

# Advanced Image Processing

## Edge detection

# Review of Filtering

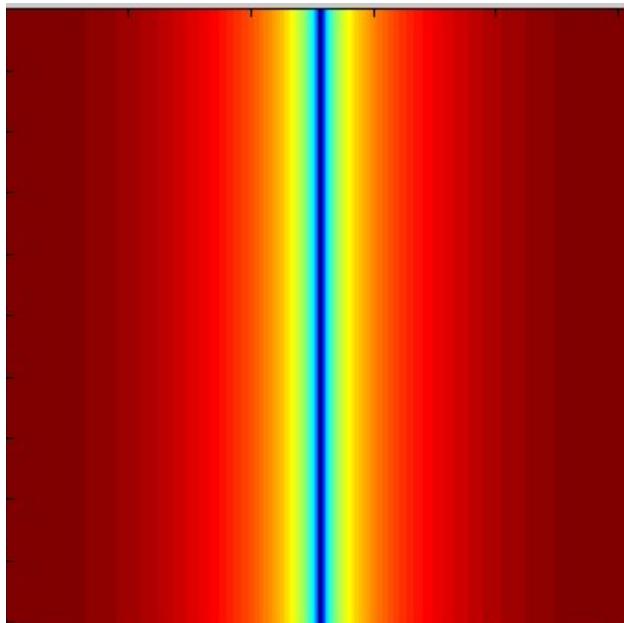
- Filtering in frequency domain
  - Can be faster than filtering in spatial domain (for large filters)
  - Can help understand effect of filter
  - Algorithm:
    1. Convert image and filter to Fourier domain (e.g., `numpy.fft.fft2()`)
    2. Element-wise multiply their decompositions
    3. Convert result to spatial domain with inverse Fourier transform (e.g., `numpy.fft.ifft2()`)

# Review of Filtering

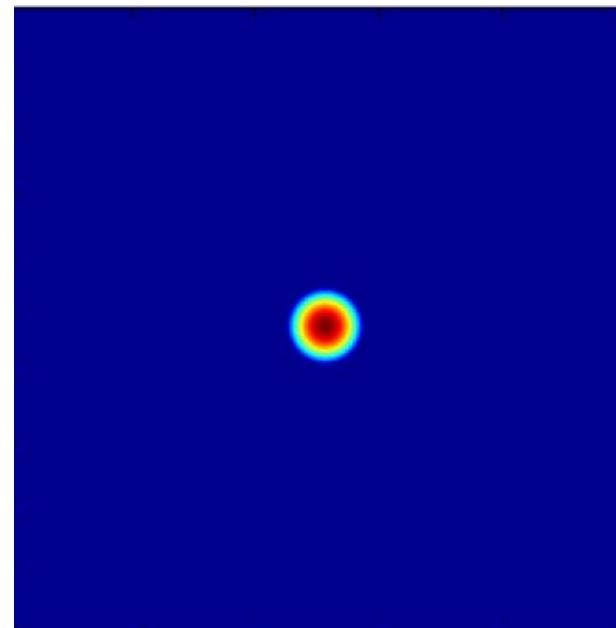
- Linear filters for basic processing
  - Edge filter (high-pass)
  - Gaussian filter (low-pass)

Blue is 0 and red is closer to high value.

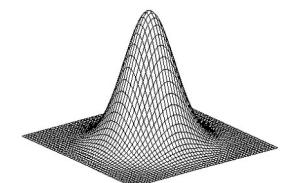
$$[-1 \ 0 \ 1]$$



FFT of Gradient Filter

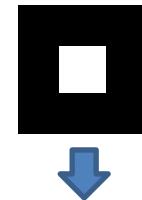


FFT of Gaussian



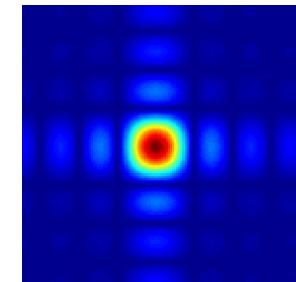
Gaussian

# Things to Remember



Sometimes it helps to think of images and filtering in the frequency domain

- Fourier analysis

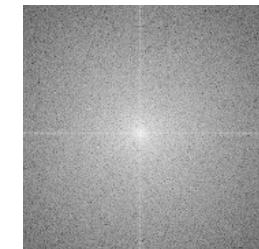


Can be faster to filter using FFT for large images

- $N \log N$  vs.  $N^2$  for convolution/correlation

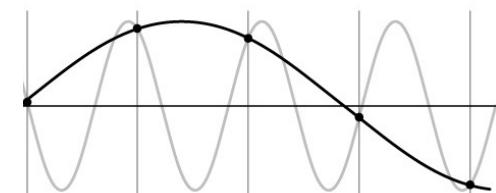
Images are mostly smooth

- Basis for compression



Remember to low-pass before sampling

- Otherwise you create aliasing



# **EDGE / BOUNDARY DETECTION**

# Edge detection

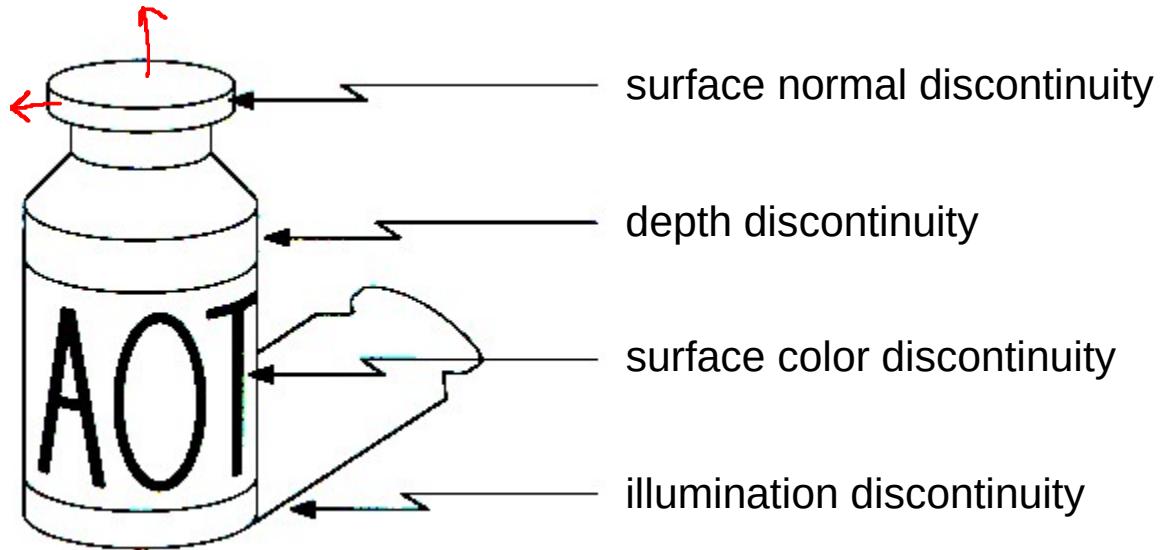
**Goal:** Identify visual changes  
**(discontinuities)** in an image.

Intuitively, semantic information  
encoded in edges.

What are some 'causes'  
of visual edges?



