CS559
Fall 2003

news
basics
calendar
assignments

project 1
project 2
project 3

C++hints
FlTk
LibTarga
zoomer
Gl Survival

Reader
Curves

Question 1A:

Part of the problem with this question is that we didn't discuss the details of convolution with non-symmetric kernels in class. As some of you noticed (and the updated notes now say), the kernel gets flipped around in time when you do a convolution.

Usually, kernels are symmetric (they are the same as they are flipped around), so you normally don't need to worry about it. Since I only did symmetric kernels in class (and forgot to tell you about the time flip), I would not be suprised if you didn't know it for this homework. Beware of this for the future. See the updated signal processing notes.

If you do reverse the kernel, then there is no answer to this deconvolution! If f(t) begins with [0 1], then convolving that with [-1 1] is going to have a negative number in it for the first non-zero entry. Even I made the mistake of not reversing the kernel.

If you don't reverse the kernel, then the answer is:
f(t) = [0 1 3 6 8 9 9 10 12 15 17 18 18]
(or basically f(t) = sum from 0 to t of h(t))

The kernel [1 -1] basically takes the discrete derivative.

Question 2:

2A:

[1 4 6 4 1]

(You should have done [1 1] * [1 1] = [1 2 1], [1 2 1] * [1 1] = [1 3 3 1], ...)

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

You get this same box whether you convolve the 5x1 kernel with the 1x5 kernel, or the 4x4 kernel with the 2x2 kernel. The 1d convolutions are a lot less work!

2B:

An ideal low-pass filter would chop off all of the high frequencies the first time through. If you apply the filter again, there are no more high frequencies to chop off, so it would do nothing on the second time through.

It's easiest to see this by looking at it in the frequency domain.

2C:

Suppose we have an ideal low pass filter with kernel g, and a signal f. What 2B tell us that: f * g = f * g *g = f * g * g * g (since the first time "through" the filter does all of the high-frequency removal, all the other applications don't do anything)

This means that if g is an ideal low pass filter, g = g * g = g * g * g.

If you have a finite size kernel of width bigger than one, each time you do the convolution, the width gets bigger. If the kernel has infinite extent, then it can't get any bigger (since its already infinitely big).

Question 3

Any signal with a sharp edge (a high frequency) will give a different result when convolved with a real low-pass filter and with a binomial or gaussian.

When applied to a sharp edge, an ideal LPF will create infinite ripples (this phenomenon is called "ringing"), whereas a gaussian or binomial will not create ripples.

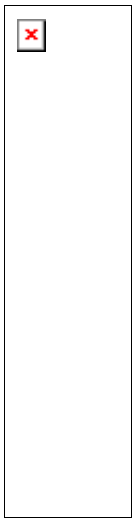
So, for example, if your image was a sharp dot, a Gaussian or Binomial would create a soft-edged dot, while the ideal LPF would create something that looked like ripples in a pond (imagine dropping a drop of water in the middle of a still bowl of water).

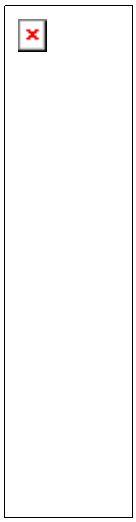
The ringing phenomenon with ripples extending infinitely far away from the dot that caused them, is not good for practical image processing.

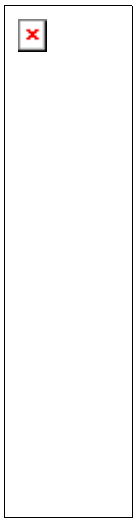
Question 4:

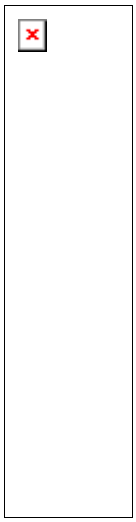
A gradual ramp from black to white would give different results for each method.

Threshold: would have 2 solid areas, half black, half white.
Ordered Dither: would have a set of "bands" that appeared to be the same level (with the same gray pattern in them)
Error Diffusion: would have a gradual fade


gray ramp


threshold


ordered dither


error diffusion

Question 5:

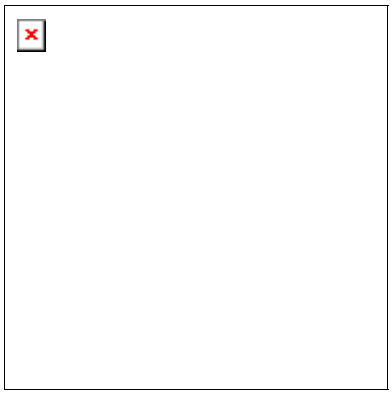
If you filter too little, then you will get aliasing - small patterns that you should not see will show up.

If you filter too much, you will lose detail.

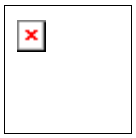
One way to check this is to create some detail that you should not be able to see, and to make sure it gets filtered. So, for example, If you have a pattern of narrow lines, 3 pixels apart, they should look the same no matter how they are shifted - when you "third" an image, you shouldn't see the lines if they are that close. If the pixels are 4 or 5 pixels apart, then you should start to see their pattern.

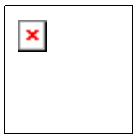
Here's a sample test pattern.

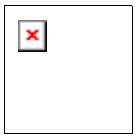
It has thin lines spaced 2, 3, 4, and 5 pixels apart. They are shifted to make "diagonal" lines.

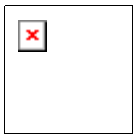


This image is 192x192, so I shrank it to 64x64. In each case, I first filtered it using a Binomial filter, and then used point sampling (nearest neighbor in Photoshop). Ideally, we would see no pattern for the width 3 spacing, and see some pattern for the width 4 spacing. However, getting the exactly right kernel isn't possible with binomial filters.


no filtering


1 2 1 binomial


(1 2 1)^2 binomial


(1 2 1)^3 binomial

I should also note (and this is an "advanced' observation): the Binomial filters are square in shape (you would expect a better filter to be round, or more radially symmetric). This created artifacts with the diagonal lines, which is part of the reason this example doesn't work out as nicely as one would hope.