# Local Feature Detectors and Descriptors

# Overview

- Local invariant features
- Keypoint localization
  - Hessian detector
  - Harris corner detector
- Scale Invariant region detection
  - Laplacian of Gaussian (LOG) detector
  - Difference of Gaussian (DOG) detector
- Local feature descriptor
  - Scale Invariant Feature Transform (SIFT)
  - Gradient Localization Oriented Histogram (GLOH)
- Examples of other local feature descriptors

# Motivation

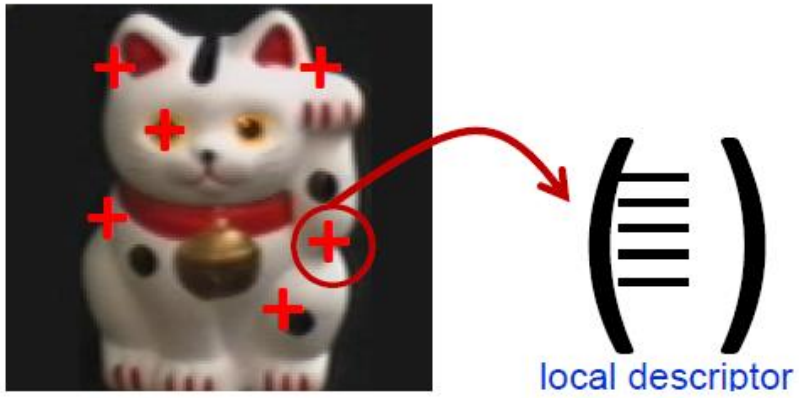
- Global feature from the whole image is often not desirable
- Instead match local regions which are prominent to the object or scene in the image
- Application Area
  - Object detection
  - Image matching
  - Image stitching

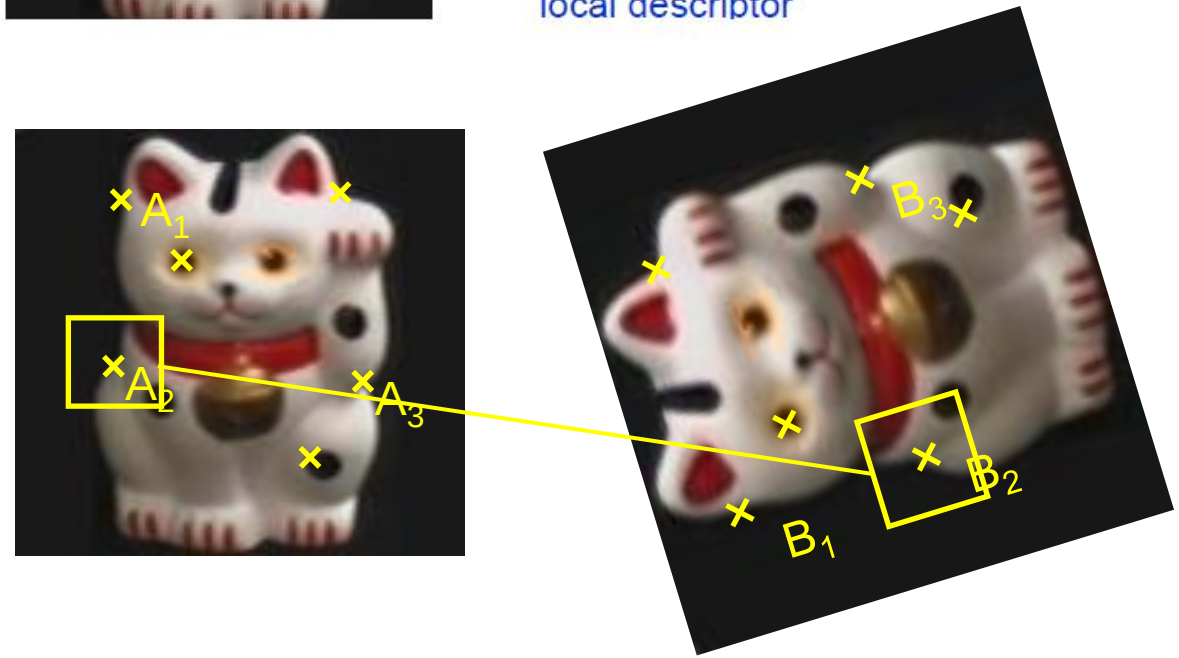
# Requirements of a local feature

- Repetitive : Detect the same points independently in each image
- Invariant to translation, rotation, scale
- Invariant to affine transformation
- Invariant to presence of noise, blur etc.
- Locality :Robust to occlusion, clutter and illumination change
- Distinctiveness : The region should contain “interesting” structure
- Quantity : There should be enough points to represent the image
- Time efficient

# General approach

- Find the interest points
- Consider the region around each keypoint
- Compute a local descriptor from the region and normalize the feature
- Match local descriptors

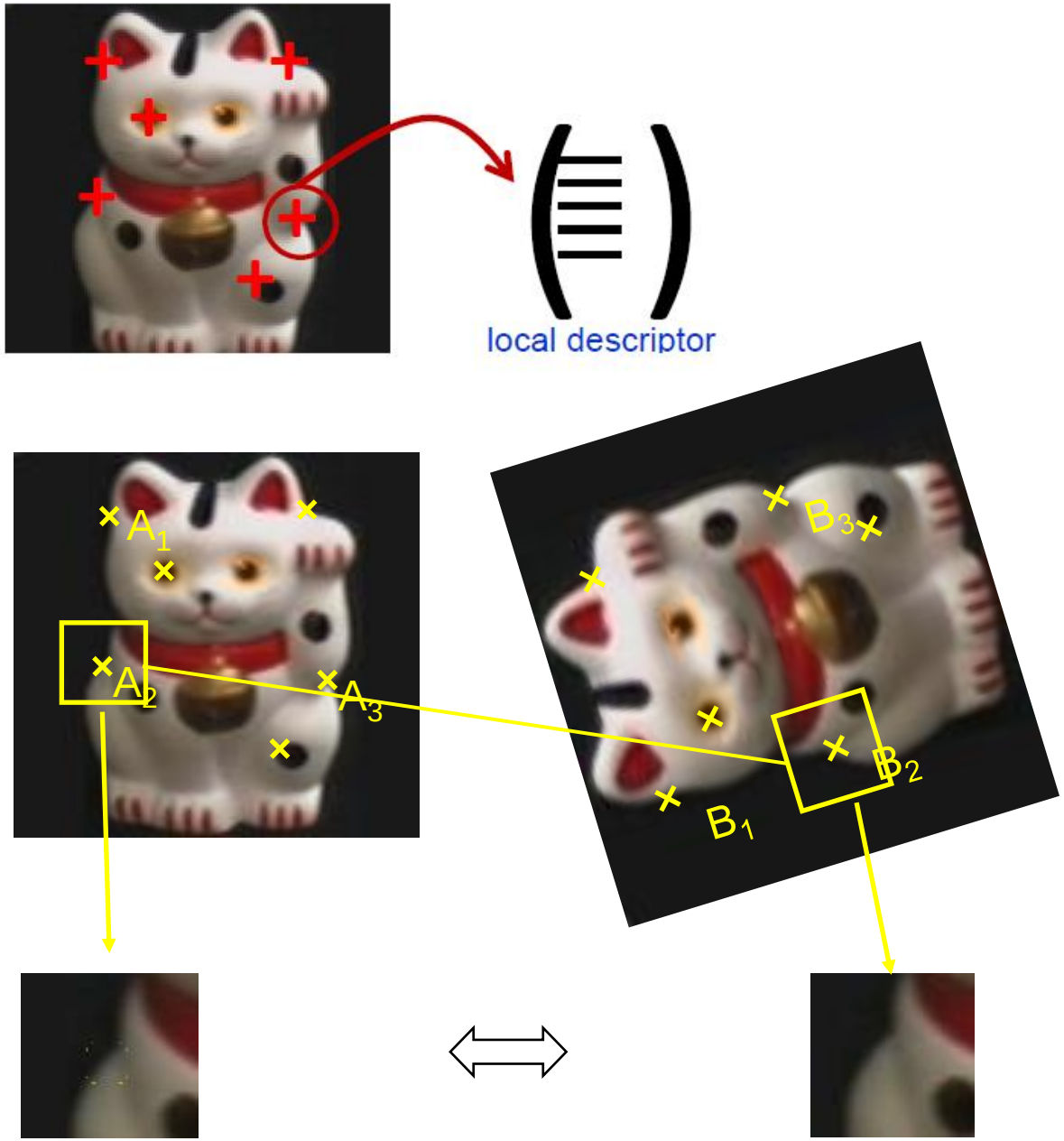



Slide credit: K. Grauman, B. Leibe

# General approach

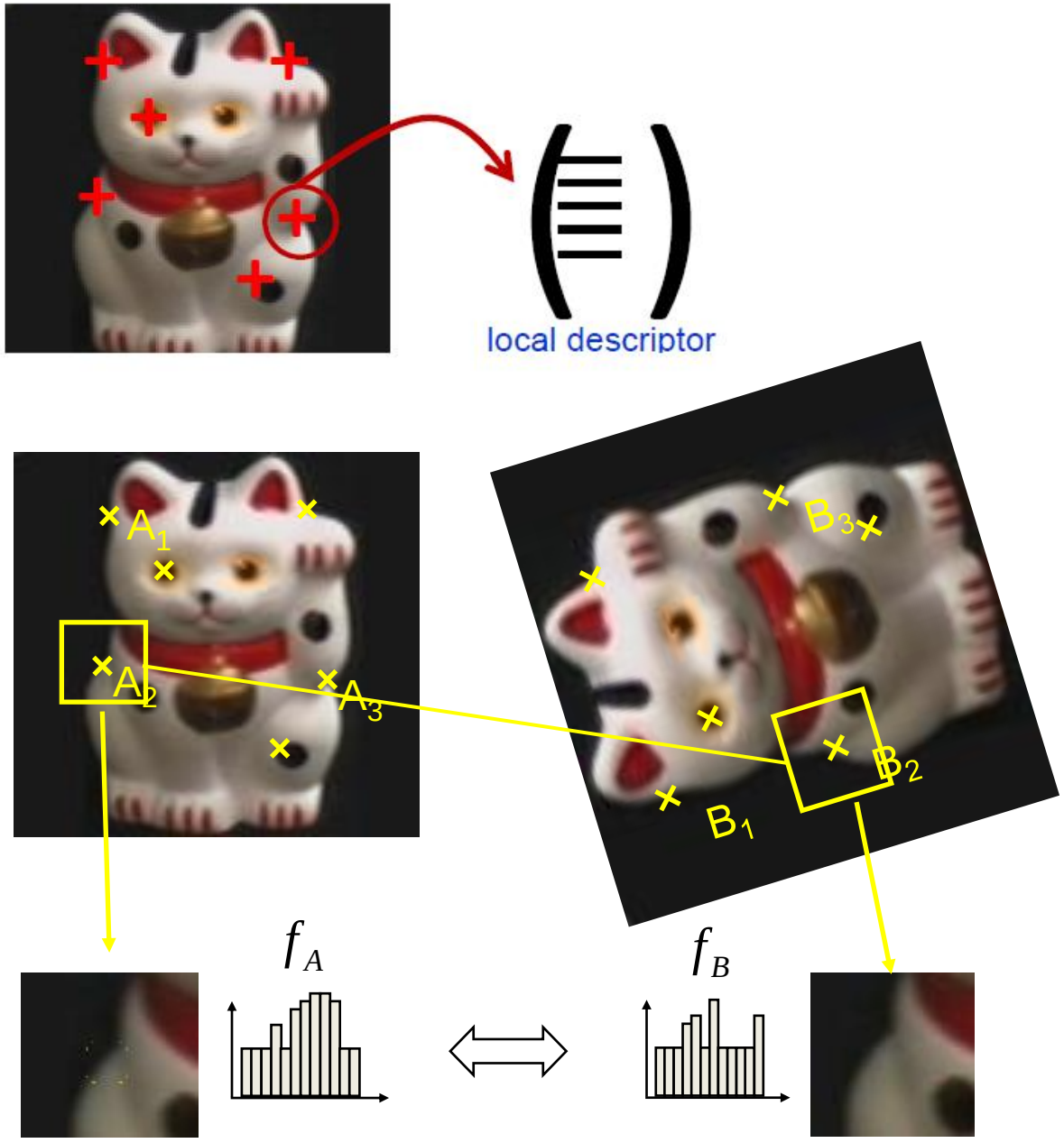
- Find the interest points
- Consider the region around each keypoint
- Compute a local descriptor from the region and normalize the feature
- Match local descriptors




Slide credit: K. Grauman, B. Leibe

# General approach

- Find the interest points
- Consider the region around each keypoint
- Compute a local descriptor from the region and normalize the feature
- Match local descriptors




Slide credit: K. Grauman, B. Leibe

# Interest points

- Note: “interest points” = “keypoints”, also sometimes called “features”
- Choosing interest points
- Where would you tell your friend to meet you?

- It is difficult to say how humans find these features. This is already programmed in our brain
- But if we look deep into some pictures and search for different patterns, we will find something interesting

# Interest points

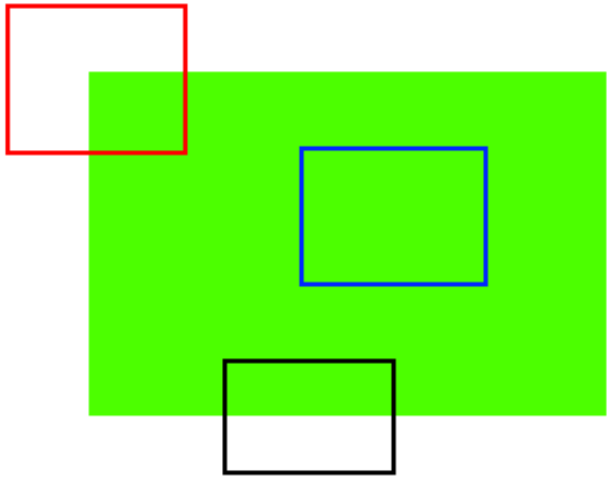
- For example, take below image

# Interest points

- The image is very simple. At the top of image, six small image patches are given
- Question for you is to find the exact location of these patches in the original image. How many correct results can you find?
- A and B are flat surfaces and they are spread over a lot of area. It is difficult to find the exact location of these patches.
- C and D are much more simple. They are edges of the building. You can find an approximate location, but exact location is still difficult. This is because the pattern is same everywhere along the edge. At the edge, however, it is different. An edge is therefore better feature compared to flat area, but not good enough (It is good in jigsaw puzzle for comparing continuity of edges).
- Finally, E and F are some corners of the building. And they can be easily found. Because at the corners, wherever you move this patch, it will look different. So they can be considered as good features.

# Interest points

- So now we move into simpler (and widely used image) for better understanding.



- Just like above, the blue patch is flat area and difficult to find and track. Wherever you move the blue patch it looks the same. The black patch has an edge. If you move it in vertical direction (i.e. along the gradient) it changes. Moved along the edge (parallel to edge), it looks the same. And for red patch, it is a corner. Wherever you move the patch, it looks different, means it is unique. So basically, corners are considered to be good features in an image. (Not just corners, in some cases blobs are considered good features).

# Feature Detection

- So now we answered our question, "what are these features?"
- But next question arises. How do we find them? Or how do we find the corners?
- We answered that in an intuitive way, i.e., look for the regions in images which have maximum variation when moved (by a small amount) in all regions around it
- So finding these image features is called **Feature Detection**

# Many Existing Detectors Available

- Hessian & Harris [Beaudet ‘78], [Harris ‘88]
- Laplacian, DoG [Lindeberg ‘98], [Lowe 1999]
- Harris-/Hessian-Laplace [Mikolajczyk & Schmid ‘01]
- Harris-/Hessian-Affine [Mikolajczyk & Schmid ‘04]
- EBR and IBR [Tuytelaars & Van Gool ‘04]
- MSER [Matas ‘02]
- Salient Regions [Kadir & Brady ‘01]
- Others…

# Harris Detector

- Corners are regions in the image with large variation in intensity in all the directions
- One early attempt to find these corners was done by Chris Harris & Mike Stephens in their paper A Combined Corner and Edge Detector in 1988, so now it is called Harris Corner Detector
- He took this simple idea to a mathematical form. It basically finds the difference in intensity for a displacement of (u,v) in all directions.

