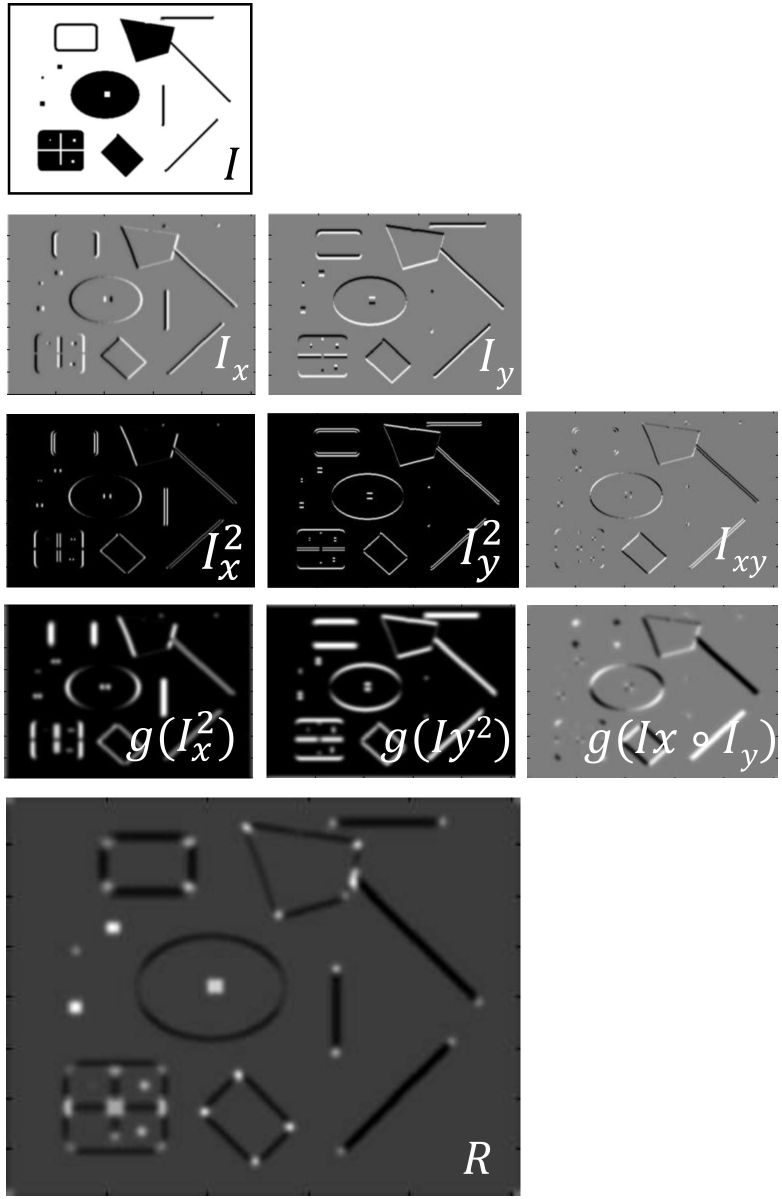
Keypoint Detection
Feature Descriptor

Harris Corner Detector [Harris88]

James Hays



0. Input image
   We want to compute M at each pixel.

1. Compute image derivatives (optionally, blur first).

2. Compute M components
   as squares of derivatives.

3. Gaussian filter *g()* with width σ

4. Compute cornerness

$$C = \det(M) - \alpha \operatorname{trace}(M)^2$$
$$= g(I_x^2) \circ g(I_y^2) - g(I_x \circ I_y)^2$$
$$-\alpha\left[g(I_x^2) + g(I_y^2)\right]^2$$

5. Threshold on C to pick high cornerness

6. Non-maxima suppression to pick peaks.

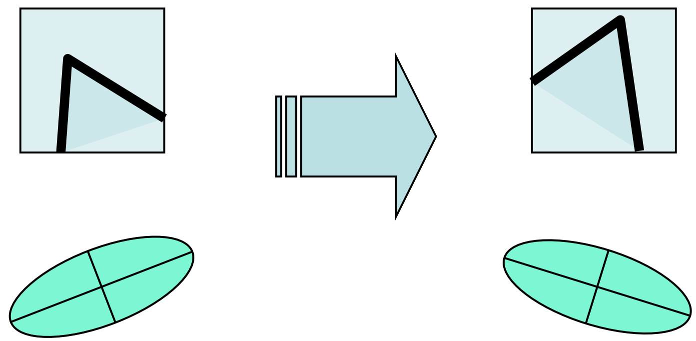
Harris Detector: Properties

- Translation invariance?

Slide credit: Kristen Grauman

Harris Detector: Properties

- Translation invariance
- Rotation invariance?



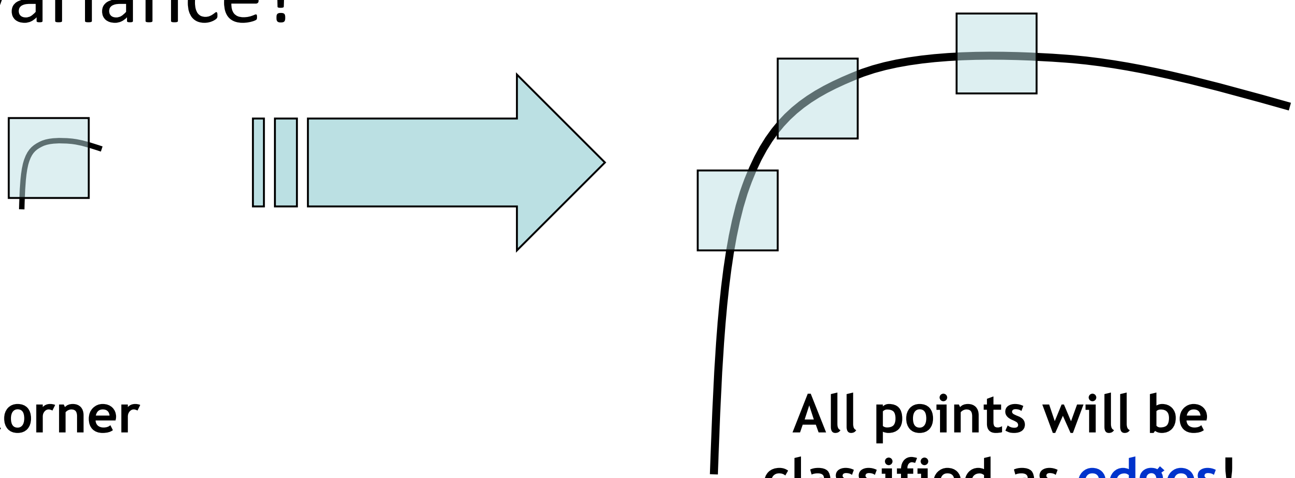
Ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner response θ **is invariant to image rotation**

Slide credit: Kristen Grauman

Harris Detector: Properties

- Translation invariance
- Rotation invariance
- Scale invariance?



Corner

All points will be classified as edges!

***Not* invariant to image scale!**

Slide credit: Kristen Grauman

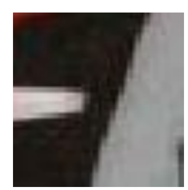
- How can we detect **scale invariant** interest points?

How to cope with transformations?

- Exhaustive search
- Invariance
- Robustness

Exhaustive search

- Multi-scale approach



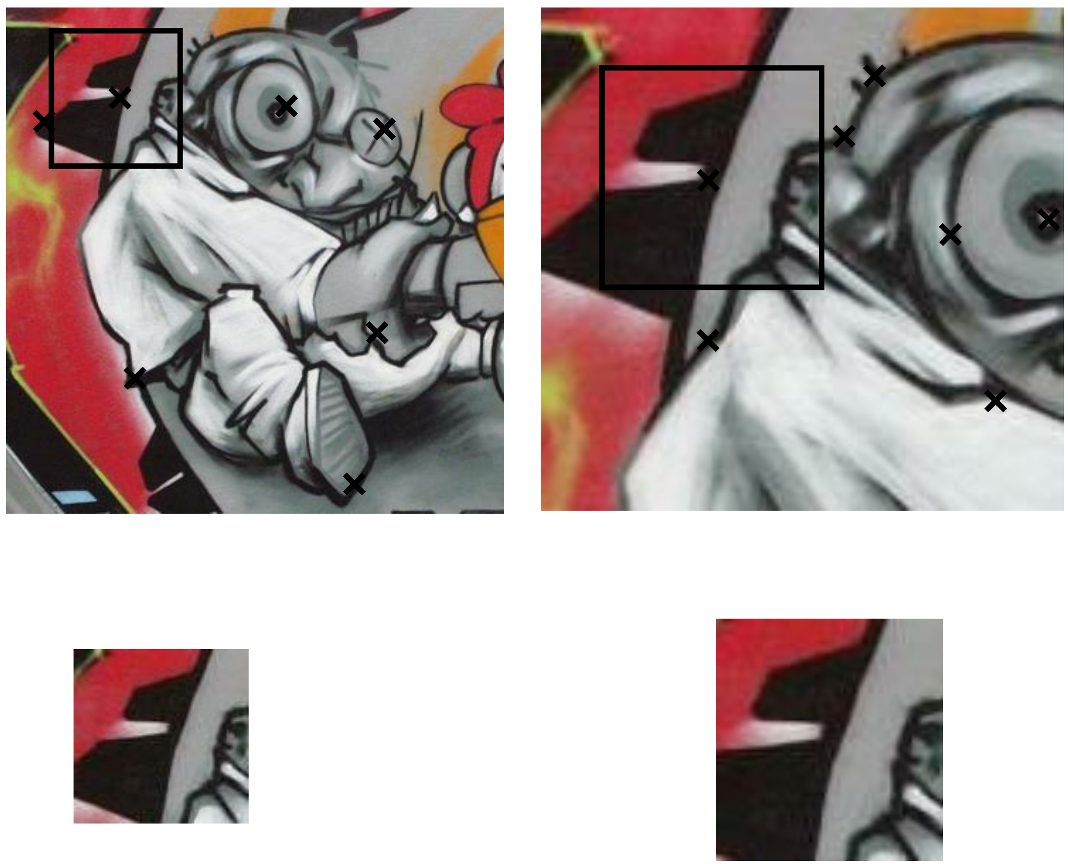
Slide from T. Tuytelaars ECCV 2006 tutorial

Exhaustive search

- Multi-scale approach

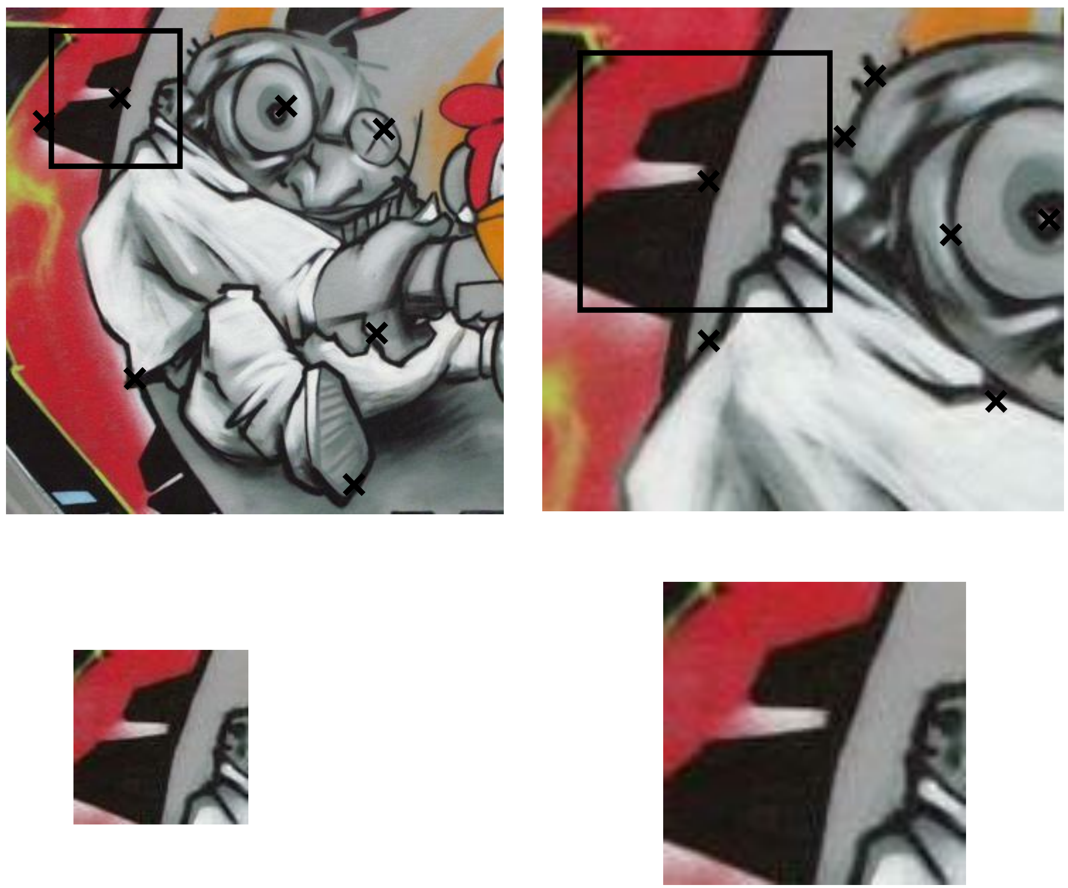
Exhaustive search

- Multi-scale approach



Exhaustive search

- Multi-scale approach



Invariance

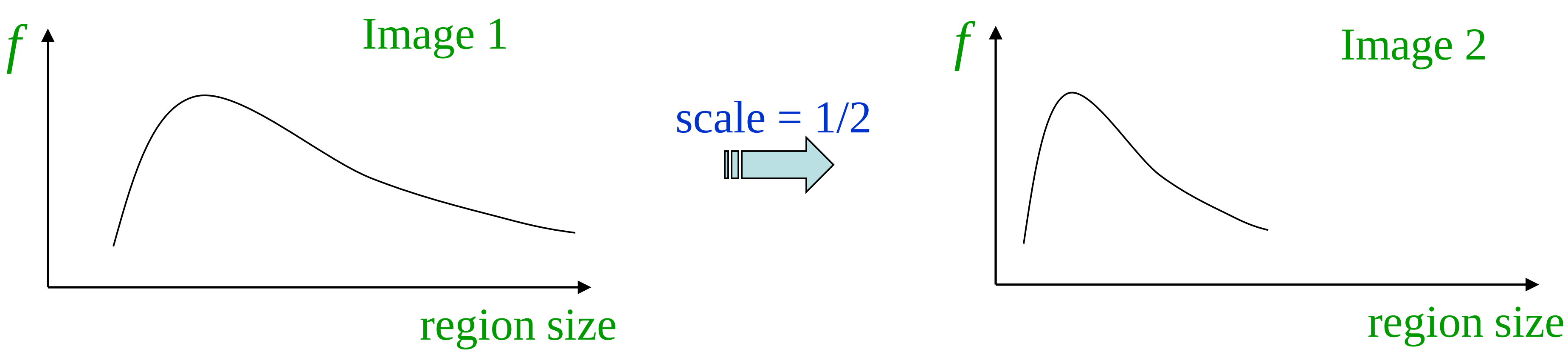
- Extract patch from each image individually

Automatic scale selection

- Solution:
  - Design a function on the region, which is “scale invariant” (*the same for corresponding regions, even if they are at different scales*)

    Example: average intensity. For corresponding regions (even of different sizes) it will be the same.

  - For a point in one image, we can consider it as a function of region size (patch width)



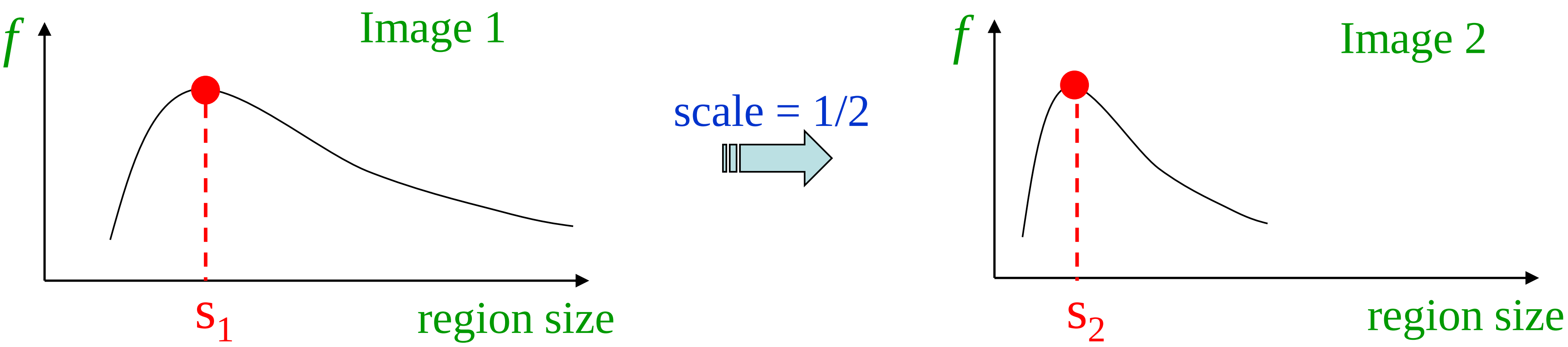
Automatic scale selection

- Common approach:

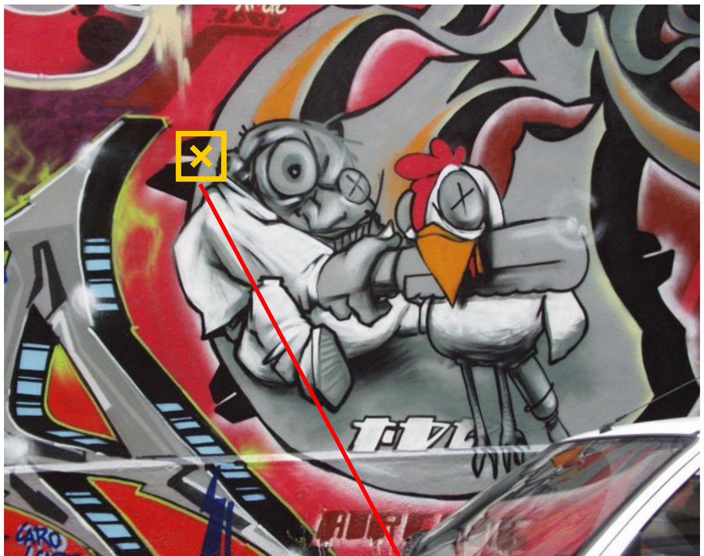
  Take a local maximum of this function

  Observation: region size, for which the maximum is achieved, should be *invariant* to image scale.

Important: this scale invariant region size is found in each image independently!



Automatic Scale Selection

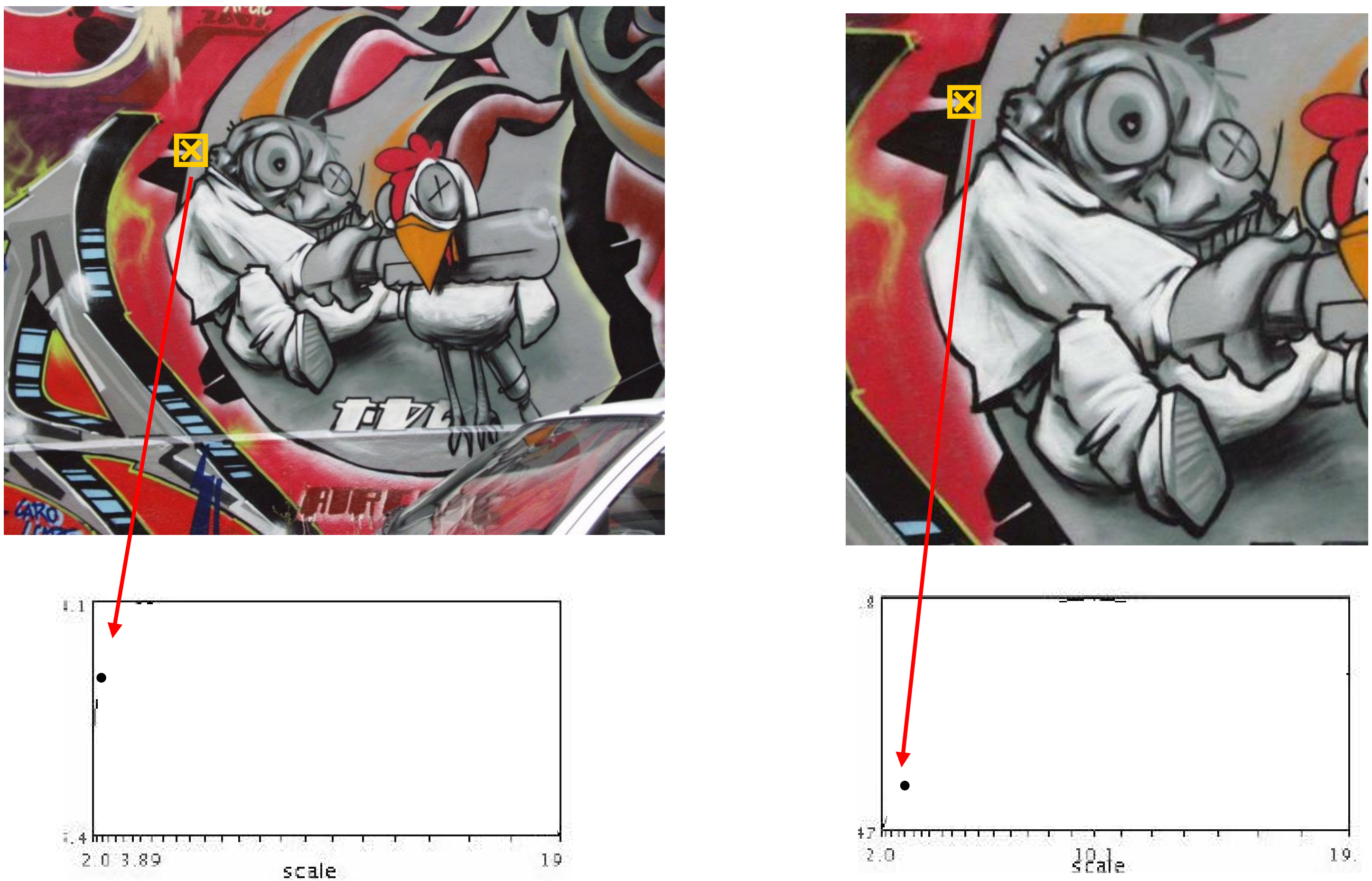


$$f(I_{i_1 \dots i_m}(x, \sigma)) = f(I_{i_1 \dots i_m}(x', \sigma'))$$

Same operator responses if the patch contains the same image up to scale factor.

