Edge Detection & Boundary Tracing

EE 528 Digital Image Processing

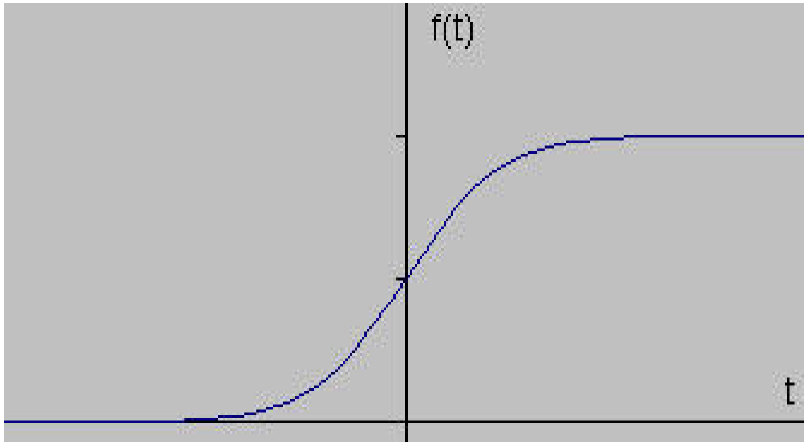
References

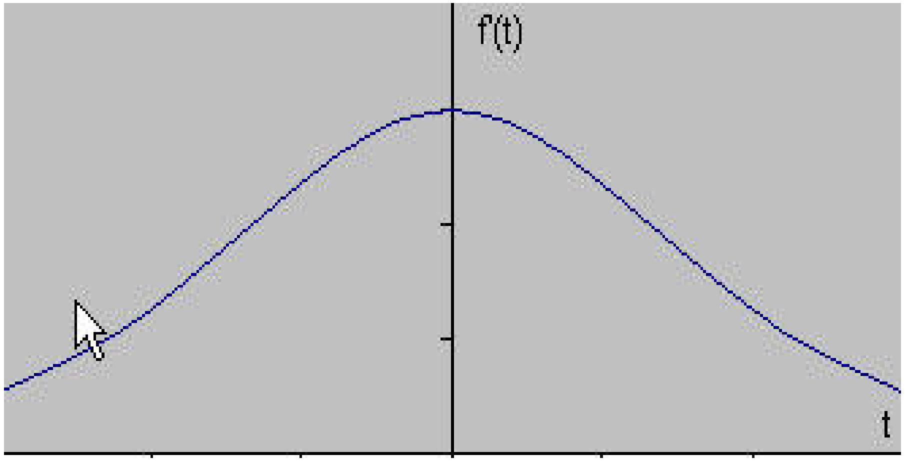
- A.K. Jain, Chapter 9
  - All details for edge detection given in this chapter
- Milan Sonka’s class notes on boundary tracing
- Other webpages listed where needed

Edge Detection Methods

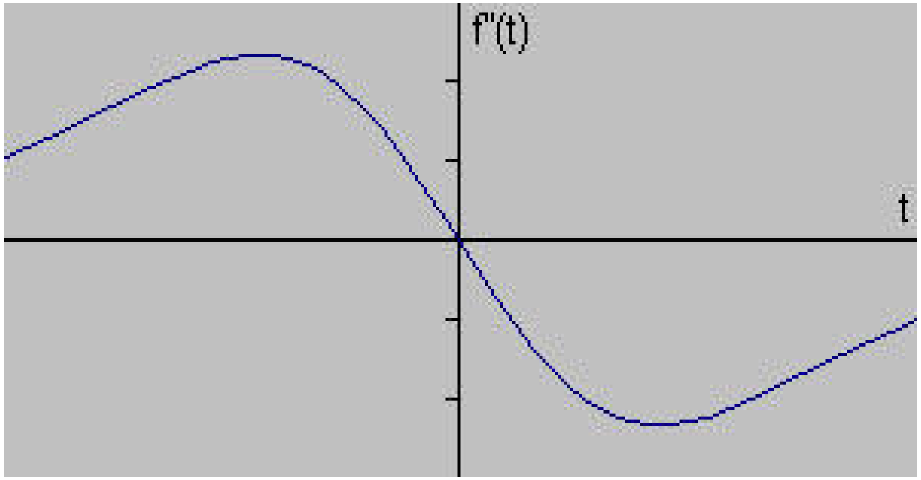
- Discrete approx. of gradient, & threshold the gradient norm image
  - Edge: large gradient magnitude
- Second derivative, & zero crossing detect
  - Edge: max or min of gradient along gradient direction
  - Weak edges (gradual variation) detected better, less chance of multiple edge responses
- Derivative: enhances noise, 2^{nd} derivative: worse
- Band-pass filtering: some smoothing followed by taking the first or second derivative, e.g. LoG
- Compass operators

