Report

The project is build using the following technologies:

- IntelliJ IDE
- Maven build automation
- Java programming language
- JavaFX GUI library
- Java Sound API

Reasons for choosing JavaFX library:

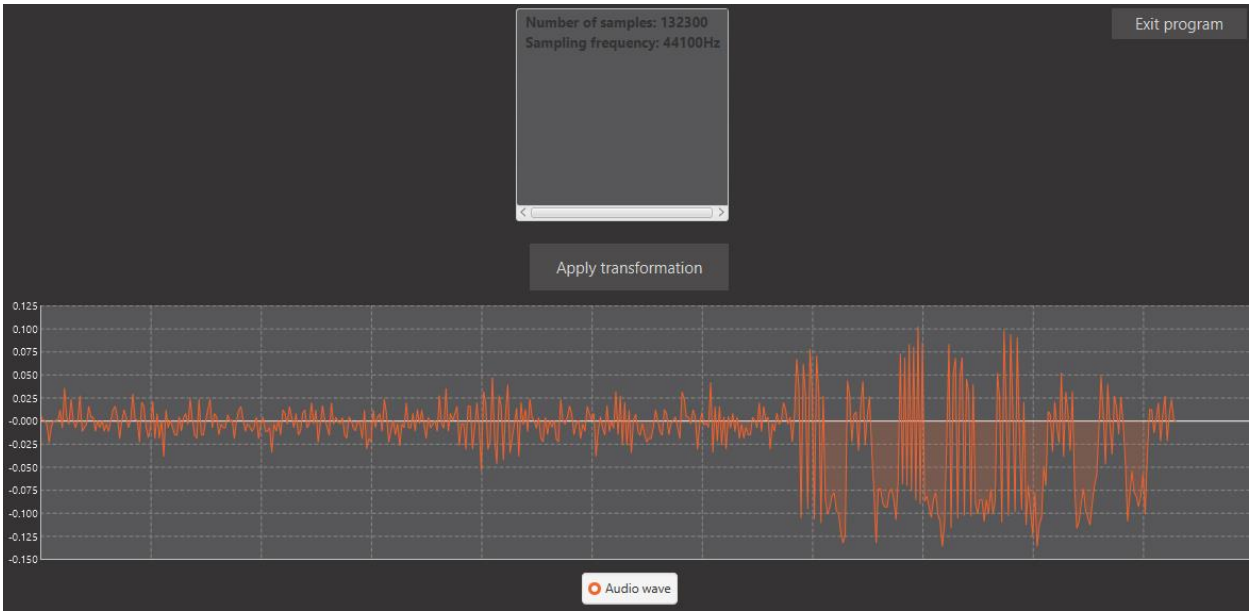
- JavaFX provides an easy framework for building GUI applications.
- Maven provides a JavaFX plugin for building and running a JavaFX application.
- JavaFX's FXML loader provides functionalities for arranging objects on the screen.
- The major reason for choosing the library is that it provides functionalities for reading and displaying image files on the screen.

Reasons for choosing Java Sound API:

- Java Sound API provides reading raw audio data byte by byte.
- Java Sound API supports .wav files.
- Java Sound API provides an easy extraction of the format of .wav files.

