CS5670: Computer Vision

Noah Snavely

Lecture 4: Harris corner detection

Reading

- Szeliski: 4.1

Announcements

- Project 1 (Hybrid Images) code due next Wednesday, Feb 14, by 11:59pm
- Artifacts due Friday, Feb 16, by 11:59pm

- Office hour rooms coming soon

- Quiz this Wednesday, 2/7 at the beginning of class (10 minutes)

Last time

- Sampling & interpolation
- Key points:
  - **Downsampling an image** can cause aliasing. Better is to blur ("pre-filter") to remote high frequencies then downsample
  - If you repeatedly blur and downsample by 2x, you get a Gaussian pyramid
  - **Upsampling an image** requires interpolation. This can be posed as convolution with a "reconstruction kernel"

Image interpolation

Original image:  x 10

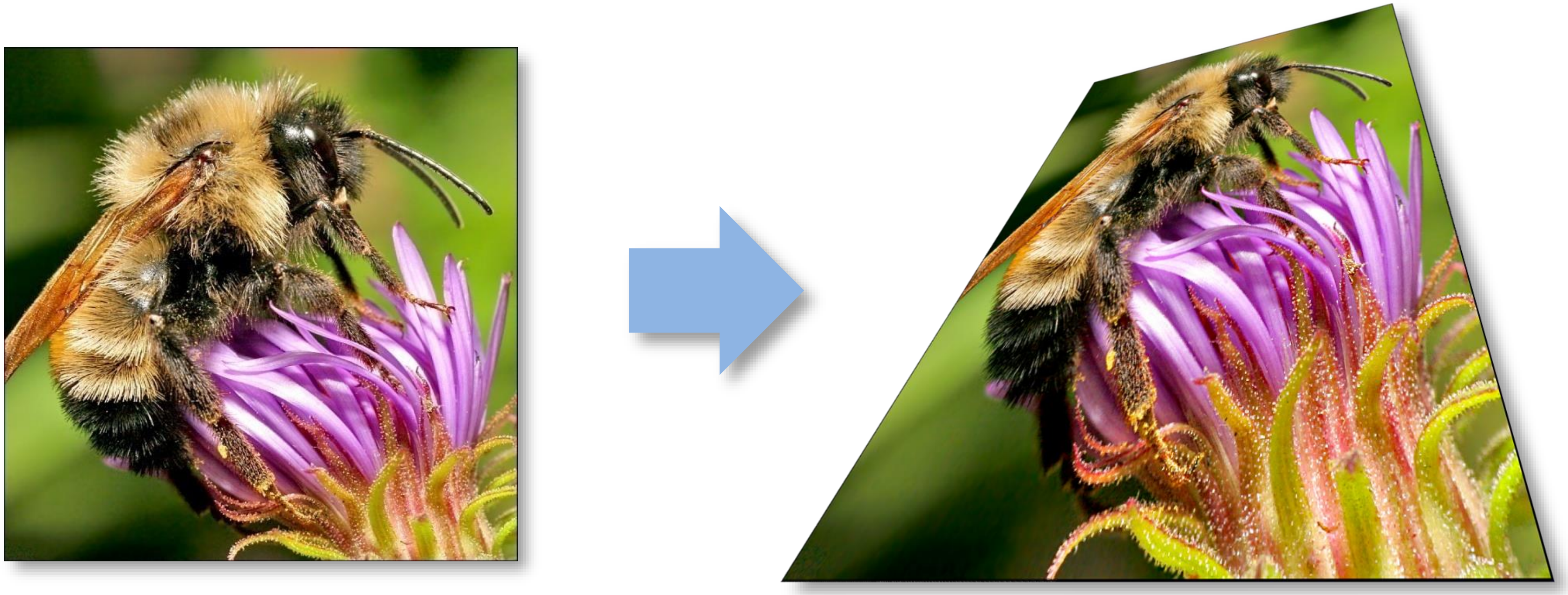
Nearest-neighbor interpolation

Bilinear interpolation

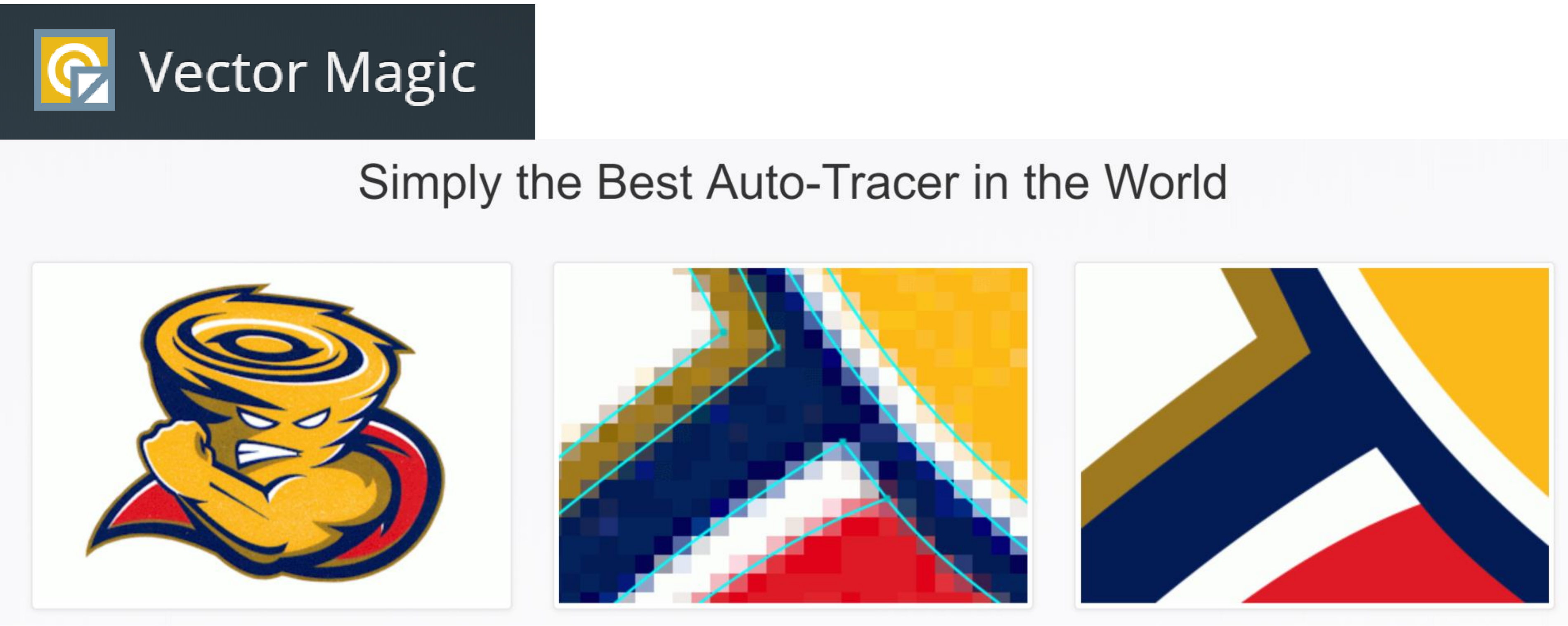
Bicubic interpolation

Image interpolation

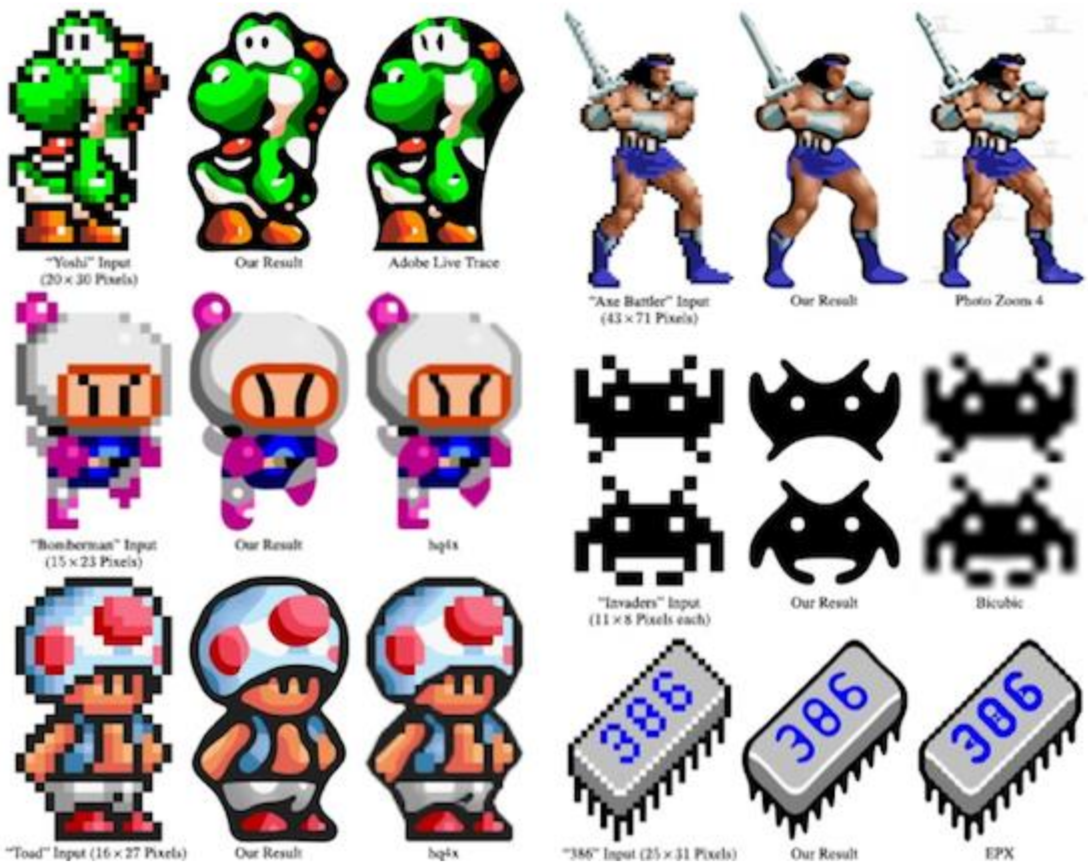
Also used for *resampling*