Edges in 1D





Taken from http://www.pages.drexel.edu/~weg22/edge.html



Edge detection operators

- First derivative: Sobel, Roberts, Prewitts operators
  - Smooth in one direction, differentiate in the other
  - Apply in x and y directions, and take norm of the result
  - Arctan(G_y/G_x) = gradient direction (perpendicular to edge direct^n)
- Second derivative + smoothing: Marr-Hildreth operator or LoG
  - Gaussian prefiltering followed by computing Laplacian
  - Works better when grey level transitions are smooth
  - An approximation to LoG is the Mexican hat (difference of Gaussians of different variance)
- Compass: directional first derivative masks (Sobel or Prewitts)
- Canny's edge detector: most commonly used.

Examples

First derivative method:
misses some edges

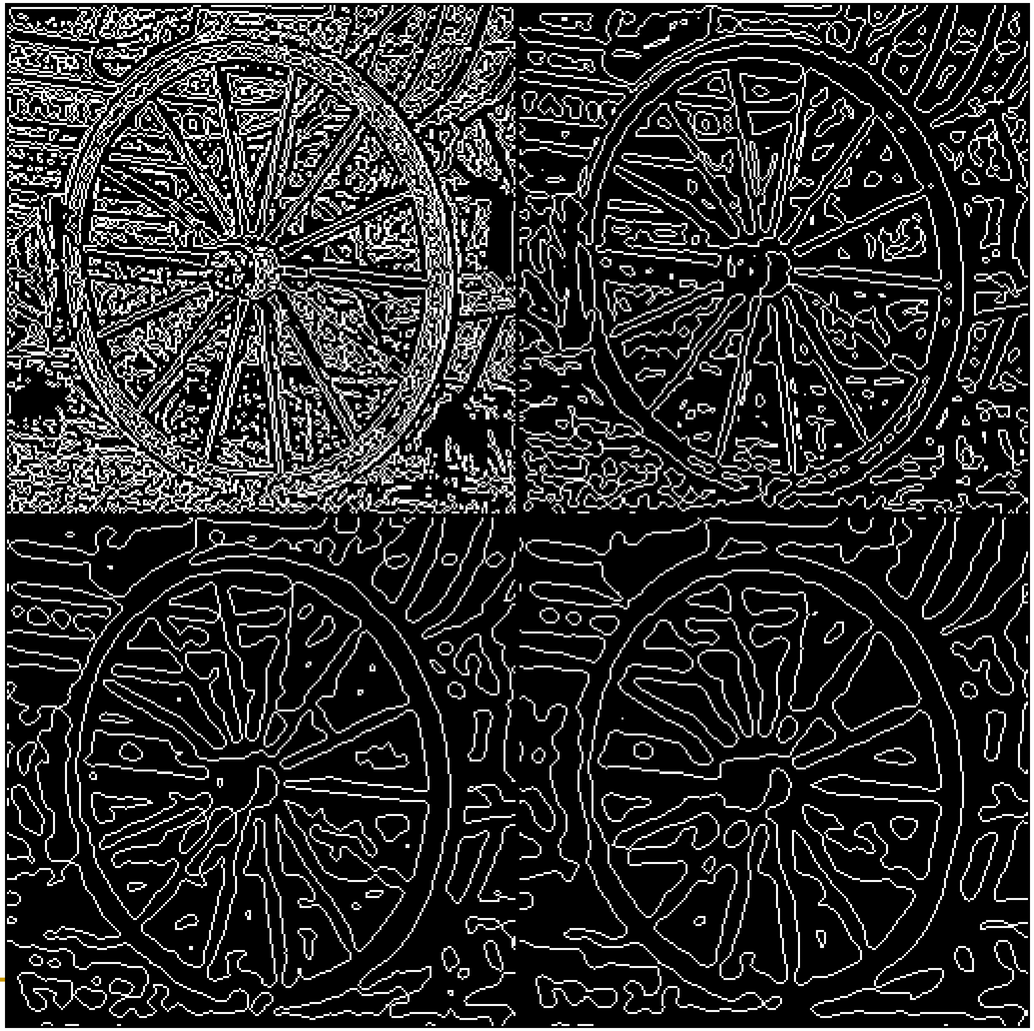
Laplacian:
more sensitive to noise

LoG edge detection

- Zero crossings always lie on closed contours and so the output from the zero crossing detector is usually a binary image with single pixel thickness lines showing the positions of the zero crossing points.

- Often occur at `edges' in images, but also occur anywhere where both x and y gradients change sign
  - e.g. occur where roughly uniform intensity (very small image gradient which increases & decreases)

LoG with increasing sigma



Detecting zero crossings

- Simplest: threshold the LoG image, i.e. mark all points with LoG magnitude below a threshold as zero
  - Problem: multiple edge responses
- Choose points where LoG magnitude smaller than all its 4 neighbors
  - Risk of missing some edge points
- Zero crossing: LoG sign change in at least one direction

Canny's edge detector

- 1983, MS student at MIT

- Designed an operator that minimizes probability of missing an edge, probability of false detection of edges, good localization

- Restricted solution to linear shift-invariant operators

Main Idea of Canny

- Smooth the image using a Gaussian kernel: reduce false alarms
