# Computer Vision für Mensch-Maschine Schnittstellen

Vorlesung, WS 2012 / 2013

**Prof. Dr. Rainer Stiefelhagen**

Institut für Anthropomatik
Karlsruher Institut für Technologie - KIT

http://cvhci.anthropomatik.kit.edu/
rainer.stiefelhagen@kit.edu

# Basics:

# Image Processing

WS 2010/11

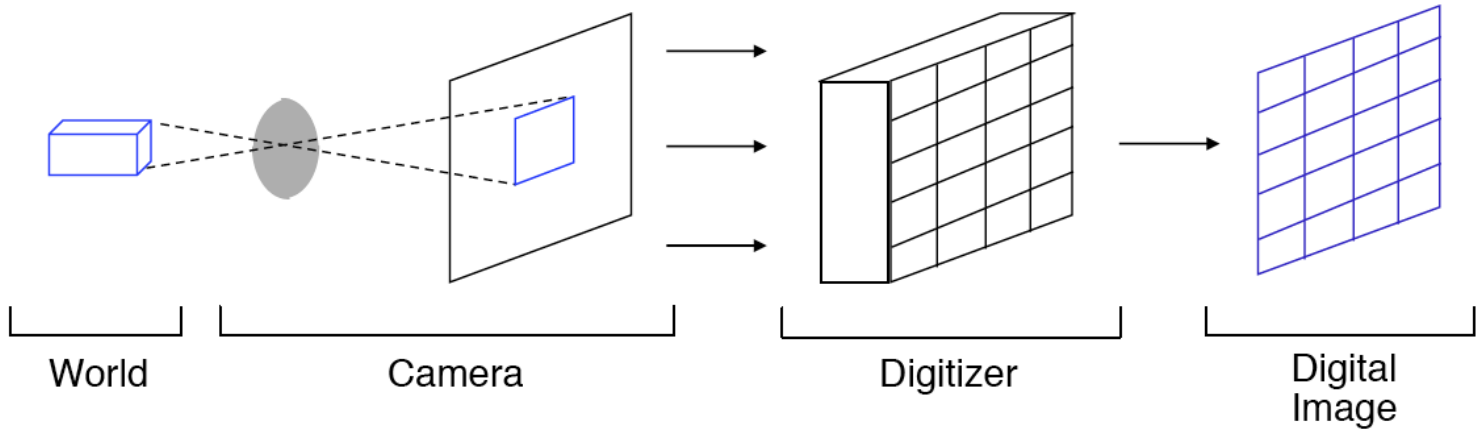
**Rainer Stiefelhagen**

***(Slides created by Dr. Edgar Seemann)***

# This Lecture

- Small intro
- Image Filtering
  - Linear Filtering
  - Convolution Kernels
  - Gaussian Filtering
  - Median Filter
- Edge Detection

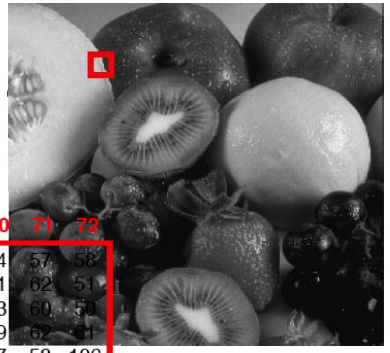
# Imaging Process




- Real-World objects are projected onto an image plane
- Computer Vision tries to ‘reverse’ the imaging process
  - i.e. given the image information, infer the image contents
  - image understanding
  - pattern recognition

# Grayscale Image

- Computer Vision algorithms often operate on grayscale images only
- An image then corresponds to an array of numbers (typically integer values between 0 and 255)



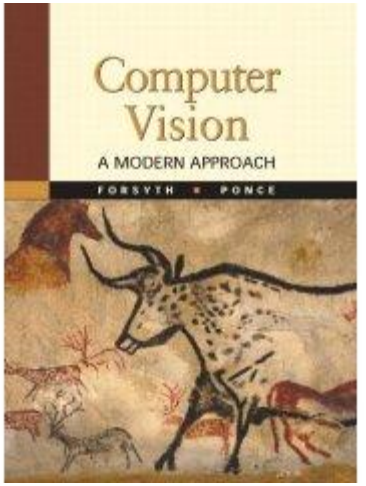
| | x = 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| y = 41 | 210 | 209 | 204 | 202 | 197 | 247 | 143 | 71 | 64 | 80 | 84 | 54 | 54 | 57 | 58 |
| 42 | 206 | 196 | 203 | 197 | 195 | 210 | 207 | 56 | 63 | 58 | 53 | 53 | 61 | 62 | 51 |
| 43 | 201 | 207 | 192 | 201 | 198 | 213 | 156 | 69 | 65 | 57 | 55 | 52 | 53 | 60 | 50 |
| 44 | 216 | 206 | 211 | 193 | 202 | 207 | 208 | 57 | 69 | 60 | 55 | 77 | 49 | 62 | 61 |
| 45 | 221 | 206 | 211 | 194 | 196 | 197 | 220 | 56 | 63 | 60 | 55 | 46 | 97 | 58 | 106 |
| 46 | 209 | 214 | 224 | 199 | 194 | 193 | 204 | 173 | 64 | 60 | 59 | 51 | 62 | 56 | 48 |
| 47 | 204 | 212 | 213 | 208 | 191 | 190 | 191 | 214 | 60 | 62 | 66 | 76 | 51 | 49 | 55 |
| 48 | 214 | 215 | 215 | 207 | 208 | 180 | 172 | 188 | 69 | 72 | 55 | 49 | 56 | 52 | 56 |
| 49 | 209 | 205 | 214 | 205 | 204 | 196 | 187 | 196 | 86 | 62 | 66 | 87 | 57 | 60 | 48 |
| 50 | 208 | 209 | 205 | 203 | 202 | 186 | 174 | 185 | 149 | 71 | 63 | 55 | 55 | 45 | 56 |
| 51 | 207 | 210 | 211 | 199 | 217 | 194 | 183 | 177 | 209 | 90 | 62 | 64 | 52 | 93 | 52 |
| 52 | 208 | 205 | 209 | 209 | 197 | 194 | 183 | 187 | 187 | 239 | 58 | 68 | 61 | 51 | 56 |
| 53 | 204 | 206 | 203 | 209 | 195 | 203 | 188 | 185 | 183 | 221 | 75 | 61 | 58 | 60 | 60 |
| 54 | 200 | 203 | 199 | 236 | 188 | 197 | 183 | 190 | 183 | 196 | 122 | 63 | 58 | 64 | 66 |
| 55 | 205 | 210 | 202 | 203 | 199 | 197 | 196 | 181 | 173 | 186 | 105 | 62 | 57 | 64 | 63 |

# Computer Vision Goals

- How can we recognize e.g. faces from one or three arrays of these numbers?
- How can we perceive depth from this array of numbers?
- …
- The problem of ‘inverse graphics’

# Digital Image Processing

- Some Basics
  - (digital signal processing, FFT, …)
  - Image Filtering (based on Ponce&Forsyth, Bill Freeman, Bernt Schiele)
    - Book: “Computer Vision: A Modern Approach”



# Image Filtering

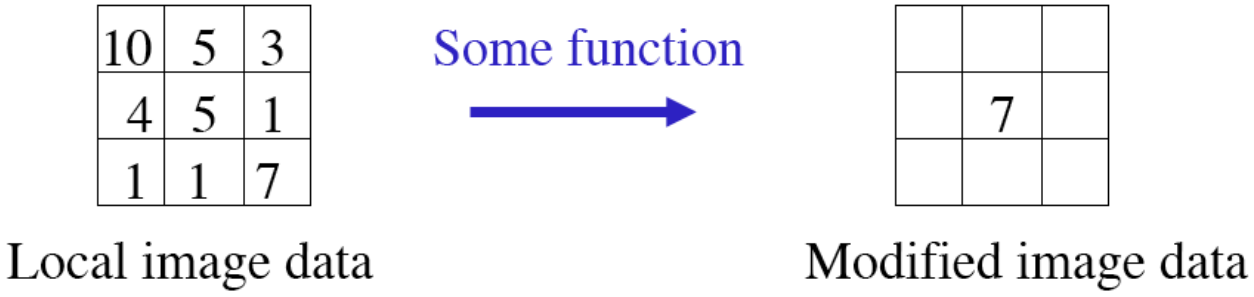
# Spatial Image Filtering

- One of the principal tools in computer vision
- Can e.g. be used for
  - Noise reduction
  - Sharpening
  - Edge detection

- Filtering can also be done in the frequency domain
  - FFT, DCT, …
  - Not discussed here

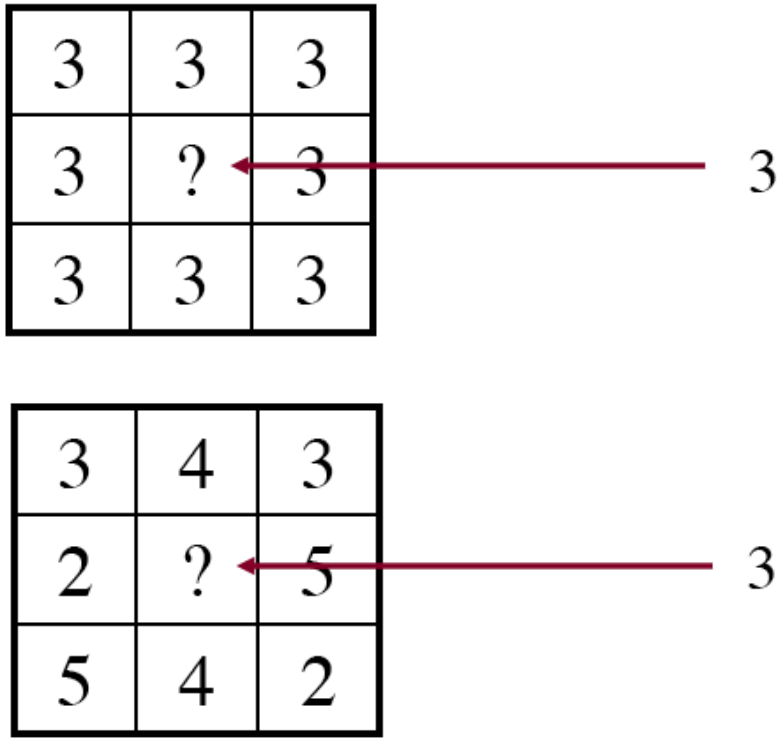
# Image Filtering

- Compute new image values based on some function and the original image data
- Typically we restrict the domain of this function to a local image area



# Image Filtering

- Some Examples:
  - what assumptions are you making to infer the center value?



# Image Filtering

- simplest case:
  - **Linear filtering:** replace each pixel by a linear combination of its neighbors

| | | |
|---|---|---|
| 10 | 5 | 3 |
| 4 | 5 | 1 |
| 1 | 1 | 7 |

Local image data

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0.5 | 0 |
| 0 | 1 | 0.5 |

kernel

| | | |
|---|---|---|
| | | |
| | 7 | |
| | | |

Modified image data

- i.e. $v_{0,0} = 0.5*v_{0,0} + 1*v_{0,1}+0.5*v_{1,1}$
- the prescription for the linear combination is called the ‘convolution kernel’

# Linear Systems

- Basic Properties:
  - Homogeneity $T[a\,X] = a\,T[X]$
  - Additivity $T[X_1 + X_2] = T[X_1] + T[X_2]$
  - Superposition $T[aX_1 + bX_2] = a\,T[X_1] + b\,T[X_2]$
  - Linear systems <=> superposition

- examples:
  - matrix operations (additions, multiplication)
  - convolutions

# Notation: 2D signals/images

- Image I:
  - Continuous: I(k,l)
  - Discrete: I[k,l] or $I_{k,l}$
- filter ‘kernel’:
  - Continuous: g(k,l)
  - Discrete: g[k,l]
- ‘filtered’ image:
  - Continuous: f(m,n)
  - Discrete: f[m,n]

# Convolution

- 1D convolution discrete

$$f[m] = I \otimes g = \sum_k I[m-k]g[k]$$

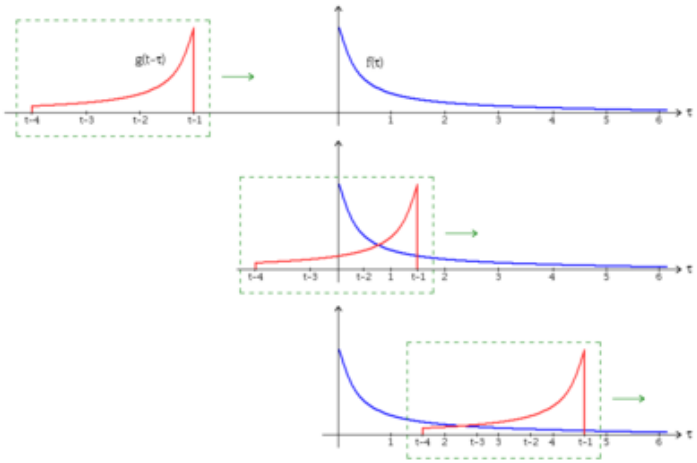
- 2D convolution discrete

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

- Properties:
  - Linearity
  - Commutativity
  - Associativity
  - Distributivity

# Convolution

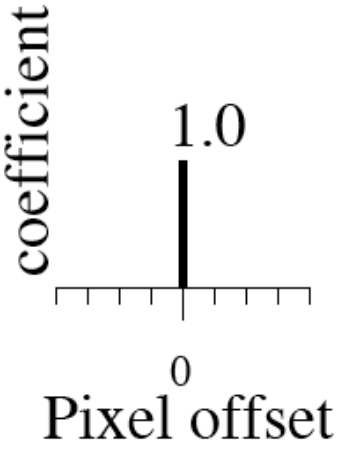
- Corresponds to computing at each point in the image
  - the sum of pixel values weighted by the entries of the convolution kernel
- The smaller the domain of the kernel, the more local the influence of the convolution



# Linear Filtering (warm-up slides)

- 1D convolution

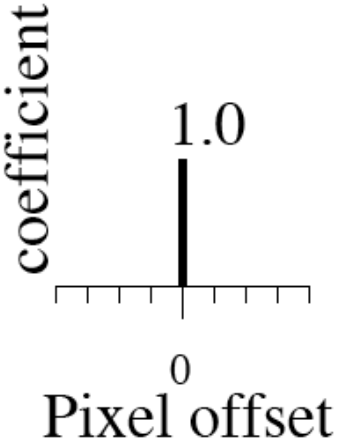
original




?

# Linear Filtering (warm-up slides)

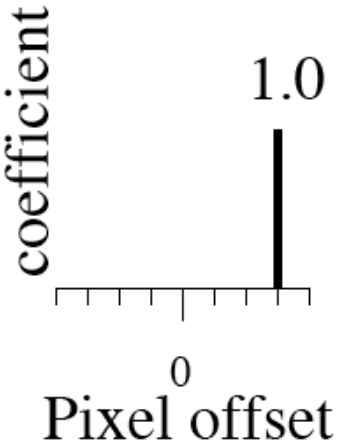
original




Filtered
(no change)

# Linear Filtering

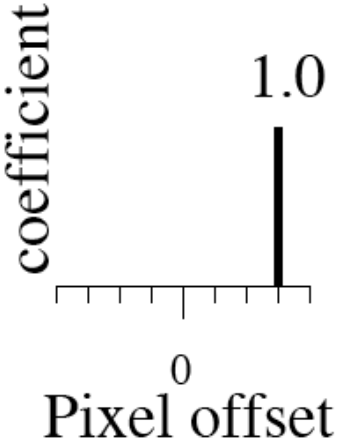
original

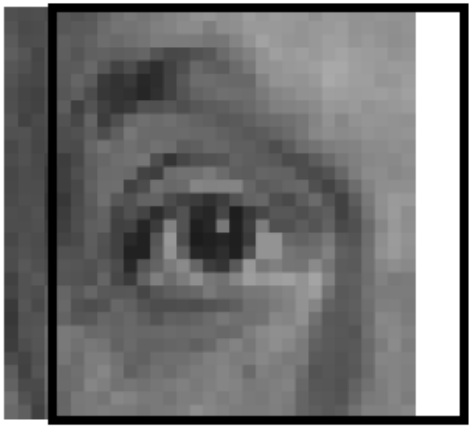



?

# Linear Filtering

original



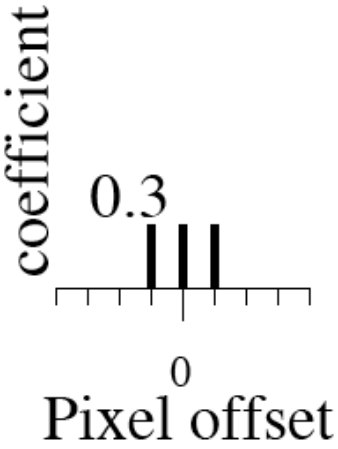



shifted

# Linear Filtering

- 1D convolution applied in both dimensions

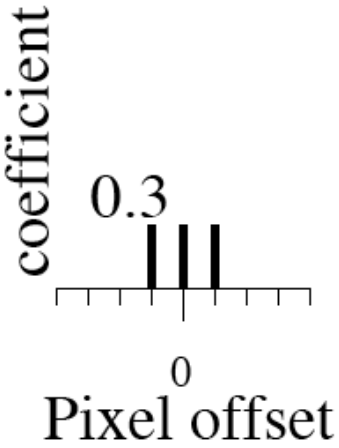
original




?