# Origin of Edges

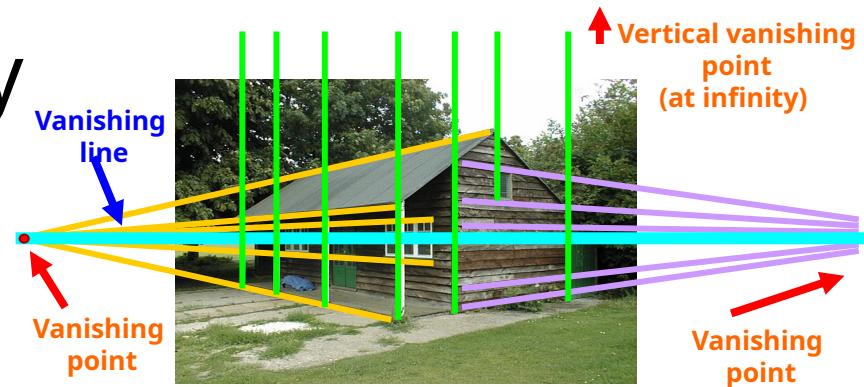


- Edges are caused by a variety of factors

# Why do we care about edges?

Extract information  
– Recognize objects

Help recover geometry  
and viewpoint

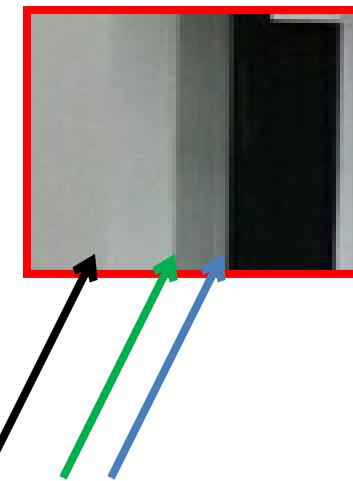


# Closeup of edges



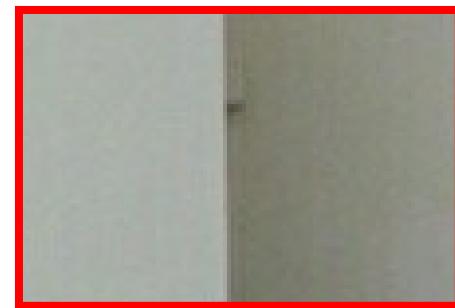
Source: D. Hoiem

# Closeup of edges



Source: D. Hoiem

# Closeup of edges



Source: D. Hoiem

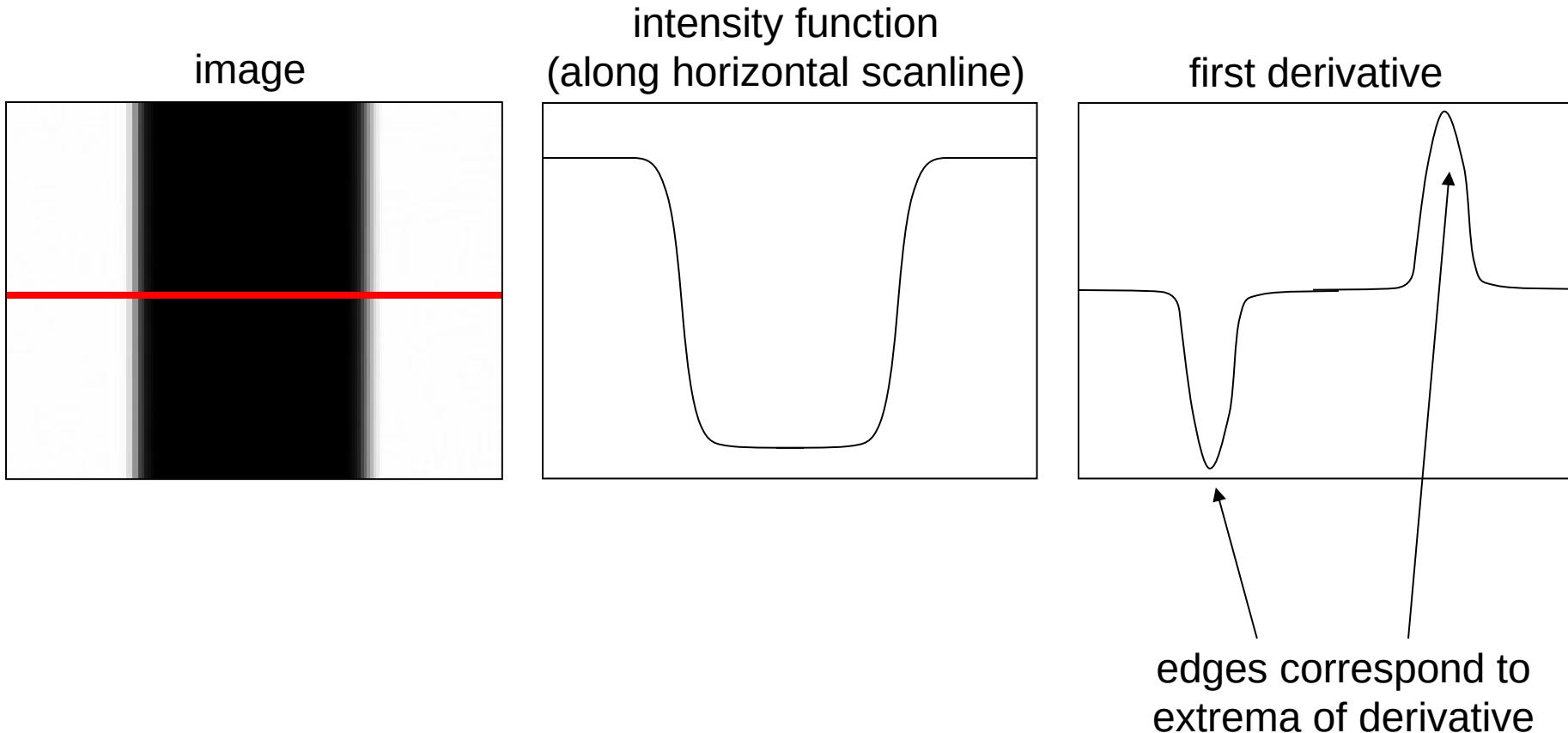
# Closeup of edges



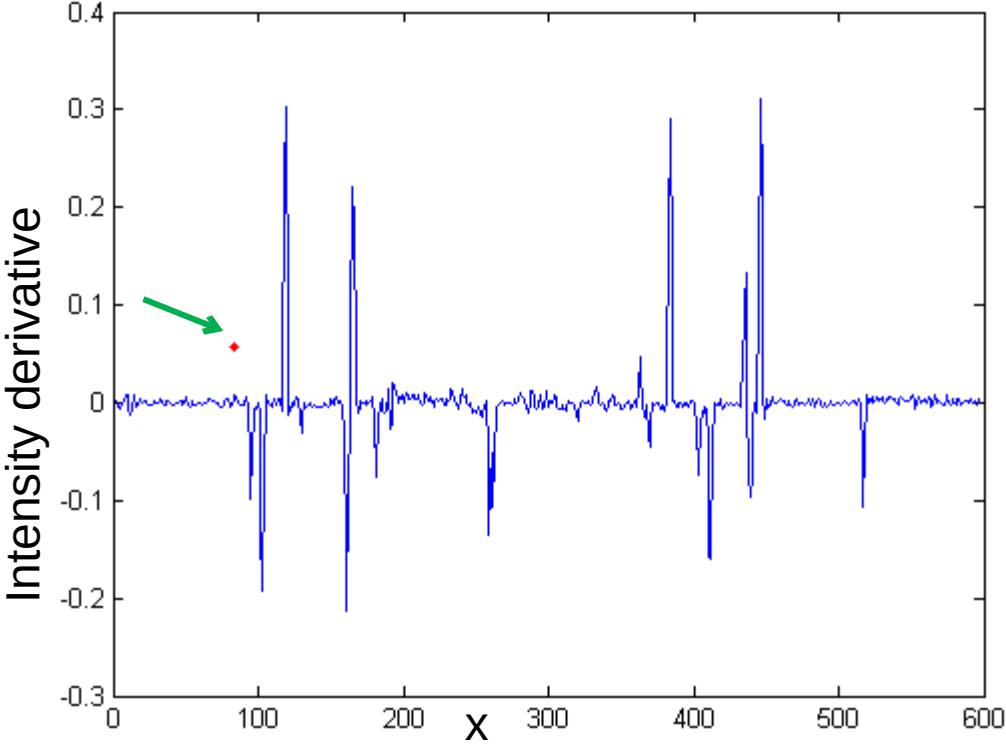
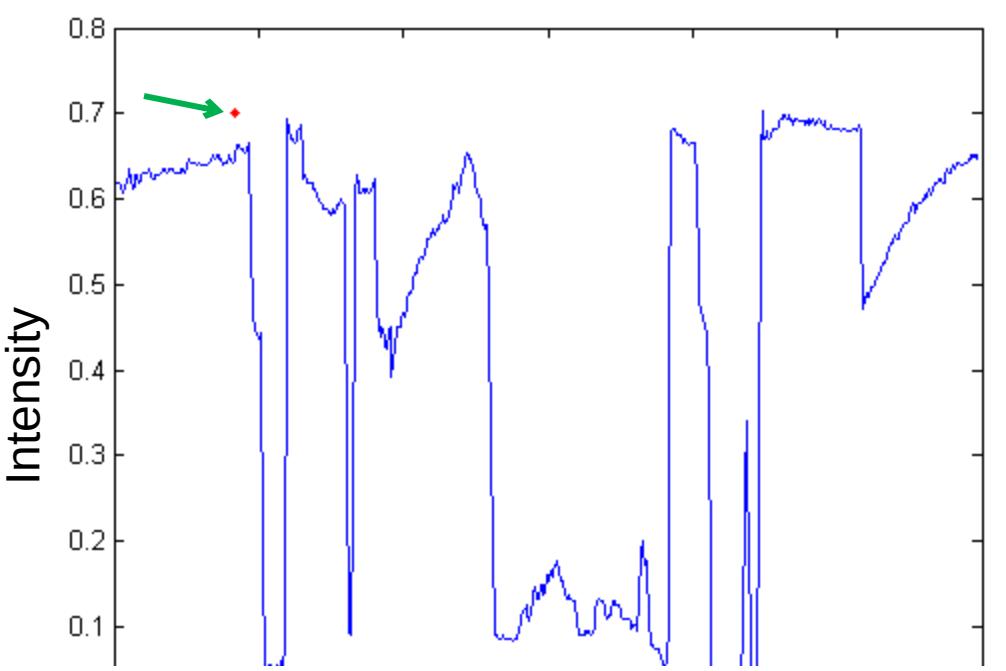
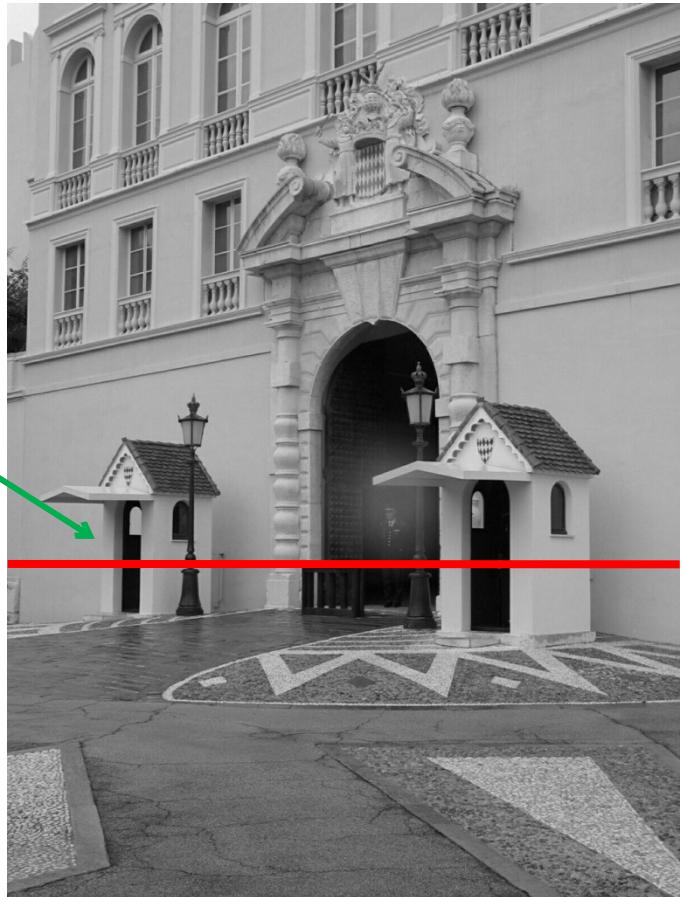
Source: D. Hoiem

# Characterizing edges

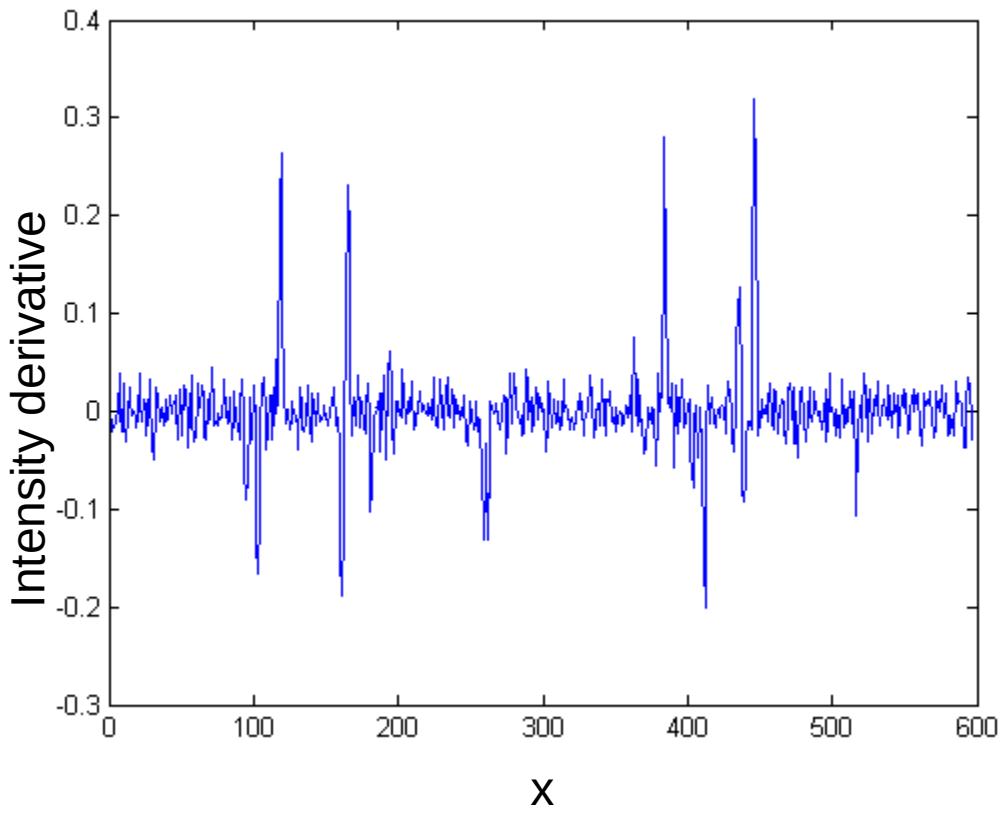
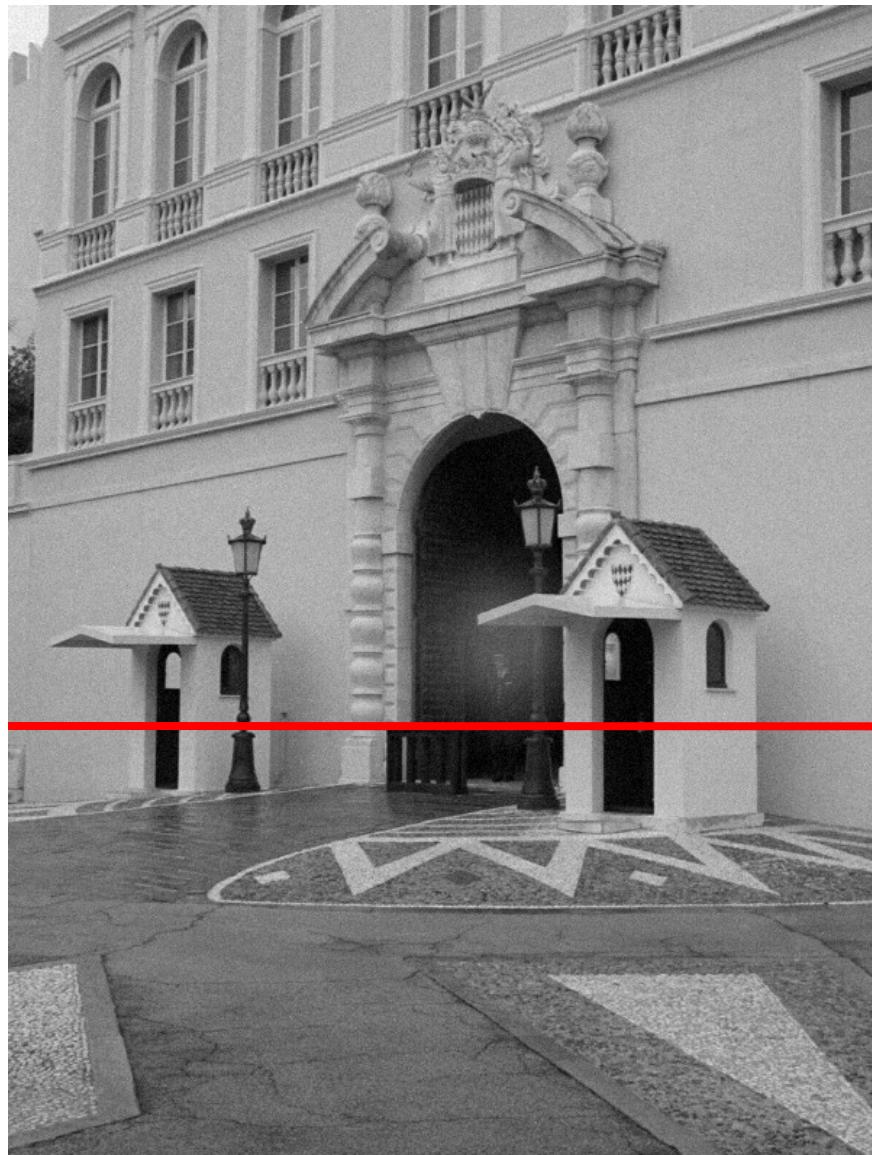
- An edge is a **place of rapid change** in the image intensity function



