Edge Detection

Computer Vision

P. Schrater

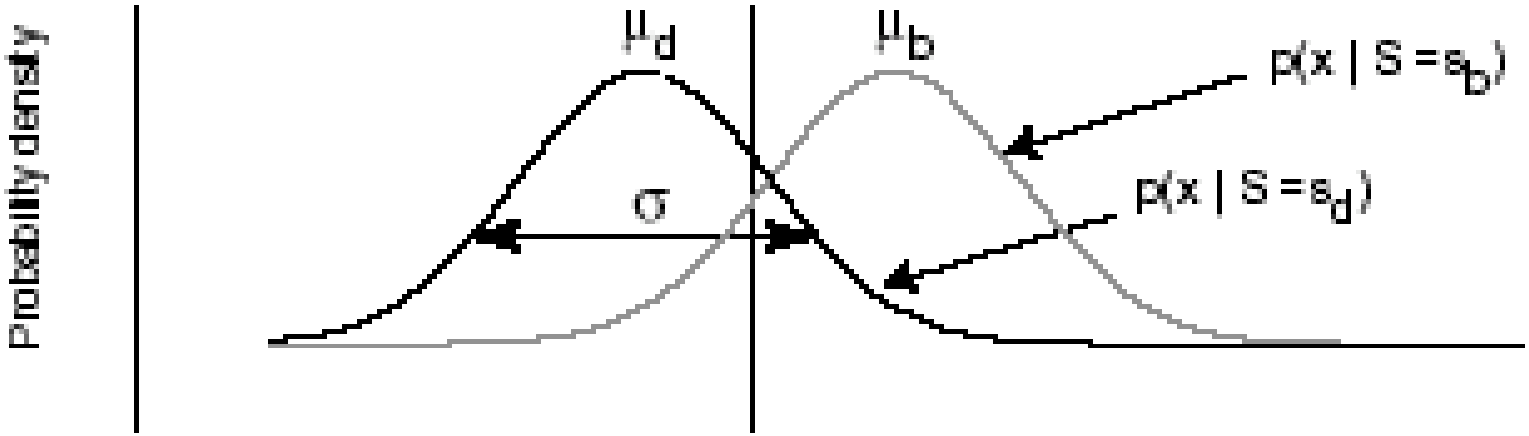
Spring 2003

$$\log \frac{P(\phi(I(x))|\ \text{on-edge})}{P(\phi(I(x))|\ \text{off-edge})} > T,$$

Simplest Model: (Canny)

Edge(x) = $a\ U(x) + n(x)$

Convolve image with U and find points with high magnitude. Choose value by comparing with a threshold determined by the noise model.



Probability of a filter response off an edge

Probability of a filter response on an edge

Fundamental limits on edge detection

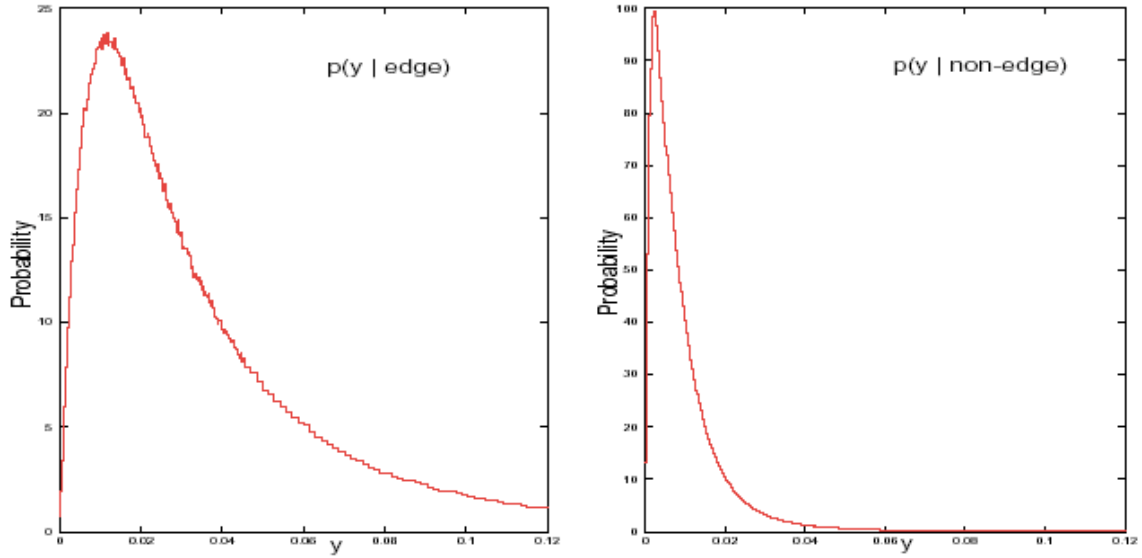


Figure 2.5 Empirical distributions for a gradient filter response on boundaries (left) and off boundaries (right).

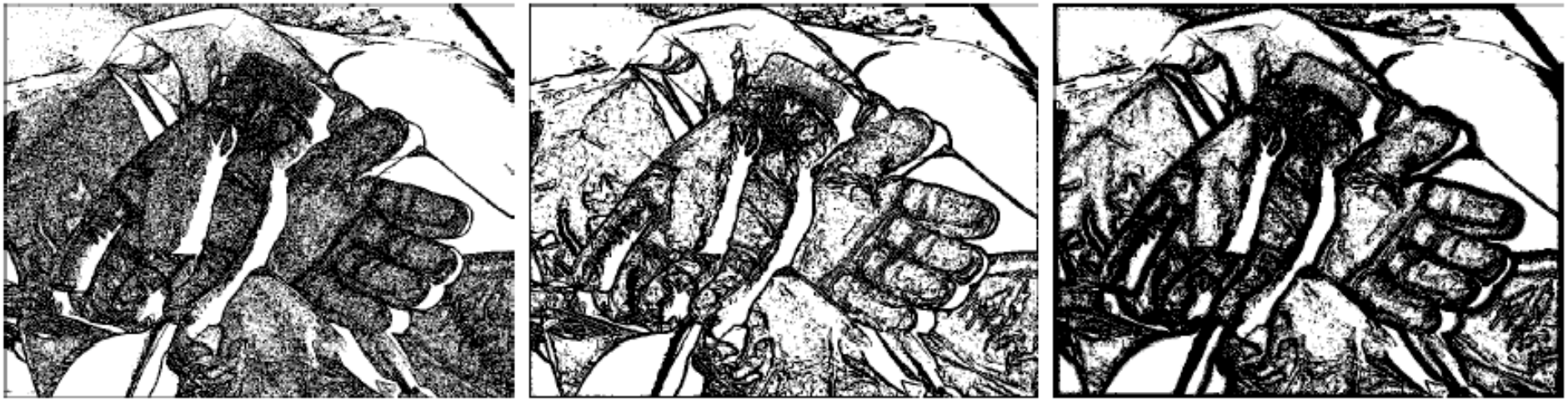
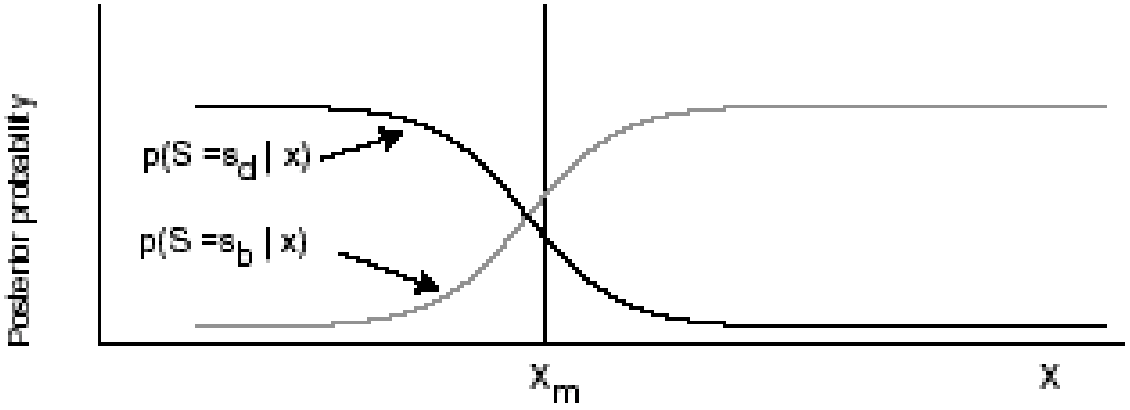


