Feature Detection and Matching

Chul Min Yeum
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

CIVE 497 – CIVE 700: Smart Structure Technology

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Last updated: 2019-02-27

Two Major Challenge in Computer Vision

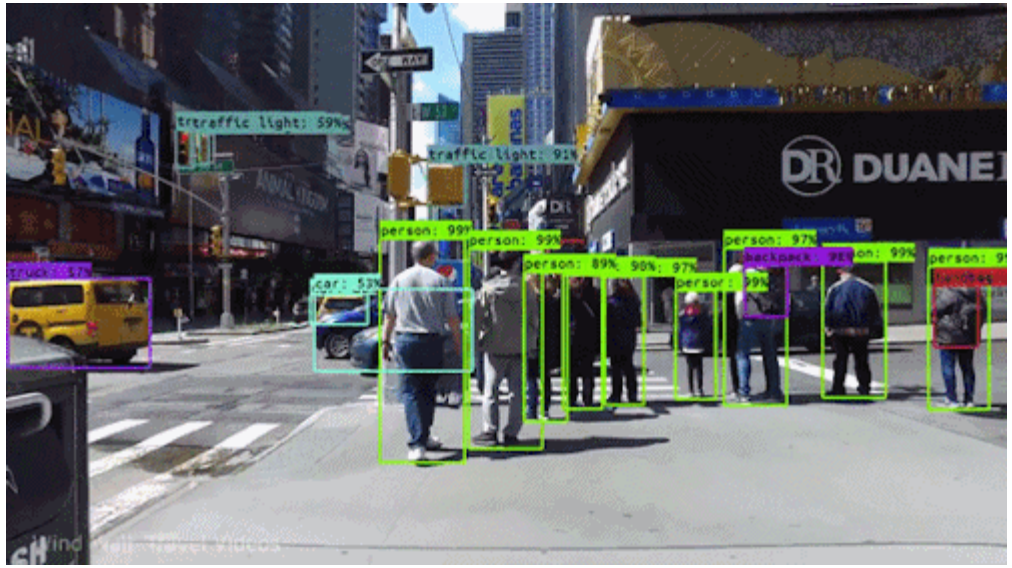
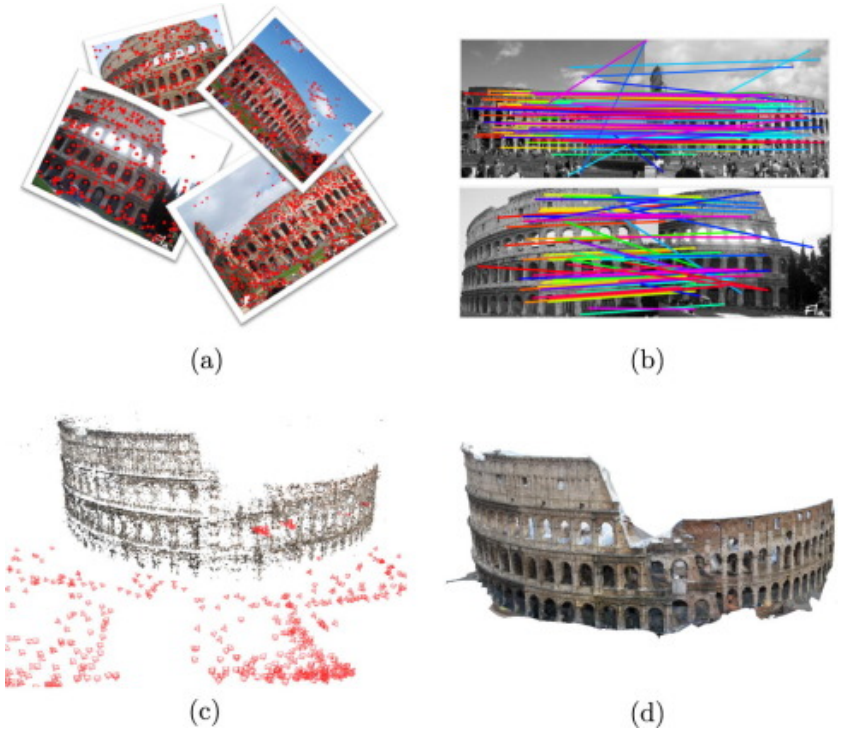


Image recognition



Feature matching

Why We Extract Features? Extracting Features

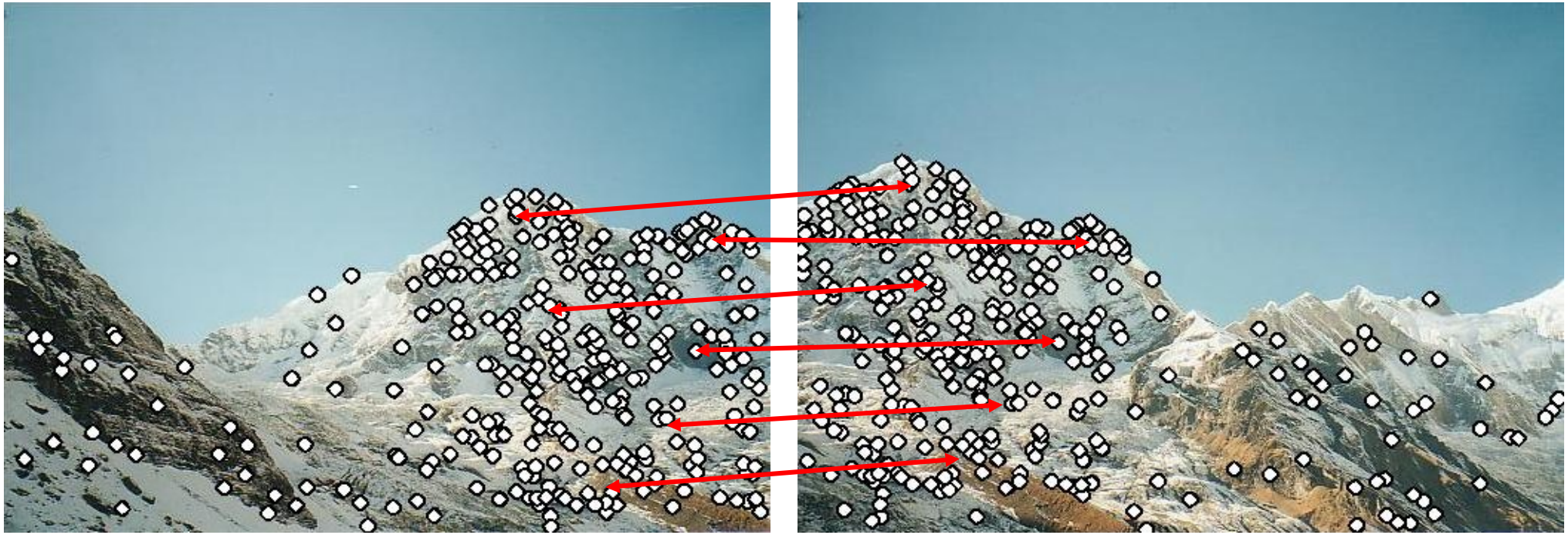
Motivation: panorama stitching

– We have two images – how do we combine them?

Why We Extract Features? Extracting Features (Continue)

Motivation: panorama stitching

– We have two images – how do we combine them?



Why We Extract Features? Extracting Features (Continue)

Motivation: panorama stitching

– We have two images – how do we combine them?

Example: Automatic Panoramas

A large volume collected images from drone

Orthophoto(10,000 × 3,656) geometrically connected to each collected images

Jongseong Choi, Chul Min Yeum, Shirley J. Dyke, and Mohammad R. Jahanshahi, “Computer-Aided Approach for Rapid Post-Event Visual Evaluation of a Building Façade,” Sensors, 18, 3017 (2018).

Example: Multiview Geometry

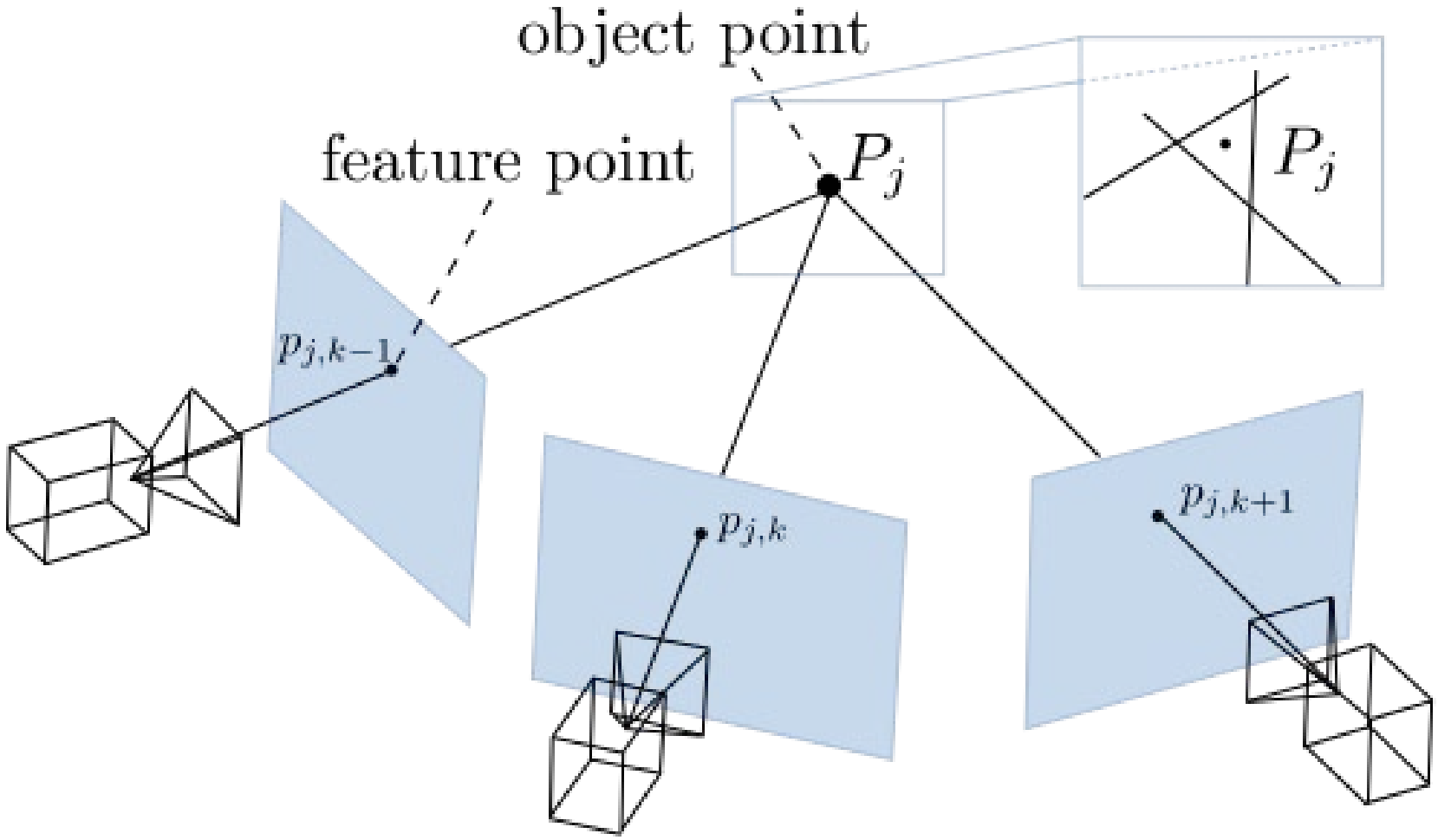
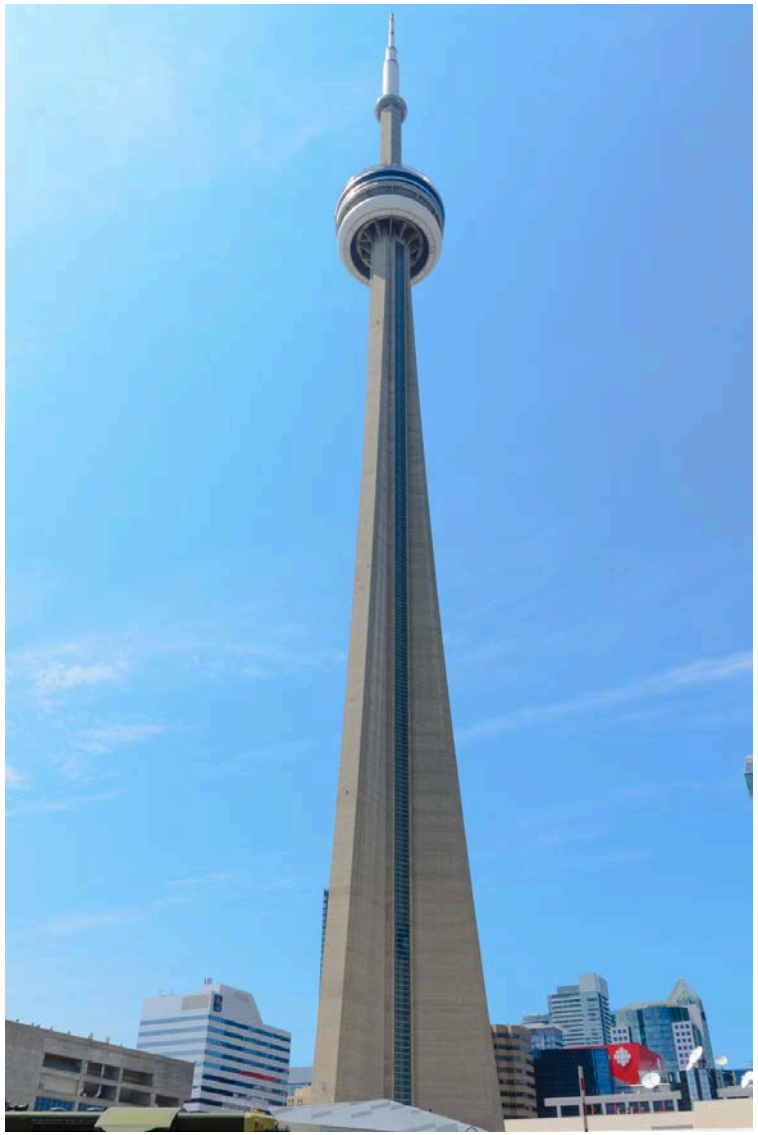


