

**EBU7240**

# **Computer Vision**

**- Features: edges, corners -**

*Semester 1, 2021*

**Changjae Oh**



# Content

---

- Edges
- Corners

# Content

---

- Edges
- Corners

# Derivatives and Edges

---

- **Edges: intensity discontinuities**
  - Ramp edge
  - Step edge

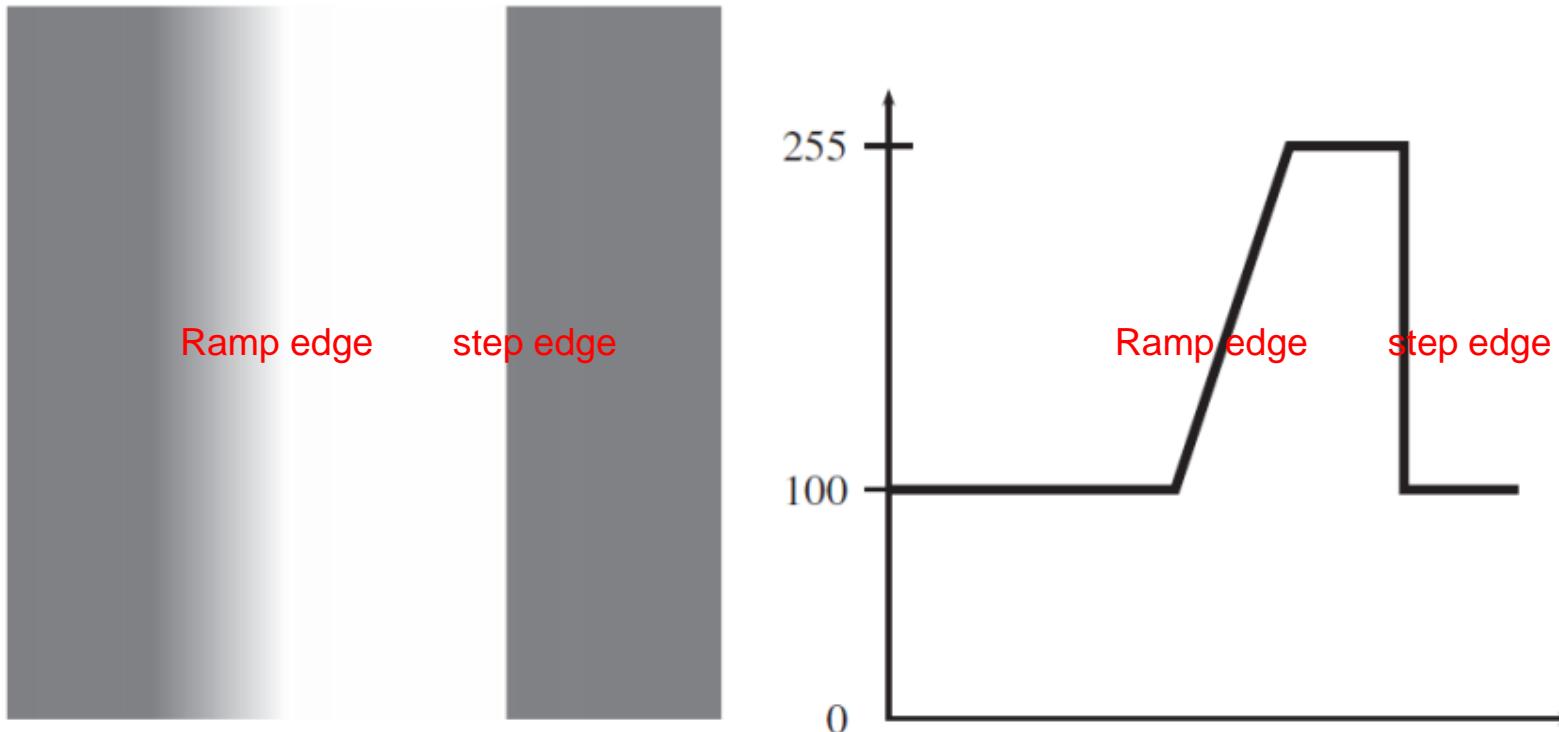


FIGURE 9.14 Edges and their profiles.

# Derivatives and Edges

---

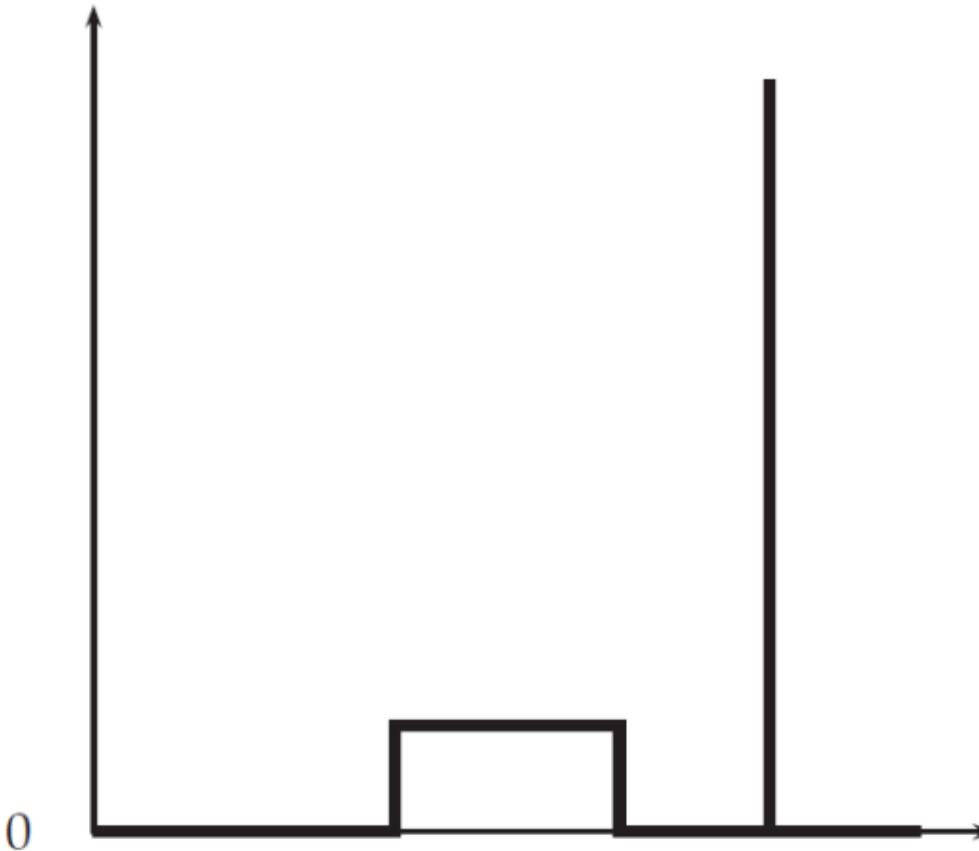
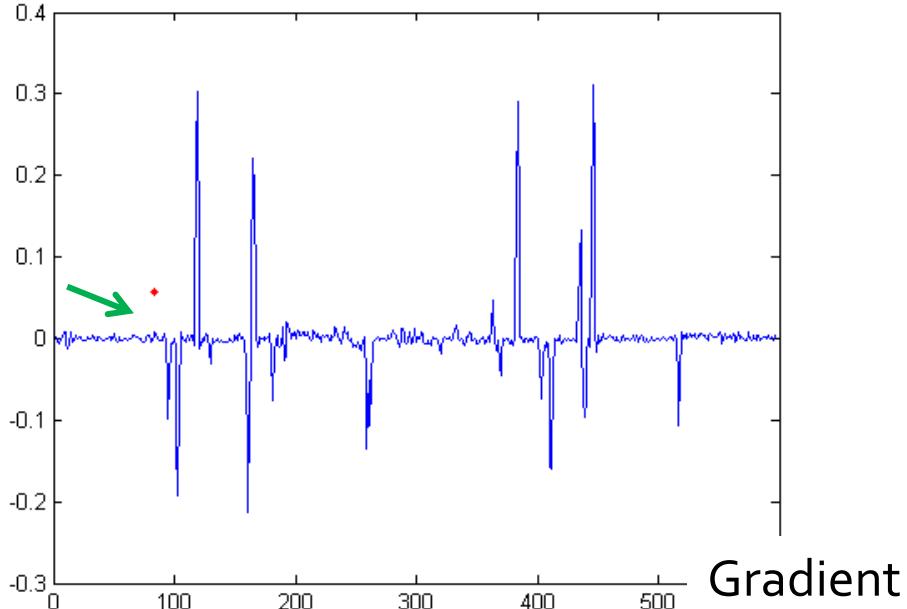
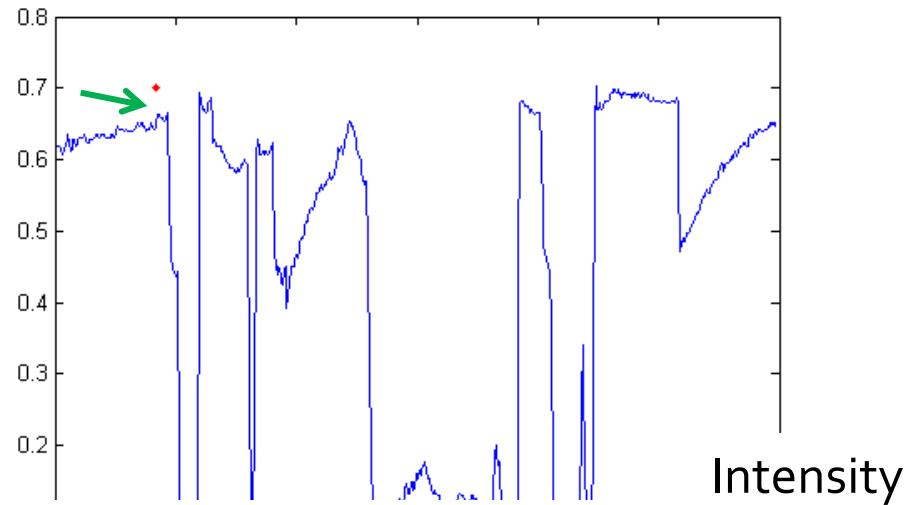
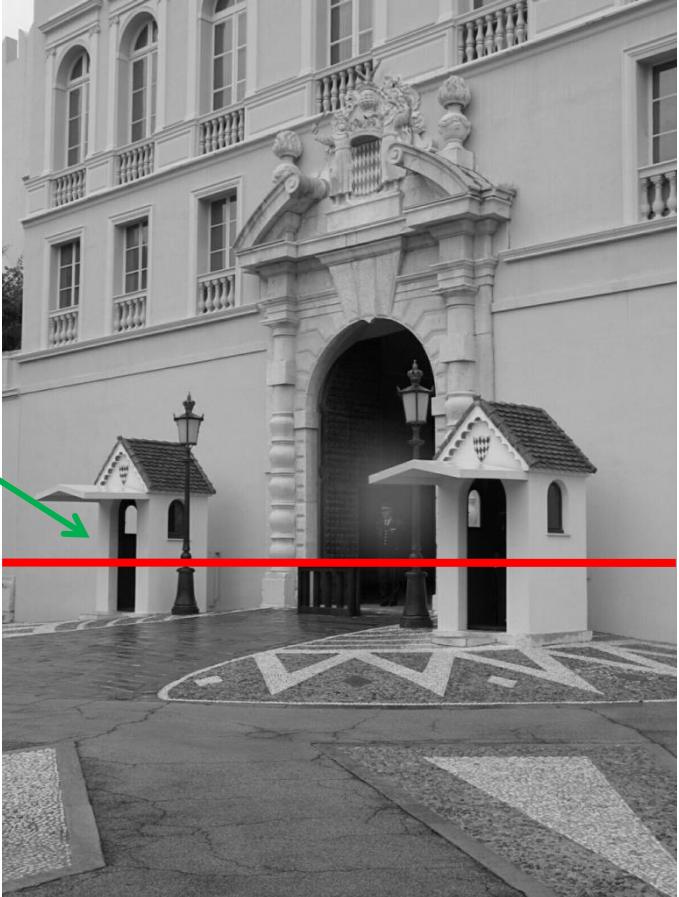


FIGURE 9.15 *The derivative of the edge profile.*

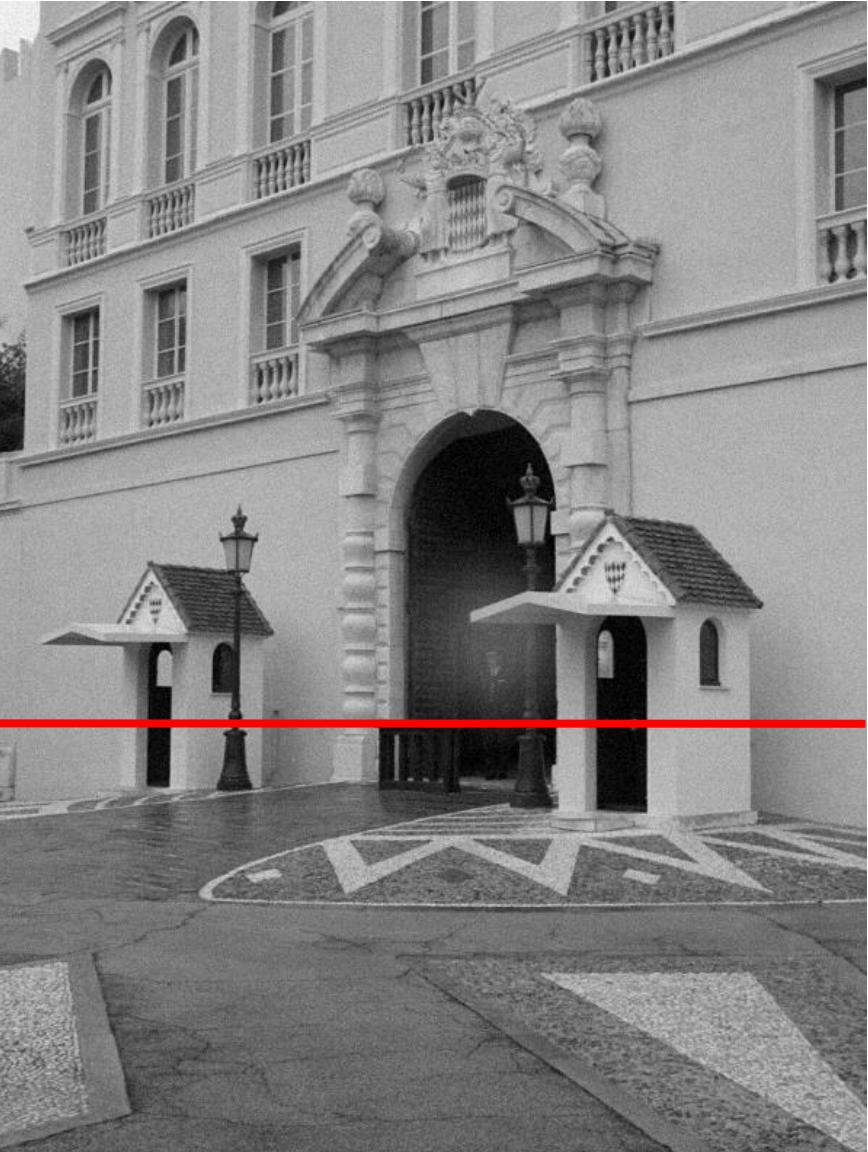
# Effects of Noise in Edge Detection

Intensity profile

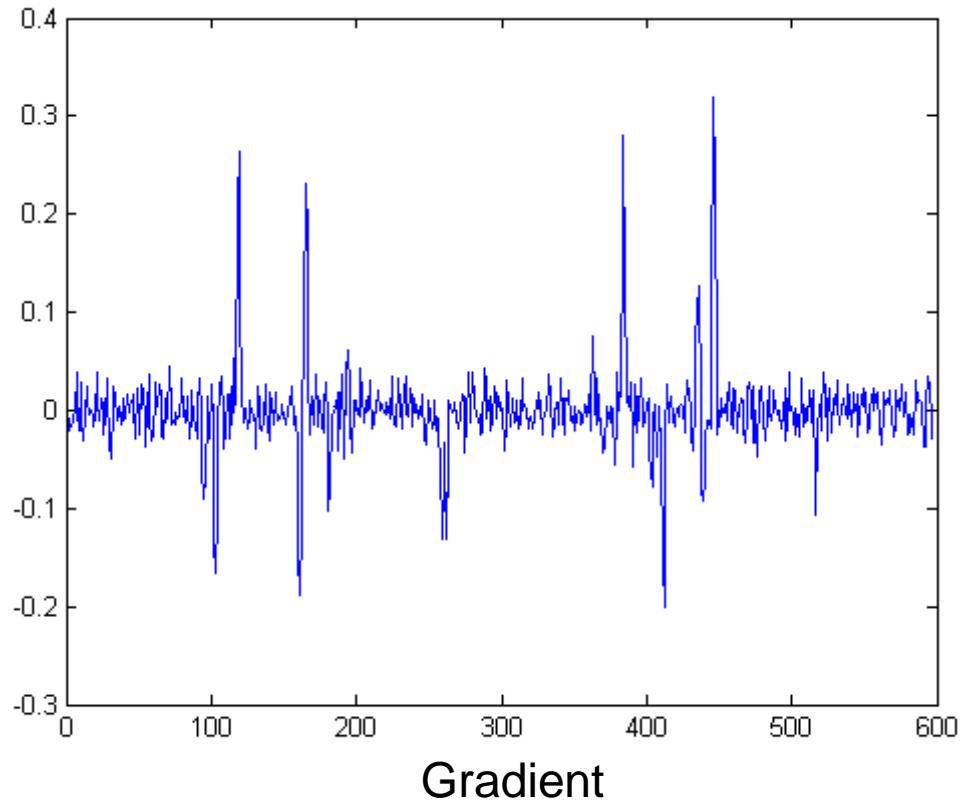


Source: D. Hoiem

# Effects of Noise in Edge Detection



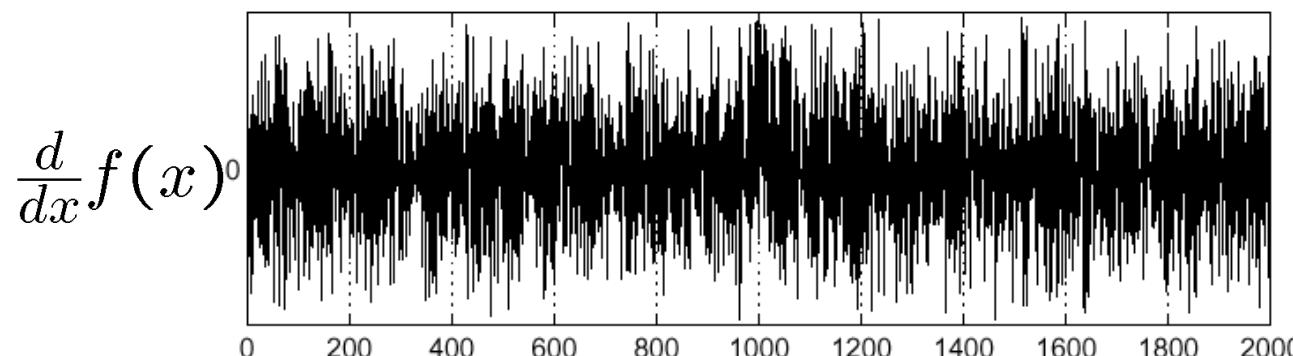
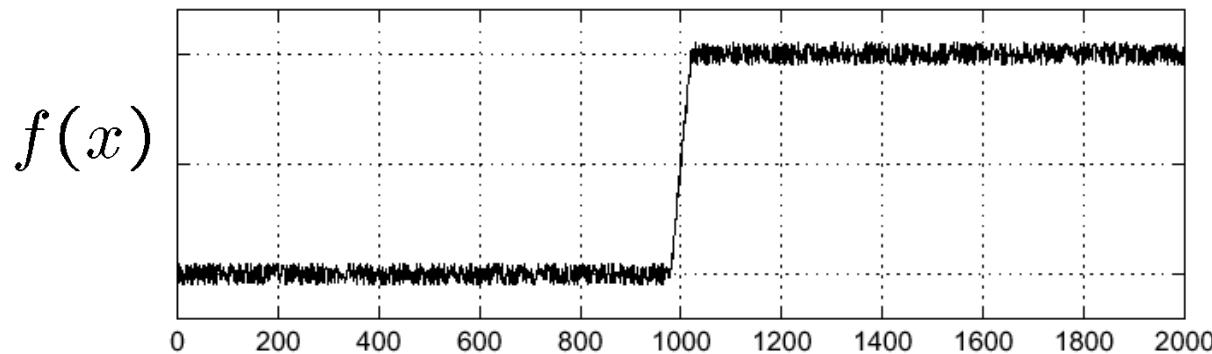
With a little Gaussian noise