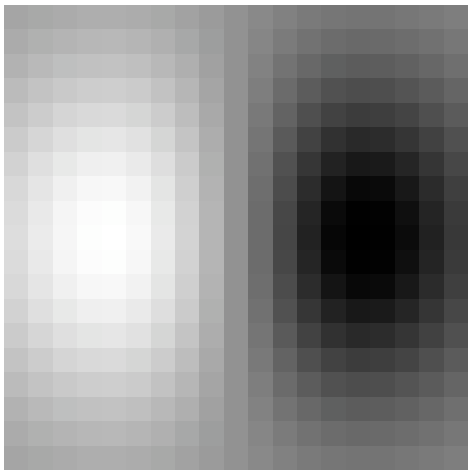
Figure 2.6 The edge estimated on the glove image by ML with the filter at scale 0 (left), filter at scale 1 (centre), and filter with scales $0, 1, 2, 4$ (right). Observe that ML significantly overestimates the number of edges in this image.

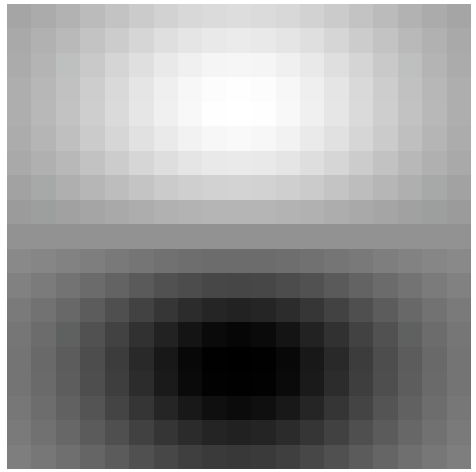
Need to take into account base edge rate.

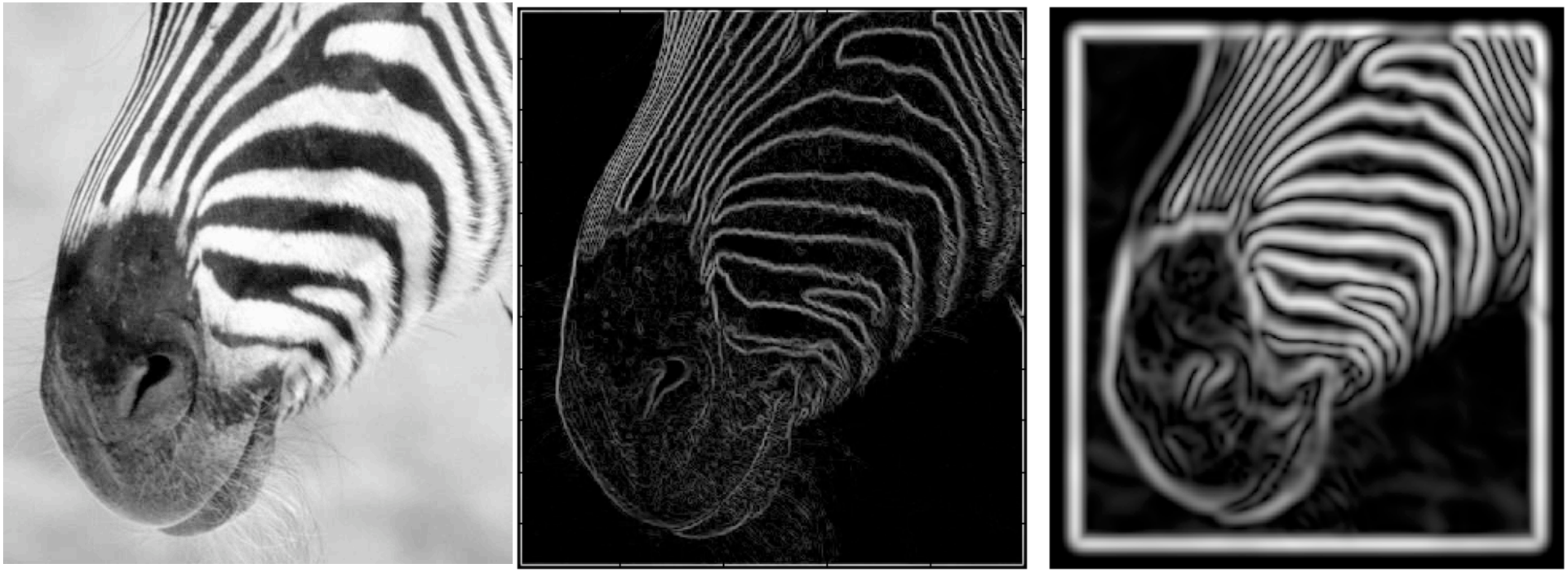


Smoothing and Differentiation

- Issue: noise
  - smooth before differentiation
  - two convolutions to smooth, then differentiate?
  - actually, no - we can use a derivative of Gaussian filter
    - because differentiation is convolution, and convolution is associative

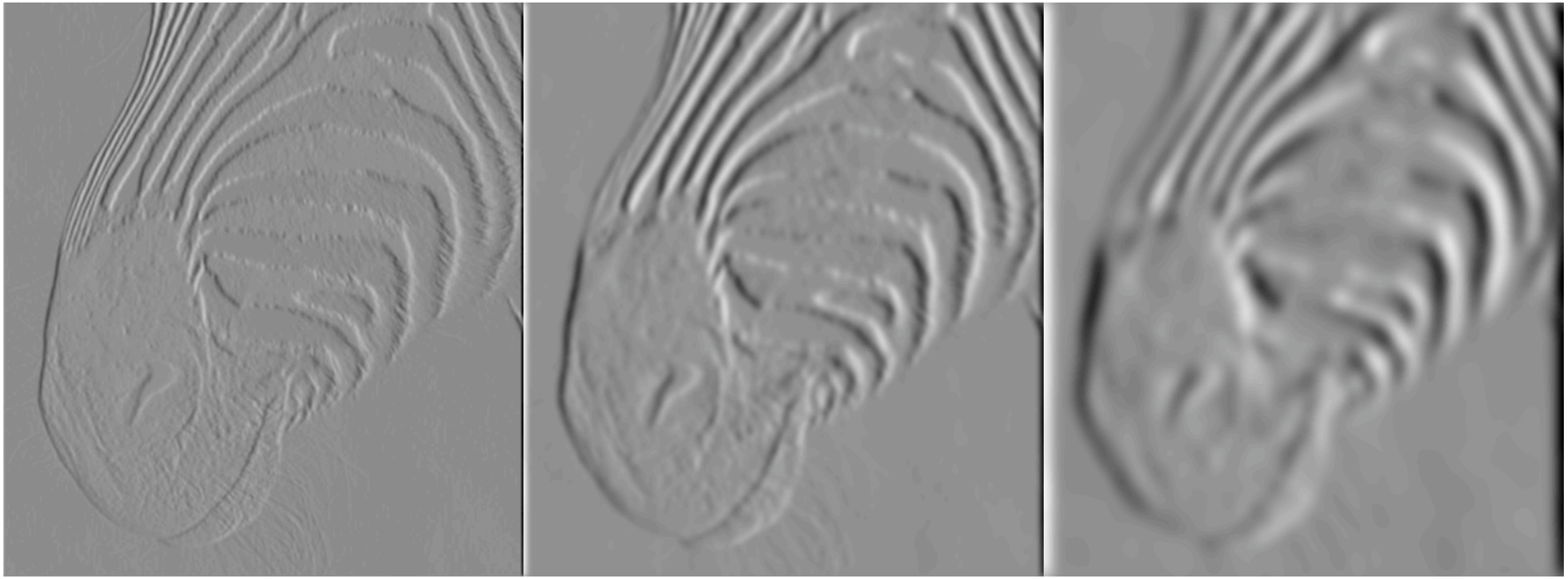






There are three major issues:

1) The gradient magnitude at different scales is different; which should we choose?
2) The gradient magnitude is large along thick trail; how do we identify the significant points?
3) How do we link the relevant points up into curves?



