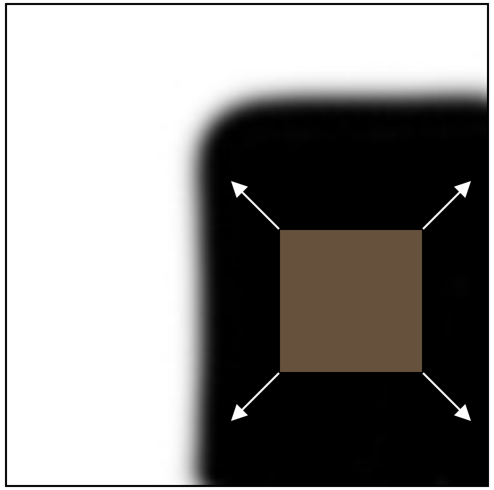
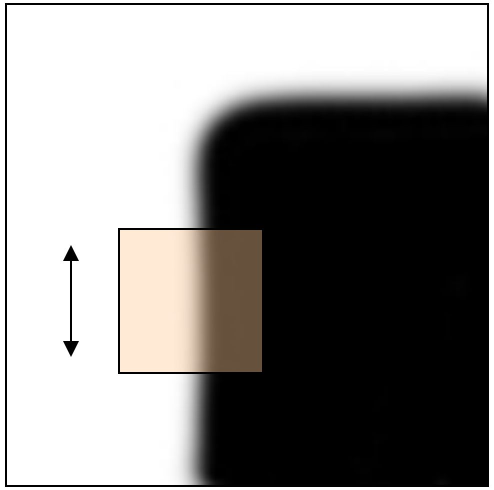
Correspondence

Corner Detection: Basic Idea

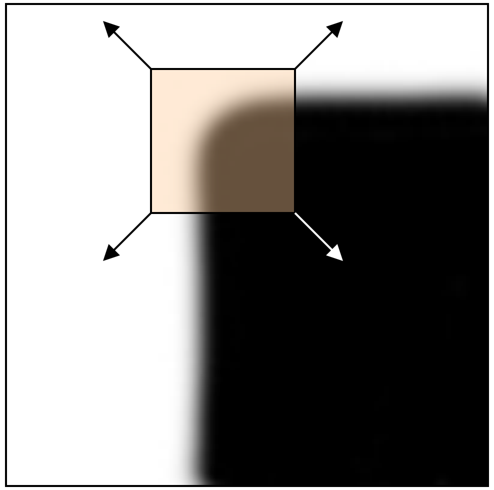
- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



"flat" region:
no change in all directions



"edge":
no change along the edge direction



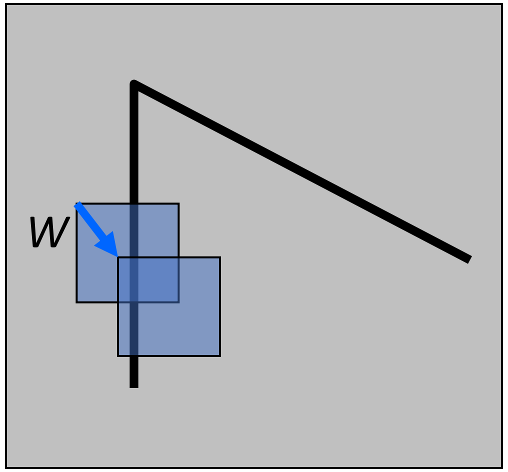
"corner":
significant change in all directions

Source: A. Efros

Corner detection: the math

Consider shifting the window *W* by (*u,v*)

- how do the pixels in W change?
- compare each pixel before and after by summing up the squared differences (SSD)
- this defines an SSD "error" *E*(*u,v*):



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

- We want E(u,v) to be *as high as possible for all u, v!*

Small motion assumption

Taylor Series expansion of *I*:

$$I(x+u, y+v) = I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \text{higher order terms}$$

If the motion (u,v) is small, then first order approximation is good

$$\begin{aligned} I(x+u, y+v) &\approx I(x, y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v \\ &\approx I(x, y) + [I_x \ I_y] \begin{bmatrix} u \\ v \end{bmatrix} \end{aligned}$$

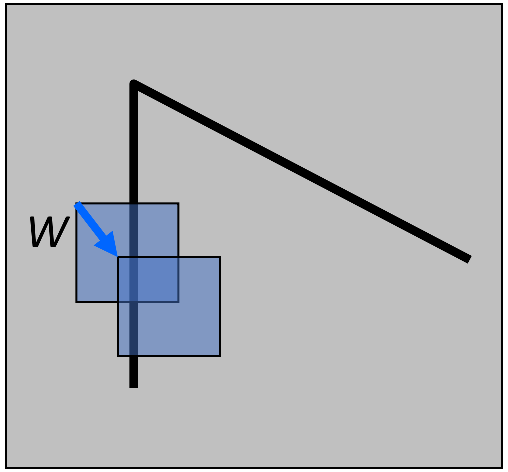
shorthand: $I_x = \frac{\partial I}{\partial x}$

Plugging this into the formula on the previous slide…

Corner detection: the math

Consider shifting the window *W* by (*u,v*)

- define an SSD "error" *E(u,v)*:

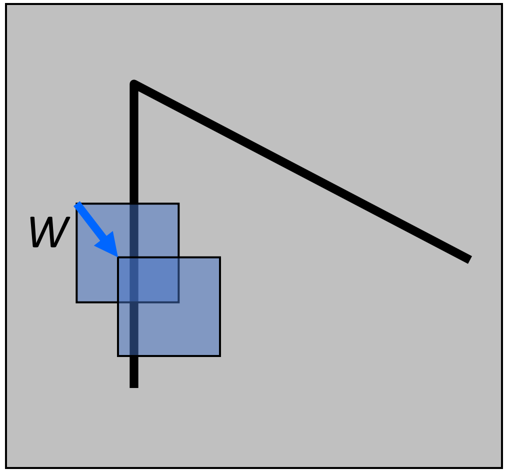


$$
\begin{aligned}
E(u,v) &= \sum_{(x,y)\in W} \left[I(x+u,y+v) - I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I(x,y) + I_x u + I_y v - I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I_x u + I_y v\right]^2
\end{aligned}
$$

Corner detection: the math

Consider shifting the window *W* by (*u,v*)

- define an "error" *E(u,v)*:



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

$$\approx Au^2 + 2Buv + Cv^2$$

$$A = \sum_{(x,y)\in W} I_x^2 \qquad B = \sum_{(x,y)\in W} I_x I_y \qquad C = \sum_{(x,y)\in W} I_y^2$$

- Thus, *E*(*u*,*v*) is locally approximated as a quadratic error function

Interpreting the second moment matrix

Recall that we want E(u,v) to be as large as possible for all u,v

What does this mean in terms of M?

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

$$\underbrace{M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}}_{\text{Second moment matrix}}$$

$$E(u,v) \approx \left[\begin{array}{cc} u & v \end{array}\right] M \left[\begin{array}{c} u \\ v \end{array}\right]$$

$$M \left[\begin{array}{c} u \\ v \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \end{array}\right] \Rightarrow E(u,v) = 0$$

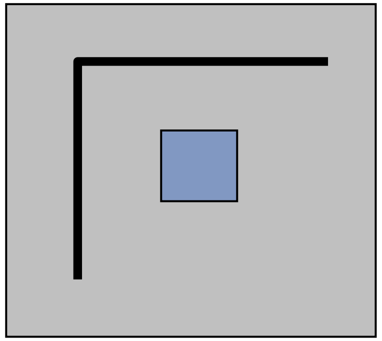
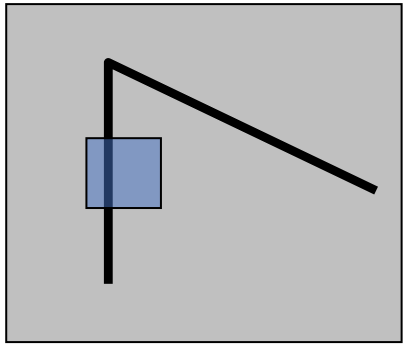
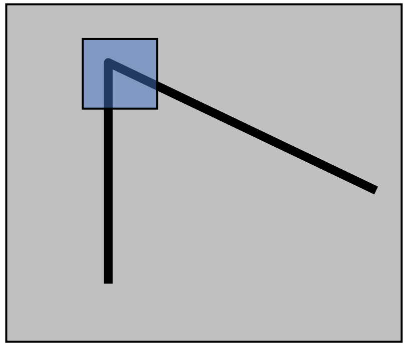
$$M \left[\begin{array}{c} u \\ v \end{array}\right] = \left[\begin{array}{c} 0 \\ 0 \end{array}\right] \Leftrightarrow E(u,v) = 0$$

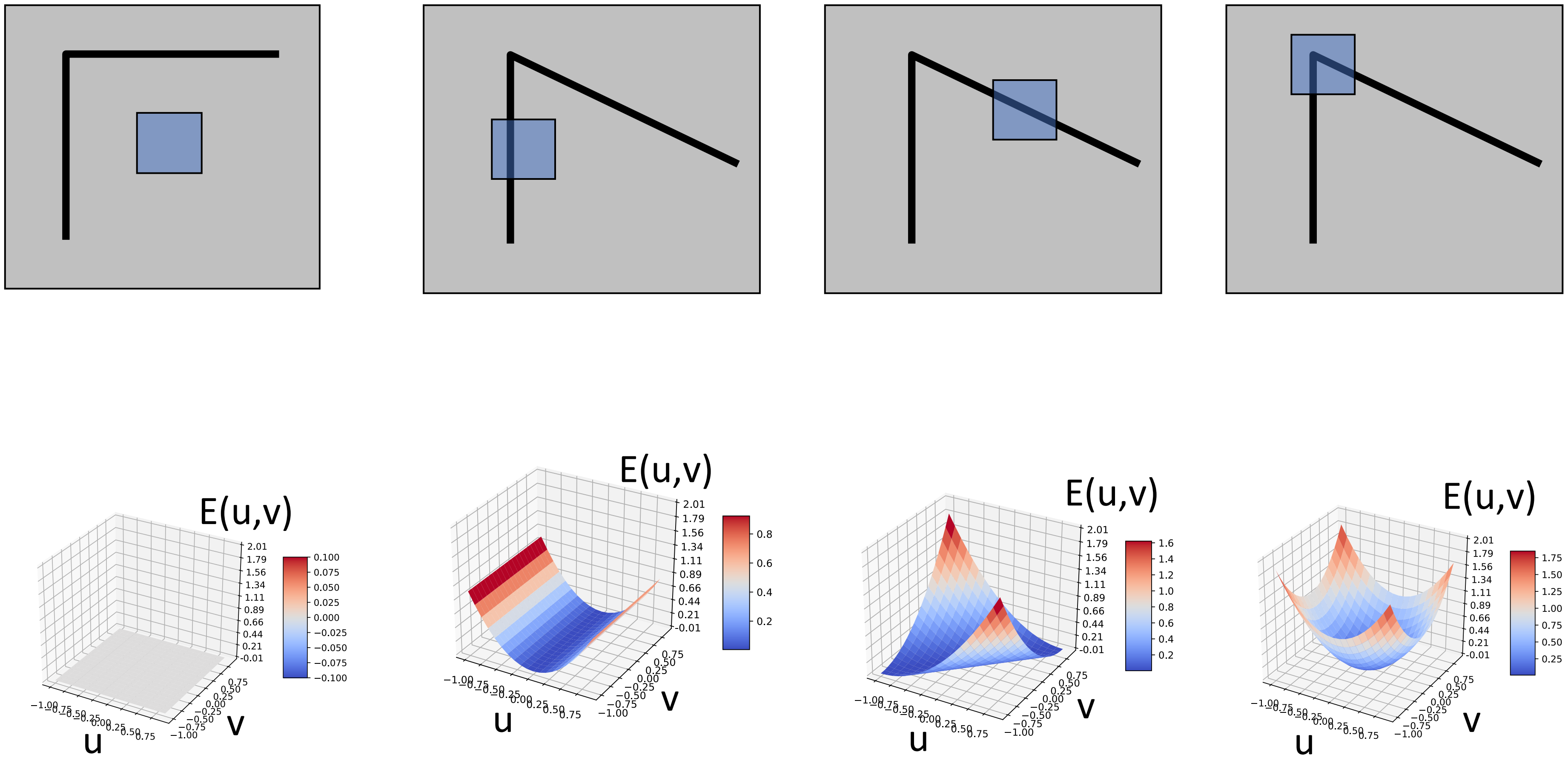
Solutions to Mx = 0 are directions for which E is 0: window can slide in this direction without changing appearance

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

Solutions to Mx = 0 are directions for which E is 0: window can slide in this direction without changing appearance

For corners, we want no such directions to exist

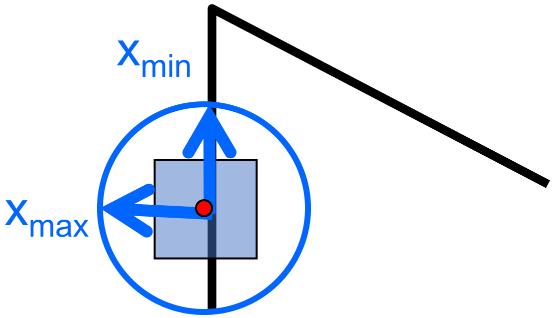

E(u,v)
u
v
E(u,v)
u
v
E(u,v)
u
v
E(u,v)
u
v

Eigenvalues and eigenvectors of M

- $Mx = 0 \Rightarrow Mx = \lambda x$: x is an eigenvector of M with eigenvalue 0
- M is 2 x 2, so it has 2 eigenvalues $(\lambda_{max}, \lambda_{min})$ with eigenvectors (x_{max}, x_{min})
- $E(x_{max}) = \ x_{max}^T M x_{max} = \lambda_{max}||x_{max}||^2 = \lambda_{max}$ (eigenvectors have unit norm)
- $E(x_{min}) = x_{min}^T M x_{min} = \lambda_{min}||x_{min}||^2 = \lambda_{min}$

Eigenvalues and eigenvectors of M

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



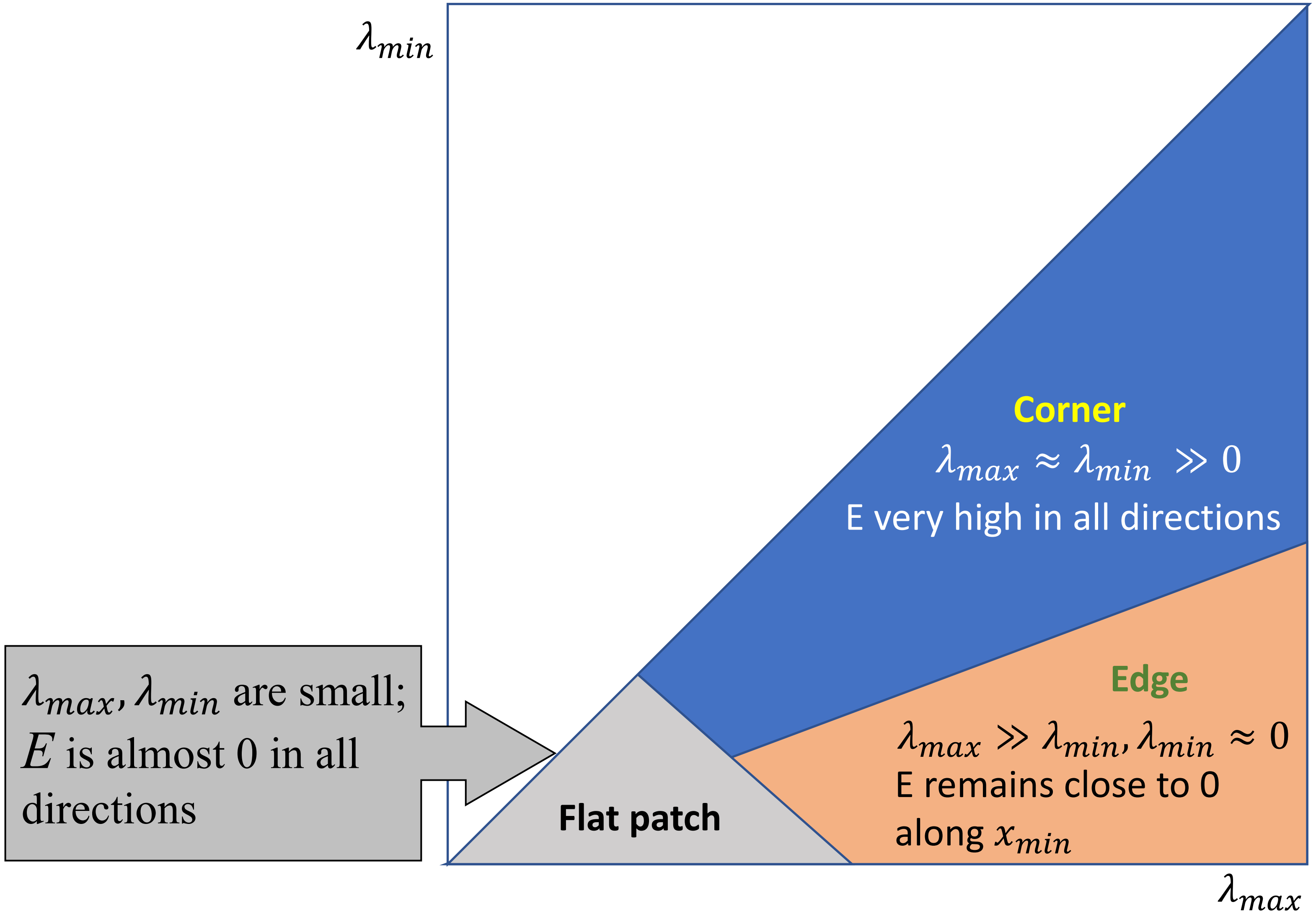
$$M\, x_{\max} = \lambda_{\max} x_{\max}$$

$$M\, x_{\min} = \lambda_{\min} x_{\min}$$

Eigenvalues and eigenvectors of M

- Define shift directions with the smallest and largest change in error
- x_{max} = direction of largest increase in *E*
- λ_{max} = amount of increase in direction x_{max}
- x_{min} = direction of smallest increase in *E*
- λ_{min} = amount of increase in direction x_{min}

Interpreting the eigenvalues



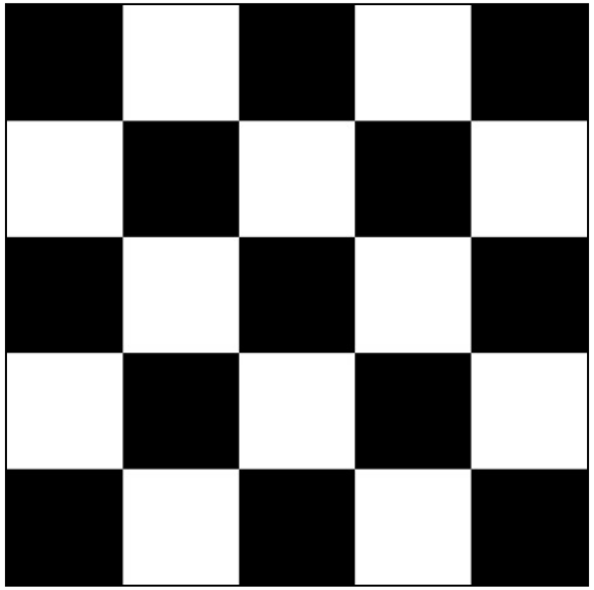
Corner detection: the math

How are λ_{max}, x_{max}, λ_{min}, and x_{min} relevant for feature detection?

