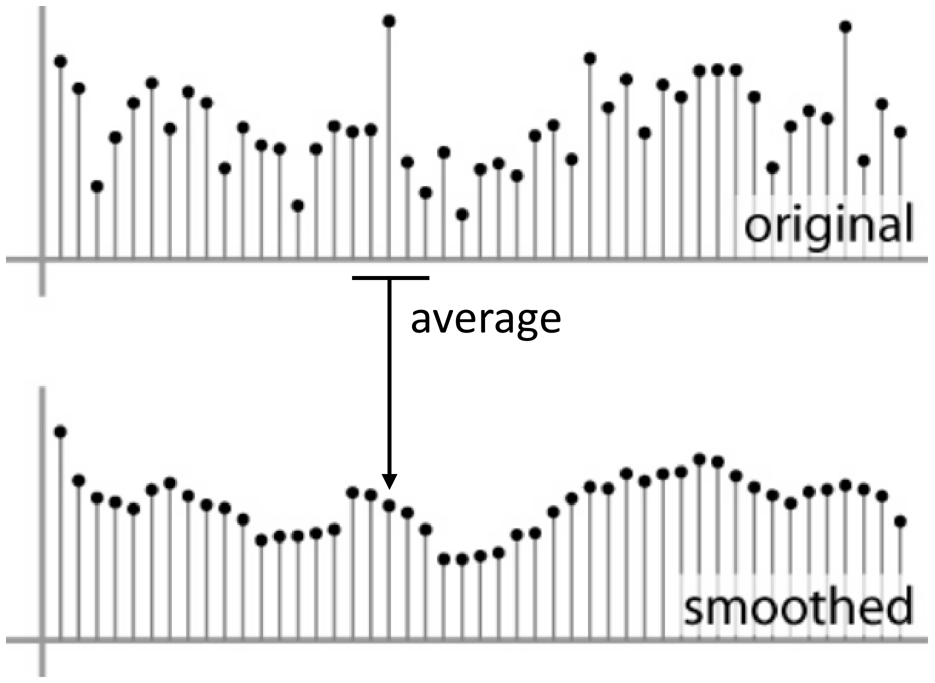
# Local Filtering and Edge Detection

Alessandro Giusti

## Local filtering

# Simple filtering in 1D: moving average

Replace every value with the average of the pixels in its neighborhood



$$b_{\text{smooth}}[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} b[j]$$

Slide credit: S. Marschner

## Convolution

A convolution is a **weighted moving average**

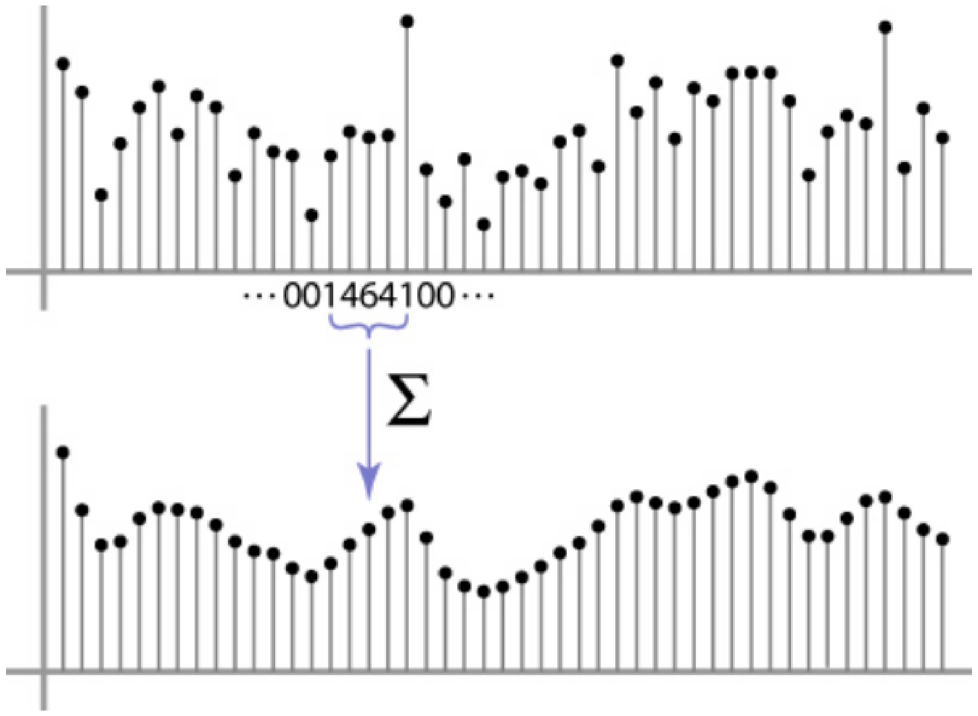
$$(a \star b)[i] = \sum_j a[j]b[i-j]$$

The sequence of weights a[j] is called filter or convolution kernel.

Slide credit: S. Marschner

# 1D convolution example

Using a bell-shaped kernel
a = [1, 4, 6, 4, 1] / 16

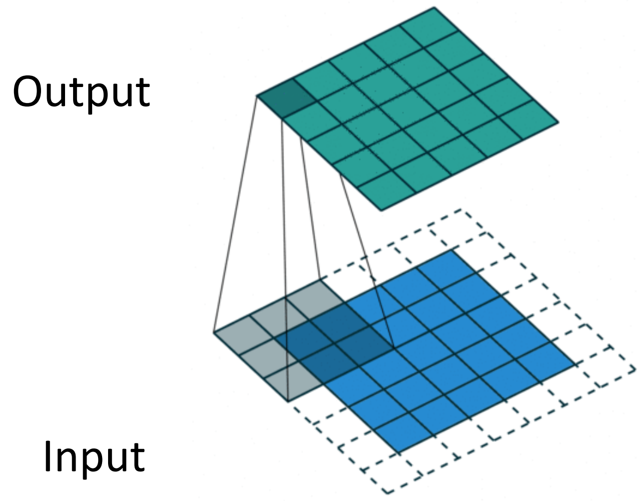


# 2D convolution

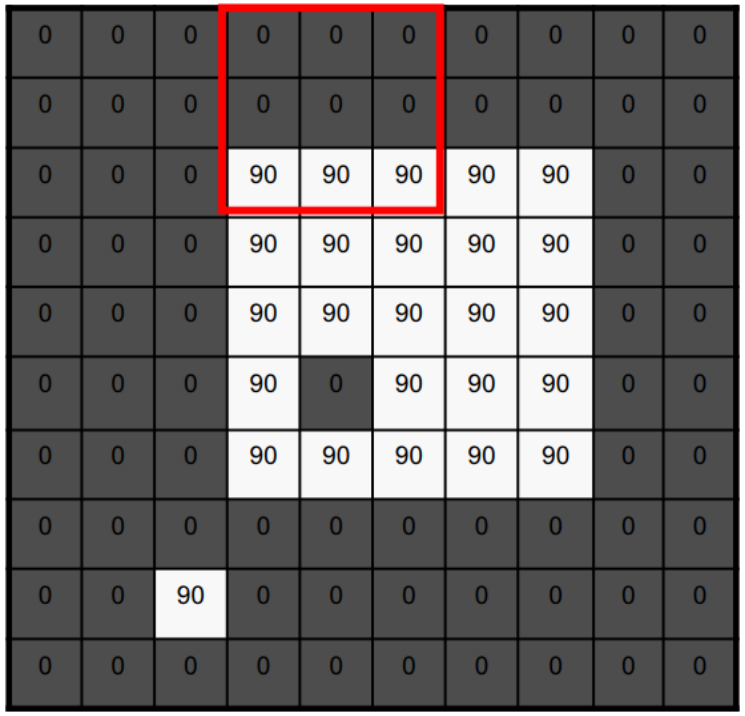
Same as 1D, with one more index

$$(a \star b)[i,j] = \sum_{i',j'} a[i',j']b[i-i',j-j']$$

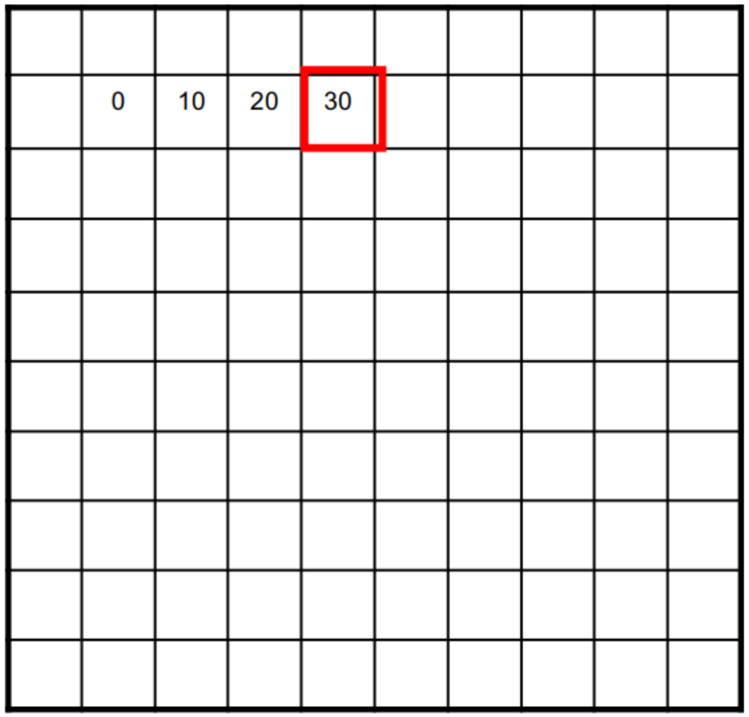
Now the filter is a rectangle you slide around over a grid of numbers



# Example

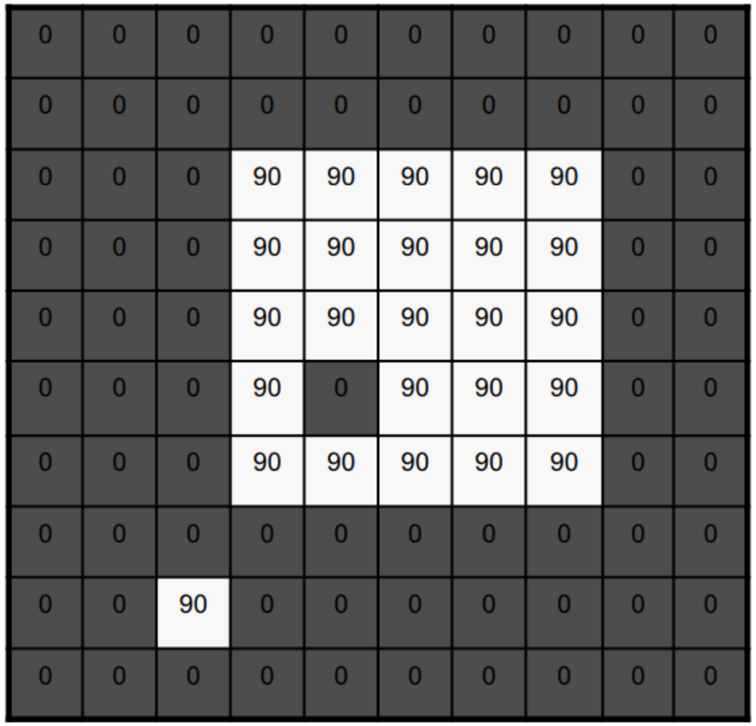


| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

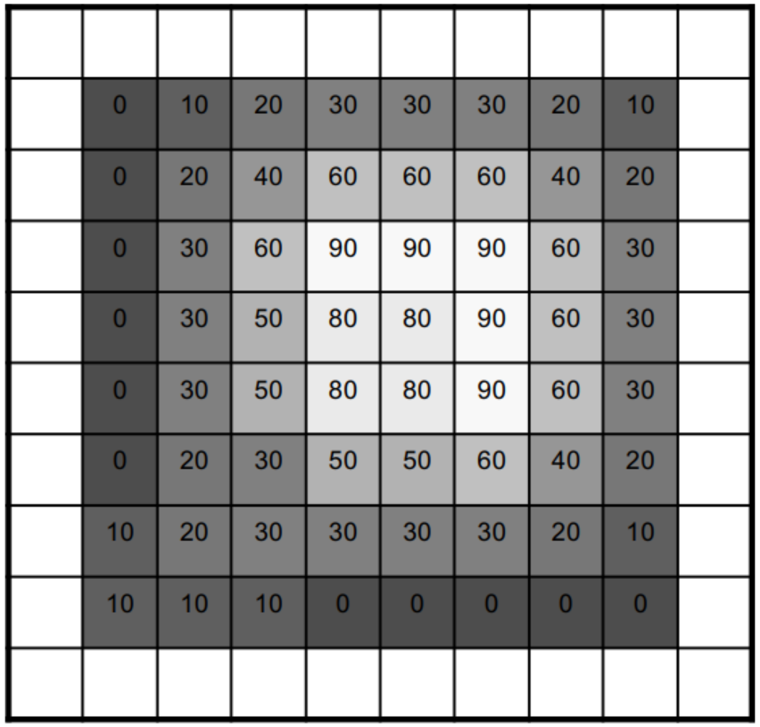


| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

# Example



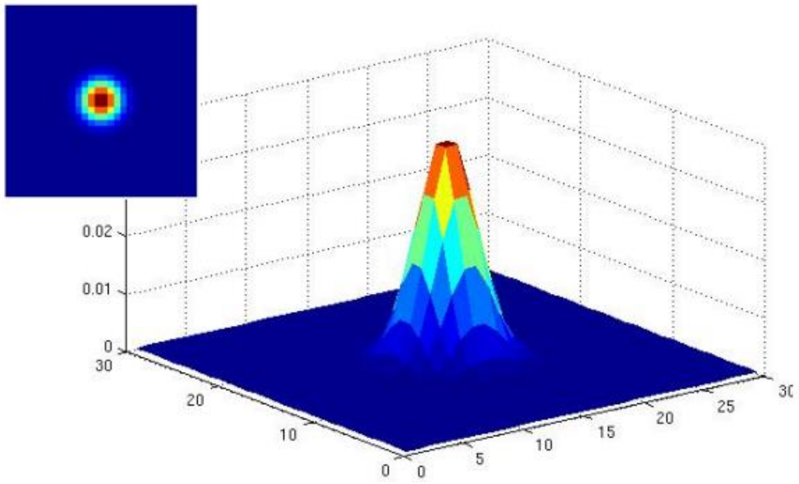
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



