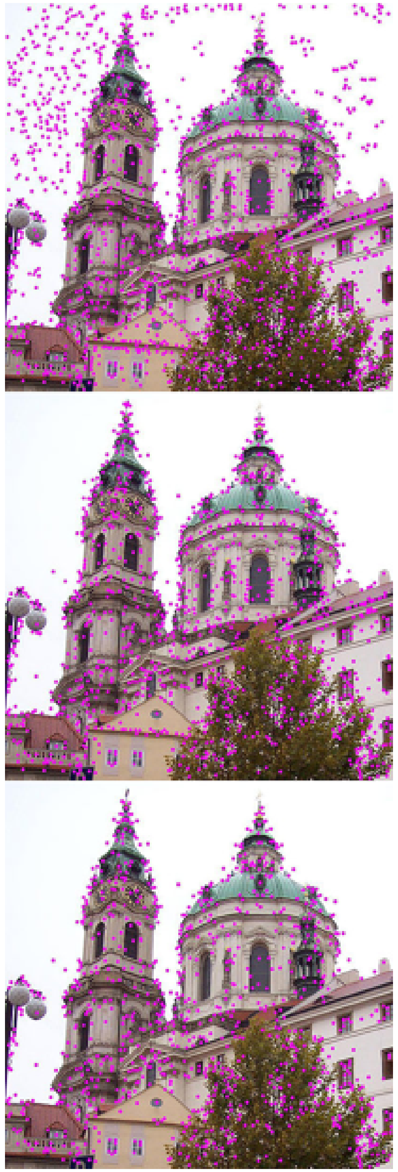
Vision and Perception

Scale Invariant Feature Transform, Image alignment and transformation

SAPIENZA
UNIVERSITÀ DI ROMA

Reading

- Szeliski: Chapter 7.1, 8.1, 8.2



SIFT

(Scale Invariant Feature Transform)

SIFT describes both a **detector** and **descriptor**

1. Multi-scale extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor

SIFT maximizes the Difference of Gaussians (DoG) in scale and in space to find same key points independently in each image.

Motivation & Improvement

Limitation of related work:

- Examine image only on a single scale
- Focus on feature detection, overlook the descriptor

SIFT

- Identify key location in scale-space
- Selected feature vectors invariant to scaling, stretching, rotation and other variation
- Improvement on feature descriptor

Well suited for:

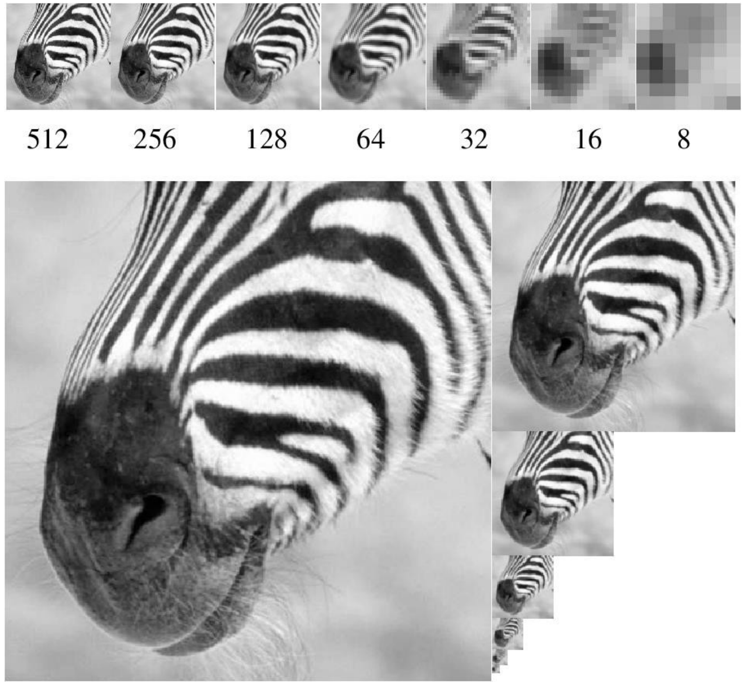
- Object recognition
- Image alignment/stiching
- Copy-move forgery detection

Stage of SIFT Object Recognition

- Feature Detection
- Local Image Description
- Matching

Scale Space

Proper scaling of objects in new image is unknown

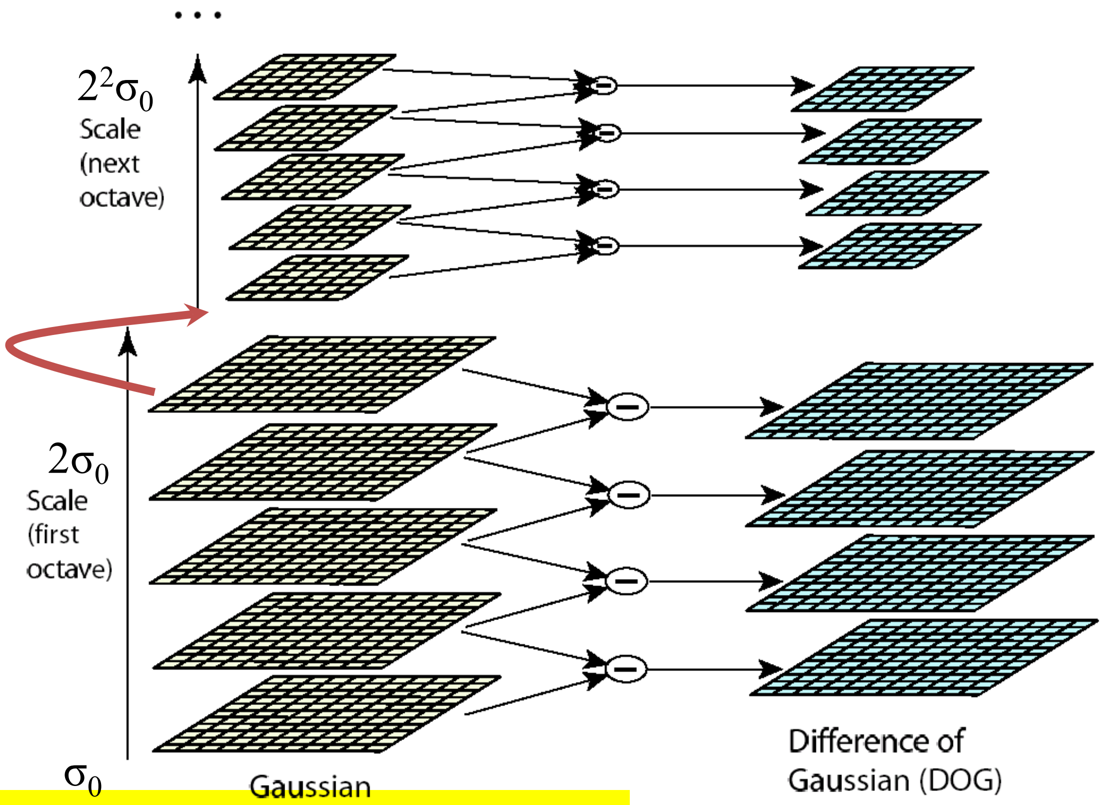


Exploring features in different scales is helpful to recognize different objects.

- Blob detector-→ LoG σ acts as a scaling parameter.
- SIFT algorithm uses Difference of Gaussians which is an approximation of LoG.

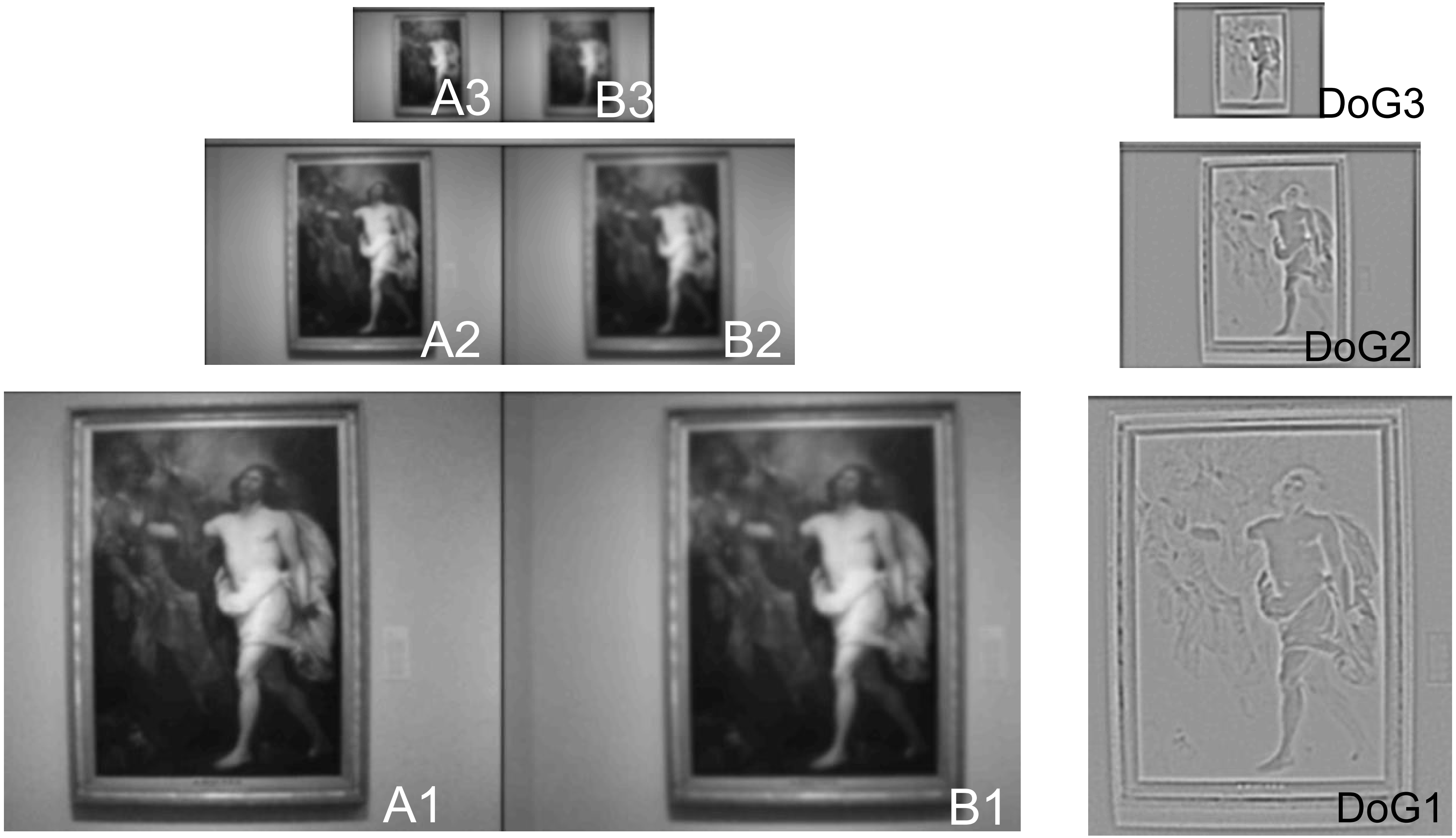
Pyramid of DoG (Octave)

- Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different σ, let it be σ and (kσ). This process is done for different octaves of the image in Gaussian Pyramid.
- DoG images are grouped by octaves
  - -An octave corresponds to doubling the value of σ
- Fixed number of scales (i.e., levels) per octave



$$D(x, y, \sigma) = [G(x, y, k\sigma) - G(x, y, \sigma)] * I(x, y), k = \sqrt{2}$$

DoG Example

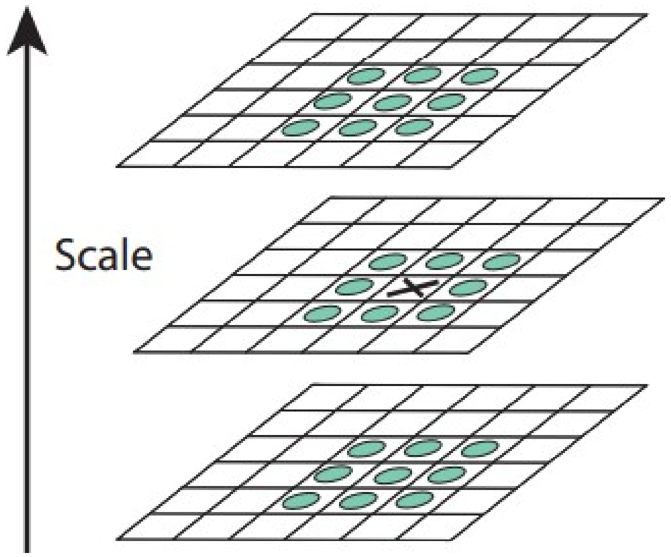


Ashley L. Kapron

Scale space extrema detection

Find maxima and minima of scale space:

- For each point on a DOG level:
  - Compare to 26 neighbors at adjacent level (within the current image, the one above and below it)
- Repeat for each DOG level
- If the point is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale
- The keypoint is represented as $\mathbf{x} = \{x, y, \sigma\}$
- We know the scale at which the keypoint was detected: scale invariance.



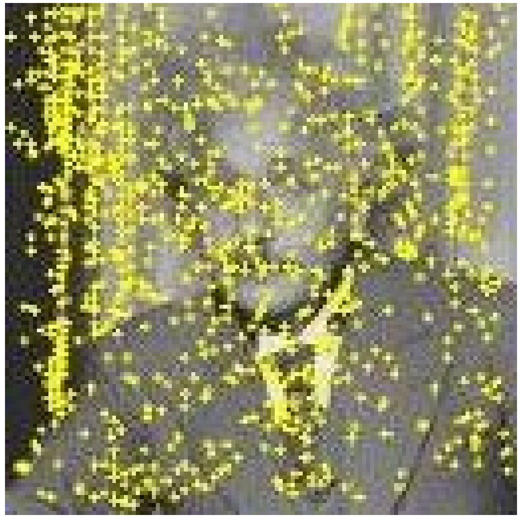
Regarding the different parameters: number of octaves = 4, number of scale levels = 5, initial $\sigma = 1.6$, $k = \sqrt{2}$ as optimal values.

David G. Lowe, IJCV 2004

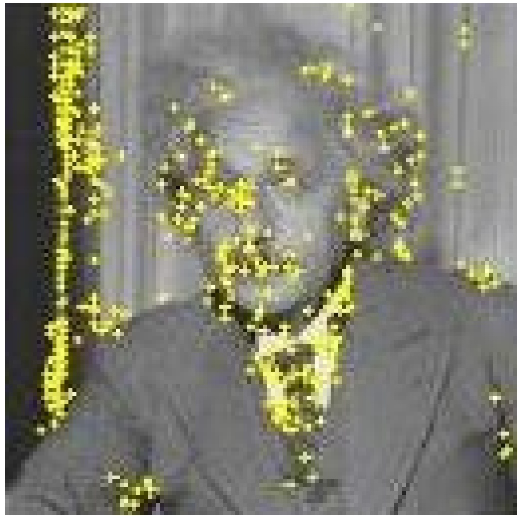
SIFT keypoint stability - Illumination

- Removing low contrast features:
  - If the magnitude of the intensity of the blurred image at the current pixel in the DoG is less than a certain value, it is rejected
    - Remove all keypoints with M_{ij} less than 0.1 times the max value
- Motivation: Low contrast is generally less reliable than high for feature points

