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 the 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 to 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 CS229 image format.
b) Write a program that reads an image file in CS229 format and displays information and statistics about it.
c) Create a simple conversion program.
d) Create simple photographic filters and utilities.
e) Convolution kernels and image processing.
f) Create a program that stitches images together.
g) Use JNI to create an interface from Java to C to compute matrix multiplication.

h) Use Java Swing and JNI to create an image viewer/editor with convolution capability.
i) 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) Magic Number. This byte indicates that the file is a CS229 image file type. It is the hex number 42.

Bytes 1, 2 and 3: Channel size

These bytes indicate the number of bits for each color pixel, or the number of bits for the black and white image pixel. Valid channel sizes are 4, 8, and 16 bits. In the case of black and white images, only byte 1 is used to determine the channel size, and the values of the other bytes are ignored (but must be present). A value of 1 indicates 4 bits, a value of 2 indicates 8 bits, and a value of 4 indicates 16 bits. Note that channels may have different sizes within the same color image. For example, the green channel may be 8 bits while the red and blue channels are 4 bits each.

Byte 4: 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 considered invalid.

Byte 5, 6, 7, and 8: Image Width

These bytes specify the image width in binary. Byte 8 contains the most significant bits and byte 5 contains the least significant bits. (Please read the bit order again – the least significant bits are in the first byte that is read.)

Byte 9, 10, 11, and 12: Image Height

These bytes specify the image height in binary. Byte 12 contains the most significant bits, and byte 9 contains the least significant bits.

Byte 13, 14 …: 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 i^{th} row precedes data from the $i^{th} + 1$ 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. NOTE that pixel data is stored from the most significant byte down to the least significant byte.

Example images and file formats:

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

Byte Hex Values:

0	0x42
1	0x02
2	0x00
3	0x00
4	0x00
5	0x03
6	0x00
7	0x00
8	0x00
9	0x02
10	0x00
11	0x00
12	0x00
13	0x0E
14	0x06
15	0x02
16	0x07
17	0x0F
18	0x09

Image:

14	6	2
7	15	9

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

Byte Hex Values:

0	0x42
1	0x01
2	0x00
3	0x00
4	0x00
5	0x03
6	0x00
7	0x00
8	0x00
9	0x02
10	0x00
11	0x00
12	0x00
13	0xE6
14	0x27
15	0xF9

Image:

14	6	2
7	15	9

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

Byte Hex Values:

0	0x42
1	0x02
2	0x02
3	0x02
4	0xFF
5	0x03
6	0x00
7	0x00
8	0x00
9	0x02
10	0x00
11	0x00
12	0x00
13	0xFF
14	0xFF
15	0xFF
16	0xF2
17	0x13
18	0x05
19	0x00
20	0x00
21	0x00
22	0x2D
23	0x93
24	0x16
25	0x02
26	0x1A
27	0x64
28	0x69
29	0xAE
30	0x0A

Image:

255, 255, 255	242, 19, 5	0, 0, 0
45, 147, 22	2, 26, 100	105, 174, 10

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 (value) if black and white.

The channel size is always 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 (-c | -bw) imagewidth imageheight squarewidth squareheight squarecolor

Examples:

```
makesquare -c 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.

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

This creates an image file of size 100 by 100 pixels with a 30 by 30 pixel black square centered on a white background.

The color white has red green blue values of 255, 255, 255.

The program must perform error checking and alert the user via standard error 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” and a program called “ppm2cs”. “cs2ppm” reads a cs229 color image format file and outputs a ppm format file. “ppm2cs” converts in the other direction. It reads a ppm format file and outputs a cs229 color image format file. The ppm format is defined below. Each program reads the input data from stdin and prints the output data to stdout. Using “cs2ppm” will allow to you actually view the images you create.

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.

For cs2ppm, only color images with channel sizes of (8, 8, 8) or (16, 16, 16) are allowed. The program indicates an error for any other type of image, to stderr as usual. Maxval must be set to the value representing white for that channel size.
For ppm2cs, the output image will have channel sizes (8, 8, 8) or (16, 16, 16) depending on maxval. All values are scaled linearly to this channel size. Scale with the formula newvalue = (oldvalue * newmax)/oldmax. When scaling, truncate; do not round (i.e. use integer division).

These restrictions ensure that no bit-packing need be done when converting to or from a ppm file.

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. All channel sizes and both color and black and white images are valid. Each filter will have its own executable program, named as indicated. Any errors are always output to stderr, NOT stdout. The filters to implement are:

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 clockwise, 90 degrees counterclockwise, or 180 degrees (upside down). The program takes a single parameter "90" (CW), "-90" (CCW), 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.

e) (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.

Original image:

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

New image. (Using rounding):

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

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 (human-readable 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 numbers are whitespace-delineated, floating-point values. (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 respectively.

For the first kernel listed here, the kernel file contents would be:

3 .1111 .1111 .1111 .1111 .1111 .1111 .1111 .1111 .1111

For an RGB kernel, it's identical to 3 regular kernel files concatenated together, separated by newlines, in the order R G B. You do define the kernel size separately each time, and there is no requirement that they be of the same size.

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 or image files, applying a color kernel to a black and white image or vice 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
```

