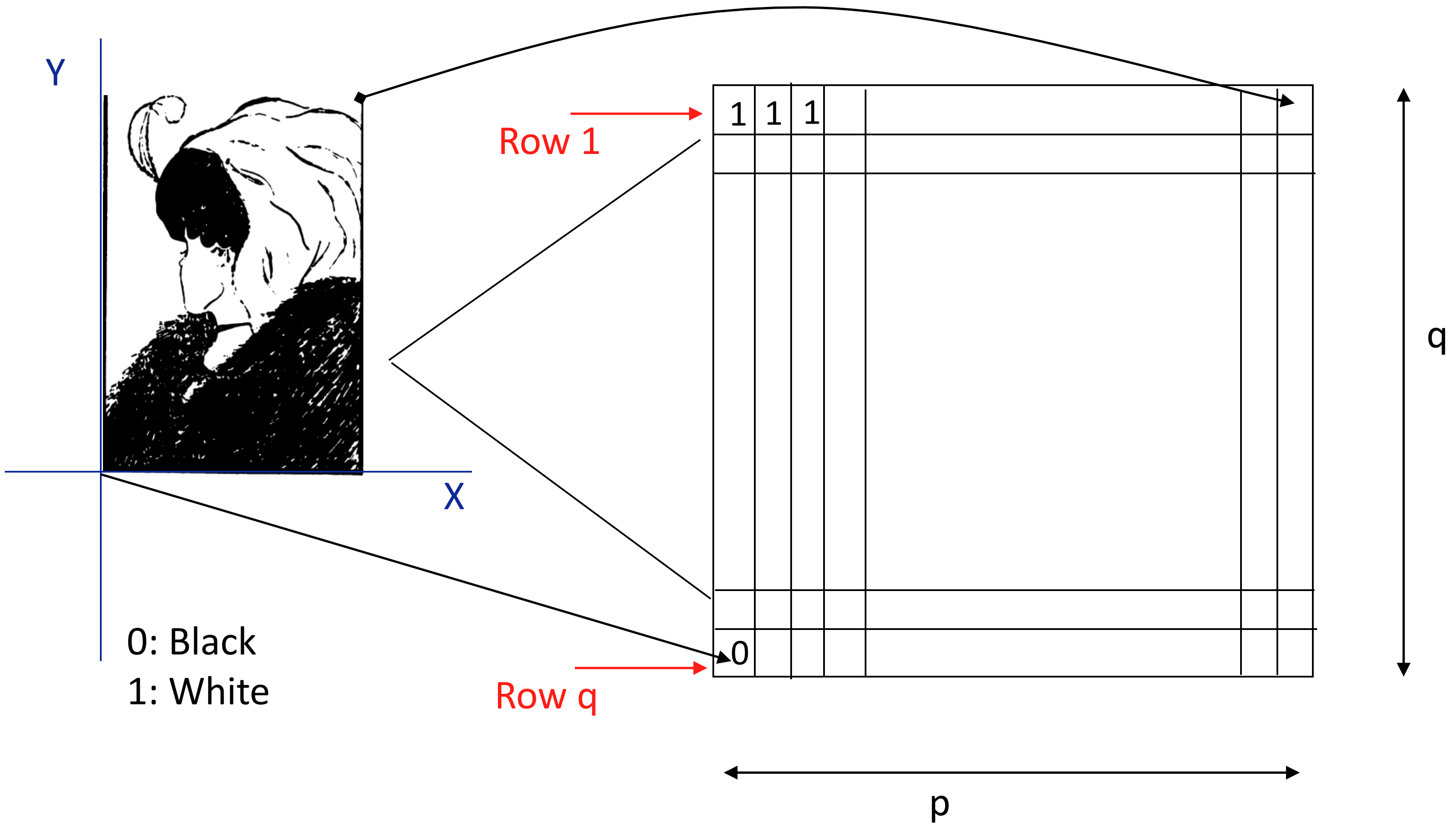
Images: Edge Detection and Image Features

Images

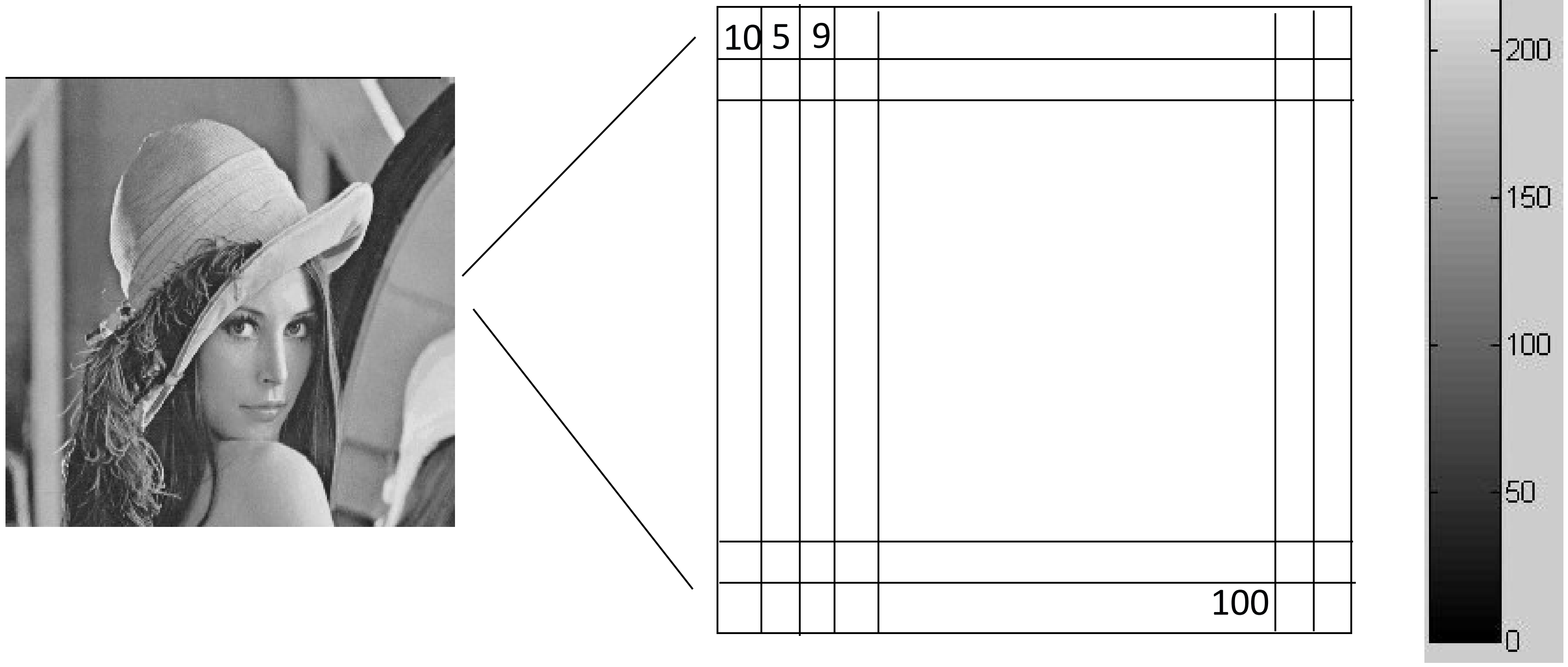
- Binary
- Gray Scale
- Color

Binary Image

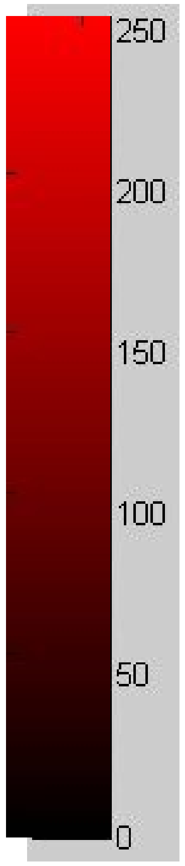


0: Black
1: White

Gray Scale Image



Gray Scale Image



Color Image (RGB)

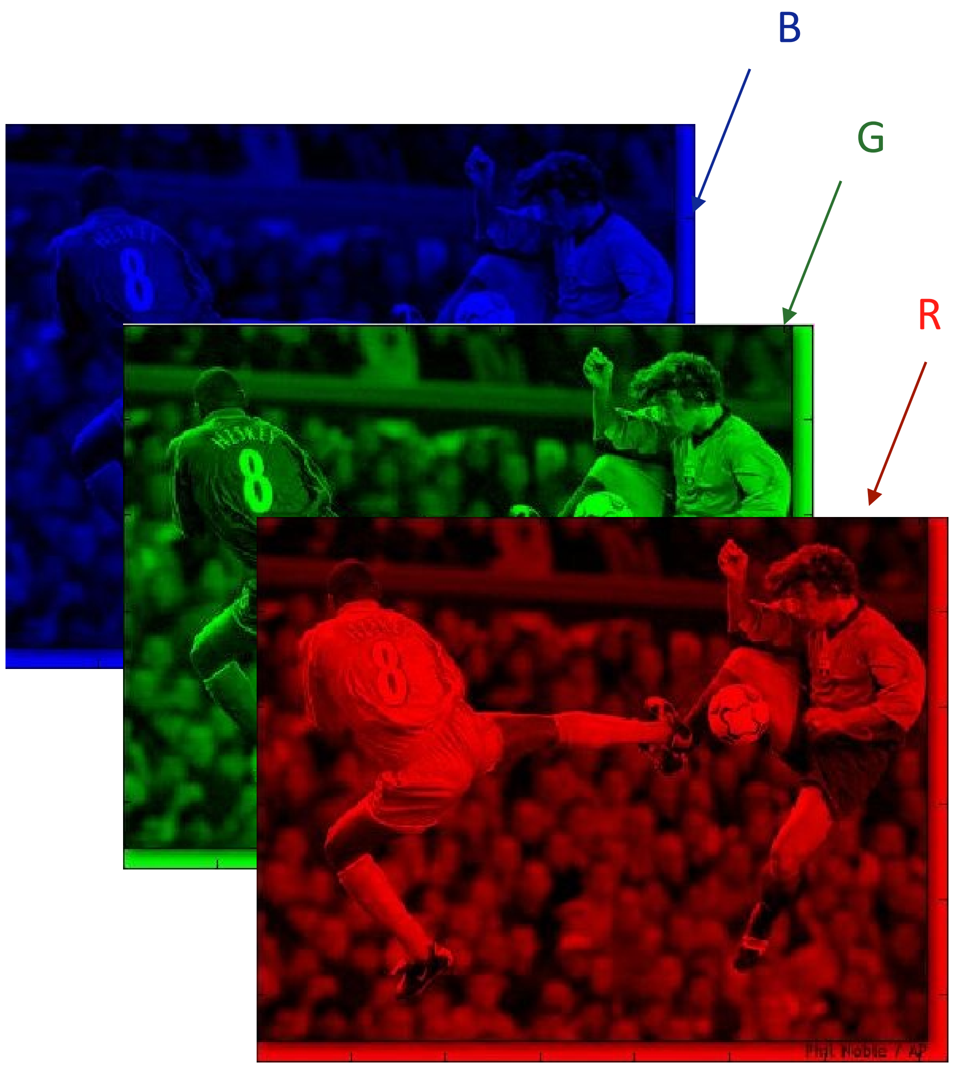
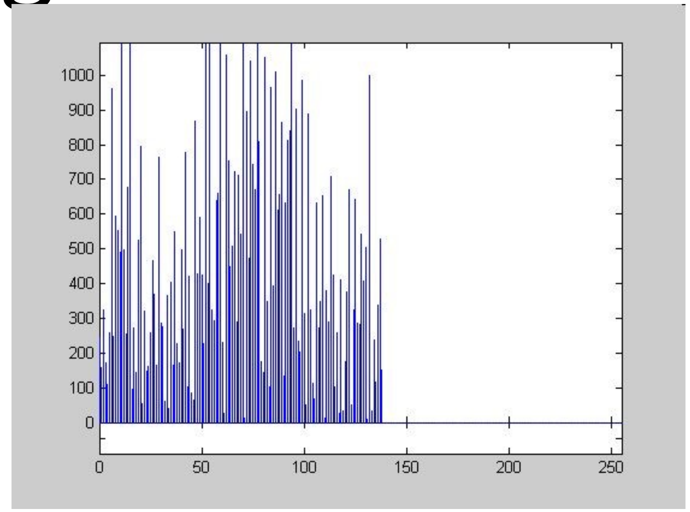


Image Histogram



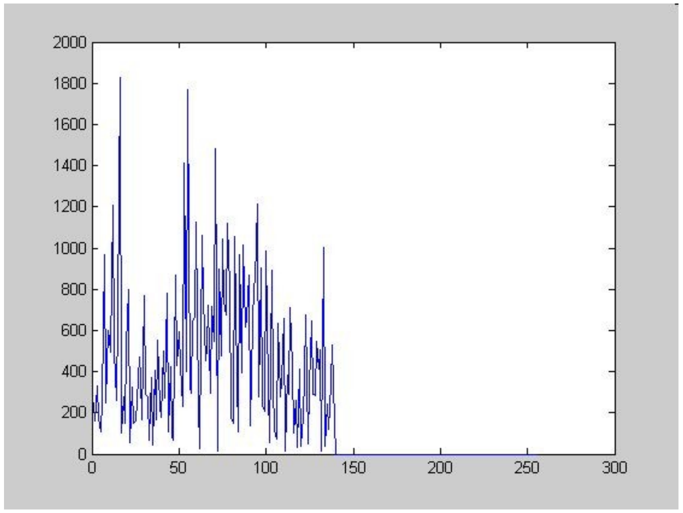


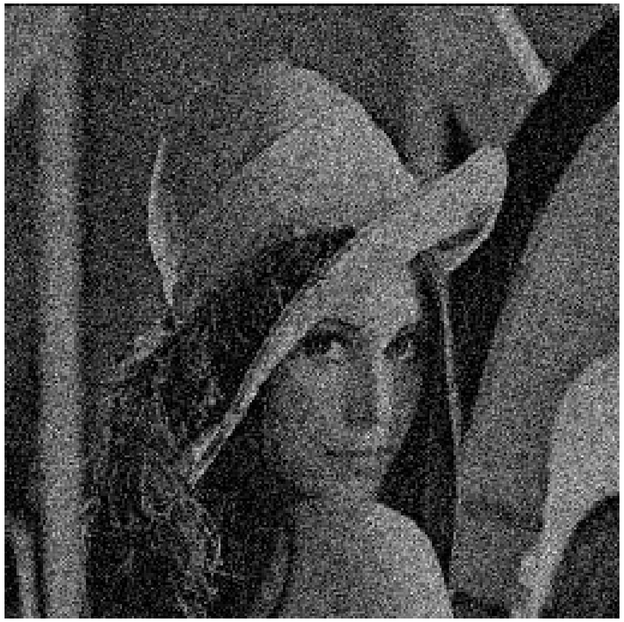
Image Noise

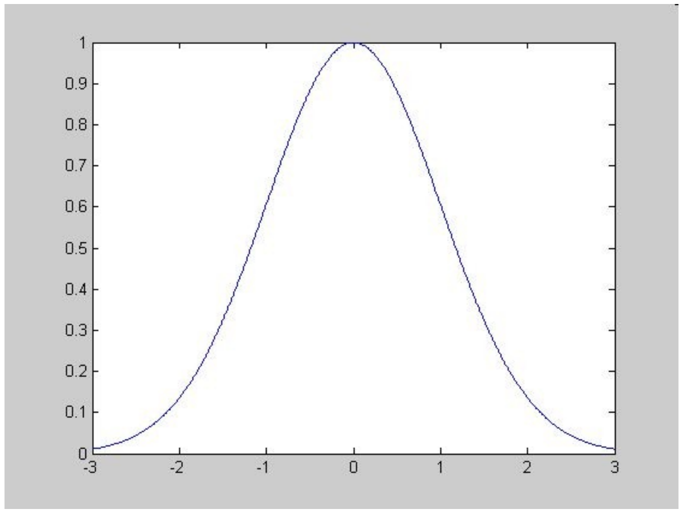
Let *I(i,j)* be the true pixel values and *n(i,j)* be the noise added to the pixel (*i,j)*

$$\hat{I}(i,j) = I(i,j) + n(i,j)$$ (Additive White Noise)