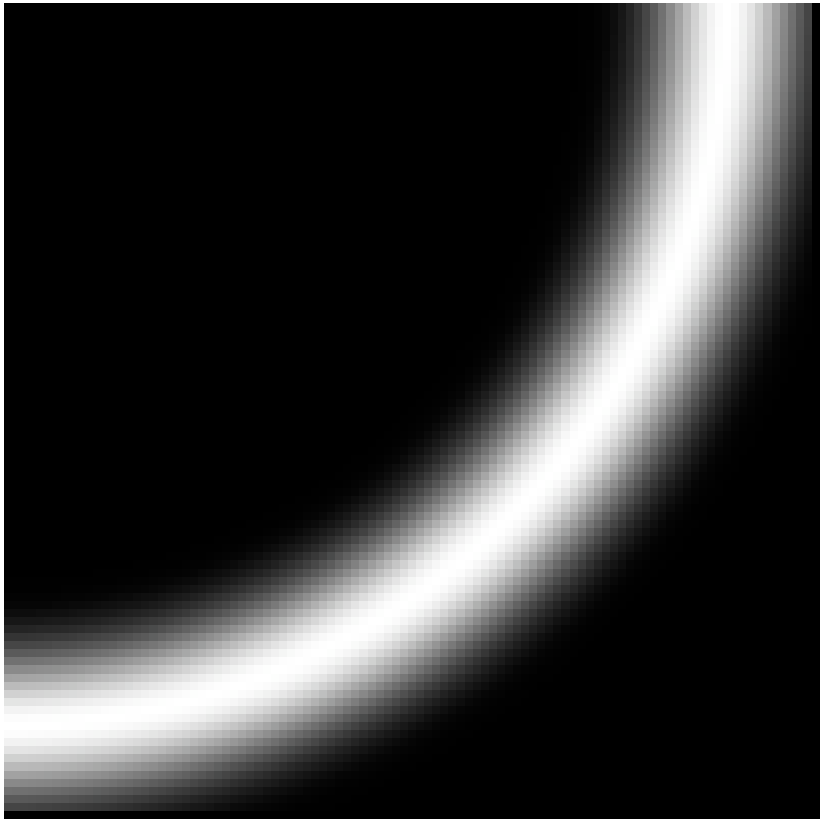
1 pixel

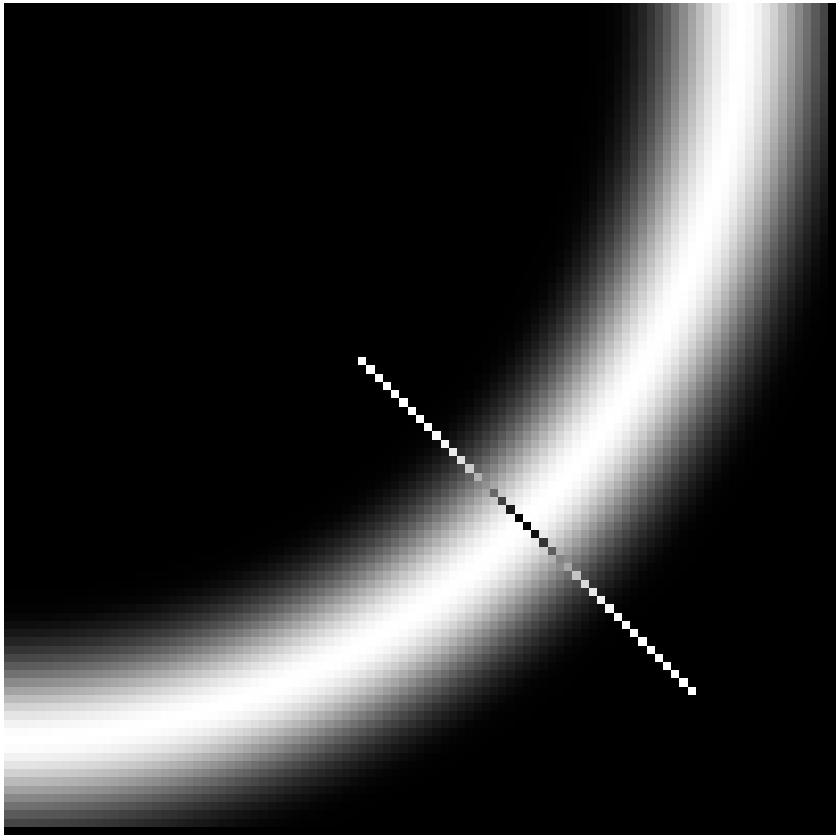
3 pixels

7 pixels

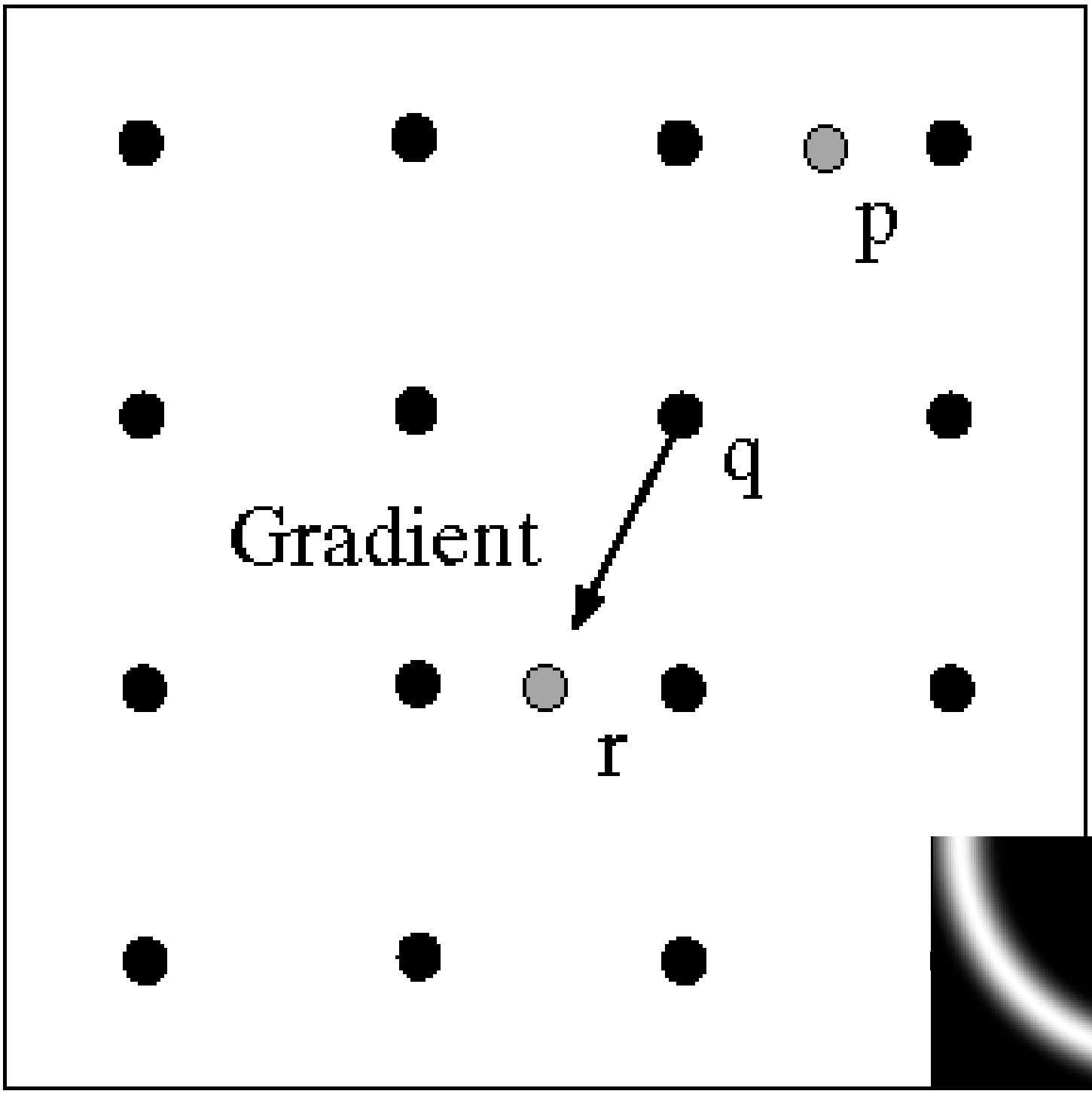
The scale of the smoothing filter affects derivative estimates, and also the semantics of the edges recovered.

