**CS1022: Assignment #1**

My report on the second computing assignment is as follows:

The assignment was split into three parts: 1.1: Brightness and Contrast, 1.2: Blur effect, 1.3: Bonus effect.

1.1: Brightness and Contrast:

This part involved processing the TCD crest image and adjusting the brightness and contrast.

I had to take pixels from an array, separating them into red/green/blue components and performing a brightness and contrast formula and store the pixel back in.

The array looked like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 |

I used a nested loop to go through the array of pixels, going through each row and each then each column in the row. I created a counter for the columns and rows, incrementing once each row or columns pixels were operated on.

row

CMP R11, R5 ;if(row!=colHeight)

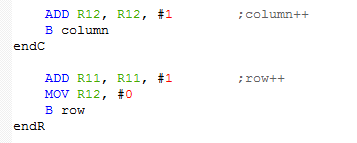
BEQ endRow ;{

column

CMP R12, R6 ;if(col!=rowWidth)

BEQ endColumn ;{

This is how the loop branched back:

I reset the column counter when it moved to the next row.(MOV R12, #0)

To extract a specific colour from a pixel I had to use logical AND with a specific buffer location for each colour.

This code takes the significant bits for the green component and shifts it right so you can operate on it.

I used #0x00FF0000 as a buffer for blue.

I used #0x0000FF00 as a buffer for green.

I used #0x000000FF as a buffer for red.

In order to operate on the code I logically shifted blue and green.(LSR#16 and LSR#8 respectively

My formula for adjusting brightness and contrast was as follows:

P’ = (P\*A/16)+B

P’=new pixel

P=old pixel

A= contrast (default=16)

B=brightness

In the formula I had to divide by 16 so I used this subroutine to make my code more readable and to follow syntax.

div ; while

CMP R1, #16 ;(remainder >= b )

BLO finDiv ; {

ADD R2, R2, #1 ; quotient = quotient + 1 ;

SUB R1, R1, #16 ; remainder = remainder - b ;

B div ; }

I adjusted my red green and blue values for overflow and exceeding the 255 limit.

If the register value was above 0xF0000000 I assumed it overflowed and set the value to 0, which is the min.

After that if the register was still more than 255 I set it to 255, which is the max.

I repeated this for all 3 colour values.

I logically shift blue and green(LSL#16, LSL#8) to return to its original positions.

The updated image gets re-displayed after all the operations have been performed.

TESTING:

|  |  |
| --- | --- |
| The brightness was turned up slightly and the contrast was left untouched. |  |
| The image was very bright in this case and the contrast was untouched. |  |
| The contrast was slightly turned up in this case. |  |
| The contrast was turned up very high and the image was intensely faded. |  |
| The contrast and brightness was increased. |  |

All the results were like expected and when I completed the assignment I had no more errors.

While working on this part I encountered numerous errors in relation to overflow, yellow tint and purple tint.

In the case of overflow I fixed the issue by setting the r/g/b to min or max values if it surpassed it at either spectrum.

The yellow tint was a problem with the blue buffer.

The purple tint was due to the blue being set to max at all times.

I resolved all the problems along the way and I’m certain that my code is working exemplary.

1.2: Blur effect

I am not pleased with my blur effect. I wasn’t able to achieve a satisfiable working answer.

My attempt goes as follows:

I still went through every row and every column through a nested loop, incrementing a counter each time and ending once the column height was hit.

I had to take pixels from an array, separating them into red/green/blue components.

I achieved this by using logical AND to buffer each specific colour into a register.

I used #0x00FF0000 as a buffer for blue.

I used #0x0000FF00 as a buffer for green.

I used #0x000000FF as a buffer for red.

I tried to save the pixels to the stack using the stack pointer and retrieve an old copy at each iteration. This would mix the different pixels around and in turn make the picture blurry.

LDR R1, =5

I set the radius to be equal 5.

1.3: Bonus effect.

For the bonus effect I decided to invert the image colours of the TCD crest.

I again went through every row and every column through a nested loop, incrementing a counter each time and ending once the column height was hit.