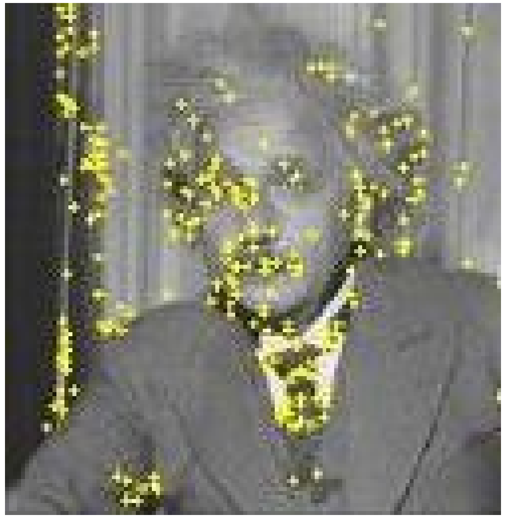
Example of keypoints



Max/mins from DOG pyramid



Filter by illumination thresholding

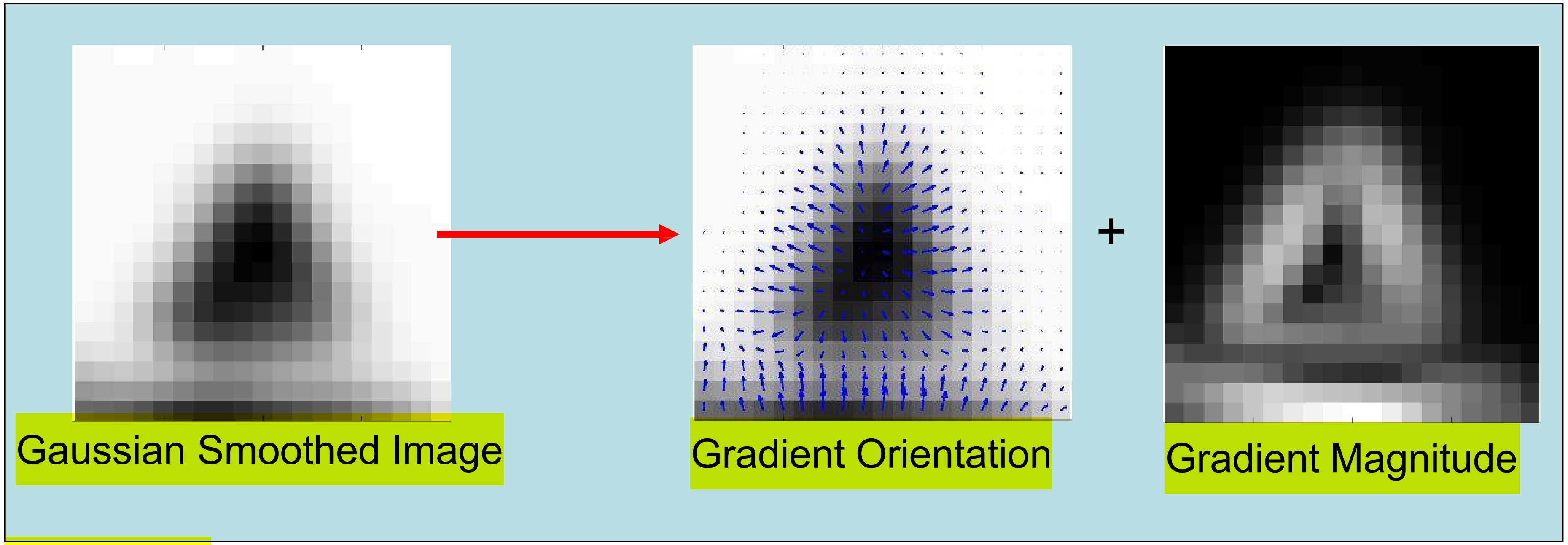


Removing edge (keep only corner)

Ashley L. Kapron

Keypoint Orientation

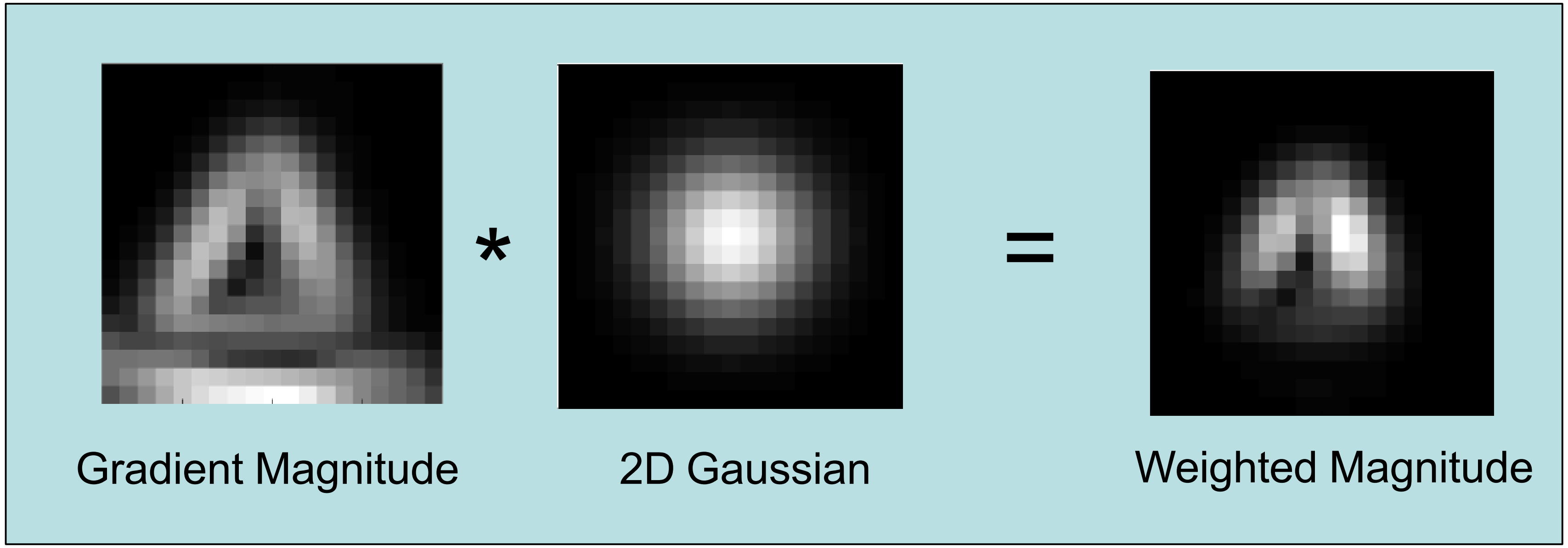
- The next thing is to assign an orientation to each keypoint.
- This orientation provides rotation invariance.
- For all levels, compute
  - Gradient magnitude and orientation (the size of the "orientation collection region" around the keypoint depends on its scale)



Ashley L. Kapron

Keypoint Orientation

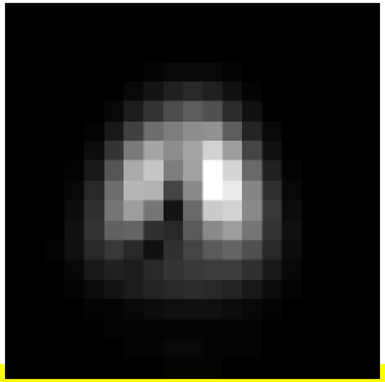
- Gradient magnitude weighted by 2D gaussian



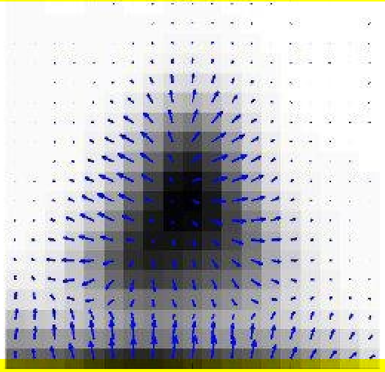
Ashley L. Kapron

Keypoint Orientation

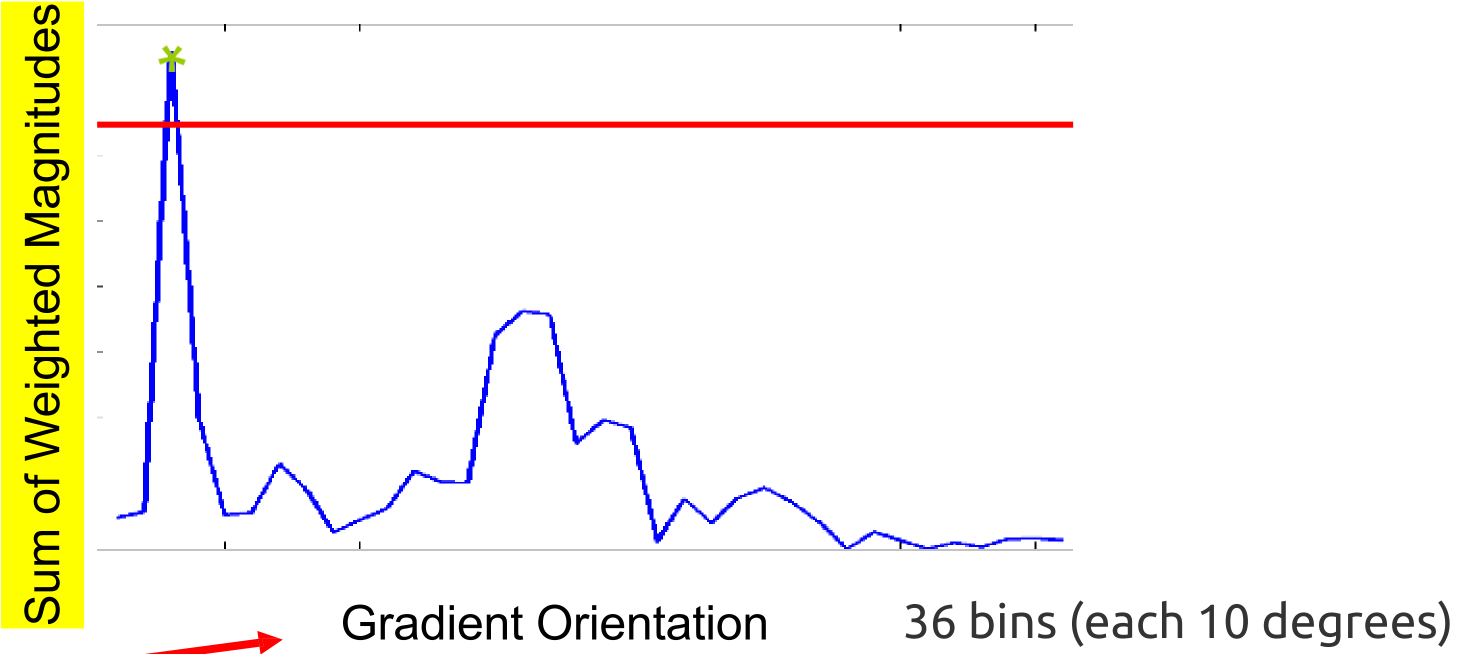
- Build a histogram of orientations (the 360 degrees of orientation are broken into 36 bins (each 10 degrees) in term of sum of weighted magnitude
- Identify peak and assign orientation and sum of magnitude to keypoint



Weighted Magnitude



Gradient Orientation



Ashley L. Kapron

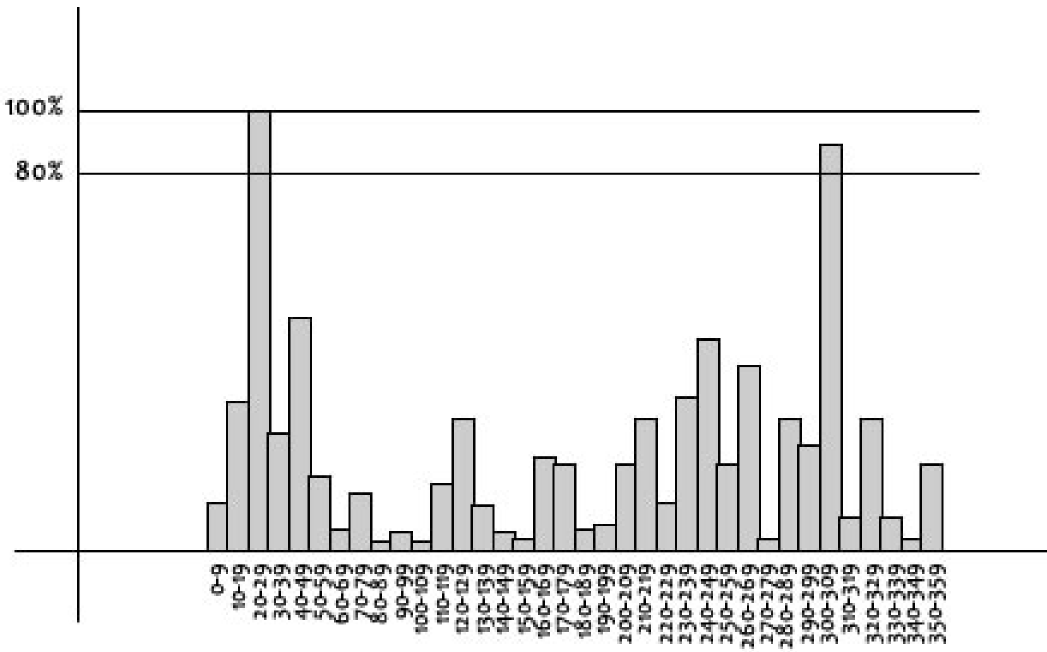
Keypoint Orientation (example)

Example: gradient direction is 18.759 degrees, then it will go into the 10-19 degree bin. And the "amount" that is added to the bin is proportional to the magnitude of gradient at that point.

- Once you've done this for all pixels around the keypoint, the histogram will have a peak at some point.
- The histogram peaks is at 20-29 degrees. So, the keypoint is assigned orientation 3 (the third bin)

- Any peaks above 80% of the highest peak are converted into a new keypoint.
This new keypoint has the same location and scale as the original. But its orientation is equal to the other peak.

So, orientation can split up one keypoint into multiple keypoints.

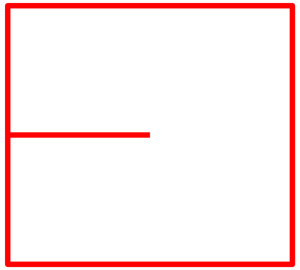


Stage of SIFT Object Recognition

- Feature Detection
- Local Image Description
- Matching

Local Image Description

- For each detected keypoint is assigned:
  - Location
  - Scale (analogous to level it was detected)
  - Orientation (assigned in previous orientation step)
- Now: Describe local image region invariant to the above transformations

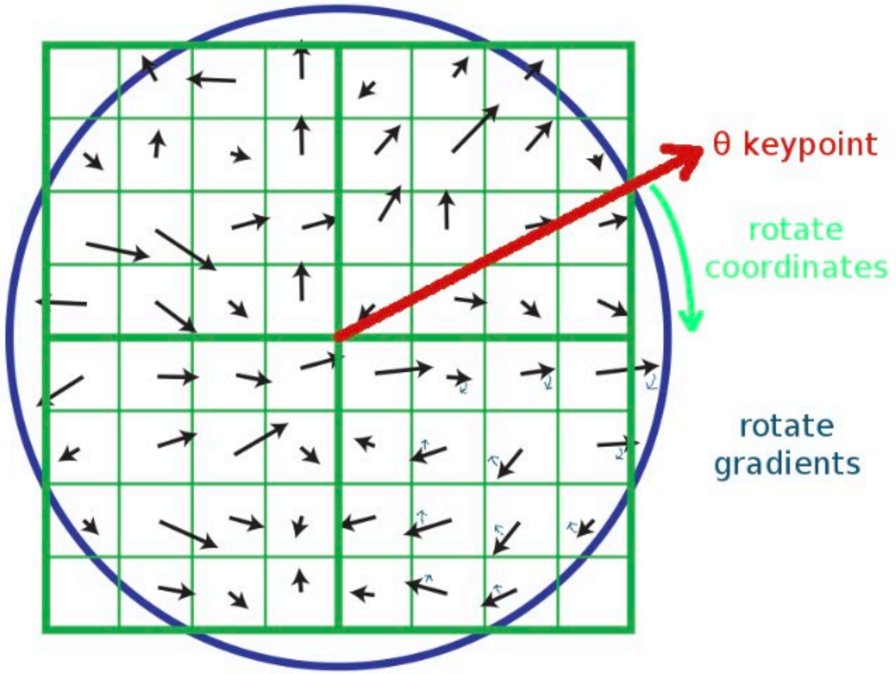


SIFT keypoint Example

SIFT descriptor

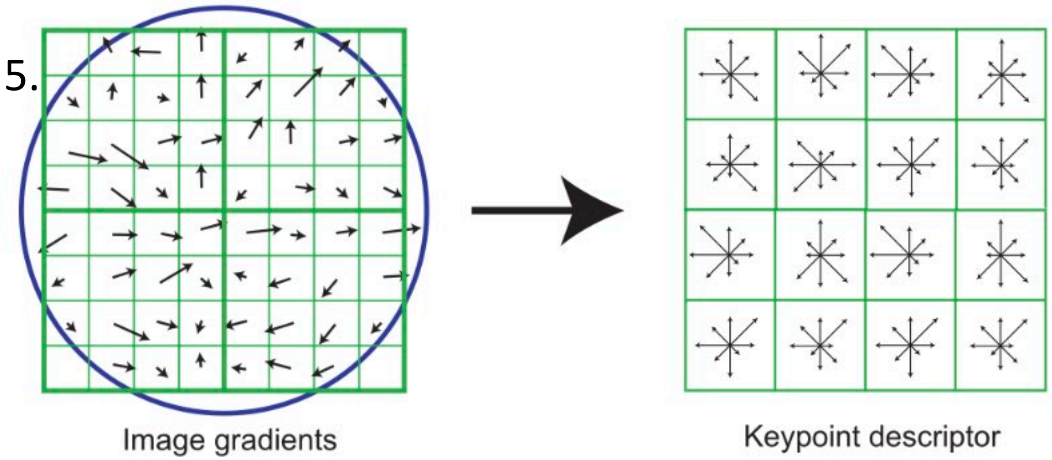
The steps of building the SIFT descriptor are as following:

1. Use the **Gaussian blurred image** associated with the keypoint's scale
2. Take **image gradients over a 16x16** square window around the detected feature
3. **Rotate the gradient** directions AND locations relative to the keypoint orientation (given by the dominant orientation)



SIFT descriptor

4. Divide the 16x16 window into a 4x4 grid of cells
5. **Compute an orientation histogram** with 8 orientations bins for each cell bins (summing the weighted gradient magnitude)



6. The resulting **SIFT descriptor is a length 128 vector** representing a 4x4 histogram array with 8 orientation bins per histogram.

