Final Project #2: Convolutional Image Filters

Final Project
Section E

Jonathon Schnell

Overview

Part two of the final project asks to write a couple image filter algorithms all of the algorithms involve using a kernel convolution. The first filter is a blur effect that asks the user for a radius of blur witch is the size of the kernel being iterated. The second filter was the sharpen filter. This is ment to work in a similar way but instead of averaging the pixel values will will make them more unique to each other. The third image filter I was asked to create was an edge detection filter. Edge detection filters will highlight areas of high contrast in the image. These areas are shown as white or black depending on the gradient of the contract change of the image. Convolution is the process of applying a 2d array to the kernel space of the image then processing and applying changes to the image.

Analysis

For the blur filter I used 4 nested 'for loops'. Two if these loops are to iterate across the whole image. The other two are to iterate across the kernel of the image. The kernel must be able to go to any size based on the selected radius and handle edge cases. In these two for loops we iterate from negative radius to radius. Then we average all the collected values of red blue and green by summating and dividing by a count. Finally we output our averaged image values to create a blur effect of radius r.

The sharpen filter was very similar to the blur filter. In the blur case we were Convoluting a three by three matrix entirely of 1s. To sharpen the image we need to change our kernel values to make the pixels near each other less similar. The kernel for sharpen looks like k={{0, -1, 0}{-1, 5, -1}{0, -1, 0}}. We can iterate this again with four nested for loops but now we want to iterate from 3 to 0 on the inner two loops so that we can convolute by multiplying kernel values with image values collecting sums averaging over a count and applying to the final image.

The edge detection was a little more complex as it requires 2 more inner for loops. This is because the sobel operator will be used in the x direction and the y direction looking for high contrast edges in the image. From there we collect values for sobel in the x direction and sobel in the y direction. By using a formula similar to pythagorem's theorem we can get our output values the apply them to the image. While the edge detection algorithm works best on black and white images it can still be used on color images if you specific edge cases.

Design

The code is broken into 3 files, a header file, a main file, and a algorithm file. The advantage to breaking your code into multiple files is that if you want to block the user from being able to read the source code. This is handy if you have a specialty algorithm that you want to hide but still need to allow a customer access to the code. What we do is compile the algorithm file into an object file. We do the same for the main file. Once this is done the source code cannot be read off of these two files but the customer can still execute the program. Another advantage to breaking your program into small chunks is to make it easier to read and understand while debugging.

Testing

This program required quite a bit of testing. One major mistake I made was not resetting the variables used to collect summs after the inner two loops. It is important that these variables and the count are reset after or before the inner two loops iterate. Initially I reset these values before the first for loops. This obviously did not work because values were only reset to zero after the entire image is processed. The next issue was dealing with the border on an image. This is needed because we don't want the kernel trying to read values that are not on the image. In the case of the blur filter we can just set the bounds of the outer two for loops to iterate from x + radius of blur to image size - radius of blur. This will mean that our kernel never reads values that are not on the image. Another edge case is if the colors exceed 255 or are less than 0. To solve this issue we can have a while loop that only allows values on a range of 0 to 255. We can execute our image processing algorithm inside this while loop.

Comment

In my opinion this lab was challenging and enjoyable. It is very rewarding to see the final image output work as intended. This lab really shows off the practical applications of c programming. Before this lab I had never heard of kernel convolution or the sobel operator. But now I have a good understand of the mathematical concepts as well as the practical applications for kernel convolution and sobel operator. With that being said this lab was still quite a challenge, it took me about a week working for 2-4 hours a day to get all three image filters working as intended. There are many online sources to reference but many are in c sharp or not intended for bitmap images. Overall running the program for the final time was very rewarding to see the algorithms working as intended.