Efficient Implementation

Both, the Box filter and the Gaussian filter are separable:

- First convolve each row of input image \( I \) with a 1-D row filter \( R \) to produce an intermediate image \( J \)
  \[ J = R * I \]

- Then convolve each column of \( J \) with a 1-D column filter \( C \).
  \[ O = C * J \]
  \[ = C * (R * I) \]
  \[ = (C * R) * I \]

The Fourier Transform and Convolution

- If \( H \) and \( G \) are images, and \( F(.) \) represents Fourier transform, then
  \[ F(H*G) = F(H)F(G) \]
  Or
  \[ H*G = F^{-1}(F(H)F(G)) \]

  - This is referred to as the Convolution Theorem
  - Fast Fourier Transform: complexity \( O(n \log n) \) -> complexity of convolution is \( O(n^2) \) not \( O(n^2) \).
  - Thus, one way of thinking about the properties of a convolution is by thinking of how it modifies the frequencies of the image to which it is applied.
  - In particular, if we look at the power spectrum, then we see that convolving image \( H \) by \( G \) attenuates frequencies where \( G \) has low power, and amplifies those which have high power.

Fourier Transform

Discrete Fourier Transform (DFT) of \( I[x,y] \)

\[ F[u,v] \equiv \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} I[x,y]e^{-\frac{2\pi i}{N}(ux+vy)} \]

Inverse DFT

\[ I[x,y] \equiv \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F[u,v]e^{\frac{2\pi i}{N}(ux+vy)} \]

x,y: spatial domain
u,v: frequency domain
N by N image
Implemented via the “Fast Fourier Transform” algorithm (FFT)

Announcements

- HW2 due on Tuesday
- HW3 will be on stereo
- Andrew will teach first stereo lecture on Tuesday
- Kriegman won’t hold office hours
- No class on Thursday. Makeup to be announced
Physical causes of edges

1. Object boundaries
2. Surface normal discontinuities
3. Reflectance (albedo) discontinuities
4. Lighting discontinuities

Noisy Step Edge

- Derivative is high everywhere.
- Must smooth before taking gradient.

Edge is Where Change Occurs: 1-D

- Change is measured by derivative in 1D

- Biggest change, derivative has maximum magnitude
- Or 2nd derivative is zero.

Numerical Derivatives

Take Taylor series expansion of \( f(x) \) about \( x_0 \):

\[
f(x) = f(x_0) + f'(x_0)(x-x_0) + \frac{1}{2} f''(x_0)(x-x_0)^2 + \cdots
\]

Consider samples taken at increments of \( h \) and first two terms, we have

\[
f(x_0+h) = f(x_0) + f'(x_0)h + \frac{1}{2} f''(x_0)h^2
\]

\[
f(x_0-h) = f(x_0) - f'(x_0)h + \frac{1}{2} f''(x_0)h^2
\]

Subtracting and adding \( f(x_0+h) \) and \( f(x_0-h) \) respectively yields

\[
f'(x_0) = \frac{f(x_0+h) - f(x_0-h)}{2h}
\]

\[
f''(x_0) = \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2}
\]

Convolv with

First Derivative: [-1 0 1]
Second Derivative: [1 -2 1]
Implementing 1-D Edge Detection

1. Filter out noise: convolve with Gaussian

2. Take a derivative: convolve with [-1 0 1]
   - We can combine 1 and 2.

3. Find the peak: Two issues:
   - Should be a local maximum.
   - Should be sufficiently high.

2D Edge Detection: Canny

1. Filter out noise
   - Use a 2D Gaussian Filter. \( J = I * G \)

2. Take a derivative
   - Compute the magnitude of the gradient:
     \[
     \nabla J = (J_x, J_y) = \left( \frac{\partial J}{\partial x}, \frac{\partial J}{\partial y} \right)
     \]

Next step

Can next step be the same for 2-D edge detector as in 1-D detector??

NO!!

Smoothing and Differentiation

- Need two derivatives, in x and y direction.
- Filter with Gaussian and then compute Gradient, OR
- Use a derivative of Gaussian filter
  - because differentiation is convolution, and convolution is associative

Directional Derivatives

\[
\cos \theta \frac{\partial G_\alpha}{\partial x} + \sin \theta \frac{\partial G_\alpha}{\partial y}
\]
There are three major issues:
1. The gradient magnitude at different scales is different; which scale 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?

There is ALWAYS a tradeoff between smoothing and good edge localization!

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: which point is the maximum, and where is the next point on the curve?
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 $s$). Link together to create Image curve.

**Hysteresis Thresholding**

- Start tracking an edge chain at pixel location that is local maximum of gradient magnitude where gradient magnitude $> \tau_{\text{high}}$.
- Follow edge in direction orthogonal to gradient.
- Stop when gradient magnitude $< \tau_{\text{low}}$.
  - i.e., use a high threshold to start edge curves and a low threshold to continue them.

$\tau_{\text{high}}$  
$\tau_{\text{low}}$
### Why is Canny so Dominant

- Still widely used after 20 years.
  1. Theory is nice (but end result same).
  2. Details good (magnitude of gradient, non-max suppression).
  3. Hysteresis an important heuristic.
  4. Code was distributed.

### Boundary Detection

- Brightness
- Color
- Texture

### Corner Detection

*Precision is the fraction of detections that are true positives rather than false positives, while recall is the fraction of true positives that are detected rather than missed.*

*From Contours to Regions: An Empirical Evaluation, Arbelaez, M. Maire C. Fowlkes 2, and J. Malik, CVPR 2008*
Feature extraction: Corners and blobs

Why extract features?
• Motivation: panorama stitching
  – We have two images – how do we combine them?

Step 1: extract features
Step 2: match features

Corners contain more info than lines.
• A point on a line is hard to match.
The Basic Idea

- We should easily recognize the point by looking through a small window.
- Shifting a window in any direction should give a large change in intensity.

Edge Detectors Tend to Fail at Corners

Finding Corners

Intuition:

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

Distribution of gradients for different image patches

Formula for Finding Corners

We look at matrix:

\[
C(x,y) = \sum I_x I_y + \sum I_x^2 + \sum I_y^2
\]

Why this?

General Case:

Because \( C \) is a symmetric positive definite matrix, it can be factored as follows:

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

Where \( R \) is a 2x2 rotation matrix and \( \lambda \) is non-negative.
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 \( \lambda_2 > 0 \)?

Since \( C \) is symmetric, we have

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

We can visualize \( C \) as an ellipse with axis lengths determined by the eigenvalues and orientation determined by \( \mathbf{R} \).

**Ellipse equation:**

\[
\begin{bmatrix} u \\ v \end{bmatrix} \begin{bmatrix} \lambda_{max}^{-1/2} & 0 \\ 0 & \lambda_{min}^{-1/2} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}
\]

So, to detect corners

- Filter image with a Gaussian.
- Compute the gradient everywhere.
- Move window over image and construct \( C \) over the window.
- Use linear algebra to find \( \lambda_1 \) and \( \lambda_2 \).
- If they are both big, we have a corner.
  1. Let \( e(x,y) = \min(\lambda_1(x,y), \lambda_2(x,y)) \)
  2. \((x,y)\) is a corner if it’s local maximum of \( e(x,y) \) and \( e(x,y) > \tau \)

Parameters: Gaussian std. dev, window size, threshold