Computing Edges via the gradient



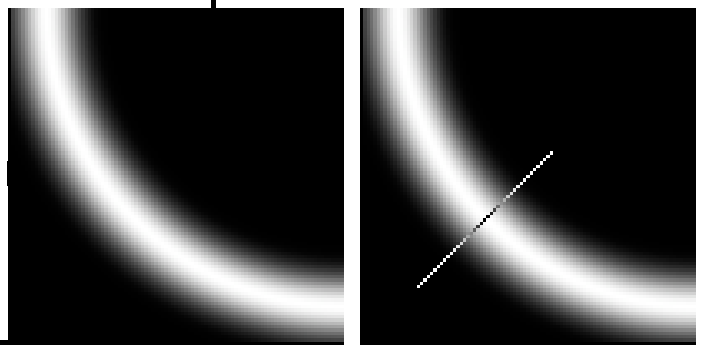


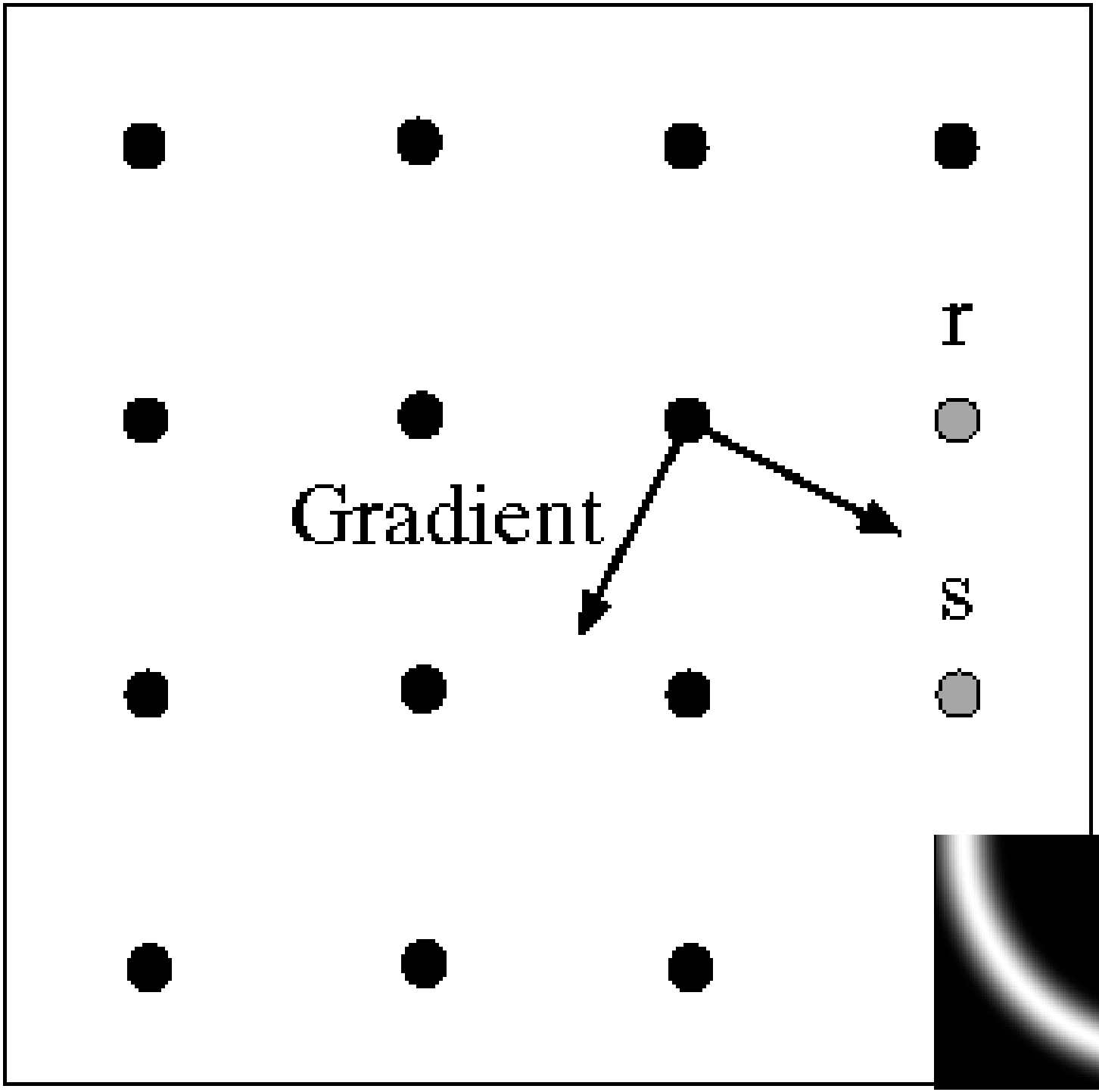
We wish to mark points along the curve where the magnitude is biggest. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?



Non-maximum suppression

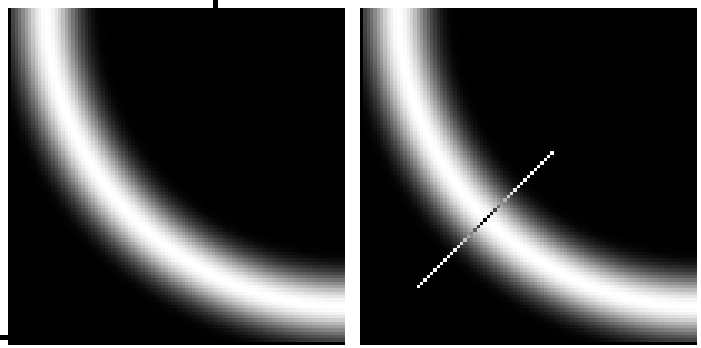
At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.





Predicting the next edge point

Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).



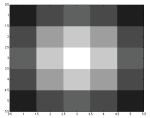
Remaining issues

- 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.

Edge Detection Algorithm

- Define Edge filter:

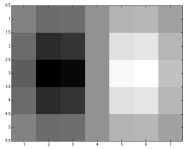
$$G_{uv} = \exp\left(-\frac{1}{2}\left(\frac{(x_u - N/2)^2}{pixelwidth^2} + \frac{(y_v - M/2)^2}{pixelwidth^2}\right)\right)$$

G = 

$$D_{ij} = [-1 \quad 0 \quad 1]$$

$$Dx = G \quad D$$

$$Dy = G \quad D^T$$

D_x = 

- Next Compute Gradient

$$\bar{I}_x = Dx \quad I$$

$$I_y = Dy \quad I$$

$$\nabla I(i,j) = \begin{bmatrix} I_x(i,j) \\ I_y(i,j) \end{bmatrix}$$

Gradient Magnitude

$$\left\|\nabla I(i,j)\right\| = \sqrt{I_x(i,j)^2 + I_y(i,j)^2}$$

$$\theta \nabla I(i,j) = \tan^{-1}\left(\frac{I_y(i,j)}{I_x(i,j)}\right)$$

Gradient Direction

Edge Detection Algorithm

- Find an initial point (i,j) such that:

$$\left\| \nabla I(i,j) \right\| > c$$ Where c is relatively large

- Find nearby points in the direction of (i,j)'s gradient

$$\vec{n}(i,j) = \begin{bmatrix} I_x(i,j) / \left\| \nabla I(i,j) \right\| \\ I_y(i,j) / \left\| \nabla I(i,j) \right\| \end{bmatrix}$$

