Reconstruction

Fundamental matrix

$$\Rightarrow 0 = \vec{\mathbf{x}}_{img}^{(2)} K_2^{-T} [\mathbf{t}]_\times R K_1^{-1} \vec{\mathbf{x}}_{img}^{(1)}$$

$$\Rightarrow 0 = \vec{\mathbf{x}}_{img}^{(2)} F \vec{\mathbf{x}}_{img}^{(1)}$$

Fundamental matrix

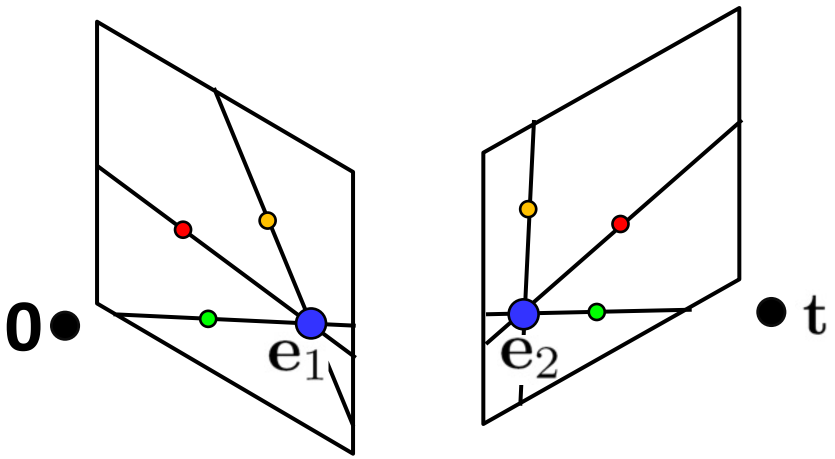
Fundamental matrix result

$$\mathbf{q}^T\mathbf{F}\mathbf{p} = 0$$

(Longuet-Higgins, 1981)

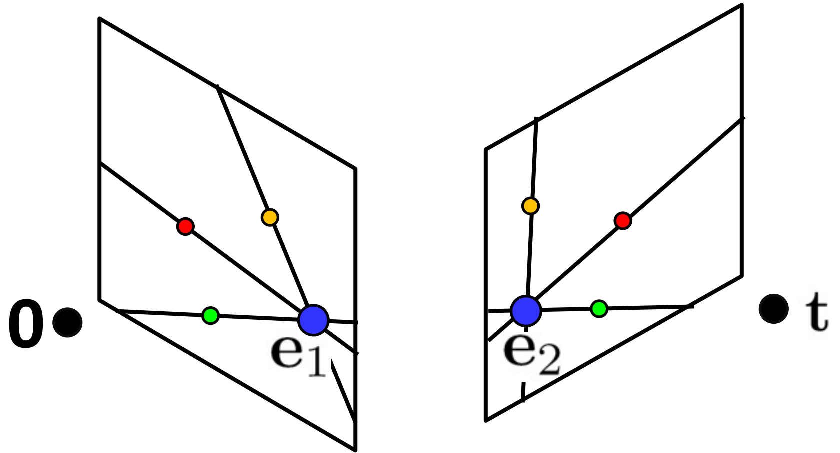
Properties of the Fundamental Matrix

- $\mathbf{F}\mathbf{p}$ is the epipolar line associated with $\mathbf{p}$
- $\mathbf{F}^T\mathbf{q}$ is the epipolar line associated with $\mathbf{q}$



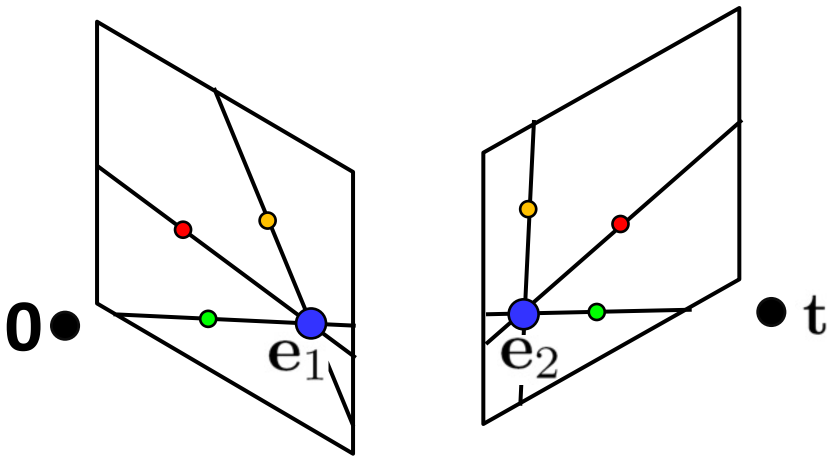
Properties of the Fundamental Matrix

- $\mathbf{F}\mathbf{p}$ is the epipolar line associated with $\mathbf{p}$
- $\mathbf{F}^T\mathbf{q}$ is the epipolar line associated with $\mathbf{q}$
- $\mathbf{F}\mathbf{e}_1 = \mathbf{0}$ and $\mathbf{F}^T\mathbf{e}_2 = \mathbf{0}$
- All epipolar lines contain epipole



Properties of the Fundamental Matrix

- $\mathbf{Fp}$ is the epipolar line associated with $\mathbf{p}$
- $\mathbf{F}^T\mathbf{q}$ is the epipolar line associated with $\mathbf{q}$
- $\mathbf{Fe}_1 = \mathbf{0}$ and $\mathbf{F}^T\mathbf{e}_2 = \mathbf{0}$
- $\mathbf{F}$ is rank 2



Why is F rank 2?

- F is a 3 x 3 matrix
- But there is a vector c_1 and c_2 such that $Fc_1 = 0$ and $F^T c_2 = 0$

Estimating **F**

- If we don’t know $\mathbf{K}_1$, $\mathbf{K}_2$, **R**, or **t**, can we estimate **F** for two images?
- Yes, given enough correspondences

Estimating F – 8-point algorithm

- The fundamental matrix F is defined by

$$\mathbf{x}'^{\mathrm{T}}\mathbf{F}\mathbf{x} = 0$$

for any pair of matches x and x' in two images.

- Let x=($u,v,1$)$^{\mathrm{T}}$ and x'=($u',v',1$)$^{\mathrm{T}}$,

$$\mathbf{F} = \begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}$$

each match gives a linear equation

$$uu' f_{11} + vu' f_{12} + u' f_{13} + uv' f_{21} + vv' f_{22} + v' f_{23} + uf_{31} + vf_{32} + f_{33} = 0$$

8-point algorithm

$$\begin{bmatrix} u_1u_1' & v_1u_1' & u_1' & u_1v_1' & v_1v_1' & v_1' & u_1 & v_1 & 1 \\ u_2u_2' & v_2u_2' & u_2' & u_2v_2' & v_2v_2' & v_2' & u_2 & v_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_nu_n' & v_nu_n' & u_n' & u_nv_n' & v_nv_n' & v_n' & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{bmatrix} = 0$$

- In reality, instead of solving $\mathbf{Af} = 0$, we seek $\mathbf{f}$ to minimize $\|\mathbf{Af}\|$, least eigenvector of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$.

8-point algorithm – Problem?

- **F** should have rank 2
- To enforce that **F** is of rank 2, F is replaced by F’ that minimizes $\|\mathbf{F}-\mathbf{F}'\|$ subject to the rank constraint.

- This is achieved by SVD. Let $\mathbf{F}=\mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}}$, where

$$\Sigma=\begin{bmatrix}\sigma_1 & 0 & 0\\ 0 & \sigma_2 & 0\\ 0 & 0 & \sigma_3\end{bmatrix}, \text{ let } \Sigma'=\begin{bmatrix}\sigma_1 & 0 & 0\\ 0 & \sigma_2 & 0\\ 0 & 0 & 0\end{bmatrix}$$

then $\mathbf{F}'=\mathbf{U}\Sigma'\mathbf{V}^{\mathrm{T}}$ is the solution.

Recovering camera parameters from F / E

- Can we recover R and t between the cameras from F?

$$F = K_2^{-T}[\mathbf{t}]_\times R K_1^{-1}$$

- No: K_1 and K_2 are in principle arbitrary matrices
- What if we knew K_1 and K_2 to be identity?

$$E = [\mathbf{t}]_\times R$$

Recovering camera parameters from E

$$E = [\mathbf{t}]_\times R$$
$$\mathbf{t}^T E = \mathbf{t}^T [\mathbf{t}]_\times R = 0$$
$$E^T \mathbf{t} = 0$$

- **t** is a solution to $E^T\mathbf{x} = 0$
- Can't distinguish between **t** and c**t** for constant scalar c
- How do we recover R?

Recovering camera parameters from E

$$E = [\mathbf{t}]_\times R$$

- We know E and **t**
- Consider taking SVD of E and $[\mathbf{t}]_X$

$$[\mathbf{t}]_\times = U\Sigma V^T$$

$$E = U'\Sigma' V'^T$$

$$U'\Sigma' V'^T = E = [\mathbf{t}]_\times R = U\Sigma V^T R$$

$$U'\Sigma' V'^T = U\Sigma V^T R$$

$$V'^T = V^T R$$

Recovering camera parameters from E

$$E = [\mathbf{t}]_\times R$$
$$\mathbf{t}^T E = \mathbf{t}^T [\mathbf{t}]_\times R = 0$$
$$E^T \mathbf{t} = 0$$

- **t** is a solution to $E^T\mathbf{x} = 0$
- Can't distinguish between **t** and c**t** for constant scalar c

8-point algorithm

- Pros: it is linear, easy to implement and fast
- Cons: susceptible to noise
- Degenerate: if points are on same plane

- Normalized 8-point algorithm: Hartley
  - Position origin at centroid of image points
  - Rescale coordinates so that center to farthest point is sqrt (2)

Structure-from-motion

- Given a bunch of uncalibrated images of a scene
  - Recover camera parameters
  - Recover 3D scene structure

- Start from correspondences
  - Estimate E
  - Recover camera parameters
  - Solve for 3D points using (multi-view) stereo

- Wherefrom correspondences?

The correspondence problem

Till now

- Geometry of image formation
- Stereo reconstruction
  - Given 3D → 2D correspondence, find K, R, t
  - Given 2 images, correspondence, K, R, t, find 3D points
  - Given 2 images, correspondence, find F, E, R, t, 3D points

Till now

- Geometry of image formation
- Stereo reconstruction
  - Given 3D → 2D correspondence, find K, R, t
  - Given 2 images, correspondence, K, R, t, find 3D points
  - Given 2 images, correspondence, find F, E, R, t, 3D points

