# Edges

Nuno Vasconcelos

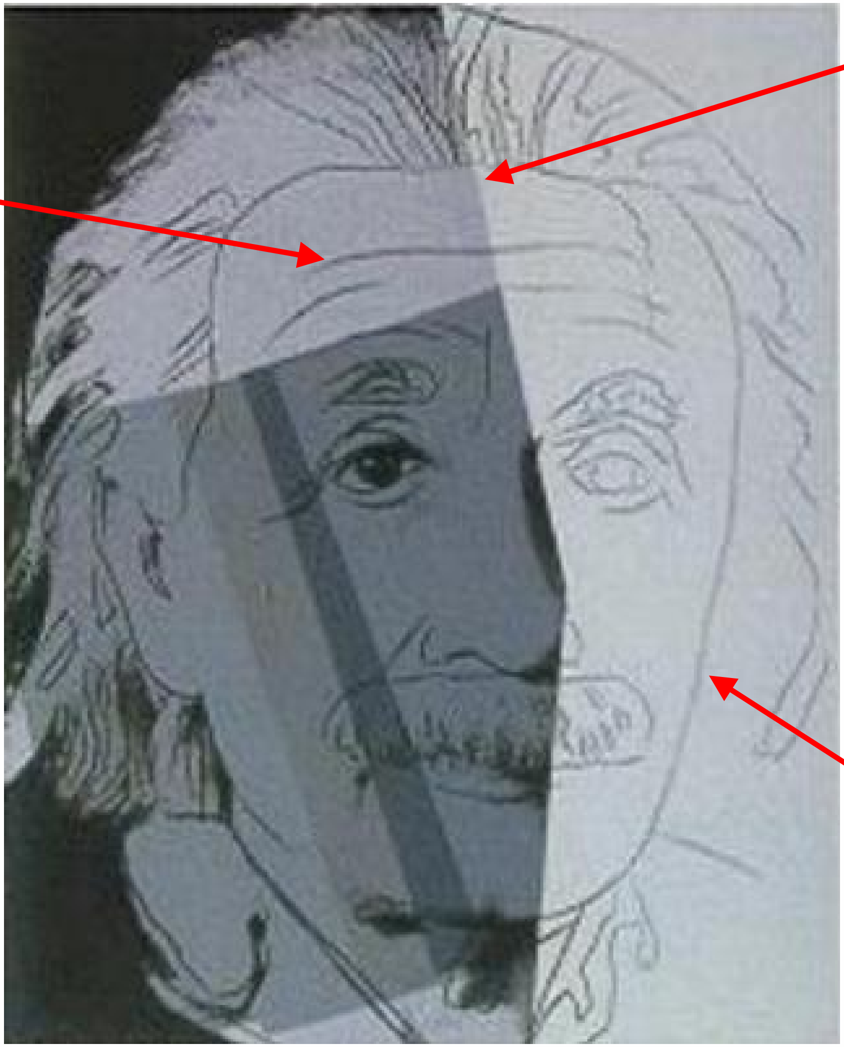
*ECE Department, UCSD*

*(with thanks to David Forsyth)*

# Gradients and edges

- for image understanding, one of the problems is that there is too much information in an image
- just smoothing is not good enough
- how to detect important (most informative) image points?
- note that derivatives are large at points of great change
  - changes in reflectance (e.g. checkerboard pattern)
  - change in object (an object boundary is different from background)
  - change in illumination (the boundary of a shadow)
- these are usually called **edge points**
- detecting them could be useful for various problems
  - segmentation: we want to know what are object boundaries
  - recognition: cartoons are easy to recognize and terribly efficient to transmit

# The importance of edges



# Gradients

- for a 2D function, *f(x,y)* the gradient at a point $(x_0,y_0)$

$$\nabla f(x_0,y_0) = \left(\frac{\partial f}{\partial x}(x_0,y_0), \frac{\partial f}{\partial y}(x_0,y_0)\right)^T$$
$$= \left(f_x(x_0,y_0),\ f_y(x_0,y_0)\right)^T$$

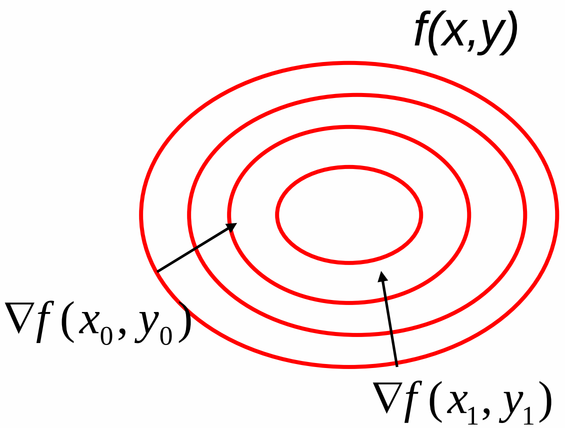
  is the direction of greatest increase at that point

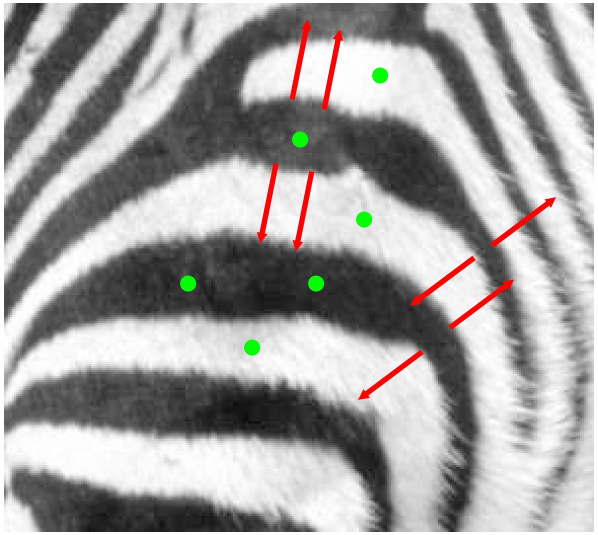
- the gradient magnitude

$$\|\nabla f(x_0,y_0)\|^2 = \left(\frac{\partial f}{\partial x}(x_0,y_0)\right)^2 + \left(\frac{\partial f}{\partial y}(x_0,y_0)\right)^2$$

  measures the rate of change

- it is large at edges!