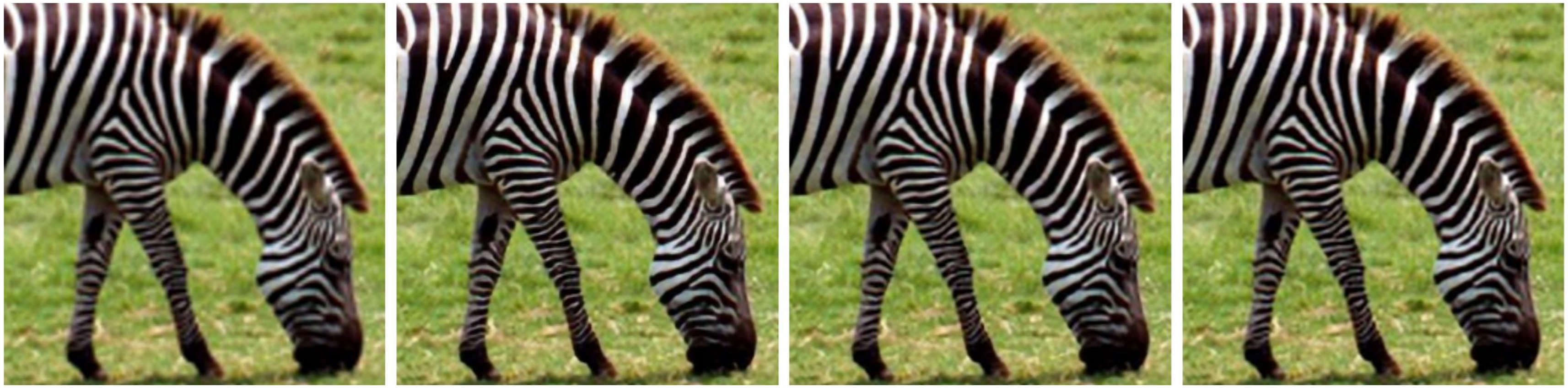
Raster-to-vector graphics



Depixelating Pixel Art



Modern methods



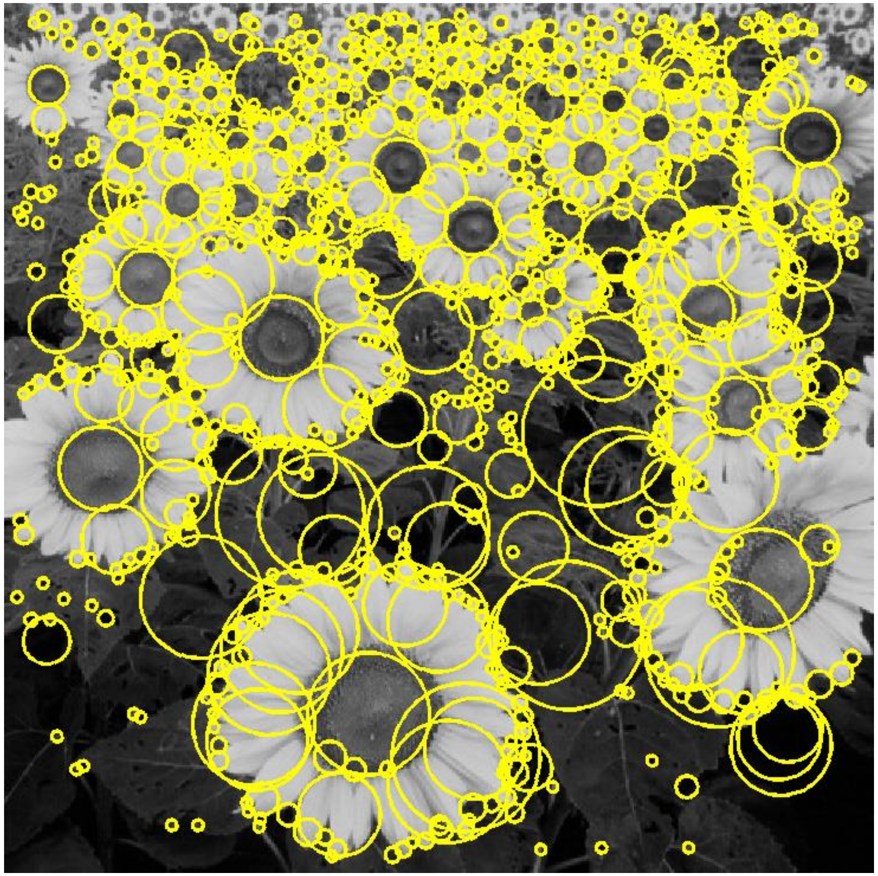
(a) Bicubic (b) SRCNN (c) A+ (d) RAISR

(e) Bicubic (f) SRCNN (g) A+ (h) RAISR

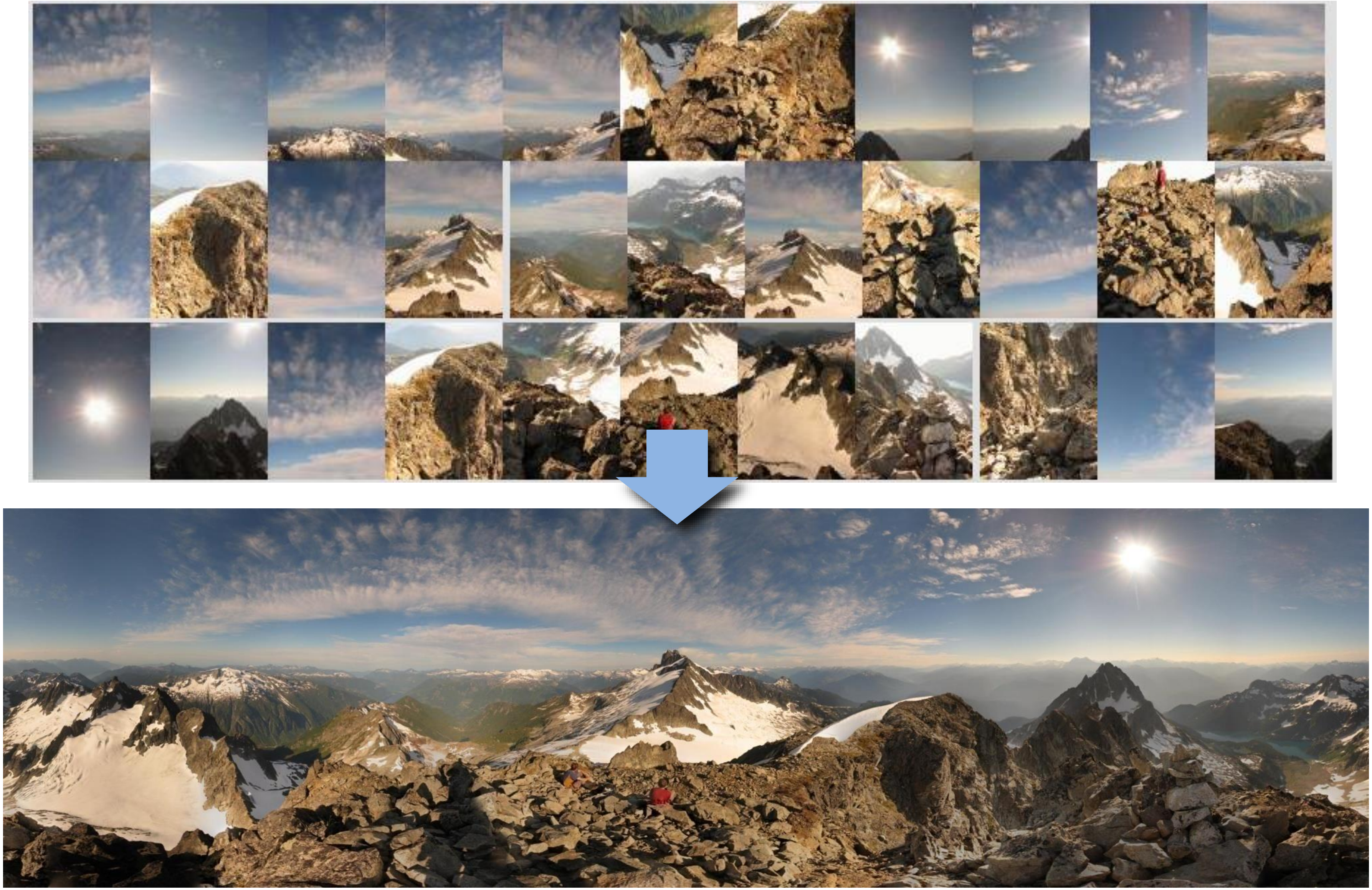
From Romano, et al: RAISR: Rapid and Accurate Image Super Resolution, https://arxiv.org/abs/1606.01299

Questions?

Feature extraction: Corners and blobs



Motivation: Automatic panoramas



Credit: Matt Brown

Motivation: Automatic panoramas

GigaPan
http://gigapan.com/

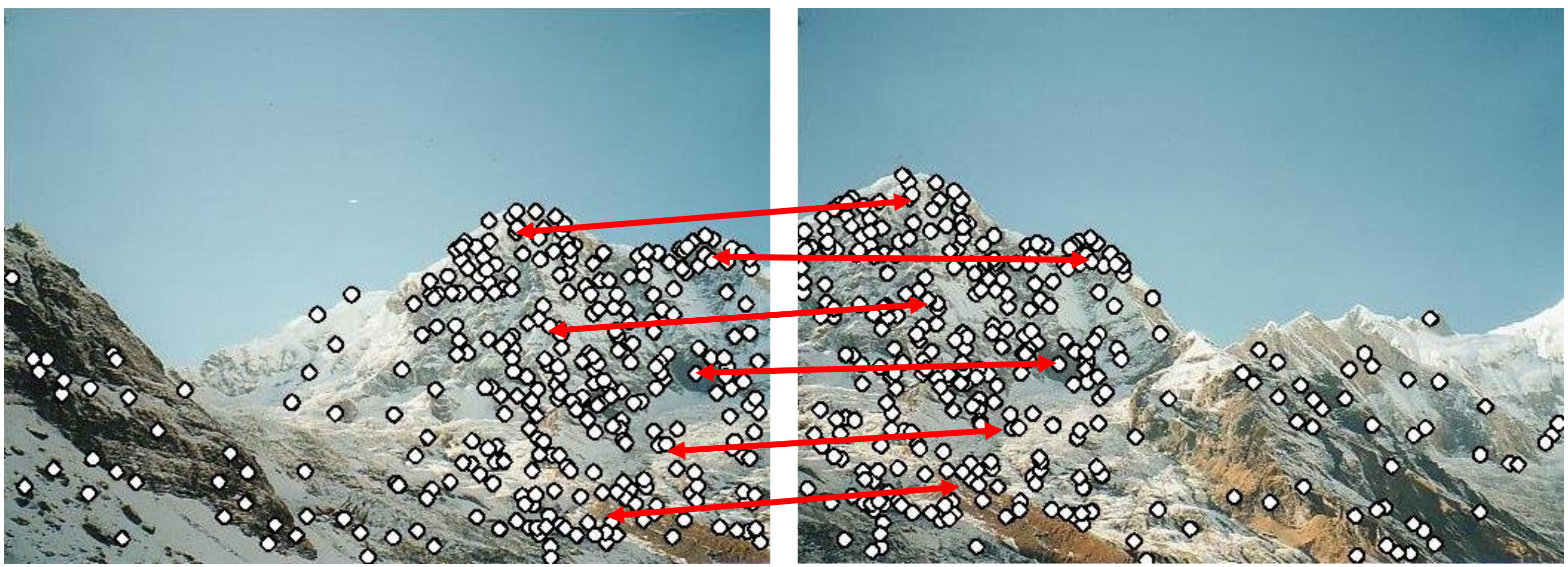
Also see Google Zoom Views:
https://www.google.com/culturalinstitute/beta/project/gigapixels

Why extract features?

- Motivation: panorama stitching
  - We have two images – how do we combine them?

Why extract features?

- Motivation: panorama stitching
  - We have two images – how do we combine them?



Step 1: extract features
Step 2: match features

Why extract features?