# Intensity profile



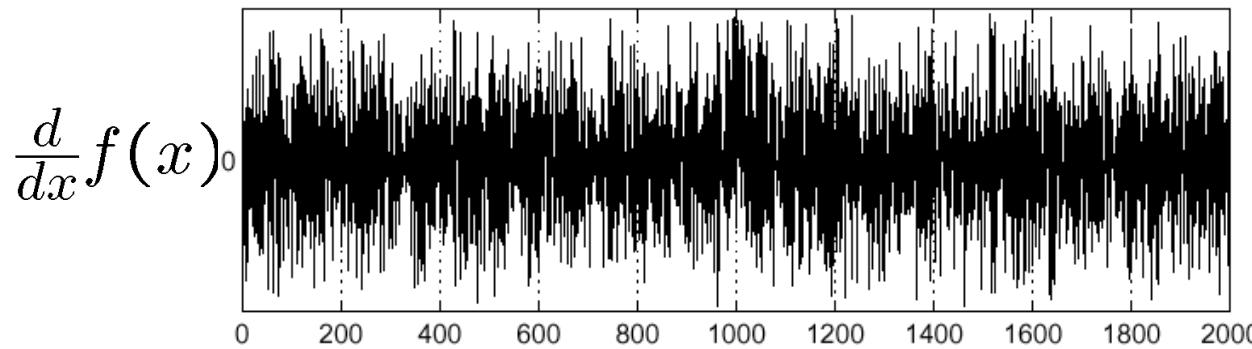
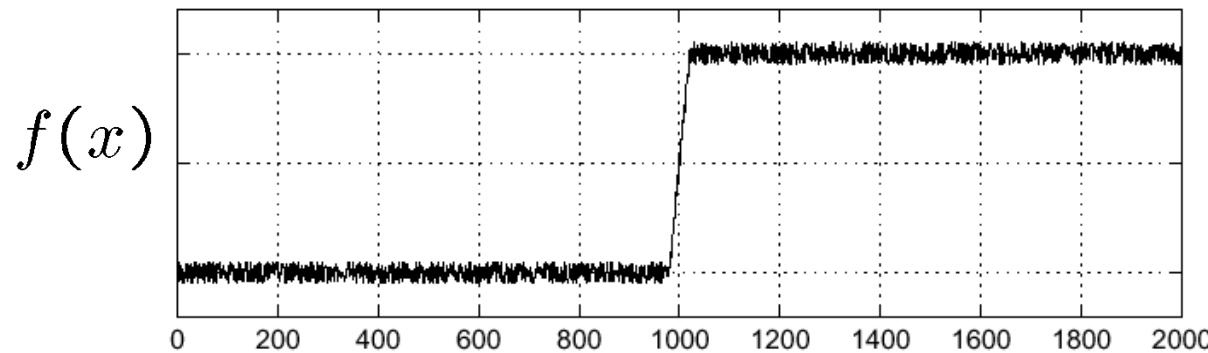
# With a little Gaussian noise



Source: D. Hoiem

# Effects of noise

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



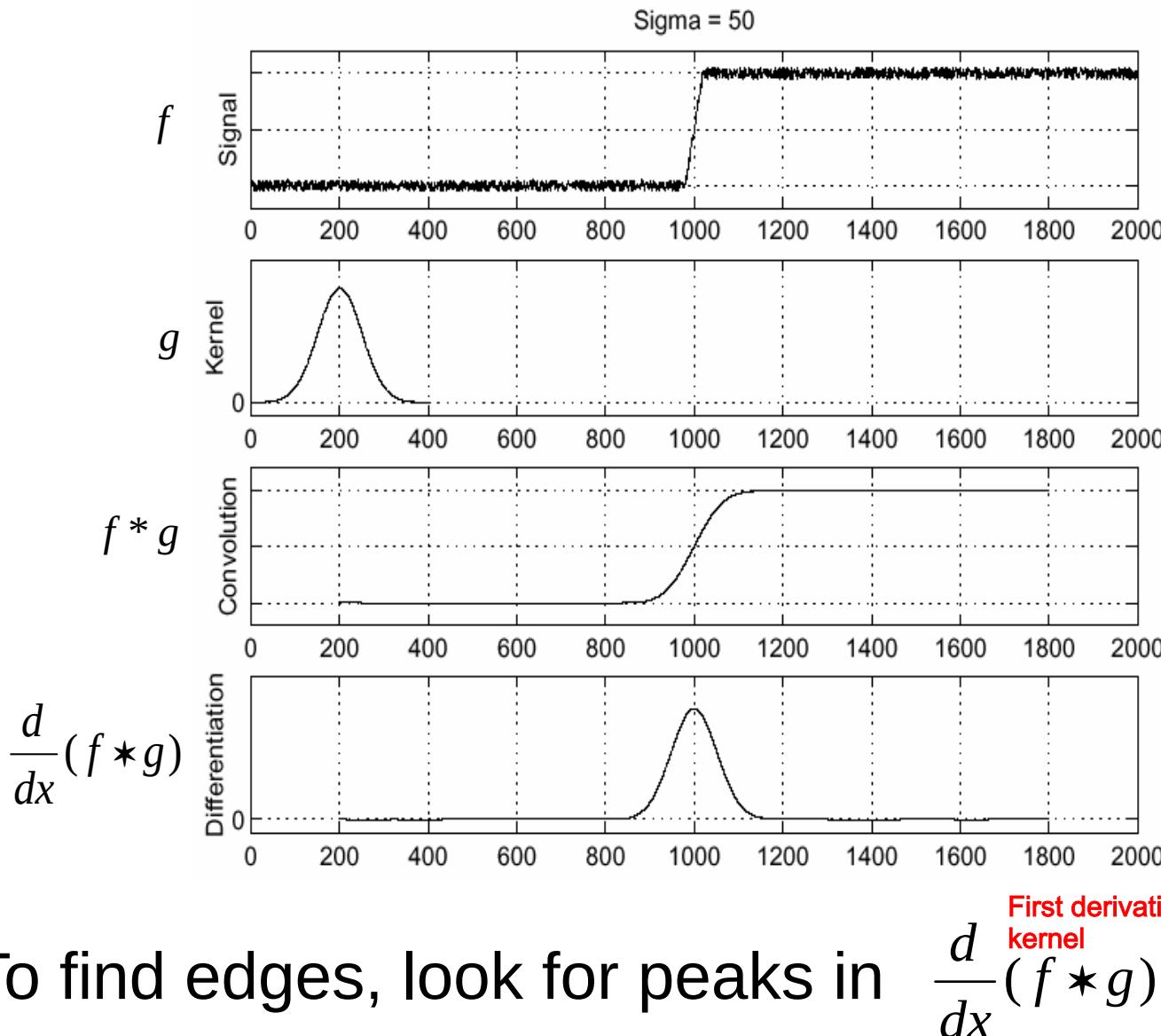
Where is the edge?

Source: S. Seitz

# Effects of noise

- Difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the stronger the response
- What can we do about it?

# Solution: smooth first



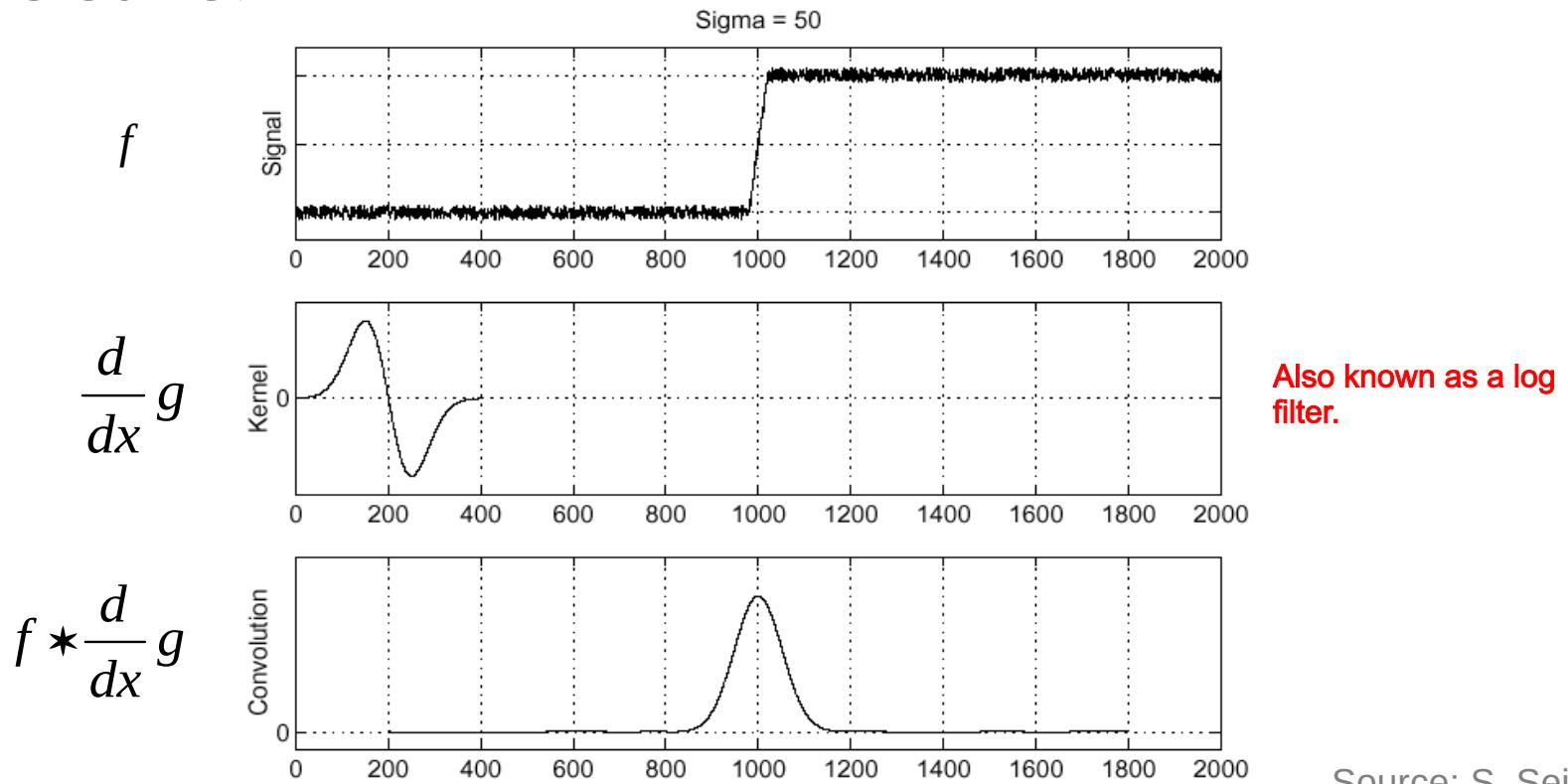
Source: S. Seitz

# Derivative theorem of convolution

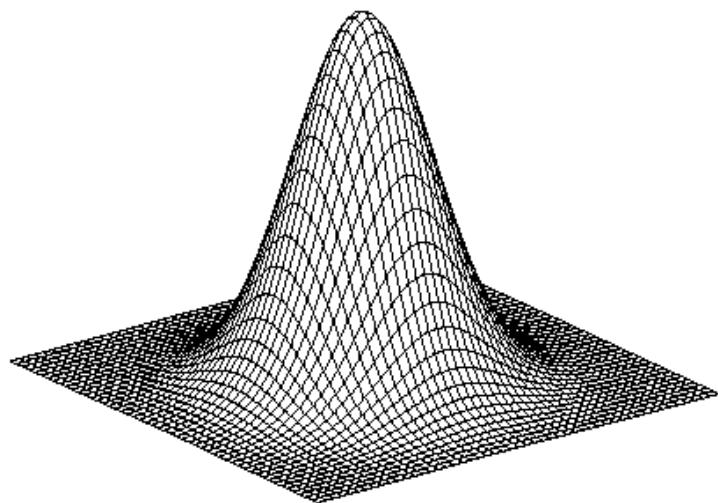
- Convolution is differentiable:

$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$$

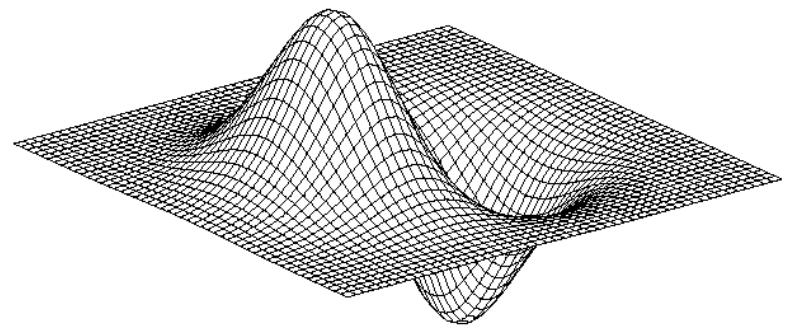
- This saves us one operation:



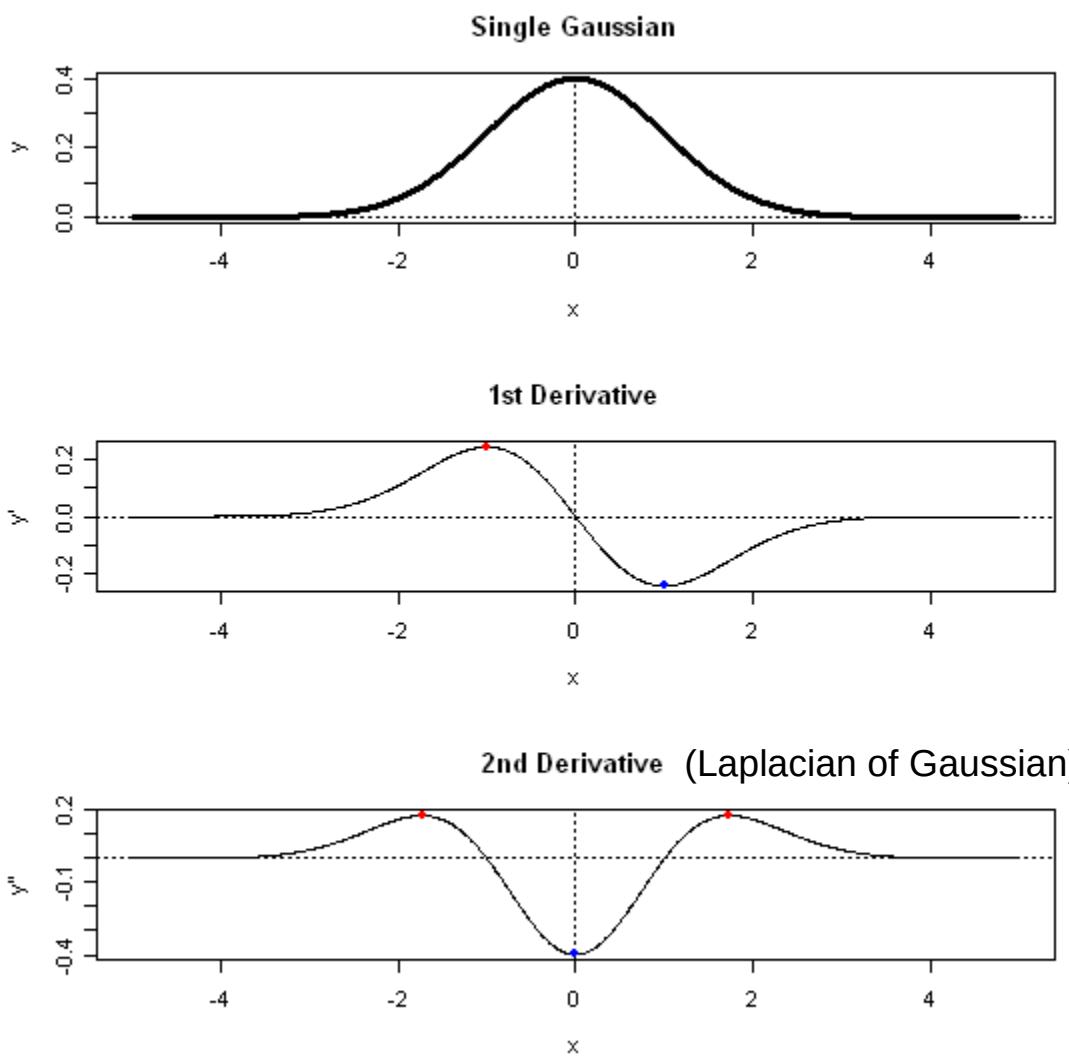
# Derivative of 2D Gaussian filter



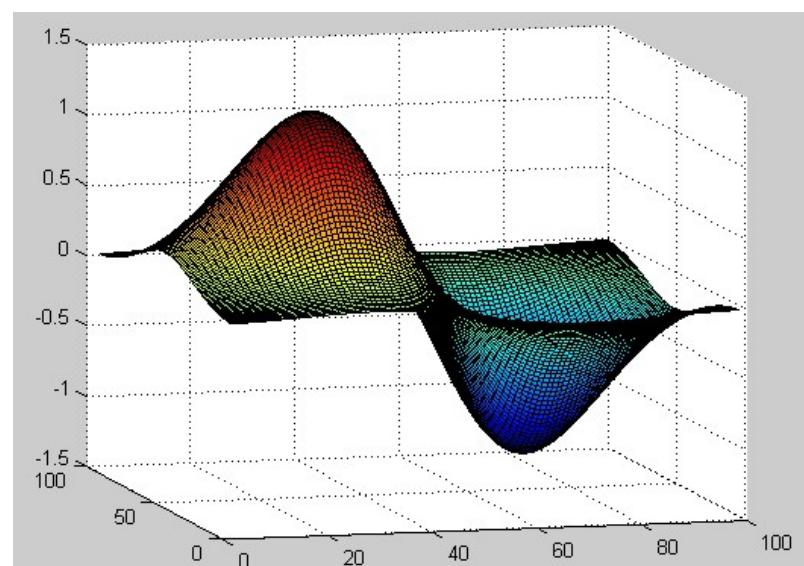
$$* [1 \ -1] =$$



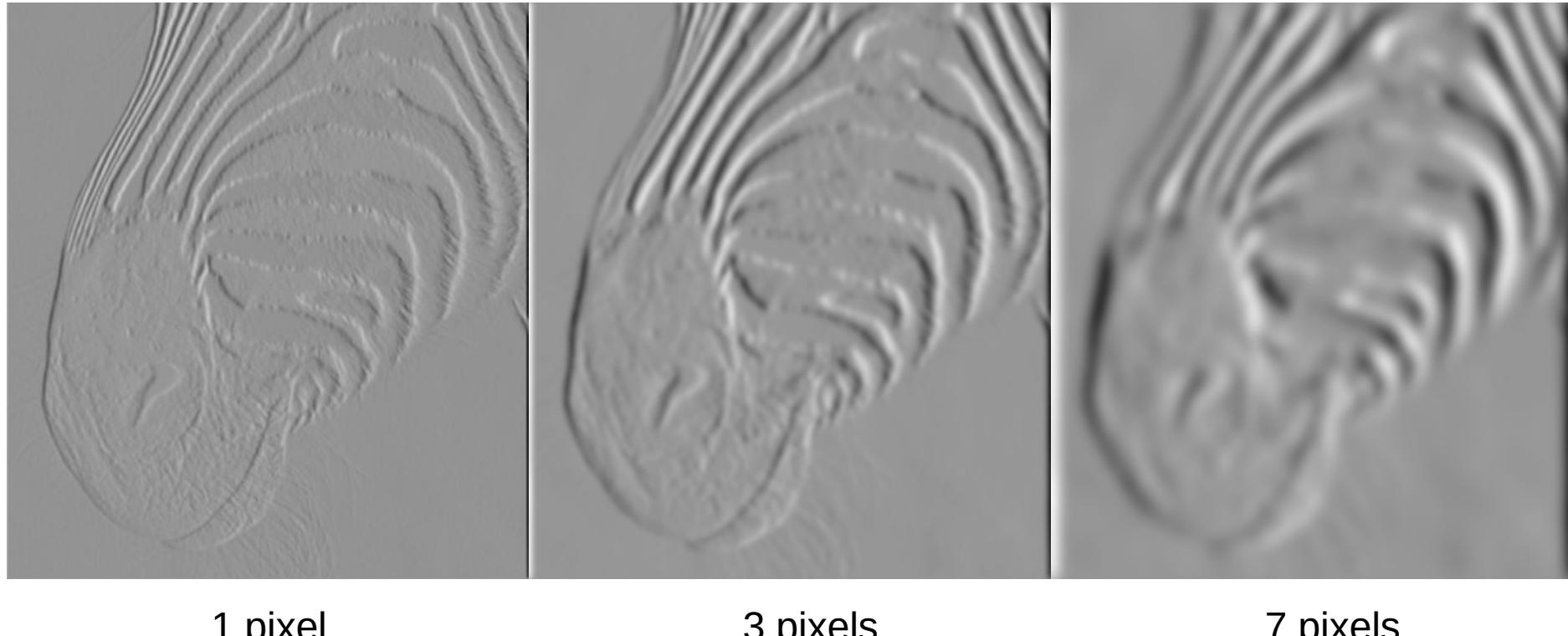
# More Useful Filters



1<sup>st</sup> Derivative of Gaussian



# Tradeoff between smoothing and localization



- Smoothed derivative removes noise, but blurs edge.

# Designing an edge detector

What is a good edge detector?

Do we lose information when we look at edges?

Are edges 'complete' as a representation of images?

# Designing an edge detector

- Criteria for a good edge detector:
  - **Good detection:** the optimal detector should find all real edges, ignoring noise or other artifacts
  - **Good localization**
    - the edges detected must be as close as possible to the true edges
    - the detector must return one point only for each true edge point
- Cues of edge detection
  - Differences in color, intensity, or texture across the boundary
  - Continuity and closure
  - High-level knowledge

# Designing an edge detector

- “All real edges”
  - We can aim to differentiate later on which edges are ‘useful’ for our applications.
  - If we can’t find all things which *could* be called an edge, we don’t have that choice.

# Elder – Are Edges Incomplete? 1999

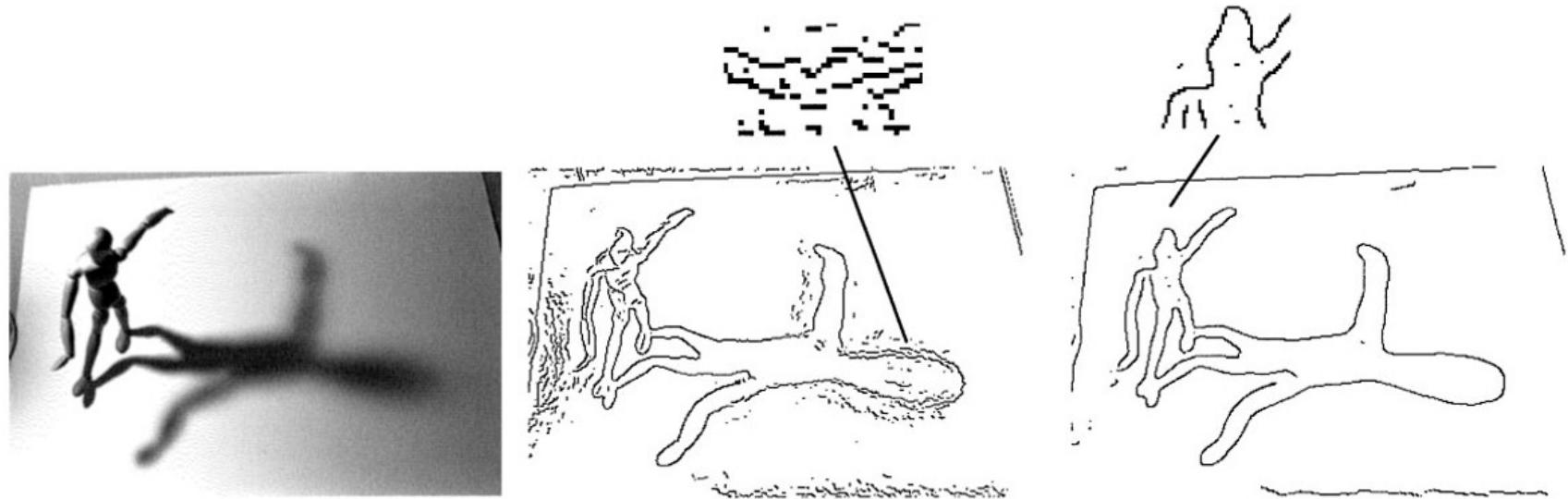


Figure 2. The problem of local estimation scale. Different structures in a natural image require different spatial scales for local estimation. The original image contains edges over a broad range of contrasts and blur scales. In the middle are shown the edges detected with a Canny/Deriche operator tuned to detect structure in the mannequin. On the right is shown the edges detected with a Canny/Deriche operator tuned to detect the smooth contour of the shadow. Parameters are  $(\alpha = 1.25, \omega = 0.02)$  and  $(\alpha = 0.5, \omega = 0.02)$ , respectively. See (Deriche, 1987) for details of the Deriche detector.