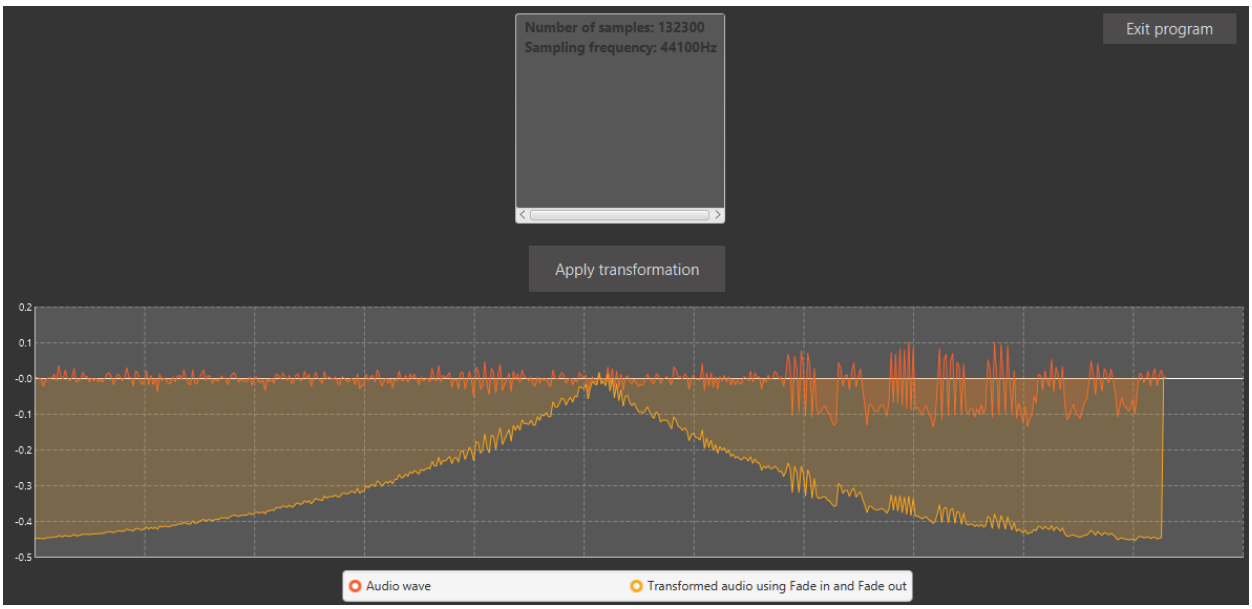
Question 1

Cowbell-mono.wav without fade-in and fade-out:

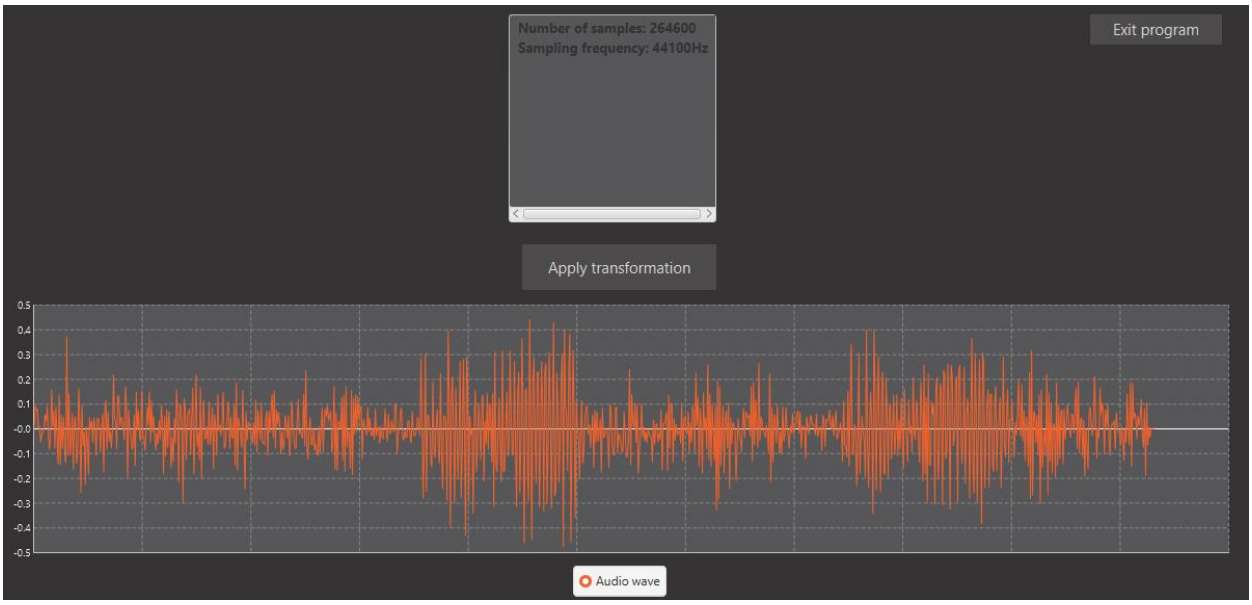


Number of samples-13230. Sampling frequency- 44100 Hz.

Cowbell-mono.wav with fade-in and fade-out:

