CSC 314 Programming Assignment 1

Fall, 2016

Due November 4

Overview

Your programming assignment is to enhance a program that can do image processing using Netpbm data files. There are six types of Netpbm file, but we will only use four of the six types: `.ppm` binary, `.ppm` ASCII, `.pgm` binary, and `.pgm` ASCII. The `.ppm` extension stands for Portable PixMap. Those files are color images. Files with a `.pgm` extension are grayscale (Portable GreyMap).

The program has already been written, and is used to process large numbers of image files, but it runs rather slowly. Your task is to convert the image processing routines into ARM assembly language and make them as fast as possible. You should be able to get sigificantly better performance overall. The following sections describe the original program, which you must modify.

PPM Image Types

There is an ASCII header in every Netpbm image. Files contain either ascii or binary data for the image, and the image may be color, grayscale, or bitmap. The first two bytes of the file contain a *magic number*, which is two ASCII characters. There are six possible magic numbers for a Netpbm image, as follows:

P1 indicates an ASCII bitmap image,

P2 indicates an ASCII greymap image,

P3 indicates an ASCII color image,

P4 indicates a binary bitmap image,

P5 indicates a binary greymap image, and

P6 indicates a binary color image.

The magic number must be followed by a newline character, but there could be other whitespace between the magic number and the newline.

All lines immediately following the magic number and starting with the `#` symbol are considered to be comments. After the optional comment lines, the width and height (in that order) of the image are given as integer values in ASCII. Whitespace separates these two values, and they are followed by whitespace. If the image is a greymap or colormap, then the next value in the file is the maximum value for each pixel in a color or graymap channel.

For the images that we use, this value will always be 255 or less. Bitmap images do not have the channel depth, as all pixels are 1 bit by definition. The ASCII header must end with a single whitespace character (usually newline). Following the header is the data that makes up the image. The data may be either ASCII or binary, depending on the magic number.

If the magic number was `P6`, the data will be in binary format. The image data is stored in the same order for both types of images. The data is supplied for each row. Row 1 being first, followed by row 2, row 3, . Row n. When working with the data, column 1s data will be first followed by column 2, column 3, .. Column n. Each column in every row has 3 values, a red channel, a green channel, and a blue channel. They are supplied in that order. The following is an example ASCII PPM file:

```
P3
# Sample color image
3 2
255
1
2
3
4 5 6
7 8 9
10 11 12
13 14 15 16 17 18
```

This specifies an ASCII color image with three colums and two rows, and each pixel is made up of three values (red, green, and blue) between 0 and 255. In image processing, the channels of a color image are often separated and processed independently. The red channel data is 1 4 7 10 13 and 16. The green channel data is 2 5 8 11 14 and 17. The blue channel data is 3 6 9 12 15 and 18. If the data was in binary format, you would not be able to read the values, but the organization and order of the data is the same.

A greymap image is very similar to a color image, but has only one value (brightness) for each pixel. The following example specifies a very small greymap image.

```
P2
# sample greymap image
3 2
255
1 2 3 4 5
6
```

Execution of your program

I will run your program from the command line and you will have to trap errors such as too many or two few arguments, and unknown arguments. Your program should accept command line arguments according to the following pattern:

```
rp06 $ ./imagetool <options> input_filename output_filename
```

A minimal run of your program must have two arguments supplied to your program, such as:

```
rp06 $ ./imagetool lena.ppm lena_binary.ppm
```

This is an example where no `option` has been supplied to change the image. This is useful just to duplicate and image. The first argument in this example specifies how the final image will be stored.

The user could specify a processing option to be applied to the image prior to output. To keep things simple, only one processing option can be specified at a time. The options are specified between the program name the input file. Only the following options are allowed:

Option Code	Option Name
`-n`	Negate
`-b <number>`	Brighten
`-p`	Sharpen
`-s`	Smooth
`-g`	Grayscale
`-c`	Contrast
`-o <a|b>`	Output in ASCII or binary

Notice that the `-b` option requires a numerical value, and the `-o` option requires a character (either `a` or `b`). If `-oa` is specified, the data will be written in ascii format. If `-ob` is specified, the data is to be in binary format.

Negate

If `-n` is specified, you need to negate each pixel in by applying the following equation to each pixel $p_{i,j}$: $p_{i,j} = 255 - p_{i,j}$.

Brighten

If `-b` is specified, the following argument must be be an integer, v, in the range of $[-128 \ldots 127]$. Apply the following formula to every pixel $p_{i,j}$: $p_{i,j} = p_{i,j} + v$. The resulting value of $p_{i,j}$ must be saturated, and not allowed to overflow or underflow.

Sharpen

if `-p` is specified, your program must sharpen the image. This can be done by subtracting the the hoizontal and vertical neighbors from four times the value of the center pixel. Apply the following formula to every pixel $p_{i,j}$: $p_{i,j} = 5p_{i,j} - (p_{i-1,j} + p_{i+1,j} + p_{i,j-1} + p_{i,j+1})$. Note that if $p_{i,j}$ is on the image border, the function must be modified slightly. As you compute each

new value, store it into a separate array. This is very important. When you have processed all pixels, the new array will replace the old array.

Smooth

Smooth: If `-s` is specified you need to smooth (blur) the image. This is done by performing a weighted average of the 9 pixels in a 3×3 neighborhood. This must be done to all three color bands on color images, or a single band for grayscale images. As you compute each new value, store it into a separate array. This is very important. When done processing a color band, the new array will replace the old array. Apply the following formula to every pixel $p_{i,j}$:

$$p_{i,j} = \frac{4p_{i,j} + 2(p_{i-1,j} + p_{i+1,j} + p_{i,j-1} + p_{i,j+1}) + p_{i-1,j-1} + p_{i+1,j-1} + p_{i-1,j-1} + p_{i-1,j+1})}{16}.$$

Grayscale Images

The next two options produce only grayscale images. When you output the new image, you will need to add a `.pgm` extension to the base name and follow the output format for a `P2` and `P5` image type.

Grayscale

If the grayscale option is specified, you will need a new array the same size as the rgb arrays to hold the grayscale values. Apply the following equation to all pixels to produce a grayscale image.

$$g_{i,j} = 0.21r_{i,j} + 0.72r_{i,j} + 0.07b_{i,j}$$

You now need to use the magic numbers `P2` or `P5` when outputting the image and change the extension to a pgm. Only output the gray array.

Contrast

If the contrast option is specified, you will first need to convert the image from color to grayscale using the procedure described above. While converting to grayscale, keep track of the min and max values. Next, compute the scale factor s using the following equation:

$$s = \frac{255}{max - min}$$

. Then traverse your grayscale image applying the following equation:

$$g_{i,j} = s \times (g_{i,j} - min) + 0.5,$$

then output the gray array as a `P2` or `P5` image.

Important Note

No matter what option you are doing, you must maintain the ranges of 0 to 255. If a value exceeds 255, then it should be set to 255. If a value becomes negative, should be set to zero. You also need to handle rounding when necessary.

There are many freely available programs that will allow you to view PPM images. At least one is installed on the machines in the Opp Lab. You do not have to write all of the code yourself. You can download the compressed `tar` file from the course web site and unpack it. It contains a complete program skeleton, which you are free to modify.