Source: D. Hoiem

# Effects of Noise in Edge Detection

- Consider a single row or column of the image
  - Plotting intensity as a function of position



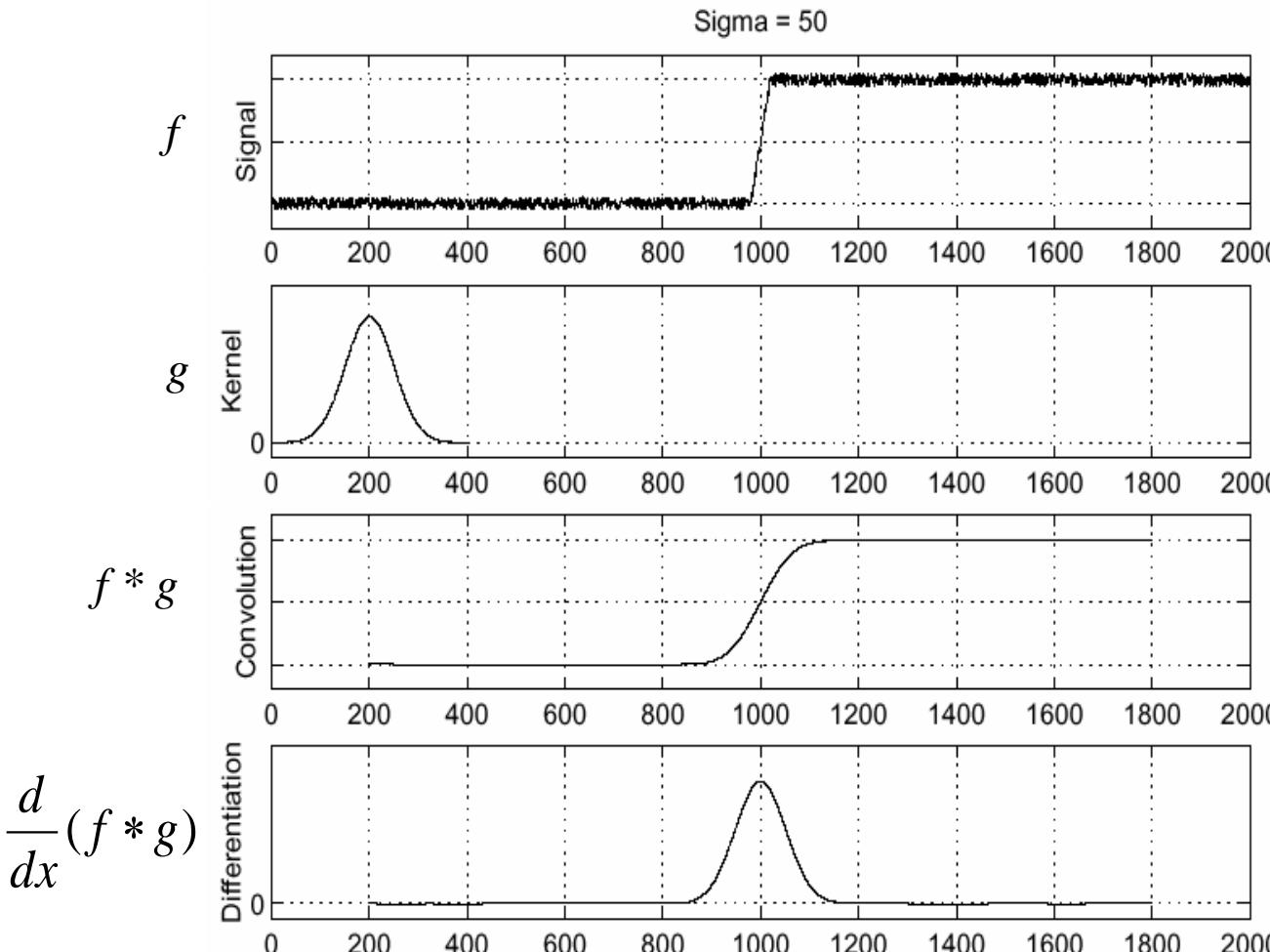
Where is the edge?

# Effects of Noise in Edge Detection

---

- **Difference filters respond strongly to noise**
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the stronger the response
- **What can we do about it?**

# Solution: Smooth First



To find edges, look for peaks in  $\frac{d}{dx}(f * g)$

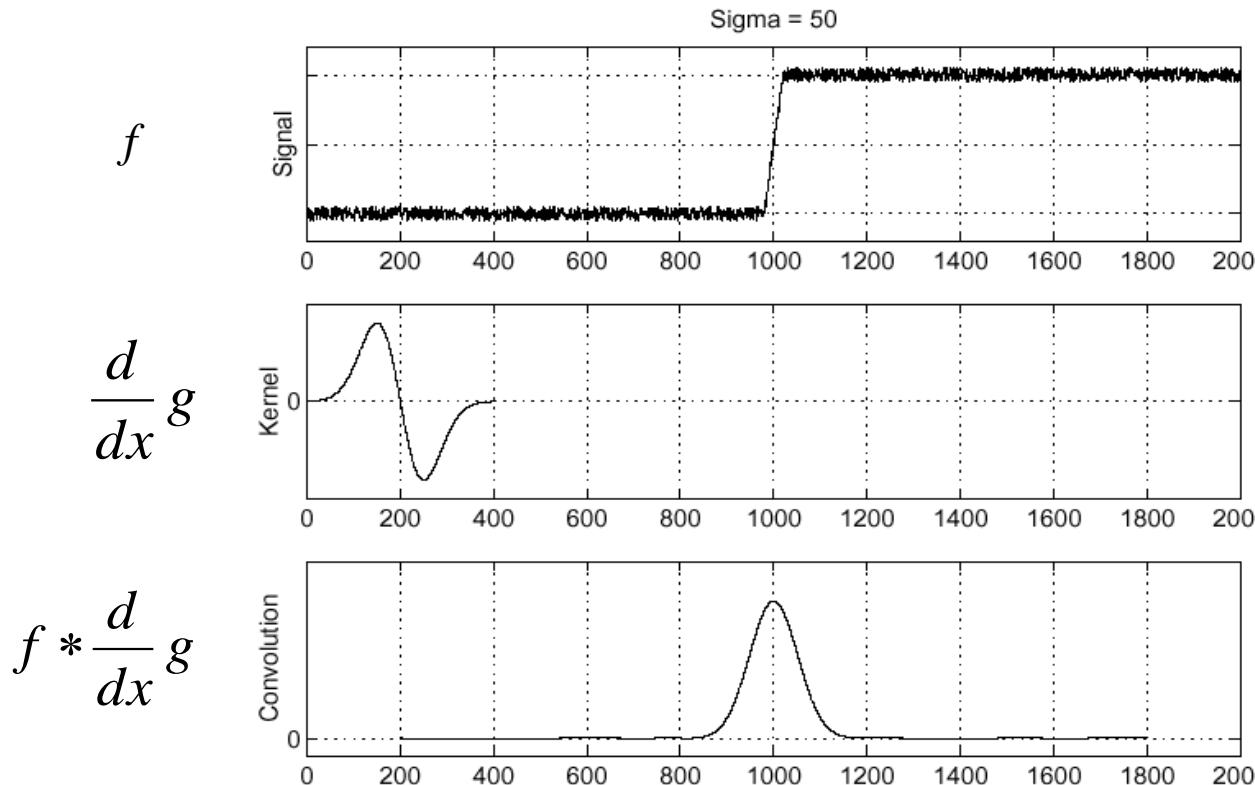
Source: S. Seitz

# Derivative Theorem of Convolution

- Differentiation is convolution, and convolution is associative:

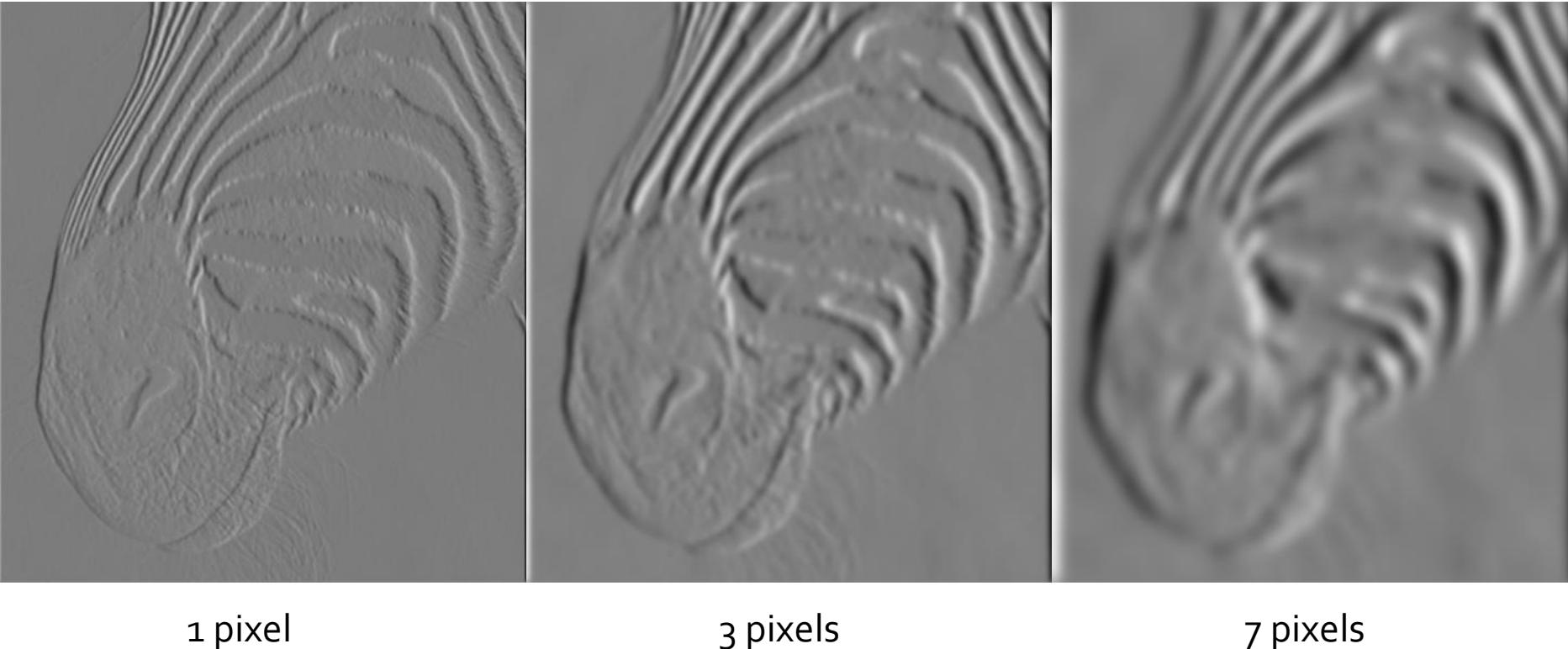
$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$$

- This saves us one operation:



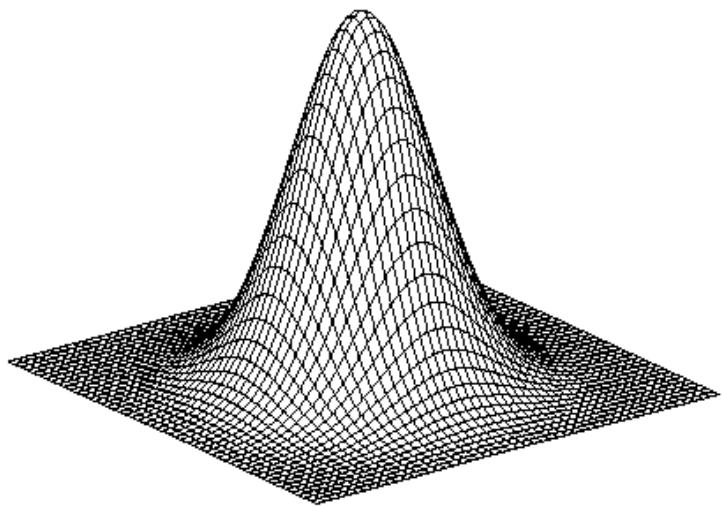
# Trade-off Between Smoothing and Localization

---

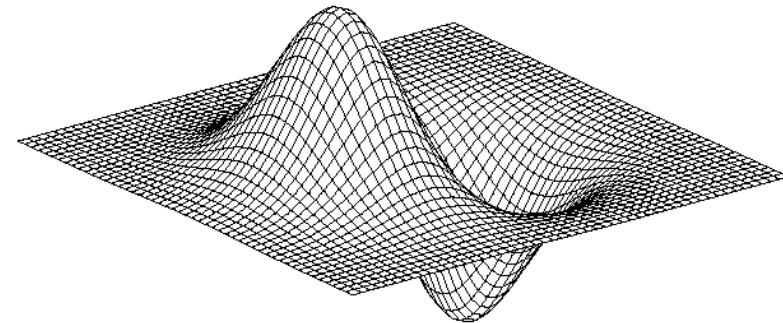


- Smoothed derivative removes noise, but blurs edge.

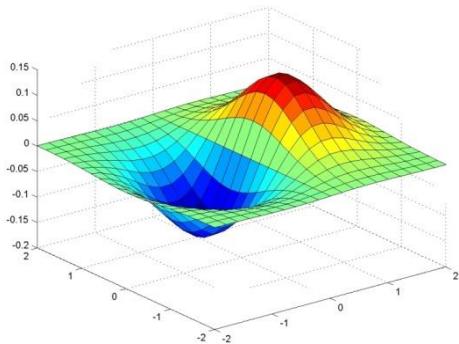
# Edge Detection: Derivative of Gaussian Filter



$\ast [1 \ -1] =$

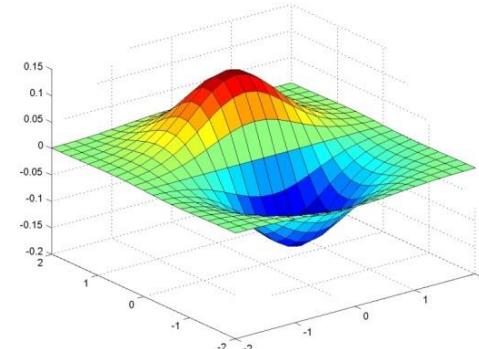


Note) Sobel operators approximate the derivative of Gaussian filter in a simple form.



$\approx$

-1	0	1
-2	0	2
-1	0	1



$\approx$

1	2	1
0	0	0
-1	-2	-1

# Edge Detection: Sobel Filter

- Sobel filter

$$\nabla I = (I_x, I_y) = (S_x * I, S_y * I)$$

Sobel filter output  $M(x, y) = \sqrt{I_x^2 + I_y^2}$  or  $|I_x| + |I_y|$

For color image,  
 $M(x, y) = (M_R(x, y) + M_G(x, y) + M_B(x, y))/3$

$S_x$

-1	0	1
-2	0	2
-1	0	1

$S_y$

-1	-2	-1
0	0	0
1	2	1

Gaussian filtering is often applied as pre-processing.

- 1) Apply Gaussian filter
- 2) Apply Sobel filter

# Edge Detection: Laplacian Filter

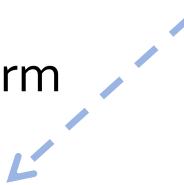
- Laplacian filter

0	1	0
1	-4	1
0	1	0

or

1	1	1
1	-8	1
1	1	1

More general form



$\frac{\alpha}{1 + \alpha}$	$\frac{1 - \alpha}{1 + \alpha}$	$\frac{\alpha}{1 + \alpha}$
$\frac{1 - \alpha}{1 + \alpha}$	$\frac{-4}{1 + \alpha}$	$\frac{1 - \alpha}{1 + \alpha}$
$\frac{\alpha}{1 + \alpha}$	$\frac{1 - \alpha}{1 + \alpha}$	$\frac{\alpha}{1 + \alpha}$

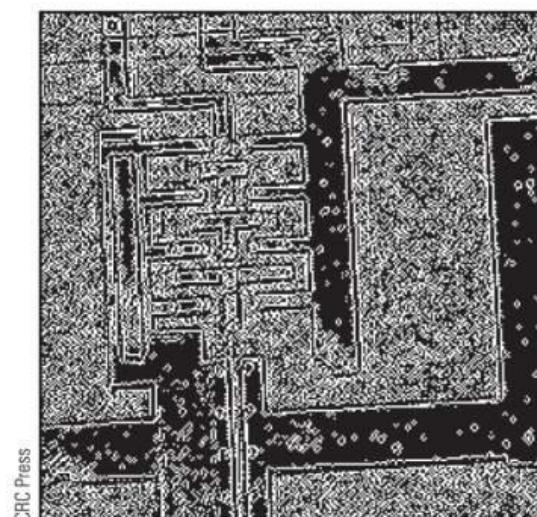
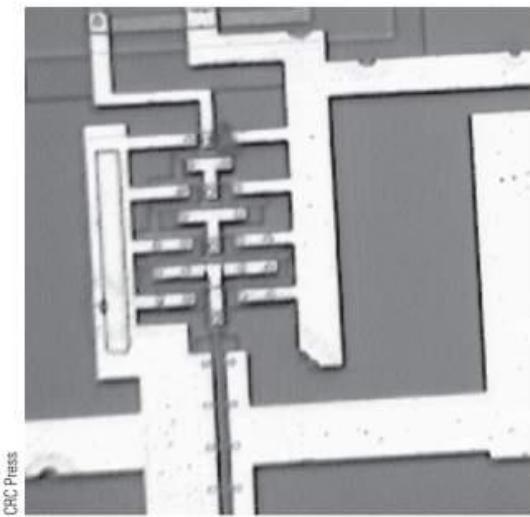


$$O = |L * I|$$

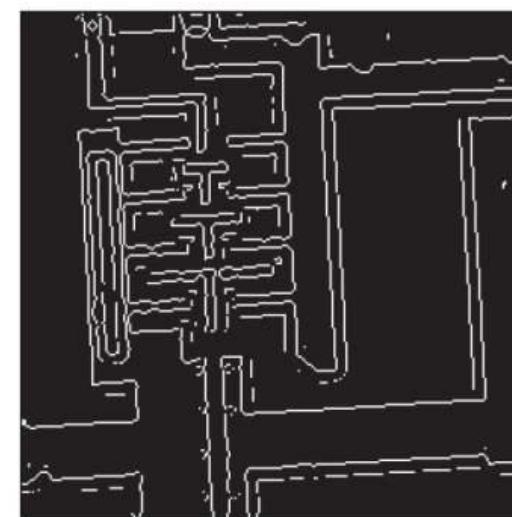
For color image,  
 $O(x, y) = (O_R(x, y) + O_G(x, y) + O_B(x, y))/3$

# Edge Detection: Laplacian of Gaussian (LoG)

- **Marr-Hildreth Method (Laplacian of Gaussian: LoG)**
  1. Apply the Gaussian filter for removing unnecessary noise.
  2. Apply the Laplacian filter



(a) Without Gaussian smoothing

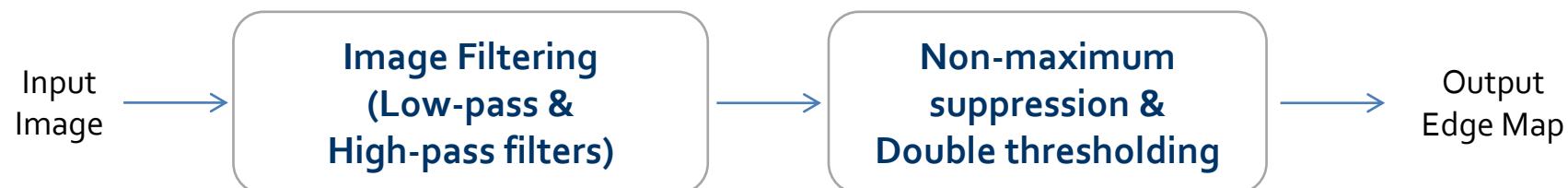


(b) With Gaussian smoothing

# Canny Edge Detector

---

- The most popular method for edge detection
- Three criteria for edge detection
  - Low error rate of detection: finding all edges
  - Localization of edges: computing precise locations of edges
  - Single response: returning a single pixel for a single edge



# Canny Edge Detector

## Step 1) Image Gradient using Low-pass & High-pass filters

1. Apply low-pass and high-pass filters

ex)

- Gaussian filter → Sobel filter ('default implementation in OpenCV')
- 1D derivative of Gaussian filter
- Difference of Gaussian filter

$$\begin{array}{c} S_x \quad \quad \quad S_y \\ \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \quad \quad \quad \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \end{array}$$

$$\nabla G = (G_x, G_y) = \left( \frac{\partial G}{\partial x}, \frac{\partial G}{\partial y} \right)$$

2. Compute the magnitude and angle of edge

$$M(x, y) = \sqrt{G_x^2 + G_y^2} \quad \quad \quad A(x, y) = \tan^{-1} \left( \frac{G_y}{G_x} \right)$$

# Canny Edge Detector

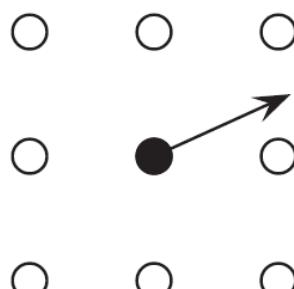
## Step 2) Non-maximum suppression

- Key idea: Survive only pixels with **a larger edge magnitude**  $M(x, y)$  within a small window

$$M(x, y) = \sqrt{G_x^2 + G_y^2} \quad A(x, y) = \tan^{-1} \left( \frac{G_y}{G_x} \right)$$

- **Procedure**

1. Within a small window (e.g.  $3 \times 3$  window) centered at  $(x, y)$ , find neighbor pixels in direction  $A(x, y)$
2. Compare the edge magnitudes  $M(x, y)$  of these two neighbor pixels



The edge direction at a pixel

FIGURE 9.26 Nonmaximum suppression in the Canny edge detector.

