Edge and corner detection

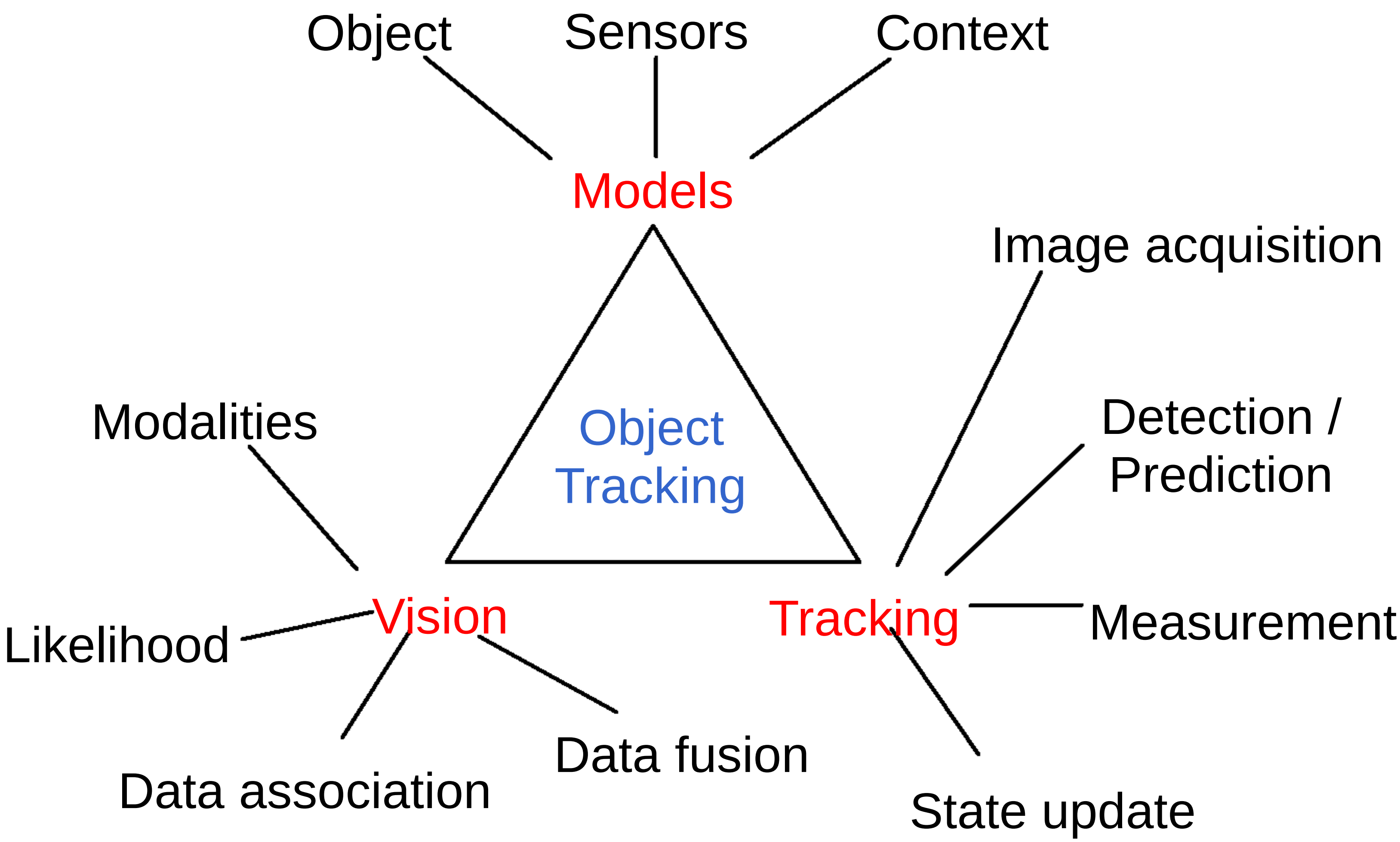
Prof. Stricker
Doz. Dr. G. Bleser

Computer Vision: Object and People Tracking

Example of the State of the Art (Video / Augmented Vision Group)

Artificial (computer) Vision: Tracking

Reminder: „tracking on one slide“



Goals

- Where is the information in an image?
- How is an object characterized?
- How can I find measurements in the image?

- The correct „Good Features“ are essential for tracking!

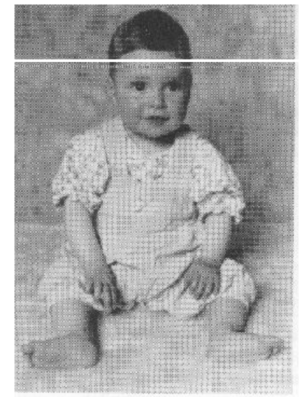
Outline

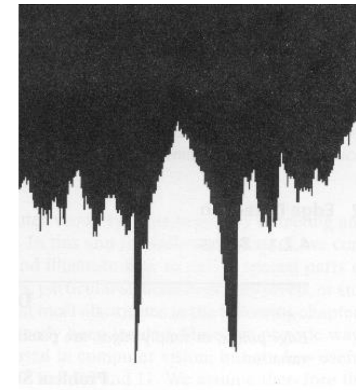
- Edge detection
- Canny edge detector
- Point extraction

Some slides from Lazebnik

Edge detection

- **Goal:** Identify sudden changes (discontinuities) in an image



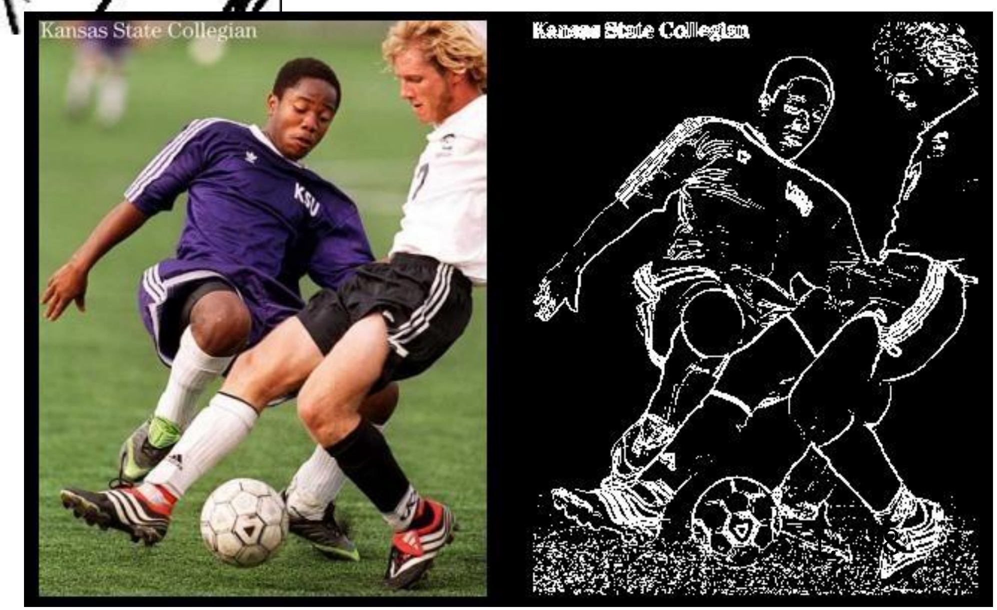


  - Intuitively, most semantic and shape information from the image can be encoded in the edges
  - More compact than pixels

- **Ideal:** artist's line drawing (but artist is also using object-level knowledge)

Source: D. Lowe and Debis

Edge Detection



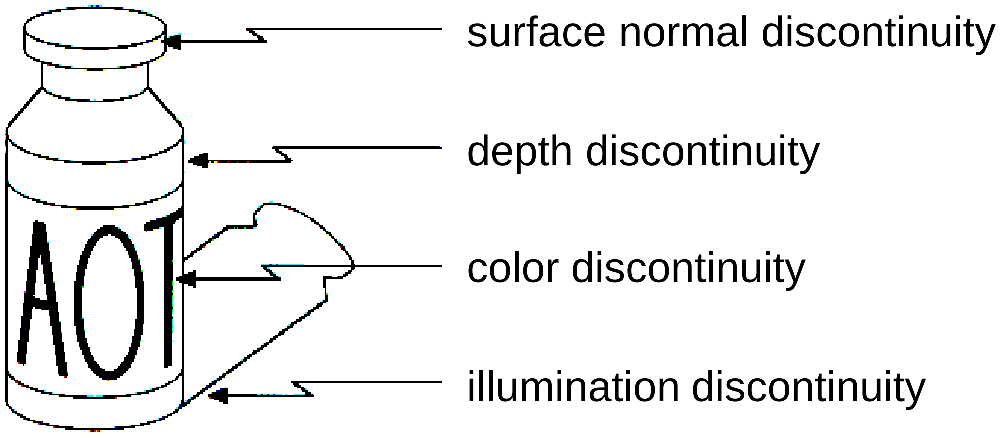
What Causes Intensity Changes?

Geometric events

- surface orientation (boundary) discontinuities
- depth discontinuities
- color and texture discontinuities

Non-geometric events

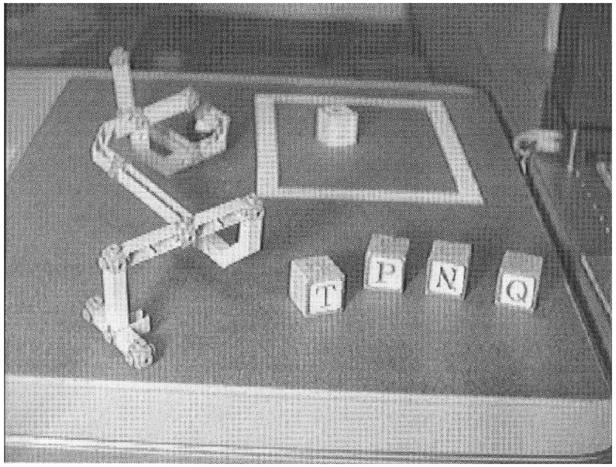
- illumination changes
- specularities
- shadows
- inter-reflections

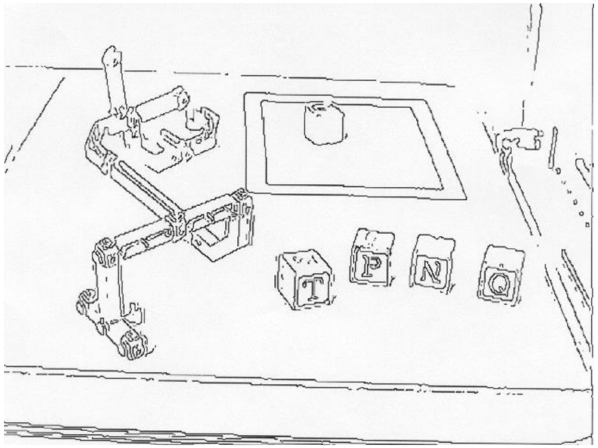


Why is Edge Detection Useful?

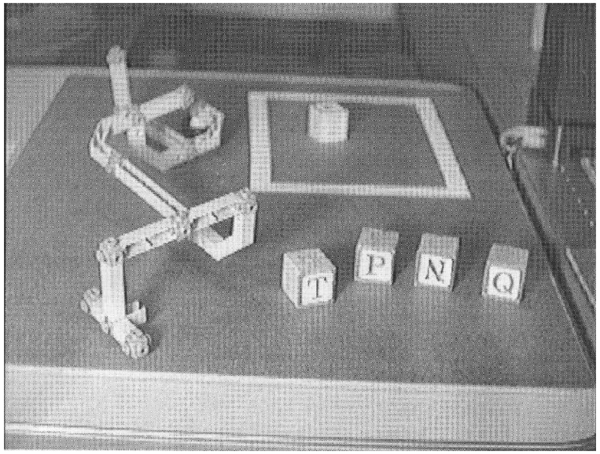
Important features can be extracted from the edges of an image (e.g., corners, lines, curves).

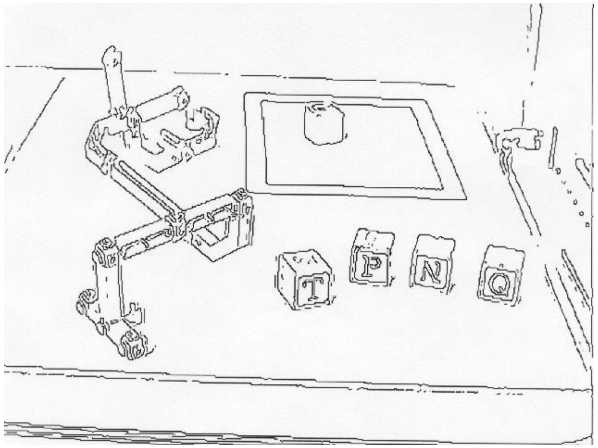
These features are used by higher-level computer vision algorithms (e.g., recognition).

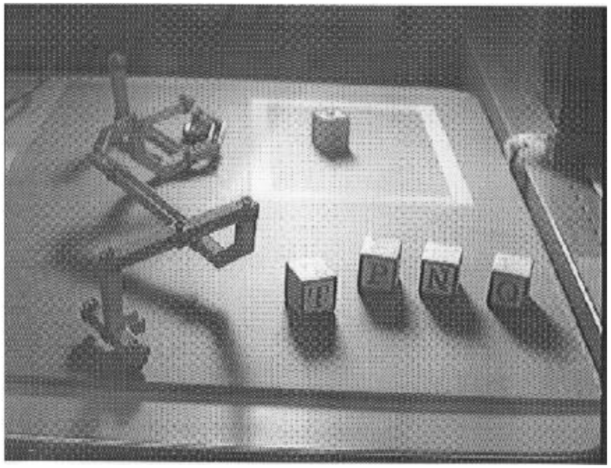


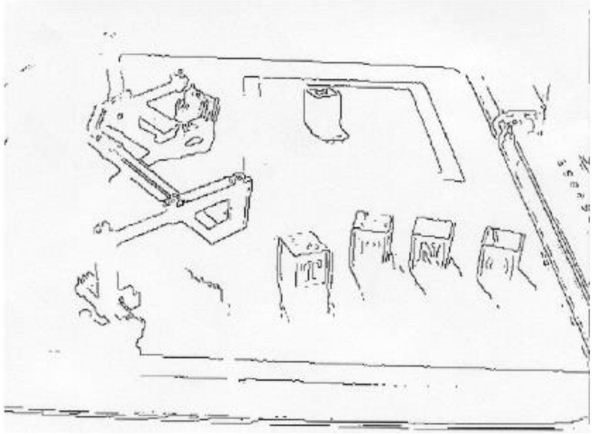


Effect of Illumination







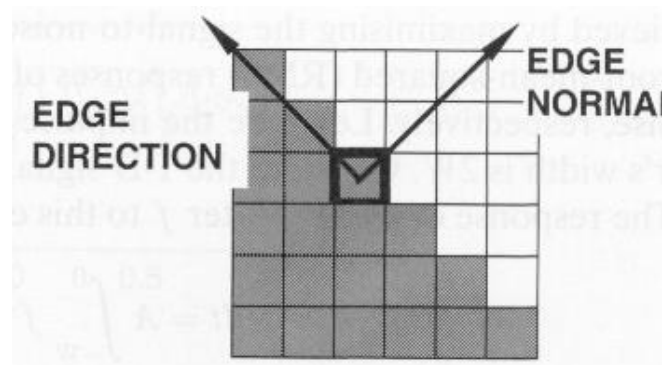


Edge Descriptors

Edge direction: perpendicular to the direction of maximum intensity change (i.e., edge normal)

Edge strength: related to the local image contrast along the normal.

Edge position: the image position at which the edge is located.



