Unit 2 – Image Features

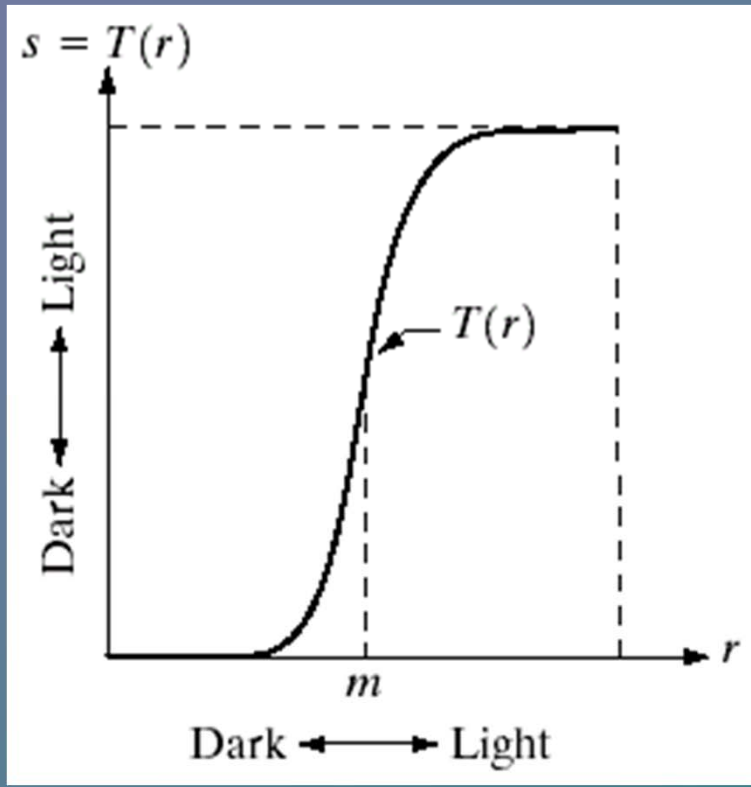
- **Intensity transformation**
- **Edge Detection**
- **Corner Detection**
- **SIFT**

Reading: Szeliski Sec. 3.1, 3.2, 4.1, 4.2.1

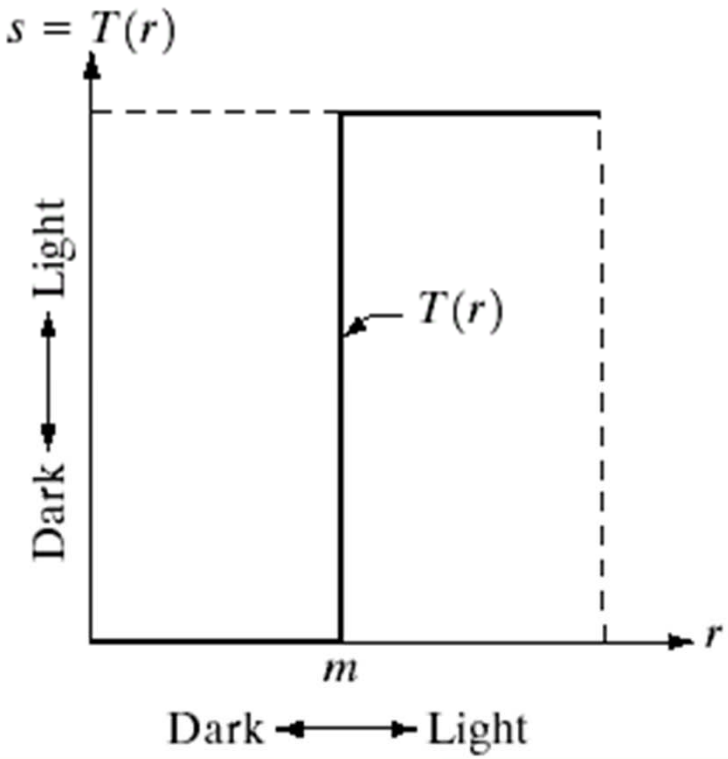
Grayscale Transformation

Point processing: s = T(r)

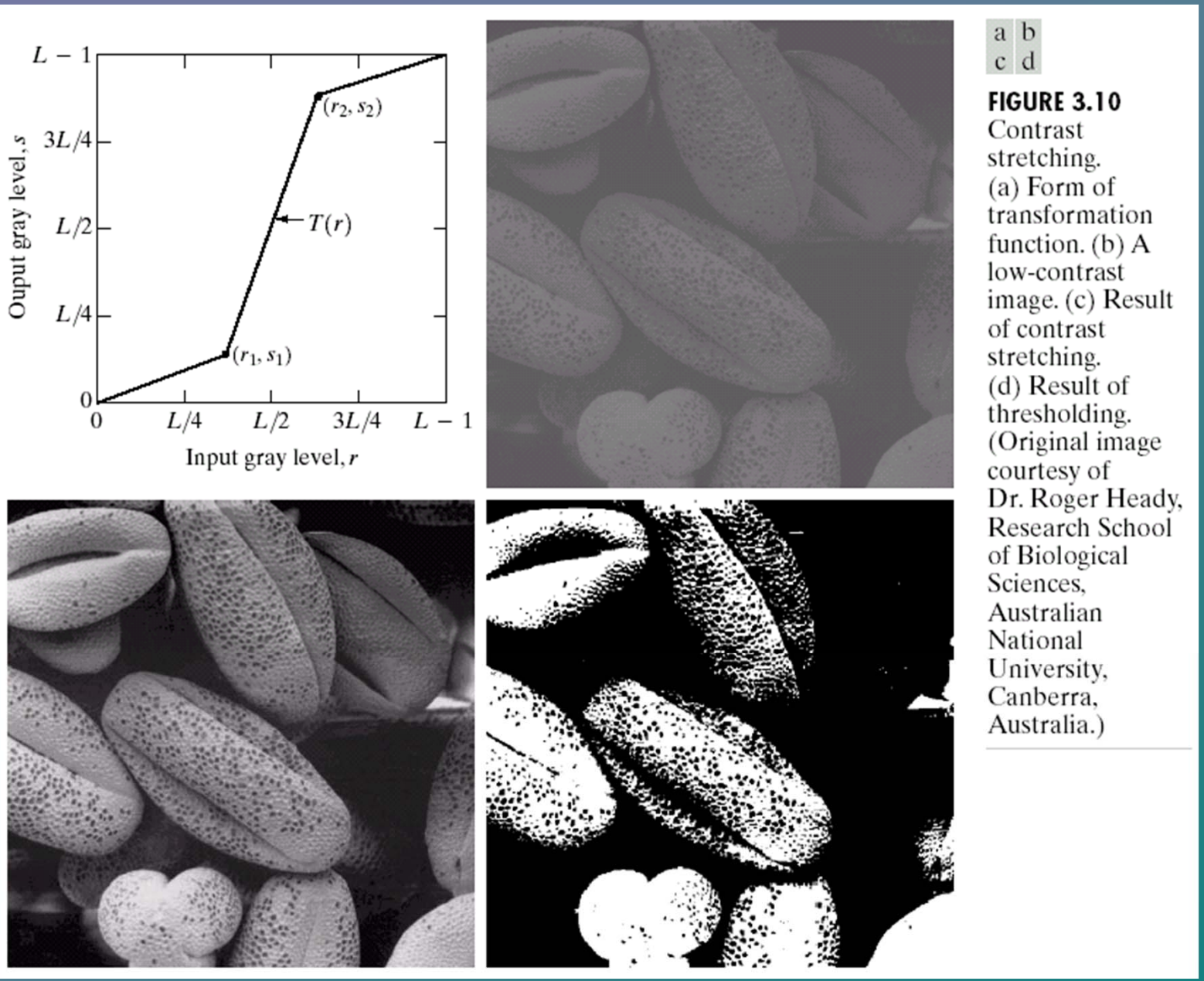
Contrast stretching



Thresholding



Contrast Stretching



a b
c d

FIGURE 3.10 Contrast stretching. (a) Form of transformation function. (b) A low-contrast image. (c) Result of contrast stretching. (d) Result of thresholding. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra, Australia.)

Algorithm 5.1: Histogram equalization

1. For an $N \times M$ image of G gray-levels (often 256), create an array H of length G initialized with 0 values.
2. Form the image histogram: Scan every pixel and increment the relevant member of H—if pixel p has intensity g_p, perform

$$H[g_p] = H[g_p] + 1 \, .$$

3. Form the cumulative image histogram H_c:

$$\begin{aligned} H_c[0] &= H[0] \, , \\ H_c[p] &= H_c[p-1] + H[p] \, , \quad p = 1, 2, \ldots, G-1 \, . \end{aligned}$$

4. Set

$$T[p] = \text{round}\left(\frac{G-1}{NM} H_c[p] \right) \, .$$

(This step obviously lends itself to more efficient implementation by constructing a look-up table of the multiples of $(G-1)/NM$, and making comparisons with the values in H_c, which are monotonically increasing.)

5. Rescan the image and write an output image with gray-levels g_q, setting

$$g_q = T[g_p] \, .$$

Histogram Equalization Example

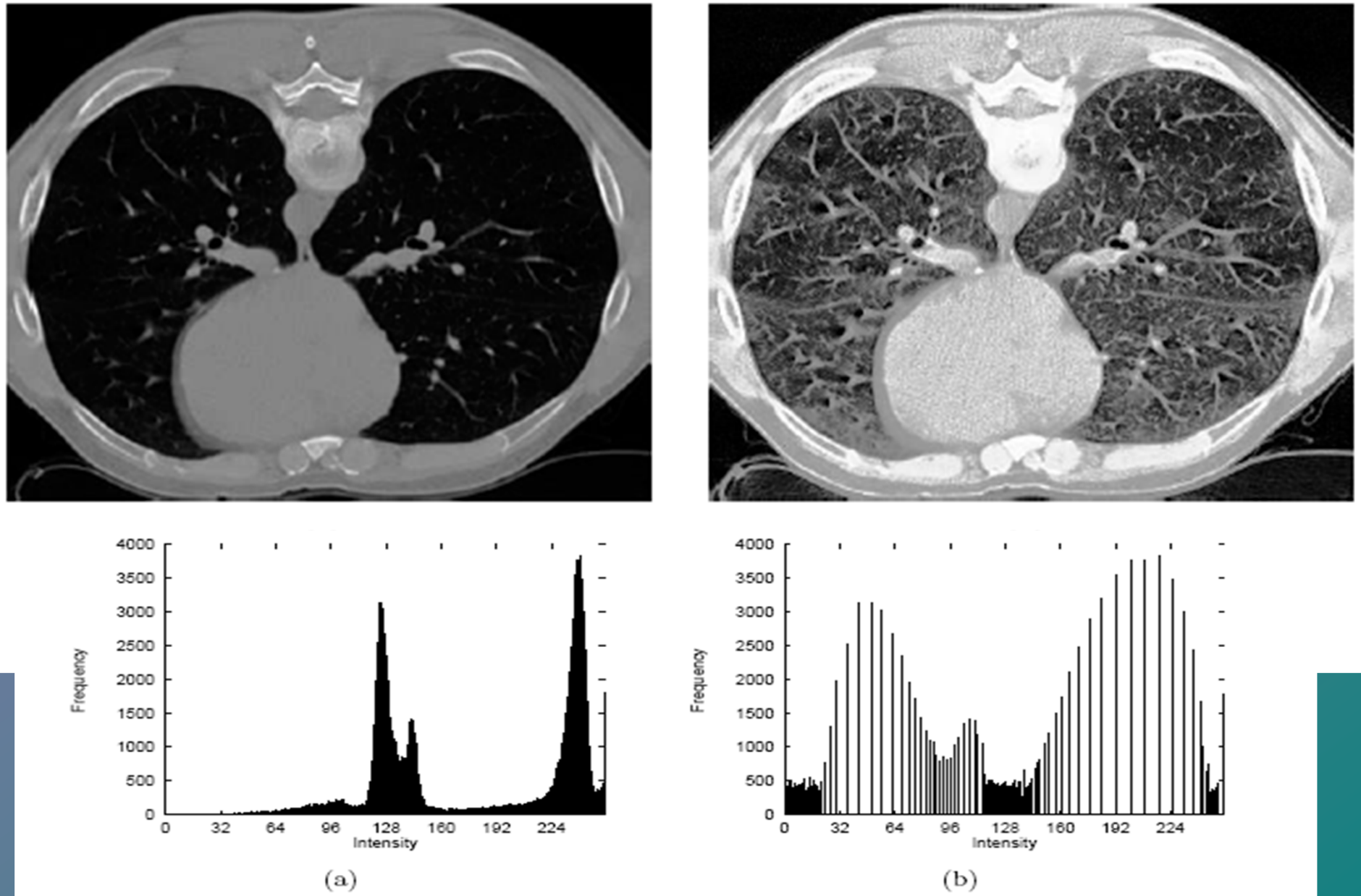


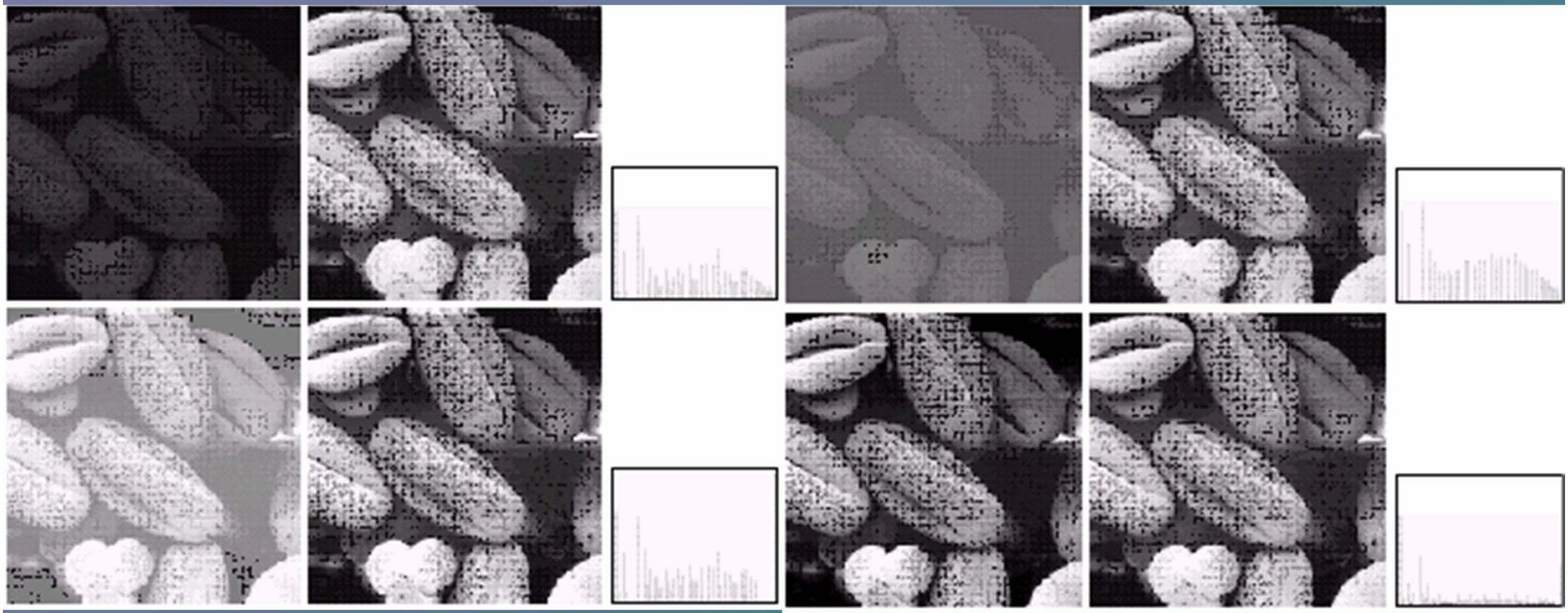
Figure 5.4: Histogram equalization: Original and equalized histograms corresponding to Figure 5.3a,b.

Histogram Equalization Example

Original image *Pout*

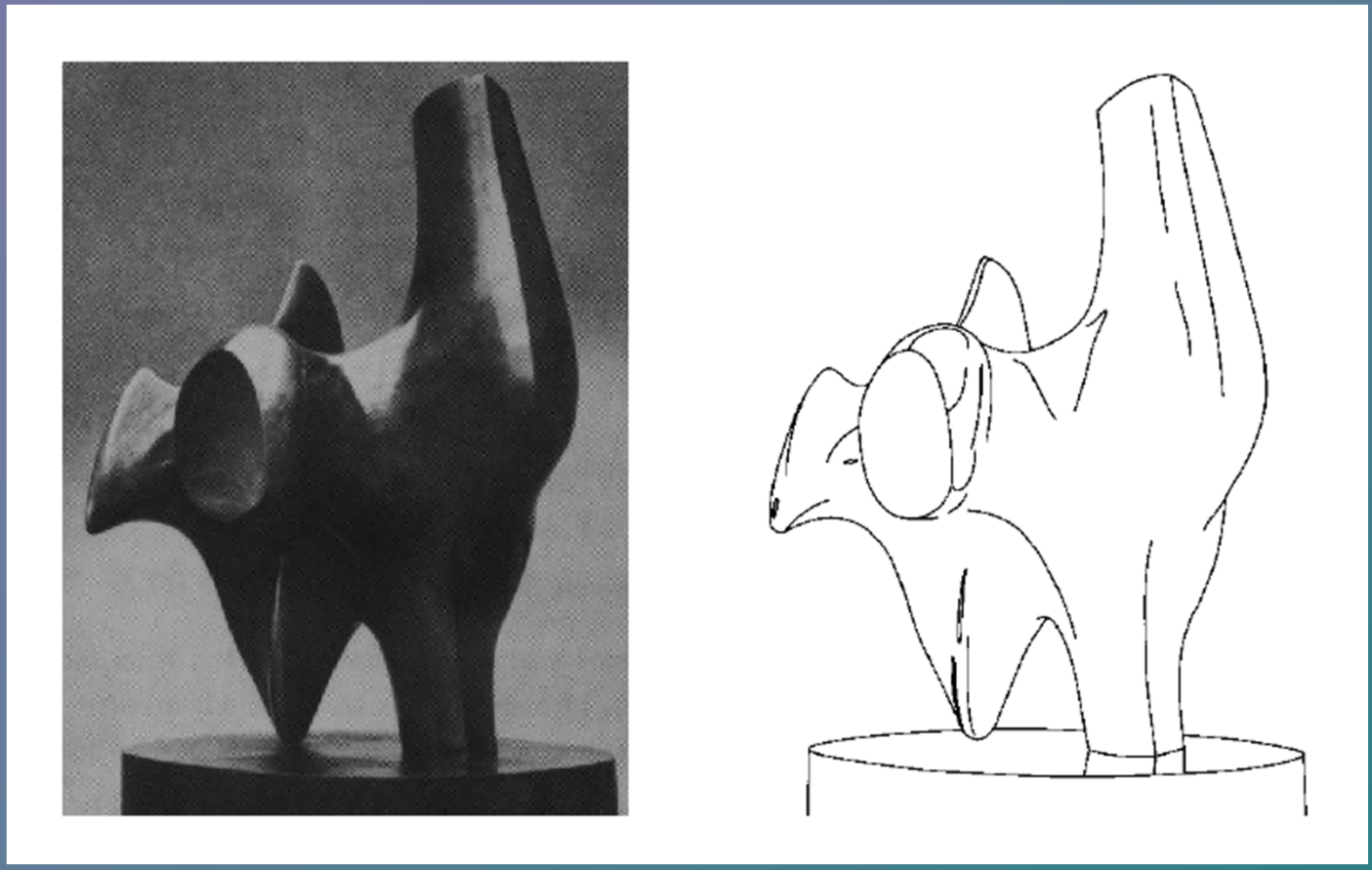
Pout
after histogram equalization

Histogram Equalization Example



It is commonly used for light normalization for image comparison or pattern recognition under different lighting conditions.

Edge detection



- Convert a 2D image into a set of curves
  - Extracts salient features of the scene
  - More compact than pixels

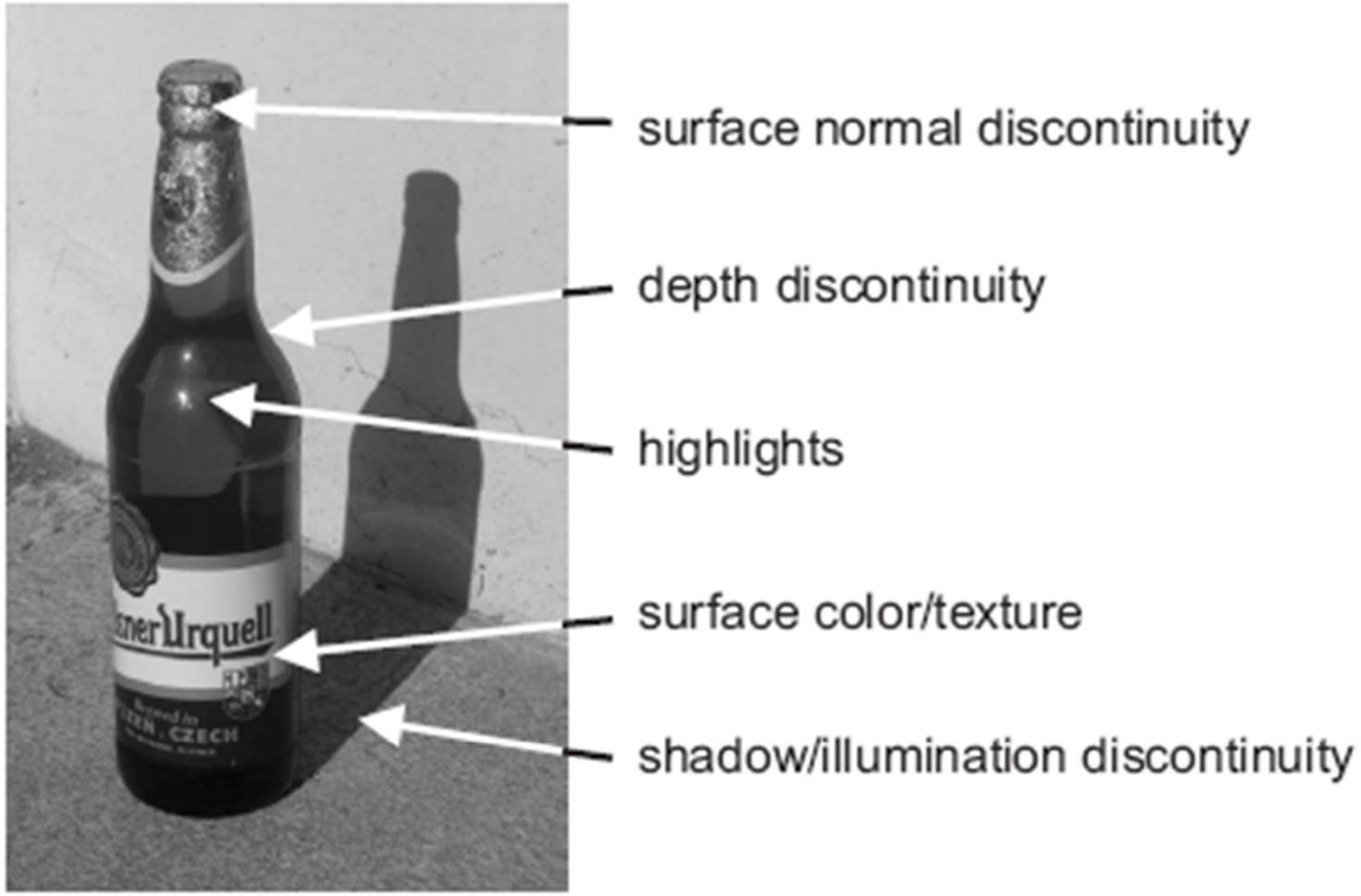


Figure 5.16: Origin of edges, i.e., physical phenomena in the image formation process which lead to edges in images.

Figure 5.17: Detected edge elements.

Edge Types

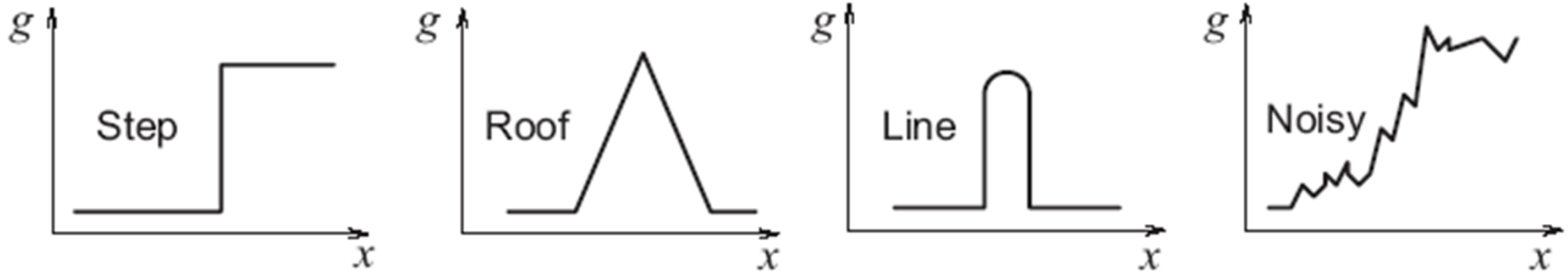


Figure 5.19: Typical edge profiles.