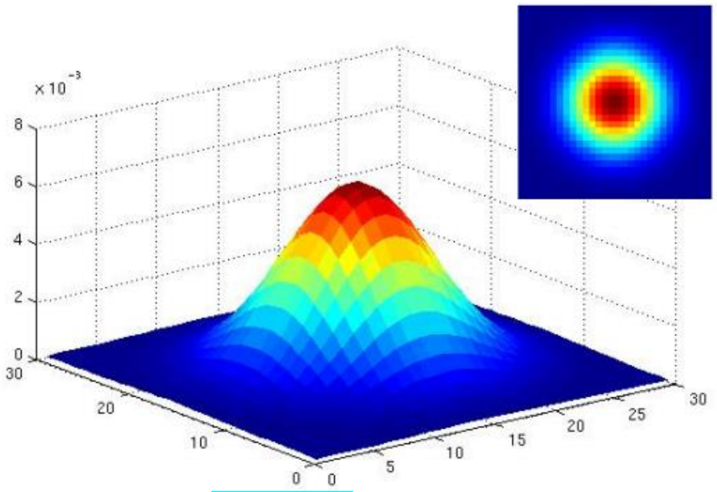
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | 30 | 20 | 10 | |
| | 0 | 20 | 40 | 60 | 60 | 60 | 40 | 20 | |
| | 0 | 30 | 60 | 90 | 90 | 90 | 60 | 30 | |
| | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 | |
| | 0 | 30 | 50 | 80 | 80 | 90 | 60 | 30 | |
| | 0 | 20 | 30 | 50 | 50 | 60 | 40 | 20 | |
| | 10 | 20 | 30 | 30 | 30 | 30 | 20 | 10 | |
| | 10 | 10 | 10 | 0 | 0 | 0 | 0 | 0 | |
| | | | | | | | | | |

# Smoothing with gaussian filters

- Variance (sigma) determines the amount of smoothing



$\sigma = 2$ with
30 x 30 kernel



$\sigma = 5$ with
30 x 30 kernel

# Example

Original

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

Kernel

# Example

Original

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

Kernel

same as original

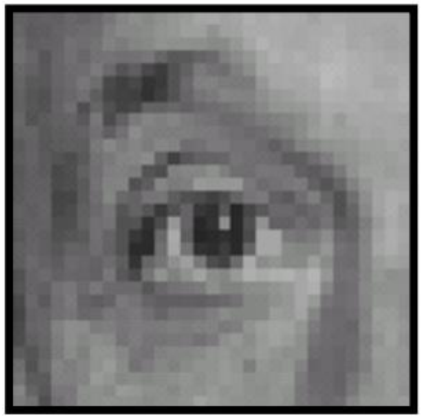
## Example

Original

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

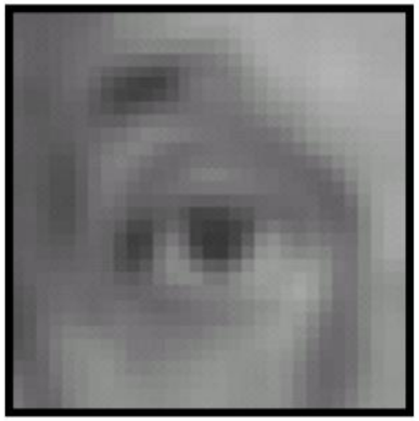
Kernel

## Example


Original

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

Kernel


smoothed

# Example

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

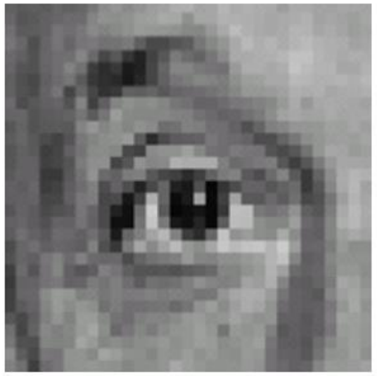
Kernel

# Example

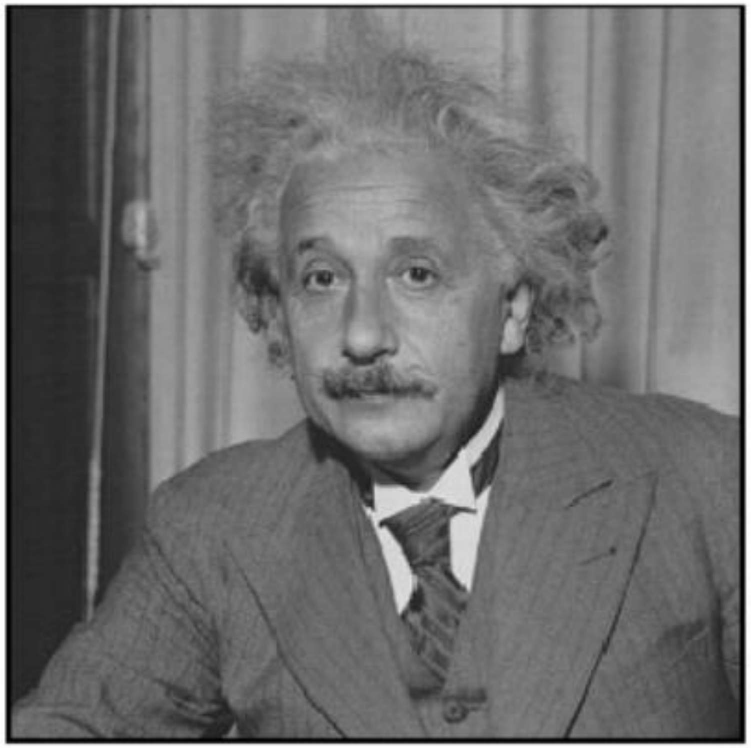
Original

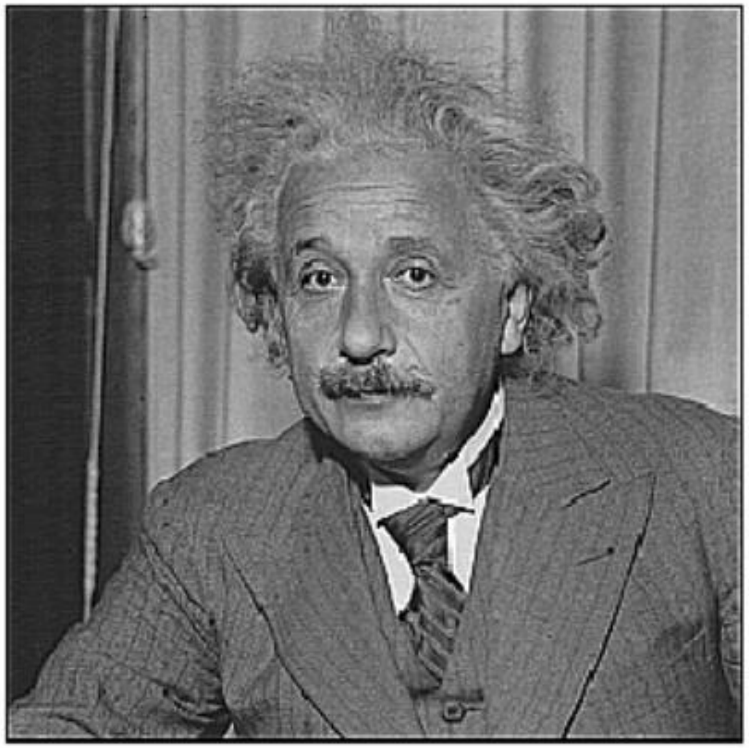
$$\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} - \frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Kernel

