# Edge Detection

Magritte,
“Decalcomania”

Computer Vision (CS 543 / ECE 549)

University of Illinois

Derek Hoiem

Many slides from Lana Lazebnik, Steve Seitz, David Forsyth, David Lowe, Fei-Fei Li

# Last class

- How to use filters for
  - Matching
  - Compression
- Image representation with pyramids
- Texture and filter banks

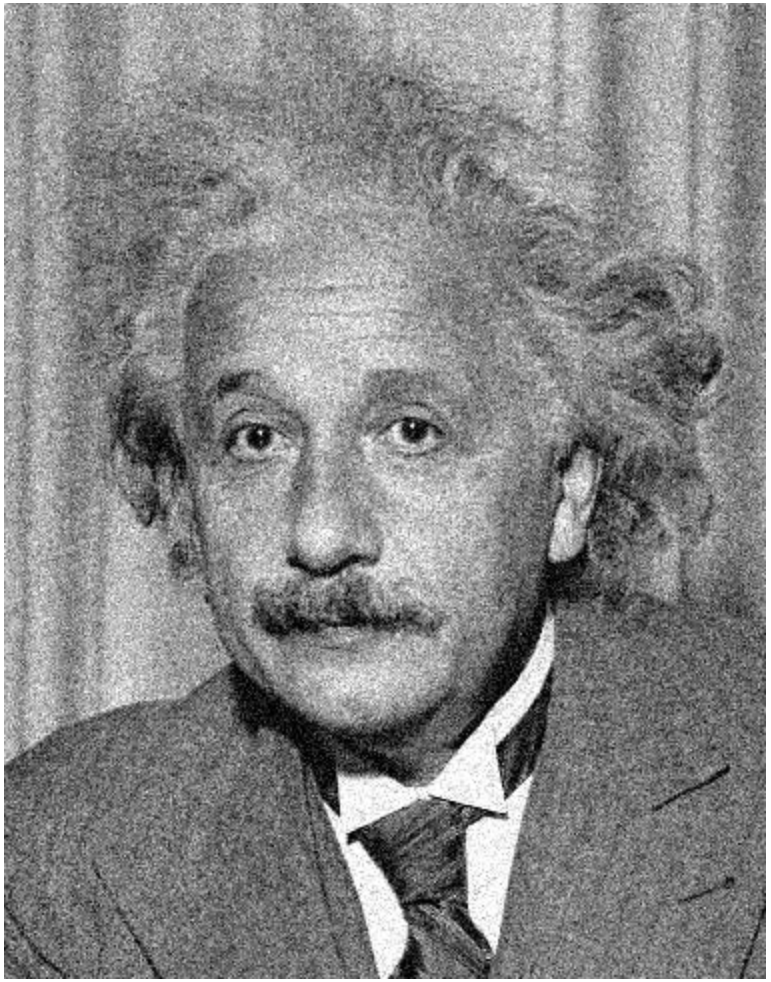
# Issue from Tuesday

- Why not use an ideal filter?

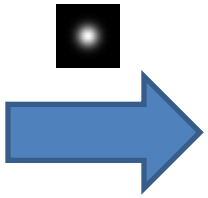
Answer: has infinite spatial extent, clipping results in ringing

Attempt to apply ideal filter in frequency domain

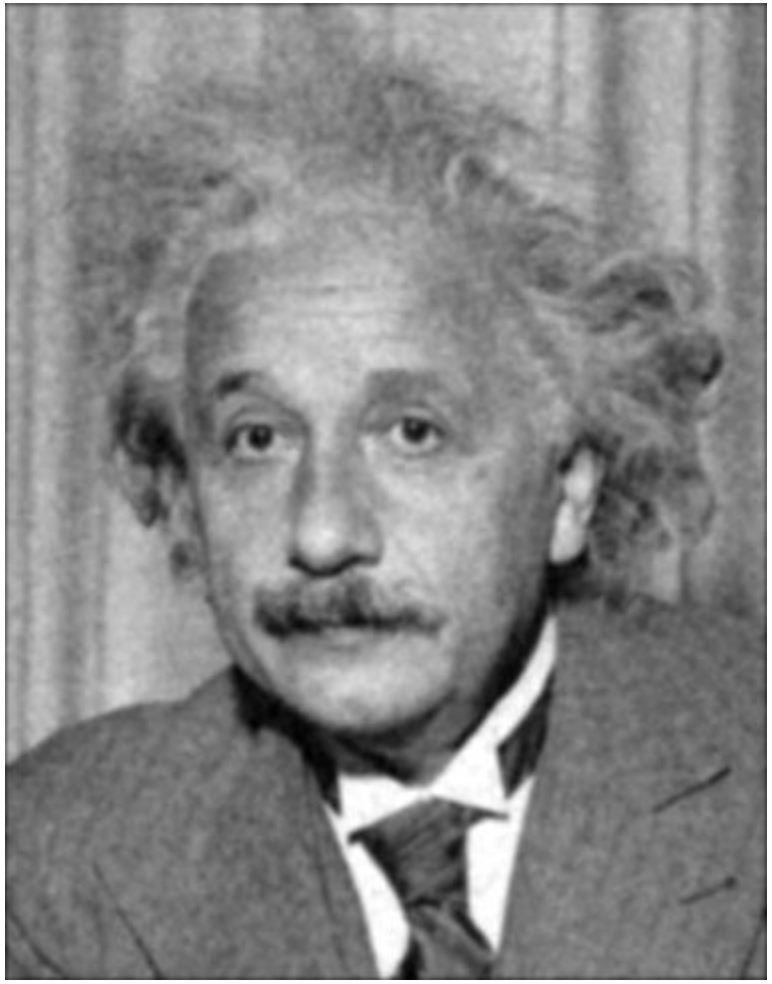
# Denoising



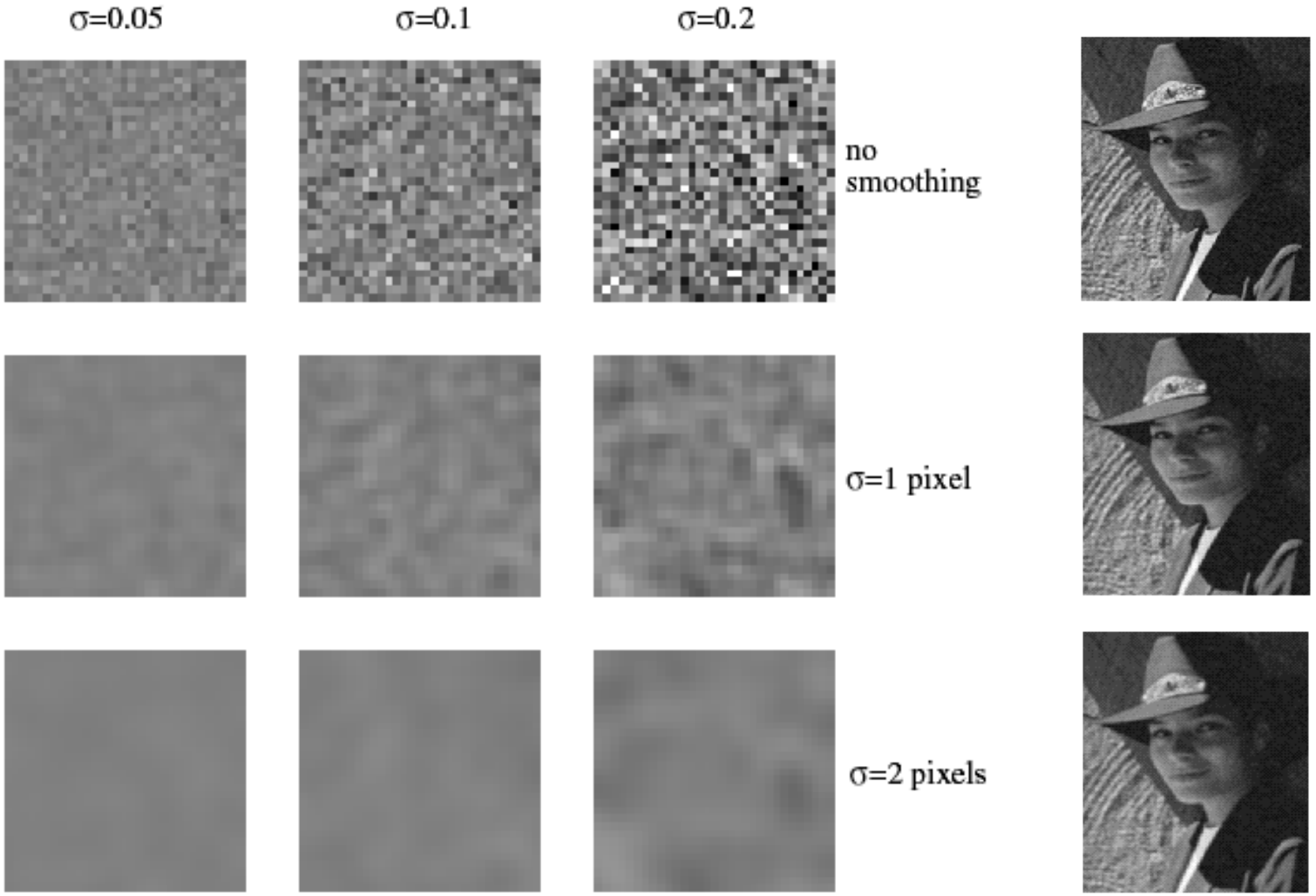
Additive Gaussian Noise



Gaussian
Filter



# Reducing Gaussian noise




Smoothing with larger standard deviations suppresses noise, but also blurs the image

Source: S. Lazebnik

# Reducing salt-and-pepper noise by Gaussian smoothing

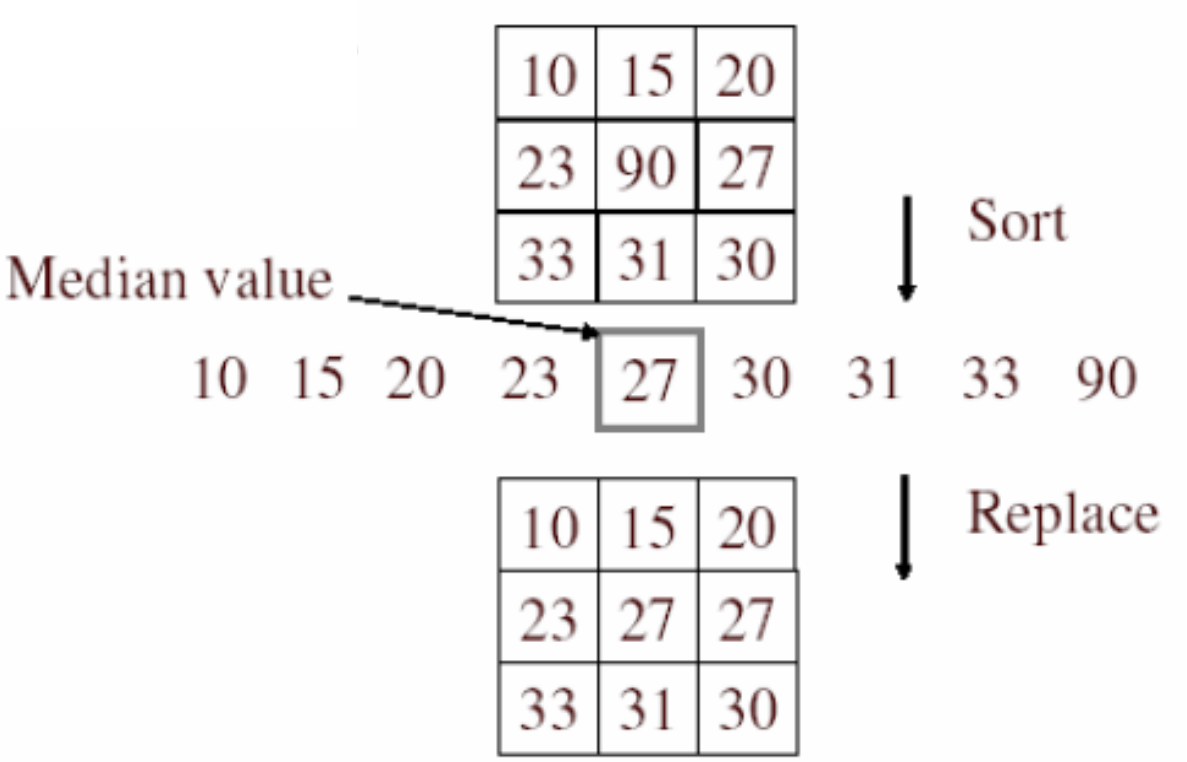
3x3

5x5

7x7

# Alternative idea: Median filtering

- A **median filter** operates over a window by selecting the median intensity in the window

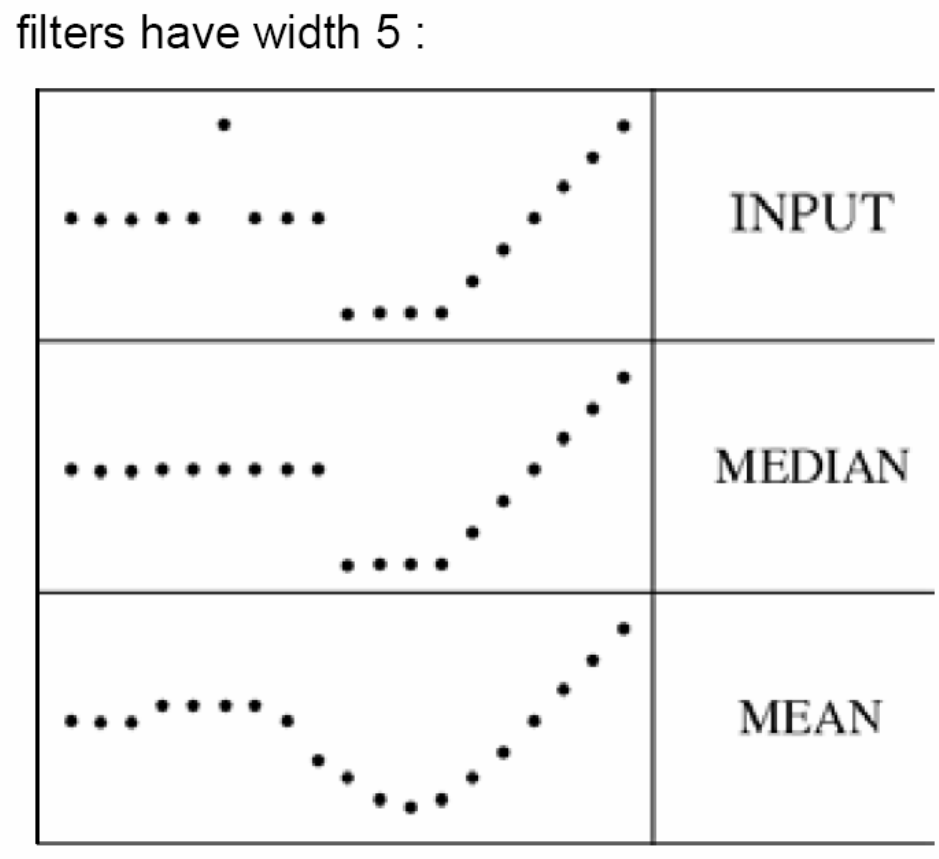



- Is median filtering linear?