Image Matching



What are Features (Keypoints)?

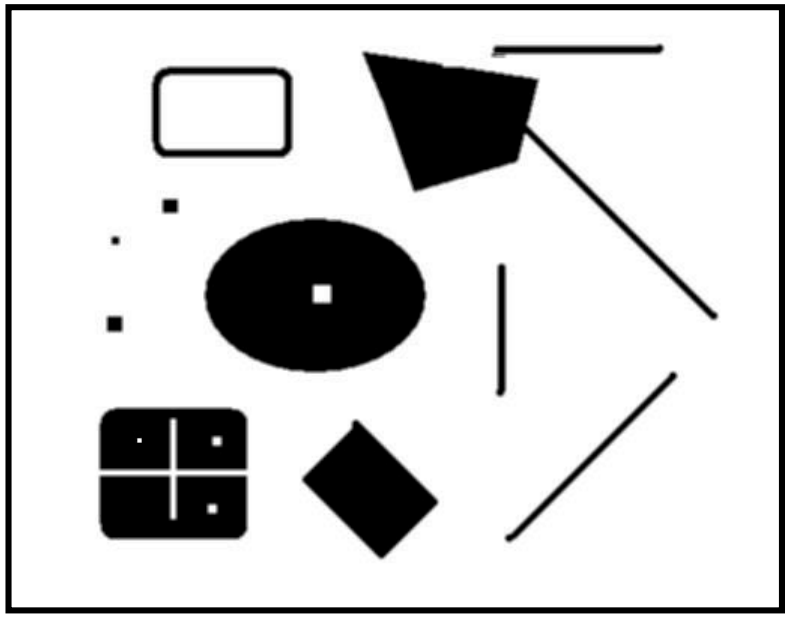
- A feature is an interesting point that is selected based on a certain criteria, such as edge, corner, or blob.
- This is represented in terms of the coordinates of the image points by pixel or sub-pixel.
- The feature likely contain and preserve the distinctive local regional information.
- Note: “interest points” = “keypoints”, also sometimes called “features”

Many applications:

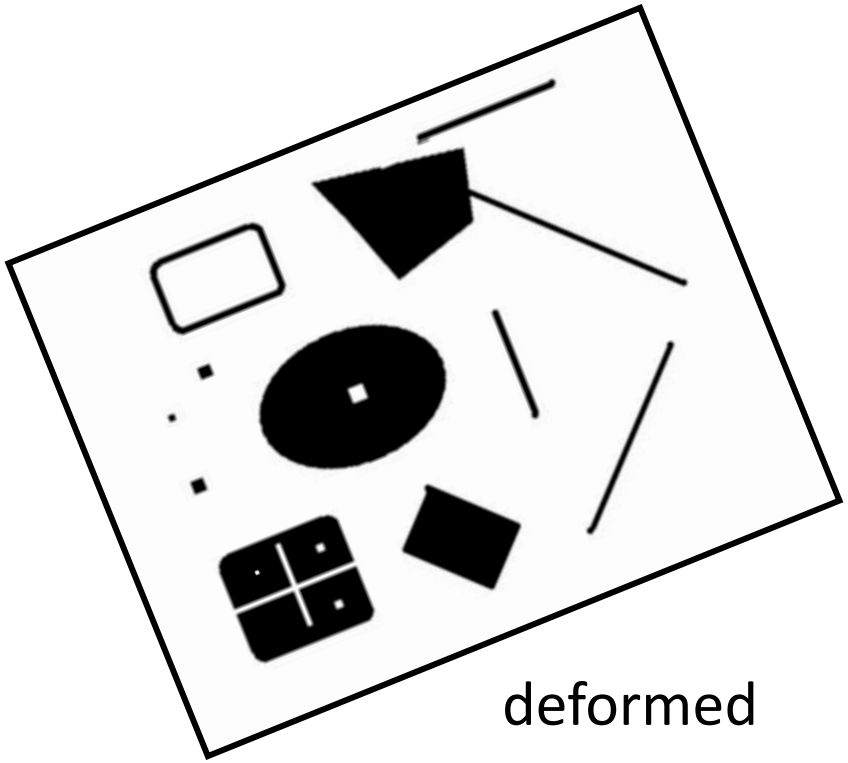
- Object/motion tracking: which points are good to track?
- Object recognition: find patches likely to tell us something about object category
- 3D scene reconstruction: find correspondences across different views

Example: Keypoints/Features

Suppose you have to click on some point, go away and come back after I deform the image, and click on the same points again. Which points would you choose?

