Edge Detection

CS 450: Computer Vision

slides by Dr. Bryan Morse

Edges and Gradients

- Edge: *local* indication of an object transition
- Edge detection: local operators that "find" edges (usually involves convolution)
- Local intensity transitions are indicated by the *gradient*:

$$\nabla I = \begin{bmatrix} \frac{\partial}{\partial x} I \\ \frac{\partial}{\partial y} I \end{bmatrix}$$

- Interpretation:
  - Gradient magnitude $\|\nabla I\|$: edge "strength"
  - Gradient orientation $\phi(\nabla I)$: cross-edge direction

Prewitt Kernels

Idea: central finite differences

Central difference:

$$\frac{dI}{dx} \approx [I(x+1) - I(x-1)]/2$$

or for two-dimensional images:

$$\frac{\partial I}{\partial x} \approx [I(x+1, y) - I(x-1, y)]/2$$

This corresponds to the following convolution kernel:

-1	0	1

-1
0
1

Prewitt Kernels

Or for more robustness to noise, smooth in the other direction:

-1	0	1
-1	0	1
-1	0	1

$\partial/\partial x$

-1	-1	-1
0	0	0
1	1	1

$\partial/\partial y$

Sobel Kernels

Or, giving more weight to the central pixels when averaging:

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

$\partial/\partial x$

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

$\partial/\partial y$

- These kernels can also be thought of as 3×3 approximations of the first derivative of a small Gaussian
- Can be though of as blurring by a small Gaussian to remove noise, then taking the derivative:

$$\frac{\partial}{\partial x}(I * G) = I * \frac{\partial}{\partial x} G \approx I * \text{Sobel(x)}$$

From Gradient Magnitude to Edges

The gradient magnitude gives a measure *at every pixel* of the "edginess" of each pixel:

$$\|\nabla I(x, y)\|$$

Somehow, you have to still find the *best* edges:

- Threshold (global or local), etc.
- Local maxima of gradient magnitude

$$\vdots$$

Using Second Derivatives

- Classic maximum test from calculus:

$$x \text{ is a extremum of } f(x) \text{ if } \frac{df}{dx}(x) = 0$$

- Extend this idea to find maximal first derivatives:

$$x \text{ is a extremum of } \frac{df}{dx}(x) \text{ if } \frac{df^2}{dx^2}(x) = 0$$

Using Second Derivatives

a b
c

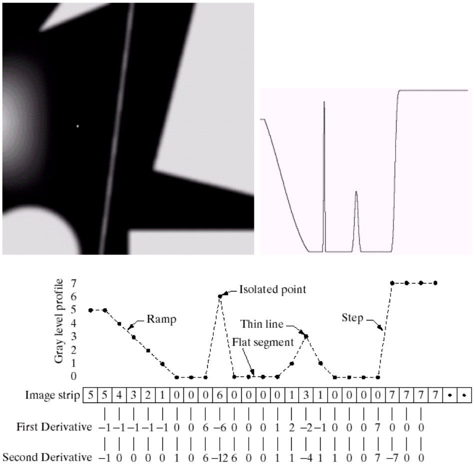

FIGURE 3.38 (a) A simple image. (b) 1-D horizontal gray-level profile along the center of the image and including the isolated noise point. (c) Simplified profile (the points are joined by dashed lines to simplify interpretation).

Laplacian

- The *Laplacian* is defined mathematically as

$$\nabla^2 = \nabla \cdot \nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

- When we apply it to an image, we get

$$\nabla^2 I = \left(\begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \right) I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Marr-Hildreth Edge Detection

Idea: approximate finding maxima of gradient magnitude (edges) by finding places where

$$\nabla^2 I(x, y) = 0$$

- Can't always find discrete pixels where the Laplacian is *exactly* zero—look for *zero crossings* instead.

Laplacian Operators

- Second difference:

$$\begin{aligned} \frac{d^2 I}{dx^2} &\approx [I(x+1) - I(x)] - [I(x) - I(x-1)] \\ &= I(x+1) - 2I(x) + I(x-1) \end{aligned}$$

- The Laplacian is one of these in the x direction added to one of these in the y direction:

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 1 & -2 & 1 \\ \hline 0 & 0 & 0 \\ \hline \end{array} \quad + \quad \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 0 & -2 & 0 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \quad = \quad \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$$

Derivatives of Gaussians

- Gaussian kernel for noise removal:

$$G(x, y) = (2\pi\sigma)^{-d/2}\, e^{-r^2/2\sigma^2}$$

  where $r^2 = x^2 + y^2$ and d is the dimensionality (images=2)
- Can solve in closed form for the first- and second-order derivatives of the Gaussian (including the Laplacian)
- Can convolve with these directly—
  exactly the same as blurring first then applying derivatives

Difference of Gaussians

- Another property of the Laplacian of Gaussian:

$$\nabla^2 G = \frac{\partial}{\partial \sigma} G$$

- We can thus approximate the Laplacian by the difference of one Gaussian and a just-smaller one:

$$\nabla^2 G \approx G(x, y; \sigma_1) - G(x, y; \sigma_2)$$

  This is the *Difference of Gaussians* (DoG) kernel.

- Ratio (σ_1/σ_2) for best approximation is about 1.6. (Some people like $\sqrt{2}$.)

Combining Both First- and Second-Derivatives

- Laplacian zero crossings:

$$\nabla^2 I = 0$$

- Problem:
  Tells you the gradient magnitude is at a maximum, *not* how strong it is—lots of spurious edges.
- Idea:
  Combine the two measures

$$\nabla^2 I = 0 \text{ and } \nabla I > T$$

Canny Edge Detector

- Problem with Laplacian zero-crossings: adds the principal curvatures together—it doesn't really determine a maximum of gradient magnitude in any one direction.
- The *Canny Edge Detector* defines edges as *zero-crossings of second derivatives in the direction of greatest first derivative*.

  The Direction of Greatest First Derivative
  : This is simply the gradient direction.

  The Second Derivative in The Direction of ...
  : We can compute this using the matrix of second derivatives (Hessian).

  Zero Crossings of...
  : As with the Marr-Hildreth edge detector, we'll use positive-negative transitions to "trap" zeroes.

- Gives connected edges much like the Laplacian operator but more accurately localize the edge.

Now What?

Now you have potential edge points, how do you get contours?

- Thresholding gradient magnitude
- Threshold, then “relax” the labeling
  - Thresholding with hysteresis (Canny)
  - Edge relaxation algorithms using other criteria
- Edge linking (including postprocessing)
- Connected loci of local maxima (ridges)
- Maximum-magnitude contours/paths
  (Turns into optimization problem—we’ll come back later.)

Representation

Once you have contours, how do you represent them?

- Chain codes (4- or 8-connected directions)
- Differential chain codes (4- or 8-connected *relative* directions)
- Polylines (fit with short line segments)
- Arc-length parameterization
  - Position
  - Tangent orientation
  - Curvature
  - Distance to some central point
  - ...
- Fourier descriptors
- Many other ways to represent curves

Representation

What do you want to do with the representation?

- Reproduction
- Matching
  - Translated
  - Rotated
  - Scaled
  - Noise and variation (smooth the curves)
  - ...

Want to be *invariant* to as much as possible