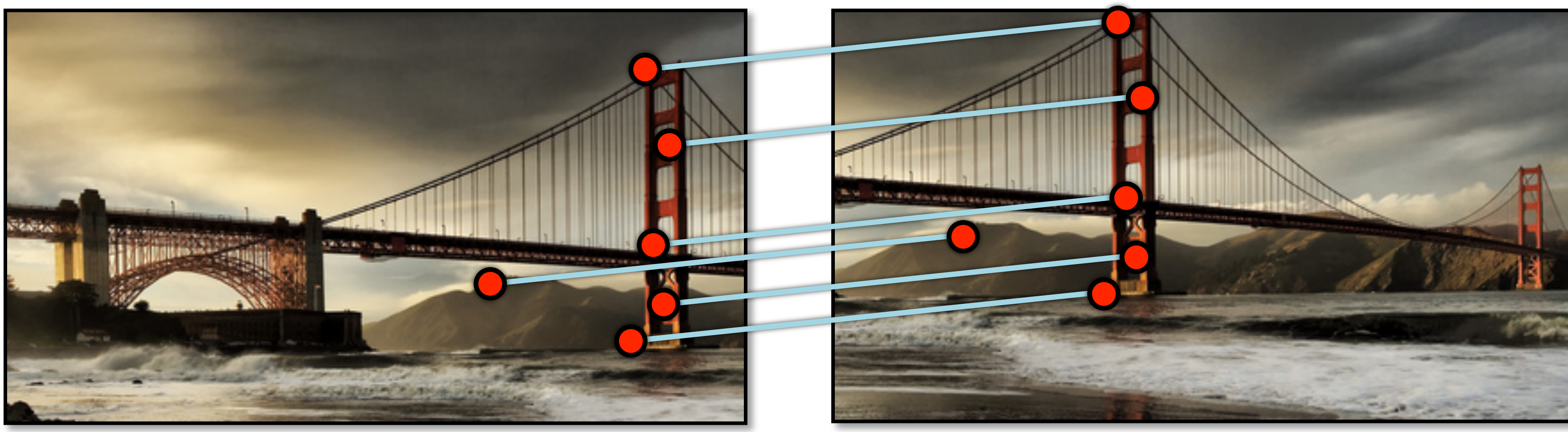
- How many corresponding points do we need to solve?

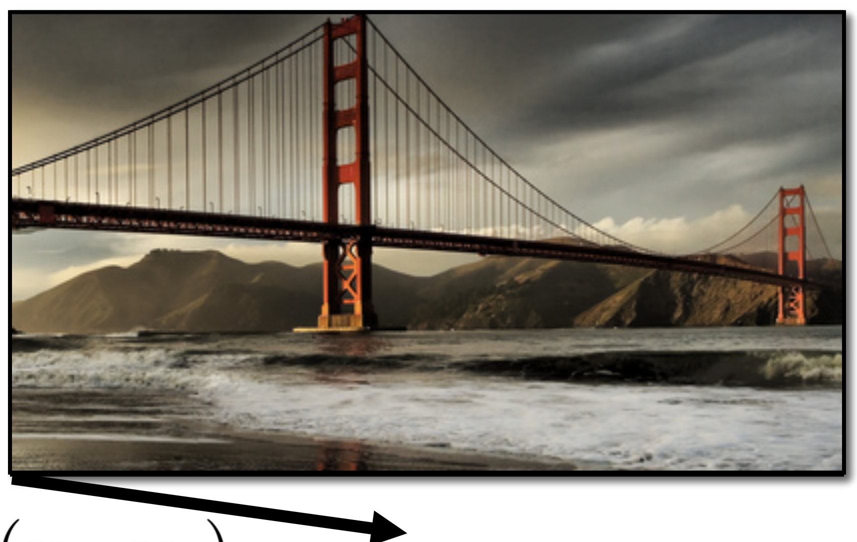
Plane perspective mosaics

– 8-parameter generalization of affine motion
  - works for pure rotation or planar surfaces
– Limitations:
  - local minima
  - slow convergence
  - difficult to control interactively



Simple case: translations

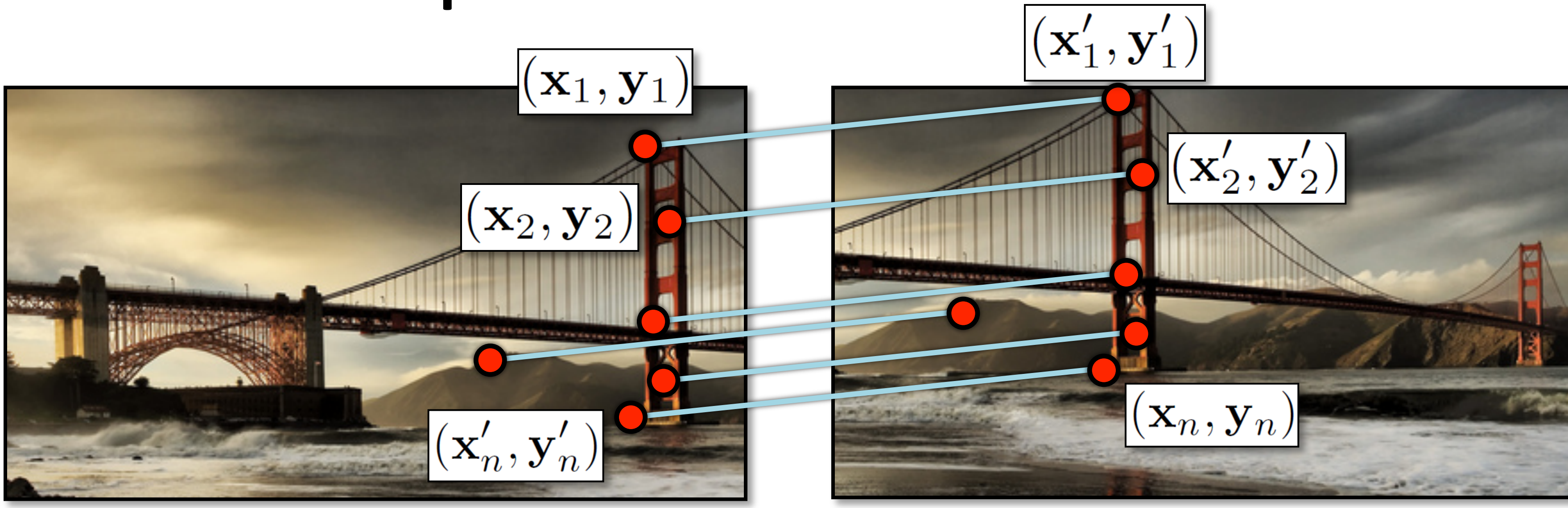




$(\mathbf{x}_t, \mathbf{y}_t)$

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

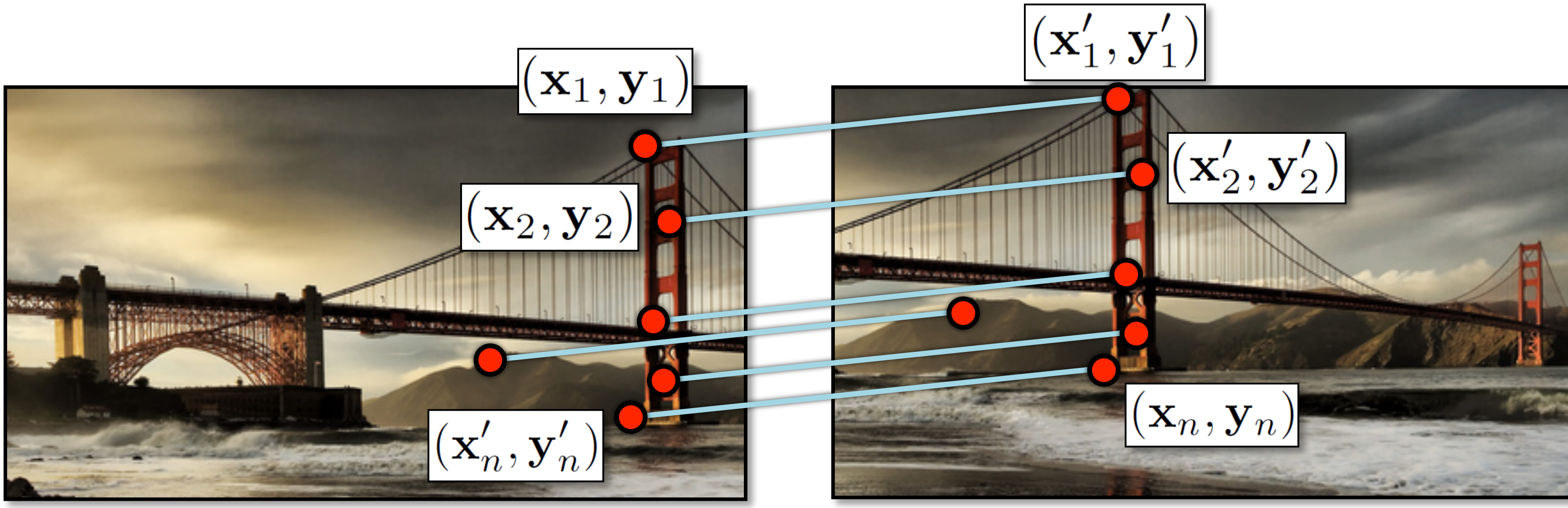
Simple case: translations



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

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

Simple case: translations

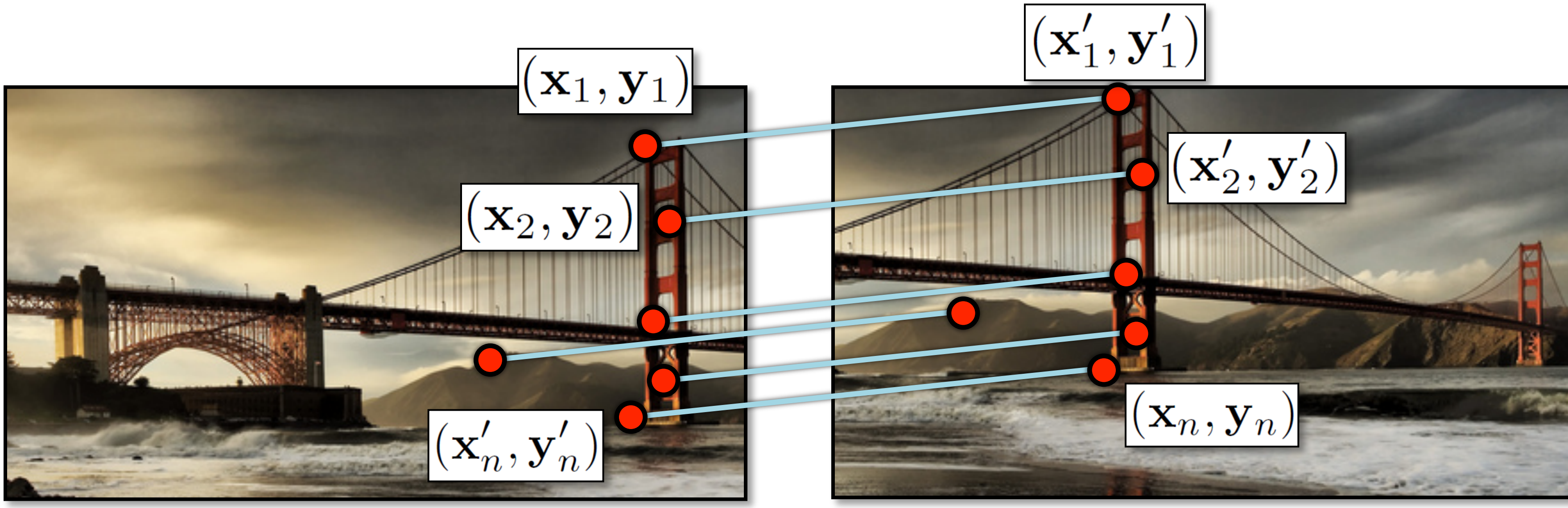


$$\mathbf{x}_i + \mathbf{x_t} = \mathbf{x}'_i$$

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

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

Simple case: translations



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

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

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

Least squares formulation

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

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

- we define the *residuals* as

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

Least squares formulation

- Goal: minimize sum of squared residuals

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

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

Least squares

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

- Find **t** that minimizes

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

- To solve, form the *normal equations*

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

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

Solving for translations

- Using least squares

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

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

$2n$ x 2   2 x 1 $2n$ x 1

Affine transformations

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

- How many unknowns?
- How many equations per match?
- How many matches do we need?

Affine transformations

- Residuals:

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

- Cost function:

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

Affine transformations

- Matrix form

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

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

Solving for homographies

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

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

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

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

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

Solving for homographies

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

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

Direct Linear Transforms

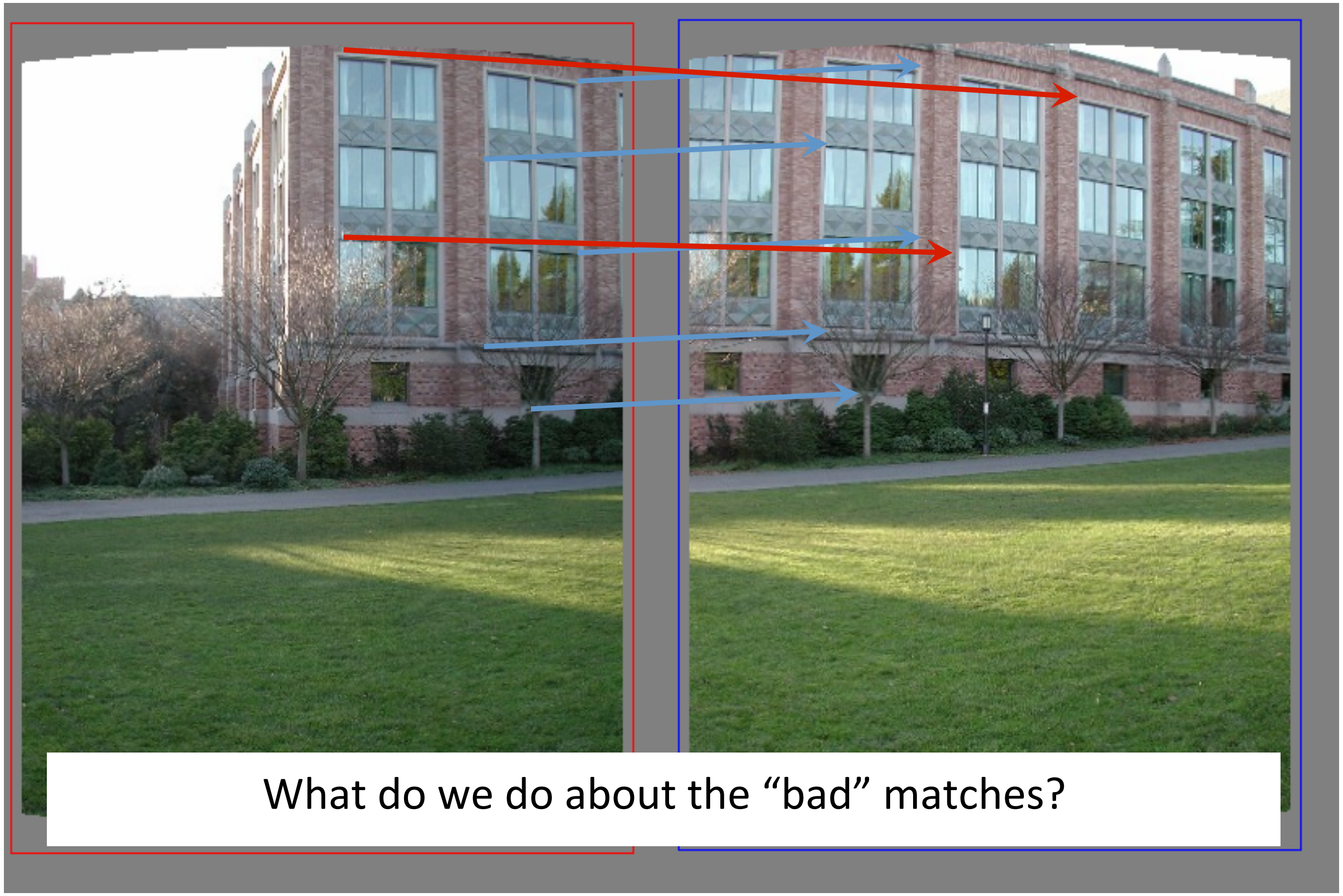
$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1 x_1 & -x'_1 y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1 x_1 & -y'_1 y_1 & -y'_1 \\ & & & & & \vdots & & & \\ x_n & y_n & 1 & 0 & 0 & 0 & -x'_n x_n & -x'_n y_n & -x'_n \\ 0 & 0 & 0 & x_n & y_n & 1 & -y'_n x_n & -y'_n y_n & -y'_n \end{bmatrix} \begin{bmatrix}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