Source: K. Grauman

# Median filter

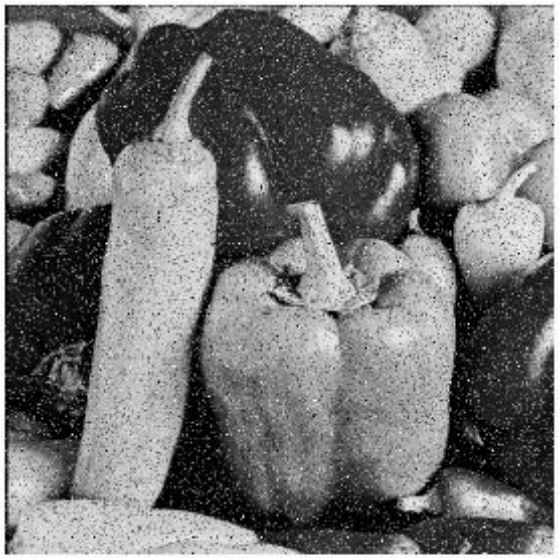
- What advantage does median filtering have over Gaussian filtering?
  - Robustness to outliers, preserves edges

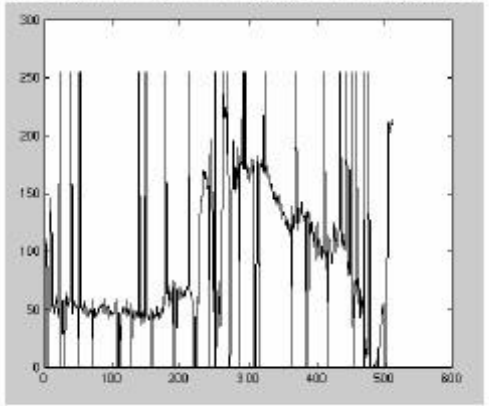



Source: K. Grauman

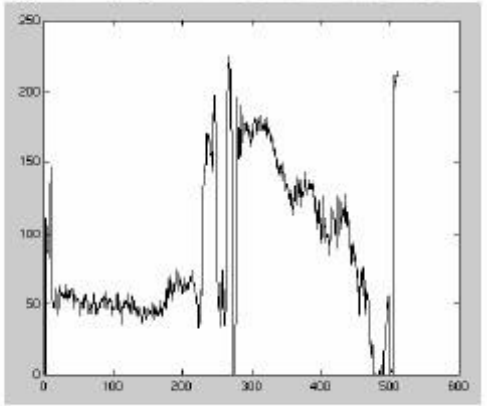
# Median filter

Salt-and-pepper noise





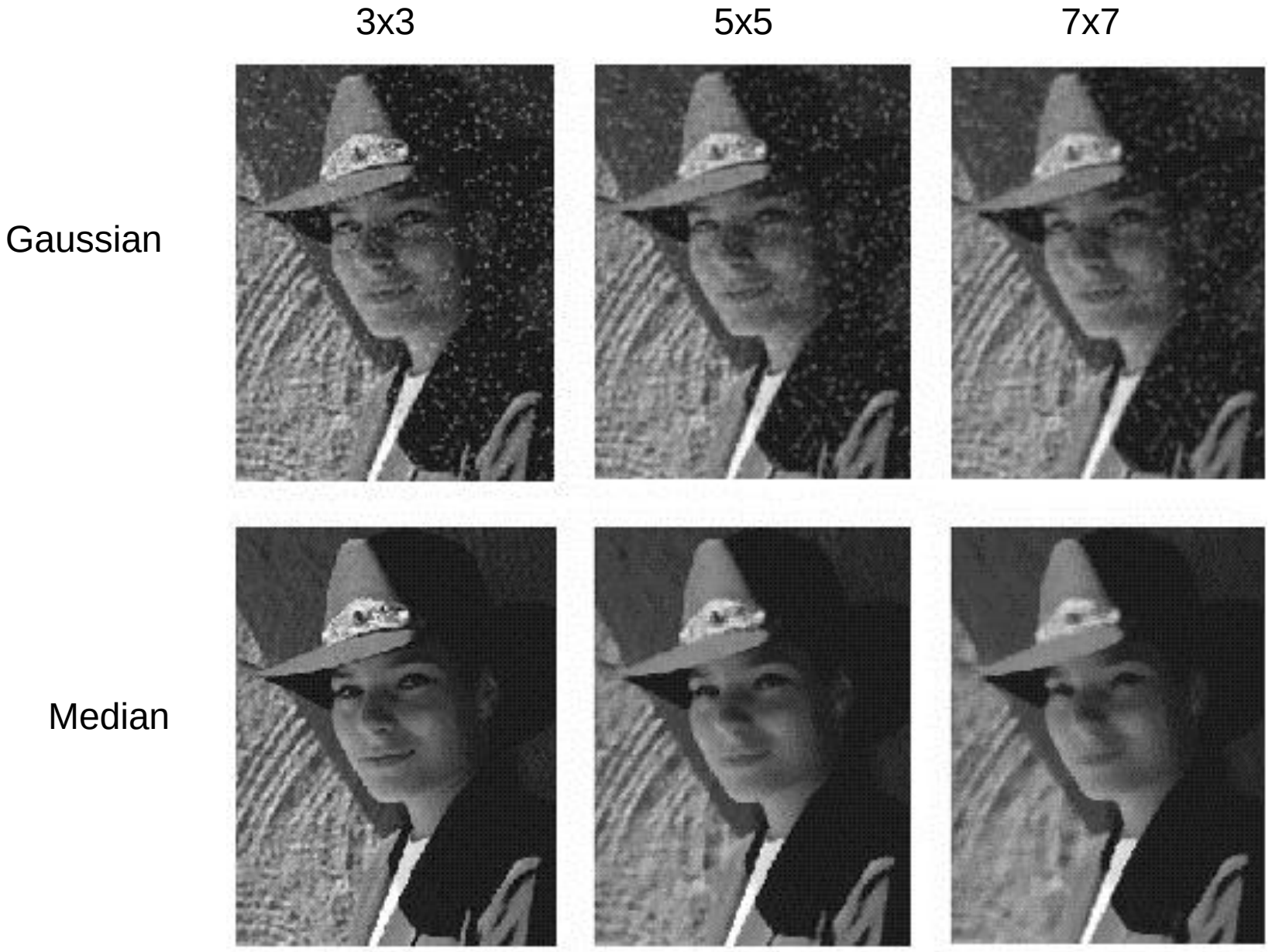

Median filtered




- MATLAB: medfilt2(image, [h w])

Source: M. Hebert

# Median vs. Gaussian filtering



# Other non-linear filters

- Weighted median (pixels further from center count less)
- Clipped mean (average, ignoring few brightest and darkest pixels)
- Max or min filter (`ordfilt2`)
- Bilateral filtering (weight by spatial distance *and* intensity difference)

Bilateral filtering

Image: http://vision.ai.uiuc.edu/?p=1455

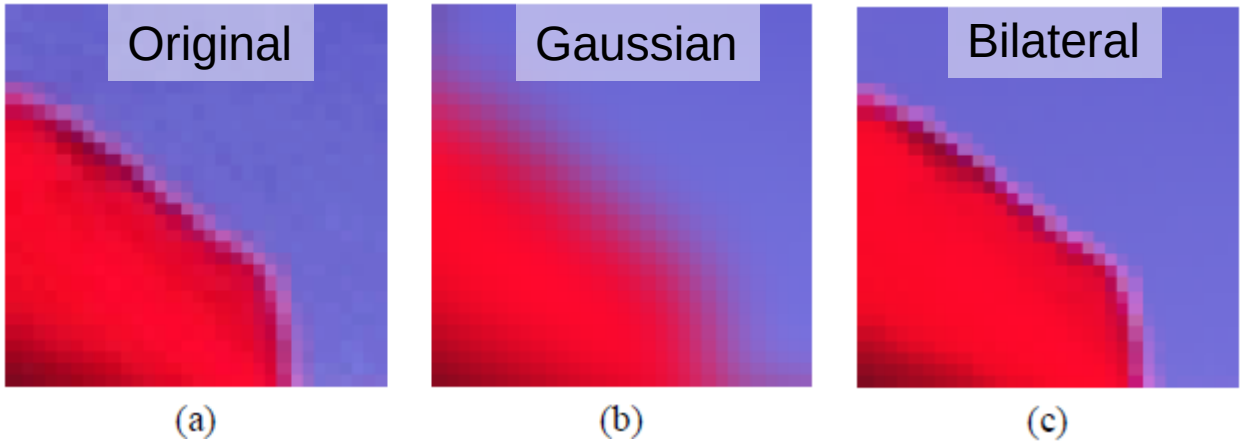
# Bilateral filters

- Edge preserving: weights similar pixels more

spatial similarity (e.g., intensity)

$$I_{\mathbf{p}}^{\mathrm{b}} = \frac{1}{W_{\mathbf{p}}^{\mathrm{b}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$

$$\text{with} \quad W_{\mathbf{p}}^{\mathrm{b}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)$$



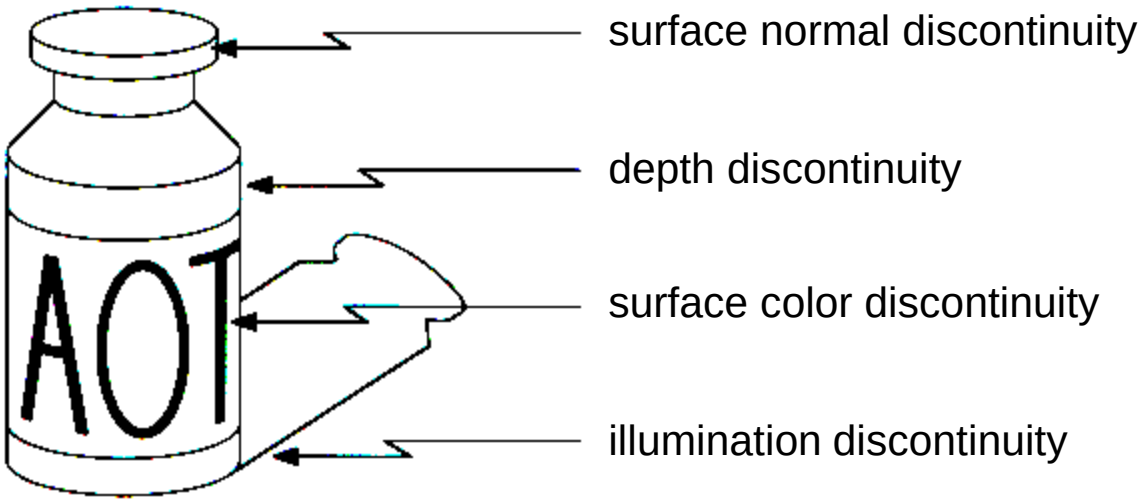

Carlo Tomasi, Roberto Manduchi, Bilateral Filtering for Gray and Color Images, ICCV, 1998.

