Image Alignment

Dr. Randy C. Hoover

Department of Electrical and Computer Engineering
South Dakota School of Mines and Technology
Rapid City, SD 57701, USA

November 5, 2012

Review

- Last few lectures:
  - Finding distinctive features in multiple images
  - Describing said features using a feature descriptor
  - Corresponding each descriptor for alignment

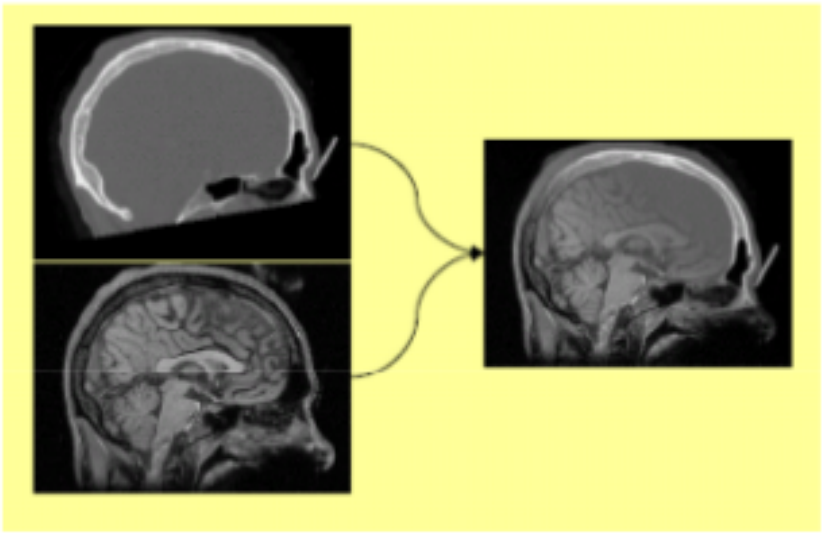
Today (and next few lectures)

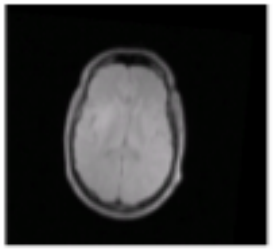
- Image alignment
  - Computing homographies
  - Rotational panoramics
  - RANSAC
  - Global alignment
  - Warping
  - Blending

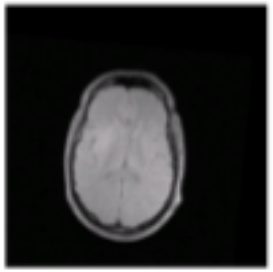
Motivation: Recognition in cluttered scenes

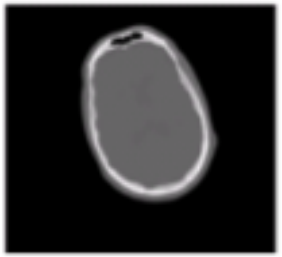
Figures from David Lowe

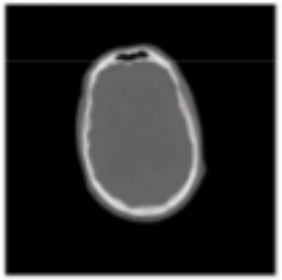
Motivation: Image registration

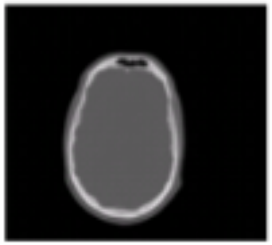












Motivation: Mosaics

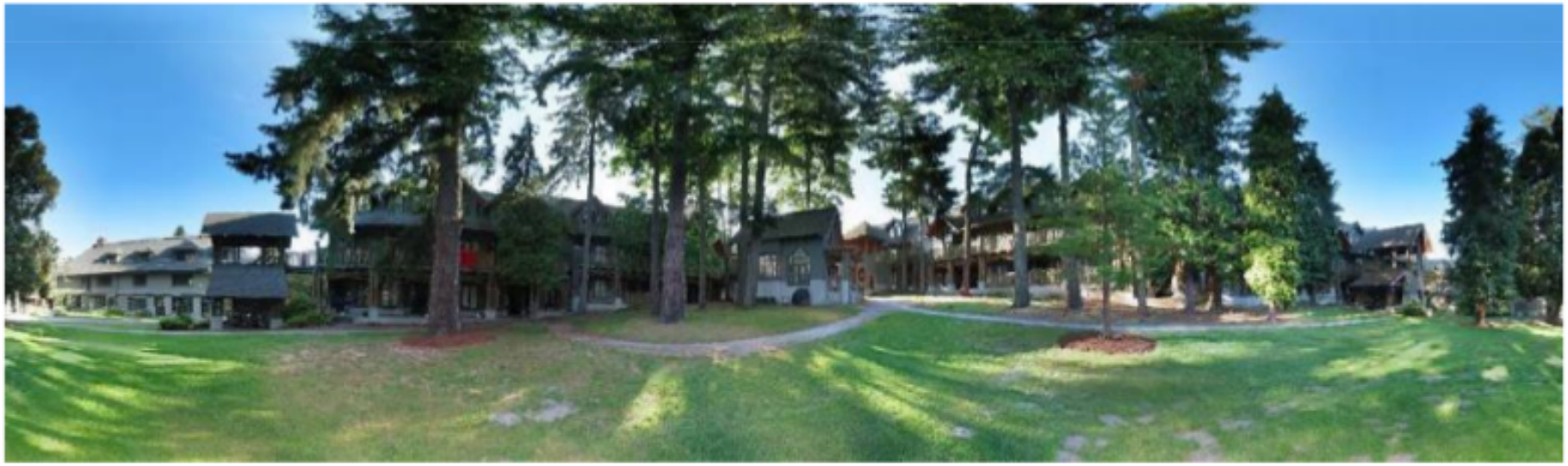
- Getting the whole picture
  - Consumer camera: $50^\circ \times 35^\circ$

Motivation: Mosaics

- Getting the whole picture
  - Consumer camera: $50^\circ \times 35^\circ$
  - Human vision: $176^\circ \times 135^\circ$

Motivation: Mosaics

- Getting the whole picture
  - Consumer camera: $50^\circ \times 35^\circ$
  - Human vision: $176^\circ \times 135^\circ$



- Panoramic Mosaic = up to 360 x 180°

Slide from Brown & Lowe 2003

General idea

- Correspondence
  - Detect interest points
  - Describe interest points
  - Match two or more points

General idea

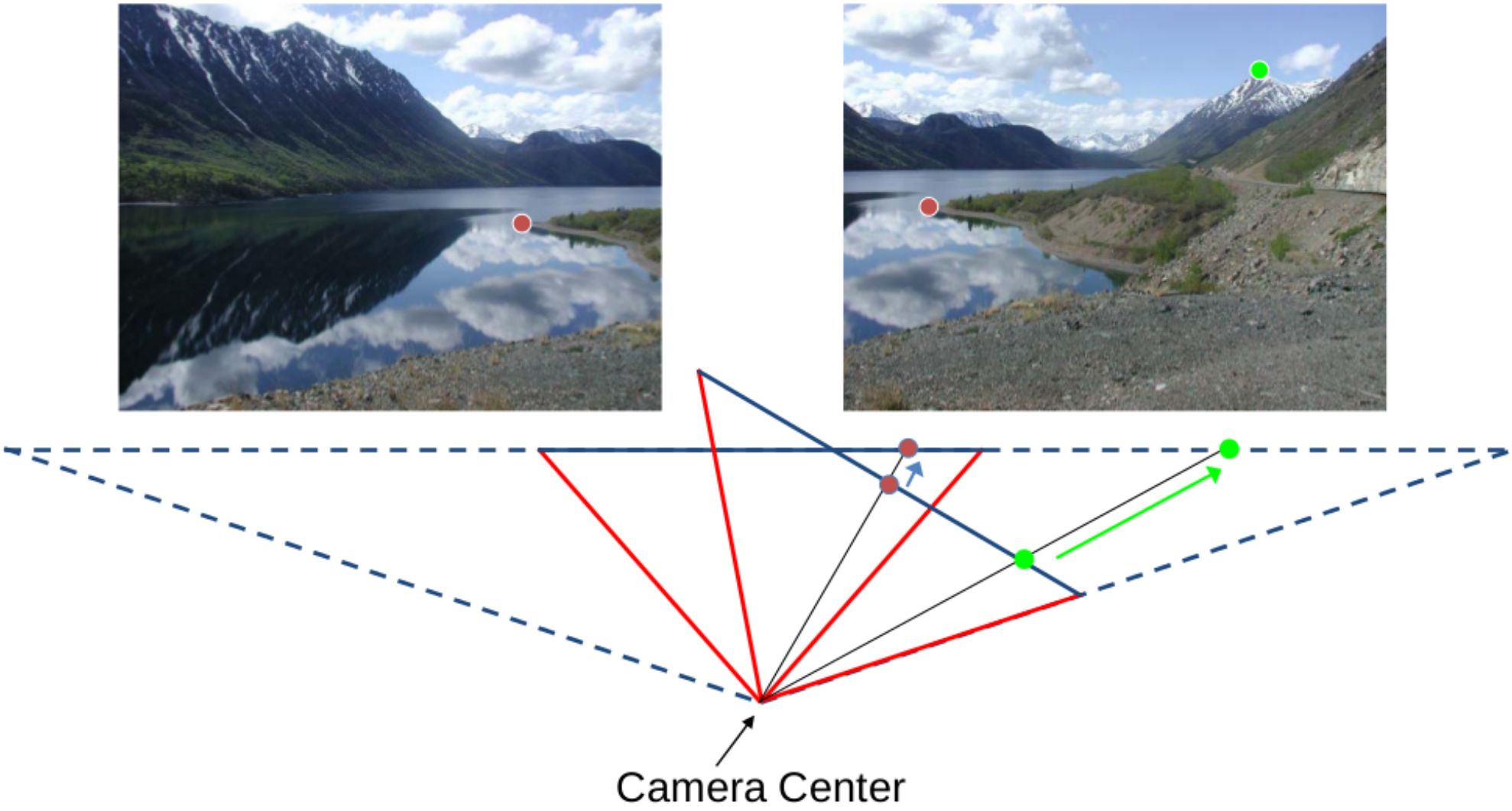
- Correspondence
  - Detect interest points
  - Describe interest points
  - Match two or more points

- Alignment and model fitting
  - Solve for motion or depth or camera movement from multiple matched points

General idea - image stitching

- Combine two or more overlapping images to make one larger one
- Cool HP vid. http://www.youtube.com/watch?v=2RPl5vPEoQk

Example



What types of motion can I get?

- What happens when we take two images from the same camera and try to align them?
  - Translation?
  - Rotation?
  - Scale changes?
  - Affine transformation?
  - Perspective?
  - Other....?

Recall: 2-D coordinate transformations

- Let $\mathbf{x} = (x, y)$
  - Translation: $\mathbf{x}' = \mathbf{x} + \mathbf{t}$
  - Rotation: $\mathbf{x}' = R\mathbf{x} + \mathbf{t}$, $R \in SO(2)$
  - Similarity: $\mathbf{x}' = \alpha R\mathbf{x} + \mathbf{t}$
  - Affine: $\mathbf{x}' = A\mathbf{x} + \mathbf{t}$
  - Perspective: $\bar{\mathbf{x}}' = H\bar{\mathbf{x}}$ where $\bar{\mathbf{x}}$ represents homogeneous coords: $\bar{\mathbf{x}} = (x, y, 1)$ and $H \in \mathbb{R}^{3\times 3}$

Recall: 2-D coordinate transformations

- We can write these as 3×3 matrices using homogeneous coords.

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

Translate

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

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

Shear

Recall: 2-D affine transformations

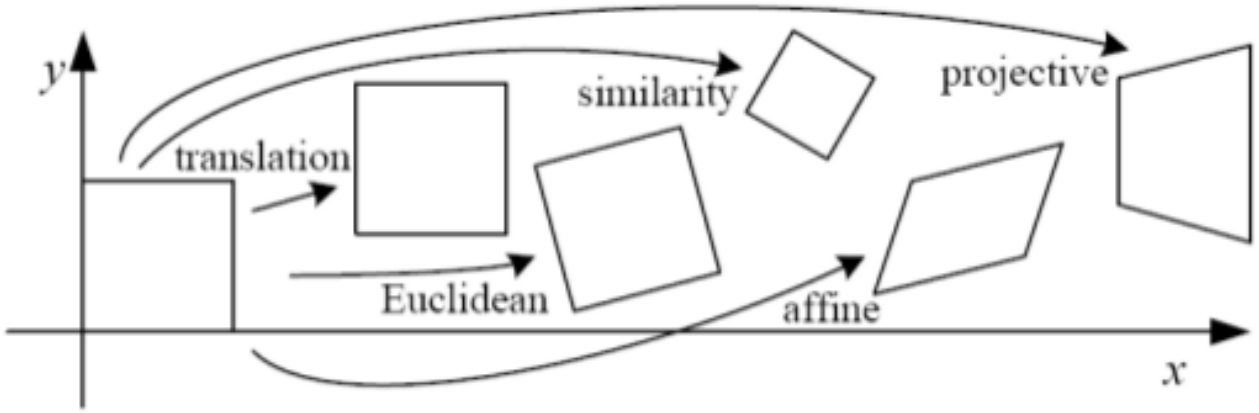
$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Affine transformations are a combination of:
  - Linear transformations, and
  - Translations
  - $\Longrightarrow$ parallel lines remain parallel

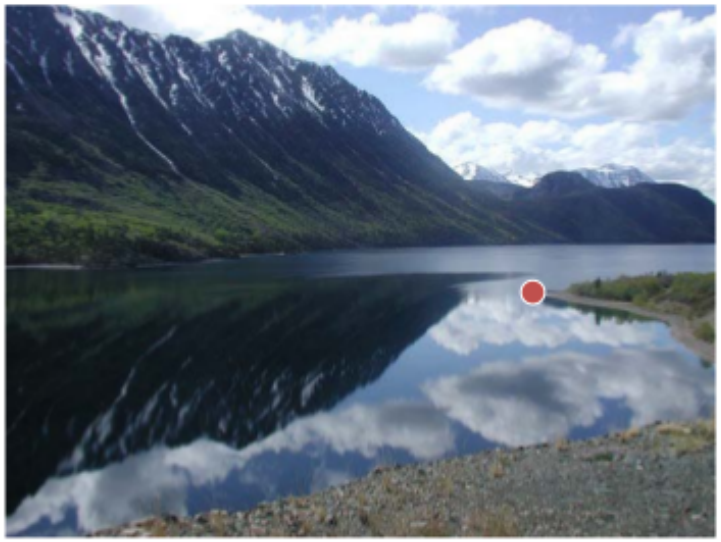
Recall: projective transformations

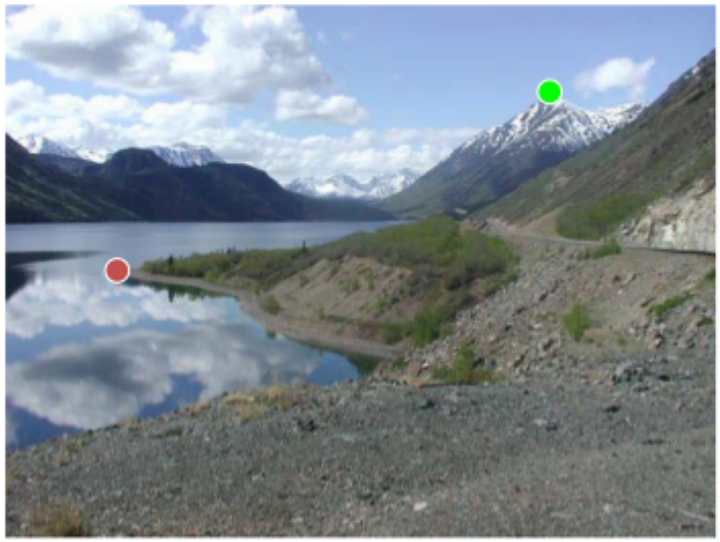
$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

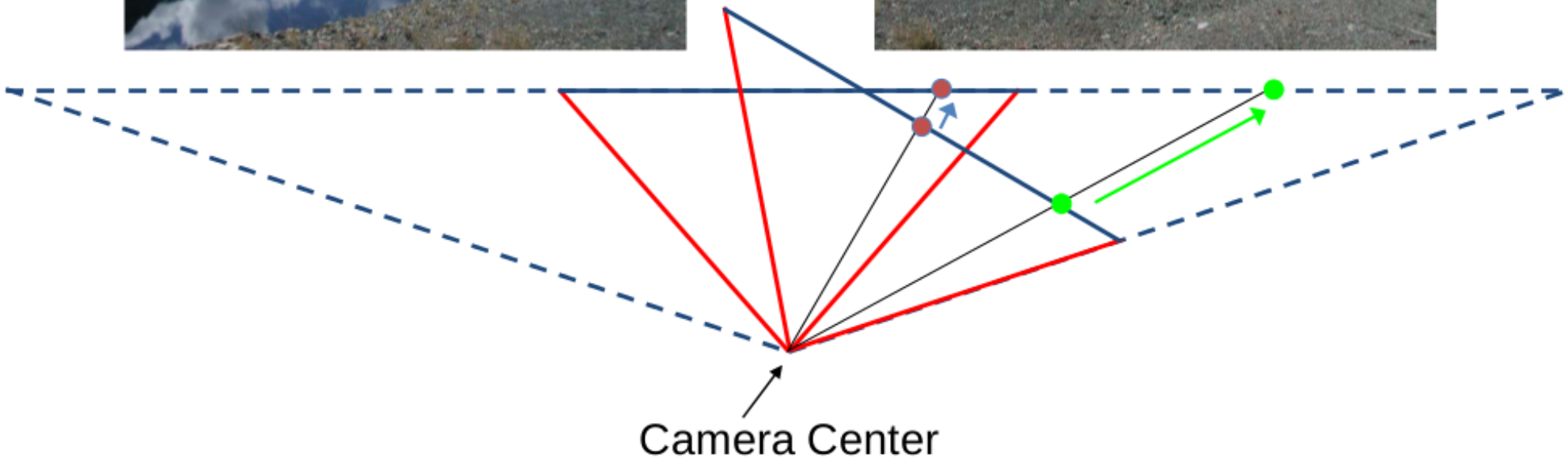
- Projective transformations are a combination of:
  - Affine transformations, and
  - Projective warping
  - $\Longrightarrow$ parallel lines **do not** remain parallel (in general)



Example - revisited







How do we analyze this?

- Review of projective geometry (in homogeneous coords)!
  - $\mathbf{x} = (\alpha R + \mathbf{t})\, X$
  - $\mathbf{x}' = (\alpha' R' + \mathbf{t}')\, X$
  - Assume camera rotates about its optical axis $\implies \mathbf{t} = \mathbf{t}' = 0$
  - This $\implies \mathbf{x}' = H\mathbf{x}$ where,
  - $H = \alpha' R' R^{-1} \alpha^{-1}$
- Typically only R and f, f' change (4-parameters), however, in general, H has 8-parameters since $H(3,3) = 1$

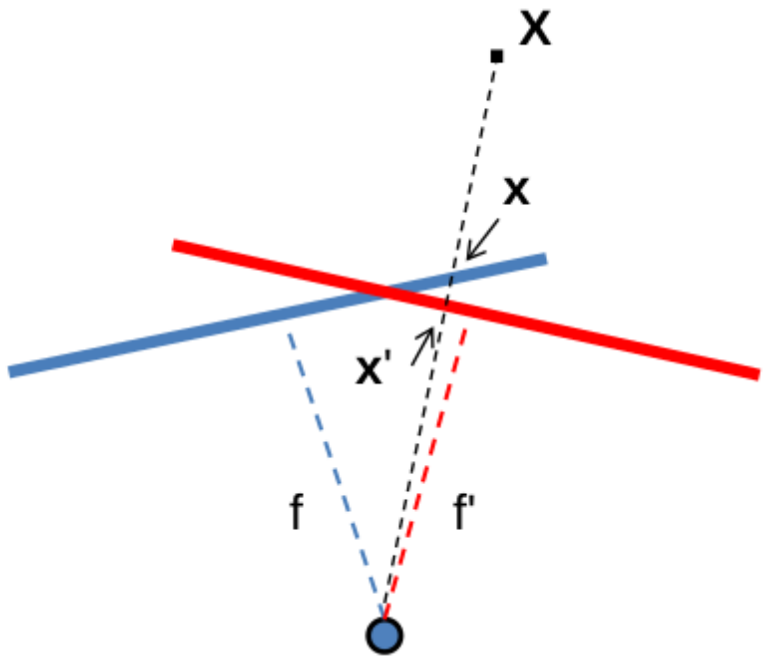
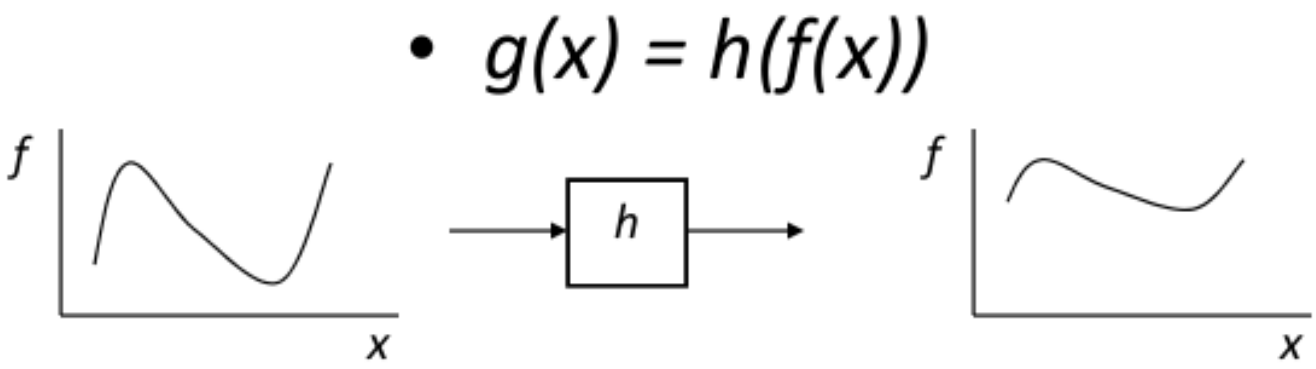


Image warping

- Image filtering: change the *range* of the image



- Image warping: change the *domain* of the image

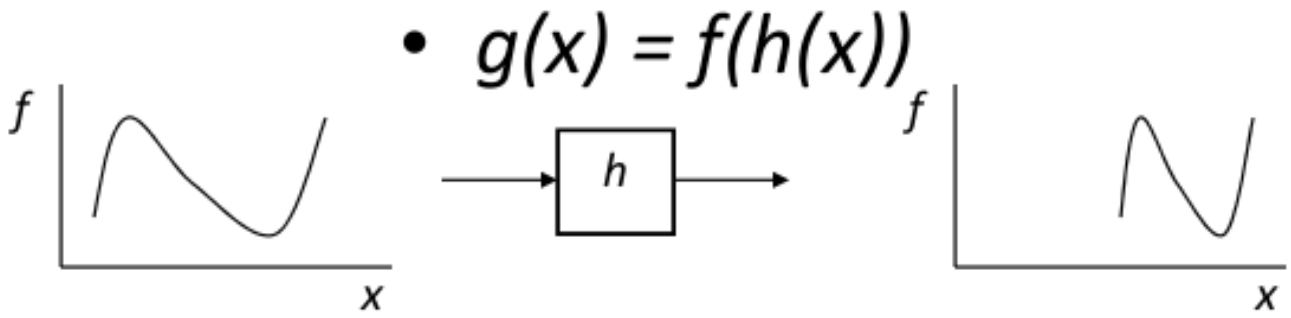
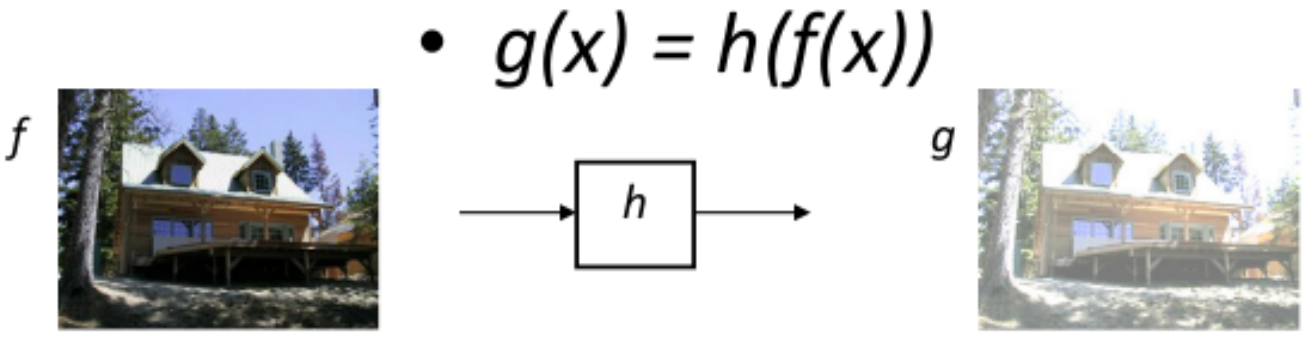
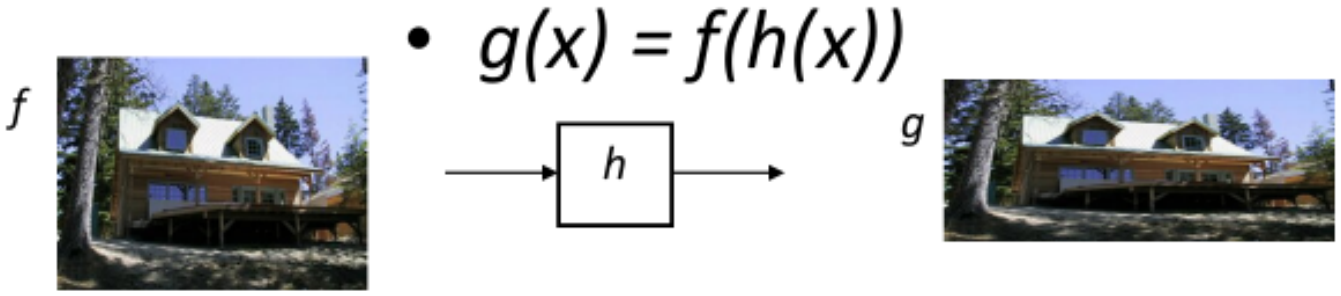