Gaussian Noise (White Noise)

$$n(i,j) = e^{\frac{-x^2}{2\sigma^2}}$$



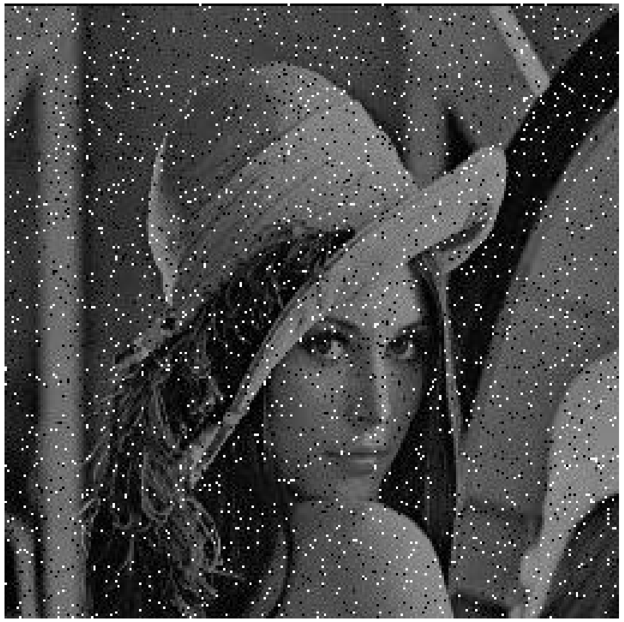


Salt and Pepper Noise

$$\hat{I}(i,j) = \begin{cases} I(i,j) & t < l \\ s_{\min} + r(s_{\max} - s_{\min}) & t \geq l \end{cases}$$

$p, q \in [0,1]$ (Uniformly distributed t random variables)

$l =$ Threshold



Edges and Contour Representation

- Edges- Significant local changes in image; occur on the boundary between 2 different regions in an image.
- Contour- Representation of linked edges for a region boundary.

  - Closed: Correspond to region boundaries; filling algorithm determines the pixels in the region.

  - Open:

    a) part of a region boundary; gaps' formation due to high edge-detection threshold or weak contrast.

    b) occur when line fragments are linked together, as in drawing or handwriting.
- Contour Representation

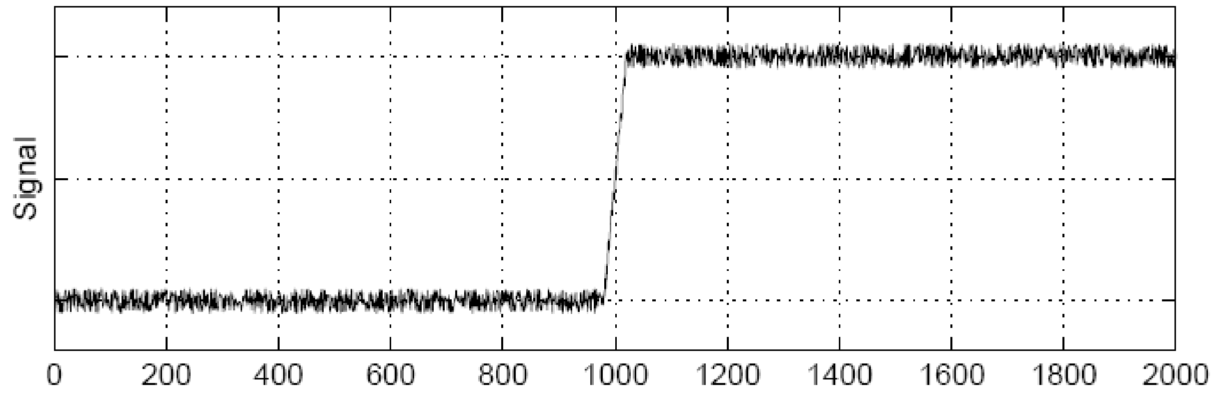
  - Ordered list of Edges

  - Curve- a mathematical model for a contour (line segments and cubic splines)

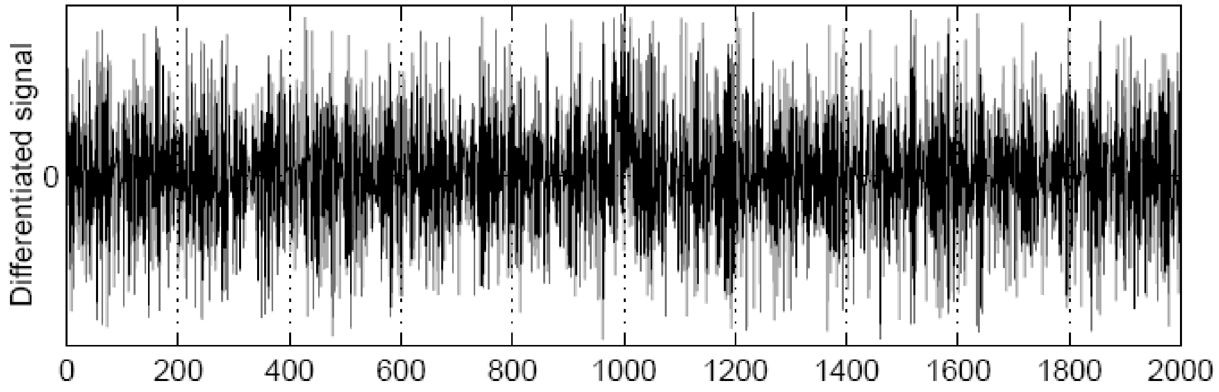
Edges

Finding Edges

- Consider a 1D signal: `I(x)`



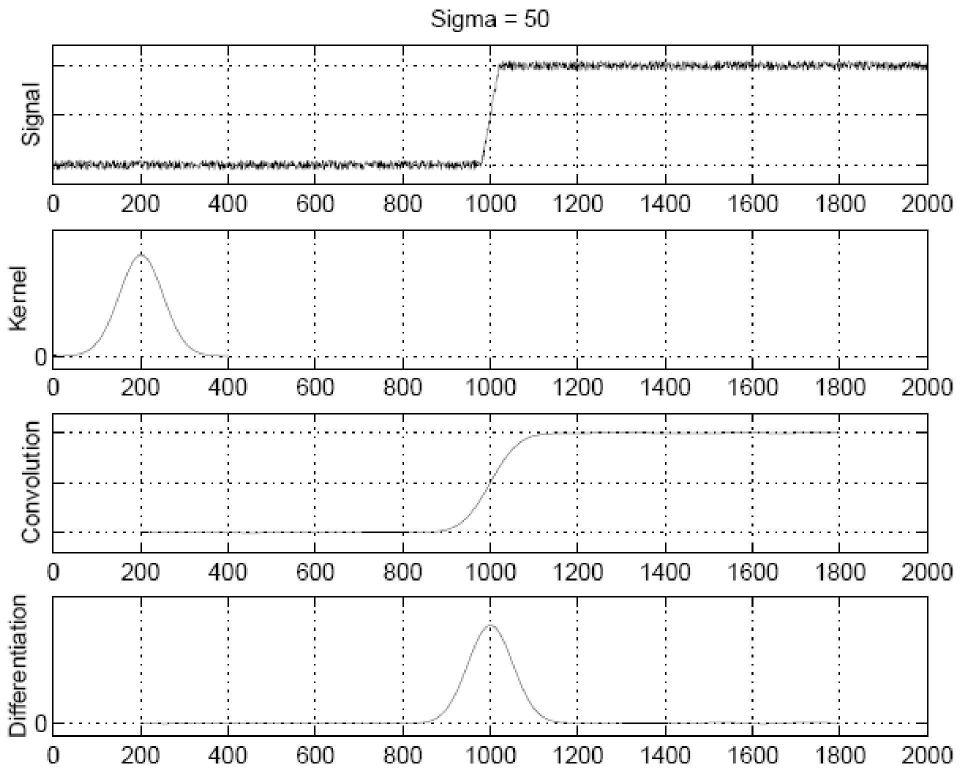
- Obvious strategy: look for maxima/minima in `I'(x)`:



- Need to smooth to suppress noise

1-D Edge Detection

- Convolve the 1-D signal with a Gaussian kernel to give s(x).
- Compute derivate of resulting smoothed signal to give s'(x).
- Find maxima/minima of s'(x)
- Threshold

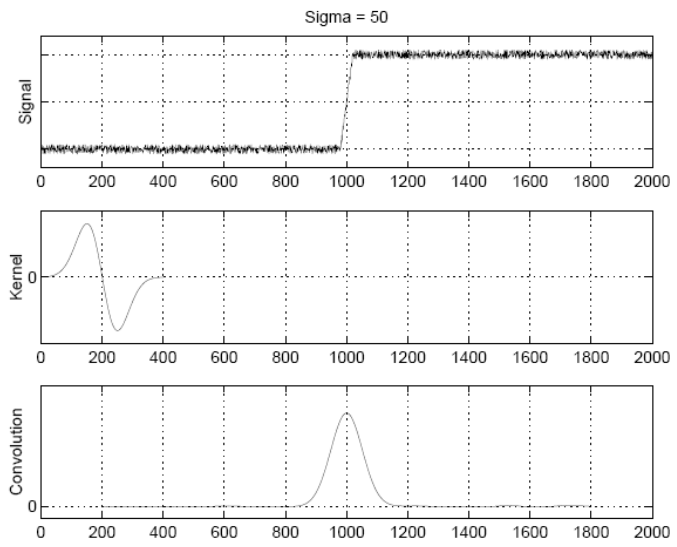


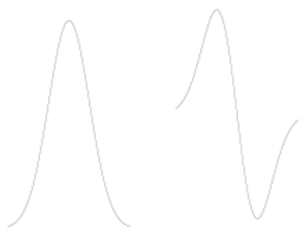
$$g_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

$$\begin{aligned} s(x) = g_\sigma(x) * I(x) &= \int_{-\infty}^{+\infty} g_\sigma(u) I(x-u)\, du \\ &= \int_{-\infty}^{+\infty} g_\sigma(x-u) I(u)\, du \end{aligned}$$

1-D Edge Detection

- Note that $s'(x) = \frac{d}{dx}\left[g_\sigma(x) * I(x)\right] = g'_\sigma(x) * I(x)$

