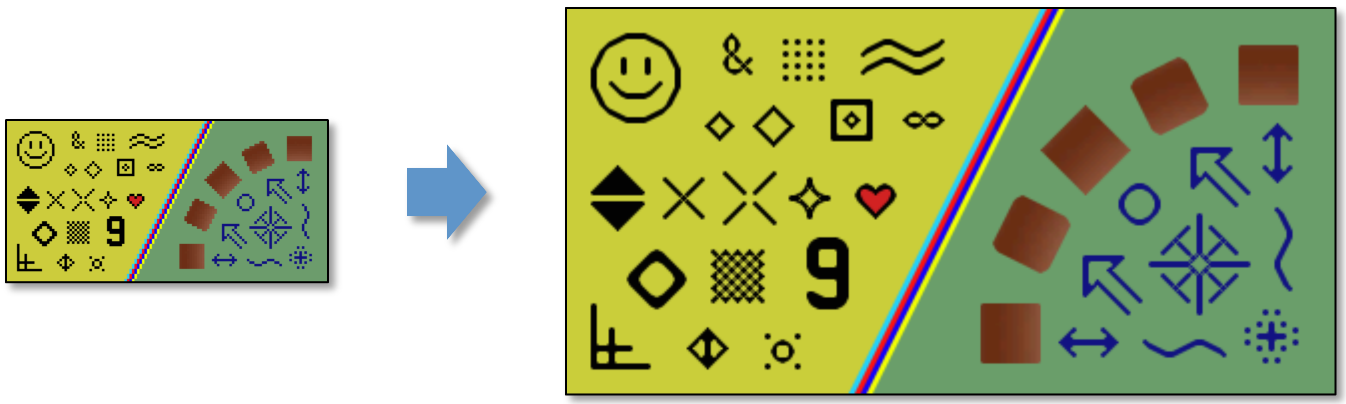
CS4670/5670: Computer Vision

Kavita Bala

Lecture 7: Resampling and Edge Detection



Announcements

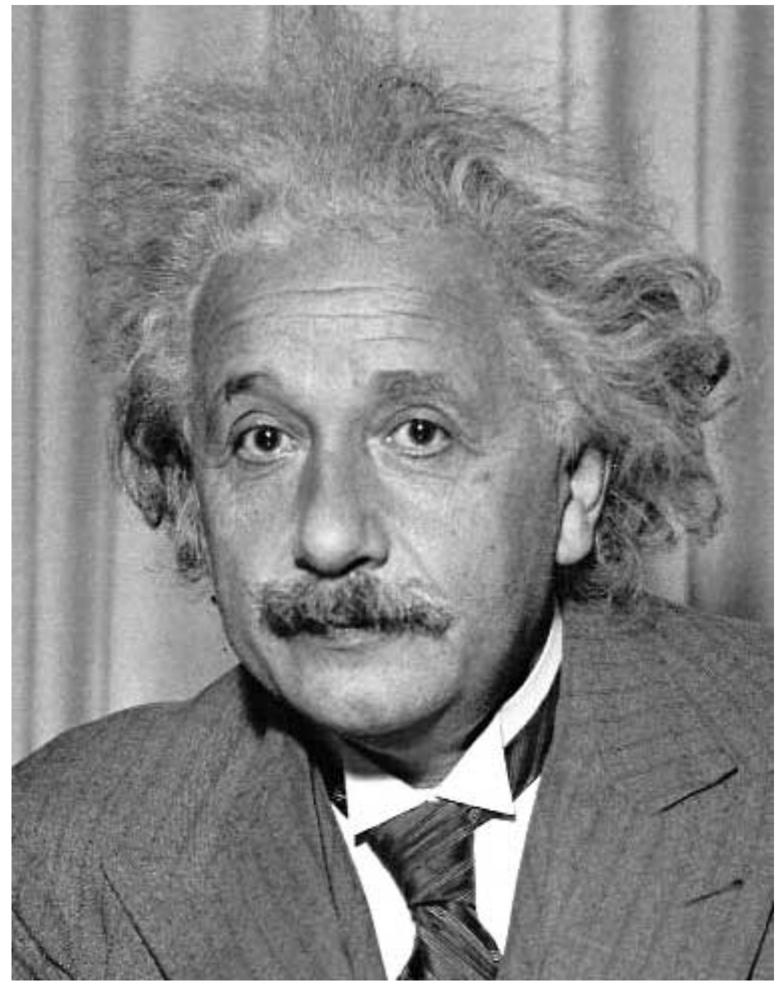
- PA1-A is out
  - Written part alone
  - Coding part in pairs
- My office hours moved
  - Dhruv has hours from 3-4

SuperBowl

- Lines on the field

Template matching

- Goal: find  in image
- Main challenge: What is a good similarity or distance measure between two patches?
  - Sum Square Difference
  - Normalized Cross Correlation

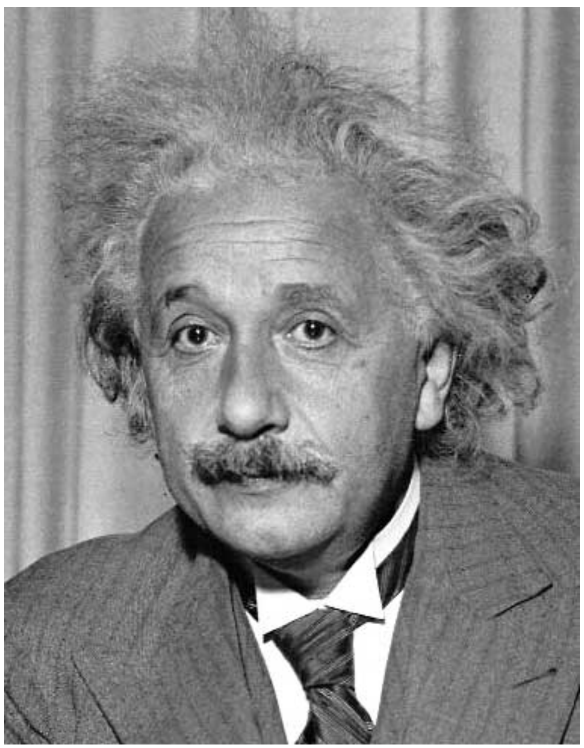


Matching with filters

- Goal: find  in image
- Method: SSD

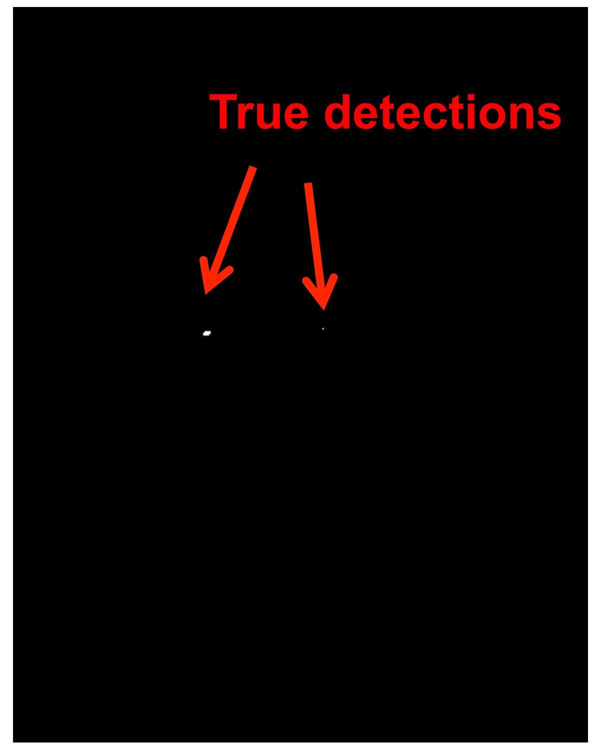
f = image
g = filter

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k, n+l])^2$$



Input

1- sqrt(SSD)



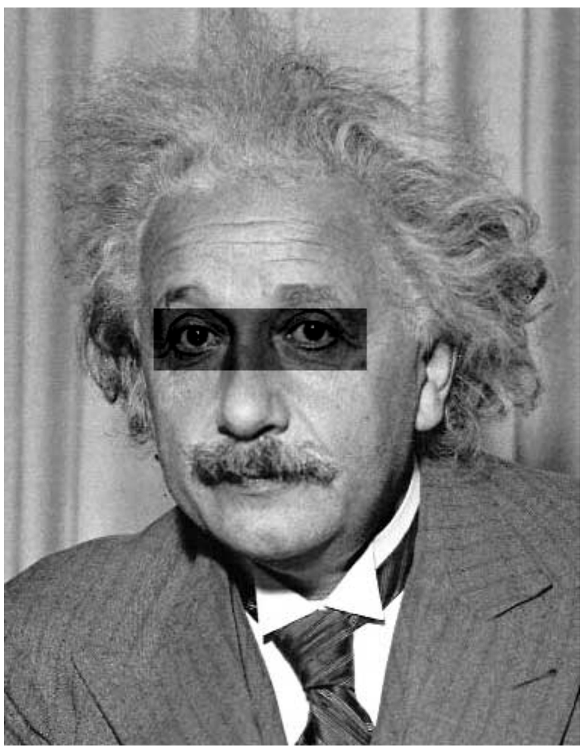
Thresholded Image

Matching with filters

- Goal: find  in image
- Method: SSD

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k, n+l])^2$$

What's the potential downside of SSD?