Characterizing edges

- An edge is a place of rapid change in the image intensity function

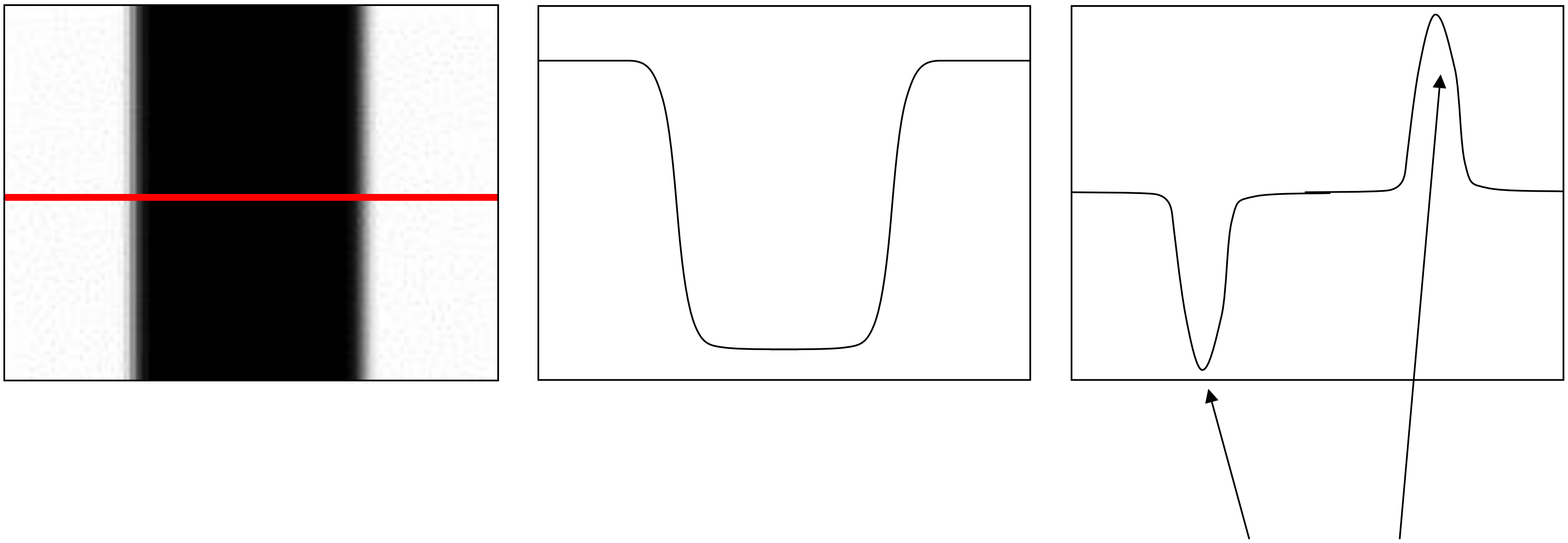
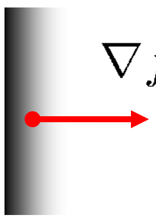


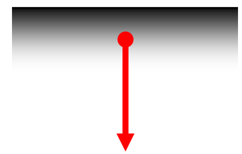
Image gradient

The gradient of an image:

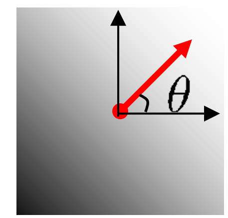
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of most rapid increase in intensity

The gradient direction is given by

- how does this relate to the direction of the edge?

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Source: Steve Seitz

Differentiation and convolution

Recall, for 2D function, f(x,y):

$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0}\left(\frac{f(x+\varepsilon, y)}{\varepsilon} - \frac{f(x,y)}{\varepsilon}\right)$$

We could approximate this as:

$$\frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$$

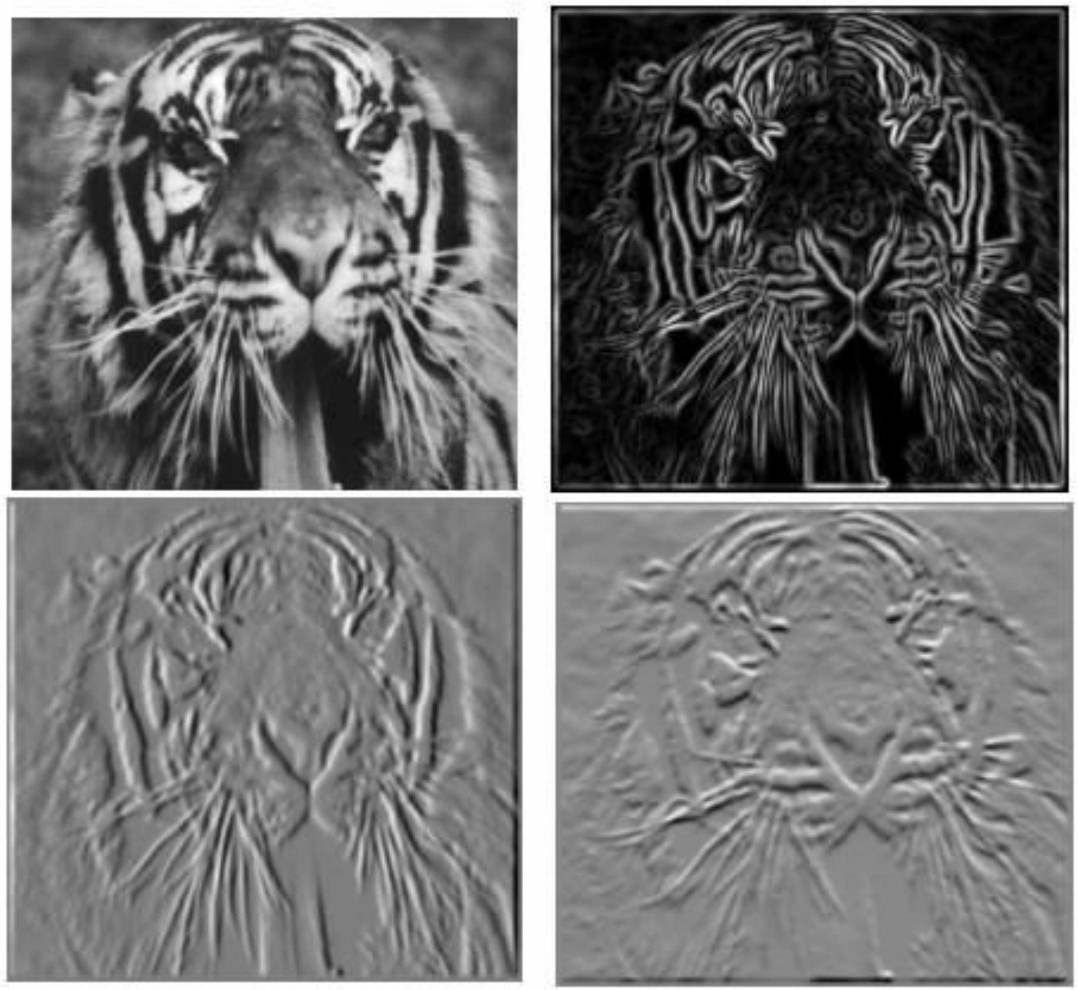
(which is obviously a convolution)

Check!

-1	**1**

Source: D. Forsyth, D. Lowe

Finite differences: example

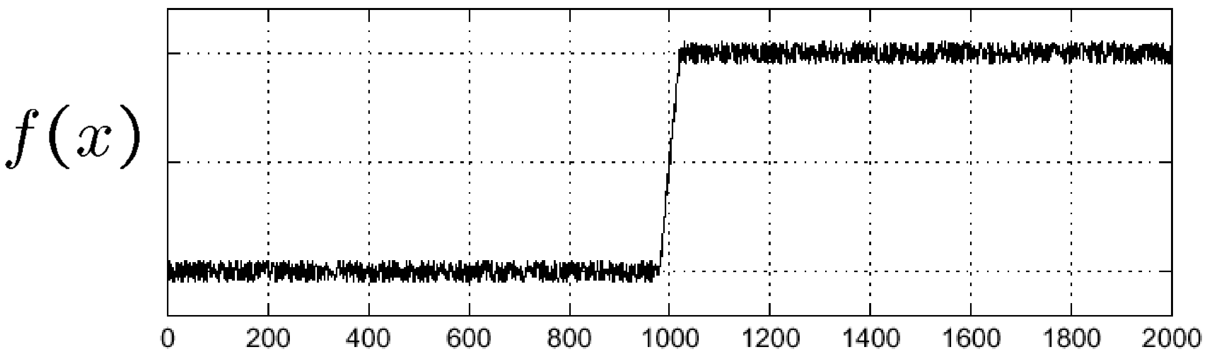


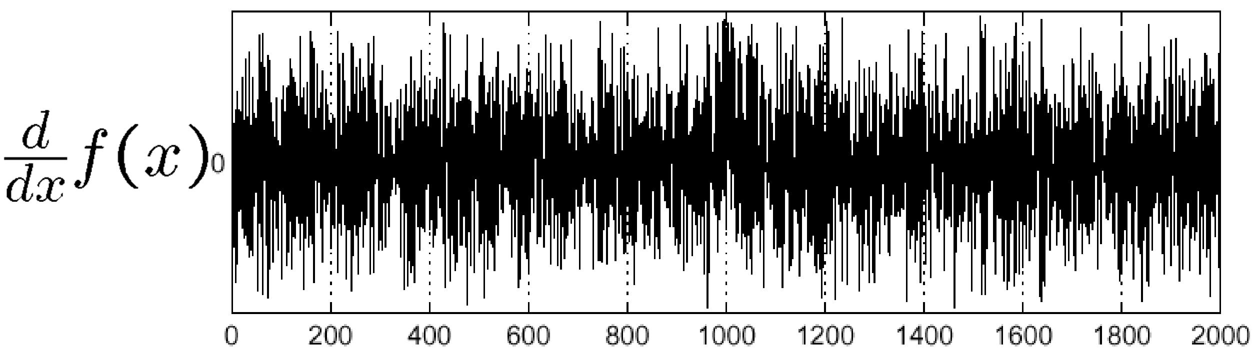
Which one is the gradient in the x-direction (resp. y-direction)?

Effects of noise

Consider a single row or column of the image

- Plotting intensity as a function of position gives a signal





Where is the edge?

Source: S. Seitz

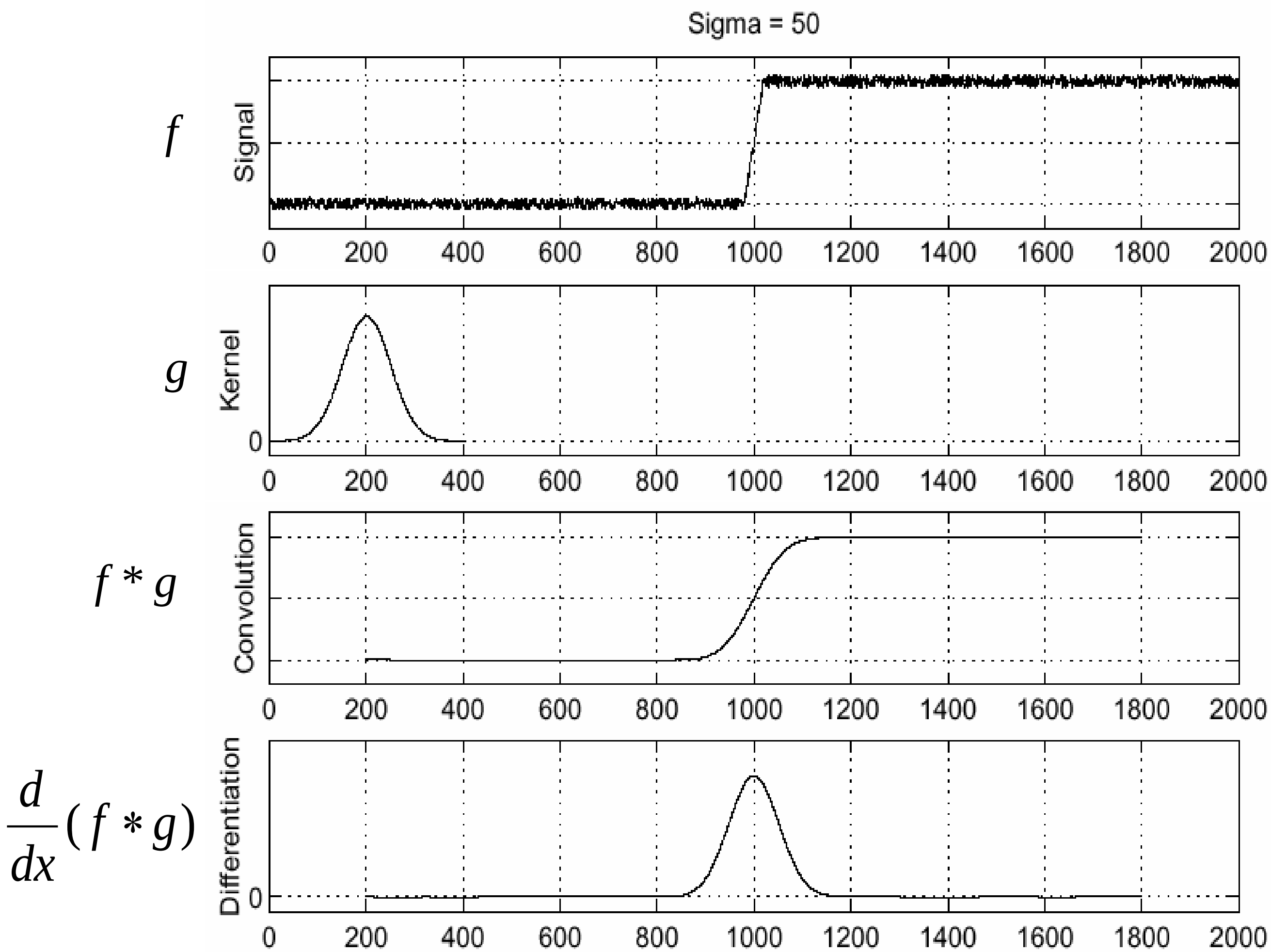
Effects of noise

- Finite difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the stronger the response

- What is to be done?
  - Smoothing the image should help, by forcing pixels different to their neighbors (=noise pixels?) to look more like neighbors

Source: D. Forsyth

Solution: smooth first



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

Source: S. Seitz

Derivative theorem of convolution

- Differentiation and convolution both linear operators: they "commute"

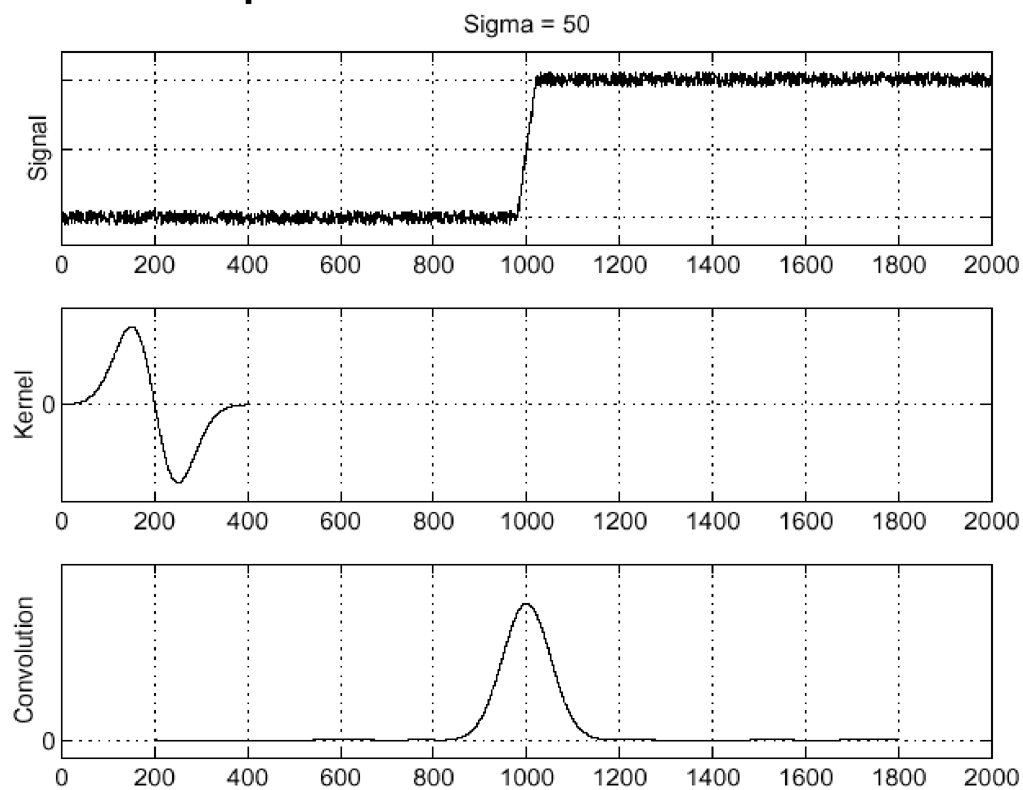
$$\frac{d}{dx}\left(f * g\right) = \frac{df}{dx} * g = f * \frac{dg}{dx}$$

