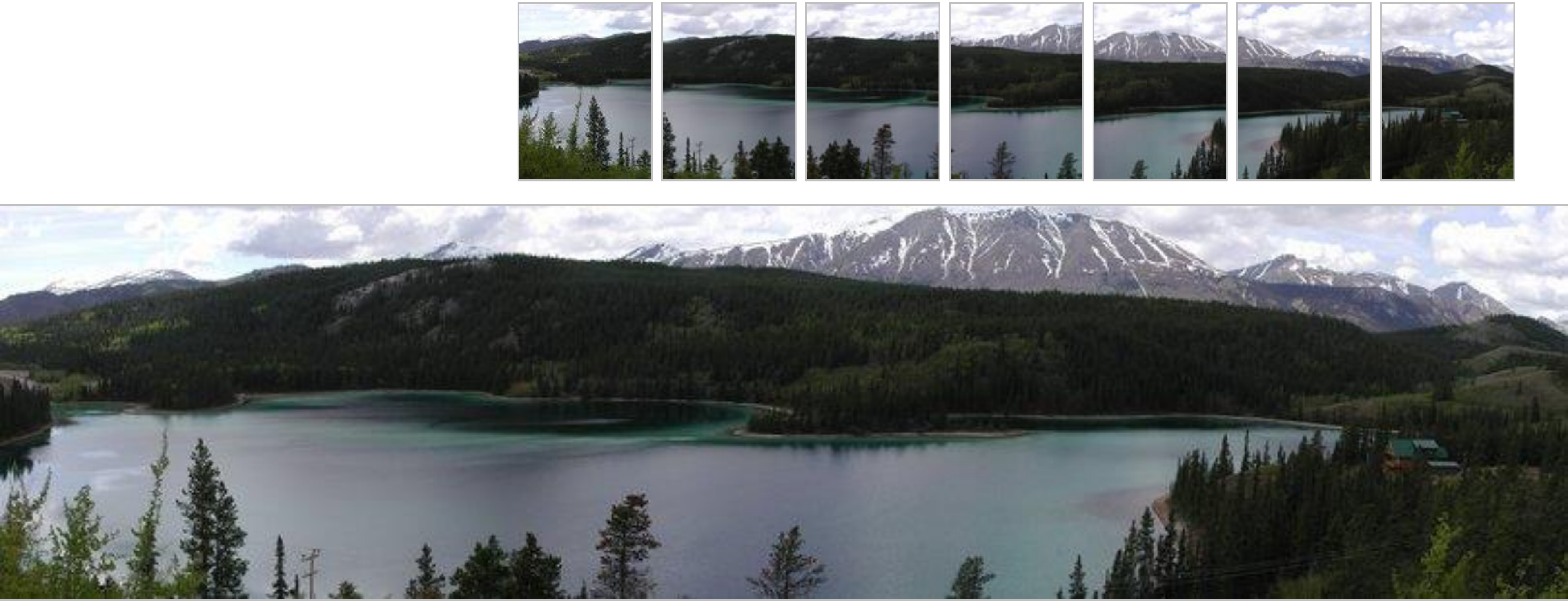
Image Stitching

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



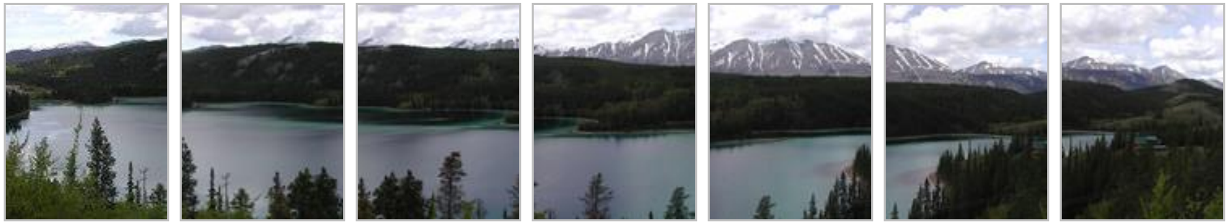
Slide credit: [Vaibhav Vaish]

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

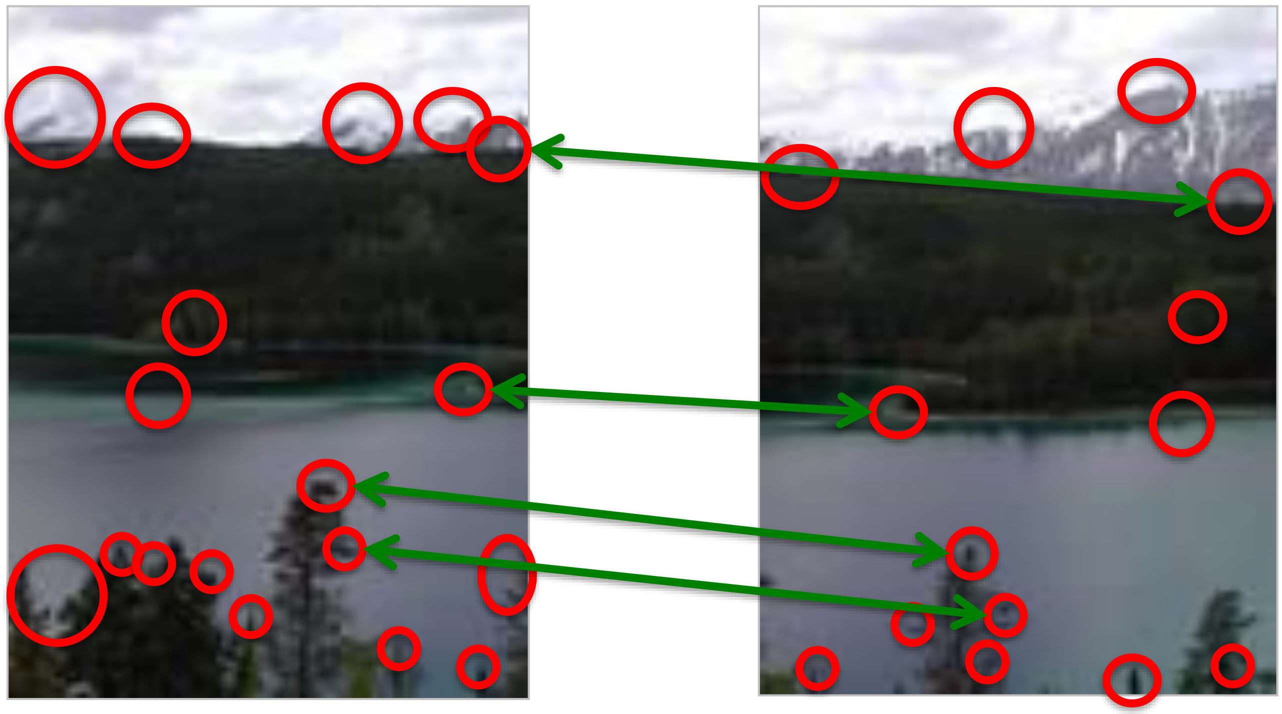
1. Take a sequence of images from the same position

- Rotate the camera about its optical center



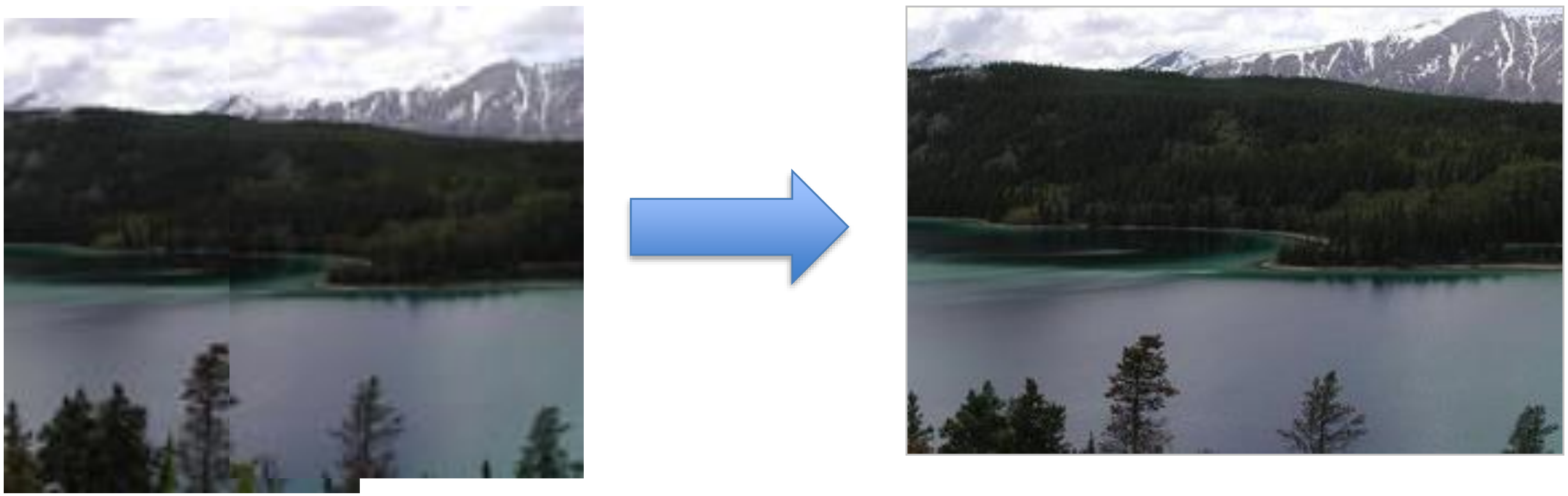
2. Compute transformation between images

- Extract interest points
- Find Matches
- Compute transformation ?

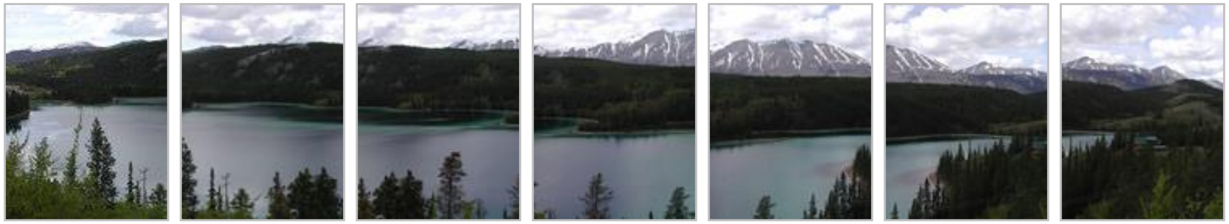


3. Shift the images to overlap

4. Blend the two together to create a mosaic



5. Repeat for all images



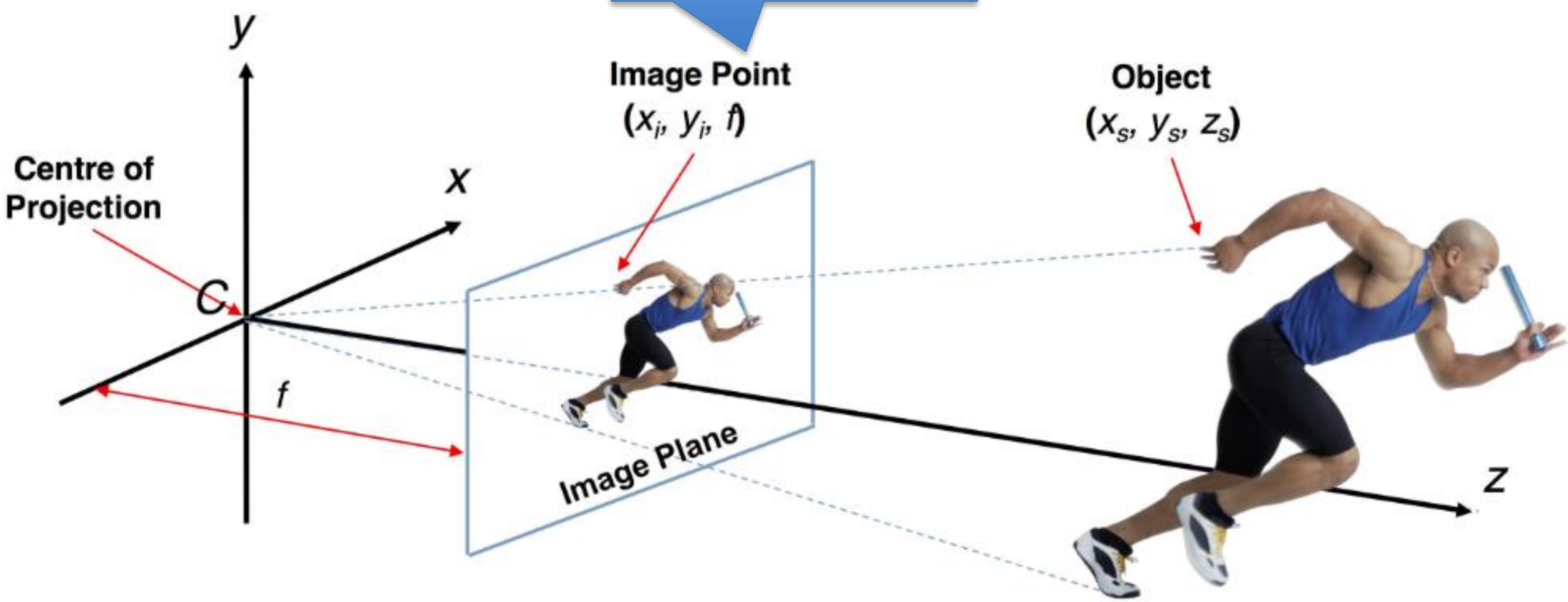
לפני שנלמד על טרנספורמציה בין תמונות

Homogeneous coordinates

מה האינטואיציה מאחורי הקורדינאטות ההומוגניות?

