# Assignment Part #3

This report describes the development of an ARM Assembly Language program for manipulating an image by applying different effects.

The report is broken into three stages:

(i) Adjust

(ii) Motion blur

(iii)Bonus effect

## 1. ADJUST

This section describes the approach taken for adjusting brightness and contrast to the TCD crest image stored in memory. The concept behind the Adjust program was to access each pixel in the image and adjust its brightness and contrast, sending the adjusted pixel value from subroutine back to the main to be stored in R4(pic address).

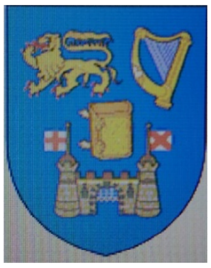
Throughout the whole program I used various subroutines for problem decomposition, the idea with was to have a neat and self-explanatory subroutines.

The subroutines were:

| | |
|---|---|
| getRed | gets the individual red value from the target pixel |
| getGreen | gets the individual green value from the target pixel |
| getBlue | gets the individual blue value from the target pixel |
| updatePixel | takes in an individual color value and adjusts the contrast and brightness. |
| recombineColors | takes in all the individual updated color values and shifts the colors values accordingly into their original positions. ie shifting red by 16 bits since red is the MSB (most significant byte) |

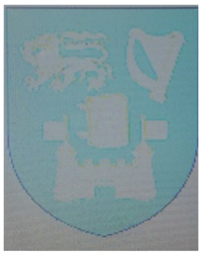
When adjusting the brightness and contrast, error checking was important to avoid overflow of the values. If the updated color value is greater than 255, set the updated color value to 255. Else if updated color value is less than 0, set the updated color value to 0. When loading the individual color value, the color value should be shifted to LSB to perform operations, when storing the updated color value, shift the color values to their corresponding original positions.

The picture shows the adjusted TCD crest with the following values:



Brightness=20

Contrast=30



brightness=200

Contrast=20

The following pseudo code is the general approach taken for the adjust program.

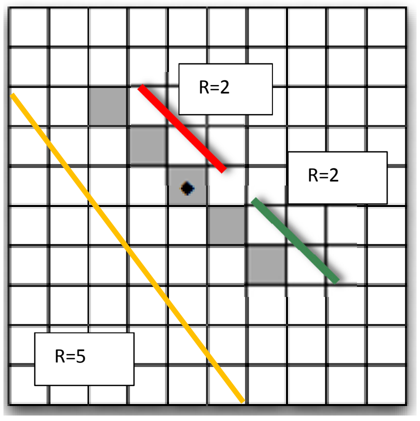
```
Color[][]adjust
for (int i=0;i<adjust.length-1;i++ )
        for(j=0;j<adjust.length-1;j++)
                int index = row*row_size + column
                int pixelValue=Memmory.word[pic adr.+ index]
                int updatedPixel = ((pixelValue*contrast)/16) + brightness
                        if(updatedPixel>255)
                        {
                                updatedPixel=255
                        }
                        else if(updatedPixel<0)
                        {
                                updatedPixel=0
                        }

                        Memmory.word[pic adr. + index] = updatedPixel
```

## 2. Motion Blur

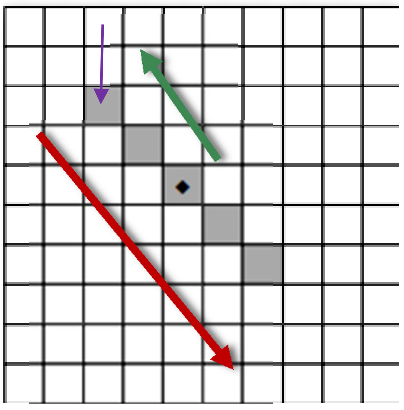
This section describes the approach taken for applying motion blur to the image. Motion blur has a

There was a slight alteration as to how I defined the radius, in the documentation the radius was defined as the TOTAL number of pixels to average. However, I defined it as the distance from the number of pixels from the target pixel. For example if radius was 2, that would mean 2 pixels to the left of the target pixel and 2 pixels to the right of the target pixel. Hence the total number of pixels would be 5.

Here is an image representation of how radius was defined, R is radius.



The diagram below represents how I obtained the diagonal pixel values.



The approach taken here was to create a copy array of R4 (picture address) into some other register which then we would access the array as a two dimensional array by having two for loops one for row and another for column.

The approach taken here was to pass the row and column of the current target pixel and also the radius to total Pixel subroutine.