- This saves us one operation:

f

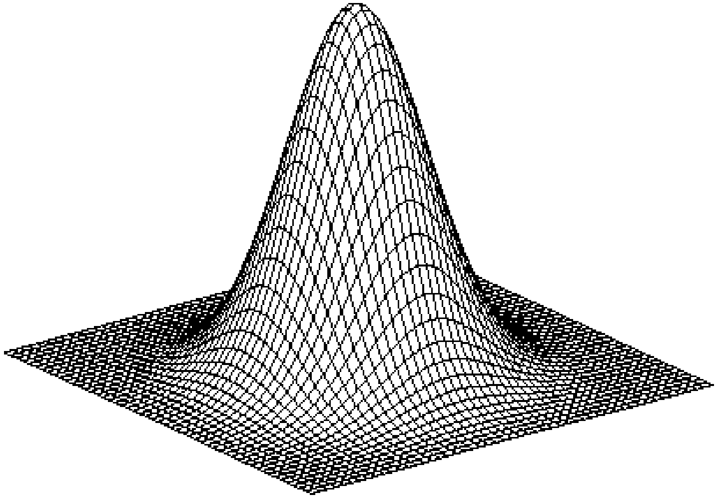
$\frac{d}{dx} g$

$f * \frac{d}{dx} g$

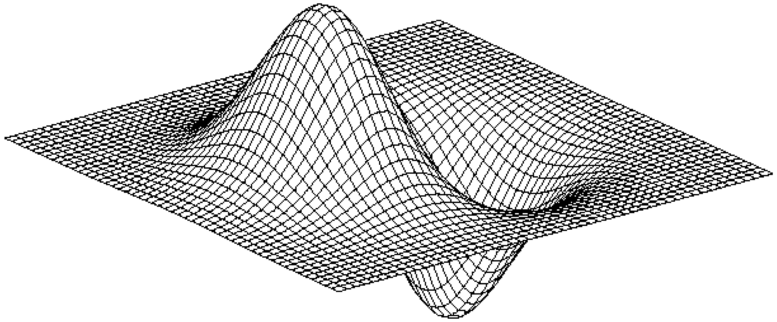


Source: S. Seitz

Derivative of Gaussian filter

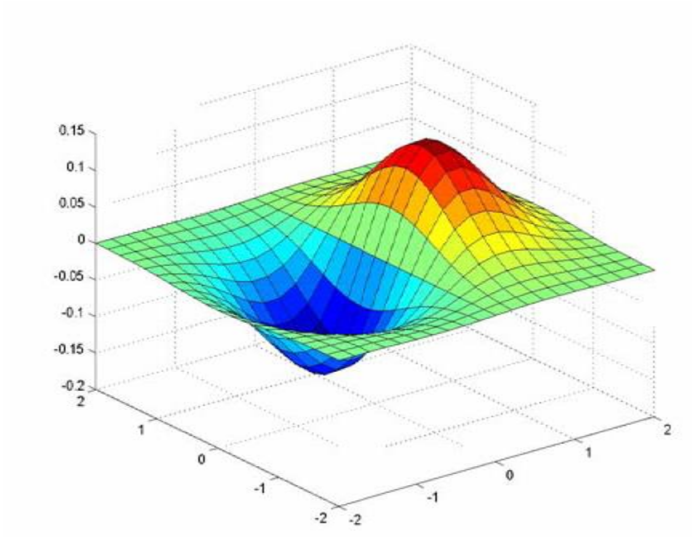


* [1 -1] =

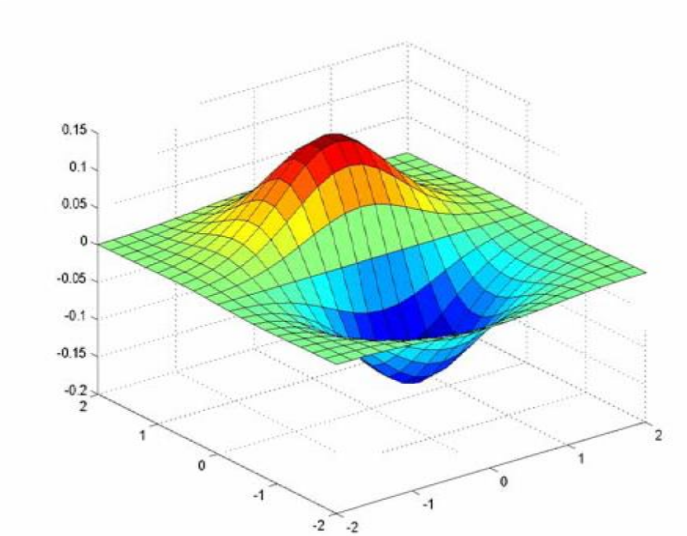


This filter is separable

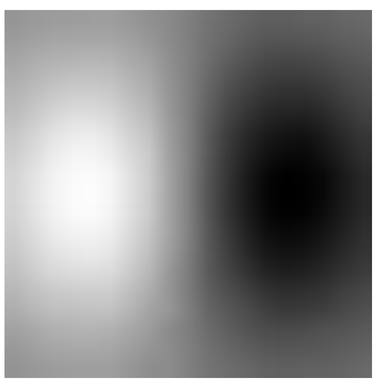
Derivative of Gaussian filter

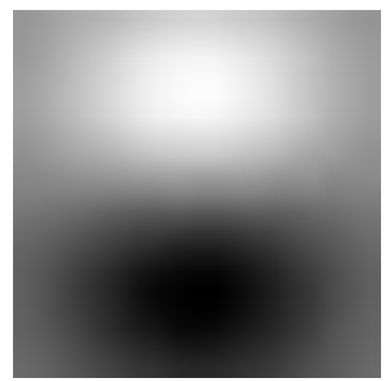


x-direction

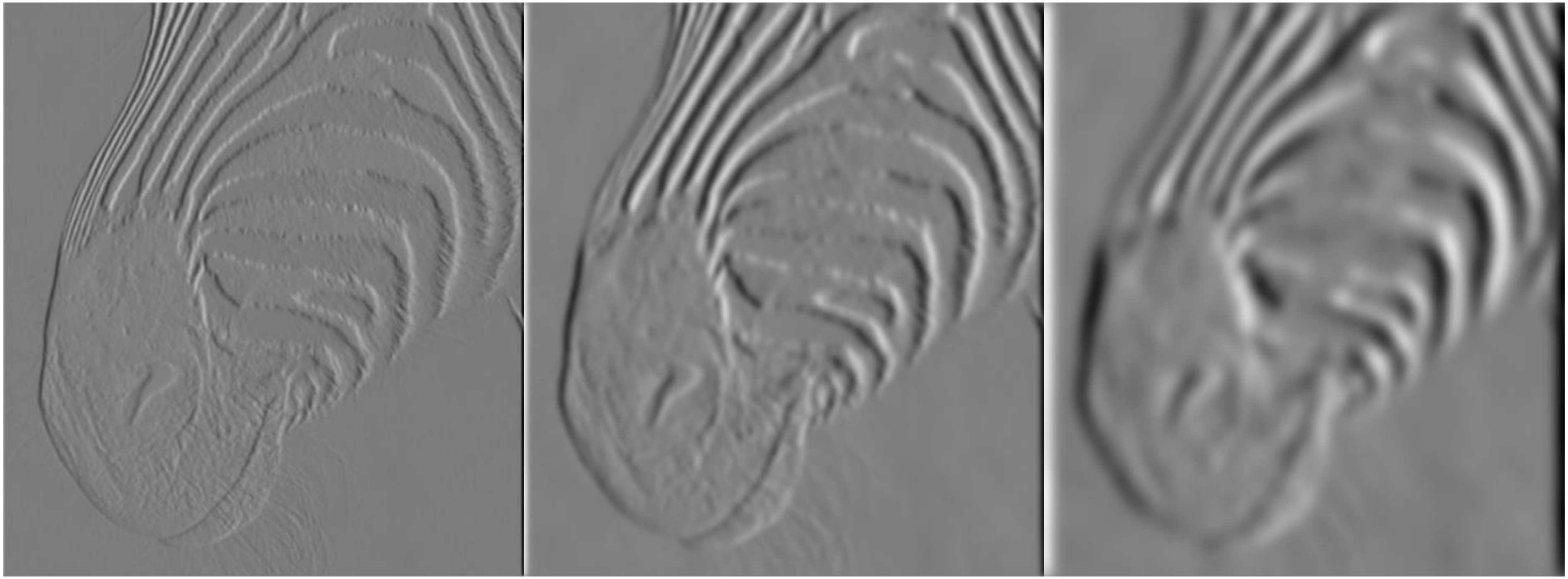


y-direction





Tradeoff between smoothing and localization



1 pixel

3 pixels

7 pixels

Smoothed derivative removes noise, but blurs edge.
Also finds edges at different "scales".

Source: D. Forsyth

Finite difference filters

- Other approximations of derivative filters exist:

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Source: K. Grauman

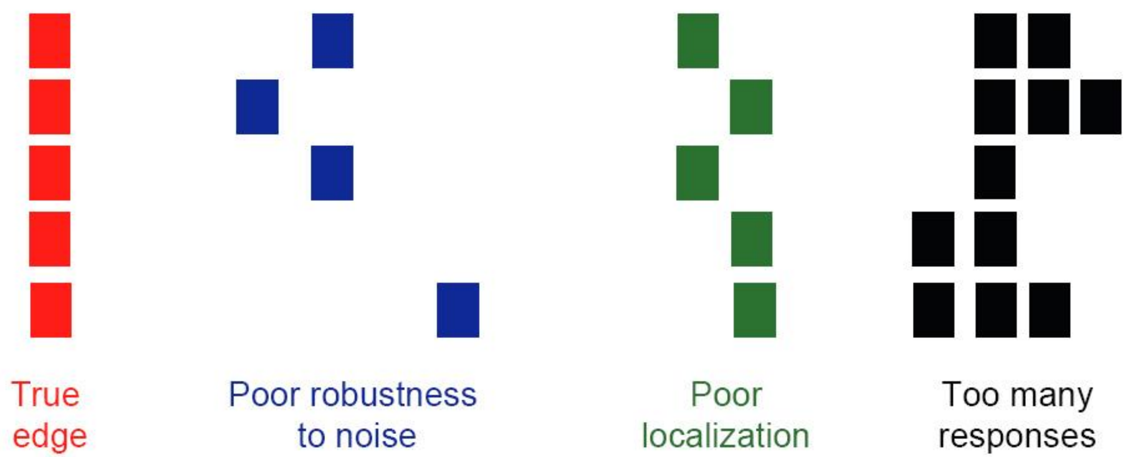
Implementation issues

- The gradient magnitude is large along a thick "trail" or "ridge", so how do we identify the actual edge points?
- How do we link the edge points to form curves?

Source: D. Forsyth

Designing an edge detector

- Criteria for an "optimal" edge detector:
  - **Good detection:** the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges)
  - **Good localization:** the edges detected must be as close as possible to the true edges
  - **Single response:** the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge



Source: L. Fei-Fei

Canny edge detector

- This is probably the most widely used edge detector in computer vision
- Theoretical model: step-edges corrupted by additive Gaussian noise
- Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of *signal-to-noise ratio* and localization

J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Source: L. Fei-Fei

Canny edge detector

1. Filter image with derivative of Gaussian

2. Find magnitude and orientation of gradient

3. Non-maximum suppression:
   - Thin multi-pixel wide “ridges” down to single pixel width

4. Linking and thresholding (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

MATLAB: edge(image, ‘canny’)

Source: D. Lowe, L. Fei-Fei

Example

original image (Lena)

Example

norm of the gradient

Example

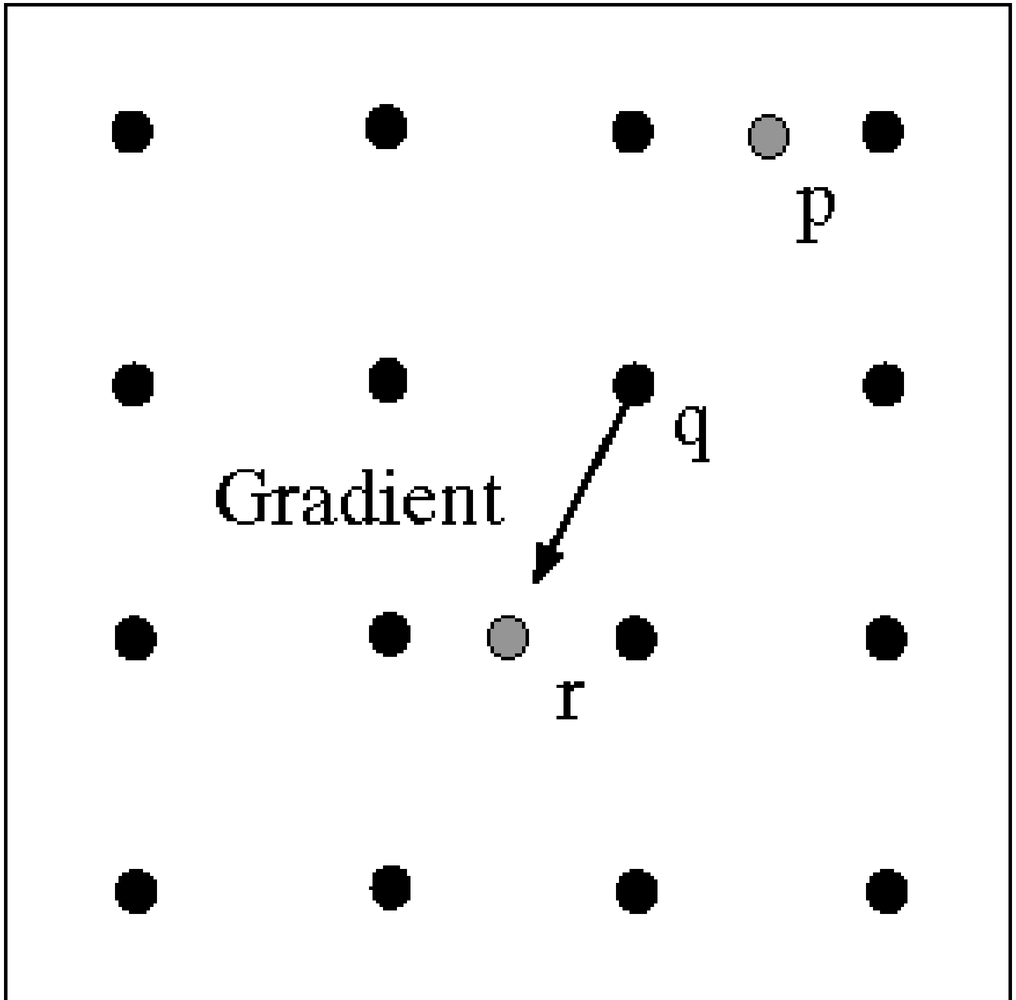
thresholding

Example

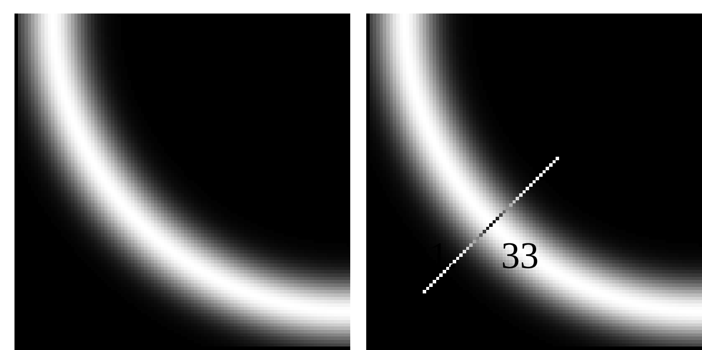
thinning

(non-maximum suppression)

Non-maximum suppression

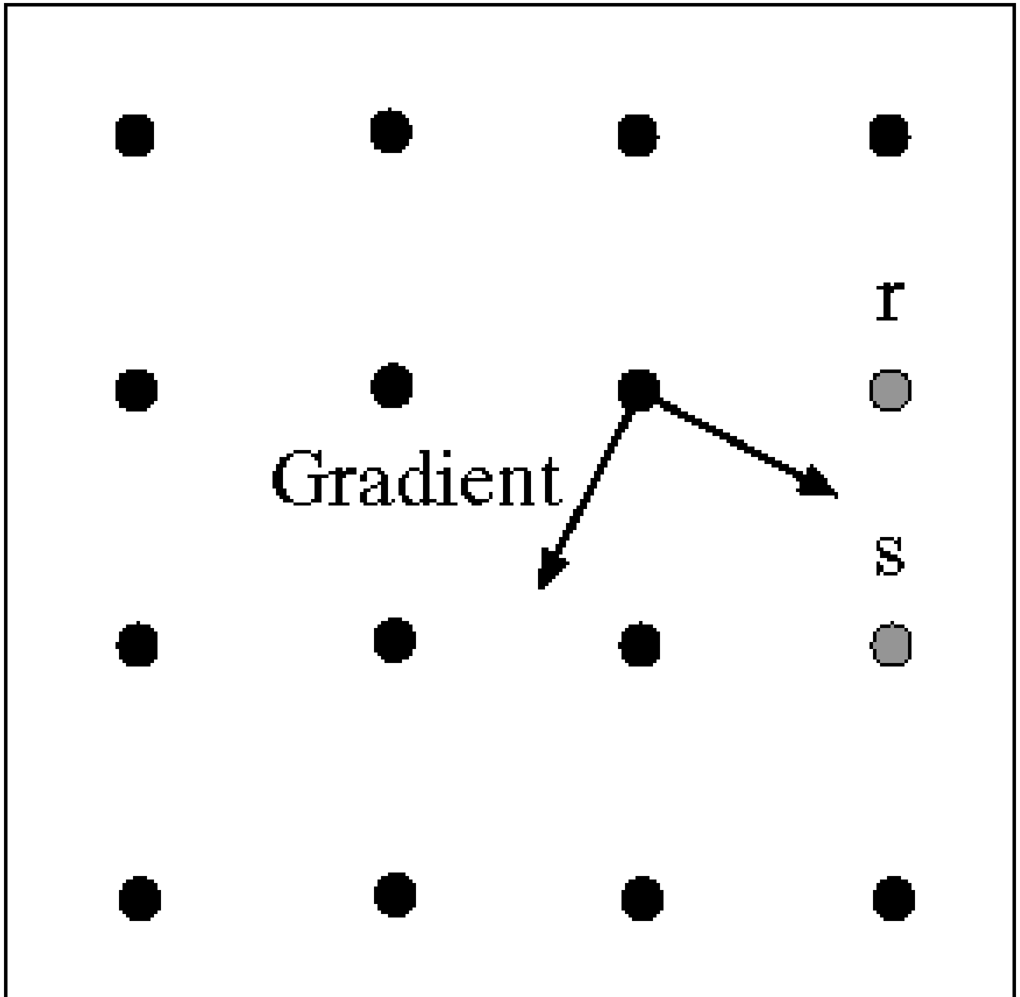


At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.