# Canny Edge Detector

- But, the image is a **discrete** signal.
  - So, quantize the gradient direction

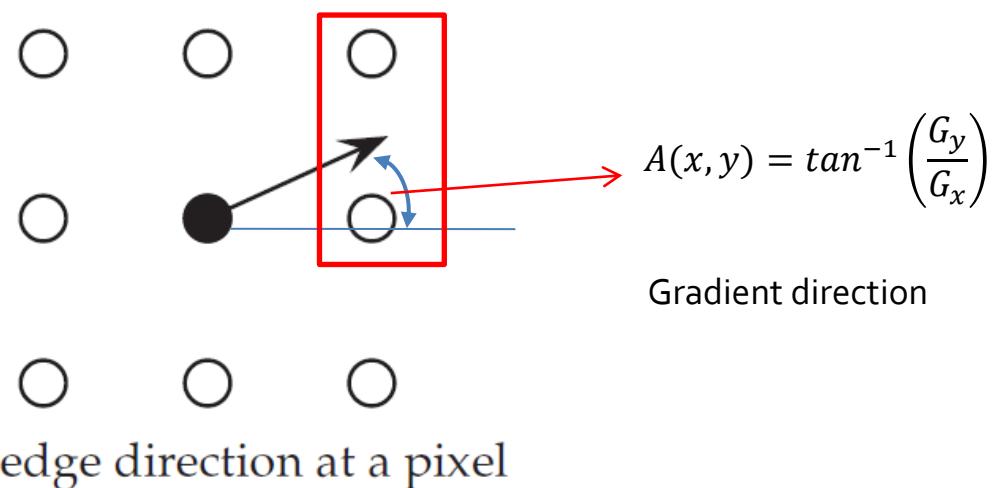
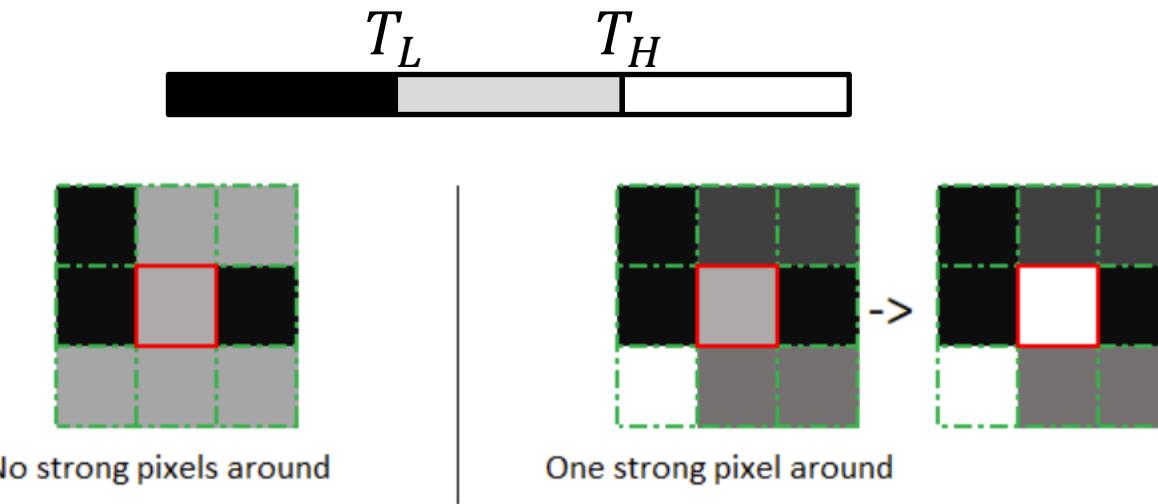


FIGURE 9.26 Nonmaximum suppression in the Canny edge detector.

# Canny Edge Detector

## Step 3) Double thresholding

- Hysteresis thresholding ( $T_L, T_H$ )



- If  $M(x, y) > T_H$ , then  $(x, y)$  is an edge
- If  $M(x, y) < T_L$ , then  $(x, y)$  is **NOT** an edge
- If  $T_L \leq M(x, y) \leq T_H$ ,
  - If the **neighboring** pixels of  $(x, y)$  is an edge, then  $(x, y)$  is an edge.
  - Otherwise, then  $(x, y)$  is **NOT** an edge.

# Canny Edge Detector Example

---



original image (Lena)

# Compute Gradients

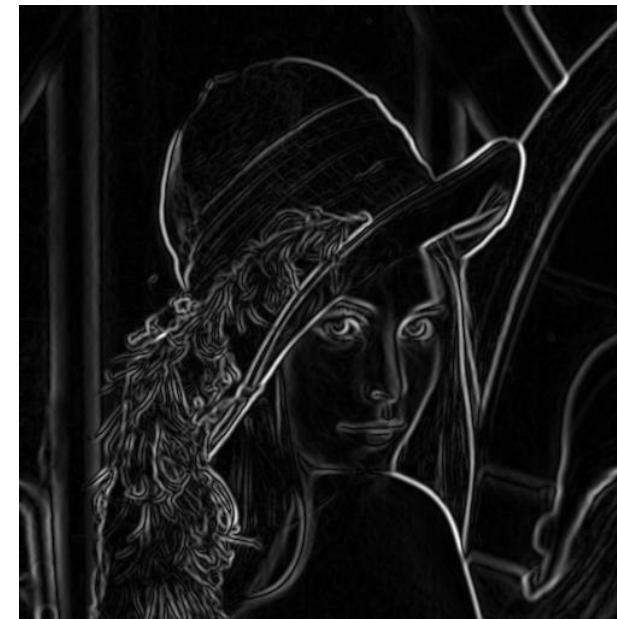
---



X-Derivative of Gaussian



Y-Derivative of Gaussian



Gradient Magnitude

# Before Non-max Suppression

---



# After Non-max Suppression

---



# Final Canny Edges

---

- After applying Hysteresis thresholding (Double thresholding)



# Content

---

- Edges
- Corners

# Interest points: Corners

---



Slides from Rick Szeliski, Svetlana Lazebnik, and Kristin Grauman

# Corners: Applications

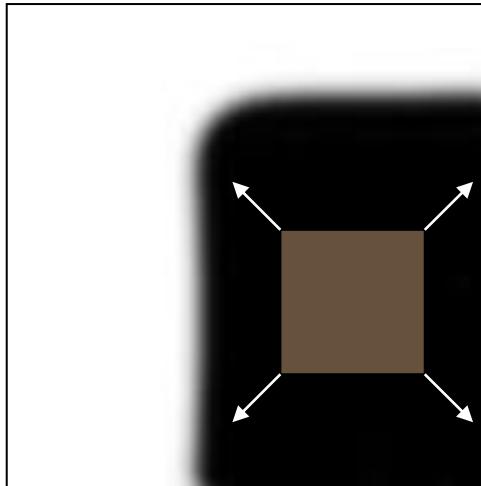
---

- **Corners are used for:**
  - Image alignment/3D reconstruction
    - Find correspondences across different views
  - Motion tracking/robot navigation
    - Which points are good to track?
  - Object recognition/Indexing and database retrieval
    - Find patches likely to tell us something about object category

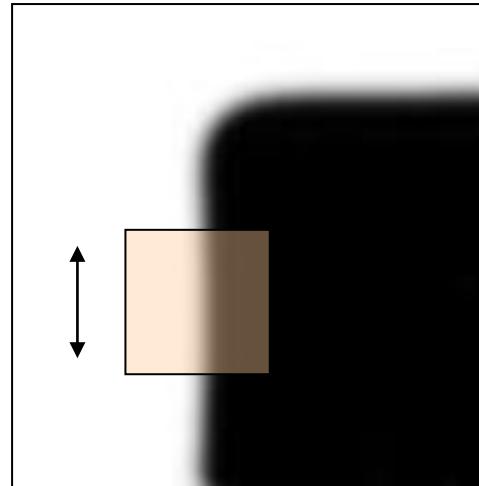


# Corner Detection: Basic Idea

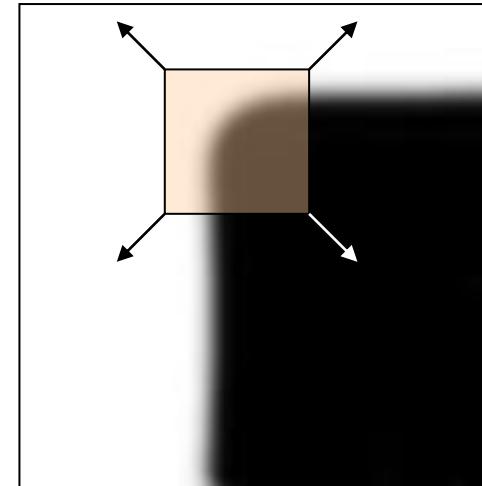
- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give a *large change* in intensity



“flat” region:  
no change in all di  
rections



“edge”:  
no change along the  
edge direction

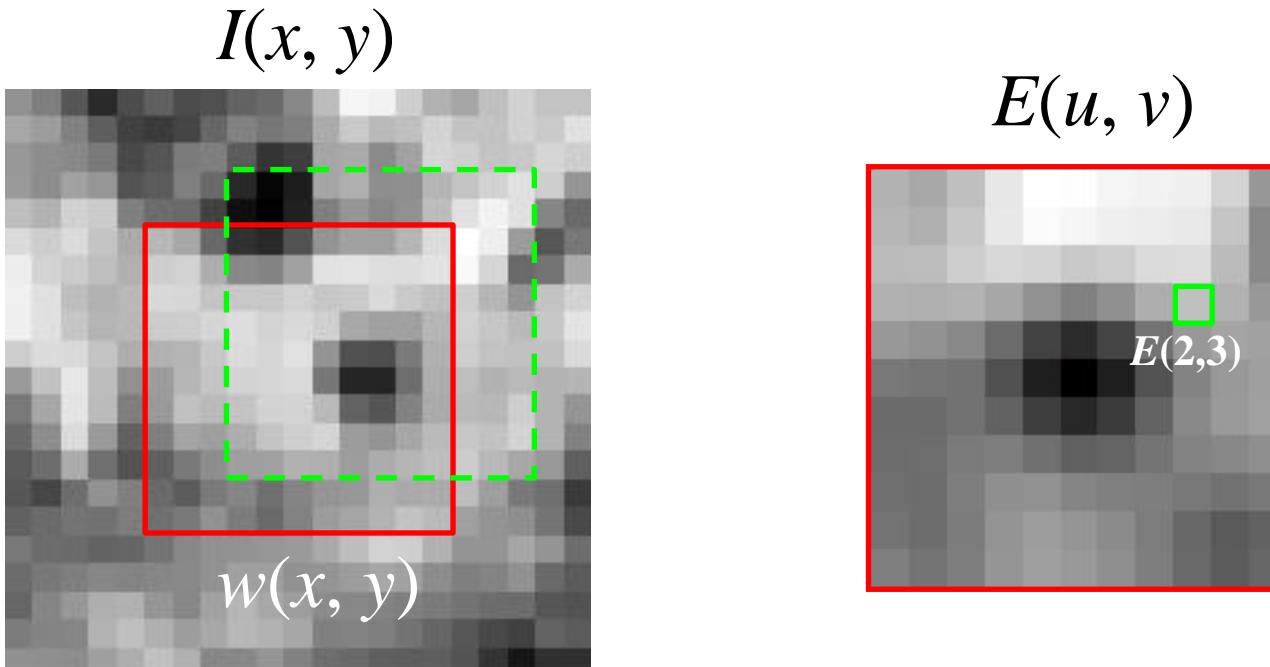


“corner”:  
significant change in  
all directions

# Corner Detection: Mathematics

Change in appearance of window  $w(x,y)$   
for the shift  $[u,v]$ :

$$E(u, v) = \sum_{x, y} w(x, y) [I(x + u, y + v) - I(x, y)]^2$$



# Corner Detection: Mathematics

---

The quadratic approximation simplifies to

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

where  $M$  is a *second moment matrix* computed from image derivatives:

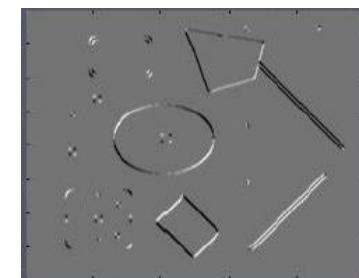
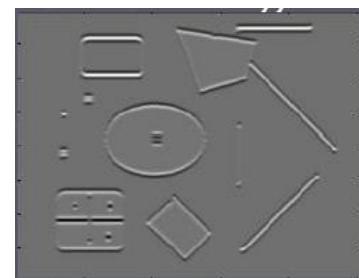
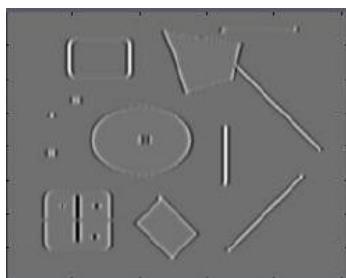
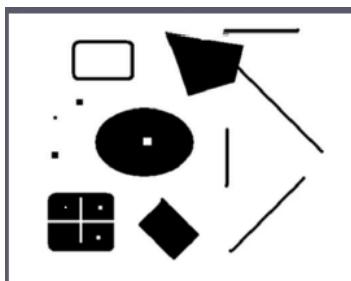
$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

$$M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \ I_y] = \sum \nabla I (\nabla I)^T$$

# Corners as Distinctive Interest Points

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

2 x 2 matrix of image derivatives  
(averaged in neighborhood of a point)



Notation:

$$I_x \Leftrightarrow \frac{\partial I}{\partial x}$$

$$I_y \Leftrightarrow \frac{\partial I}{\partial y}$$

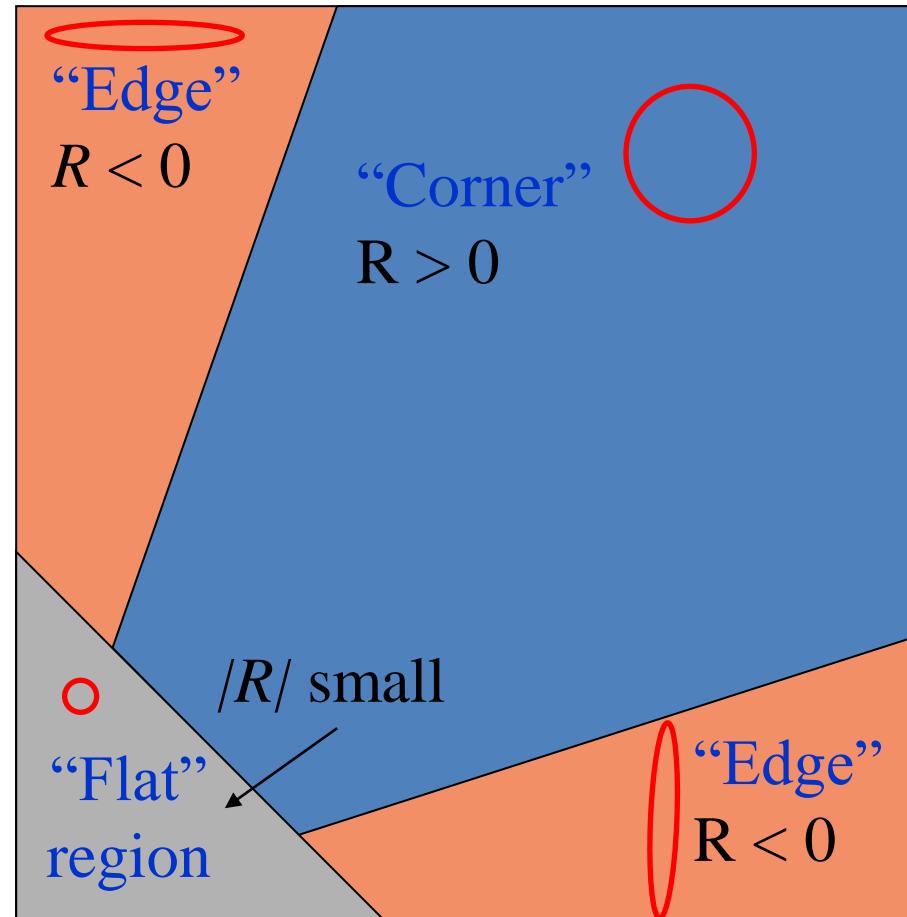
$$I_x I_y \Leftrightarrow \frac{\partial I}{\partial x} \frac{\partial I}{\partial y}$$

# Corner Response Function

$$R = \det(M) - \alpha \operatorname{trace}(M)^2 = \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

$\alpha$ : constant (0.04 to 0.06)

Note) The eigenvalues are not needed. Instead, we can use the determinant and trace of the second moment matrix.



# Harris Corner Detector: Procedure

---

Step 1) Compute  $M$  matrix for each window to get their *cornerness scores*.

