

**Spring 2024**

# INTRODUCTION TO COMPUTER VISION

---

**Atlas Wang**

Associate Professor, The University of Texas at Austin

**Visual Informatics Group@UT Austin**  
<https://vita-group.github.io/>

# Famous tale in computer vision

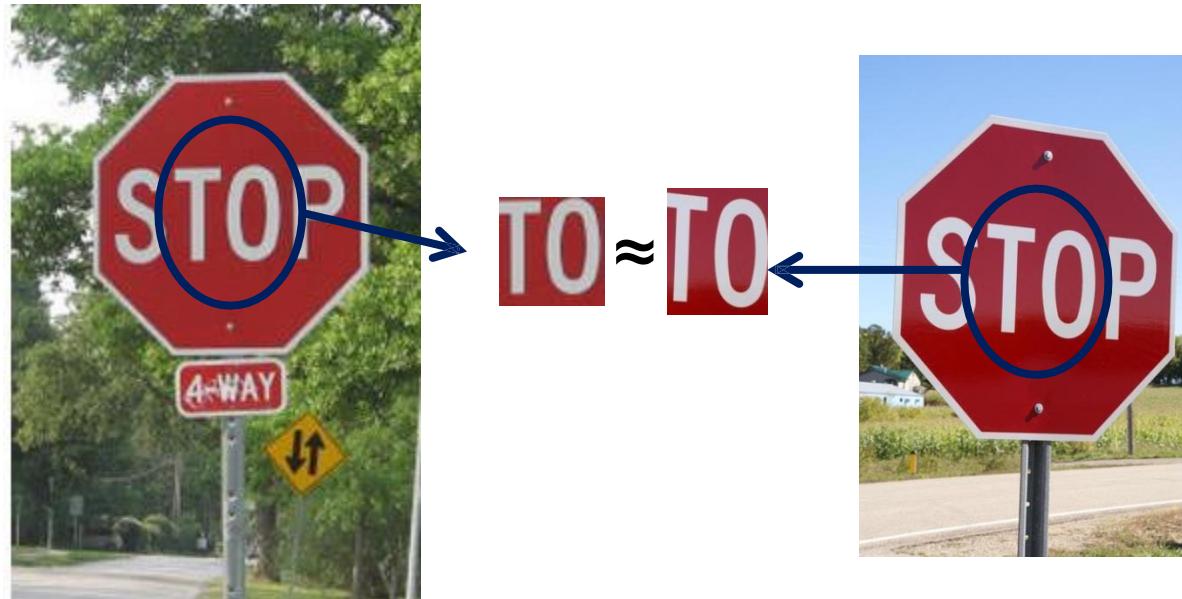
- Once, a CMU graduate student asked the famous computer vision scientist **Takeo Kanade**: *"What are the three most important problems in computer vision?"*
- Takeo replied: "**Correspondence, correspondence, correspondence!**"



# Visual Correspondence across views

Matching points, patches, edges, or regions across images.

- *Sparse or local correspondence* (picking some “keypoints”)
- *Dense correspondence* (at every pixel)



# Fundamental to Applications

- Image alignment
- 3D reconstruction
- Motion tracking (robots, drones, AR)
- Indexing and database retrieval
- Object recognition



Hays

# Example application: Panorama stitching

---

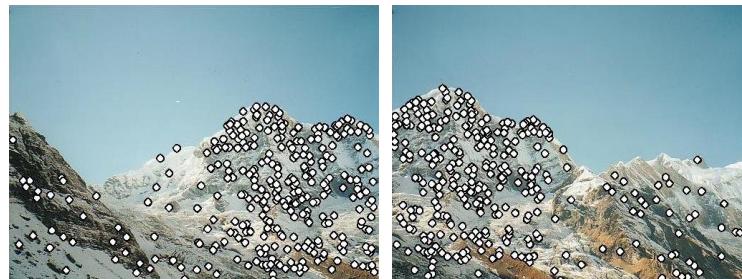
We have two images –  
how do we estimate how to overlay them?



# Local features: main components

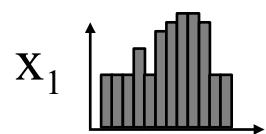
## 1) Detection:

Find a set of distinctive key points.

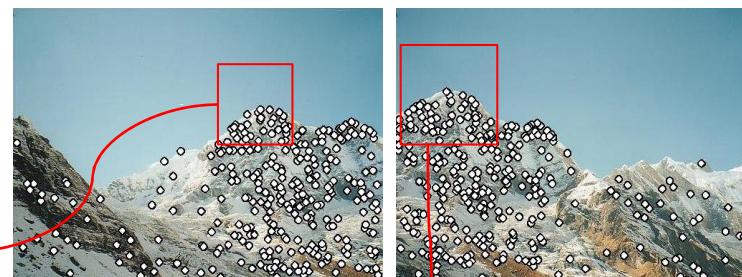


## 2) Description:

Extract feature descriptor around each interest point as vector.



$$\mathbf{x}_1 = [x_1^{(1)}, \dots, x_d^{(1)}]$$

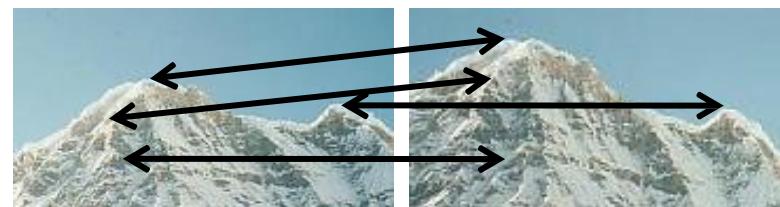


$$\mathbf{x}_2 = [x_1^{(2)}, \dots, x_d^{(2)}]$$

## 3) Matching:

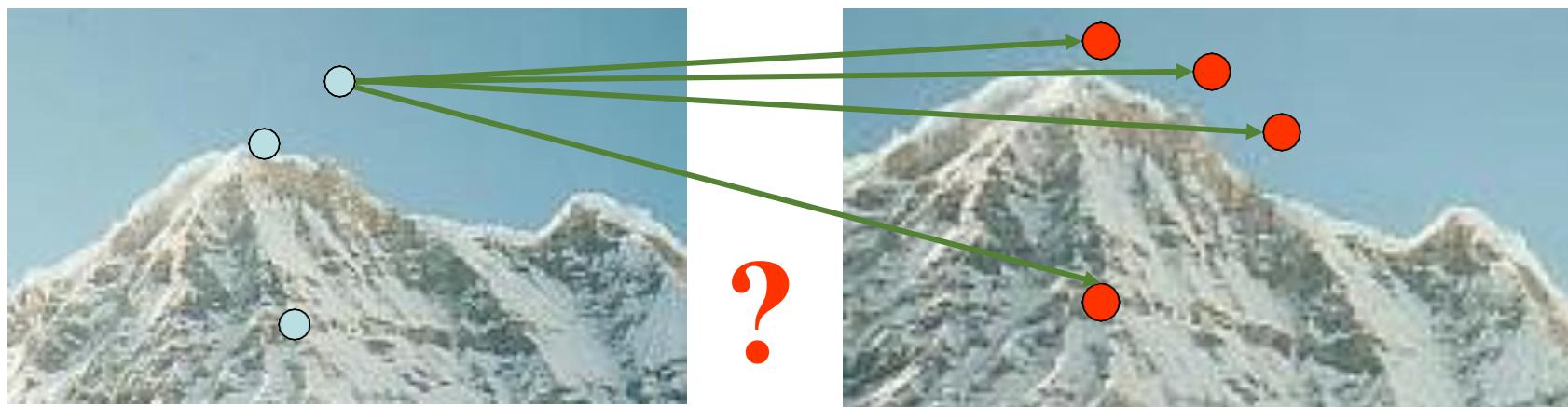
Compute distance between feature vectors to find correspondence.

$$d(\mathbf{x}_1, \mathbf{x}_2) < T$$



# Goal: Distinctiveness

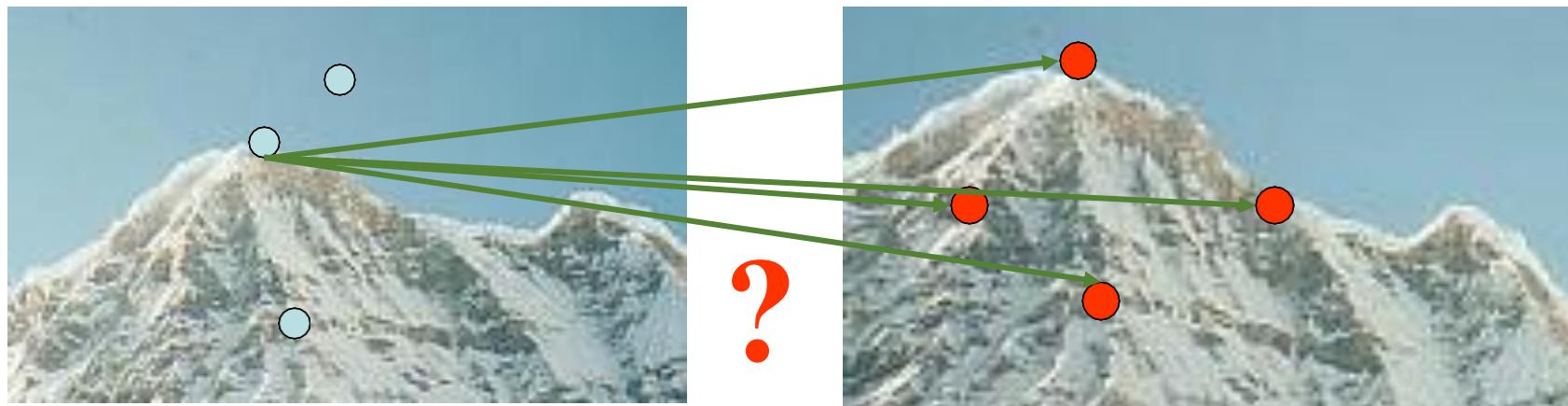
We want to be able to reliably determine which point goes with which.



May be difficult in structured environments with repeated elements!

# Goal: Distinctiveness

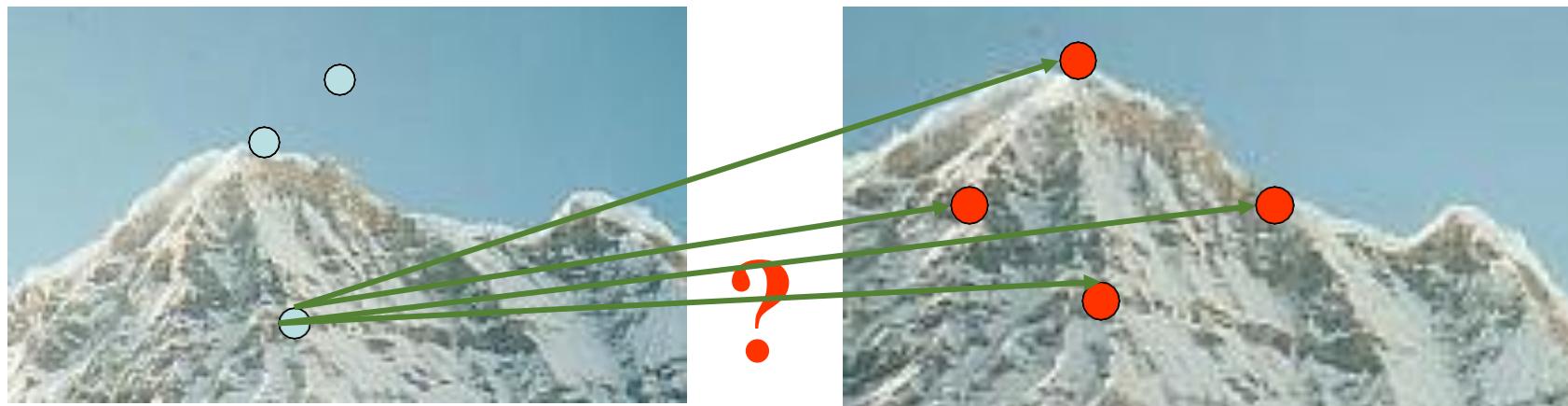
We want to be able to reliably determine which point goes with which.



May be difficult in structured environments with repeated elements!

# Goal: Distinctiveness

We want to be able to reliably determine which point goes with which.



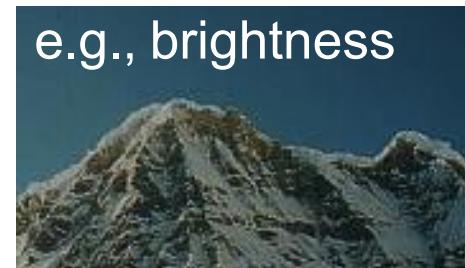
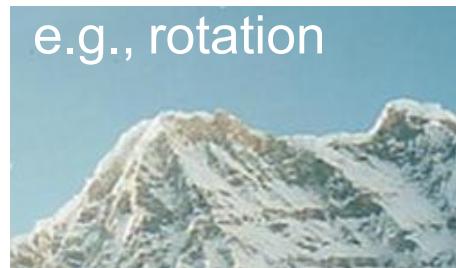
May be difficult in structured environments with repeated elements!

# Goal: Repeatability

We want to detect (at least some of)  
the same points in both images.



Under geometric and  
photometric variations.



# Goal: Compactness and Efficiency

We want the representation to be as small and as fast as possible

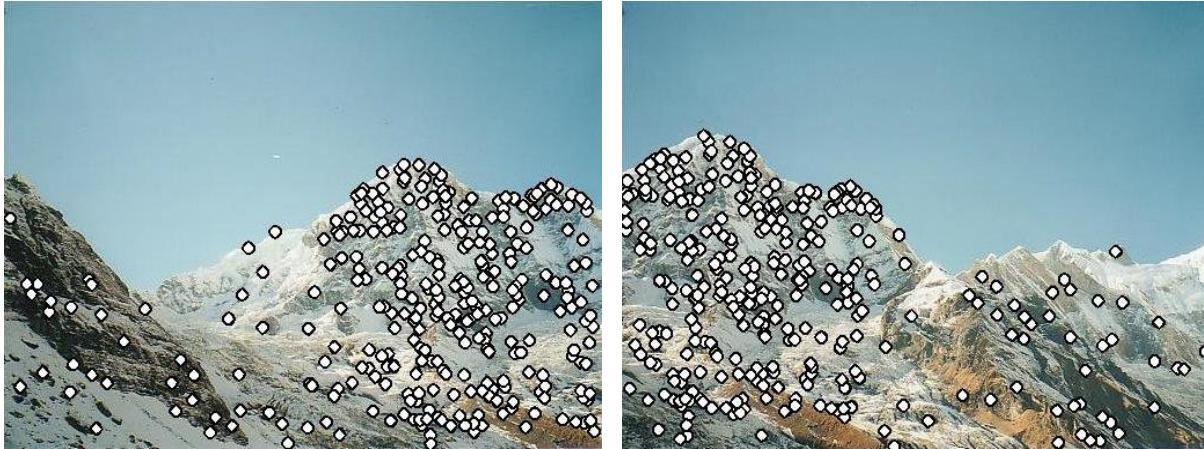
- Much smaller than a whole image

Sometimes, we'd like to run the detection procedure *independently* per image

- Match just the compact descriptors for speed.
- *Difficult!* We don't get to see 'the other image' at match time, e.g., object detection.

# Characteristics of good features

---



## Distinctiveness

Each feature can be uniquely identified

## Repeatability

The same feature can be found in several images despite differences:

- geometrically (translation, rotation, scale, perspective)
- photometrically (reflectance, illumination)

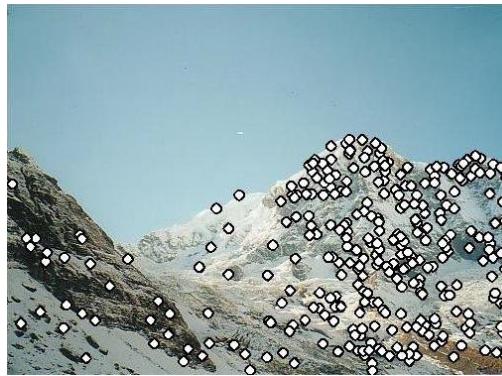
## Compactness and efficiency

Many fewer features than image pixels; run independently per image

# Local features: main components

## 1) Detection:

Find a set of distinctive key points.



## 2) Description:

Extract feature descriptor around each interest point as vector.

## 3) Matching:

Compute distance between feature vectors to find correspondence.

# Detection: Basic Idea

We do not know which other image locations the feature will end up being matched against ...