- Motivation: panorama stitching
  - We have two images – how do we combine them?

Step 1: extract features
Step 2: match features
Step 3: align images

Application: Visual SLAM

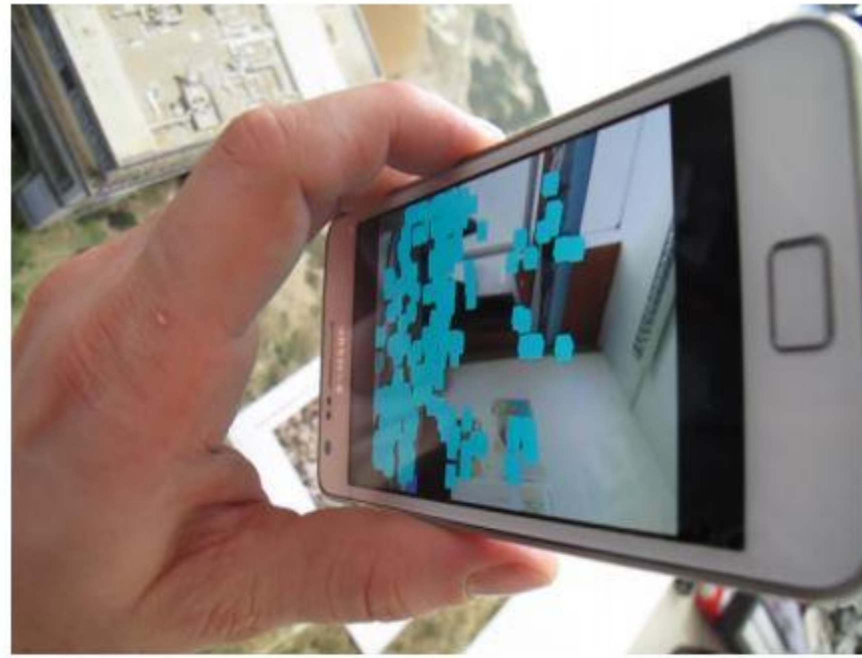


Image matching

by [Diva Sian]

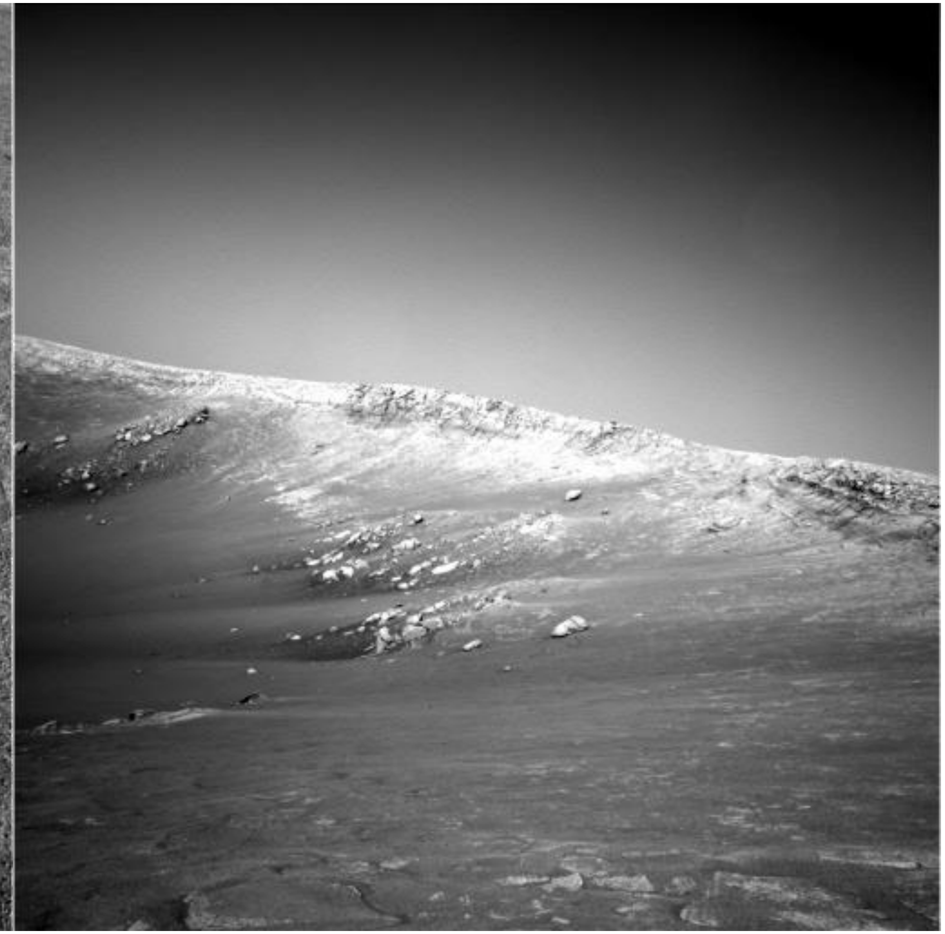
by [swashford]

Harder case

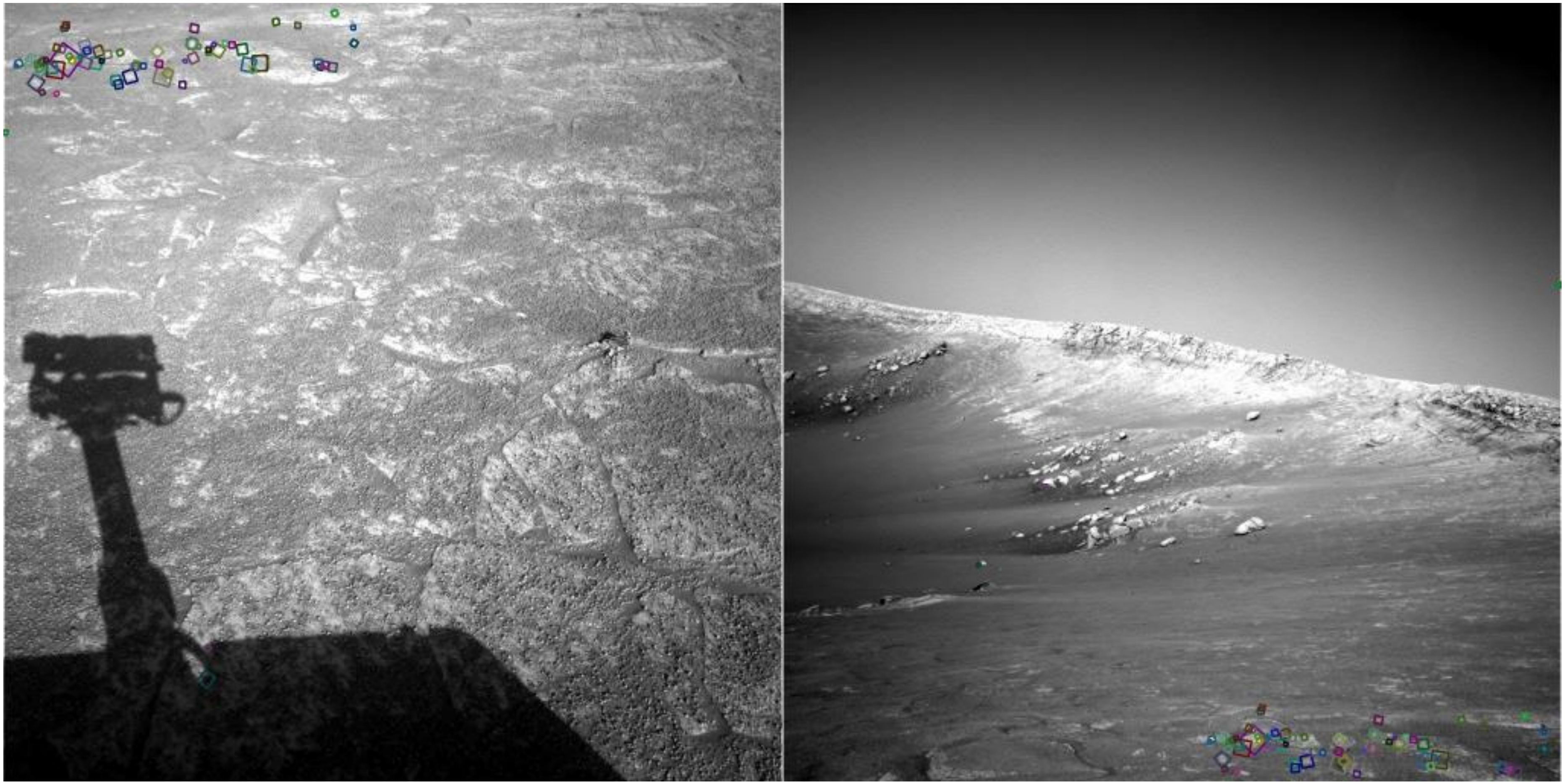
by Diva Sian

by scgbt

Harder still?

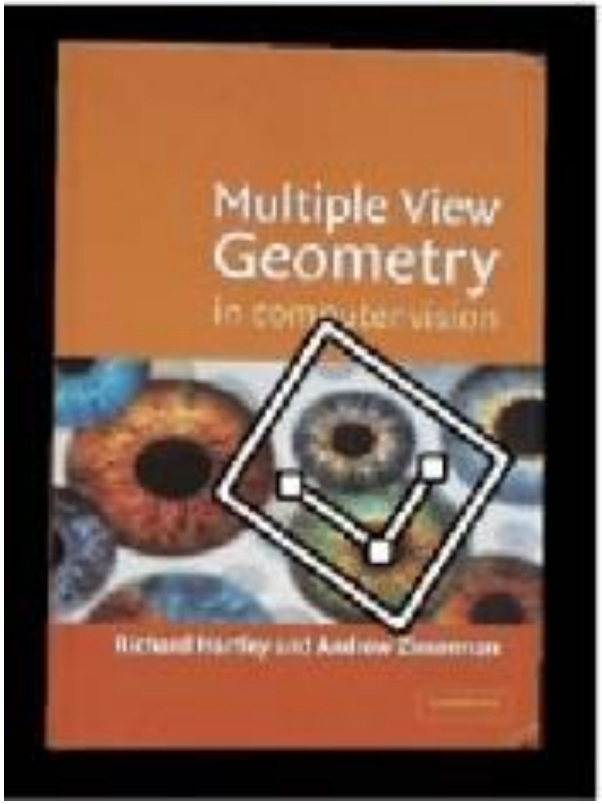


Answer below (look for tiny colored squares…)