SIFT descriptor properties

- invariant to rotation because we rotated the gradients: **we are assuming the rotated image will generate a key point at the same location as the original image**.

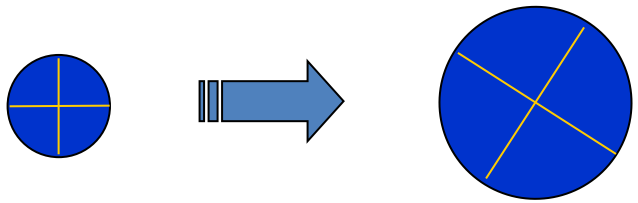
- Invariant to scale because we worked with the scaled image from DoG.

- Invariant/robustness to illumination variation since we worked with the orientation and we don't take in consideration the magnitude of the gradient.

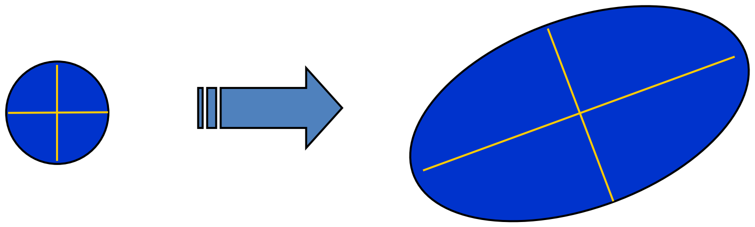
- Slightly robustness to affine transformation and to noise (empirically found).

Affine Invariant Detection

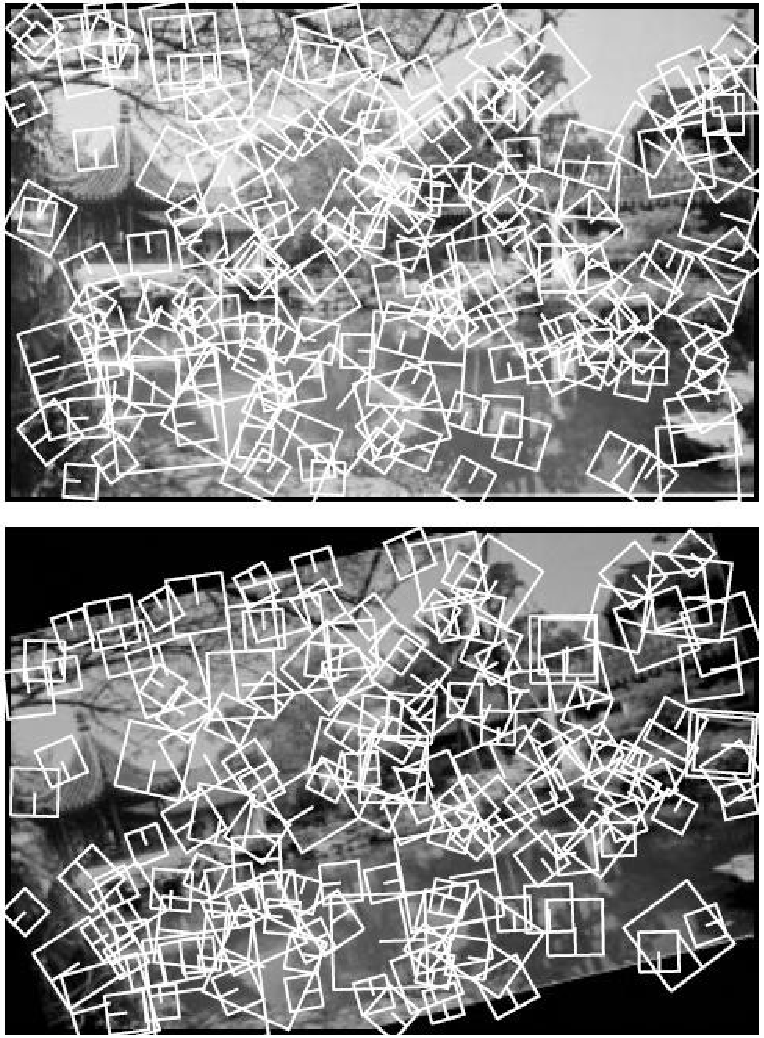
Similarity transform (rotation + uniform scale)



Affine transform (rotation + non-uniform scale)



Stability Test

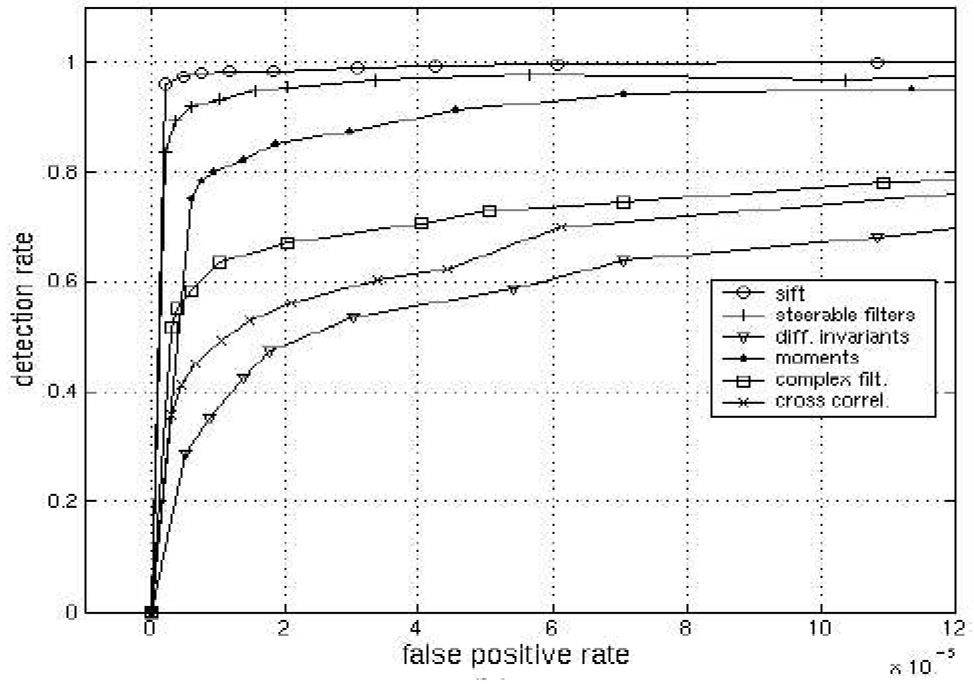


78% of the keys survive from rotation, scaling, stretching, change of brightness and contrast, and addition of pixel noise.

Stability Test

Empirically found[2] to show very good performance, invariant to *image rotation*, *scale*, *intensity change*, and to moderate *affine* transformations

Scale = 2.5
Rotation = 45^0



1 D.Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. IJCV 2004
[2] K.Mikolajczyk, C.Schmid. “A Performance Evaluation of Local Descriptors”. CVPR 2003

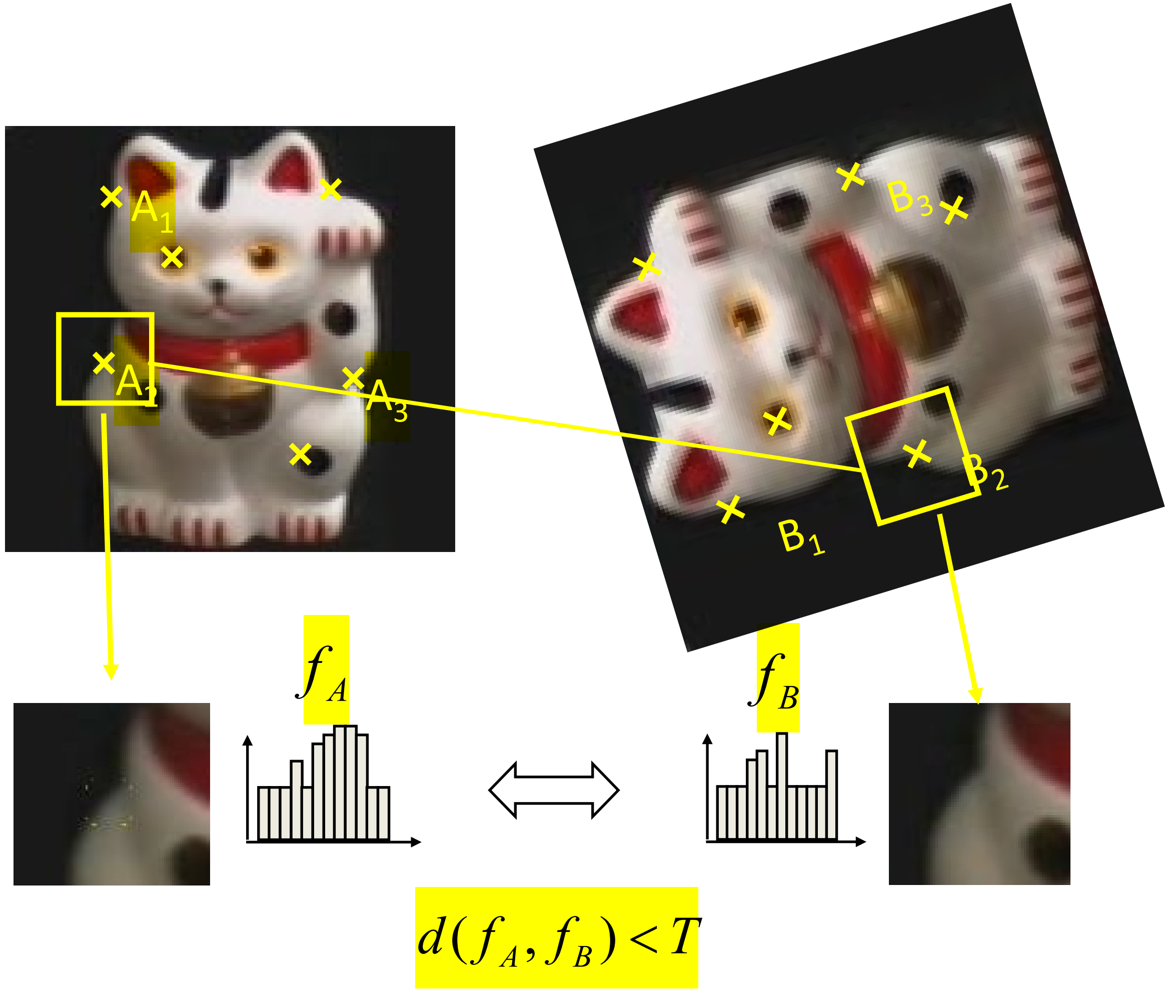
Stage of SIFT Object Recognition

- Feature Detection
- Local Image Description
- Matching

Image Matching

- Find all key points identified in a target image
  - Each keypoint will have 2D location, scale and orientation, as well as invariant descriptor vector ***(x,y,s,theta,d)***
- For each keypoint, search similar descriptor vectors in a reference image
  - Descriptor vector may match more than one reference descriptor vectors

Overview of keypoint matching



1. Find a set of distinctive keypoints
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

Feature Matching

Given a feature in I_1, how to find the best match in I_2?

1. Define a distance function that compares two descriptors
2. Test all the features in I_2, find the one with min distance

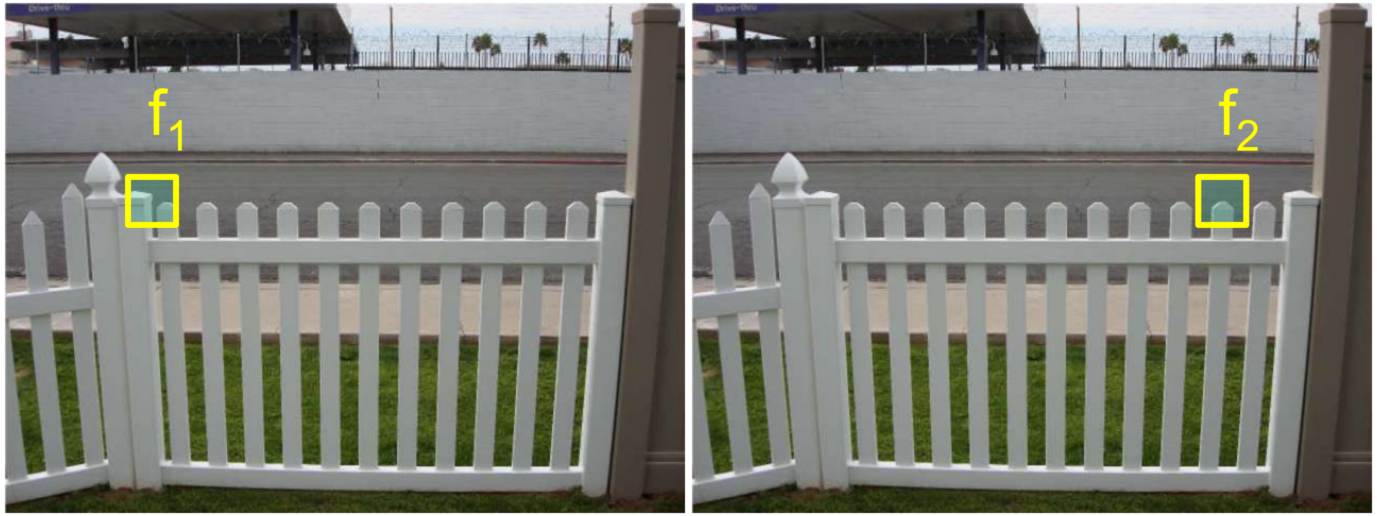
I_1

I_2

Feature distance

How to define the difference between two features f_1, f_2?

- Simple approach: L_2 distance, $||f_1 - f_2||$
    - But can give small distances for ambiguous (incorrect) matches



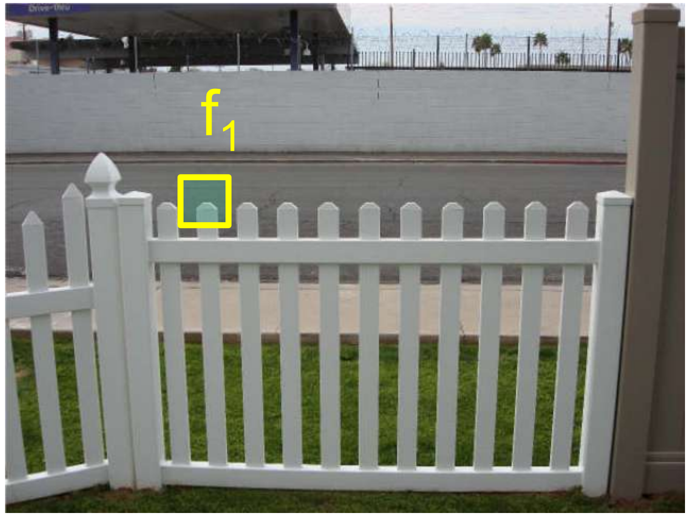
I_1 I_2

Feature distance

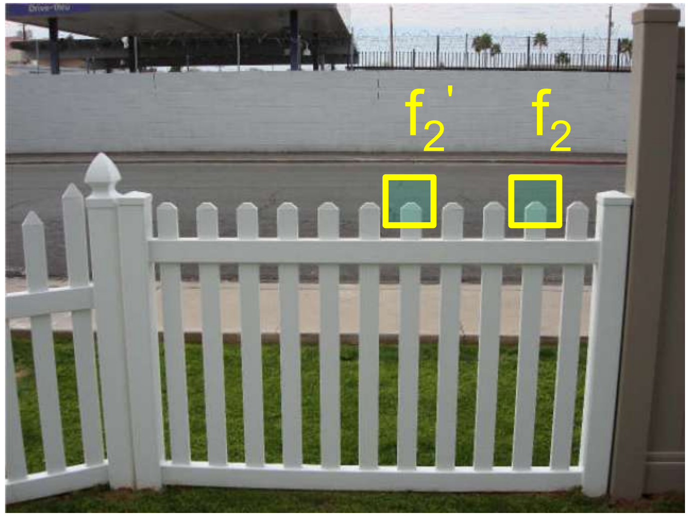
How to define the difference between two features f_1, f_2?