*But* can compute how stable a location is in appearance with respect to small variations in its position

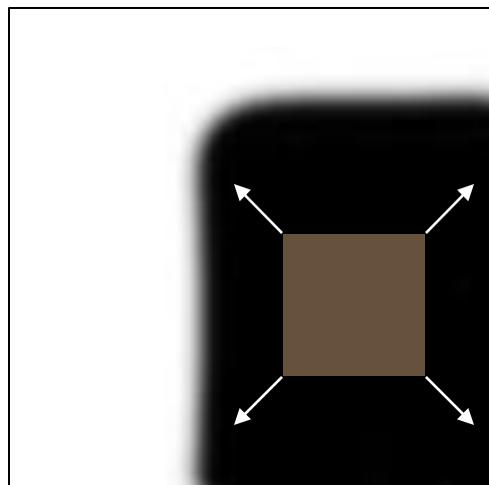
Something that “meaningfully stands out”!

*Strategy: Compare image patch against local neighbors*

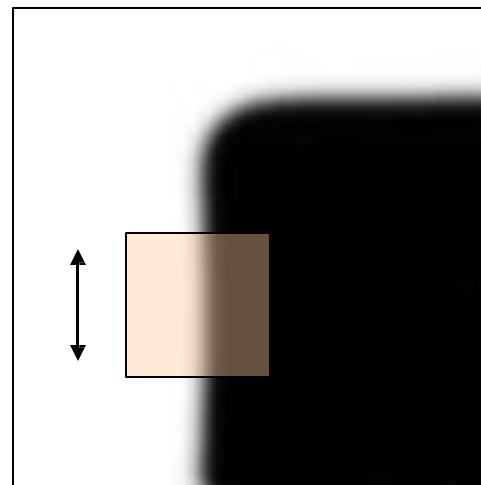
# Detection: Basic Idea

Recognize corners by looking at small window.

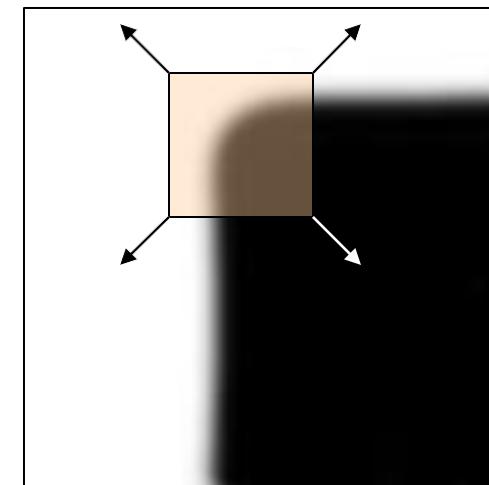
We want a window shift in *any direction* to give a *large change* in intensity.



“Flat” region:  
no change in  
all directions



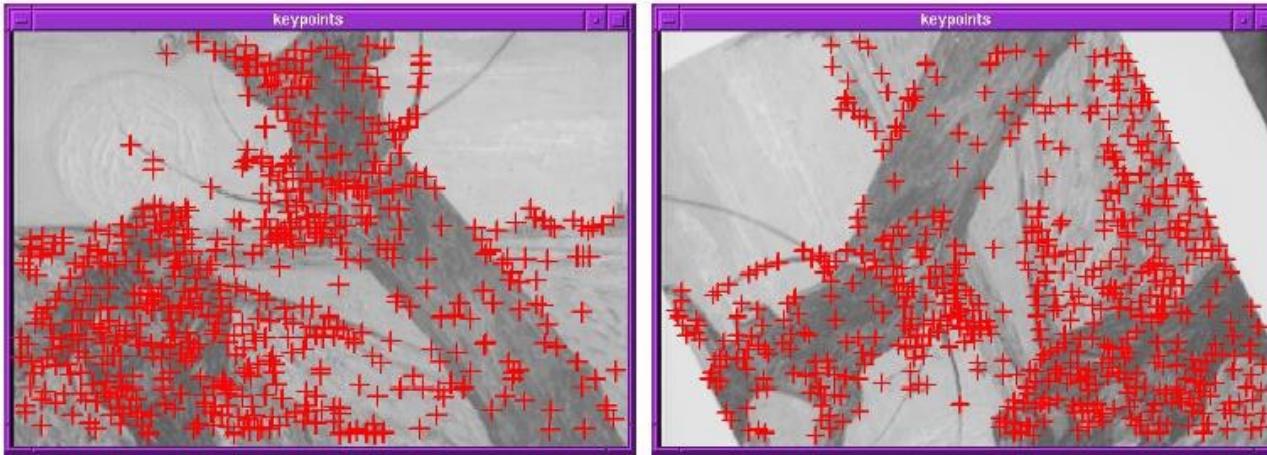
“Edge”:  
no change  
along the edge  
direction



“Corner”:  
significant  
change in all  
directions

# Finding Corners

---



- Key property: in the region around a corner, image gradient has two or more dominant directions
- Corners are repeatable and distinctive

C.Harris and M.Stephens. "[A Combined Corner and Edge Detector.](#)"  
*Proceedings of the 4th Alvey Vision Conference:* pages 147--151.

# Corner Detection by Auto-correlation

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

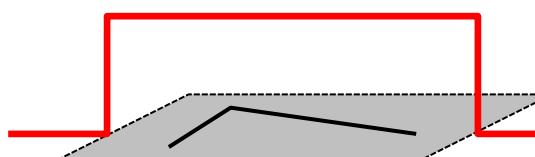
$$E(u,v) = \sum_{x,y} w(x,y) [I(x+u, y+v) - I(x, y)]$$

Diagram illustrating the components of the auto-correlation function:

- Window function:  $w(x,y)$
- Shifted intensity:  $I(x+u, y+v)$
- Intensity:  $I(x, y)$

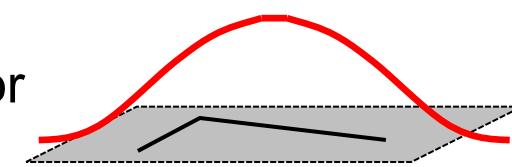
Also called ‘sum of squared differences’

Window function  $w(x,y) =$



1 in window, 0 outside

or

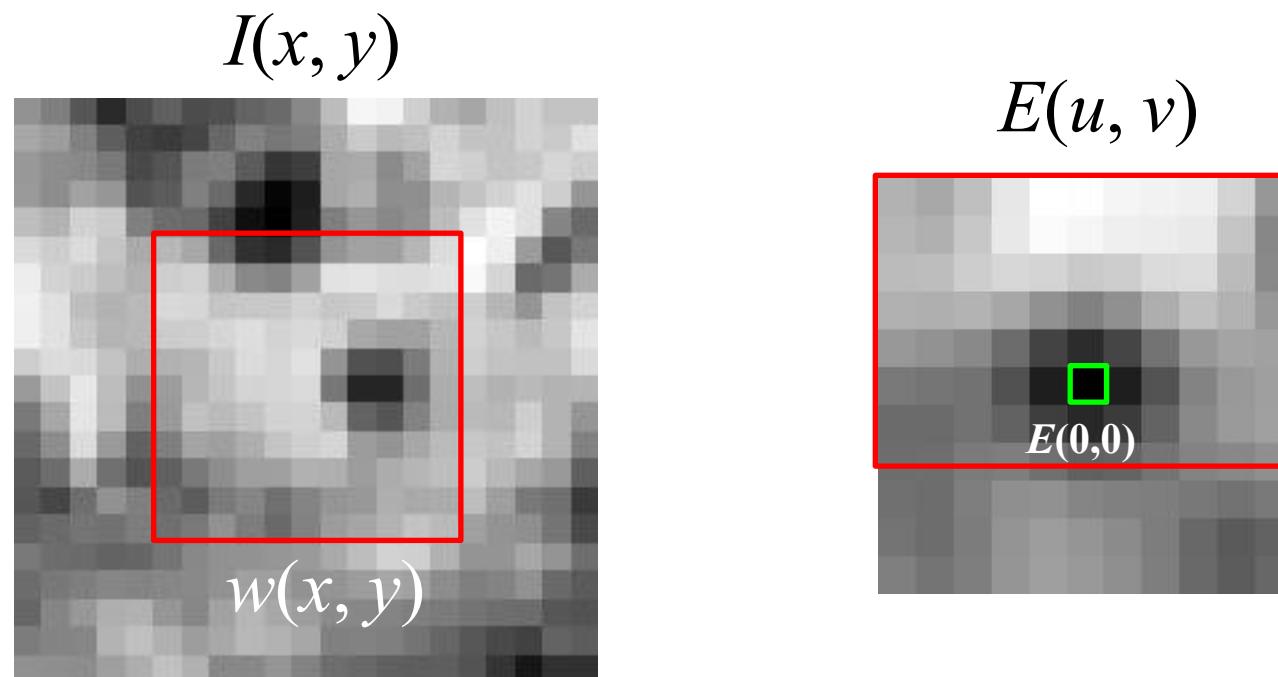


Gaussian

# Corner Detection by Auto-correlation

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

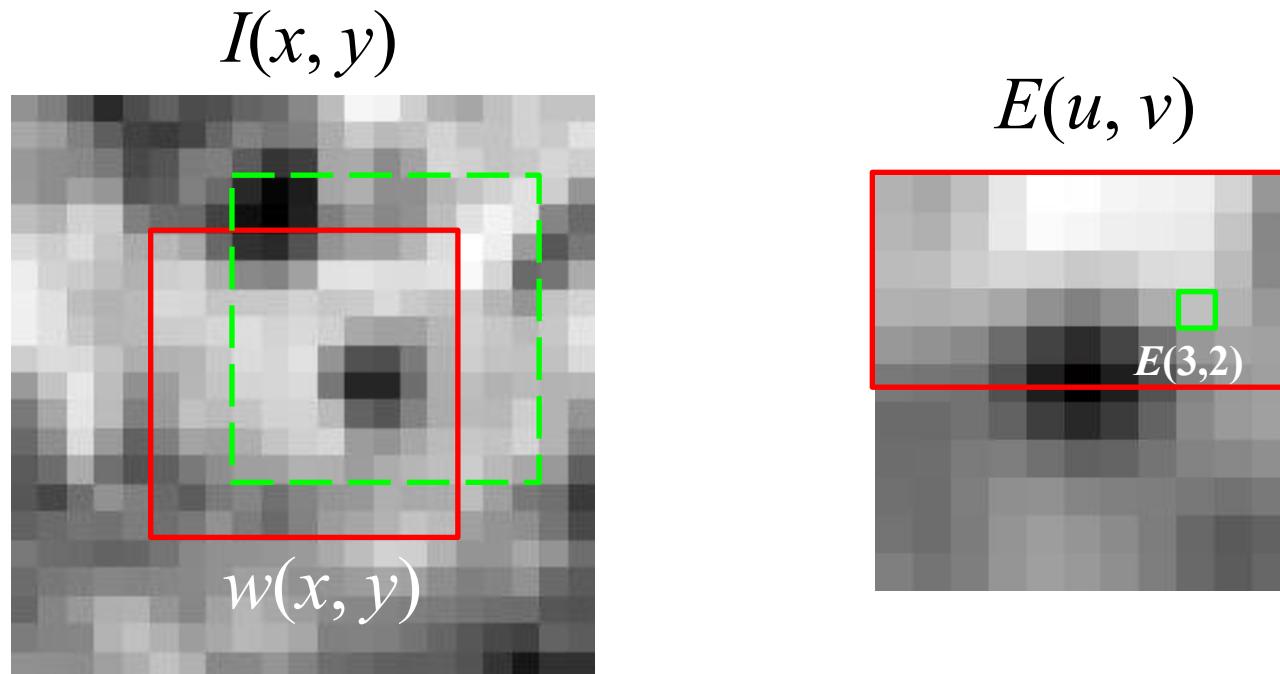
$$E(u, v) = \sum_{x,y} w(x, y) [I(x+u, y+v) - I(x, y)]$$



# Corner Detection by Auto-correlation

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

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



# Corner Detection by Auto-correlation

---

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

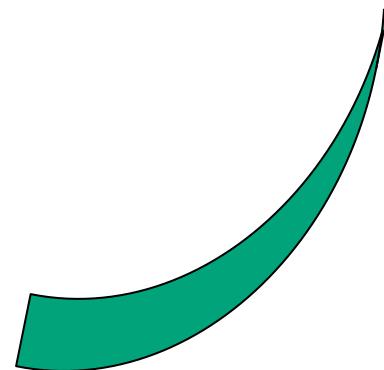
$$E(u, v) = \sum_{x,y} w(x, y) [I(x+u, y+v) - I(x, y)]$$



**We want to discover how E behaves for small shifts  
(corner = function value change fast w.r.t small shifts)**

But this is very slow to compute naively.

$O(\text{window\_width}^2 * \text{shift\_range}^2 * \text{image\_width}^2)$



$O( 11^2 * 11^2 * 600^2 ) = 5.2 \text{ billion of these 14.6k ops per image pixel}$

# Corner Detection by Auto-correlation

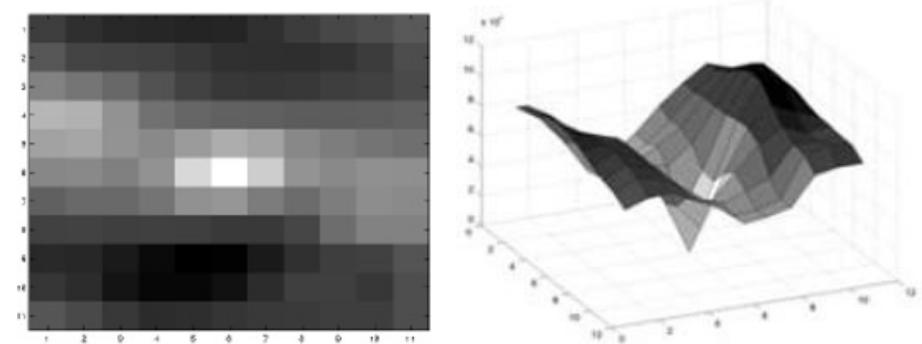
---

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

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

....But we know the response in  $E$  that we are looking for – **strong peak!**

- $E$  needs to “change” fast w.r.t.  $u$  &  $v$
- (from  $u = 0, v = 0$ )



## Recall: Taylor series expansion

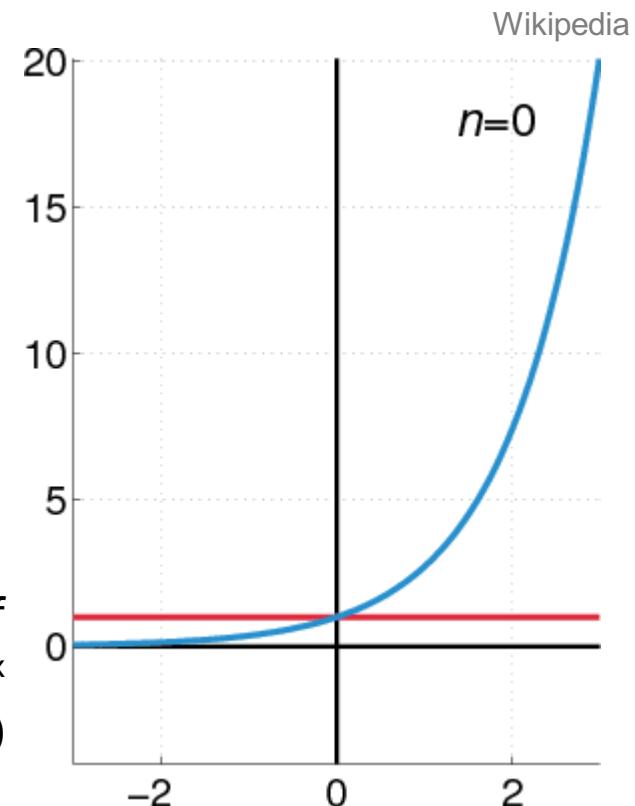
---

A function  $f$  can be represented by an infinite series of its derivatives at a single point  $a$ :

$$f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \frac{f'''(a)}{3!}(x - a)^3 + \dots$$

As we care about window centered, we set  $a = 0$   
(MacLaurin series)

Approximation of  
 $f(x) = e^x$   
centered at  $f(0)$



# Corner Detection: Mathematics (Simplified)

---

- First-order Taylor approximation for small shifts  $(u, v)$ :

$$I(x + u, y + v) \approx I(x, y) + I_x u + I_y v$$

(Why first-order is good enough?)