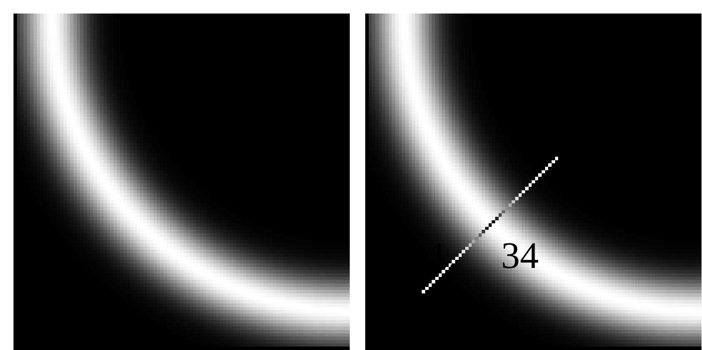


Source: D. Forsyth

Edge linking



Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).

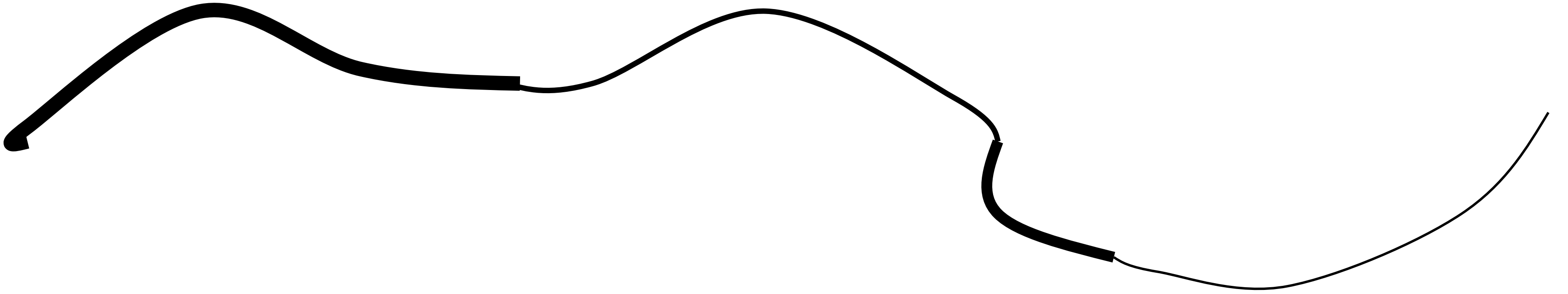


Source: D. Forsyth

Hysteresis thresholding

Check that maximum value of gradient value is sufficiently large

- drop-outs? use **hysteresis**
  - use a high threshold to start edge curves and a low threshold to continue them.



Source: S. Seitz

Hysteresis thresholding

original image

high threshold
(strong edges)

low threshold
(weak edges)

hysteresis threshold

Source: L. Fei-Fei

Effect of σ (Gaussian kernel spread/size)

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

Source: S. Seitz

Edge detection is just the beginning…

Berkeley segmentation database:

http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

image | human segmentation | gradient magnitude

Features

Image Matching

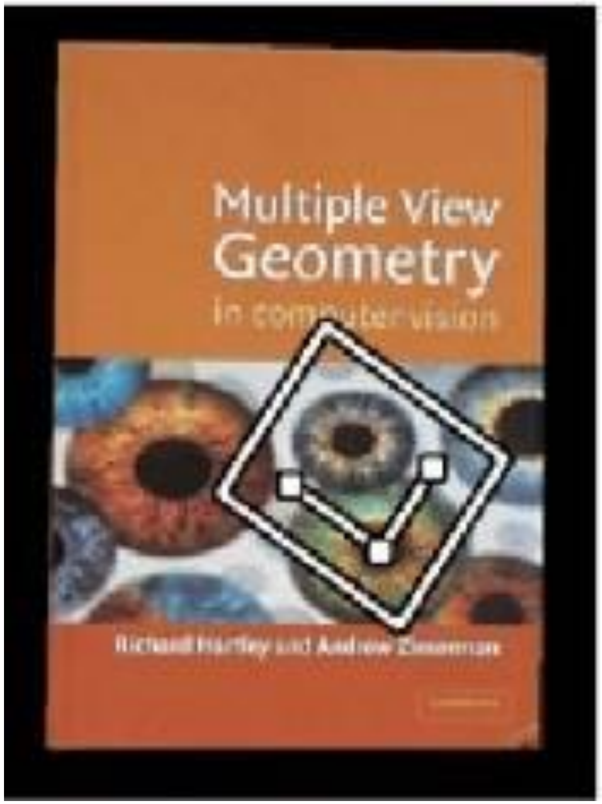
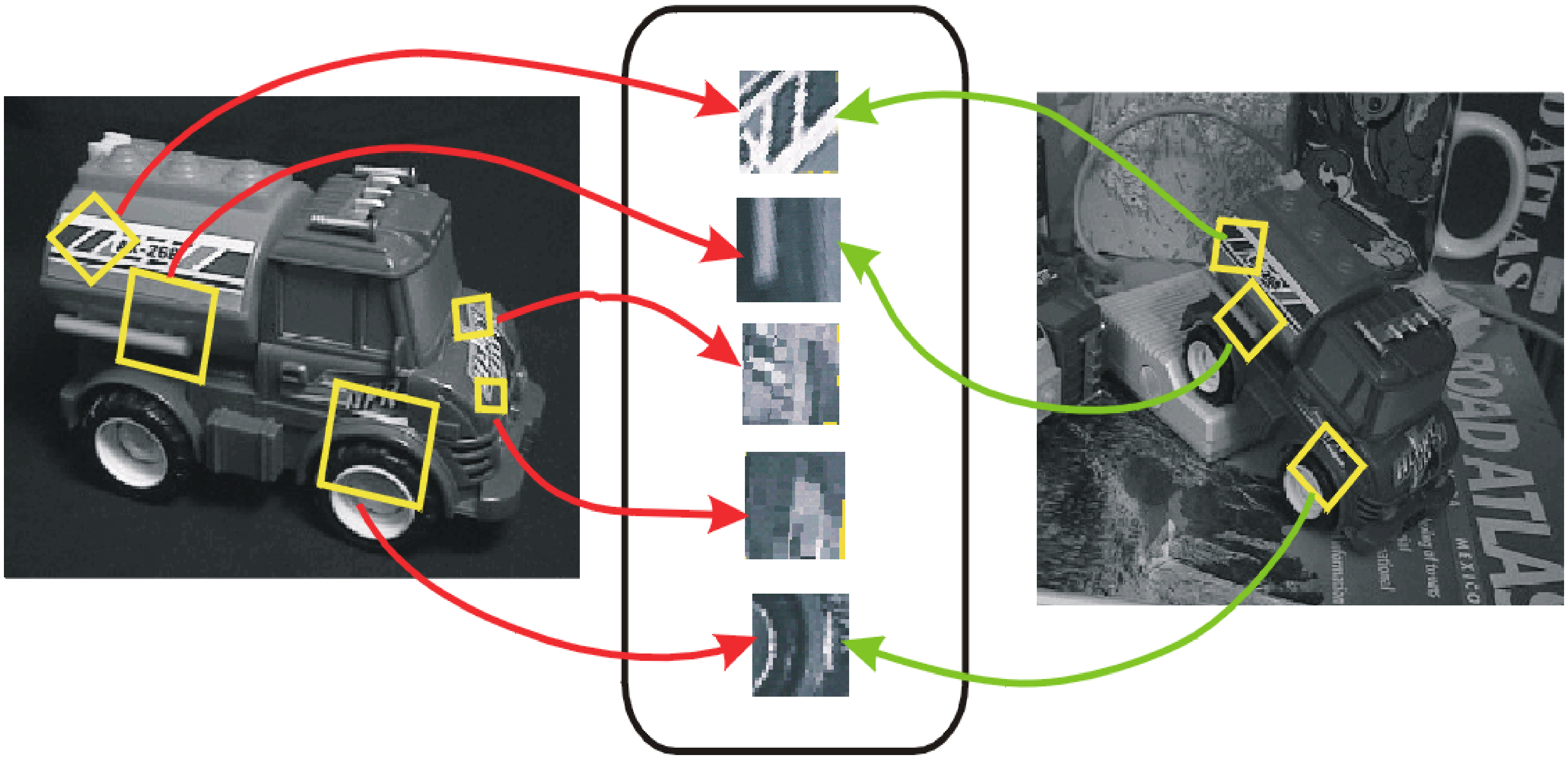


Image Matching

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

Distinctiveness

- can differentiate a large database of objects

Quantity

- hundreds or thousands in a single image

Efficiency

- real-time performance achievable

Generality

- exploit different types of features in different situations

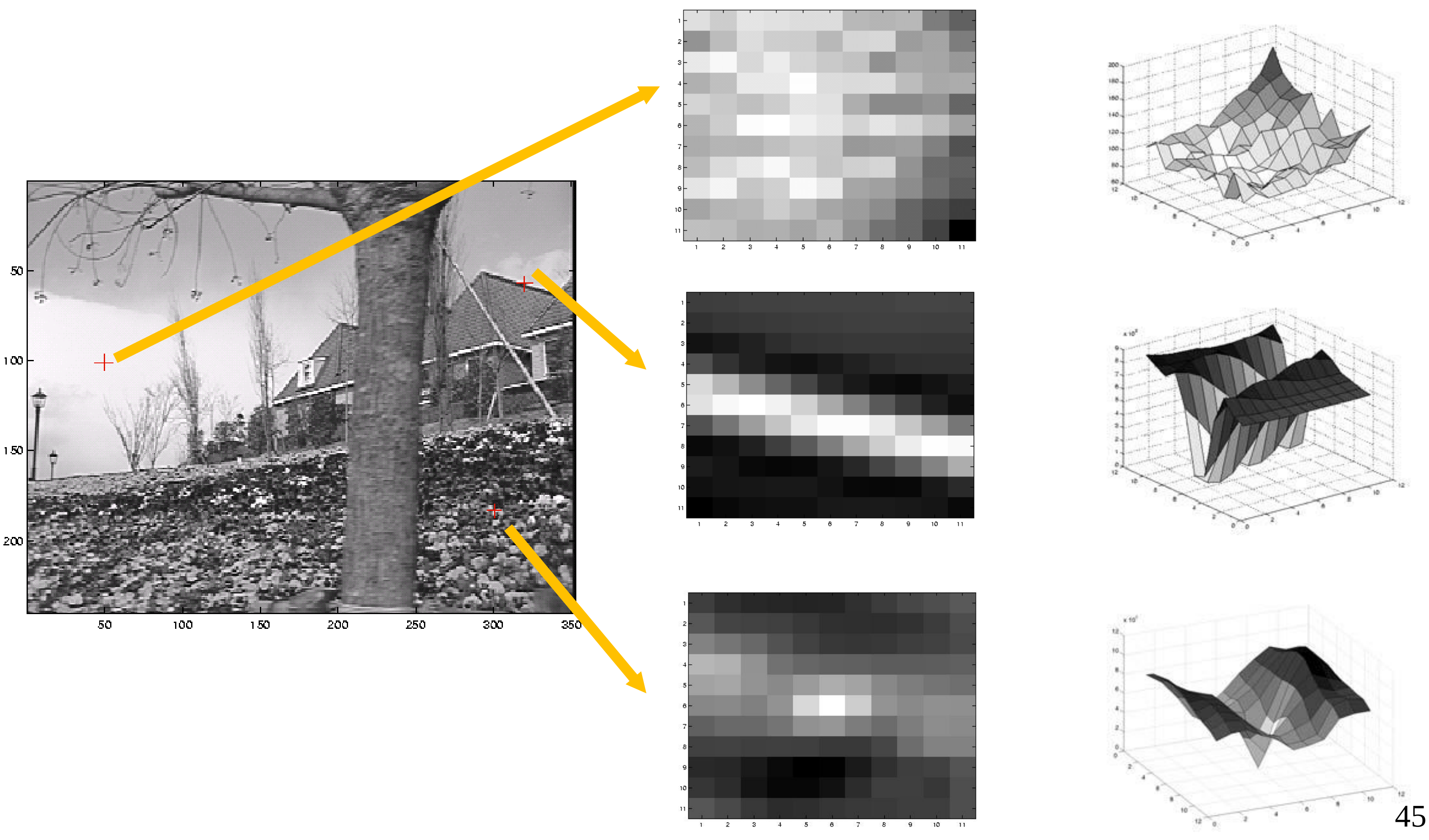
More motivation…

Feature points are used for:

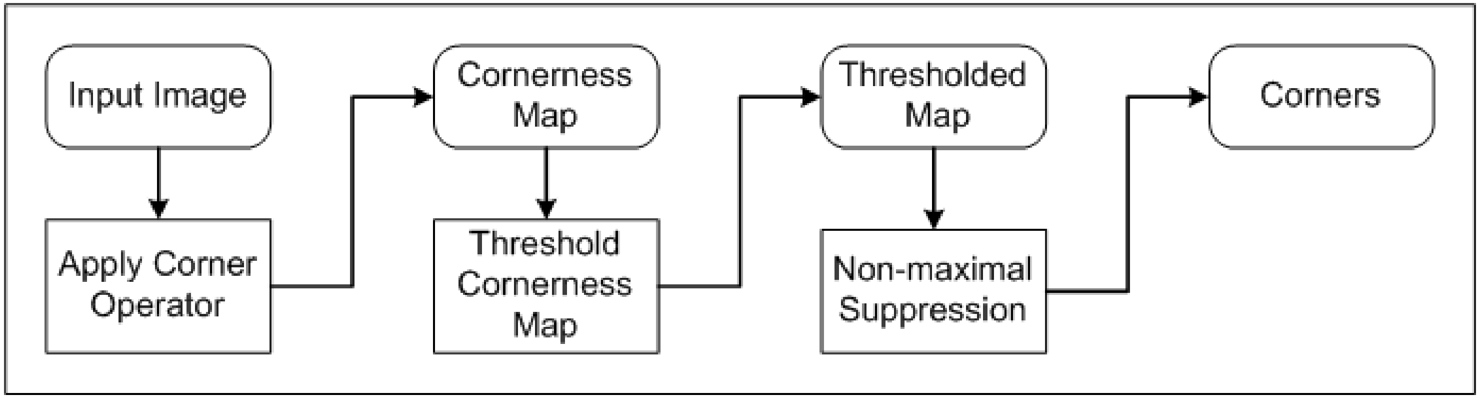
- Image alignment (e.g., mosaics)
- 3D reconstruction
- Motion tracking
- Object recognition
- Indexing and database retrieval
- Robot navigation
- … other