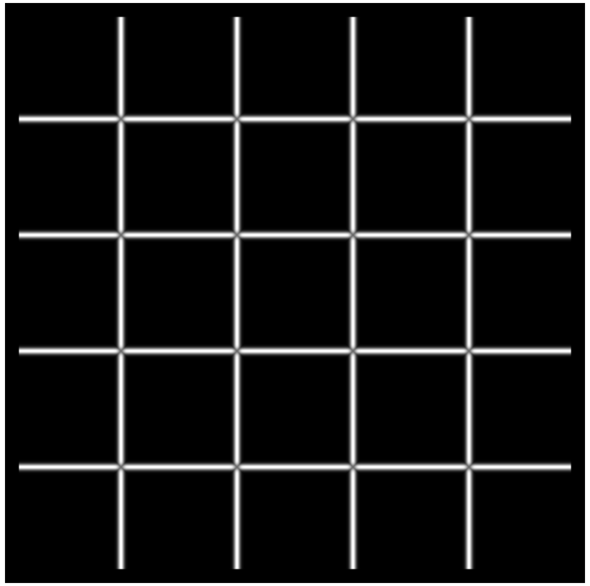
- Need a feature scoring function

Want $E(u,v)$ to be large for small shifts in all directions

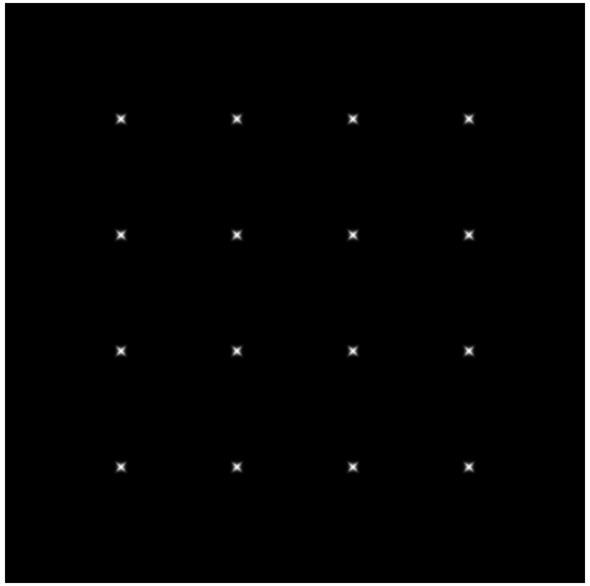
- the minimum of $E(u,v)$ should be large, over all unit vectors $[u\ v]$
- this minimum is given by the smaller eigenvalue (λ_{min}) of M



I



$\lambda_{\max}$

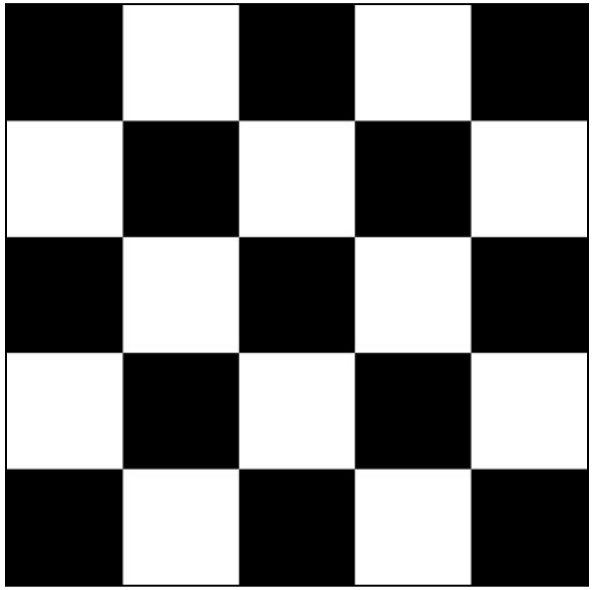


$\lambda_{\min}$

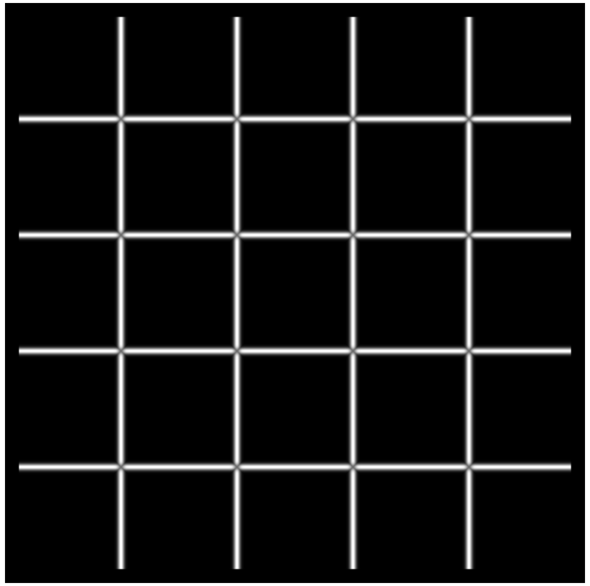
Corner detection summary

Here's what you do

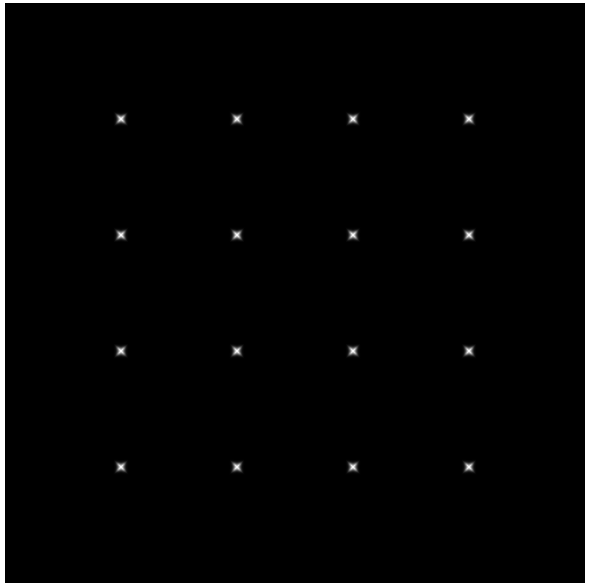
- Compute the gradient at each point in the image
- Create the *M* matrix from the entries in the gradient
- Compute the eigenvalues
- Find points with large response (λ_{min} > threshold)
- Choose those points where λ_{min} is a local maximum as features



I



$\lambda_{\max}$

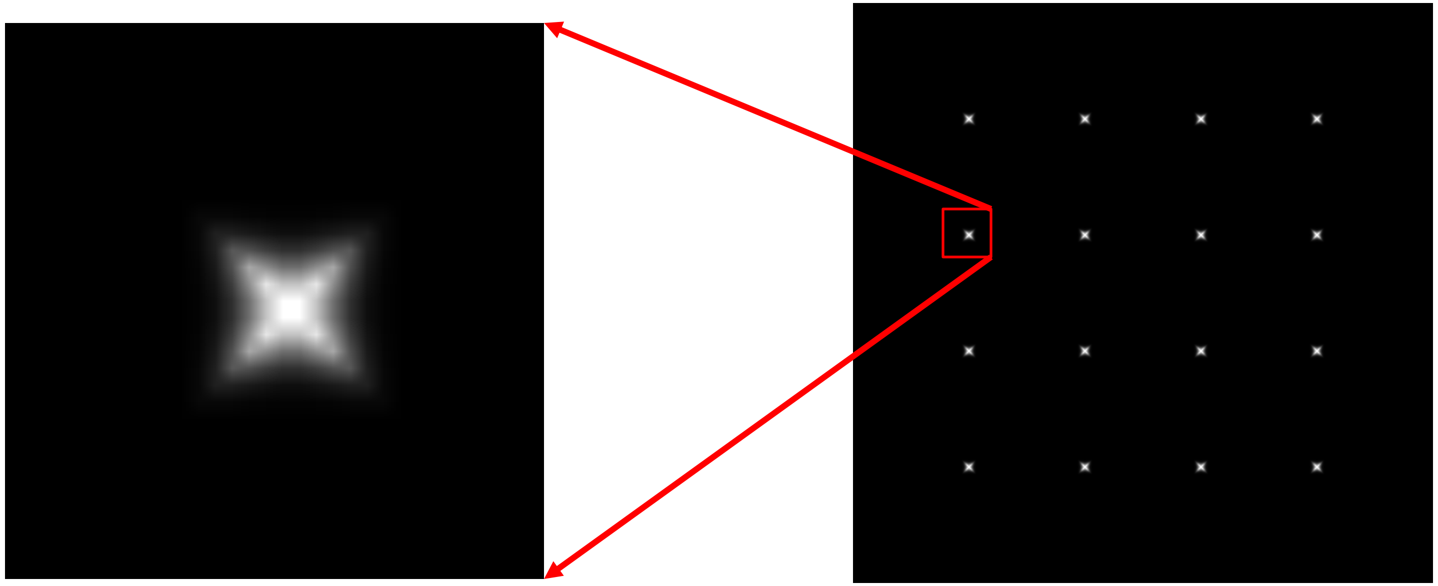


$\lambda_{\min}$

Corner detection summary

Here's what you do

- Compute the gradient at each point in the image
- Create the *H* matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response (λ_{min} > threshold)
- Choose those points where λ_{min} is a local maximum as features



$\lambda_{\min}$

The Harris operator

λ_{min} is a variant of the "Harris operator" for feature detection

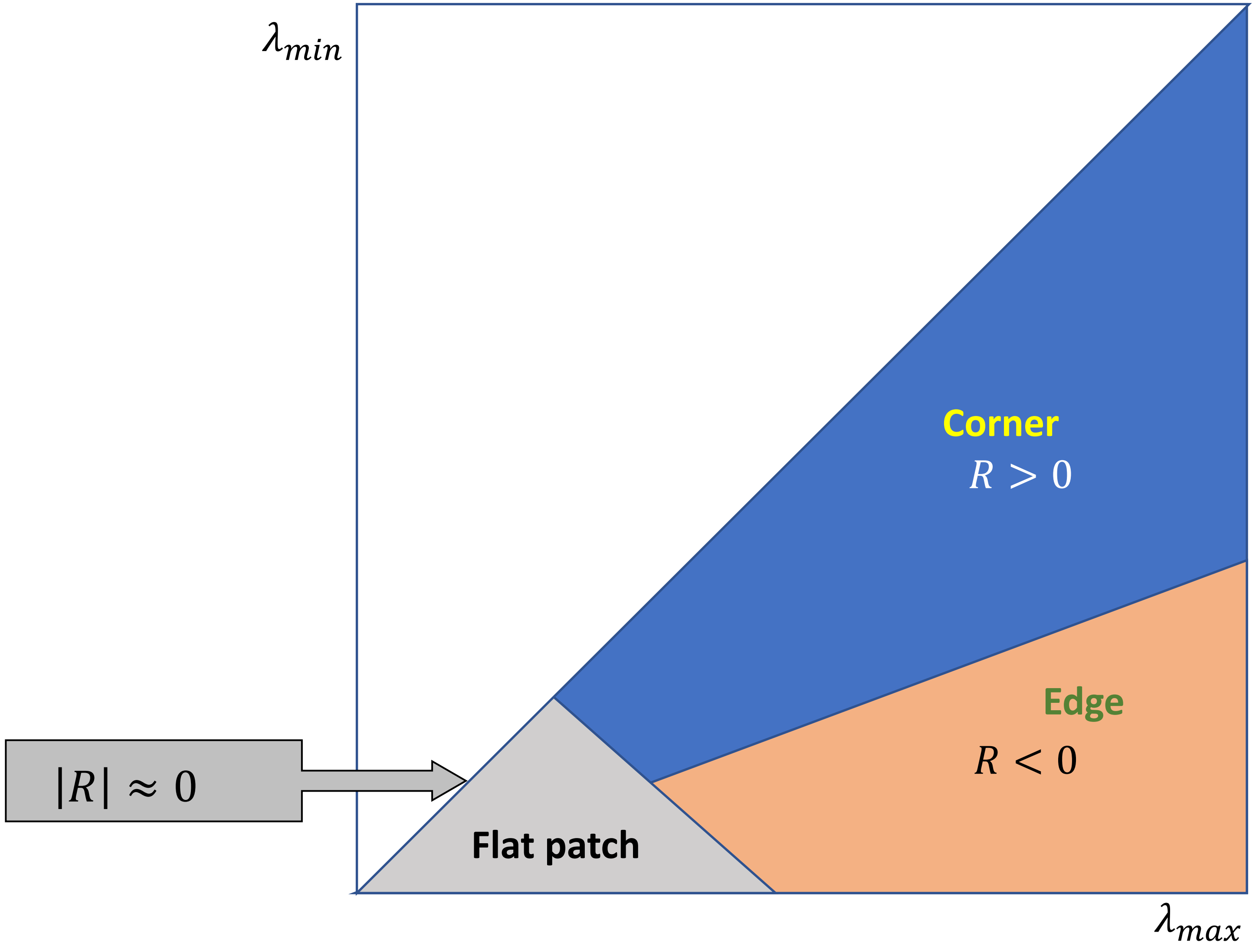
$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$

$$= \frac{determinant(H)}{trace(H)}$$

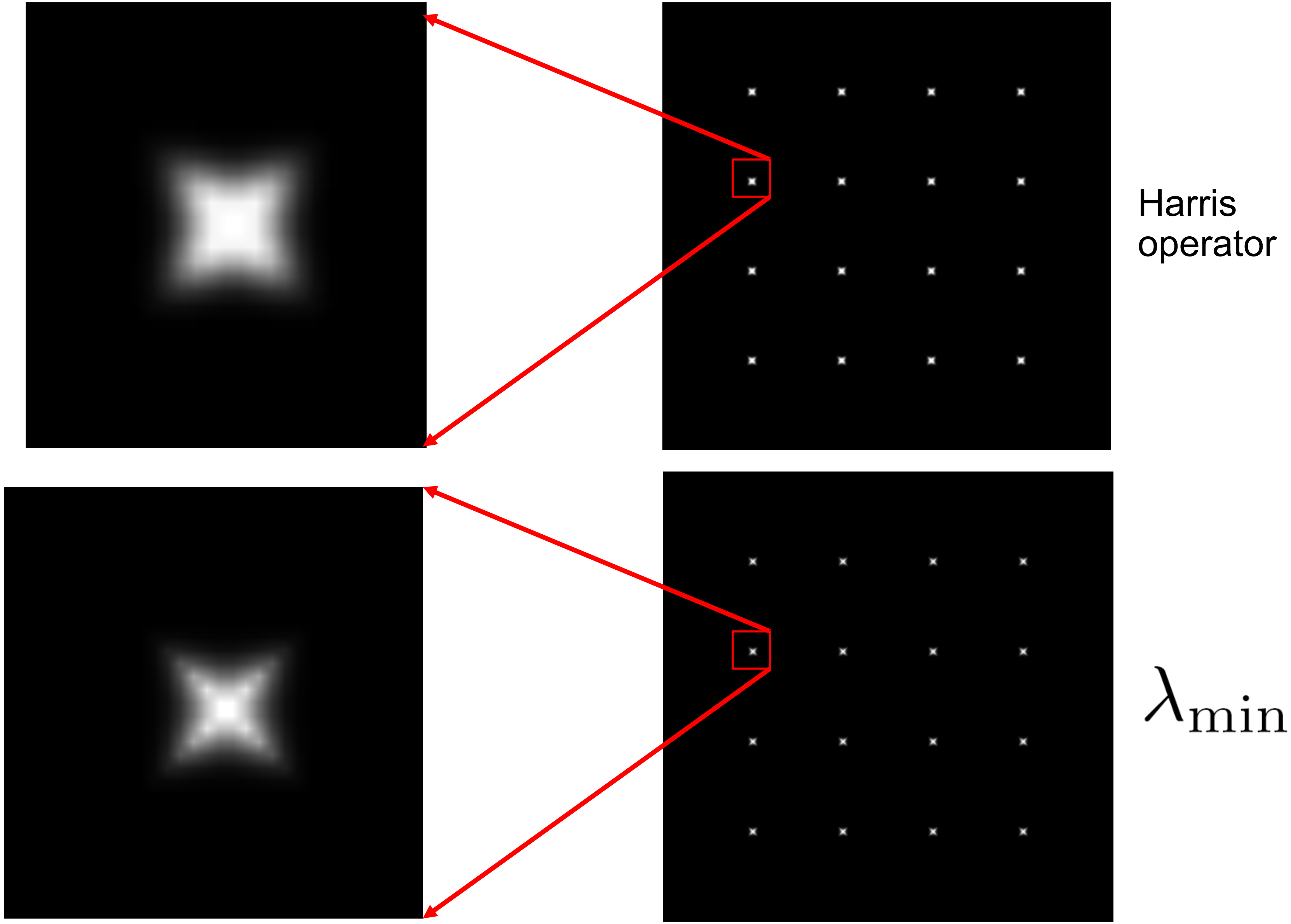
- The *trace* is the sum of the diagonals, i.e., $trace(H) = h_{11} + h_{22}$
- Very similar to λ_{min} but less expensive (no square root)
- Called the "Harris Corner Detector" or "Harris Operator"
    - Actually the Noble variant of the Harris Corner Detector
- Lots of other detectors, this is one of the most popular

Corner response function

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

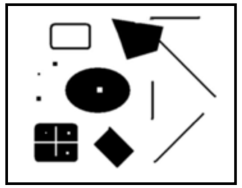


The Harris operator



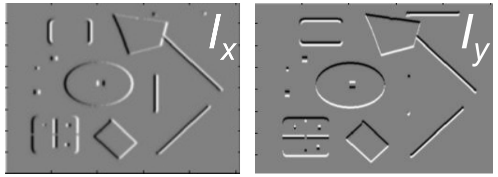
Harris Detector [Harris88]

- Second moment matrix

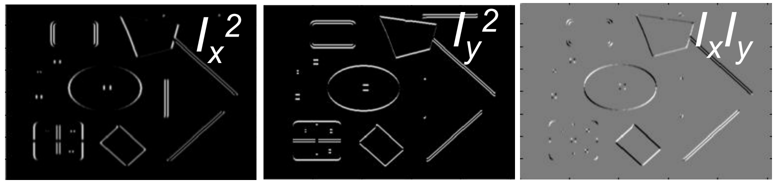


$$\mu(\sigma_I,\sigma_D)=g(\sigma_I)*\begin{bmatrix} I_x^2(\sigma_D) & I_xI_y(\sigma_D) \\ I_xI_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

1. Image derivatives (optionally, blur first)



2. Square of derivatives

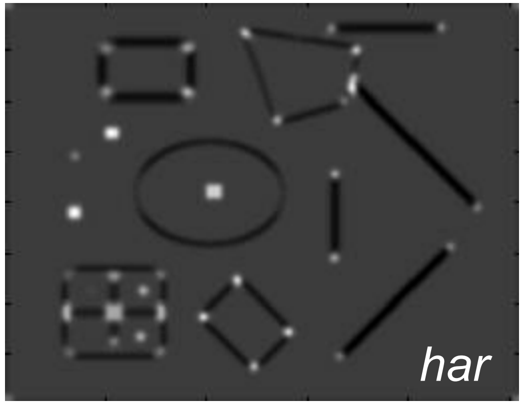


$$\det M = \lambda_1\lambda_2$$
$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