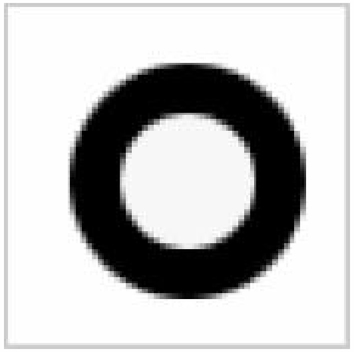
– large gradient magnitude

– small gradient magnitude

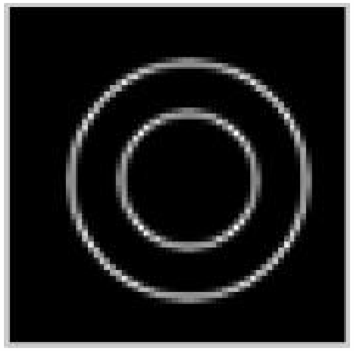
# Gradients

- here is an example

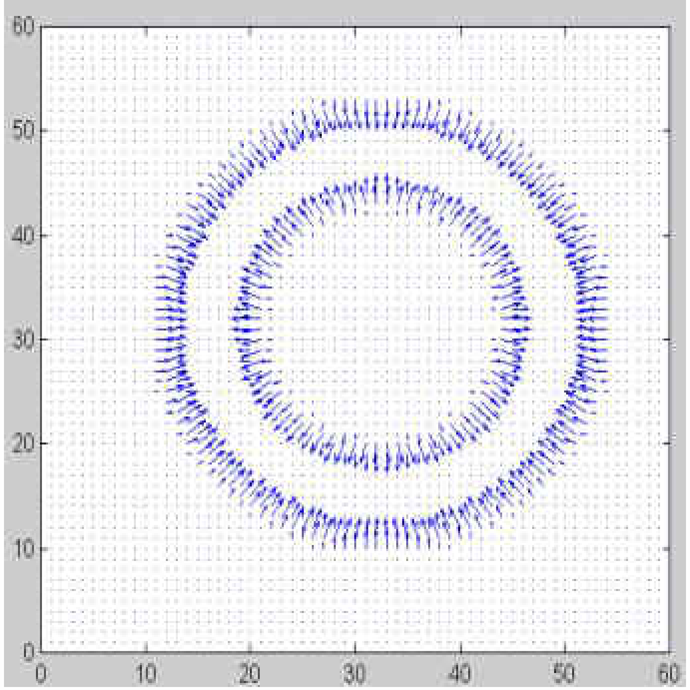
image



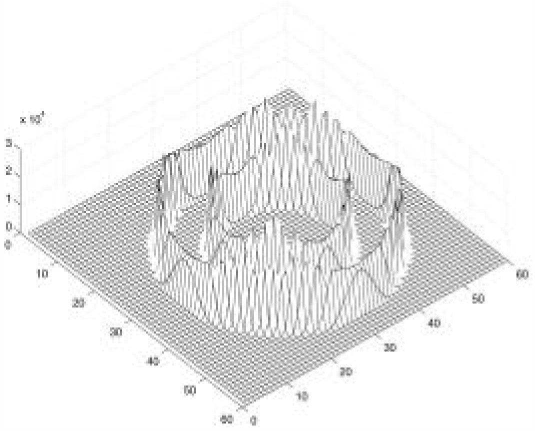
grad. magnitude



gradient




gradient magnitude



# Derivatives and convolution

- recall that a derivative is defined as

$$\frac{\partial f(x)}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

- linear and shift invariant, so must be the result of a convolution.
- we could approximate as

$$\frac{\partial f(n)}{\partial n} = \frac{f(n+1) - f(n)}{1} = f(n+1) - f(n) = f * h(n)$$

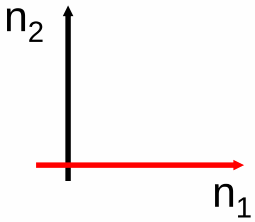
- where the derivative kernel is

$$h(n) = \delta(n+1) - \delta(n)$$

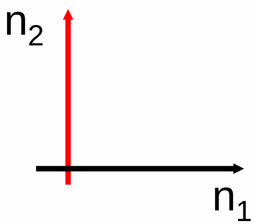
# Finite difference kernels

- in two dimensions we have various possible kernels
- e.g. , $N_1=2, N_2=3$, derivative along $n_1$, *(line $n_2=k$)* (horizontal)

$$\begin{matrix} 0 & 0 \\ 1 & -1 \\ 0 & 0 \end{matrix} \qquad\qquad \begin{matrix} 1 & -1 \\ 1 & -1 \\ 1 & -1 \end{matrix}$$

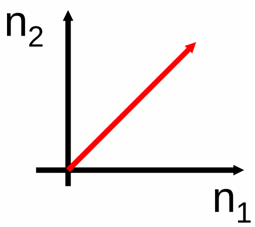



- derivative along $n_{2,}$ *(line $n_1=k$)* (vertical)




$$\begin{matrix} 0 & -1 & 0 \\ 0 & 1 & 0 \end{matrix} \qquad\qquad \begin{matrix} -1 & -1 & -1 \\ 1 & 1 & 1 \end{matrix}$$

- derivative along line $n_1=n_2$ (diagonal)




$$\begin{matrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{matrix} \qquad\qquad \begin{matrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{matrix}$$

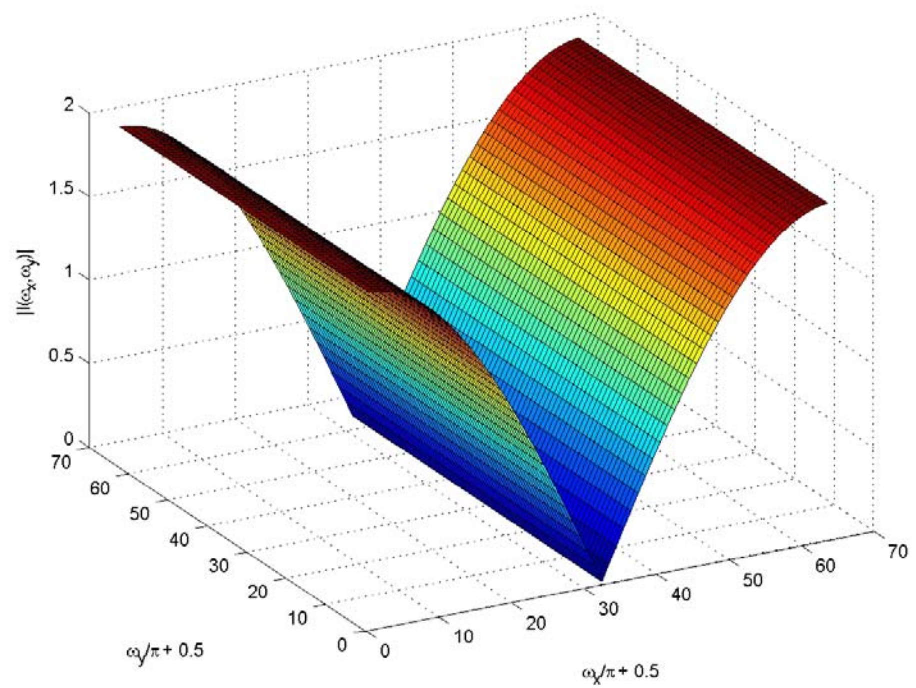
# Finite difference kernels

- note that, when

$$h(n_1,n_2)= \begin{matrix} 0 & 0 \\ 1 & -1 \\ 0 & 0 \end{matrix}$$

- we have

$$\begin{aligned} H(\varpi_1,\varpi_2) &= e^{j\varpi_1} - 1 \\ &= \left( e^{j\frac{\varpi_1}{2}} - e^{-j\frac{\varpi_1}{2}} \right) e^{j\frac{\varpi_1}{2}} \\ &= 2j\sin\left(\frac{\varpi_1}{2}\right) e^{j\frac{\varpi_1}{2}} \end{aligned}$$




- derivative is a high-pass filter
- hw: check that this holds for all others
- intuitive, because a derivative is a measure of the rate of change of a function

# Finite differences

- Q: which one do we have here? (gray=0,white=+,dark=-)

# Finite differences and noise

- because they perform high-pass filtering, finite difference filters respond strongly to noise
- generally, the larger the noise the stronger the response
- for noisy images it is usually best to apply some smoothing before computing derivatives
- what do mean by noise?
- we only consider the simplest model
  - independent stationary additive Gaussian noise
  - the noise value at each pixel is given by an independent draw from the same normal probability distribution