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \; [\underbrace{I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

# Harris Detector

- Window function is either a rectangular window or gaussian window which gives weights to pixels underneath
- We have to maximize the function $E(u,v)$ for corner detection
- That means, we have to maximize the second term
- Applying Taylor Expansion to above equation and using some mathematical steps, we get the final equation as:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

# Harris Detector

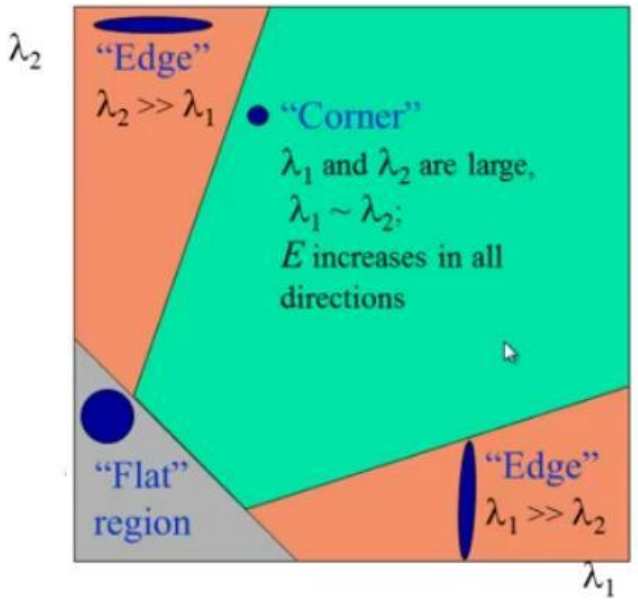
- Here, $I_x$ and $I_y$ are image derivatives in *x* and *y* directions respectively. (Can be easily found out using cv2.Sobel())
- After this, a score is created which will determine if a window can contain a corner or not

$$R = det(M) - k(trace(M))^2$$

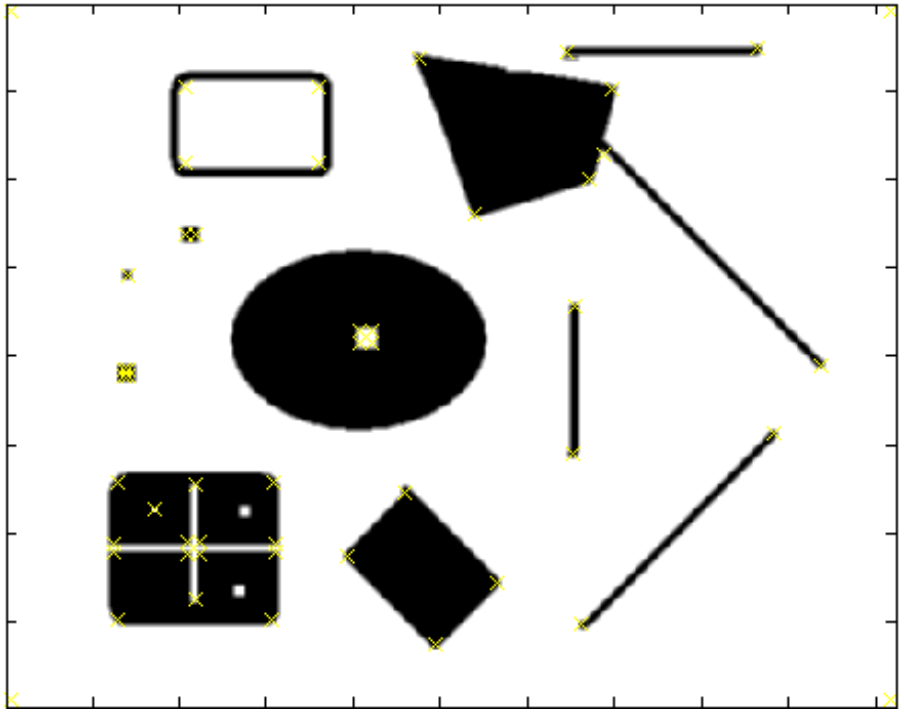
- $det(M) = \lambda_1 \lambda_2$
- $trace(M) = \lambda_1 + \lambda_2$
- $\lambda_1$ and $\lambda_2$ are the eigen values of M

# Harris Detector

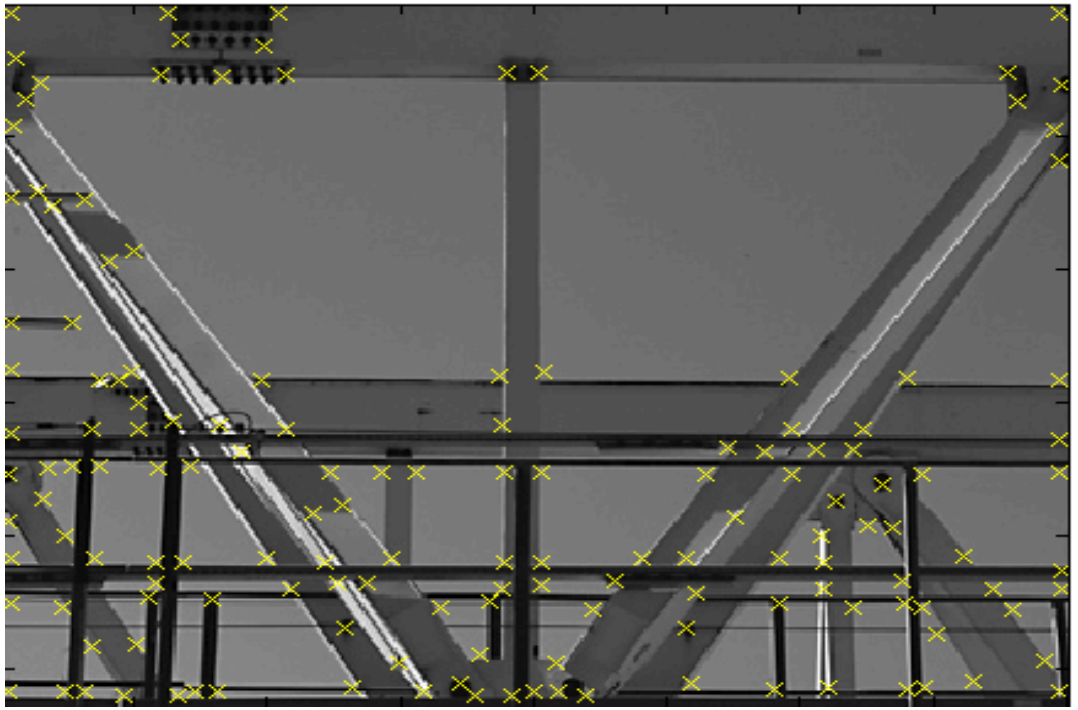
- When |R| is small, which happens when λ1 and λ2 are small, the region is flat
- When R<0, which happens when λ1>>λ2 or vice versa, the region is edge
- When R is large, which happens when λ1 and λ2 are large and λ1∼λ2, the region is a corner
- The result of Harris Corner Detection is a grayscale image with these scores. Thresholding for a suitable give you the corners in the image.



# Harris Detector – Responses



***Effect:*** **A very precise corner detector**



# Harris Detector - Responses

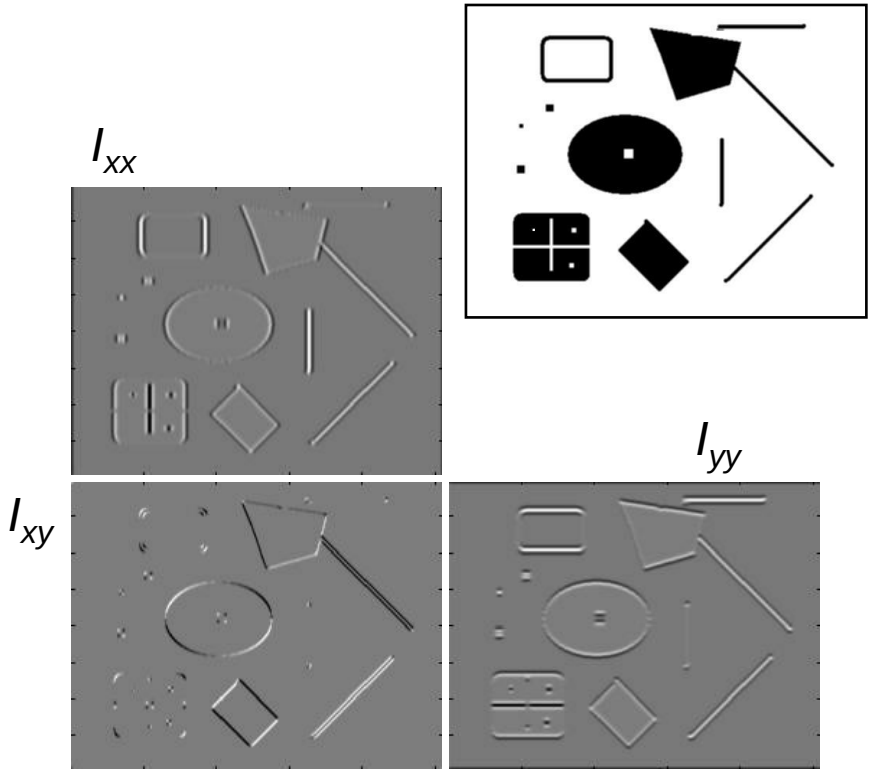
# Harris Detector

- cv2.cornerHarris(src, blockSize, ksize, k[, dst[, borderType]])
  - src – Input single-channel 8-bit or floating-point image
  - dst – Image to store the Harris detector responses. It has the type CV_32FC1 and the same size as src
  - blockSize – Neighborhood size (see the details on cornerEigenValsAndVecs() )
  - ksize – Aperture parameter for the Sobel() operator
  - k – Harris detector free parameter in the equation
  - borderType – Pixel extrapolation method. See borderInterpolate()

# Hessian Detector

- Hessian determinant

$$Hessian(I) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$$




***Intuition:*** **Search for strong curvature in two orthogonal directions**

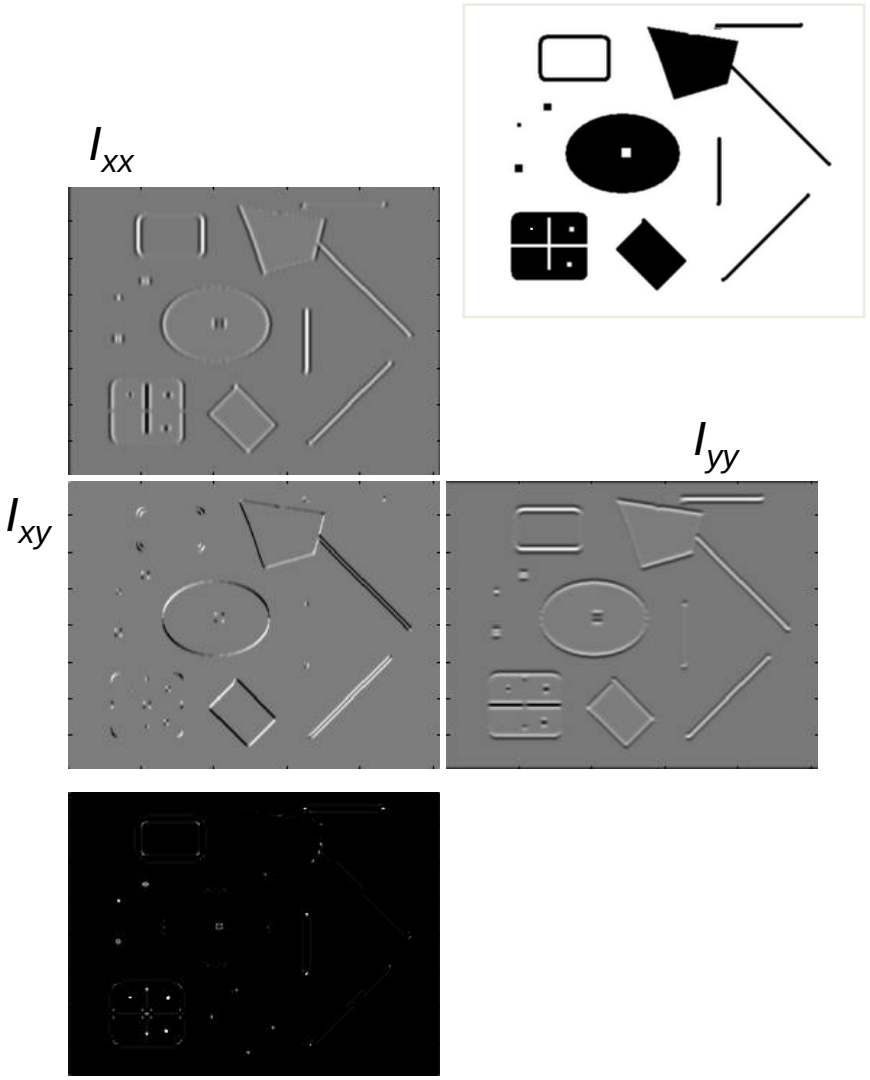
# Hessian Detector

- Hessian determinant

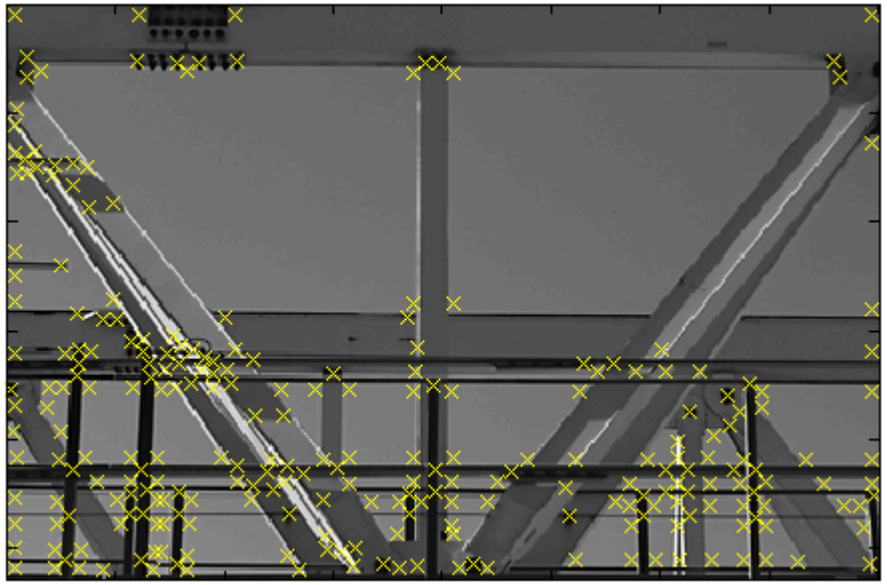
$$Hessian(I) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$$