Input

1- sqrt(SSD)

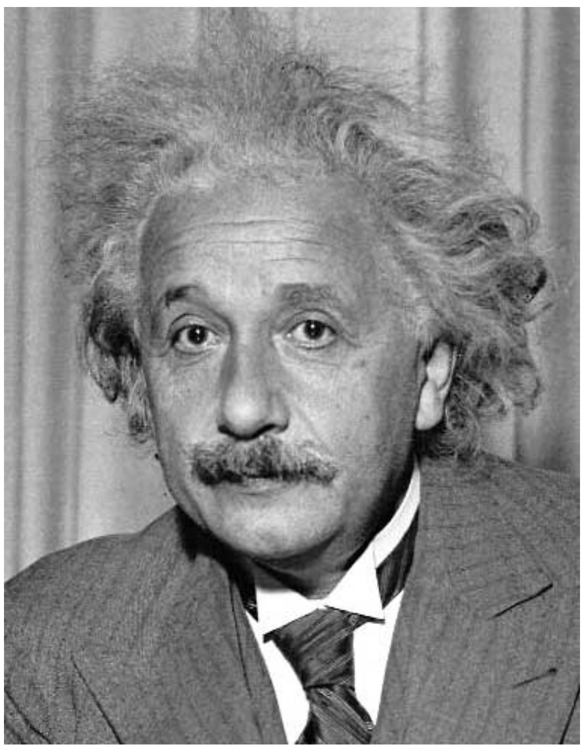
Matching with filters

- Goal: find  in image
- Method: Normalized cross-correlation

$$h[m,n] = \frac{\sum_{k,l} (g[k,l] - \overset{\text{mean template}}{\bar{g}})(f[m-k,n-l] - \overset{\text{mean image patch}}{\bar{f}_{m,n}})}{\left(\sum_{k,l} (g[k,l] - \bar{g})^2 \sum_{k,l} (f[m-k,n-l] - \bar{f}_{m,n})^2 \right)^{0.5}}$$

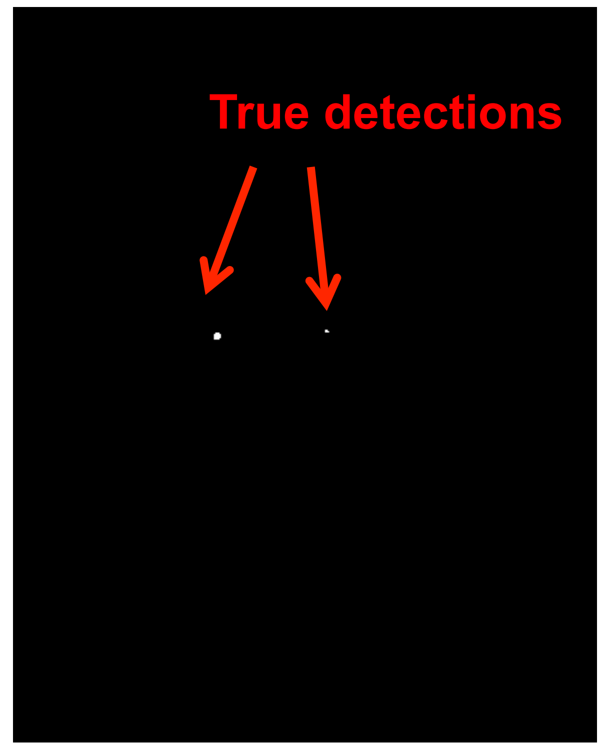
Matching with filters

- Goal: find  in image
- Method: Normalized cross-correlation



Input

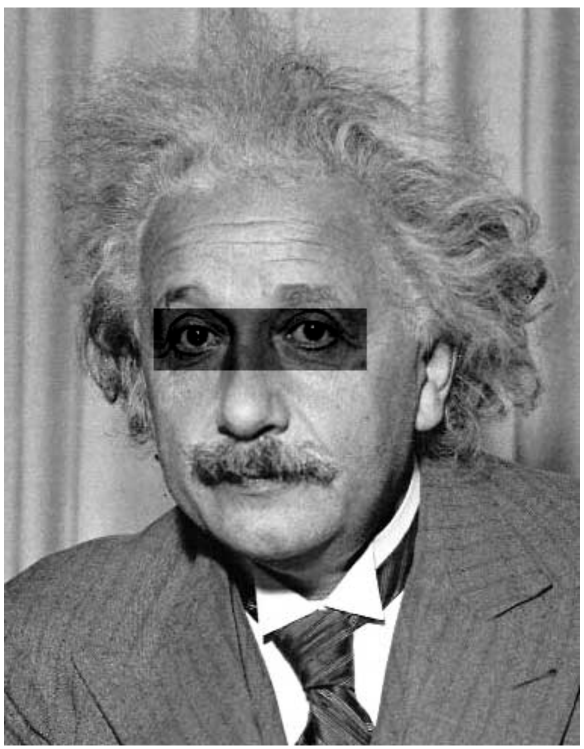
Normalized X-Correlation



Thresholded Image

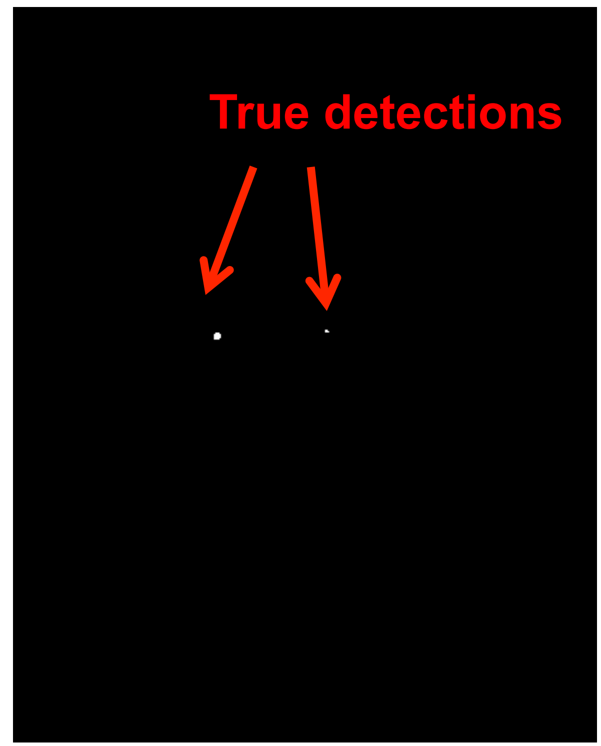
Matching with filters

- Goal: find  in image
- Method: Normalized cross-correlation



Input

Normalized X-Correlation



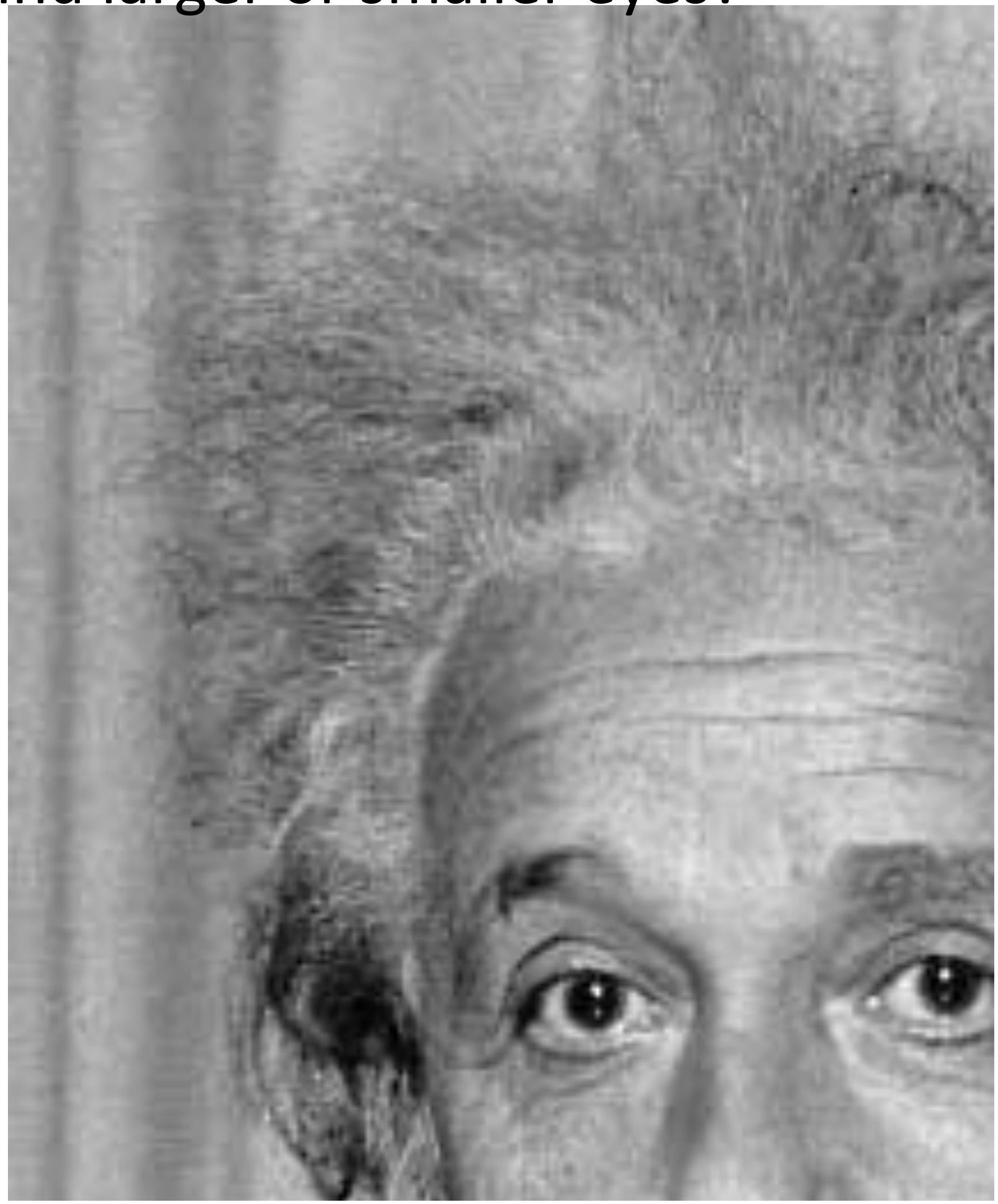
Thresholded Image

Q: What is the best method to use?

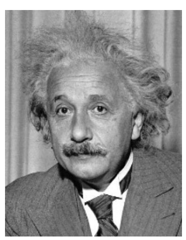
A: Depends

- SSD: faster, sensitive to overall intensity
- Normalized cross-correlation: slower, invariant to local average intensity and contrast
- But really, neither of these baselines are representative of modern recognition

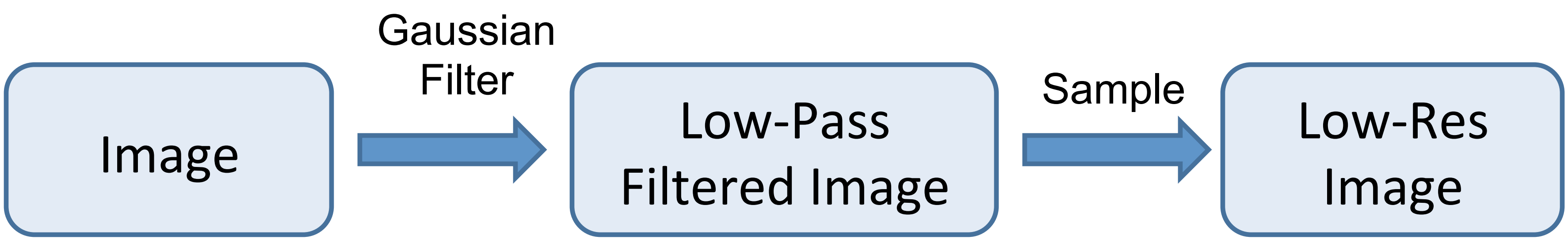
Q: What if we want to find larger or smaller eyes?