Step 2) Find points whose surrounding window gave large corner response ( $R >$  threshold)

Step 3) Take the points of local maxima,  
i.e., perform non-maximum suppression

# Harris Corner Detector: Input

---

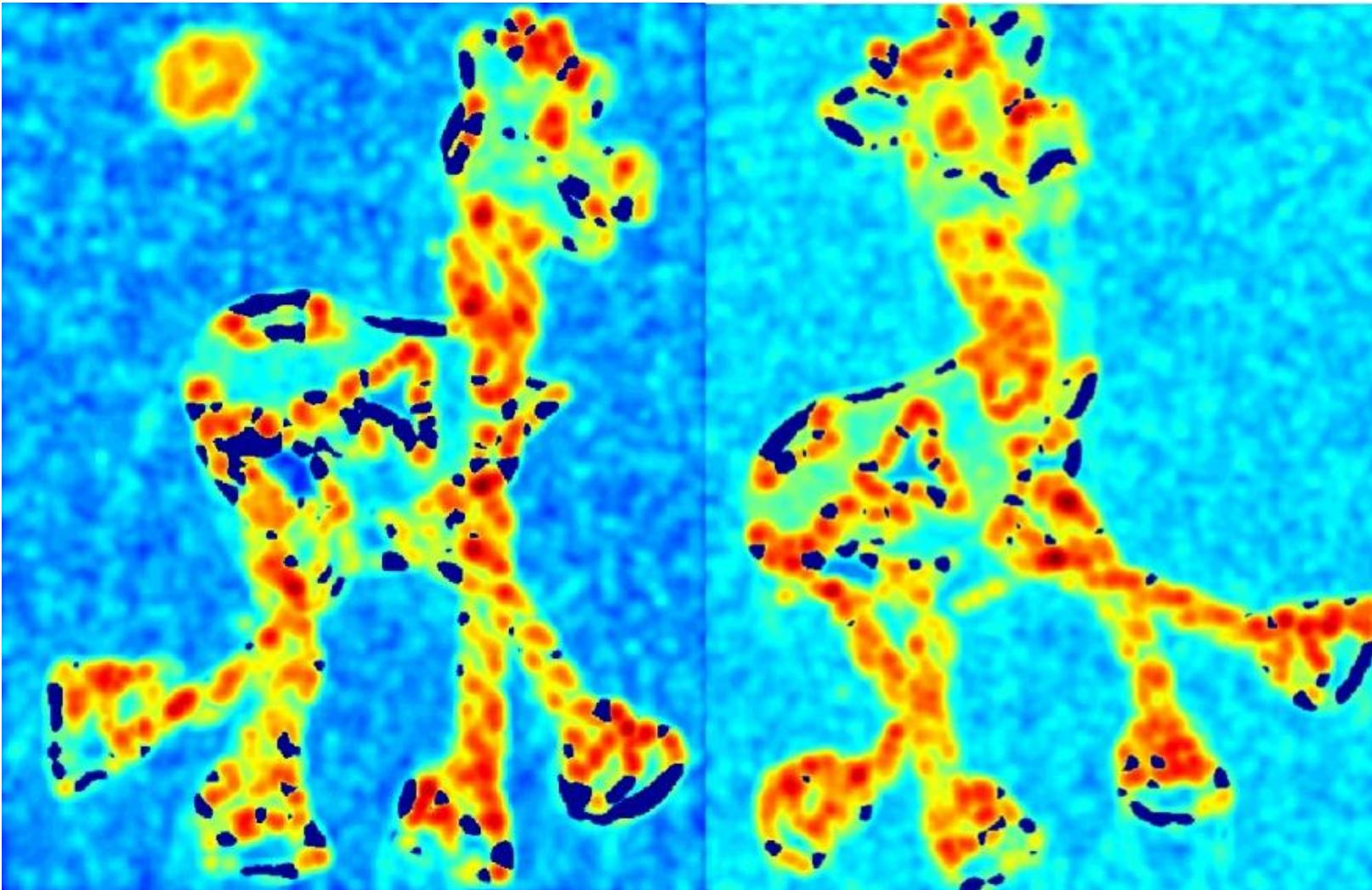
Performing corner detection to each image independently



# Harris Corner Detector: Step 1)

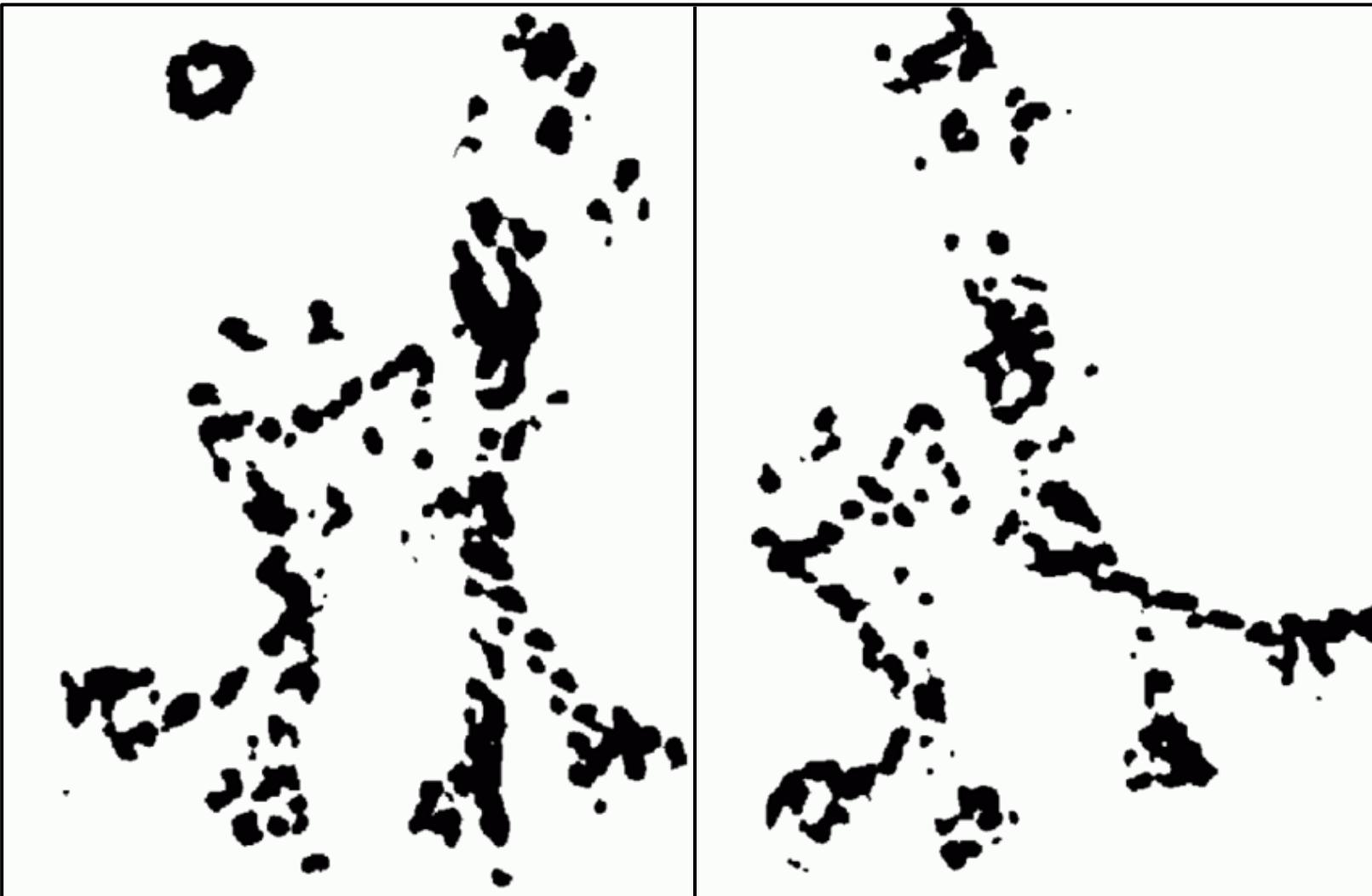
---

Compute corner response  $R$



# Harris Corner Detector: Step 2)

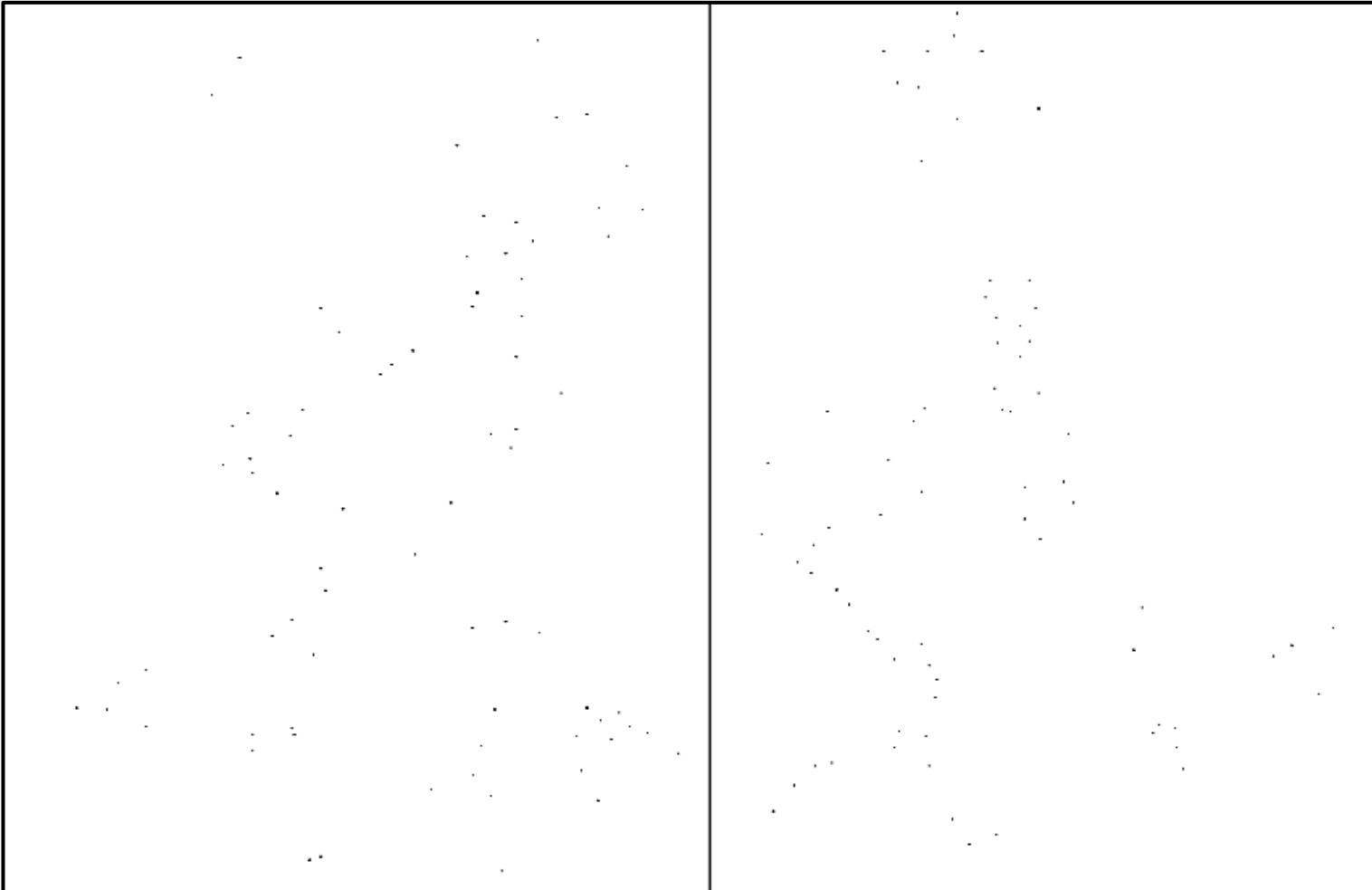
Find points with large corner response:  $R > \text{threshold}$



# Harris Corner Detector: Step 3)

---

Take only the points of local maxima of  $R$



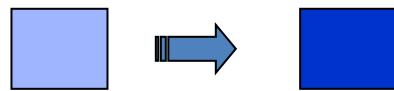
# Harris Corner Detector: Results

---

Advantages? Drawbacks?

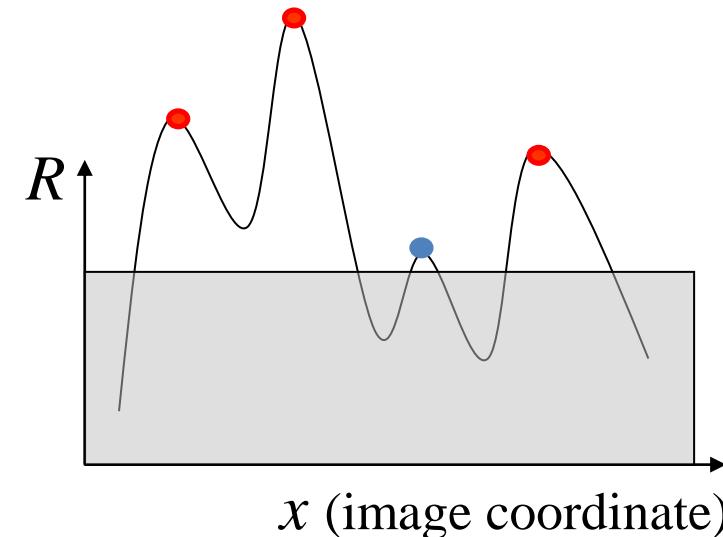
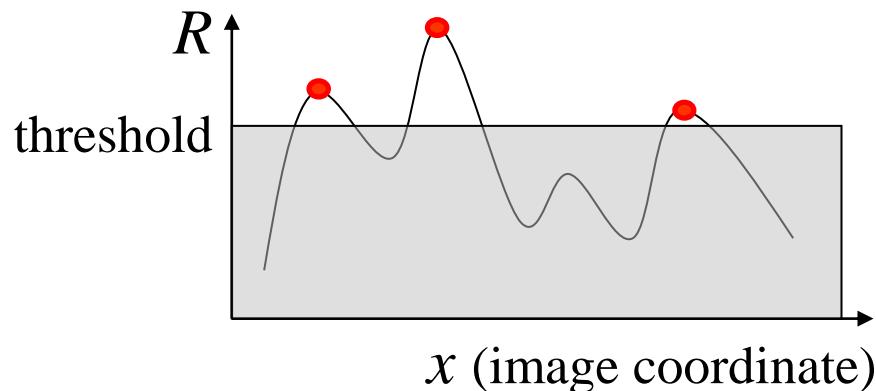


# Affine intensity change



$$I \rightarrow aI + b$$

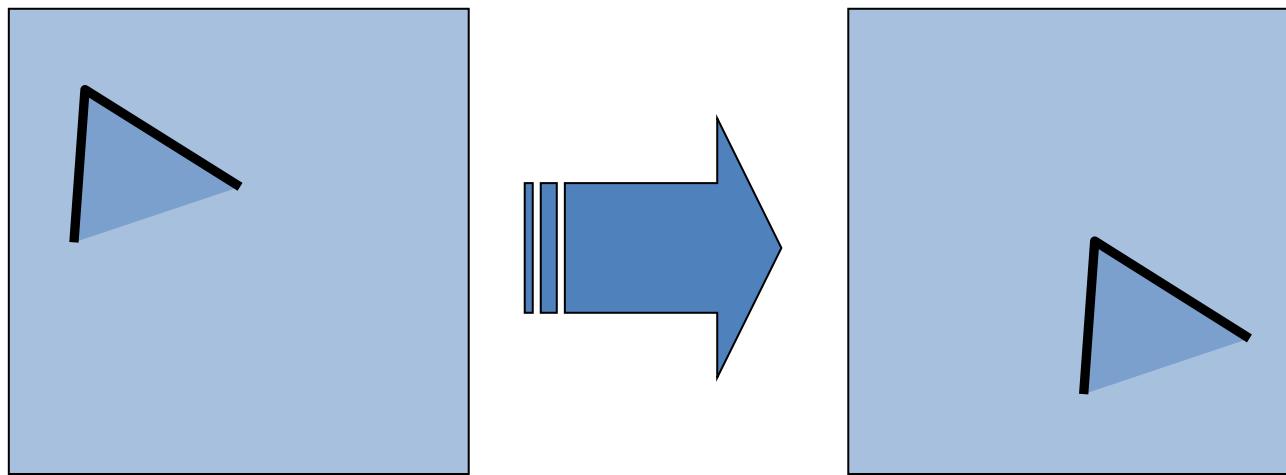
- Only derivatives are used  
=> invariance to intensity shift:  $I \rightarrow I + b$
- Intensity scaling:  $I \rightarrow aI$



- *Partially invariant* to affine intensity change

# Image translation

---

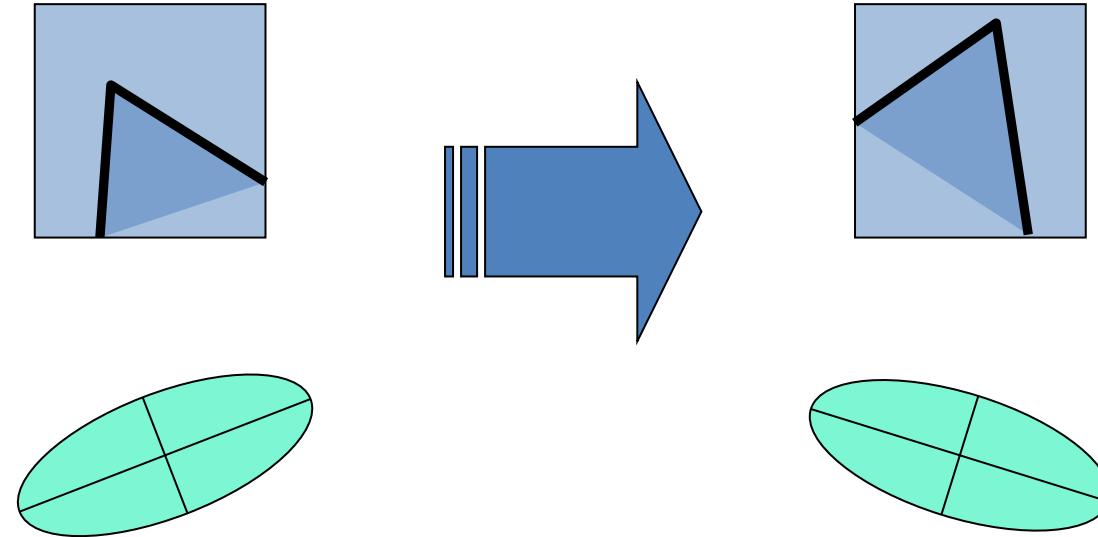


- Derivatives and window function are shift-invariant

Corner location is covariant w.r.t. translation

# Image rotation

---

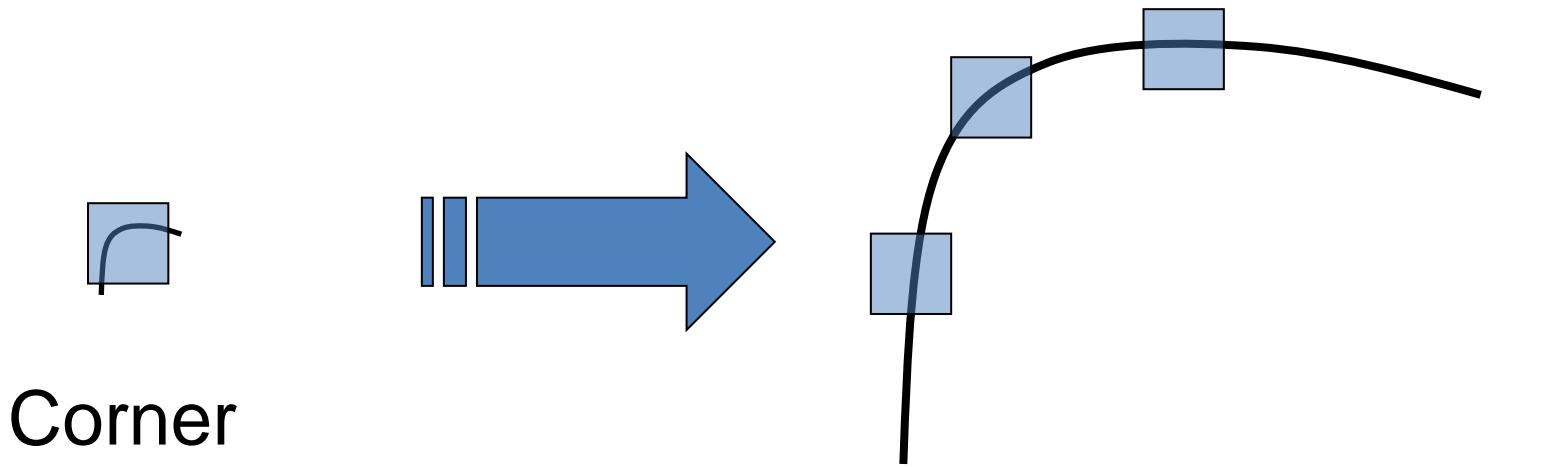


Second moment ellipse rotates but its shape (i.e.  
. eigenvalues) remains the same

Corner location is covariant w.r.t. rotation

# Scaling

---

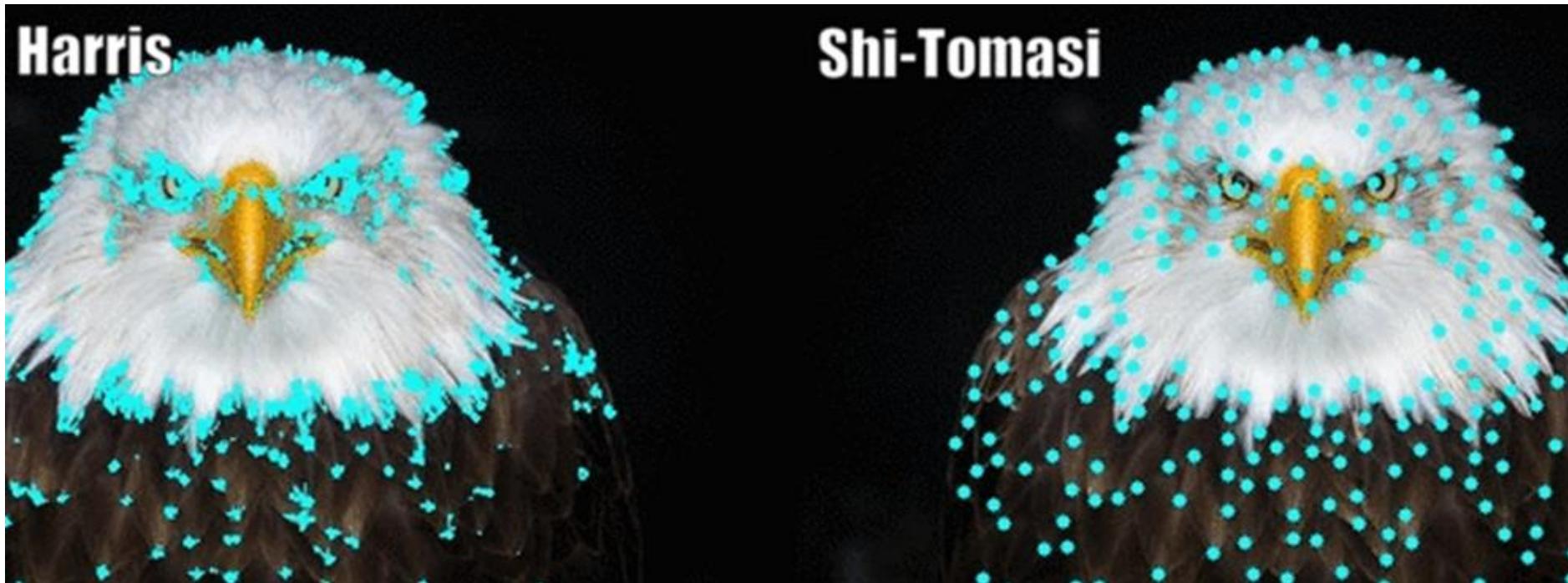


All points will be  
classified as **edges**

Corner location is not covariant to scaling!