Image gradient

- The gradient of an image:

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

- The gradient points in the direction of most rapid change in intensity

- The gradient direction is given by:

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

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

- The *edge strength* is given by the gradient magnitude

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

Gradient and edge directions

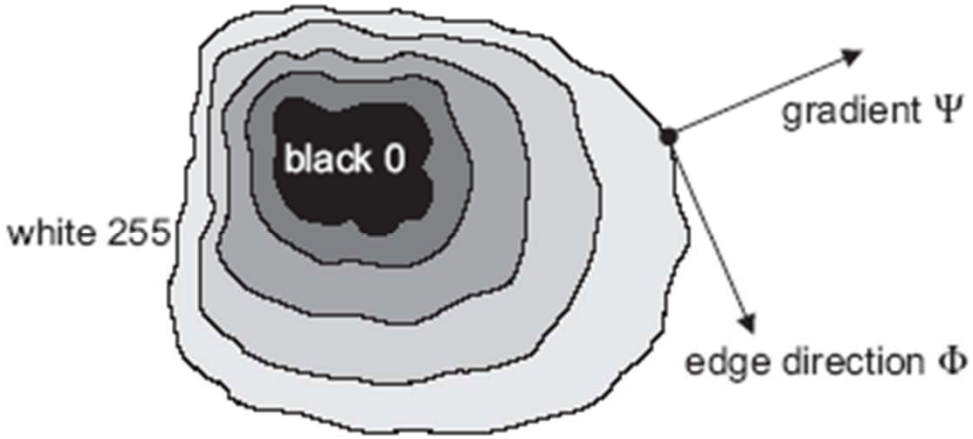
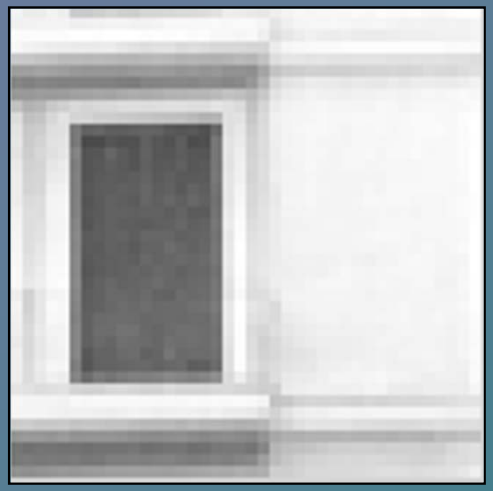
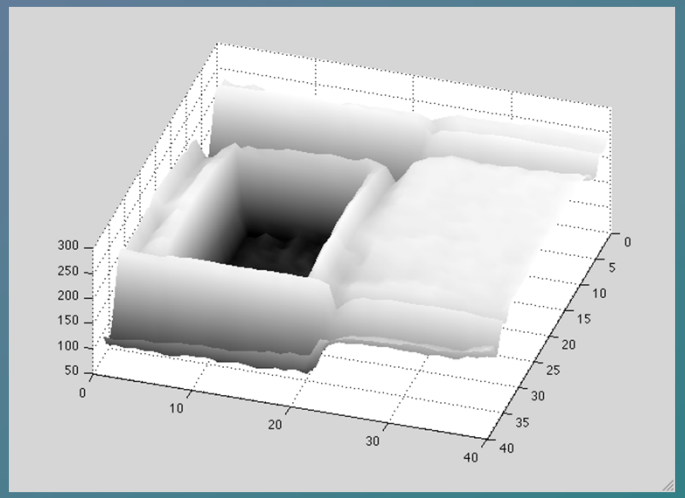
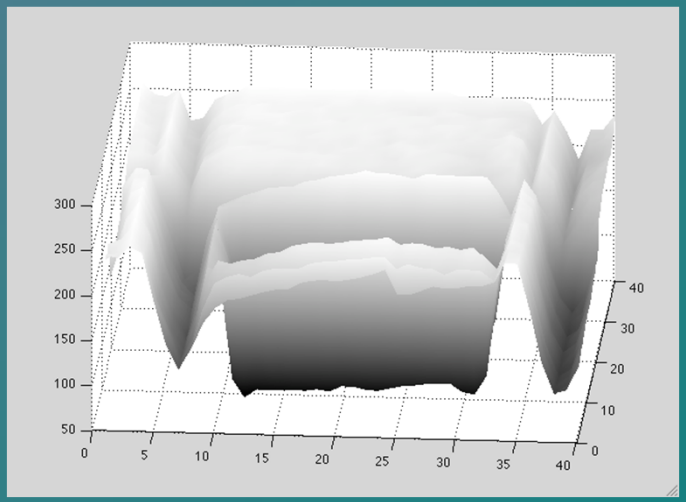


Figure 5.18: Gradient direction and edge direction.

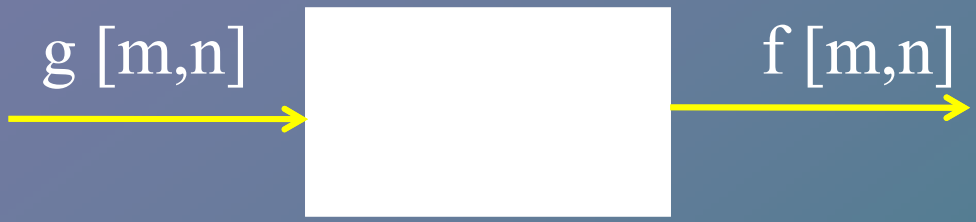
The image as a “surface”







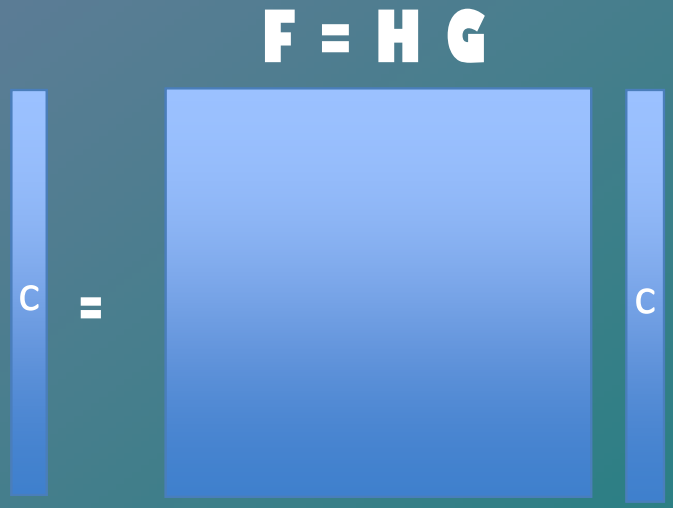
Linear filtering



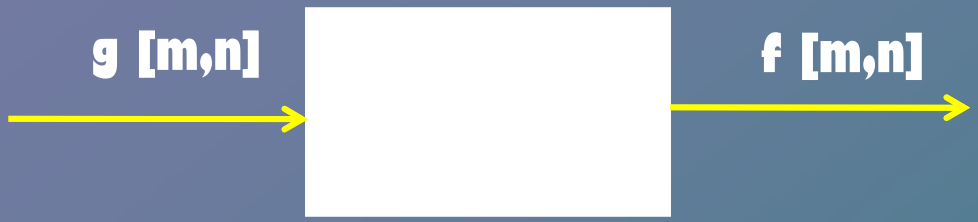
For a linear system, each output is a linear combination of all the input values:

$$f[m,n] = \sum_{k,l} h[m,n,k,l]\,g[k,l]$$

In matrix form:

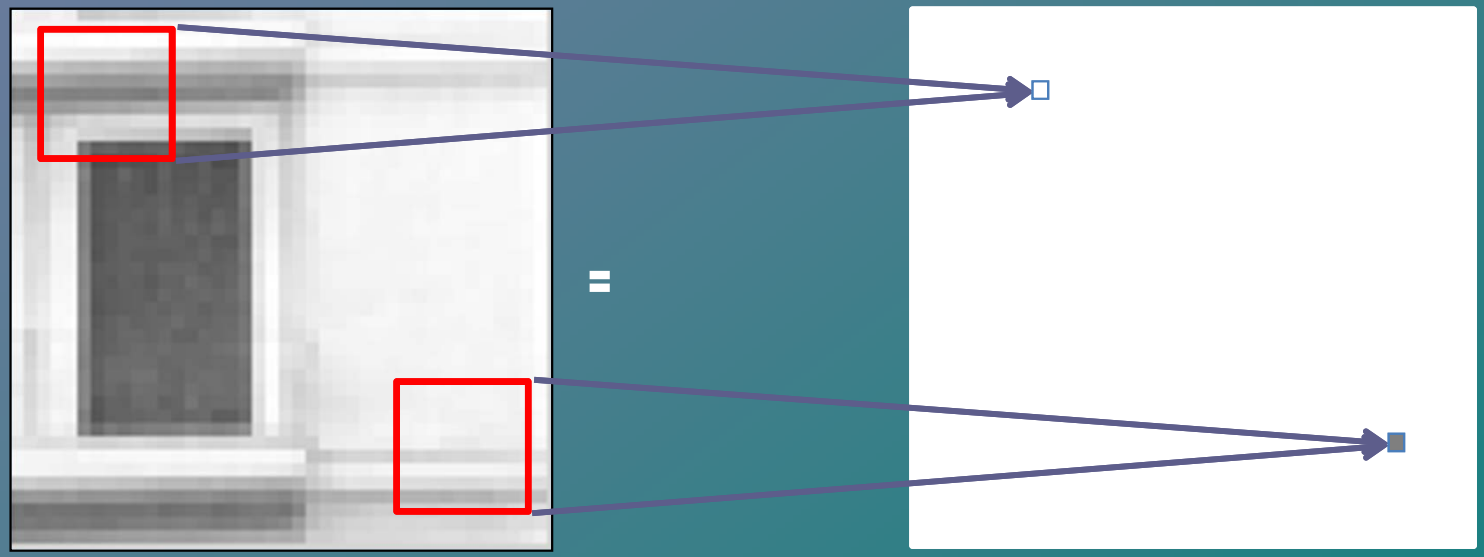


Linear filtering



In vision, many times, we are interested in operations that are spatially invariant. For a linear spatially invariant system:

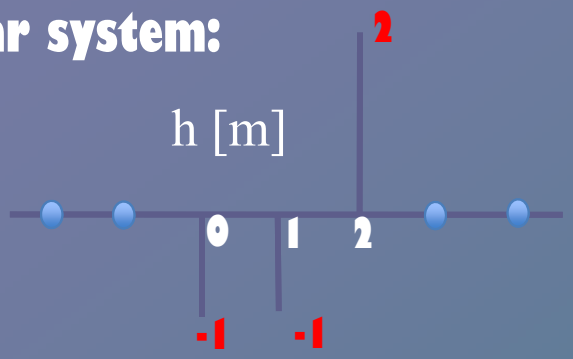
$$f[m,n] = I \otimes g = \sum_{k,l} h[m-k, n-l] g[k,l]$$



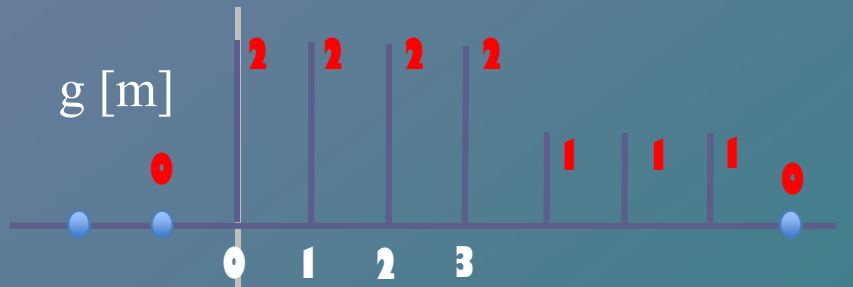
Linear filtering

$$f[m,n] = I \otimes g = \sum_{k,l} h[m-k, n-l] g[k,l]$$

Linear system:

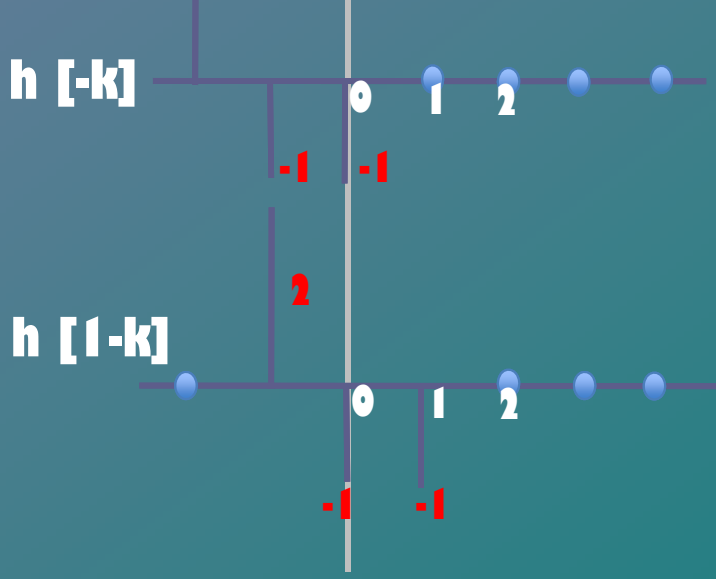


Input:



Output?

$$f[m=0] = \sum_k h[-k] g[k]$$



f [m=0]=-2

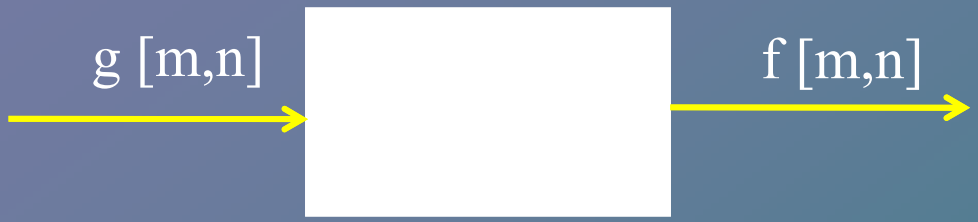
$$f[m=1] = \sum_k h[1-k] g[k]$$

f [m=1]=-4

$$f[m=2] = \sum_k h[2-k] g[k]$$

f [m=2]=0

Linear filtering



For a linear spatially invariant system

$$f[m,n] = I \otimes g = \sum_{k,l} h[m-k, n-l] g[k,l]$$

m=0 1 2 …

111	115	113	111	112	111	112	111
135	138	137	139	145	146	149	147
163	168	188	196	206	202	206	207
180	184	206	219	202	200	195	193
189	193	214	216	104	79	83	77
191	201	217	220	103	59	60	68
195	205	216	222	113	68	69	83
199	203	223	228	108	68	71	77

g[m,n]

⊗

-1	2	-1
-1	2	-1
-1	2	-1

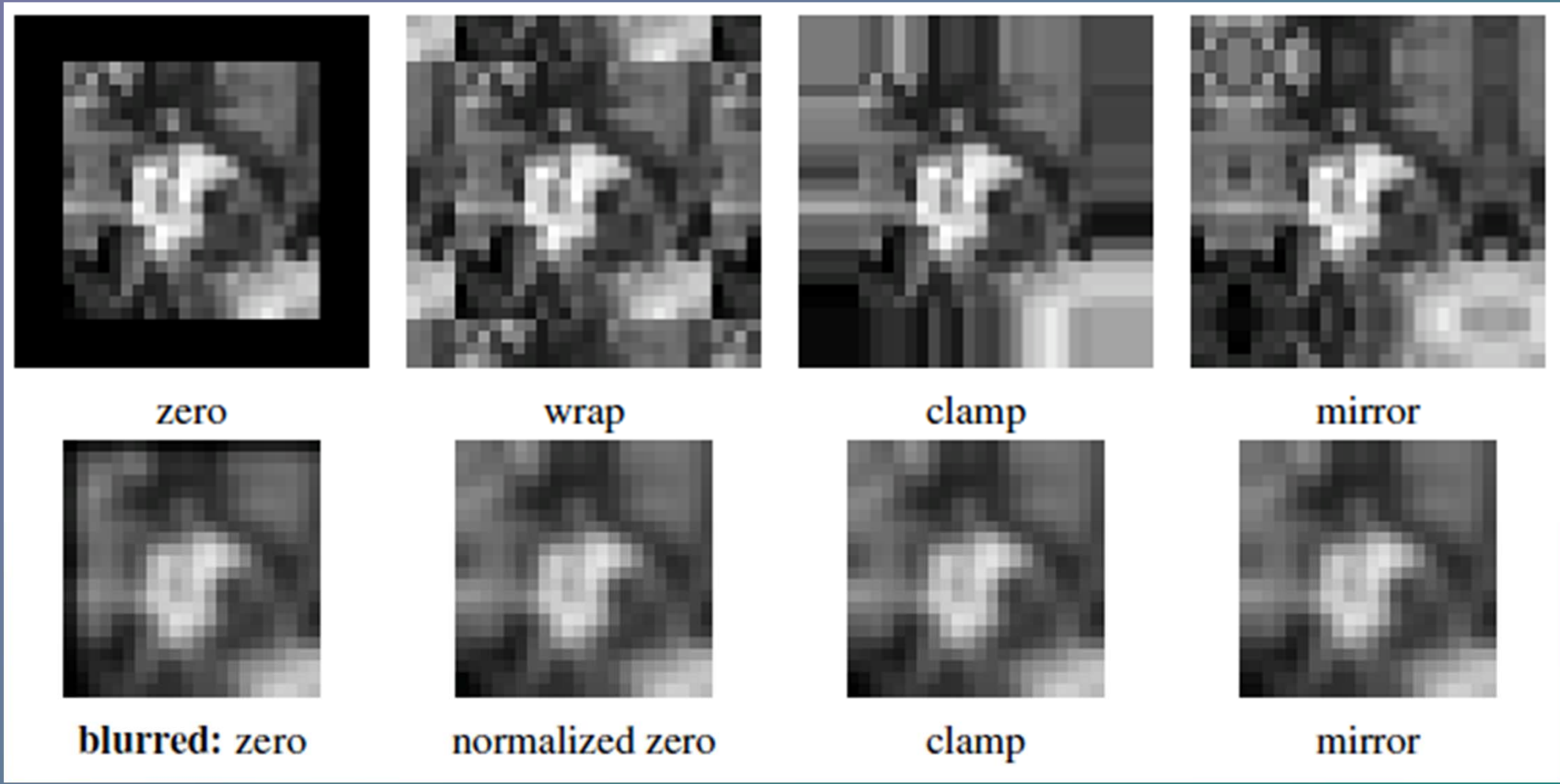
h[m,n]

=

?	?	?	?	?	?	?	?
?	-5	9	-9	21	-12	10	?
?	-29	18	24	4	-7	5	?
?	-50	40	142	-88	-34	10	?
?	-41	41		-175	-71	0	?
?	-24	37		-224	-120	-10	?
?	-23	33		-217	-134	-23	?
?	?	?	?	?	?	?	?

f[m,n]

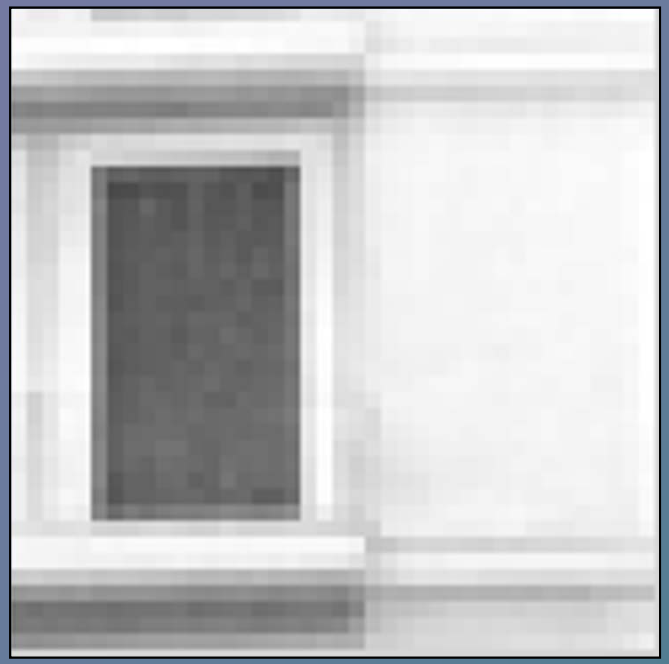
Borders



From Rick's book

Impulse

$$f[m,n] = I \otimes g = \sum_{k,l} h[m-k, n-l] g[k,l]$$



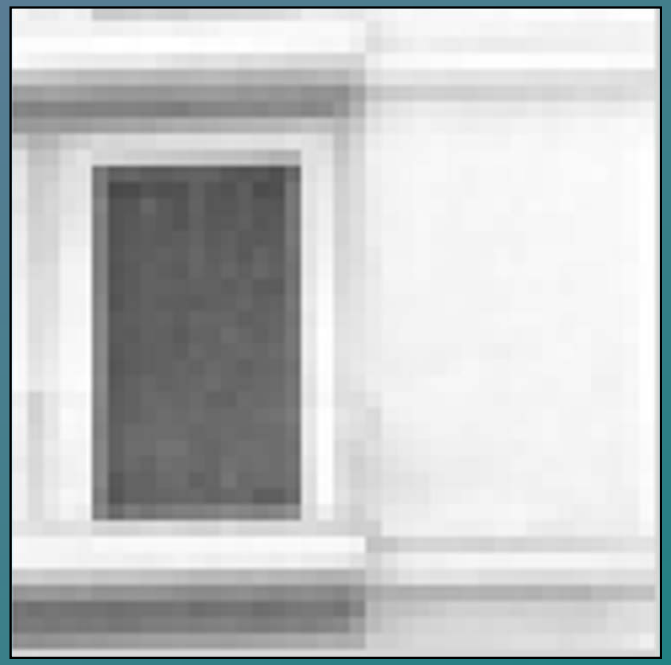
g[m,n]

$\otimes$

0	0	0	0	0
0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0

h[m,n]

=

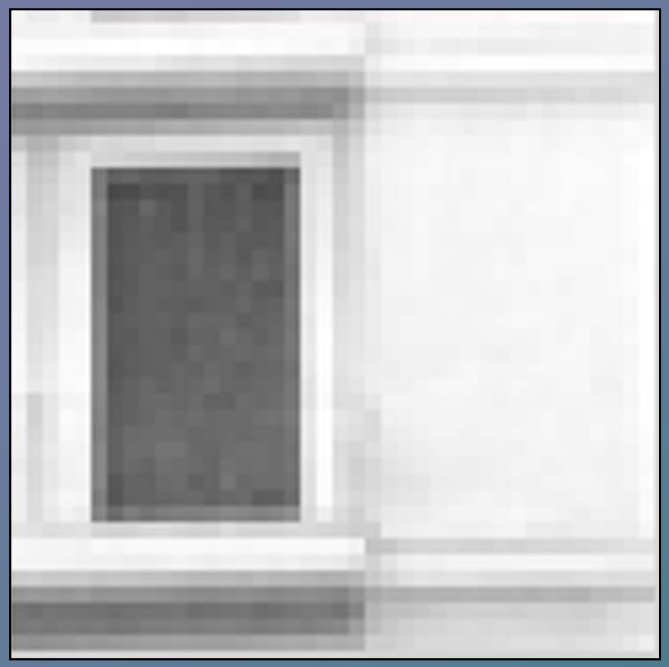


f[m,n]

Shifts

$$f[m,n] = I \otimes g = \sum_{k,l} h[m-k, n-l] g[k,l]$$

2 pixels →



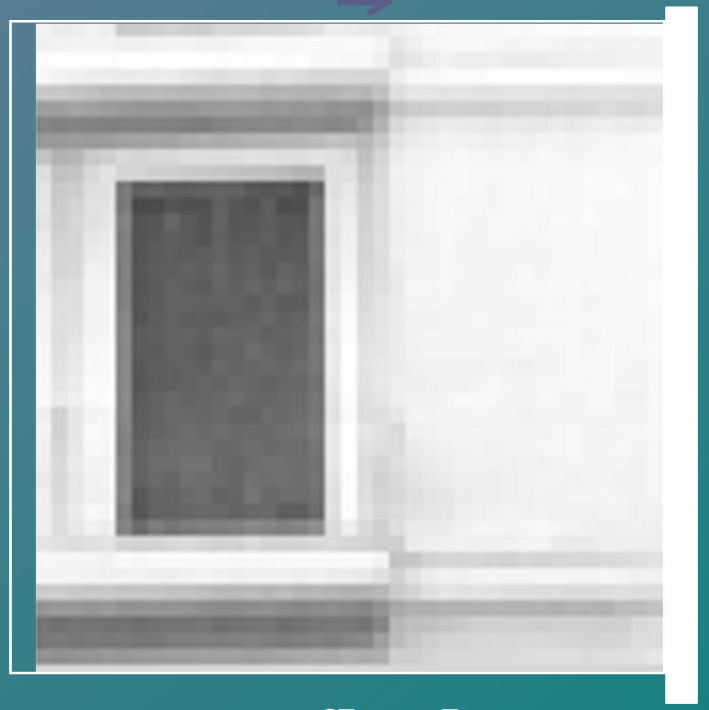
g[m,n]

$\otimes$

0	0	0	0	0
0	0	0	0	0
0	0	0	0	1
0	0	0	0	0
0	0	0	0	0

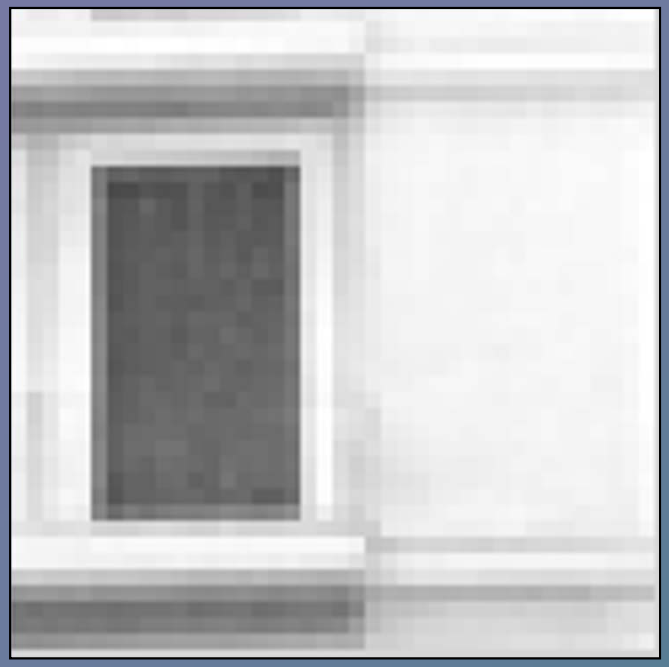
h[m,n]

=



f[m,n]

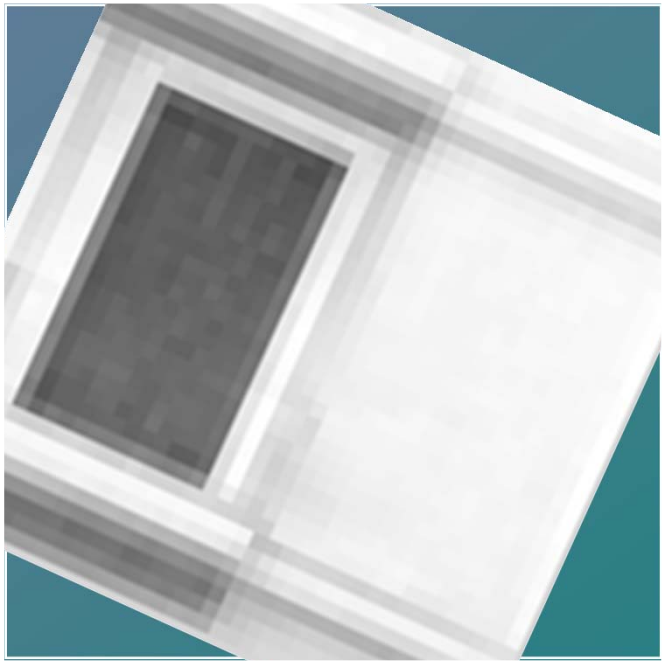
Image rotation



g[m,n]

$\otimes$? =

h[m,n]



It is linear, but not a spatially invariant operation. There is not convolution.

Rectangular filter

g[m,n]

⊗

h[m,n]

=

f[m,n]

Rectangular filter

$\otimes$

$=$

g[m,n]

h[m,n]

f[m,n]

Rectangular filter

g[m,n]

⊗

h[m,n]

=

f[m,n]

Differentiation and Convolution

- Recall

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

- We could approximate this as

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

- Now this is linear and shift invariant, so must be the result of a convolution.