A: Image Pyramid



Review of Sampling



Template Matching with Image Pyramids

Input: Image, Template

1. Match template at current scale

2. Downsample image

3. Repeat 1-2 until image is very small

4. Take responses above some threshold, perhaps with non-maxima suppression

Image representation

- Pixels: great for spatial resolution, poor access to frequency
- Fourier transform: great for frequency, not for spatial info
- Pyramids/filter banks: balance between spatial and frequency information

Major uses of image pyramids

- Compression

- Object detection
  - Scale search
  - Features

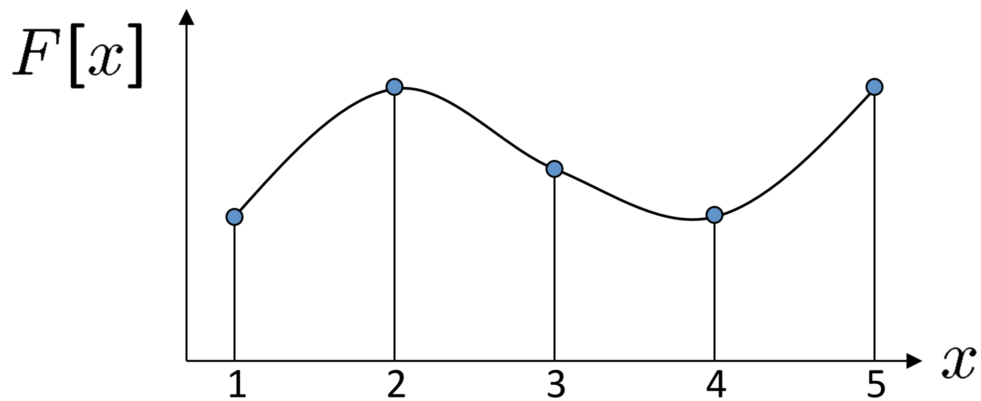
- Detecting stable interest points

- Registration
  - Course-to-fine

Upsampling

- This image is too small for this screen: 
- How can we make it 10 times as big?
- Simplest approach:
  repeat each row
  and column 10 times
- (“Nearest neighbor
  interpolation”)

Image interpolation



d = 1 in this example

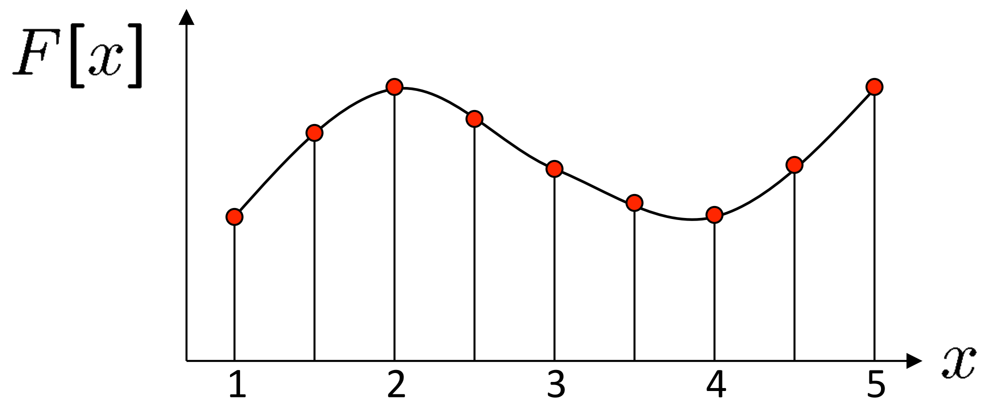
Recall how a digital image is formed

$$F[x,y] = \text{quantize}\{f(xd, yd)\}$$

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

Adapted from: S. Seitz

Image interpolation



d = 1 in this example

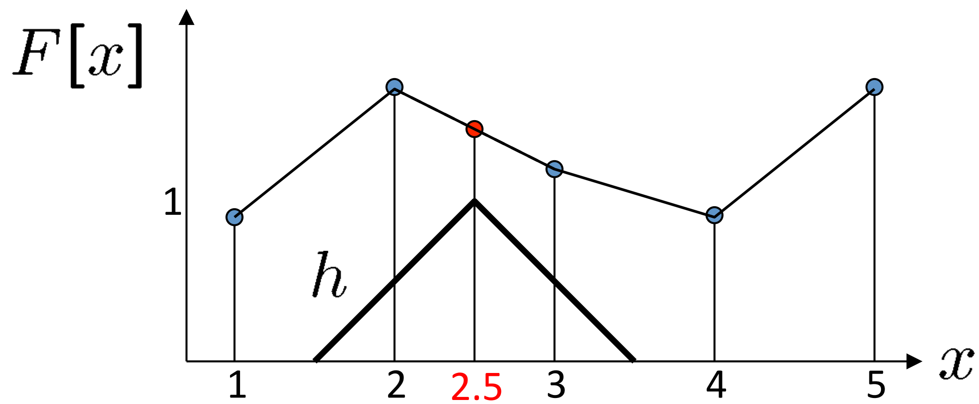
Recall how a digital image is formed

$$F[x,y] = \text{quantize}\{f(xd, yd)\}$$

- It is a discrete point-sampling of a continuous function
- If we could somehow reconstruct the original function, any new image could be generated, at any resolution and scale

Adapted from: S. Seitz

Image interpolation



d = 1 in this example

- What if we don't know f ?
  - Guess an approximation: $\tilde{f}$
  - Can be done in a principled way: filtering
  - Convert F to a continuous function:

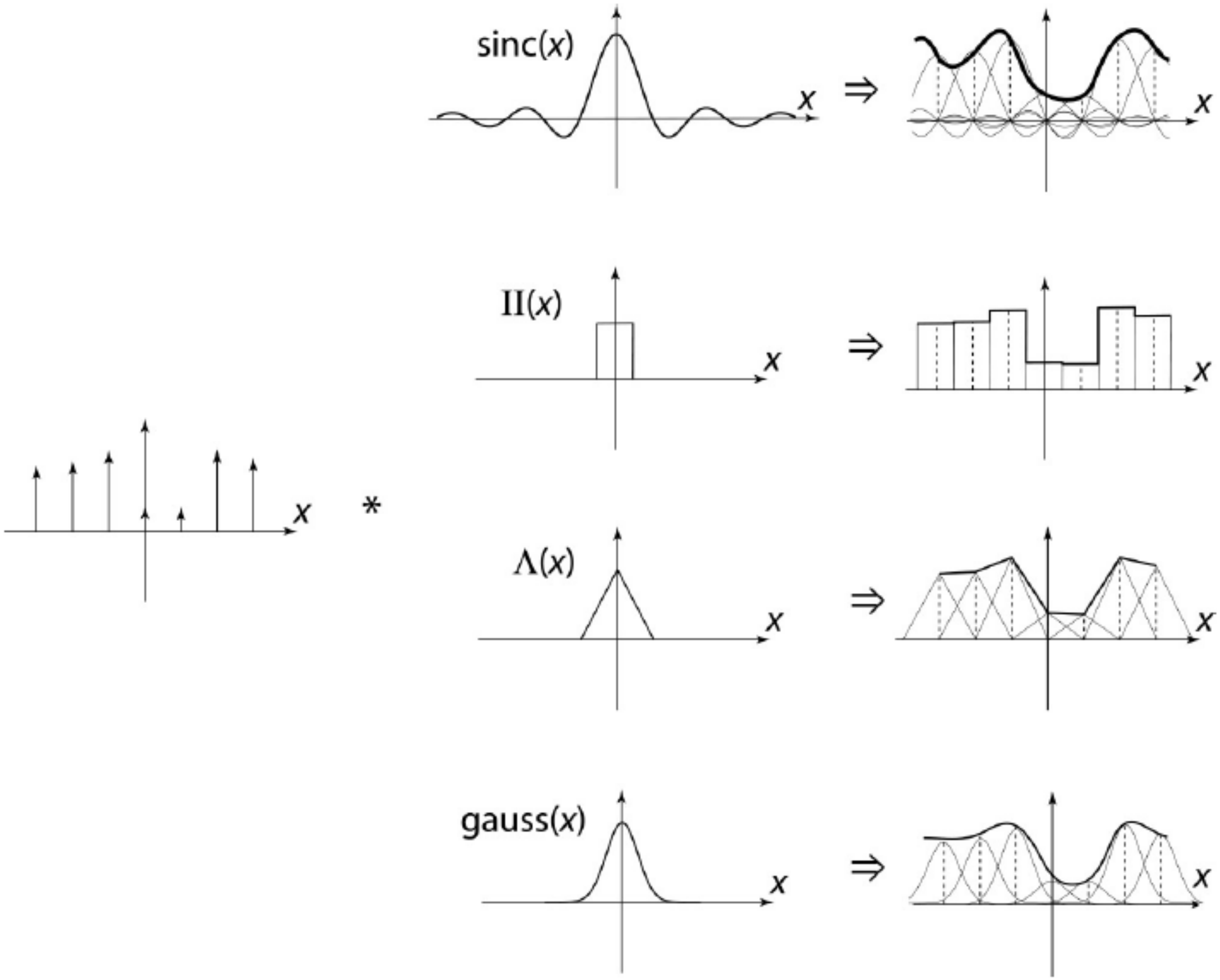
$$f_F(x) = F(\tfrac{x}{d}) \text{ when } \tfrac{x}{d} \text{ is an integer, 0 otherwise}$$

  - Reconstruct by convolution with a *reconstruction filter, h*

$$\tilde{f} = h * f_F$$

Adapted from: S. Seitz

Image interpolation



"Ideal" reconstruction

Nearest-neighbor interpolation

Linear interpolation

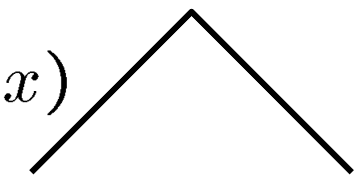
Gaussian reconstruction

Source: B. Curless

Reconstruction filters

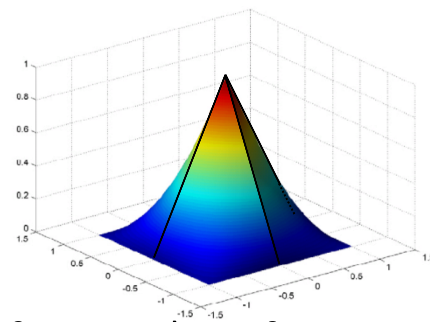
- What does the 2D version of this hat function look like?

