Computer Science 229
Homework Project 1
2000 points

C Programming Project (fun with image processing)

Notes

Aspects of this project may contain some mathematics. This is not a class in image processing nor the mathematics of image processing. However, in an attempt to create more interesting assignments for computer scientists, engineers, and other majors, this programming project (and the next) are real-world applications, or an approximation of real-world applications. However, I do not expect you to derive or understand some of the mathematics in parts of these projects. You will be able to understand how to program the equations, without understanding he basic mathematics behind them. There may be parts of this assignment that state that certain aspects of the program or assignment will be discussed in class. This means that I will probably give you big hints on how to approach a particular problem or issue. You should take this as a strong hint to not miss class; however, if it is necessary to miss class, make sure you have a friend or someone in class that will share their notes with you.

For logistic reasons, detailed instructions for later parts of this assignment will be amended to the assignment at a later time. However, to give you an idea of what to expect, all of the parts of the assignment are summarized below. This may help you plan how divide source code in files. However, it is likely that you may need to refactor your code and/or placement of code at a later time in the project. Note that only the checkpoint code is due initially for each part. 80% of the points will be determined with your final submission which will consist of all parts.

Introduction

In this project you will create a series of programs that can manipulate and process images. There are several parts to this project which are summarized below. The order below may not be the order of the checkpoint due dates.

a) Create a program that creates an image of a specified size with a square placed in the middle of a specified size and color in CS 229 image format.
b) Write a program that reads an image file in CS229 format and displays information and statistics about it.
c) Simple conversion program
d) Simple photographic filters and utilities.

e) Create a program that takes images specified in the command line and places items that are not in a specified range of colors in front of the prior picture. (green screening)
f) Convolution kernels and image processing.
g) Use JNI to create an interface from java to C to compute convolutions.
h) Use Java Swing and the JNI in part g to create a mini picture processing lab.
i) Create a barcode reader.
j) Create a complete make system for the project.

Central to all parts of this project is the file format for images. There are many standard formats, but in this class we will use our own format. Here are the detailed specifications for 229 image format.

Byte 0: (the first byte in the file) Color/BW selector

This byte indicates if the file is color or black and white. A value of 0 indicates black and white, a value of 255 indicates color. All other values in this byte are invalid.

Bytes 1, 2 and 3: Channel size

These bytes indicate the number of bits for each color pixel , or the number of bits for black and white image pixel. Valid channel sizes are 4, 8, 12, and 16 bits. In the case of black and white images, only byte 1 is used to determine the channel size.

Byte 4, 5, 6, and 7: Image Width

These bytes specify the image width in binary. Byte 4 is the most significant bits and byte 7 is the least significant bits.

Byte 8, 9, 10, and 11: Image Height

These bytes specify the image height in binary. Byte 8 is the most significant bits, and byte 11 is the least significant bits.

Byte 12, 13 …: Image Data

Pixel data for the image. For black and white (single channel) images, pixel data is packed with most significant bits before least significant bits. Pixels are stored in row-major order in which data from the ith row precedes data from the ith + 1 row and within the row data is stored left to right. Color image pixel data is stored the same as black and white pixel data except each pixel consists of 3 channels, (red, green, and blue, in that order) packed from most significant bit to least significant bit for each channel. Image data must contain exactly the number of pixels specified by the image height and image width.

Example images and file formats.

0	0
1	8
2	0
3	0
4	0
5	0
6	0
7	3
8	0
9	0
10	0
11	2
12	14
13	6
14	2
15	7
16	15
17	9

14	6	2
7	15	9

Example black and white image with pixel values and the file format that represents the image with 8 bits per pixel.

0	0
1	4
2	0
3	0
4	0
5	0
6	0
7	3
8	0
9	0
10	0
11	2
12	230 (E6 hex)
13	27
14	249 (F9 hex)
15	7
16	15
17	9

14	6	2
7	15	9

Example black and white image with pixel values and the file format that represents the image with 4 bits per pixel, packed.

0	255
1	8
2	8
3	8
4	0
5	0
6	0
7	3
8	0
9	0
10	0
11	2
12	255
13	255
14	255
15	242
16	19
17	25
18	0
19	0
20	0
21	45
22	147
23	22
24	2
25	26
26	100
27	105
28	174
29	10

255, 255, 255	242, 19, 25	0, 0, 0
45, 147, 22	2, 26, 100	105, 174, 10

Example color image with pixel values and the file format that represents the image with 8 bits per pixel.

a) (200 points) Create a program called “makesquare” that generates a CS229 image file containing an image of specified size with a square placed in the middle of a specified size and color on a white background. The size and color of the square and the size of the image is specified by either command line arguments, or by standard input, but not both. If no command line parameters are present (as in when the program is executed by just typing the name of the program) then parameters are read from standard input. The program asks the user for the specified input in the following order.

color or black and white image
image width
image height
square width
square height
square color (red, green, blue) if color or single value if black and white. The channel size is 8 bits.

