Project 1 Report

Q1. Audio: Mp3 and WAV.

Image: JPEG and PNG.

Video: MPEG and AVI.

I Google it and find the answers through Quora and Stackoverflow. There are multiple other answers and I chose two that are most representative for every category. Mp3 is generally used for music and for anything else the standard format is wav. JPEG and MPEG are the main format classes that contain several sub-categories for image and video, respectively. AVI stands for Audio / Video Interlaced, and PNG is the short for Portable Network Graphics.

Q2.

Since we assume that the wav file is mono, there is only one channel and each sample consists of 16 bits (two bytes). So every two adjacent bytes are taken as one sample frame when reading the input byte array.

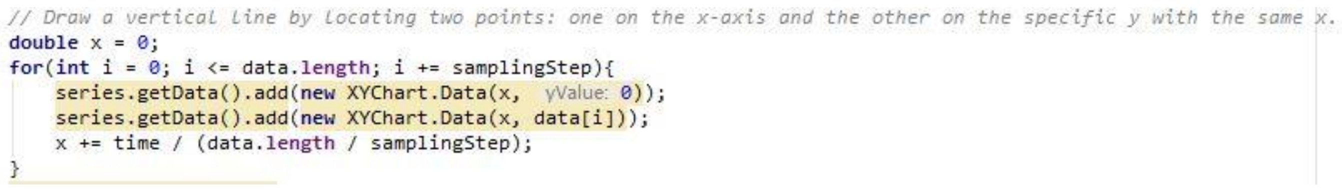
One common problem is that typically the number of samples in the raw data files exceeds 10 thousand and regular computer screens do not support such high resolution to display that much samples. Therefore, it might be necessary to do subsampling. In my waveform graph, the total number of samples is a little more than 1000 – I did it by taking one sample in the original sample array and taking another by ignoring evenly many samples in between the two samples I need. Thus the final waveform displayed is just an approximation.

Q3.

One challenge of the bmp file input is that we need to know the resolution of the image which is also the dimension of the bitmap. Another problem is to get rid of the header as we only use the pixel array in the bmp file. Fortunately, both the horizontal and vertical resolutions and the size of the header in bytes are available through indexing in the header. So we can construct the bitmap properly as every three adjacent bytes(RGB) represent a pixel.

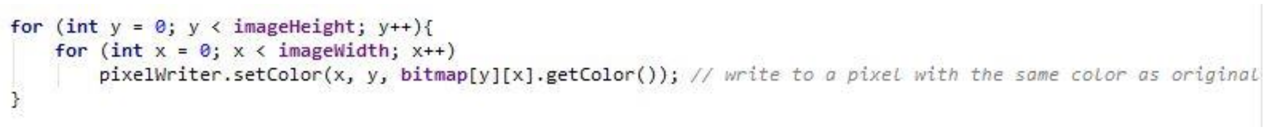
All pixels are written from left to right, from top to bottom. For the original, pixels are written using the color represented by the RGB value in the bitmap. For the brighter, the pixel's RGB values are raised to 1.5 times of itself but no more than 255. For the grayscale, the 8-bit grayscale value is used, which is calculated from the RGB value of each pixel. For the dithering, I used the dithering matrix provided in the lecture note, which produces images in quite good quality. I have tried several other matrices that are revised based on the one provided, but no satisfying as I expected. I made three histograms for color distribution for each RGB channel on my second page. They may have fairly similar distributions for the given sample bmp files. By the way, since the images whose dimensions are too large will not be tested, they will not be considered in the project since there will be error.

Screenshot of codes on how waveform is drawn:



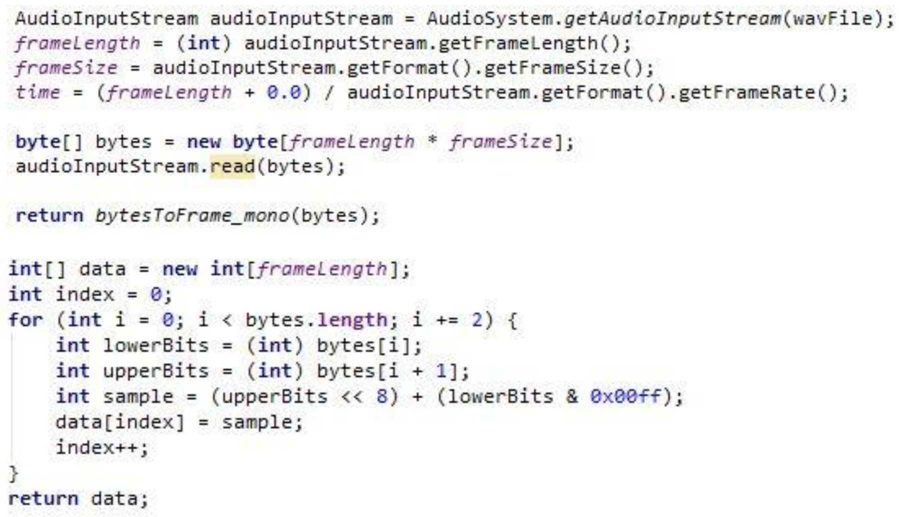
```
// Draw a vertical line by locating two points: one on the x-axis and the other on the specific y with the same x.
double x = 0;
for(int i = 0; i <= data.length; i += samplingStep){
    series.getData().add(new XYChart.Data(x,  yValue: 0));
    series.getData().add(new XYChart.Data(x, data[i]));
    x += time / (data.length / samplingStep);
}
```