f) (300 points) Image stitching is the process by which multiple images of the same scene are stitched together to make one image as discussed in lecture. In this part, you will write a program that will stitch two images together at the correct position. To

accomplish this, we must define how "close" a section of two pictures are to each other. We first define the distance between two pixel values. The following formula gives a measure of how close two pixels are to each other. Note that if two pixels match perfectly, then this formula gives a value of 0. If they do not match, then this formula gives a positive value.
Let p1 and p2 be two pixels with p1.r, p1.g, and p1.b be the red, green and blue values for the pixel p1. Define the red, green and blue values for p2 similarly. The distance between p1 and p2 is then defined as follows.

$$distance = (p1.r - p2.r)^2 + (p1.g - p2.g)^2 + (p1.b - p2.b)^2$$

To find the best place to stitch the two images together, consider all possible overlaps of the two images where the overlap contains at least 10 percent of the smaller image, measured in terms of the number of pixels. For example, if the two images to be stitched are of sizes 100 by 100 and 150 by 150 in size, then at least 10 percent of the total pixels in the 100 by 100 image must be contained in the overlap. That is, at least 0.1x100x100 = 1000 pixels must be in the overlap. To find the total error in a particular overlap, simply sum the distance (as defined above) between each pixel value of the two images in the overlap area. To find the *weighted* overlap, divide the total overlap by the number of pixels in the overlap area (weightedOverlap = totalOverlap / sizeOfOverlap). The best stitching is then the overlap that has the smallest *weighted* error. Using this overlap, create a new image that is the smallest image that contains both images. The pixels in the overlap area are computed by taking the average between the two pixels from each image. Because an image must be rectangular, any pixel outside of either image is the color white.

Assumptions you may make.

- Both images are oriented correctly. That is, rotations are NOT required to stitch the two images.
- The image channel sizes are all 8 bits color images and 8 bits only for black and white images. Proper error checking should be employed through stderr for images that are not of this form.
- To find the stitching point for a black and white image you may utilize the color algorithm with two of the channels set to zero.

The program you are to write is called "stitch" and executes with three parameters (command line arguments). The first two are the names of the two input image files to stitch. The third parameter is the output image file. Thus, for this program you do not use stdin or stdout at all. All image files are in CS229 image format. Here is an example command line:

```
stitch infile1 infile2 outfile
```

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) (250 points) For this section, write a Java Swing program that allows the user to view and edit cs229 images. The GUI must have an area in which to display an image and three buttons. The first will open a fileopen dialog box (JFileChooser) that allows the user to select the file that is then loaded into the display area. The second button will perform a vertical line detect convolution on the image with a convolution kernel size of 5x5. The third button will open another dialog box to save the altered image. Loading a new image while an altered, unsaved image is displayed should prompt the user to save or discard changes to the original image, then load the new image.

The display area is to be 600 by 600. The image must be displayed at full size and maintaining the aspect ratio of the original image. If the user attempts to load an invalid image, or one that is too large to be displayed, print an error gracefully and offer to load another image. If you wish, you may allow larger images to be displayed, and accessed via scrollbars.

To receive full points, you must use JNI to call your convolute code from part e, above. You may do this simply by abstracting the contents of your main method from part e into a function, and then calling that function both from the part e main method and through JNI for this section. If you elect not to use JNI, you will receive a maximum of 200 out of the possible 250 points. To do this, you may call your part e executable itself, or re-write the code in Java.

The .java file as well as the .class result that contain the main method should be called DisplayRunner.

i) (150 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 removes 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.