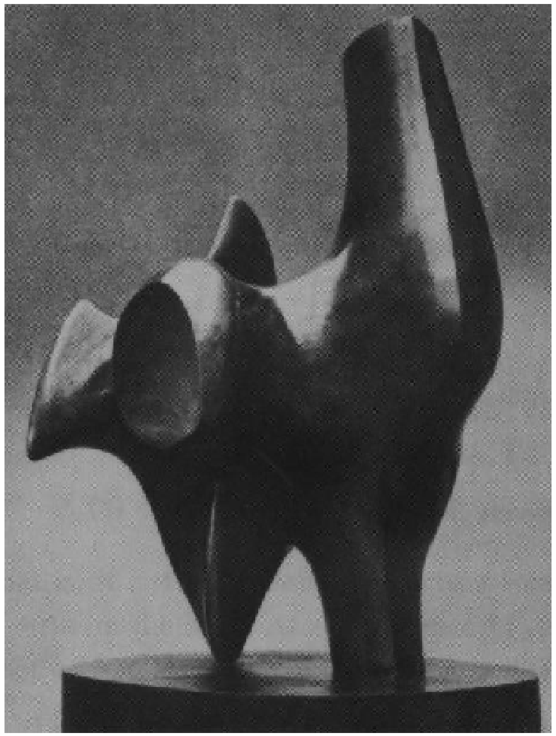
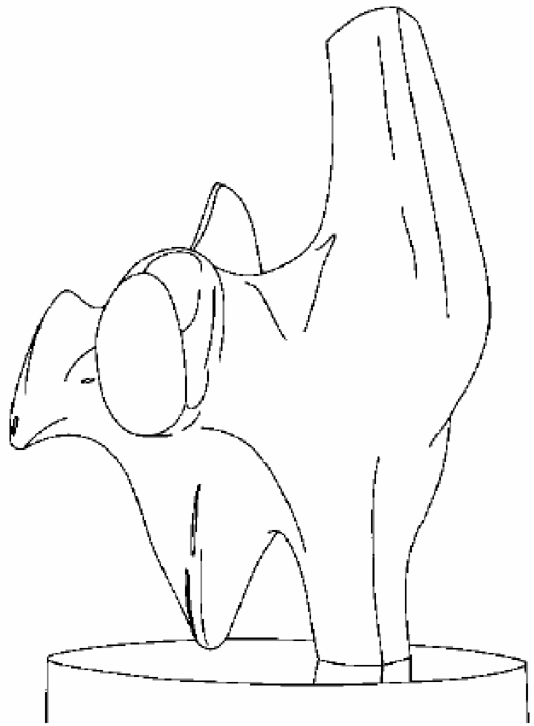
Edge Detection

Edge detection

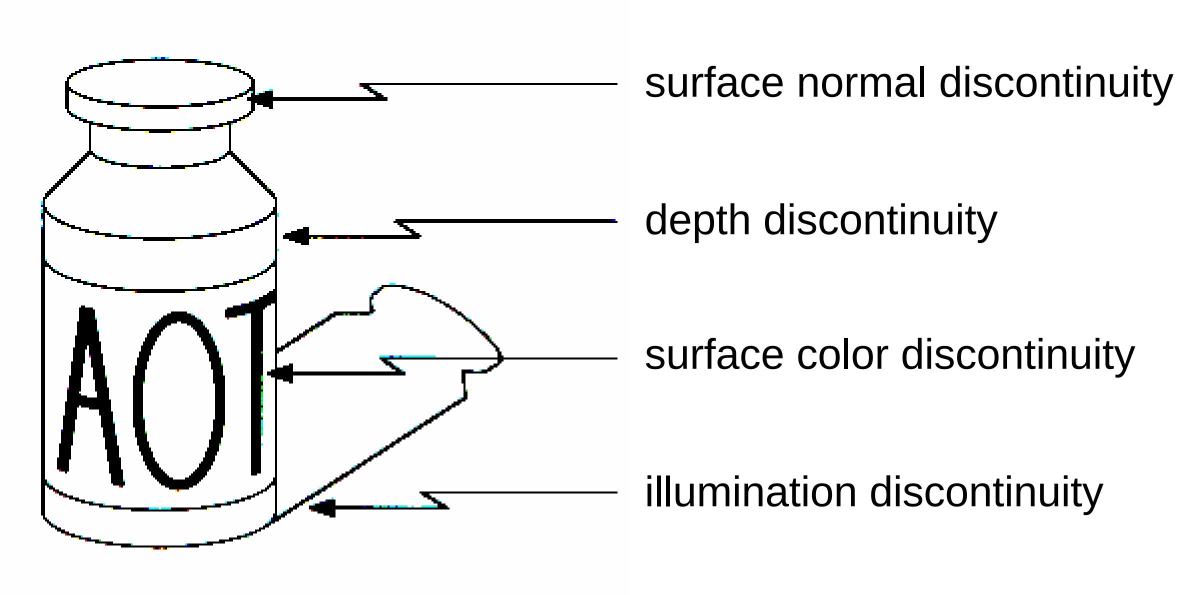




Convert a 2D image into a set of curves

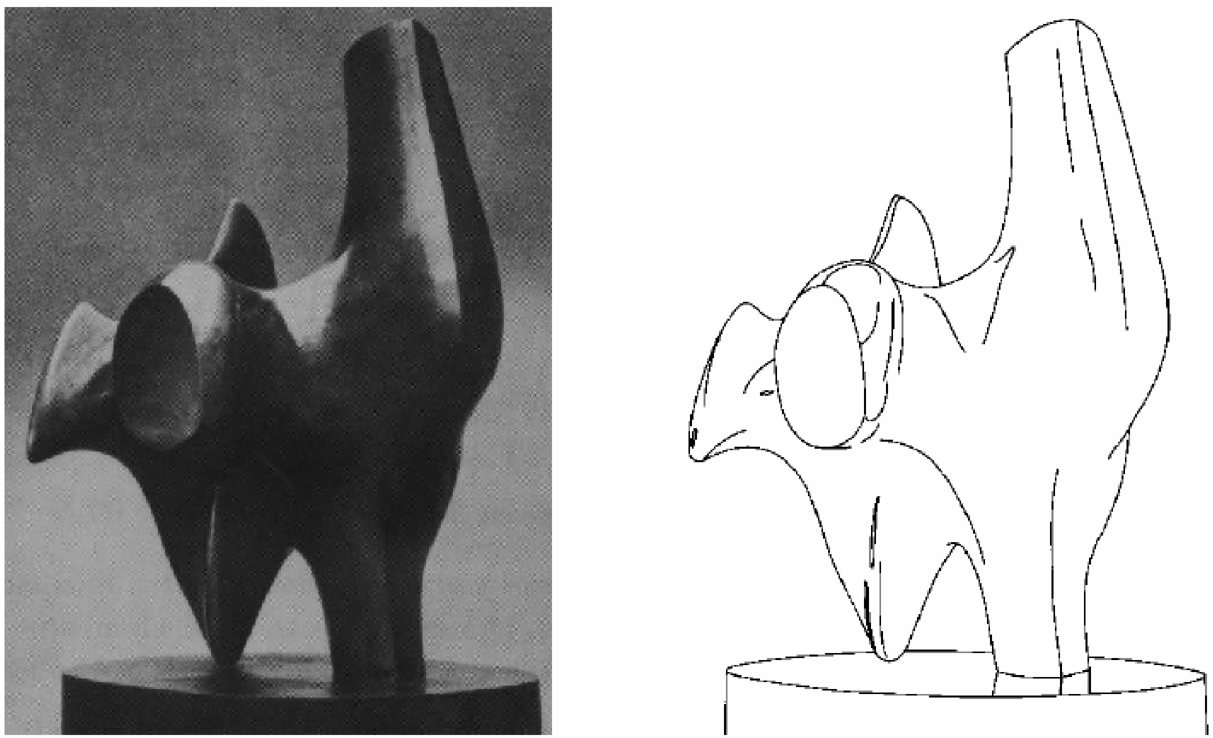
- Extracts salient features of the scene
- More compact than pixels

Origin of Edges



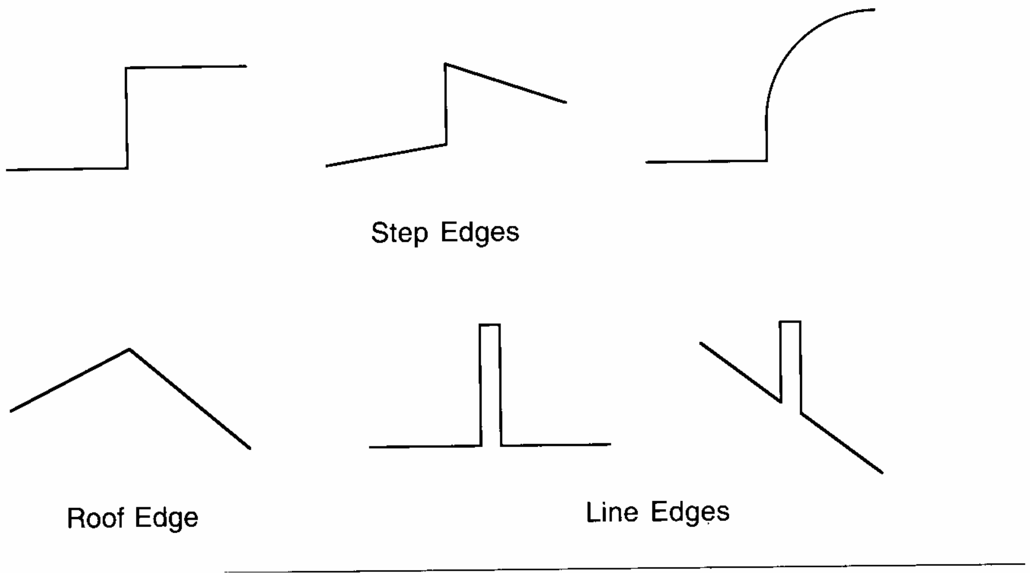
Edges are caused by a variety of factors

Edge detection



How can you tell that a pixel is on an edge?