$h(x)$



performs
linear interpolation

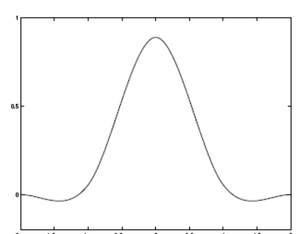
$h(x,y)$



(tent function) performs
bilinear interpolation

Better filters give better resampled images

- **Bicubic** is common choice



Cubic reconstruction filter

$$r(x) = \frac{1}{6}\begin{cases} (12-9B-6C)|x|^3 + (-18+12B+6C)|x|^2 + (6-2B) & |x| < 1 \\ ((-B-6C)|x|^3 + (6B+30C)|x|^2 + (-12B-48C)|x| + (8B+24C) & 1 \le |x| < 2 \\ 0 & \textit{otherwise} \end{cases}$$

Image interpolation

Original image:  x 10

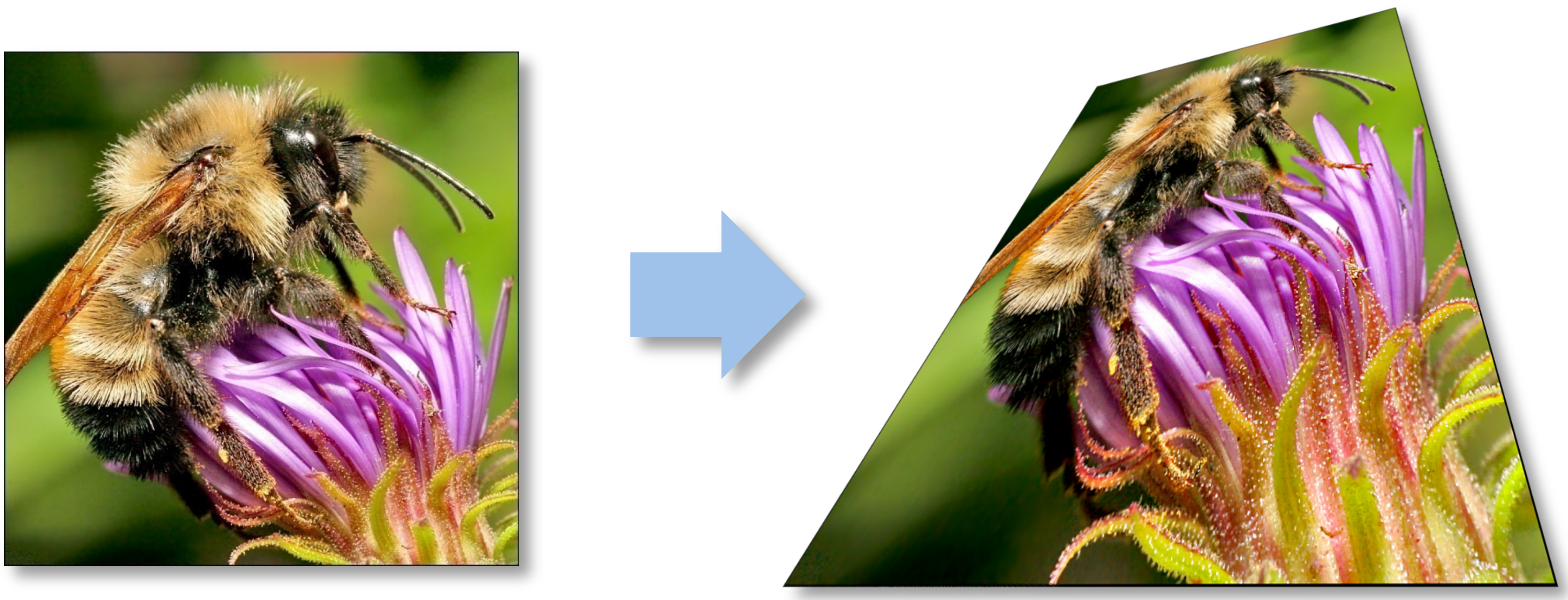
Nearest-neighbor interpolation

Bilinear interpolation

Bicubic interpolation

Image interpolation

Also used for *resampling*



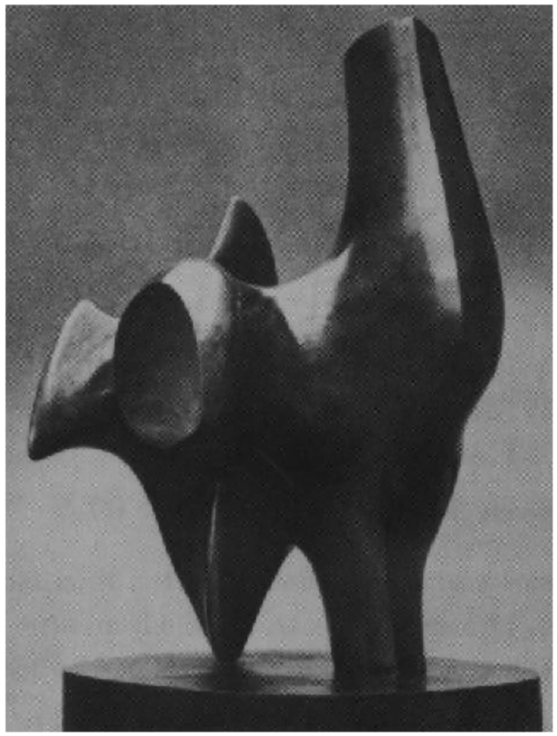
CS4670/5670: Computer Vision

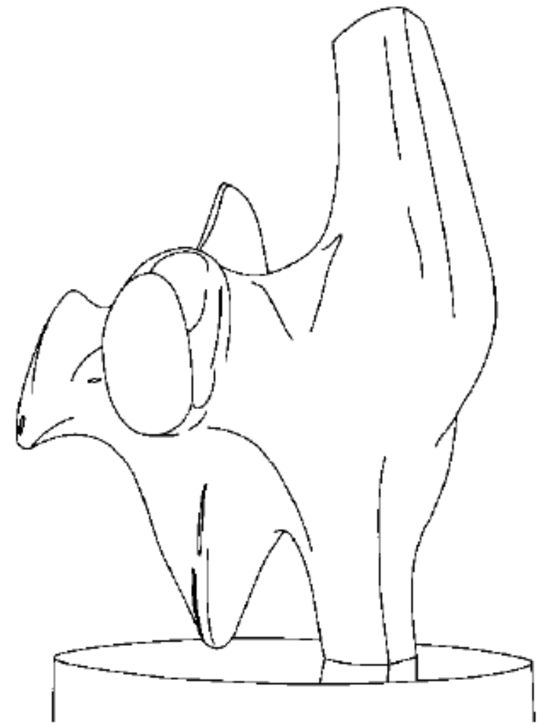
Kavita Bala

Edge detection

From [Sandlot Science]

Why edges?



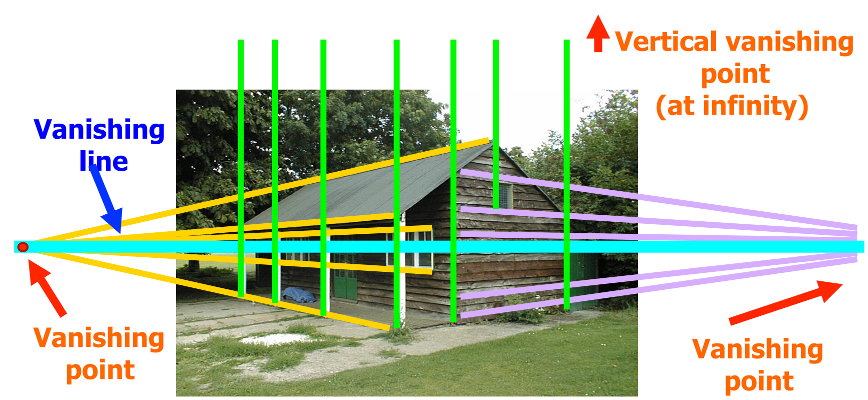


- Humans are sensitive to edges
- Convert a 2D image into a set of curves
  - Extracts salient features of the scene, more compact

Why do we care about edges?

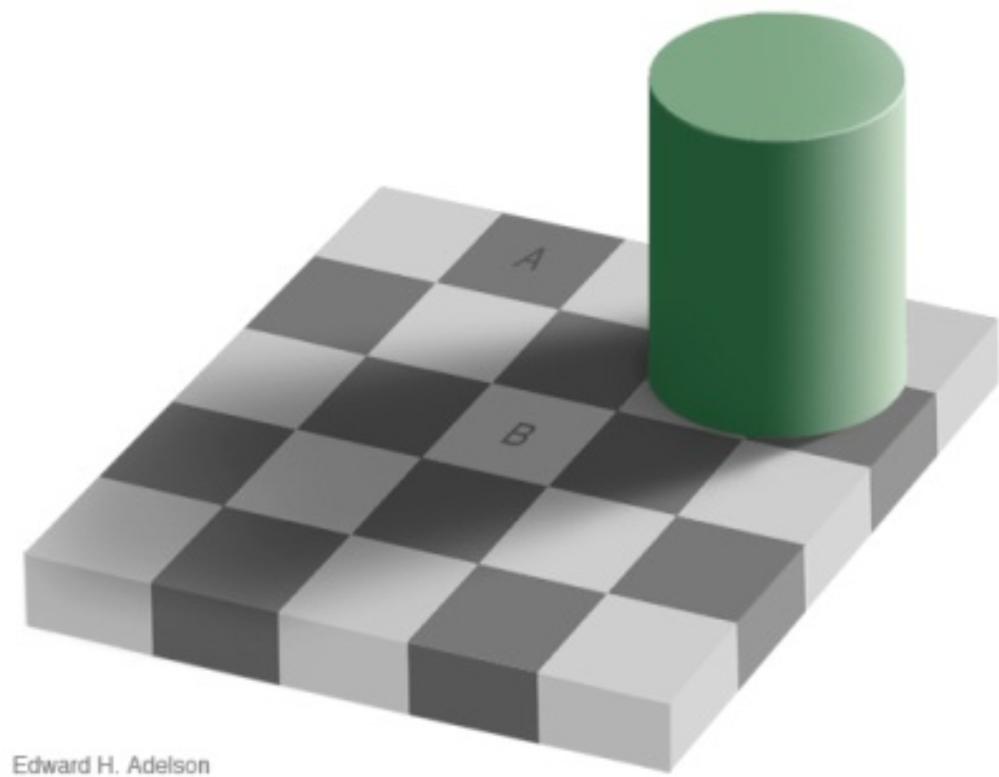
- Extract information, recognize objects

