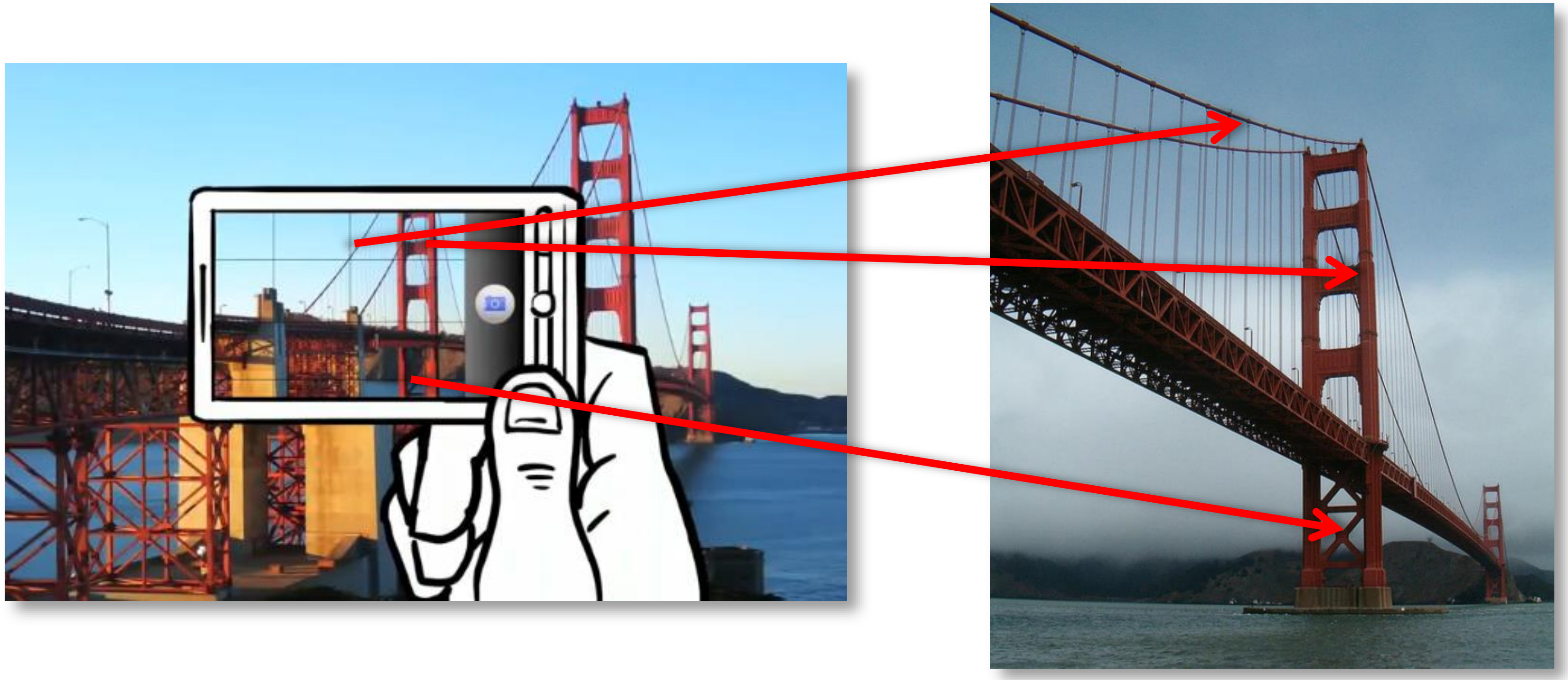
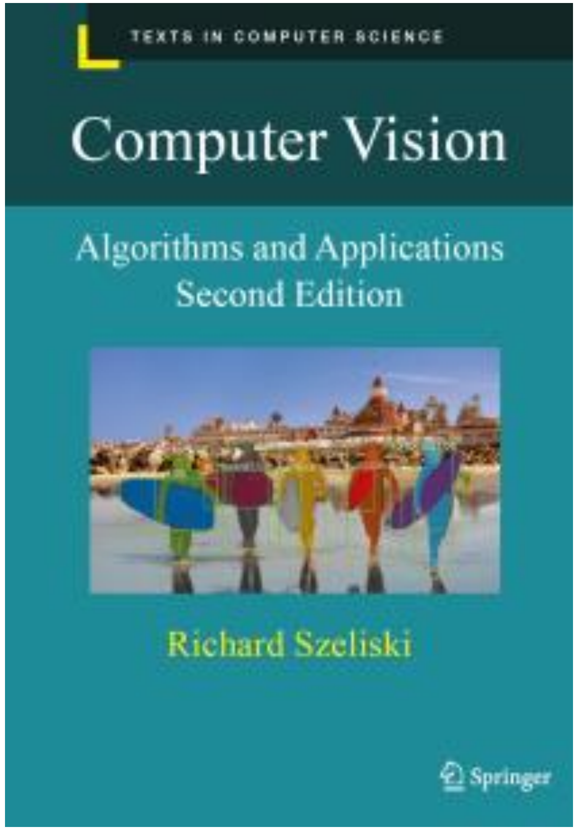
# Computer Vision

## Feature descriptors and feature matching



# Important information



## Textbook

Rick Szeliski, *Computer Vision: Algorithms and Applications* online at: http://szeliski.org/Book/

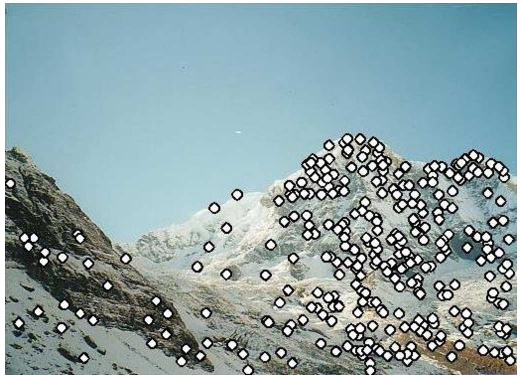
Many of the slides in this course are modified from the excellent class notes of similar courses offered in other schools by Noah Snavely, Prof Yung-Yu Chuang, Fredo Durand, Alyosha Efros, Bill Freeman, James Hays, Svetlana Lazebnik, Andrej Karpathy, Fei-Fei Li, Srinivasa Narasimhan, Silvio Savarese, Steve Seitz, Richard Szeliski, and Li Zhang. The instructor is extremely thankful to the researchers for making their notes available online. Please feel free to use and modify any of the slides, but acknowledge the original sources where appropriate.
All readings are from Richard Szeliski, Computer Vision: Algorithms and Applications, 2nd Edition, unless otherwise noted.

# Reading

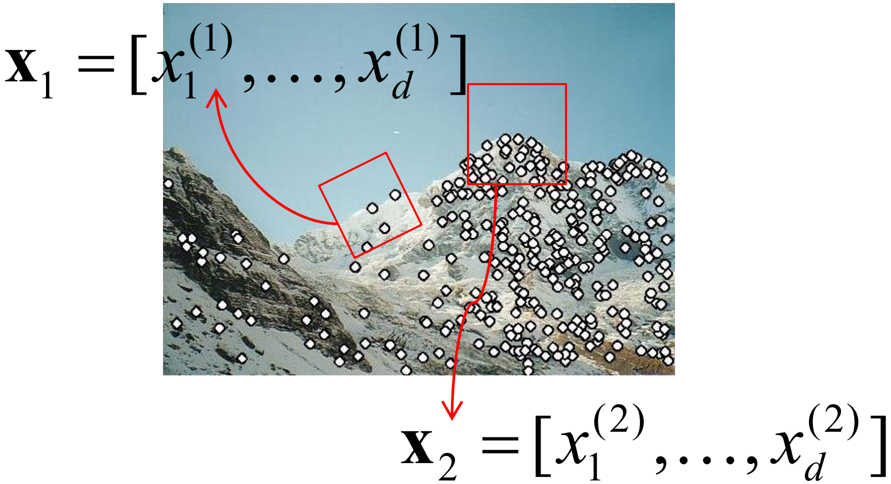
- Szeliski (2nd edition) 7.1

# Local features: main components

1) Detection: Identify the interest points



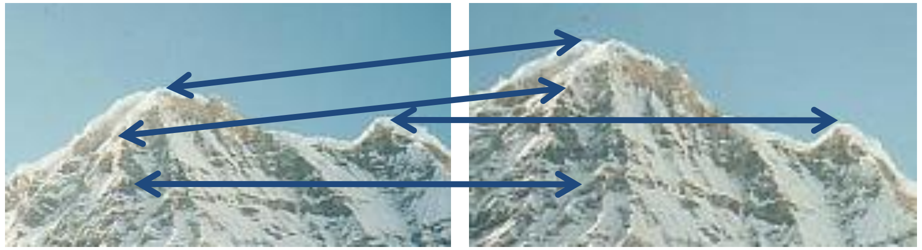
2) **Description: Extract vector feature descriptor surrounding each interest point.**



$$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$$

$$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$$

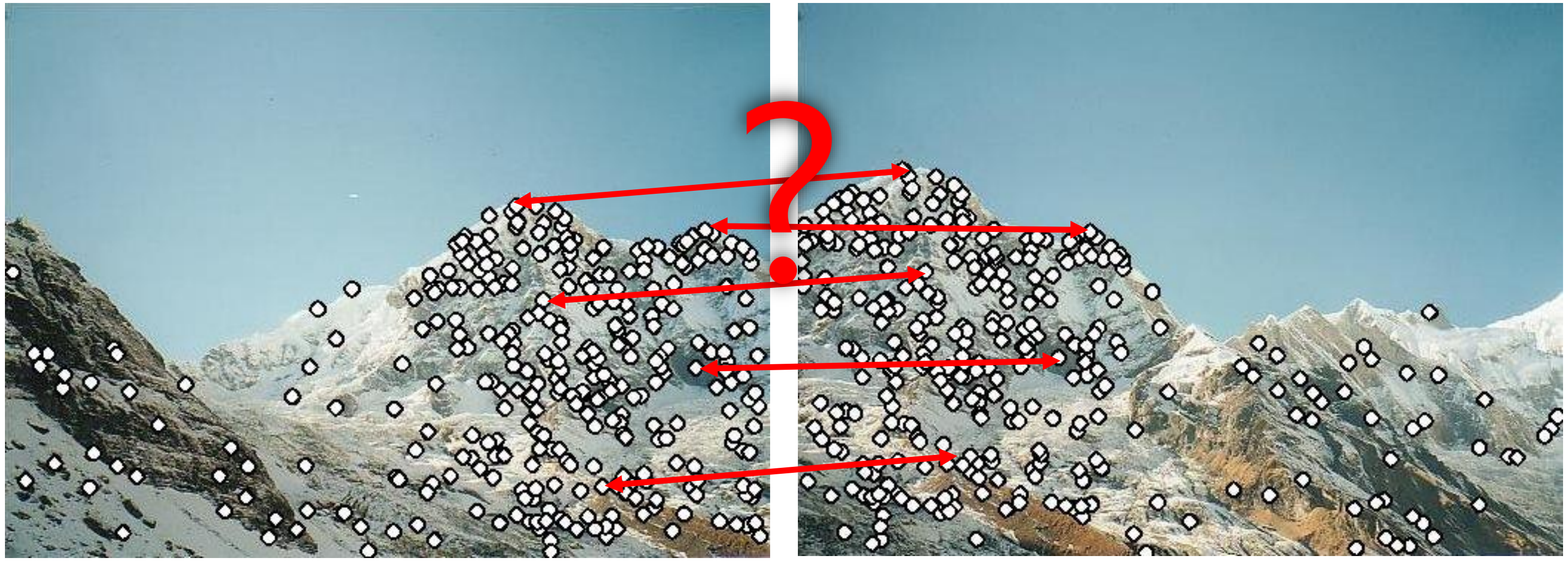
3) Matching: Determine correspondence between descriptors in two views



