

# Feature Descriptor

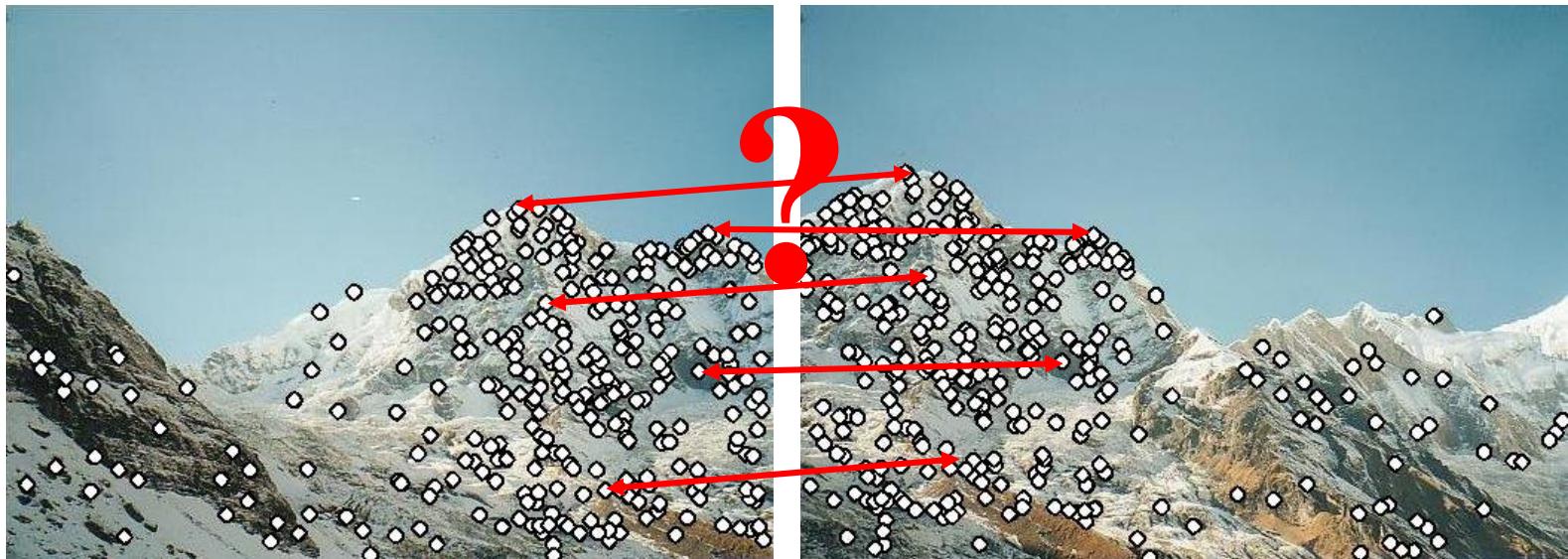
# Main questions

- Where will the interest points come from?
  - What are salient features that we'll *detect* in multiple views?
- How to *describe* a local region?
- How to establish *correspondences*, i.e., compute matches?

# Local descriptors

- We know how to detect points
- Next question:

**How to *describe* them for matching?**

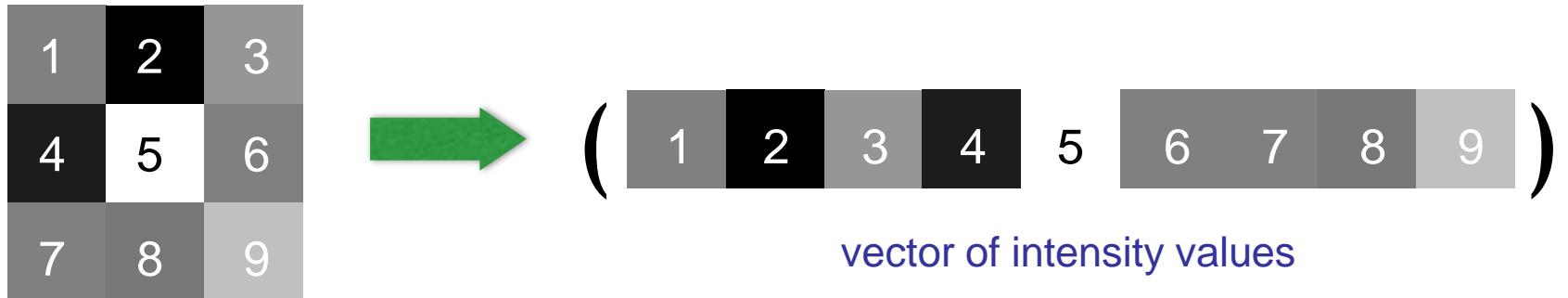


Point descriptor should be:

1. Invariant
2. Distinctive

# Image patch

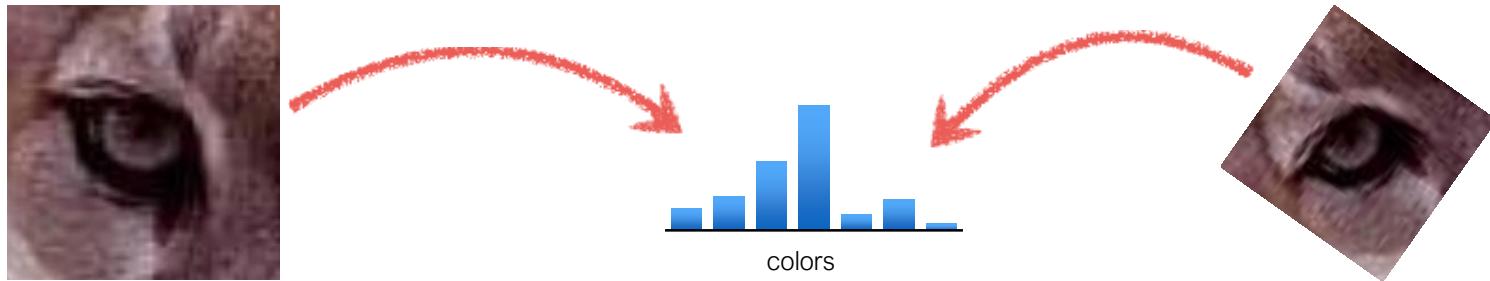
Just use the pixel values of the patch



Perfectly fine if geometry and appearance is unchanged (a.k.a. template matching)

# Color histogram

Count the colors in the image using a histogram

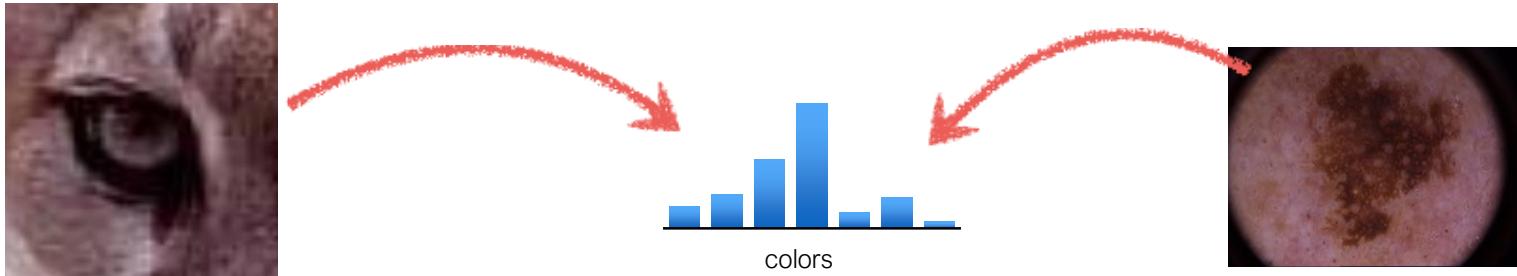


Invariant to changes in scale and rotation

*What are the problems?*

# Color histogram

Count the colors in the image using a histogram

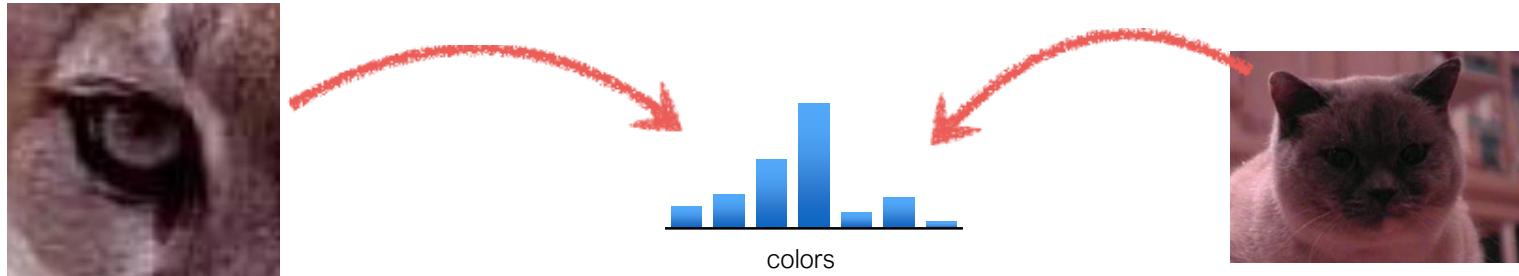


Invariant to changes in scale and rotation

*What are the problems?*

# Color histogram

Count the colors in the image using a histogram



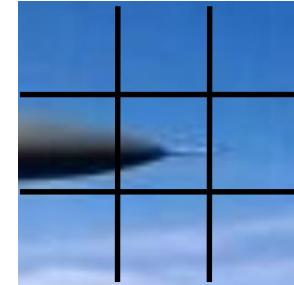
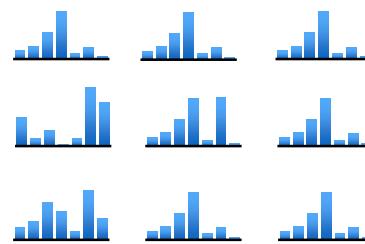
Invariant to changes in scale and rotation

*What are the problems?*

*How can you be more sensitive to spatial layout?*

# Spatial histograms

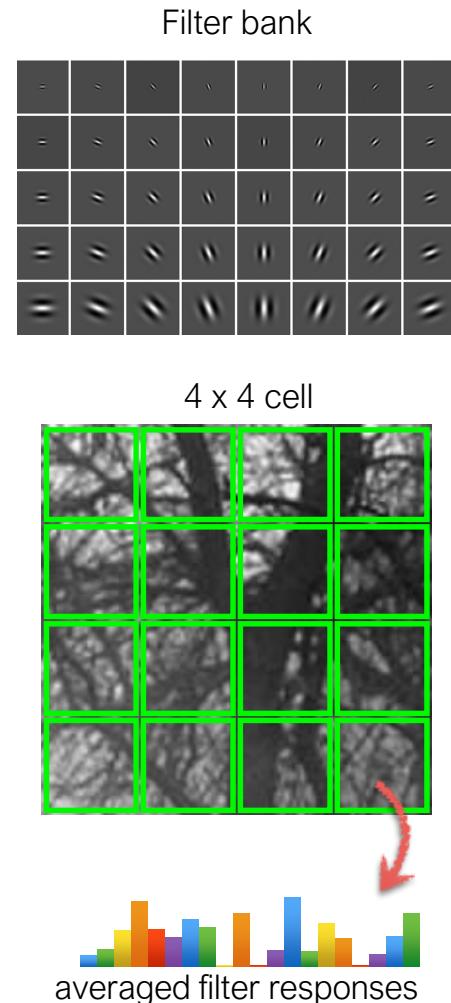
Compute histograms over spatial ‘cells’



Retains rough spatial layout  
Some invariance to deformations

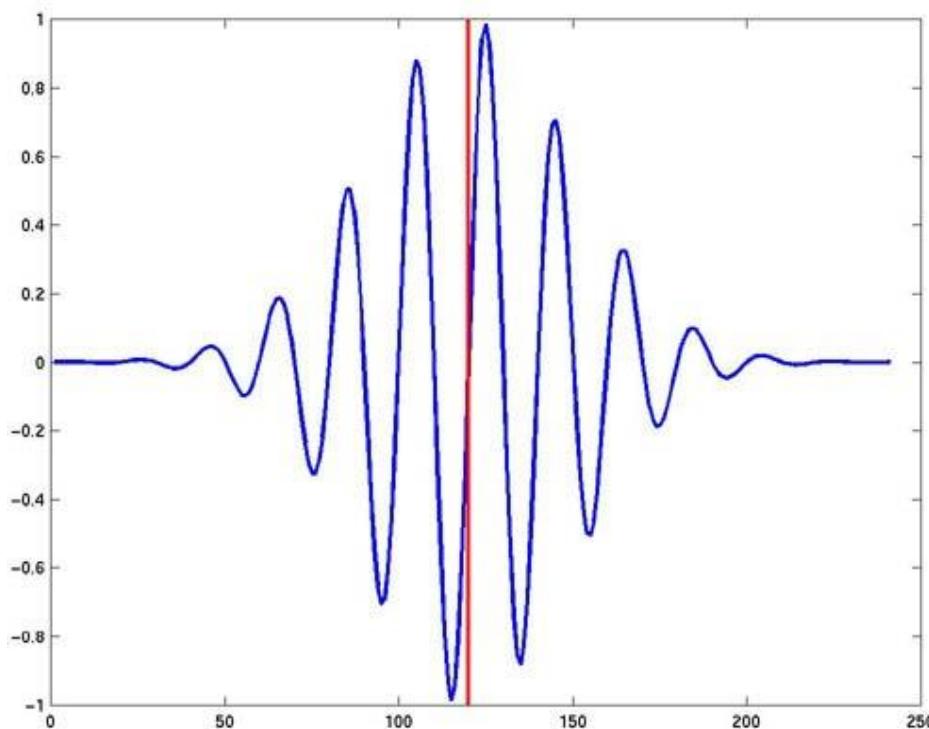
# GIST

1. Compute filter responses (filter bank of Gabor filters)
2. Divide image patch into  $4 \times 4$  cells
3. Compute filter response averages for each cell
4. Size of descriptor is  $4 \times 4 \times N$ , where  $N$  is the size of the filter bank

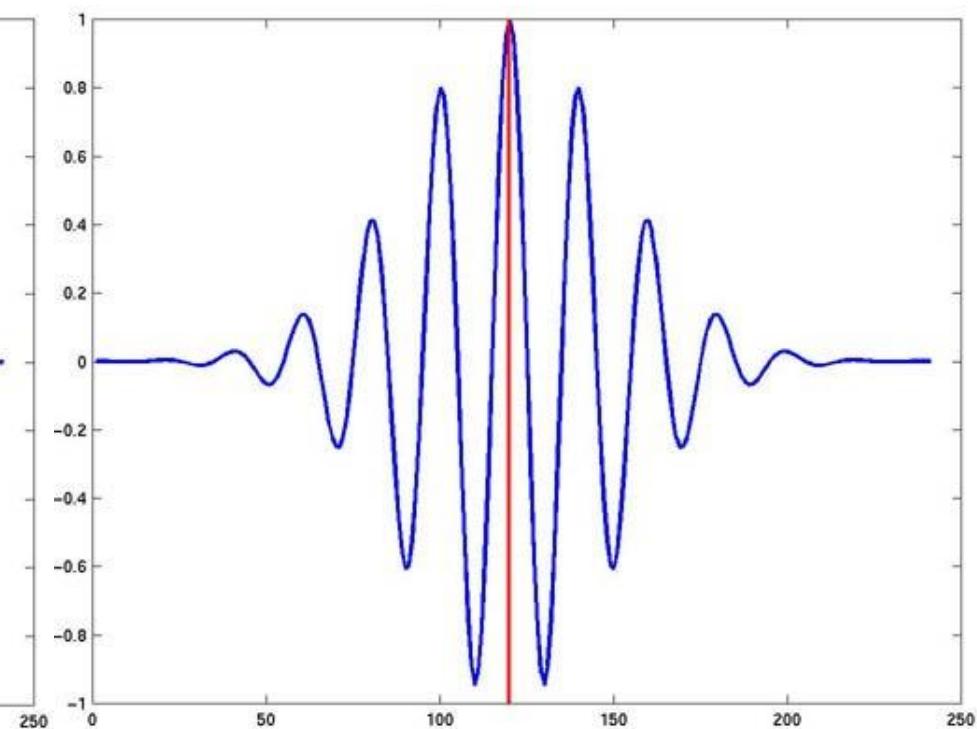


# Gabor Filters

(1D examples)



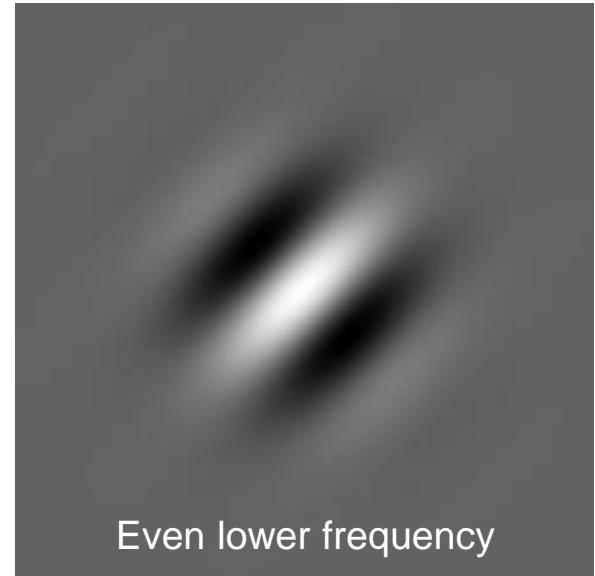
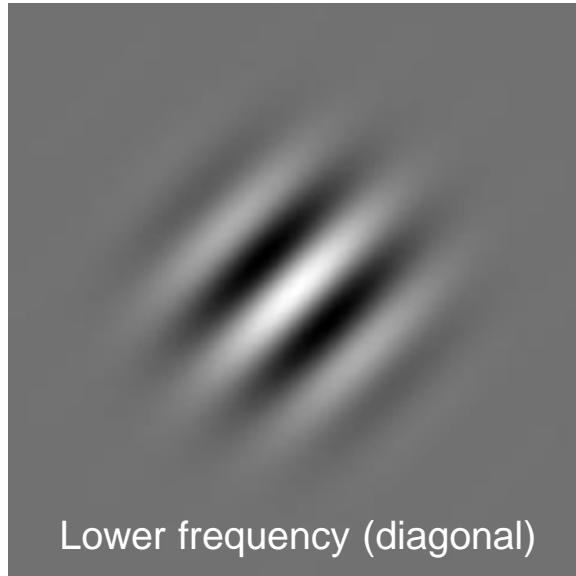
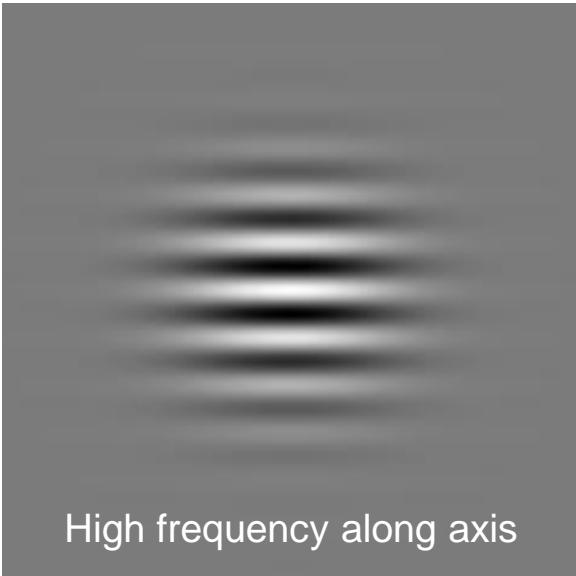
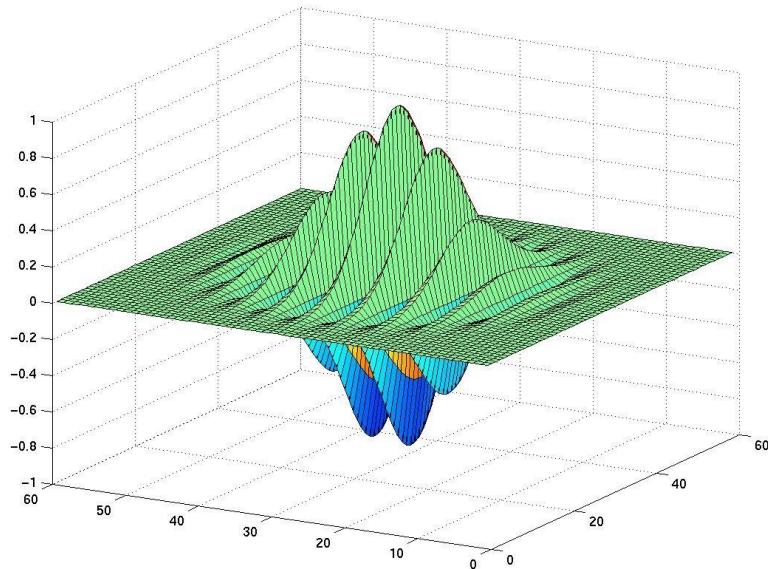
$$e^{-\frac{x^2}{2\sigma^2}} \sin(2\pi\omega x)$$