Homogeneous coordinates

נקודה אשר חותכת
את המישור ב (x,y,1)



Homogeneous coordinates

2D Points:

$$p = \begin{bmatrix} x \\ y \end{bmatrix} \longrightarrow p' = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad p' = \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \longrightarrow p = \begin{bmatrix} x'/w' \\ y'/w' \end{bmatrix}$$

Projective plan

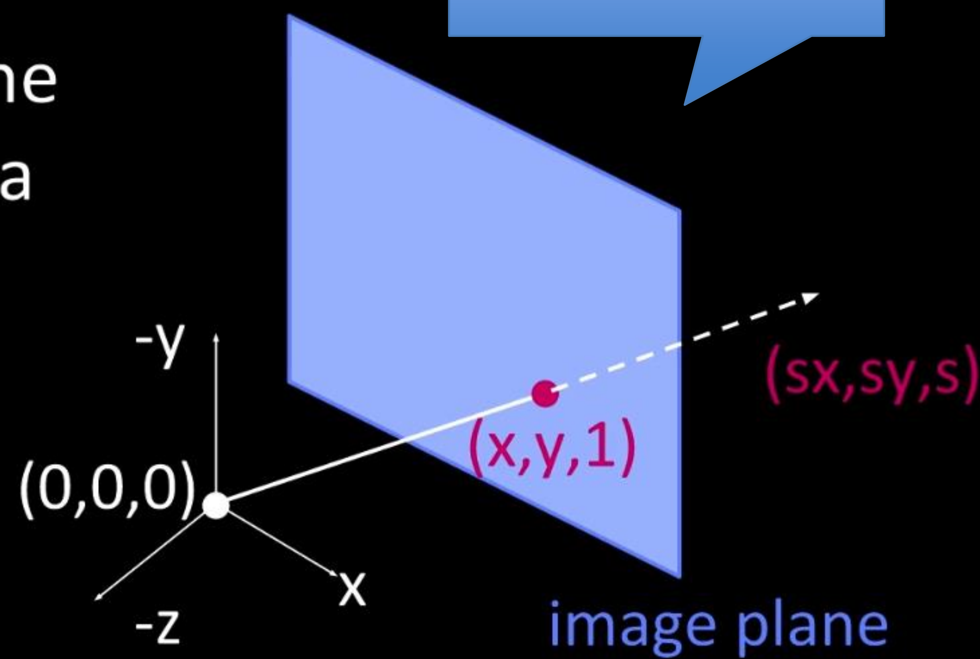
The projective plane

Each *point* (x,y) on the plane (at z=1) is represented by a *ray* (sx,sy,s)

All points on the ray are equivalent:

$$(x, y, 1) \cong (sx, sy, s)$$

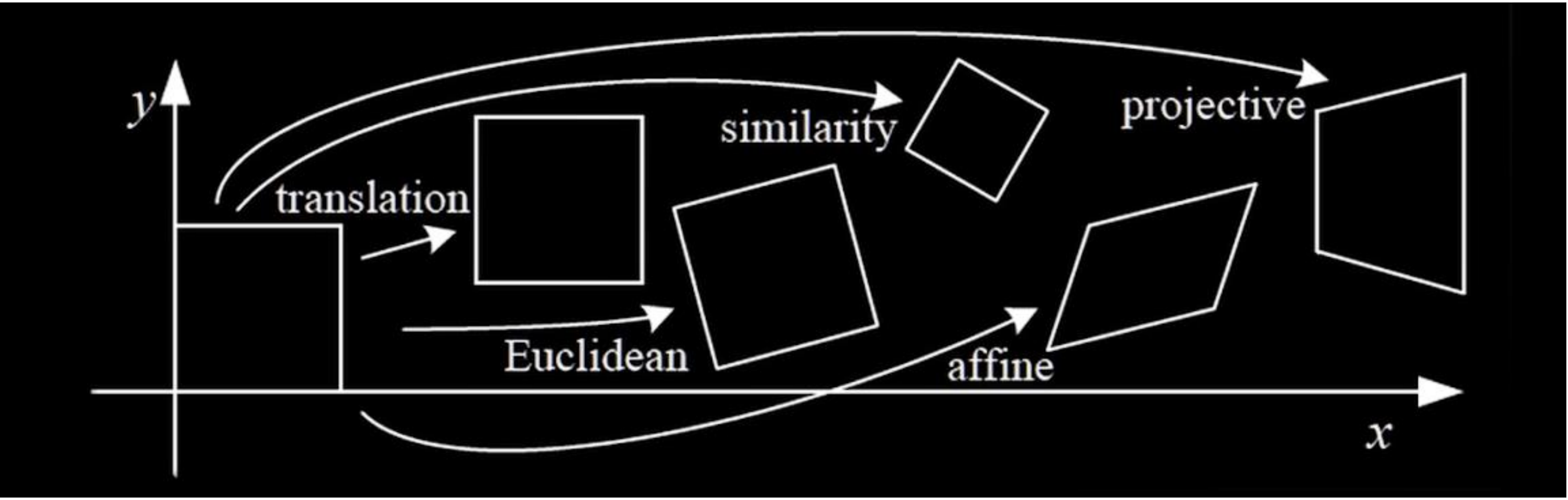
נקודה אשר חותכת
את המישור ב (x,y,1)