- Take gradient & compute gradient magnitude & gradient direction (quantify direction to multiples of 45 degrees)
- Perform non-maximal suppression: localization
  - Suppress a point if its gradient magnitude is smaller than either of its two neighbors along the gradient direction
- Hysteresis: two thresholds TL, TH
  - Suppress all points with magnitude < TH
  - If a point has magnitude > TL and is linking two points with magnitude > TH, then keep it : reduce misses

Applying Canny

Taken from http://www.pages.drexel.edu/~weg22/edge.html

Some edges missing

(before hysteresis step)

Compass Operators

- Compute magnitude of directional derivative in 4 directions – 0, 45, 90, 135 degree
- Maximum of the derivative values gives gradient magnitude
- Threshold gradient magnitude

- Alternatively, if only want to look for 45 degree edges: can do that.

Boundary Tracing & Edge Linking

Boundary Tracing

- Given a “segmented” image (an image with foreground pixels labeled 1 and background pixels labeled zero), trace either boundary of the foreground
  - May need to trace inner boundary (outermost pixels of foreground) or outer boundary (innermost pixels of background): bwtraceboundary command in MATLAB
  - Or if foreground, background labeled 1, -1, may use a zero level set searching method to get subpixel coordinates of boundary: contour command in MATLAB
- Segmentation: discussed in next handout, simplest way to segment is to threshold intensity values

Boundary Tracing Algorithm links

- http://www.mathworks.com/access/helpdesk_r13/help/toolbox/images/enhanc11.html

- http://www.icaen.uiowa.edu/~dip/LECTURE/Segmentation2.html#tracing

- http://www.imageprocessingplace.com/DIP/dip_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/index.html

MATLAB functions

- contour: gives you the contour location in sub-pixel coordinates. Need to have object & background labeled as 1, -1 (not 1,0)
- Inner boundary: gives the locations of the outermost pixels of the object
  - bwtraceboundary
  - bwboundaries (for multiple objects)