Image warping

- Image filtering: change the *range* of the image



- Image warping: change the *domain* of the image



Parametric (global) warping

- Examples of parametric warping:

translation

rotation

aspect

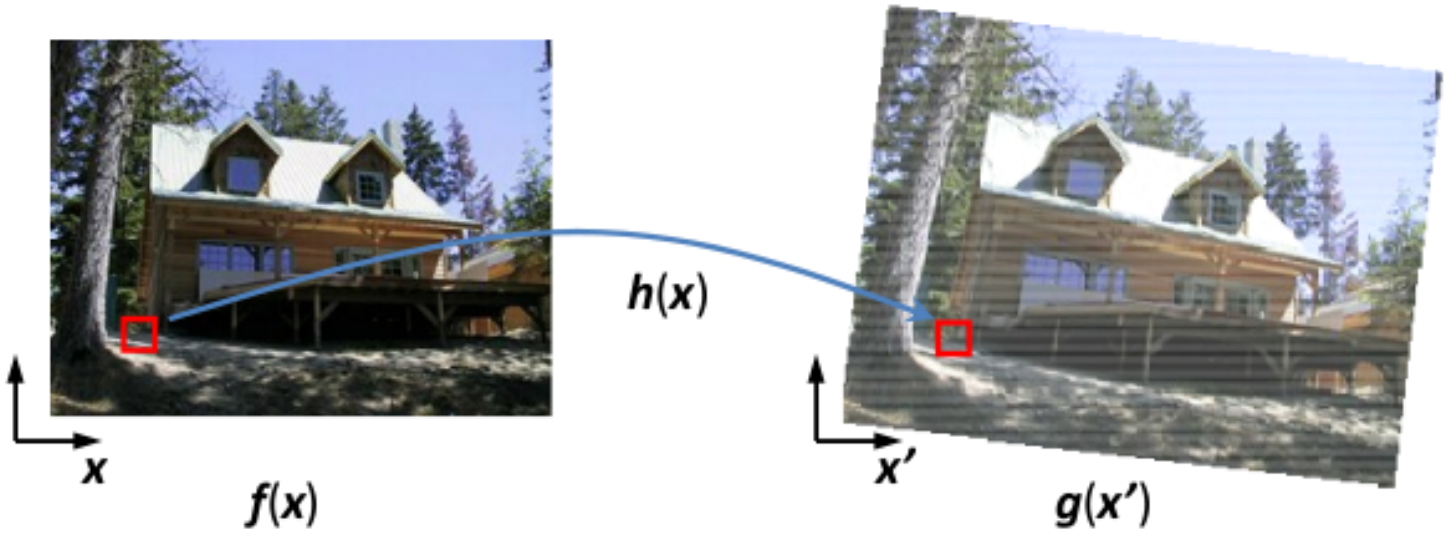
affine

perspective

cylindrical

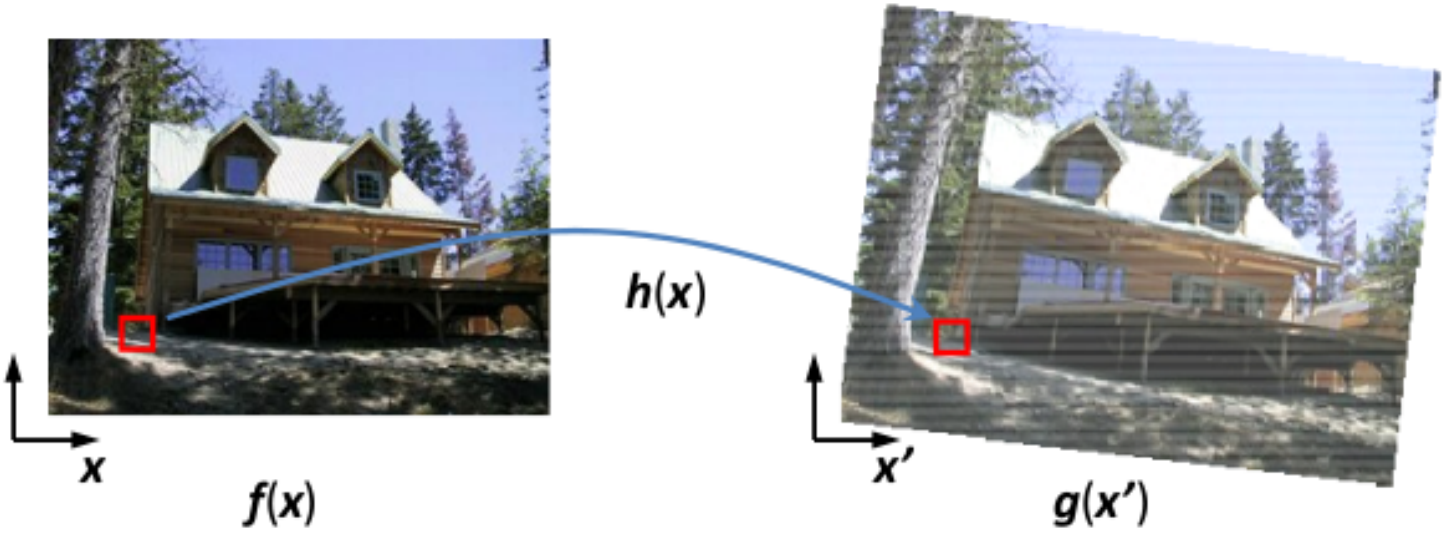
Image warping

- Given a coordinate transformation $x' = h(x)$ and a source image $f(x)$, how do we compute transformed image $g(x') = f(h(x))$?



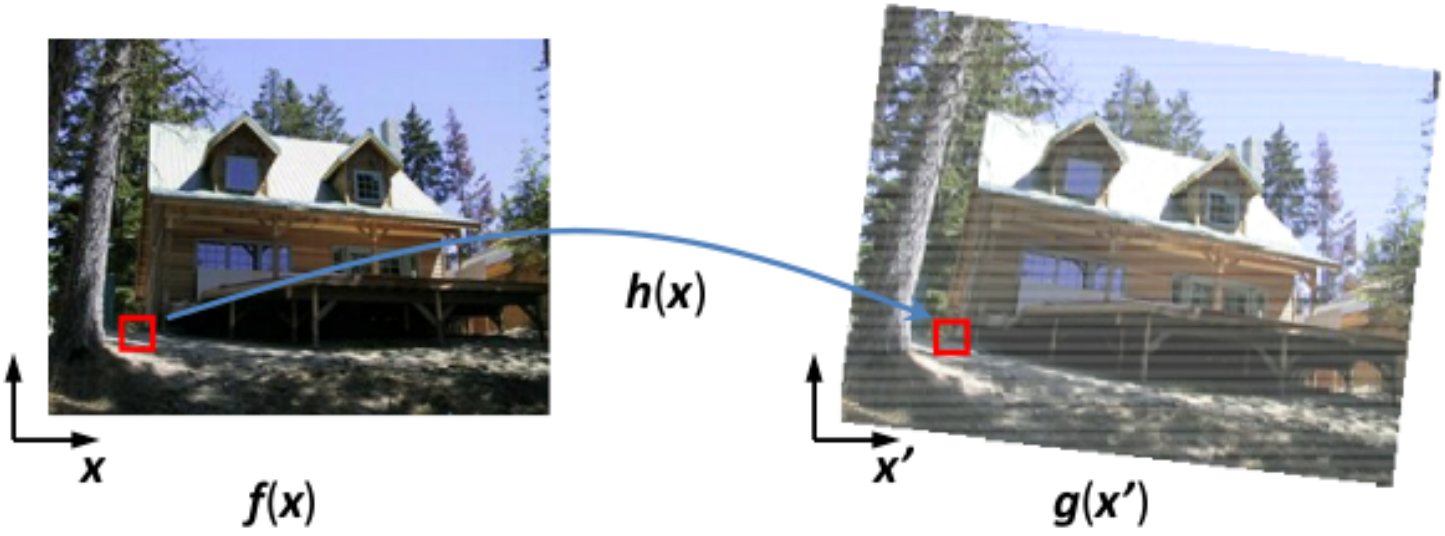
Forward warping

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



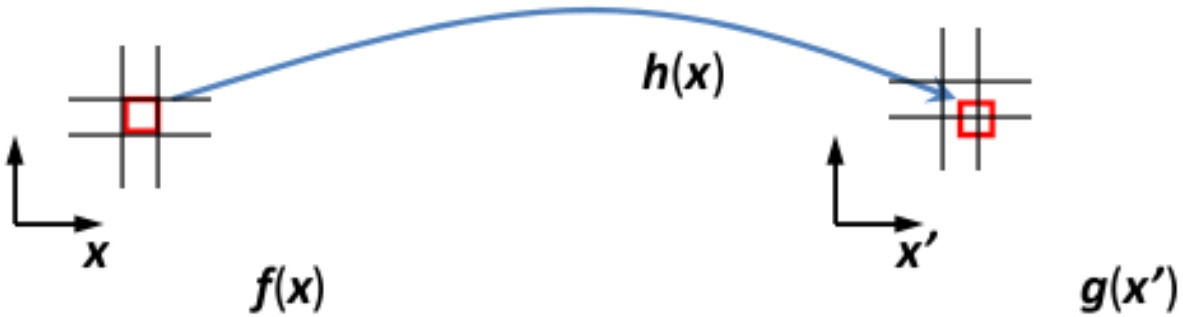
Forward warping

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



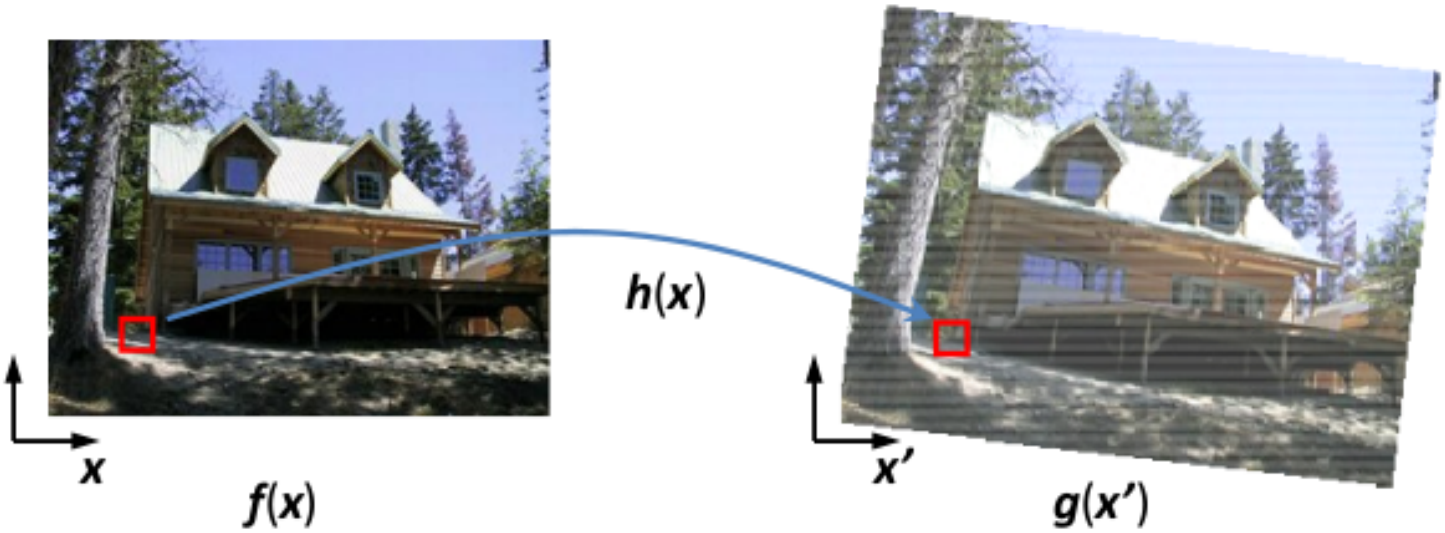
Forward warping

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



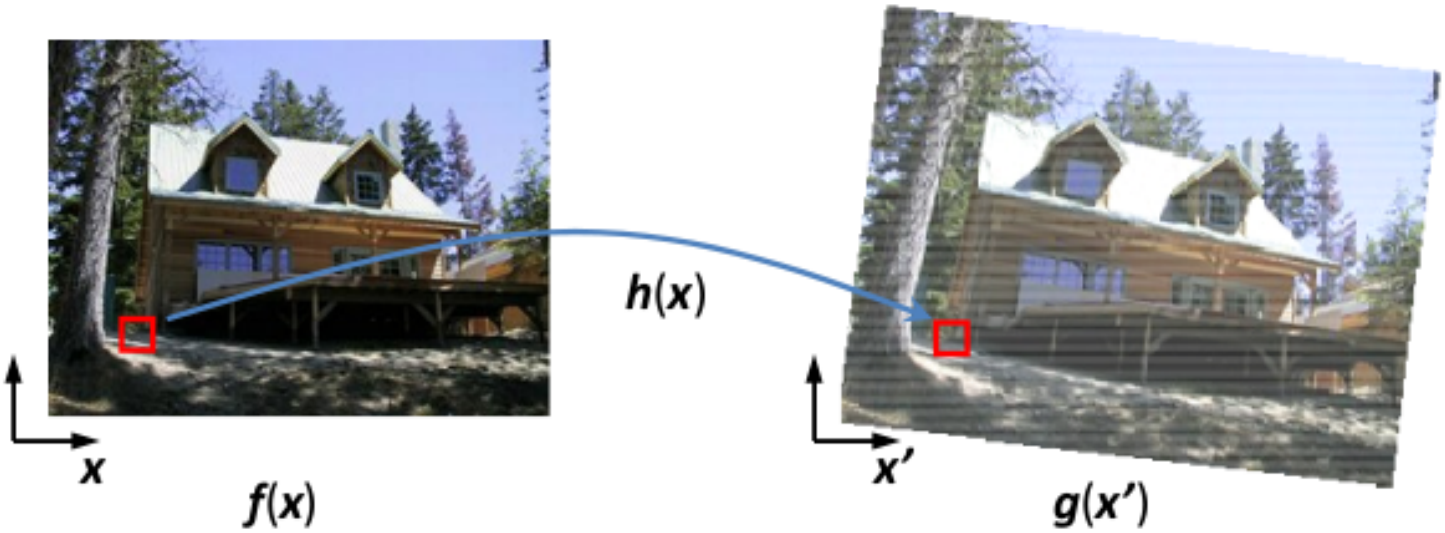
Inverse warping

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



Inverse warping

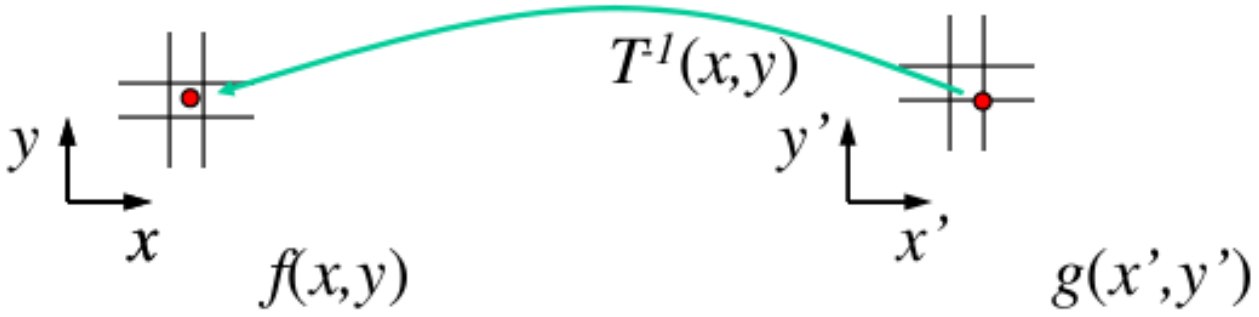
- Get each pixel $g(x')$ from its corresponding location $x' = h(x)$ in $f(x)$
- What if a pixel "comes from between" two pixels?



Inverse warping

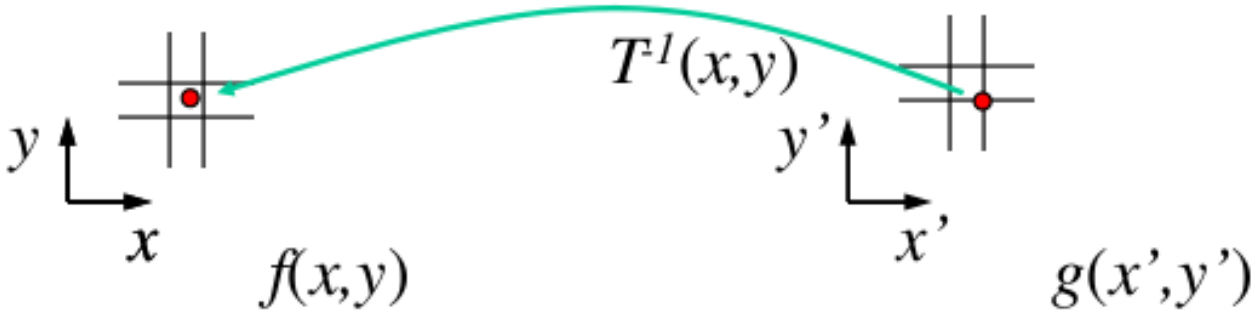
- Get each pixel $g(x')$ from its corresponding location $x' = h(x)$ in $f(x)$
- What if a pixel "comes from between" two pixels?
- Answer: re-sample color value from *interpolated* (pre-filtered) source image

Inverse warping



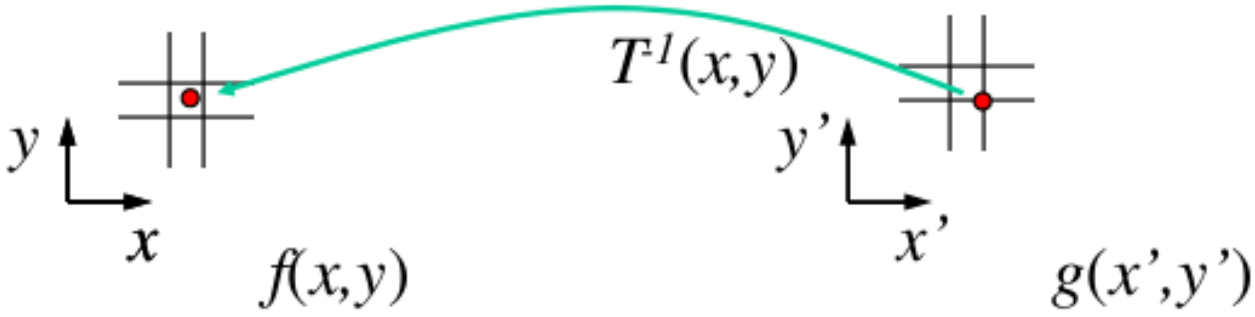
- Get each pixel $g(x', y')$ from its corresponding location
  - $(x, y) = T^{-1}(x', y')$ in the first image

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 a pixel "comes from between" two pixels?

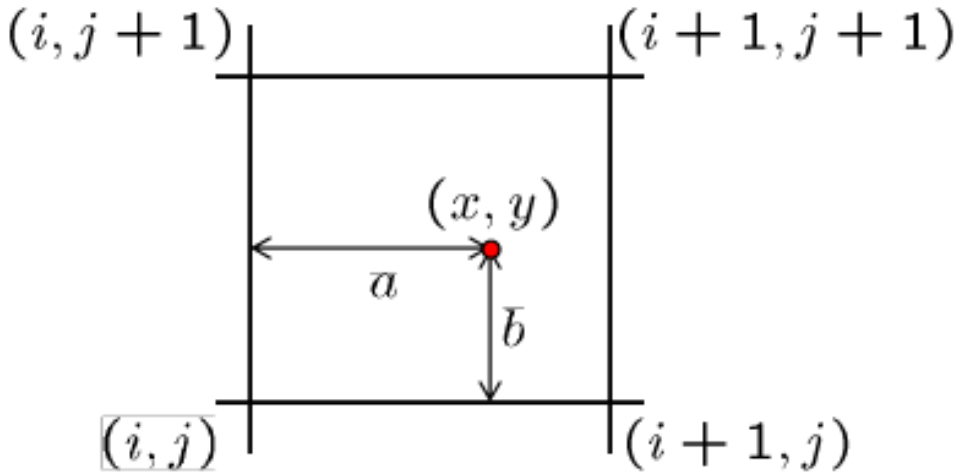
Inverse warping



- Get each pixel $g(x', y')$ from its corresponding location
  - $(x, y) = T^{-1}(x', y')$ in the first image
- Q: What if a pixel "comes from between" two pixels?
- A: Interpolate color values from neighbors (MATLAB: `help interp2`)
  - nearest neighbor, bilinear, bicubic, etc.

Bilinear interpolation

- Sampling at some $f(x, y)$



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

Prefiltering

- Essential for *down-sampling* (decimation) to prevent *aliasing*
- MIP-mapping [Williams' 83] - similar to image pyramid but no down-sampling (transform to mipmap space)
  1. Build a pyramid using a decimation filter:
     - block averaging
     - Burt & Adelson (5-tap bimodal)
     - 7-tap wavelet-based filter (best)
  2. *trilinear* interpolation
     - bilinear within each 2 adjacent levels
     - linear blend *between* levels (based on pixel size)

Look at Forsyth & Ponce for details

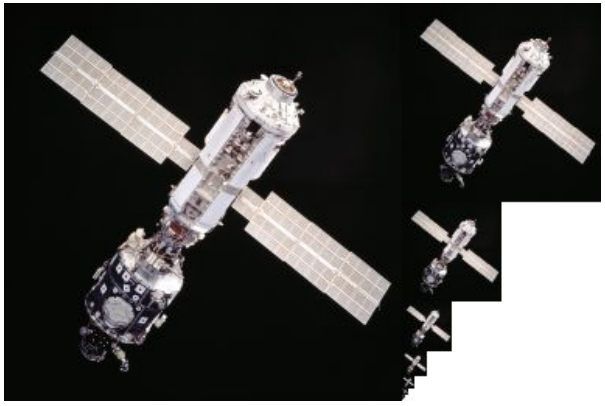
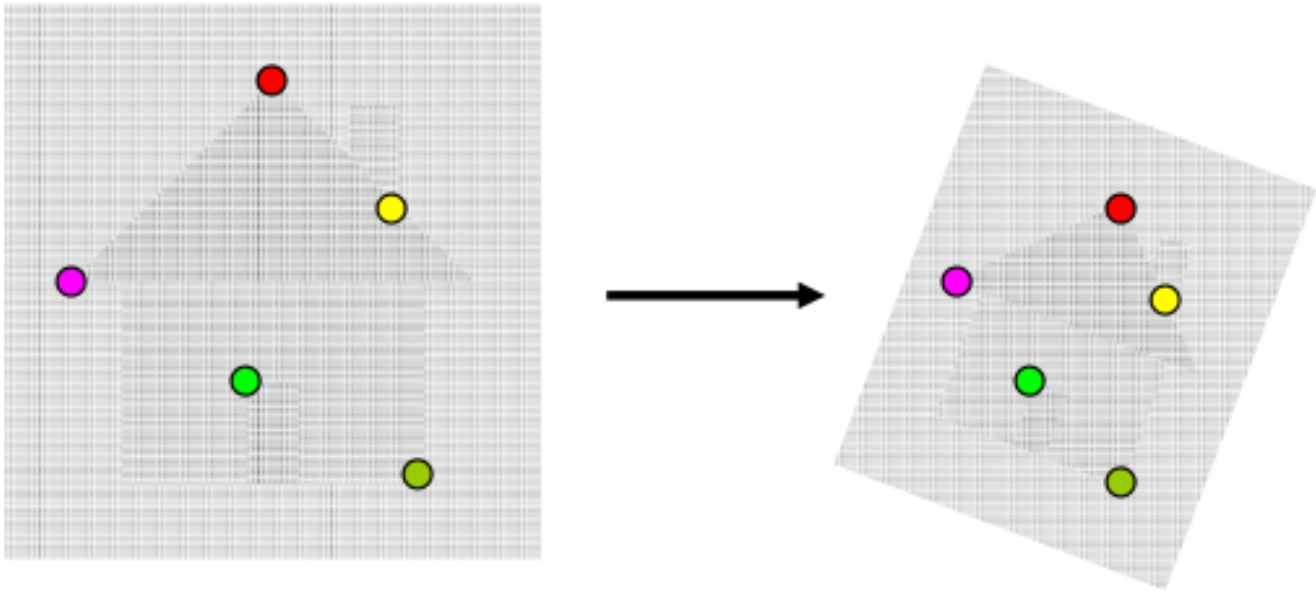
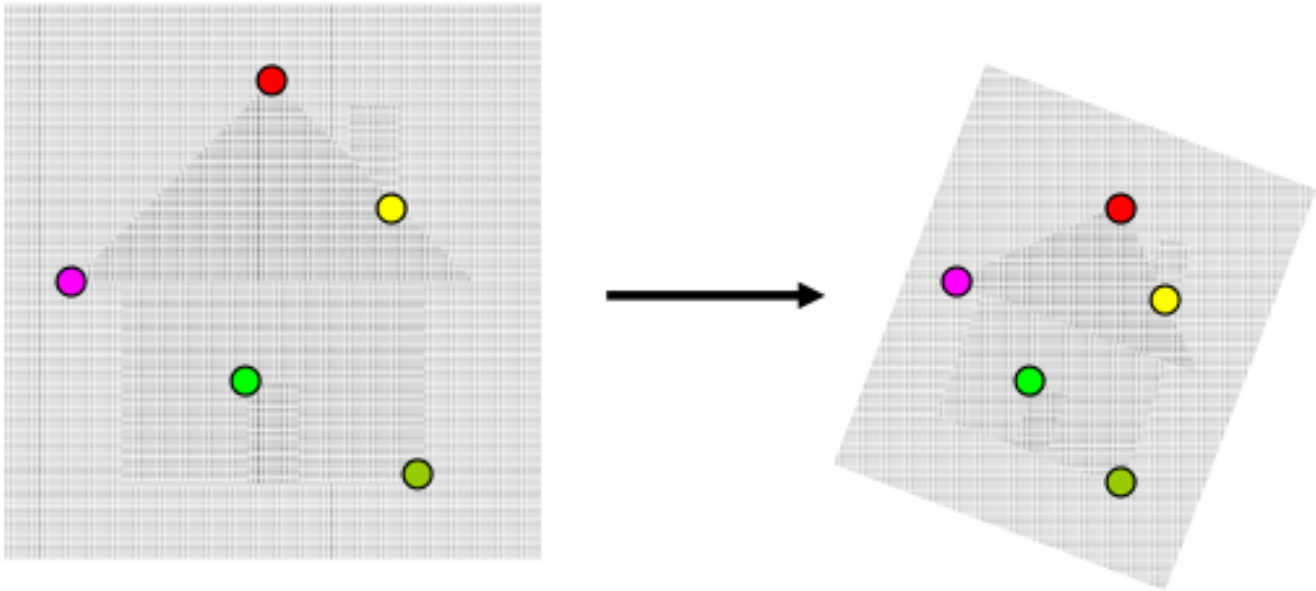