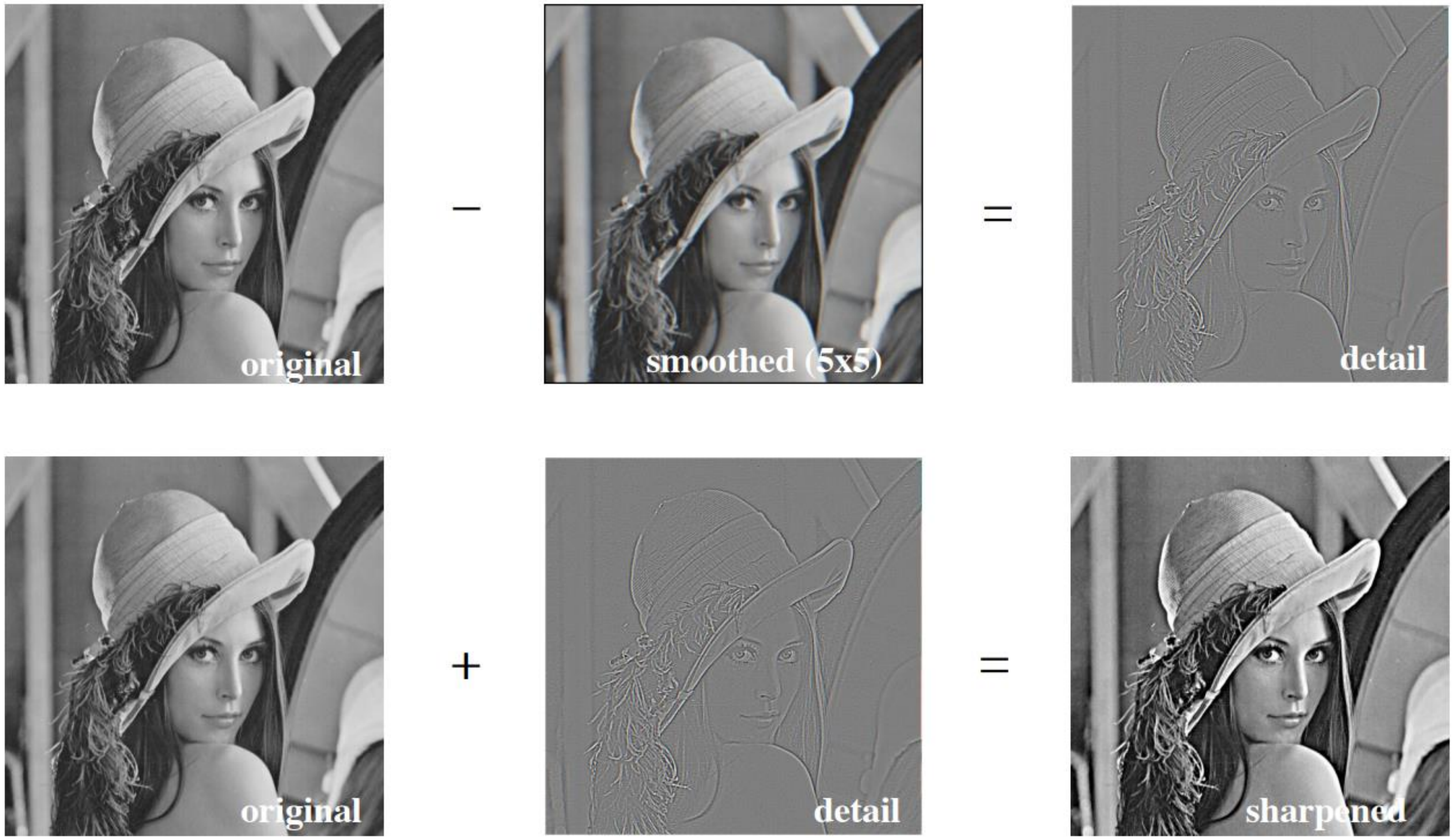

sharpened

# Sharpening


before


after

# Why does sharpening work?



## Edge detection

# Edges are important features when we look at images!

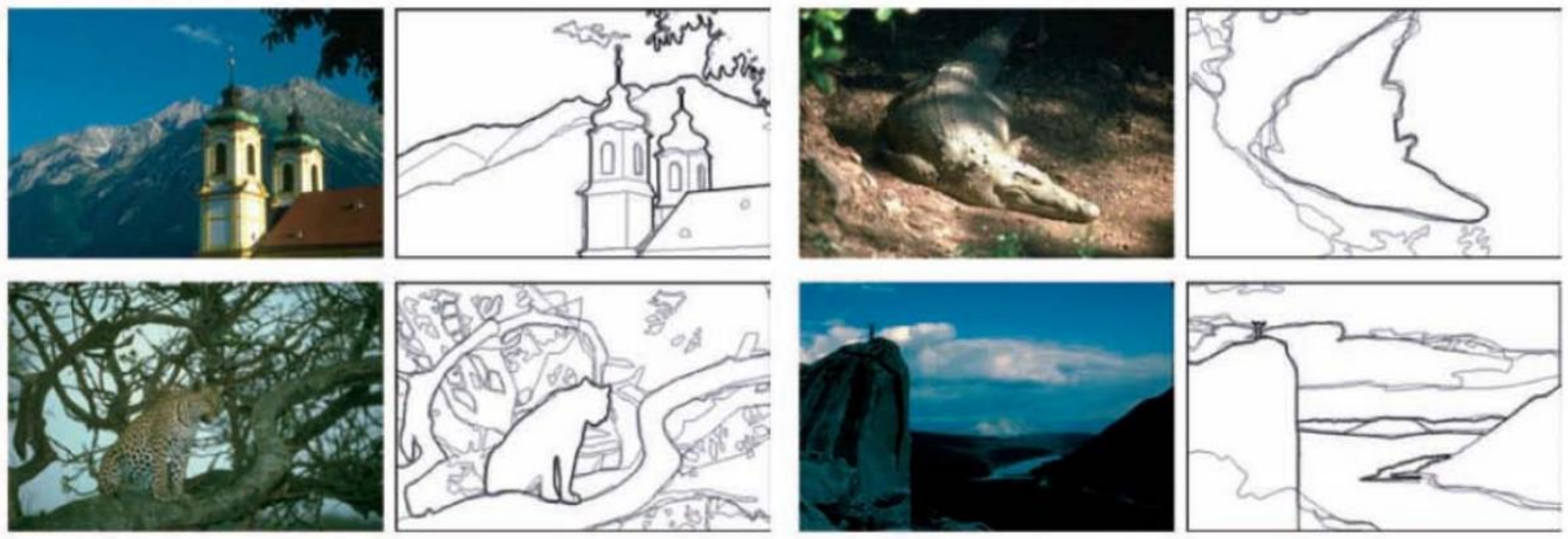
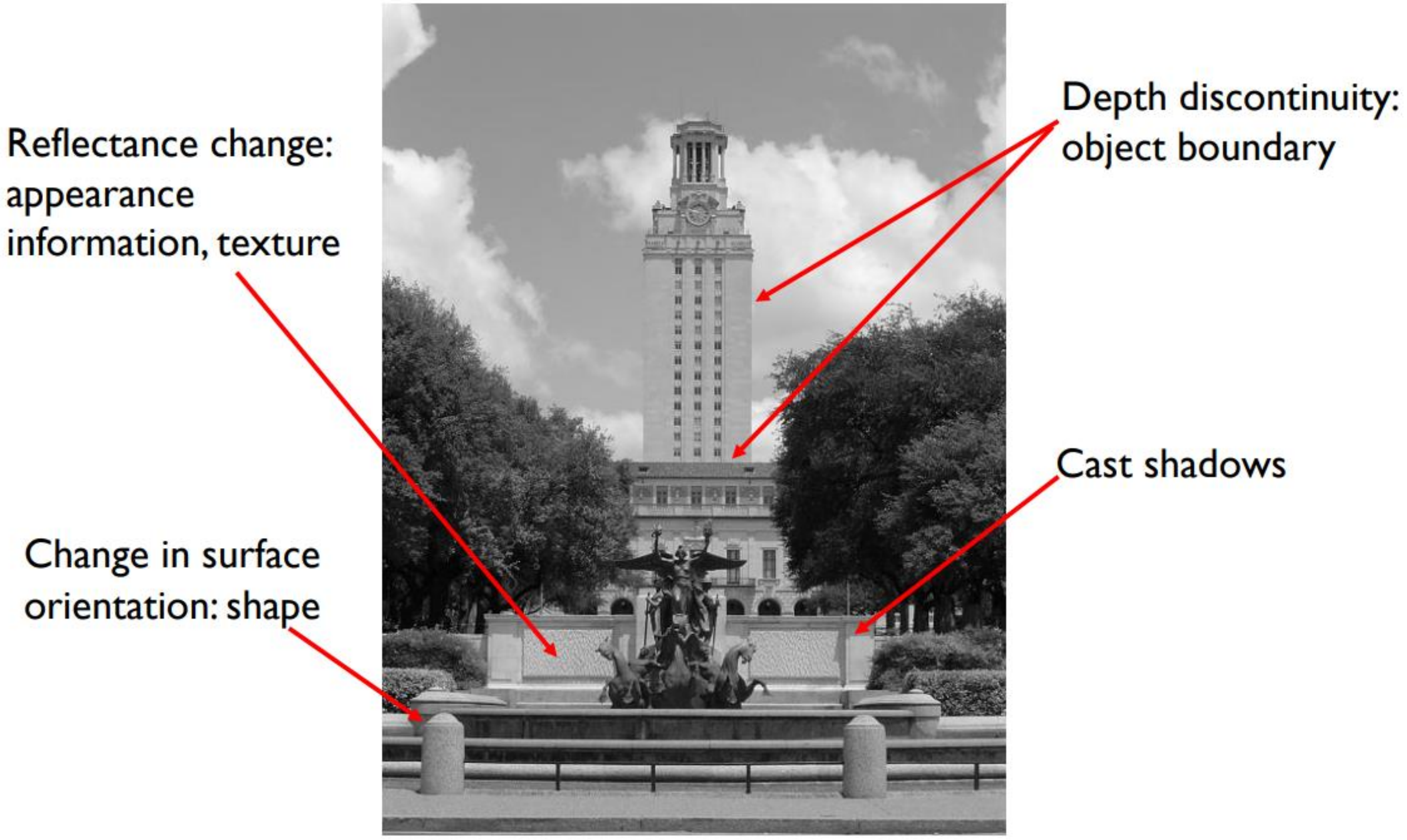


**Figure 4.31** Human boundary detection (Martin, Fowlkes, and Malik 2004) © 2004 IEEE. The darkness of the edges corresponds to how many human subjects marked an object boundary at that location.

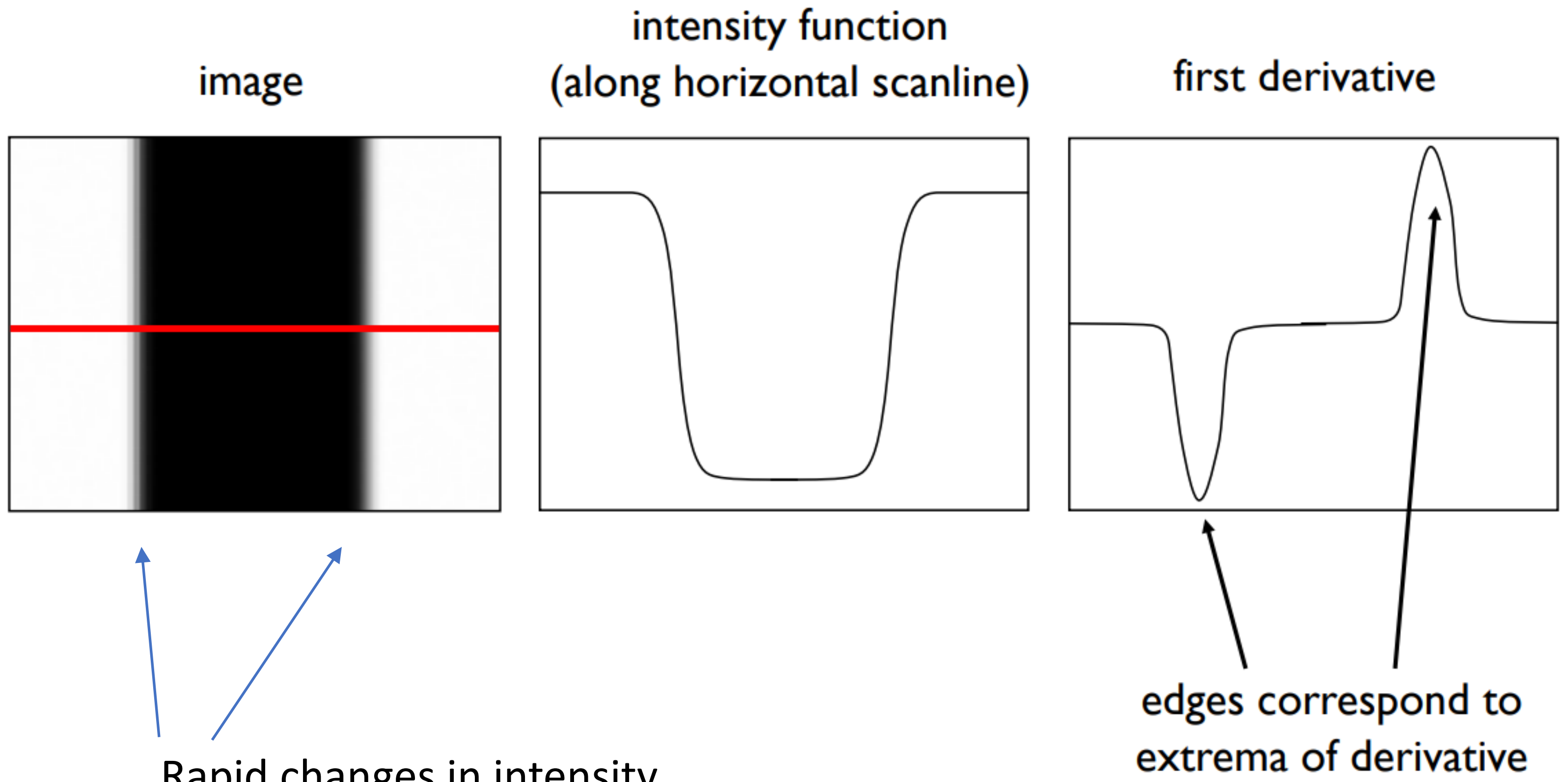
Szelinski: Computer Vision: theory and Applications

# Causes of edges


Reflectance change: appearance information, texture
Depth discontinuity: object boundary
Cast shadows
Change in surface orientation: shape

Slide credit: K. Grauman

# What is an edge?



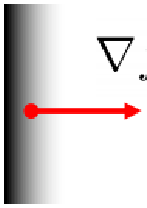
Rapid changes in intensity
(edges)

Slide credit: K. Grauman

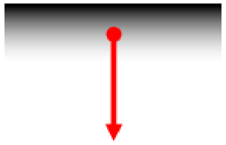
# In 2D…

- In 2D, the derivative corresponds to the **gradient**
- The gradient is a vector defined for every point in the image
- It points to the direction of most rapid increase of the intensity

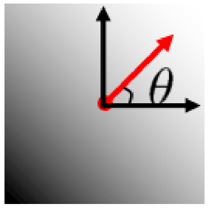
The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$



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



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

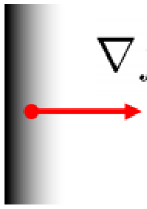


$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

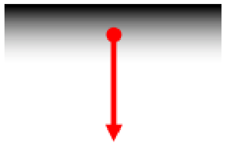
Slide credit: S. Seitz

# Gradient direction and strength

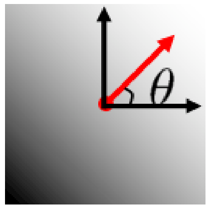
The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$



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



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



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

- Gradient direction (orthogonal to the edge):