NASA Mars Rover images
with SIFT feature matches

Feature Matching

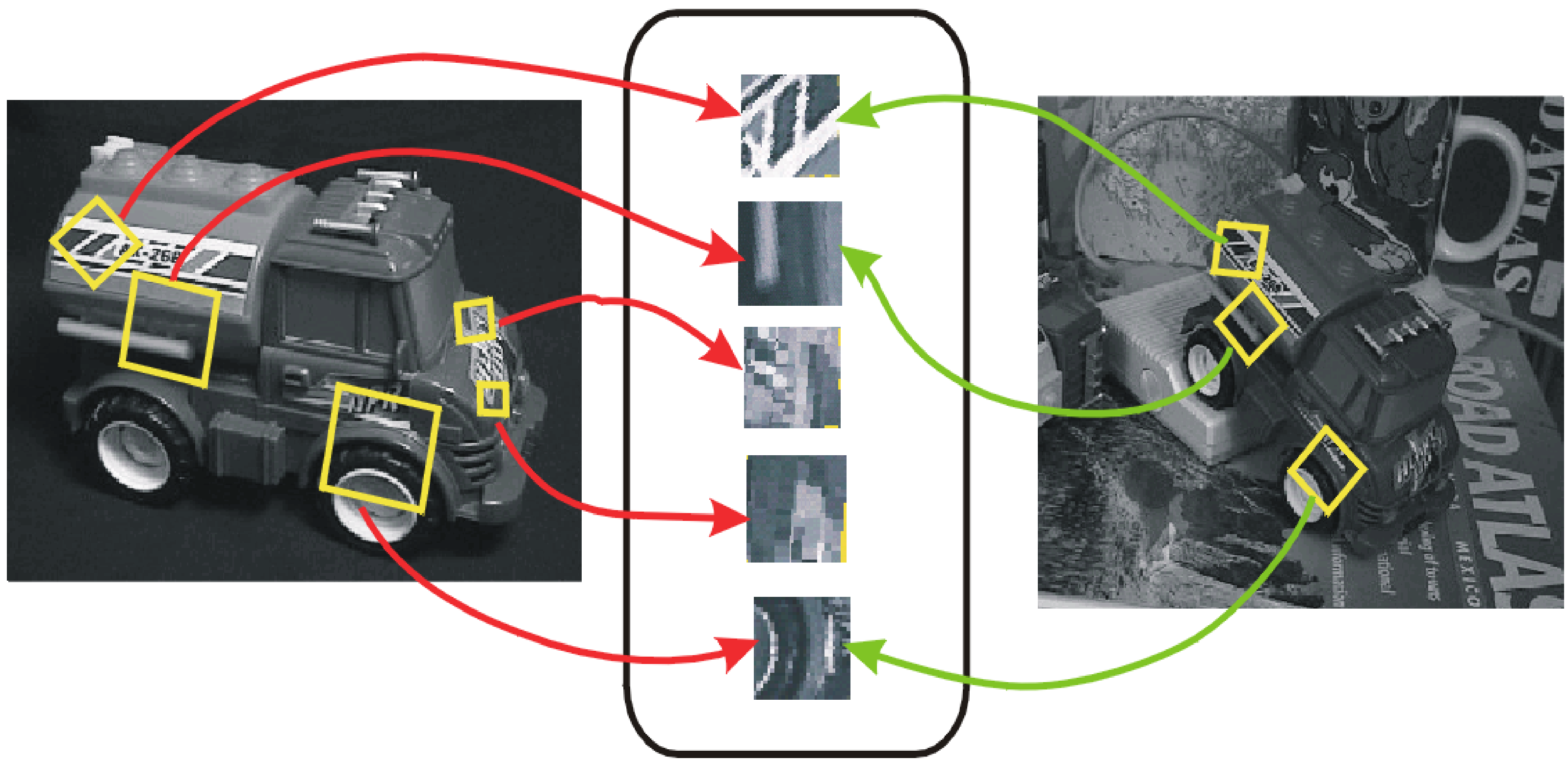


Feature Matching

Invariant local features

Find features that are invariant to transformations

- geometric invariance: translation, rotation, scale
- photometric invariance: brightness, exposure, …



Feature Descriptors

Advantages of local features

Locality
- features are local, so robust to occlusion and clutter

Quantity
- hundreds or thousands in a single image

Distinctiveness:
- can differentiate a large database of objects

Efficiency
- real-time performance achievable

More motivation…

Feature points are used for:

- Image alignment
  - (e.g., mosaics)
- 3D reconstruction
- Motion tracking
  - (e.g. for AR)
- Object recognition
- Image retrieval
- Robot navigation
- … other

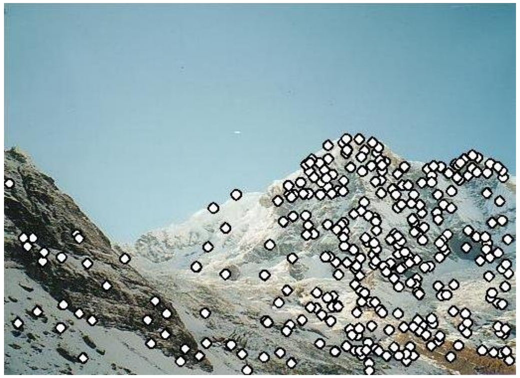
Approach

1. **Feature detection**: find it
2. **Feature descriptor**: represent it
3. **Feature matching**: match it

Feature tracking: track it, when motion

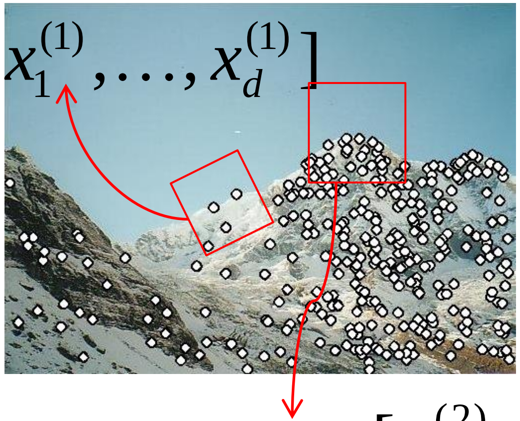
Local features: main components

1) **Detection:** Identify the interest points



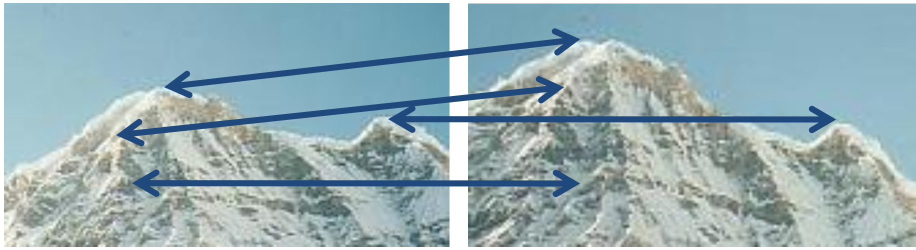
2) **Description**: Extract vector feature descriptor surrounding each interest point.

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



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

3) **Matching:** Determine correspondence between descriptors in two views



Kristen Grauman

What makes a good feature?

Want uniqueness

Look for image regions that are unusual

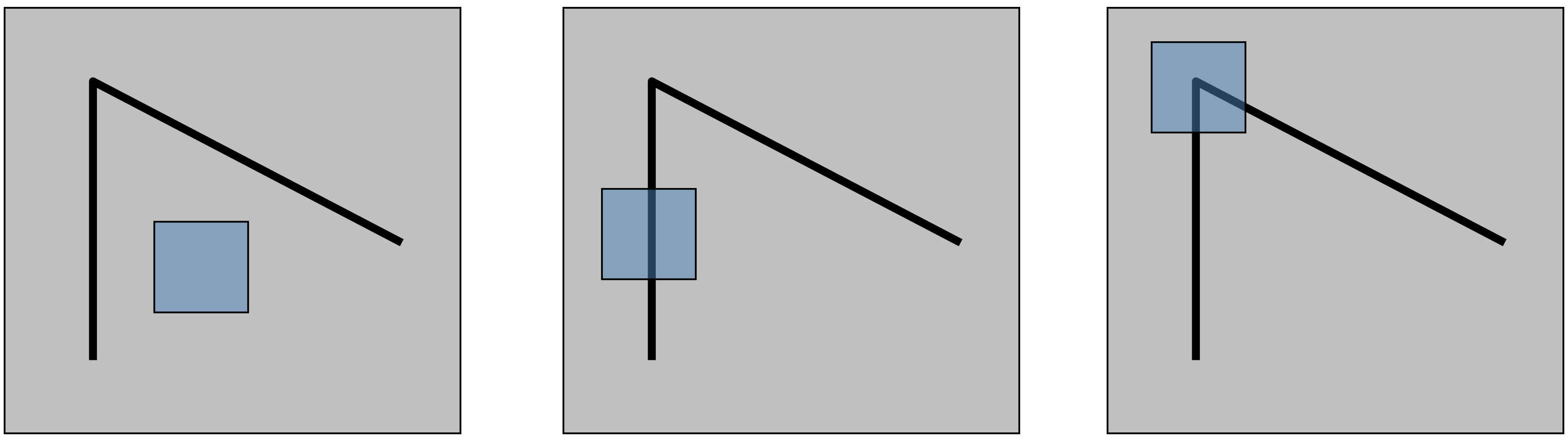
- Lead to unambiguous matches in other images

How to define “unusual”?

Local measures of uniqueness

Suppose we only consider a small window of pixels

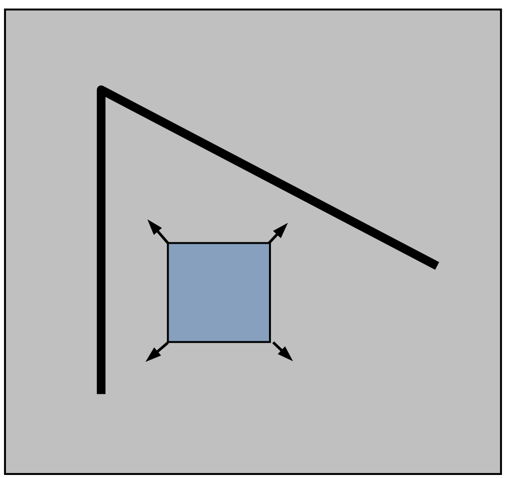
- What defines whether a feature is a good or bad candidate?



