Linear Filters

- General process:
  - Form new image whose pixels are a weighted sum of original pixel values, using the same set of weights at each point.
- Properties
  - Output is a linear function of the input
  - Output is a shift-invariant function of the input (i.e. shift the input image two pixels to the left, the output is shifted two pixels to the left)
- Example: smoothing by averaging
  - form the average of pixels in a neighborhood
- Example: smoothing with a Gaussian
  - form a weighted average of pixels in a neighbourhood
- Example: finding a derivative
  - form a weighted average of pixels in a neighbourhood

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x,y]$

	0	10							
	0	20	40	60					
	0	40			90	90			
	0	60				90			
		40				90			
		20							
		10							
		0							

R[x,y]

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

I

$F[x, y]$

$f = \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

R[x,y]

Convolution

$f(x+y) = f(x) + f(y)$

$f(ax) = a f(x)$

- Represent these weights as an image, H
- H is usually called the **kernel**
- Operation is called **convolution**
  - it's associative

-
- Notice wierd order of indices
  - all examples can be put in this form
  - it's a result of the derivation expressing any shift-invariant linear operator as a convolution.

$$(I * f)(x,y) \equiv \sum_{u=-\infty}^{\infty} \sum_{v=-\infty}^{\infty} I(x-u, y-v)\; f(u,v)$$

+1 +1

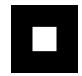
−1 −1

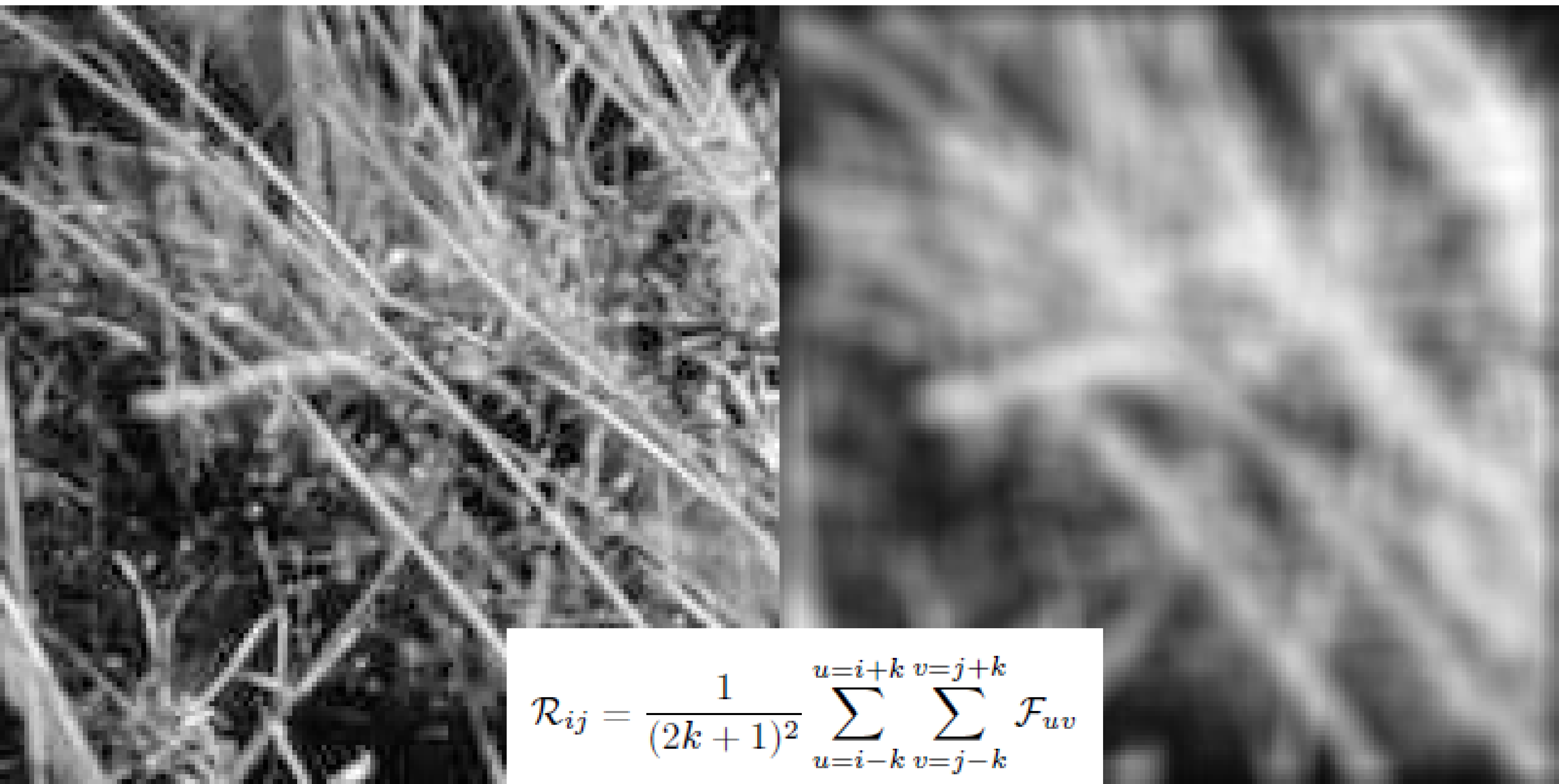
$*$

5, 5

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Example: Smoothing by Averaging

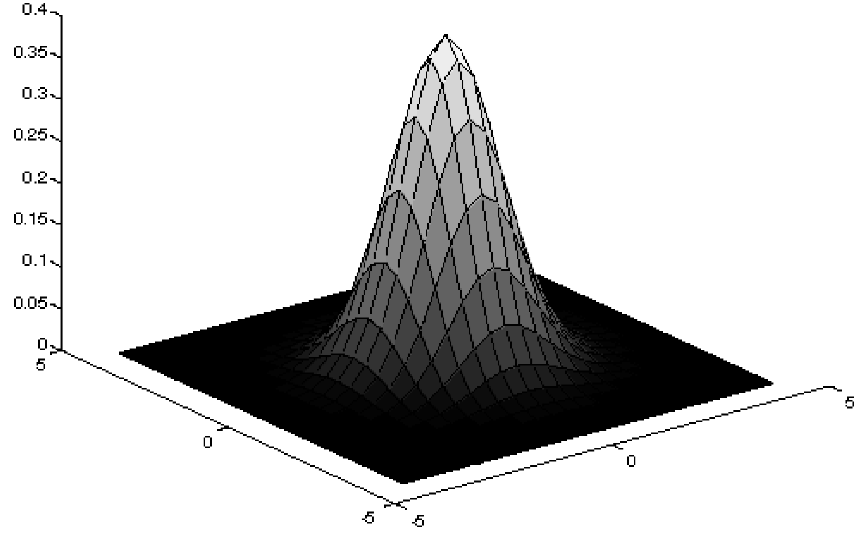




$$\mathcal{R}_{ij} = \frac{1}{(2k+1)^2} \sum_{u=i-k}^{u=i+k} \sum_{v=j-k}^{v=j+k} \mathcal{F}_{uv}$$

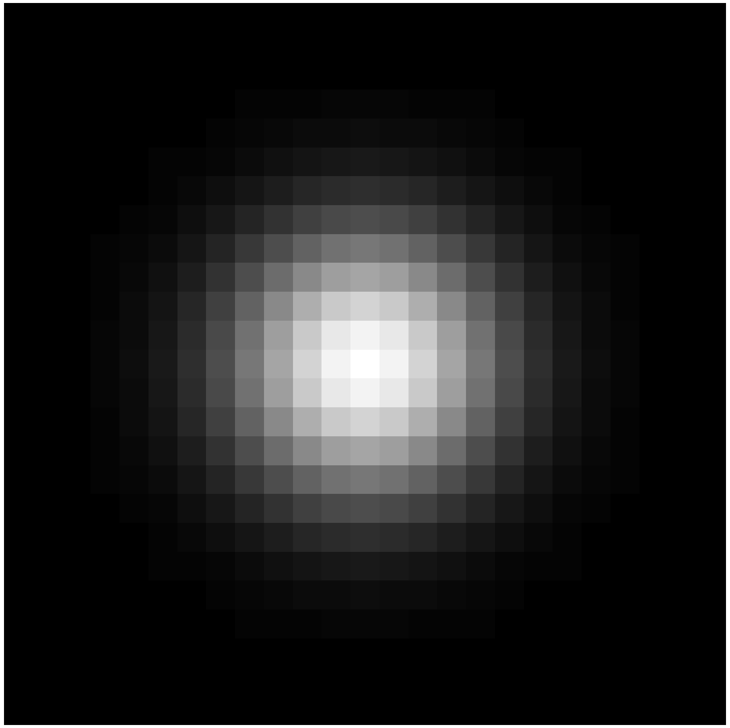
Smoothing with a Gaussian

- Smoothing with an average actually doesn’t compare at all well with a defocussed lens
  - Most obvious difference is that a single point of light viewed in a defocussed lens looks like a fuzzy blob; but the averaging process would give a little square.