# Not always the best

---



# Corner detection: pros & cons

---

- Corner detection can localize in x-y, but not scale



# Summary

---

- **Edge detection**
  - Identify sudden changes (discontinuities) in an image
  - Derivative of Gaussian, Sobel, Laplacian, Laplacian of Gaussian, Canny detector
- **Canny edge detector**
  - The most popular edge detector
    1. Applying low- and high-pass filtering
    2. Non-maximum suppression
    3. Double thresholding
- **Harris corner detector**
  - 1. Compute the second moment matrix
  - 2. Non-maximum suppression

**EBU7240**

# **Computer Vision**

**- Features: SIFT, SURF, and matching -**

*Semester 1, 2021*

**Changjae Oh**

# Contents

---

- **Feature Descriptor**
  - Scale Invariant Feature Transform (SIFT) descriptor
  - Speeded Up Robust Features (SURF) descriptor
- **Feature Matching**
  - Nearest Neighbor (NN) Matching

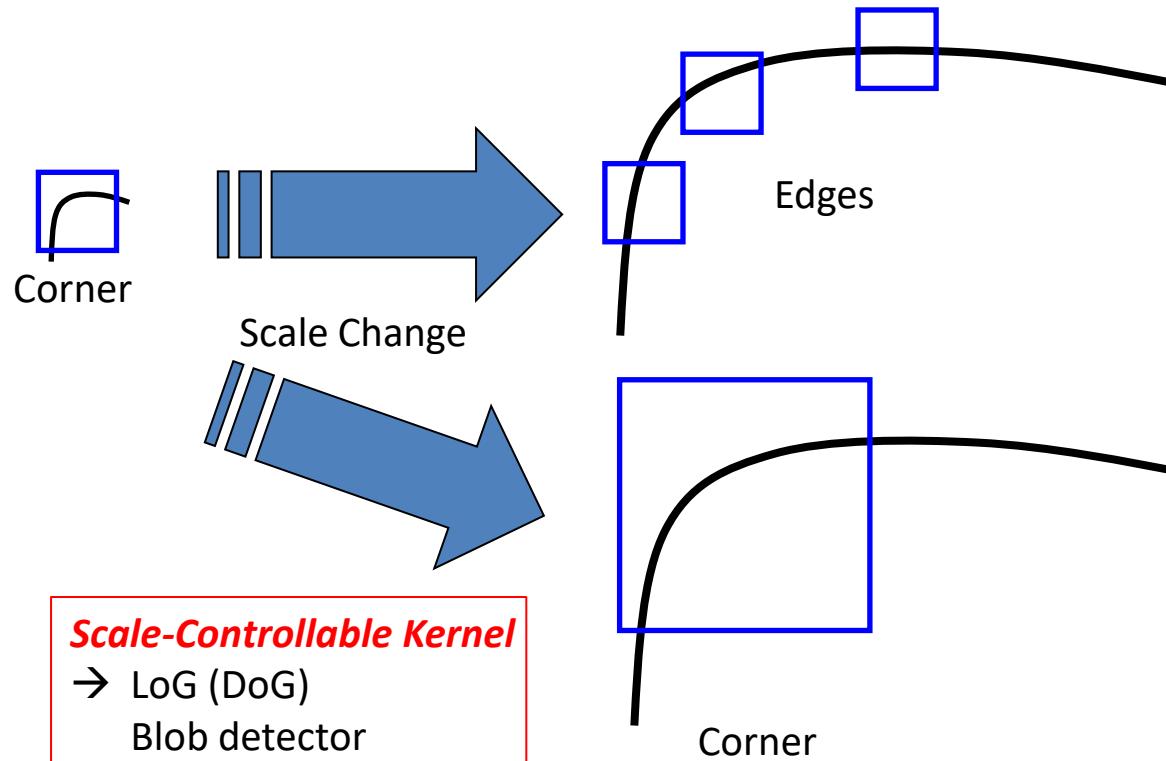
# Contents

---

- **Feature Descriptor**
  - Scale Invariant Feature Transform (SIFT) descriptor
  - Speeded Up Robust Features (SURF) descriptor
- **Feature Matching**
  - Nearest Neighbor (NN) Matching

# Harris corner detector – Pros & Cons

- Robust against shift/rotation transformations and brightness changes,
- Not robust against scale changes.
- Solution: Scale invariant blob detector by SIFT

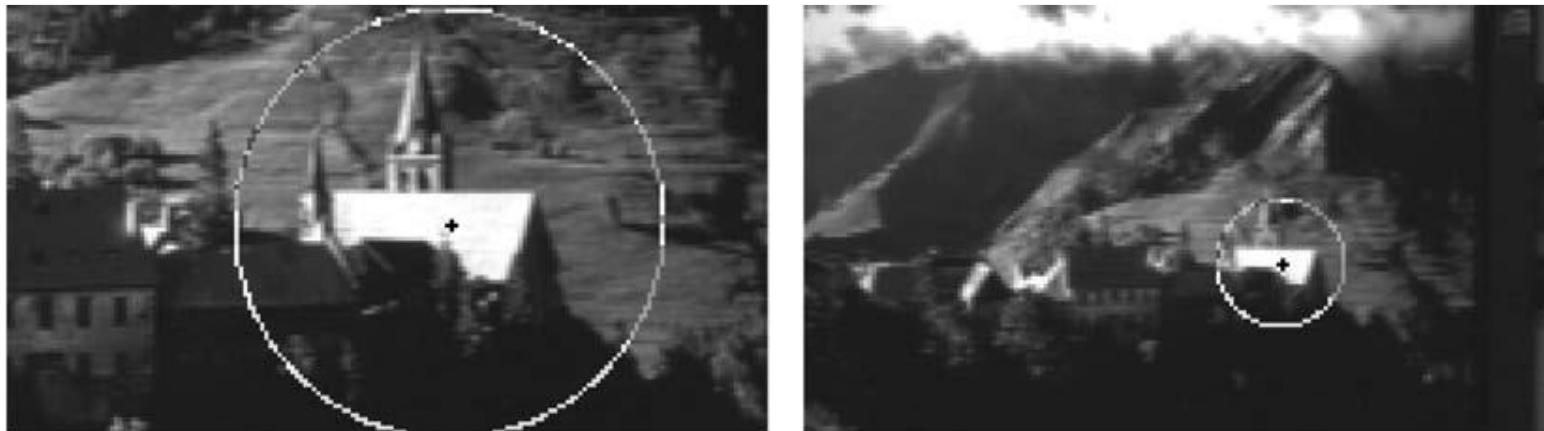


[Ref: Distinctive Image Features from Scale-Invariant Keypoints, IJCV 2004]

# Keypoint detection with scale selection

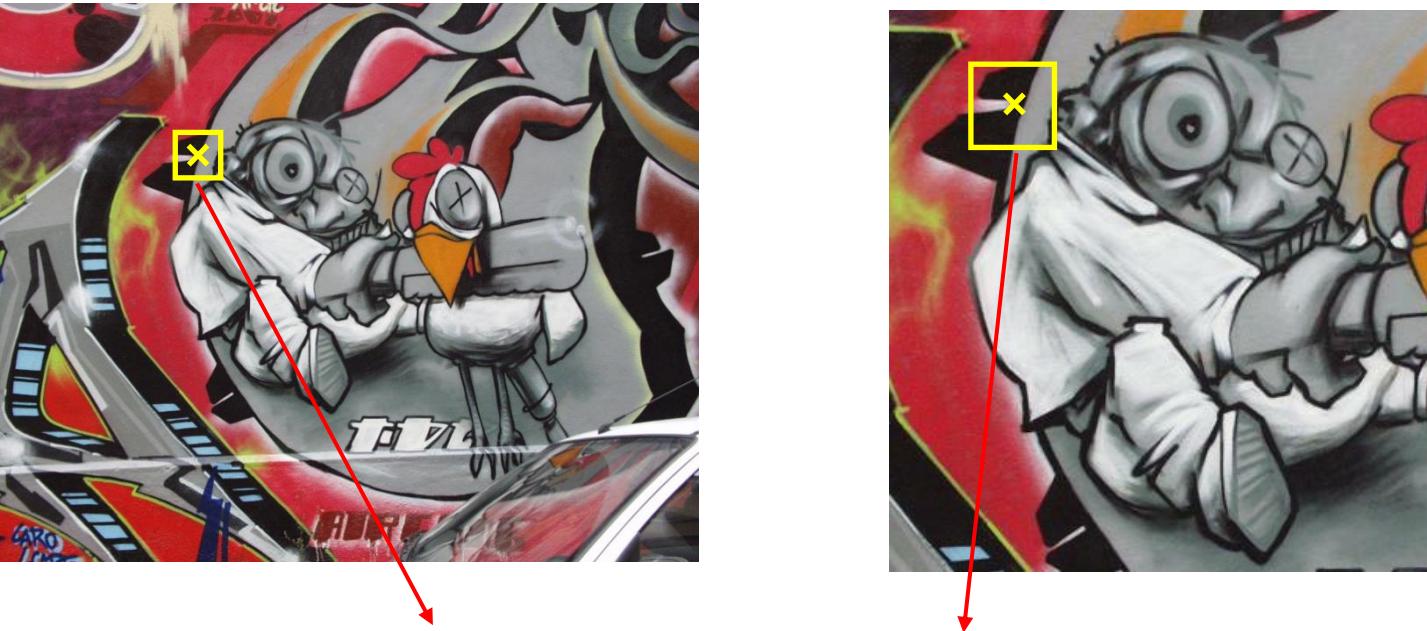
---

- We want to extract keypoints with characteristic scales that are covariant with respect to the image transformation



# Automatic scale selection

---

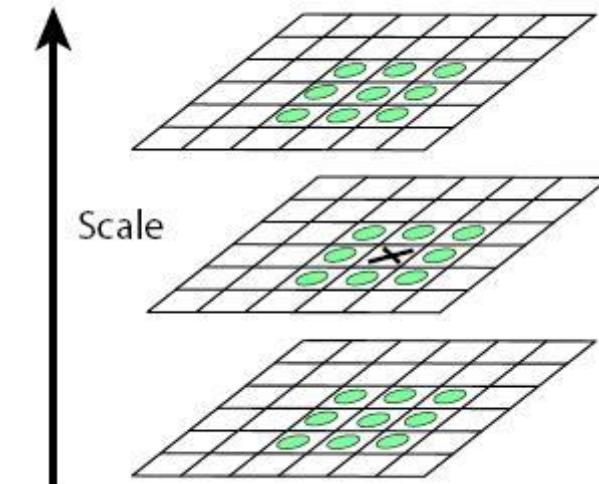
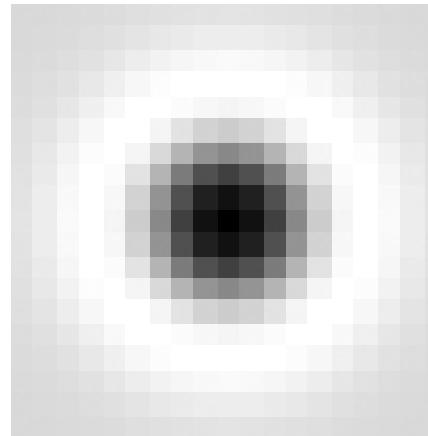
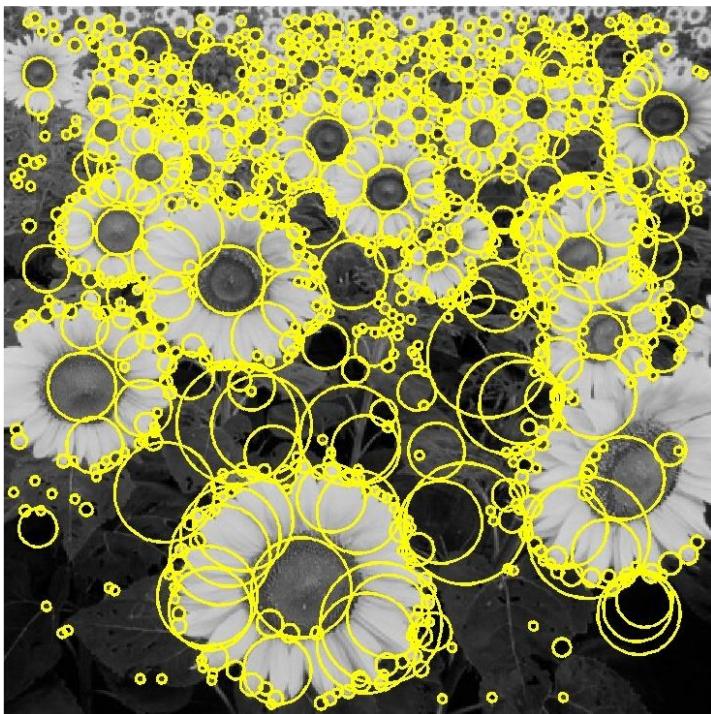


$$f(I_{i_1 \dots i_m}(x, \sigma)) = f(I_{i_1 \dots i_m}(x', \sigma'))$$

How to find corresponding patch sizes?  
**A:** Blob detector

# Basic idea

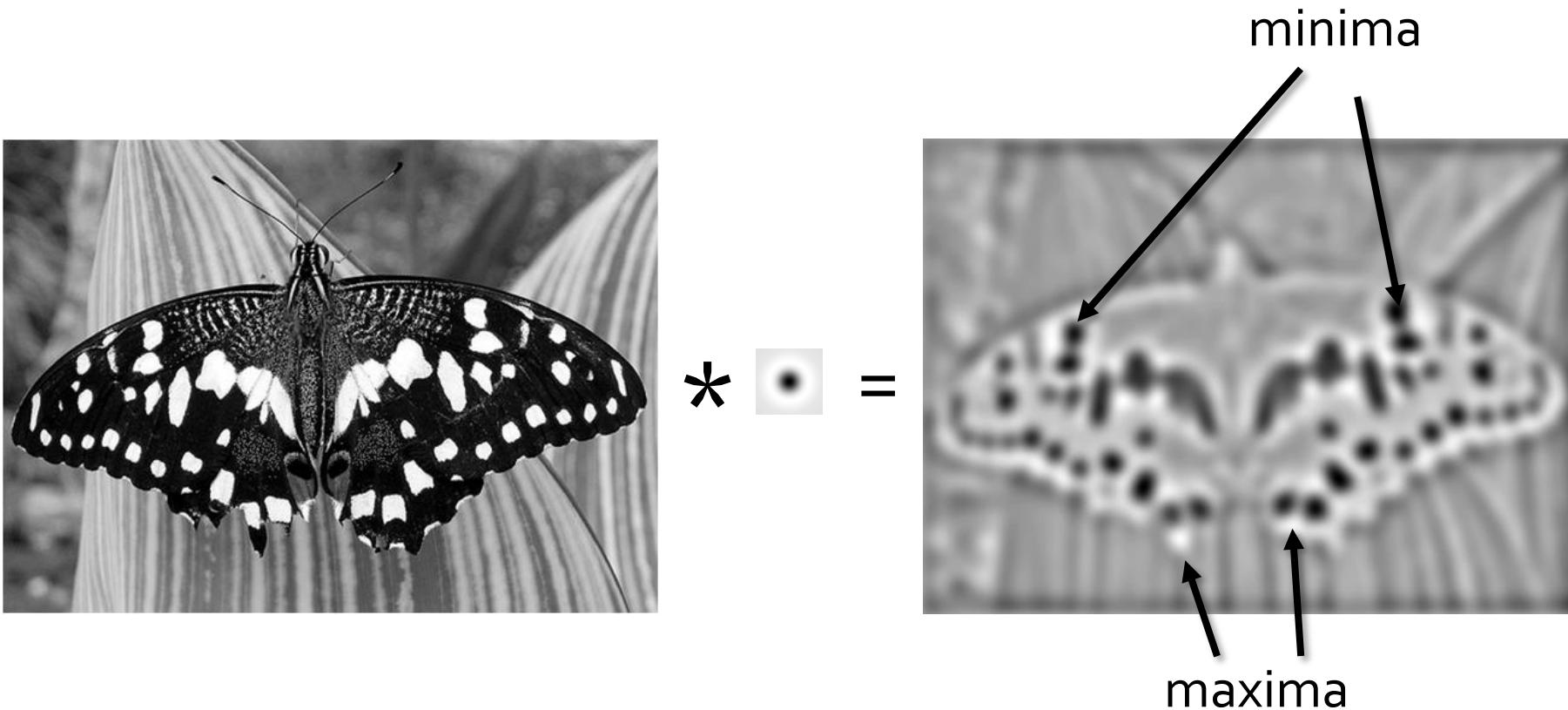
- Convolve the image with a “blob filter” at multiple scales and look for extrema of filter response in the resulting scale space



T. Lindeberg, [Feature detection with automatic scale selection](#),  
*IJCV* 30(2), pp 77-116, 1998

# Blob detection

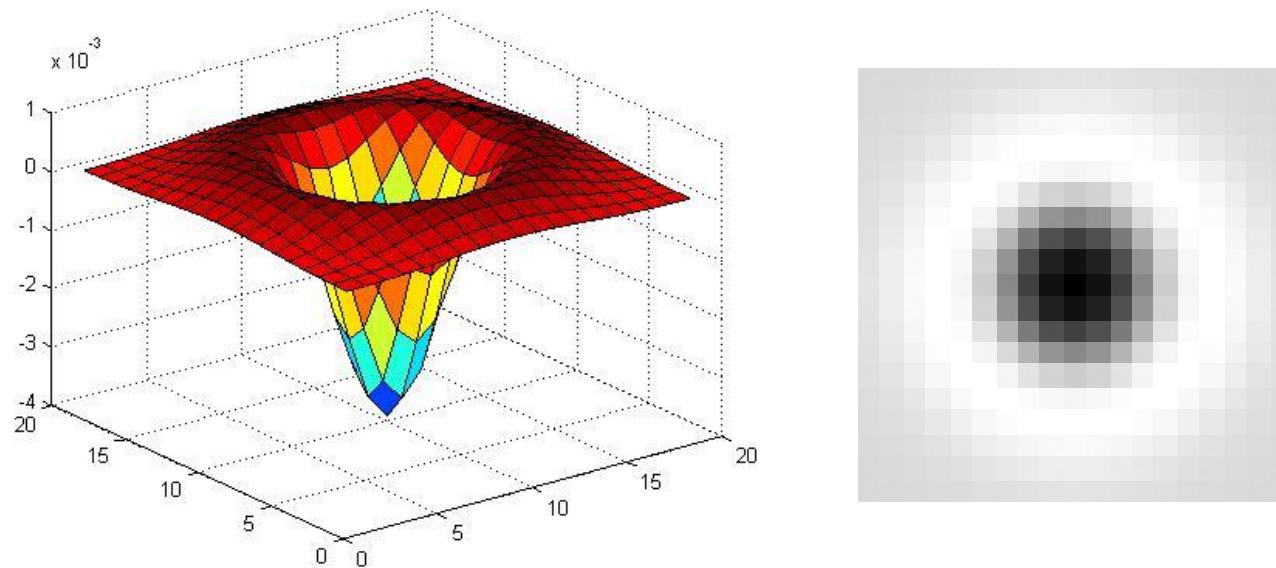
- Find *maxima* and *minima* of blob filter response in space and scale