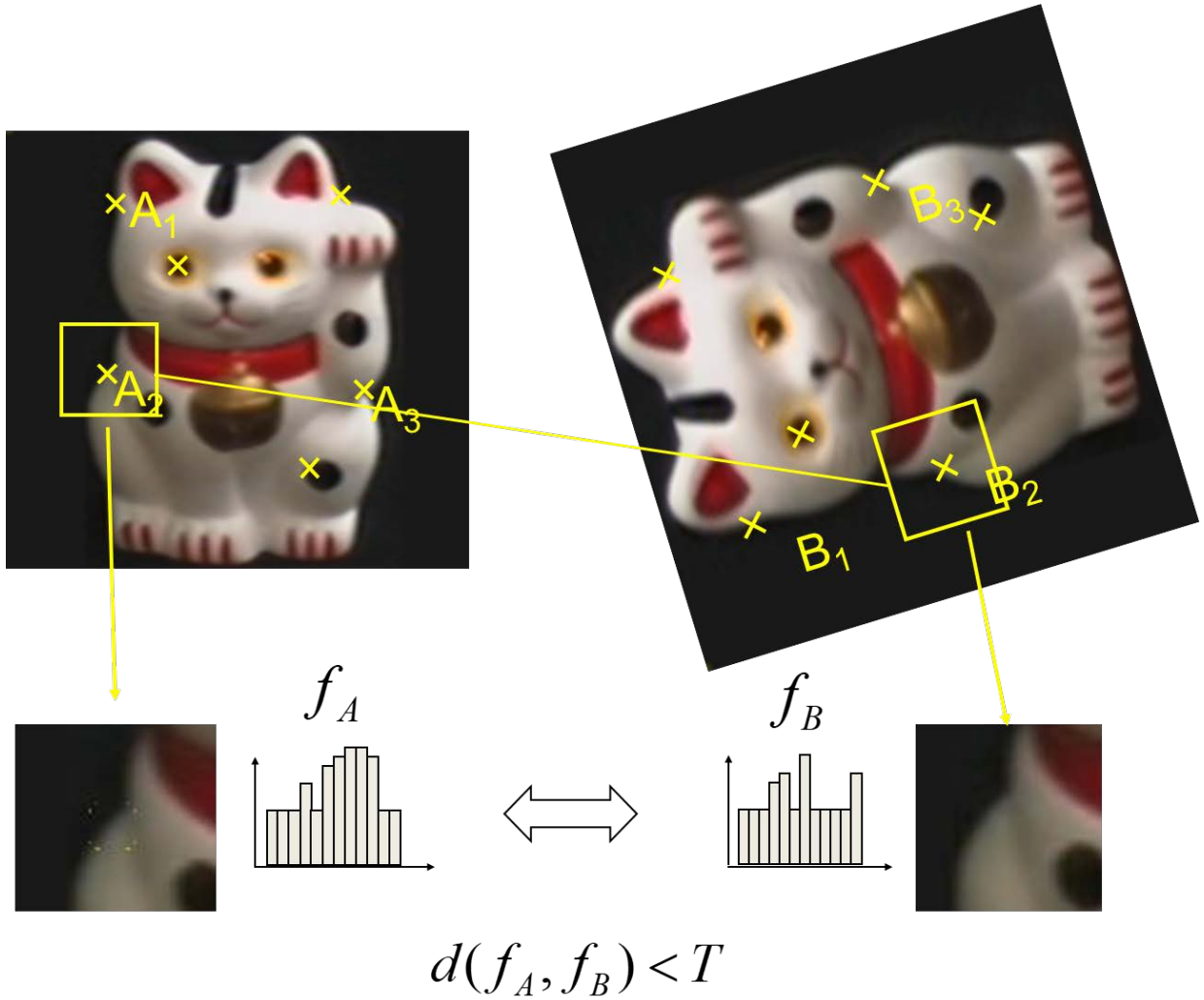


original



deformed

Overview of Feature Matching



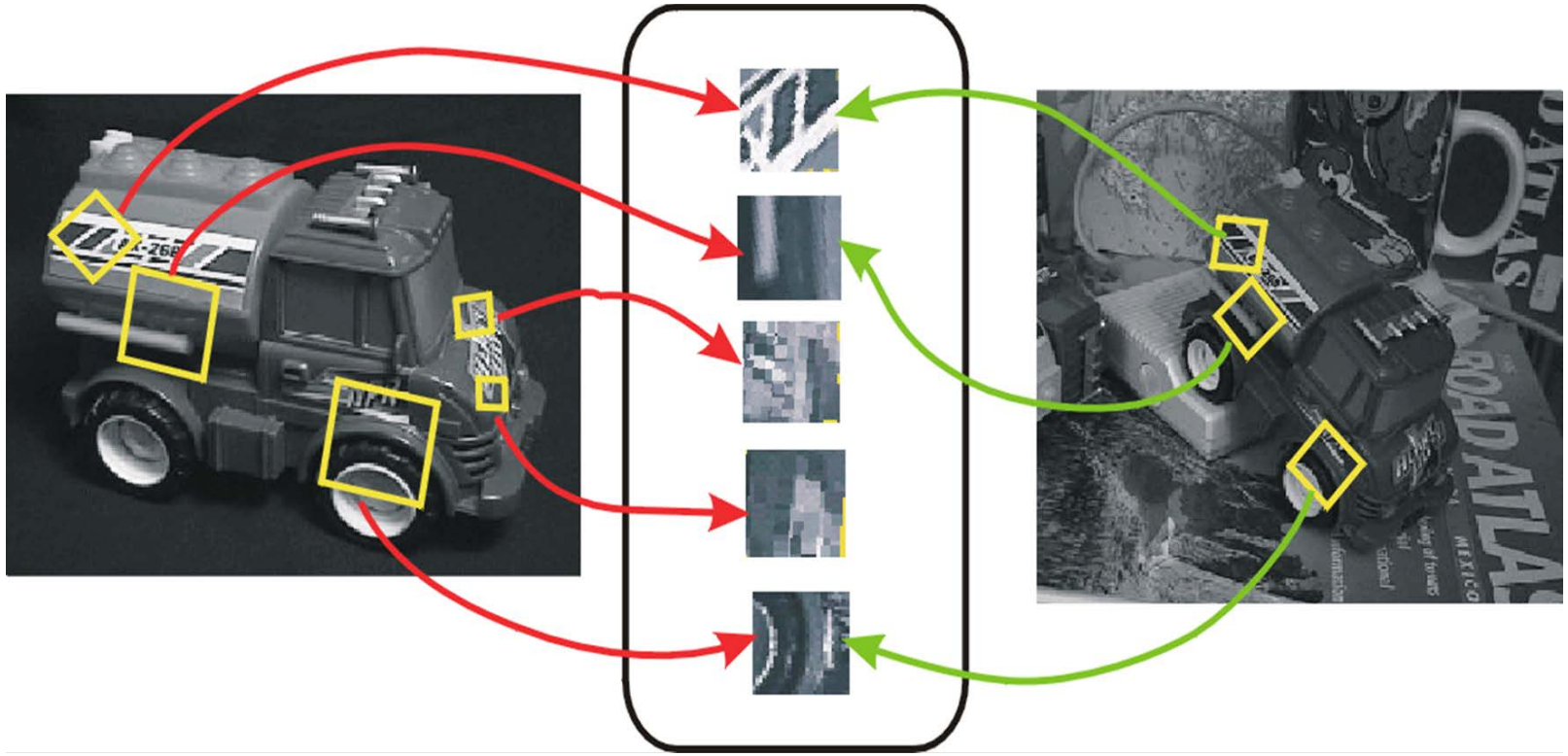
1. Find a set of distinctive key-points
2. Define a region around each keypoint
3. Extract and normalize the region content
4. Compute a local descriptor from the normalized region
5. Match local descriptors

Goal for Features

Detect points that are *repeatable* and *distinctive*

Invariant Local Features

Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



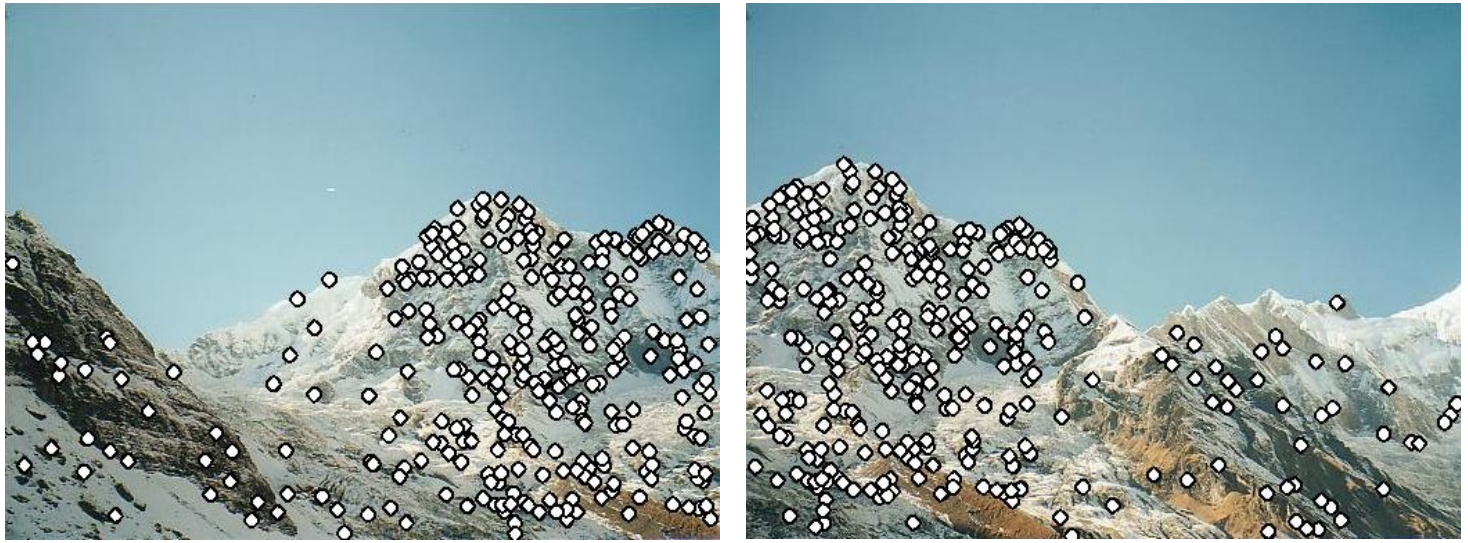
Characteristic of Good Features

Repeatability: The same feature can be found in several images despite geometric and photometric transformations

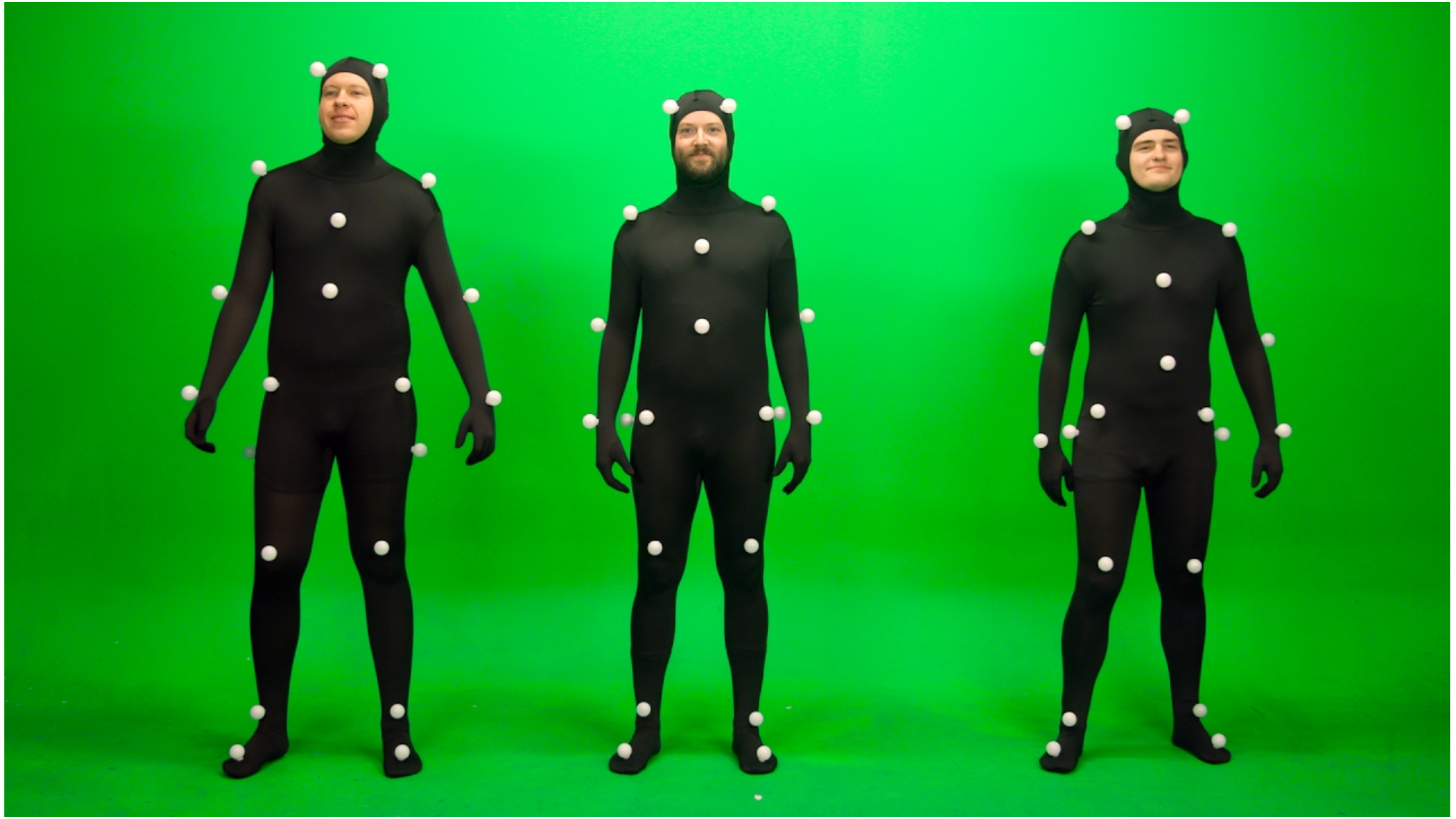
Saliency: Each feature is distinctive

Compactness and efficiency: Many fewer features than image pixels

Locality: A feature occupies a relatively small area of the image; robust to clutter and occlusion