# Today's class

- Detecting edges

- Finding straight lines

# Origin of Edges

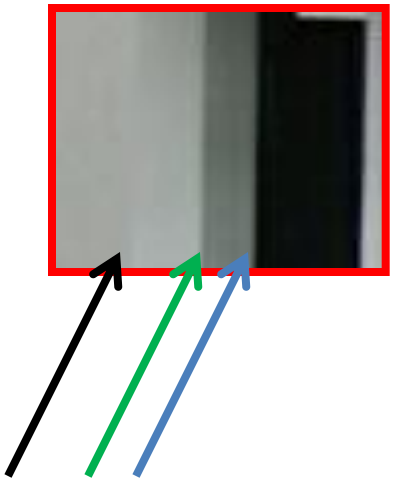



- Edges are caused by a variety of factors

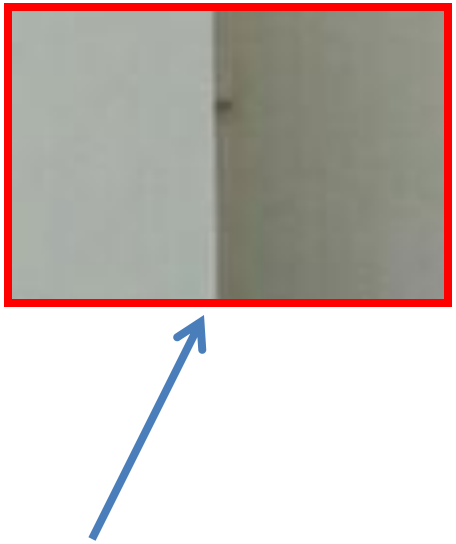
Source: Steve Seitz

# Closeup of edges

# Closeup of edges



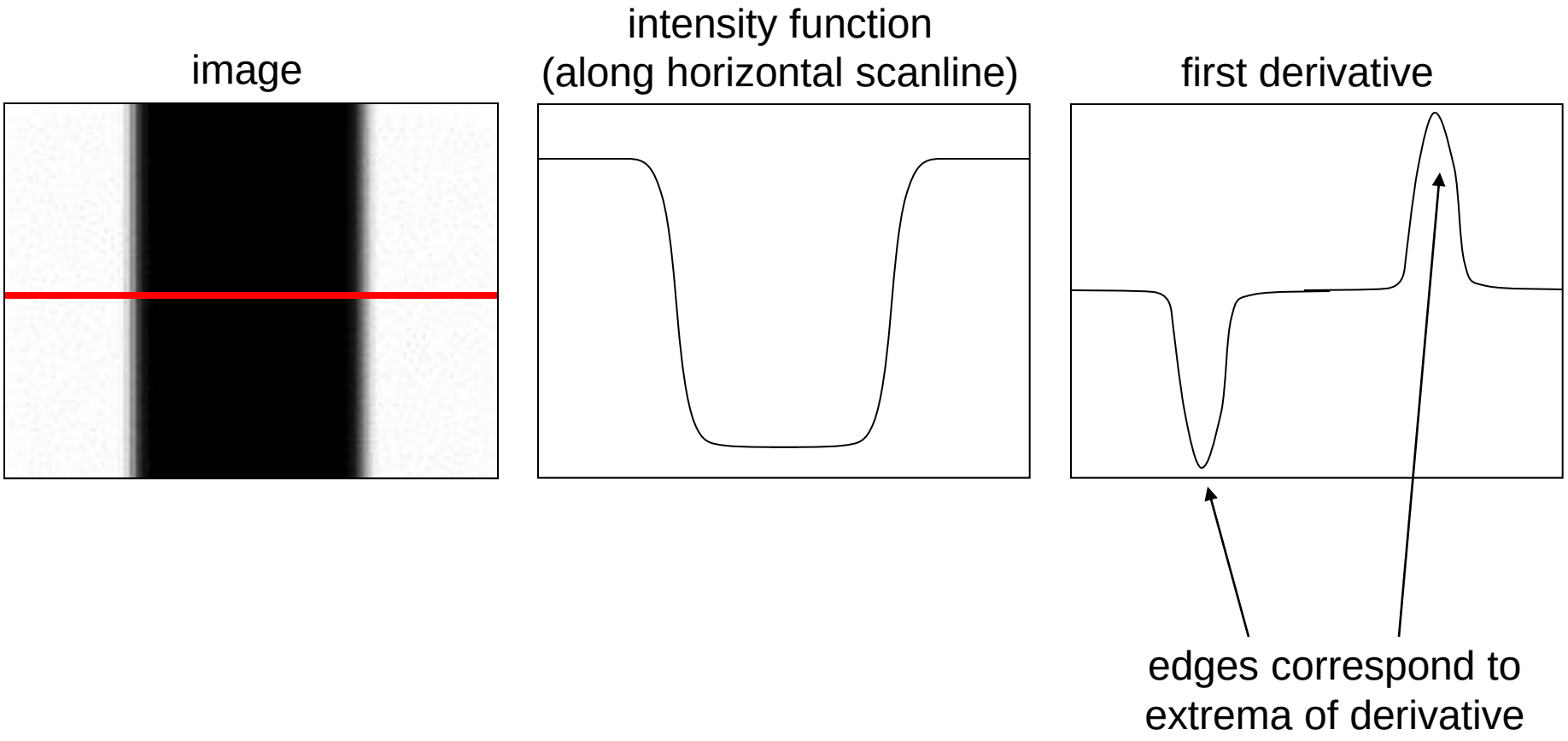
# Closeup of edges



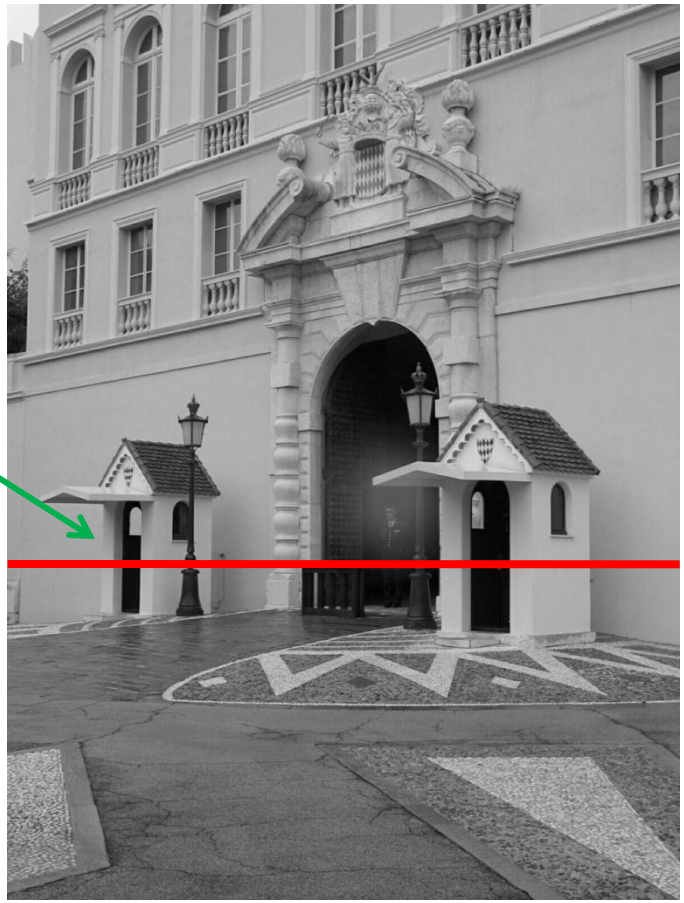
# Closeup of edges

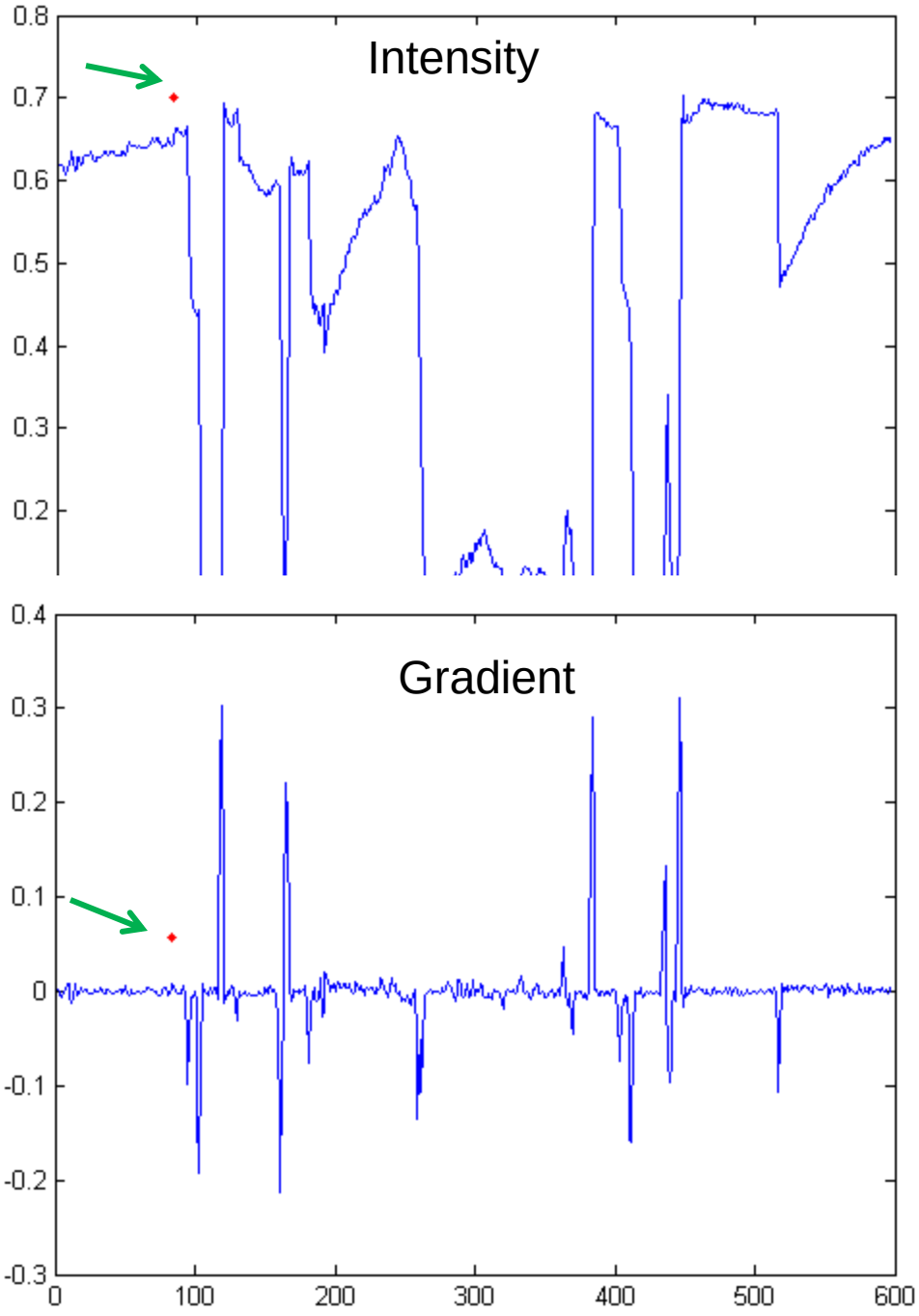
# Characterizing edges

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

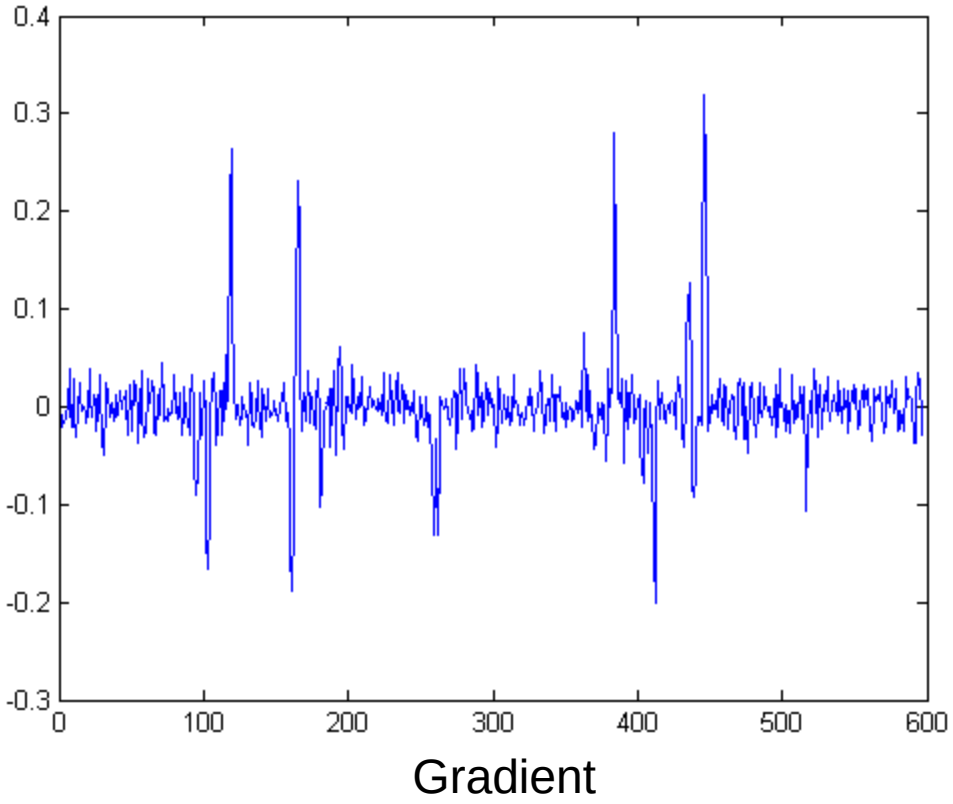


# Intensity profile



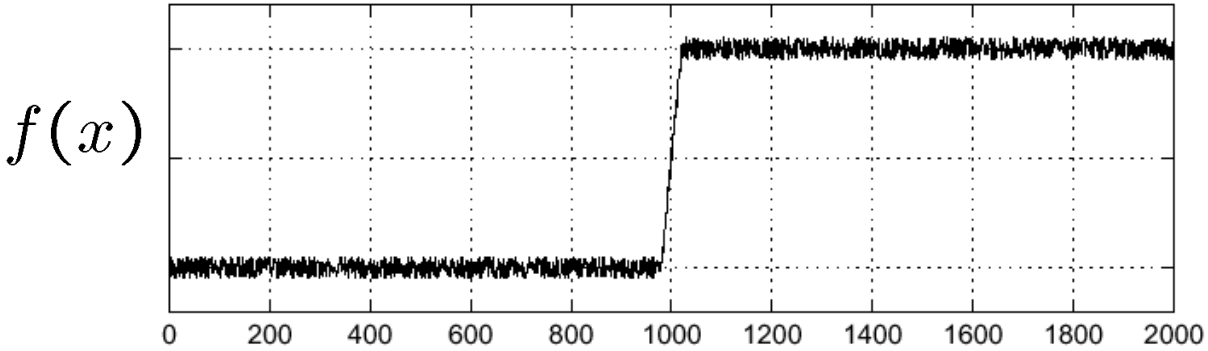


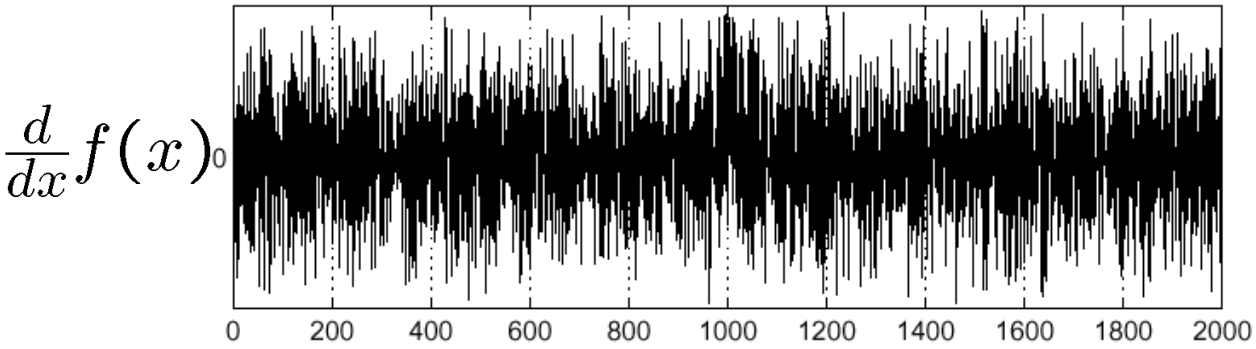
# With a little Gaussian noise



# 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?

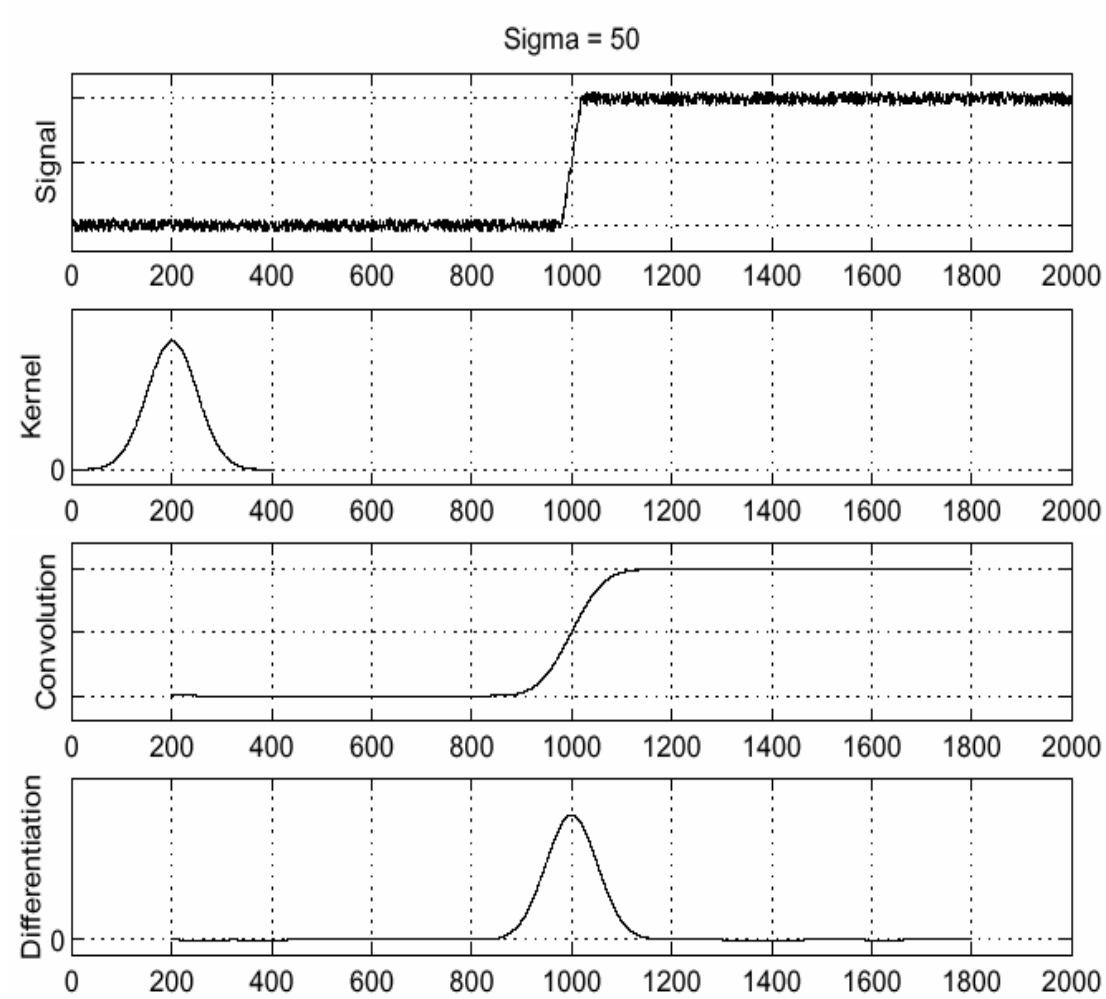