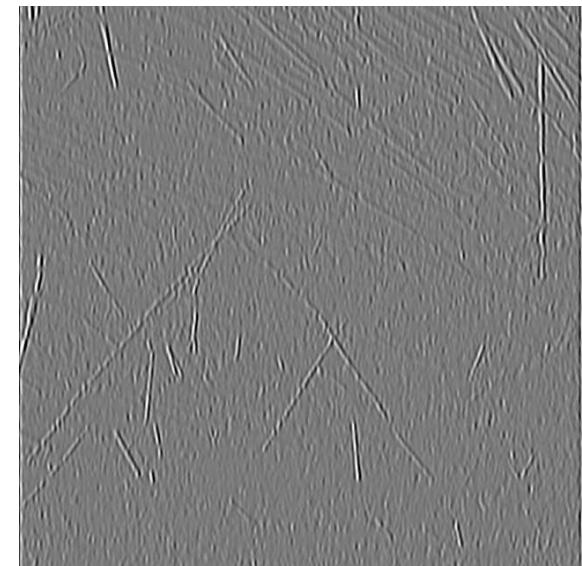
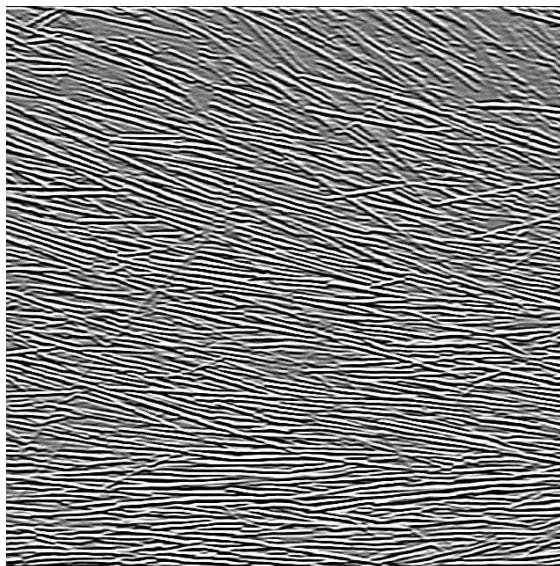
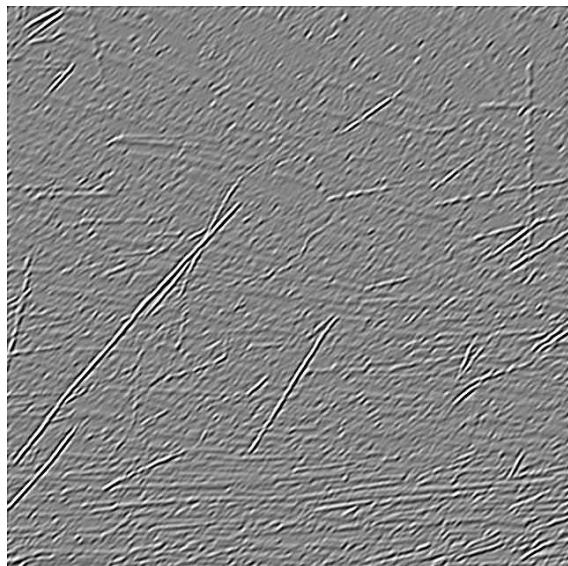
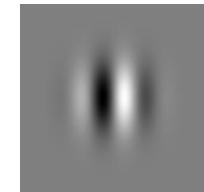
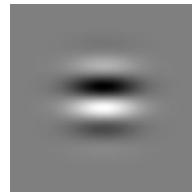
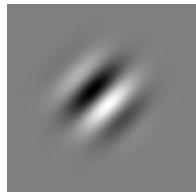
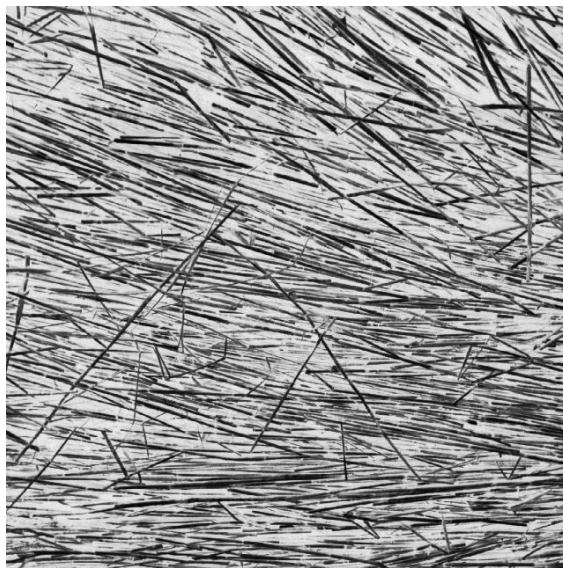


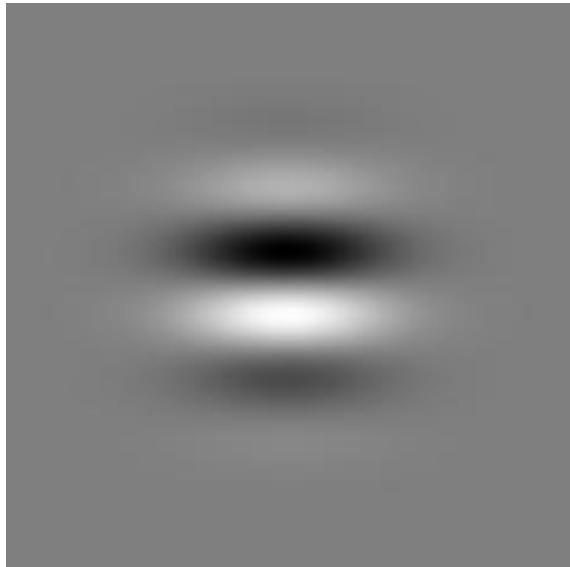
$$e^{-\frac{x^2}{2\sigma^2}} \cos(2\pi\omega x)$$

# 2D Gabor Filters

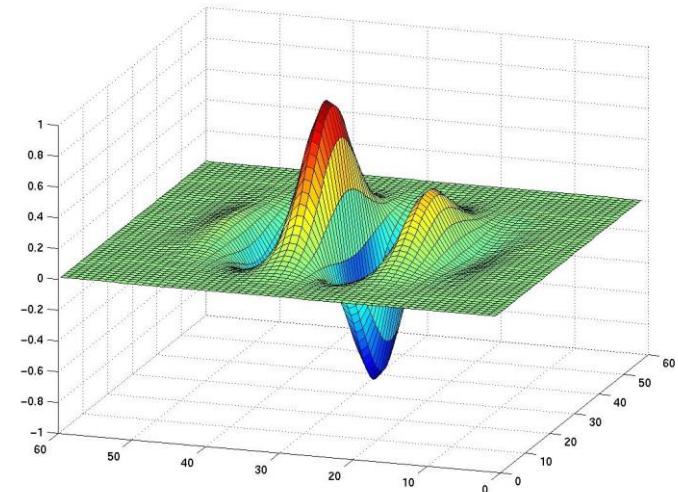
$$e^{-\frac{x^2+y^2}{2\sigma^2}} \cos(2\pi(k_x x + k_y y))$$



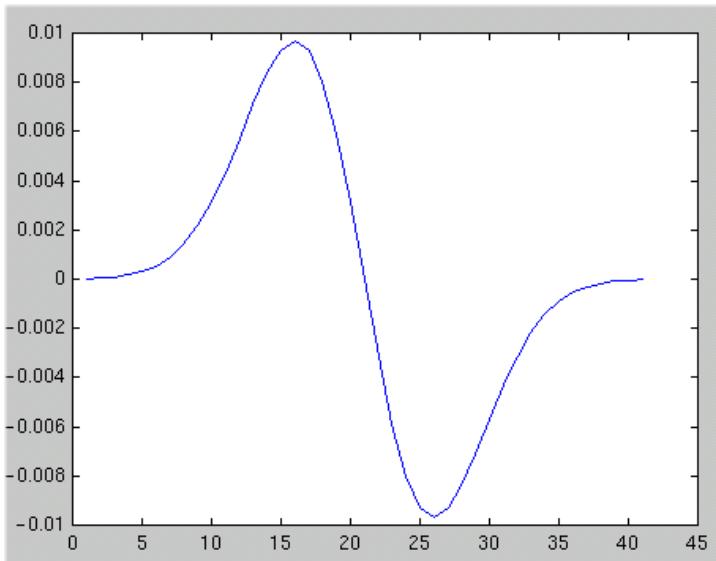




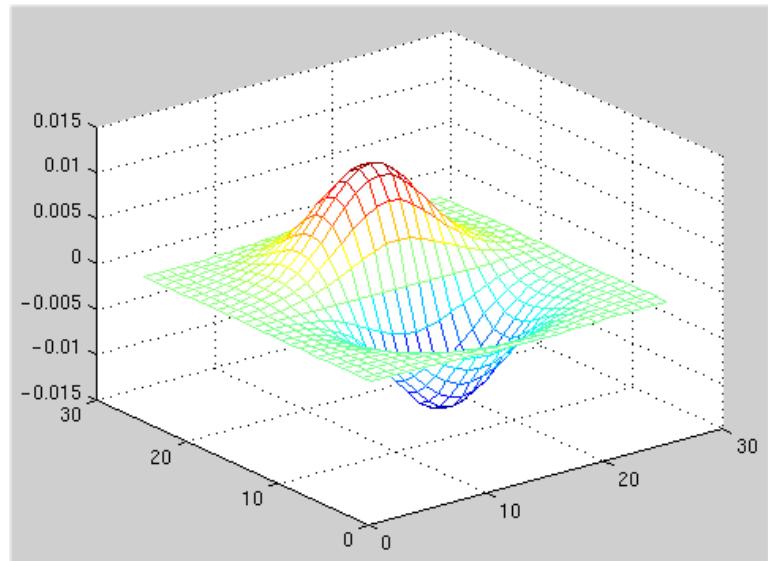
Odd  
Gabor  
filter

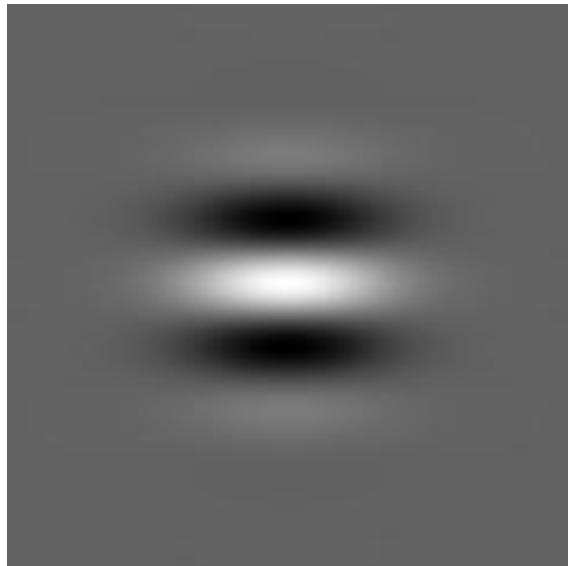


... looks a lot like...

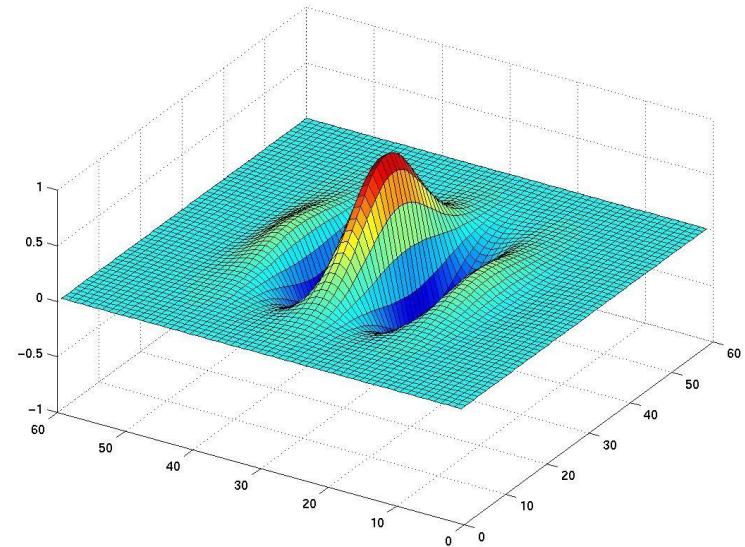


Gaussian  
Derivative

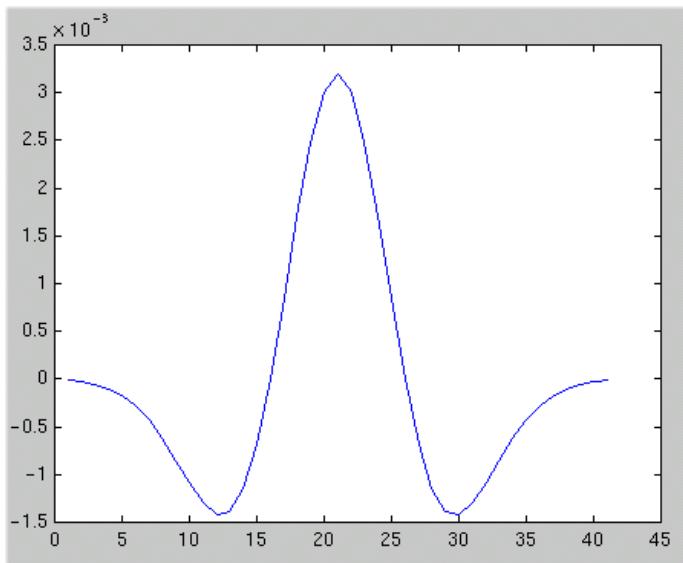




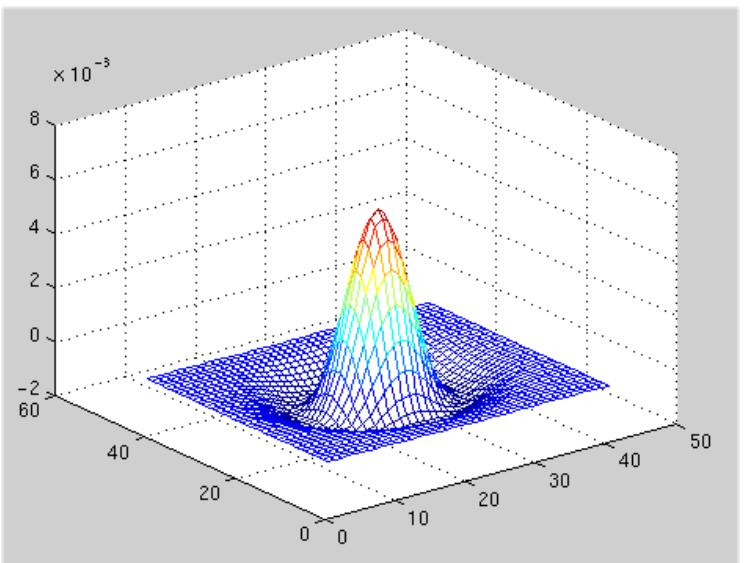
Even  
Gabor  
filter



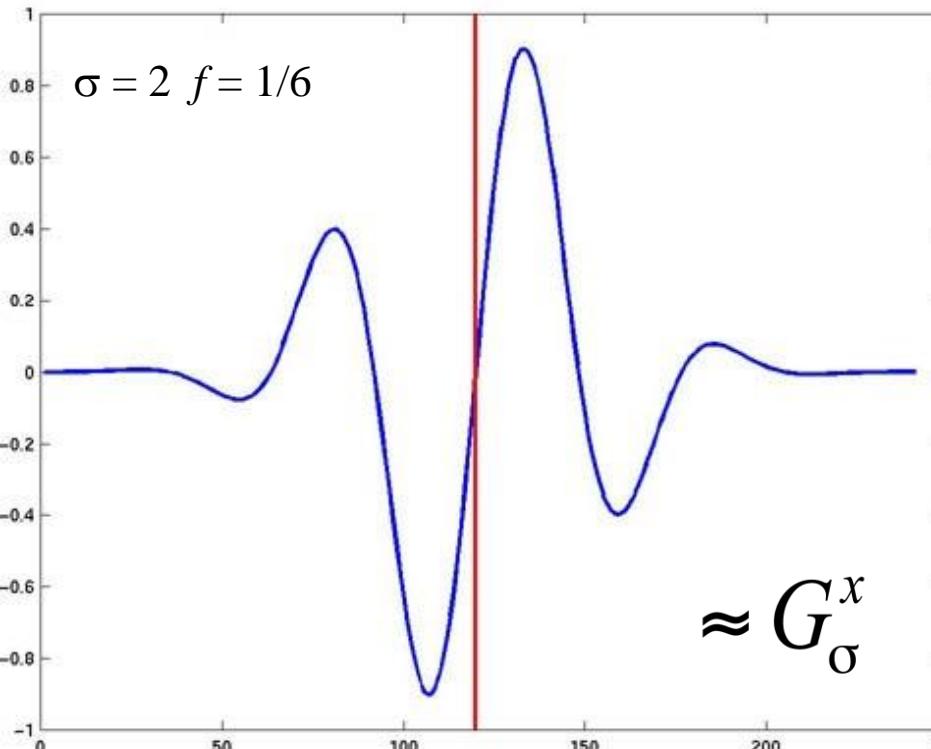
... looks a lot like...