- Let's plug this into  $E(u, v)$ :

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

(We ignore  $W$  here for simplicity)

$$\approx \sum_{(x,y) \in W} [I(x, y) + I_x u + I_y v - I(x, y)]^2$$

$$= \sum_{(x,y) \in W} [I_x u + I_y v]^2 = \sum_{(x,y) \in W} I_x^2 u^2 + 2I_x I_y uv + I_y^2 v^2$$

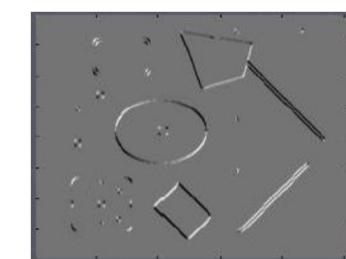
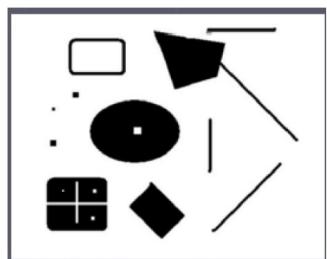
# Corners as distinctive interest points

---

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

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

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



Notation:

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

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

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

# Corners as distinctive interest points

---

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

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

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

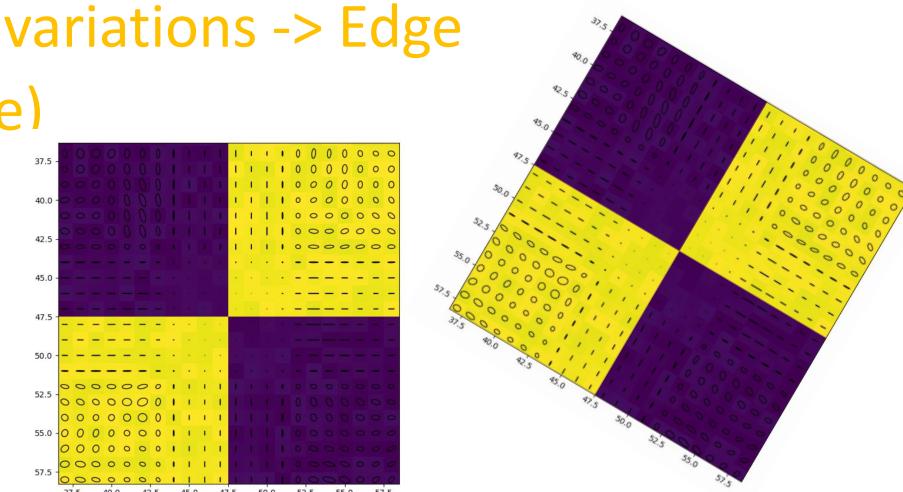
## Reminder/Refresher:

- Our goal is to find  $(x, y)$  likely at corner.  $(u, v)$  denotes a small neighborhood near  $(x, y)$
- $E(u, v)$  is evaluated at each  $(x, y)$ . Its “parameter” depends on  $(x, y)$ , e.g.,  $M$
- For each  $(x, y)$ , we want to find “extreme” values for  $E(u, v)$  -- now reducing to analyzing  $M$
- $M$  encodes the “**variation**” level of  $E(u, v)$  in the small  $(u, v)$  neighborhood – how to decode?

# Let's go back to our goal: corner detection

- For detecting “cornerness”:
  - Do we care about the change orientation? **No**
  - Do we care about the change “steepness”? **Yes, that is “all we need”**
- So, looking at the  $M$  approximation now, what we really want?
  - **What if  $I_x^2, I_y^2, I_x I_y$  are all small?** No variations -> flat area
  - **What if only  $I_x^2$  is large?** Only x-direction has large variations -> Edge
  - **How about only large  $I_y^2$ , or  $I_x I_y$ ?** Same thing (edge)
  - **Then, how about letting  $I_x^2, I_y^2, I_x I_y$  all be large?**
    - Sufficient, but not necessary...
    - **The missing key: Rotation Invariance**

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



# Eigenvalue Analysis (your old friend: PCA)

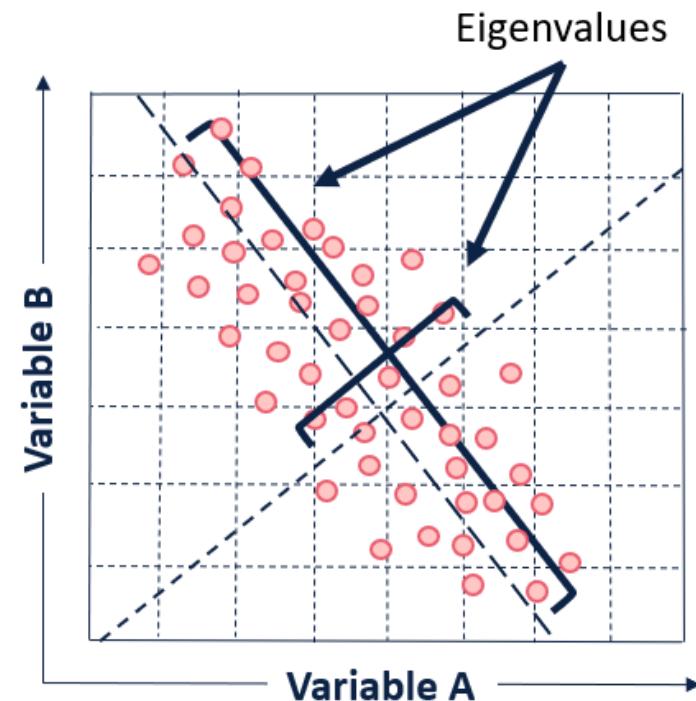
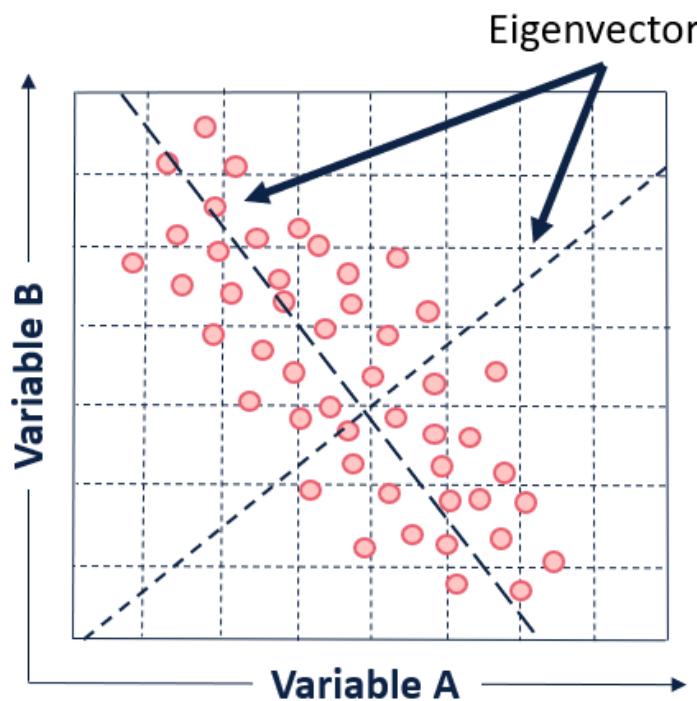
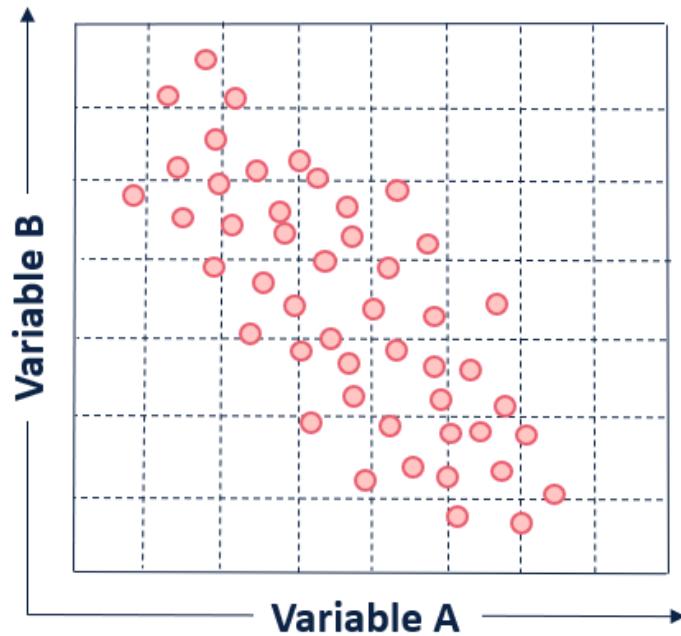
- **Goal:** Describe the “overall intensity variations” in the window, *regardless of rotation!*

... by **eigenvalue analysis**

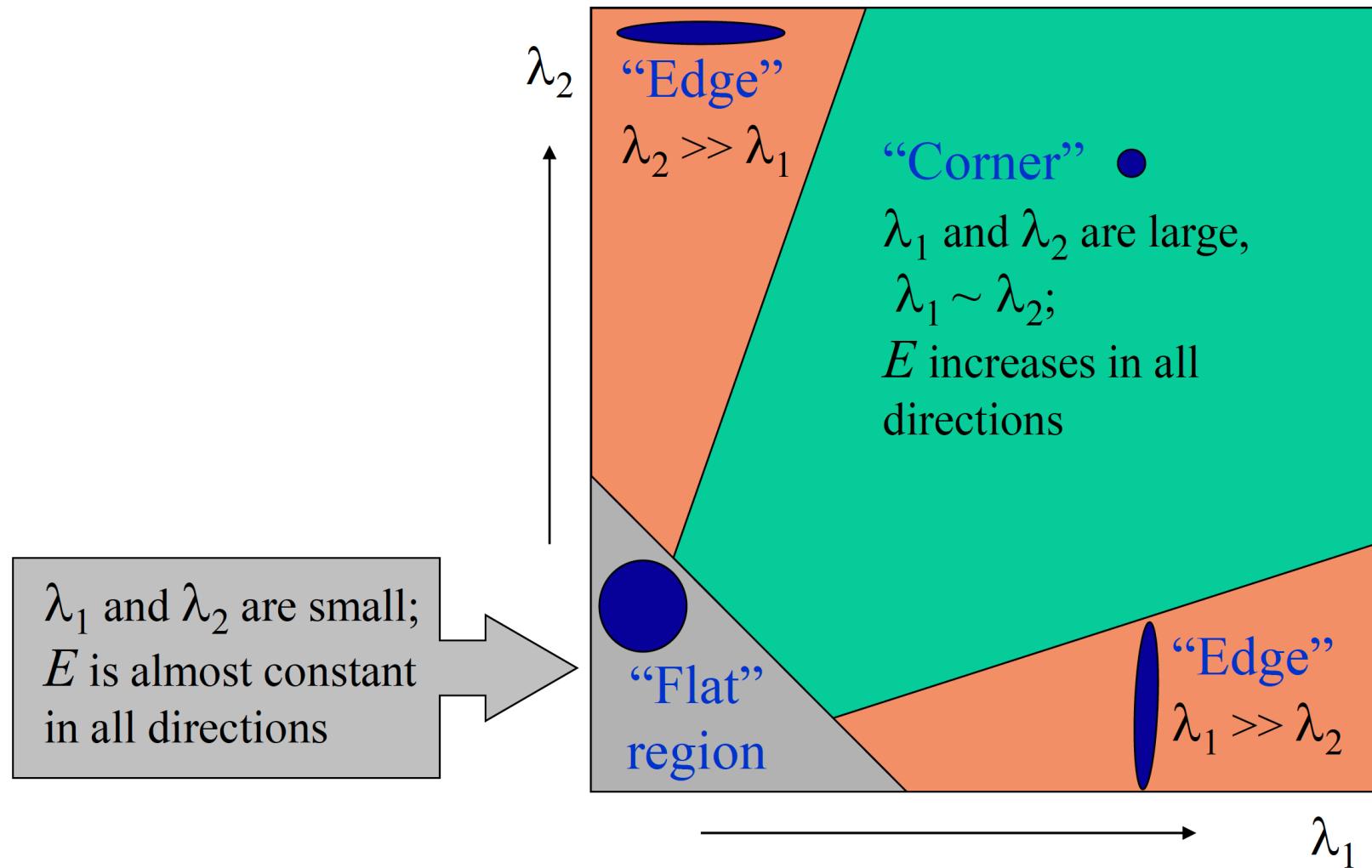
$$E(u, v) \cong [u, v] M \begin{bmatrix} u \\ v \end{bmatrix} \quad \lambda_1, \lambda_2 - \text{eigenvalues of } M$$

***What PCA can tell us about the overall “variations”***

- *Eigenvectors told us the 1<sup>st</sup>/2<sup>nd</sup>/3<sup>rd</sup> ... major directions of change*
- *Correspondingly, eigenvalues capture “change rate” along each of those directions*



# Categorizing image points using $M$ eigenvalues



# Categorizing image points using $M$ eigenvalues

Cornerness score:

$$C = \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

$\alpha$ : some small constant ( $\sim 0.04$  to  $0.06$ )

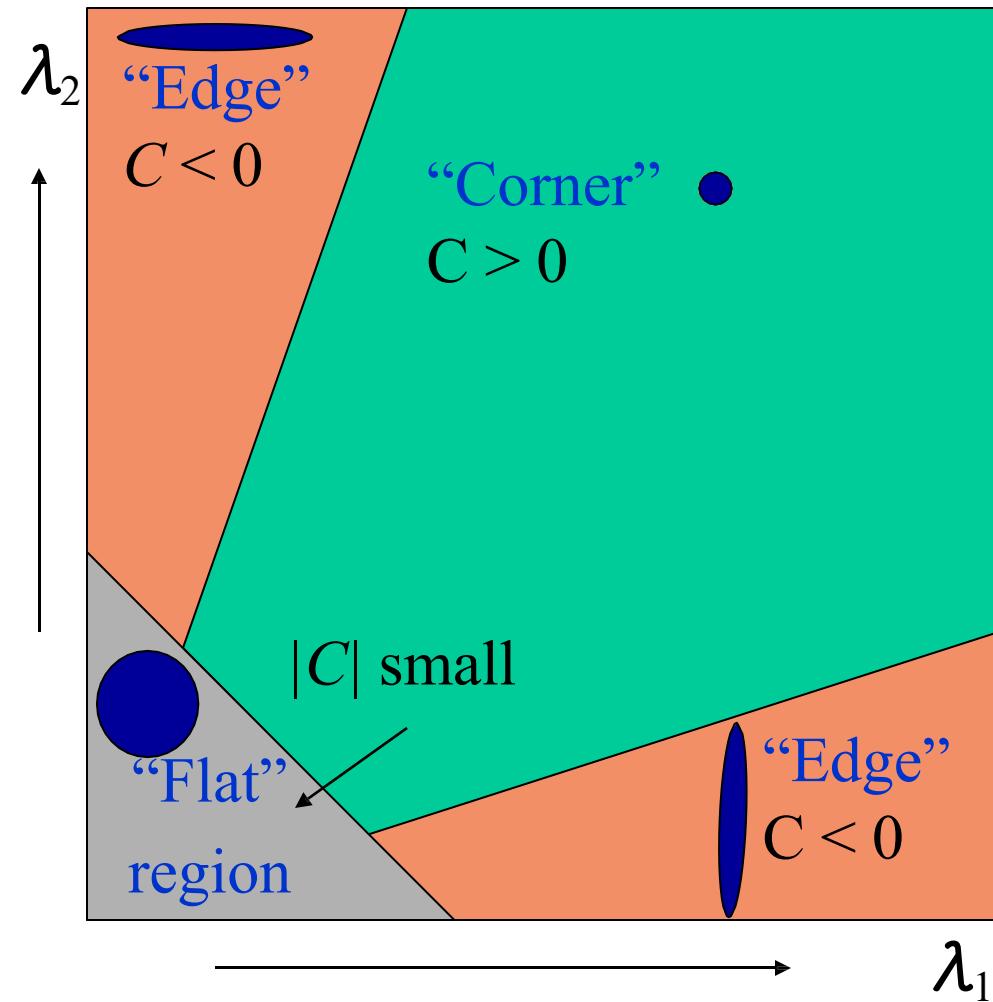
**To see why:**

Assume  $\lambda_1 = k\lambda_2$ ,

$$C = [k - \alpha(k+1)^2] \lambda_2^2$$

Then analyze:  $k - \alpha(k+1)^2$

**What if  $k$  is very large? very small? around 1?**



# Categorizing image points using $M$ eigenvalues

Cornerness score:

$$C = \lambda_1 \lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

$\alpha$ : some constant ( $\sim 0.04$  to  $0.06$ )