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

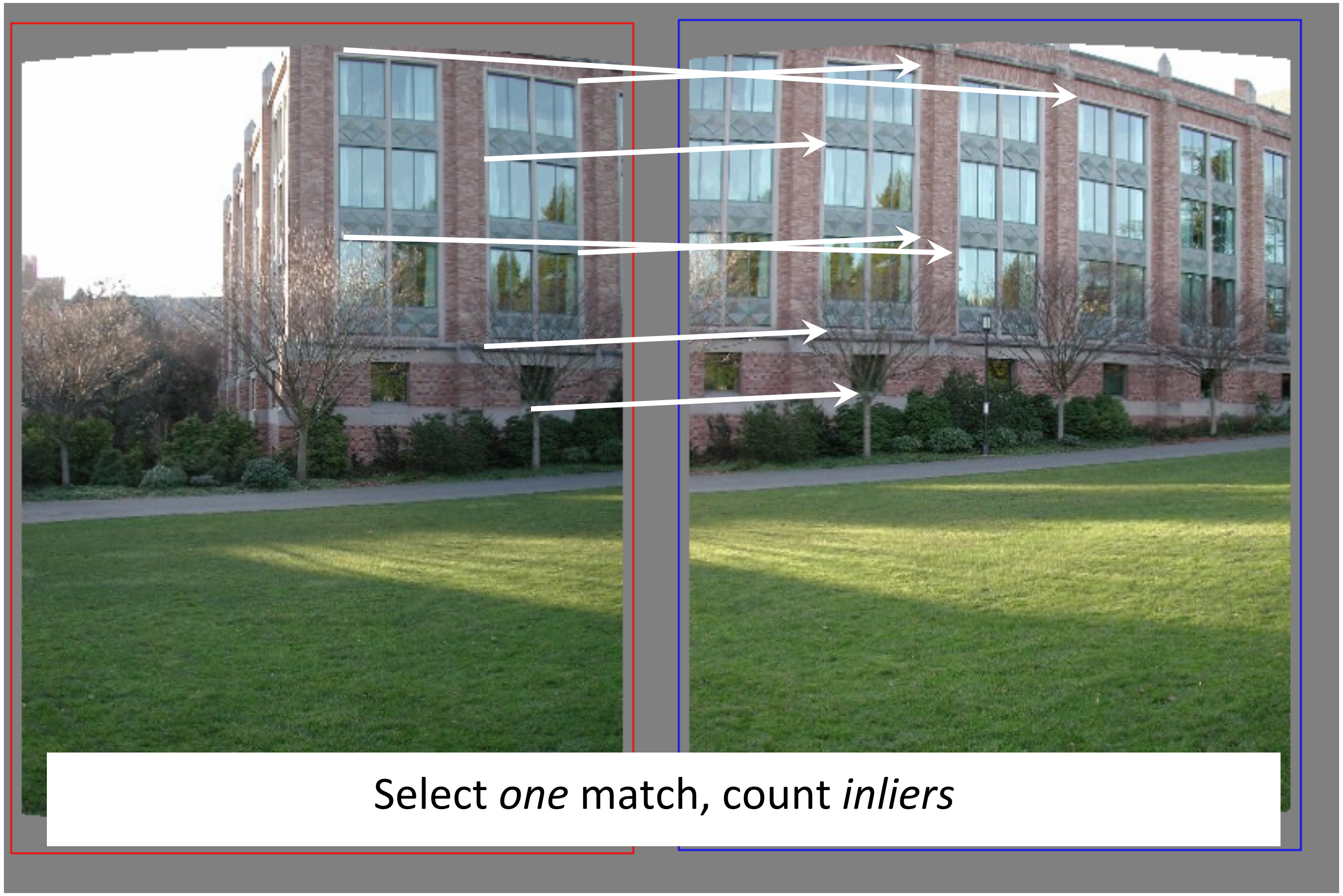
Defines a least squares problem: minimize $\|\mathbf{A}\mathbf{h} - \mathbf{0}\|^2$

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

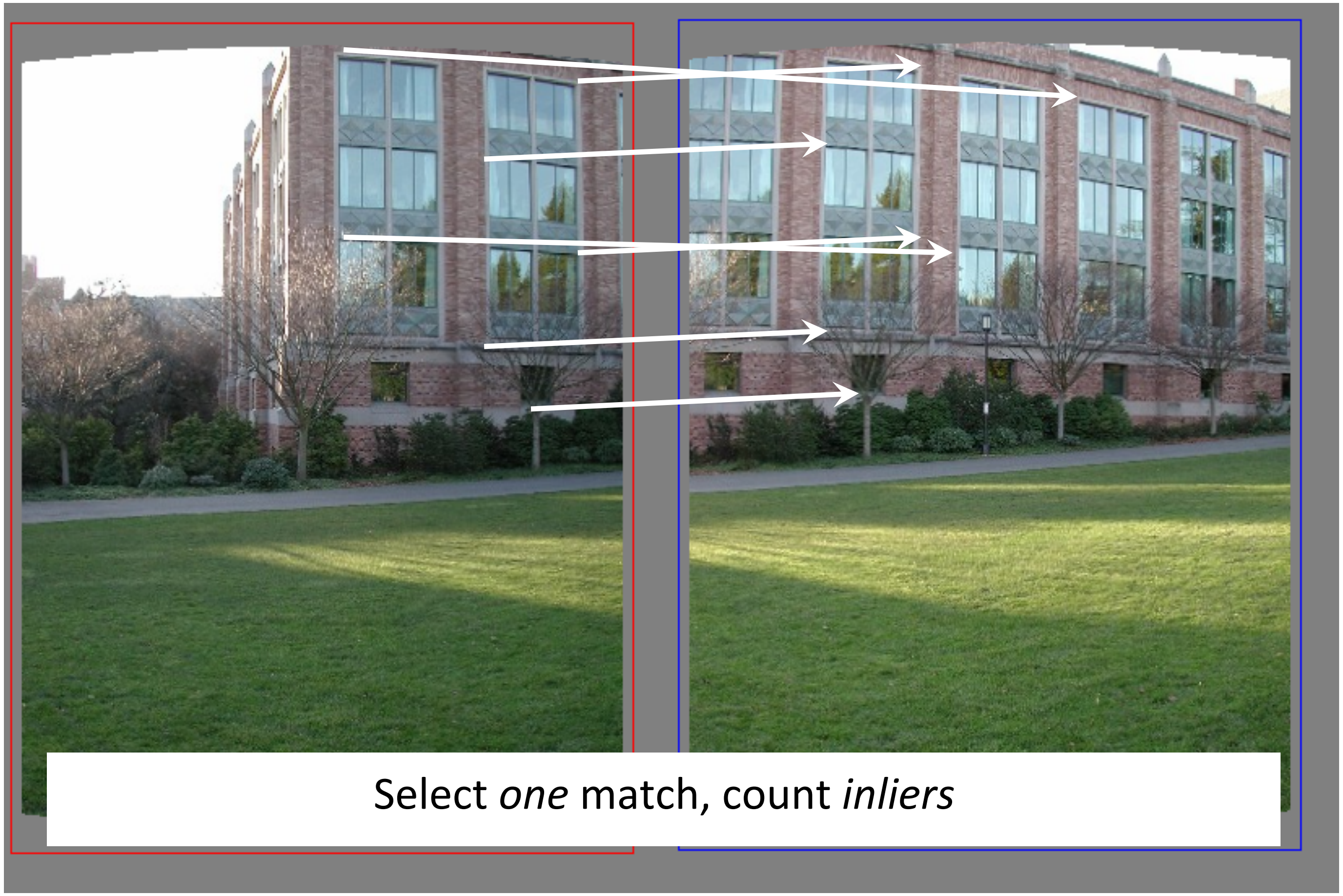
Matching features



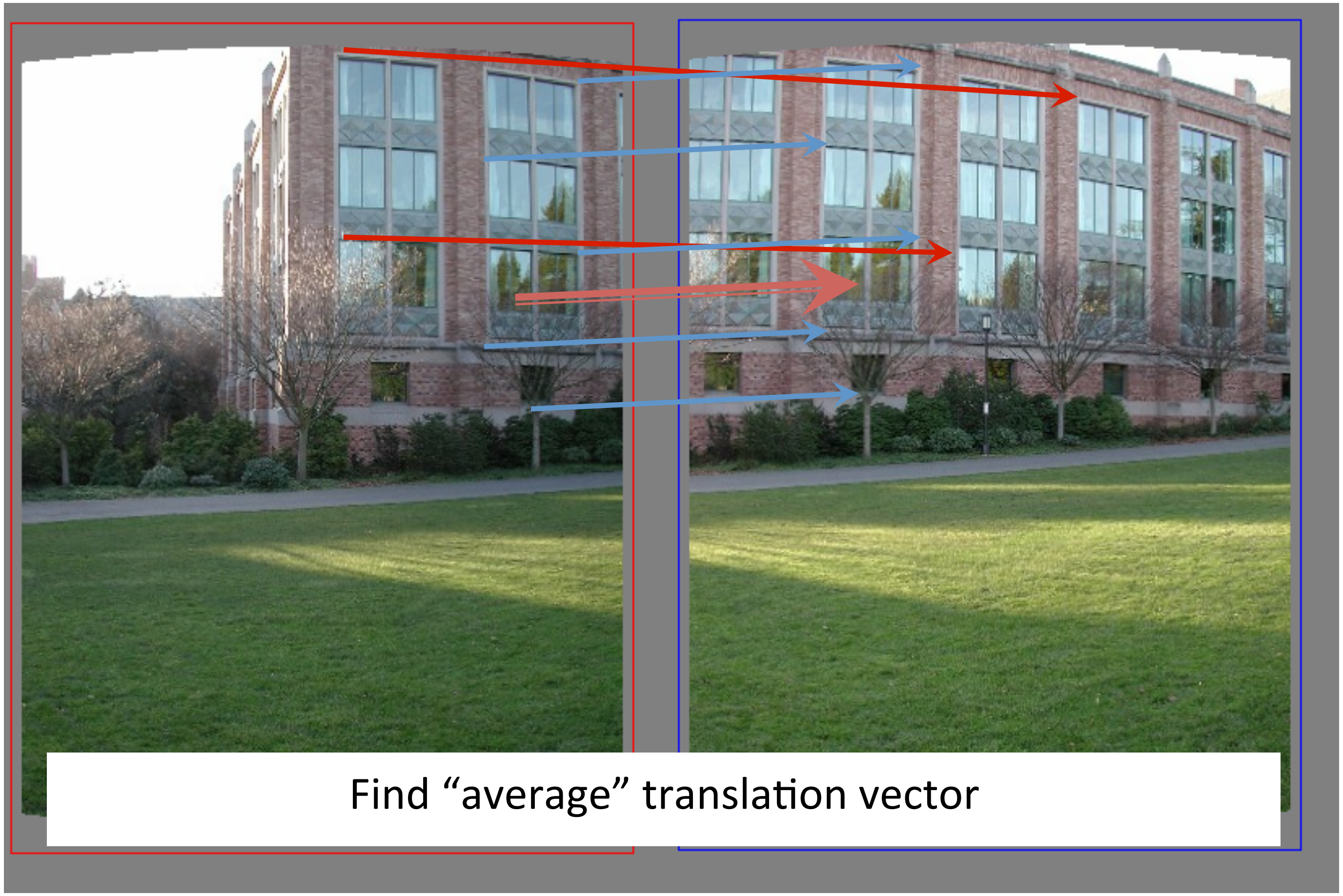
RAndom SAmple Consensus



RAndom SAmple Consensus



Least squares fit



Find "average" translation vector

RANSAC for estimating homography

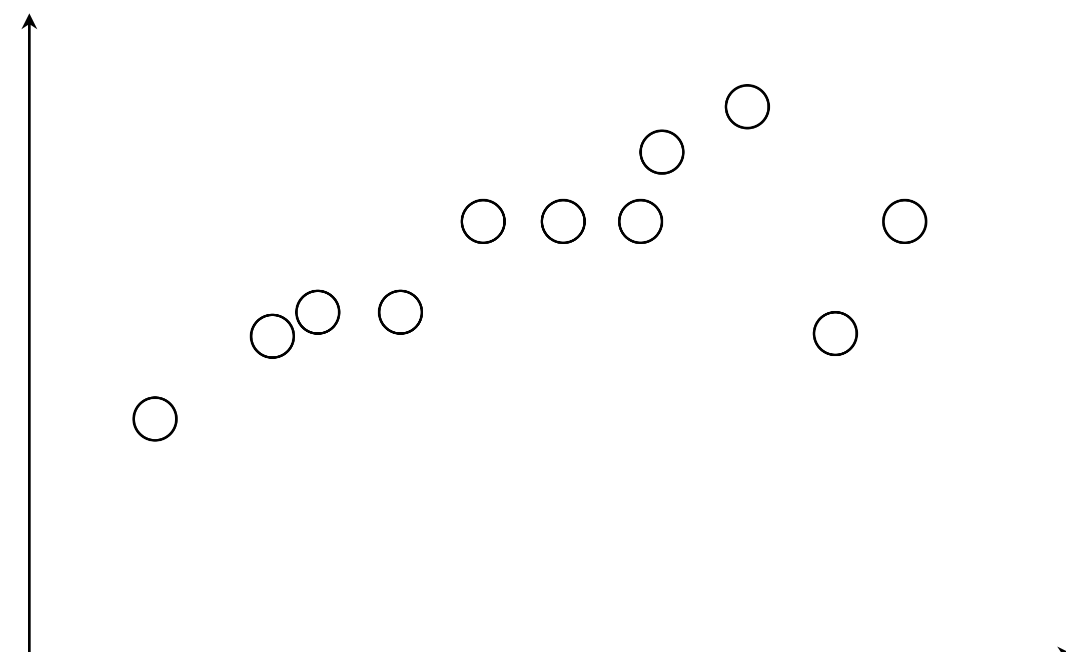
- RANSAC loop:

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

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

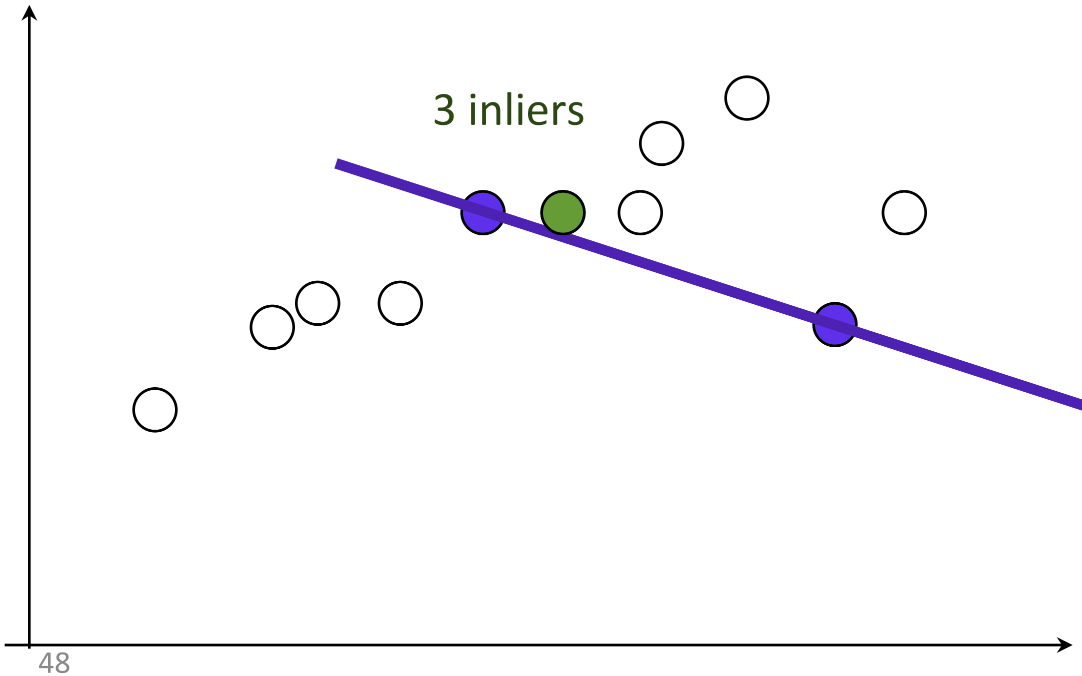
Simple example: fit a line

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



Simple example: fit a line

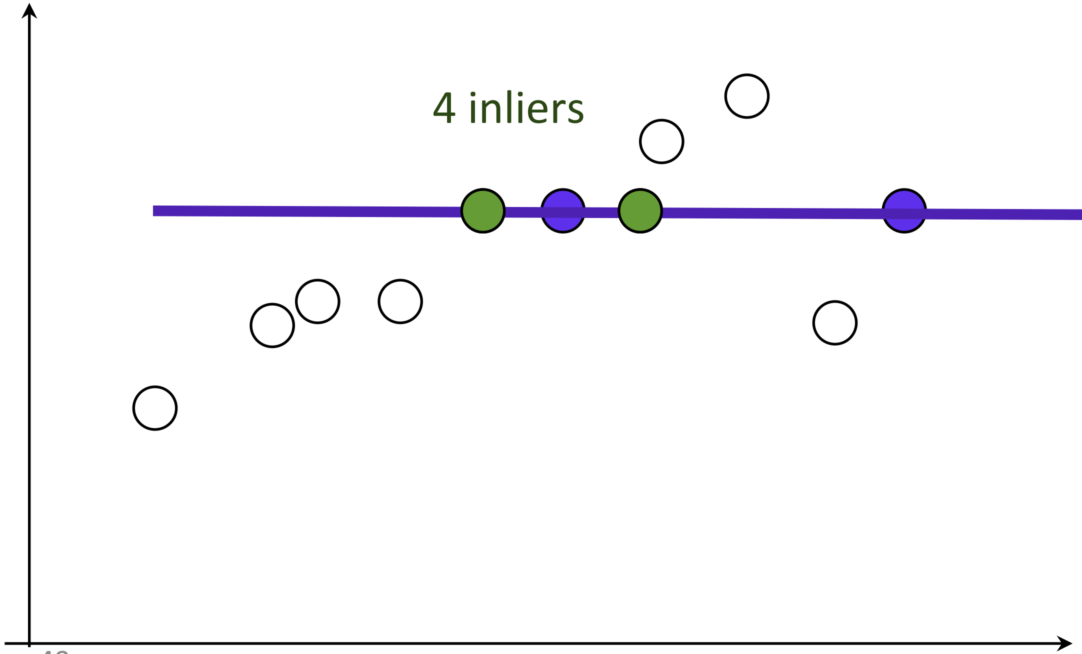
- Pick 2 points
- Fit line
- Count inliers



48

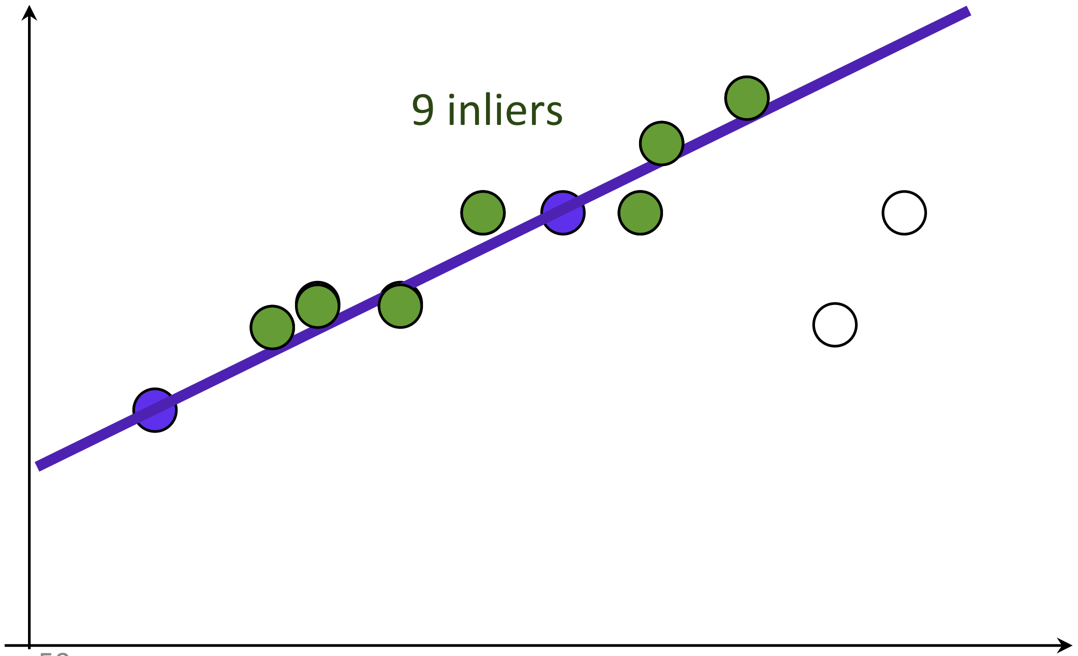
Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers



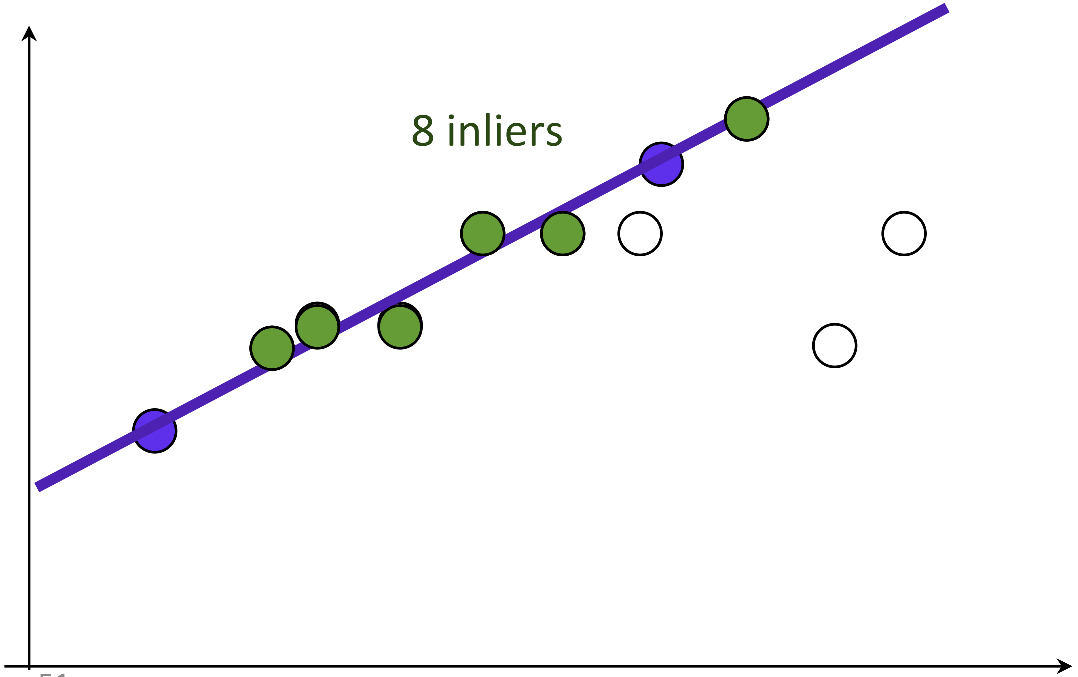
Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers



Simple example: fit a line

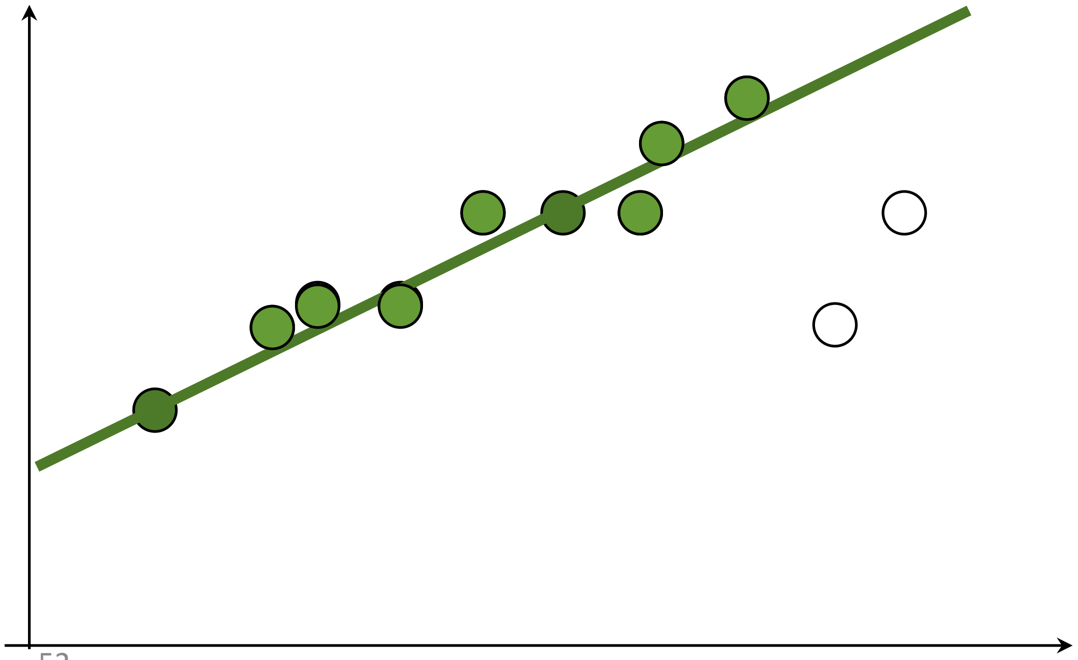
- Pick 2 points
- Fit line
- Count inliers



51

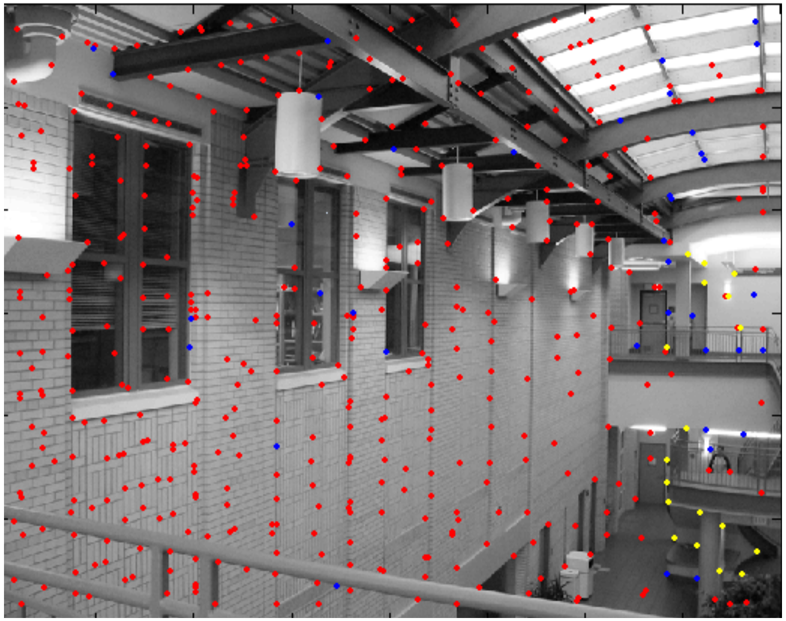
Simple example: fit a line

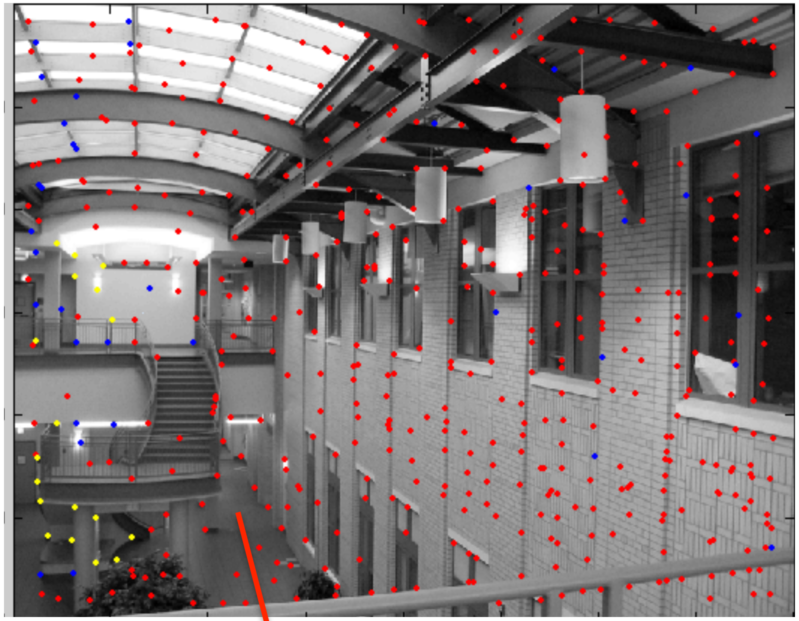
- Use biggest set of inliers
- Do least-square fit



52

RANSAC





Red:

rejected by 2nd nearest neighbor criterion

Blue:

Ransac outliers

Yellow:

inliers

How many rounds?

- If we have to choose *s* samples each time
  - with an outlier ratio *e*
  - and we want the right answer with probability *p*

For probability p of no outliers:

$$N = \log(1 - p) / \log(1 - (1 - \epsilon)^s)$$

- N, number of samples
- s, size of sample set
- ϵ, proportion of outliers

e.g. for $p = 0.95$

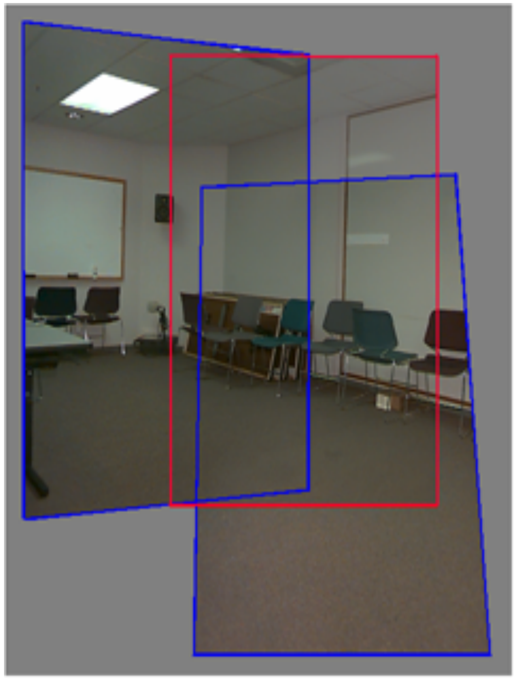
Sample size	Proportion of outliers ϵ						
s	5%	10%	20%	25%	30%	40%	50%
2	2	2	3	4	5	7	11
3	2	3	5	6	8	13	23
4	2	3	6	8	11	22	47
5	3	4	8	12	17	38	95
6	3	4	10	16	24	63	191
7	3	5	13	21	35	106	382
8	3	6	17	29	51	177	766

Rotational mosaics

– Directly optimize rotation and focal length

– Advantages:

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



Rotational mosaic

- Projection equations

1. Project from image to 3D ray

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

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

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

Computing homography

- Assume we have four matched points: How do we compute homography **H**?