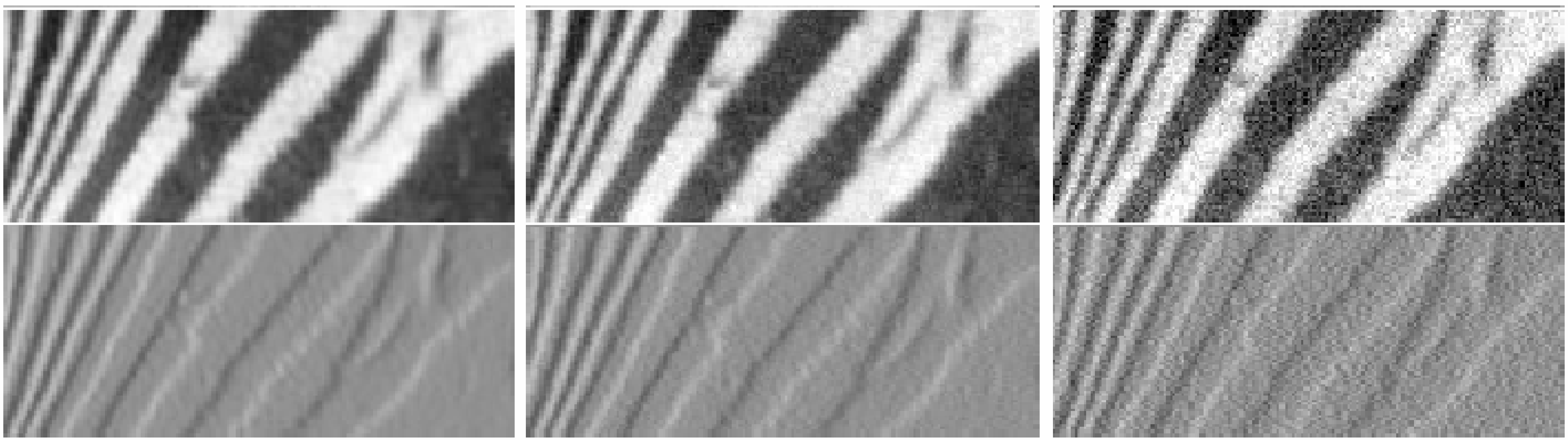
$$y(n_1,n_2) = x(n_1,n_2) + \varepsilon(n_1,n_2), \qquad \varepsilon \sim N(0,\sigma^2)$$

sigma=1

sigma=16

# Finite differences responding to noise

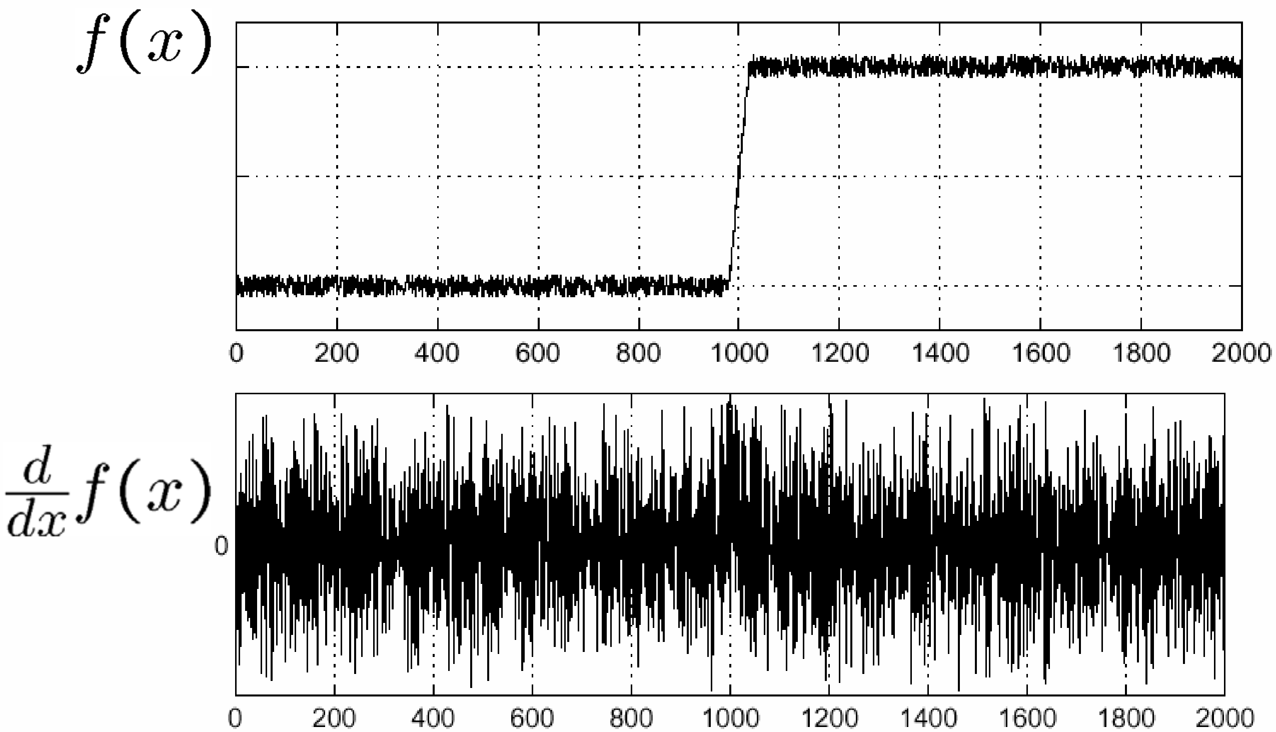
increasing noise variance



- note that as the noise variance increases the estimates of the image derivative are also very noisy

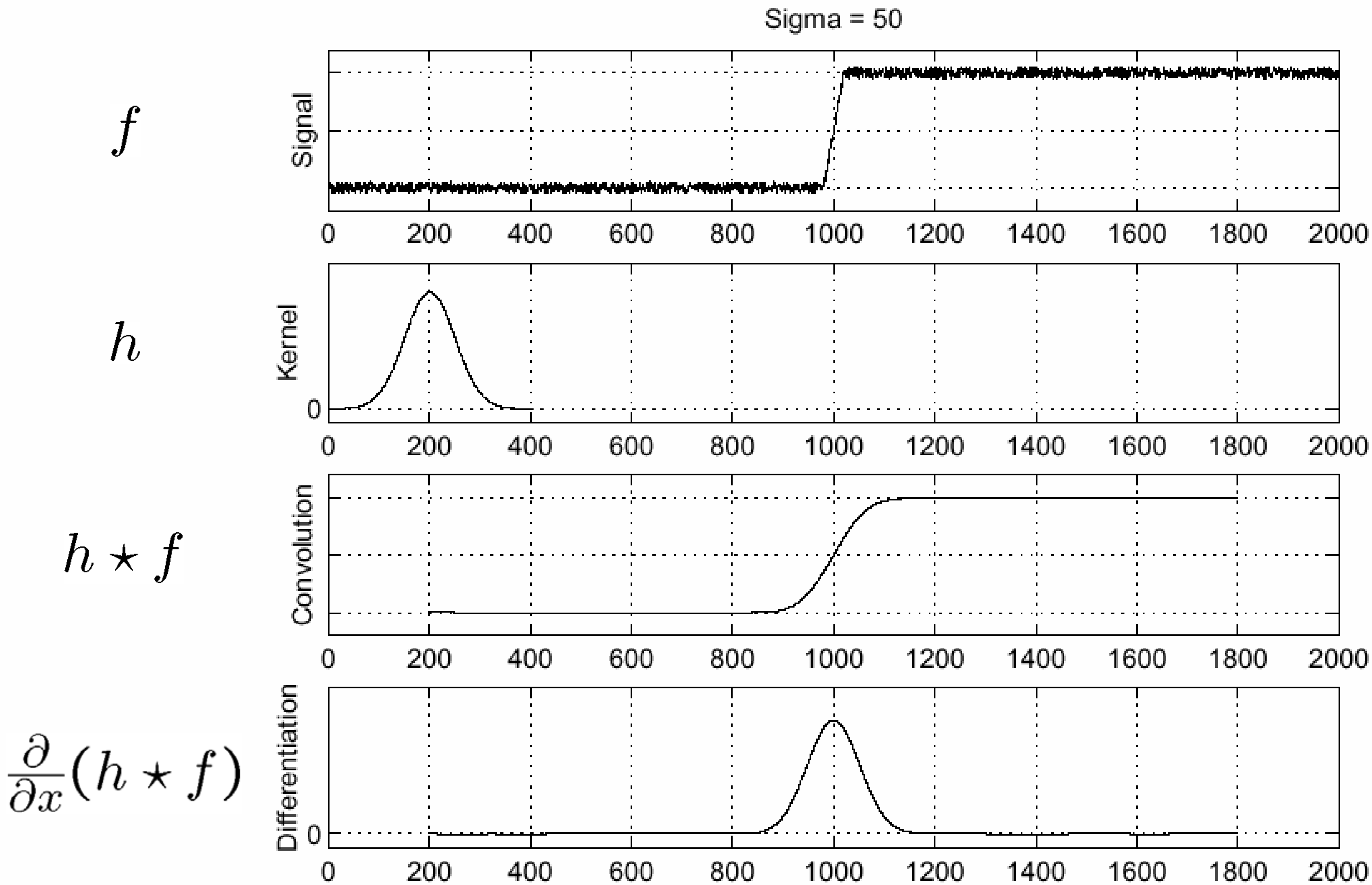
# Effects of noise

- this can be seen even in 1D
  - consider a single row or column of the image
  - plotting intensity as a function of position gives a signal




- where is the edge?