Other applications of correspondence

- Image alignment
- Motion tracking
- Robot navigation

Correspondence can be challenging

Fei-Fei Li

Correspondence

by Diva Sian

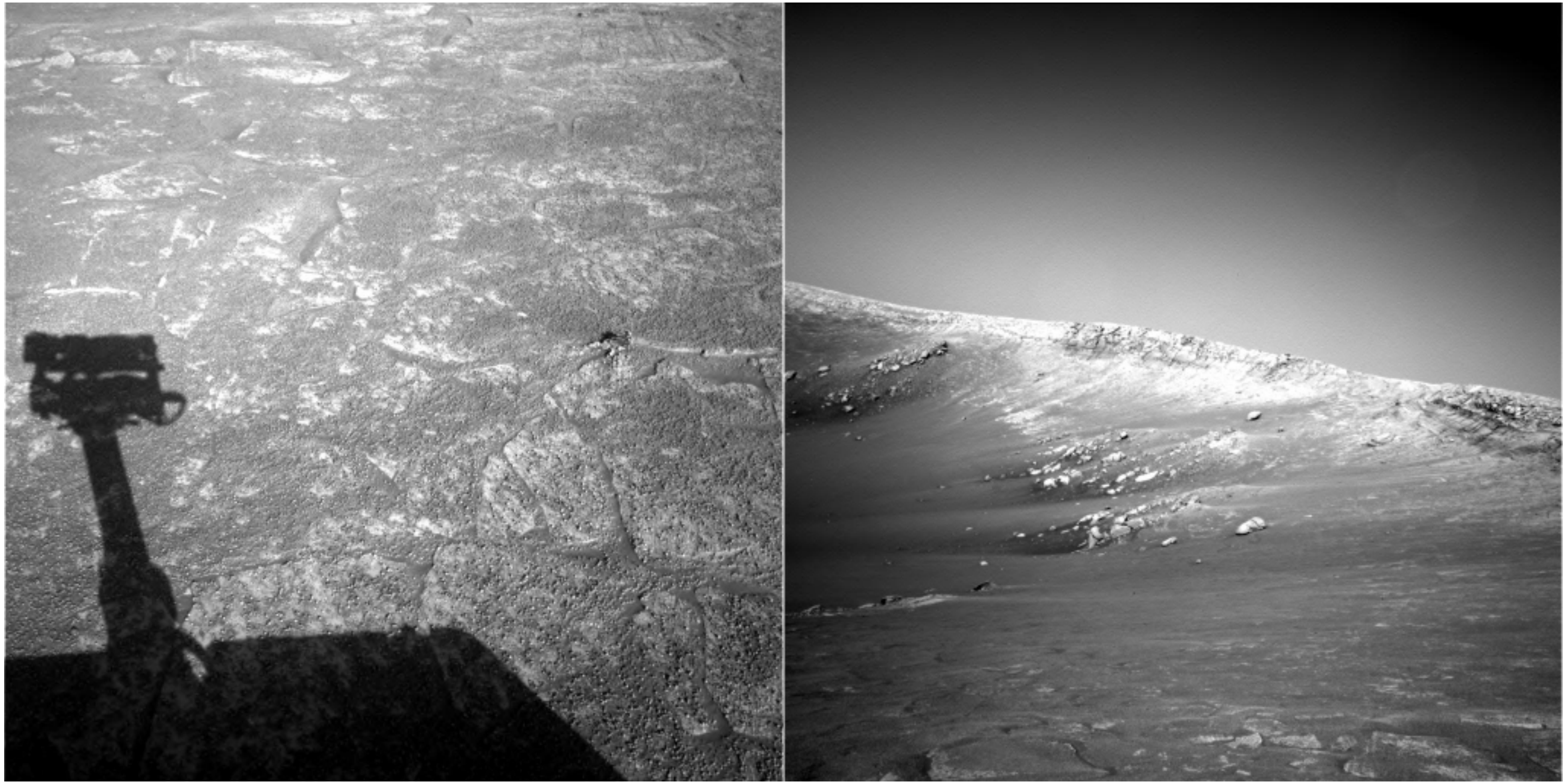
by swashford

Harder case

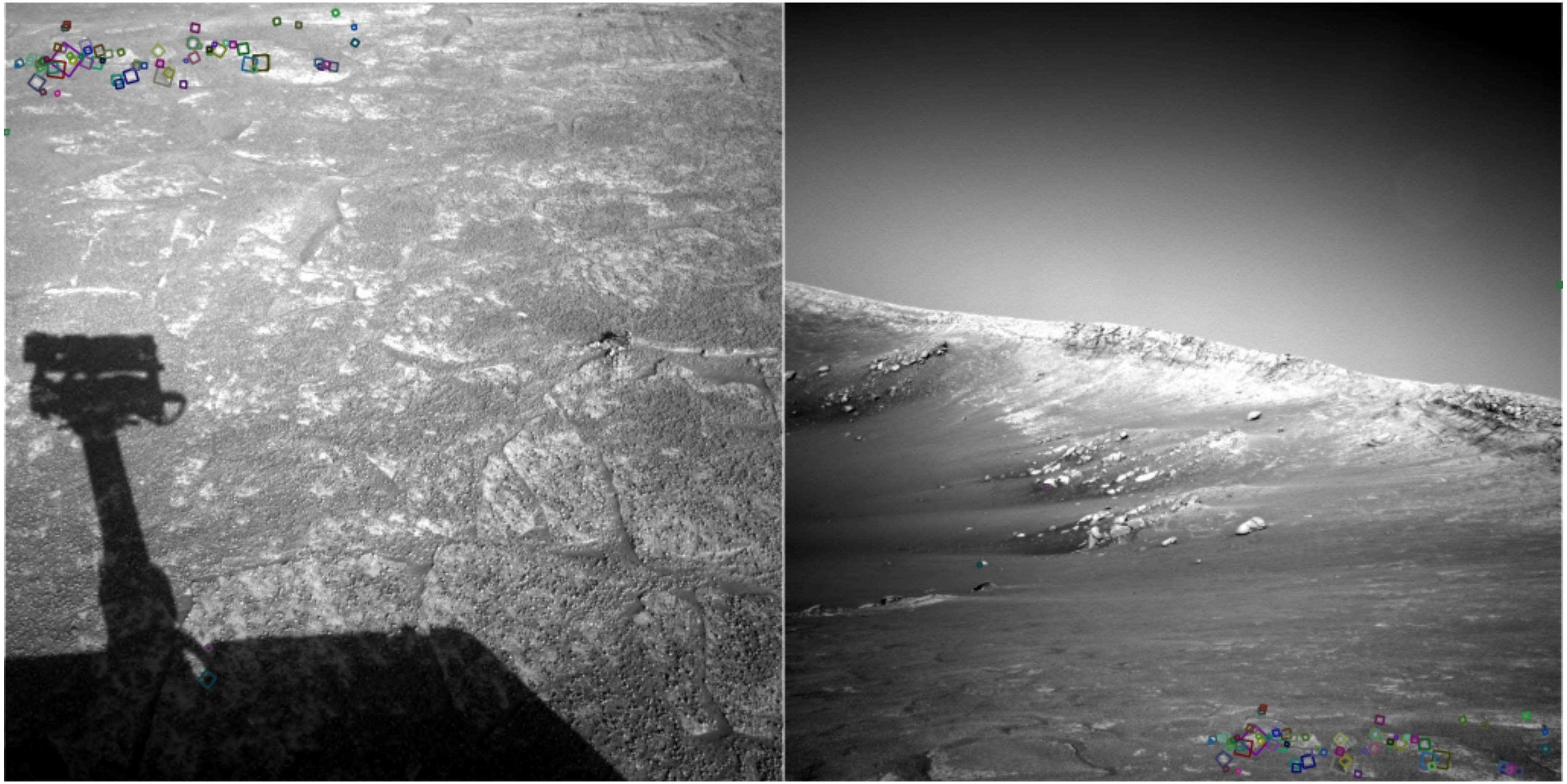
by Diva Sian

by scgbt

Harder still?



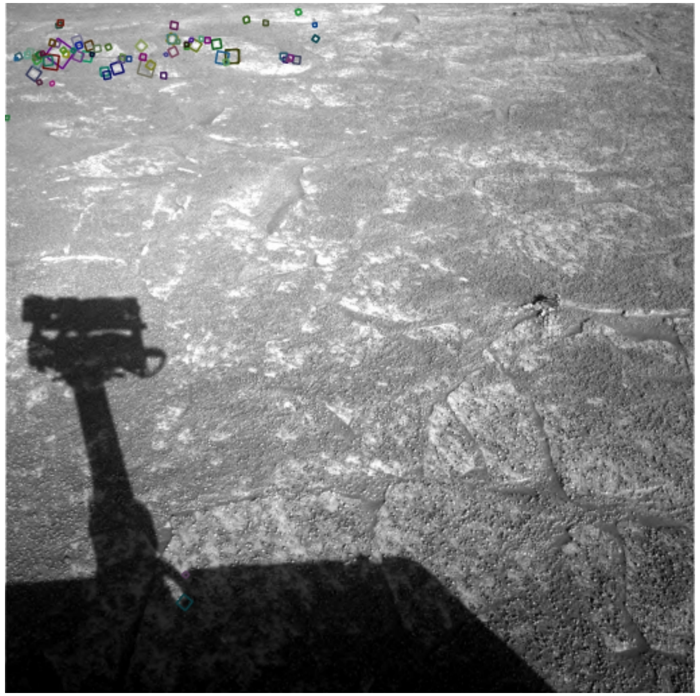
Answer below (look for tiny colored squares…)

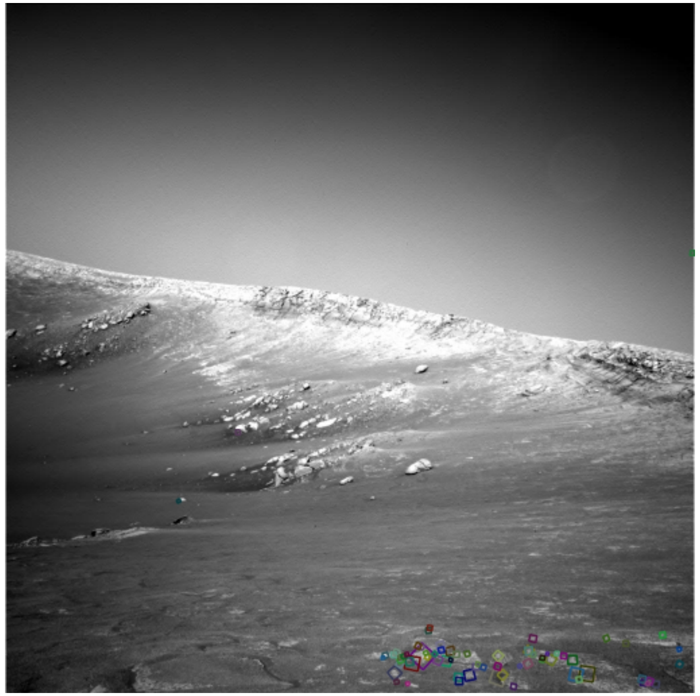


NASA Mars Rover images
with SIFT feature matches

Sparse vs dense correspondence

- Sparse correspondence: produce a few, high confidence matches
  - Good enough for estimating pose or relationship between cameras
- Dense correspondence: try to match every pixel
  - Needed if we want 3D location of every pixel





Sparse correspondence

- Which pixels should be searching correspondence for?
  - *Feature points / keypoints*

What makes a good feature point?

Characteristics of good feature points

- Repeatability / invariance
  - The same feature point can be found in several images despite geometric and photometric transformations
- Saliency / distinctiveness
  - Each feature point is distinctive
  - Fewer ”false” matches

Goal: repeatability

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

No chance to find true matches!

- Yet we have to be able to run the detection procedure *independently* per image.

Kristen Grauman

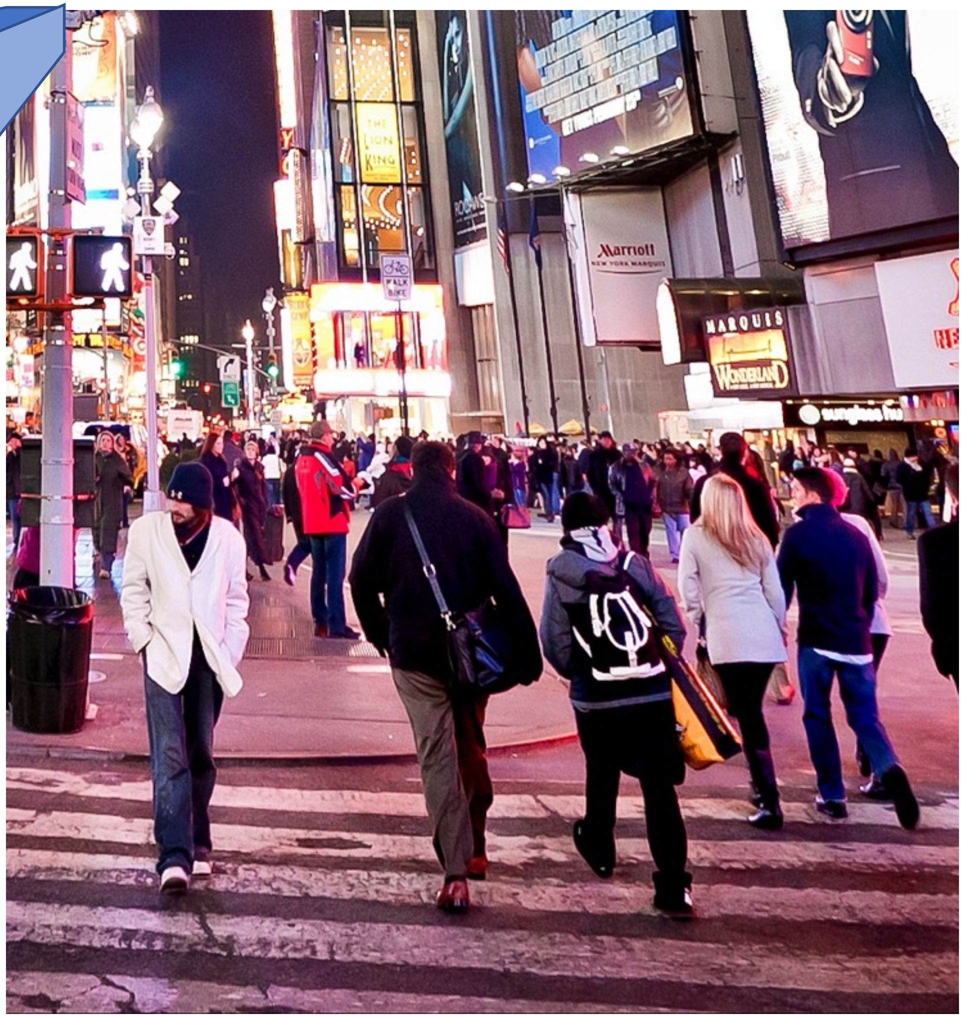
Goal: distinctiveness

- The feature point should be distinctive enough that it is easy to match
  - Should *at least* be distinctive from other patches nearby

The correspondence problem

What does an image look like?

71	109	61	71	86
66	96	77	76	77
63	98	82	79	64
65	105	73	92	83
90	107	80	96	113

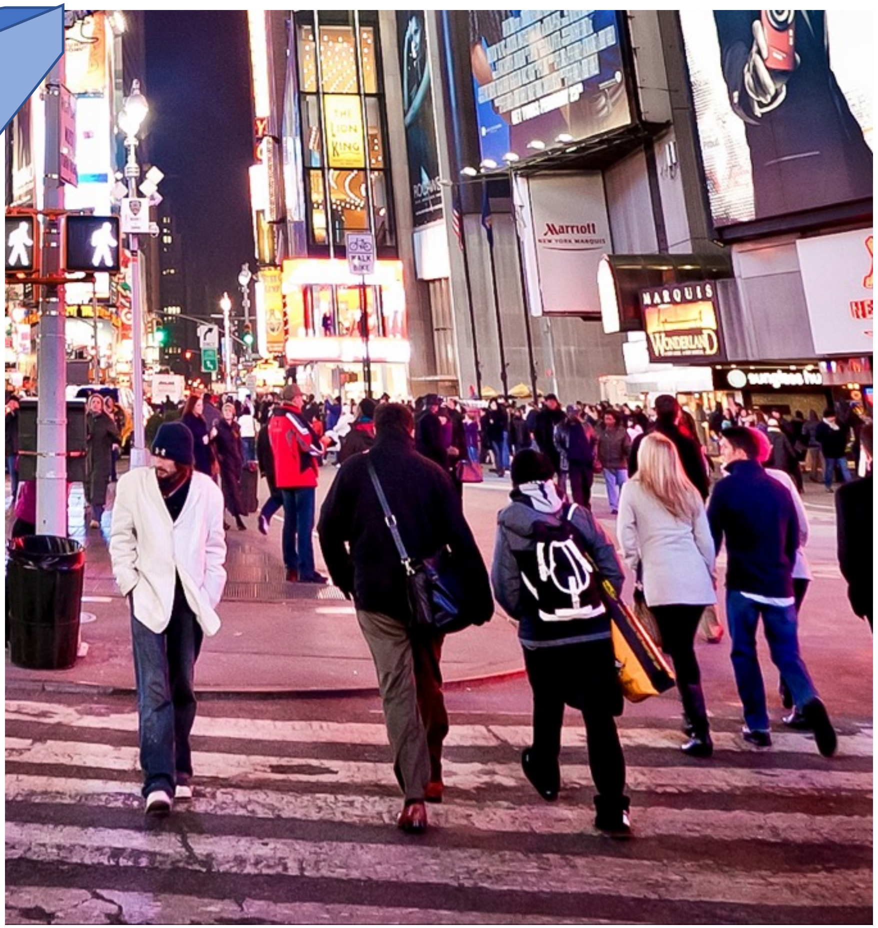


The aperture problem

The aperture problem

- Individual pixels are ambiguous
- Idea: look at whole patch!

71	109	61	71	86
66	96	77	76	77
63	98	82	79	64
65	105	73	92	83
90	107	80	96	113



The aperture problem

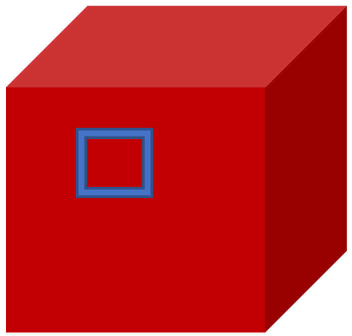
- Individual pixels are ambiguous
- Idea: Look at whole patches!

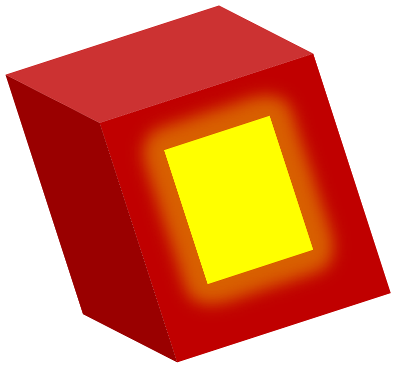
The aperture problem

- Individual pixels are ambiguous
- Idea: Look at whole patches!