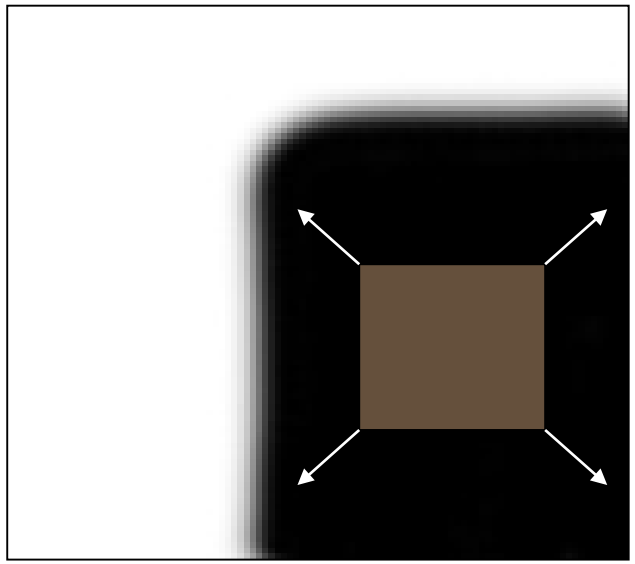
Example: Motion Capture



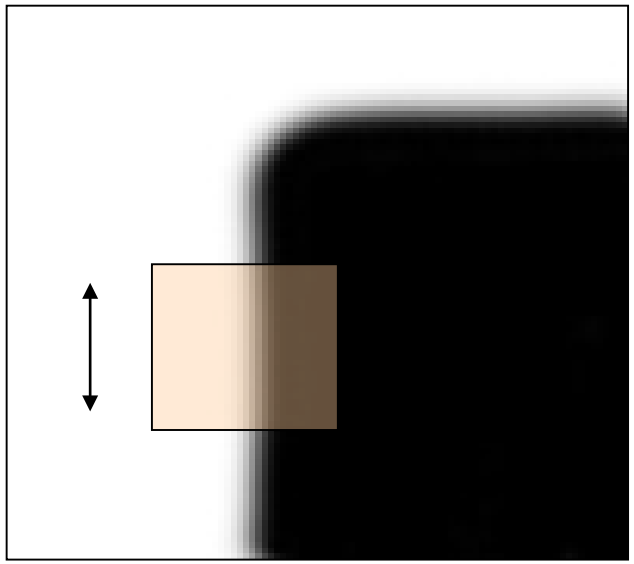
What Points would You Choose?

Corner Detection: Basic Idea

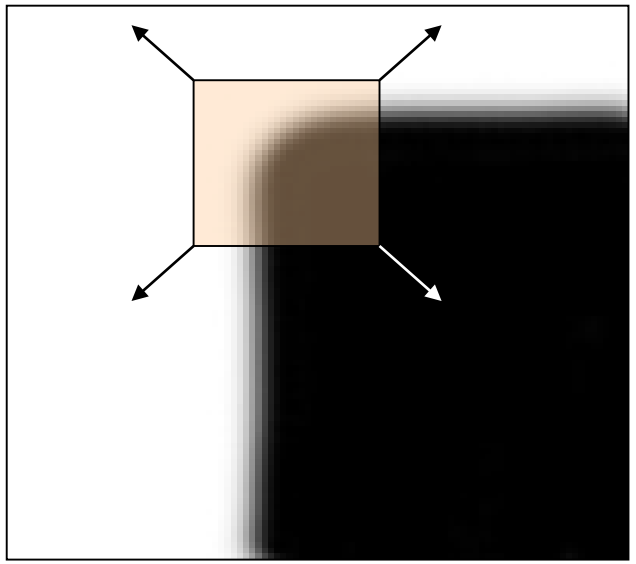
- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



Flat region
no change in all directions



Edge
no change along the edge direction

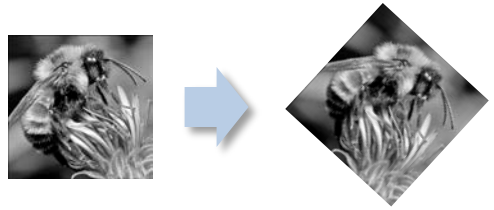


Corner
significant change in all directions

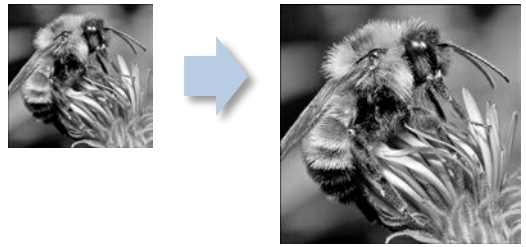
Image Transformations

- Geometric

Rotation

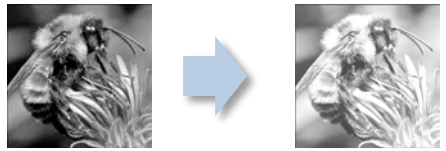


Scale



- Photometric

Intensity change



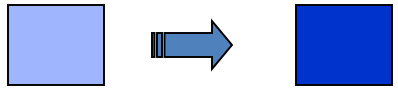
Invariance and Covariance

We want corner locations to be *invariant* to photometric transformations and *covariant* to geometric transformations

Invariance: image is transformed and corner locations do not change

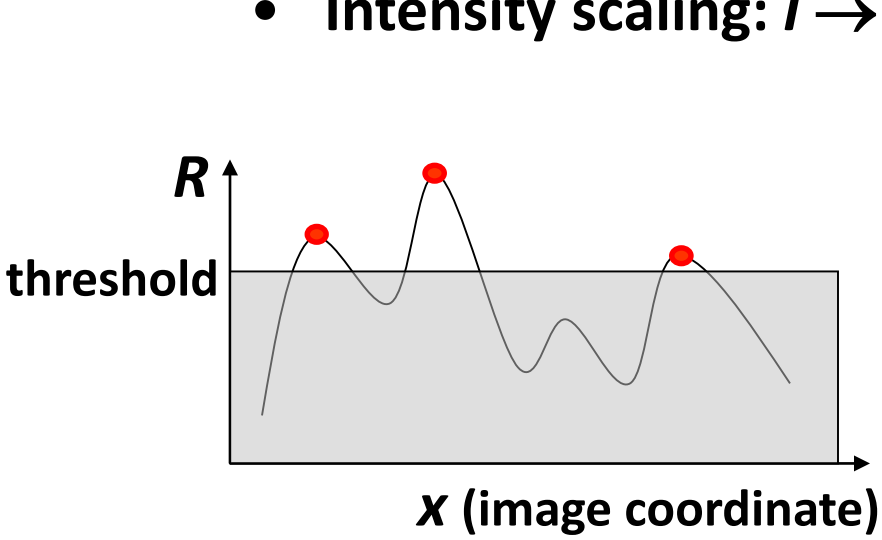
Covariance: if we have two transformed versions of the same image, features should be detected in corresponding locations

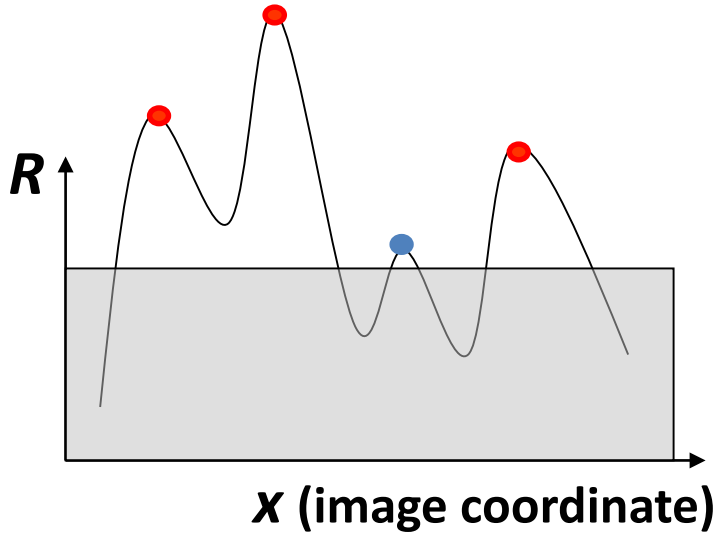
Intensity Change



$I \rightarrow a\,I + b$

- Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$
- Intensity scaling: $I \rightarrow a\,I$