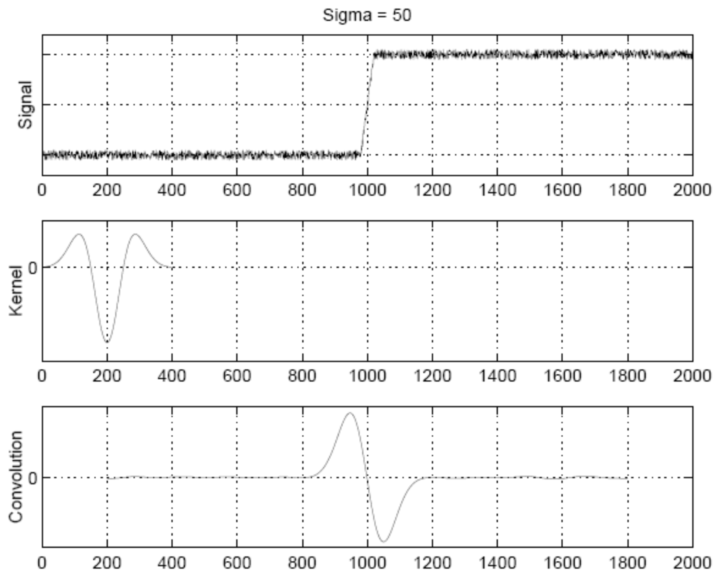




$$s''(x) = g''_\sigma(x) * I(x)$$

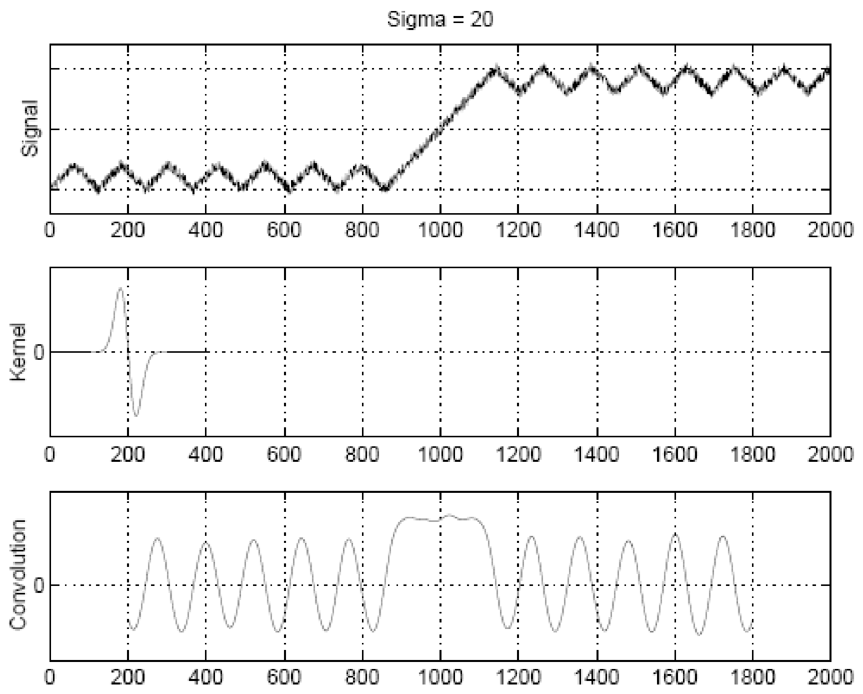
1-D Edge Detection

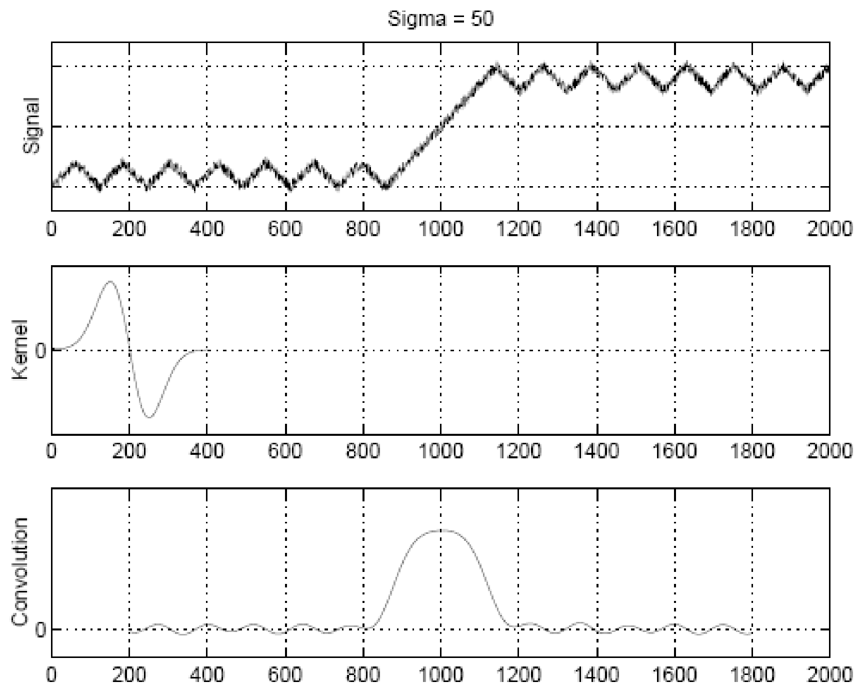
- Looking for maxima/minima of s'(x) is same as zero-crossings of s''(x), so easier to convolve with Laplacian of Gaussian, $g_\sigma''(x)$:



Multi-scale 1D edge detection

- As σ increases, the smoothing increases and only the central edge survives.





Detection of Discontinuities

- Point Detection Example:
  - Apply a high-pass filter.
  - A point is detected if the response is larger than a positive threshold.

$$|R| > T$$

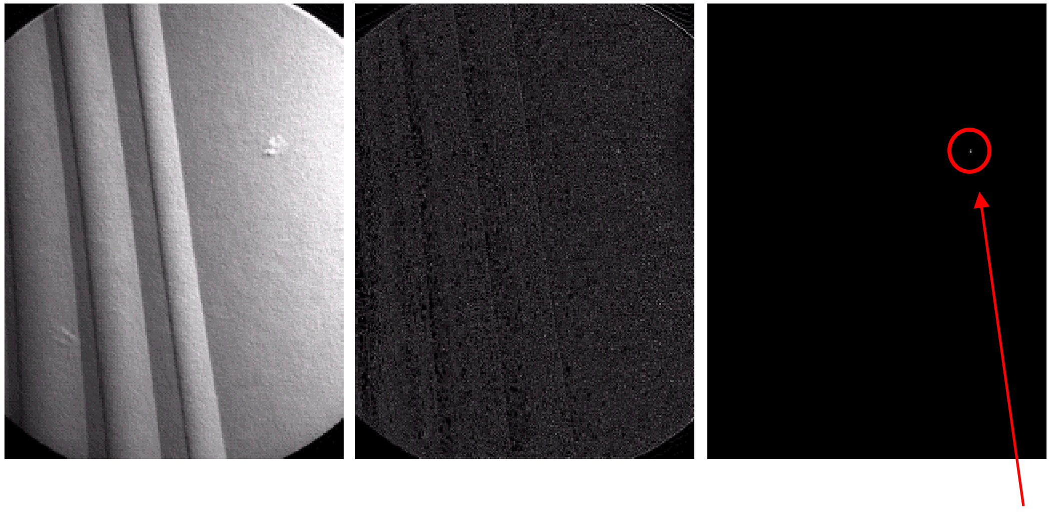
Threshold

  - The idea is that the gray level of an isolated point will be quite different from the gray level of its neighbors.

Detection of Discontinuities

- Point Detection

−1	−1	−1
−1	8	−1
−1	−1	−1

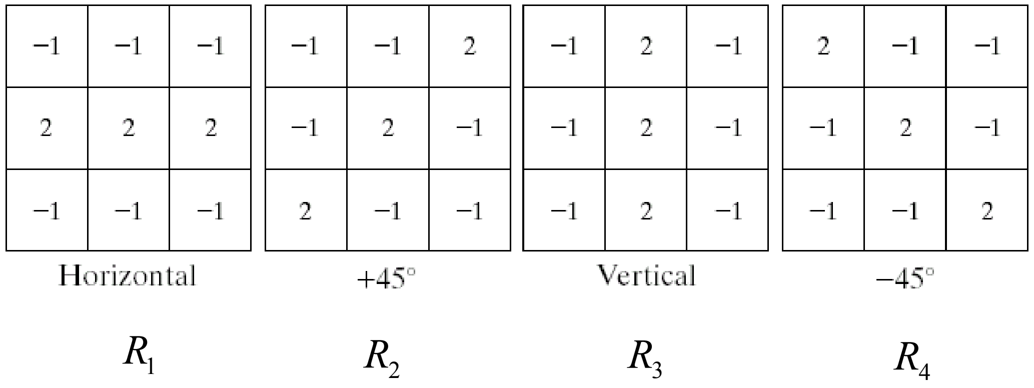


(a) Point detection mask. (b) X-ray image of a turbine blade with a porosity. (c) Result of point detection. (d) Result of using Eq. (10.1-2). (Original image courtesy of X-TEK Systems Ltd.)

Detection of Discontinuities

- Line Detection Example:

FIGURE 10.3 Line masks.



−1	−1	−1
2	2	2
−1	−1	−1

Horizontal

R_1

−1	−1	2
−1	2	−1
2	−1	−1

+45°

R_2

−1	2	−1
−1	2	−1
−1	2	−1

Vertical

R_3

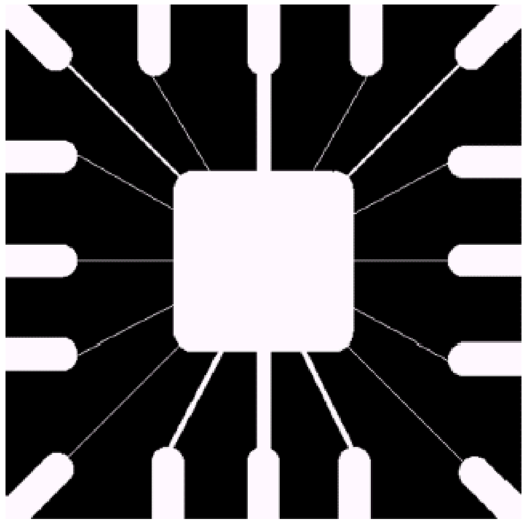
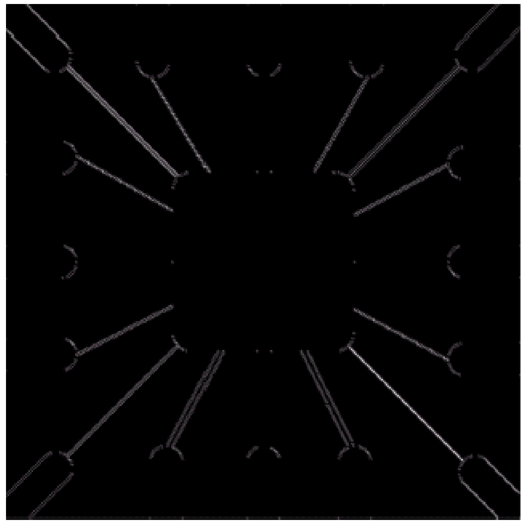
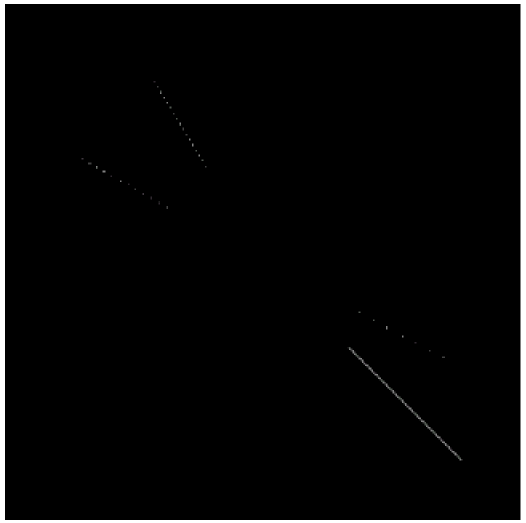
2	−1	−1
−1	2	−1
−1	−1	2

−45°

R_4

Detection of Discontinuities

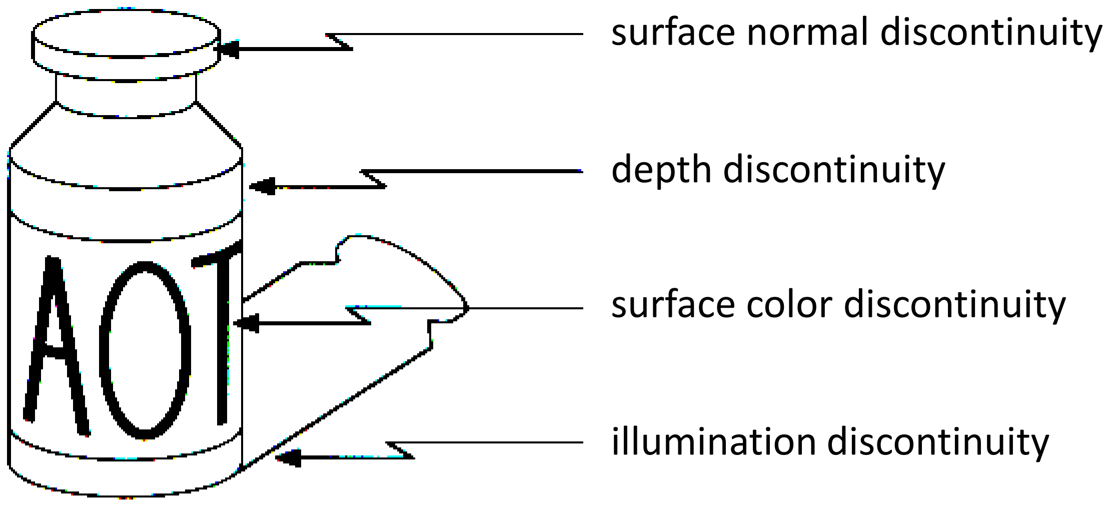
- Line Detection Example:

Detection of Discontinuities

- Edge Detection:
  - An edge is the boundary between two regions with relatively distinct gray levels.
  - Edge detection is by far the most common approach for detecting meaningful discontinuities in gray level. The reason is that isolated points and thin lines are not frequent occurrences in most practical applications.
  - The idea underlying most edge detection techniques is the computation of a local derivative operator.

Origin of Edges


surface normal discontinuity
depth discontinuity
surface color discontinuity
illumination discontinuity
AOT

- Edges are caused by a variety of factors

Profiles of image intensity edges

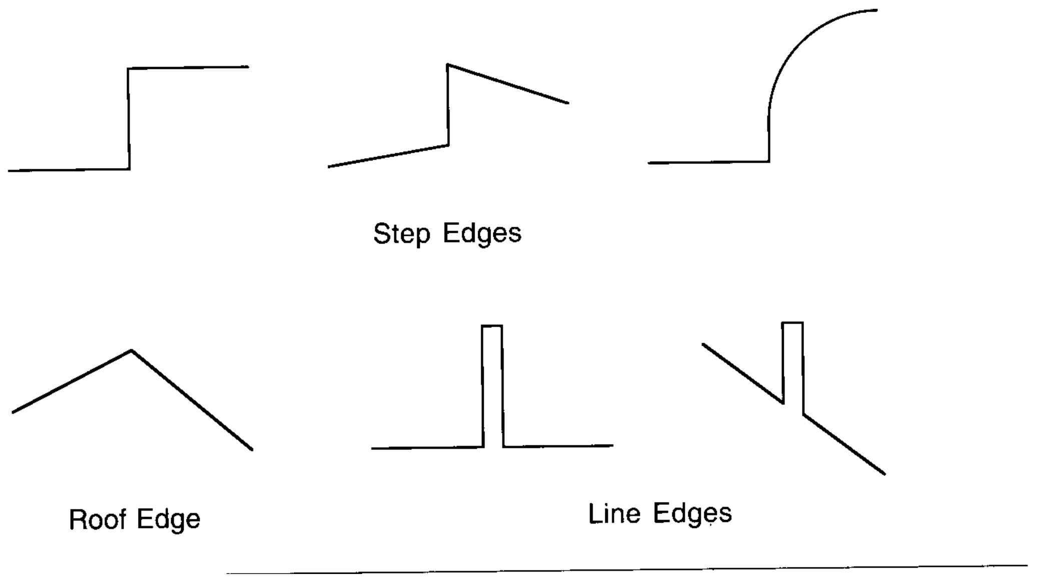
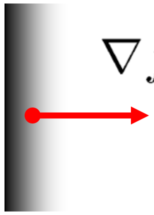