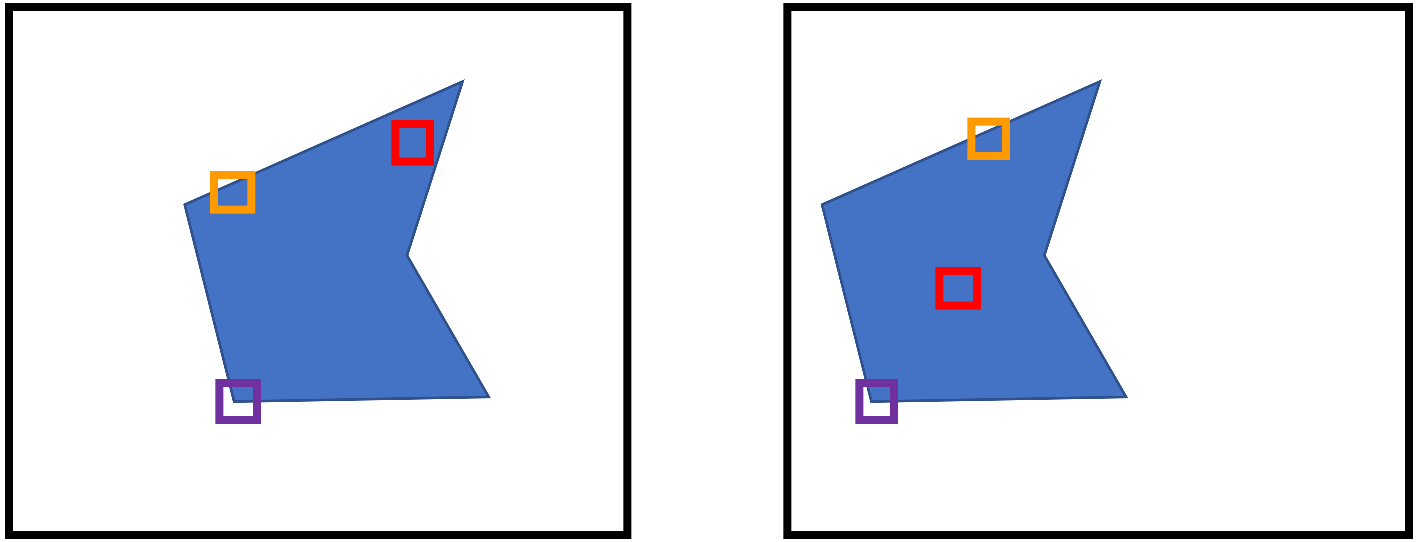
The aperture problem

- *Some local neighborhoods* are ambiguous





The aperture problem



Sparse correspondences

- For many applications, a few good correspondences suffice
  - Camera calibration
  - Estimating essential matrix
  - Reconstructing a sparse cloud of 3D points

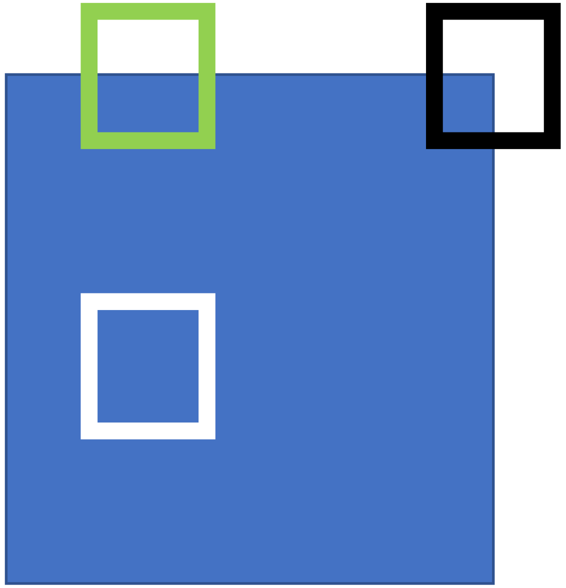
- Detect points that will produce good correspondences
- Match detected points from both images

Interest point detectors

- Informative: Must be able to reliably match from two views
- Reproducible: Must be detected in both views

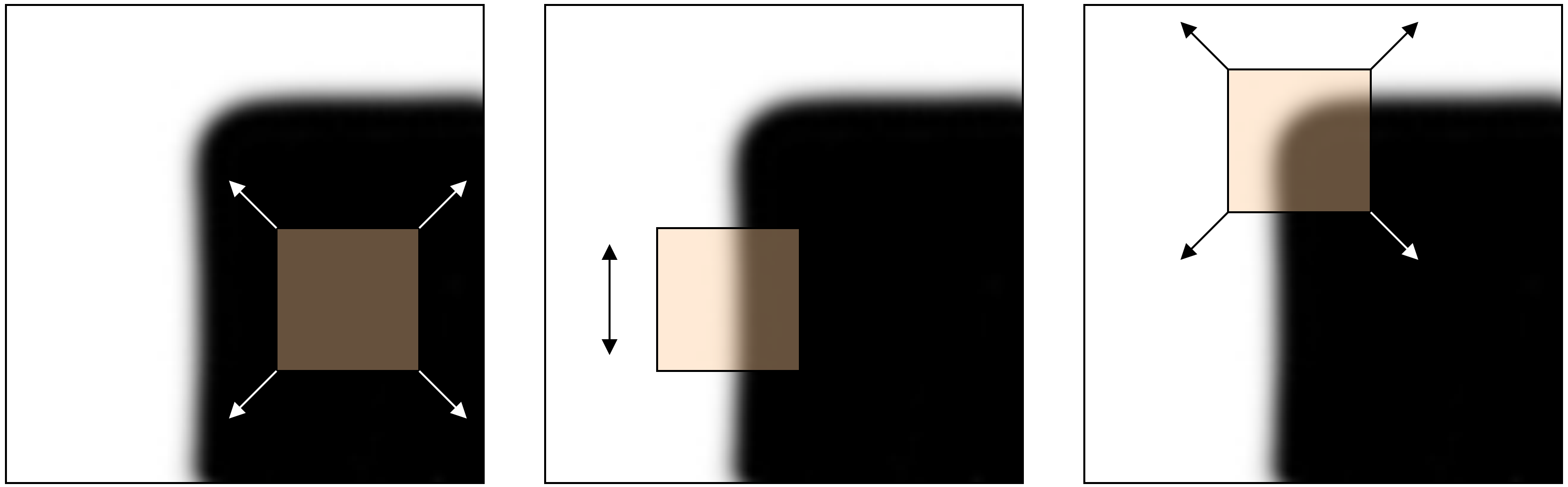
Harris corner detector

- Main idea: Translating patch should cause large differences
- An example of an *interest point detector*



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

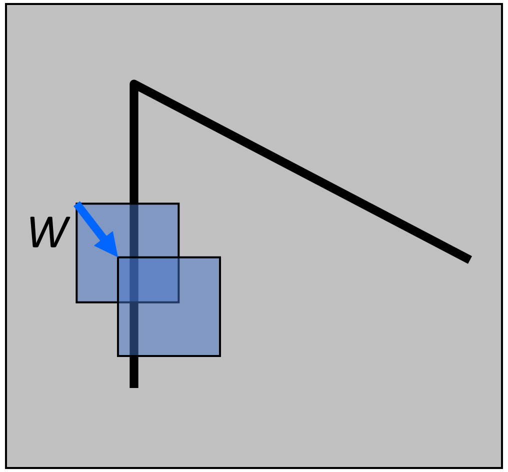
"edge": no change along the edge direction

"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?
- Write pixels in window as a vector:



$$\phi_0 = [I(0,0), I(0,1), \ldots, I(n,n)]$$

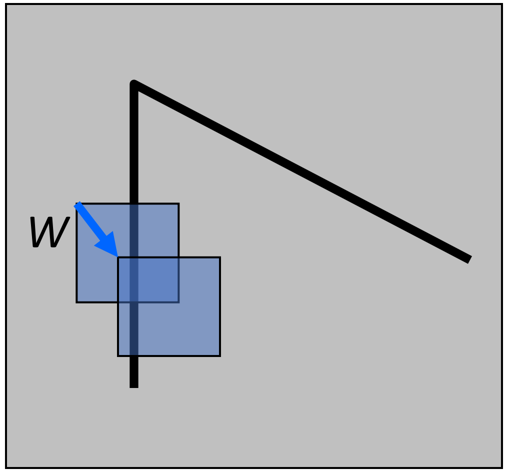
$$\phi_1 = [I(0+u, 0+v), I(0+u, 1+v), \ldots, I(n+u, n+v)]$$

$$E(u,v) = \|\phi_0 - \phi_1\|_2^2$$

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*):



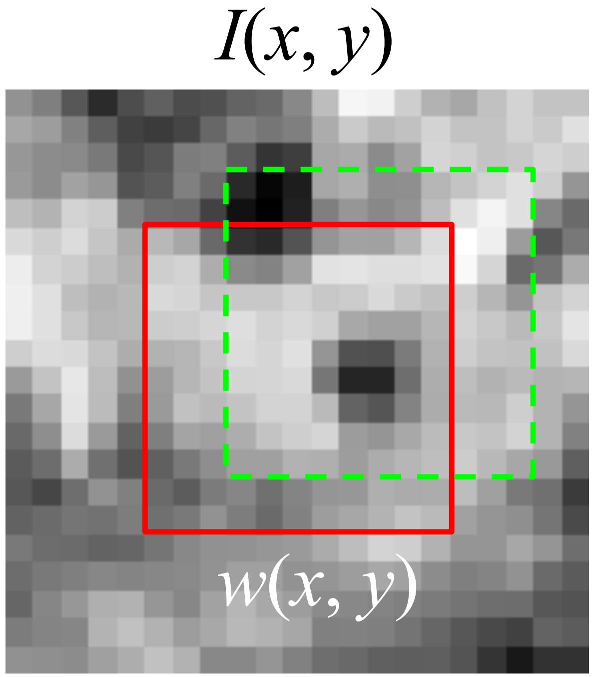
$$E(u,v) = \sum_{(x,y)\in W} \left[I(x+u, y+v) - I(x,y)\right]^2$$

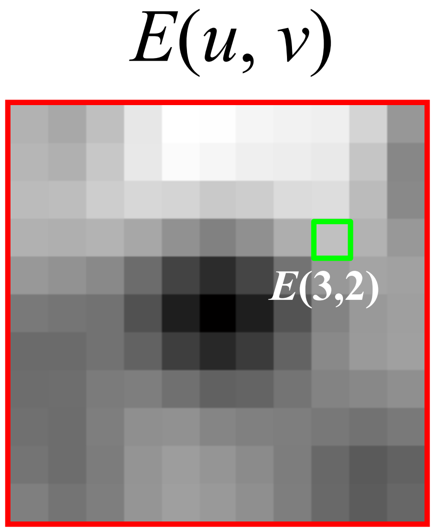
- We want E(u,v) to be *as high as possible for all u, v!*

Corner Detection: Mathematics

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

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





Corner Detection: Mathematics

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

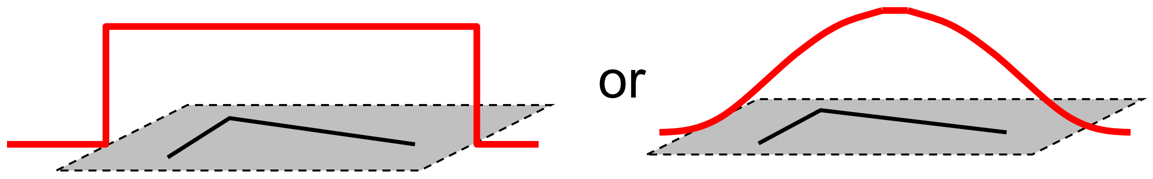
$$E(u,v)=\sum_{x,y} w(x,y)\left[I(x+u,y+v)-I(x,y)\right]^2$$

Window function

Shifted intensity

Intensity

Window function $w(x,y) =$



1 in window, 0 outside

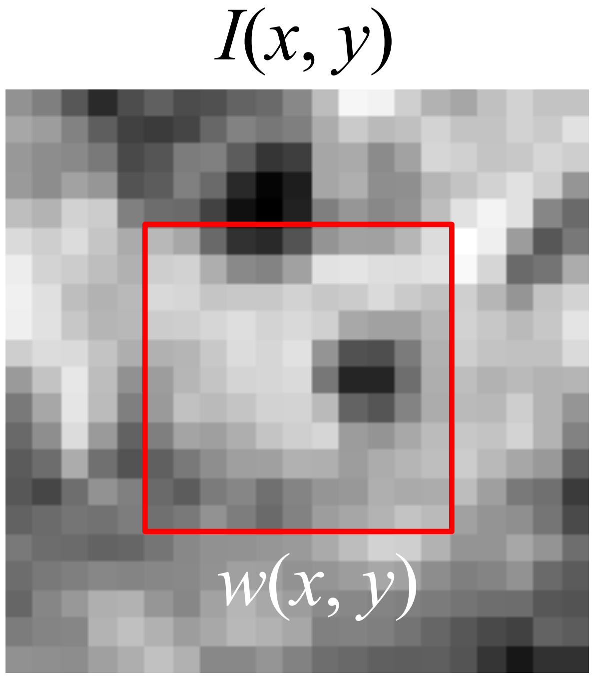
Gaussian

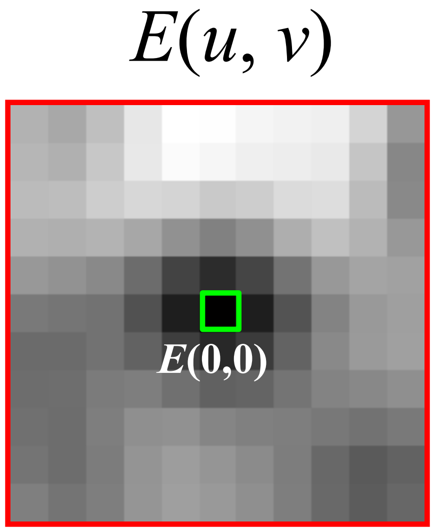
Source: R. Szeliski

Corner Detection: Mathematics

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

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





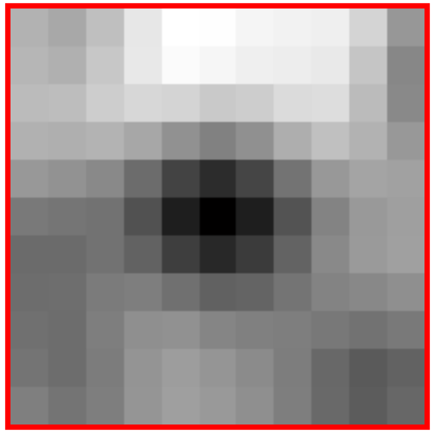
Corner Detection: Mathematics

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

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

We want to find out how this function behaves for small shifts

$E(u, v)$



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

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

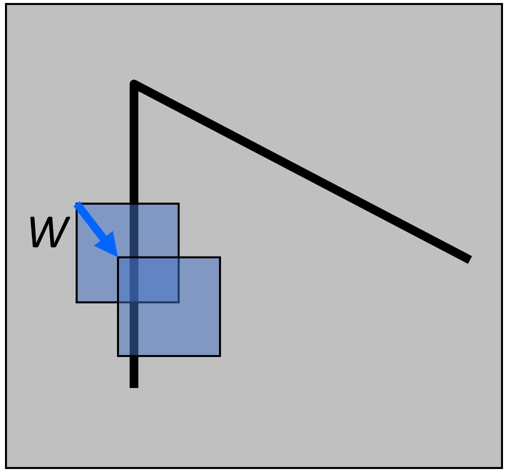
shorthand: $I_x = \frac{\partial I}{\partial x}$

Plugging this into the formula on the previous slide…

Corner detection: the math

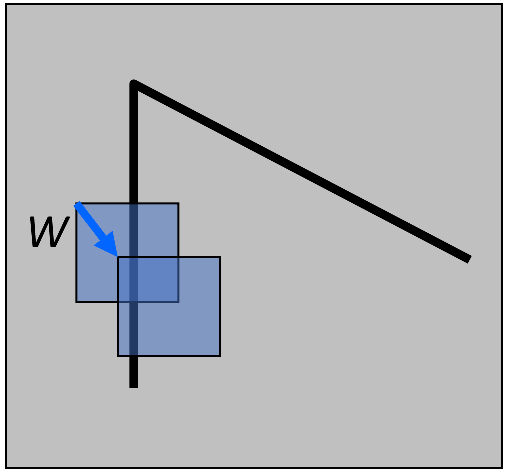
Consider shifting the window *W* by (*u,v*)

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



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

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?

