02/02/17

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

Creating the Gaussian/Laplacian Pyramid

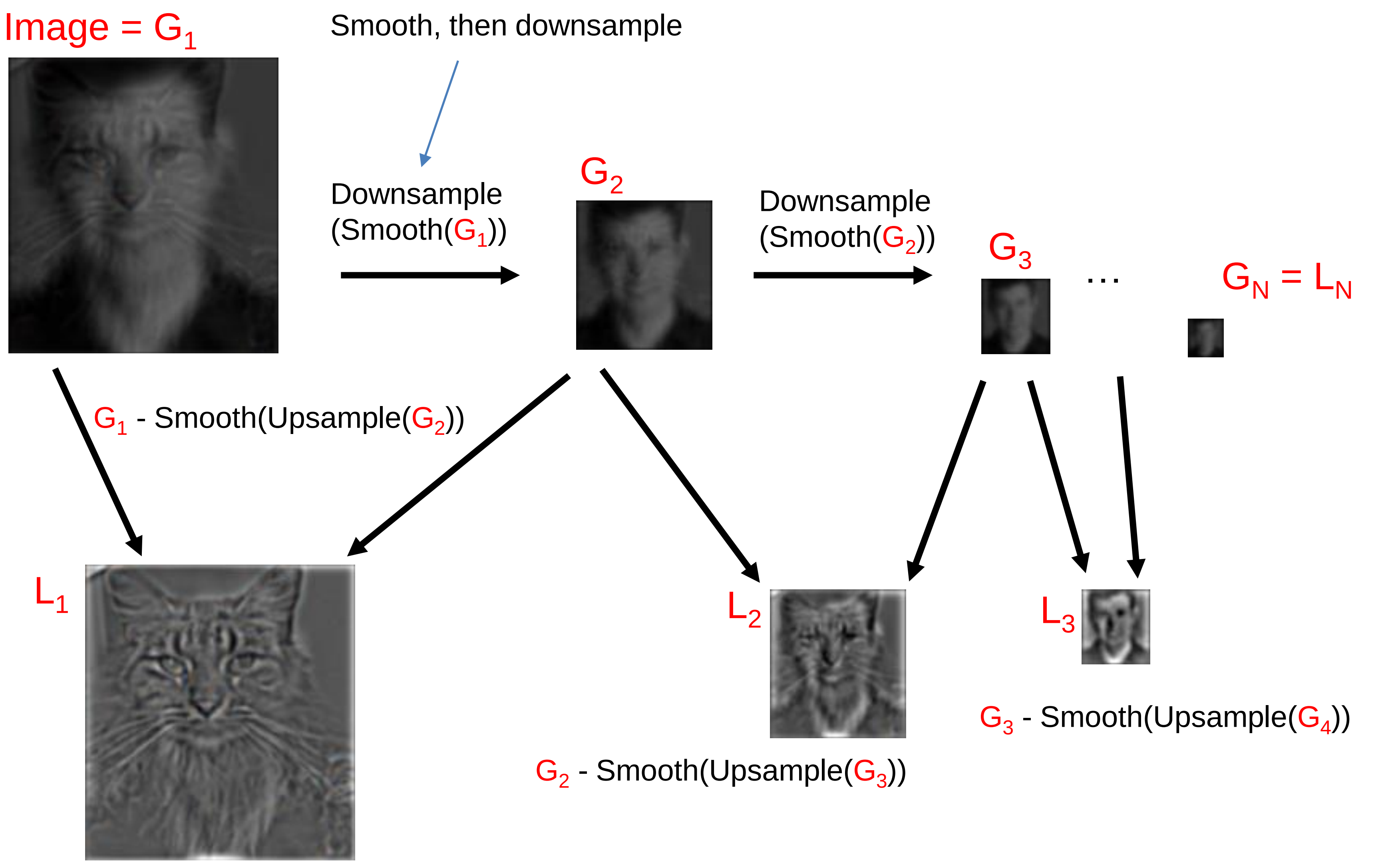


Image = G_1
Smooth, then downsample
Downsample (Smooth(G_1))
G_2
Downsample (Smooth(G_2))
G_3
…
$G_N = L_N$
G_1 - Smooth(Upsample(G_2))
L_1
L_2
L_3
G_3 - Smooth(Upsample(G_4))
G_2 - Smooth(Upsample(G_3))

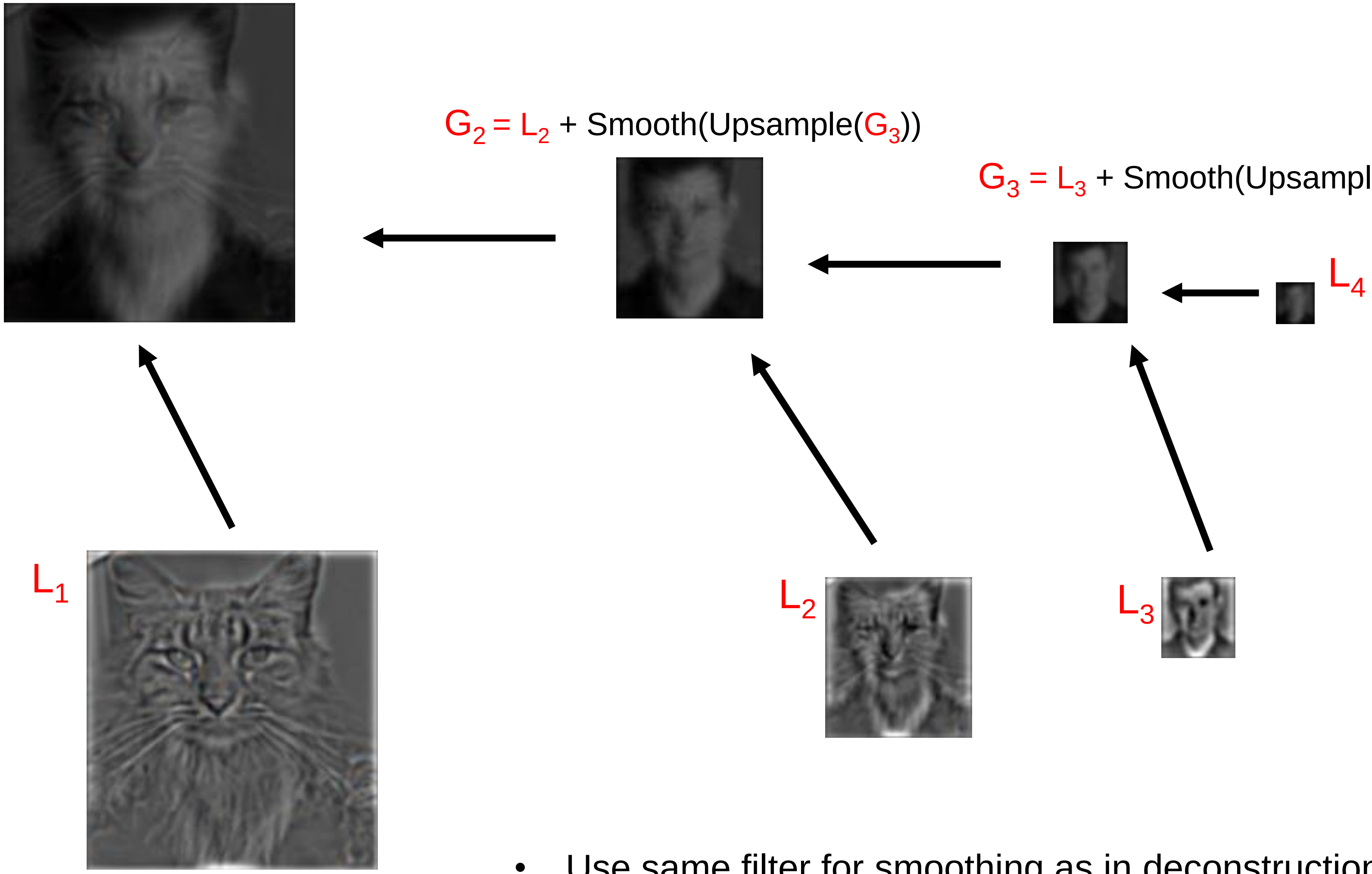
- Use same filter for smoothing in each step (e.g., Gaussian with $\sigma = 2$)
- Downsample/upsample with “nearest” interpolation

Reconstructing image from Laplacian pyramid

$\text{Image} = L_1 + \text{Smooth}(\text{Upsample}(G_2))$

$G_2 = L_2 + \text{Smooth}(\text{Upsample}(G_3))$

$G_3 = L_3 + \text{Smooth}(\text{Upsample}(L_4))$



- Use same filter for smoothing as in deconstruction
- Upsample with “nearest” interpolation
- Reconstruction will be lossless

Comments/questions from earlier

- Computational complexity of coarse-to-fine search?

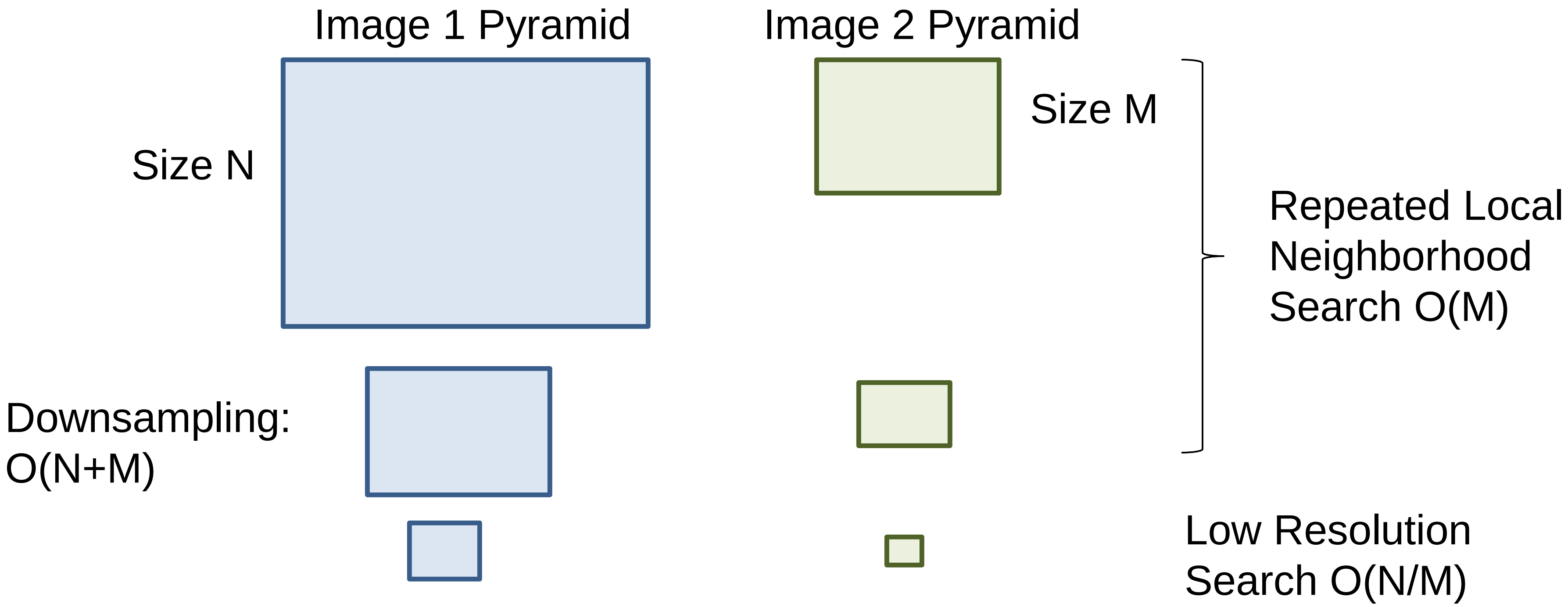
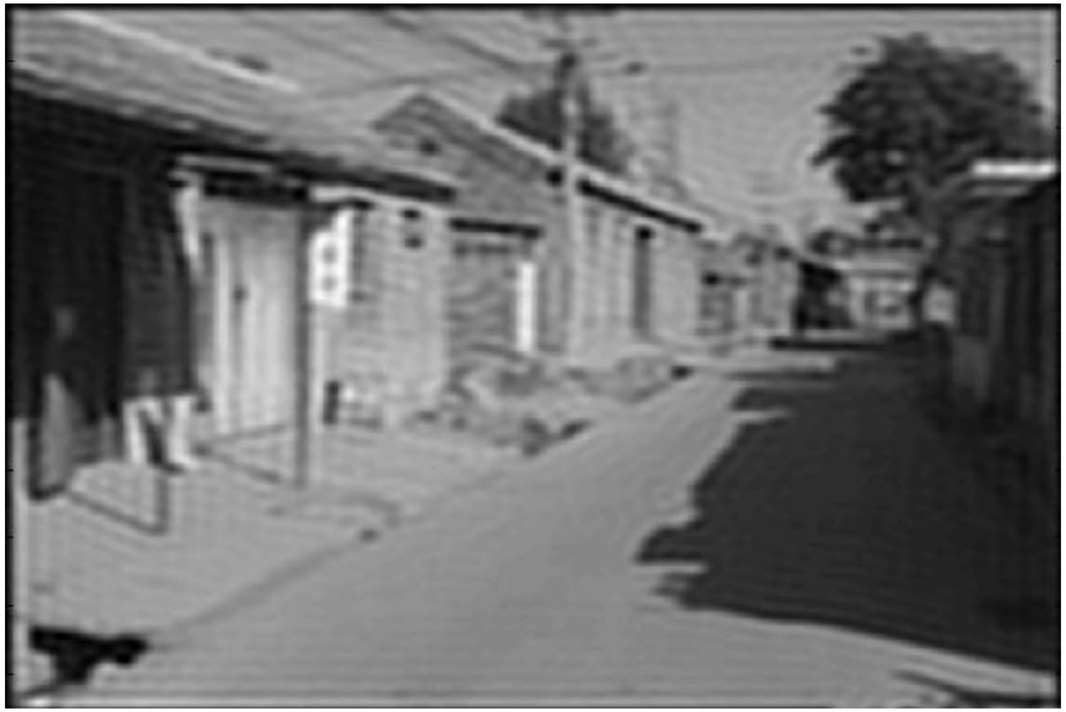


Image 1 Pyramid
Image 2 Pyramid
Size N
Size M
Repeated Local Neighborhood Search O(M)
Downsampling: O(N+M)
Low Resolution Search O(N/M)

Overall complexity: O(N+M)
Original high-resolution full search: O(NM) or O(N logN)

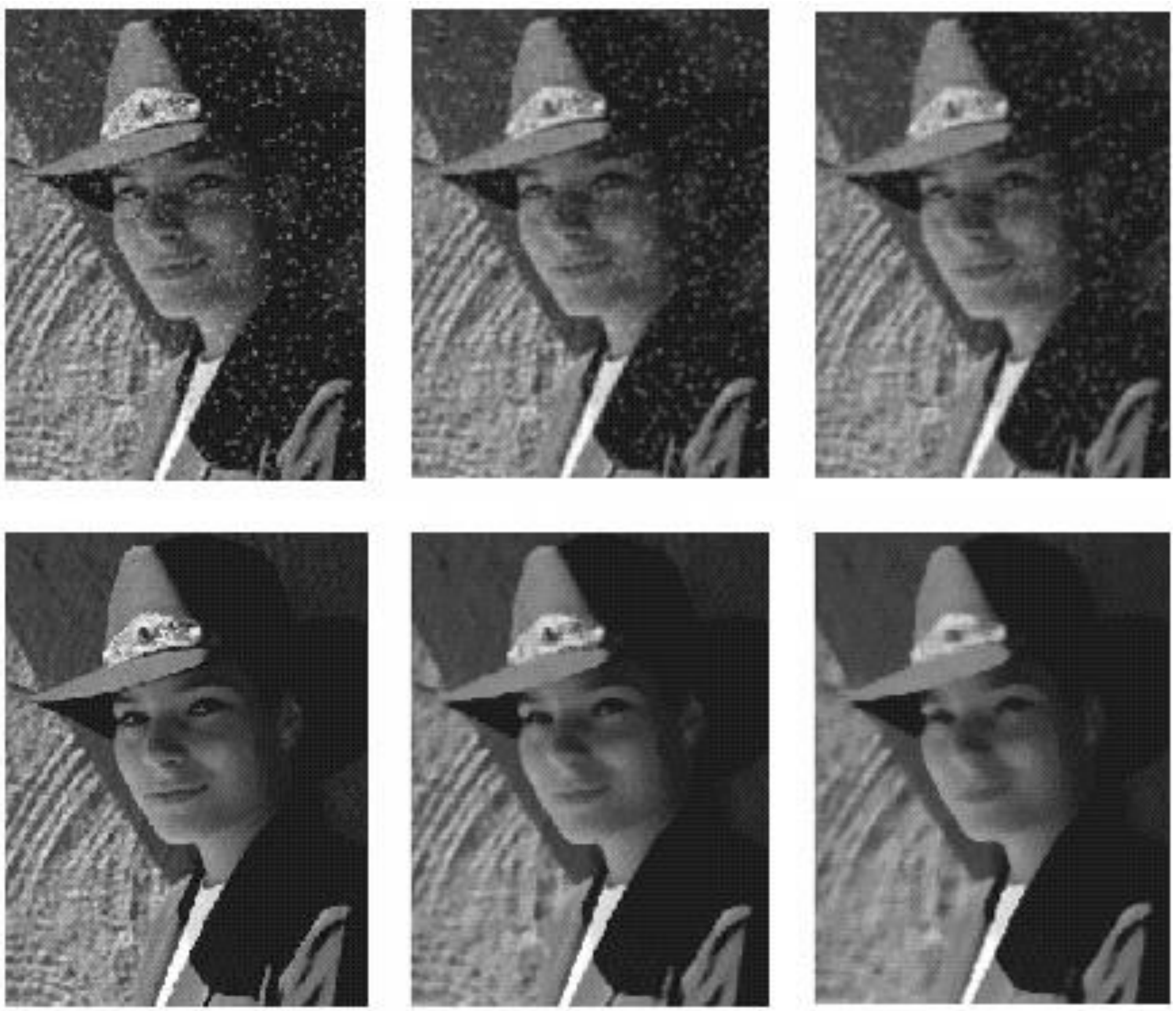
- Why not use an ideal filter?

Answer: has infinite spatial extent, clipping results in ringing



Attempt to apply ideal filter in frequency domain

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)

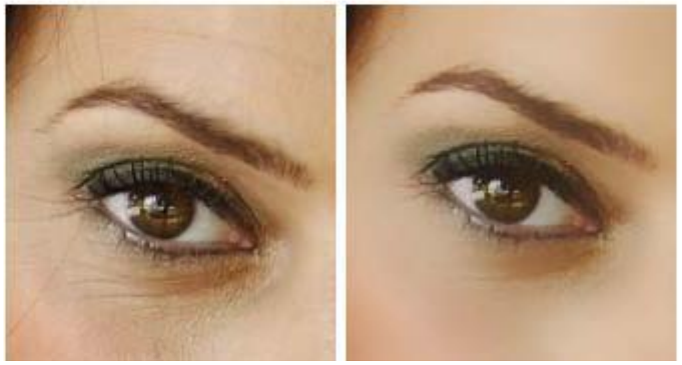

Bilateral filtering

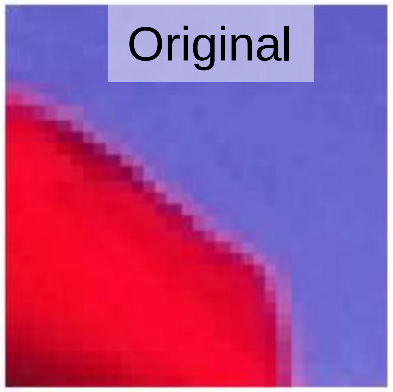
Image: http://vision.ai.uiuc.edu/?p=1455

Bilateral filters

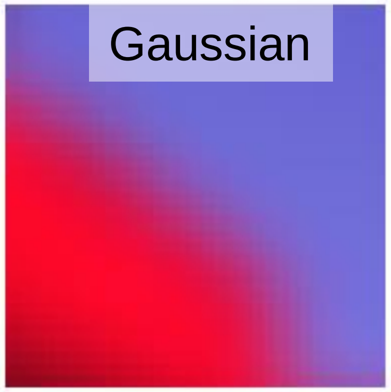
- Edge preserving: weights similar pixels more

$$I^{\mathrm{b}}_{\mathbf{p}} = \frac{1}{W^{\mathrm{b}}_{\mathbf{p}}} \sum_{\mathbf{q}\in\mathcal{S}} \underbrace{G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|)}^{\text{spatial}} \, \overbrace{G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)}^{\text{similarity (e.g., intensity)}} \, I_{\mathbf{q}}$$

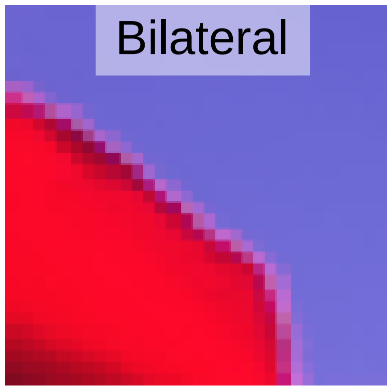
$$\text{with} \quad W^{\mathrm{b}}_{\mathbf{p}} = \sum_{\mathbf{q}\in\mathcal{S}} G_{\sigma_{\mathrm{s}}}(\|\mathbf{p}-\mathbf{q}\|) \, G_{\sigma_{\mathrm{r}}}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)$$



(a)



(b)



(c)

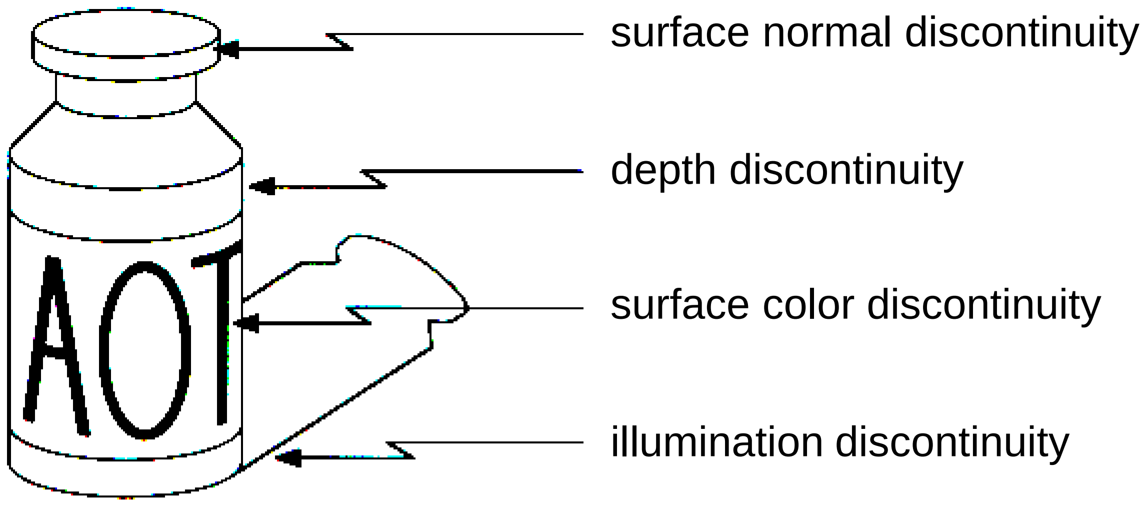
Carlo Tomasi, Roberto Manduchi, Bilateral Filtering for Gray and Color Images, ICCV, 1998.