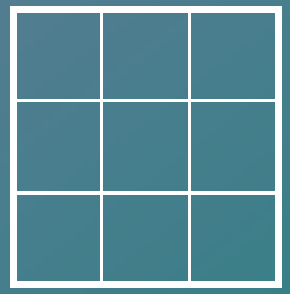
- (which is obviously a convolution; it's not a very good way to do things, as we shall see)

The discrete gradient

- How can we differentiate a *digital* image F(x,y)?
  - Option 1: reconstruct a continuous image, then take gradient
  - Option 2: take discrete derivative (finite difference)

$$\frac{\partial F}{\partial x}[x, y] \approx F[x+1, y] - F[x, y]$$

- How would you implement this as a cross-correlation?



H

filter demo

The Sobel Operator

- Better approximations of the derivatives exist
  - The *Sobel* operators below are commonly used

$\frac{1}{8}$

-1	0	1
-2	0	2
-1	0	1

H_x

$\frac{1}{8}$

1	2	1
0	0	0
-1	-2	-1

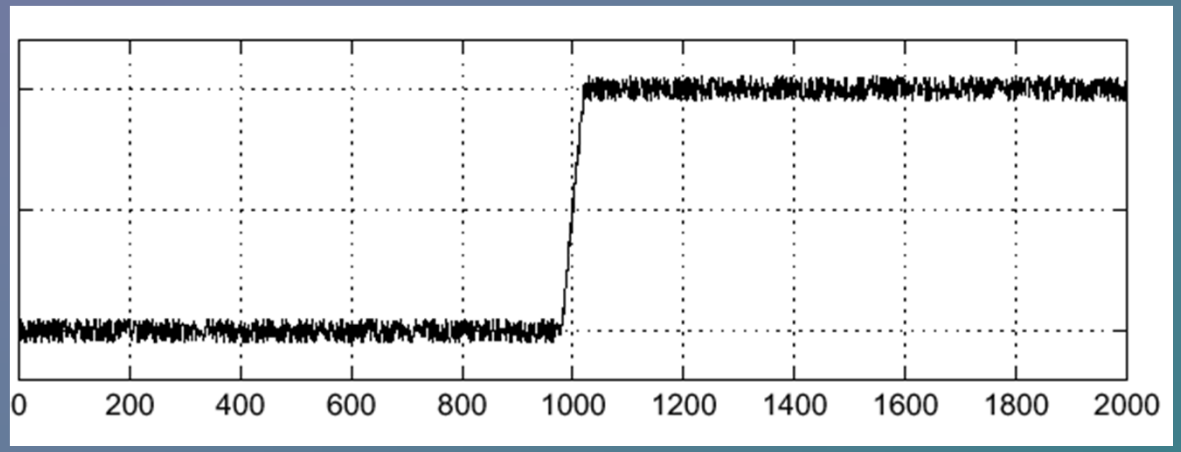
H_y

- The standard defn. of the Sobel operator omits the 1/8 term
  - doesn't make a difference for edge detection
  - the 1/8 term **is** needed to get the right gradient value, however

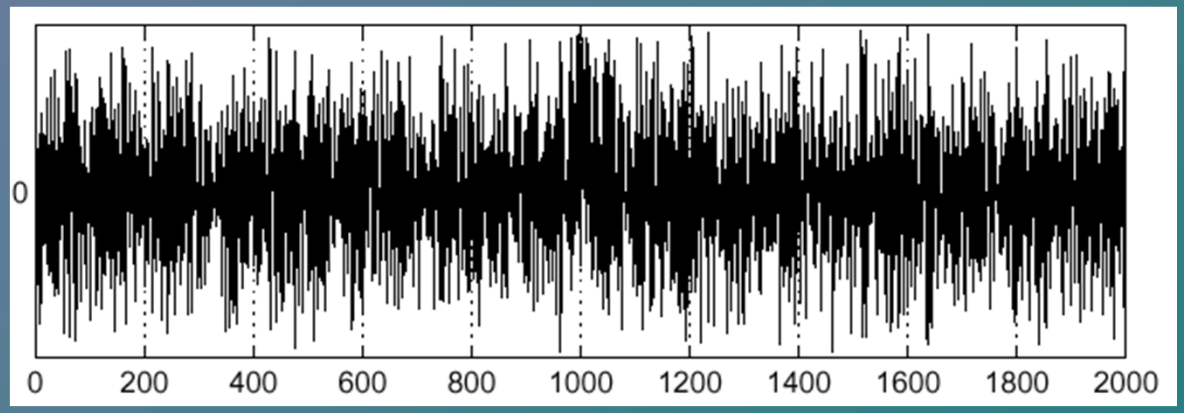
Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a *signal*

$f(x)$



$\frac{d}{dx} f(x)$



- Where is the edge?

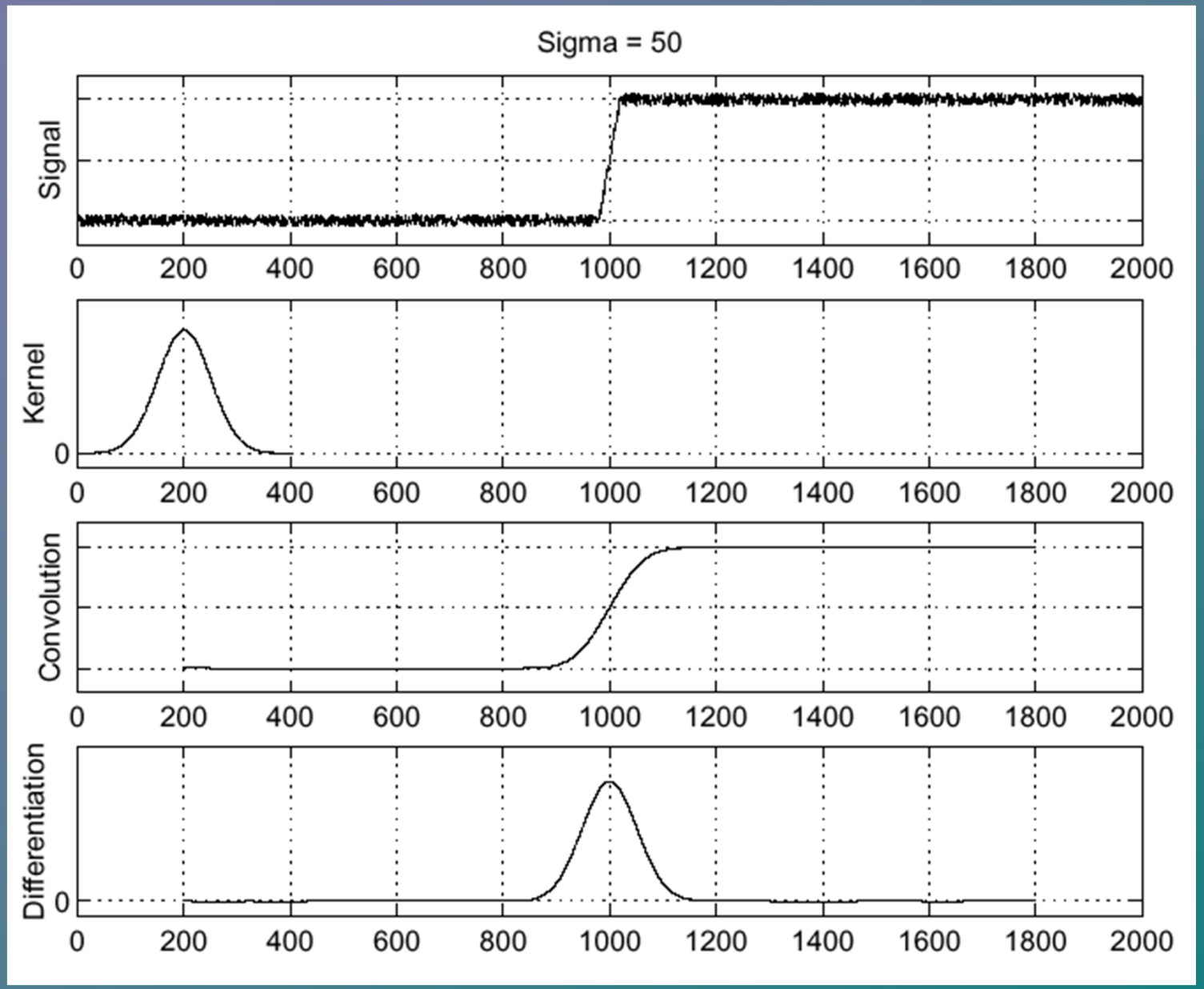
Solution: smooth first

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



- Where is the edge? Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Derivative theorem of convolution

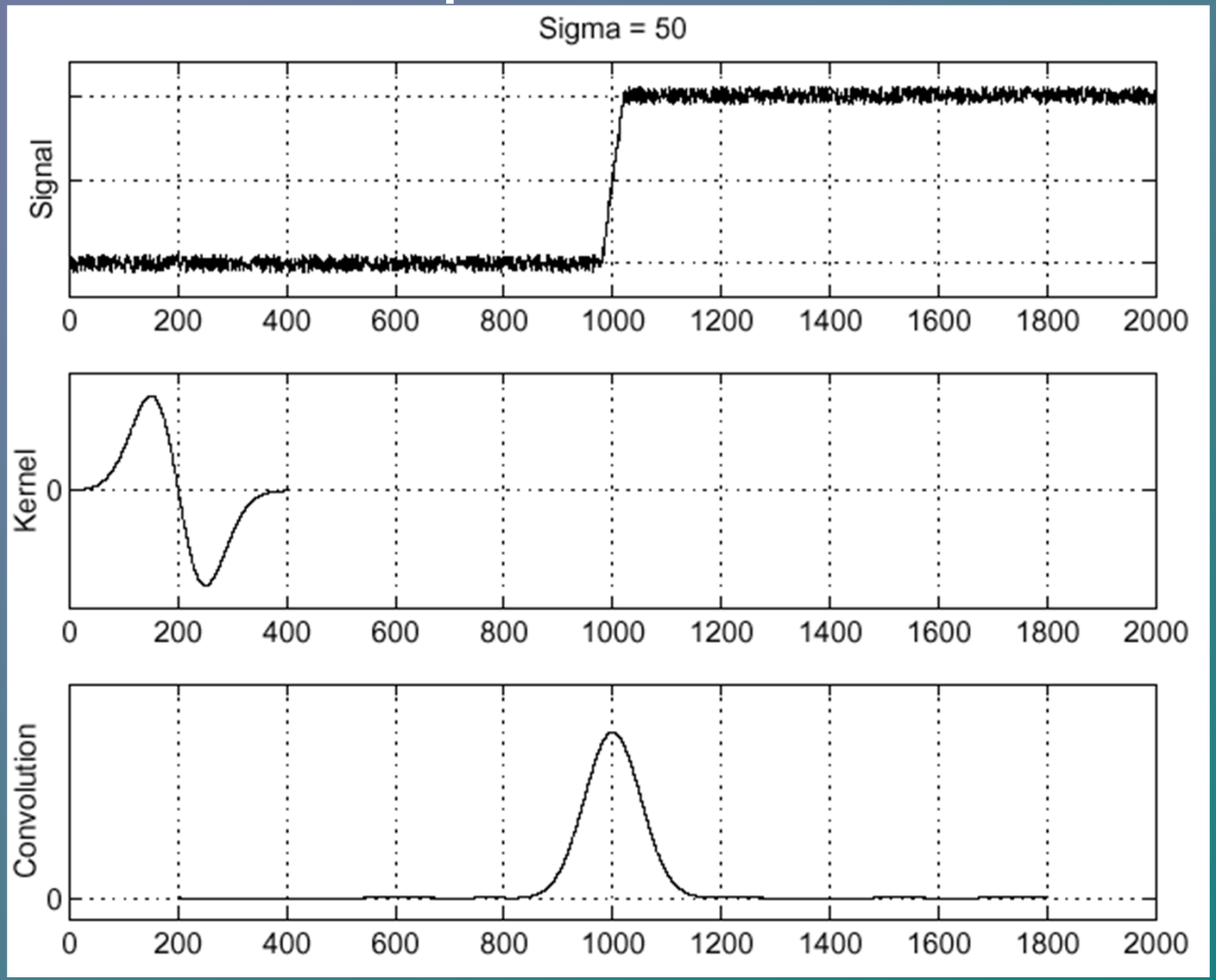
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

- This saves us one operation:

f

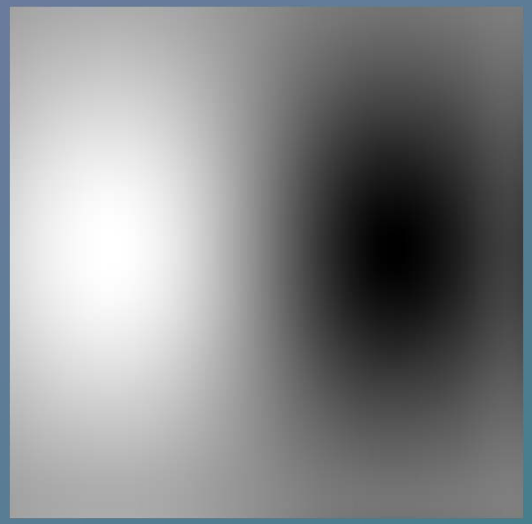
$\frac{\partial}{\partial x}h$

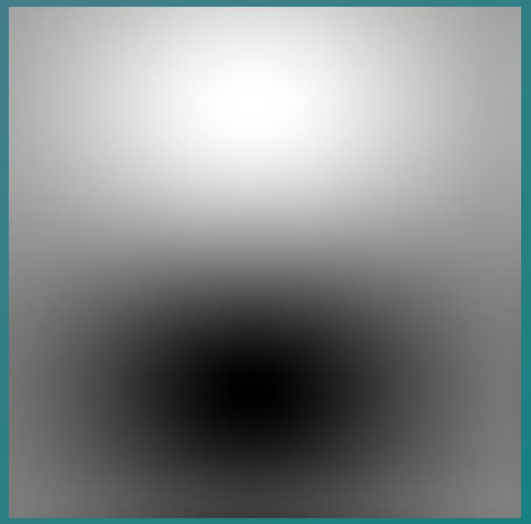
$(\frac{\partial}{\partial x}h) \star f$



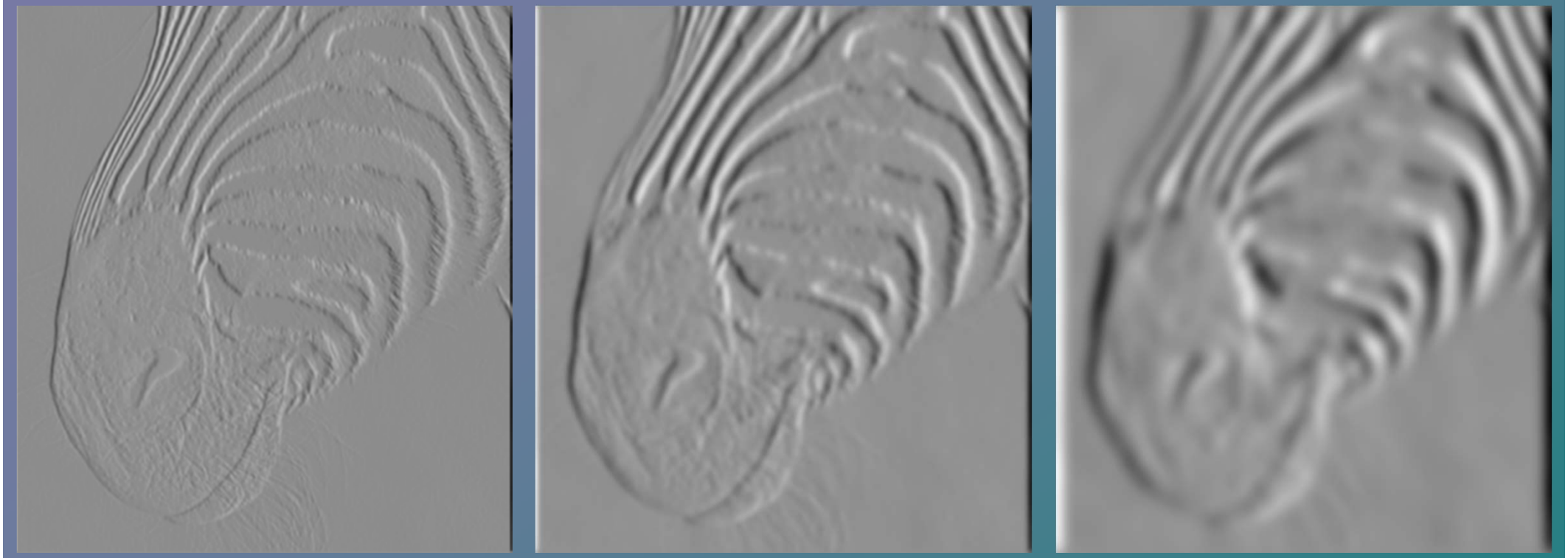
Smoothing and Differentiation

- Issue: noise
  - smooth before differentiation
  - two convolutions to smooth, then differentiate?
  - actually, no - we can use a derivative of Gaussian filter
    - because differentiation is convolution, and convolution is associative





Edge Detection at Different Scales



σ =1 pixel σ = 3 pixels σ = 7 pixels

The scale of the smoothing filter affects derivative estimates, and also the semantics of the edges recovered.

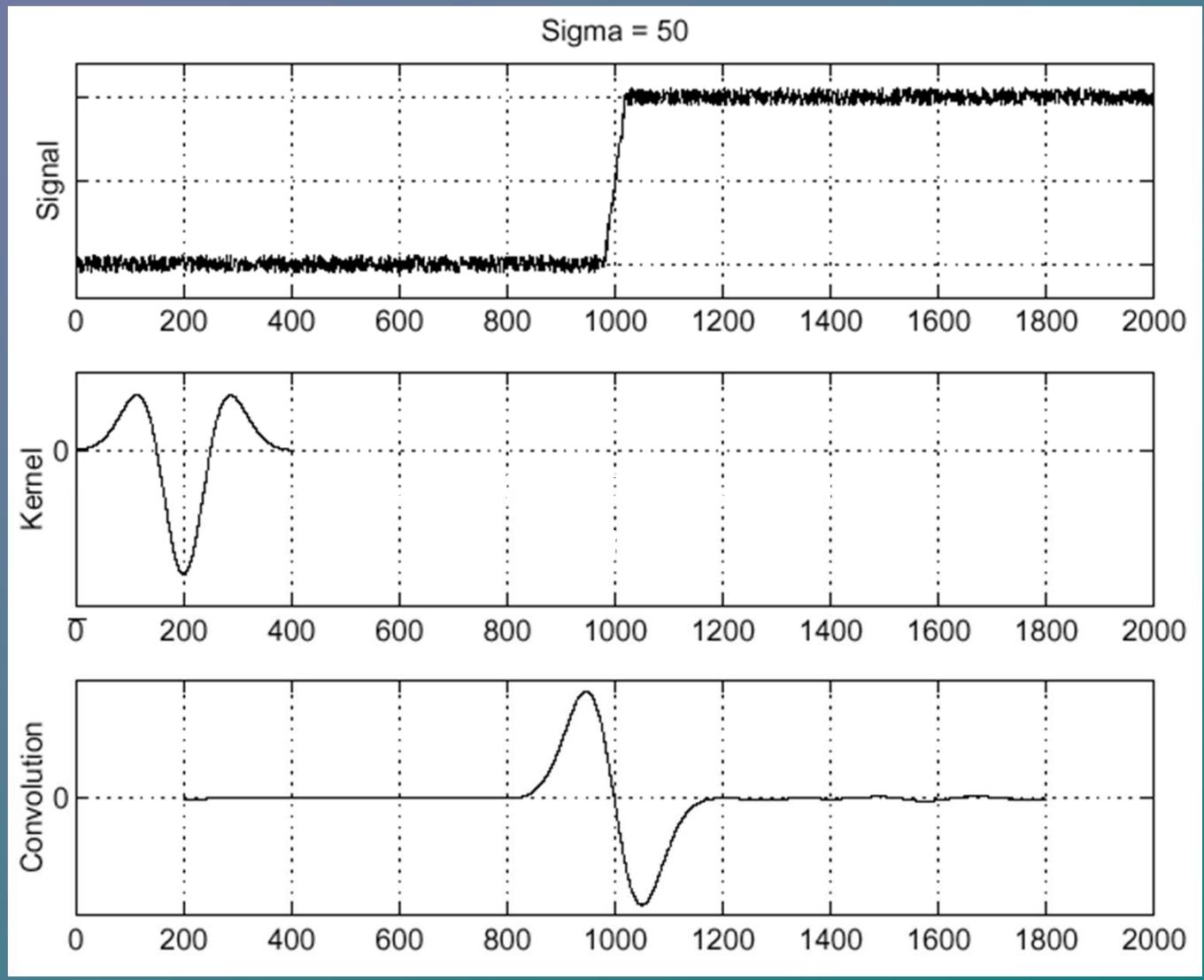
Laplacian of Gaussian

- Look for zero-crossings of $\frac{\partial^2}{\partial x^2}(h \star f)$

f

$\frac{\partial^2}{\partial x^2} h$

$(\frac{\partial^2}{\partial x^2} h) \star f$



1D Example in LoG Result

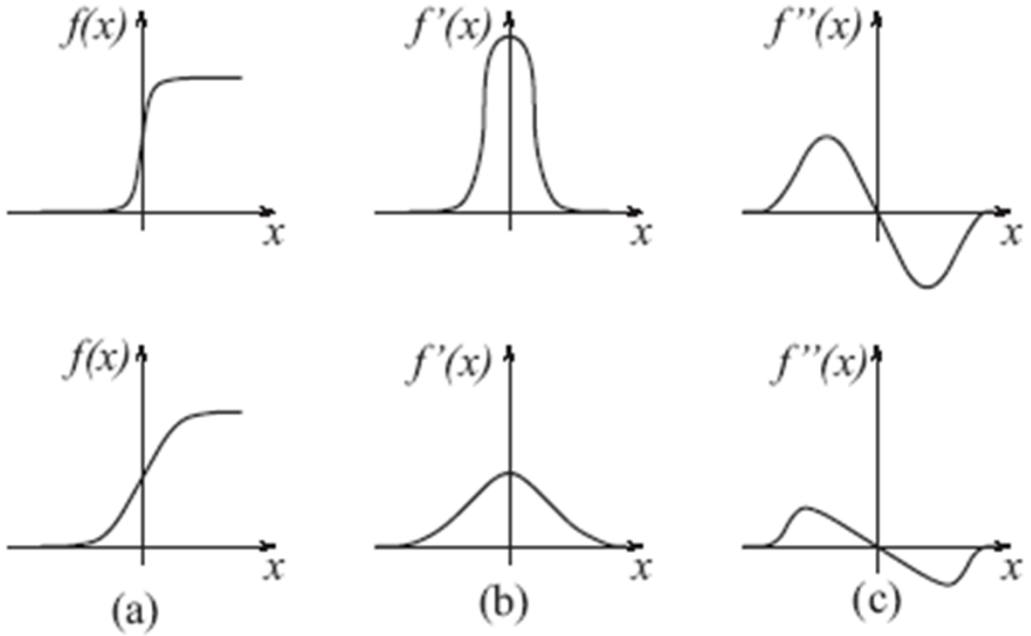
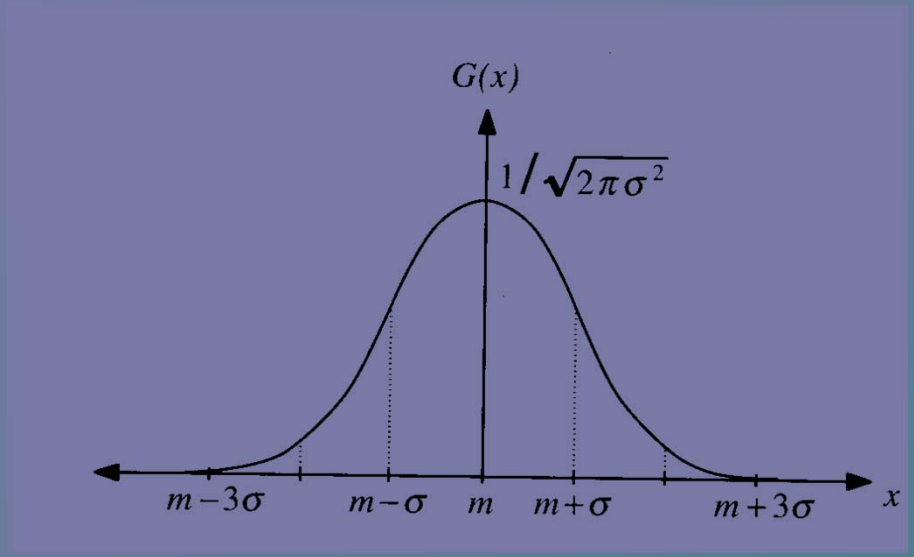


Figure 5.22: 1D edge profile of the zero-crossing.