What information would we need to  
'invert' the edge detection process?

# Elder – Are Edges Incomplete? 1999

Edge 'code':

- position,
- gradient magnitude,
- gradient direction,
- blur size.

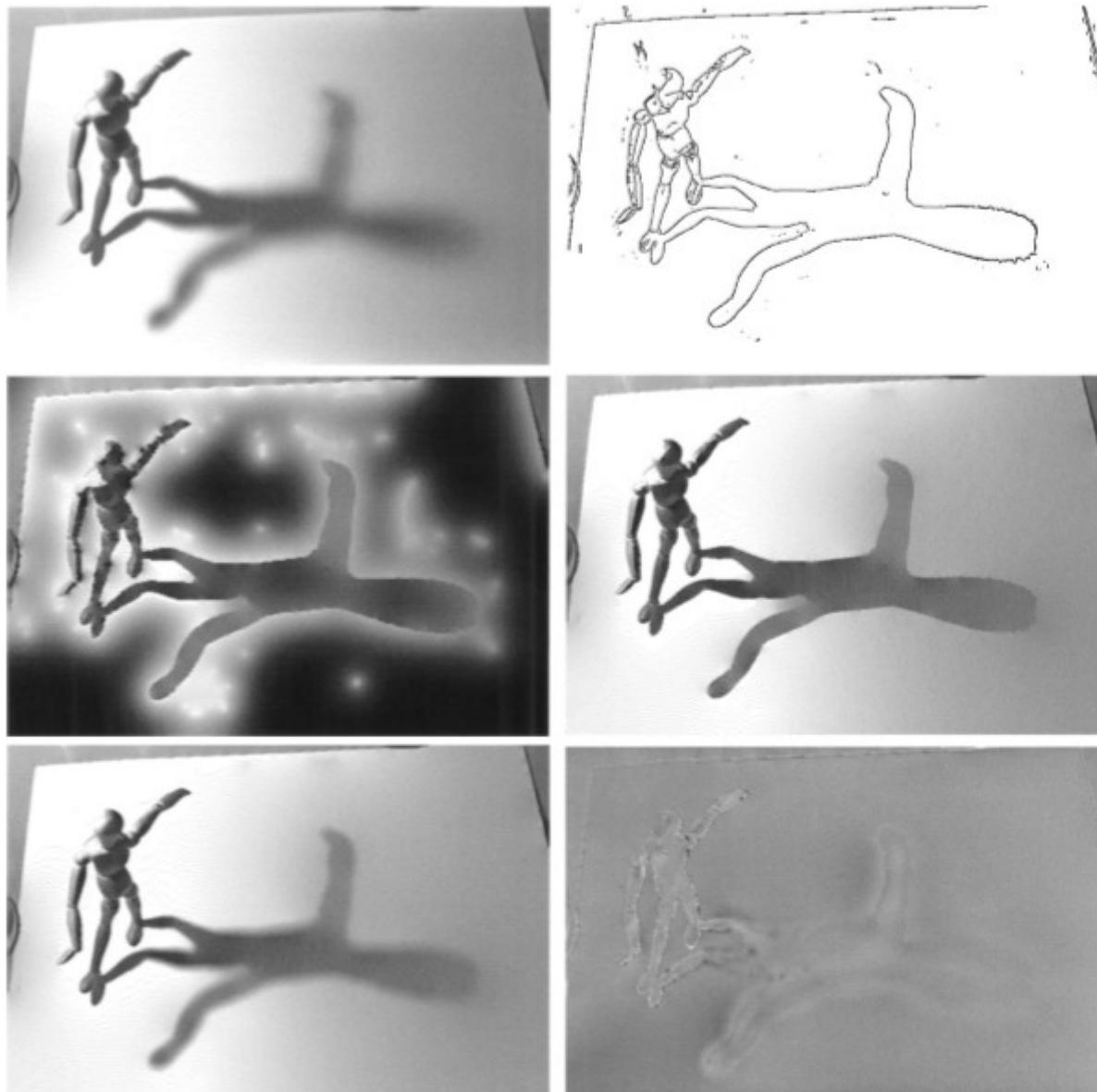


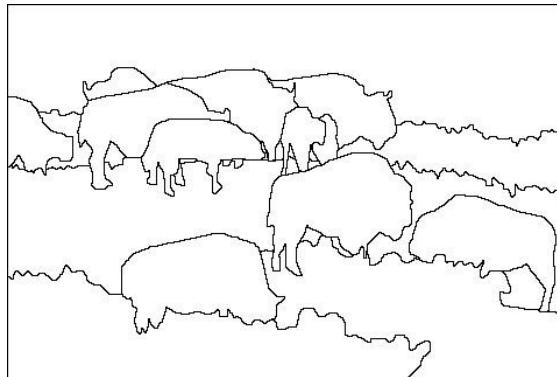
Figure 8. Top left: Original image. Top right: Detected edge locations. Middle left: Intermediate solution to the heat equation. Middle right: Reconstructed luminance function. Bottom left: Reblurred result. Bottom right: Error map (reblurred result—original). Bright indicates overestimation of intensity; dark indicates underestimation. Edge density is 1.7%. RMS error is 10.1 grey levels, with a 3.9 grey level DC component, and an estimated 1.6 grey levels due to noise removal.

# Where do humans see boundaries?

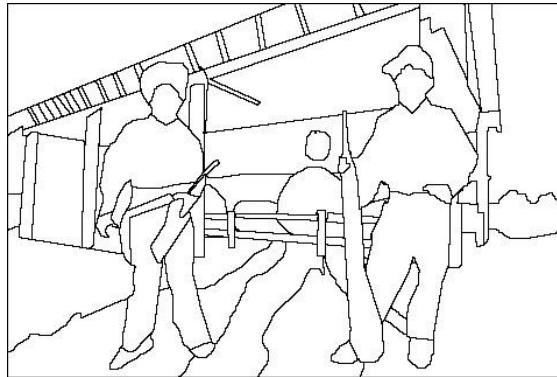
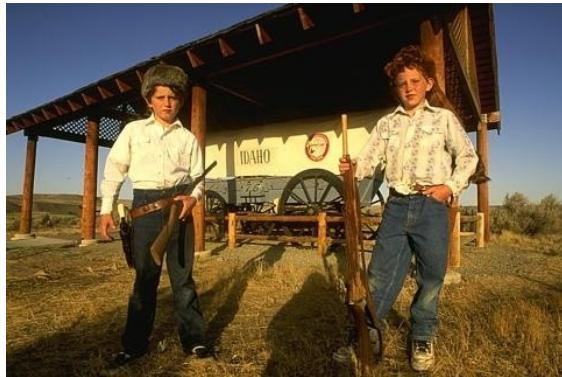
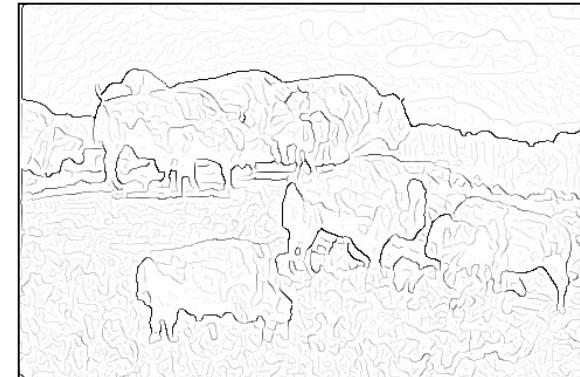
image



human segmentation



gradient magnitude



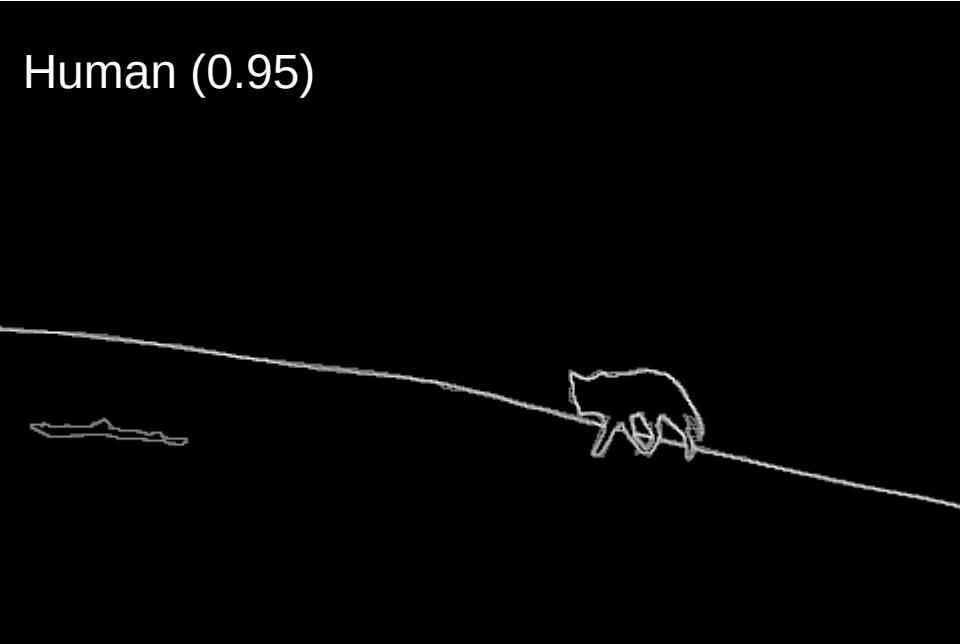
- Berkeley segmentation database:

[http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbe\\_nch/](http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbe_nch/)

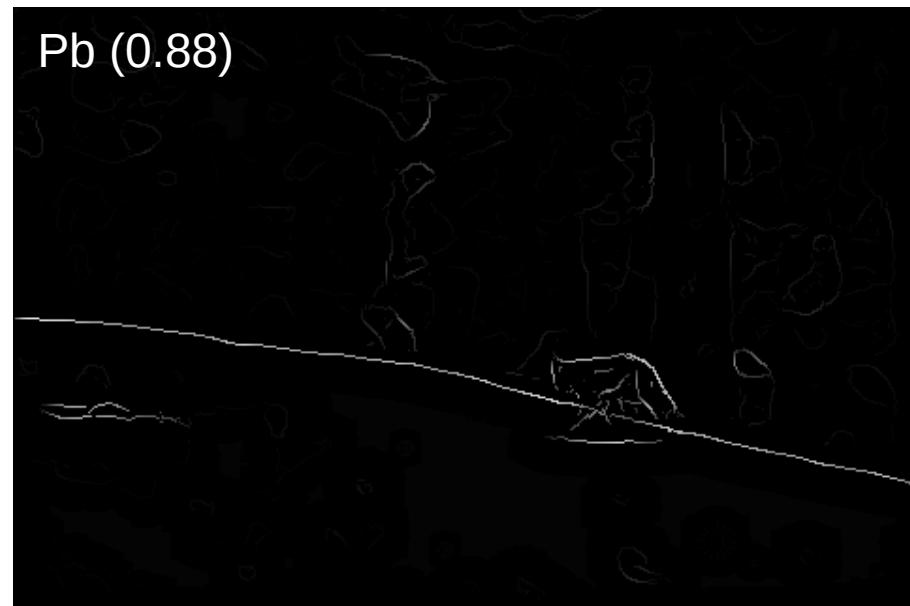
# Results



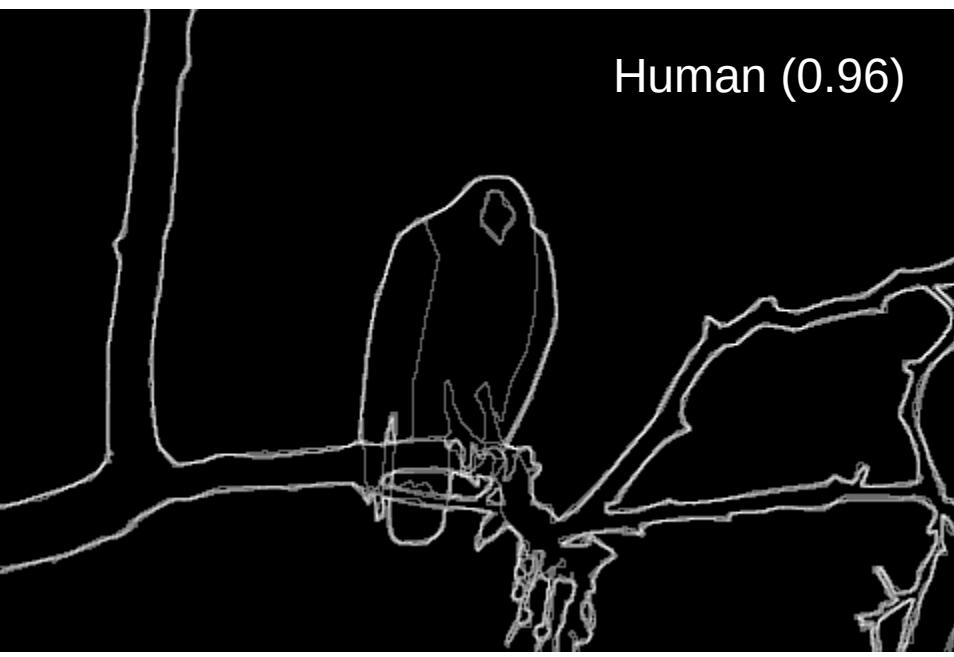
Human (0.95)



Score = confidence of edge.  
For humans, this is averaged across  
multiple participants.