Profiles of image intensity edges



Edge detection

1. Detection of short linear edge segments (edgels)

2. Aggregation of edgels into extended edges
   (maybe parametric description)

Edgel detection

- Difference operators
- Parametric-model matchers

Edge is Where Change Occurs

Change is measured by derivative in 1D

Biggest change, derivative has maximum magnitude

Or 2nd derivative is zero.

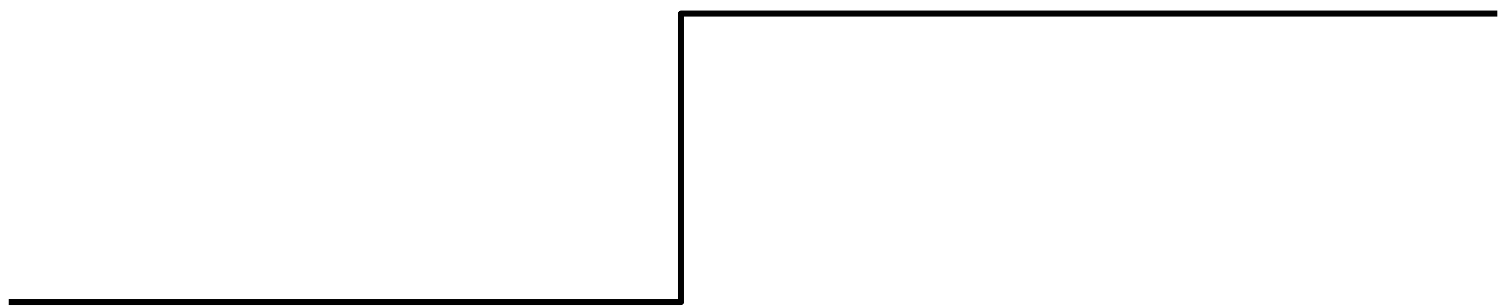
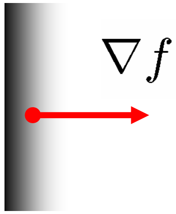


Image gradient

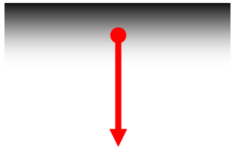
The gradient of an image:

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

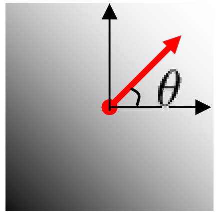
The gradient points in the direction of most rapid change in intensity



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



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



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

The 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?

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 discrete gradient

How can we differentiate a *digital* image f[x,y]?

- Option 1: reconstruct a continuous image, then take gradient
- 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 cross-correlation?

H

The Sobel operator

Better approximations of the derivatives exist

- The *Sobel* operators below are very commonly used

$$\frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad \frac{1}{8}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

$$s_x \qquad\qquad s_y$$

- The standard defn. of the Sobel operator omits the 1/8 term
  - doesn't make a difference for edge detection
  - the 1/8 term **is** needed to get the right gradient value, however

Gradient operators

$$\Delta_1 \quad \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix} \qquad\qquad \Delta_2 \quad \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix}$$

(a)

$$\Delta_1 \quad \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix} \qquad\qquad \Delta_2 \quad \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

(b)

$$\Delta_1 \quad \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \qquad\qquad \Delta_2 \quad \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

(c)

$$\Delta_1 \quad \begin{matrix} -3 & -1 & 1 & 3 \\ -3 & -1 & 1 & 3 \\ -3 & -1 & 1 & 3 \\ -3 & -1 & 1 & 3 \end{matrix} \qquad\qquad \Delta_2 \quad \begin{matrix} 3 & 3 & 3 & 3 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -3 & -3 & -3 & -3 \end{matrix}$$

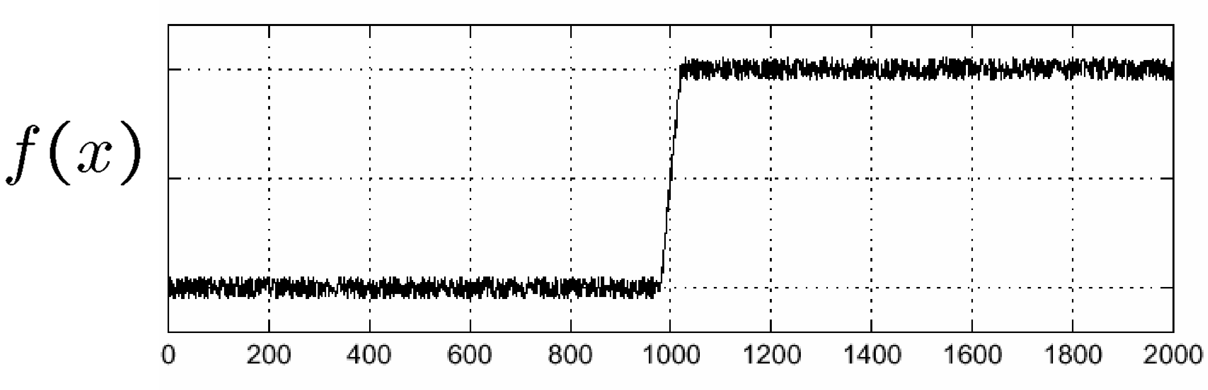
(d)

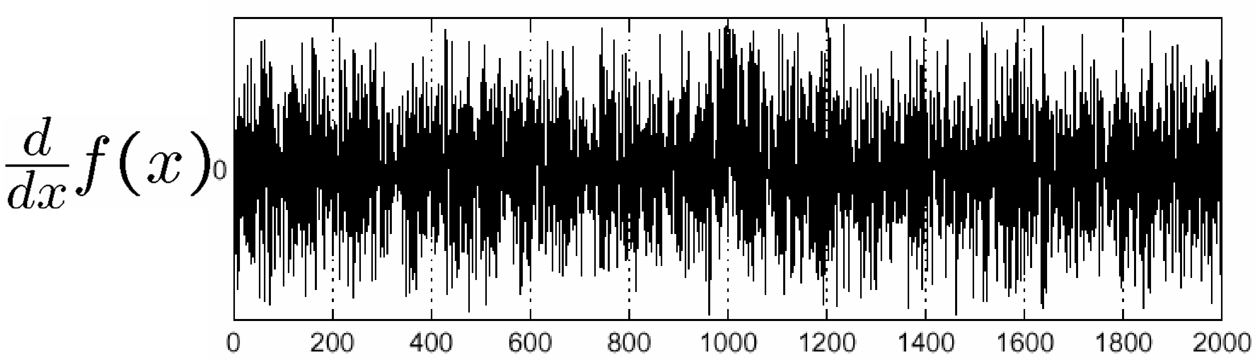
(a): Roberts' cross operator (b): 3x3 Prewitt operator
(c): Sobel operator (d) 4x4 Prewitt operator

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?