- A Gaussian gives a good model of a fuzzy blob

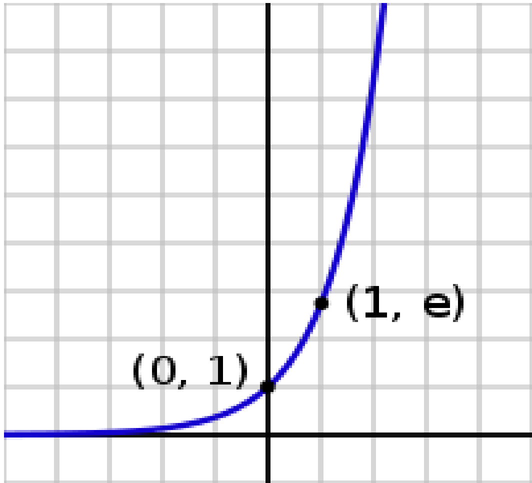
An Isotropic Gaussian



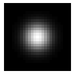
- The picture shows a smoothing kernel proportional to

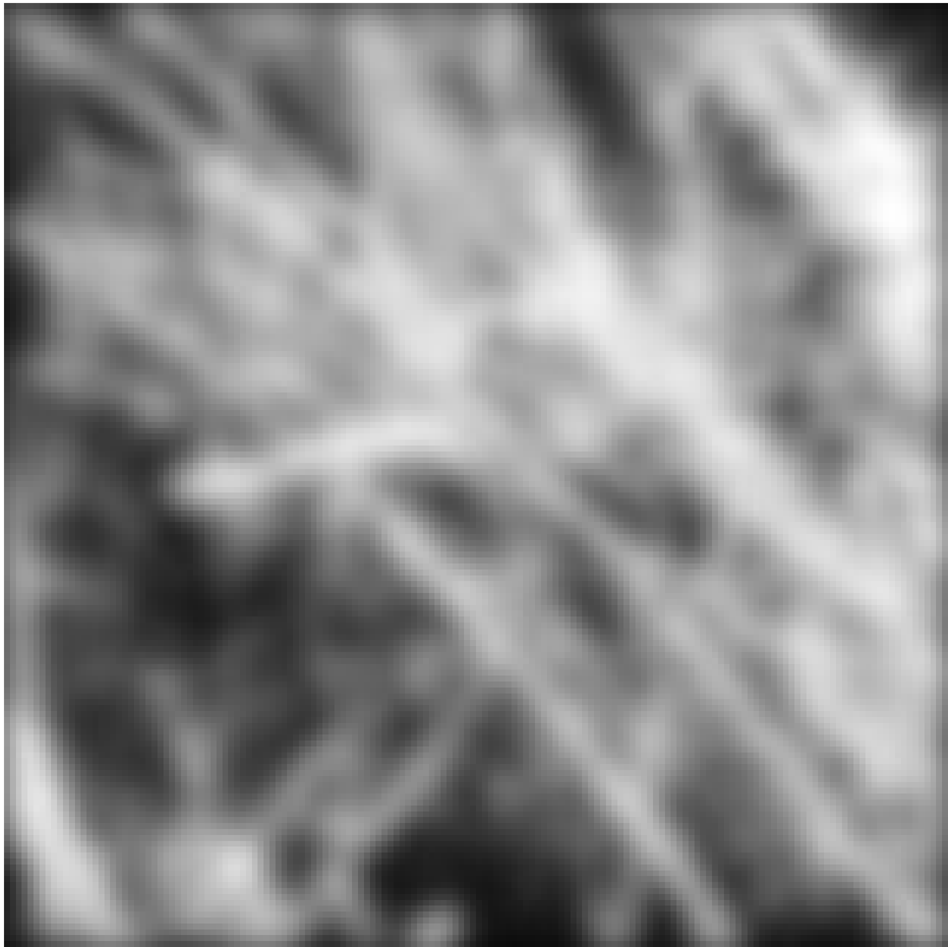
$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(x^2+y^2)}{2\sigma^2}\right)$$

(which is a reasonable model of a circularly symmetric fuzzy blob)



Smoothing with a Gaussian





Differentiation and convolution

- Recall

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

- We could approximate this as

$$\frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$$

- Now this is linear and shift invariant, so must be the result of a convolution.

(which is obviously a convolution; it's not a very good way to do things, as we shall see)

This is the same as a convolution, where the convolution kernel is

$$\mathcal{H} = \left\{ \begin{array}{ccr} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{array} \right\}$$

Finite differences

Noise

- Simplest noise model
  - independent stationary additive Gaussian noise
  - the noise value at each pixel is given by an independent draw from the same normal probability distribution

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

- Issues
  - this model allows noise values that could be greater than maximum camera output or less than zero
  - for small standard deviations, this isn't too much of a problem - it's a fairly good model
  - independence may not be justified (e.g. damage to lens)
  - may not be stationary (e.g. thermal gradients in the ccd)

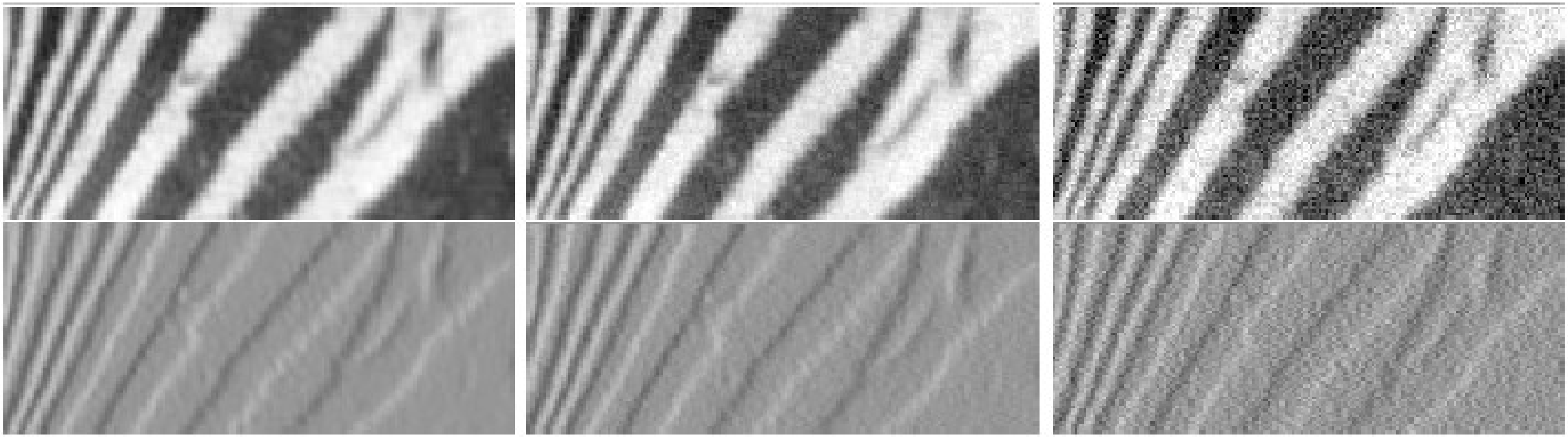
sigma=1

sigma=16

Finite differences and noise

- Finite difference filters respond strongly to noise
  - obvious reason: image noise results in pixels that look very different from their neighbours
- Generally, the larger the noise the stronger the response
- What is to be done?
  - intuitively, most pixels in images look quite a lot like their neighbours
  - this is true even at an edge; along the edge they’re similar, across the edge they’re not
  - suggests that smoothing the image should help, by forcing pixels different to their neighbours (=noise pixels?) to look more like neighbours