3. Gaussian filter $g(\sigma_I)$

4. Cornerness function – both eigenvalues are strong

$$har = \det[\mu(\sigma_I,\sigma_D)] - \alpha[\operatorname{trace}(\mu(\sigma_I,\sigma_D))^2] =$$
$$g(I_x^2)g(I_y^2) - [g(I_xI_y)]^2 - \alpha[g(I_x^2)+g(I_y^2)]^2$$



5. Non-maxima suppression

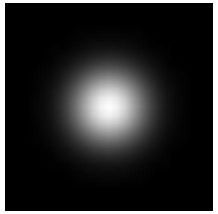
Weighting the derivatives

- In practice, using a simple window *W* doesn't work too well

$$H = \sum_{(x,y)\in W} \left[\begin{array}{cc} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{array} \right]$$

- Instead, we'll *weigh* d on its distance from the center pixel

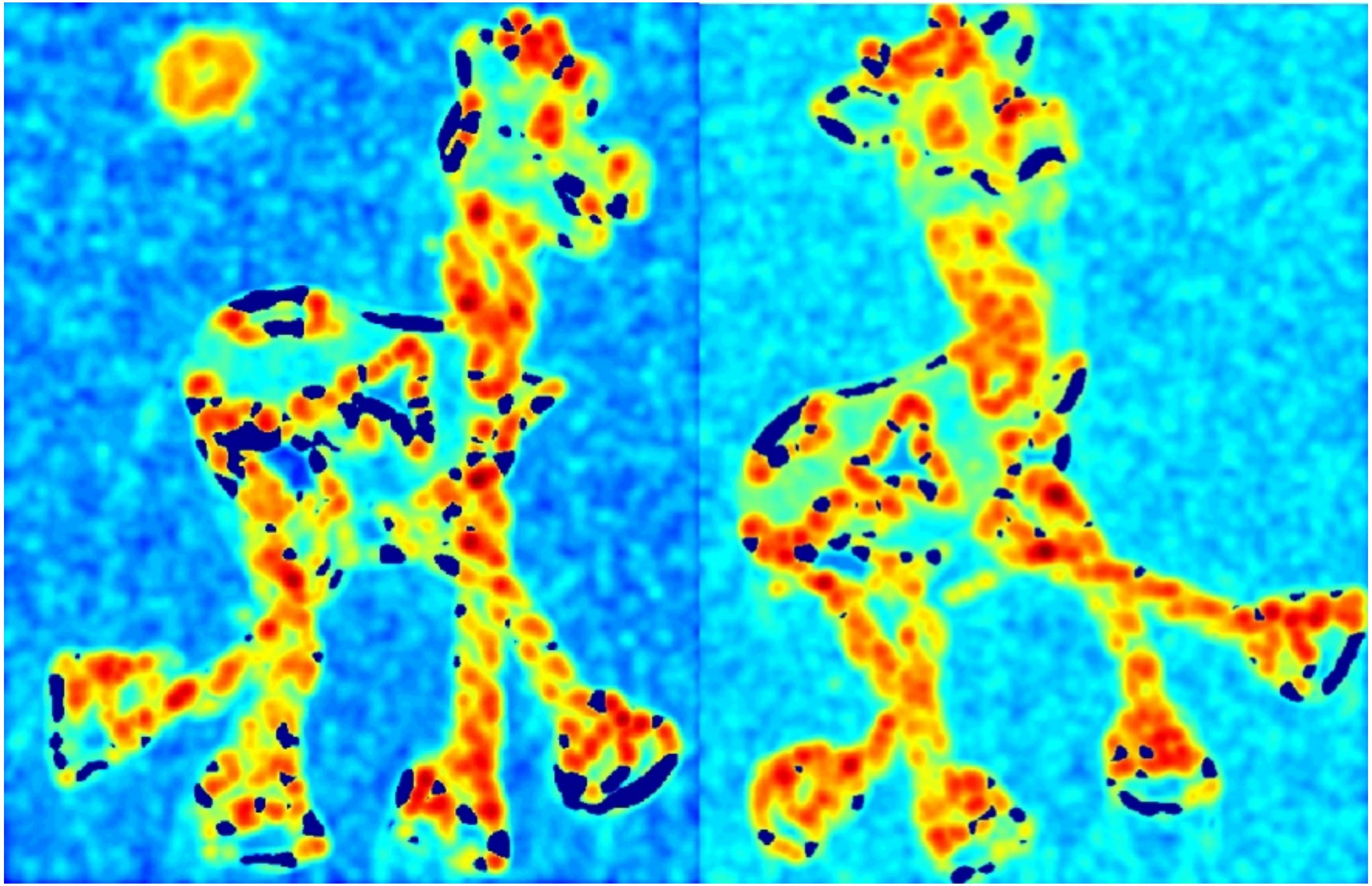
$$H = \sum_{(x,y)\in W} w_{x,y} \left[\begin{array}{cc} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{array} \right]$$



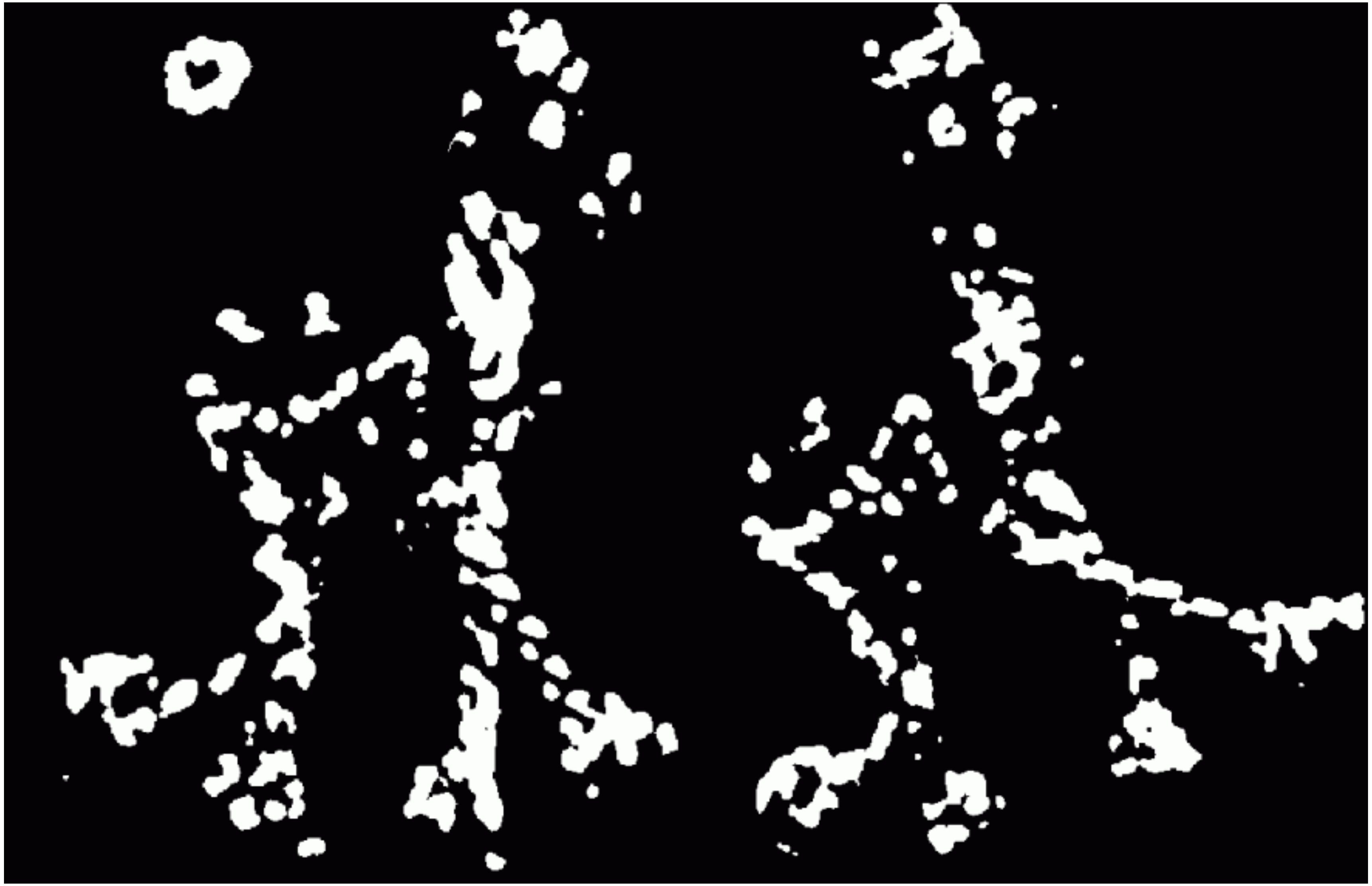
$w_{x,y}$

Harris detector example

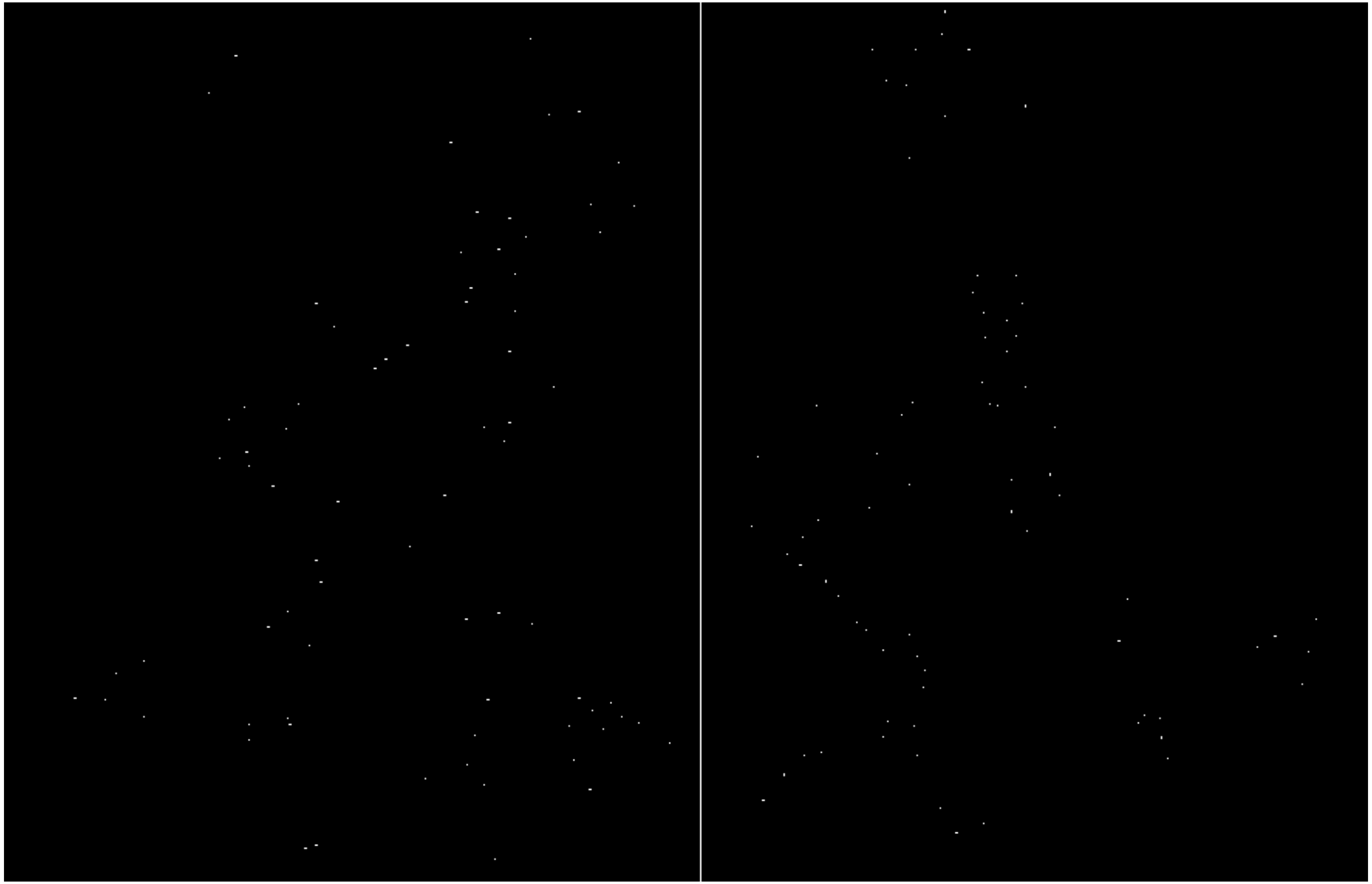
f value (red high, blue low)



Threshold (f > value)



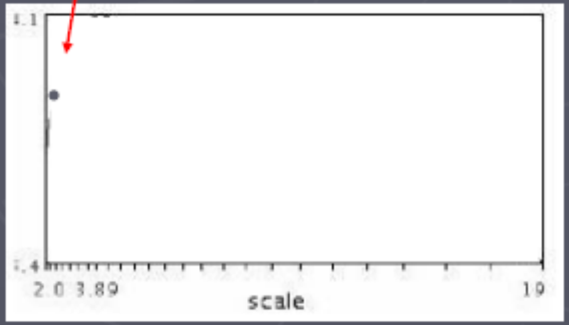
Find local maxima of f



Harris features (in red)

Automatic scale selection

Lindeberg et al., 1996

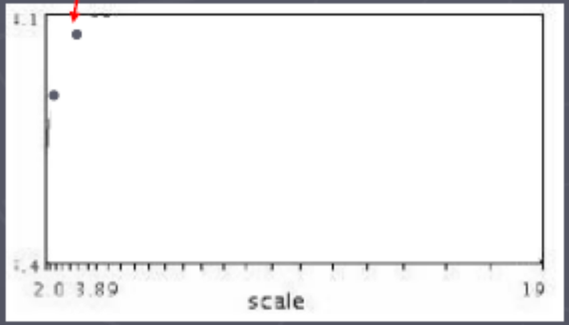


$f(I_{i_1 \ldots i_m}(x,\sigma))$

Slide from Tinne Tuytelaars

Automatic scale selection

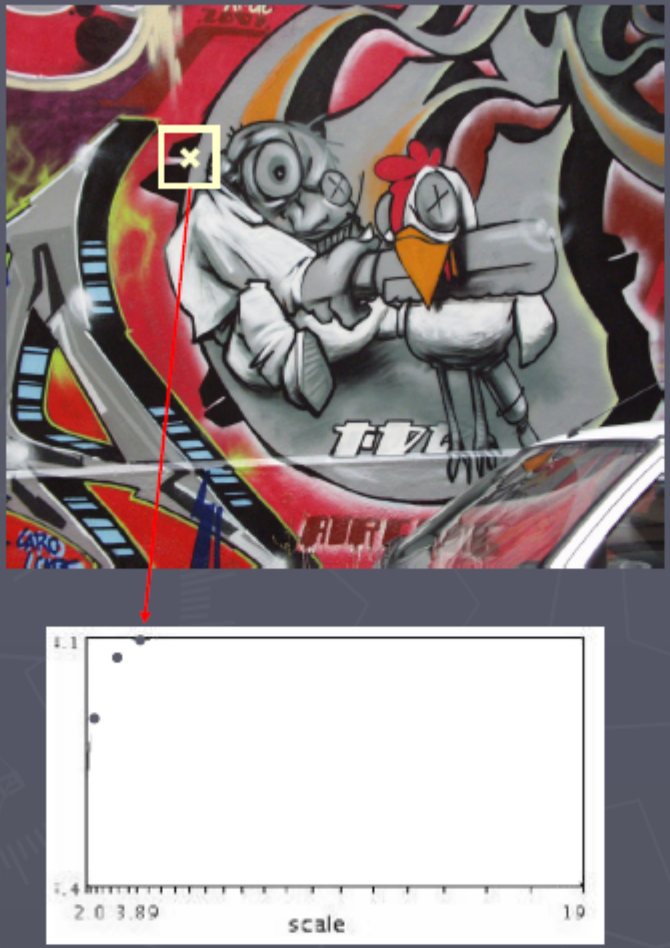
Function responses for increasing scale
Scale trace (signature)



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

Automatic scale selection

Function responses for increasing scale
Scale trace (signature)

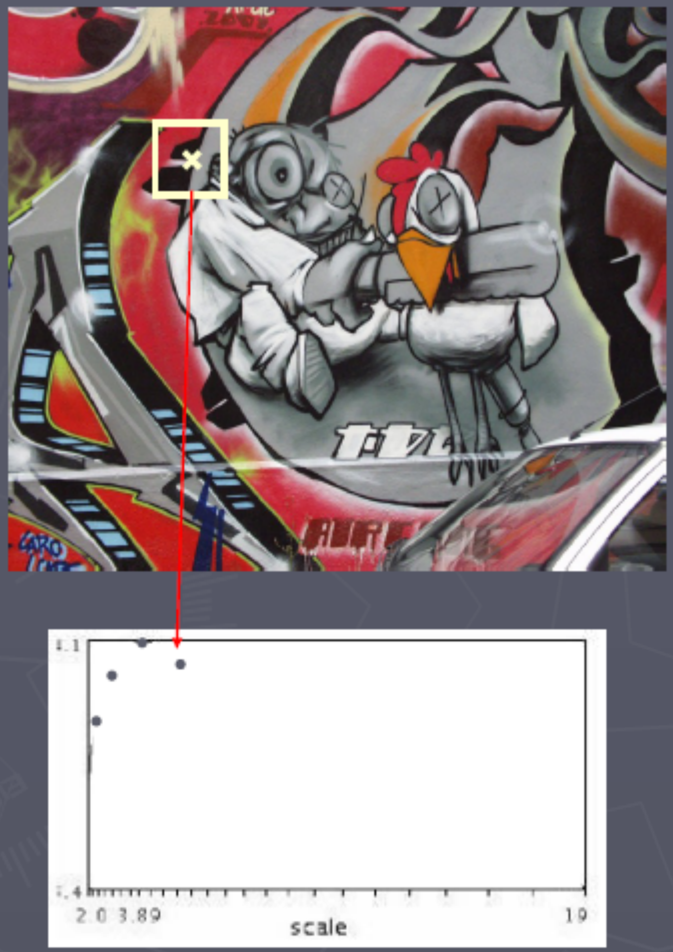


$f(I_{i_1 \ldots i_m}(x,\sigma))$

Automatic scale selection

Function responses for increasing scale

Scale trace (signature)