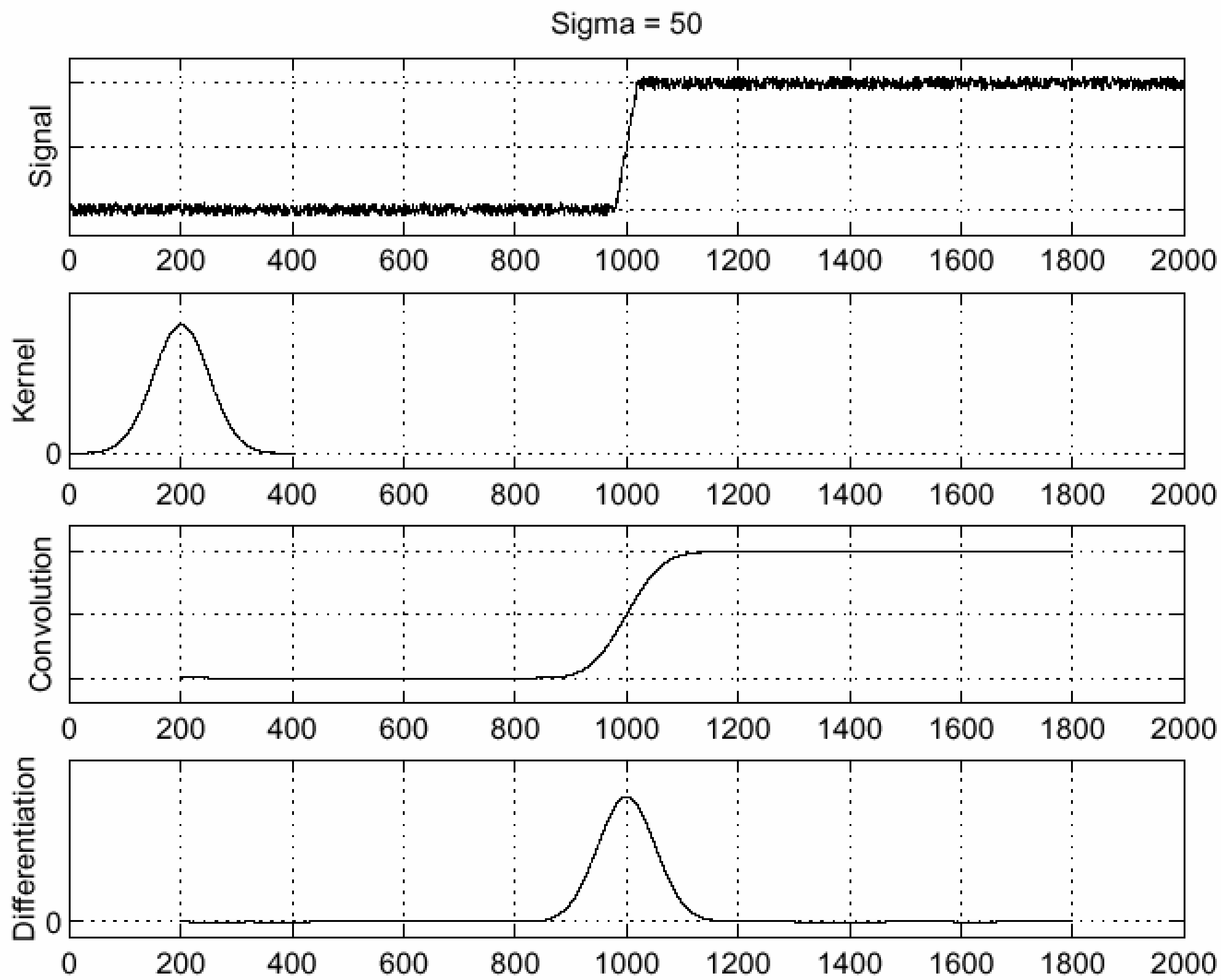
Solution: smooth first

f

h

$h \star f$

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



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

Derivative theorem of convolution

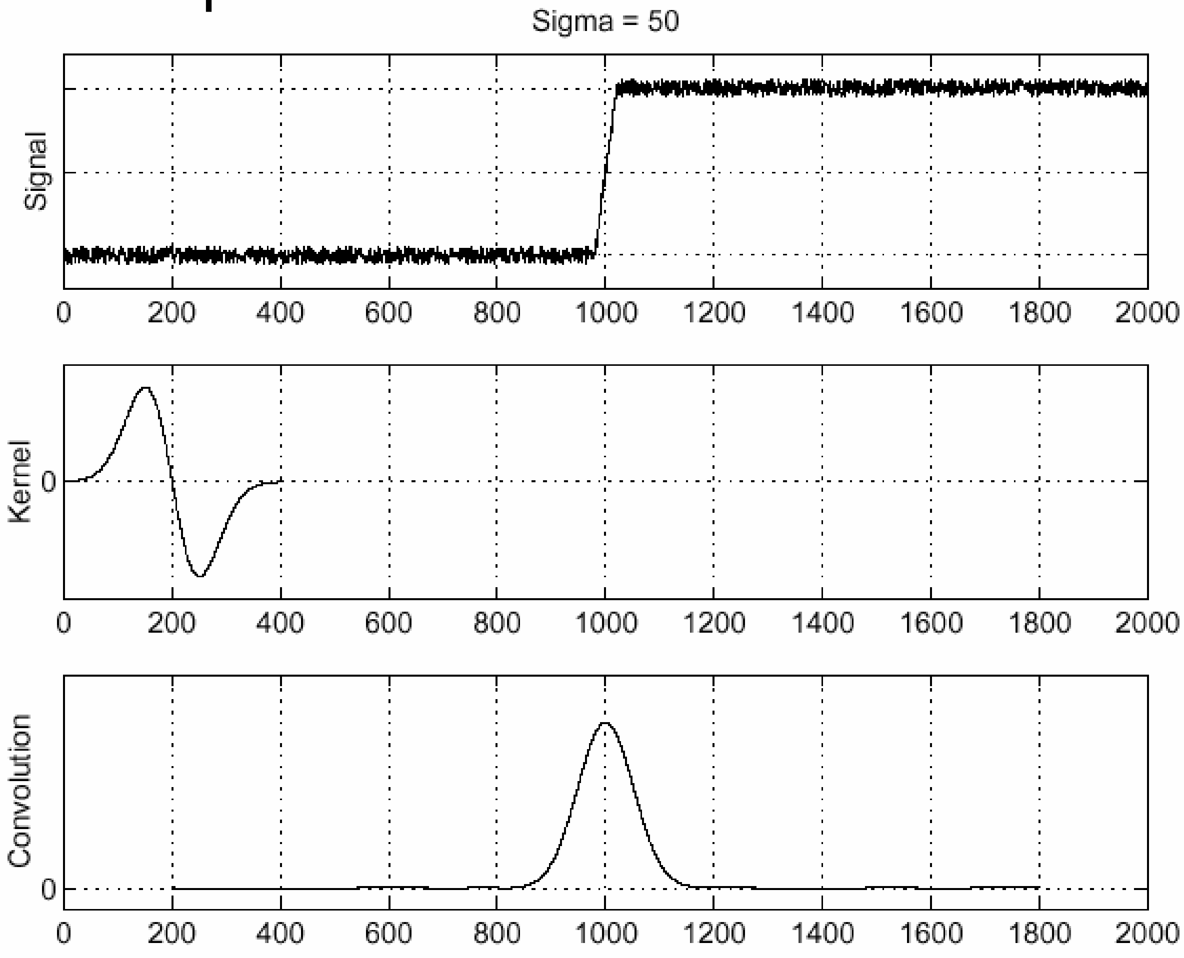
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

This saves us one operation:

$$f$$

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

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



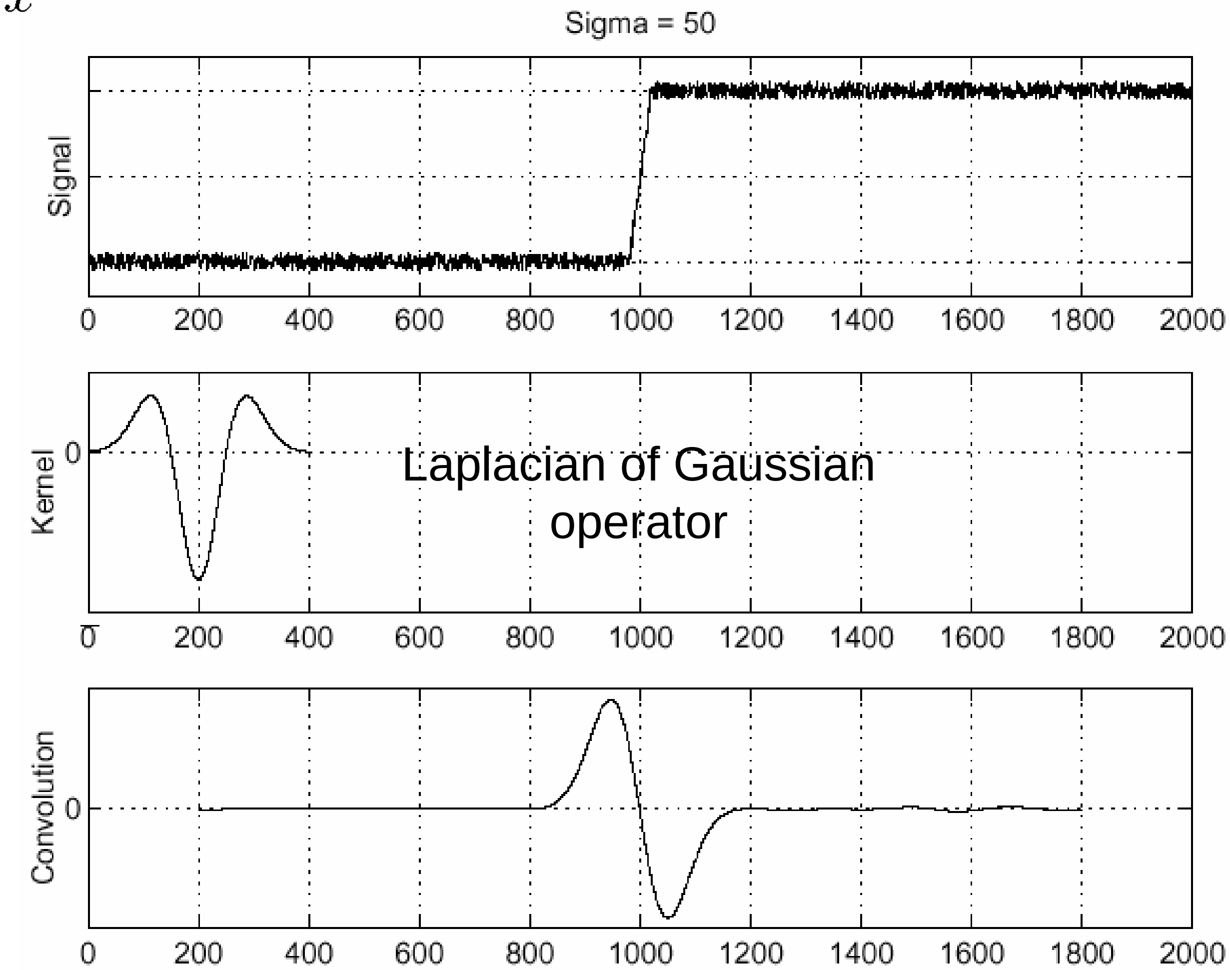
Laplacian of Gaussian

Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

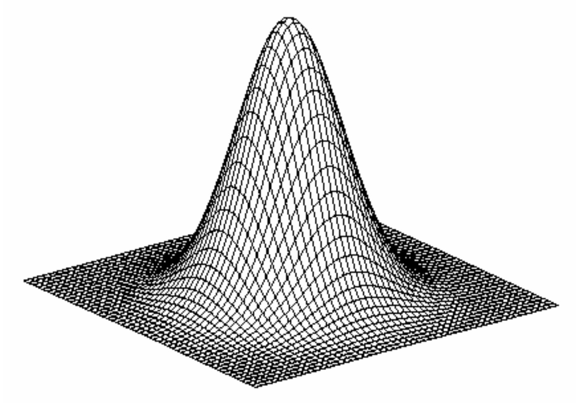
$\frac{\partial^2}{\partial x^2}h$

$(\frac{\partial^2}{\partial x^2}h) \star f$



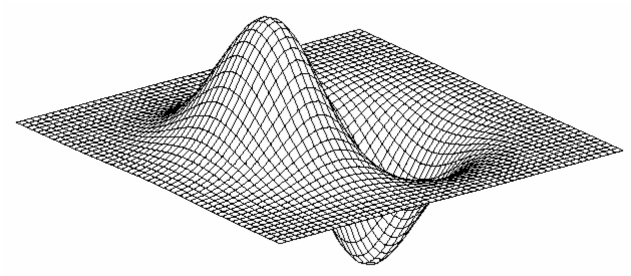
Where is the edge? Zero-crossings of bottom graph

2D edge detection filters



Gaussian

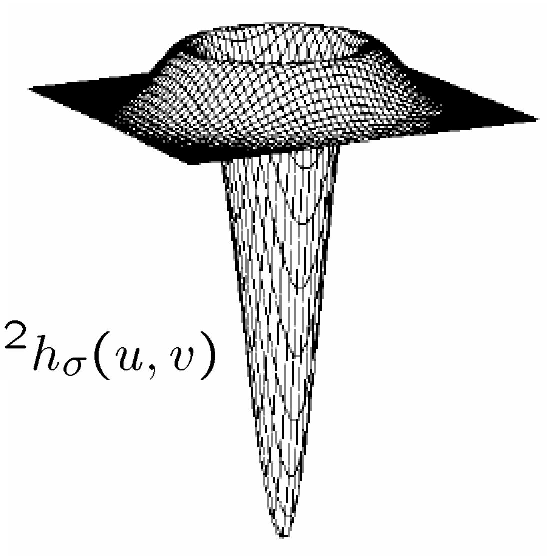
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

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

Laplacian of Gaussian



$$\nabla^2 h_\sigma(u,v)$$

∇^2 is the **Laplacian** operator:

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

Optimal Edge Detection: Canny

Assume:

- Linear filtering
- Additive iid Gaussian noise

Edge detector should have:

- Good Detection. Filter responds to edge, not noise.
- Good Localization: detected edge near true edge.
- Single Response: one per edge.