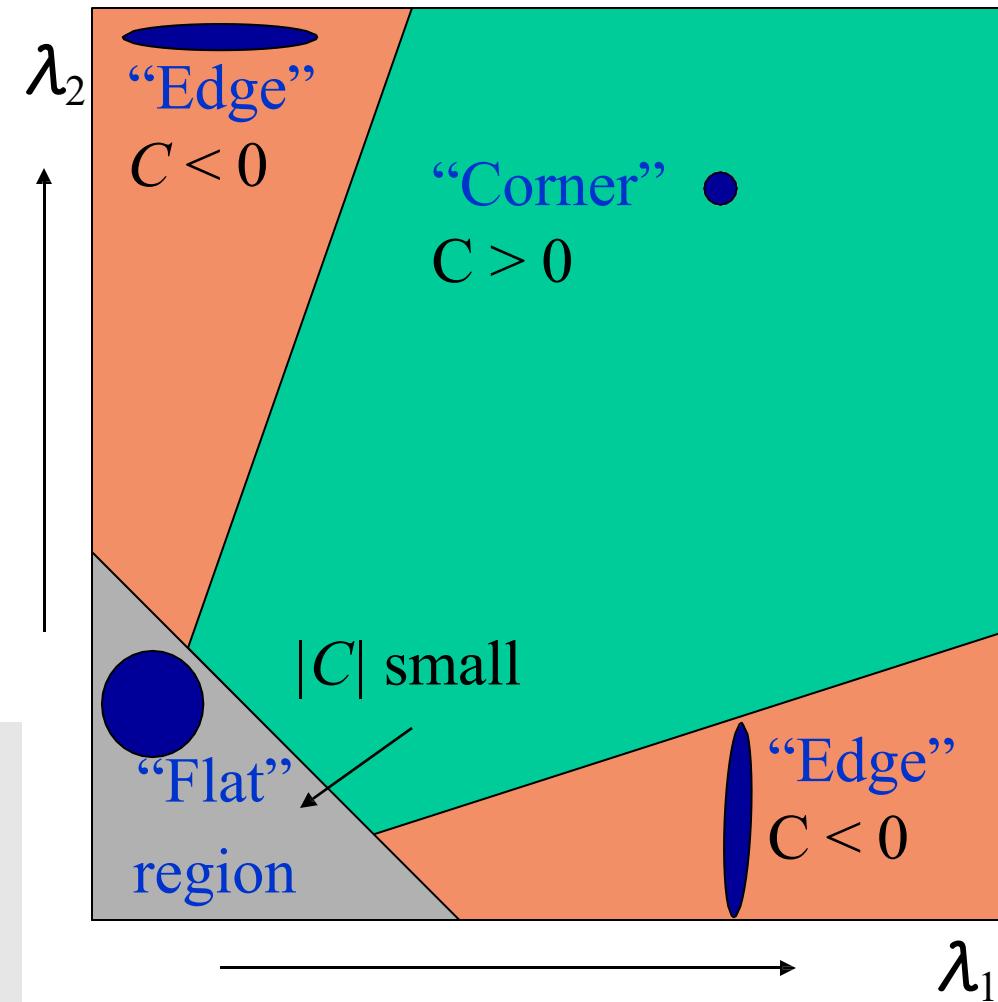
*Remember your linear algebra:*

Determinant:  $\det(A) = \prod_{i=1}^n \lambda_i = \lambda_1 \lambda_2 \dots \lambda_n$ .  
(diagonal matrices)

Trace:  $\text{tr}(A) = \sum_i \lambda_i$ .

$$C = \det(M) - \alpha \text{Tr}^2(M)$$

Avoids explicit eigenvalue computation!  
**(why?)**



# This is the "notorious" Harris corner detector!

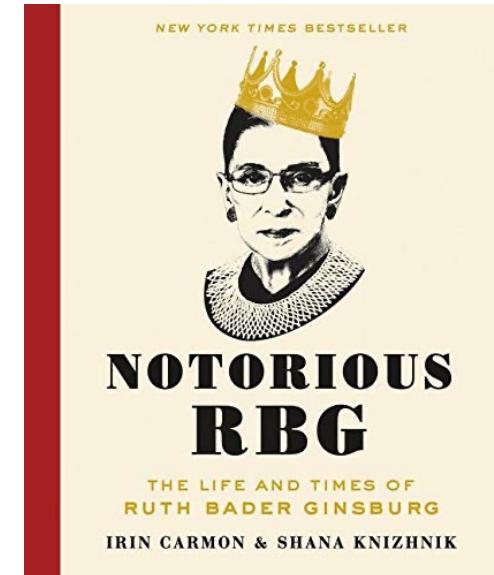
---

- 1) Compute  $M$  matrix for each window to recover the *cornerness* score  $C$ .

Note: We can build  $M$  purely from the per-pixel image derivatives!

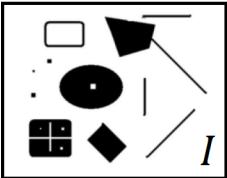
- 2) Threshold to find pixels which give large corner response ( $C >$  threshold).

- 3) Find the local maxima pixels, i.e., non-maximal suppression.



C.Harris and M.Stephens. "[A Combined Corner and Edge Detector.](#)"  
*Proceedings of the 4th Alvey Vision Conference*: pages 147—151, 1988.

# Harris Corner Detector [Harris88]

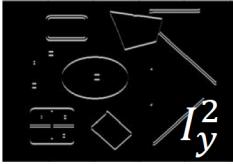


0. Input image

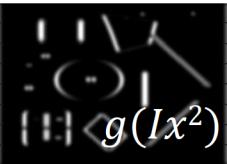
We want to compute  $M$  at each pixel.



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



2. Compute  $M$  components  
as squares of derivatives.



3. Gaussian filter  $g()$  with width  $\sigma$

$$= g(I_x^2), g(I_y^2), g(I_x \circ I_y)$$

Reminder:  $a \circ b$  is  
Hadamard product  
(element-wise  
multiplication)

4. Compute cornerness

$$\begin{aligned} C &= \det(M) - \alpha \operatorname{trace}(M)^2 \\ &= g(I_x^2) \circ g(I_y^2) - g(I_x \circ I_y)^2 \\ &\quad - \alpha [g(I_x^2) + g(I_y^2)]^2 \end{aligned}$$



5. Threshold on  $C$  to pick high cornerness

6. Non-maximal suppression to pick peaks.

# Harris Detector: Steps

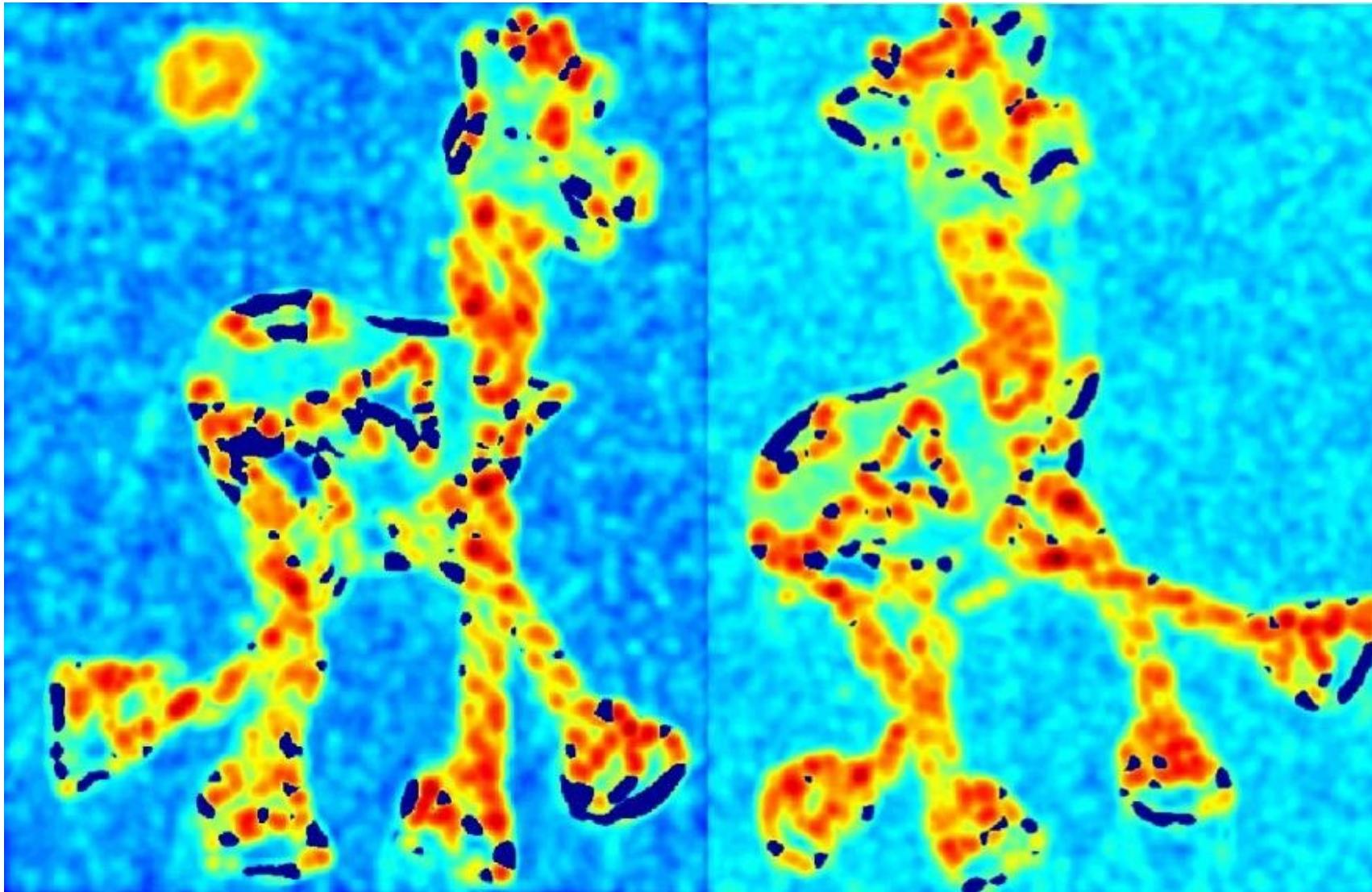
---



# Harris Detector: Steps

---

Compute corner response  $C$



# Harris Detector: Steps

---

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



# Harris Detector: Steps

---

Take only the points of local maxima of  $C$



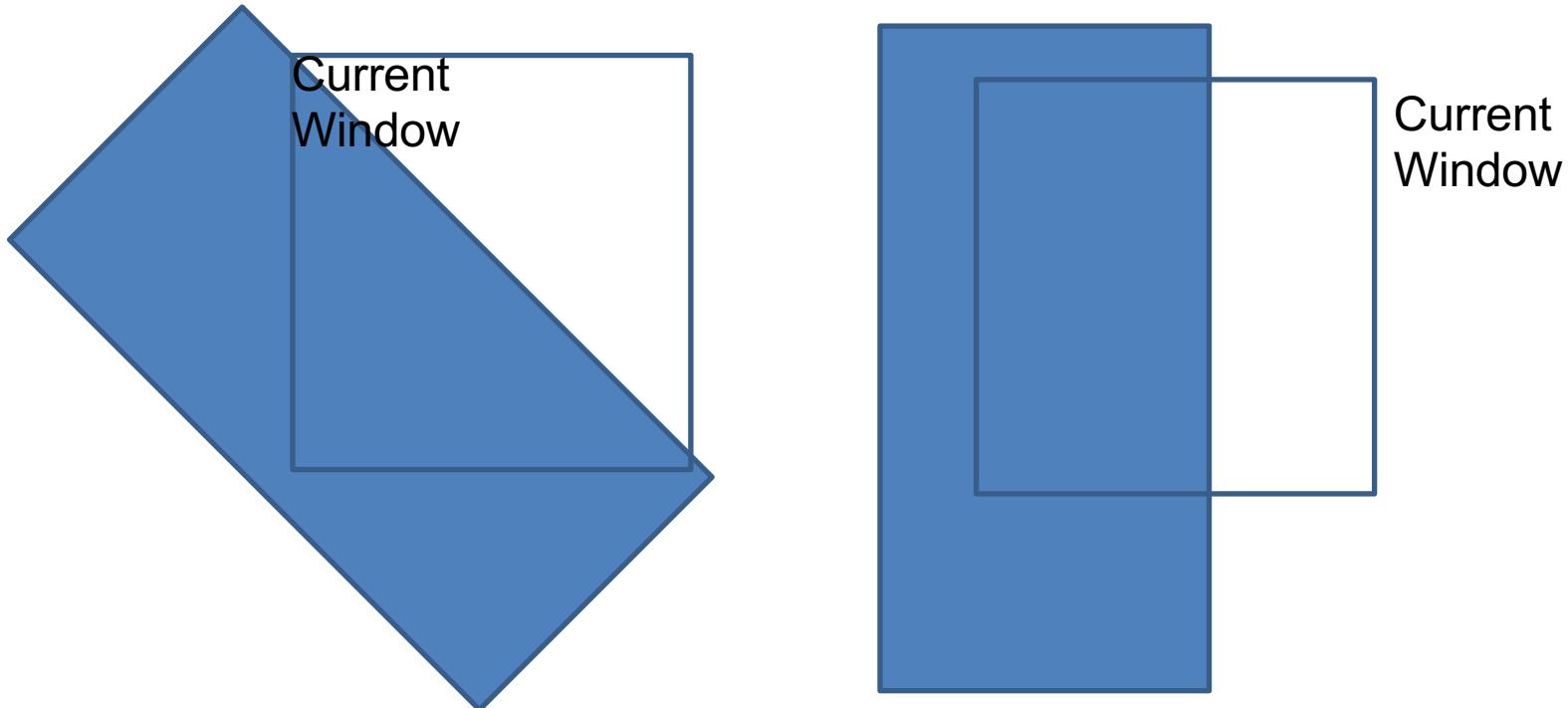
# Harris Detector: Steps

---



# Harris Corners – Why so complicated?

- Can't we just check for regions with lots of gradients in the x and y directions (or any specific)?
  - No! A diagonal line or alike would satisfy that criteria



# Invariance and covariance

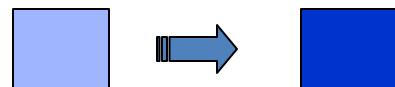
---

Are locations *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

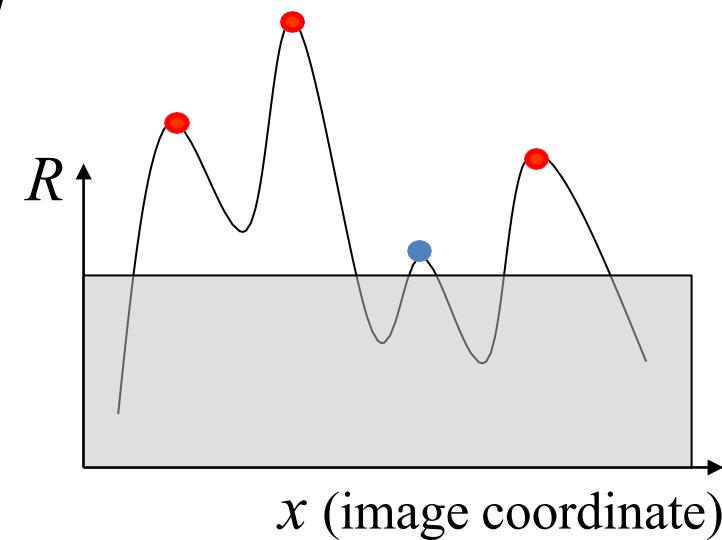
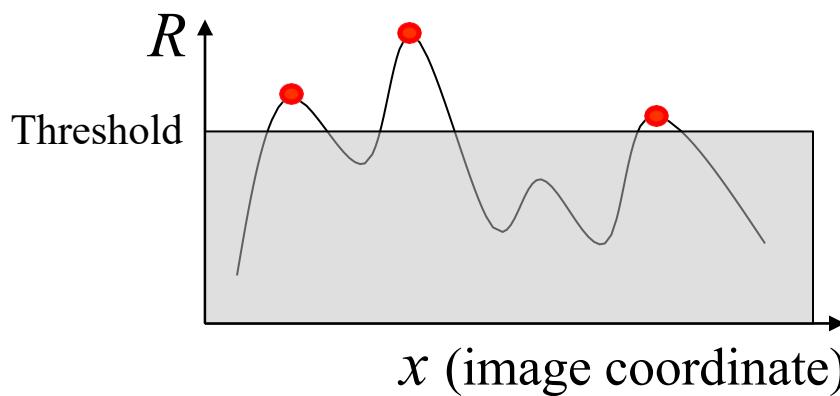