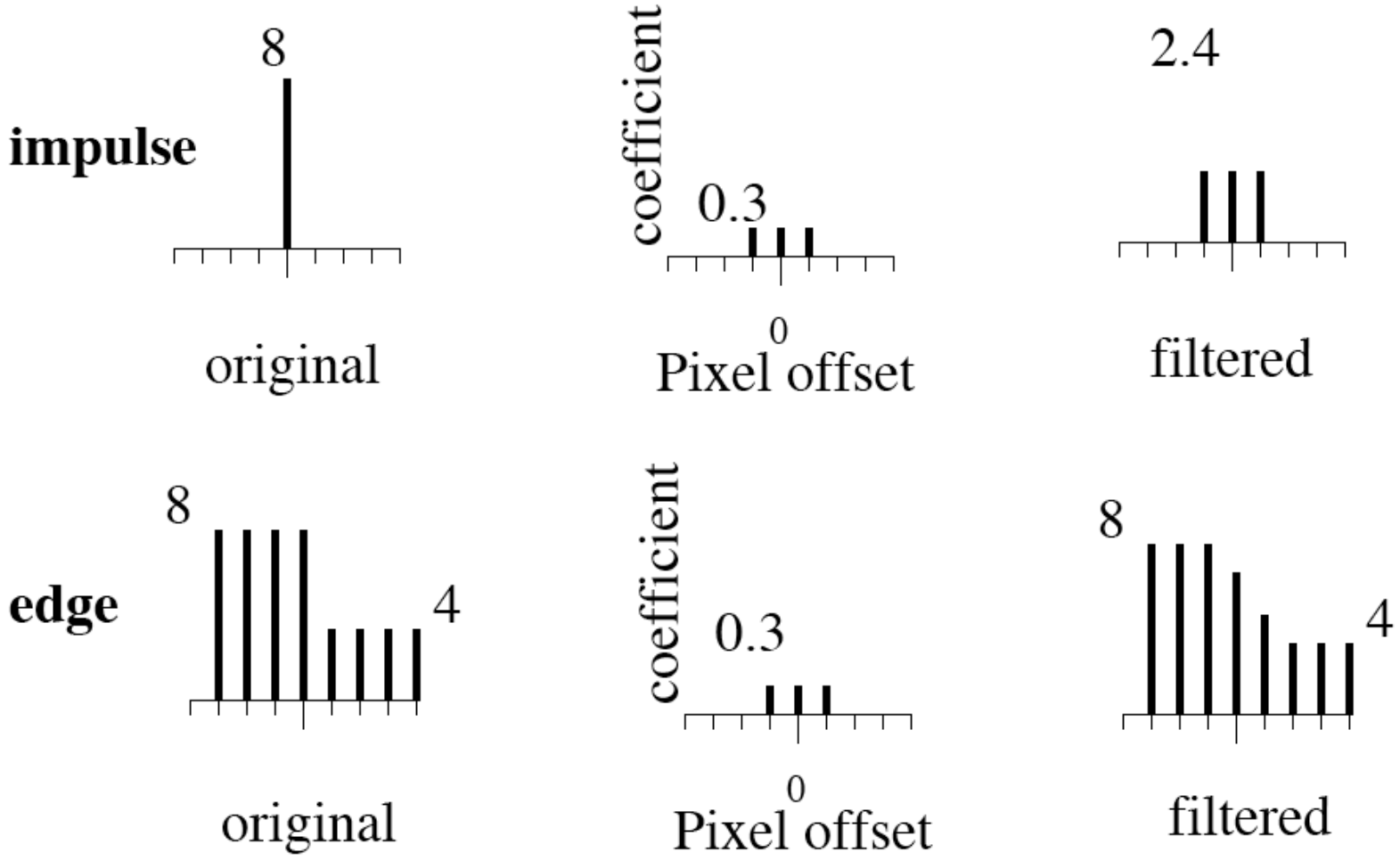
# Blurring

original



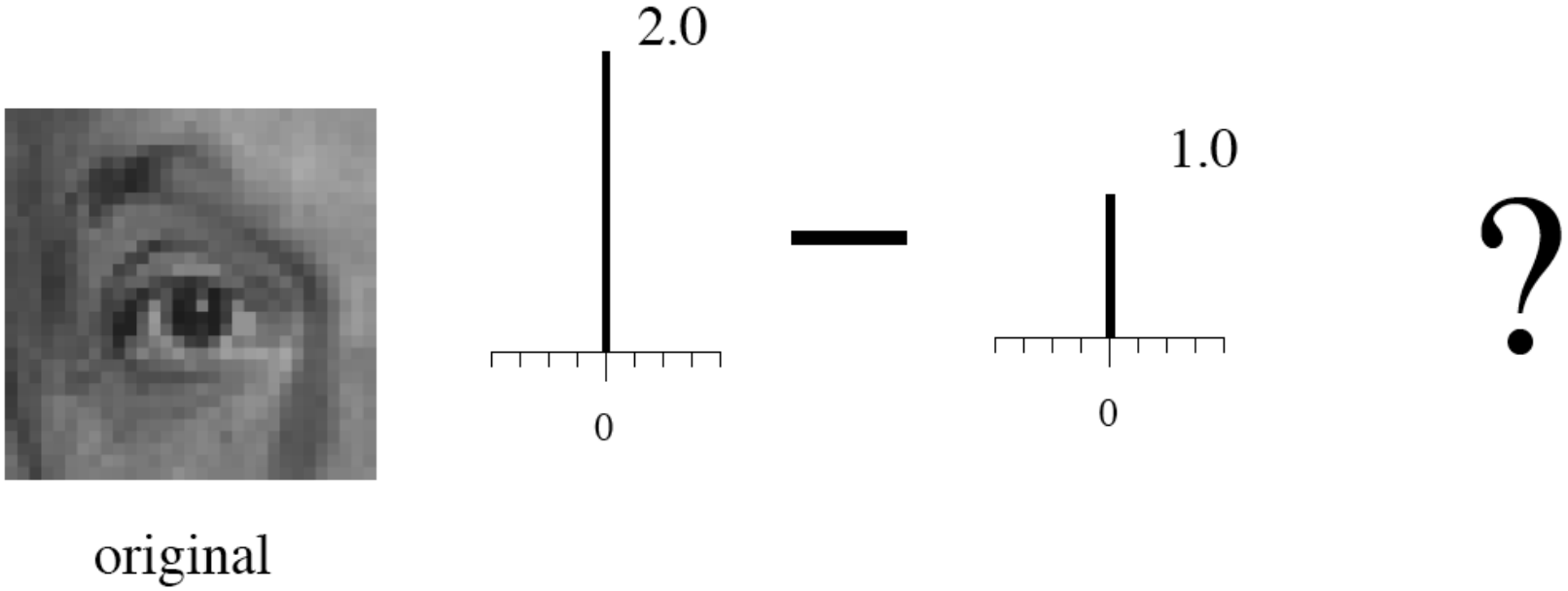

Blurred (filter applied in both dimensions).

# Blurring Examples

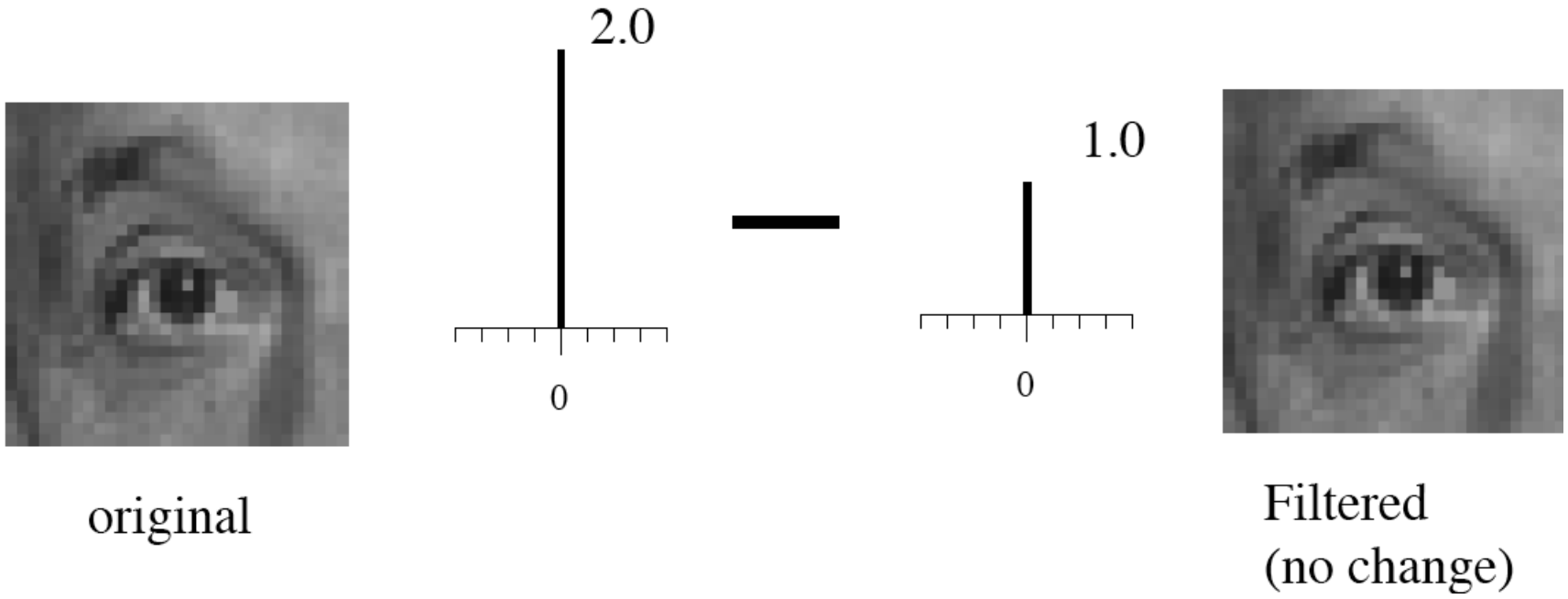


# Linear Filtering (warm-up slide)

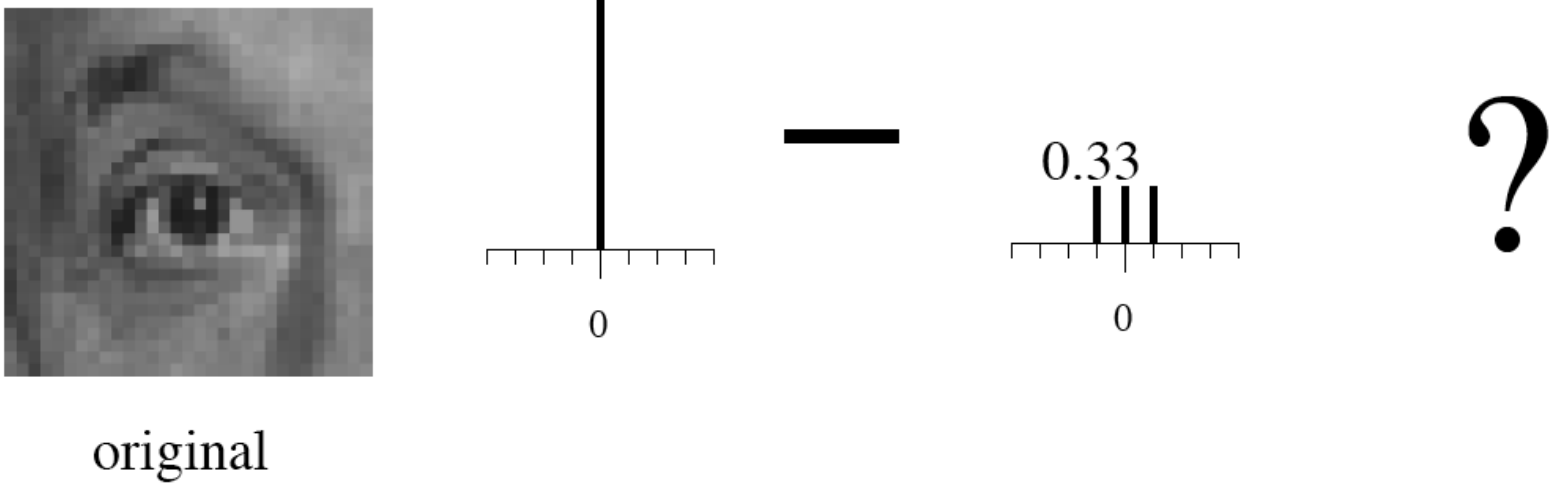
- Subtracting filter kernels



# Linear Filtering (warm-up slide)

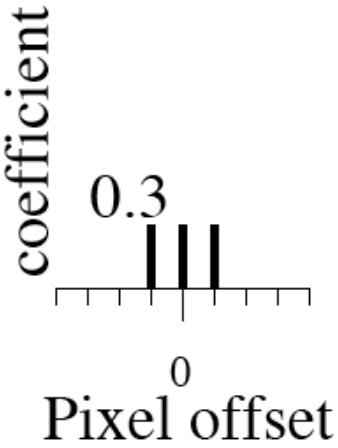


# Linear Filtering



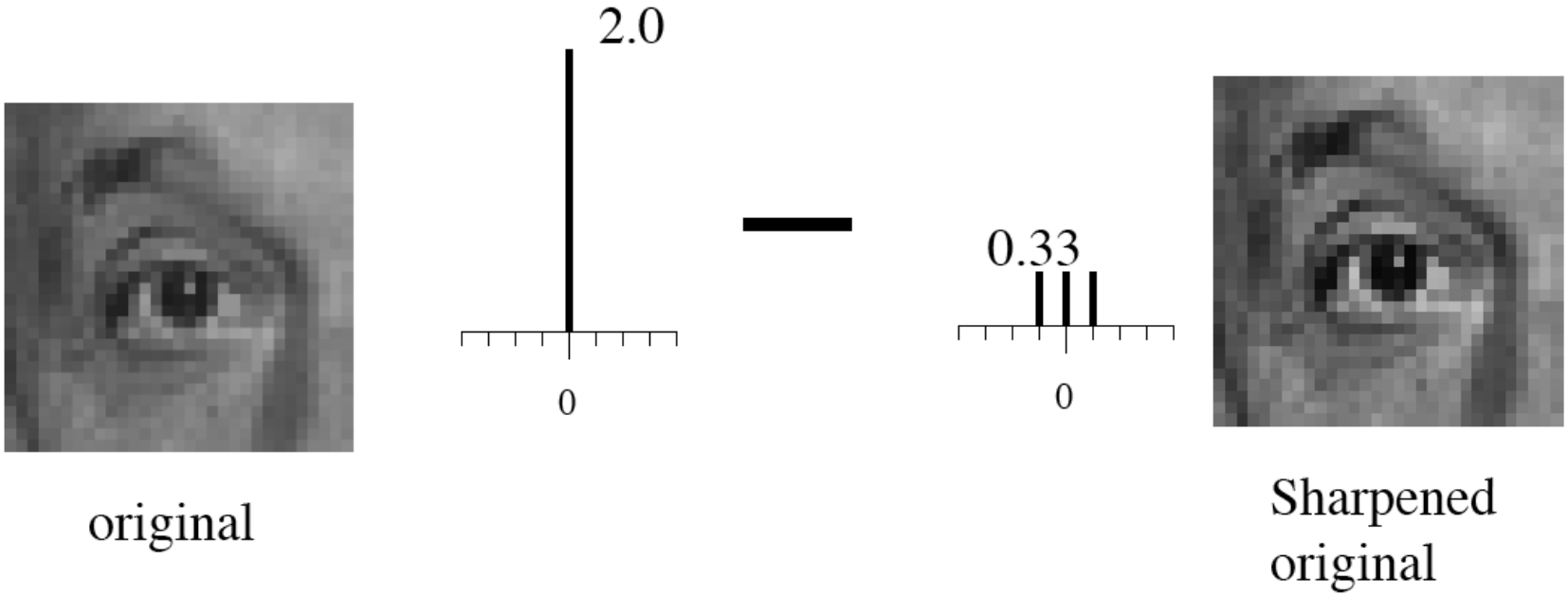
# (remember blurring)

original

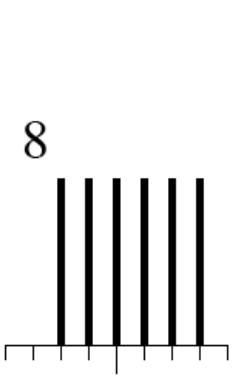



Blurred (filter applied in both dimensions).