Automatic Scale Selection

- **Function responses for increasing scale (scale signature)**

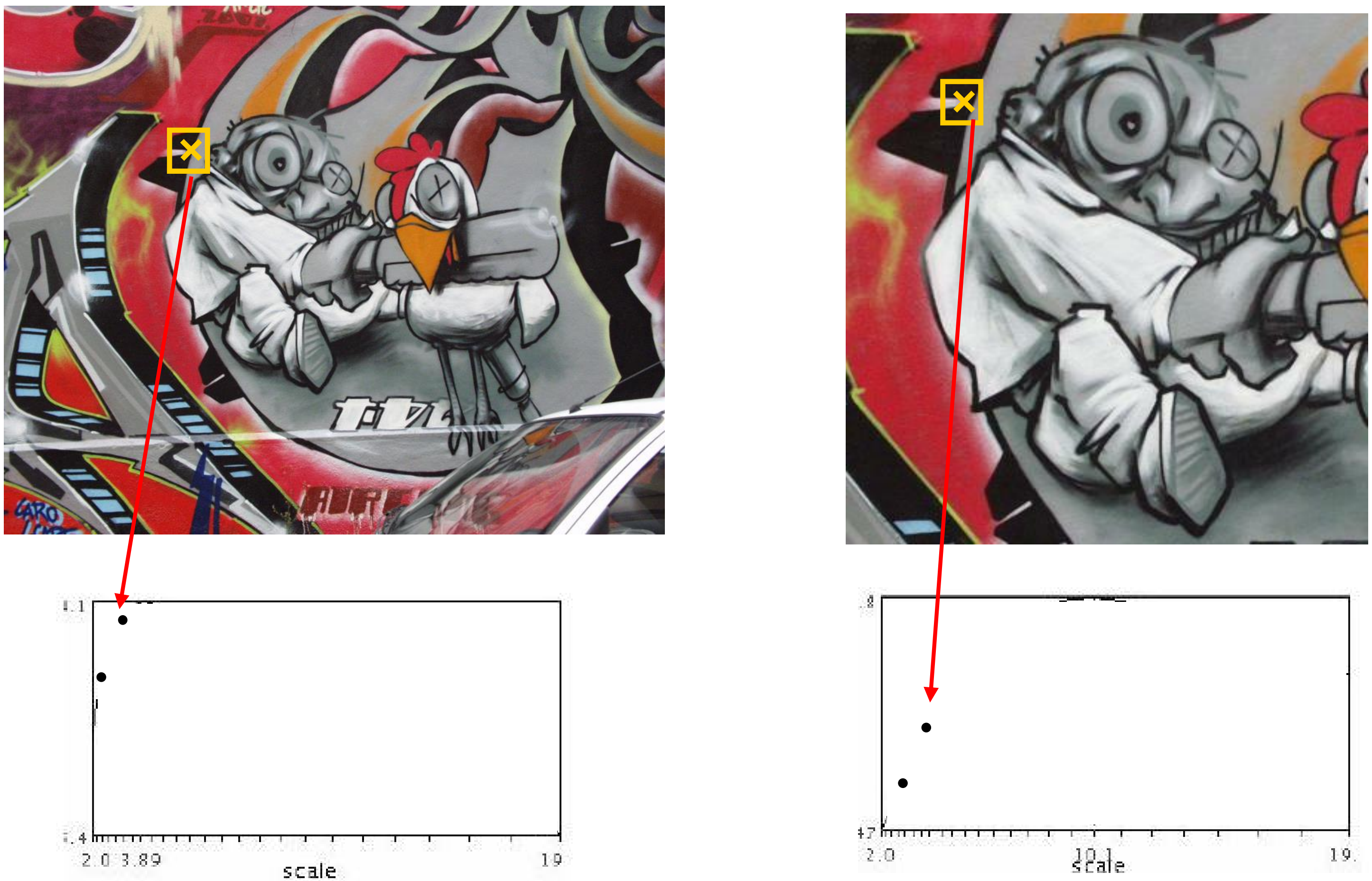


$$f(I_{i_1 \ldots i_m}(x, \sigma))$$

$$f(I_{i_1 \ldots i_m}(x', \sigma))$$

Automatic Scale Selection

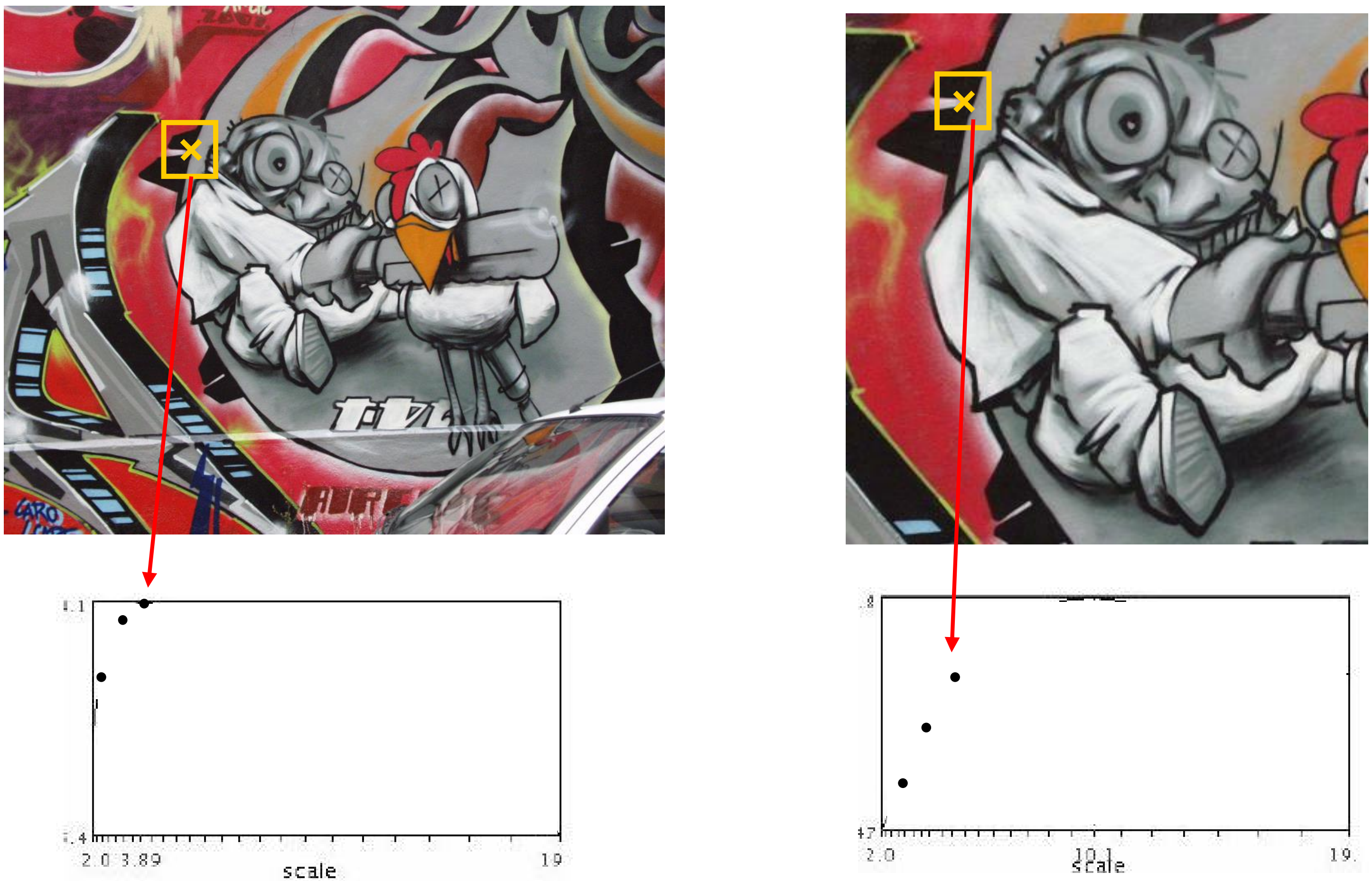
- **Function responses for increasing scale (scale signature)**



$f(I_{i_1 \ldots i_m}(x, \sigma))$ $\quad$ $f(I_{i_1 \ldots i_m}(x', \sigma))$

Automatic Scale Selection

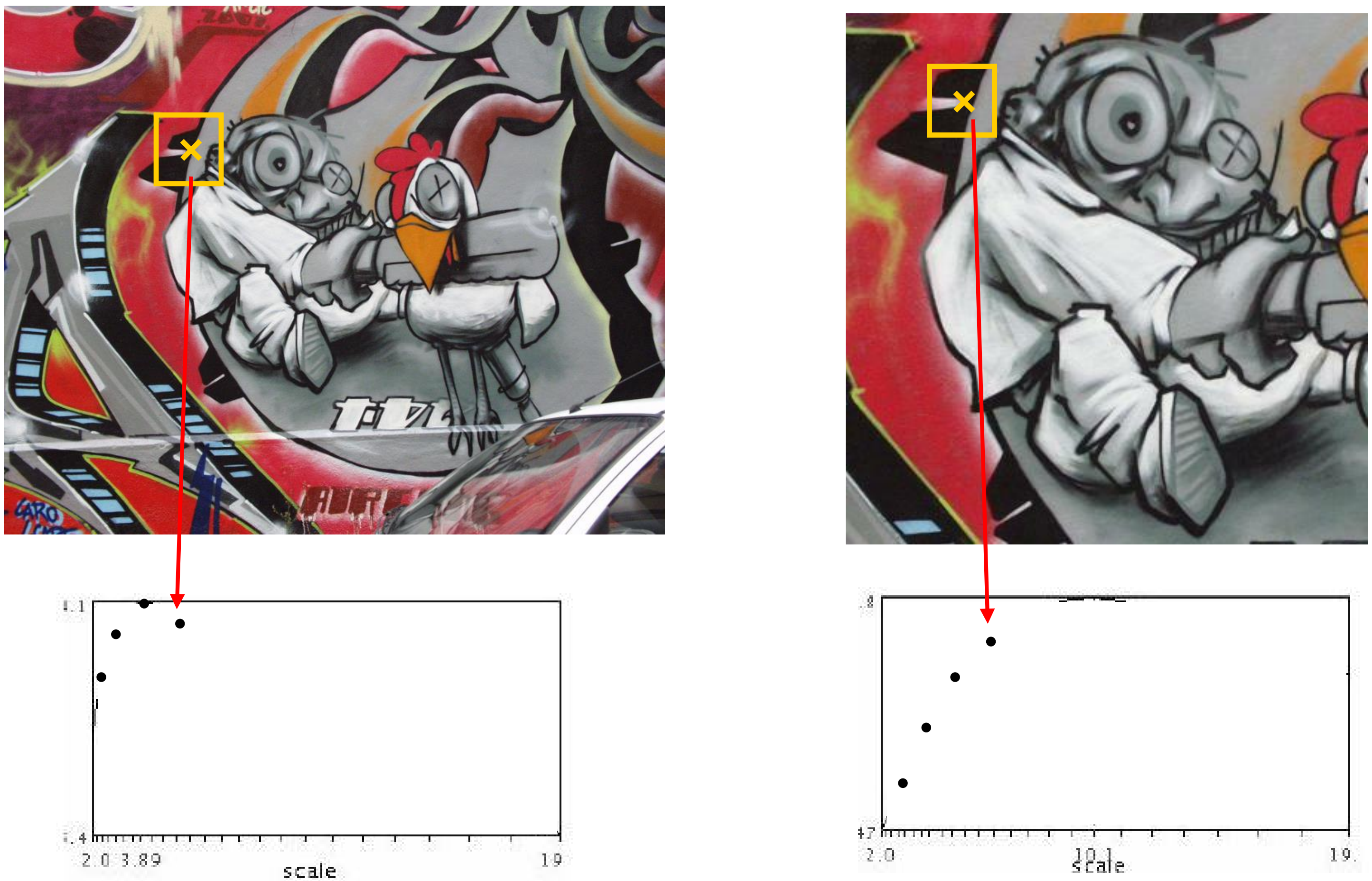
- **Function responses for increasing scale (scale signature)**



$f(I_{i_1 \ldots i_m}(x, \sigma))$ $\quad$ $f(I_{i_1 \ldots i_m}(x', \sigma))$

Automatic Scale Selection

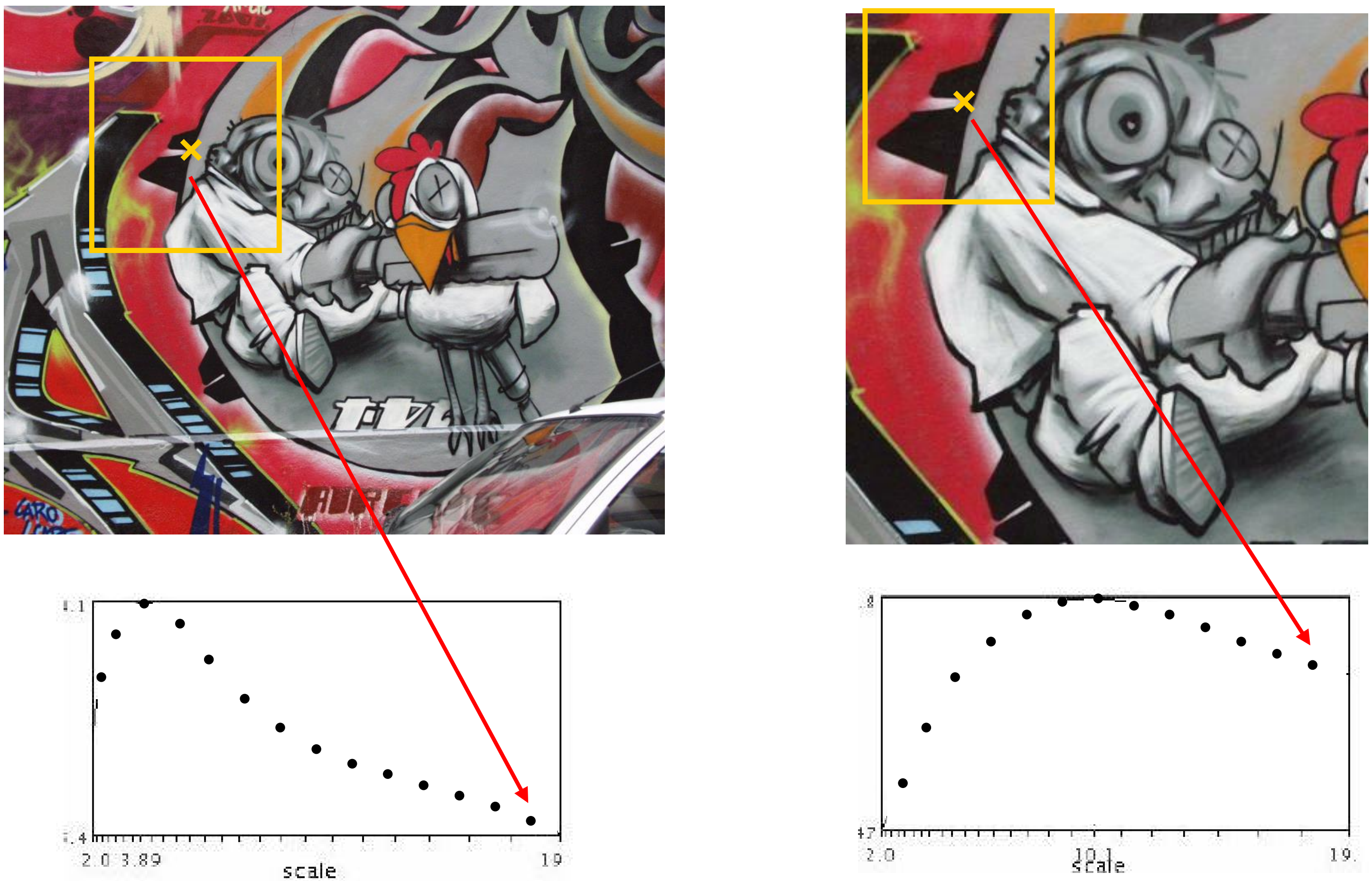
- **Function responses for increasing scale (scale signature)**



$f(I_{i_1 \ldots i_m}(x, \sigma))$ $f(I_{i_1 \ldots i_m}(x', \sigma))$

Automatic Scale Selection

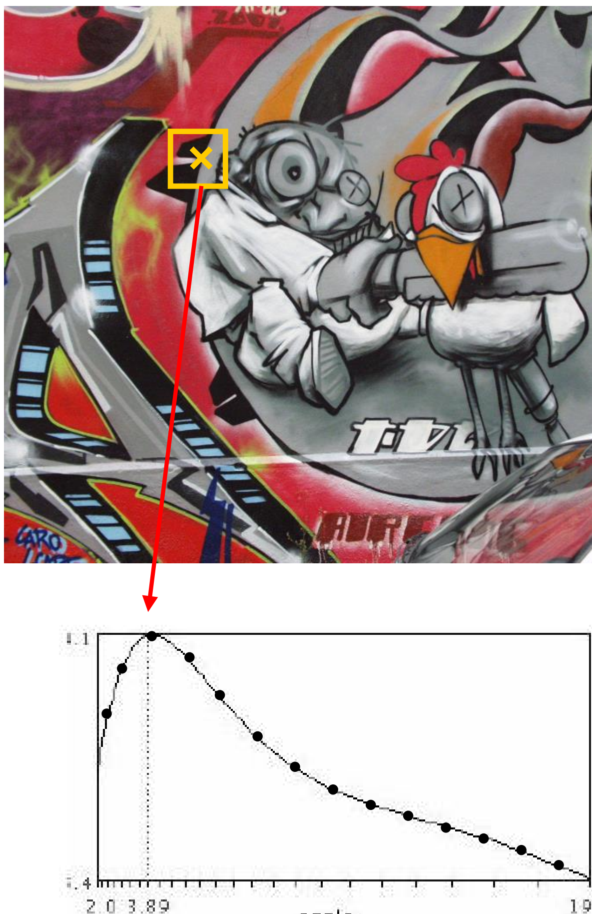
- **Function responses for increasing scale (scale signature)**



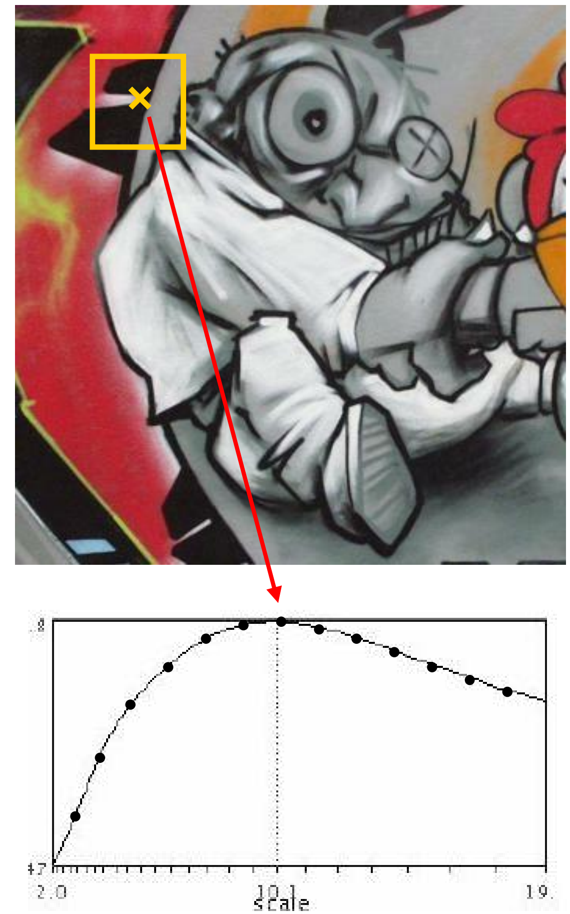
$f(I_{i_1 \ldots i_m}(x,\sigma))$ $f(I_{i_1 \ldots i_m}(x',\sigma))$

Automatic Scale Selection

- **Function responses for increasing scale (scale signature)**



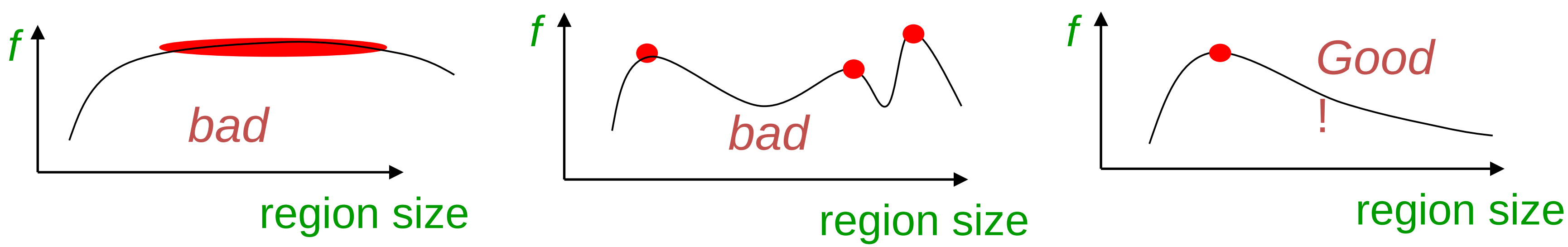
$f(I_{i_1 \dots i_m}(x, \sigma))$



$f(I_{i_1 \dots i_m}(x', \sigma'))$

Scale Invariant Detection

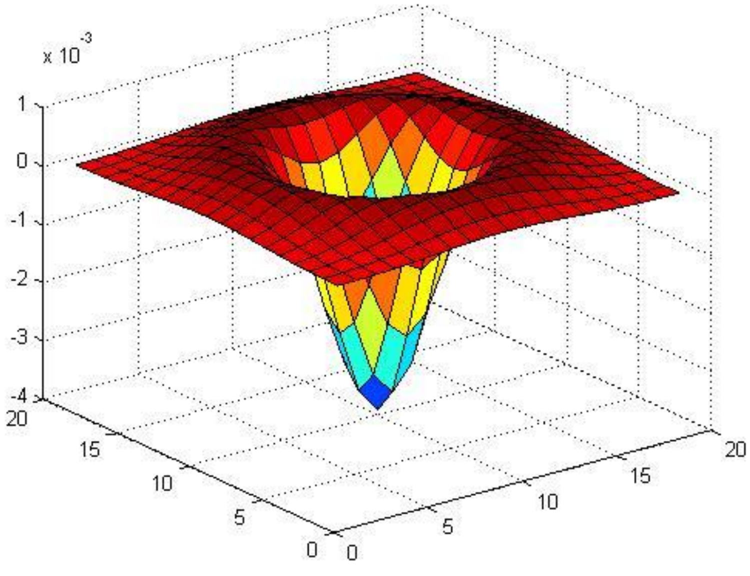
- A "good" function for scale detection:
  has one stable sharp peak

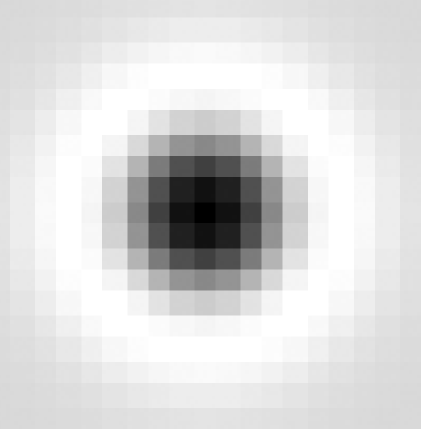


- For usual images: a good function would be one which responds to contrast (sharp local intensity change)

What Is A Useful Signature Function *f* ?

Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D

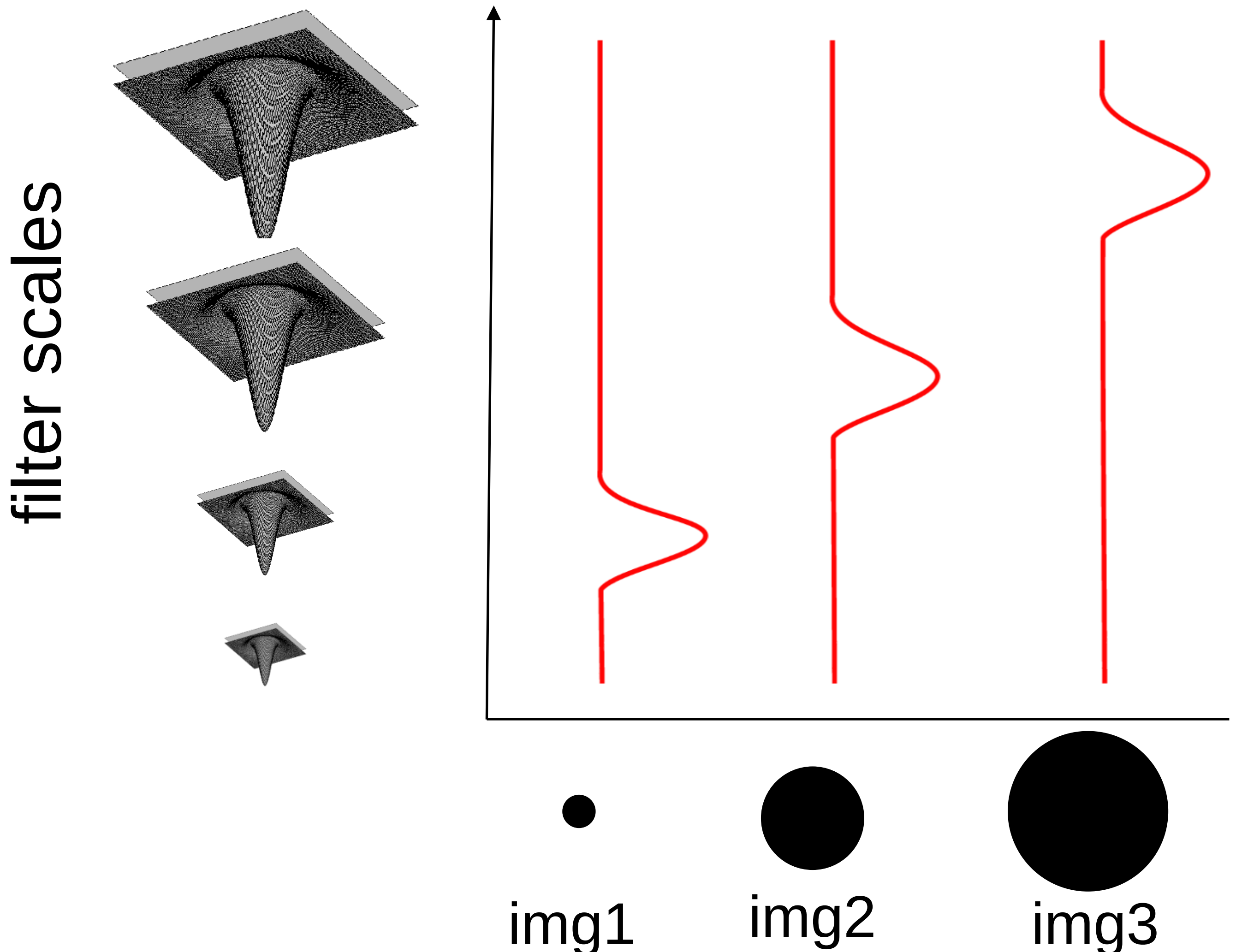




$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

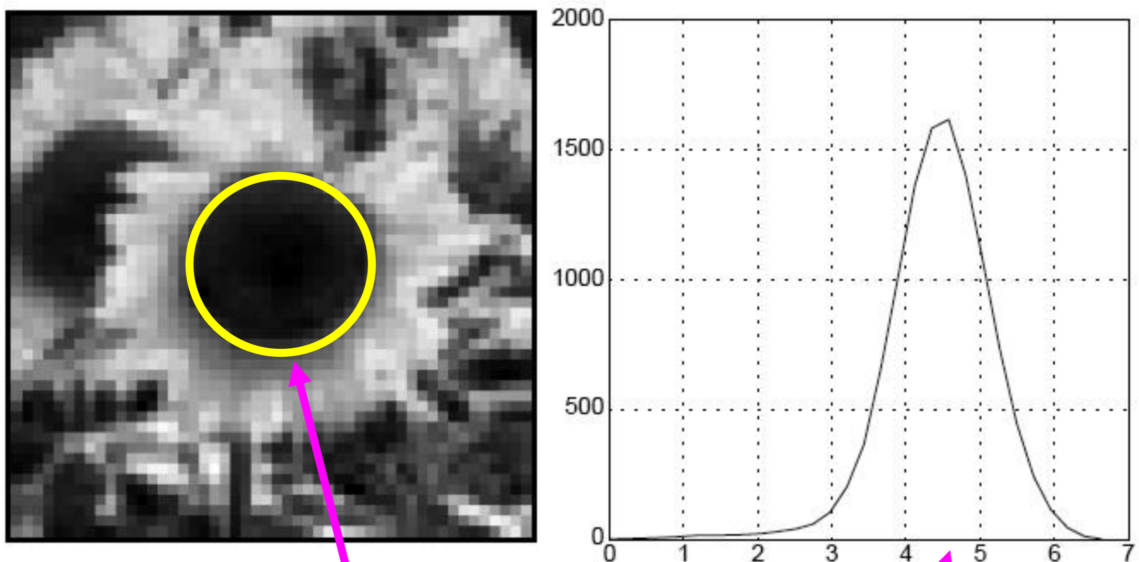
Blob detection in 2D: scale selection

Laplacian-of-Gaussian = "blob" detector $\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$



Blob detection in 2D

We define the *characteristic scale* as the scale that produces peak of Laplacian response



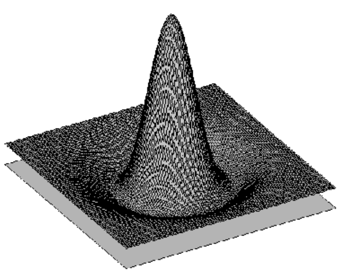
characteristic scale

Slide credit: Lana Lazebnik

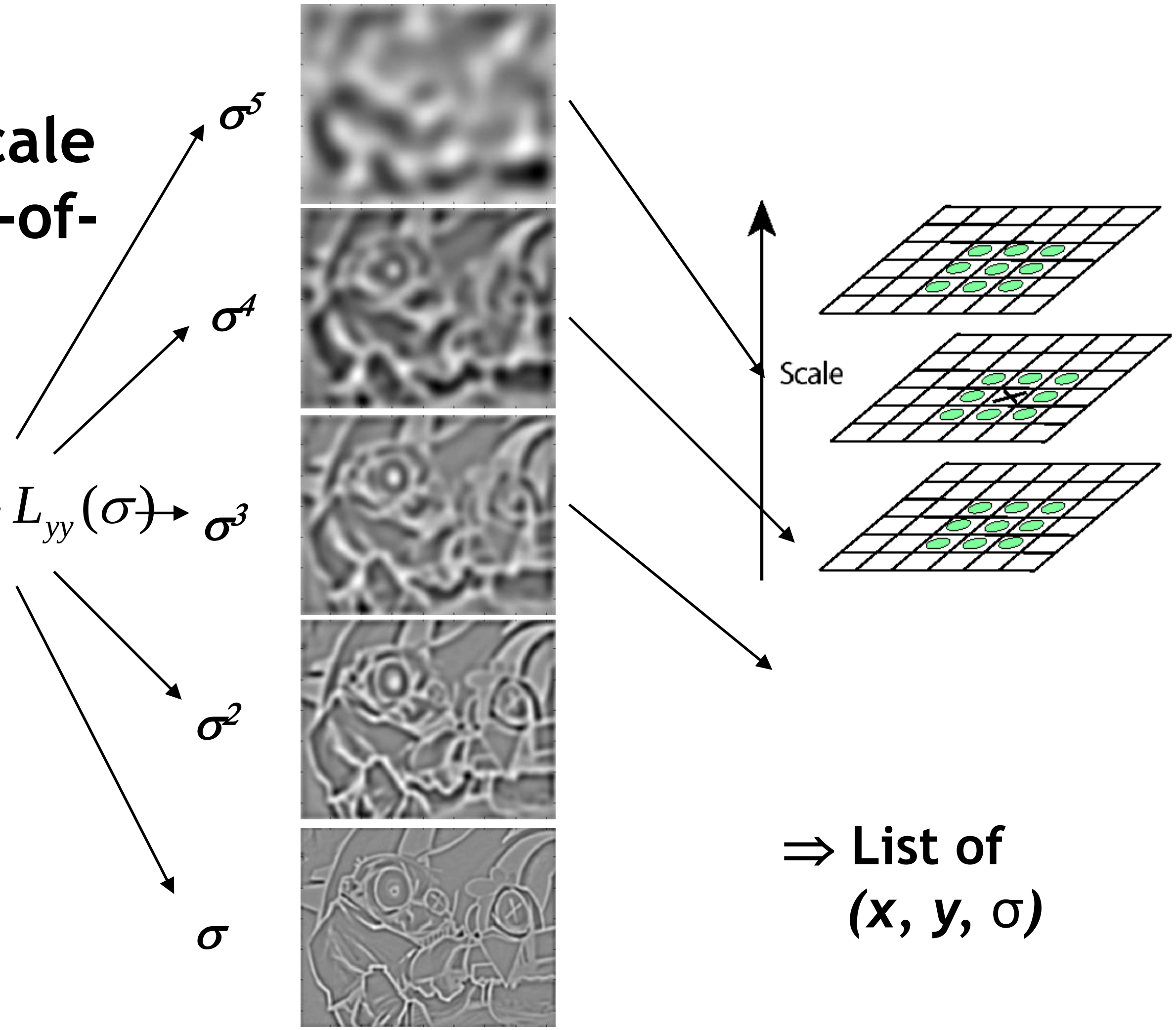
Laplacian-of-Gaussian (LoG)