- Better approach: distance ratio = $||f_1 - f_2|| / || f_1 - f_2' ||$
  - f_2 is best SSD match to f_1 in I_2
  - f_2' is 2^{nd} best SSD match to f_1 in I_2

  - Sorting by this ratio puts matches in order of confidence.
  - set the **distance ratio threshold (ρ)** to around 0.5, which means that we require our best match to be at least twice as close as our second best match to our initial features descriptor. Thus discarding our ambiguous matches and retaining the good ones



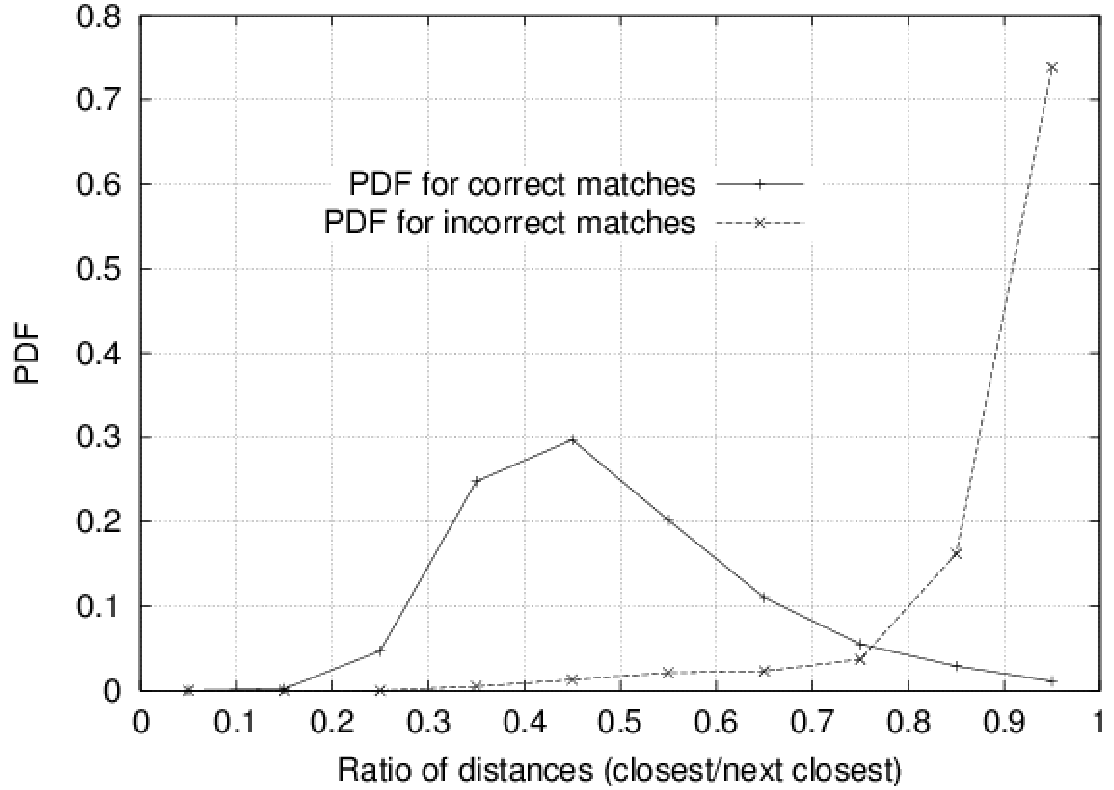
I_1



I_2

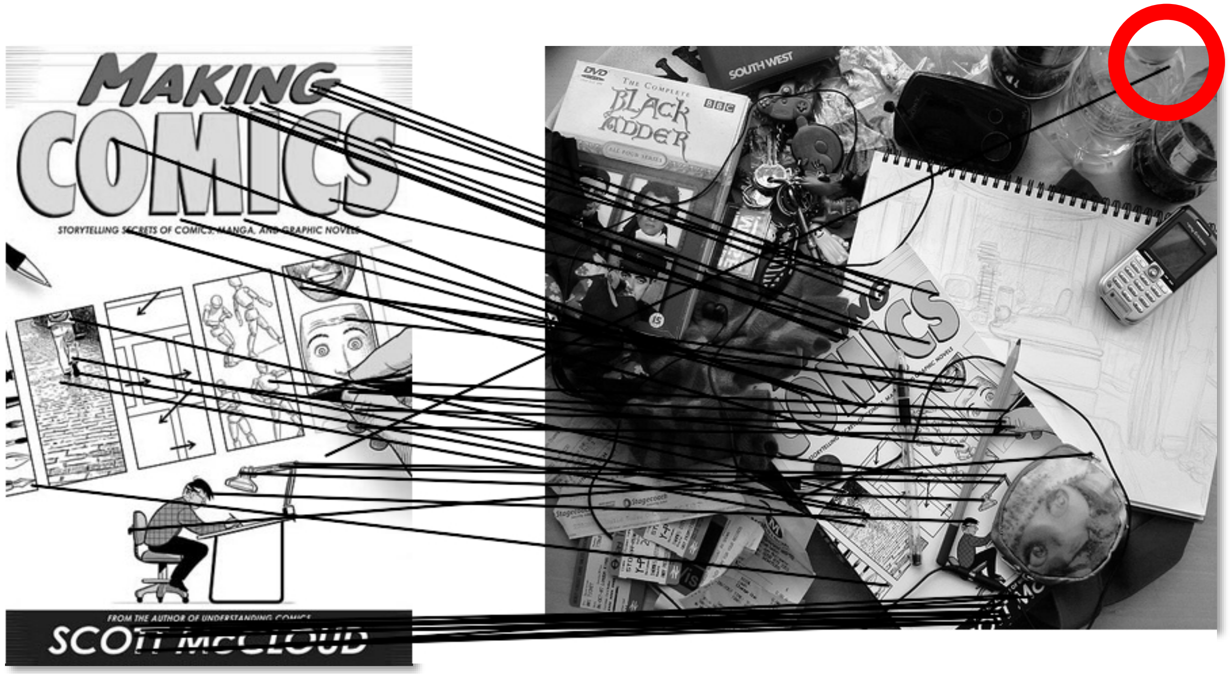
Matching SIFT features

- Accept a match if $SSD(f_1, f_2) / SSD(f_1, f_2') < t$
- t=0.8 has given good results in object recognition.
- Eliminated 90% of false matches.
- Discarded less than 5% of correct matches



Features matching example

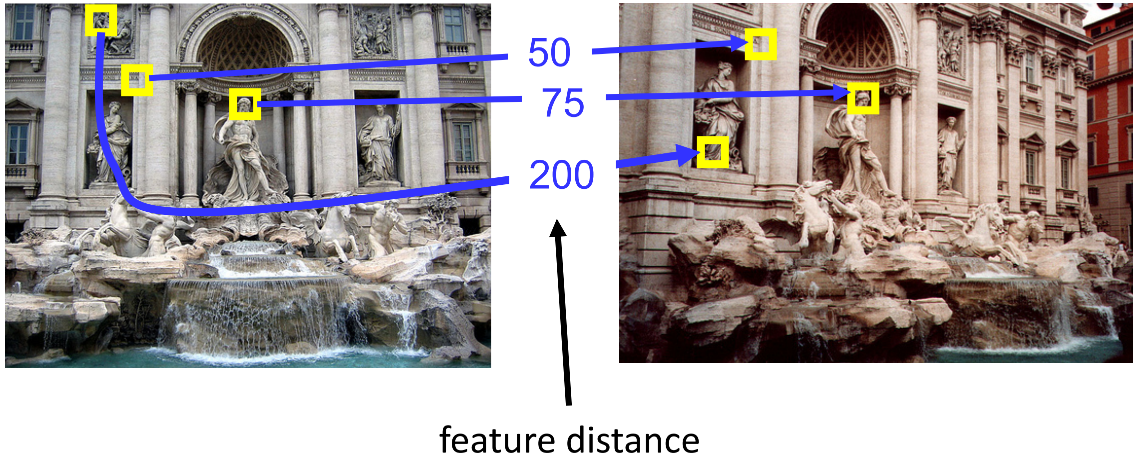
We need to deal with **outliers**



51 matches (thresholded by ratio score)

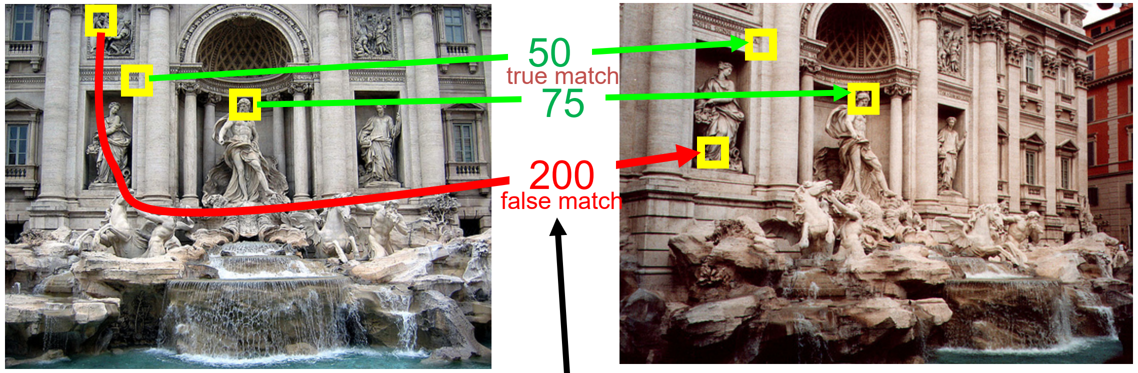
Evaluating the results

How can we measure the performance of a feature matcher?



True/false positives

How can we measure the performance of a feature matcher?



feature distance

The distance threshold affects performance

True positives = # of detected matches that are correct

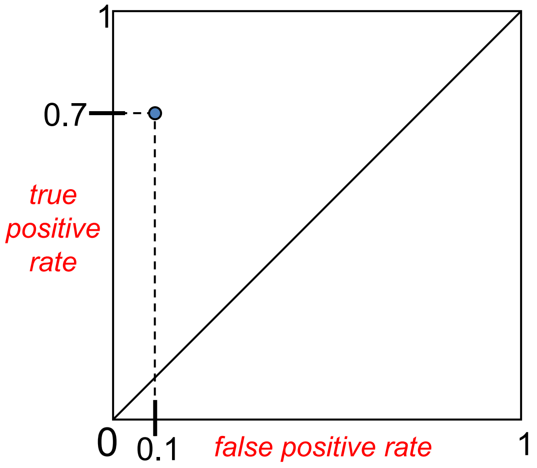
False positives = # of detected matches that are incorrect

Evaluating the results

How can we measure the performance of a feature matcher?

$$\frac{\text{\# true positives}}{\text{\# matching features (positives)}}$$

recall



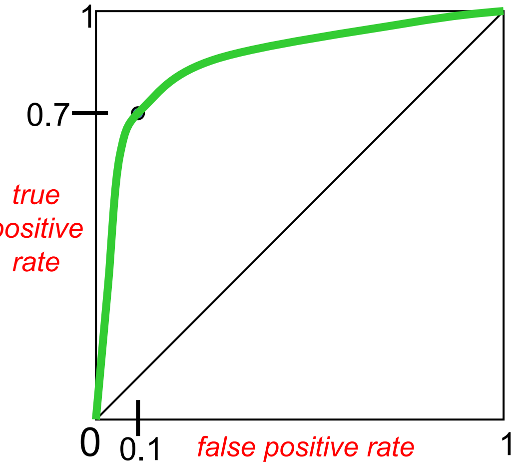
$$\frac{\text{\# false positives}}{\text{\# unmatched features (negatives)}}$$

1 - *specificity*

Evaluating the results

How can we measure the performance of a feature matcher?

ROC curve ("Receiver Operator Characteristic")



$\frac{\text{\# true positives}}{\text{\# matching features (positives)}}$

recall

$\frac{\text{\# false positives}}{\text{\# unmatched features (negatives)}}$

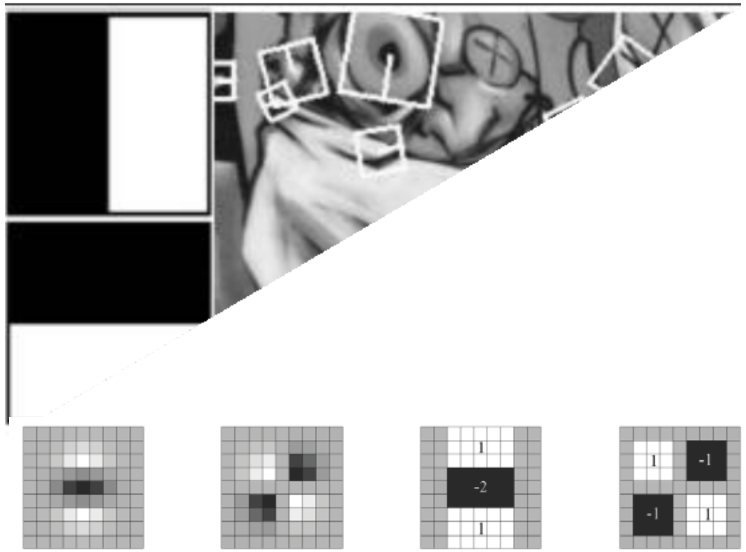
1 - *specificity*

Single number: Area Under the Curve (AUC)

E.g. AUC = 0.87

1 is the best

Local descriptors: SURF, ORB, BRISK



SURF: fast approximation of SIFT idea

- Efficient computation by 2D box filters & integral images
- $\Rightarrow$ 6 times faster than SIFT
- Equivalent quality for object identification

GPU implementation available

- **Feature extraction @ 200Hz**
- **(detector + descriptor, 640×480 img)**
- http://www.vision.ee.ethz.ch/~surf

[Bay, ECCV'06], [Cornelis, CVGPU'08]

K. Grauman, B. Leibe

- Many local feature detectors have executables available online:
  - http://www.robots.ox.ac.uk/~vgg/research/affine
  - http://www.cs.ubc.ca/~lowe/keypoints/
  - http://www.vision.ee.ethz.ch/~surf

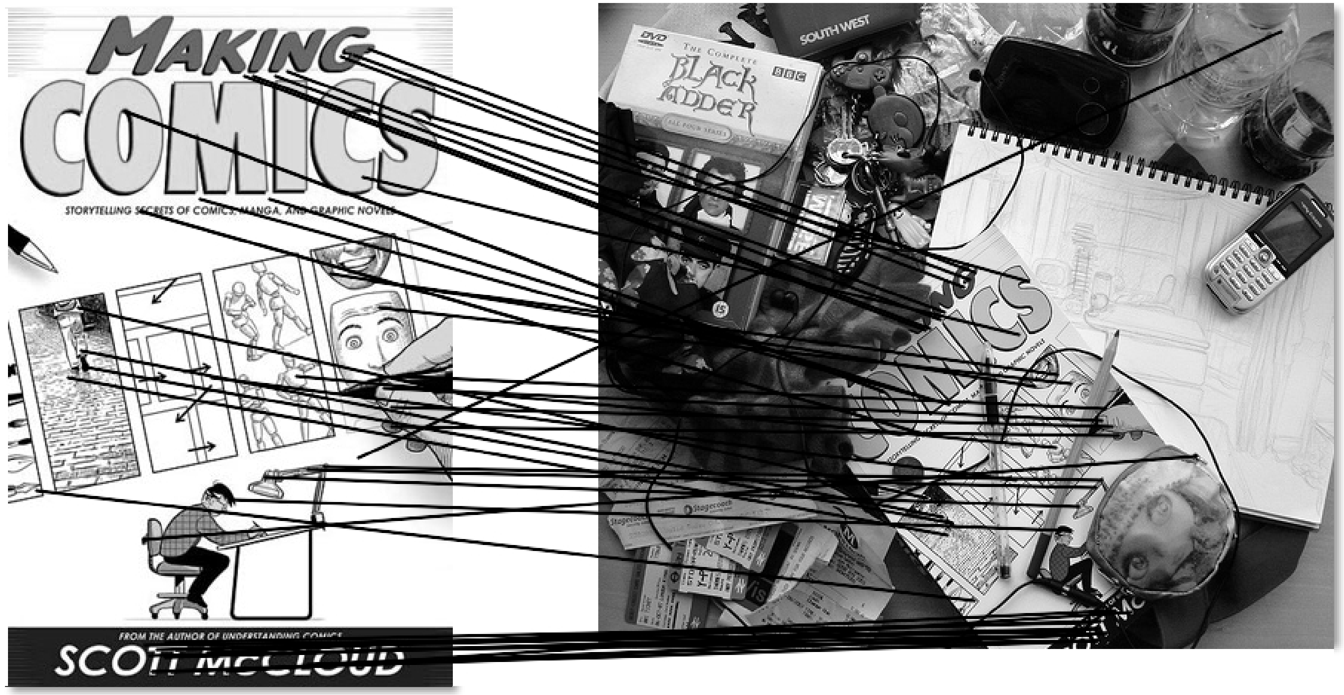
Image alignment

Reading

- Szeliski: Chapter 8.1, 8.2

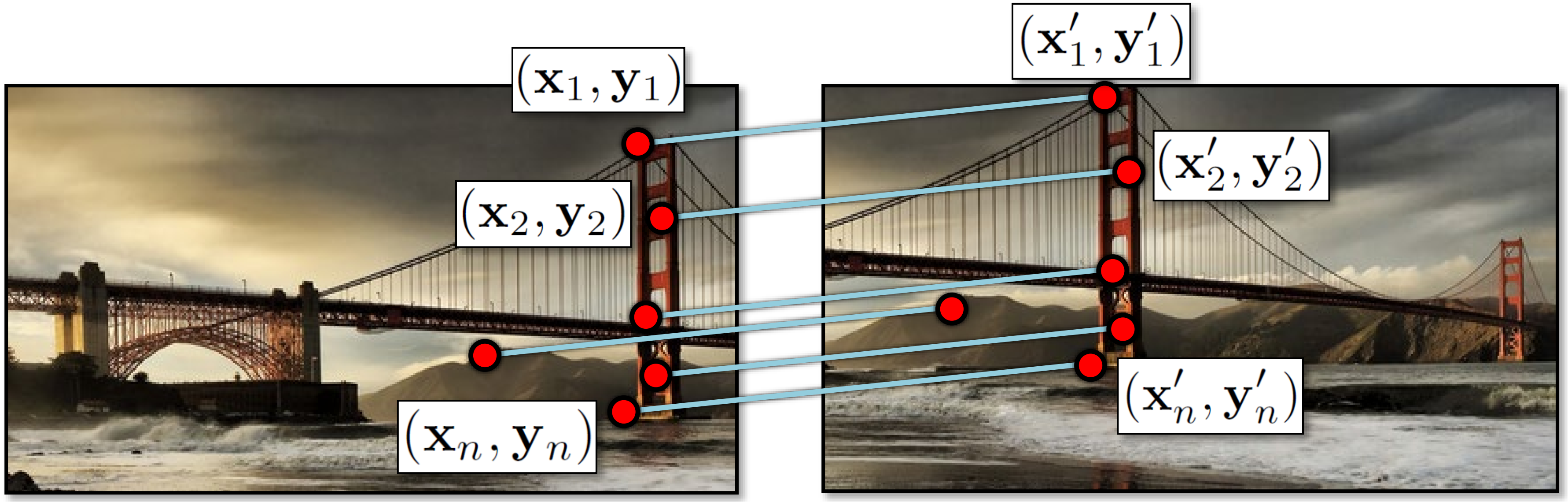
Computing transformations

- Given a set of matches between images A and B
  – How can we compute the transform T from A to B?