Image alignment



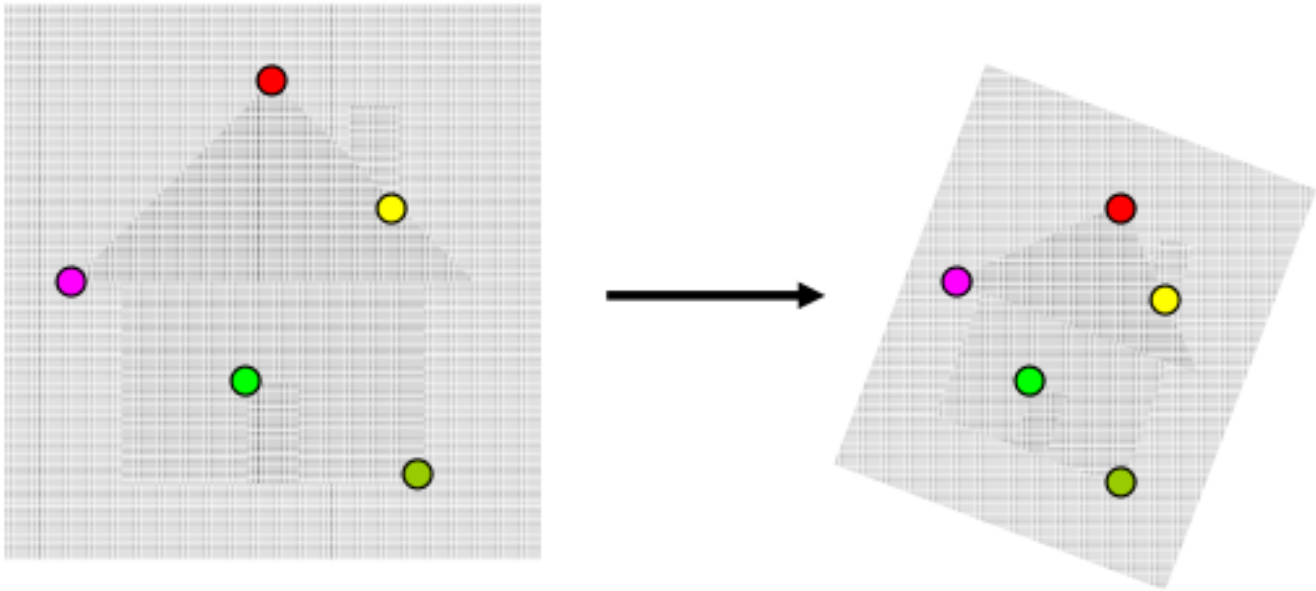
- Two broad approaches:
  - Direct (pixel-based) alignment:
    - Search for alignment where the most pixels agree

Image alignment



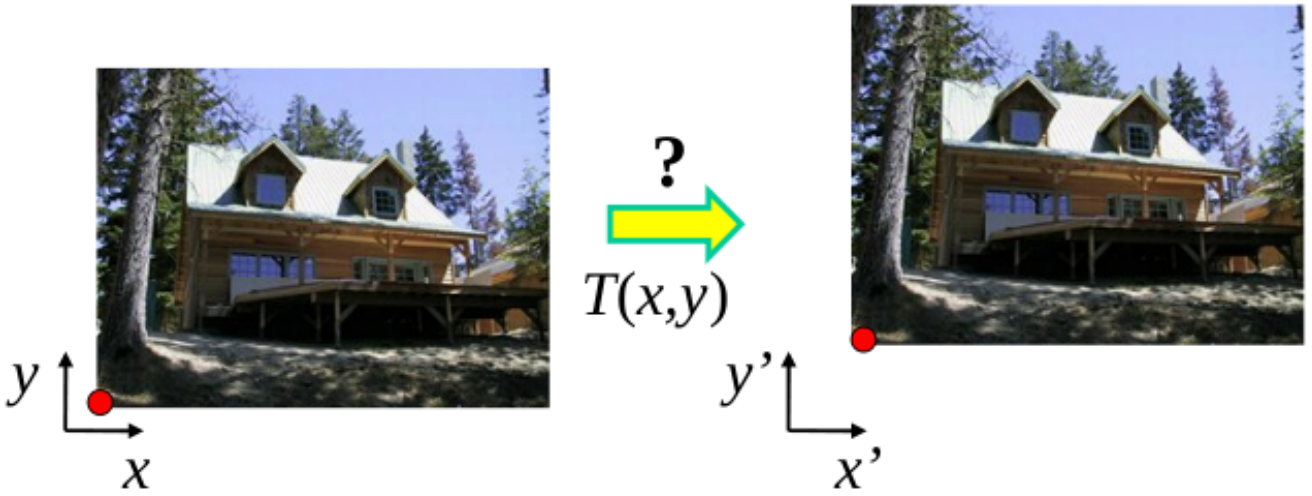
- Two broad approaches:
  - Direct (pixel-based) alignment:
    - Search for alignment where the most pixels agree
  - Feature-based alignment:
    - Search for alignment where *extracted features* agree
    - Can be verified using pixel-based alignment

Image alignment



- Two broad approaches:
  - Direct (pixel-based) alignment:
    - Search for alignment where the most pixels agree
  - Feature-based alignment:
    - Search for alignment where *extracted features* agree
    - Can be verified using pixel-based alignment
  - How many features do we need?

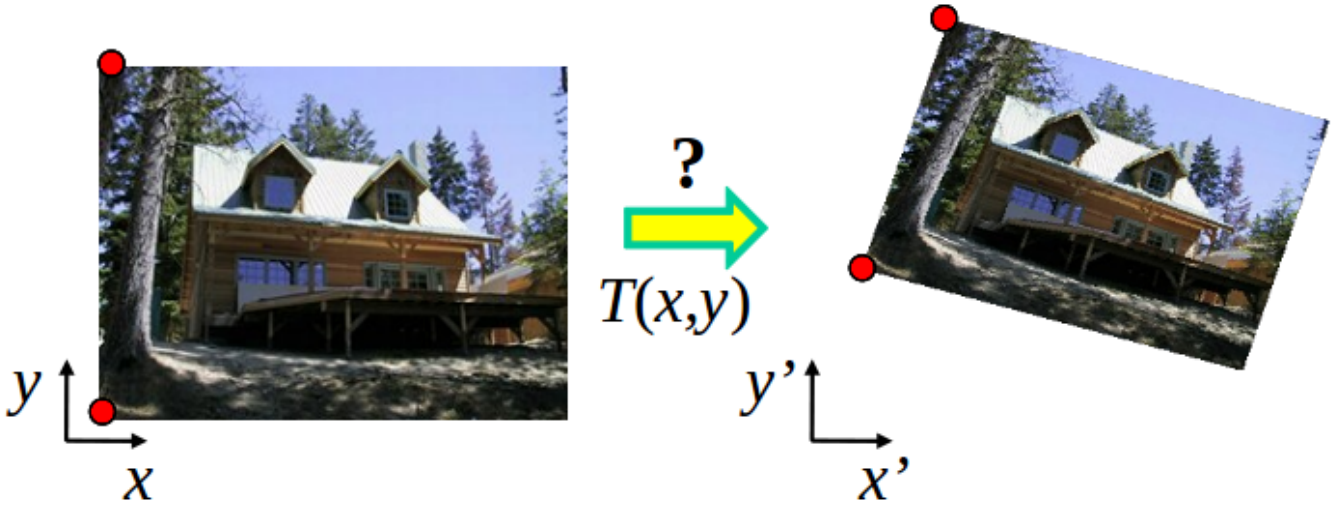
Translation: # correspondences?



- How many correspondences needed for translation?
- How many degrees of freedom?
- What is the transformation matrix?

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

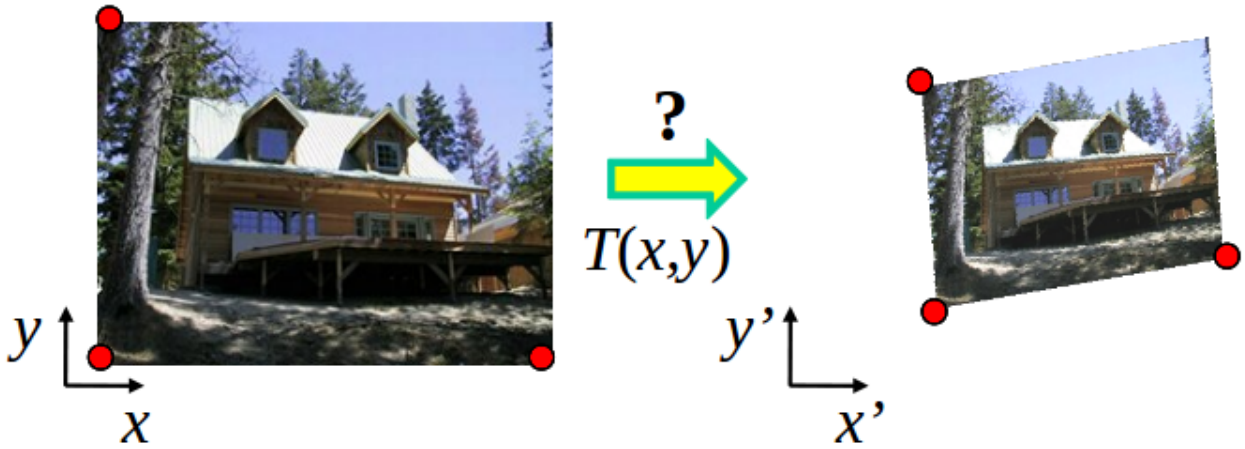
Euclidean: # correspondences?



- How many correspondences needed for translation + rotation?
- How many degrees of freedom?
- What is the transformation matrix?

$$T = \begin{bmatrix} \cos(\theta) & \sin(\theta) & x' - x \\ -\sin(\theta) & \cos(\theta) & y' - y \\ 0 & 0 & 1 \end{bmatrix}$$

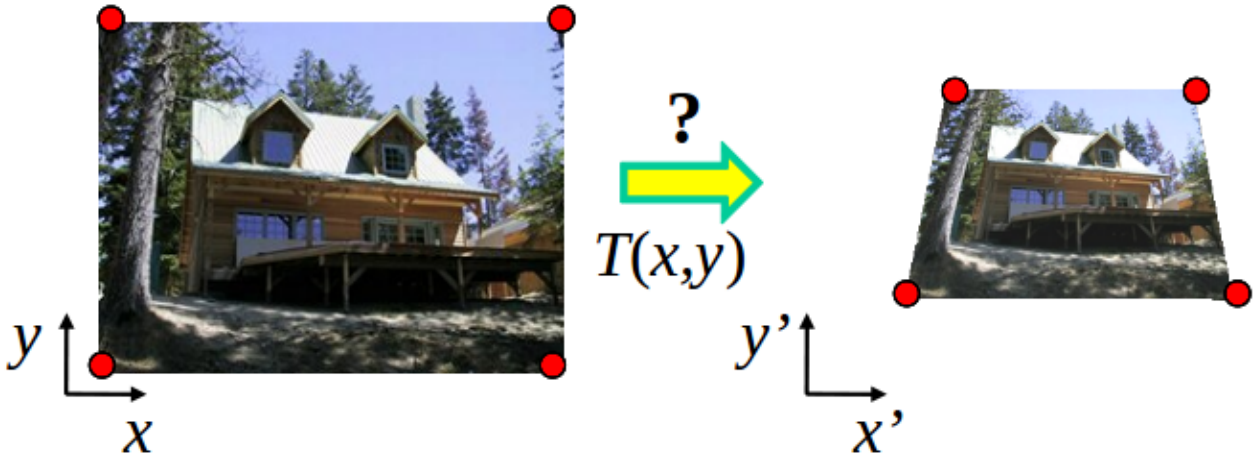
Affine: # correspondences?



- How many correspondences needed for affine?
- How many degrees of freedom?
- What is the transformation matrix?

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

Projective: # correspondences?



- How many correspondences needed for translation?
- How many degrees of freedom?
- What is the transformation matrix?

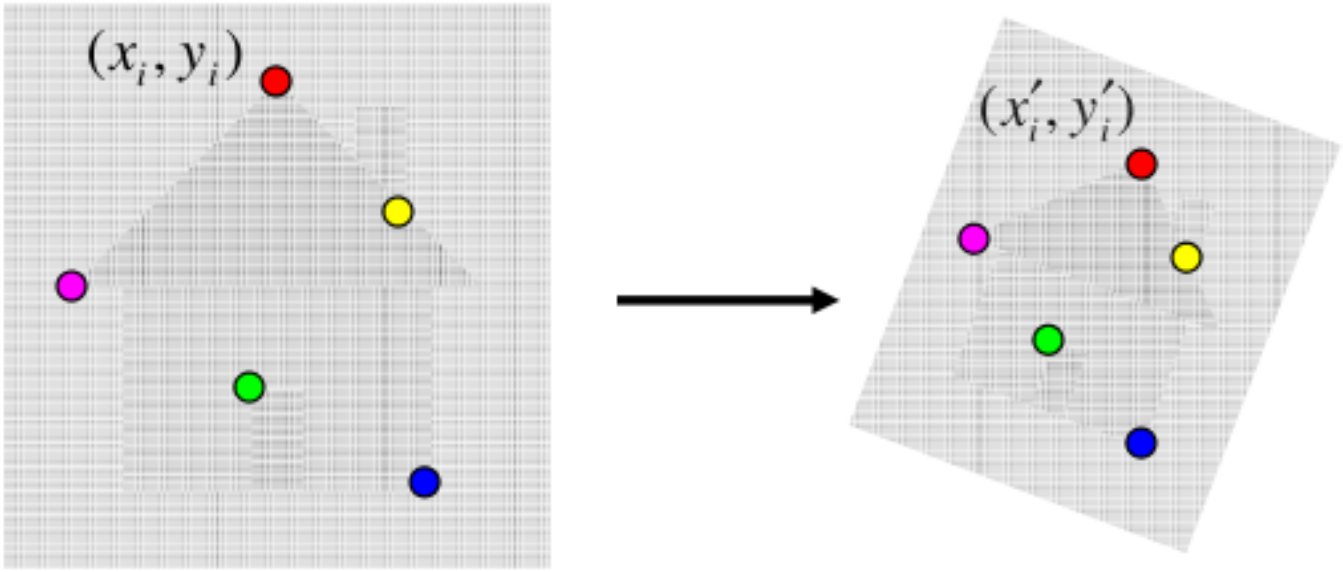
$$T = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Fitting an affine transformation

- Translation and Euclidean are trivial to compute (you should start with these for practice!)
- Let's look at affine models: **approximates** perspective projection of planar objects.

Fitting an affine transformation

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



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

Fitting an affine transformation

- How the heck do we solve for this guy (standard Gaussian elimination)???

Fitting an affine transformation

- How the heck do we solve for this guy (standard Gaussian elimination)???

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \Longrightarrow$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_i & y_i & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_i' \\ y_i' \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

Fitting an affine transformation

- How the heck do we solve for this guy (standard Gaussian elimination)???
  - How many corresponding points do we need?

$$\begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \Longrightarrow$$

$$\begin{bmatrix} x_i & y_i & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_i & y_i & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_i' \\ y_i' \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

Creating panoramas

- General idea: Given a few images of the same scene, create a wider view by combining.

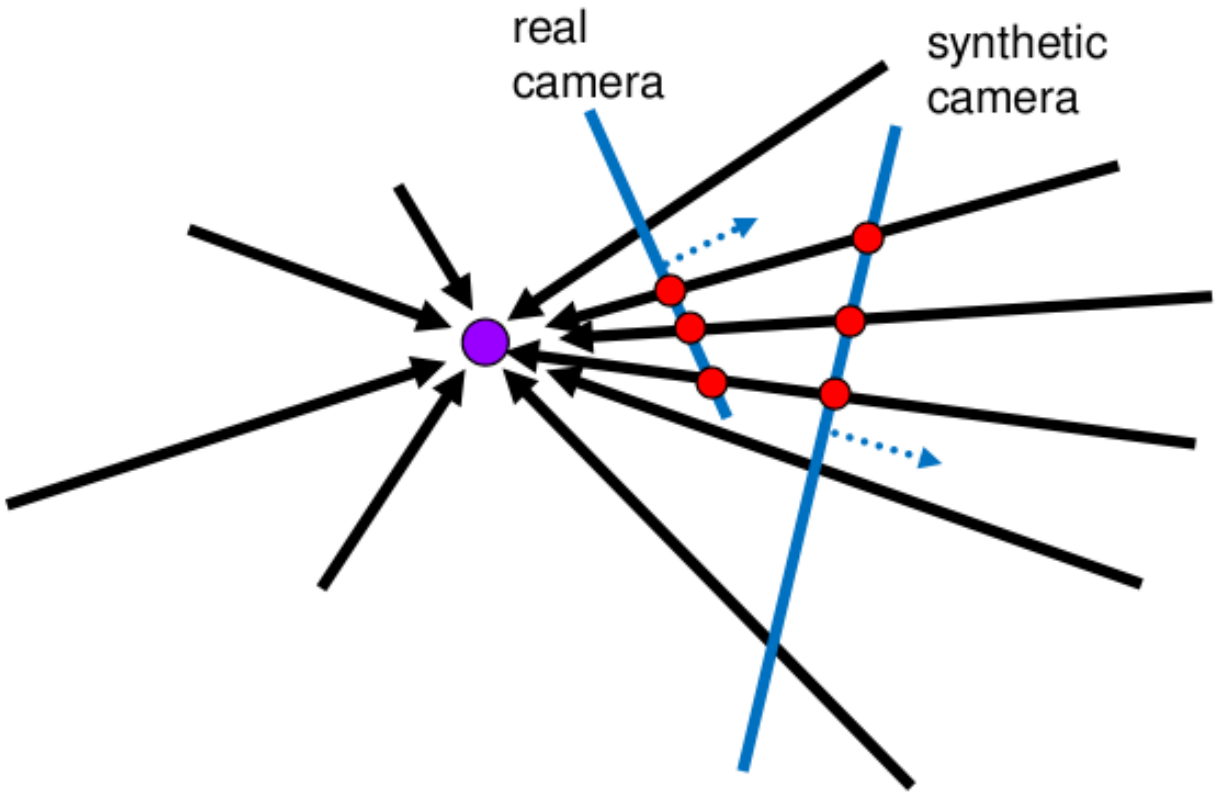
How?

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

How?

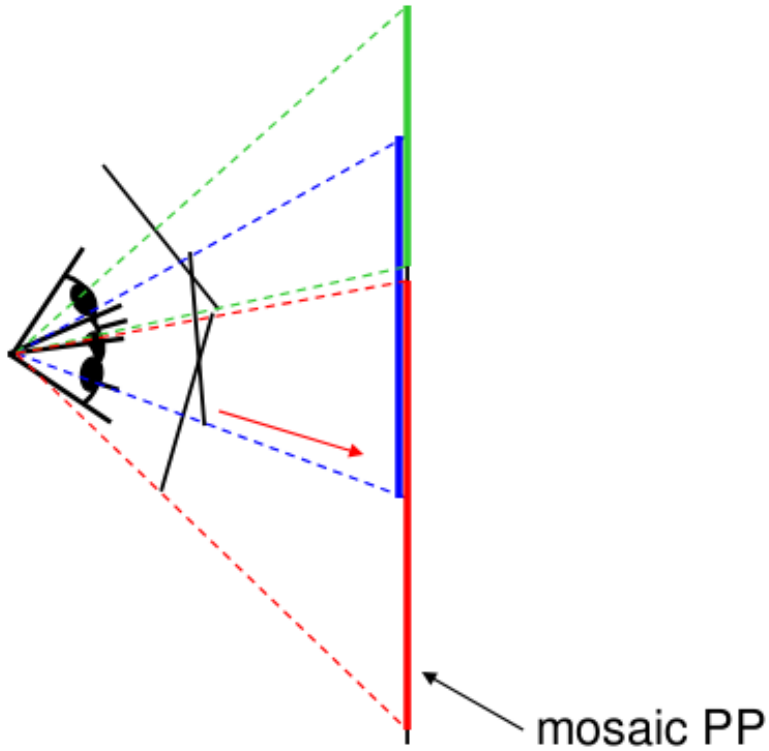
- Basic procedure:
  - Take a sequence of images from the same position
    - Rotate the camera about its optical center
  - Compute the transformation between the second image and the first
  - Transform (warp) the second image to overlap with the first
  - Blend the two together
  - If there are more than a single image, repeat!
- ...**hold on a minute!** why does this work?
  - What about the 3-D geometry of the scene?
  - Why aren't we using it?

Generating synthetic views



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

Image reprojection

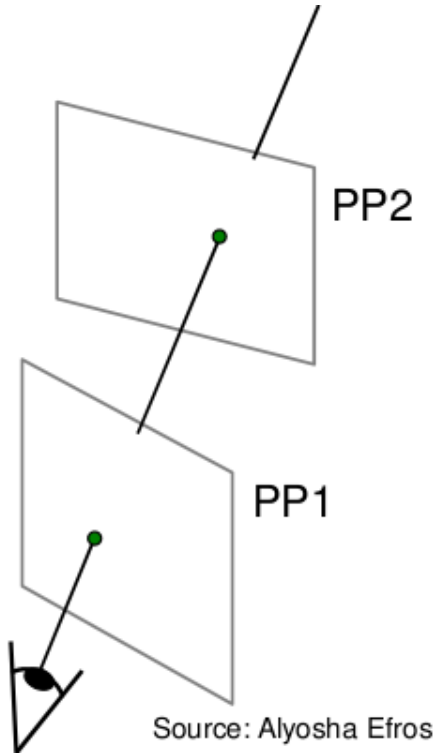


- The mosaic has a natural interpretation in 3-D
  - The images are reprojected onto a common plane
  - Form the mosaic on this new plane
  - This implies that the mosaic is really just a *synthetic wide-angle camera*

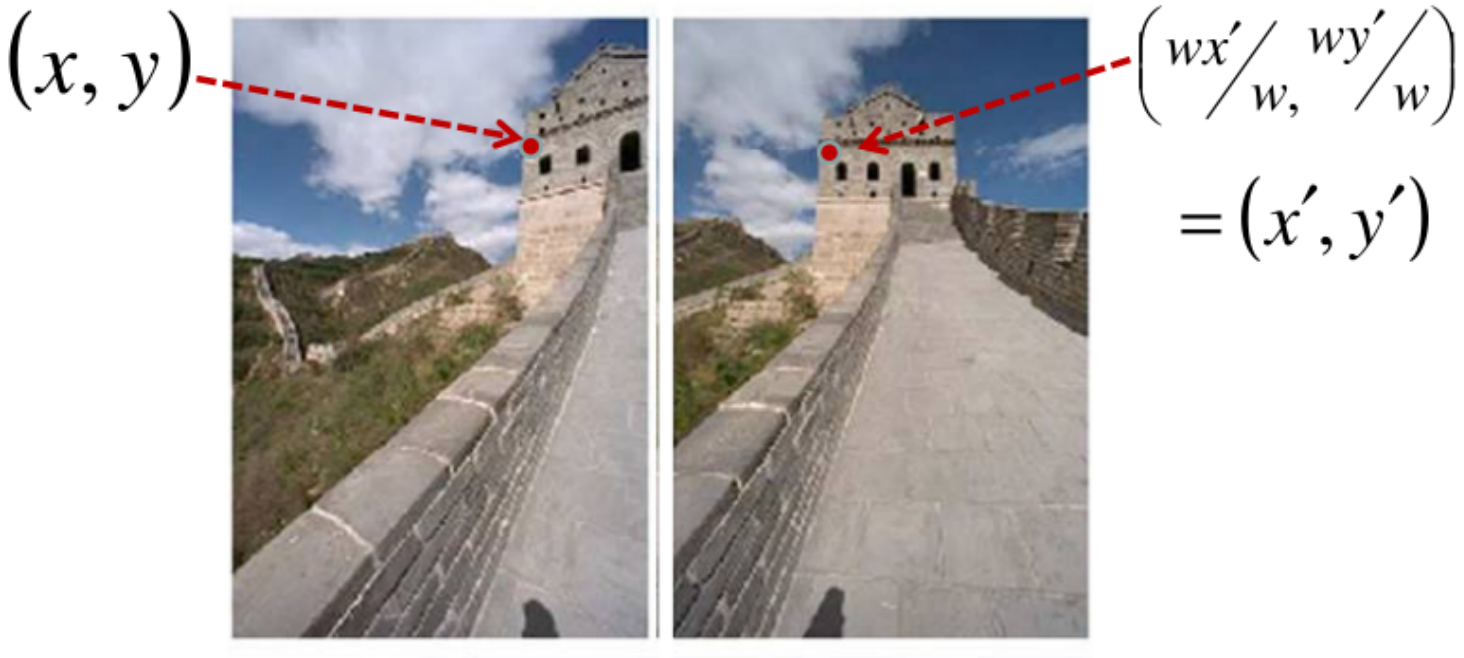
Homography

- How do we relate two images from the same camera center?
  - i.e., how do we map a pixel from PP1 to PP2?
- We need to think of this as a 2-D image warp from one image to another
- A projective transformation is a mapping between any two PPs with the same center of projection!
  - A rectangle maps to an arbitrary quadrilateral
  - Parallel lines do not remain parallel
  - However straight lines are preserved!
- We refer to this transformation as a **homography**

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



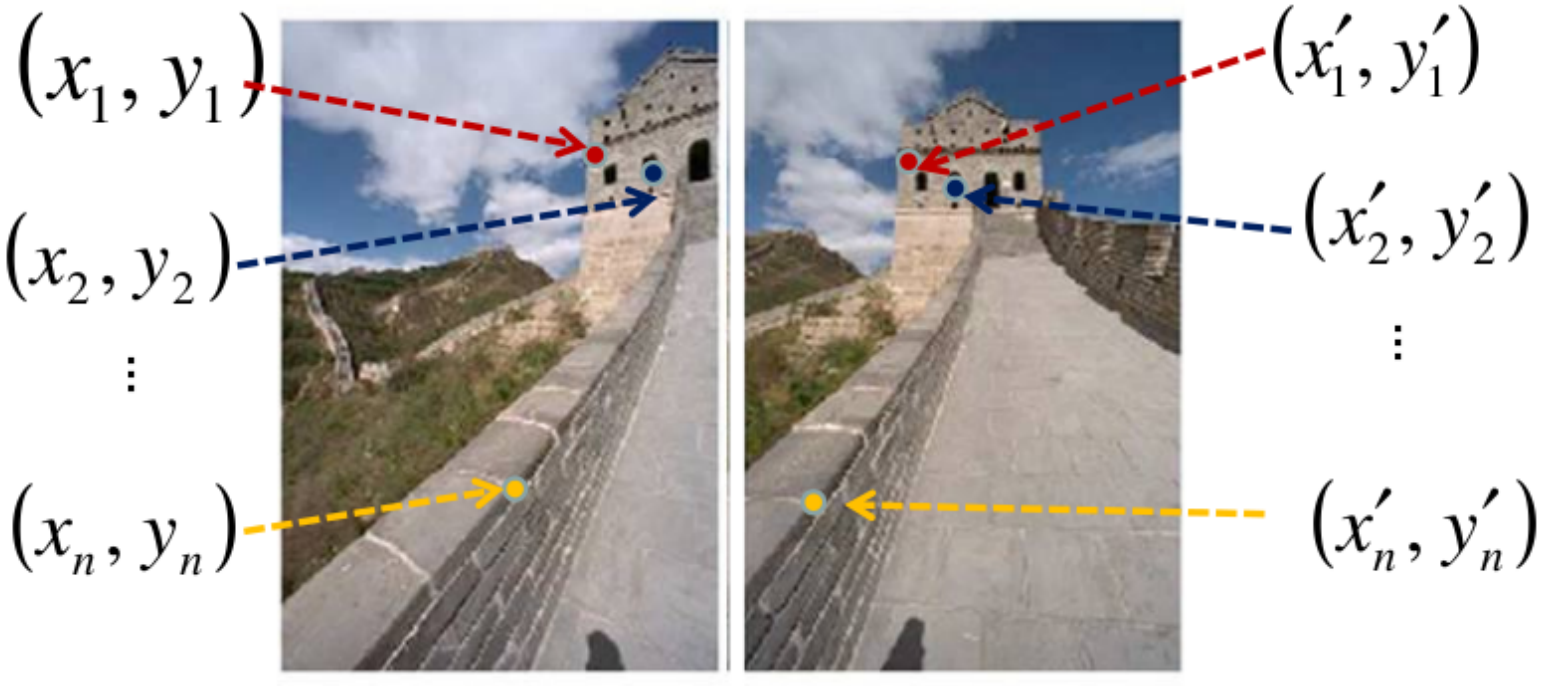
Homography



- To **apply** our homography H
  - Compute $\mathbf{p}' = H\mathbf{p}$
  - Convert $\mathbf{p}'$ from homogeneous coords to image coords (normalize)

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

Homography



- To **compute** our homography H given pairs of corresponding feature points n two images, **similar to our affine transformation**, we need to set up an equation where the parameters of H are the unknowns

How??? - Direct Linear Transform (DLT)

- Recall, we're trying to solve: $\mathbf{p}' = H\mathbf{p}$ where

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

- **Important note:** These are homogeneous vectors! $\therefore$ $\mathbf{p}' \neq H\mathbf{p}$
  - They have the same direction but differ in magnitude by a non-zero scale factor!
- We also note that $\mathbf{p}' = H\mathbf{p}$ can be written as $\mathbf{p}' \times H\mathbf{p} = 0$ (Proof? Trivial and left to you!)