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

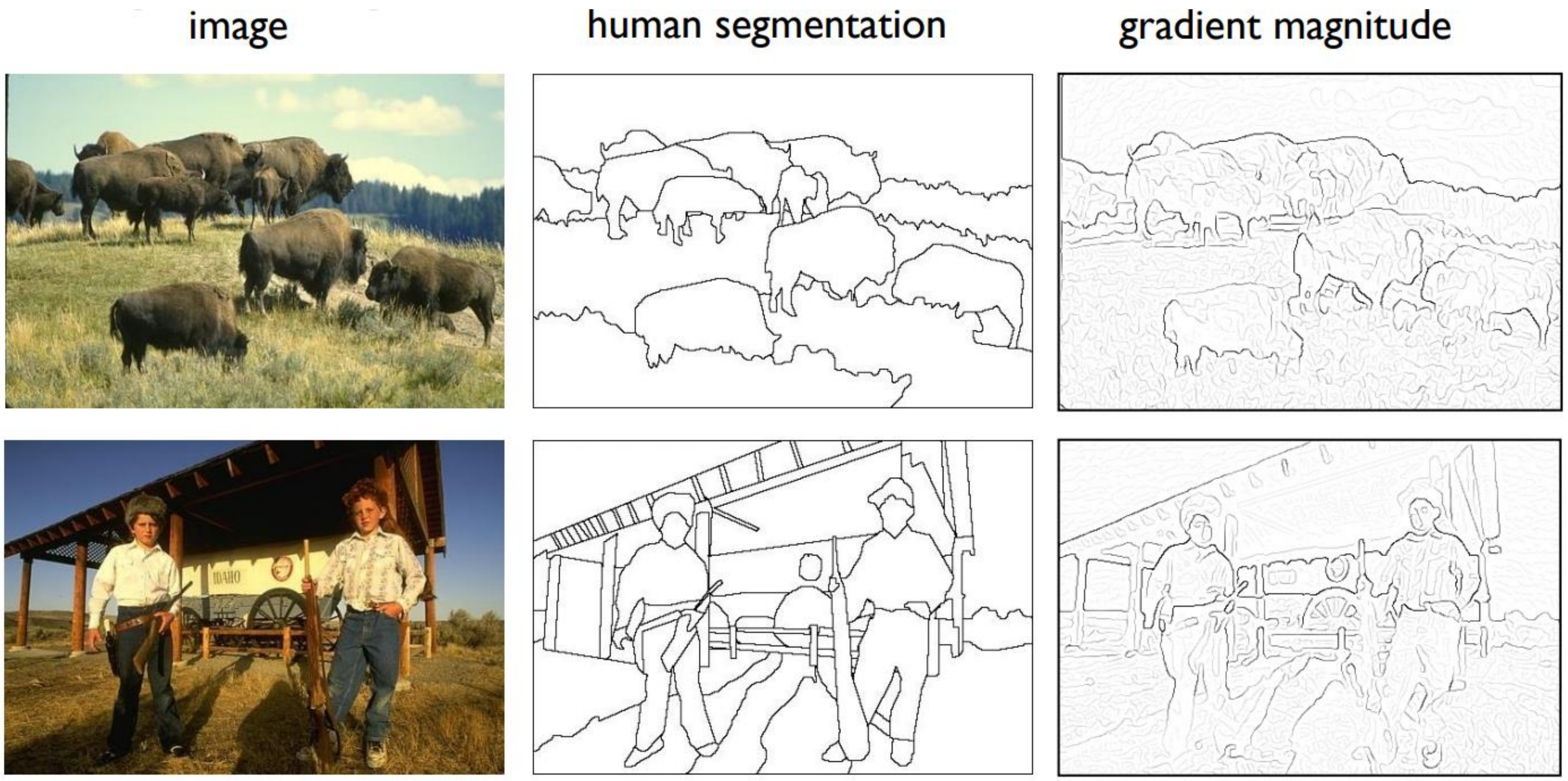
- Gradient magnitude (strength of the edge):

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

Slide credit: S. Seitz

## Keep in mind:

- Not all important edges have strong gradients
- Not all strong gradients are important edges



## Computing the gradient on an image

- How do we approximate $\frac{\partial f}{\partial x}$ ?

- How do we approximate $\frac{\partial f}{\partial y}$ ?

# Computing the gradient on an image

- How do we approximate $\frac{\partial f}{\partial x}$ ? convolve with

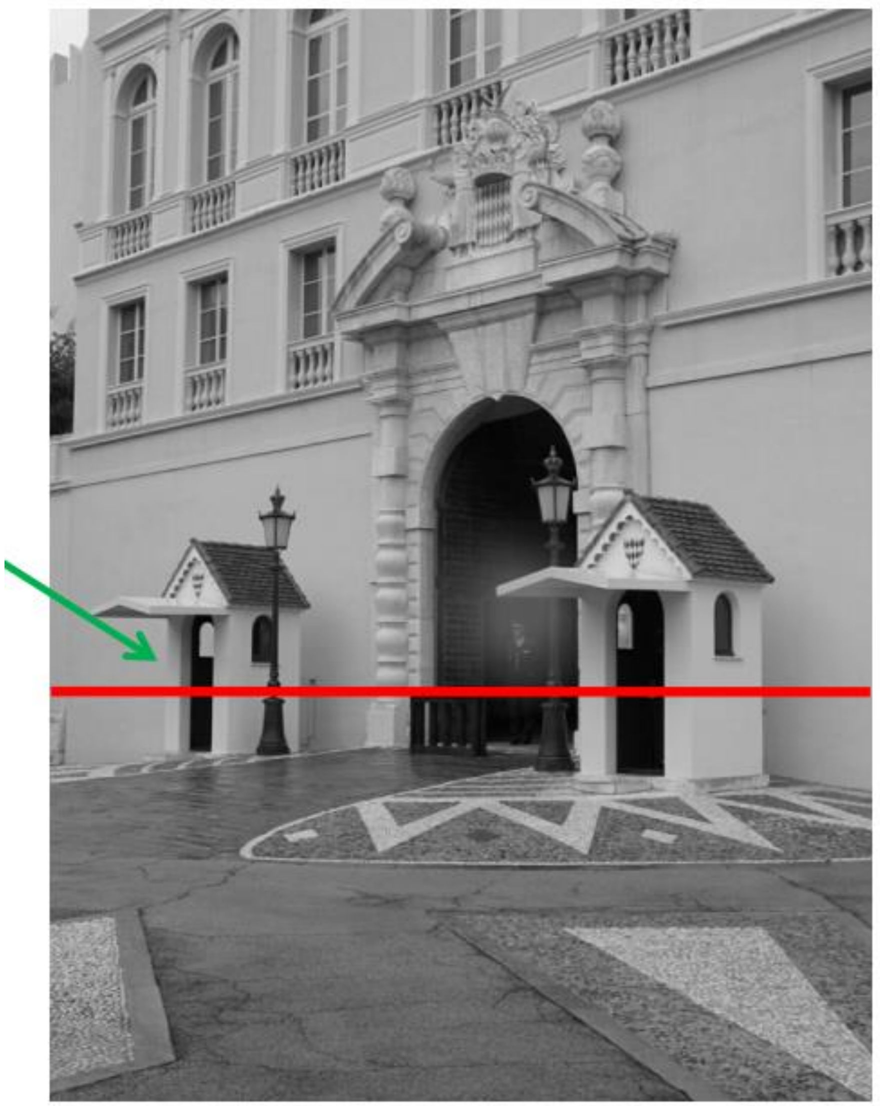
| | |
|---|---|
| -1 | 1 |

- How do we approximate $\frac{\partial f}{\partial y}$ ? convolve with

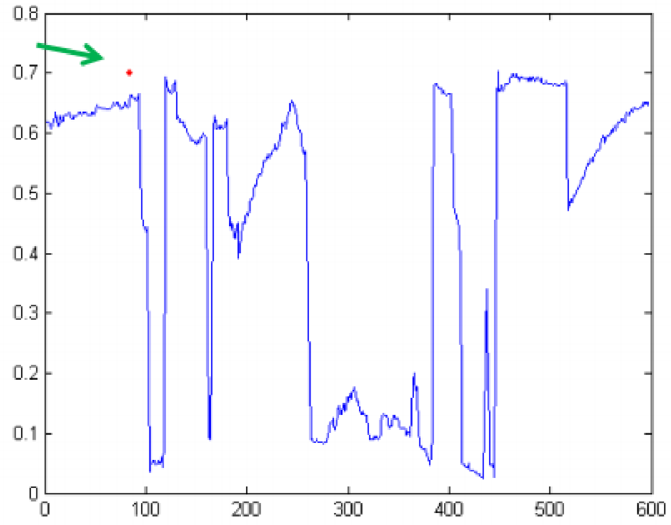
| |
|---|
| -1 |
| 1 |

Finite differencing!