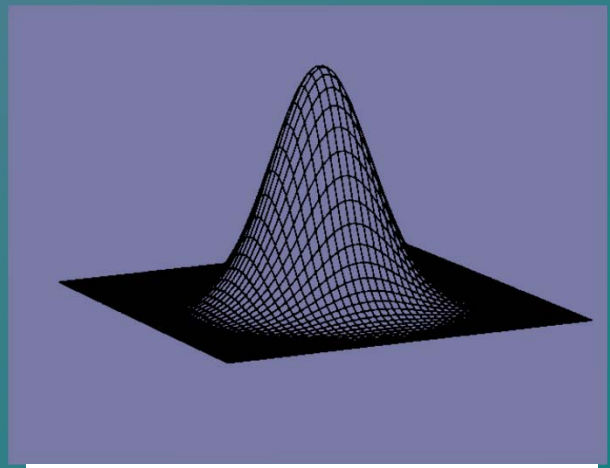
Gaussian Functions in 1-D and 2-D



$$G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-m)^2}{2\sigma^2}\right)$$

Gaussians are Separable

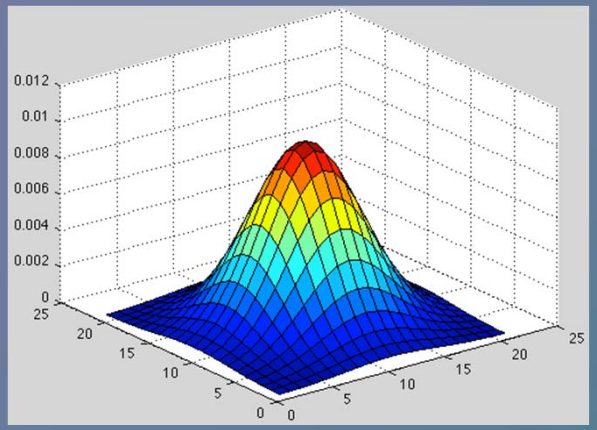
$$\begin{aligned} G_\sigma(x,y) &= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right) \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x^2)}{2\sigma^2}\right)\right) \times \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^2)}{2\sigma^2}\right)\right) \end{aligned}$$

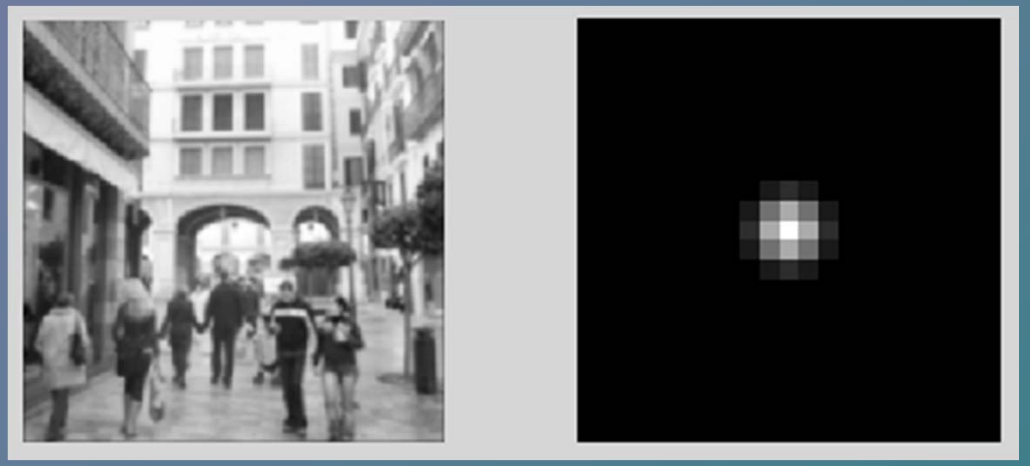


$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

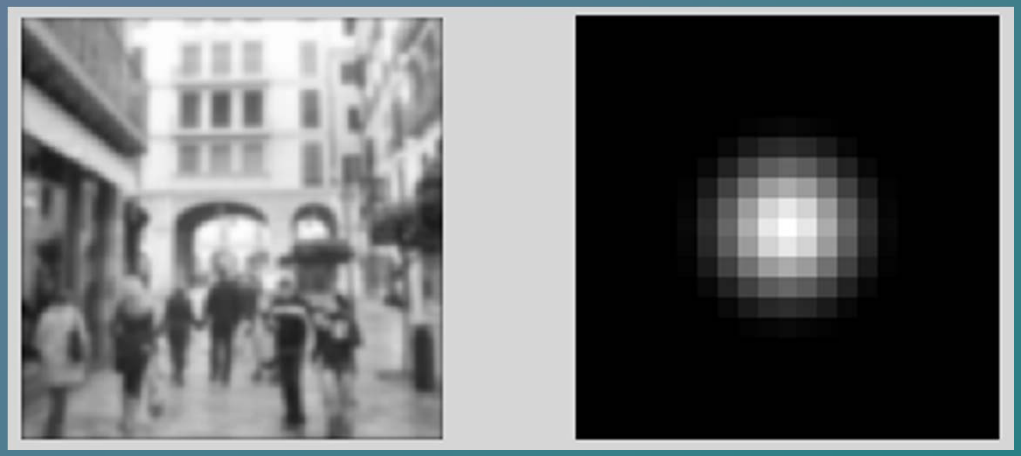
Gaussian filtering

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

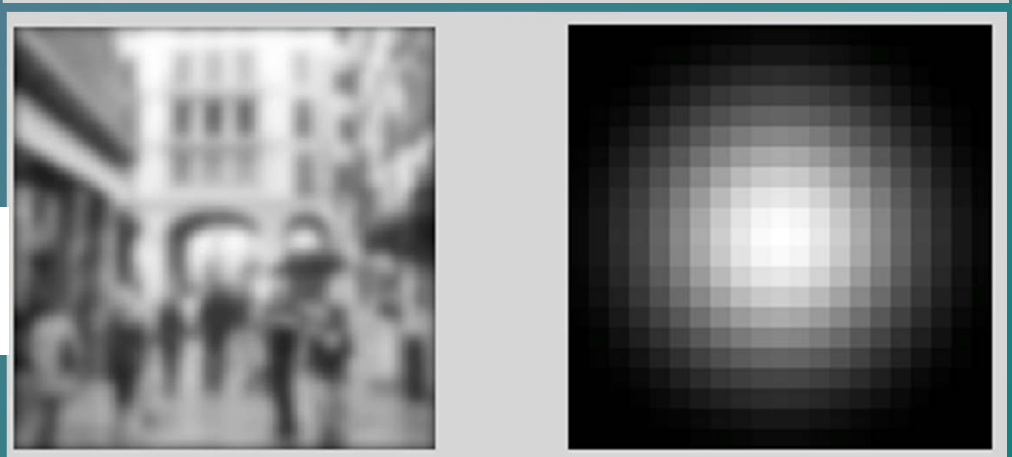




σ=1



σ=2



σ=4

$$f[m,n] = I \otimes G = \sum_{k,l} I[m-k, n-l] G[k,l]$$

SECOND ORDER EDGE OPERATOR

MARR-HILDRETH OPERATOR

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

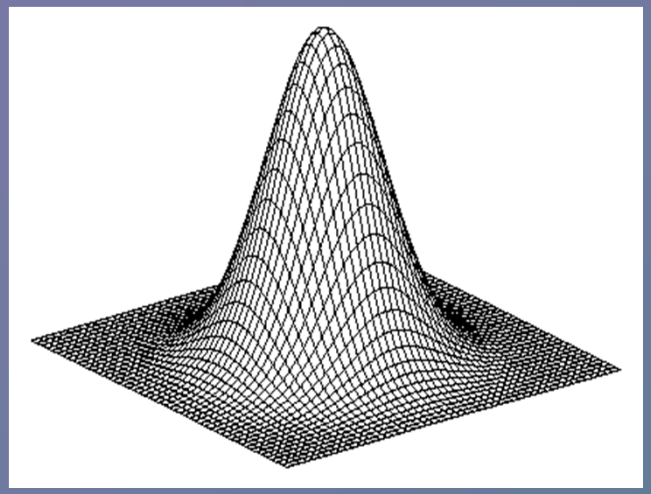
0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

Combine this with Gaussian smoothing.

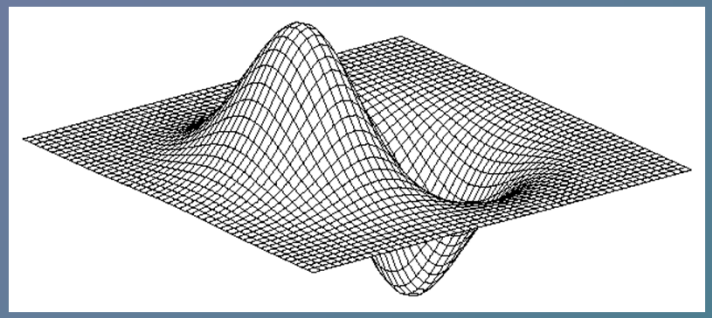
Detect edge at 'Zero-Crossing' looking for positive and negative peaks on either side.

2D edge detection filters



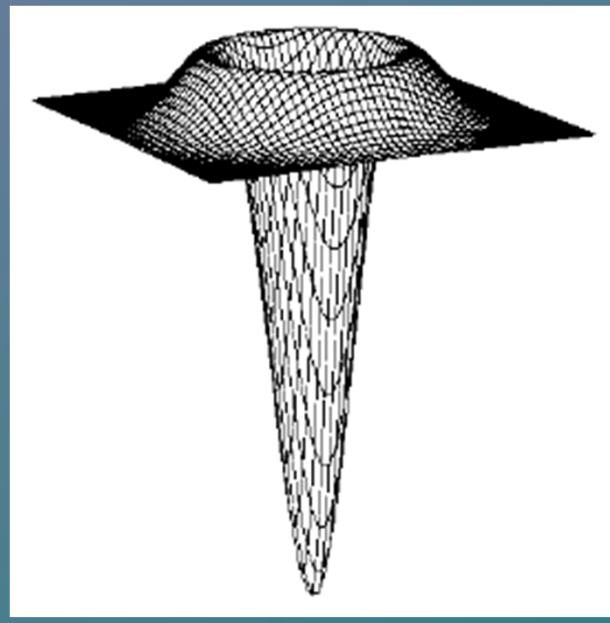
Gaussian

$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

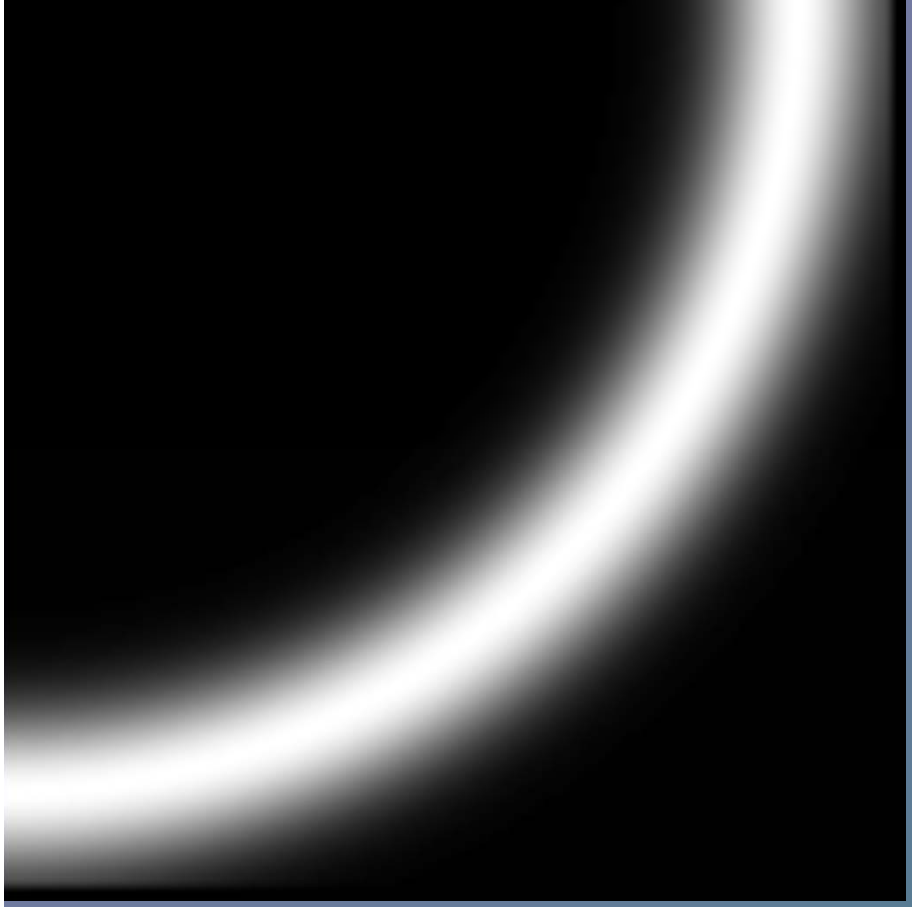


Laplacian of Gaussian

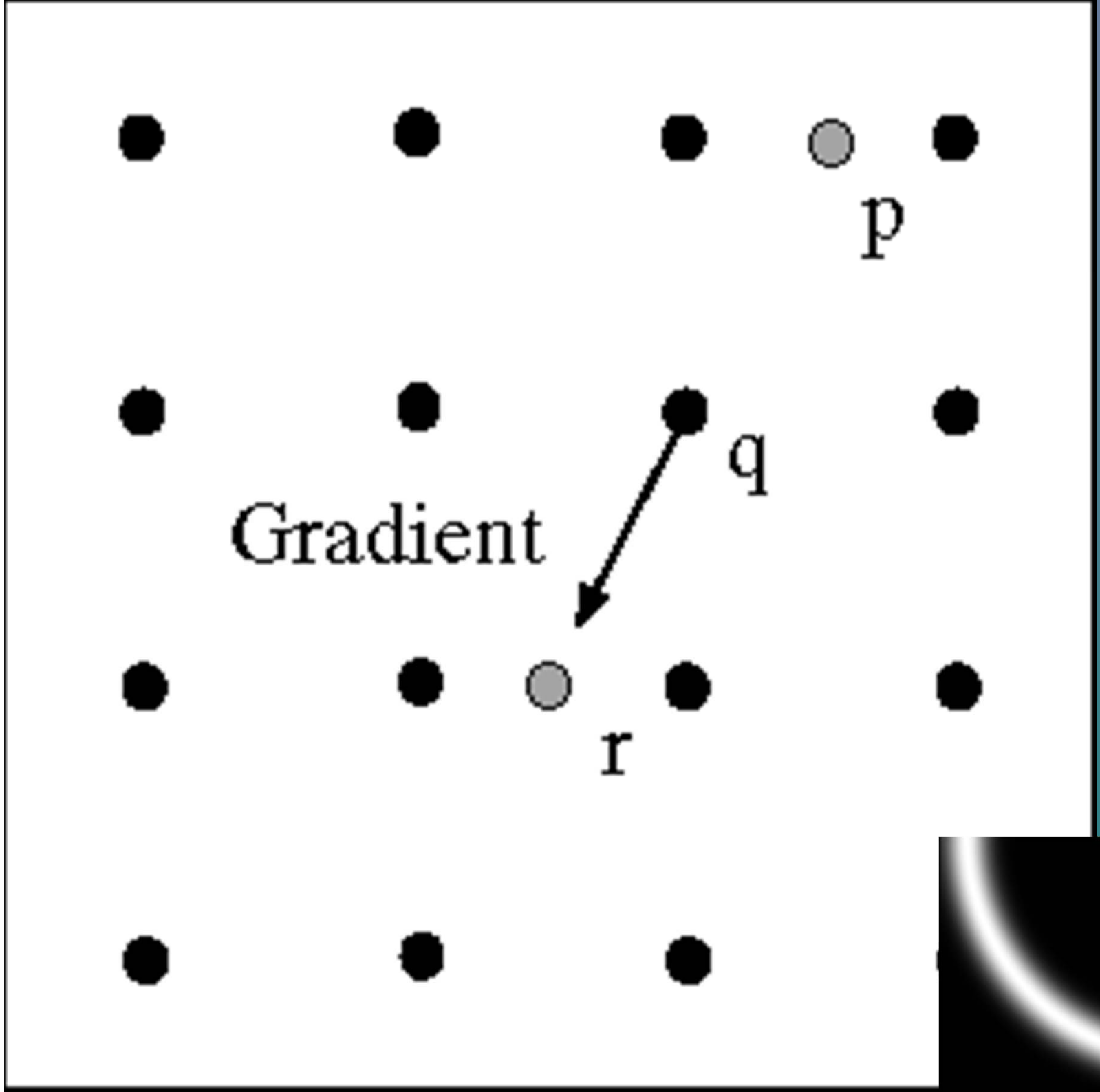
$$\nabla^2 h_\sigma(u,v)$$

- ∇^2 is the **Laplacian** operator:

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

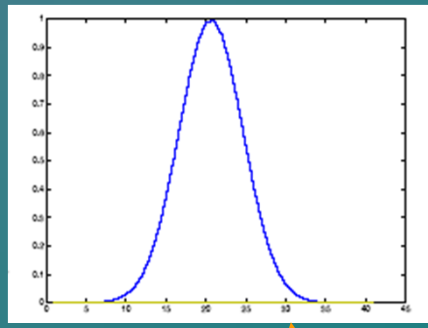


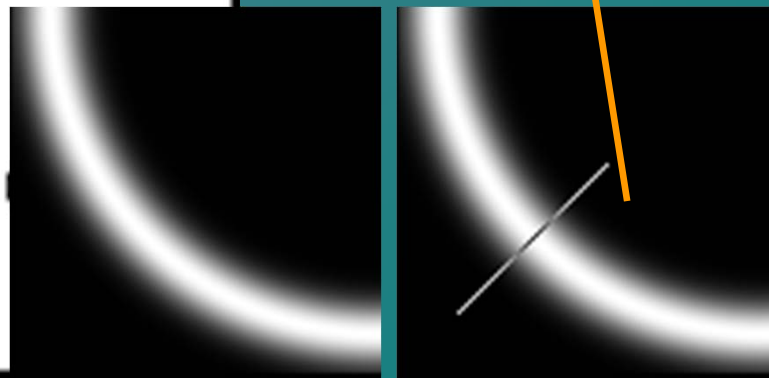
We wish to mark points along the curve where the magnitude is biggest. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?



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.



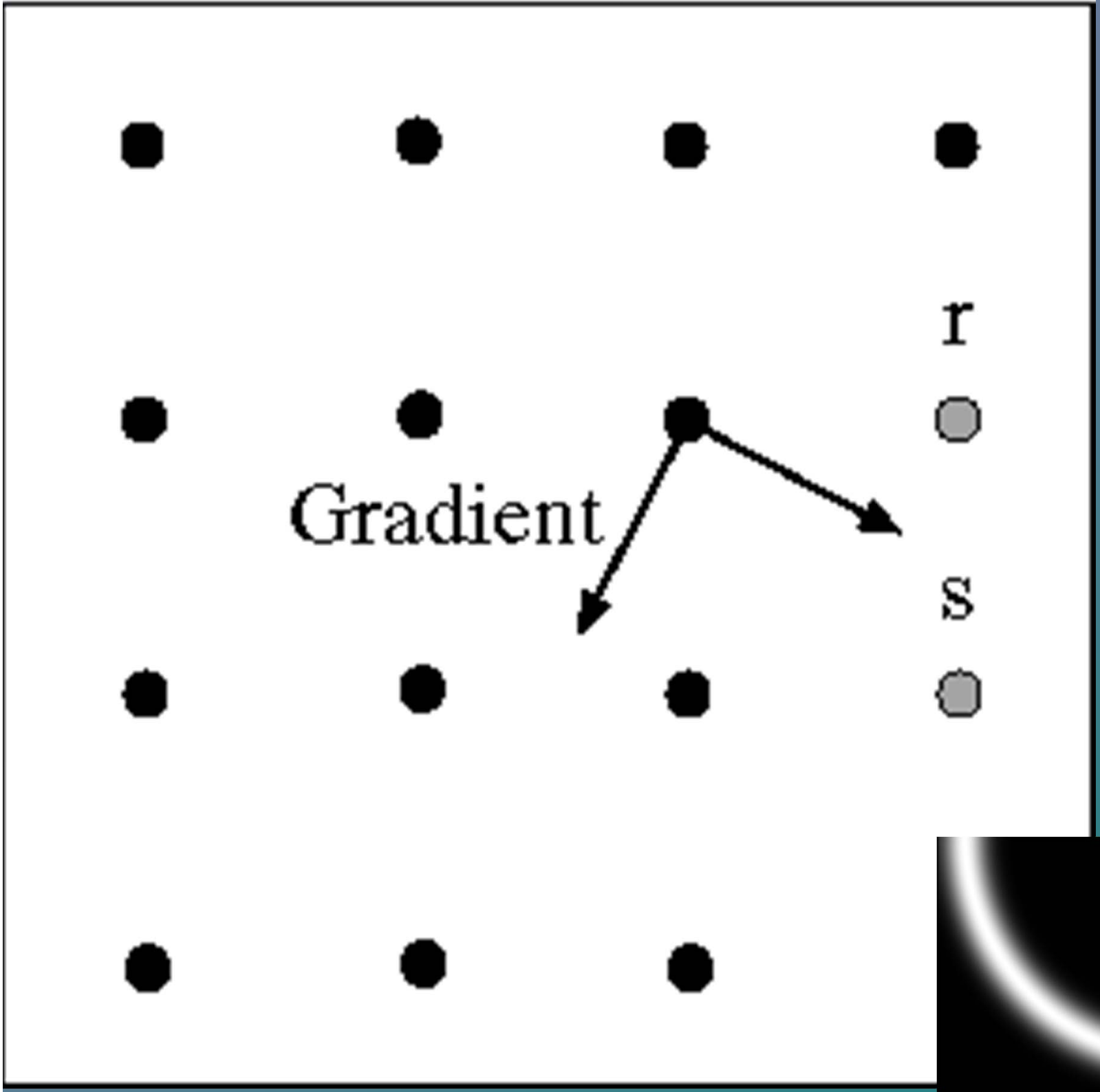


Non-Maximum Suppression

```
While there are points with high gradient
that have not been visited

  Find a start point that is a local maximum in the
    direction perpendicular to the gradient
    erasing points that have been checked

  while possible, expand a chain through
    the current point by:
      1) predicting a set of next points, using
        the direction perpendicular to the gradient

      2) finding which (if any) is a local maximum
        in the gradient direction

      3) testing if the gradient magnitude at the
        maximum is sufficiently large

      4) leaving a record that the point and
        neighbours have been visited

      record the next point, which becomes the current point

  end

end
```

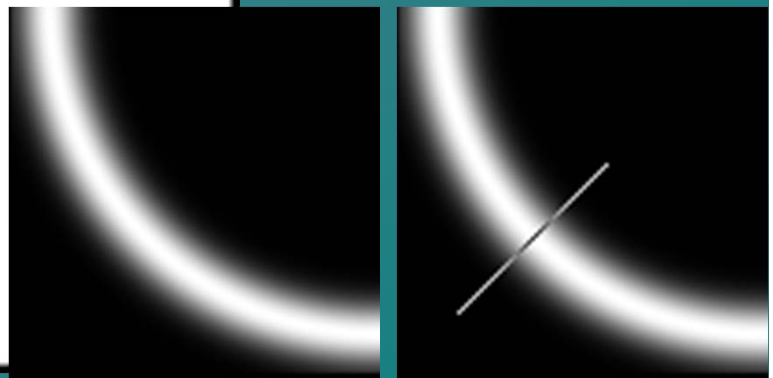
Simple Gradient-Based Edge Detection

```
form an estimate of the image gradient

obtain the gradient magnitude from this estimate

identify image points where the value
  of the gradient magnitude is maximal
  in the direction perpendicular to the edge
  and also large; these points are edge points
```



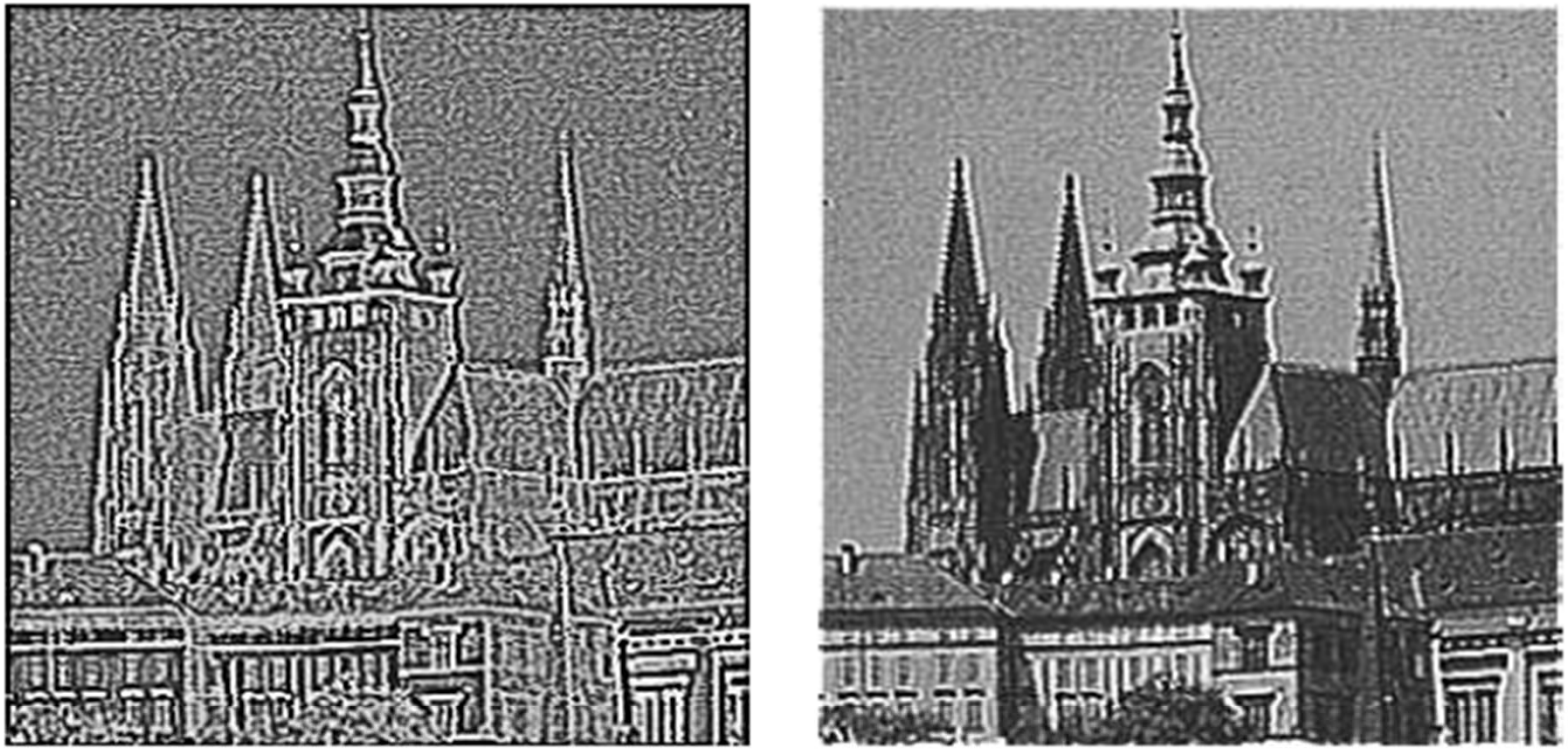
Predicting the next edge point

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).



Algorithm 5.4: Canny edge detector

1. Convolve an image f with a Gaussian of scale σ.
2. Estimate local edge normal directions $\mathbf{n}$ using equation (5.58) for each pixel in the image.
3. Find the location of the edges using equation (5.60) (non-maximal suppression).
4. Compute the magnitude of the edge using equation (5.61).
5. Threshold edges in the image with hysteresis (Algorithm 6.5) to eliminate spurious responses.
6. Repeat steps (1) through (5) for ascending values of the standard deviation σ.
7. Aggregate the final information about edges at multiple scale using the 'feature synthesis' approach.



(a) (b)

Figure 5.20: Laplace gradient operator. (a) Laplace edge image using the 8-connectivity mask. (b) Sharpening using the Laplace operator (equation (5.35), $C = 0.7$). Compare the sharpening effect with the original image in Figure 5.10a.