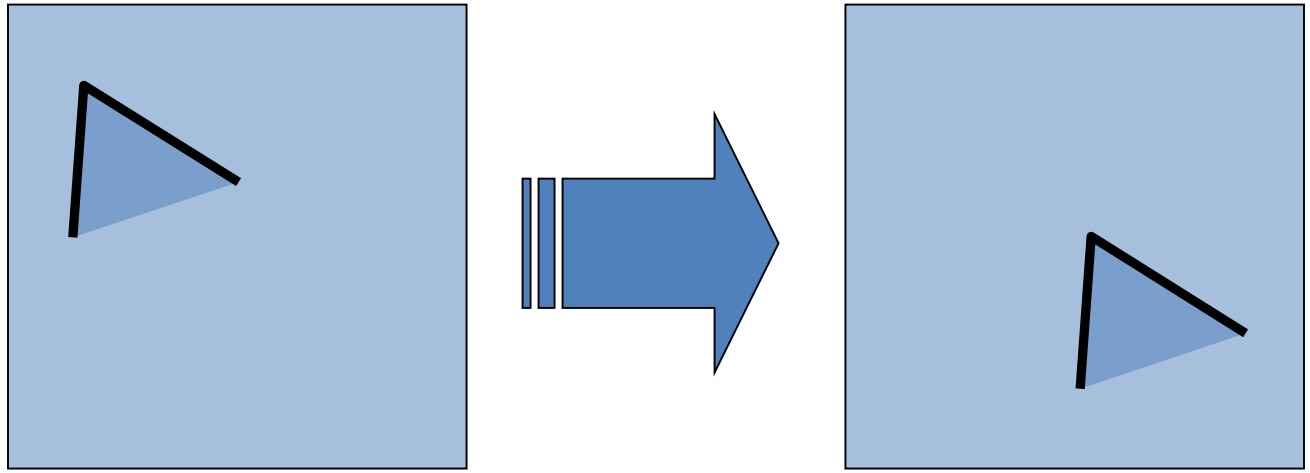




Partially invariant to affine intensity change

Source: James Hayes

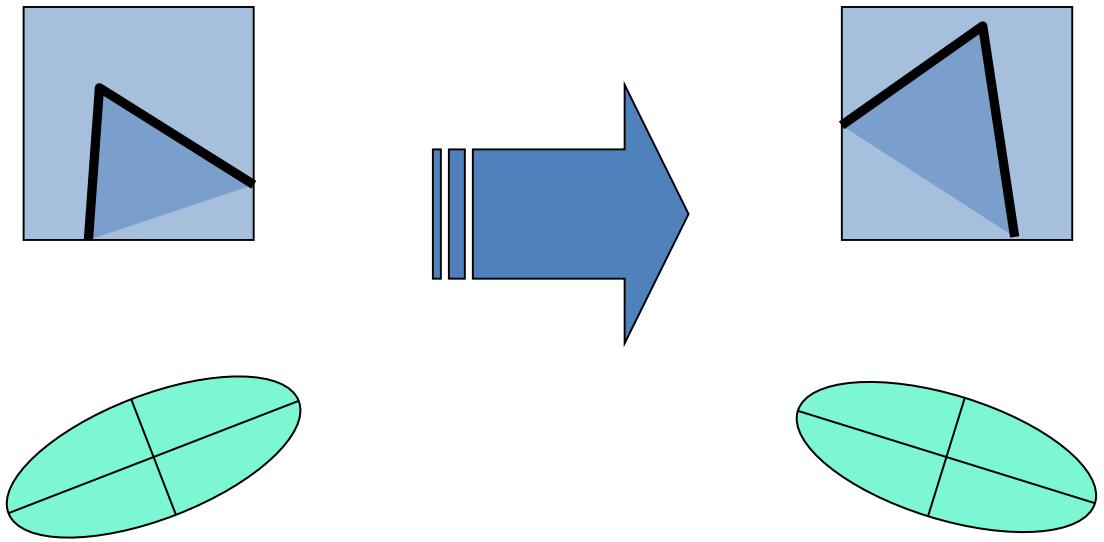
Image Translation



- **Derivatives and window function are shift-invariant**

Corner location is invariant to image translation

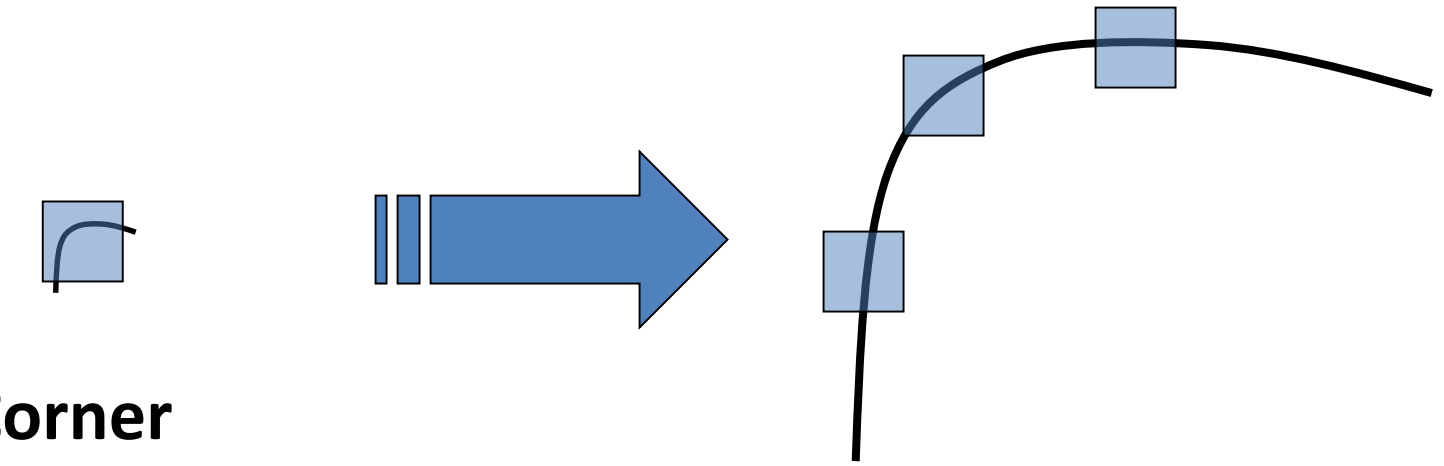
Image Rotation



Second moment ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner location is invariant to image rotation

Scaling



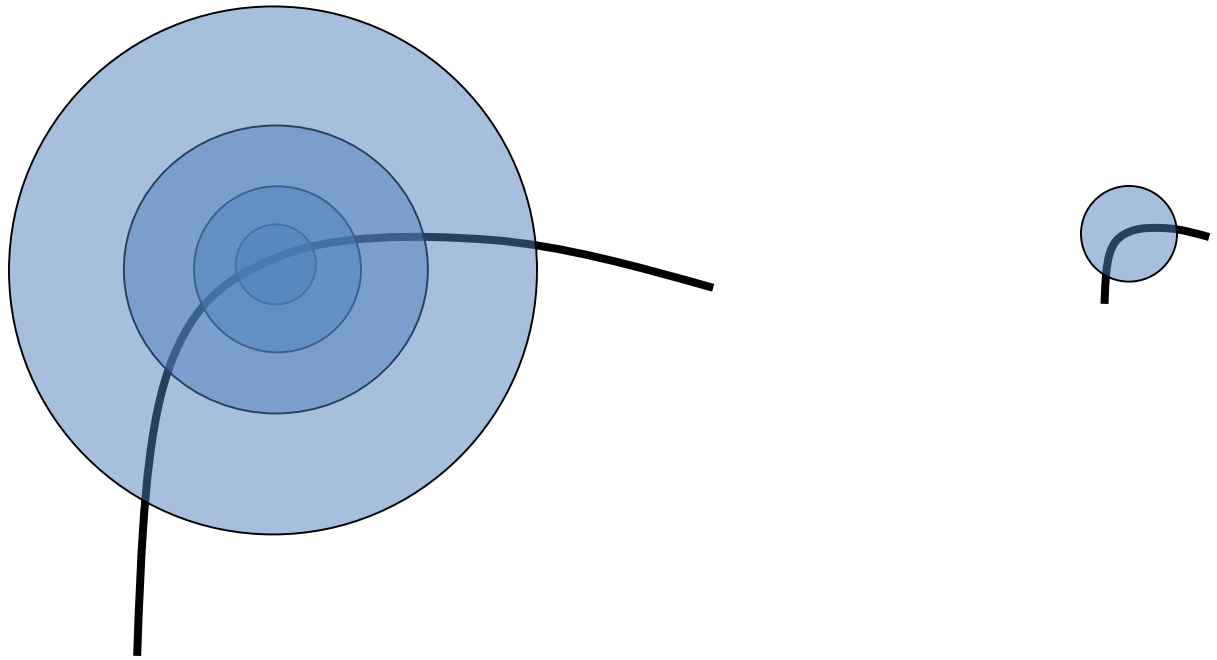
Corner

All points will be classified as edges

Corner location is not invariant to image scale!

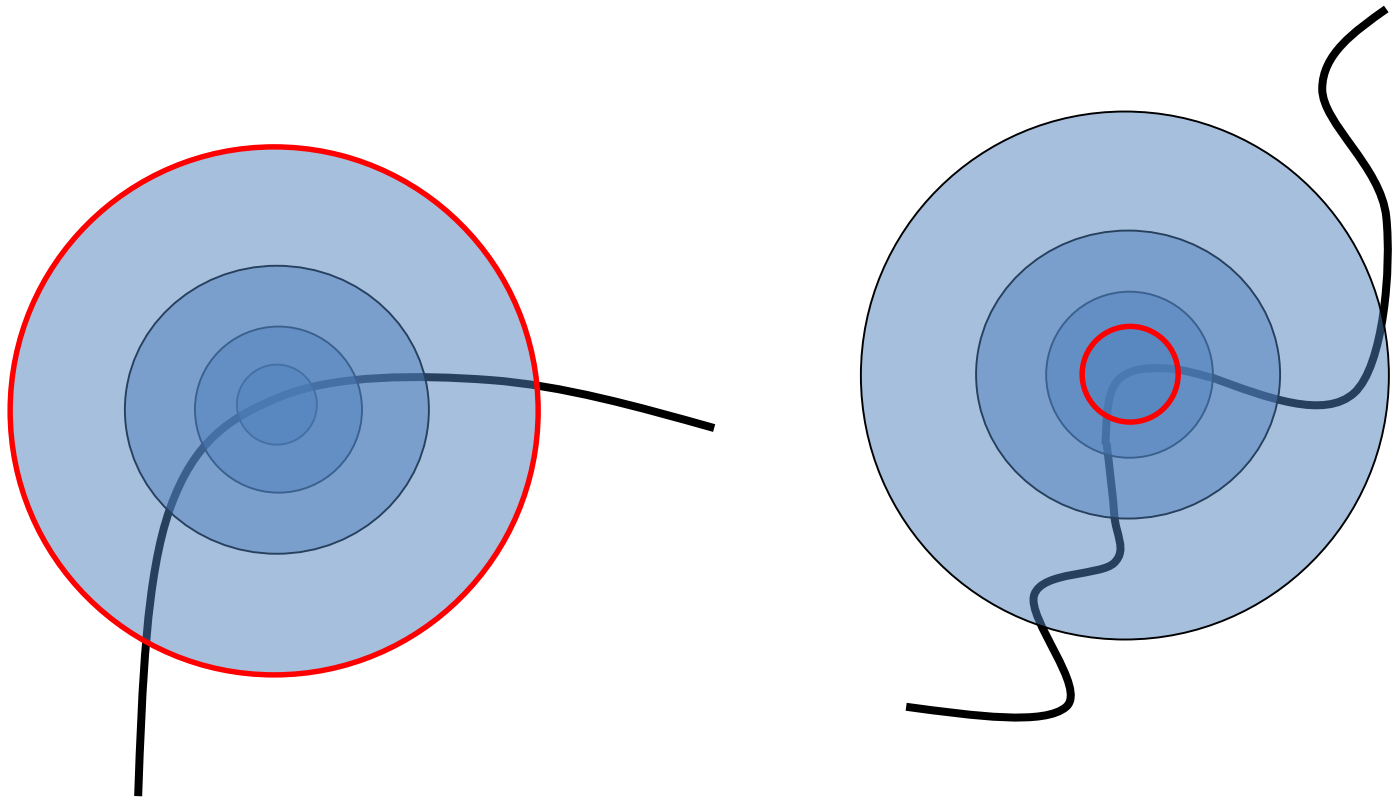
Scale Invariant Detection

- Consider regions (e.g. circles) of different sizes around a point
- Regions of corresponding sizes will look the same in both images



Scale Invariant Detection (Continue)

- The problem: how do we choose corresponding circles ***independently*** in each image?
- Choose the scale of the “best” corner



Example: Scale Invariance

SIFT: Scale-Invariant Feature Transform

Distinctive Image Features from Scale-Invariant Keypoints

David G. Lowe
Computer Science Department
University of British Columbia
Vancouver, B.C., Canada
lowe@cs.ubc.ca

January 5, 2004

Abstract

This paper presents a method for extracting distinctive invariant features from images that can be used to perform reliable matching between different views of an object or scene. The features are invariant to image scale and rotation, and are shown to provide robust matching across a a substantial range of affine distortion, change in 3D viewpoint, addition of noise, and change in illumination. The features are highly distinctive, in the sense that a single feature can be correctly matched with high probability against a large database of features from many images. This paper also describes an approach to using these features for object recognition. The recognition proceeds by matching individual features to a database of features from known objects using a fast nearest-neighbor algorithm, followed by a Hough transform to identify clusters belonging to a single object, and finally performing verification through least-squares solution for consistent pose parameters. This approach to recognition can robustly identify objects among clutter and occlusion while achieving near real-time performance.