Image gradient

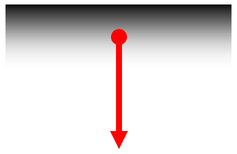
- The gradient of an image:

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

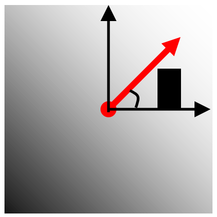
- The gradient points in the direction of most rapid change in intensity



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



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



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

- The gradient direction is given by:

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

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

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

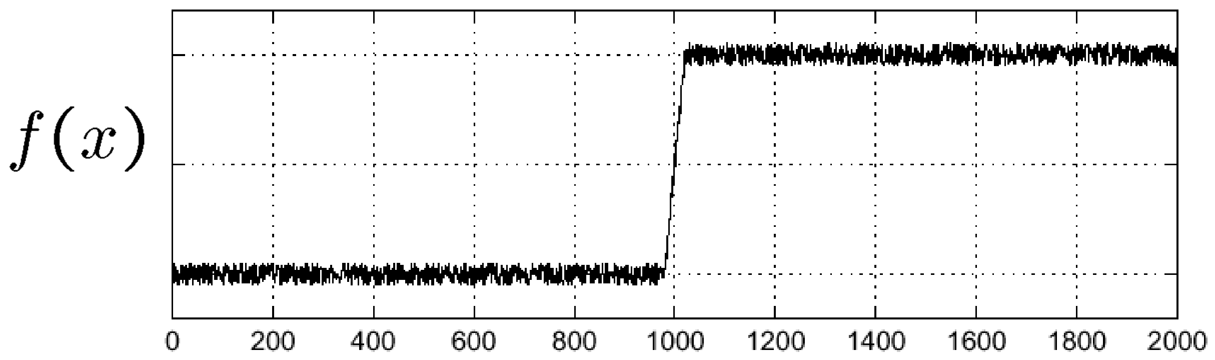
The discrete gradient

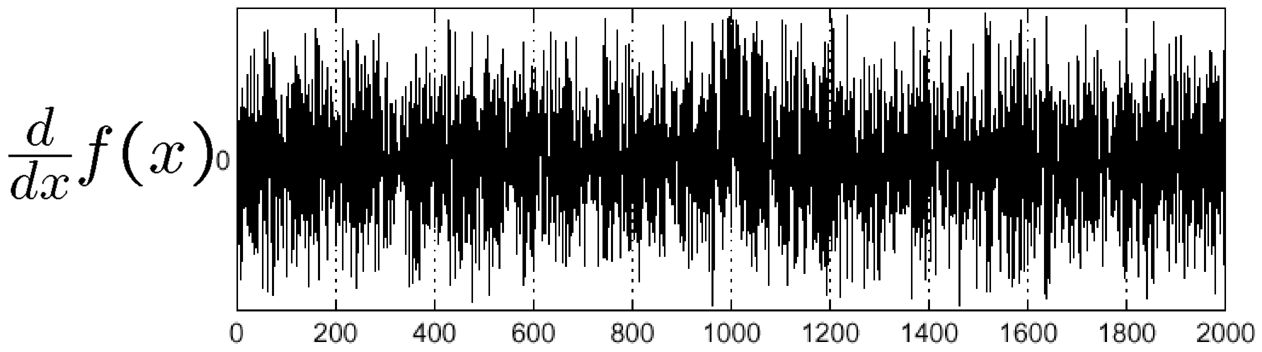
- How can we differentiate a *digital* image f[x,y]?
  - Option 1: reconstruct a continuous image, then take gradient
  - Option 2: take discrete derivative (finite difference)

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

Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal





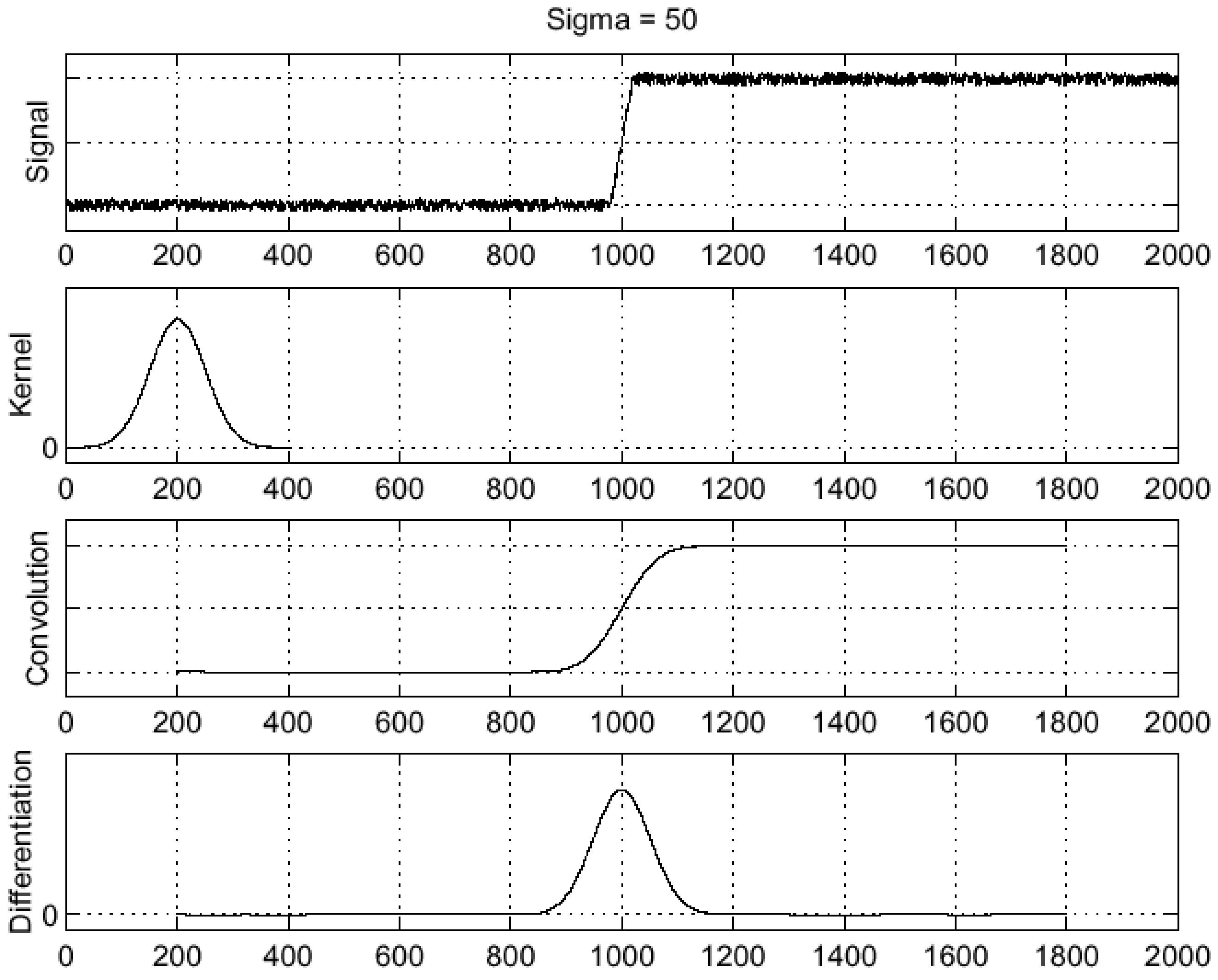
Solution: smooth first

f

h

$h \star f$

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



Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Derivative theorem of convolution

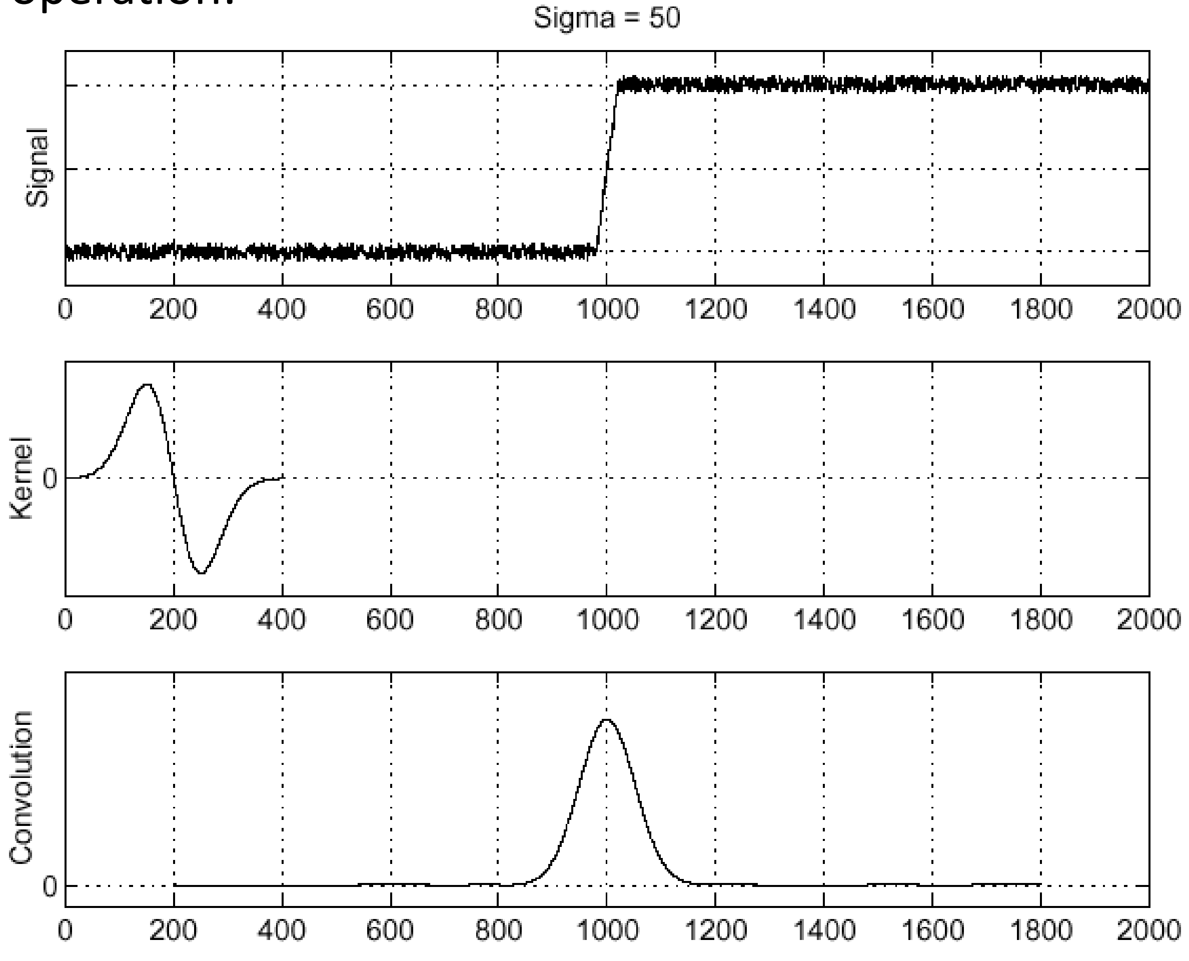
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

- This saves us one operation:

f

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

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



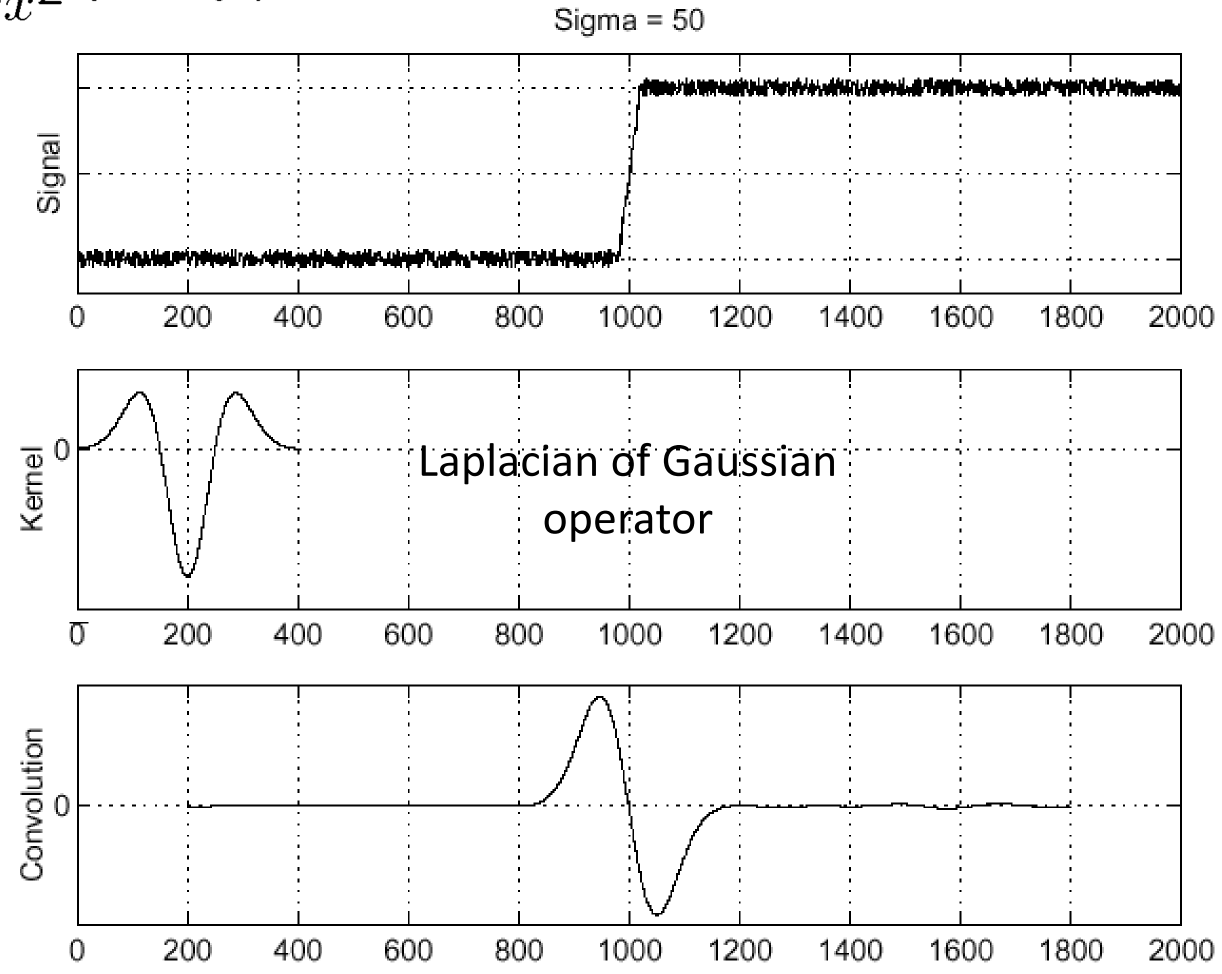
Laplacian of Gaussian

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

f

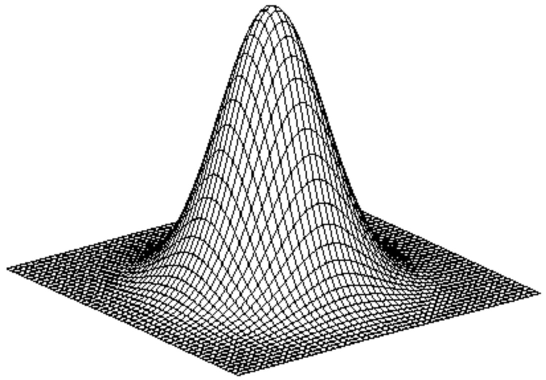
$\frac{\partial^2}{\partial x^2}h$

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



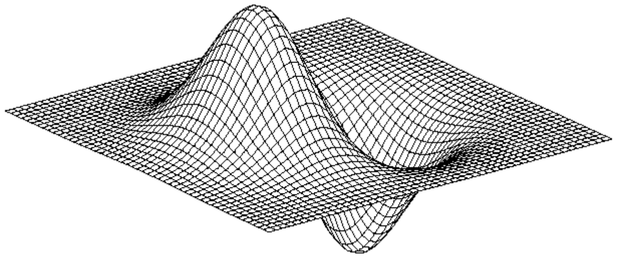
Zero-crossings of bottom graph

2D edge detection filters



Gaussian

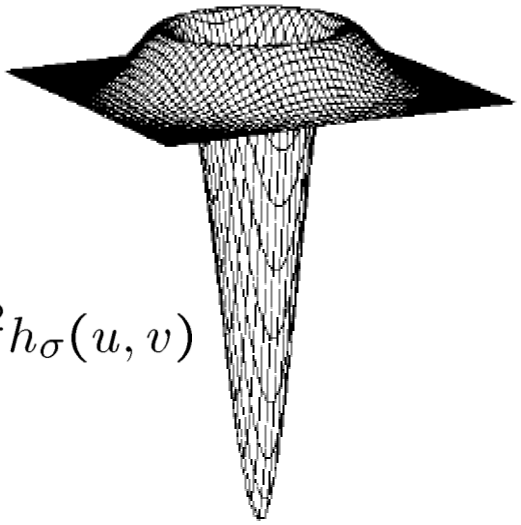
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



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:

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

Edge Detection

Possible filters to find gradients along vertical and horizontal directions:

−1	−1	−1
0	0	0
1	1	1

−1	0	1
−1	0	1
−1	0	1

Prewitt

Averaging provides noise suppression

−1	−2	−1
0	0	0
1	2	1

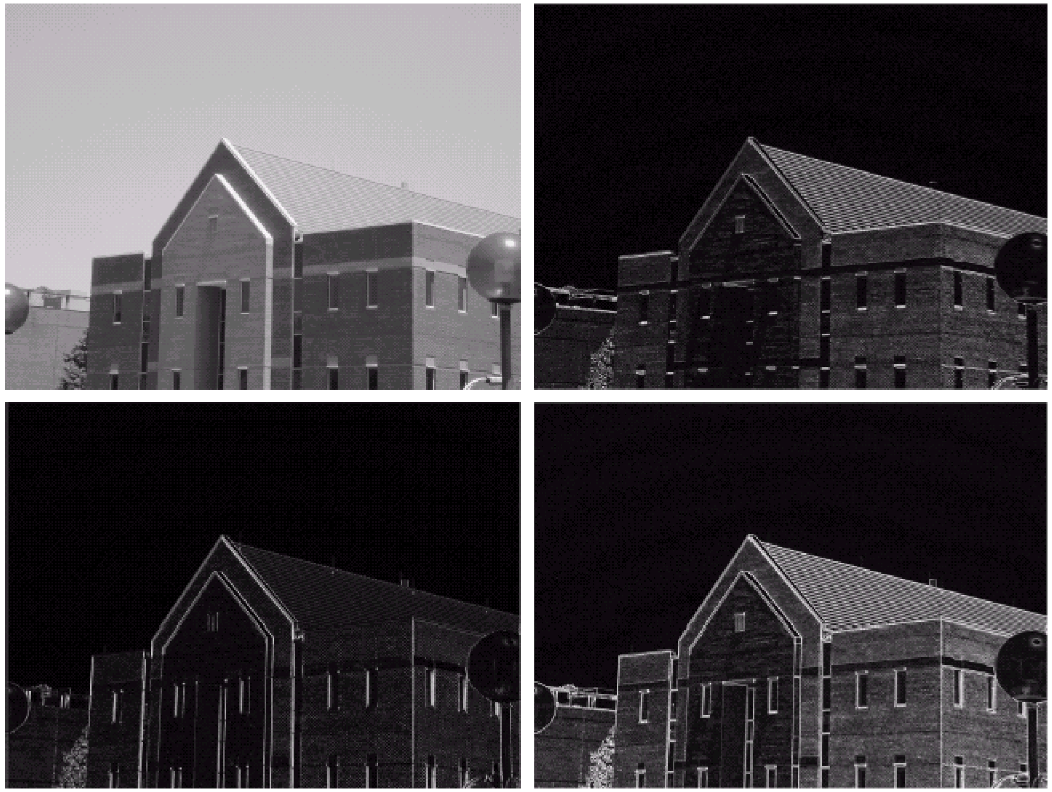
−1	0	1
−2	0	2
−1	0	1

Sobel

This gives more importance to the center point.

Edge Detection

(a) Original image. (b) $|G_x|$, component of the gradient in the x-direction. (c) $|G_y|$, component in the y-direction. (d) Gradient image, $|G_x| + |G_y|$.



Edge Detection

- The Laplacian of an image f(x,y) is a second-order derivative defined as

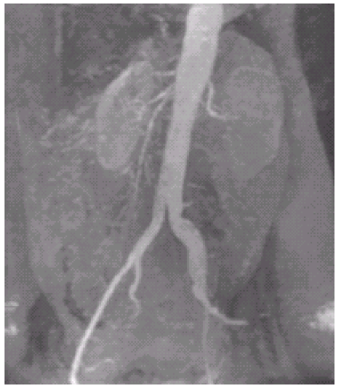
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

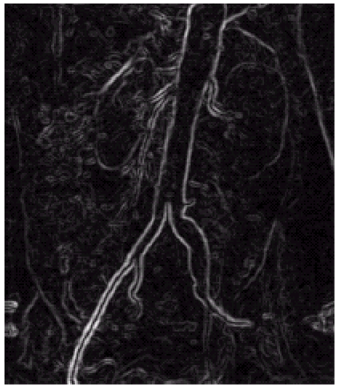
Digital approximations:

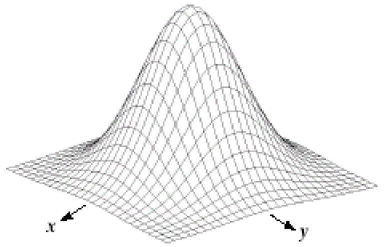
0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

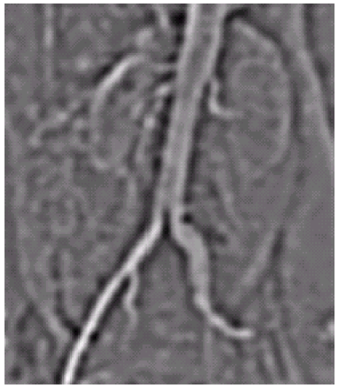
Edge Detection

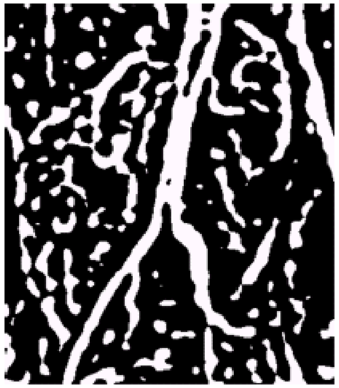


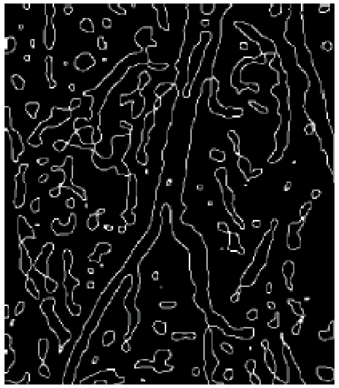




–1	–1	–1
–1	8	–1
–1	–1	–1







One simple method to find zero-crossings is black/white thresholding:

1. Set all positive values to white
2. Set all negative values to black
3. Determine the black/white transitions.

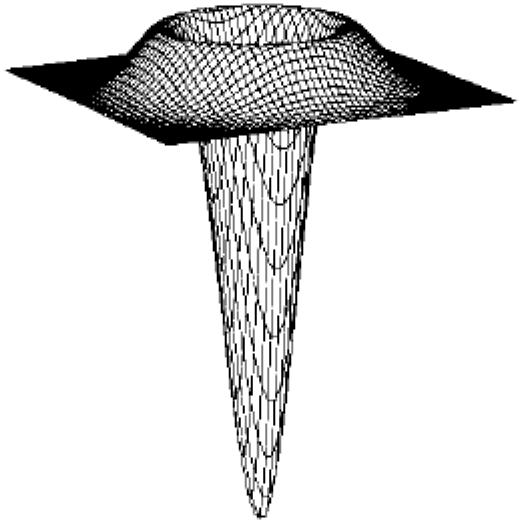
Compare (b) and (g):

•Edges in the zero-crossings image is thinner than the gradient edges.

•Edges determined by zero-crossings have formed many closed loops.

Edge Detection

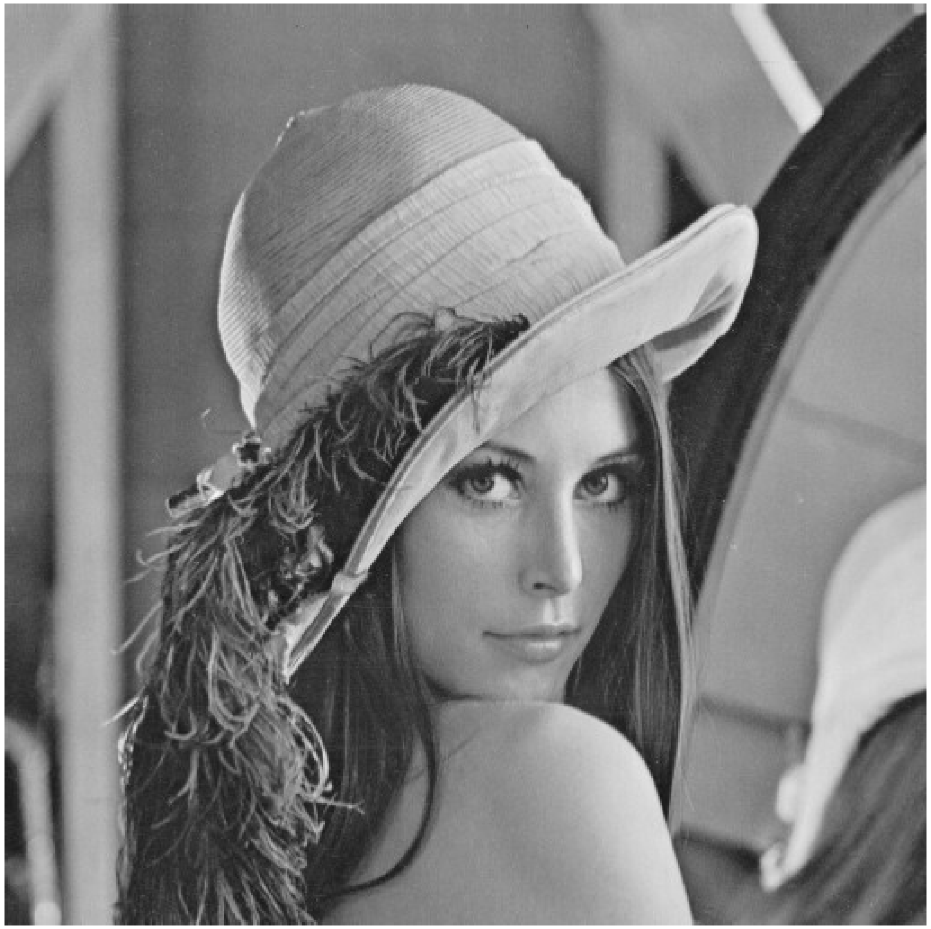
- The Laplacian of a Gaussian filter



A digital approximation:

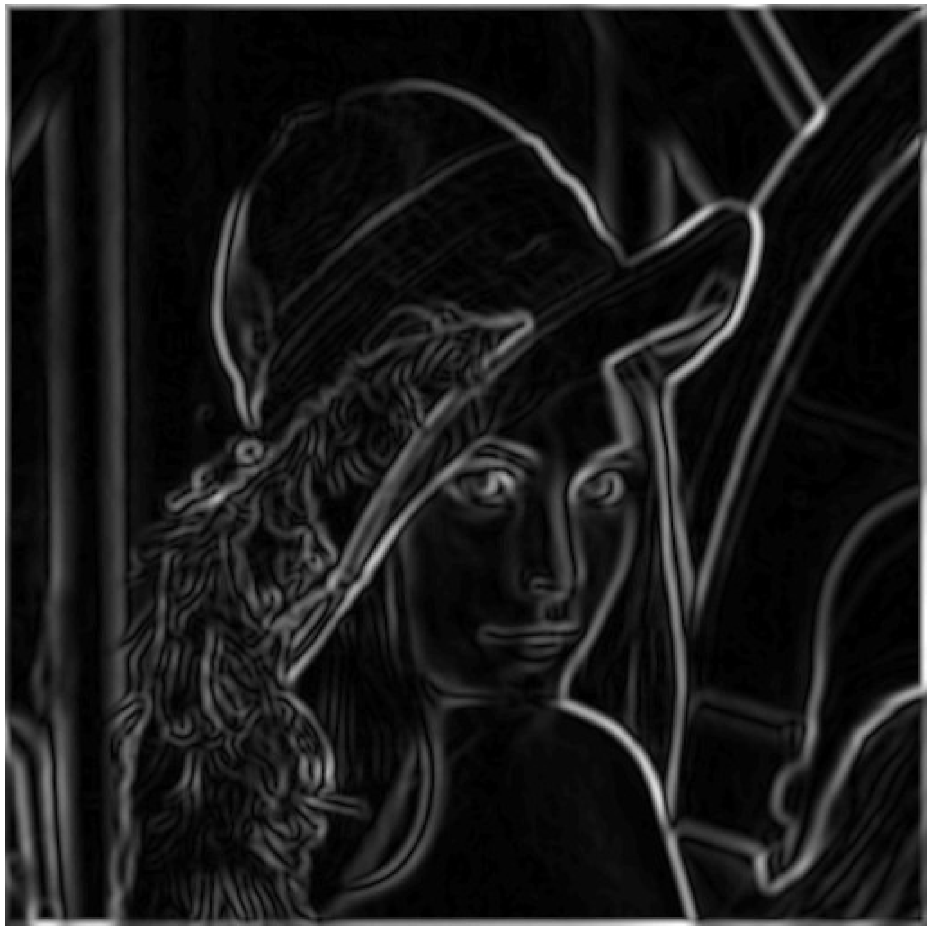
0	0	1	0	0
0	1	2	1	0
1	2	-16	2	1
0	1	2	1	0
0	0	1	0	0

The Canny edge detector



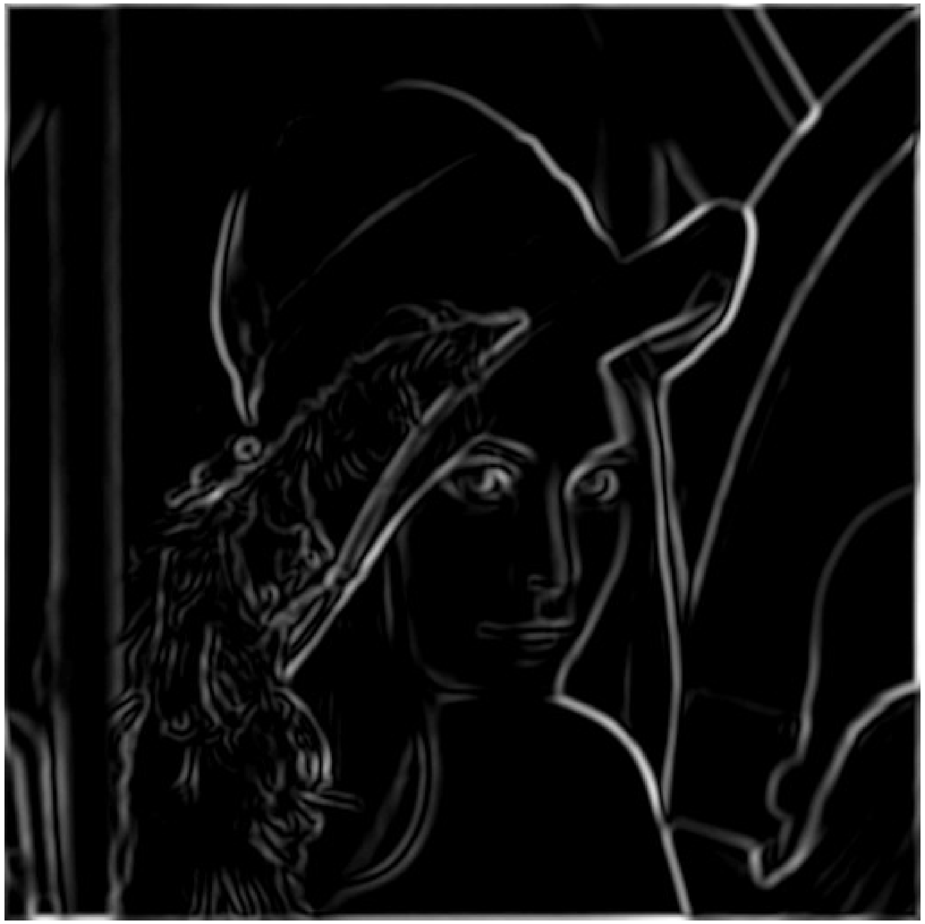
- original image (Lena)

The Canny edge detector



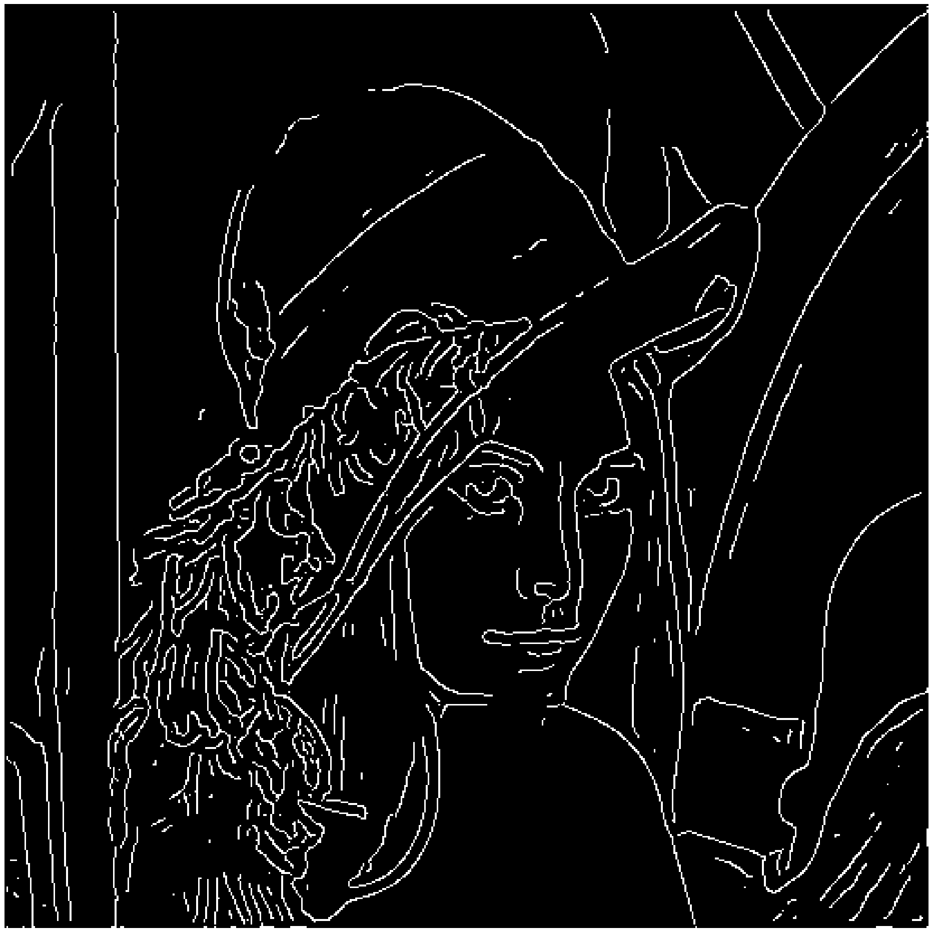
- norm of the gradient

The Canny edge detector



- thresholding

The Canny edge detector

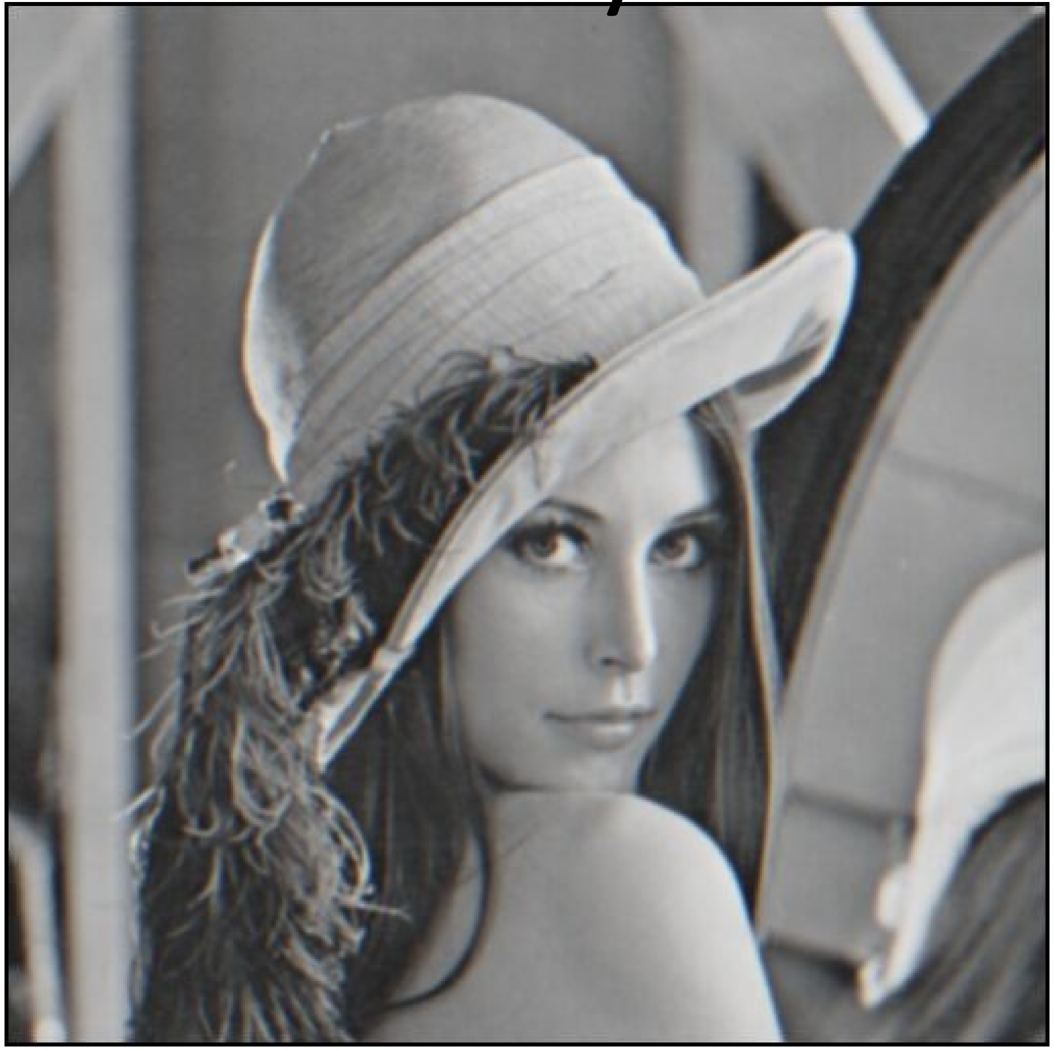


- thinning
  - (non-maximum suppression)

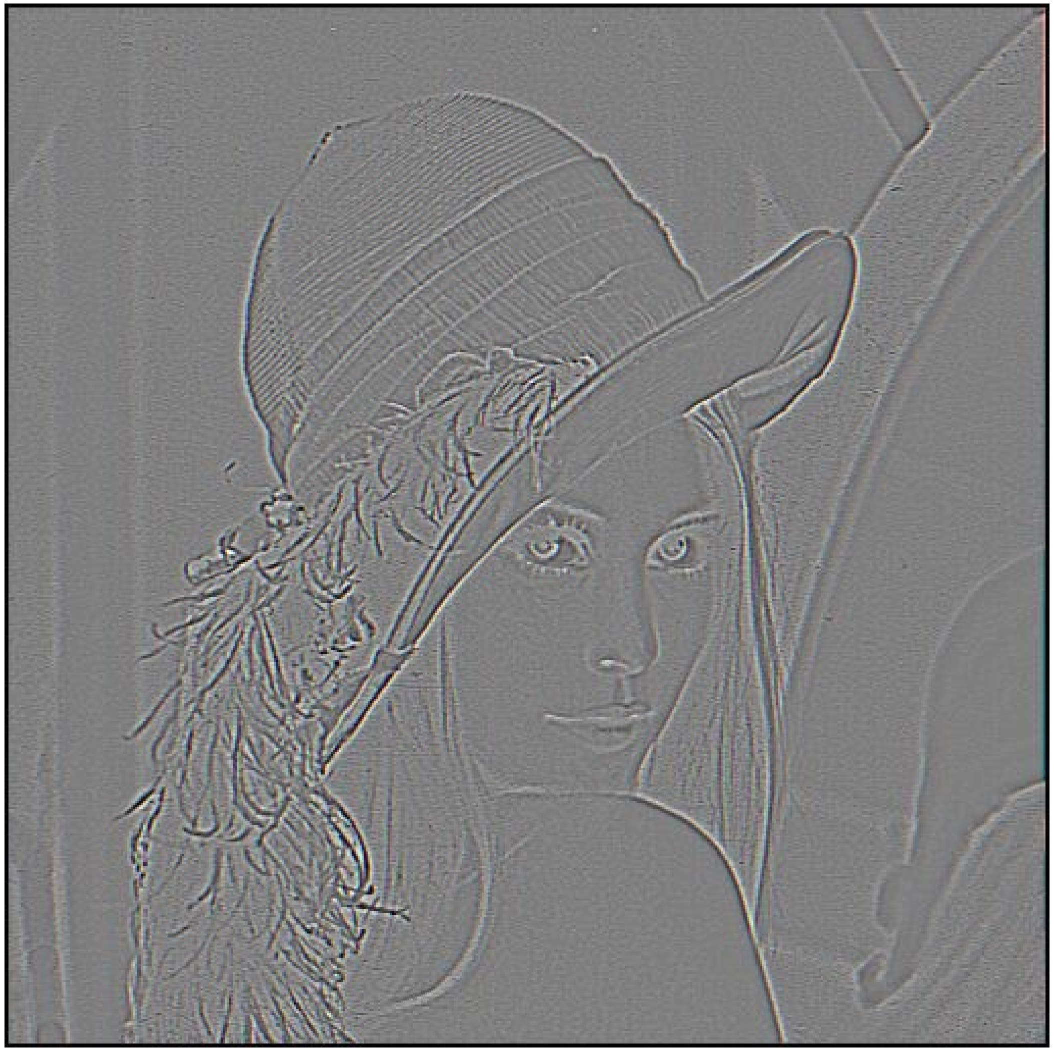
Edge detection by subtraction

original

Edge detection by subtraction

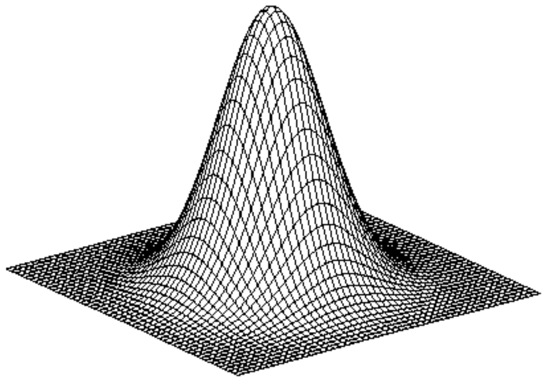

smoothed (5x5 Gaussian)