Normalized DLT

1. Normalize coordinates for each image
   a) Translate for zero mean
   b) Scale so that average distance to origin is ~sqrt(2)

$$\tilde{\mathbf{x}} = \mathbf{T}\mathbf{x} \qquad \tilde{\mathbf{x}}' = \mathbf{T}'\mathbf{x}'$$

   – This makes problem better behaved numerically

2. Compute $\tilde{\mathbf{H}}$ using DLT in normalized coordinates
3. Unnormalize: $\mathbf{H} = \mathbf{T}'^{-1}\tilde{\mathbf{H}}\mathbf{T}$

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

Computing homography

- Assume we have matched points with outliers: How do we compute homography **H**?

Automatic Homography Estimation with RANSAC

1. Choose number of samples *N*
2. Choose 4 random potential matches
3. Compute **H** using normalized DLT
4. Project points from **x** to **x'** for each potentially matching pair: $\mathbf{x}'_i = \mathbf{H}\mathbf{x}_i$
5. Count points with projected distance < t
   - E.g., t = 3 pixels
6. Repeat steps 2-5 *N* times
   - Choose **H** with most inliers

HZ Tutorial '99

Automatic Image Stitching

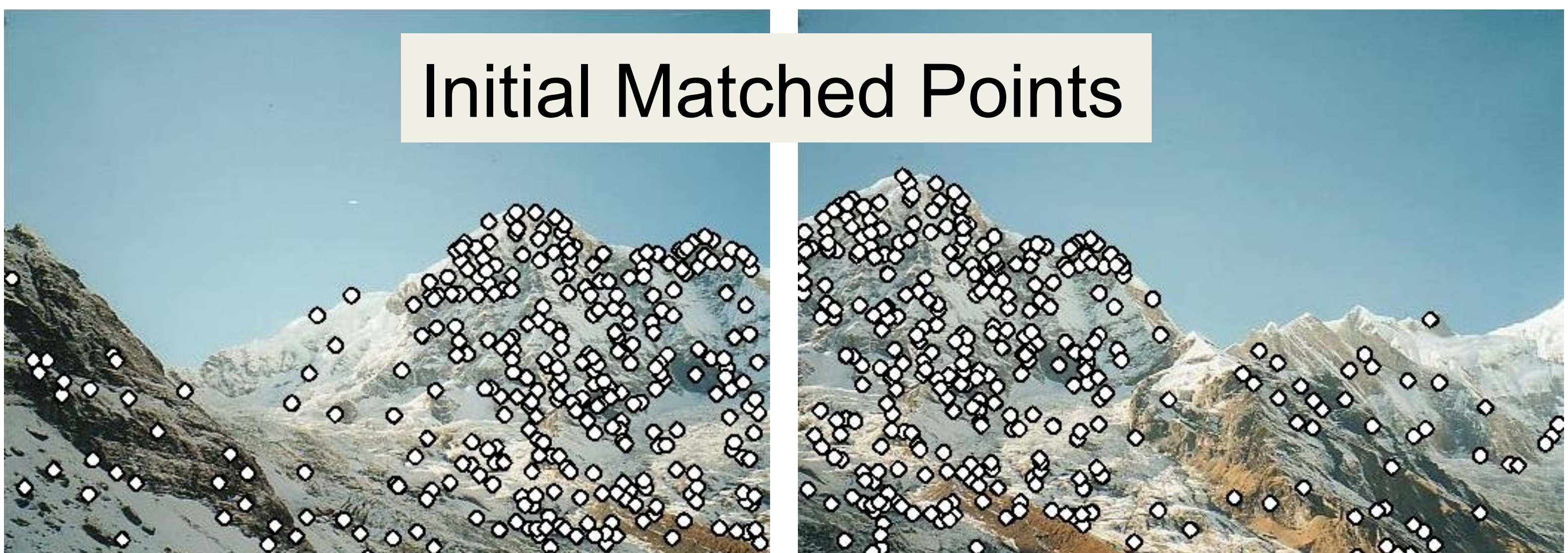
1. Compute interest points on each image

2. Find candidate matches

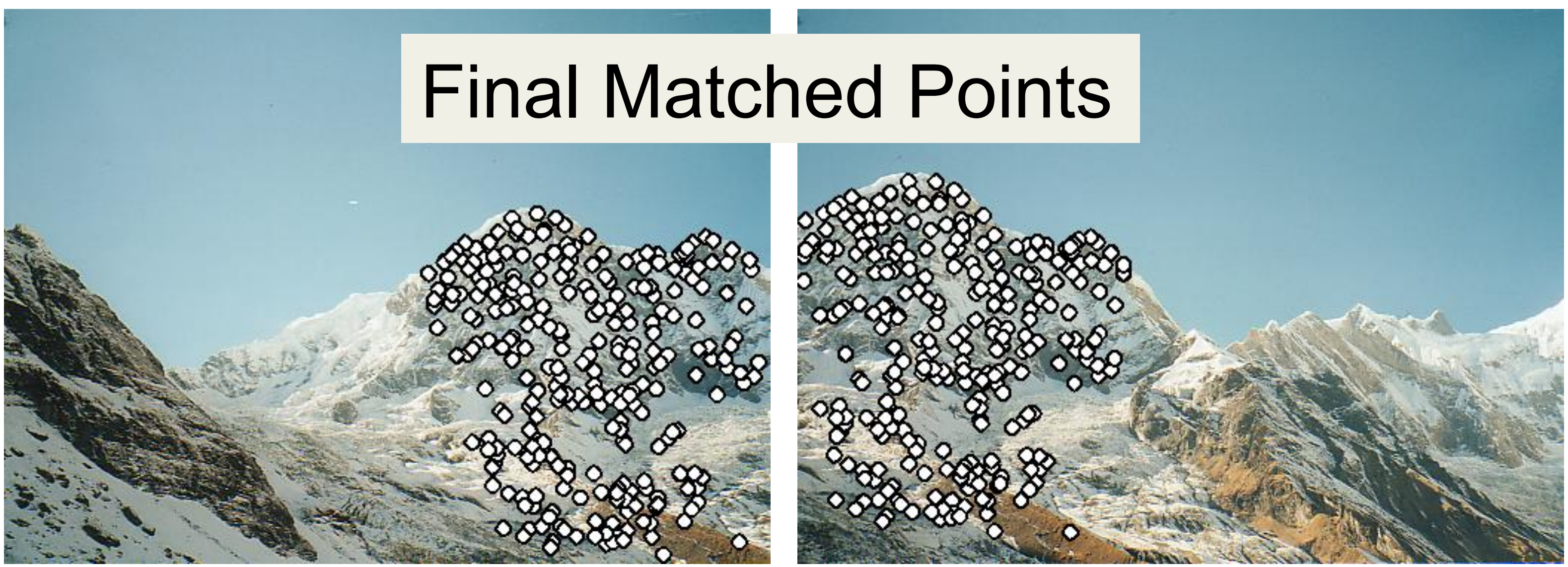
3. Estimate homography **H** using matched points and RANSAC with normalized DLT

4. Project each image onto the same surface and blend

RANSAC for Homography



RANSAC for Homography



Final Matched Points

RANSAC for Homography

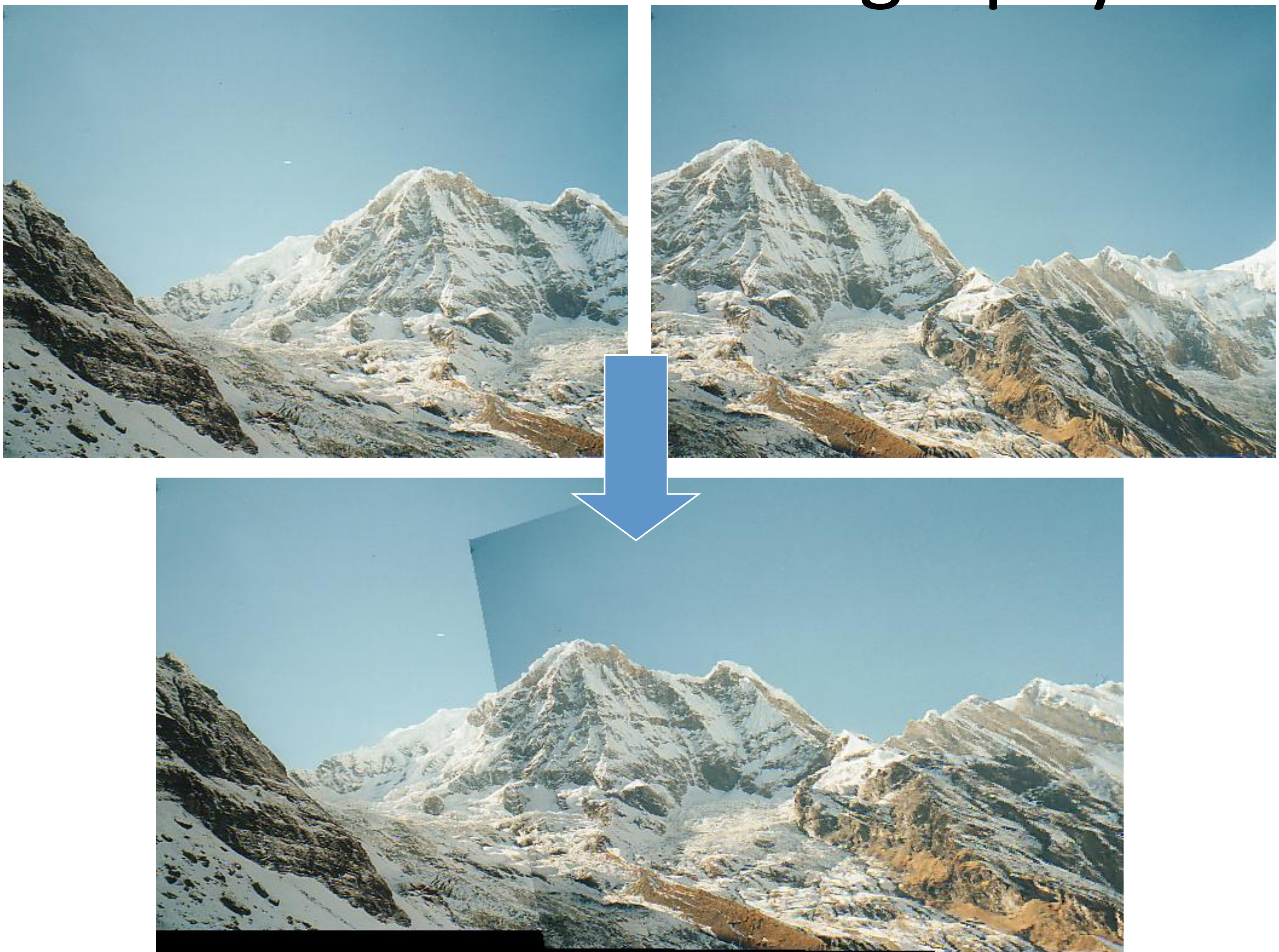
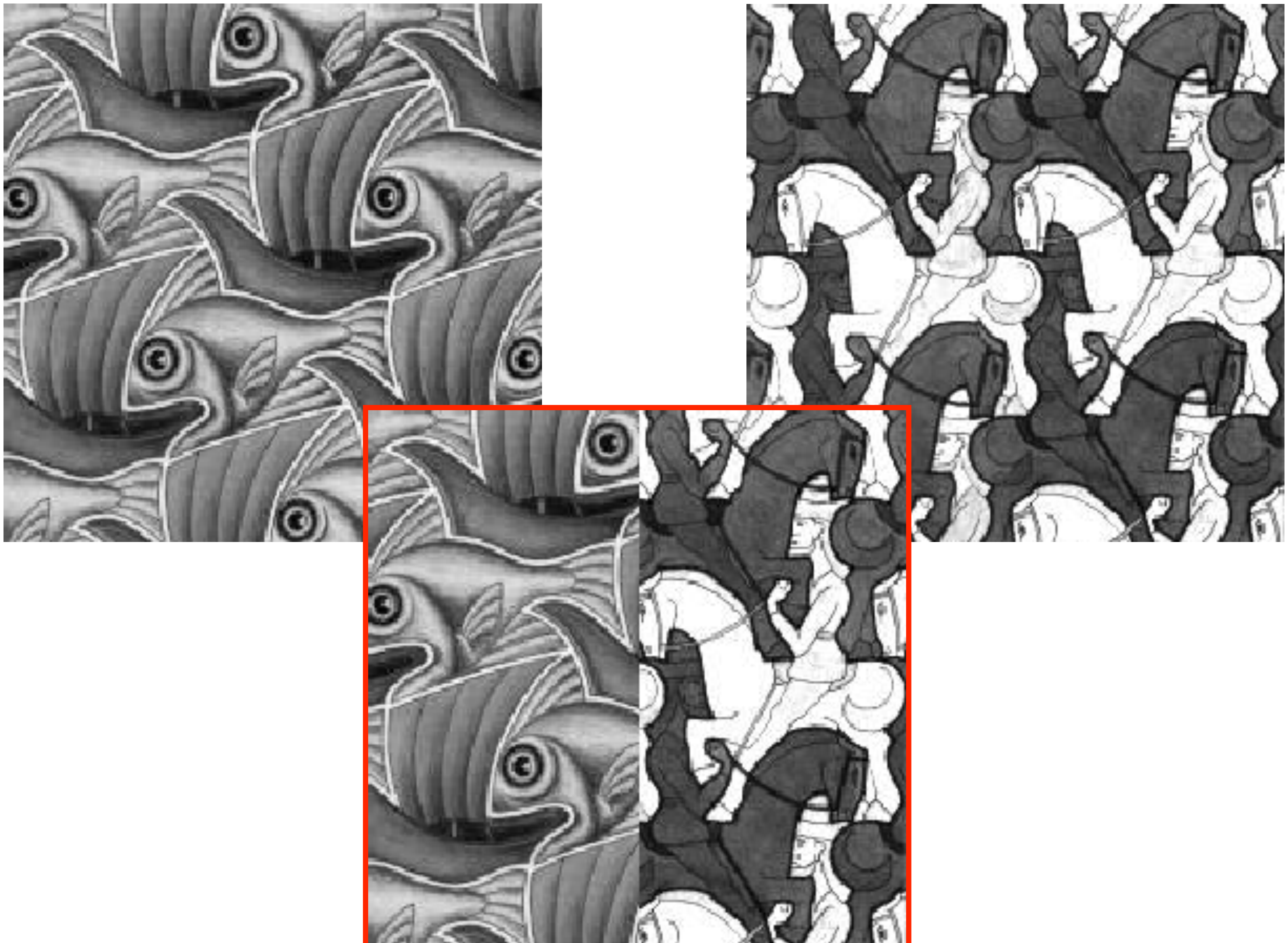
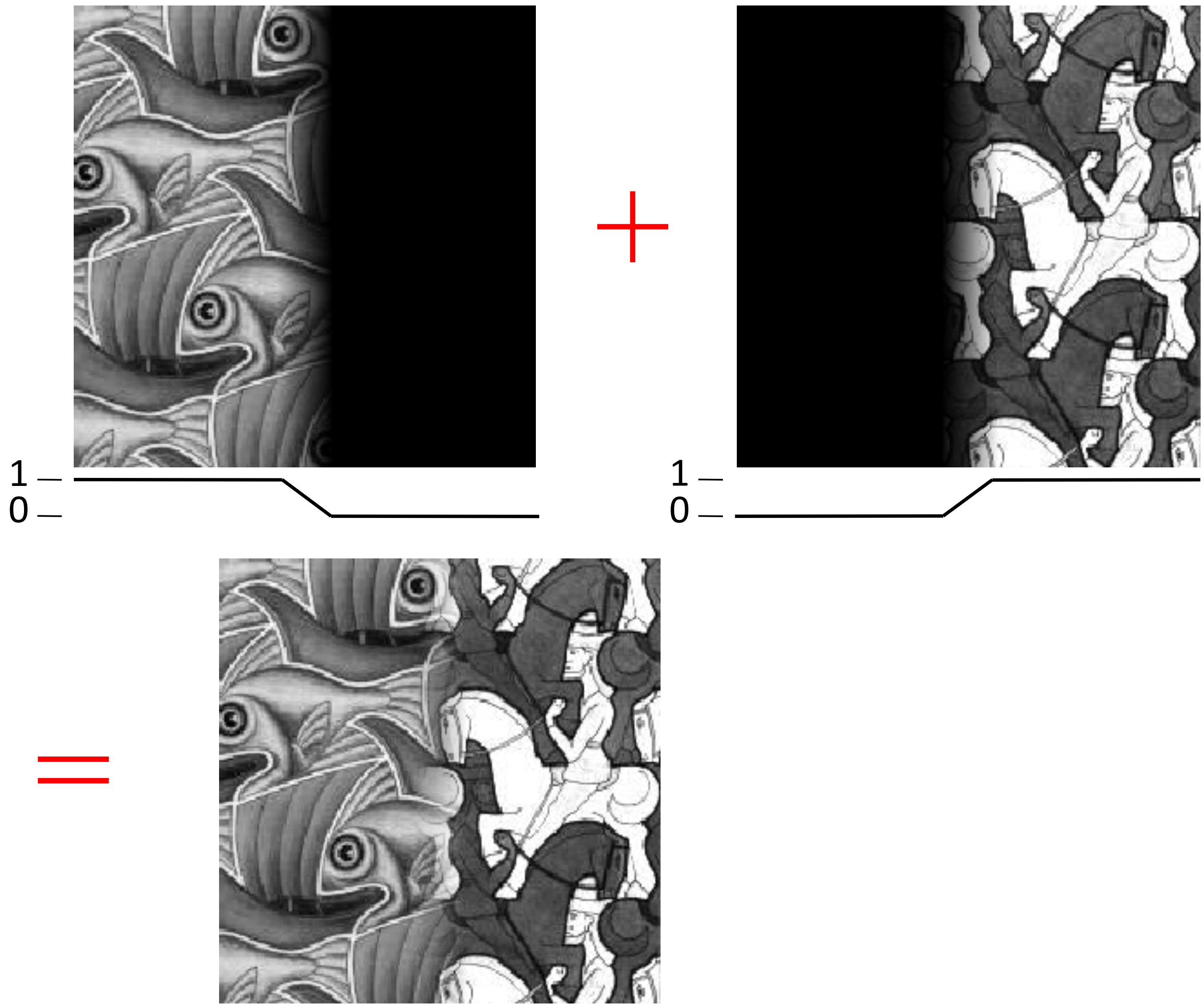