Kristen Grauman

# Feature descriptors

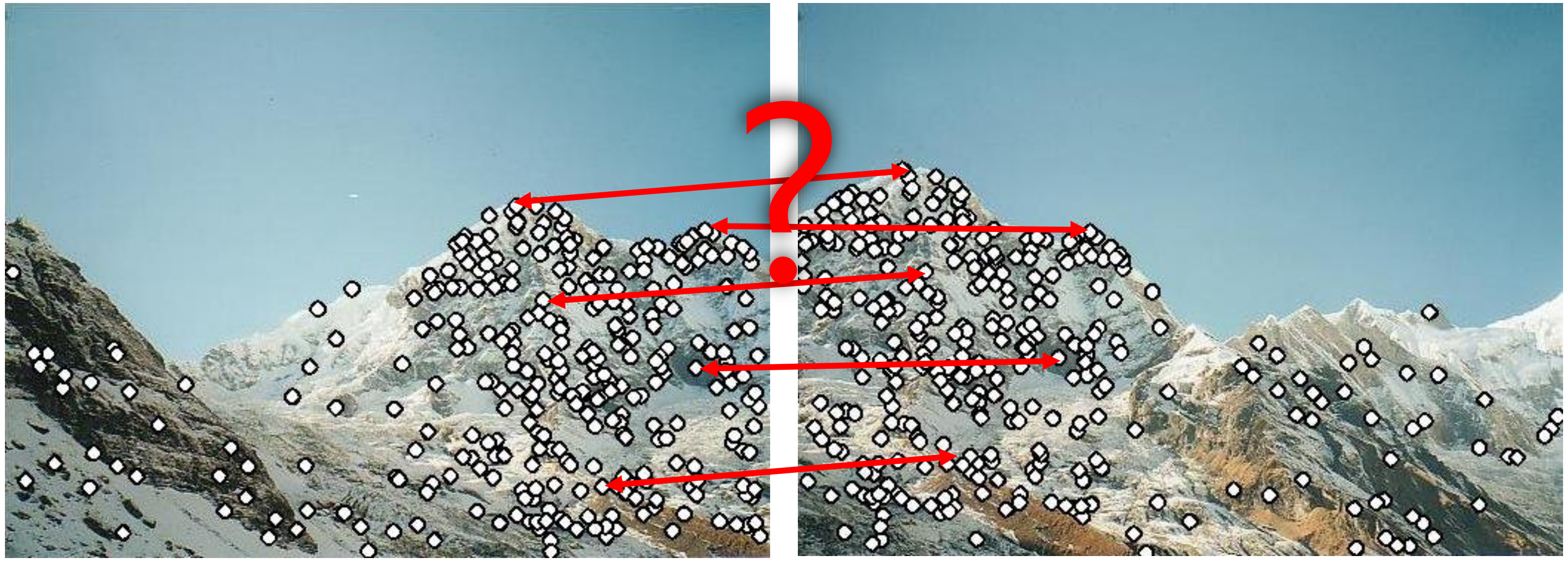
We know how to detect good points
Next question: **How to match them?**



**Answer:** Come up with a *descriptor* for each point,
find similar descriptors between the two images

# Feature descriptors

We know how to detect good points
Next question: **How to match them?**



Lots of possibilities

- Simple option:  match square windows around the point
- State of the art approach:  SIFT
  - David Lowe, UBC  http://www.cs.ubc.ca/~lowe/keypoints/

# Invariance vs. discriminability

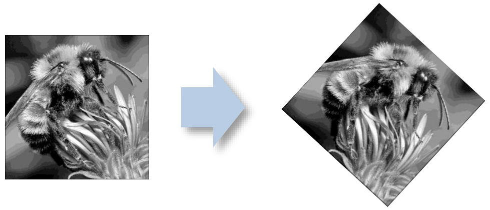
- Invariance:
  - Descriptor shouldn’t change even if image is transformed

- Discriminability:
  - Descriptor should be highly unique for each point

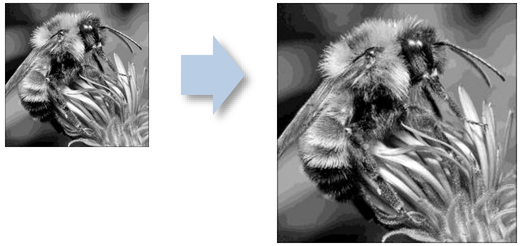
# Image transformations revisited

- Geometric

**Rotation**

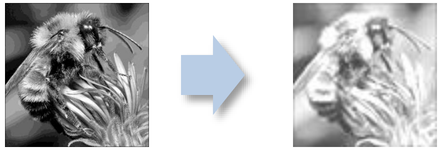


**Scale**



- Photometric

**Intensity change**



# Invariant descriptors

- We looked at invariant / equivariant **detectors**

- Most feature **descriptors** are also designed to be invariant to:
  - Translation, 2D rotation, scale

- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transforms (some are fully affine invariant)
  - Limited illumination/contrast changes

# How to achieve invariance

Need both of the following:

1. Make sure your detector is invariant
2. Design an invariant feature descriptor
   - Simplest descriptor: a single 0
     - What’s this invariant to?
   - Next simplest descriptor:  a square, axis-aligned 5x5 window of pixels
     - What’s this invariant to?
   - Let’s look at some better approaches…

# Rotation invariance for feature descriptors

- Find dominant orientation of the image patch (this orientation should be *equivariant* to image rotation)
  - E.g., given by $\mathbf{x}_{\mathbf{max}}$, the eigenvector of the second moment matrix **H** corresponding to $\lambda_{\text{max}}$ (the larger eigenvalue)
  - Or (better) simply the orientation of the (smoothed) gradient
  - Rotate the patch according to this angle

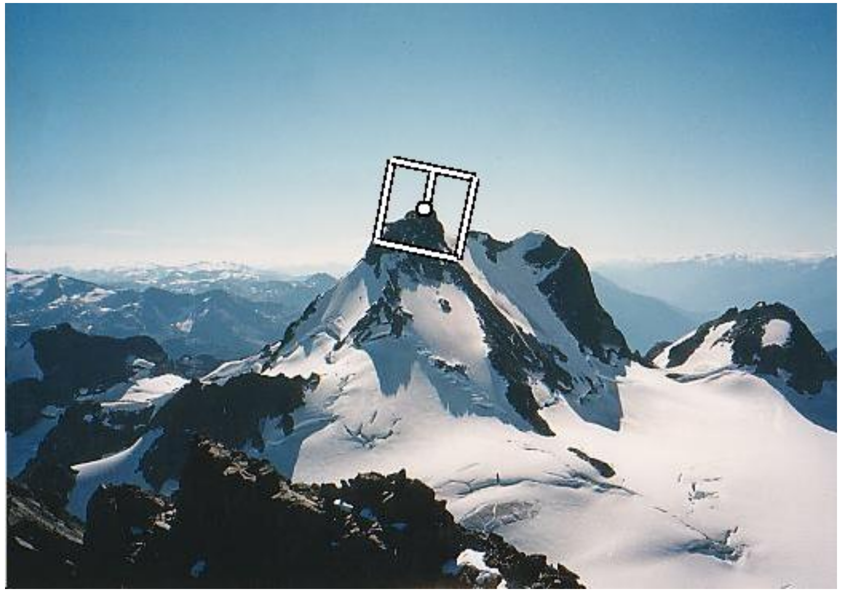
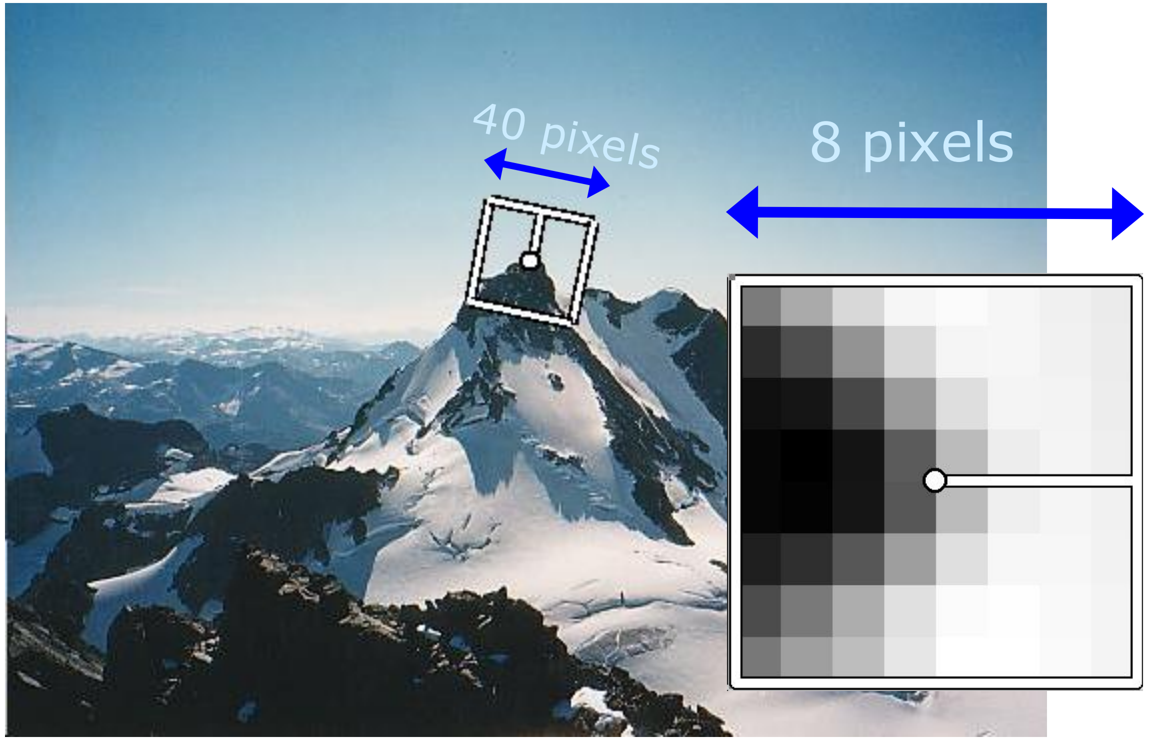


Figure by Matthew Brown

# Multiscale Oriented PatcheS descriptor

Take 40x40 square window around detected feature

- Scale to 1/5 size (using prefiltering)
- Rotate to horizontal
- Sample 8x8 square window centered at feature
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window (why?)



Adapted from slide by Matthew Brown

# Detections at multiple scales

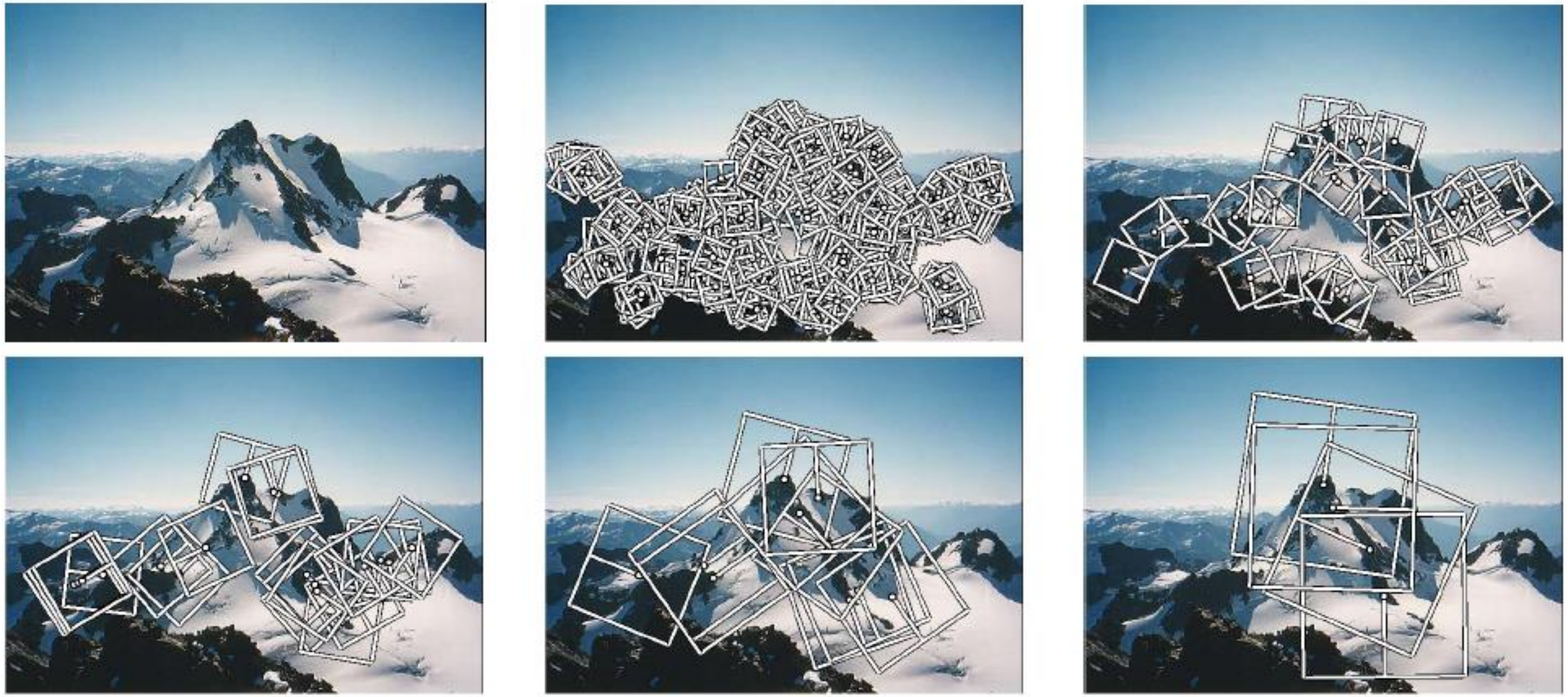


*Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.*

# Scale Invariant Feature Transform

Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations
- Shift the bins so that the biggest one is first