Edge detection by subtraction

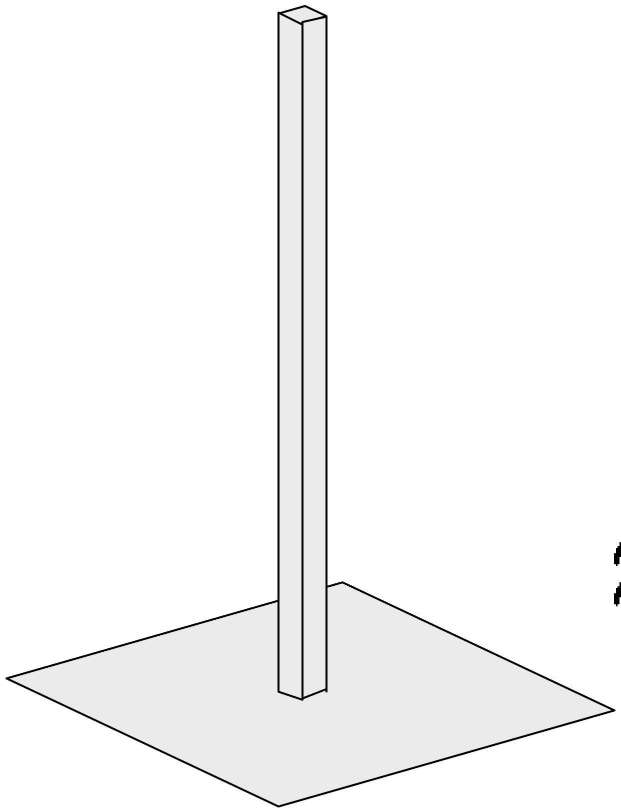


smoothed – original

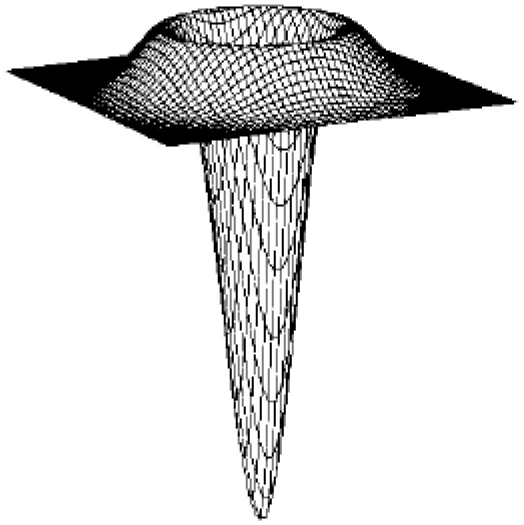
Gaussian - image filter


Gaussian

$-$


delta function

$\approx$


Laplacian of Gaussian

Criteria for a Good Contour Representation

- Efficiency: A Simple, Compact representation.
- Accuracy: Fit image features accurately.
- Effectiveness: Suitable for the operations performed in the later stages of the application.

Accuracy determined by:

- Form of the curve used to model the contour.
- Performance of the Curve fitting Algorithm.
- Accuracy of the estimates of edge location.

Contour

Representing a Contour

The simplest representation of a contour is an ordered list of edge pixels.

- Accurate: As accurate as the location estimates for the edges.
- Efficiency: Not efficient; least compact and simplest.
- Effectiveness: Not effective for subsequent image analysis.

Representing a Contour

A more powerful representation is to fit a curve having some analytical description (polygonal curves, line-segments etc…).

- Accuracy: ↑ (∵ errors in edge location reduced through averaging)
- Efficiency: ↑ (∵ provides more compact representation)
- Effectiveness: ↑ (∵ more appropriate representation provided for subsequent operations)

Concepts of Contour

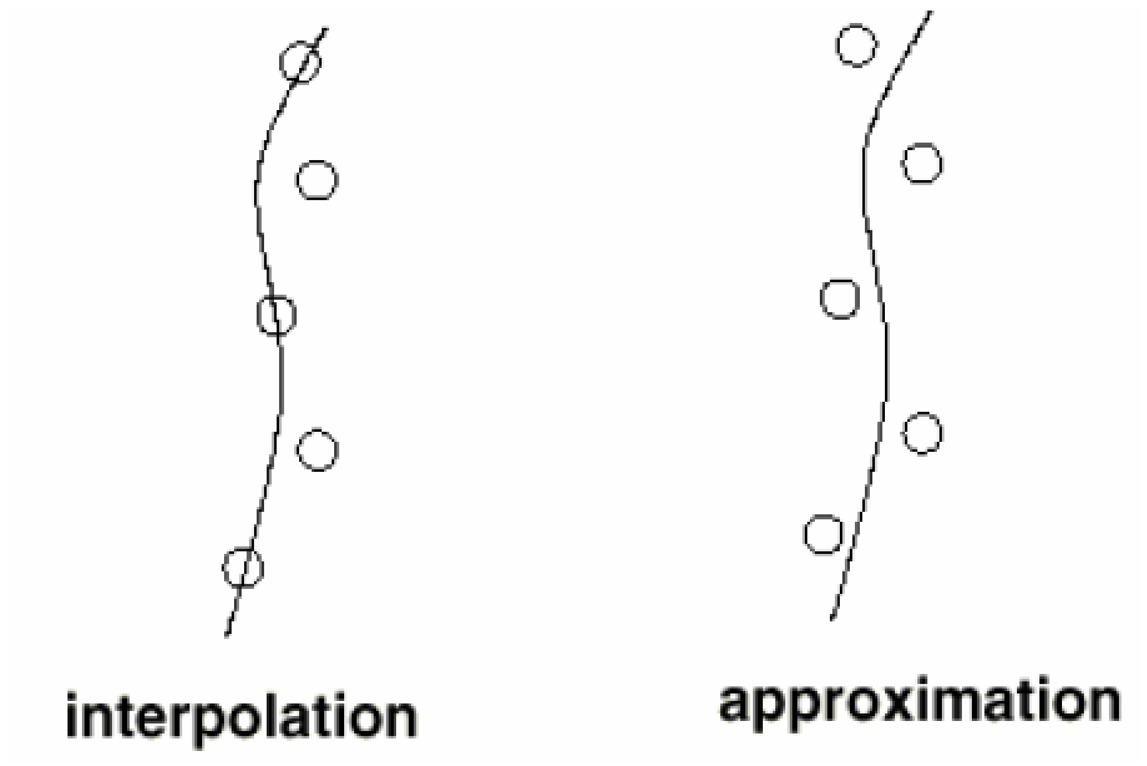
- A contour is a sequence of 8-connected pixels in an image where each pixel can be reached from any other pixel in the contour. An open contour has two end pixels, each with only one neighbor, and zero or more interior pixels, each with exactly two neighbors. A closed contour has no end pixels. All pixels in a closed contour have exactly two neighbors.

- The length of a contour is computed from the sum of distances between adjacent pixels in the contour. Two adjacent pixels that lie horizontally or vertically are considered to be apart by 1 pixel. All other adjacent pixels are considered to be apart by 2 pixels.

Definition

- A curve *interpolates* a set of points if it passes through the points.
- *Approximation* is fitting a curve passing close to the points but not necessarily through the points.
- An *edge list* is an ordered set of edge points/fragments.
- A *contour* is an edge list or the curve that has been used to represent the edge list.
- A *boundary* is the closed contour that surrounds a region.

Interpolation v/s Approximation



Geometry of Curves

- Representing Planar Curves
  - Explicit form y = f(x)
  - Implicit form f(x,y) = 0
  - Parametric form (x(u), y(u)), u being some parameter
- Length of a curve is given by the arc length:

$$\int_{u1}^{u2} \sqrt{\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2}\, du$$

- Unit tangent vector:

$$\mathbf{t}(u) = \frac{\mathbf{p}'(u)}{|\mathbf{p}'(u)|} \qquad \mathbf{p}(u) = (x(u),y(u))$$

- The Normal to the curve is the derivative of the tangent:

$$\mathbf{n}(u) = \mathbf{p}''(u)$$

Digital Curves

- Let $p_i = (x_i, y_i)$ be the coordinates of edge 'i' in the edge list.

- Estimate the slope using edge points that are not adjacent :
  - Left k-slope: direction from p_{i-k} to p_i
  - Right k-slope: direction from p_i to p_{i+k}
  - K-curvature: difference between the left and right k-slopes.
- Length of digital curve

$$S = \sum_{i=2}^{n} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

- Distance between end points

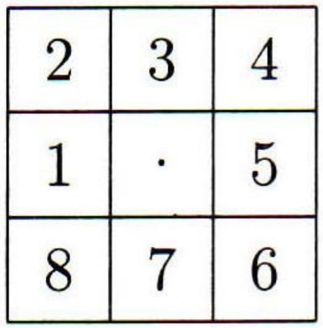
$$D = \sqrt{(y_n - y_1)^2 + (x_n - x_1)^2}$$

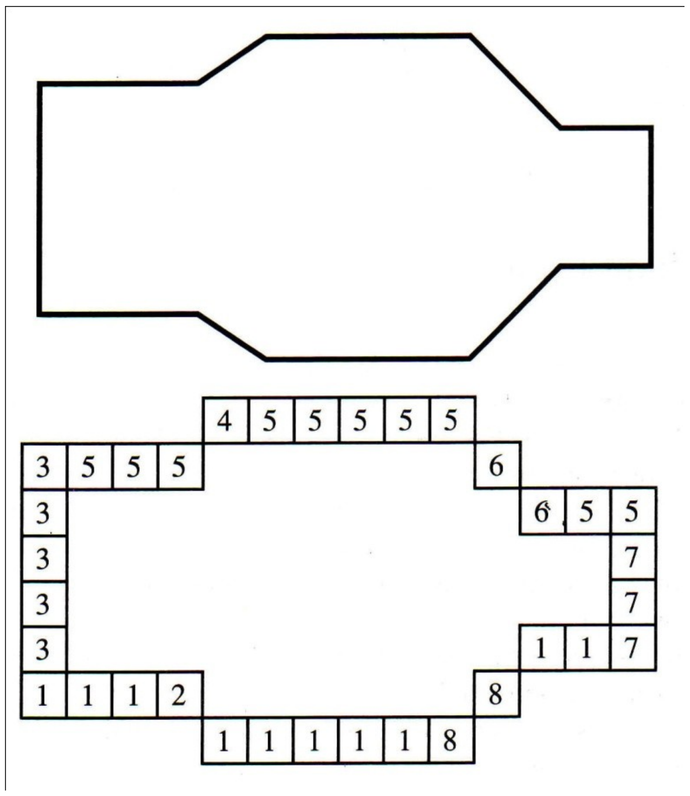
Chain Code

- A notation for recording list of edge points along a contour.
- Specifies contour direction at each edge in the list.
- Directions are quantized into one of the 8 directions.
- Starting at 1st edge in the list and going clockwise around the contour, the direction to the next edge is specified using one of the 8 chain codes. The direction is the chain code for the 8-neighbour of the edge.
- Represents an edge list by the 1st edge coordinates and the list of chain codes leading to the subsequent edges.

Chain Code: Example

A Curve and its Chain Code





Chain Code

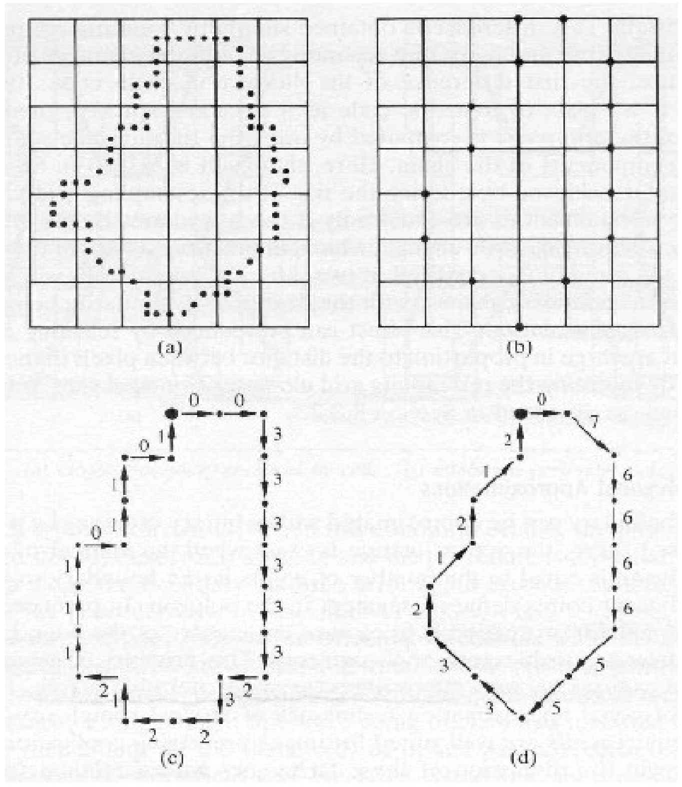
Algorithm:

1. Start at any boundary pixel A.

2. Find the nearest edge pixel and code its orientation. In case of a tie, choose the one with the largest (or smallest) code value.

3. Continue until there are no more boundary pixels.

- Chain code of a boundary depends on the starting point.
- Limited set of directions.
- Sensitive to noise.