# Solution: smooth first




- Where is the edge?
- Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

# Smoothing reduces noise

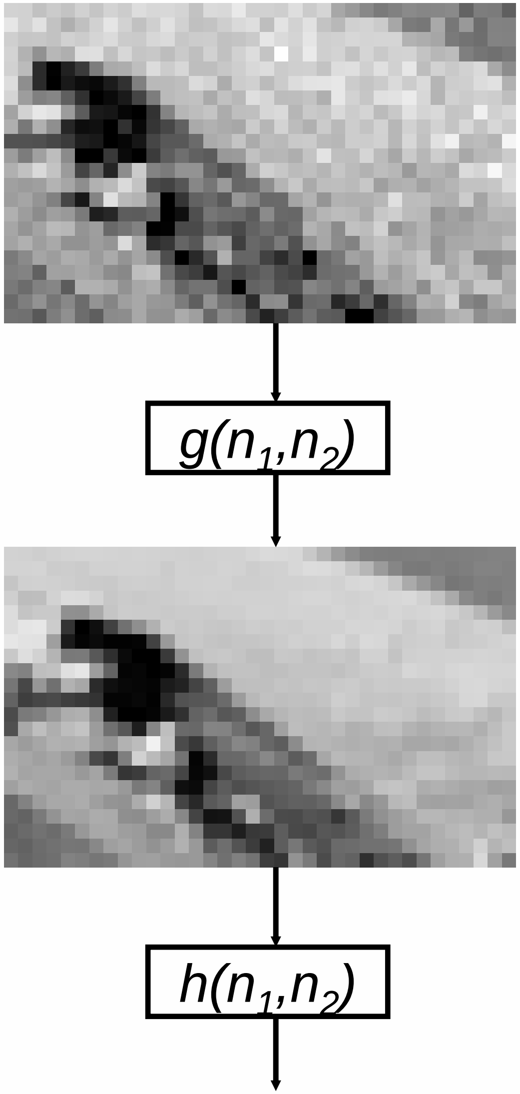
- noise has a lot of high-frequencies
- strategy:
  1. start by low-pass filtering, to suppress noise
  2. compute derivative on smoothed image
- i.e. for a smoothing filter $g(n_1,n_2)$ compute

$$h*\left(g*x\right)$$

- note that, by associativity of convolution, this is equal to

$$\left(h*g\right)*x$$

- i.e. filter the image with the filter $h*g$




smoothed derivatives

# Derivative theorem of convolution

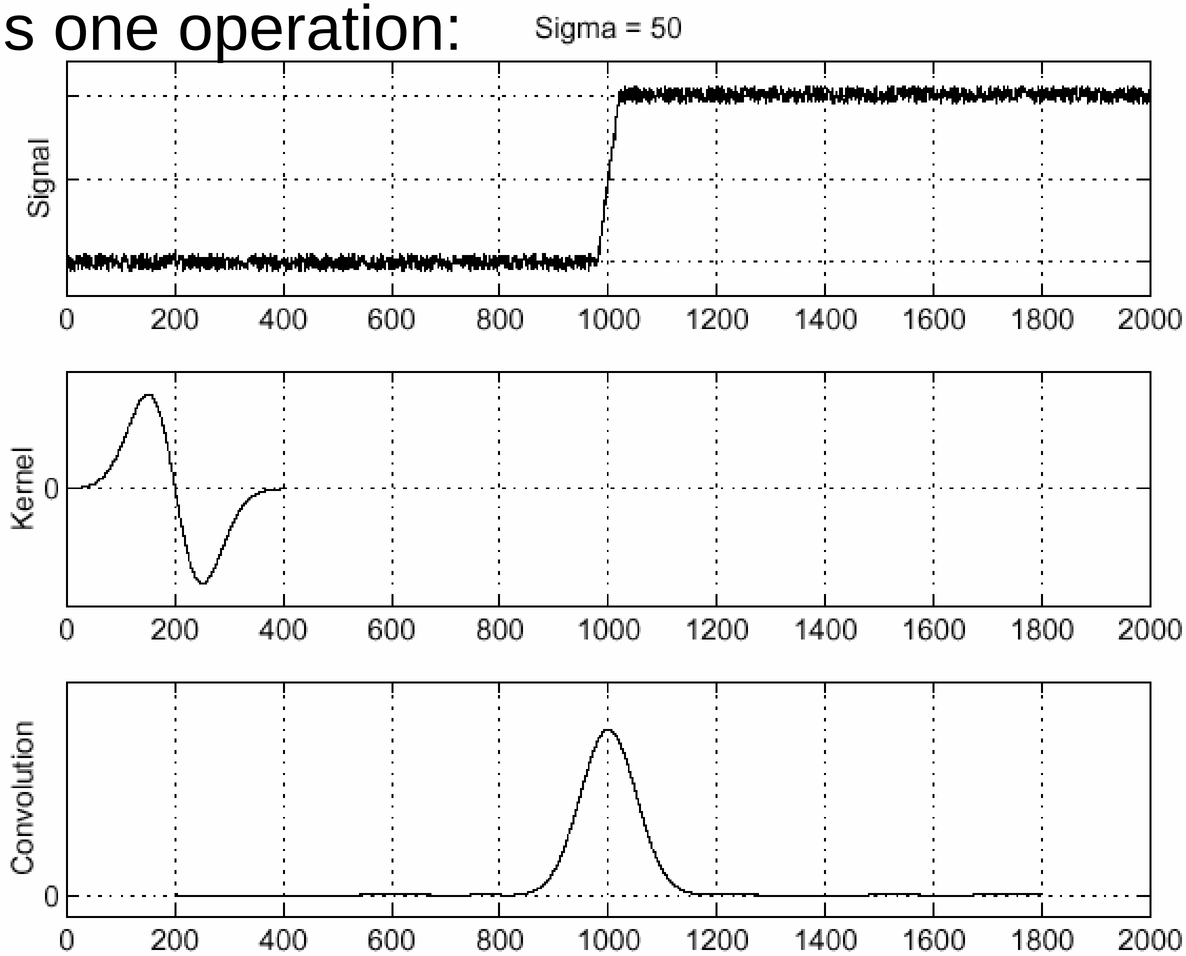
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

- This saves us one operation:

$f$

$\frac{\partial}{\partial x}h$

$(\frac{\partial}{\partial x}h) \star f$



# The derivative of a Gaussian

- let's consider, for example,

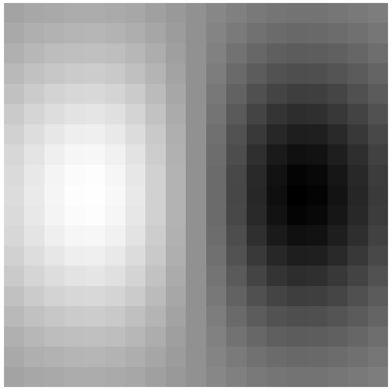
$$h(n_1, n_2) = \delta(n_1 + 1, n_2) - \delta(n_1, n_2)$$