If the parameters are entered by command line input, the format is as follows. (If any command line parameters are entered, then no input from standard input is performed.)

makesquare (-color | -bw) imagewidth imageheight squarewidth squareheight squarecolor

Examples:

```
makesquare -color 100 100 30 30 0 0 255
```

This creates an image file of size 100 by 100 pixels with a 30 by 30 pixel blue square centered on a white background. The color white has red green blue values of 255, 255, 255.

```
makesquare –bw 100 100 30 30 0
```

The program must perform error checking and alert the user via standard output on any error condition. Some error conditions include the square larger than the image, color values out of range, and illegal height or width values. It is up to you to think of all the error conditions that can occur.

b) (100 points) Write a program called “imagestats” that scans an image file in CS229 image format and reports back if the image is color or black and white, the size of the image, the number of bits per channel, the percentage of white pixels in the image, and the percentage of black pixels in the image. The image file is read via standard input by I/O redirection, or by the file name using command line parameters. Example:

```
imagestats < myfile.dat
```

```
imagestats myfile.dat
```

These two executions of the program should output the same thing.

c) (200 points) Write a program called "cs2ppm" that reads a cs229 color image format file and outputs a ppm format file. The ppm format is defined below. The program reads the cs229 formatted data from stdin and outputs the ppm formatted data to stdout. This will allow you to view images that you create and process. You must also write a program called "ppm2cs" that converts in the other direction.

Each PPM image consists of the following: (as specified in http://netpbm.sourceforge.net/doc/ppm.html)

1. A "magic number" for identifying the file type. A ppm image's magic number is the two characters "P6".
2. Whitespace (blanks, TABs, CRs, LFs).
3. A width, formatted as ASCII characters in decimal.
4. Whitespace.
5. A height, again in ASCII decimal.
6. Whitespace.
7. The maximum color value (Maxval), again in ASCII decimal. Must be less than 65536 and more than zero.
8. A single whitespace character (usually a newline).
9. A raster of Height rows, in order from top to bottom. Each row consists of Width pixels, in order from left to right. Each pixel is a triplet of red, green, and blue samples, in that order. Each sample is represented in pure binary by either 1 or 2 bytes. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The most significant byte is first.

The program only converts color images that have 8 or 16 bit channels with the red, green, and blue channels of the same size. The program indicates an error for any other type of image.

Note that you can use this program to convert CS229 images to "png" format files that you can easily view. The command to do this is:

```
cs2ppm < "myfile.dat" | pnmtopng > outfile.png
```

d) (300 points) In this part you will create several filters that input an image and output a processed image. All input to a filter program will be though stdin and all output of a filter program will be to stdout. This will allow for the chaining of filters together to process an image with several filters with a single command line. Here are the filters to implement.

1. Darken. The “darken” program darkens an image by reducing each pixel value by a specified percentage. For color images, all red, green, and blue pixels are reduced by the specified percentage. A single command line argument specifies the percentage to darken. A percentage of 0 will leave the image unchanged, and a percentage of 100 will leave all pixels in the image completely black. Here is an example that darkens the image by 50 percent.

   ```
   darken 50 < in_image.dat > out_image.dat
   ```

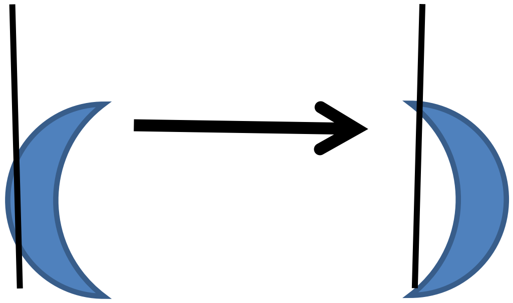
2. Lighten. The “lighten” program lightens an image by increasing each pixel value by a specified percentage. For color images, all red, green, and blue pixels are increased by the specified percentage. A single command line argument specifies the percentage to lighten. A percentage of 0 will leave the image unchanged, and a percentage of 100 will leave all the pixels in the image completely white (max possible value for the channel size) Here is an example that lightens the image by 25 percent.