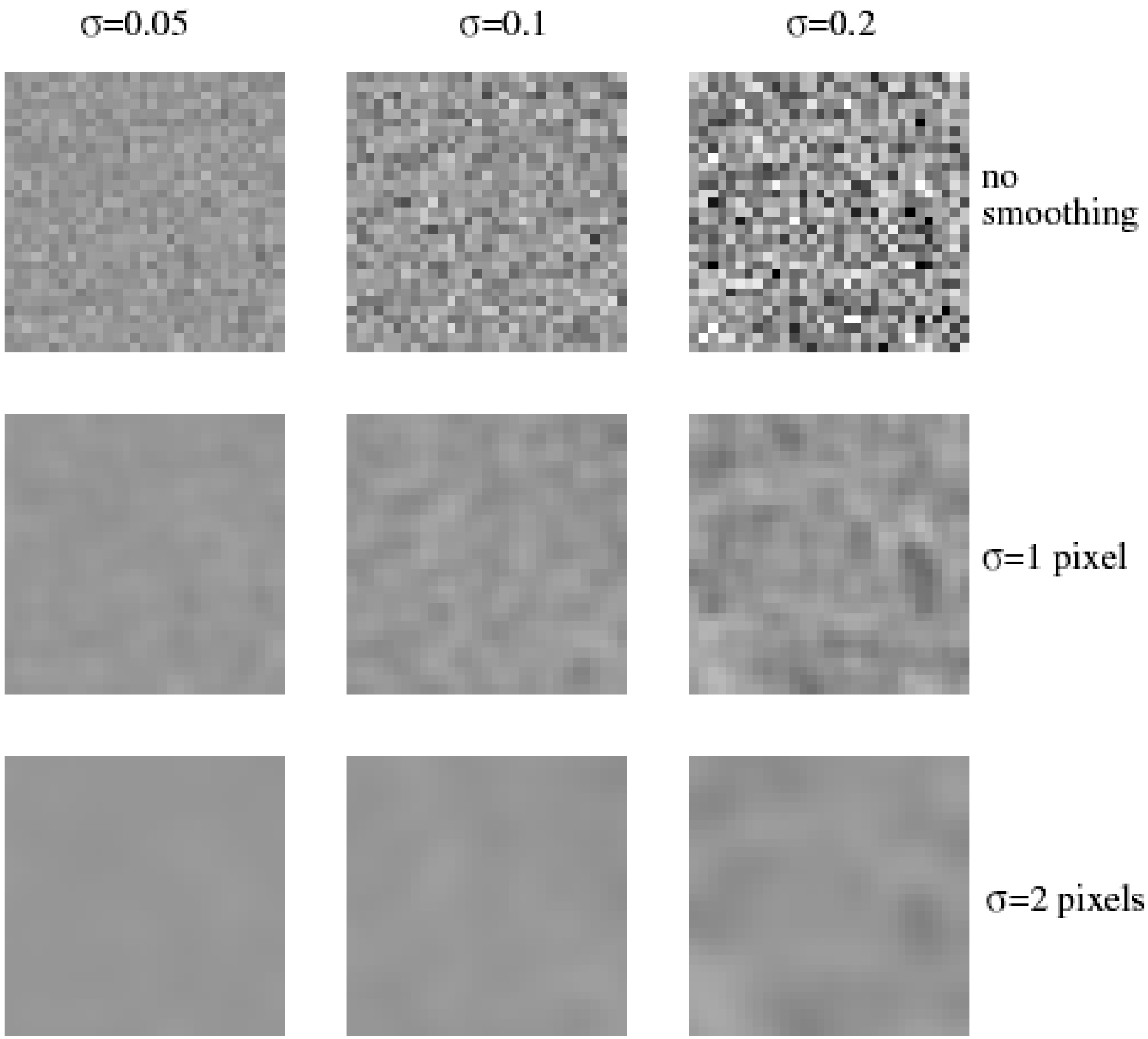
Finite differences responding to noise



Increasing noise ->
(this is zero mean additive gaussian noise)

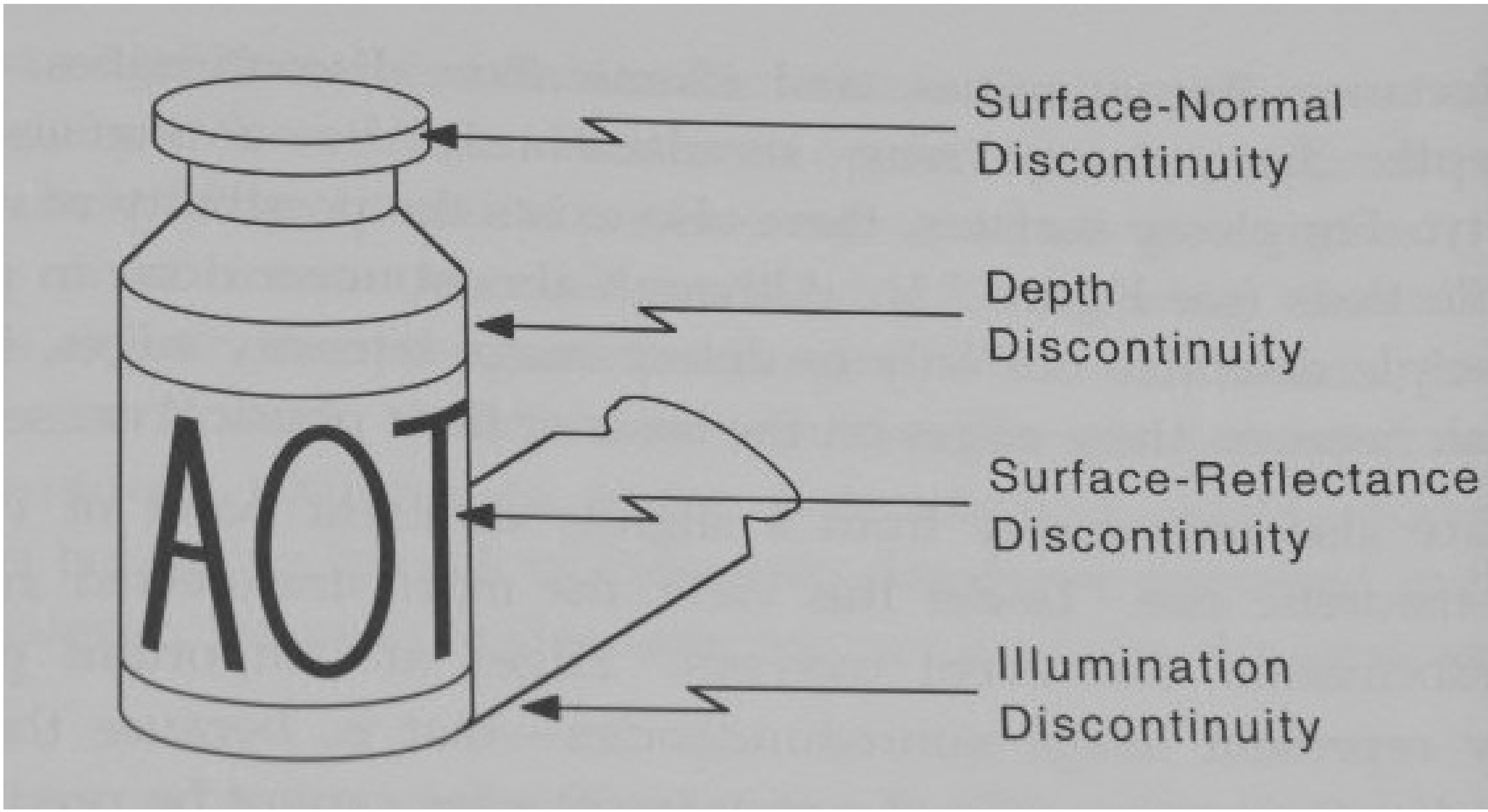
Smoothing reduces noise

- Generally expect pixels to "be like" their neighbours
  - surfaces turn slowly
  - relatively few reflectance changes
- Generally expect noise processes to be independent from pixel to pixel
- Implies that smoothing suppresses noise, for appropriate noise models
- Scale
  - the parameter in the symmetric Gaussian
  - as this parameter goes up, more pixels are involved in the average
  - and the image gets more blurred
  - and noise is more effectively suppressed



The effects of smoothing
Each row shows smoothing with gaussians of different width; each column shows different realisations of an image of gaussian noise.

Edges as Image Features



Gradients and edges

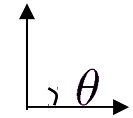
- Points of sharp change in an image are interesting:
  - change in reflectance
  - change in object
  - change in illumination
  - noise
- Sometimes called **edge points**

- General strategy
  - determine image gradient
  - now mark points where gradient magnitude is particularly large wrt neighbors (ideally, curves of such points).

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

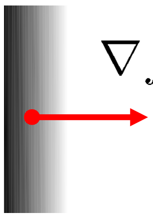
$$\mathbf{\Theta} = \arctan\left(\frac{\mathbf{G}_y}{\mathbf{G}_x}\right)$$

Image gradient



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

It points in the direction of most rapid change in intensity

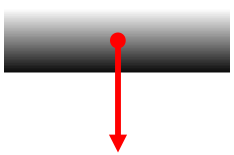


$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$

The gradient direction is given by:

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

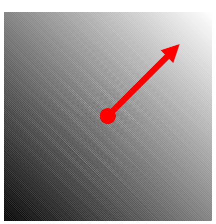
-
- how does this relate to the direction of the edge?



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

The *edge strength* is given by the gradient magnitude

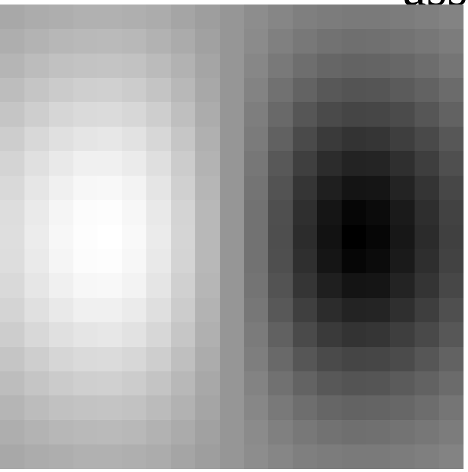
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