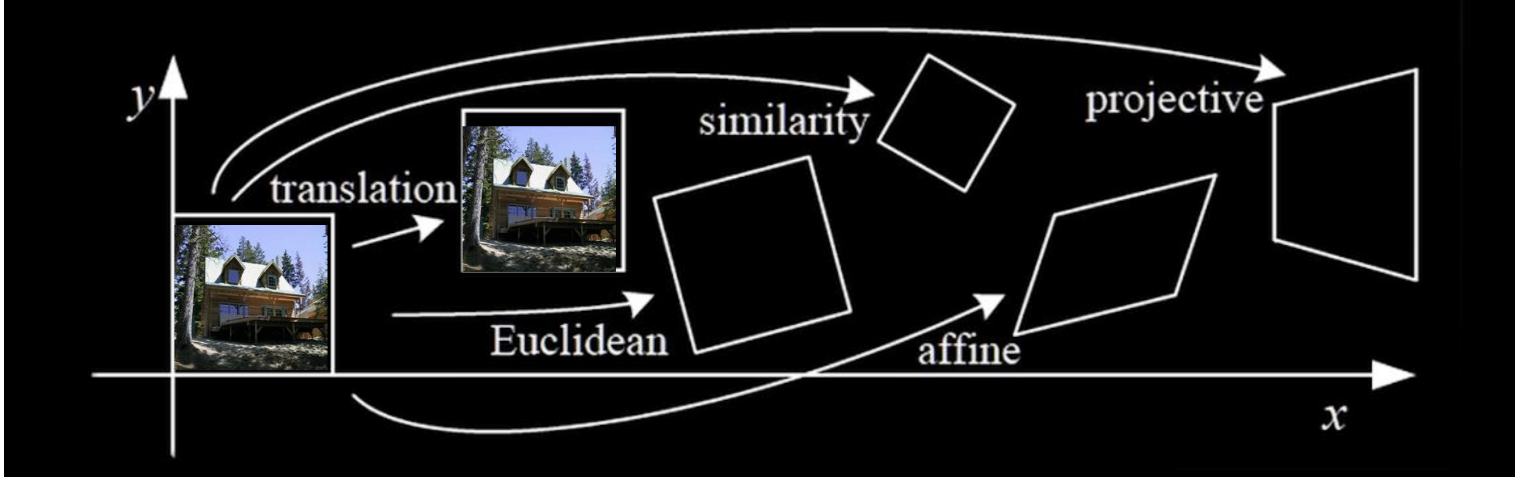


נקודות לאורך הישר מקיימות

טרנספורמציה בין תמונות

טרנספורמציה בין תמונות

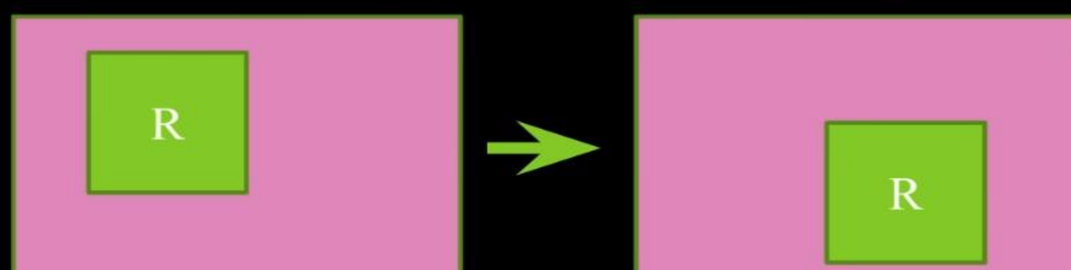


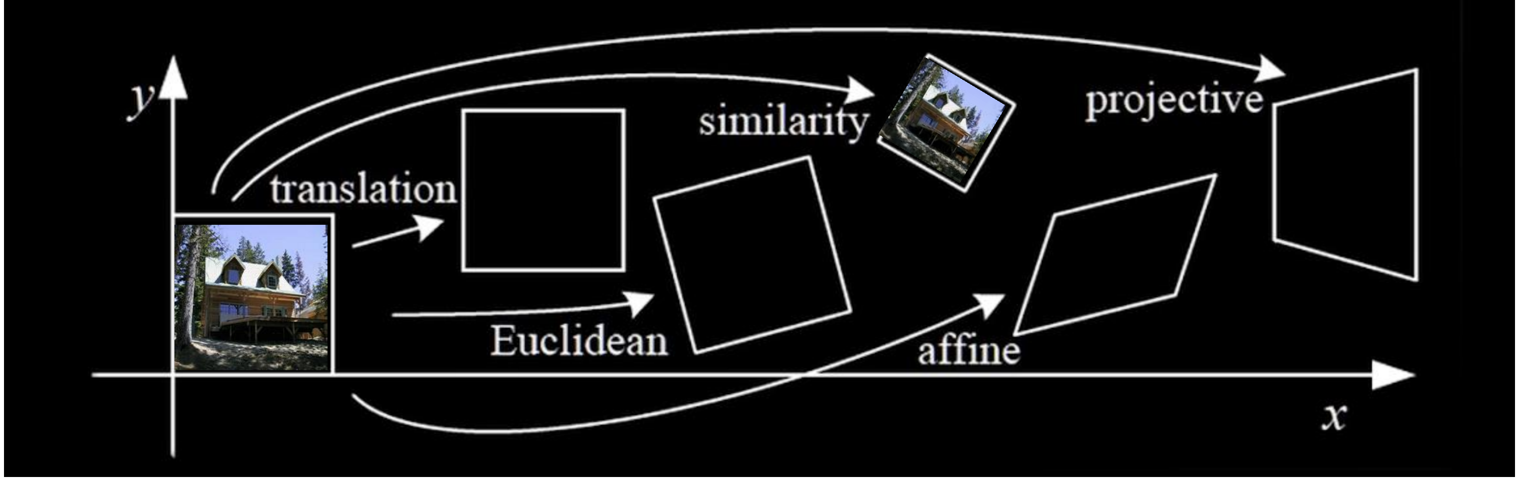


- Translation

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

- Preserves:
  - Lengths/Areas
  - Angles
  - Orientation
  - Lines

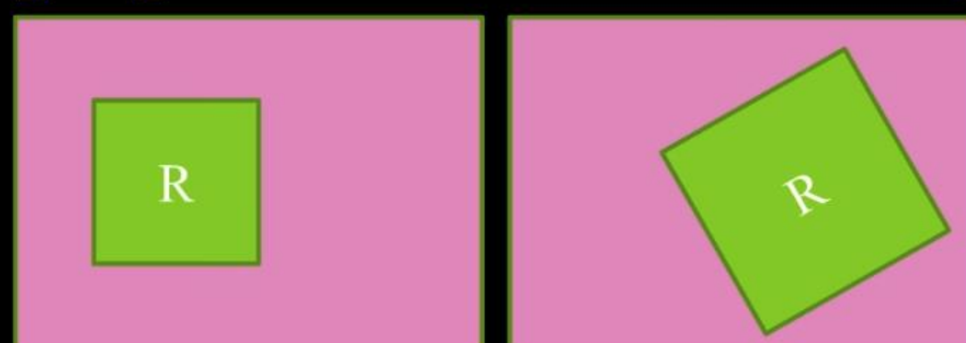


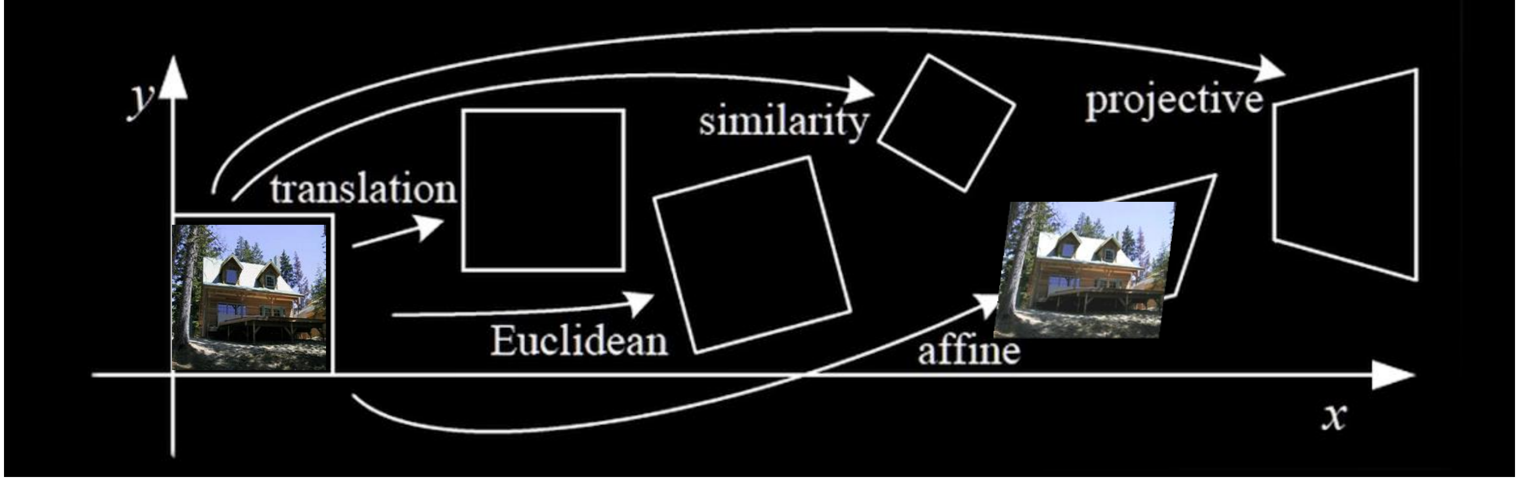


- Similarity (trans, rot, scale) transform

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a\cos(\theta) & -a\sin(\theta) & t_x \\ a\sin(\theta) & a\cos(\theta) & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Preserves:
  - Lengths/Areas
  - Angles
  - Lines

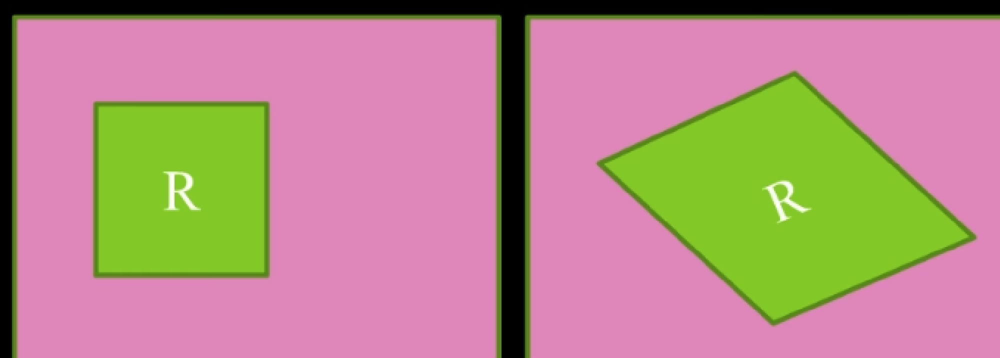


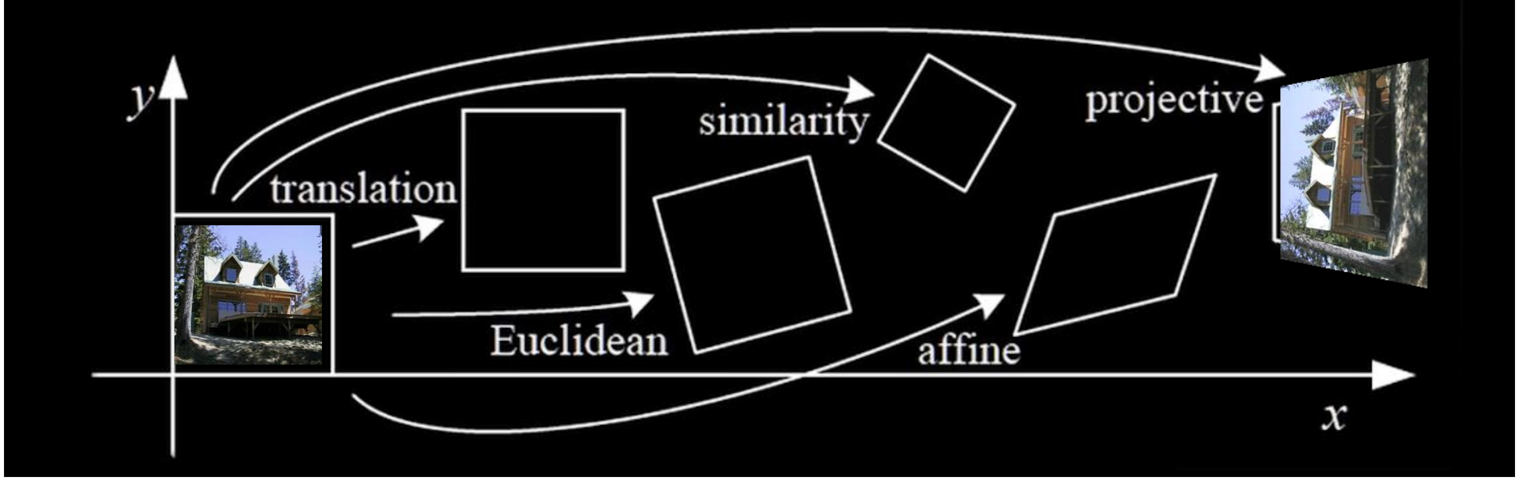


- Affine transform

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

- Preserves:
  - Parallel Lines
  - Ratio of Areas
  - Lines

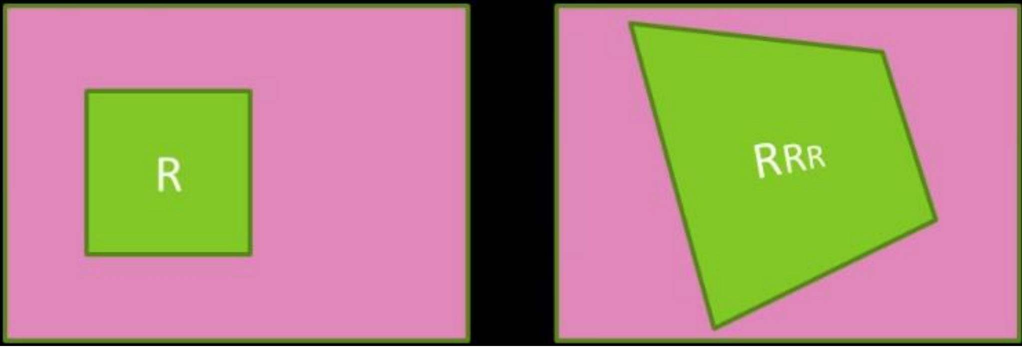




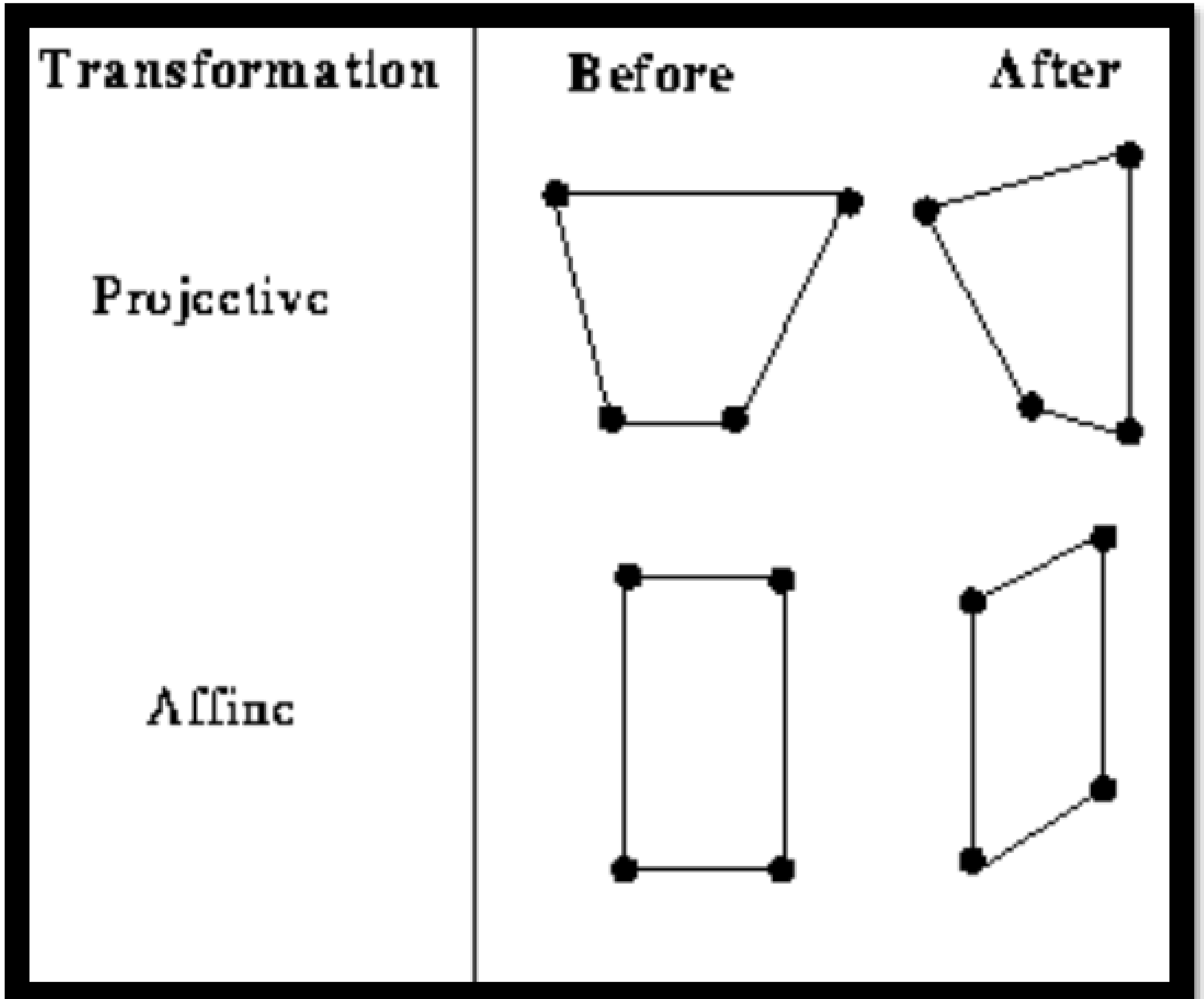
- General projective transform (or Homography)

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

- Preserves:
  - Lines
  - Also cross ratios (maybe later)



נסכם



Quiz 1

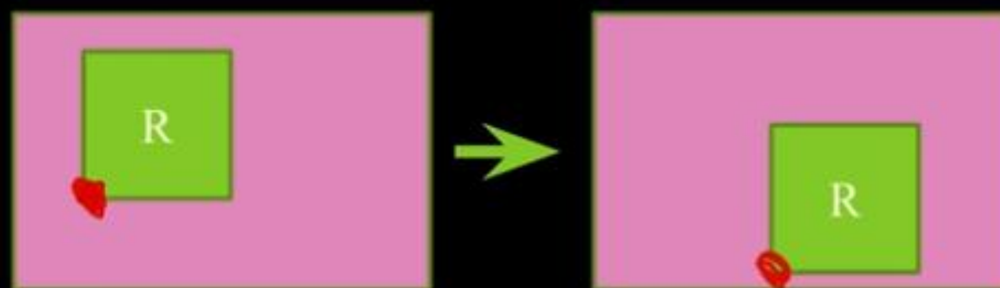
Suppose I told you the transform from image A to image B is a ***translation***. How many pairs of corresponding points would you need to know to compute the transformation?

a) 3
b) 1
c) 2
d) 4

Quiz 1 – answer

- Translation: a 1 point transformation

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



Quiz 2

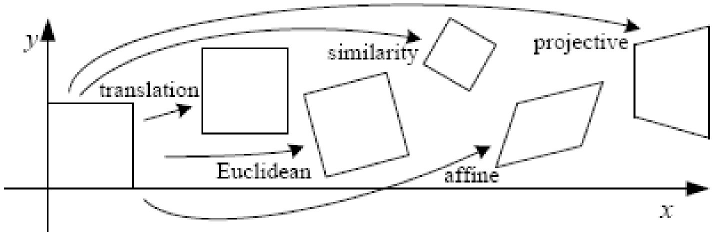
Suppose I told you the transform from image A to image B is ***affine***. How many pairs of corresponding points would you need to know to compute the transformation?