- in which case

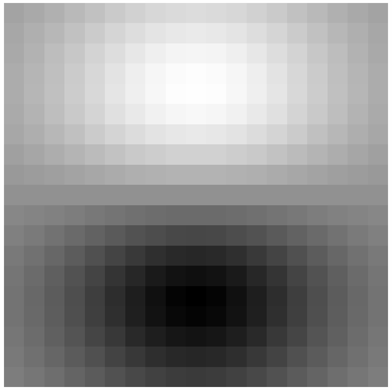
$$h * g(n_1, n_2) = g(n_1 + 1, n_2) - g(n_1, n_2)$$

  is a difference of two Gaussians
- this is the derivative of a Gaussian (DoG) filter
- for other definitions of *h* we have a similar result

DoG along $n_1$

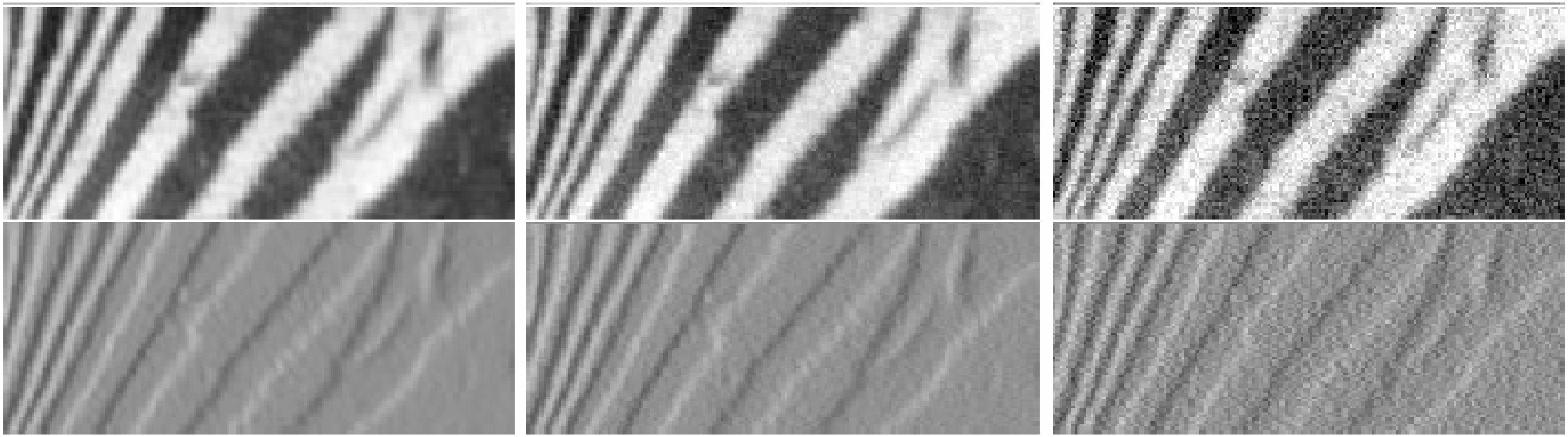


DoG along $n_2$

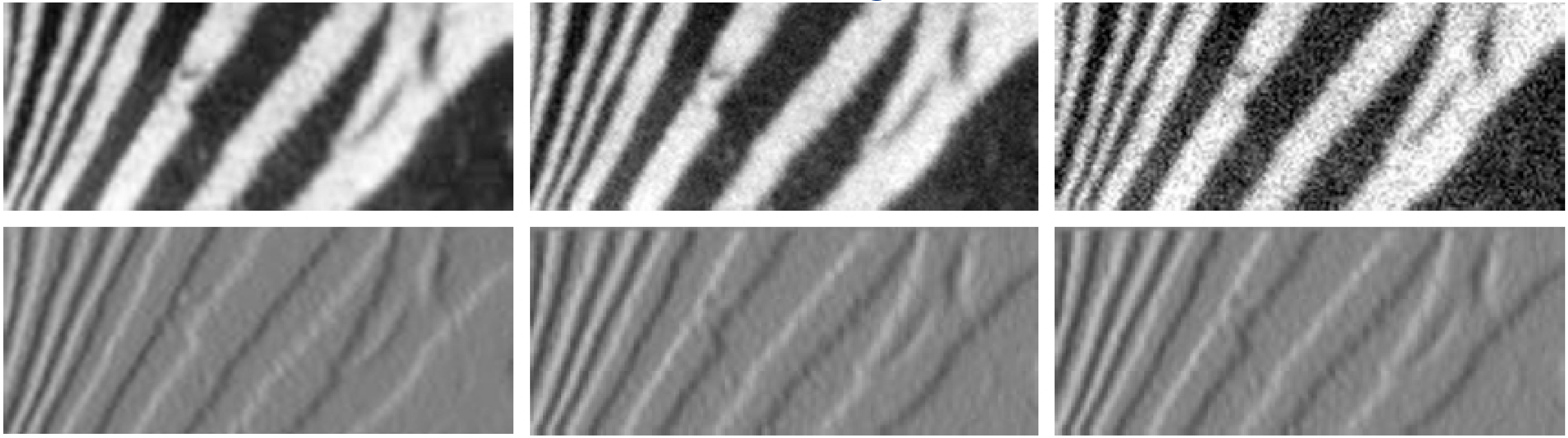


# Smoothed derivatives

no smoothing



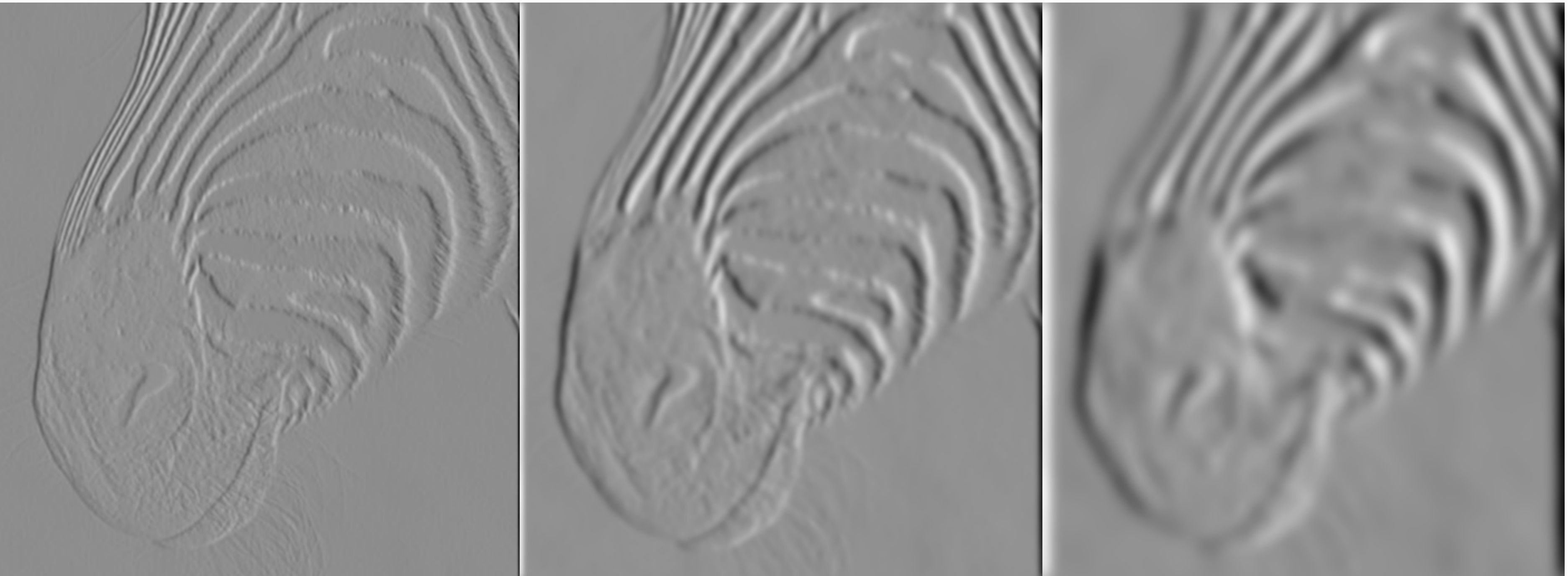
with smoothing



# Choosing the right scale

- the scale of the smoothing filter affects
  - derivative estimates,
  - the semantics of the derivative image
- trade-off between noise and ability to detect detail