  – Find transform T that best “agrees” with the matches

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
- 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

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

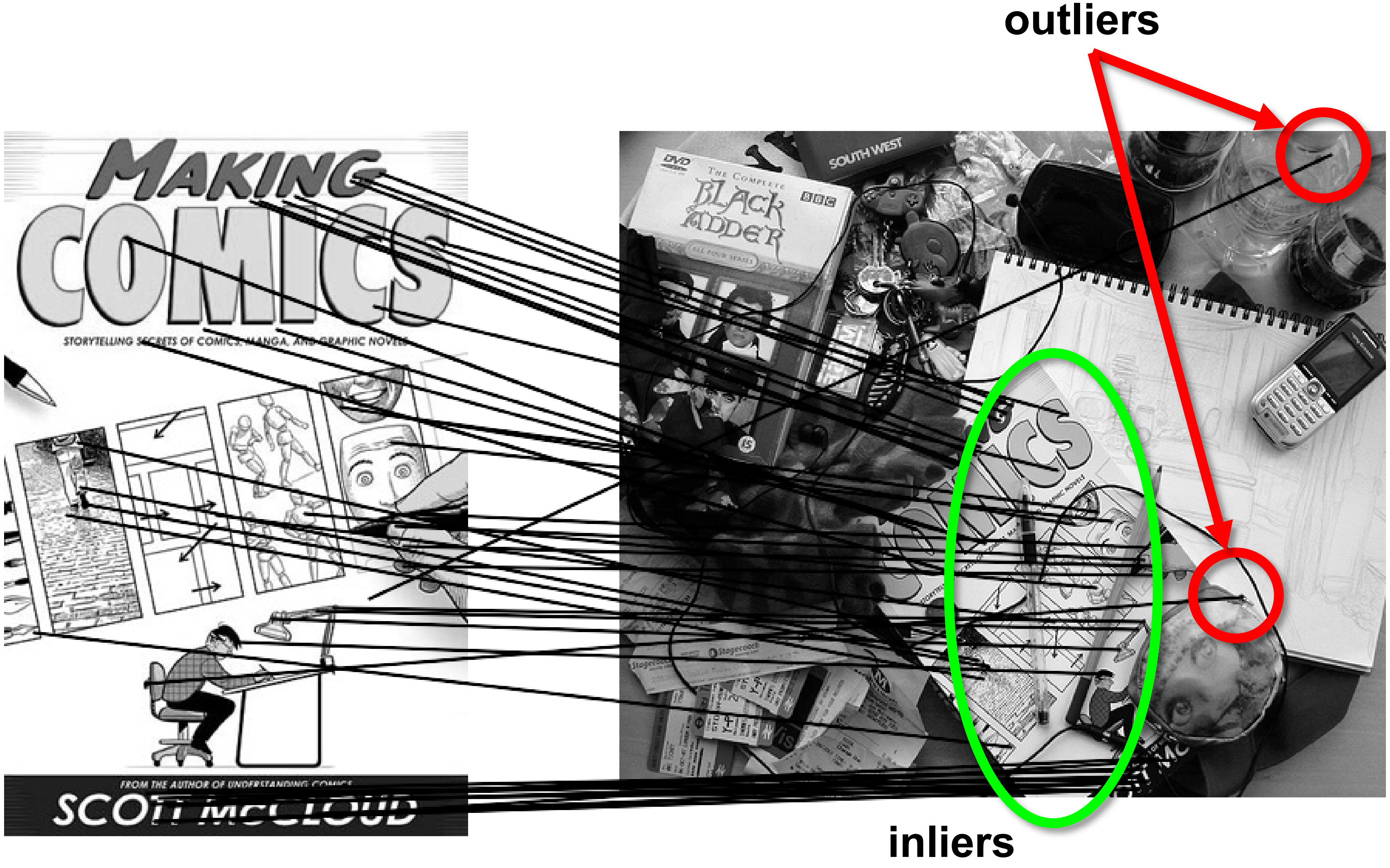
- "Least squares" solution
- For translations, is equal to mean (average) displacement

Image Alignment Algorithm

Given images A and B

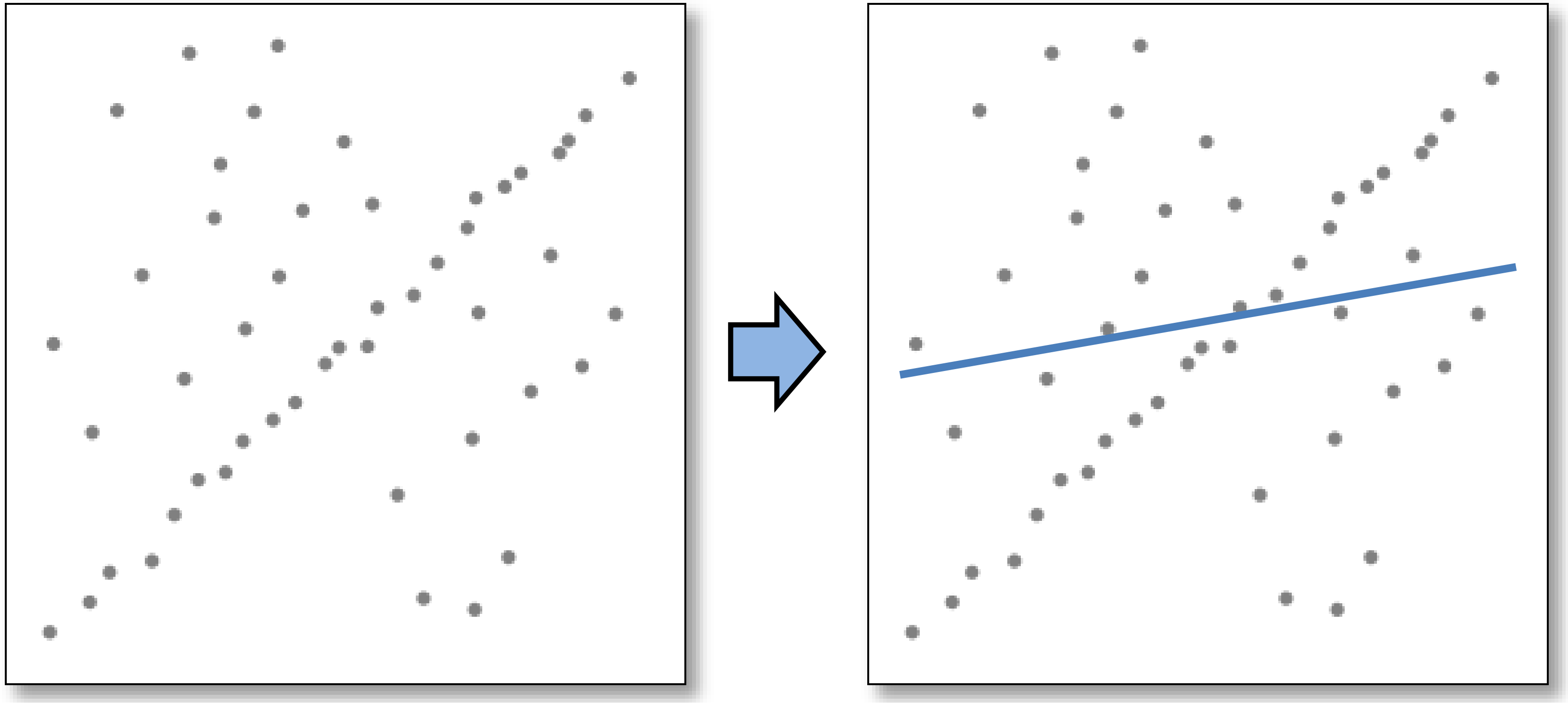
1. Compute image features for A and B
2. Match features between A and B
3. Compute homography between A and B using least squares formulation (linear regression) on set of matches

Outliers


outliers
MAKING
COMICS
STORYTELLING SECRETS OF COMICS, MANGA, AND GRAPHIC NOVELS
FROM THE AUTHOR OF UNDERSTANDING COMICS
SCOTT McCLOUD
SOUTH WEST
DVD
THE COMPLETE
BLACK ADDER
BBC
inliers

Robustness

- Let’s consider the problem of linear regression



Problem: Fit a line to these datapoints

Least squares fit

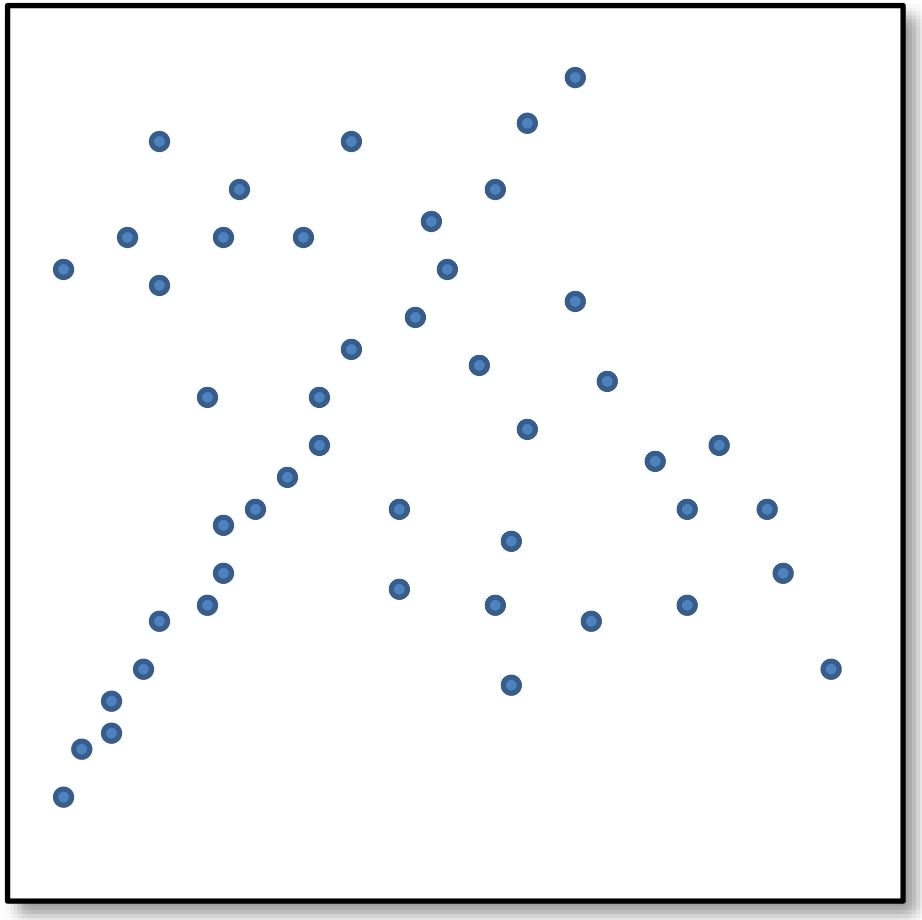
- How can we fix this?

Idea

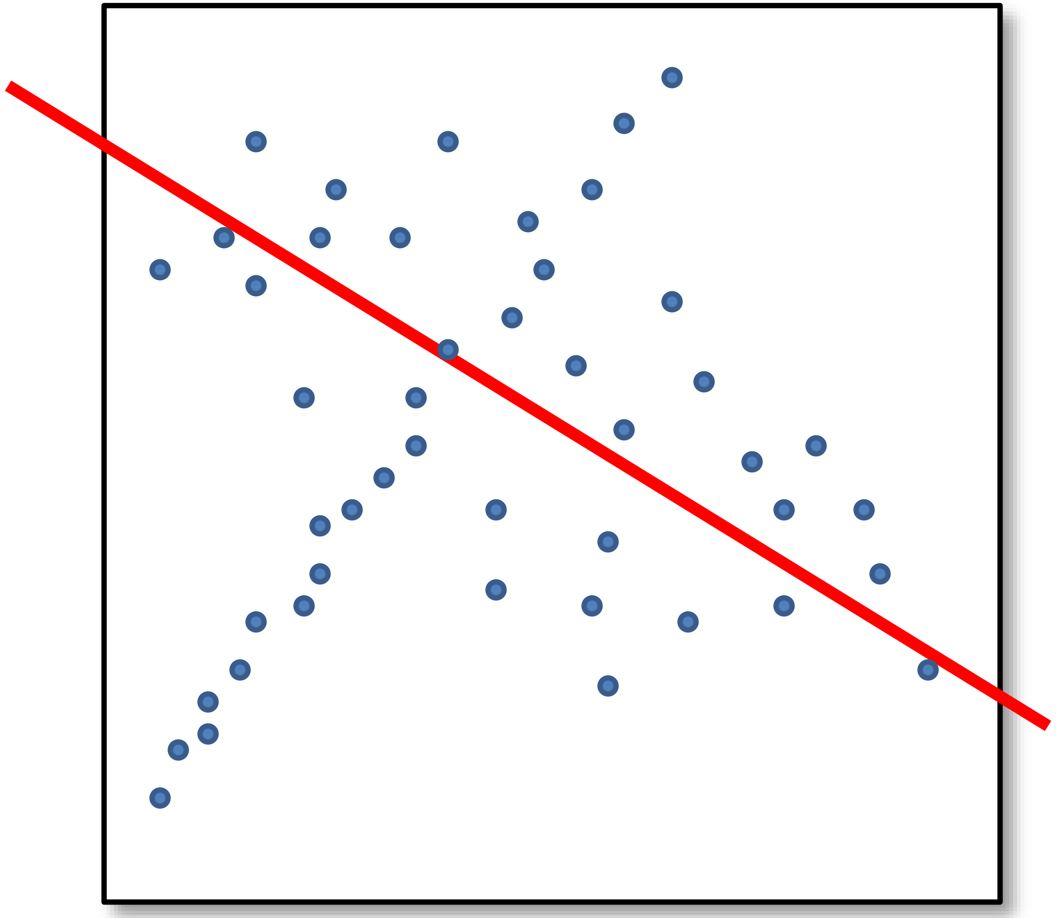
- Given a hypothesized line
- Count the number of points that “agree” with the line
  - “Agree” = within a small distance of the line
  - I.e., the **inliers** to that line