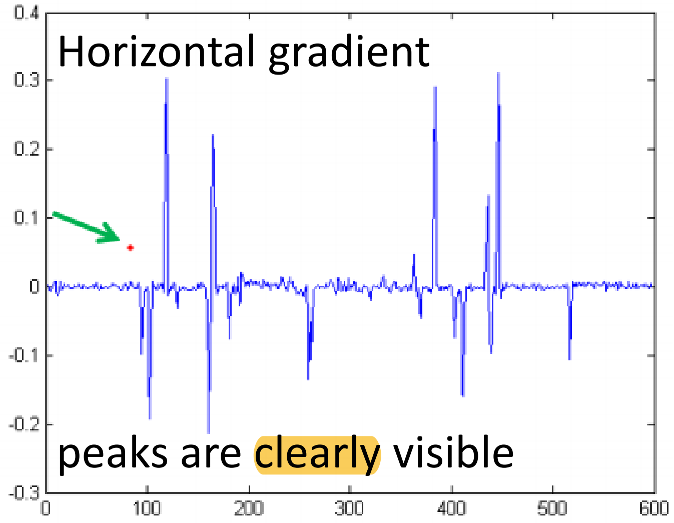
# Example of finite differencing



Intensity along profile

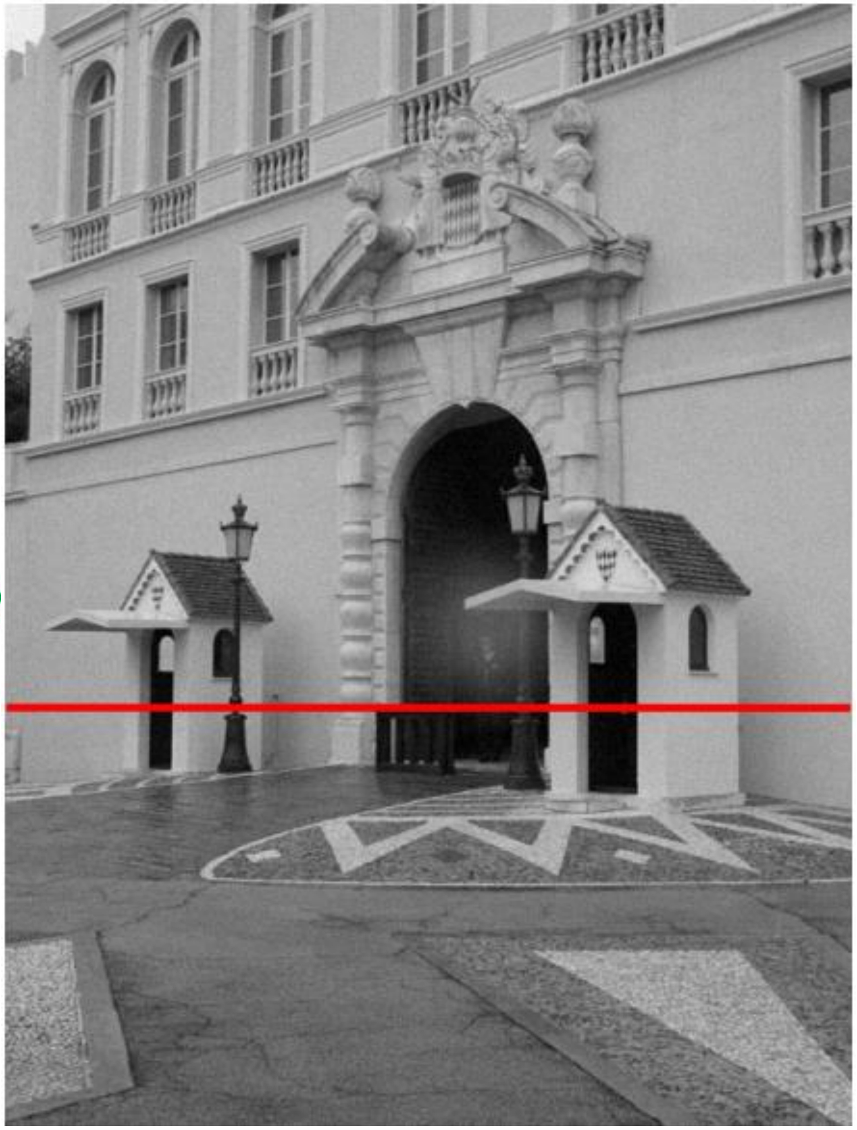


Horizontal gradient

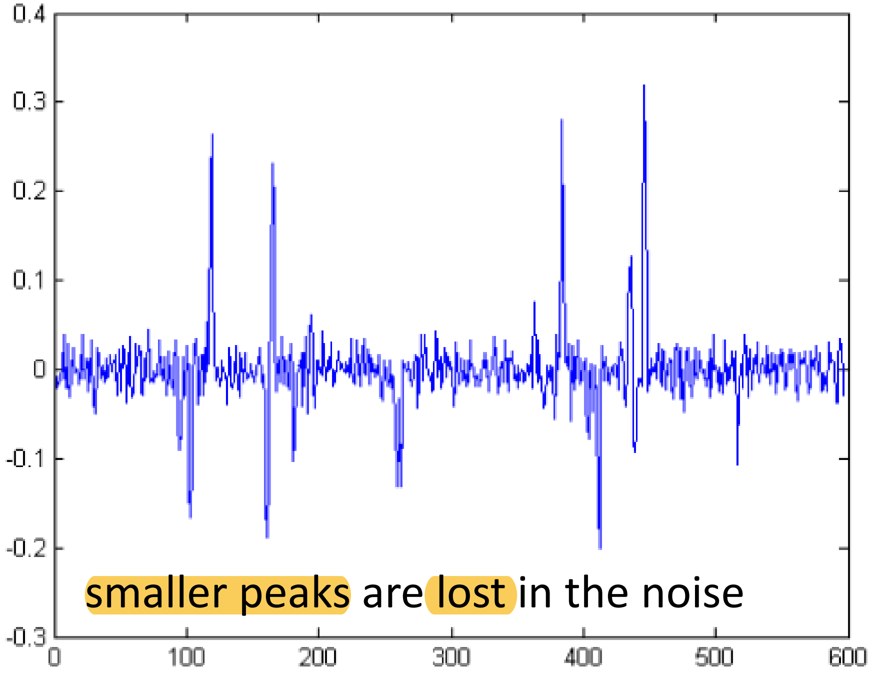


peaks are clearly visible

# If we add a tiny amount of noise to the image...



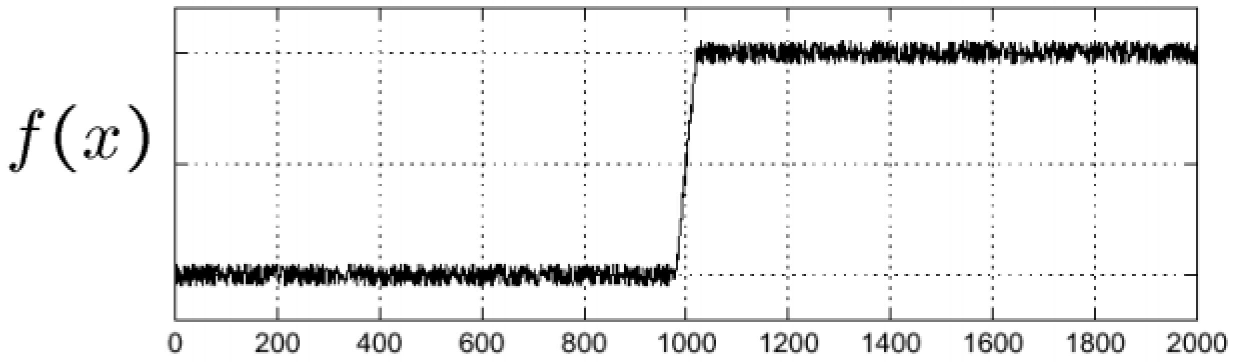
Horizontal gradient

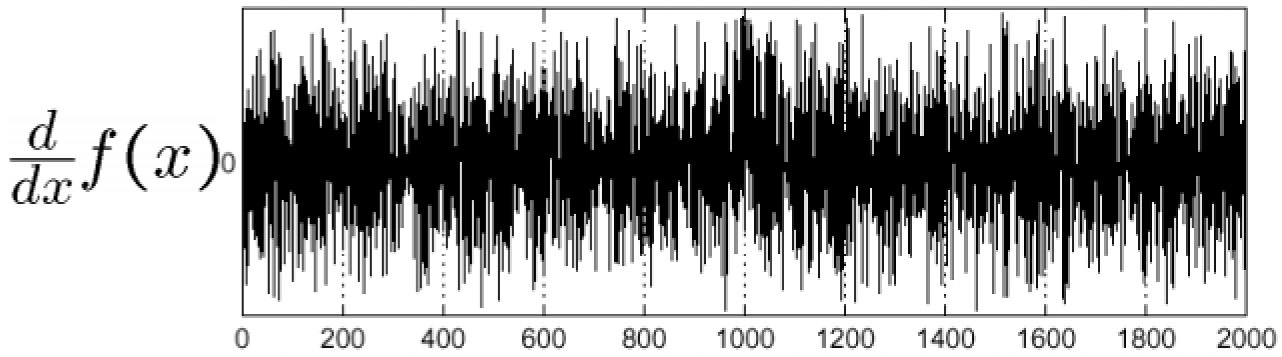


smaller peaks are lost in the noise

Slide credit: D. Hoiem

## Another 1D example

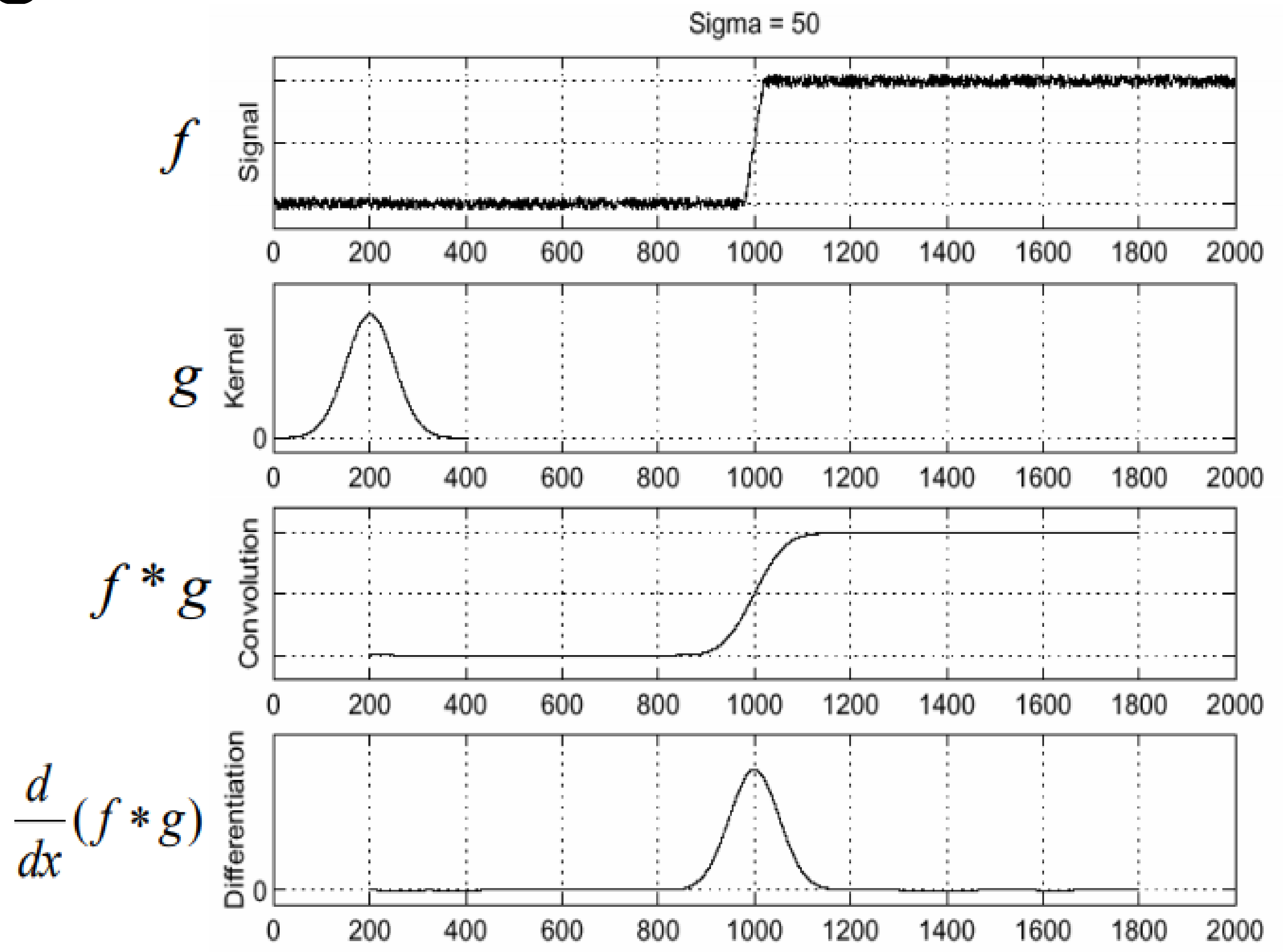




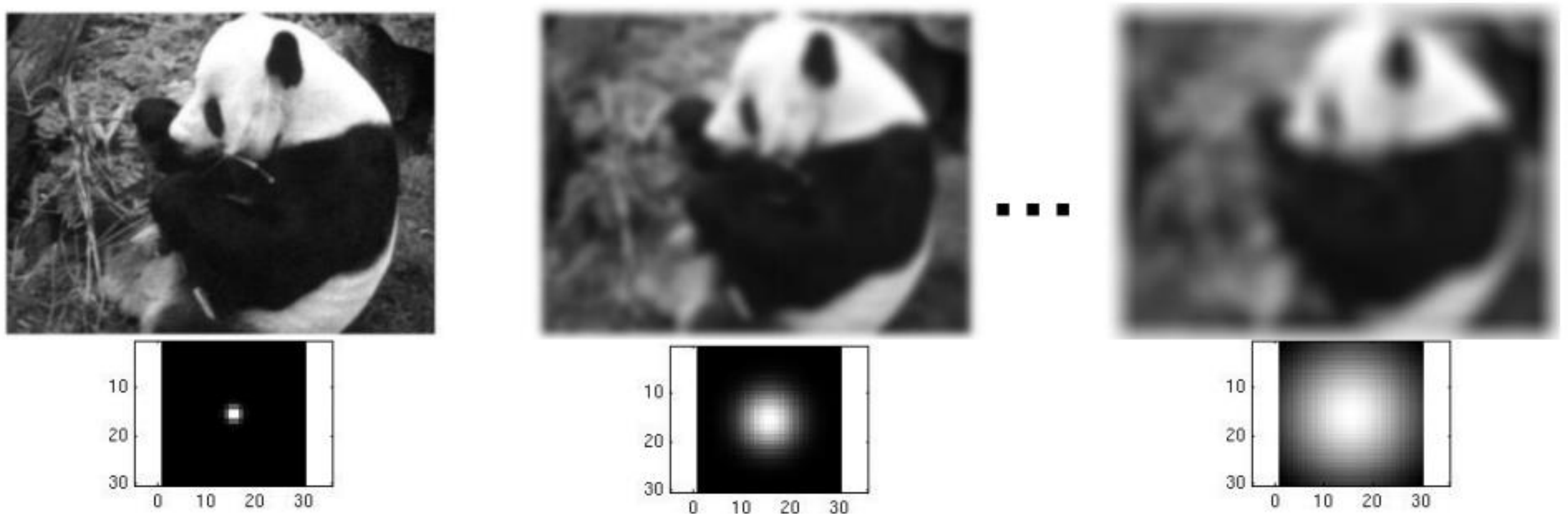
Where is the edge?!?

Slide credit: S. Seitz

## Solution: smooth, then compute gradient



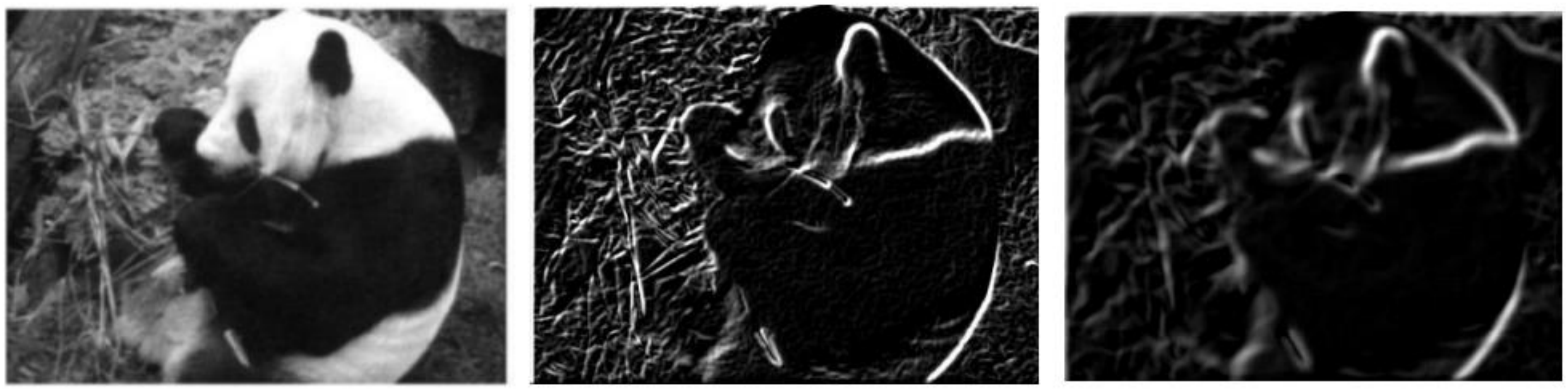
# Different amounts of smoothing...



Amount of smoothing depends on the value of sigma (width of the gaussian)

Slide credit: K. Grauman

## .... show different structures in the gradients


σ = 1 pixel σ = 3 pixels

Larger values of sigma: larger scale edges detected

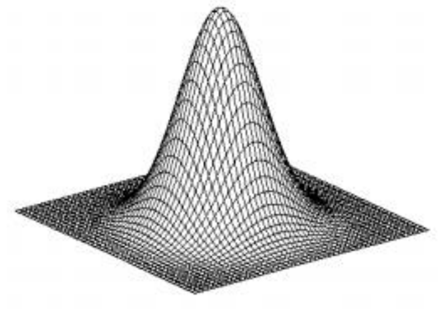
Smaller values of sigma: finer details detected

Slide credit: K. Grauman

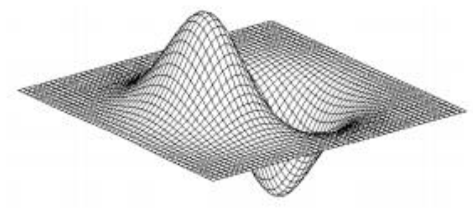
## In practice

To compute gradients, convolve with a derivative-of-gaussian filter.

This is equivalent to smoothing with a gaussian and then taking the derivative.