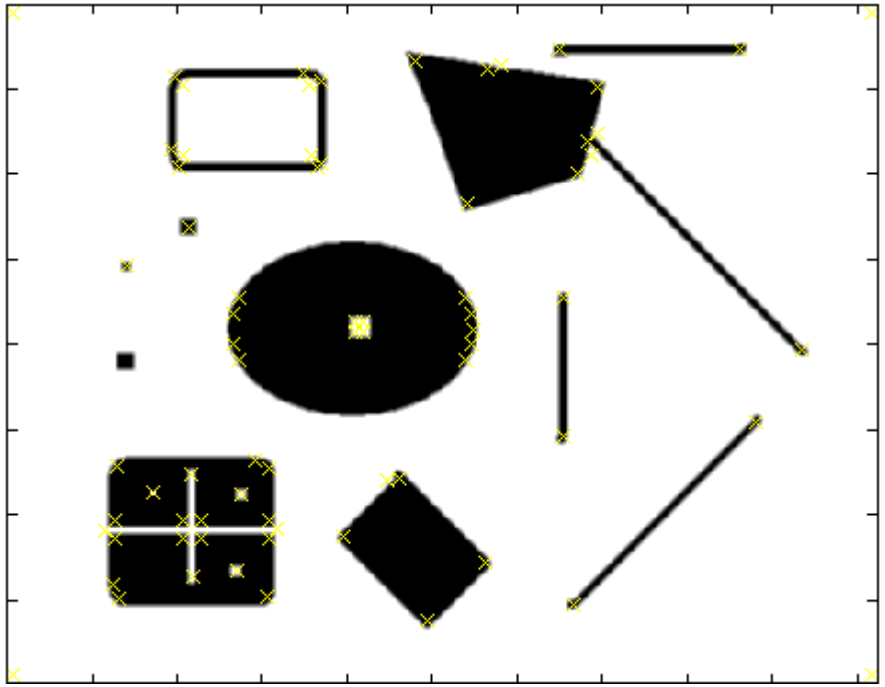
$$\det M = \lambda_1 \lambda_2$$
$$\text{trace}\, M = \lambda_1 + \lambda_2$$

$$\det(Hessian(I)) = I_{xx} I_{yy} - I_{xy}^2$$



# Hessian Detector



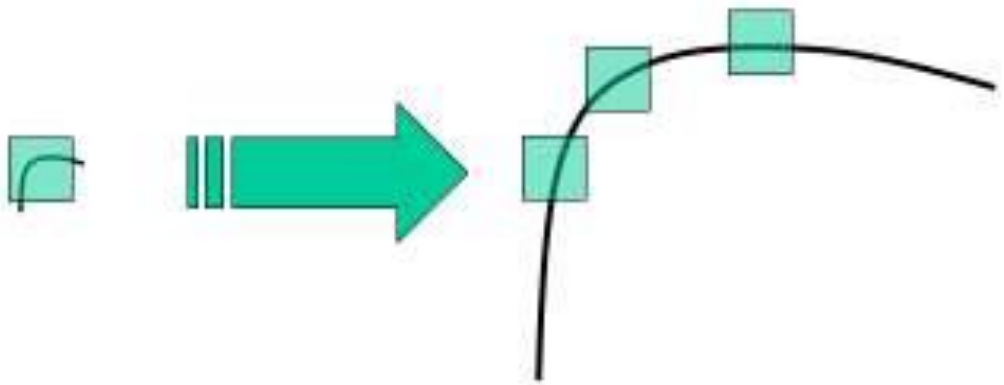


***Effect:*** **Responses mainly on corners and strongly textured areas.**

# Hessian Detector

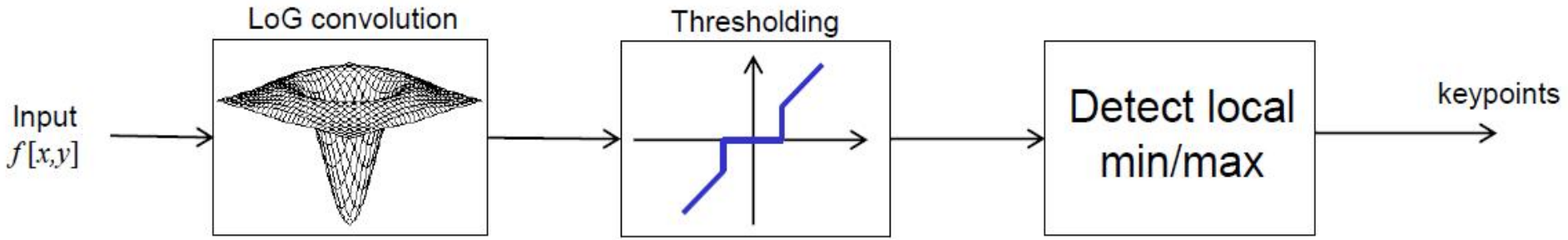
# So far: can localize in x-y, but not scale

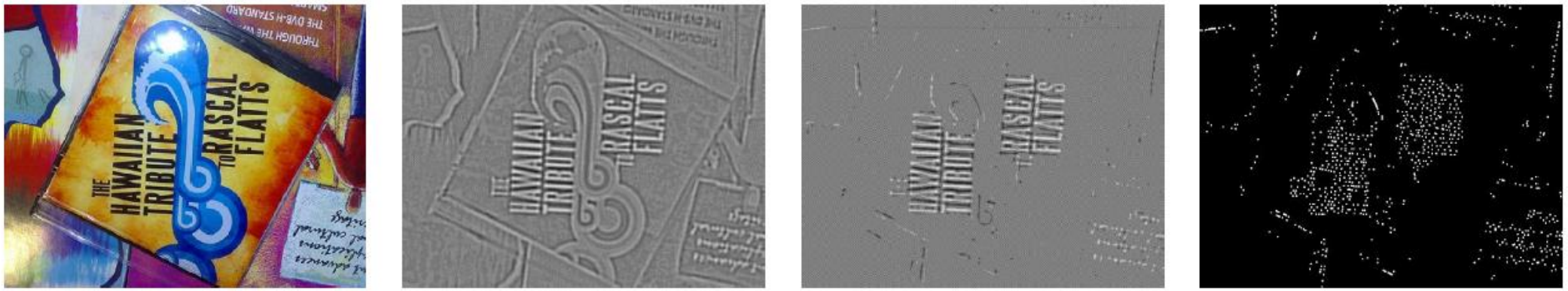
- Harris detector is rotation-invariant, which means, even if the image is rotated, we can find the same corners
- What about scaling? A corner may not be a corner if the image is scaled.
- For example, check a simple image below. A corner in a small image within a small window is flat when it is zoomed in the same window. So Harris corner is not scale invariant!



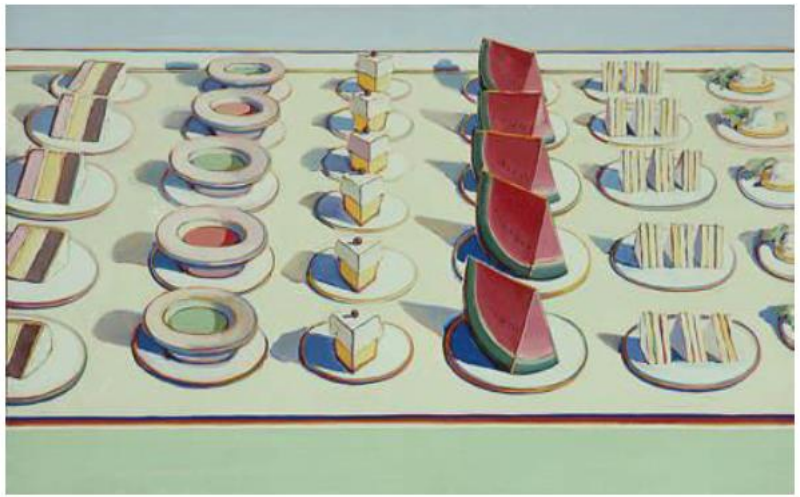
- From the image above, it is obvious that we can’t use the same window to detect keypoints with different scale
- It is good with small corner. But to detect larger corners we need larger windows
- For this, scale-space filtering is used
  - Laplacian of Gaussian is found for the image with various sigma values
  - LoG acts as a blob detector which detects blobs in various sizes due to change in sigma
  - In short, sigma acts as a scaling parameter
  - For exmaple, in the above image, gaussian kernel with low sigma gives high value for small corner while guassian kernel with high sigma fits well for larger corner
  - So, we can find the local maxima across the scale and space which gives us a list of (x,y,sigma) values which means there is a potential keypoint at (x,y) at sigma scale