Interest point candidates

auto-correlation



Steps in Corner Detection

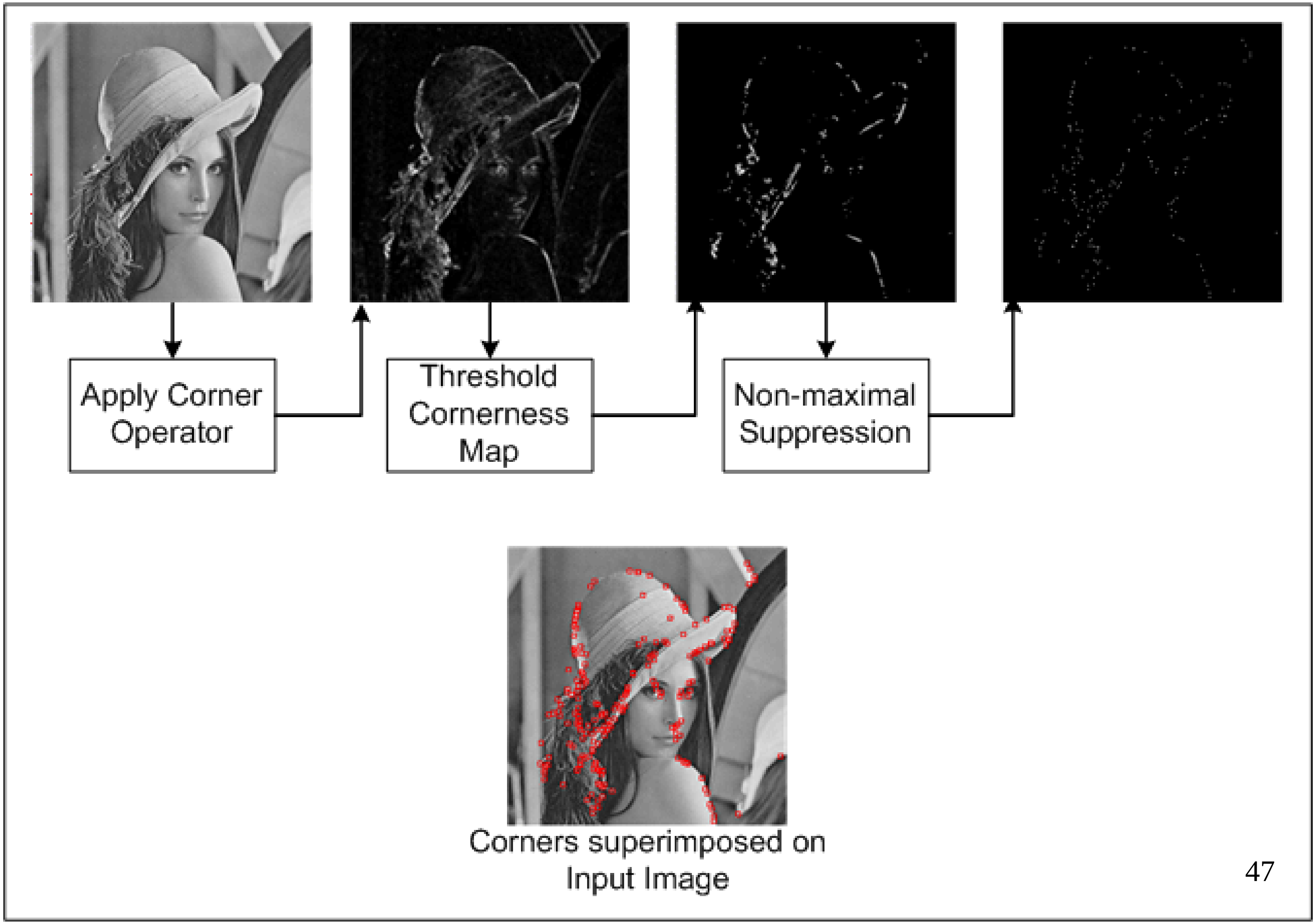


1. For each pixel, the corner operator is applied to obtain a cornerness measure for this pixel.

2. Threshold cornerness map to eliminate weak corners.

3. Apply non-maximal suppression to eliminate points whose cornerness measure is not larger than the cornerness values of all points within a certain distance.

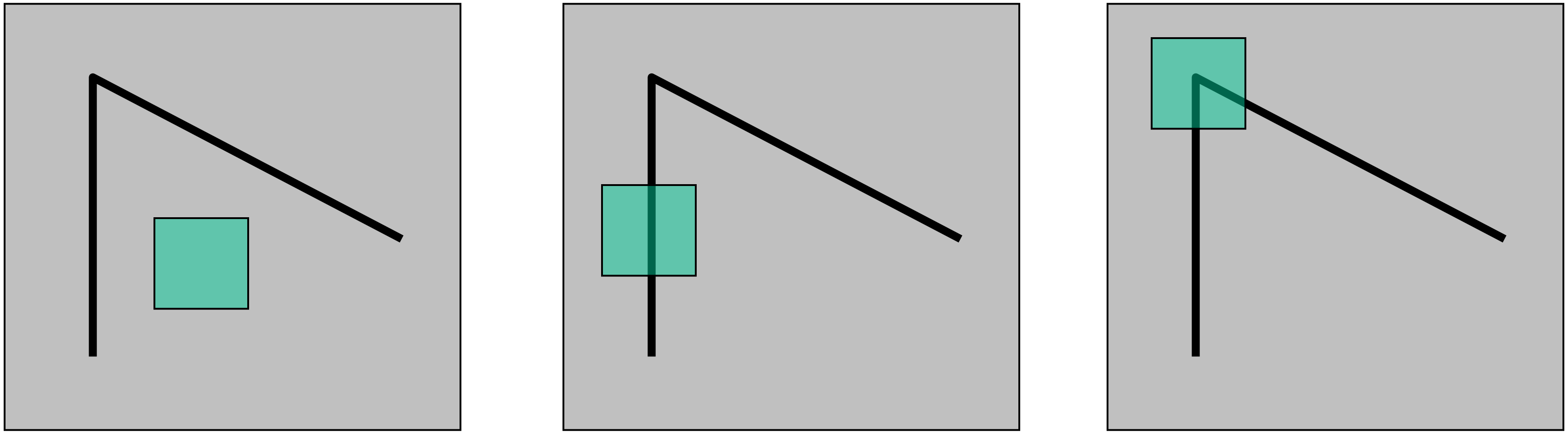
Steps in Corner Detection (cont'd)



Local measures of uniqueness

Suppose we only consider a small window of pixels

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

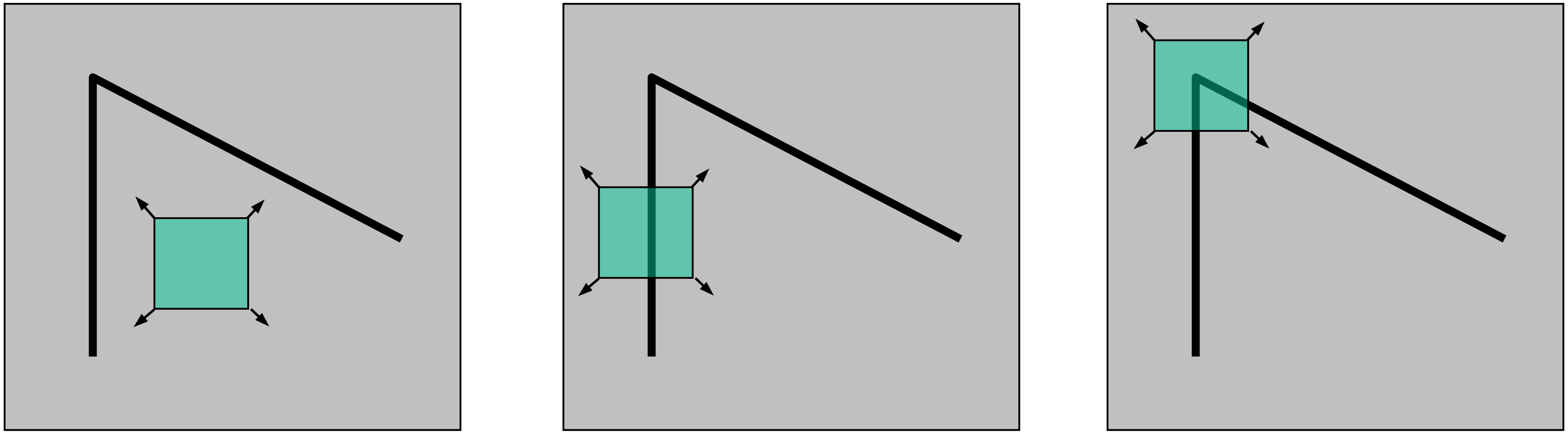


Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute.

Feature detection

Local measure of feature 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

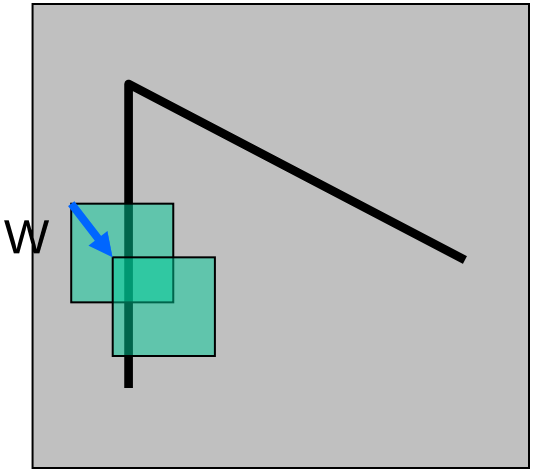
"corner":
significant change in all directions

Slide adapted from Darya Frolova, Denis Simakov, Weizmann Institute.

Feature 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” of *E(u,v)*:



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

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 approx is good

$$I(x+u, y+v) \approx I(x,y) + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v$$

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

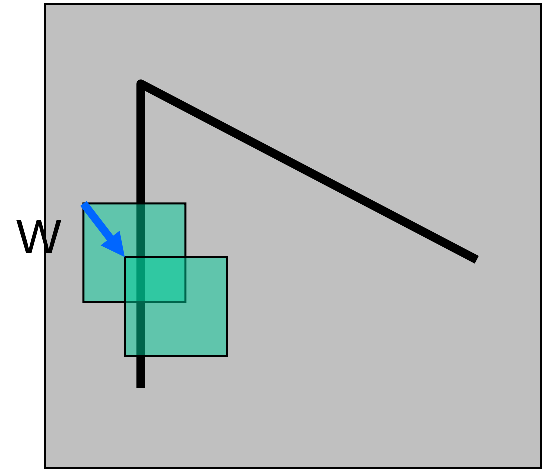
$$\text{shorthand: } I_x = \frac{\partial I}{\partial x}$$

Plugging this into the formula on the previous slide…

Feature 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
- this defines an "error" of E(u,v):



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

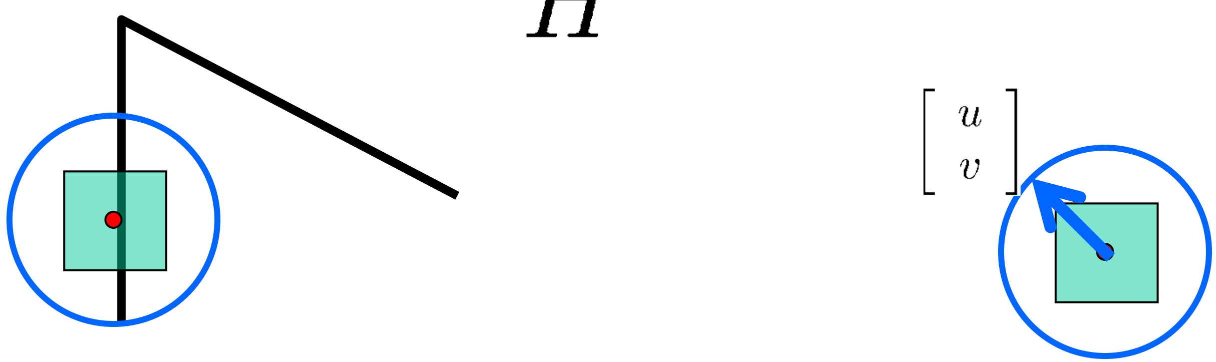
$$\approx \sum_{(x,y)\in W} \left[I(x,y) + [I_x \; I_y] \begin{bmatrix} u \\ v \end{bmatrix} - I(x,y)\right]^2$$

$$\approx \sum_{(x,y)\in W} \left[[I_x \; I_y] \begin{bmatrix} u \\ v \end{bmatrix}\right]^2$$

Feature detection: the math

This can be rewritten:

$$E(u,v) = \sum_{(x,y)\in W} [u \;\; v] \underbrace{\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$



For the example above

- You can move the center of the green window to anywhere on the blue unit circle
- Which directions will result in the largest and smallest E values?
- We will show that we can find these directions by looking at the eigenvectors of *H*

Feature detection: the error function

➢ A new corner measurement by investigating the **shape** of the error function

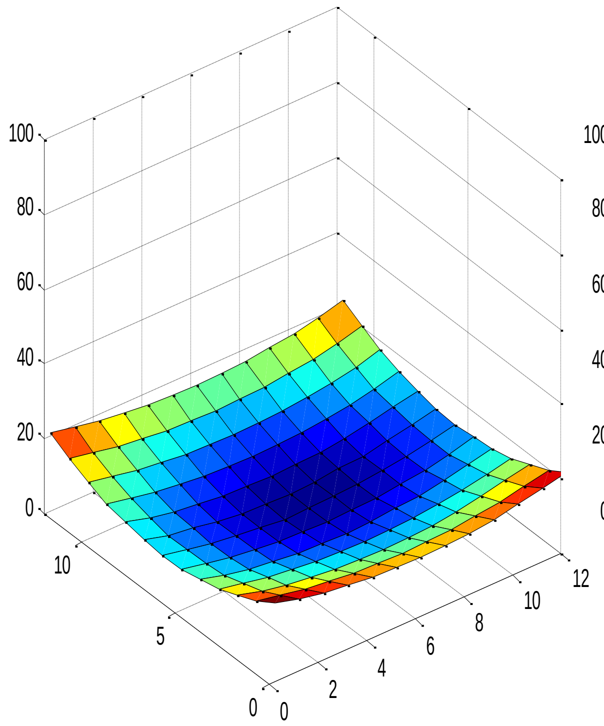
$$E(u,v) = \sum_{(x,y)\in W} [u\ v] \underbrace{\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

$\mathbf{u}^T H \mathbf{u}$ represents a **quadratic function**;

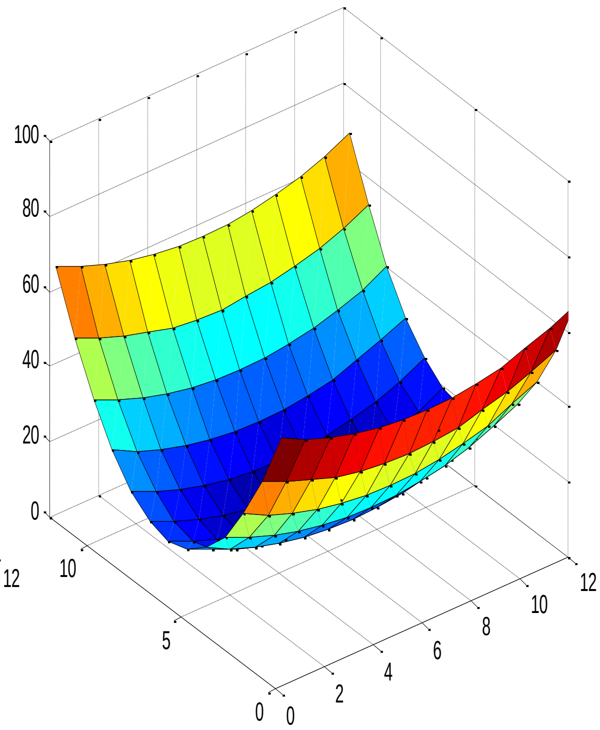
Thus, we can analyze E's shape by <u>looking at the property of $\mathbf{H}$</u>

Feature detection: the error function

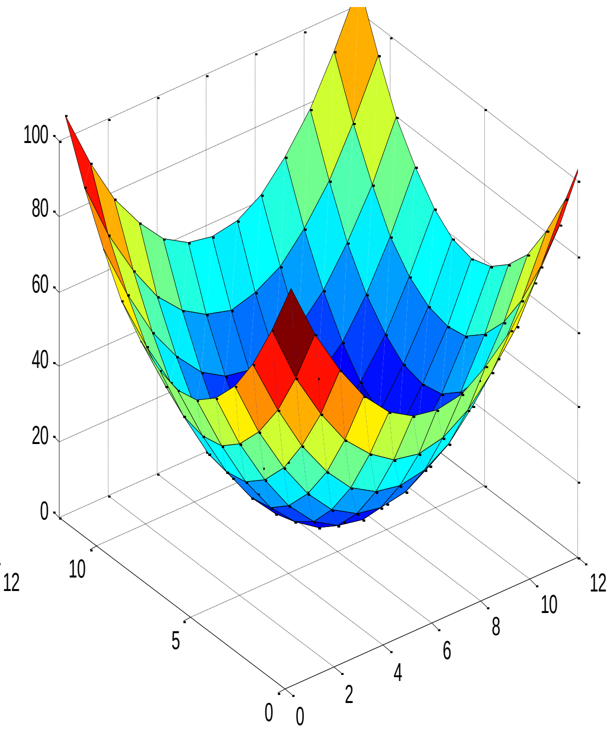
High-level idea: what shape of the error function will we prefer for features?



flat



edge



corner

Quadratic forms

Quadratic form (homogeneous polynomial of degree two) of n variables x_i

$$\sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \le j}}^{n} c_{ij} x_i x_j$$

Examples

$$4x_1^2 + 5x_2^2 + 3x_3^2 + 2x_1x_2 + 4x_1x_3 + 6x_2x_3$$

$$= \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} 4 & 1 & 2 \\ 1 & 5 & 3 \\ 2 & 3 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Symmetric matrices

Quadratic forms can be represented by a real symmetric matrix A where

$$a_{ij} = \begin{cases} c_{ij} & \text{if } i = j, \\ \frac{1}{2}c_{ij} & \text{if } i < j, \\ \frac{1}{2}c_{ji} & \text{if } i > j. \end{cases}$$

$$\sum_{\substack{i=1 \\ i \le j}}^{n} \sum_{j=1}^{n} c_{ij} x_i x_j = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j$$

$$= \begin{pmatrix} x_1 & \ldots & x_n \end{pmatrix} \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

$$= \mathbf{x}^t A \mathbf{x}$$

Eigenvalues of symmetric matrices

suppose $A \in \mathbf{R}^{n\times n}$ is symmetric, *i.e.*, $A = A^T$

fact: the eigenvalues of A are real

suppose $Av = \lambda v$, $v \neq 0$, $v \in \mathbf{C}^n$

$$\overline{v}^T A v = \overline{v}^T (Av) = \lambda \overline{v}^T v = \lambda \sum_{i=1}^{n} |v_i|^2$$

$$\overline{v}^T A v = \overline{(Av)}^T v = \overline{(\lambda v)}^T v = \overline{\lambda} \sum_{i=1}^{n} |v_i|^2$$

we have $\lambda = \overline{\lambda}$, *i.e.*, $\lambda \in \mathbf{R}$

(hence, can assume $v \in \mathbf{R}^n$)

Eigenvectors of symmetric matrices

suppose $A \in \mathbf{R}^{n \times n}$ is symmetric, $i.e.$, $A = A^T$

fact: there is a set of orthonormal eigenvectors of A

$$A = Q \Lambda Q^T$$

where Q is an orthogonal matrix (the columns of which are eigenvectors of A), and Λ is real and diagonal (having the eigenvalues of A on the diagonal).

