Assignment #1
2-Dimensional Arrays – Image Manipulation

20 February 2017

This is a marked assignment and will count towards your final coursework mark for CS1022. It is an individual assignment.

You will be asked to demonstrate your solution during the labs on Friday 10th March 2017. Marks will be awarded for this demonstration.

You must submit your solutions using Blackboard no later than 23:59 on Monday 13th March 2017. Late submissions without a satisfactory explanation will receive zero marks.

Submit your `.s` assembly language source files and your report **in PDF format** as attachments to "Mid-Term Assignment" in Blackboard.

Your `.s` assembly language source files must be suitably commented.

Introduction

In each of the template µVision projects accompanying this assignment, an image of the (old) TCD crest has been stored in memory using an RBG (Red Green Blue) colour model. The image is stored as a two-dimensional array of word-size values. Each word-size value corresponds to a single pixel in the image. The Blue, Green and Red components of a pixel are stored in bytes 0, 1 and 2 of the word-size value respectively (with byte 0 being the least-significant byte and byte 3 being the most significant byte). Byte 3 (the most significant byte) is not used and is always zero.

You are asked to design, write and test three subroutines to manipulate the image stored in memory. In each case, your solution must make use of further subroutines and must demonstrate *problem decomposition*.

A number of other subroutines are provided to assist you and the labels and interfaces for these subroutines are shown in Table 1.

Part 1 – Brightness and Contrast [20%]

The *brightness* of the pixels in an image can be increased (or decreased) in a simple manner by adding a positive (or negative) constant value to the red, green and blue components of each pixel of the image. Similarly, the contrast can be adjusted in a simple manner by multiplying the red, green and blue components of each pixel by a constant value.

The formulae below express how the Red (R), Green (G) and Blue (B) components of a pixel at coordinates i, j might be be modified to change brightness (β) and contrast (α). Note that the

subroutine label	interface

getPicAddr	`; getPicAddr subroutine`
	`; Returns the starting address of an image stored in memory`
	`; Parameters none`
	`; Return values R0: starting address of image`

getPicWidth	`; getPicWidth subroutine`
	`; Returns the width in pixels of an image stored in memory`
	`; Parameters none`
	`; Return values R0: width in pixels`

getPicHeight	`; getPicHeight subroutine`
	`; Returns the height in pixels of an image stored in memory`
	`; Parameters none`
	`; Return values R0: height in pixels`

putPic	`; putPic subroutine`
	`; Displays the image stored in memory on the LCD`
	`; Parameters none`
	`; Return values none`

Table 1: Subroutine interfaces

contrast is decreased if $\alpha < 16$, increased if $\alpha > 16$ and is unchanged if $\alpha = 16$.

$$R'_{i,j} = \left(\frac{R_{i,j} \times \alpha}{16}\right) + \beta$$
$$G'_{i,j} = \left(\frac{G_{i,j} \times \alpha}{16}\right) + \beta$$
$$B'_{i,j} = \left(\frac{B_{i,j} \times \alpha}{16}\right) + \beta$$

Design, write and test an ARM Assembly Language subroutine that will adjust the brightness and contrast of the TCD crest image stored in memory. Take care to handle cases where the adjustment causes the value of the red, green or blue components to exceed 255 (or fall below 0).

Part 2 – Blur Effect [25%]

A "motion blur" effect can be applied to an image in a simple way by replacing the Red, Green and Blue values of each pixel with the average of the corresponding colour channels from the pixels in a line through each pixel. For example, in Figure 1 below, the pixel marked ♦ is replaced with the average colour of the diagonal line of shaded pixels. (Note that the original pixel marked ♦ is also included in the average.)

The "blurriness" of the resulting image is determined by the radius of the effect, r, in pixels. In the example in Figure 1, the radius is 5 pixels. The image in Figure 2 illustrates the effect on the TCD crest.

Alternatively you can try any other type of blur that you want. The only stipulation is that the

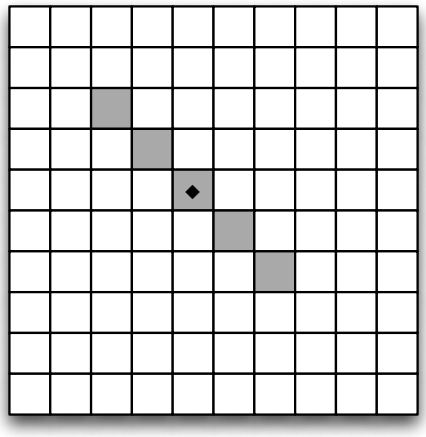

Figure 1: The shaded pixels are used to compute the new value of the pixel marked ⧫.

Figure 2: Motion blur effect applied to the TCD crest.

output value of a pixel must depend on the input of at least 1 pixel other than itself. Some possibilities are Gaussian blur, posterisation , pixelisation.

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 (or other blur effect). Test your effect using different radii.

Part 3 – Bonus Effect [30%]

Design, write and test a subroutine that will apply an effect of your choice to the TCD crest image stored in memory. Try to choose an effect that requires each pixel to be replaced with a value based on other pixels. (The blurring effect above replaces each pixel with a value based on other pixels but the brightness effect above does not.) Marks will be awarded for the quality of your solution and also its sophistication. (More complex effects will attract more marks.) Examples of more complex effects include: (i) sharpening, (ii) another type of blurring, (iii) edge detection, (iv) unsharp masking or (v) generic application of a convolution matrix (vi) lens effect.

Documentation

You are required to document your approach to the development of all three subroutines. Your documentation should be in the form of a typed document **in PDF format**. This document must be submitted with your assembly language `.s` source files.

Your report must contain a detailed description of your approach to developing your solution.

Your description should make use of examples, diagrams and pseudo-code (properly formatted) where appropriate. You must provide a description of the interface for each subroutine that you write. Avoid writing a narrative that simply describes what each instruction in your program does. Instead, try to describe how your program works at a higher level. Concentrate on the approach that you have taken, not the minute details of the assembly language.

The nature of this assignment means that your subroutines cannot be tested with different input values (in most cases). You should instead simply insert a photograph of the output displayed on the LCD screen for each subroutine and explain whether the image was as expected and, if not, try to explain this.

Note that marks will be awarded for both the content and presentation of your document and will account for 25% of the overall marks available for the assignment.