Source: S. Seitz

# 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?

Source: D. Forsyth

# Solution: smooth first




$f$

$g$

$f * g$

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

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

Source: S. Seitz

# Derivative theorem of convolution

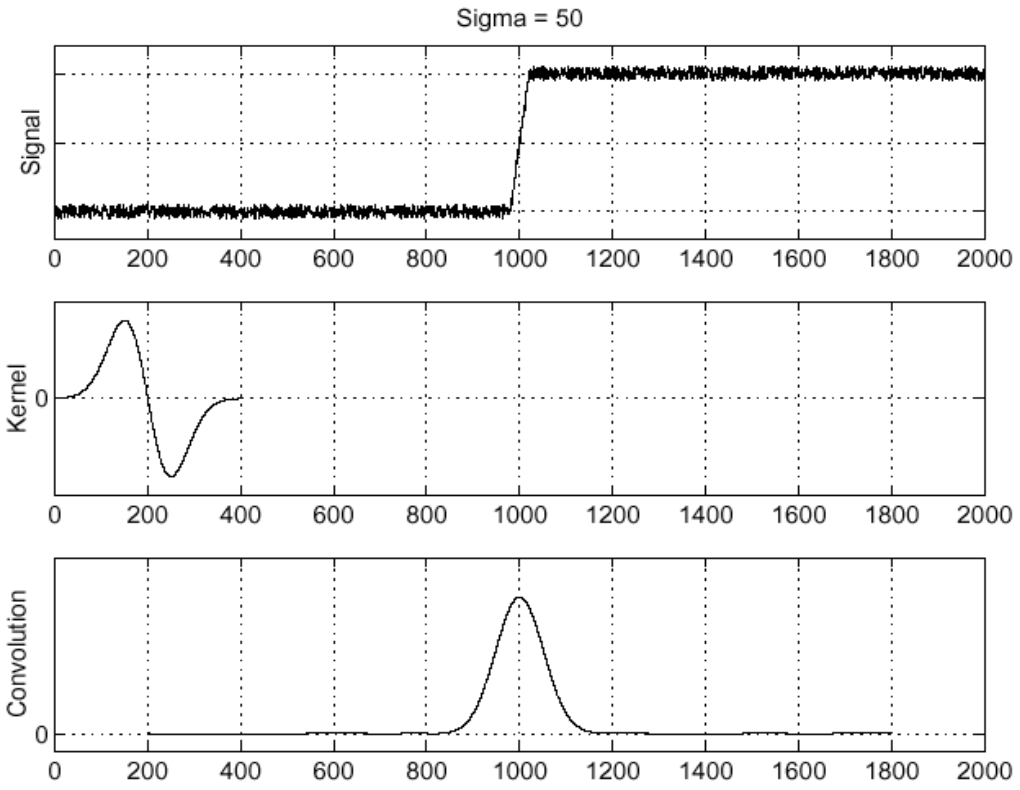
- Differentiation is convolution, and convolution is associative: $$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$$

- This saves us one operation:

$f$

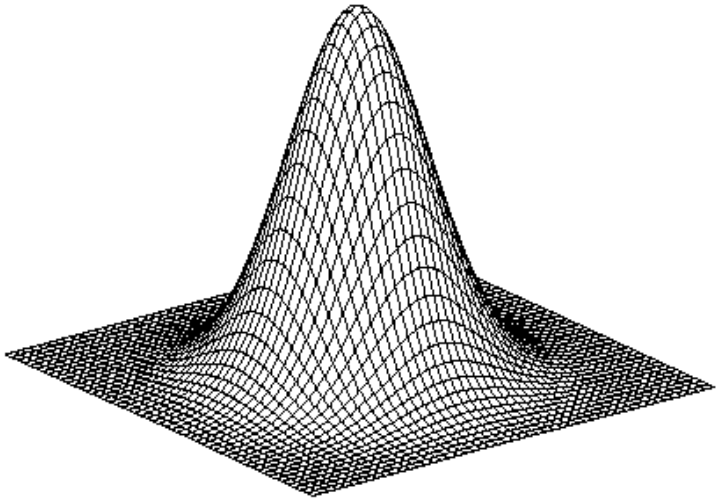
$\frac{d}{dx} g$

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

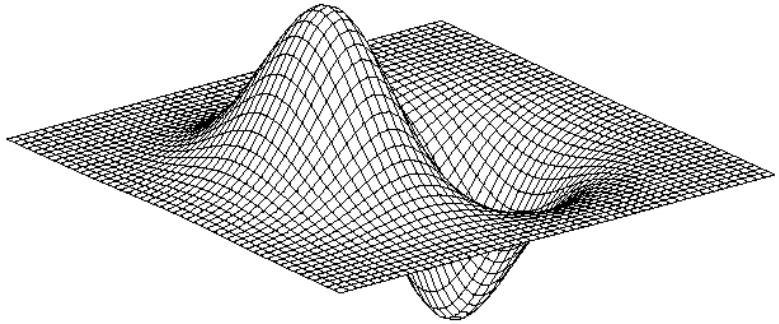



Source: S. Seitz

# Derivative of Gaussian filter

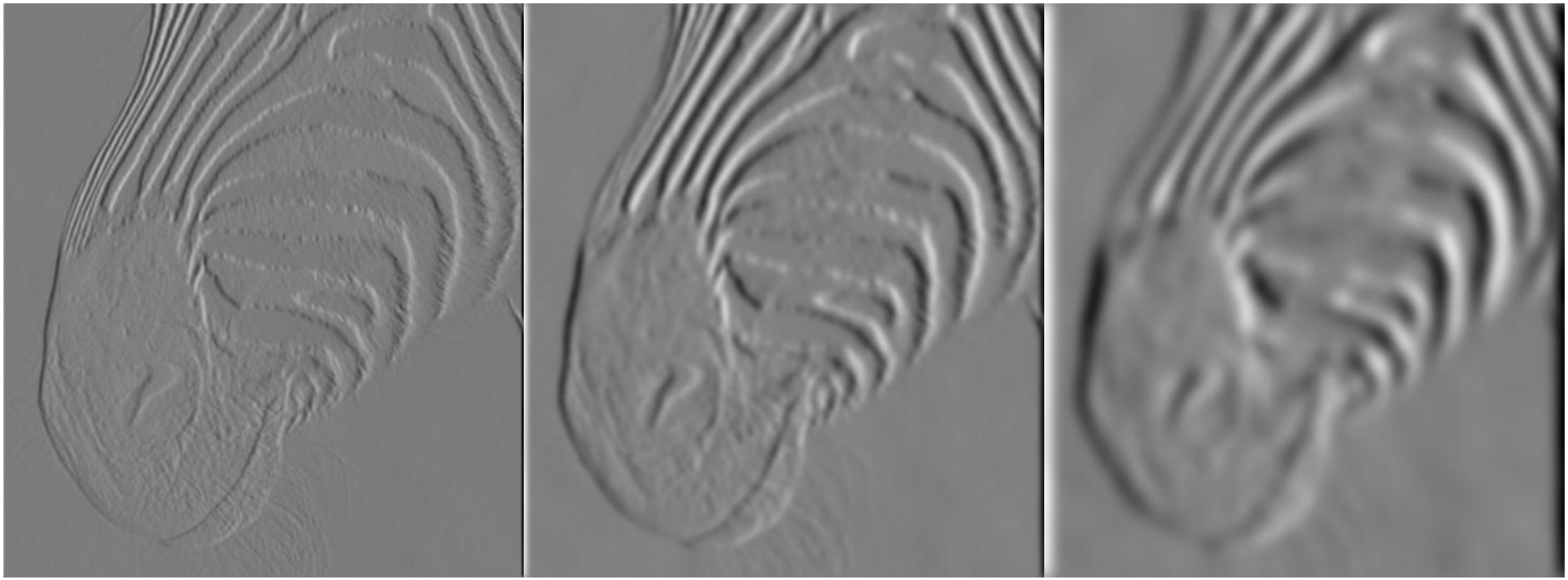


* [1 0 -1] =



- Is this filter separable?