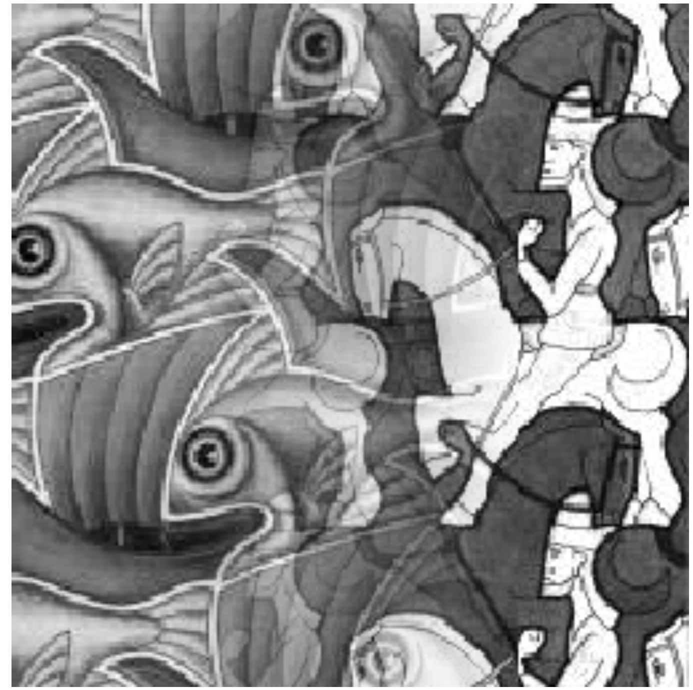
Image Blending

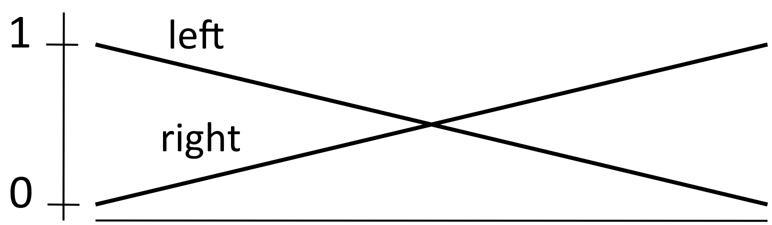


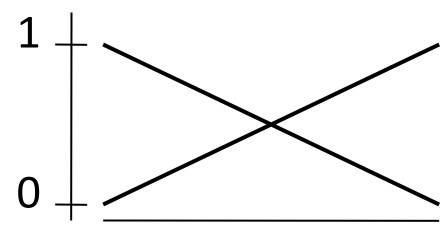
Feathering



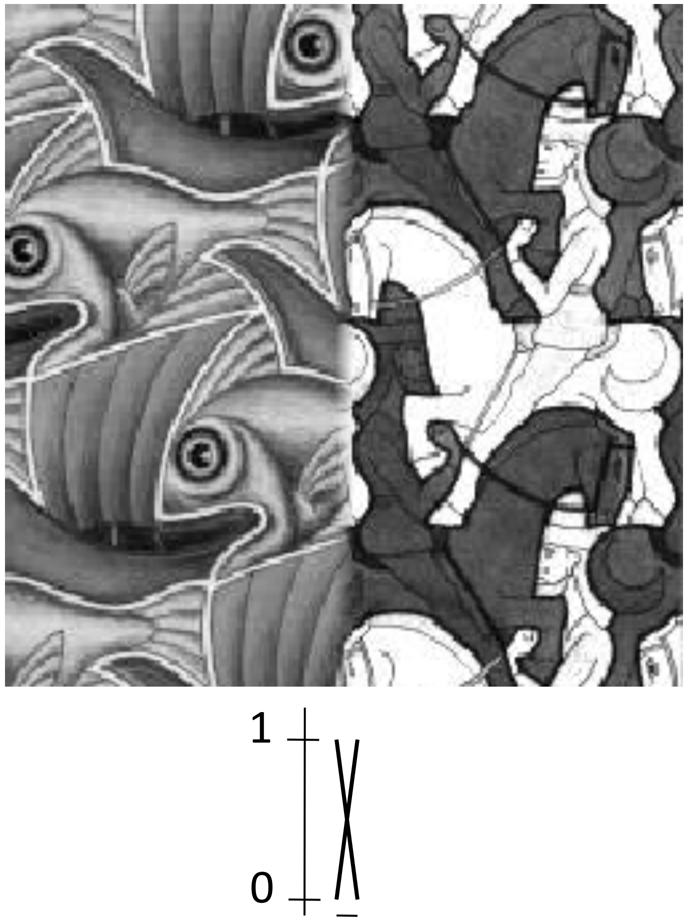
Effect of window (ramp-width) size

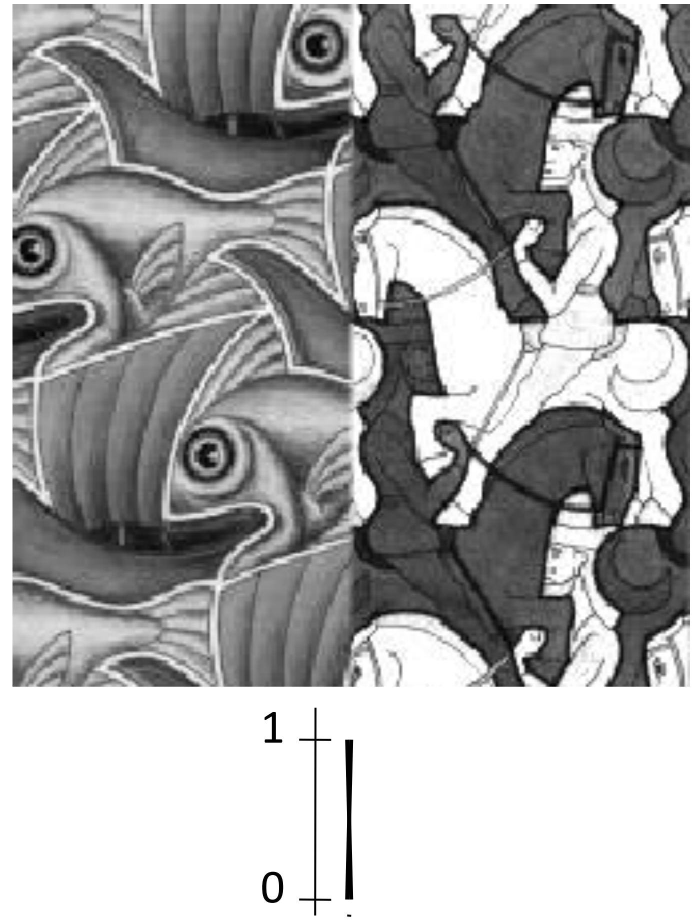




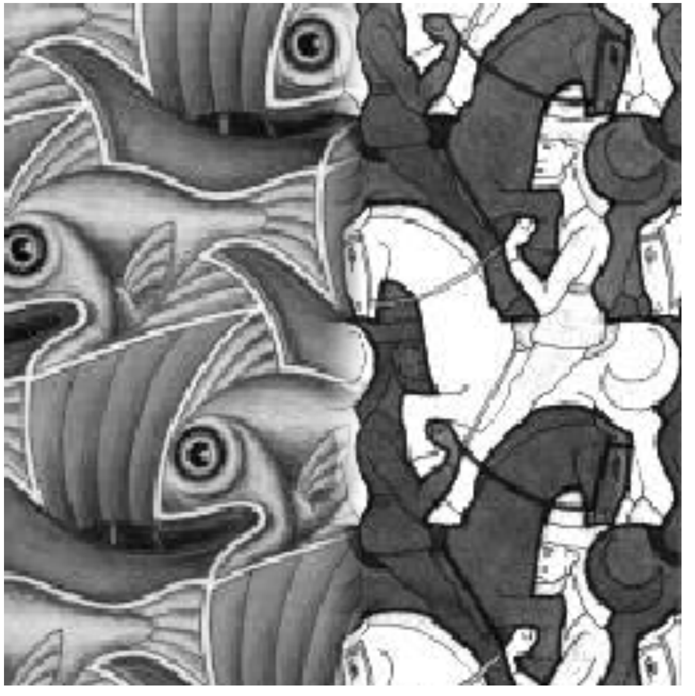


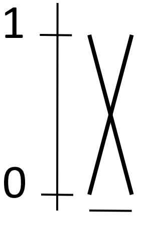
Effect of window size





Good window size

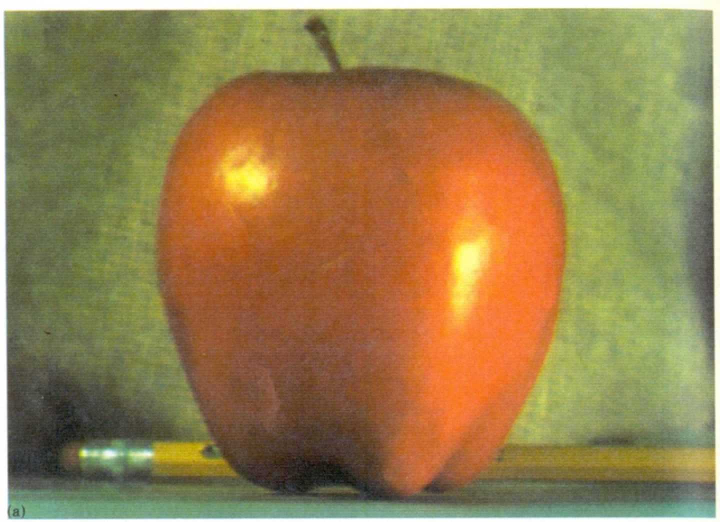


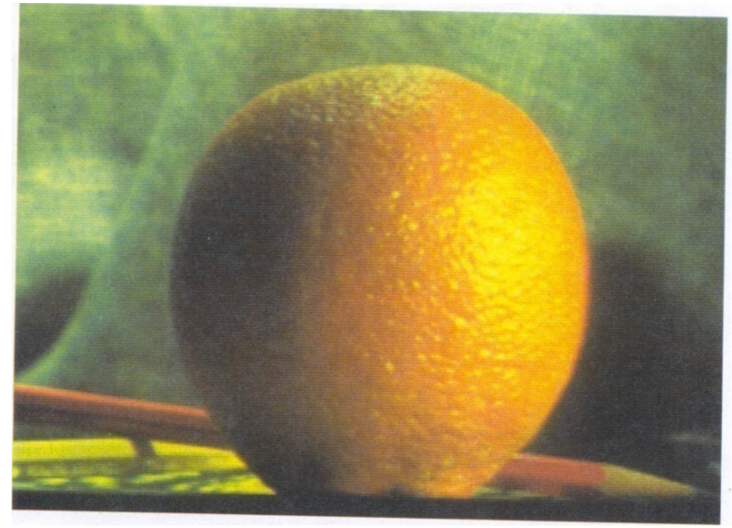


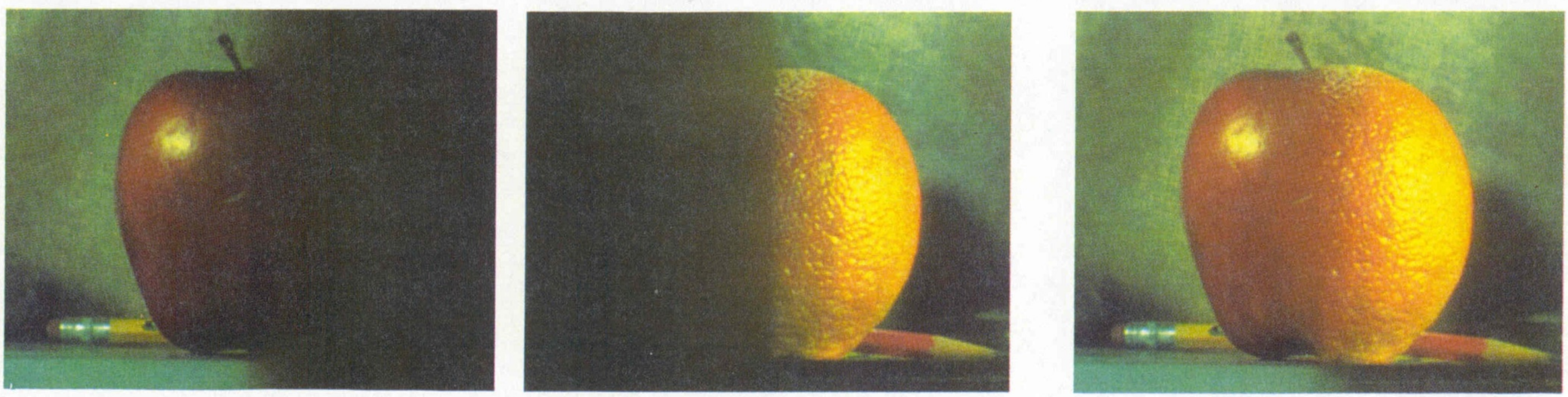
“Optimal” window: smooth but not ghosted

- Doesn’t always work...

