

Assignment 3 Edge Detection

19th October 2006

1 Introduction

In this assignment, you will be familiar with the concept of linear filter, convolution and FFT. And You need to implement edge detection in matlab. Part 2 is due on Tue,Oct 24. Part 3 is due on Tue,Oct 31.

2 Warm up for 1D

$w = conv(u, v)$ convolves vectors u and v . Algebraically, convolution is the same operation as multiplying the polynomials whose coefficients are the elements of u and v .

Let $m = length(u)$ and $n = length(v)$. Then w is the vector of length $m + n - 1$ whose k^{th} element is

$$w(k) = \sum_j u(j)v(k + 1 - j)$$

In matlab, 1D convolution operation is simple by calling $conv(u, v)$. For example the input 1D signal is $u = [1, 1, 8, 1, 1]$, and the filter kernel is $v = [-1, 1]$, the results of $conv(u, v)$ will be $[-1, 0, -7, 7, 0, 1]$ with the length of $5 + 2 - 1 = 6$.

On the other hand, the Fourier transform of the convolution of two functions is the product of their Fourier transforms.

$$F[g * h] = F[g]F[h]$$

Thus instead of doing convolution between image g and filter kernel h by $g * h$, we have

$$g * h = F^{-1}(F[g]F[h])$$

Which means whenever convolving the image with an filter, we could first do Fourier transform to both of them, multiply them piece-wisely and then do inverse Fourier transform.

$$g * h = F^{-1}(F[g]F[h])$$

Take the same u, v above as the example. Considering the length of the result of the convolution has length $u + v - 1$, we do zero-padding before Fourier transform. We have

$$F(u) = \text{fft}(u, \text{zeros}(1, \text{length}(v) - 1))$$

$$F(v) = \text{fft}(v, \text{zeros}(1, \text{length}(u) - 1))$$

And then, we do inverse Fourier transform, what it will be?

$$\text{ifft}(F(u) .* F(v)) = [-1, 0, -7, 7, 0, 1]$$

It's the same as $\text{conv}(u, v)$.

2.1 Deliverables

Input 1D signal of your choice, and 1D simple filter, show the steps above to prove that

$$\text{conv}(g, h) = \text{ifft}(\text{fft}(g)\text{fft}(h))$$

3 Edge detection for 2D image

Please download the images lena.jpg online. You will implement simple edge detection with sobel operator.

3.1 Convolution vs FFT

The Sobel operator performs a 2-D spatial gradient measurement on an image. Typically it is used to find the approximate absolute gradient magnitude at each point in an input grayscale image. The sobel edge detector uses a pair of 3×3 convolution masks, one estimating the gradient in the x-direction(columns) and the other estimating the gradient magnitude in the y-direction(rows). A convolution mask is usually much smaller than the actual image. As a result, the mask is slid over the image, manipulation a square of pixels at a time. The actual So-

bel mask for x-direction and y-direction are

-1	0	+1	+1	+2	+1
-2	0	+2	and	0	0
-1	0	+1		-1	-2

respectively.

For example, if you just convolve the image with Sobel mask for x-direction, you will get the edge which is stronger in x-direction.

1. Convert the source color image C to grayscale image A .

2. Smooth the image A with Gaussian filter to get image S . This will remove the high frequency noise to facilitate the edge detection. To generate a Gaussian filter, you could use matlab command *fspecial*.
3. Convolute the image S with sobel mask for x-direction and you get image G_x . For 2D convolution, you could use matlab command *conv2(A, filter, 'same')*.
4. Instead of using *conv2* in previous step, you could use Fourier transform which we mentioned earlier to achieve the same goal. The related code for 2D fft will be

$$F(g) = fft2(g)$$

$$F(h) = fft2(h, size(g, 1), size(g, 2))$$

Which also adds zero-padding. And *ifft2(F(g) .* F(h))* will lead to the same result as *conv2*. When the image or the filter is really big, then FFT will speed up the image process.

3.1.1 Deliverables

The deliverables are the image of *conv2* in step 3 and the image you got from step 4.

3.2 The whole pass for edge detection

For a simple edge detection, you need to follow the following steps:

1. Convert the source color image C to grayscale image A .
2. Convolute the image S with sobel mask for x-direction and you get image G_x . For 2D convolution, you could use matlab command *conv2*.
3. Convolute the image S with sobel mask for y-direction and you get image G_y .
4. Compute the gradient magnitude image G using

$$|G| = \sqrt{G_x^2 + G_y^2}$$

5. Threshold the gradient magnitude image G so that the pixel with the value larger than the threshold is kept. The output is edge image E .
6. Blend the original color image C with the edge image E to get a cartoony image I . The easiest way is to set the the pixel of C to be black when its corresponding position in E is nonzero. You could also blur image C or set it with some transparency before blending, which will be more interesting.

3.2.1 Deliverables

The first three steps are the same as in part 3.1. The deliverables are the results of image S, G_x, G_y, E , and the final cartoony image I .

4 Applying the edge detection for a video

This is only for fun! You could use Quartz Composer in Mac or some other software to decompose a video to image frames, apply the edge detector and compose them to video. Then you 'll get a cartoony video! Show me the result if you'd like.