How??? - Direct Linear Transform (DLT)

- Denote the j^{th}-row of H by $\mathbf{h}^j$, then for the i^{th} point correspondence we get

$$H\mathbf{p}_i = \begin{bmatrix} \mathbf{h}^1\mathbf{p}_i \\ \mathbf{h}^2\mathbf{p}_i \\ \mathbf{h}^3\mathbf{p}_i \end{bmatrix}$$

Letting $\mathbf{p}' = [x'_i, y'_i, w'_i]^T$, the cross product becomes

$$\mathbf{p'}_i \times H\mathbf{p}_i = \begin{bmatrix} y'_i\mathbf{h}^3\mathbf{p}_i - w'_i\mathbf{h}^2\mathbf{p}_i \\ w'_i\mathbf{h}^1\mathbf{p}_i - x'_i\mathbf{h}^3\mathbf{p}_i \\ x'_i\mathbf{h}^2\mathbf{p}_i - y'_i\mathbf{h}^1\mathbf{p}_i \end{bmatrix}$$

Note: $\mathbf{h}^j\mathbf{p}_i = [\mathbf{p}_i]^T[\mathbf{h}^j]^T$ (i.e. this is a dot-product!)

How??? - Direct Linear Transform (DLT)

- Therefore, for the i^{th} point correspondence, we can re-write this as:

$$\begin{bmatrix} \mathbf{0}^T & -w_i'\mathbf{p}_i^T & y_i'\mathbf{p}_i^T \\ w_i'\mathbf{p}_i^T & \mathbf{0}^T & -x_i'\mathbf{p}_i^T \\ -y_i'\mathbf{p}_i^T & x_i'\mathbf{p}_i^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} (\mathbf{h}^1)^T \\ (\mathbf{h}^2)^T \\ (\mathbf{h}^3)^T \end{bmatrix} = \mathbf{0}$$

  Notice however that row three is a linear combination of rows one and two! (multiply row 1 by $-\frac{x_i'}{w_i'}$ and row two by $-\frac{y_i'}{w_i'}$ and add them to get row three)

- Therefore, w.l.o.g., we can take the first two rows to get:

$$\underbrace{\begin{bmatrix} \mathbf{0}^T & -w_i'\mathbf{p}_i^T & y_i'\mathbf{p}_i^T \\ w_i'\mathbf{p}_i^T & \mathbf{0}^T & -x_i'\mathbf{p}_i^T \end{bmatrix}}_{A_i} \underbrace{\begin{bmatrix} (\mathbf{h}^1)^T \\ (\mathbf{h}^2)^T \\ (\mathbf{h}^3)^T \end{bmatrix}}_{\mathbf{h}} = \mathbf{0}$$

  which is of the form $A_i\mathbf{h} = 0$ where $A_i \in \mathbb{R}^{2\times 9}$ and $\mathbf{h} \in \mathbb{R}^{9\times 1}$ contains the entries of H

How??? - Direct Linear Transform (DLT)

- Because we are looking for a projective transformation, we need a minimum of 4-corresponding points, i.e., $i = 1, 2, 3, 4$
- This will yield some $A\mathbf{h} = 0$ where $A \in \mathbb{R}^{8\times 9}$ and again $\mathbf{h} \in \mathbb{R}^{9\times 1}$
- Question: What is the rank of A?

How??? - Direct Linear Transform (DLT)

- Because we are looking for a projective transformation, we need a minimum of 4-corresponding points, i.e., $i = 1, 2, 3, 4$
- This will yield some $A\mathbf{h} = 0$ where $A \in \mathbb{R}^{8\times 9}$ and again $\mathbf{h} \in \mathbb{R}^{9\times 1}$
- Question: What is the rank of A?
- Answer: At most A has rank 8!

How??? - Direct Linear Transform (DLT)

- Because we are looking for a projective transformation, we need a minimum of 4-corresponding points, i.e., $i = 1, 2, 3, 4$
- This will yield some $A\mathbf{h} = 0$ where $A \in \mathbb{R}^{8\times 9}$ and again $\mathbf{h} \in \mathbb{R}^{9\times 1}$
- Question: What is the rank of A?
- Answer: At most A has rank 8!
- Question: What does such an $\mathbf{h}$ represent?

How??? - Direct Linear Transform (DLT)

- Because we are looking for a projective transformation, we need a minimum of 4-corresponding points, i.e., $i = 1, 2, 3, 4$
- This will yield some $A\mathbf{h} = 0$ where $A \in \mathbb{R}^{8\times 9}$ and again $\mathbf{h} \in \mathbb{R}^{9\times 1}$
- Question: What is the rank of A?
- Answer: At most A has rank 8!
- Question: What does such an $\mathbf{h}$ represent?
- Answer: A must have a 1-dimensional NULL space that is spanned by $\mathbf{h}$

How??? - Direct Linear Transform (DLT)

- Because we are looking for a projective transformation, we need a minimum of 4-corresponding points, i.e., $i = 1, 2, 3, 4$
- This will yield some $A\mathbf{h} = 0$ where $A \in \mathbb{R}^{8\times 9}$ and again $\mathbf{h} \in \mathbb{R}^{9\times 1}$
- Question: What is the rank of A?
- Answer: At most A has rank 8!
- Question: What does such an $\mathbf{h}$ represent?
- Answer: A must have a 1-dimensional NULL space that is spanned by $\mathbf{h}$
- How on earth do we find a non-trivial vector such that $A\mathbf{h} = 0$?

How??? - Direct Linear Transform (DLT)

- Because we are looking for a projective transformation, we need a minimum of 4-corresponding points, i.e., $i = 1, 2, 3, 4$
- This will yield some $A\mathbf{h} = 0$ where $A \in \mathbb{R}^{8\times 9}$ and again $\mathbf{h} \in \mathbb{R}^{9\times 1}$
- Question: What is the rank of A?
- Answer: At most A has rank 8!
- Question: What does such an $\mathbf{h}$ represent?
- Answer: A must have a 1-dimensional NULL space that is spanned by $\mathbf{h}$
- How on earth do we find a non-trivial vector such that $A\mathbf{h} = 0$?
- Use the SVD!

Recall: singular value decomposition

- Given a matrix $A \in \mathbb{R}^{8\times 9}$, the SVD is

$$A = U\Sigma V^T$$

where $UU^T = I \in \mathbb{R}^{8\times 8}$, $VV^T = I \in \mathbb{R}^{9\times 9}$, and $\Sigma \in \mathbb{R}^{8\times 9} = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_8)$ with $\sigma_9 = 0$

- Because U and V are unitary, we can write

$$AV = U\Sigma$$

and letting $U = [\mathbf{u}_1, \mathbf{u}_2, \ldots]$, $V = [\mathbf{v}_1, \mathbf{v}_2, \ldots]$ we get

$$A\mathbf{v}_i = \mathbf{u}_i\sigma_i = \sigma_i\mathbf{u}_i$$

Recall: singular value decomposition

- So what happens for $i = 9$?

$$A\mathbf{v}_9 = \sigma_9 \mathbf{u}_9$$

  but there is no $\mathbf{u}_9$ and $\sigma_9 = 0$, so we must get that $A\mathbf{v}_9 = 0$ (try it!)
- What does this tell us?

Recall: singular value decomposition

- So what happens for $i = 9$?

$$A\mathbf{v}_9 = \sigma_9 \mathbf{u}_9$$

  but there is no $\mathbf{u}_9$ and $\sigma_9 = 0$, so we must get that $A\mathbf{v}_9 = 0$ (try it!)
- What does this tell us?
- We have found our NULL space vector!

Recall: singular value decomposition

- So what happens for $i = 9$?

$$A\mathbf{v}_9 = \sigma_9 \mathbf{u}_9$$

but there is no $\mathbf{u}_9$ and $\sigma_9 = 0$, so we must get that $A\mathbf{v}_9 = 0$ (try it!)

- What does this tell us?
- We have found our NULL space vector!
- If $\mathbf{v}_9 = \mathbf{h} = [h1, h2, h3, \ldots, h_9]^T$, then

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

Algorithmically - numerically unstable!

- Assume we have at least four matched points: How do we compute the homography H?
- Using the above formulation can behave poorly due to scaling (homogeneous coords. $\implies \mathbf{p} = ([750, 250, 1]^T)^2$)
- Instead, we should normalize the coordinates (we can always use a similarity transformation to find H)!
- Compute the Normalized DLT:
  1. Translate each coordinate to have zero mean
  2. Scale each point so the average distance to the origin is $\sqrt{2}$
  3. Both of these are done using a similarity transformation:

$$\tilde{\mathbf{p}} = T\mathbf{p} \quad \tilde{\mathbf{p}}' = T'\mathbf{p}'$$

  4. Compute $\tilde{H}$ using the standard DLT in normalized coordinates
  5. Unnormalize: $H = T'^{-1}\tilde{H}T \rightarrow \mathbf{p}' = T'^{-1}\tilde{H}T\mathbf{p} = H\mathbf{p}$

Similarity transformation

- Great, so how do we find such a similarity transformation?

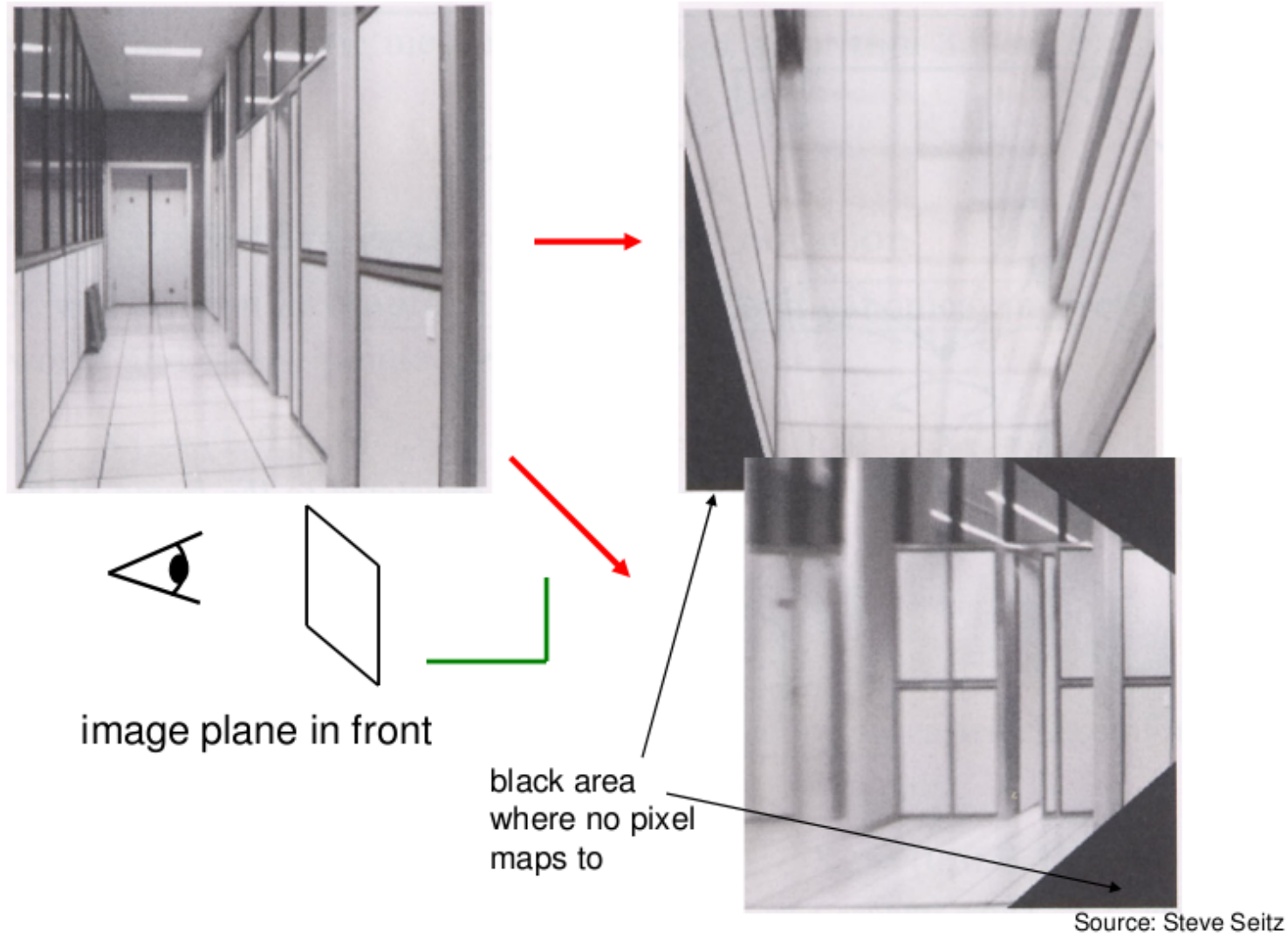
$$T = \begin{bmatrix} \alpha & 0 & -\alpha\mu_x \\ 0 & \alpha & -\alpha\mu_y \\ 0 & 0 & 1 \end{bmatrix}$$

1. Compute the mean of your points $\mathbf{p}$ and denote them by μ_x, μ_y
2. Compute the scale factor such that the average distance of each point is $\sqrt{2}$ from the origin

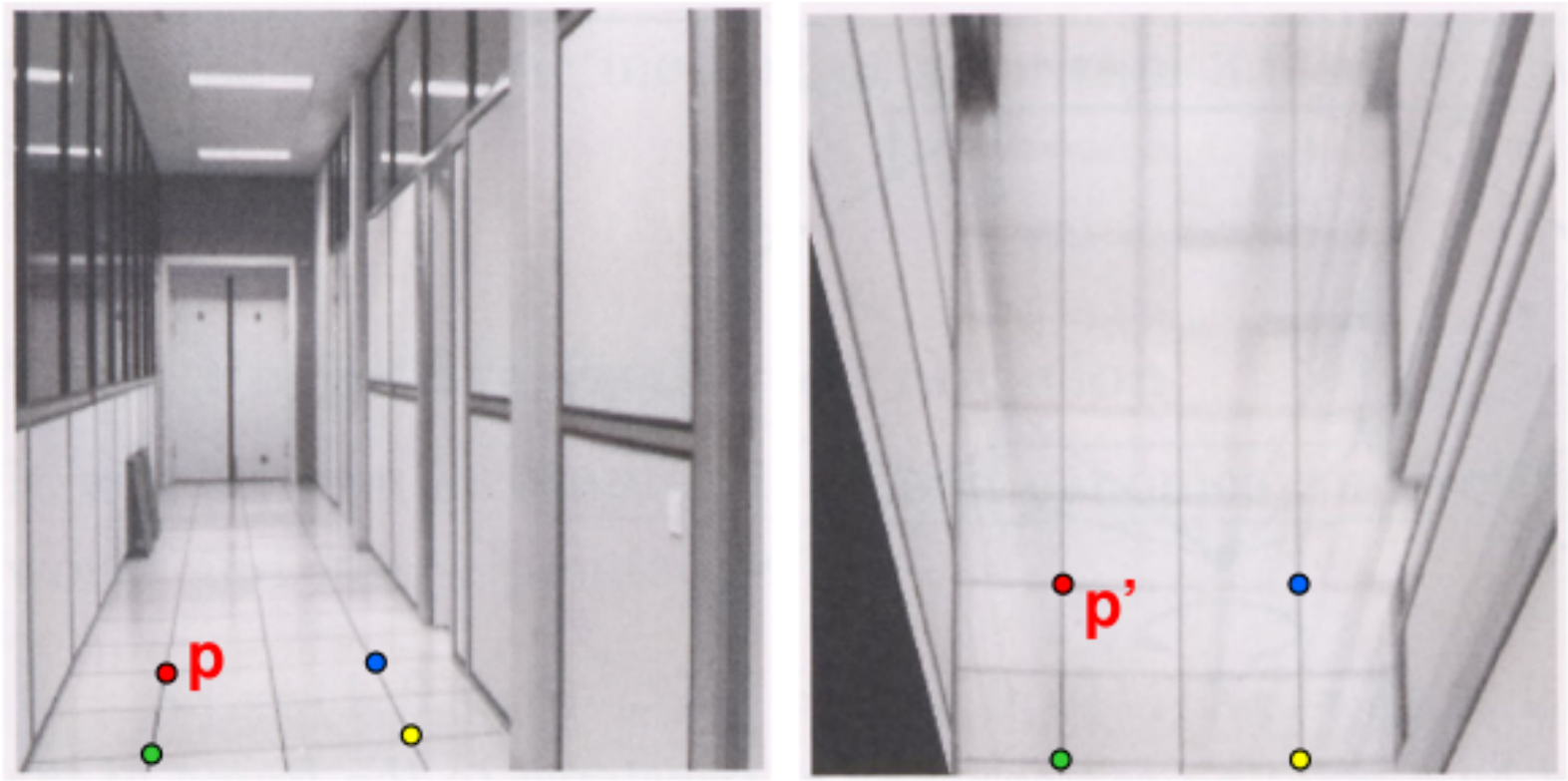
$$\frac{1}{n}\sum_{i=1}^{n} \|\mathbf{p}_i - \mu\| = \sqrt{2} \implies \alpha = \frac{n\sqrt{2}}{\sum_{i=1}^{n} \|\mathbf{p}_i - \mu\|}$$

3. Only on the 2-D points - inhomogeneous!

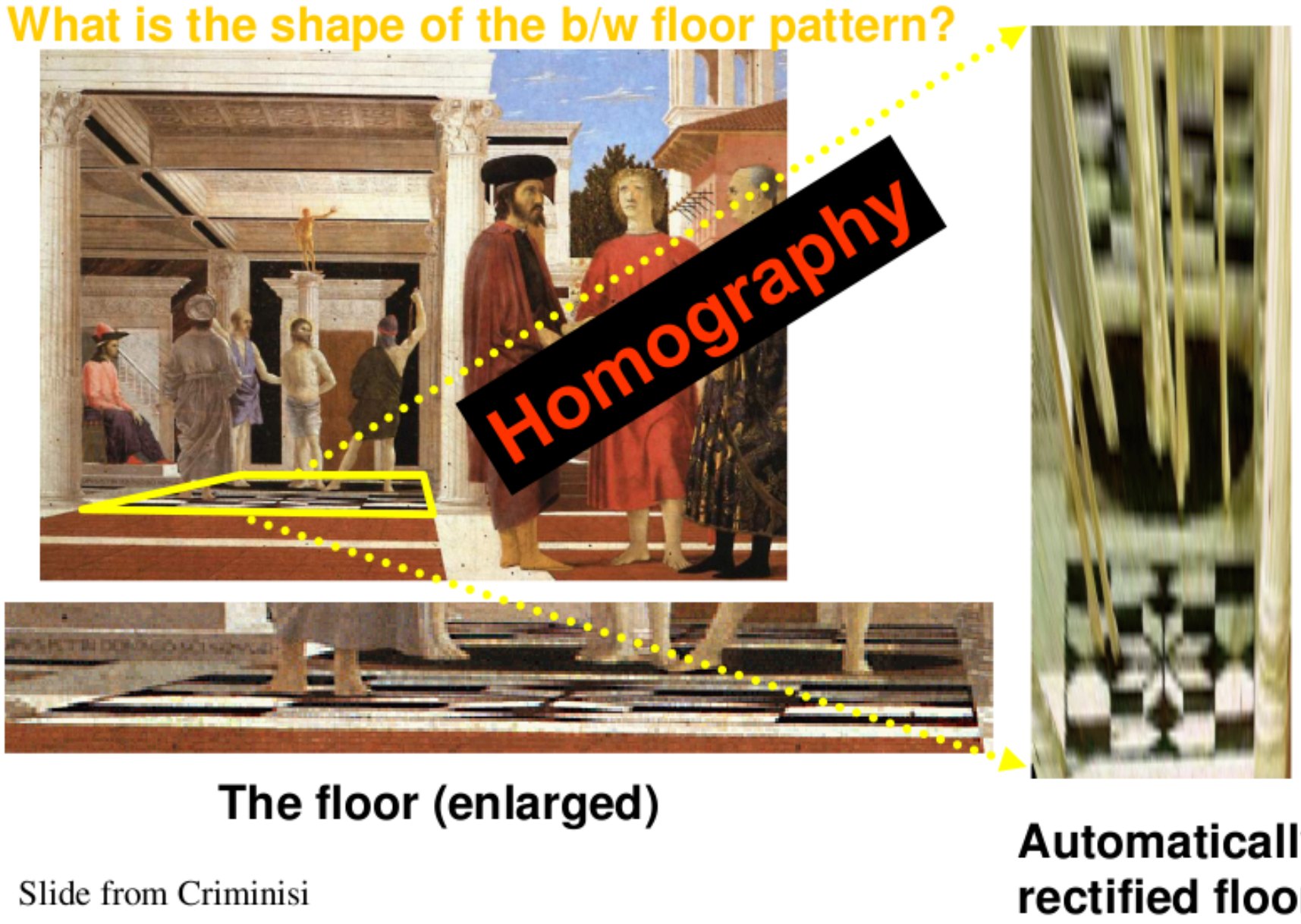
Examples - Image warping with homographies



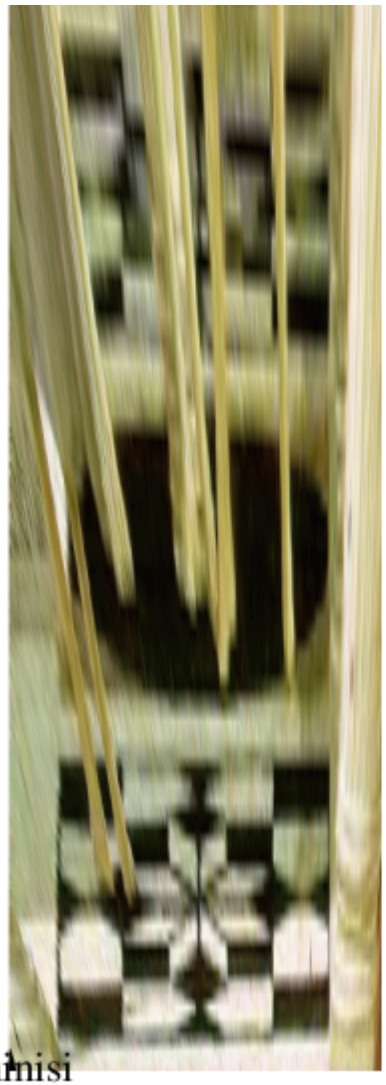
Examples - Image warping with homographies



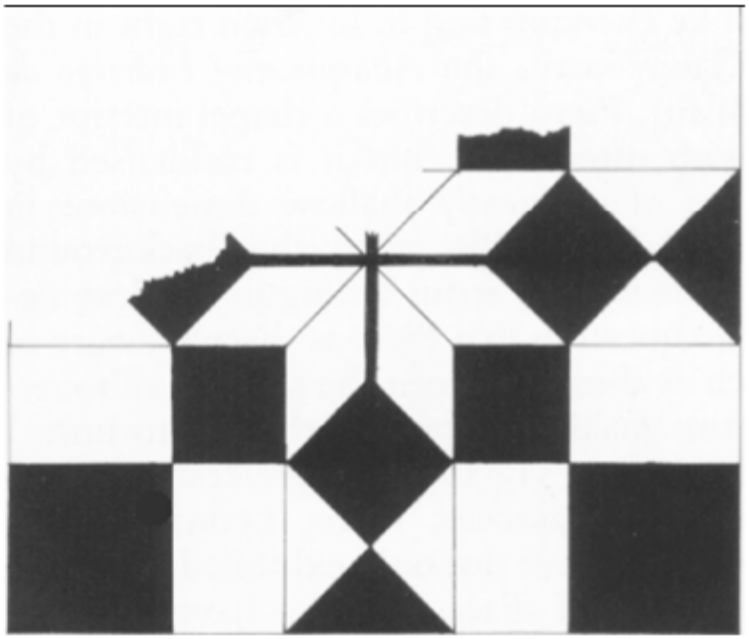
Examples - Image warping with homographies



Examples - Image warping with homographies



Slide from Criminisi



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

Examples - Image warping with homographies

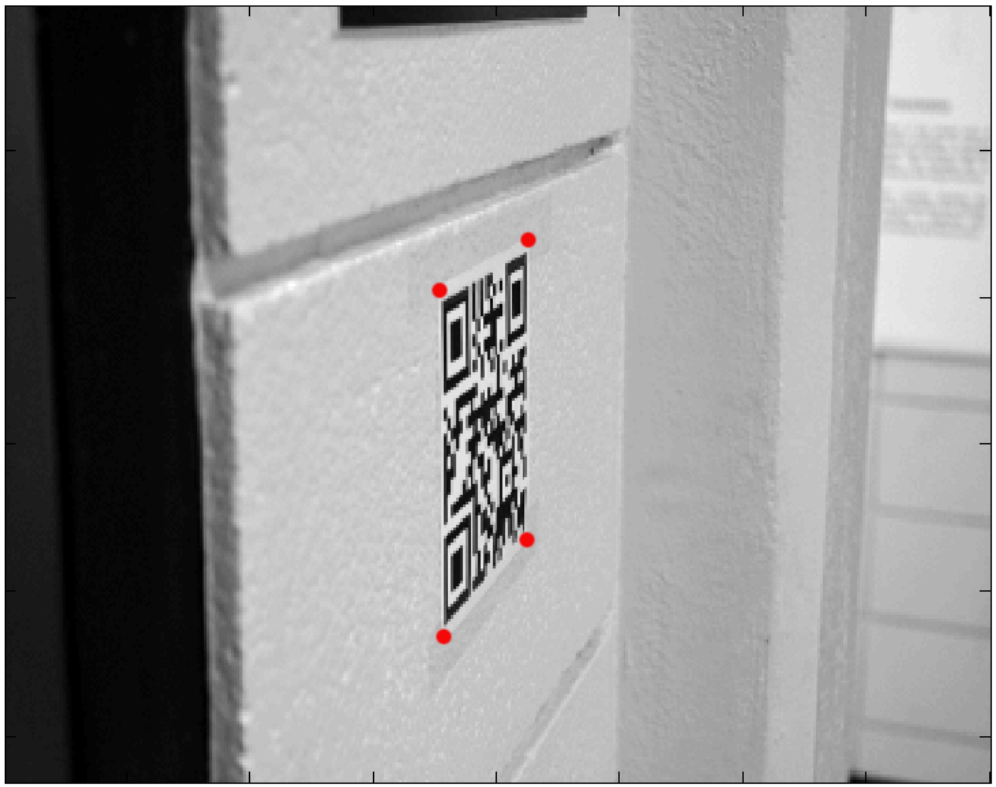
***St. Lucy Altarpiece,* D. Veneziano**

Slide from Criminisi

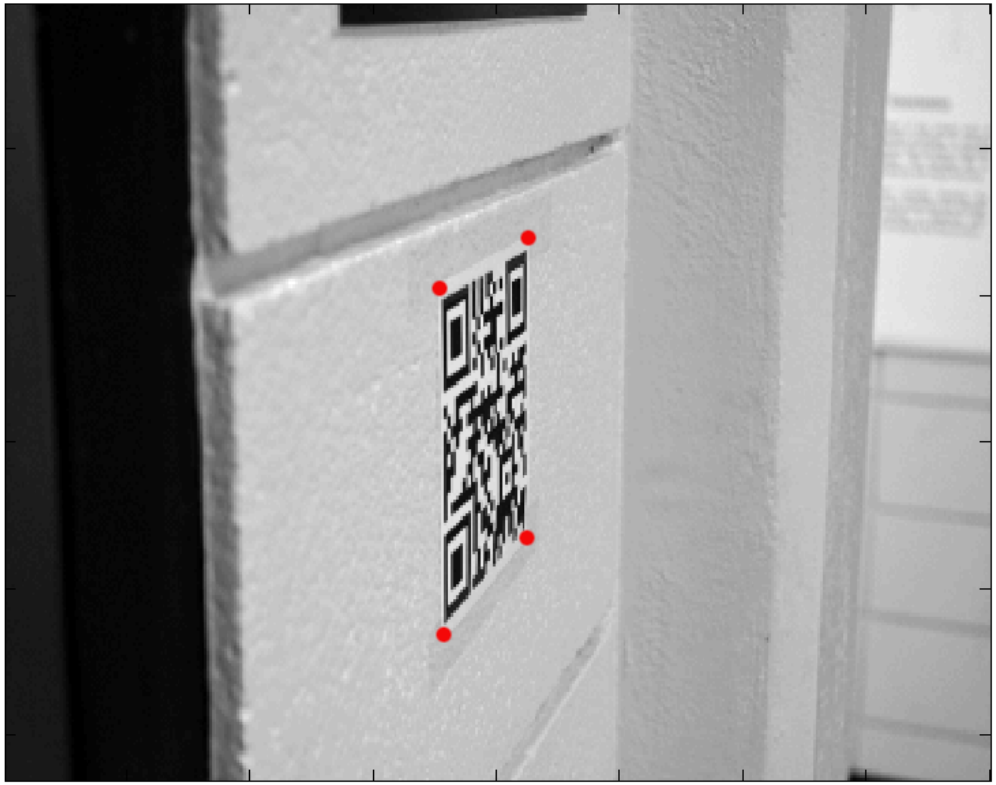
What is the (complicated) shape of the floor pattern?