4 connected versus 8 connected

- 4 connected neighbors
- 8 connected neighbors

Algorithm 5.6: Inner boundary tracing

1. Search the image from top left until a pixel of a new region is found; this pixel P_0 then has the minimum column value of all pixels of that region having the minimum row value. Pixel P_0 is a starting pixel of the region border. Define a variable *dir* which stores the direction of the previous move along the border from the previous border element to the current border element. Assign

   (a) $dir = 0$ if the border is detected in 4-connectivity (Figure 5.13a)

   (b) $dir = 7$ if the border is detected in 8-connectivity (Figure 5.13b)

2. Search the 3×3 neighborhood of the current pixel in an anti-clockwise direction, beginning the neighborhood search in the pixel positioned in the direction

   (a) $(dir + 3)\ mod\ 4$ (Figure 5.13c)

   (b) $(dir + 7)\ mod\ 8$ if dir is *even* (Figure 5.13d)
   $(dir + 6)\ mod\ 8$ if dir is *odd* (Figure 5.13e)

   The first pixel found with the same value as the current pixel is a new boundary element P_n. Update the *dir* value.

3. If the current boundary element P_n is equal to the second border element P_1, and if the previous border element P_{n-1} is equal to P_0, stop. Otherwise repeat step (2).

4. The detected inner border is represented by pixels $P_0 \ldots P_{n-2}$.

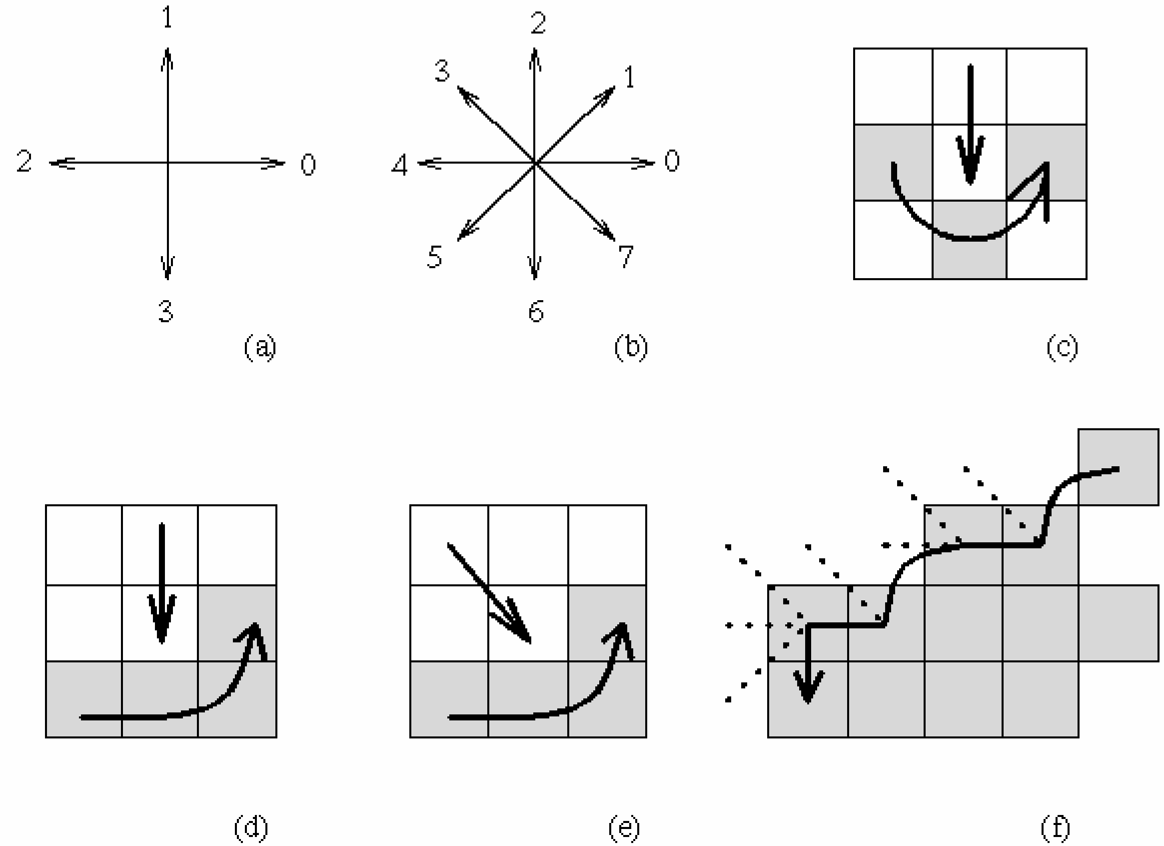


Figure 5.13 *Inner boundary tracing: (a) Direction notation, 4-connectivity, (b) 8-connectivity, (c) pixel neighborhood search sequence in 4-connectivity, (d),(e) search sequence in 8-connectivity, (f) boundary tracing in 8-connectivity (dashed lines show pixels tested during the border tracing).*