# Affine intensity change



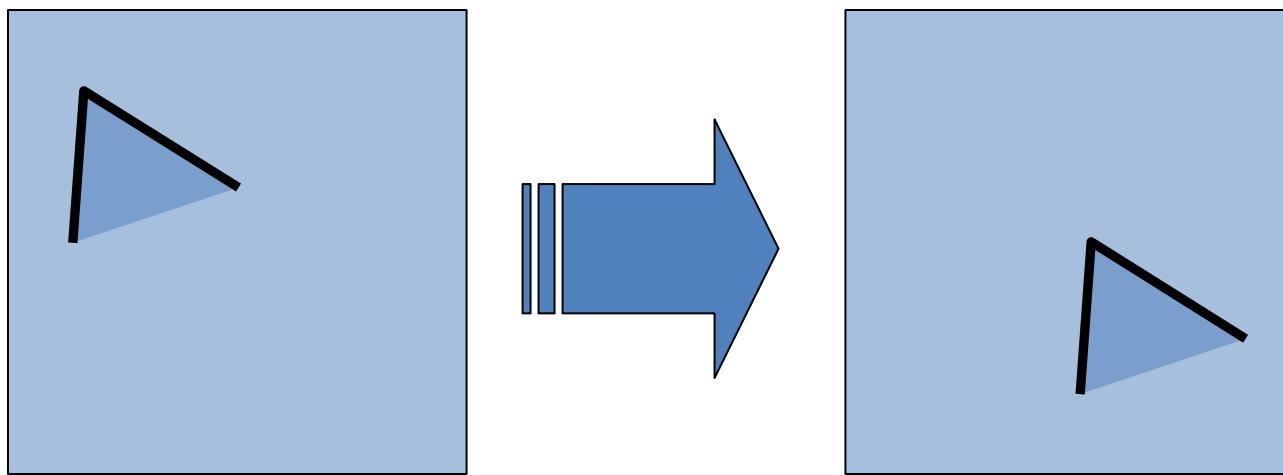
$$I \rightarrow a I + b$$

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



*Partially invariant to affine intensity change*

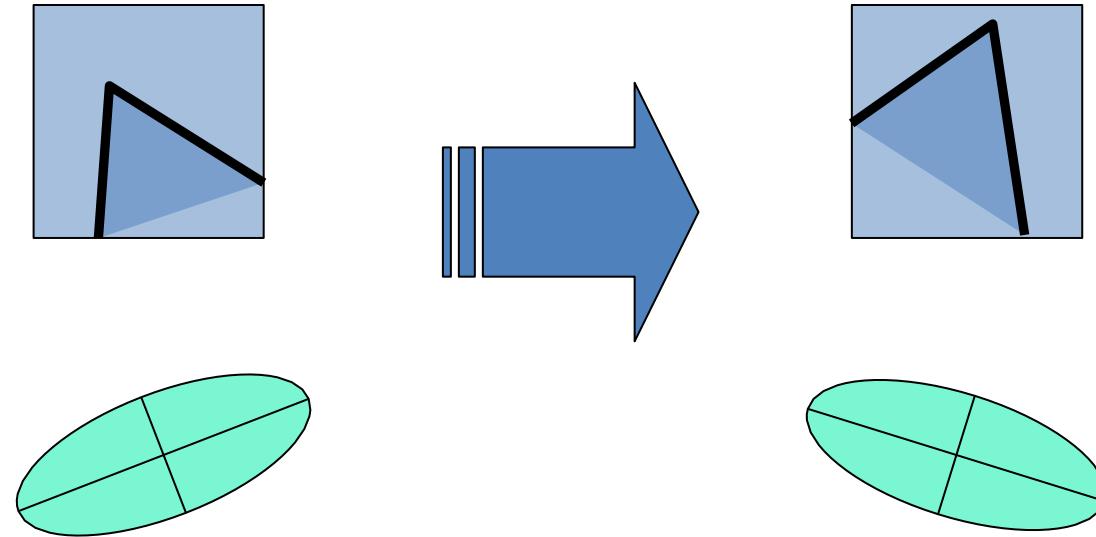
# Image translation



- Derivatives and window function are shift-invariant.

Corner location is covariant w.r.t. translation

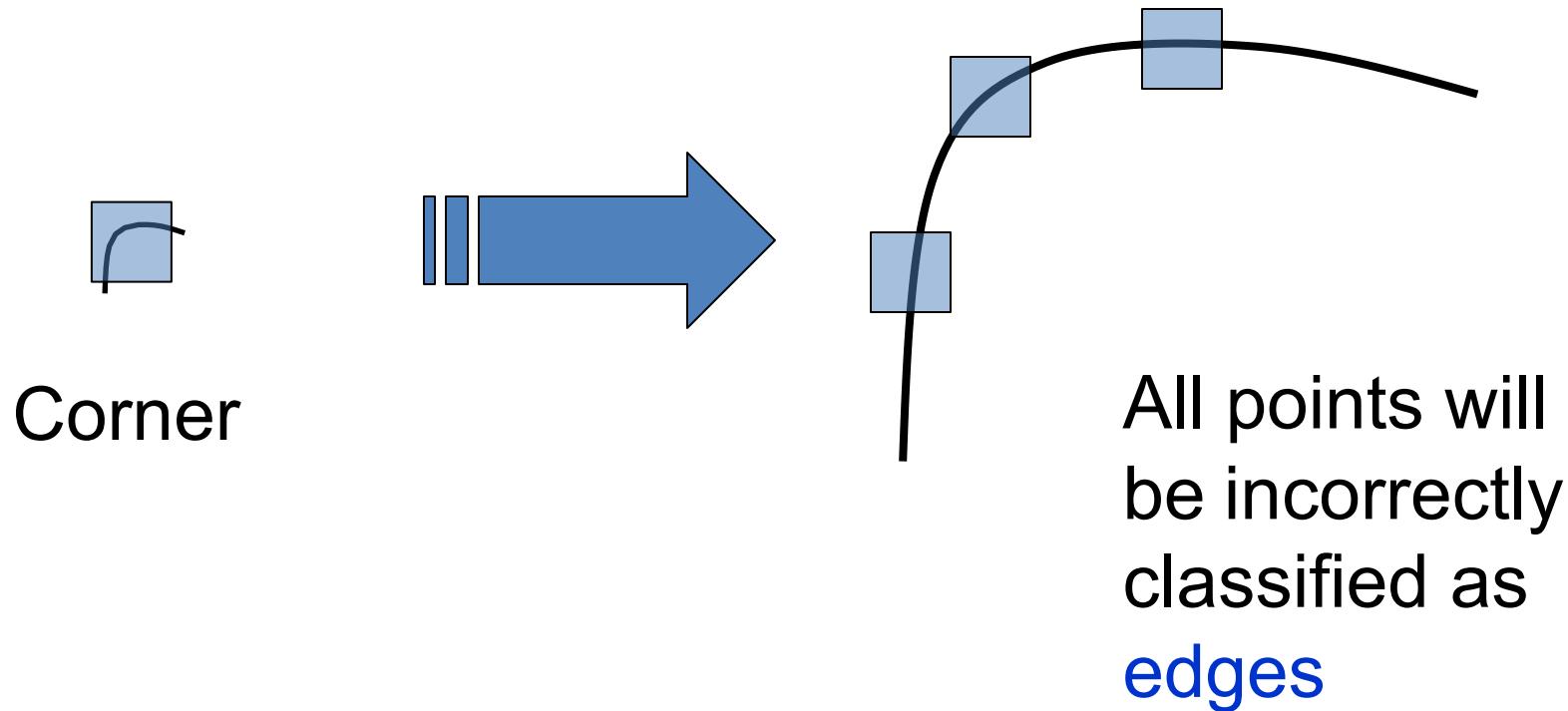
# Image rotation



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

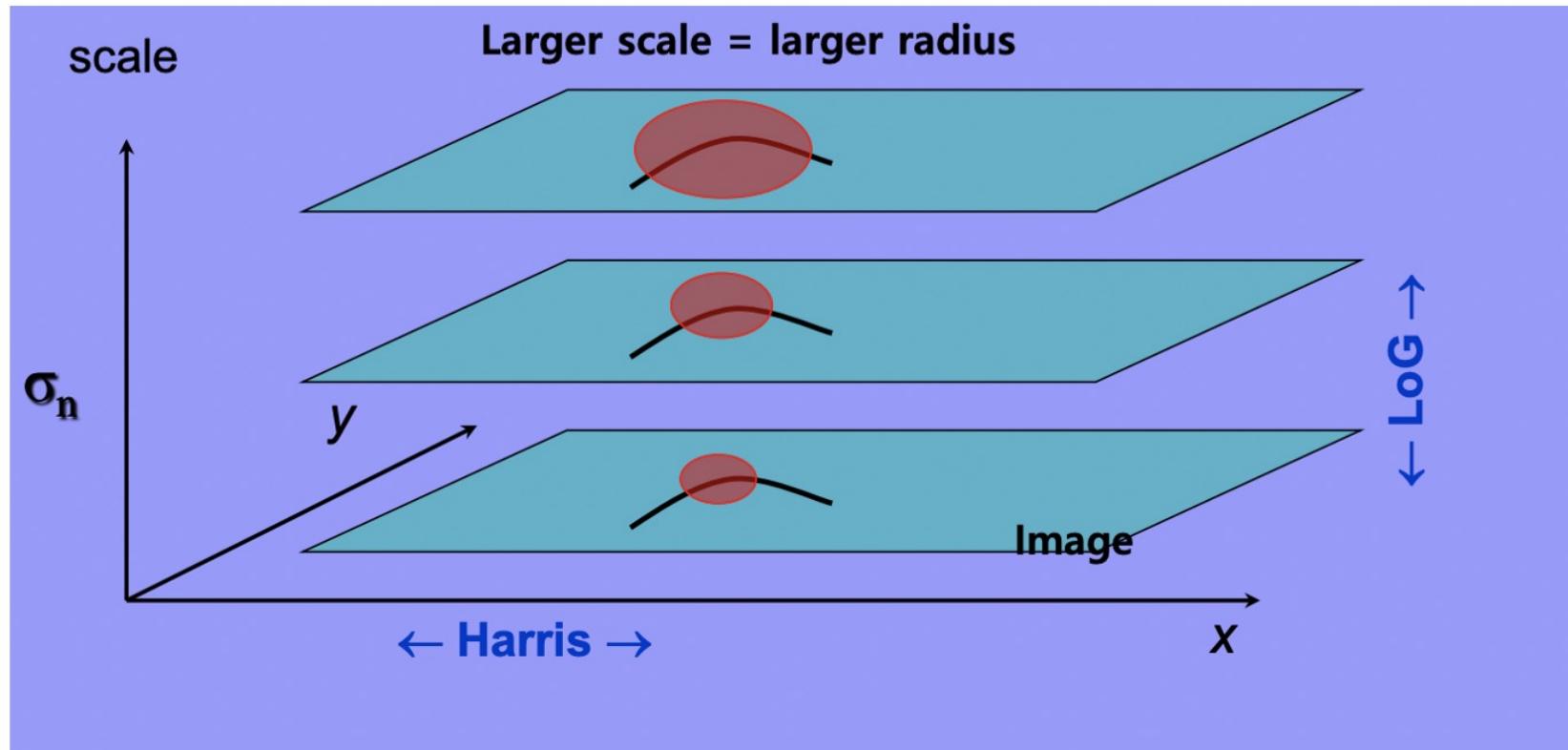
Corner location is covariant w.r.t. rotation

# Scaling



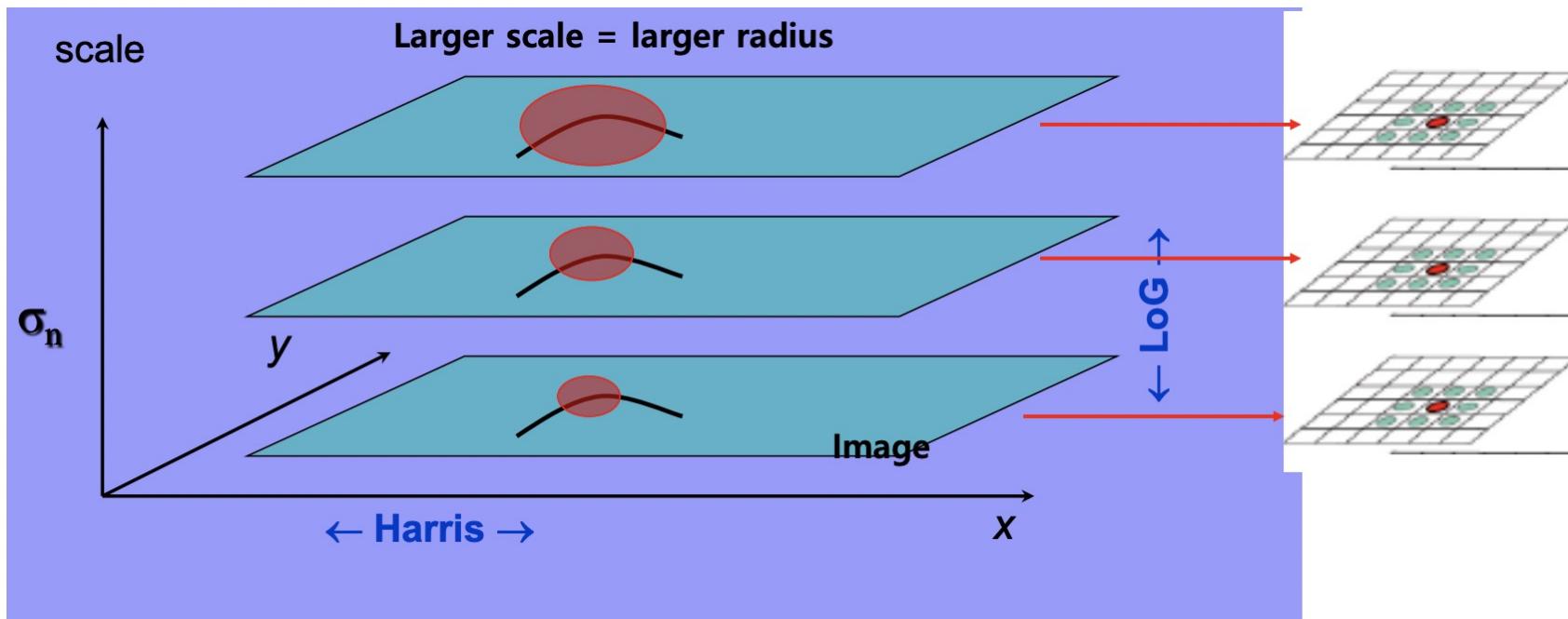
Corner location is not covariant to scaling!