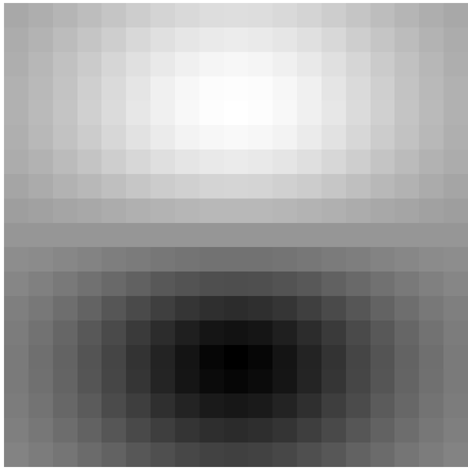


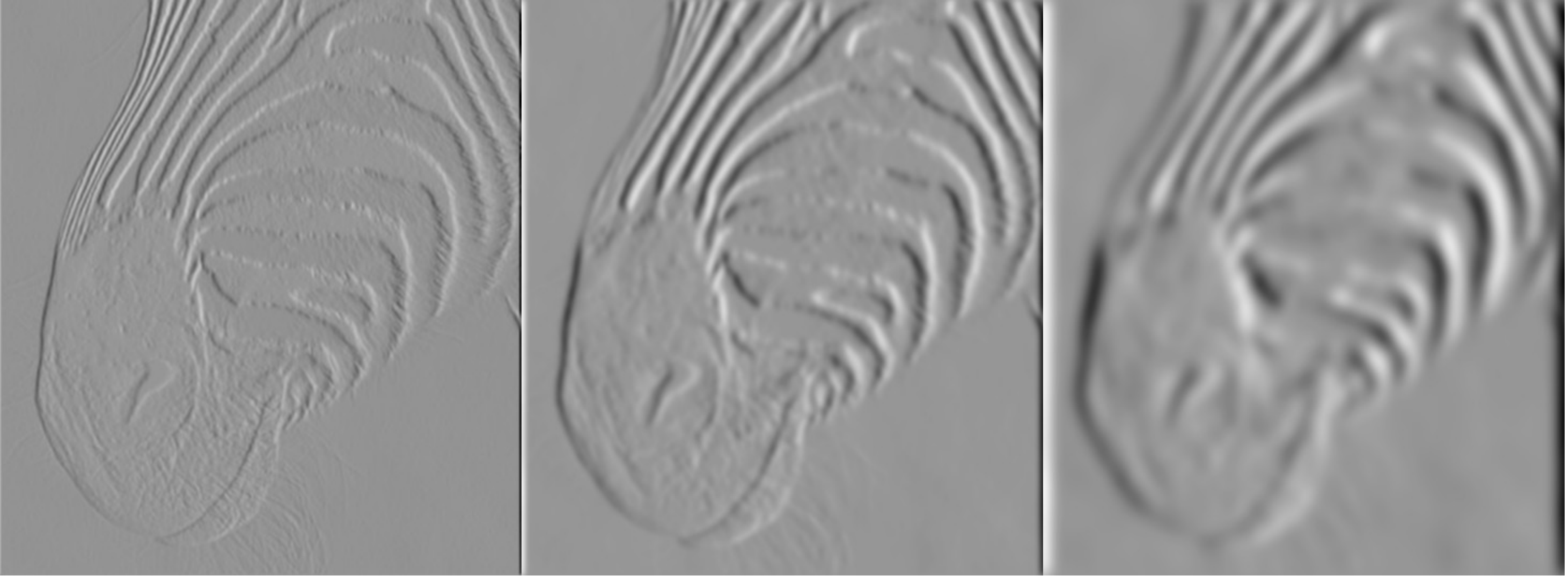
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

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

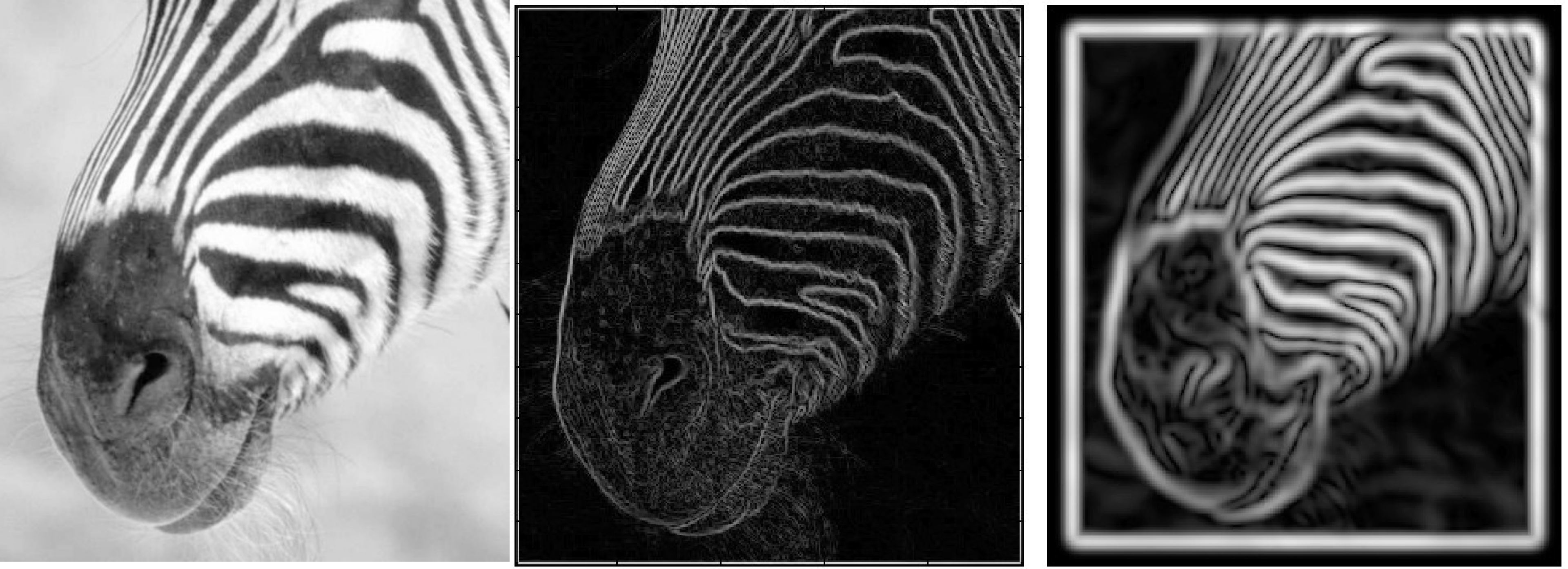






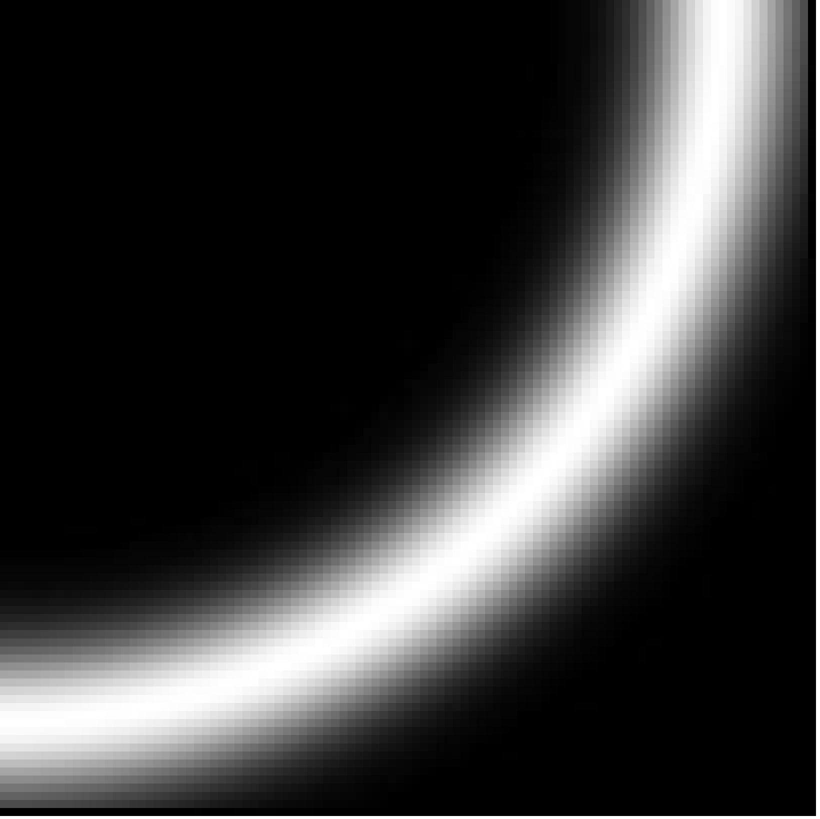
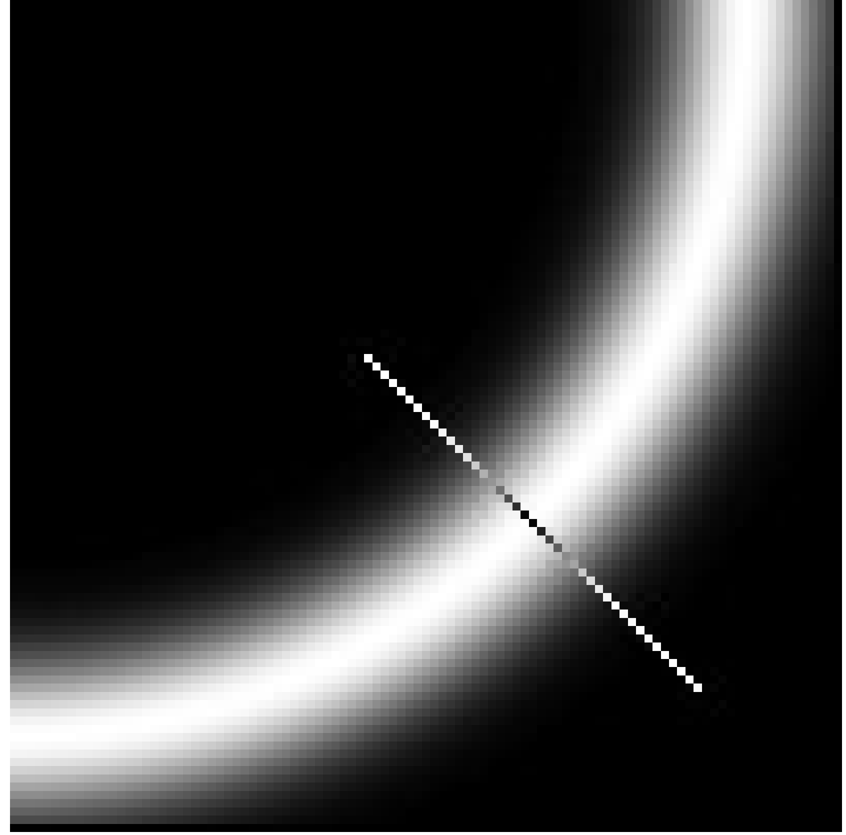
1 pixel 3 pixels 7 pixels