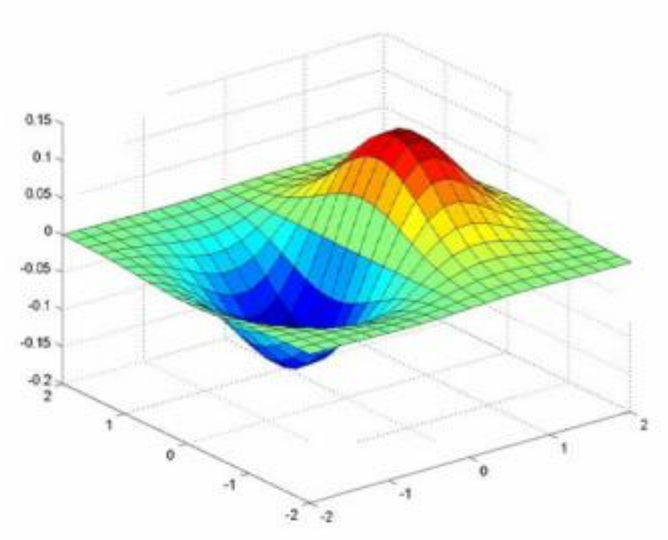


* [1 -1] =

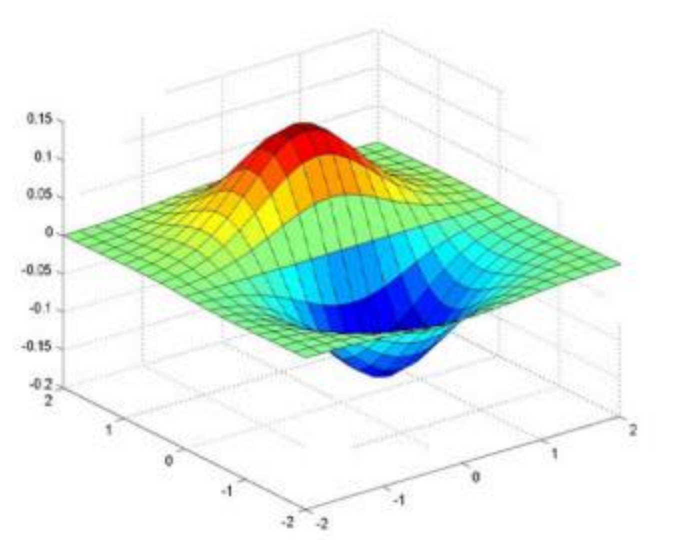


Slide credit:  S. Lazebnik

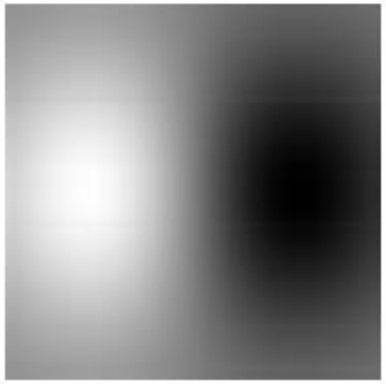
## Derivative-of-gaussian filters

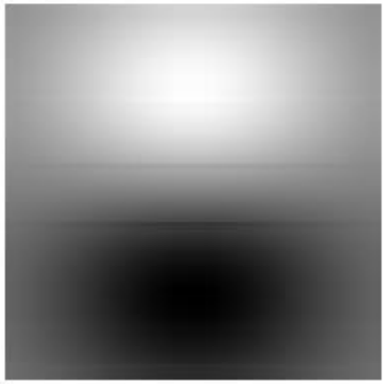


*x*-direction



*y*-direction





Slide credit: S. Lazebnik

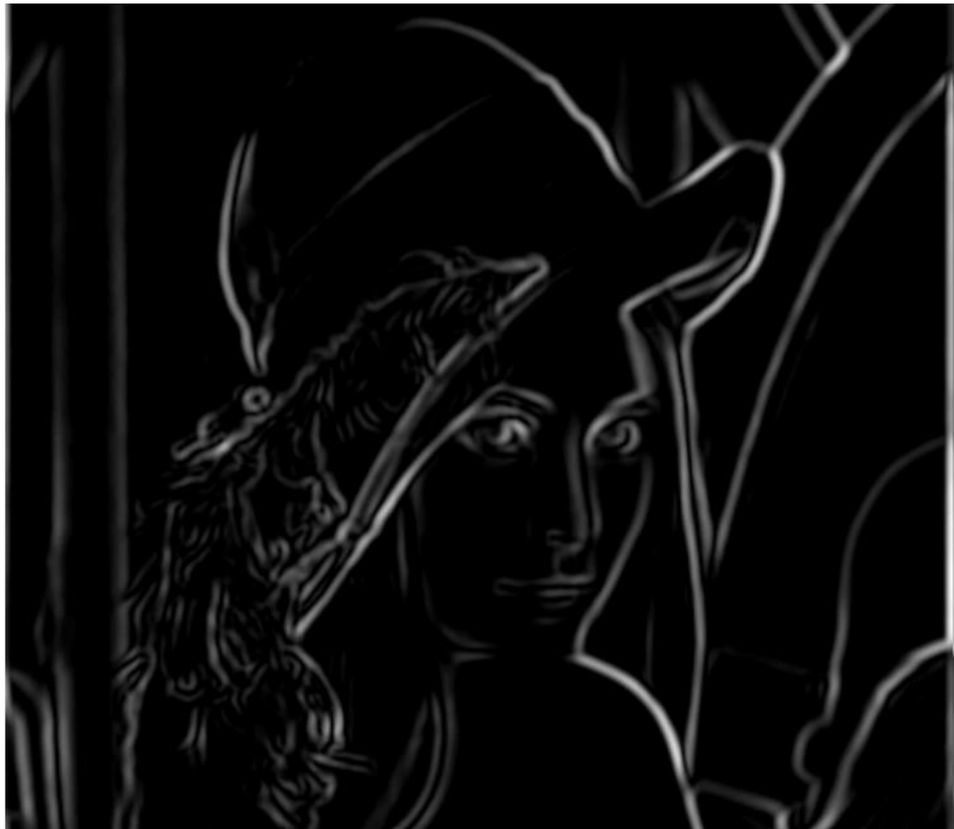
## Notes

- Gaussian smoothing filter
  - removes “high-frequency” components
  - values sum to one
- Derivative filters
  - contain some negative values
  - values sum to 0
  - yield large responses at points with high contrast (e.g. edges)

## Edge detection summary

$\frac{\partial f}{\partial x}$ $\frac{\partial f}{\partial y}$

- Approximate gradients along x and y by convolving with derivative-of-gaussian filters
- Compute gradient magnitude as $\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$



# Canny edge detection algorithm

$\frac{\partial f}{\partial x}$ $\frac{\partial f}{\partial y}$

- Approximate gradients along x and y by convolving with derivative-of-gaussian filters
- Compute gradient magnitude as $\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$
- Make edges 1-pixel-wide (thinning):

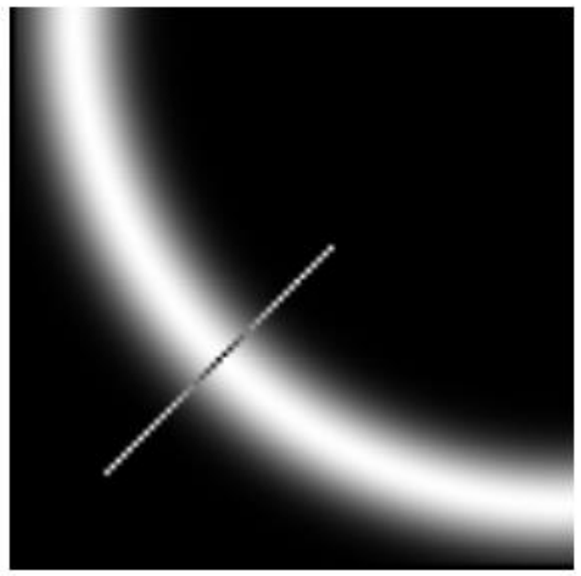
  **non-maxima suppression** along perpendicular direction to edge, i.e. direction of gradient
- Only keep strong edges

  **hysteresis thresholding**

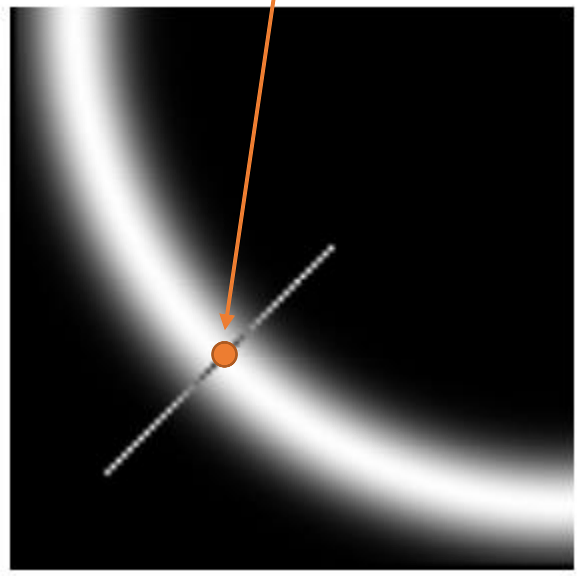
# Non-maxima suppression

Intuitively:

of all the possible edge pixels along the line...



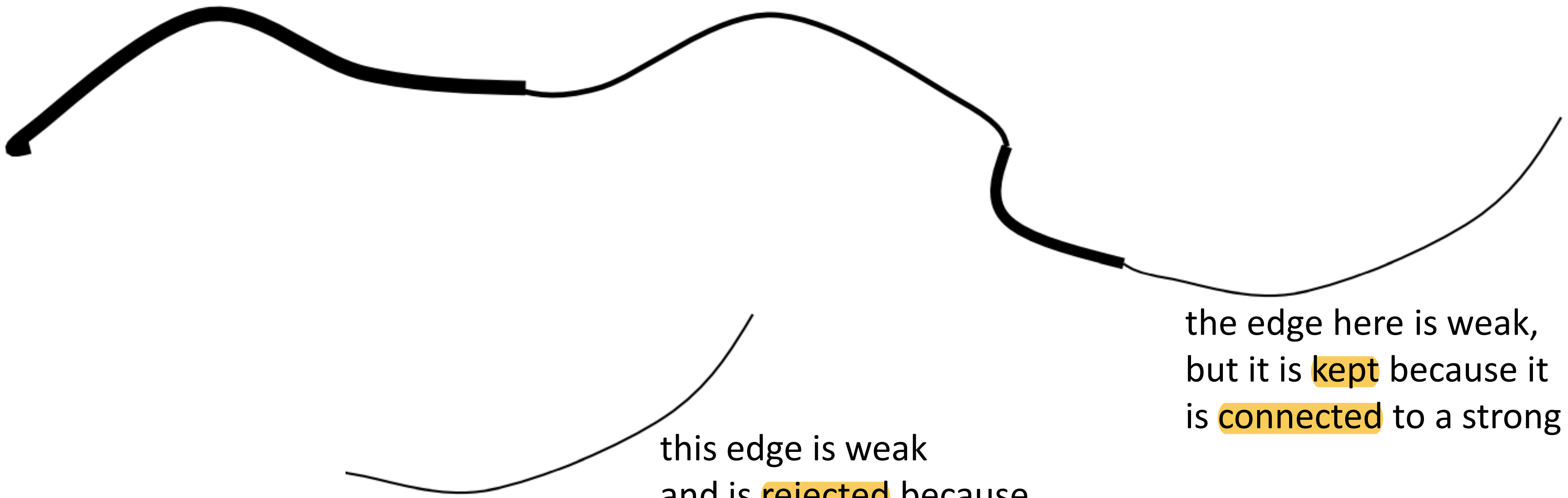
keep only this one



... then repeat along the whole edge.  Results in thinning the edge to a 1-pixel width

## Hysteresis thresholding

Intuitively:
Use a high threshold to start curves and a low threshold to continue them