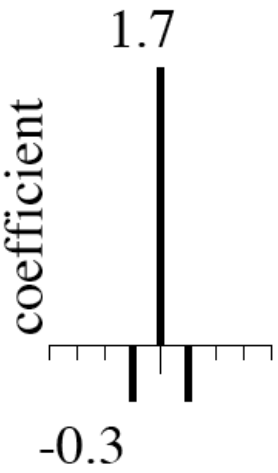
# Sharpening

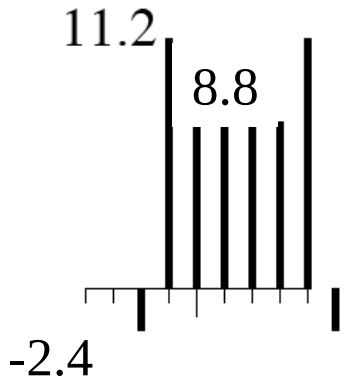


# Sharpening Example



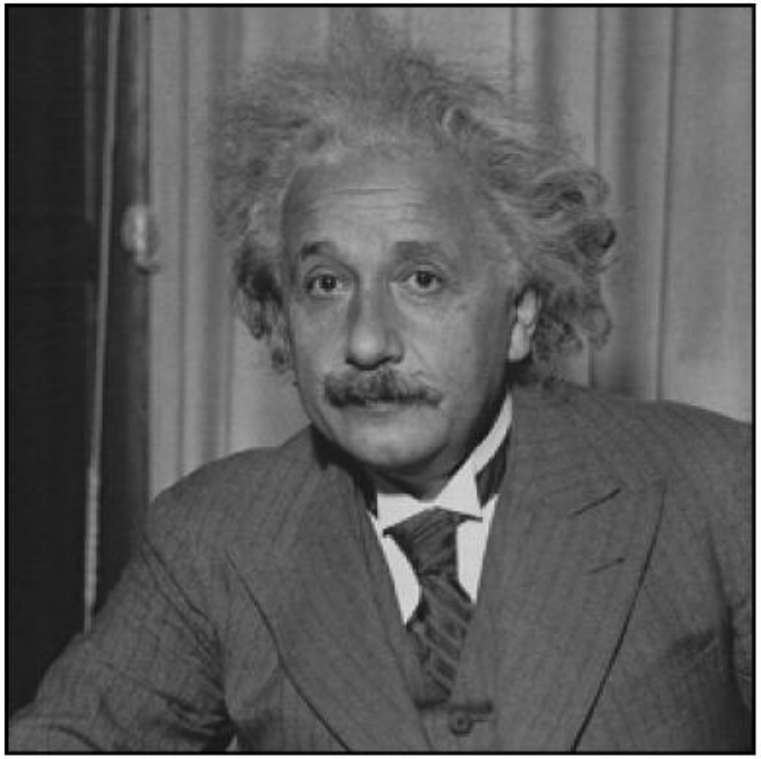

original



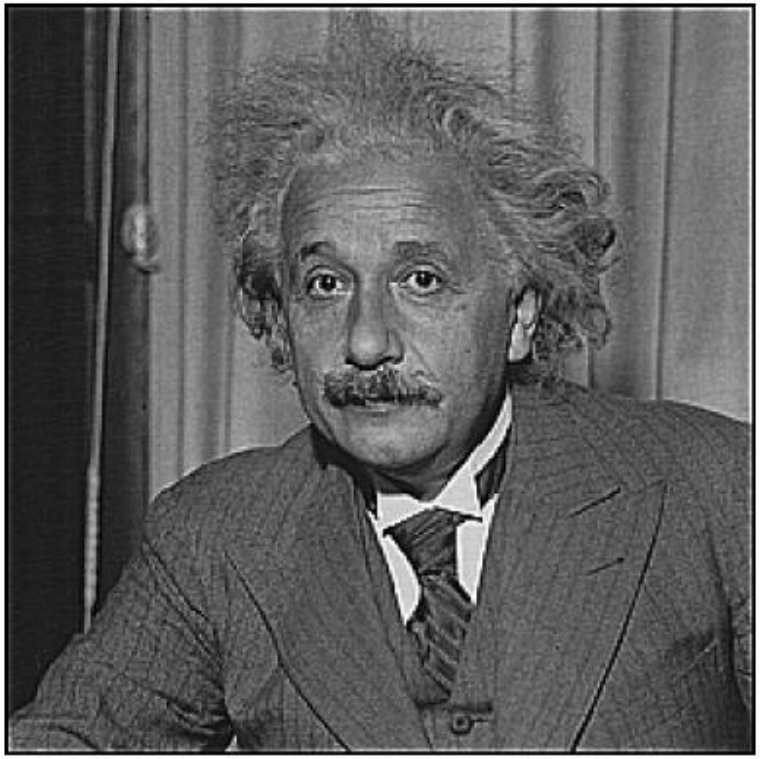




Sharpened (differences are accentuated; constant areas are left untouched).

# Sharpening



before



after

# Filtering to Reduce Noise

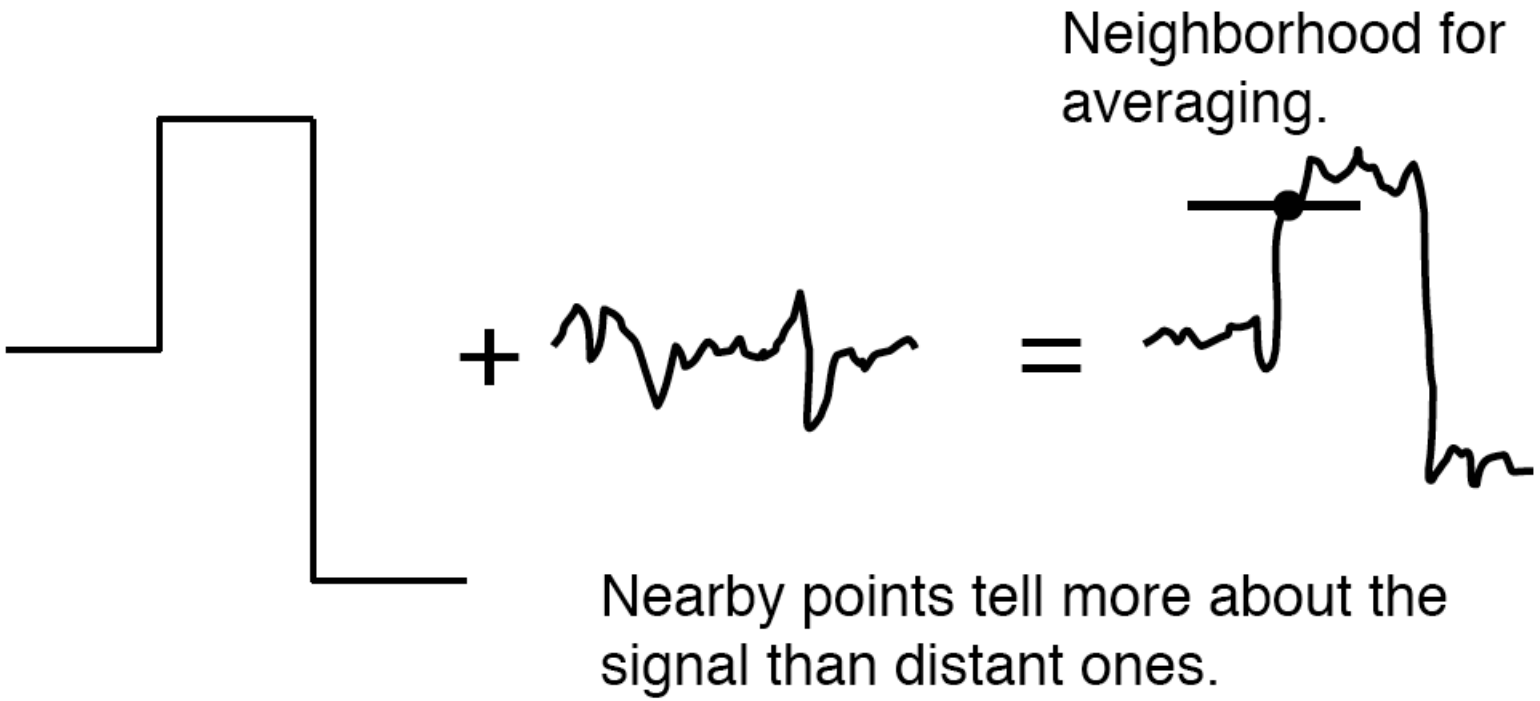
- 'Noise' is what we're not interested in
  - low-level noise: light fluctuations, sensor noise, quantization effects, finite precision, …
- Assumption:
  - the pixel's neighborhood contains information about its intensity
- averaging noise reduces its effect

# Model: Additive Noise

- Image I = Signal S + Noise N
  - I.e. noise does not depend on the signal

- we consider:
  - $I_i = s_i + n_i$ with $E(n_i) = 0$
  - $s_i$ deterministic
  - $n_i, n_j$ independent for $n_i \neq n_j$
  - $n_i, n_j$ i.i.d. (independent, identically distributed)

# Smoothing as Inference about the Signal



# Average Filter

- Mask with positive entries that sum to 1
- replaces each pixel with an average of its neighborhood
- if all weights are equal, it is called a BOX filter

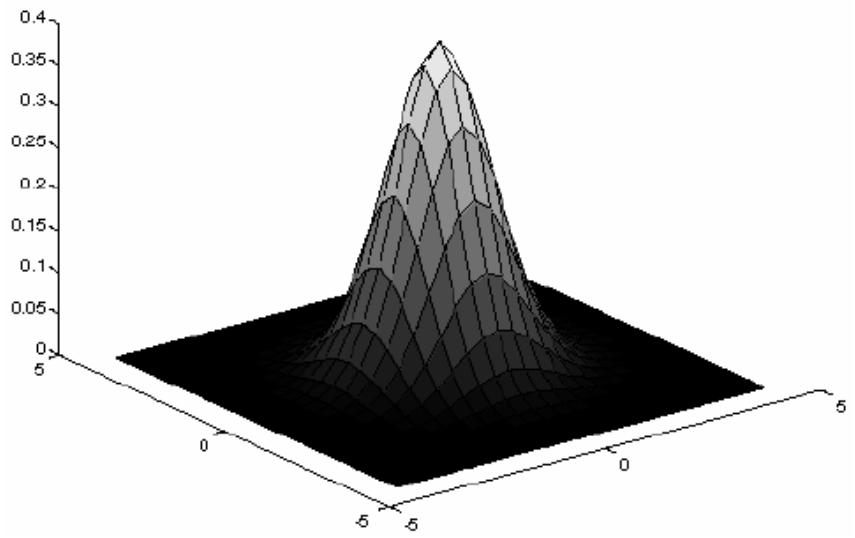
F

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

# Example: Smoothing by Averaging

# Gaussian Averaging

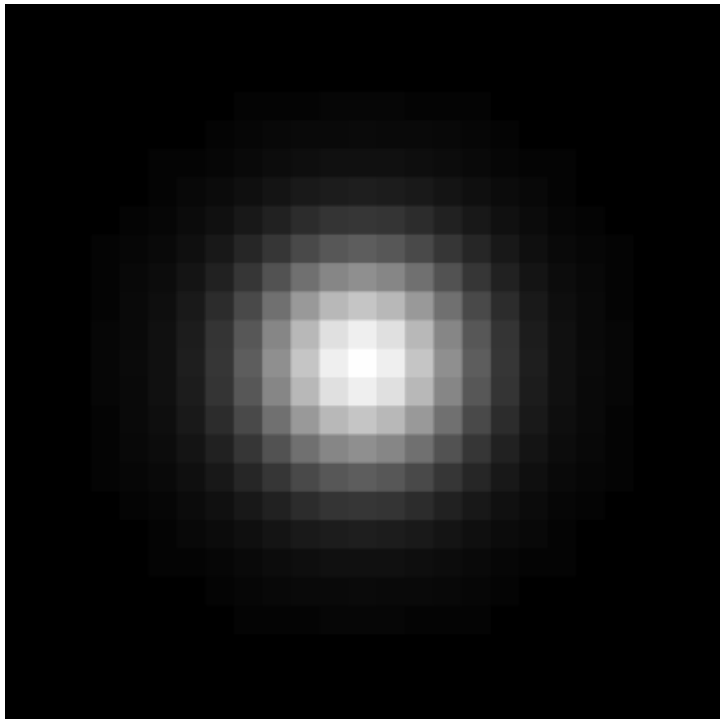
- Rotationally symmetric
- Weights nearby pixels more than distant ones
  - this makes sense as 'probabilistic' inference




- a Gaussian gives a good model of a fuzzy blob

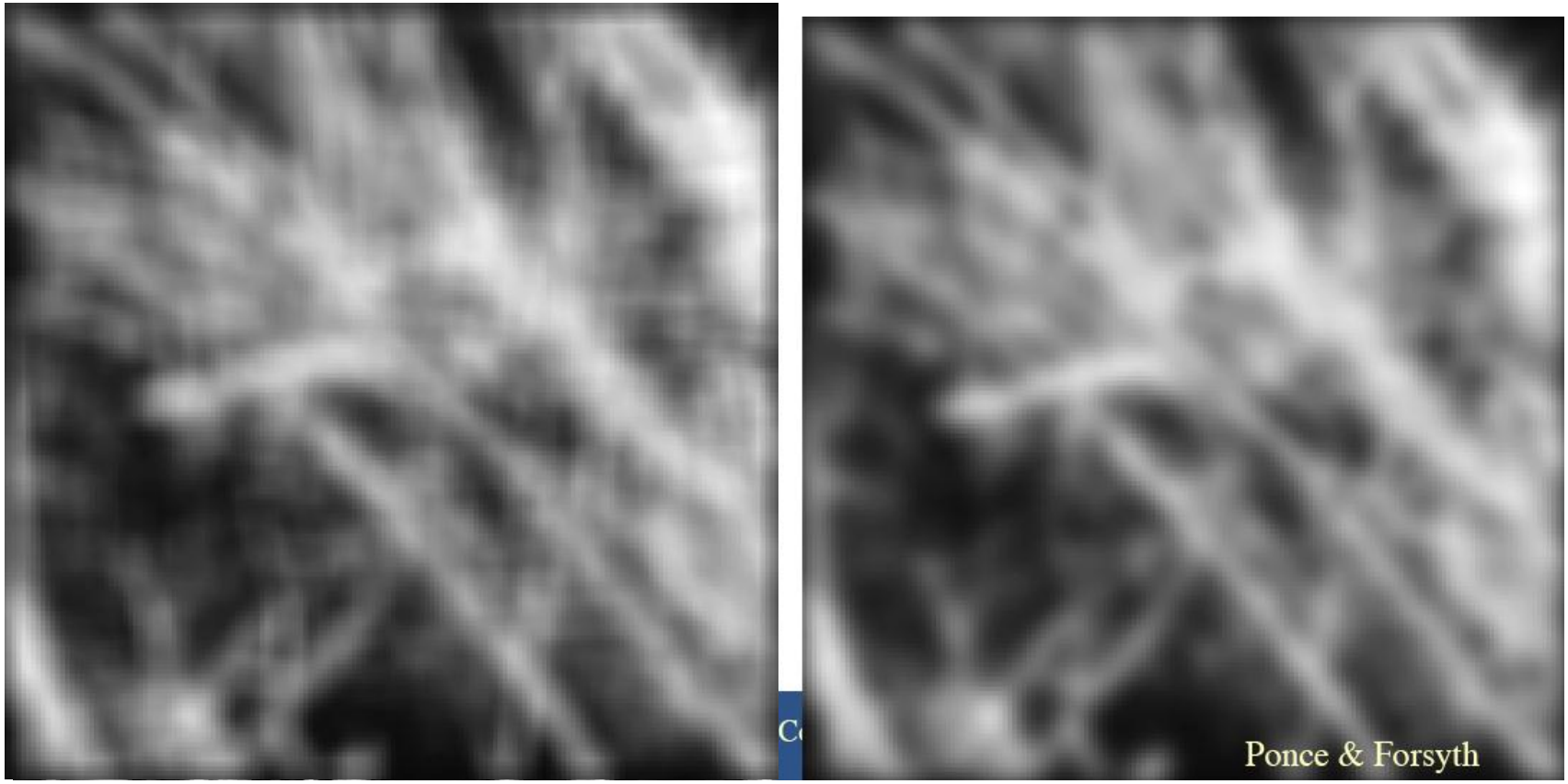
$$g(x,y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)$$