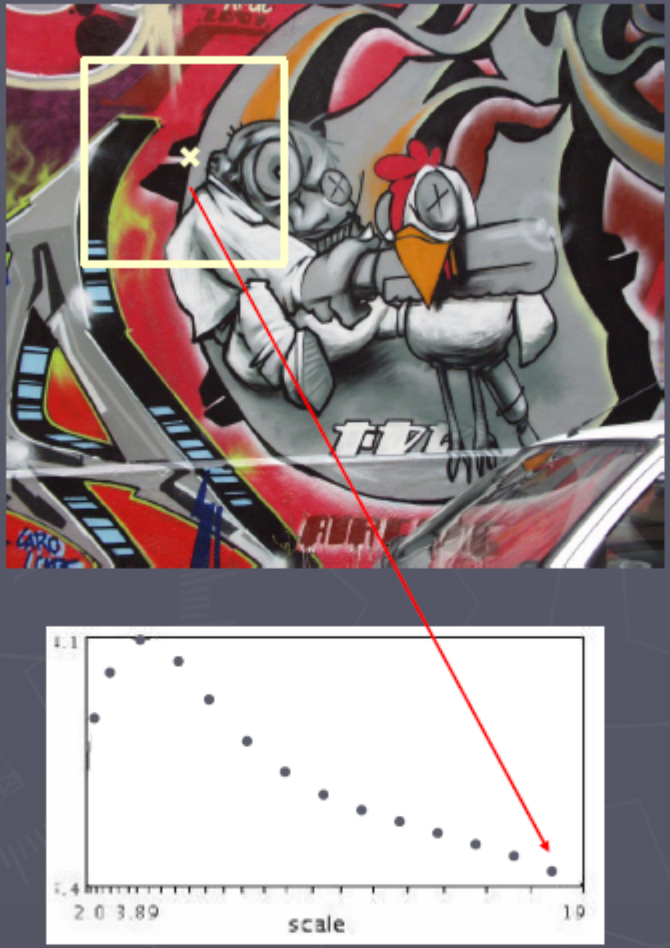


$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection

Function responses for increasing scale

Scale trace (signature)

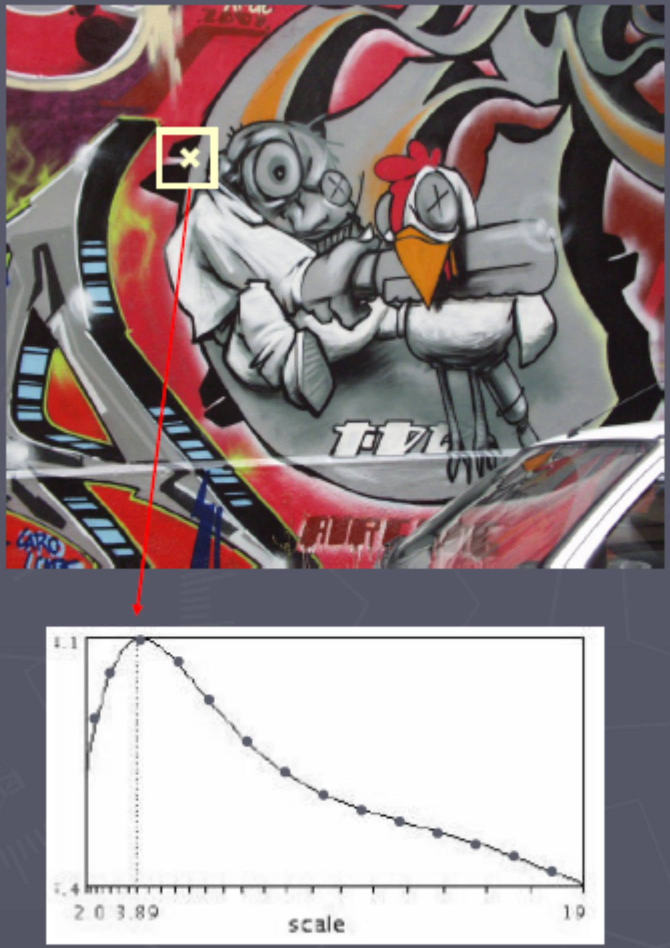


$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection

Function responses for increasing scale

Scale trace (signature)

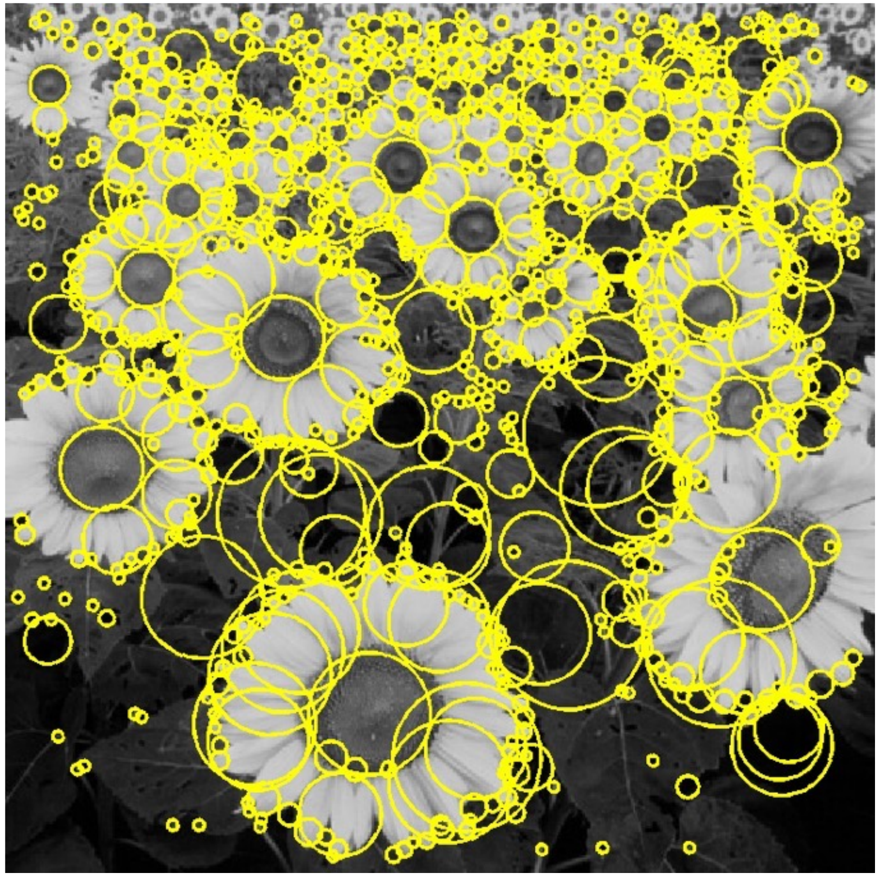


$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Implementation

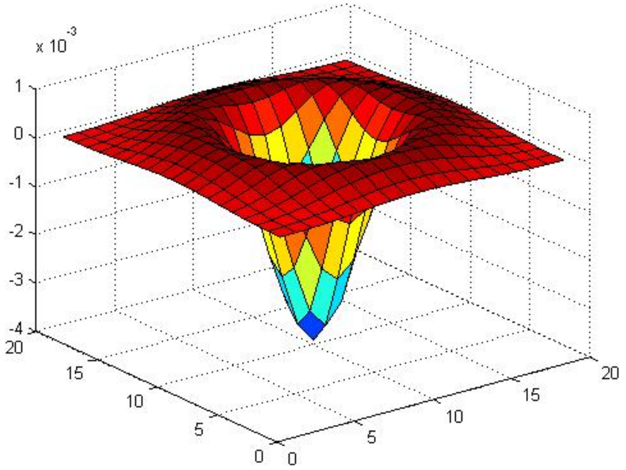
- Instead of computing *f* for larger and larger windows, we can implement using a fixed window size with a Gaussian pyramid

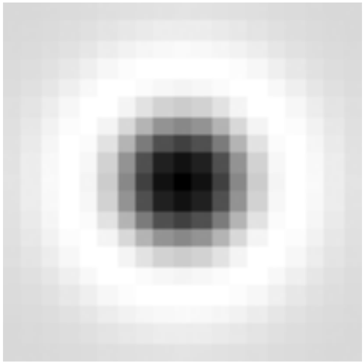
Feature extraction: Corners and blobs



Another common definition of *f*

- The *Laplacian of Gaussian* (*LoG*)



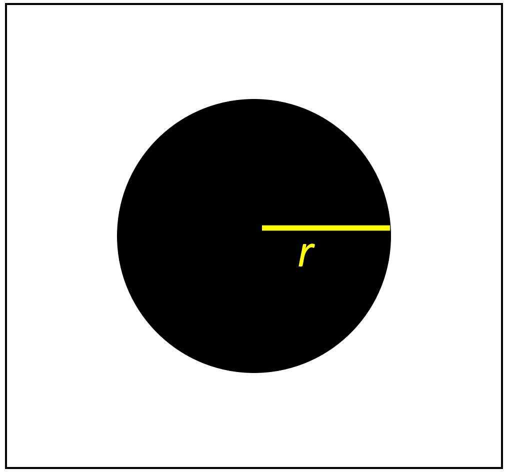


$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

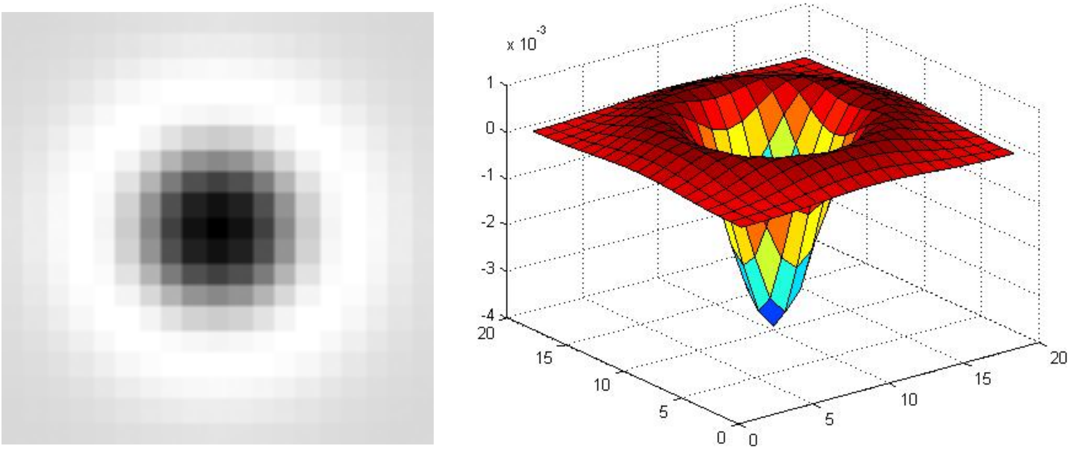
(very similar to a Difference of Gaussians (DoG) – i.e. a Gaussian minus a slightly smaller Gaussian)

Scale selection

- At what scale does the Laplacian achieve a maximum response for a binary circle of radius r?



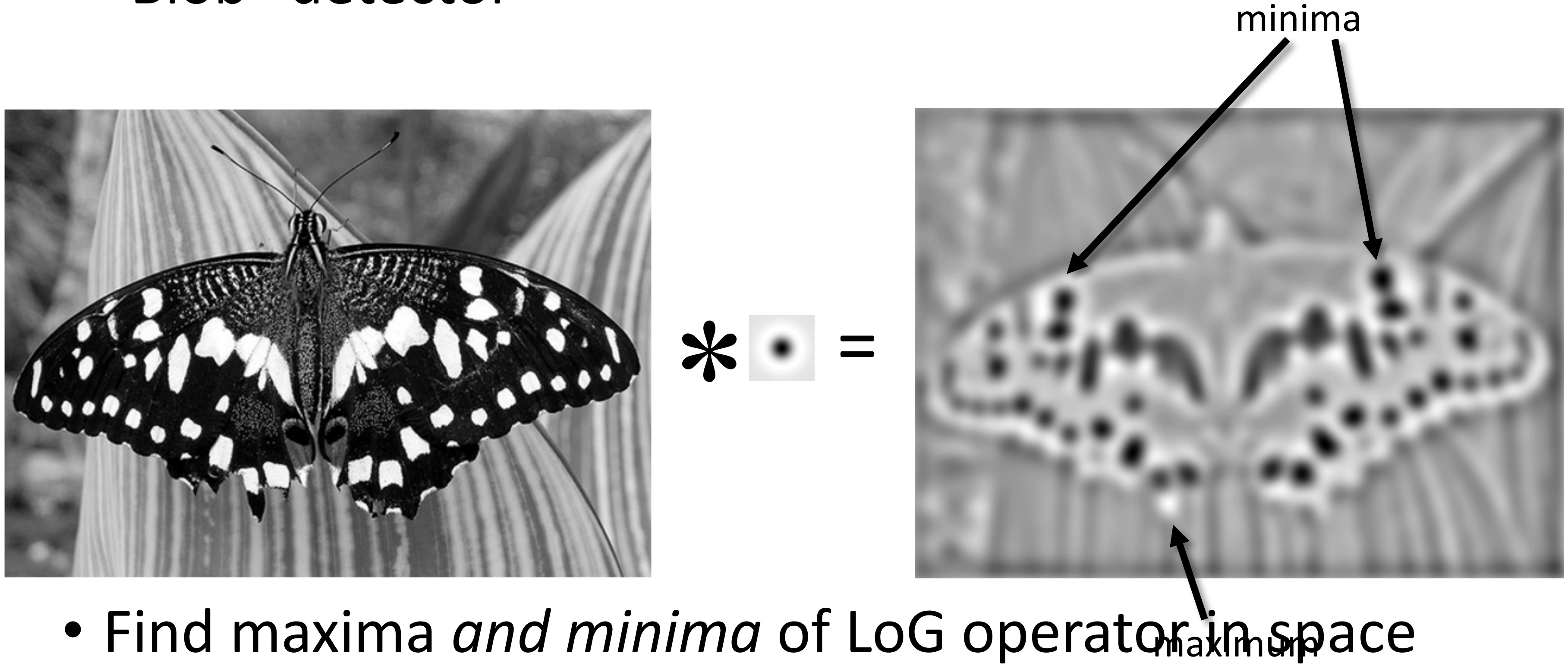
image



Laplacian

Laplacian of Gaussian

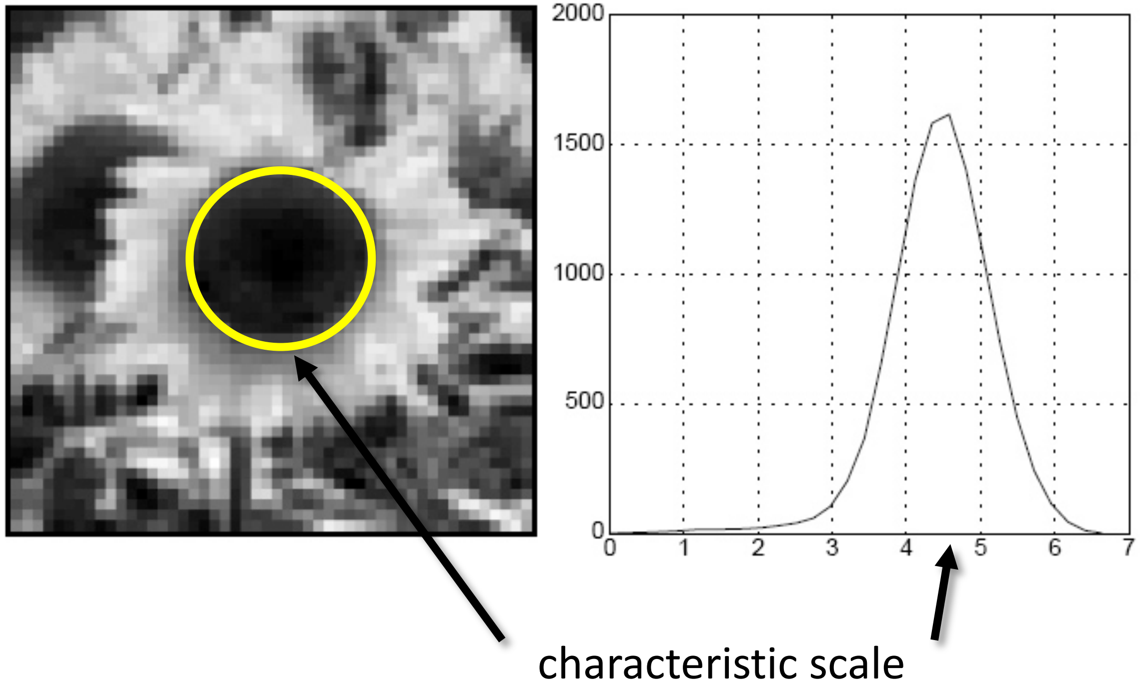
- "Blob" detector



- Find maxima *and minima* of LoG operator in space and scale

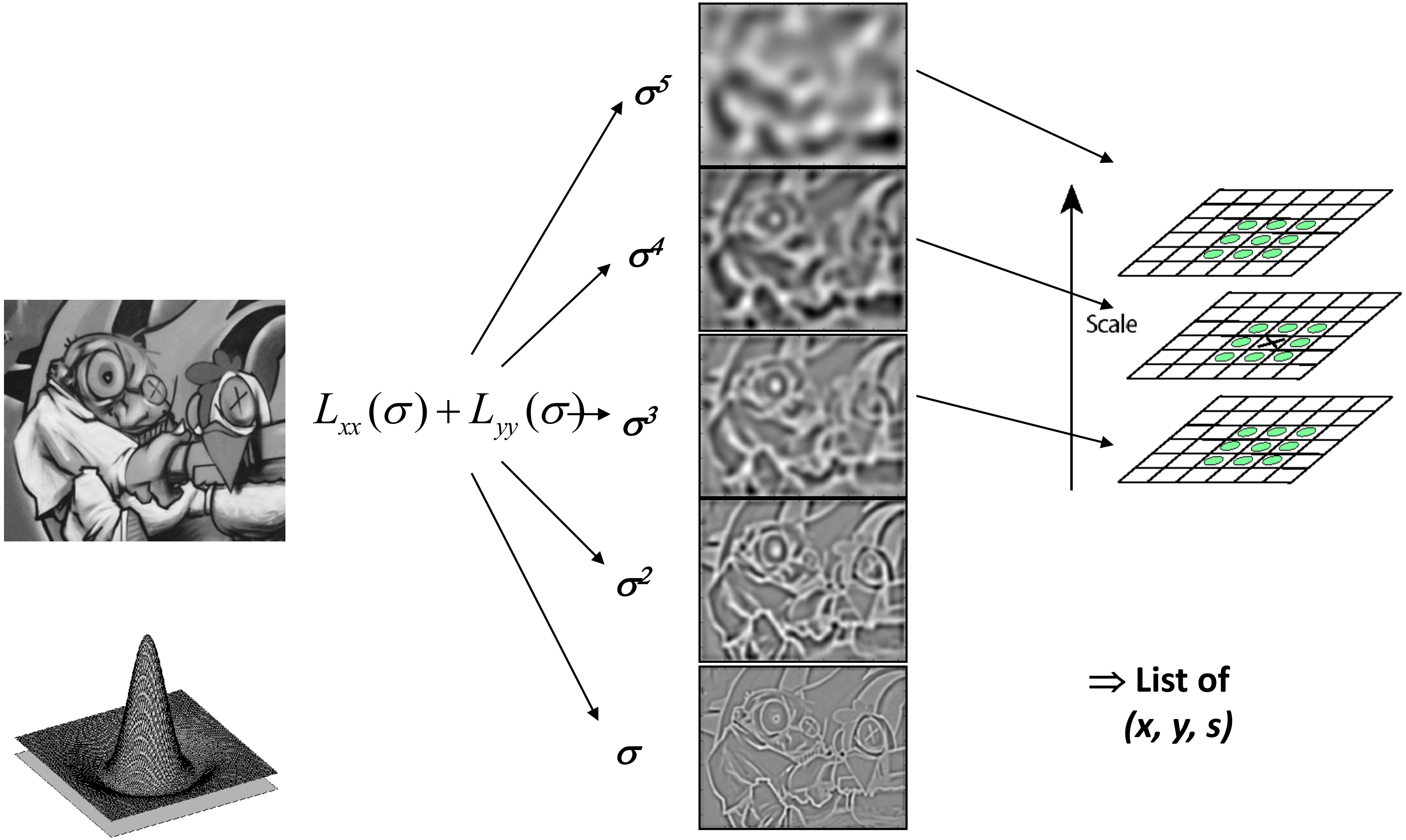
Characteristic scale

- The scale that produces peak of Laplacian response



T. Lindeberg (1998). ["Feature detection with automatic scale selection."]() *International Journal of Computer Vision* **30** (2): pp 77--116.