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

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

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

$$M \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow E(u,v) = 0$$

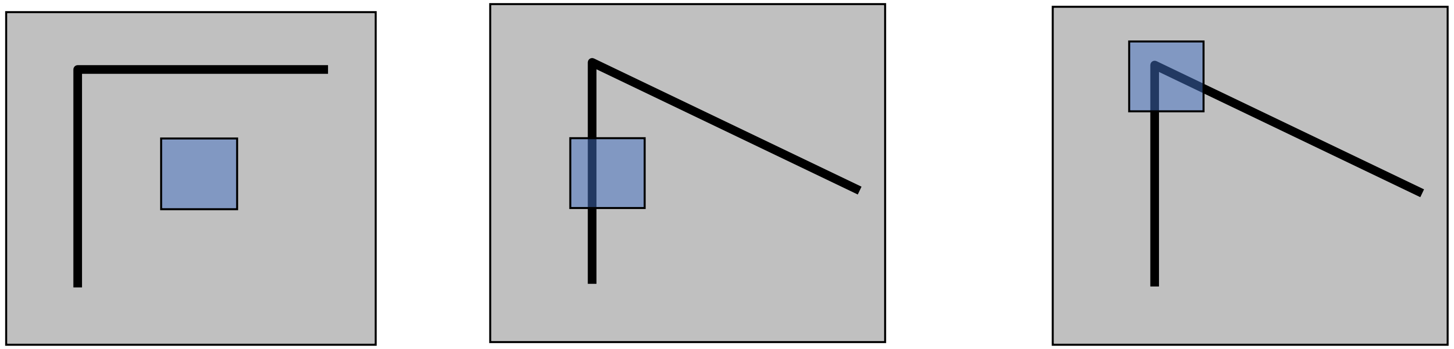
$$M \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Leftrightarrow E(u,v) = 0$$

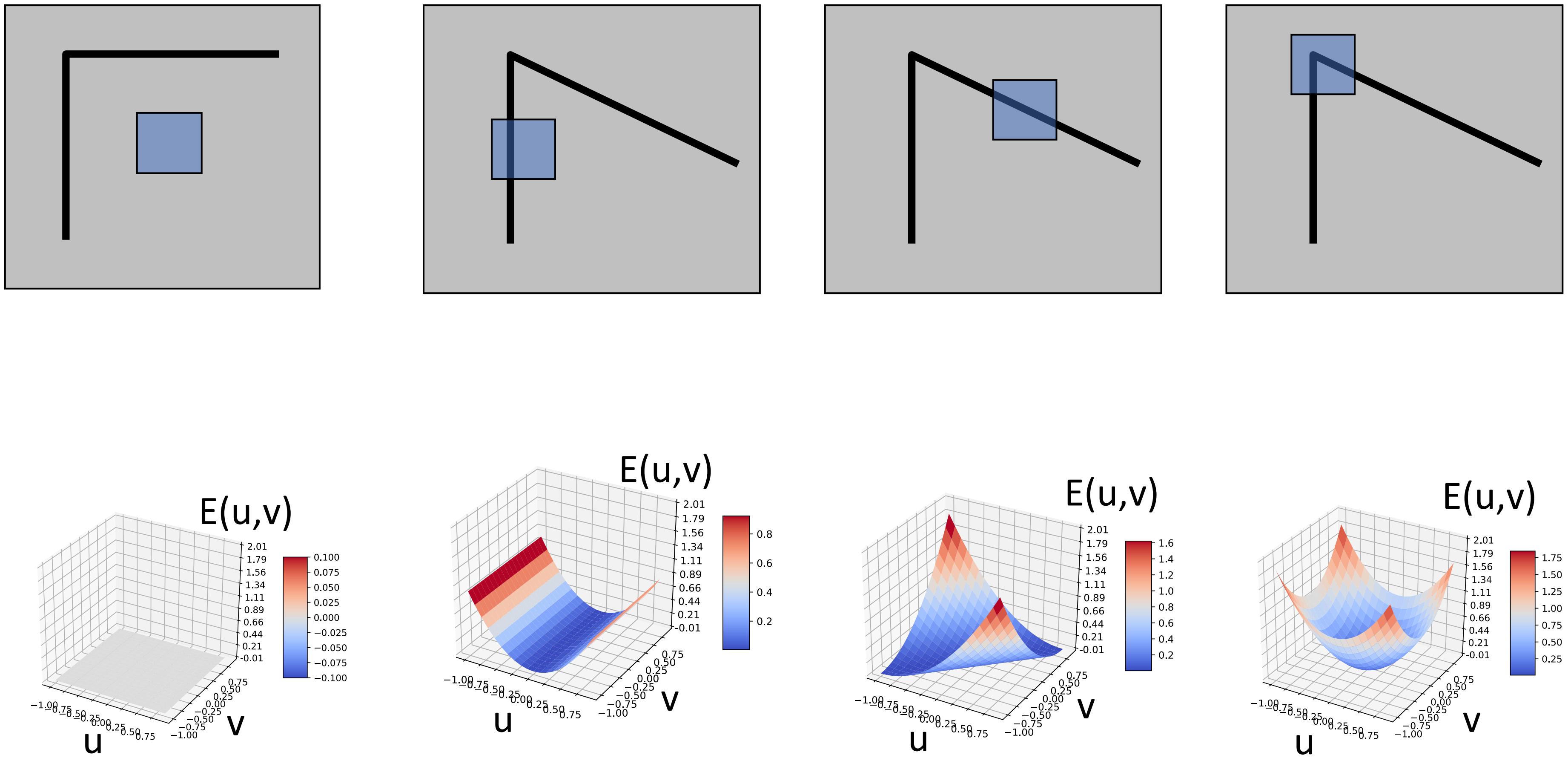
Solutions to Mx = 0 are directions for which E is 0: window can slide in this direction without changing appearance

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

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





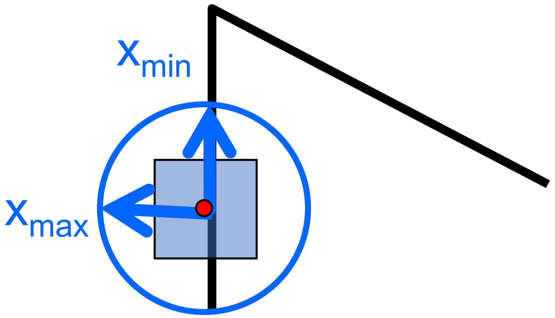
E(u,v)
u
v
E(u,v)
u
v
E(u,v)
u
v
E(u,v)
u
v