# Laplacian keypoint detector

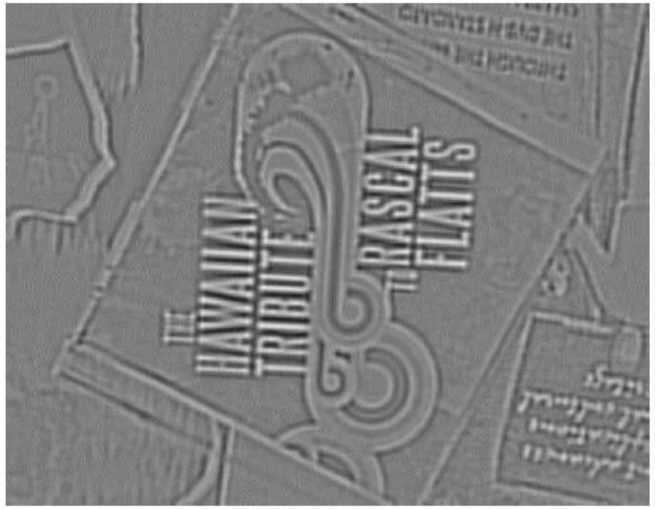


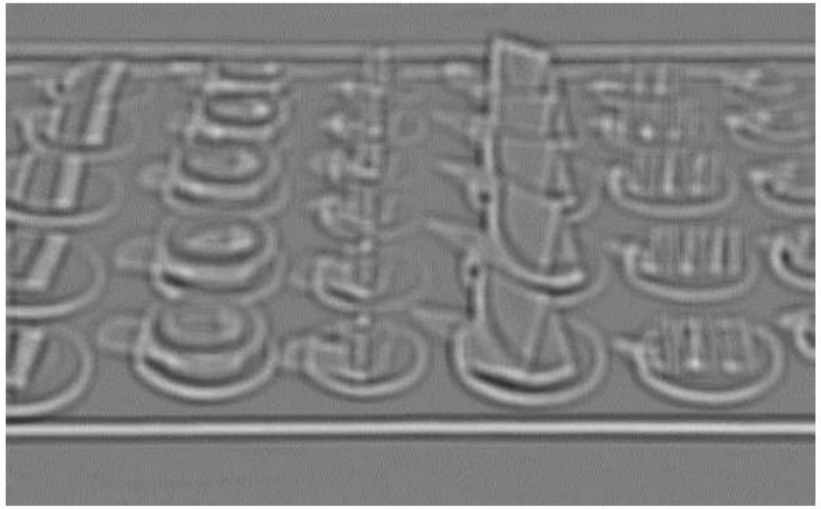



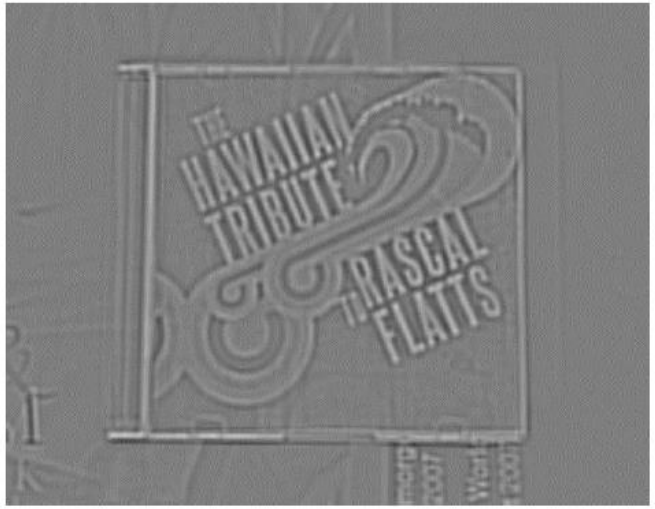
# Example images

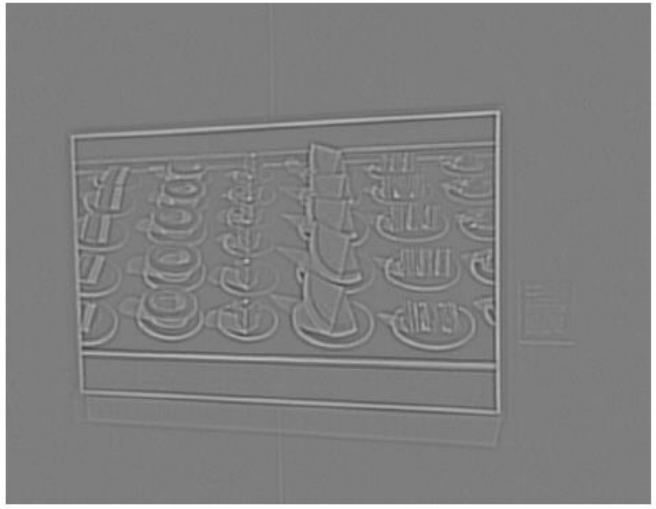




# LoG response

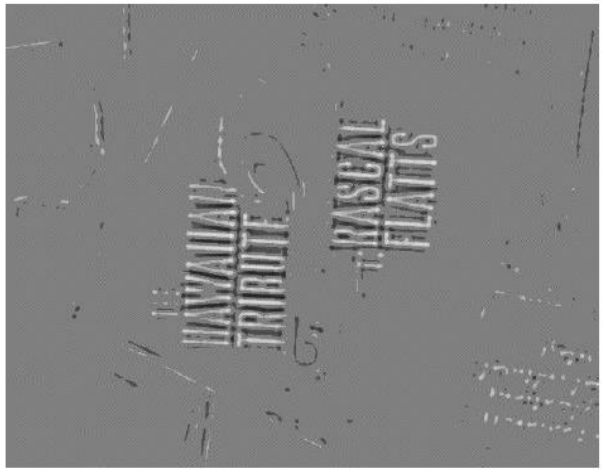


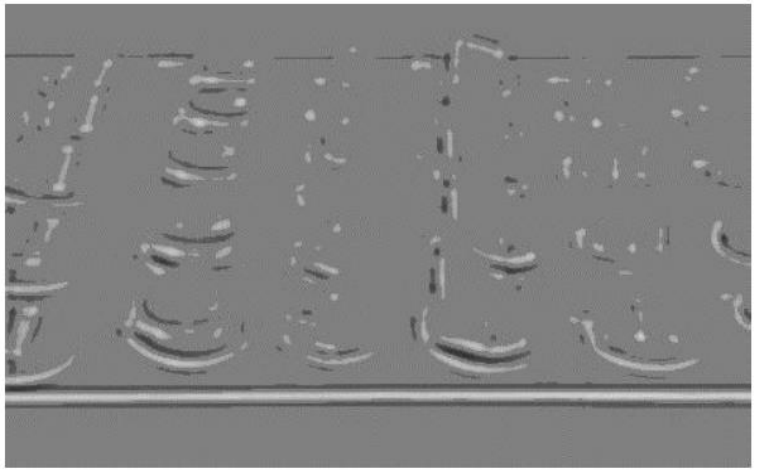



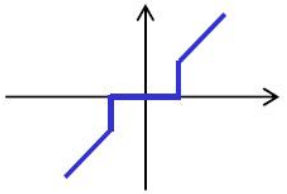


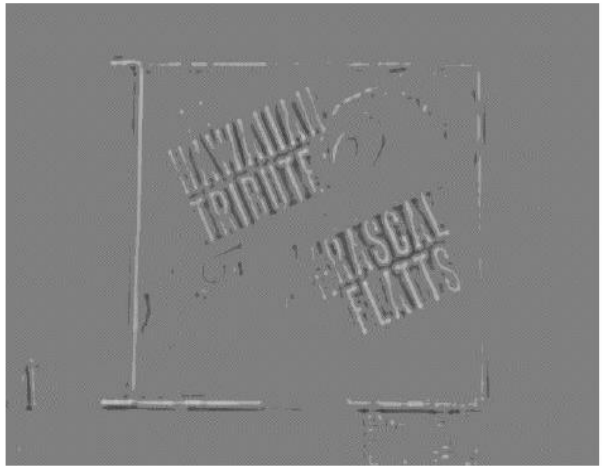



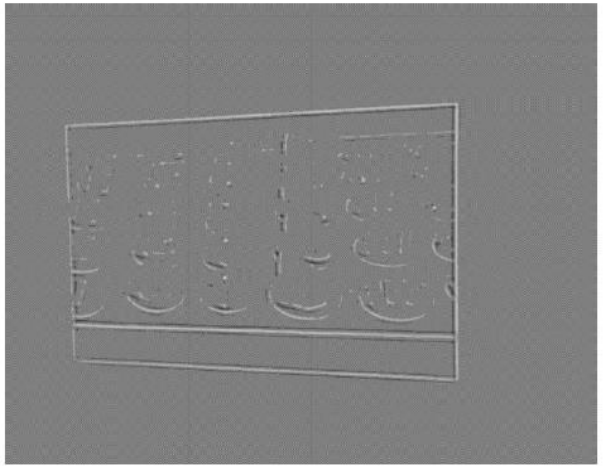
# Thresholded LoG response



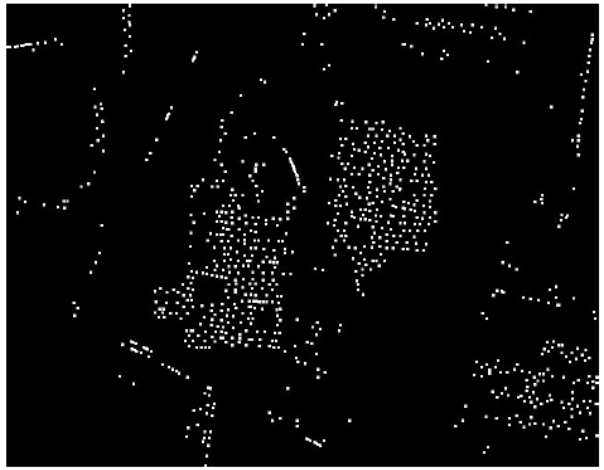



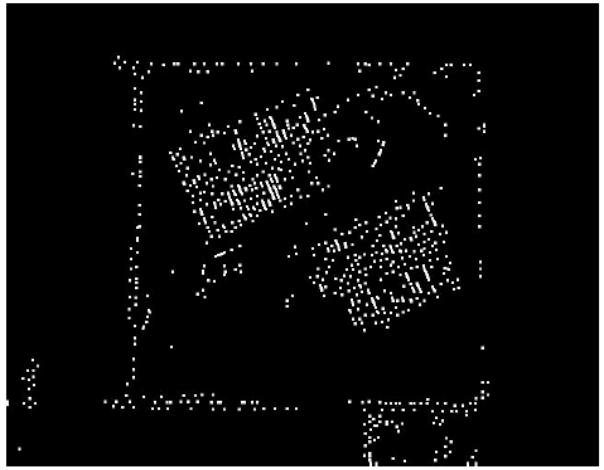
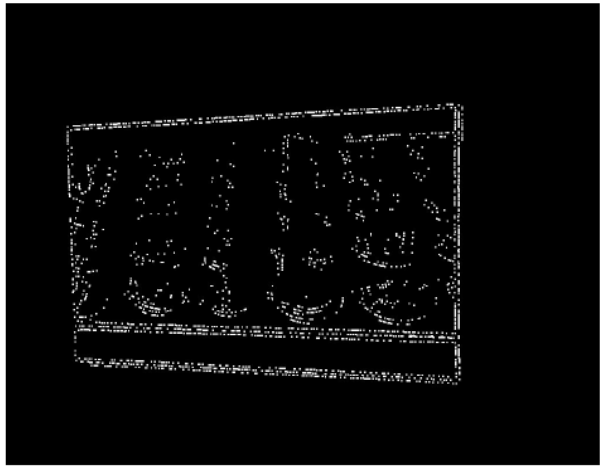




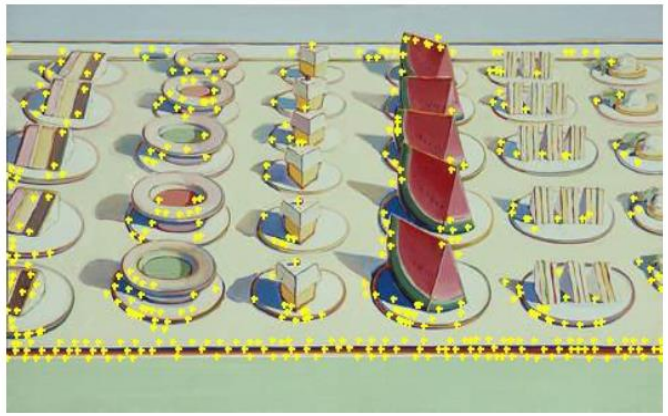



# Local extrema of thresholded LoG response

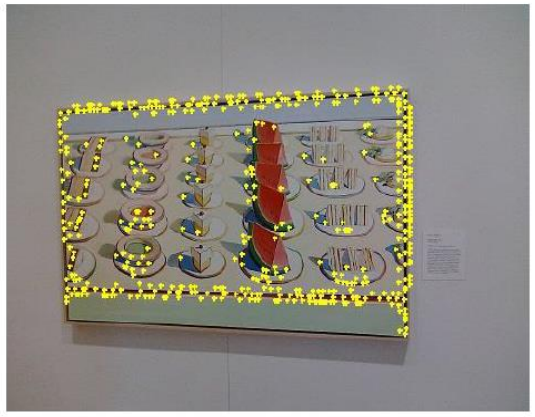


# Superimposed LoG keypoints



500 strongest keypoints



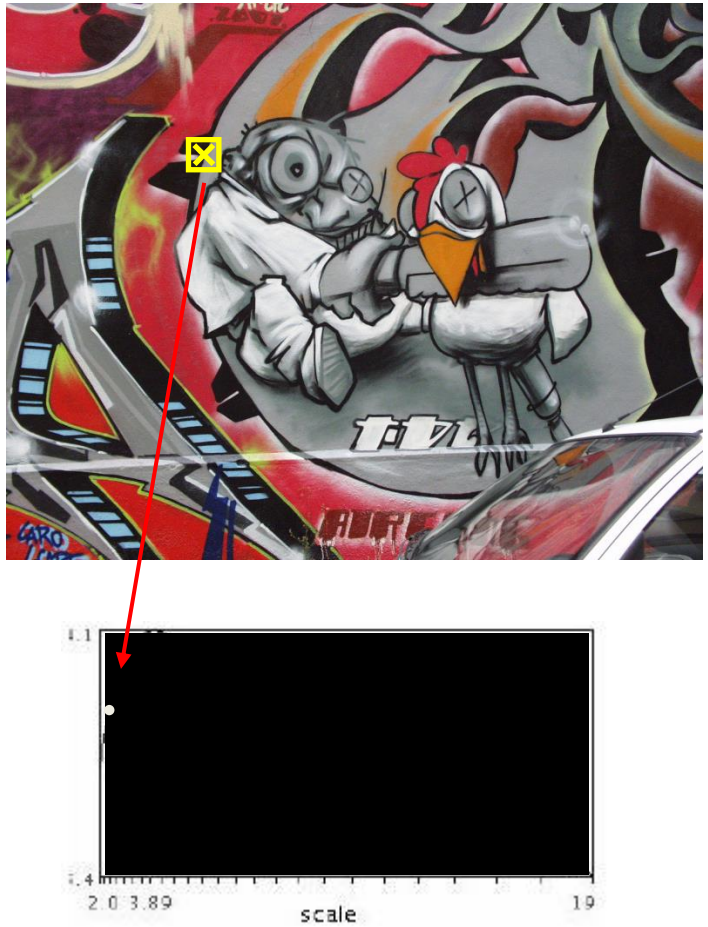
# Automatic Scale Selection

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

How to find corresponding patch sizes?

# Automatic Scale Selection

- Function responses for increasing scale (scale signature)

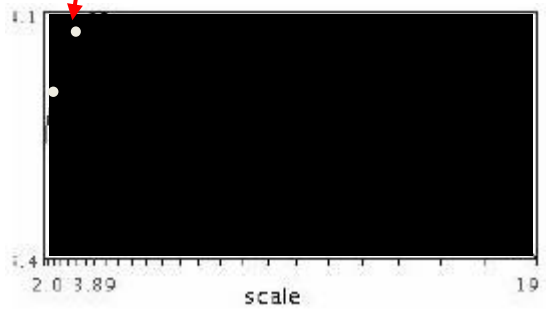



$f(I_{i_1 \ldots i_m}(x,\sigma))$

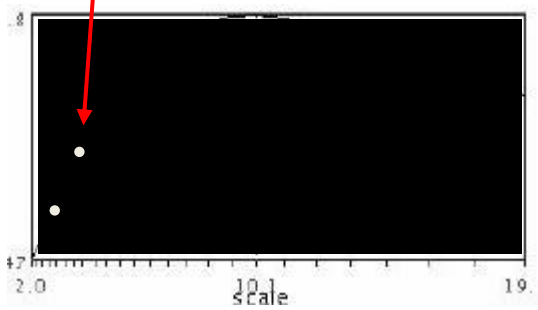

$f(I_{i_1 \ldots i_m}(x',\sigma))$

# Automatic Scale Selection

- Function responses for increasing scale (scale signature)



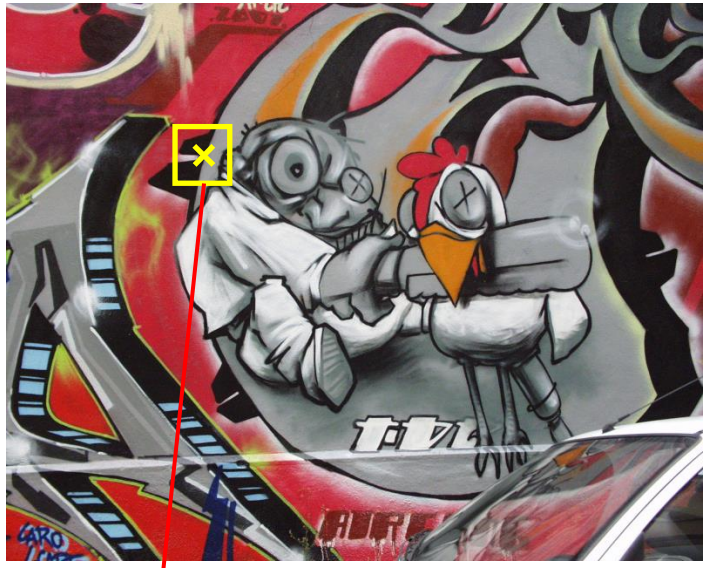

$f(I_{i_1 \ldots i_m}(x,\sigma))$

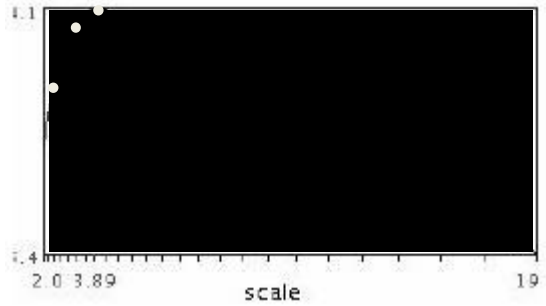



$f(I_{i_1 \ldots i_m}(x',\sigma))$

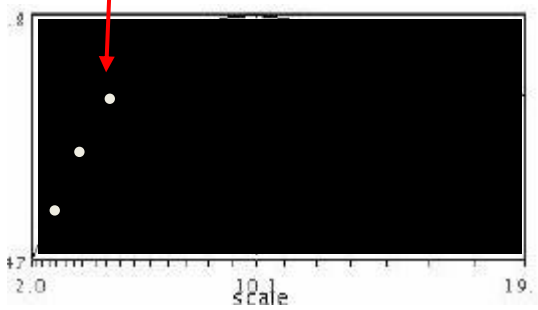
# Automatic Scale Selection

- Function responses for increasing scale (scale signature)





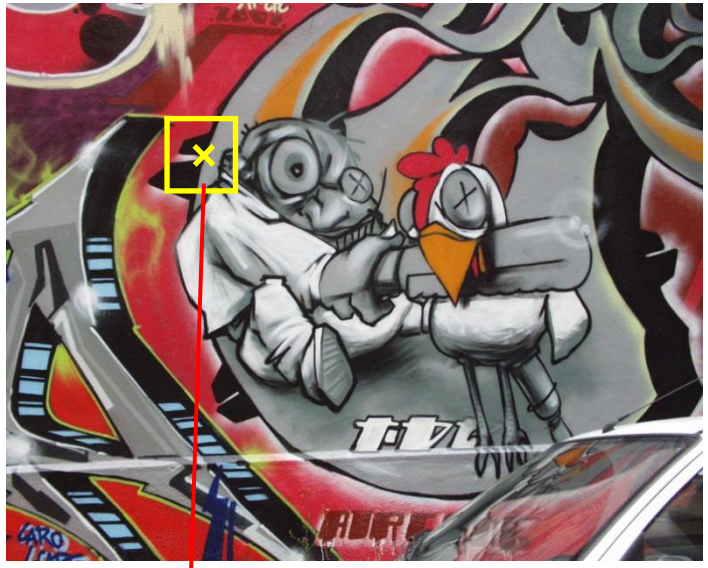

$f(I_{i_1 \ldots i_m}(x,\sigma))$

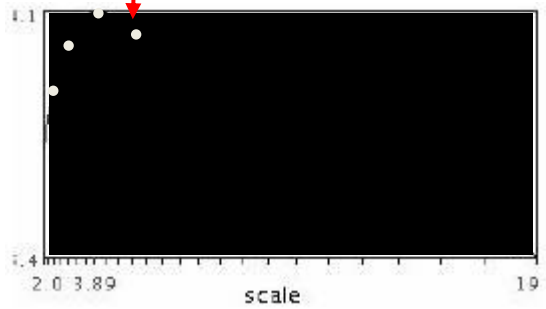



$f(I_{i_1 \ldots i_m}(x',\sigma))$

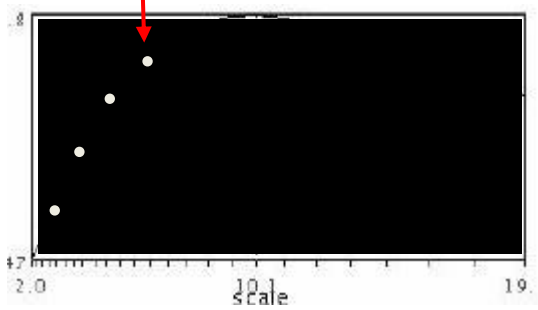
# Automatic Scale Selection

- Function responses for increasing scale (scale signature)