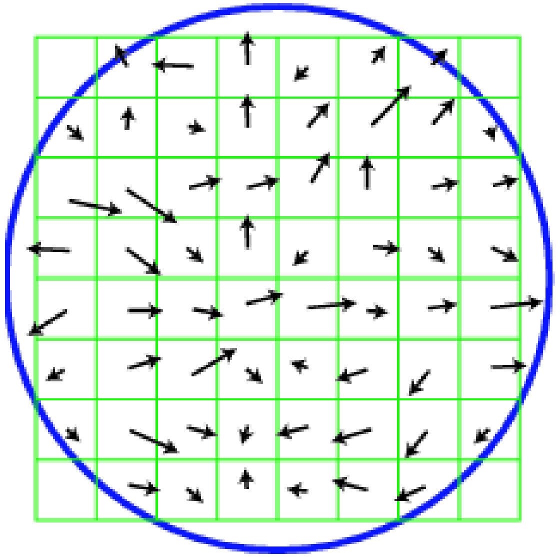
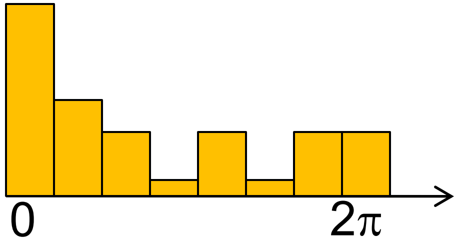
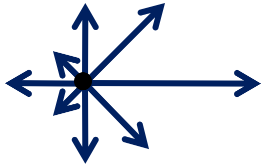


Image gradients



angle histogram



Adapted from slide by David Lowe

# SIFT descriptor

Full version

- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor

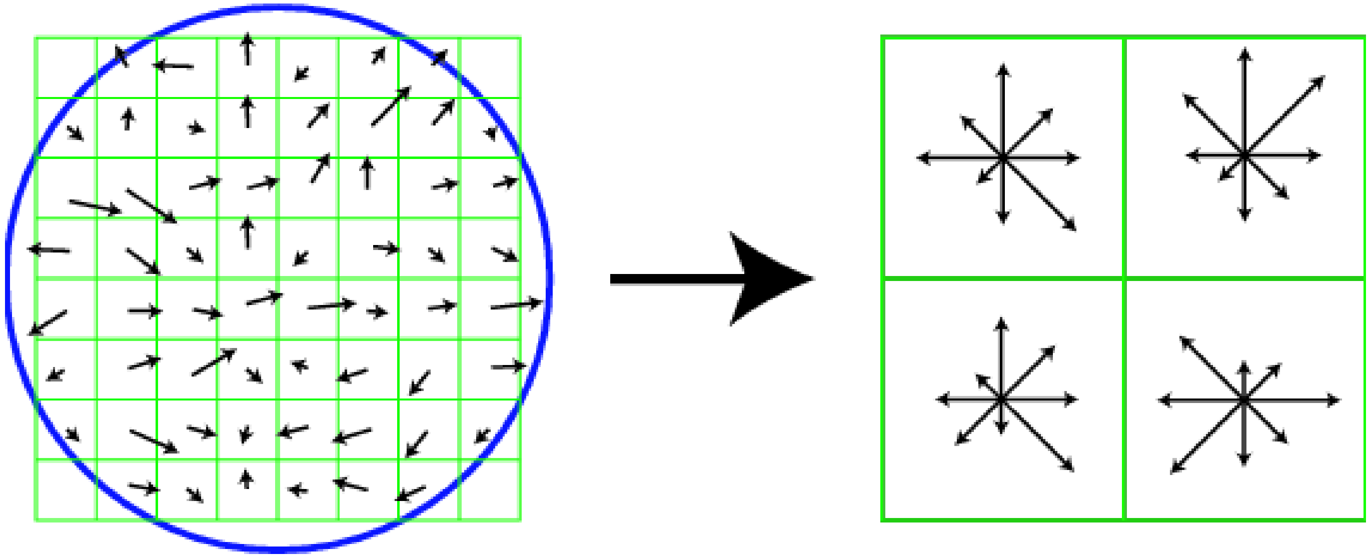


Image gradients

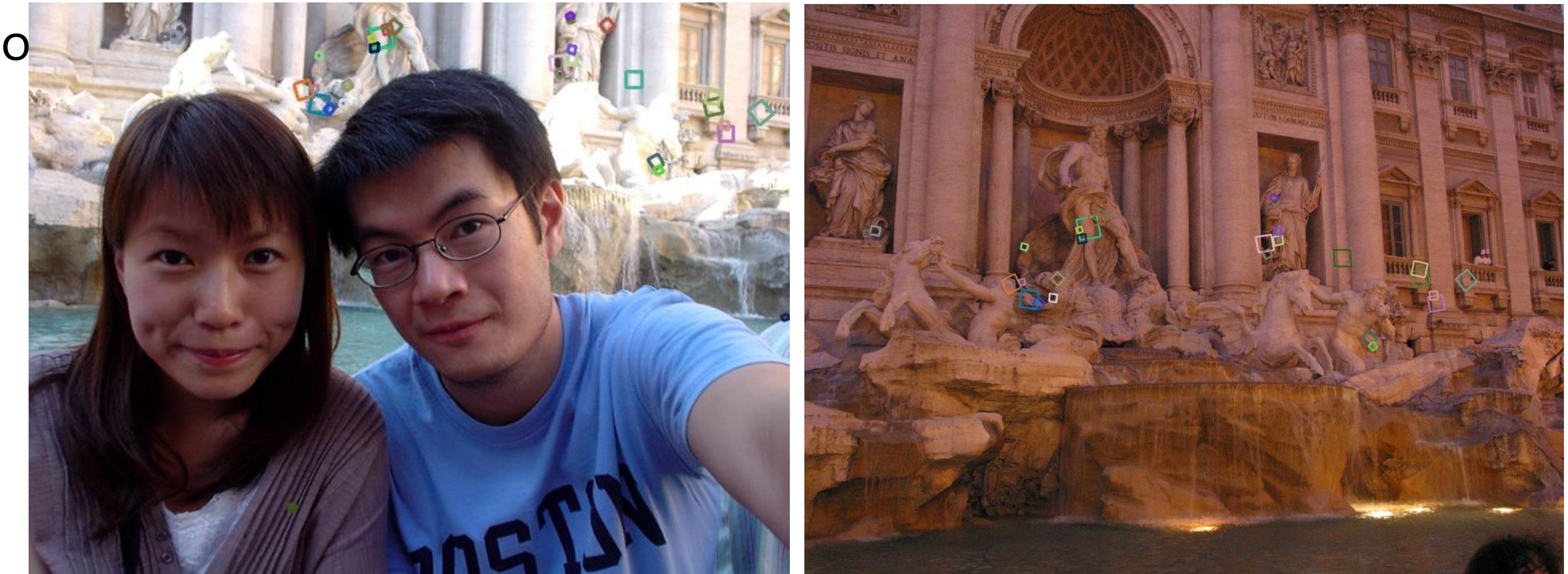
Keypoint descriptor

Adapted from slide by David Lowe

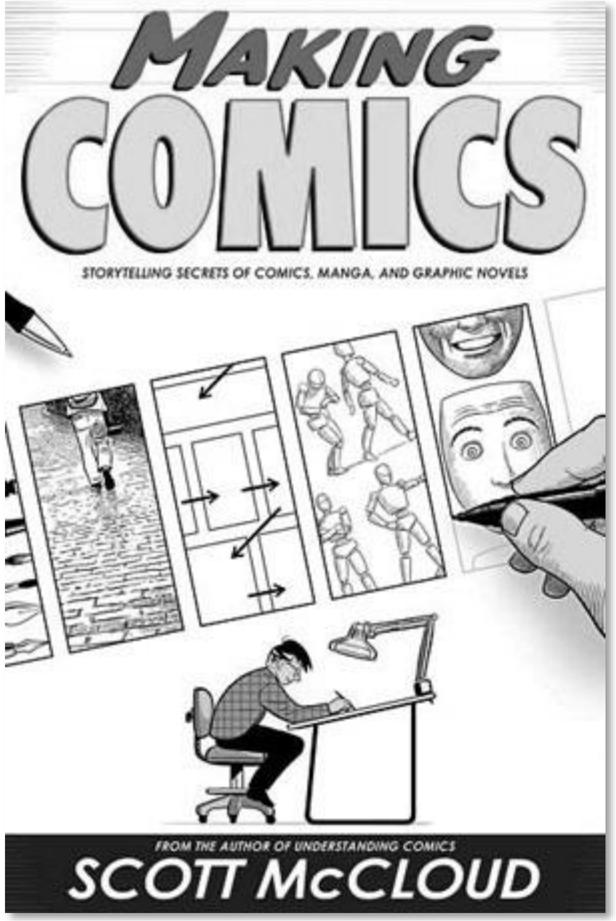
# Properties of SIFT

Extraordinarily robust matching technique

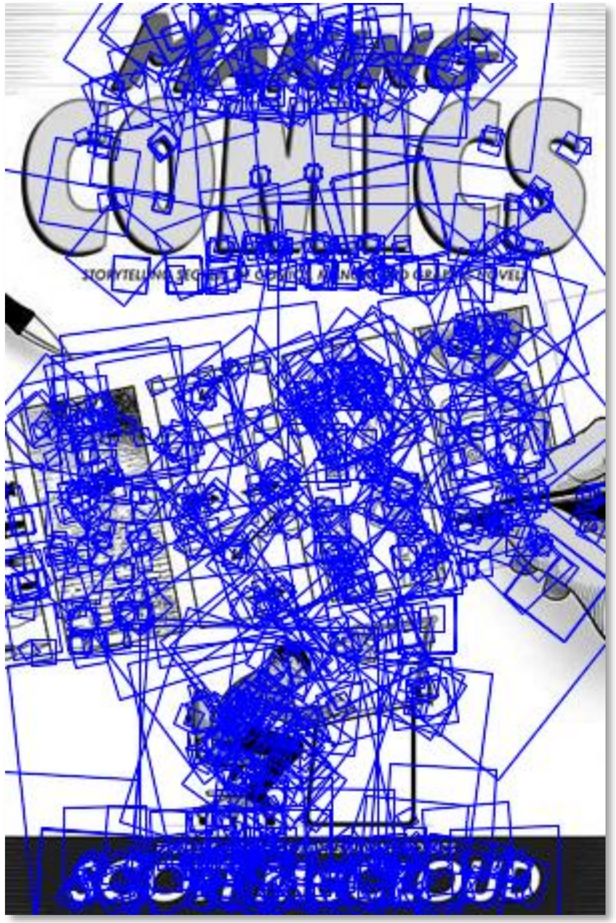
- Can handle changes in viewpoint (up to about 60 degree out of plane rotation)
- Can handle significant changes in illumination (sometimes even day vs. night (below))
- Pretty fast—hard to make real-time, but can run in <1s for moderate image sizes
- Lo



# SIFT Example

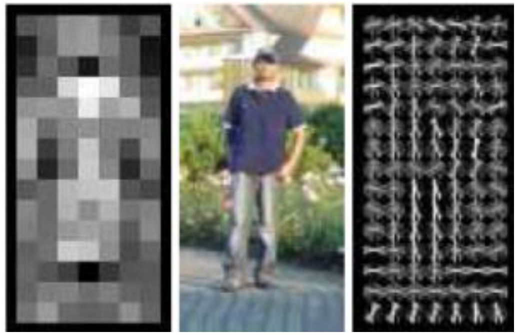

MAKING
COMICS
STORYTELLING SECRETS OF COMICS, MANGA, AND GRAPHIC NOVELS
FROM THE AUTHOR OF UNDERSTANDING COMICS
SCOTT McCLOUD

sift



**868 SIFT features**

# Other descriptors



- HOG: Histogram of Gradients (HOG)
  - – Dalal/Triggs
  - – Sliding window, pedestrian detection

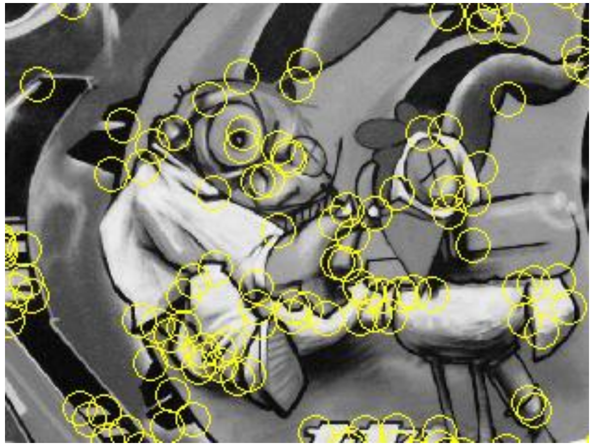
- FREAK (Fast Retina Keypoint) or ORB features
  - – Can run in real-time; used in Visual SLAM on-device

- LIFT: Learned Invariant Feature Transform
  - – Learned via deep learning – along with many other recent features

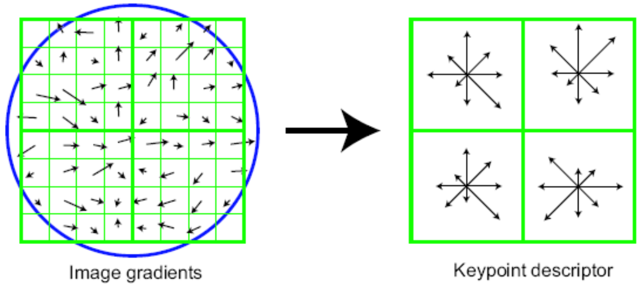
# Questions?