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

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



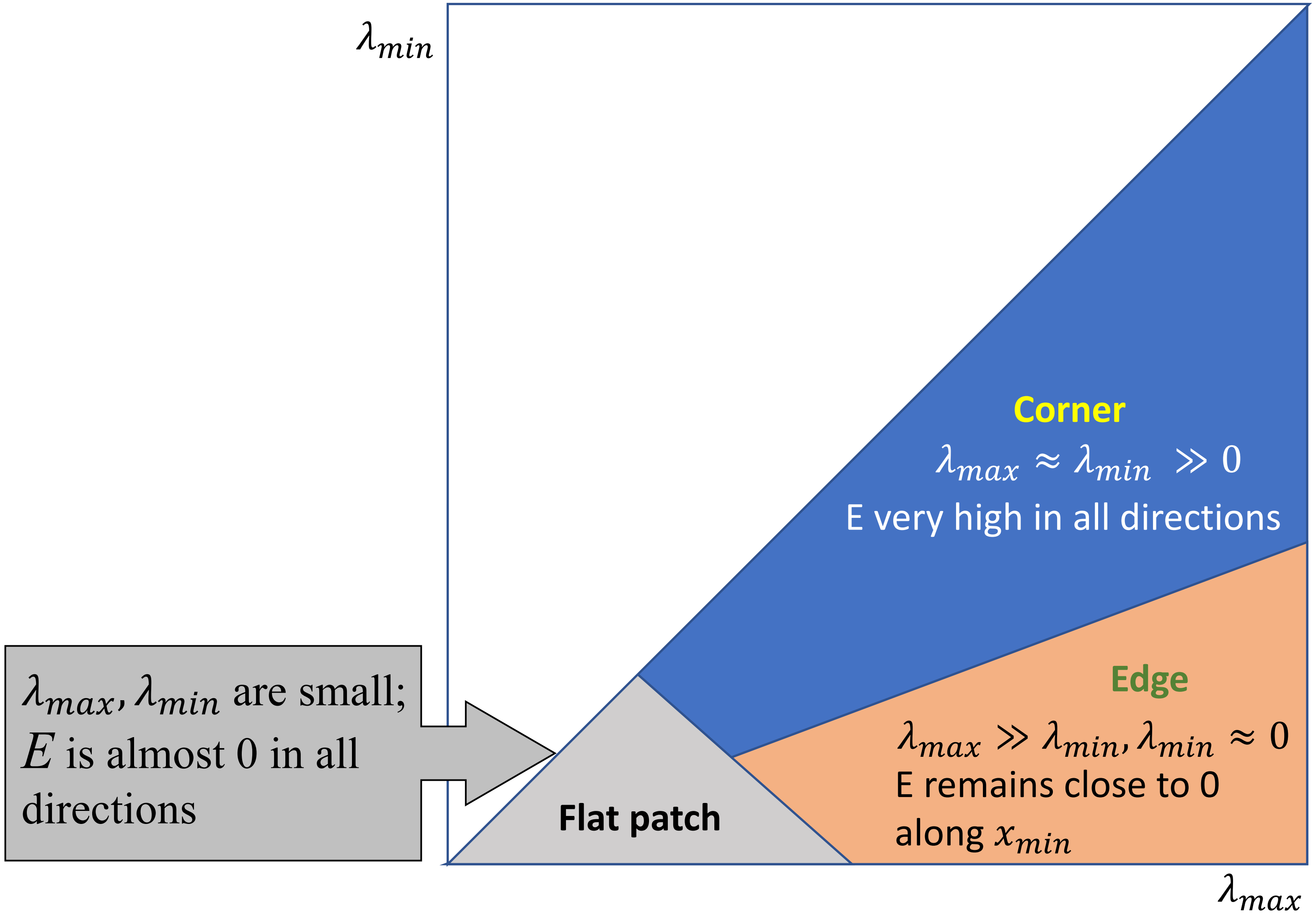
M $x_{\max} = \lambda_{\max} x_{\max}$

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



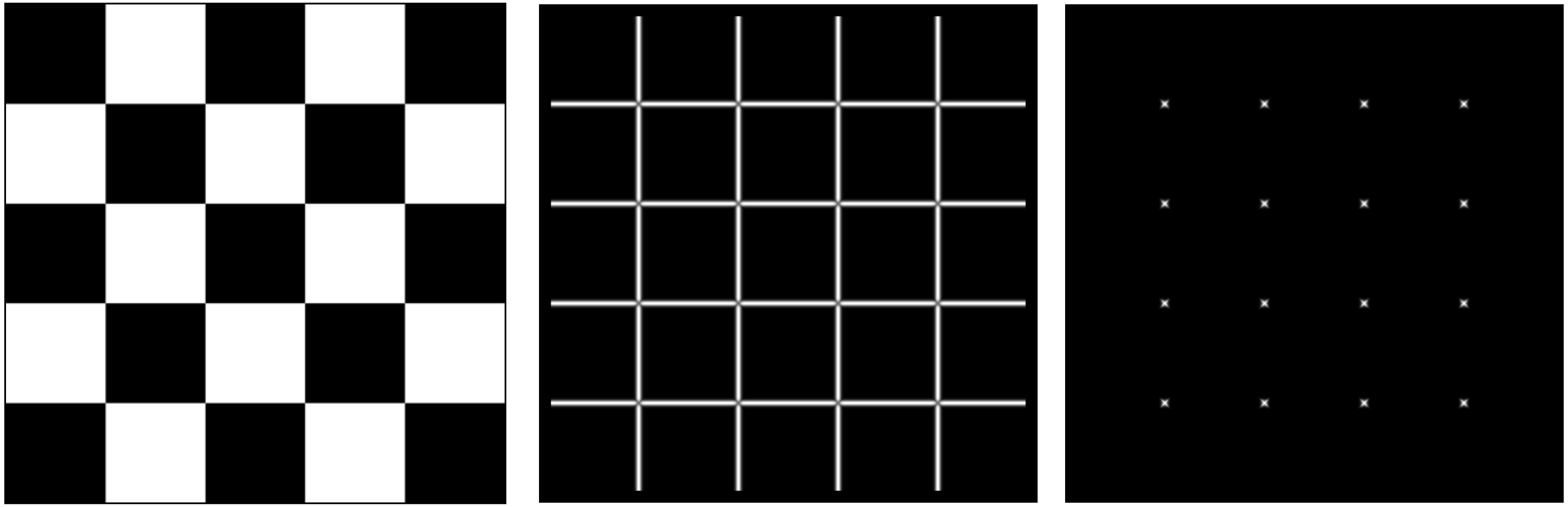
Corner detection: the math

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

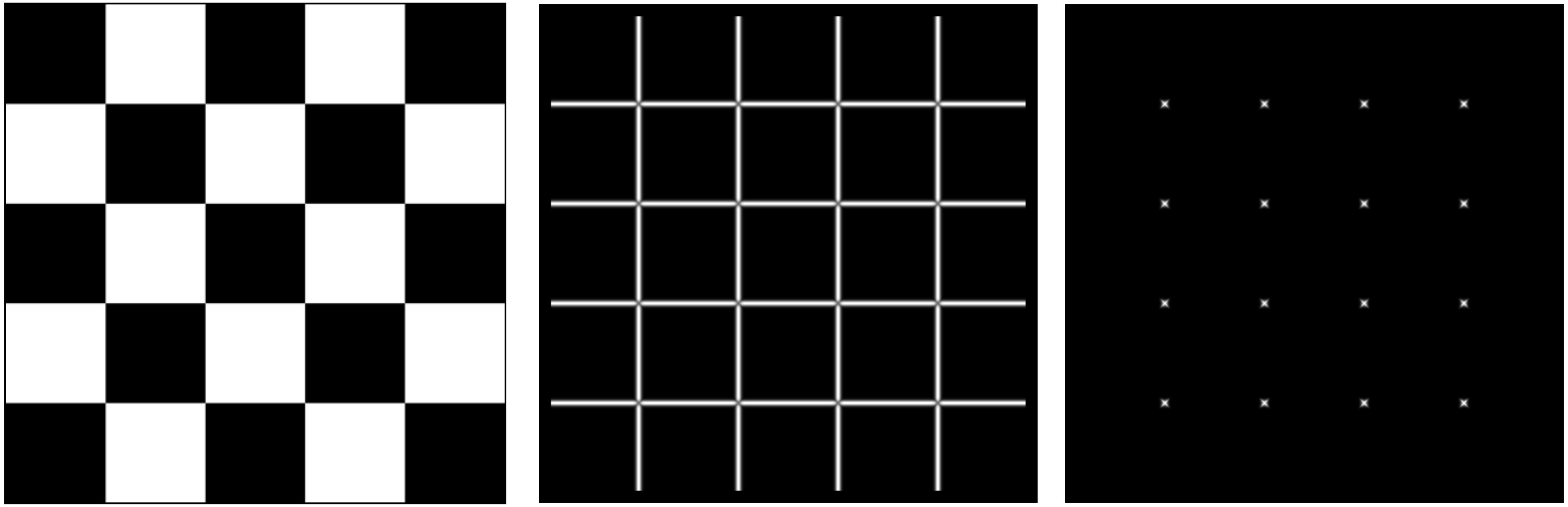


I $\qquad$ $\lambda_{\max}$ $\qquad$ $\lambda_{\min}$

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

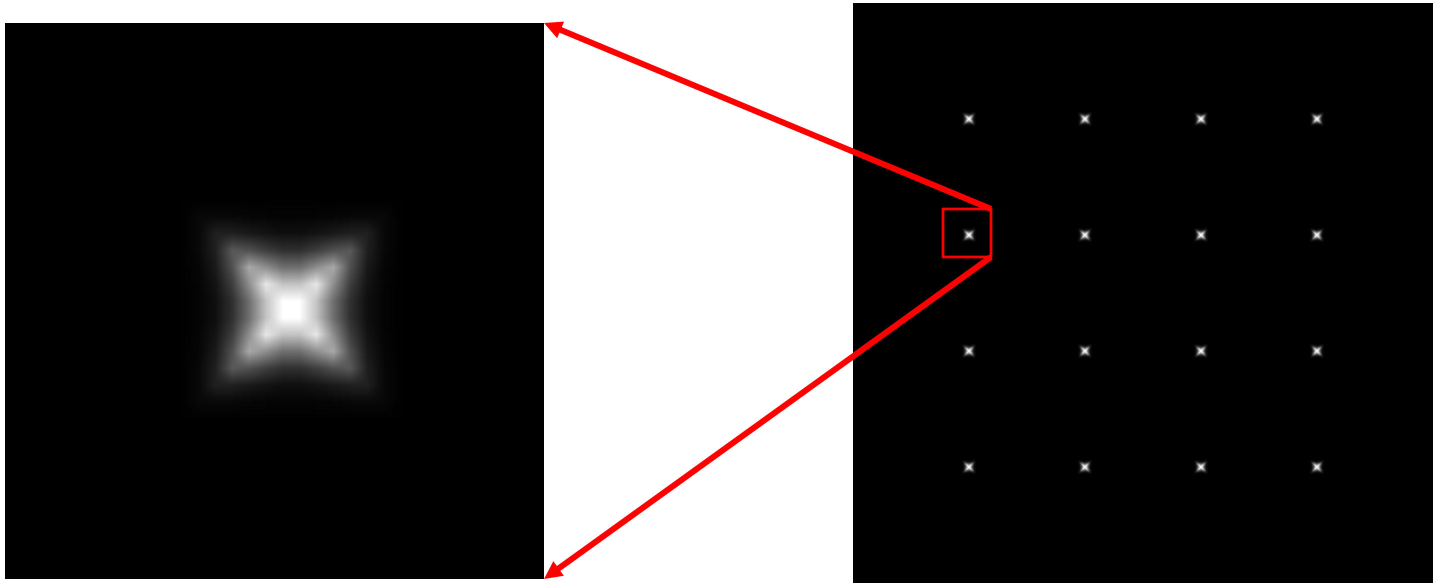


I $\quad$ $\lambda_{\max}$ $\quad$ $\lambda_{\min}$

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

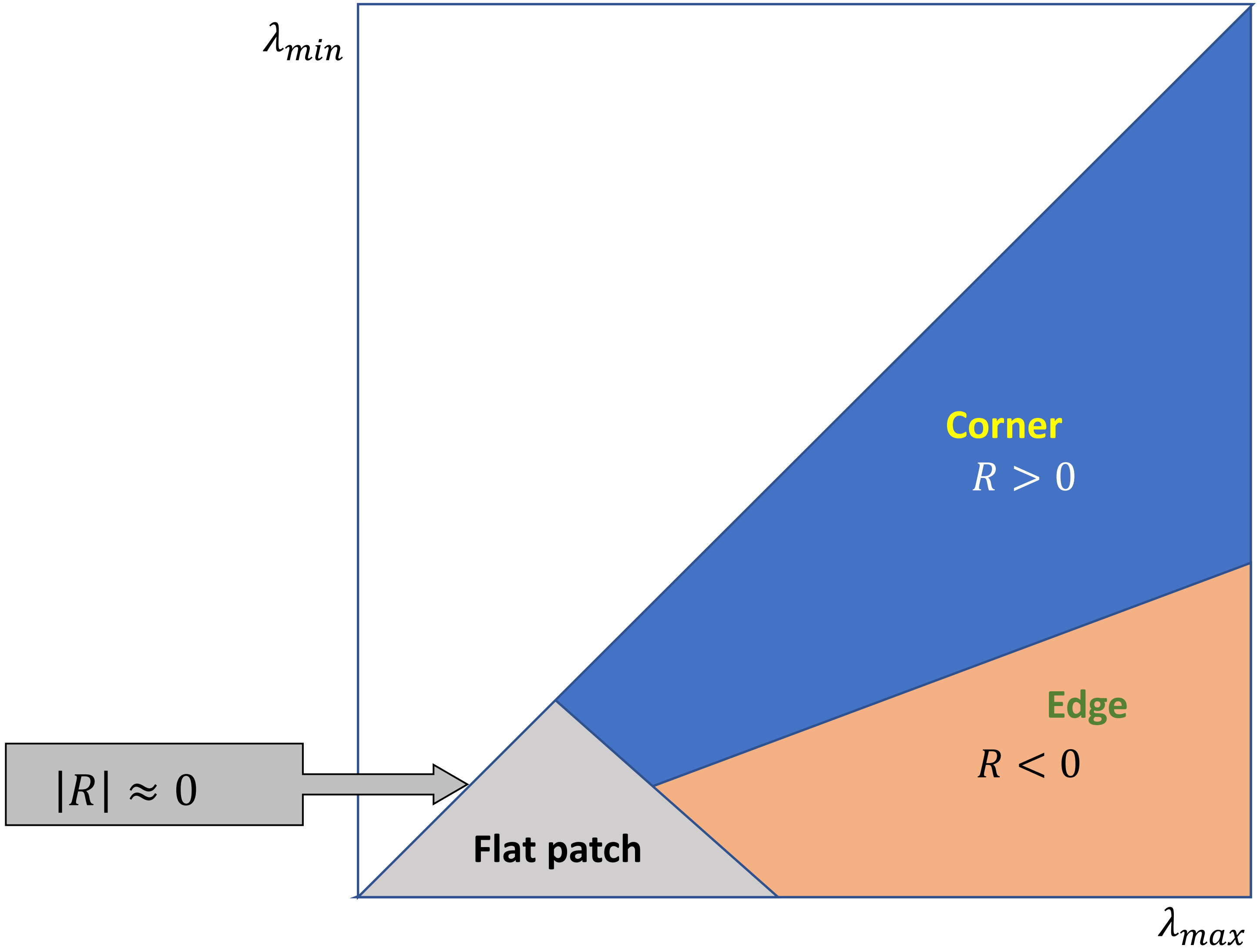
$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$

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

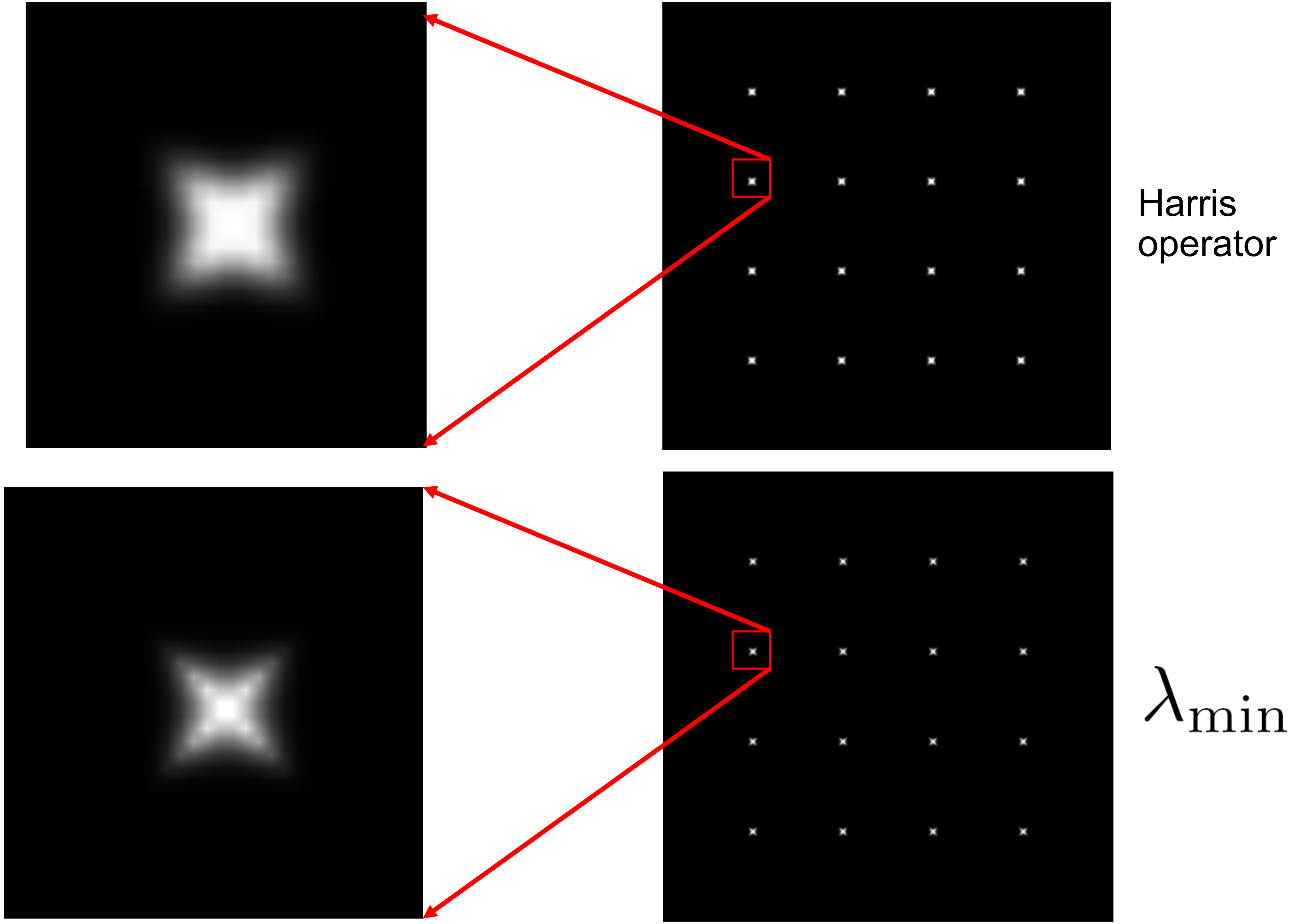
- The *trace* is the sum of the diagonals, i.e., $trace(H) = h_{11} + h_{22}$
- Very similar to λ_{min} but less expensive (no square root)
- Called the "Harris Corner Detector" or "Harris Operator"
    - Actually the Noble variant of the Harris Corner Detector
- Lots of other detectors, this is one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

Corner response function

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

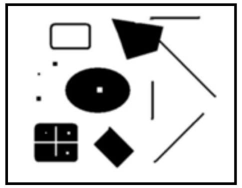


The Harris operator



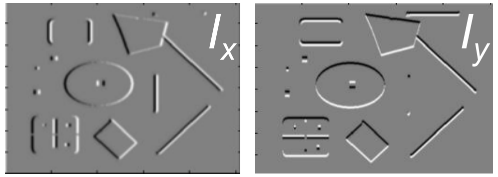
Harris Detector [Harris88]

- Second moment matrix



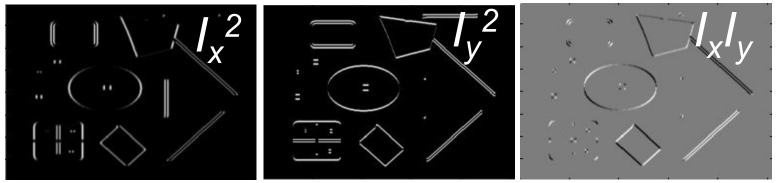
$$\mu(\sigma_I,\sigma_D)=g(\sigma_I)*\begin{bmatrix} I_x^2(\sigma_D) & I_xI_y(\sigma_D) \\ I_xI_y(\sigma_D) & I_y^2(\sigma_D) \end{bmatrix}$$

