Computer Science 631
Lecture 10: Local operations, edge detection

Ramin Zabih
Computer Science Department
CORNELL UNIVERSITY

Outline

- Dynamic programming for local operations
  - Crow's method
  - Sliding sums (box filtering)
- Edge detection
  - Sobel filtering
  - Canny-style (gradient magnitude)
  - Marr-style (zero crossings)
- Smoothing

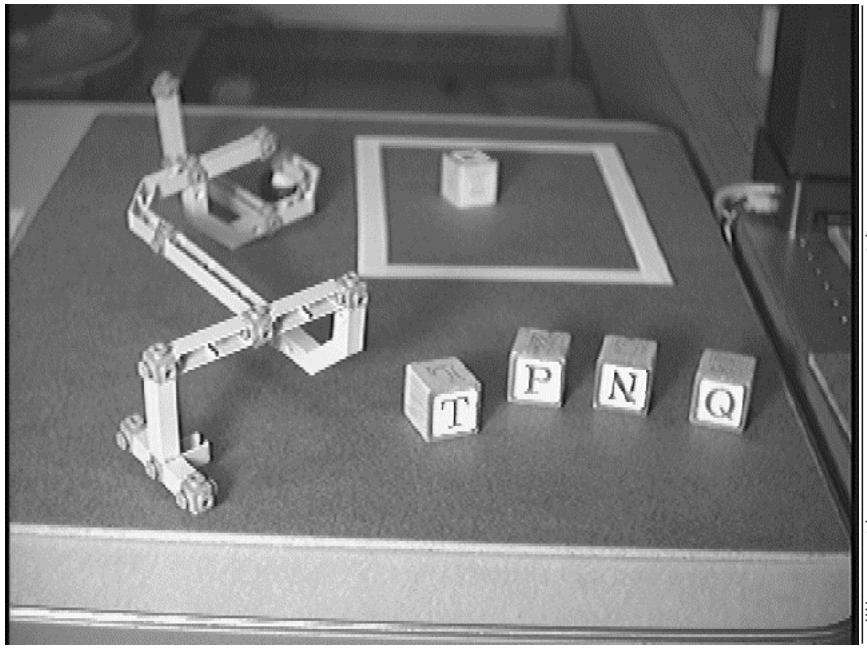
Local operations

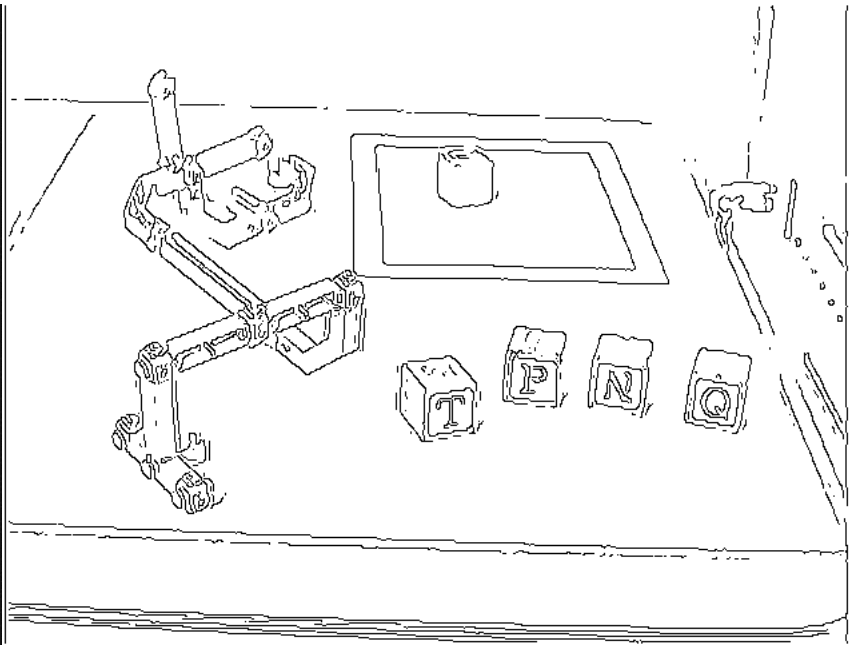
- Most image distortions involve
  - Coordinate changes
  - Color
  - Different spatial frequencies
- These last class of distortions center on local operations
  - Every pixel computes some function of its local neighborhood (window)
  - We will assume a square of radius *r*