# Tradeoff between smoothing and localization



1 pixel 3 pixels 7 pixels

- Smoothed derivative removes noise, but blurs edge. Also finds edges at different “scales”.

Source: D. Forsyth

# 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

Source: L. Fei-Fei

# Canny edge detector

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

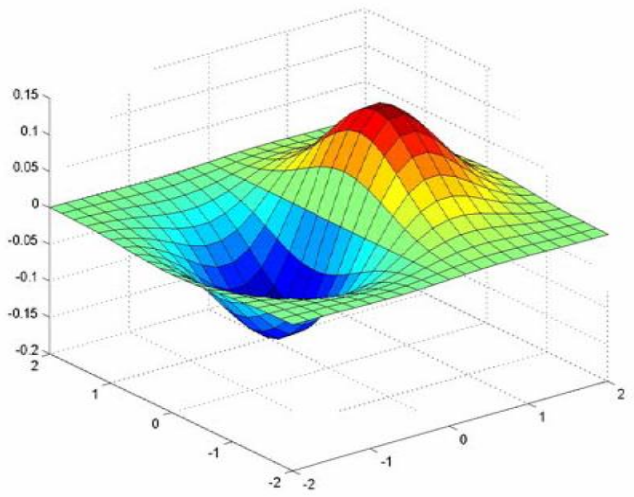
J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Source: L. Fei-Fei

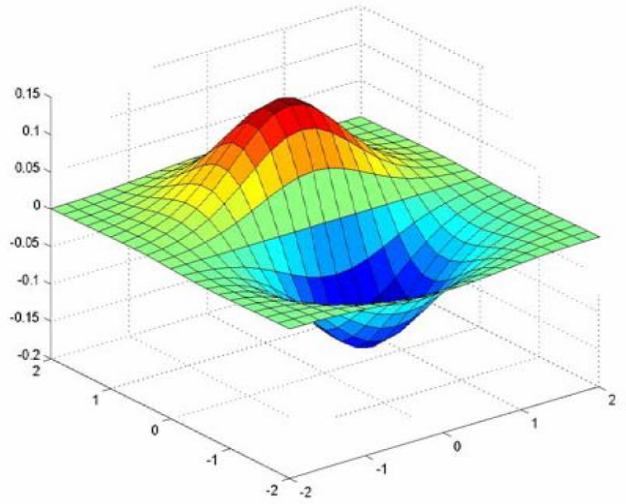
# Example

input image (“Lena”)

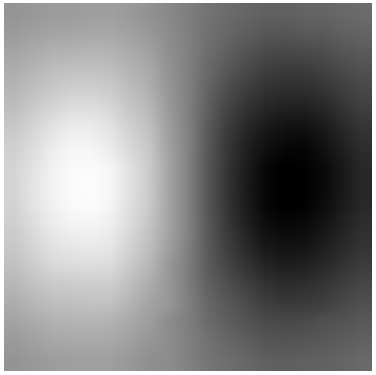
# Derivative of Gaussian filter

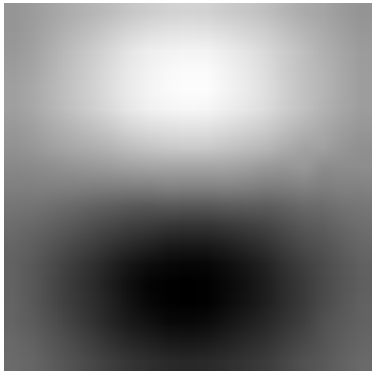



*x*-direction

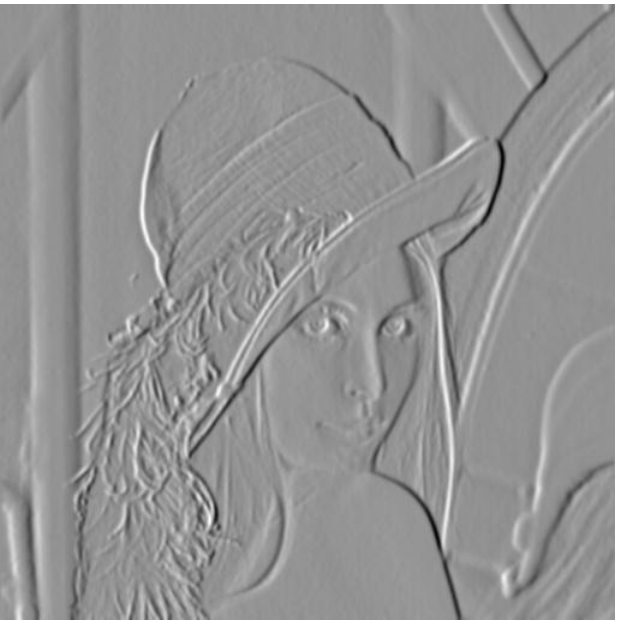



*y*-direction





# Compute Gradients (DoG)



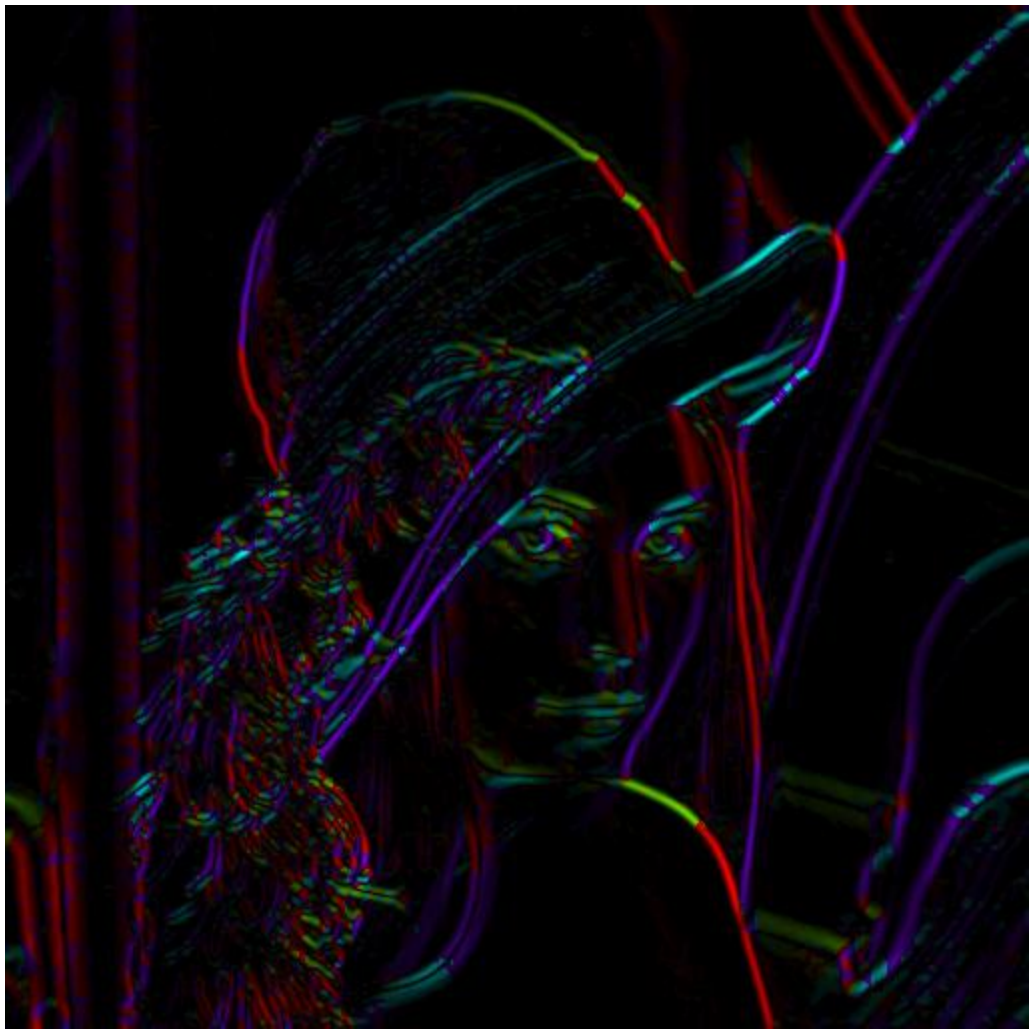
X-Derivative of Gaussian

Y-Derivative of Gaussian

Gradient Magnitude

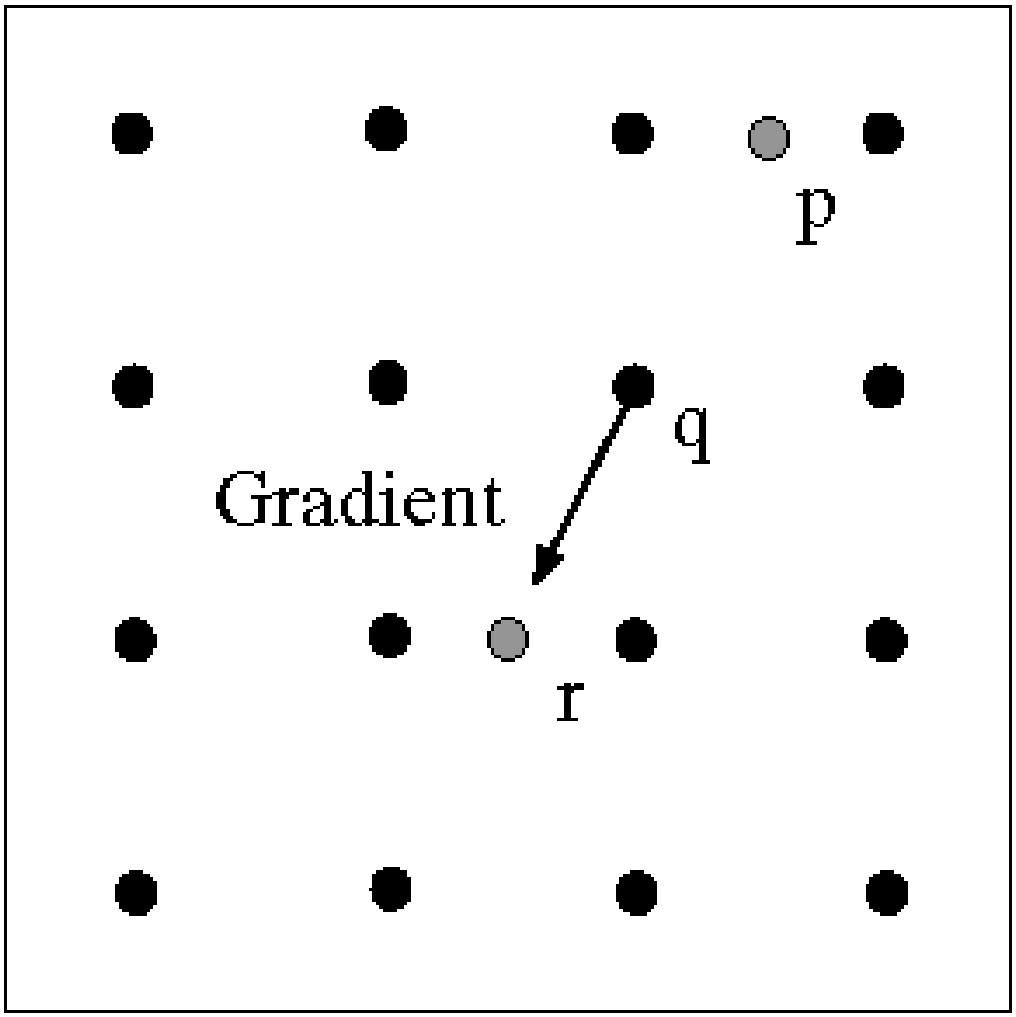
# Get Orientation at Each Pixel

- Threshold at minimum level
- Get orientation

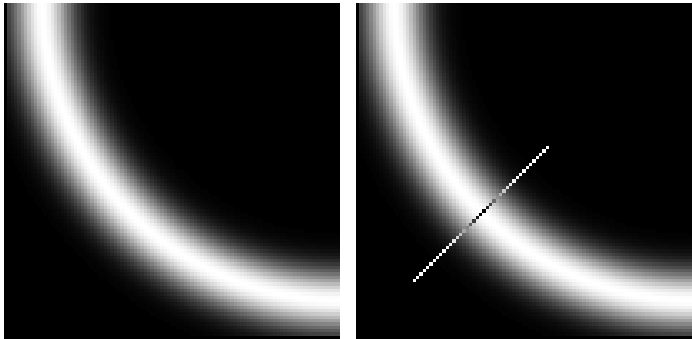


theta = atan2(-gy, gx)

# Non-maximum suppression for each orientation



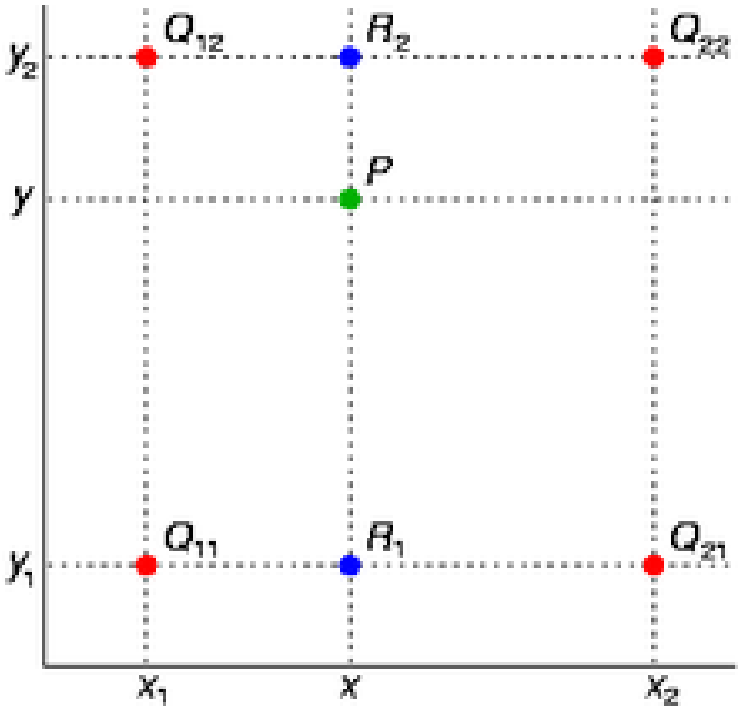

At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.