- Recover geometry and viewpoint

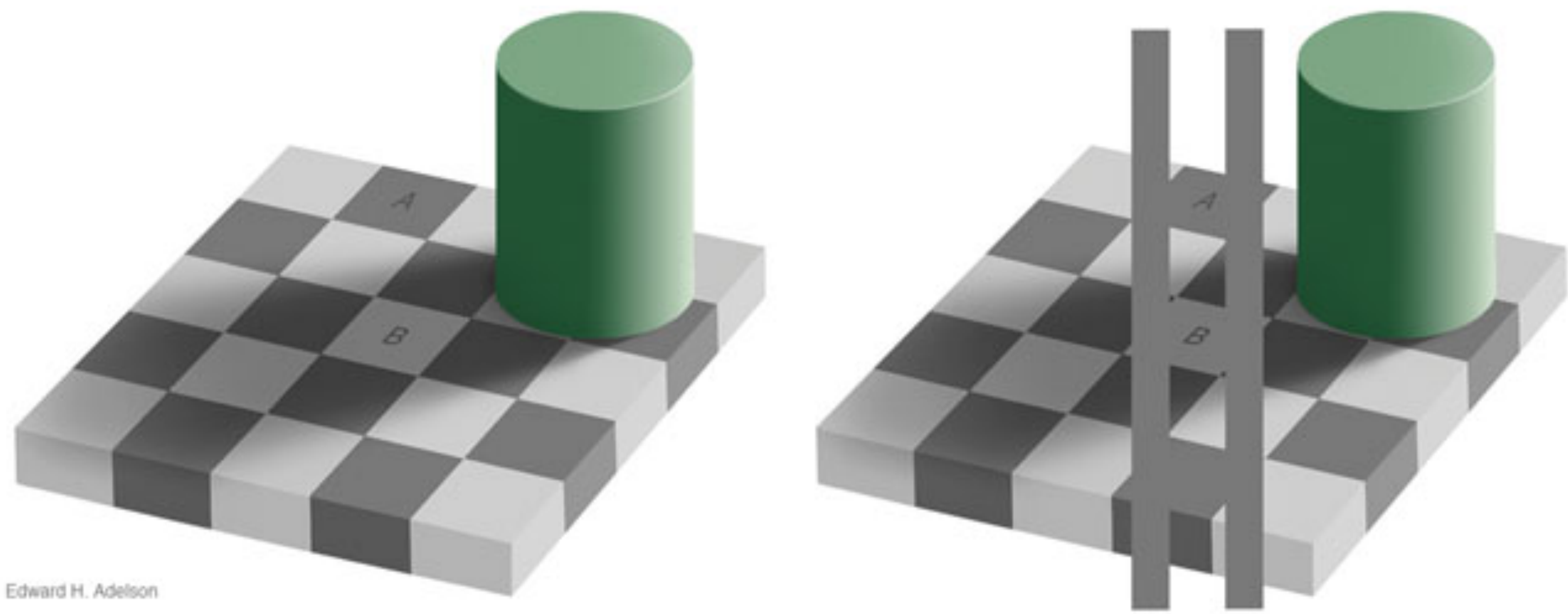


Locating Structural Features

- Edges are curves in the image, across which the brightness changes “a lot”
- Corners/Junctions



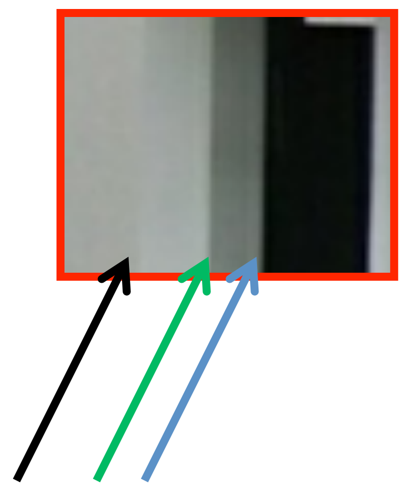
Aside



Closeup of edges

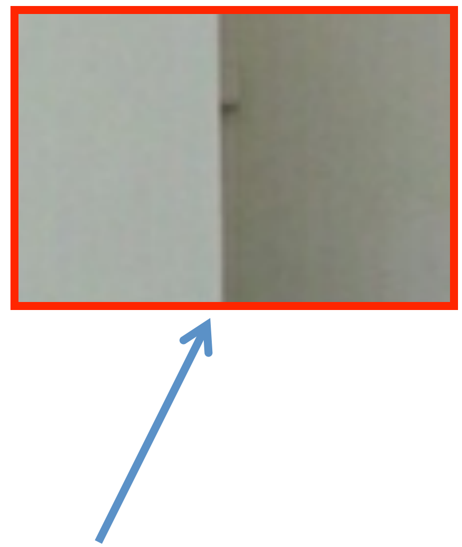
Source: D. Hoiem

Closeup of edges



Source: D. Hoiem

Closeup of edges



Source: D. Hoiem

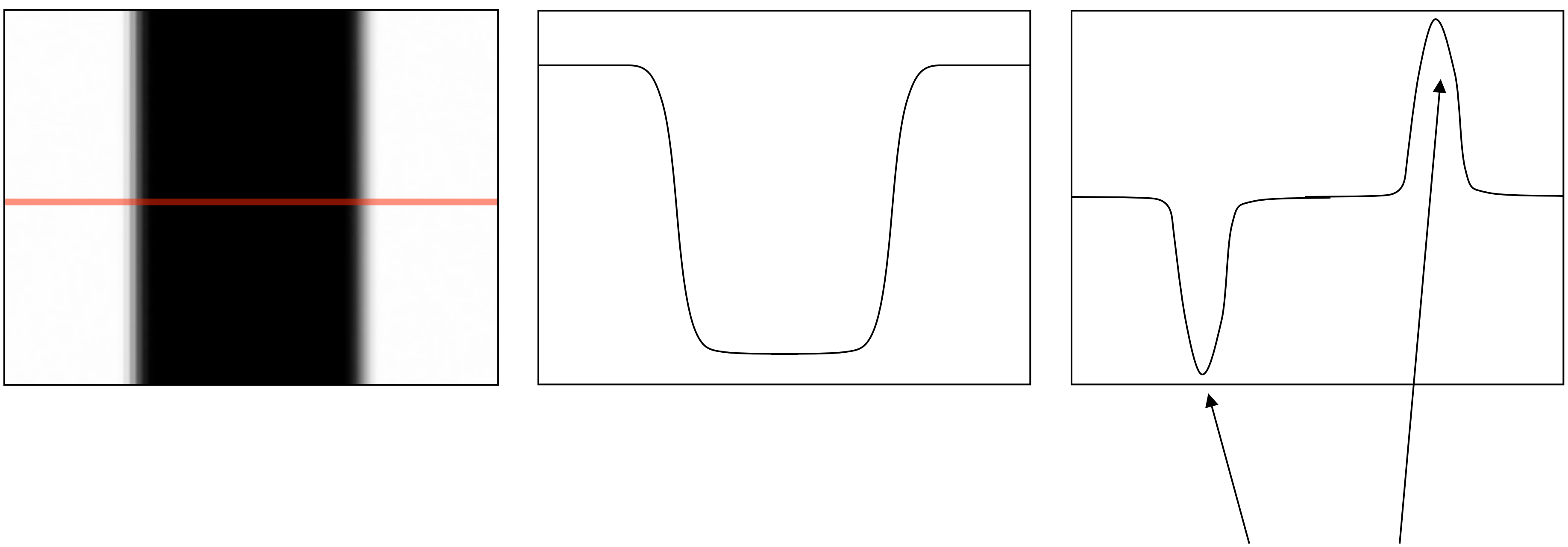
Closeup of edges

Source: D. Hoiem

Characterizing edges

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

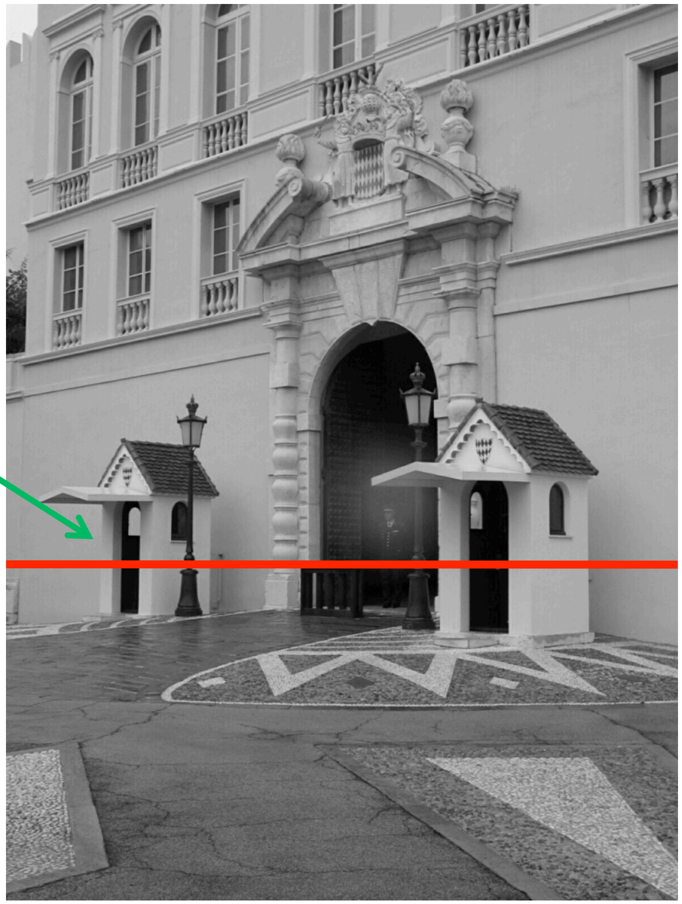
image | intensity function (along horizontal scanline) | first derivative