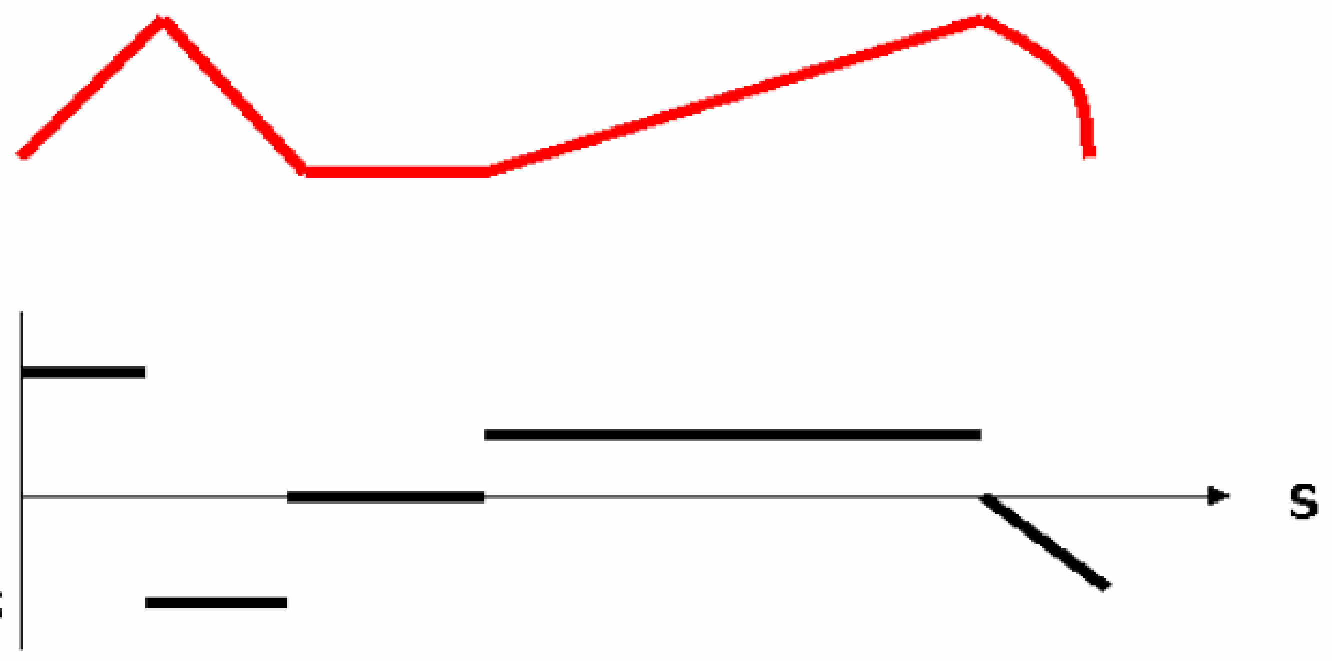


chain code: 0033333...01

Slope Representation (Ψ-s plot)

- A continuous version of the chain code.
- For contour representation using arbitrary tangent directions, rather than the limited set of tangent directions allowed by the chain code.
- Representation of the contour shape / a compact description of the original contour shape.
- Different starting points cause a shift in the s-axis.
- Rotations cause a shift in the Ψ axis.
- Not very tolerant to noise.
- For a closed contour this plot is periodic.

Slope Representation



Slope Representation

- Start at the first edge point and go clockwise around the contour.

- Estimate the slope Ψ and the arc length s.

- Plot the Slope versus the Arc-length.

Slope Density Function

- Histogram of the slopes (tangent angles) along a contour.
- Orientation of the object can be determined using correlation of slope histograms of model contour with that of an image contour.
- It can be very useful for object recognition.

Curve Fitting: Four Models

Next, we approximate a contour to get a better compact representation. We start with straight lines, and then move on to circular arcs or more complicated primitives.

Four Curve Fitting Models

- Line Segments
- Circular arcs
- Conic sections
- Cubic splines

Techniques for fitting the curve models to edges are carried out with the assumption that the edge locations are sufficiently accurate that selected edge points can be used to determine the fit. Further, effective approximation methods that can handle errors in the edge locations are discussed.

Goodness of a Fit

Let d_i be the distance of edge point i from a line.

All methods depend on the error between the fitted curve and the candidate points forming the curve.

- **Maximum absolute error** measures how much the points deviate from the curve in the worst case.

$$MAE = \max_i |d_i|$$

- **Mean squared error** gives an overall measure of the deviation of the curve from the edge points.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} d_i^2$$

Goodness of a Fit

- **Normalized maximum error** is the ratio of the maximum absolute error to the length of the curve (error becomes independent of the length of the curve, i.e., unitless.)

$$NMAE = \frac{\max_i |d_i|}{L}$$

- **Ratio of curve length to the end point distance** is a good measure of the complexity of the curve.

Curve fitting methods

- The use of straight line segments (polylines) is appropriate if the image consists of straight lines. It is the starting point for fitting other models.
- Circular arcs are a useful representation for estimating curvature.
- Conic sections provide a convenient way to represent sequences of line segments and circular as well as elliptic and hyperbolic arcs .
- Cubic splines are good to model smooth curves.

Polyline Representation

- A polyline is a sequence of line segments joined end to end. The ends of each line are edge points in the original edge list. The points where line segments are joined are called vertices.
- A quick revision
  - Equation of line passing from two points (*x*1, *y*1), (*x*2, *y*2):

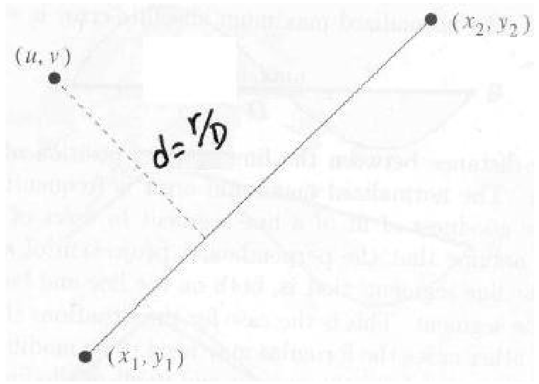
$$x(y_1 - y_2) + y(x_1 - x_2) + y_2x_1 - y_1x_2 = 0$$

  - Distance *d* of edge point (*u*, *v*) from the line:

$$d = \frac{r}{D}$$

$$r = u(y_1 - y_2) + v(x_1 - x_2) + y_2x_1 - y_1x_2$$

  - *D* is the distance between (*x*1, *y*1) and (*x*2, *y*2).



Types of Polyline Representation

Polyline representations are more economical than edge points. We can make the errors as small as desired by splitting the contour into very small line segments.

Types:

- Polyline Splitting (top-down)
- Segment Merging (bottom-up)
- Split and Merge
- Hop-Along Algorithm

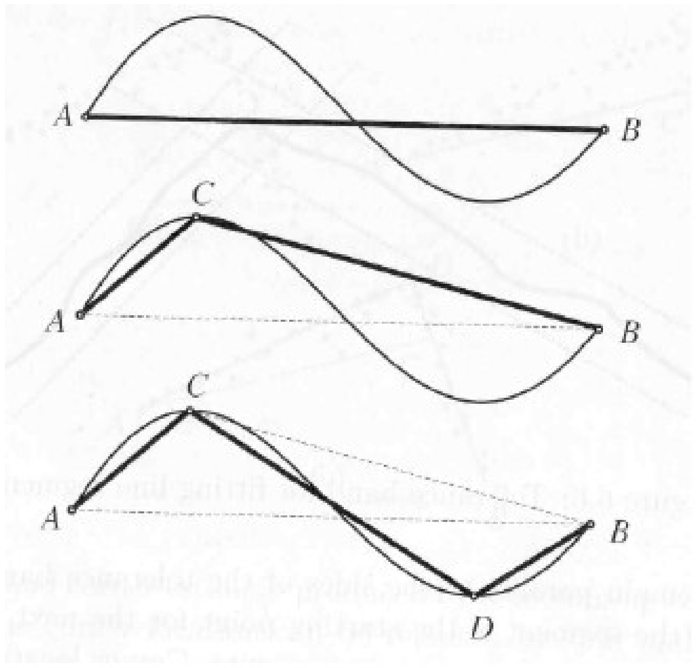
Polyline Splitting (Top-Down)

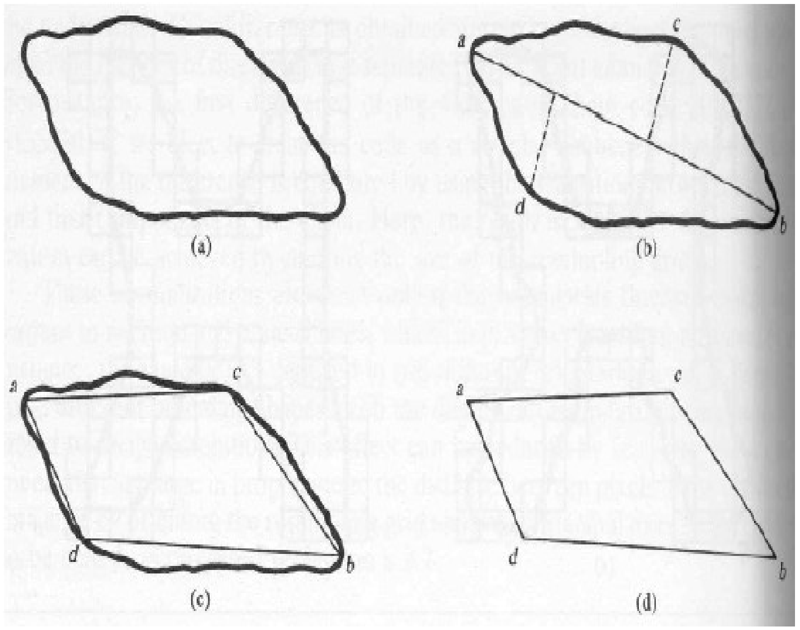
- Algorithm:
  - Take the line segment connecting the end points of the contour (if the contour is closed, consider the line segment connecting the two farthest points).
  - Find the farthest edge point from the line segment.
  - If the normalized maximum error of that point from the line segment is above a threshold, split the segment into two segments at that point (i.e., new vertex).
  - Repeat the same procedure for each of the two sub-segments until the error is very small.

Polyline Splitting- Examples

Open Contour

Closed Contour





Segment Merging (Bottom-Up)

- Algorithm:
  - Use the first two edge points to define a line segment.
  - Add a new edge point if it does not deviate too far from the current line segment.
  - Update the parameters of the line segment using least-squares.
  - Start a new line segment when edge points deviate too far from the line segment.

Split and Merge

The accuracy of line segment approximations can be improved by interleaving merge and split operations.

- Algorithm:

  - After recursive subdivision (split), allow adjacent segments to be replaced by a single segment (merge).

  - Alternate applications of split and merge until no segments are merged or split.

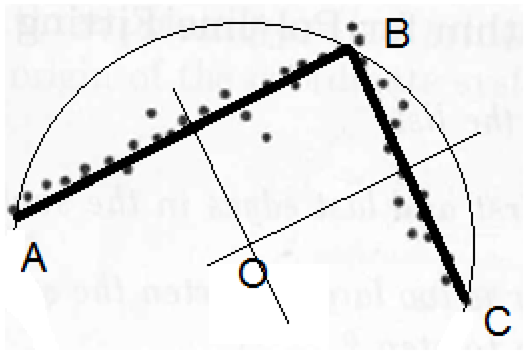
Hop-Along Algorithm

- Algorithm:

1. Start with the first k edges from the list.
2. Fit a line segment between the first and the last edges in the sub list.
3. If the normalized maximum error is too large, shorten the sub list to the point of maximum error. Return to step 2.
4. If the line fit succeeds, compare the orientation of the current line segment with that of the previous line segment. If the lines have similar orientations, replace the two line segments with a single line segment.
5. Make the current line segment the previous line segment and advance the window of edges so that there are k edges in the sub-list. Return to step 2.

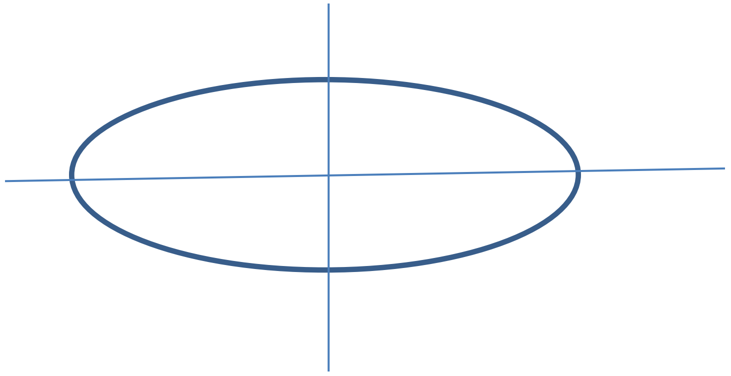
Circular Arcs

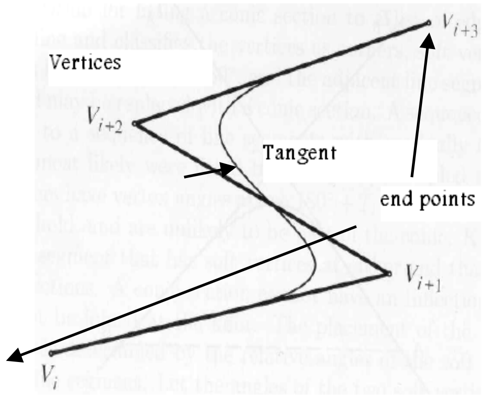
- Subsequences of the line segments can be replaced by circular arcs if desired. Replacing by arcs involves fitting circular arcs through the end points of two or more line segments, i.e. fitting is done on vertices in the polyline.
- The circular arc is a fit between the first and last end points and one other point. There can be several methods depending on how the middle point is chosen.



Conic Sections and Cubic Splines

- Conic sections correspond to hyperbolas, parabolas, and ellipses (2nd degree polynomials).
- Cubic Splines correspond to 3rd degree polynomials.
- Conic sections and cubic Splines allow more complex contours to be represented using fewer segments.
- Conic sections can be fit between three vertices in the polyline approximation.





References

[1] Three Dimension Computer Vision, O. Faugeras, MIT press, 1993.

[2] CV Course slide, Rob Fergus, New York Unv., http://cs.nyu.edu/~fergus/teaching/vision/index.html

[3] http://www-ee.uta.edu/Online/Devarajan/EE6358/

[4] http://www.ece.lsu.edu/gunturk/EE4780/EE4780.html