Find nearby points $\begin{bmatrix} u \\ v \end{bmatrix}$ such that $n(u,v)^T \vec{n}(i,j) <$

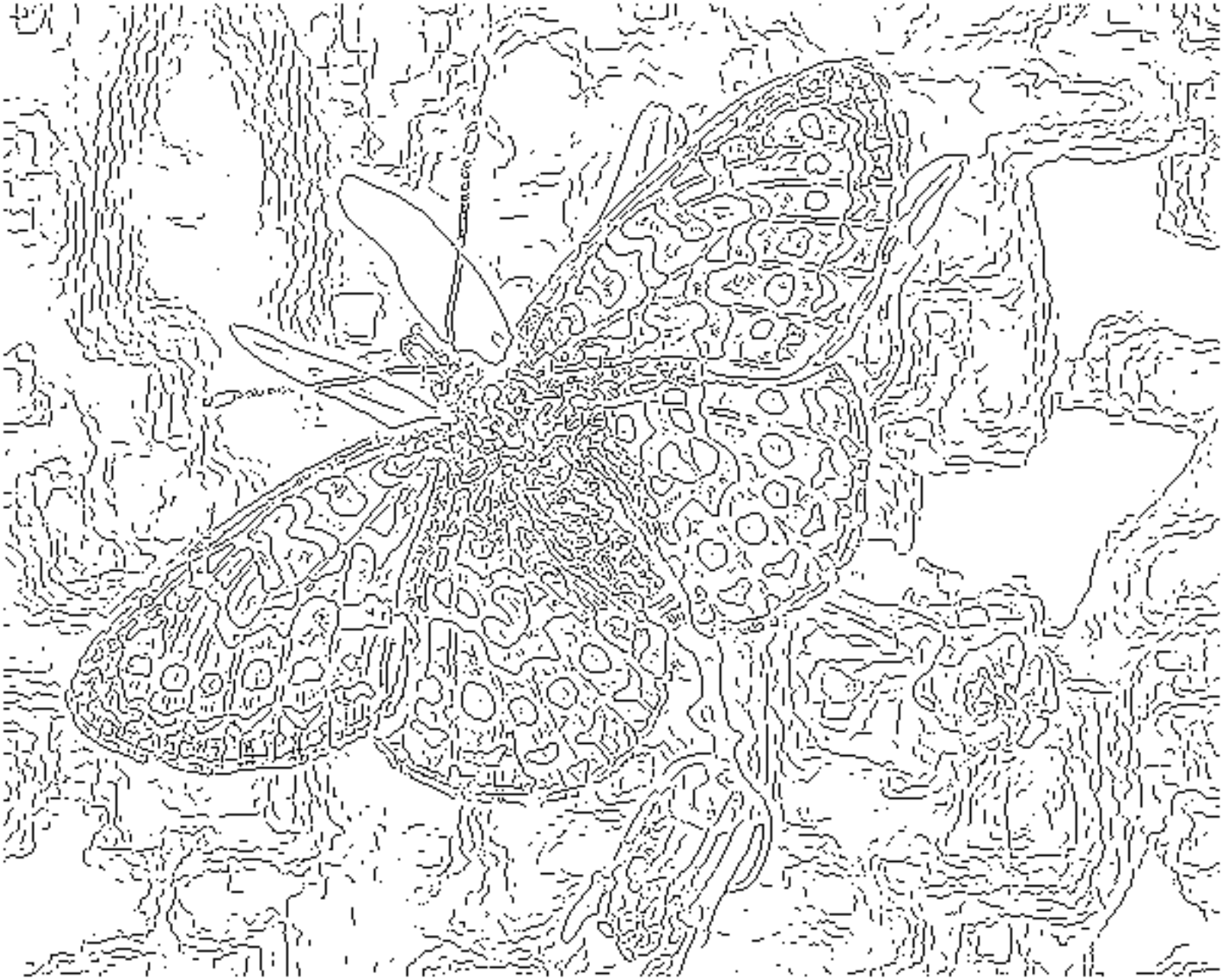
Choose $\begin{bmatrix} u \\ v \end{bmatrix}$ with maximal gradient magnitude

Edge Detection Algorithm

- While set of *(i,j)* with $\|\nabla I(i,j)\| > \alpha$ not visited
  - Find a start point, erasing points that have been checked.
    - While possible, expand a chain through the current point by:
      - 1) predicting a next set of points (not visited) using the direction perp. to gradient
      - 2) Finding which of set (if any) is a local maximum
      - 3) Test if grad mag for local max > k
      - 4) Recording all the set as visited
    - record point and set to current point.
  - end
- end

Notice

- Something nasty is happening at corners
- Scale affects contrast
- Edges aren’t bounding contours