- a) 3
- b) 1
- c) 2
- d) 4

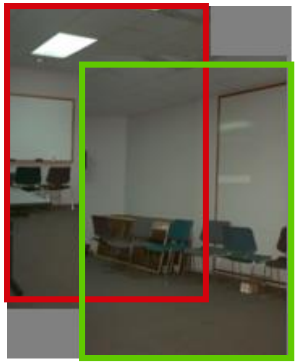
Finding the transformation

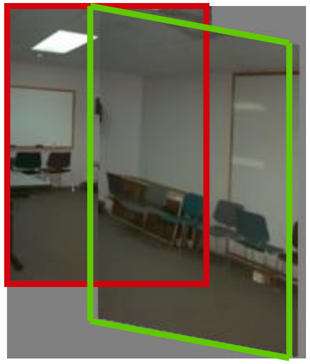
- Translation = 2 degrees of freedom
- Similarity = 4 degrees of freedom
- Affine = 6 degrees of freedom
- Homography = 8 degrees of freedom
  - How many corresponding points do we need to solve?

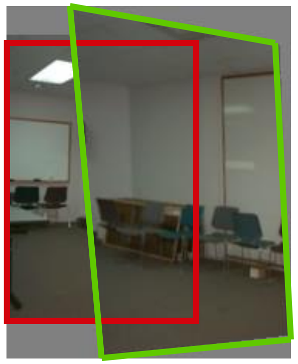
Motion models



Translation **Affine** **Perspective**



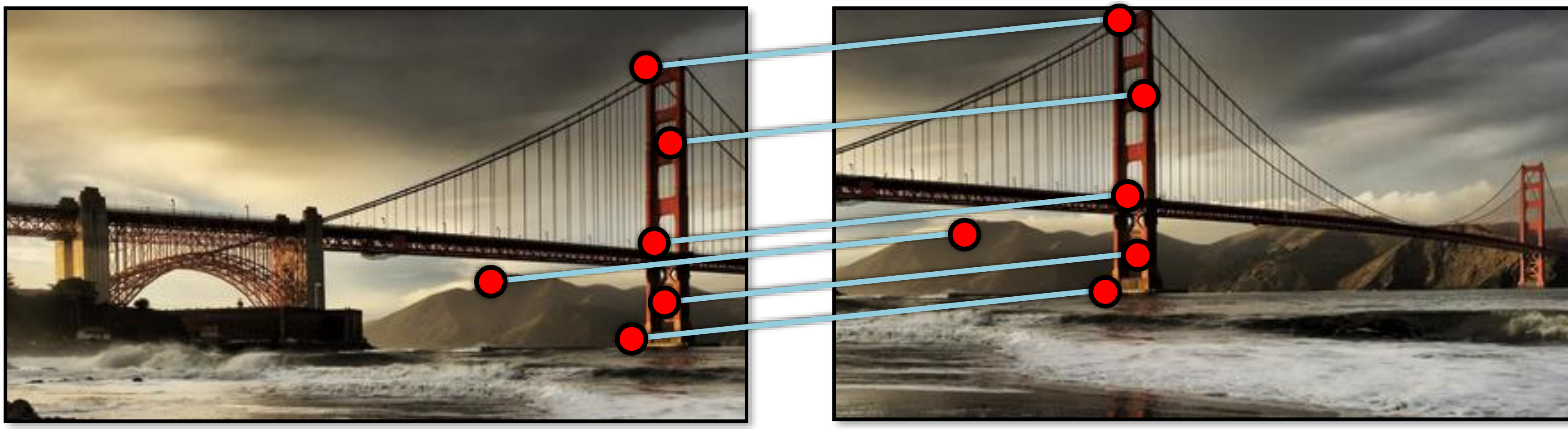


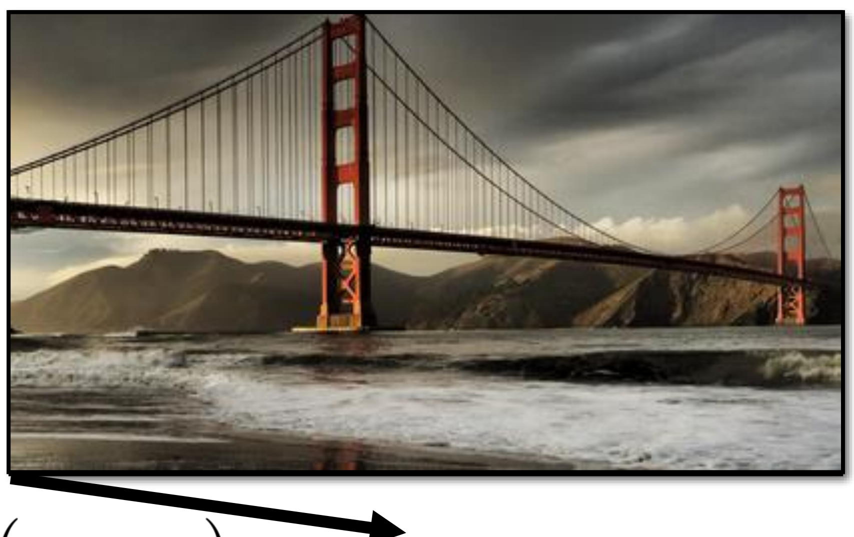


2 unknowns **6 unknowns** **8 unknowns**

איך נמצא את הטרנספורמציה ?

Simple case: translations

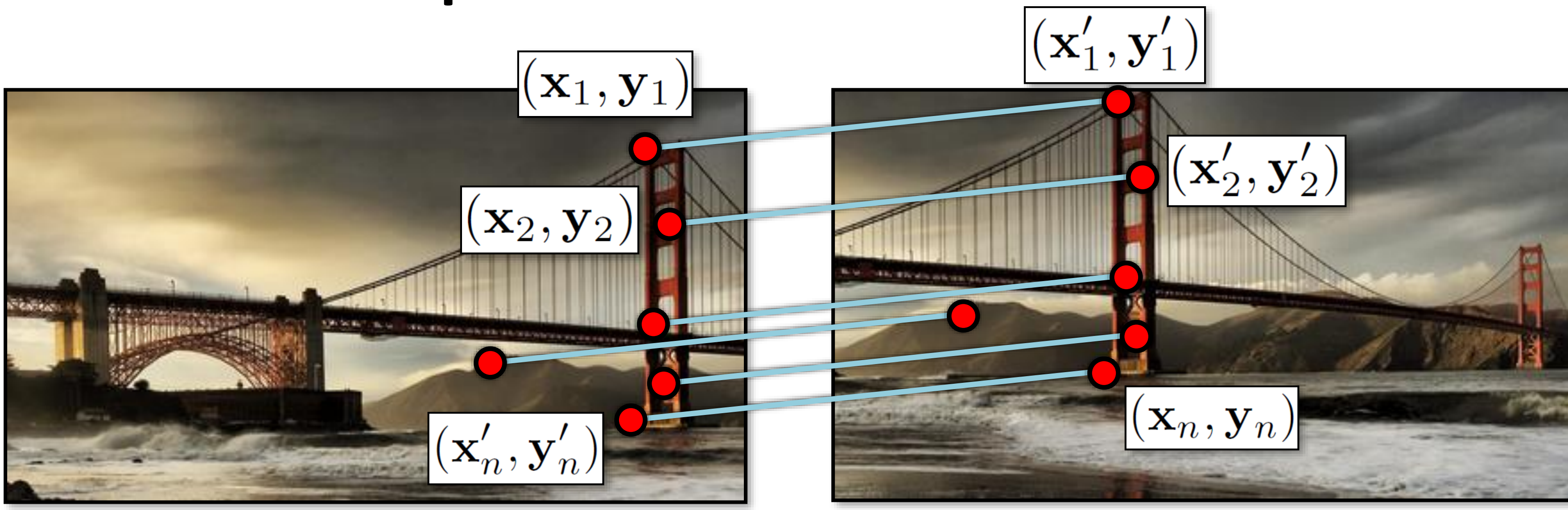




$(\mathbf{x}_t, \mathbf{y}_t)$

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

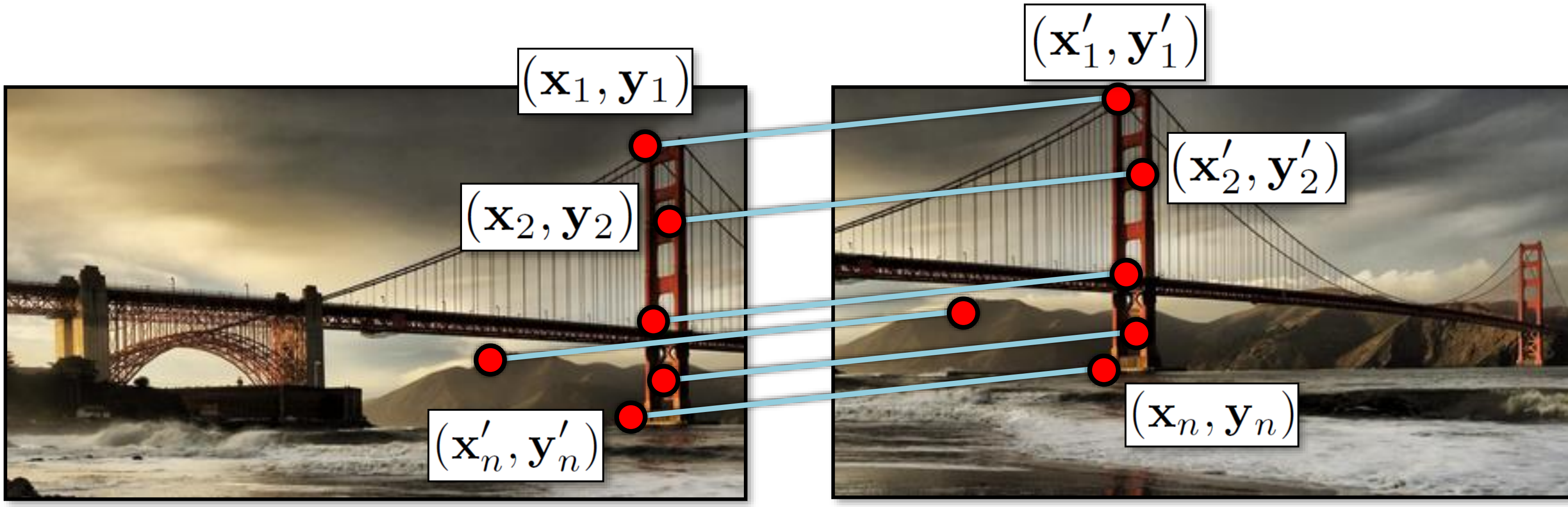
Simple case: translations



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



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

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

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

Least squares formulation

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

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

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

Solving for translations

- Using least squares

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

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

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*

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

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

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:

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

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

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

$$\begin{aligned} x'_i &= \frac{h_{00}x_i + h_{01}y_i + h_{02}}{h_{20}x_i + h_{21}y_i + h_{22}} \\ y'_i &= \frac{h_{10}x_i + h_{11}y_i + h_{12}}{h_{20}x_i + h_{21}y_i + h_{22}} \end{aligned}$$

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

Solving for homographies

$$
\begin{aligned}
x_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{00}x_i + h_{01}y_i + h_{02} \\
y_i'(h_{20}x_i + h_{21}y_i + h_{22}) &= h_{10}x_i + h_{11}y_i + h_{12}
\end{aligned}
$$

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

Direct Linear Transforms

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

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

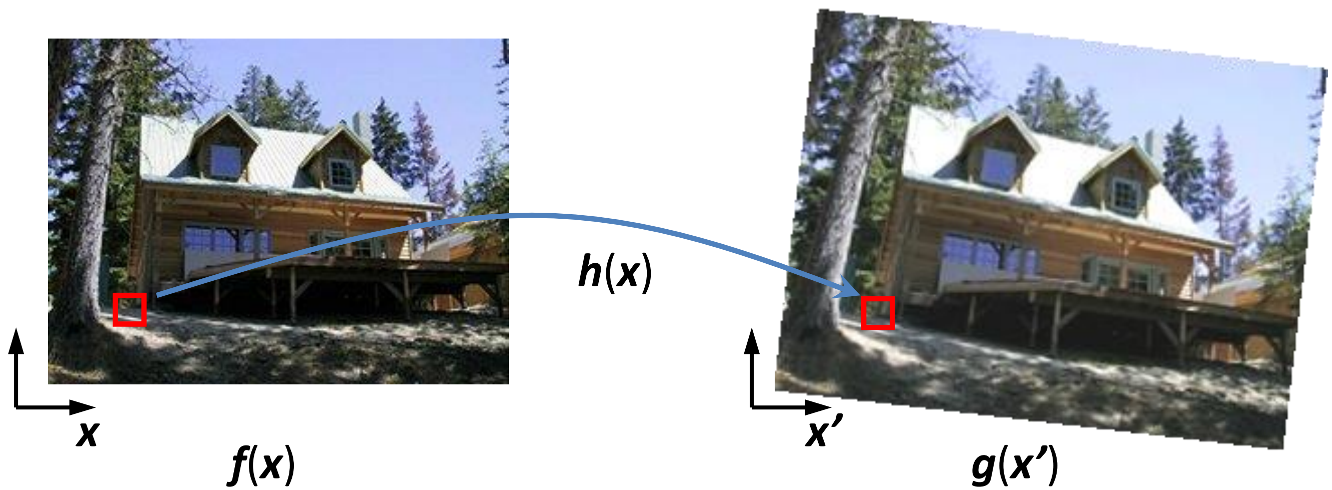
Defines a least squares problem: minimize $\|\mathbf{Ah} - \mathbf{0}\|^2$