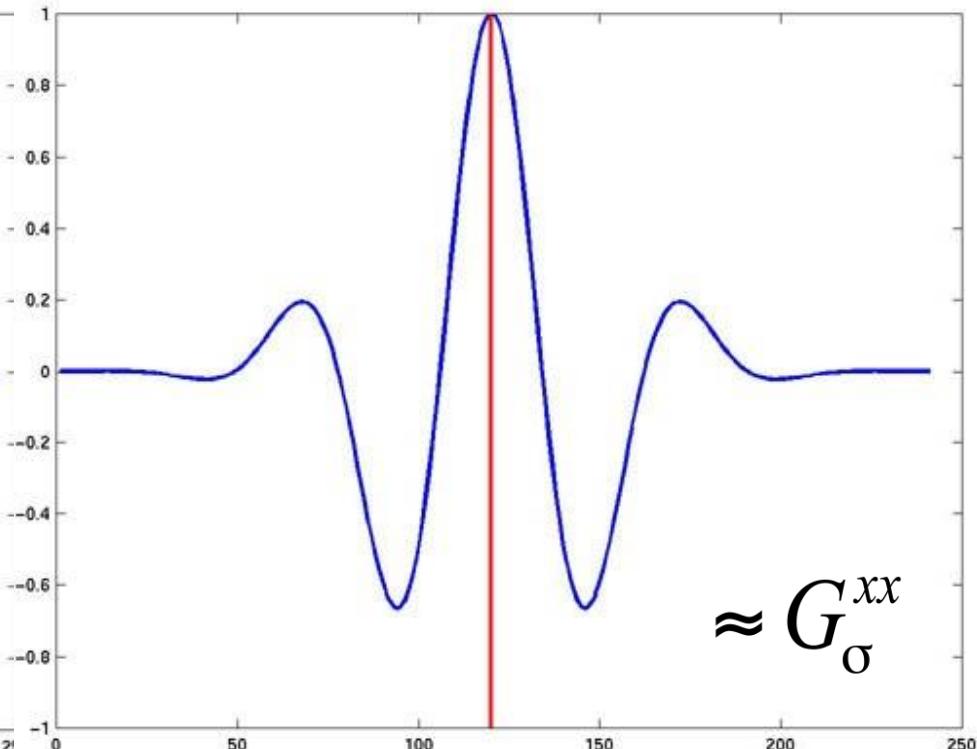
Laplacian



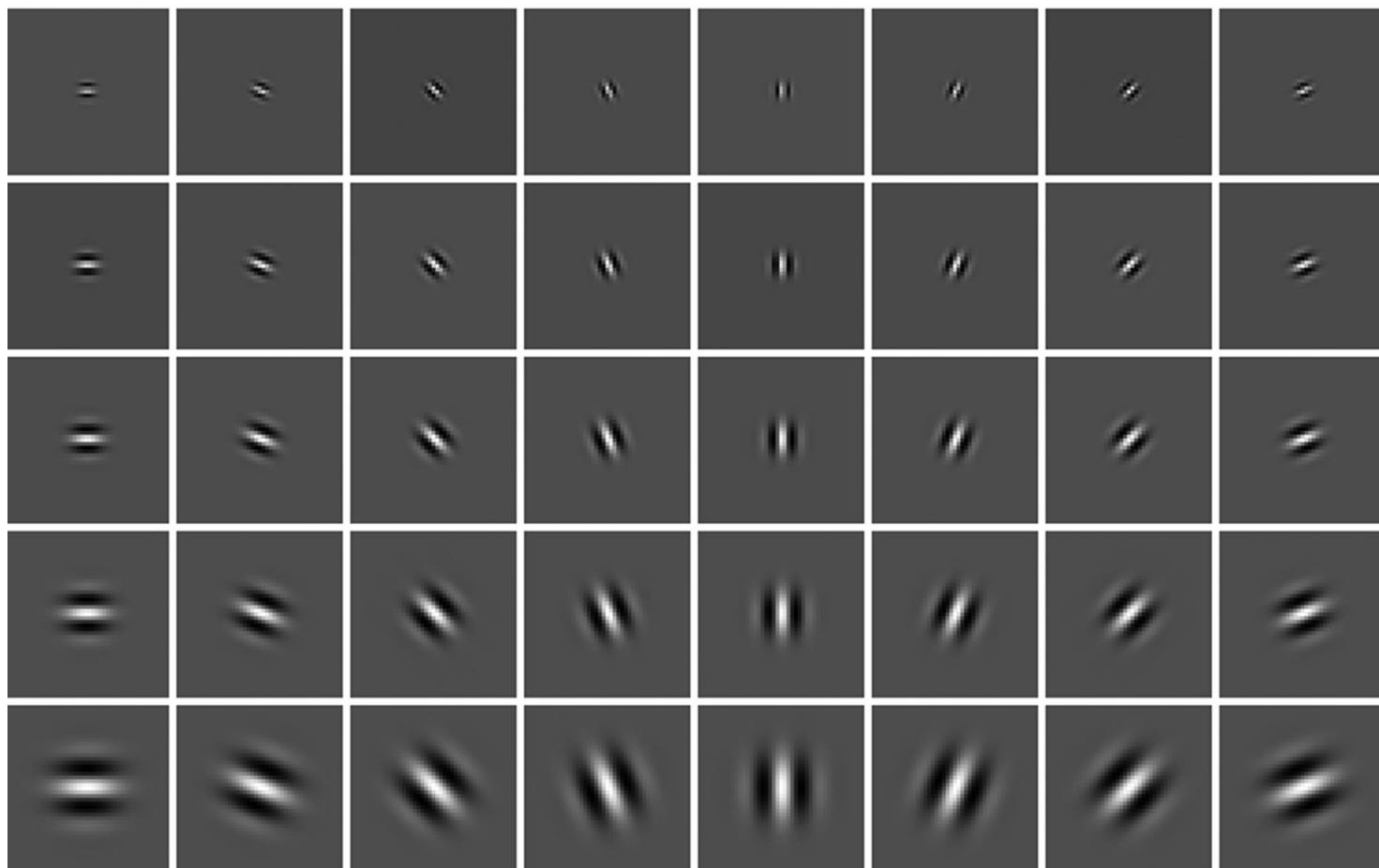
If scale small compared to inverse frequency, the Gabor filters become derivative operators



$$\approx G_{\sigma}^x$$



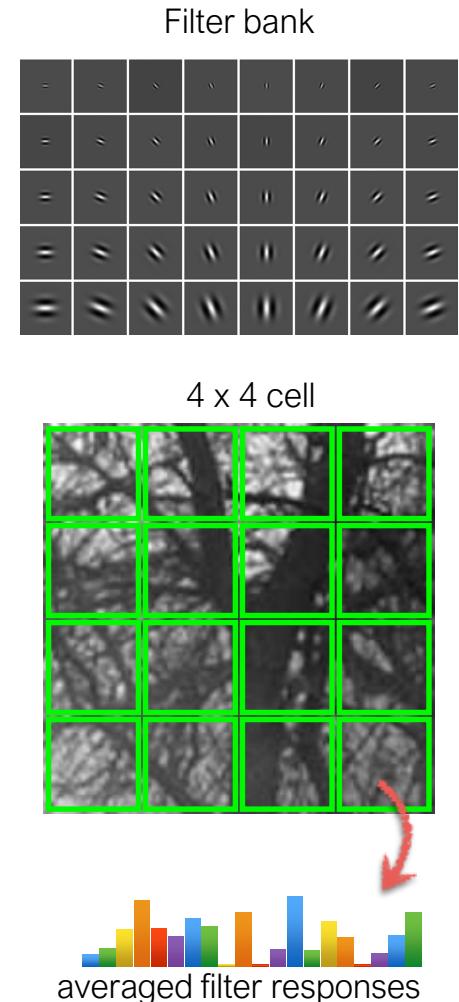
## Directional edge detectors



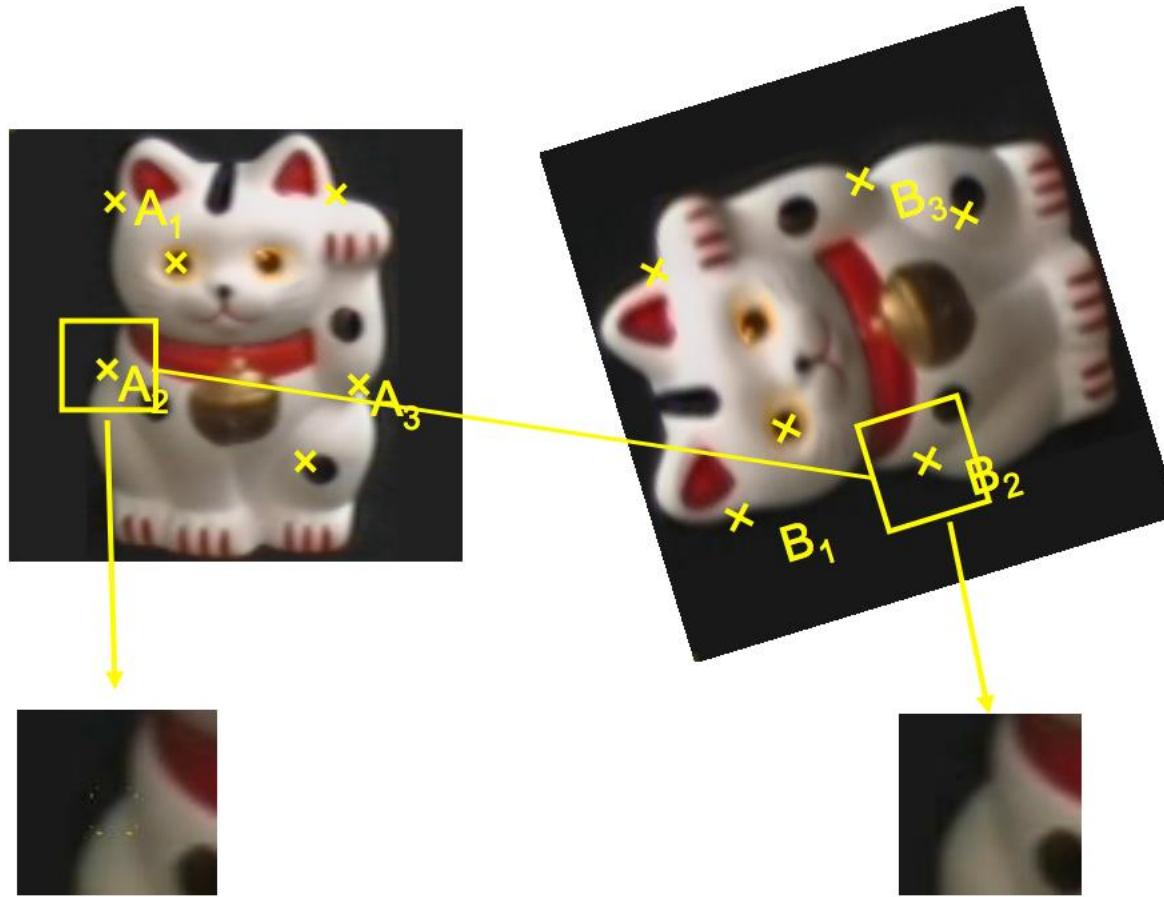
# GIST

1. Compute filter responses (filter bank of Gabor filters)
2. Divide image patch into  $4 \times 4$  cells
3. Compute filter response averages for each cell
4. Size of descriptor is  $4 \times 4 \times N$ , where  $N$  is the size of the filter bank

*What is the GIST descriptor encoding?*



# Becoming rotation invariant



# Orientation Assignment

- Use scale of point to choose correct image:

$$L(x, y) = G(x, y, \sigma) * I(x, y)$$

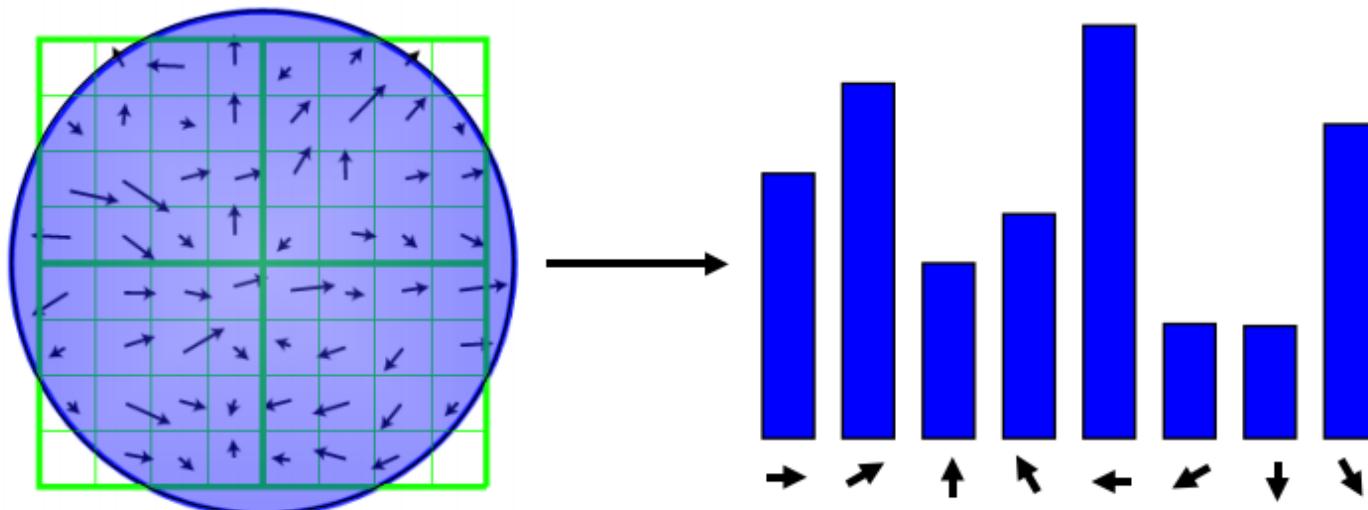
- Compute gradient magnitude and orientation using finite differences:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

$$\theta(x, y) = \tan^{-1} \left( \frac{(L(x, y+1) - L(x, y-1))}{(L(x+1, y) - L(x-1, y))} \right)$$

# Orientation Assignment

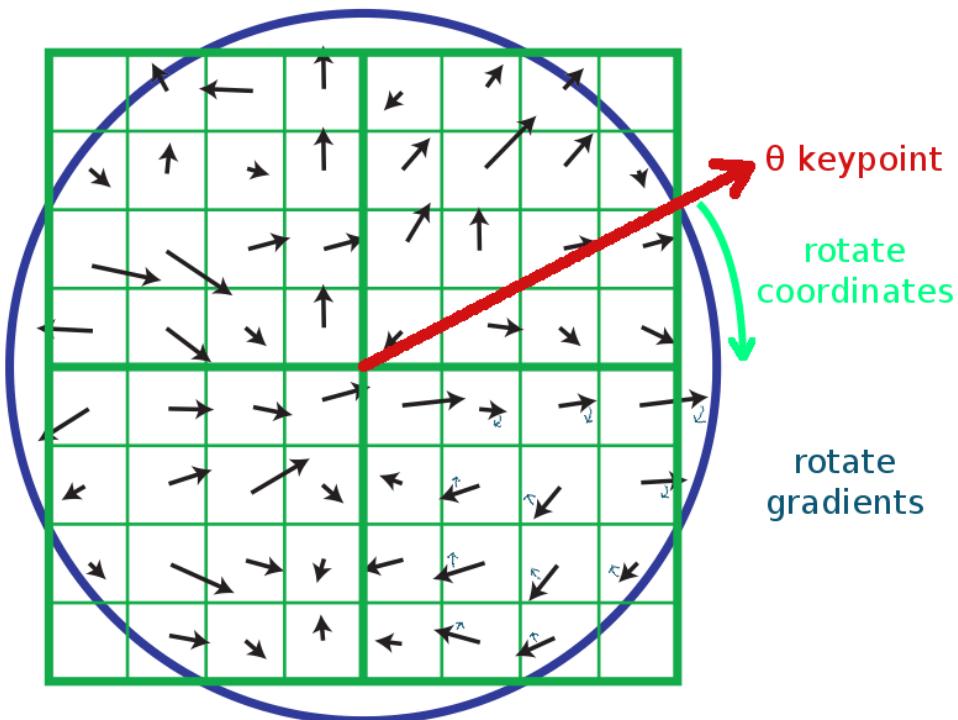
- Create gradient histogram (36 bins)
  - Weighted by magnitude and Gaussian window ( $\sigma$  is 1.5 times that of the scale of a keypoint)



# Orientation Assignment

- Any peak within 80% of the highest peak is used to create a keypoint with that orientation
- ~15% assigned multiple orientations, but contribute significantly to the stability
- Finally a parabola is fit to the 3 histogram values closest to each peak to interpolate the peak position for better accuracy

# SIFT descriptor formation



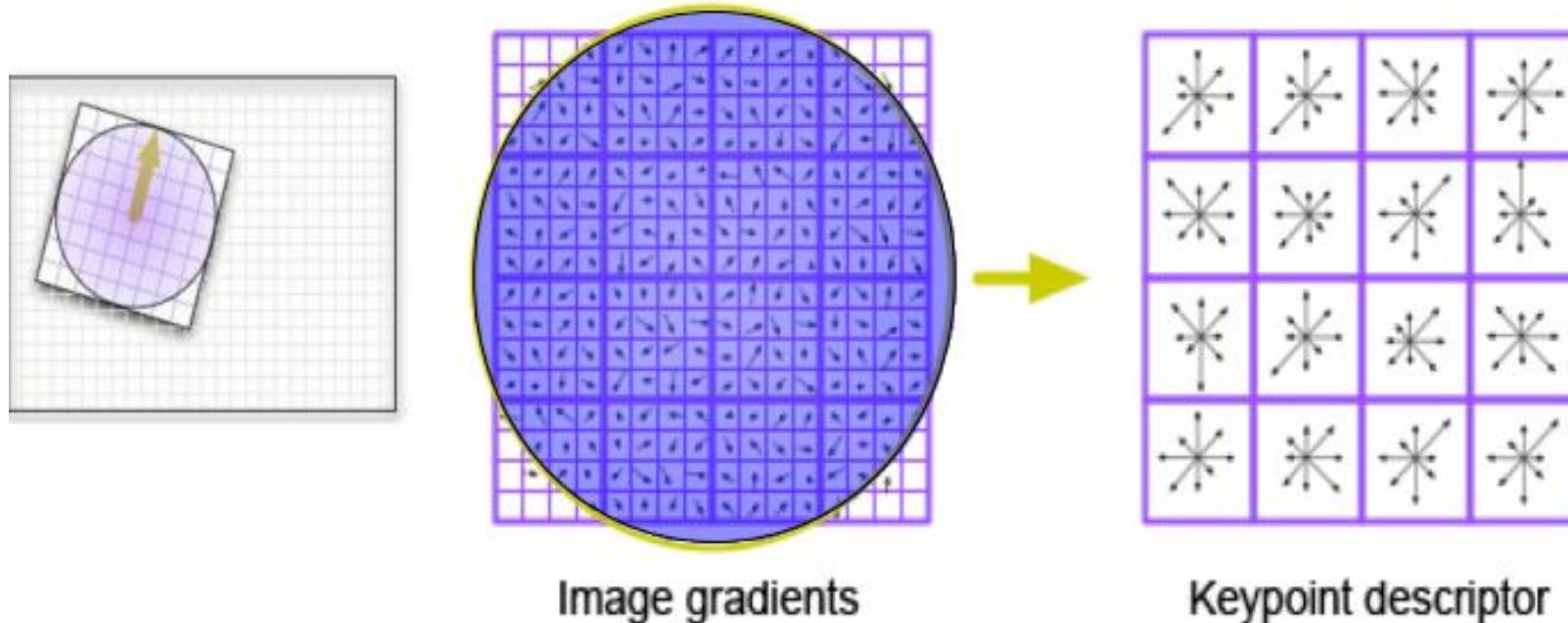
- To become rotation invariant, rotate the gradient directions AND locations by (-keypoint orientation)
  - Now we've cancelled out rotation and have gradients expressed at locations **relative** to keypoint orientation  $\theta$
  - We could also have just rotated the whole image by  $-\theta$ , but that would be slower.