- **Interest points:**

  Local maxima in scale space of Laplacian-of-Gaussian



$$L_{xx}(\sigma) + L_{yy}(\sigma)$$



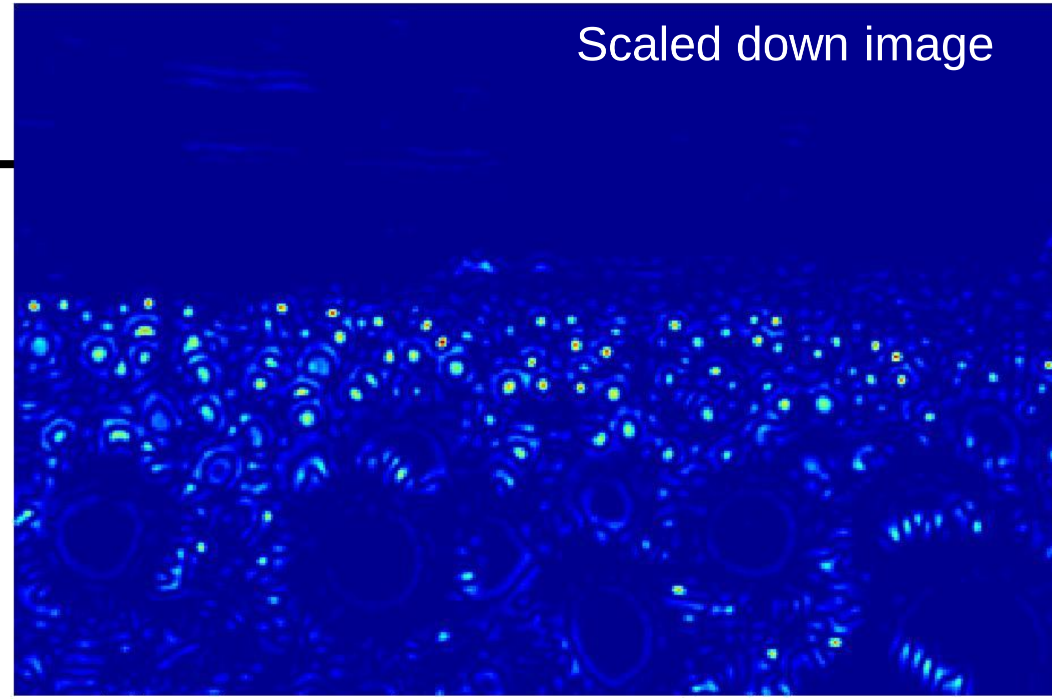
$\Rightarrow$ **List of (x, y, σ)**

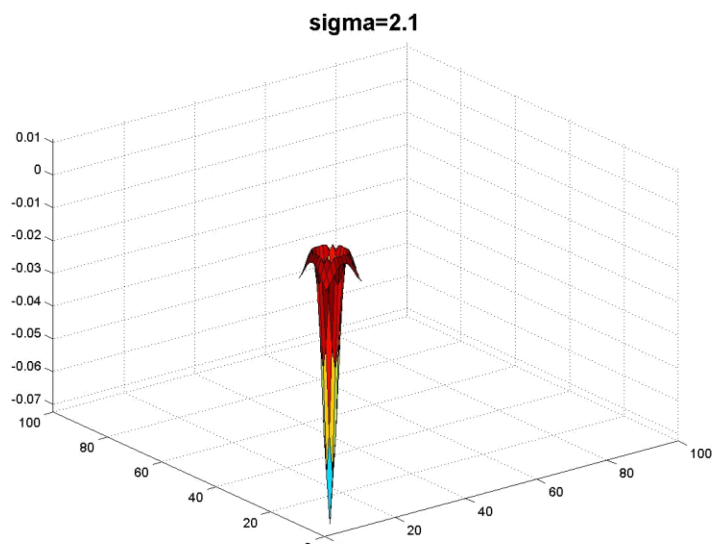
Example

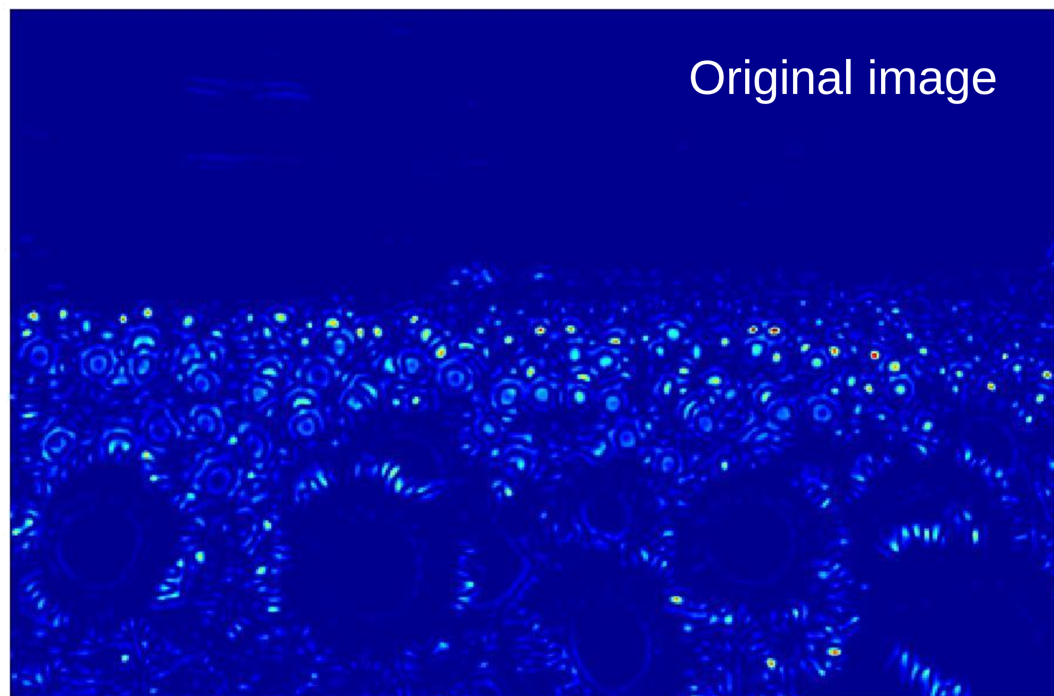
Original image
at ¾ the size

Slide credit: Kristen Grauman

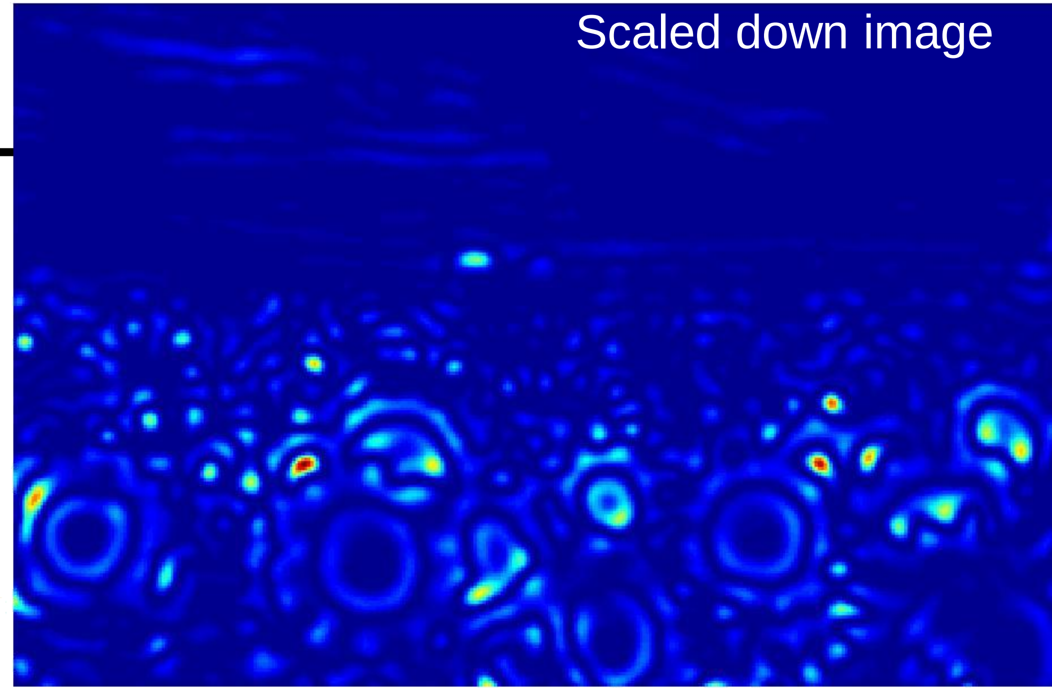
Original image
at ¾ the size

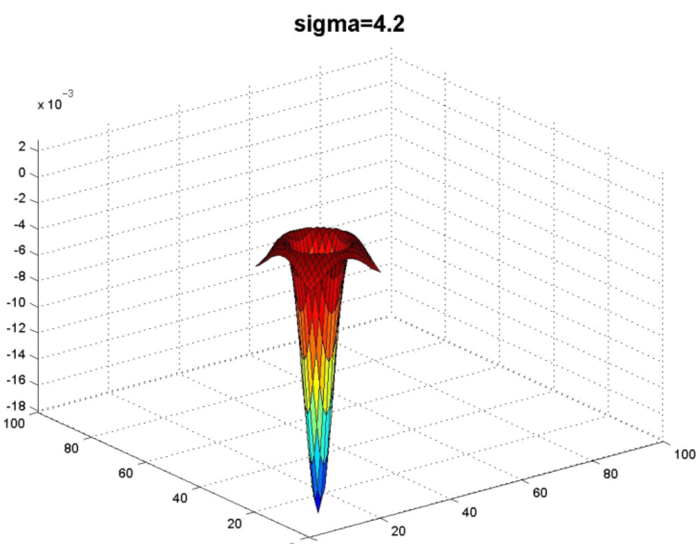


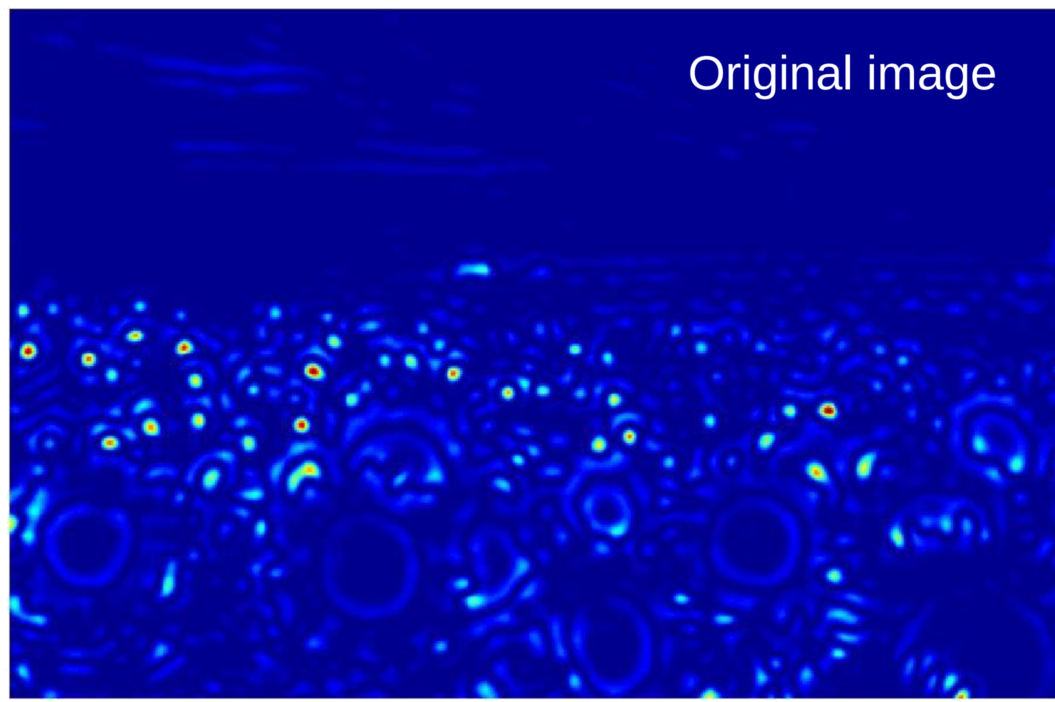




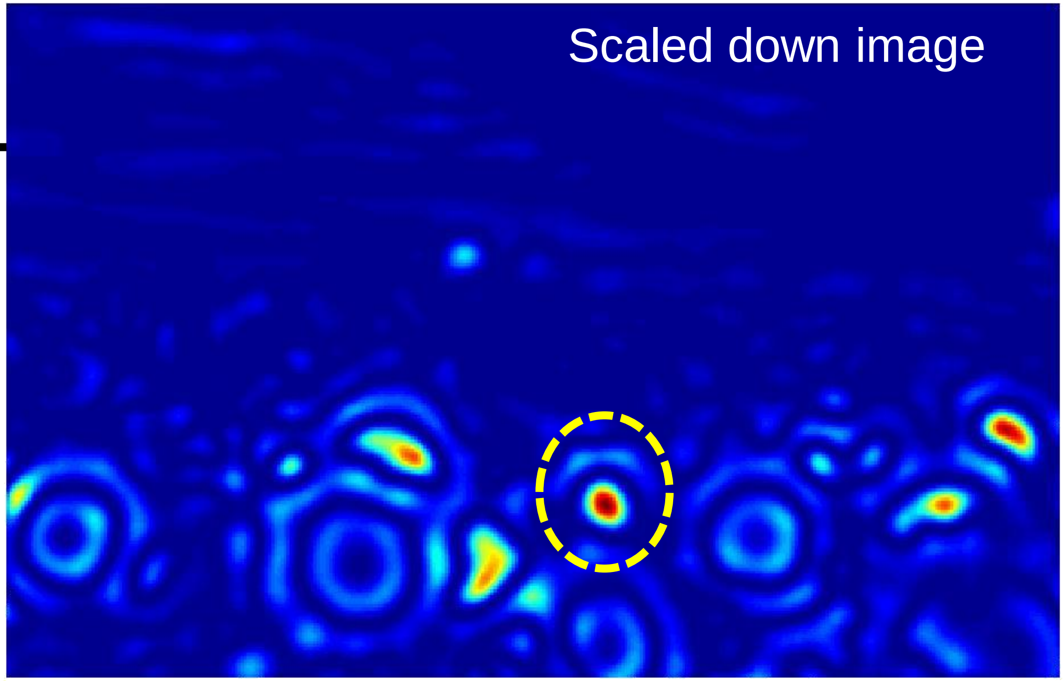
Slide credit: Kristen Grauman

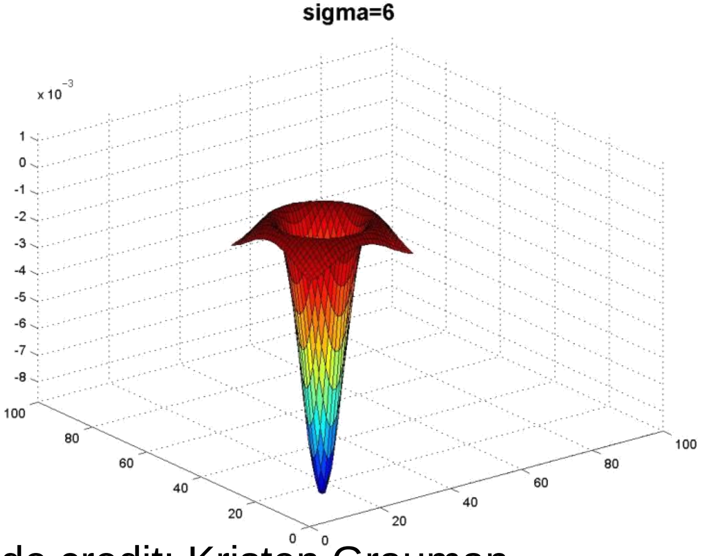


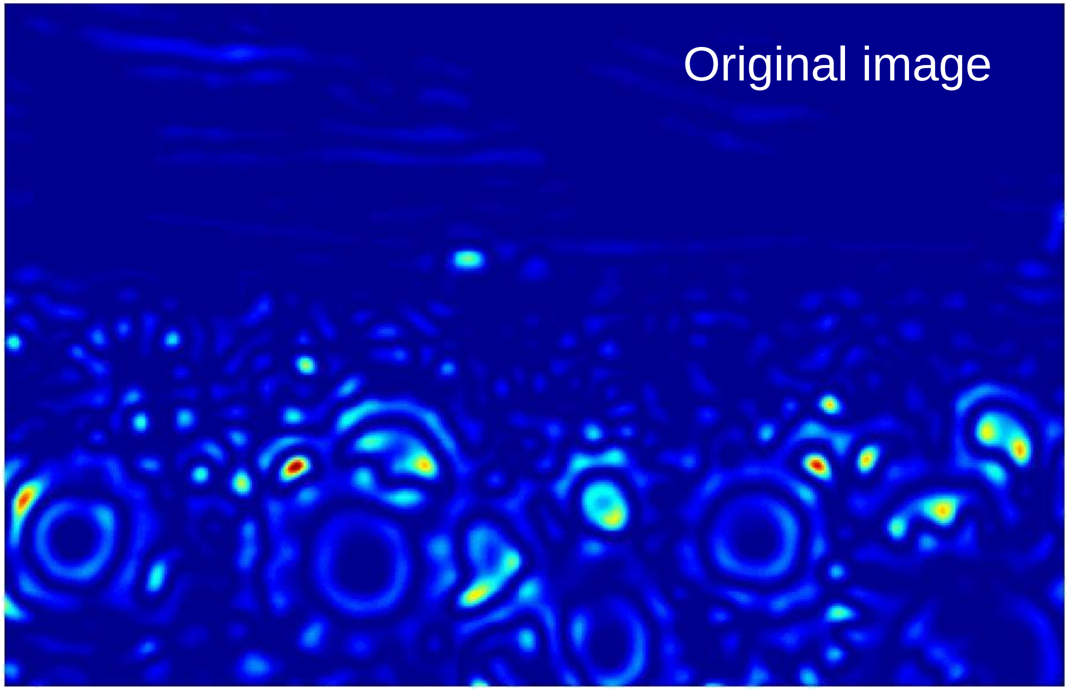




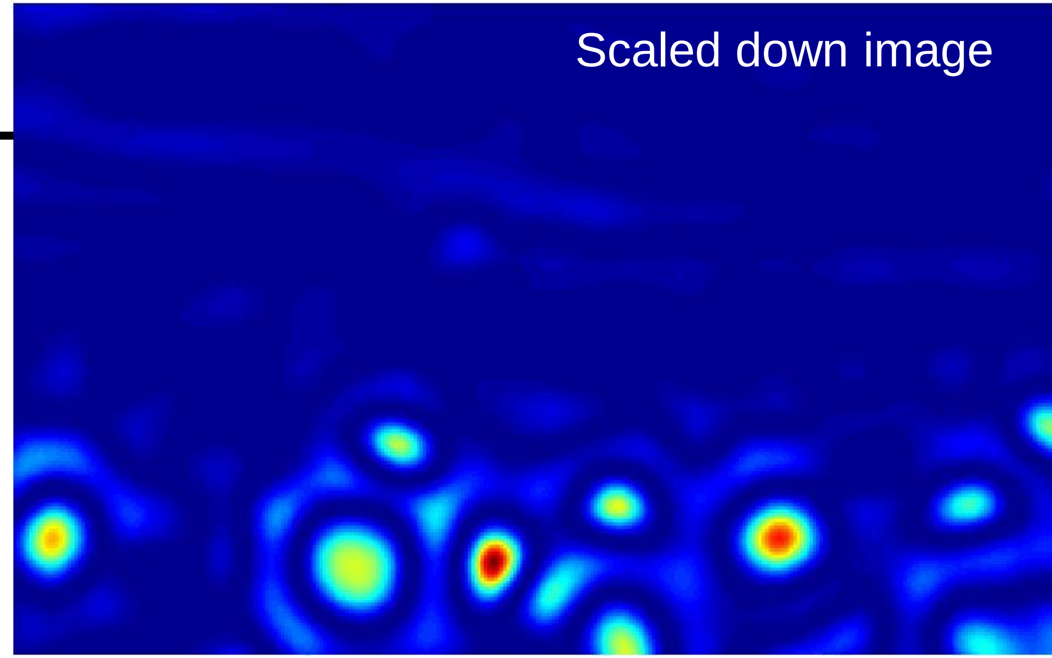
Slide credit: Kristen Grauman

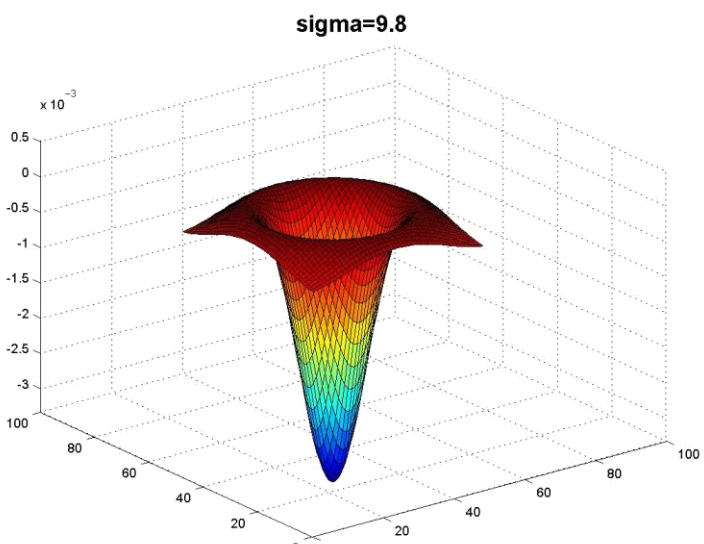


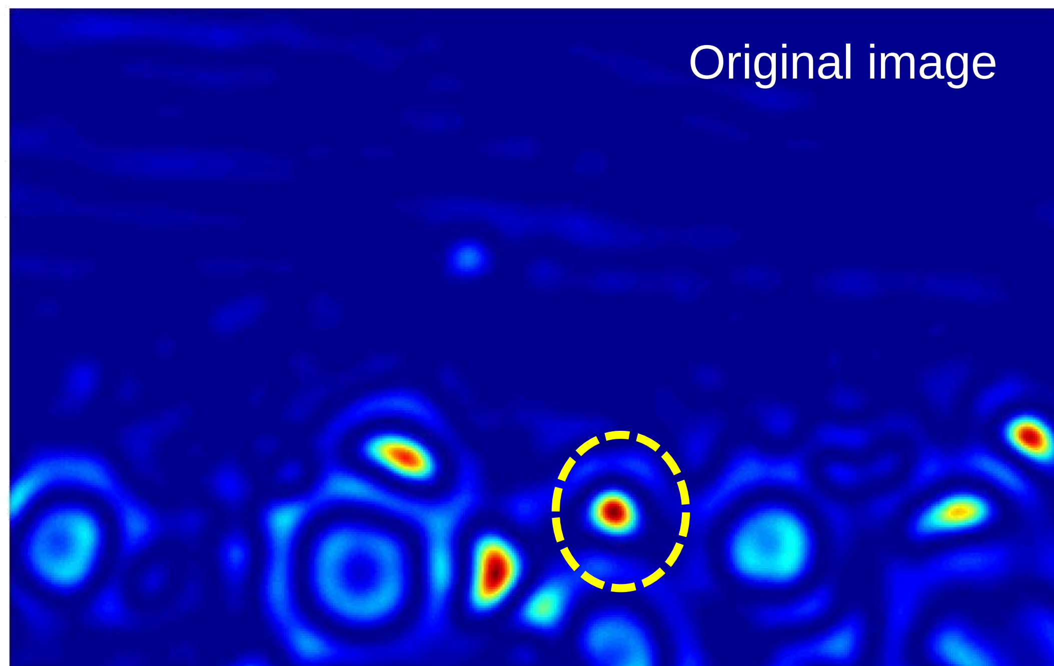




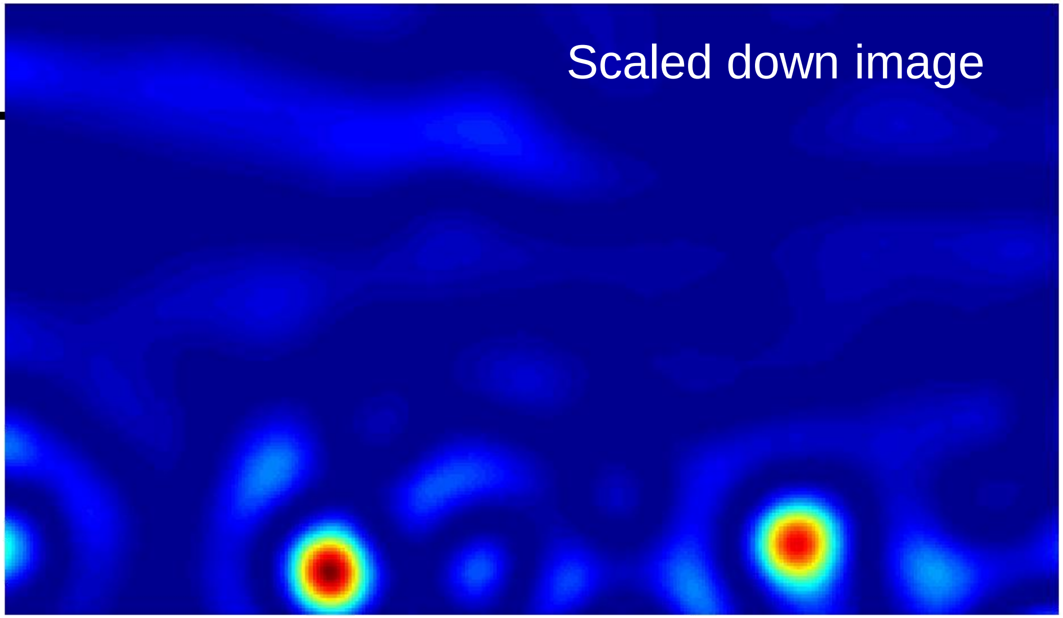
Slide credit: Kristen Grauman

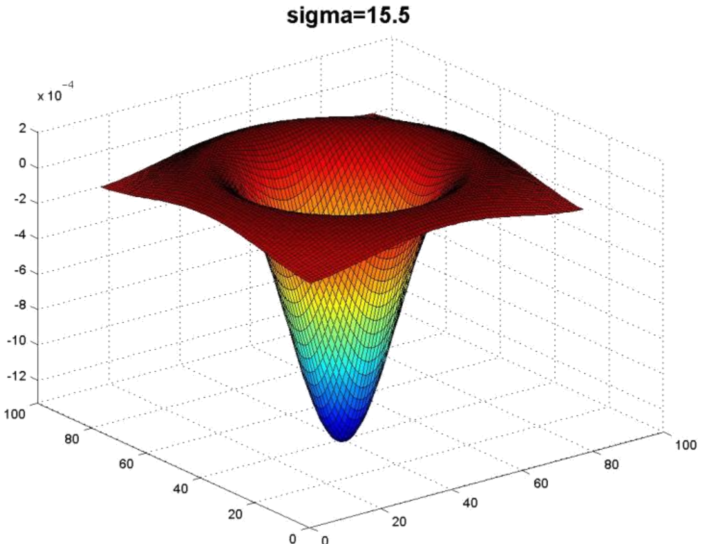


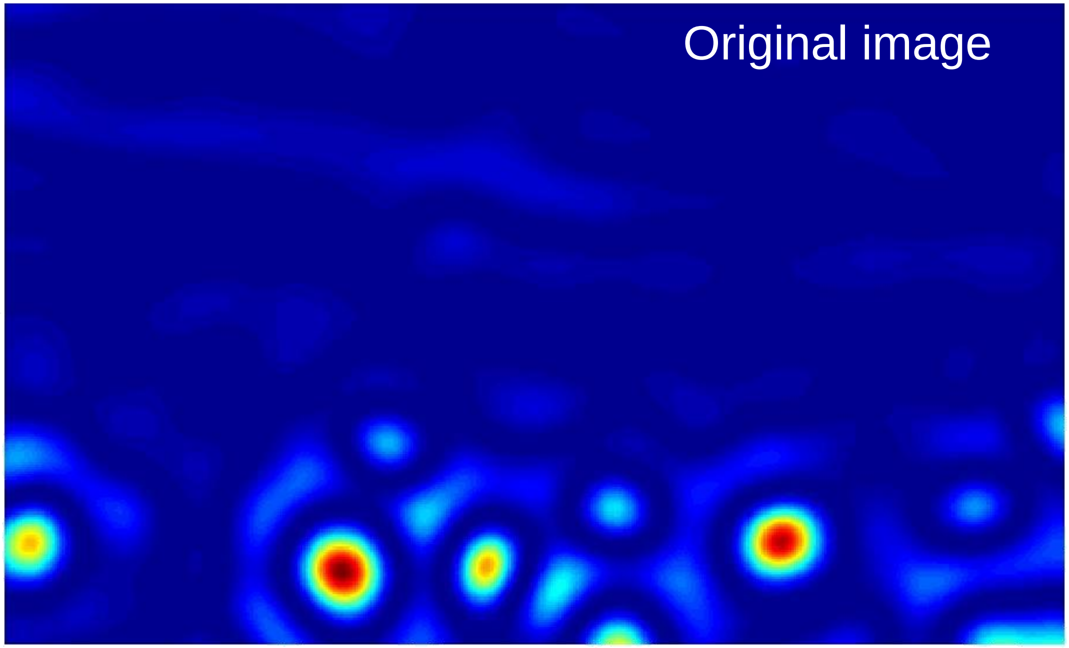




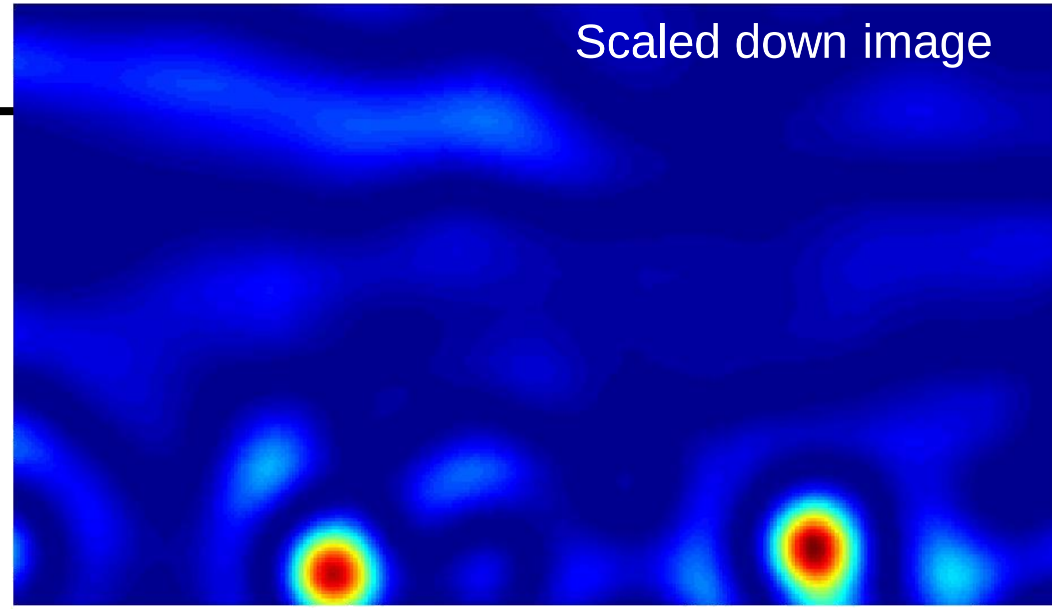
Slide credit: Kristen Grauman

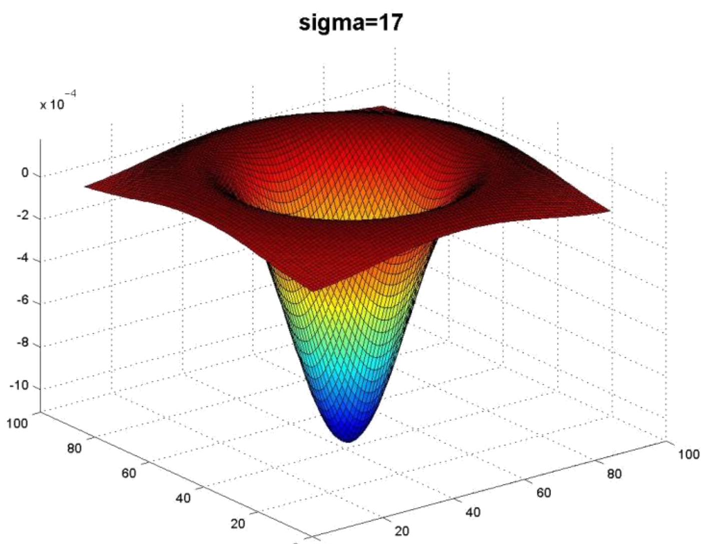


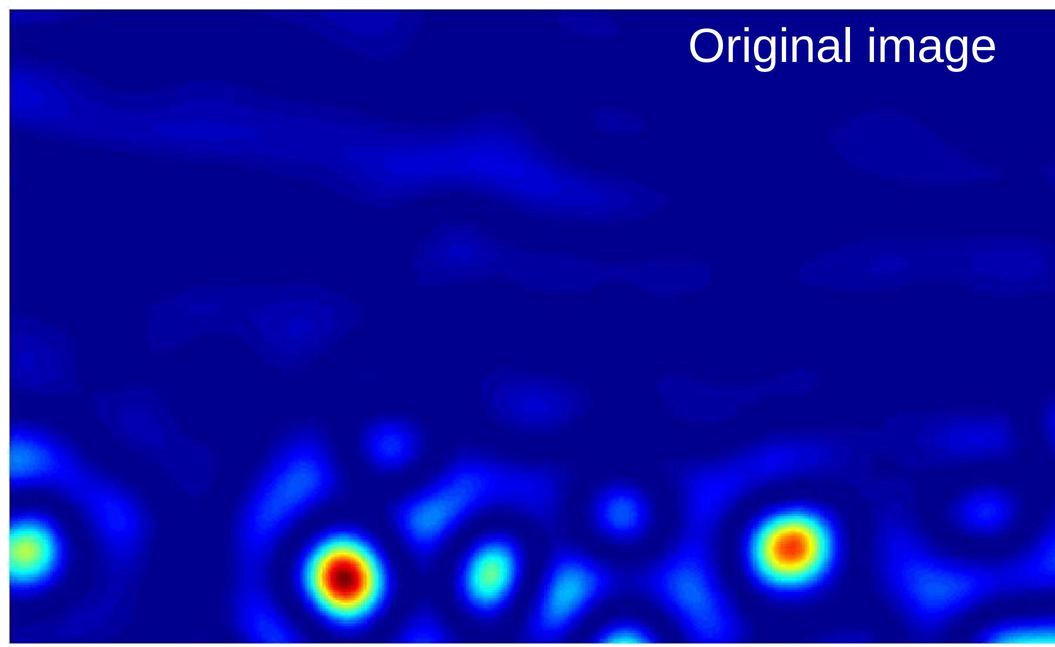




Slide credit: Kristen Grauman





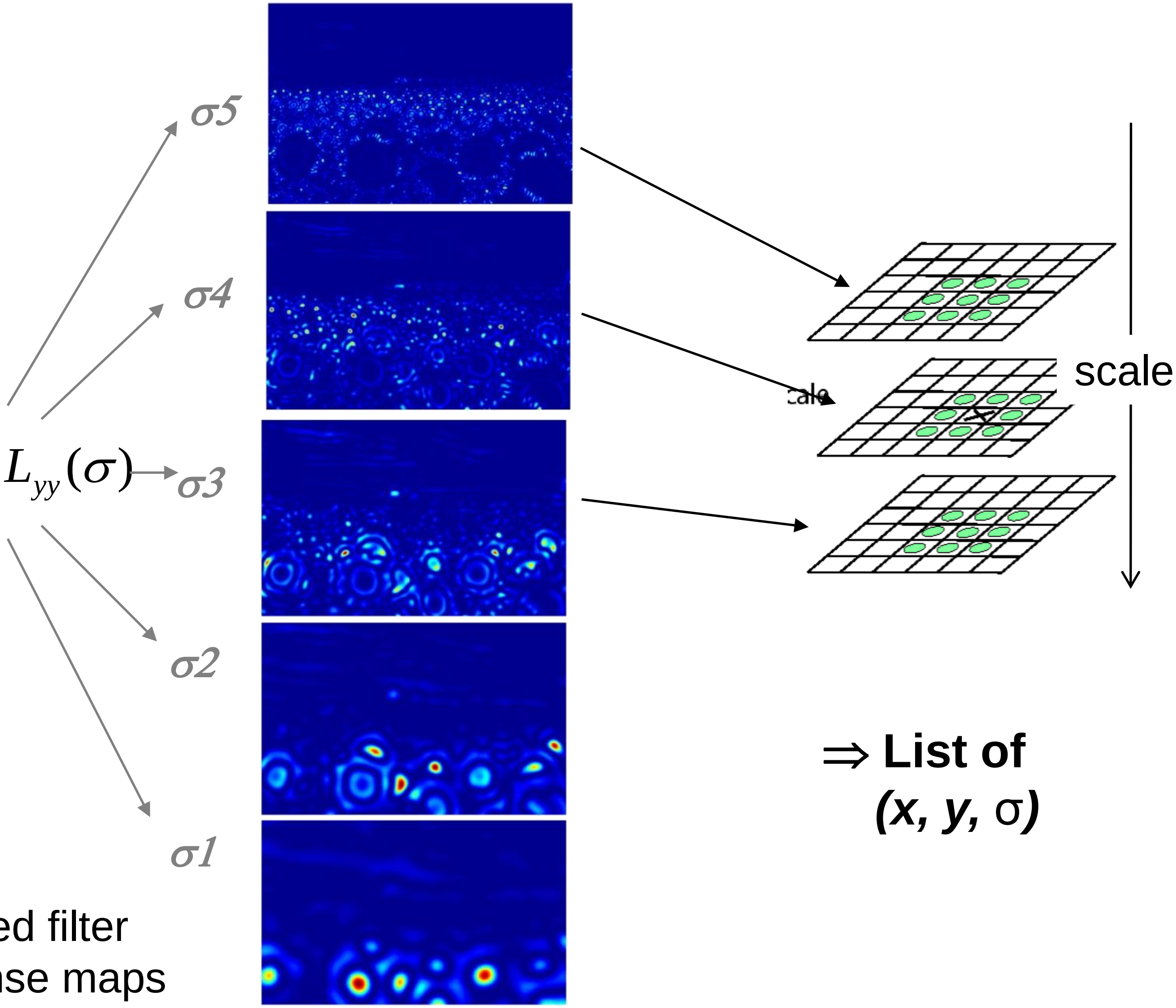


Slide credit: Kristen Grauman