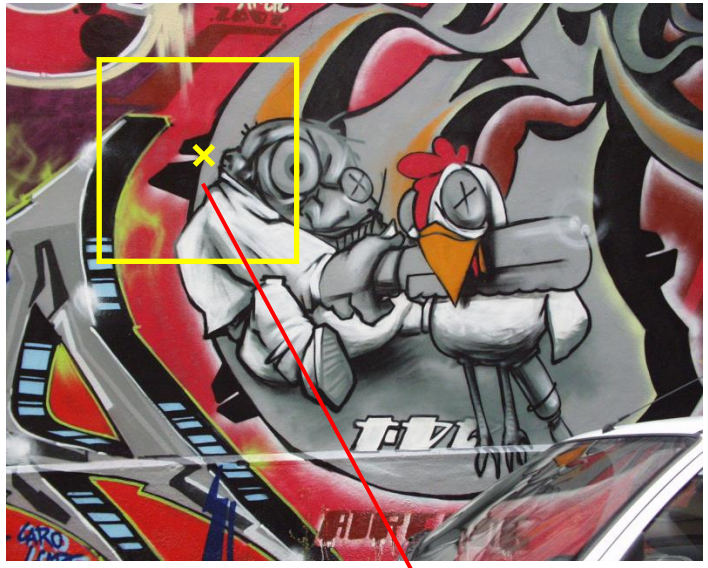

$f(I_{i_1 \ldots i_m}(x,\sigma))$

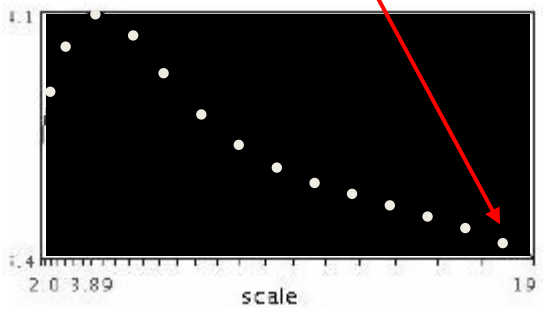



$f(I_{i_1 \ldots i_m}(x',\sigma))$

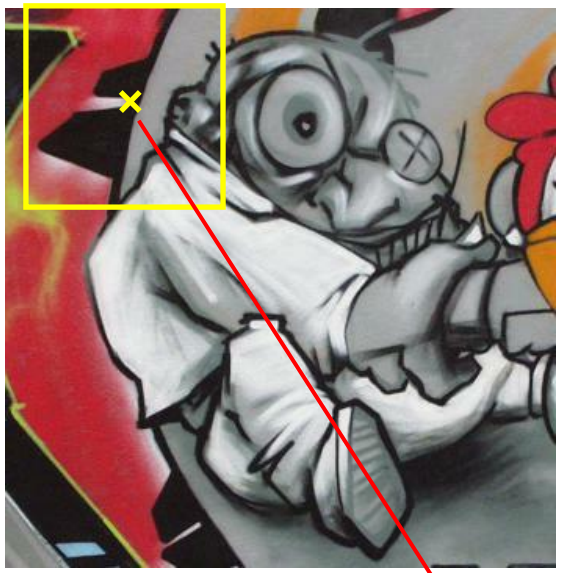
# Automatic Scale Selection

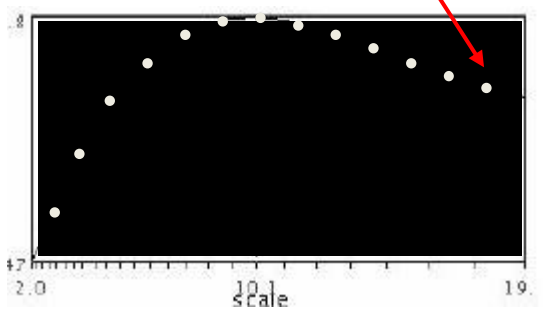
- Function responses for increasing scale (scale signature)






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

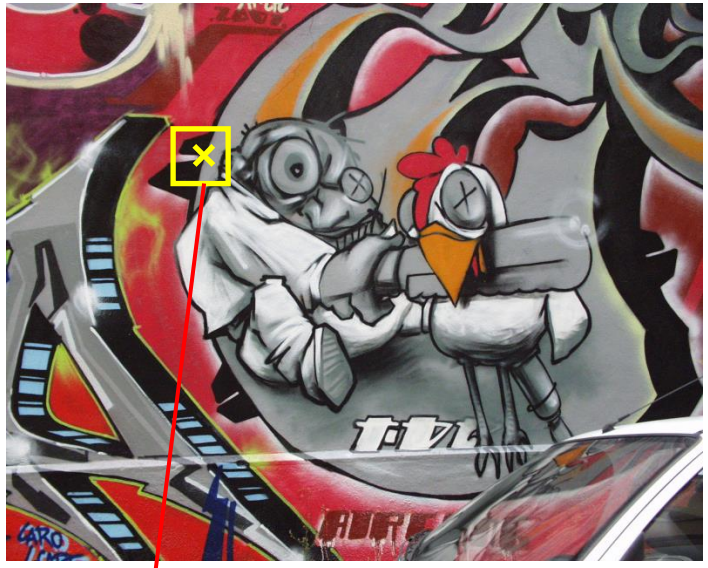


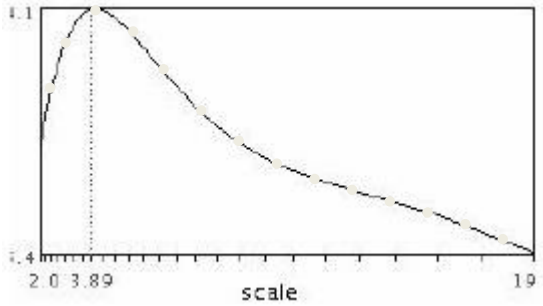



$f(I_{i_1 \ldots i_m}(x',\sigma))$

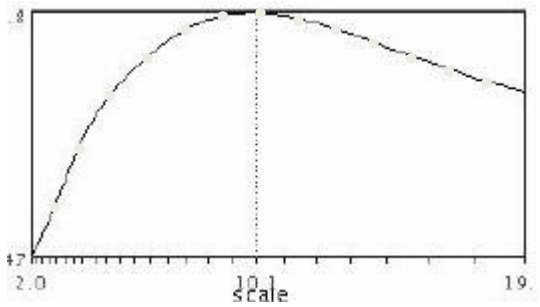
# Automatic Scale Selection

- Function responses for increasing scale (scale signature)






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



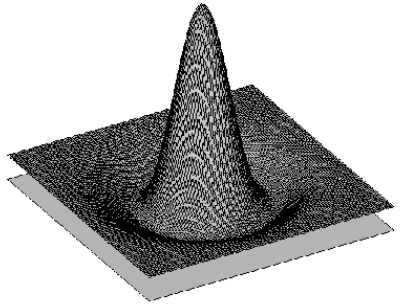

$f(I_{i_1 \ldots i_m}(x',\sigma'))$

# Laplacian keypoint detector

- LoG is a costly
- Use Difference of Gaussians which is an approximation of LoG
- Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different sigma parameters

# Difference-of-Gaussian (DoG)

Difference of Gaussian is obtained as the difference of Gaussian blurring of an image with two different sigma values



# SIFT (Scale-Invariant Feature Transform)

## David Lowe

FOLLOW

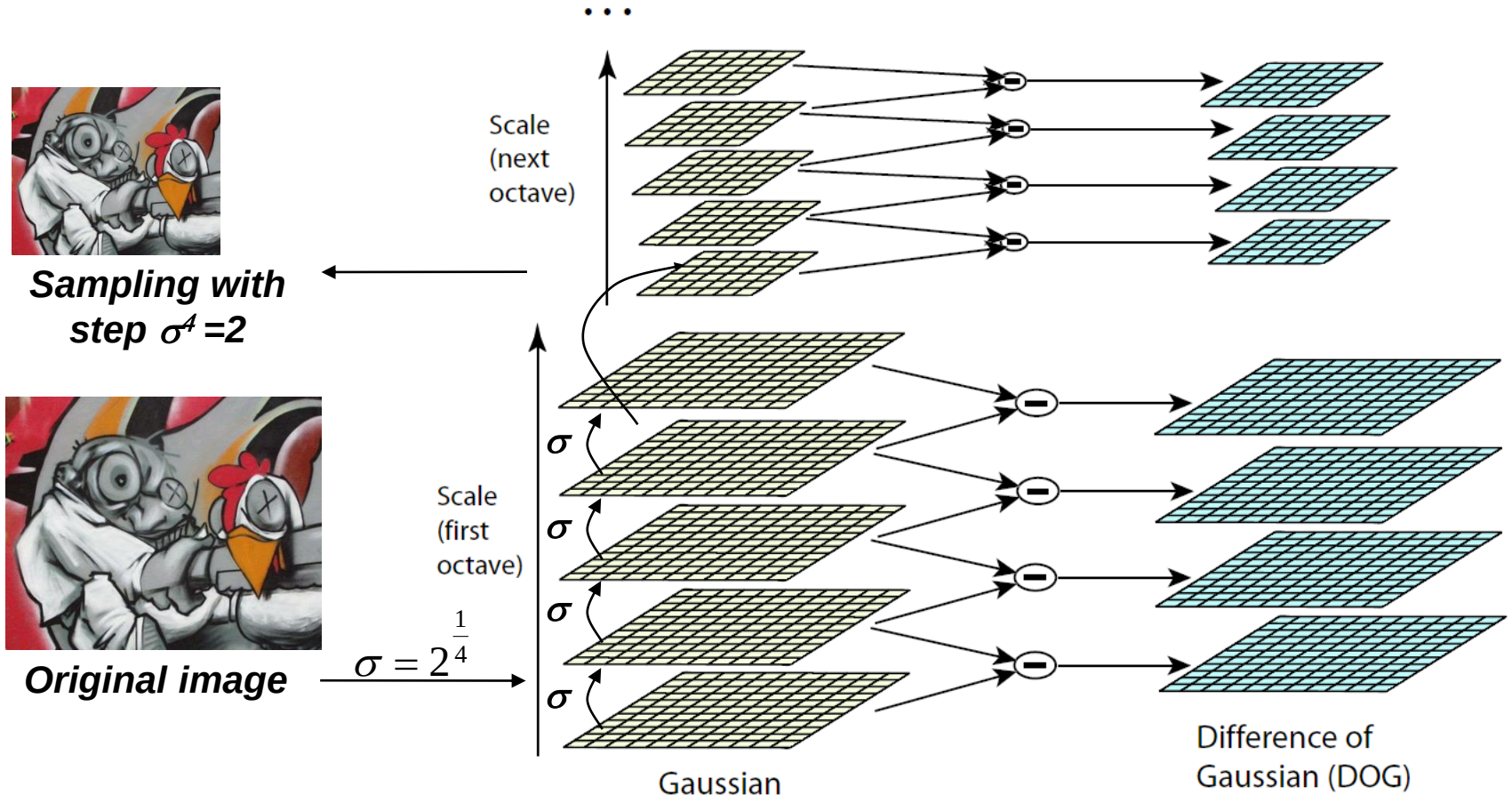
Computer Science Dept., University of British Columbia
Verified email at cs.ubc.ca - Homepage

Computer Vision  Object Recognition

| TITLE | CITED BY | YEAR |
|---|---|---|
| Distinctive image features from scale-invariant keypoints<br>DG Lowe<br>International journal of computer vision 60 (2), 91-110 | 56545 | 2004 |

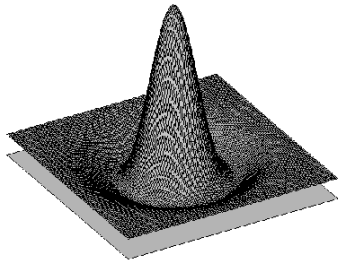
# DoG – Efficient Computation

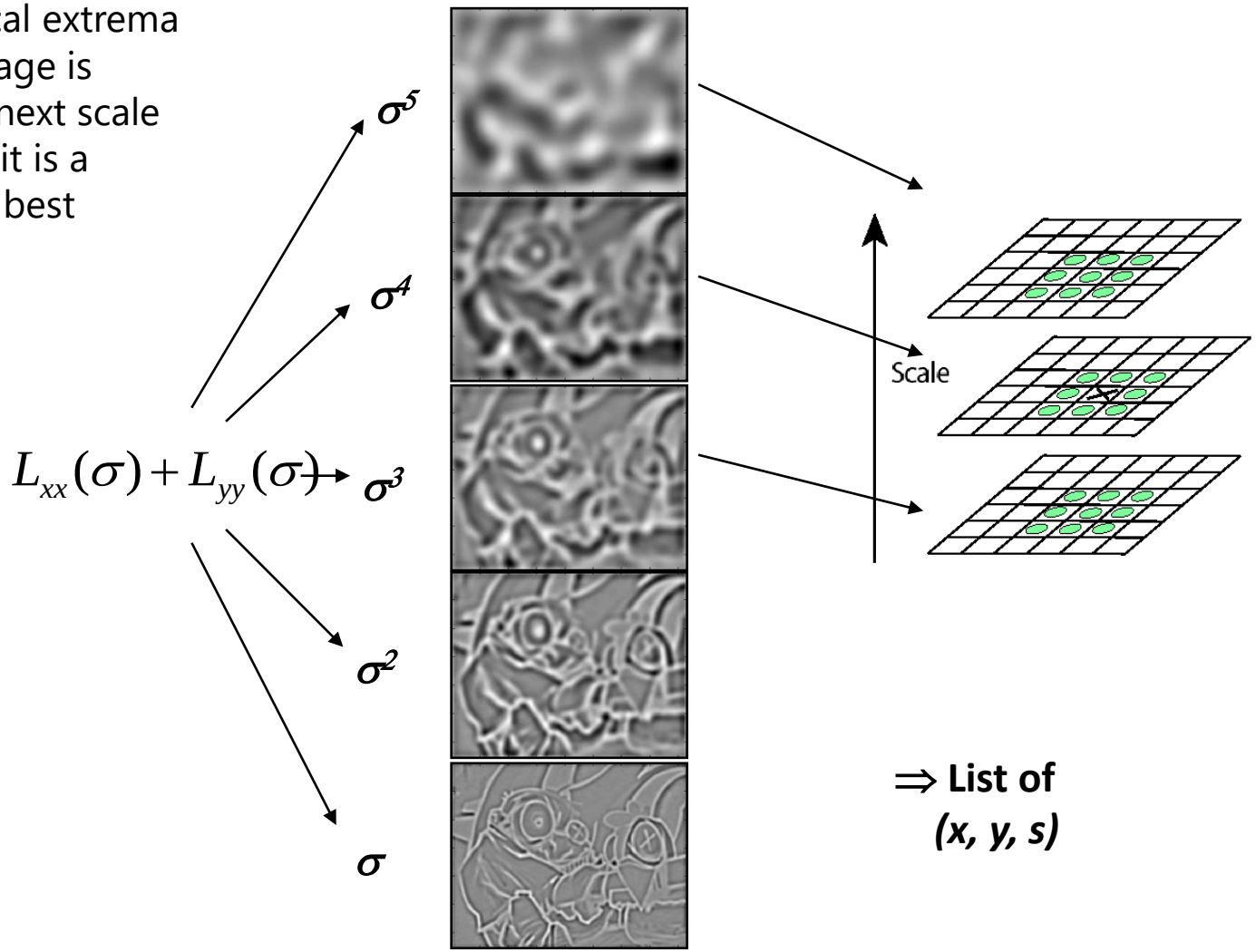
- This process is done for different octaves of the image in Gaussian Pyramid



# Find local maxima in position-scale space of Difference-of-Gaussian

Once this DoG are found, images are searched for local extrema over scale and space. For example, one pixel in an image is compared with its 8 neighbours as well as 9 pixels in next scale and 9 pixels in previous scales. If it is a local extrema, it is a potential keypoint. It basically means that keypoint is best represented in that scale





# Difference-of-Gaussian (DoG) - parameters

- Regarding different parameters, the paper gives some empirical data which can be summarized as:
  - number of octaves = 4
  - number of scale levels = 5
  - initial sigma=1.6
  - k=sqrt{2}

# SIFT keypoints localization

- Once potential keypoints locations are found, they have to be refined to get more accurate results. They used Taylor series expansion of scale space to get more accurate location of extrema, and if the intensity at this extrema is less than a threshold value (0.03 as per the paper), it is rejected. This threshold is called contrastThreshold in OpenCV

- DoG has higher response for edges, so edges also need to be removed. For this, a concept similar to Harris corner detector is used. They used a 2x2 Hessian matrix (H) to compute the pricipal curvature. We know from Harris corner detector that for edges, one eigen value is larger than the other. So here they used a simple function:

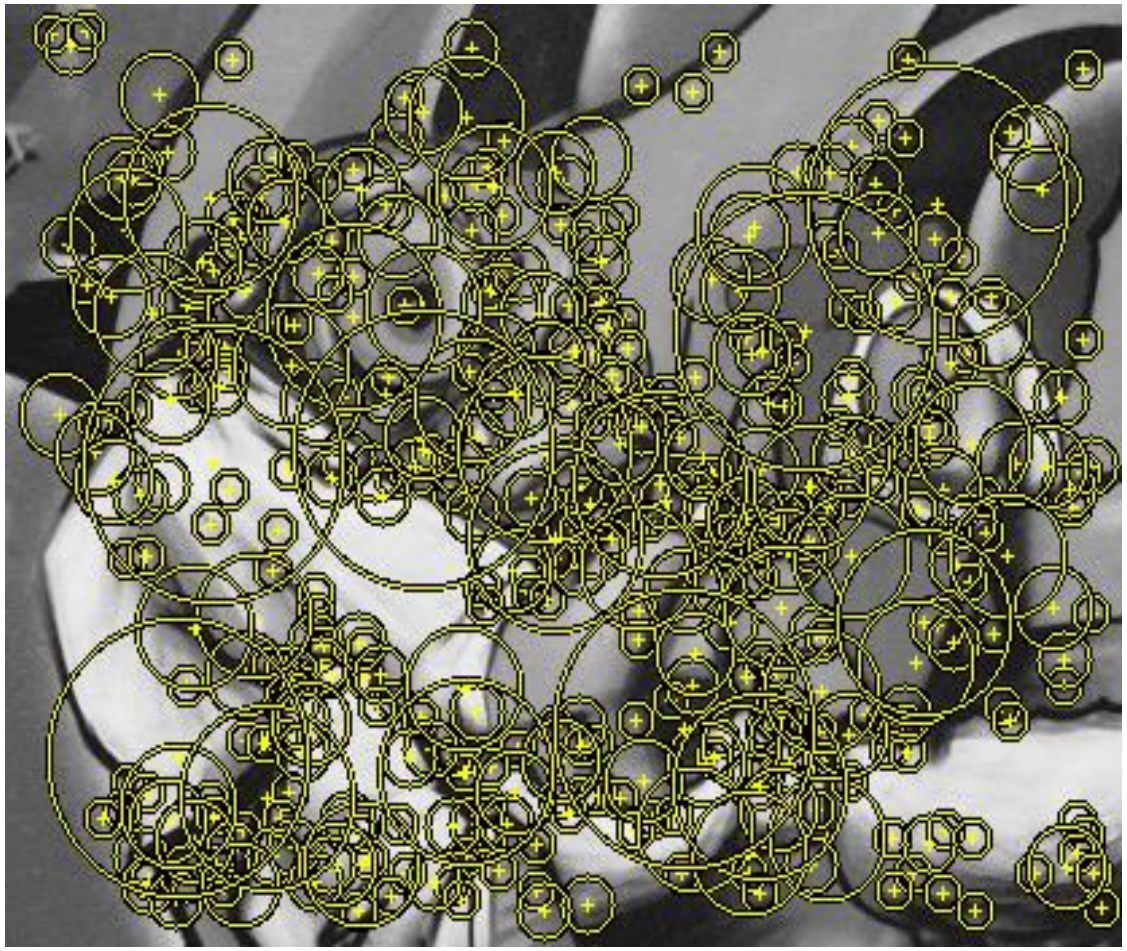
  - If this ratio is greater than a threshold, called edgeThreshold in OpenCV, that keypoint is discarded. It is given as 10 in paper

- So it eliminates any low-contrast keypoints and edge keypoints and what remains is strong interest points

# Orientation Assignment

- Orientation is assigned to each keypoint to achieve invariance to image rotation
- A neigbourhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction is calculated in that region
- An orientation histogram with 36 bins covering 360 degrees is created
- It is weighted by gradient magnitude and gaussian-weighted circular window with sigma equal to 1.5 times the scale of keypoint
- The highest peak in the histogram is taken and any peak above 80% of it is also considered to calculate the orientation
- It creates keypoints with same location and scale, but different directions to contribute to stability of matching

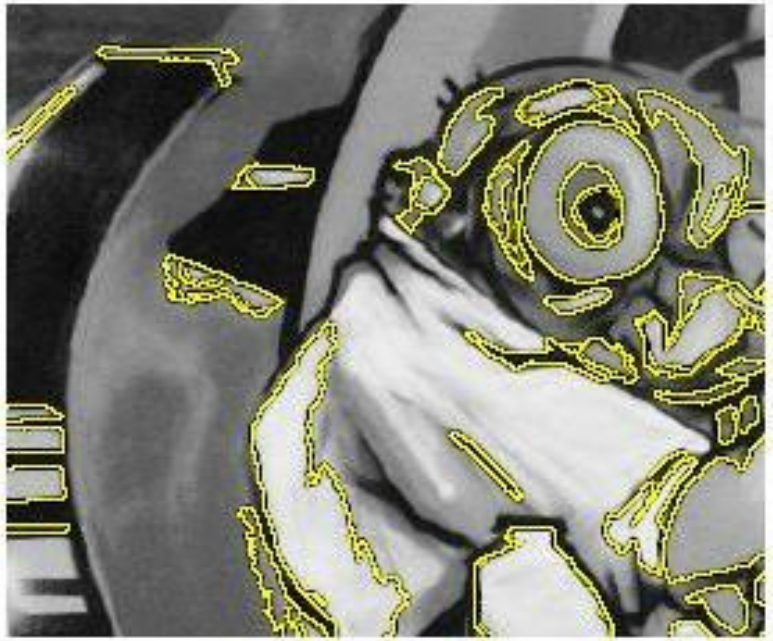
# Results: Difference-of-Gaussian



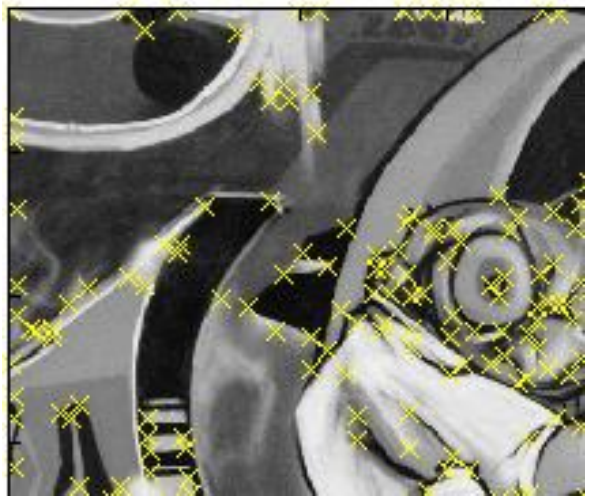
# Maximally Stable Extremal Regions

- Based on Watershed segmentation algorithm
- Select regions that stay stable over a large parameter range

# Example Results: MSER

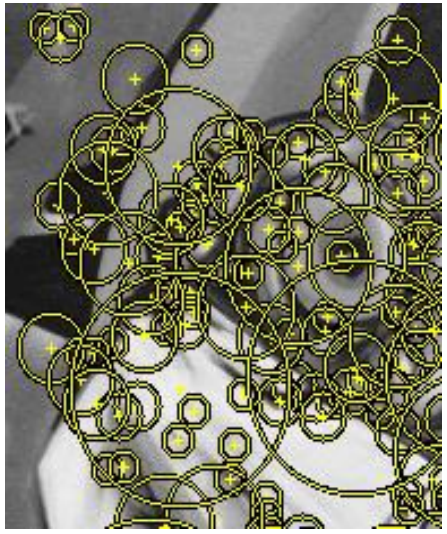



# Comparison

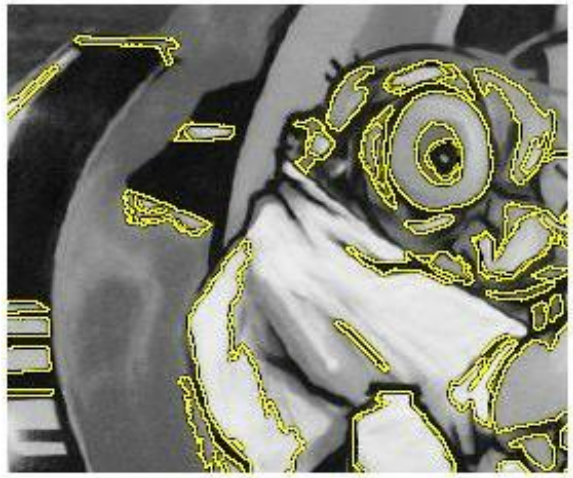


Harris

Hessian



LoG



MSER

# Available at a web sites…

- For most local feature detectors, executables are 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

# Choosing a detector

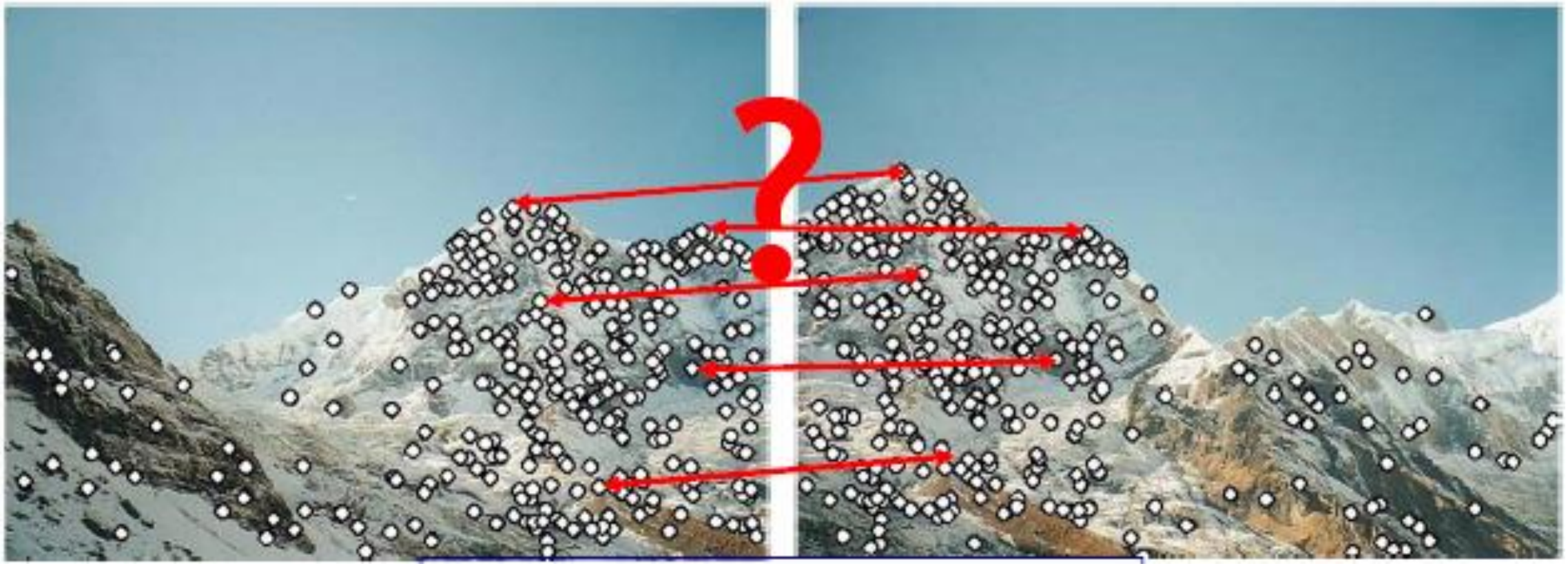
- What do you want it for?
  - Precise localization in x-y: Harris
  - Good localization in scale: Difference of Gaussian
  - Flexible region shape: MSER

- Best choice often application dependent
  - Harris-/Hessian-Laplace/DoG work well for many natural categories
  - MSER works well for buildings and printed things

- Why choose?
  - Get more points with more detectors

- There have been extensive evaluations/comparisons
  - [Mikolajczyk et al., IJCV'05, PAMI'05]
  - All detectors/descriptors shown here work well

# Feature Description



How to match the points across different images of the same object?

# Feature Description

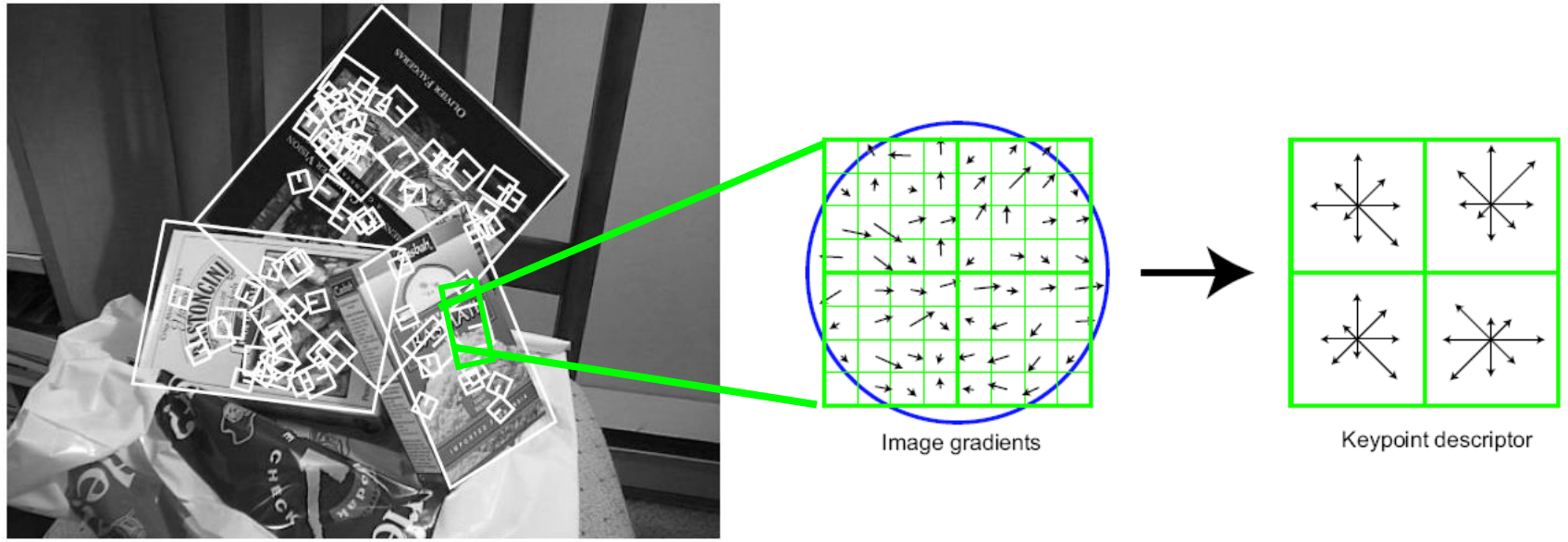
- We found the features in the images. Once you have found it, you should be able to find the same in the other images. How is this done?
- We take a region around the feature, we explain it in our own words, like "upper part is blue sky, lower part is region from a building, on that building there is glass etc" and you search for the same area in the other images.
- Basically, you are describing the feature. Similarly, a computer also should describe the region around the feature so that it can find it in other images.
- So called description is called **Feature Description**. Once you have the features and its description, you can find same features in all images and align them, stitch them together or do whatever you want.