Eigenvectors of symmetric matrices

suppose $A \in \mathbf{R}^{n \times n}$ is symmetric, $i.e.$, $A = A^T$

fact: there is a set of orthonormal eigenvectors of A

$$A = Q\Lambda Q^T$$

$$\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$$

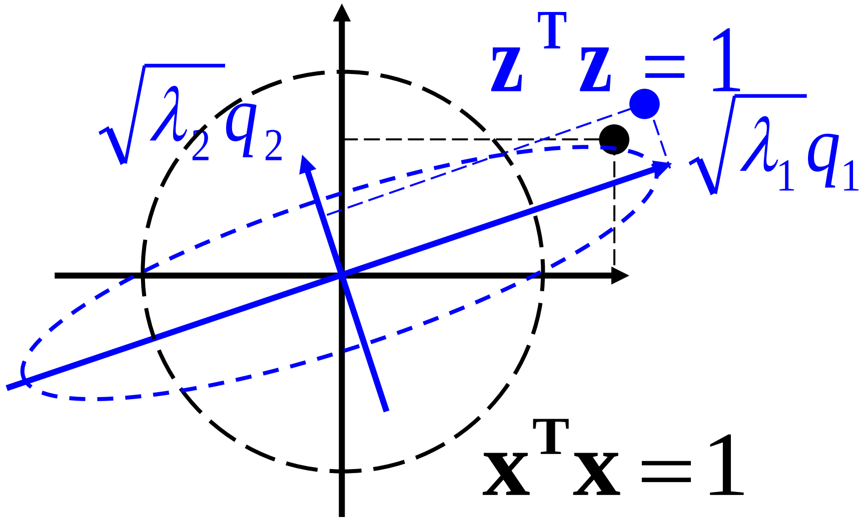
$$= \mathbf{x}^{\mathrm{T}}\mathbf{Q}\,\boldsymbol{\Lambda}\,\mathbf{Q}^{\mathrm{T}}\mathbf{x}$$

$$= \left(\mathbf{Q}^{\mathrm{T}}\mathbf{x}\right)^{\mathrm{T}}\boldsymbol{\Lambda}\left(\mathbf{Q}^{\mathrm{T}}\mathbf{x}\right)$$

$$= \mathbf{y}^{\mathrm{T}}\boldsymbol{\Lambda}\mathbf{y}$$

$$= \left(\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{y}\right)^{\mathrm{T}}\left(\boldsymbol{\Lambda}^{\frac{1}{2}}\mathbf{y}\right)$$

$$= \mathbf{z}^{\mathrm{T}}\mathbf{z}$$



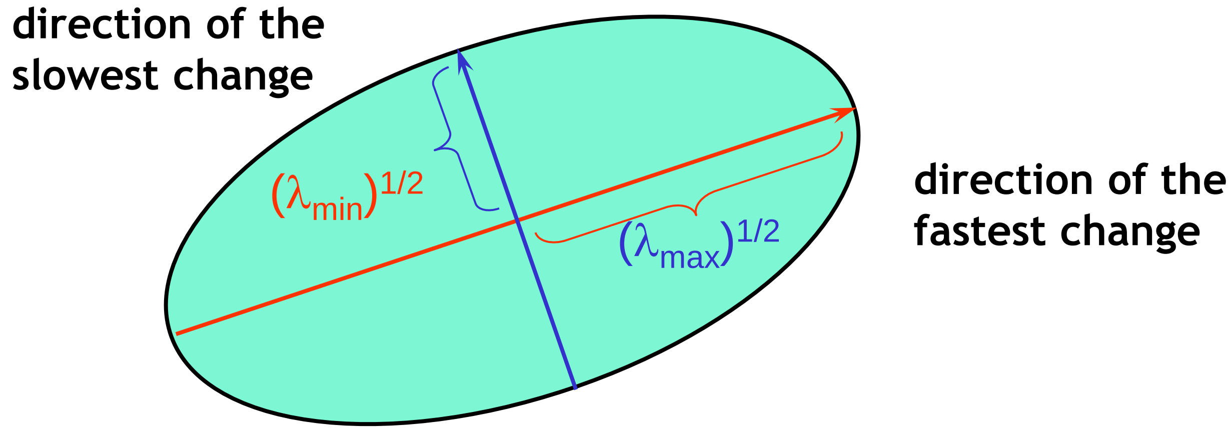
Harris corner detector

Intensity change in shifting window: eigenvalue analysis

$$E(u,v) \cong [u,v]\, H \begin{bmatrix} u \\ v \end{bmatrix}$$

λ_-, λ_+ – eigenvalues of $\mathbf{H}$

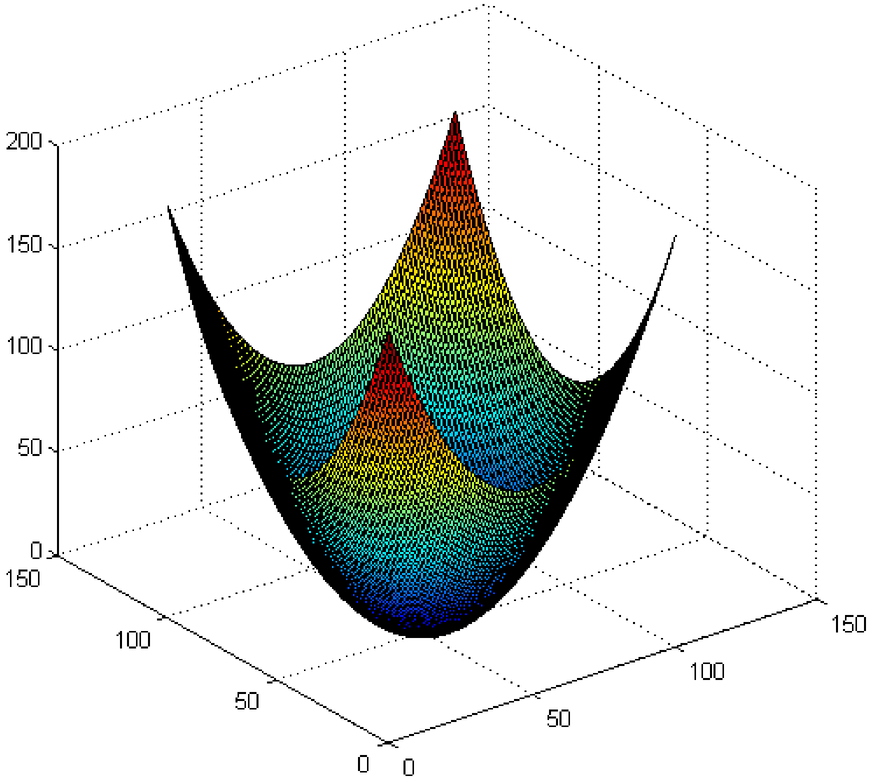
We can visualize H as an ellipse with axis lengths and directions determined by its eigenvalues and eigenvectors.

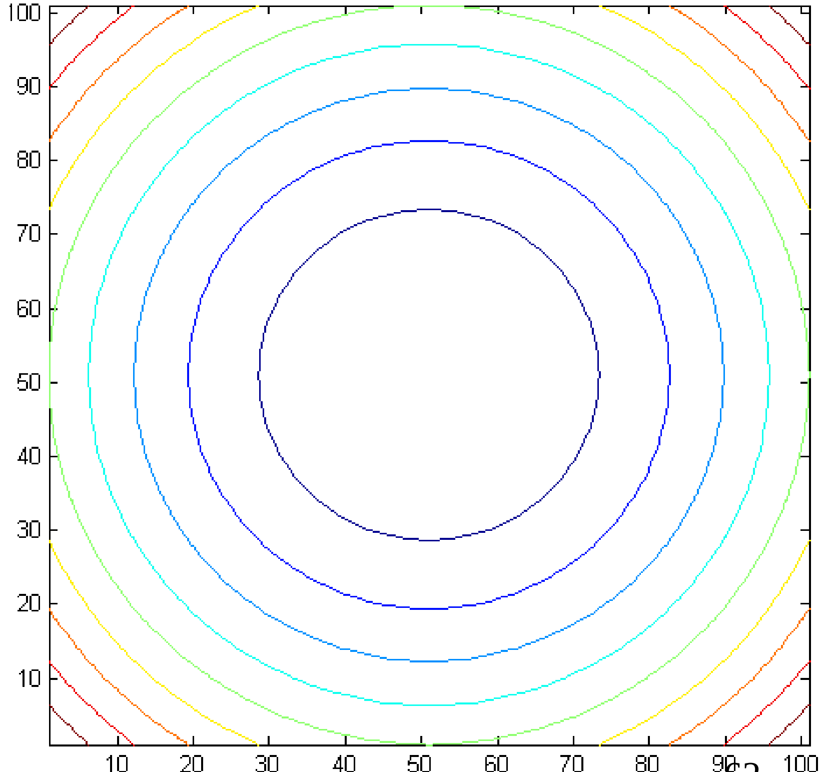


Ellipse $E(u,v)$ = const

Visualize quadratic functions

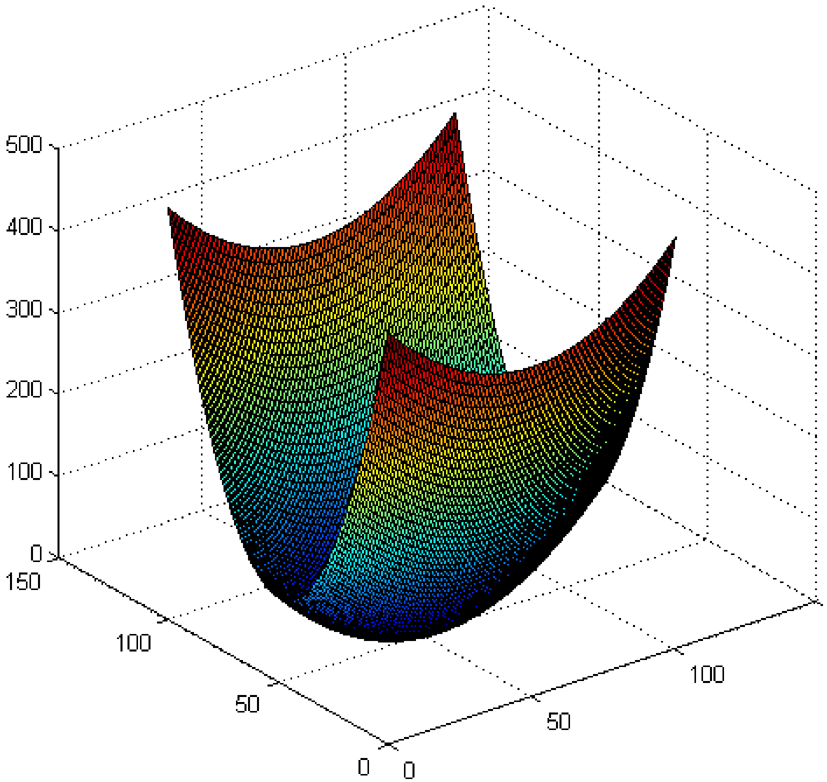
$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T$$

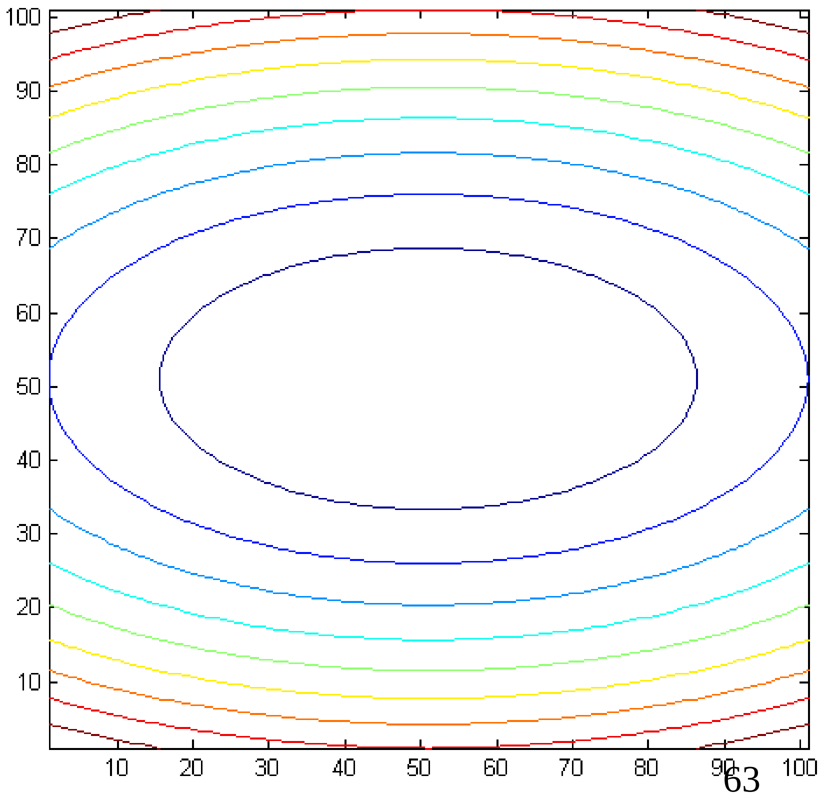




Visualize quadratic functions

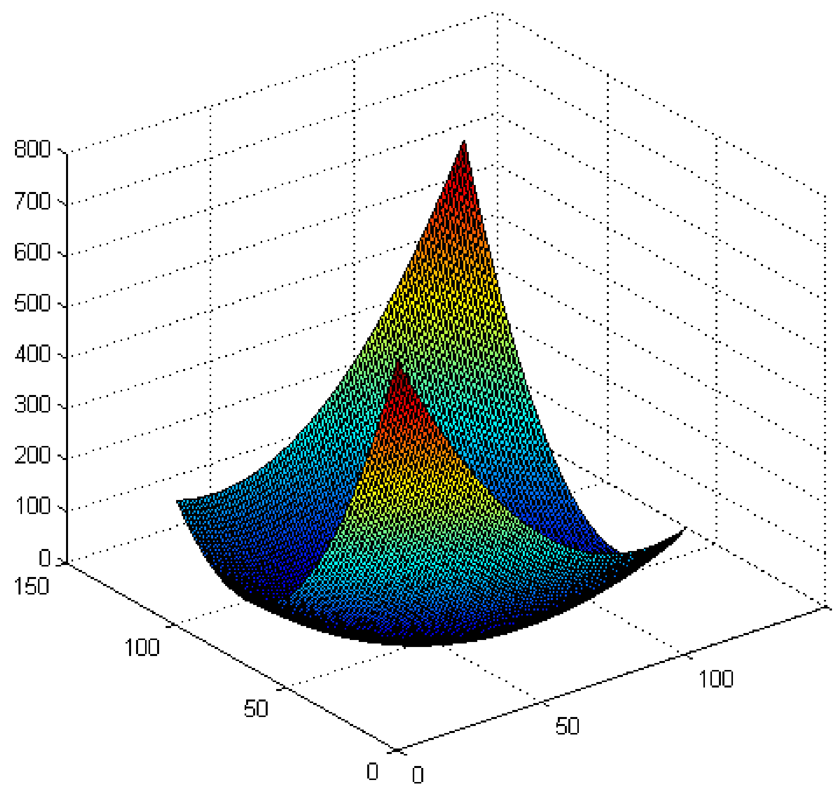
$$\mathbf{A} = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T$$