Scale invariant interest points

Interest points are local maxima in both position and scale.

$$L_{xx}(\sigma) + L_{yy}(\sigma)$$



Squared filter response maps

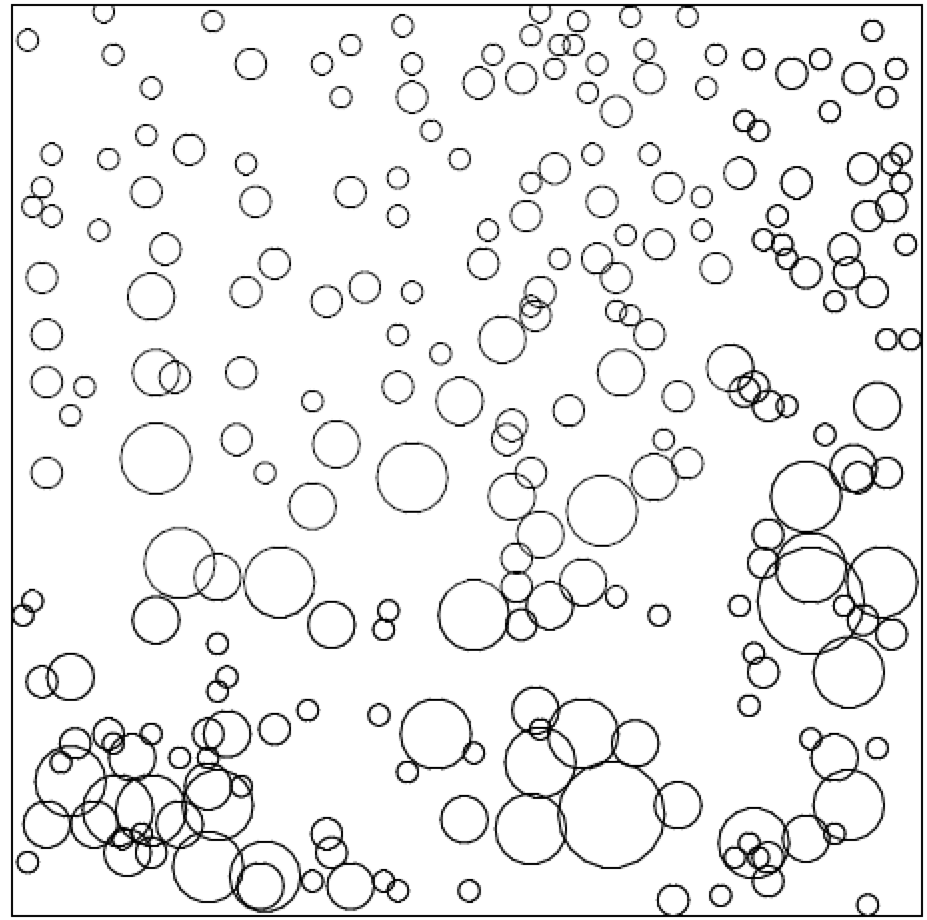
⇒ **List of *(x, y,* σ*)***

Slide credit: Kristen Grauman

Scale-space blob detector: Example

original image

scale-space maxima of $(\nabla^2_{norm} L)^2$



T. Lindeberg. Feature detection with automatic scale selection. IJCV 1998.

Scale-space blob detector: Example

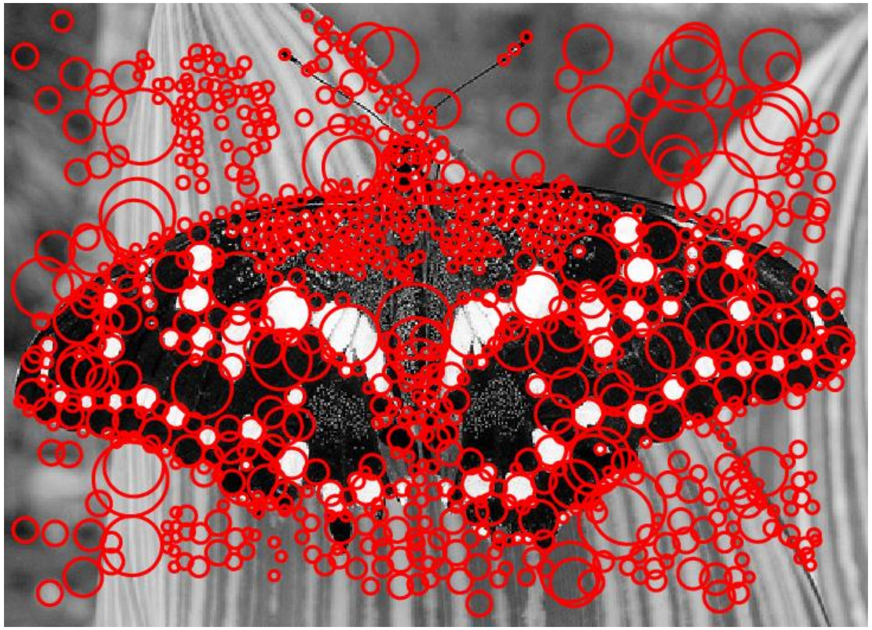


Image credit: Lana Lazebnik

What Is A Useful Signature Function f ?

- Functions for determining scale $f = \text{Kernel} * \text{Image}$

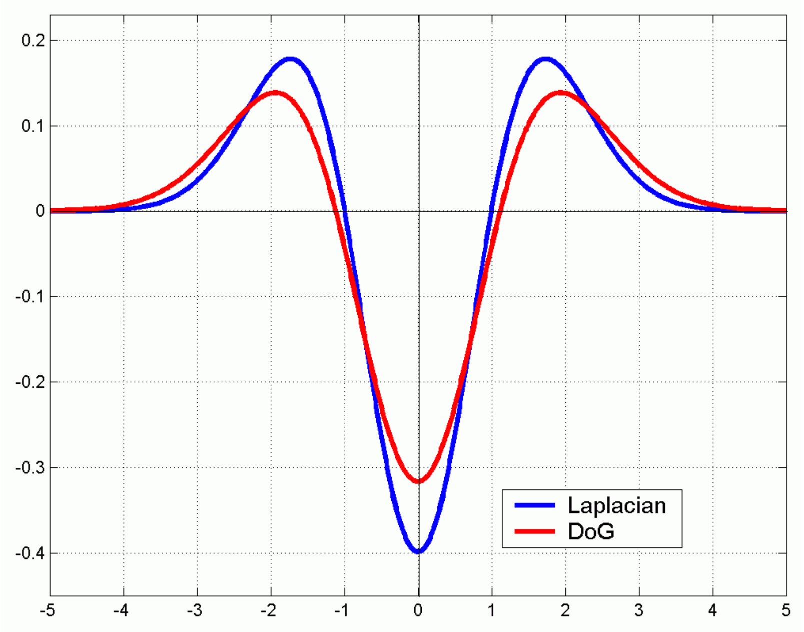
Kernels:

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(Laplacian)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)



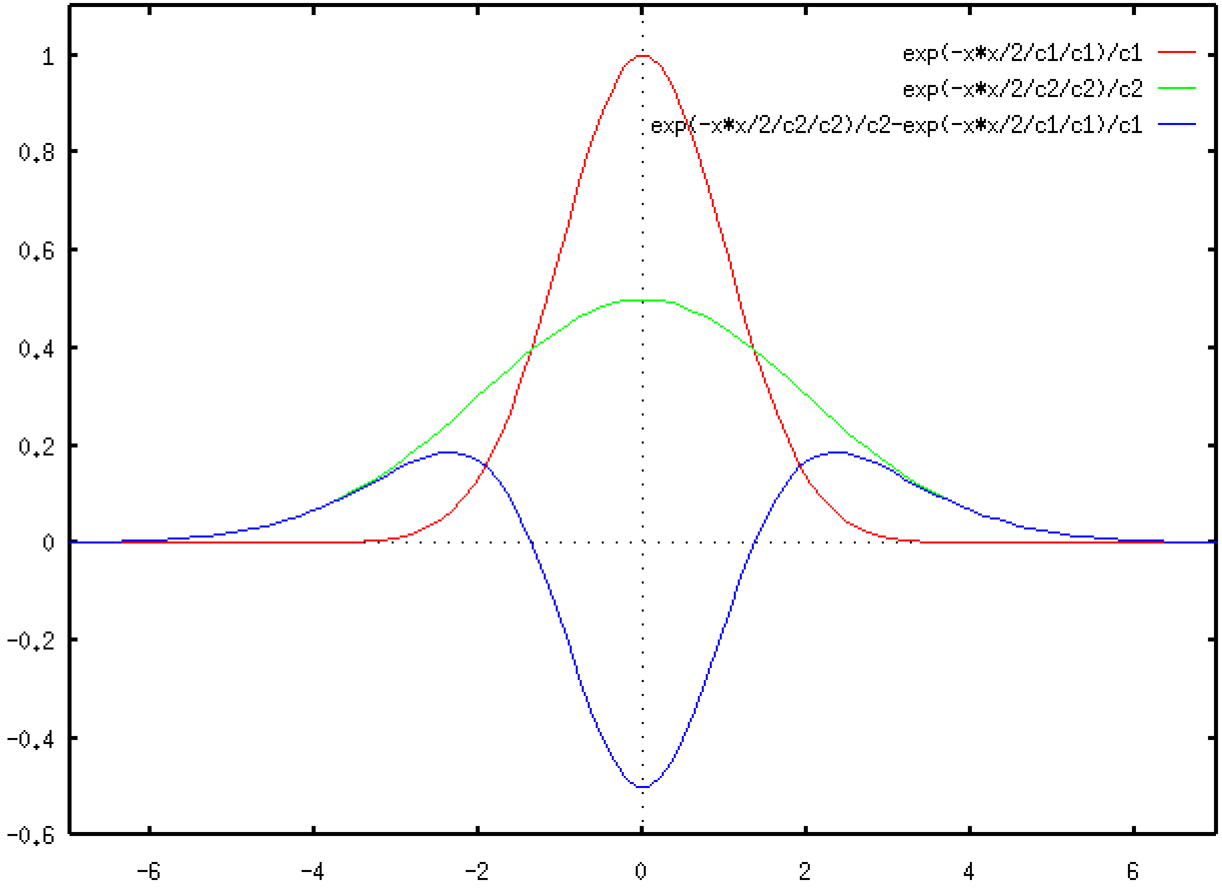
where Gaussian

$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

Note: both kernels are invariant to *scale* and *rotation*

Alternative approach

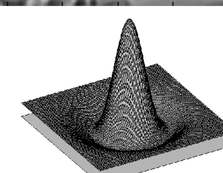
Approximate LoG with Difference-of-Gaussian (DoG).



Alternative approach

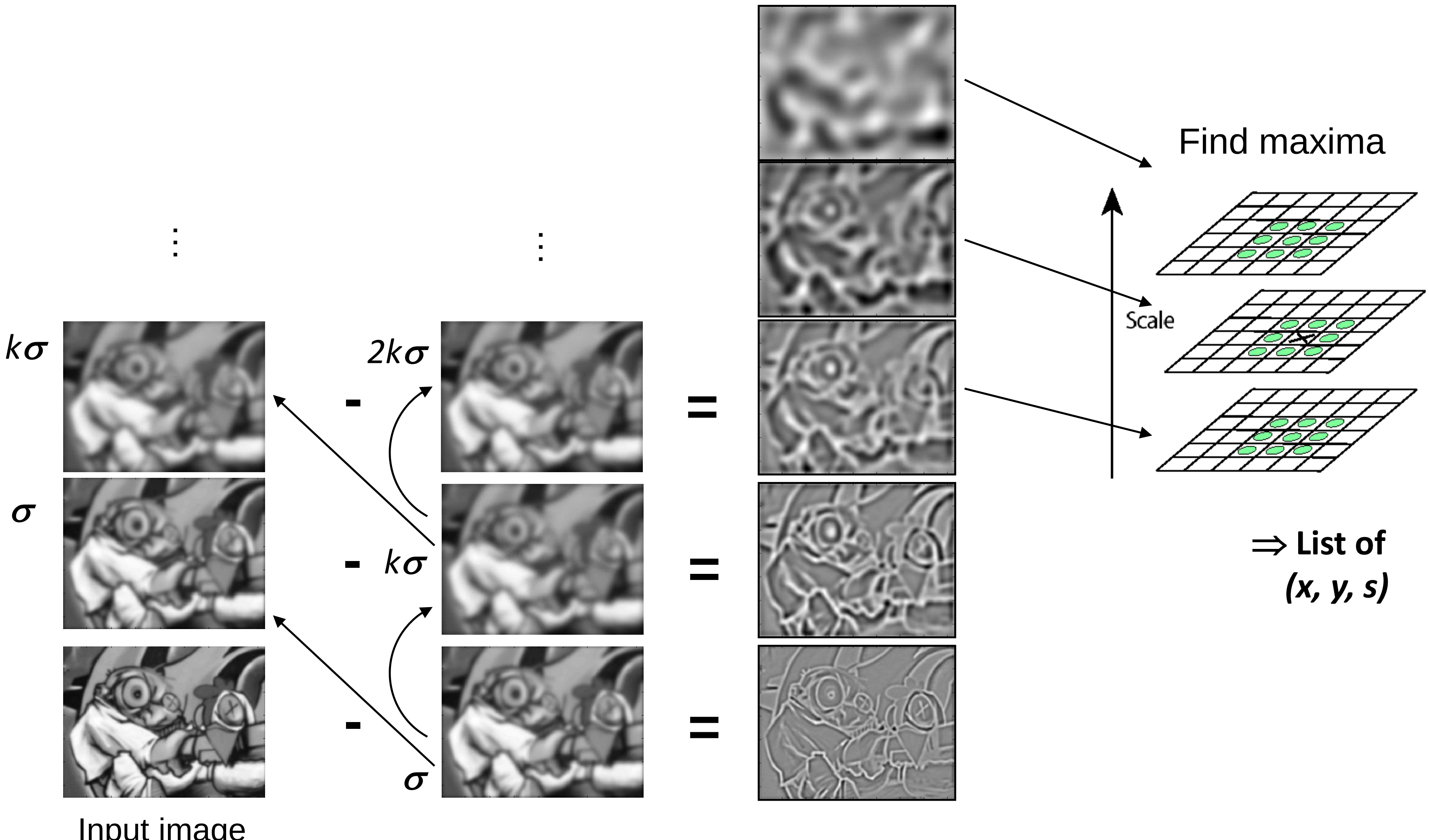
Approximate LoG with Difference-of-Gaussian (DoG).