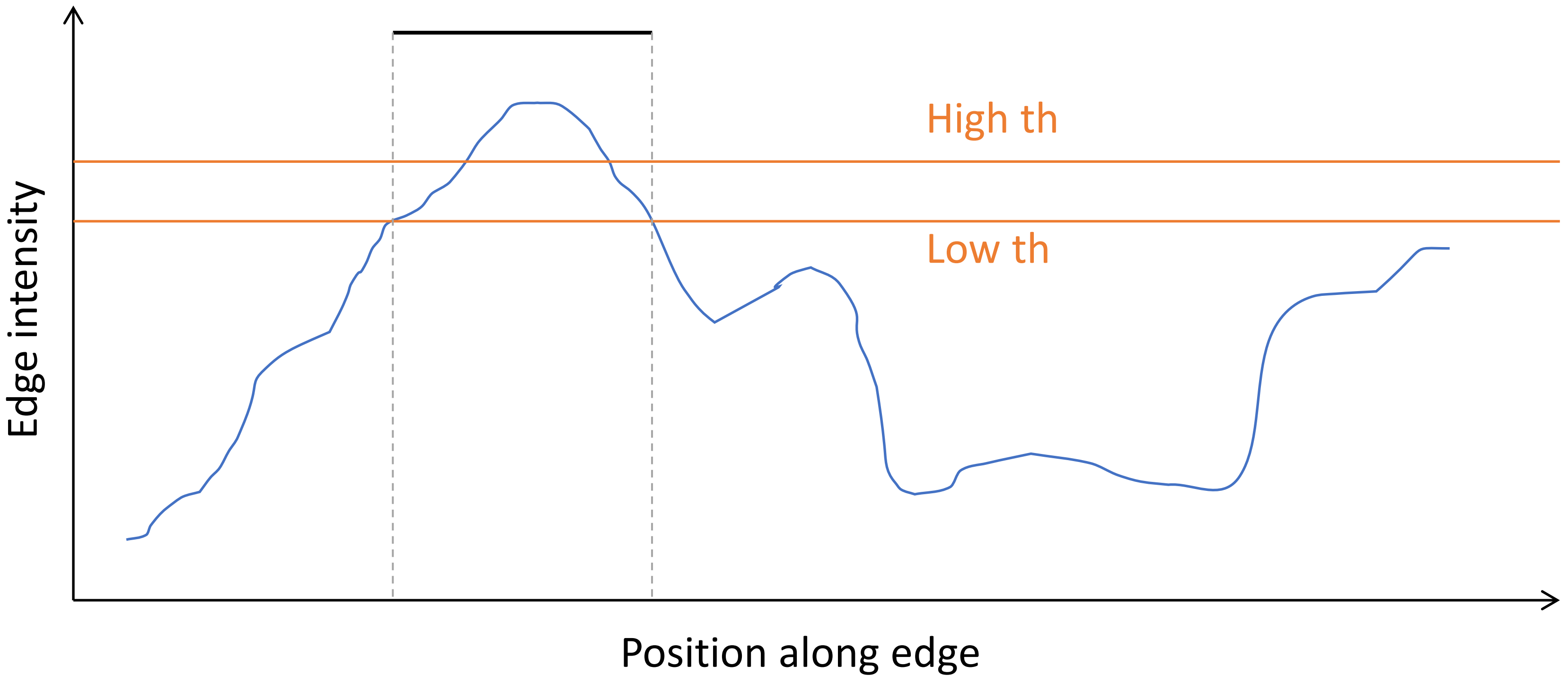


the edge here is weak,
but it is kept because it
is connected to a strong edge

this edge is weak
and is rejected because
it is not connected to any strong edge

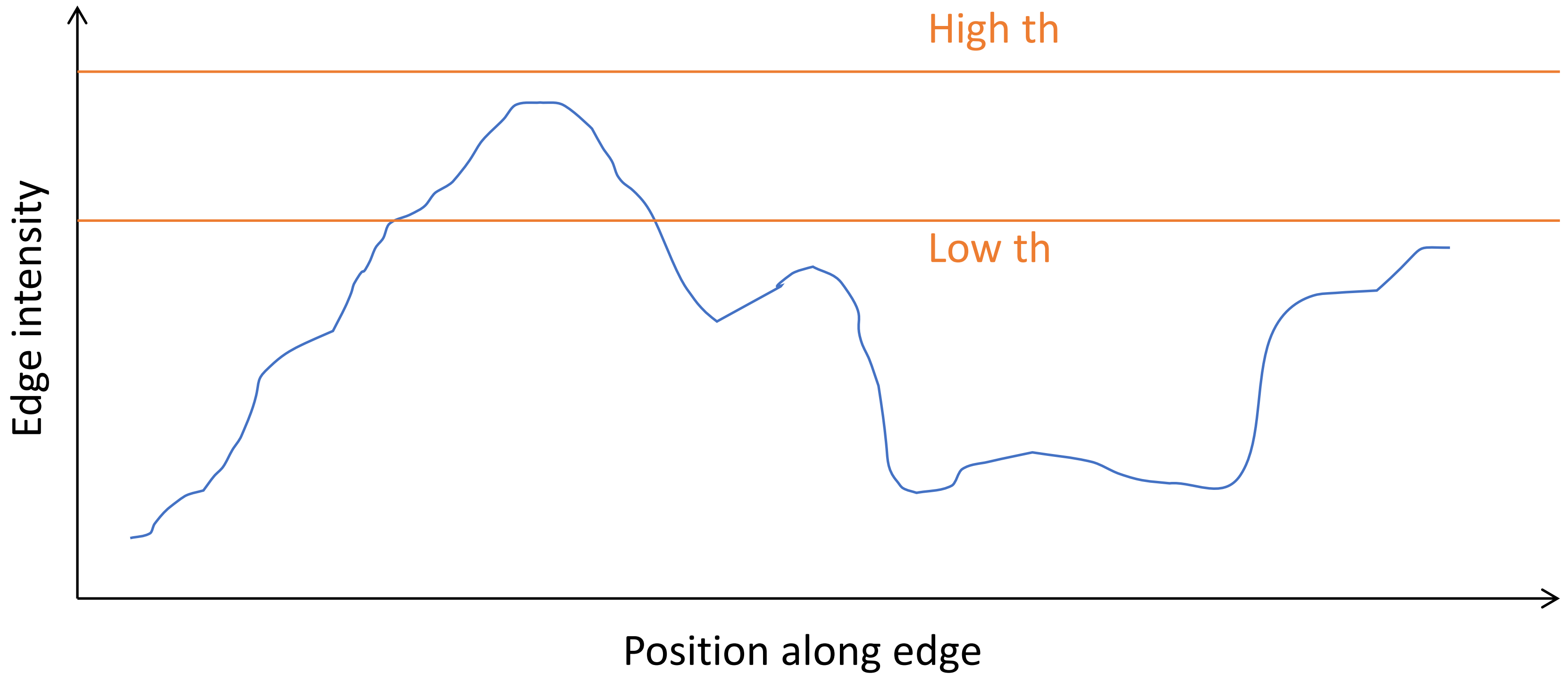
# Hysteresis thresholding

Quiz: which parts of the edge would survive?



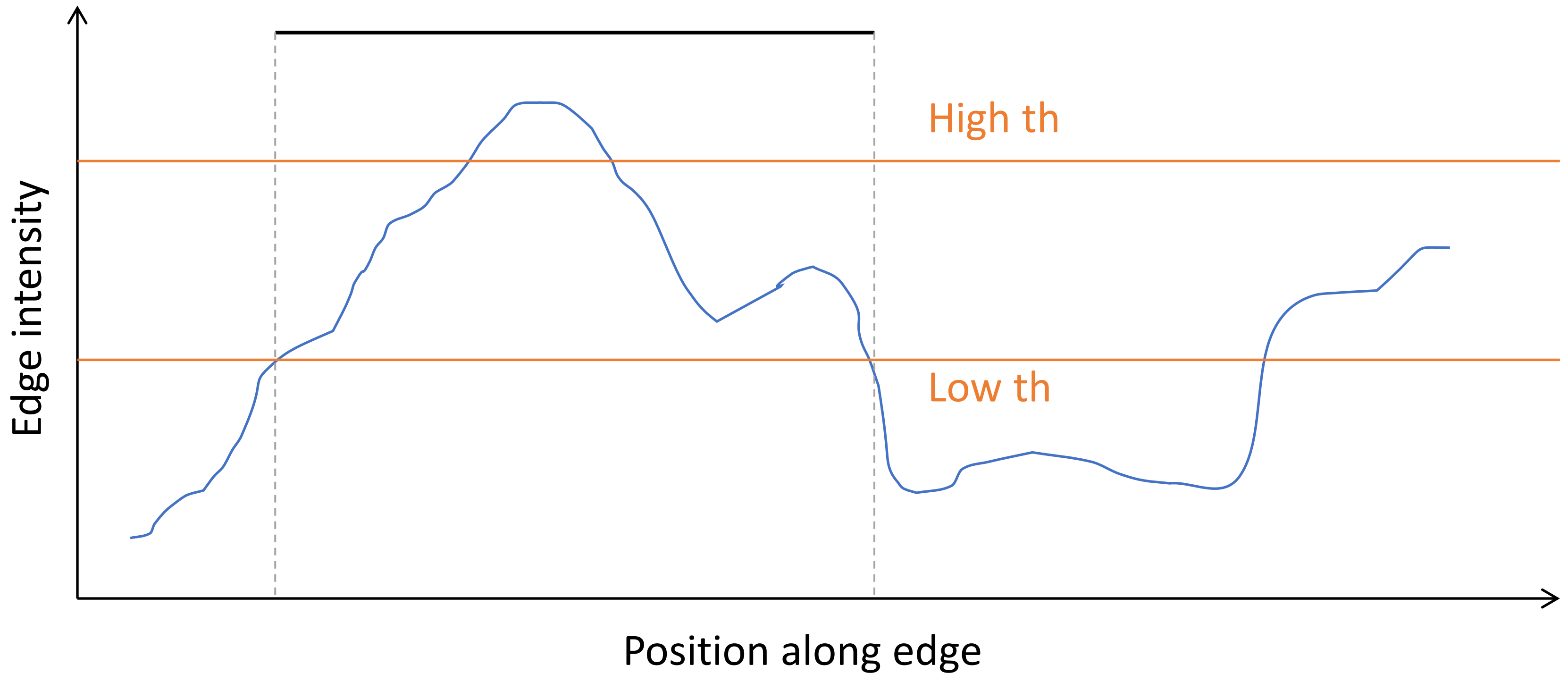
# Hysteresis thresholding

Quiz: which parts of the edge would survive? None!



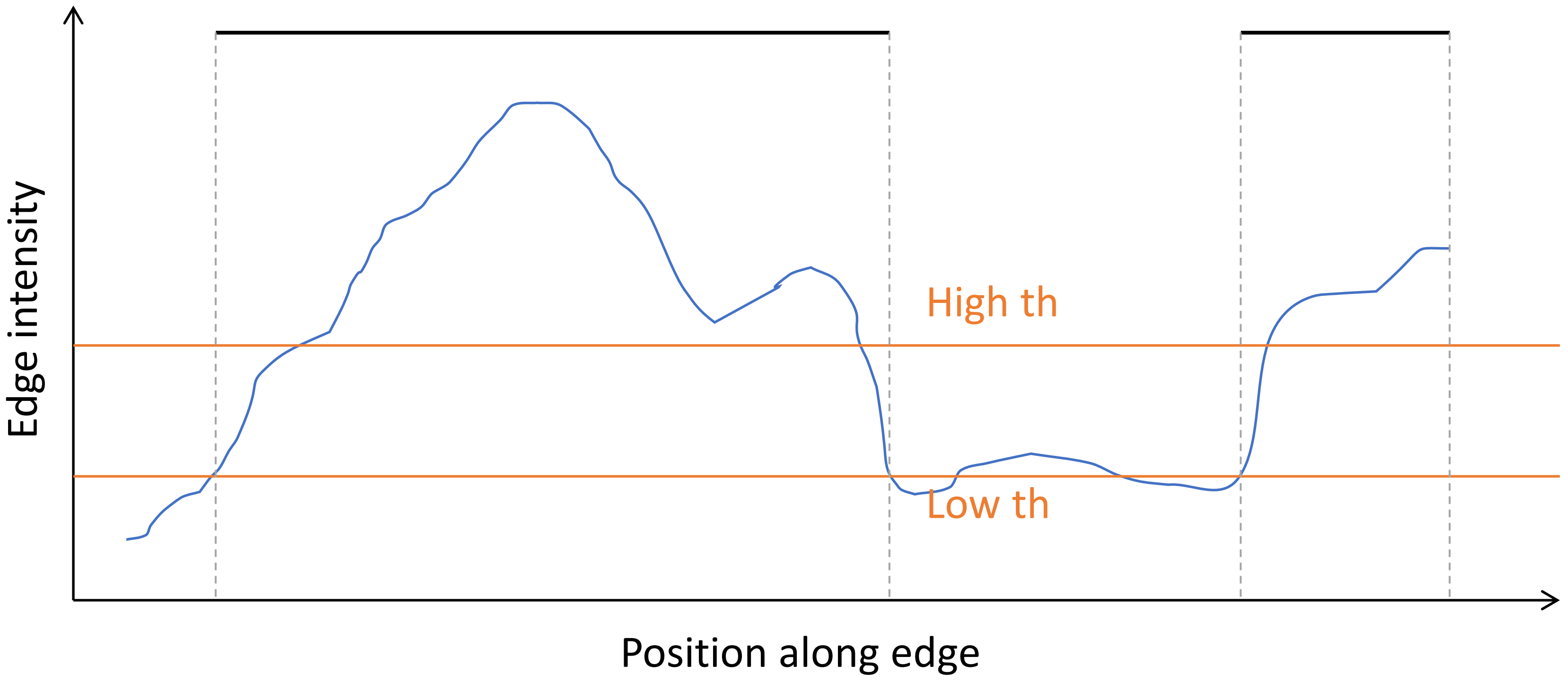
# Hysteresis thresholding

Quiz: which parts of the edge would survive?