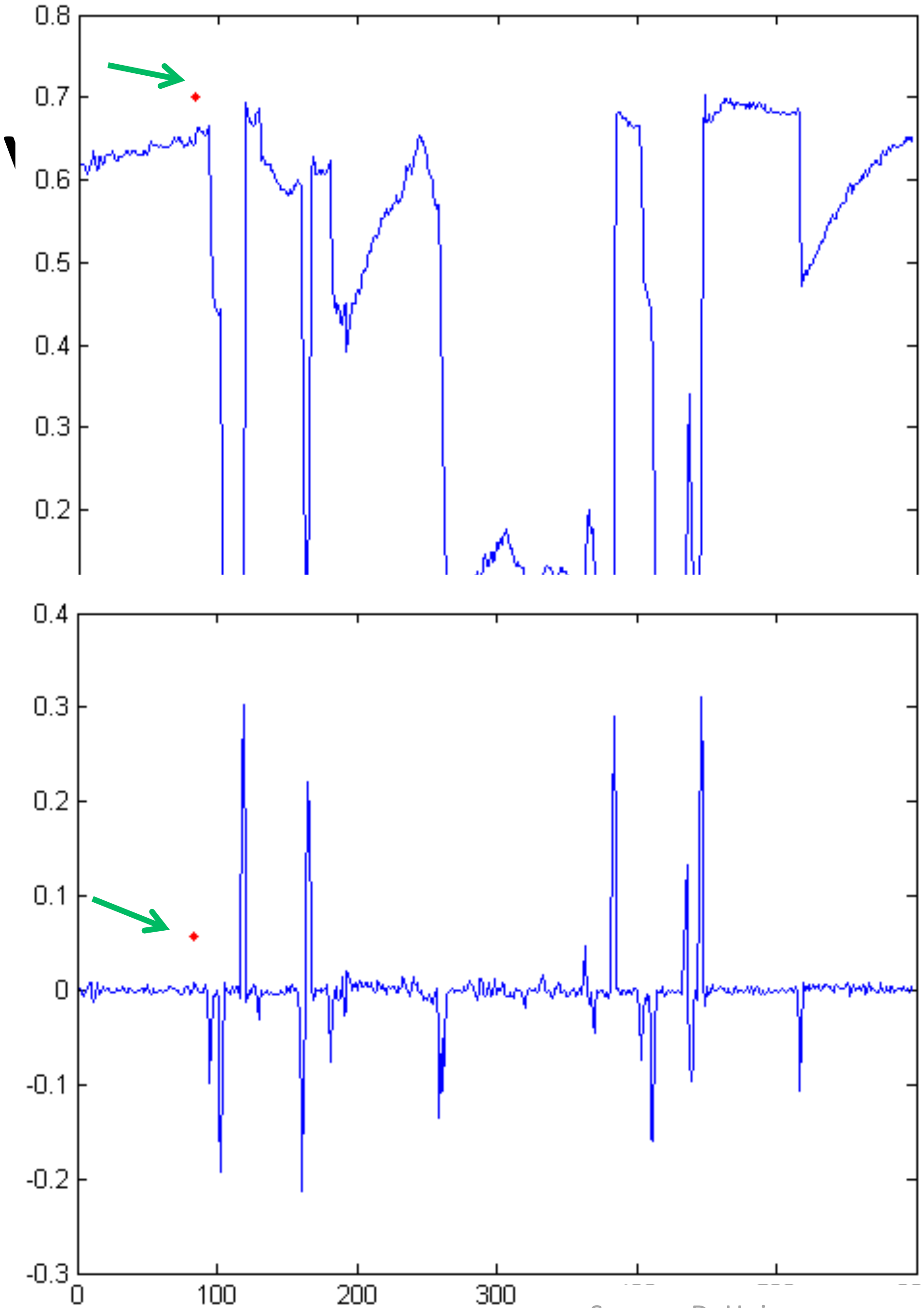


edges correspond to extrema of derivative

Source: L. Lazebnik

Intensit





Source: D. Hoiem

Image derivatives

- How can we differentiate a *digital* image F[x,y]?
  - Option 1: reconstruct a continuous image, *f,* then compute the derivative
  - Option 2: take discrete derivative (finite difference)

$$\frac{\partial f}{\partial x}[x, y] \approx F[x+1, y] - F[x, y]$$

How would you implement this as a linear filter?

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

H_x

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

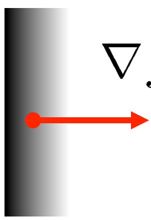
H_y

Source: S. Seitz

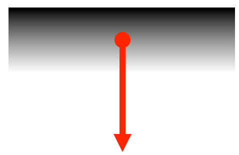
Image gradient

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

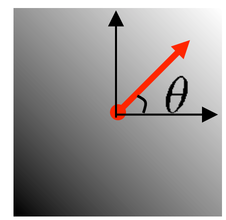
The gradient points in the direction of most rapid increase in intensity



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



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



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

The *edge strength* is given by the gradient magnitude:

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

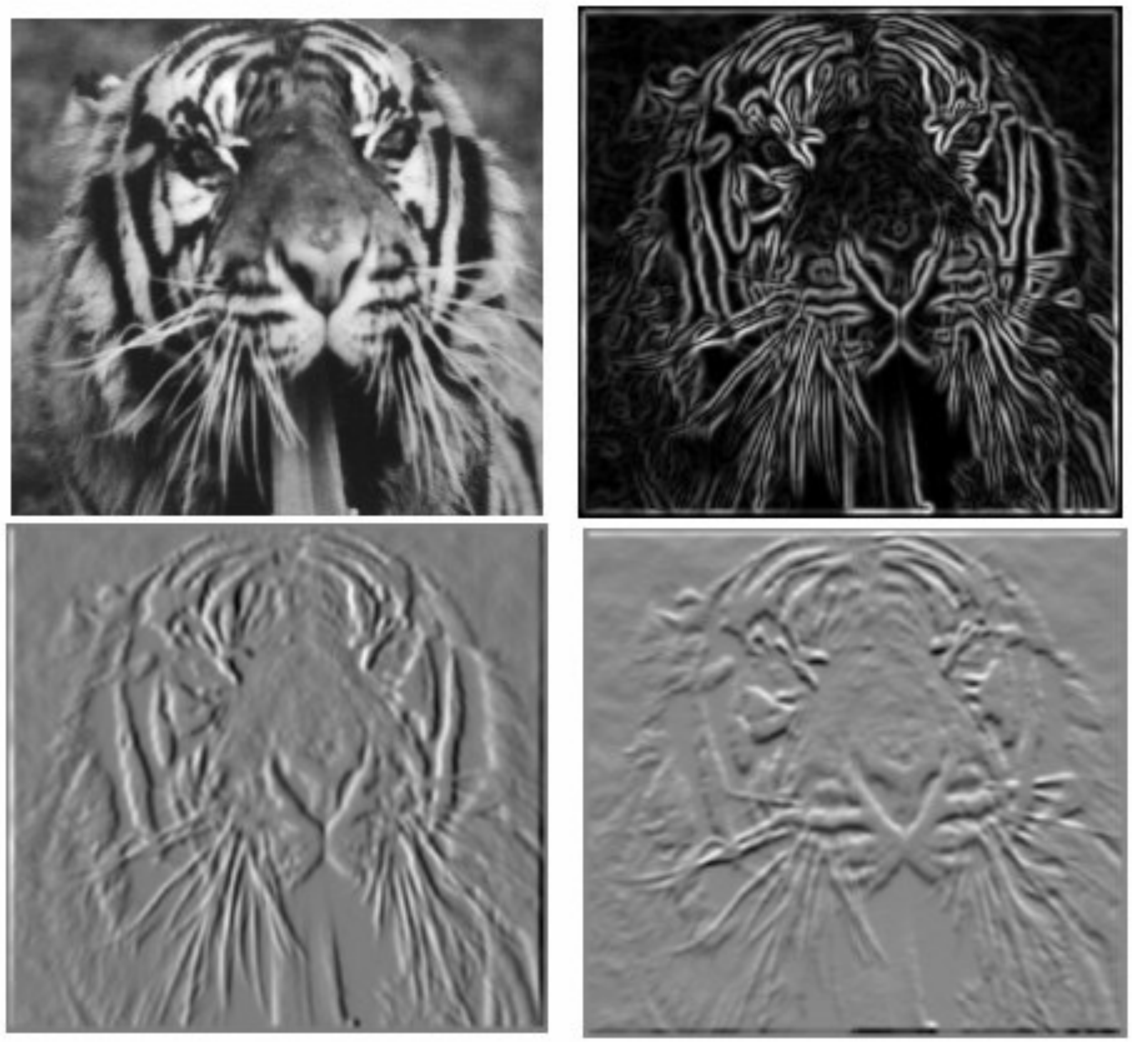
The gradient direction is given by:

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

- how does this relate to the direction of the edge?