Optimal Edge Detection: Canny (continued)

Optimal Detector is approximately Derivative of Gaussian.

Detection/Localization trade-off

- More smoothing improves detection
- And hurts localization.

This is what you might guess from (detect change) + (remove noise)

The Canny edge detector

original image (Lena)

The Canny edge detector

norm of the gradient

The Canny edge detector

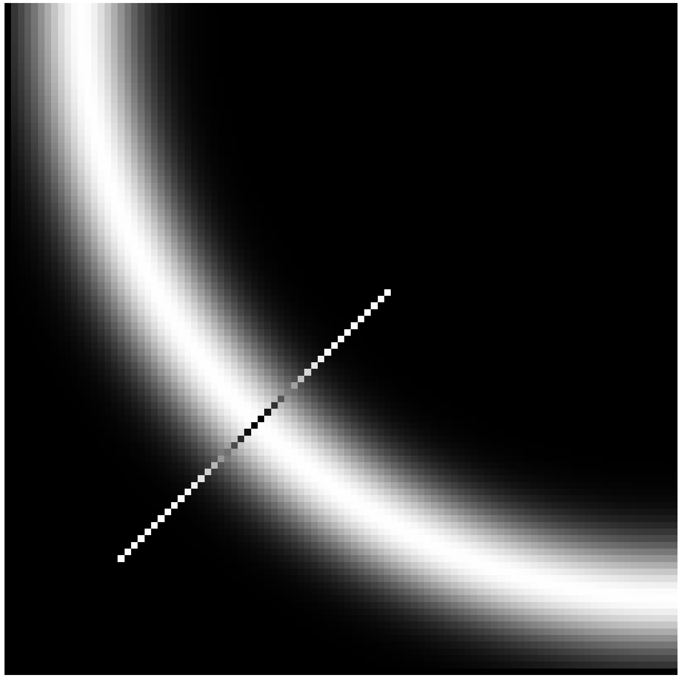
thresholding

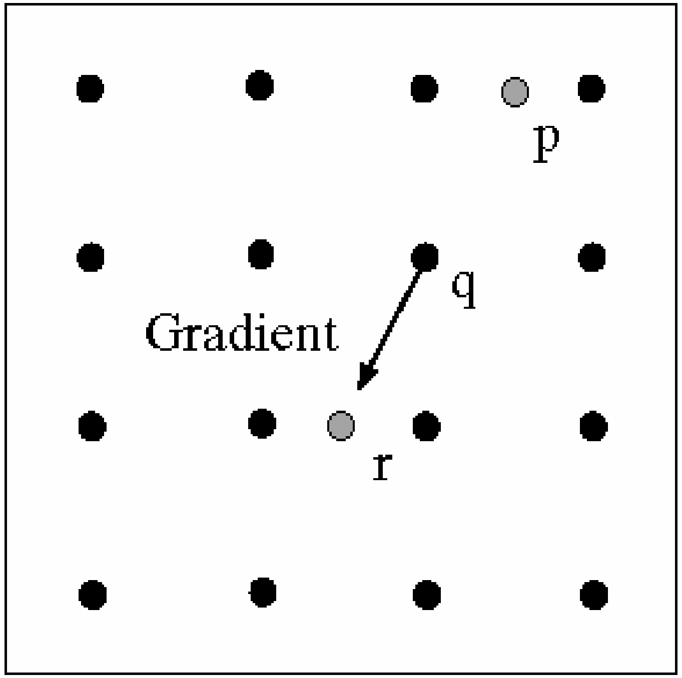
The Canny edge detector

thinning

(non-maximum suppression)

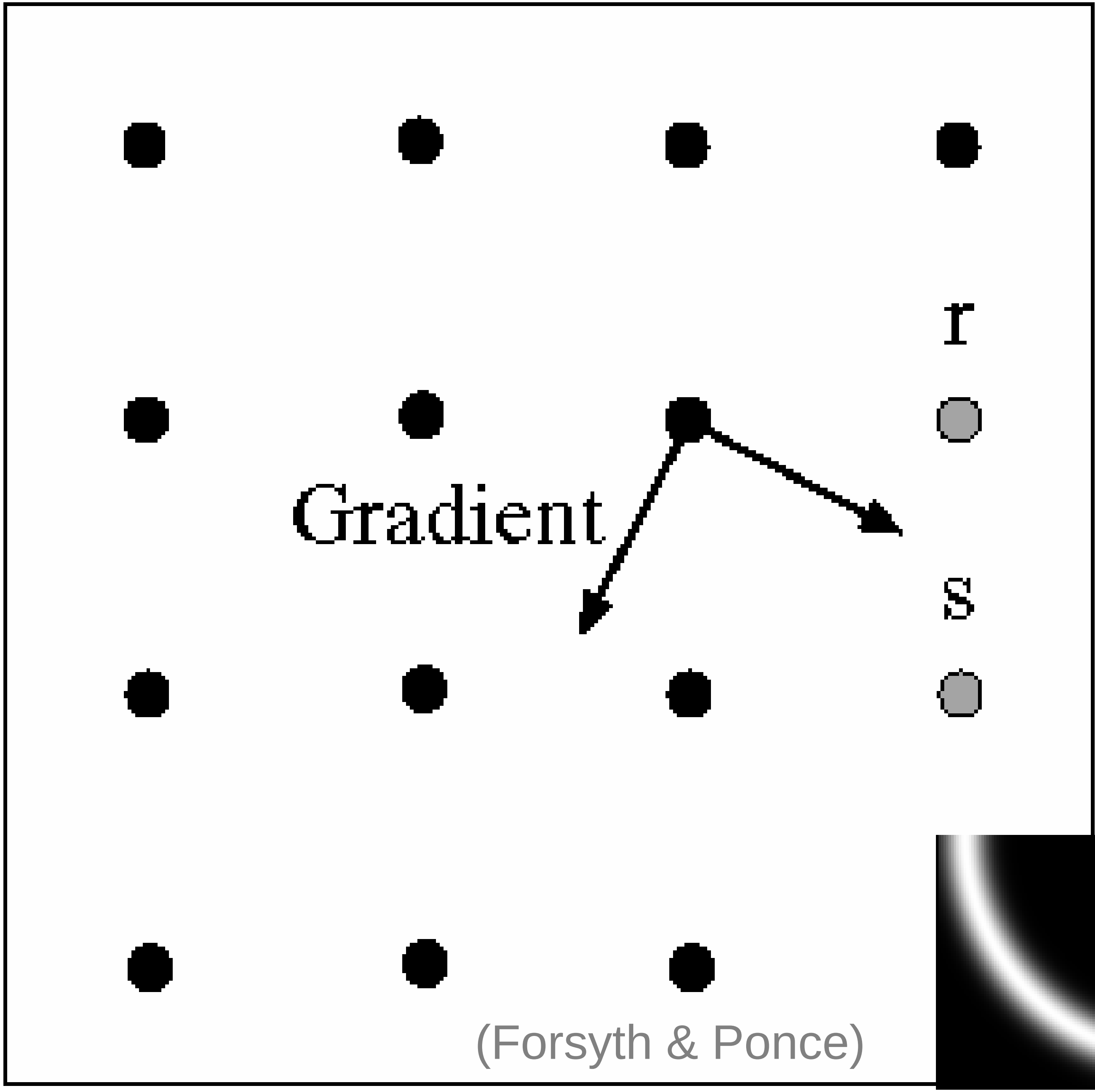
Non-maximum suppression





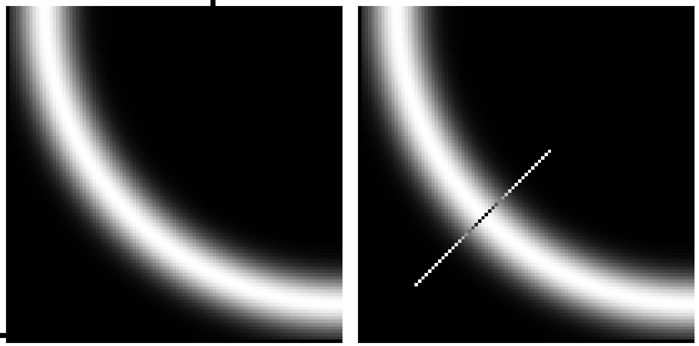
Check if pixel is local maximum along gradient direction

- requires checking interpolated pixels p and r



Predicting the next edge point

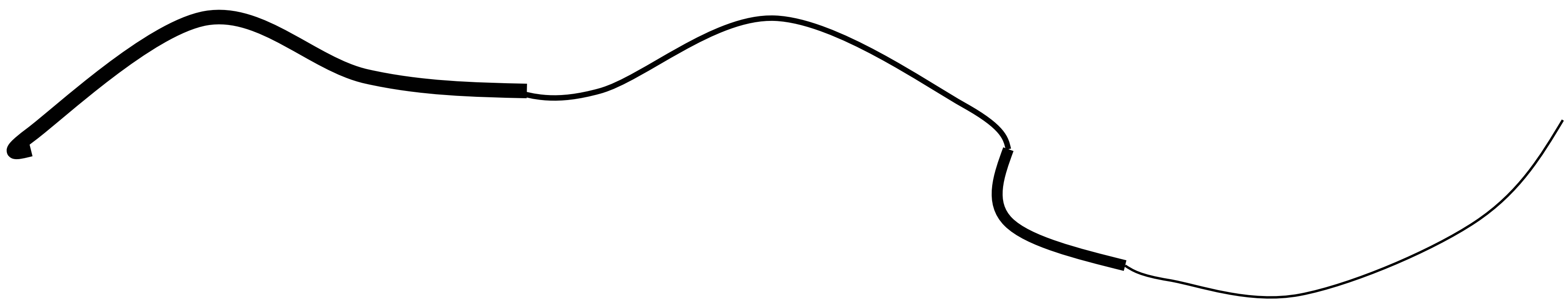
Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).