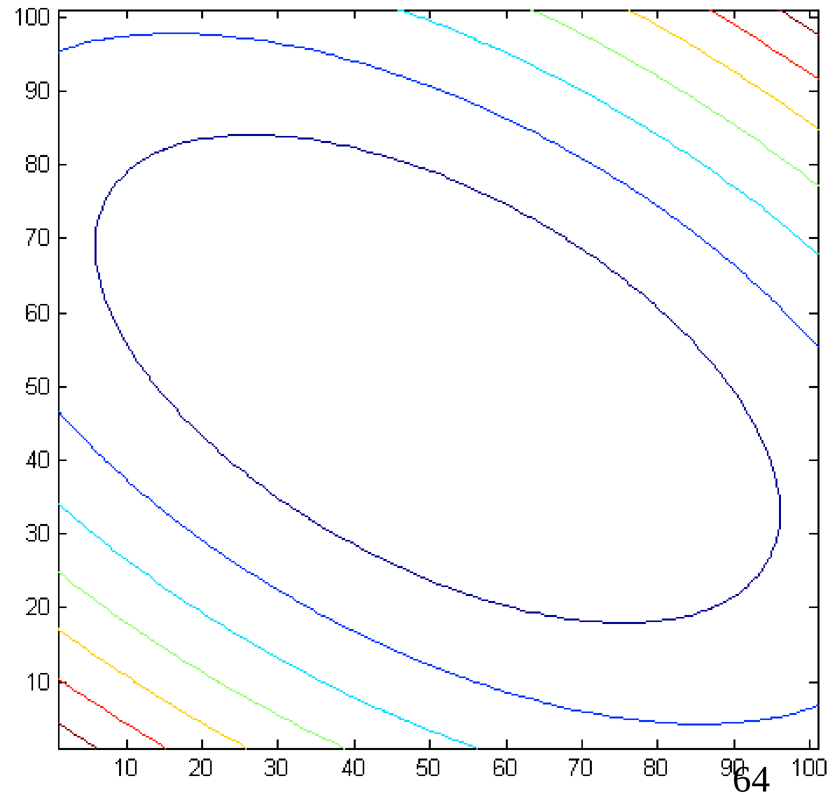




Visualize quadratic functions

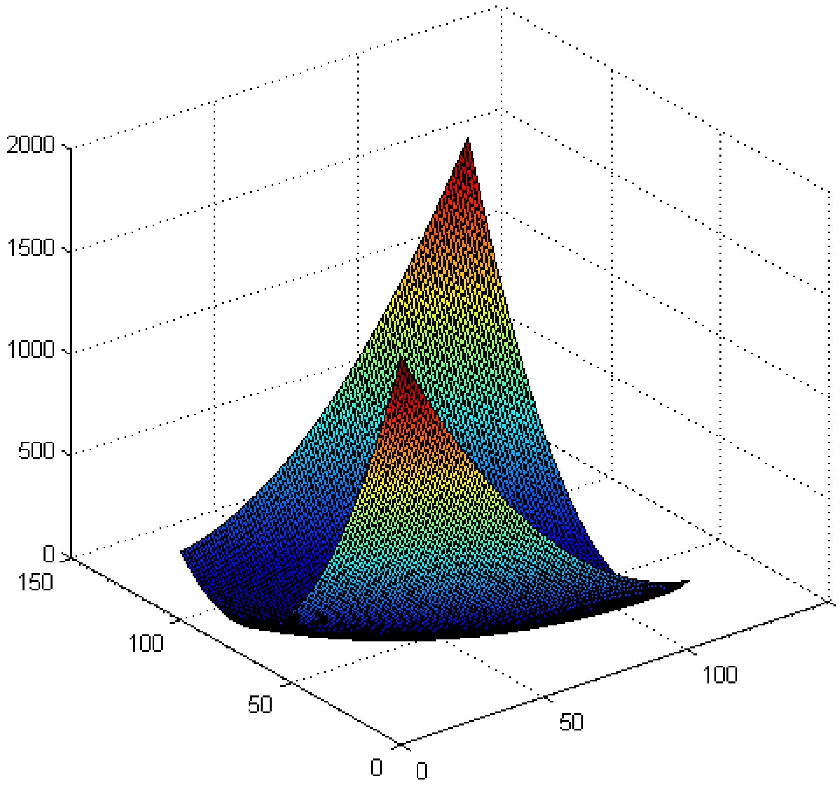
$$\mathbf{A} = \begin{bmatrix} 3.25 & 1.30 \\ 1.30 & 1.75 \end{bmatrix} = \begin{bmatrix} 0.50 & -0.87 \\ -0.87 & -0.50 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 0.50 & -0.87 \\ -0.87 & -0.50 \end{bmatrix}^T$$

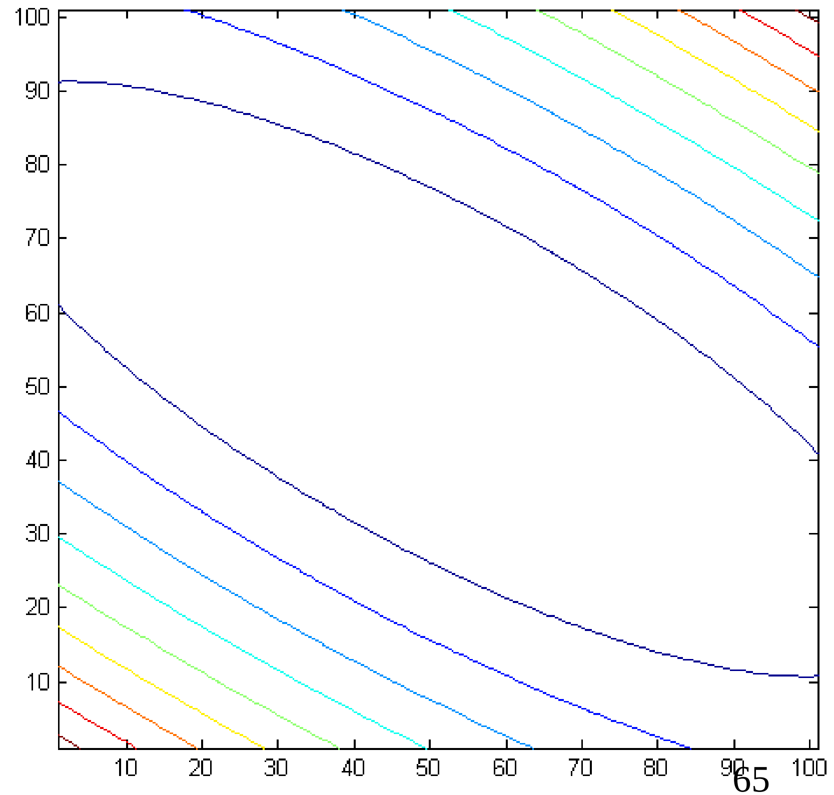




Visualize quadratic functions

$$\mathbf{A} = \begin{bmatrix} 7.75 & 3.90 \\ 3.90 & 3.25 \end{bmatrix} = \begin{bmatrix} 0.50 & -0.87 \\ -0.87 & -0.50 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix} \begin{bmatrix} 0.50 & -0.87 \\ -0.87 & -0.50 \end{bmatrix}^T$$

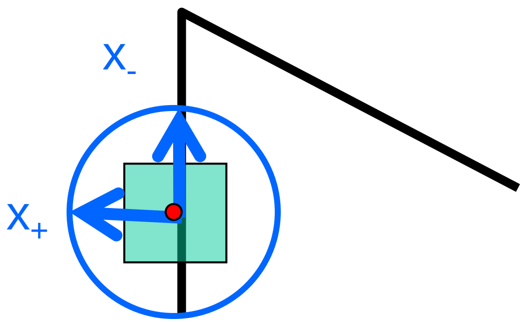


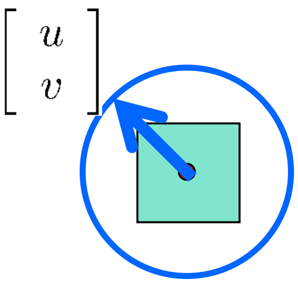


Feature detection: the math

This can be rewritten:

$$E(u,v) = \sum_{(x,y)\in W} [u\ v] \underbrace{\begin{bmatrix} I_x^2 & I_xI_y \\ I_yI_x & I_y^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$





Eigenvalues and eigenvectors of H

- Define shifts with the smallest and largest change (E value)
- x_+ = direction of largest increase in E.
- λ_+ = amount of increase in direction x_+
- x_- = direction of smallest increase in E.
- λ- = amount of increase in direction x_+

$$\begin{aligned} Hx_+ &= \lambda_+ x_+ \\ Hx_- &= \lambda_- x_- \end{aligned}$$

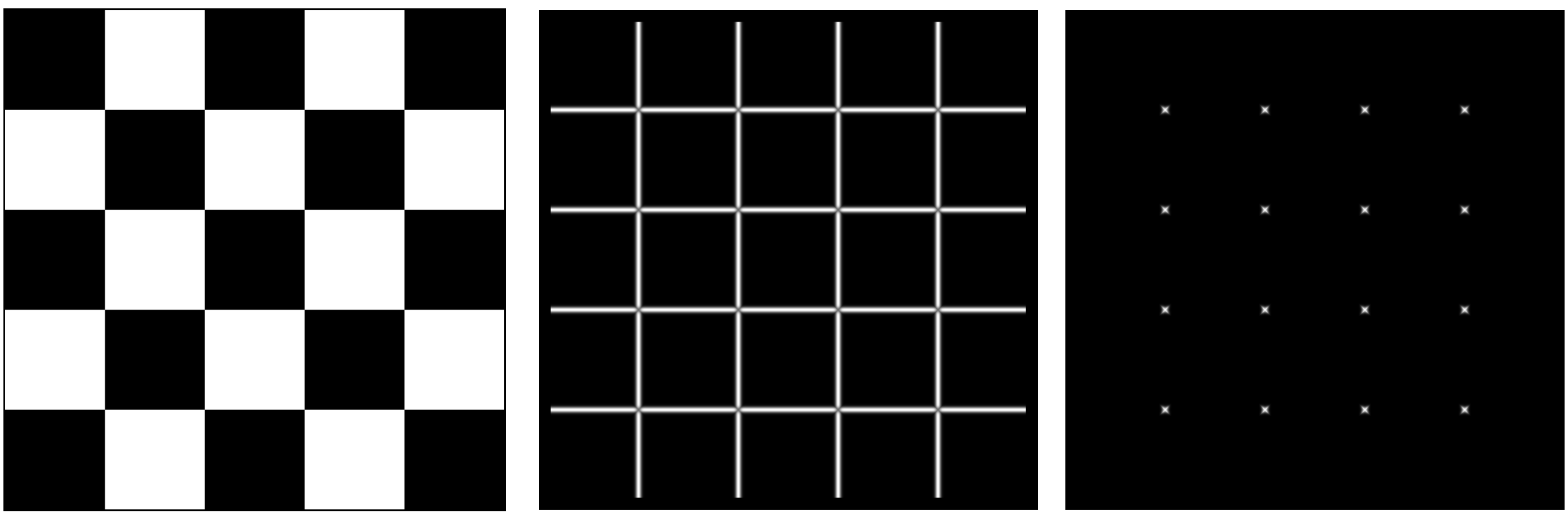
Feature detection: the math

How are λ_+, x_+, λ_-, and x_+ 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_-) of *H*

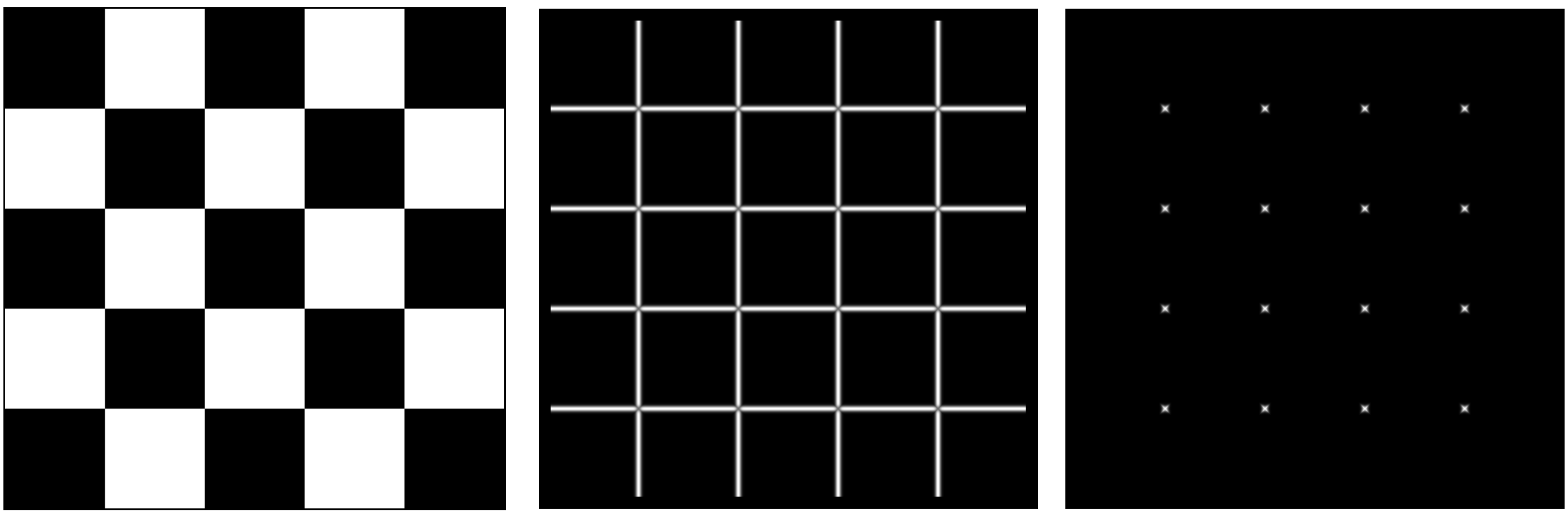


I λ_+ λ_-

Feature detection summary (Kanade-Tomasi)

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_- > threshold)
- Choose those points where λ_- is a local maximum as features



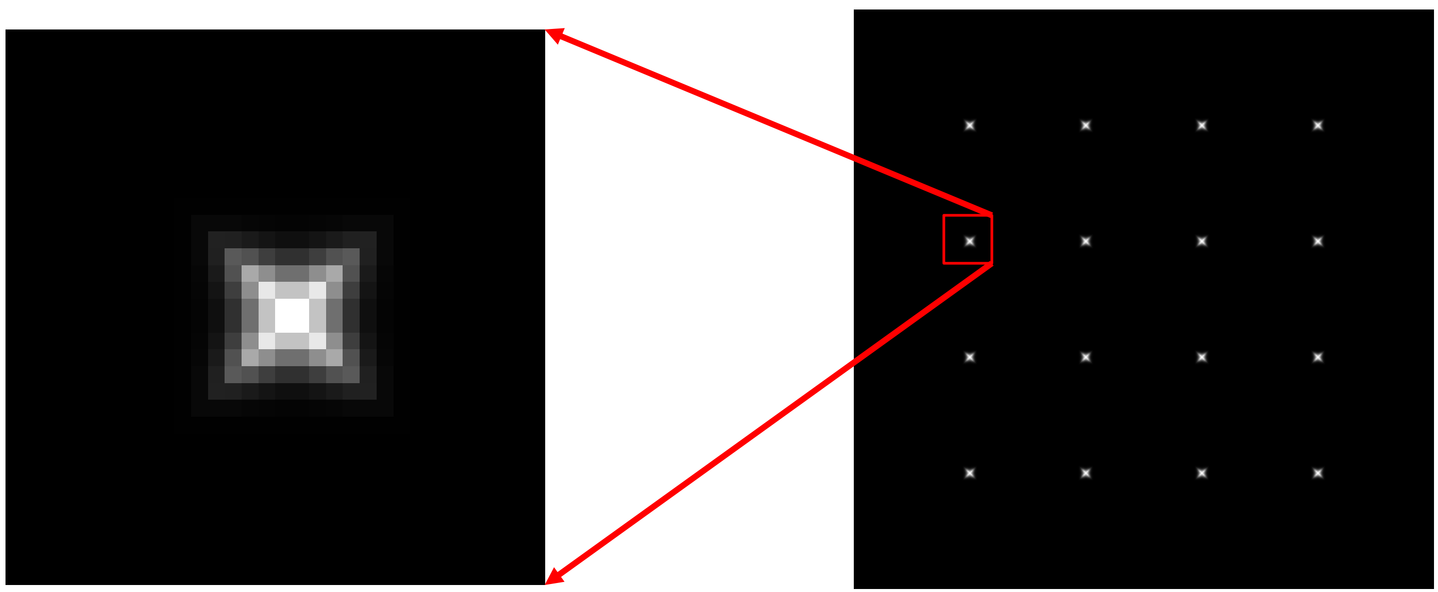
I λ_+ λ_-

J. Shi and C. Tomasi (June 1994). "Good Features to Track". *9th IEEE Conference on Computer Vision and Pattern Recognition*. Springer.