Sliding sums

- There is a similar trick for computing the sum in all fixed-size rectangles
  - Exactly what we need for local averaging
  - To get the new sum, start with the old,
  - Then add (at right) and subtract (left) a column sum
  - To get a new column sum, take the column sum directly above
    - Then add (below) and subtract (above) an intensity

Edge detection



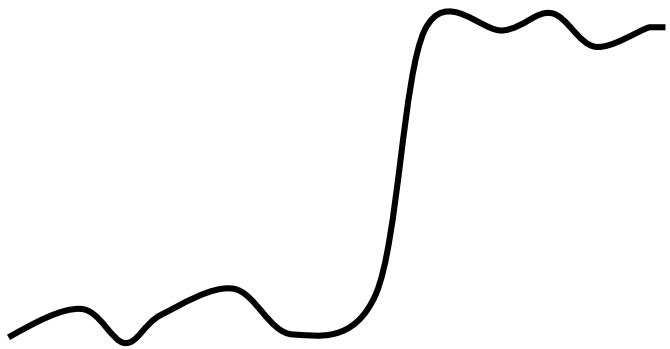


Input

Output

Basic idea behind edge detection

- Find places where the intensity is changing fast
  - Best illustrated with 1D examples
  - A cross-section of an edge might look like this:



How to find edges with calculus

- Recall that the first derivative tells you how fast the function is changing
  - So, find places where the first derivative is large!
- In 2D, we look at the gradient
  - Vector pointing “downhill” on the intensity surface
  - Gradient magnitude is the steepness

Gradient magnitude edge detection

- Historically, invented first
  - Also called “Canny-style”
- Compute the gradient, then threshold its magnitude
  - Can pick threshold adaptively, such as 3% edges
- Issue: how to compute the gradient?
  - How wide an area of support (local window)?

Local filtering and stencils

- Compute a weighted sum of local intensities
- Example: take the difference between the pixel on your left and on your right

  new[x,y] = old[x-1,y] - old[x+1,y]
- Natural representation is via a *stencil*

  [+1 0 -1] for the above example

  To compute a radius 1 average:

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Early edge detector stencils

- To compute a gradient we use two stencils
  - One gives *x* component, one *y*
  - "Look for" horizontal versus vertical edges
- Roberts (1965) used $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$
- One problem: answer is not on the pixel grid

Prewitt operator (1970)

- Prewitt: $\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ and $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$

- In a large window, most of the pixels are far away
  - With a 5-by-5 window, over half are at distance 2
- Solution: weigh the pixels near the center more