Today’s class

- Detecting edges

- Finding straight lines

Origin of Edges


AOT
surface normal discontinuity
depth discontinuity
surface color discontinuity
illumination discontinuity

Edges are caused by a variety of factors

Source: Steve Seitz

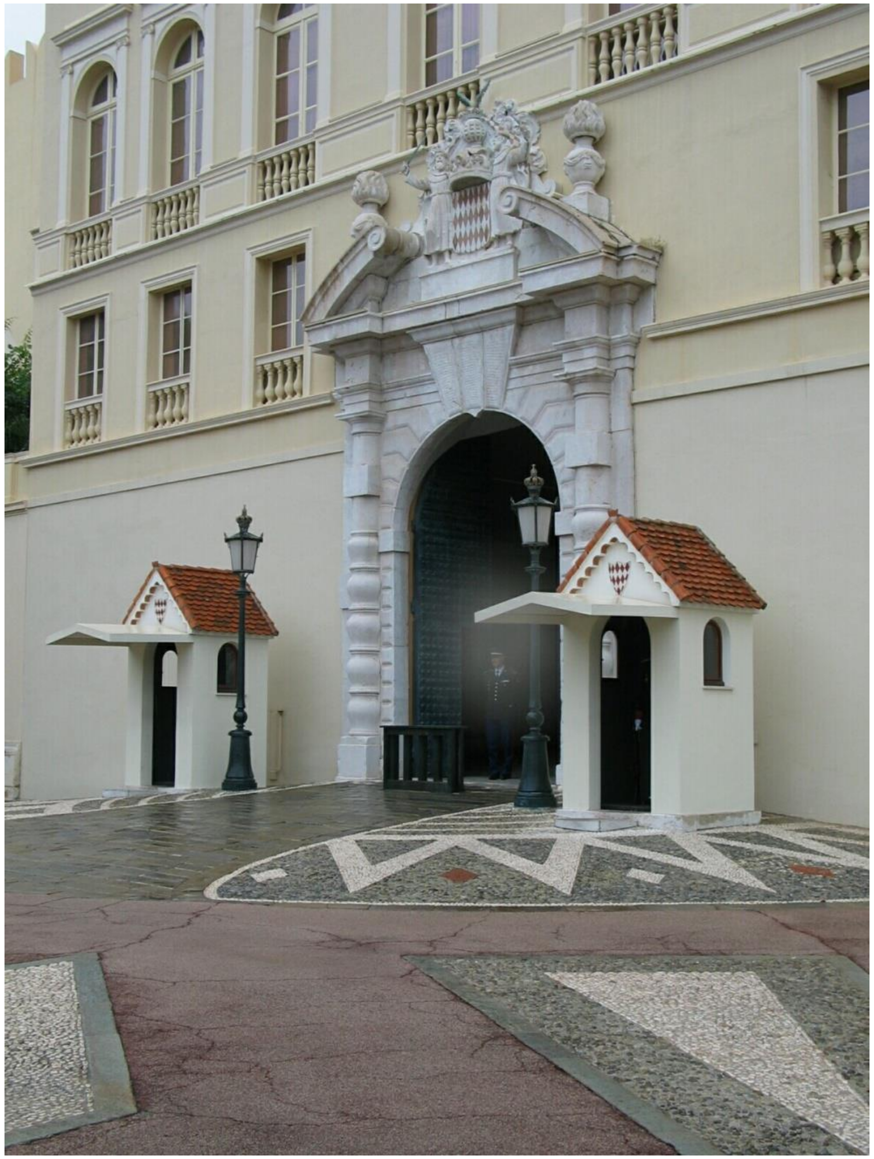
Why finding edges is important

- Group pixels into objects or parts

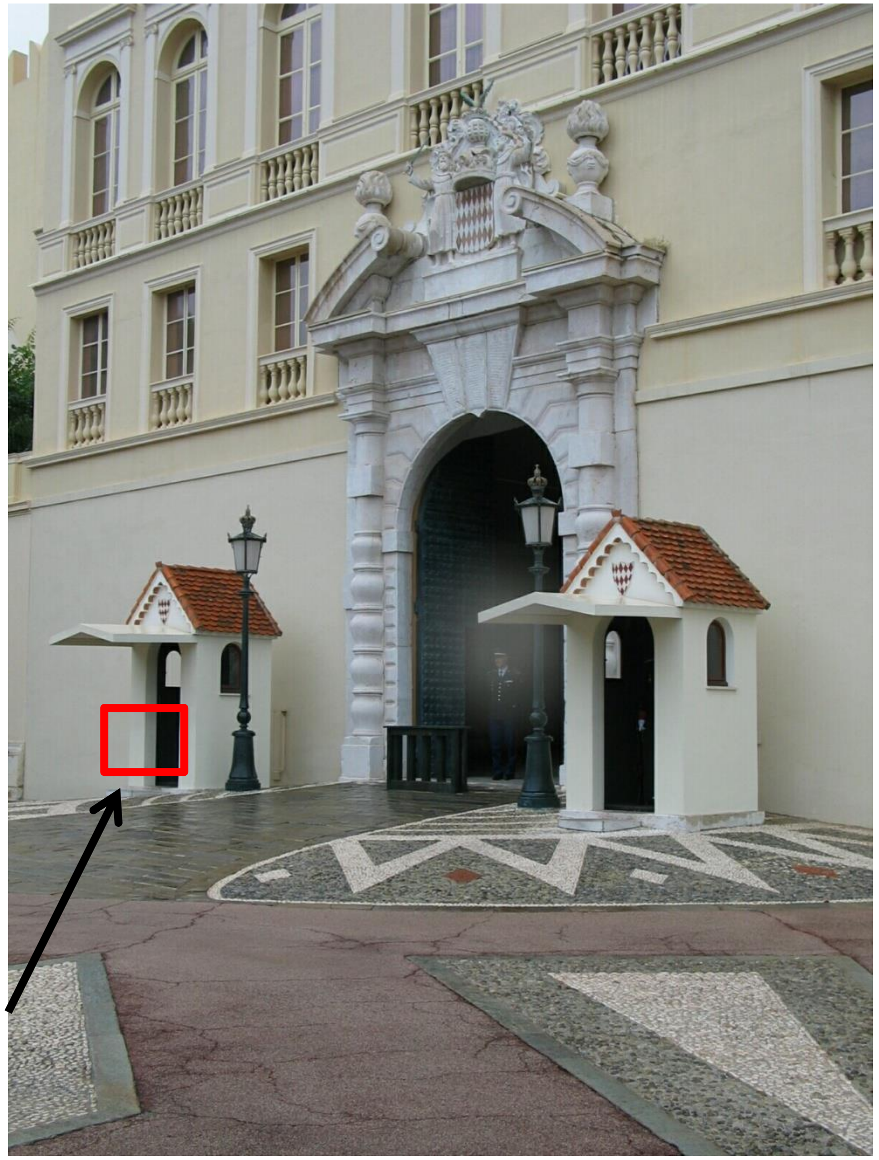
- Cues for 3D shape

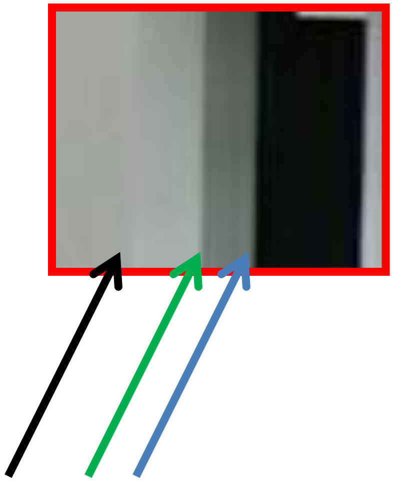
- Guiding interactive image editing

Closeup of edges

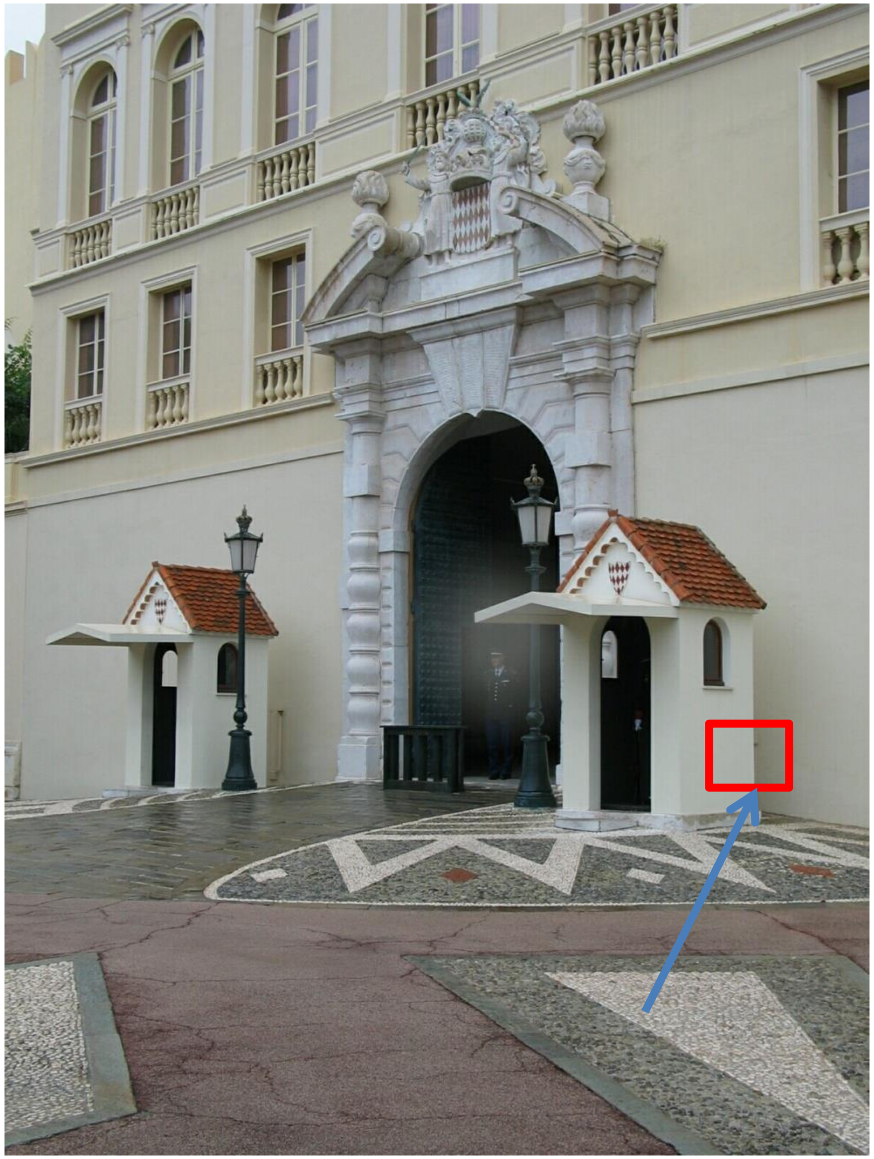


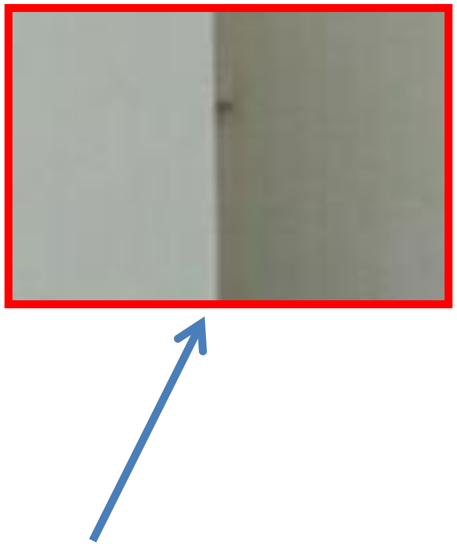
Closeup of edges



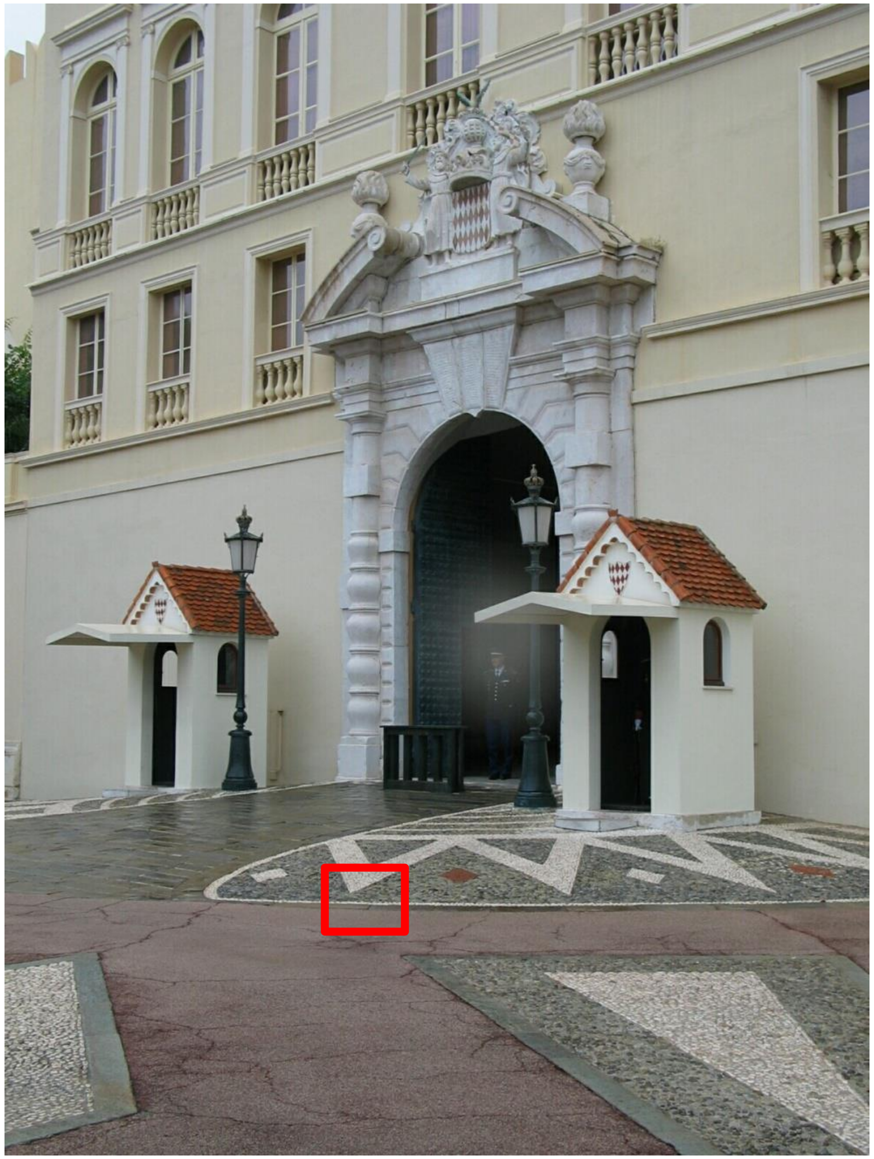


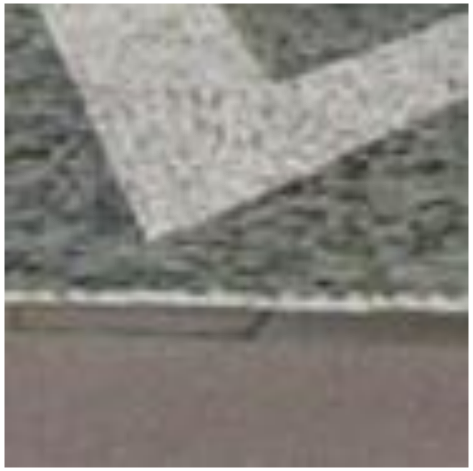
Closeup of edges





Closeup of edges





Characterizing edges

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

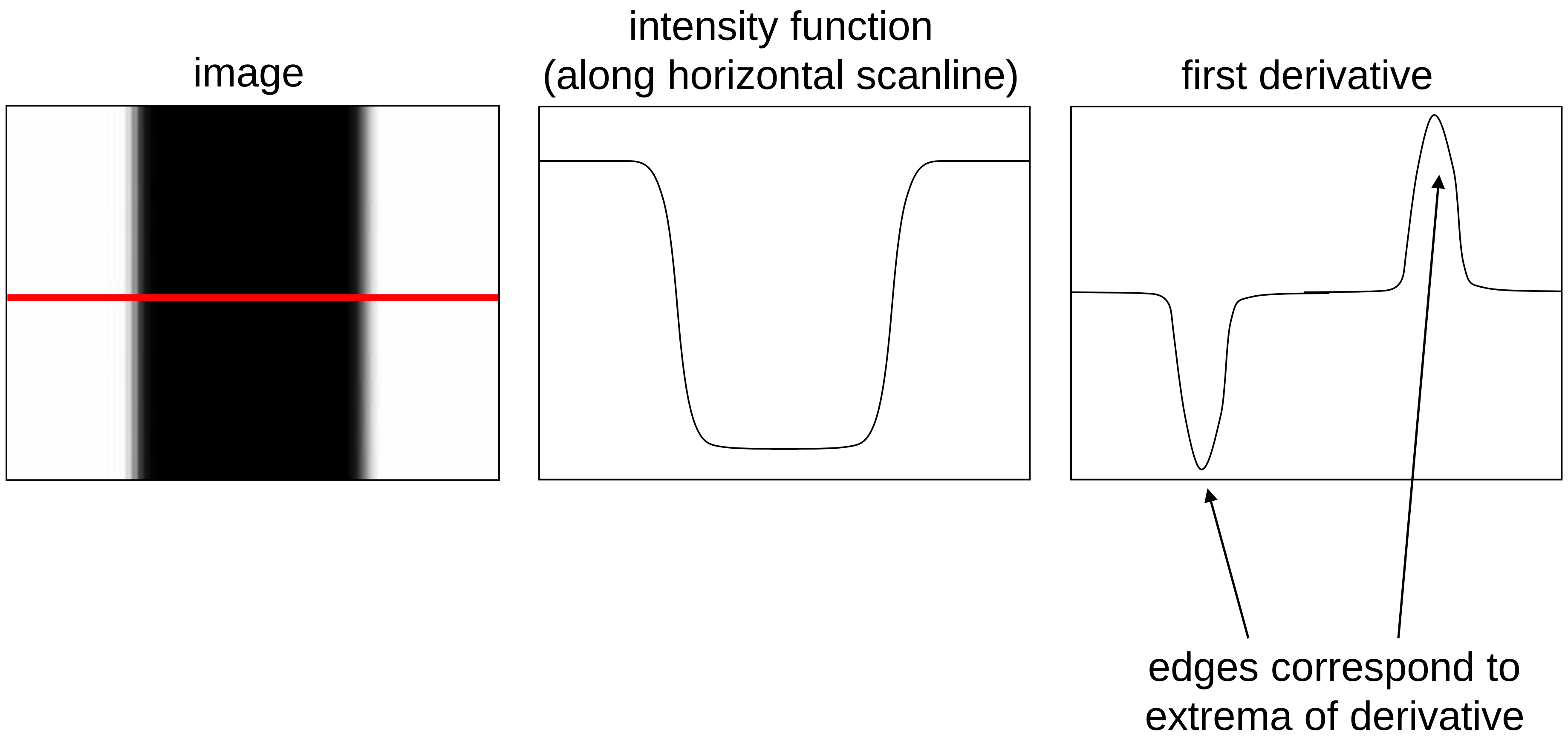
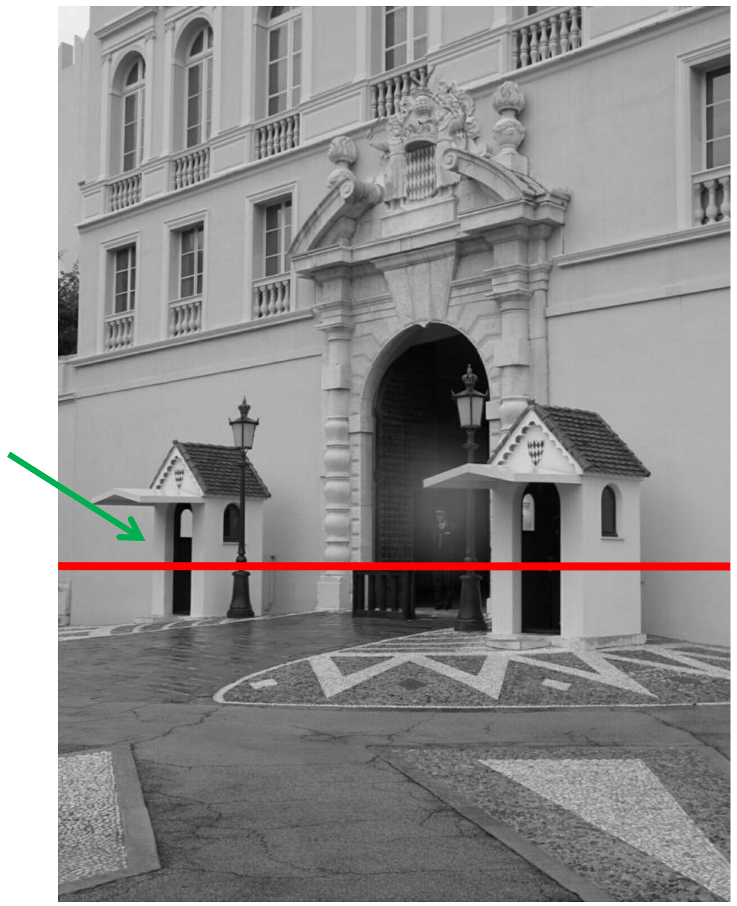
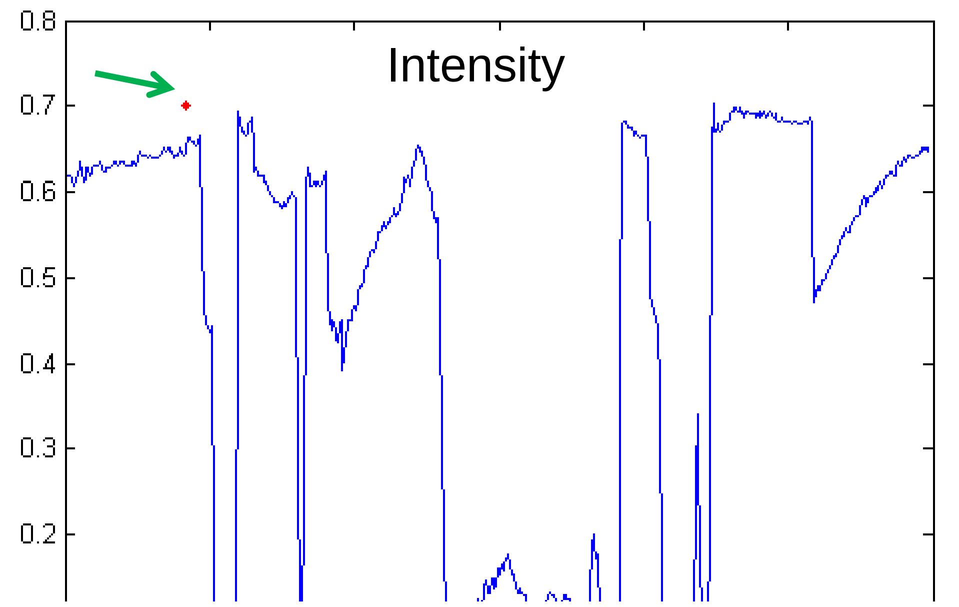
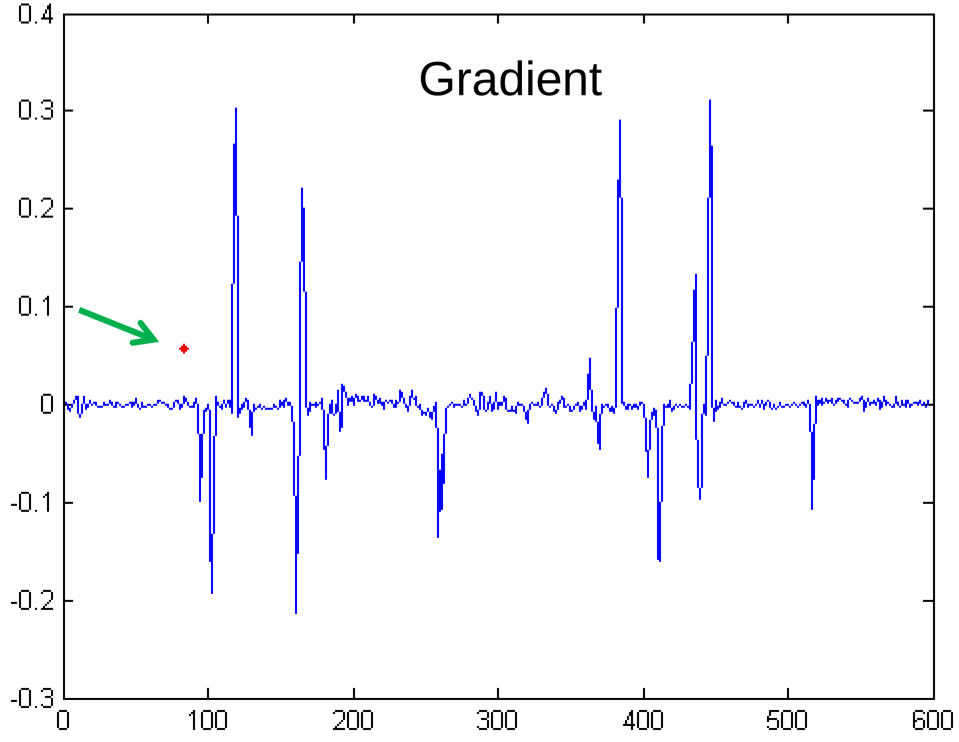


image
intensity function (along horizontal scanline)
first derivative
edges correspond to extrema of derivative