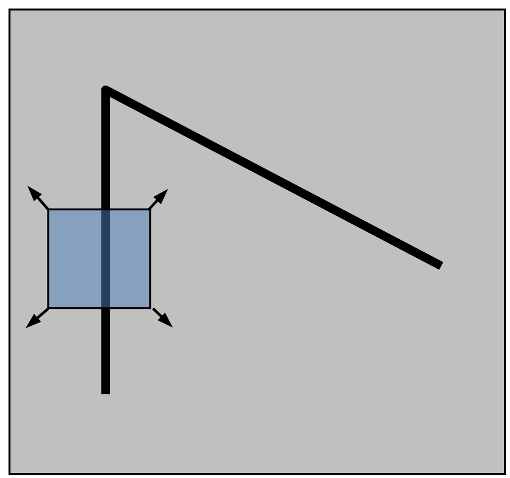
Credit: S. Seitz, D. Frolova, D. Simakov

Local measures of uniqueness

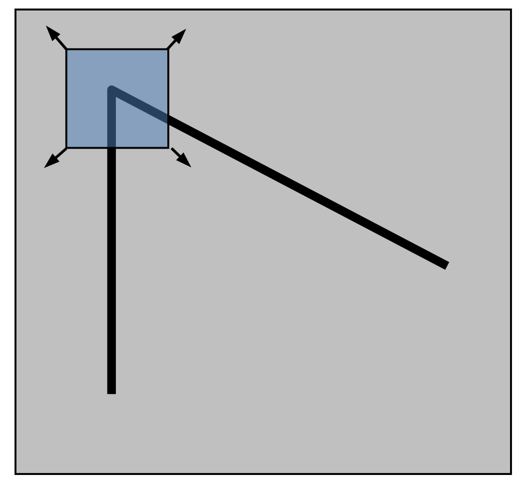
- How does the window change when you shift it?
- Shifting the window in any direction causes a big change



"flat" region:
no change in all directions



"edge":
no change along the edge direction



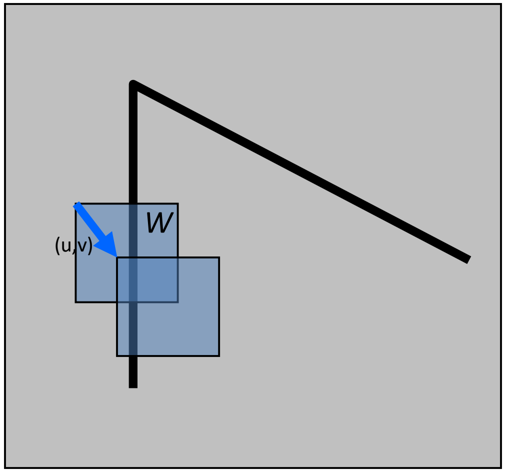
"corner":
significant change in all directions

Credit: S. Seitz, D. Frolova, D. Simakov

Harris 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 are happy if this error is high
- Slow to compute exactly for each pixel and each offset (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

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

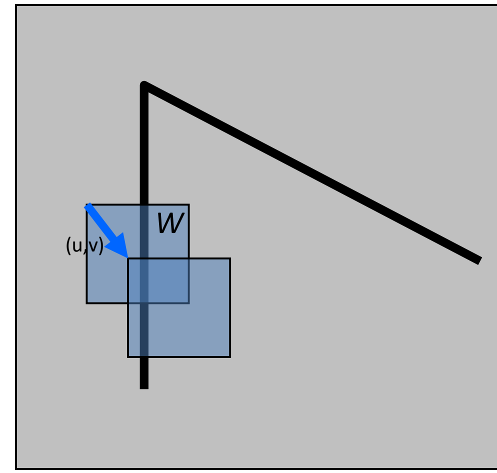
$$\text{shorthand: } I_x = \frac{\partial I}{\partial x}$$

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)*:

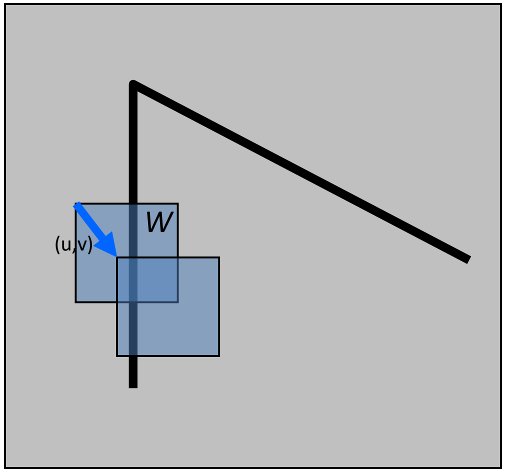


$$
\begin{aligned}
E(u,v) &= \sum_{(x,y)\in W} \left[I(x+u,y+v)-I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I(x,y)+I_x u+I_y v-I(x,y)\right]^2 \\
&\approx \sum_{(x,y)\in W} \left[I_x u+I_y v\right]^2
\end{aligned}
$$

Corner detection: the math

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

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



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

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

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

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

The second moment matrix

The surface $E(u,v)$ is locally approximated by a quadratic form.

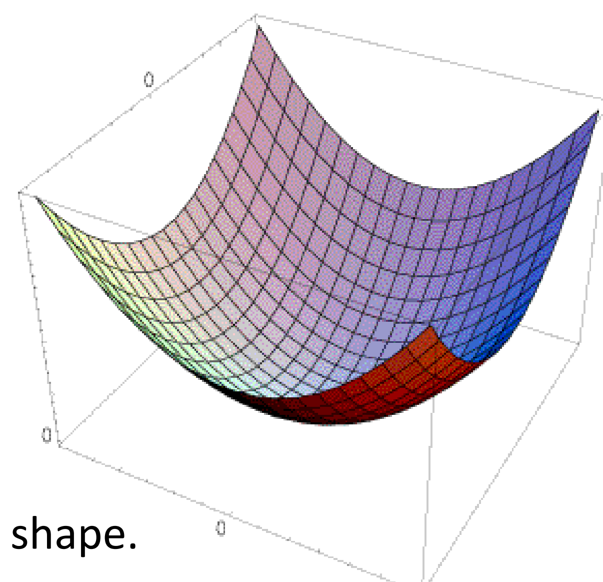
$$E(u,v) \approx Au^2 + 2Buv + Cv^2$$

$$\approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

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

$$B = \sum_{(x,y)\in W} I_x I_y$$

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



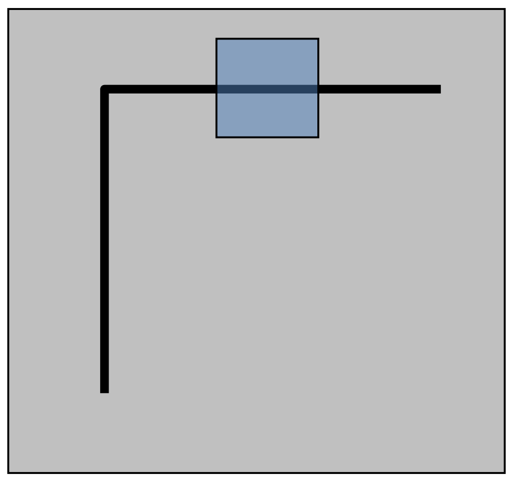
Let’s try to understand its shape.

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

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

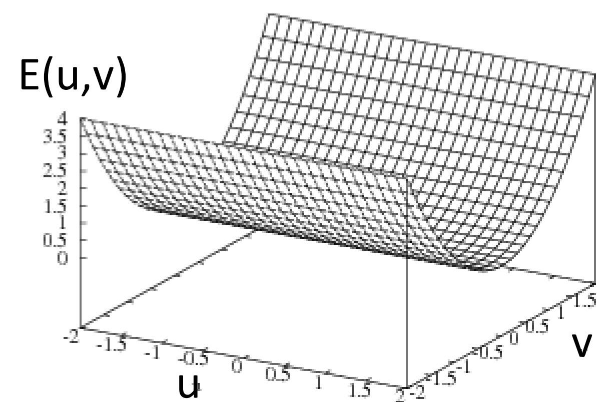
$$B = \sum_{(x,y)\in W} I_x I_y$$

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



Horizontal edge: $I_x = 0$

$$H = \begin{bmatrix} 0 & 0 \\ 0 & C \end{bmatrix}$$

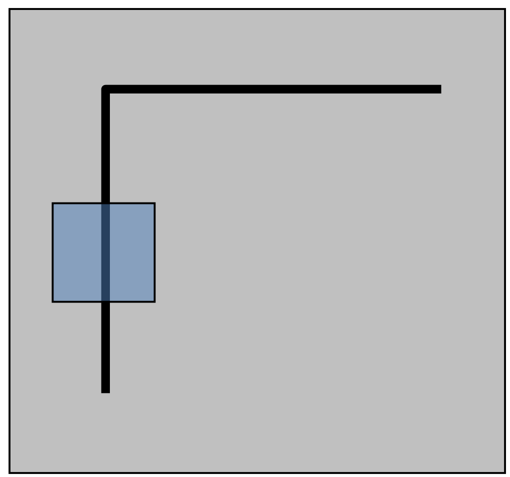


$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

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

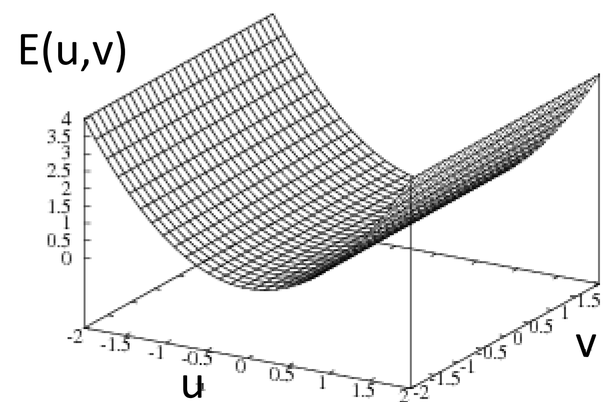
$$B = \sum_{(x,y)\in W} I_x I_y$$

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



Vertical edge: $I_y = 0$

$$H = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix}$$

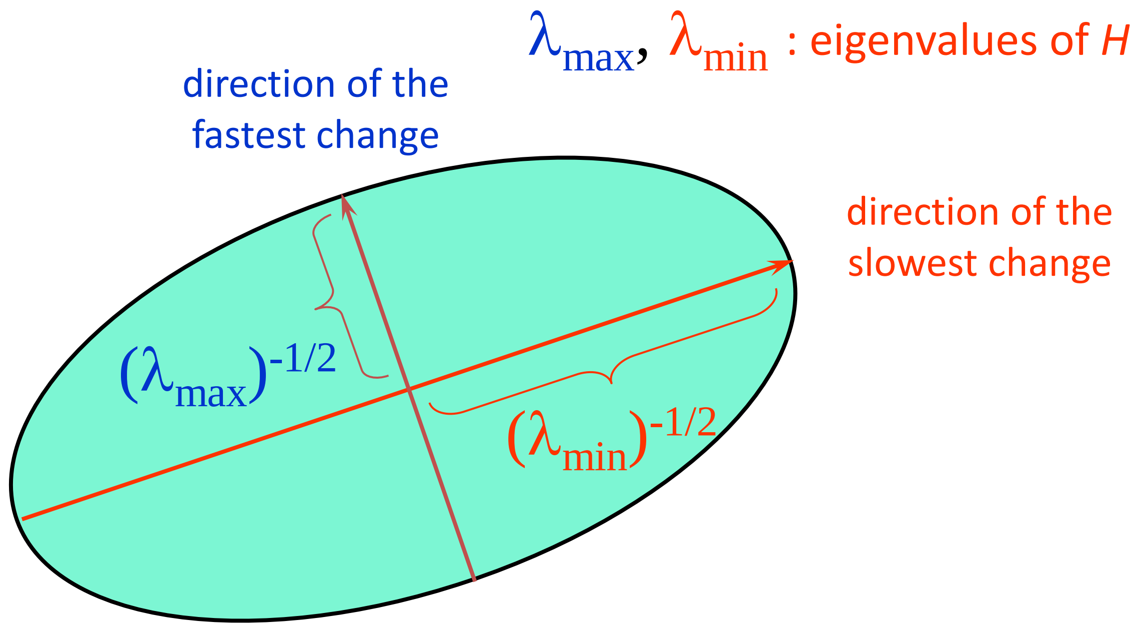


General case

We can visualize H as an ellipse with axis lengths determined by the *eigenvalues* of H and orientation determined by the *eigenvectors* of H