# Harris-Laplace detector [Mikolajczyk '01]



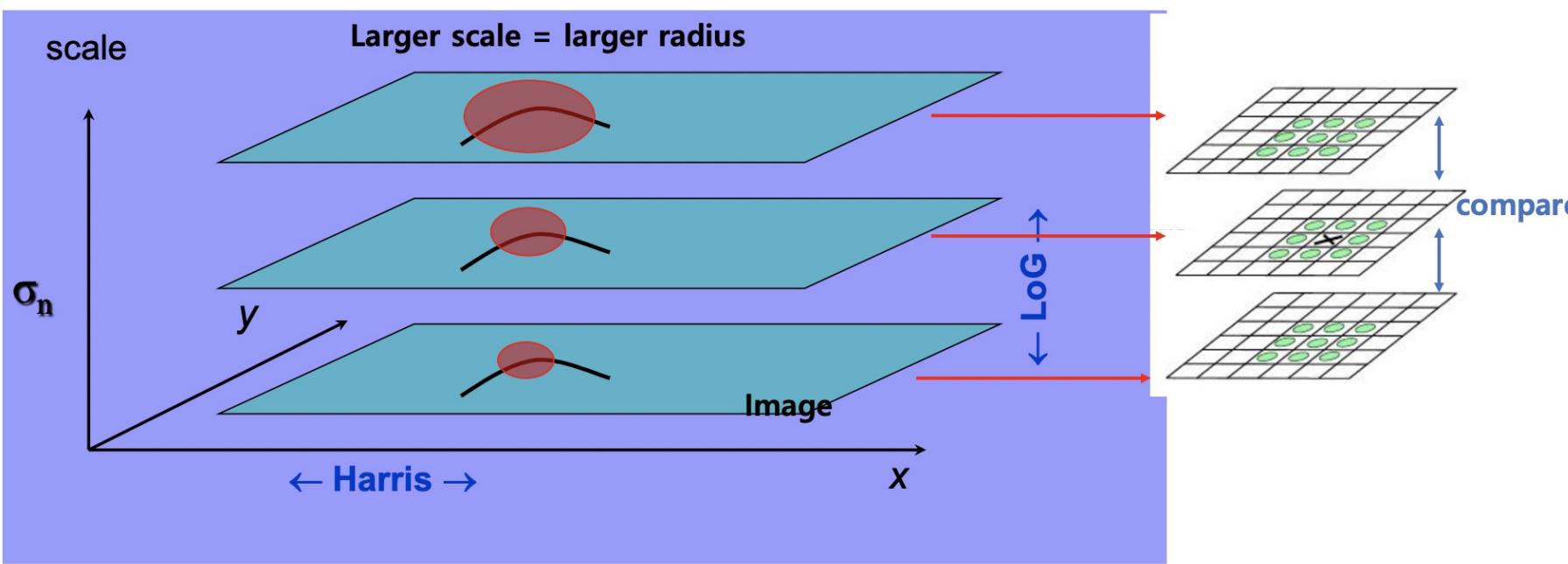
- **Step 1.** Build the Laplacian Pyramid of one image

# Harris-Laplace detector [Mikolajczyk '01]



- **Step 1.** Build the Laplacian Pyramid of one image
- **Step 2.** Run the *Harris* detector to compute interest points *at each scale*

# Harris-Laplace detector [Mikolajczyk '01]



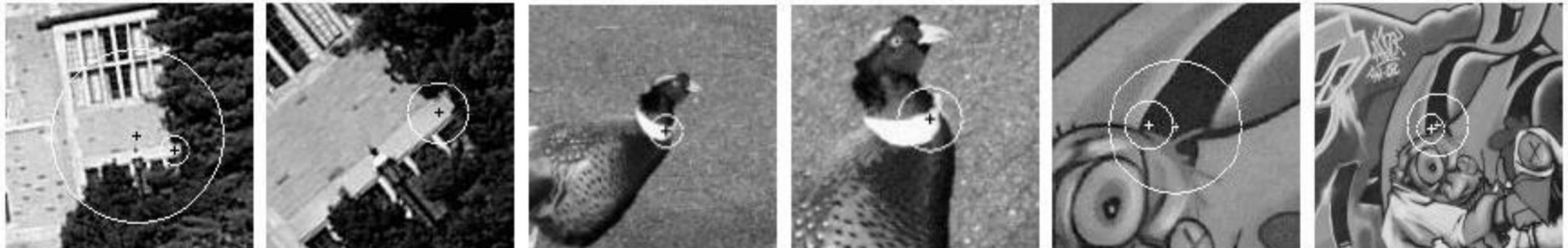
- **Step 1.** Build the Laplacian Pyramid of one image

**Step 2.** Run the *Harris* detector to compute interest points *at each scale*

**Step 3.** Non-maximal suppression, not only at each scale, **but also at adjacent scales**

# Harris-Laplace detector [Mikolajczyk '01]

- A scale-invariant detector!
  - Automatically search for the right scale to detect corners, by “multi-scaling then max-pooling”



Harris-Laplace points

# A Longer List of Local Keypoint Detectors...

Table 7.1 Overview of feature detectors.

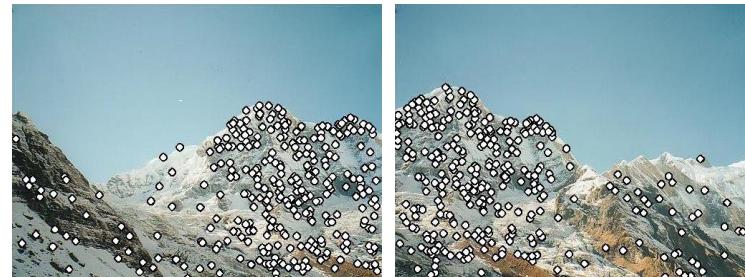
Feature Detector	Corner	Blob	Region	Rotation invariant	Scale invariant	Affine invariant	Repeatability	Localization accuracy	Robustness	Efficiency
Harris	√			√			+++	+++	+++	++
Hessian		√		√			++	++	++	+
SUSAN	√			√			++	++	++	+++
Harris-Laplace	√	(√)		√	√		+++	+++	++	+
Hessian-Laplace	(√)	√		√	√		+++	+++	+++	+
DoG	(√)	√		√	√		++	++	++	++
SURF	(√)	√		√	√		++	++	++	+++
Harris-Affine	√	(√)		√	√	√	+++	+++	++	++
Hessian-Affine	(√)	√		√	√	√	+++	+++	+++	++
Salient Regions	(√)	√		√	√	(√)	+	+	++	+
Edge-based	√			√	√	√	+++	+++	+	+
MSER			√	√	√	√	+++	+++	++	+++
Intensity-based			√	√	√	√	++	++	++	++
Superpixels			√	√	(√)	(√)	+	+	+	+

- It is always of interest to collect more points with more detectors, for more possible matches
- Need consider location preciseness, variation robustness, and flexibility in region shapes
- Best choice often application dependent
  - Harris/Harris-Laplace work well for many natural image categories
  - MSER works well for buildings and printed things
  - Although no “silver bullet”, all detectors/descriptors shown here work well in general

# Local features: main components

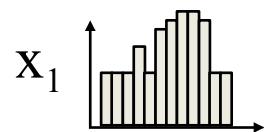
## 1) Detection:

Find a set of distinctive key points.

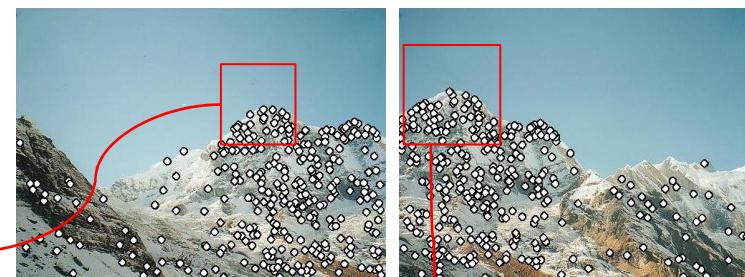


## 2) Description:

Extract feature descriptor around each interest point as vector.



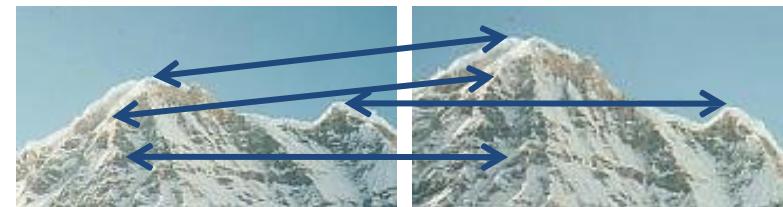
$$\mathbf{x}_1 = [x_1^{(1)}, \dots, x_d^{(1)}]$$



## 3) Matching:

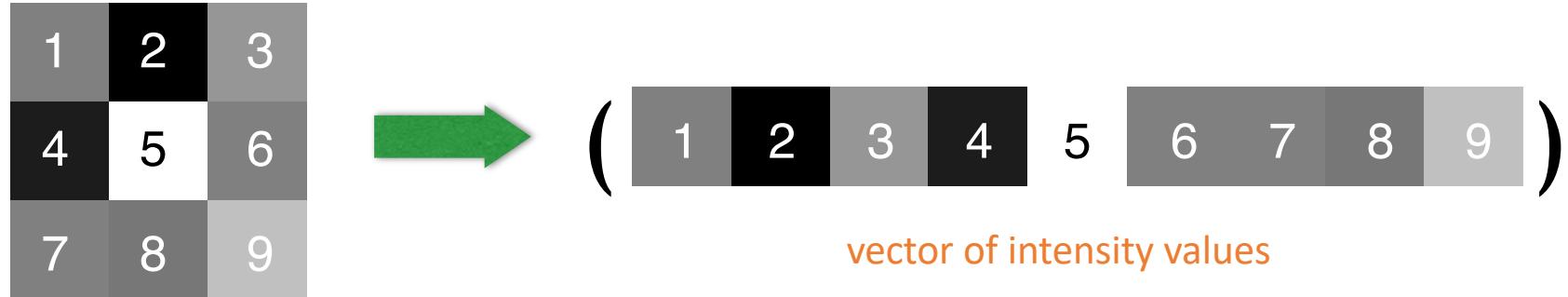
Compute distance between feature vectors to find correspondence.

$$d(\mathbf{x}_1, \mathbf{x}_2) < T$$



# Image patch

Just use the pixel values of the patch



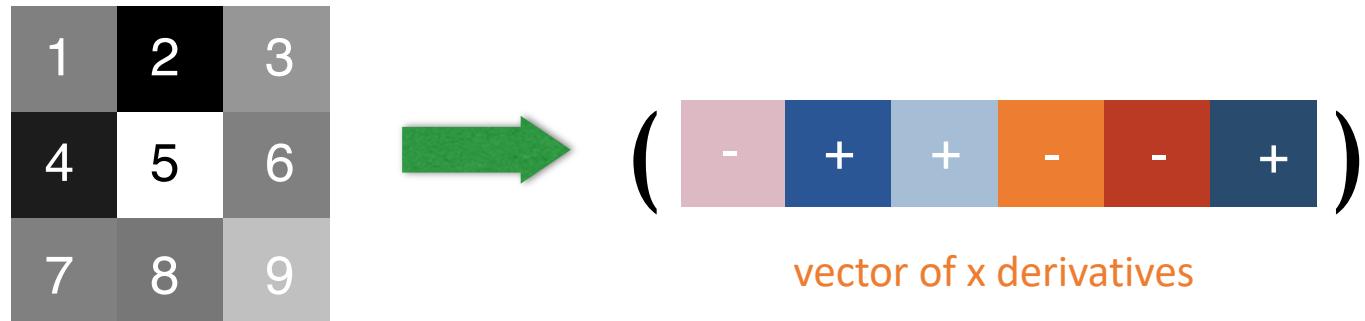
Perfectly fine if geometry and appearance is unchanged (a.k.a. template matching)

*What are the problems?*

*How can you be less sensitive to absolute intensity values?*

# Image gradients

Use pixel differences

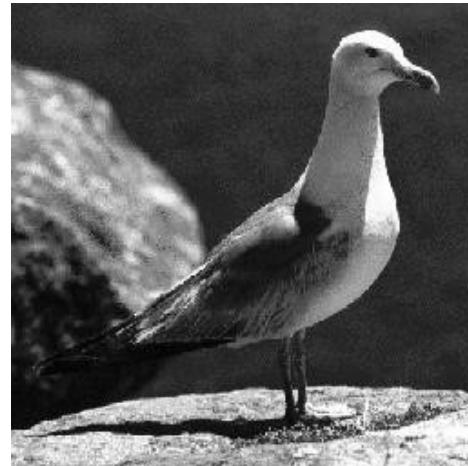
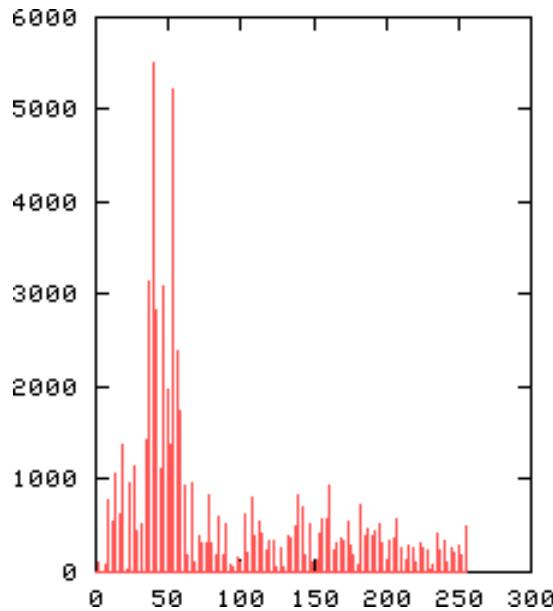


Feature is invariant to absolute intensity values

*What are the problems?*

*How can you be less sensitive to deformations?*

# Image Representations: Histograms



**Motivation:** We want some sensitivity to spatial layout, but **not too much**, so blocks of histograms give us that

Global histogram to represent distribution of features

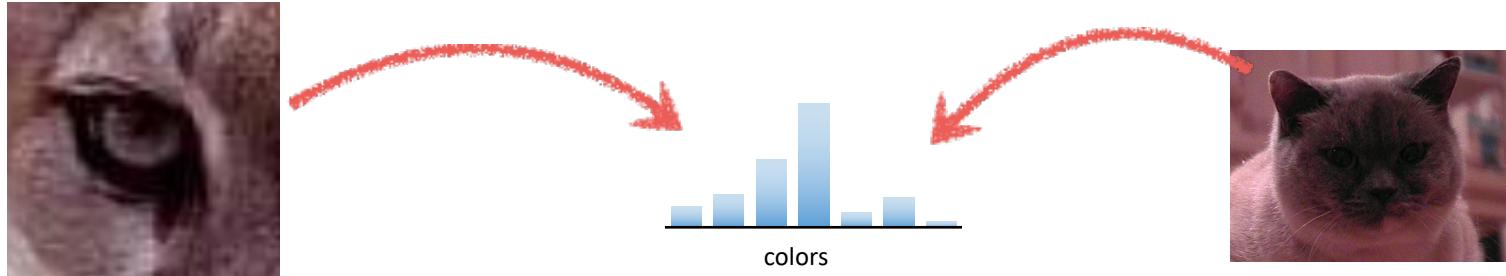
- How ‘well exposed’ a photo is

What about a local histogram per detected point?



# Intensity/Color histogram

Count the colors in the image using a histogram



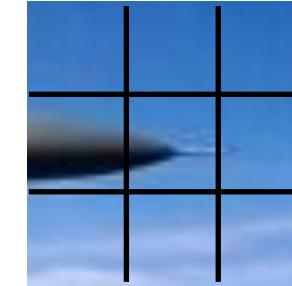
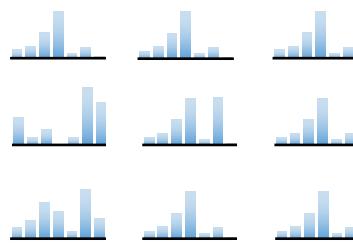
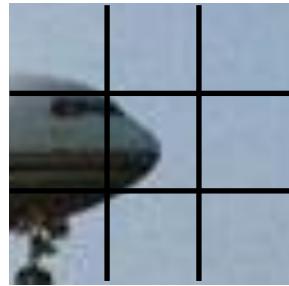
Invariant to changes in scale and rotation

*What are the problems?*

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

# Spatial histograms

Compute histograms over spatial ‘cells’



Retains rough spatial layout

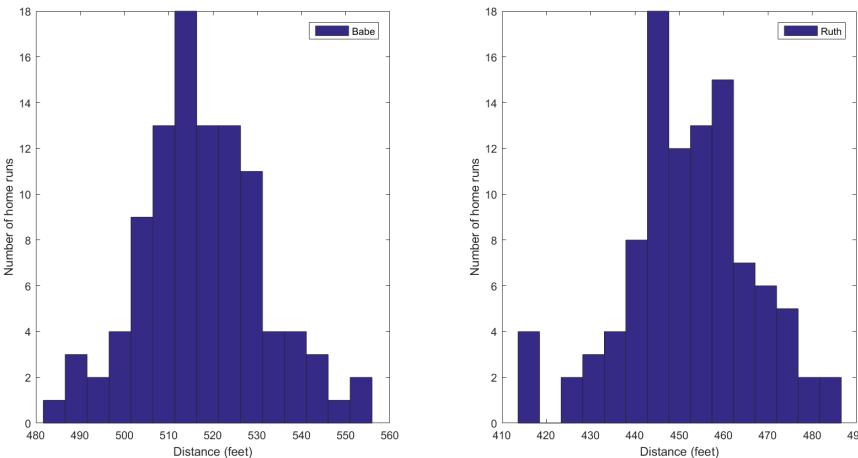
Some invariance to deformations

*What are the problems?*

*How can you be completely invariant to rotation?*

# Computing histogram distance

$$\text{histint}(h_i, h_j) = 1 - \sum_{m=1}^K \min(h_i(m), h_j(m))$$



Histogram intersection (assuming normalized histograms)

$$\chi^2(h_i, h_j) = \frac{1}{2} \sum_{m=1}^K \frac{[h_i(m) - h_j(m)]^2}{h_i(m) + h_j(m)}$$

Chi-squared Histogram matching distance



Cars found by color histogram matching using chi-squared

# Histograms: Implementation issues

- Quantization
  - Grids: fast but applicable only with few dimensions
  - Clustering: slower but can quantize data in higher dimensions



Few Bins  
Need less data  
Coarser representation

Many Bins  
Need more data  
Finer representation

- Matching
  - Histogram intersection or Euclidean may be faster
  - Chi-squared often works better
  - Earth mover's distance is good for when nearby bins represent similar values

# For what things might we compute histograms?

- From Color to Texture/Keypoints
- Histograms of oriented gradients (HOG)
- Scale Invariant Feature Transform (SIFT)
  - Extremely popular (63k citations in 2021)