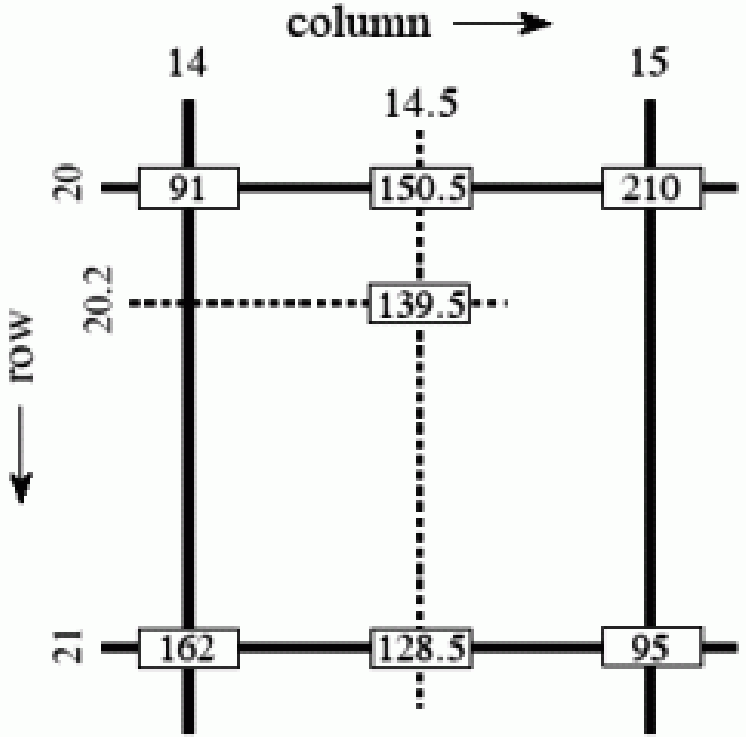


Source: D. Forsyth

# Bilinear Interpolation

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}.$$




http://en.wikipedia.org/wiki/Bilinear_interpolation

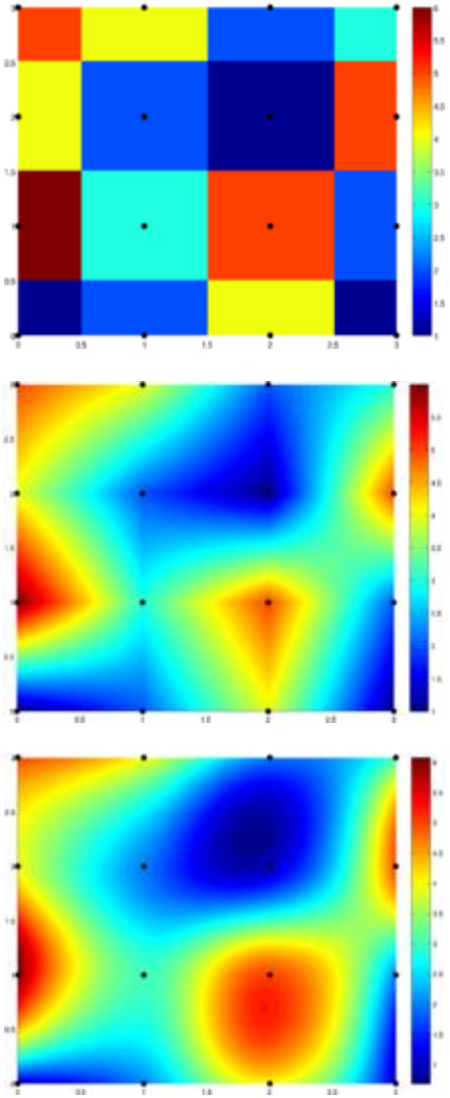
# Sidebar: Interpolation options

- imx2 = imresize(im, 2, interpolation_type)

- ‘nearest’
  - Copy value from nearest known
  - Very fast but creates blocky edges

- ‘bilinear’
  - Weighted average from four nearest known pixels
  - Fast and reasonable results

- ‘bicubic’ (default)
  - Non-linear smoothing over larger area
  - Slower, visually appealing, may create negative pixel values



Examples from http://en.wikipedia.org/wiki/Bicubic_interpolation

# Before Non-max Suppression

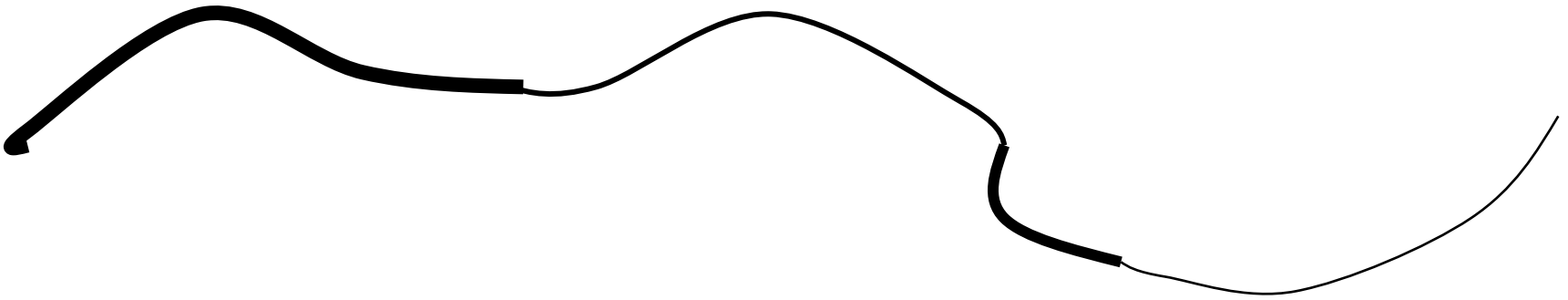
# After non-max suppression

# Hysteresis thresholding

- Threshold at low/high levels to get weak/strong edge pixels
- Do connected components, starting from strong edge pixels

# Hysteresis thresholding

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.



Source: S. Seitz

# Final Canny Edges

# 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” down to single pixel width
4. Thresholding and linking (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

- MATLAB: edge(image, ‘canny’)

Source: D. Lowe, L. Fei-Fei

# Effect of σ (Gaussian kernel spread/size)

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

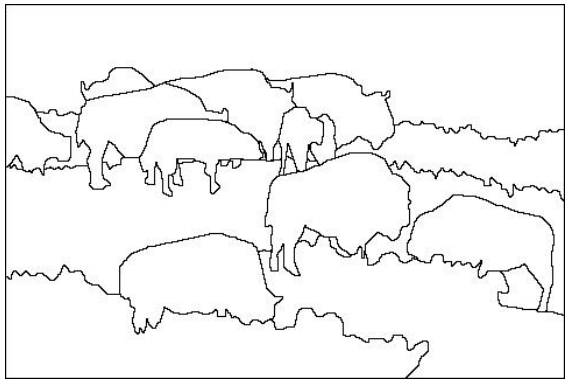
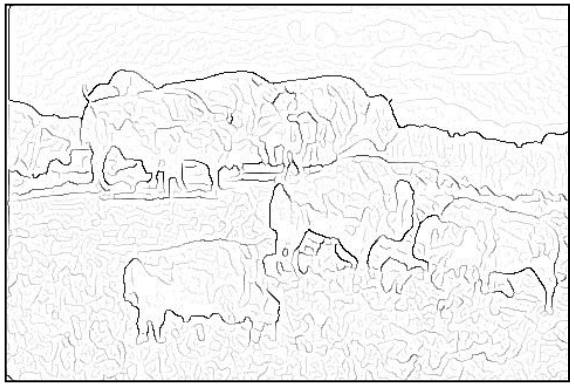
The choice of σ depends on desired behavior

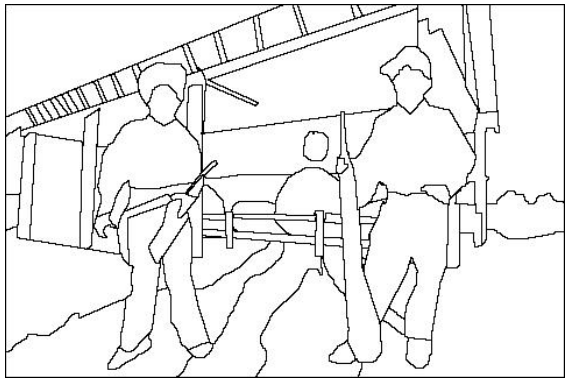
- large σ detects large scale edges
- small σ detects fine features

Source: S. Seitz

# Learning to detect boundaries

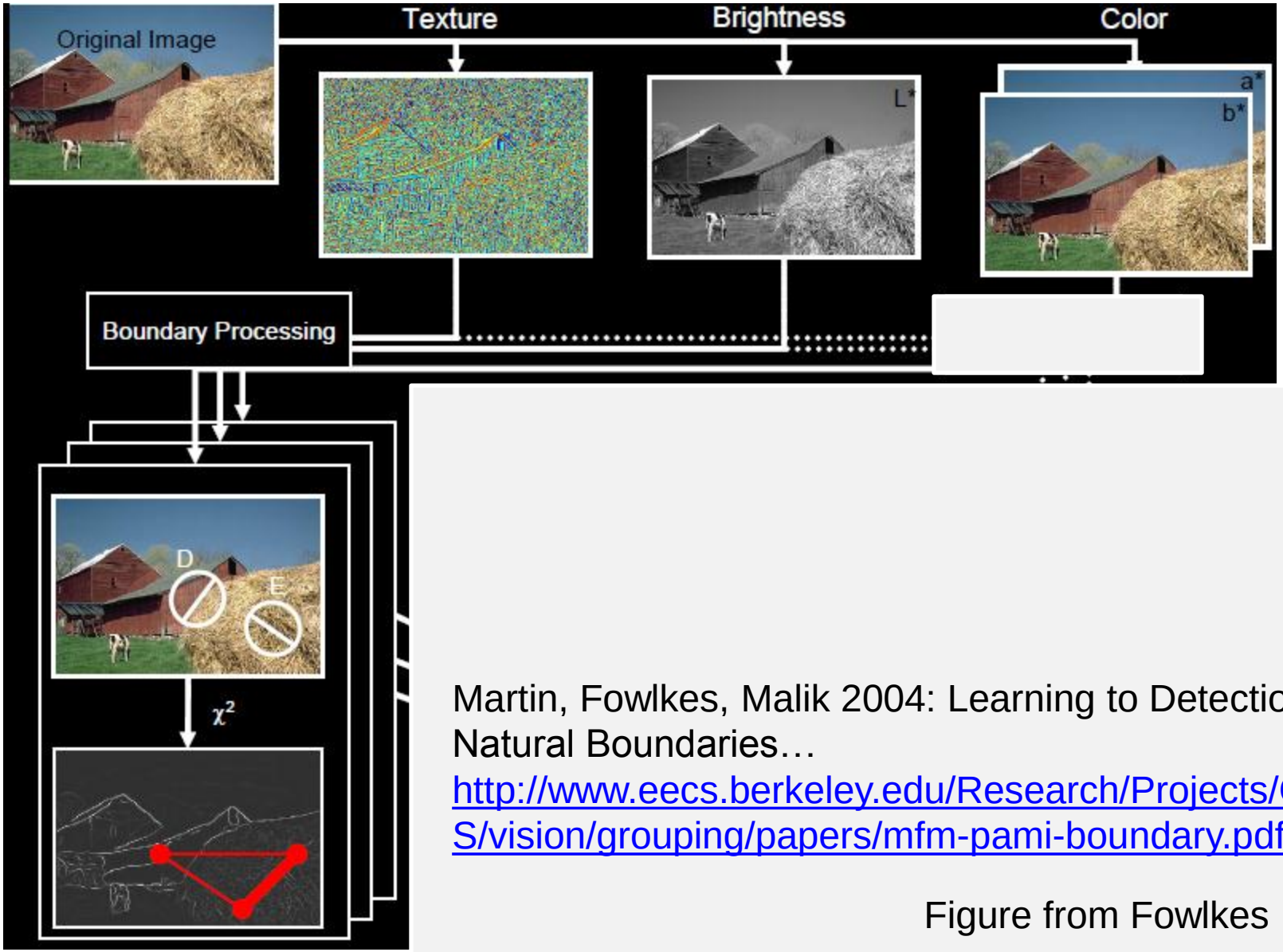
image | human segmentation | gradient magnitude