Find local maxima in position-scale space



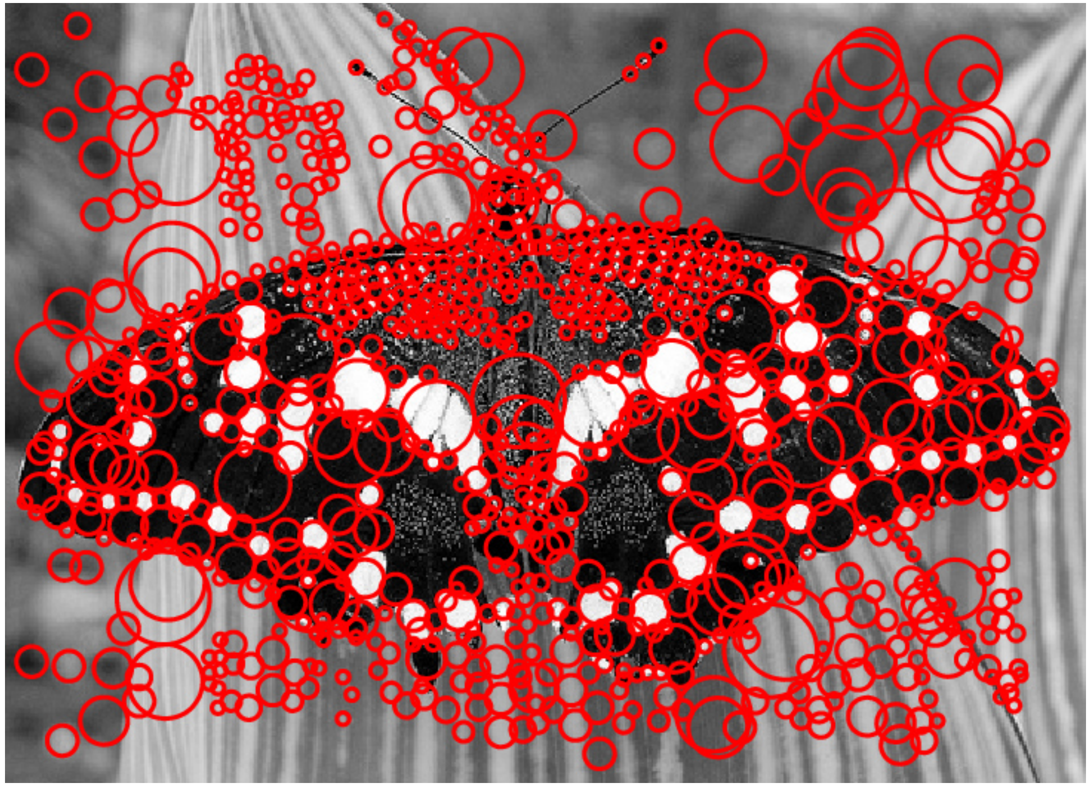
$\Rightarrow$ **List of**
(x, y, s)

Scale-space blob detector: Example

Scale-space blob detector: Example

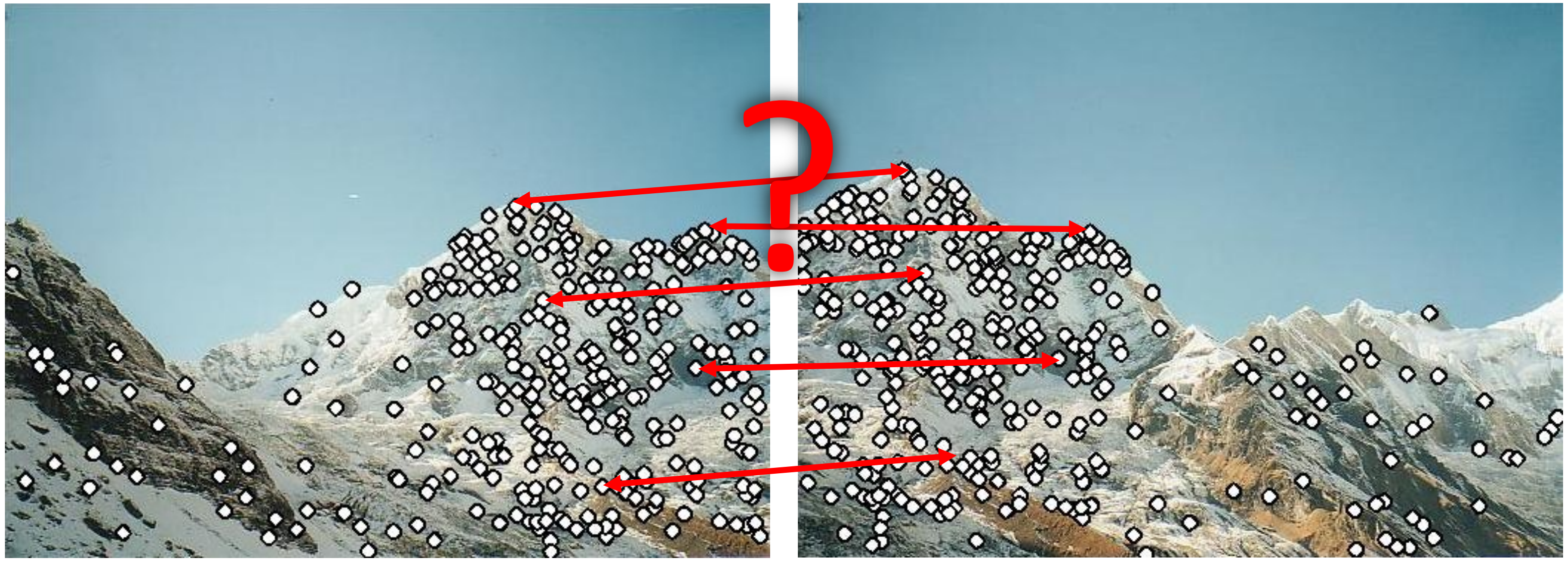
sigma = 11.9912

Scale-space blob detector: Example



Matching feature points

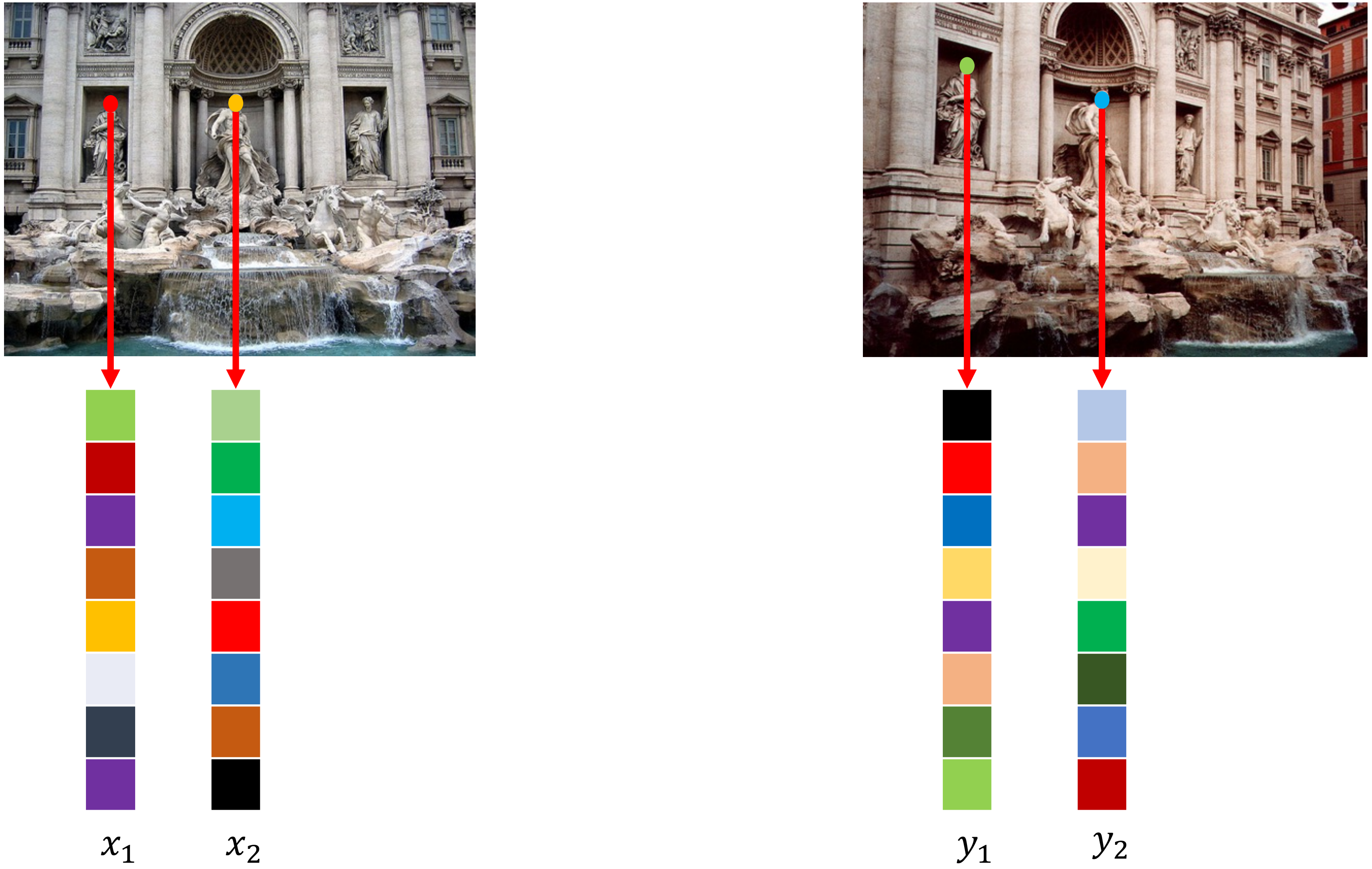
We know how to detect good points
Next question: **How to match them?**



Two interrelated questions:

1. How do we *describe* each feature point?
2. How do we *match* descriptions?

Feature descriptor



Feature matching

- Measure the distance between (or similarity between) every pair of descriptors

	y_1	y_2
x_1	$d(x_1, y_1)$	$d(x_1, y_2)$
x_2	$d(x_2, y_1)$	$d(x_2, y_2)$

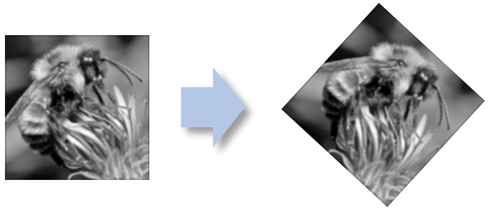
Invariance vs. discriminability

- Invariance:
  - Distance between descriptors should be small even if image is transformed
- Discriminability:
  - Descriptor should be highly unique for each point (far away from other points in the image)

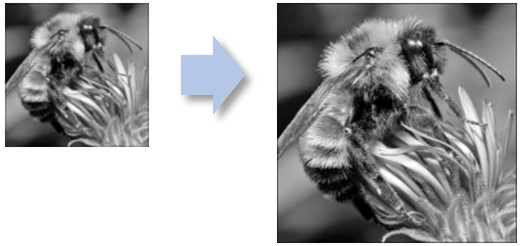
Image transformations

- Geometric

Rotation

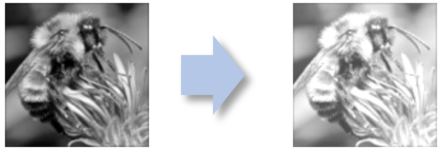


Scale



- Photometric

Intensity change



Invariance

- Most feature descriptors are designed to be invariant to
  - Translation, 2D rotation, scale

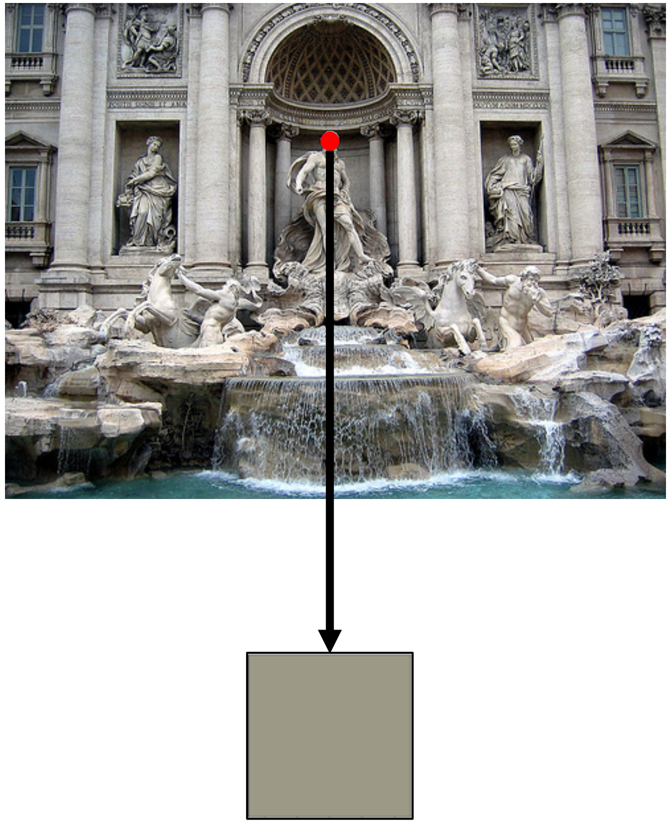
- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transformations (some are fully affine invariant)
  - Limited illumination/contrast changes

How to achieve invariance

Design an invariant feature descriptor

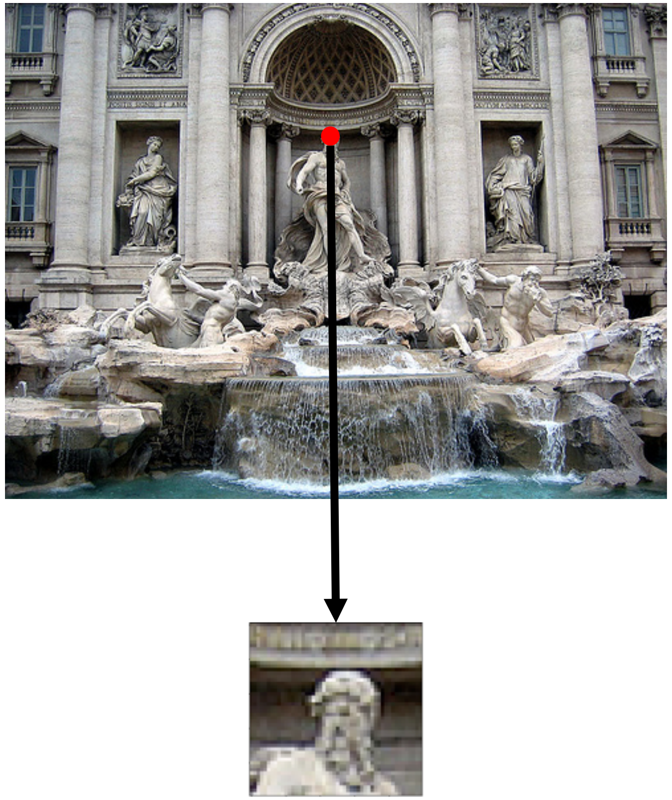
- Simplest descriptor: a single 0
  - What’s this invariant to?
  - Is this discriminative?
- Next simplest descriptor: a single pixel
  - What’s this invariant to?
  - Is this discriminative?

The aperture problem



The aperture problem

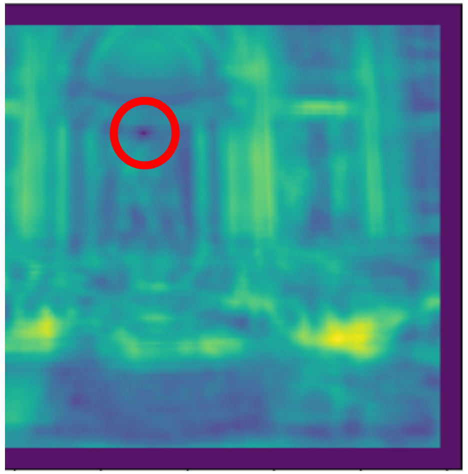
- Use a whole patch instead of a pixel?