- For all possible lines, select the one with the largest number of inliers

Counting inliers

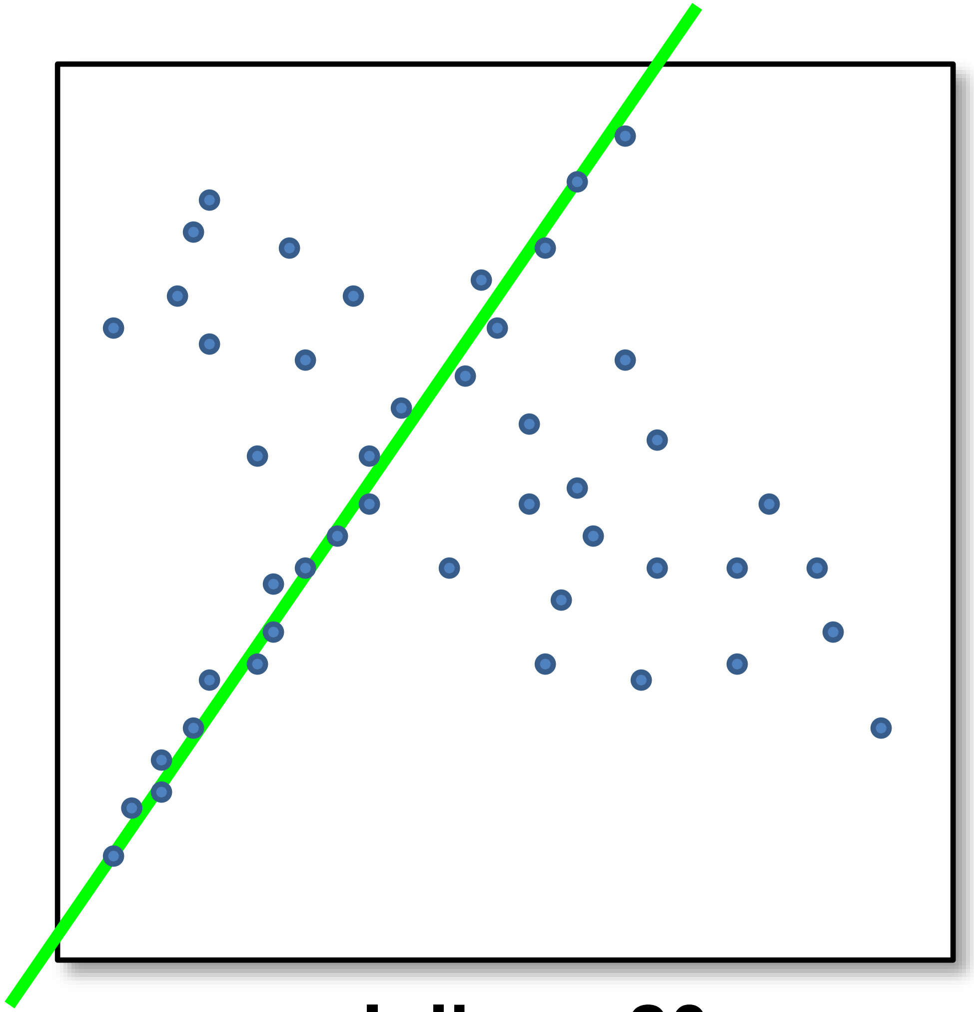


Counting inliers



Inliers: 3

Counting inliers

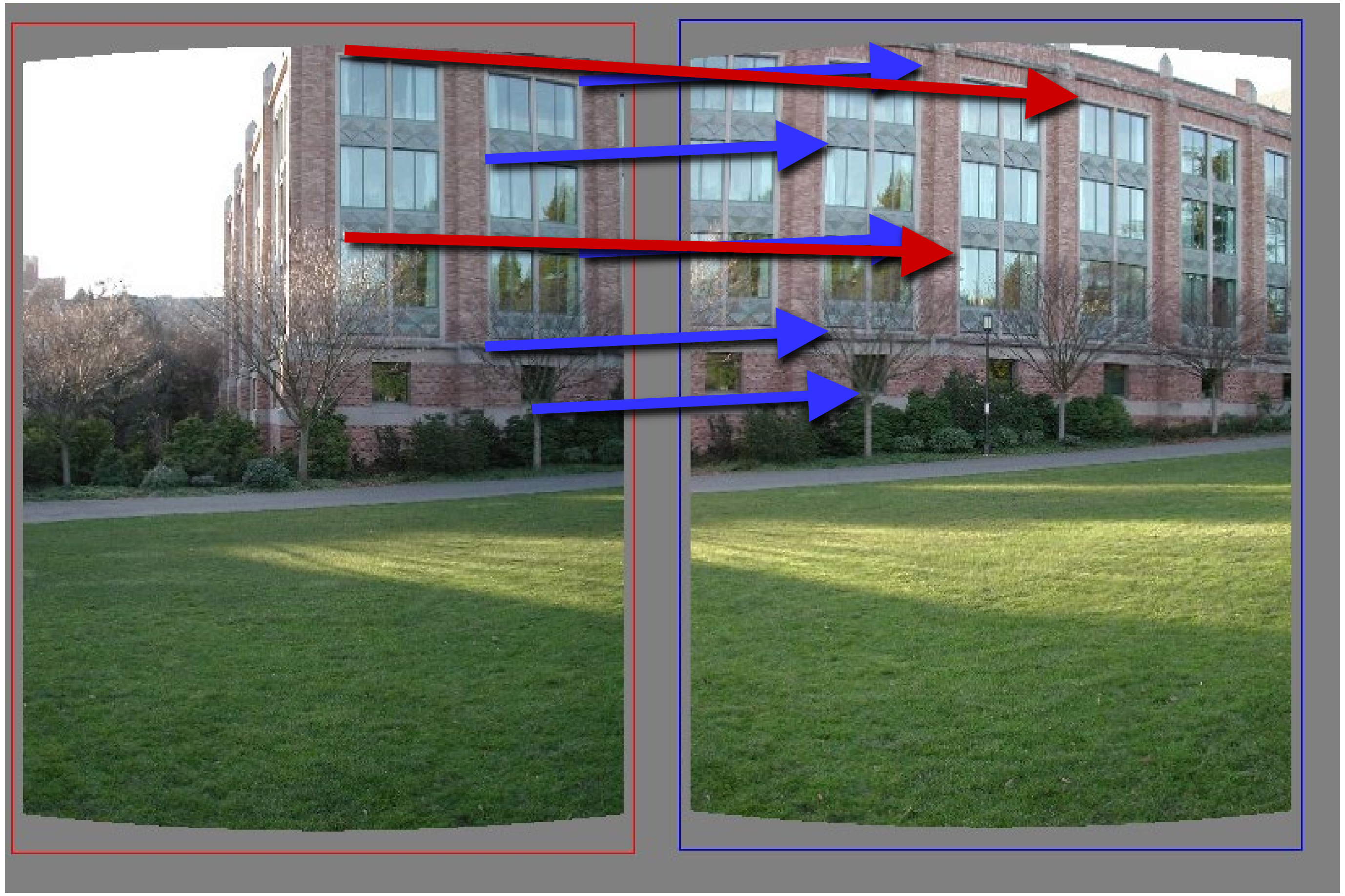


Inliers: 20

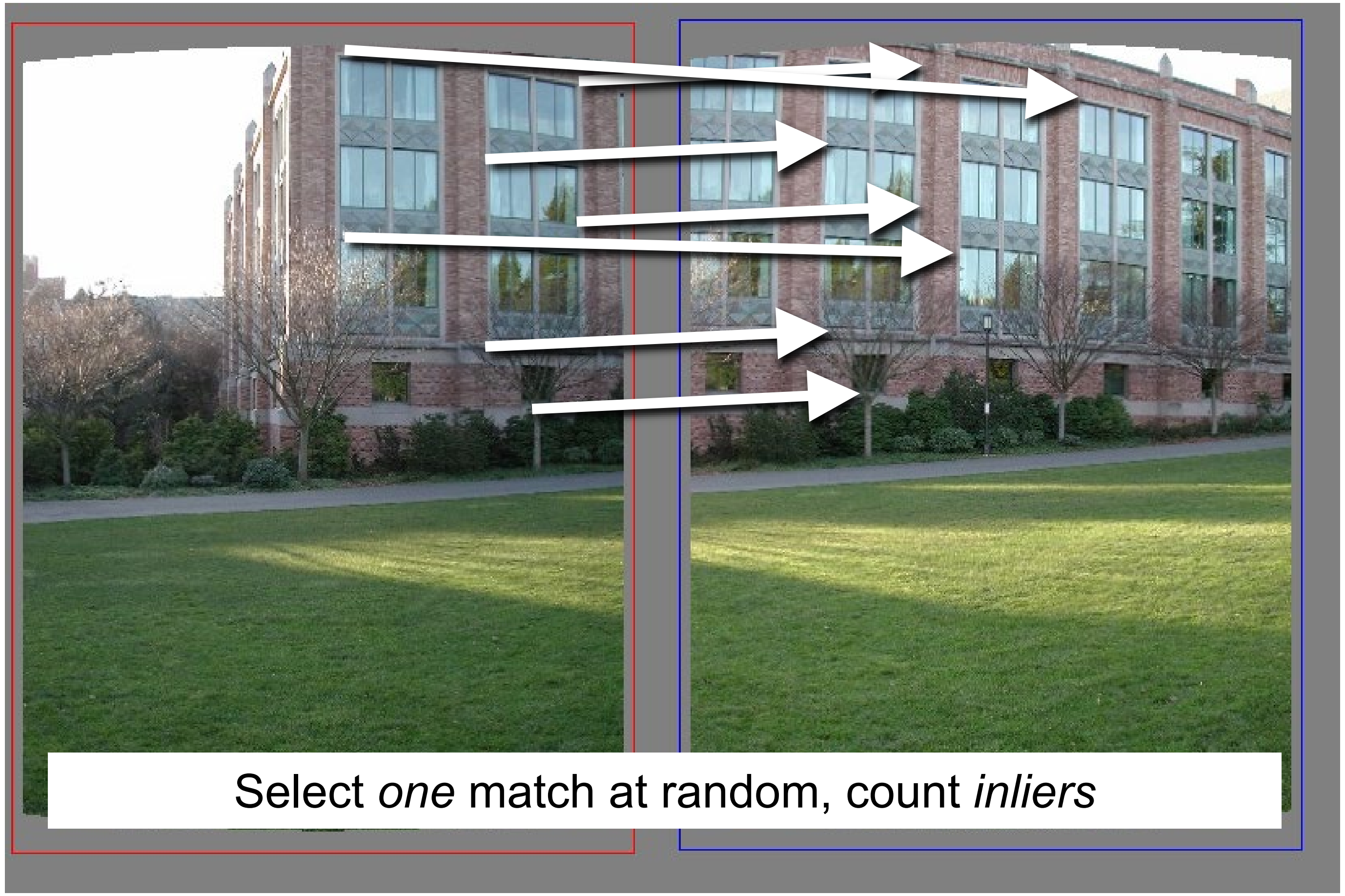
How do we find the best line?

- Unlike least-squares, no simple closed-form solution
- Hypothesize-and-test
  - Try out many lines, keep the best one
  - Which lines?

Translations

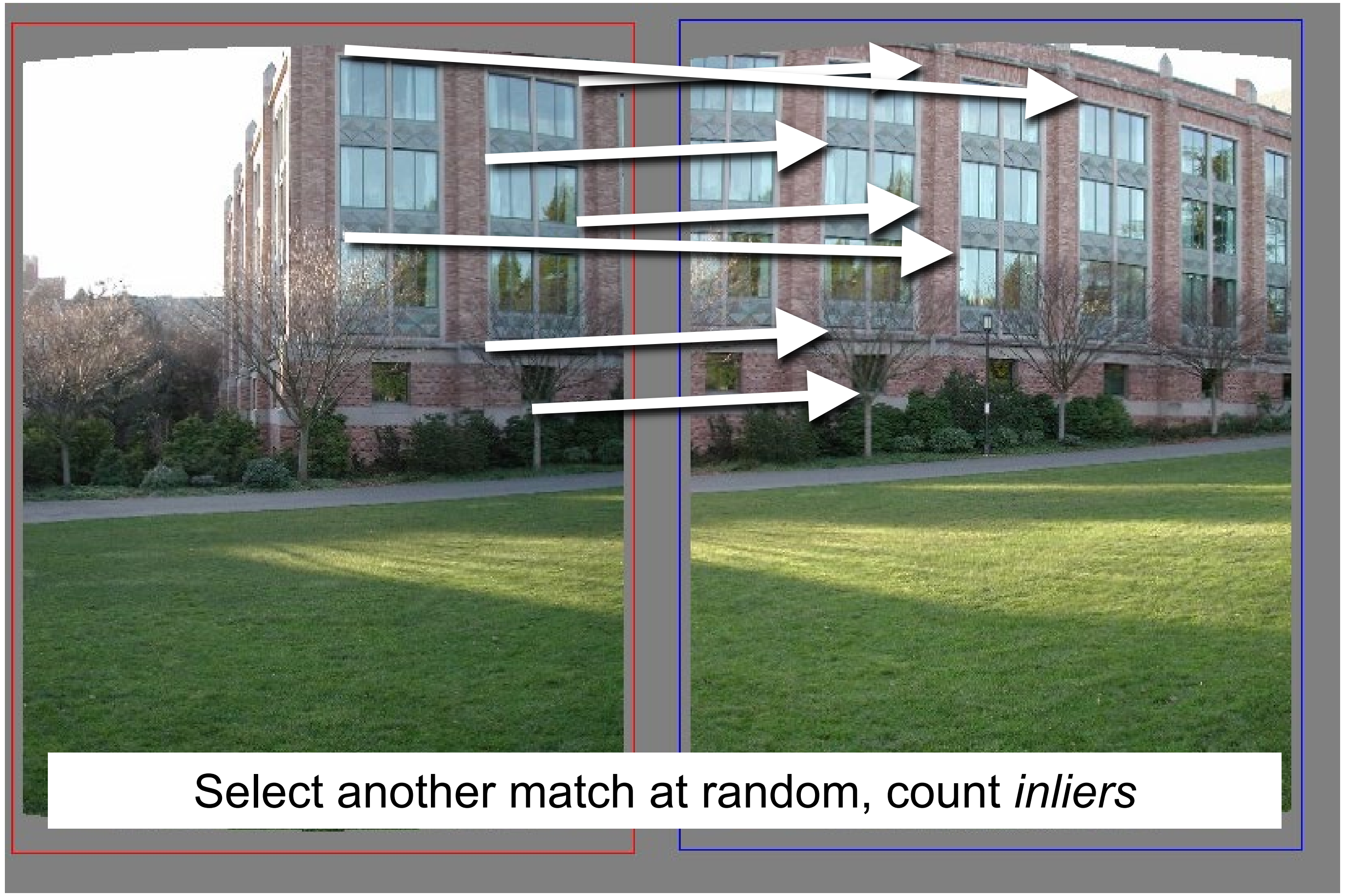


RAndom SAmple Consensus (RANSAC)