- Berkeley segmentation database:
  http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

# pB boundary detector




Martin, Fowlkes, Malik 2004: Learning to Detection Natural Boundaries…
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/papers/mfm-pami-boundary.pdf

Figure from Fowlkes

# pB Boundary Detector

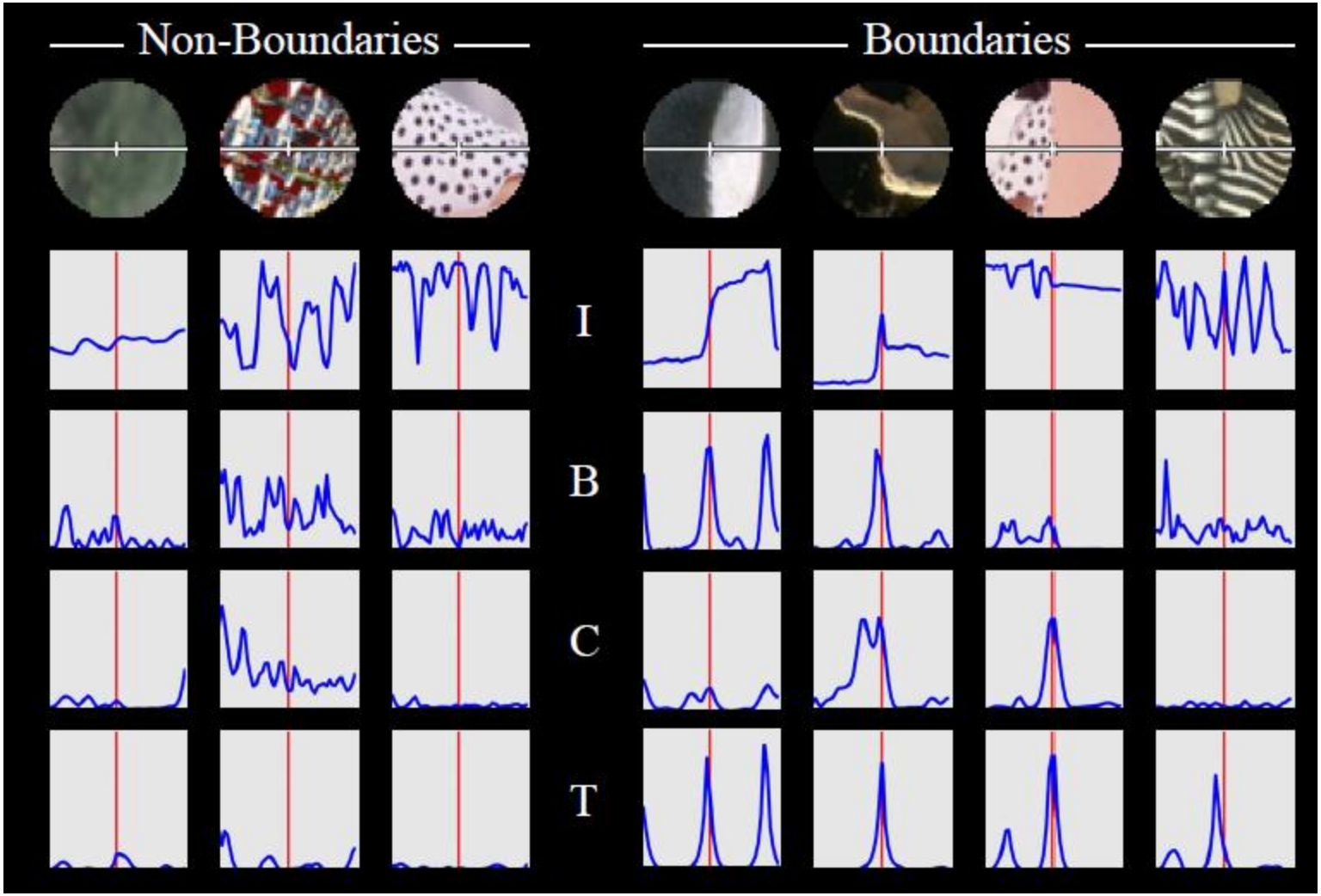



Figure from Fowlkes

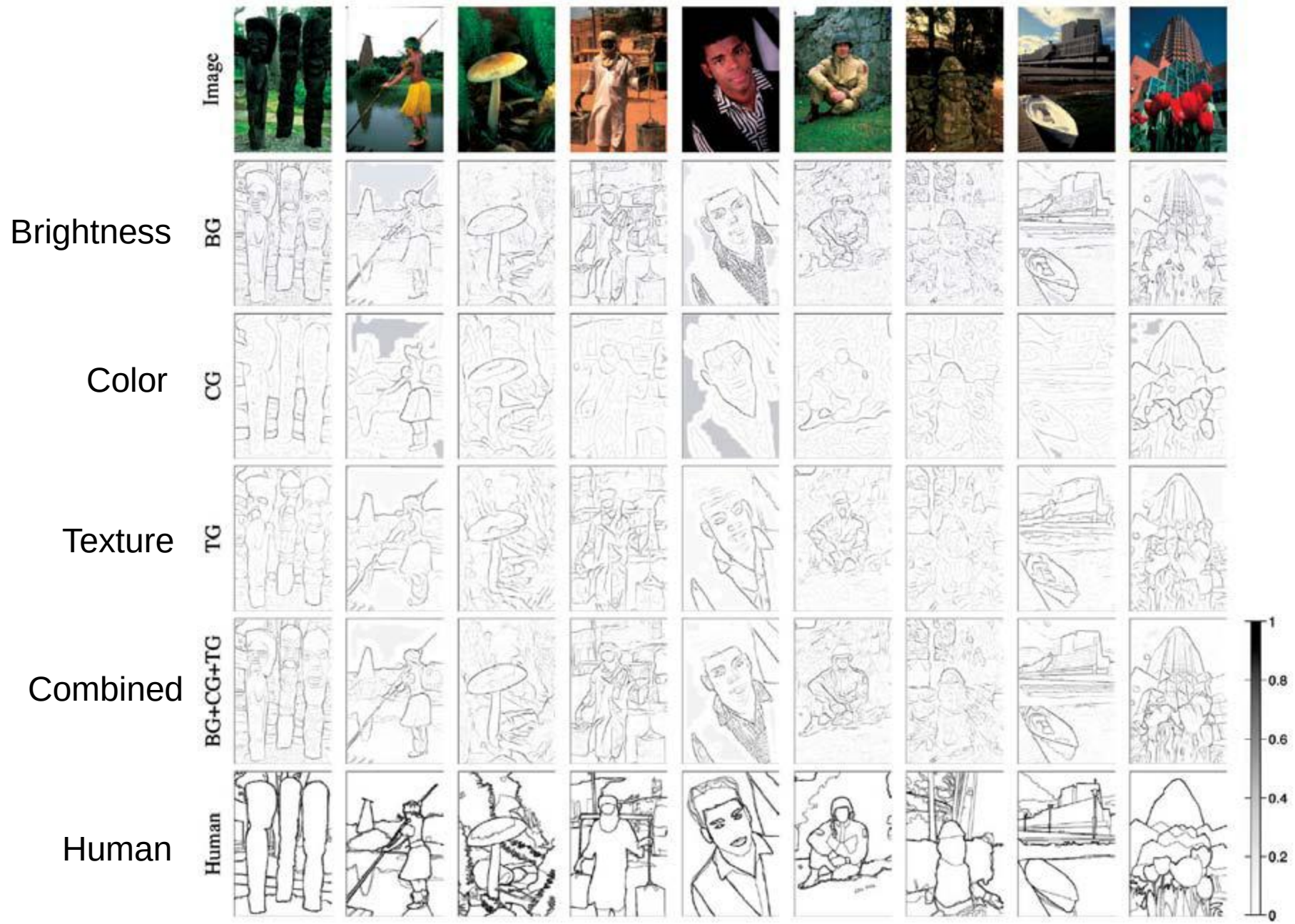
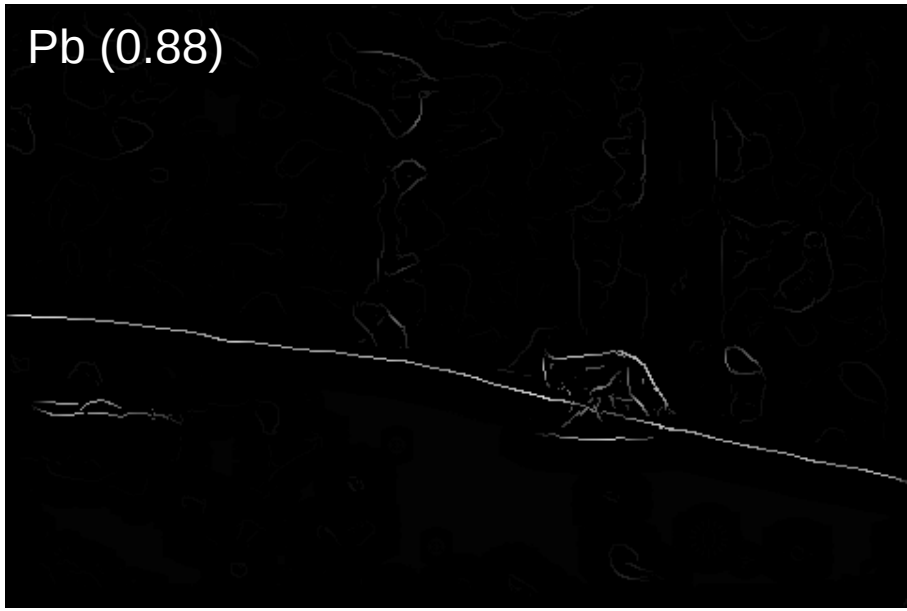
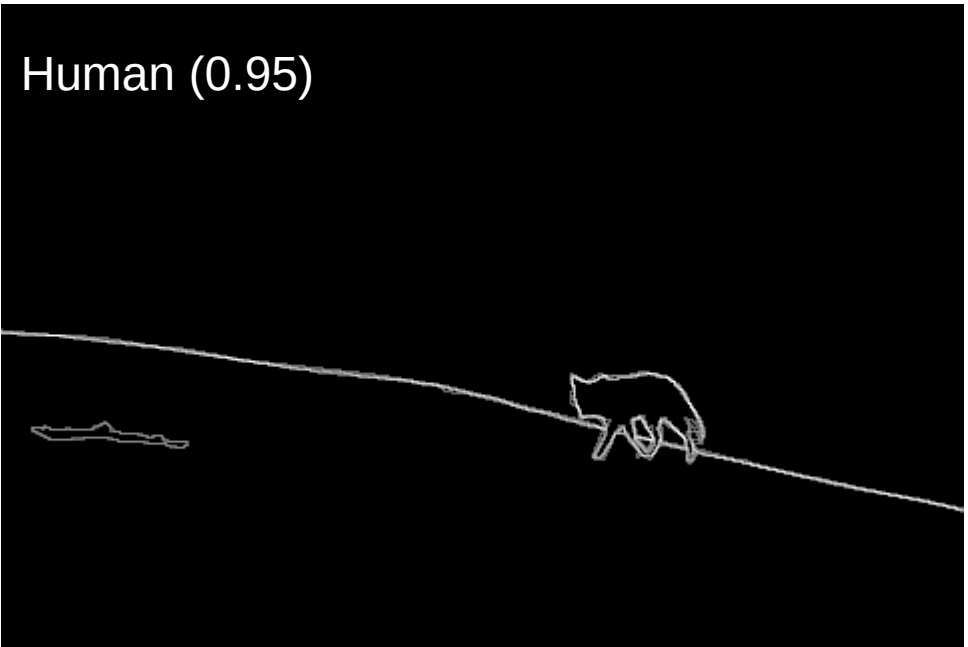


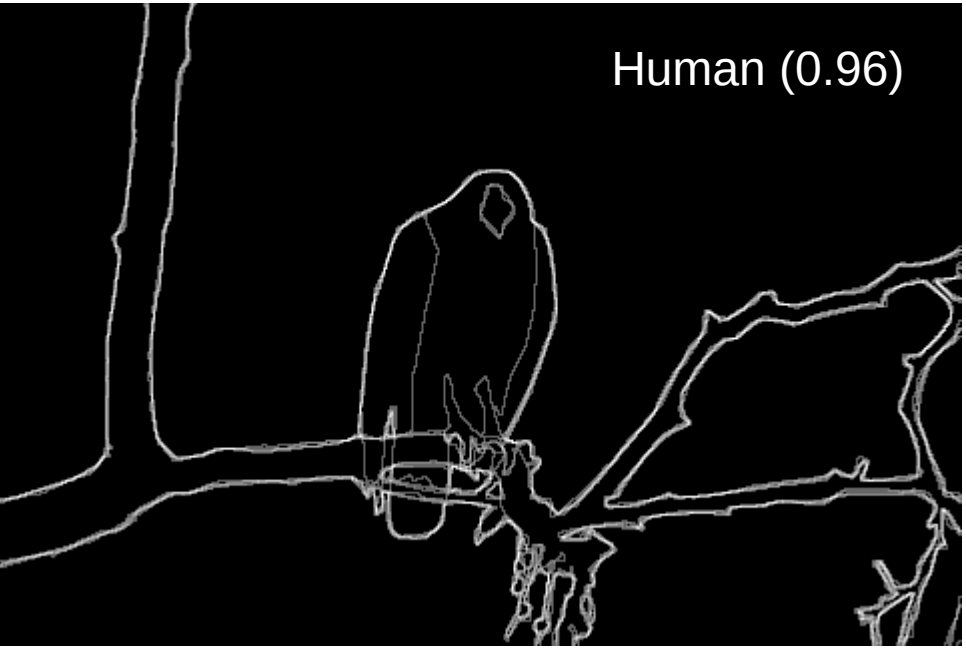
Image
BG
CG
TG
BG+CG+TG
Human
Brightness
Color
Texture
Combined
Human
1
0.8
0.6
0.4
0.2
0

