CS1022 - ASSIGNMENT 1

CS1022

'Image Manipulation' by Brandon Dooley

Throughout the following report I will document the various stages of the development of my ARM Assembly Language programs for Assignment 1. I will explore various tests that were made throughout the stages of development and difficulties that I encountered respectively.

CS1022 - Assignment 1

'IMAGE MANIPULATION' BY BRANDON DOOLEY

PART ONE | Brightness and Contrast

"Design write and test an ARM Assembly Language subroutine that will adjust the brightness and contrast of the TCD crest image stored in memory."

This program begins by reading the dimensions of the image in terms of pixel rows and columns. It then takes user defined adjustment values of *B* and C for the brightness and contrast respectively. Following this it iterates through each pixel in the image and uses the following formulae to manipulate the Red, Green and Blue components of each pixel.

$$\text{PixelRedComponent[i][j]} = \left(\frac{PixelRedComponent * C}{16}\right) + B$$

$$\text{PixelGreenComponent[i][j]} = \left(\frac{PixelGreenComponent * C}{16}\right) + B$$

$$\text{PixelBlueComponent[i][j]} = \left(\frac{PixelBlueComponent * C}{16}\right) + B$$

The program first checks that each RGB components new value does not exceed 255. If it does the value is set to 255. The new RGB value of each pixel is then stored in memory at the current address of each pixel.

Extra Credit: Upon further analysis of the program, it came to my attention that the majority of the image is made up of pixels with the same RGB values. To prevent computational time being wasted on calculating the new RGB value of pixels with the same initial value, the program compares the current pixel with the next pixel.

If RGB(nextPixel) = RGB(currentPixel) the nextPixel is also replaced with the same adjusted RGB value that was found for the current pixel.

PART ONE | Testing

Brightness Val= 10

Contrast Val= 6

Original Image

Adjusted Image

Brightness Val= 12

Contrast Val= 40

Original Image

Adjusted Image

Brightness Val= 19

Contrast Val= 20

Original Image

Adjusted Image

Brightness Val= 40

Contrast Val= 12

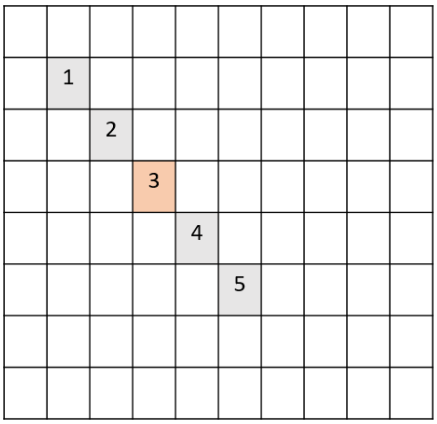
Original Image

Adjusted Image

PART TWO | Motion Blur

"Design write and test an ARM Assembly Language subroutine that will apply a motion blur effect parallel to a diagonal line from the top-left to the bottom-right of the image."

This program begins by taking the *Blur Length* value, defined by the user. For each pixel in the image it then takes the average red, green and blue component from the pixels diagonally above and below it within the *Blur Length*. It then stores this result in the memory address of the current pixel.



In the diagram above the *Blur Length* is set to 5 pixels. The program takes the average of the *Red, Green and Blue* components of the pixels 1 through 5. It then replaces the current RGB value of pixel 3 with the average found.

$$\text{PixelRedComponent}[3] = \left(\frac{Red(1)+Red(2)+Red(3)+Red(4)+Red(5)}{Blur\ Length} \right)$$

$$\text{PixelGreenComponent}[3] = \left(\frac{Green(1)+Green(2)+Green(3)+Green(4)+Green(5)}{Blur\ Length} \right)$$

$$\text{PixelBlueComponent}[3] = \left(\frac{Blue(1)+Blue(2)+Blue(3)+Blue(4)+Blue(5)}{Blur\ Radius} \right)$$

Extra Credit: To prevent previously found averages being used in the calculation of new average values for pixels, the resultant image is stored at a different address to the

original image. When the *Blur Value* has been calculated for each pixel the original image is then overwritten by the new blurred image.