David G. Lowe

Canadian computer scientist

David G. Lowe is a Canadian computer scientist working for Google as a Senior Research Scientist. He was a former professor in the Computer Science Department at the University of British Columbia and New York University. Wikipedia

Known for: Scale-invariant feature transform

Residence: Seattle, Washington, United States

Alma maters: The University of British Columbia, Stanford University (1985, PhD)

Academic advisor: Thomas Binford

Notable student: Ken Perlin

Steps for Extracting SIFT Keypoints

Scale-space extrema detection: Difference-of-Gaussian function is used to identify potential interest points that are invariant to scale and orientation.

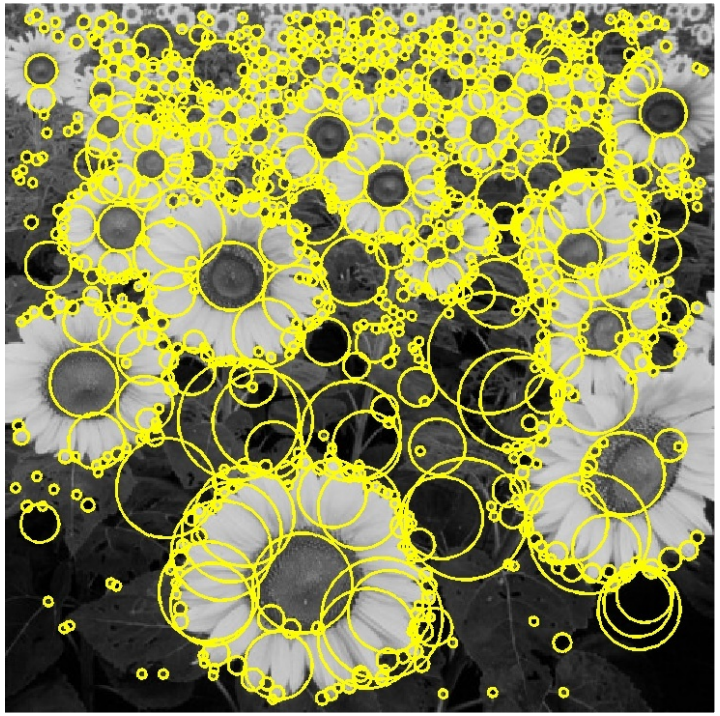
Keypoint localization: At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability.

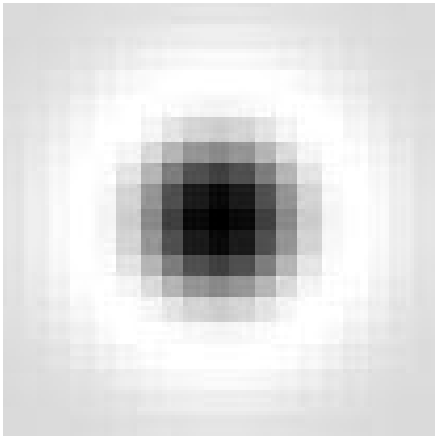
Orientation assignment: One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.

Keypoint descriptor: The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

Basic Idea

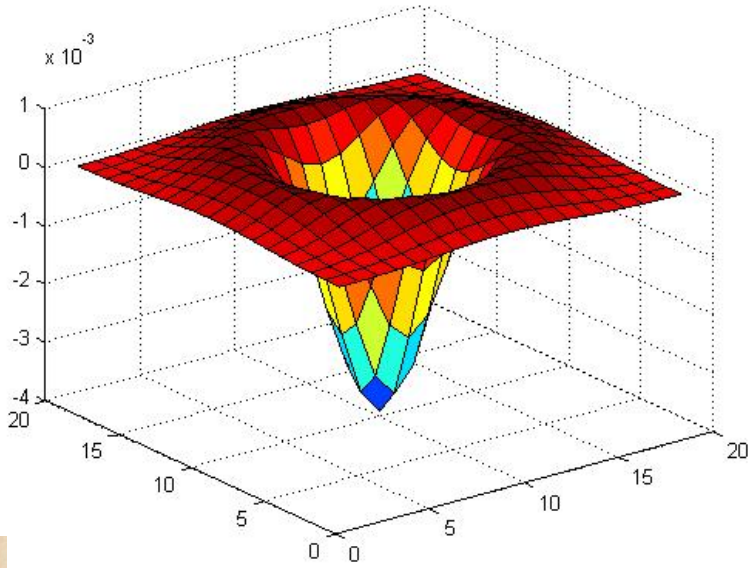
- Convolve the image with a “blob filter” at multiple scales and look for extrema of filter response in the resulting *scale space*

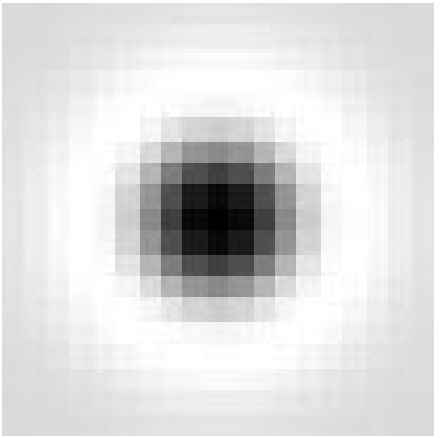




Blob Filter

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

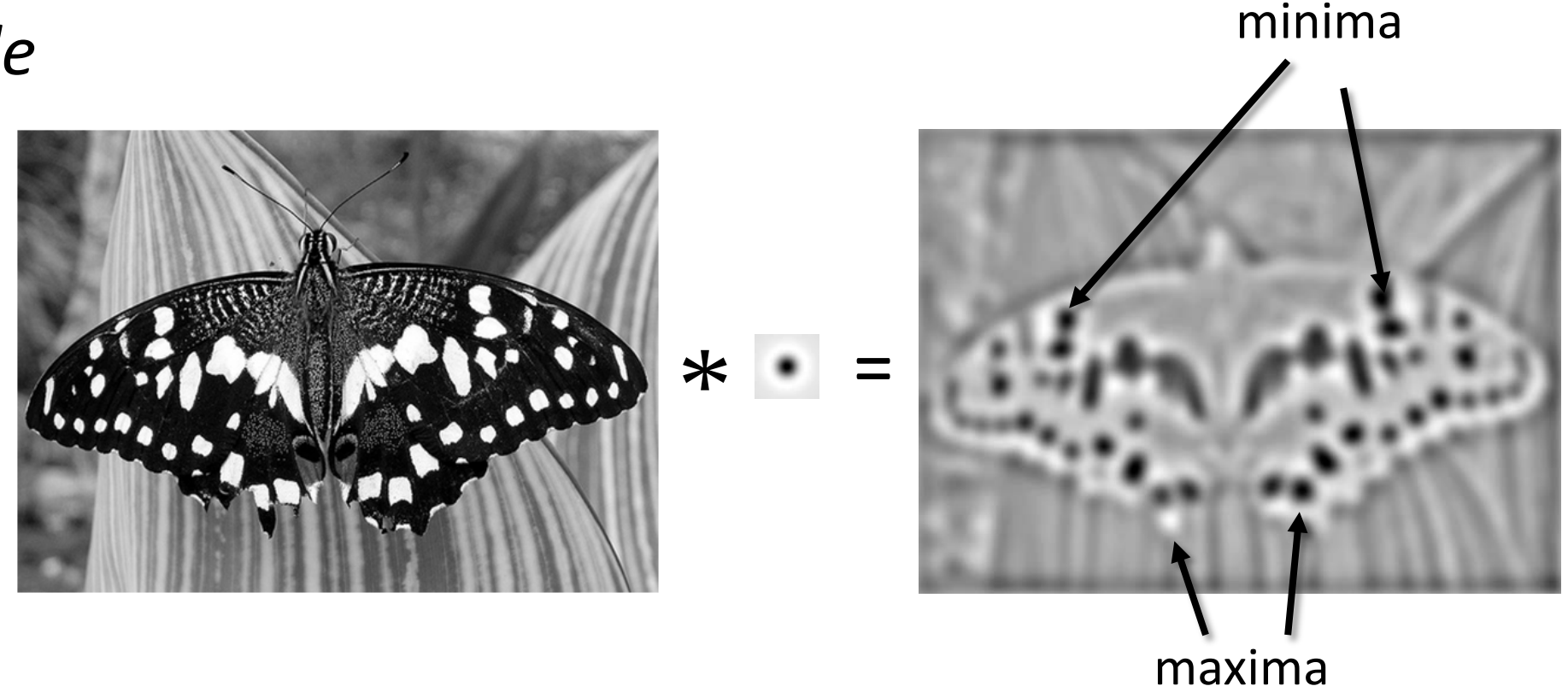




$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

Blob Detection

- Find maxima *and minima* of blob filter response in space *and scale*



Separable Filter

Assume you have a $N \times M$ sized image.

If you know take what is classically used, a square filter kernel, of let's say size $L \times L$, you'd need to convolve that with the picture – which gives you $N \times M$ pixels, each needing L^2 multiply-accumulates. So you end up with $A_{2D} = L^2 MN$ operations.

Now, if you can decompose that filter into an L-sized horizontal and an L-sized vertical 1D-filter, you could first do all rows – that's M values per row, each needing L operations, so LMN for all rows – and then you'd do the same with the vertical filter, so LNM for all columns – and you end up with $A_{1D} = 2LMN$, and you'd only need to show that

$$
\begin{aligned}
& A_{1D} < A_{2D} \\
\iff & \\
& 2LMN < L^2 MN \qquad || : LMN, \text{ legal since } L, M, N > 0 \\
\iff & \\
& 2 < L
\end{aligned}
$$

most filters are larger than 2.

$$\mathbf{G_x} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} +1 & 0 & -1 \end{bmatrix} * A$$

https://dsp.stackexchange.com/questions/36962/why-does-the-separable-filter-reduce-the-cost-of-computing-the-operator

Efficient Implementation

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

(Laplacian)