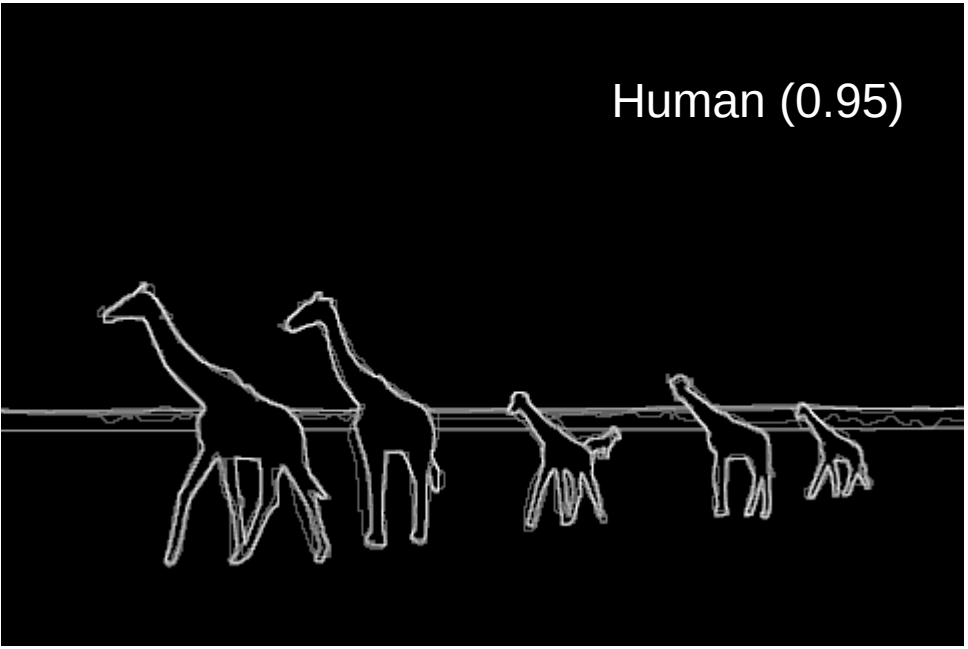
# Results



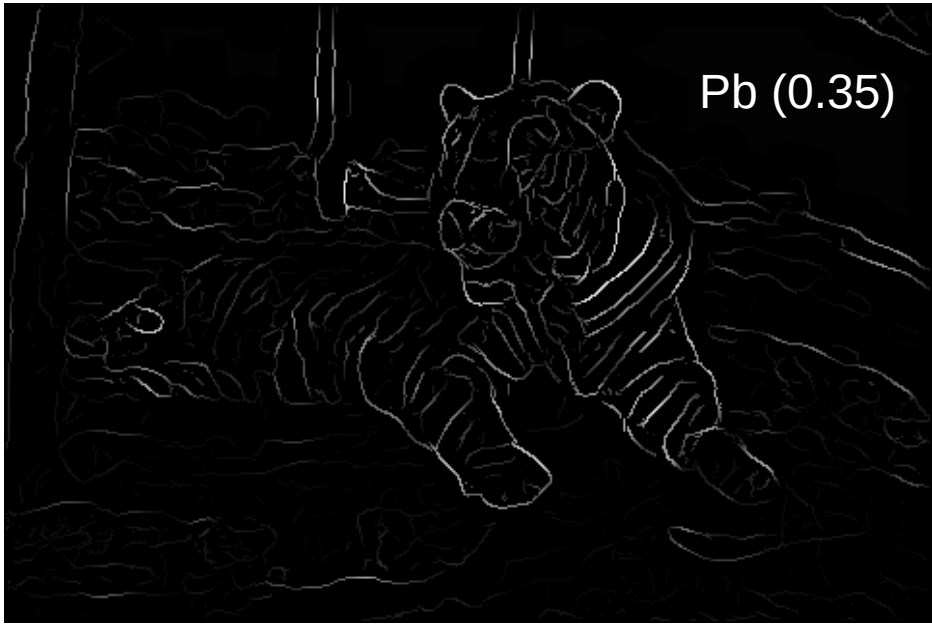



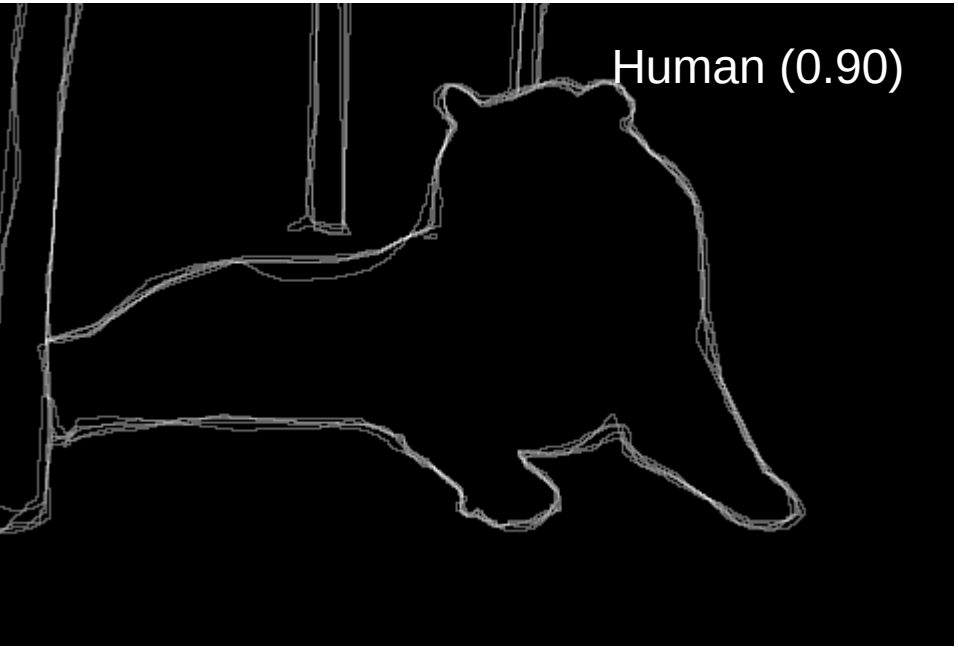
# Results





Human (0.95)


Pb (0.63)




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

# Global pB boundary detector

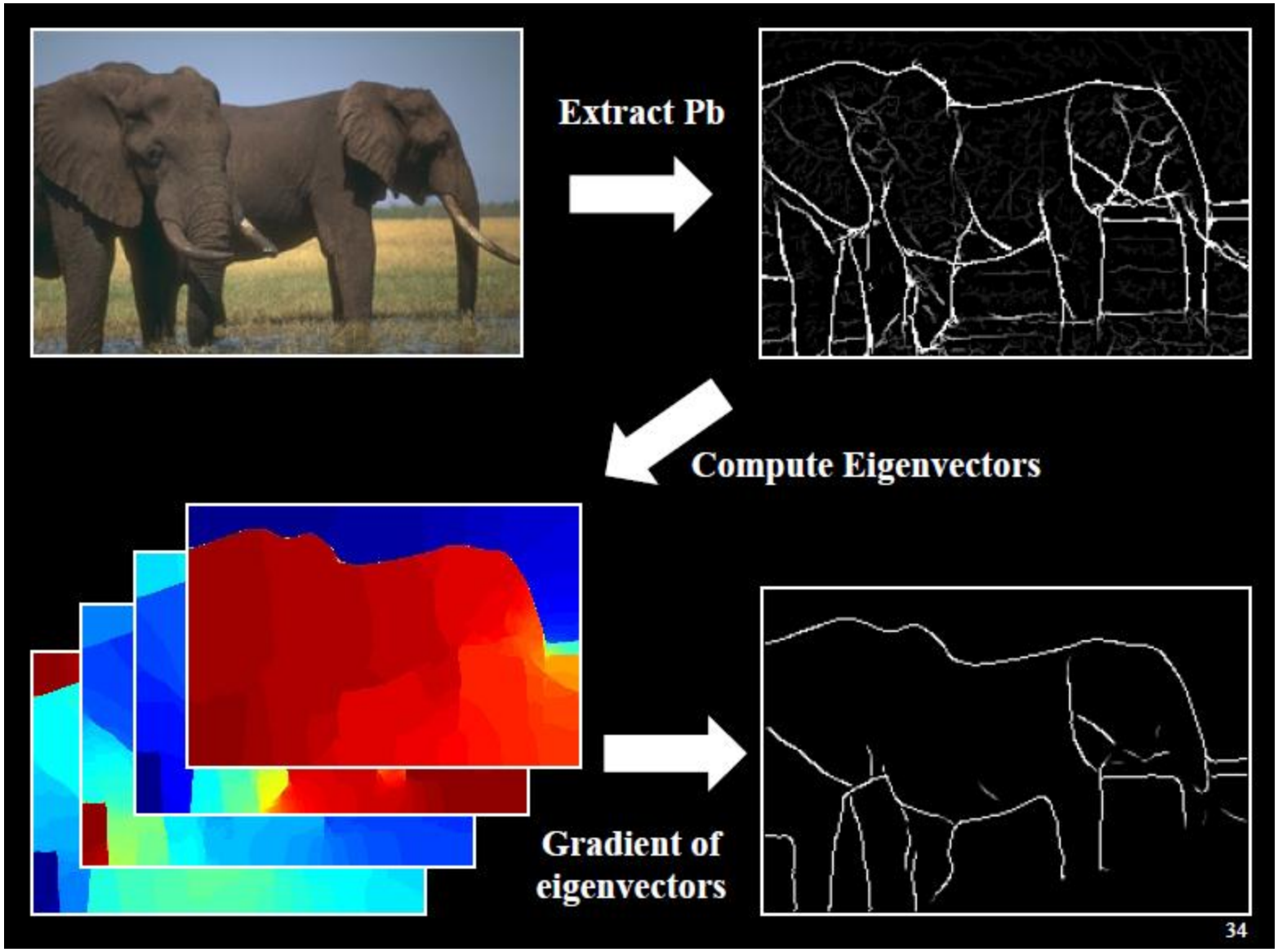



Figure from Fowlkes

# State of edge detection

- Local edge detection is mostly solved
  - Intensity **gradient**, color, **texture**
- Some methods to take into account longer contours, but could probably do better
- Poor use of object and high-level information

# Finding straight lines

# Finding line segments using connected components

1. Compute canny edges
   - Compute: gx, gy (DoG in x,y directions)
   - Compute: theta = atan(gy / gx)
2. Assign each edge to one of 8 directions
3. For each direction d, get edgelets:
   - find connected components for edge pixels with directions in {d-1, d, d+1}
4. Compute straightness and theta of edgelets using eig of x,y 2nd moment matrix of their points

$$\mathbf{M} = \begin{bmatrix} \sum (x-\mu_x)^2 & \sum (x-\mu_x)(y-\mu_y) \\ \sum (x-\mu_x)(y-\mu_y) & \sum (y-\mu_y)^2 \end{bmatrix} \qquad [v, \lambda] = \text{eig}(\mathbf{M})$$

Larger eigenvector

$$\theta = \text{atan}\,2(v(2,2), v(1,2))$$

$$conf = \lambda_2 / \lambda_1$$

5. Threshold on straightness, store segment

# 2. Canny lines → … →straight edges

# Homework 1

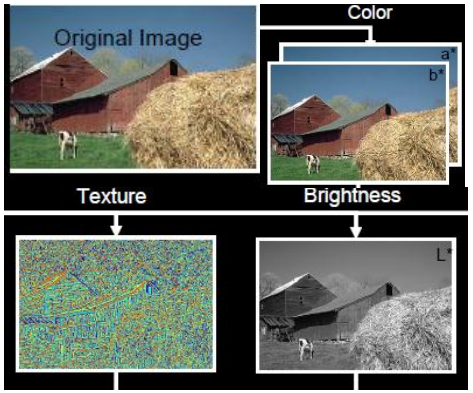
- Due Feb 14, but try to finish by Tues (HW 2 will take quite a bit more time)

http://www.cs.illinois.edu/class/sp12/cs543/hw/CV_Spring12_HW1.pdf

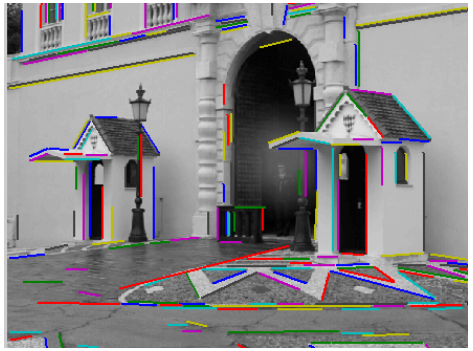
# Things to remember

- Canny edge detector = smooth → derivative → thin → threshold → link

- Pb: learns weighting of gradient, color, texture differences




- Straight line detector = canny + gradient orientations → orientation binning → linking → check for straightness



# Next classes: Correspondence and Alignment

- Detecting interest points
- Tracking points
- Object/image alignment and registration
  - Aligning 3D or edge points
  - Object instance recognition
  - Image stitching