Feature 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_- > threshold)
- Choose those points where λ_- is a local maximum as features



λ_-

The Harris operator

λ_- is a variant of the "Harris operator" for feature detection ($\lambda_- = \lambda_1$; $\lambda_+ = \lambda_2$)

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$
$$= \frac{determinant(H)}{trace(H)}$$

- The *trace* is the sum of the diagonals, i.e., $trace(H) = h_{11} + h_{22}$
- Very similar to λ_- 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

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

The Harris operator

Measure of corner response (Harris):

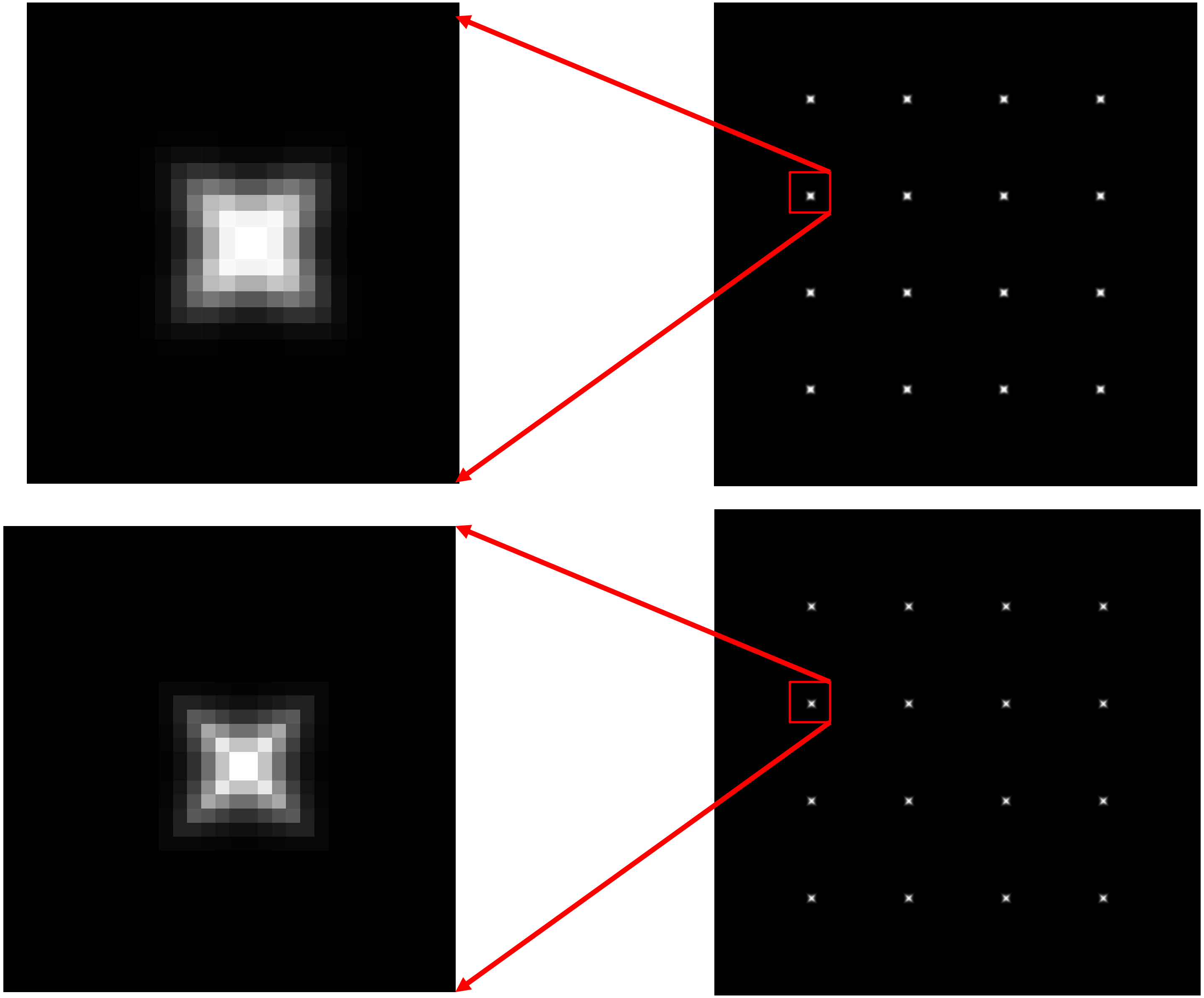
$$R = \det H - k\left(\operatorname{trace} H\right)^2$$

With:

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

(*k* – empirical constant, *k* = 0.04-0.06)

The Harris operator

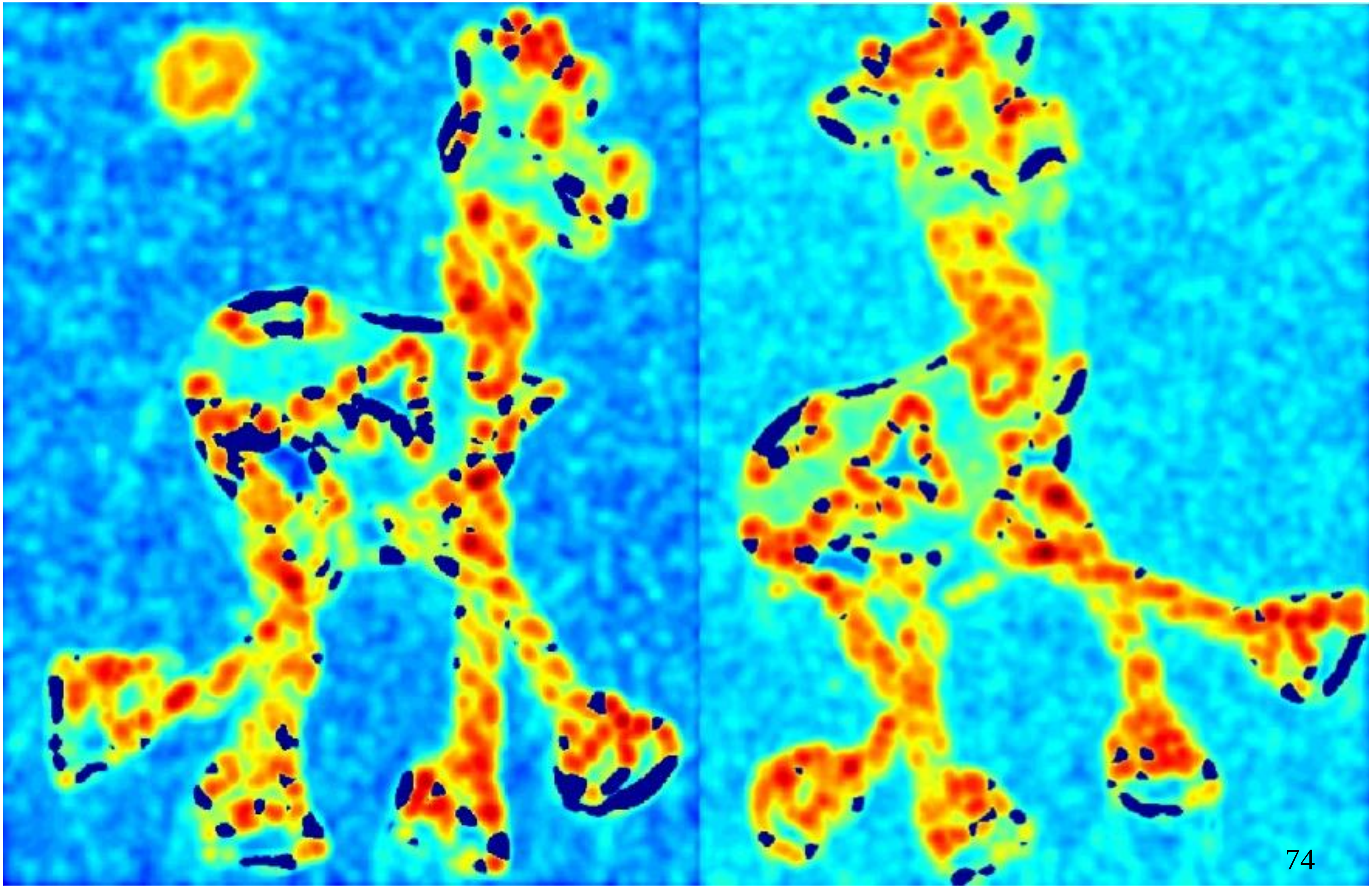


Harris operator

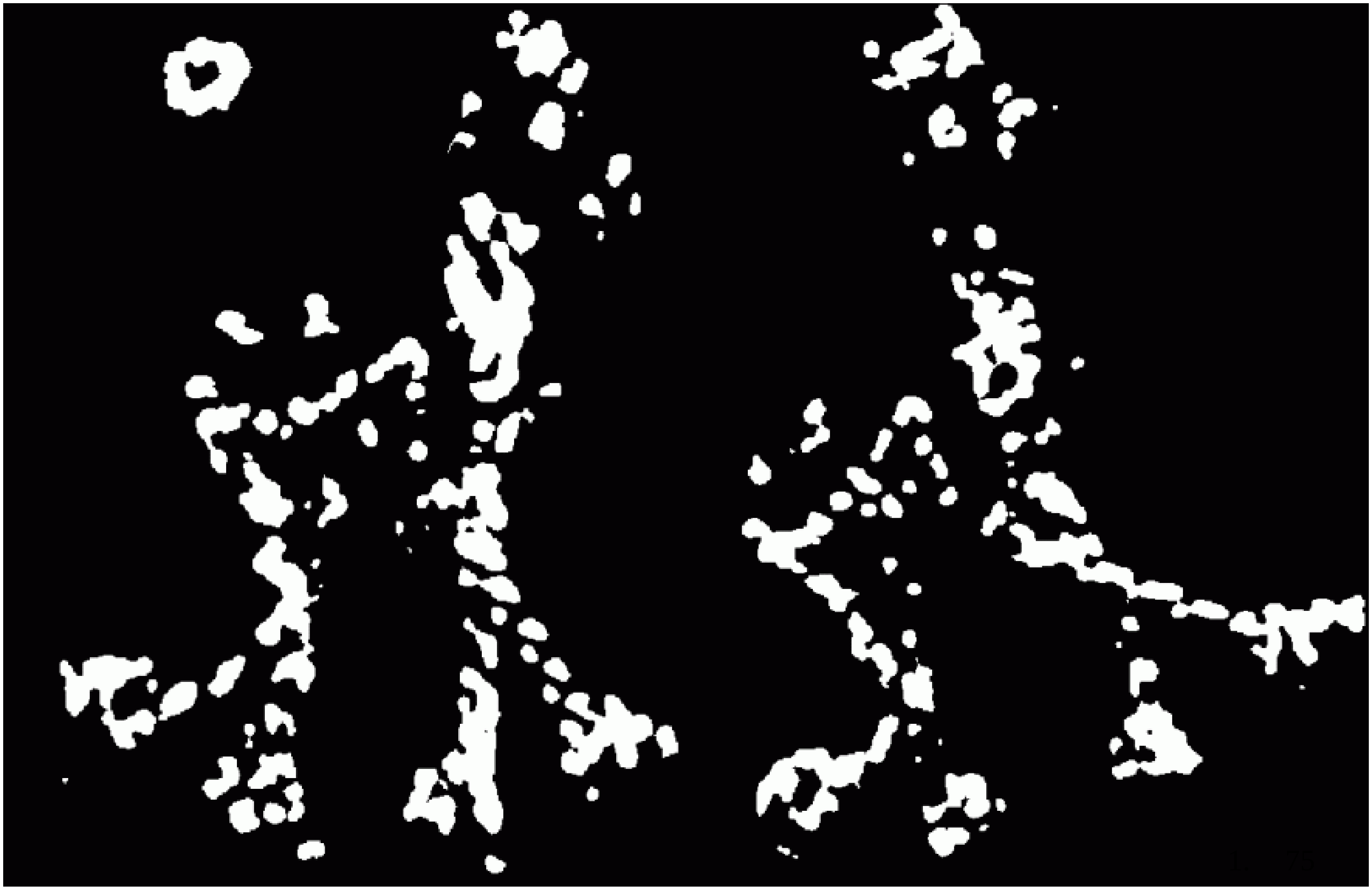
λ_-

Harris detector example

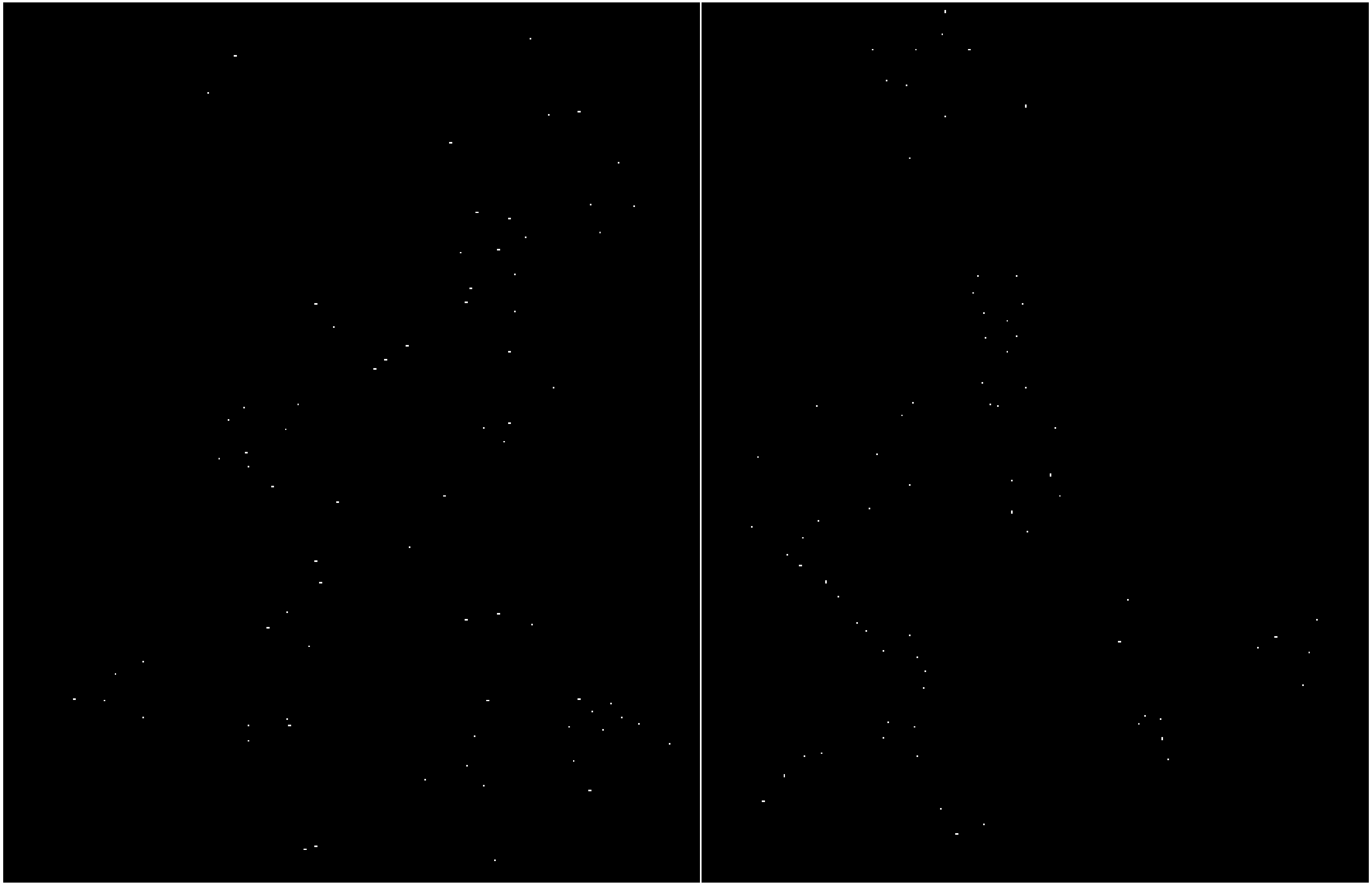
f value (red high, blue low)



Threshold (f > value)



Find local maxima of f



Harris features (in red)

Harris detector: Steps

1. Compute Gaussian derivatives at each pixel
2. Compute second moment matrix *H* in a Gaussian window around each pixel
3. Compute corner response function *R*
4. Threshold *R*
5. Find local maxima of response function (non-maximum suppression)

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

α: constant (0.04 to 0.06)

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

Thank you!

Quick review: eigenvalue/eigenvector

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

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