# An Isotropic Gaussian

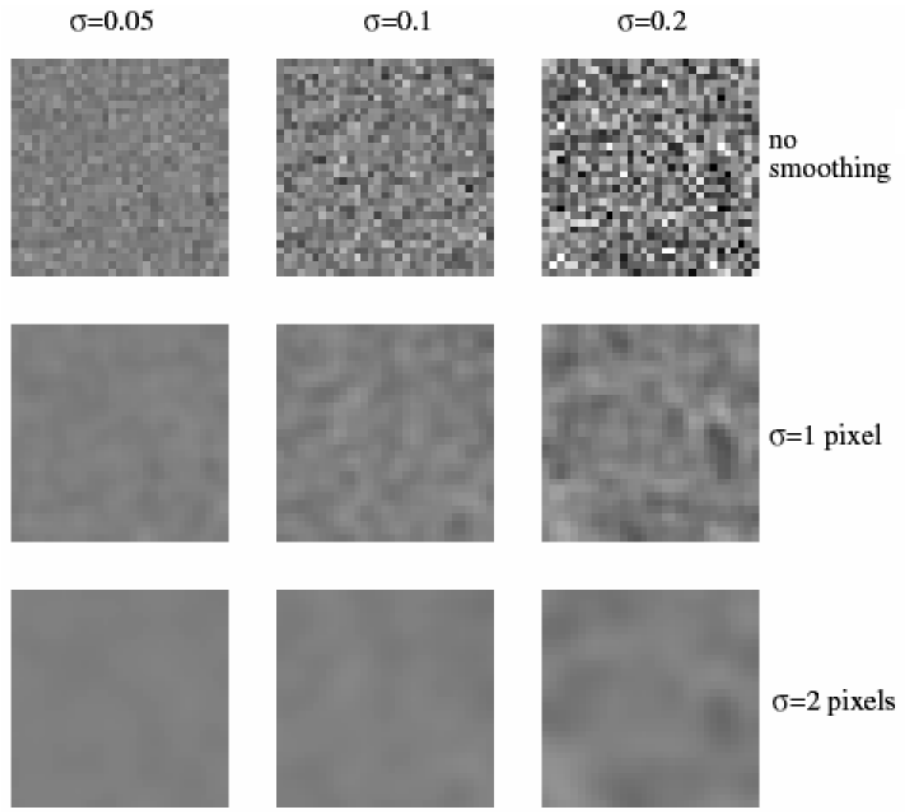


- the picture shows a smoothing kernel proportional to a gaussian distribution

# Smoothing with a Gaussian



# Smoothing with a Gaussian




- the effects of smoothing:
  - each row shows smoothing with Gaussians of different width
  - each column shows realizations of an image of Gaussian noise

# Efficient Implementation

- Both, the BOX filter and the Gaussian filter are separable:
  - first convolve each row with a 1D filter
  - then convolve each column with a 1D filter

- remember:
  - convolution is linear - associative and commutative

$$f_x \otimes f_y \otimes I = f_x \otimes \left( f_y \otimes I \right) = \left( f_x \otimes f_y \right) \otimes I$$

# Separable Gaussian

- Gaussian in x-direction

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-x^2/(2\sigma^2))$$

- Gaussian in y-direction

$$g(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-y^2/(2\sigma^2))$$

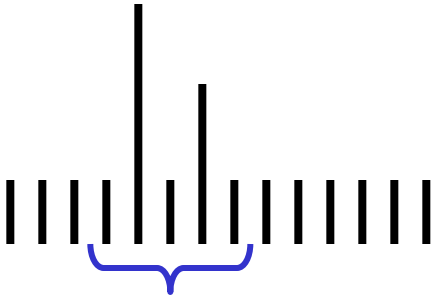
- Gaussian in both directions

$$g(x,y) = \frac{1}{2\pi\sigma^2} \exp(-(x^2+y^2)/(2\sigma^2))$$

# Median filter

Replace each pixel by the median over N pixels (5 pixels, for these examples).
Generalizes to “rank order” filters.

In:



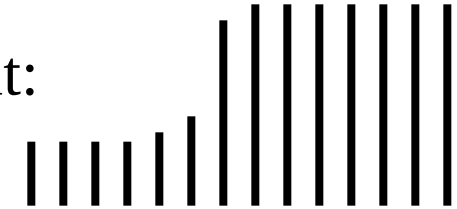
5-pixel neighborhood

Out:

Spike noise is removed

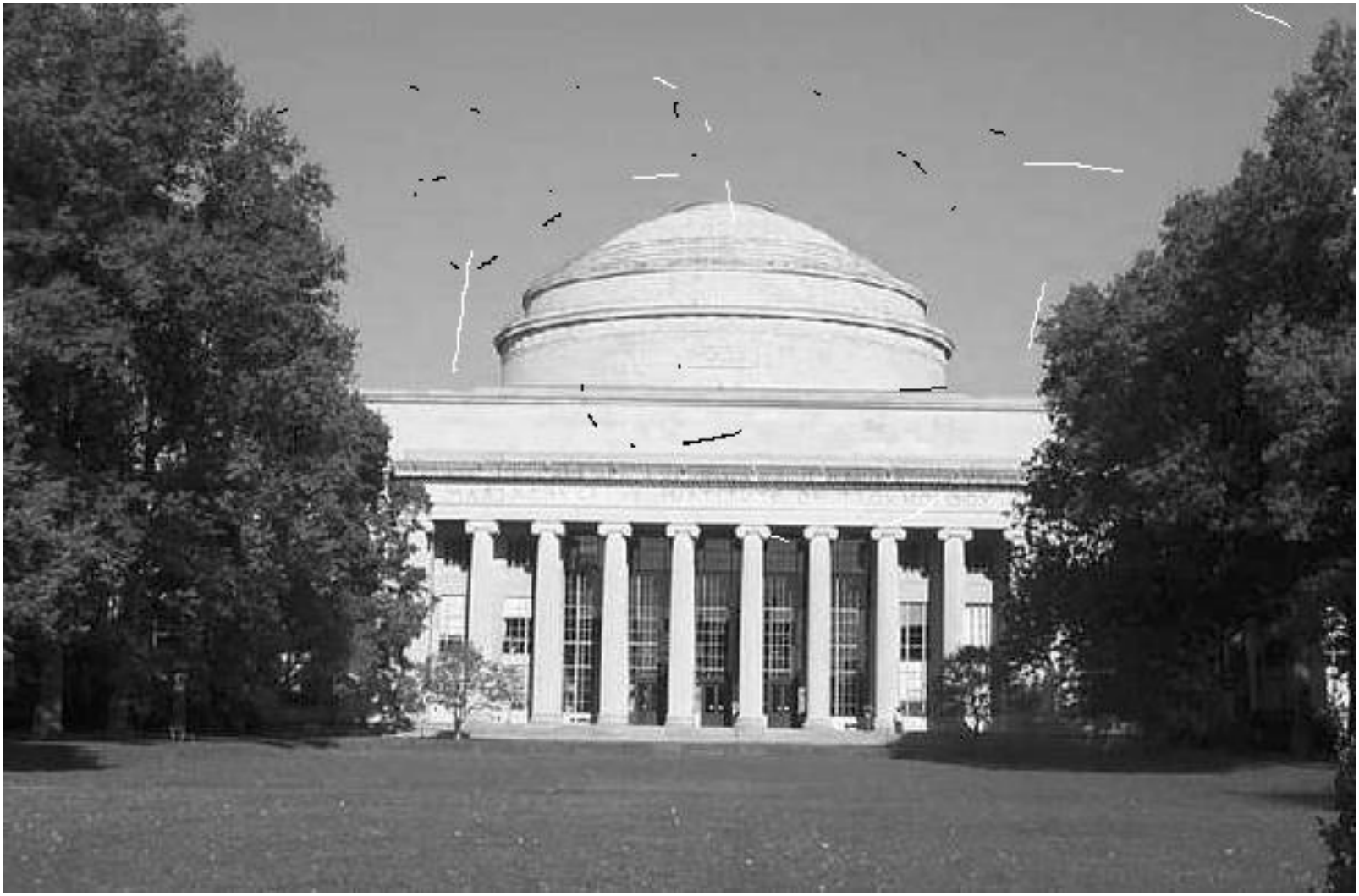
In:

Out:



Monotonic edges remain unchanged

# Degraded image

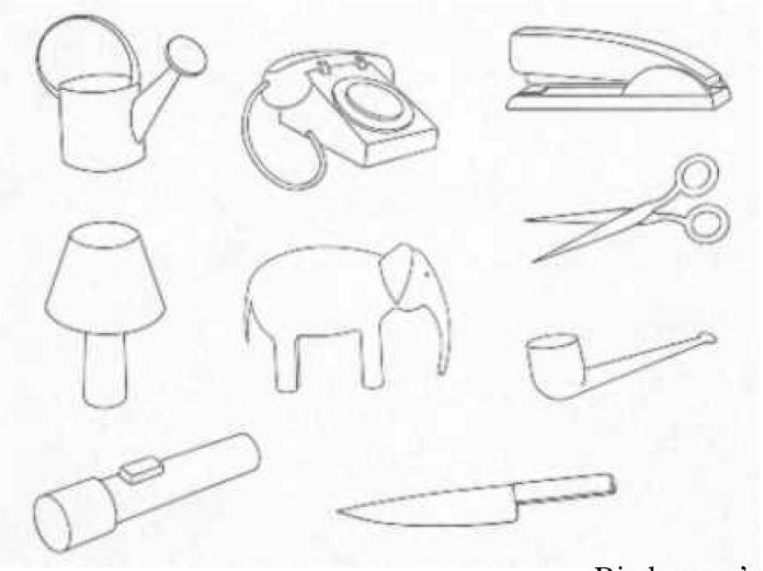


# Radius 1 median filter

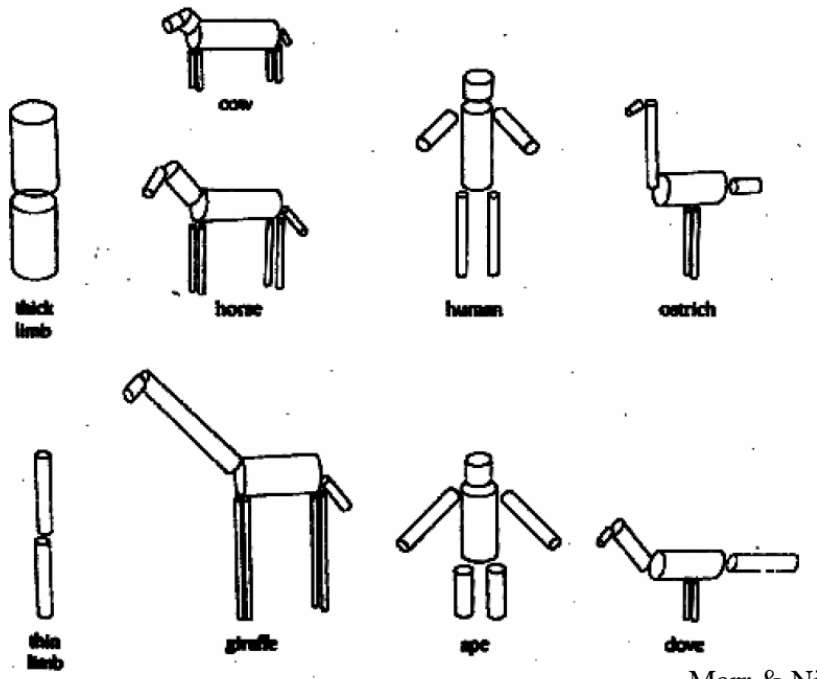
# Radius 2 median filter

# Edges

# Edges for Recognition



Biederman's

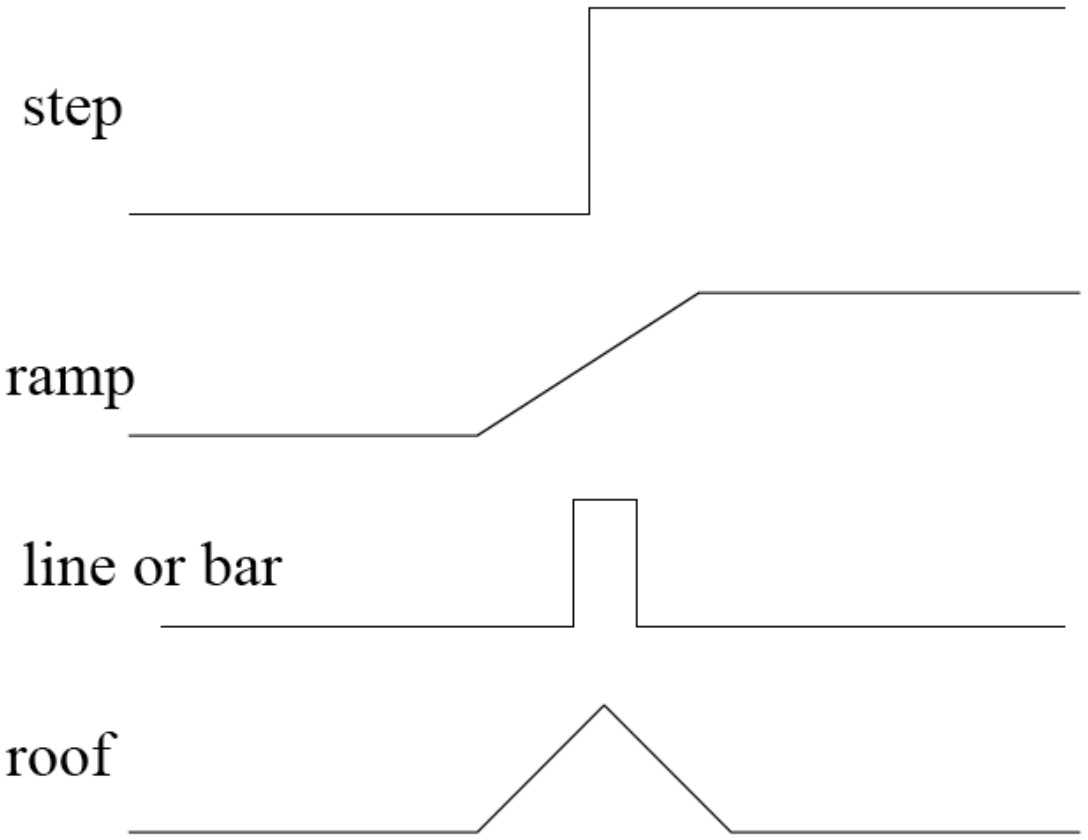



Marr & Nishihara

- possible approach: if line drawings are easy to recognize - then we should find lines and edges first

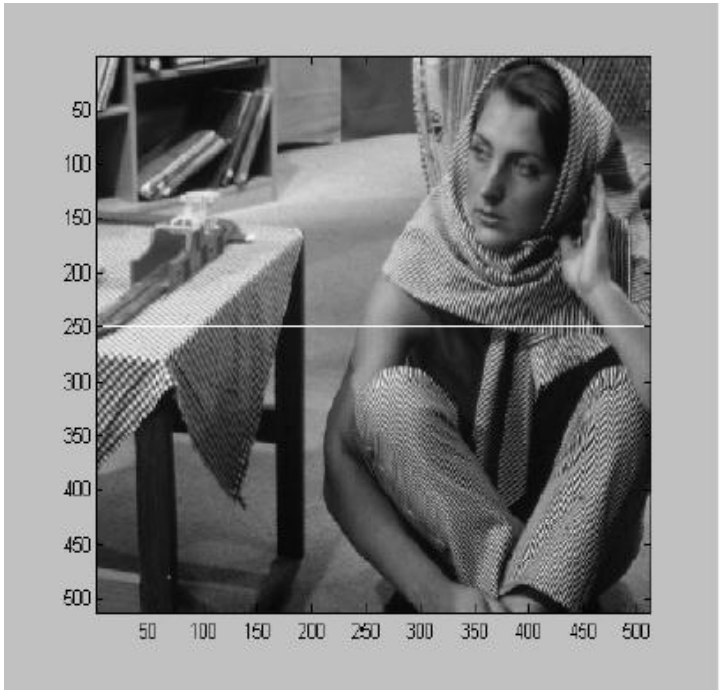
# What are ‘edges’ (1D)

- Idealized:

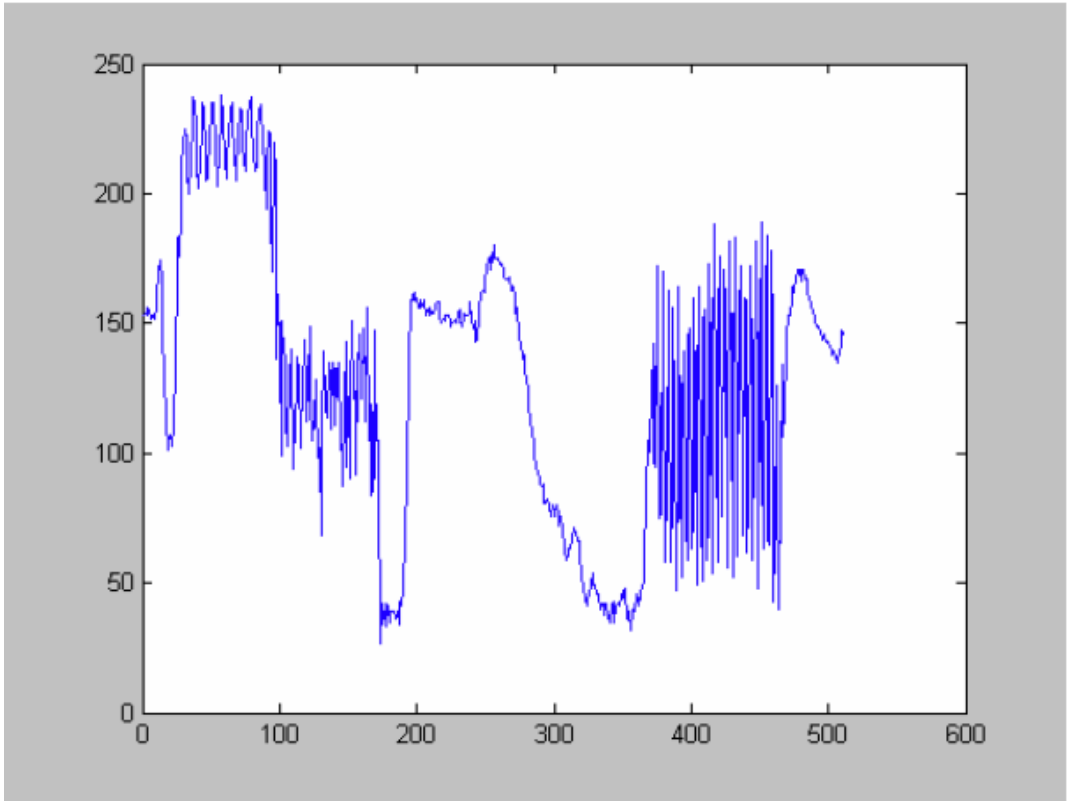


# Actual 1D profile

Barbara image:

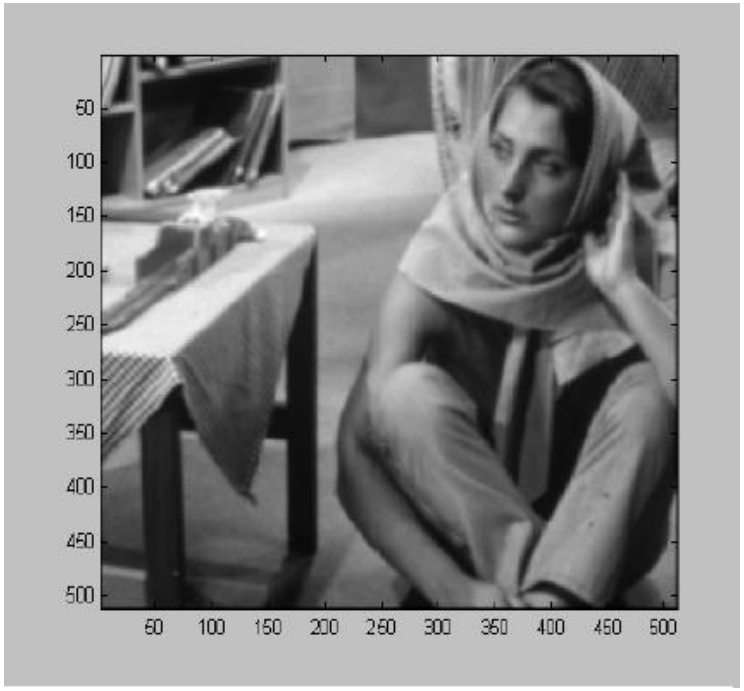



- line 250 of Barbara image

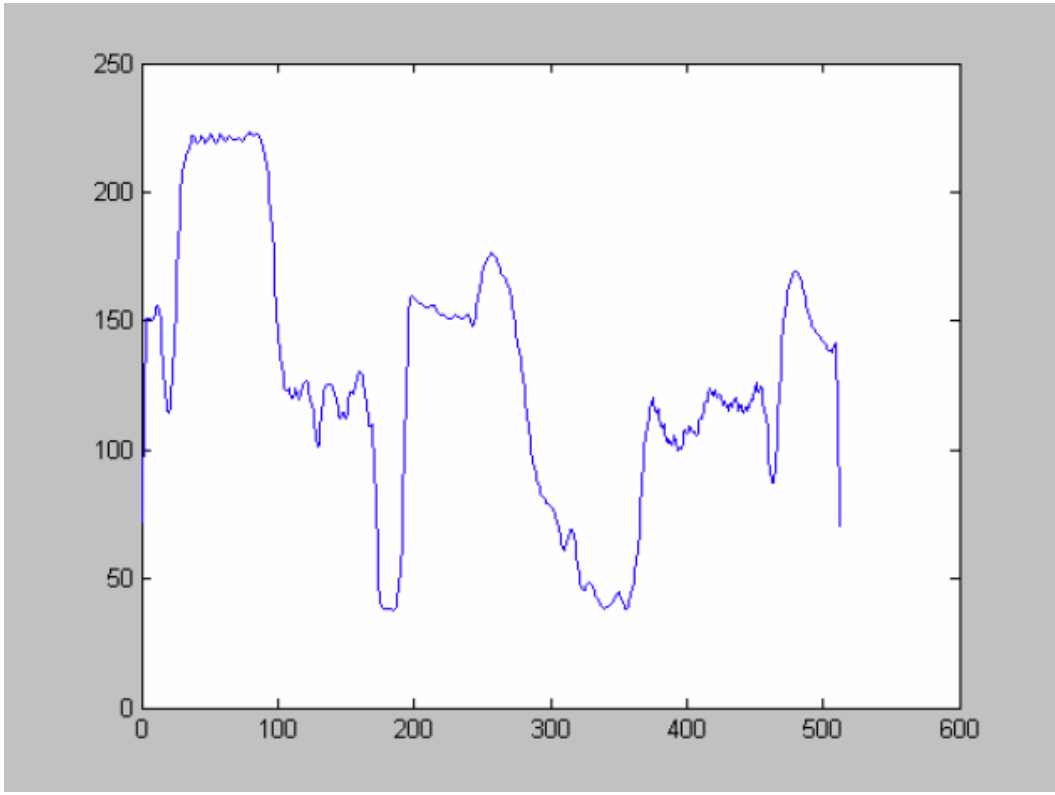


# smoothed with a Gaussian

- Barbara image:

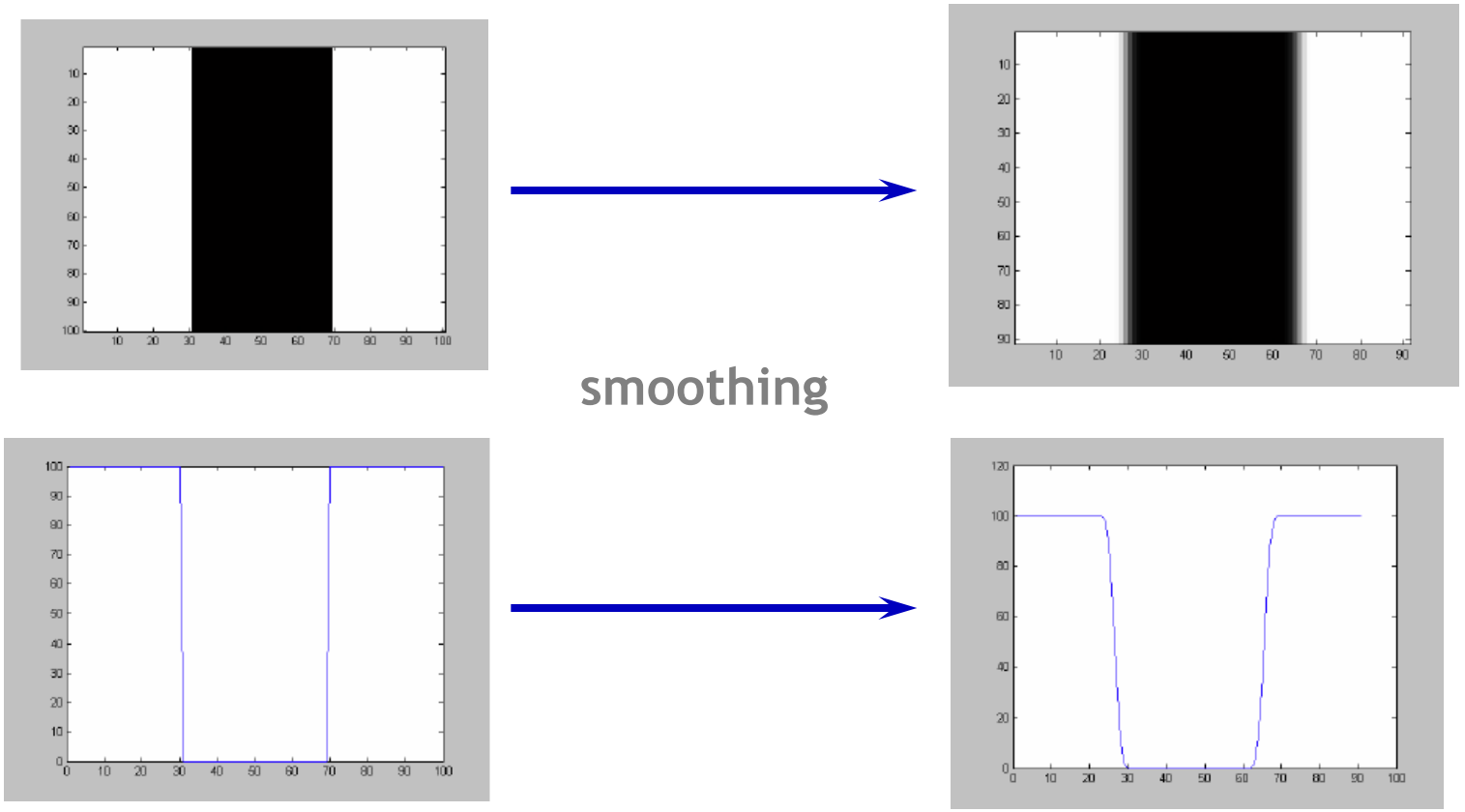



- line 250 of Barbara image

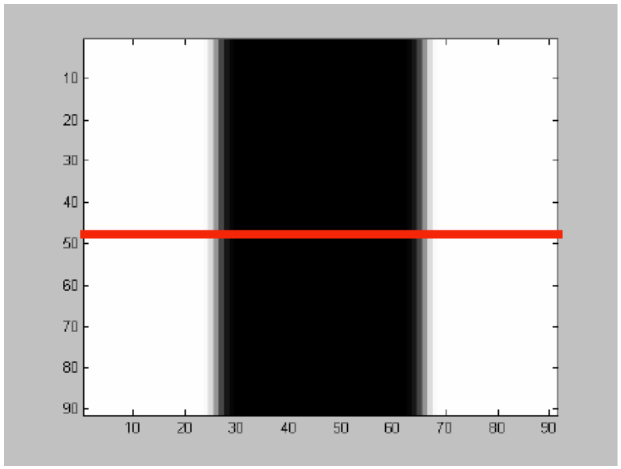


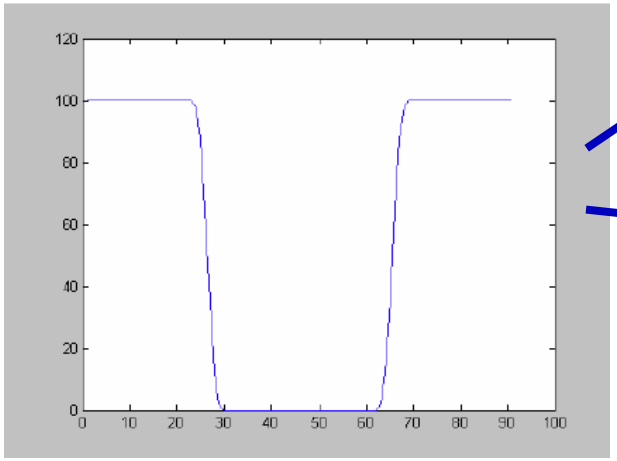
# Edges

- correspond to fast changes
- where the magnitude of the derivative is large

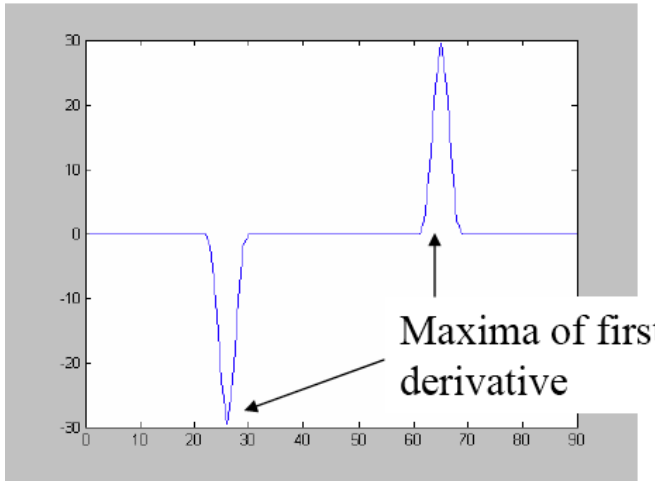


# Edges & Derivatives…

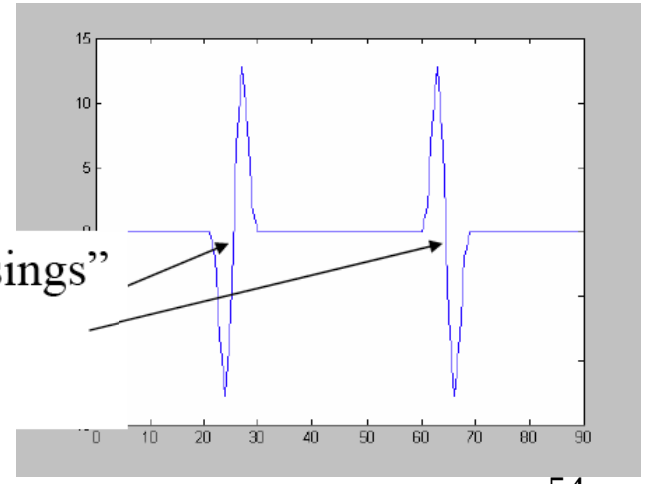




1st derivative




2nd derivative

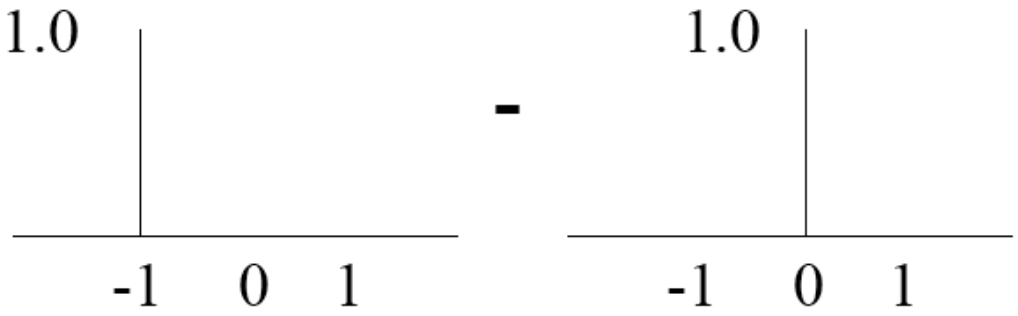


# Compute Derivatives

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx f(x+1) - f(x)$$

- we can implement this as a linear filter:
  - $[-1 \quad 1]$
  - or symmetric

$$[-1 \quad 0 \quad 1]$$



# compute 2nd order derivatives

- 1st derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx f(x+1) - f(x)$$

- 2nd derivative:

$$f''(x) = \lim_{h \to 0} \frac{f'(x+h) - f'(x)}{h} \approx f'(x+1) - f'(x)$$

$$= f(x+2) - 2f(x+1) + f(x)$$

- mask for
  - 1st derivative:

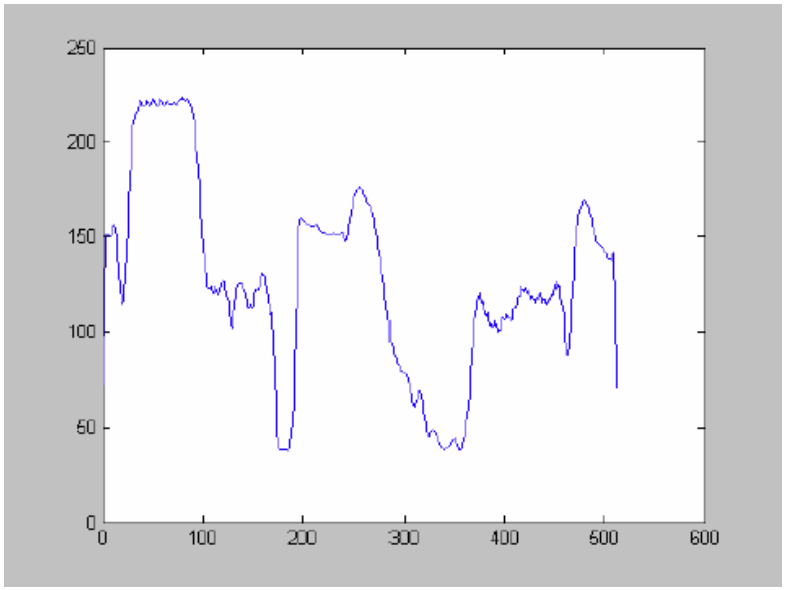
$$\begin{bmatrix} -1 & 1 \end{bmatrix}$$

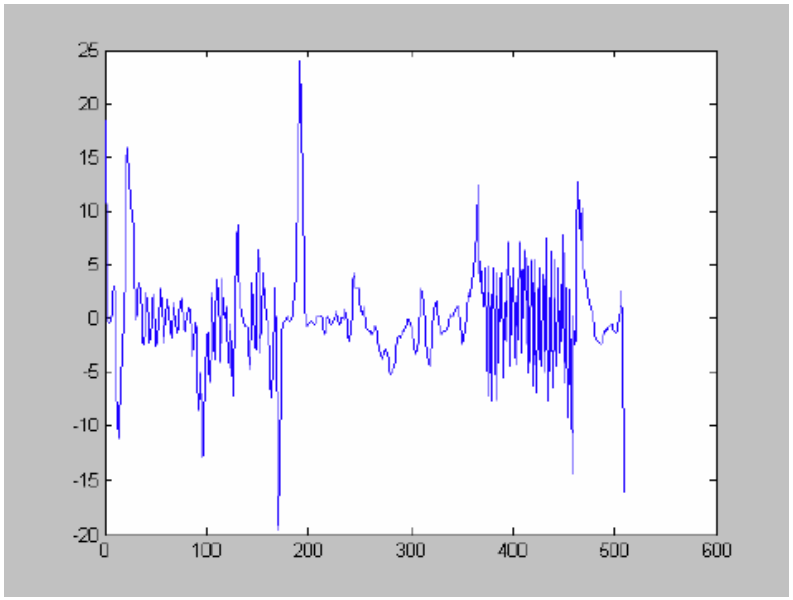
  2nd derivative:

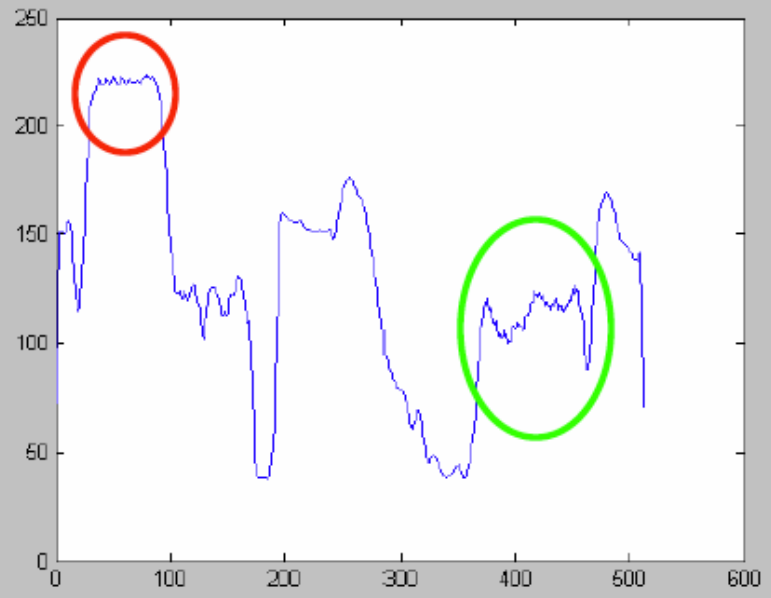
$$\begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$$


# 1D Barbara signal

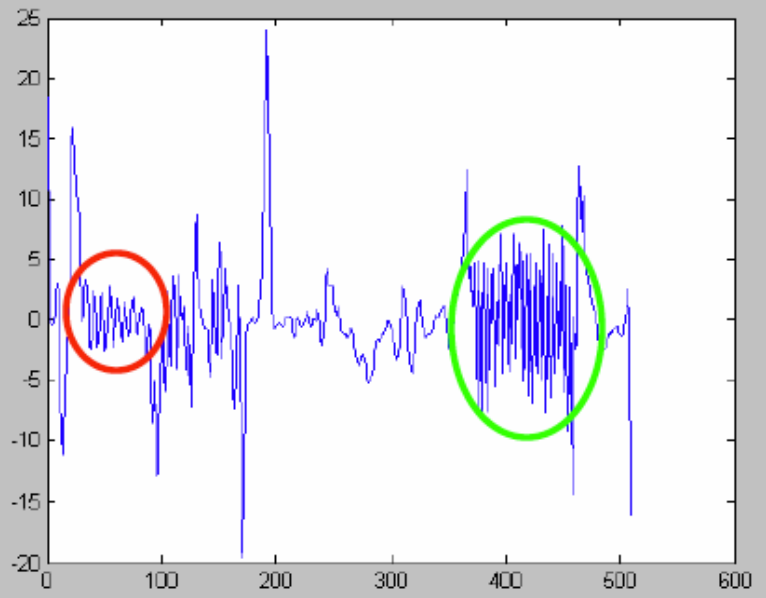
- 1st derivative:




Smoothed Signal




First Derivative

# implementing 1D edge detection

- algorithmically:
  - find peak in the derivative
  - but
    - should be a local maxima
    - should be ‘sufficiently’ large

# Going 2D

- Two approaches

1. Explicitely search for points, where the magnitude of the gradient is extremal

2. Look, where 2D second derivative vanishes
   - (→ Laplacian)

# Going 2D - Partial Derivatives

- partial derivatives
  - in x direction:

$$\frac{\partial}{\partial x} I(x,y) = I_x \approx I \otimes D_x$$

  - often approximated with simple filters (finite differences):

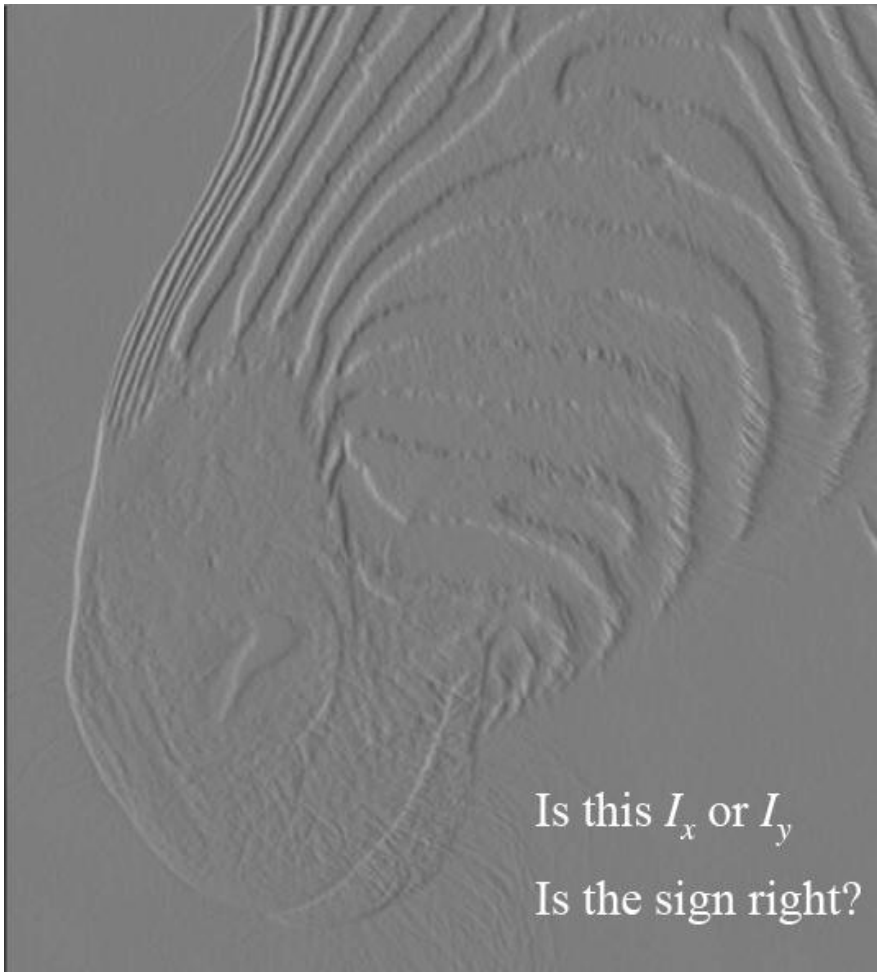
$$D_x = \frac{1}{3}\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

  - in y direction:

$$\frac{\partial}{\partial y} I(x,y) = I_y \approx I \otimes D_y$$

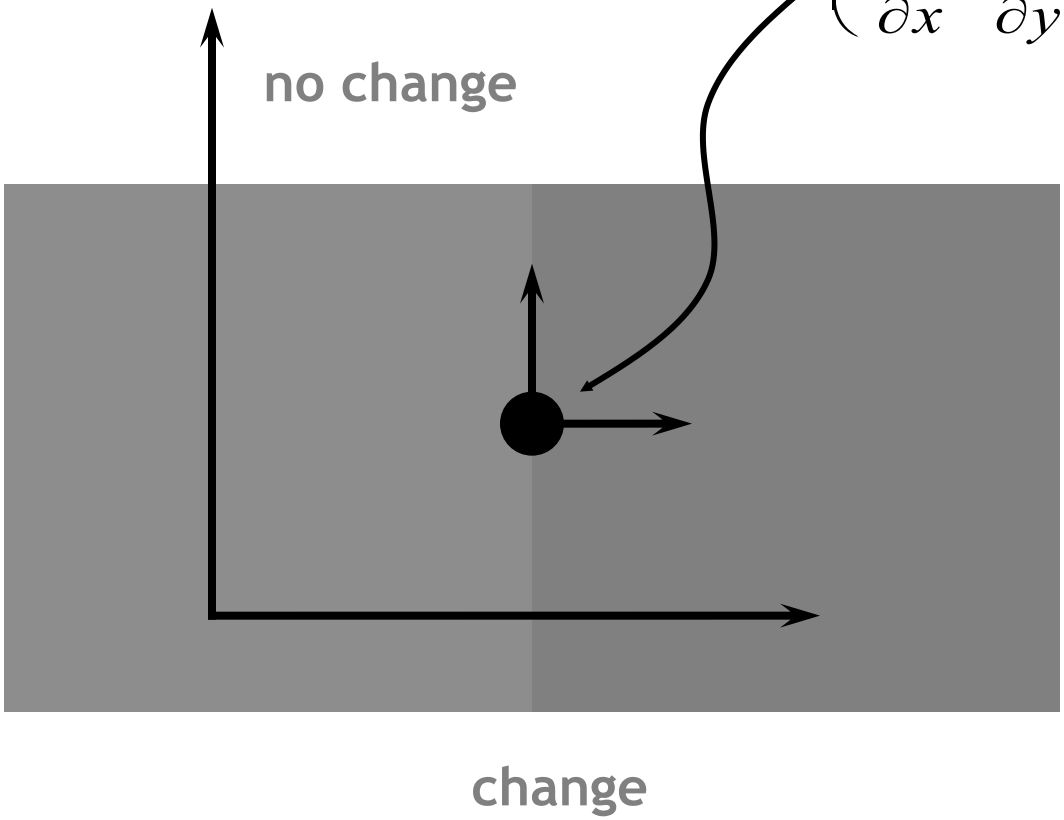
$$D_y = \frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

# Finite Differences

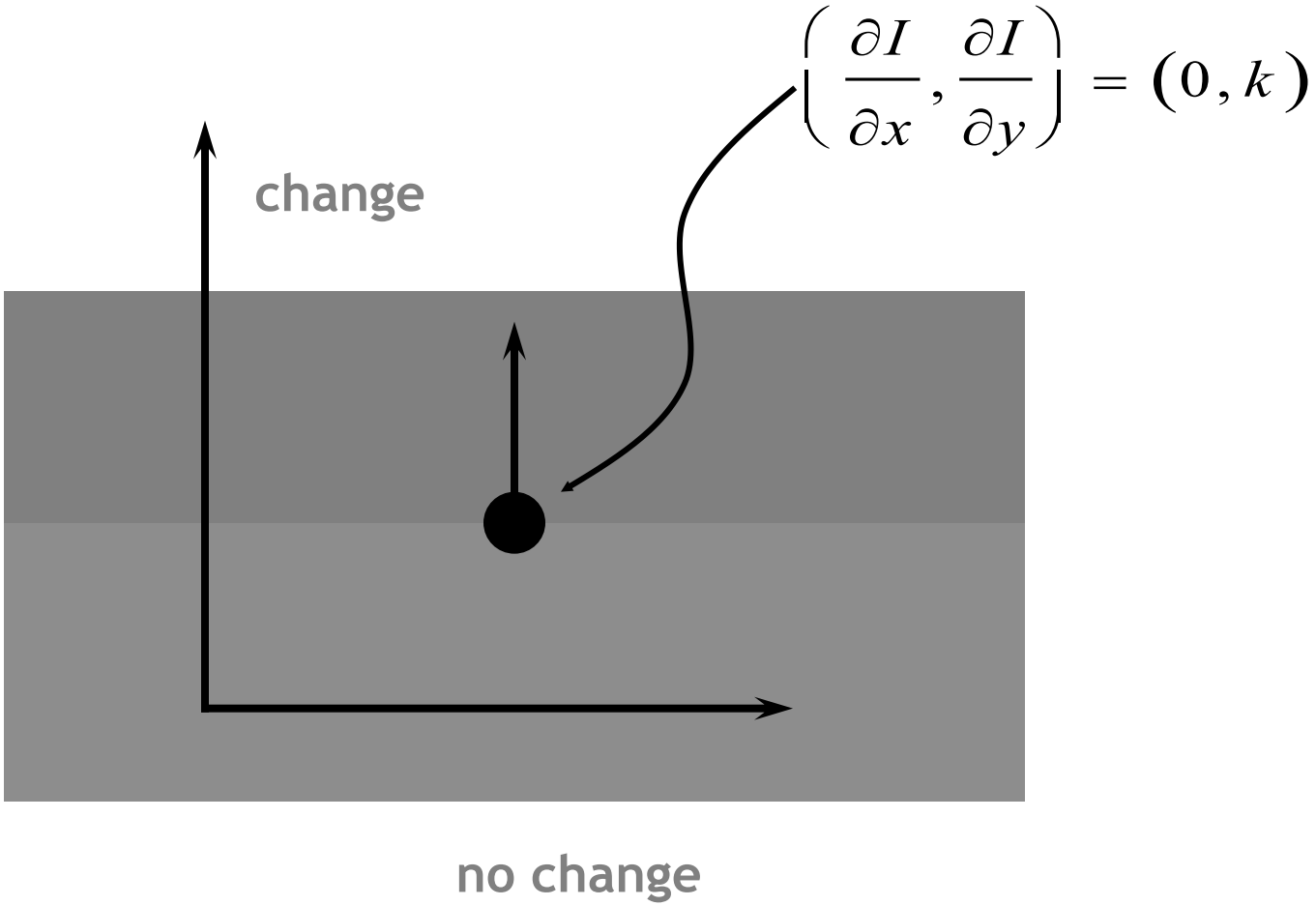


# what is the gradient ?

$$\left( \frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) = (k, 0)$$

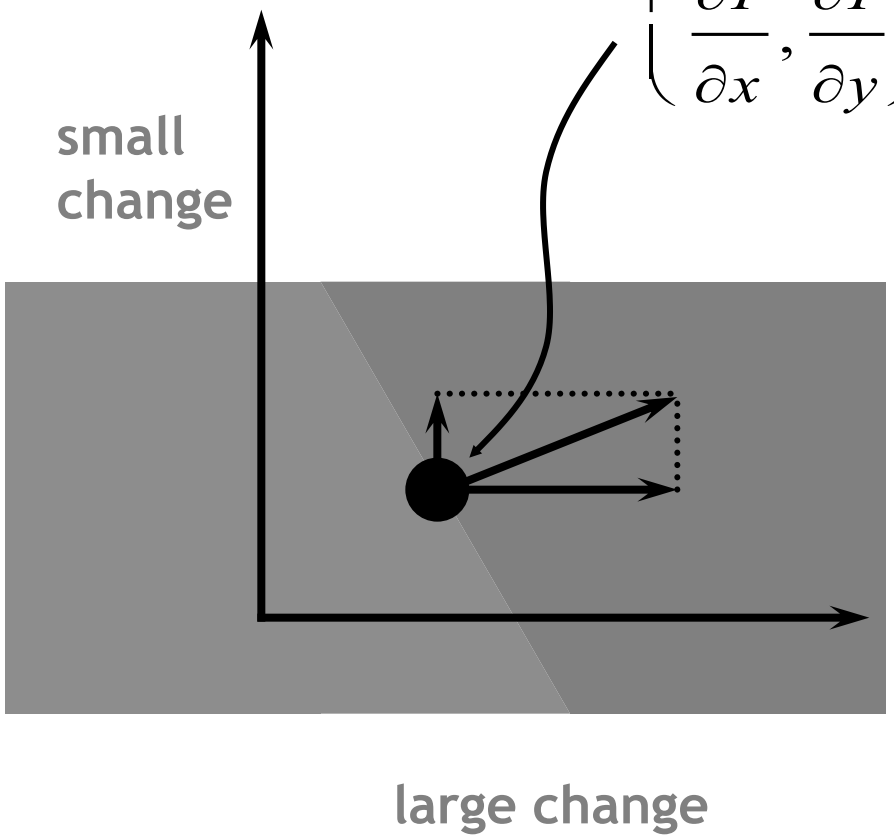


# what is the gradient ?



# what is the gradient ?

$$\left( \frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right) = (k_x, k_y)$$

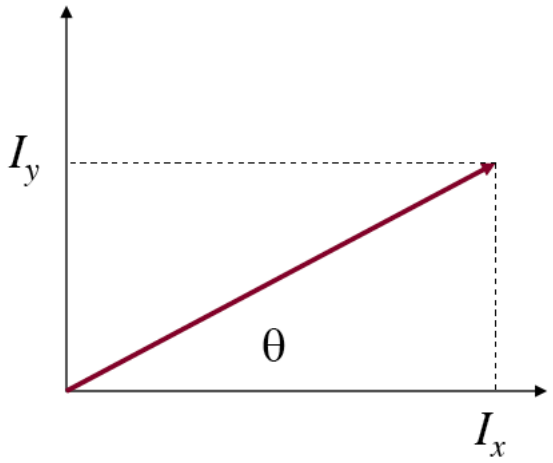



- gradient direction is perpendicular to edge
- gradient magnitude measures edge strength