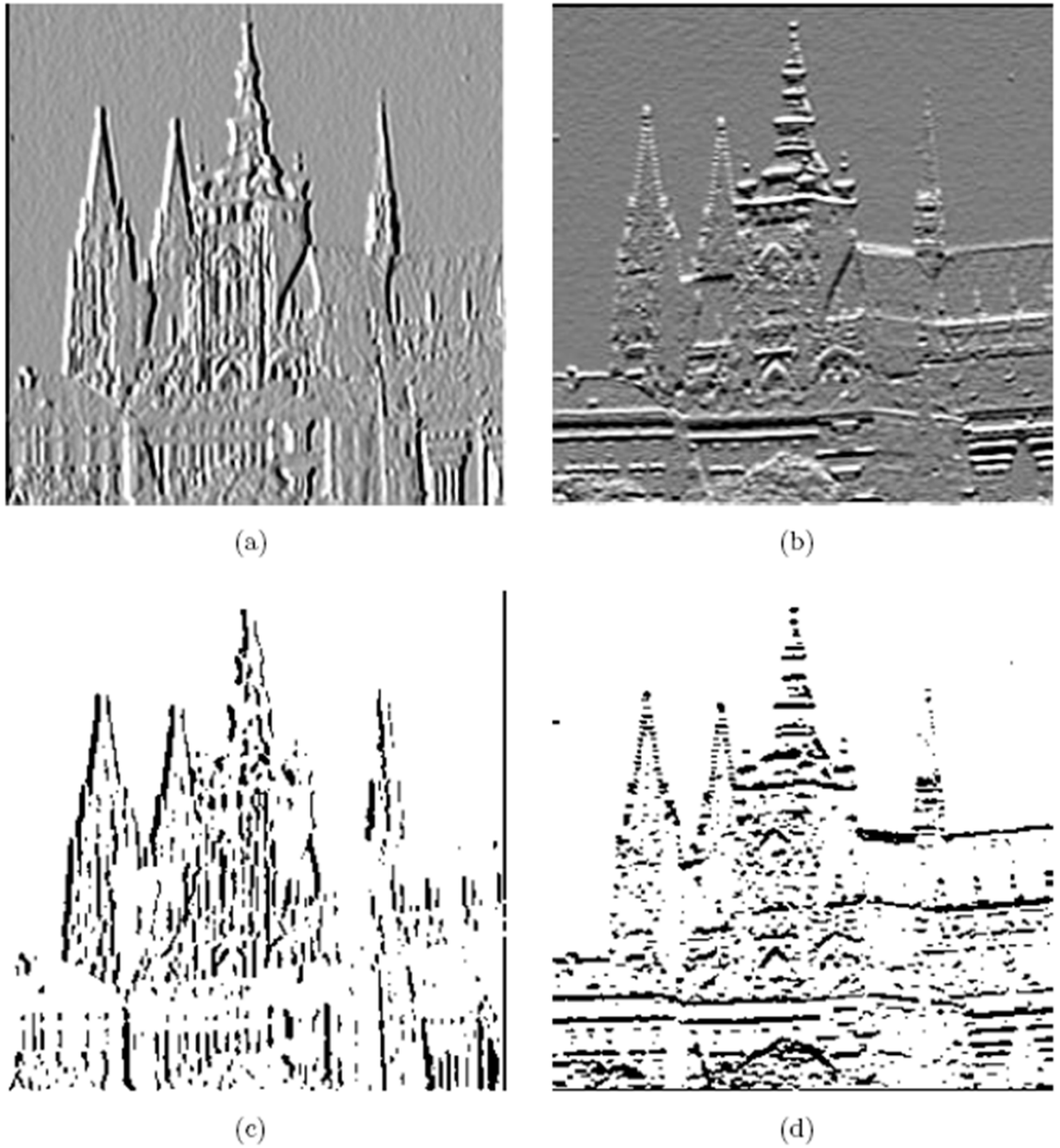


Figure 5.21: First-derivative edge detection using Prewitt compass operators. (a) North direction (the brighter the pixel value, the stronger the edge). (b) East direction. (c) Strong edges from (a). (d) Strong edges from (b).

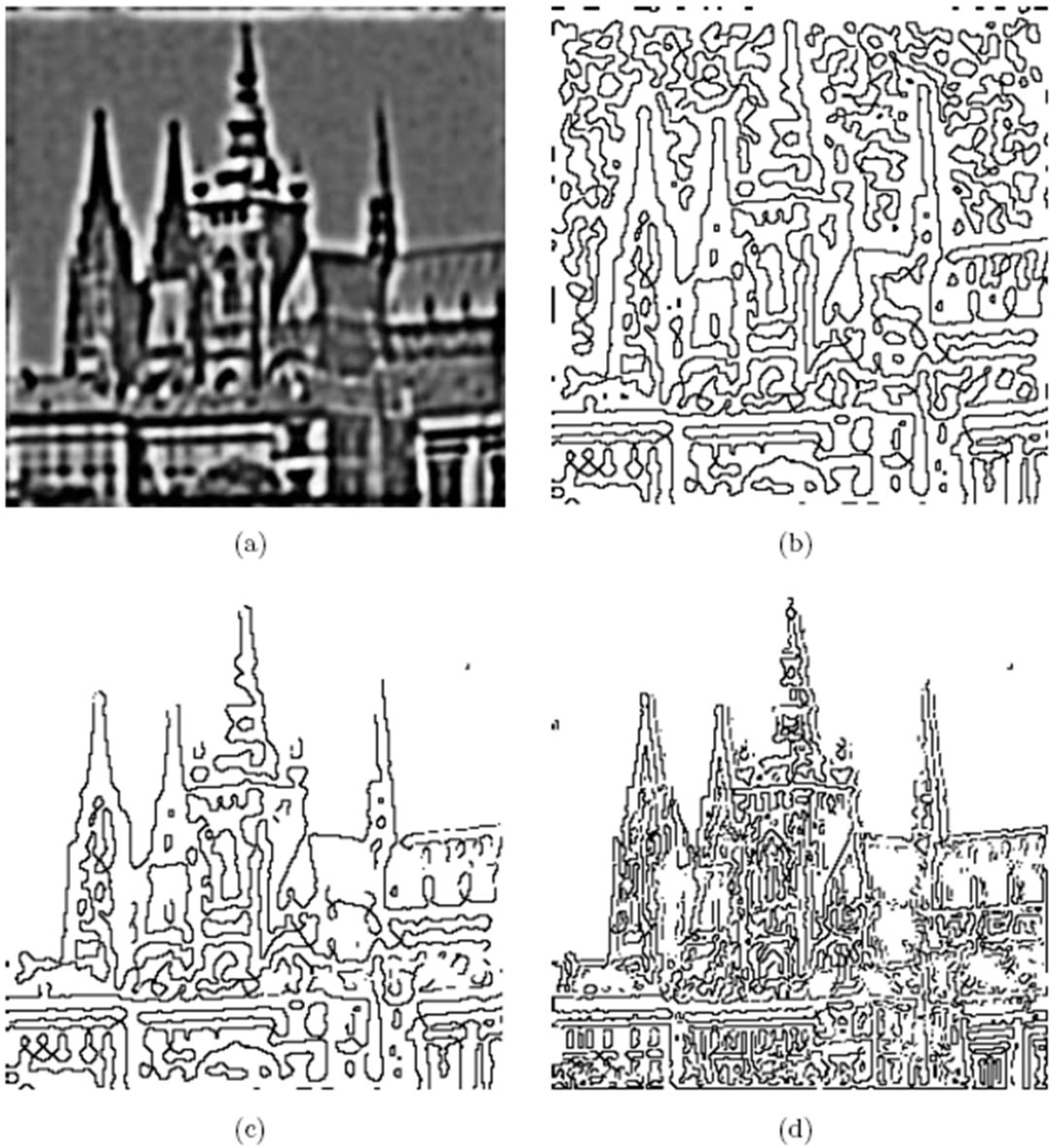


Figure 5.23: Zero-crossings of the second derivative, see Figure 5.10a for the original image. (a) DoG image ($\sigma_1 = 0.10, \sigma_2 = 0.09$), dark pixels correspond to negative DoG values, bright pixels represent positive DoG values. (b) Zero-crossings of the DoG image. (c) DoG zero-crossing edges after removing edges lacking first-derivative support. (d) LoG zero-crossing edges ($\sigma = 0.20$) after removing edges lacking first-derivative support—note different scale of edges due to different Gaussian smoothing parameters.

Example of Canny Edge Detection

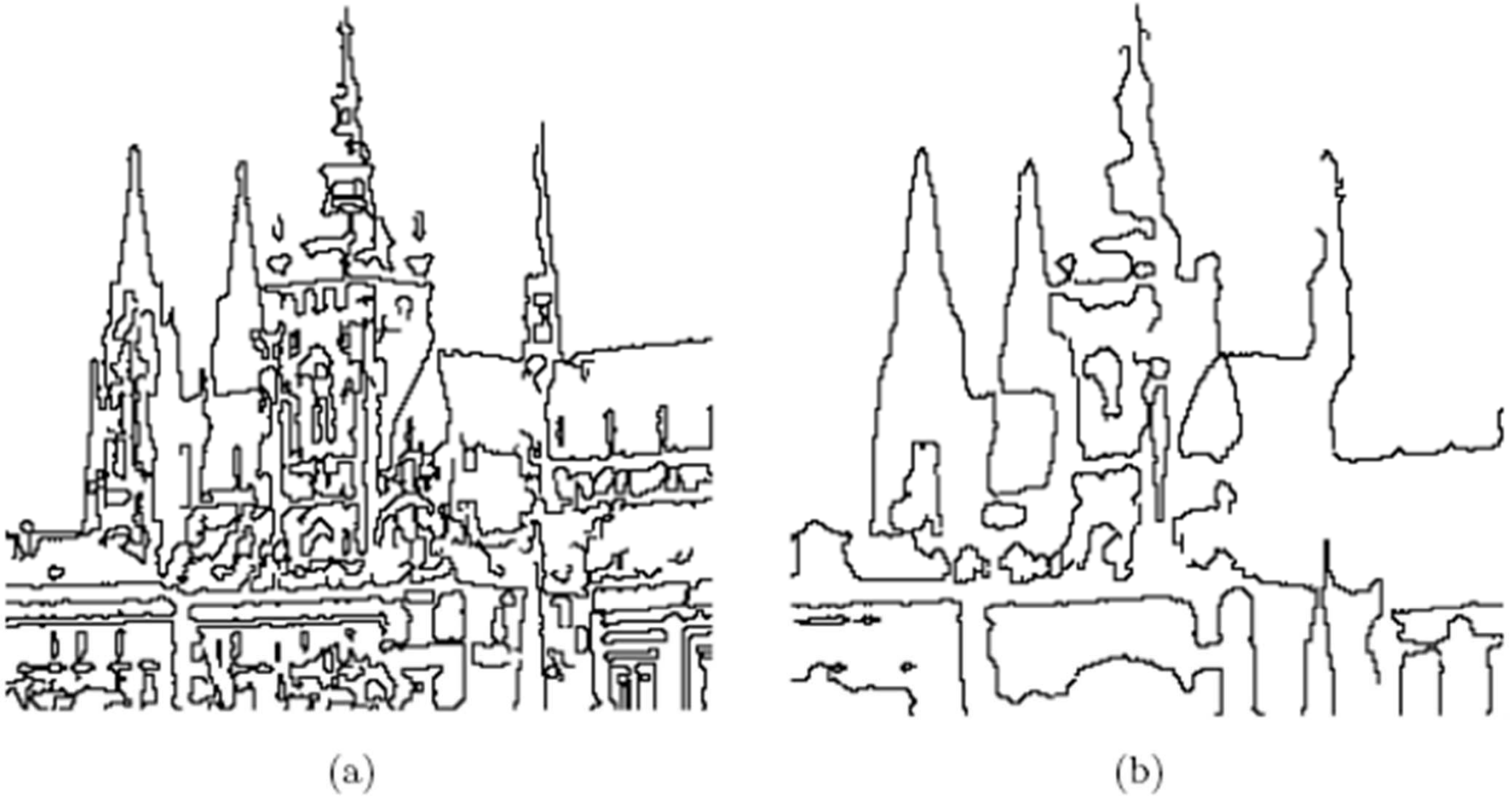


Figure 5.25: Canny edge detection at two different scales.

Corner (Interest Point) Detection

- Correspondence problem
- Aperture problem

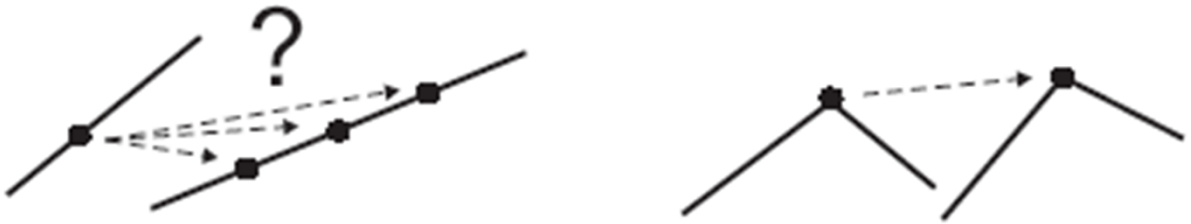


Figure 5.34: Ambiguity of lines for matching and unambiguity of corners.

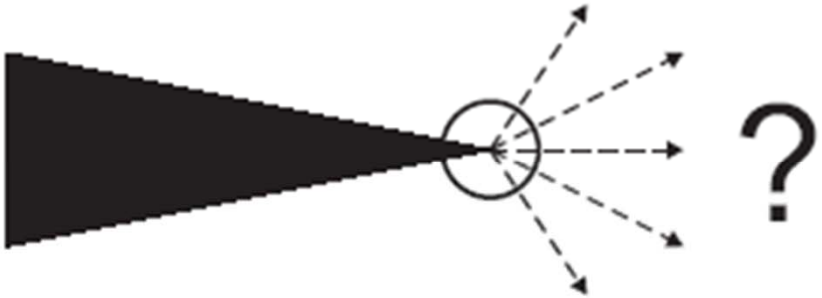


Figure 5.35: Ambiguity of edge detector at the corner tip.

Moravec Corner Detector

- Moravec detector is maximal in pixels with high contrast. These points are on corners and sharp edges.

$$MO(i,j) = \frac{1}{8} \sum_{k=i-1}^{i+1} \sum_{l=j-1}^{j+1} |g(k,l) - g(i,j)| \qquad (5.71)$$

Zuniga-Haralick(ZH) Corner Detector

- The image function f is approximated in the neighborhood of the pixel (i,j) by a cubic polynomial with coefficients c_k. This is a cubic facet model.

$$g(i,j) = c_1 + c_2 x + c_3 y + c_4 x^2 + c_5 xy + c_6 y^2 + c_7 x^3 + c_8 x^2 y + c_9 x y^2 + c_{10} y^3$$

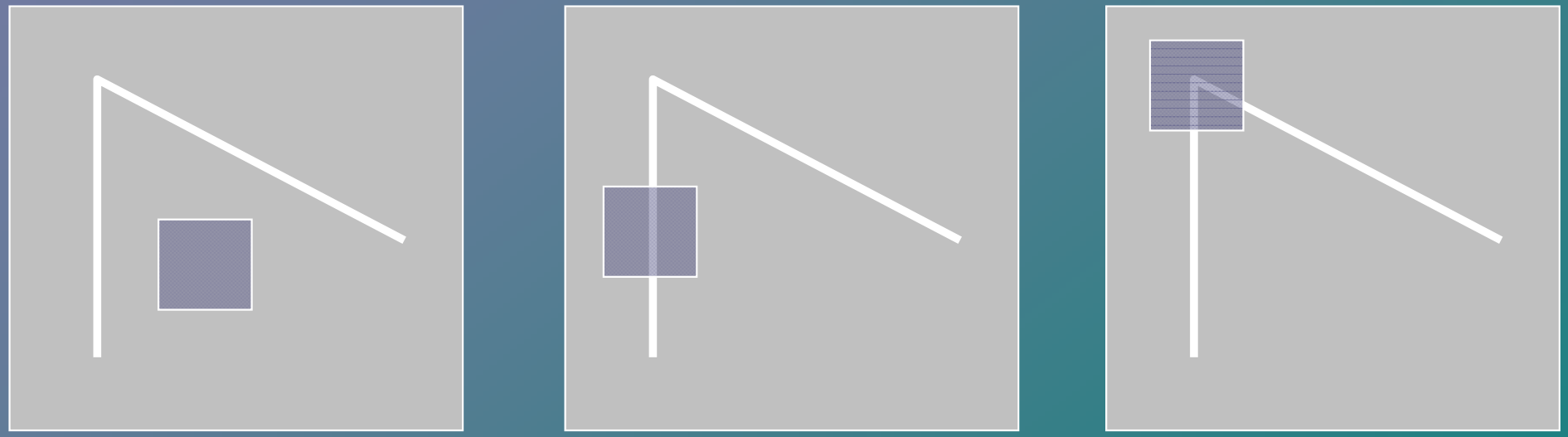
- The ZH operator estimates the corner strength based on the coefficients of the cubic facet model

$$ZH(i,j) = \frac{-2\left(c_2^2 c_6 - c_2 c_3 c_5 - c_3^2 c_4\right)}{\left(c_2^2 + c_3^2\right)^{\frac{3}{2}}} \qquad (5.73)$$

Local measures of uniqueness

Suppose we only consider a small window of pixels

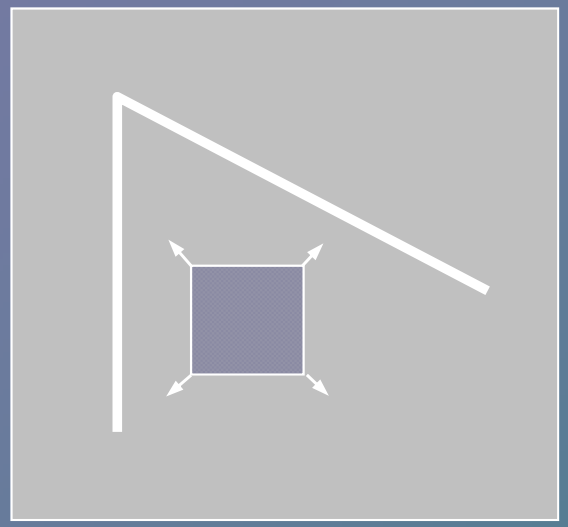
- What defines whether a feature is a good or bad candidate?



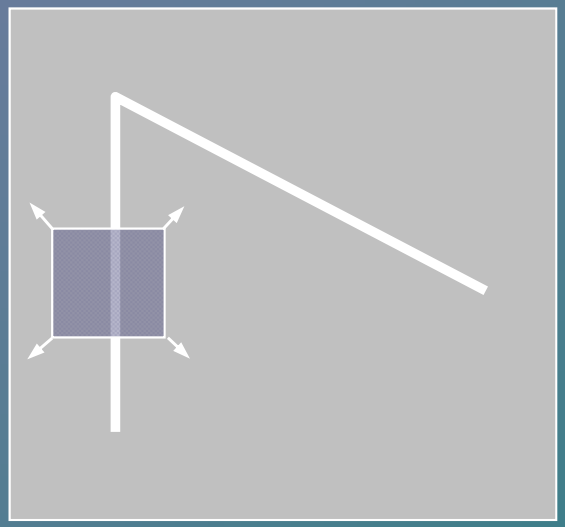
Corner detection

Local measure of feature uniqueness

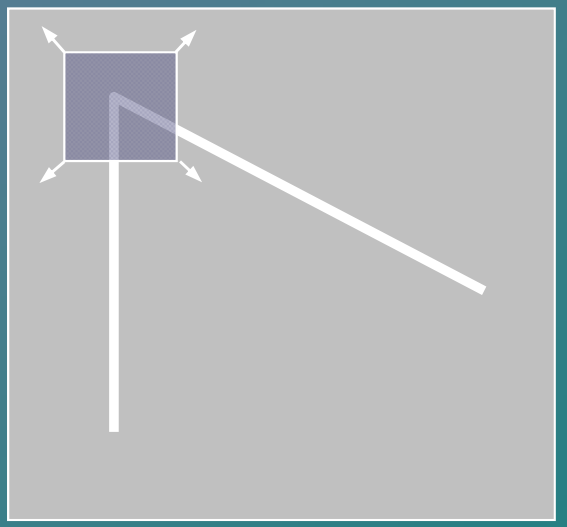
- How does the window change when you shift it?
- Shifting the window in *any direction* causes a *big change*



"flat" region:
no change in all directions



"edge":
no change along the edge direction

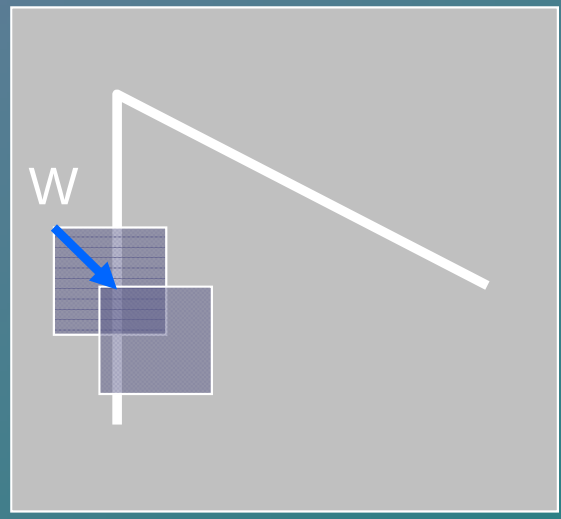


"corner":
significant change in all directions

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" of *E(u,v)*:



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

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

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

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

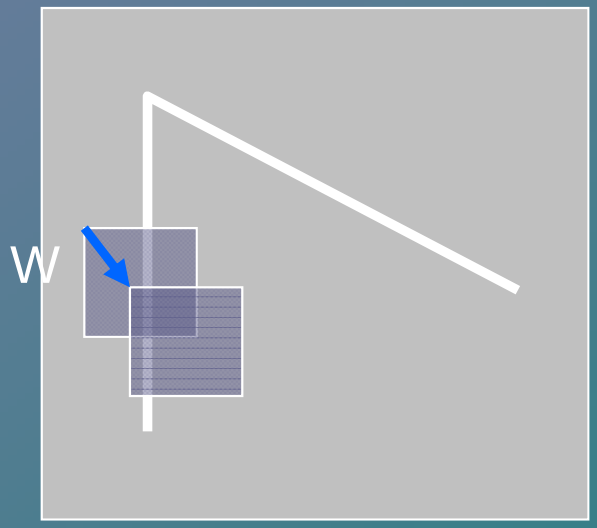
shorthand: $I_x = \frac{\partial I}{\partial x}$

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):



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

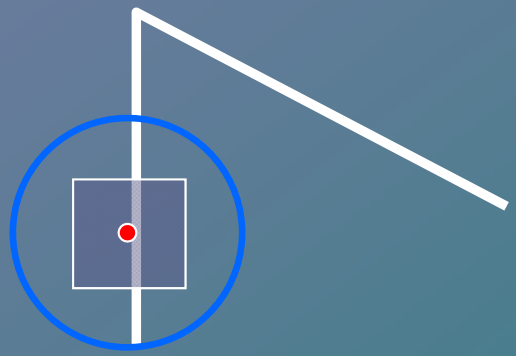
$$\approx \sum_{(x,y)\in W} [I(x,y) + [I_x \; I_y] \begin{bmatrix} u \\ v \end{bmatrix} - I(x,y)]^2$$

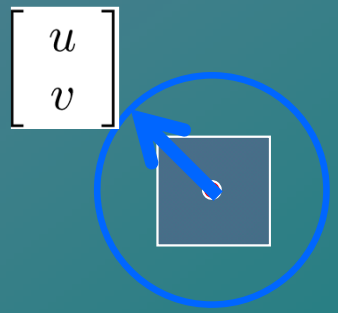
$$\approx \sum_{(x,y)\in W} \left[[I_x \; I_y] \begin{bmatrix} u \\ v \end{bmatrix} \right]^2$$

Corner detection: the math

This can be rewritten:

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





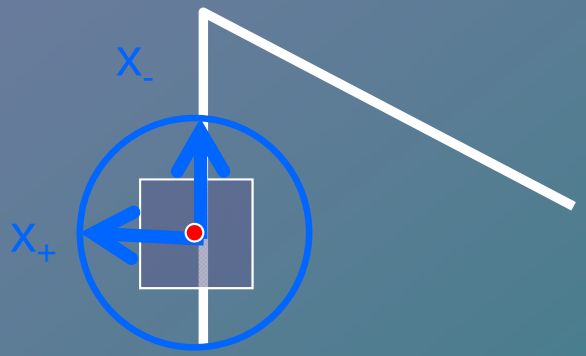
For the example above

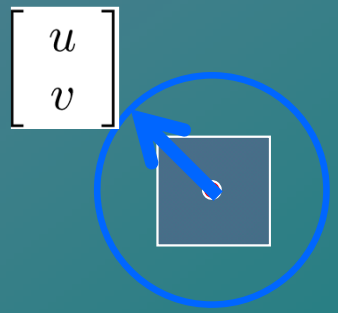
- You can move the center of the green window to anywhere on the blue unit circle
- Which directions will result in the largest and smallest E values?
- We can find these directions by looking at the eigenvectors of *H*

Feature detection: the math

This can be rewritten:

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



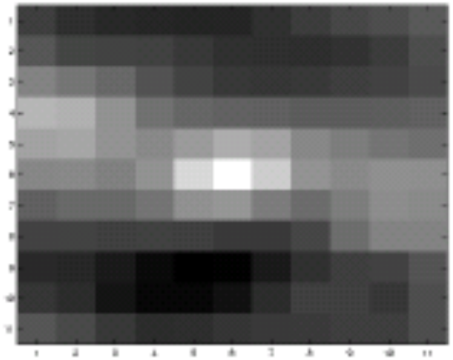


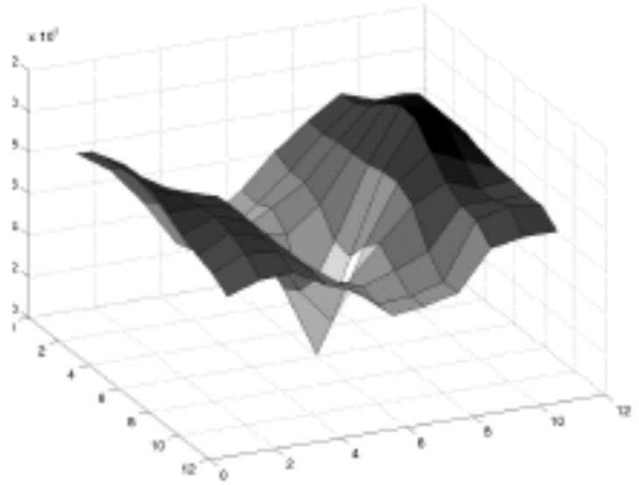
Eigenvalues and eigenvectors of H

- 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_+

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

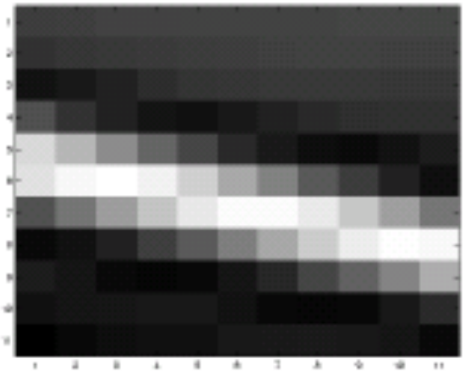
Interest Point Detection

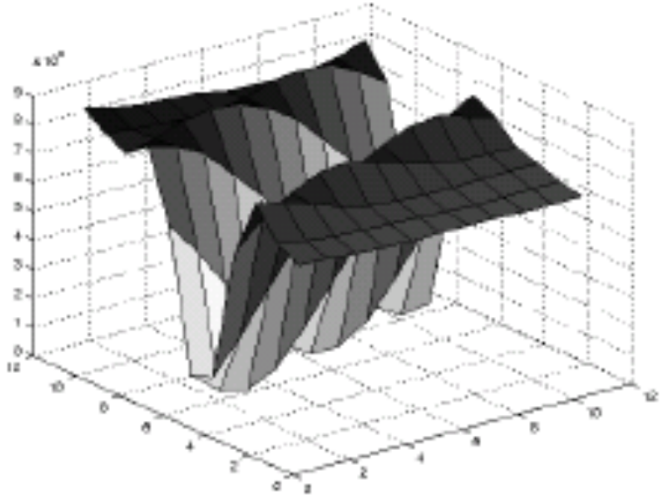




Both λ_+ and λ_- are large

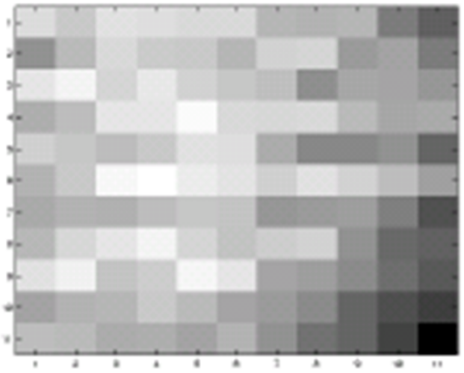
Interest Point Detection

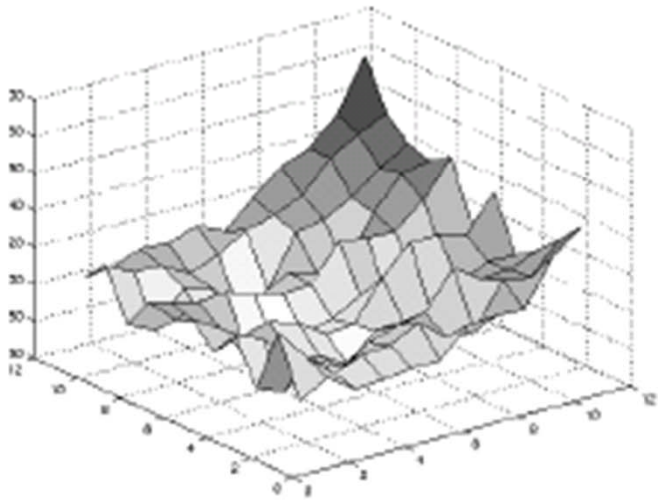




Large λ_+ and small λ_-

Interest Point Detection





Small λ_+ and small λ_-

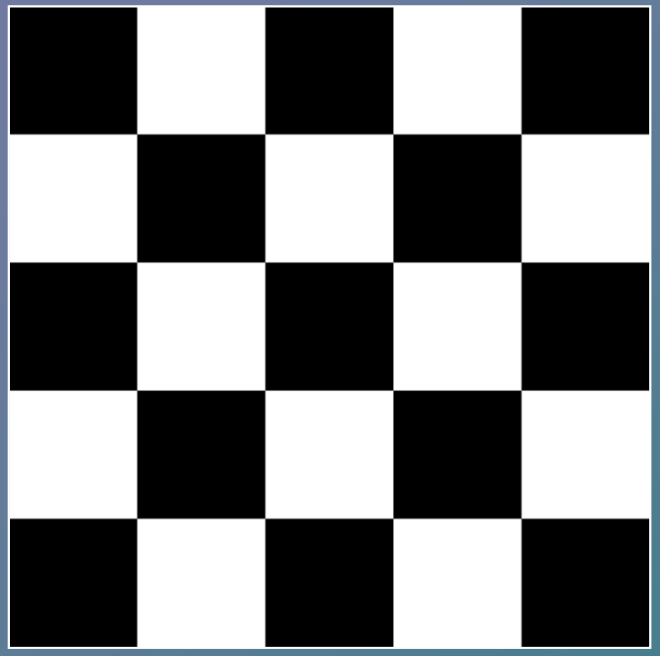
Corner detection: the math

How are λ_+, x_+, λ_-, and x_+ relevant for feature detection?

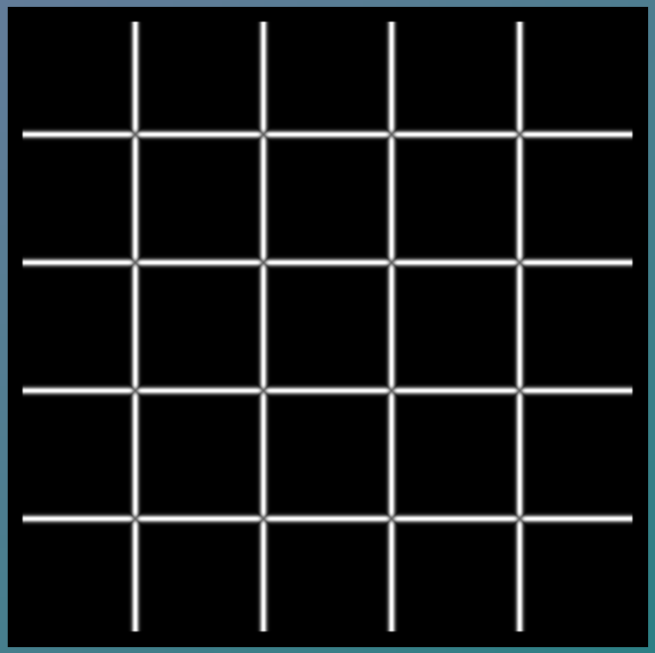
- What's our feature scoring function?

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

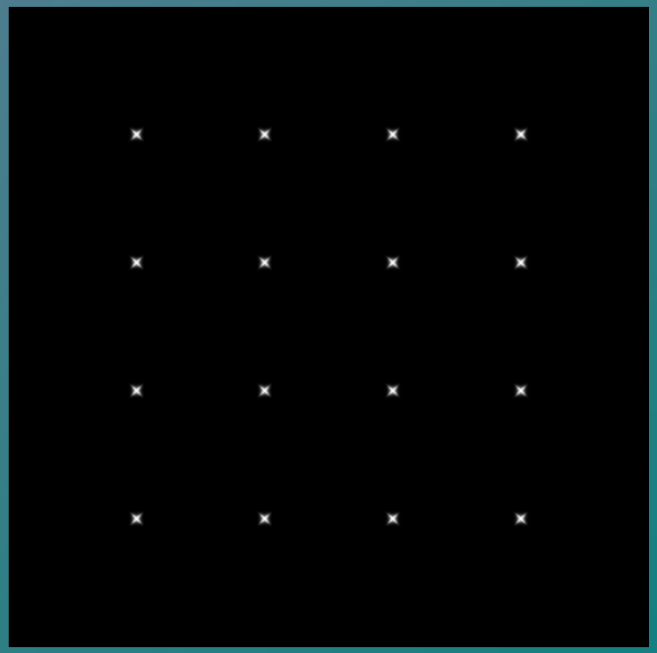
- the *minimum* of *E(u,v)* should be large, over all unit vectors [u v]
- this minimum is given by the smaller eigenvalue (λ_-) of *H*



I



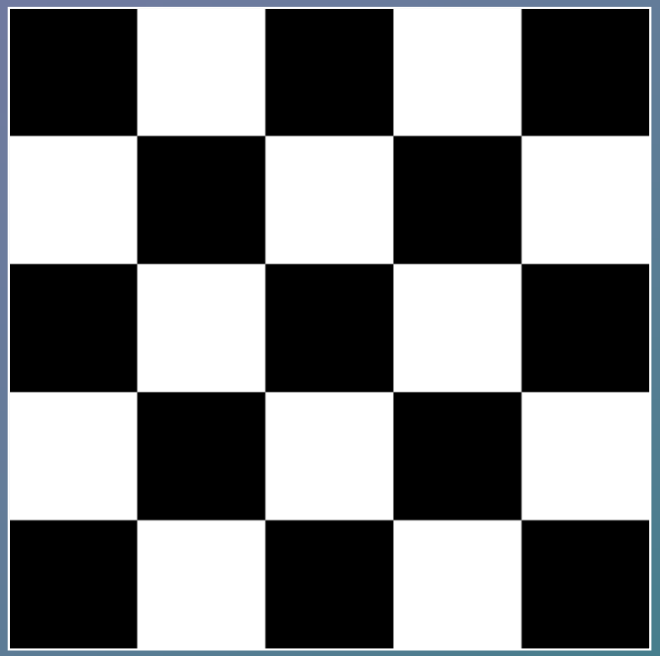
λ_+



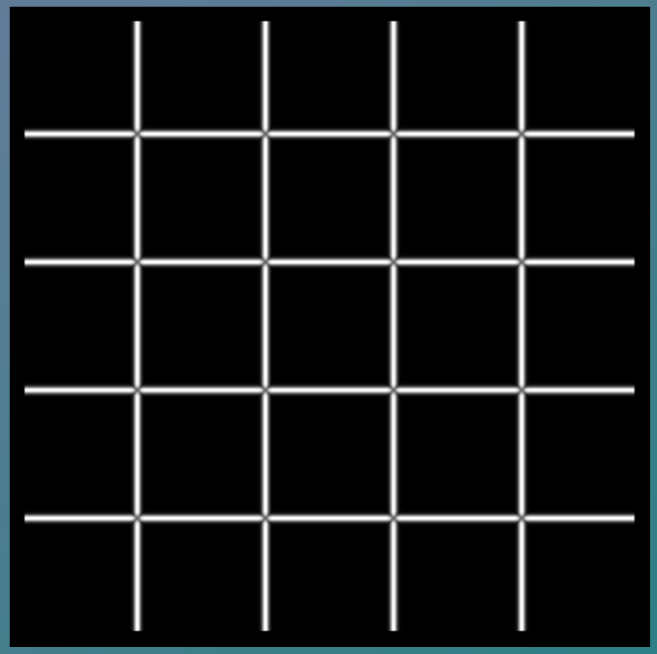
λ_-

Corner detection summary

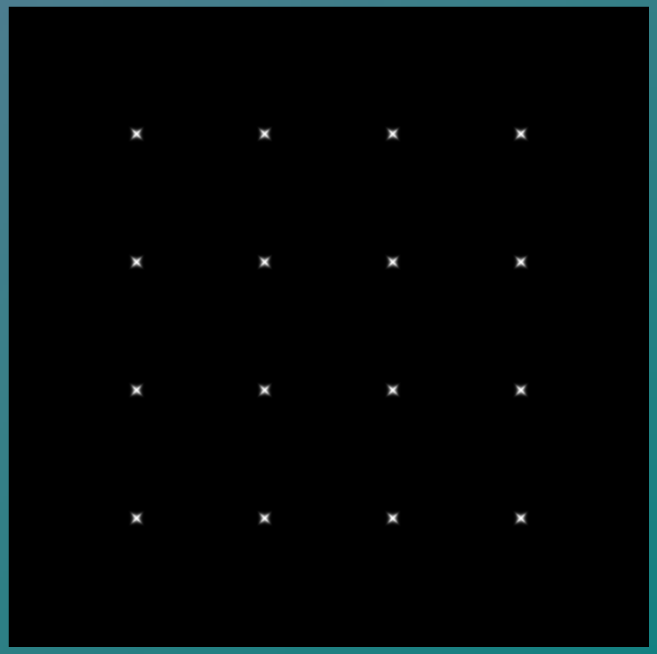
- Compute the gradient at each point in the image
- Create the *H* matrix from the entries in the gradient
- Compute the eigenvalues.
- Find points with large response (λ_- > threshold)
- Choose those points where λ_- is a local maximum as features



I



λ_+

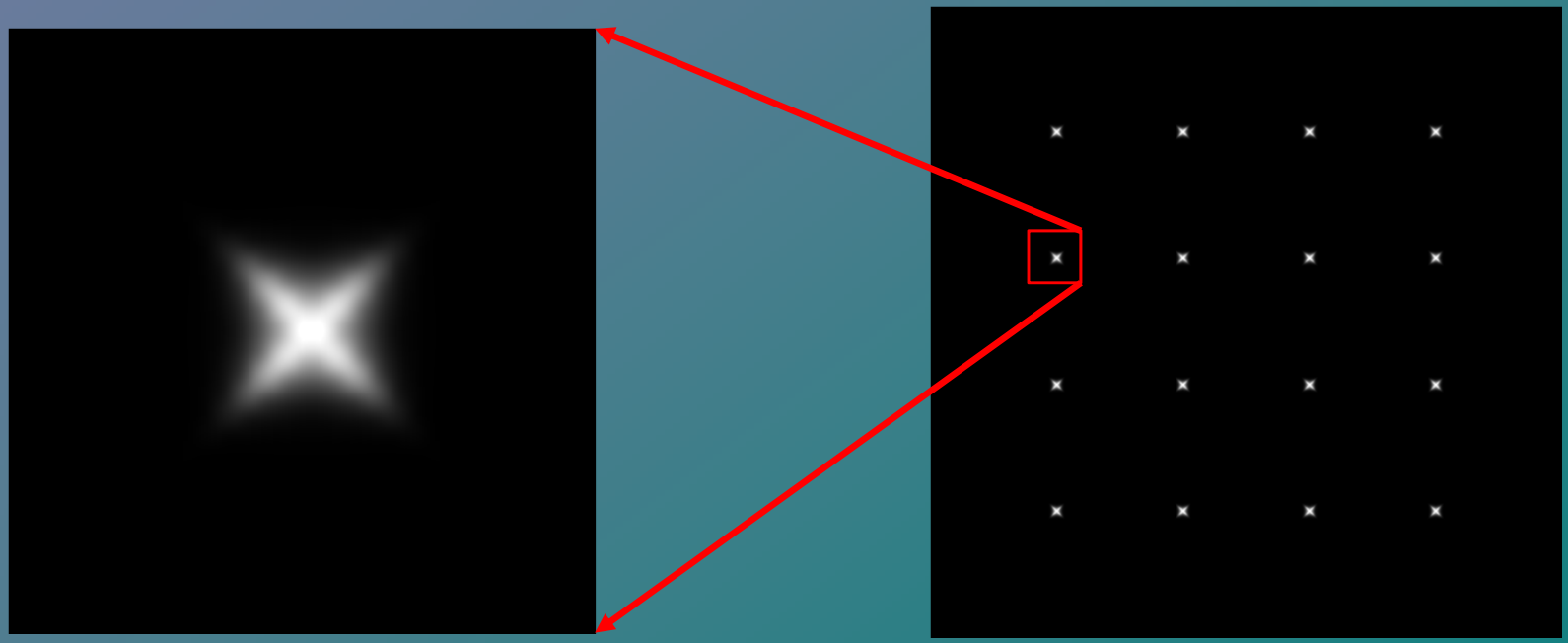


λ_-

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



The Harris operator

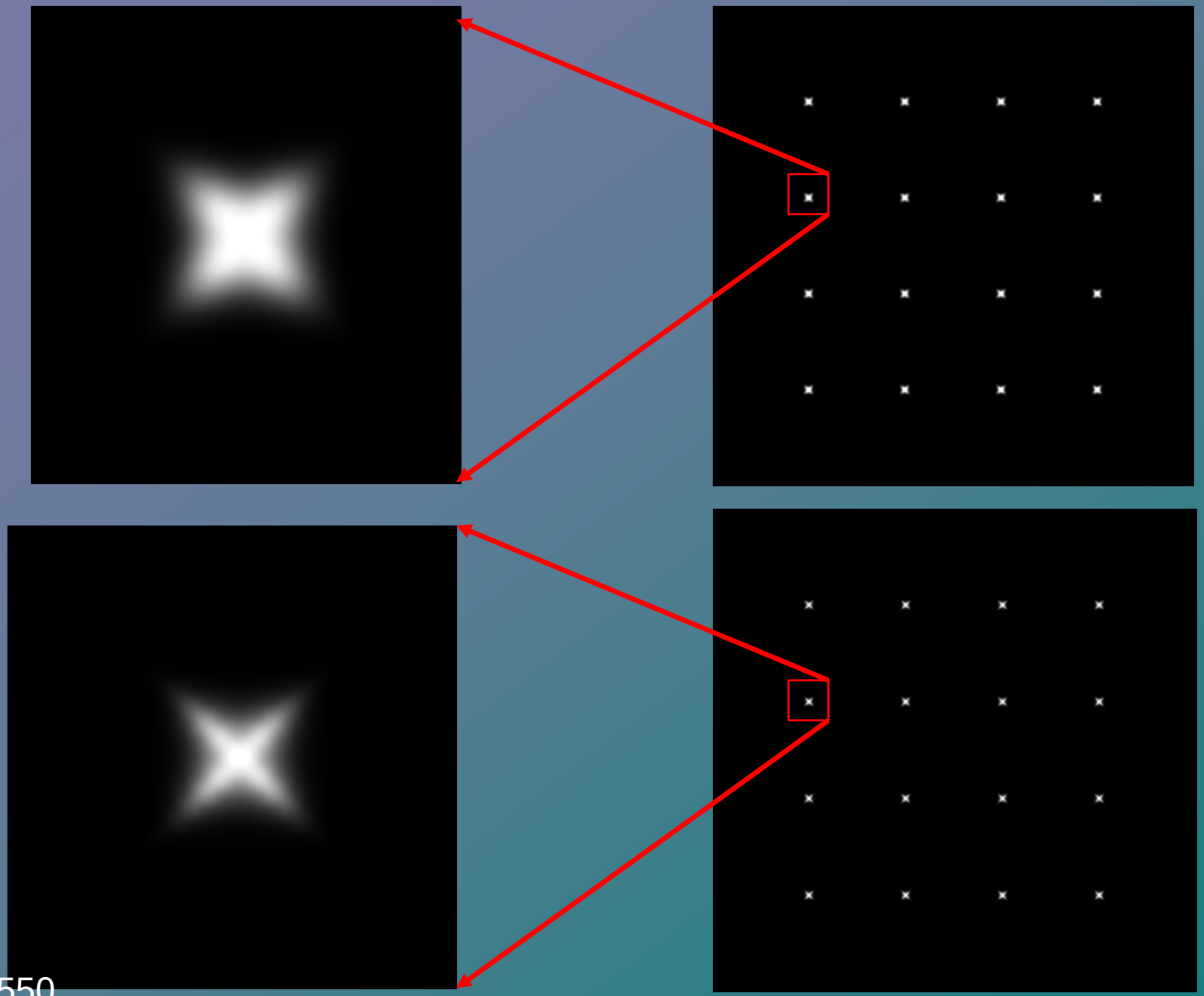
λ_- is a variant of the "Harris operator" for feature detection

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

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

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



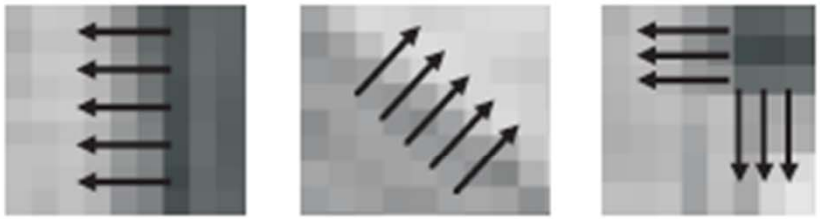


Figure 5.36: Illustration of the decision within Harris corner detector according to eigenvalues of the local structure matrix. (a), (b) Ridge detected, no corner at this position. (c) Corner detected.

The corner detection algorithm we have been describing is due to Harris (1987). It is necessary to calculate A at every pixel and mark corners where the quantity $\lambda_1\lambda_2 - \kappa(\lambda_1+\lambda_2)^2$ exceeds some threshold ($\kappa \approx 0.04$ makes the detector a little "edge-phobic"). Note that $\det A = \lambda_1\lambda_2$ and trace $A = \lambda_1 + \lambda_2$.

Low threshold

High threshold

Harris Detector: Mathematics

Measure of corner response:

$$R = \det \mathbf{A} - k(trace\,\mathbf{A})^2$$

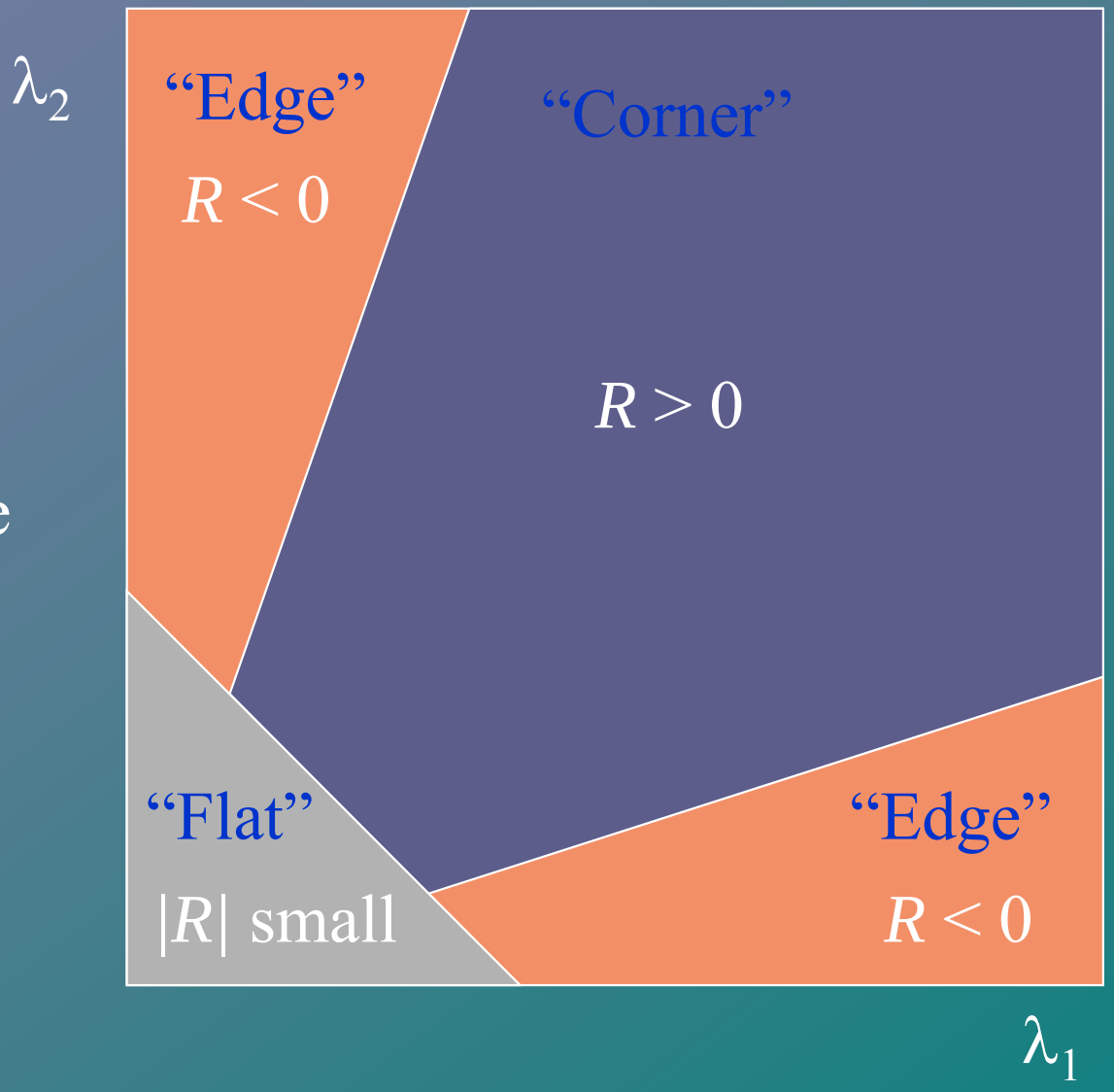
$$\det \mathbf{A} = \lambda_1 \lambda_2$$

$$trace\,\mathbf{A} = \lambda_1 + \lambda_2$$

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

Harris Detector

- R depends only on eigenvalues of M
- R is large for a corner
- R is negative with large magnitude for an edge
- $|R|$ is small for a flat region



Harris Corner Detector

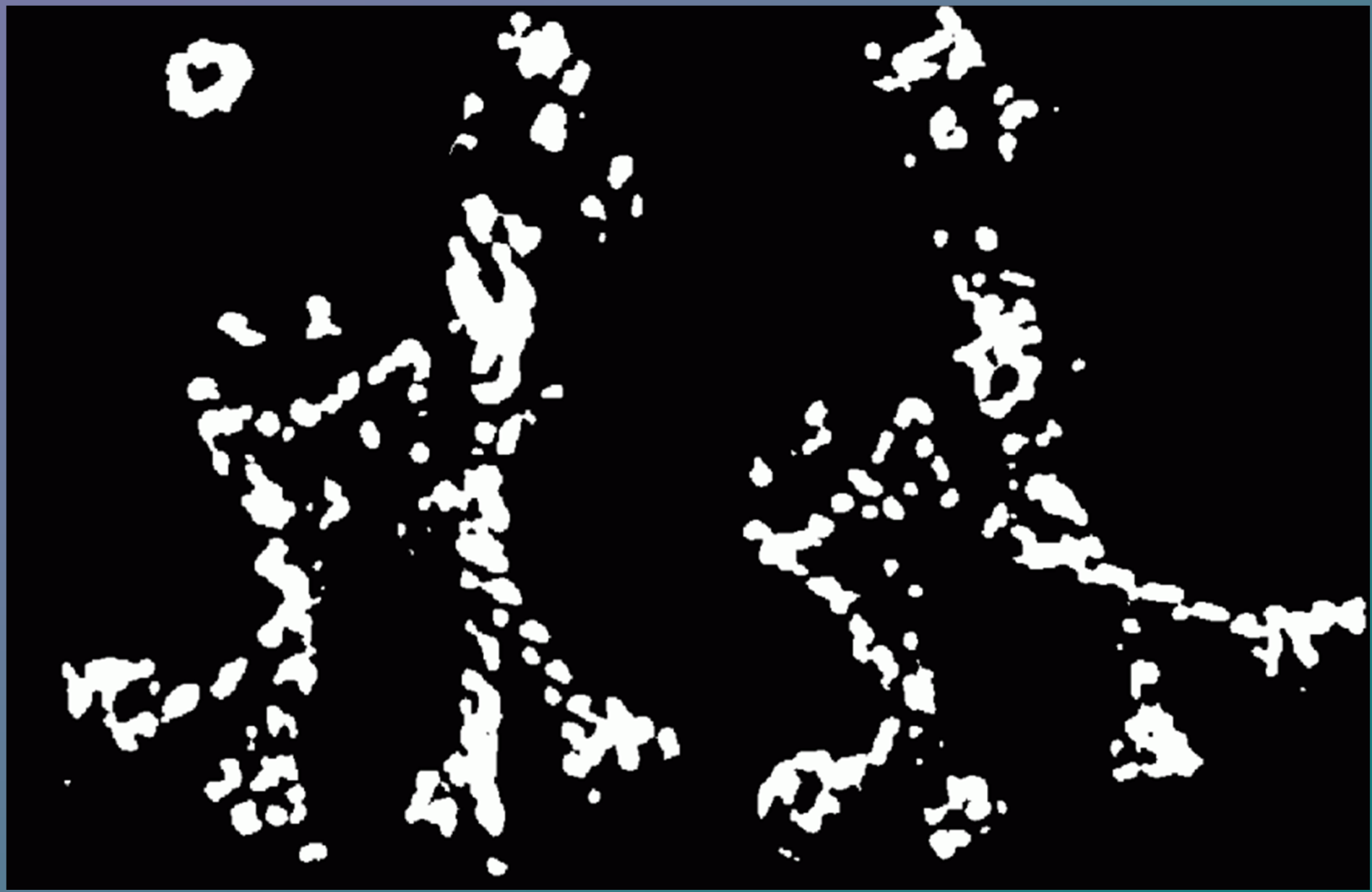
Algorithm 5.5: Harris corner detector

1. Filter the image with a Gaussian.
2. Estimate intensity gradient in two perpendicular directions for each pixel, $\frac{\partial f(x,y)}{\partial x}$, $\frac{\partial f(x,y)}{\partial y}$. This is performed by twice using a 1D convolution with the kernel approximating the derivative.
3. For each pixel and a given neighborhood window:
   - Calculate the local structure matrix A.
   - Evaluate the response function $R(A)$.
4. Choose the best candidates for corners by selecting a threshold on the response function $R(A)$ and perform non-maximal suppression.