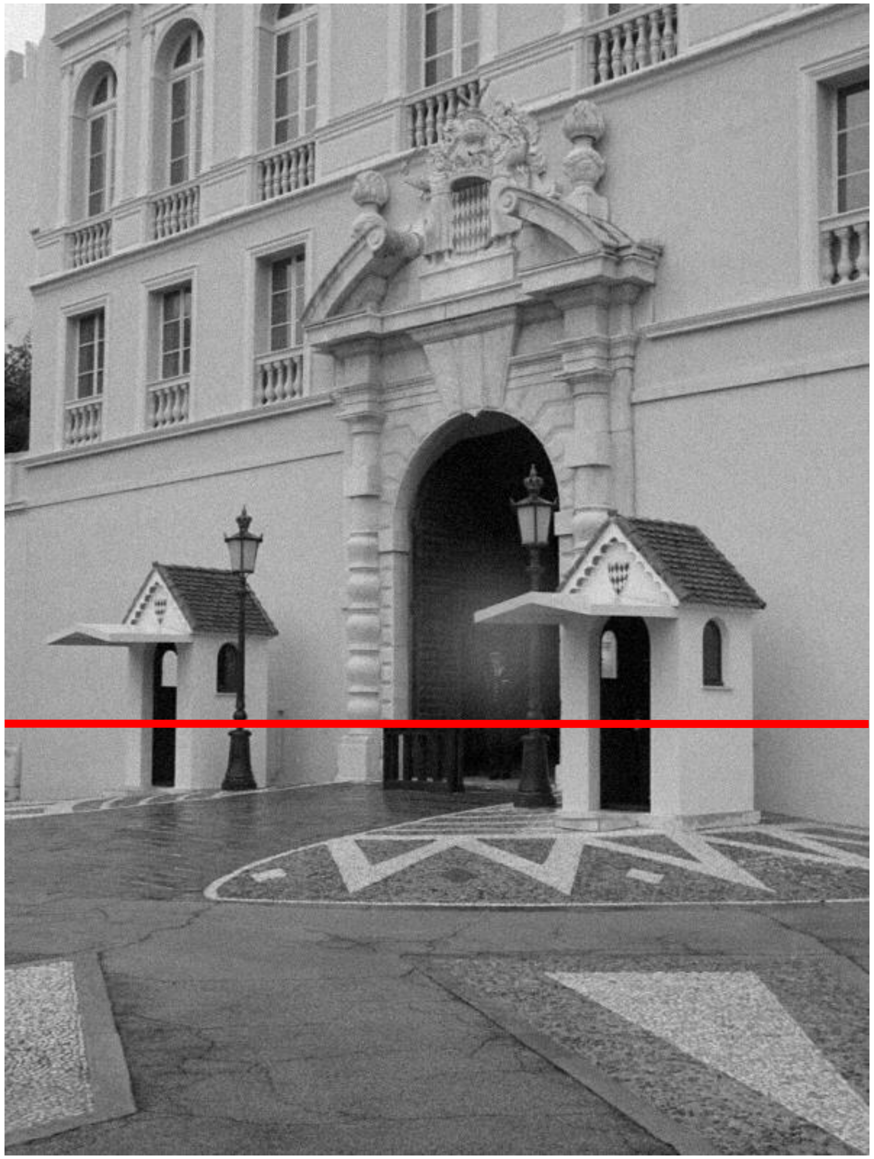
Intensity profile

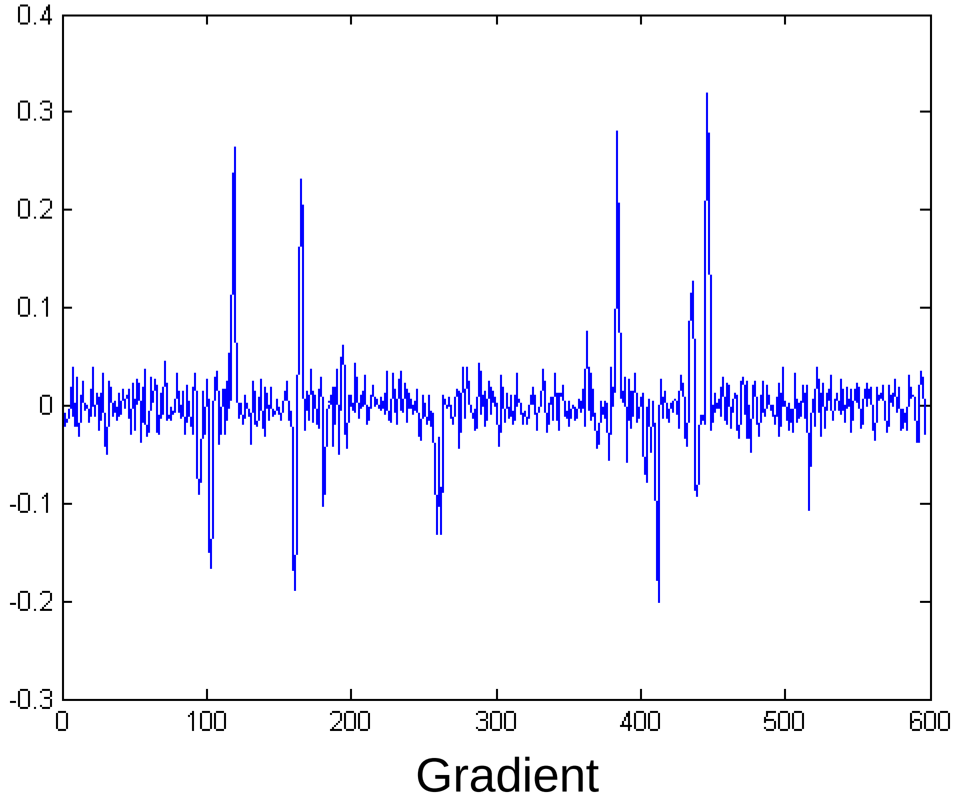






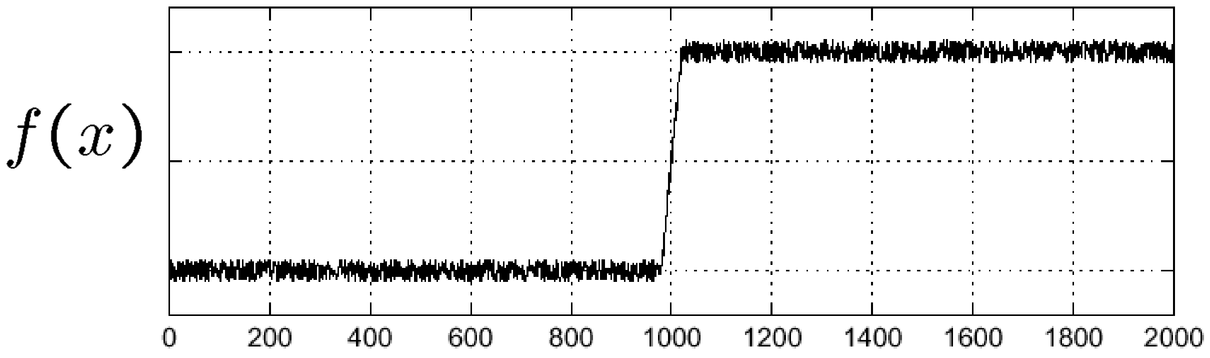
With a little Gaussian noise

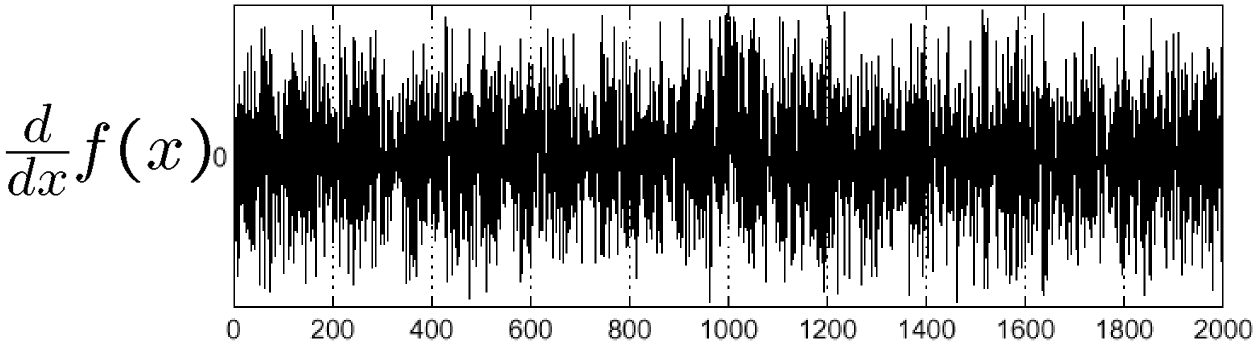




Effects of noise

- 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

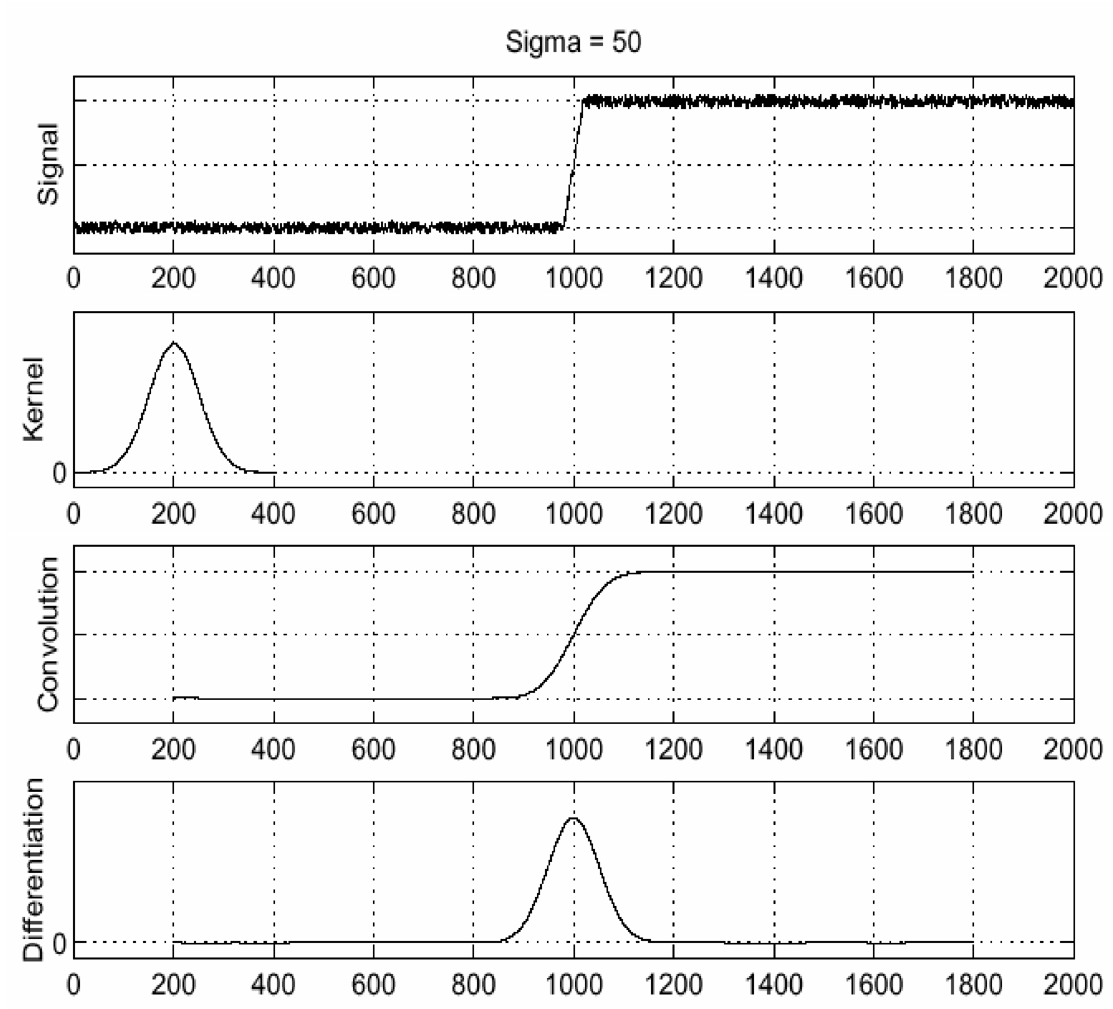
Solution: smooth first

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

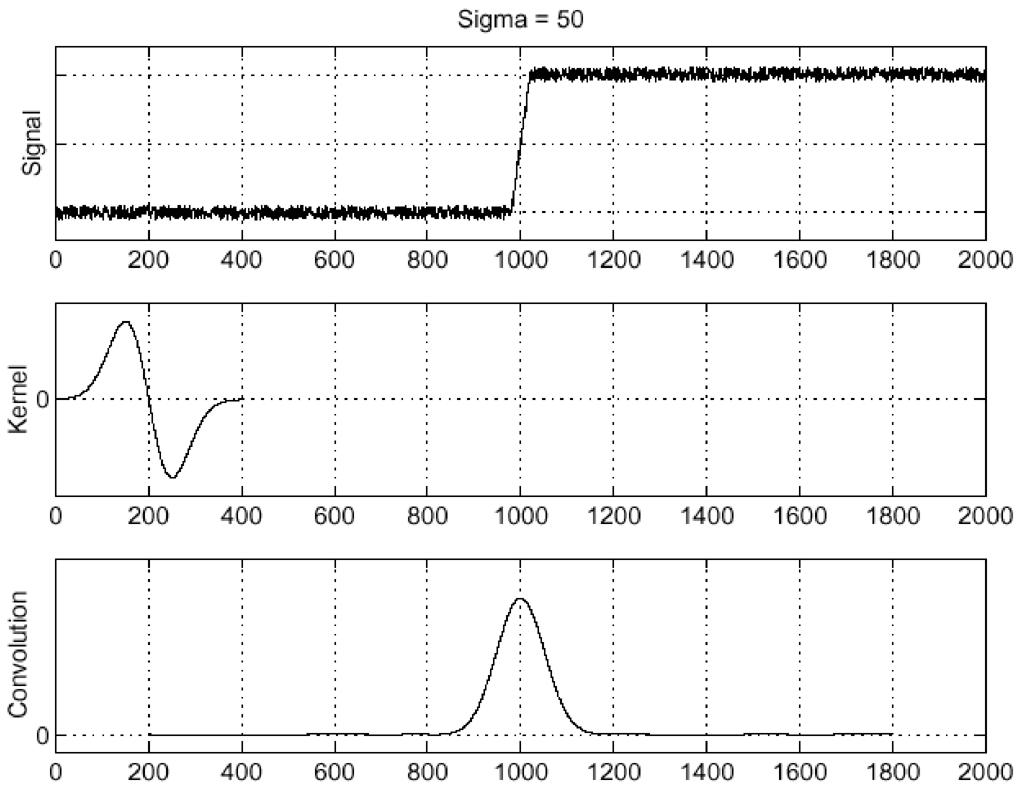
- Differentiation is convolution, and convolution is associative: $$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$$

- This saves us one operation:

f

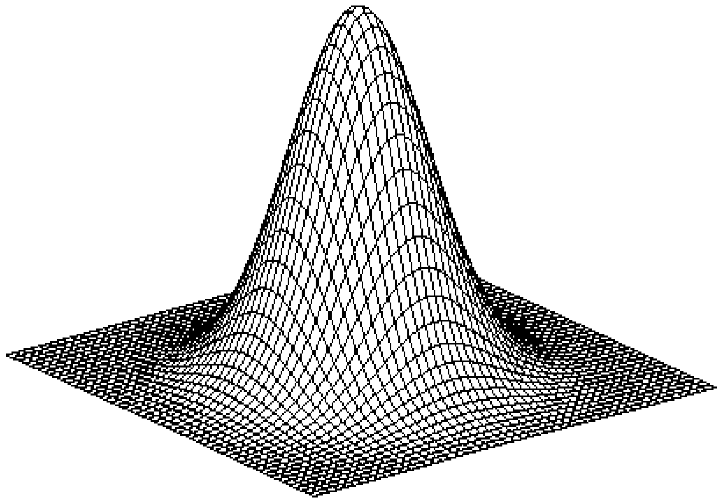
$\frac{d}{dx} g$

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

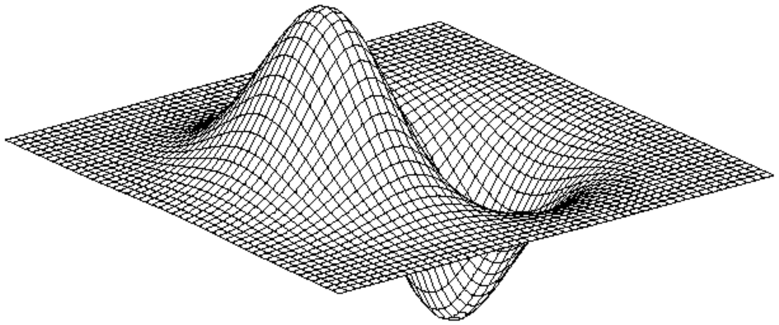


Source: S. Seitz

Derivative of Gaussian filter

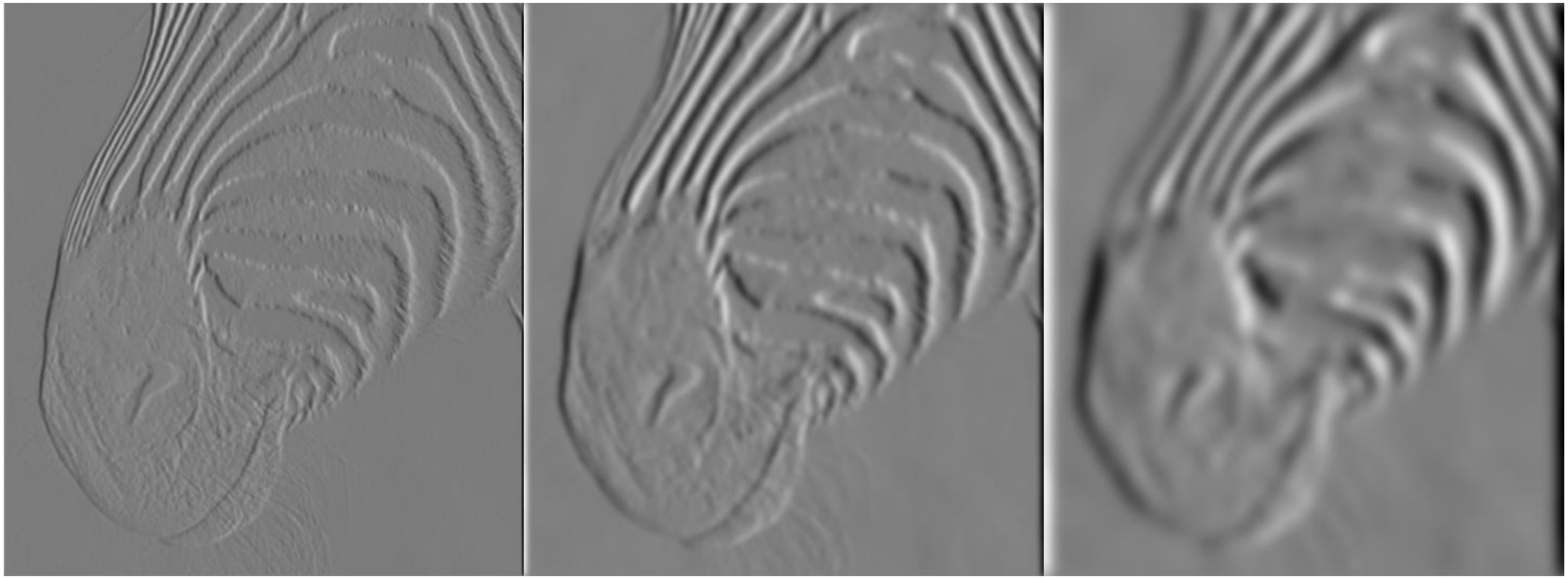


* [1 0 -1] =



- Is this filter separable?

Tradeoff between smoothing and localization



- 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:** find all real edges, ignoring noise or other artifacts
  - **Good localization**
    - detect edges as close as possible to the true edges
    - 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

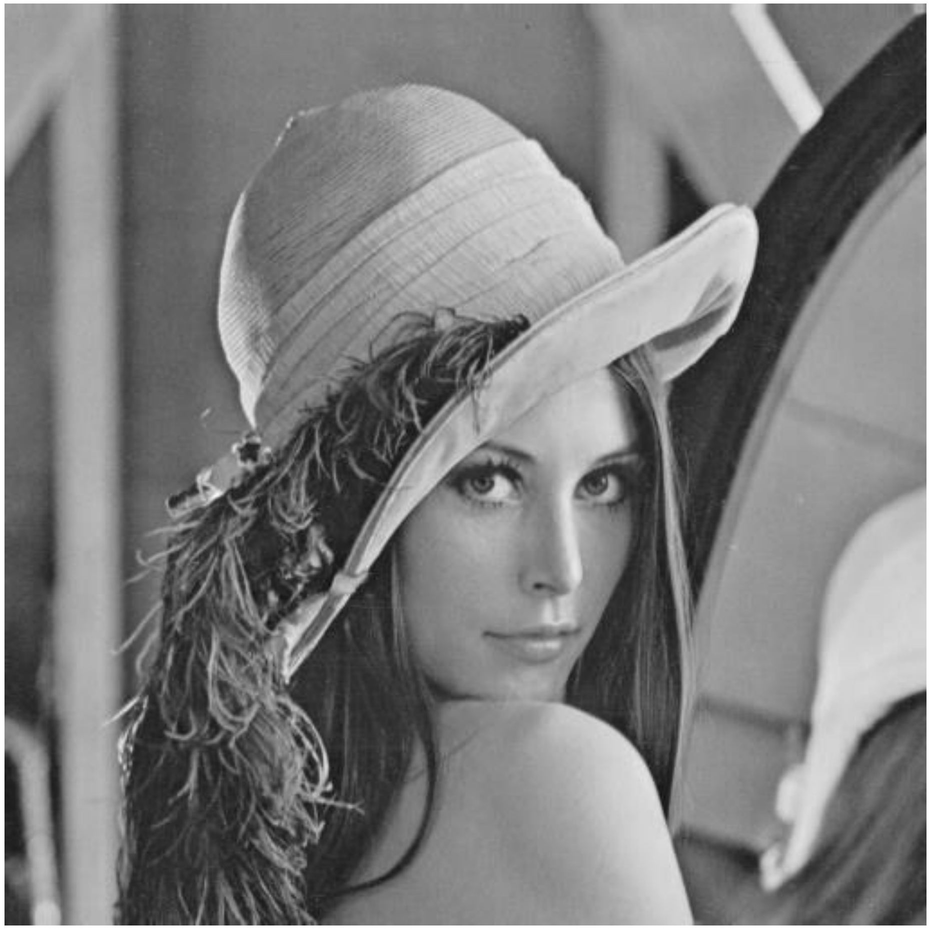
Canny edge detector

- 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.