   ```
   lighten 25 < in_image.dat > out_image.dat
   ```

3. Rotate. The “rotate” program rotates an image 90 degrees left, 90 degrees right, or 180 degrees (upside down). The program takes a single parameter “90”, “-90”, or “180” to specify the rotation. Example:

   ```
   rotate -90 < in_image.dat > out_image.dat
   ```

4. Flip. The ‘flip” program mirrors the image either horizontally or vertically. The program takes a single argument of either “h” or “v” that specifies vertical or horizontal mirroring. For a horizontal flip, the image pixels are mirrored around the vertical. Similarly, for vertical flip, the pixels are mirrored around the horizontal. (see example below)



This is an example of a horizontal flip produced by the command line:

```
flip h < in_image.dat > out_image.dat
```

e) (200 points) In this part you are to write a program called “matte” that allows you to superimpose one image on another in a specified area. Call the two images image1 and image2, and consider a set of colors defined by r1, r2, g1, g2, b1, b2, where the red values are between r1 and r2 inclusive, the green values are between g1 and g2 inclusive, and the blue values are between b1 and b2 inclusive. The result of the matte program is for all the pixels in image1 that fall in the color range defined by r1, r2, g1, g2, b1, and b2, substitute the pixel at that location that is in image2.

Image1 and image2, as well as r1, r2, g1, g2, b1, and b2 are specified on the command line in that order. Image1, and image2, and the output image are all in CS229 image format. The output image is written to standard out. Any errors are to be written to stderr.

Errors may occur if image1 or image2 are not of the correct format, or if they are not of the same width and height.

f) (300 points) Convolution is an easy way to achieve a wide variety of effects in images. The basic idea is to define a small matrix that is used to perform a mathematical operation on each pixel. The diagram below shows a 3 by 3 kernel. To process a black and white image with this kernel, place the center square of the kernel over every pixel in the image and evaluate the sum of the product of each pixel under the kernel. The result is the value of the pixel under the center square of the kernel. If part of the kernel is outside the image, then the pixel values under kernel at those locations are zero.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

For example, for the above kernel and the 6 by 4 black and white image below, the result of a convolution is shown.

5	3	2	1	3	5
6	2	1	2	4	1
5	5	5	5	1	4
4	4	1	2	1	1

Original image.

2	2	1	1	2	1
3	4	3	3	3	2
3	4	3	2	2	1
2	3	2	2	2	1

New image. (Using rounding)

The result of this kernel will smooth the image. For this assignment, kernels must be square of an odd number size. (3 by 3, 5 by 5, 7 by 7, etc) The maximum size of a kernel is 15 by 15. Other interesting kernels are:

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

-1	1	0
1	0	-1
0	-1	1

These kernels will accentuate lines of various directions in the image.

A kernel file is an ASCII file (text file) that contains the definition of the kernel. The first number is the size of the kernel. The rest of the file contains the numbers (n * n of them) where n is the size of the kernel. For color images, the kernel file contains the 3 different kernels, each may be of a different size. (The size is defined before each kernel in the file in the order of red, green, and blue.) When a kernel is applied to a color image, each kernel for the red, green and blue pixels is applied to the red, green, and blue pixels of the image.

Write a program called “convolve” that specifies a kernel file as a parameter on the command line, and takes as input from stdin a cs229 image file. The resulting process image is output to stdout. Any error is reported to stderr. Errors include illegal format for the kernel file, image files, applying a color kernel to a black and white image or visa versa. Other errors that you think of or come across should also be reported through stderr. An example command line is:

```
convolve kfile.txt < input.dat > output.dat
```

g) (100 points) Using the Java native Interface, write a simple Java console program that reads two 3 by 3 matrices from the user, multiplies them together, and returns the result. The signature for the JNI call from Java is:

```
public native void( int arr1[][], int arr2[][], int arr3[][] );
```

where arr1 and arr2 are the two arrays that are multiplied together and the result is placed in arr3.

h) (200 points) Write a Java GUI interface using Java Swing that displays and processes images in CS229 format. Your program need only display images of size 640 wide by 480 high. You may error if the user tries to load a file of a differing size, or you may support arbitrary sizes. The program must include buttons to rotate the image left and right, vertical and horizontal mirroring, blur/soften the image, and sharpen lines. These last two operations are accomplished through convolution. The result of any image operation is displayed in the user interface. You must include written documentation using a “readme” file that tells the user how to run and use your program.

i) (300 points) Write a program that takes as input a CS 229 image file that contains a UPC barcode. The barcode may be positioned anywhere in the image, but only a small number of points will be assigned to finding a barcode at an angle other then left to right horizontal. Your goal is to output the numeric UPC barcode in the image to stdout. The specification for UPC-A can be found in many places. Here is one such reference. http://en.wikipedia.org/wiki/Universal_Product_Code Note that it is guaranteed that there is only one barcode in the image.

j) (100 points) Write an appropriate makefile for all programs above. Also include a target called all (which should also be the default target) that creates everything. Also include a target called clean that remove all object files, and a target called cleandist that remove all generated files created.

Documentation

Documentation and proper style will be at least 15% of the grade.