Hysteresis

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.



Effect of σ (Gaussian kernel size)

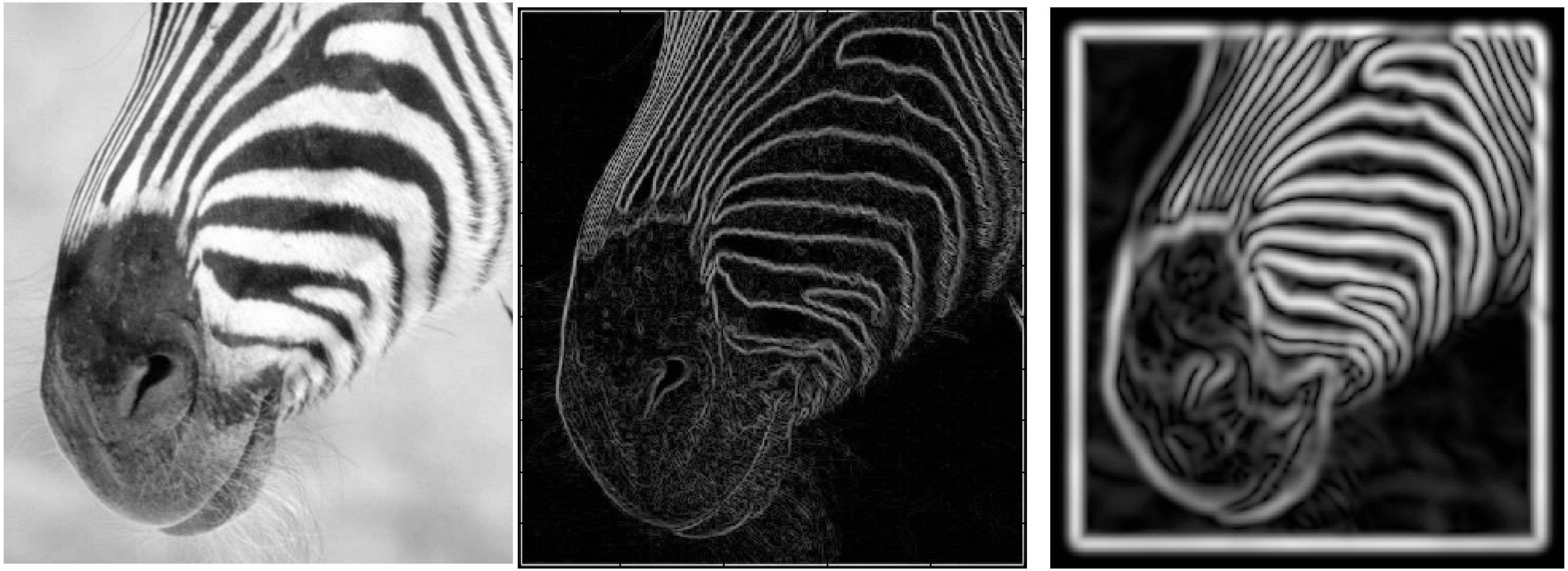
original

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



Scale

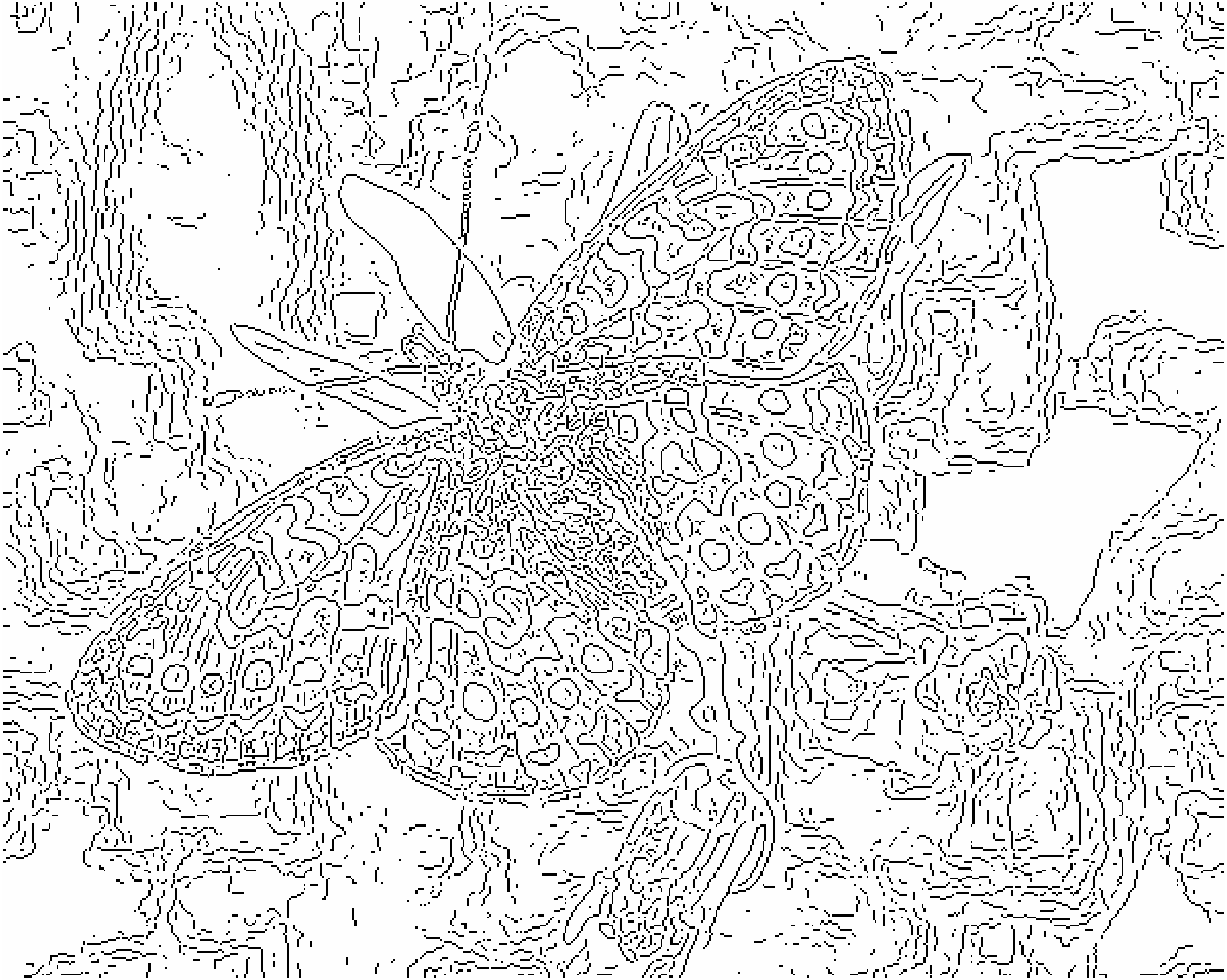
Smoothing

Eliminates noise edges.

Makes edges smoother.

Removes fine detail.

(Forsyth & Ponce)