1. Blur image with σ Gaussian kernel
2. Blur image with *k*σ Gaussian kernel
3. Subtract 2. from 1.



K. Grauman, B. Leibe

Find local maxima in position-scale space of DoG



K. Grauman, B. Leibe

Scale Invariant Detection: Summary

- Given: two images of the same scene with a large *scale difference* between them
- Goal: find *the same* interest points *independently* in each image
- Solution: search for *maxima* of suitable functions in *scale* and in *space* (over the image)

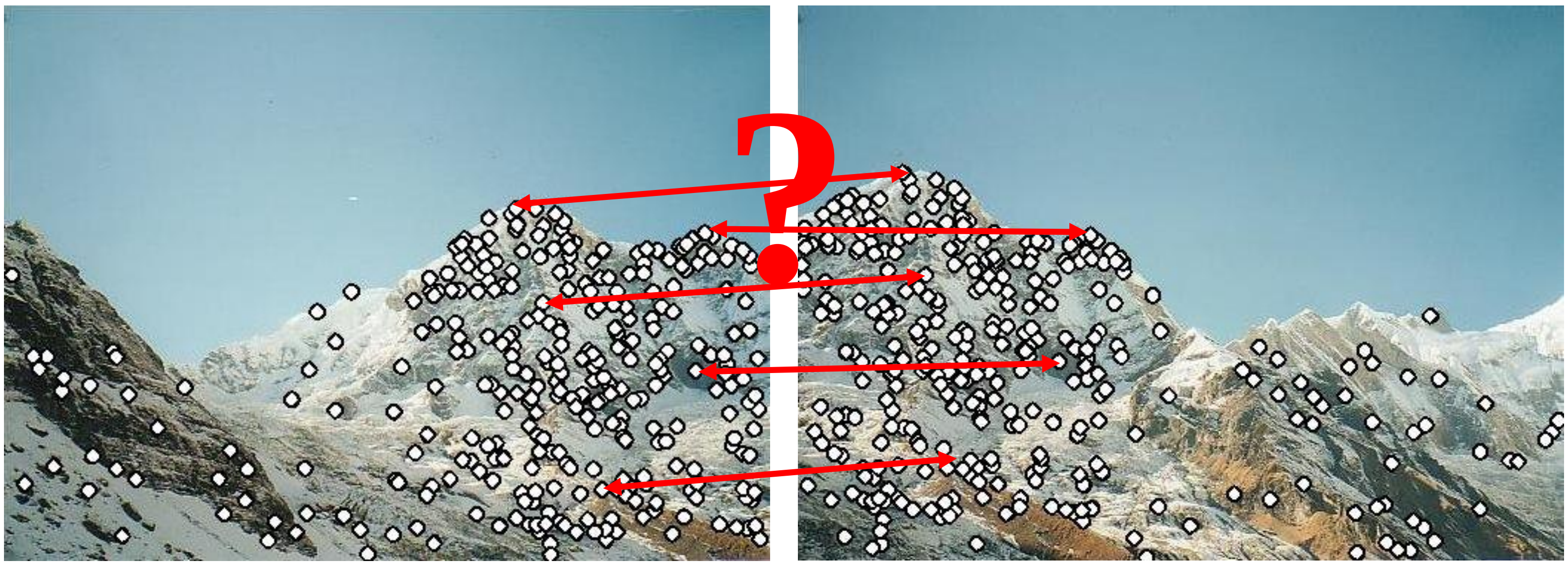
Main questions

- 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:

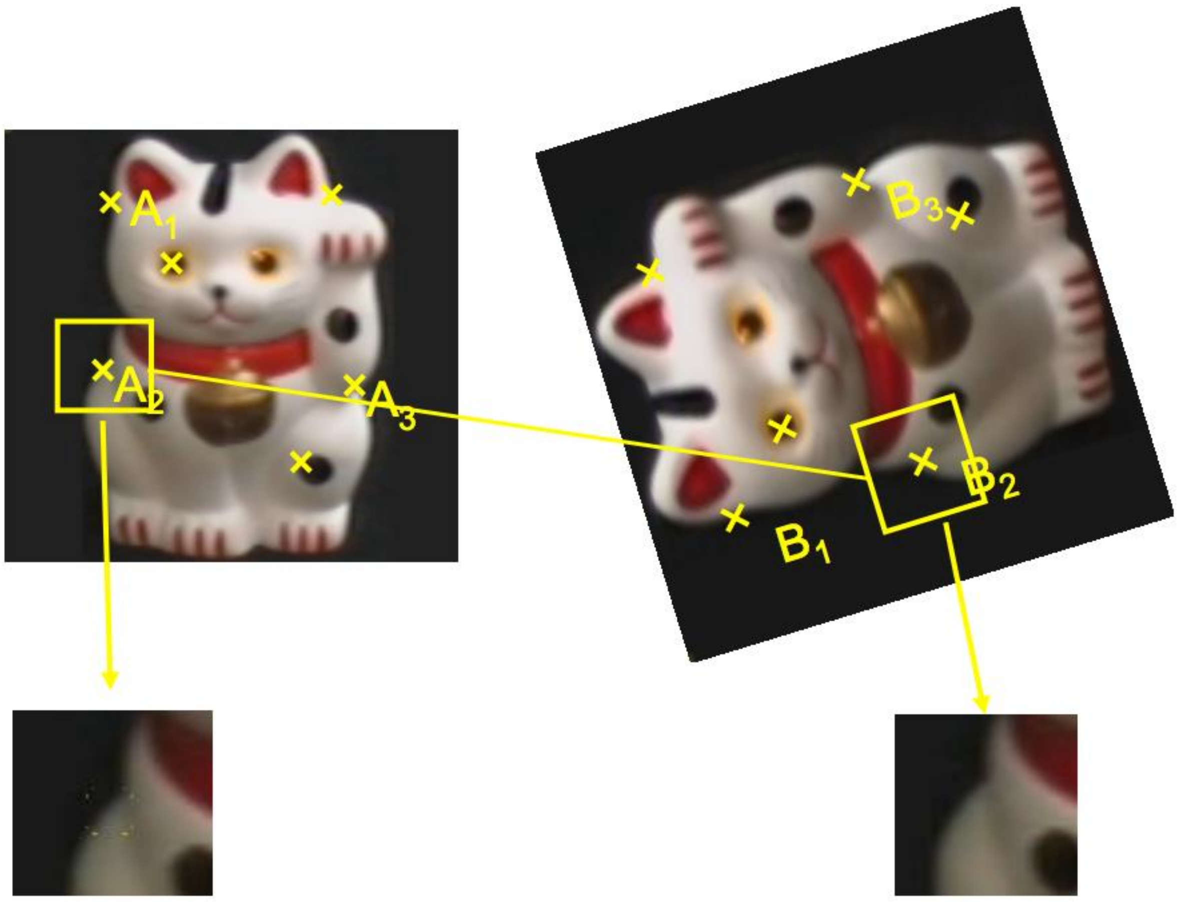
How to *describe* them for matching?



Point descriptor should be:

1. Invariant
2. Distinctive

Becoming rotation invariant



Orientation Assignment

- Use scale of point to choose correct image:

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

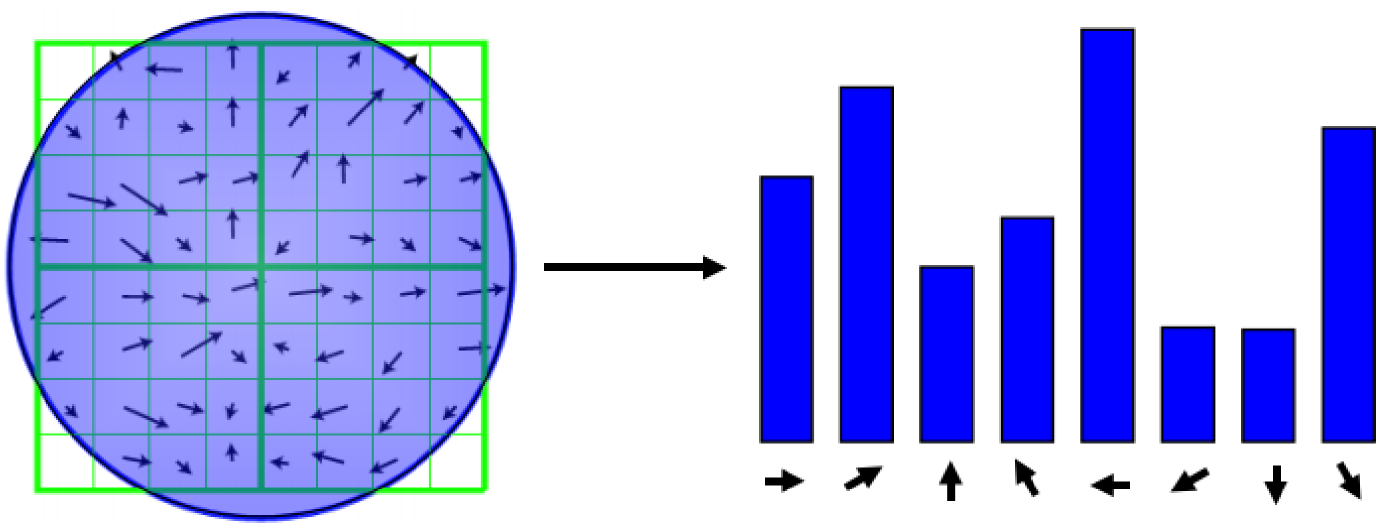
- Compute gradient magnitude and orientation using finite differences:

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

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

Orientation Assignment

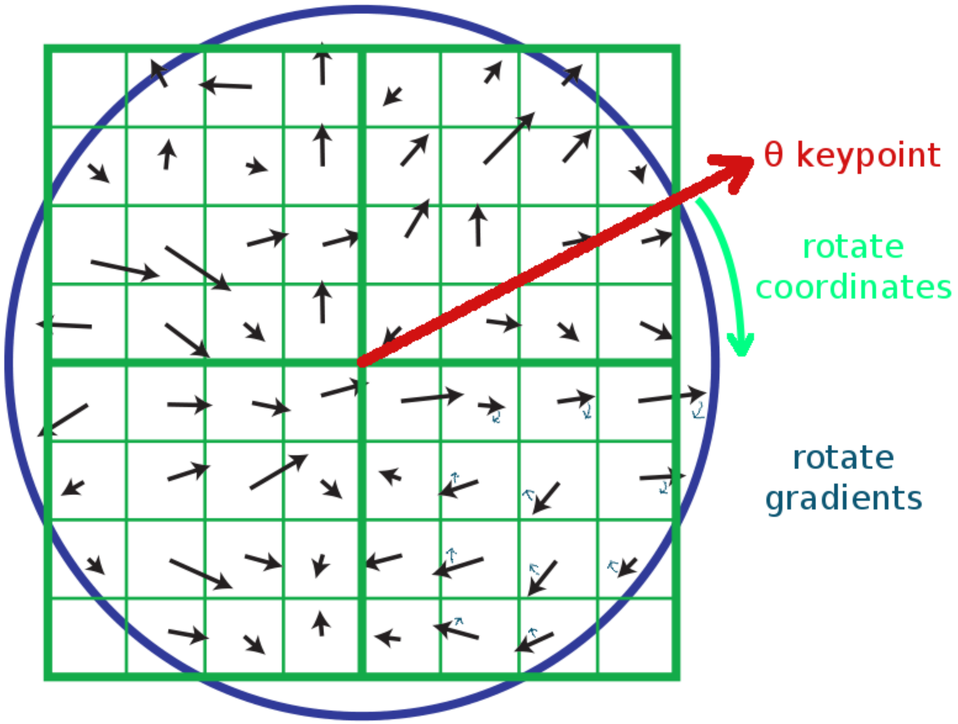
- Create gradient histogram (36 bins)
  - Weighted by magnitude and Gaussian window (σ is 1.5 times that of the scale of a keypoint)



Orientation Assignment

- Any peak within 80% of the highest peak is used to create a keypoint with that orientation
- ~15% assigned multiplied 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



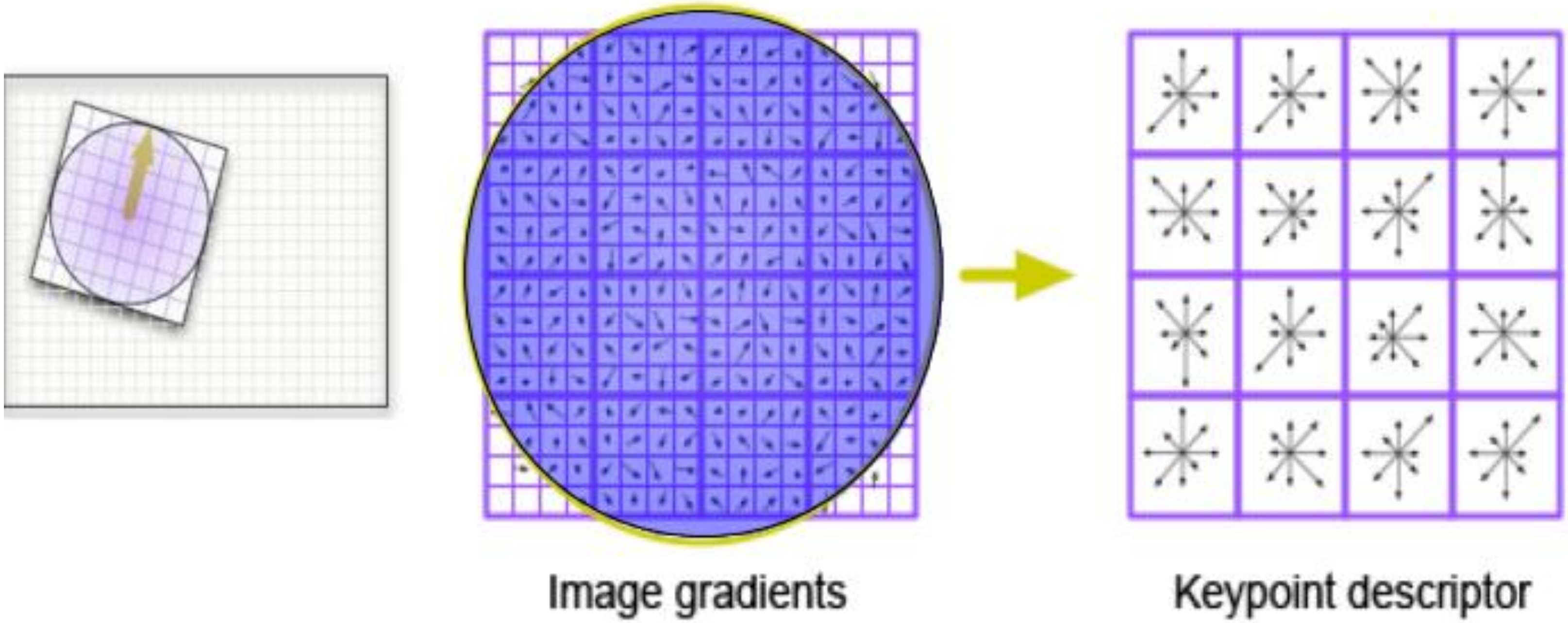
- To become rotation invariant, rotate the gradient directions AND locations by (-keypoint orientation)
  - Now we’ve cancelled out rotation and have gradients expressed at locations **relative** to keypoint orientation θ
  - We could also have just rotated the whole image by -θ, but that would be slower.

SIFT descriptor formation

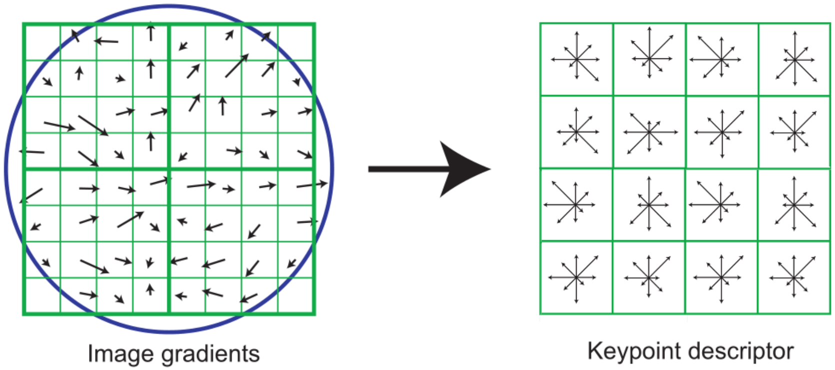
- Each point so far has x, y, σ, m, θ
- Now we need a descriptor for the region
  - Could sample intensities around point, but…
    - Sensitive to lighting changes
    - Sensitive to slight errors in x, y, θ

SIFT descriptor formation

- 4x4 Gradient window
- Histogram of 4x4 samples per window in 8 directions
- Gaussian weighting around center(σ is 0.5 times that of the scale of a keypoint)
- 4x4x8 = 128 dimensional feature vector



SIFT descriptor formation

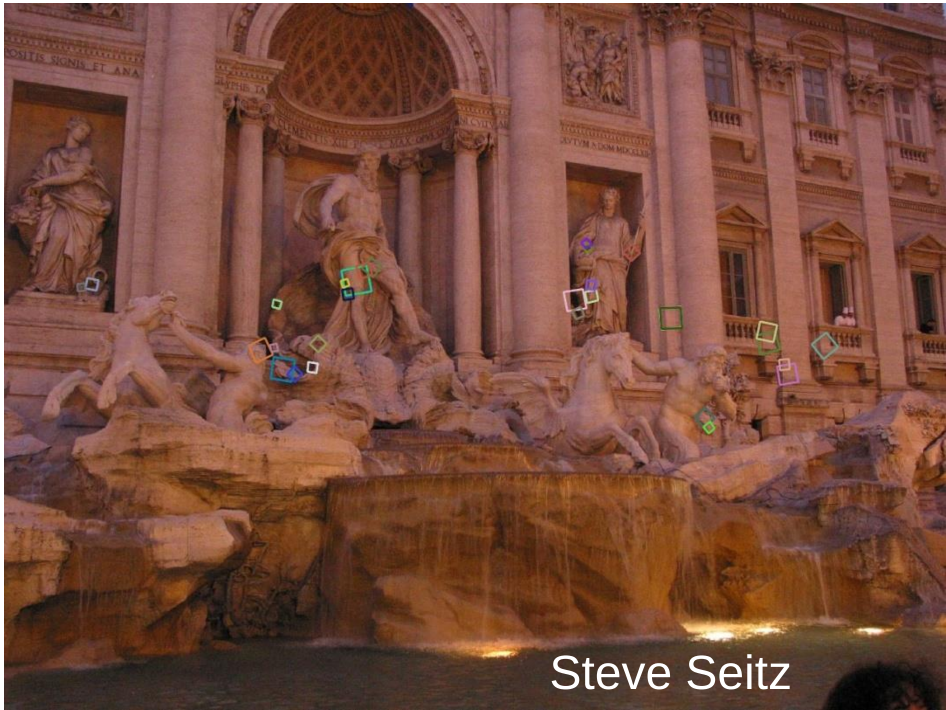


- Adding robustness to illumination changes:
- Remember that the descriptor is made of gradients (differences between pixels), so it's already invariant to changes in brightness (e.g. adding 10 to all image pixels yields the exact same descriptor)
- A higher-contrast photo will increase the magnitude 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 ≤ 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

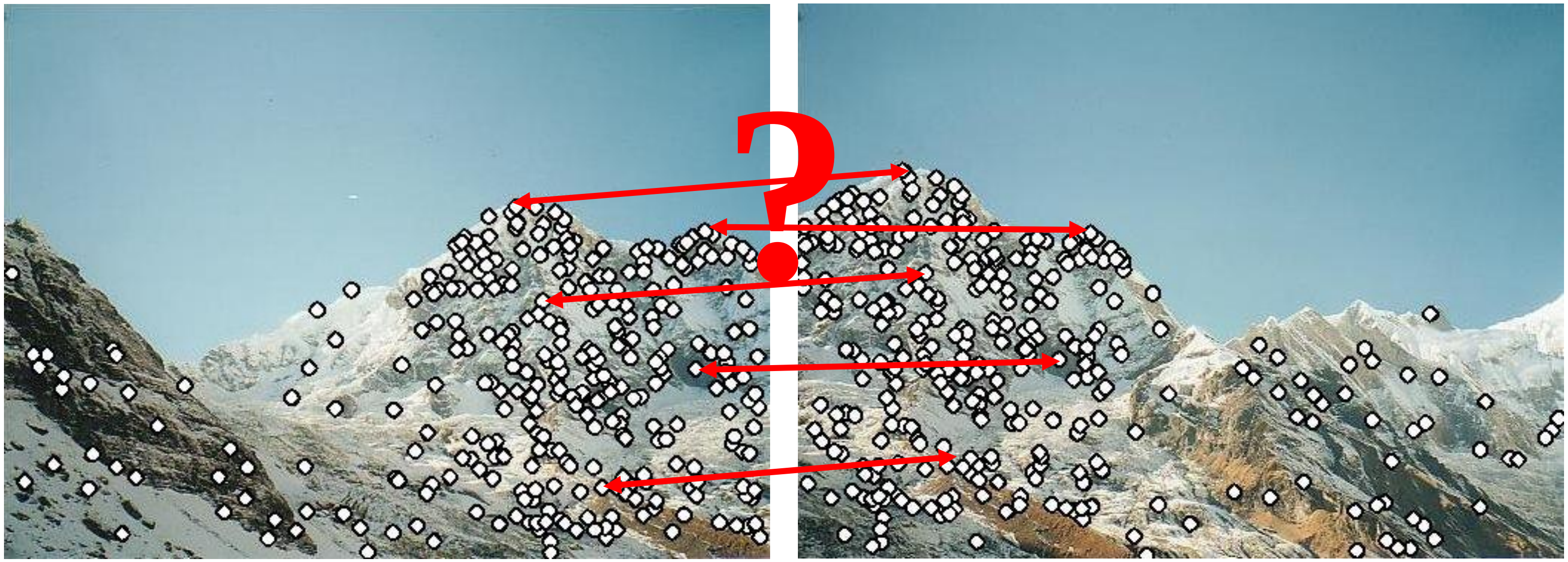


Main questions

- 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

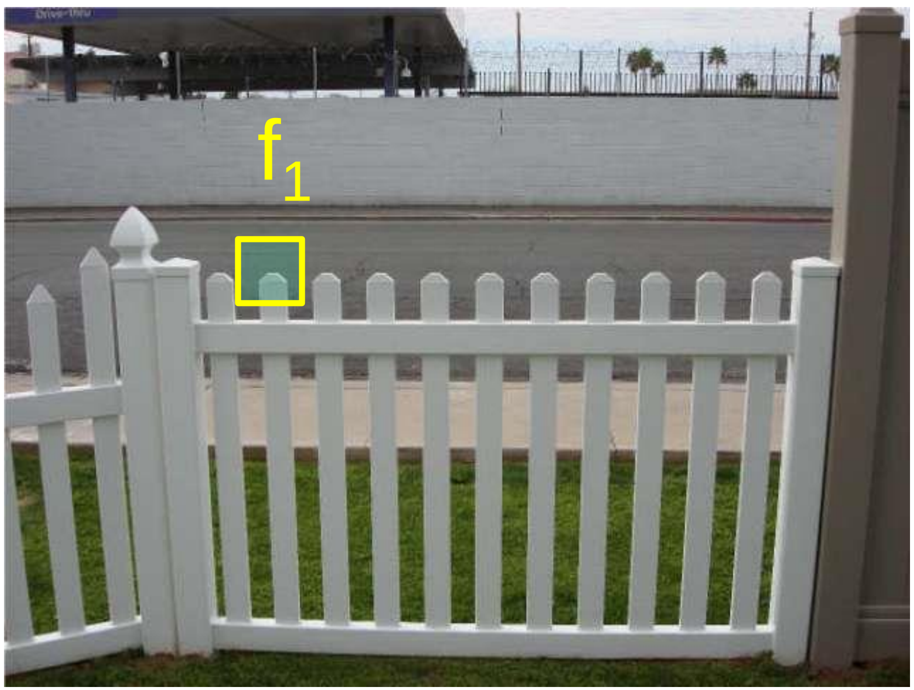
Given a feature in I_1, how to find the best match in I_2?