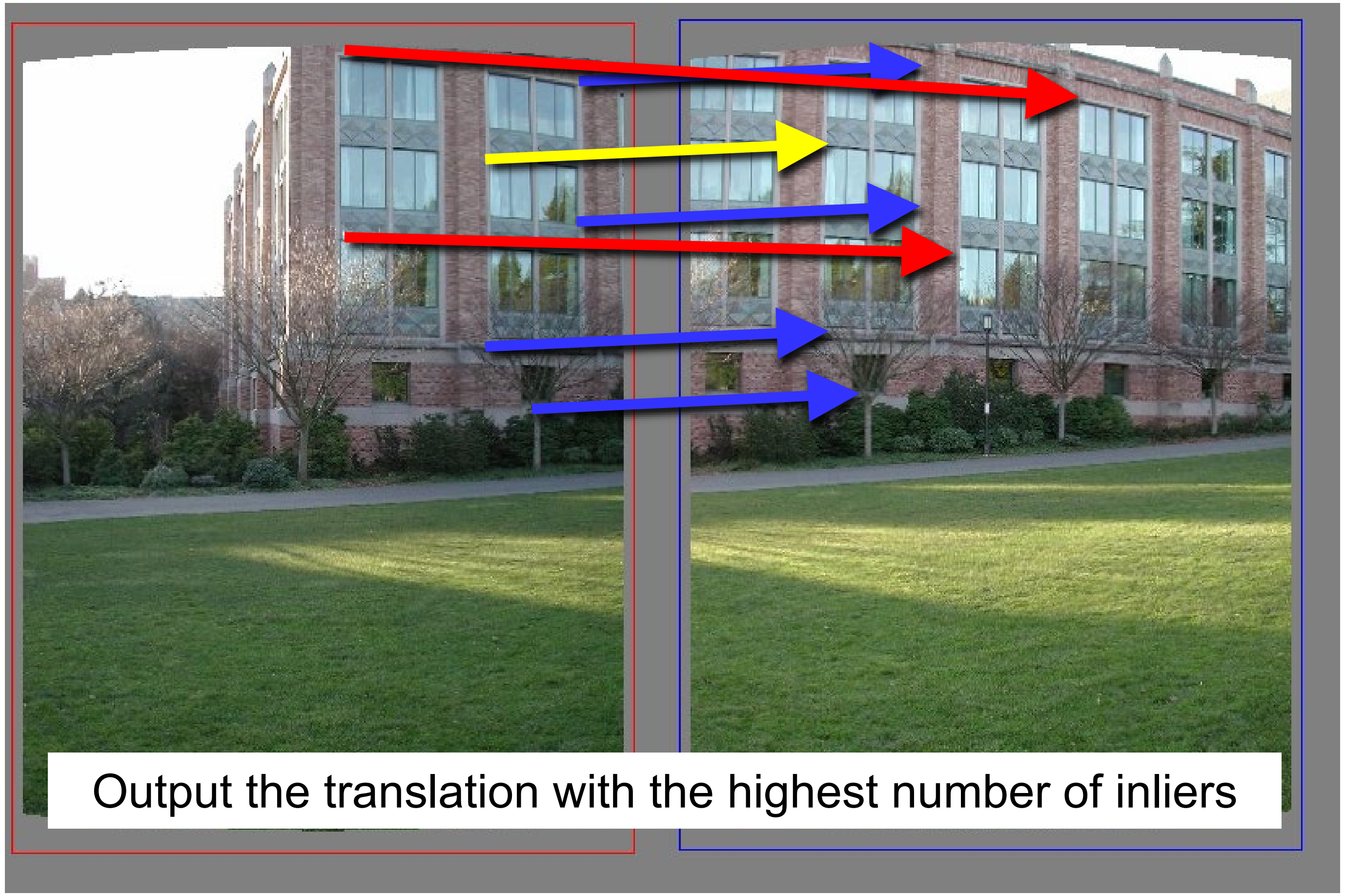


Select *one* match at random, count *inliers*

RAndom SAmple Consensus



RAndom SAmple Consensus



Output the translation with the highest number of inliers

RANSAC

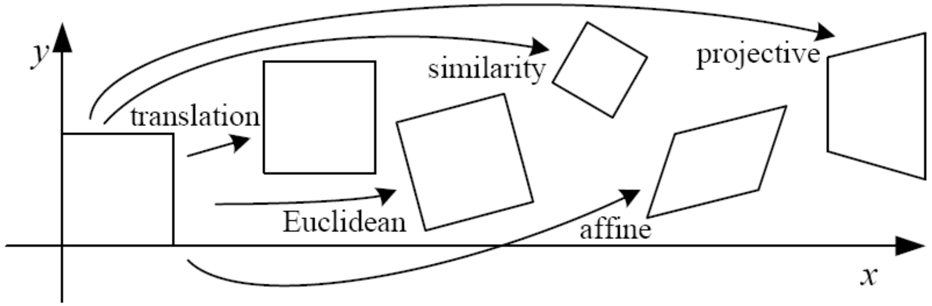
- Idea:
  - All the inliers will agree with each other on the translation vector; the (hopefully small) number of outliers will (hopefully) disagree with each other
    - RANSAC has guarantees of success in selection the right transformation if there are < 50% outliers

RANSAC

- General version:
  1. Randomly choose *s* samples
     - Typically *s* = minimum sample size that lets you fit a model
  2. Fit a model (e.g., line) to those samples
  3. Count the number of inliers that approximately fit the model
  4. Repeat *N* times
  5. Choose the model that has the largest set of inliers

How big is s?

- For alignment, depends on the motion model
  - Here, each sample is a correspondence (pair of matching points)



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} I \mid t \end{bmatrix}_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} R \mid t \end{bmatrix}_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} sR \mid t \end{bmatrix}_{2\times 3}$	4	angles $+\cdots$	
affine	$\begin{bmatrix} A \end{bmatrix}_{2\times 3}$	6	parallelism $+\cdots$	
projective	$\begin{bmatrix} \tilde{H} \end{bmatrix}_{3\times 3}$	8	straight lines	

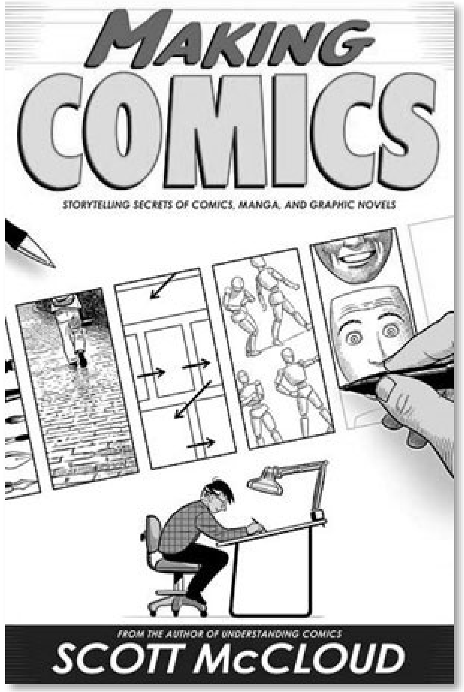
Image stitching

- Now we know how to create panoramas

- Given two images:
  - Step 1: Detect features
  - Step 2: Match features
  - Step 3: Compute a homography using RANSAC
  - Step 4: Combine the images together

Transformations and warping

What is the geometric relationship between these two images?

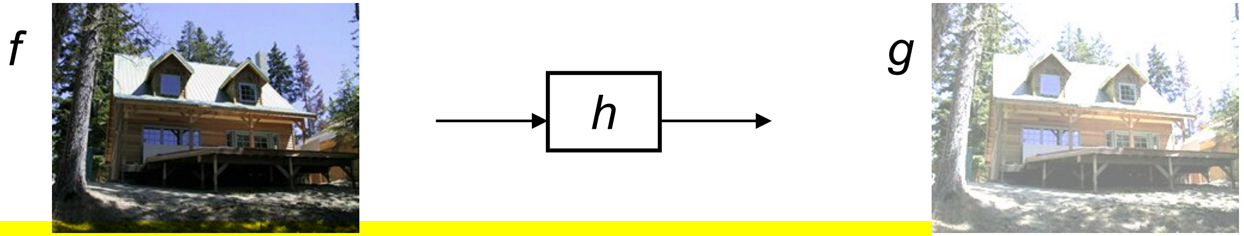


?

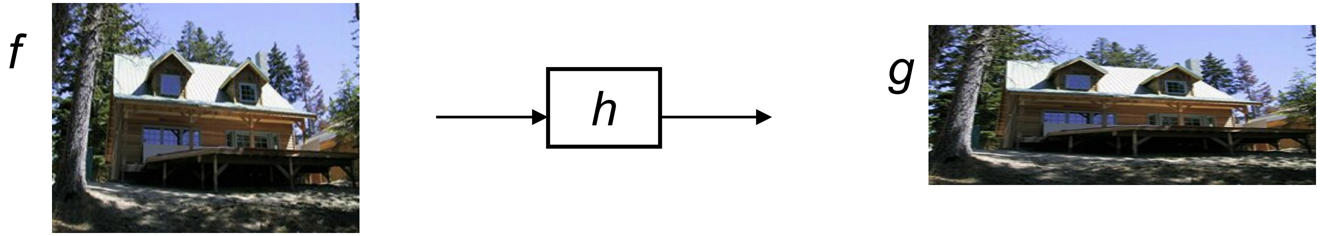
Answer: Similarity transformation (translation, rotation, uniform scale)

Image Warping

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



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



Parametric (global) warping

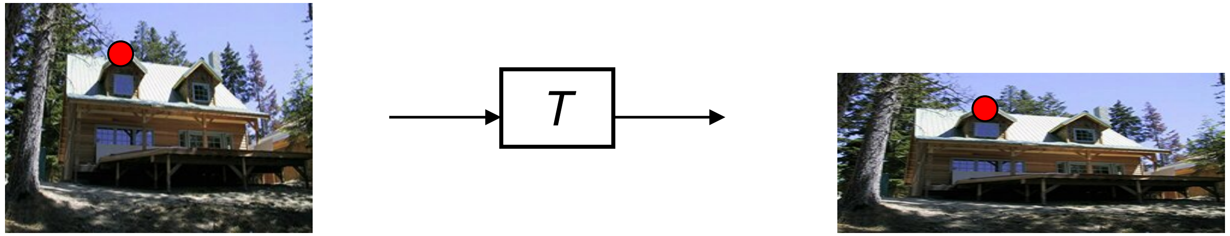
- Examples of parametric warps:

translation

rotation

aspect

Parametric (global) warping



p = (x,y) **p'** = (x',y')

- Transformation T is a coordinate-changing machine:

  p' = *T*(**p**)

- What does it mean that *T* is global?
  - Is the same for any point **p**

- Let's consider *linear* transformation (can be represented by a 2x2 matrix):

$$\mathbf{p}' = \mathbf{T}\mathbf{p} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \end{bmatrix}$$

Common linear transformations

- Uniform scaling by *s*:

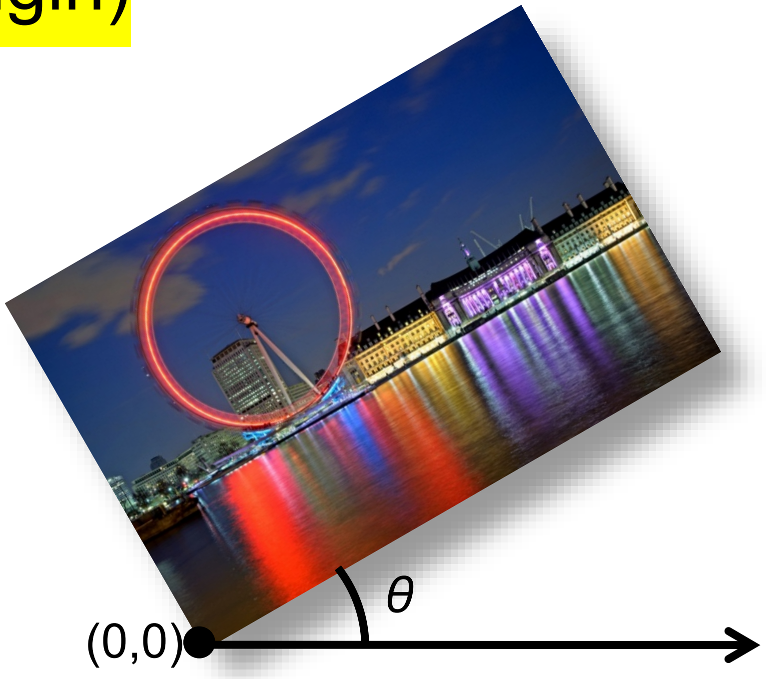
(0,0)

(0,0)

$$\mathbf{S} = \begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$$

Common linear transformations

- Rotation by angle θ (about the origin)



$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

What is the inverse?

For rotations:

$$\mathbf{R}^{-1} = \mathbf{R}^T$$

Transformation with 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D mirror about Y axis

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

2D mirror across line y = x

$$\begin{aligned} x' &= y \\ y' &= x \end{aligned} \qquad \mathbf{T} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

All 2D Linear Transformations

- Linear transformations are combinations of
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

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

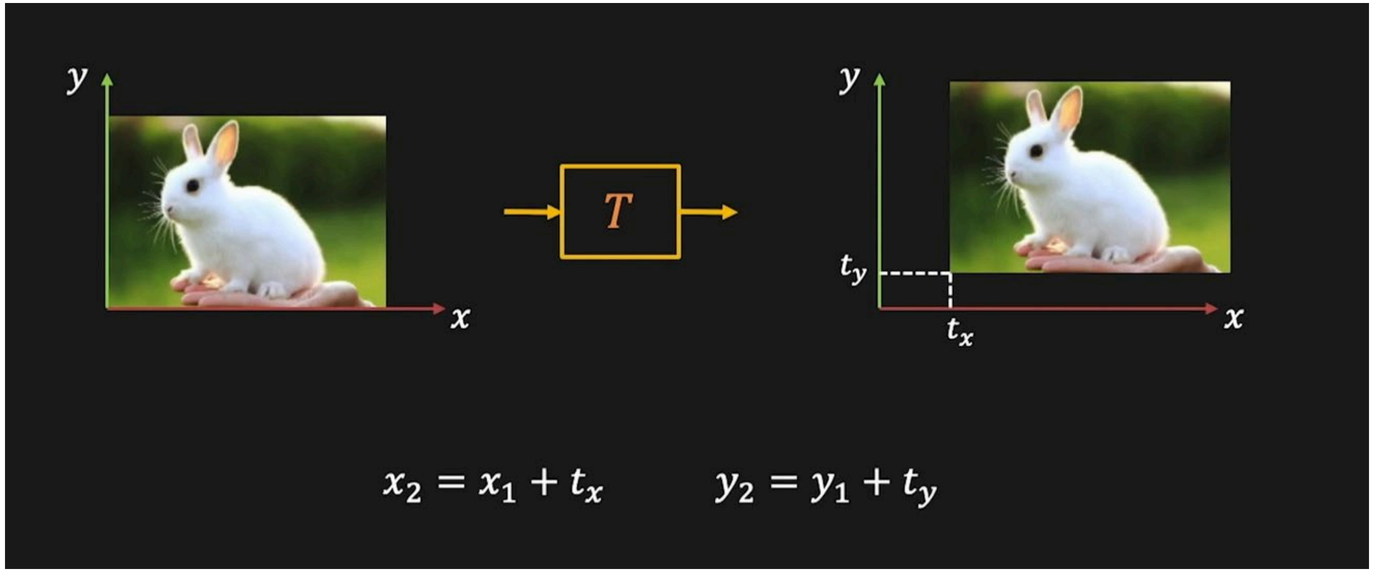
- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} \begin{bmatrix} i & j \\ k & l \end{bmatrix}}_{T'=T1T2T3} \begin{bmatrix} x \\ y \end{bmatrix}$$

Transformation with 2x2 Matrices

- What types of transformations can be represented with a 2x2 matrix?

2D Translation?



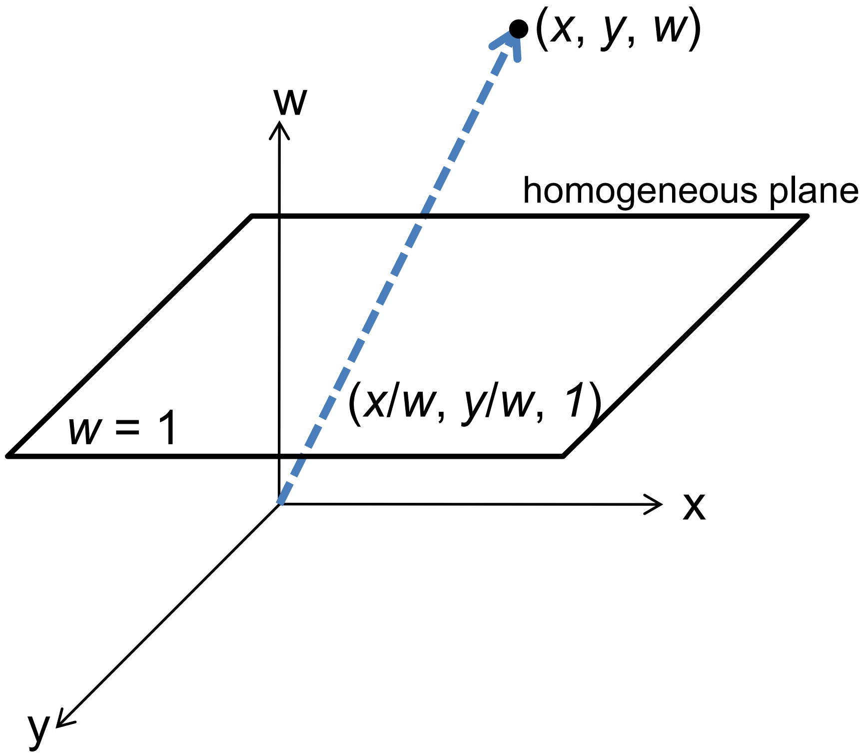
Translation is not a linear operation on 2D coordinates

Homogeneous coordinates

Trick: add one more coordinate:

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates



Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

Translation

- Solution: homogeneous coordinates

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

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

Affine transformations

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

any transformation represented by a 3x3 matrix with last row [0 0 1] is named *affine* transformation

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

Basic affine transformations

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

Translate

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{s}_x & 0 & 0 \\ 0 & \boldsymbol{s}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Scale

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

2D *in-plane* rotation

$$\begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{sh}_x & 0 \\ \boldsymbol{sh}_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ 1 \end{bmatrix}$$

Shear/Deformation

Affine transformations

- Affine transformations are combinations of:
  – Linear transformations, and
  – Translations

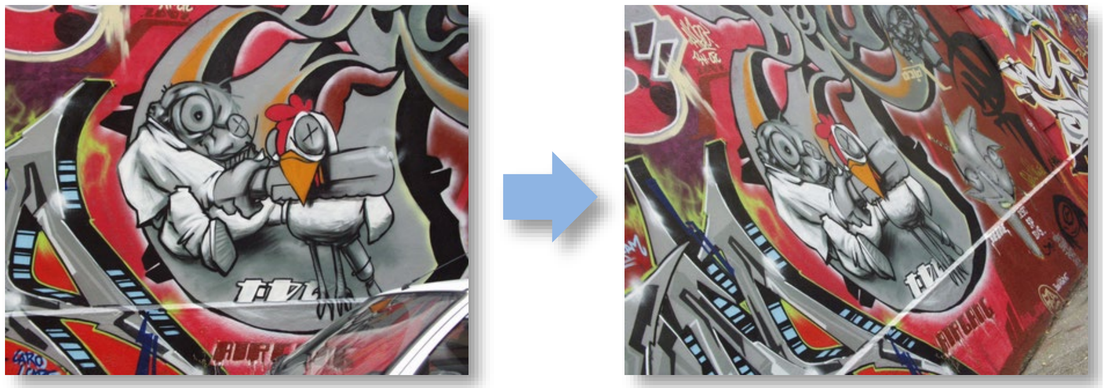
$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Properties of affine transformations:
  – Origin does not necessarily map to origin
  – Lines map to lines
  – Parallel lines remain parallel
  – Ratios are preserved
  – Closed under composition