fine scale
high
threshold

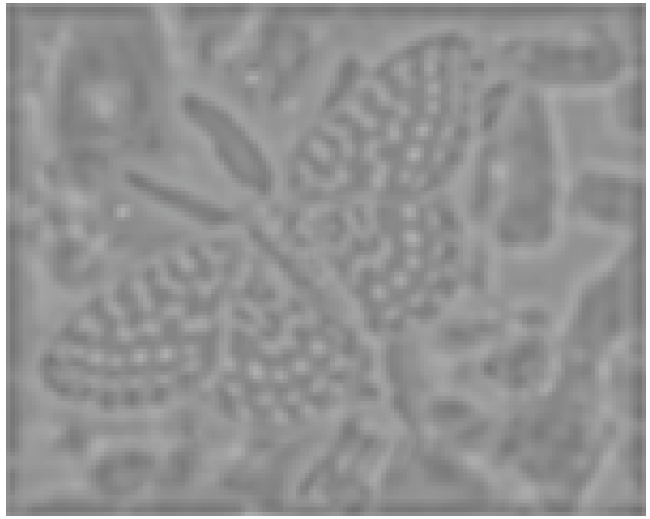
coarse scale, high threshold


coarse
scale
low
threshold

Positive responses

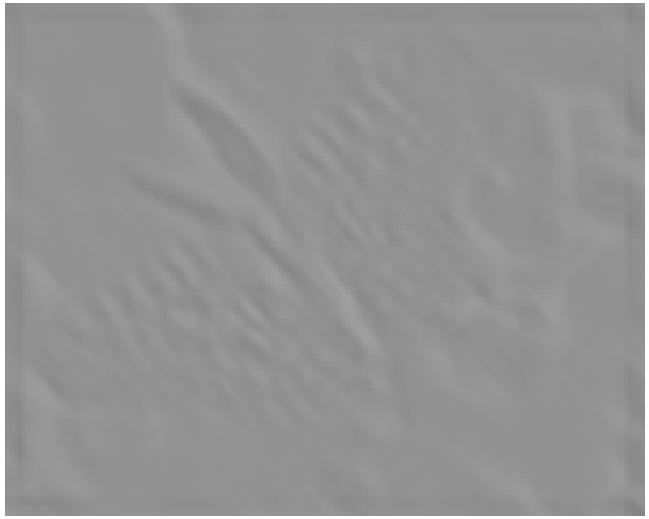
Zero mean image, -1:1 scale

Zero mean image, -max:max scale



Positive responses

Zero mean image, -1:1 scale



Zero mean image, -max:max scale

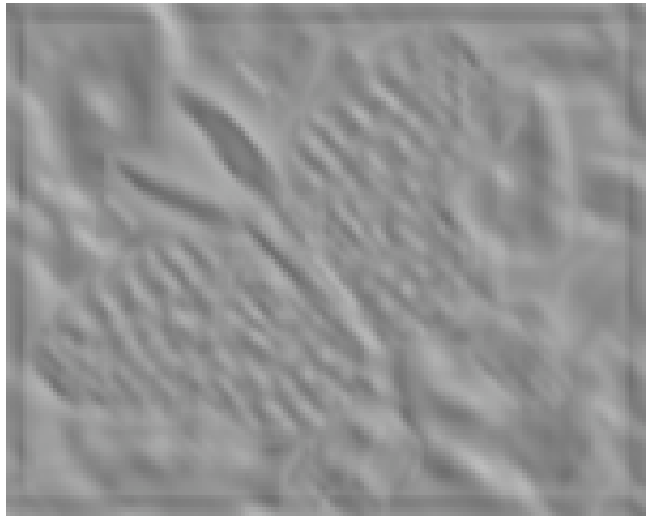


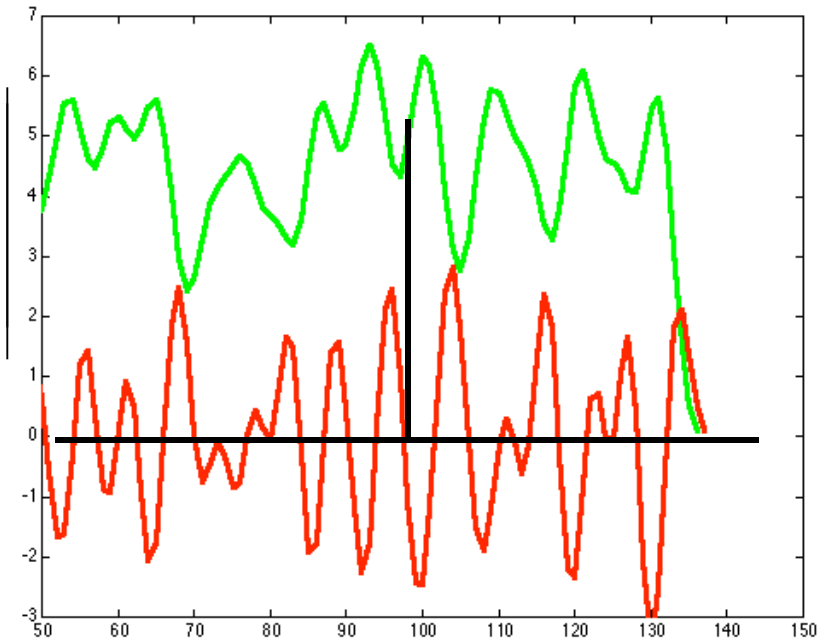
Figure from “Computer Vision for Interactive Computer Graphics,” W.Freeman et al, IEEE Computer Graphics and Applications, 1998 copyright 1998, IEEE

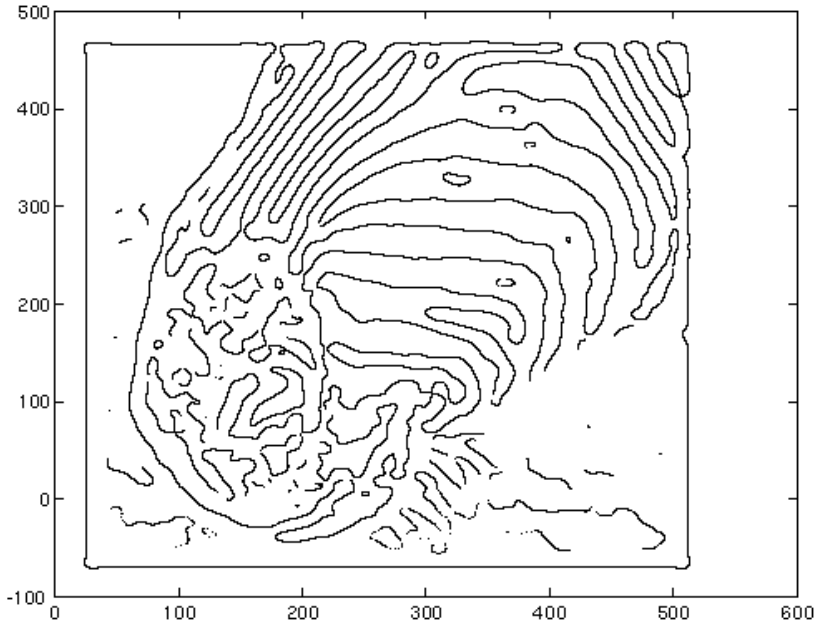
The Laplacian of Gaussian

- Another way to detect an extremal first derivative is to look for a zero second derivative
- In 2D:

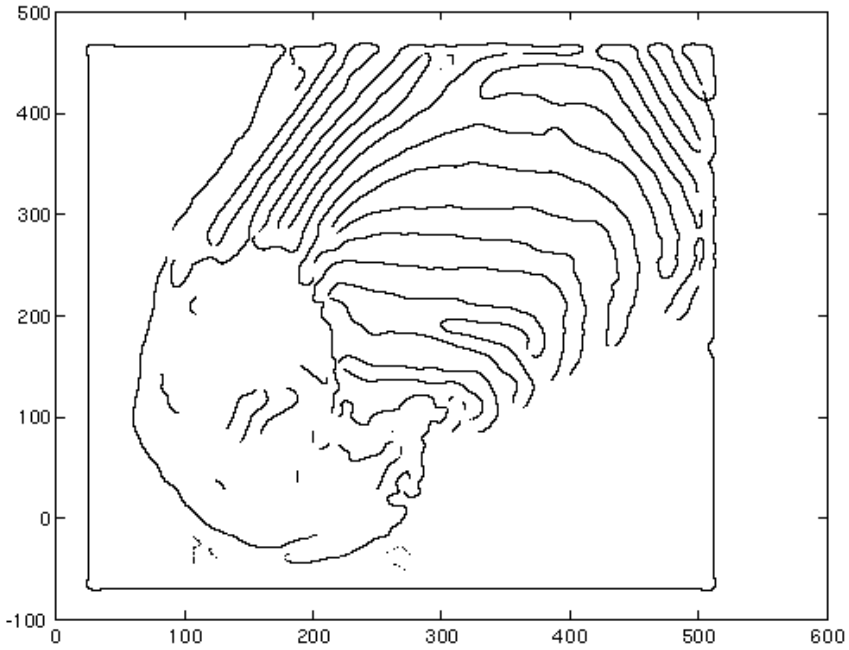
$$\nabla^2 I(x,y) = \frac{\partial^2 I(x,y)}{\partial x^2} + \frac{\partial^2 I(x,y)}{\partial y^2}$$

- Bad idea to apply a Laplacian without smoothing
  - smooth with Gaussian, apply Laplacian
  - this is the same as filtering with a Laplacian of Gaussian filter
- Now mark the zero points *where there is a sufficiently large derivative, and enough contrast*