# Local Descriptors

- The ideal descriptor should be
  - Robust
  - Distinctive
  - Compact
  - Efficient

- Most available descriptors focus on edge/gradient information
  - Capture texture information
  - Color rarely used

# Local Descriptors: SIFT (Scale-Invariant Feature Transform) Descriptor




Histogram of oriented gradients

- Captures important texture information
- Robust to small translations / affine deformations

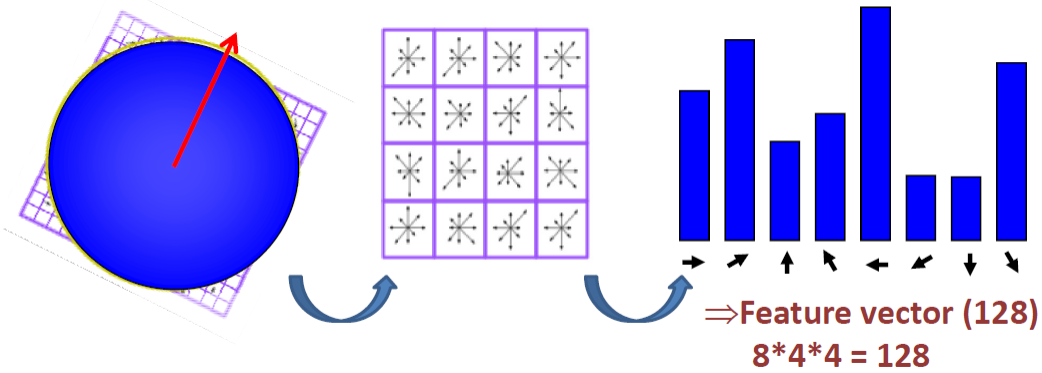
[Lowe, ICCV 1999]

# Details of Lowe's SIFT algorithm

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r}$$

- Run DoG detector
  - Find maxima in location/scale space
  - Remove edge points
- Find all major orientations
  - Bin orientations into 36 bin histogram
    - Weight by gradient magnitude
    - Weight by distance to center (Gaussian-weighted mean)
  - Return orientations within 0.8 of peak
    - Use parabola for better orientation fit



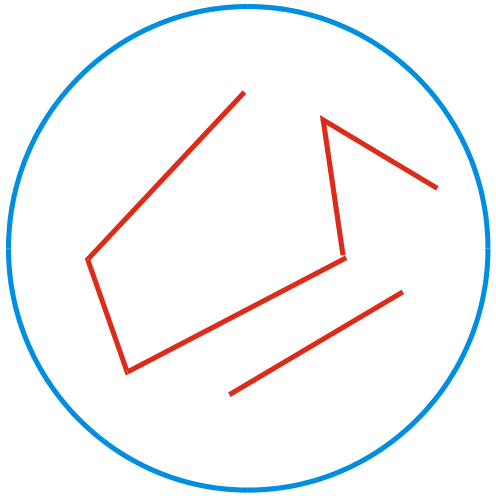

- For each (x,y,scale,orientation), create descriptor:
  - Consider a rectangular grid 16*16 in the direction of the dominant orientation of the region
  - Divide the region into 4*4 sub-regions
  - For each sub-block, 8 bin orientation histogram is created
  - Threshold values to max of 0.2, divide by L2 norm to make illumination invariant
  - Final descriptor: 4x4x8 normalized histograms

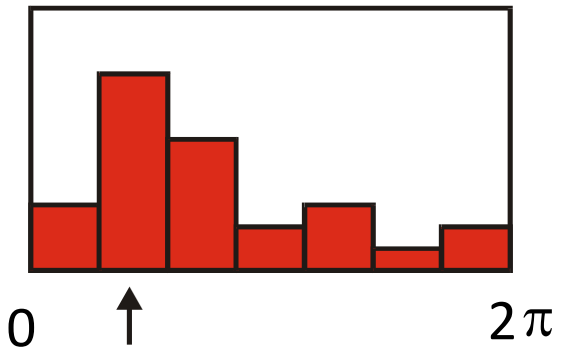
Lowe IJCV 2004

# Orientation Normalization

- Compute orientation histogram
- Select dominant orientation
- Normalize: rotate to fixed orientation

[Lowe, SIFT, 1999]

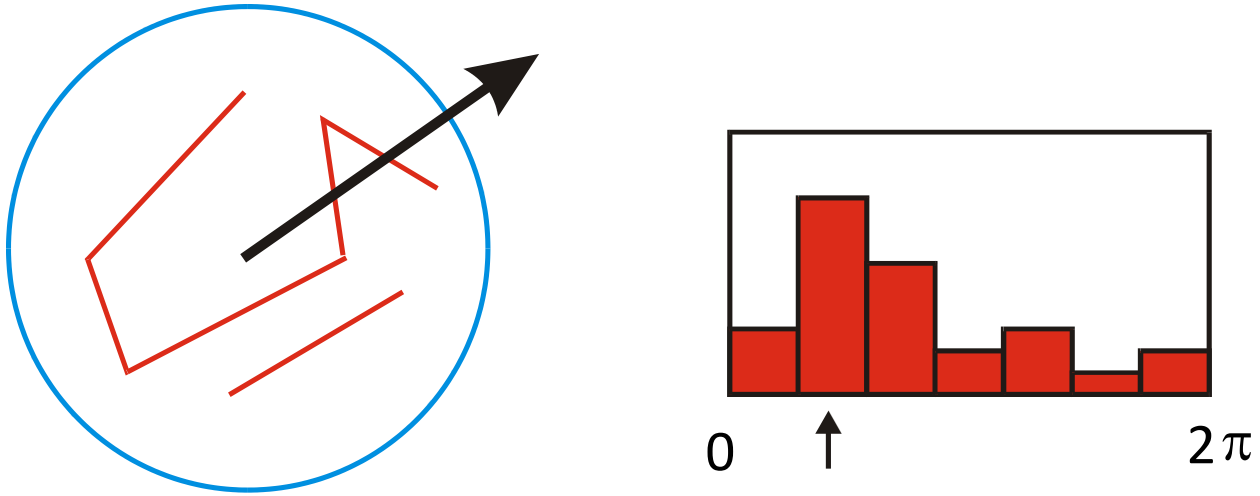




# Orientation Normalization

- Compute orientation histogram
- Select dominant orientation
- Normalize: rotate to fixed orientation

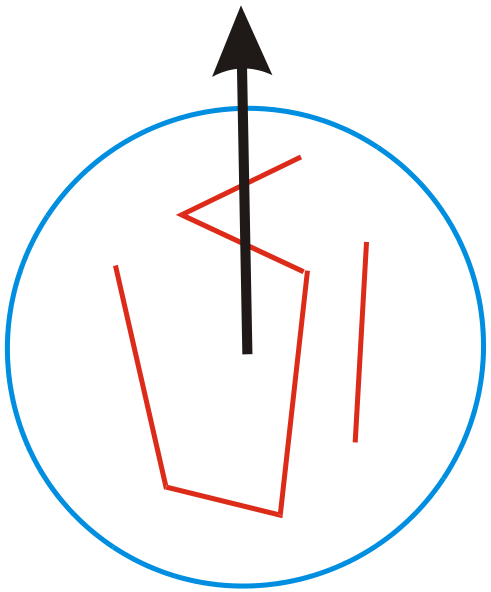
[Lowe, SIFT, 1999]

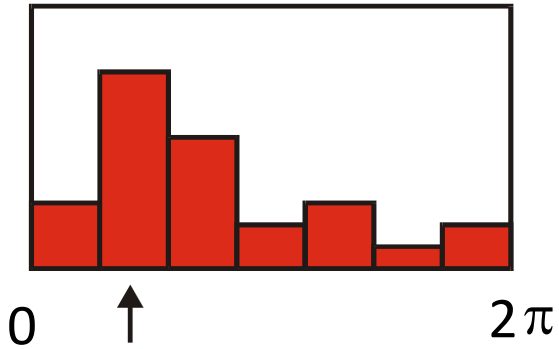


# Orientation Normalization

- Compute orientation histogram
- Select dominant orientation
- Normalize: rotate to fixed orientation

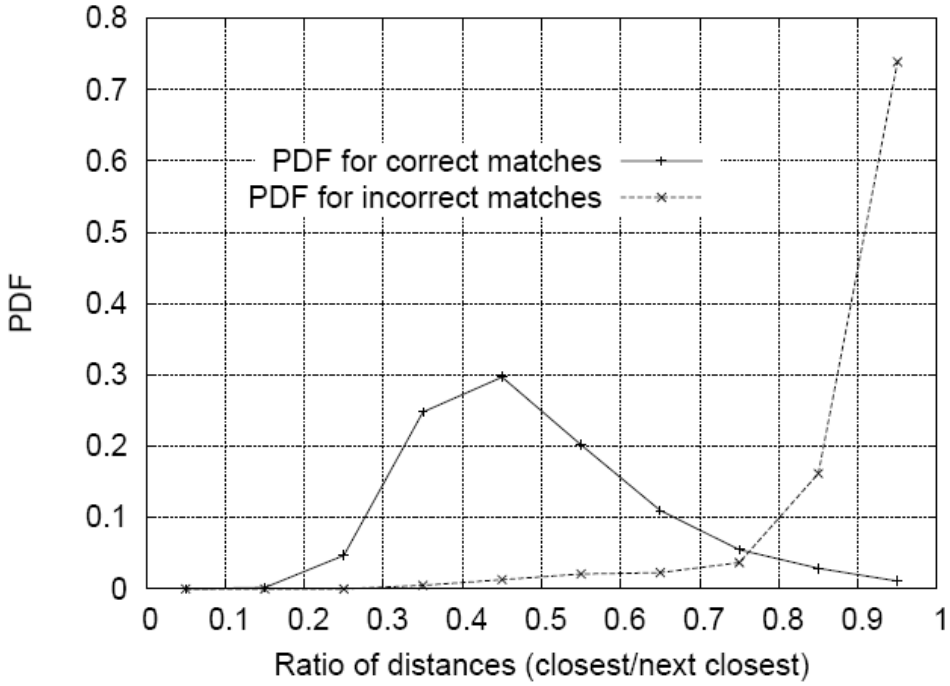
[Lowe, SIFT, 1999]





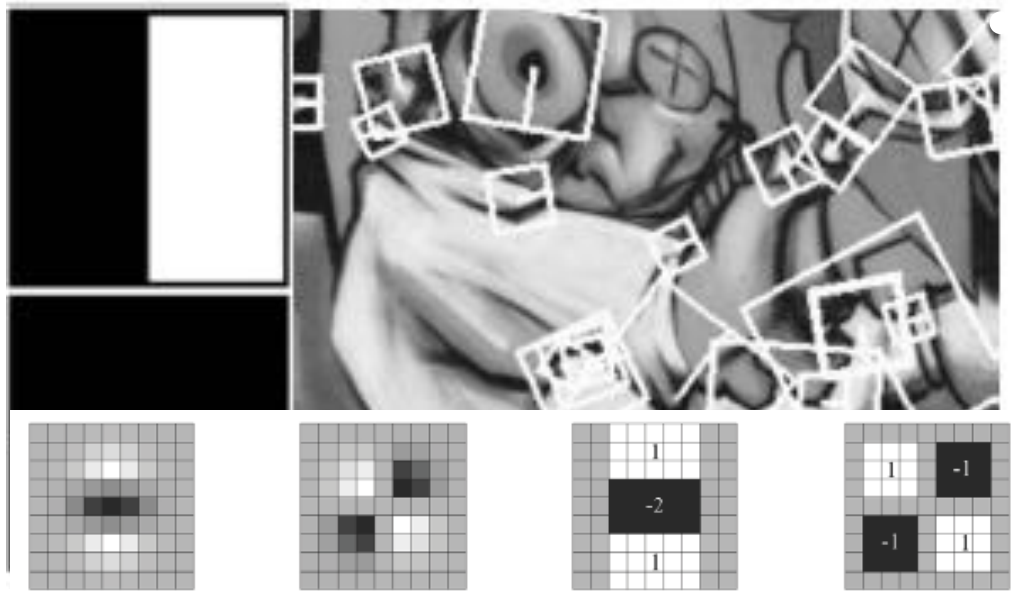
# Matching SIFT Descriptors

- Keypoints between two images are matched by identifying their nearest neighbors (Euclidean distance)
- But in some cases, the second closest-match may be very near to the first. It may happen due to noise or some other reasons. In that case, ratio of closest-distance to second-closest distance is taken. If it is greater than 0.8, they are rejected
- It eliminates around 90% of false matches while discards only 5% correct matches, as per the paper