# SIFT descriptor formation

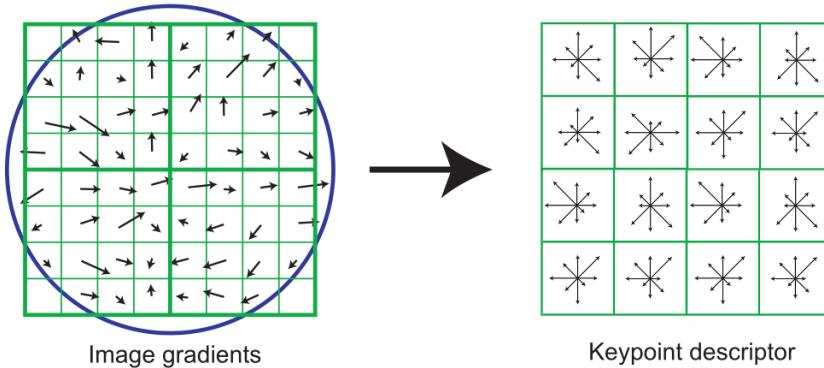
- Each point so far has  $x, y, \sigma, m, \theta$
- Now we need a descriptor for the region
  - Could sample intensities around point, but...
    - Sensitive to lighting changes
    - Sensitive to slight errors in  $x, y, \theta$

# SIFT descriptor formation

- 4x4 Gradient window
- Histogram of 4x4 samples per window in 8 directions
- Gaussian weighting around center ( $\sigma$  is 0.5 times that of the scale of a keypoint)
- $4 \times 4 \times 8 = 128$  dimensional feature vector



# SIFT descriptor formation



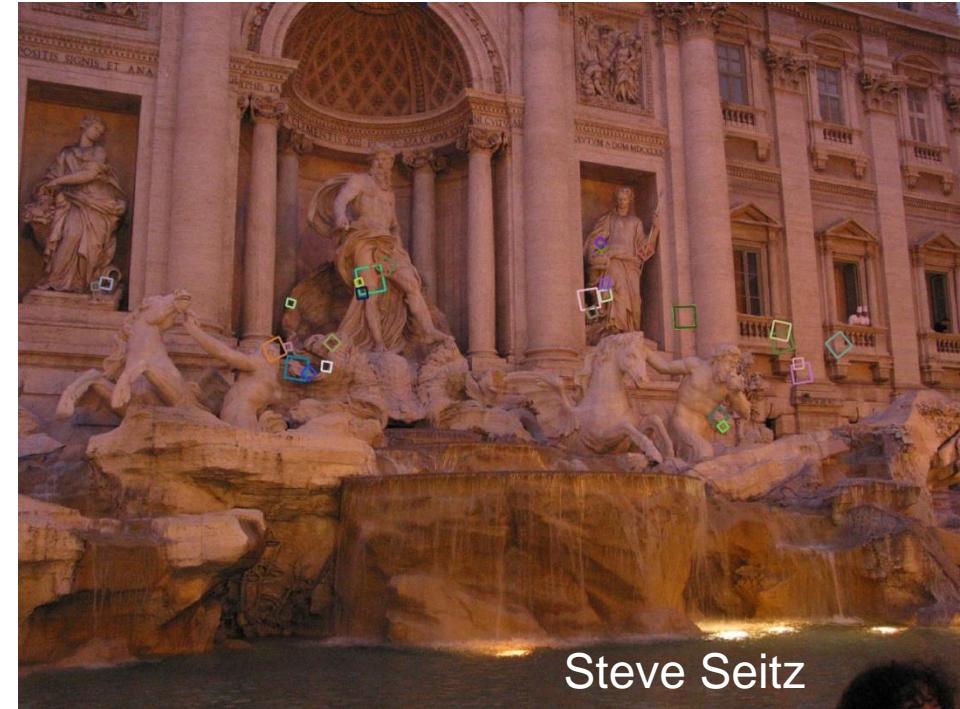
- Adding robustness to illumination changes:
- Remember that the descriptor is made of gradients (differences between pixels), so it's already invariant to changes in brightness (e.g. adding 10 to all image pixels yields the exact same descriptor)
- A higher-contrast photo will increase the magnitude of gradients linearly. So, to correct for contrast changes, normalize the vector (scale to length 1.0)
- Very large image gradients are usually from unreliable 3D illumination effects (glare, etc). So, to reduce their effect, clamp all values in the vector to be  $\leq 0.2$  (an experimentally tuned value). Then normalize the vector again.
- Result is a vector which is fairly invariant to illumination changes.

# Feature descriptors: 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)
- Fast and efficient—can run in real time
- Lots of code available
  - [http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known\\_implementations\\_of\\_SIFT](http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT)



Steve Seitz

# Histogram of Oriented Gradients

---

Find robust feature set that allows object form to be discriminated.

## Challenges

- Wide range of pose and large variations in appearances
- Cluttered backgrounds under different illumination
- “Speed” for mobile vision

## References

- [1] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In CVPR, pages 886-893, 2005
- [2] Chandrasekhar et al. CHoG: Compressed Histogram of Gradients - A low bit rate feature descriptor, CVPR 2009

---

# Histogram of Oriented Gradients

Local object appearance and shape can often be characterized rather well by the distribution of local intensity gradients or edge directions.



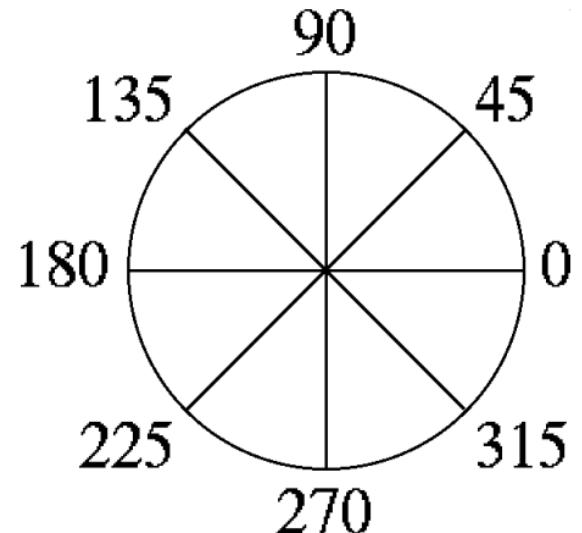
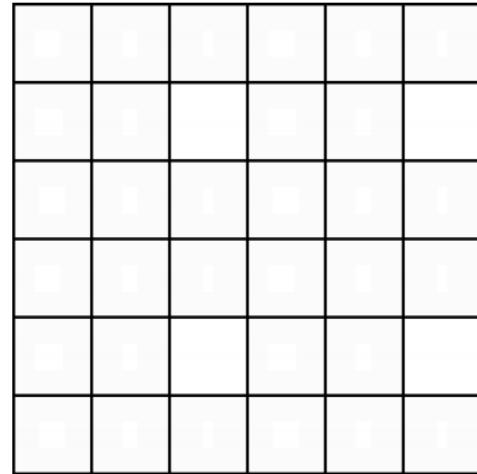
---

# Histogram of Oriented Gradients

Dividing the image window into small spatial regions (cells)

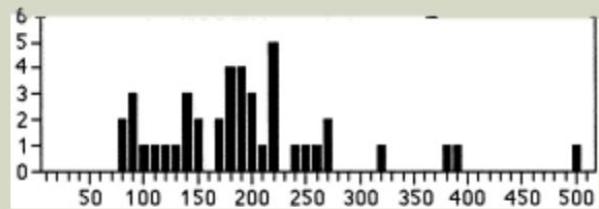
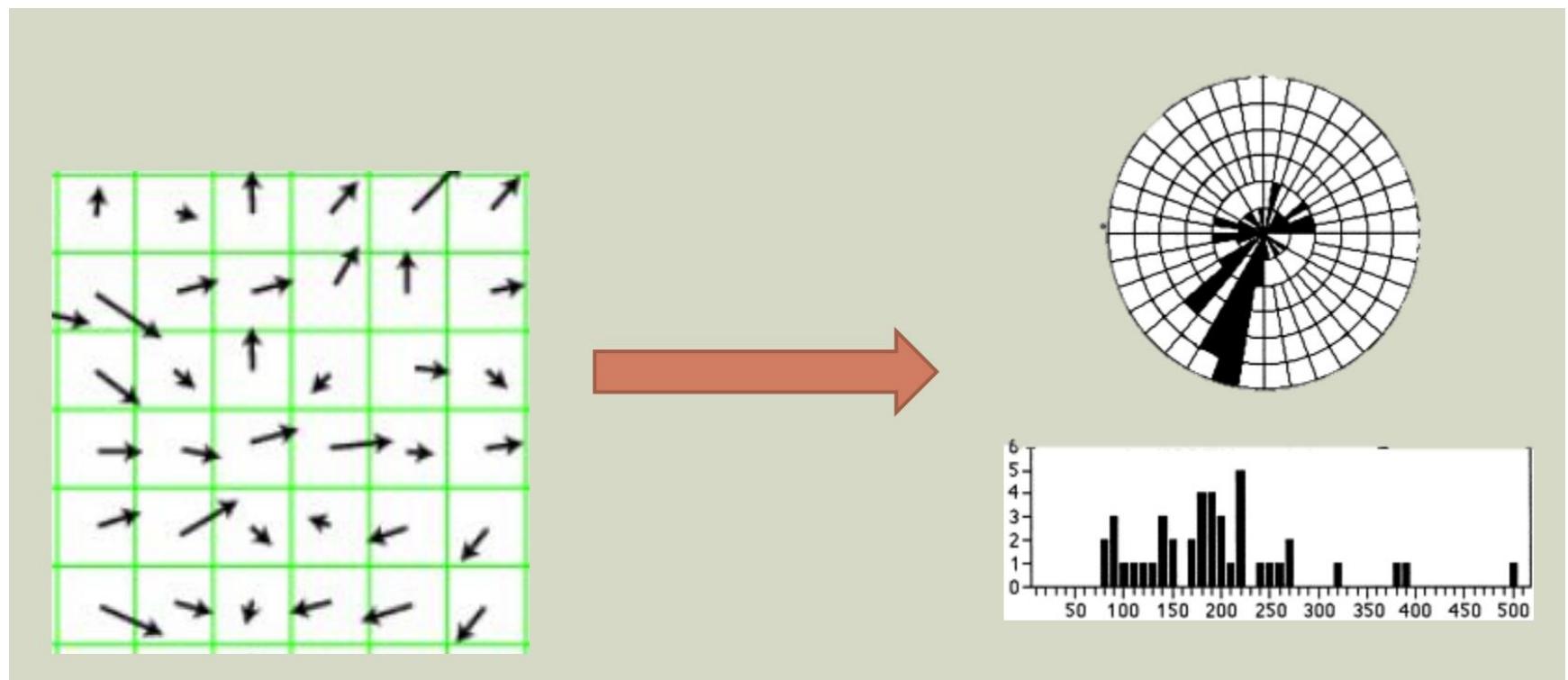
Cells can be either rectangle or radial.

Each cell accumulating a weighted local 1-D histogram of gradient directions over the pixels of the cell.



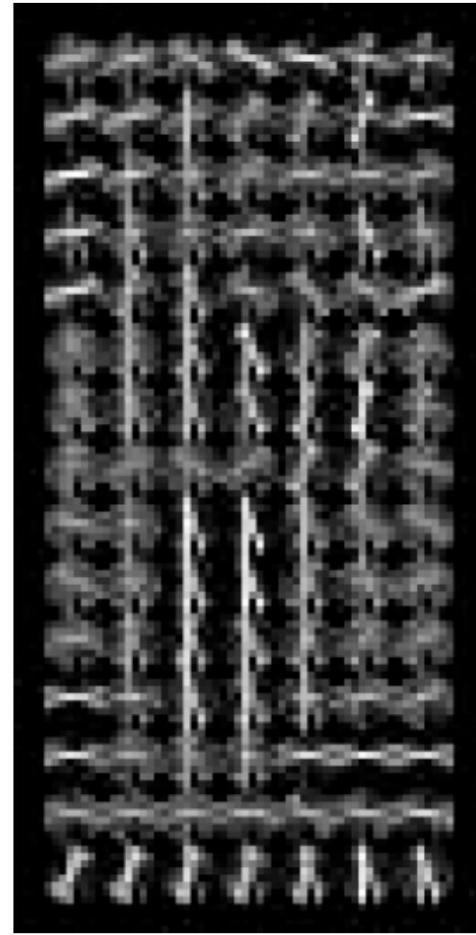
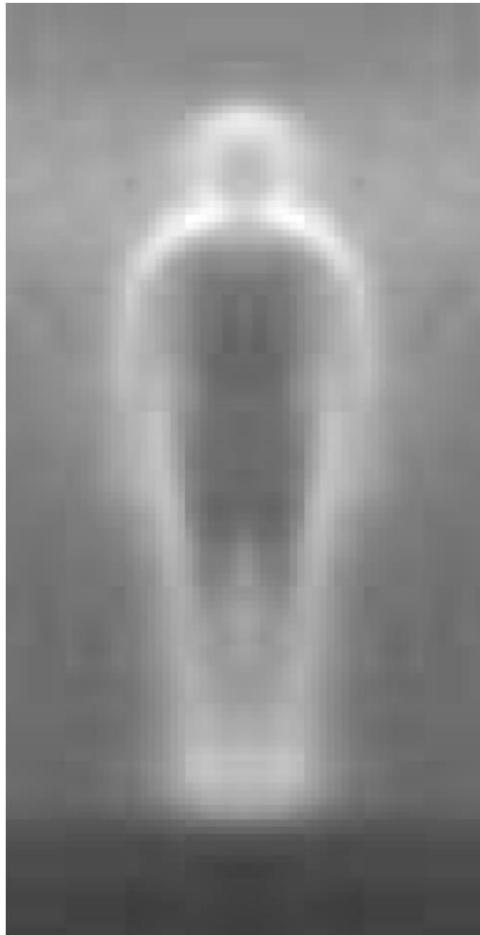
# Histogram of Oriented Gradients

---



# Histogram of Oriented Gradients

---

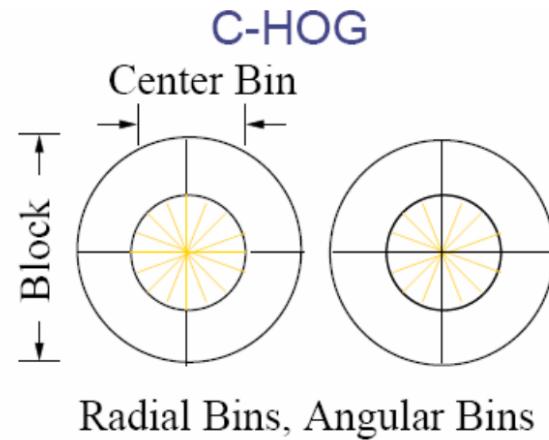
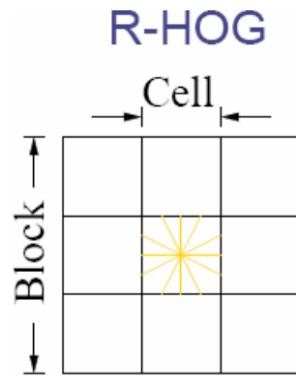


# Normalization

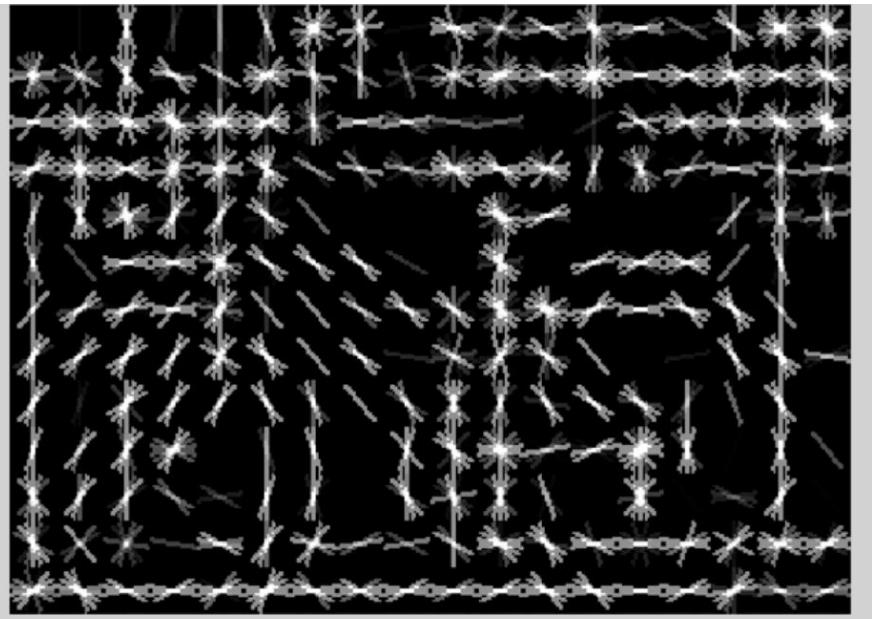
---

For better invariance to illumination and shadowing. it is useful to contrast-normalize the local responses before using them.