# Summary

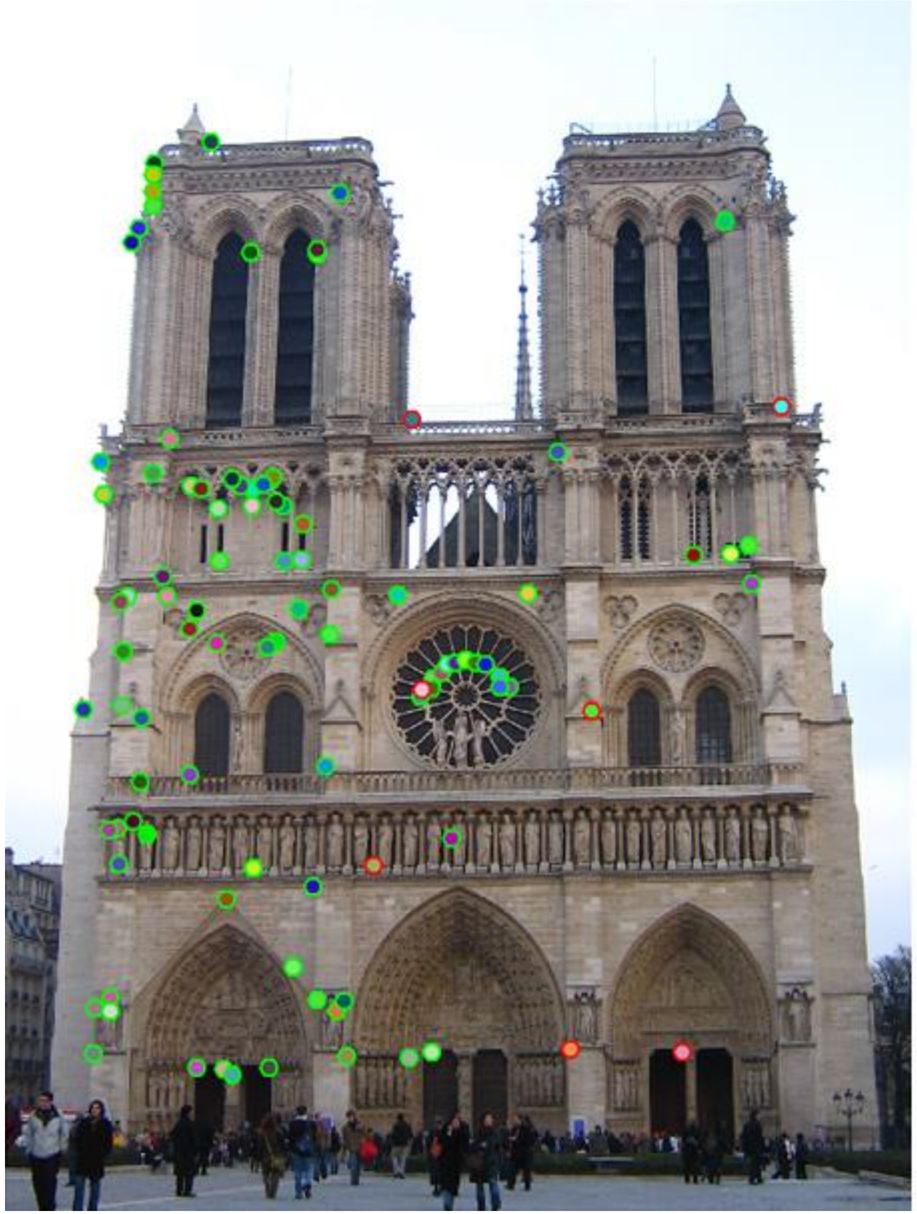
- Keypoint detection: repeatable and distinctive
  - Corners, blobs
  - Harris, DoG

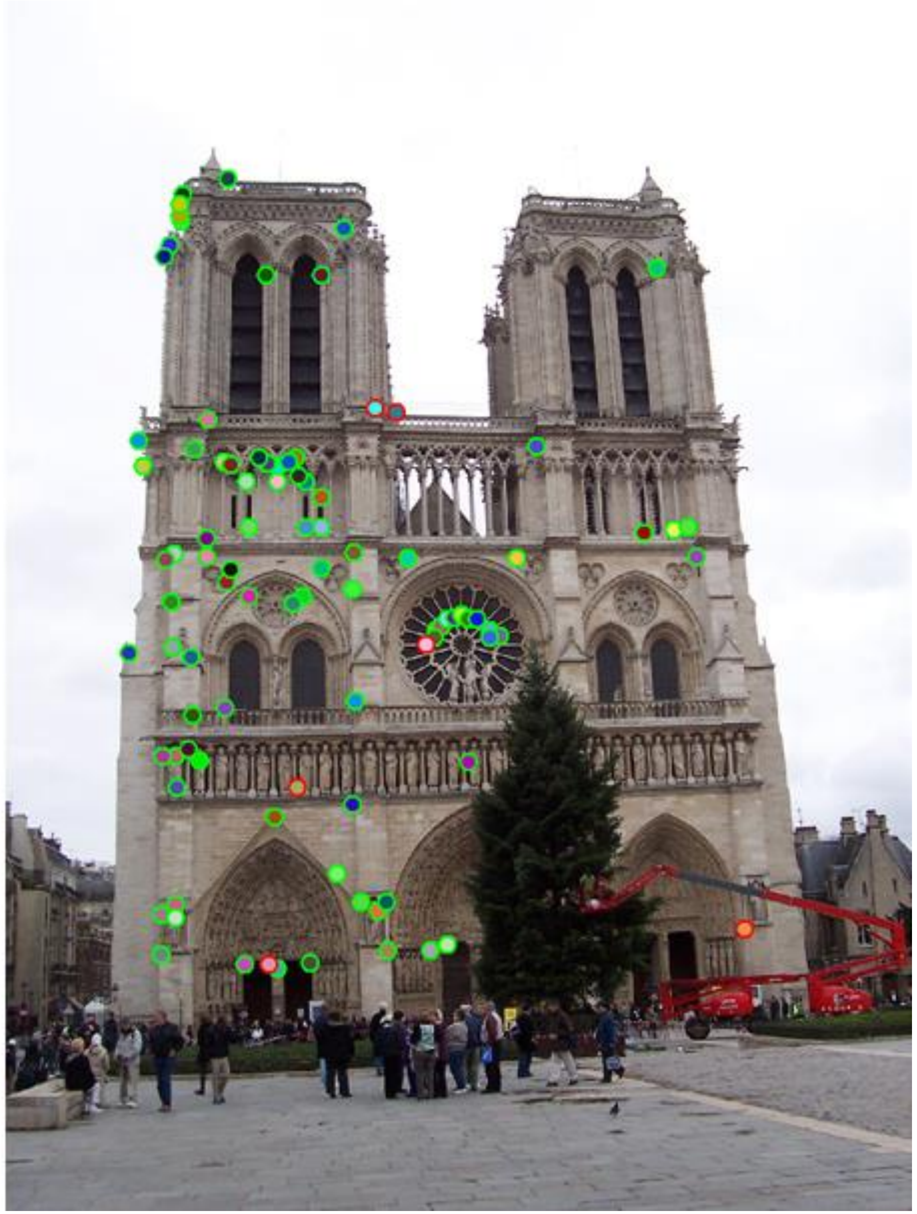


- Descriptors: robust and selective
  - spatial histograms of orientation
  - SIFT and variants are typically good for stitching and recognition
  - But, need not stick to one



# Which features match?





# Feature matching

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

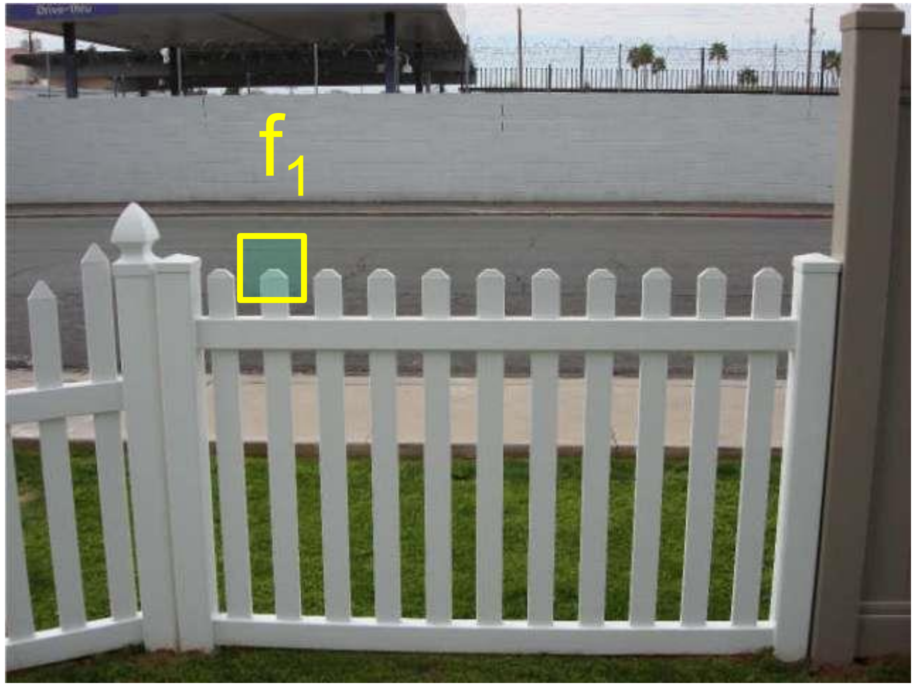
1. Define distance function that compares two descriptors
2. Test all the features in $I_2$, find the one with min distance

(can be accelerated with a nearest neighbors search data structure, like a *kd*-tree)