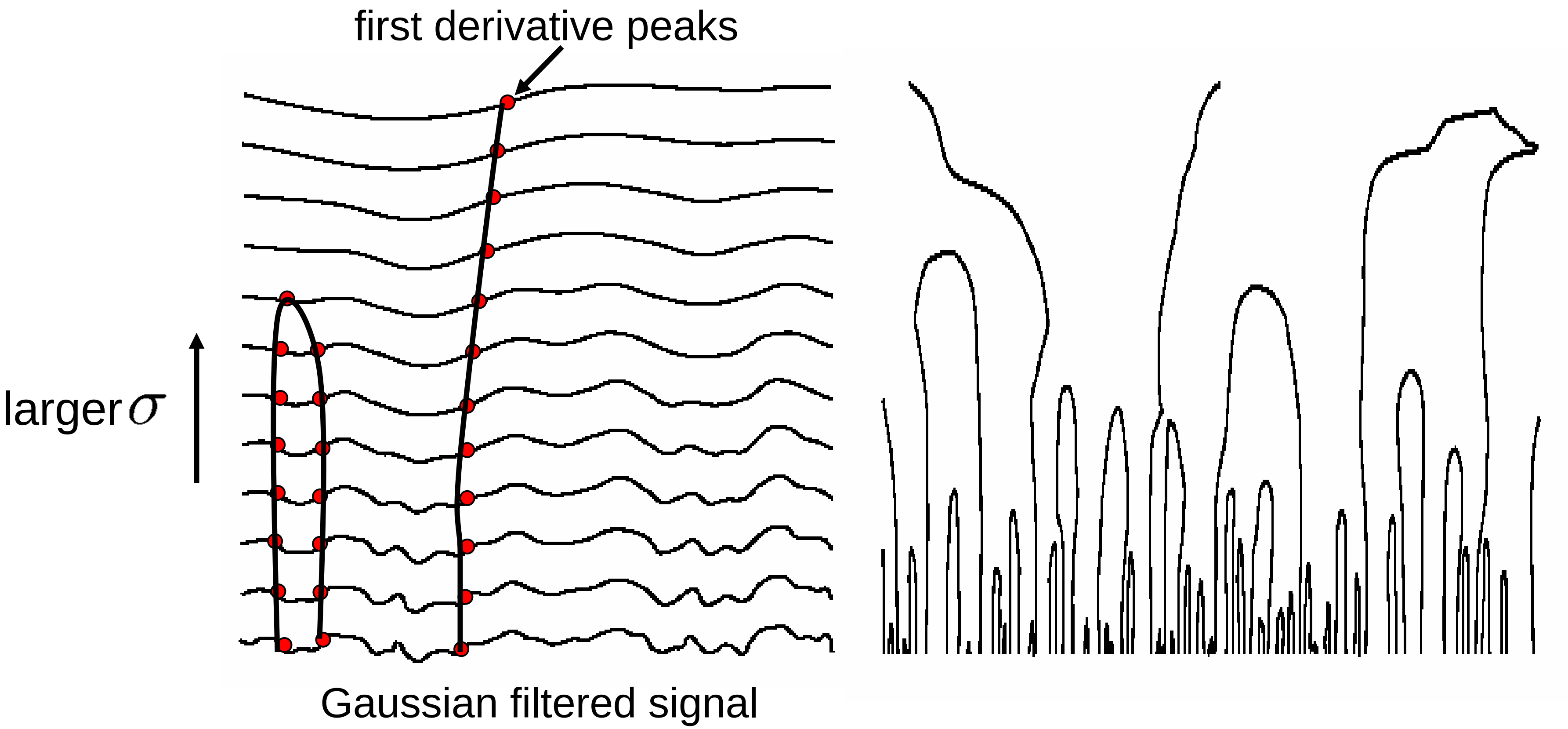


fine scale
high
threshold

coarse
scale,
high
threshold

coarse
scale
low
threshold

Scale space (Witkin 83)



Properties of scale space (w/ Gaussian smoothing)

- edge position may shift with increasing scale (σ)
- two edges may merge with increasing scale
- an edge may ***not*** split into two with increasing scale

Edge detection by subtraction

original

Edge detection by subtraction

smoothed (5x5 Gaussian)

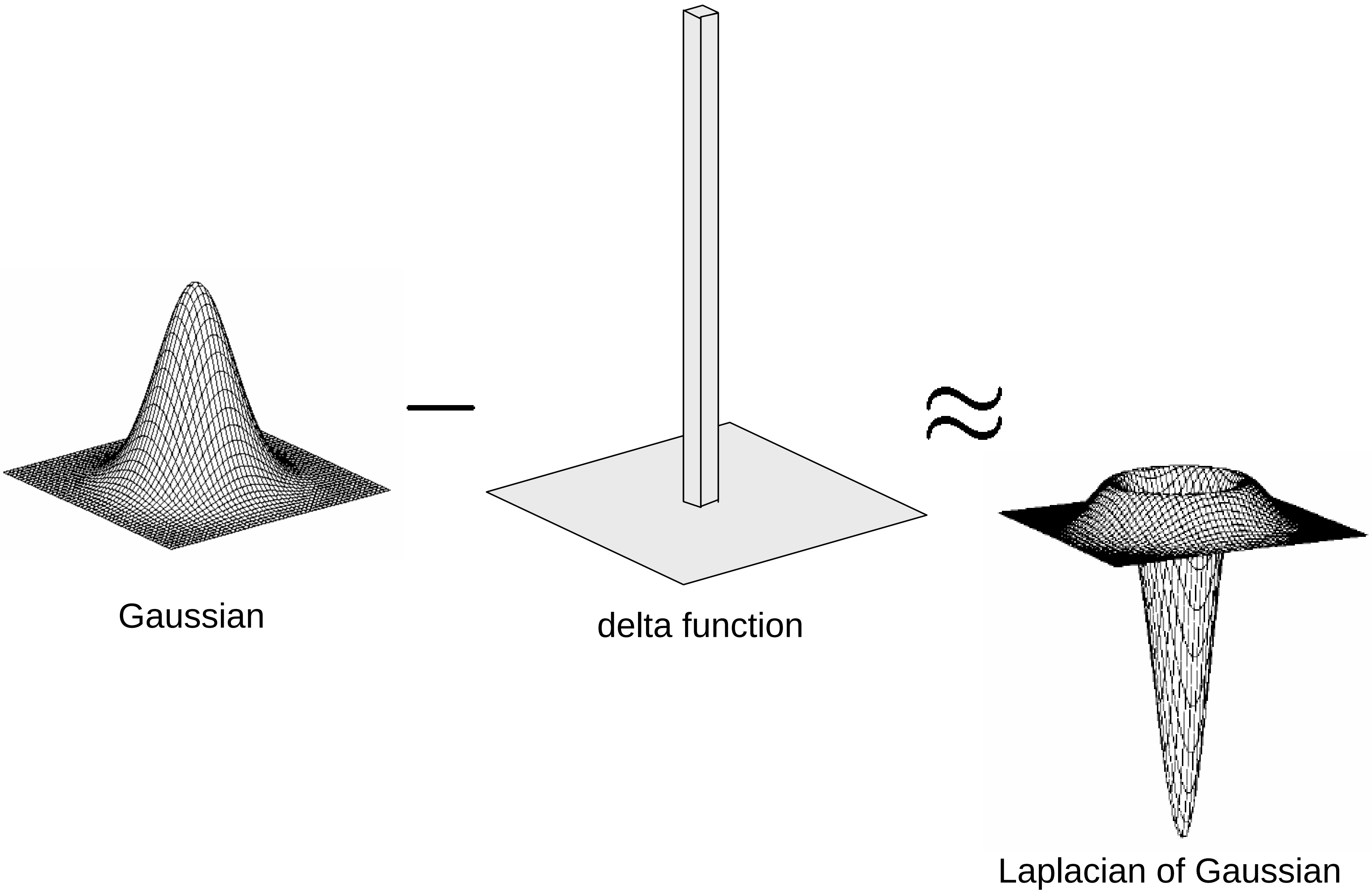
Edge detection by subtraction

smoothed – original
(scaled by 4, offset +128)

Why does this work?

filter demo

Gaussian - image filter



An edge is not a line...

How can we detect ***lines*** ?

Finding lines in an image

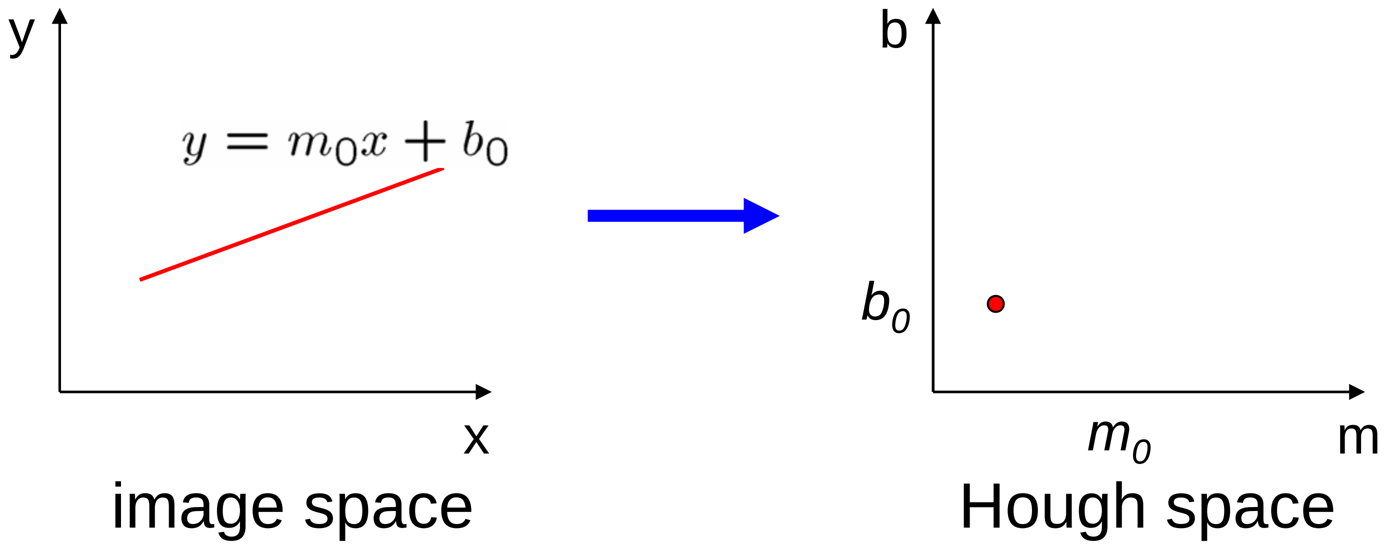
Option 1:

- Search for the line at every possible position/orientation
- What is the cost of this operation?