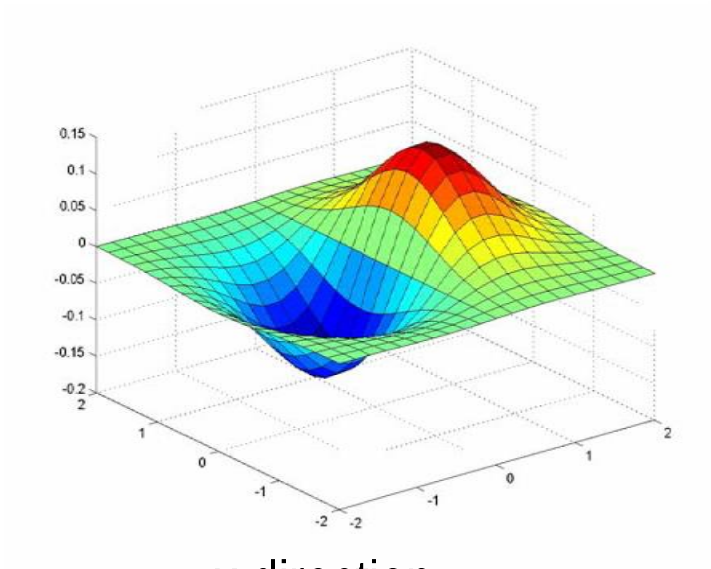
Source: L. Fei-Fei

Example

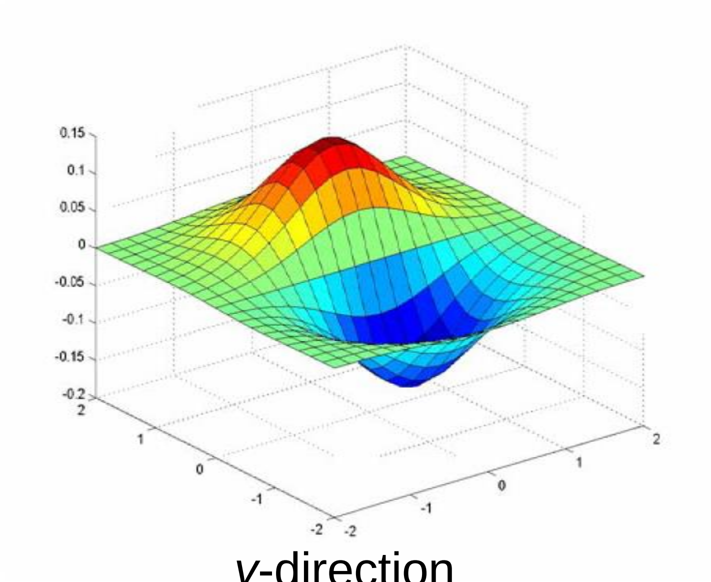


input image (“Lena”)

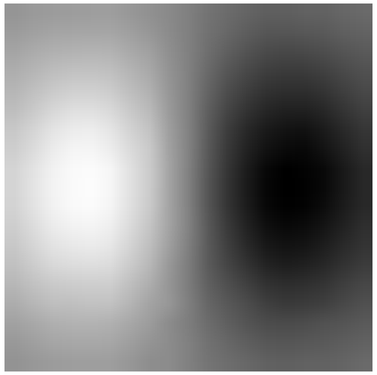
Derivative of Gaussian filter

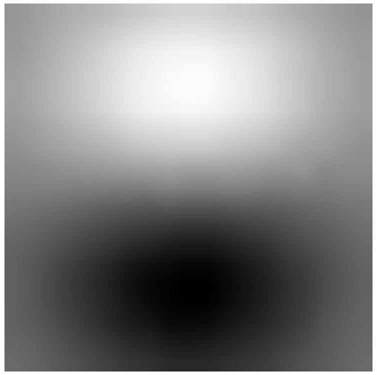


x-direction

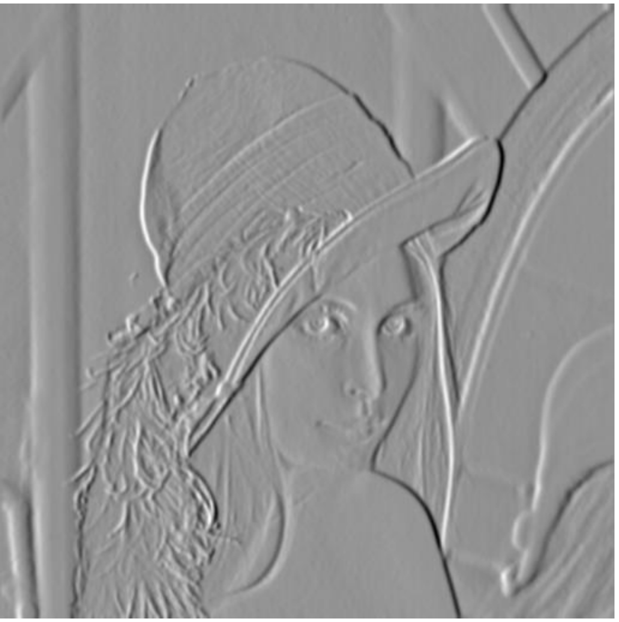


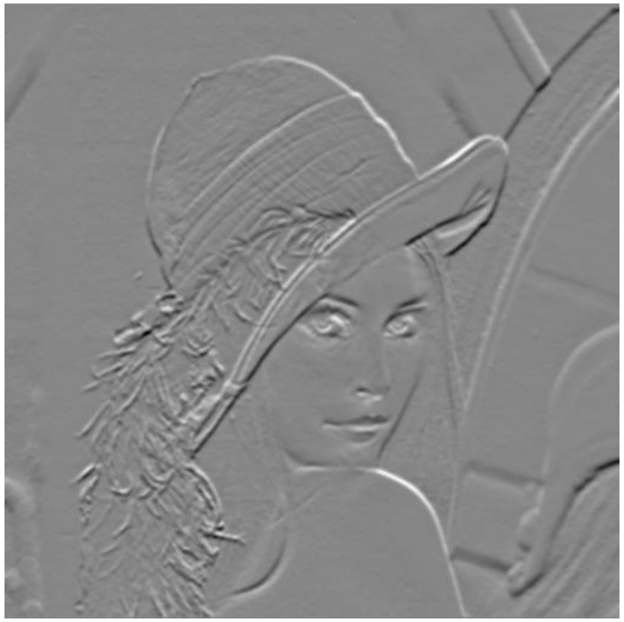
y-direction

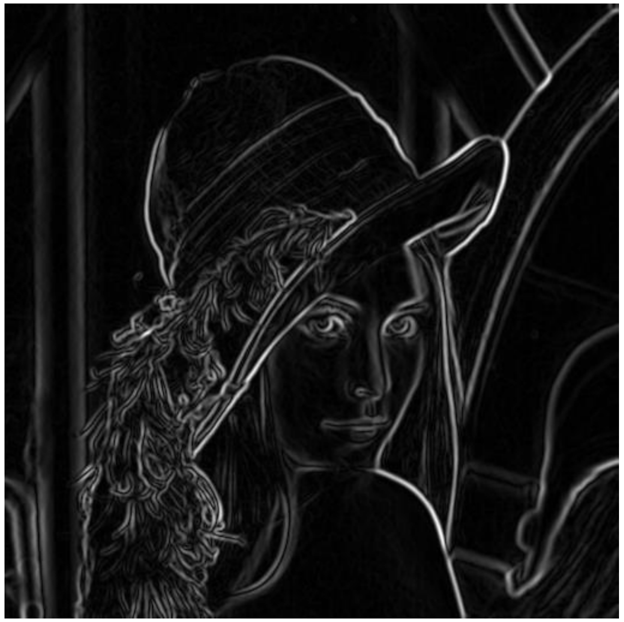




Compute Gradients (DoG)

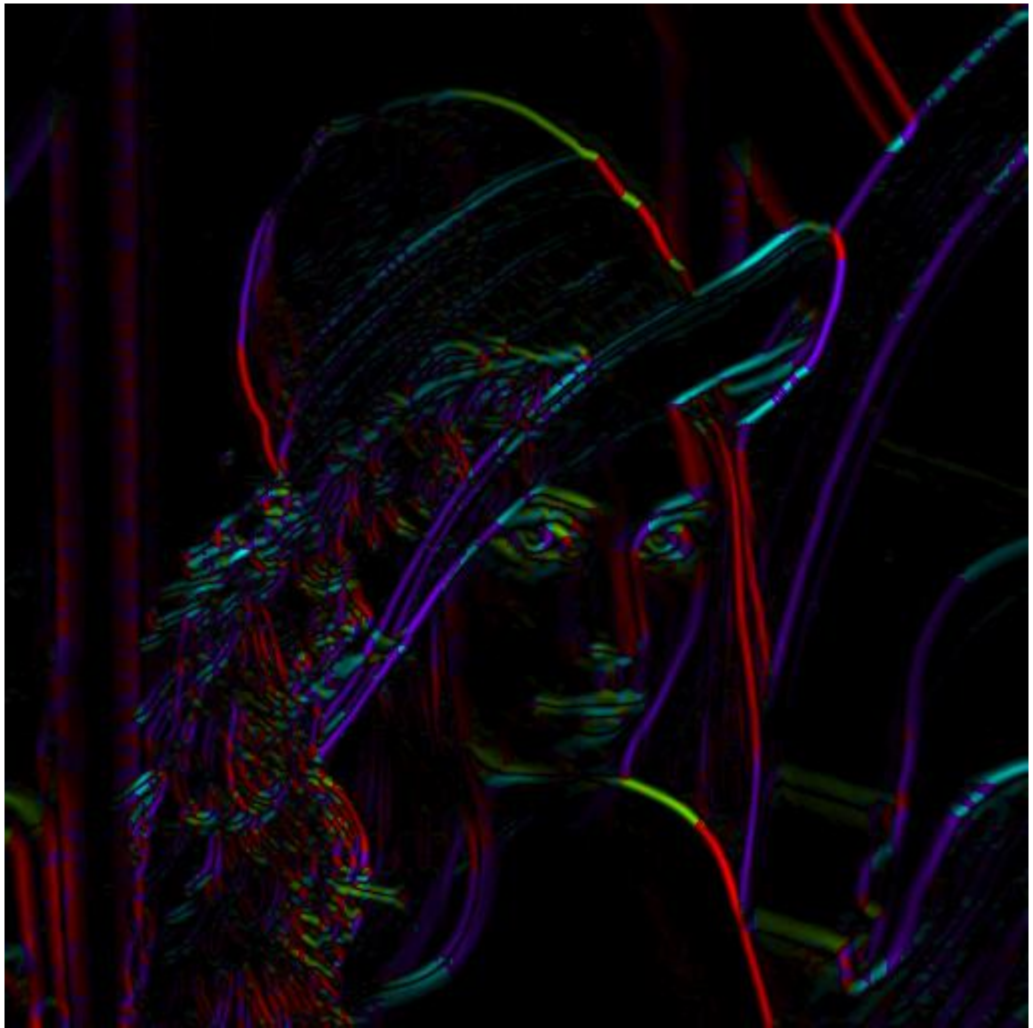

X-Derivative of Gaussian


Y-Derivative of Gaussian


Gradient Magnitude

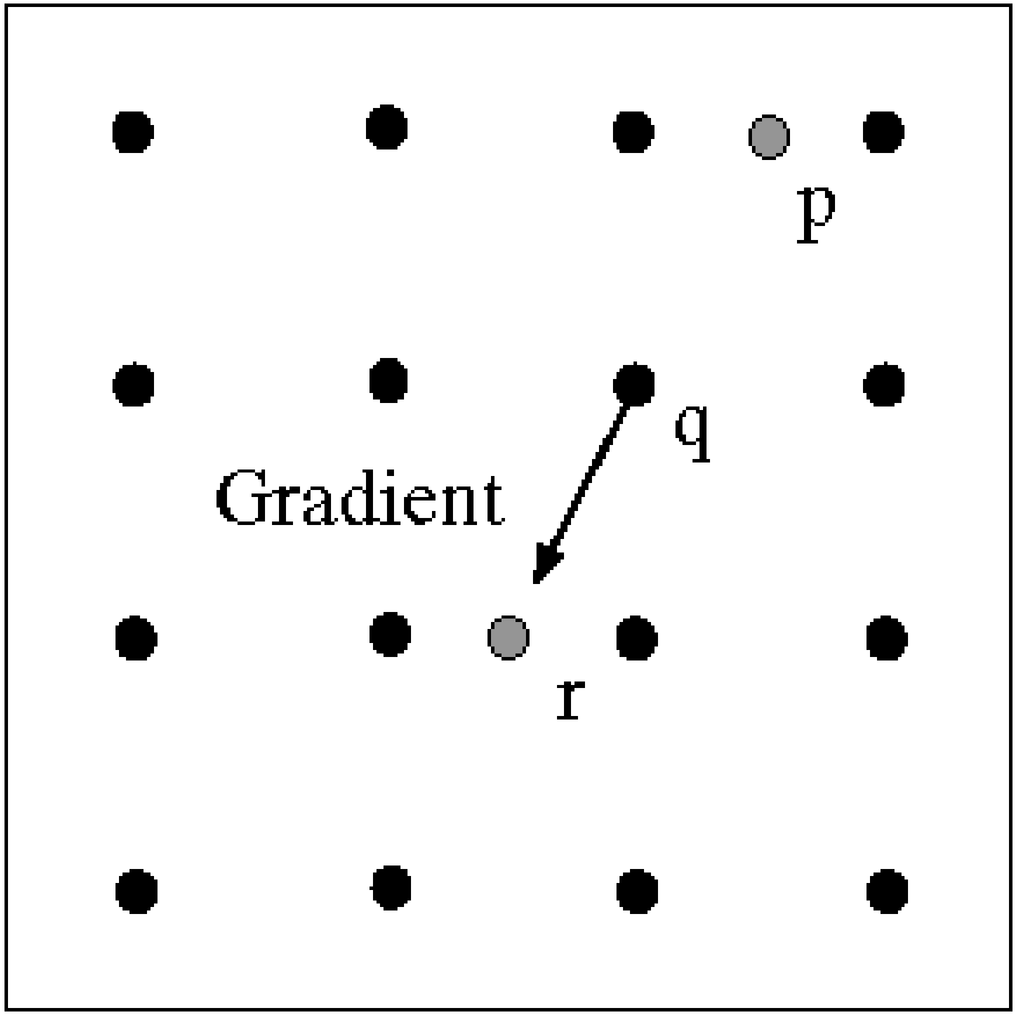
Get Orientation at Each Pixel

- Threshold at minimum level
- Get orientation

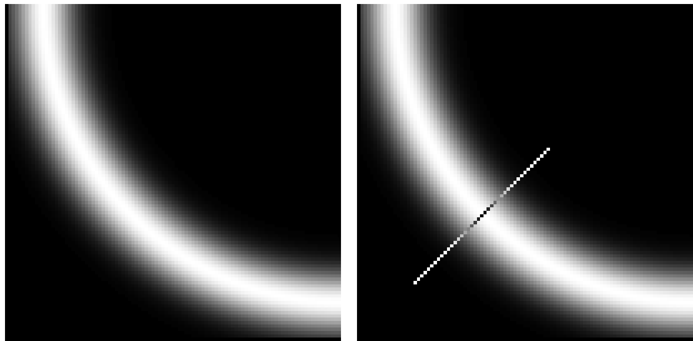


theta = atan2(-gy, gx)

Non-maximum suppression for each orientation



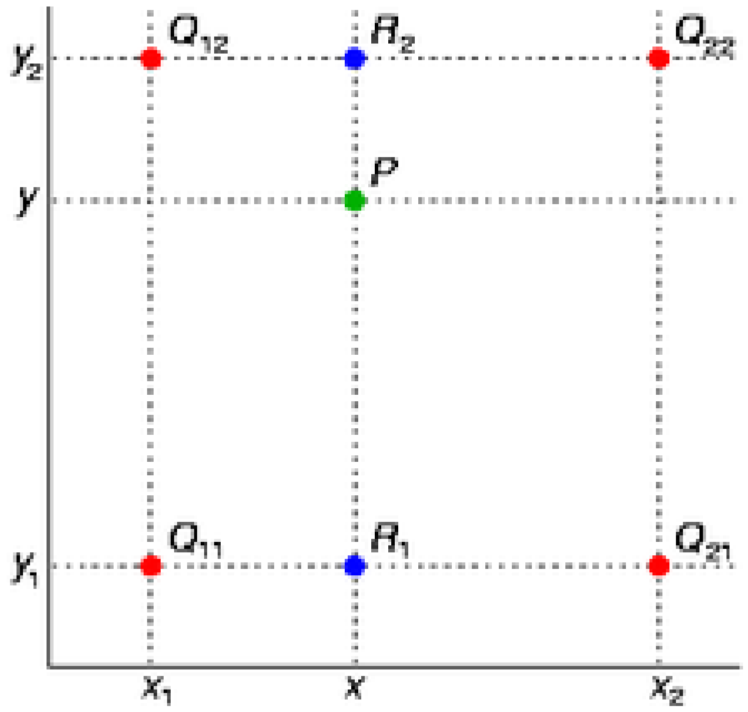
At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.

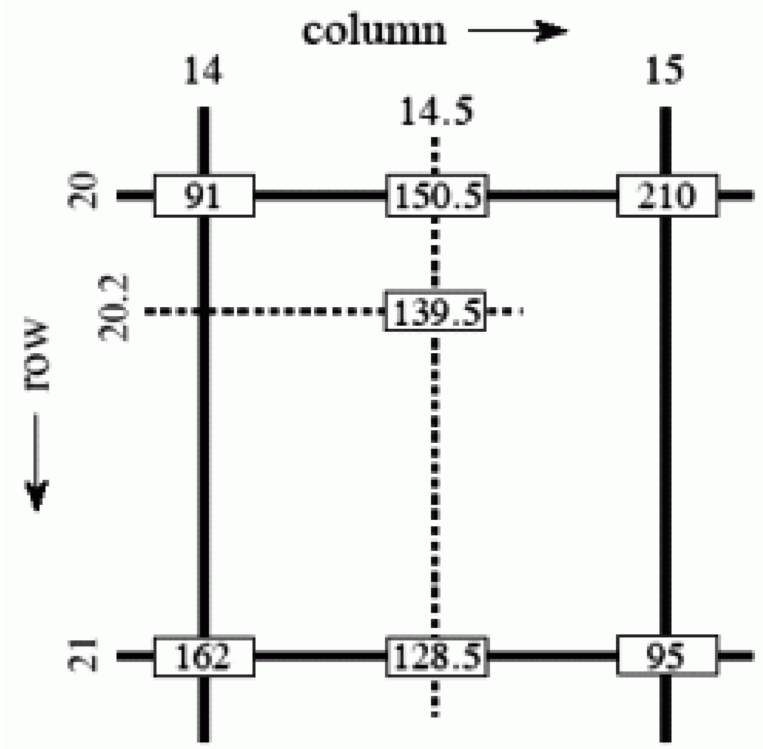


Source: D. Forsyth

Bilinear Interpolation

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



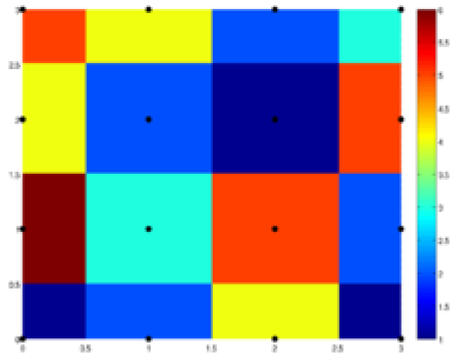


http://en.wikipedia.org/wiki/Bilinear_interpolation

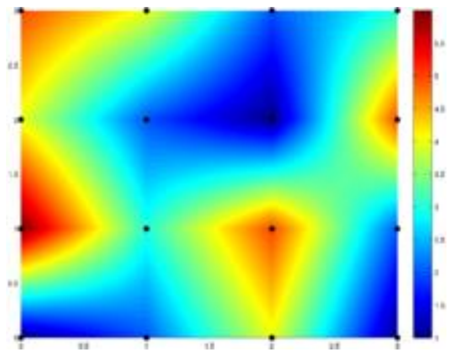
Sidebar: Interpolation options

- imx2 = imresize(im, 2, interpolation_type)

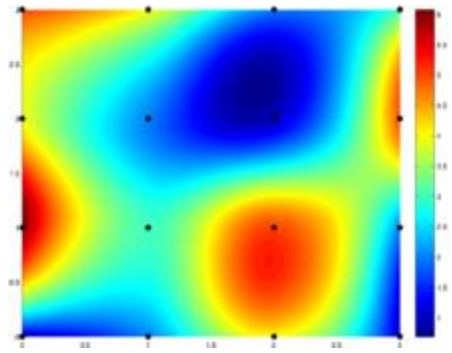
- 'nearest'
  - Copy value from nearest known
  - Very fast but creates blocky edges



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

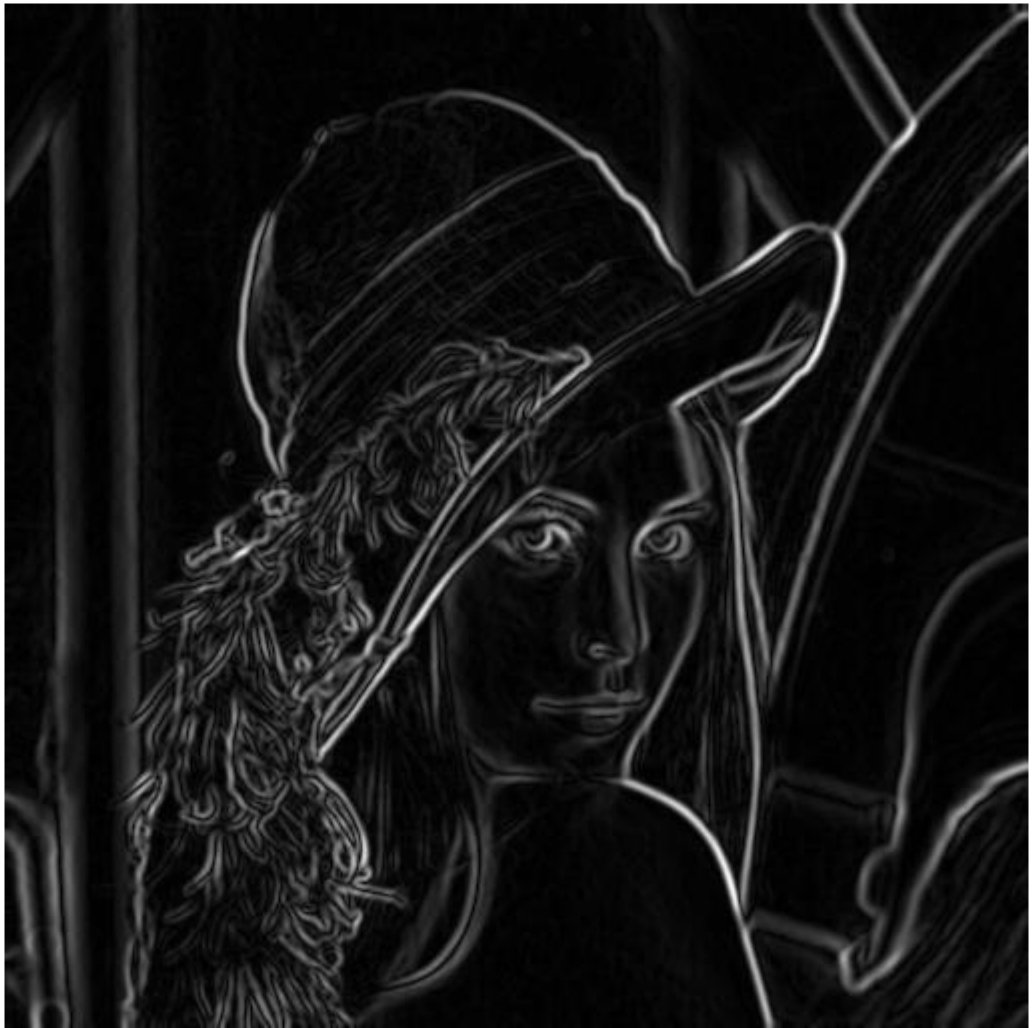


- 'bicubic' (default)
  - Non-linear smoothing over larger area
  - Slower, visually appealing, may create negative pixel values