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



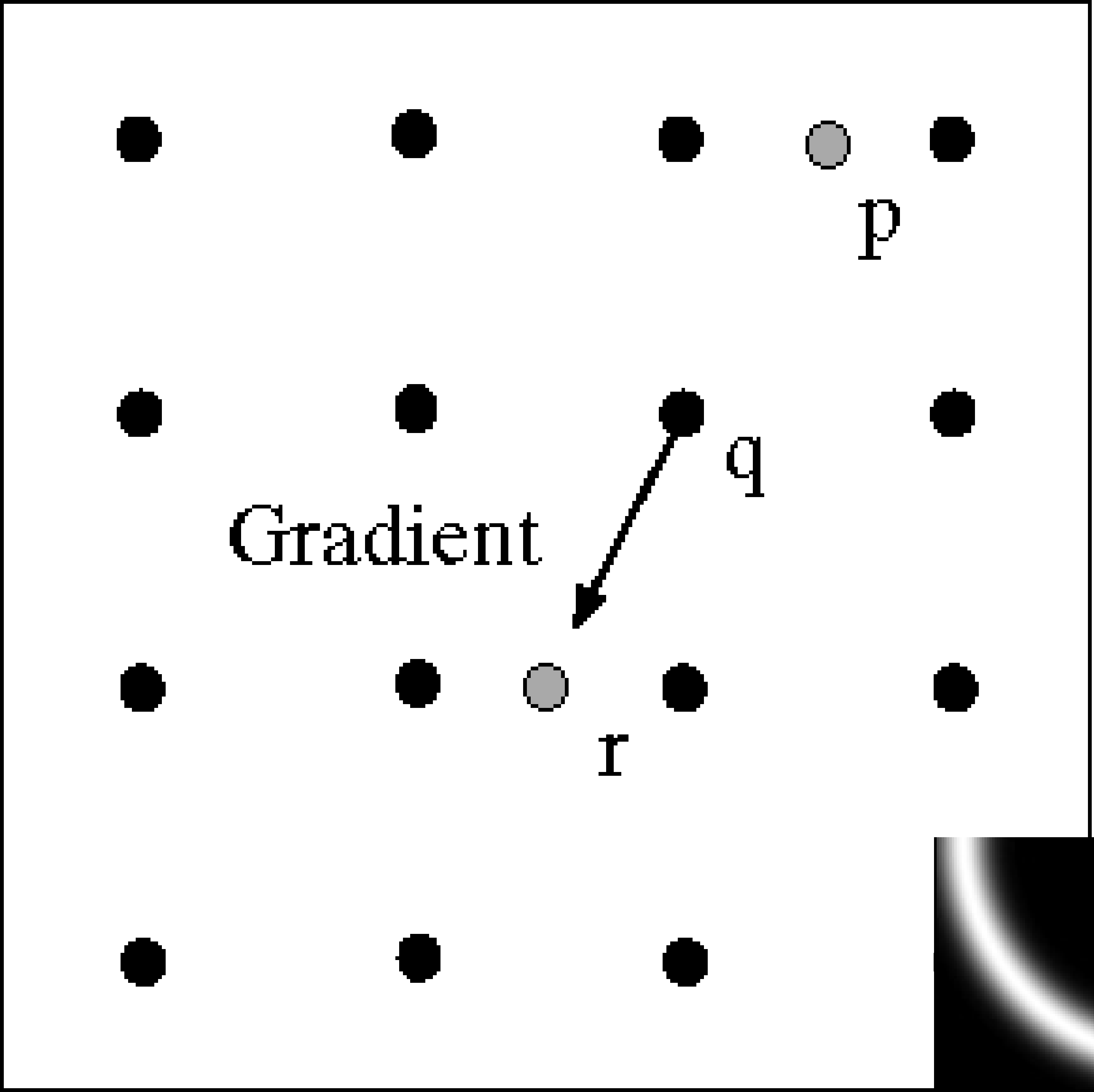
There are three major issues:

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?

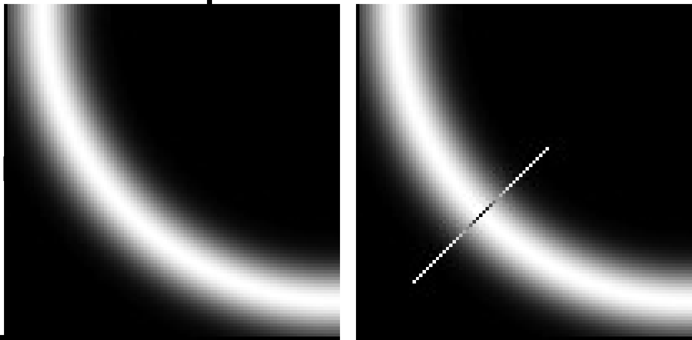
 

We wish to mark points along the curve where the magnitude is biggest. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?

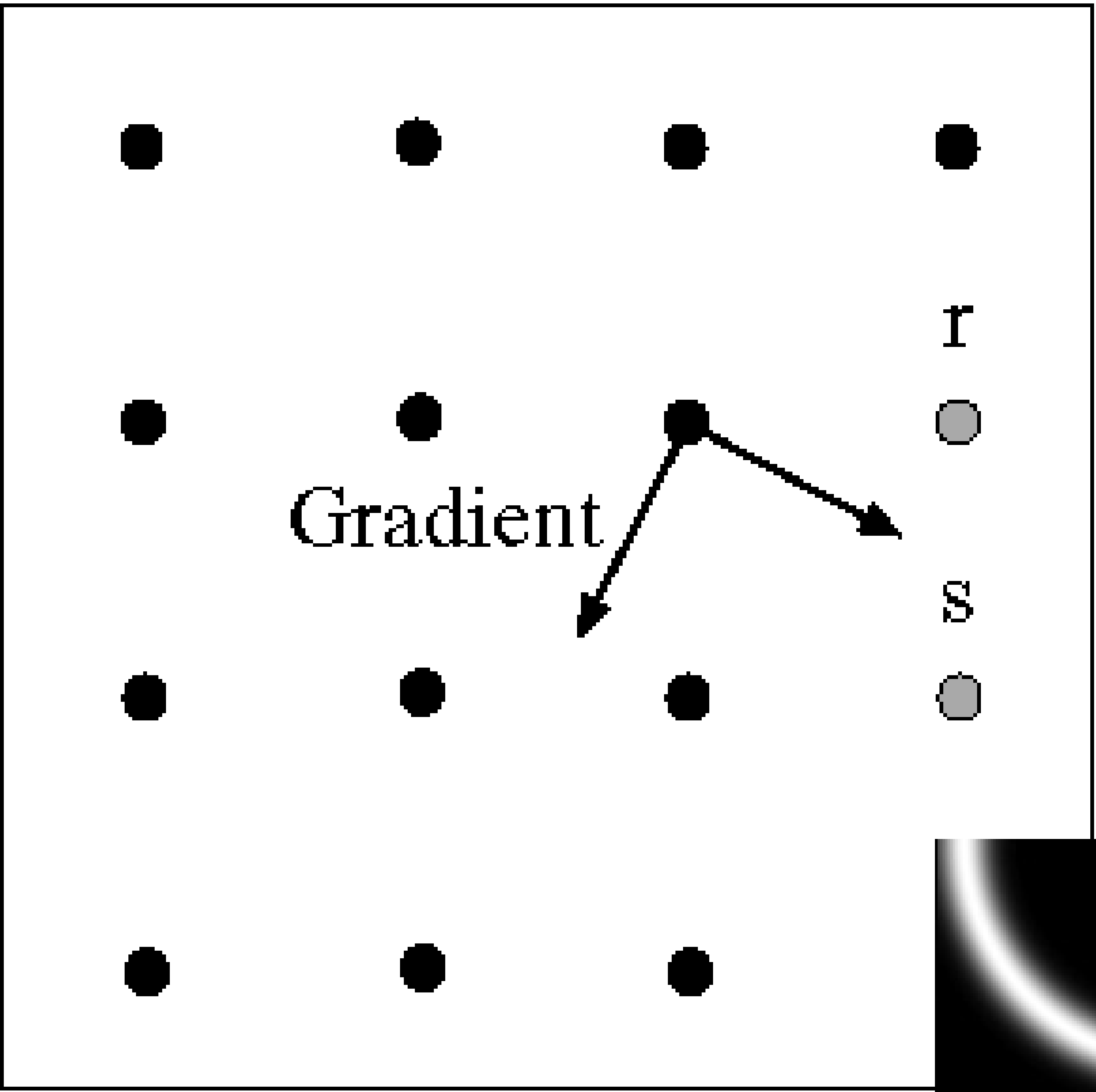
Non-maximum suppression



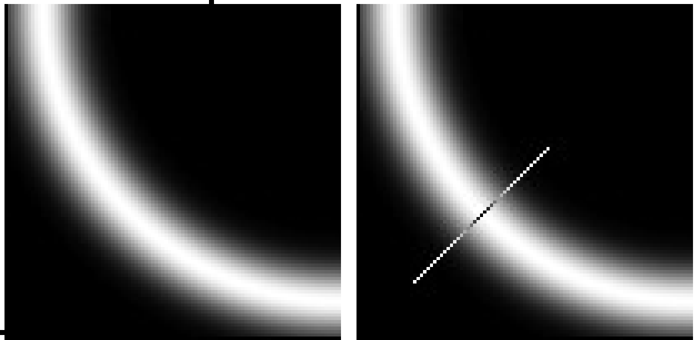
At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.