Ellipse equation:

$$[u \;\; v] \;\; H \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$



Quick eigenvalue/eigenvector review

The **eigenvectors** of a matrix **A** are the vectors **x** that satisfy:

$$Ax = \lambda x$$

The scalar λ is the **eigenvalue** corresponding to **x**

– The eigenvalues are found by solving:

$$det(A - \lambda I) = 0$$

– In our case, ***A*** = ***H*** is a 2x2 matrix, so we have

$$det \begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} = 0$$

– The solution:

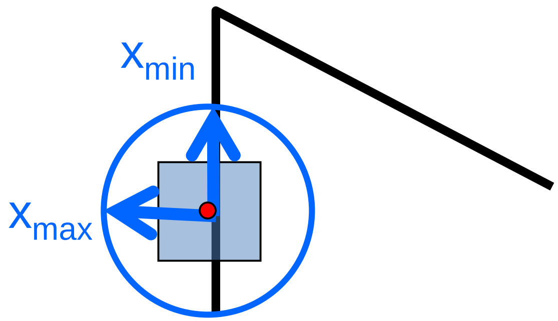
$$\lambda_{\pm} = \frac{1}{2}\left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2}\right]$$

Once you know λ, you find **x** by solving

$$\begin{bmatrix} h_{11} - \lambda & h_{12} \\ h_{21} & h_{22} - \lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 0$$

Corner detection: the math

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \underbrace{\begin{bmatrix} A & B \\ B & C \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$



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

$$Hx_{\min} = \lambda_{\min} x_{\min}$$

Eigenvalues and eigenvectors of H

- 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}

Corner detection: the math

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

- What's our feature scoring function?

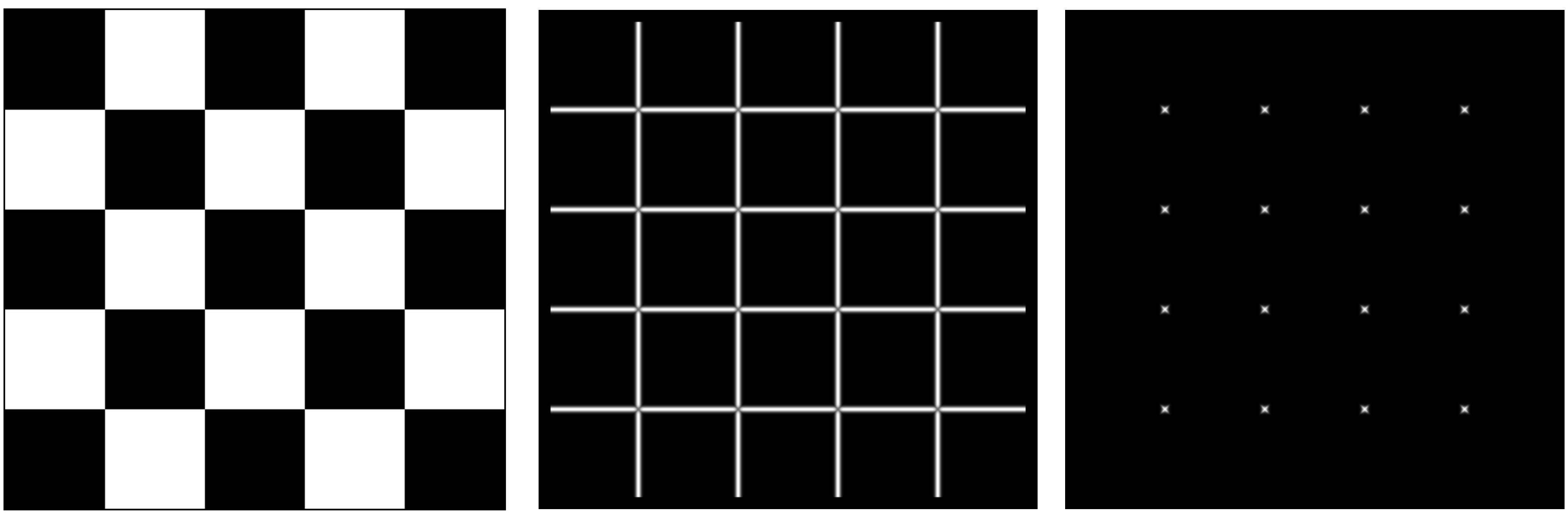
Corner detection: the math

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

- What's our feature scoring function?

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

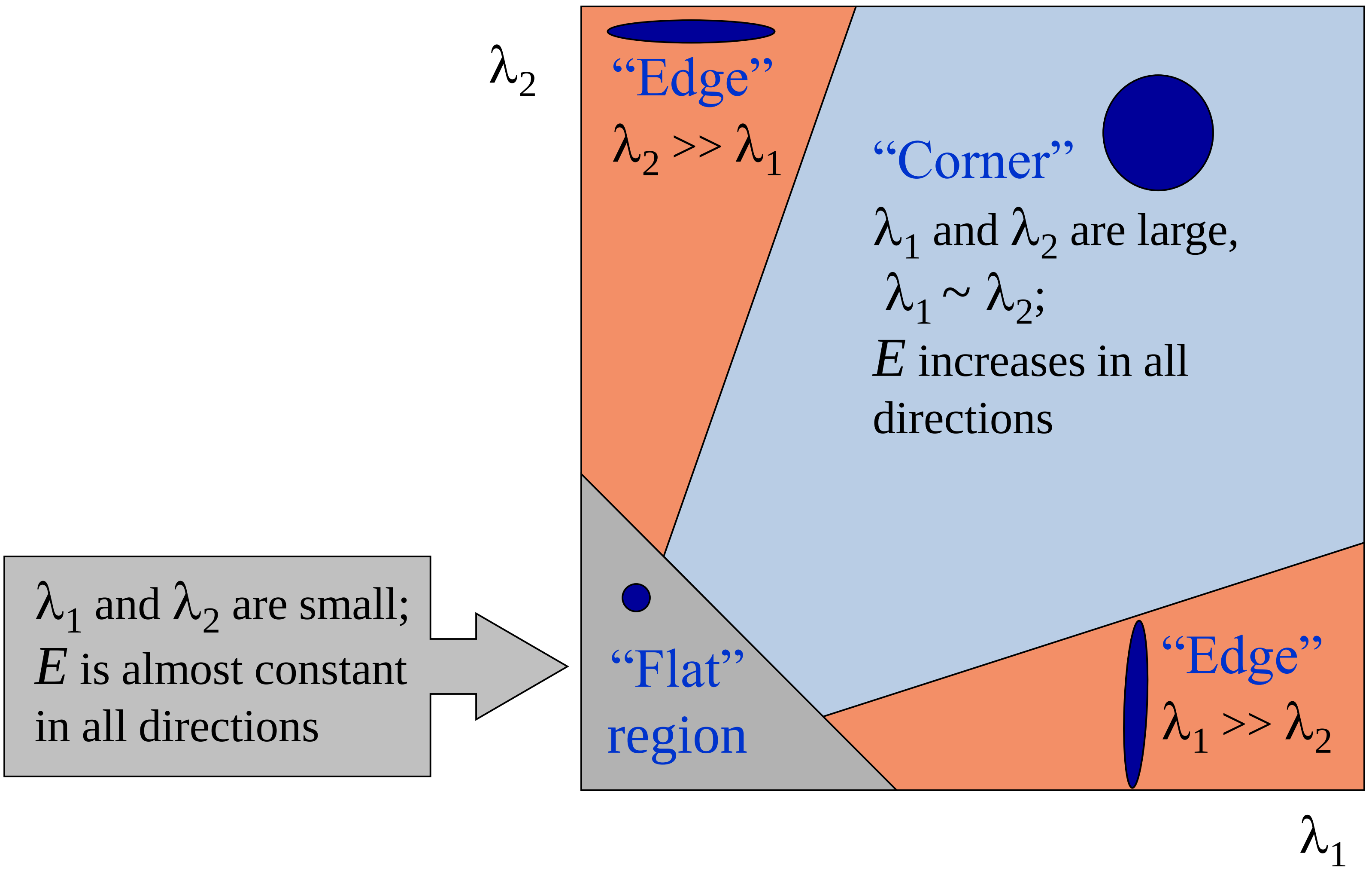
- the minimum of $E(u,v)$ should be large, over all unit vectors $[u\ v]$
- this minimum is given by the smaller eigenvalue (λ_{min}) of H



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

Interpreting the eigenvalues

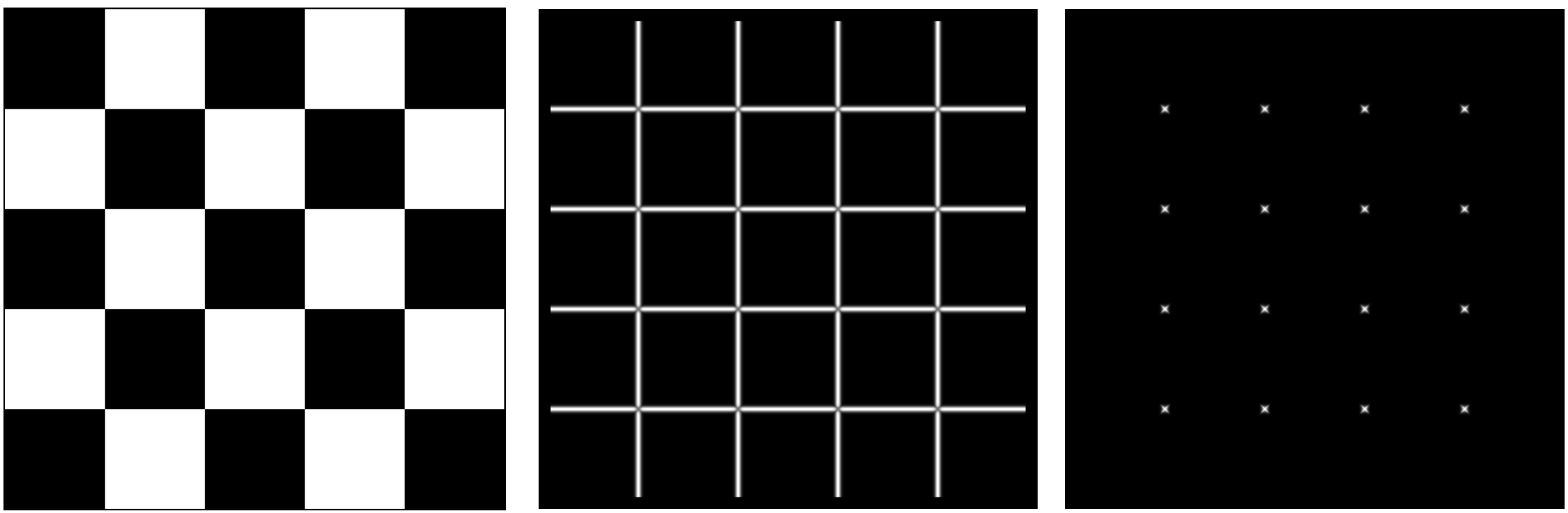
Classification of image points using eigenvalues of *M*:



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

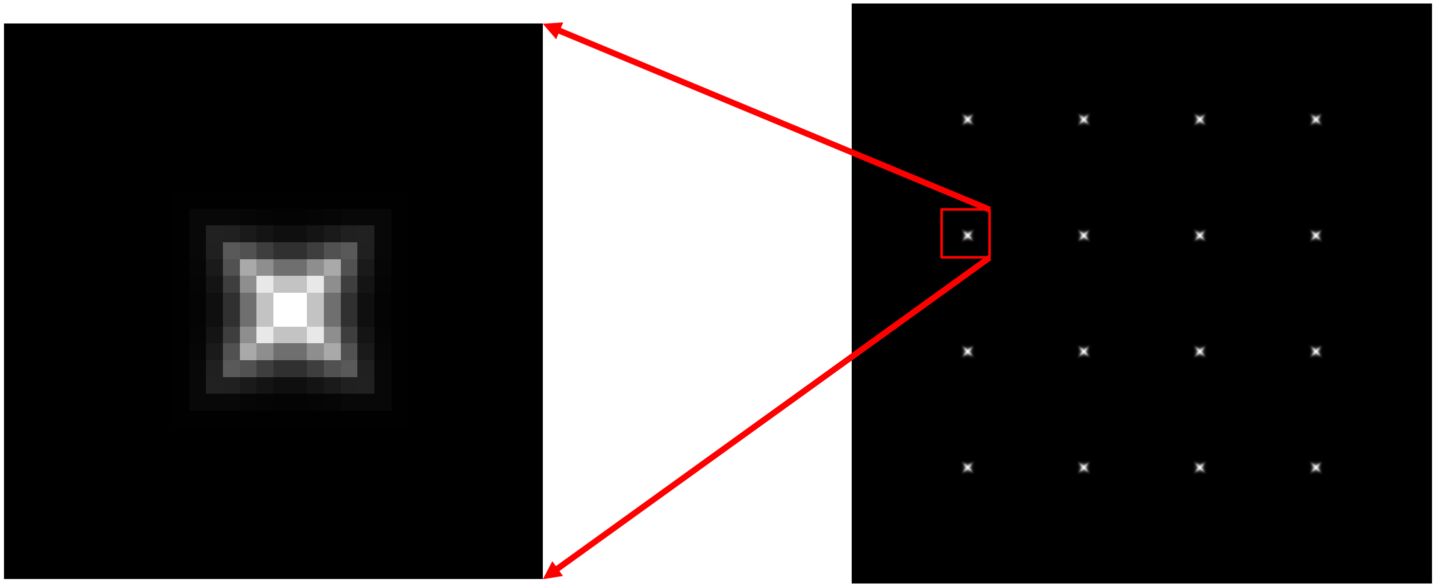


I λ_{max} λ_{min}

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

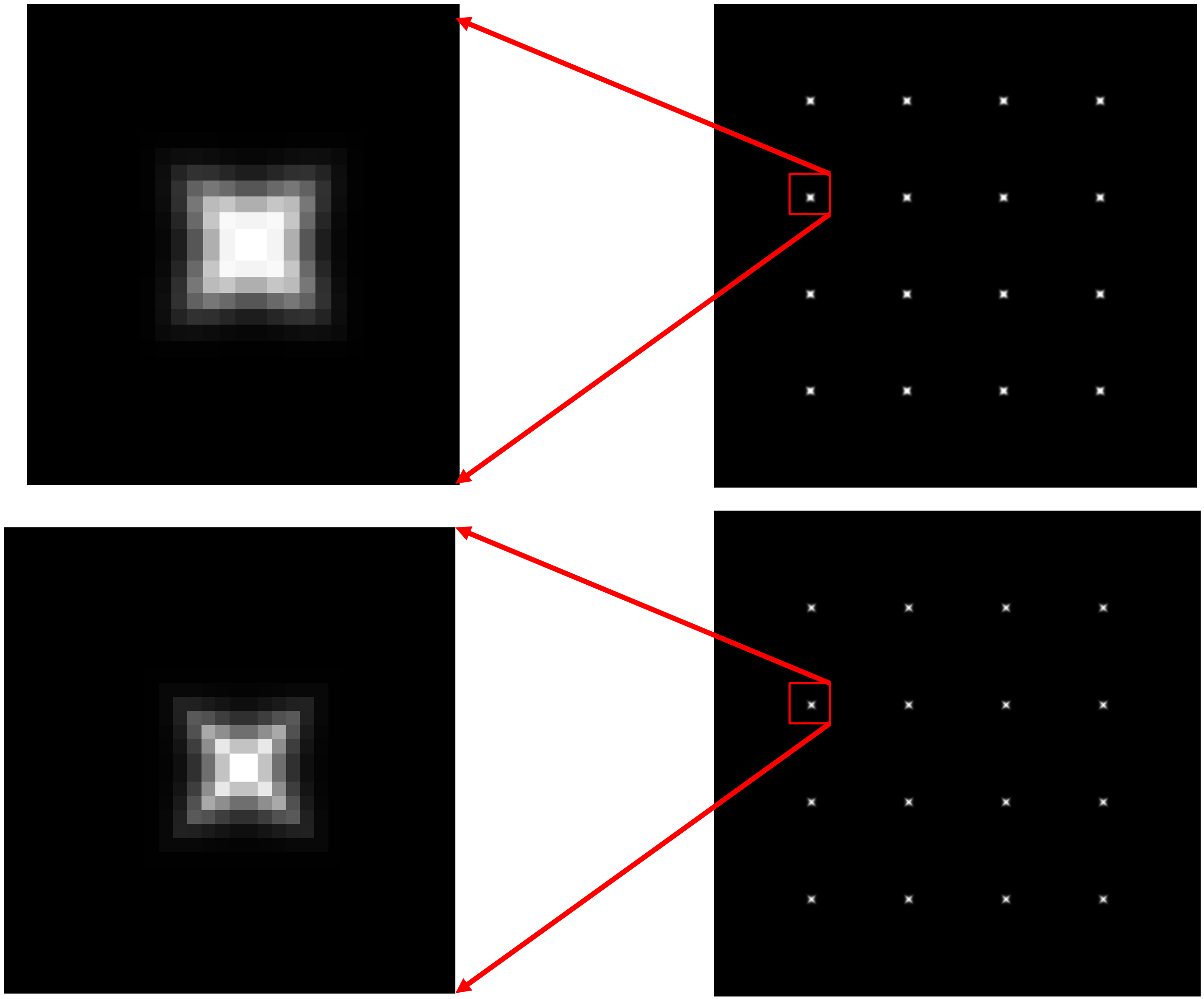
λ_{min} is a variant of the "Harris operator" for feature detection

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

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

- 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"
- Lots of other detectors, this is one of the most popular