# Results

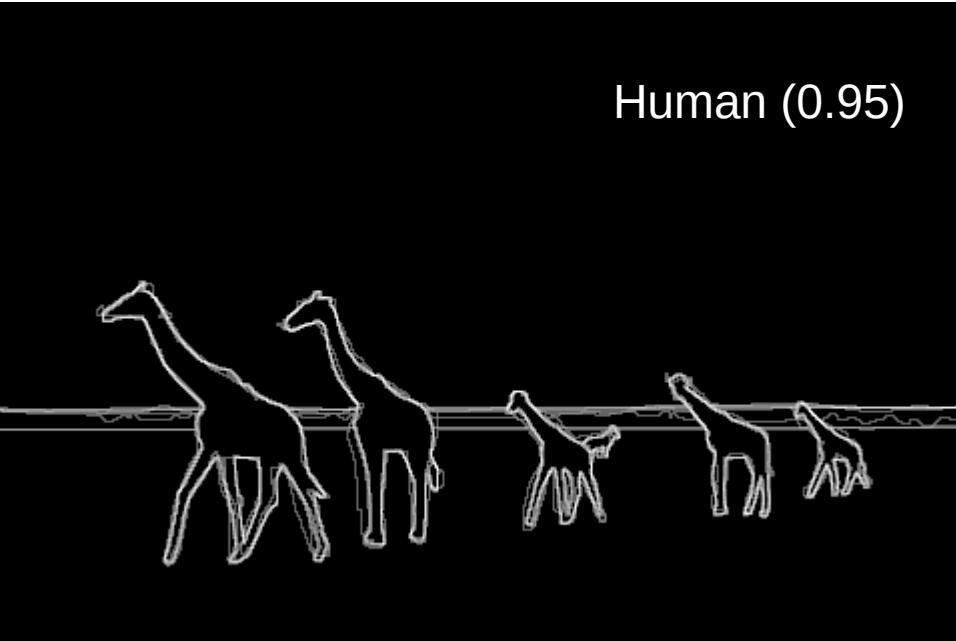


Score = confidence of edge.  
For humans, this is averaged across  
multiple participants.

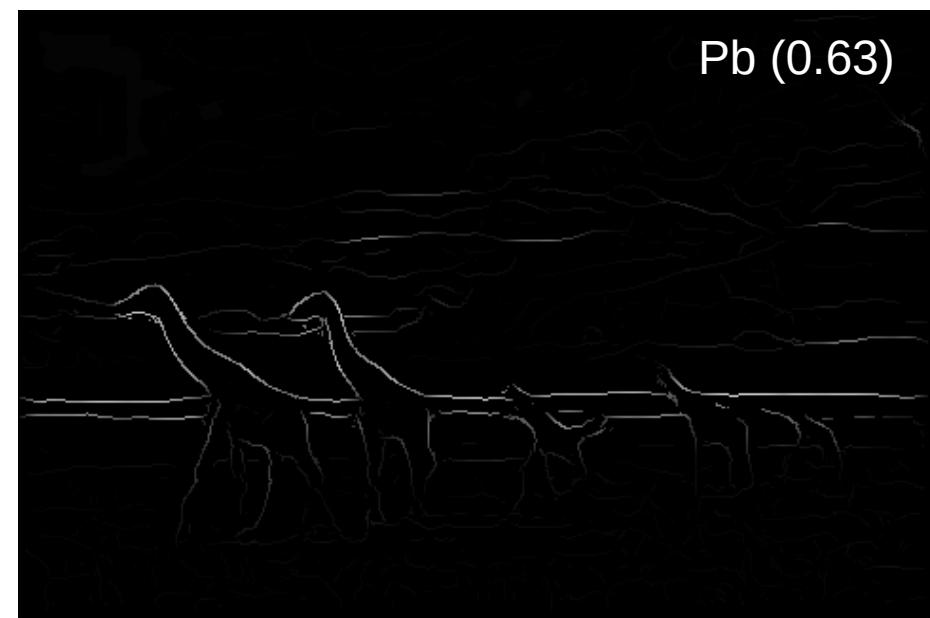




Human (0.95)

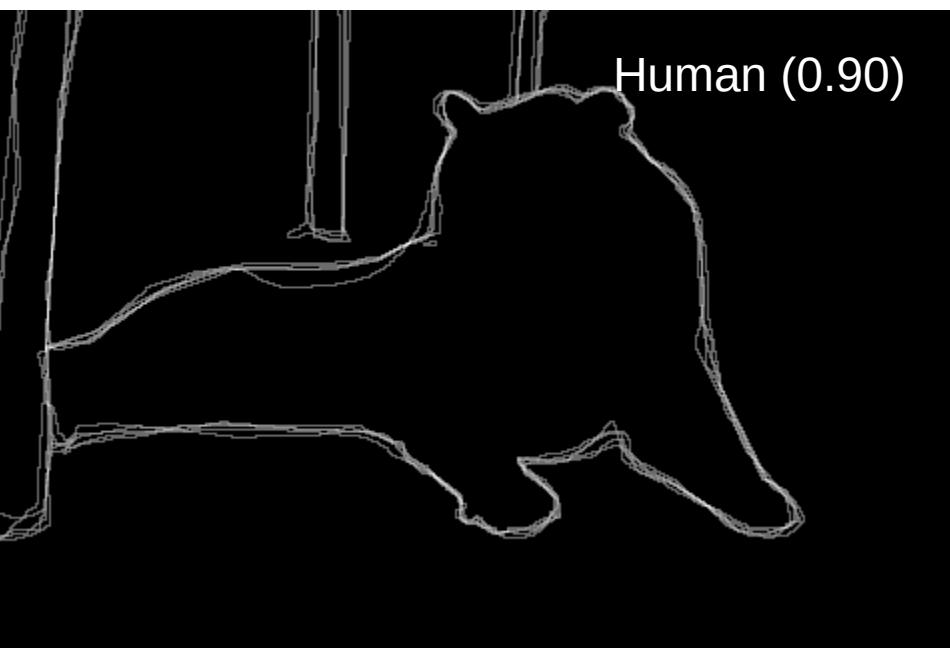


Score = confidence of edge.  
For humans, this is averaged across  
multiple participants.





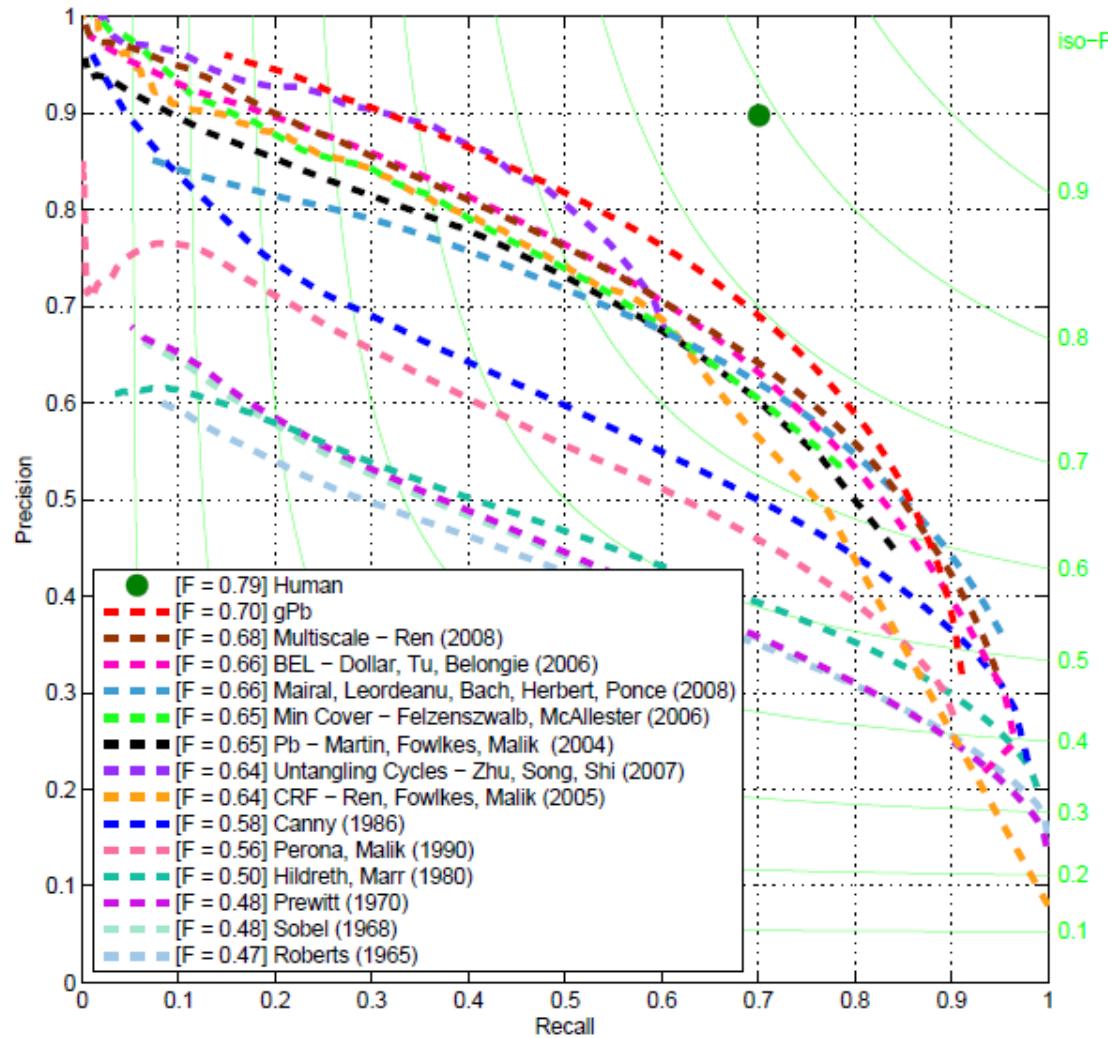
Score = confidence of edge.  
For humans, this is averaged across  
multiple participants.



For more:

<http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/bench/html/108082-color.html>

# 45+ years of boundary detection



# State of edge detection

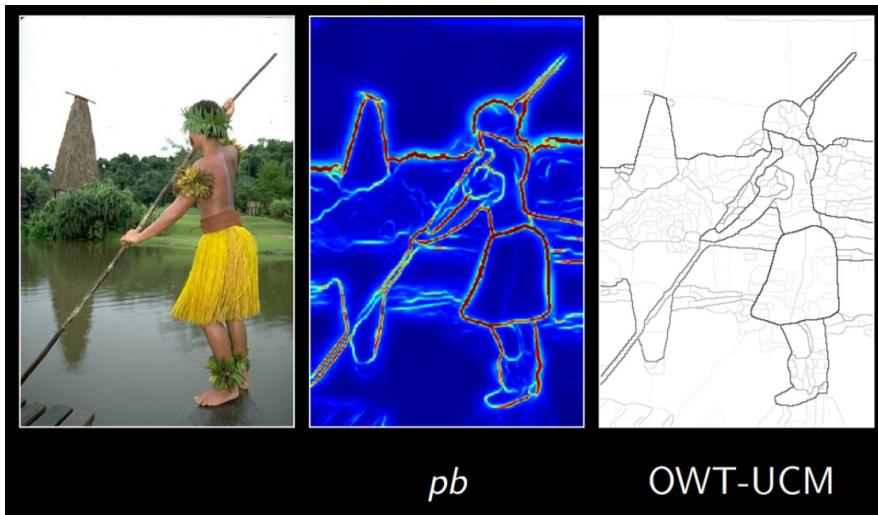
Local edge detection works well

- ‘False positives’ from illumination and texture edges (depends on our application).