Accumulate local histogram “energy” over a larger regions (“blocks”) to normalize all of the cells in the block.

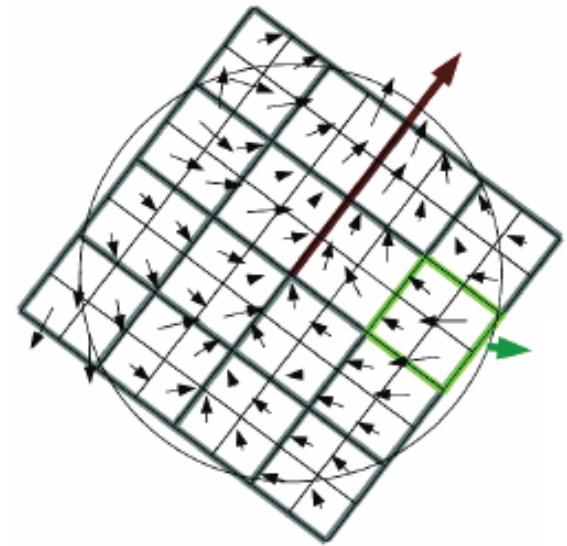


# Visualizing HoG

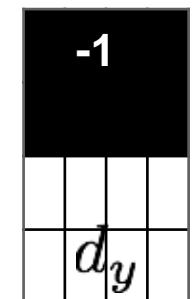
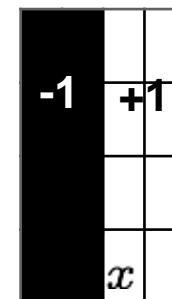


# SURF descriptor

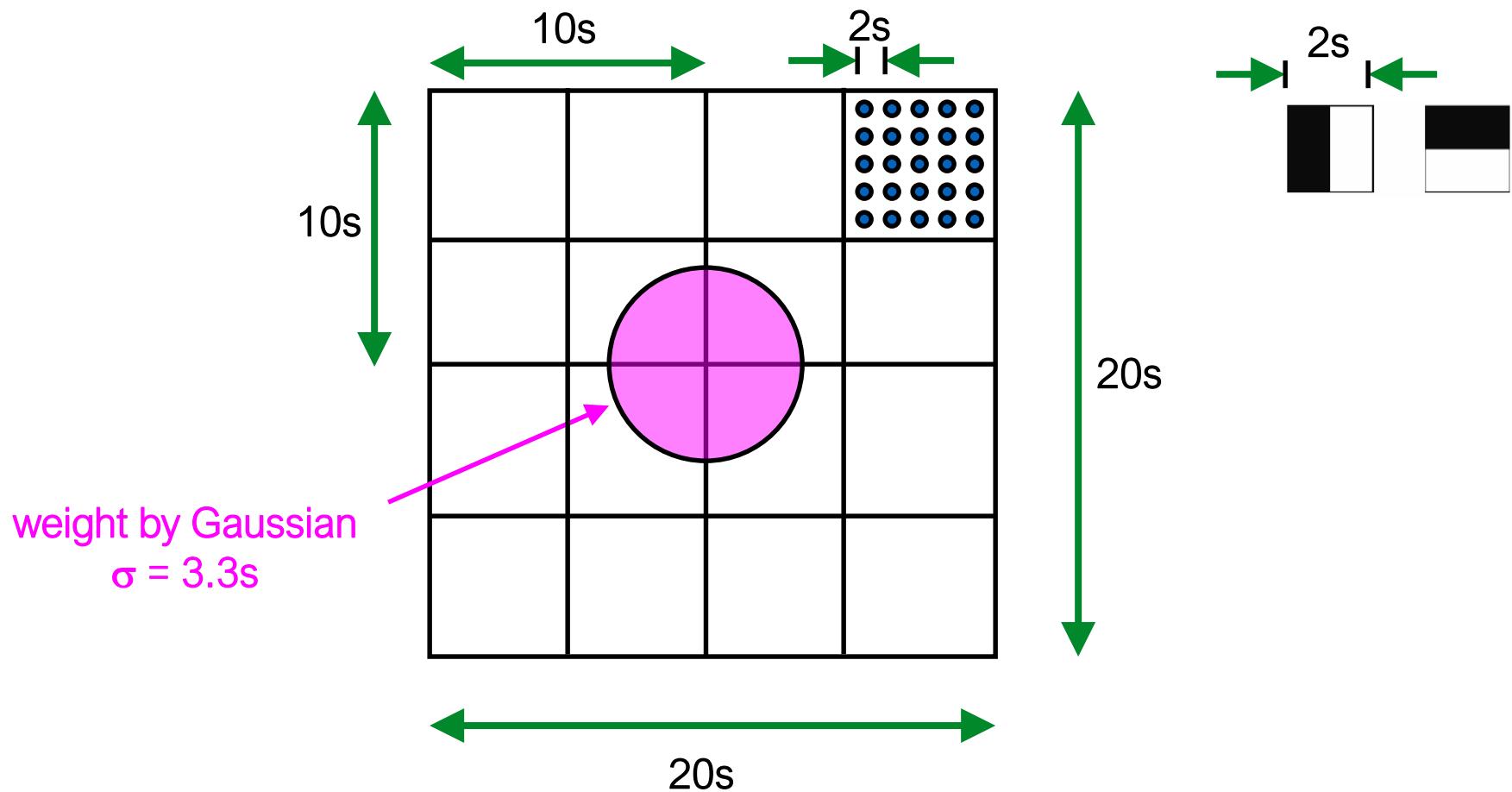
- Once interest point has been found,
  - Place window around point
  - Divide into 4x4 sub-regions
  - In each sub-regions:
    - measure at 25 (5x5) places:  $d_x$  and  $d_y$
    - sum over all 25 places to get 4 values:
$$[\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|]$$
- Compute Haar wavelet response at each place in sub-regions



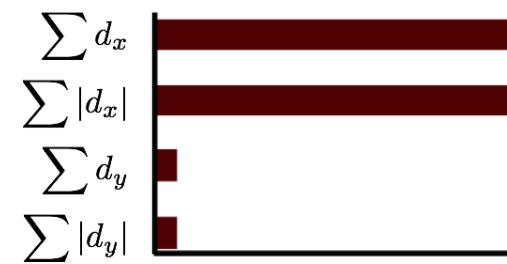
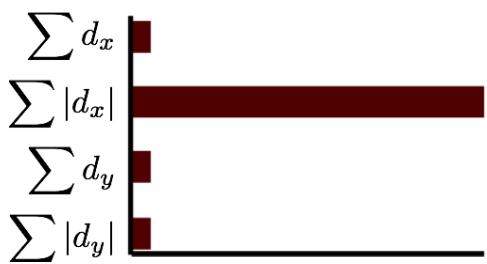
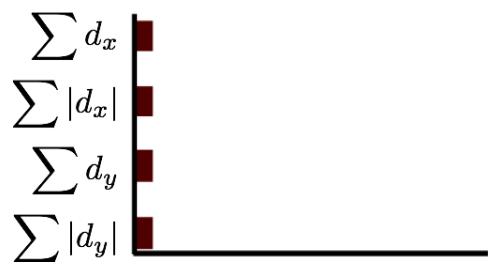
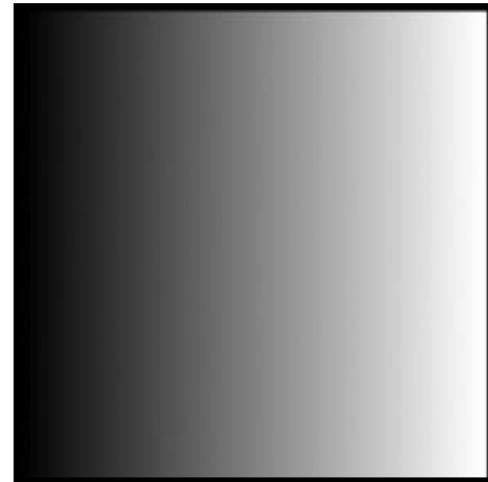
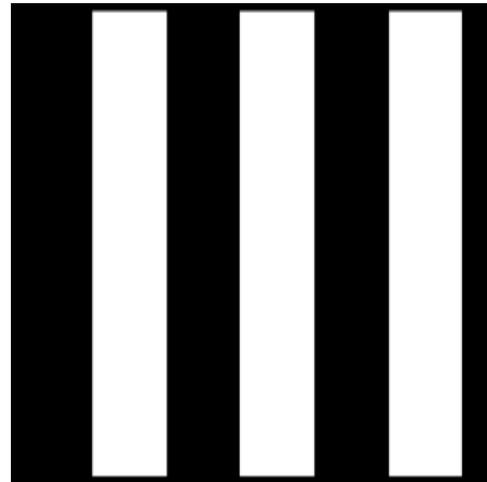
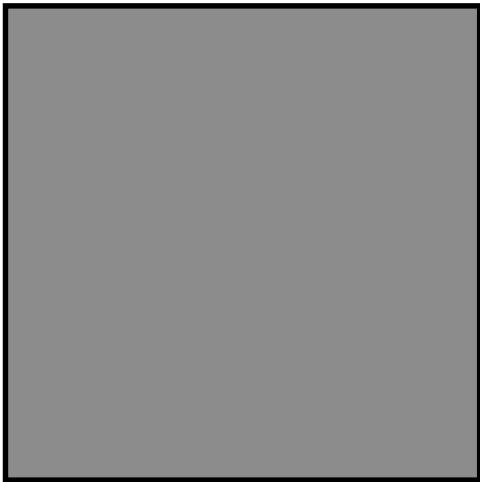
Haar wavelets filters



# Details



→ 65 values per interest point ( $16 \times 4 + 1$ )



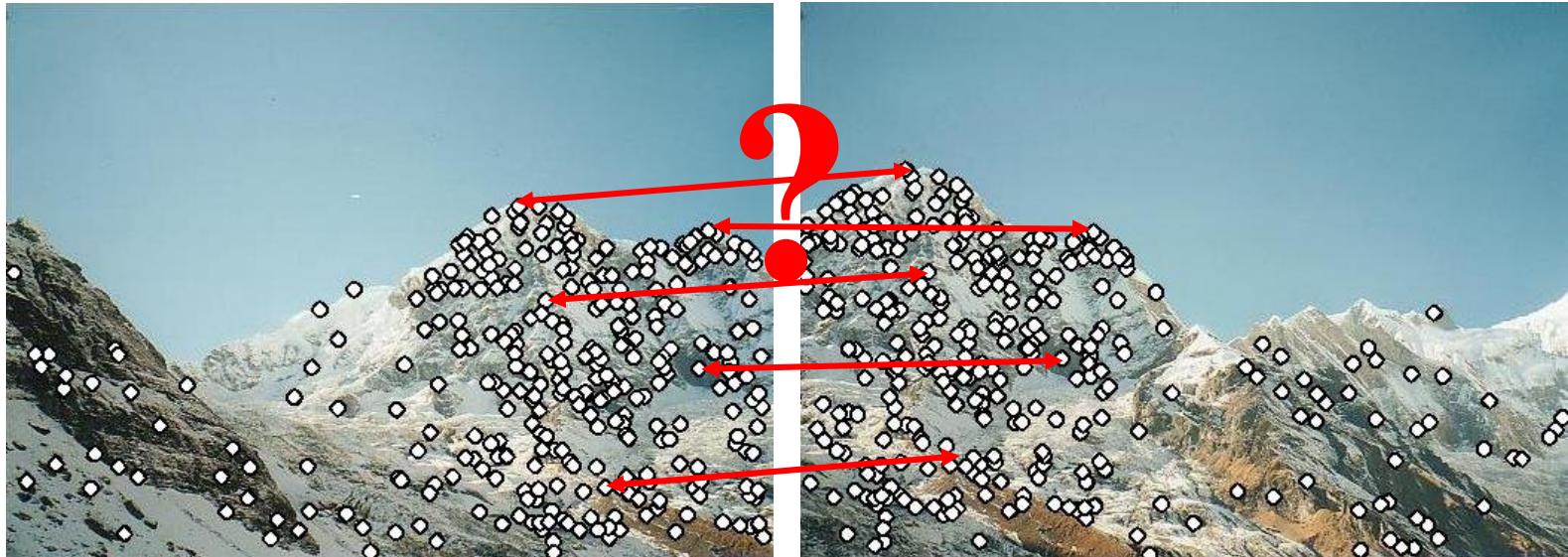
# Main questions

- Where will the interest points come from?
  - What are salient features that we'll *detect* in multiple views?
- How to *describe* a local region?
- How to establish *correspondences*, i.e., compute matches?

# Feature descriptors

---

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



# Feature matching

---

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

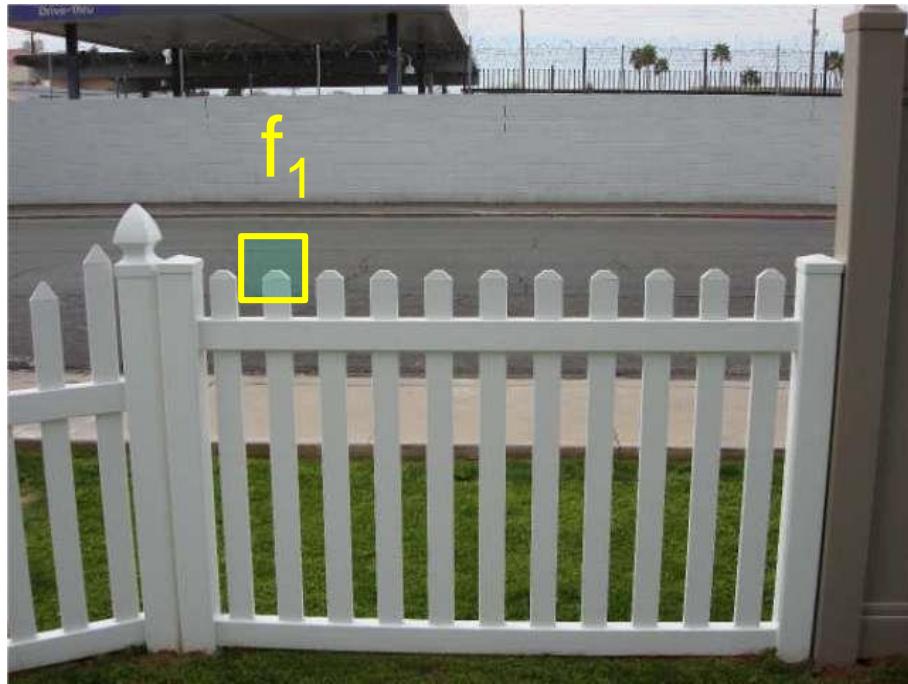
1. Define distance function that compares two descriptors
2. Test all the features in  $I_2$ , find the one with min distance

# Feature distance

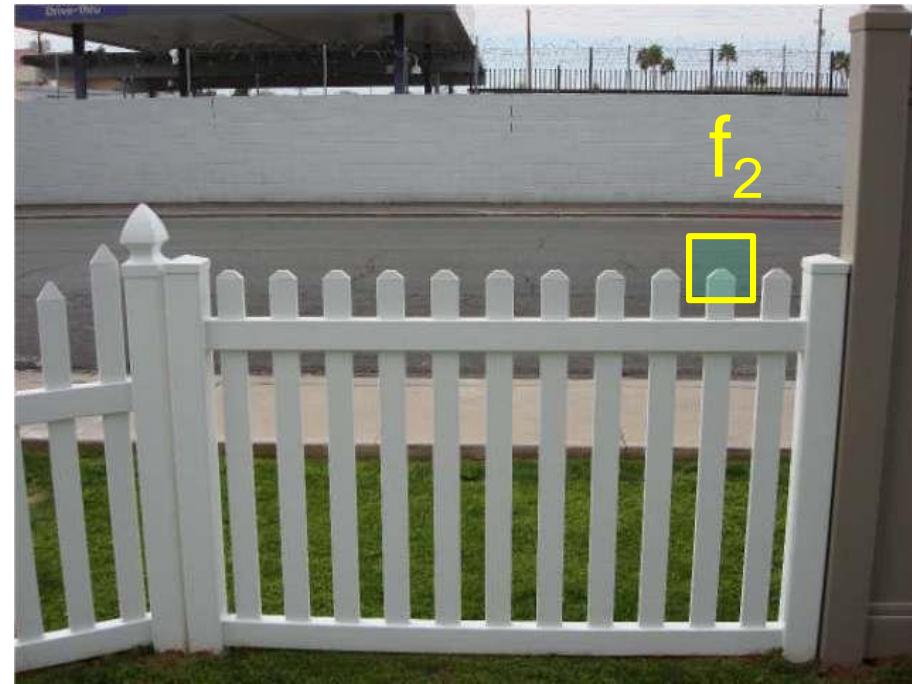
---

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

- Simple approach is  $\text{SSD}(f_1, f_2)$ 
  - sum of square differences between entries of the two descriptors
  - can give good scores to very ambiguous (bad) matches