1 pixel 3 pixels 7 pixels

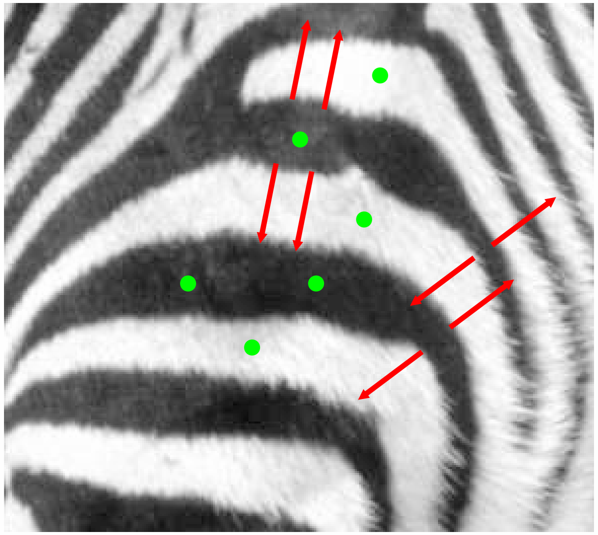


# Gradients and edges

- in general the optimal amount of smoothing depends on
  - how noisy image is
  - how much detail we want to preserve
- remember that edges are points of large gradient magnitude
- edge detection strategy
  1. determine magnitude of image gradient

$$\left\|\nabla f(x_0, y_0)\right\|^2 = \left(\frac{\partial f}{\partial x}(x_0, y_0)\right)^2 + \left(\frac{\partial f}{\partial y}(x_0, y_0)\right)^2$$

  2. mark points where gradient magnitude is particularly large wrt neighbours (ideally, curves of such points)

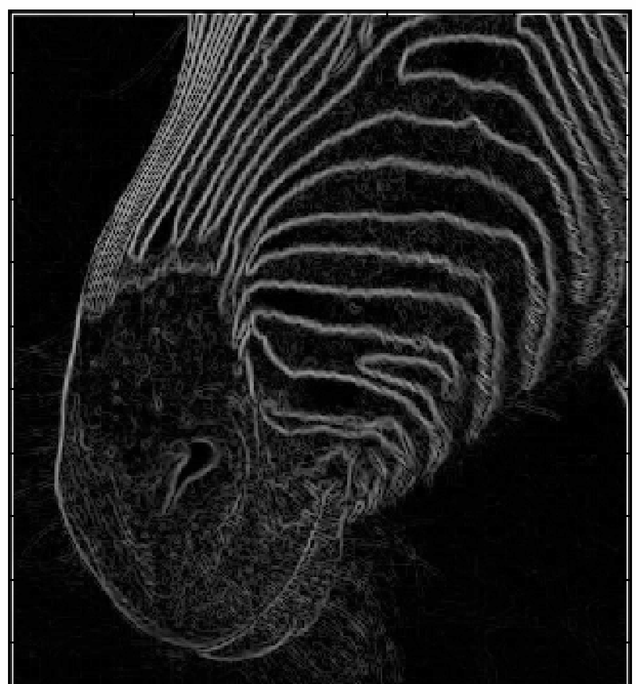
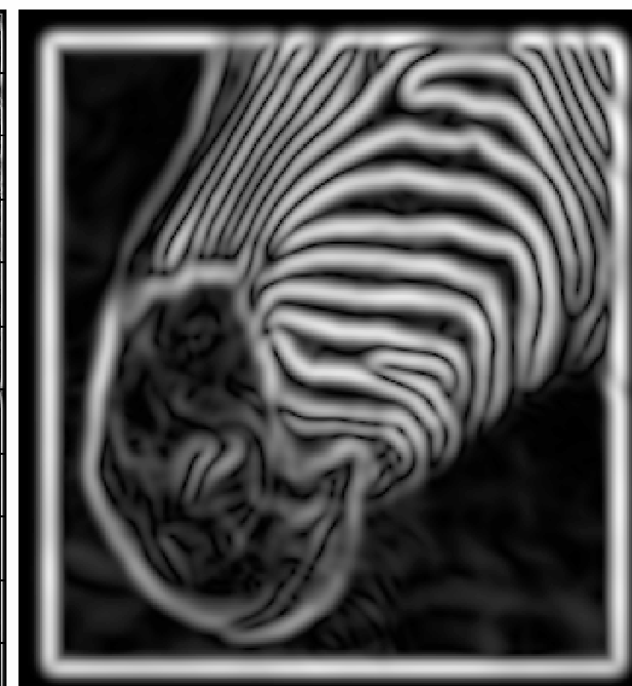


– large gradient magnitude

– small gradient magnitude

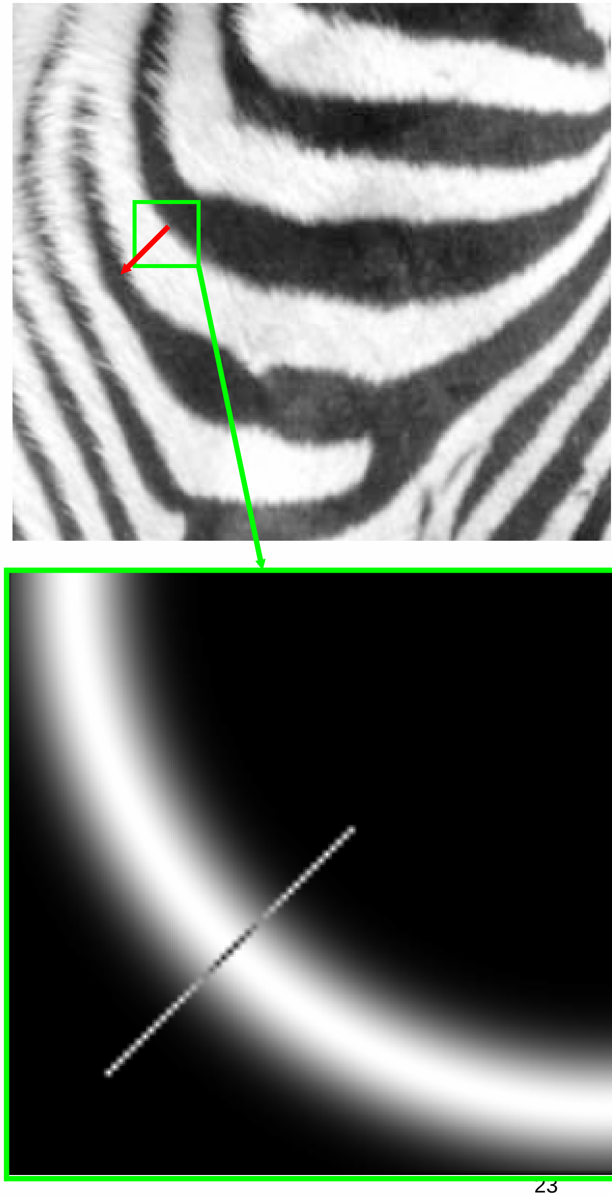
# Detecting edge points

- know how to compute gradient, still three major issues:
  - 1) gradient magnitude at different scales is different (see below); which should we choose?
  - 2) gradient magnitude is large along thick trail; what are the significant points?
  - 3) how do we link the relevant points up into curves?