Pyramid blending







(d) (h) (l)

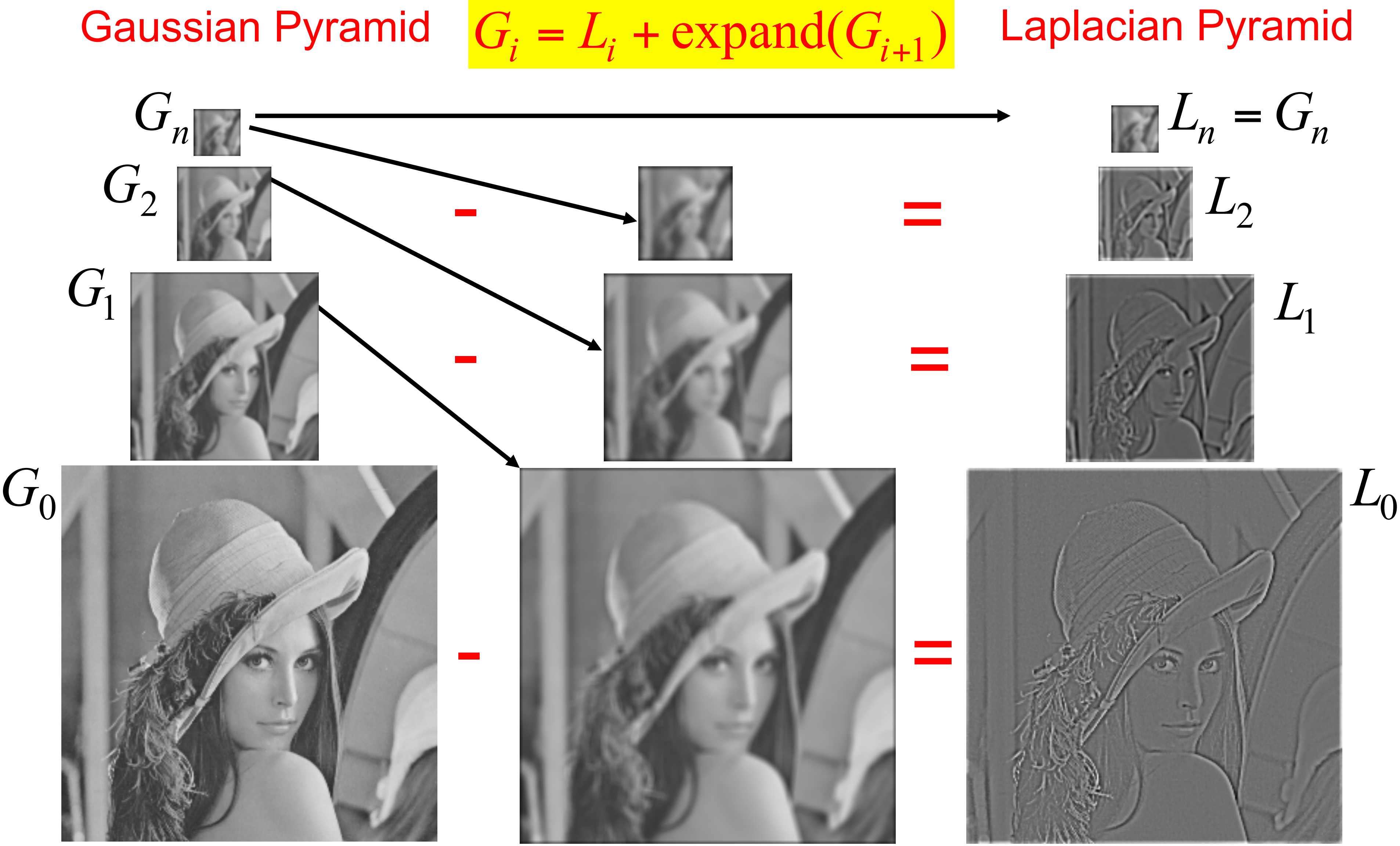
Create a Laplacian pyramid, blend each level

- 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.

The Laplacian Pyramid

$$L_i = G_i - \text{expand}(G_{i+1})$$

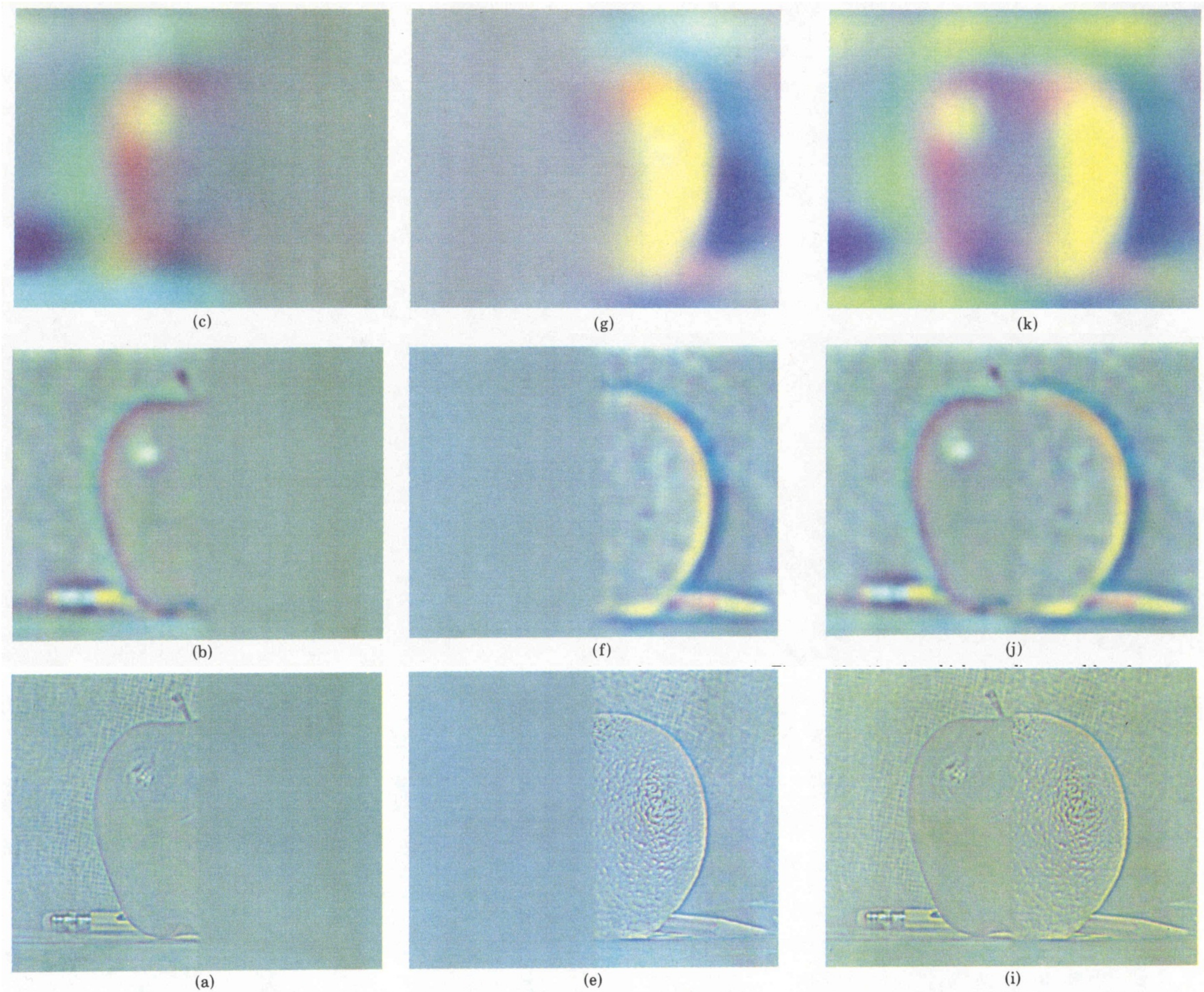
$$G_i = L_i + \text{expand}(G_{i+1})$$



Laplacian level 4

Laplacian level 2

Laplacian level 0



left pyramid

right pyramid

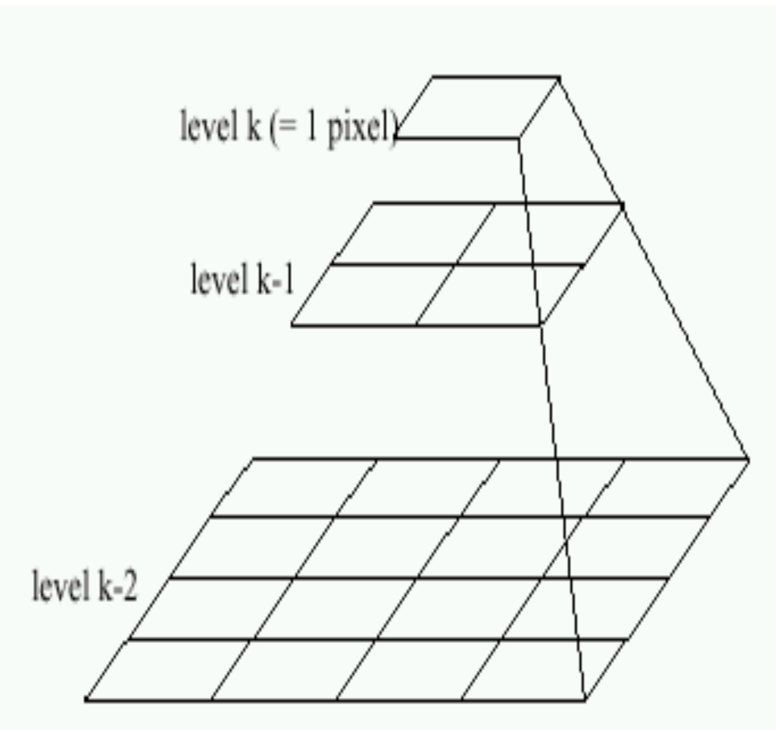
blended pyramid

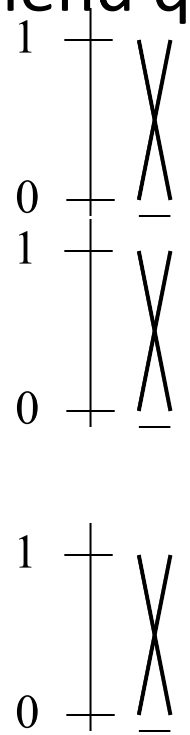
Laplacian image blend

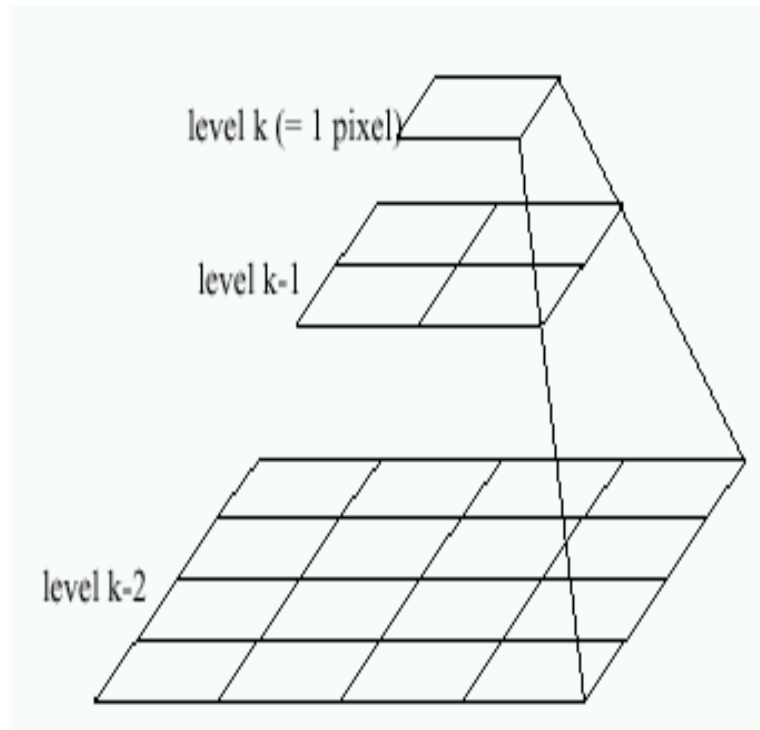
1. Compute Laplacian pyramid
2. Compute Gaussian pyramid on *weight* image
3. Blend Laplacians using Gaussian blurred weights
4. Reconstruct the final image

Multiband Blending with Laplacian Pyramid

- At low frequencies, blend slowly
- At high frequencies, blend quickly







Left pyramid

blend

Right pyramid

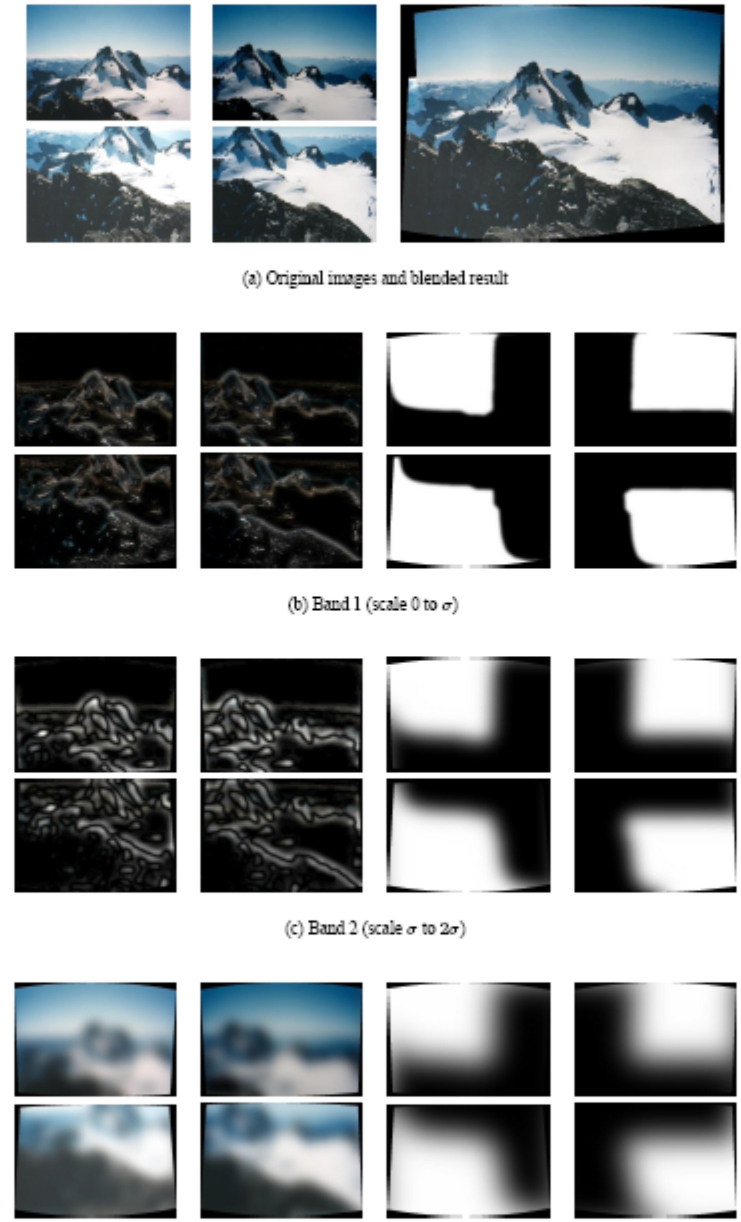
Multiband blending

Laplacian pyramids

1. Compute Laplacian pyramid of images and mask

2. Create blended image at each level of pyramid

3. Reconstruct complete image



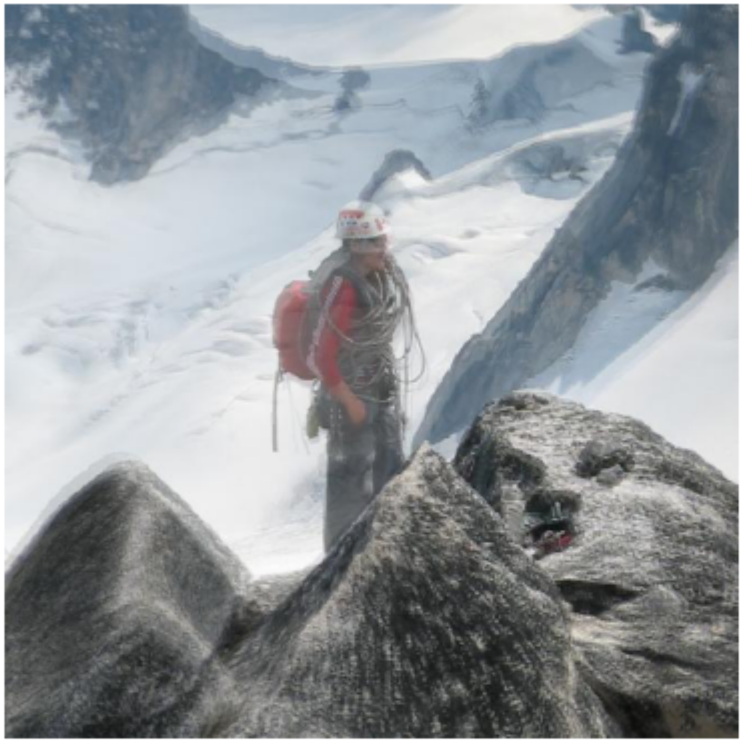
(a) Original images and blended result

(b) Band 1 (scale 0 to σ)