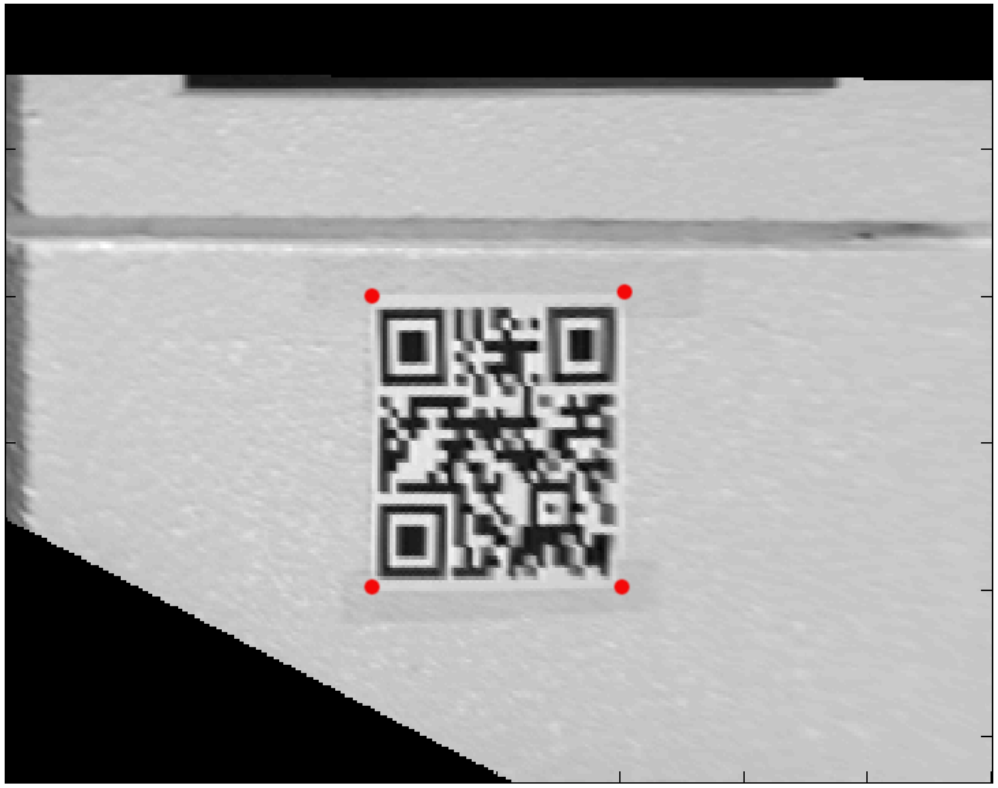
Automatically rectified floor

Examples - Image warping with homographies

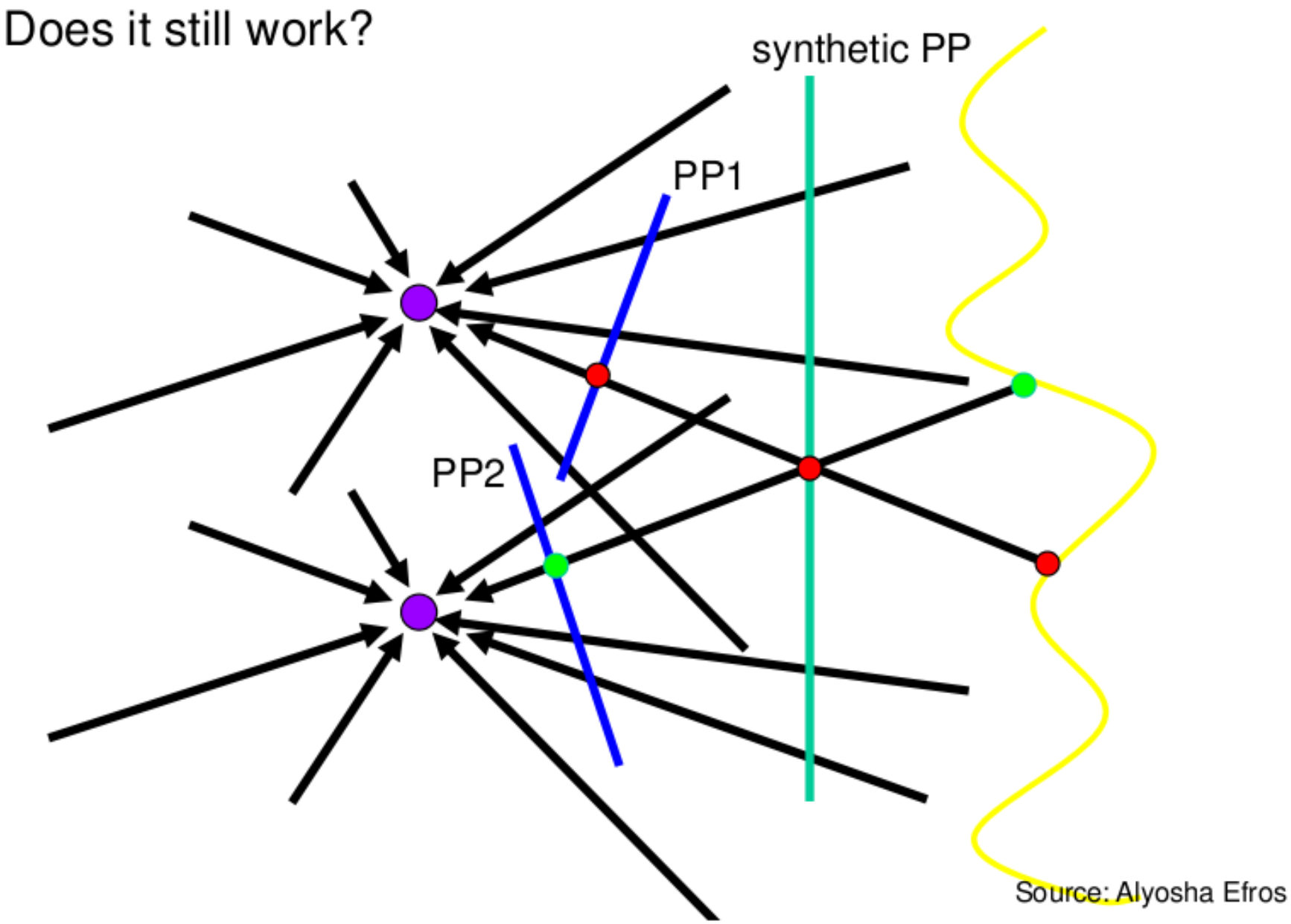


Examples - Image warping with homographies

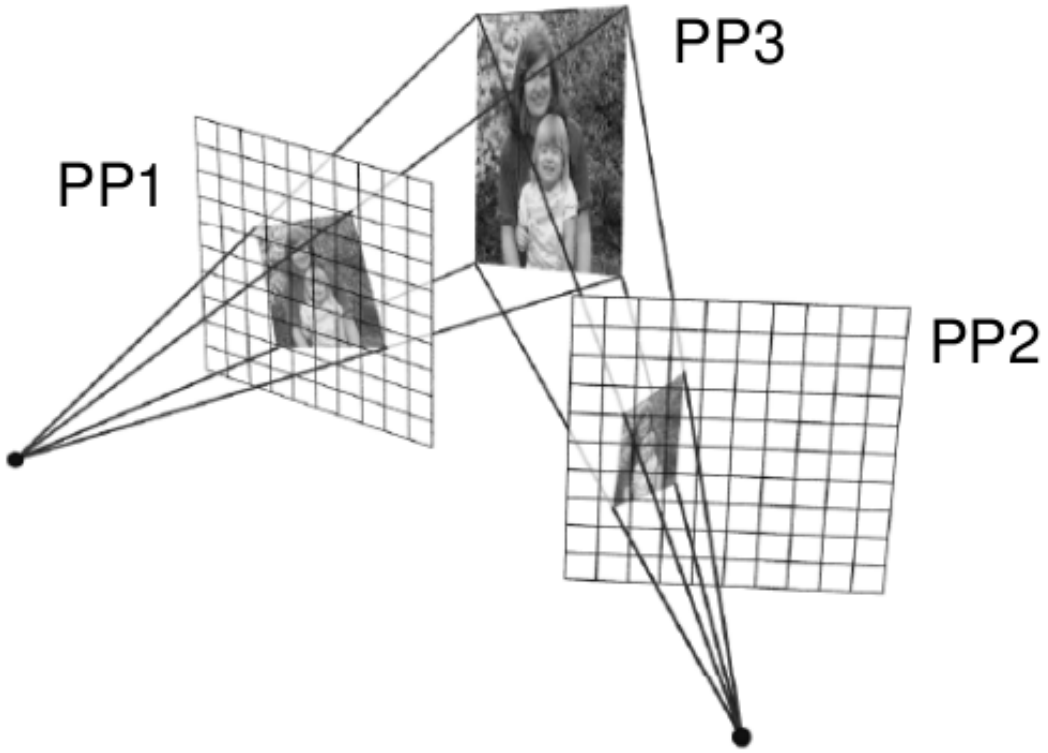




What happens if we change the camera center?



What happens if we change the camera center?



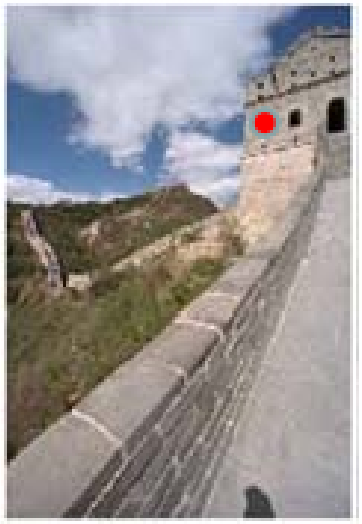
- PP3 is the projection plane for both centers of projection, so we are okay as long as the scene is relatively far from the camera's local frame
  - This is how big aerial photographs are stitched!

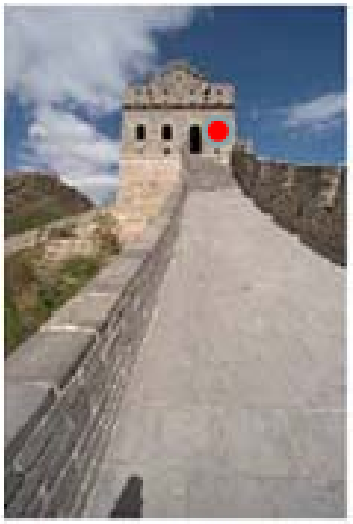
Recap: How to stitch together a panorama?

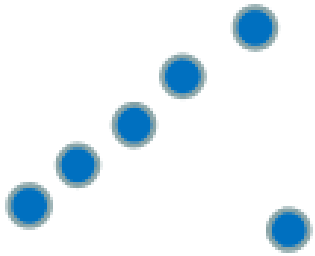
- Take a sequence of images from the same position
  - Rotate the camera about its optical center
- Compute the transformation between the second image and the first by finding common features (correspondences)
- Transform the second image to overlap the first
- Blend the two images together (later)
- If there are more than two images, repeat!

What if some your correspondences are off? - Outliers

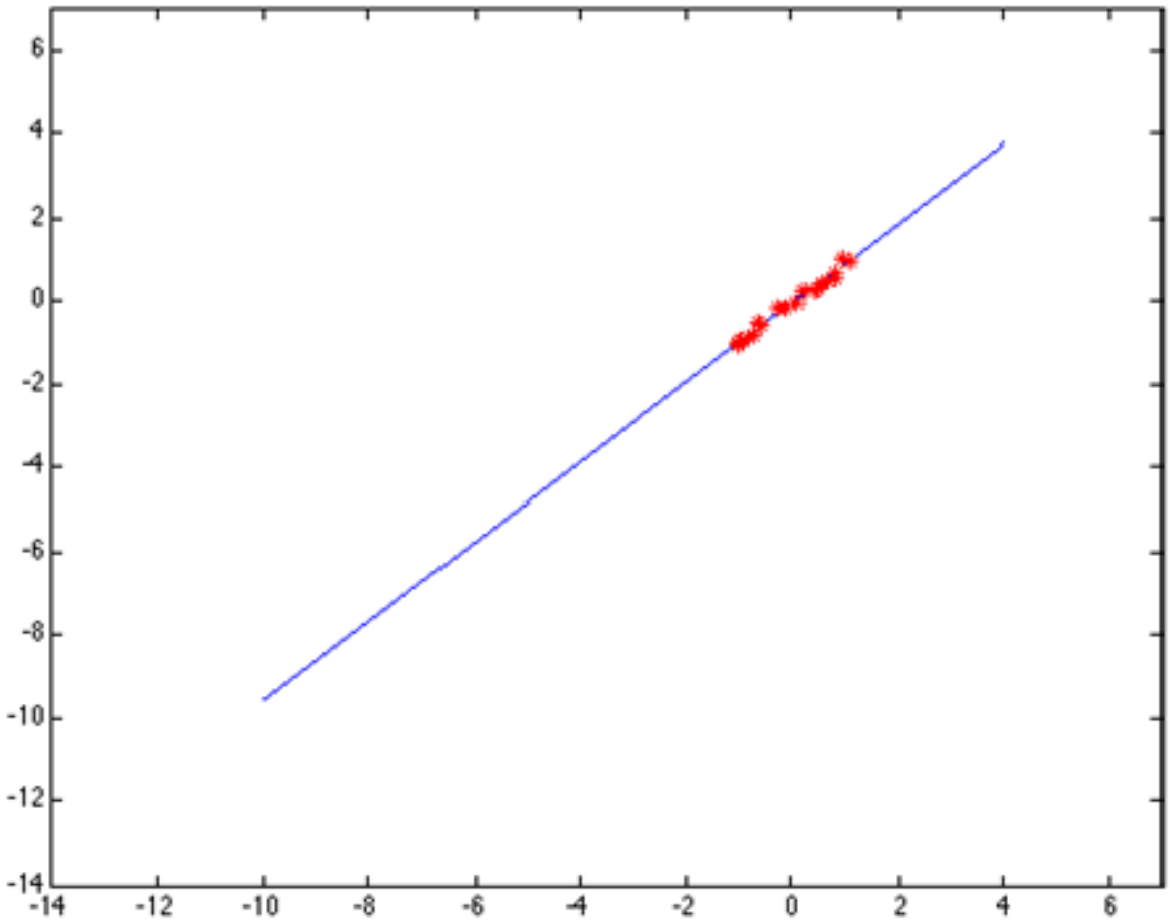
- **Outliers** will ruin the quality of the homography estimate
  - Points that you think correspond but don't
  - Edge point that is really noise or doesn't belong to the line we're fitting



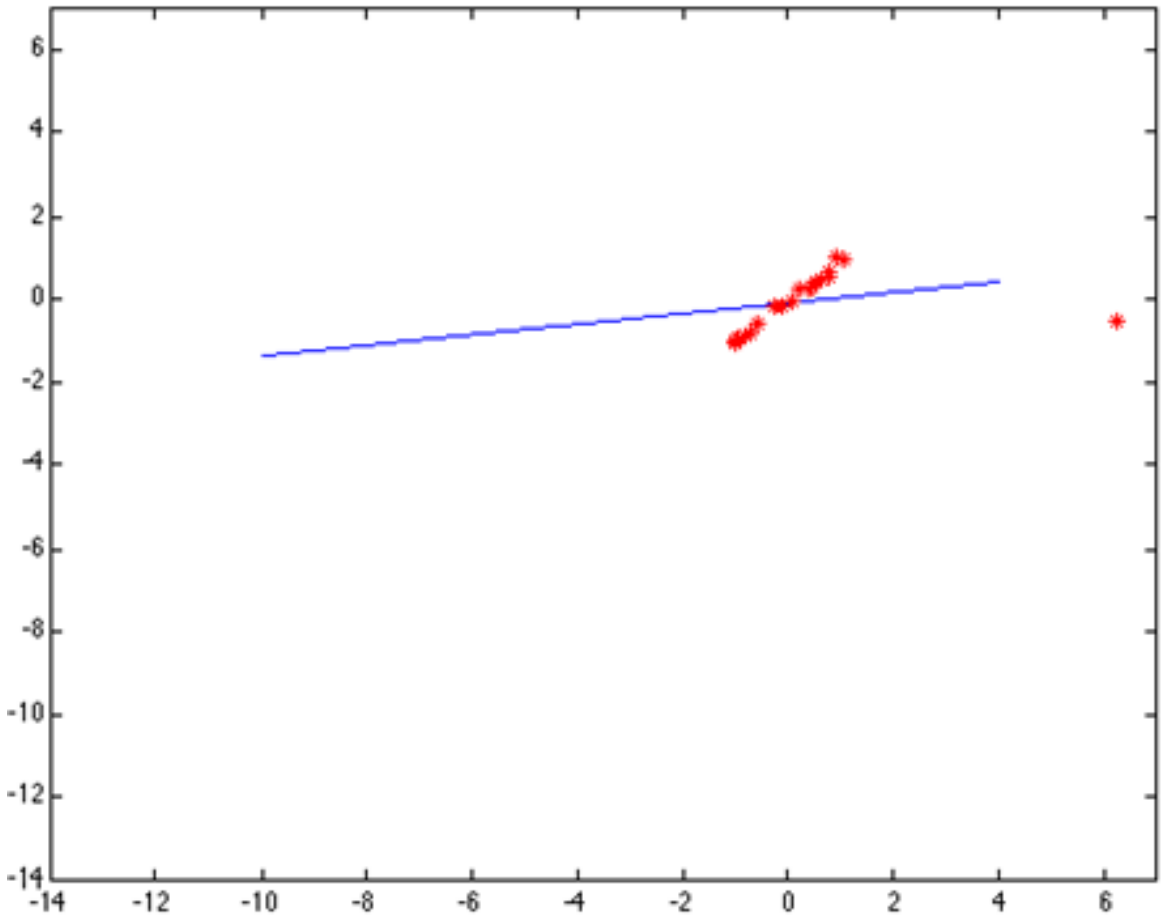




Example: outliers effect our least squares fit



Example: outliers effect our least squares fit



How can we cope with this? - RANSAC

- RANdom Sample and Consensus
- **Approach:** we want to avoid the impact of outiers, so let's look for *inliers* and only use those for our computation!
- **intuition:** if an outlier is chosen to compute the current fit, then the resulting line won't have much support from the rest of the points.

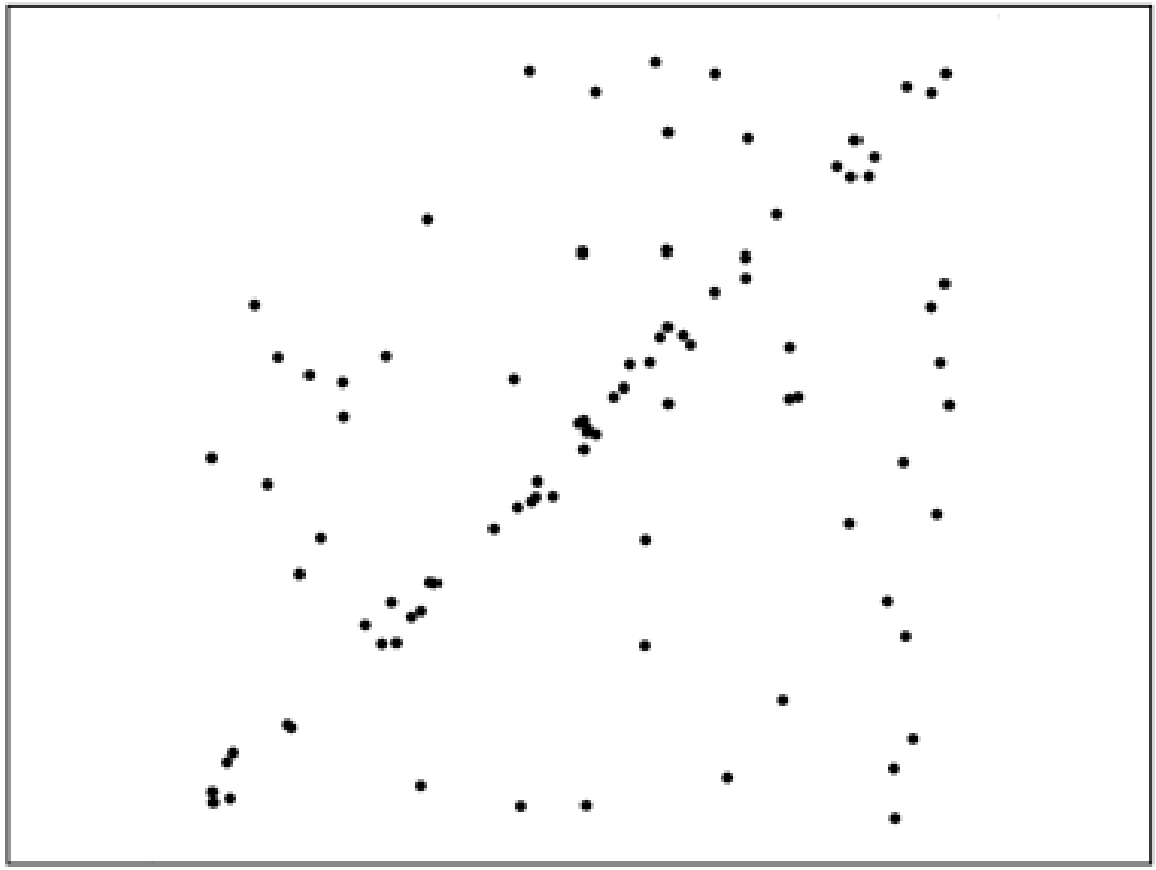
RANSAC: General form

- RANSAC loop:

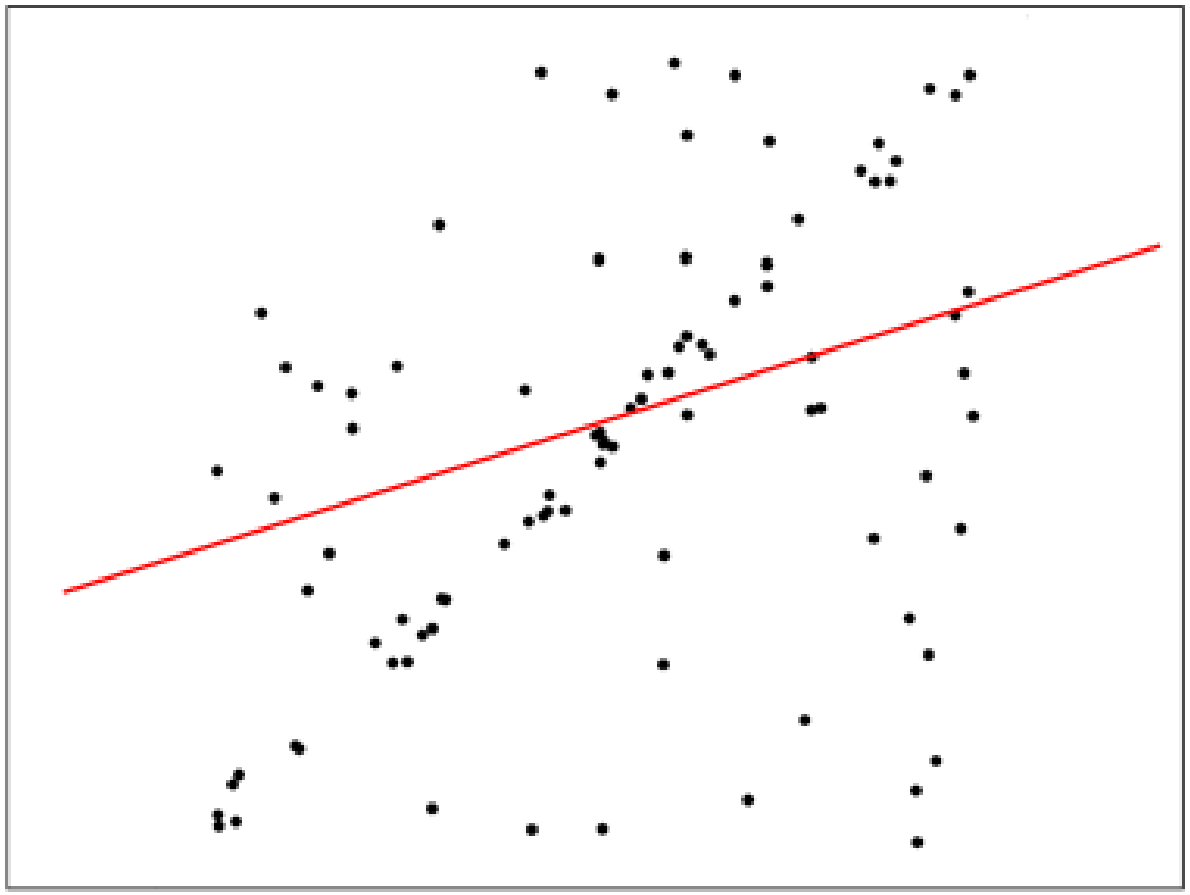
1. Randomly select a *seed group* of points on which to compute our estimation (e.g. a group of punitive matches)
2. Compute the transform (homography in our case) from this *seed* group
3. Find the *inliers* resulting from this transform
4. If the number of inliers is sufficiently large, re-compute an estimate of the transform using all the inliers
5. If NOT, repeat the process!