1. Image derivatives (optionally, blur first)



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

2. Square of derivatives

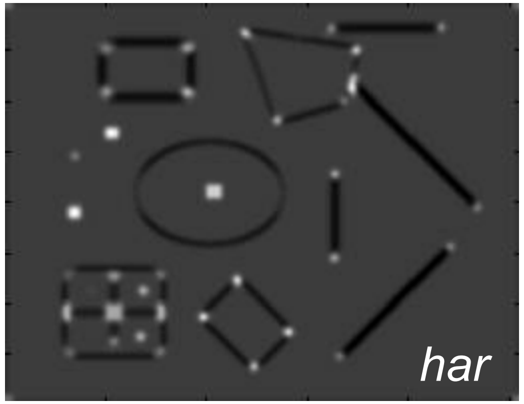


3. Gaussian filter $g(\sigma_I)$

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



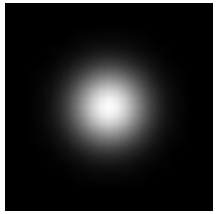
Weighting the derivatives

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

$$H = \sum_{(x,y) \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

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

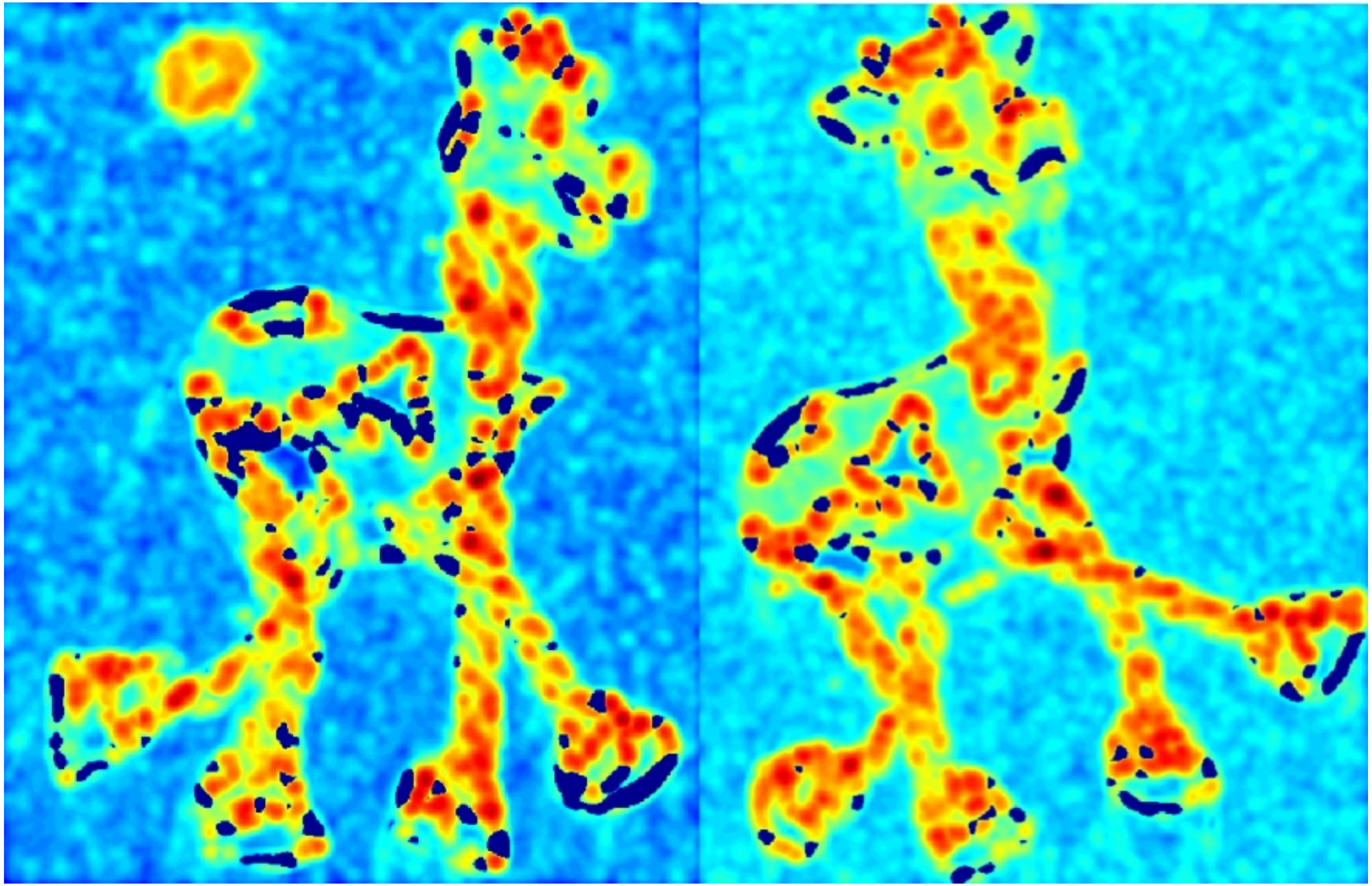
$$H = \sum_{(x,y) \in W} w_{x,y} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



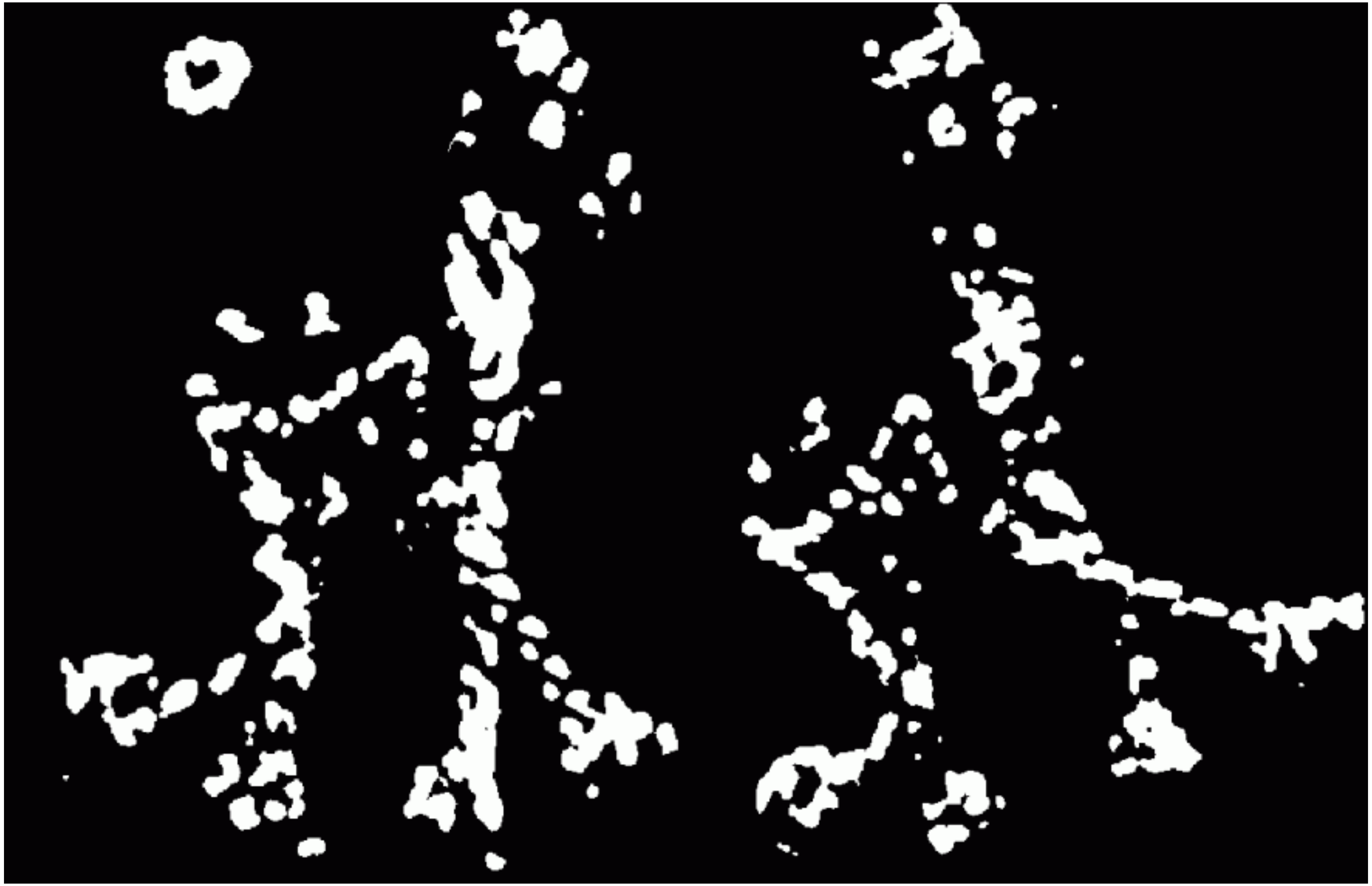
$w_{x,y}$

Harris detector example

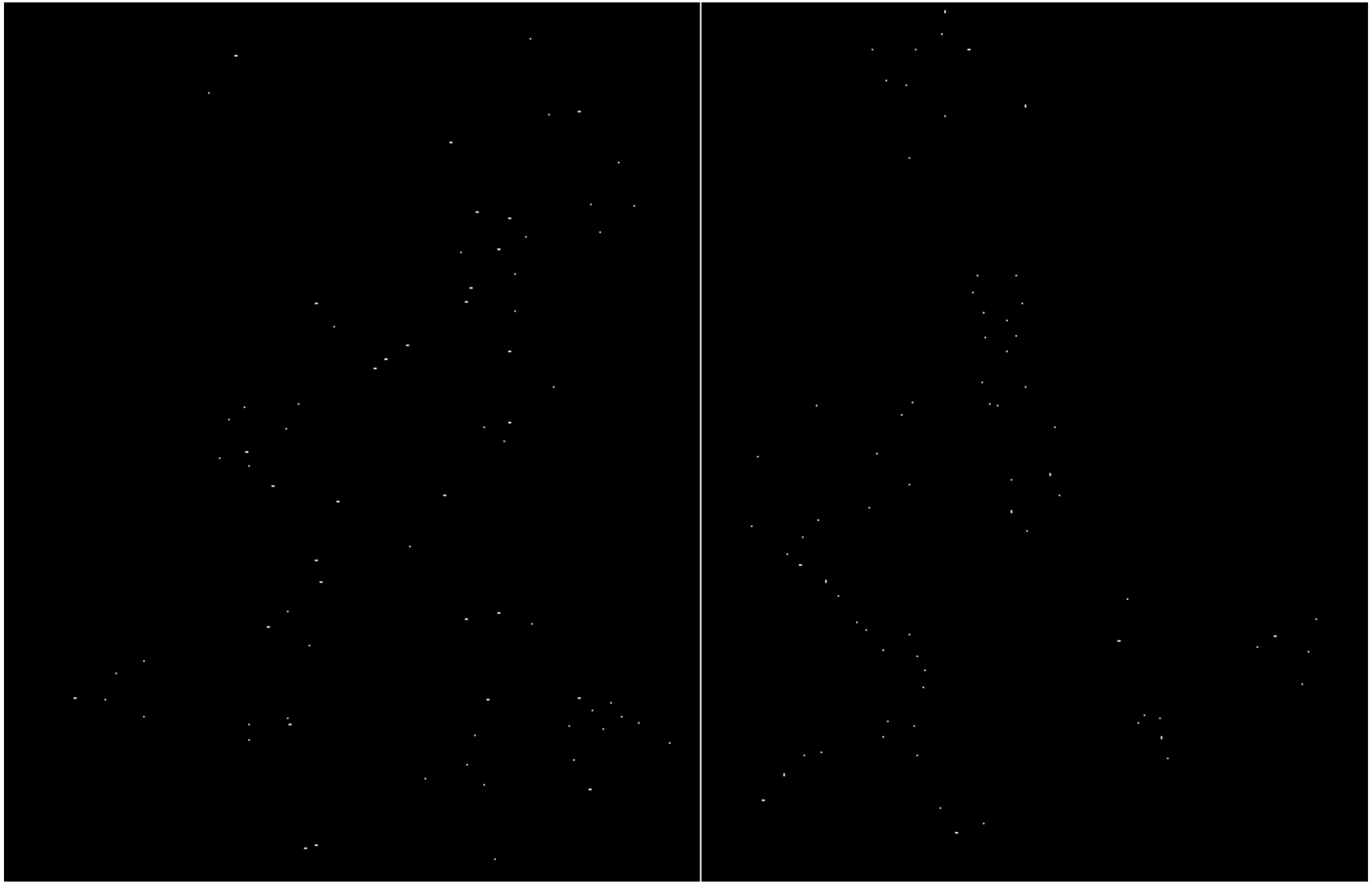
f value (red high, blue low)



Threshold (f > value)



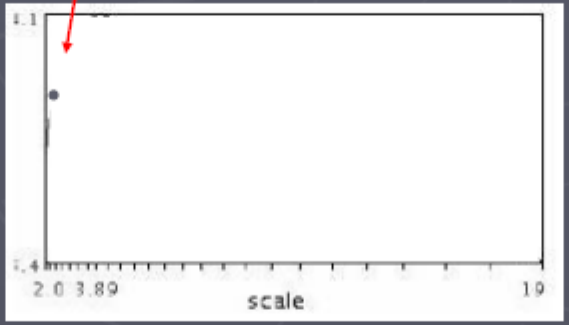
Find local maxima of f



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

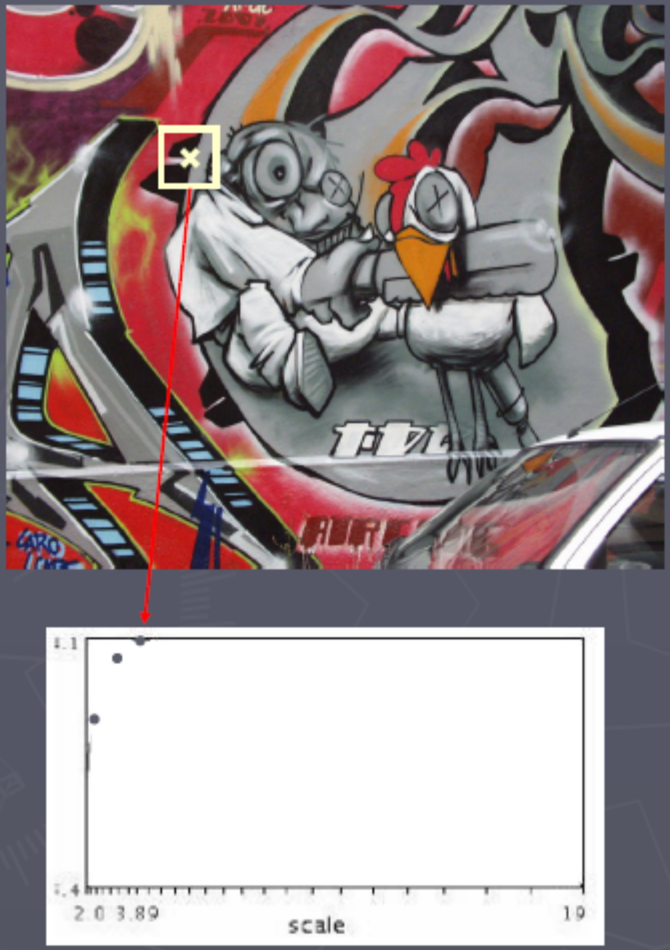
Function responses for increasing scale

Scale trace (signature)