- 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

Image Warping

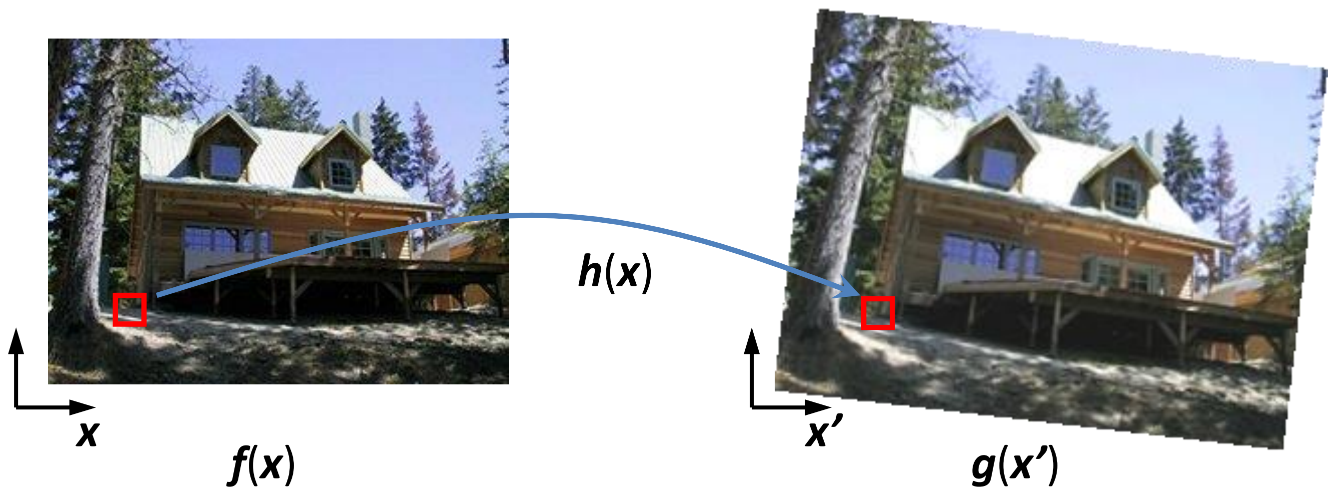
Image Warping

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



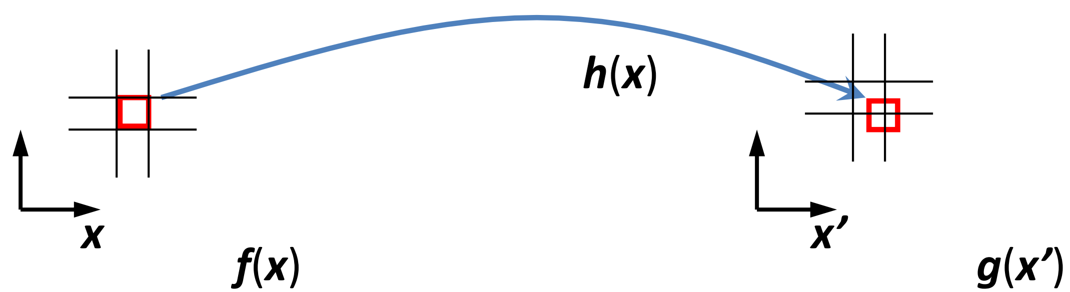
Forward Warping

- 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?



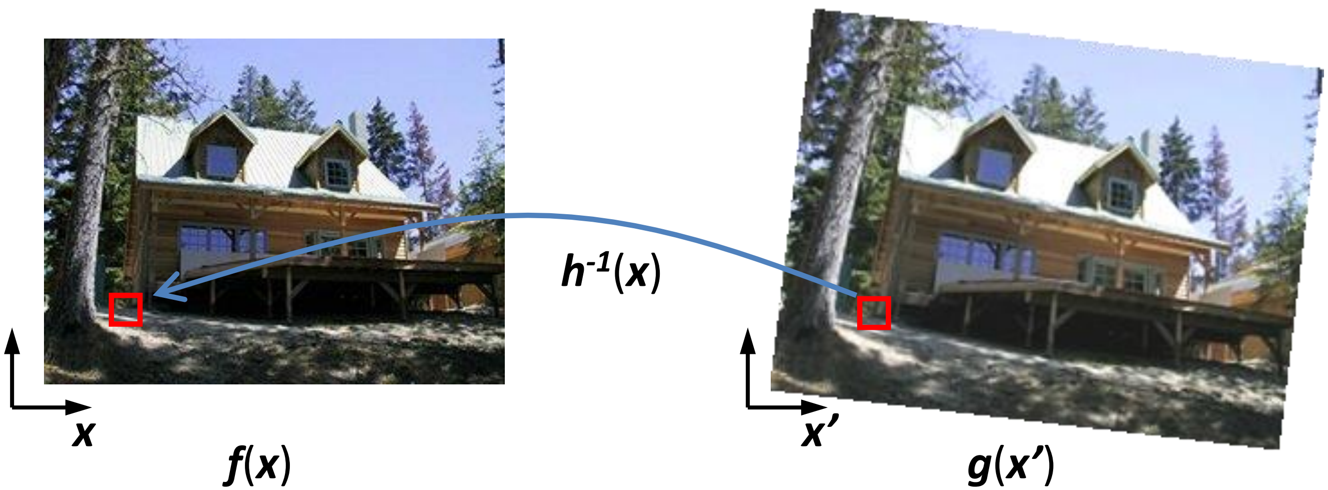
Forward Warping

- 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*)



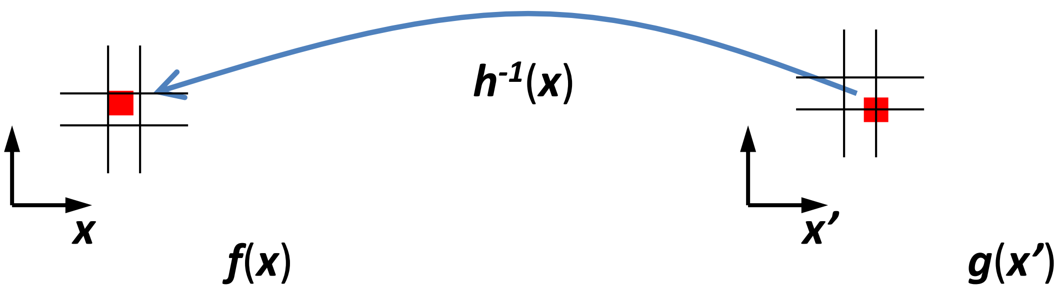
Inverse Warping

- 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'***) 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

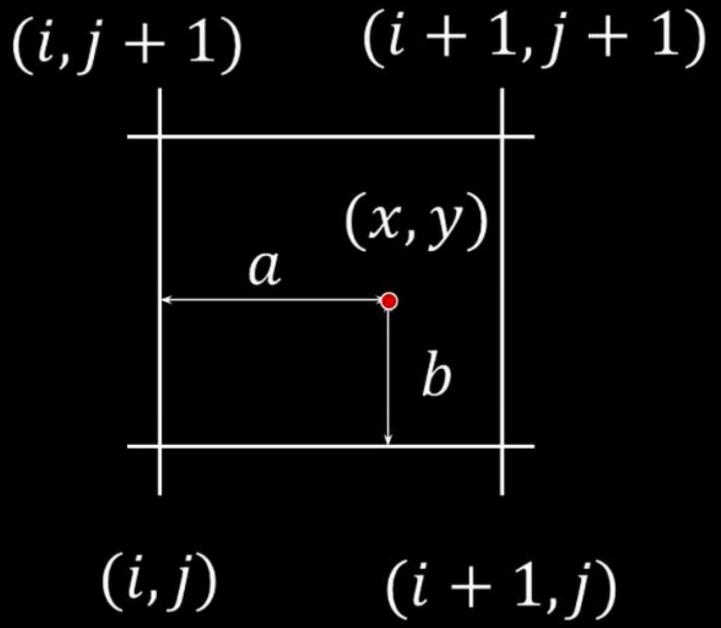


Interpolation

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

Bilinear interpolation

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

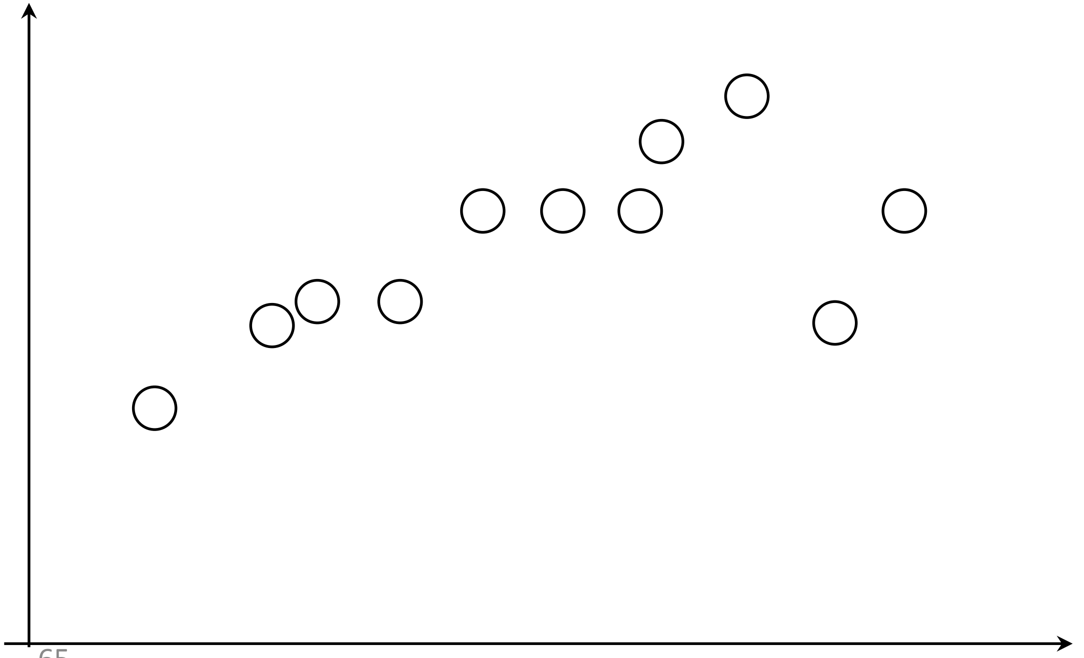


RANSAC

RANSAC for fitting a line

Simple example: fit a line

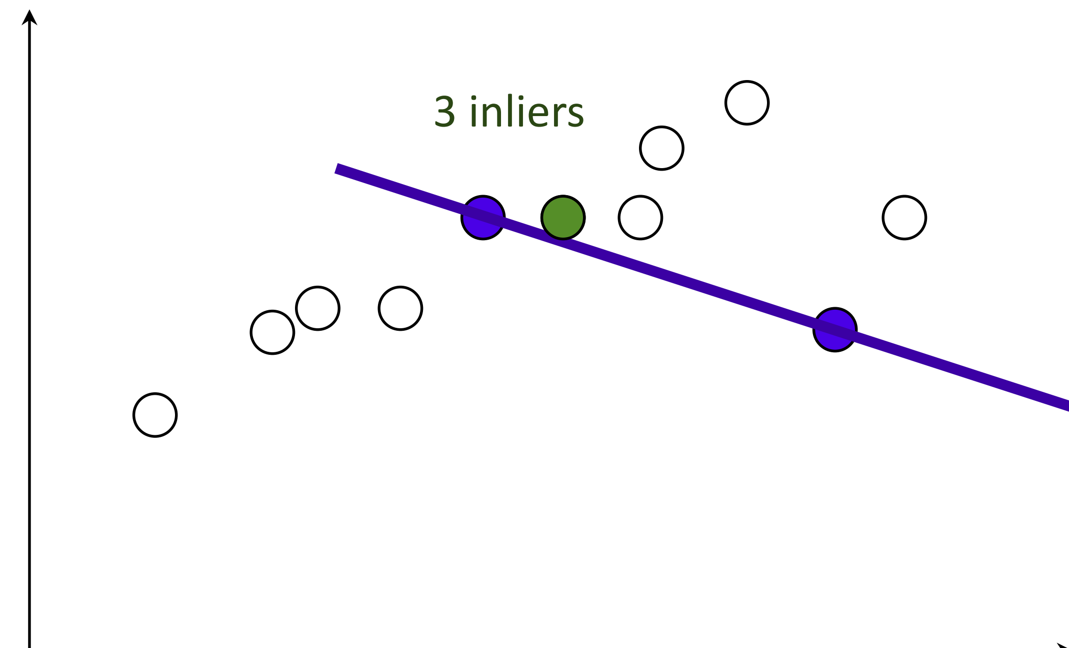
- fit y=ax+b (2 numbers a, b) to 2D pairs