# Maxima of gradient magnitude

- let's leave scale open for now
- maxima of gradient magnitude:
  - the point to remember is that the gradient is perpendicular to the edge
  - we look for the maximum in the direction of the gradient
  - this is called non-maximum suppression
- two algorithmic issues:
  - at which point is the maximum?
  - where is the next one?
- we next see how the Canny edge detector solves these

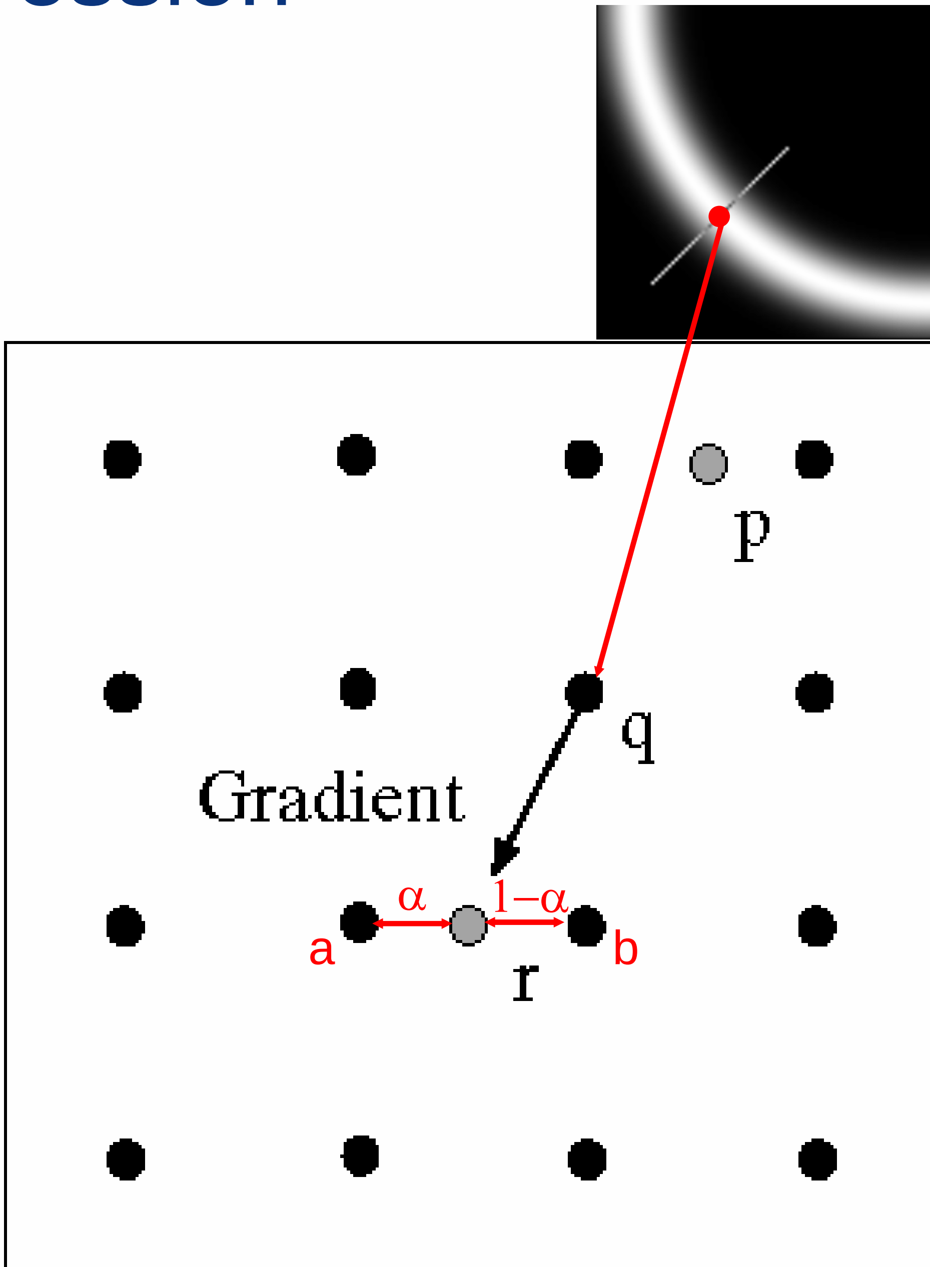


# Non-maximum suppression

- is there a maximum at q?
- yes, if value at q is larger than those at both p and r
- p and r are the pixels in the direction of the gradient that are 1 pixel apart from q
- typically they do not fall in the pixel grid
- we need to interpolate, e.g.

$$r = \alpha\, b + (1-\alpha)\, a$$

- will come back to this later

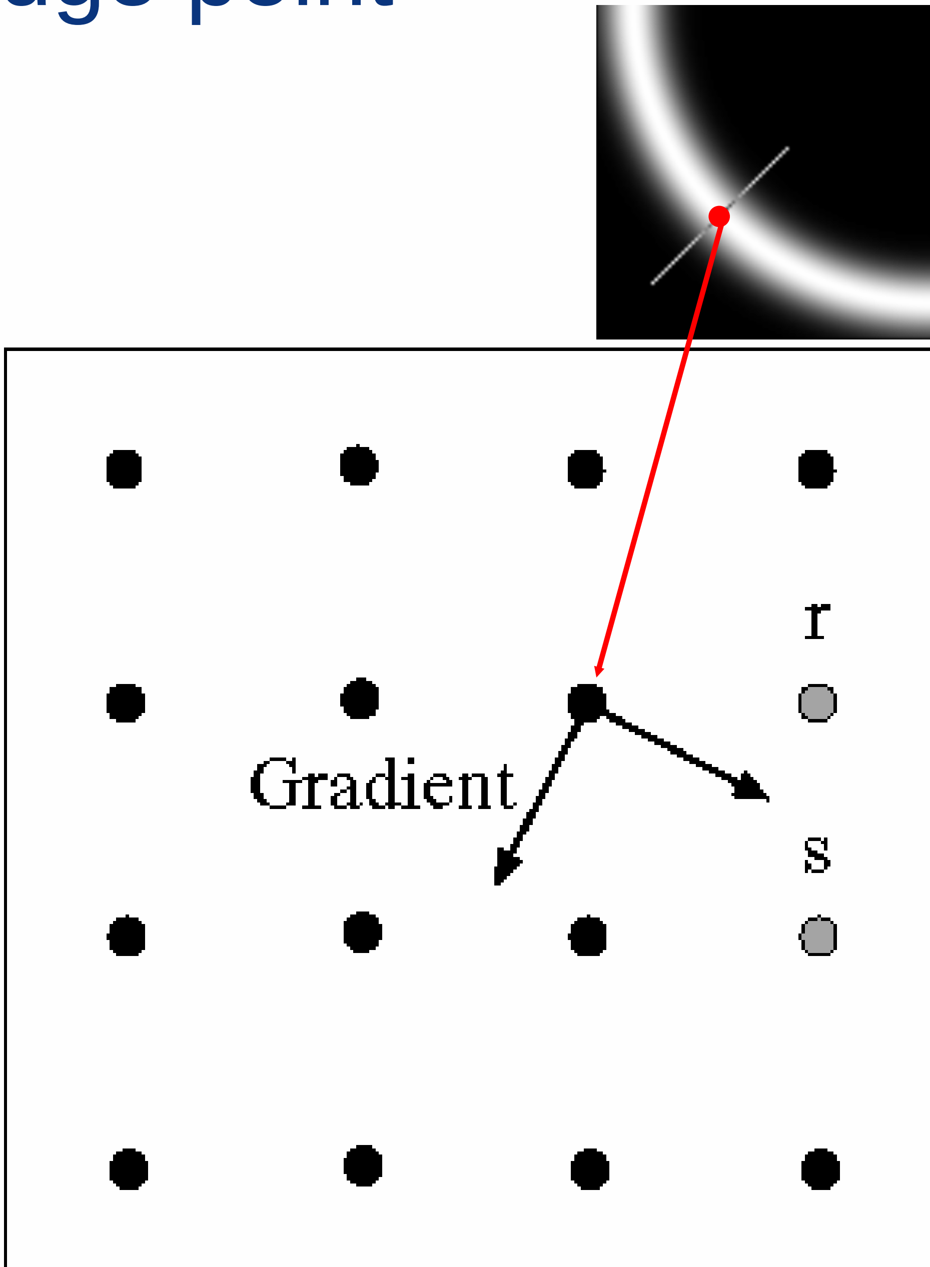


# Predicting the next edge point

- assume the marked point is an edge point
- we construct the tangent to the edge curve (which is normal to the gradient at that point)

$$t(x,y) = \left(-f_y(x,y), f_x(x,y)\right)^T$$

- use this to predict the next points (here either r or s).