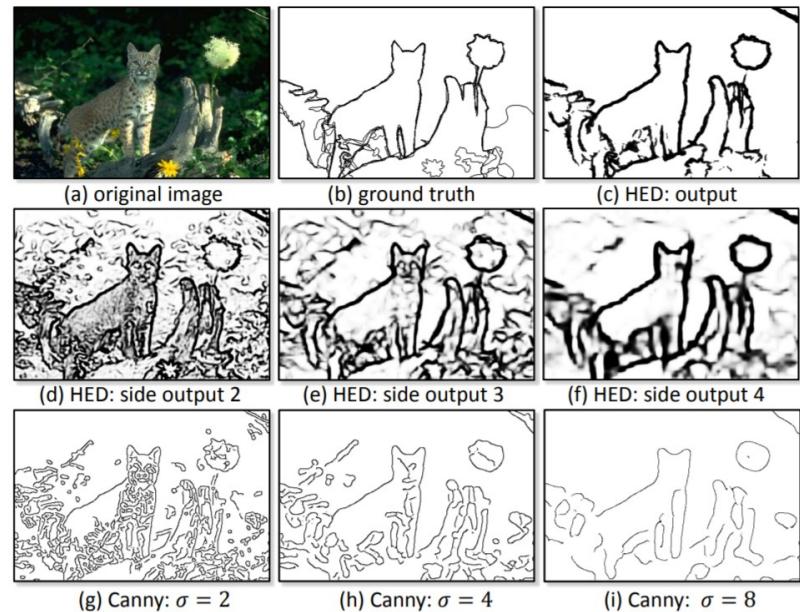
Some methods take into account longer contours

Modern methods actually “learn” from data however they have poor use of object and high-level information – processing low level pixels without reasoning!

# State of The Art-ish



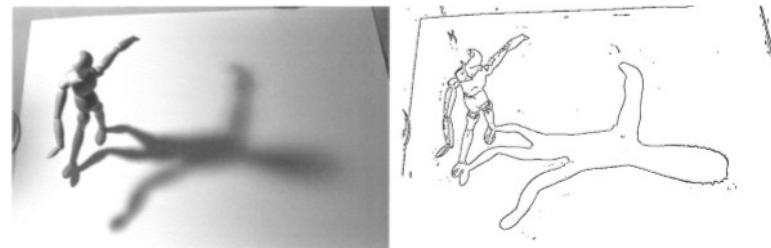
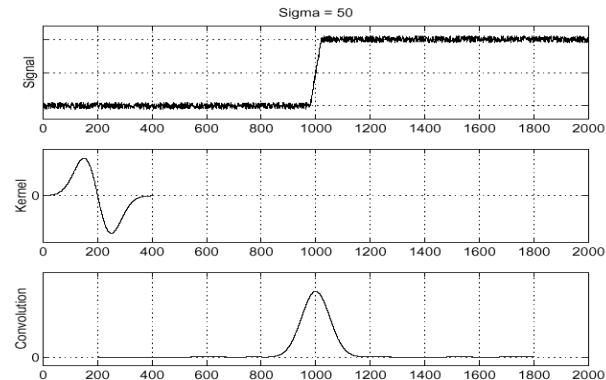
Boundary Extraction in Natural Images Using Ultrametric Contour Maps, Arbelaez 2006



Holistically-Nested Edge Detection, Xie 2015

# Summary: Edges

- Edge detection to identify visual change in image
- Derivative of Gaussian and linear combination of convolutions
- What is an edge?  
What is a good edge?



# Canny edge detector

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

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

22,000 citations!

L. Fei-Fei

# Demonstrator Image

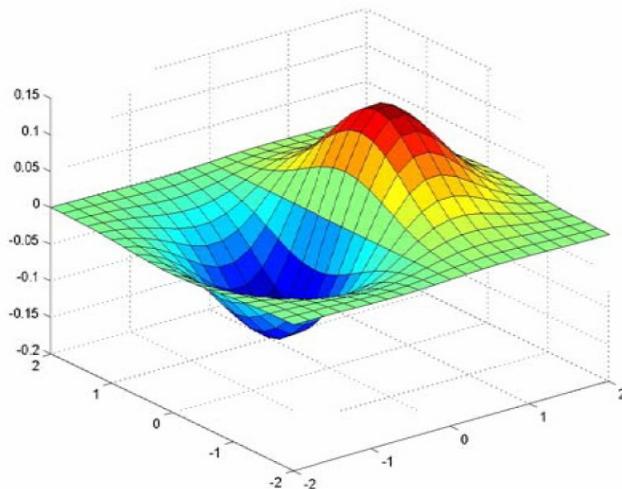
```
rgb2gray('img.png')
```



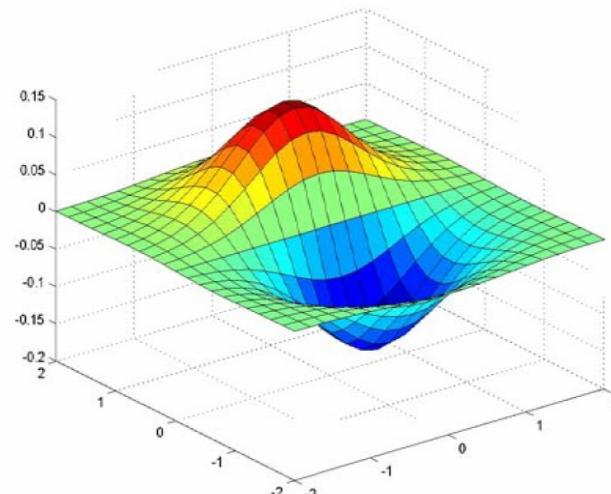
# Canny edge detector

1. Filter image with x, y derivatives of Gaussian

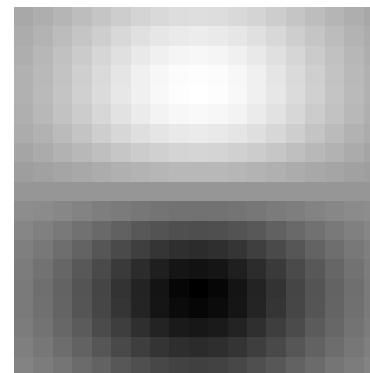
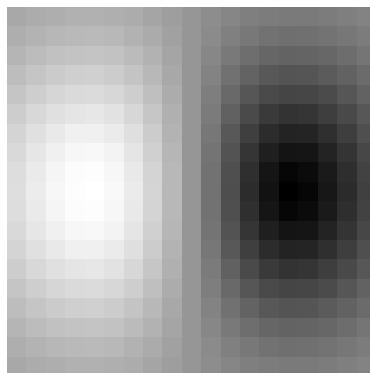
# Derivative of Gaussian filter



x-direction



y-direction



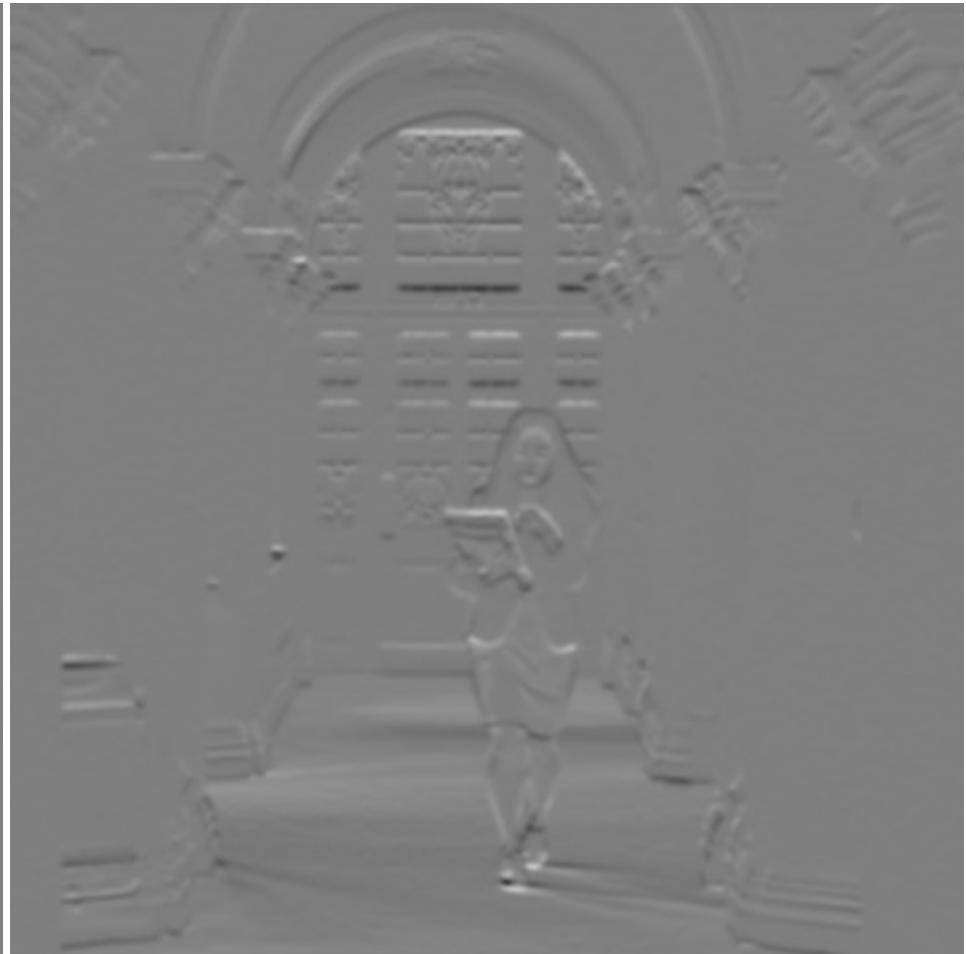
# Compute Gradients



X Derivative of Gaussian



Y Derivative of Gaussian



$(x^2 + 0.5 \text{ for visualization})$

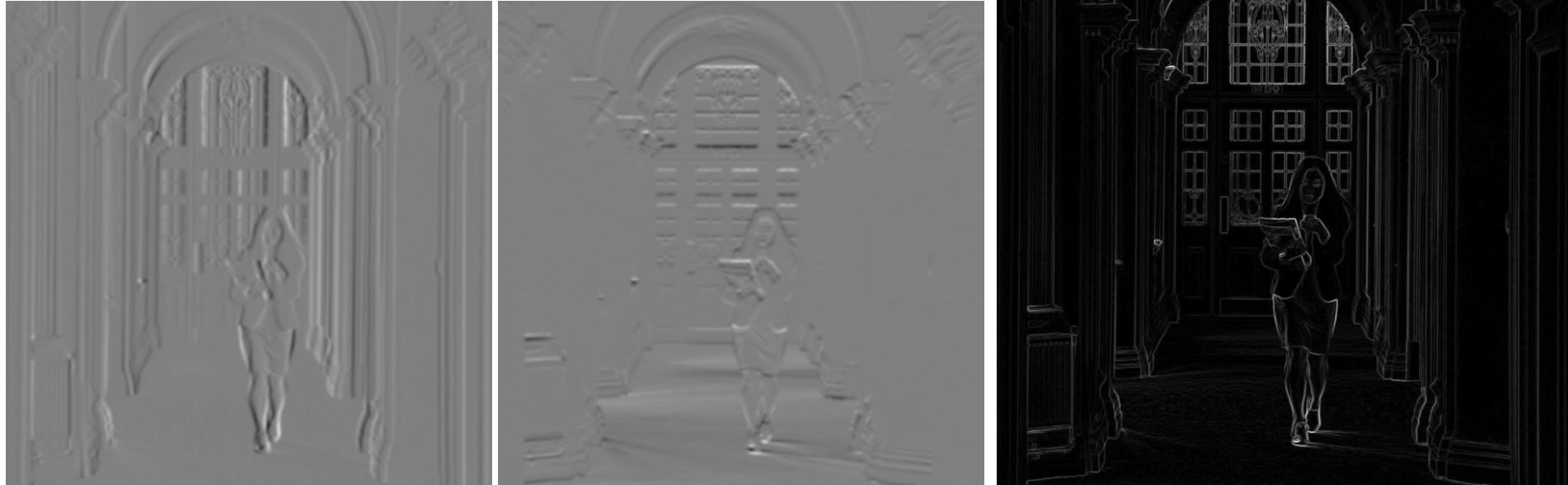
# Canny edge detector

1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient

# Compute Gradient Magnitude



$\text{sqrt}(\text{XDerivOfGaussian} .^2 + \text{YDerivOfGaussian} .^2)$  = gradient magnitude