In the total Pixel subroutine:

Keep subtracting row and column from the original row and column until the leftmost row and column is reached. When the left extreme boundary is reached, each pixel value is obtained by increasing row and column by 1 until the total number of pixels is reached. By increasing row and column by 1, pixels are obtained diagonally.

The radius was parameterized hence the radius could have been easily changed by only changing the value of the radius once in the main method and this would have been taken into account in the other subroutines. I also programmed so the radius would increase automatically over time, giving an increased blurring effect over time. For demonstration purposes I placed the BL put Pic after all the pixels were updated for a given radius, I could have easily placed put Pic subroutine after each row was updated to have a smoother blurring transition.

Below is a general pseudo-code for the approach of motion blur effect.

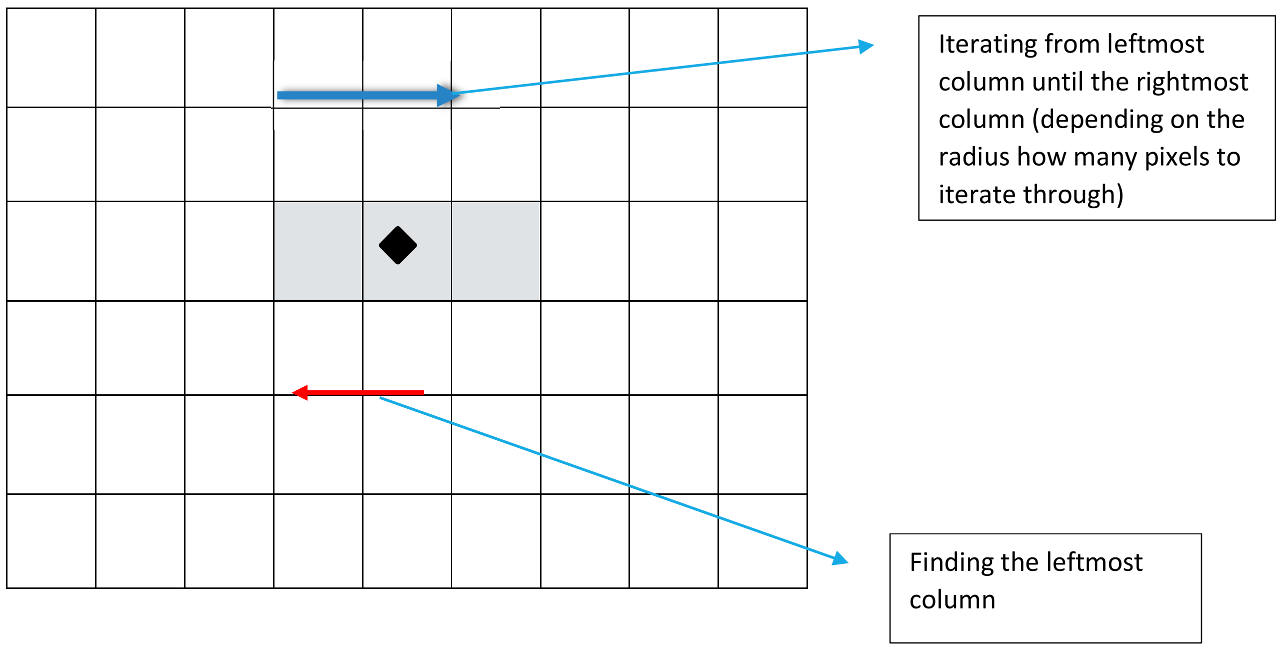
```
Color motionBlur[][] = image.clone()
            for (int row=0;row<motion.length;row++)
                  for(int col.=0;col<motionBlur.length;col.++)
                        int counter=0
                        index=(row*motionBlur.length) + col.
            Store original row and column onto system stack
//get leftmost pixel
                        while(counter<radius)
                        {
                        column--
                        row--
                        counter++
                        }
//while not reached end of desired distance continue to get diagonal pixels and add to total
                  Store row and column onto system stack
                  int otherCounter=0
                        while (otherCounter<((radius*2))+1)
                        {
                              Total red = total red + image[row][col].getRed()
                              Total green = total green + image[row][col].getgreen()
                              Total blue = total blue + image[row][col].getblue()
                              row++
                              column++
                              othercounter++
                        }
                  totalColors.average()
                  load original row and column of target pixel
                  blurred image = new Color(average red, avg.green,average blue)
                  Memmory.Word[pic.adr + index] = blurred image.
```