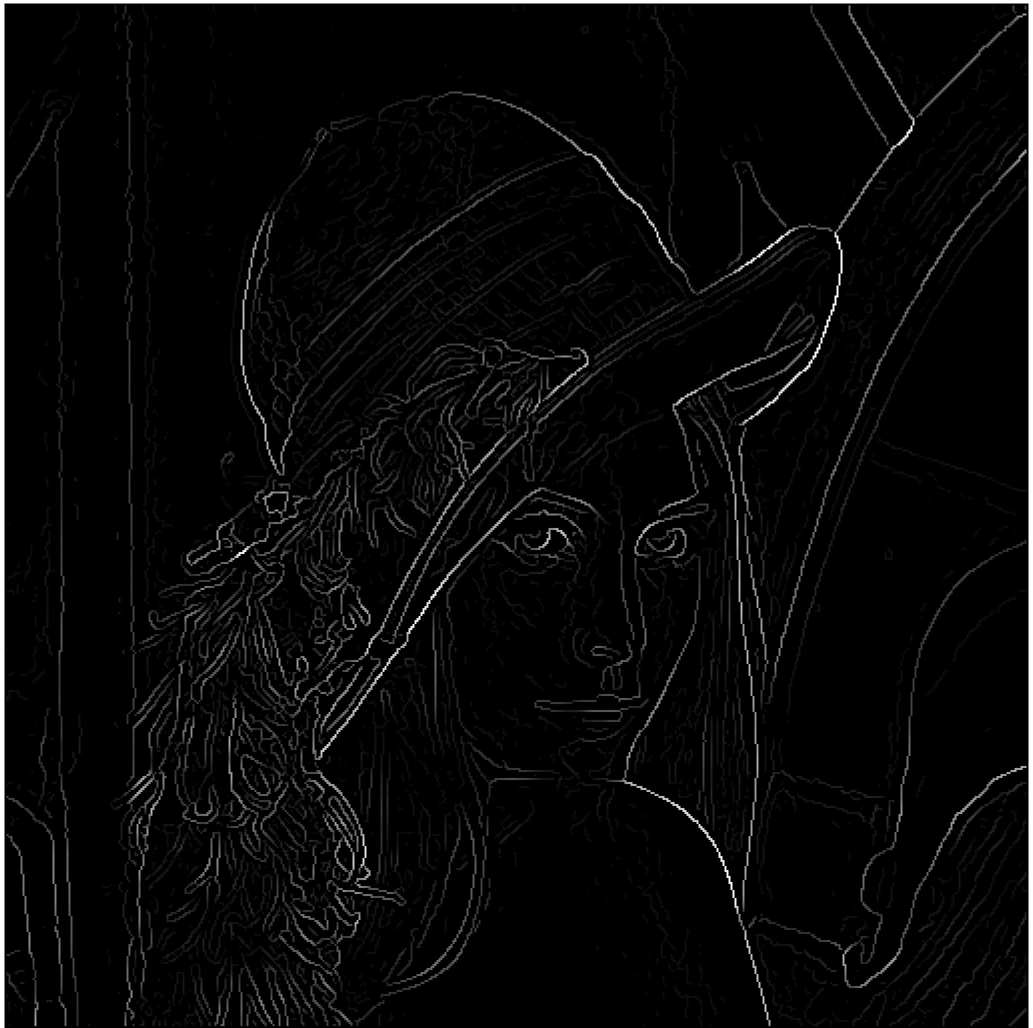


Examples from http://en.wikipedia.org/wiki/Bicubic_interpolation

Before Non-max Suppression

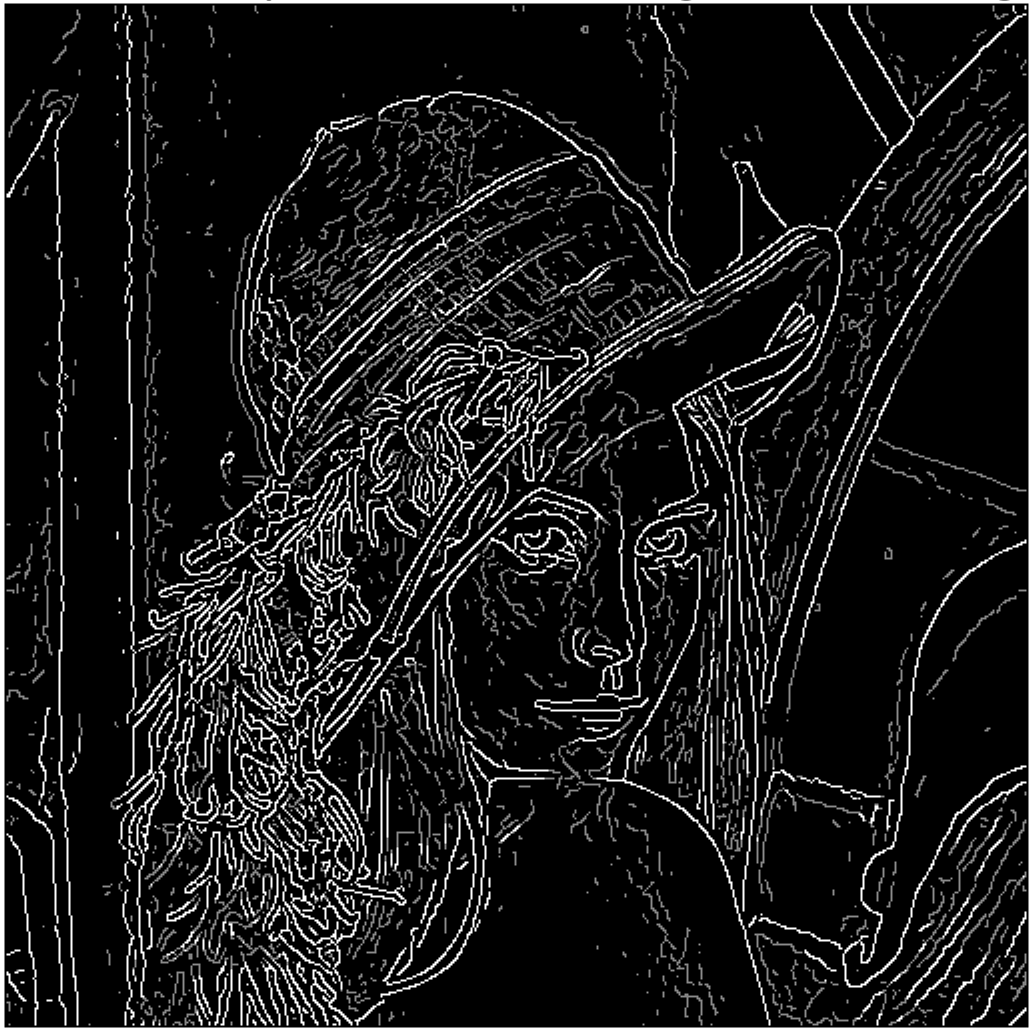


After non-max suppression



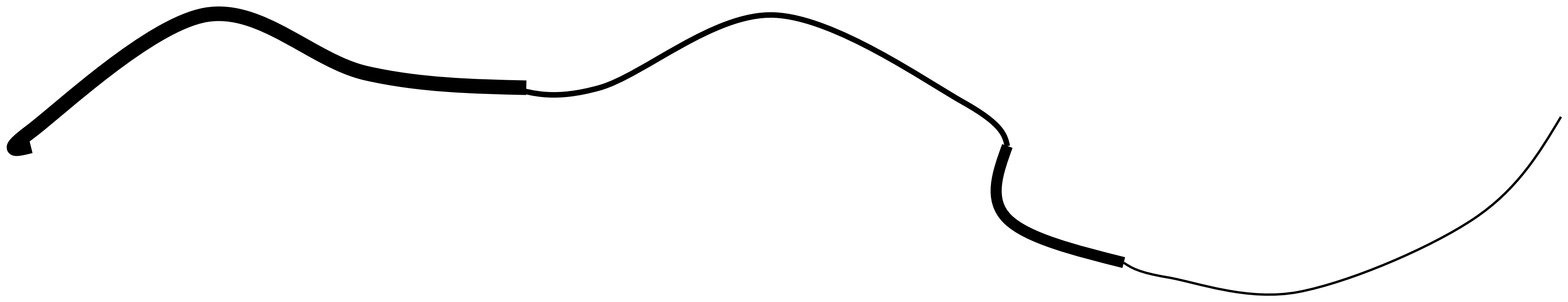
Hysteresis thresholding

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



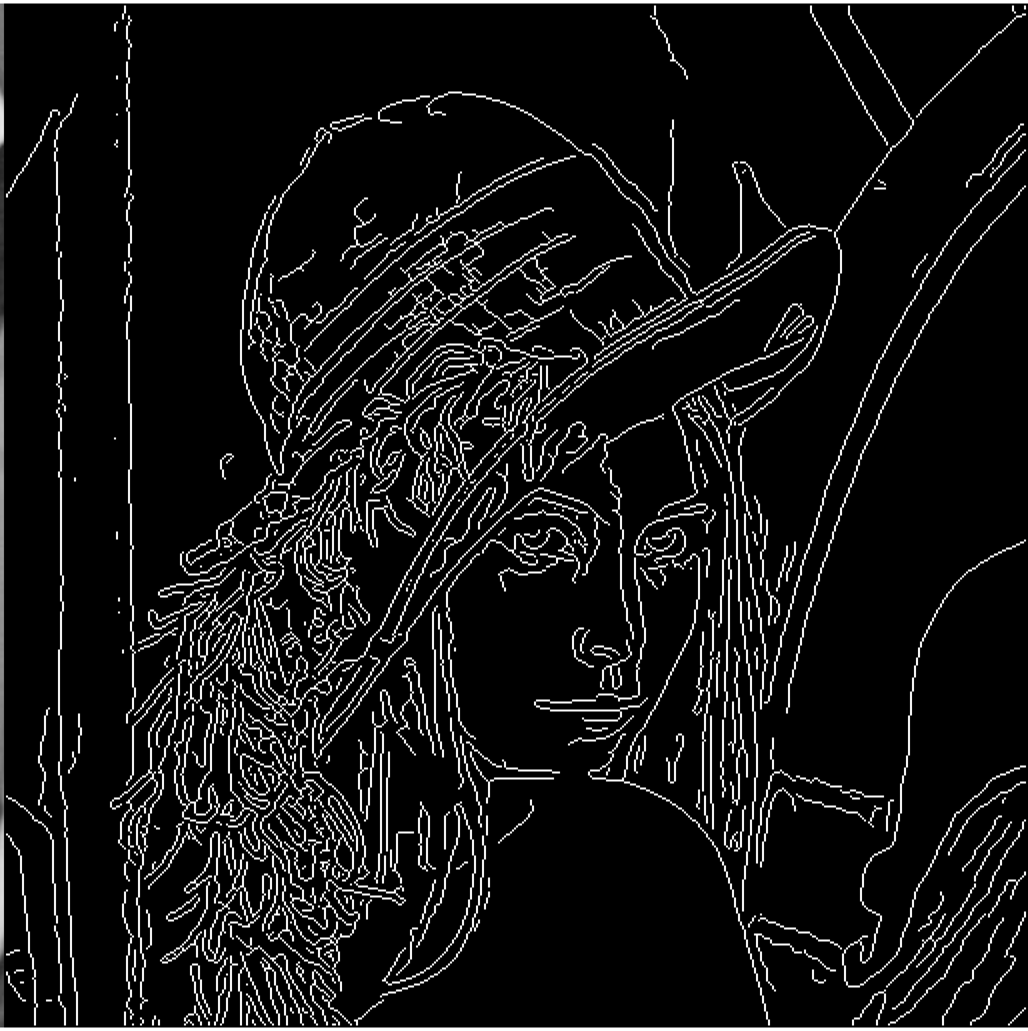
Hysteresis thresholding

- 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



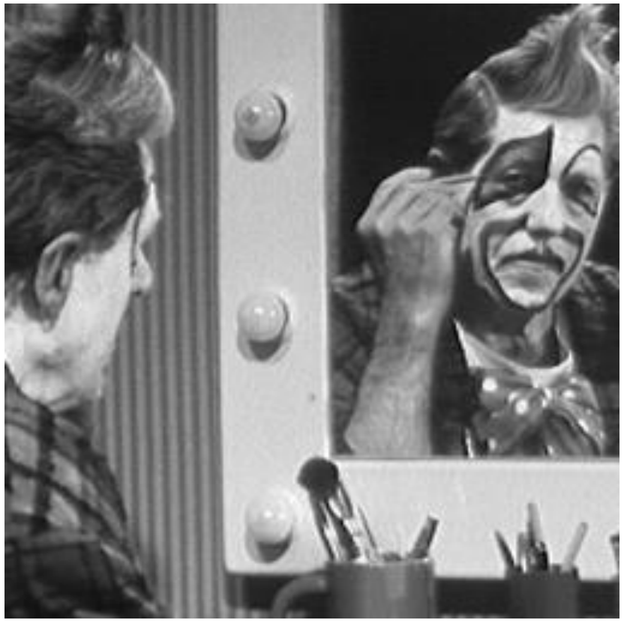
Canny edge detector

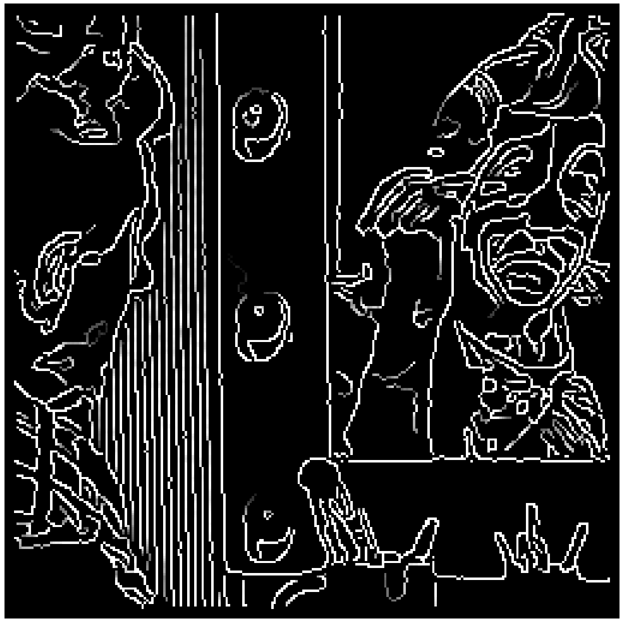
1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
   – Thin multi-pixel wide “ridges” down to single pixel width
4. Thresholding and linking (hysteresis):
   – Define two thresholds: low and high
   – Use the high threshold to start edge curves and the low threshold to continue them

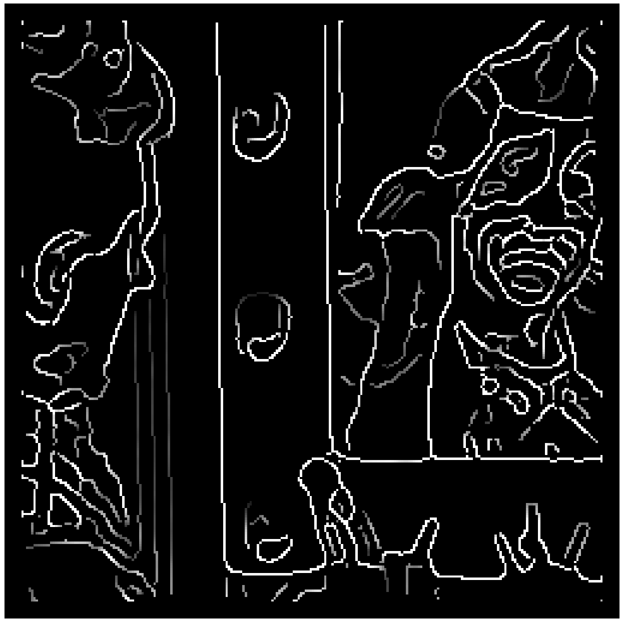
- MATLAB: edge(image, ‘canny’)

Source: D. Lowe, L. Fei-Fei

Effect of σ (Gaussian kernel spread/size)