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).



```
While there are points with high gradient
that have not been visited

  Find a start point that is a local maximum in the
    direction perpendicular to the gradient
    erasing points that have been checked

  while possible, expand a chain through
    the current point by:
      1) predicting a set of next points, using
        the direction perpendicular to the gradient

      2) finding which (if any) is a local maximum
        in the gradient direction

      3) testing if the gradient magnitude at the
        maximum is sufficiently large

      4) leaving a record that the point and
        neighbours have been visited

      record the next point, which becomes the current point

  end

end
```

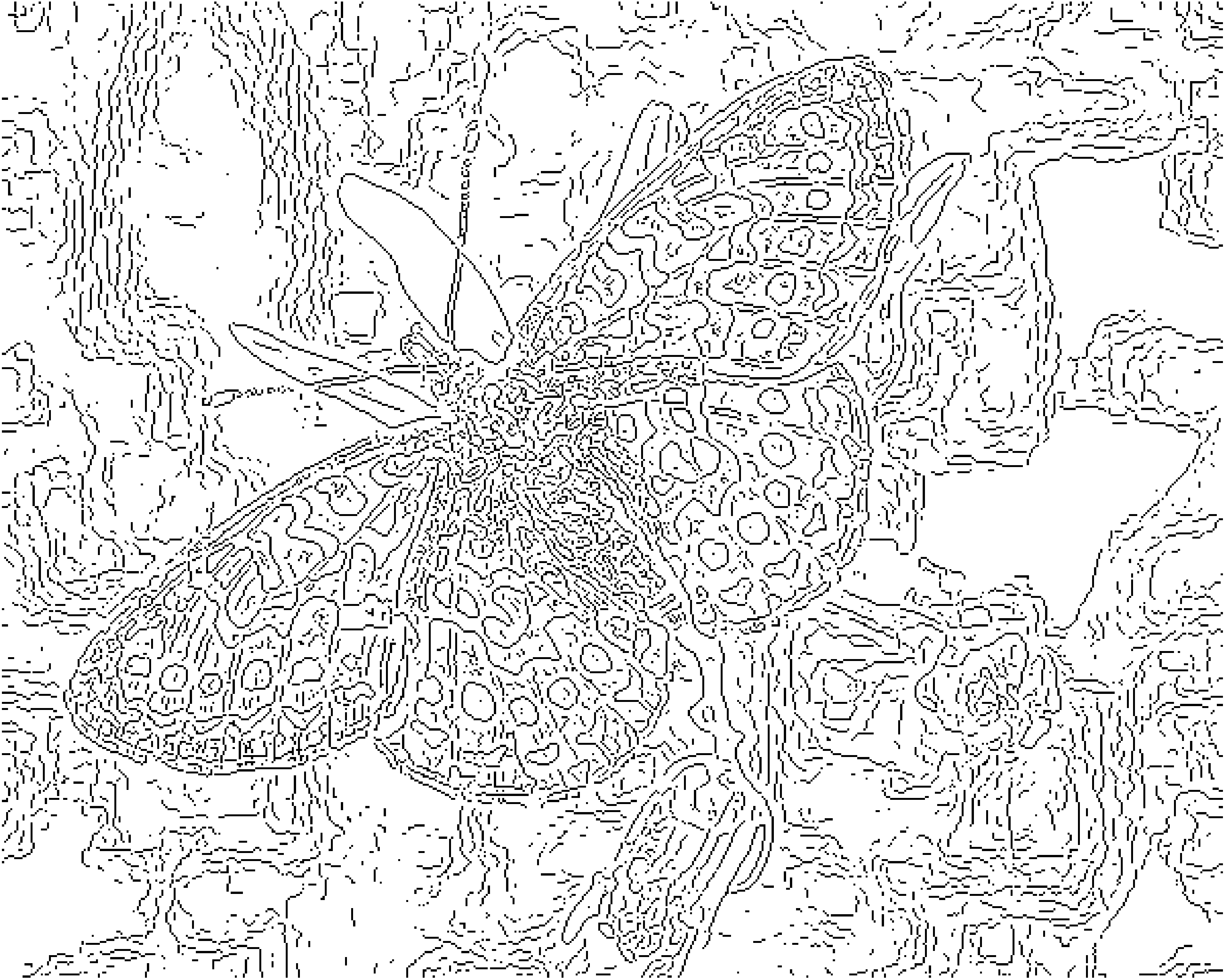
Remaining issues

- 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.

Notice

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

Scale Issues



fine scale
high
threshold

coarse
scale,
high
threshold

coarse
scale
low
threshold

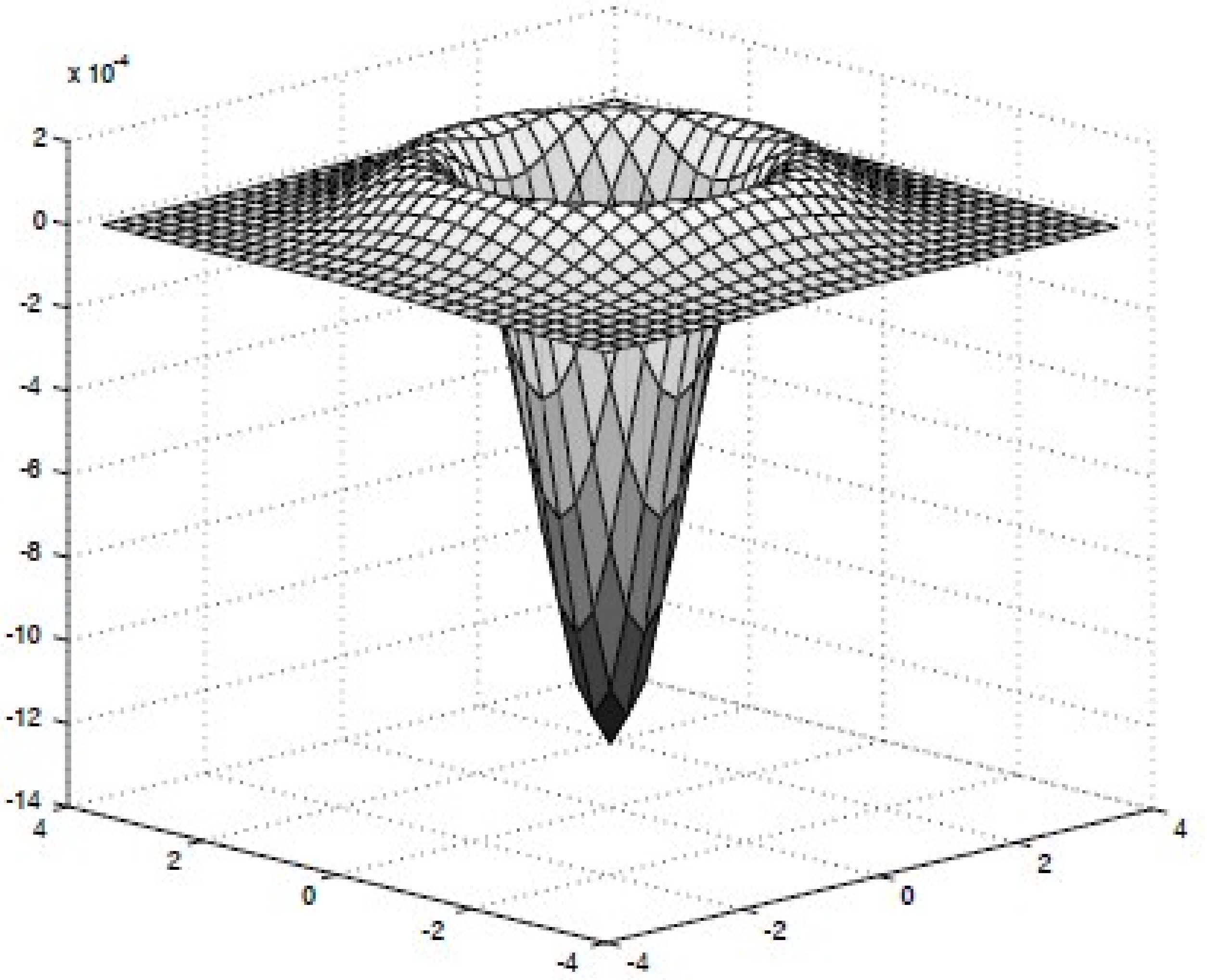
The Laplacian of Gaussian

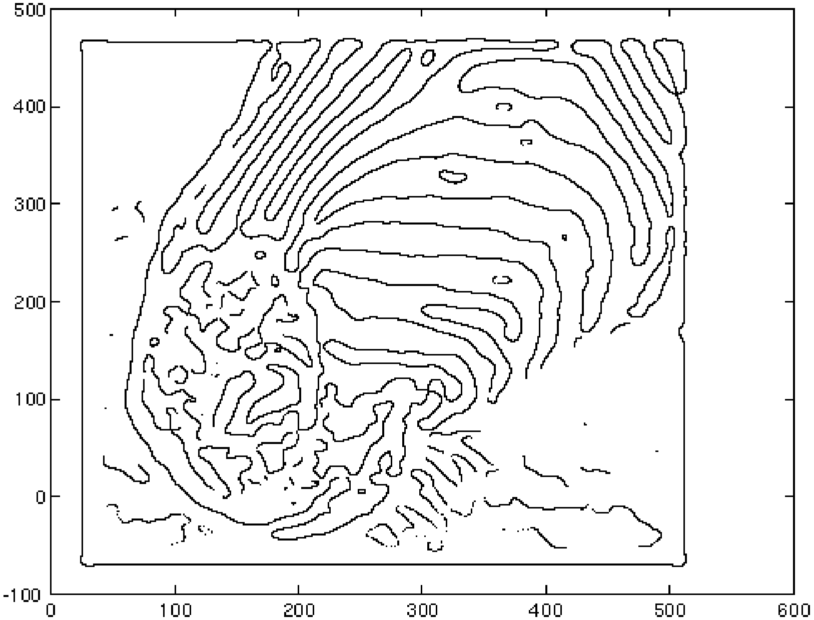
- Another way to detect an extremal first derivative is to look for a zero second derivative
- Appropriate 2D analogy is rotation invariant
  - the Laplacian