## 3. Bonus Effect

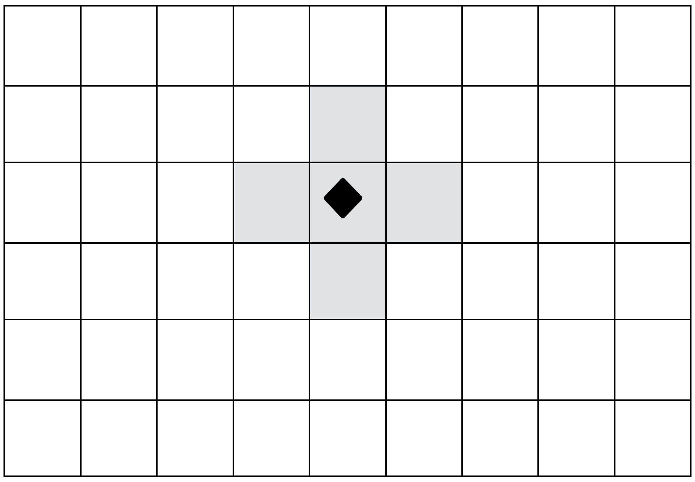
This section describes the approach taken for the Bonus effect. For bonus effect I chose simple blur, which has a stationary blurring effect on the image.

The approach taken here was to create a copy array of R4 (picture address) into some other register which then we would access the array as a two dimensional array by having two for loops one for row and another for column.
The idea was to pass the row and column to subroutine which would total the horizontal pixel values by increasing column by one each time, top pixel value by decreasing row and bottom pixel value by increasing row.
As each word size pixel value was obtained, I performed bit clear operation to obtain RGB colour separately and having a separate total colour value for each colour, and adding respective colours into their respective total colour register for all the pixel values. Once the required pixel values were taken into account, I would average each total colour value by the number of pixels.
For example: Average red = total red / number of pixels
Once the average of all the RGB colour was calculated, the colour was recombined by shifting and OR'ing the updated colour values and then beingstored into R4 (picture address).

The diagram below shows how I obtained the horizontal pixels

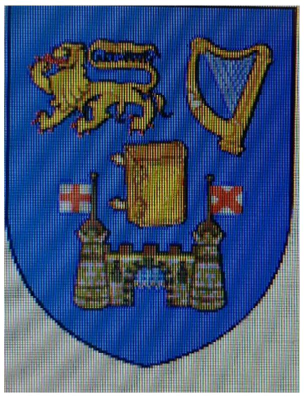

Iterating from leftmost column until the rightmost column (depending on the radius how many pixels to iterate through)
Finding the leftmost column

The diagram shows the pixels that were taken into account for blurring of the image, as you can see I am taking into account the surrounding pixels and replacing the target pixels's value with the average of all the grey boxeed pixels. This would give a blurring effect on the crest.



Below are the original and blurred image for bonus effect. The effect of the blur could be increased by changing the values of radius.

original image



blurred image

The following pseudo-code is the general outline for the approach of bonus effect.

```
Color simpleBlur[][] = image.clone()
            for (int row=0;row<simpleBlur.length;row++)
                  for(int col.=0;col<simpleBlur.length;col.++)
                        int counter=0
                        index=(row*simpleBlur.length) + col.


                  Store original row and column onto system stack
            //get leftmost column
                  while(counter<radius){
                  column--
                  counter++
                  }
            //while not reached end of desired distance get horizontal pixels and total
                  Store row and column onto system stack
                  int othercounter=0
                  while(othercounter<((radius*2))+1){
                  Total red = total red + image[row][col].getRed()
                  Total green = total green + image[row][col].getgreen()
                  Total blue = total blue + image[row][col].getblue()
                  column++
                  }
                  Restore row and column from system stack
                  Store row and column onto system stack
                  Row++
                  Total red = total red + image[row][col].getRed();
Total green = total green + image[row][col].getGreen()

Total blue = total blue + image[row][col.].getBlue()


Restore row and column form the system stack


//get top pixel

Row - -

Total red = total red + image[row][col].getRed()

Total green = total green + image[row][col].getgreen()

Total blue = total blue + image[row][col].getblue()


//average colors

totalRed.average()

totalGreen.average()

totalBlue.average()


load original row and column of target pixel

blurred image = new Color(total red, total green,total blue)

Memmory.Word[pic.adr + index] = blurred image.
```