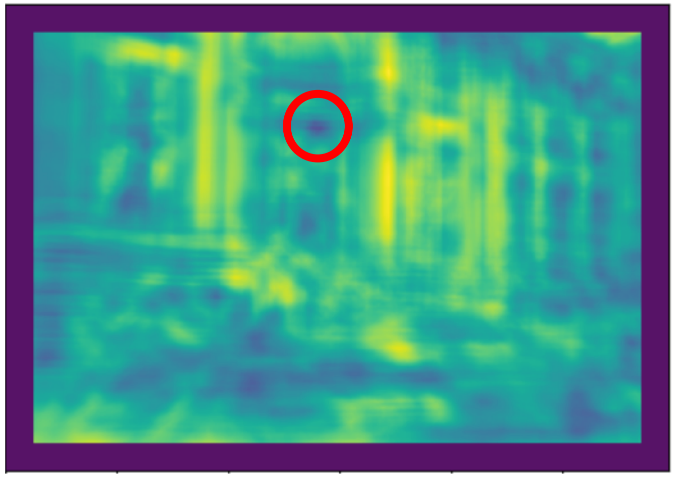
SSD

- Use as descriptor the whole patch
- Match descriptors using euclidean distance
- $d(x, y) = ||x - y||^2$

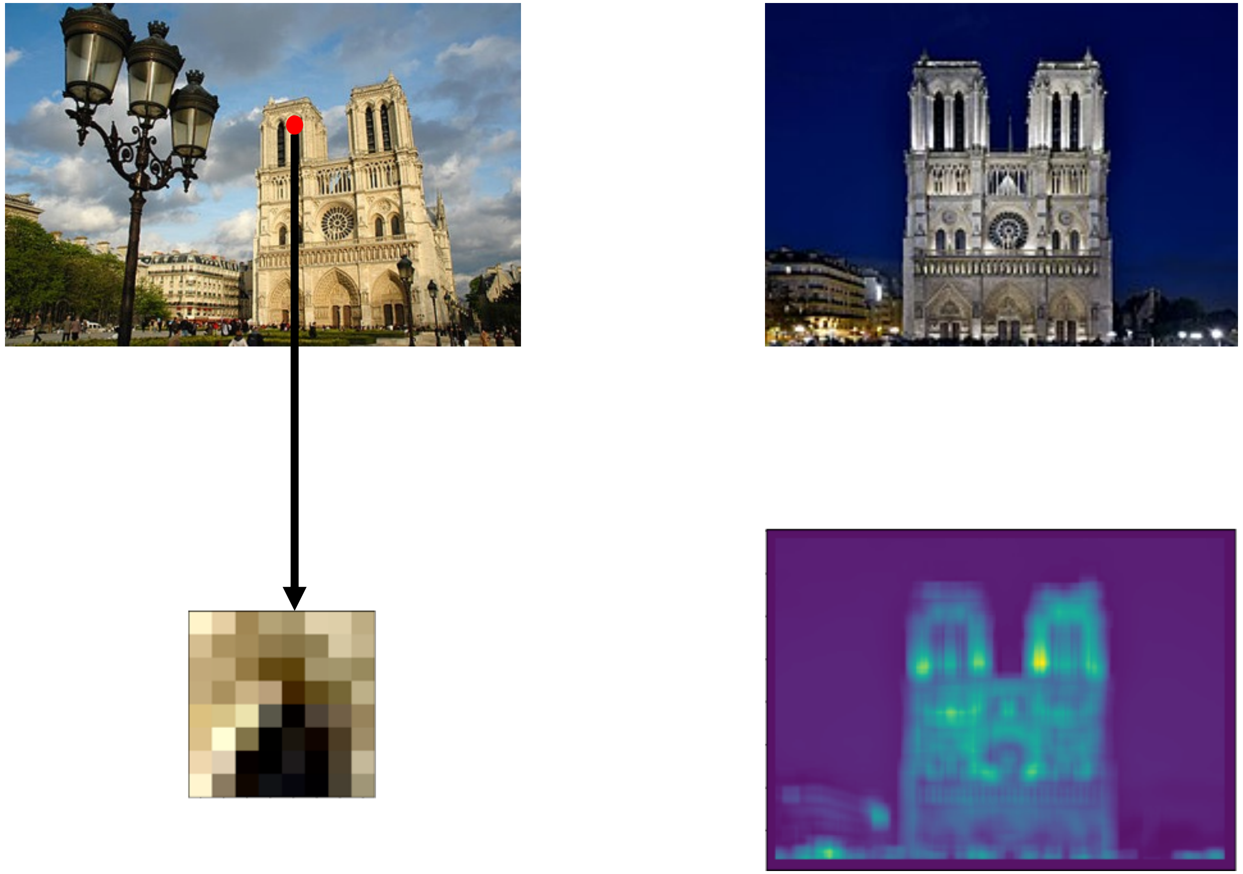




SSD



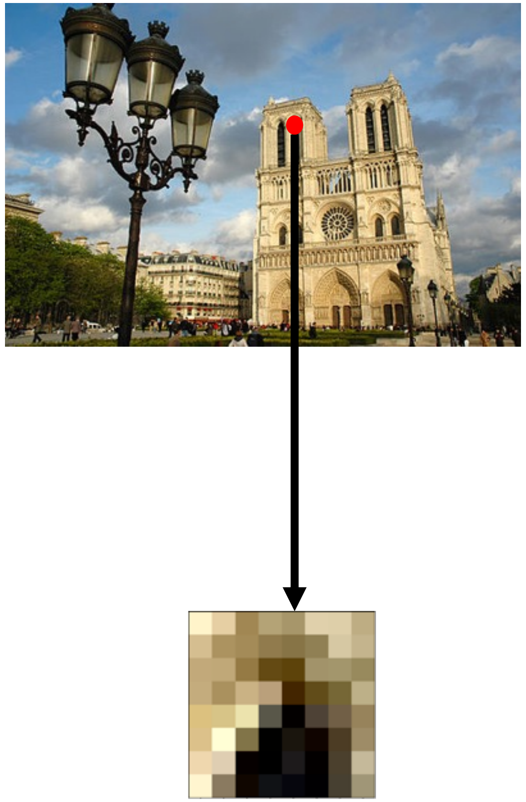
SSD

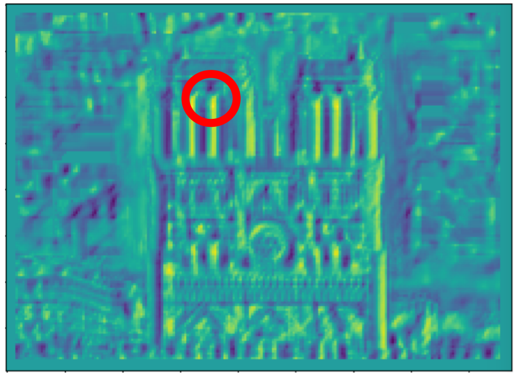


NCC - Normalized Cross Correlation

- Lighting and color change pixel intensities
- Example: increase brightness / contrast
- $I' = \alpha I + \beta$
- Subtract patch mean: invariance to β
- Divide by norm of vector: invariance to α
- $x' = x - < x >$
- $x'' = \frac{x'}{||x'||}$
- *similarity* = $x'' \cdot y''$

NCC - Normalized cross correlation





Basic correspondence

- Image patch as descriptor, NCC as similarity
- Invariant to?
  - Photometric transformations?
  - Translation?
  - Rotation?

Rotation invariance for feature descriptors

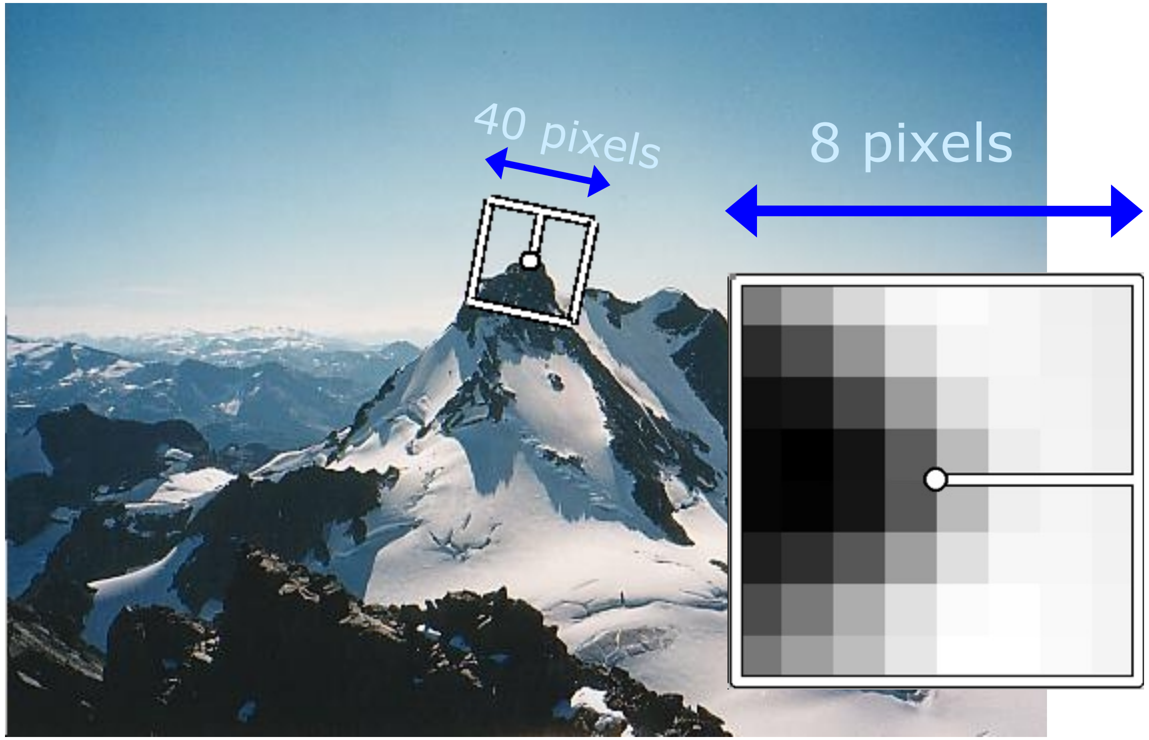
- Find dominant orientation of the image patch
  - This is given by $\mathbf{x}_{\mathbf{max}}$, the eigenvector of **M** corresponding to $\lambda_{\max}$ (the *larger* eigenvalue)
  - Rotate the patch according to this angle

Figure by Matthew Brown

Multiscale Oriented PatcheS descriptor

Take 40x40 square window around detected feature

- Scale to 1/5 size (using prefiltering)
- Rotate to horizontal
- Sample 8x8 square window centered at feature
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window



Adapted from slide by Matthew Brown

Detections at multiple scales

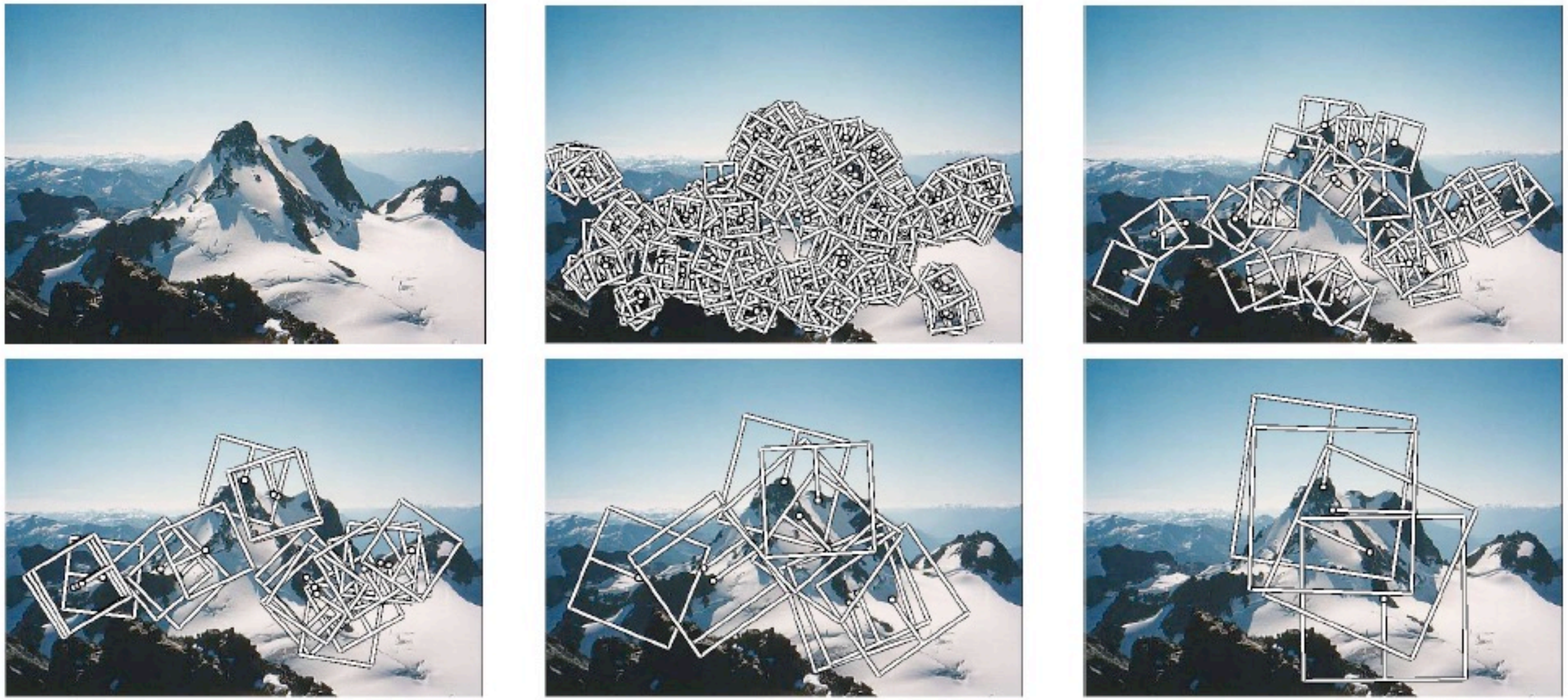


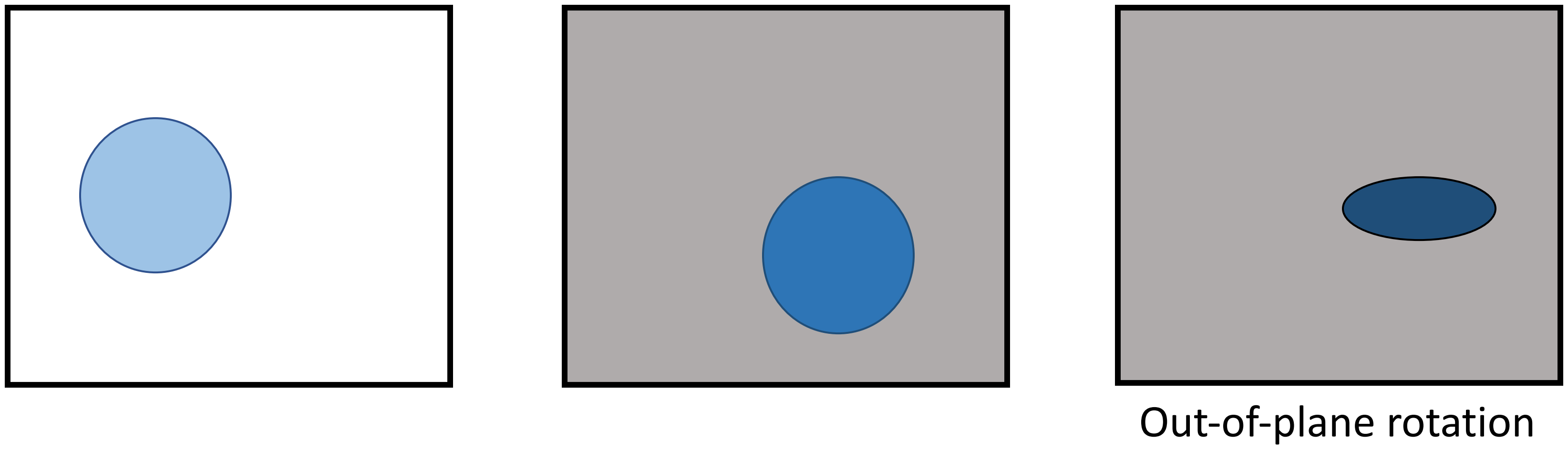
Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.

Invariance of MOPS

- Intensity
- Scale
- Rotation

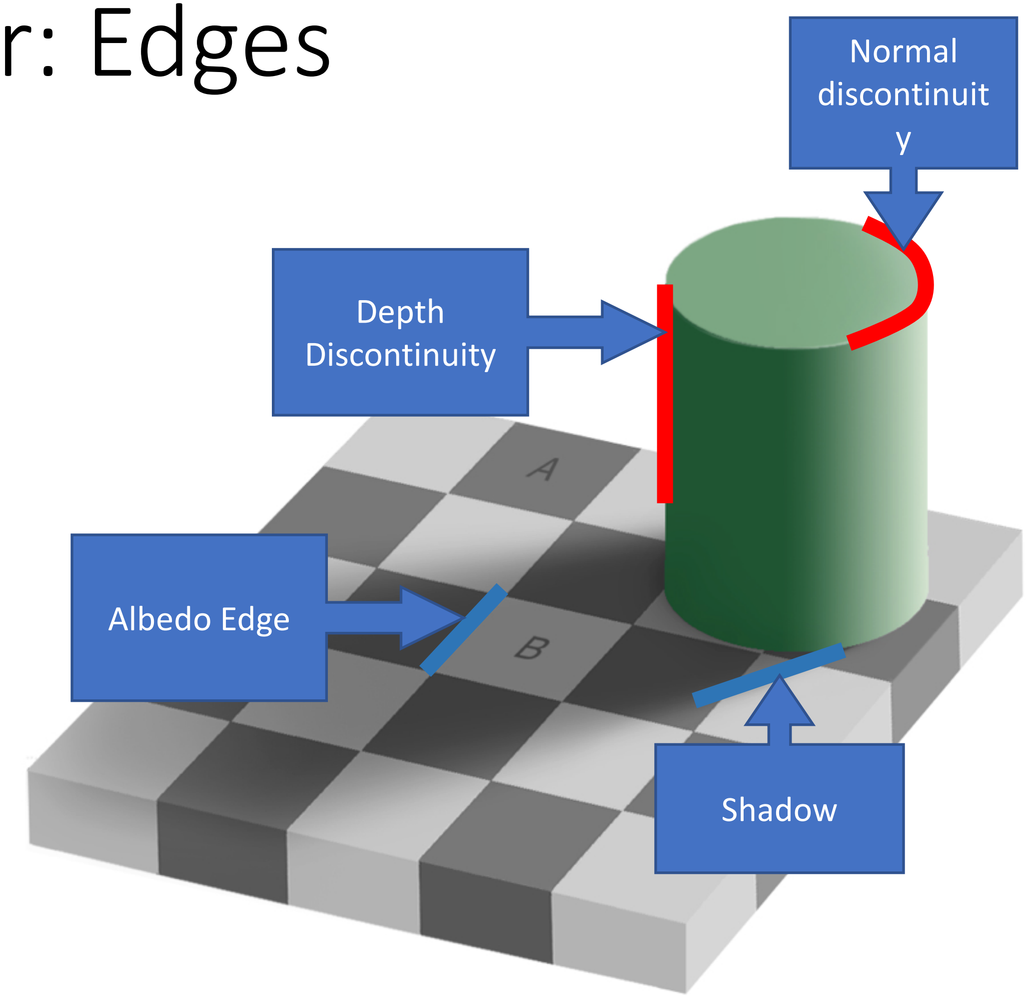
Color and Lighting

Out-of-plane rotation



Discussion

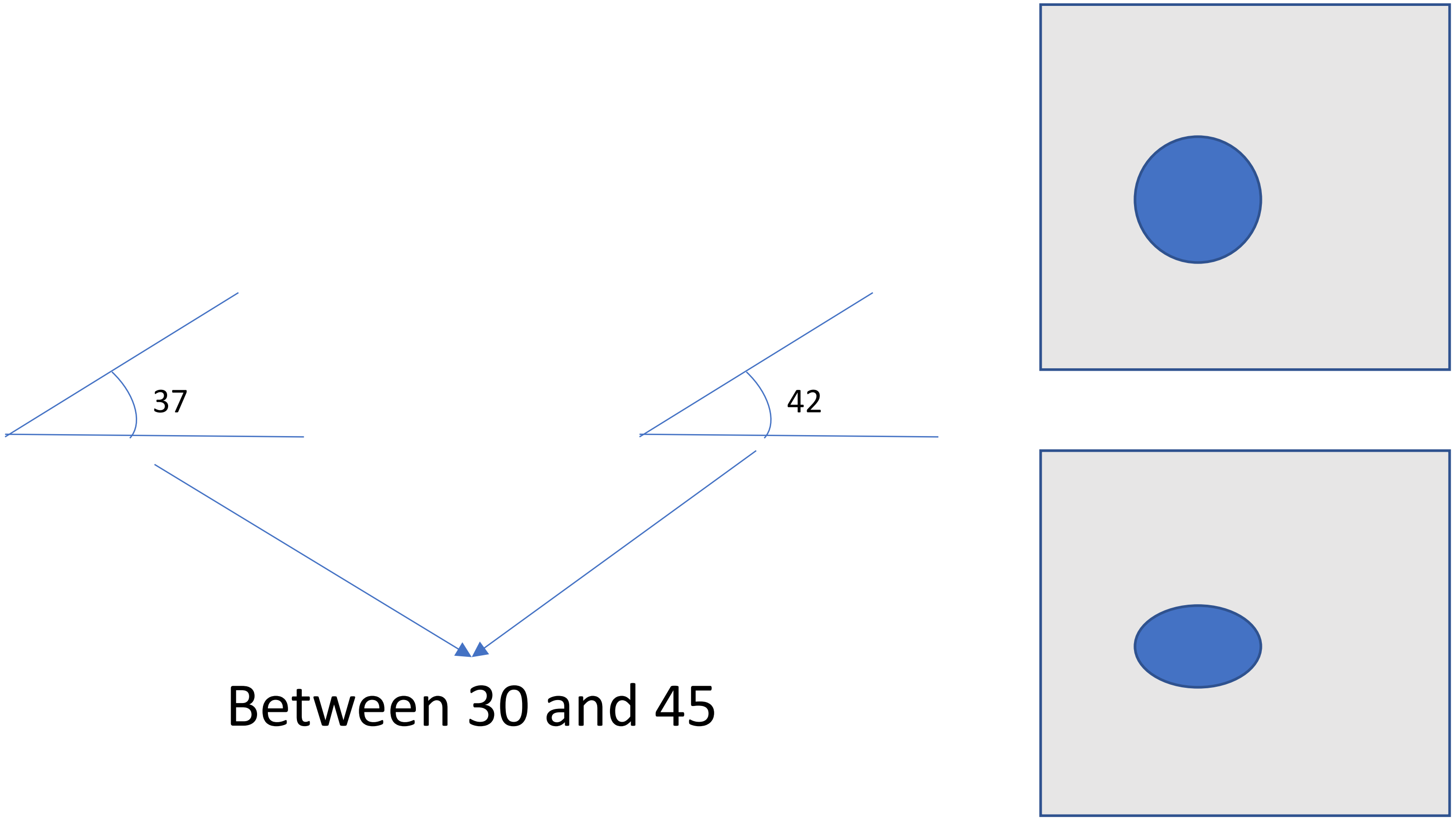
Better representation than color: Edges



Towards a better feature descriptor

- Match *pattern of edges*
  - Edge orientation – clue to shape
- Be resilient to *small deformations*
  - Deformations might move pixels around, but slightly
  - Deformations might change edge orientations, but slightly