# 2D Edge Detection

- calculate derivative
  - use the magnitude of the gradient
  - the gradient is:

$$\nabla I = (I_x, I_y) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)$$




  - the magnitude of the gradient is:

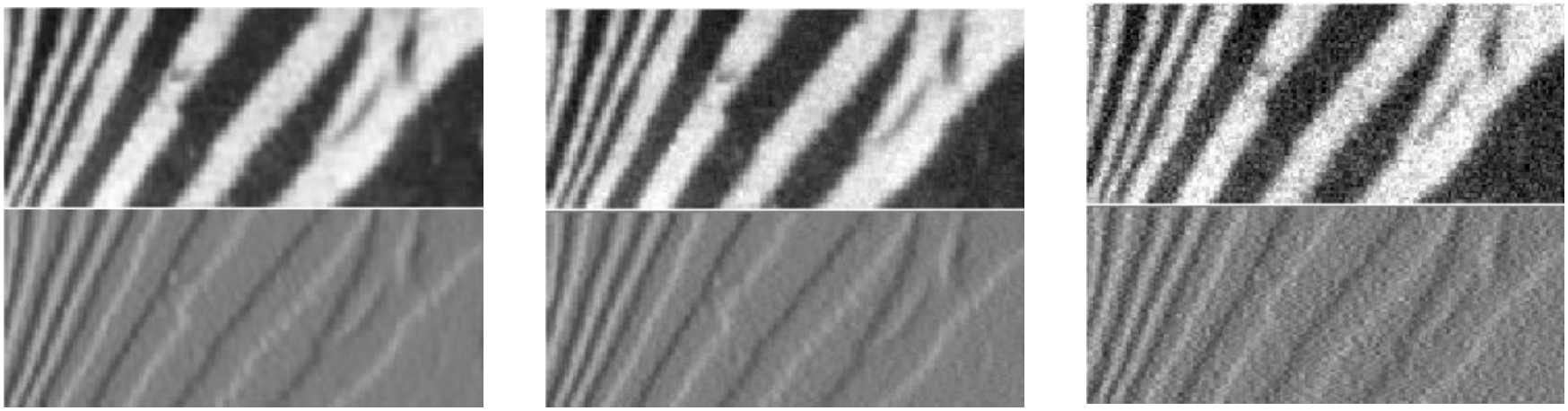
$$\|\nabla I\| = \sqrt{I_x^2 + I_y^2}$$

  - the direction of the gradient is:

$$\theta = \arctan(I_y, I_x)$$

# Finite Differences responding to noise

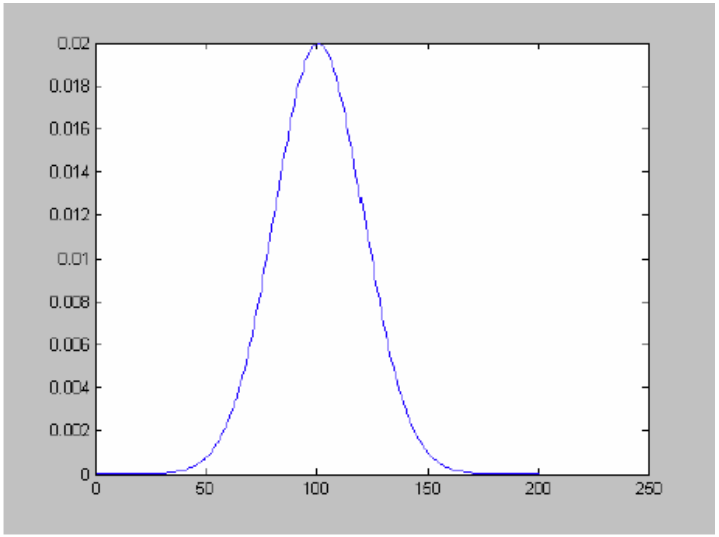
- increasing noise level (from left to right)
  - noise: zero mean additive Gaussian noise

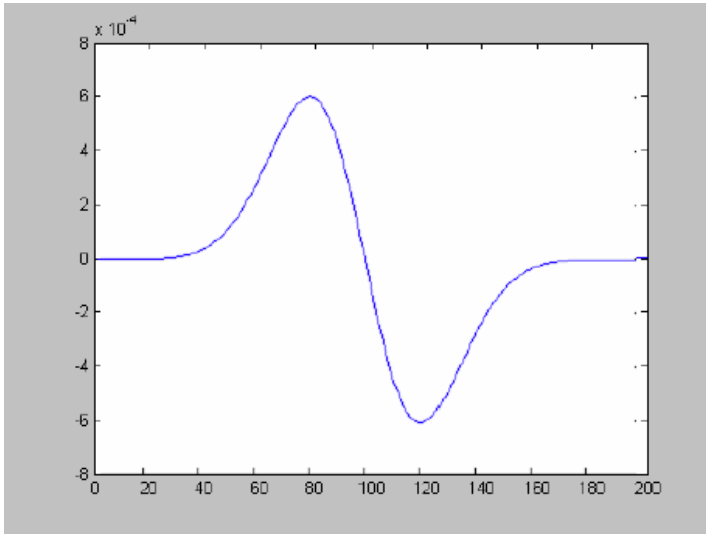


# Derivatives and Smoothing

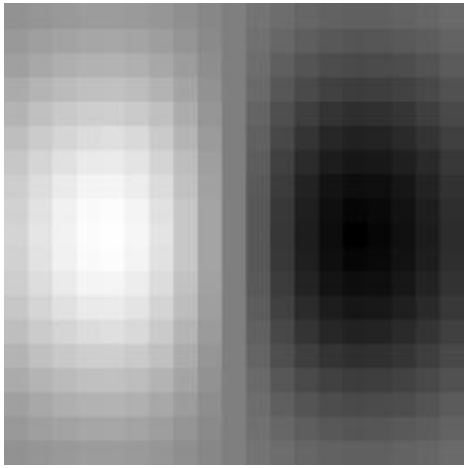
$$D_x \otimes (G \otimes I) = (D_x \otimes G) \otimes I$$

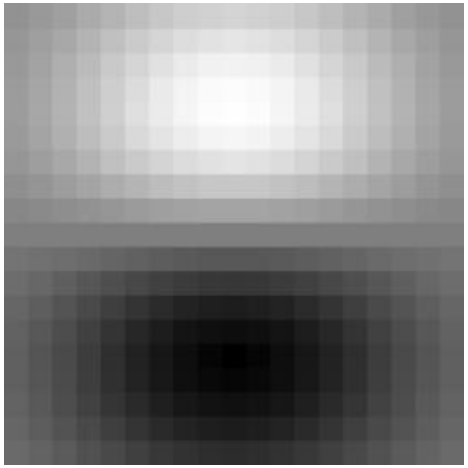
- in 1D:



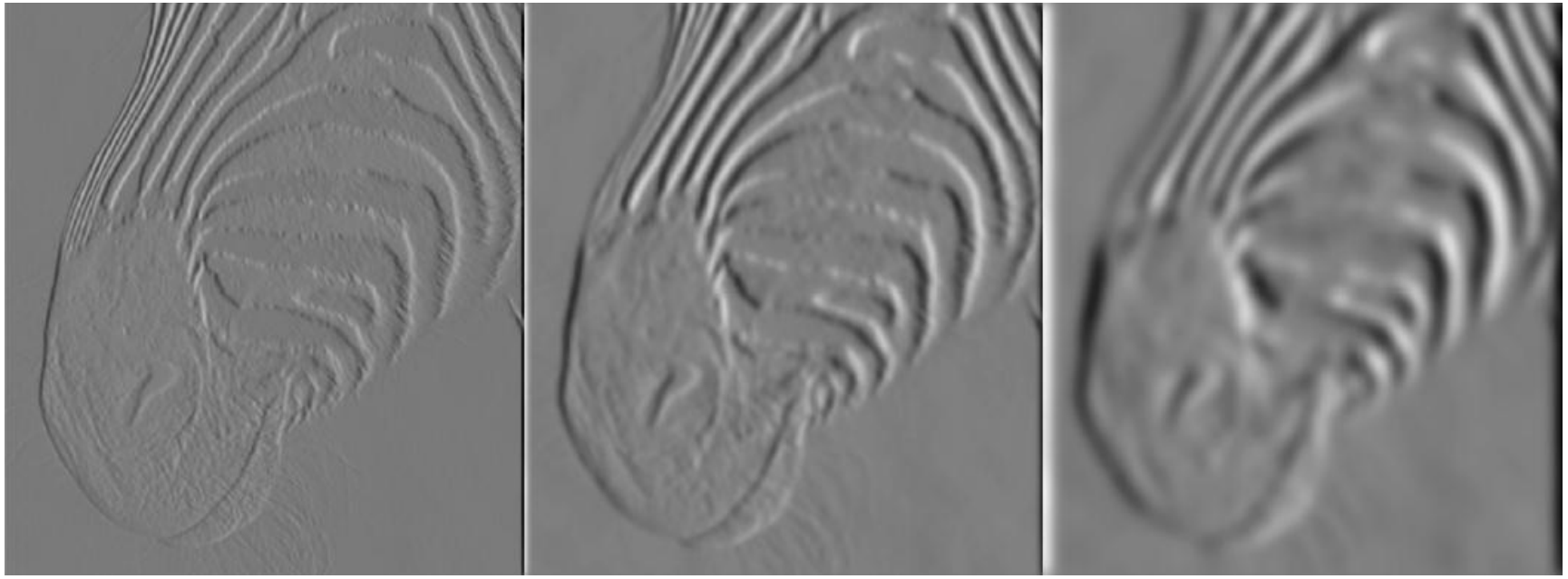

- in 2D:



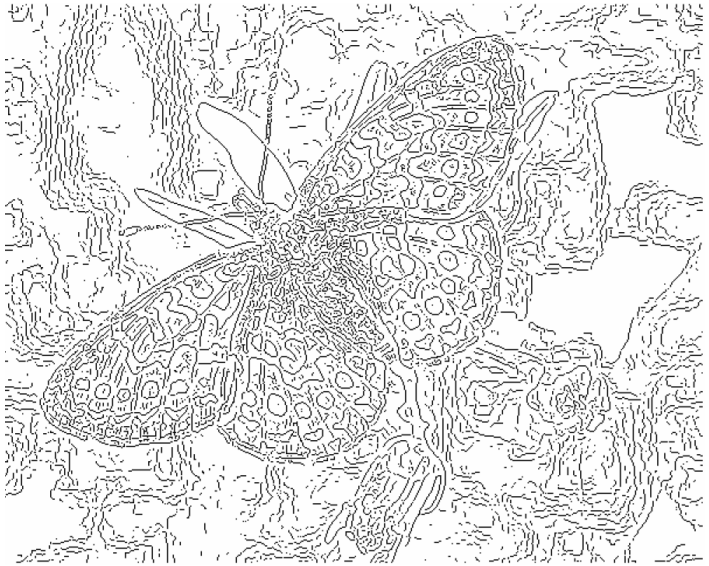


- the scale of the smoothing filter affects derivative estimates, and also the semantics of the edges recovered
  - note: strong edges persist across scales