- Goal: keep the transform with the largest number of inliers!

RANSAC Example

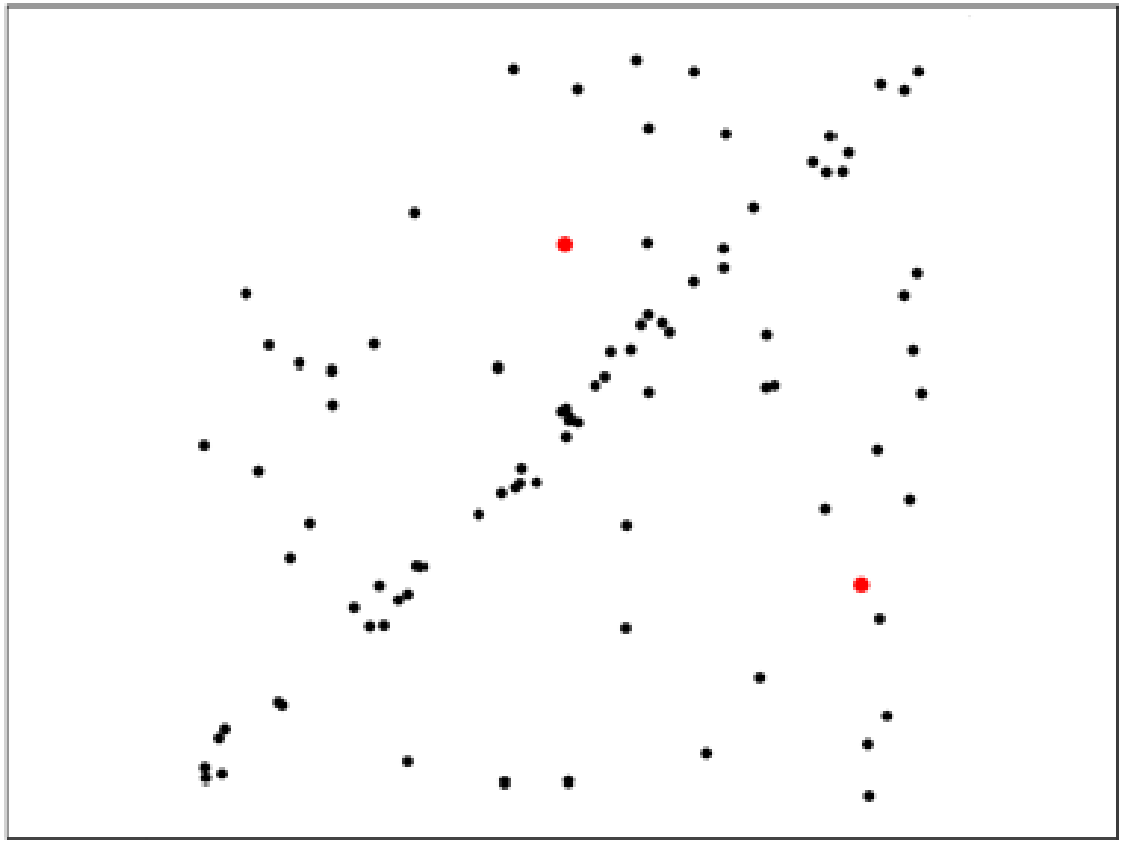


RANSAC Example



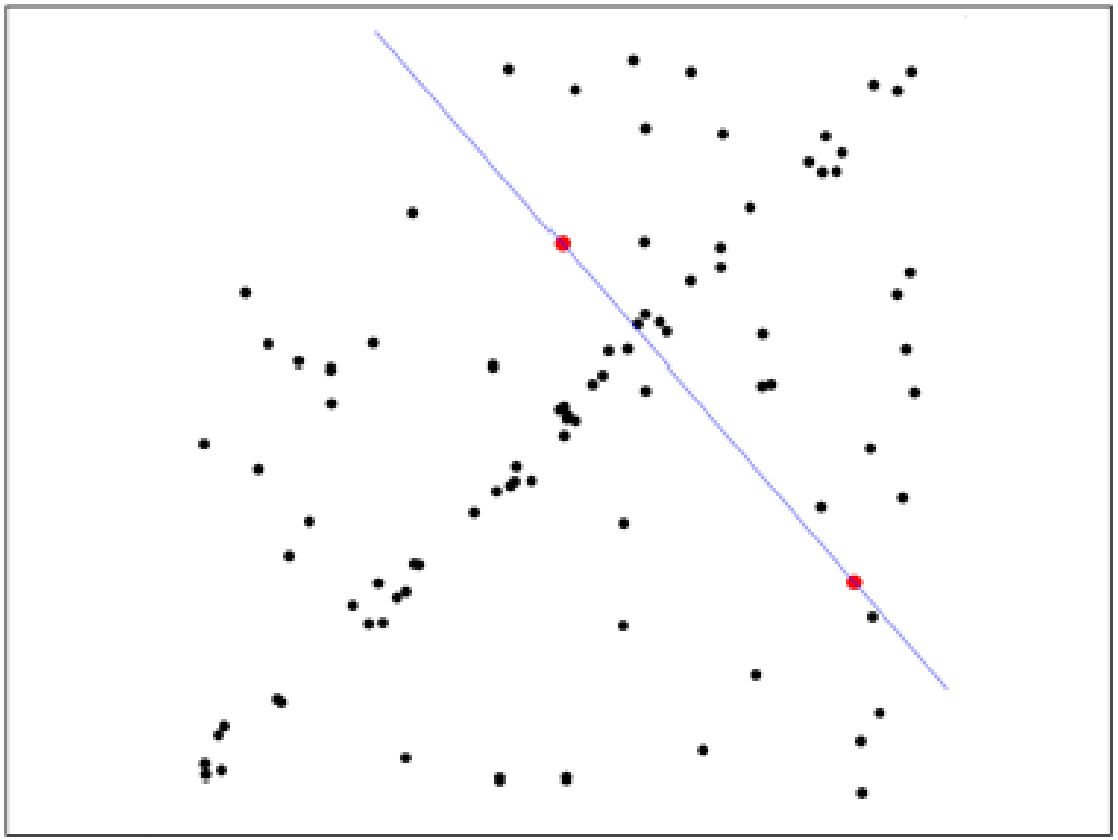
Least-squares fit

RANSAC Example



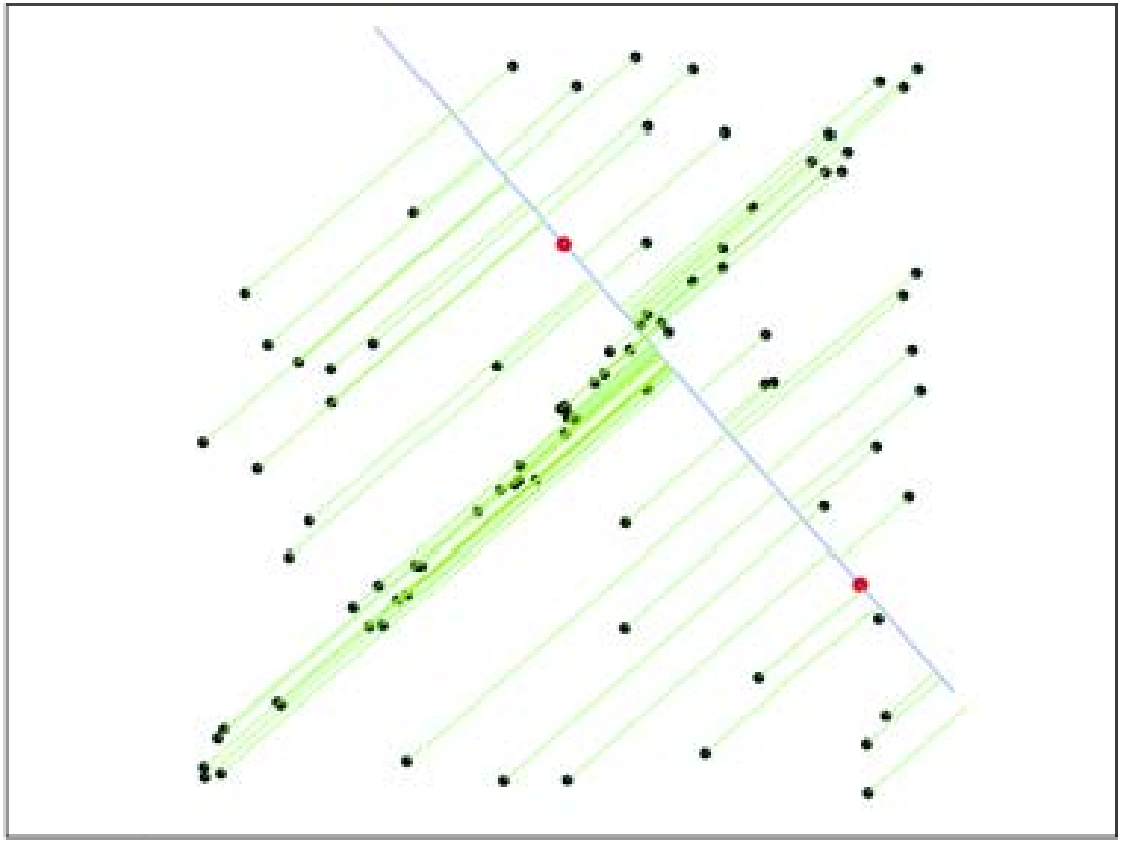
1. Randomly select minimal subset of points

RANSAC Example



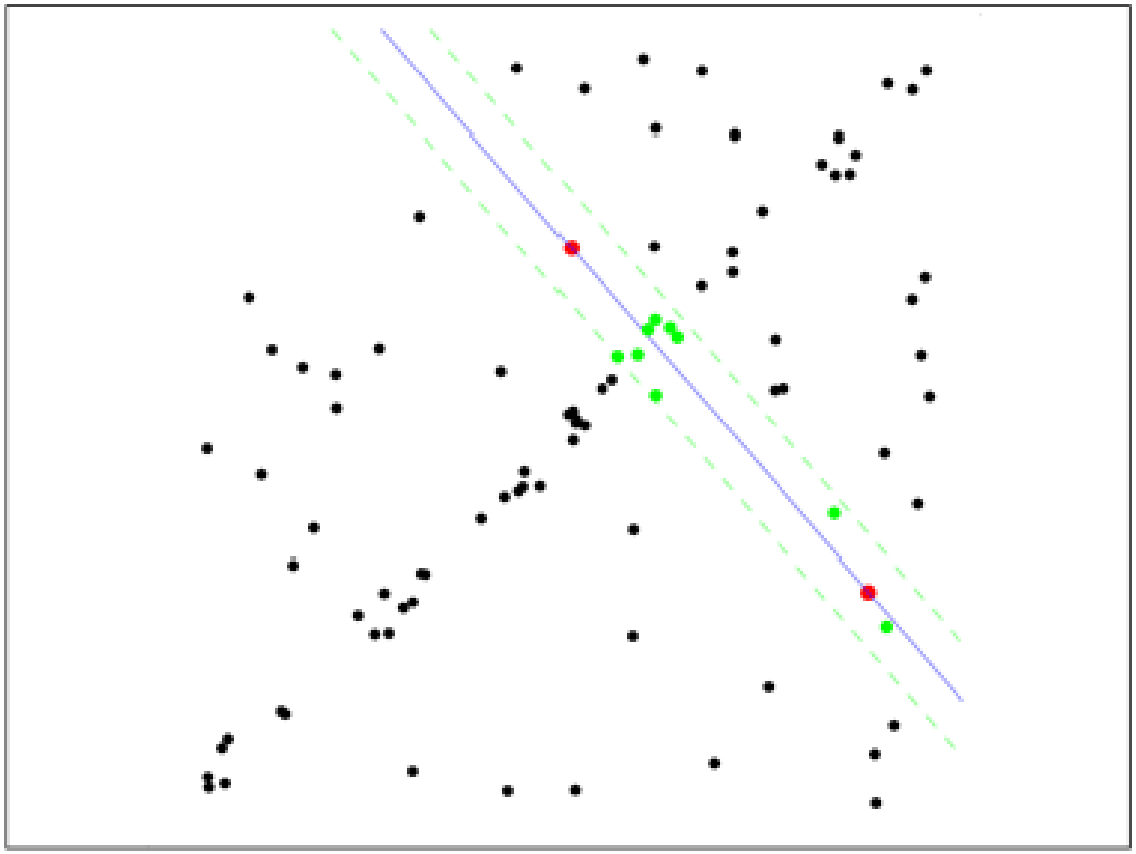
1. Randomly select minimal subset of points
2. Hypothesize a model

RANSAC Example



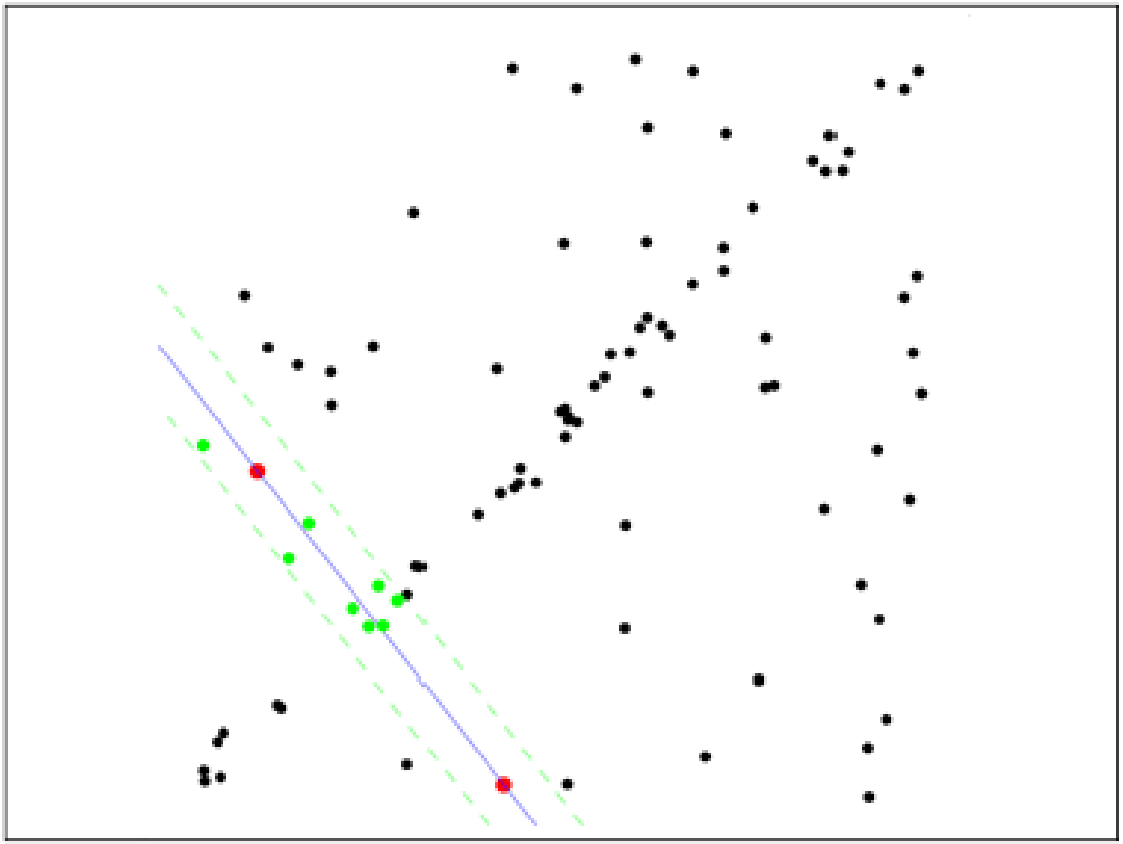
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function

RANSAC Example



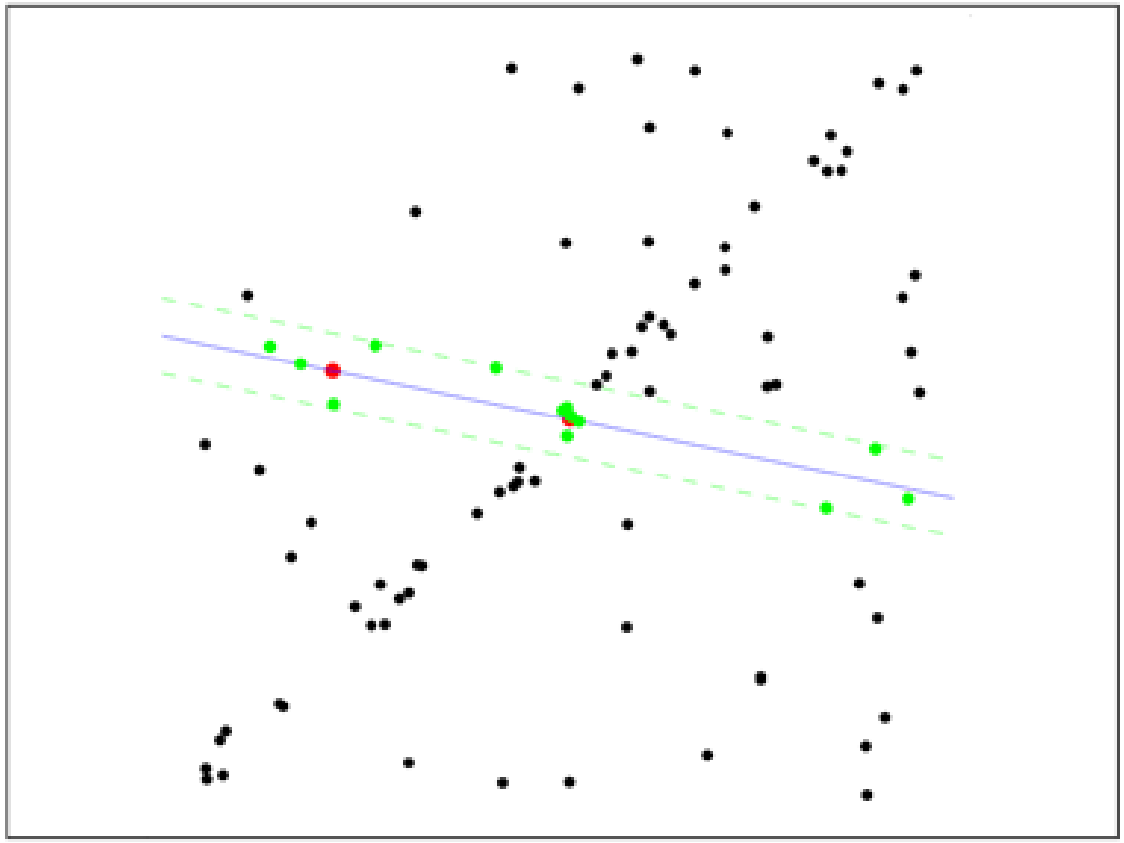
1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model

RANSAC Example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

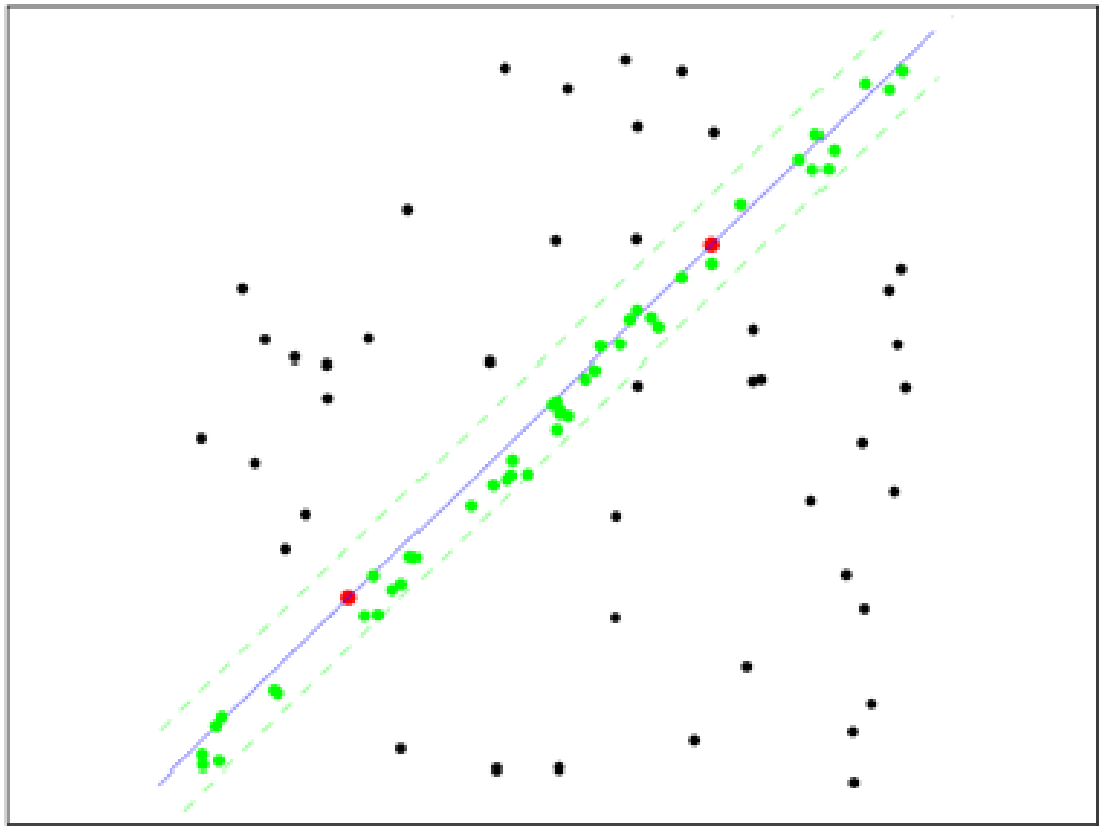
RANSAC Example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

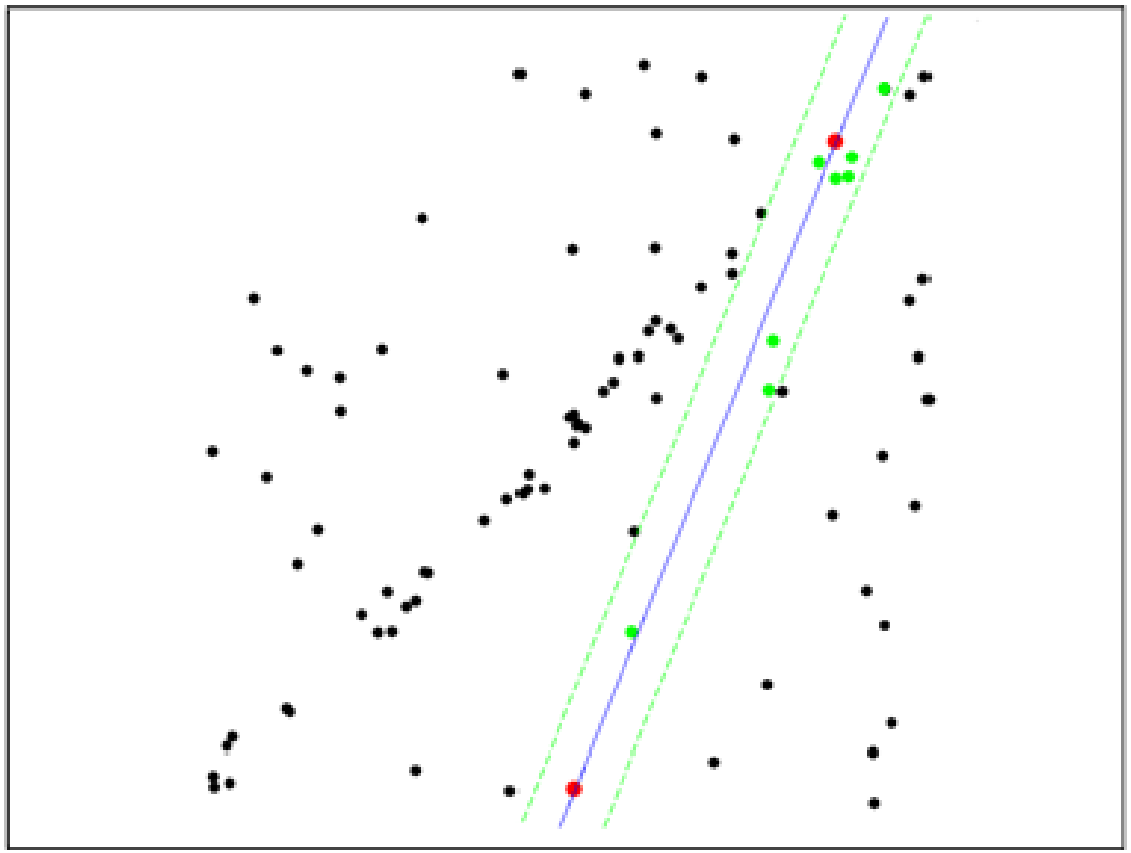
RANSAC Example

Uncontaminated sample



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

RANSAC Example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

RANSAC: for line fitting

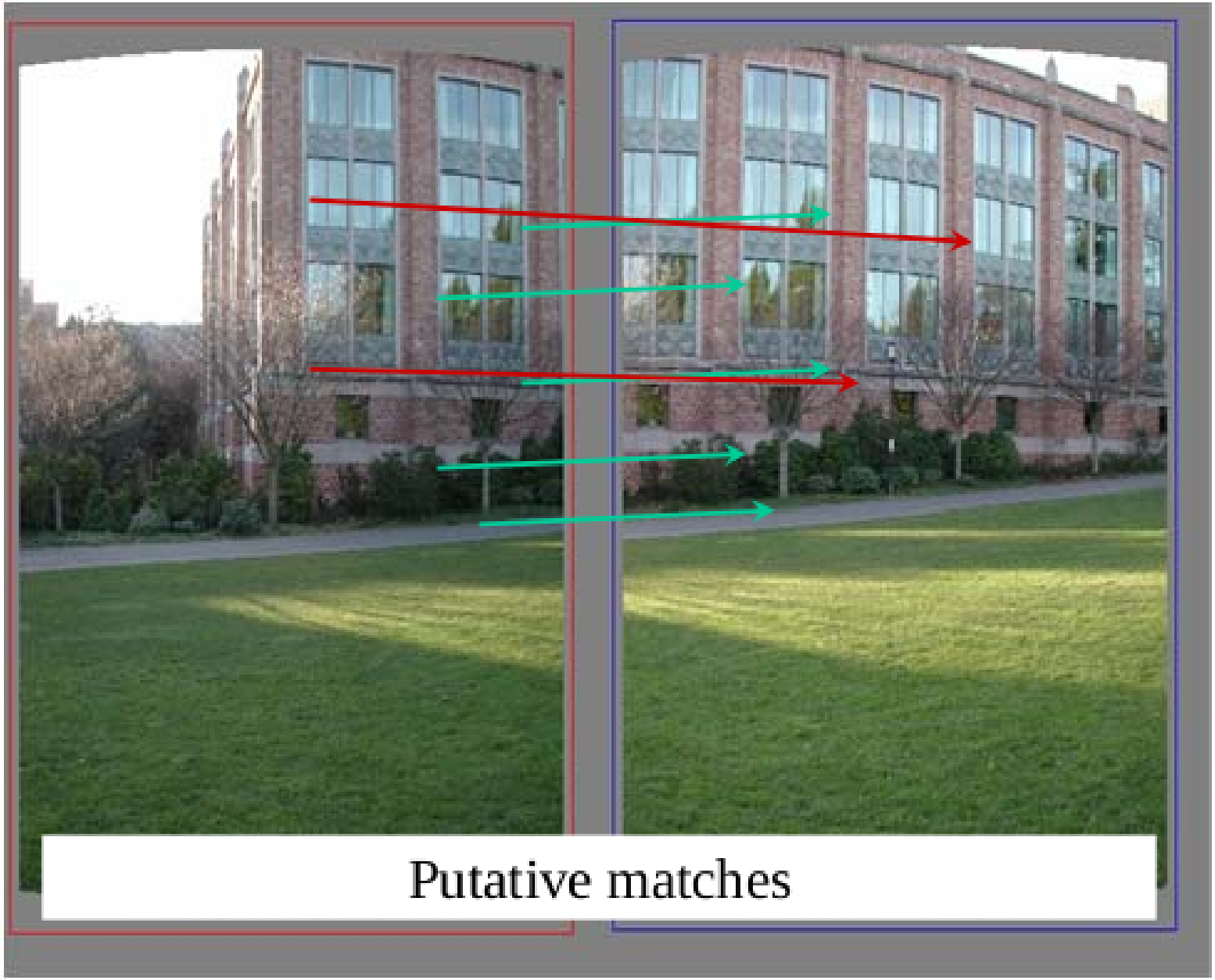
- Repeat **N** times
  1. Randomly select a **s** points at random
  2. Fit a line to these points
  3. Find the inliers to this line among the remaining points (i.e., points whose distance from the line is less than **t**)
  4. If there are **d** or more inliers, accept the line and refit using all the inliers