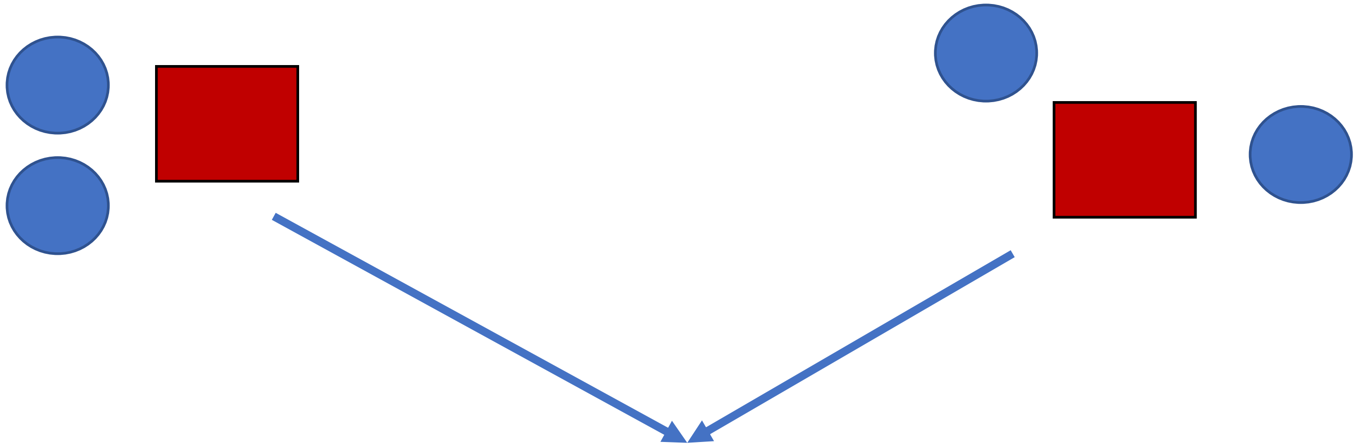
Invariance to deformation by quantization



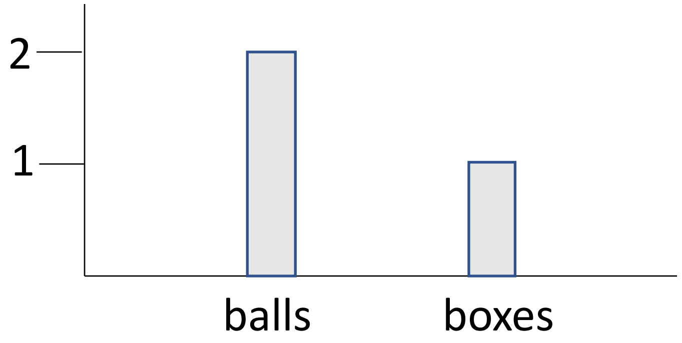
Invariance to deformation by quantization

$$g(\theta) = \begin{cases} 0 & \text{if } 0 < \theta < 2\pi/N \\ 1 & \text{if } 2\pi/N < \theta < 4\pi/N \\ 2 & \text{if } 4\pi/N < \theta < 6\pi/N \\ & \dots \\ N-1 & \text{if } 2(N-1)\pi/N \end{cases}$$

Spatial invariance by histograms



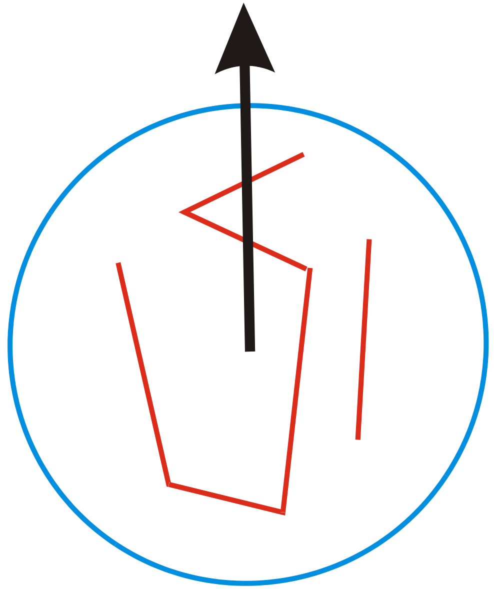
2 blue balls, one red box

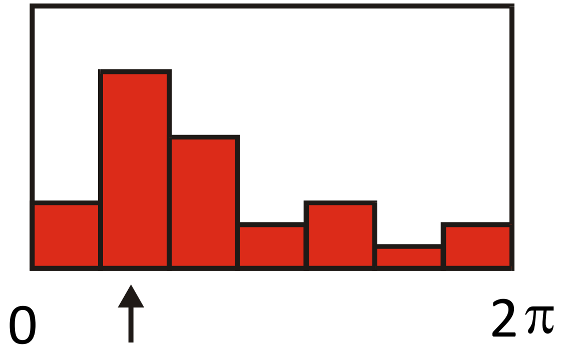


Rotation Invariance by Orientation Normalization

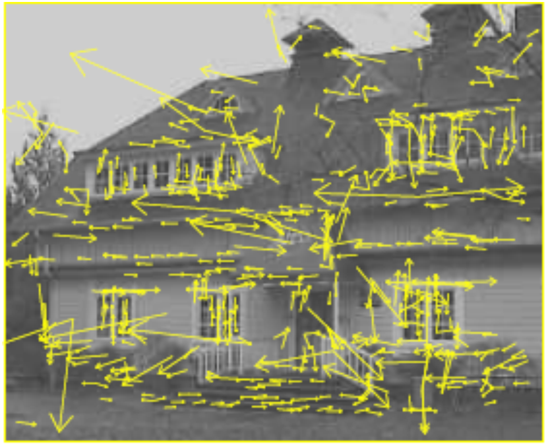
[Lowe, SIFT, 1999]

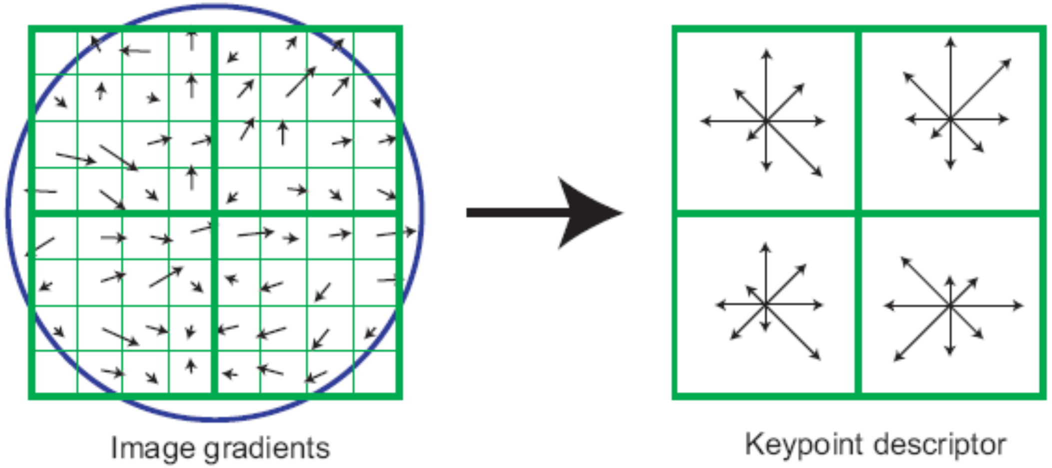
- Compute orientation histogram
- Select dominant orientation
- Normalize: rotate to fixed orientation





The SIFT descriptor





SIFT – Lowe IJCV 2004

Scale Invariant Feature Transform

Basic idea:

- DoG for scale-space feature detection
- Take 16x16 square window around detected feature
    - Compute gradient orientation for each pixel
    - Throw out weak edges (threshold gradient magnitude)
    - Create histogram of surviving edge orientations

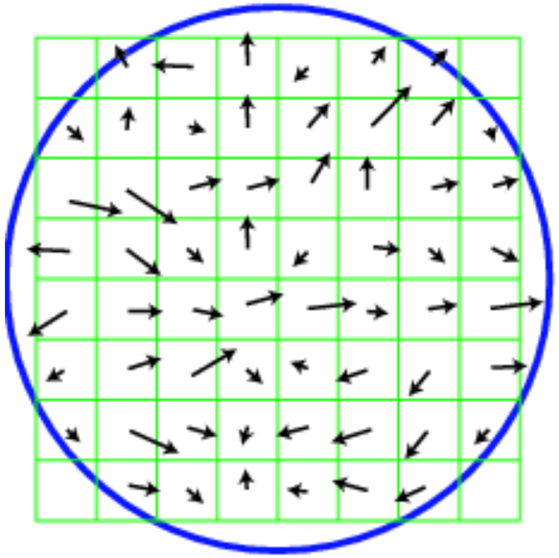
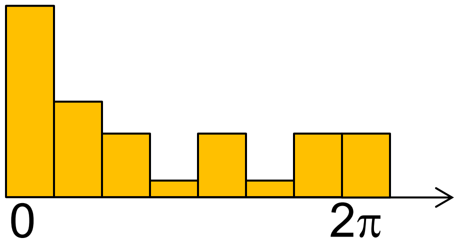
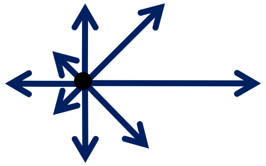


Image gradients



angle histogram



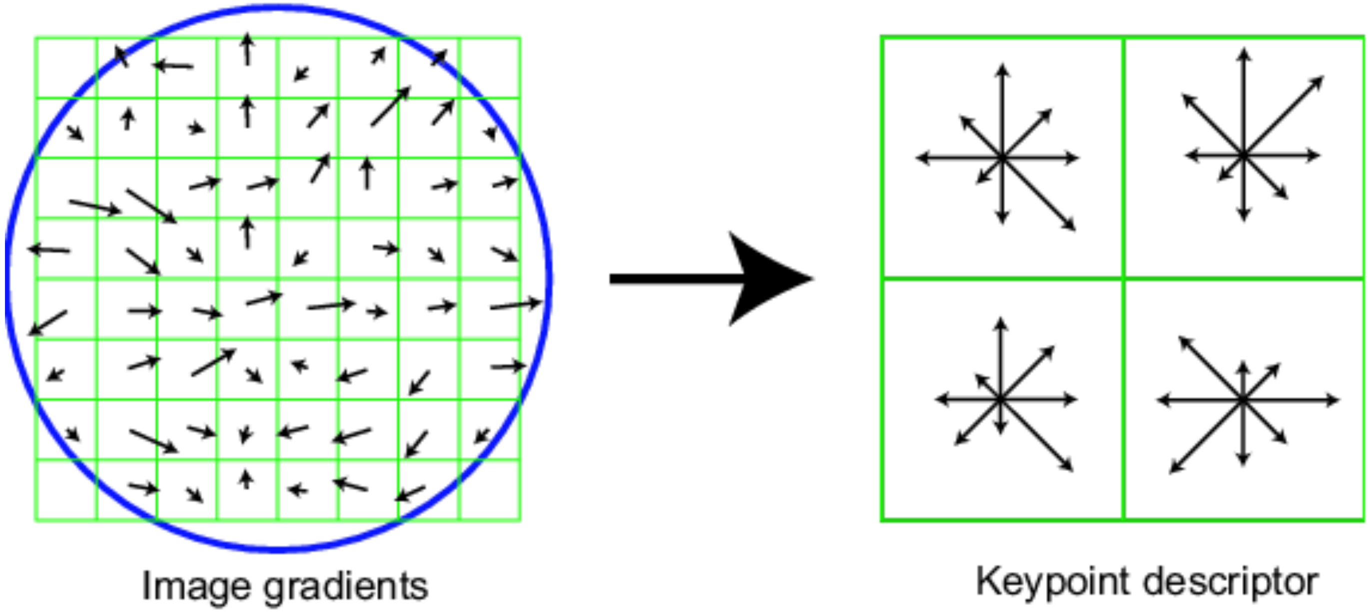
Keypoint descriptor

Adapted from slide by David Lowe

SIFT descriptor

Create histogram

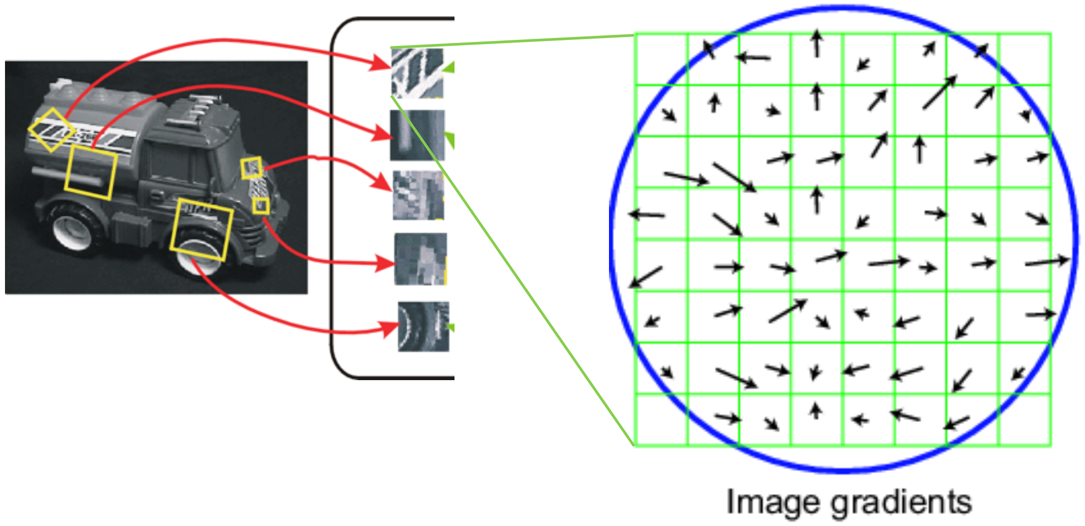
- Divide the 16x16 window into a 4x4 grid of cells (2x2 case shown below)
- Compute an orientation histogram for each cell
- 16 cells * 8 orientations = 128 dimensional descriptor



Adapted from slide by David Lowe

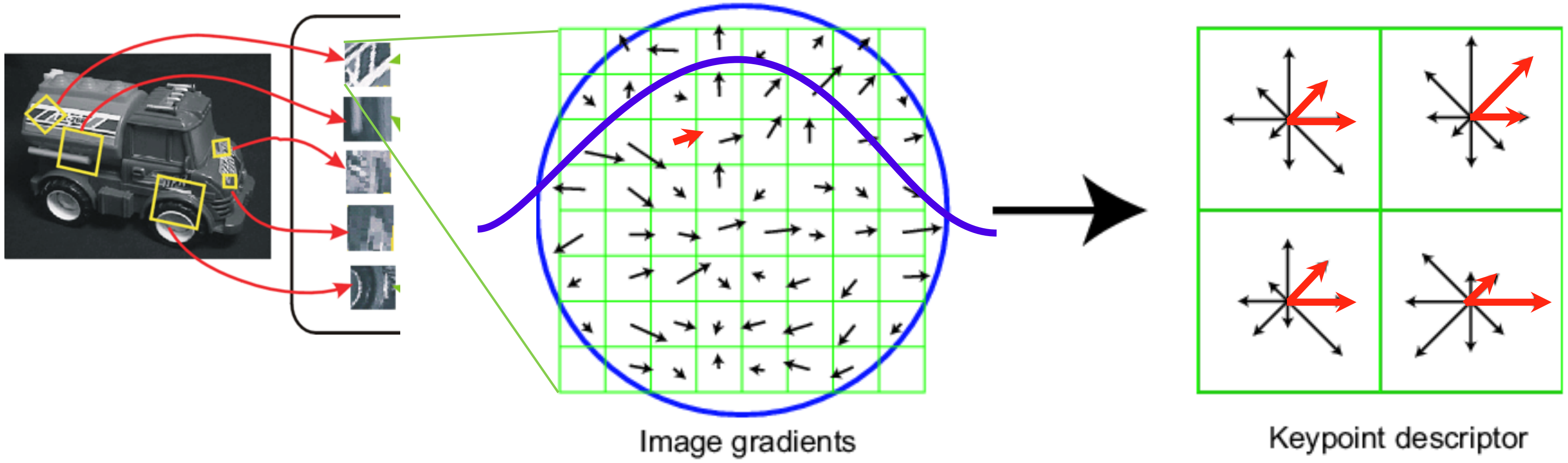
SIFT vector formation

- Computed on rotated and scaled version of window according to computed orientation & scale
    - resample the window
- Based on gradients weighted by a Gaussian