65

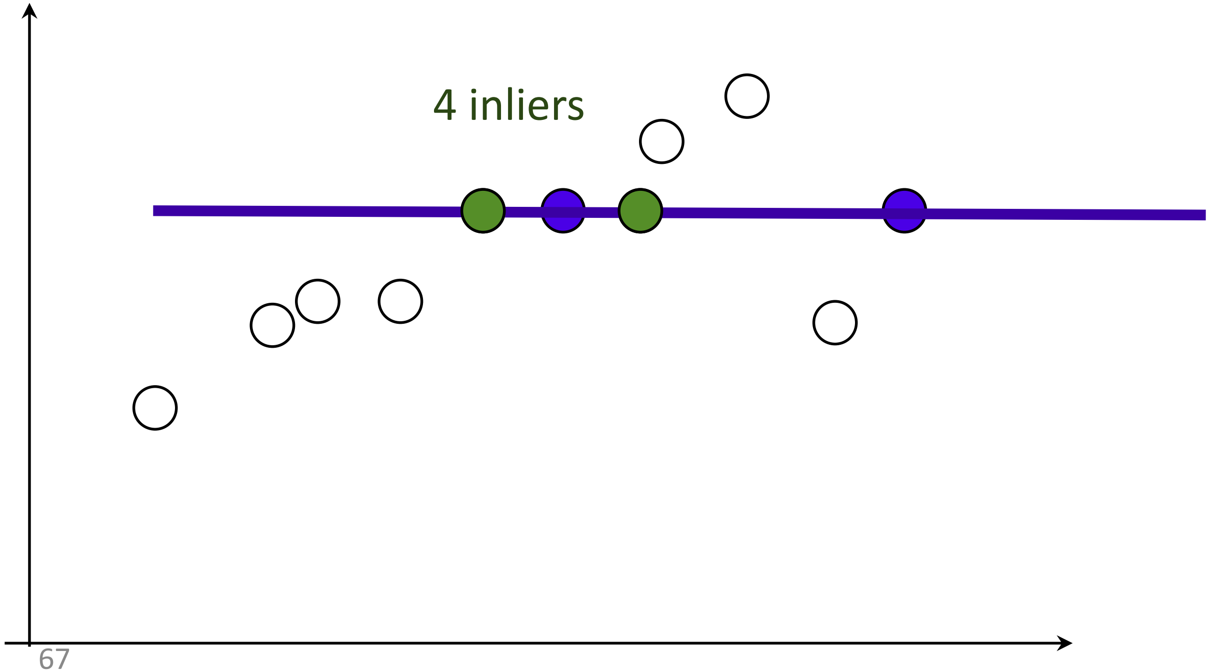
Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers



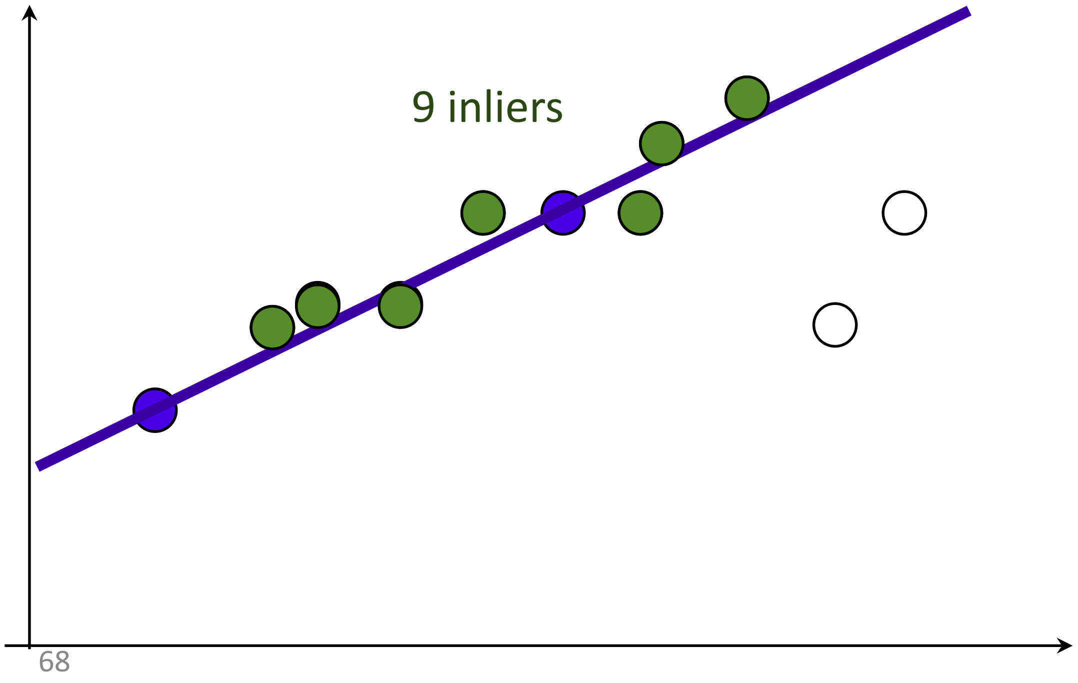
Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers



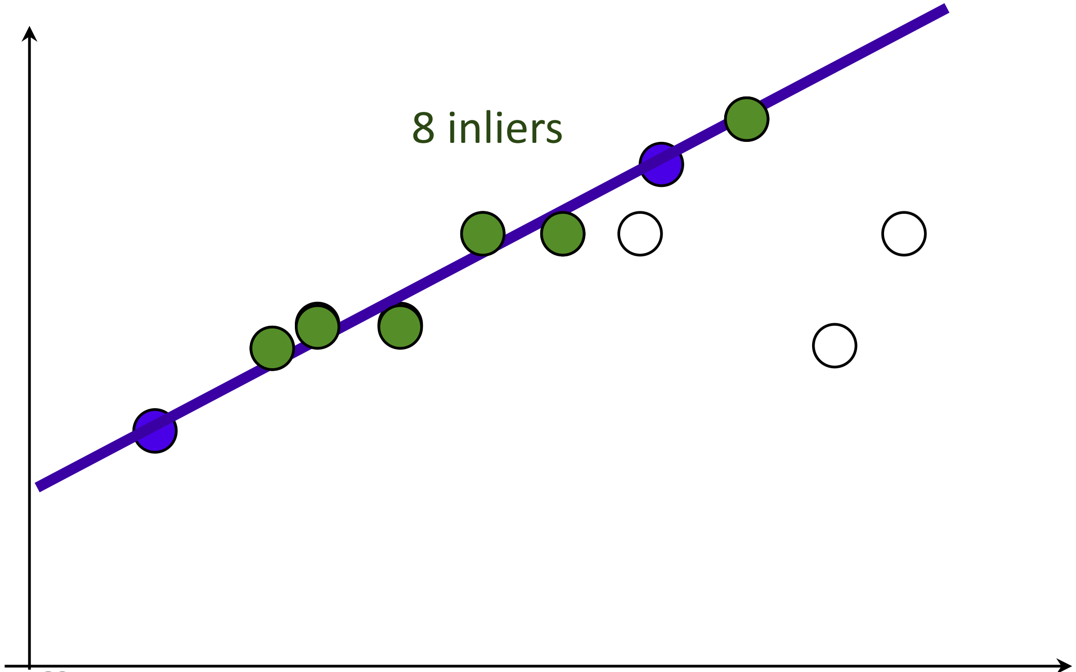
Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers



Simple example: fit a line

- Pick 2 points
- Fit line
- Count inliers



69

Simple example: fit a line

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

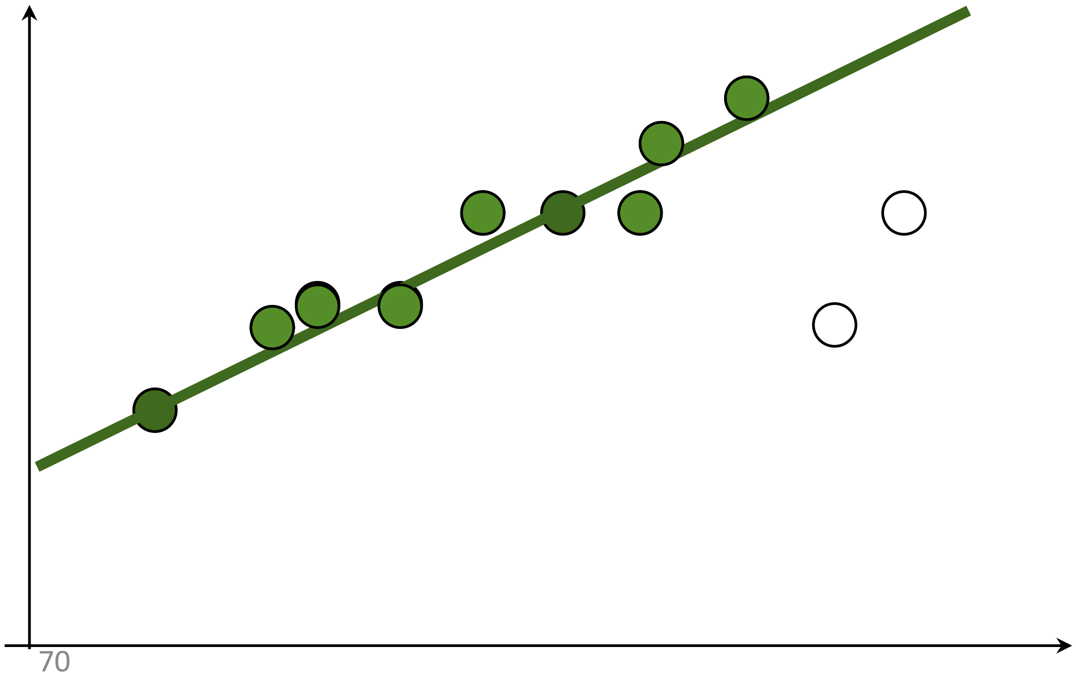


Image Stitching

all together

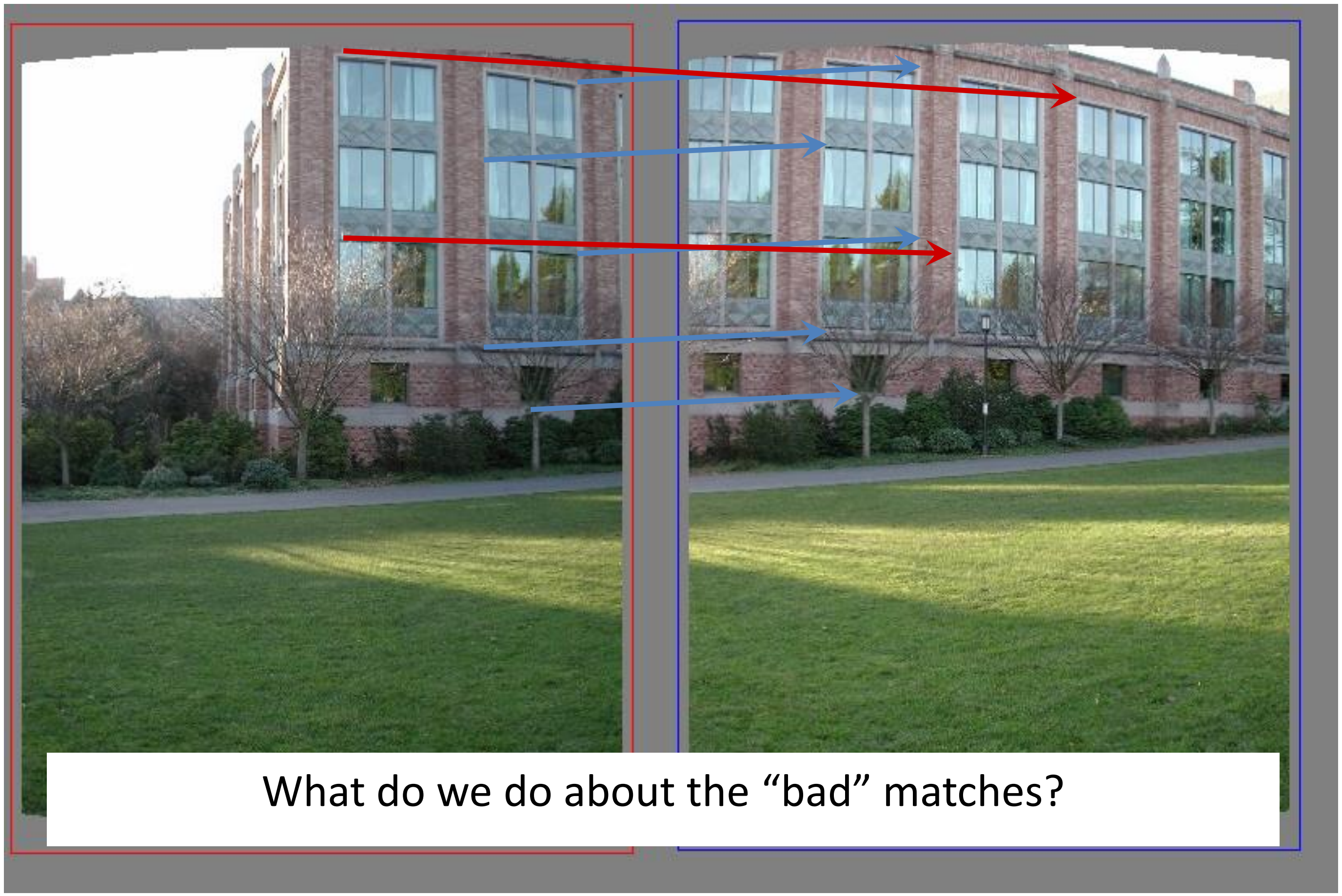
RANSAC for estimating homography

- RANSAC loop:

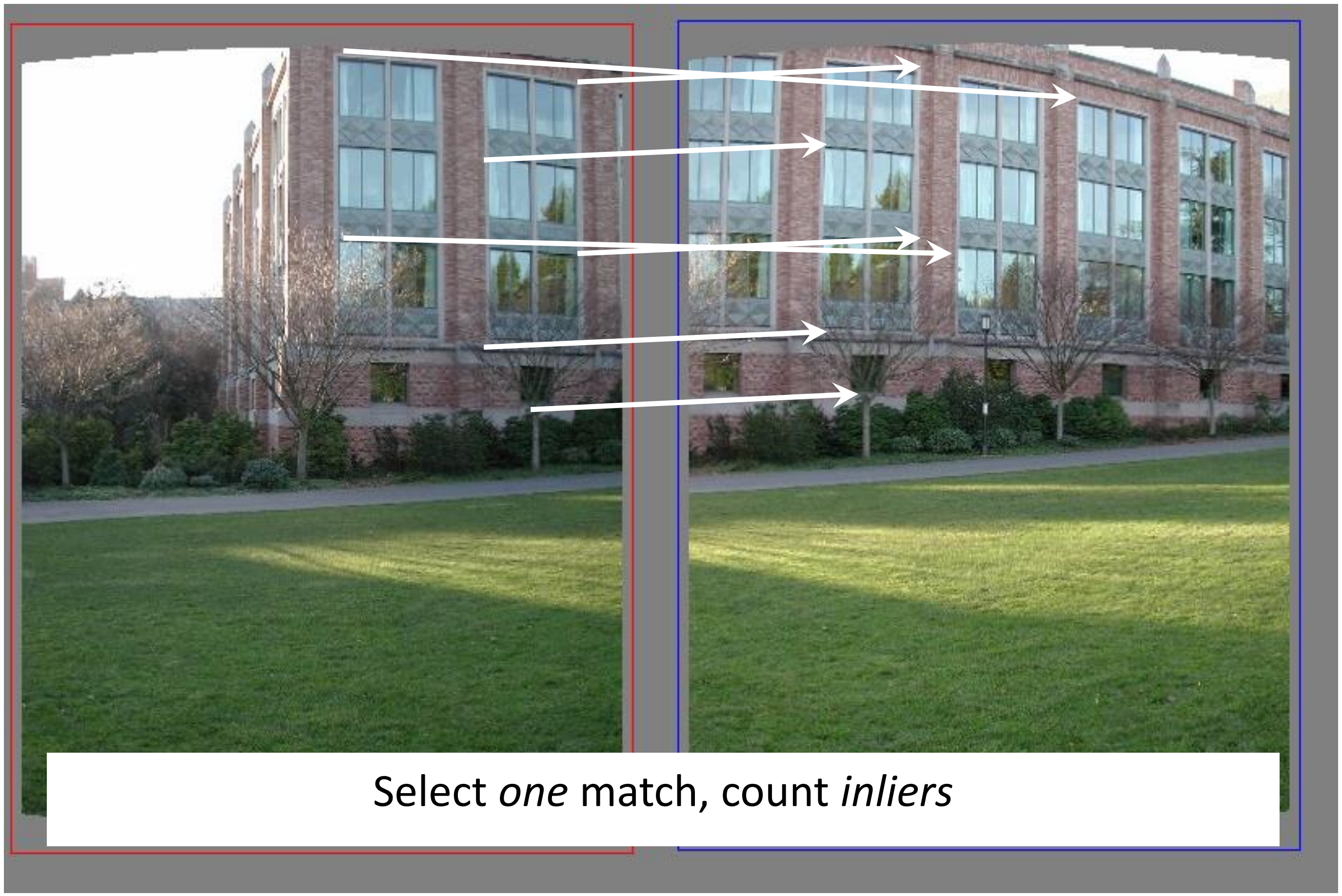
1. Select four feature pairs (at random)
2. Compute homography $\boldsymbol{H}$ (exact)
3. Compute inliers where $\|p_i', \boldsymbol{H}\, p_i\| < \varepsilon$