$|_1$



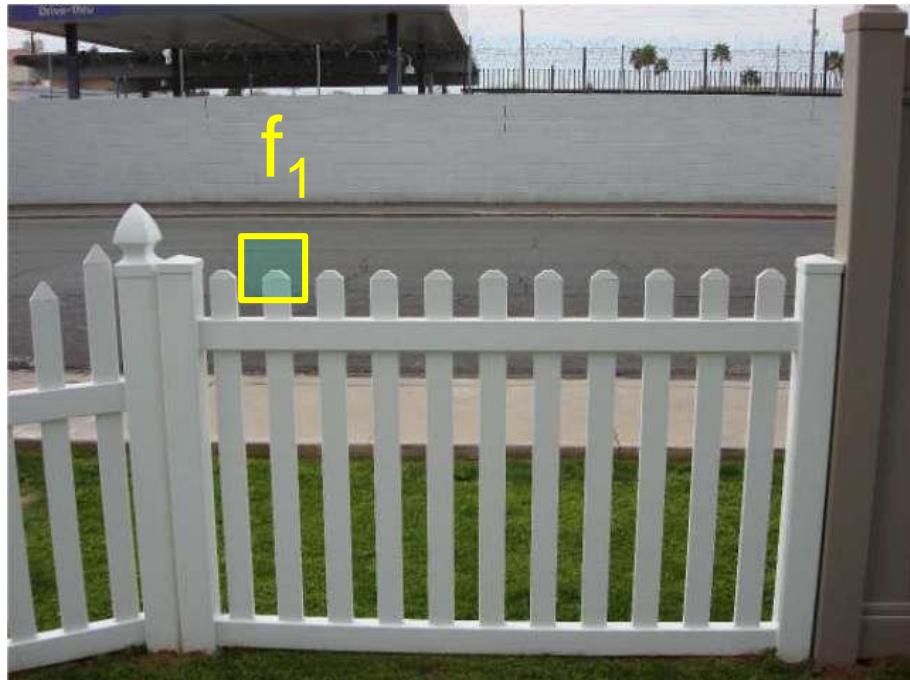
$|_2$

# Feature distance

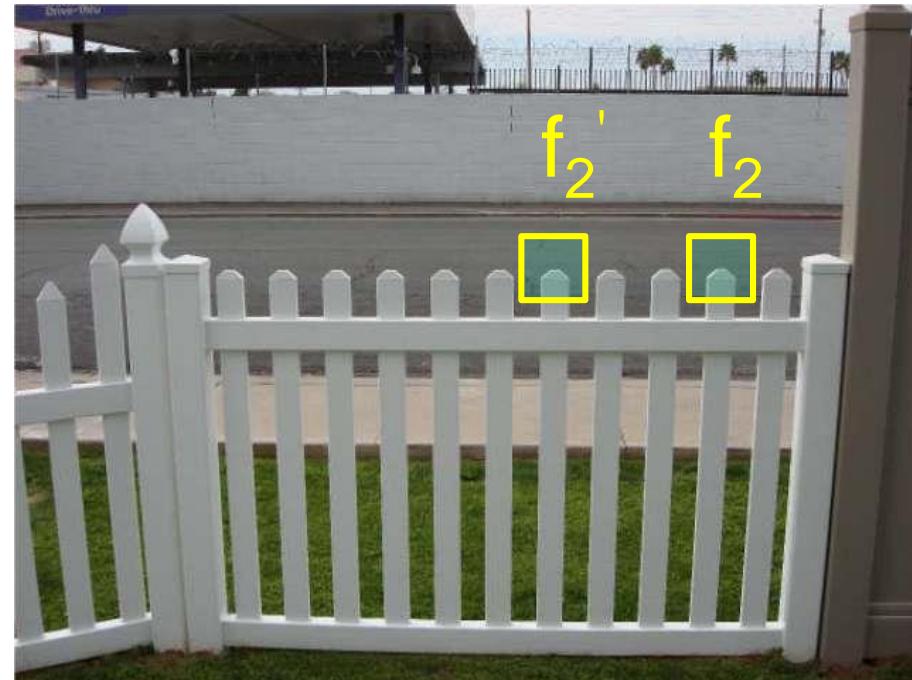
---

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

- Better approach: ratio distance =  $\text{SSD}(f_1, f_2) / \text{SSD}(f_1, f_2')$ 
  - $f_2$  is best SSD match to  $f_1$  in  $I_2$
  - $f_2'$  is 2<sup>nd</sup> best SSD match to  $f_1$  in  $I_2$
  - An ambiguous/bad match will have ratio close to 1
  - Look for unique matches which have low ratio



$I_1$

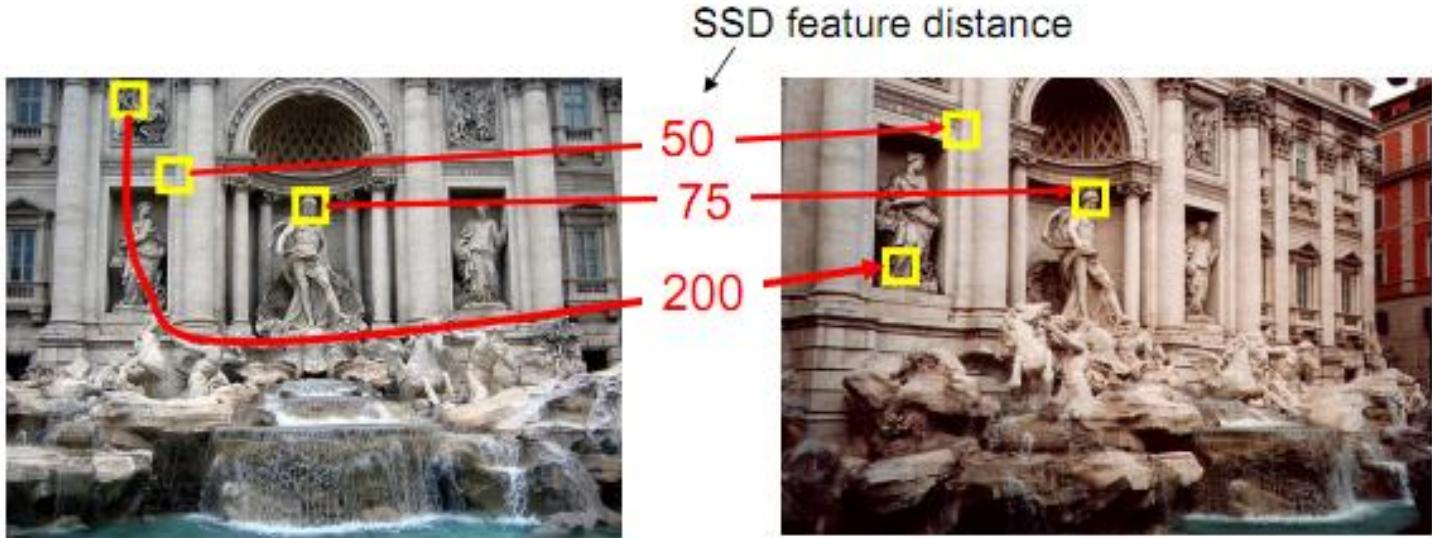


$I_2$

# Evaluating the results

---

How can we measure the performance of a feature matcher?



**Suppose we use SSD**

**Small values are possible matches but how small?**

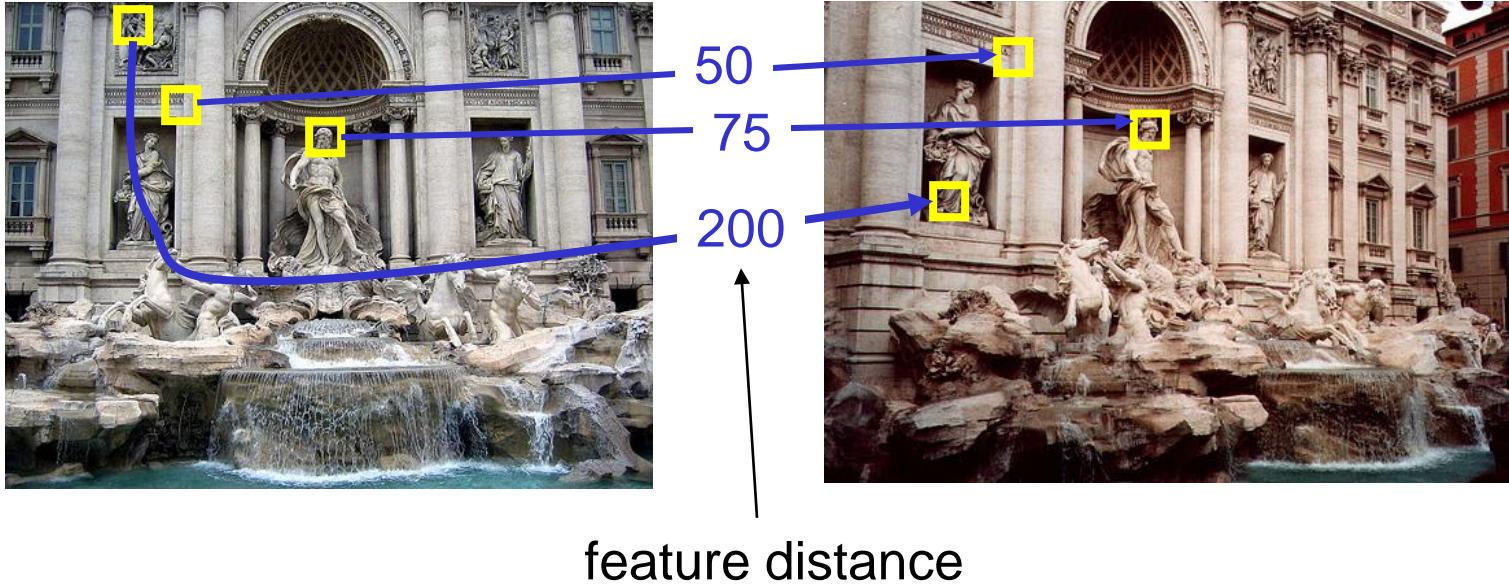
**Decision rule: Accept match if  $SSD < T$**   
**where  $T$  is a threshold**

**What is the effect of choosing a particular  $T$ ?**

# Evaluating the results

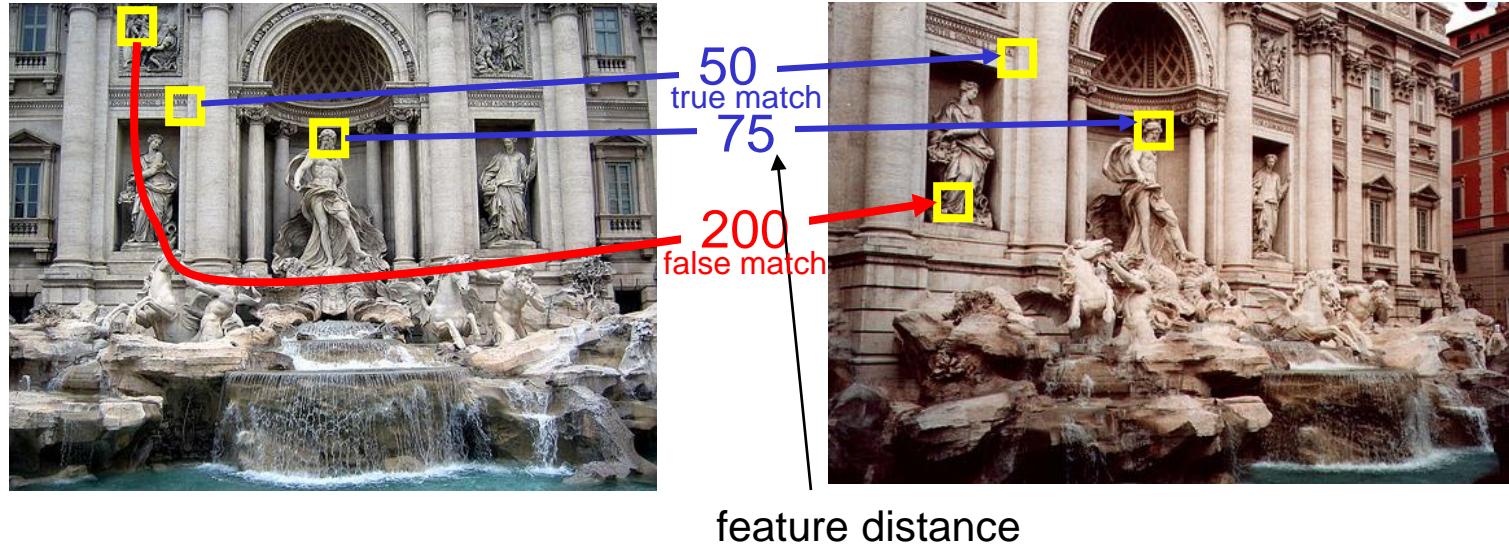
---

How can we measure the performance of a feature matcher?



# True/false positives

---

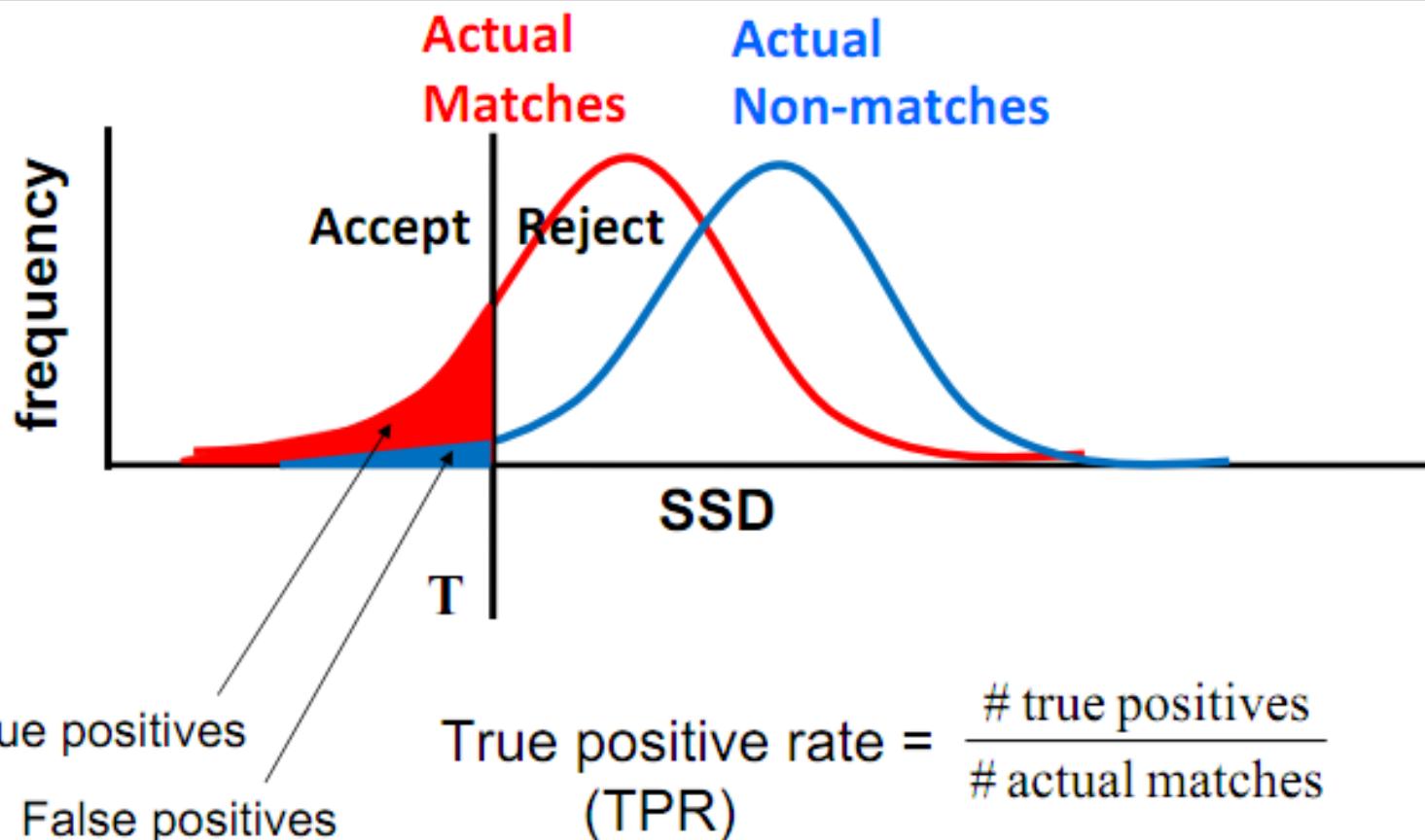


The distance threshold affects performance

- True positives = # of detected matches that are correct
  - Suppose we want to maximize these—how to choose threshold?
- False positives = # of detected matches that are incorrect
  - Suppose we want to minimize these—how to choose threshold?

# True/false positives

---

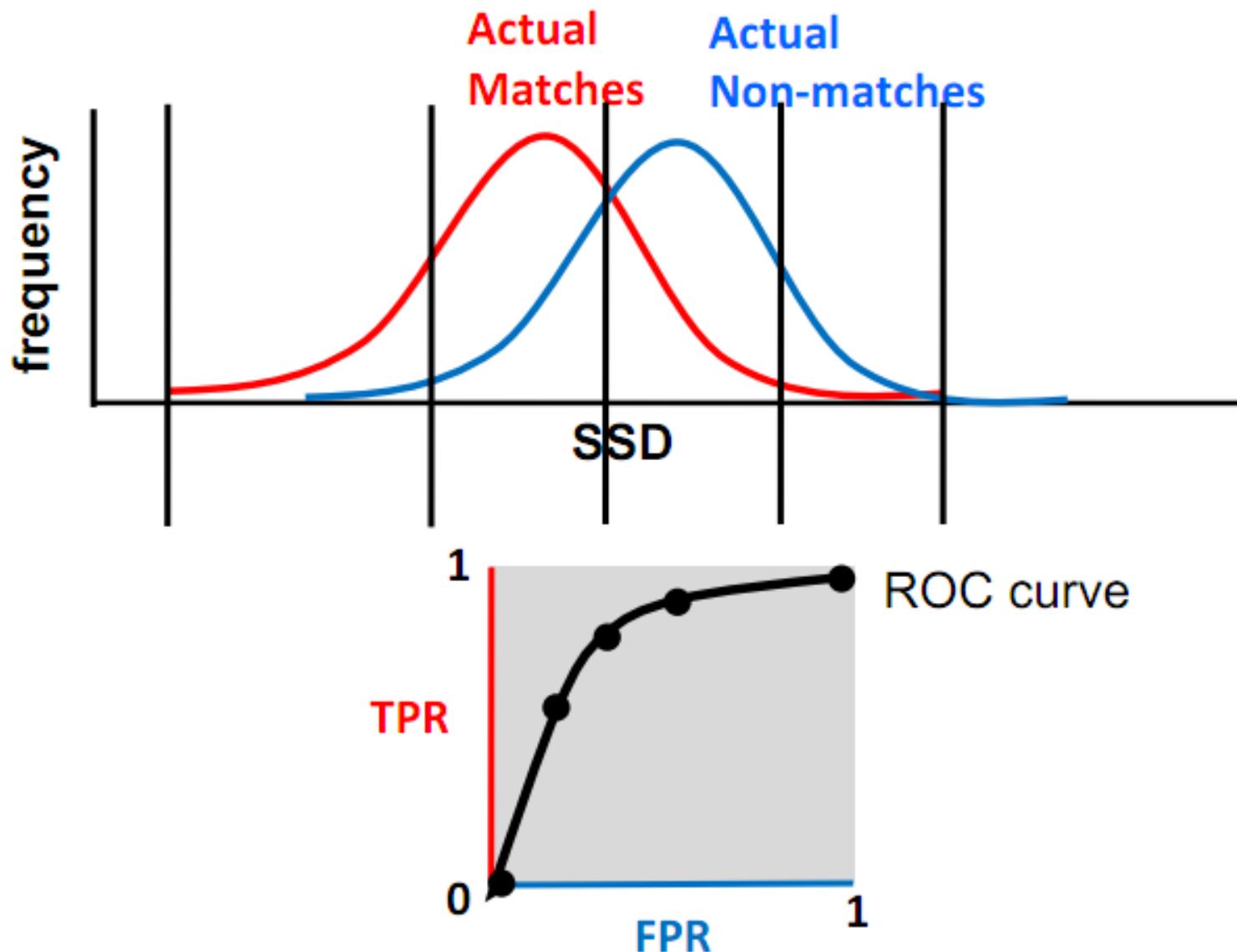


True positive rate =  $\frac{\text{\# true positives}}{\text{\# actual matches}}$