1 pixel 3 pixels 7 pixels

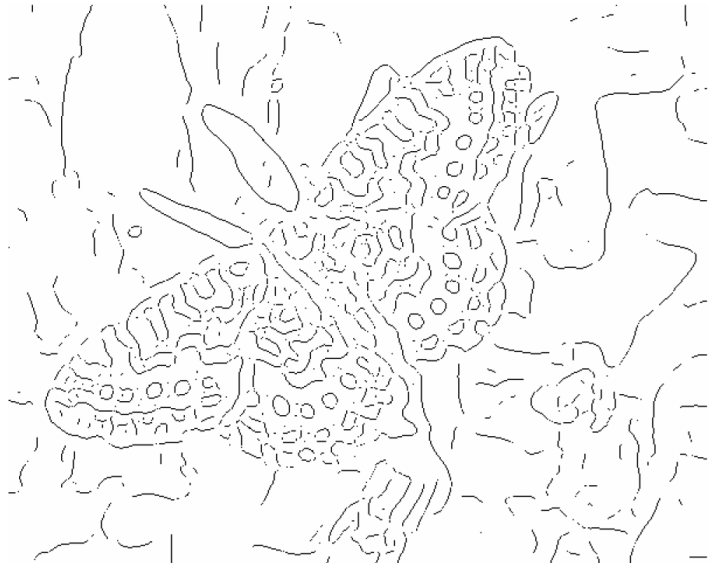


# Butterfly Example (Ponce & Forsyth)



fine scale
high
threshold

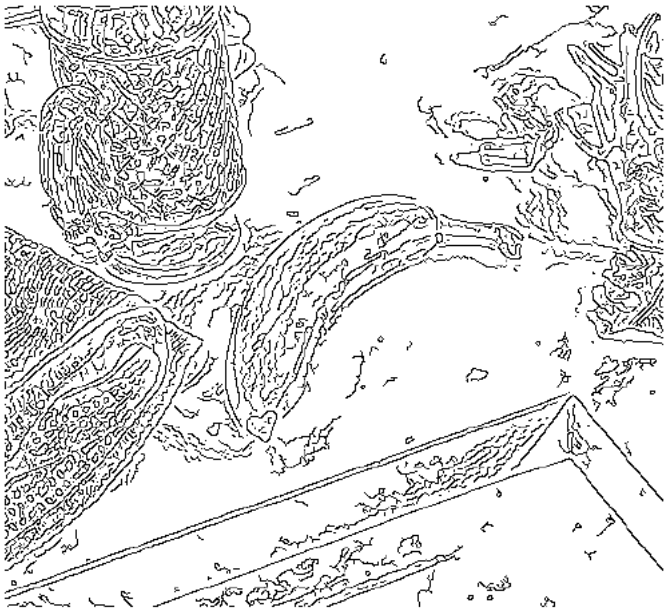
coarse
scale
low
threshold



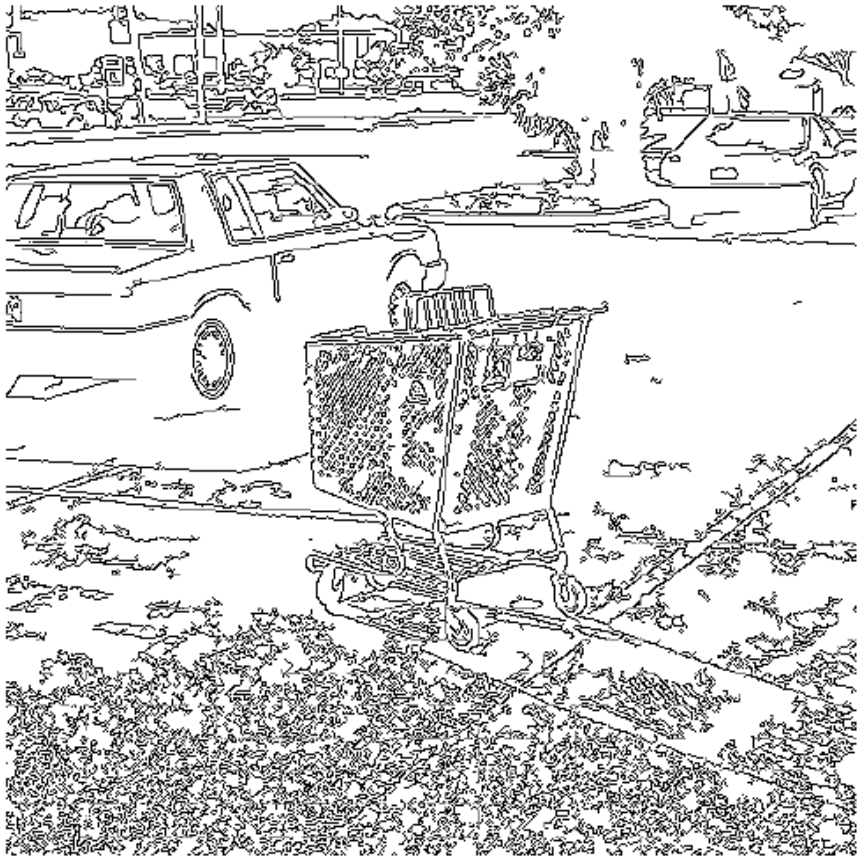
coarse
scale,
high
threshold

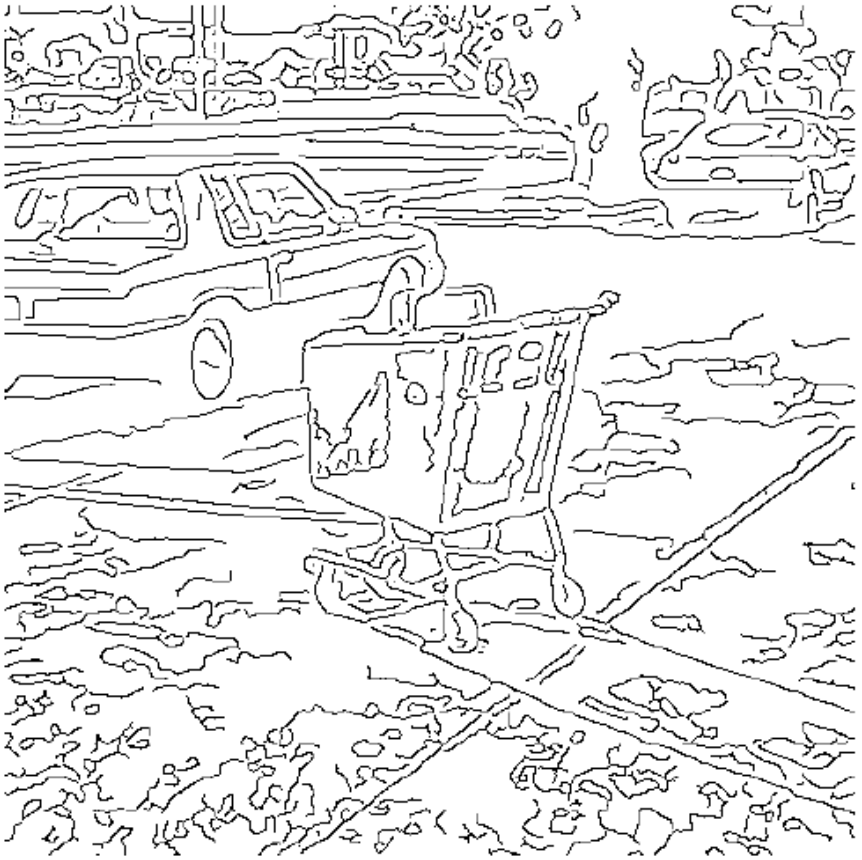
# line drawing vs. edge detection

- more examples…

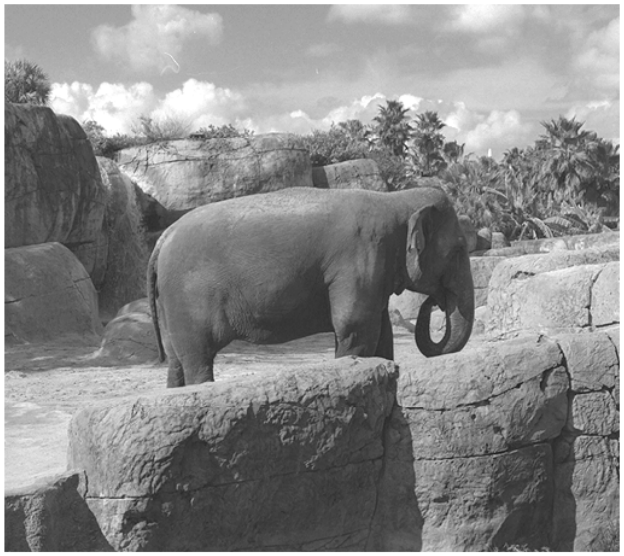


# Edges…

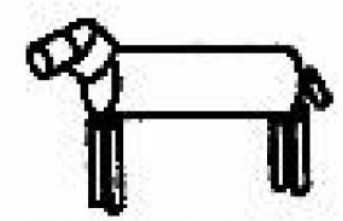


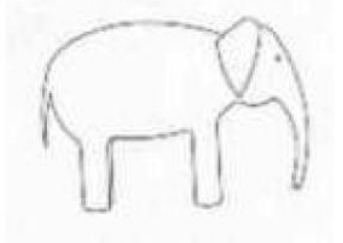


University of South Florida



University of South Florida

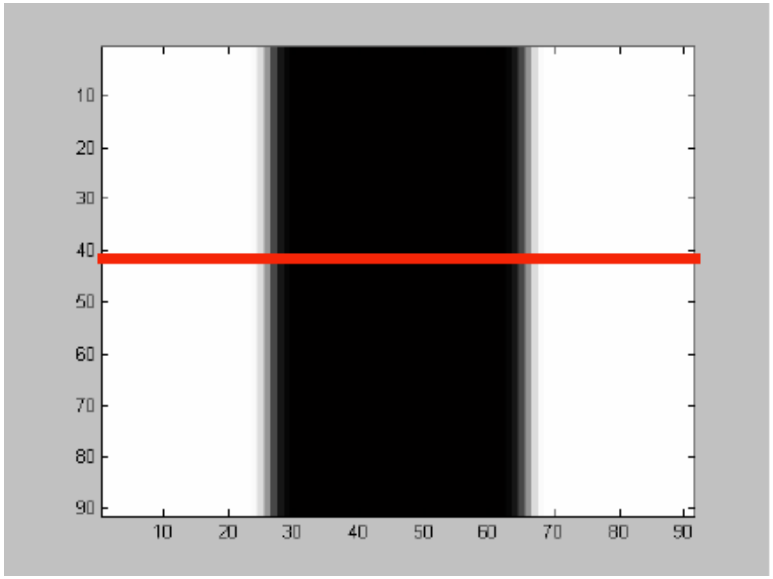


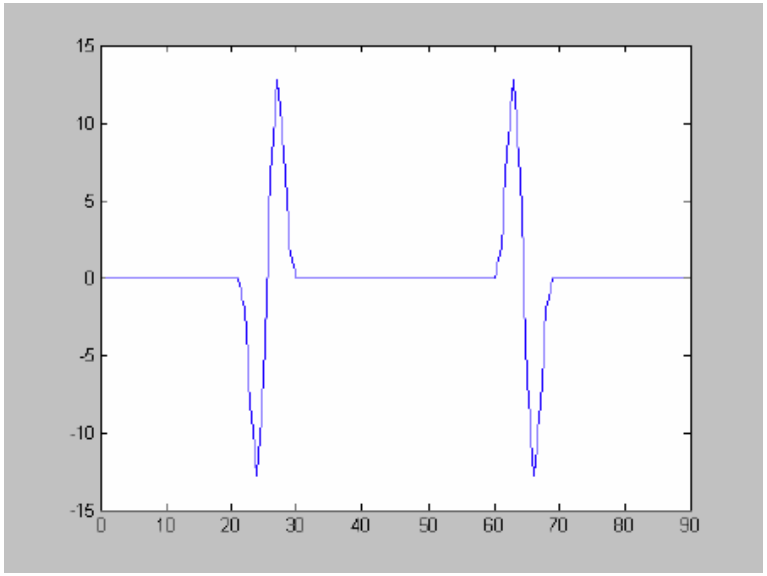


Match “model” to measurements?

# Step Edge

- recall:
  - the zero-crossings of the second derivative tell us the location of edges



# The Laplacian

- The Laplacian:

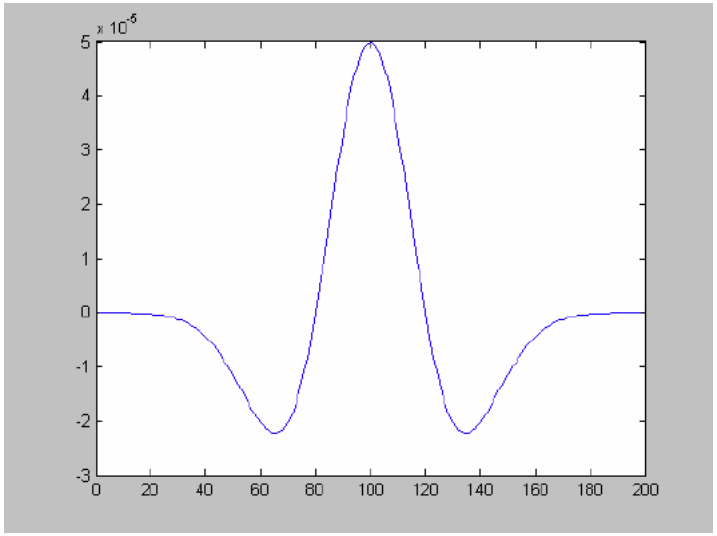
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

  - just another linear filter:

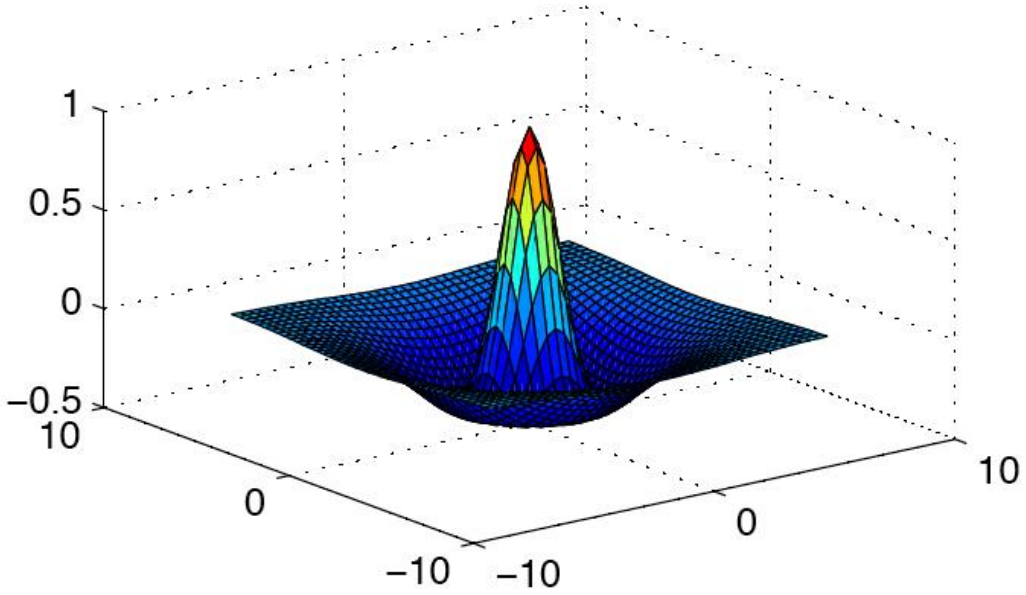
$$\nabla^2 (G \otimes f) = \nabla^2 G \otimes f$$

# Second Derivative of Gaussian

- in 1D:

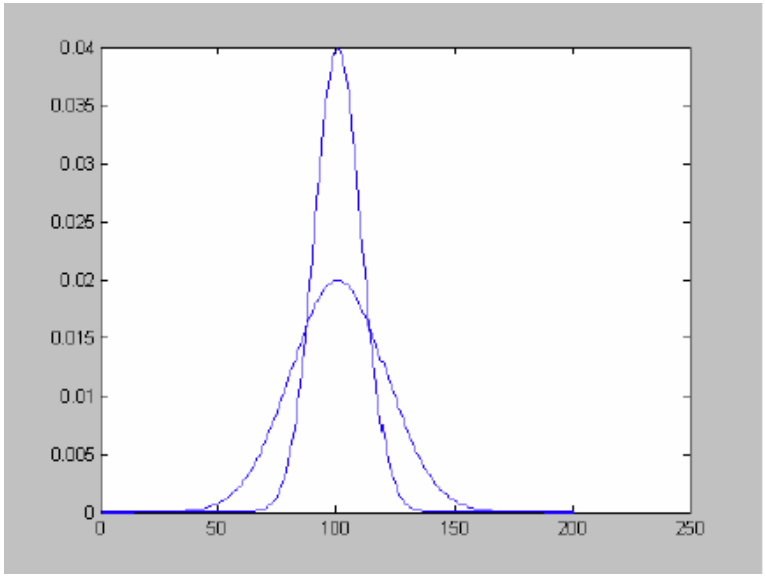


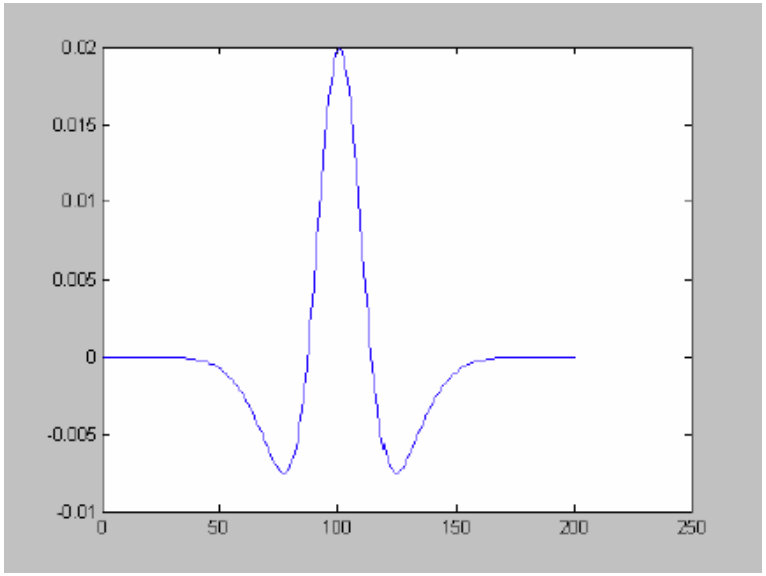

- in 2D ('mexican hat')



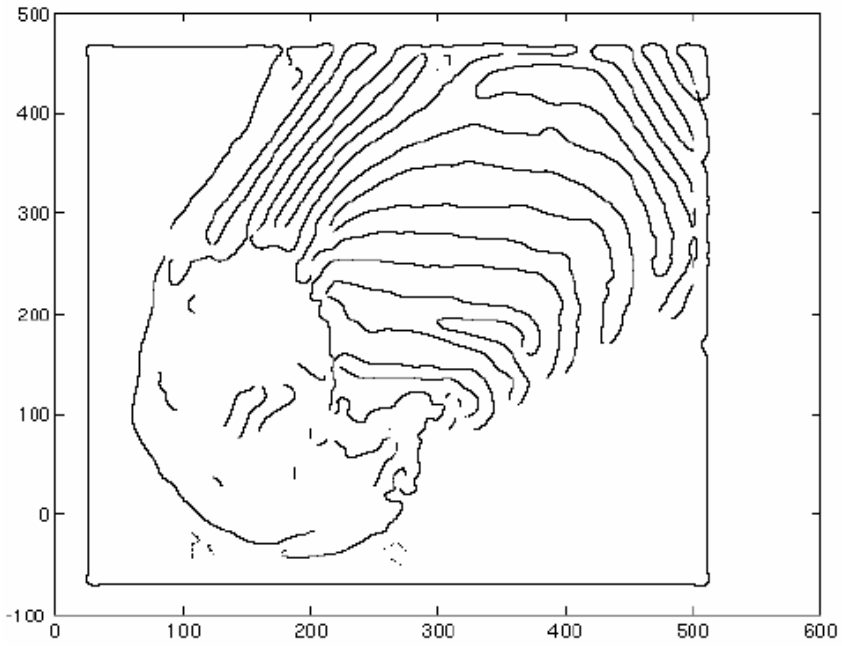
# Approximating the Laplacian

- Difference of Gaussians at different scales:

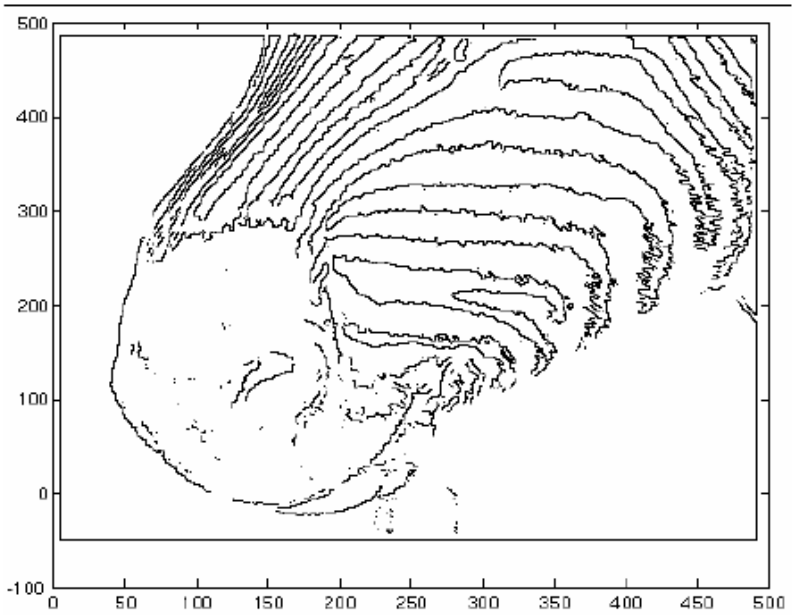


- sigma = 4




- sigma = 2



# Problems of Laplacian

- Filter is not oriented, response is an average across an edge and along an edge
- Poor behavior at corners
- Components along the edge tend to increase sensitivity to noise → leads to imprecisions in localization

# End of Lecture

# Color Images

- Store 3 of these arrays of numbers
  - One array for red information
  - One array for green information
  - One array for blue information
- Sounds simple, but how do we sense this information with a camera?

# What are some approaches to sensing color images?

- Scan 3 times (temporal multiplexing)
- Use 3 detectors (3-ccd camera, and color film)
- Use offset color samples (spatial multiplexing)

# Some approaches to color sensing

- Scan 3 times (temporal multiplexing)
  - Flat-bed scanners
  - Russian photographs from 1800's
- Use 3 detectors
  - High-end 3-tube or 3-ccd video cameras
  - Photographic film
- Use spatially offset color samples (spatial multiplexing)
  - Single-chip CCD color cameras
  - Human eye

# CCD color filter pattern

detector