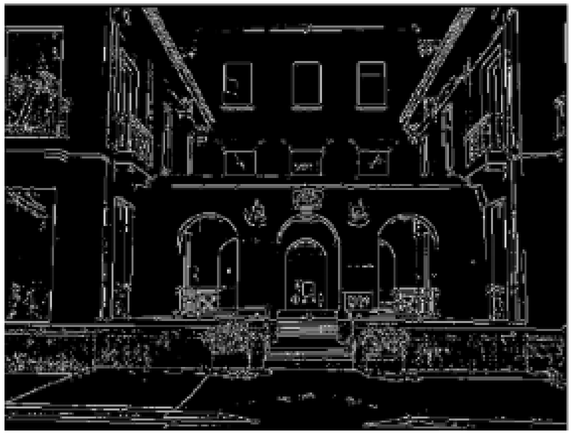


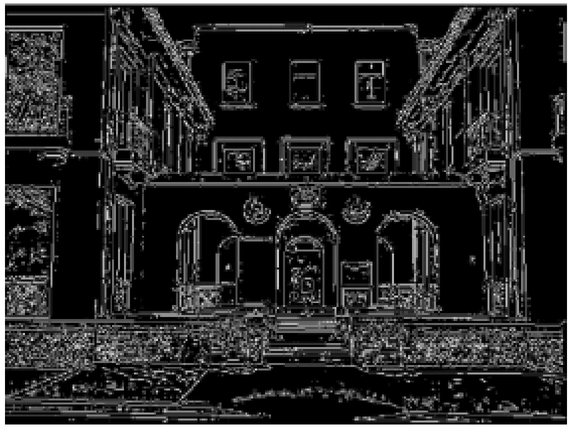
# Hysteresis thresholding

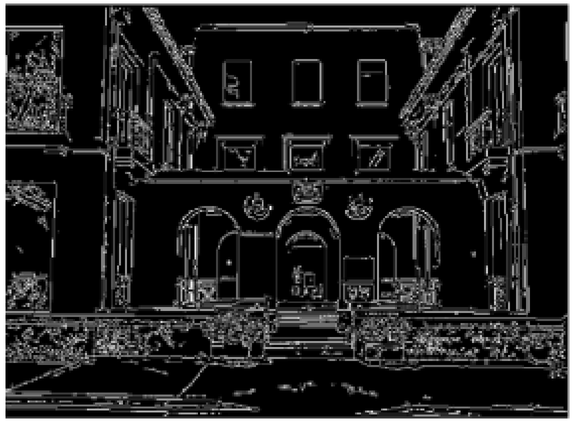
Quiz: which parts of the edge would survive?



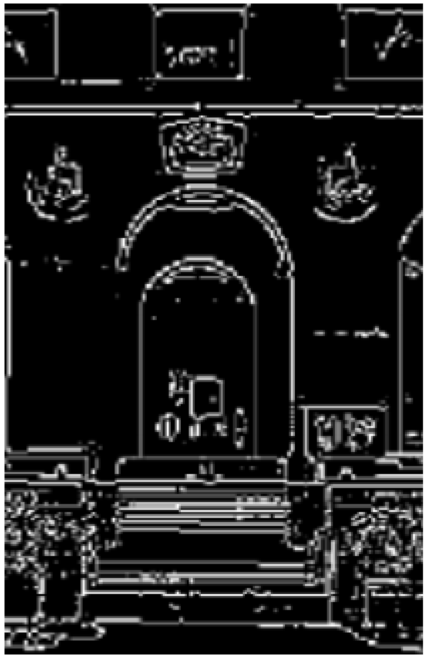
original image

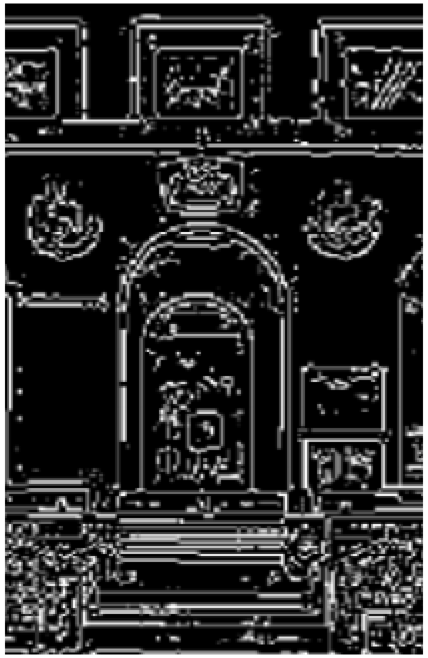

high threshold
(strong edges)

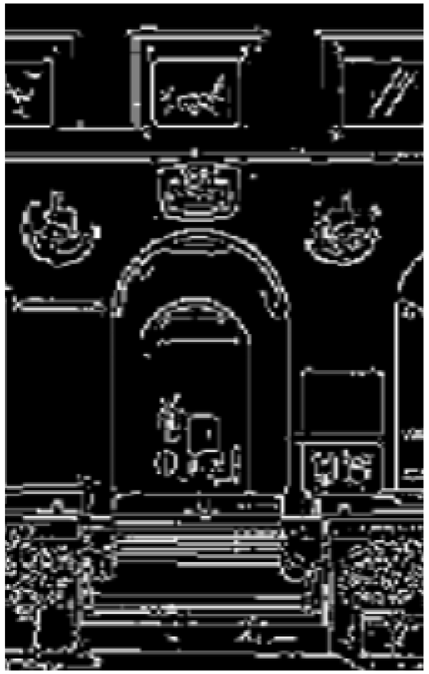

low threshold
(weak edges)


hysteresis threshold

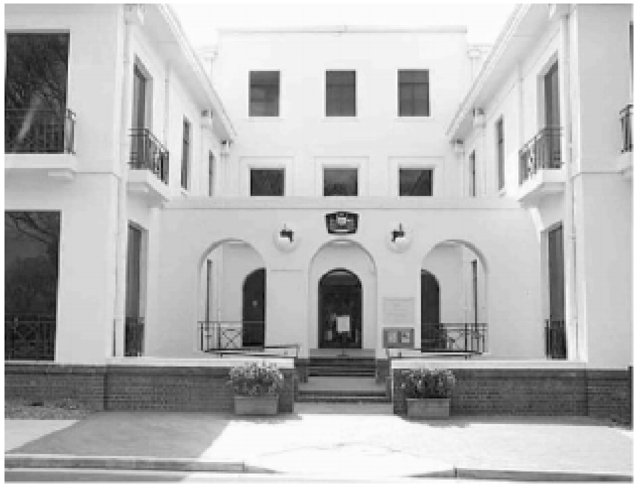
Slide credit: L. Fei-Fei


high threshold
(strong edges)

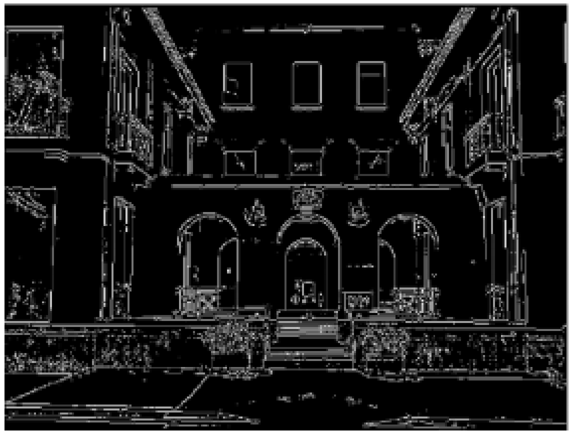

low threshold
(weak edges)


hysteresis threshold

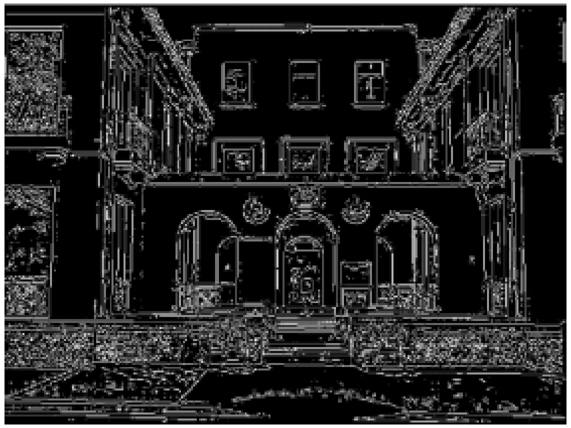
Slide credit: L. Fei-Fei



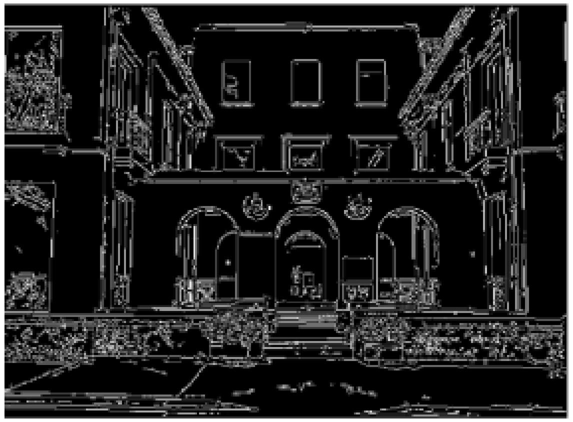
original image



high threshold
(strong edges)



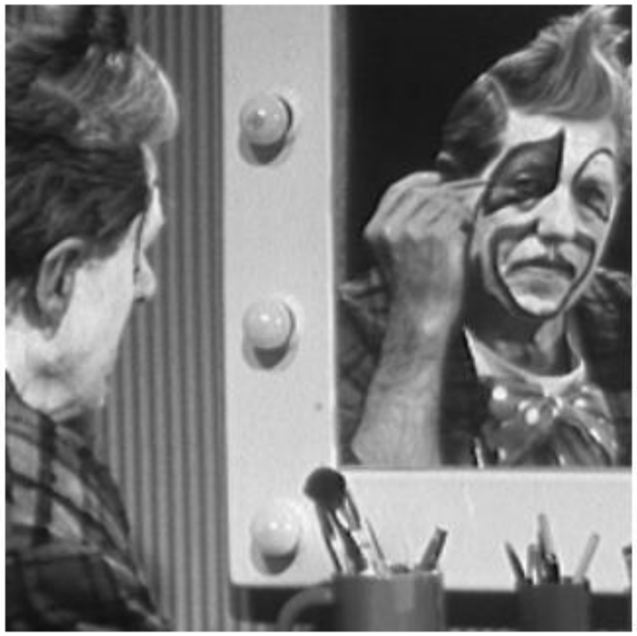
low threshold
(weak edges)

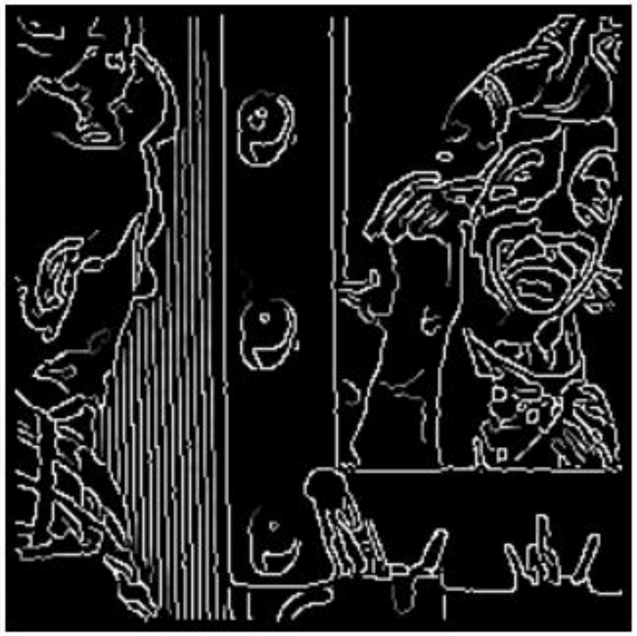


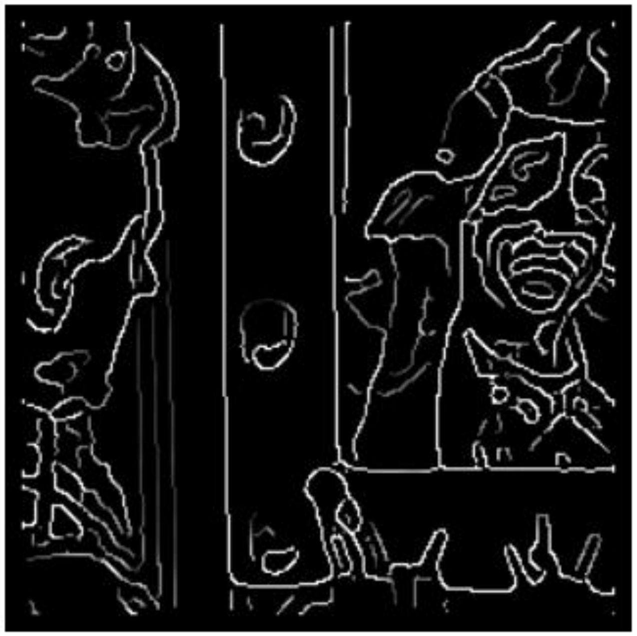
hysteresis threshold

Slide credit: L. Fei-Fei

# Effect of gaussian sigma


original


Canny with $\sigma = 1$


Canny with $\sigma = 2$

# canny

`skimage.feature.canny`(*image*, *sigma=1.0*, *low_threshold=None*, *high_threshold=None*, *mask=None*, *use_quantiles=False*) [source]

Edge filter an image using the Canny algorithm.

**Parameters:**

**image : 2D array**

Grayscale input image to detect edges on; can be of any dtype.

**sigma : float**

Standard deviation of the Gaussian filter.

**low_threshold : float**

Lower bound for hysteresis thresholding (linking edges). If None, low_threshold is set to 10% of dtype's max.

**high_threshold : float**

Upper bound for hysteresis thresholding (linking edges). If None, high_threshold is set to 20% of dtype's max.

**mask : array, dtype=bool, optional**

Mask to limit the application of Canny to a certain area.

**use_quantiles : bool, optional**

If True then treat low_threshold and high_threshold as quantiles of the edge magnitude image, rather than absolute edge magnitude values. If True then the thresholds must be in the range [0, 1].

**Returns:**

**output : 2D array (image)**

The binary edge map.