**CMPT 365 Multimedia Systems**

**Project 1**

**Q. 1.**

Since I use iPhone, I already know from experience and personal usage that it, iOS, supports at least the following formats. All these formats I discovered them on my iPhone library after organizing everything by “type”.

**Video Formats**: AVI, MP4

**Audio Formats**: MP3, WAV

**Image Formats**: GIF, BMP, PNG, JPEG,

A complete list of all the supported video, audio, and image formats is also publicly available on Apple’s website for customers.

<https://support.apple.com/kb/PH16864?locale=en_US>

As far as I know, I am also certain that Android phones also support those basic types of formats I listed because before using iPhone I used to use Android for many years.

**Q. 2.**

**How is waveform produced?**

In the “Helper.java” and “Bresenham.java” file there are methods that I implemented to help draw the waveforms; below is a list of methods I use together in order to produce the final waveform.

Helper.java

public static double[] normalizeData(byte[] b, int pos, int length);

public int mapRange(double x, int in\_min, int in\_max, int out\_min, int out\_max)

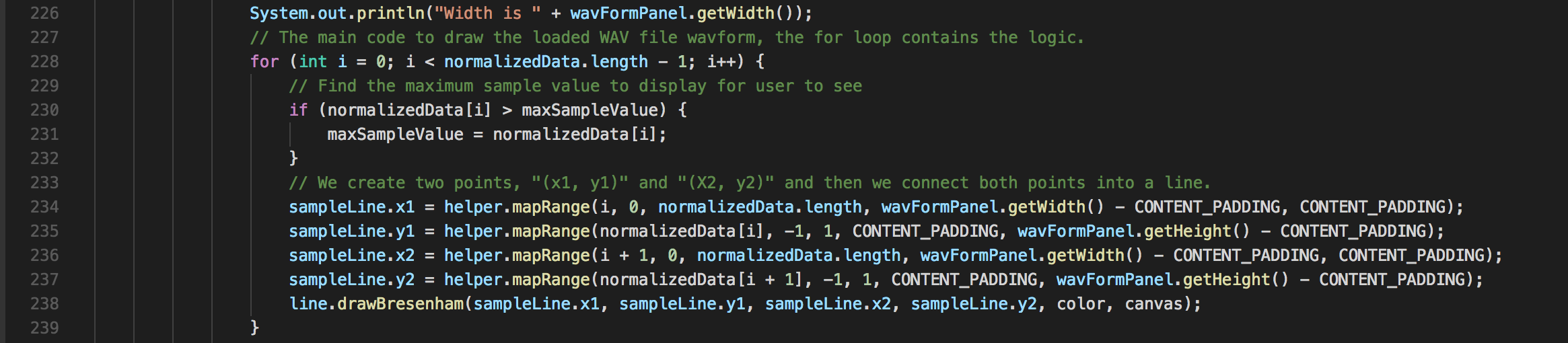
Bresenham.java

public void drawBresenham(int x1, int y1, int x2, int y2, int color, BufferedImage canvas)

In general, I first obtain the data from the WAV file and normalize the data so that it is within -1 to 1. This is because we are dealing with 16-bit data, so it is better to normalize to smaller values that we can work with. Then I map the data onto the range of the drawing space I allocated in my program display or else the waveform will look really tiny. Finally, I go through the array of normalized data and I set pixels on the display and connect the pixels with my line drawer method known as “Bresenham” which draws a line from point A to point B. The actual logic that draws the waveform is in a for loop located on lines 217 – 227 in “WavBmp.java” or copy pasted below:

WavBmp.java

This code is part of the openFileBrowser() method.



**Q. 3.**

**How is image produced?**

Since we are not dealing with any type of line drawing, it was only necessary to start reading the data from the BMP file. Every three consecutive bytes of data is in the sequence BLUE, GREEN, RED and they represent a single pixel. After converting the byte data to type INT for each of the three-color channels I then combine them into a single color by creating a new “**Color**”. After combining them into a single color I simply call the basic system method **setRGB** to draw a single pixel at a coordinate on the display that I mapped to from the BMP file dimensions with the color. For obtaining color spaces such as luminance, grayscale, and RGB the following operations are used in source code:

WavBmp.java

This code is part of the openFileBrowser() method.

//Grab the blue, green, and red color channels which are 3 consecutive bytes

blue = (int) (wav[(int)myBmpFile.dataOffSet + x] & 0xff);

green = (int) (wav[(int)myBmpFile.dataOffSet + x + 1] & 0xff);

red = (int) (wav[(int)myBmpFile.dataOffSet + x + 2] & 0xff);

pixelColor = new Color(red, green, blue);

// This is the matrix operations needed to get Y when you have all three color

channels R, G, B.

luminance = (int) (0.299 \* (red)) + (int) (0.587 \* (green)) + (int) (0.114 \* (blue));

grayScalePixel = new Color(luminance, luminance, luminance);

The methods I used are nothing special but listed below is the main code used to draw the original image when loaded.:

WavBmp.java

This code is part of the openFileBrowser() method.

****

**Choice of Dither Matrix**

The dither matrix I chose to use was of size 5x5 and actually I made it by first taking the 4x4 dither matrix Professor Jiang showed in class and then inserting the remaining values of 17 to 26 into it randomly to create the 5x5. This surprisingly yielded good dither results and so I decided to use this method. The dither matrix is the following:

**{ 0, 23, 2, 10, 26 }**

**{ 12, 22, 14, 6, 17 }**

**{ 3, 11, 1, 9, 24 }**

**{ 15, 7, 13, 5, 19 }**

**{ 18, 20, 21, 4, 8 }**

For all other key source codes on drawing images like the histograms, they are also located in the Helper.java file and called in the WavBmp.java file.:

Helper.java

public void drawHistogram(int[][] arr, int offset, int color, BmpFile myBmpFile,

BufferedImage canvas, SampleLine sampleLine)

**Screenshots of Q3 Steps 2-5**

Due to there being too many screenshots, please refer to the folder called “Q3 Screenshots” included in the archive to see the screenshots.

**Input Code Implementation**

The implementation of reading input to the final step of storing the entire file contents into a “byte” array is rather simple. First I call the “**JFileChooser**” system method to open a file browser dialog for users to select either a BMP or WAV file. Then I declare a new “**File**” object and store the file obtained from JFileChooser into it. Finally, I use “**FileInputStream**” and transfer the contents into a byte array I created. Here is a screenshot of the main logic that handles this:

WavBmp.java



In order to actually make convert the byte array into useful information I use bit shifting to re-arrange specific sections of the data because it is in big-endian format. Here are two methods I used to help do some conversions from byte to base 10.

Helper.java

public static int byteArrayToInt(byte[] b, int pos);

public static long byteArrayToLong(byte[] b, int pos, int length);

Note that both WAV and BMP files share this implementation of reading the file contents into a byte array which I then read the byte information to determine if it is either a BMP or WAV file.