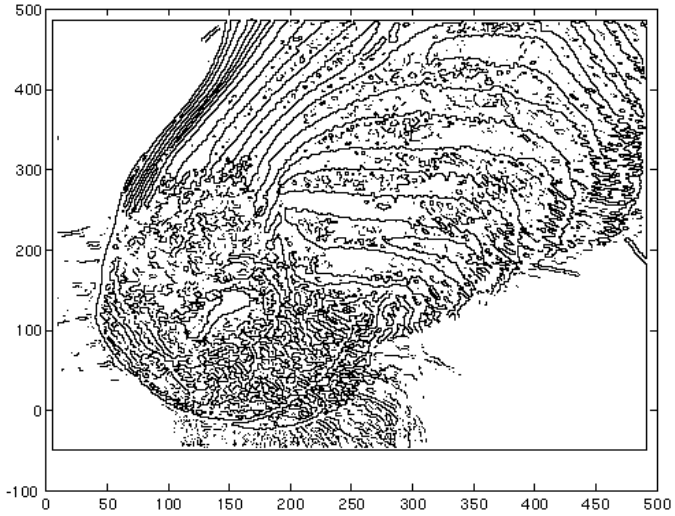
sigma=4



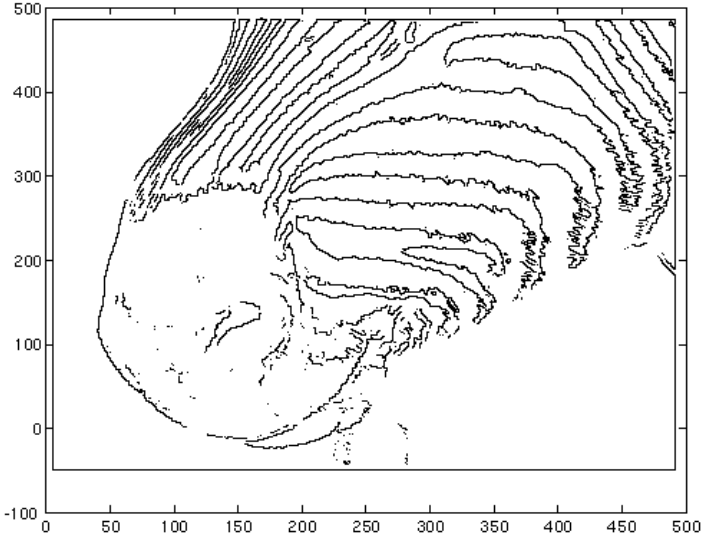
contrast=1

LOG zero crossings

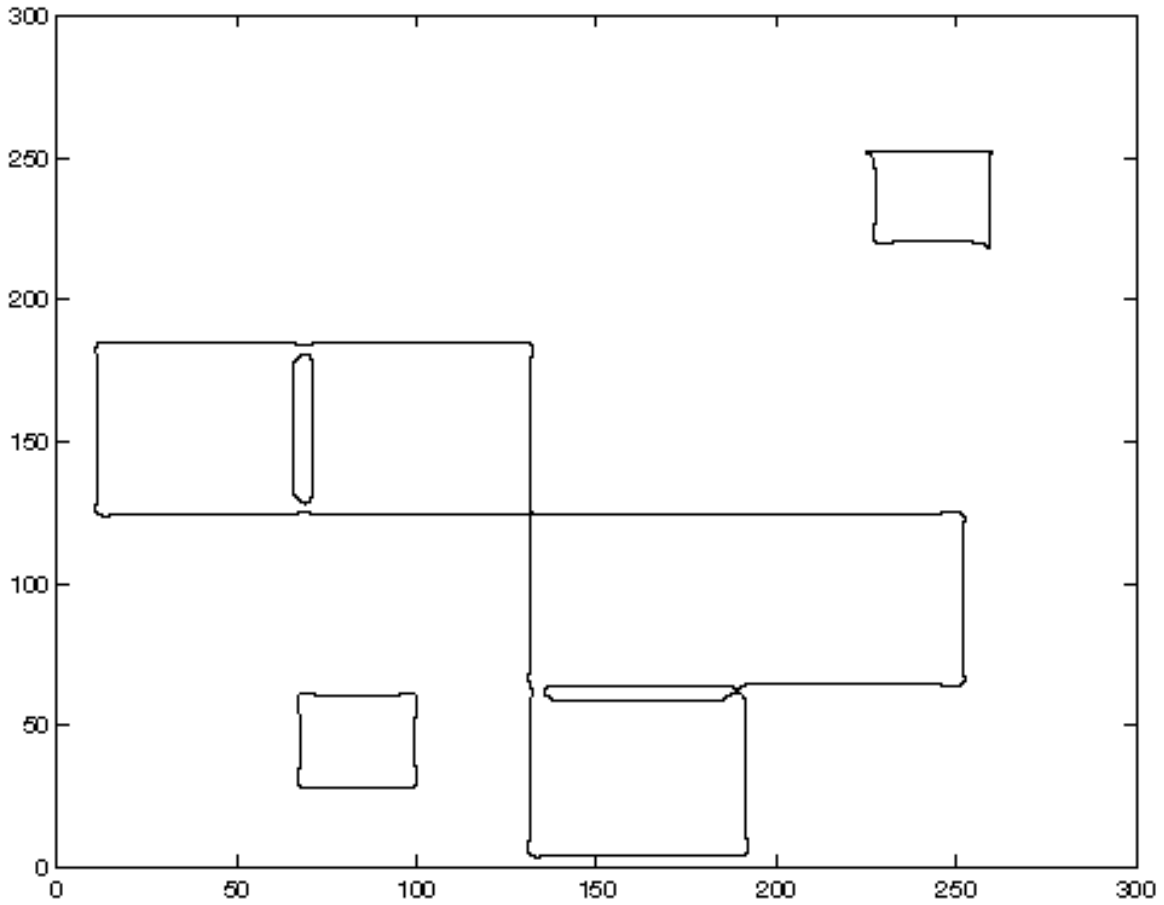
contrast=4



sigma=2



We still have unfortunate behaviour at corners



Orientation representations

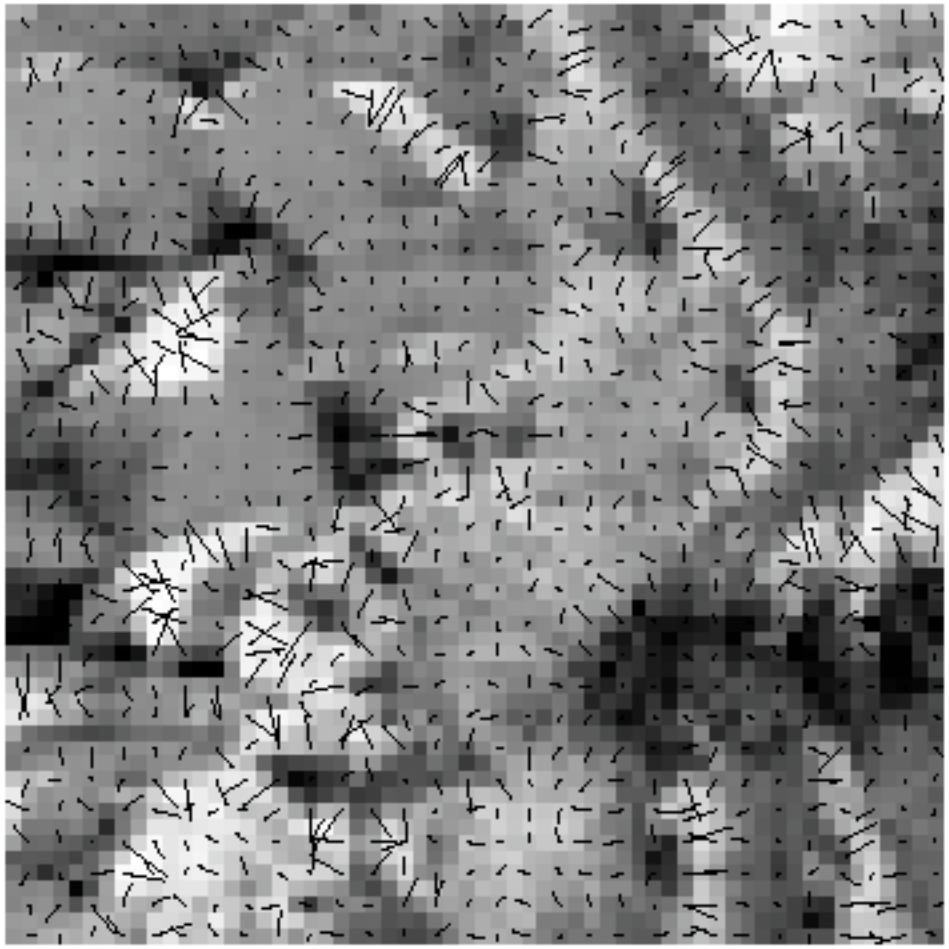
- The gradient magnitude is affected by illumination changes
  - but it's direction isn't
- We can describe image patches by the swing of the gradient orientation