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

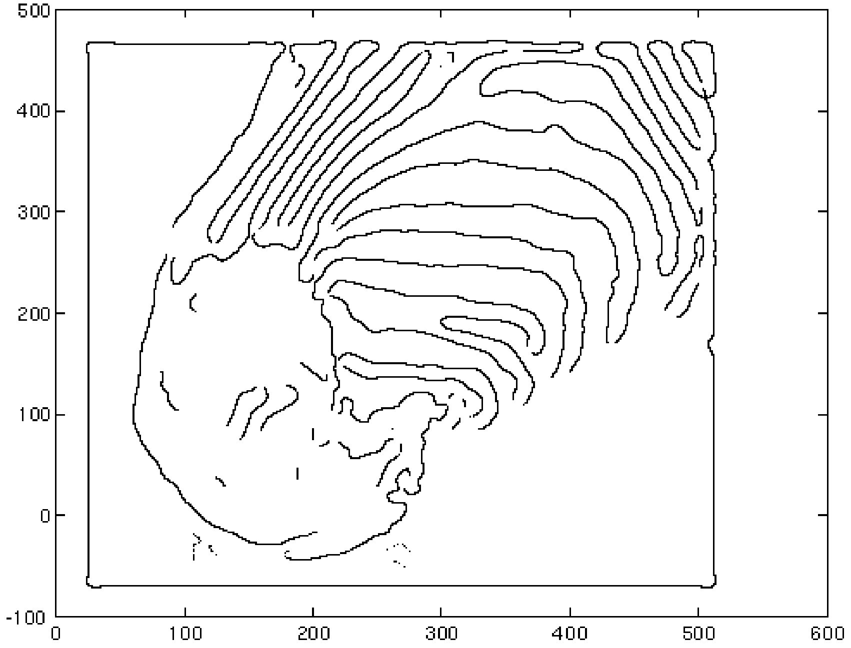
- Bad idea to apply a Laplacian without smoothing
  - smooth with Gaussian, apply Laplacian
  - this is the same as filtering with a Laplacian of Gaussian filter
- Now mark the zero points *where there is a sufficiently large derivative, and enough contrast*