(c) Band 2 (scale σ to 2σ)

(d) Band 3 (scale lower than 2σ)

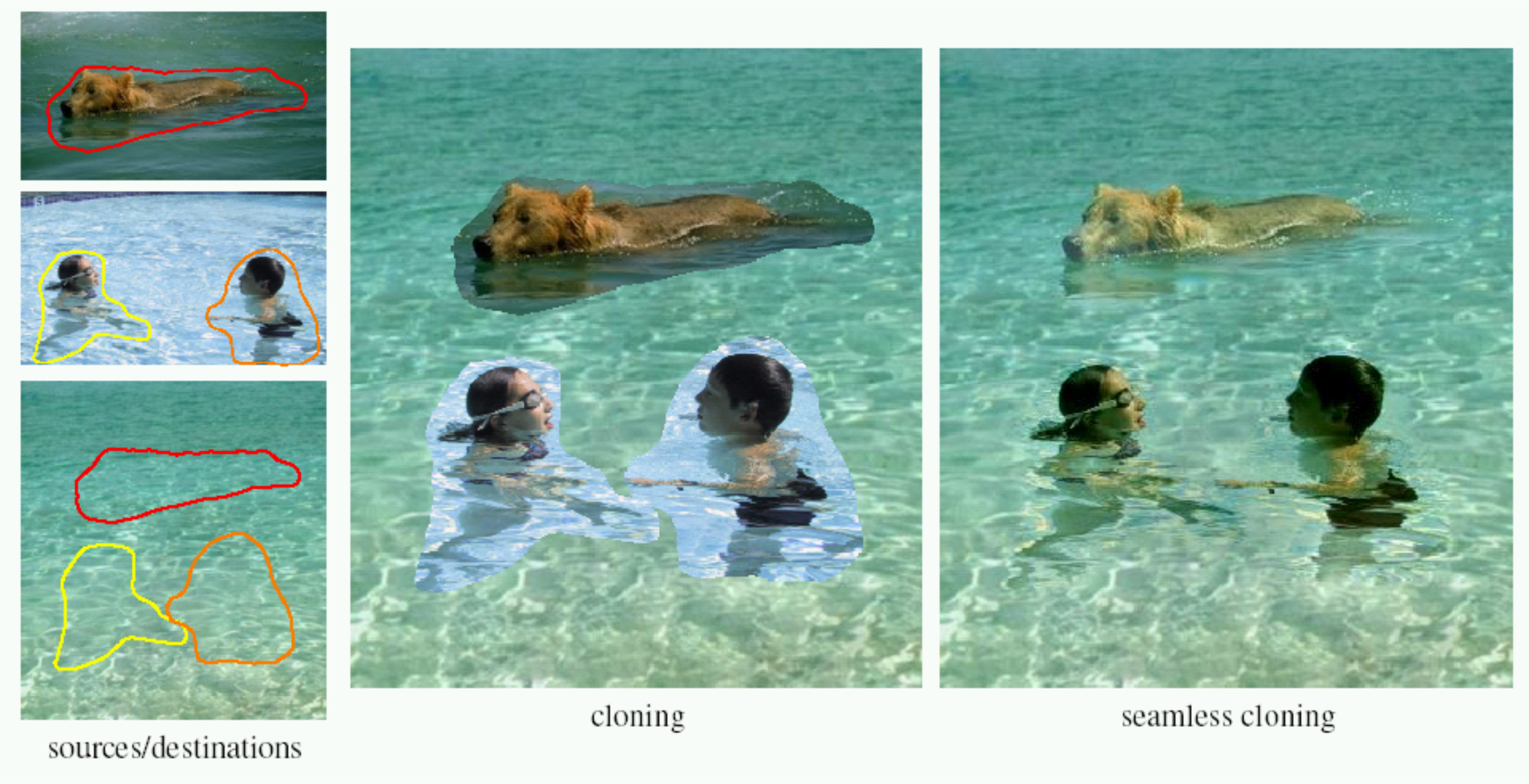
Blending comparison (IJCV 2007)



(a) Linear blending

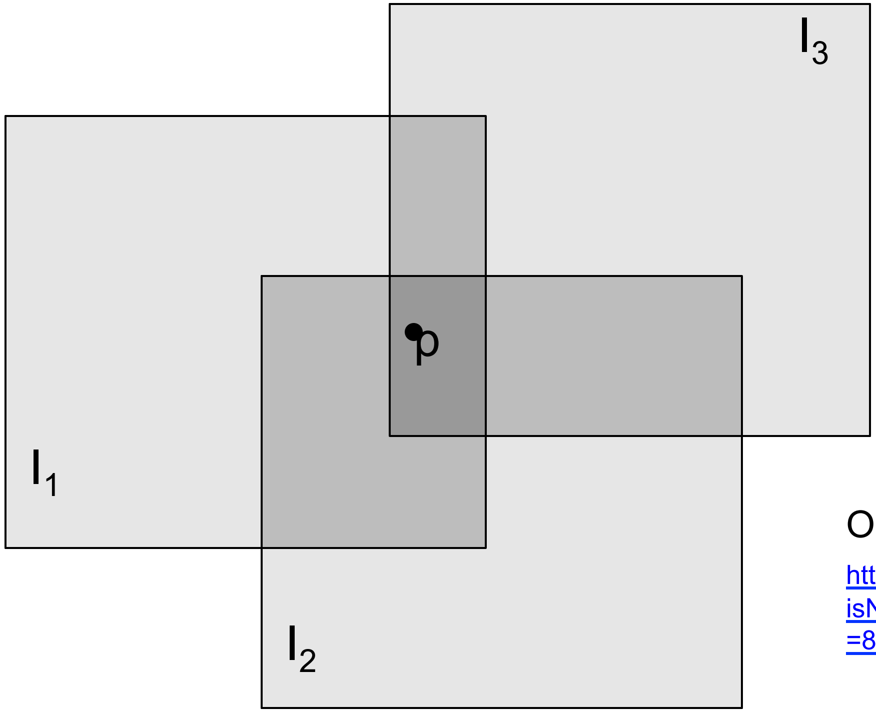
(b) Multi-band blending

Poisson Image Editing



- For more info: Perez et al, SIGGRAPH 2003
  - http://research.microsoft.com/vision/cambridge/papers/perez_siggraph03.pdf

Alpha Blending



Optional: see Blinn (CGA, 1994) for details:
http://ieeexplore.ieee.org/iel1/38/7531/00310740.pdf?isNumber=7531&prod=JNL&arnumber=310740&arSt=83&ared=87&arAuthor=Blinn%2C+J.F.

Encoding blend weights: $I(x,y) = (\alpha R, \alpha G, \alpha B, \alpha)$

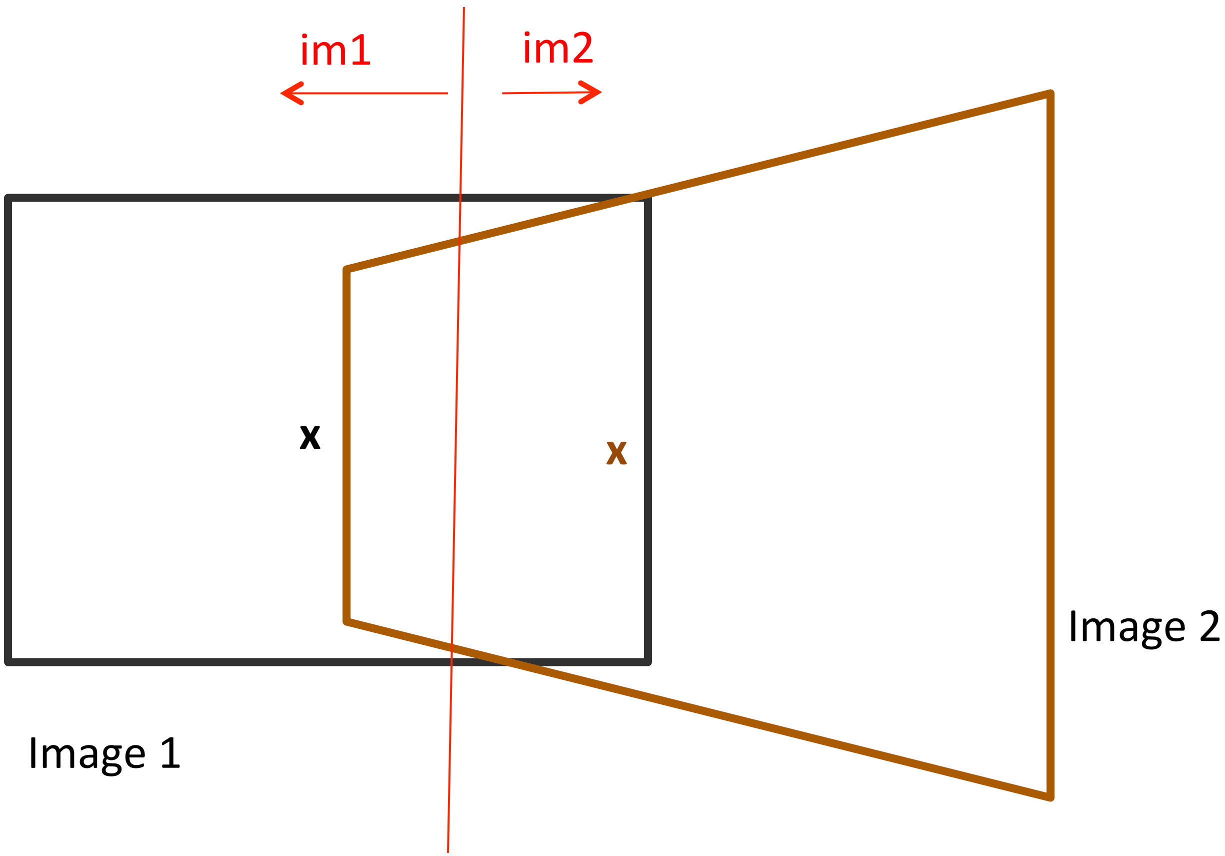
color at p = $$\frac{(\alpha_1 R_1,\ \alpha_1 G_1,\ \alpha_1 B_1) + (\alpha_2 R_2,\ \alpha_2 G_2,\ \alpha_2 B_2) + (\alpha_3 R_3,\ \alpha_3 G_3,\ \alpha_3 B_3)}{\alpha_1 + \alpha_2 + \alpha_3}$$

Implement this in two steps:

1. accumulate: add up the (α premultiplied) RGBα values at each pixel
2. normalize: divide each pixel's accumulated RGB by its α value

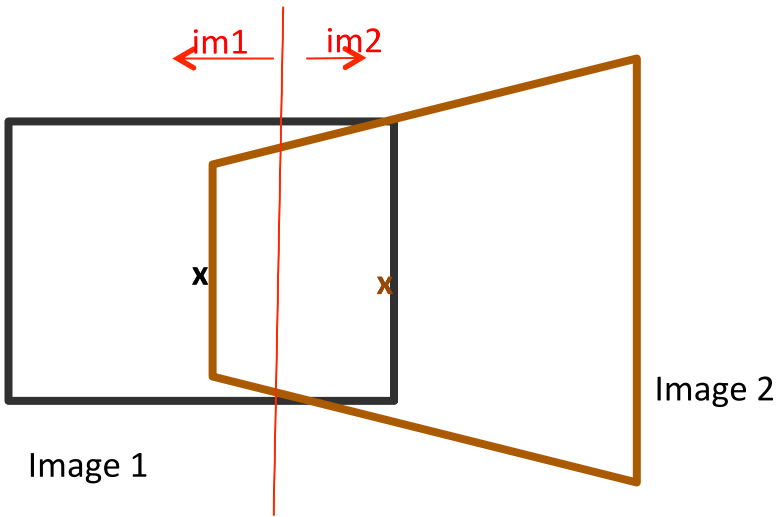
Choosing seams

- Easy method
  - Assign each pixel to image with nearest center



Choosing seams

- Easy method
  - Assign each pixel to image with nearest center
  - Create a mask:
  - Smooth boundaries (“feathering”):
  - Composite