Harris Detector: Workflow

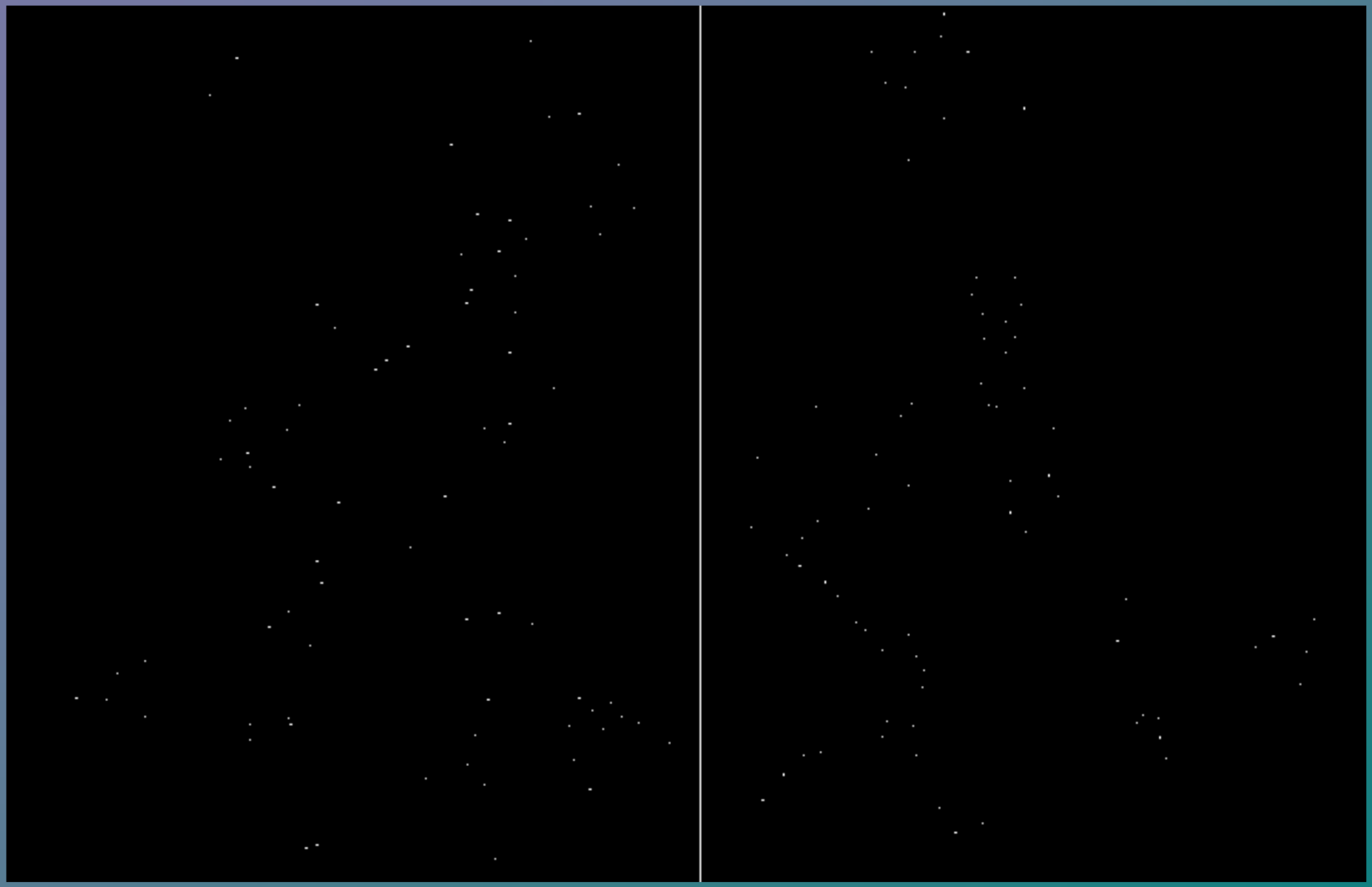
Harris Detector: Workflow

Find points with large corner response: R>threshold



Harris Detector: Workflow

Take only the points of local maxima of R



Harris Detector: Workflow

Example of Harris Corner Detection

Figure 5.37: Example of Harris corners in the image. *Courtesy of Martin Urban, Czech Technical University in Prague, who used such images for 3D reconstruction. A color version of this figure may be seen in the color inset—Plate 7.*

Feature Invariance

Suppose you **rotate** the image by some angle

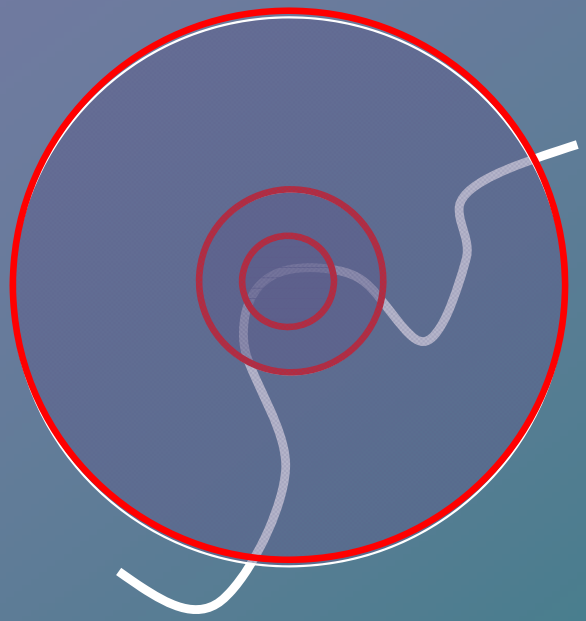
- Will you still pick up the same features?

What if you change the brightness?

Scale?

Scale invariant detection

Suppose you're looking for corners



Key idea: find scale that gives local maximum of f

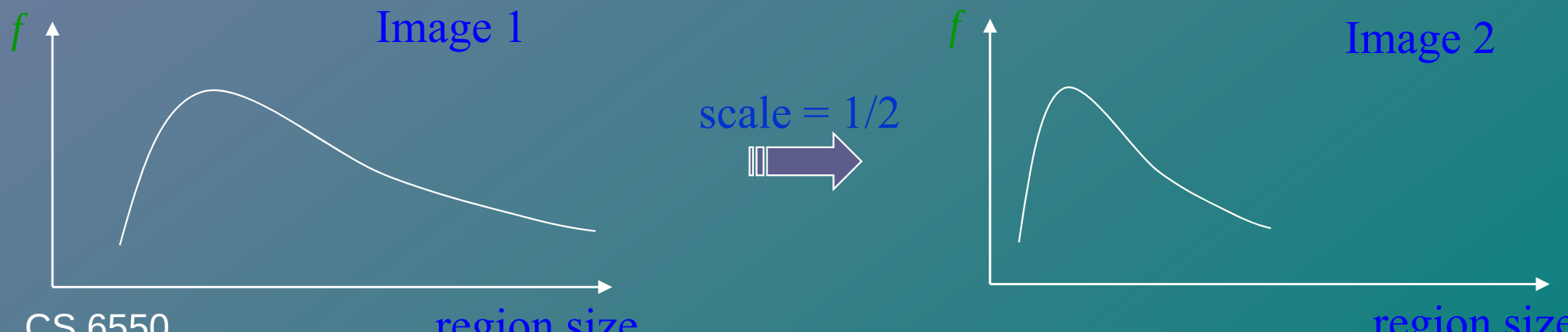
- f is a local maximum in both position and scale
- Common definition of f: Laplacian
  (or difference between two Gaussian filtered images with different sigmas)

Scale Invariant Detection

- Solution:
  - Design a function on the region (circle), which is "scale invariant" (the same for corresponding regions, even if they are at different scales)

Example: average intensity. For corresponding regions (even of different sizes) it will be the same.

  - For a point in one image, we can consider it as a function of region size (circle radius)



Scale Invariant Detection

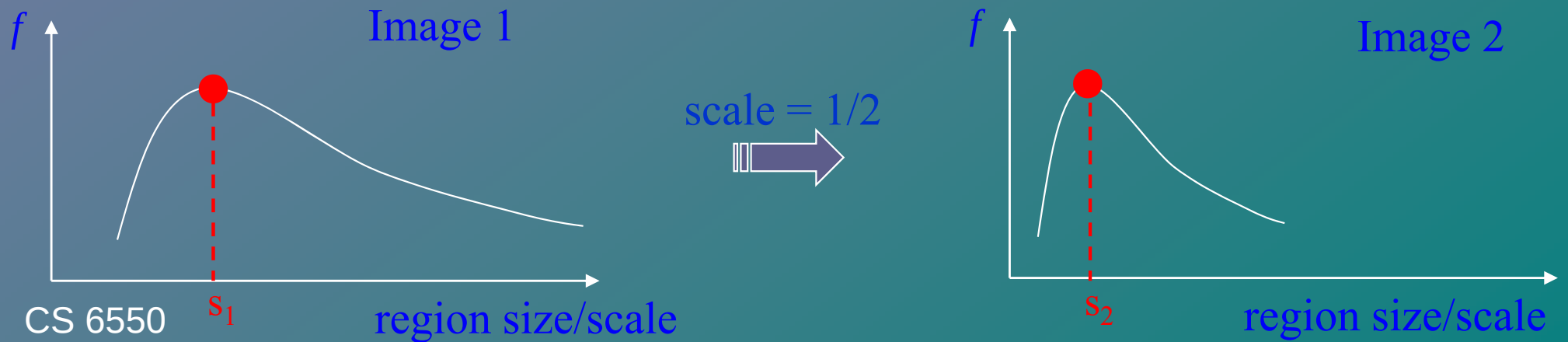
- Common approach:

  Take a local maximum of this function

  Observation: region size (scale), for which the maximum is achieved, should be *invariant* to image scale.

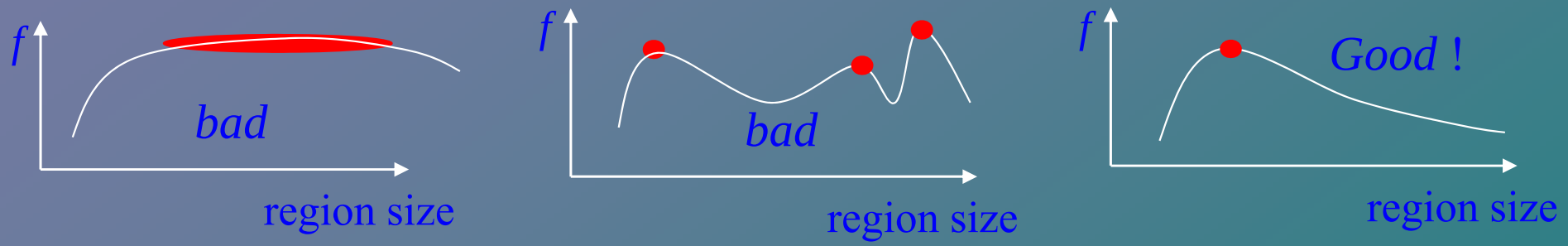
Important: this scale invariant region size is found in each image independently!

Max. is called characteristic scale



Scale Invariant Detection

- A “good” function for scale detection:
  has one stable sharp peak



- For usual images: a good function would be the one with contrast (sharp local intensity change)

Scale Invariant Detection

- Functions for determining scale

$$f = \text{Kernel} * \text{Image}$$

Kernels:

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

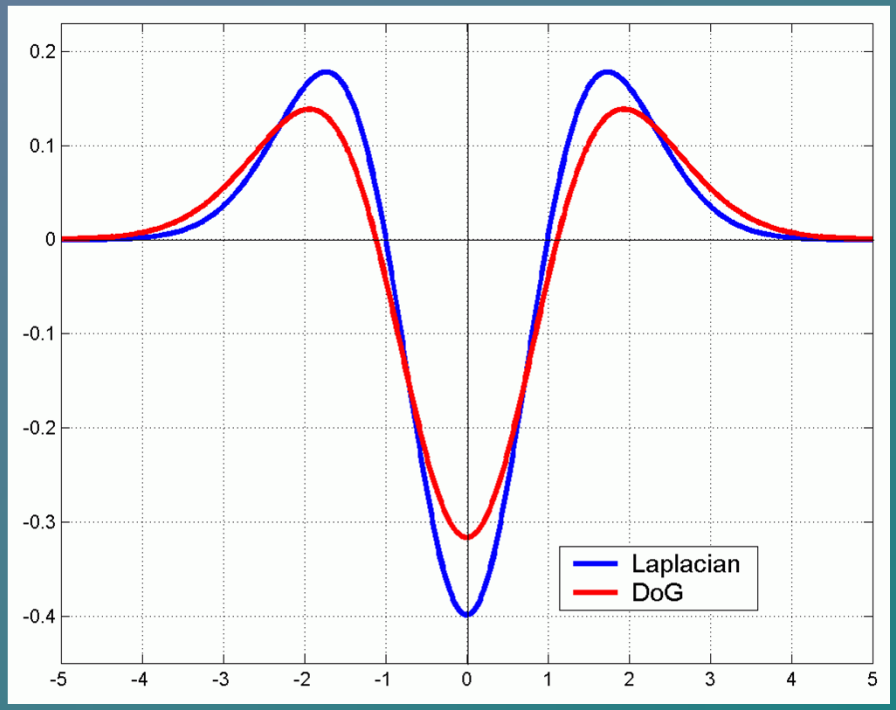
(Laplacian)

$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)

where Gaussian

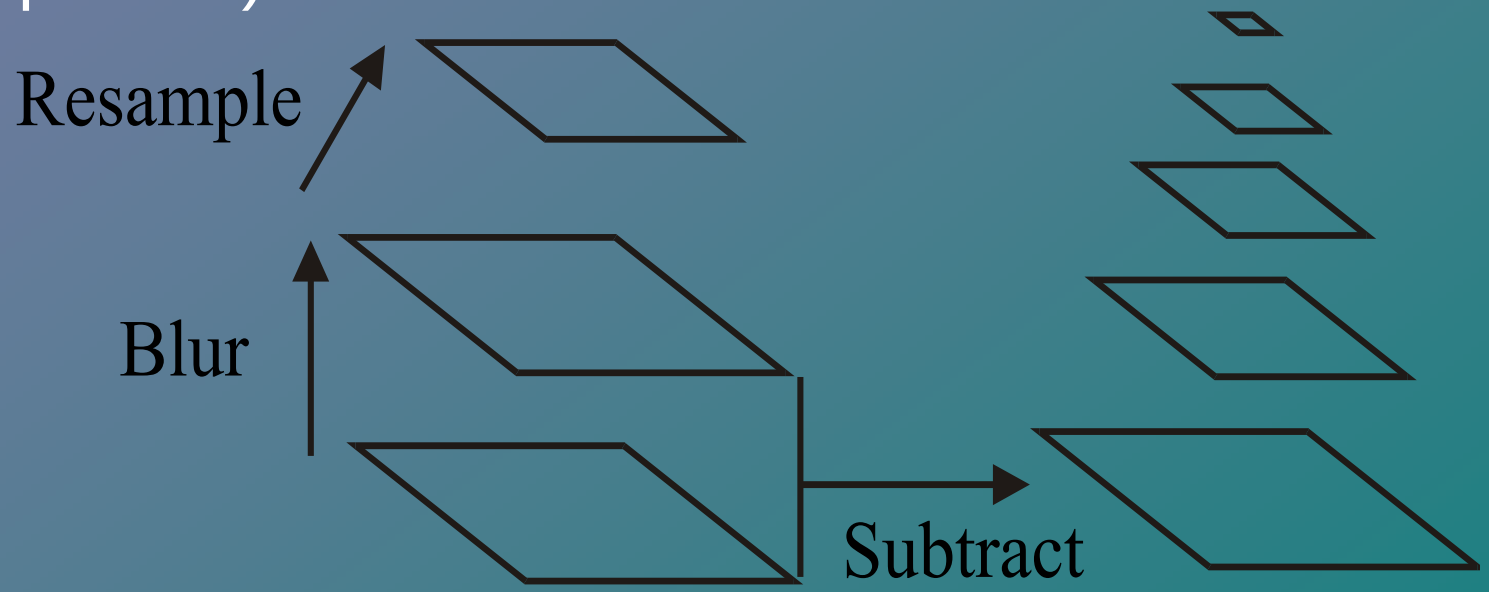
$$G(x, y, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2 + y^2}{2\sigma^2}}$$



Note: both kernels are invariant to *scale* and *rotation*

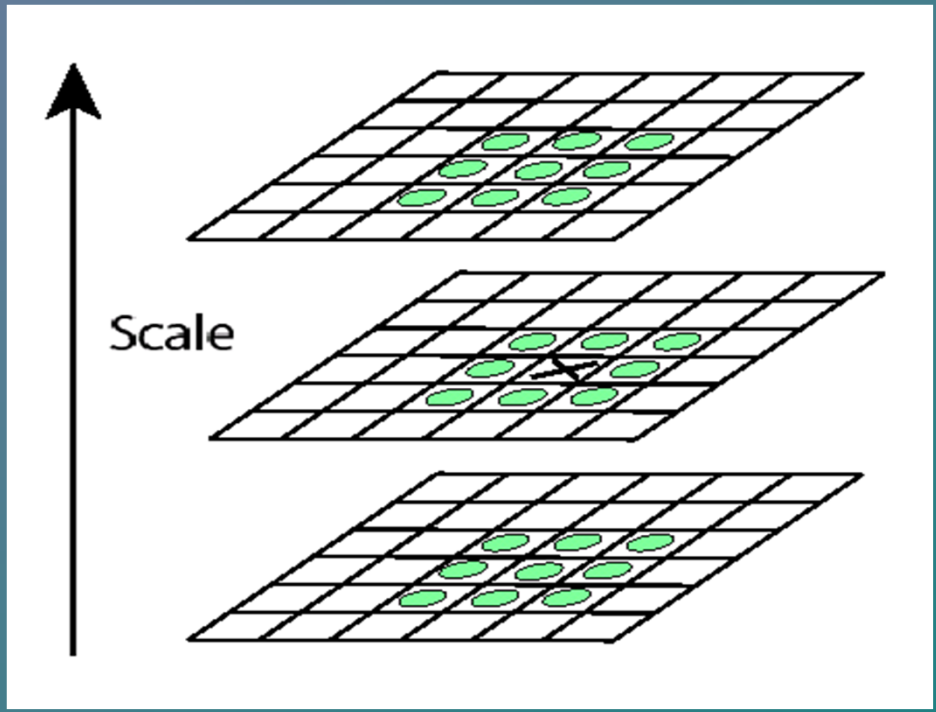
Build Scale-Space Pyramid

- All scales must be examined to identify scale-invariant features
- An efficient function is to compute the Difference of Gaussian (DOG) pyramid (Burt & Adelson, 1983) (or Laplacian)



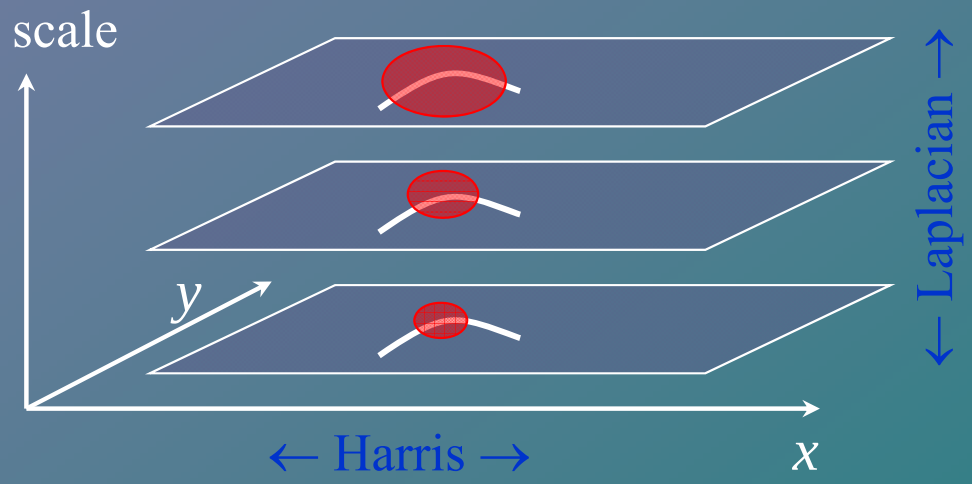
Key point localization

- Detect maxima and minima of difference-of-Gaussian in scale space

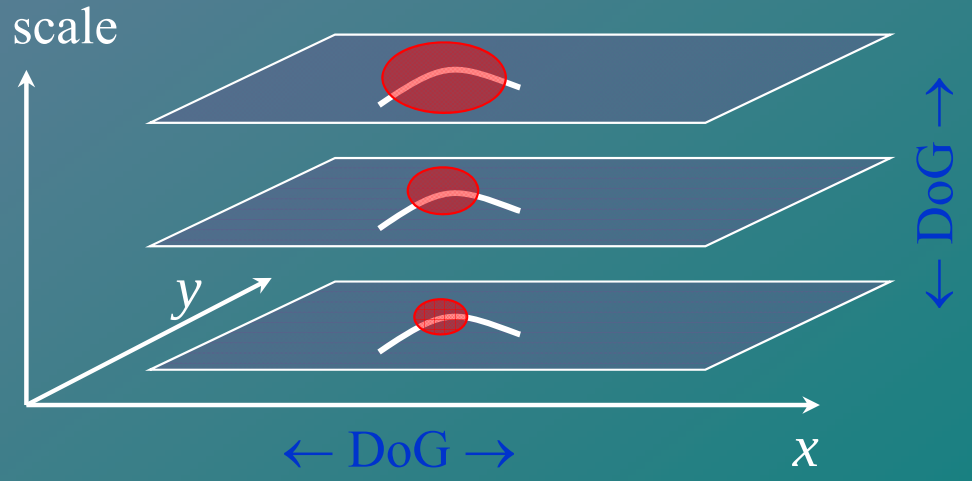


Scale Invariant Detectors

- Harris-Laplacian[1]
  Find local maximum of:
  - Harris corner detector in image space
  - Laplacian in scale

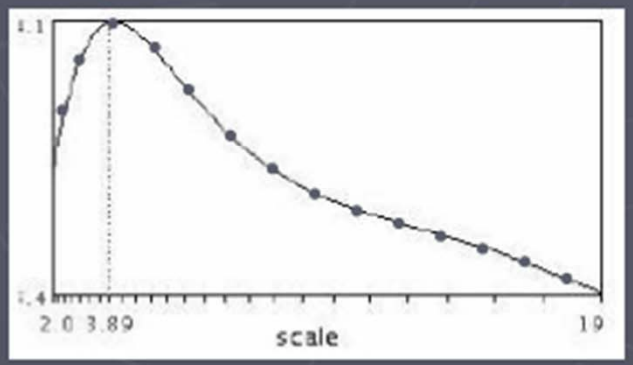


- SIFT (Lowe)[2]
  Find local maximum of:
  - Difference of Gaussians in space and scale

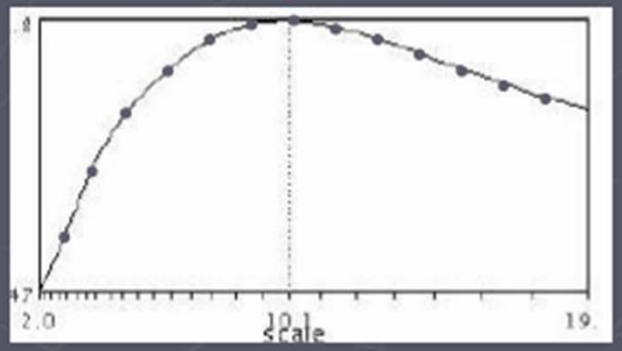


[1] K.Mikolajczyk, C.Schmid. "Indexing Based on Scale Invariant Interest Points". ICCV 2001
[2] D.Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". Accepted to IJCV 2004