RANSAC: for line fitting

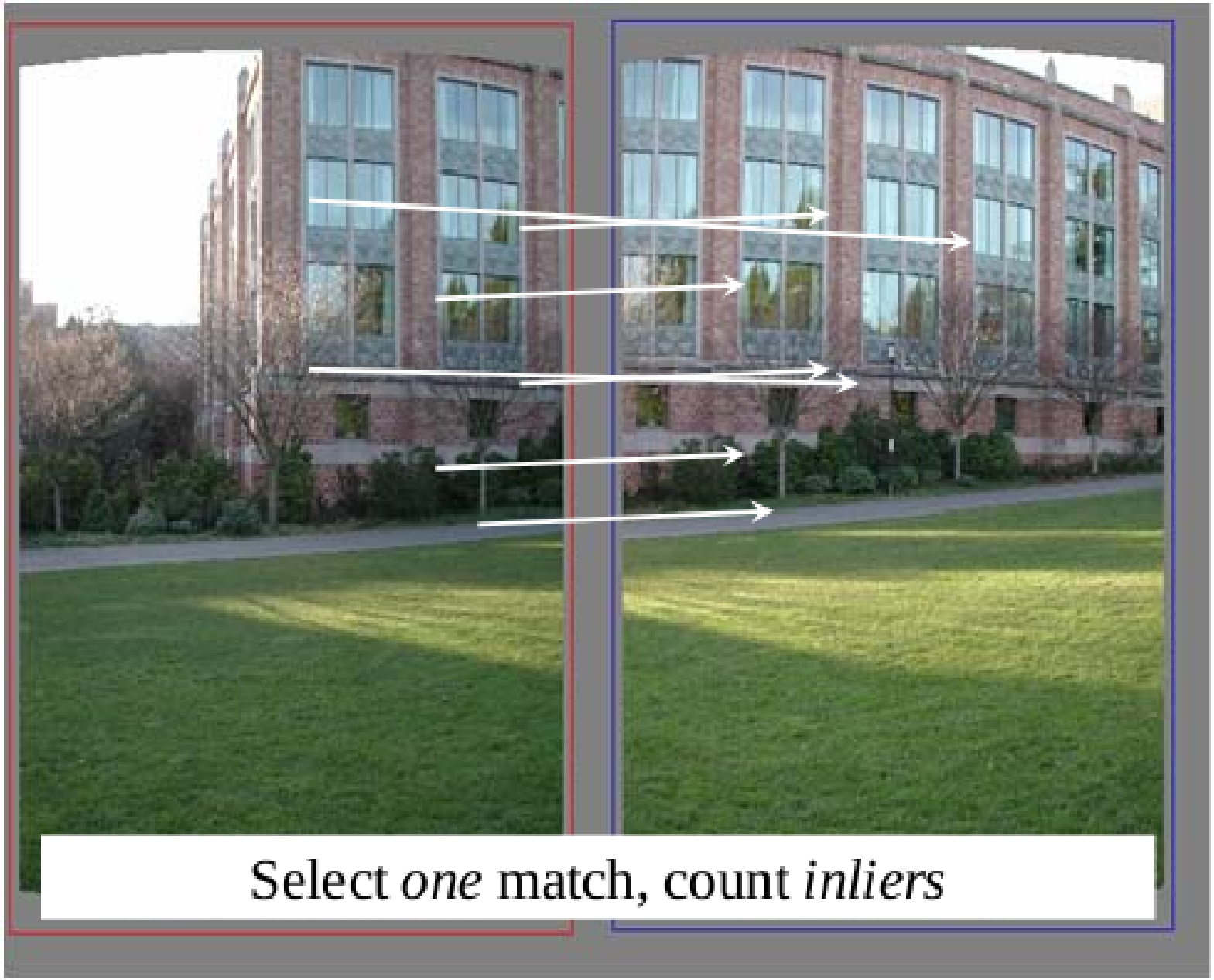
- This is an example of how to fit a model (line)!

- How would we do this for a general transformation?

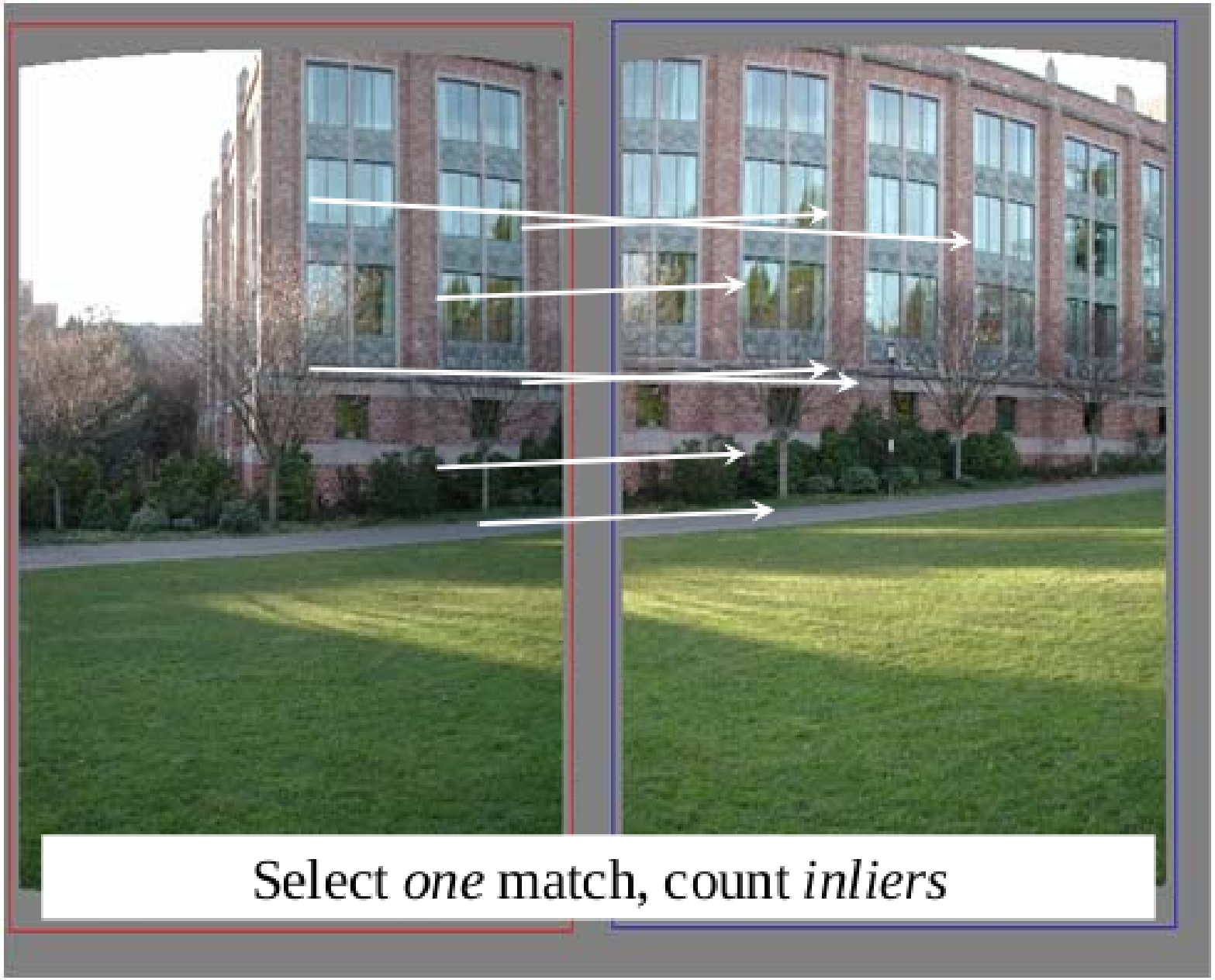
RANSAC Example: Translation



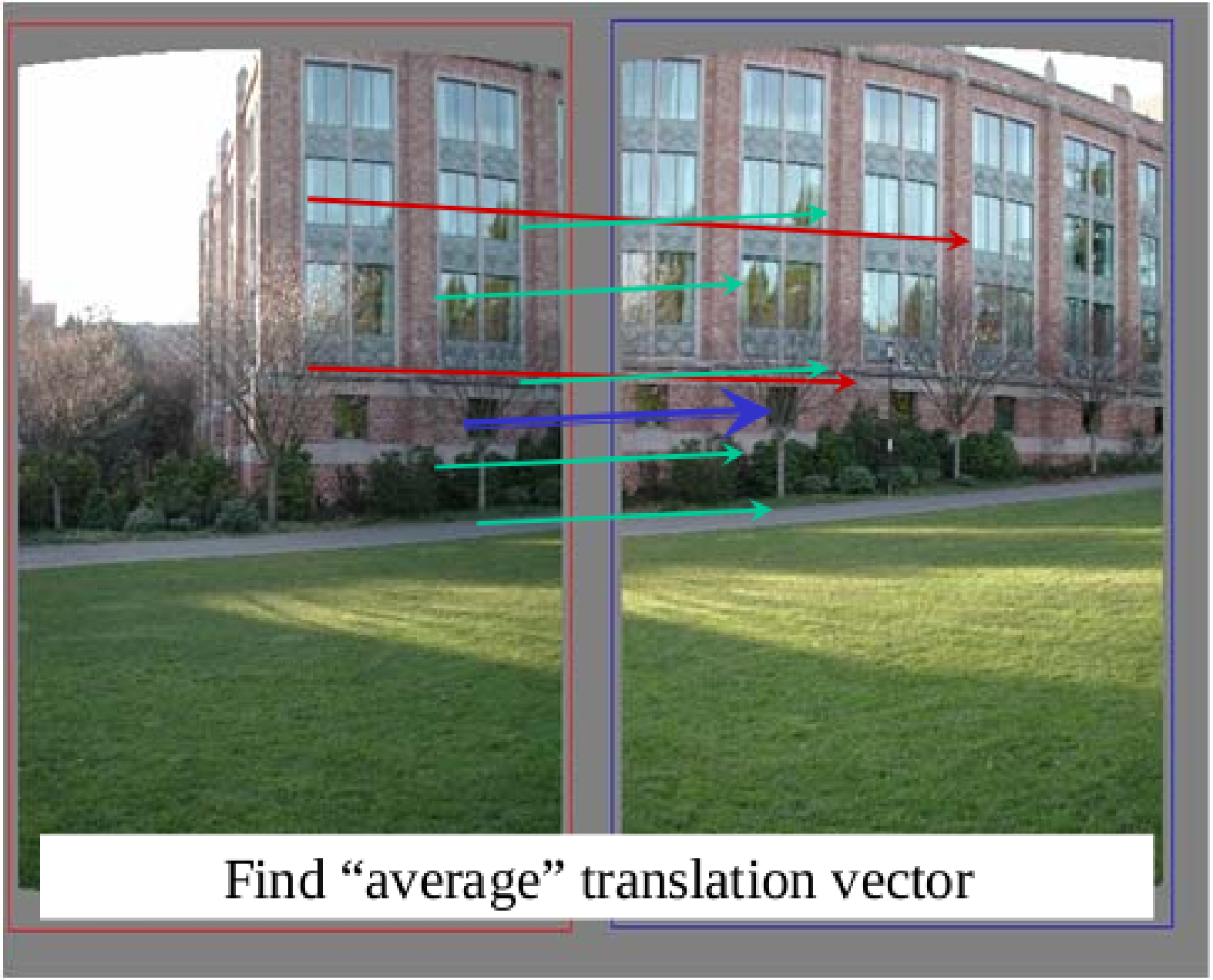
RANSAC Example: Translation



RANSAC Example: Translation



RANSAC Example: Translation

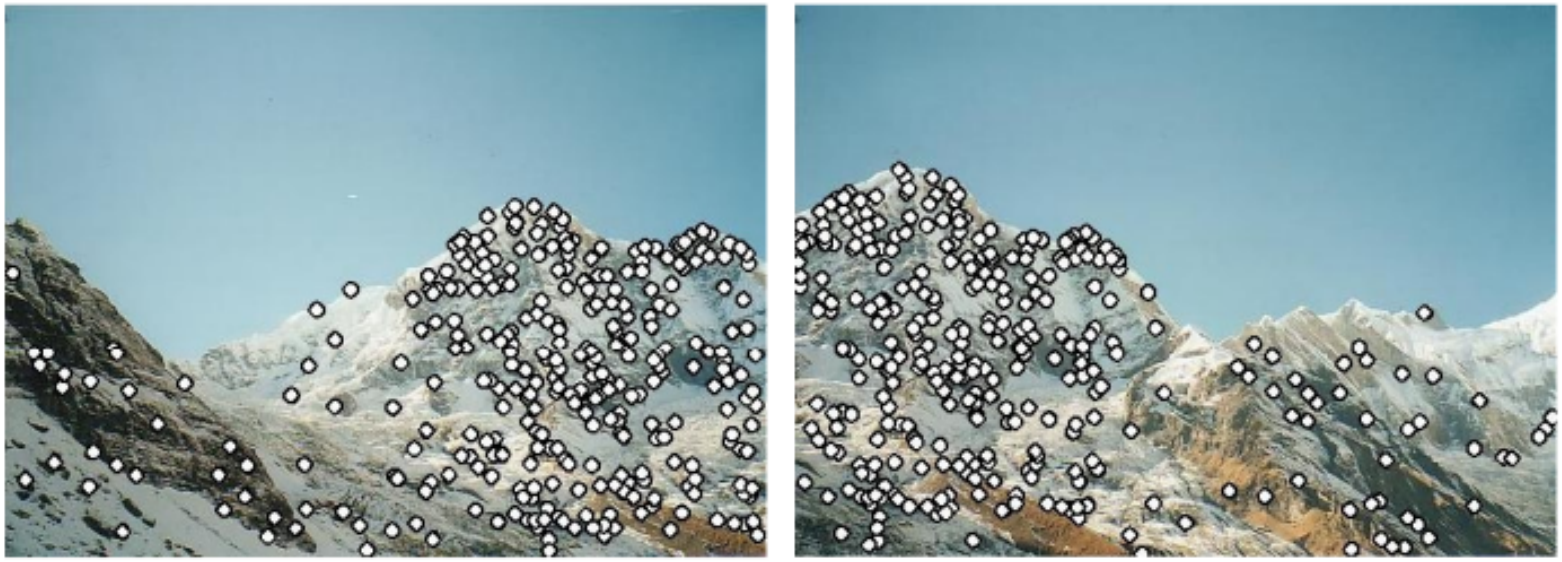


RANSAC pros and cons

- Pros
  - Simple and general
  - Applicable to many different problems
  - Often works well in practice
- Cons
  - Lots of parameters to tune
  - Doesn't work well for low inlier ratios (too many iterations or can fail completely)
  - Can't always get a good initialization of the model based on the minimum number of samples

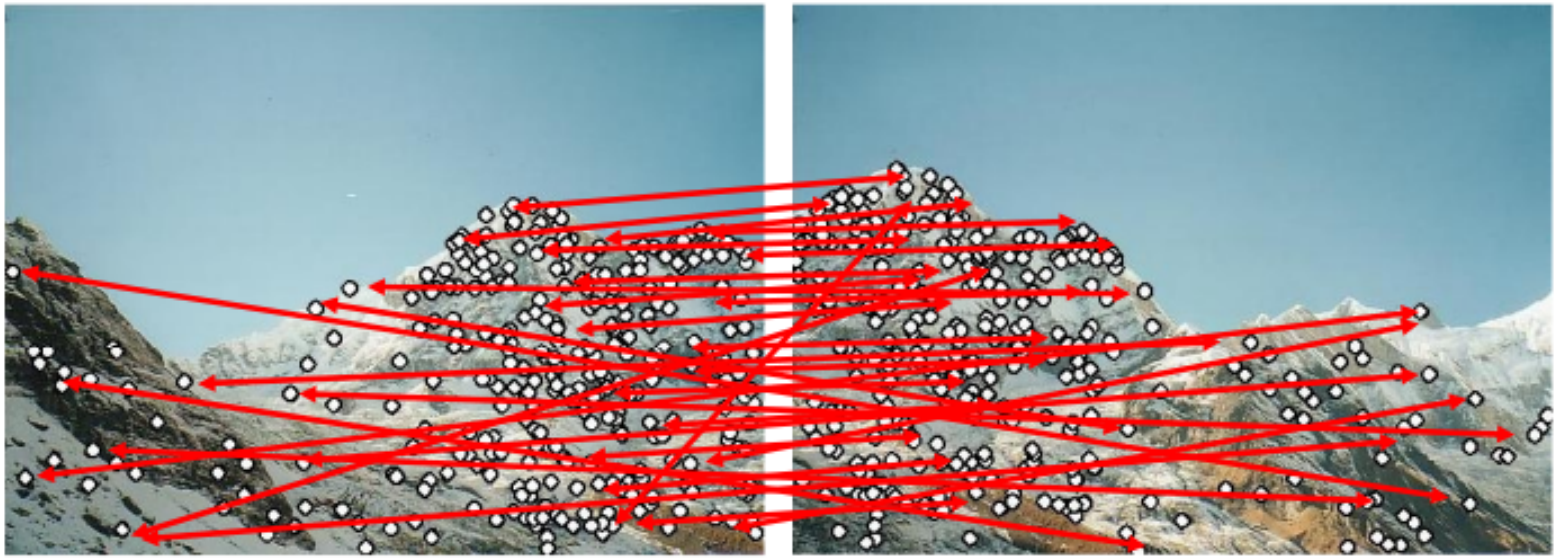
Recap: Feature-based alignment

Recap: Feature-based alignment



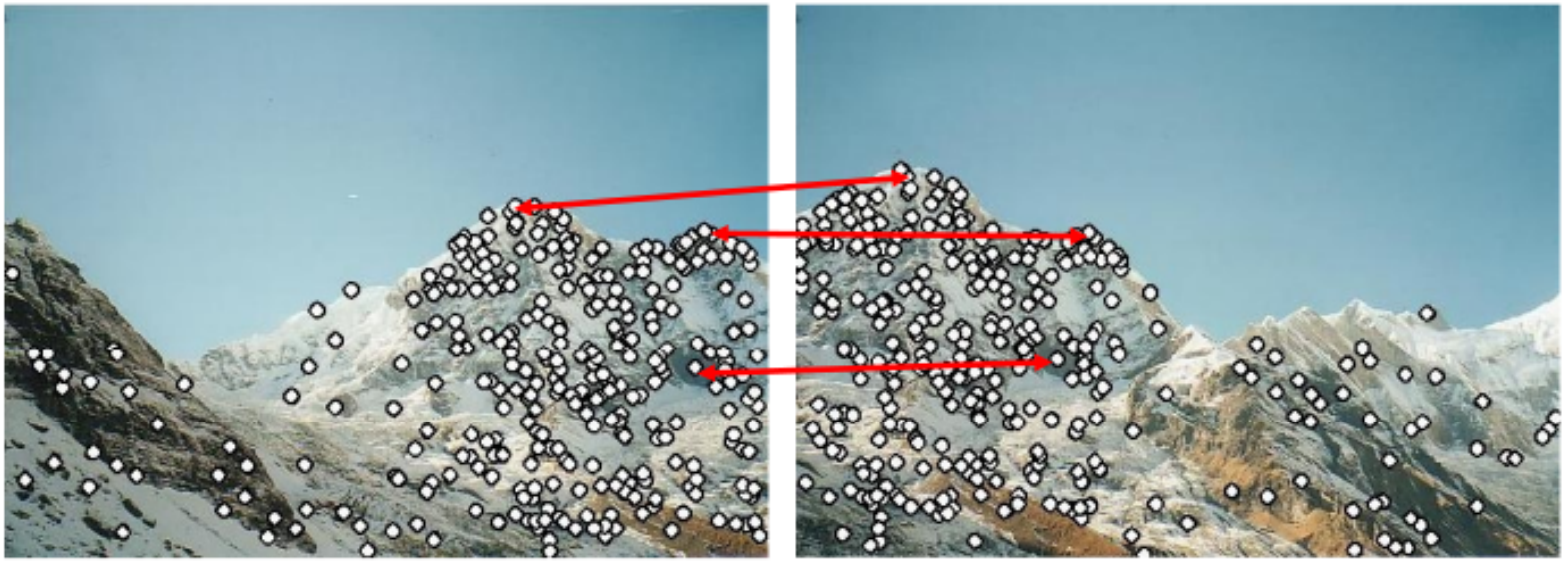
- Extract features

Recap: Feature-based alignment



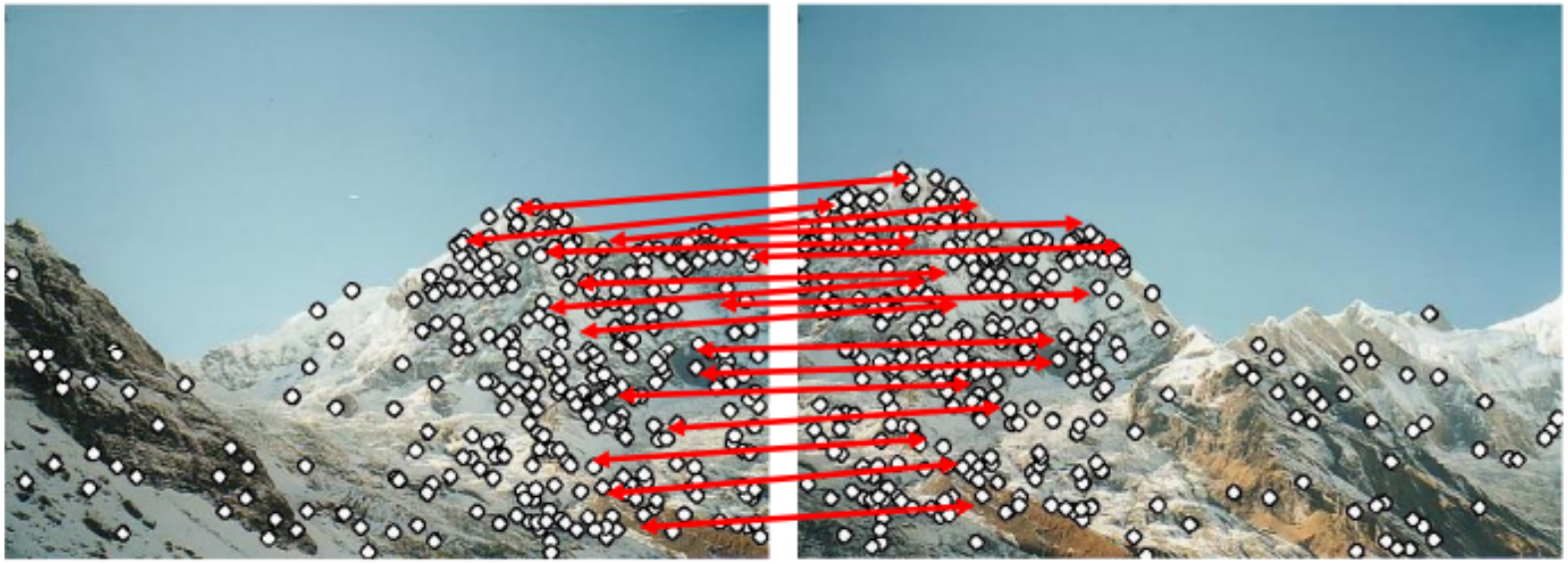
- Extract features
- Compute *punitive* matches