Projective Transformations *aka* Homographies *aka* Planar Perspective Maps

$$\mathbf{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

Called a *homography*
(or *planar perspective map*)



Homographies

- Homographies
  - Affine transformations, and
  - Projective warps

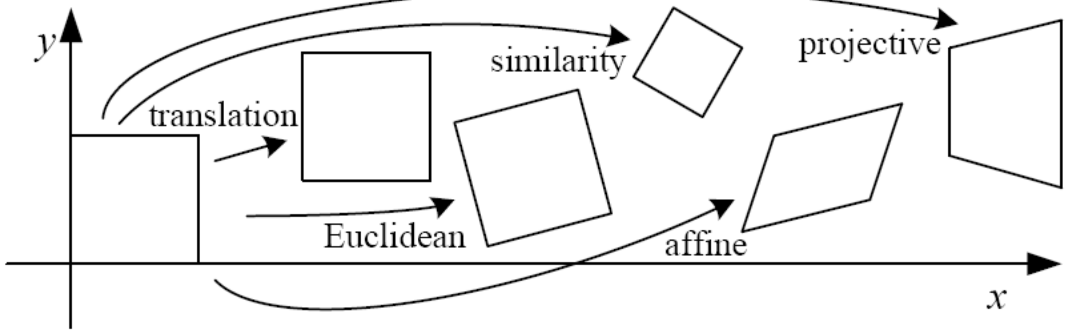
$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

- Properties of projective transformations:
  - Origin does not necessarily map to origin
  - Lines map to lines
  - Parallel lines do not necessarily remain parallel
  - Ratios are not preserved
  - Closed under composition

Alternate formulation for homographies

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

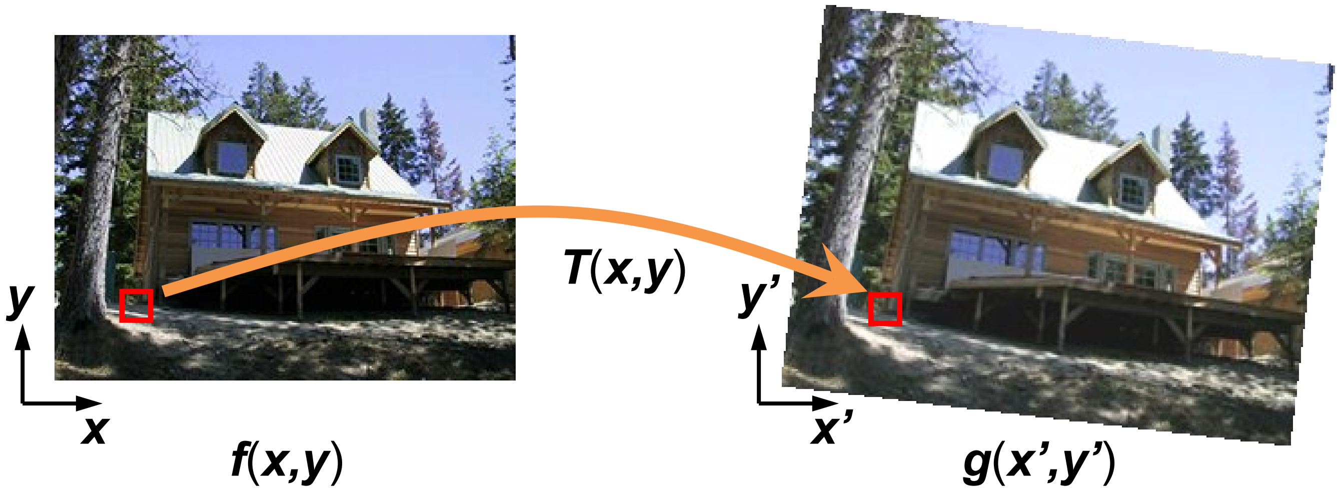
2D image transformations



Name	Matrix	# D.O.F.	Preserves:	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation $+\cdots$	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths $+\cdots$	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles $+\cdots$	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism $+\cdots$	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

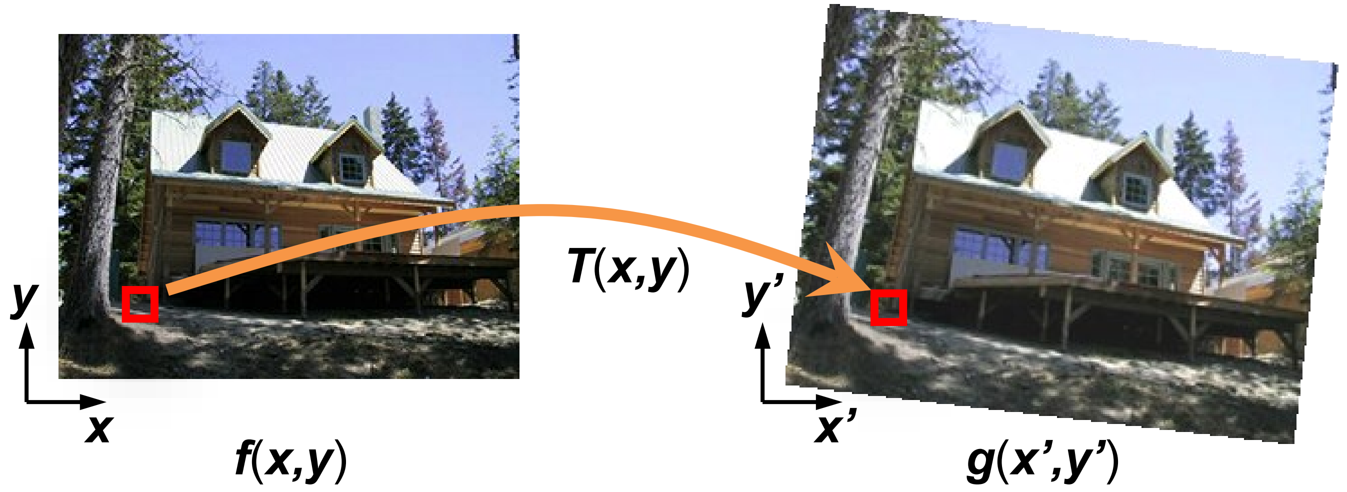
Implementing image warping

- Given a coordinate transformation $(\boldsymbol{x'},\boldsymbol{y'}) = \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y})$ and a source image $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$, how do we compute a transformed image $\boldsymbol{g}(\boldsymbol{x'},\boldsymbol{y'}) = \boldsymbol{f}(\boldsymbol{T}(\boldsymbol{x},\boldsymbol{y}))$?



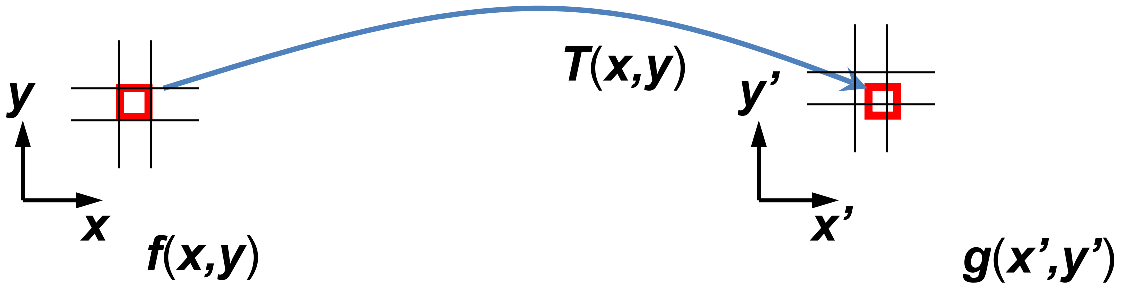
Forward Warping

- Send each pixel $\boldsymbol{f}(\boldsymbol{x})$ to its corresponding location $(\boldsymbol{x'},\boldsymbol{y'}) = \boldsymbol{T}(\boldsymbol{x},\boldsymbol{y})$ in $\boldsymbol{g}(\boldsymbol{x'},\boldsymbol{y'})$
  - What if pixel lands "between" two pixels?



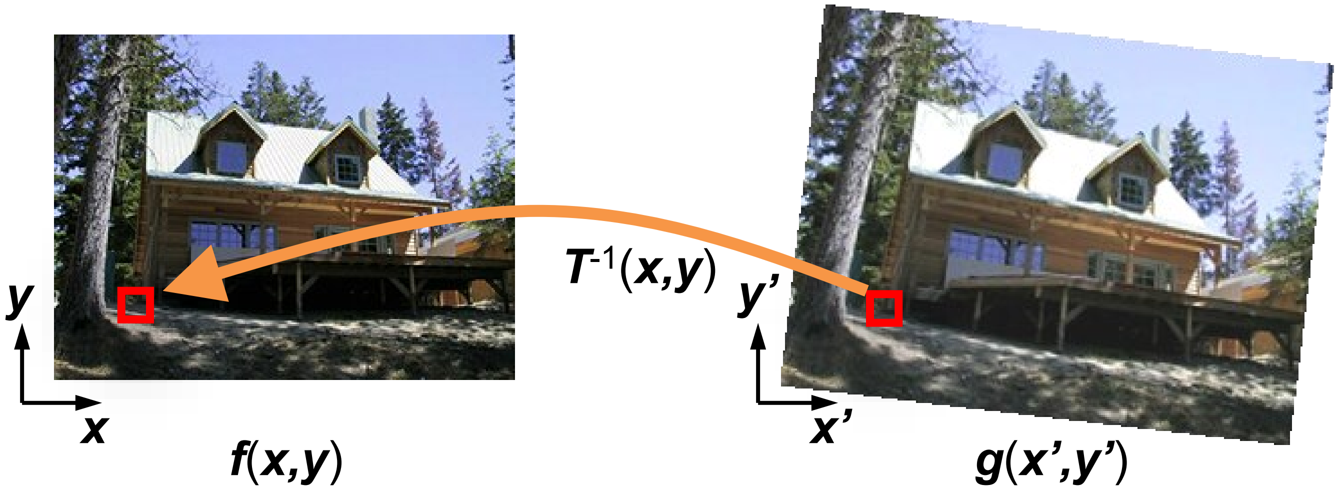
Forward Warping

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



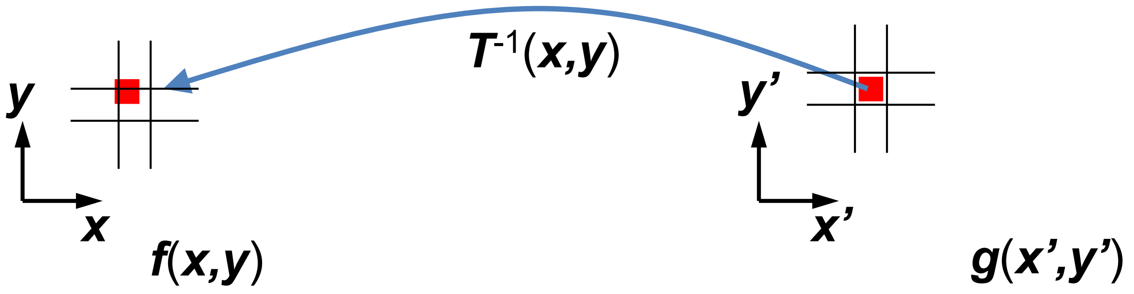
Inverse Warping

- Get each pixel $\boldsymbol{g}(\boldsymbol{x'},\boldsymbol{y'})$ from its corresponding location $(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{T}^{-1}(\boldsymbol{x},\boldsymbol{y})$ in $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$
  - Requires taking the inverse of the transform
  - What if pixel comes from “between” two pixels?



Inverse Warping

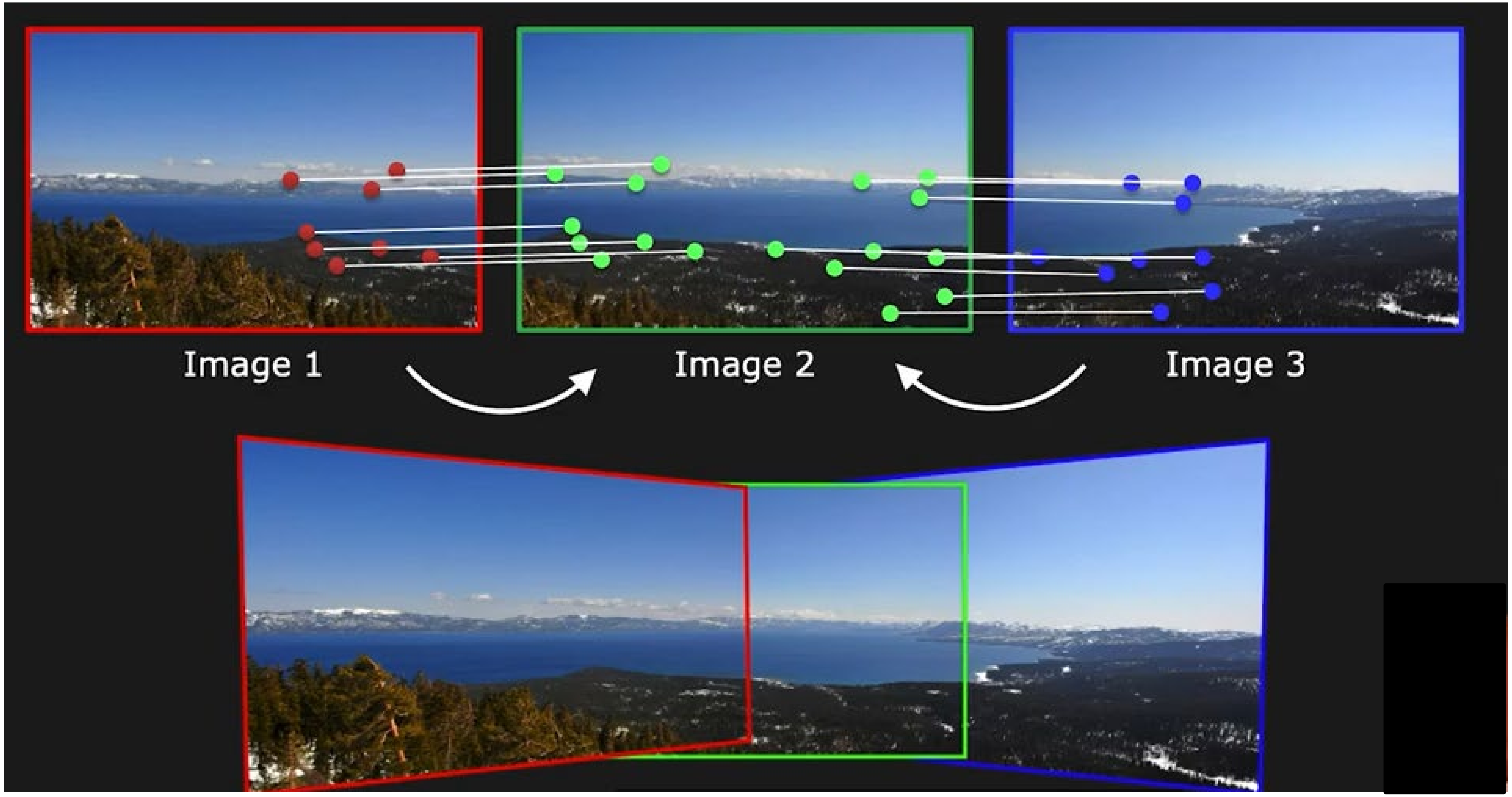
- Get each pixel $\boldsymbol{g}(\boldsymbol{x'})$ from its corresponding location $\boldsymbol{x'} = \boldsymbol{h}(\boldsymbol{x})$ in $\boldsymbol{f}(\boldsymbol{x})$
  - What if pixel comes from “between” two pixels?
  - Answer: *resample* color value from *interpolated* (*prefiltered*) source image



Interpolation

- Possible interpolation filters:
  - nearest neighbor
  - bilinear
  - bicubic
  - sinc
- Needed to prevent artifacts

Image stitching



Acknowledgement: some slides and material from Bernt Schiele, Mario Fritz, Michael Black, Bill Freeman, Fei-Fei, Justin Johnson, Serena Yeung, R. Szelisky, Ioannis Gkioulekas. K. Nayar