(x4 for visualization)

# Compute Gradient Orientation

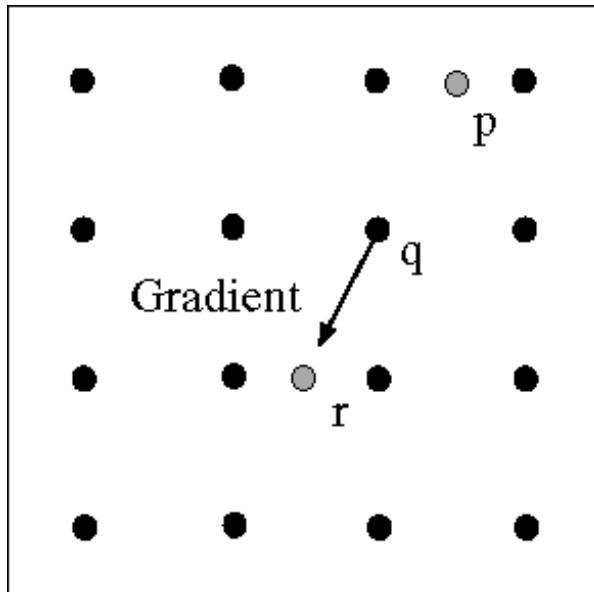
- Threshold magnitude at minimum level
- Get orientation via  $\theta = \text{atan2}(y\text{Deriv}, x\text{Deriv})$



# Canny edge detector

1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
  - Thin multi-pixel wide “ridges” to single pixel width

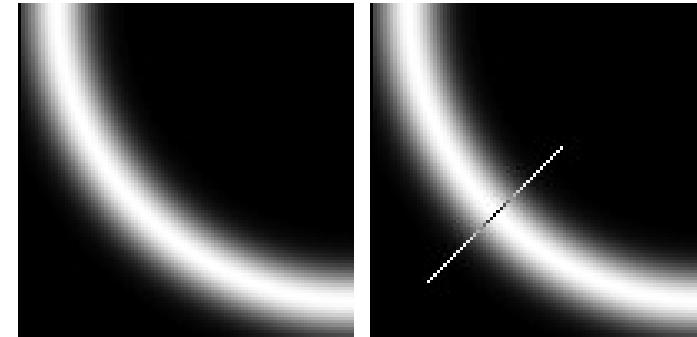
# Non-maximum suppression for each orientation



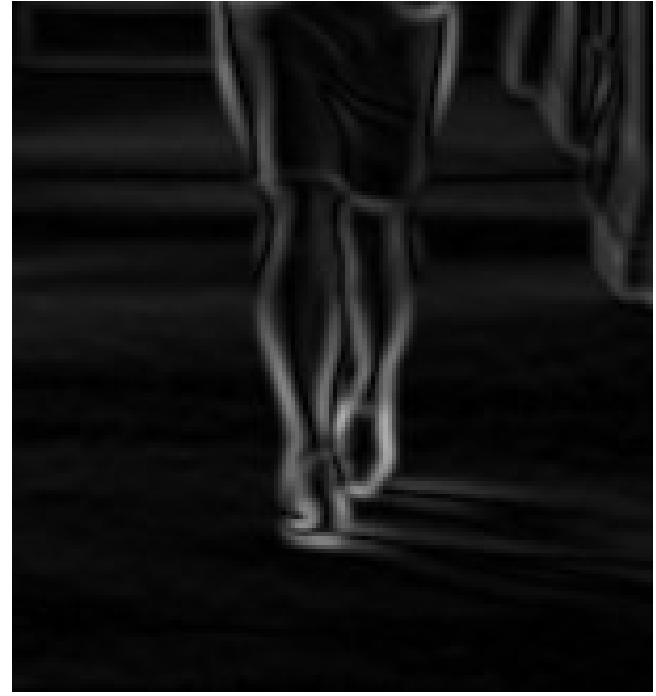
At pixel q:

We have a maximum if the value is larger than those at both p and at r.

Interpolate along gradient direction to get these values.

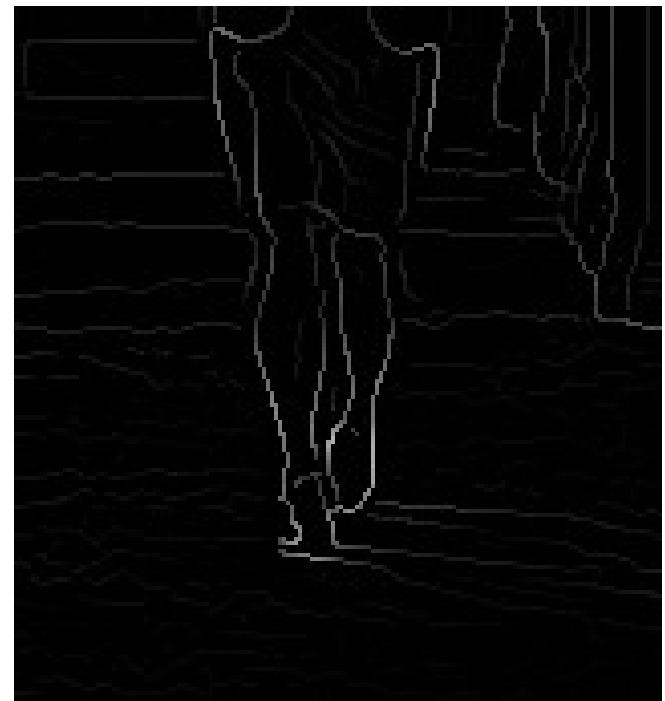


# Before Non-max Suppression



Gradient magnitude (x4 for visualization)

# After non-max suppression



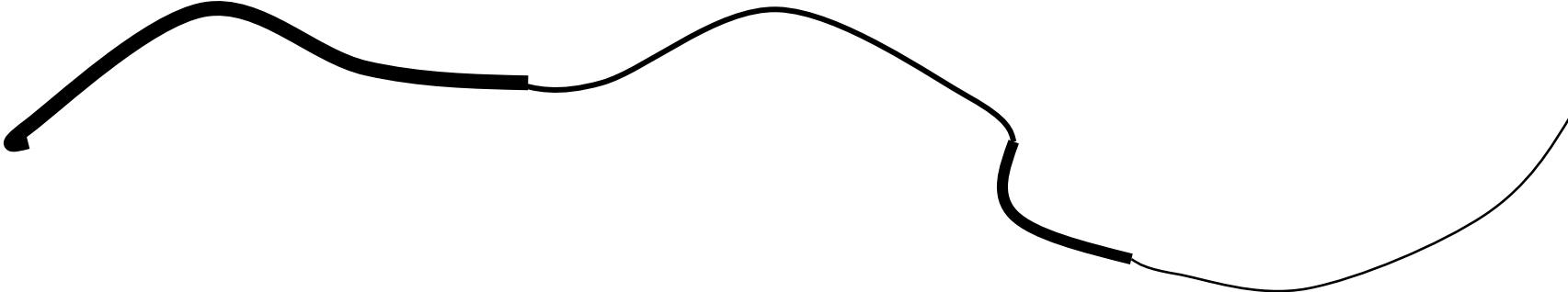
Gradient magnitude (x4 for visualization)

# Canny edge detector

1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
  - Thin multi-pixel wide “ridges” to single pixel width
4. ‘Hysteresis’ Thresholding

# 'Hysteresis' thresholding

- Two thresholds – high and low
  - Grad. mag. > high threshold? = strong edge
  - Grad. mag. < low threshold? noise
  - In between = weak edge
- 
- 'Follow' edges starting from strong edge pixels
  - Continue them into weak edges
    - Connected components (Szeliski 3.3.4)

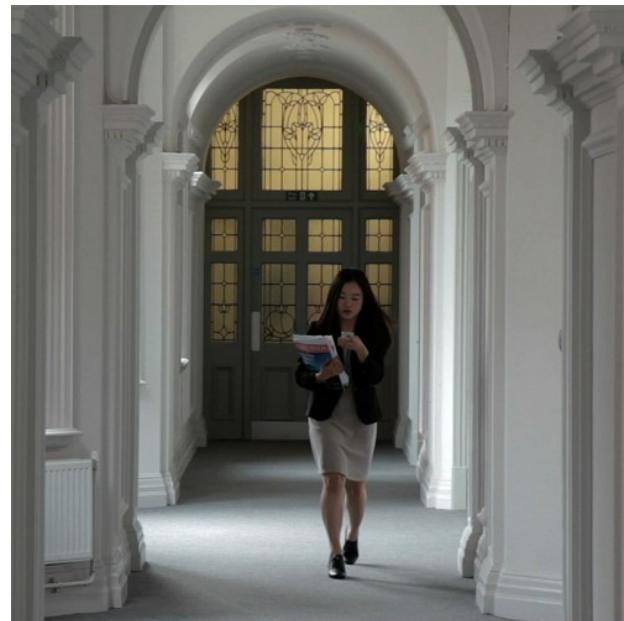


# Final Canny Edges

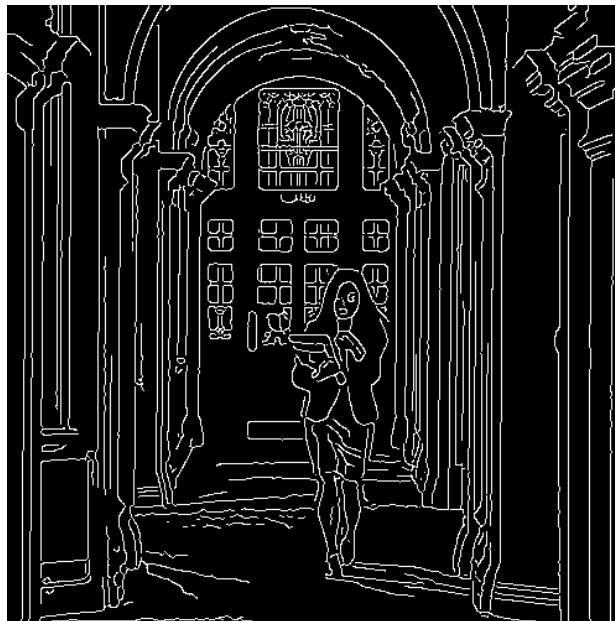
$$\sigma = \sqrt{2}, t_{low} = 0.05, t_{high} = 0.1$$



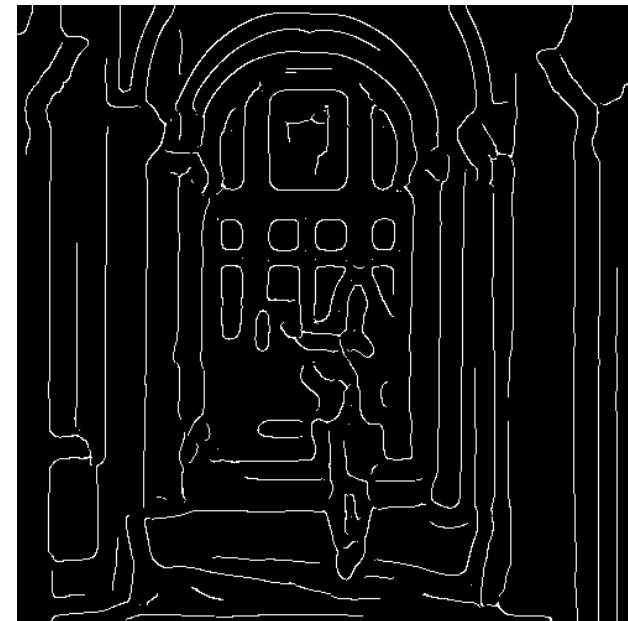
# Effect of (Gaussian kernel spread/size)



Original



$$\sigma = \sqrt{2}$$



$$\sigma = 4\sqrt{2}$$

The choice of  $\sigma$  depends on desired behavior

- Large sigma detects large scale edges
- Small sigma detects fine features

# Canny edge detector

1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
  - Thin multi-pixel wide “ridges” to single pixel width
4. ‘Hysteresis’ Thresholding:
  - Define two thresholds: low and high
  - Use the high threshold to start edge curves and the low threshold to continue them
  - ‘Follow’ edges starting from strong edge pixels
    - Connected components
- Python: e.g., `skimage.feature.canny()`