PART TWO| Testing

Blur Length = 3

Original Image

Adjusted Image

Blur Length = 5

Original Image

Adjusted Image

Blur Length = 9

Original Image

Adjusted Image

Blur Length = 12

Original Image

Adjusted Image

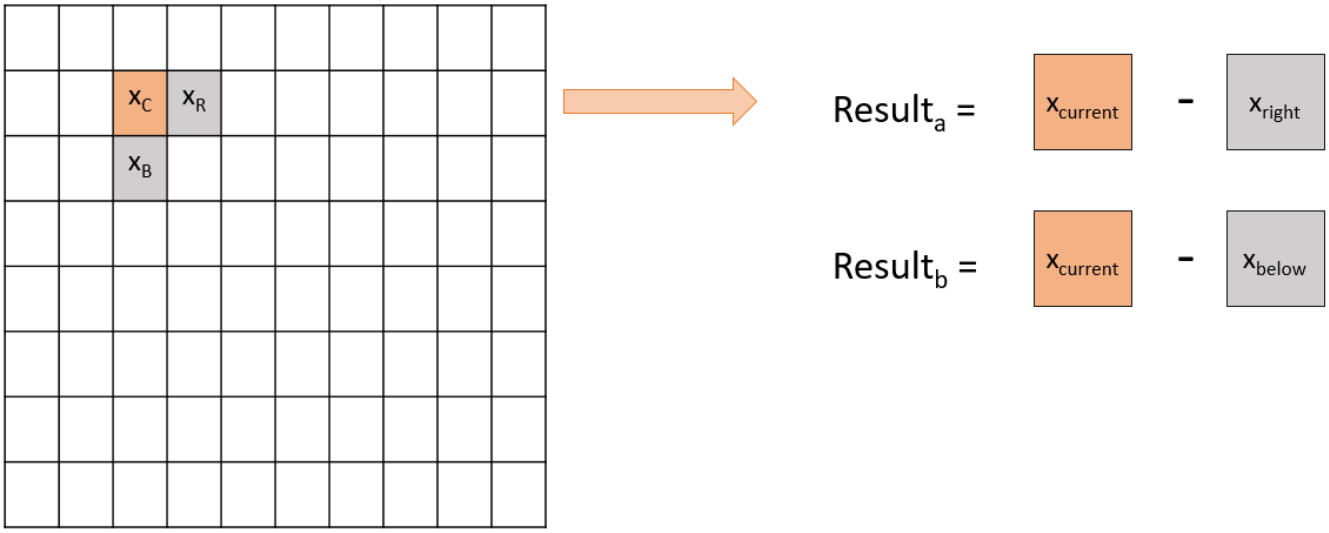
PART THREE | Edge Detection

"Design, write and test a subroutine that will apply an effect of your choice to the TCD crest image stored in memory."

Chosen Effect: *Edge Detection*

This program begins by taking a user defined *Threshold Value*. This is used to determine the threshold to be used when detecting edges within the image.

The program then iterates through each pixel one at a time. To calculate whether the current pixel is an edge or not it compares it to the pixel to the right of it and the pixel below it. It uses the following formula to determine whether a pixel is an edge or not.



If($Result_a$ > *Threshold Value* || $Result_b$ > *Threshold Value*)

If the statement above is *True*, the pixel is therefore considered to be an edge and the pixel is therefore set to a white color. However, if the statement returns *False* the pixel is considered to not be an edge and is set to a black color.

PART THREE | Testing

Threshold = 4

Original Image

Adjusted Image

Threshold = 20

Original Image

Adjusted Image

Threshold= 64

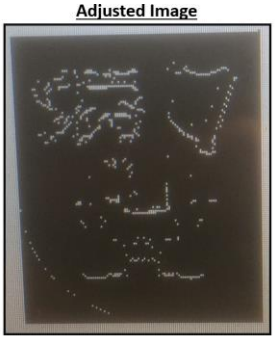
Original Image

Adjusted Image

Threshold = 71

Original Image

Adjusted Image



ANALYSIS| Overall Program

I feel that the programs on an overall level serve their individual purposes both efficiently and effectively. It allows the user to perform three different image manipulation techniques, whilst allowing them to change the strength of each technique and the various other variables respectively.

I felt the assignment itself was challenging yet rewarding. I feel I have gained immense knowledge of the assembly language itself and learned how to use subroutines to divide a problem on a larger scale to much simpler, smaller problems. The use of subroutines combined with the system stack further allowed me to make much greater use of the limited registers at my disposal.

Brandon Dooley (#16327446)

CS1022 Assignment 1 – Image Manipulation

Submitted 20/03/2017