The Laplacian of Gaussian





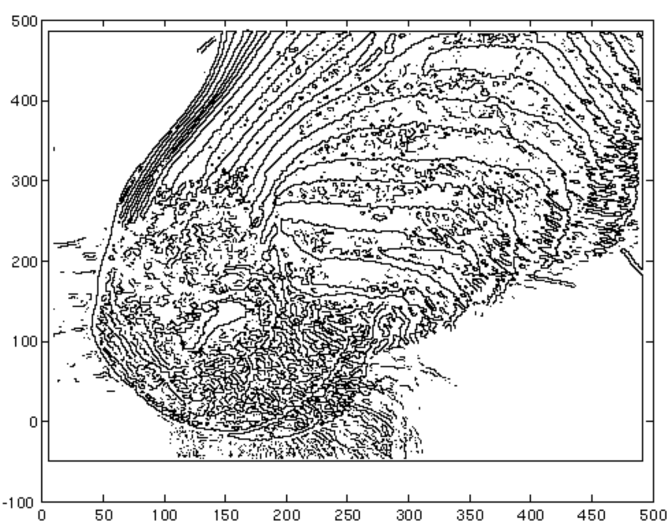
sigma=4



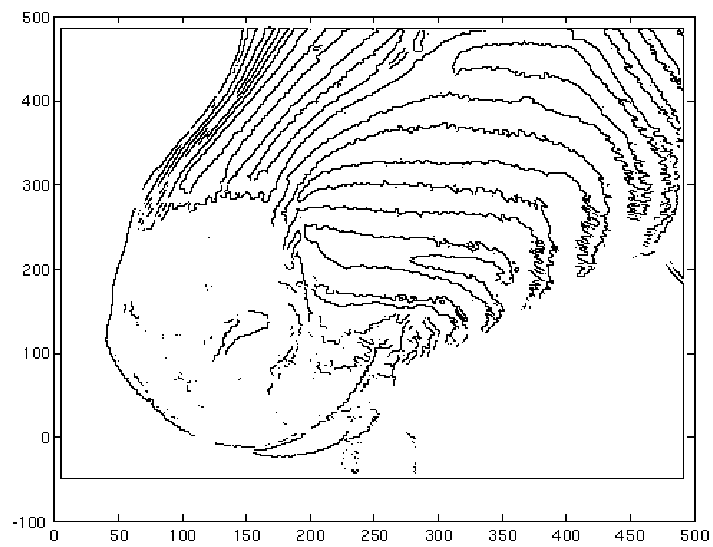
contrast=1

LOG zero crossings

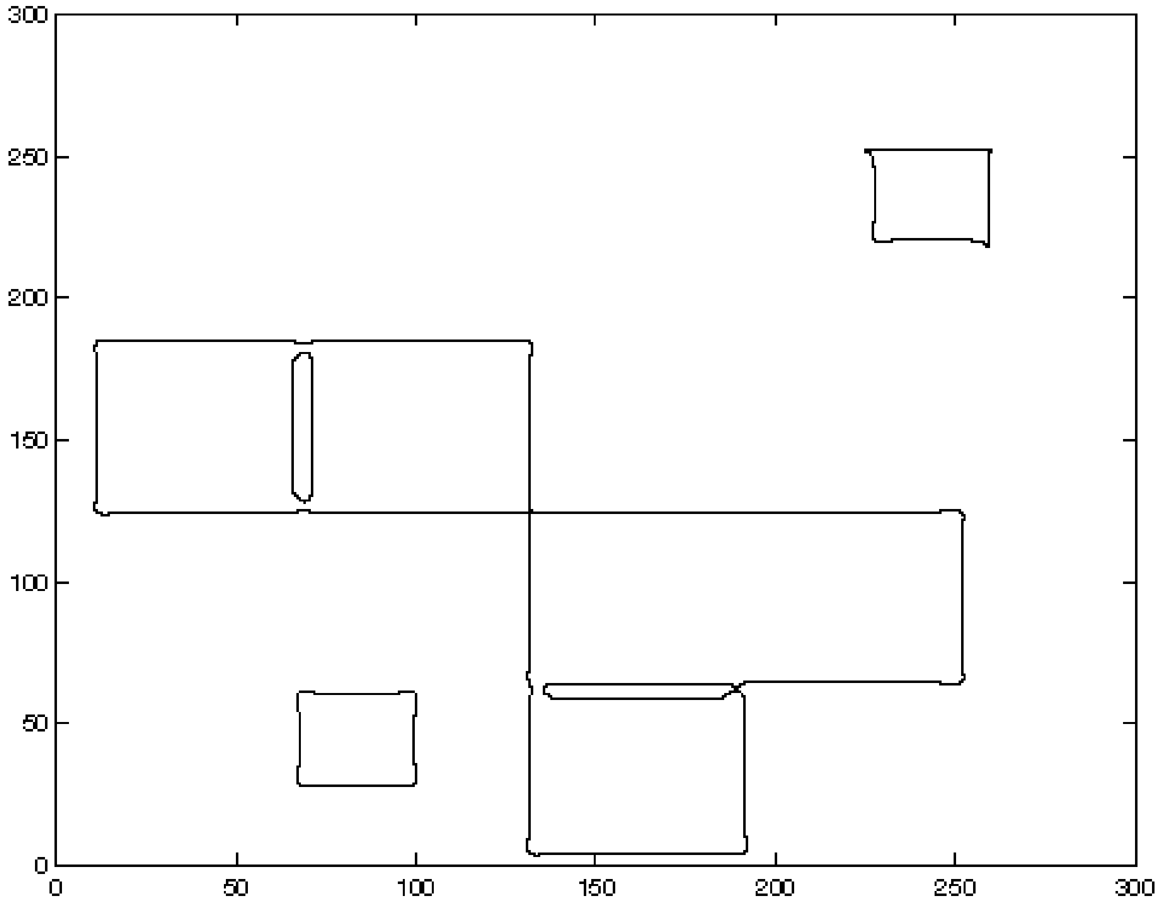
contrast=4



sigma=2



We still have unfortunate behaviour at corners

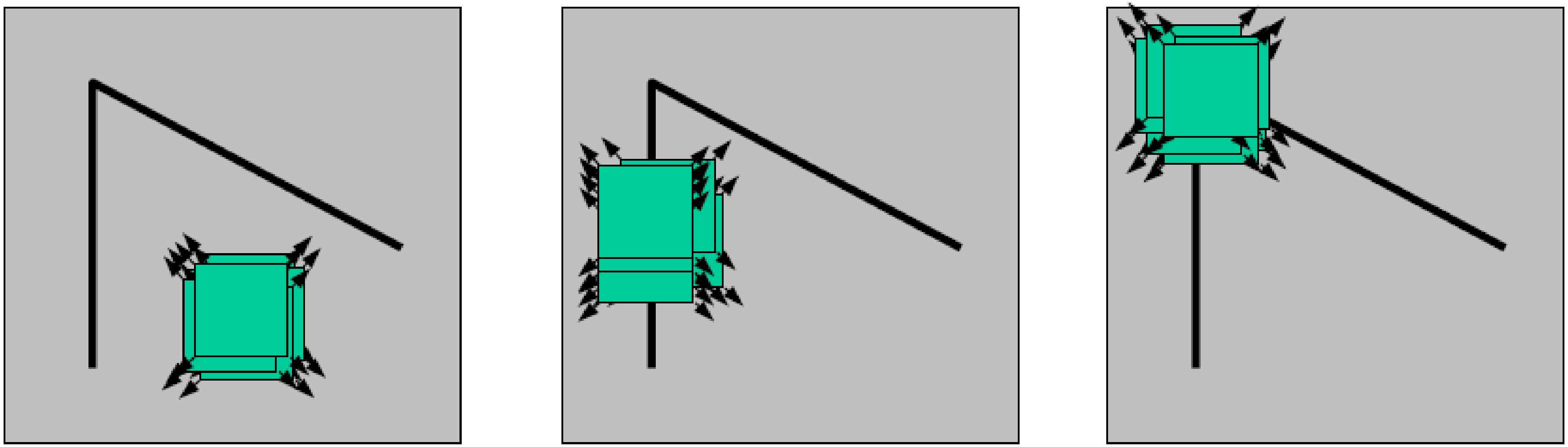


Corners

Types of windows

- Constant windows, where the grey level is approximately constant;
- Edge windows, where there is a sharp change in image brightness that runs along a single direction within the window;
- Flow windows, where there are several fine parallel stripes — say hair or fur— within the window;
- And 2D windows, where there is some form of 2D texture — say spots, or a corner — within the window.

Harris Detector: Basic Idea



"flat" region:
no change in
all directions

"edge":
no change along
the edge direction

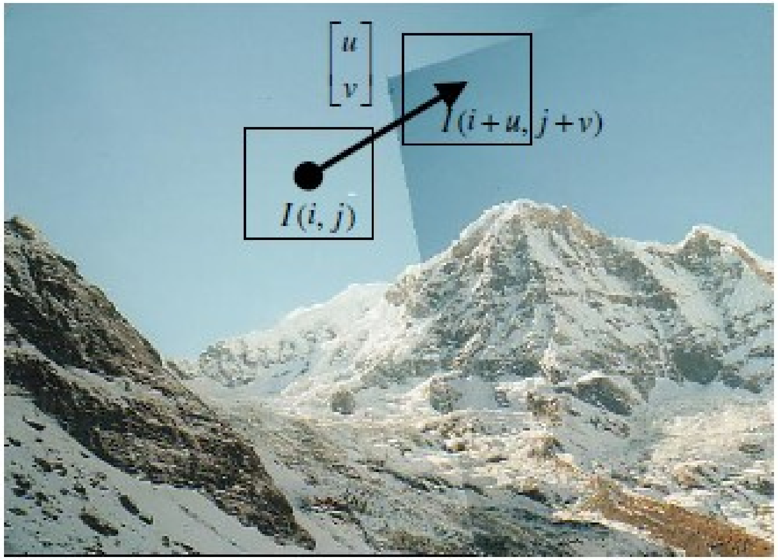
"corner":
significant change
in all directions

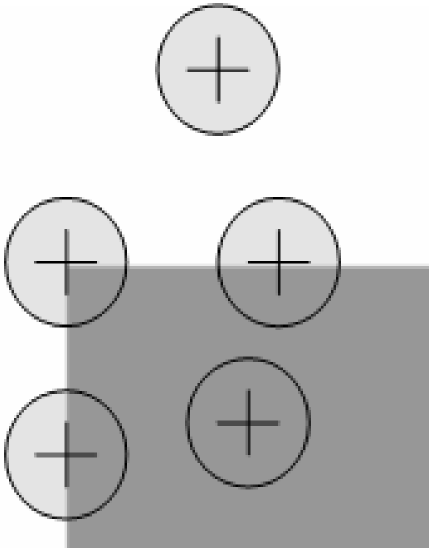
C.Harris, M.Stephens. "A Combined Corner and Edge Detector". 1988

Change of Intensity

The intensity change along some direction can be quantified by sum-of-squared-difference (SSD).

$$D(u,v) = \sum_{i,j} \left(I(i+u, j+v) - I(i,j) \right)^2$$





Change Approximation

If u and v are small, by Taylor theorem:

$$I(i+u, j+v) \approx I(i, j) + I_x u + I_y v$$

where $\quad I_x = \frac{\partial I}{\partial x} \quad and \quad I_y = \frac{\partial I}{\partial y}$

therefore

$$\begin{aligned}
\left(I(i+u, j+v) - I(i, j)\right)^2 &= \left(I(i, j) + I_x u + I_y v - I(i, j)\right)^2 \\
&= \left(I_x u + I_y v\right)^2 \\
&= I_x^2 u^2 + 2 I_x I_y uv + I_y^2 v^2 \\
&= \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}
\end{aligned}$$

Gradient Variation Matrix

$$D(u,v) = \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Matrix C characterizes how intensity changes in a certain direction.

Eigenvalue Analysis – simple case

First, consider case where:

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

This means dominant gradient directions align with x or y axis

If either λ is close to 0, then this is **not** a corner, so look for locations where both are large.

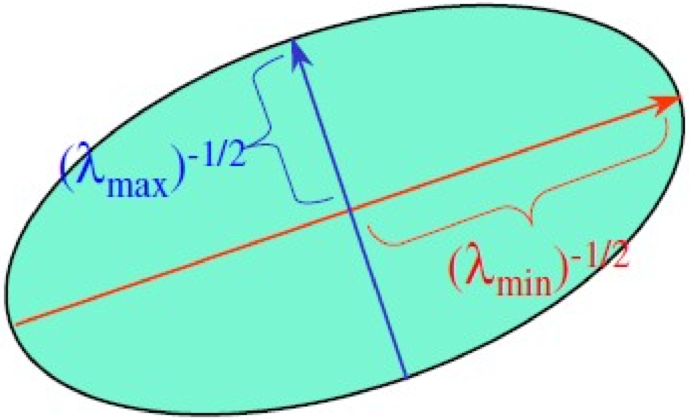
Slide credit: David Jacobs

General Case

It can be shown that since C is symmetric:

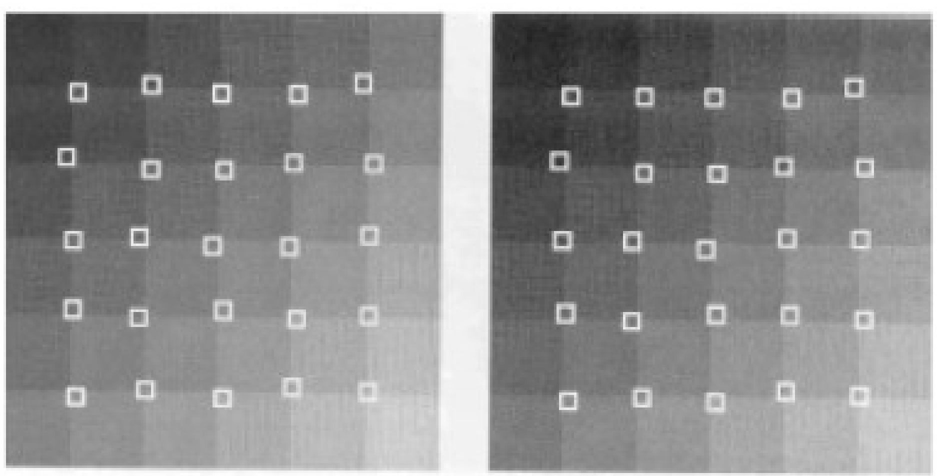
$$C = Q^T \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} Q$$

So every case is like a rotated version of the one on last slide.



Corner Detection Summary

- if the area is a region of constant intensity, both eigenvalues will be very small.
- if it contains an edge, there will be one large and one small eigenvalue (the eigenvector associated with the large eigenvalue will be parallel to the image gradient).
- if it contains edges at two or more orientations (i.e., a corner), there will be two large eigenvalues (the eigenvectors will be parallel to the image gradients).



Corner Detection Algorithm

Algorithm

Input: image f, threshold t for λ_2, size of Q

(1) Compute the gradient over the entire image f

(2) For each image point p:

(2.1) form the matrix C over the neighborhood Q of p
(2.2) compute λ_2, the smaller eigenvalue of C
(2.3) if $\lambda_2 > t$, save the coordinates of p in a list L

(3) Sort the list in decreasing order of λ_2

(4) Scanning the sorted list top to bottom: delete all the points that appear in the list that are in the same neighborhood Q with p

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