Automatic scale selection



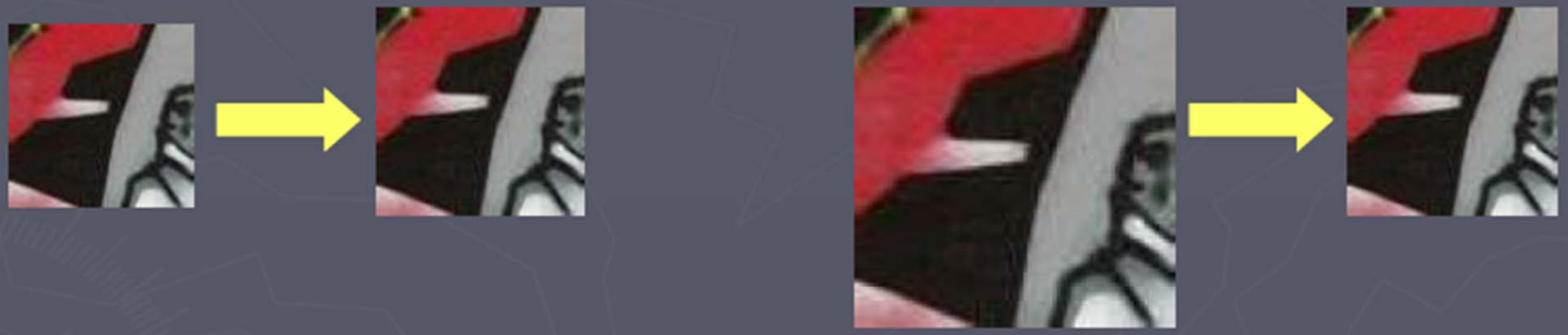
$f(I_{i_1 \ldots i_m}(x,\sigma))$



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

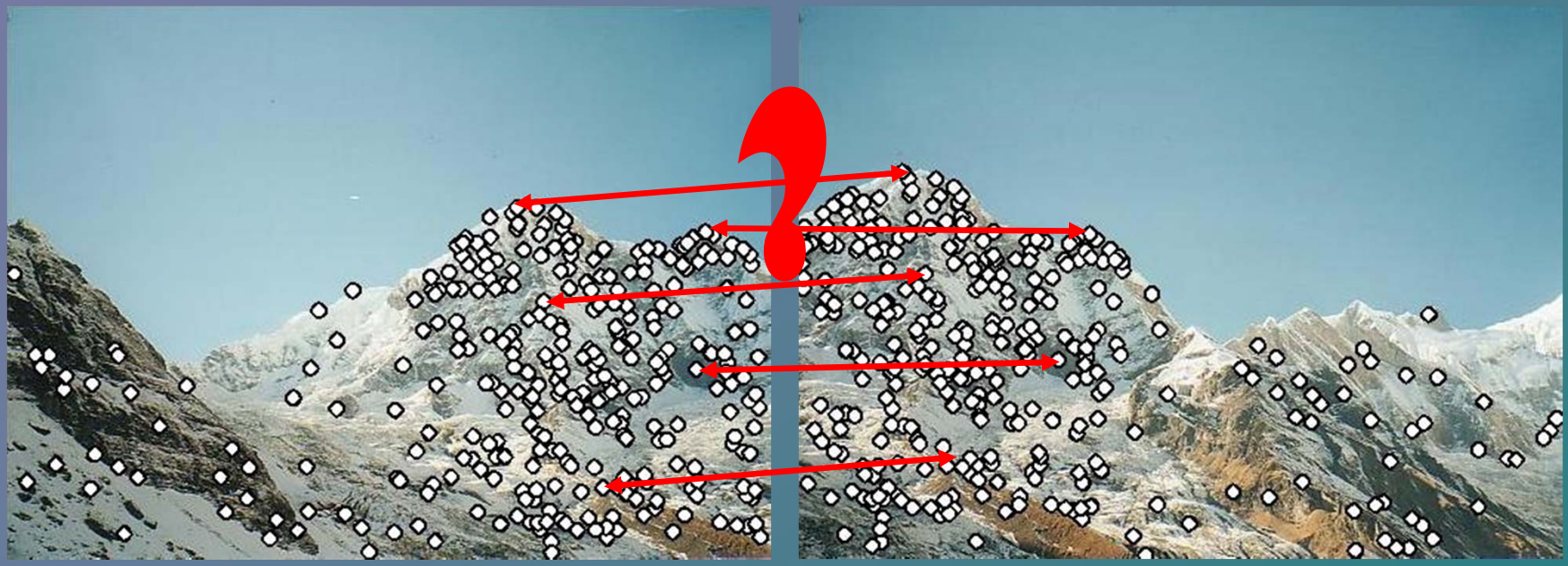
Automatic scale selection

Normalize: rescale to fixed size



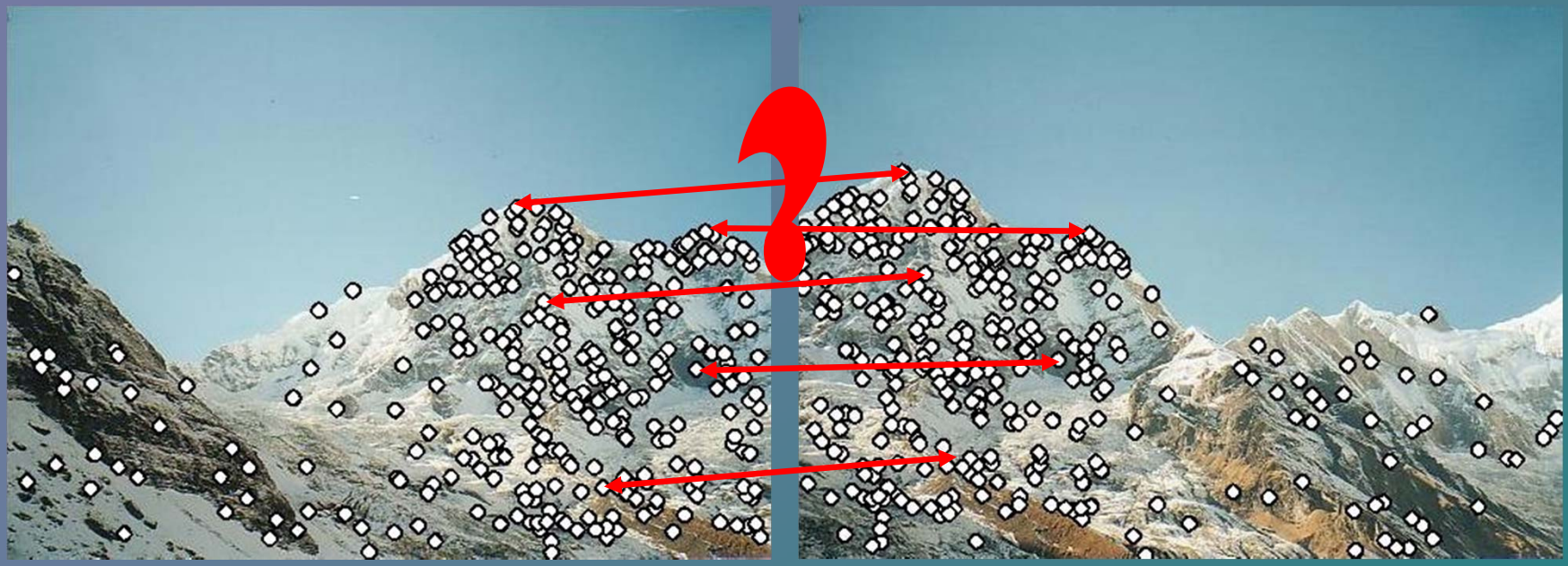
Feature descriptors

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



Feature descriptors

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



Lots of possibilities (this is a popular research area)

- Simple option: match square windows around the point
- State of the art approach: SIFT
  - David Lowe, UBC http://www.cs.ubc.ca/~lowe/keypoints/

Invariance

- Suppose we are comparing two images I_1 and I_2
  - I_2 may be a transformed version of I_1
  - What kinds of transformations are we likely to encounter in practice?
- We'd like to find the same features regardless of the transformation
  - This is called transformational ***invariance***
  - Most feature methods 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

Need both of the following:

1. Make sure your detector is invariant
   - Harris is invariant to translation and rotation
   - Scale is trickier
     - common approach is to detect features at many scales using a Gaussian pyramid (e.g., MOPS)
     - More sophisticated methods find “the best scale” to represent each feature (e.g., SIFT)
2. Design an invariant feature *descriptor*
   - A descriptor captures the information in a region around the detected feature point
   - The simplest descriptor: a square window of pixels
     - What’s this invariant to?
   - Let’s look at some better approaches…

Rotation invariance for feature descriptors

Find dominant orientation of the image patch

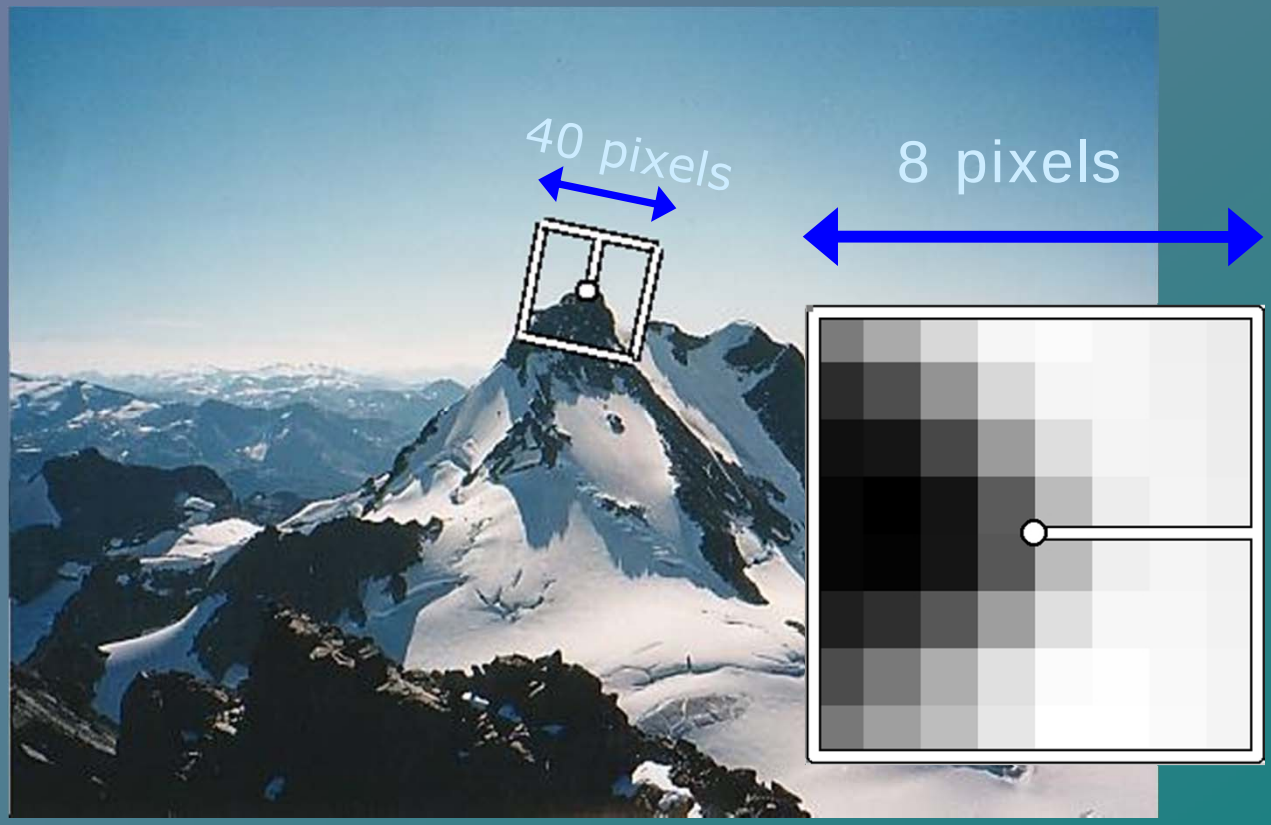
- This is given by $\mathbf{x}_+$, the eigenvector of **H** corresponding to λ_+
  - λ_+ is the *larger* eigenvalue
- Rotate the patch according to this angle

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

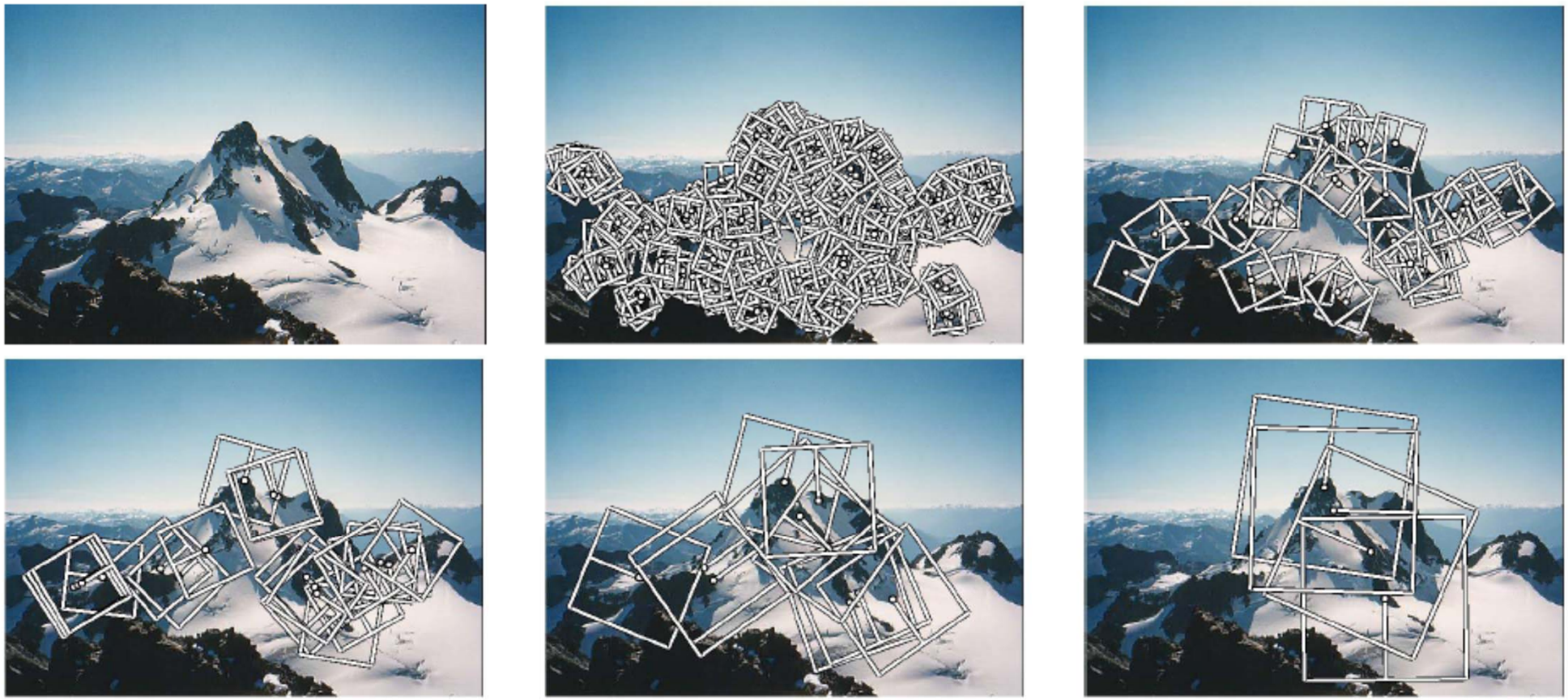
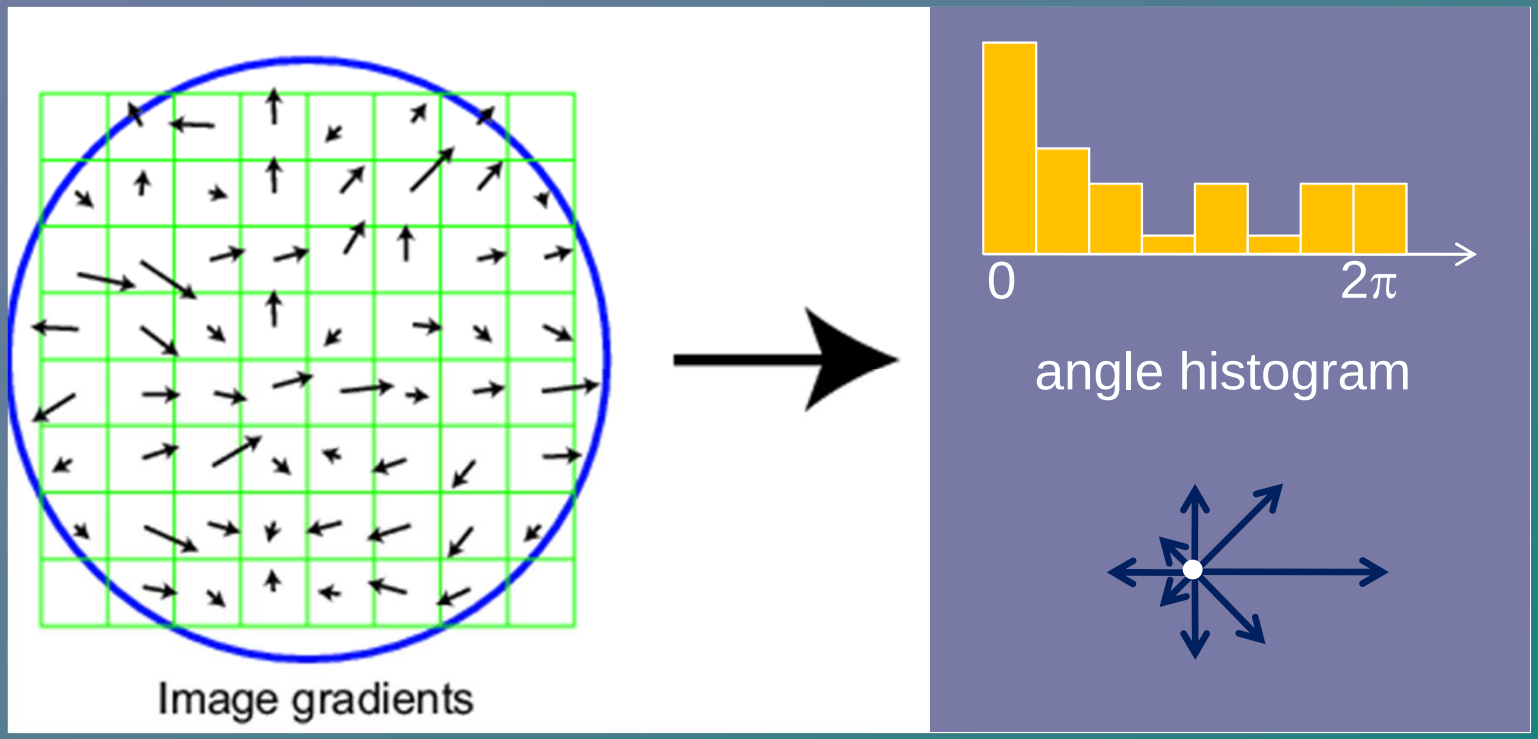


Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.

Scale Invariant Feature Transform

Basic idea:

- Take 16x16 square window around detected feature
- Compute edge orientation (angle of the gradient - 90°) for each pixel
- Throw out weak edges (threshold gradient magnitude)
- Create histogram of surviving edge orientations

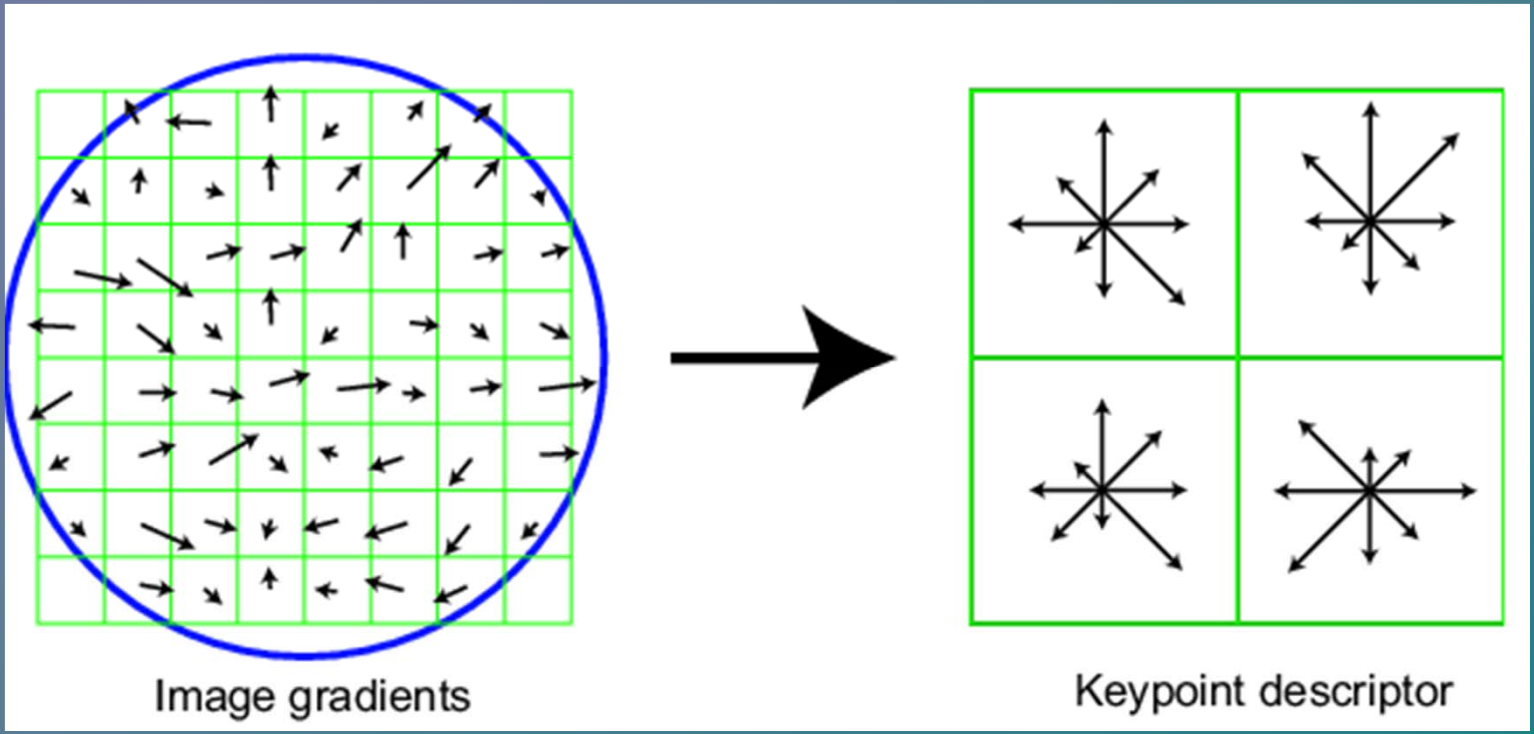


Adapted from slide by David Lowe

SIFT descriptor

Full version

- 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

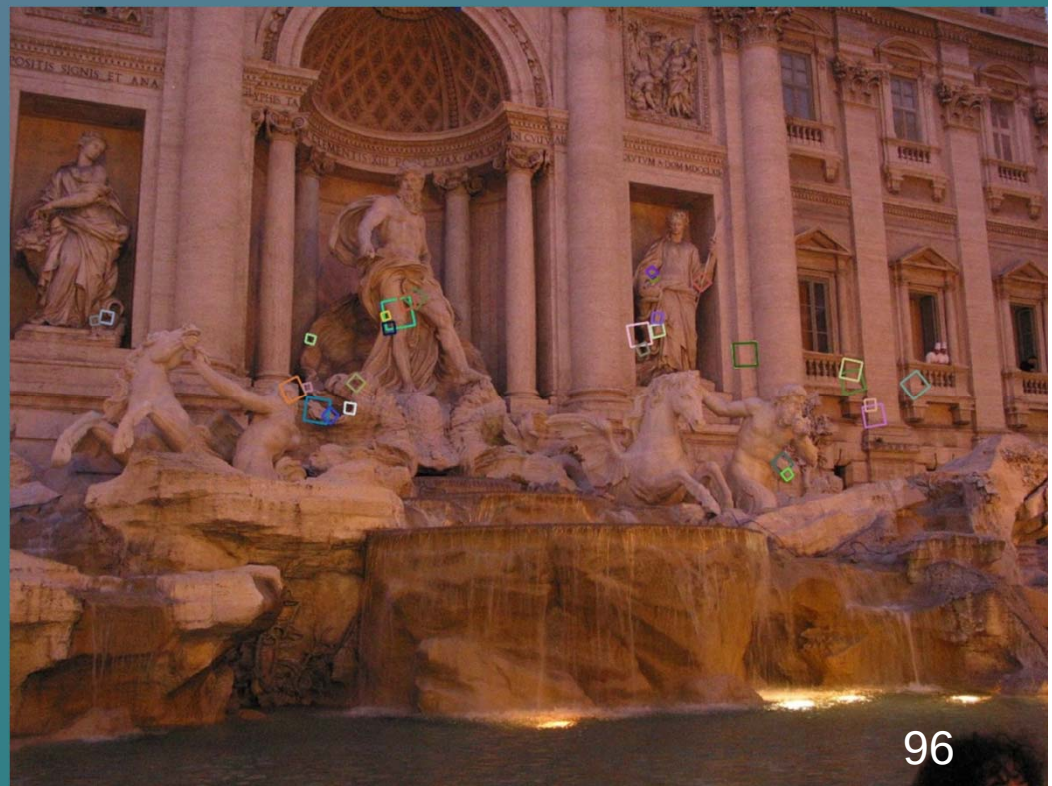


Adapted from slide by David Lowe

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

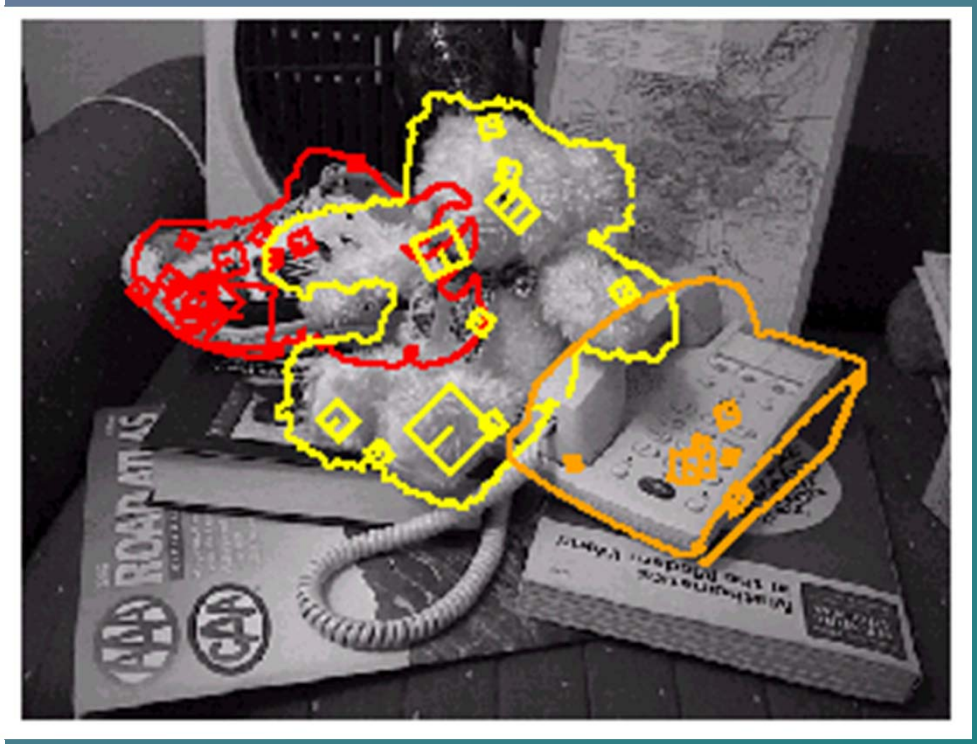


Lots of applications

Features are used for:

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

Object recognition



Images from David Lowe

Summary

- Things to take away from this unit
  - Brightness transformation
  - Edge detection by differentiation
  - Image gradients
  - Laplacian operator
    - Laplacian of Gaussian (LoG)
  - Canny edge detector (basic idea)
    - Effects of varying sigma parameter
  - Corner Detection
  - SIFT