Option 2:

- Use a voting scheme: Hough transform

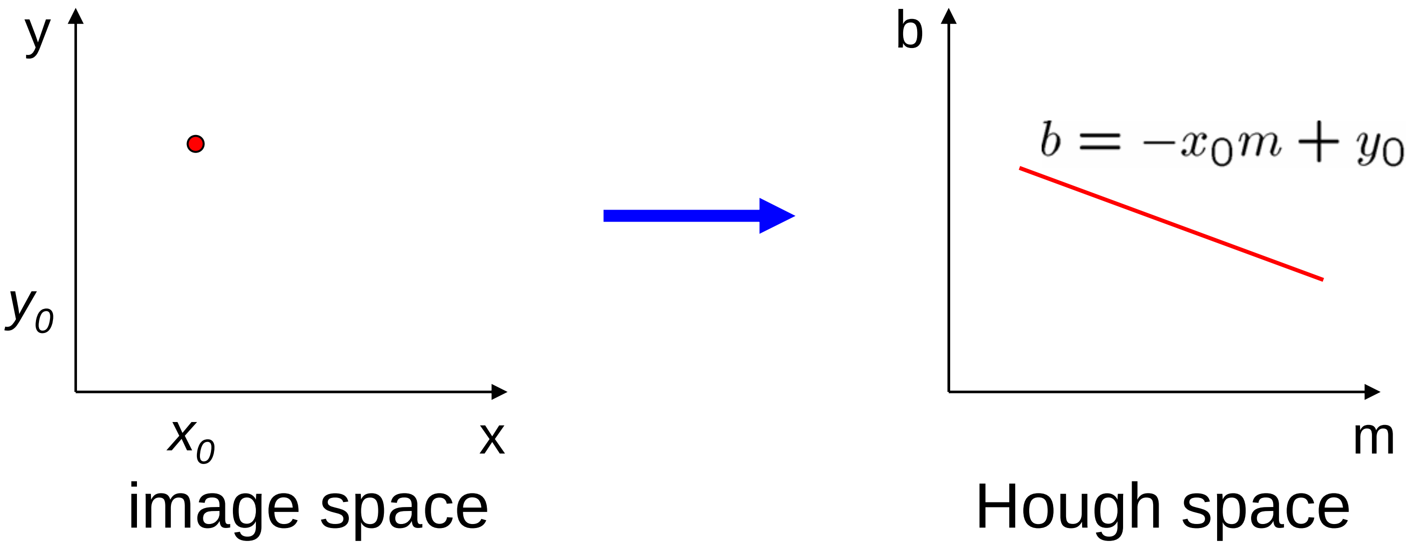
Finding lines in an image



Connection between image (x,y) and Hough (m,b) spaces

- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
  - given a set of points (x,y), find all (m,b) such that y = mx + b

Finding lines in an image



Connection between image (x,y) and Hough (m,b) spaces

- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
  - given a set of points (x,y), find all (m,b) such that y = mx + b
- What does a point (x_0, y_0) in the image space map to?
  - A: the solutions of $b = -x_0 m + y_0$
  - this is a line in Hough space

Hough transform algorithm

Typically use a different parameterization

$$d = x cos\theta + y sin\theta$$

- d is the perpendicular distance from the line to the origin
- θ is the angle this perpendicular makes with the x axis
- Why?

Hough transform algorithm

Typically use a different parameterization

$$d = x\cos\theta + y\sin\theta$$

- d is the perpendicular distance from the line to the origin
- θ is the angle this perpendicular makes with the x axis
- Why?

Basic Hough transform algorithm

1. Initialize H[d, θ]=0
2. for each edge point I[x,y] in the image

   for θ = 0 to 180

   $$d = x\cos\theta + y\sin\theta$$

   H[d, θ] += 1
3. Find the value(s) of (d, θ) where H[d, θ] is maximum
4. The detected line in the image is given by $d = x\cos\theta + y\sin\theta$

What's the running time (measured in # votes)?

Extensions

Extension 1: Use the image gradient

1. same
2. for each edge point I[x,y] in the image

   compute unique (d, θ) based on image gradient at (x,y)

   H[d, θ] += 1
3. same
4. same

What's the running time measured in votes?

Extensions

Extension 1: Use the image gradient

1. same
2. for each edge point I[x,y] in the image

   compute unique (d, θ) based on image gradient at (x,y)

   H[d, θ] += 1
3. same
4. same

What's the running time measured in votes?

Extension 2

- give more votes for stronger edges

Extension 3

- change the sampling of (d, θ) to give more/less resolution

Extension 4

- The same procedure can be used with circles, squares, or any other shape

Hough demos

Line : http://www/dai.ed.ac.uk/HIPR2/houghdemo.html

http://www.dis.uniroma1.it/~iocchi/slides/icra2001/java/hough.html

Circle : http://www.markschulze.net/java/hough/

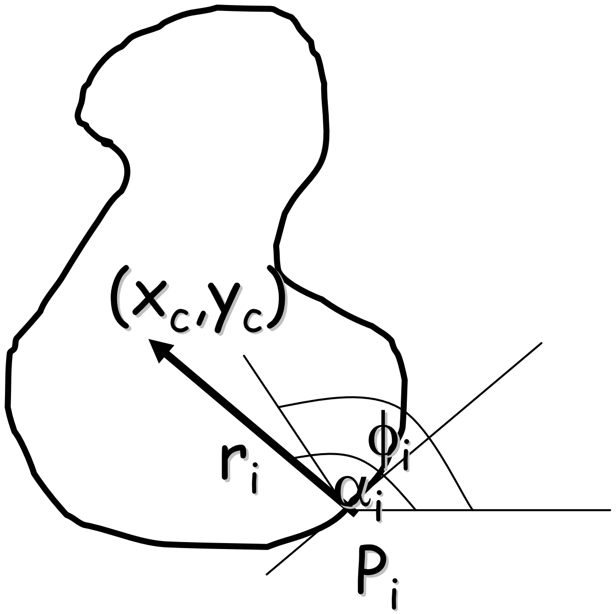
Hough Transform for Curves

The H.T. can be generalized to detect any curve that can be expressed in parametric form:

- Y = f(x, a1,a2,…ap)
- a1, a2, … ap are the parameters
- The parameter space is p-dimensional
- The accumulating array is LARGE!

Generalizing the H.T.

The H.T. can be used even if the curve has not a simple analytic form!



$$x_c = x_i + r_i\cos(\alpha_i)$$

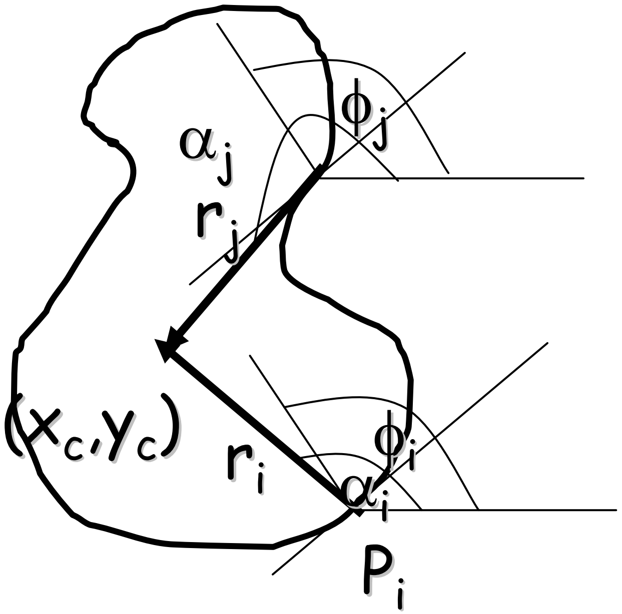
$$y_c = y_i + r_i\sin(\alpha_i)$$

1. Pick a reference point (x_c,y_c)
2. For $i = 1,\ldots,n$:
    1. Draw segment to P_i on the boundary.
    2. Measure its length r_i, and its orientation α_i.
    3. Write the coordinates of (x_c,y_c) as a function of r_i and α_i
    4. Record the gradient orientation ϕ_i at P_i.
3. Build a table with the data, indexed by ϕ_i .

Generalizing the H.T.

Suppose, there were m ***different*** gradient orientations:

(m <= n)



$x_c = x_i + r_i\cos(\alpha_i)$

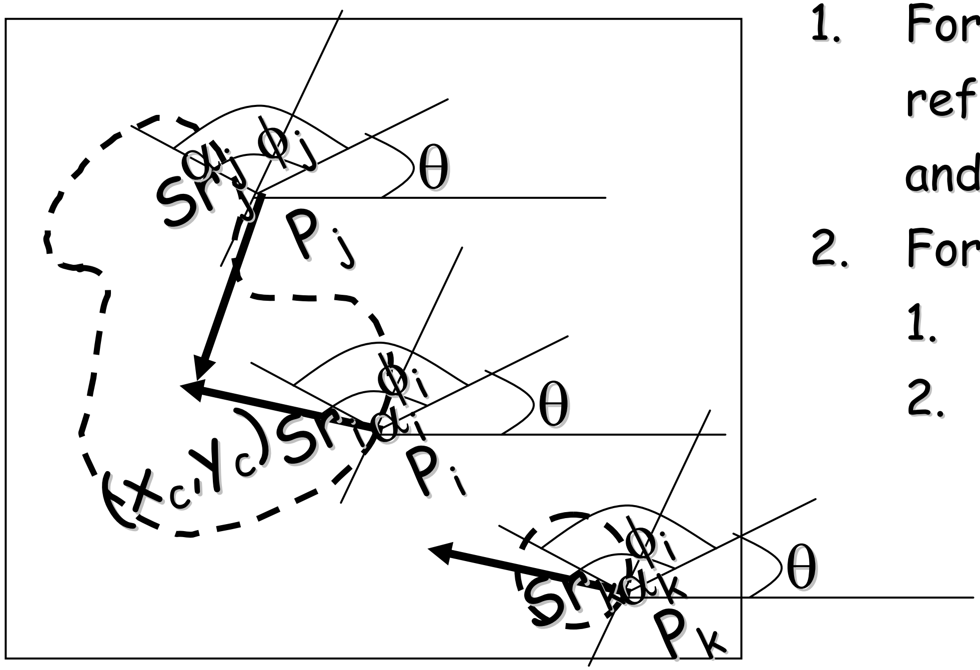
$y_c = y_i + r_i\sin(\alpha_i)$

ϕ_1	$(r^1{}_1,\alpha^1{}_1),(r^1{}_2,\alpha^1{}_2),\ldots,(r^1{}_{n1},\alpha^1{}_{n1})$
ϕ_2	$(r^2{}_1,\alpha^2{}_1),(r^2{}_2,\alpha^1{}_2),\ldots,(r^2{}_{n2},\alpha^1{}_{n2})$
. . .	. . .
ϕ_m	$(r^m{}_1,\alpha^m{}_1),(r^m{}_2,\alpha^m{}_2),\ldots,(r^m{}_{nm},\alpha^m{}_{nm})$

H.T. table

Generalized H.T. Algorithm:

Finds a rotated, scaled, and translated version of the curve:



$$x_c = x_i + r_i\cos(\alpha_i)$$

$$y_c = y_i + r_i\sin(\alpha_i)$$

1. Form an A accumulator array of possible reference points (x_c,y_c), scaling factor S and Rotation angle θ.
2. For each edge (x,y) in the image:
    1. Compute $\phi(x,y)$
    2. For each (r,α) corresponding to $\phi(x,y)$ do:
        1. For each S and θ:
            1. $x_c = x_i + r(\phi)\, S \cos[\alpha(\phi) + \theta]$
            2. $y_c = y_i + r(\phi)\, S \sin[\alpha(\phi) + \theta]$
            3. $A(x_c,y_c,S,\theta)$ ++
3. Find maxima of A.