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

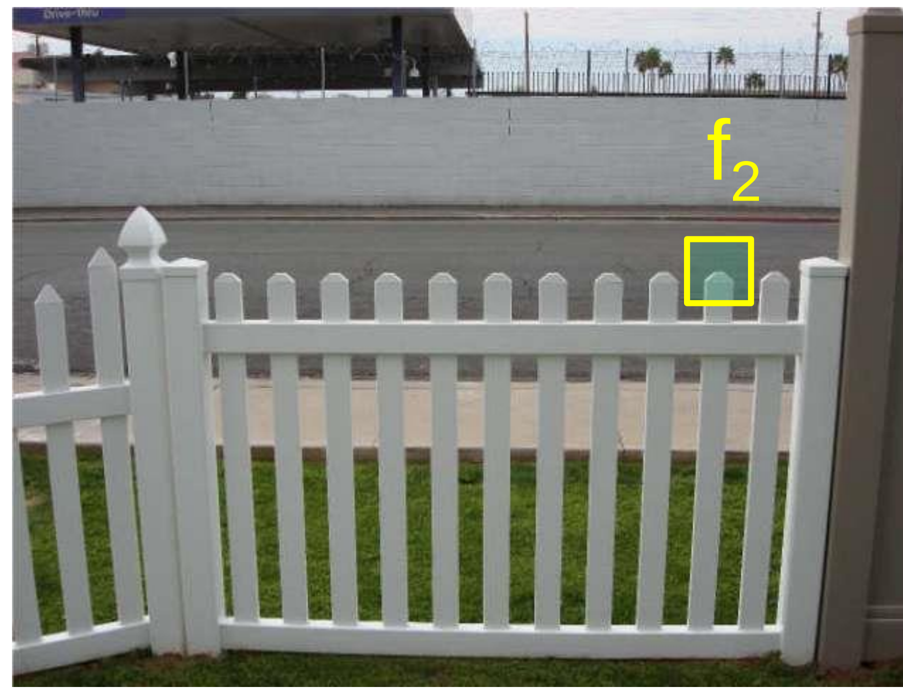
Feature distance

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

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



I_1

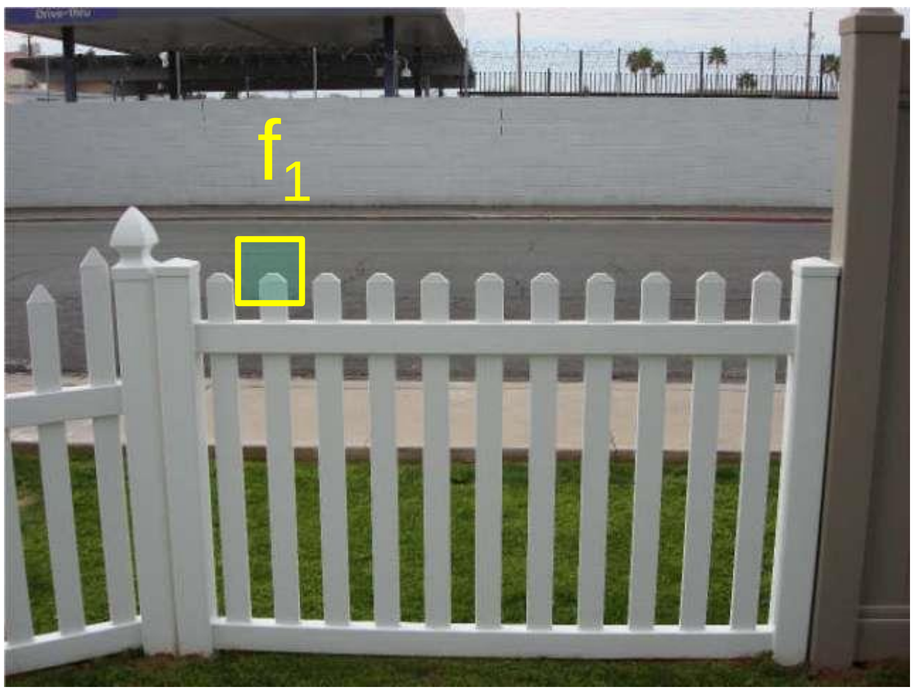


I_2

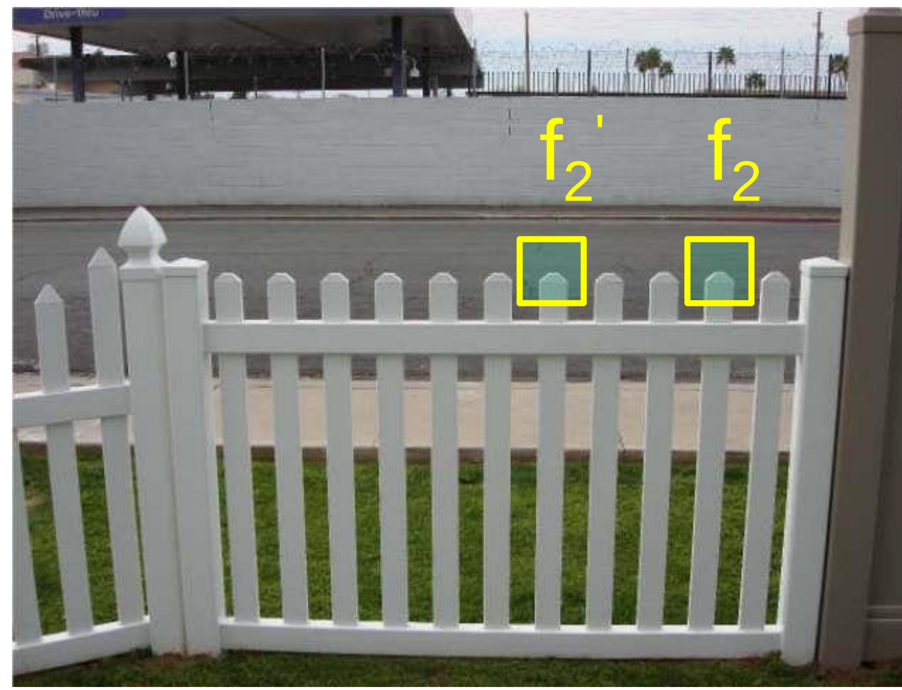
Feature distance

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

- Better approach: ratio distance = $SSD(f_1, f_2) / SSD(f_1, f_2')$
  - 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
  - An ambiguous/bad match will have ratio close t
  - Look for unique matches which have low ratio



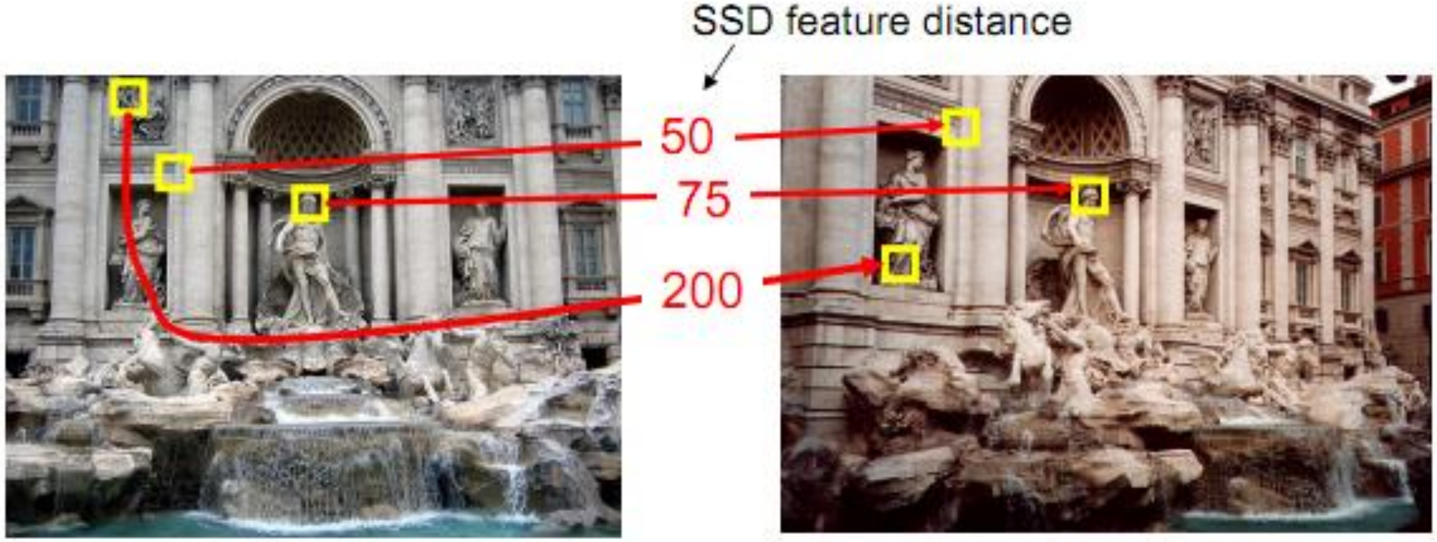
I_1



I_2

Evaluating the results

How can we measure the performance of a feature matcher?



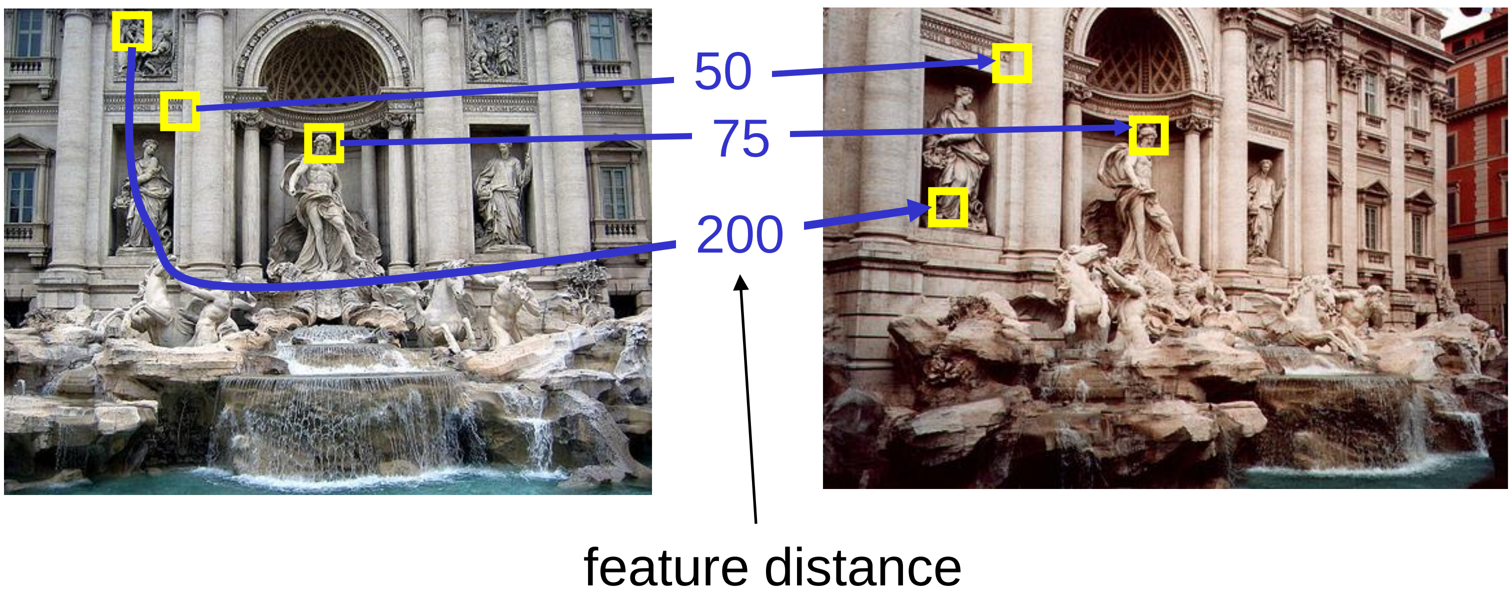
Suppose we use SSD
Small values are possible matches but how small?

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

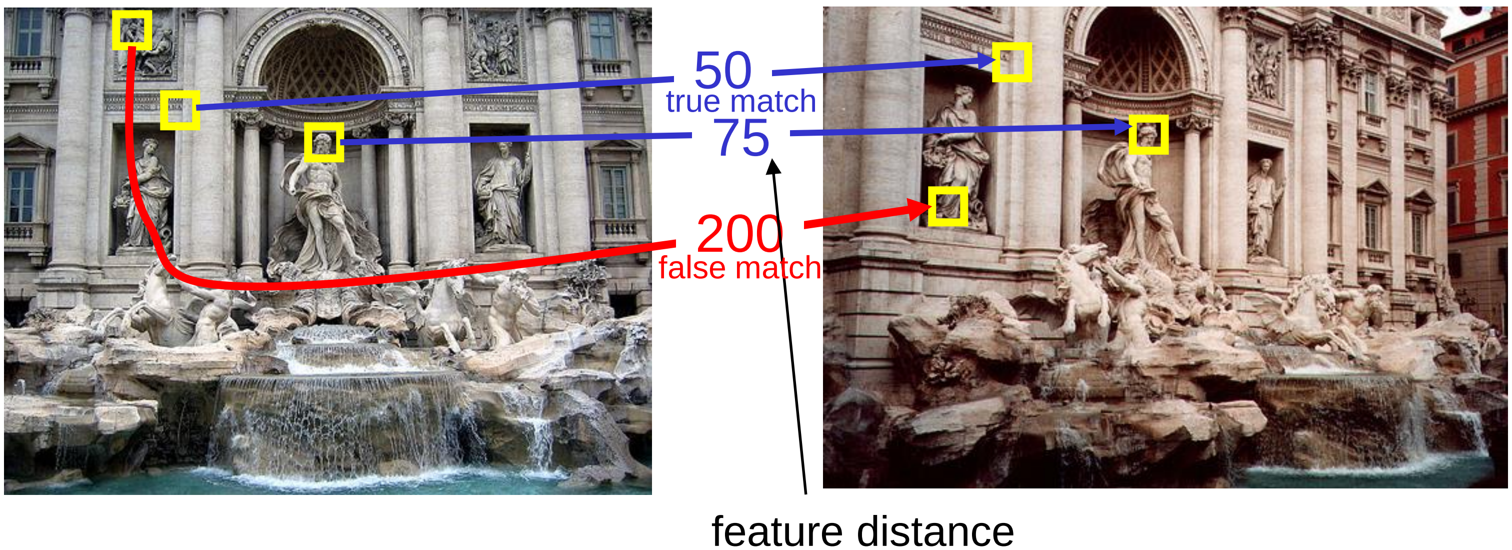
What is the effect of choosing a particular T?

Evaluating the results

How can we measure the performance of a feature matcher?



True/false positives



The distance threshold affects performance

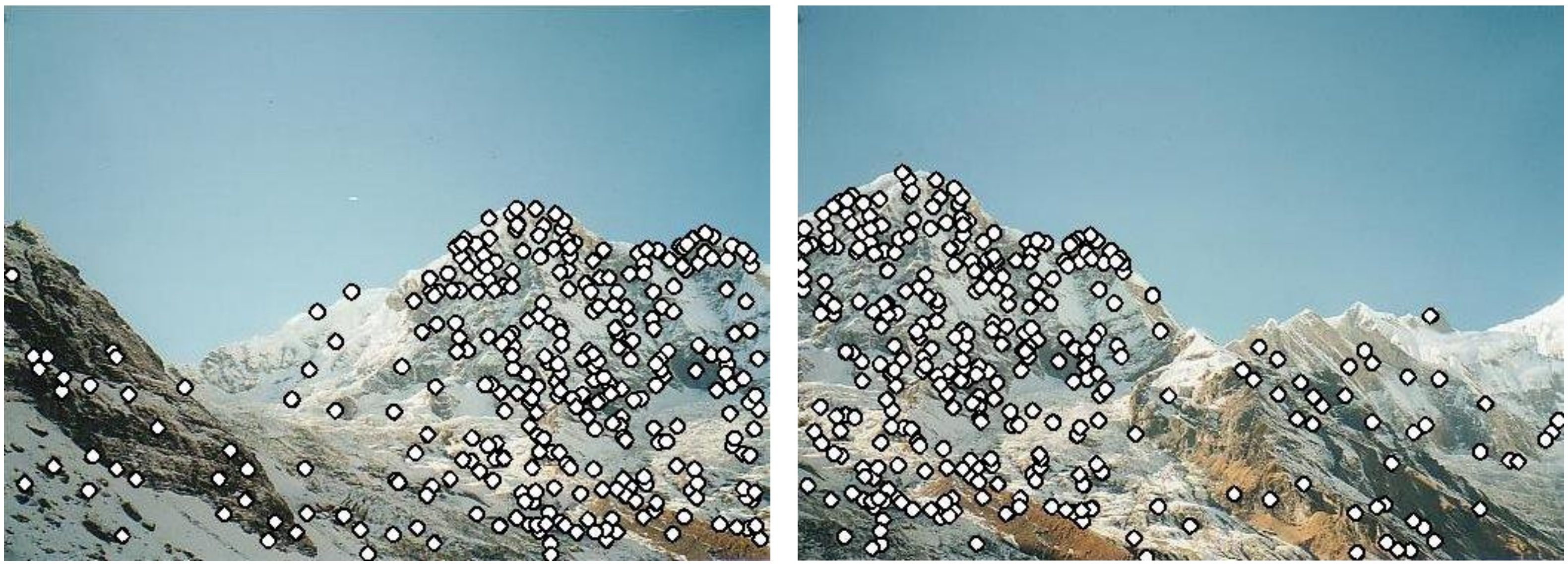
- True positives = # of detected matches that are correct
  - Suppose we want to maximize these—how to choose threshold?
- False positives = # of detected matches that are incorrect
  - Suppose we want to minimize these—how to choose threshold?

Application: Image Stitching

Slide credit: Darya Frolova, Denis Simakov

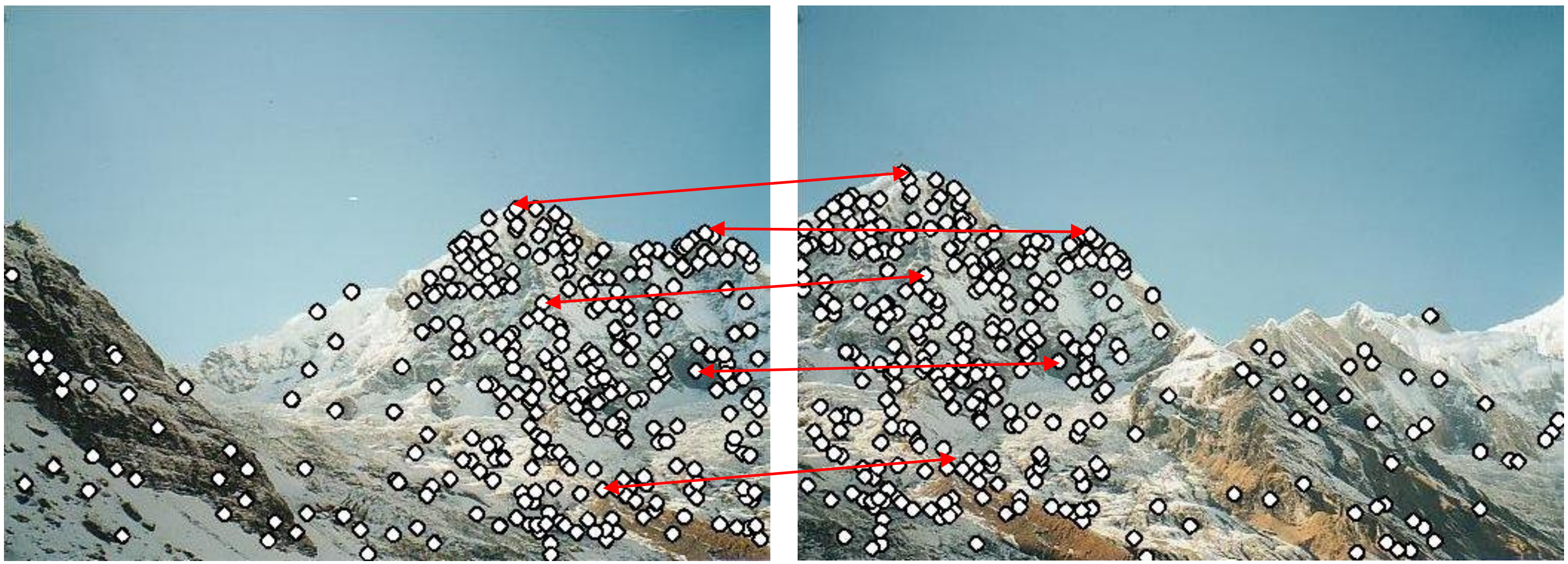
Application: Image Stitching



- Procedure:
  - Detect feature points in both images

Slide credit: Darya Frolova, Denis Simakov

Application: Image Stitching



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

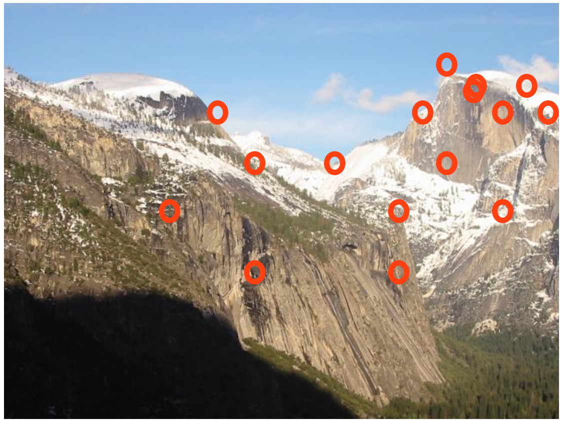
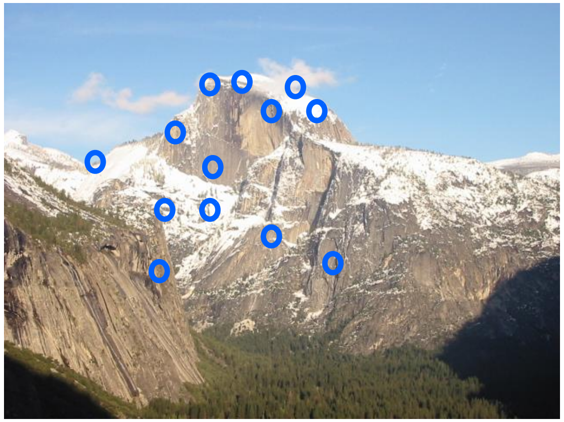
Slide credit: Darya Frolova, Denis Simakov