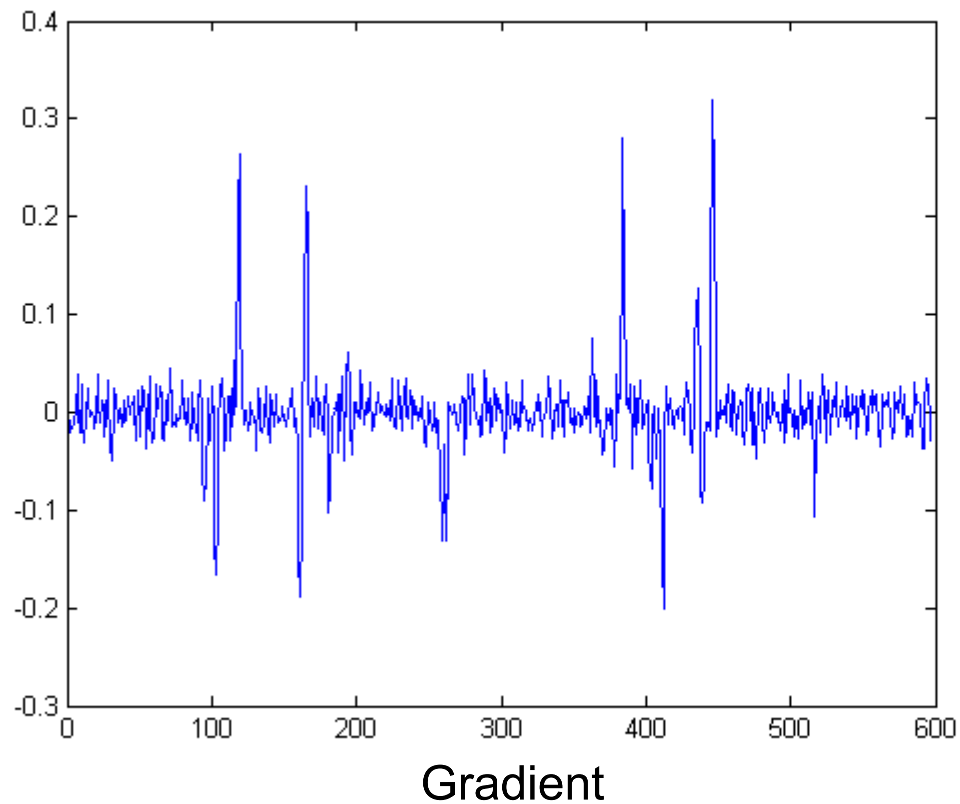
Source: Steve Seitz

Image gradient



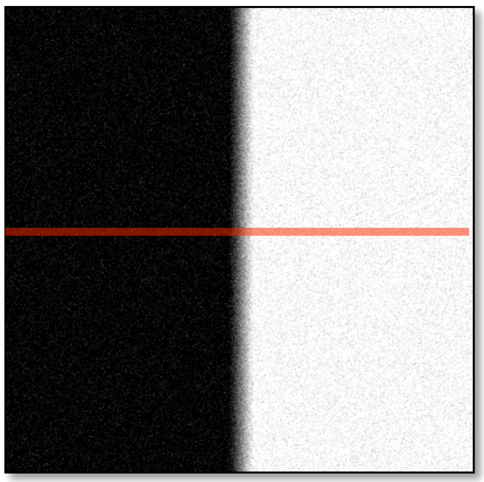
Source: L. Lazebnik

With a little Gaussian noise

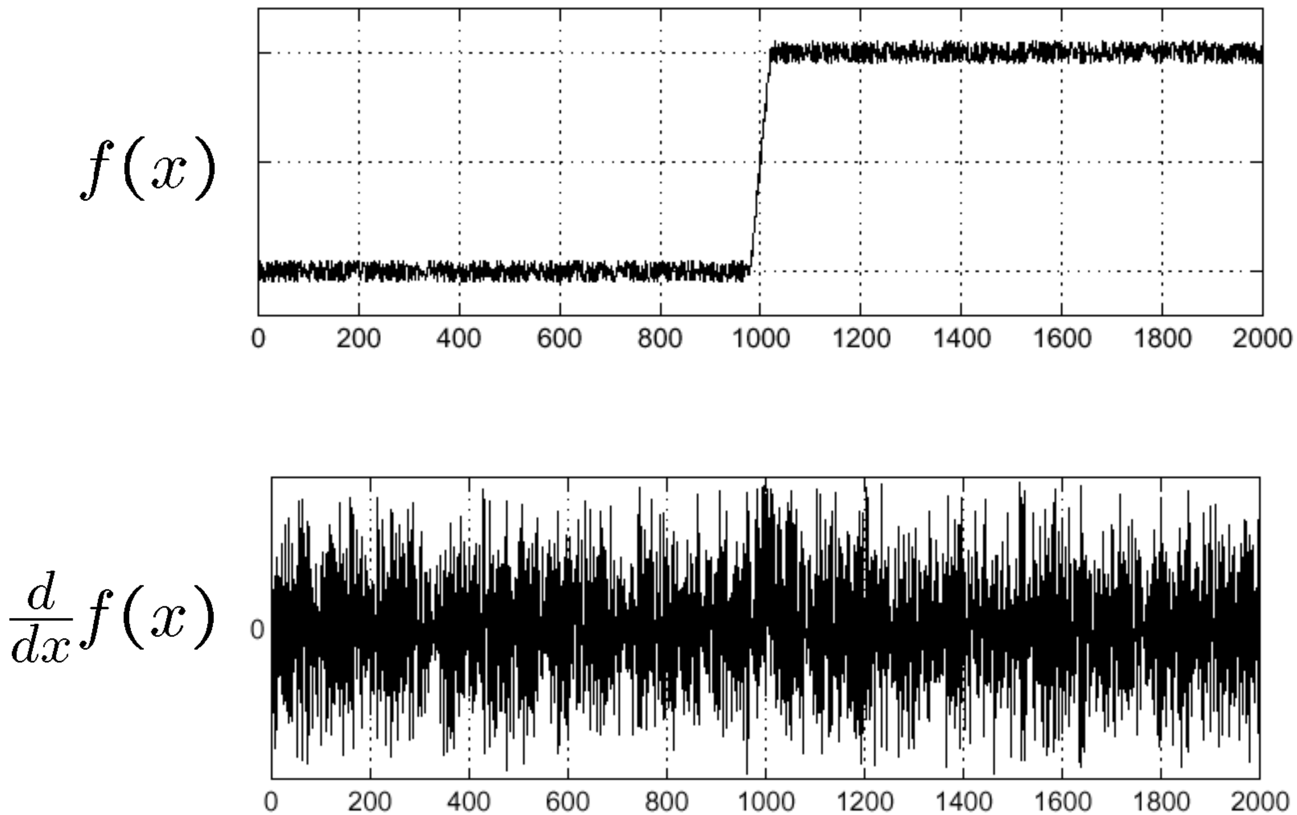


Source: D. Hoiem

Effects of noise



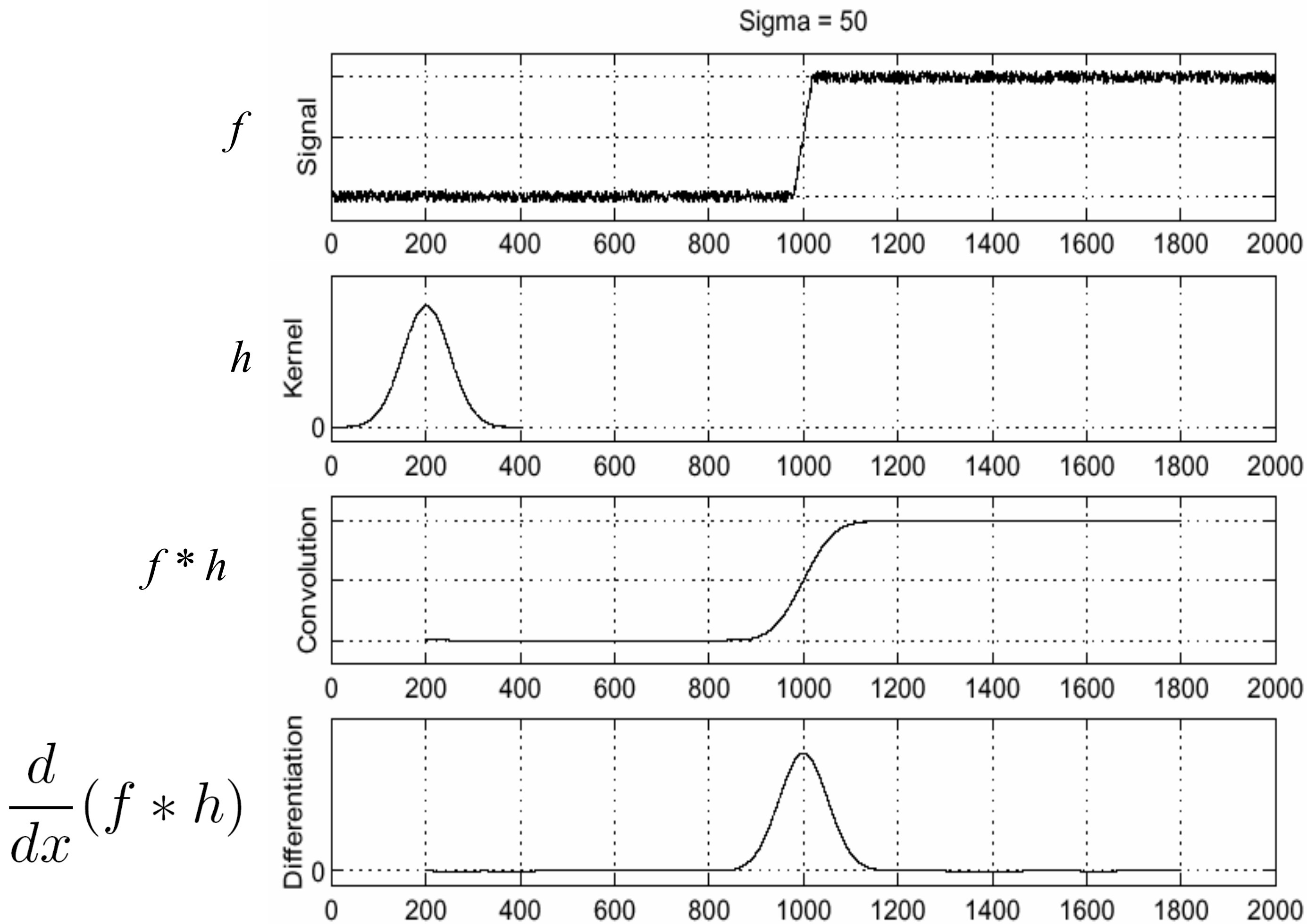
Noisy input image



Where is the edge?

Source: S. Seitz

Solution: smooth first



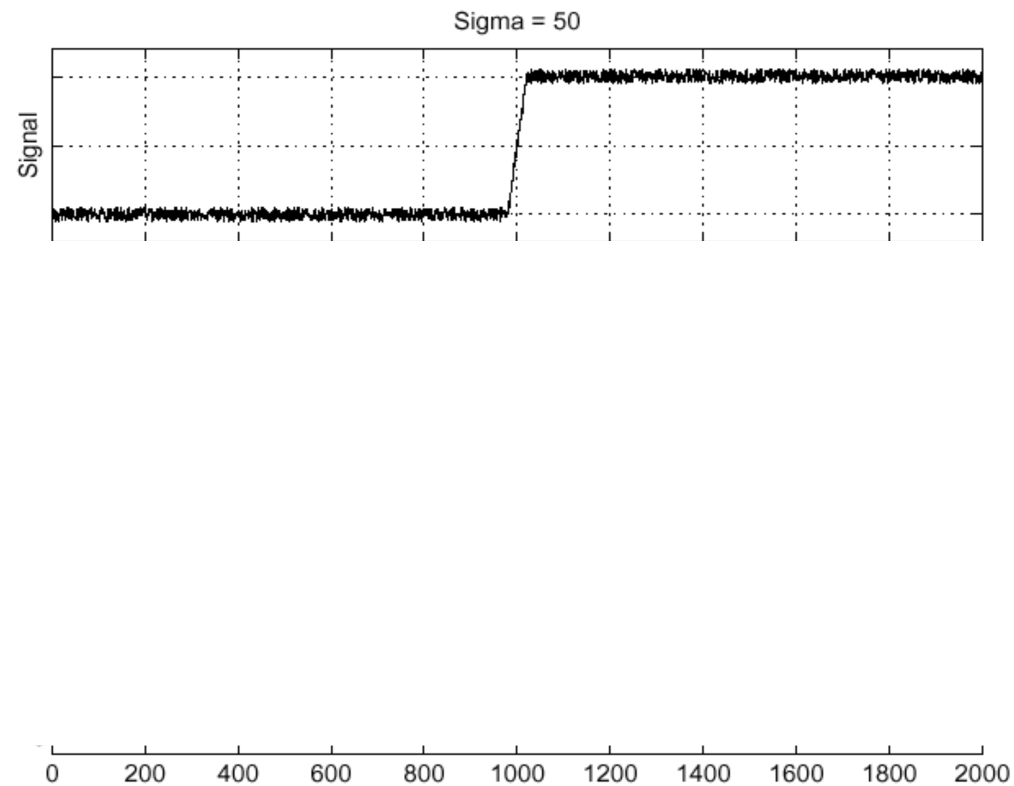
To find edges, look for peaks in $\frac{d}{dx}(f * h)$