# Blob filter

---

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



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

# Laplacian of Gaussian (LoG)

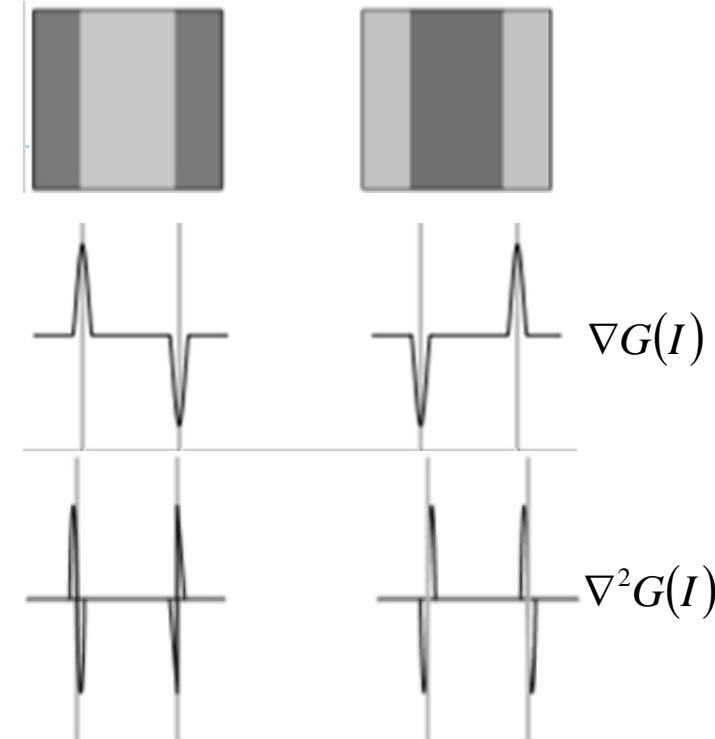
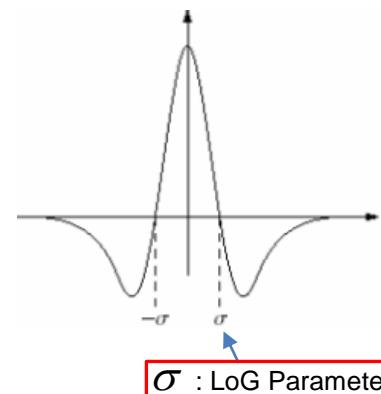
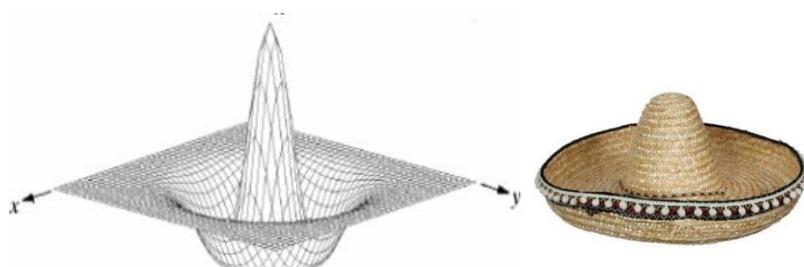
$$\nabla^2 I = \frac{\partial^2 I(x, y)}{\partial x^2} + \frac{\partial^2 I(x, y)}{\partial y^2} \quad \text{Laplacian} \quad I \xrightarrow{\text{G}} \xrightarrow{\text{L}}$$

$$\nabla^2 * (G * I) = (\nabla^2 * G) * I \quad \text{LoG} \quad I \xrightarrow{\text{LoG}}$$

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

$$\begin{aligned} \nabla^2 G &= \left( \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2} \right) \\ &= \left[ \frac{x^2 + y^2 - 2\sigma^2}{2\pi\sigma^6} \right] \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \end{aligned}$$

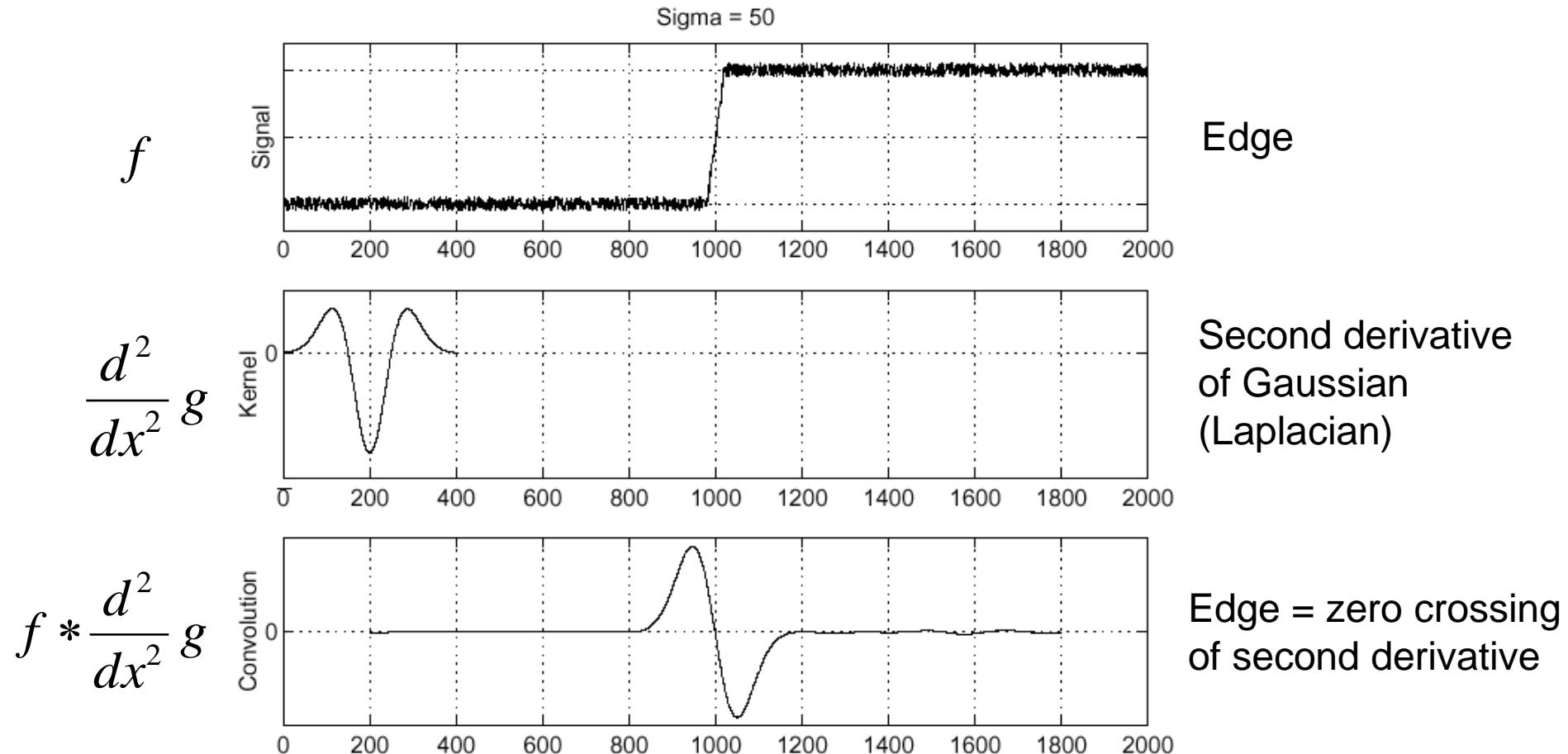
LoG function



0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

$\sigma$  : LoG Parameter

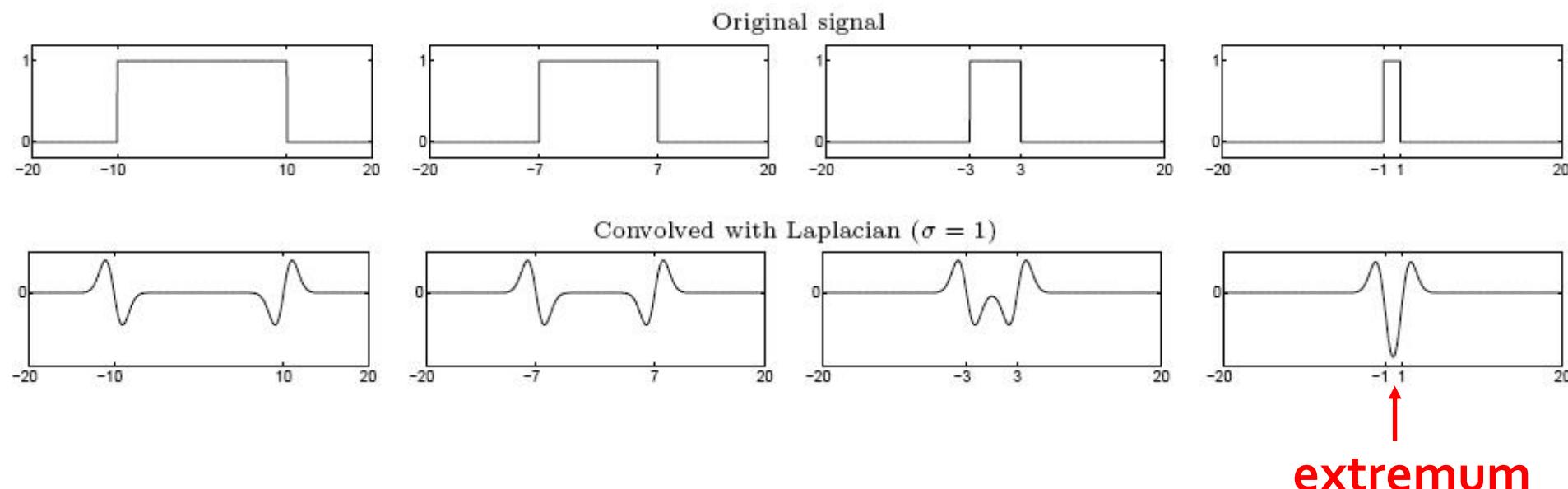
# Recall: Edge detection- Laplacian



# Laplacian of Gaussian (LoG)

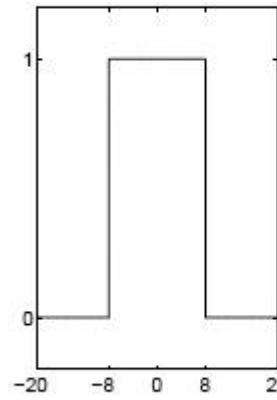
- LoG applied on Edge
  - Ripple
- LoG applied on Pulse (Blob):
  - Superposition of two ripples, generating an extremal value at the center of the pulse (blob)

Convolving with a pulse signal, 1-D LoG generates an extremal pulse when its sigma matches the width of the pulse.

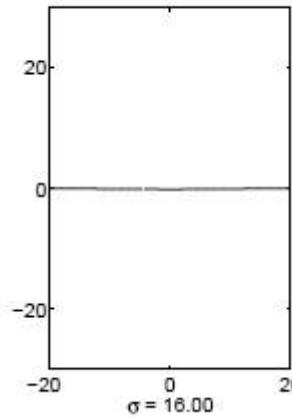
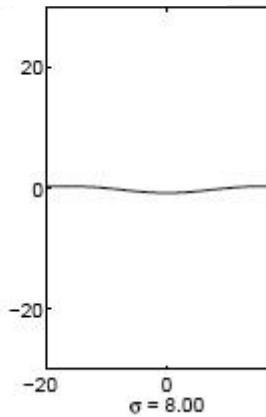
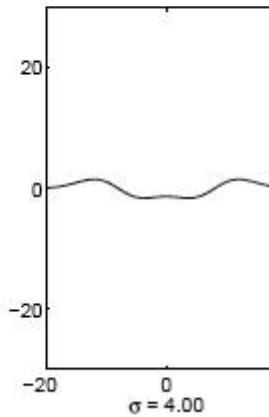
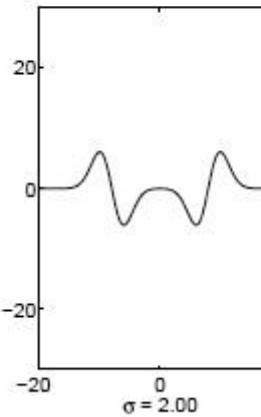
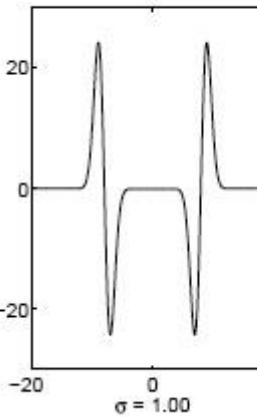


# Scale-normalized LoG

Original signal

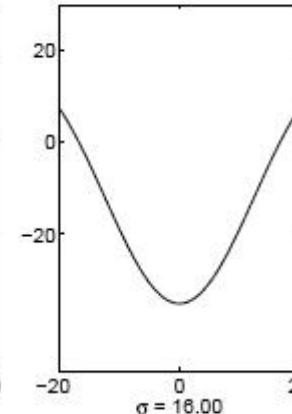
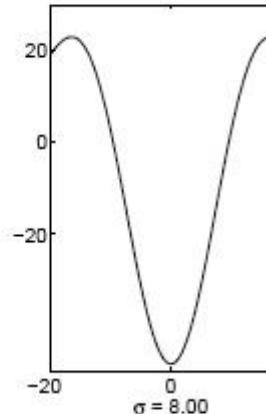
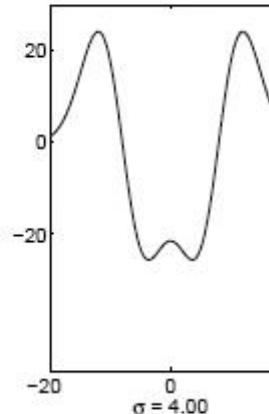
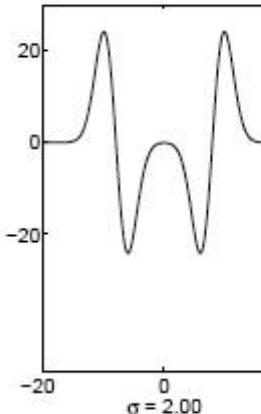
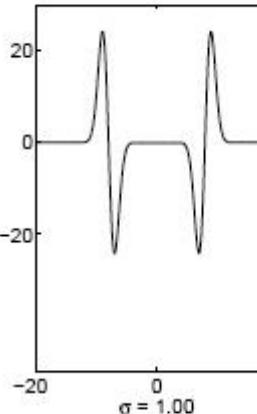


Unnormalized Laplacian response



Need to multiply  $\sigma^2$  to LoG for the scale normalization

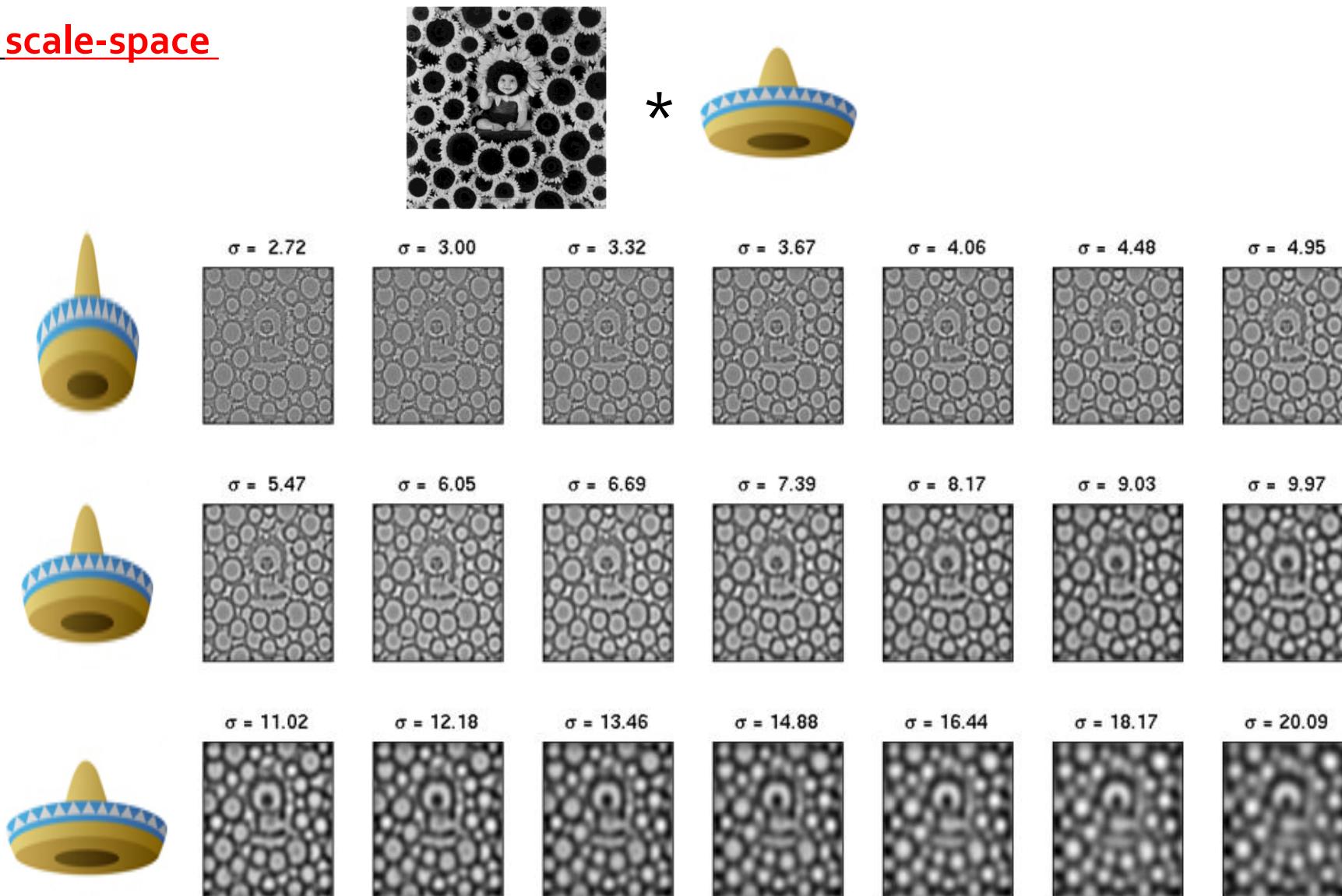
Scale-normalized Laplacian response



extremum

# Blob Detection with 2-D LoG

LoG filtered scale-space

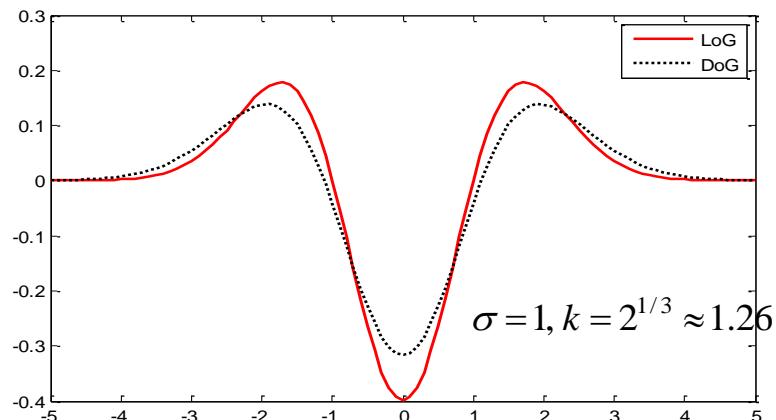


Source: <http://www.cs.utah.edu/~jfishbau/advimproc/project1/>

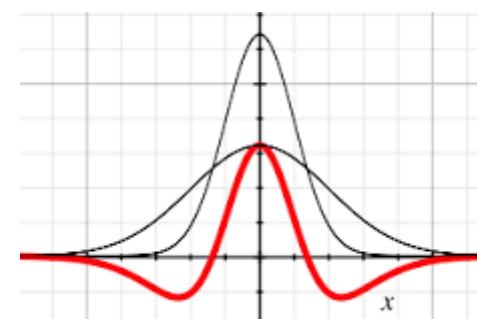
# DoG (Difference of Gaussian)

Normalized LoG  $\approx (k-1) \times$  Difference of Gaussian (DoG)

$$LoG_{norm} = \sigma^2 (G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma)) \quad DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$



$$DoG(x, y, \sigma) \approx (k - 1)LoG_{norm}(x, y, \sigma)$$



- Use DoG instead of LoG
  - 1) Gaussian Filtering :  $G(x, y, \sigma), G(x, y, k\sigma)$
  - 2) Subtraction

$$\begin{aligned} D(x, y, \sigma) &= I * G(x, y, k\sigma) - I * G(x, y, \sigma) \\ &= I * (G(x, y, k\sigma) - G(x, y, \sigma)) = I * DoG(x, y, k\sigma, \sigma) \end{aligned}$$