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

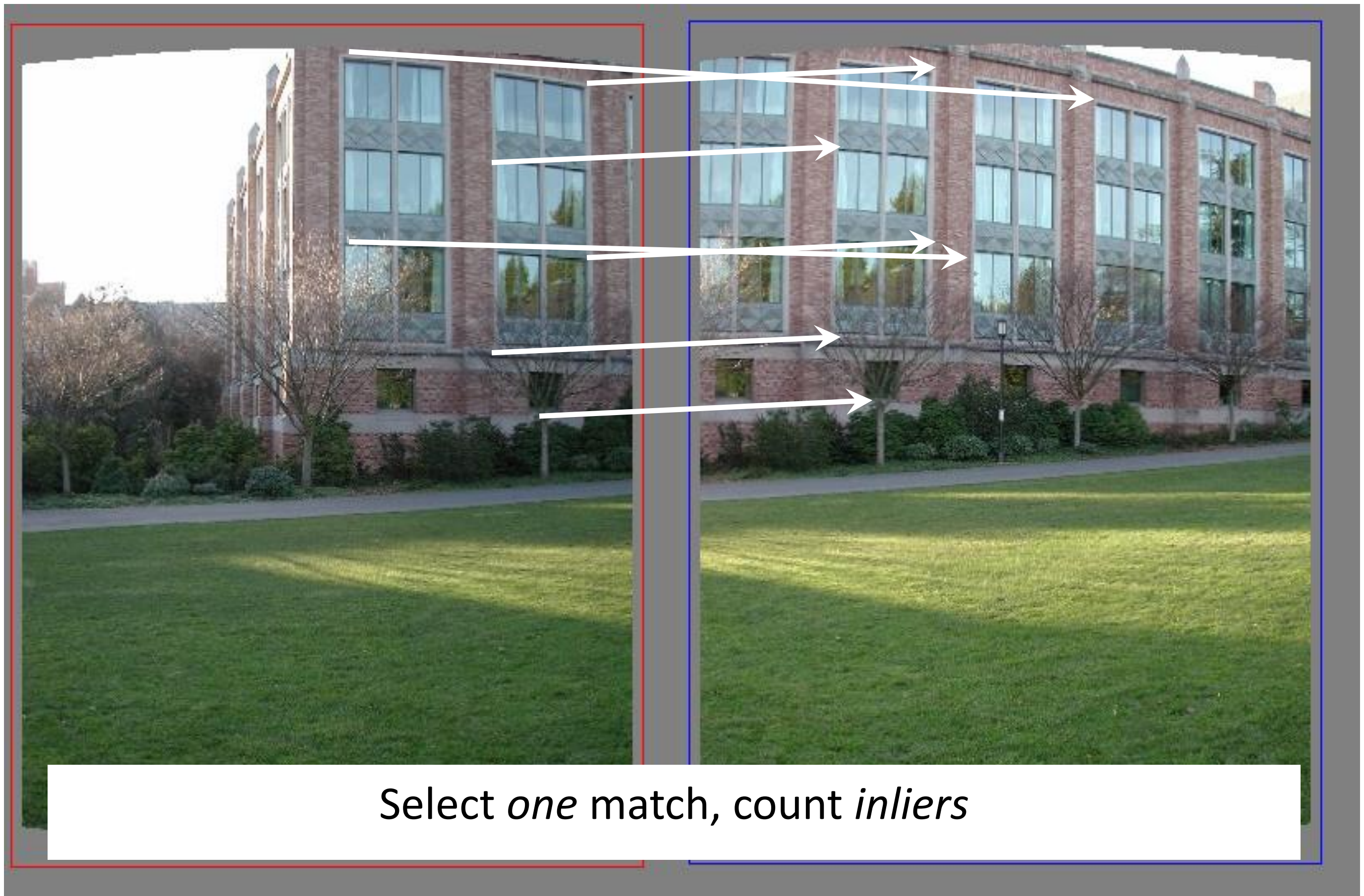
Matching features



RAndom SAmple Consensus

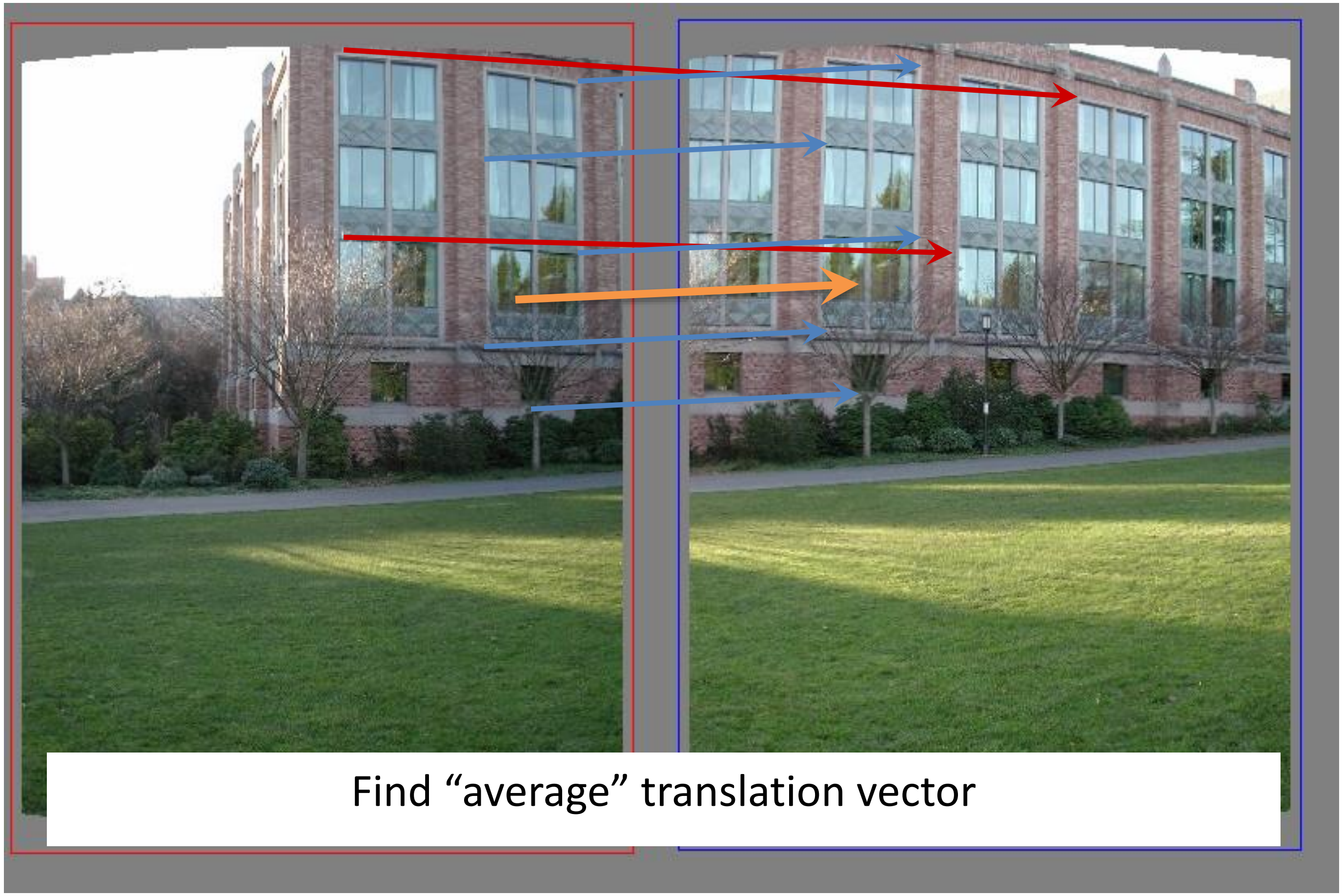


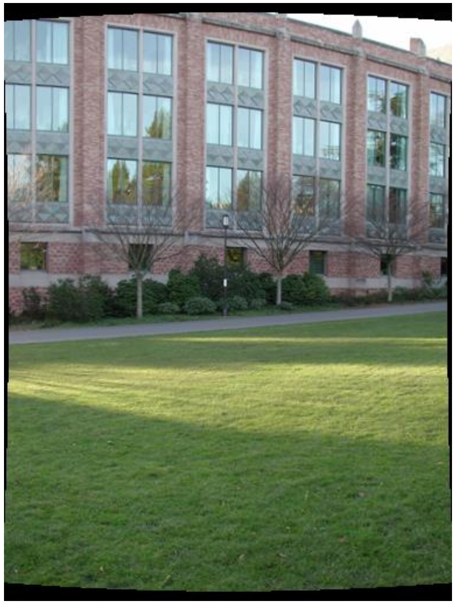
RAndom SAmple Consensus



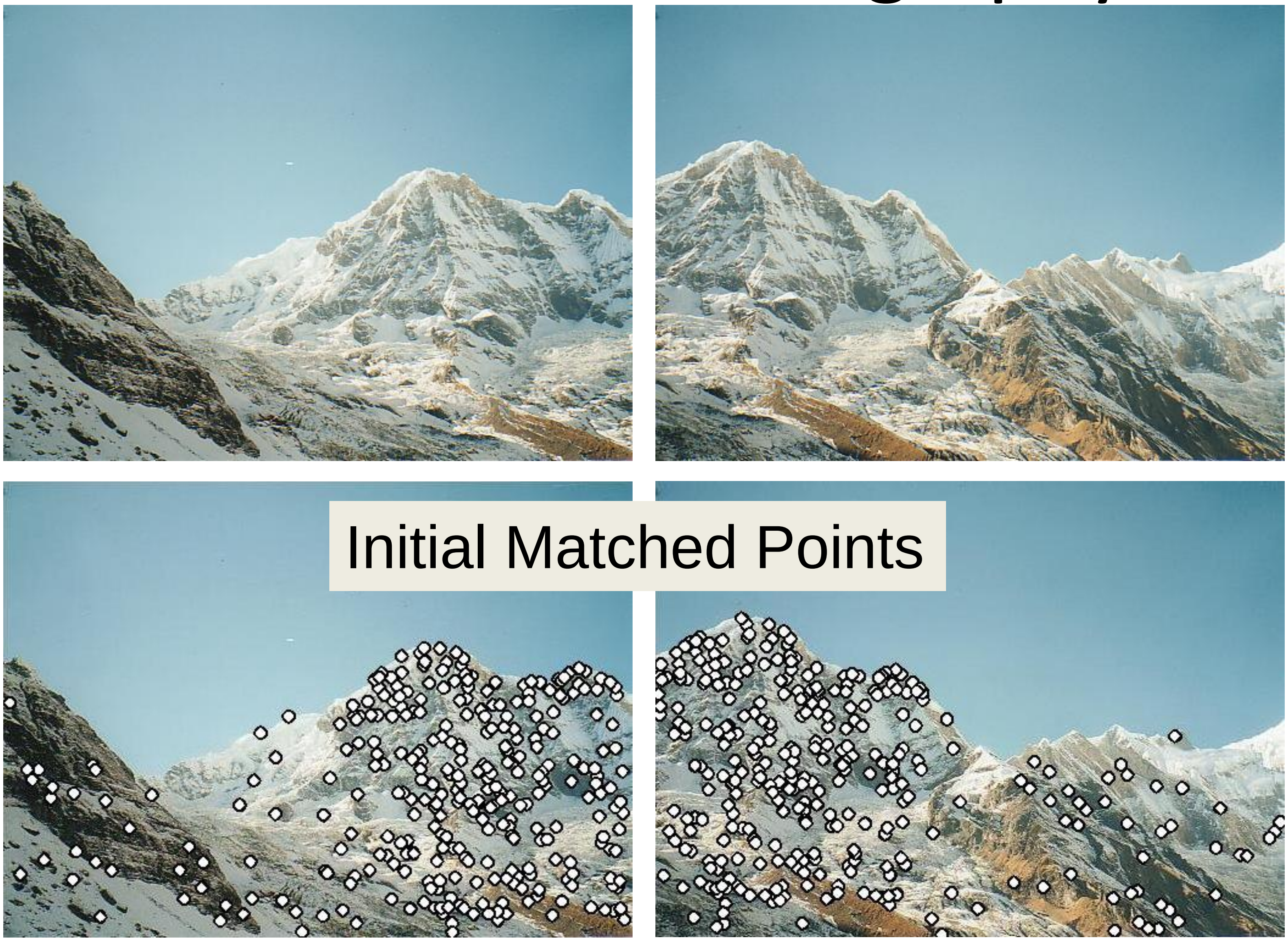
Select *one* match, count *inliers*

Least squares fit

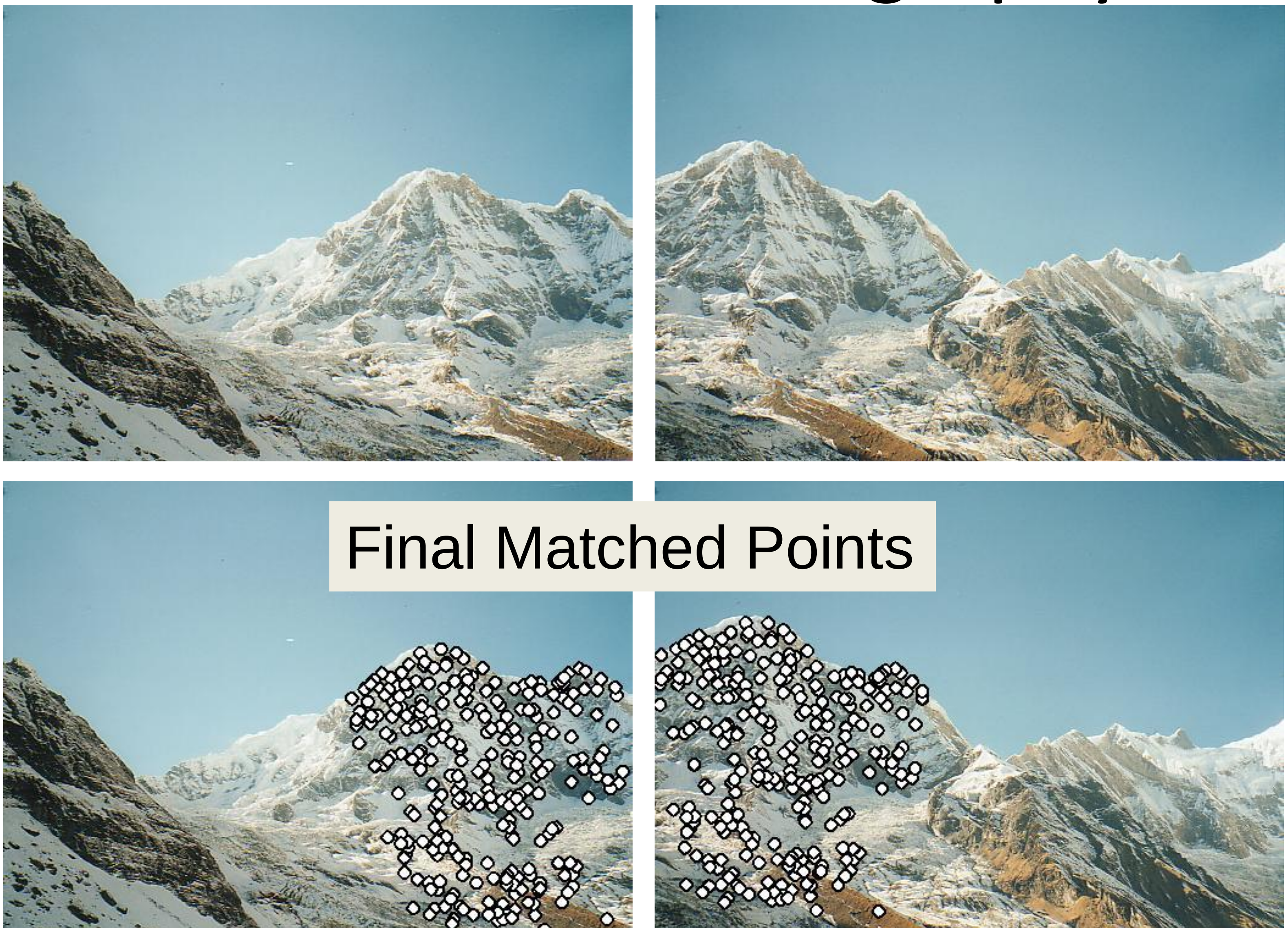


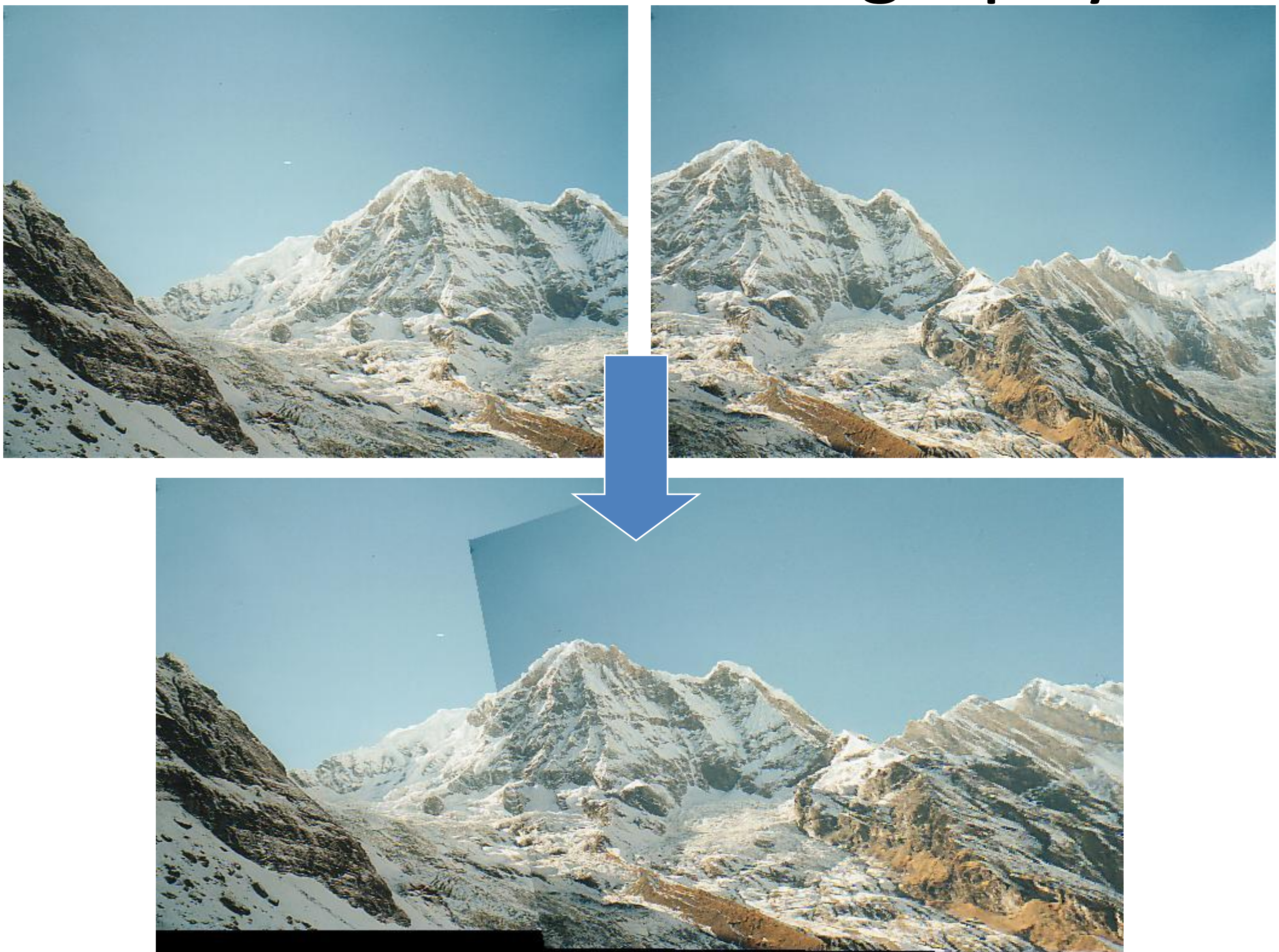
RANSAC for Homography



RANSAC for Homography



RANSAC for Homography



Quick code

https://www.pyimagesearch.com/2016/01/11/opencv-panorama-stitching/

```python
def stitch(self, images, ratio=0.75, reprojThresh=4.0,
    showMatches=False):
    # unpack the images, then detect keypoints and extract
    # local invariant descriptors from them
    (imageB, imageA) = images
    (kpsA, featuresA) = self.detectAndDescribe(imageA)
    (kpsB, featuresB) = self.detectAndDescribe(imageB)

    # match features between the two images
    M = self.matchKeypoints(kpsA, kpsB,
        featuresA, featuresB, ratio, reprojThresh)

    # if the match is None, then there aren't enough matched
    # keypoints to create a panorama
    if M is None:
        return None
    # otherwise, apply a perspective warp to stitch the images
    # together
    (matches, H, status) = M
    result = cv2.warpPerspective(imageA, H,
        (imageA.shape[1] + imageB.shape[1], imageA.shape[0]))
    result[0:imageB.shape[0], 0:imageB.shape[1]] = imageB
```

```python
    def detectAndDescribe(self, image):
        # convert the image to grayscale
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

        # check to see if we are using OpenCV 3.X
        if self.isv3:
            # detect and extract features from the image
            descriptor = cv2.xfeatures2d.SIFT_create()
            (kps, features) = descriptor.detectAndCompute(image, None)

        # otherwise, we are using OpenCV 2.4.X
        else:
            # detect keypoints in the image
            detector = cv2.FeatureDetector_create("SIFT")
            kps = detector.detect(gray)

            # extract features from the image
            extractor = cv2.DescriptorExtractor_create("SIFT")
            (kps, features) = extractor.compute(gray, kps)

        # convert the keypoints from KeyPoint objects to NumPy
        # arrays
        kps = np.float32([kp.pt for kp in kps])

        # return a tuple of keypoints and features
        return (kps, features)
```

```python
def matchKeypoints(self, kpsA, kpsB, featuresA, featuresB,
    ratio, reprojThresh):
    # compute the raw matches and initialize the list of actual
    # matches
    matcher = cv2.DescriptorMatcher_create("BruteForce")
    rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
    matches = []

    # loop over the raw matches
    for m in rawMatches:
        # ensure the distance is within a certain ratio of each
        # other (i.e. Lowe's ratio test)
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            matches.append((m[0].trainIdx, m[0].queryIdx))
    # computing a homography requires at least 4 matches
    if len(matches) > 4:
        # construct the two sets of points
        ptsA = np.float32([kpsA[i] for (_, i) in matches])
        ptsB = np.float32([kpsB[i] for (i, _) in matches])

        # compute the homography between the two sets of points
        (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC,
            reprojThresh)

        # return the matches along with the homograpy matrix
        # and status of each matched point
        return (matches, H, status)

    # otherwise, no homograpy could be computed
    return None
```