original


Canny with $\sigma = 1$

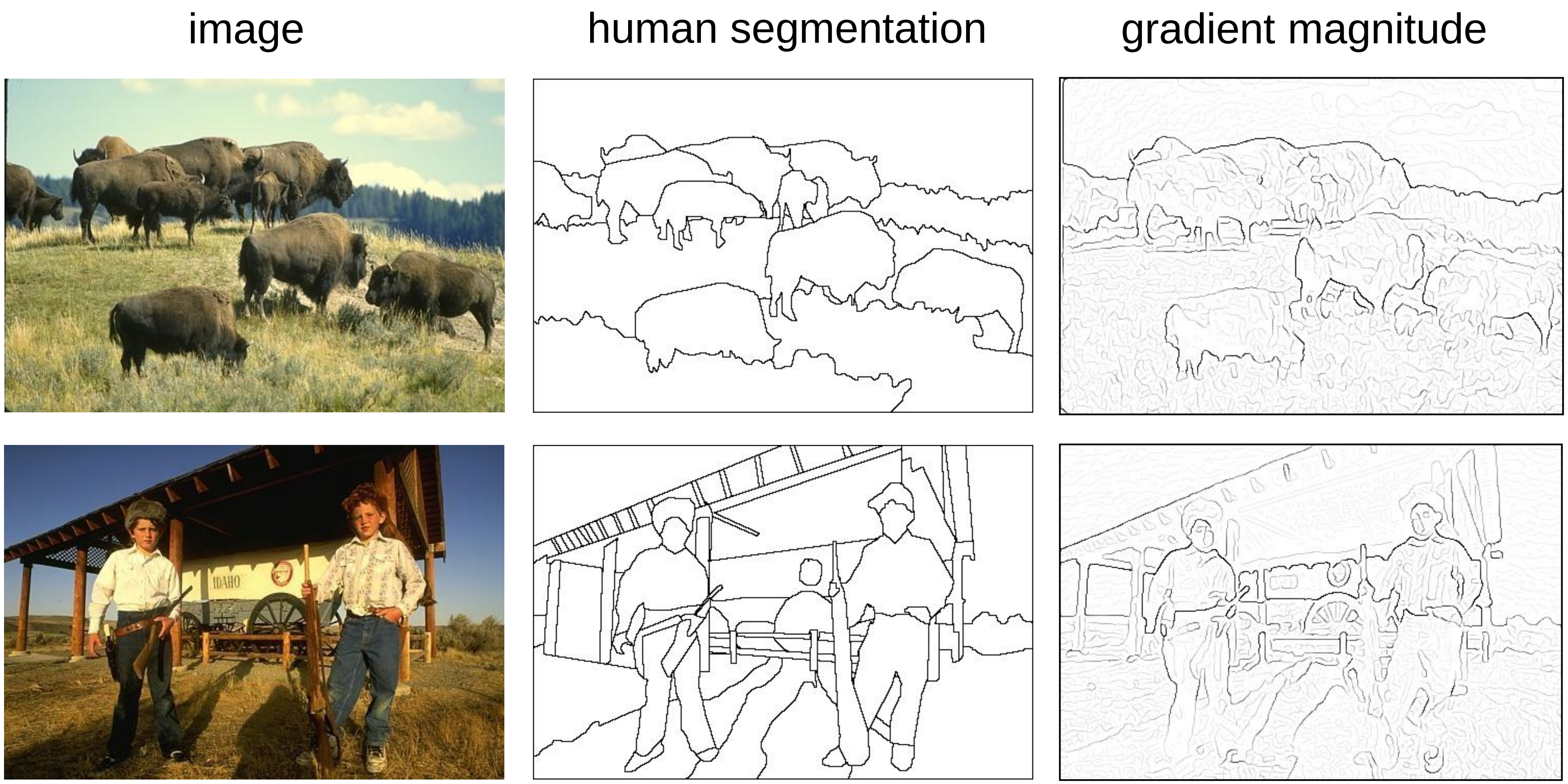

Canny with $\sigma = 2$

The choice of σ depends on desired behavior

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

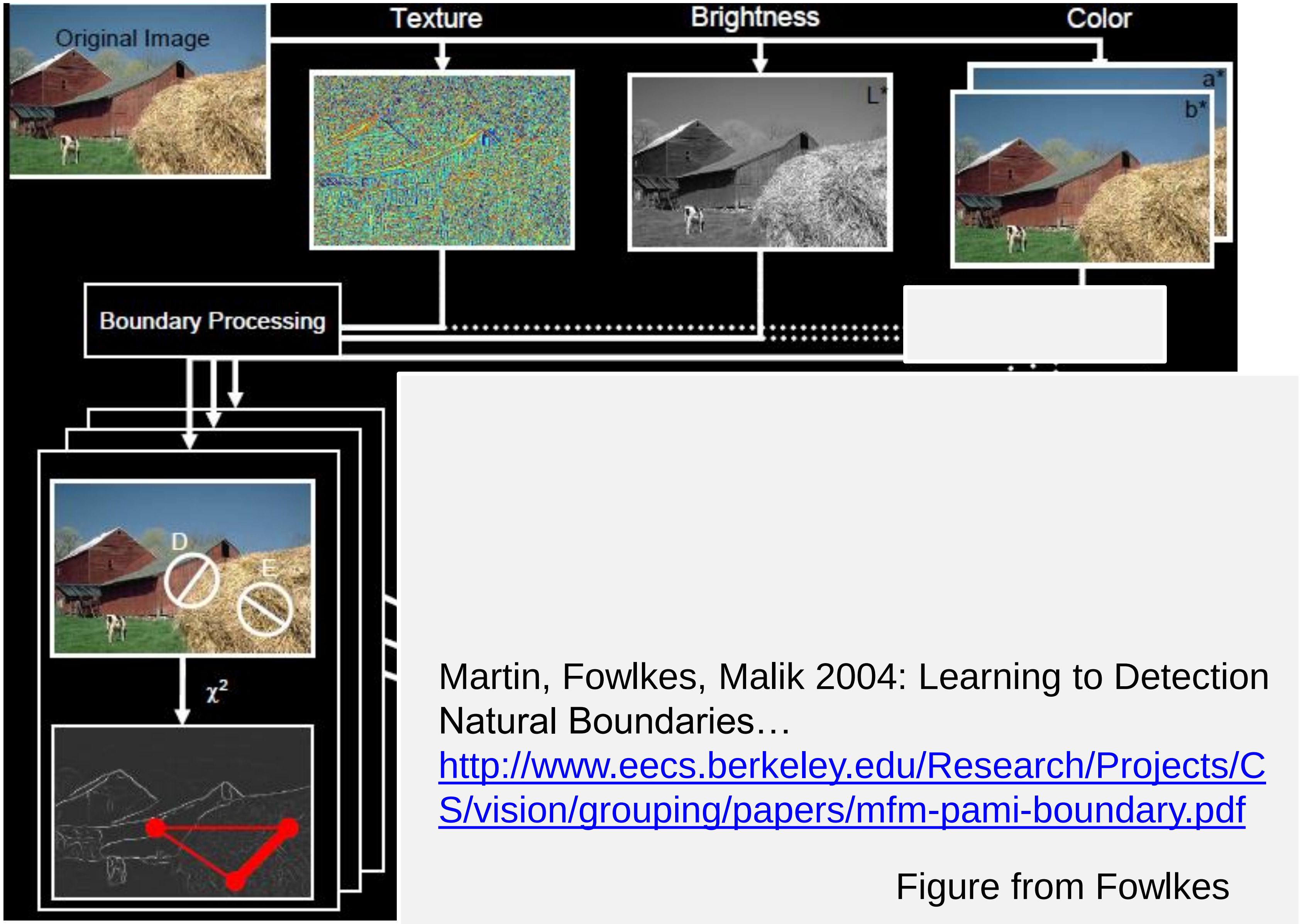
Source: S. Seitz

Learning to detect boundaries



- 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

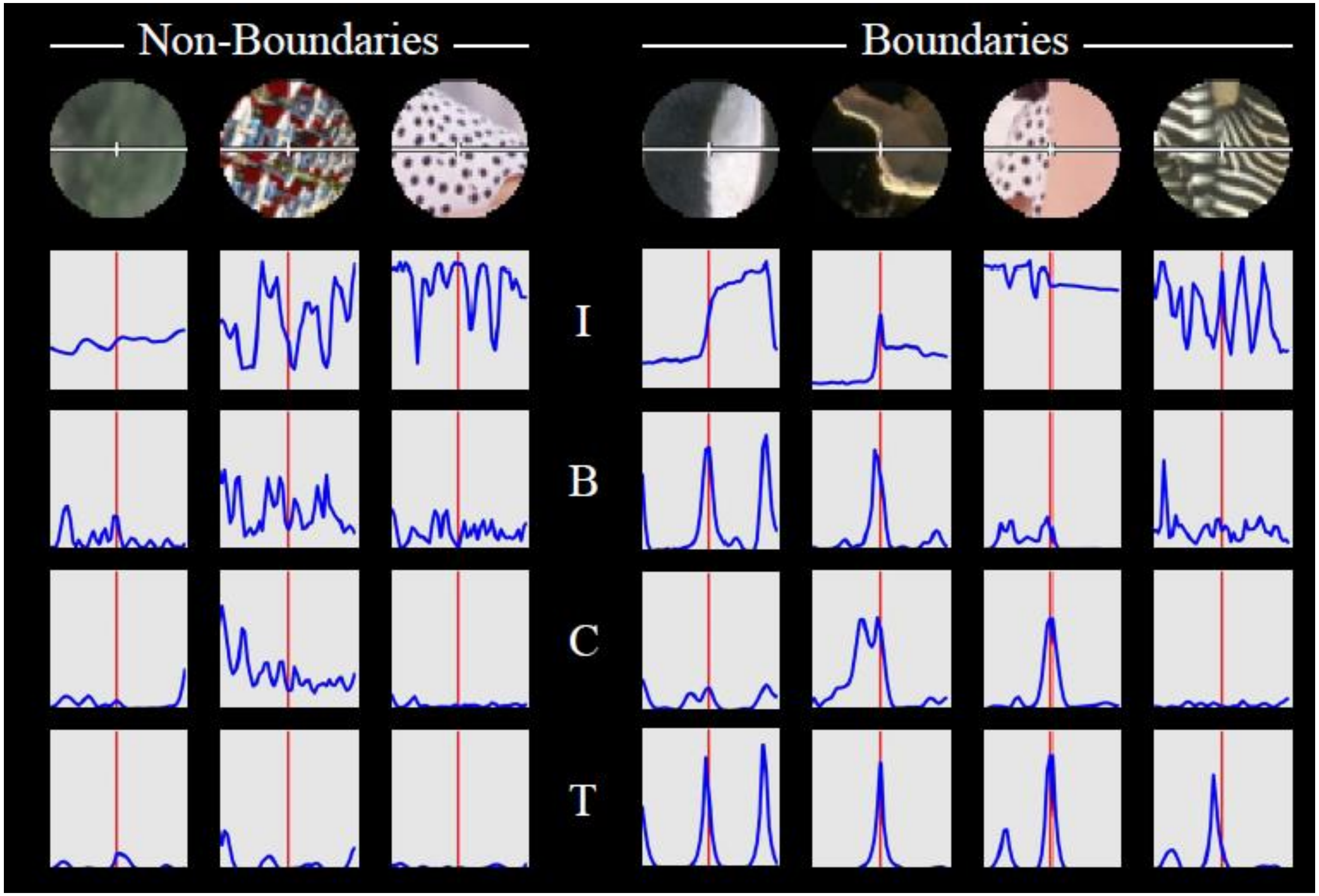


Figure from Fowlkes

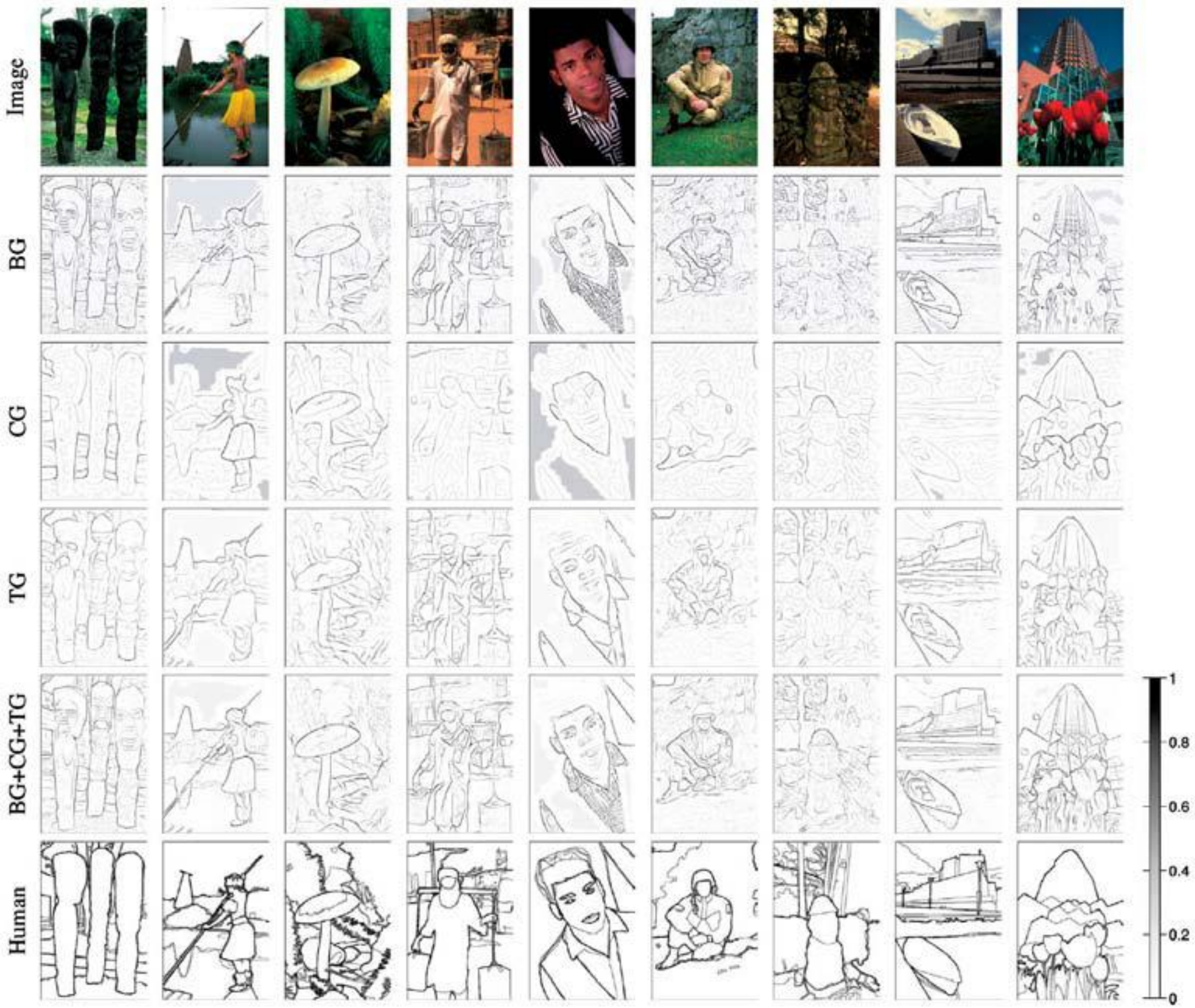
Brightness

Color

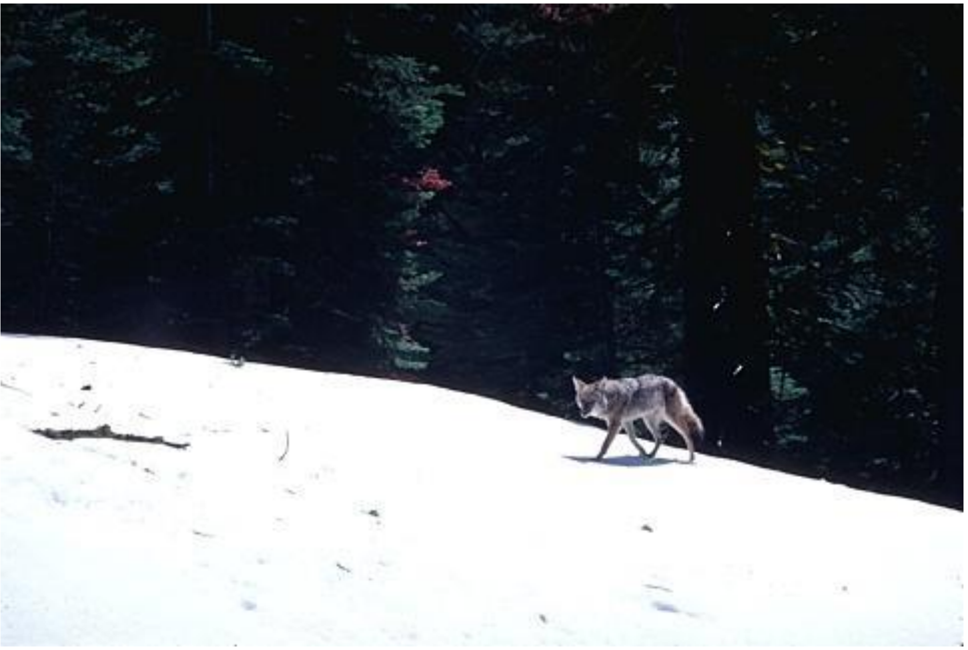
Texture

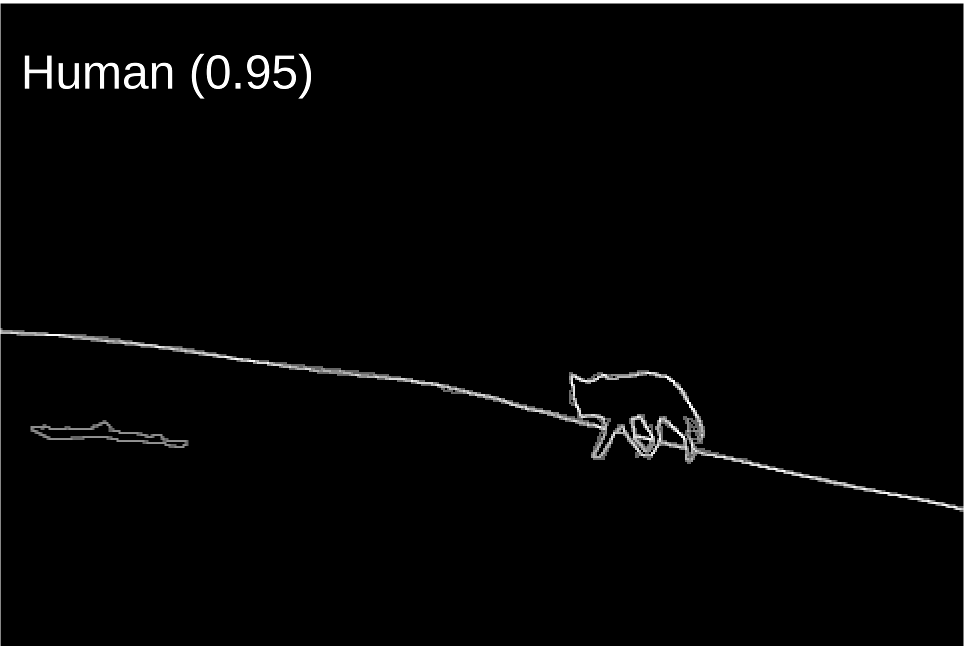
Combined

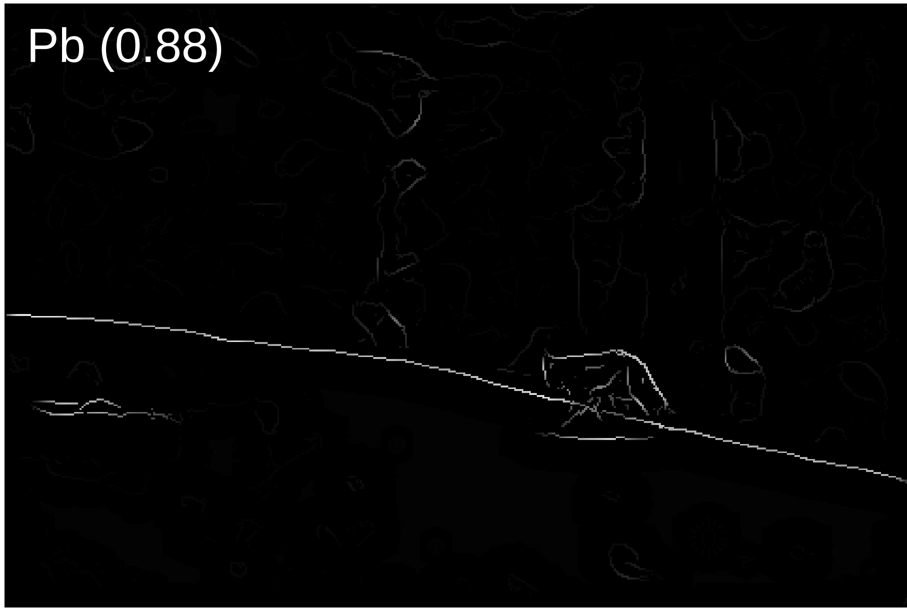
Human



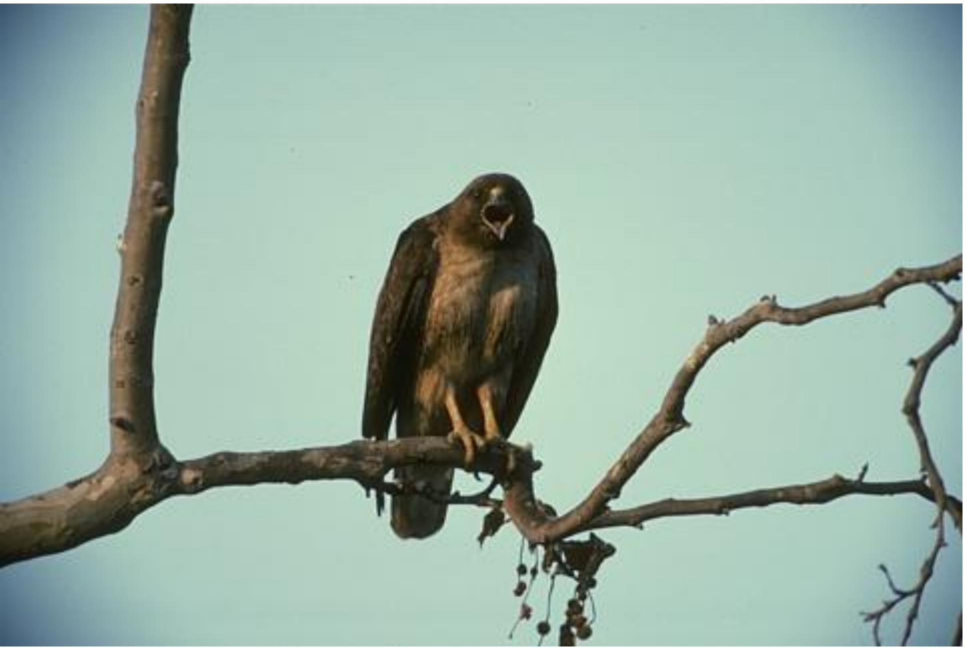
Results

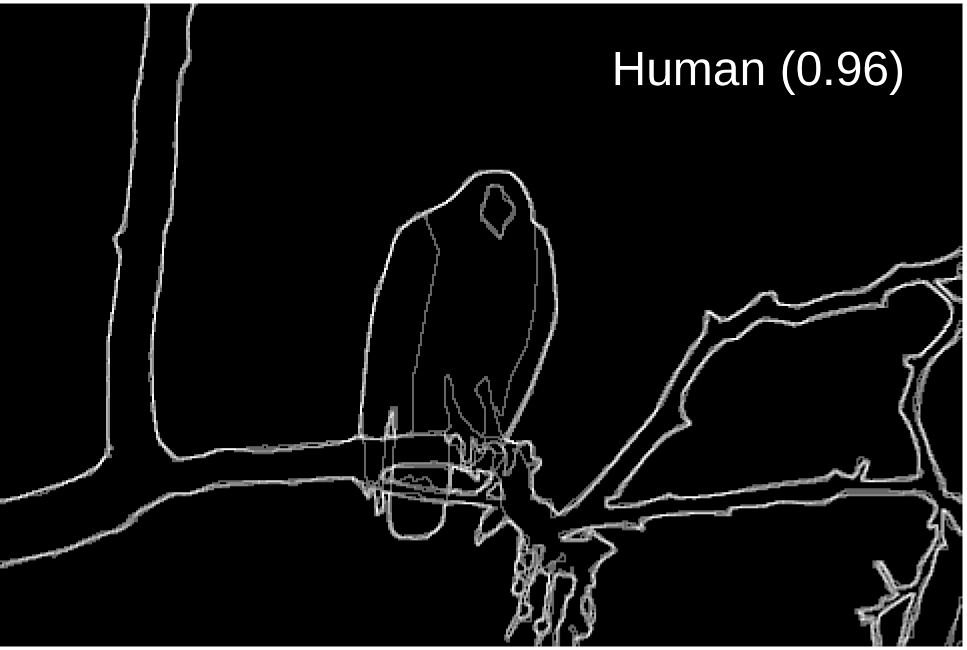


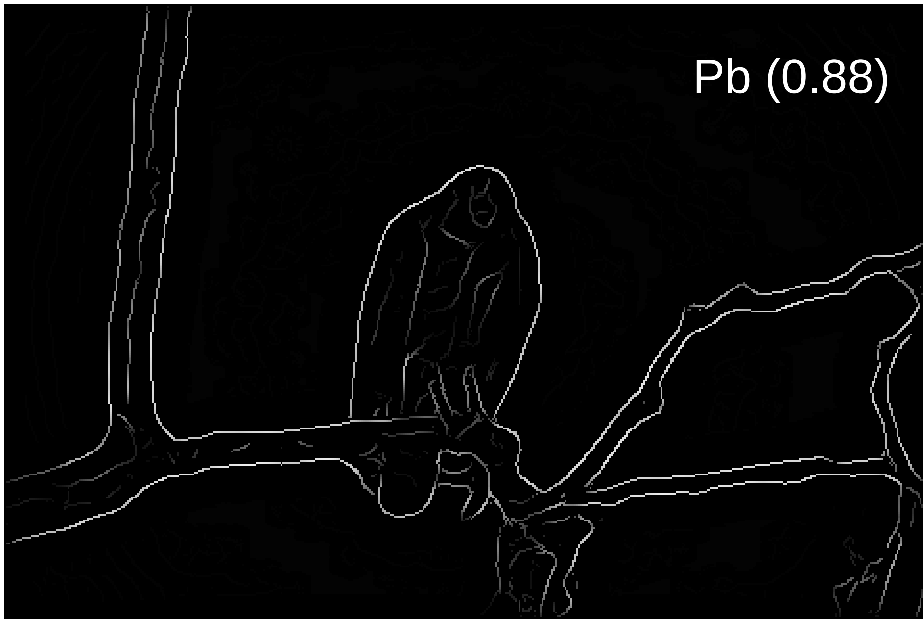


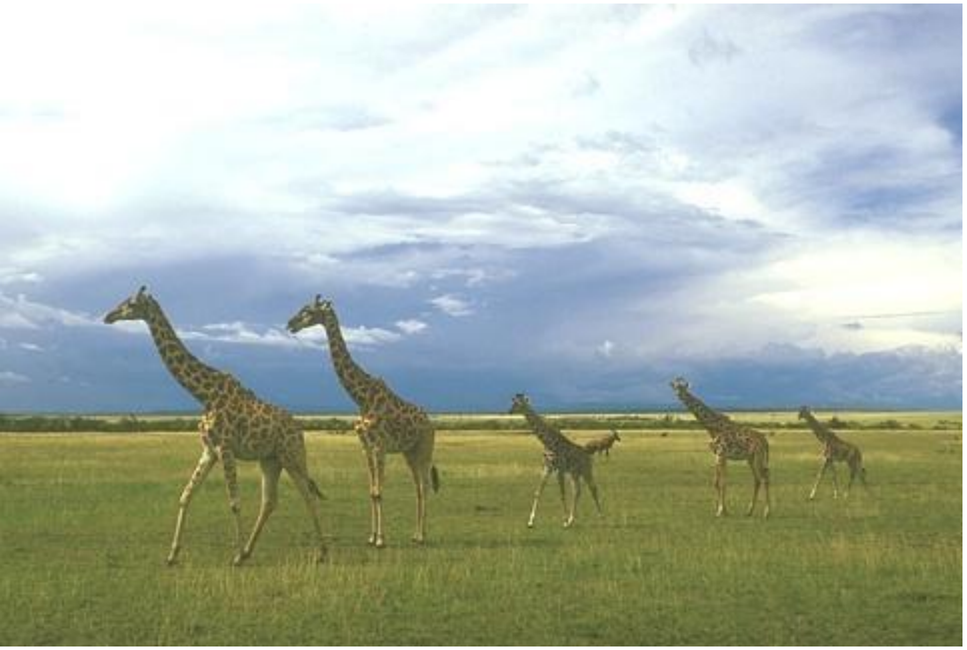


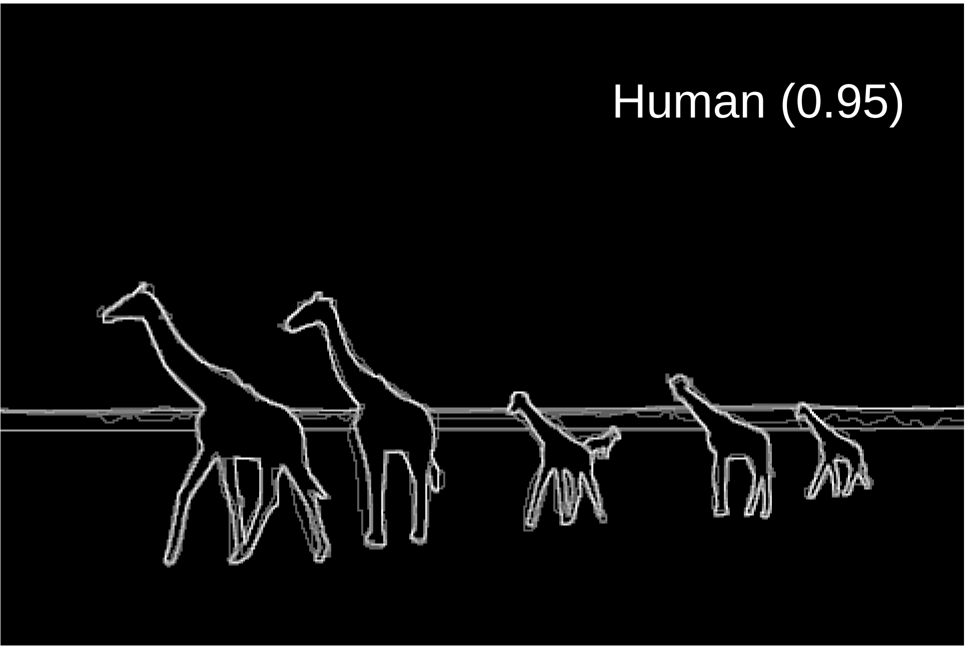
Results

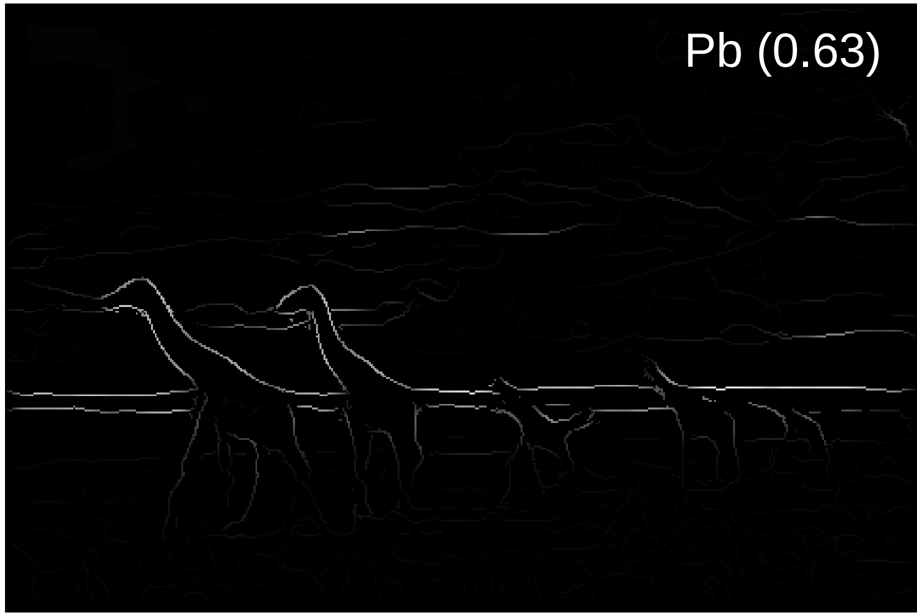


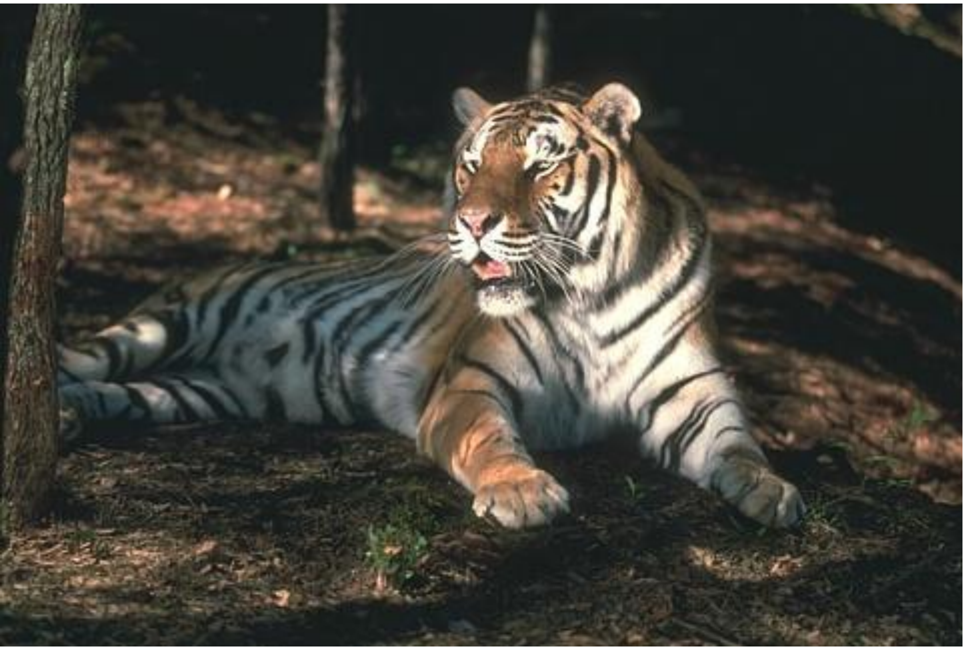


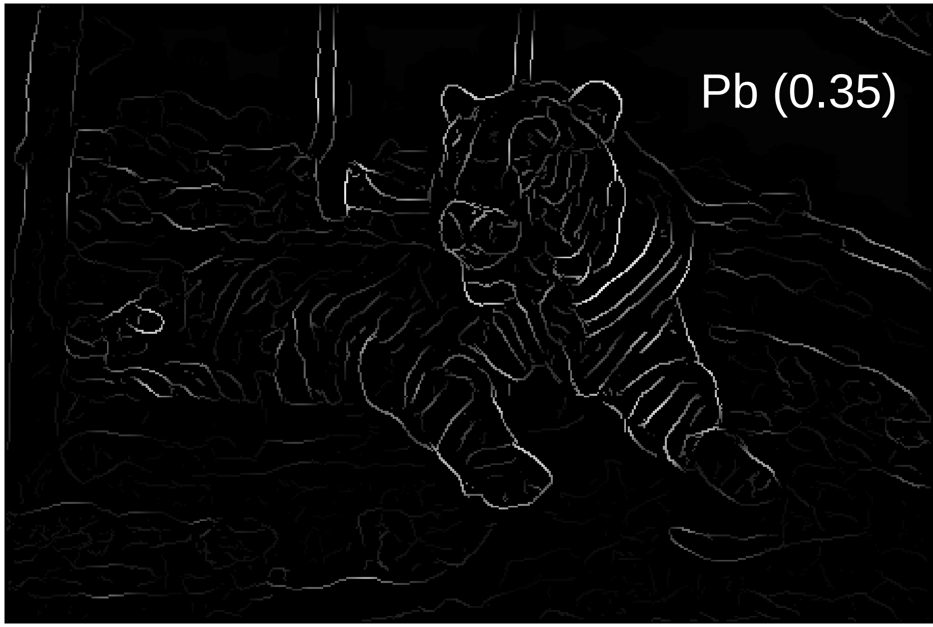


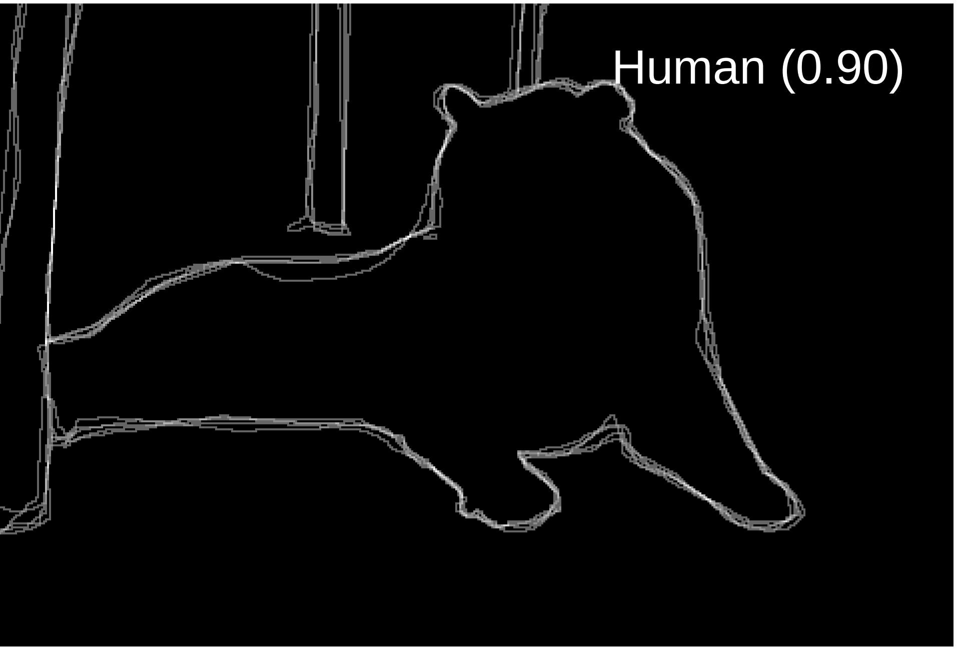



Human (0.95)


Pb (0.63)







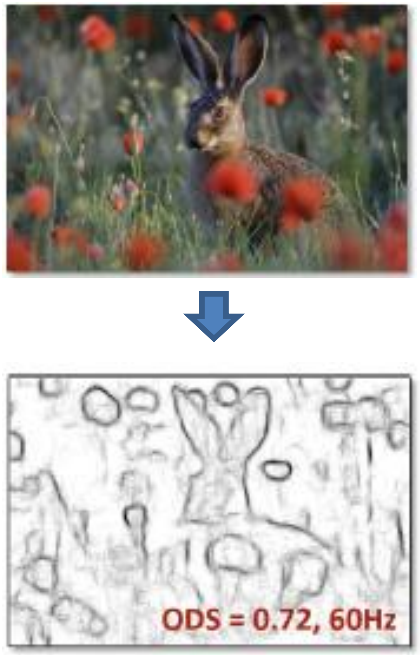
For more:
http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/bench/html/108082-color.html

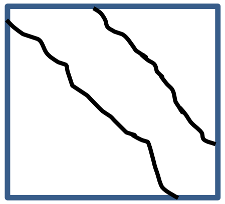
Edge Detection with Structured Random Forests (Dollar Zitnick ICCV 2013)

- Goal: quickly predict whether each pixel is an edge

- Insights
  - Predictions can be learned from training data
  - Predictions for nearby pixels should not be independent

- Solution
  - Train structured random forests to split data into patches with similar boundaries based on features
  - Predict boundaries at patch level, rather than pixel level, and aggregate (average votes)