I used a nested loop again to go through the array of pixels, going through each row and each then each column in the row. I created a counter for the columns and rows, incrementing once each row or columns pixels were operated on.

row

CMP R11, R5 ;if(row!=colHeight)

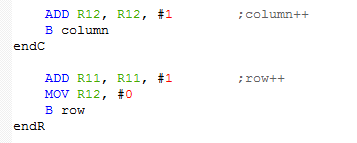
BEQ endRow ;{

column

CMP R12, R6 ;if(col!=rowWidth)

BEQ endColumn ;{

This is how the loop branched back:



I had to take pixels from an array, separating them into red/green/blue components and manipulate them to invert and store the pixel back in.

To extract a specific colour from a pixel I had to use logical AND with a specific buffer location for each colour.

This code takes the significant bits for the green component and shifts it right so you can operate on it.

I used #0x00FF0000 as a buffer for blue.

I used #0x0000FF00 as a buffer for green.

I used #0x000000FF as a buffer for red.

In order to operate on the code I logically shifted blue and green.(LSR#16 and LSR#8 respectively).

After I subtracted each colour from 255. This inverted the red, green and blue.

I adjusted my red green and blue values for overflow and exceeding the 255 limit.

If the register value was above 0xF0000000 I assumed it overflowed and set the value to 0, which is the min.

After that if the register was still more than 255 I set it to 255, which is the max.

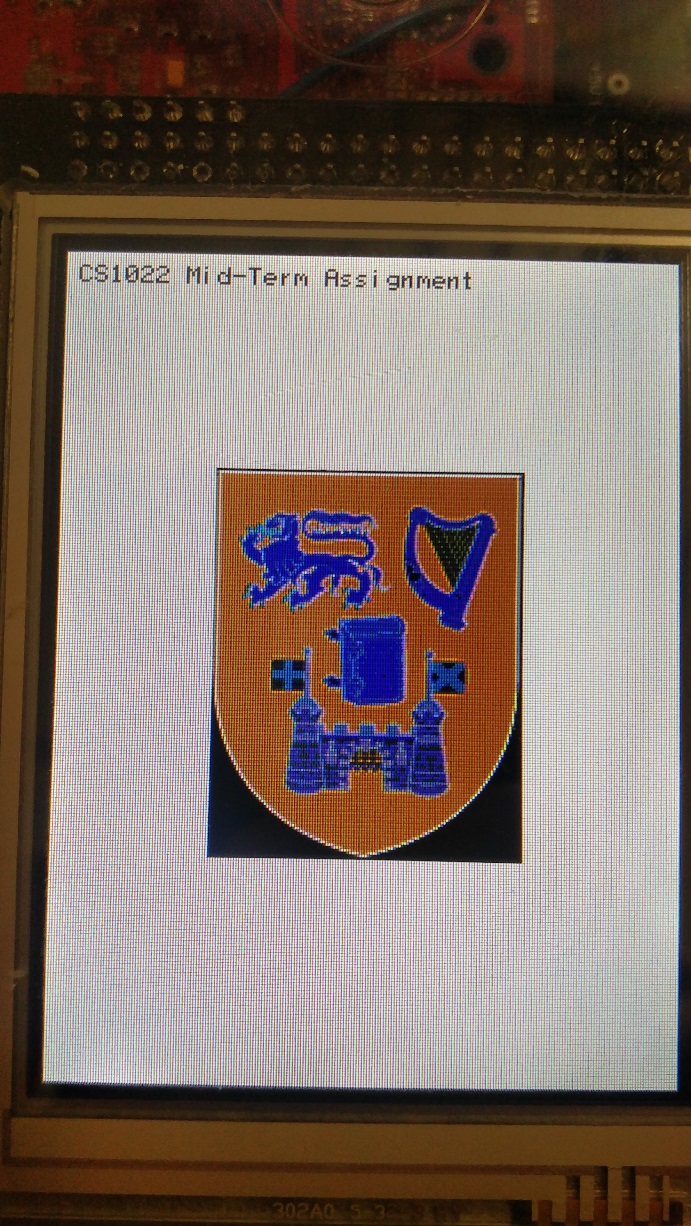
I repeated this for all 3 colour values.

I logically shift blue and green(LSL#16, LSL#8) to return to its original positions.

The red, green and blue now in their original positions.

The updated image gets re-displayed after all the operations have been performed.

TESTING:



The program worked as intended.

The bright blue was inverted to orange.

The yellow to dark blue.

The white to black.