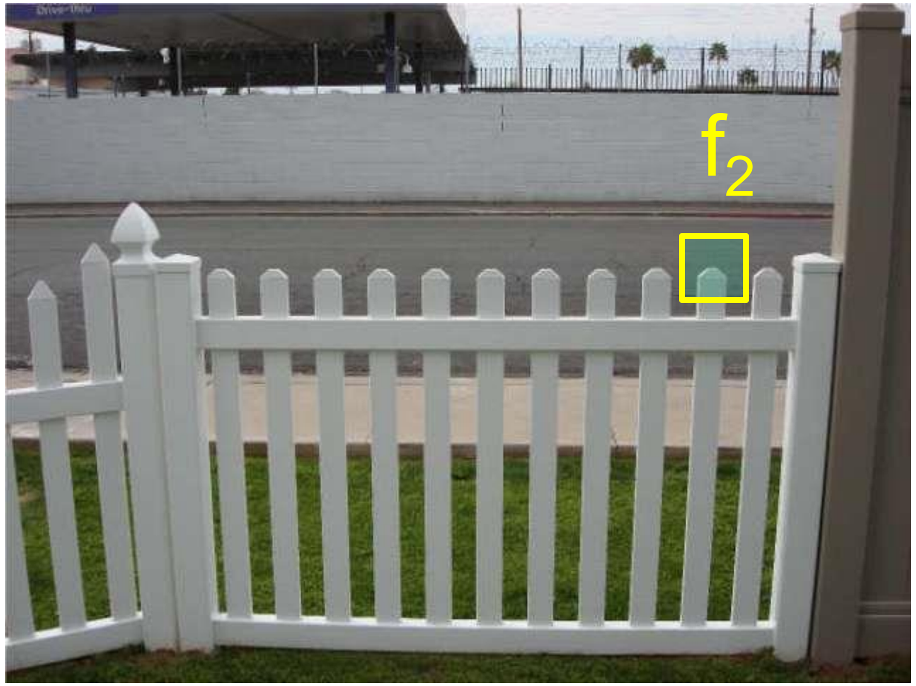
# Feature distance

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

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



$I_1$

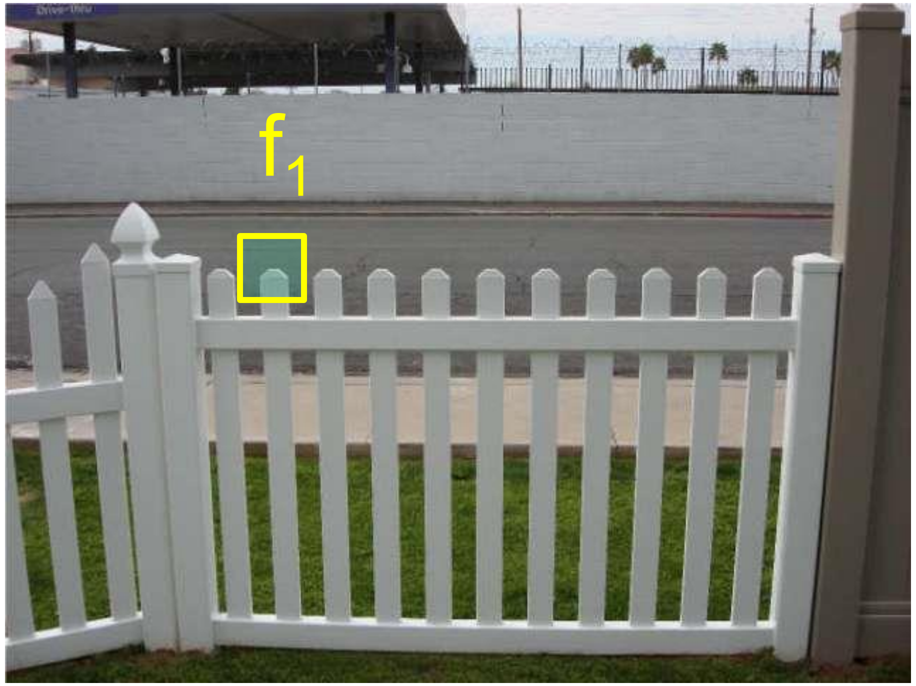


$I_2$

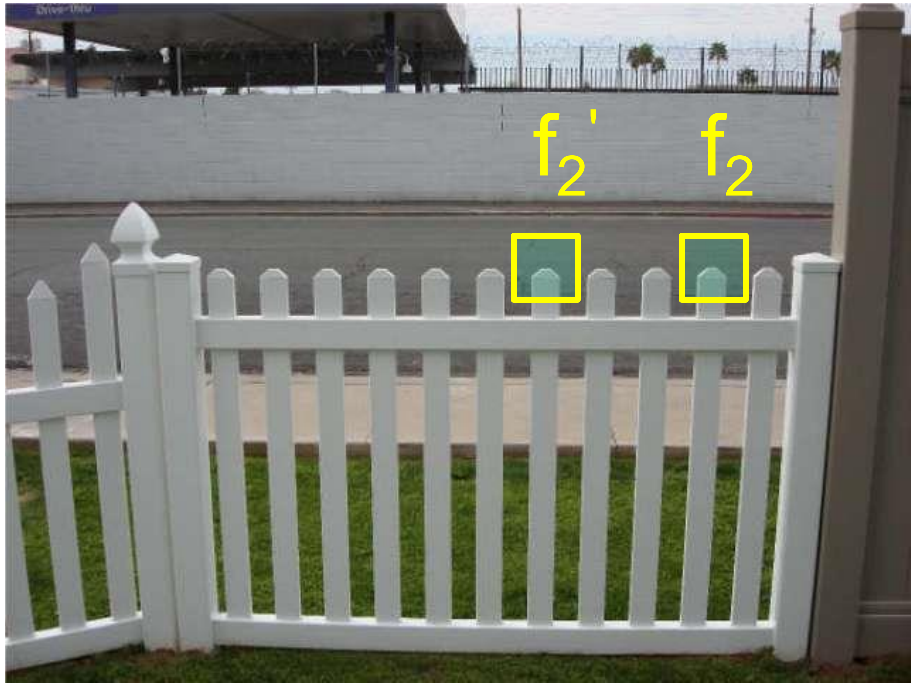
# Feature distance

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

- Better approach: ratio distance = $|| f_1 - f_2 || / || f_1 - f_2' ||$
  - $f_2$ is the best SSD match to $f_1$ in $I_2$
  - $f_2'$ is the 2nd best SSD match to $f_1$ in $I_2$
  - gives large values for ambiguous matches



$I_1$

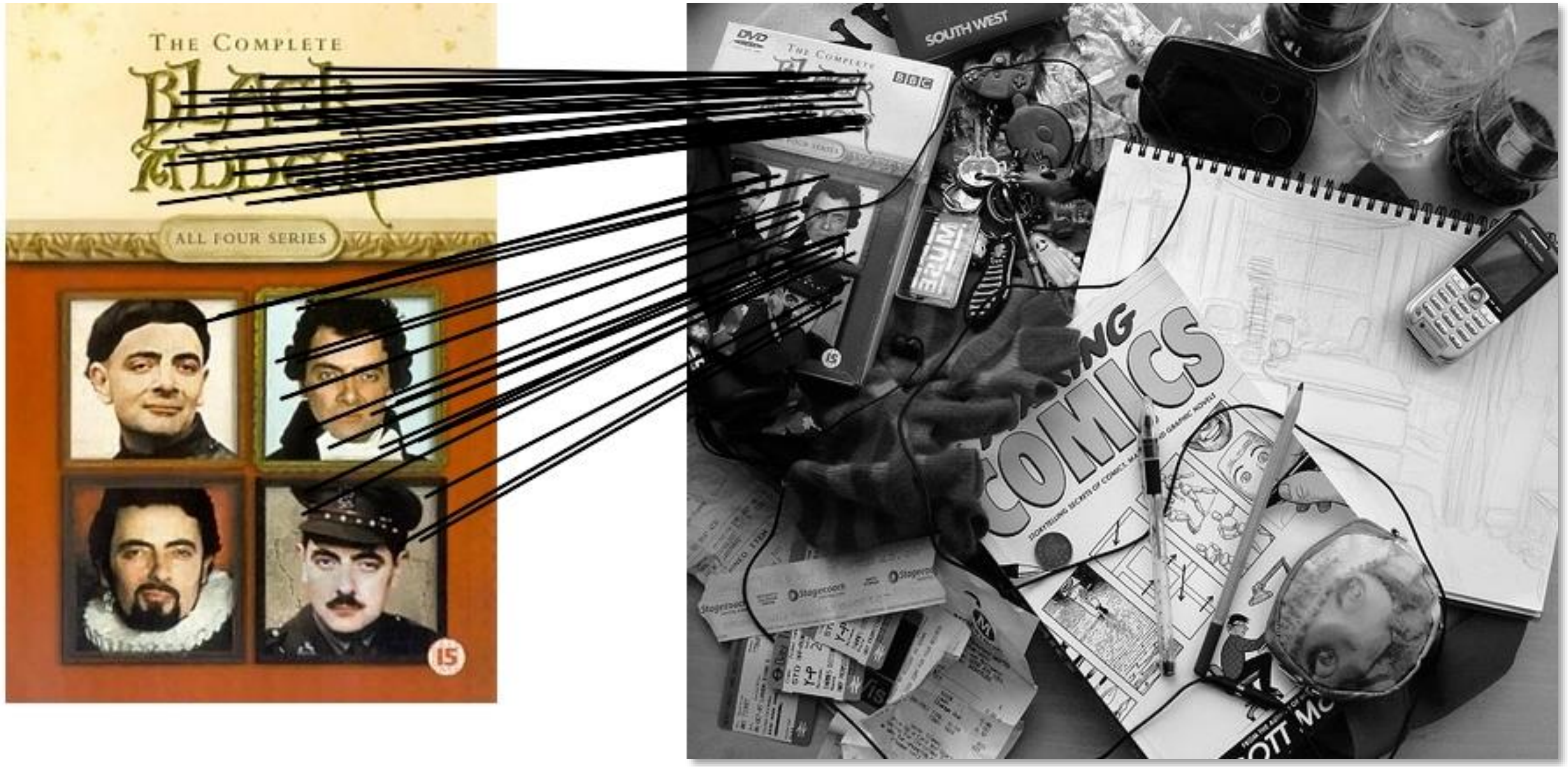


$I_2$

# Feature distance

- Does the SSD vs “ratio distance” change the best match to a given feature in image 1?
- No, but it changes the distance, and it can change the ordering of matches from good to bad
- After we compute a set of matches, we *threshold* by distance (that is, throw out matches with distance > threshold)