# The Canny edge detector

- original image (Lena)

# The Canny edge detector

- norm of the gradient

# The Canny edge detector

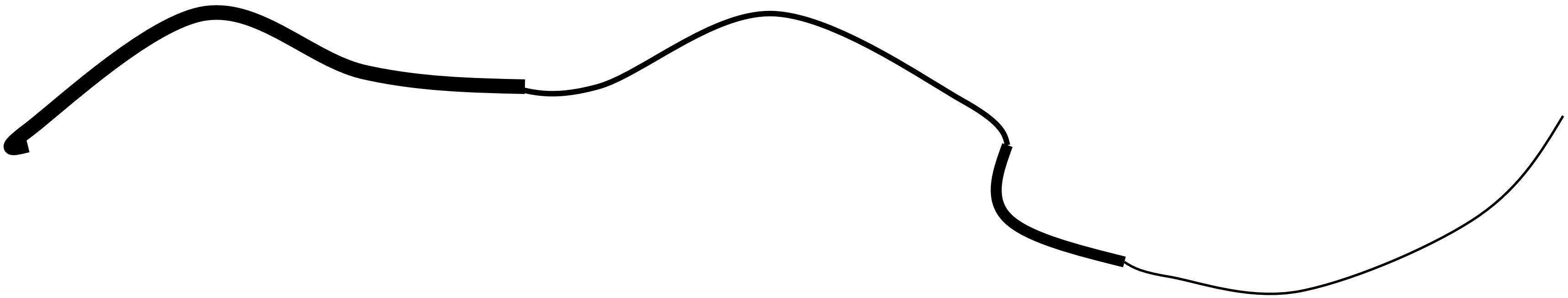
- thinning
- (non-maximum suppression)

# Cleaning up

- even when gradient is ~ zero, there are maxima due to noise
- check that **maximum value of gradient value is large enough** (threshold)
- once we are following an edge we must avoid gaps due to similarity with background
- use **hysteresis**
  - use a high threshold to start edge curves and a low threshold to continue them.

# Hysteresis

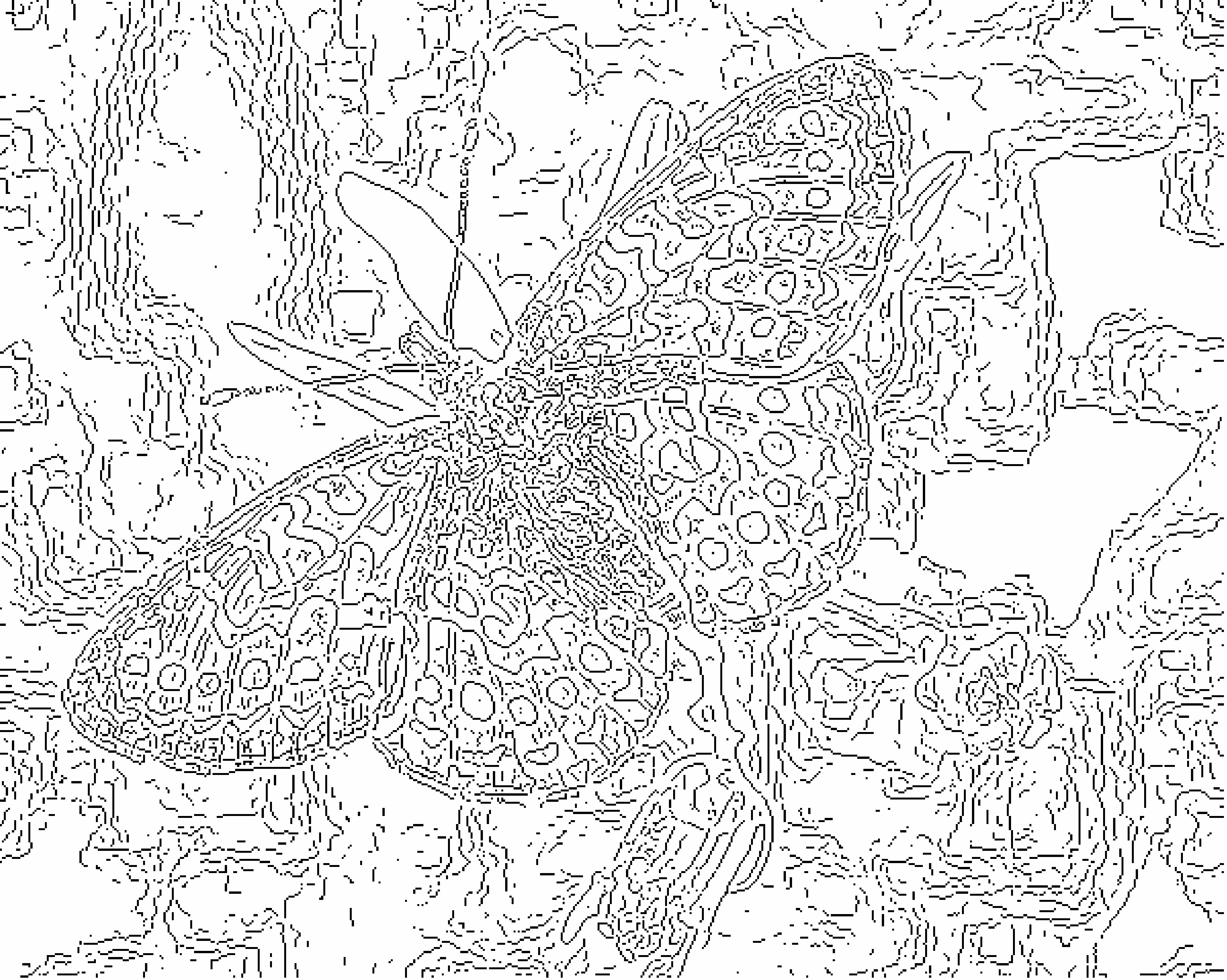
- suppose this is a curve that we are following
  - thickness represents the magnitude of the gradient



  - we require a large magnitude to start, i.e. above a threshold $T_1$
  - once we start we keep going even if the magnitude falls below the threshold
  - we only declare the contour as done if it falls below a second threshold $T_2$, where $T_2 < T_1$
  - once again, the optimal values of these thresholds are image dependent

# Parameter tuning

- in summary, the combination of
  - smoothed derivatives,
  - detection of maxima of gradient magnitude,
  - edge following
- is the essence of most modern edge detectors
- the classical is the "Canny edge detector" which implements all this steps
- as we have seen there are a number of parameters
  - smoothing scale
  - two hysteresis thresholds
- in practice these can have significant effect on the quality of the resulting edge maps
- unfortunately there are no universally good values



fine scale
high
threshold

too many
false
edges

coarse scale, high threshold

we loose edge points

coarse
scale
low
threshold

we still
have gaps
but once
again
false
edges

# The Canny edge detector

- there are many implementations available
  - matlab has one
  - there is freely available C code on the web
  - there are various applets that allow you to play with the parameters
  - an example is
  - http://www.cs.washington.edu/research/imagedatabase/demo/edge/
  - make sure you experiment and get a feel for how the parameters influence the edge detection results
  - the Canny edge detectors is the closest that you will find to a standard solution to a vision problem

# Any questions?