False positive rate =  $\frac{\text{\# false positives}}{\text{\# actual nonmatches}}$

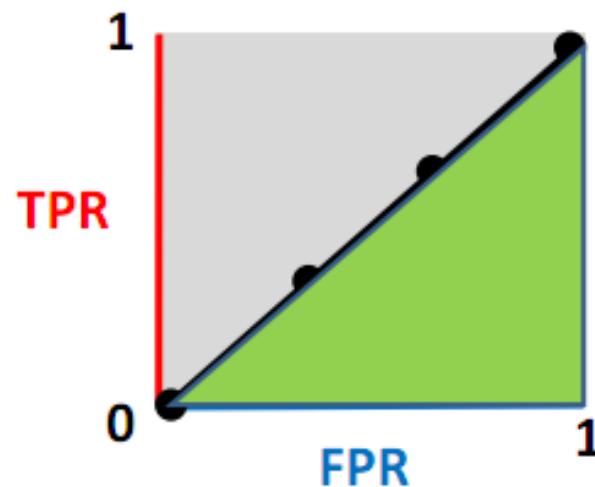
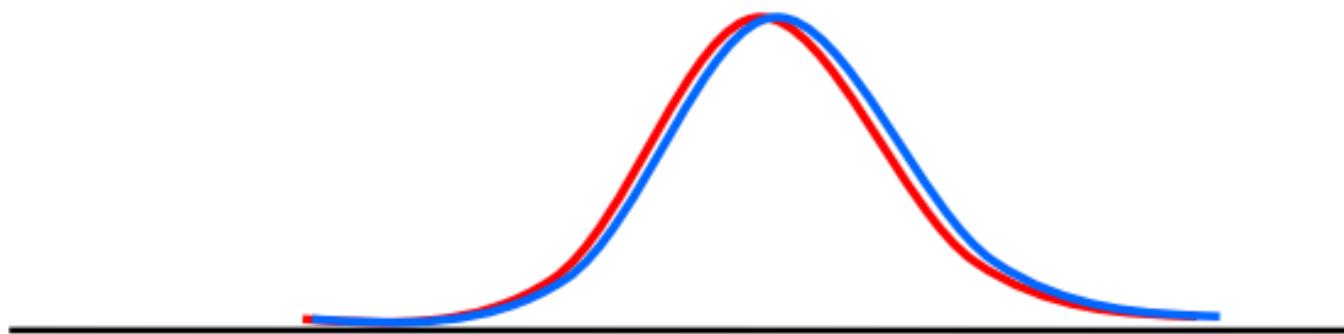
# Receiver Operating Characteristic (ROC) curve

---



# Bad feature

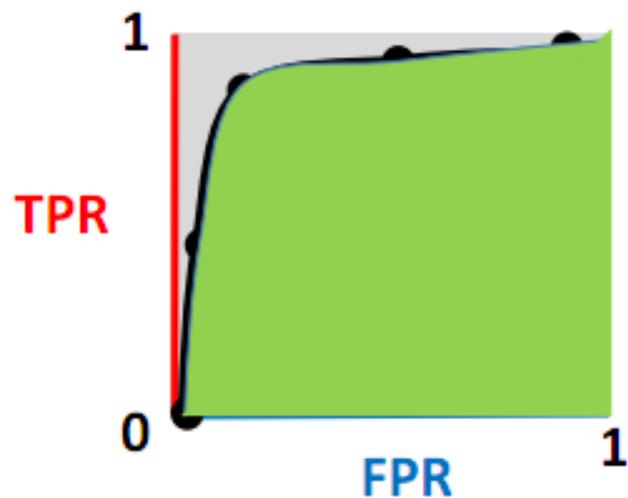
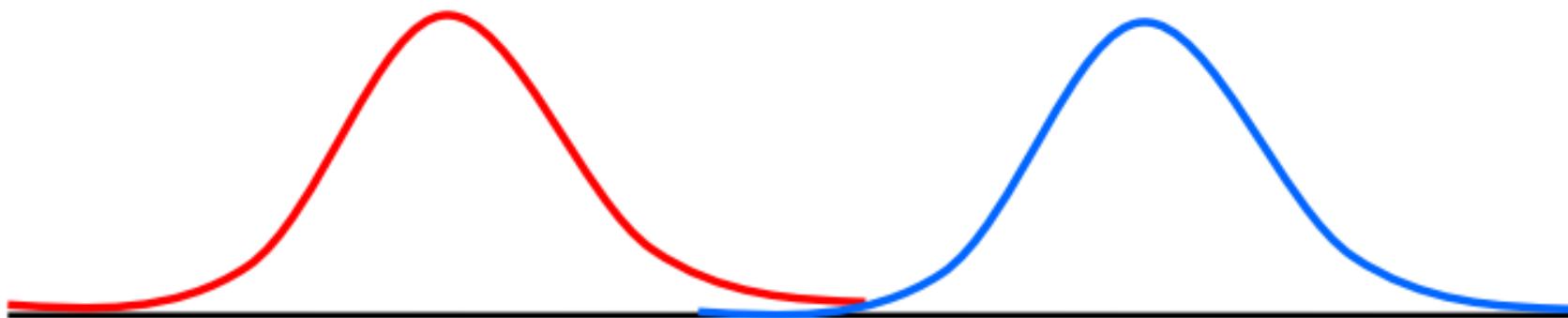
---



Adapted from a slide by Shin Kira

# Good feature

---

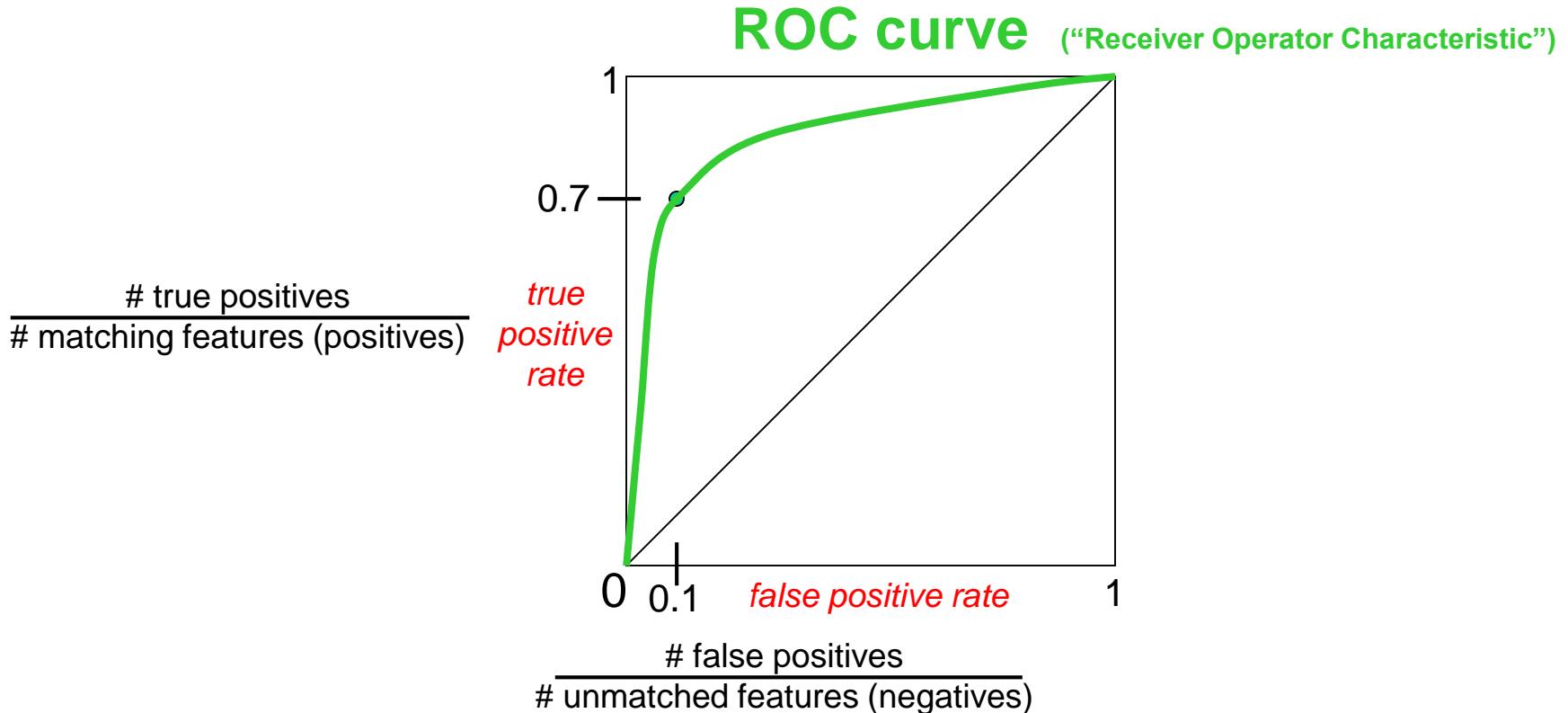


Adapted from a slide by Shin Kira

# Evaluating the results

---

How can we measure the performance of a feature matcher?



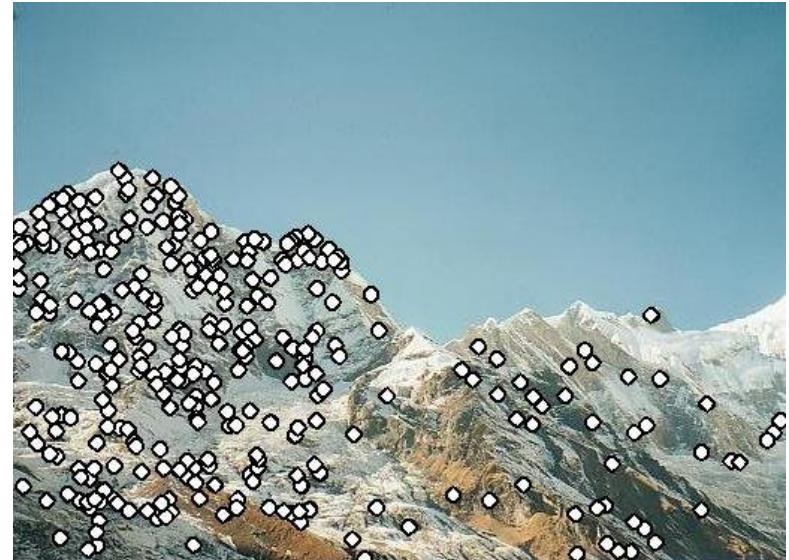
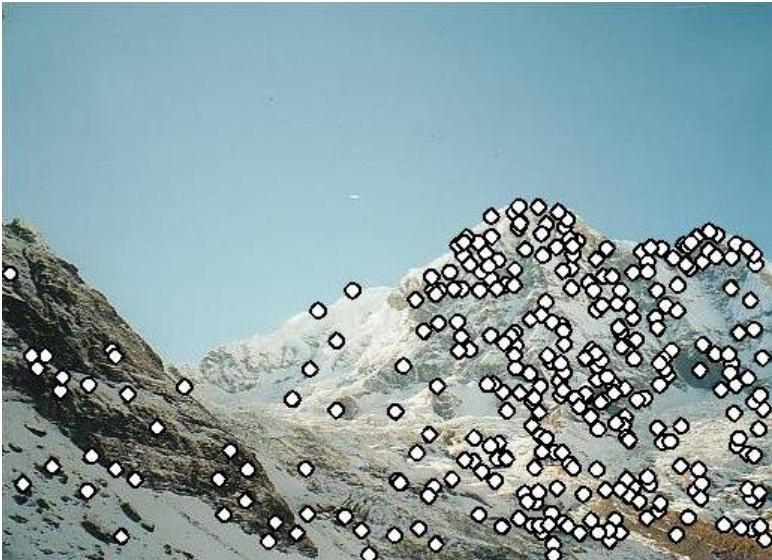
## ROC Curves

- Generated by counting # current/incorrect matches, for different thresholds
- Want to maximize area under the curve (AUC)
- Useful for comparing different feature matching methods
- For more info: [http://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](http://en.wikipedia.org/wiki/Receiver_operating_characteristic)

# Application: Image Stitching

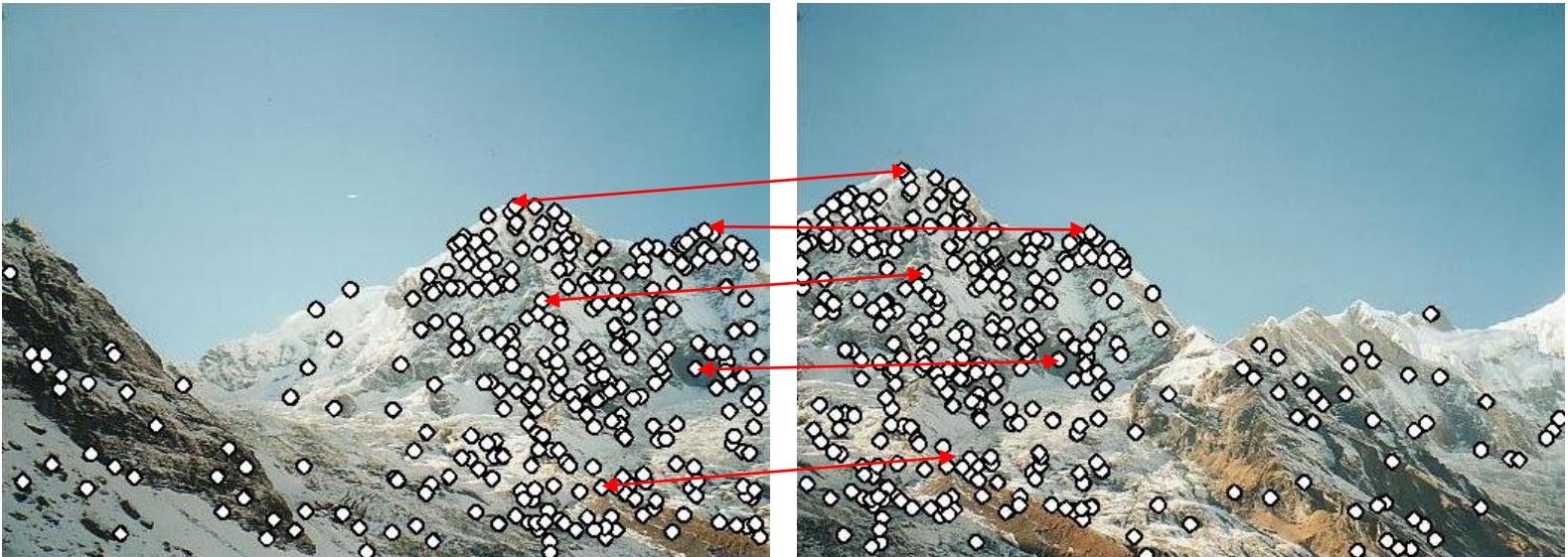


# Application: Image Stitching



- Procedure:
  - Detect feature points in both images

# Application: Image Stitching



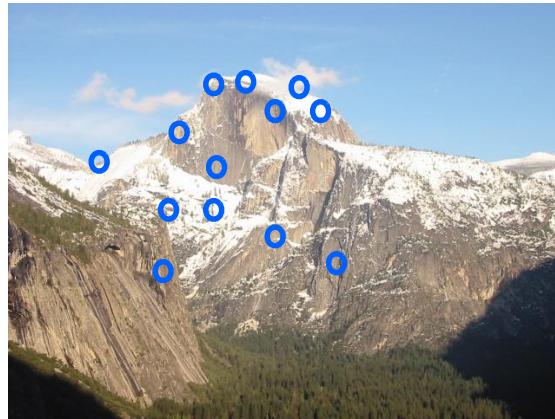
- Procedure:
  - Detect feature points in both images
  - Find corresponding pairs

# Application: Image Stitching



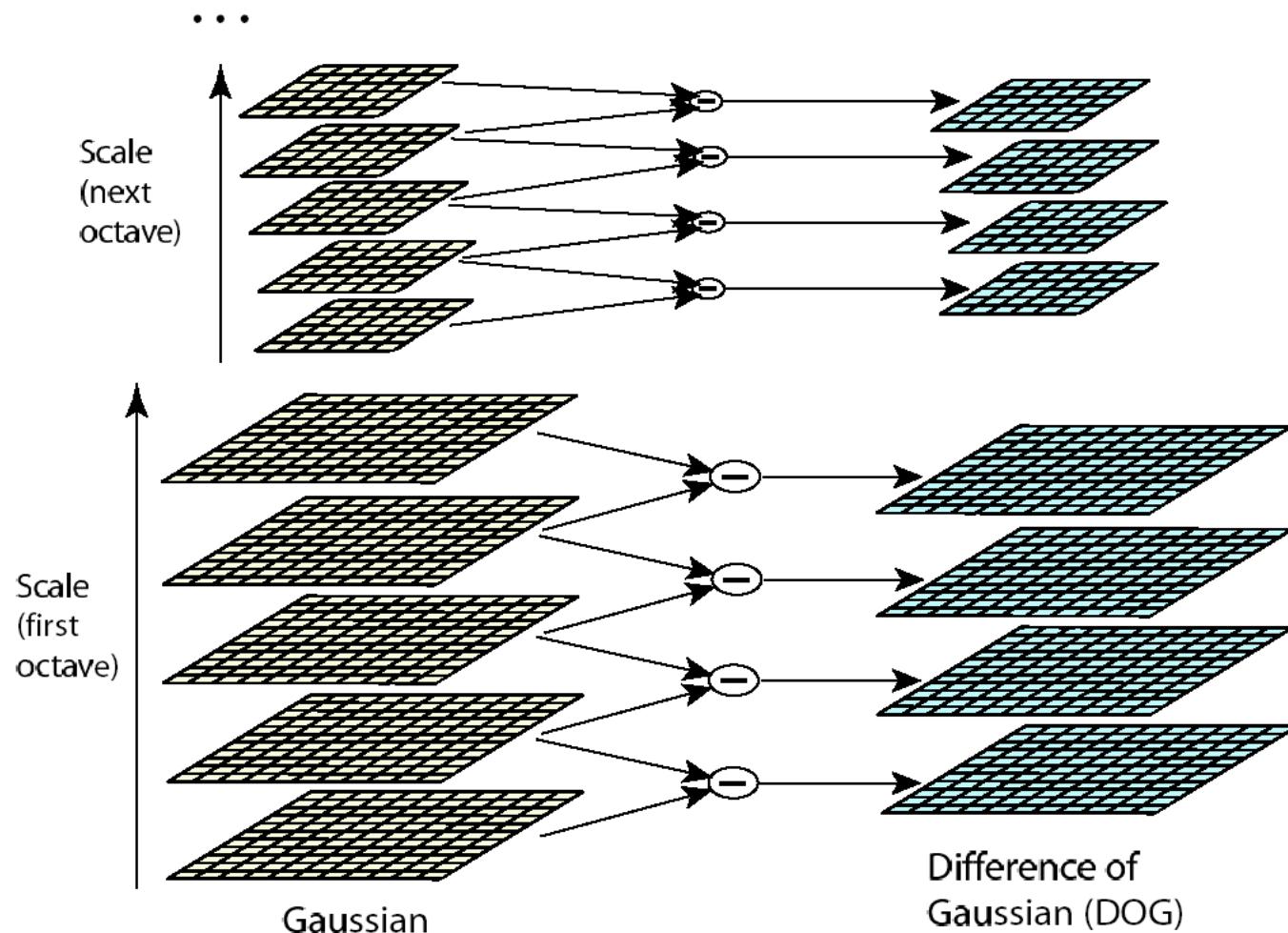
- Procedure:
  - Detect feature points in both images
  - Find corresponding pairs
  - Use these pairs to align the images