Outer Boundary tracing

Algorithm 5.7: Outer boundary tracing

1. Trace the inner region boundary in 4-connectivity until done.
2. The outer boundary consists of all non-region pixels that were tested during the search process; if some pixels were tested more than once, they are listed more than once in the outer boundary list.

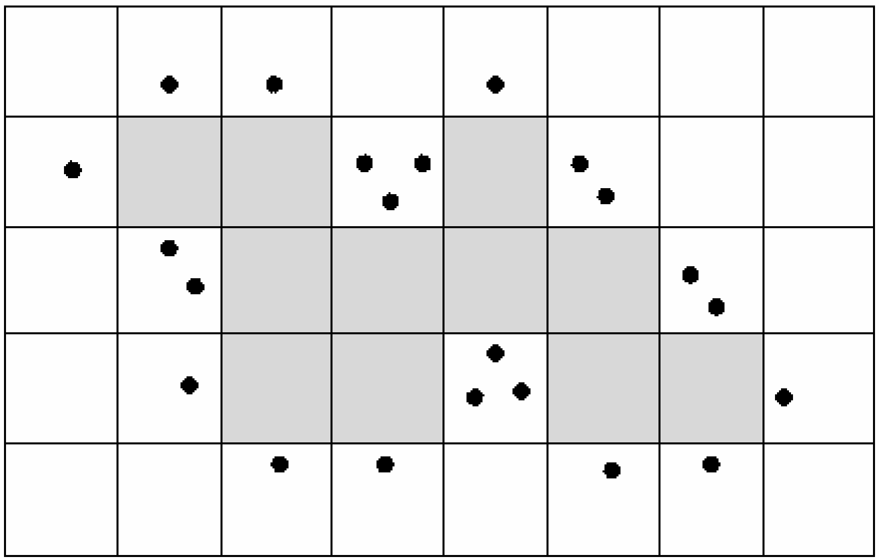


Figure 5.14 *Outer boundary tracing; • denotes outer border elements. Note that some pixels may be listed several times.*

Edge Linking

- Goal: take edge map, convert to a linked boundary representation
  - Can be closed or open boundary
- Convert the edge map with gradient magnitude and gradient directions into a weighted graph
- Use dynamic programming (based on Bellman's optimality principle) to find the shortest path from origin to destination
  - Much faster than brute force optimization
  - Idea explained in class
  - Also read pages 359-362 of AK Jain.

Main idea of Dynamic Programming

- Given an objective function S(x1,…xN) where x1,...xN are the vertices and S is the sum of edge weights when traversing in the sequence x1, x2,…xN
- Find ϕ(xN) = $\max_{x1,\ldots x\{N-1\}}$ S(x1,…xN). The argument maximizing this gives you the path
  - S can be split as:

  S(x1,…xN) = S(x1,…x{N-1}) + f(x_{N-1},x_N)
- Whenever the above holds, the max can be simplified as

ϕ(xN) = $\max_{x\{N-1\}}$[f(x{N-1},xN) + $\max_{x1,\ldots x\{N-2\}}$ S(x1,…x{N-1})]

= $\max_{x\{N-1\}}$[f(x{N-1},xN) + ϕ(x{N-1})]

  - This can be implemented as a recursive algorithm
- Details in class or in the book