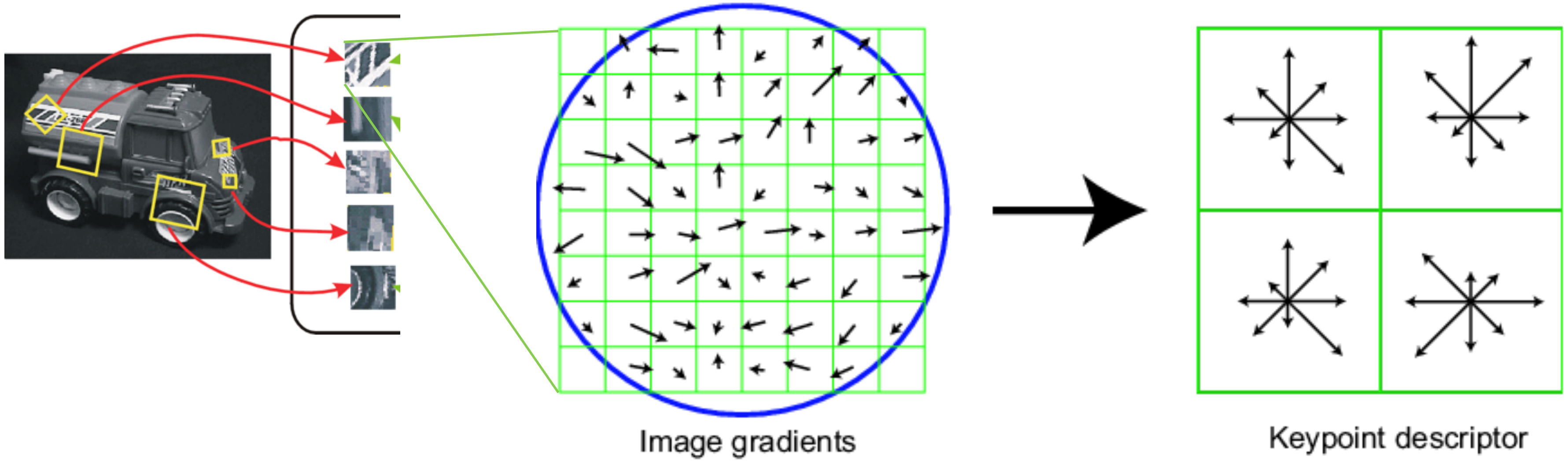
Ensure smoothness

- Trilinear interpolation
  - a given gradient contributes to 8 bins: 4 in space times 2 in orientation



Reduce effect of illumination

- 128-dim vector normalized to 1
- Threshold gradient magnitudes to avoid excessive influence of high gradients
  - after normalization, clamp gradients >0.2
  - renormalize



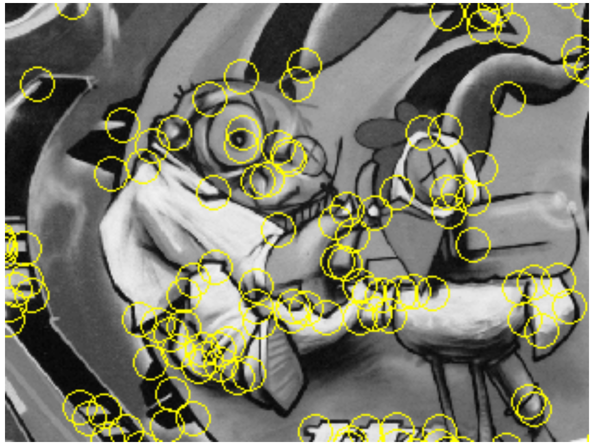
Properties of SIFT

Extraordinarily robust matching technique

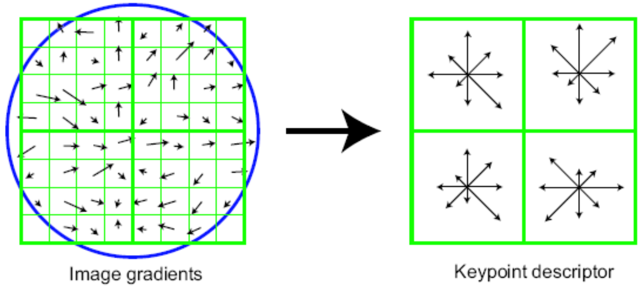
- Can handle changes in viewpoint
  - Up to about 60 degree out of plane rotation
- Can handle significant changes in illumination
  - Sometimes even day vs. night (below)
- Fast and efficient—can run in real time
- Lots of code available:
  http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT

Summary

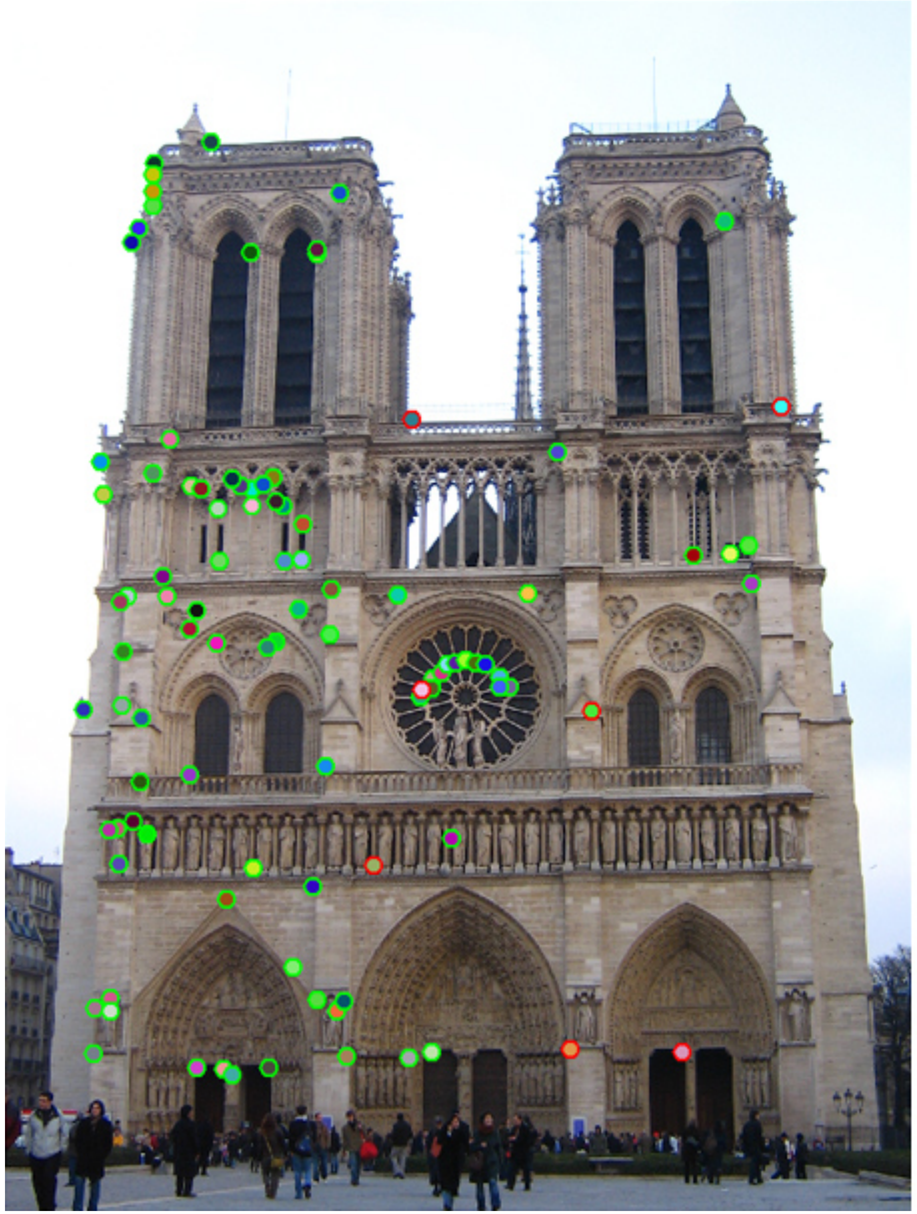
- Keypoint detection: repeatable and distinctive
  - Corners, blobs, stable regions
  - Harris, DoG

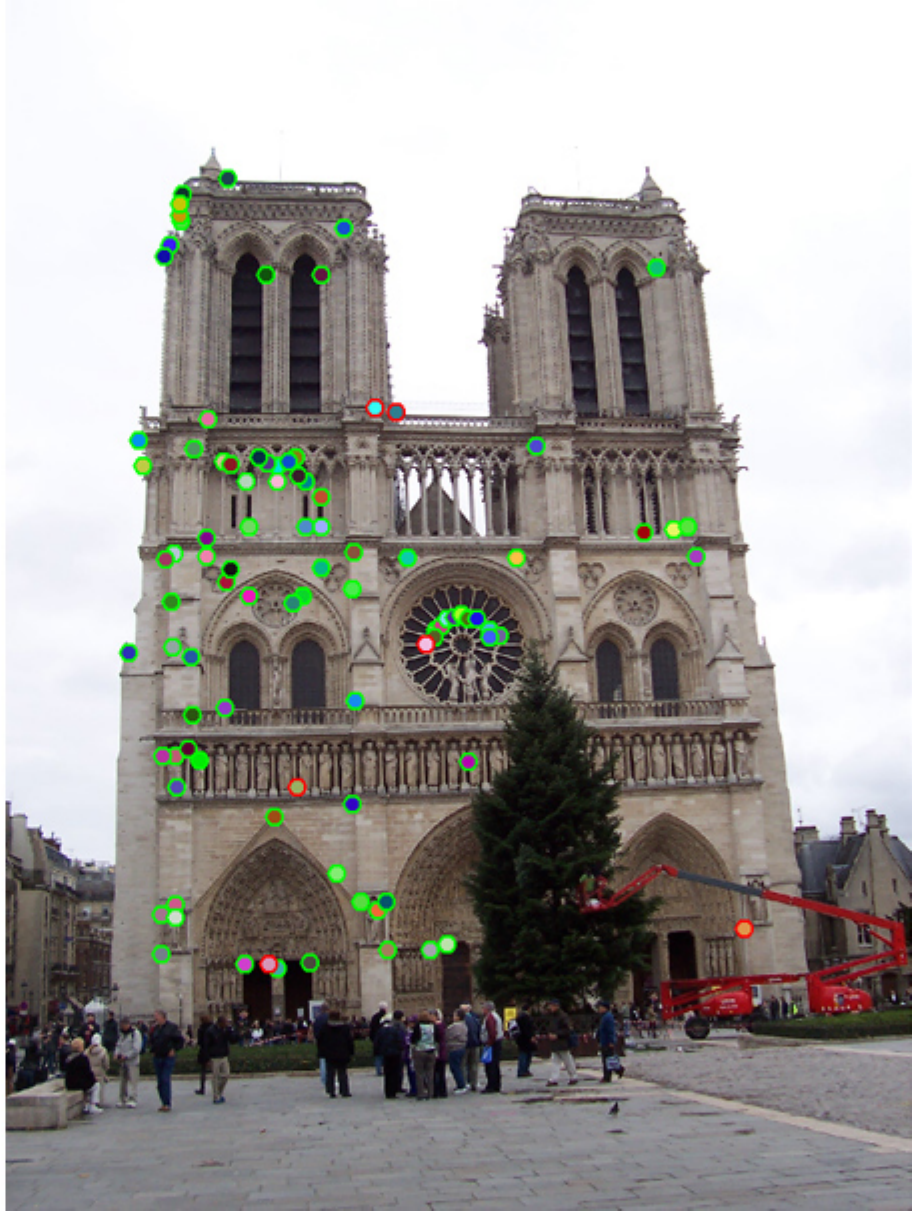


- Descriptors: robust and selective
  - spatial histograms of orientation
  - SIFT and variants are typically good for stitching and recognition
  - But, need not stick to one



Which features match?





Feature matching

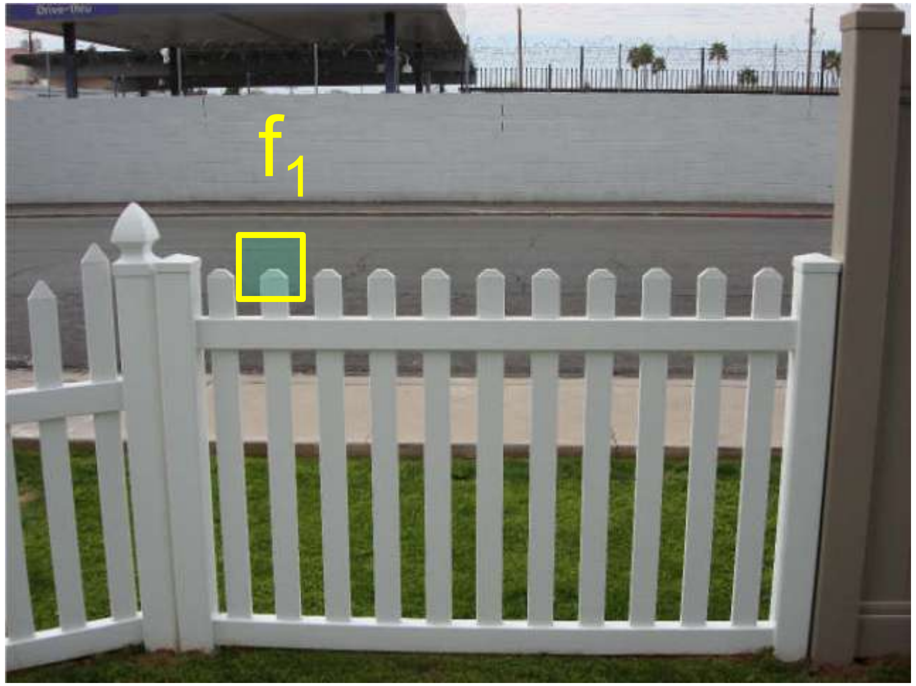
Given a feature in I_1, how to find the best match in I_2?

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

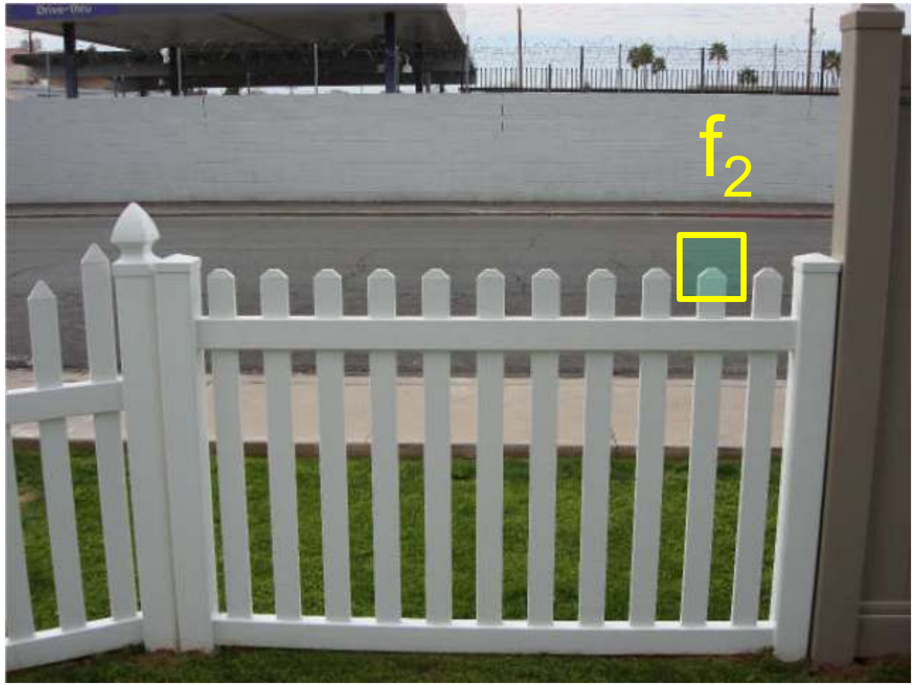
Feature distance

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

- Simple approach: L_2 distance, $||f_1 - f_2||$
- can give good scores to ambiguous (incorrect) matches



I_1

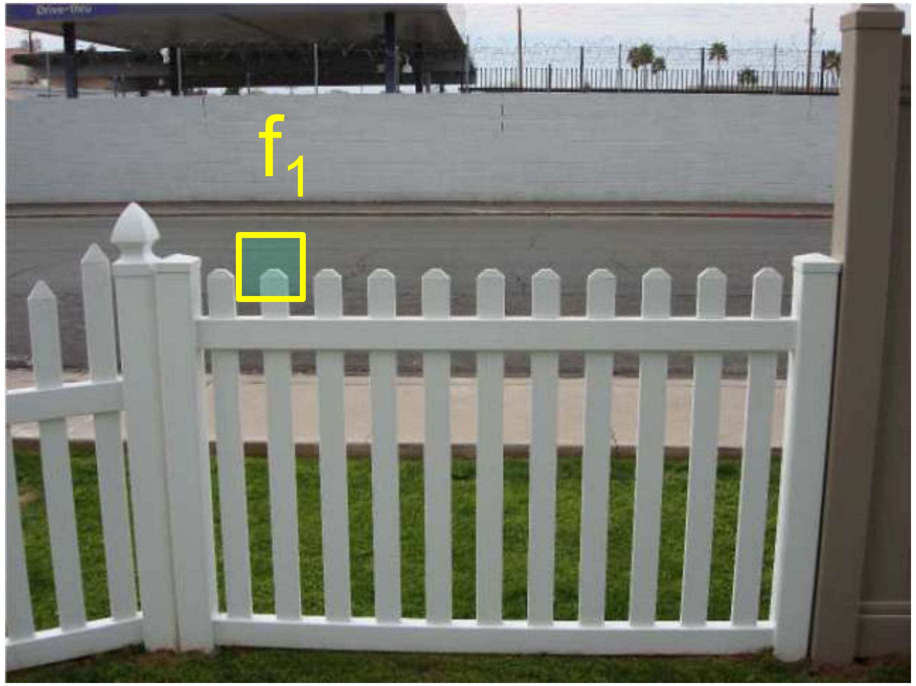


I_2

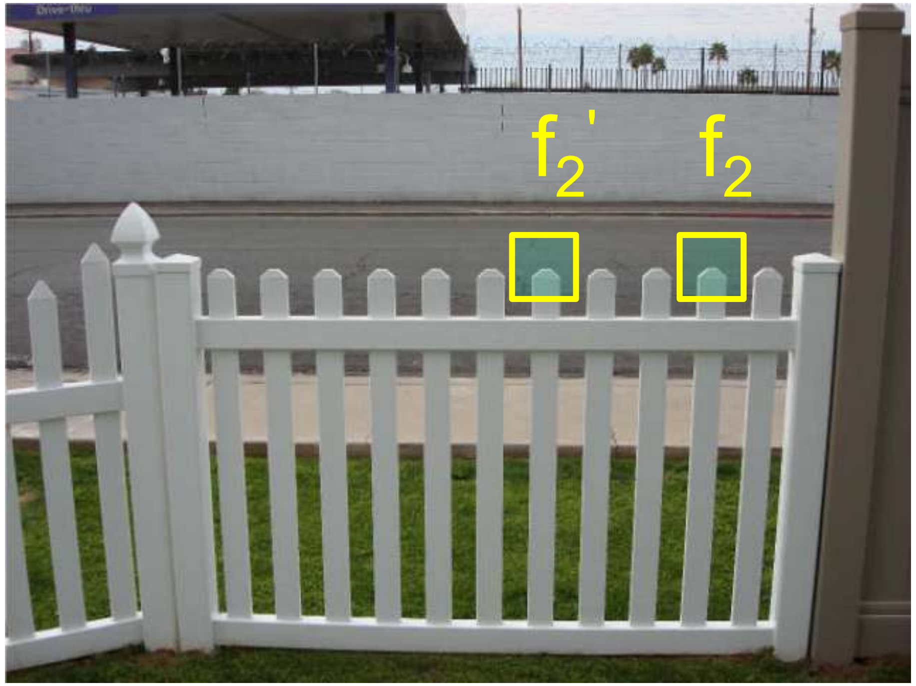
Feature distance

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

- Better approach: ratio distance = $||f_1 - f_2|| \,/\, ||f_1 - f_2'||$
  - f_2 is best SSD match to f_1 in I_2
  - f_2' is 2^{nd} best SSD match to f_1 in I_2
  - gives large values for ambiguous matches



I_1



I_2