*IMHO, one of the  
most elegant  
designs ever in CV*

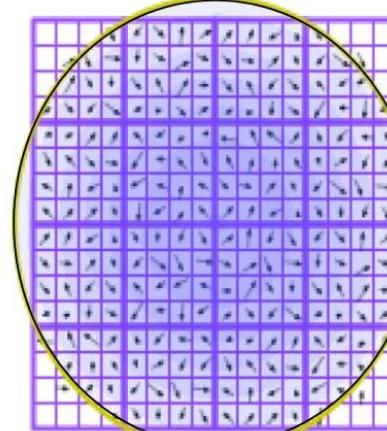
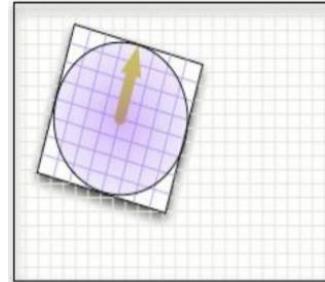
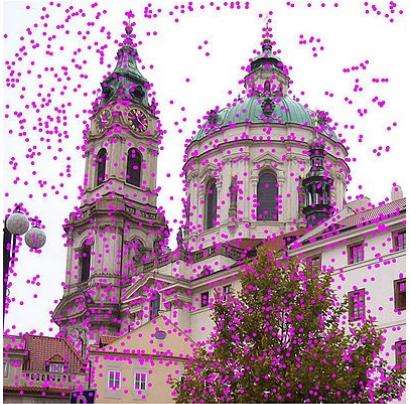


Image gradients

*	*	*	*
*	*	*	*
*	*	*	*
*	*	*	*

Keypoint descriptor

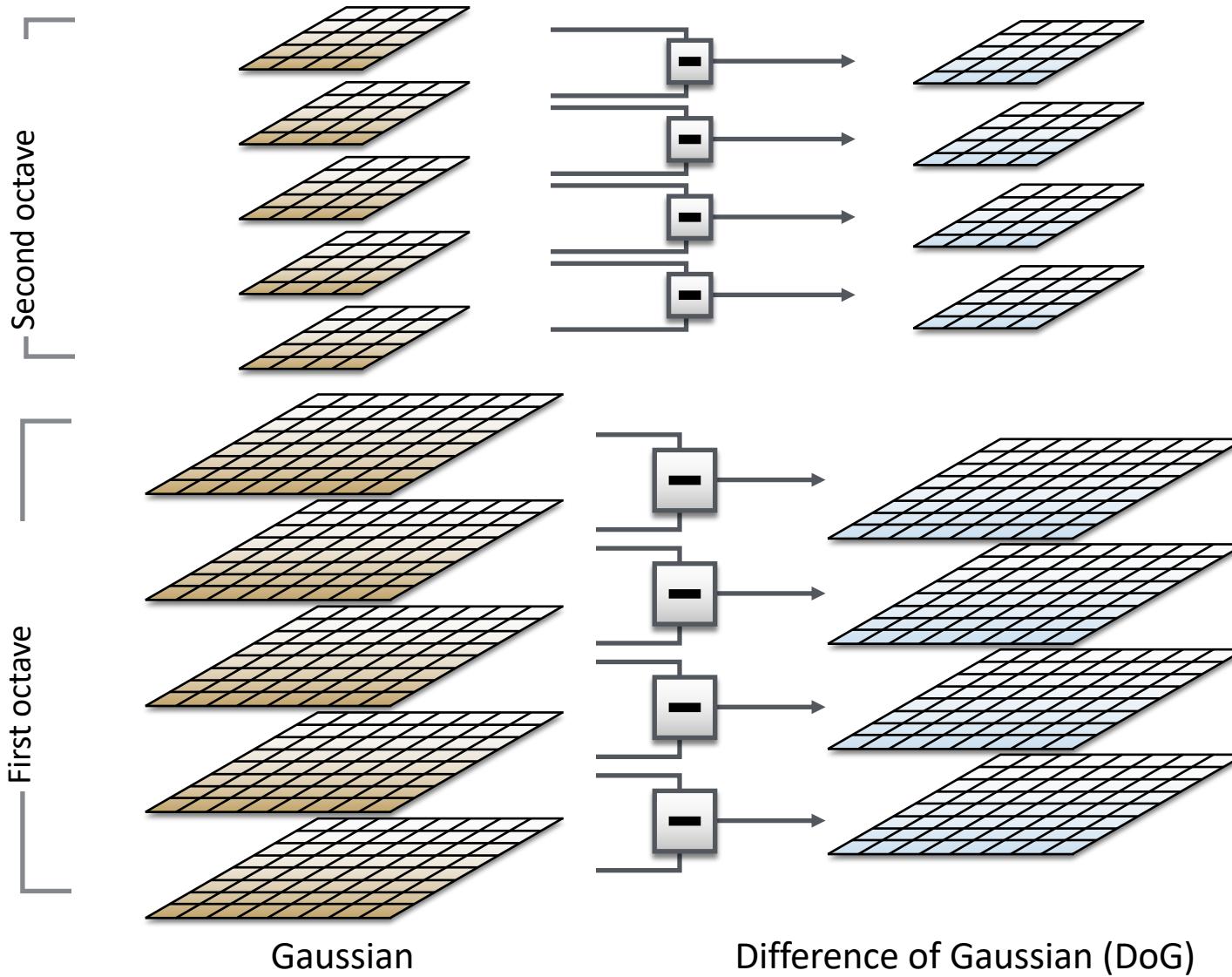


# SIFT (Scale Invariant Feature Transform)

SIFT describes both a **detector** and **descriptor**

1. Multi-scale extrema detection
2. Keypoint localization
3. Orientation assignment
4. Keypoint descriptor

# 1. Multi-scale extrema detection



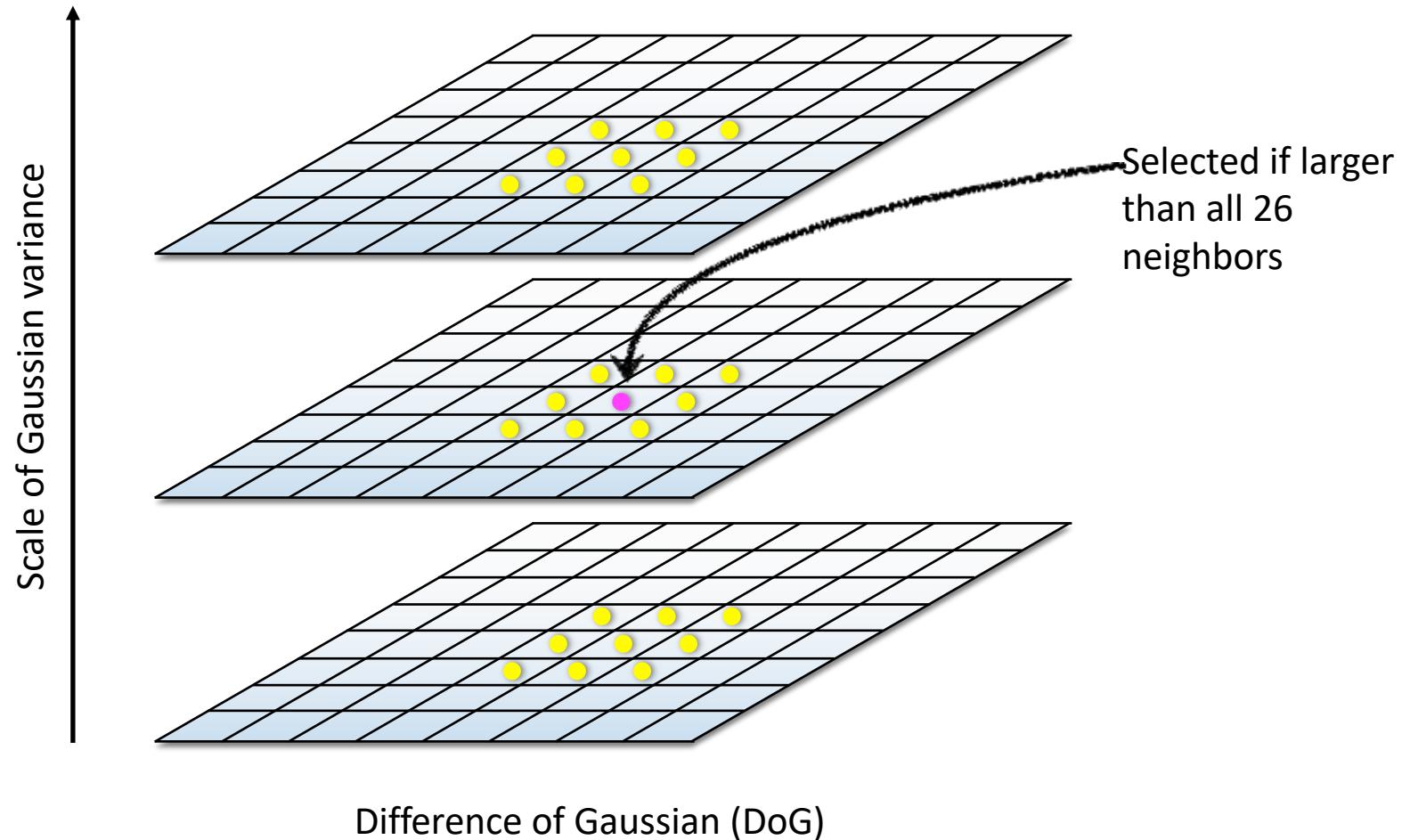


Gaussian



Laplacian

# Scale-space extrema



## 2. Keypoint localization (why this step?)

- Reject flats:

- $|D(\hat{x})| < 0.03$

- Reject edges:

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

Let  $\alpha$  be the eigenvalue with larger magnitude and  $\beta$  the smaller.

$$\text{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\text{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

Let  $r = \alpha/\beta$ .  
So  $\alpha = r\beta$

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

- $r < 10$

$(r+1)^2/r$  is at a min when the 2 eigenvalues are equal.

### 3. Orientation assignment

For a keypoint,  $\mathbf{L}$  is the **Gaussian-smoothed** image at its selected scale,

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

x-derivative                                    y-derivative

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

Detection process returns

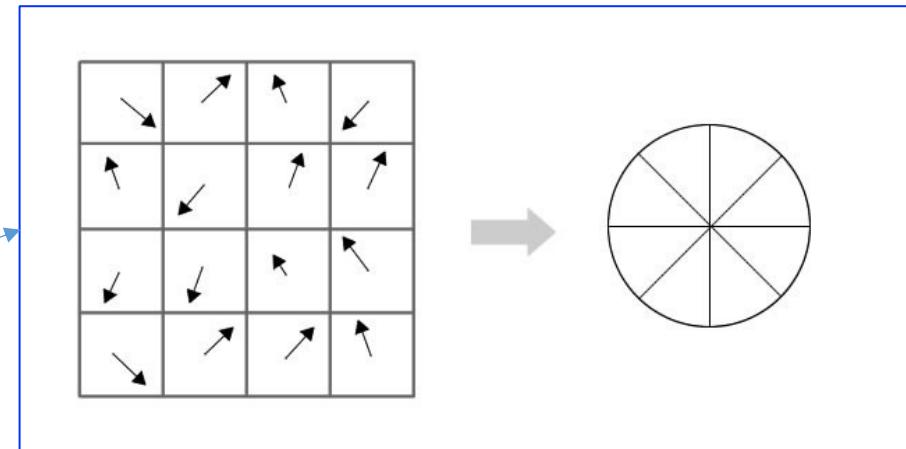
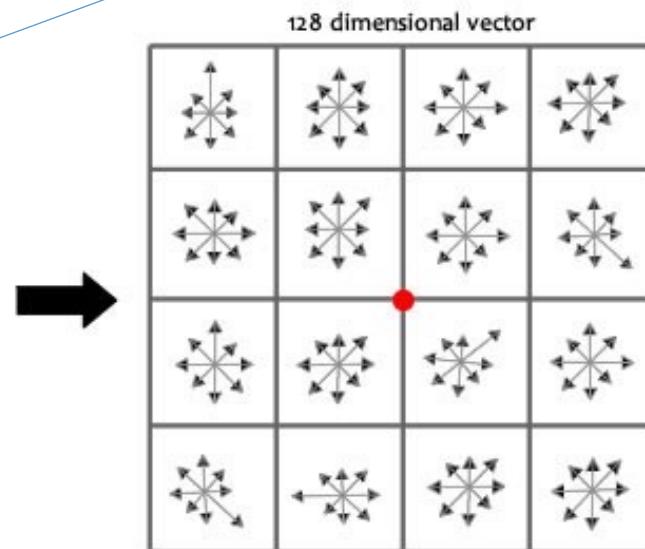
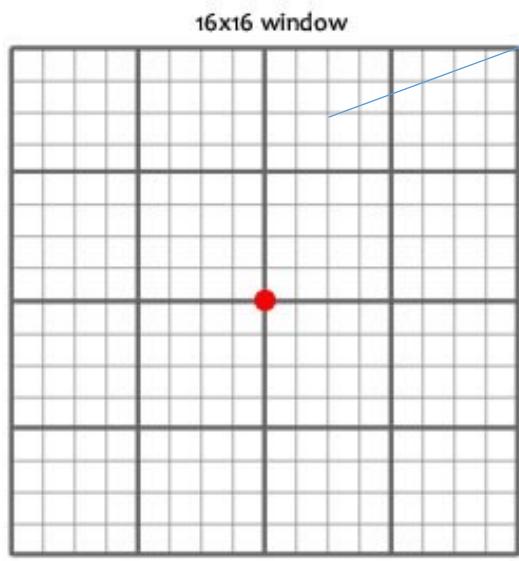
$$\{x, y, \sigma, \theta\}$$

location    scale    orientation

# 4. Keypoint descriptor

*At this moment, each survived keypoint has {location, scale, orientation}...*

- Use local image gradients at selected scale and rotation to describe each keypoint region.