Source: S. Seitz

Associative property of convolution

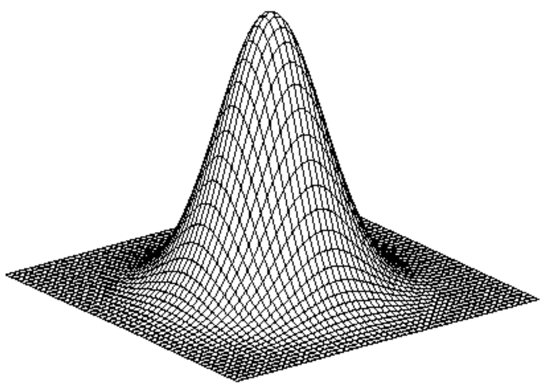
- Differentiation is a convolution
- Convolution is associative: $\frac{d}{dx}(f * h) = f * \frac{d}{dx}h$
- This saves us one operation:

f



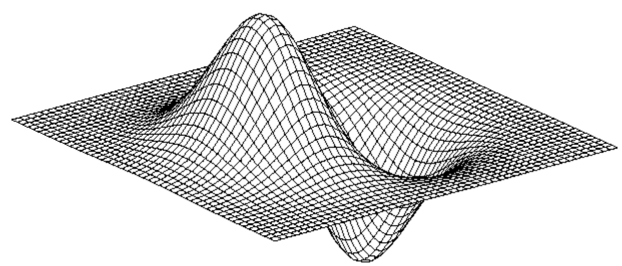
Source: S. Seitz

2D edge detection filters



Gaussian

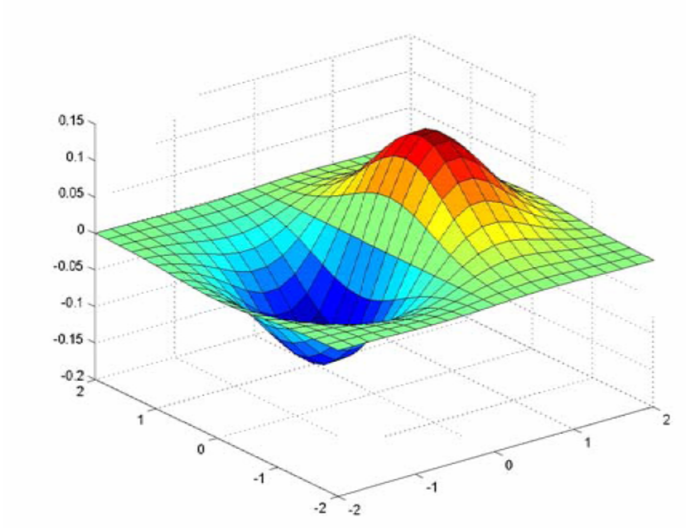
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



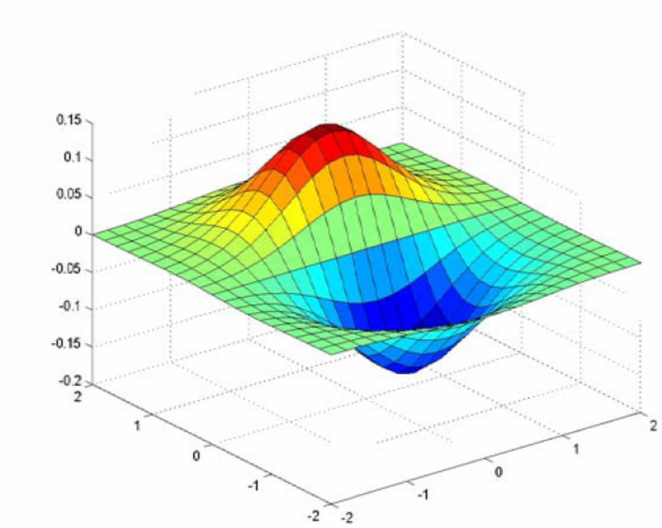
derivative of Gaussian (x)

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

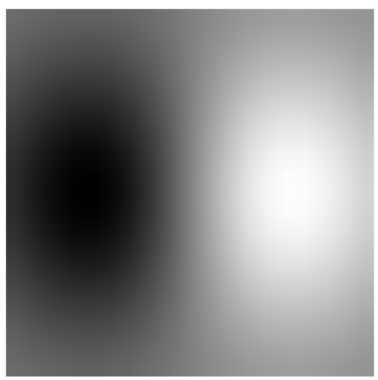
Derivative of Gaussian filter

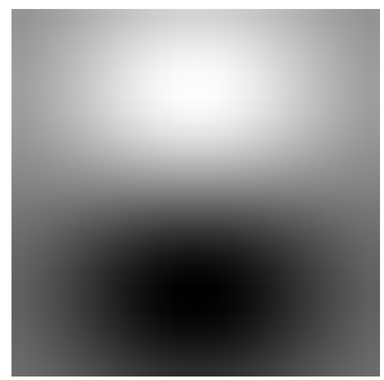


x-direction



y-direction





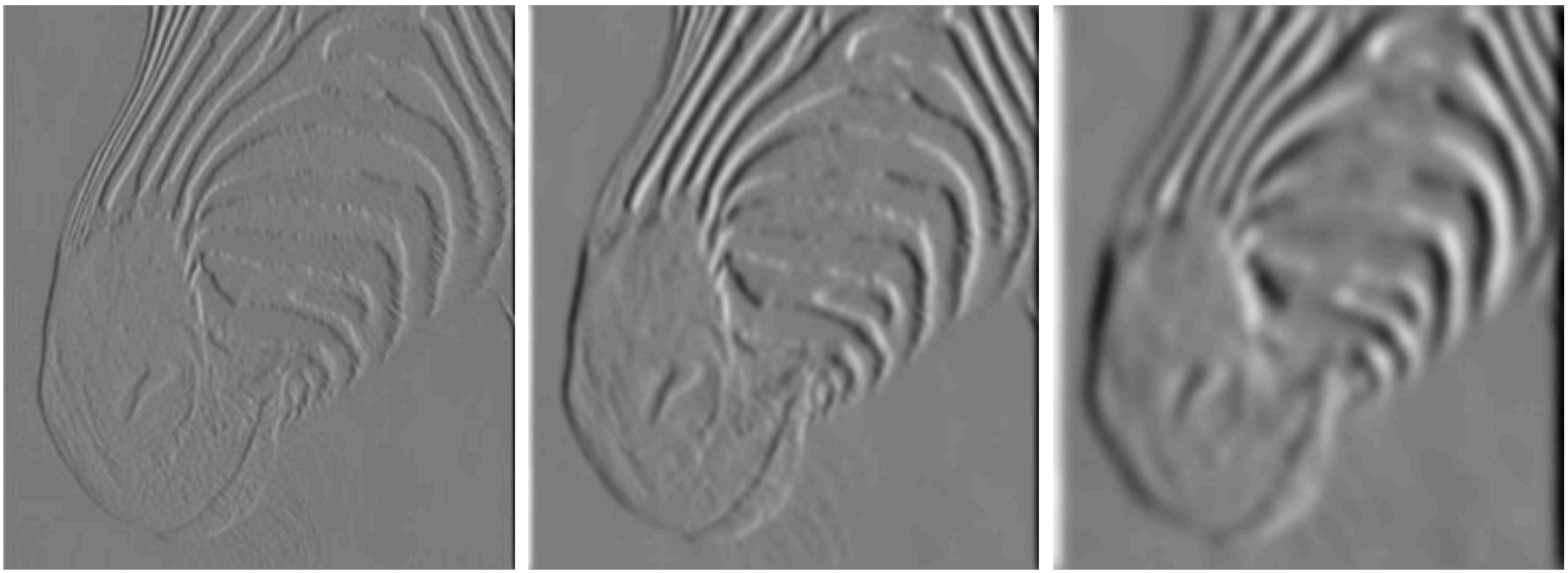


FIGURE 5.3: The scale (i.e., σ) of the Gaussian used in a derivative of Gaussian filter has significant effects on the results. The three images show estimates of the derivative in the x direction of an image of the head of a zebra obtained using a derivative of Gaussian filter with σ one pixel, three pixels, and seven pixels (**left** to **right**). Note how images at a finer scale show some hair, the animal's whiskers disappear at a medium scale, and the fine stripes at the top of the muzzle disappear at the coarser scale.

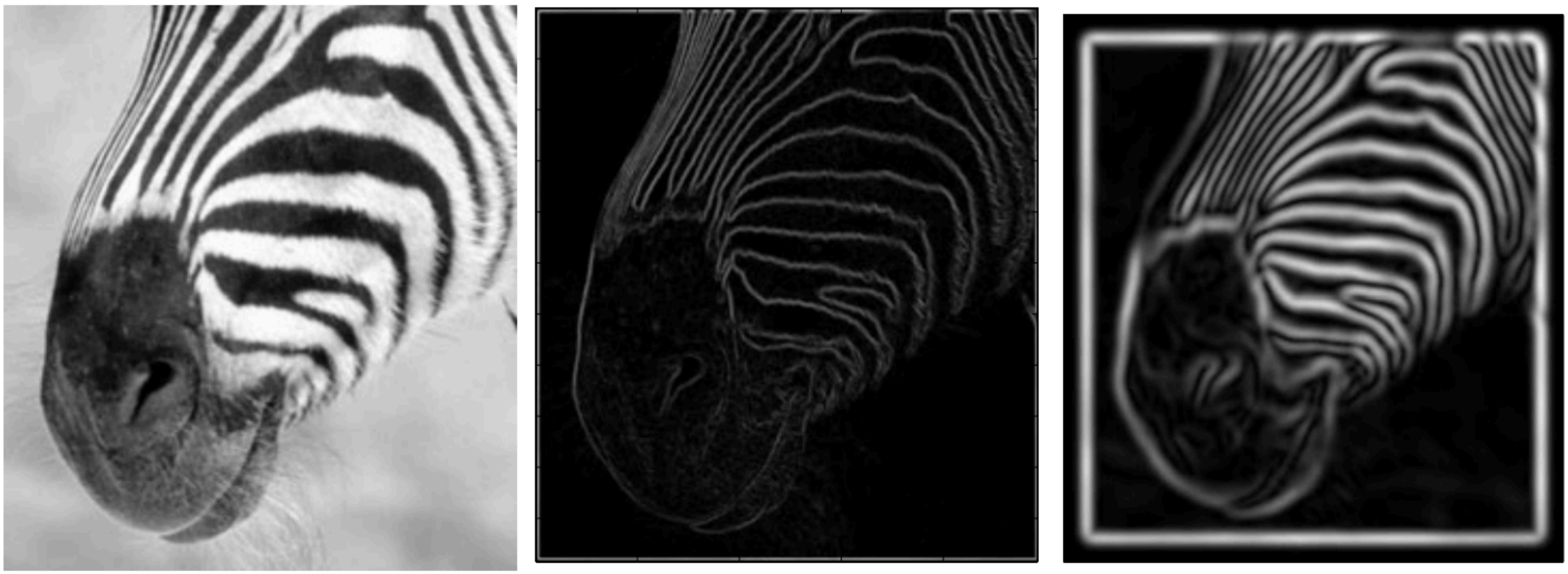


FIGURE 5.4: The gradient magnitude can be estimated by smoothing an image and then differentiating it. This is equivalent to convolving with the derivative of a smoothing kernel. The extent of the smoothing affects the gradient magnitude; in this figure, we show the gradient magnitude for the figure of a zebra at different scales. At the **center**, gradient magnitude estimated using the derivatives of a Gaussian with $\sigma = 1$ pixel; and on the **right**, gradient magnitude estimated using the derivatives of a Gaussian with $\sigma = 2$ pixel. Notice that large values of the gradient magnitude form thick trails.