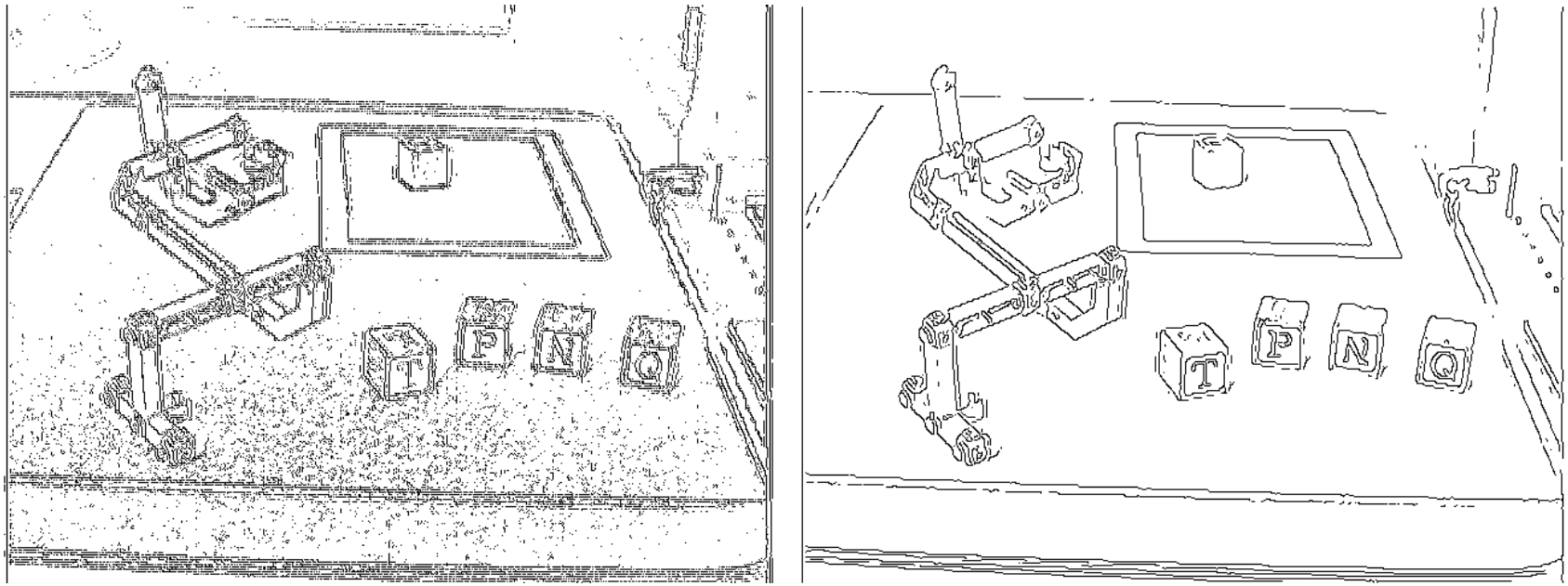
Sobel operator

- Standard (low-tech) edge detector
  - Increase the weight in the center a bit

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

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

These don’t work terribly well



Local operator solution

Smoothing and noise reduction

- One reason is that there is noise in the image formation process
  - Suppose that a single pixel has a large value due to noise
    - "Salt and pepper"
  - This will cause an edge
  - Solution: smooth the image by computing a local average

Local averaging

- The obvious way to smooth the image is by computing a local average,
  - Such as

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

- The problem with this is that it introduces “ringing”
  - What happens with our 1 noisy pixel example?

Weighted averaging

- The solution is to make the weights decay with distance from the center pixel
- The principled way to do this is to use a gaussian (normal) distribution
  - Standard $\exp(-x^2)$ weighting
- The process of computed a weighted average (i.e., applying a stencil to an image) is called *convolution*

Convolution

- Convolution has many important properties
  - Extensively covered in signal processing courses
- A major property involves *linear shift invariant systems* (LSI)
  - Linear: F(a*I) = a*F(I)
    - a is a constant (scalar), I an image
  - Shift invariant: F(shift(I,dx,dy)) = shift(F(I),dx,dy)

Convolution and LSI systems

- LSI systems are “well behaved”
  - Bounded input ==> bounded output
  - Shifting the input just shifts the output
- One famous operation is not LSI!
- Convolution with any stencil is an LSI system (operation)
  - The window is local, thus shift invariant
  - We multiply local intensities by the values in the stencil, which is linear

Convolution and LSI systems, continued

- More surprisingly, any LSI operation can be represented as a convolution!
  - To see why this is, consider the effect of an LSI operation on an image that is 1 at a single pixel in the middle, 0 elsewhere
    - Where have we seen these images before?
    - What are they called?

Deriving the stencil of an LSI system

- An image consists of shifted and scaled pixels
  - Doing the LSI operation to an image is the same as doing it to a single pixel, and shifting and scaling the result
    - Because the operation is LSI!
  - Therefore, any LSI system is convolution with a stencil
    - Stencil obtained by the systems response to a pixel
    - Sometimes called “unit impulse response”

Convolution and the frequency domain

- Theorem: convolution in the spatial domain is the same as multiplication in the frequency domain, and vice-versa
  - Convolving an image with a stencil is the same as multiplying the image's Fourier transform by the stencil's Fourier transform, and then doing an inverse Fourier transform
  - Odd consequence: if you want to multiply two signals, you can do it via convolution!

Effect of convolution

- An image can be represented in the frequency domain in terms of a combination of lower and higher frequencies
  - It's the high frequencies that we want to get rid off, because they cause spurious edges
  - The solution is to convolve with a stencil that cuts off the high-frequency components
  - This is called *low-pass filtering* (compare high-pass and bandpass)