H.T. Summary

H.T. is a “voting” scheme

- points vote for a set of parameters describing a line or curve.

The more votes for a particular set

- the more evidence that the corresponding curve is present in the image.

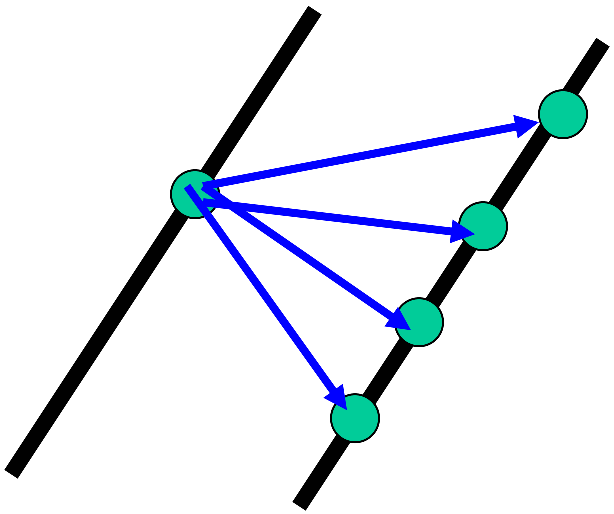
Can detect MULTIPLE curves in one shot.

Computational cost increases with the number of parameters describing the curve.

Corner detection

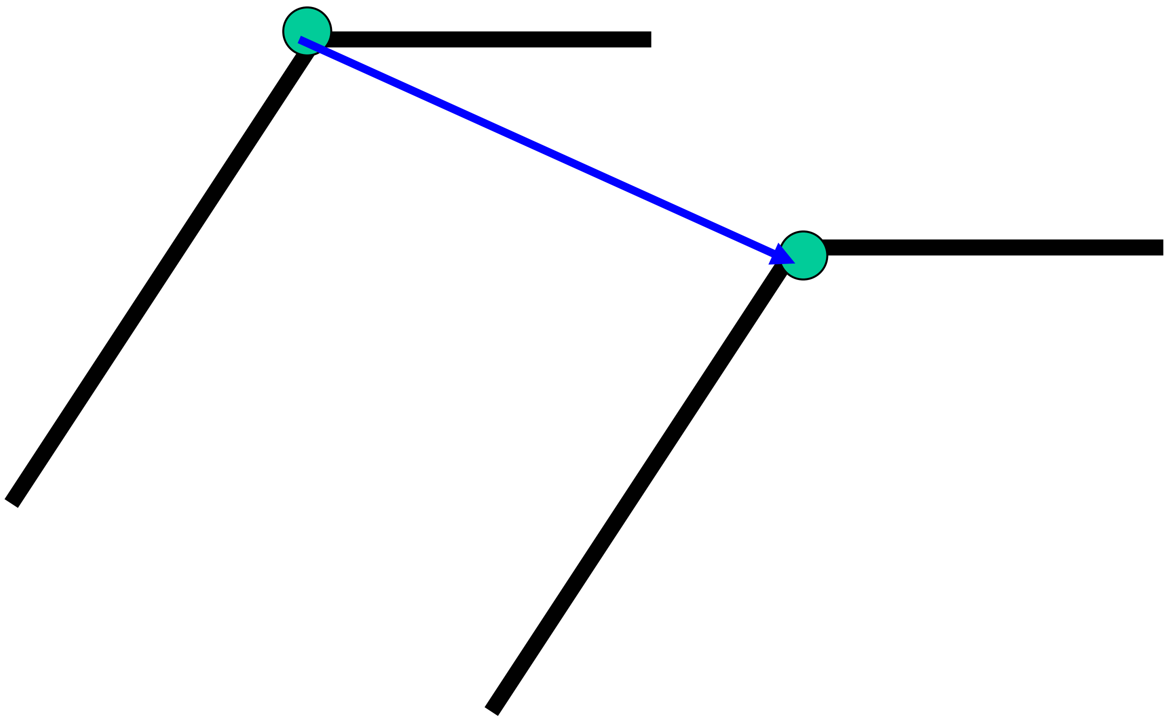
Corners contain more edges than lines.

A point on a line is hard to match.

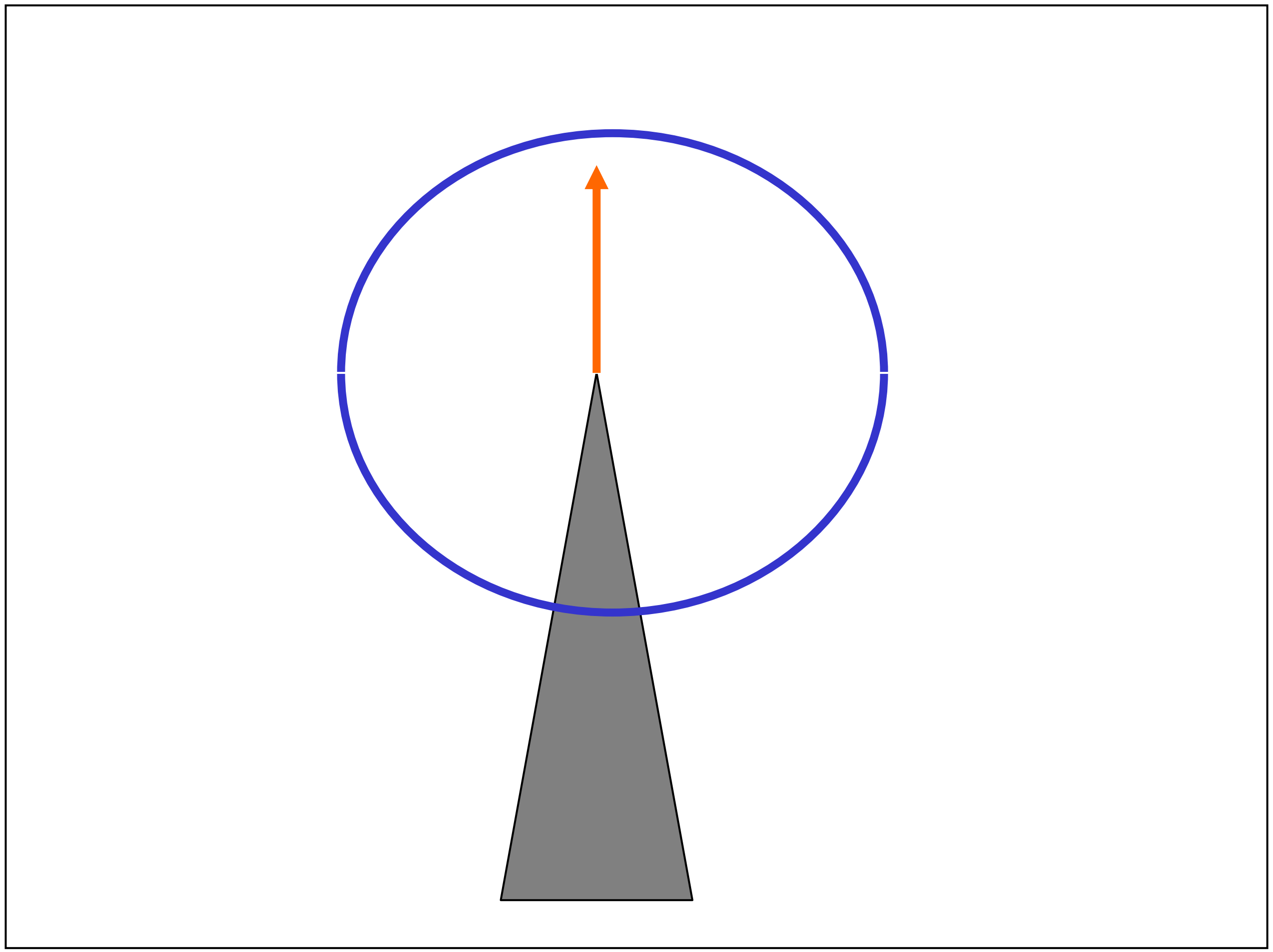


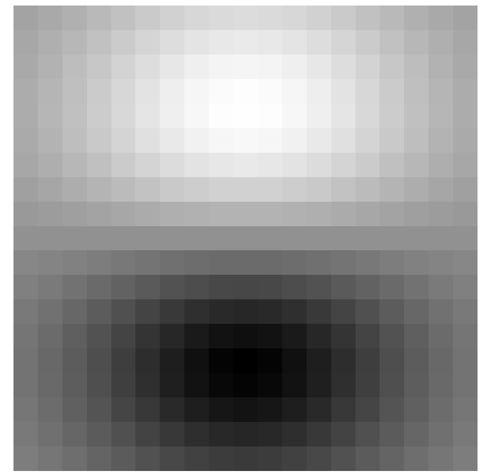
Corners contain more edges than lines.

A corner is easier



Edge Detectors Tend to Fail at Corners





Finding Corners

Intuition:

- Right at corner, gradient is ill defined.
- Near corner, gradient has two different values.

Formula for Finding Corners

We look at matrix:

Sum over a small region, the hypothetical corner

Gradient with respect to x, times gradient with respect to y

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

Matrix is symmetric

WHY THIS?

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 all gradients in neighborhood are:

(k,0) or (0, c) or (0, 0) (or off-diagonals cancel).

What is region like if:

1. $\lambda 1 = 0$?
2. $\lambda 2 = 0$?
3. $\lambda 1 = 0$ and $\lambda 2 = 0$?
4. $\lambda 1 > 0$ and $\lambda 2 > 0$?

General Case:

From Linear Algebra, it follows that because C is symmetric:

$$C = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

With R a rotation matrix.

So every case is like one on last slide.

So, to detect corners

Filter image.

Compute magnitude of the gradient everywhere.

We construct C in a window.

Use Linear Algebra to find $\lambda1$ and $\lambda2$.

If they are both big, we have a corner.