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

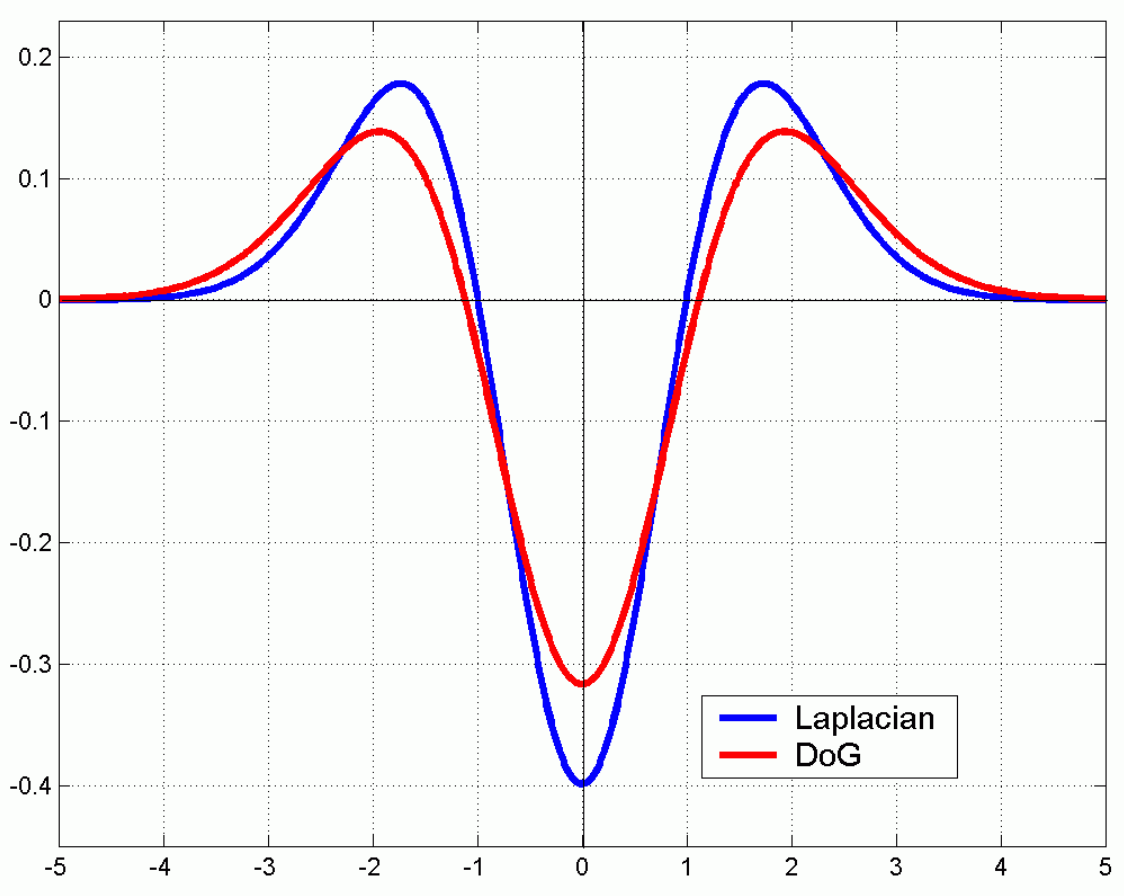
(Difference of Gaussians)

$$\frac{\partial G}{\partial \sigma} = \sigma \Delta^2 G$$ Heat Equation

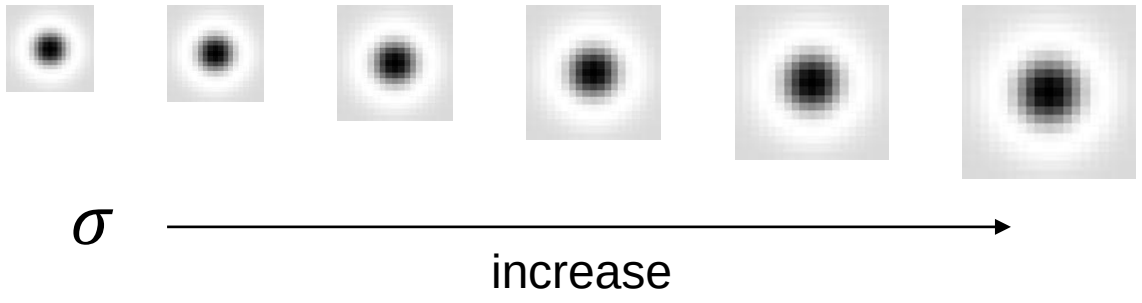
$$\sigma \Delta^2 G = \frac{\partial G}{\partial \sigma} = \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

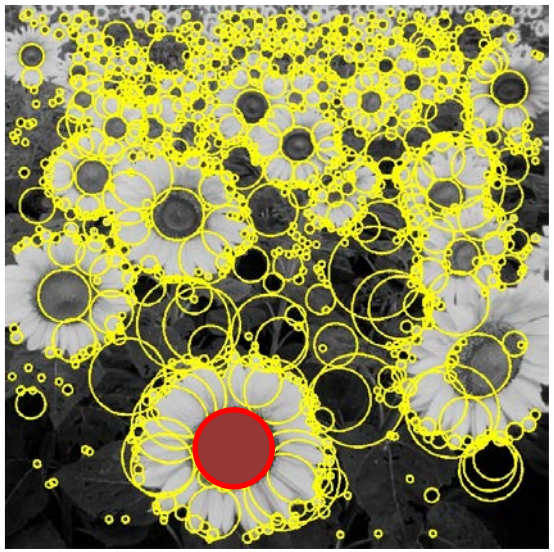
$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\sigma^2 \Delta^2 G$$

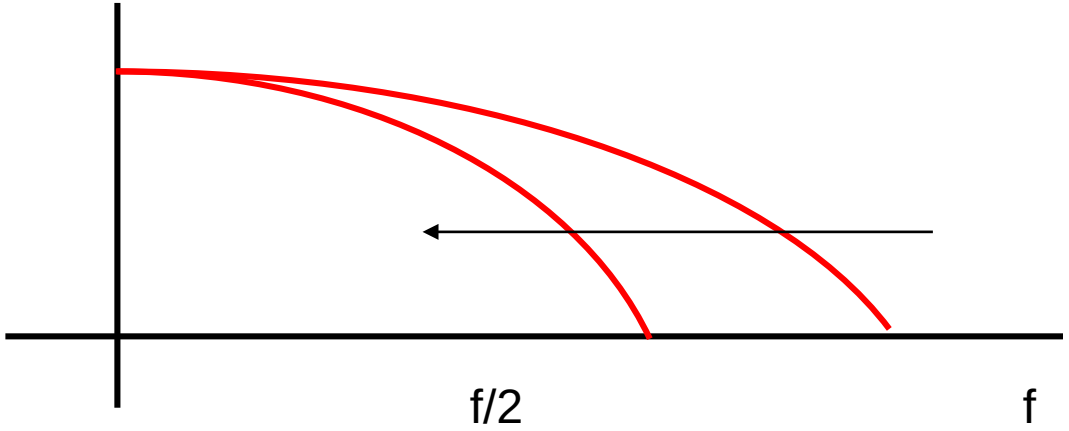
$$\text{Typical values}: \sigma = 1.6;\ k = \sqrt{2}$$