Application: Image Stitching

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

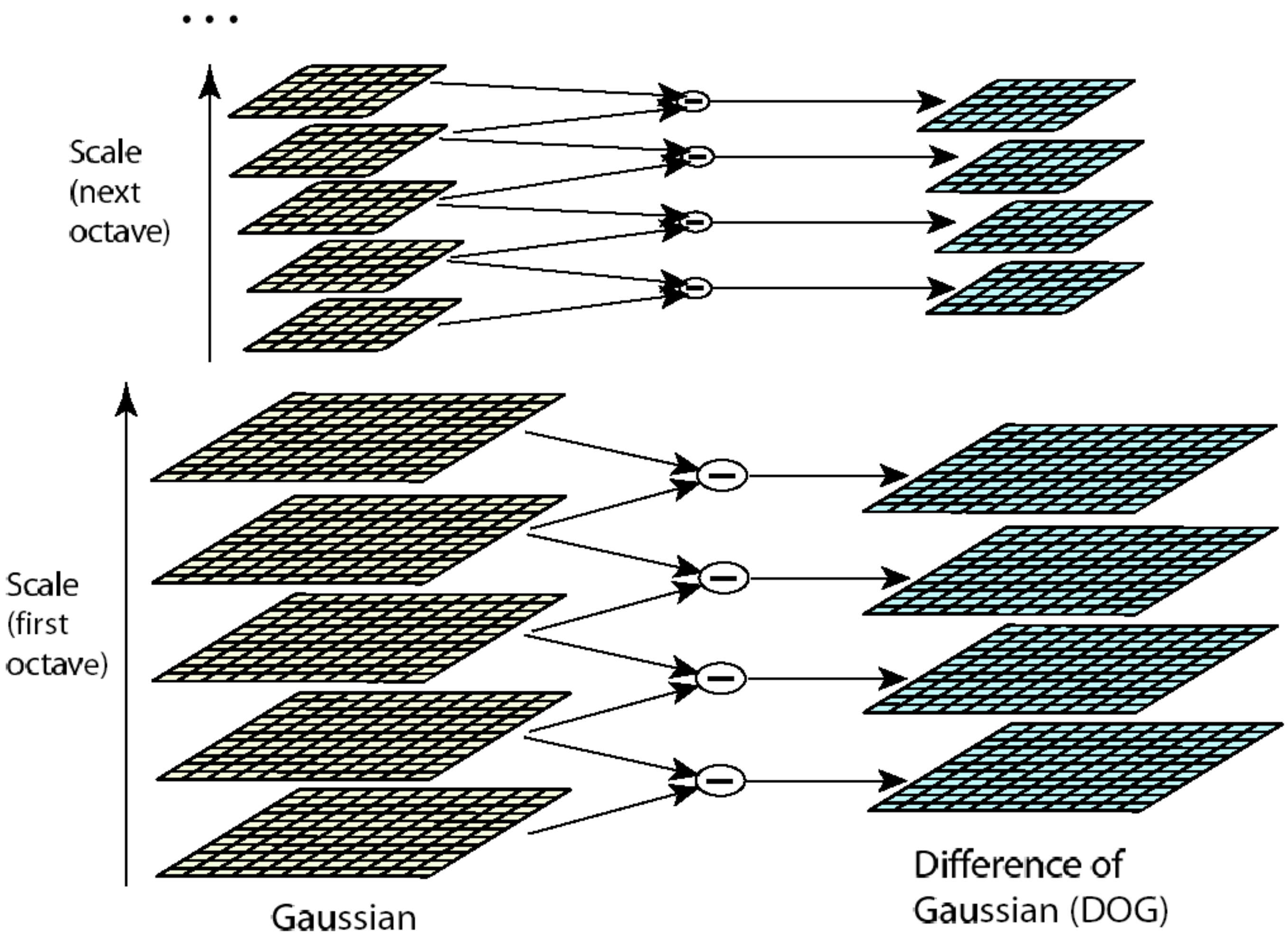
Slide credit: Darya Frolova, Denis Simakov

Main Flow

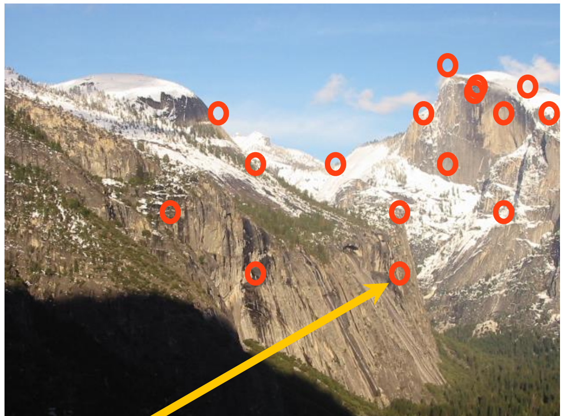
 

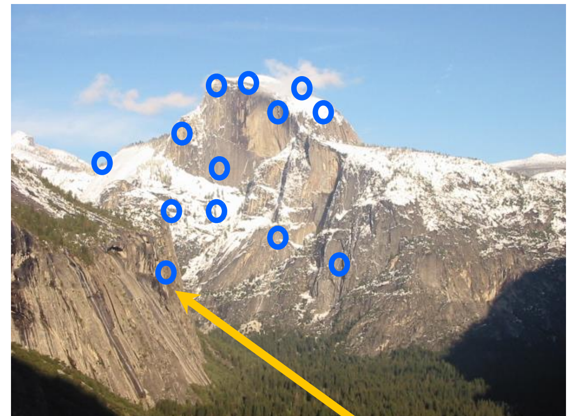
- Detect key points

Detect Key Points



Main Flow



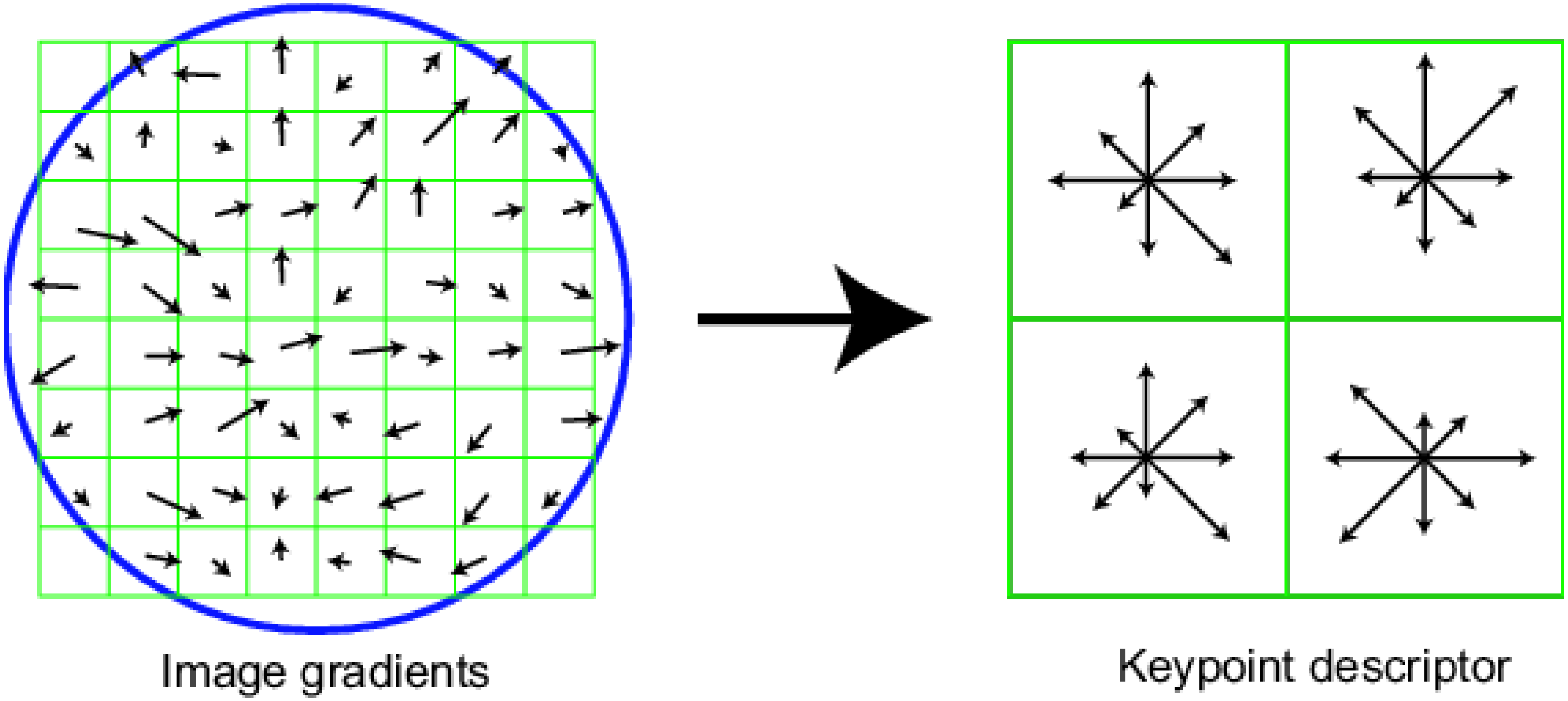


$$(u_1, u_2, \ldots, u_{128})$$

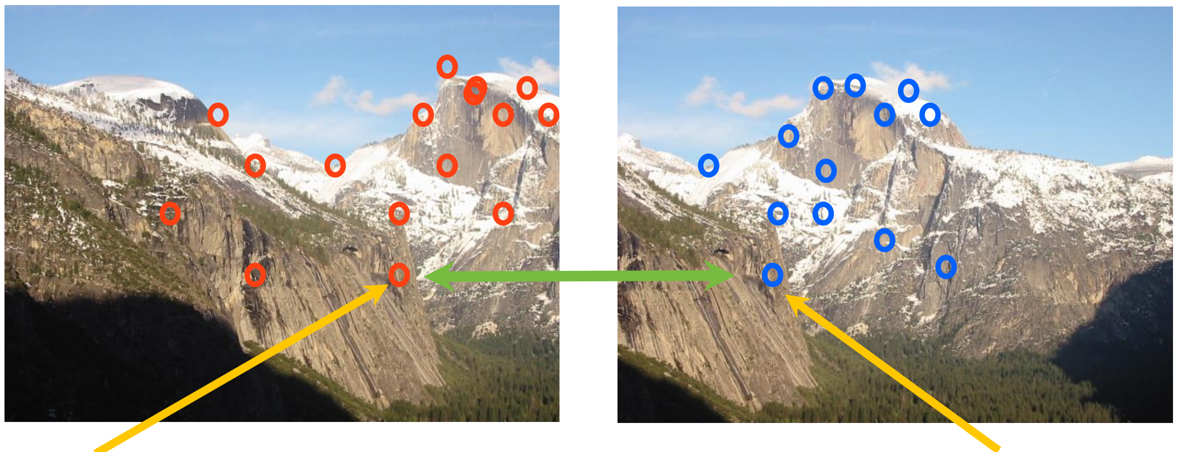
$$(v_1, v_2, \ldots, v_{128})$$

- Detect key points
- Build the SIFT descriptors

Build the SIFT Descriptors



Main Flow

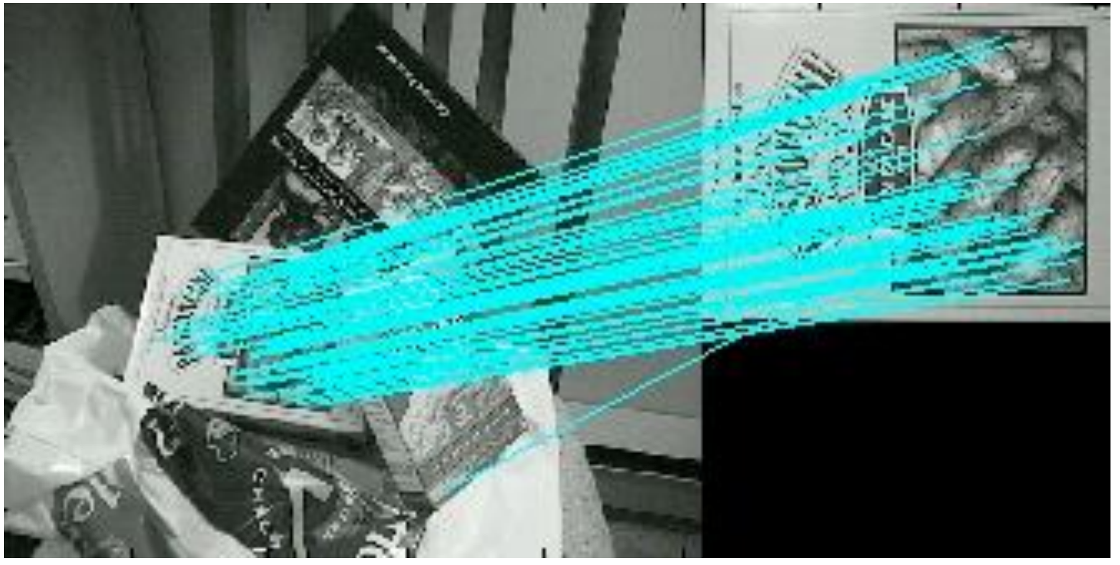


$(u_1, u_2, \ldots, u_{128})$ $(v_1, v_2, \ldots, v_{128})$

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

Match SIFT Descriptors

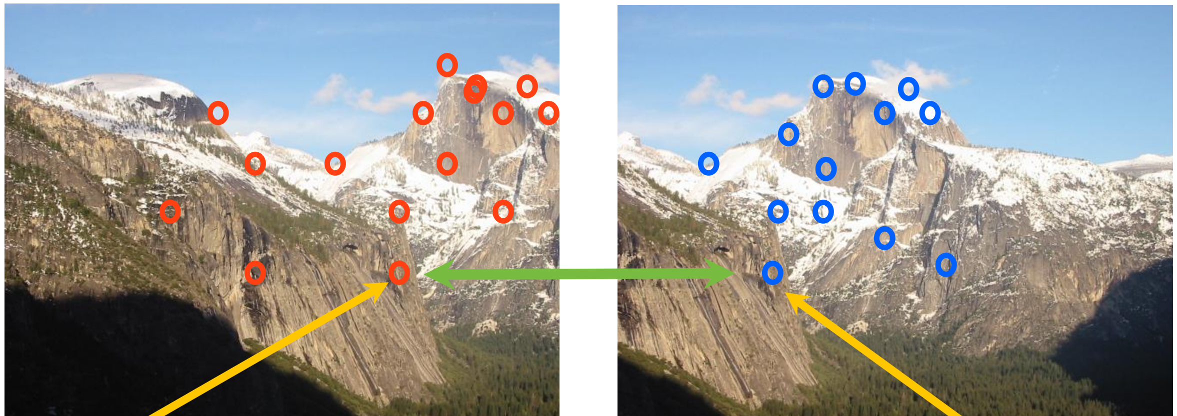
- Euclidean distance between descriptors



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



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

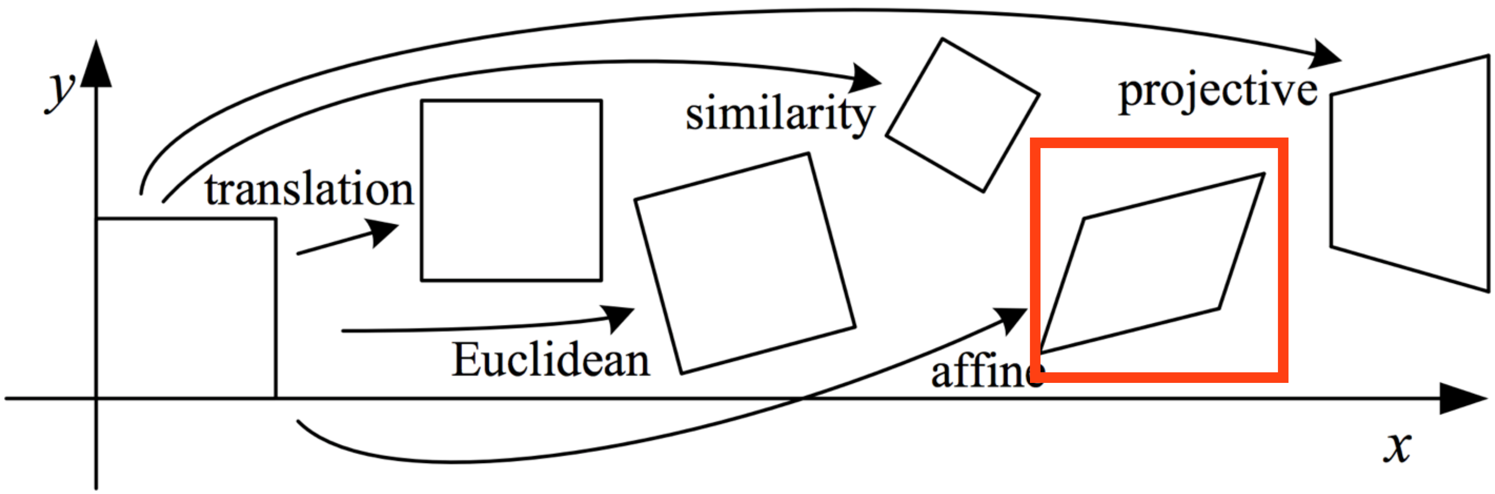
$(v_1, v_2, \ldots, v_{128})$

- 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



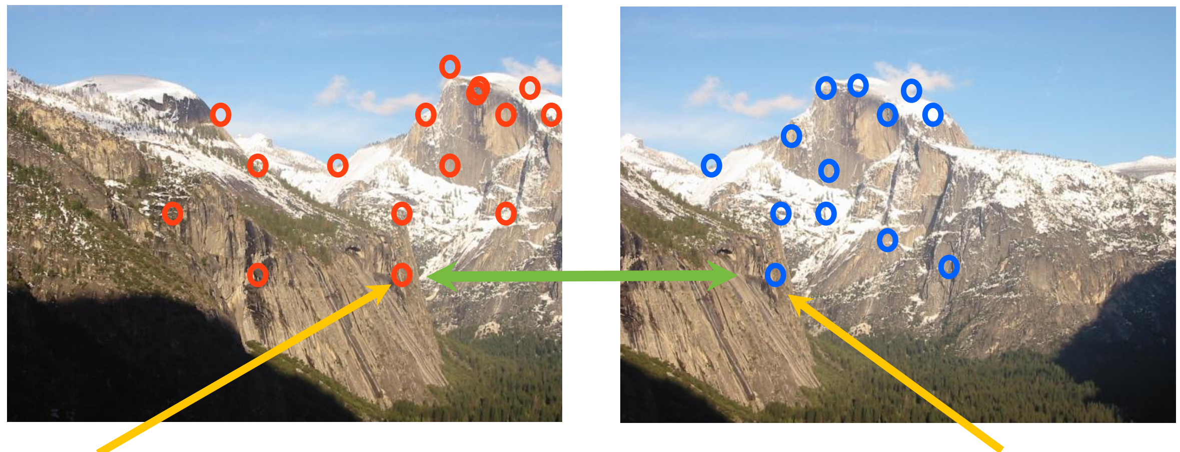
Skeleton Code

- 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



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

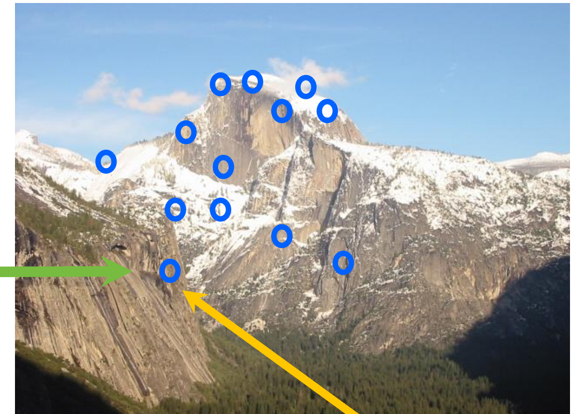
- Detect key points
- Build the SIFT descriptors
- Match SIFT descriptors
- Fitting the transformation
- RANSAC

Skeleton Code

- RANSAC
  - ComputeError

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

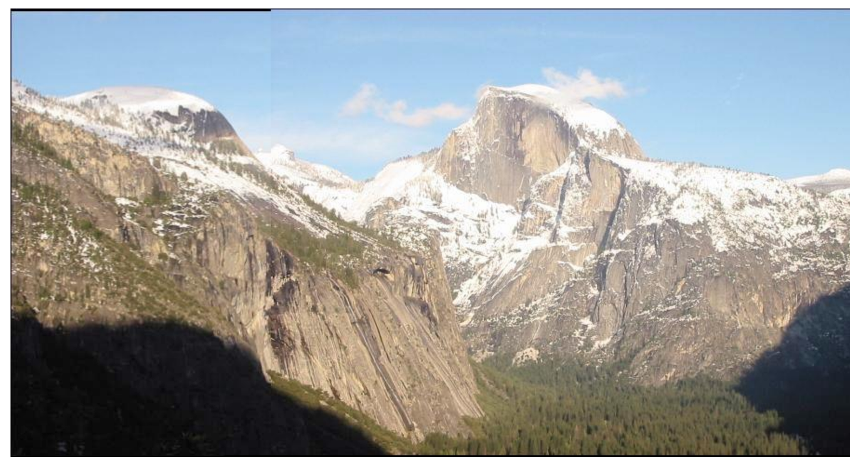
Main Flow



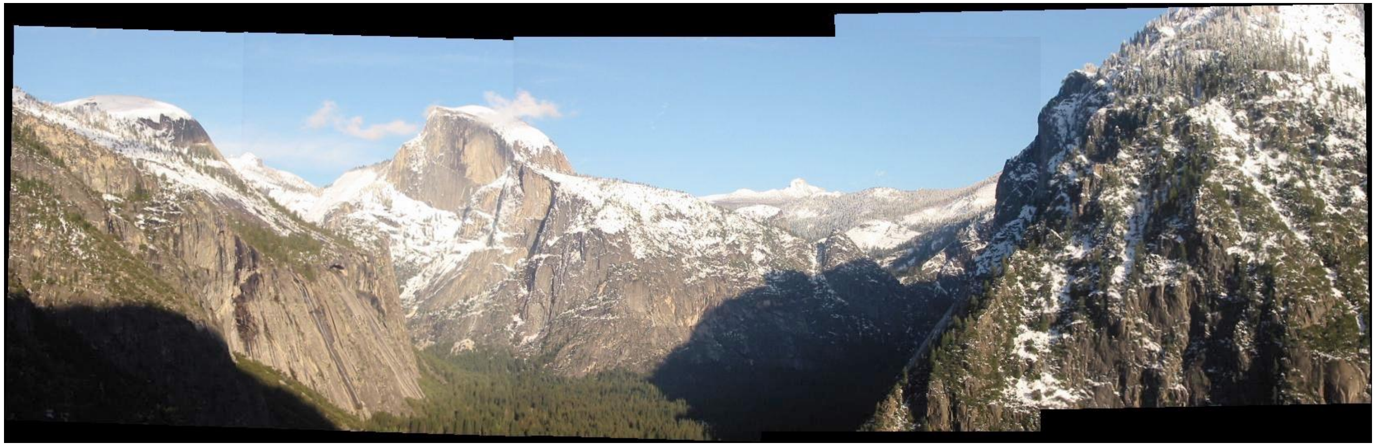
$$(u_1, u_2, \ldots, u_{128})$$

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

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



Results



Results