# Main Flow

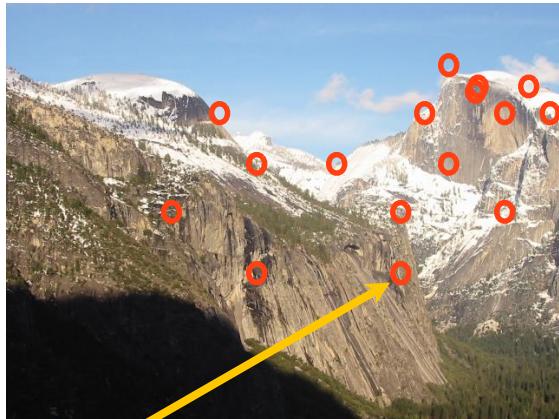


- Detect key points

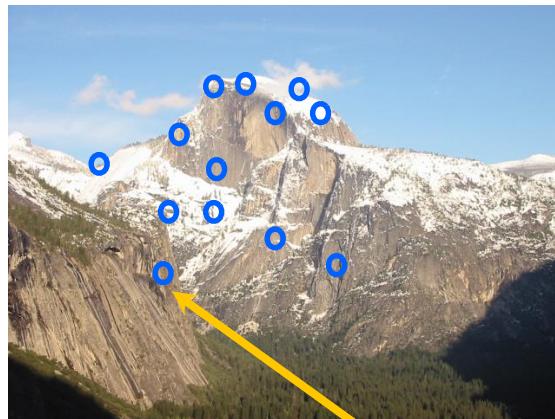
# Detect Key Points



# Main Flow



$(u_1, u_2, \dots, u_{128})$



$(v_1, v_2, \dots, v_{128})$

- Detect key points
- Build the SIFT descriptors

# Build the SIFT Descriptors

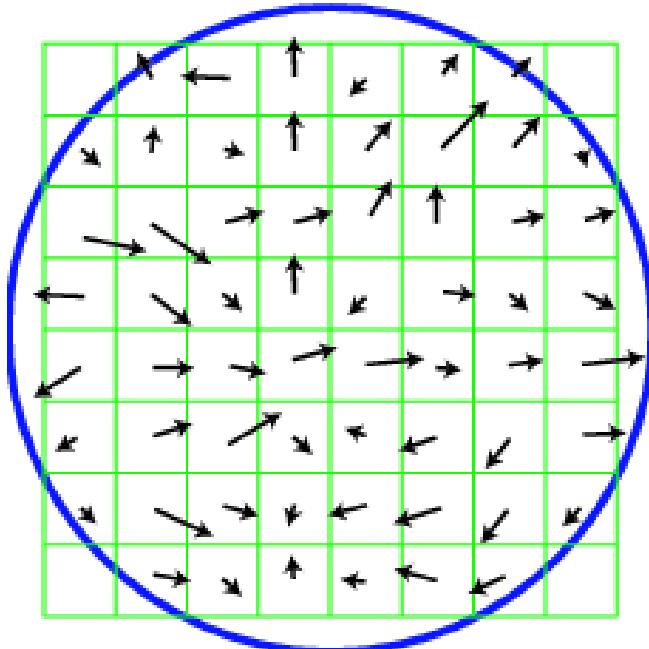
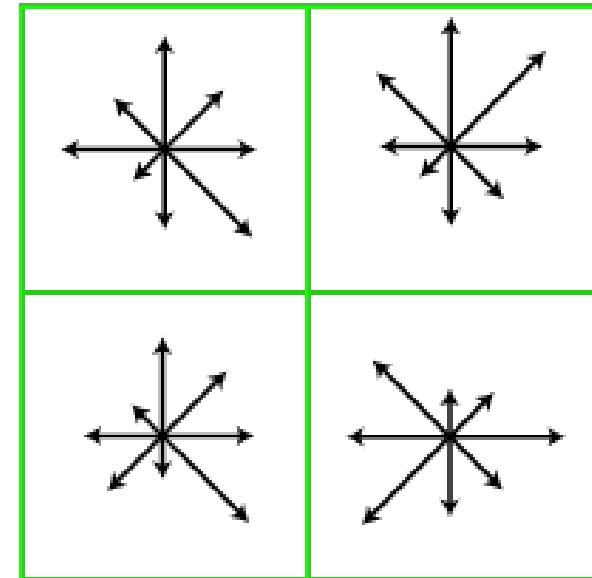
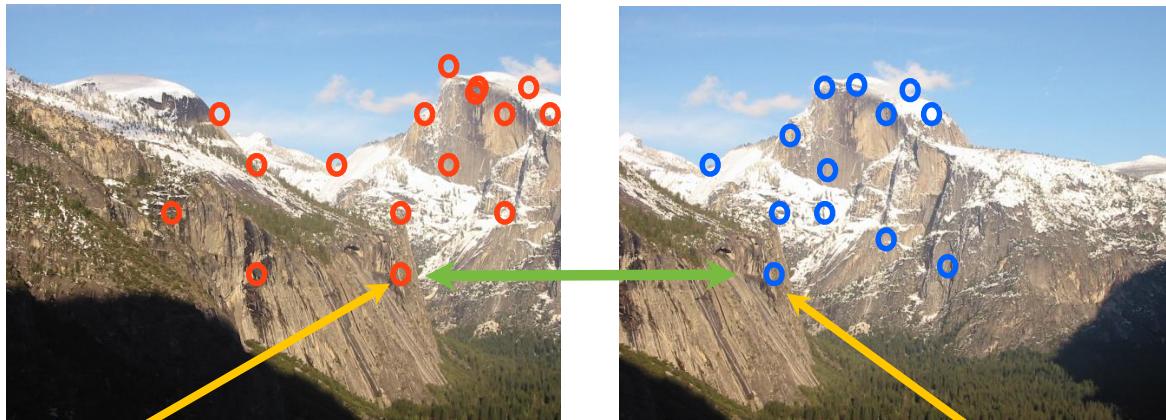


Image gradients



Keypoint descriptor

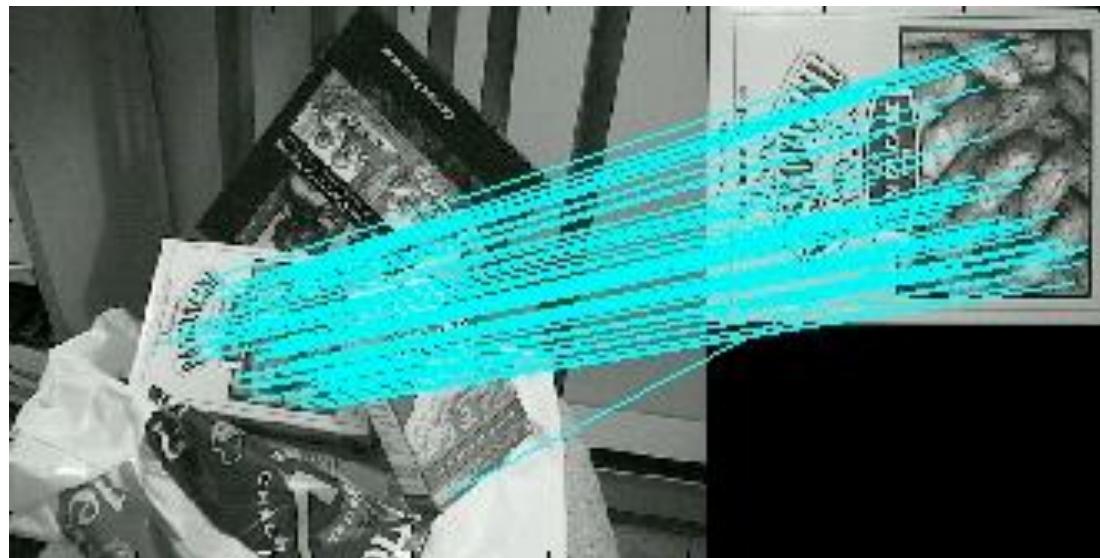
# Main Flow


$$(u_1, u_2, \dots, u_{128})$$
$$(v_1, v_2, \dots, v_{128})$$

- Detect key points
- Build the SIFT descriptors
- Match SIFT descriptors

# Match SIFT Descriptors

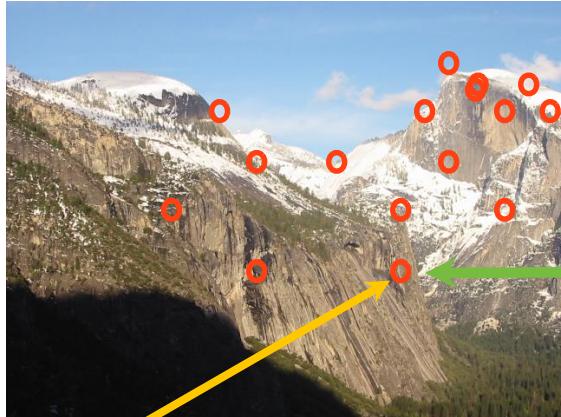
- Euclidean distance between descriptors



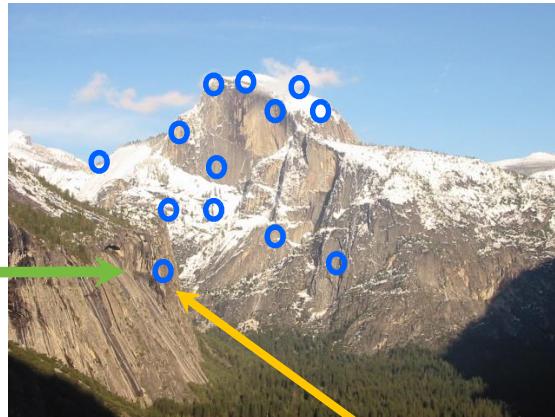
# Skeleton Code

- Match SIFT descriptors (6 lines of code)
  - Input: D1, D2, thresh (default 0.7)
  - Output: match [D1's index, D2's index]
  - Try to use *one* *for* loop

# Main Flow



$(u_1, u_2, \dots, u_{128})$



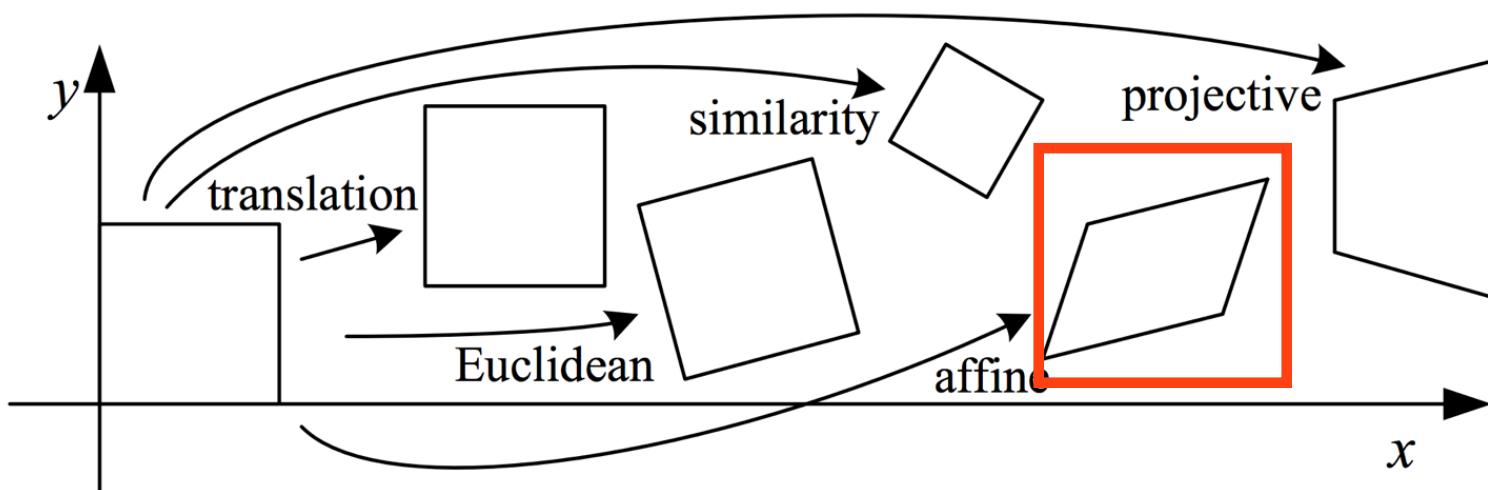
$(v_1, v_2, \dots, v_{128})$

- Detect key points
- Build the SIFT descriptors
- Match SIFT descriptors
- Fitting the transformation

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

# Fitting the transformation

- 2D transformations



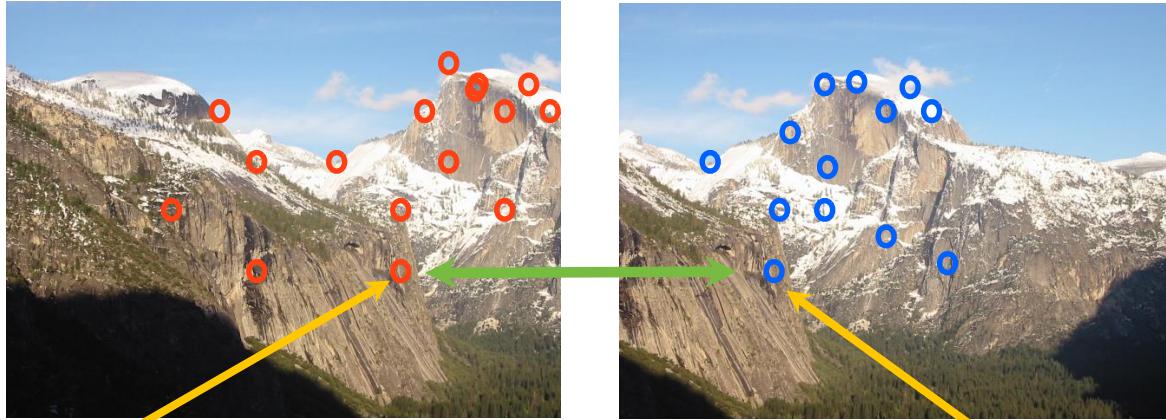
# Skeleton Code

- Fit the transformation matrix

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

- Six variables
  - each point give two equations
  - at least three points
- Least squares

# Main Flow



$(u_1, u_2, \dots, u_{128})$

- Detect key points
- Build the SIFT descriptors
- Match SIFT descriptors
- Fitting the transformation
- RANSAC

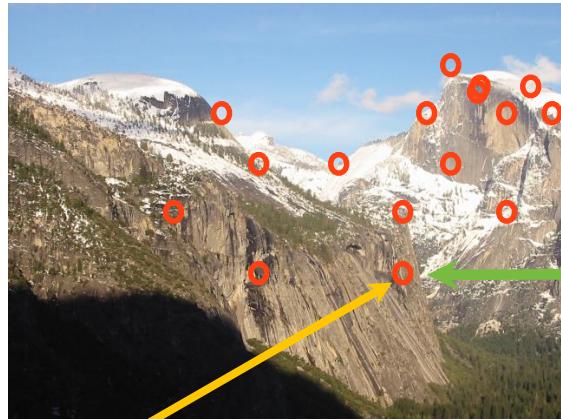
$(v_1, v_2, \dots, v_{128})$

# Skeleton Code

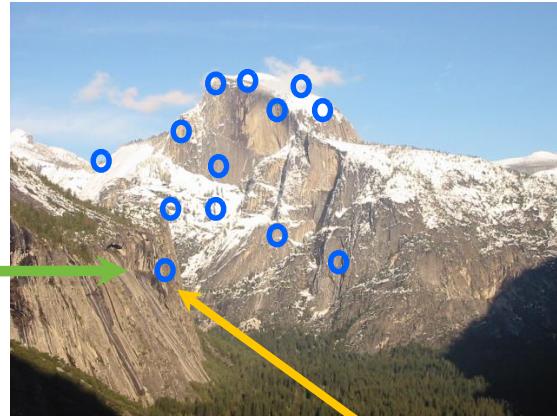
- RANSAC
  - ComputeError

$$\left\| \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} - H \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \right\|_2$$

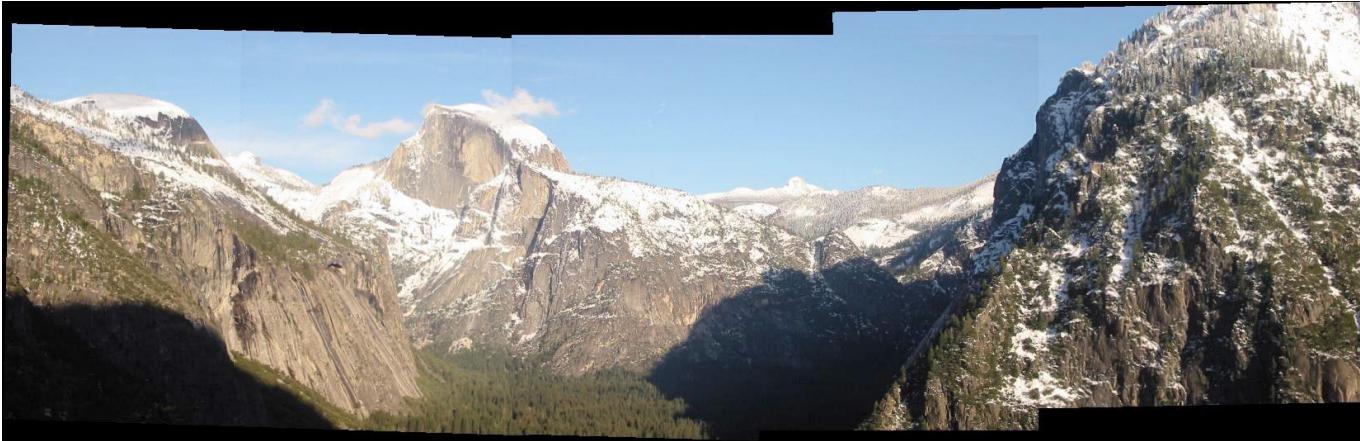
# Main Flow


$$(u_1, u_2, \dots, u_{128})$$

- Detect key points
- Build the SIFT descriptors
- Match SIFT descriptors
- Fitting the transformation
- RANSAC


$$(v_1, v_2, \dots, v_{128})$$


# Results



# Results