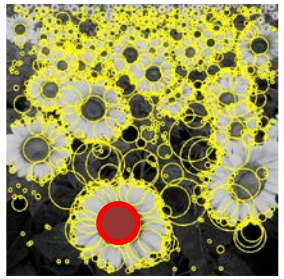


Octave





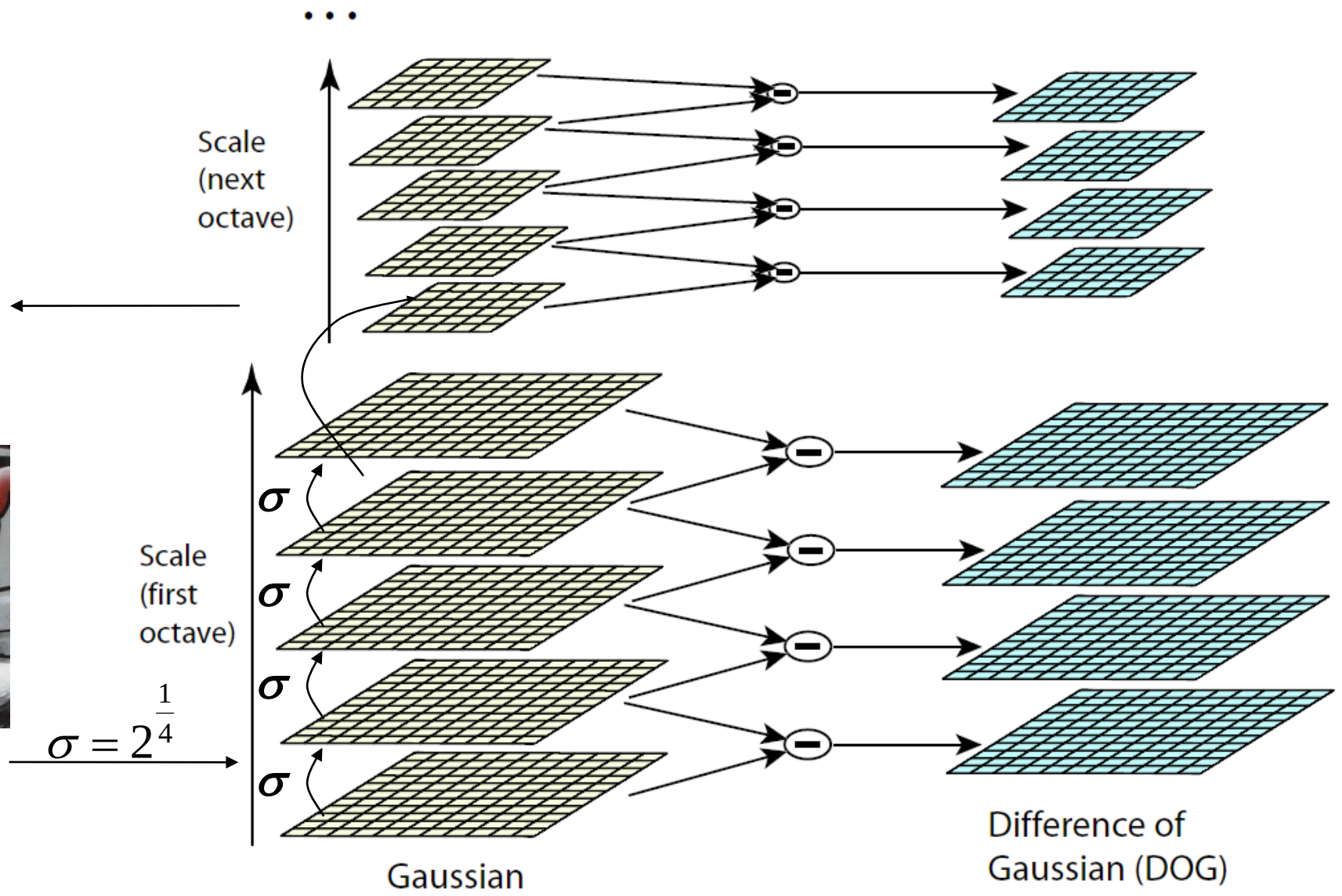




Gaussian Scale Pyramid

Sampling with step $\sigma^4 = 2$

Original image



Example: Sunflower

Gaussian Smoothing

sigma = sigma0 * 1.270

sigma = sigma0 * 1.600

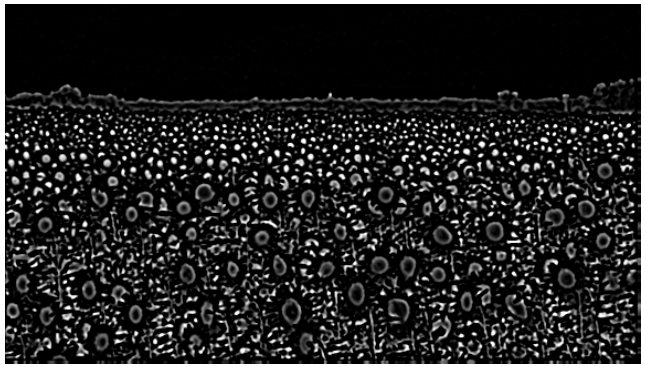
sigma = sigma0 * 2.016

sigma = sigma0 * 2.540

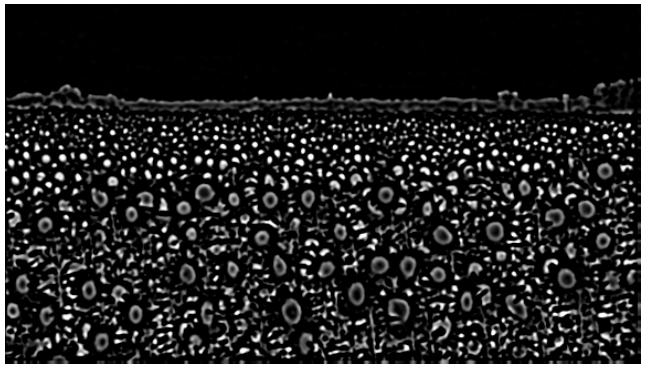
sigma = sigma0 * 3.200

Difference of Gaussian

Sigma0*1.270

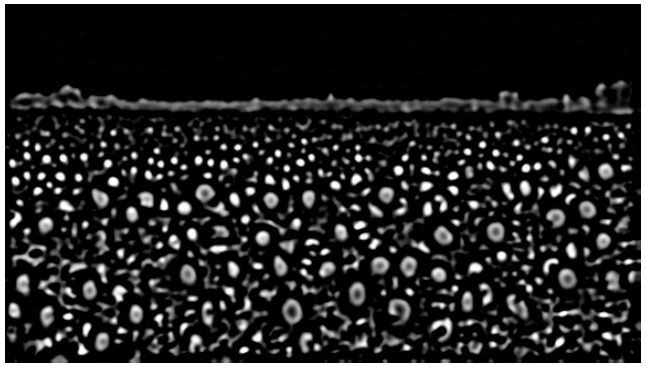


Sigma0*1.6



Sigma0*2.0

Sigma0*2.5



Detection of scale-space extrema

The scale space of an image is defined as a function, $L(x,y,\sigma)$, that is produced from the convolution of a variable-scale Gaussian, $G(x,y,\sigma)$, with an input image, $I(x,y)$:

where * is the convolution operation in *x* and *y*, and

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

Difference-of-Gaussian function:

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

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

Orientation Assignment

By assigning a consistent orientation to each keypoint based on local image properties, the keypoint descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation.

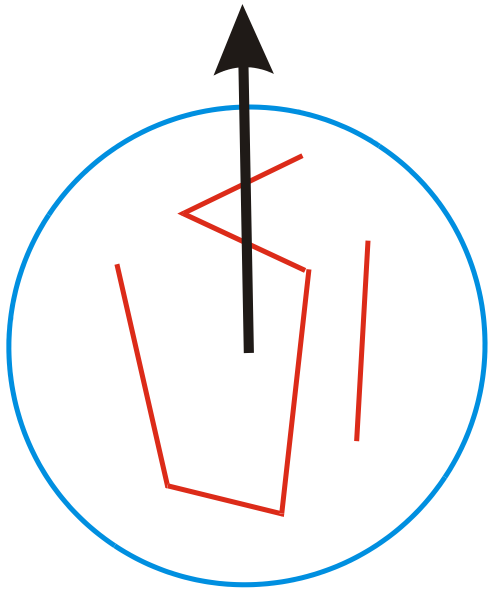
For each image sample, $L(x,y,\sigma)$, at this scale, the gradient magnitude, $m(x,y)$, and orientation, $\theta(x,y)$, is precomputed using pixel differences:

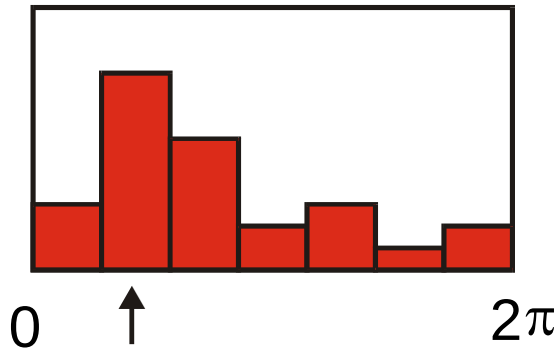
$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

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

Orientation Normalization

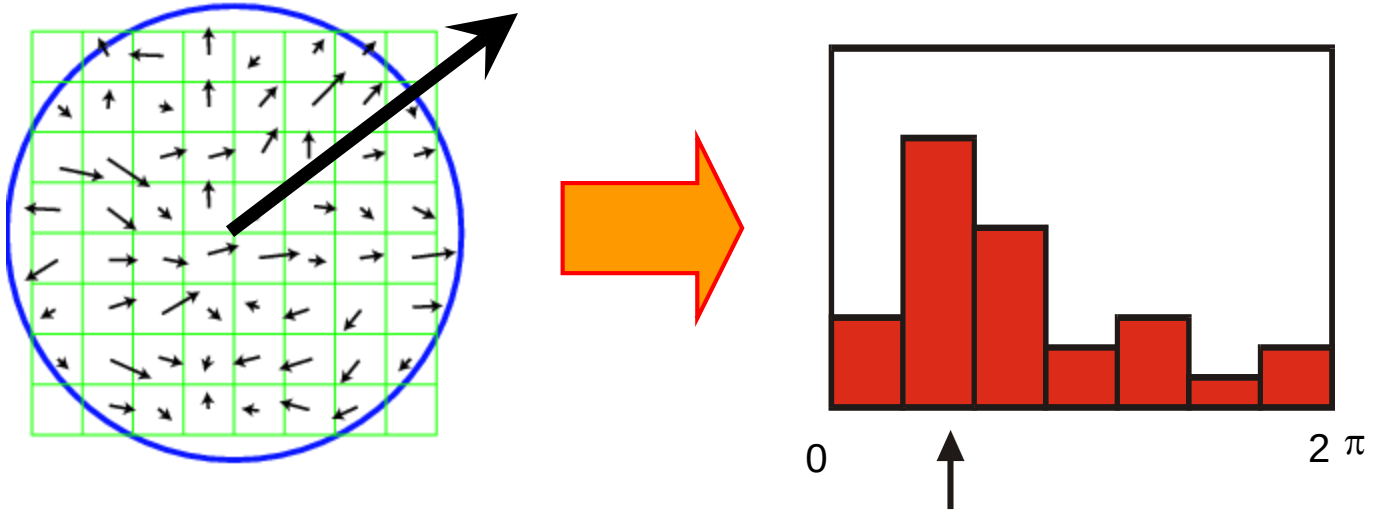
- Compute orientation histogram
- Select dominant orientation
- Normalize: rotate to fixed orientation



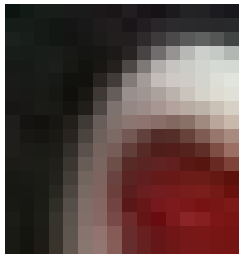


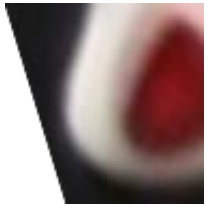
Eliminating Rotation Ambiguity

- To assign a unique orientation to circular image windows:
  - Create histogram of local gradient directions in the patch
  - Assign canonical orientation at peak of smoothed histogram

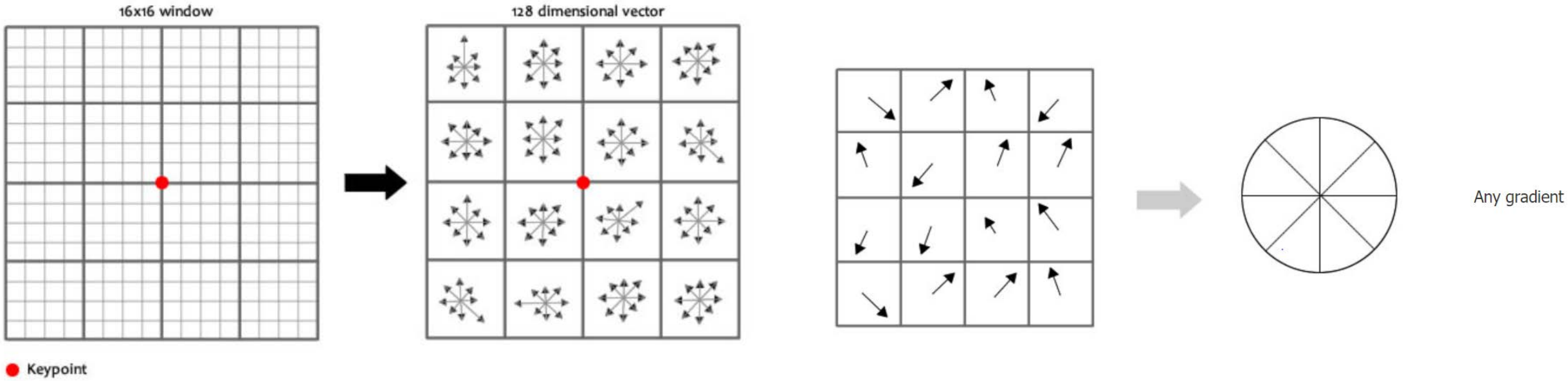


Example: Orientation





SIFT Descriptor



Feature Matching Procedure

- Each extracted feature has a 128-element descriptor vector assigned to it.

- The Euclidean distances between each feature's descriptor vector in the *reference* image and any of the feature descriptor vectors in the *input* image are computed.

- If $\frac{\text{distance between a feature in the reference image and its nearest feature in the input image}}{\text{distance between a feature in the reference image and its } 2^{nd} \text{ nearest feature in the input image}} < \tau$,

  the nearest feature is accepted as the matching feature otherwise the feature in the reference image does not have a match. Where τ is a threshold

SIFT Library

F = VL_SIFT(I) computes the SIFT frames [1] (keypoints) F of the image I. I is a gray-scale image in single precision. Each column of F is a feature frame and has the format [X;Y;S;TH], where X,Y is the (fractional) center of the frame, S is the scale and TH is the orientation (in radians).

[F,D] = VL_SIFT(I) computes the SIFT descriptors [1] as well. Each column of D is the descriptor of the corresponding frame in F. A descriptor is a 128-dimensional vector of class UINT8.

VL_SIFT() accepts the following options:

http://www.vlfeat.org/overview/sift.html

Slide Credits and References

- Lecture notes: S. Narasimhan
- Lecture notes: Gordon Wetzstein
- Lecture notes: Mohammad Jahanshahi
- Lecture notes: Noah Snavely
- Lecture notes: L. Fei-Fei
- Lecture notes: D. Frosyth
- Lecture notes: James Hayes
- Lecture notes: Yacov Hel-Or
- Lecture notes: K. Grauman, B. Leibe