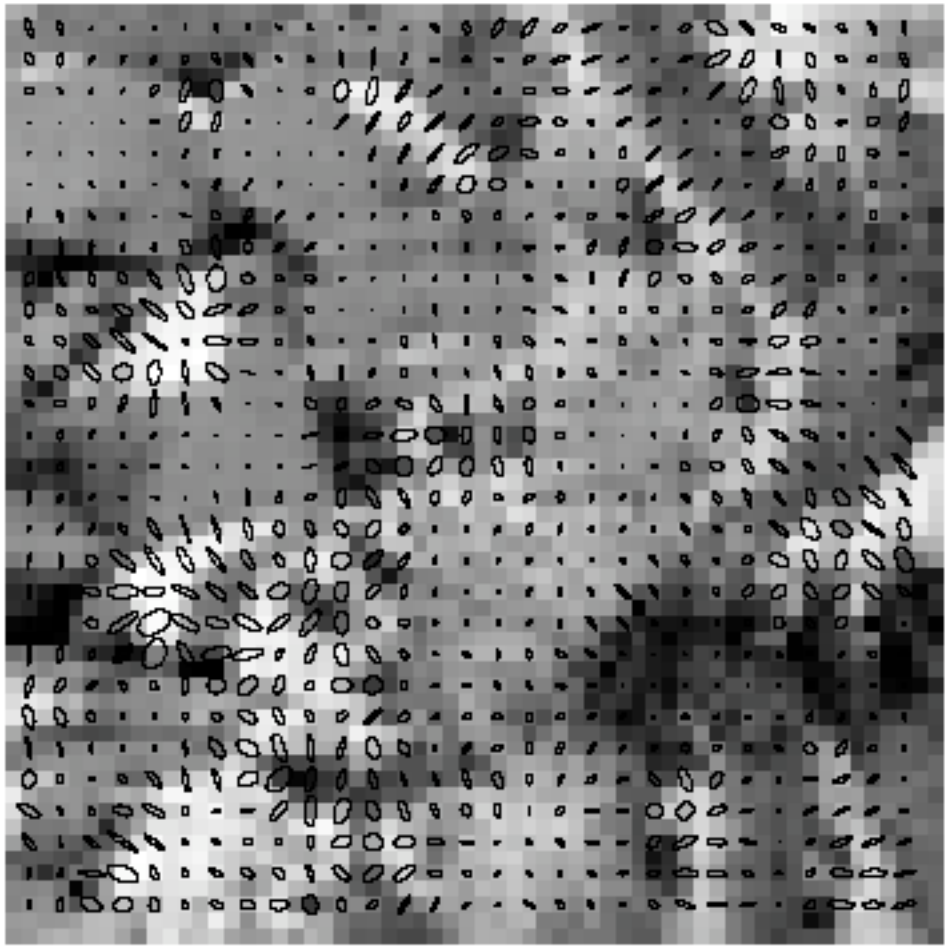
- Important types:
  - constant window
    - small gradient mags
  - edge window
    - few large gradient mags in one direction
  - flow window
    - many large gradient mags in one direction
  - corner window
    - large gradient mags that swing

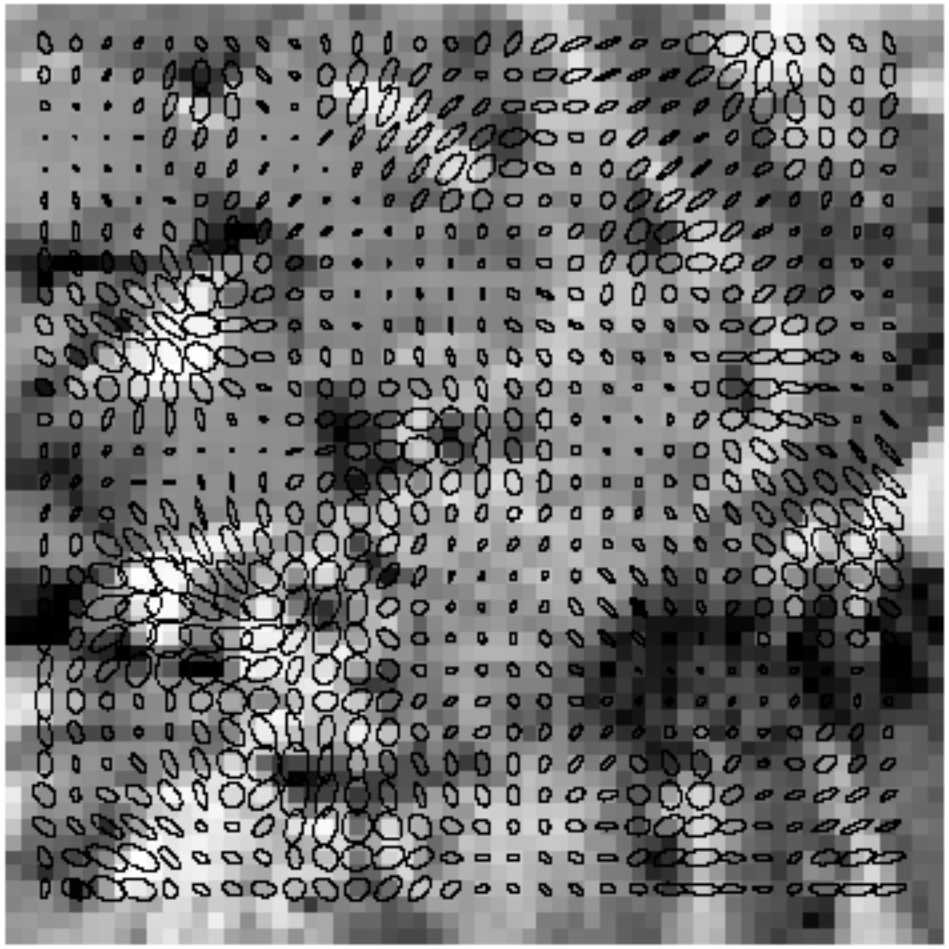
Representing Windows

- Types
  - constant
    - small eigenvalues
  - Edge
    - one medium, one small
  - Flow
    - one large, one small
  - corner
    - two large eigenvalues

$$H = \sum_{window} (\nabla I)(\nabla I)^T$$



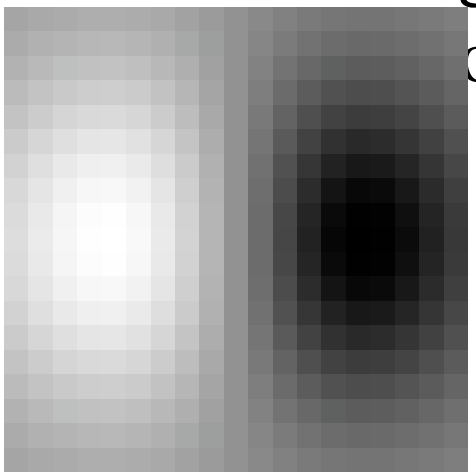


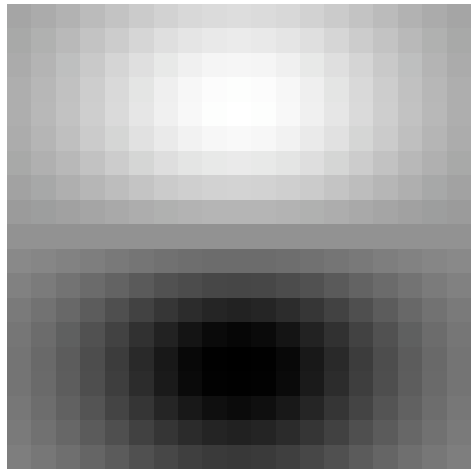


Filters are templates

- Applying a filter at some point can be seen as taking a dot-product between the image and some vector
- Filtering the image is a ot products

- Insight
  - filters look like the effects they are intended to find
  - filters find effects they look like





Normalized correlation

- Think of filters of a dot product
  - now measure the angle
  - i.e normalised correlation output is filter output, divided by root sum of squares of values over which filter lies
- Tricks:
  - ensure that filter has a zero response to a constant region (helps reduce response to irrelevant background)
  - subtract image average when computing the normalizing constant (i.e. subtract the image mean in the neighbourhood)
  - absolute value deals with contrast reversal