# Scale-space blob detector: Example

---



# Scale-space blob detector: Example

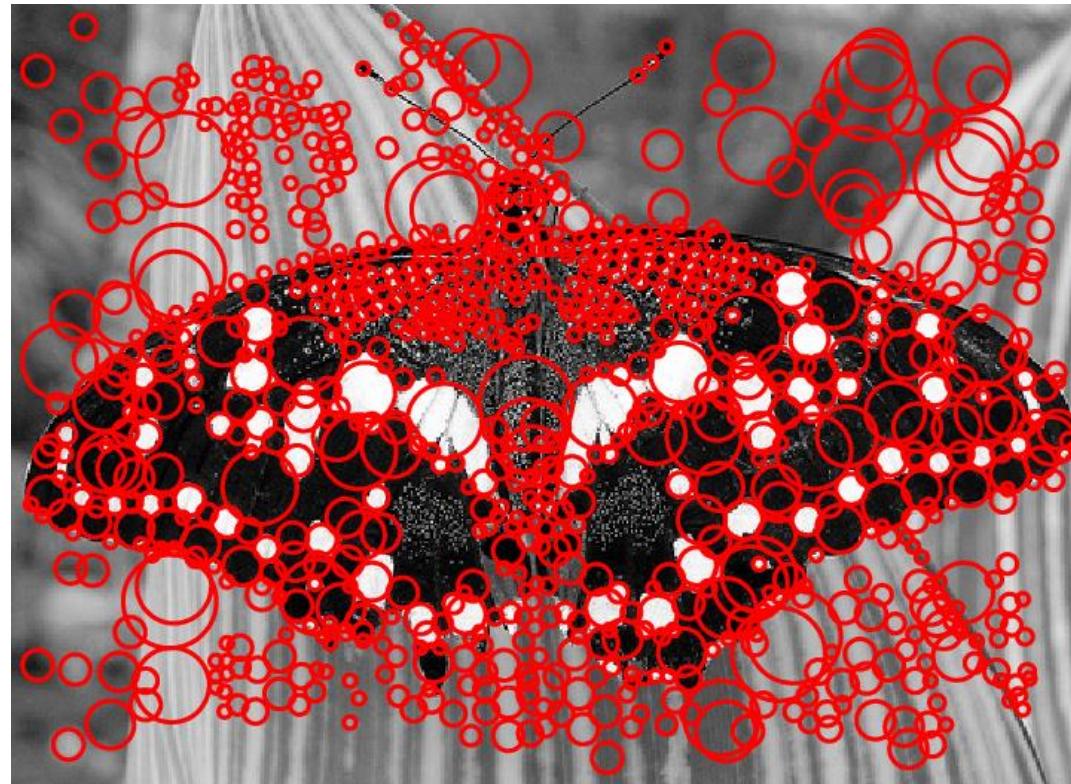
---



$\sigma = 11.9912$

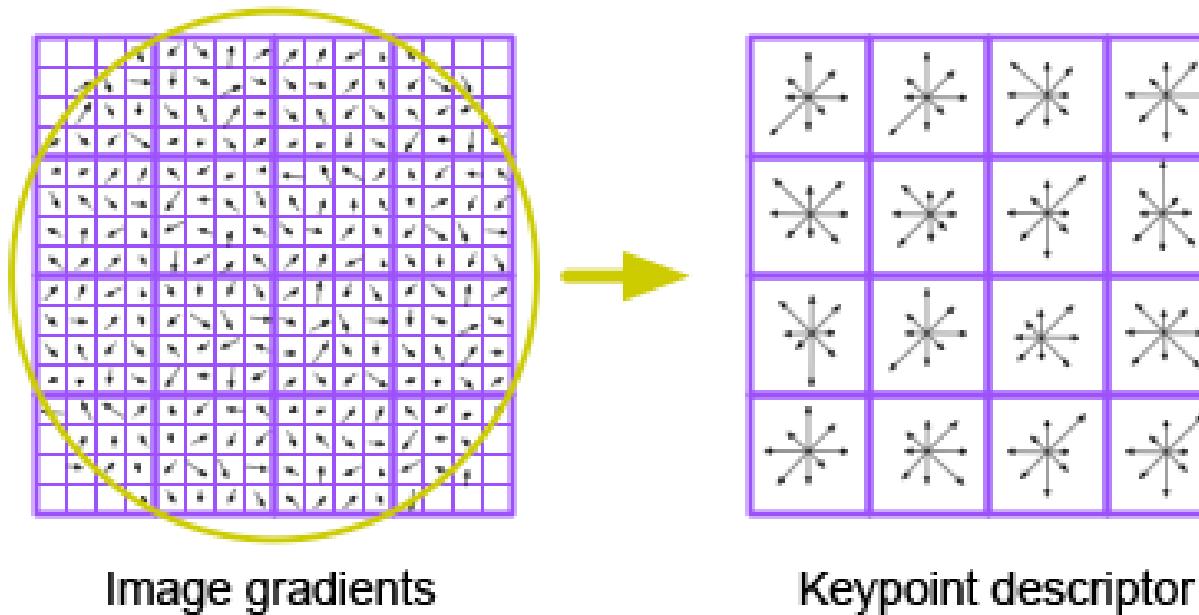
# Scale-space blob detector: Example

---



# SIFT descriptors

- Inspiration: complex neurons in the primary visual cortex

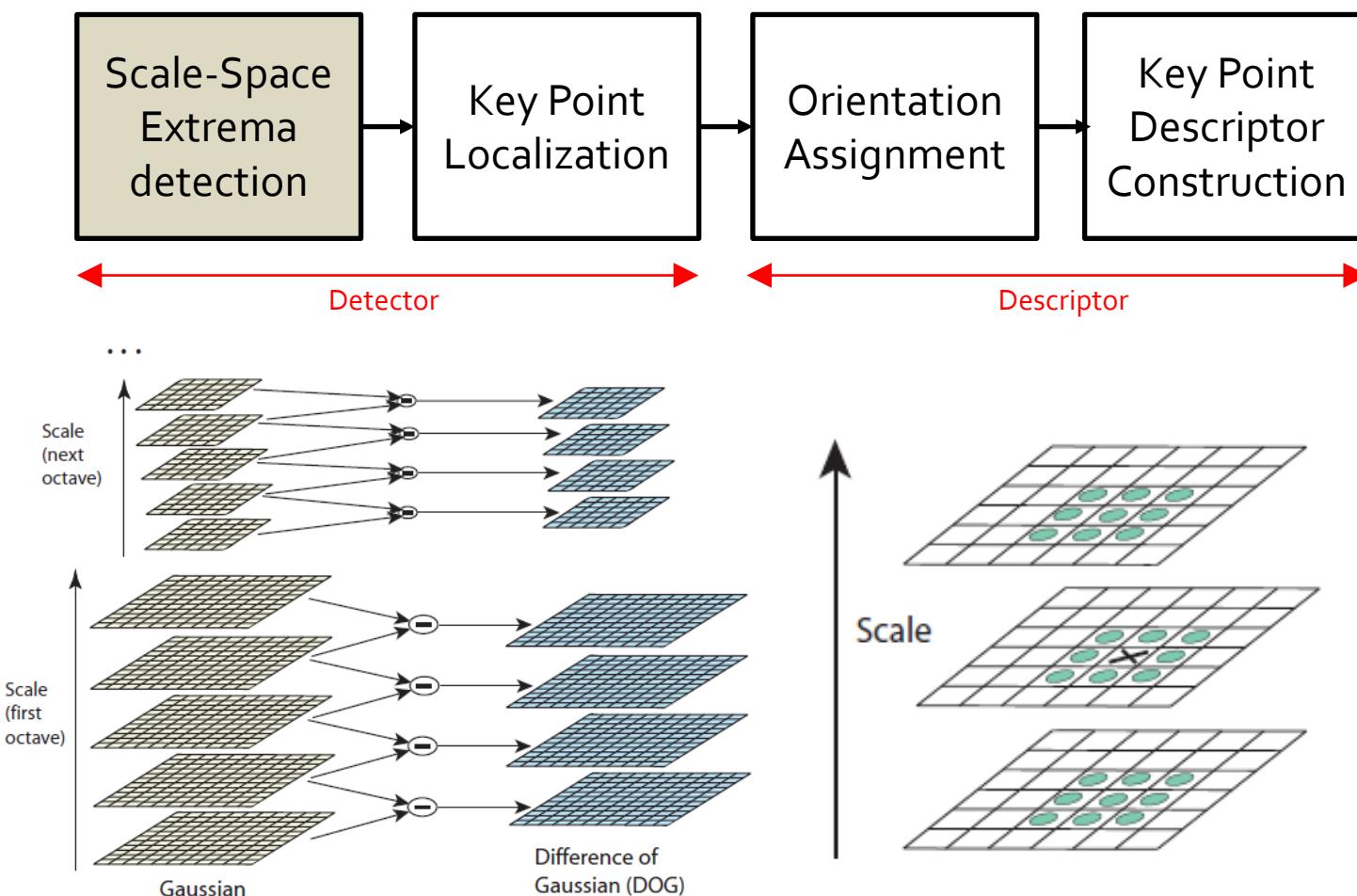


D. Lowe, [Distinctive image features from scale-invariant keypoints](#),  
IJCV 60 (2), pp. 91-110, 2004

# SIFT: Step 1) Scale Space Extrema Detection

## Scale-Space Extrema Detection

Detect the candidates of interest points, which are extrema points in the scale-space domains.



Source: D. Lowe

# SIFT: Step 1) Scale Space Extrema Detection

Ex) 2 Levels ( $S=2$ )

2 Octave

- Need to generate  $S+3=2+3=5$

blurred images per octave

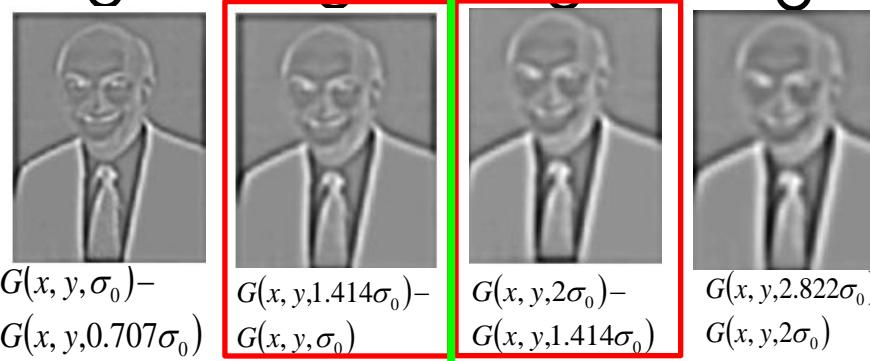
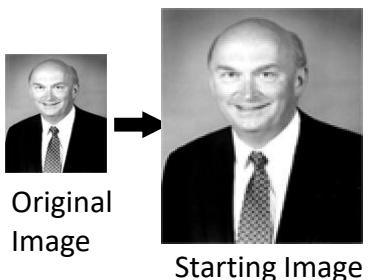
$$k = 2^{1/S} = 2^{1/2} = 1.414$$

$$\sigma_0 k^{-1} = 0.707\sigma_0 \quad \sigma_0 \quad k\sigma_0 = 1.414\sigma_0 \quad k^2\sigma_0 = 2\sigma_0 \quad k^3\sigma_0 = 2.828\sigma_0$$



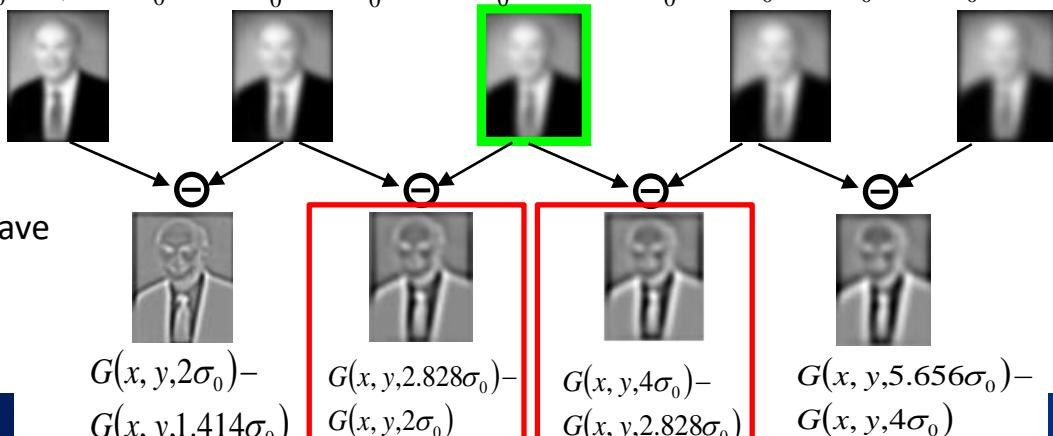
First Octave

FirstOctave : -1



Second Octave

$$k\sigma_0 = 1.414\sigma_0 \quad k^2\sigma_0 = 2\sigma_0 \quad k^3\sigma_0 = 2.828\sigma_0 \quad k^4\sigma_0 = 4\sigma_0 \quad k^5\sigma_0 = 5.656\sigma_0$$



# SIFT: Step 1) Scale Space Extrema Detection

## Scale-Space Extrema Detection

- Compare a pixel with 26 pixels in current and adjacent scales (Green circles)
- Select a pixel as an extrema if larger/smaller than neighboring 26 pixels
- Needs further localization and sampling



Source: D. Lowe

# SIFT: Step 2) Key Point Localization

- (1) Sub-pixel localization and removal of extrema points with low contrast:

Use Taylor series expansion of the scale-space function D to find maximum of surface

$$D(X) = D + \frac{\partial D^T}{\partial X} X + \frac{1}{2} X^T \frac{\partial^2 D}{\partial X^2} X$$

D and its derivatives are evaluated at the sample point

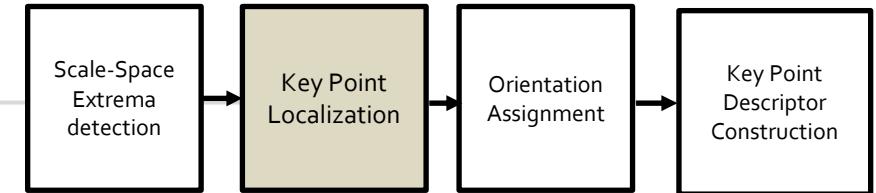
The sub-pixel location of the extremum

$$\hat{X} = - \left( \frac{\partial^2 D}{\partial X^2} \right)^{-1} \frac{\partial D}{\partial X}$$

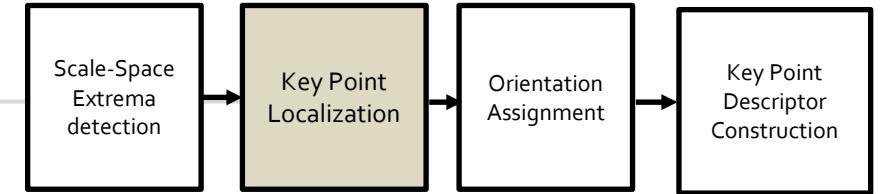
The function value at the extremum,  $D(\hat{X})$

$$D(\hat{X}) = D + \frac{1}{2} \frac{\partial D^T}{\partial X} \hat{X}$$

If  $|D(\hat{X})| < Th$ , then discard the extremum.



# SIFT: Step 2) Key Point Localization



(2) Delete edge-like features by calculating the curvature

H: Hessian matrix

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \quad r = \frac{\text{Trace}(H)^2}{|H|} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \lambda_2} = \frac{\left(1 + \frac{\lambda_2}{\lambda_1}\right)^2}{\frac{\lambda_2}{\lambda_1}}$$

If  $r > Th$ , then delete the extreme point.

# SIFT: Step 3) Orientation Assignment

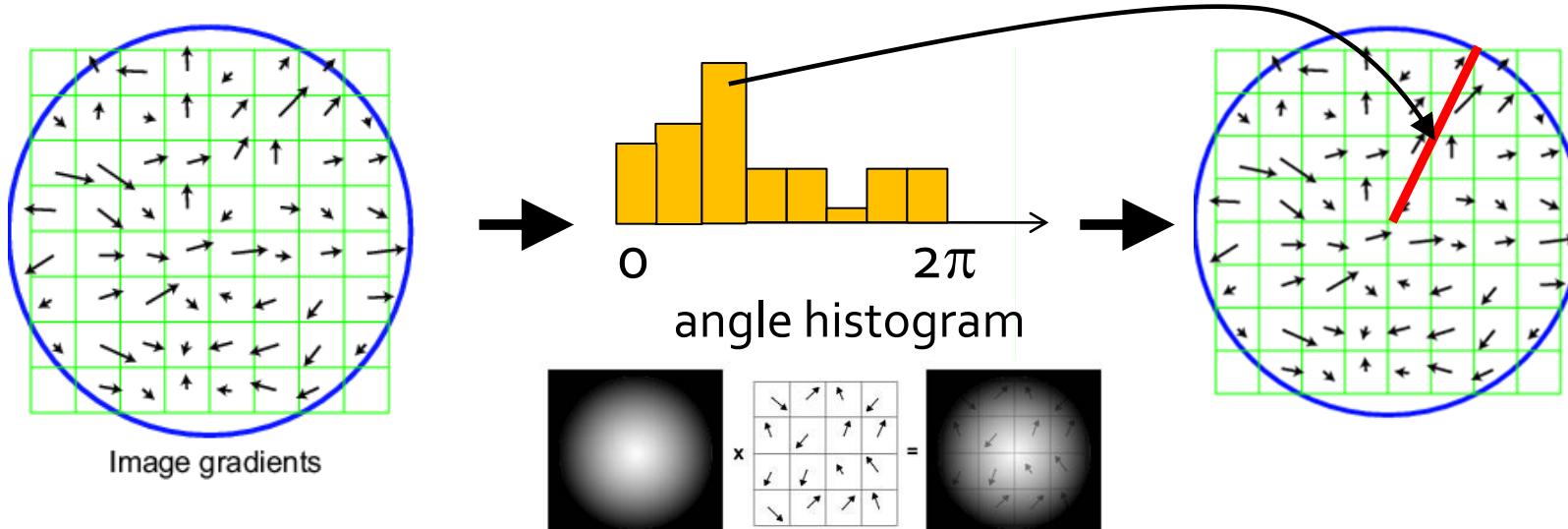
1. Take  $16 \times 16$  square window

2. Compute edge orientation for each  $2 \times 2$  block in  $16 \times 16$  square

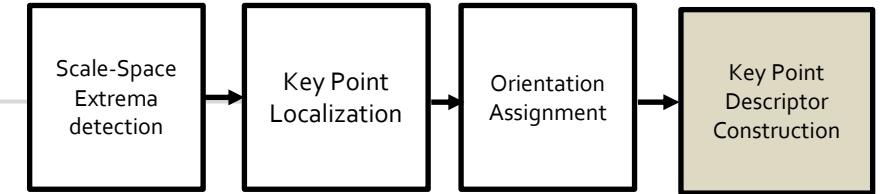
$$L(x, y) = G(x, y, \sigma)^* I(x, y) \quad m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$
$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1)) / (L(x+1, y) - L(x-1, y)))$$

3. Throw out weak edges (threshold gradient magnitude)

4. Create histogram by accumulating the Gaussian weighted edge magnitude

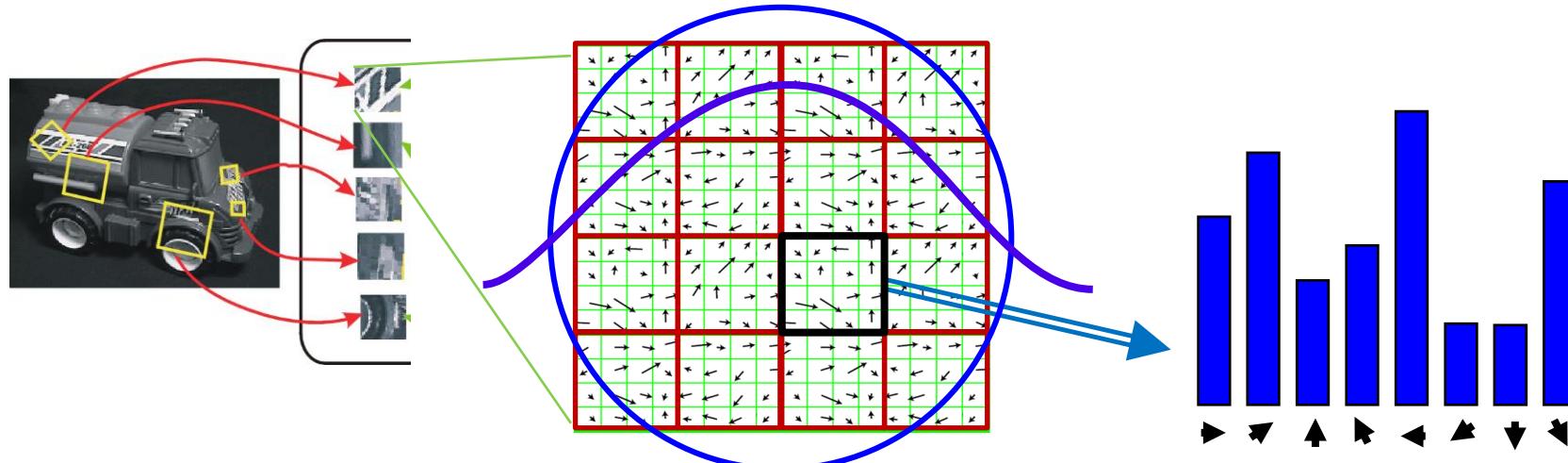


# SIFT: Step 4) Descriptor Construction



1. Normalize the window as **16x16** window using orientation/scale.
2. For each **4x4** block, compute gradient histogram over **8** directions.  
Gradient magnitudes are weighted by a Gaussian of variance half the window (for smooth fall-off)

SIFT descriptor: **128-D** vector

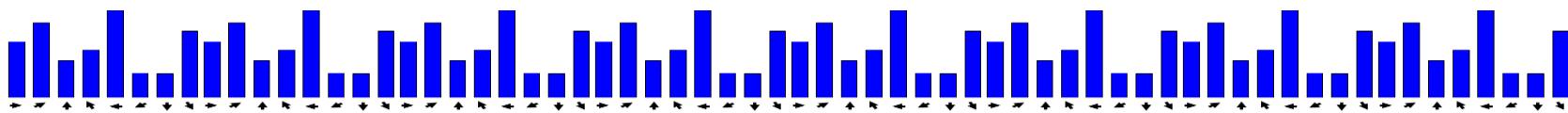


# SIFT: Step 4) Descriptor Construction

---

3. Concatenate 8-D vectors of  $4 \times 4$  arrays and normalise the magnitude 128-D vector  $t$

o 1.