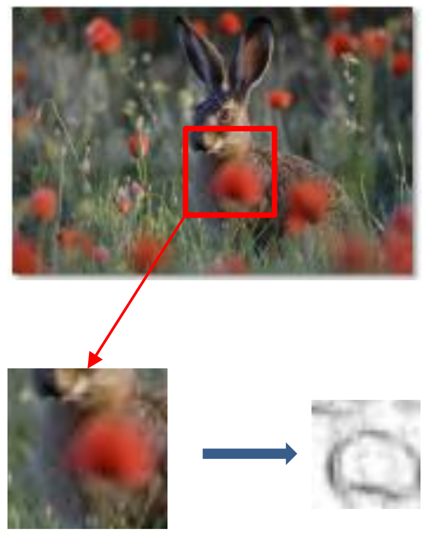


Boundaries in patch

http://research.microsoft.com/pubs/202540/DollarICCV13edges.pdf

Edge Detection with Structured Random Forests

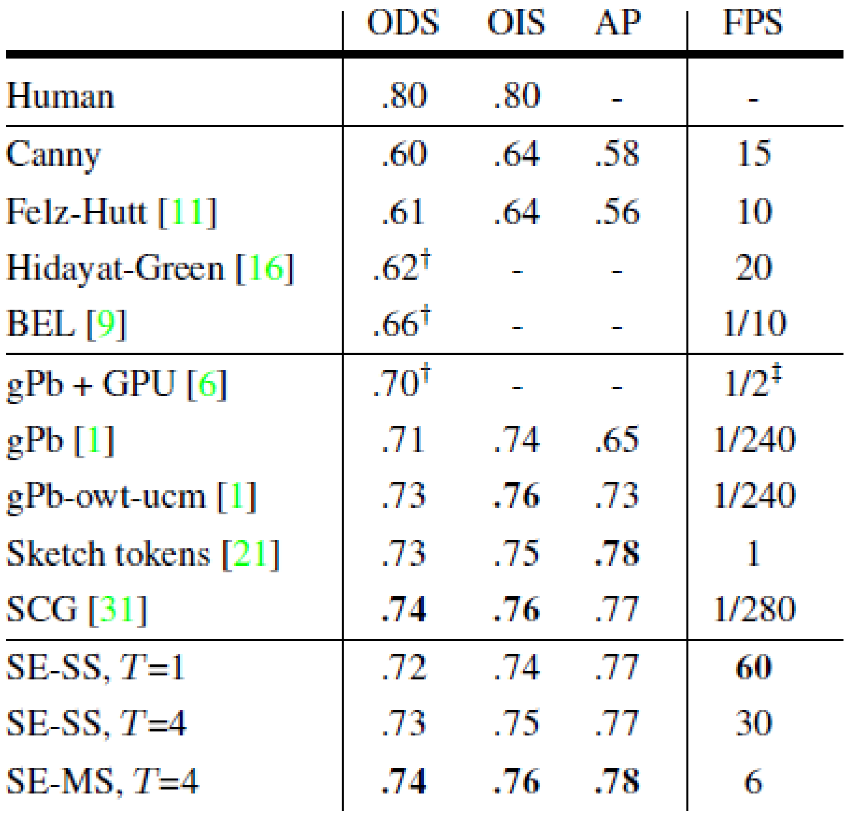
- Algorithm
  1. Extract overlapping 32x32 patches at three scales
  2. Features are pixel values and pairwise differences in feature maps (LUV color, gradient magnitude, oriented gradient)
  3. Predict T boundary maps in the central 16x16 region using T trained decision trees
  4. Average predictions for each pixel across all patches



Edge Detection with Structured Random Forests

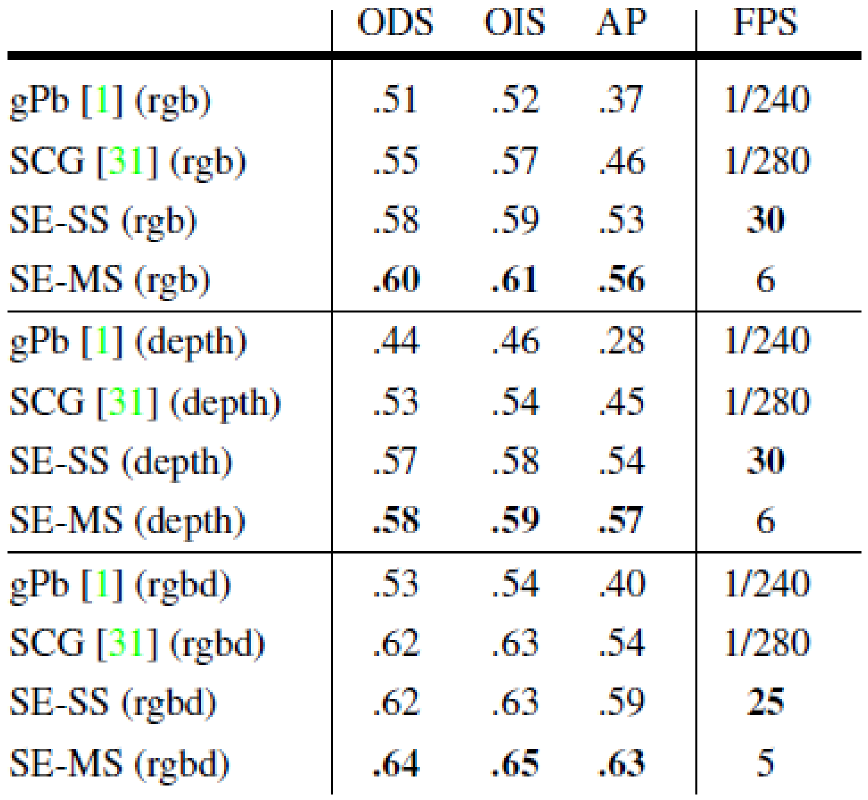
Results

BSDS 500



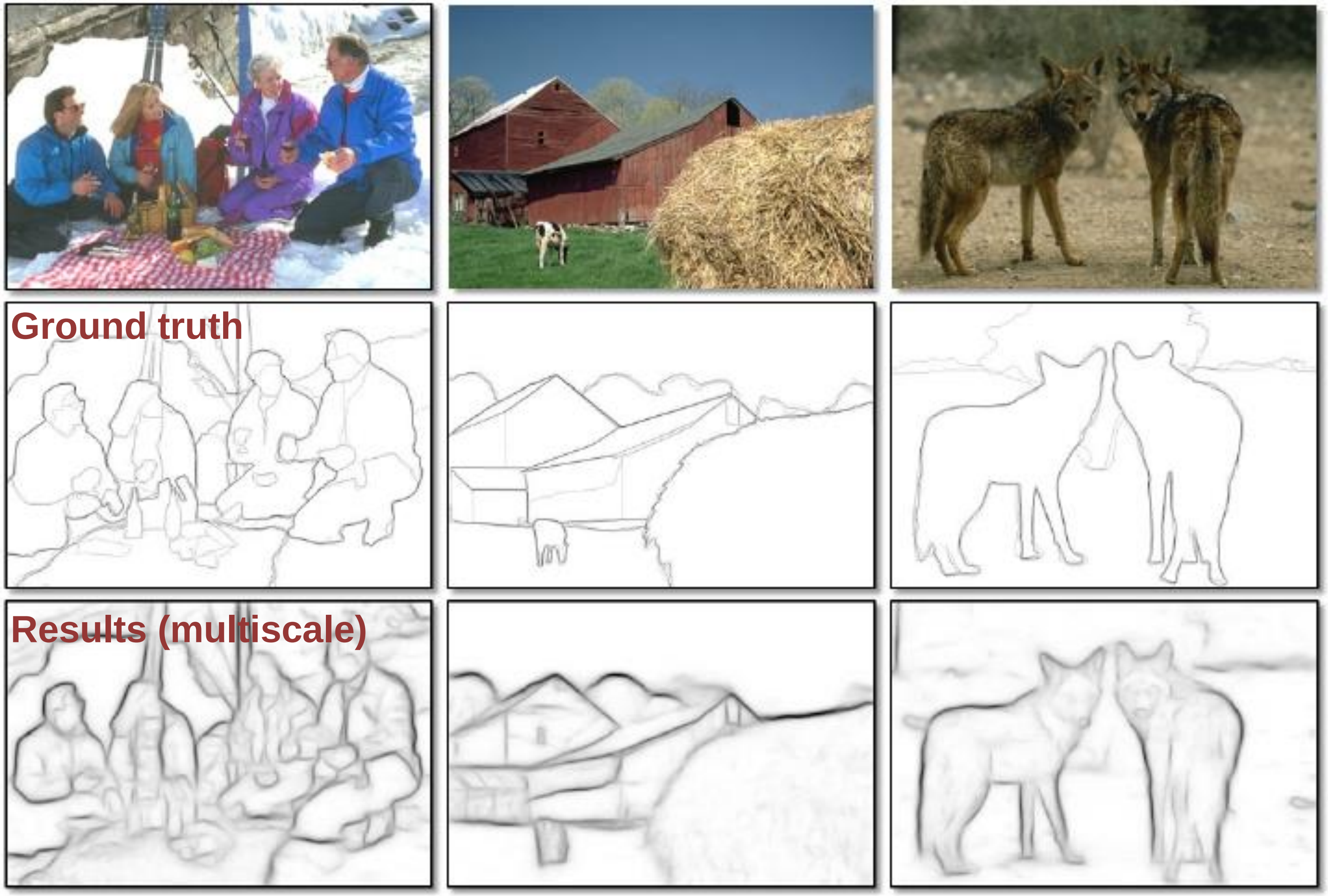
	ODS	OIS	AP	FPS
Human	.80	.80	-	-
Canny	.60	.64	.58	15
Felz-Hutt [11]	.61	.64	.56	10
Hidayat-Green [16]	.62†	-	-	20
BEL [9]	.66†	-	-	1/10
gPb + GPU [6]	.70†	-	-	1/2‡
gPb [1]	.71	.74	.65	1/240
gPb-owt-ucm [1]	.73	**.76**	.73	1/240
Sketch tokens [21]	.73	.75	**.78**	1
SCG [31]	**.74**	**.76**	.77	1/280
SE-SS, T=1	.72	.74	.77	**60**
SE-SS, T=4	.73	.75	.77	30
SE-MS, T=4	**.74**	**.76**	**.78**	6

NYU Depth dataset edges

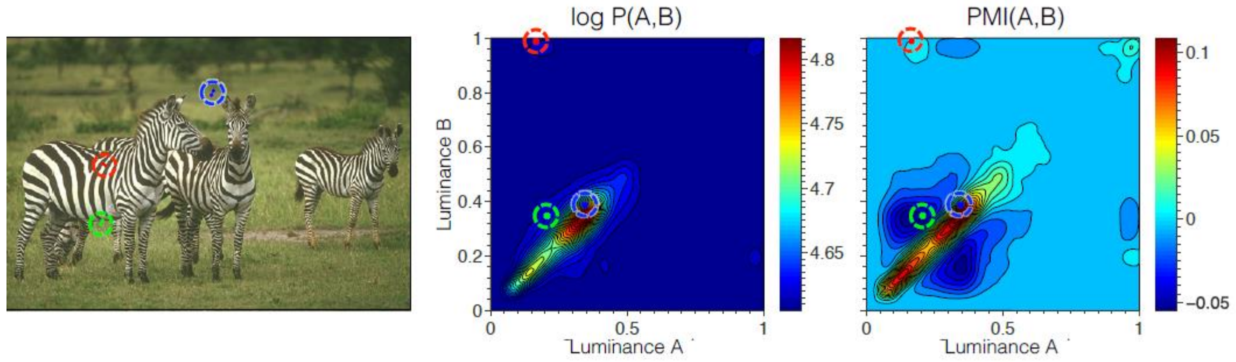


	ODS	OIS	AP	FPS
gPb [1] (rgb)	.51	.52	.37	1/240
SCG [31] (rgb)	.55	.57	.46	1/280
SE-SS (rgb)	.58	.59	.53	**30**
SE-MS (rgb)	**.60**	**.61**	**.56**	6
gPb [1] (depth)	.44	.46	.28	1/240
SCG [31] (depth)	.53	.54	.45	1/280
SE-SS (depth)	.57	.58	.54	**30**
SE-MS (depth)	**.58**	**.59**	**.57**	6
gPb [1] (rgbd)	.53	.54	.40	1/240
SCG [31] (rgbd)	.62	.63	.54	1/280
SE-SS (rgbd)	.62	.63	.59	**25**
SE-MS (rgbd)	**.64**	**.65**	**.63**	5

Edge Detection with Structured Random Forests



Crisp Boundary Detection using Pointwise Mutual Information (Isola et al. ECCV 2014)