Screenshot of codes on how image is drawn:



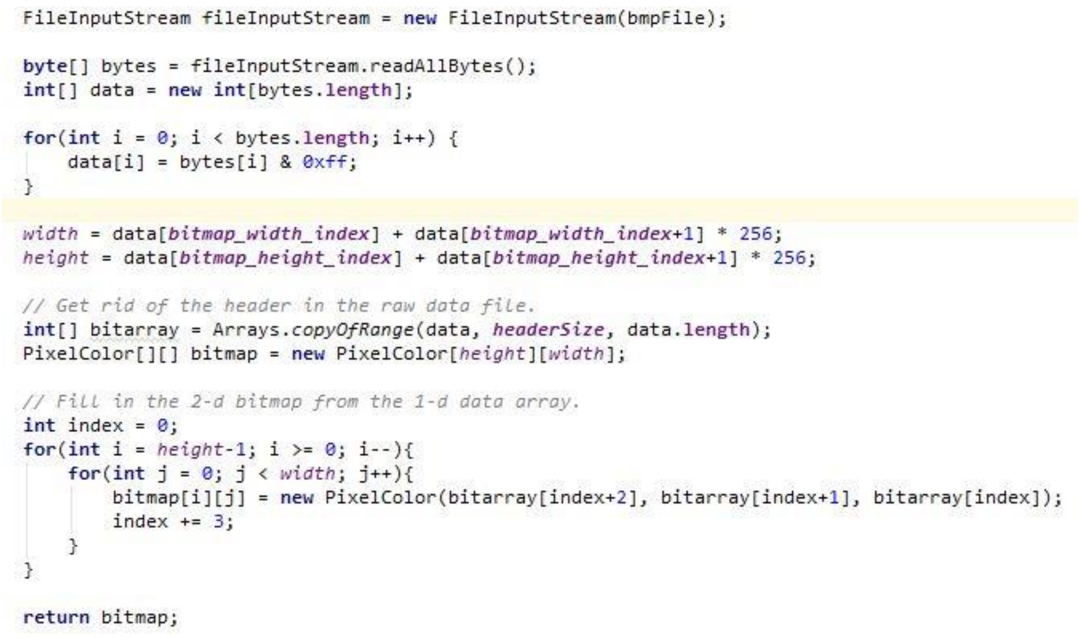
```
for (int y = 0; y < imageHeight; y++){
    for (int x = 0; x < imageWidth; x++)
        pixelWriter.setColor(x, y, bitmap[y][x].getColor()); // write to a pixel with the same color as original
}
```

Screenshot of codes on how WAV files are read:



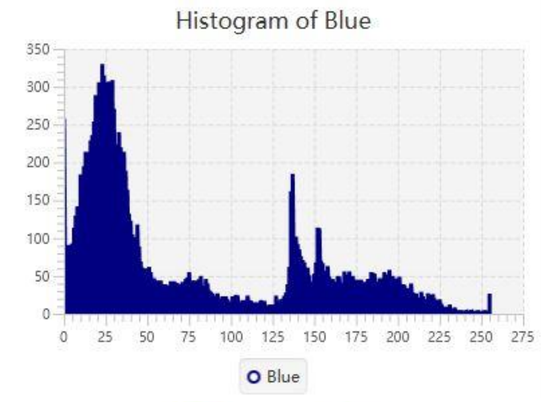
```
AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(wavFile);
frameLength = (int) audioInputStream.getFrameLength();
frameSize = audioInputStream.getFormat().getFrameSize();
time = (frameLength + 0.0) / audioInputStream.getFormat().getFrameRate();

byte[] bytes = new byte[frameLength * frameSize];
audioInputStream.read(bytes);

return bytesToFrame_mono(bytes);

int[] data = new int[frameLength];
int index = 0;
for (int i = 0; i < bytes.length; i += 2) {
    int lowerBits = (int) bytes[i];
    int upperBits = (int) bytes[i + 1];
    int sample = (upperBits << 8) + (lowerBits & 0x00ff);
    data[index] = sample;
    index++;
}
return data;
```