Choosing seams

- Better method: dynamic program to find seam along well-matched regions

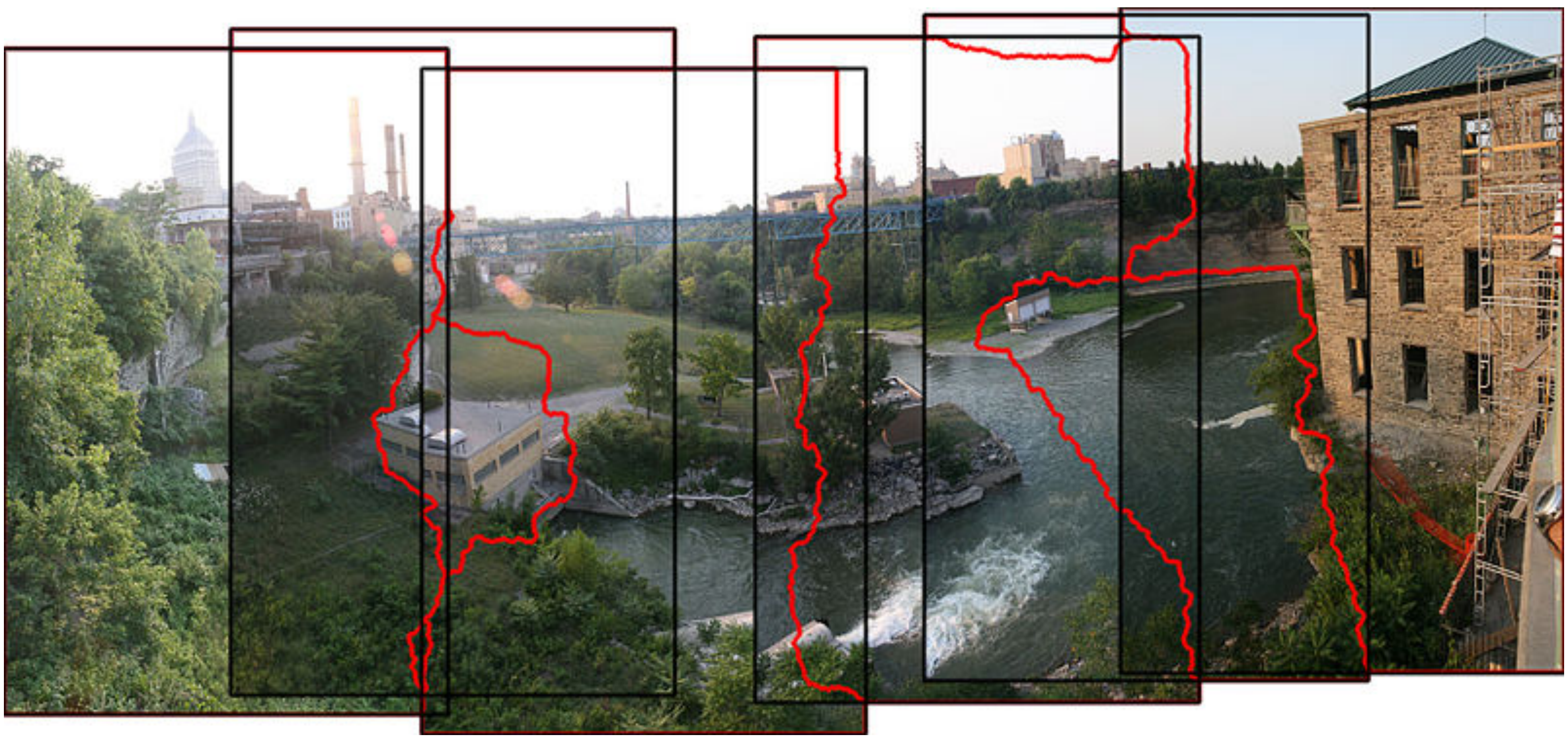
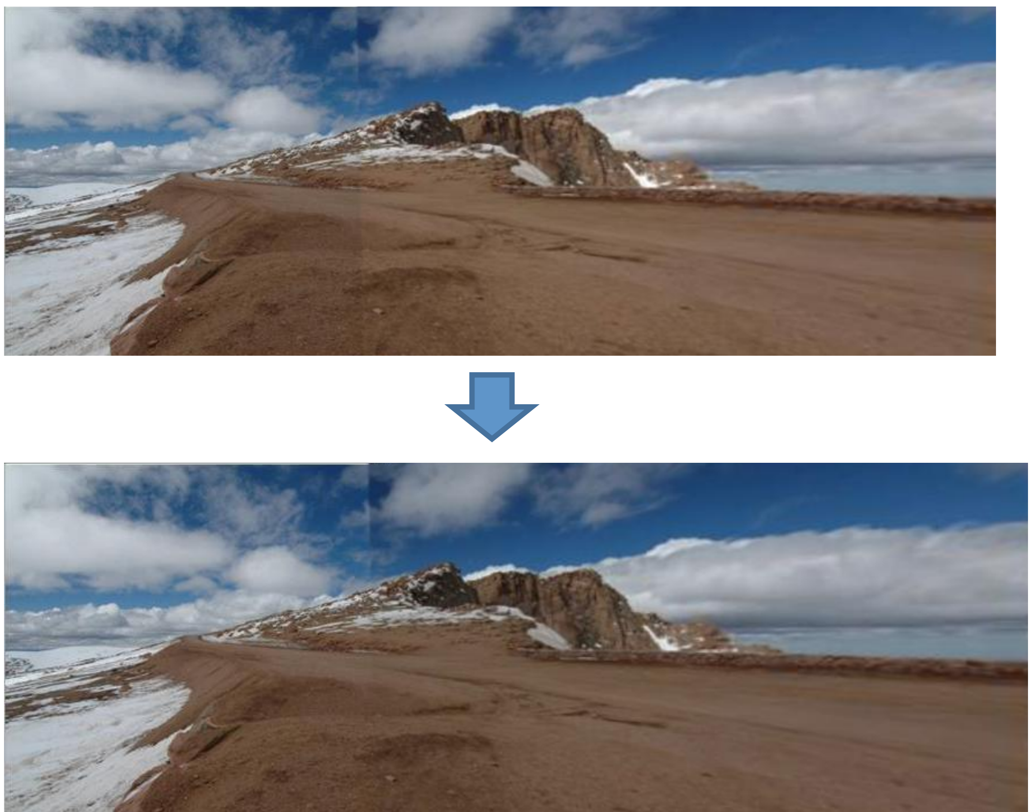
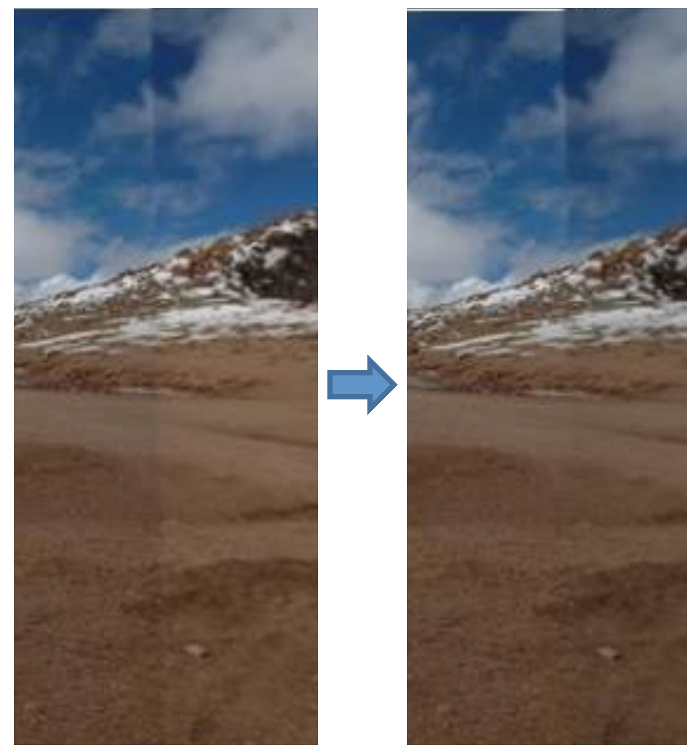


Illustration: http://en.wikipedia.org/wiki/File:Rochester_NY.jpg

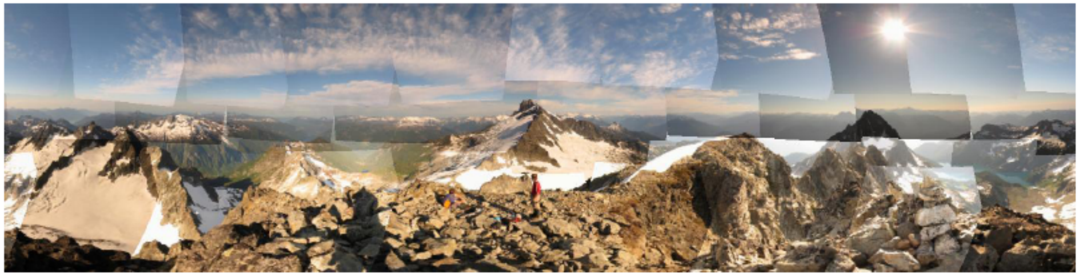
Gain compensation

- Simple gain adjustment
  - Compute average RGB intensity of each image in overlapping region
  - Normalize intensities by ratio of averages



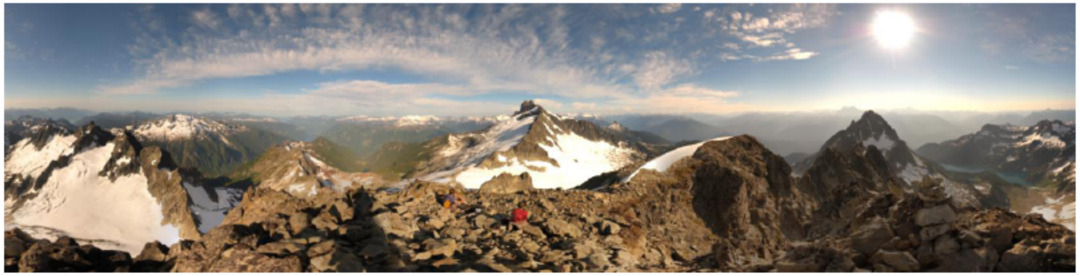


Blending Comparison



(b) Without gain compensation

(c) With gain compensation



(d) With gain compensation and multi-band blending

Recognizing Panoramas

Some of following material from Brown and Lowe 2003 talk

Brown and Lowe 2003, 2007

Recognizing Panoramas

Input: N images

1. Extract SIFT points, descriptors from all images
2. Find K-nearest neighbors for each point (K=4)
3. For each image
   a) Select M candidate matching images by counting matched keypoints (m=6)
   b) Solve homography $\mathbf{H}_{ij}$ for each matched image

Recognizing Panoramas

Input: N images

1. Extract SIFT points, descriptors from all images
2. Find K-nearest neighbors for each point (K=4)
3. For each image
   a) Select M candidate matching images by counting matched keypoints (m=6)
   b) Solve homography $\mathbf{H}_{ij}$ for each matched image
   c) Decide if match is valid ($n_i > 8 + 0.3\, n_f$)

\# inliers

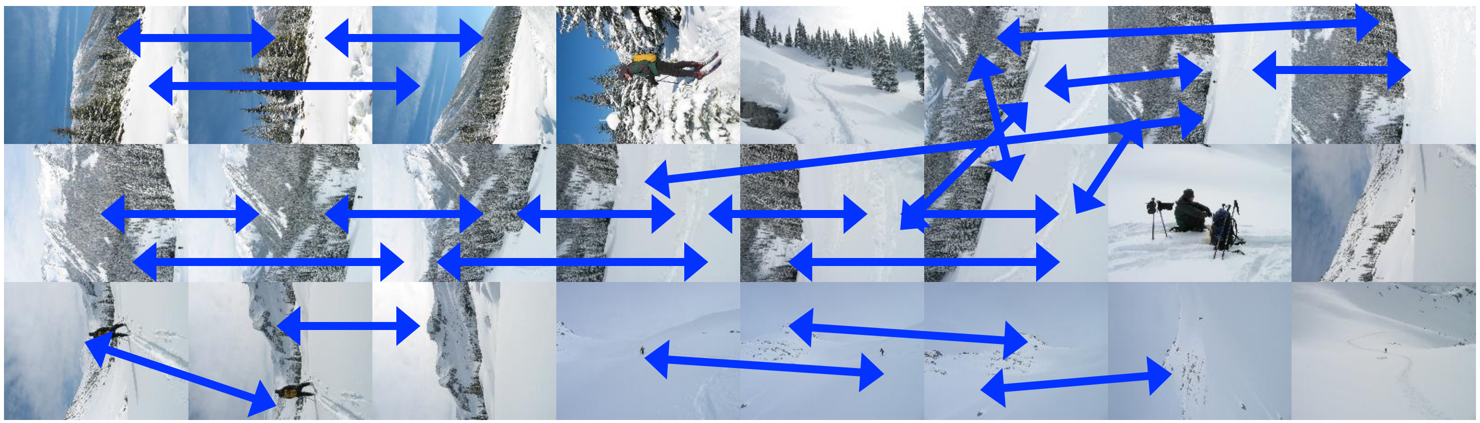
\# keypoints in overlapping area

Recognizing Panoramas (cont.)

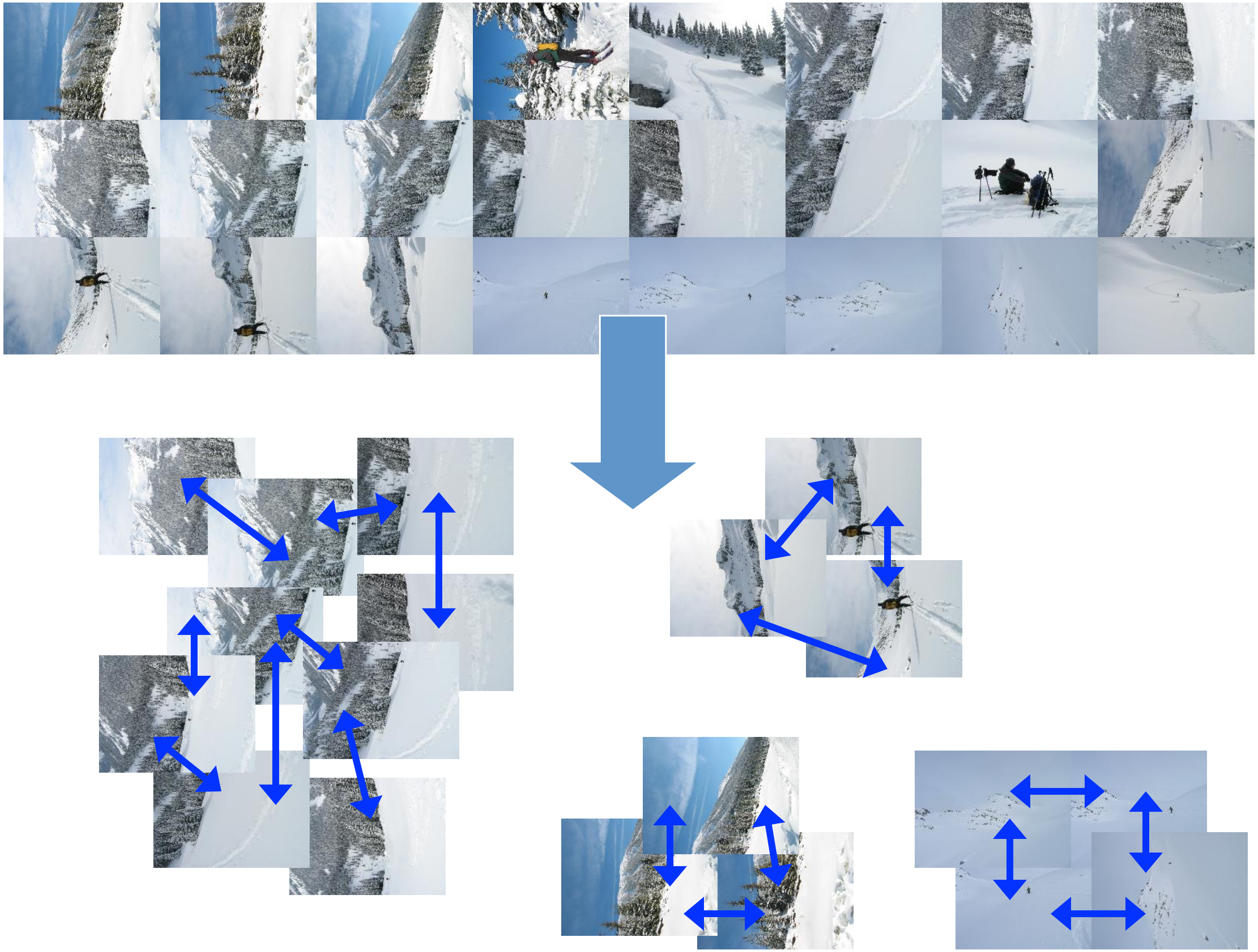
(now we have matched pairs of images)

4. Find connected components

Finding the panoramas



Finding the panoramas



Finding the panoramas

Recognizing Panoramas (cont.)

(now we have matched pairs of images)

4. Find connected components
5. For each connected component
   a) Solve for rotation and f
   b) Project to a surface (plane, cylinder, or sphere)
   c) Render with multiband blending