SIFT Descriptor: Binning of Spatial Coordinates and Gradient Orientations

4. **Threshold** gradient magnitudes to avoid excessive influence of high gradients

1. after normalization, clamp gradients  $> 0.2$
2. Renormalize the vector

# Properties of SIFT

---

- **Extraordinarily robust detection and description 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
  - Fast and efficient—can run in real time



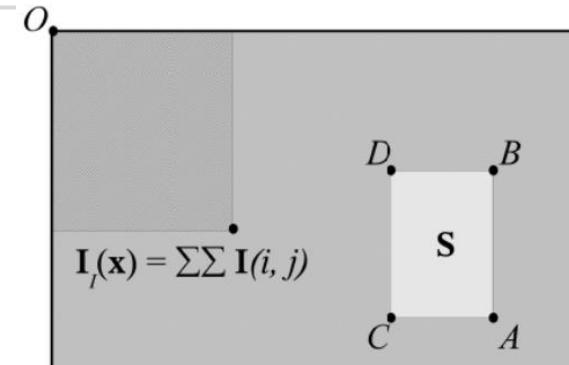
Source: N. Snavely

# SURF (Speeded Up Robust Features)

- SIFT is one of the best but slow
- Using *integral images* for an efficient implementation
- Detect and describe SIFT like features
- SURF describes image 3 times faster than SIFT
- SURF is not as well as SIFT on invariancae to illumination change and viewpoint change
- Keypoint detection based on Hessian matrix: blob-like features

$$H(p, \sigma) = \begin{bmatrix} L_{xx}(p, \sigma) & L_{xy}(p, \sigma) \\ L_{xy}(p, \sigma) & L_{yy}(p, \sigma) \end{bmatrix} \quad |H(p, \sigma)| = L_{xx}(p, \sigma)L_{yy}(p, \sigma) - L_{xy}(p, \sigma)^2$$

where  $L_{xx}(p, \sigma)$  is the convolution of the second order Gaussian derivative  $\partial^2 G(p, \sigma)/\partial x^2$  with the image  $I$  in  $p$  and similarly for  $L_{xy}(p, \sigma)$  and  $L_{yy}(p, \sigma)$ .



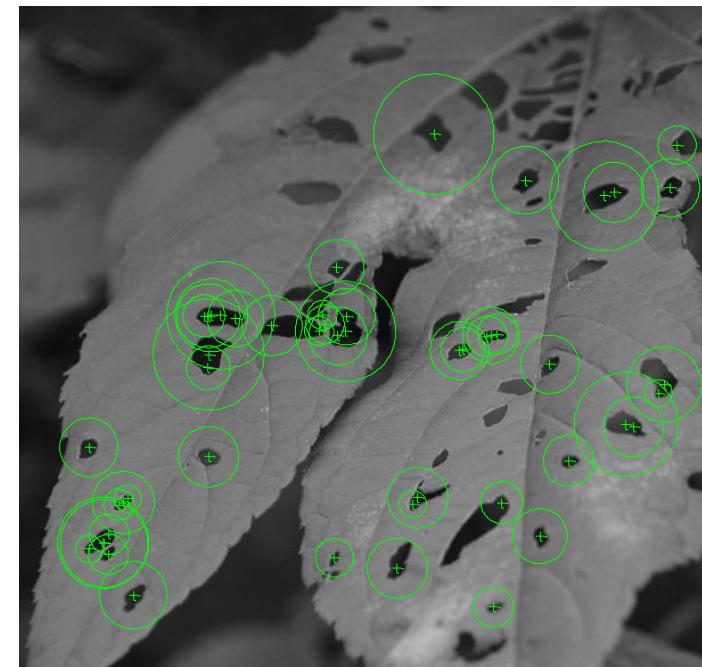
$$S = (A - B - C + D)$$

**Integral images:** accumulated sum of gray scale pixel values of images

# SURF

---

- SURF is more than three times faster than SIFT
- SURF is inferior to SIFT for luminance and viewpoint changes.
- SURF integrates the gradient information within a subpatch, whereas SIFT depends on the orientations of the individual gradients. This makes SURF *less sensitive to noise*.



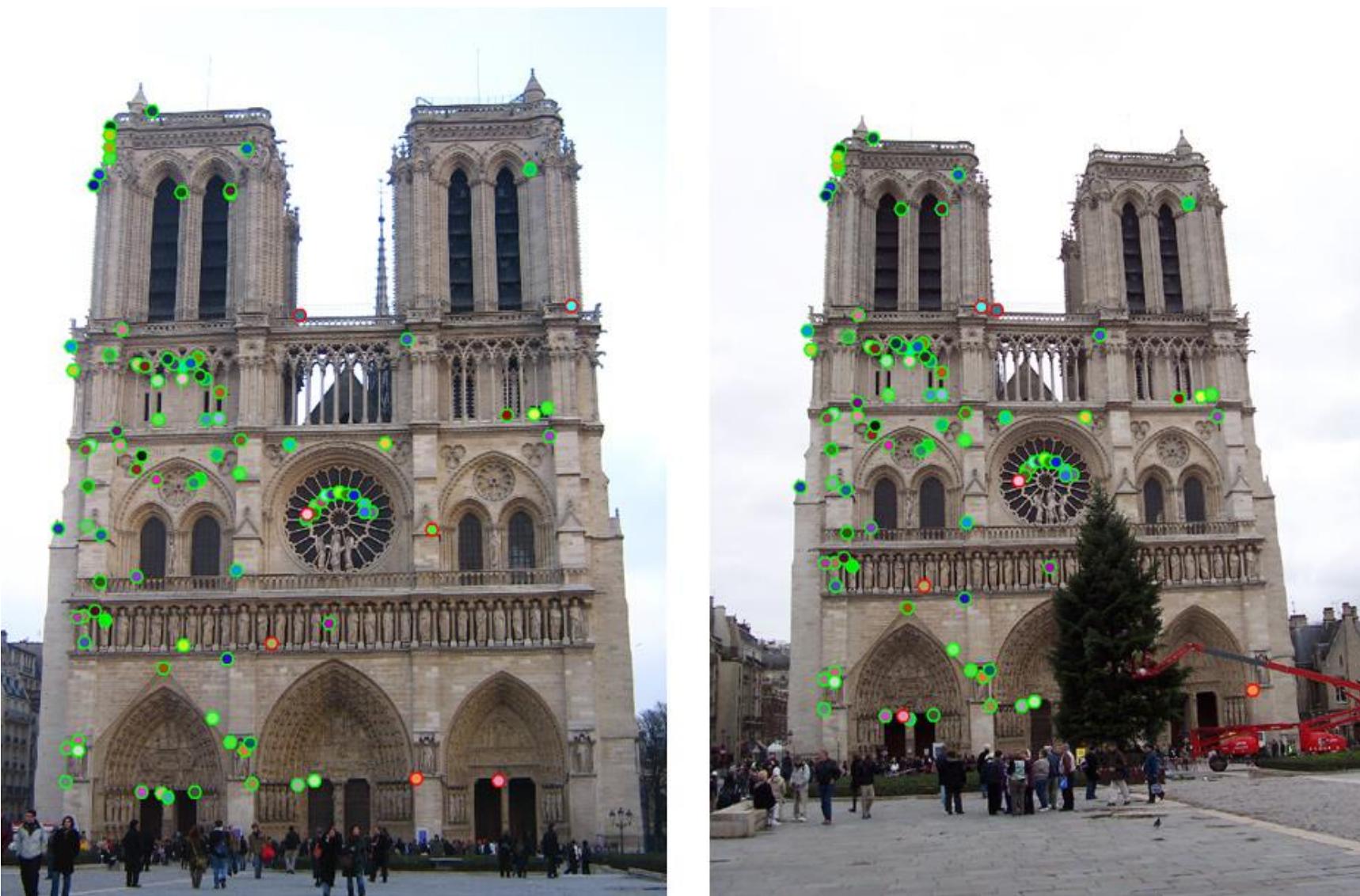
# Contents

---

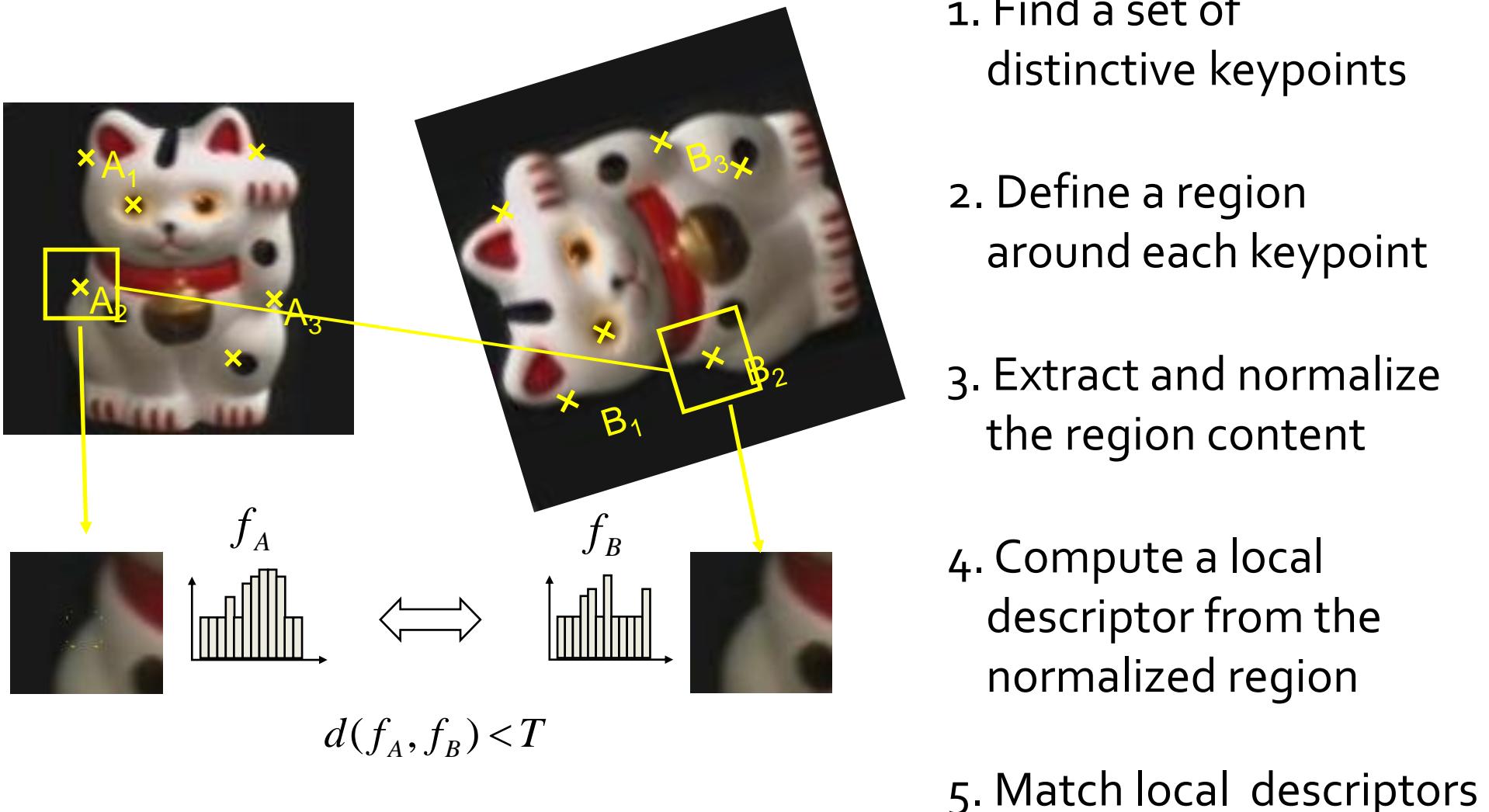
- **Feature Descriptor**
  - Scale Invariant Feature Transform (SIFT) descriptor
  - Speeded Up Robust Features (SURF) descriptor
- **Feature Matching**
  - Nearest Neighbor (NN) Matching

# How do we decide which features match?

---



# Overview of Feature Matching



# Feature Matching

---

- **Nearest neighbor matching**
  - One feature matches to another if those features are nearest neighbors and their distance is below some threshold.
$$\{f_i | i = 1, \dots, N\} \text{ for } I_1 \text{ and } \{g_j | j = 1, \dots, M\} \text{ for } I_2$$
$$k = \min_j dist(f_i, g_j) \quad \& \quad dist(f_i, g_k) < T \rightarrow NN(f_i) = g_k$$
- **Problems**
  - Threshold  $T$  is difficult to set
  - Non-distinctive features could have lots of close matches, only one of which is correct

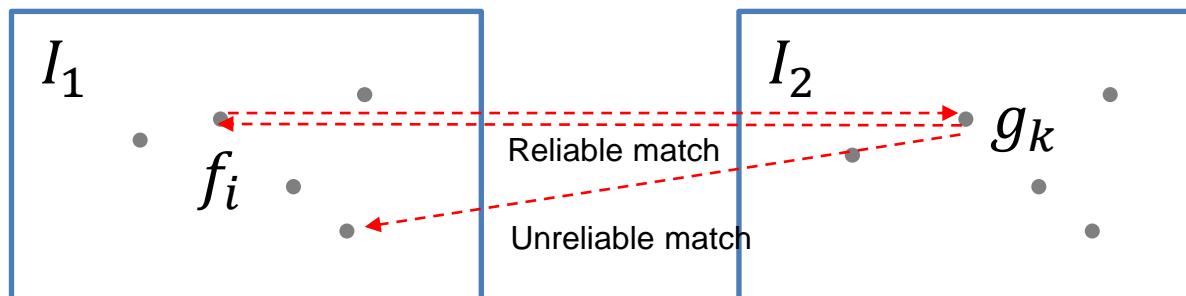
# Feature Matching

- Simple Solution: Cross-checking technique

$$k = \min_j dist(f_i, g_j) \rightarrow NN(f_i) = g_k$$

$$l = \min_i dist(f_i, g_k) \rightarrow NN(g_k) = f_l$$

If  $i = l$ , the matching is assumed to be reliable.  
Otherwise, the matching is unreliable.



# Feature Matching

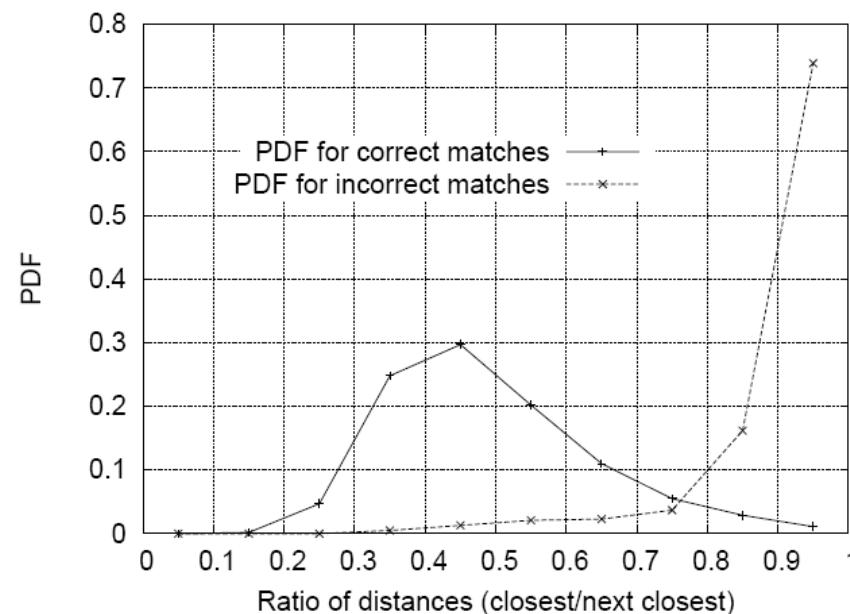
- Simple Solution
  - Refine matched points using threshold ratio of nearest to 2<sup>nd</sup> nearest descriptor

$$k_1 = \min_j dist(f_i, g_j)$$

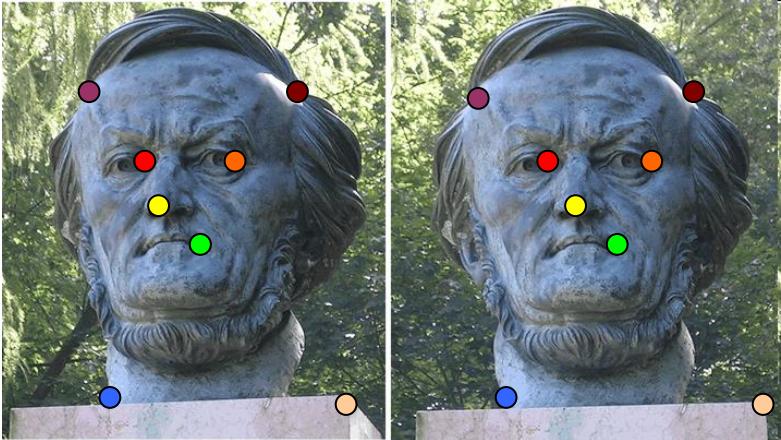
$$k_2 = \text{second } \min_j dist(f_i, g_j)$$

$$\frac{dist(f_i, g_{k_1})}{dist(f_i, g_{k_2})) < T_r} \rightarrow NN(f_i) = g_{k_1}$$

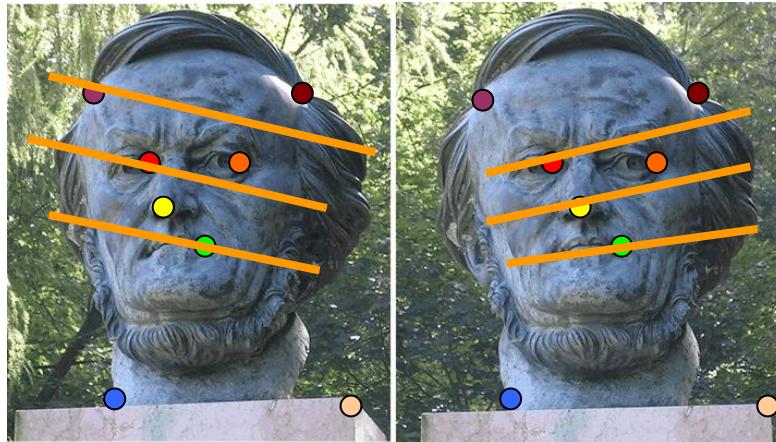
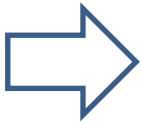
This gets rid of 90% false matches,  
5% of true matches in Lowe's study



# Feature Matching and Fitting



Feature matching



Fundamental  
matrix estimation



Feature matching

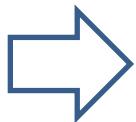


Image stitching using  
geometric transform  
estimation

# Next topic

---

- We've learned how to process pixels and detect features, e.g. edges, corners, blobs.

Now what?

- A higher-level, more compact representation of the features in the image by grouping multiple features according to a simple model
  - Prerequisite
    - Review EBU6230 Image/Video Processing – Week3: Interest points
    - Review EBU7240 Computer Vision – Week1: feature (what you learned today)