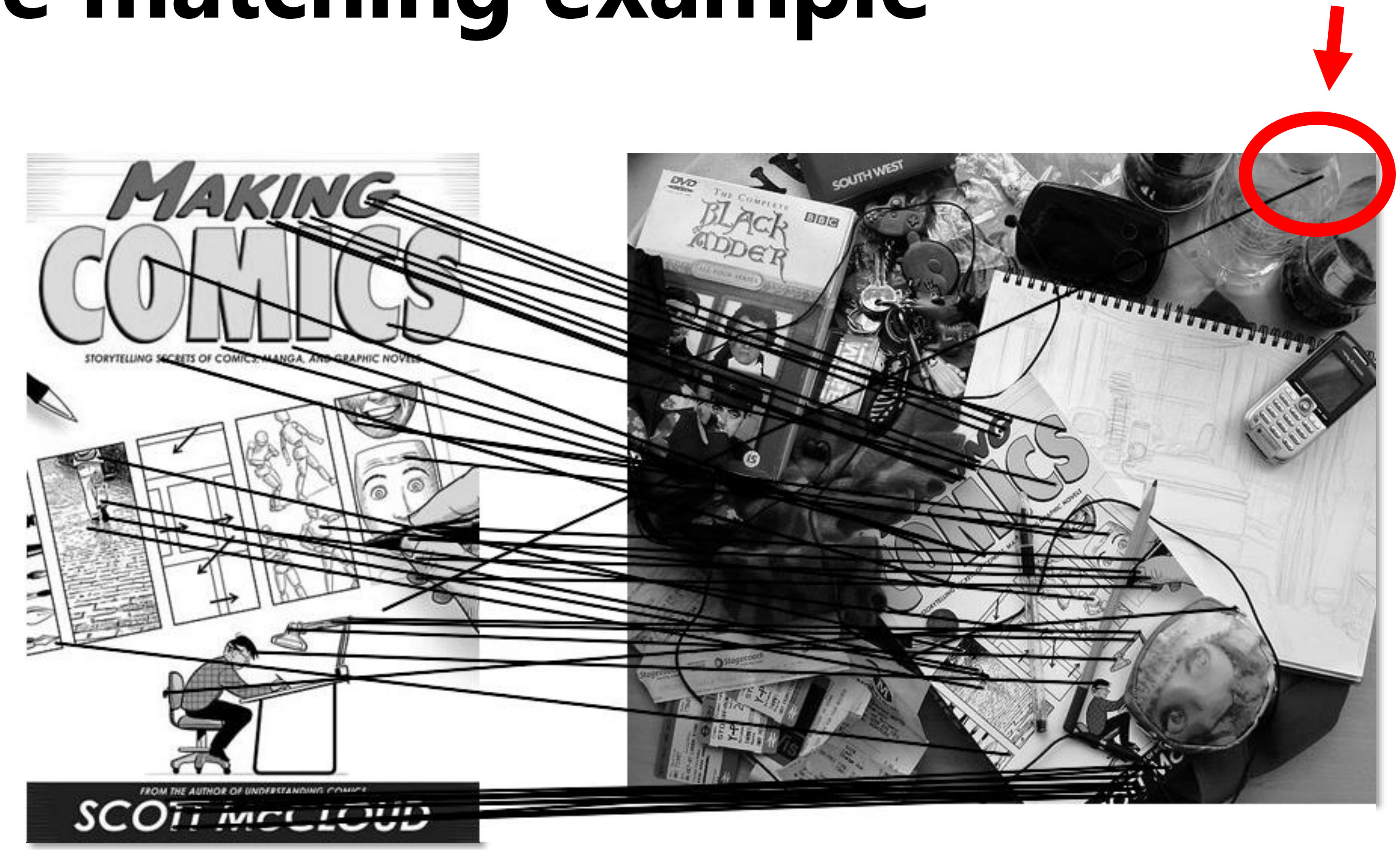
# Feature matching example



**58 matches (thresholded by ratio score)**

# Feature matching example

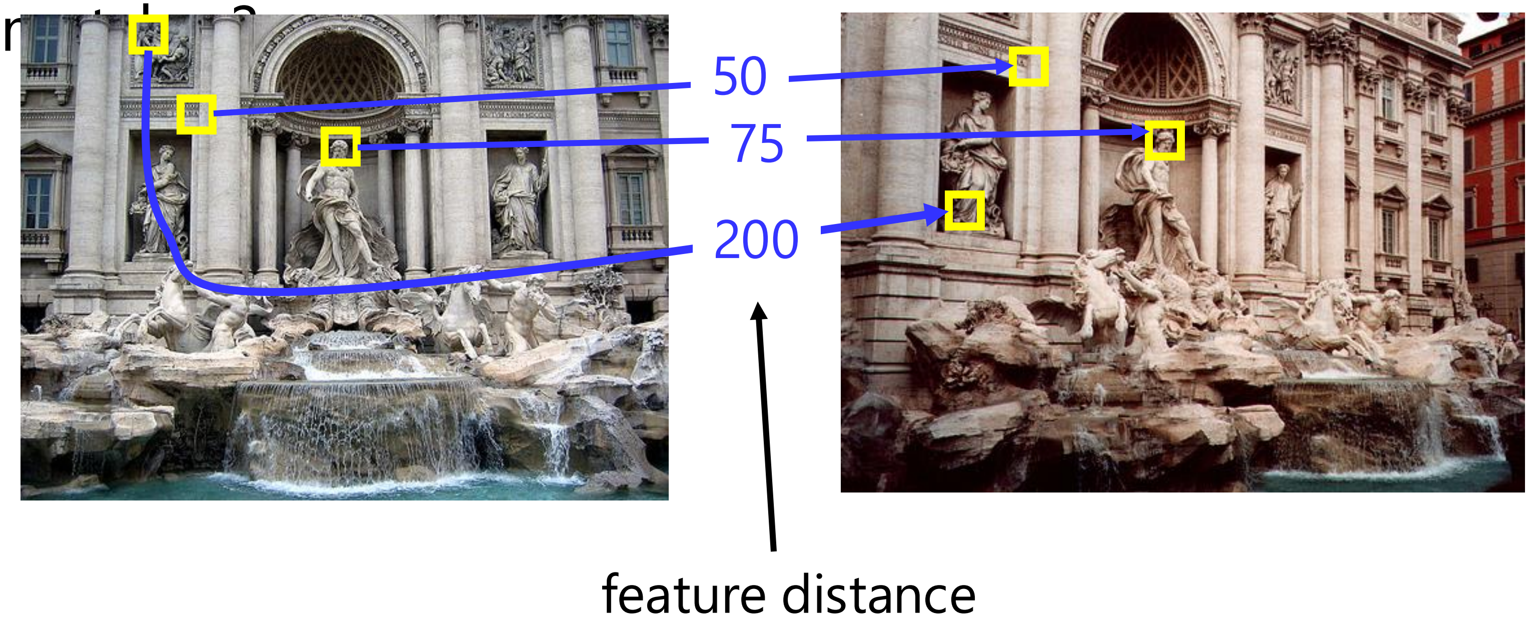
We'll deal with **outliers** later


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

**51 matches (thresholded by ratio score)**

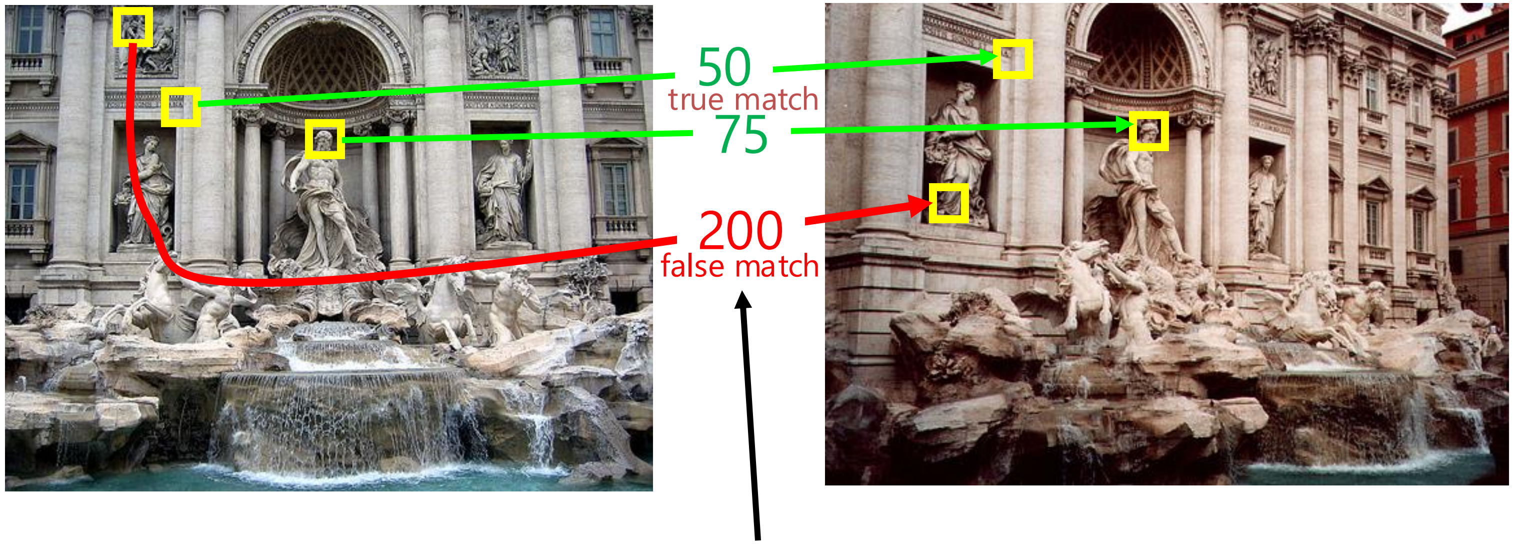
# Evaluating the results

How can we measure the performance of a feature matcher?



# True/false positives

How can we measure the performance of a feature matcher?



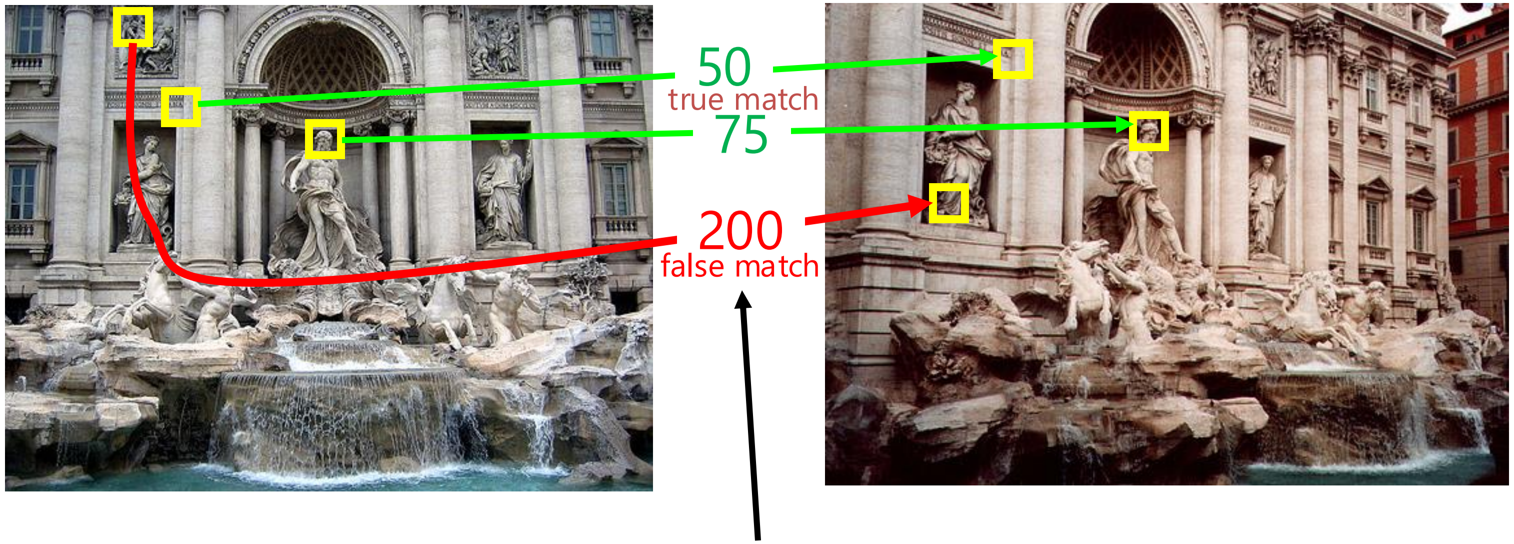
feature distance

The distance threshold affects performance

- True positives = # of detected matches that survive the threshold that are correct
- False positives = # of detected matches that survive the threshold that are incorrect

# True/false positives

How can we measure the performance of a feature matcher?

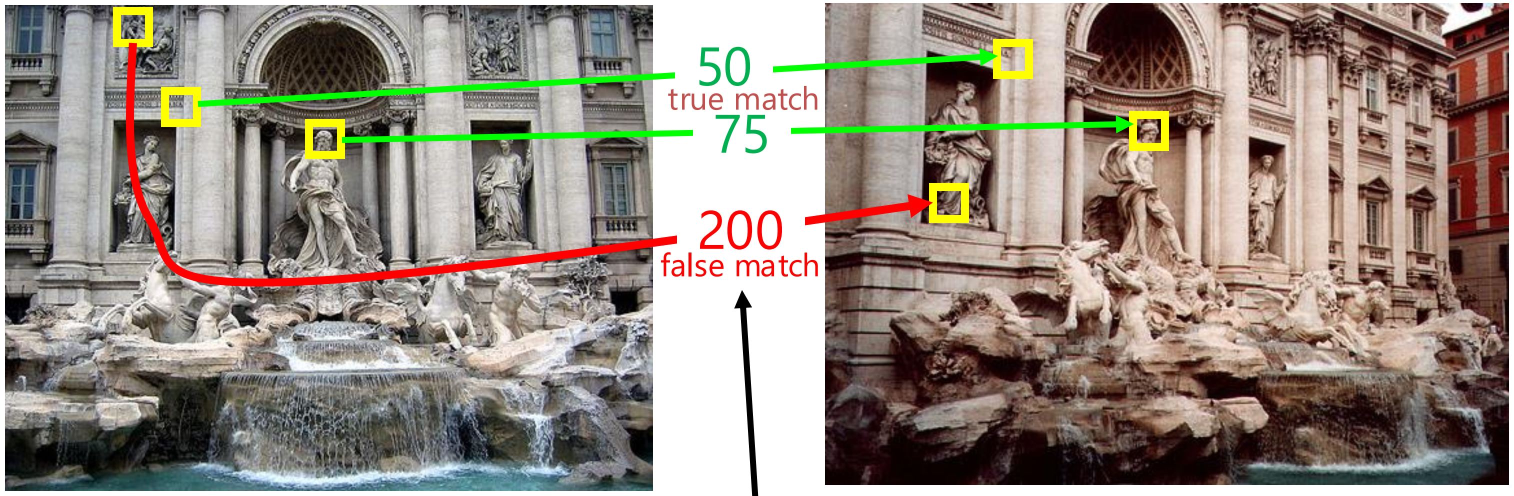


feature distance

Suppose we want to **maximize true positives**. How do we set the threshold? (Note: we keep all matches with distance below the threshold.)

# True/false positives

How can we measure the performance of a feature matcher?



feature distance

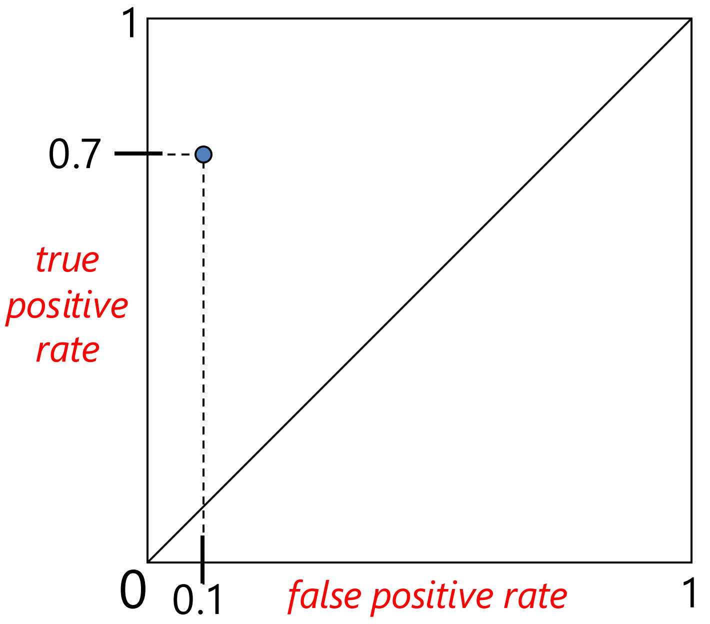
Suppose we want to **minimize false positives**. How do we set the threshold? (Note: we keep all matches with distance below the threshold.)