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

Automatic scale selection

Function responses for increasing scale
Scale trace (signature)

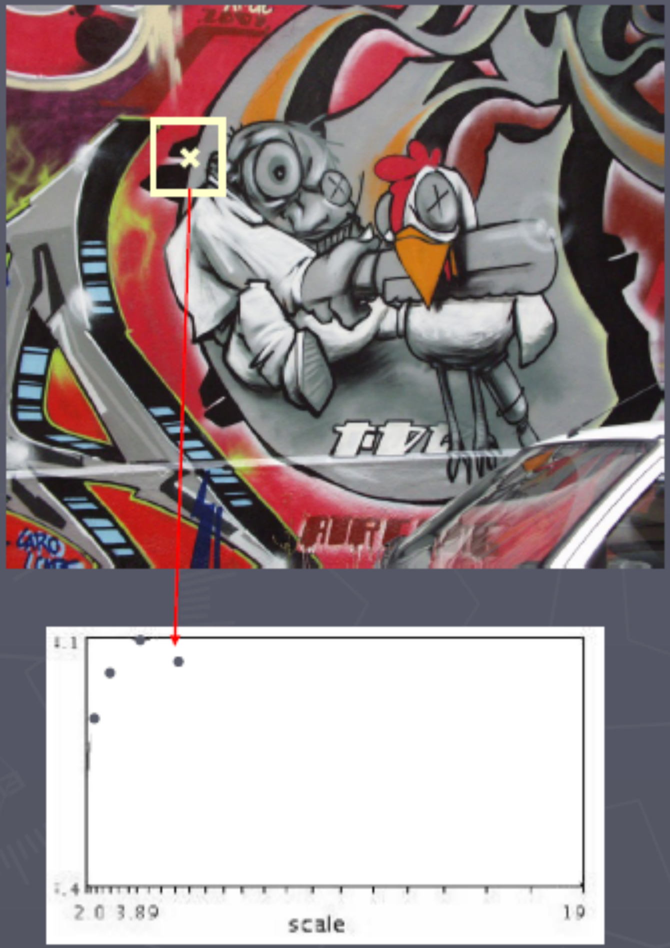


$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection

Function responses for increasing scale

Scale trace (signature)

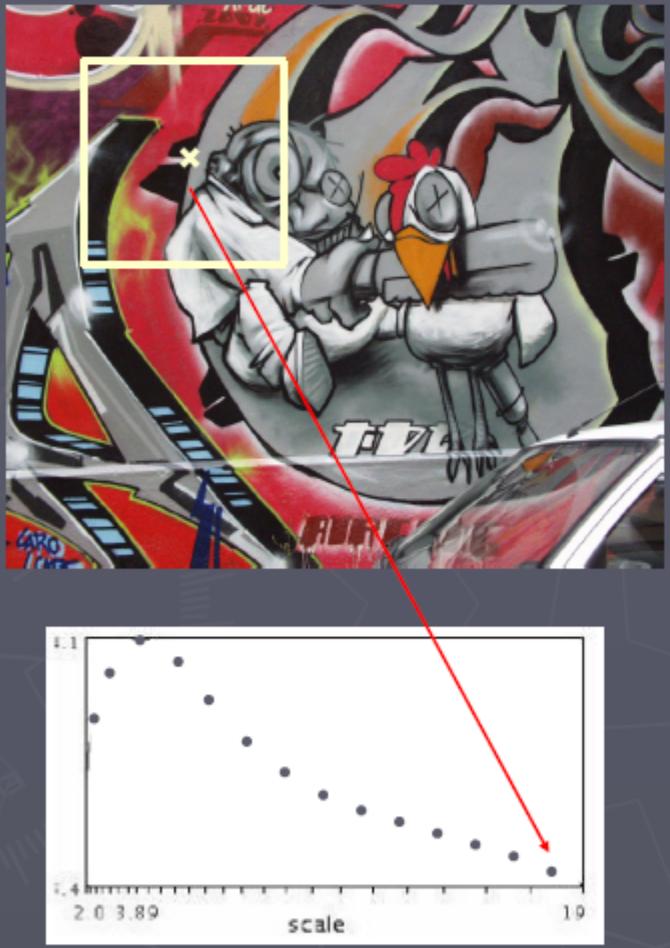


$$f(I_{i_1 \ldots i_m}(x,\sigma))$$

Automatic scale selection

Function responses for increasing scale

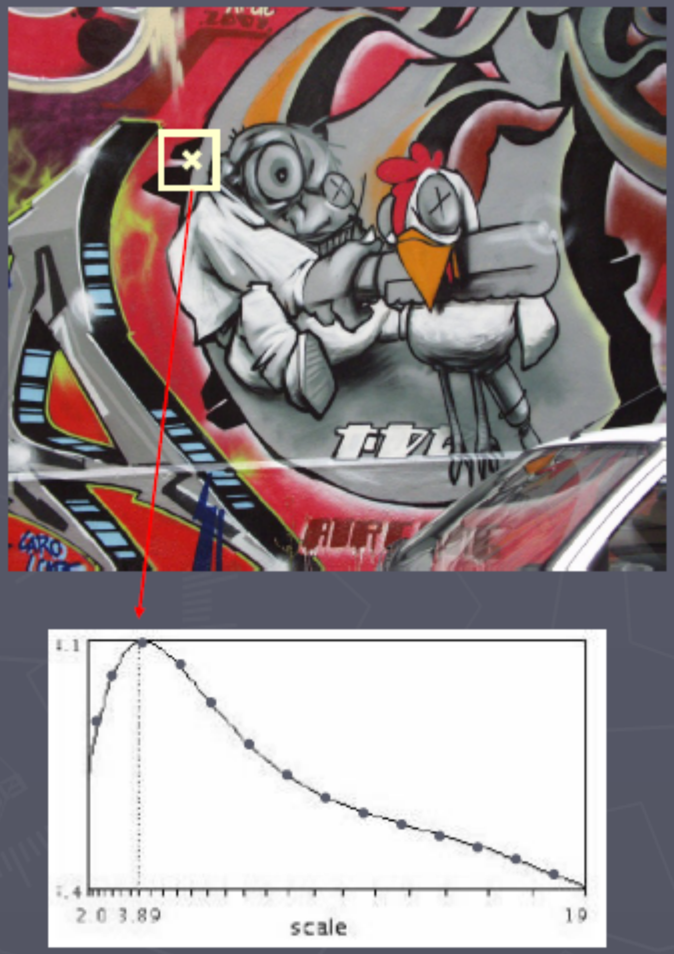
Scale trace (signature)



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

Automatic scale selection

Function responses for increasing scale
Scale trace (signature)

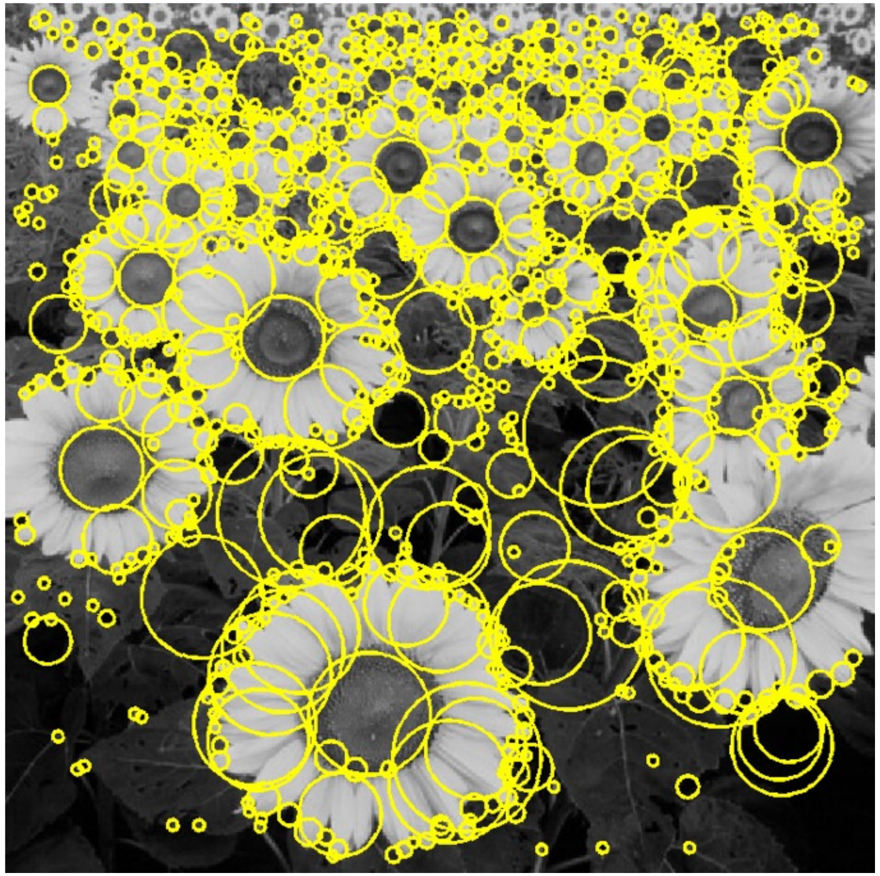


$$f(I_{i_1 \dots i_m}(x,\sigma))$$

Implementation

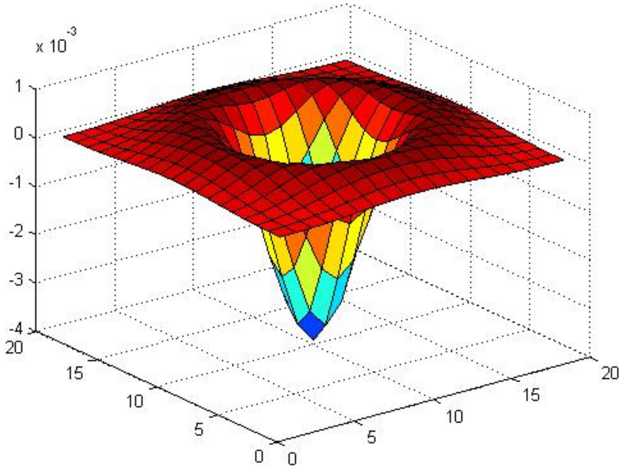
- Instead of computing *f* for larger and larger windows, we can implement using a fixed window size with a Gaussian pyramid

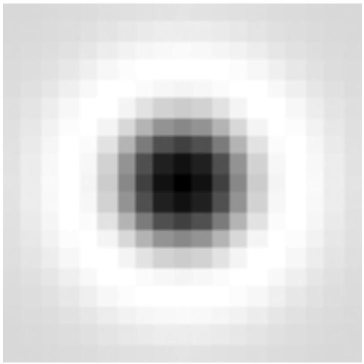
Feature extraction: Corners and blobs



Another common definition of *f*

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



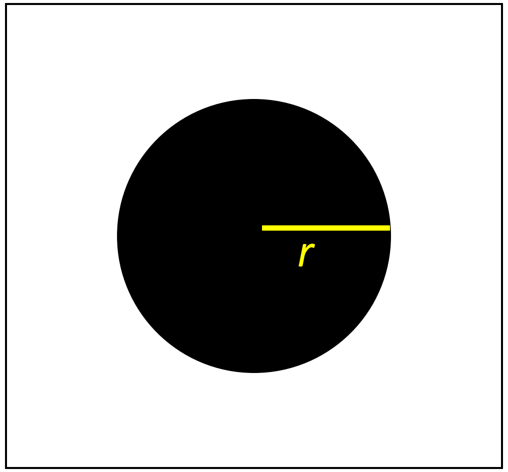


$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

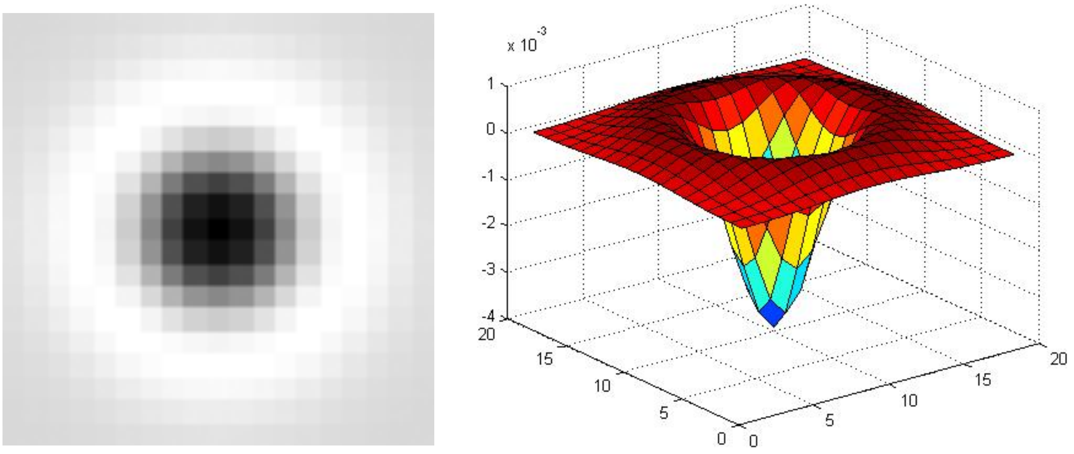
(very similar to a Difference of Gaussians (DoG) – i.e. a Gaussian minus a slightly smaller Gaussian)