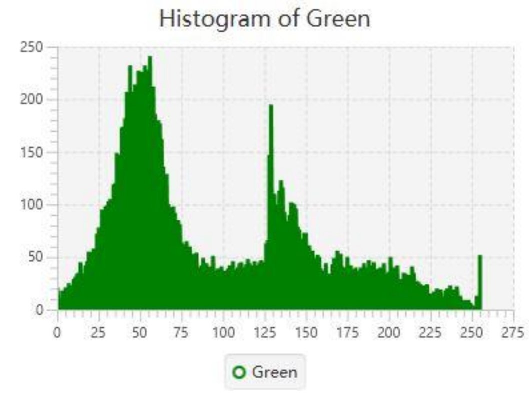
Screenshots of codes on how BMP files are read:

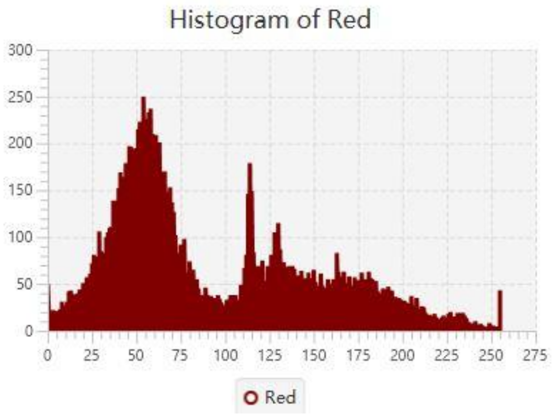


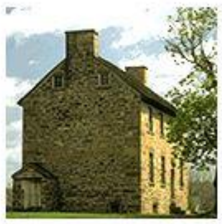
```
FileInputStream fileInputStream = new FileInputStream(bmpFile);

byte[] bytes = fileInputStream.readAllBytes();
int[] data = new int[bytes.length];

for(int i = 0; i < bytes.length; i++) {
    data[i] = bytes[i] & 0xff;
}

width = data[bitmap_width_index] + data[bitmap_width_index+1] * 256;
height = data[bitmap_height_index] + data[bitmap_height_index+1] * 256;

// Get rid of the header in the raw data file.
int[] bitarray = Arrays.copyOfRange(data, headerSize, data.length);
PixelColor[][] bitmap = new PixelColor[height][width];

// Fill in the 2-d bitmap from the 1-d data array.
int index = 0;
for(int i = height-1; i >= 0; i--){
    for(int j = 0; j < width; j++){
        bitmap[i][j] = new PixelColor(bitarray[index+2], bitarray[index+1], bitarray[index]);
        index += 3;
    }
}

return bitmap;
```

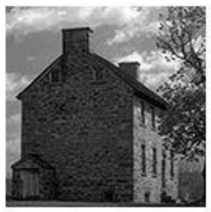
Screenshot of steps 2 – 5 of given sample files:

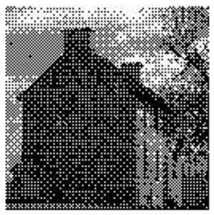


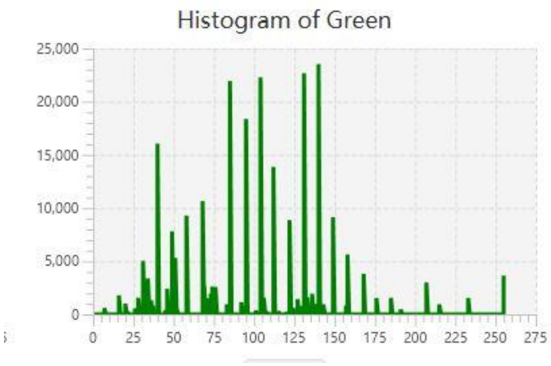


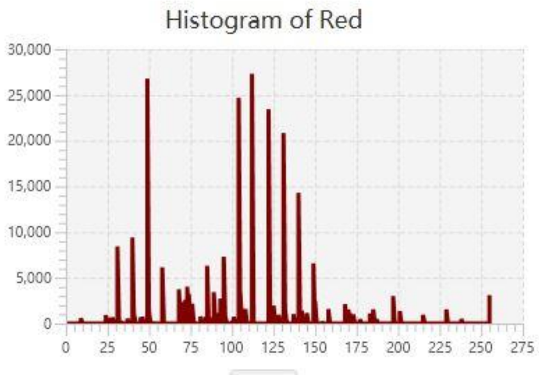


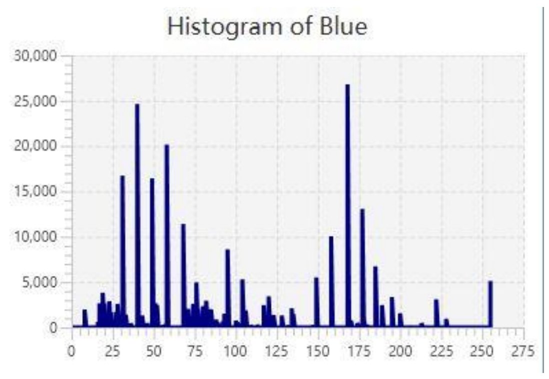











Histogram of Blue
30,000
25,000
20,000
15,000
10,000
5,000
0
0
25
50
75
100
125
150
175
200
225
250
275