# Example

- Suppose our matcher computes 1,000 matches between two images
  - 800 are correct matches, 200 are incorrect (according to an oracle that gives us ground truth matches)
  - A given threshold (e.g., ratio distance = 0.6) gives us 600 correct matches and 100 incorrect matches that survive the threshold
  - True positive rate = 600 / 800 = ¾
  - False positive rate = 100 / 200 = ½

# Evaluating the results

How can we measure the performance of a feature matcher?



$$\frac{\text{\# true positives surviving threshold}}{\text{\# total correct matches (positives)}}$$

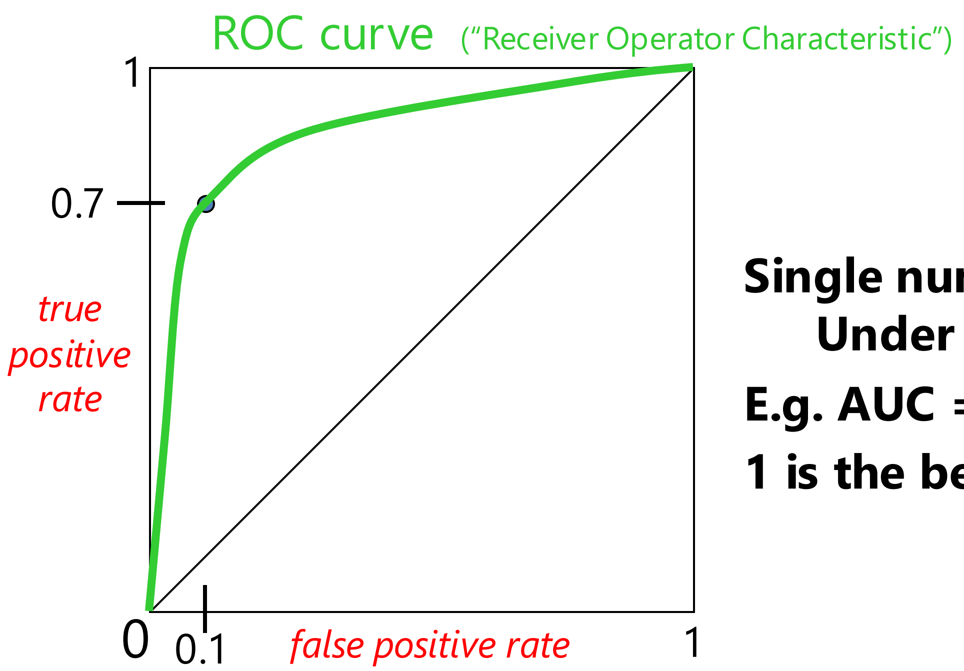
*recall*

$$\frac{\text{\# false positives surviving threshold}}{\text{\# total incorrect matches (negatives)}}$$

1 - *specificity*

# Evaluating the results

How can we measure the performance of a feature matcher?



$$\frac{\#\text{ true positives surviving threshold}}{\#\text{ total correct matches (positives)}}$$

*recall*

$$\frac{\#\text{ false positives surviving threshold}}{\#\text{ total incorrect matches (negatives)}}$$

1 - *specificity*

**Single number: Area Under the Curve (AUC)**

**E.g. AUC = 0.87**

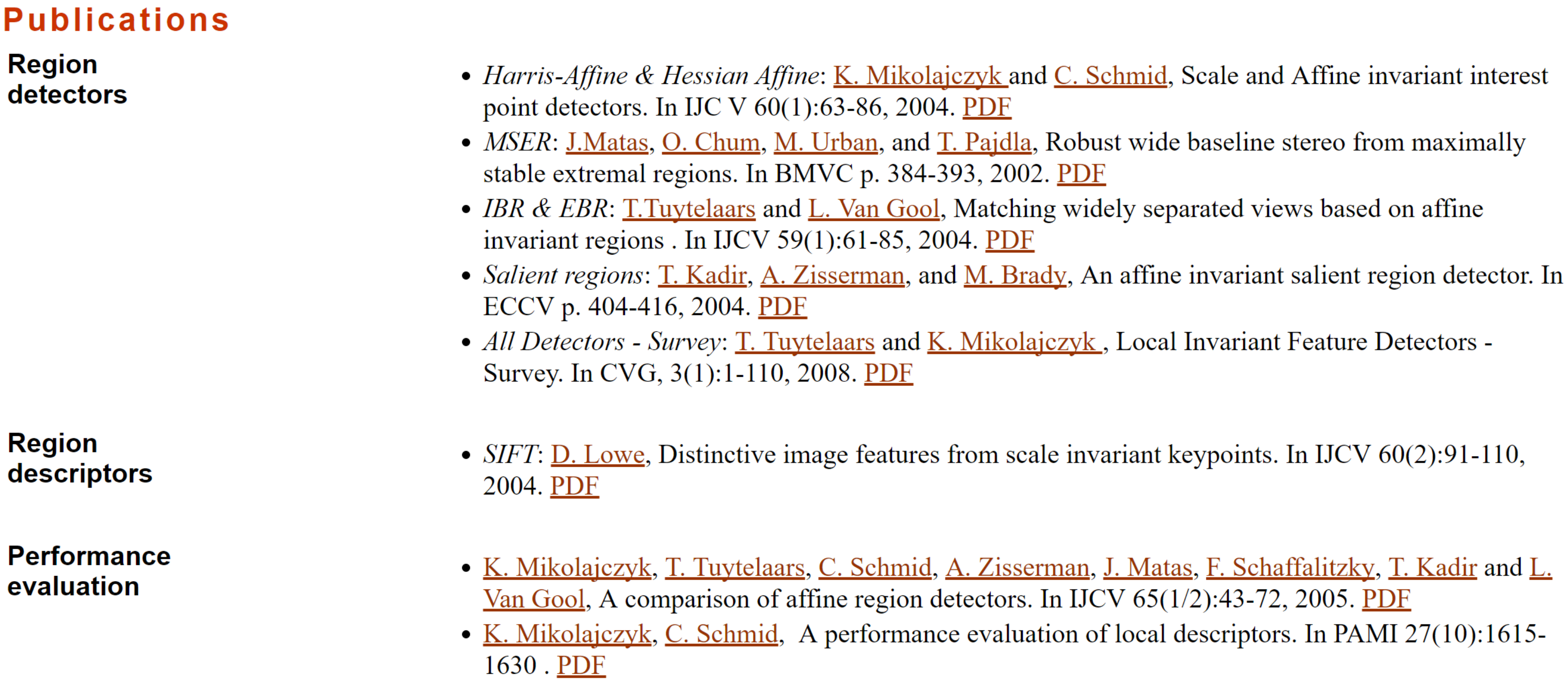
**1 is the best**

# ROC curves – summary

- By thresholding the match distances at different thresholds, we can generate sets of matches with different true/false positive rates
- ROC curve is generated by computing rates at a set of threshold values swept through the full range of possible thresholds
- Area under the ROC curve (AUC) summarizes the performance of a feature pipeline (higher AUC is better)

# More on feature detection/description

http://www.robots.ox.ac.uk/~vgg/research/affine/
http://www.cs.ubc.ca/~lowe/keypoints/
http://www.vision.ee.ethz.ch/~surf/



## Publications

**Region detectors**

- *Harris-Affine & Hessian Affine*: K. Mikolajczyk and C. Schmid, Scale and Affine invariant interest point detectors. In IJC V 60(1):63-86, 2004. PDF
- *MSER*: J.Matas, O. Chum, M. Urban, and T. Pajdla, Robust wide baseline stereo from maximally stable extremal regions. In BMVC p. 384-393, 2002. PDF
- *IBR & EBR*: T.Tuytelaars and L. Van Gool, Matching widely separated views based on affine invariant regions . In IJCV 59(1):61-85, 2004. PDF
- *Salient regions*: T. Kadir, A. Zisserman, and M. Brady, An affine invariant salient region detector. In ECCV p. 404-416, 2004. PDF
- *All Detectors - Survey*: T. Tuytelaars and K. Mikolajczyk , Local Invariant Feature Detectors - Survey. In CVG, 3(1):1-110, 2008. PDF

**Region descriptors**

- *SIFT*: D. Lowe, Distinctive image features from scale invariant keypoints. In IJCV 60(2):91-110, 2004. PDF

**Performance evaluation**

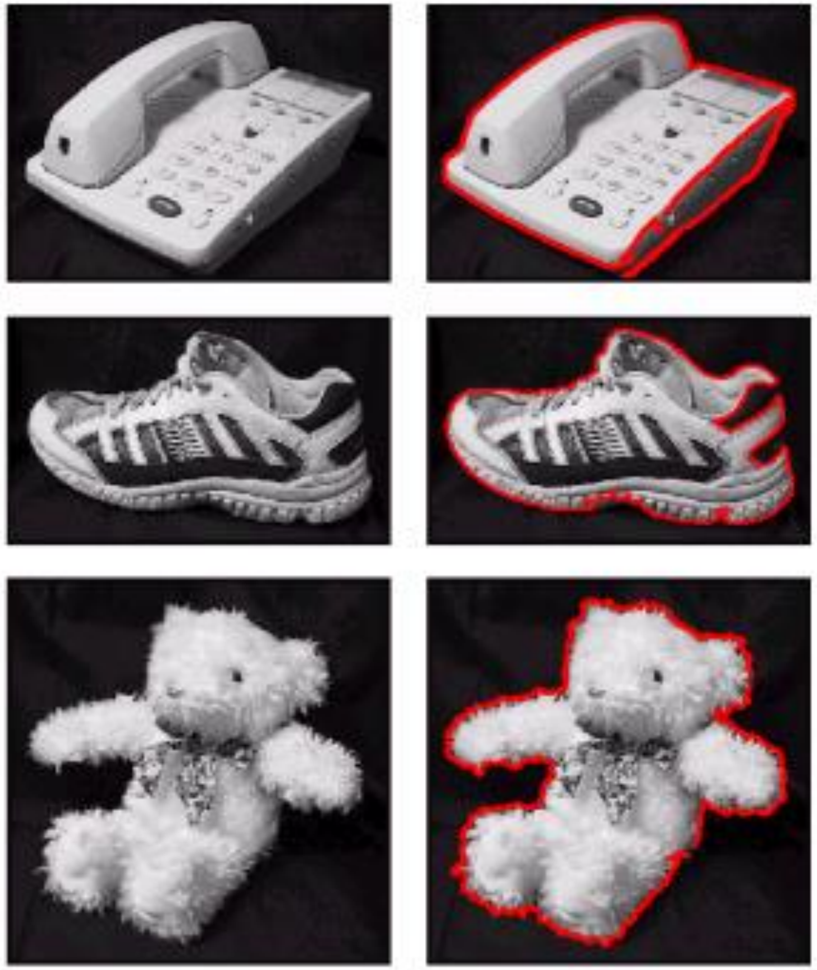
- K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir and L. Van Gool, A comparison of affine region detectors. In IJCV 65(1/2):43-72, 2005. PDF
- K. Mikolajczyk, C. Schmid, A performance evaluation of local descriptors. In PAMI 27(10):1615-1630 . PDF

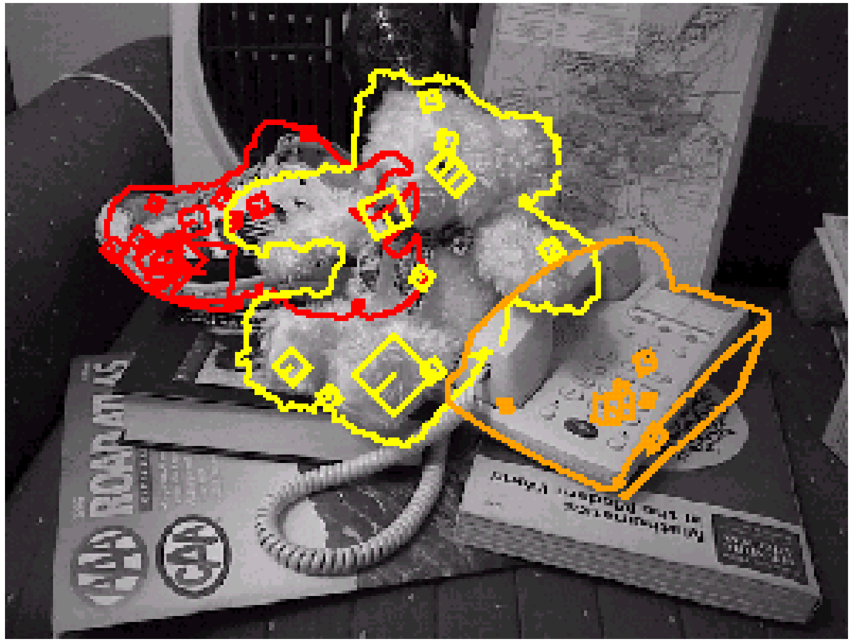
# Lots of applications

Features are used for:

- Image alignment (e.g., mosaics)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- … other

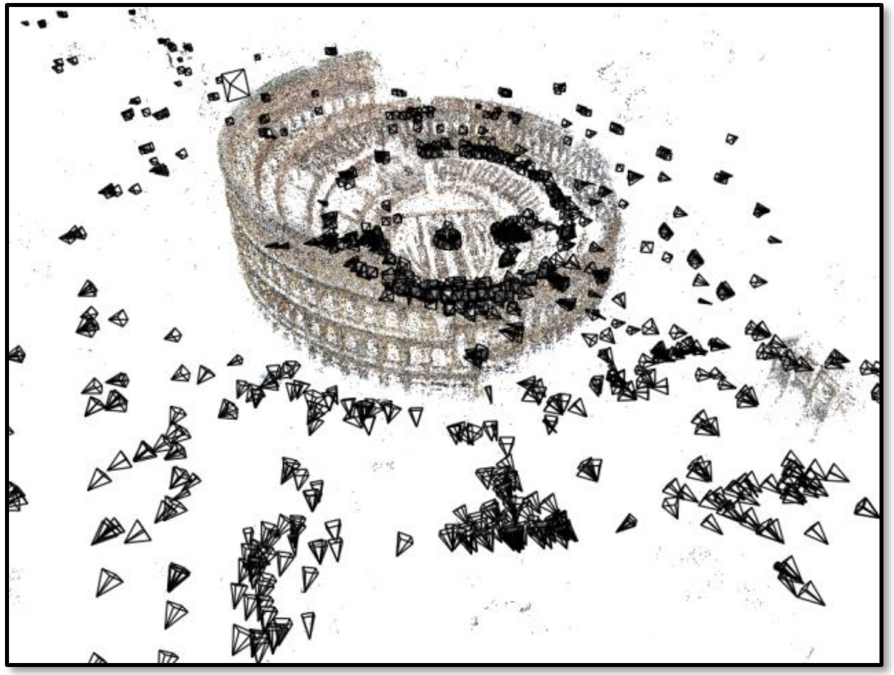
# Object recognition (David Lowe)





# 3D Reconstruction

Internet Photos (“Colosseum”)

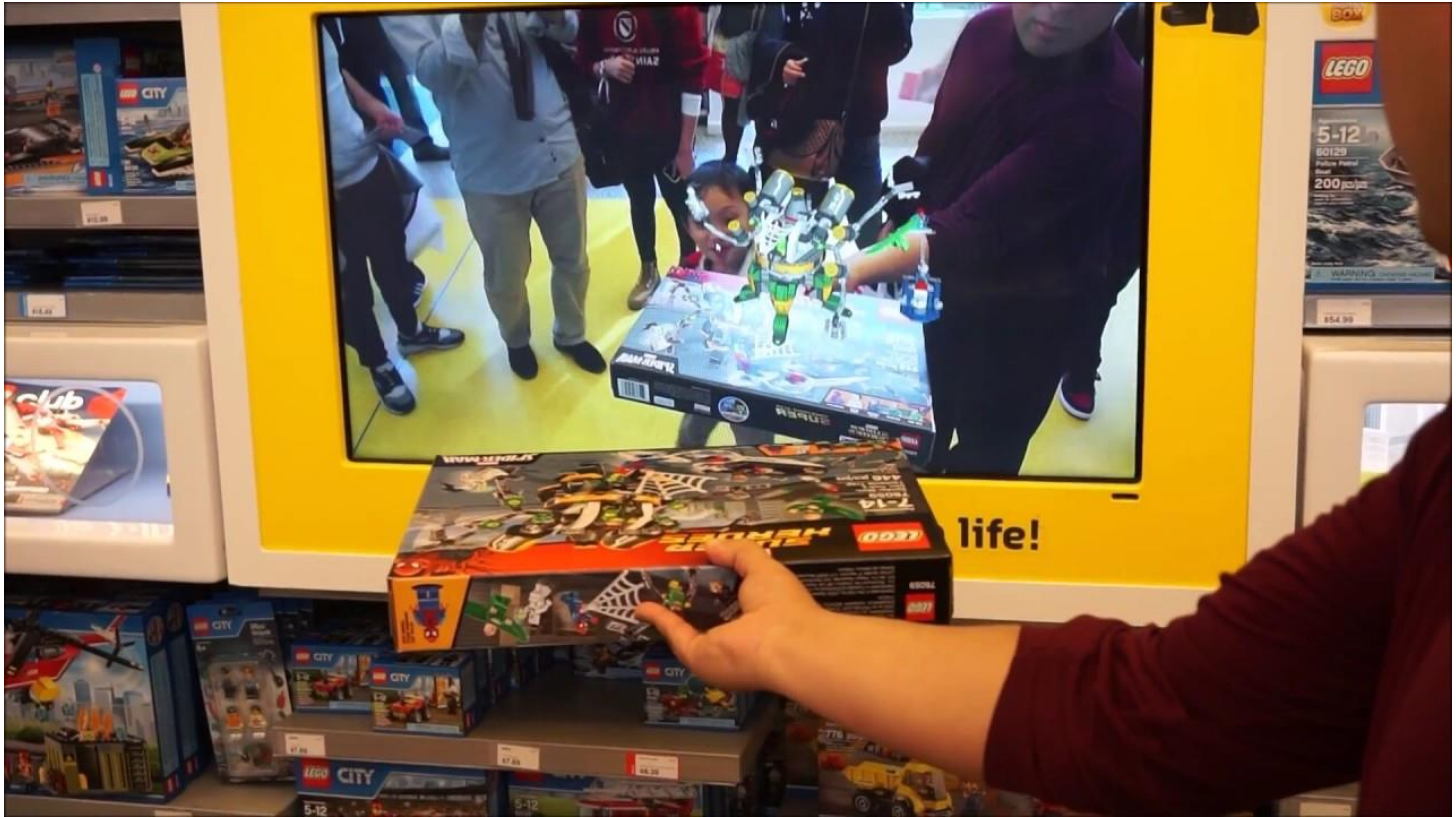


Reconstructed 3D cameras and points

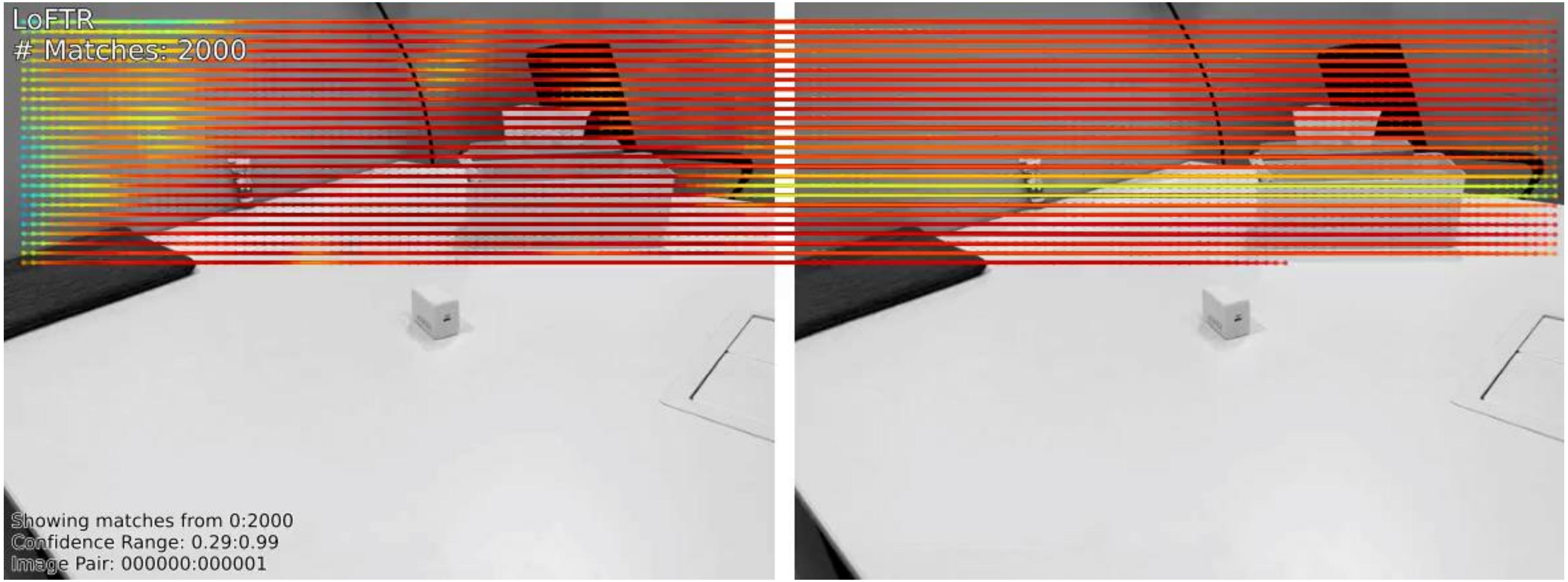
# WildGaussians: first reconstruct cameras using feature matching, then create dense model

Kulhanek, *et al*. WildGaussians. NeurIPS 2024

# Augmented Reality

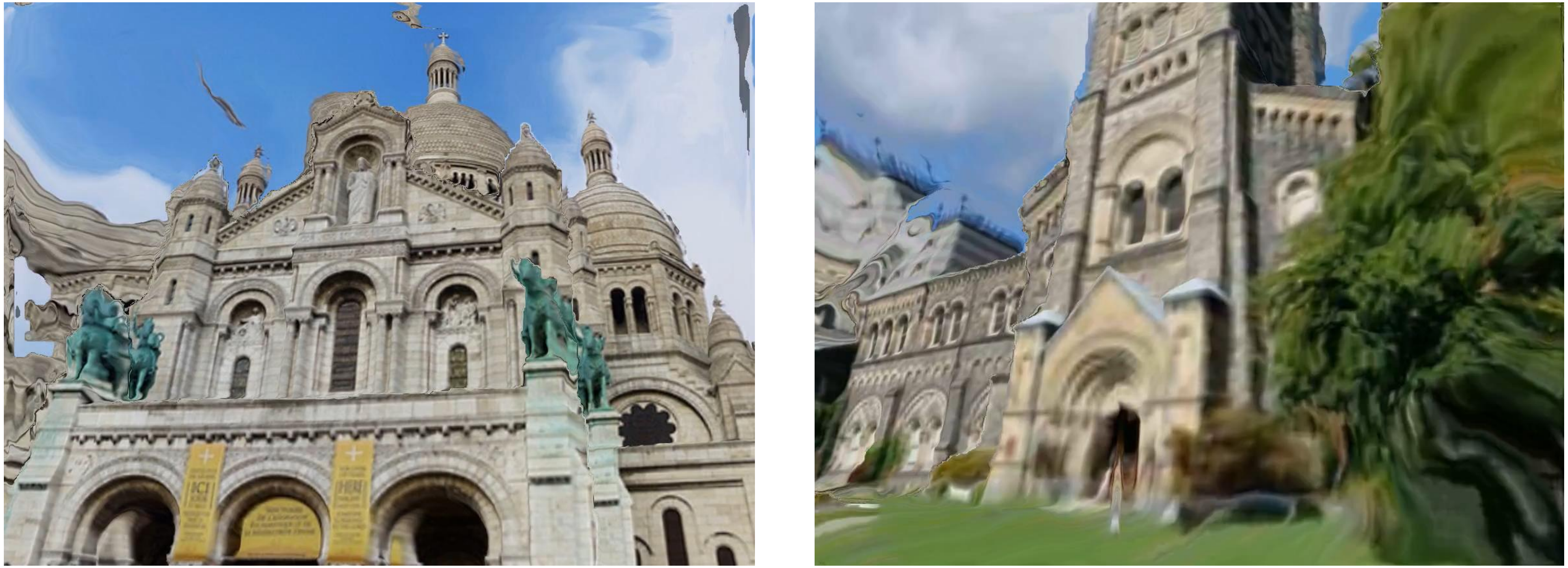


# Learned feature matchers


LoFTR
# Matches: 2000
Showing matches from 0:2000
Confidence Range: 0.29:0.99
Image Pair: 000000:000001

LoFTR: Detector-Free Local Feature Matching with Transformers
CVPR 2021 [Sun, Shen, Wang, *et al.*]

# Learned feature matchers



RoMa: Robust Dense Feature Matching, CVPR 2024

# Questions?