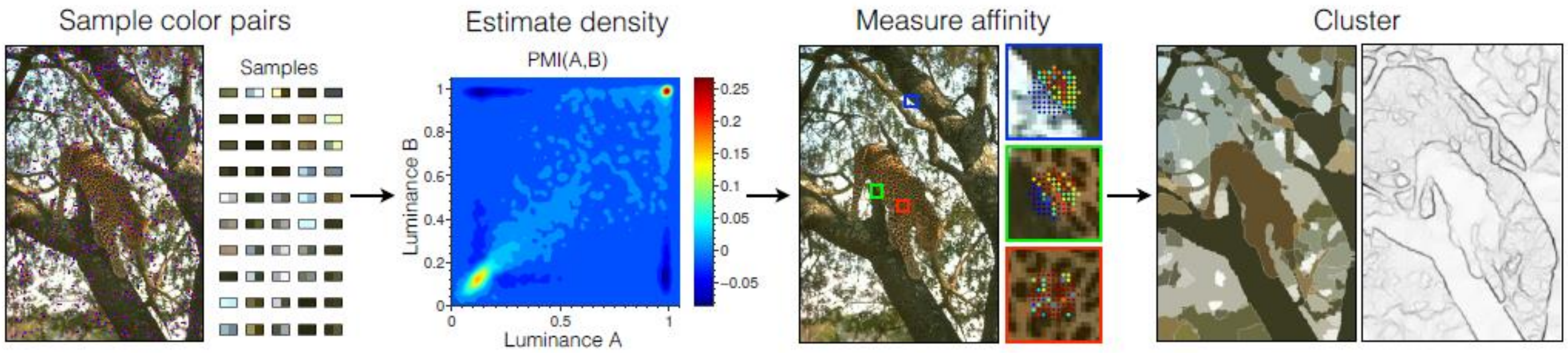
$$\mathrm{PMI}_\rho(A,B) = \log \frac{P(A,B)^\rho}{P(A)P(B)}$$

Pixel combinations that are unlikely to be together are edges

Algorithm:

Kernel density estimation

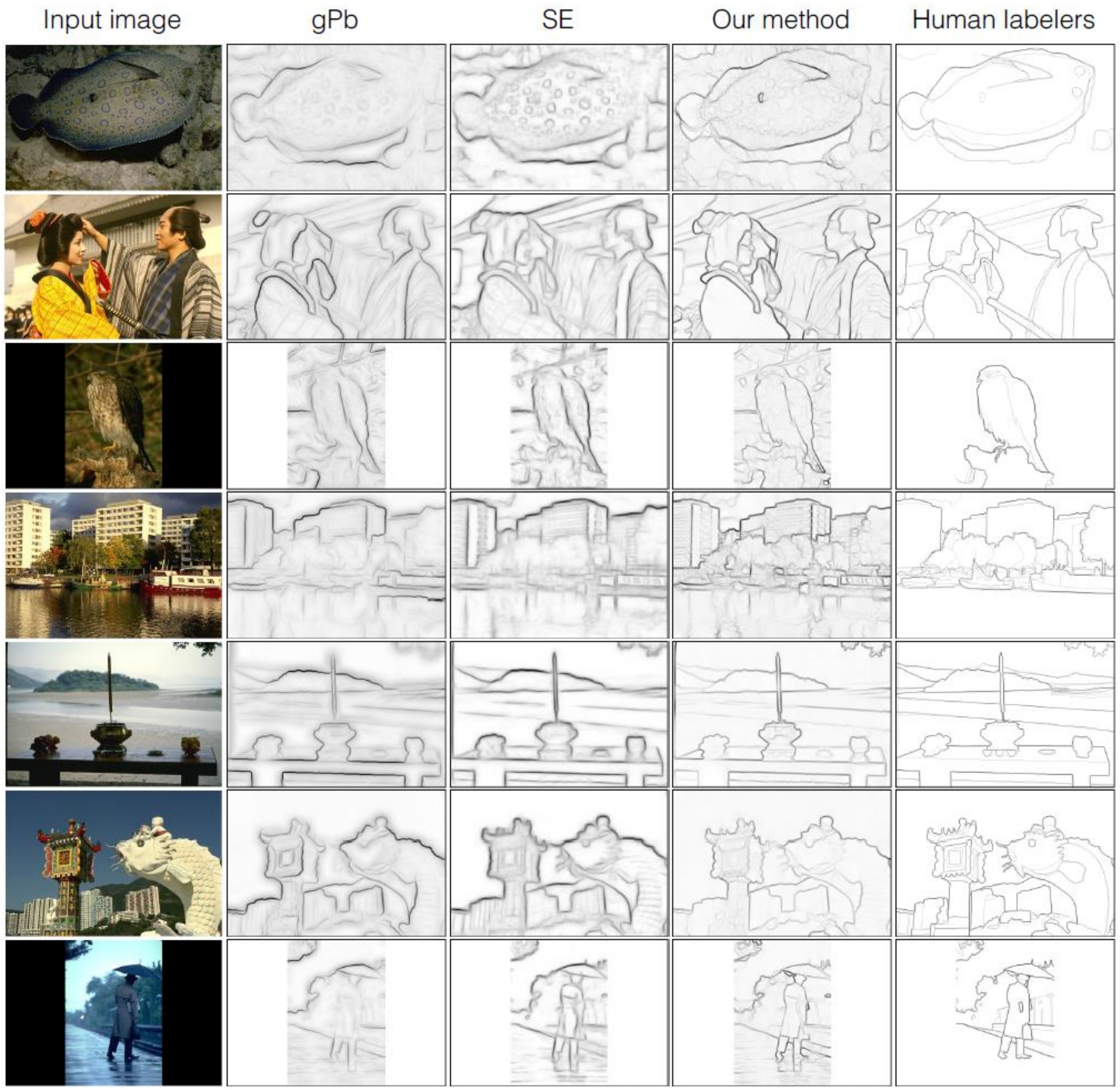
Spectral clustering



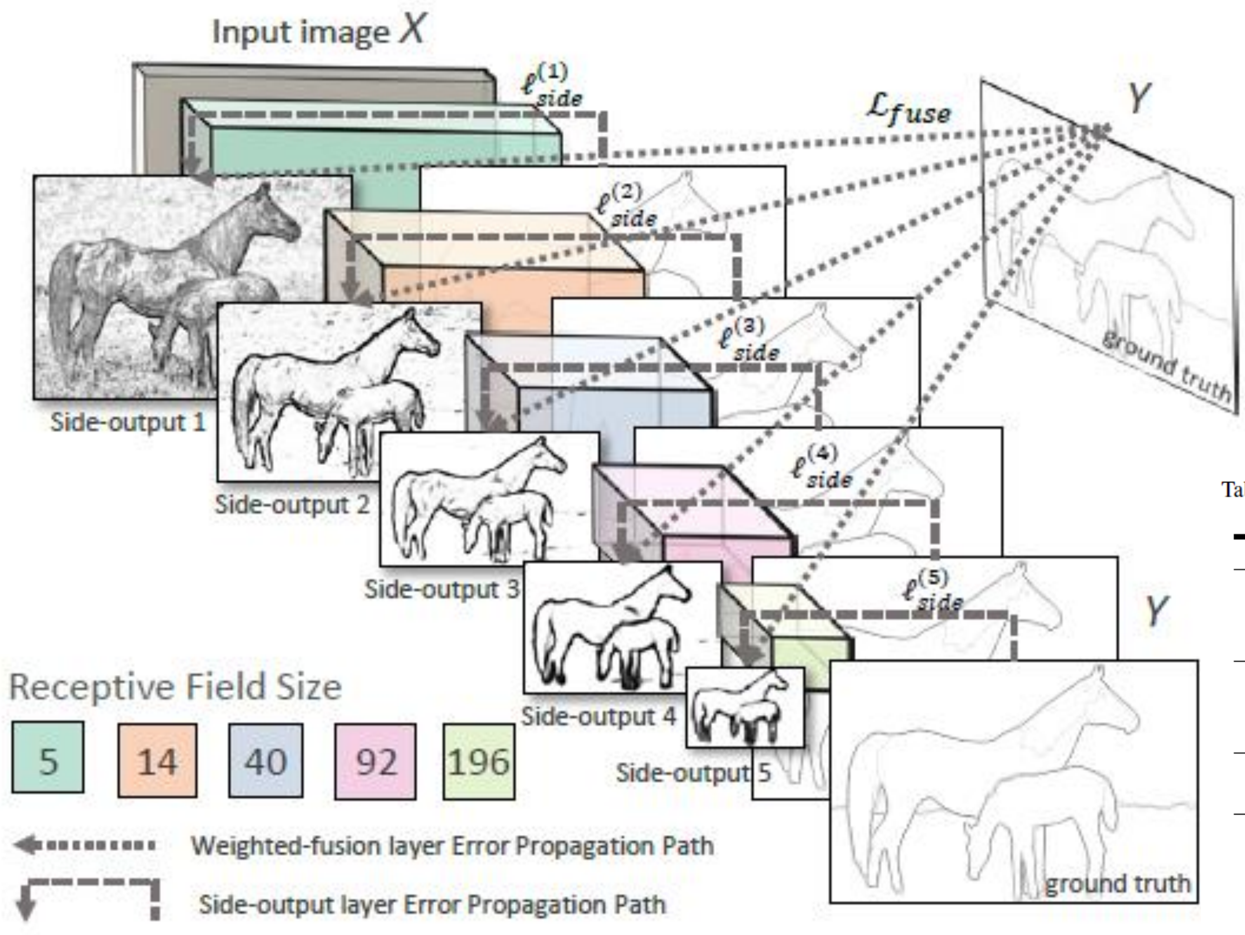
http://web.mit.edu/phillipi/www/publications/crisp_boundaries.pdf

Crisp Boundary Detection using Pointwise Mutual Information

Algorithm	ODS	OIS	AP
Canny [14]	0.60	0.63	0.58
Mean Shift [36]	0.64	0.68	0.56
NCuts [37]	0.64	0.68	0.45
Felz-Hutt [38]	0.61	0.64	0.56
gPb [1]	0.71	0.74	0.65
gPb-owt-ucm [1]	0.73	0.76	0.73
SCG [9]	**0.74**	0.76	0.77
Sketch Tokens [7]	0.73	0.75	0.78
SE [8]	**0.74**	0.76	0.78
Our method – SS, color only	0.72	0.75	0.77
Our method – SS	0.73	0.76	**0.79**
Our method – MS	**0.74**	**0.77**	0.78



Holistically nested edge detection



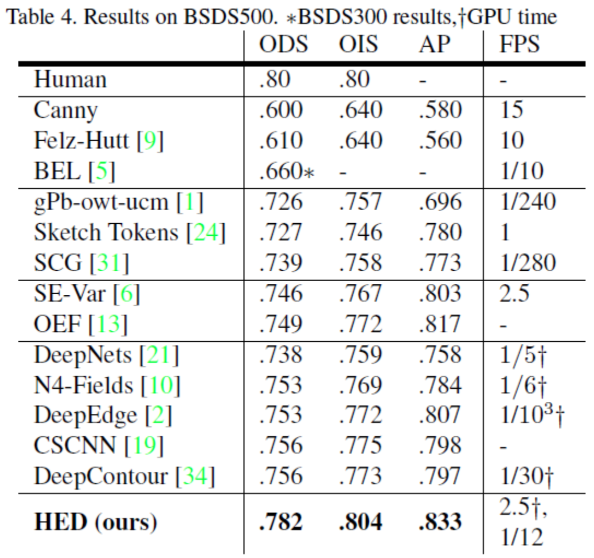


Table 4. Results on BSDS500. ∗BSDS300 results,†GPU time

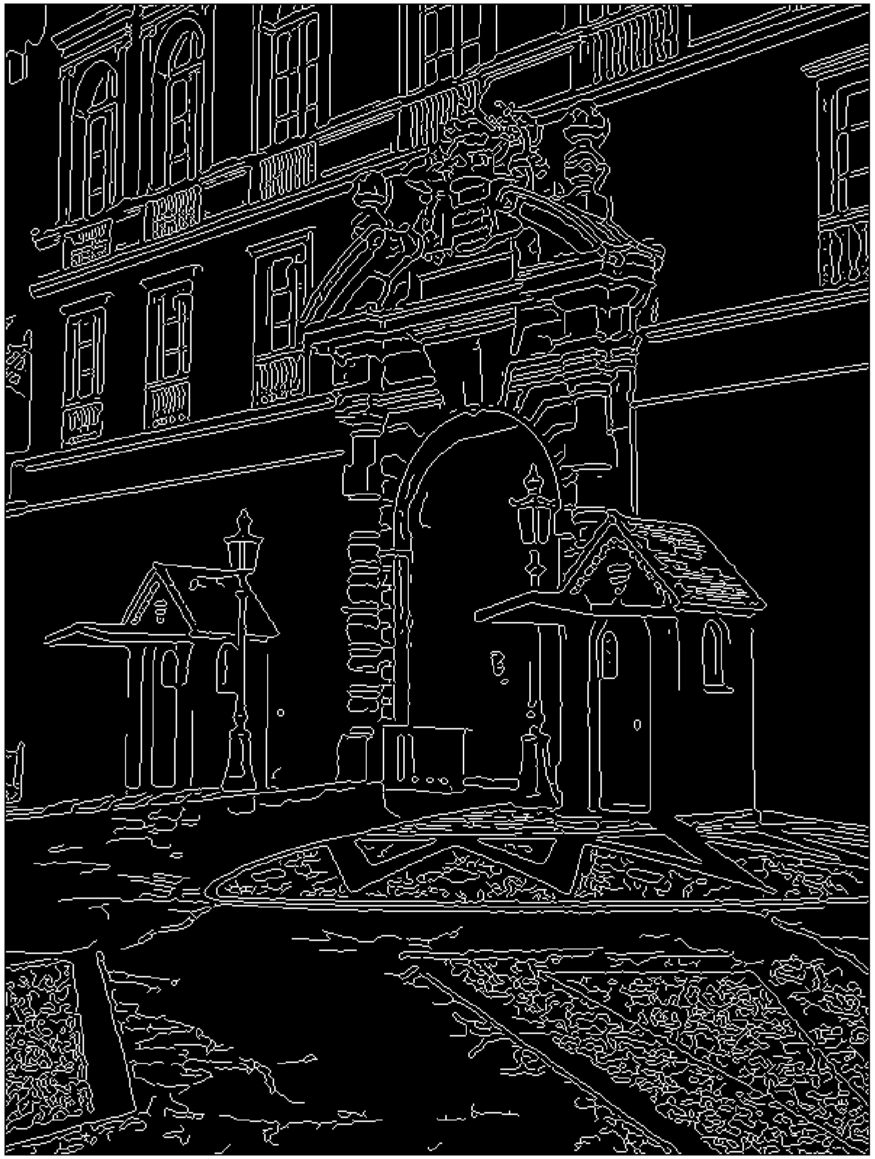
	ODS	OIS	AP	FPS
Human	.80	.80	-	-
Canny	.600	.640	.580	15
Felz-Hutt [9]	.610	.640	.560	10
BEL [5]	.660∗	-	-	1/10
gPb-owt-ucm [1]	.726	.757	.696	1/240
Sketch Tokens [24]	.727	.746	.780	1
SCG [31]	.739	.758	.773	1/280
SE-Var [6]	.746	.767	.803	2.5
OEF [13]	.749	.772	.817	-
DeepNets [21]	.738	.759	.758	1/5†
N4-Fields [10]	.753	.769	.784	1/6†
DeepEdge [2]	.753	.772	.807	$1/10^3$†
CSCNN [19]	.756	.775	.798	-
DeepContour [34]	.756	.773	.797	1/30†
HED (ours)	**.782**	**.804**	**.833**	2.5†, 1/12

https://arxiv.org/pdf/1504.06375.pdf

State of edge detection

- Local edge detection is mostly solved
  - Intensity gradient, color, texture
  - HED on BSDS 500 is near human performance
- Some room for improvement by taking advantage of higher-level knowledge (e.g., objects)
- Still hard to produce all objects within a small number of regions

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

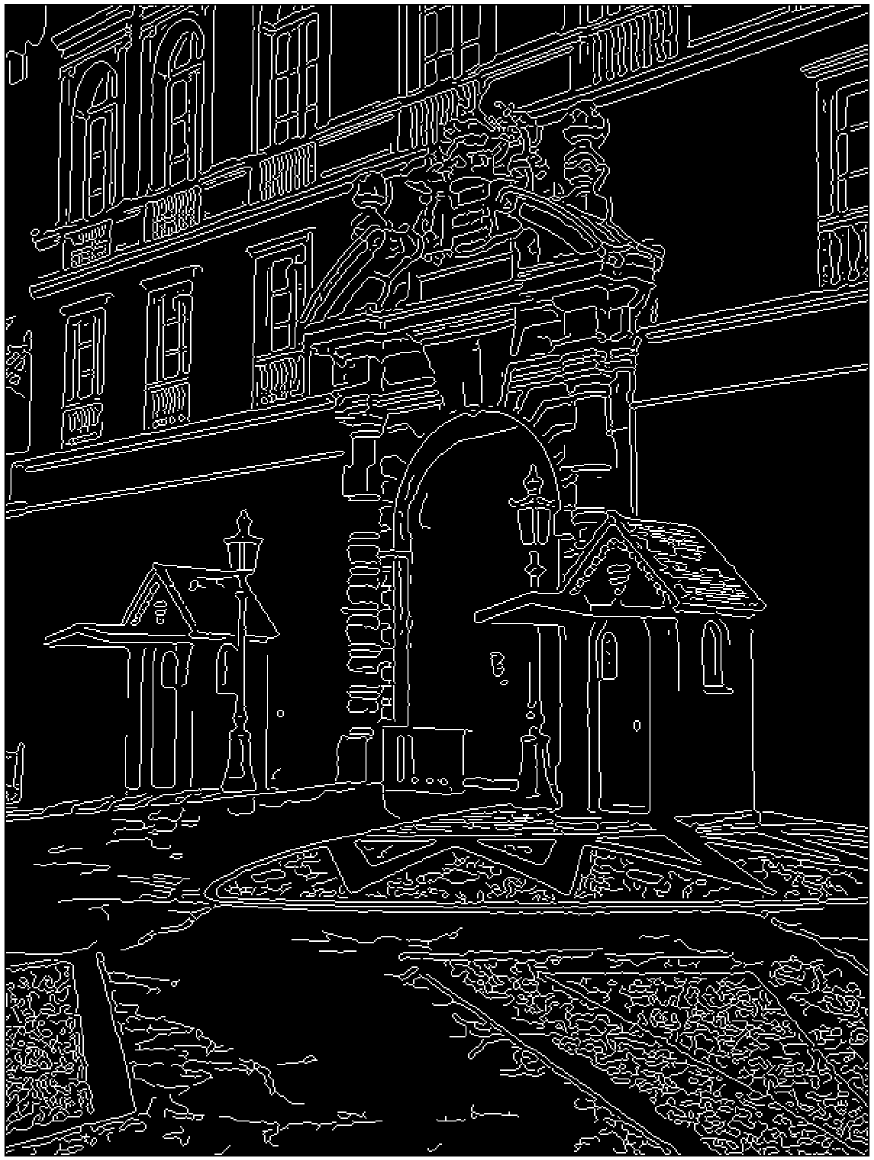
$$\mathbf{M} = \begin{bmatrix} \sum (x-\mu_x)^2 & \sum (x-\mu_x)(y-\mu_y) \\ \sum (x-\mu_x)(y-\mu_y) & \sum (y-\mu_y)^2 \end{bmatrix} \qquad [v, \lambda] = \mathrm{eig}(\mathbf{M})$$

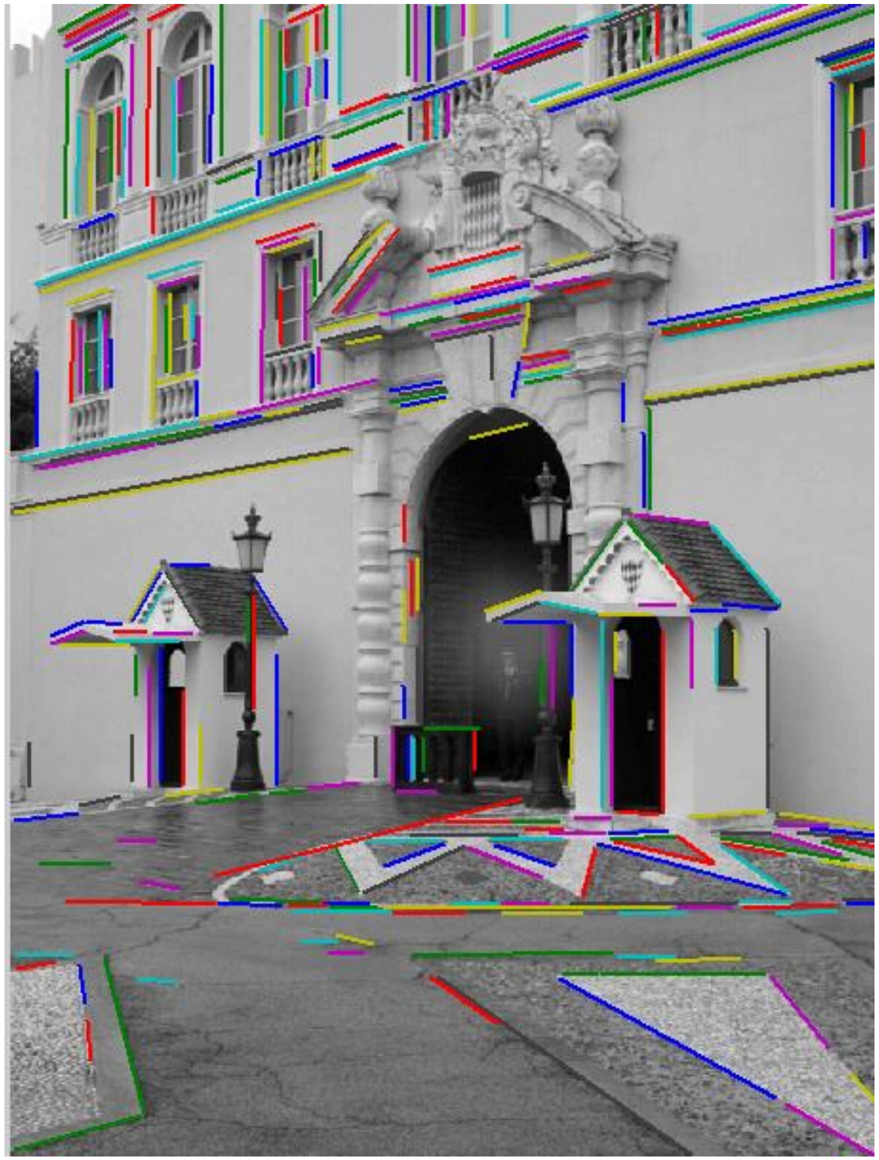
$$\theta = \mathrm{atan}\,2(\underset{\swarrow}{\overset{\text{Larger eigenvector}}{v}}(2,2), v(1,2))$$

$$conf = \lambda_2 / \lambda_1$$

5. Threshold on straightness, store segment

2. Canny lines → … →straight edges





Homework 1

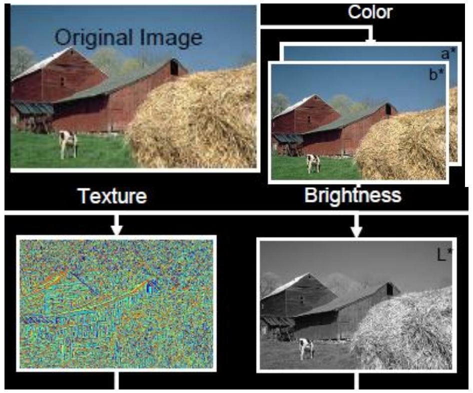
- Due Feb 13, but try to finish sooner (HW 2 will take quite a bit more time)

https://courses.engr.illinois.edu/cs543/hws/hw1_cs543_sp17.pdf

Things to remember

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

- Pb: learns weighting of gradient, color, texture differences
  - More recent learning approaches give at least as good accuracy and are faster



- 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