The Harris operator

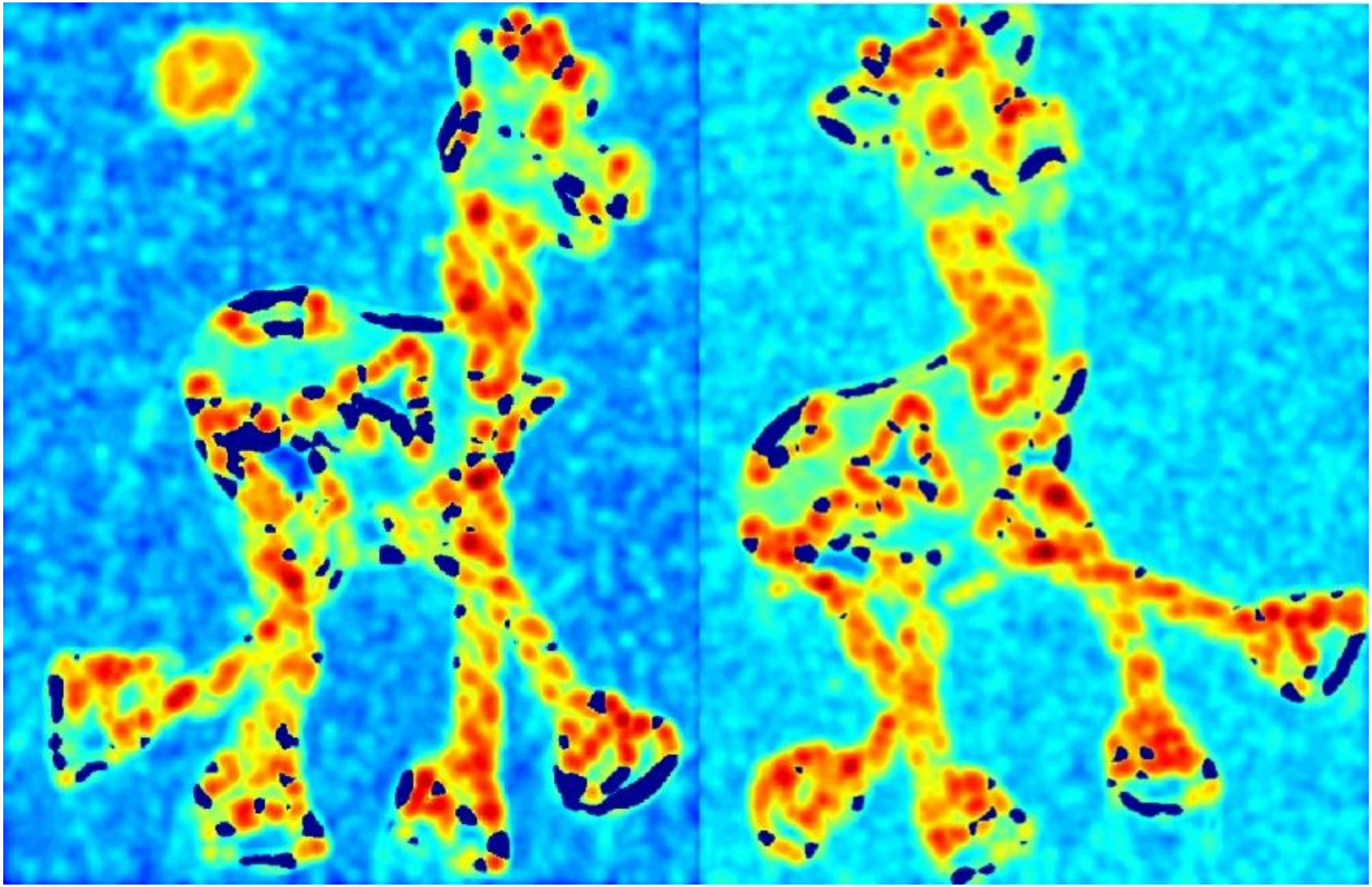


Harris operator

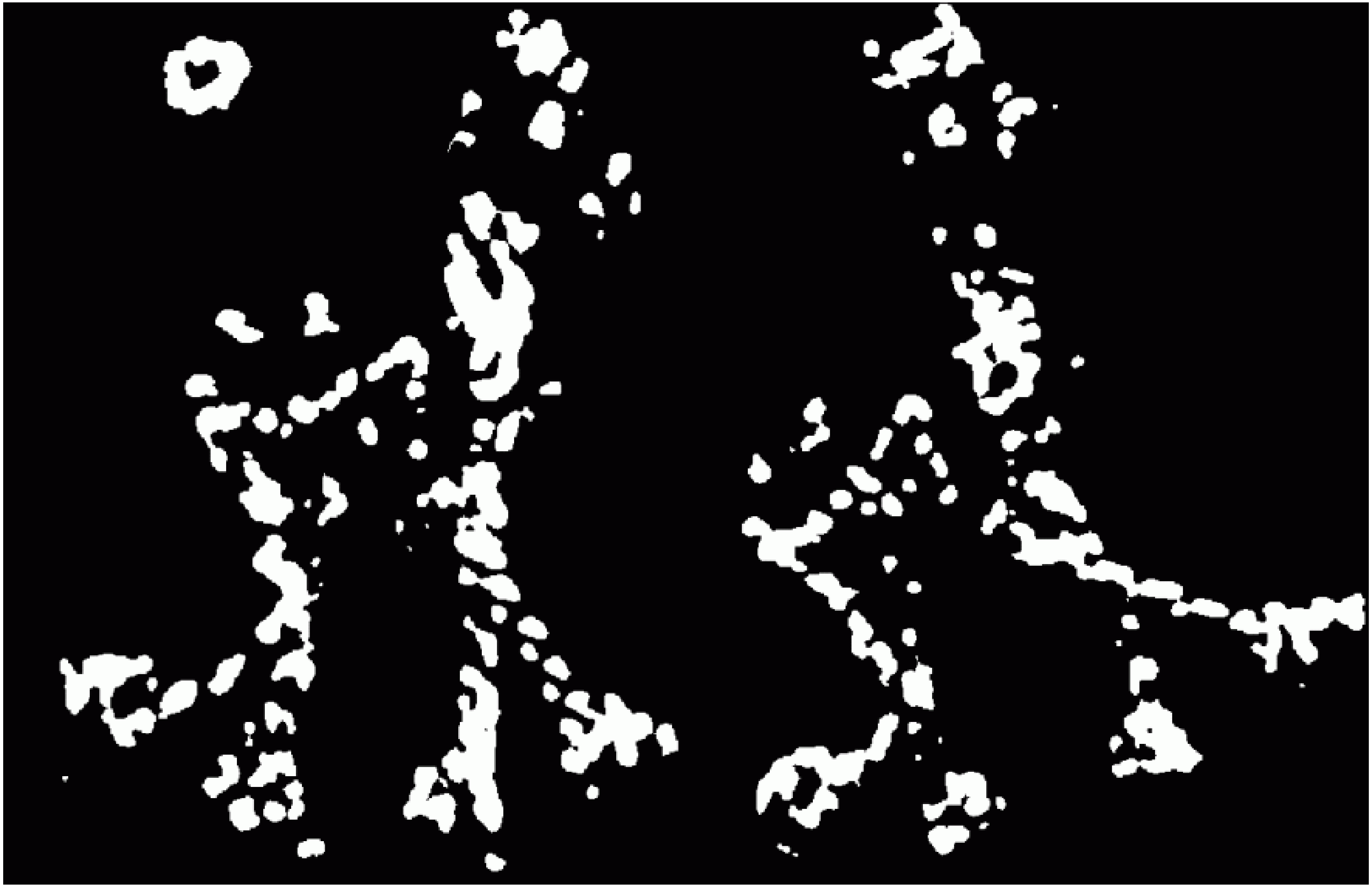
$\lambda_{\min}$

Harris detector example

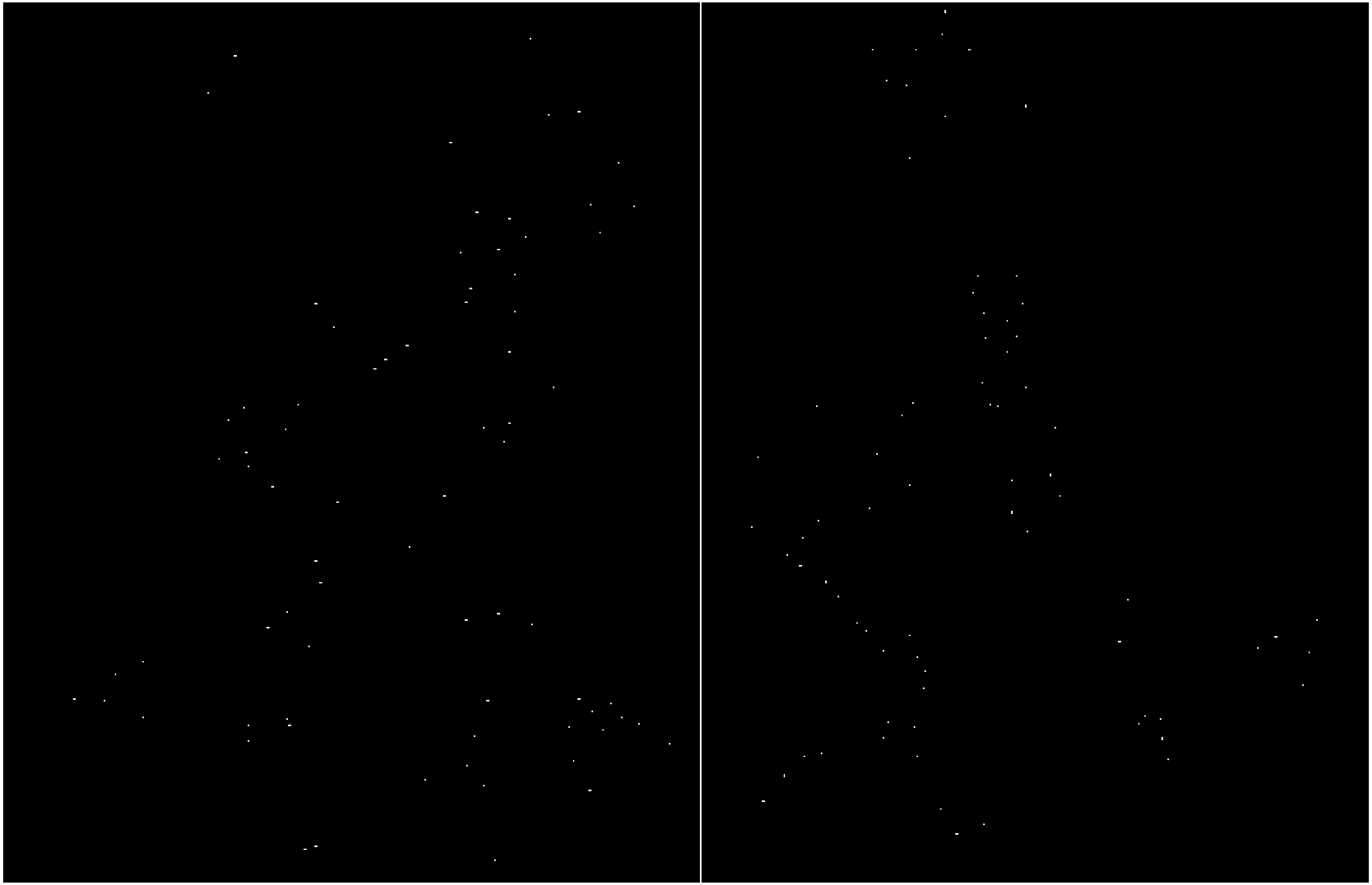
f value (red high, blue low)



Threshold (f > value)



Find local maxima of f



Harris features (in red)

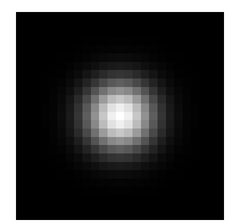
Weighting the derivatives

- 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 *weight* each derivative value based on its distance from the center pixel

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

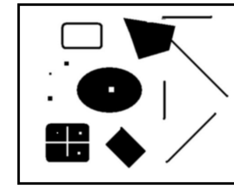


$w_{x,y}$

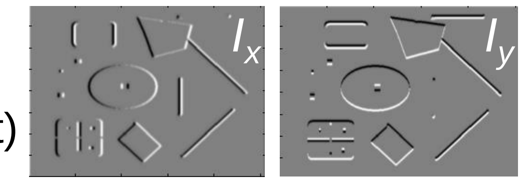
Harris Detector [Harris88]

- Second moment matrix

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

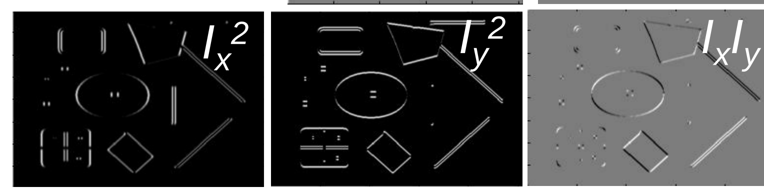


1. Image derivatives (optionally, blur first)



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

2. Square of derivatives



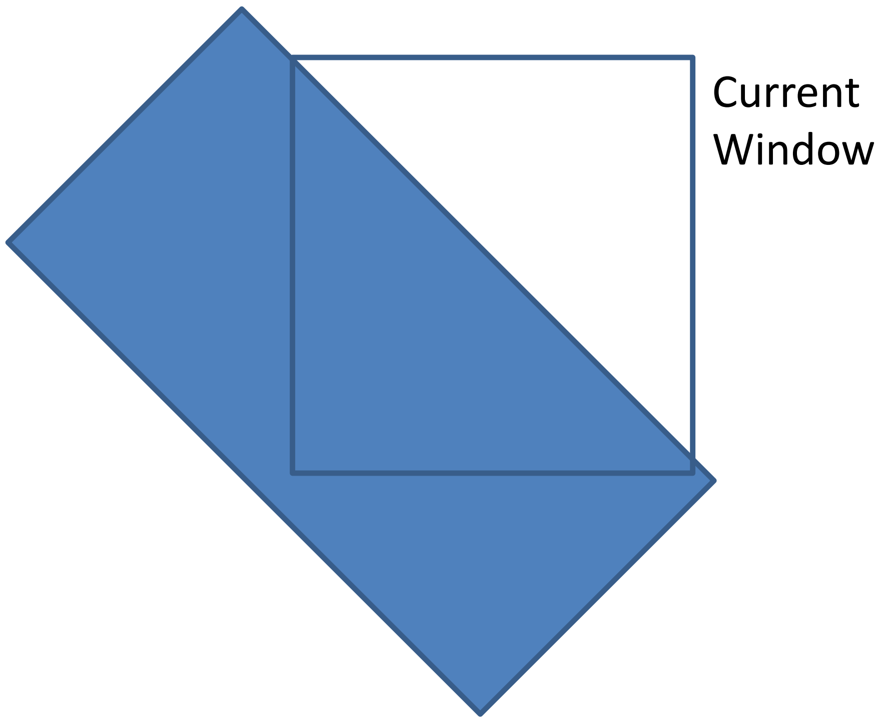
3. Gaussian filter $g(\sigma_I)$

4. Cornerness function – both eigenvalues are strong

5. Non-maxima suppression

Harris Corners – Why so complicated?

- Can’t we just check for regions with lots of gradients in the x and y directions?
  - No! A diagonal line would satisfy that criteria



Questions?