Recap: Feature-based alignment



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

Recap: Feature-based alignment



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

Recap: Feature-based alignment

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

Image blending - feathering

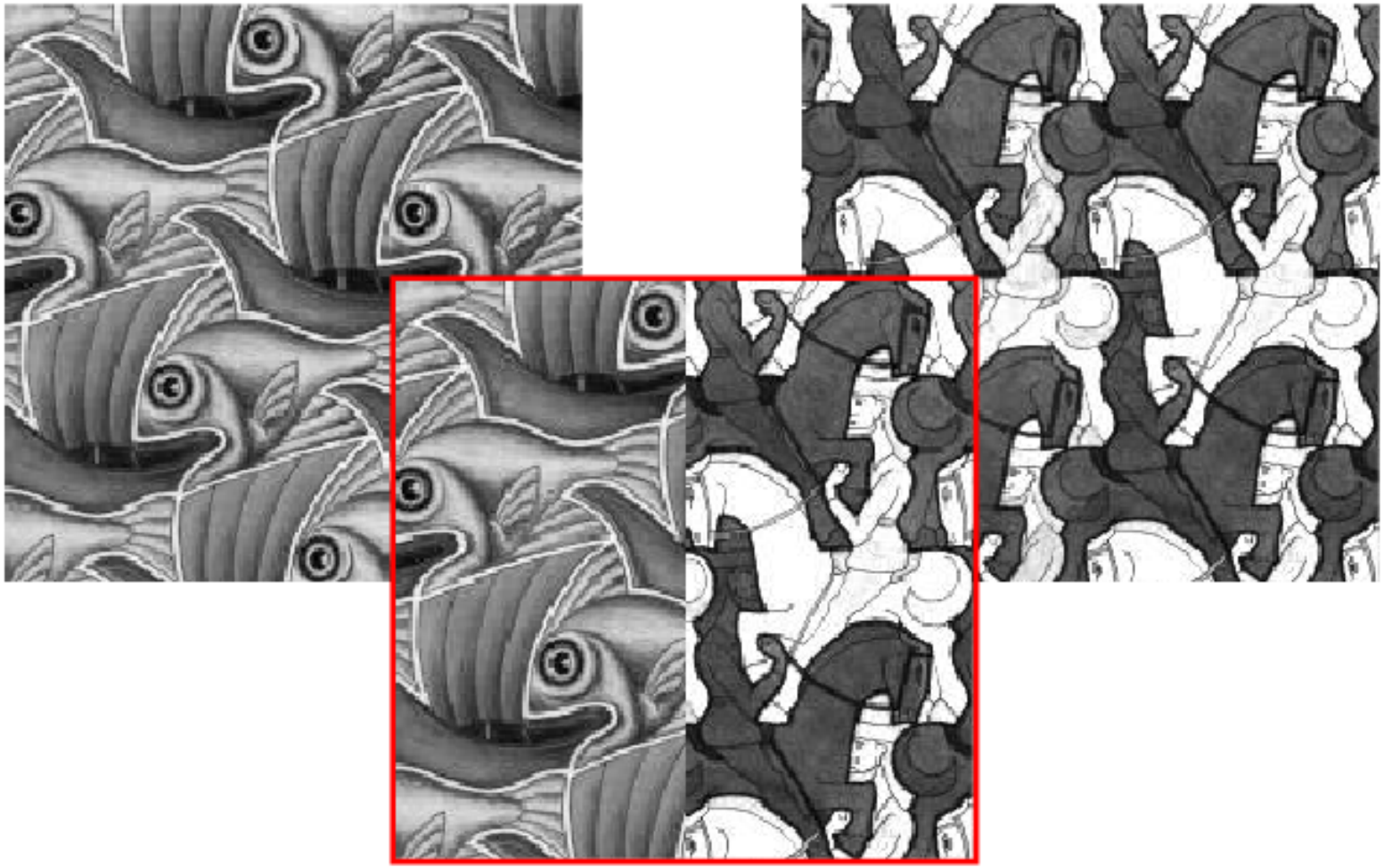


Image blending - feathering

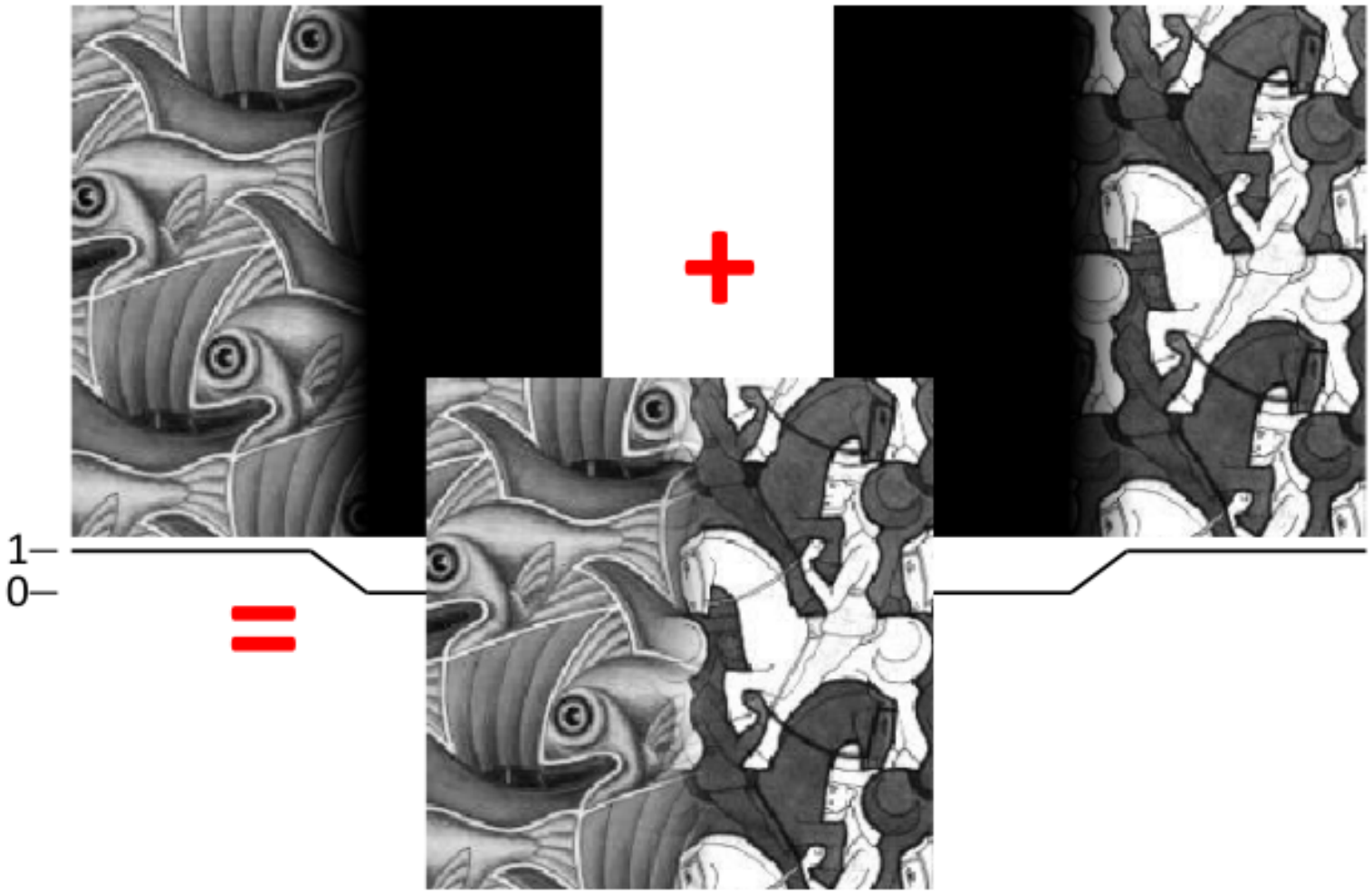
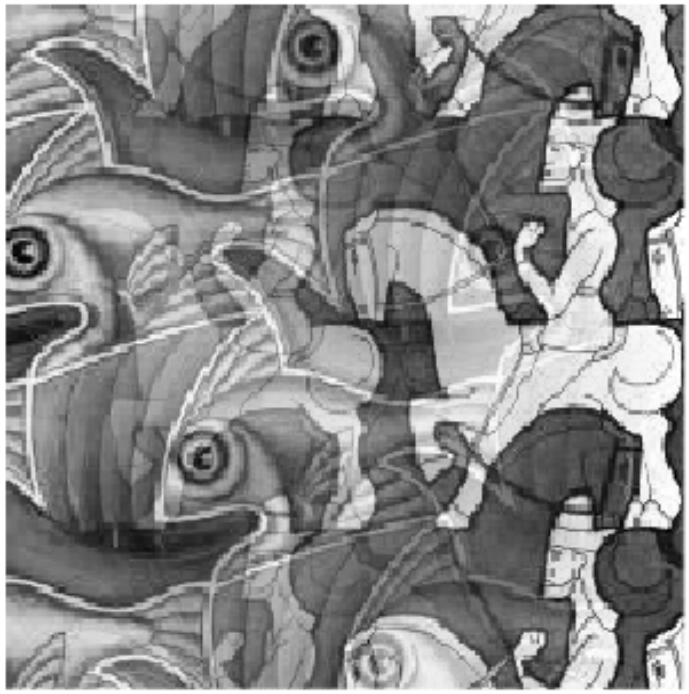
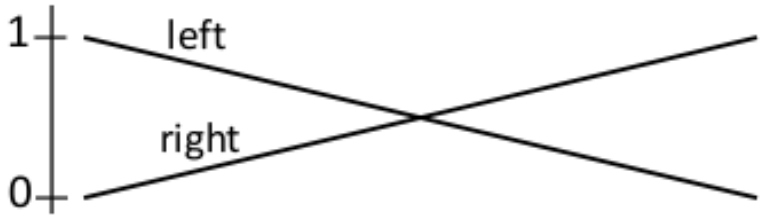
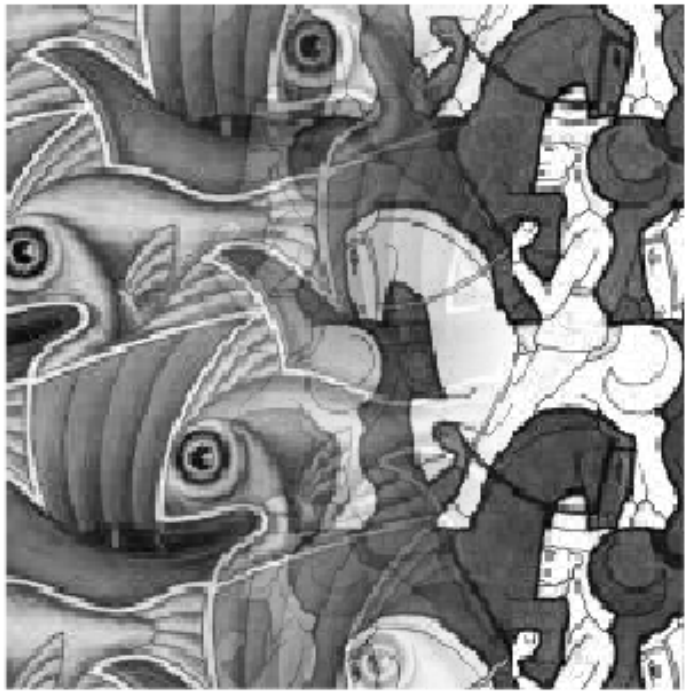


Image blending - feathering







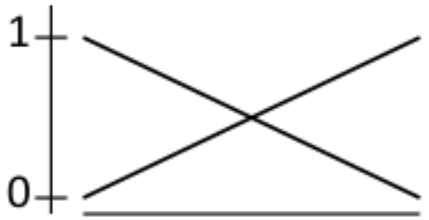
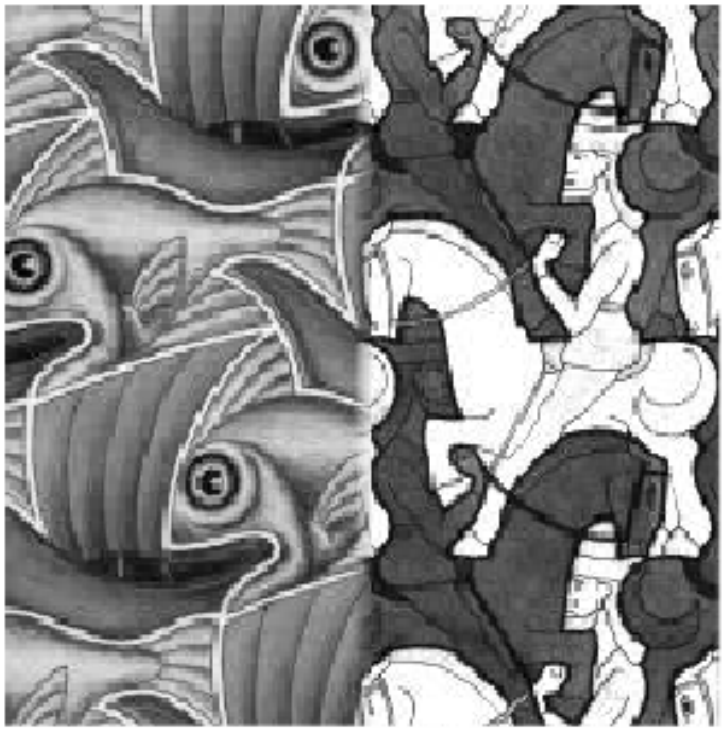


Image blending - feathering



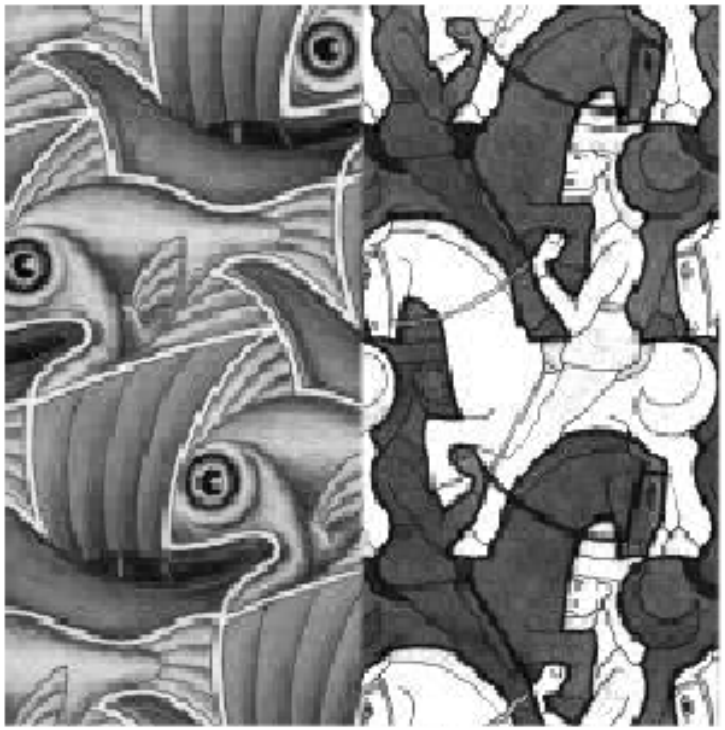
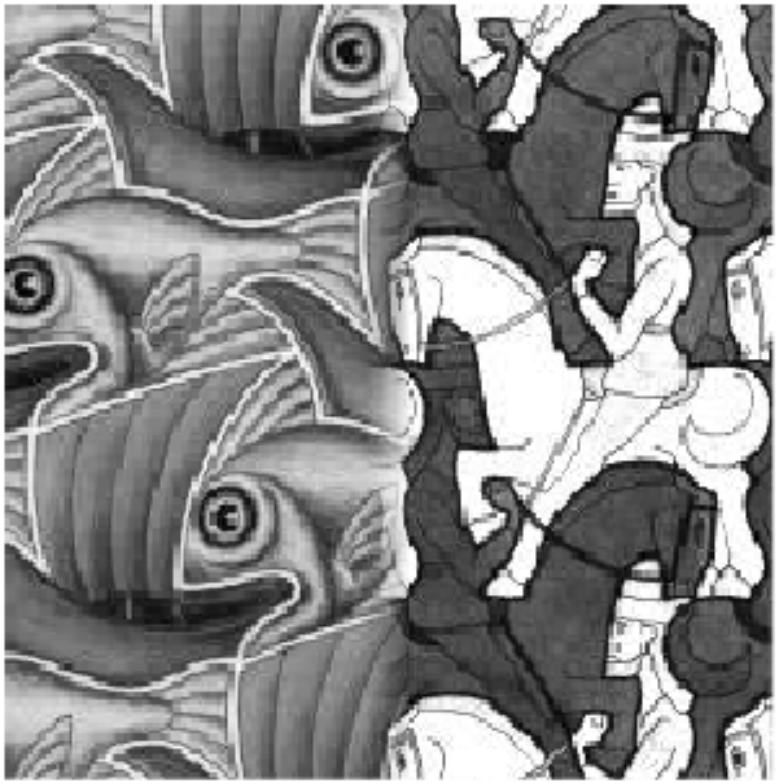


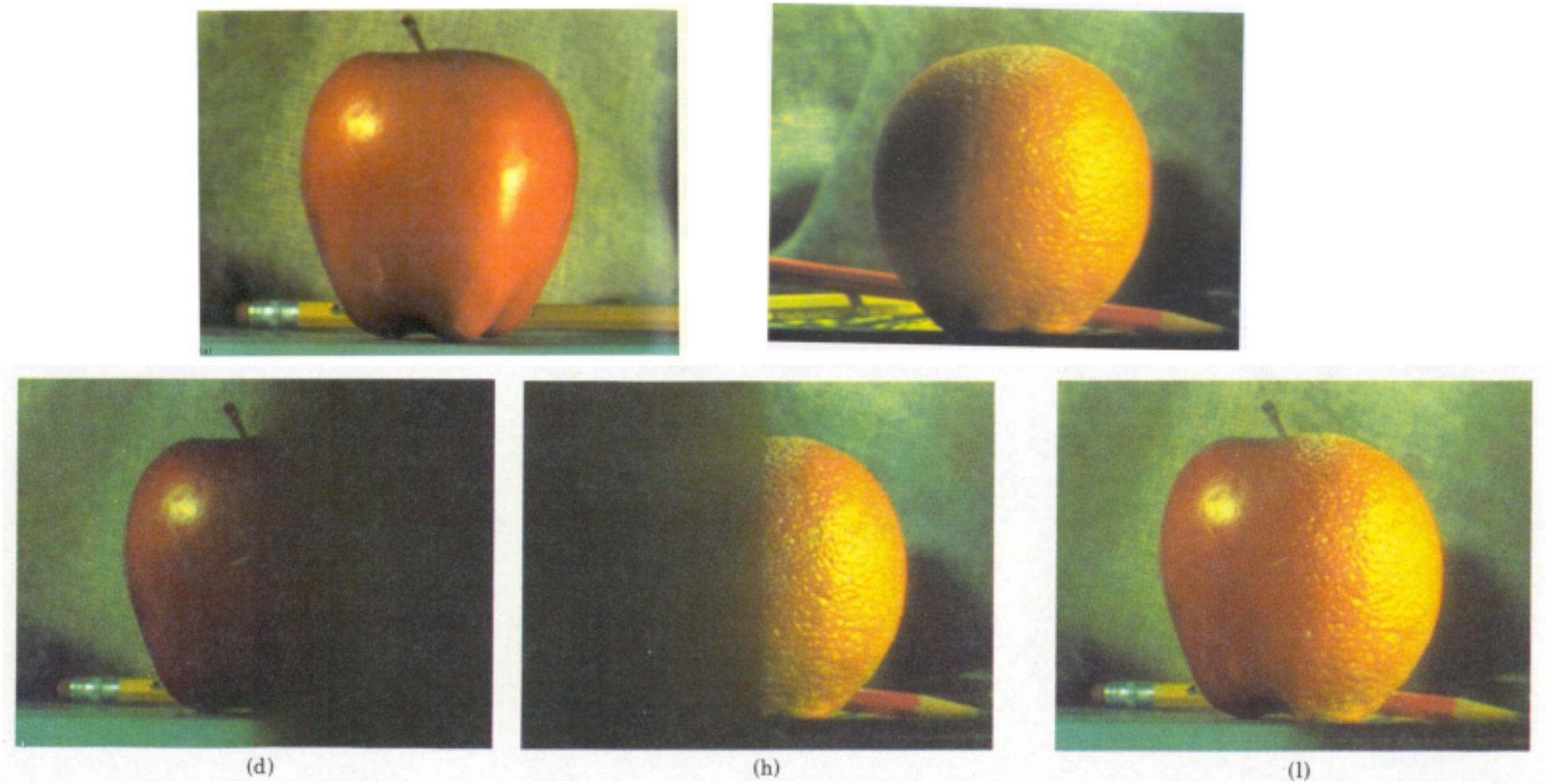
Image blending - feathering



"Optimal" window: smooth but not ghosted

- Doesn't always work...

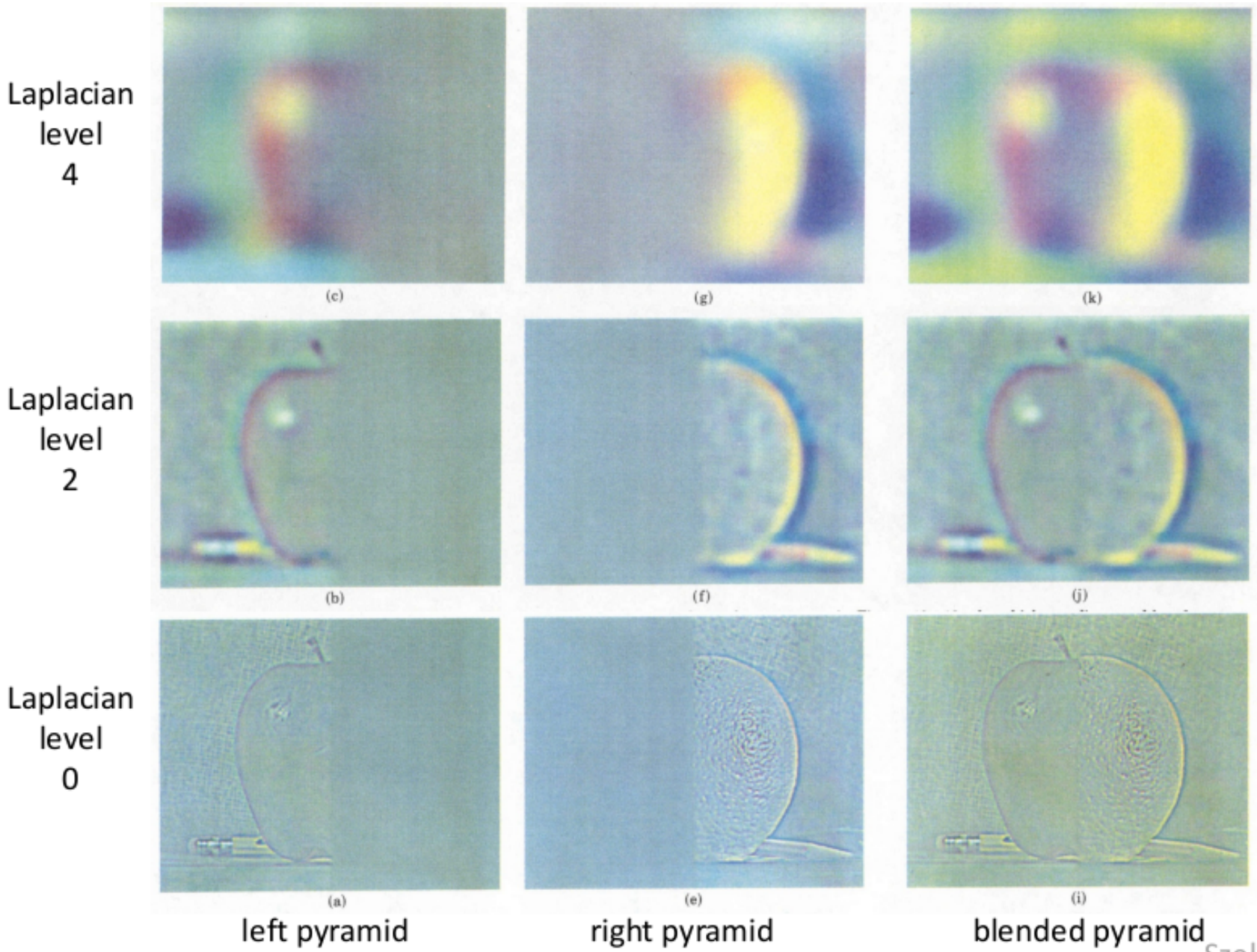
Image blending - pyramid blending



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

Image blending - pyramid blending



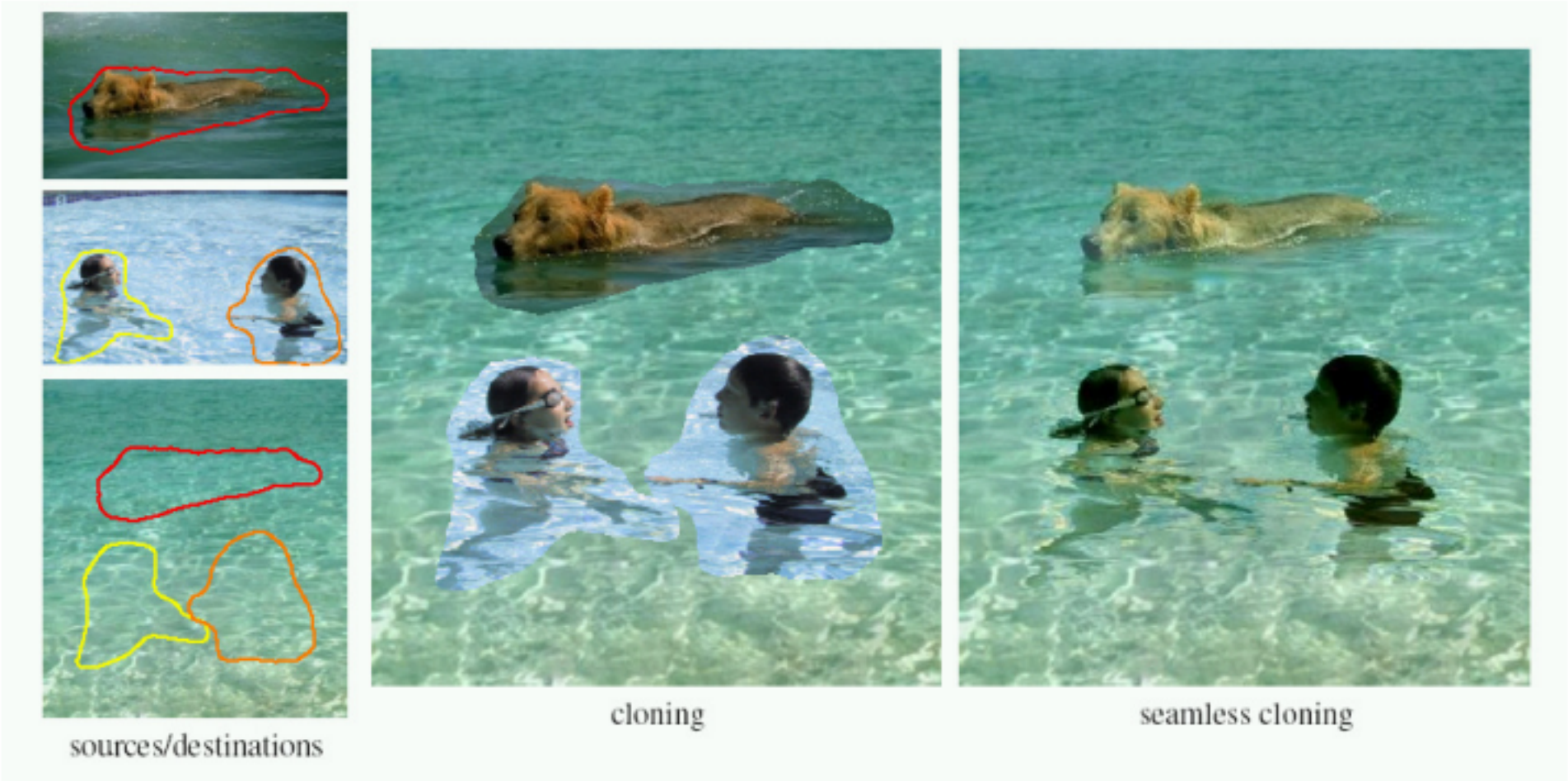
Szeliski

Image blending - pyramid blending

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?

Image blending - Poisson Image Editing



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

Slide credits

- Trever Darrell
- Kristen Grauman
- Forsyth and Ponce
- Rich Szeliski
- David Lowe
- Mathew Brown