# Adding more invariances to SIFT

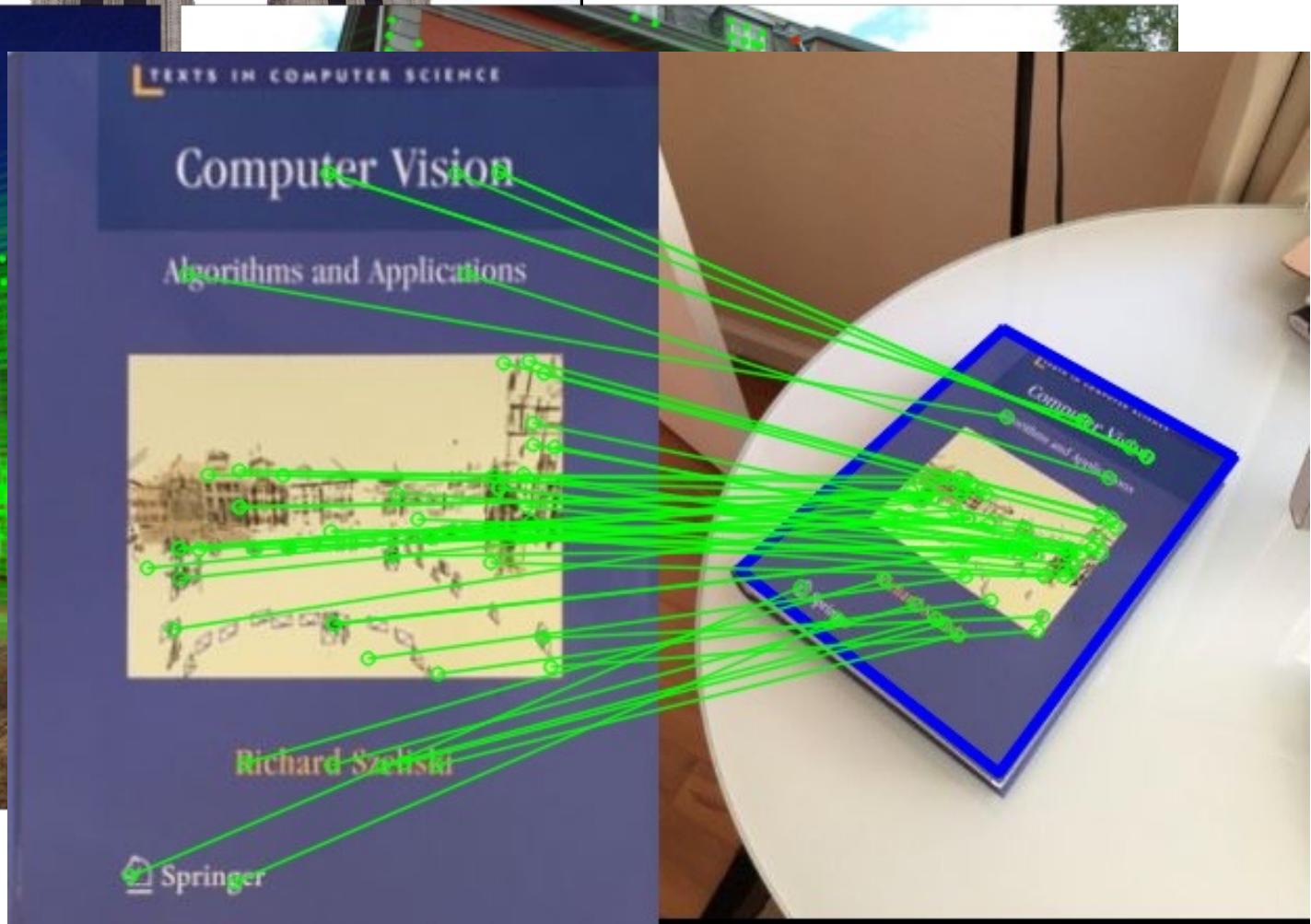
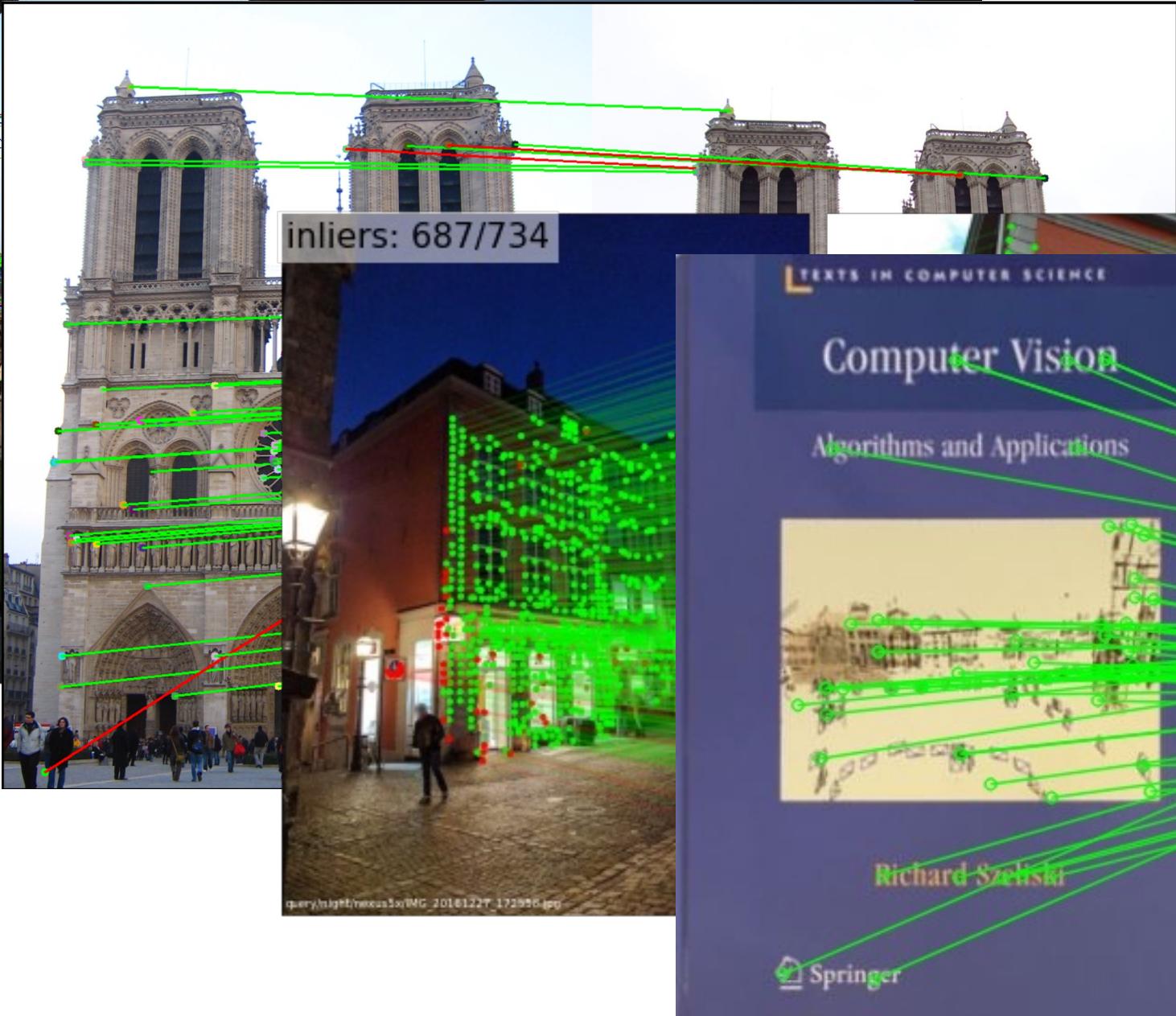
## Rotation Invariance:

- The feature vector uses gradient orientations. So if you rotate the image, everything changes!
- SIFT adopts “**relative rotation**”: the keypoint’s own rotation is subtracted from each orientation. Thus each gradient orientation is relative to the keypoint’s orientation.

## Illumination Invariance:

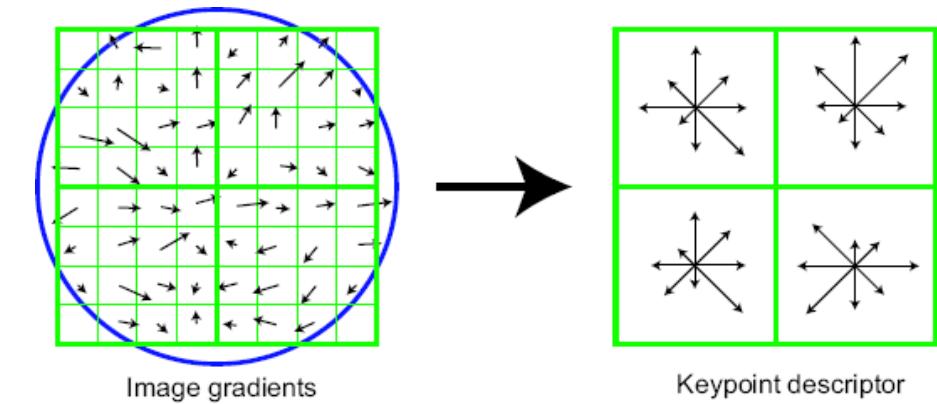
- All keypoints’ 128-dim vector are normalized to 1
- Sometimes we have “outlier illumination” ...
  - Practically, after normalization, we clamp all gradients  $> 0.2$ , then renormalize to  $[0,1]$

*SIFT achieves an extremely elegant and robust balance between **global layout** (histogram) versus **local feature** (full gradient), discriminativeness versus resilience*



# Review: Local Descriptors

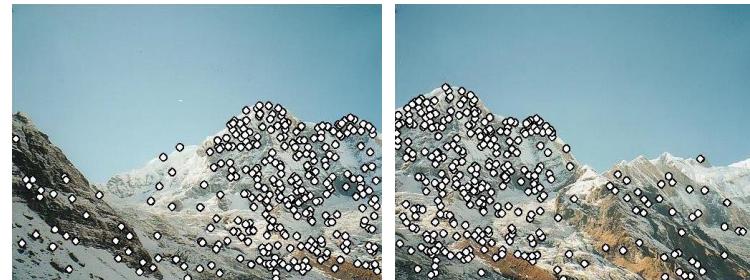
- Most features can be thought of as templates, histograms (counts), or combinations
- The ideal descriptor should be
  - Robust and Distinctive
  - Compact and Efficient
- Most available descriptors focus on edge/gradient information
  - Capture texture information
  - Color rarely used



# Local features: main components

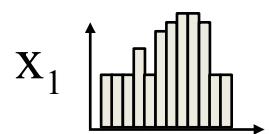
## 1) Detection:

Find a set of distinctive key points.

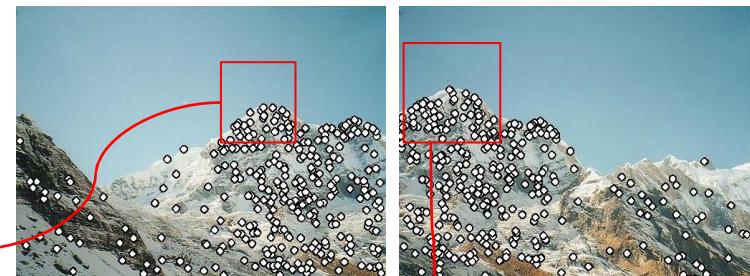


## 2) Description:

Extract feature descriptor around each interest point as vector.



$$\mathbf{x}_1 = [x_1^{(1)}, \dots, x_d^{(1)}]$$



$$\mathbf{x}_2 = [x_1^{(2)}, \dots, x_d^{(2)}]$$

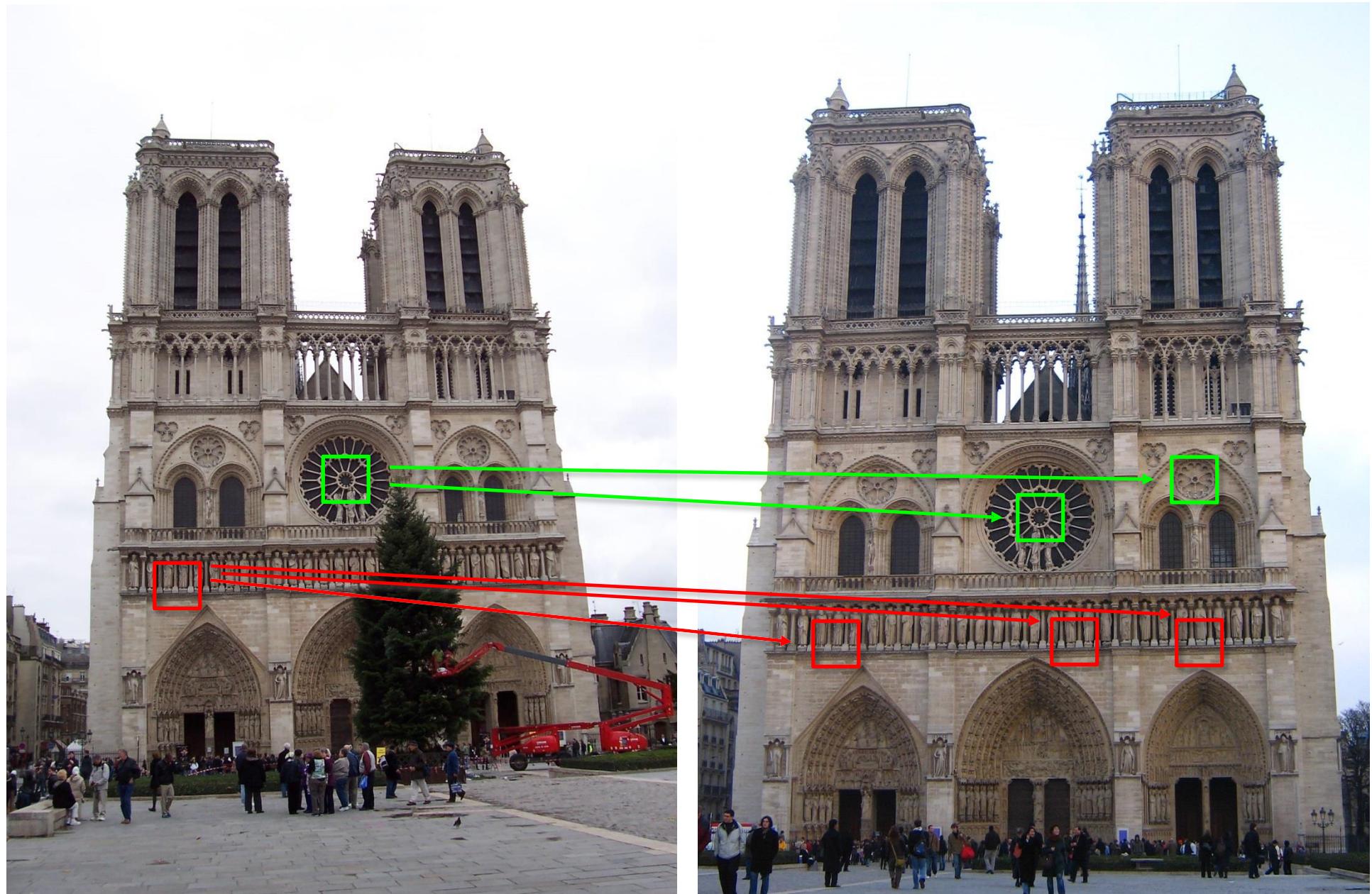
## 3) Matching:

Compute distance between feature vectors to find correspondence.



Ok, now we have local features...

But how similar can the two features be called “match”?

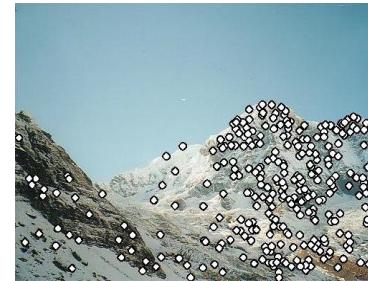


Distance: 0.34, 0.30, 0.40

Distance: 0.61, 1.22

# What to Consider in the Design of Feature Matching

- Two images,  $I_1$  and  $I_2$
- Two sets  $X_1$  and  $X_2$  of feature points
  - Each feature point  $x_1$  has a descriptor  $\mathbf{x}_1 = [x_1^{(1)}, \dots, x_d^{(1)}]$
- Distance, bijective/injective/surjective, noise, confidence, computational complexity, generality...



# Euclidean distance vs. Cosine Similarity

- Euclidean distance:

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

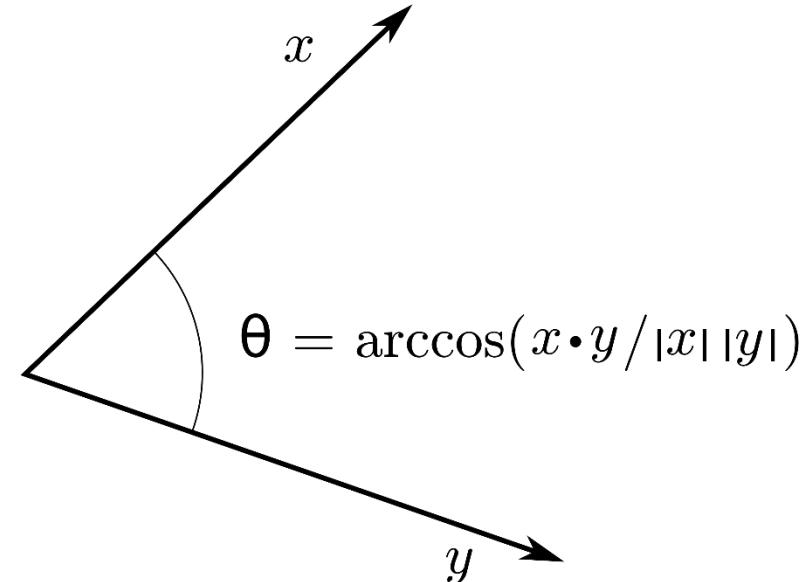
$$= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}.$$

$$\|\mathbf{q} - \mathbf{p}\| = \sqrt{(\mathbf{q} - \mathbf{p}) \cdot (\mathbf{q} - \mathbf{p})}.$$

- Cosine similarity:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\|_2 \|\mathbf{b}\|_2 \cos \theta$$

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|_2 \|\mathbf{B}\|_2}$$



# Feature Matching

- Criteria 1:
  - Compute distance in feature space, e.g., Euclidean distance between 128-dim SIFT descriptors
  - Match point to lowest distance (nearest neighbor)
- Problems:
  - Does everything have a match?

# Feature Matching

- Criteria 2:
  - Compute distance in feature space, e.g., Euclidean distance between 128-dim SIFT descriptors
  - Match point to lowest distance (nearest neighbor)
  - Ignore anything higher than threshold (no match!)
- Problems:
  - Threshold is hard to pick
  - Non-distinctive features could have lots of close matches, only one of which is correct

# Nearest Neighbor Distance Ratio

*Compare distance of closest (NN1) and second-closest (NN2) feature vector neighbor.*

- If  $NN1 \approx NN2$ , ratio  $\frac{NN1}{NN2}$  will be  $\approx 1 \rightarrow$  matches too close.
- As  $NN1 \ll NN2$ , ratio  $\frac{NN1}{NN2}$  tends to 0.

Sorting by this ratio puts matches in order of confidence.

Threshold ratio – but how to choose?

- depends on your application's view on the trade-off between the number of false positives and true positives! **You need to tune...**

# Visual Similarity is still/forever an OPEN problem



"While they're similar,  
same and similar don't  
mean the same thing."

-Robert Lee Brewer



The University of Texas at Austin  
**Electrical and Computer  
Engineering**  
*Cockrell School of Engineering*