Lowe IJCV 2004

# Local Descriptors: SURF




## Fast approximation of SIFT idea

**Efficient computation by 2D box filters & integral images**

**⇒ 6 times faster than SIFT**

**Equivalent quality for object identification**

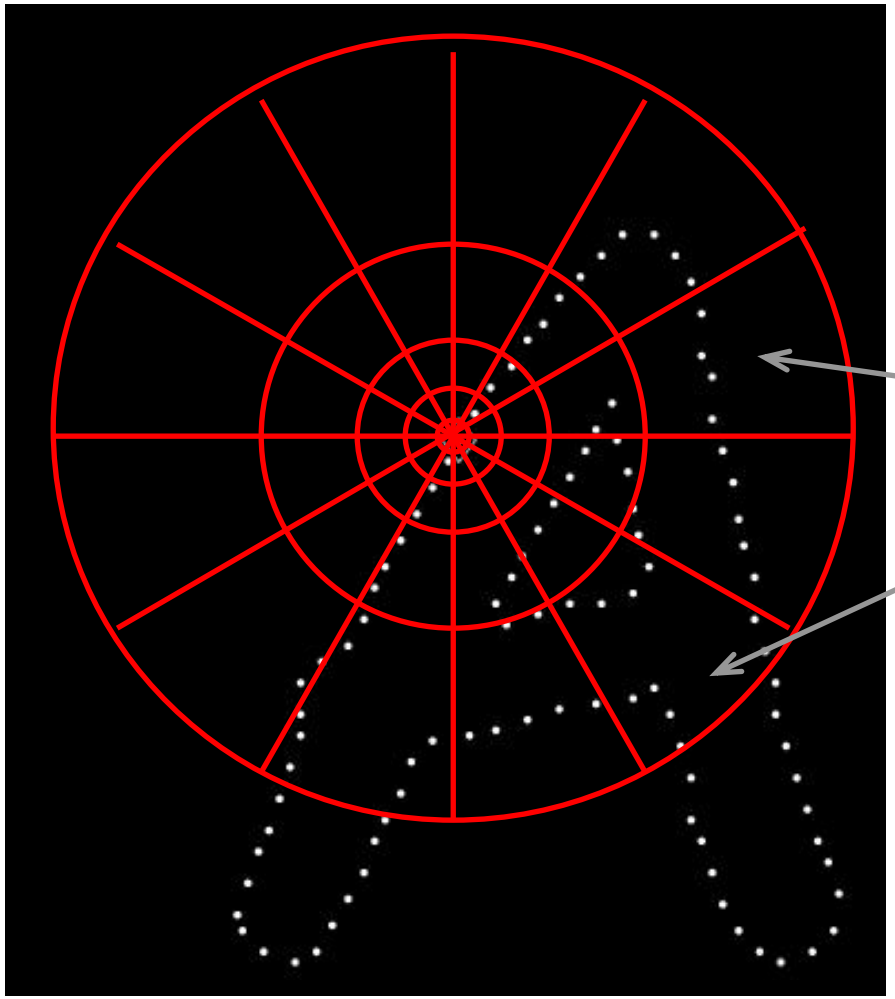
## GPU implementation available

**Feature extraction @ 200Hz (detector + descriptor, 640×480 img)**

http://www.vision.ee.ethz.ch/~surf

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

# Local Descriptors: Shape Context



**Count the number of points inside each bin, e.g.:**

**Count = 4**

⋮

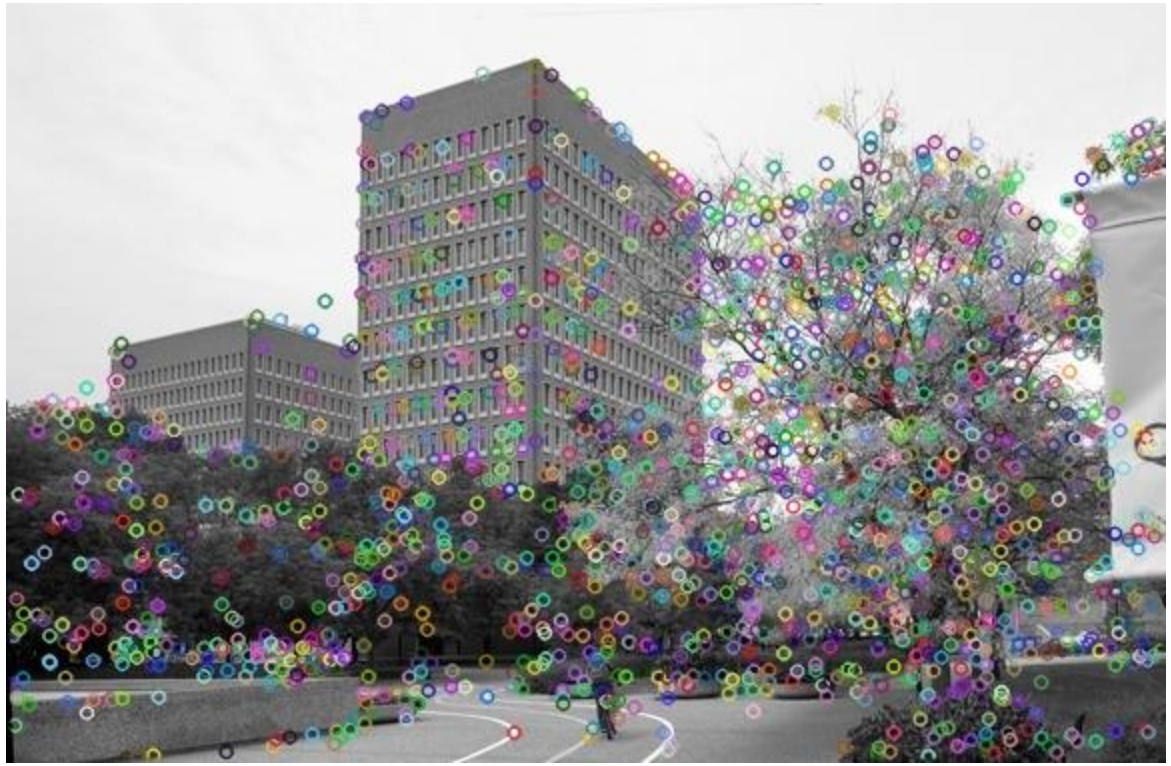
**Count = 10**

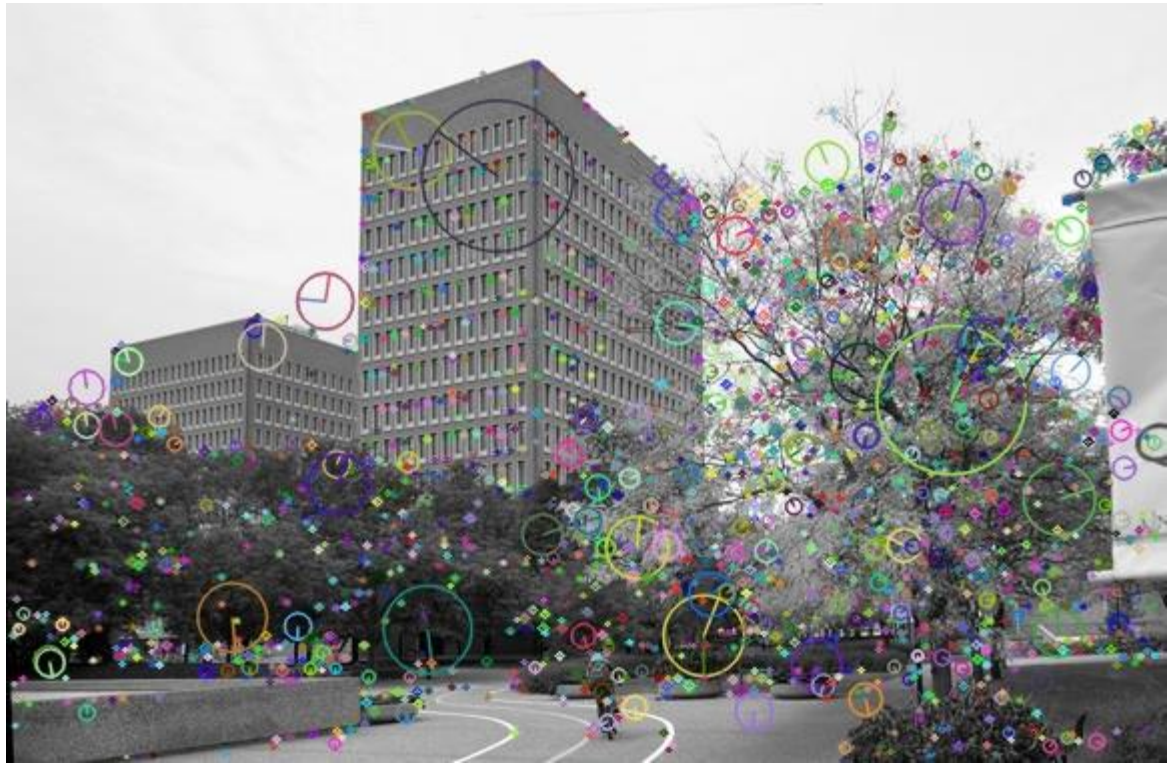
**Log-polar binning: more precision for nearby points, more flexibility for farther points.**

Belongie & Malik, ICCV 2001

# SIFT keypoints in OpenCV

```
sift = cv2.xfeatures2d.SIFT_create()
(kps, descs) = sift.detectAndCompute(gray, None)
```

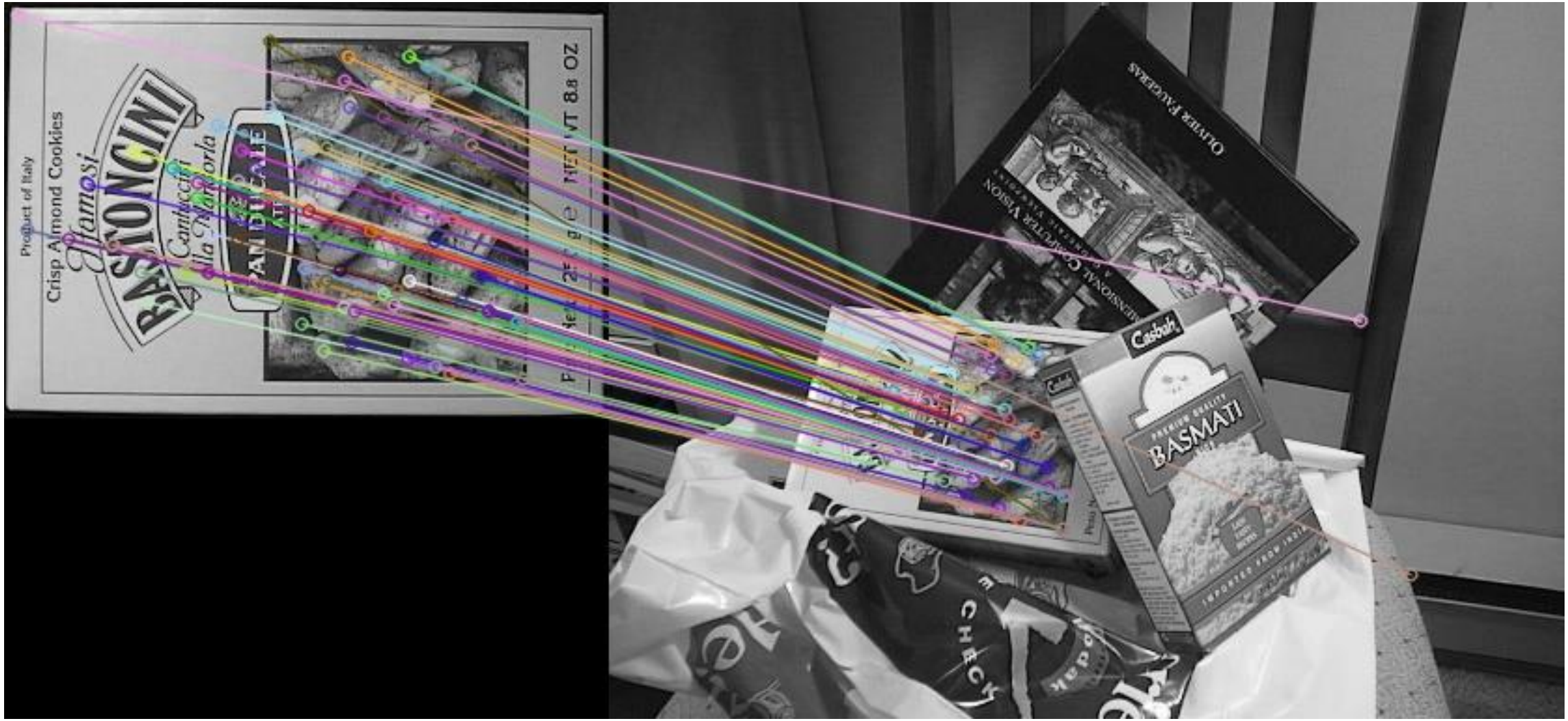




# SIFT keypoints in OpenCV

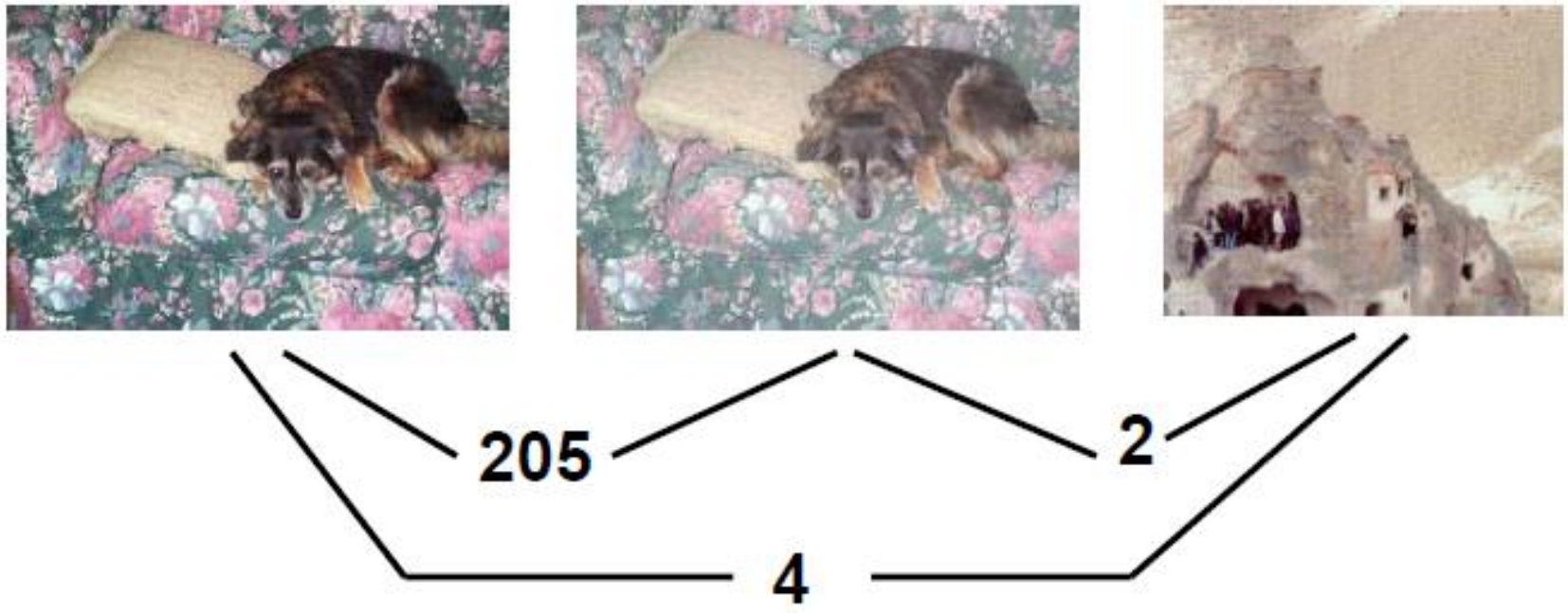
```
[  9.  61.  14.   0.   1.   1.   0.   0. 121. 123.   6.   1.   1.   8.
  53.  25.  27.  12.  15.  48.   6.   5. 126.  46.   2.   0.  12. 135.
  21.   2.  16.  20.  25.  16.   2.   0.   0.   0.   0.   4. 135.  99.
  12.   4.   4.   1.   0.  30.  55.  25.  39. 135. 115.   2.   0.   7.
   0.   0.  13. 135.  73.   5.   1.   4.   4.   2.   0.   0.   0.   0.
   0.   1.  64. 135. 113.   6.   1.   0.   0.   4.   9.  58. 135.  84.
  64.   2.   0.   1.   2.   8.  35.  44.  83.  23.   2.   5.   0.   0.
   0.   0.   0.   0.   0.   0.   0.   8.  48.   5.   0.   0.   0.   1.
  27.  31.  97.   8.   0.   0.   0.   6.  23.  20.  22.  13.  12.   7.
10. ]
...
```

# Image matching using SIFT



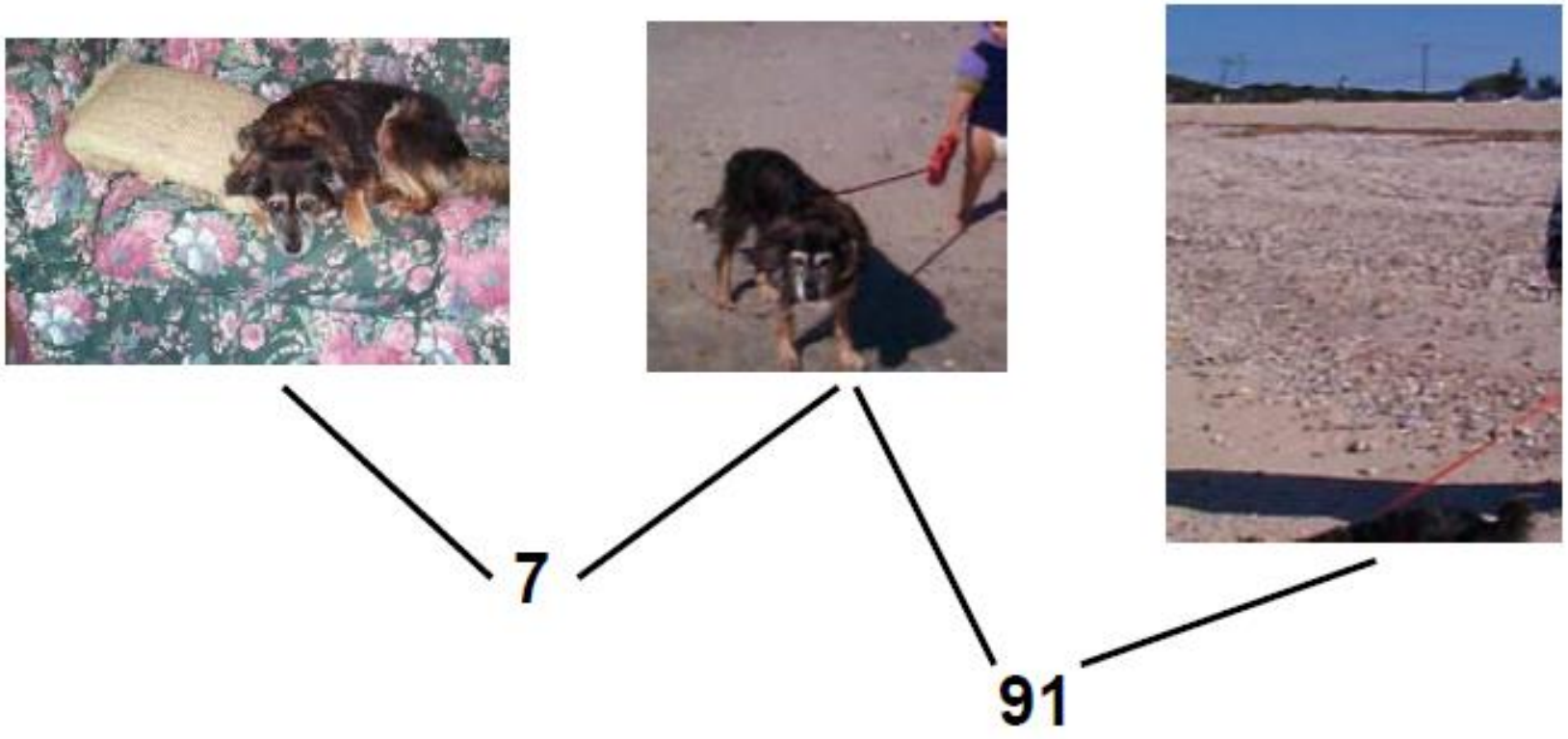
# SIFT does well in 2D matching

- Number of matched SIFT key-points
- Meeting a challenge



# SIFT fails in 3D object matching

- Number of matched SIFT key-points



# Problems: high dimensional data

- The dimensionality of the feature vectors is normally of the order $10^2$
- 1000+ key-points per image
- Non-Euclidean similarity measure
- Solution: convert this approach to a text retrieval representation