Scale selection

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



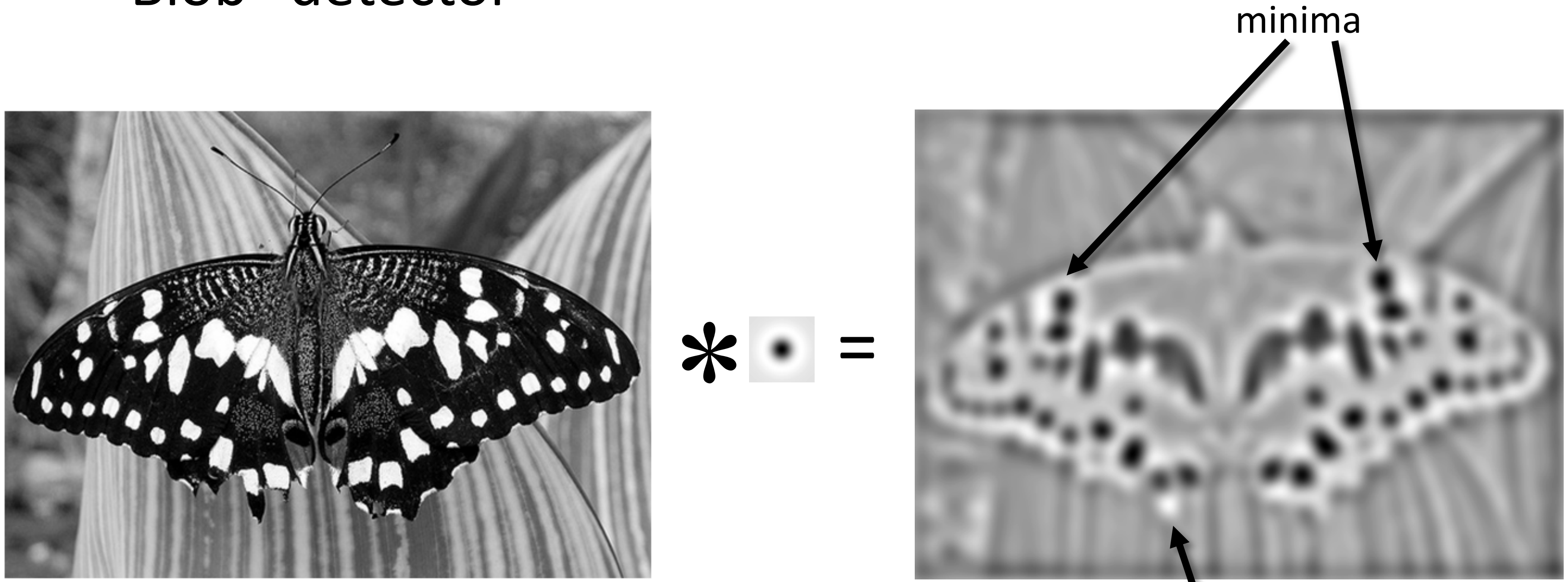
image



Laplacian

Laplacian of Gaussian

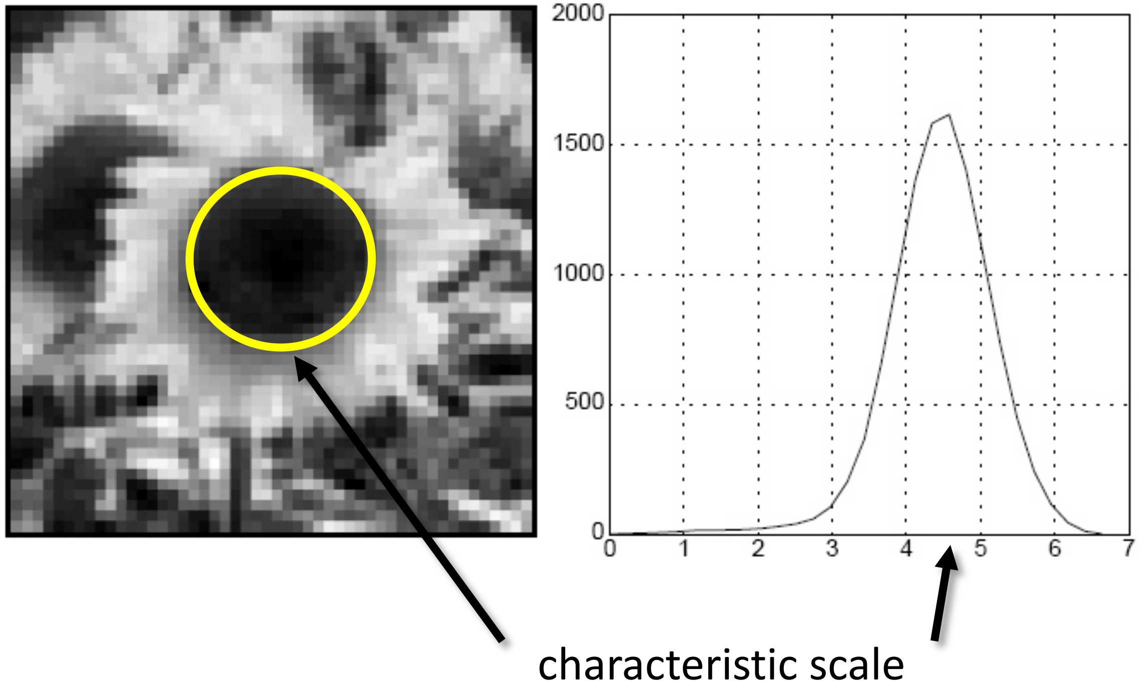
- "Blob" detector



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

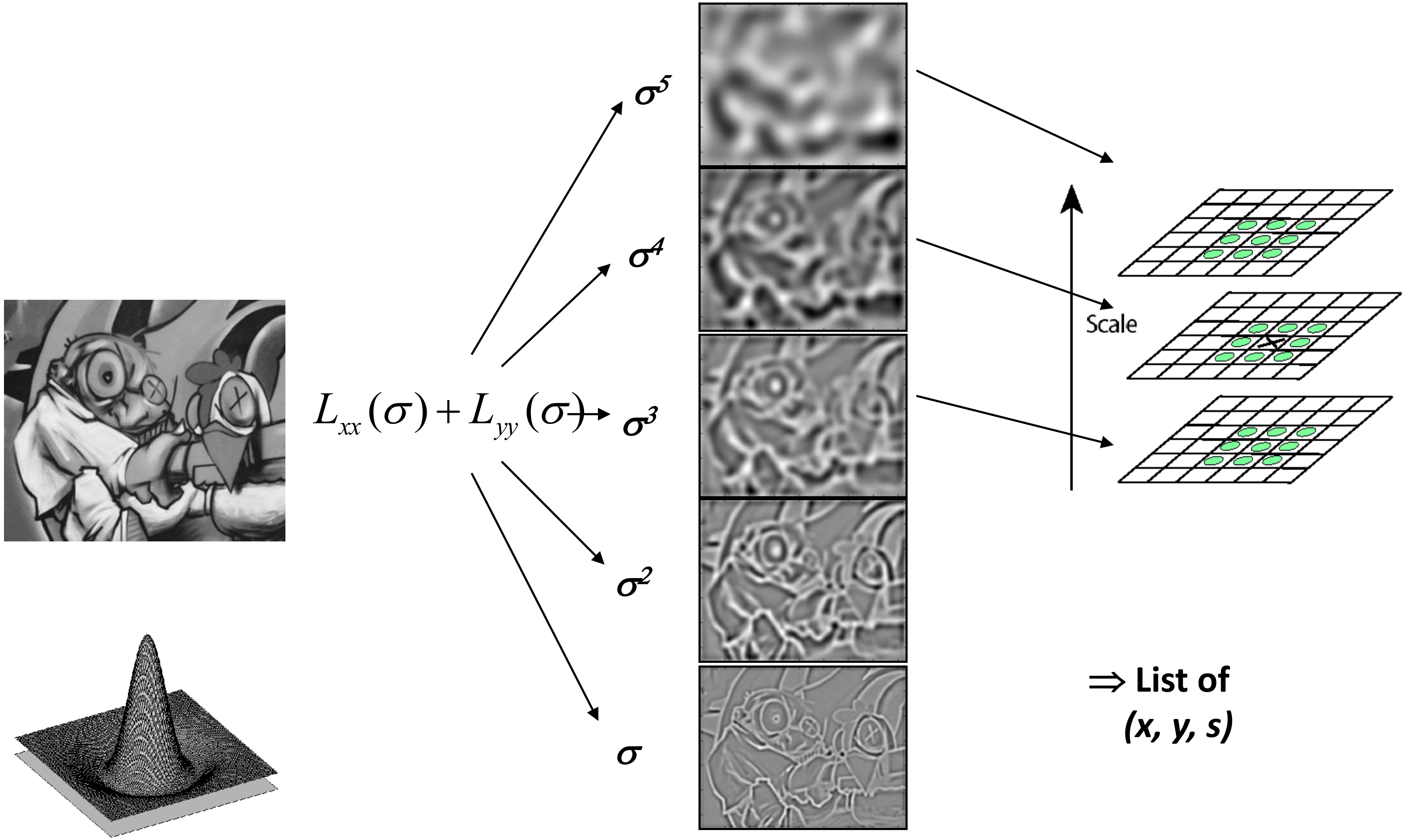
Characteristic scale

- 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)***

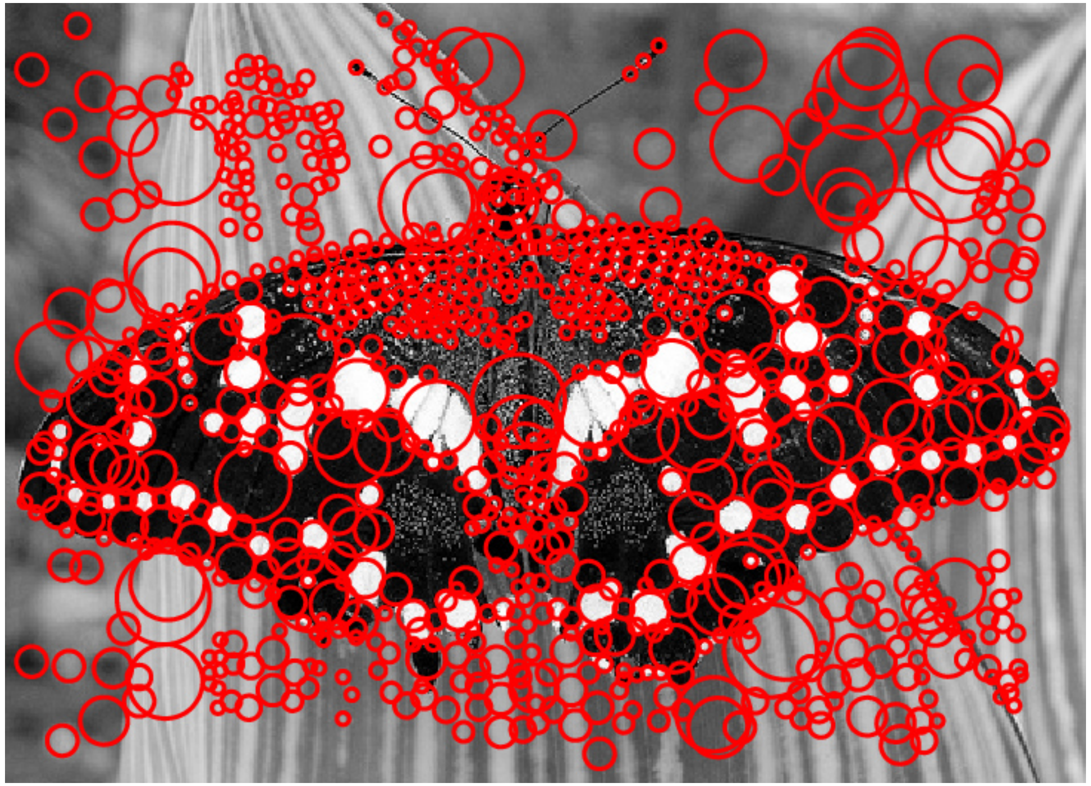
K. Grauman, B. Leibe

Scale-space blob detector: Example

Scale-space blob detector: Example

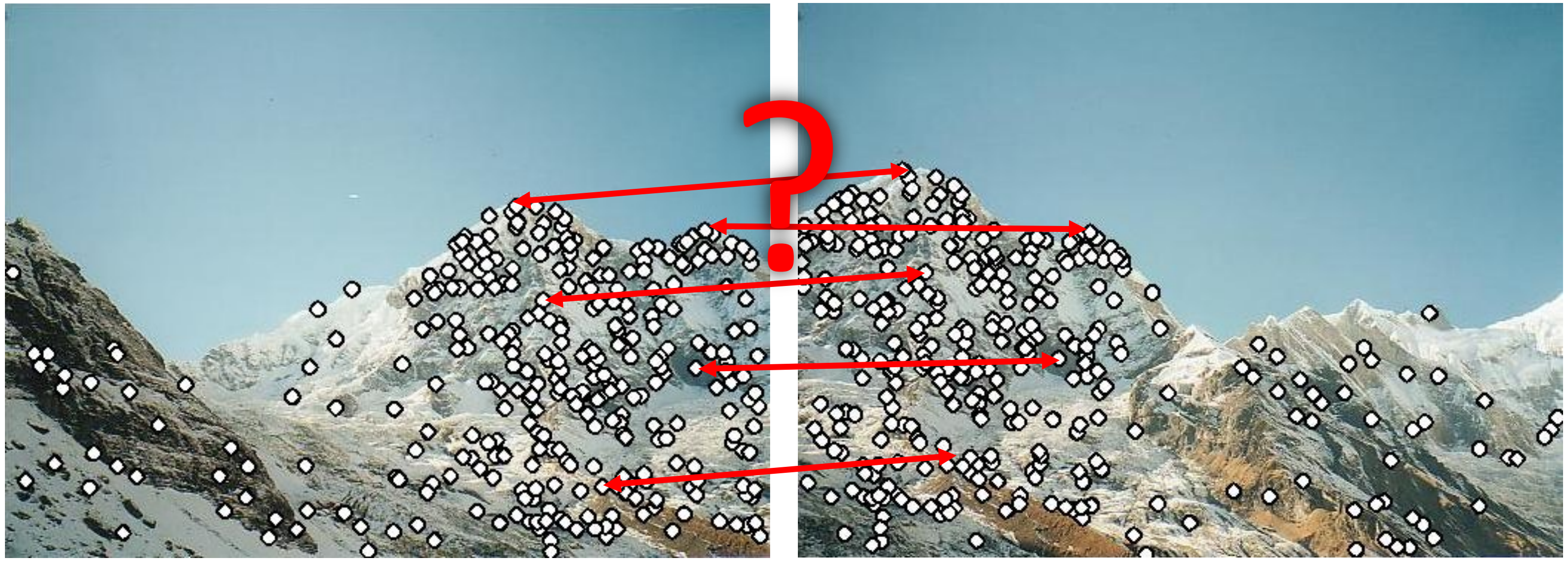
sigma = 11.9912

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?