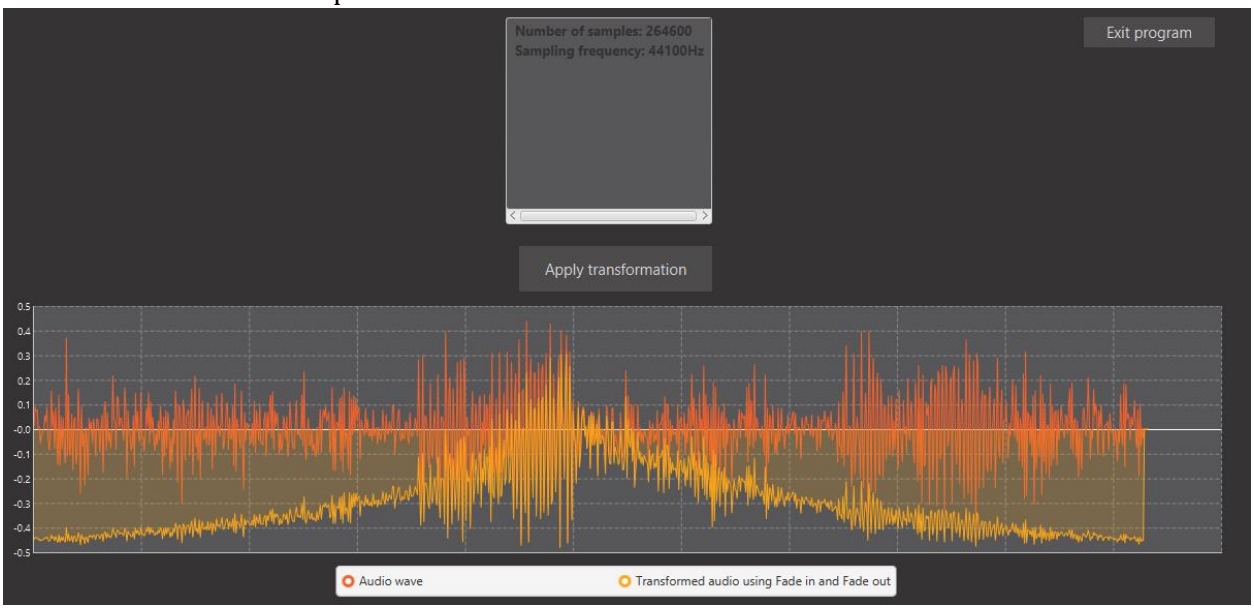


ImperialMarch-Mono.wav without fade-in and fade-out:

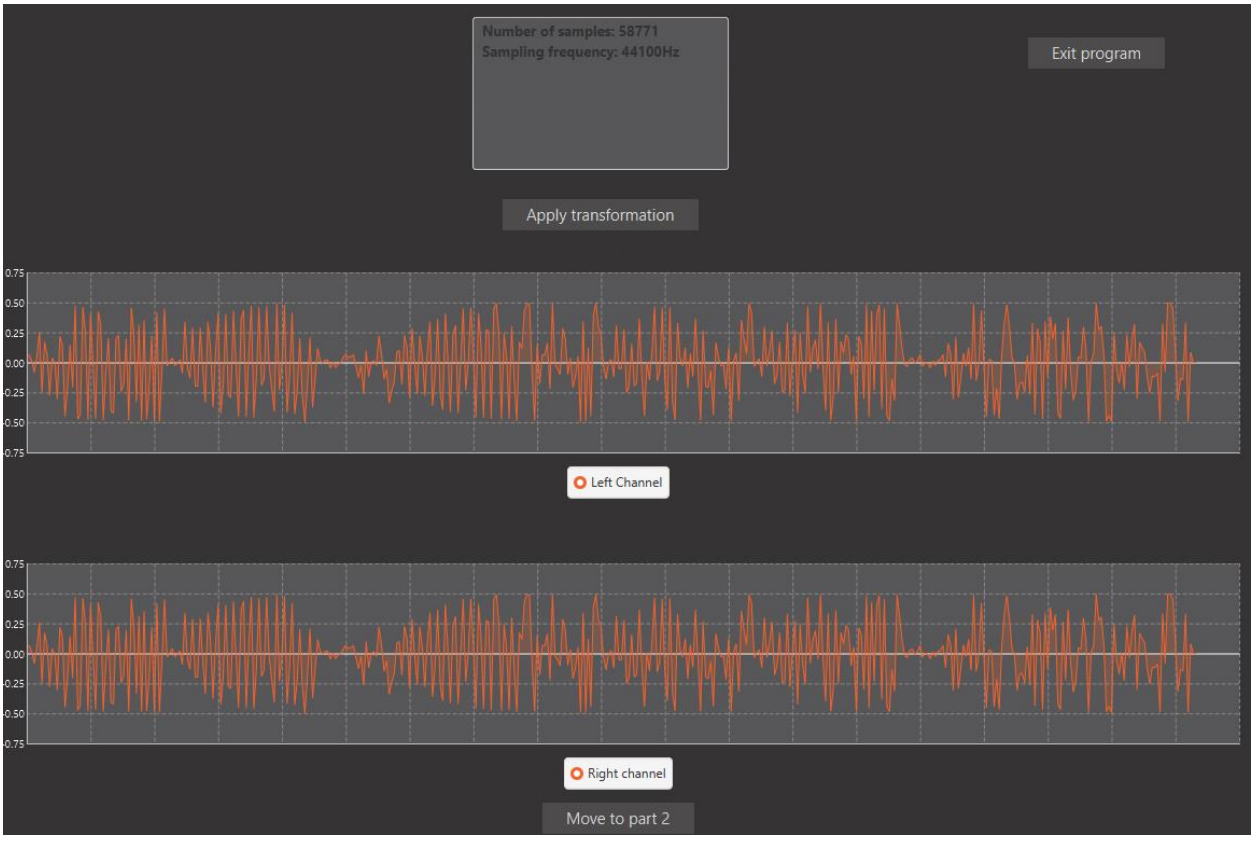


Number of samples-264600. Sampling frequency – 44100 Hz

ImperialMarch-Mono.wav with fade-in and fade-out:

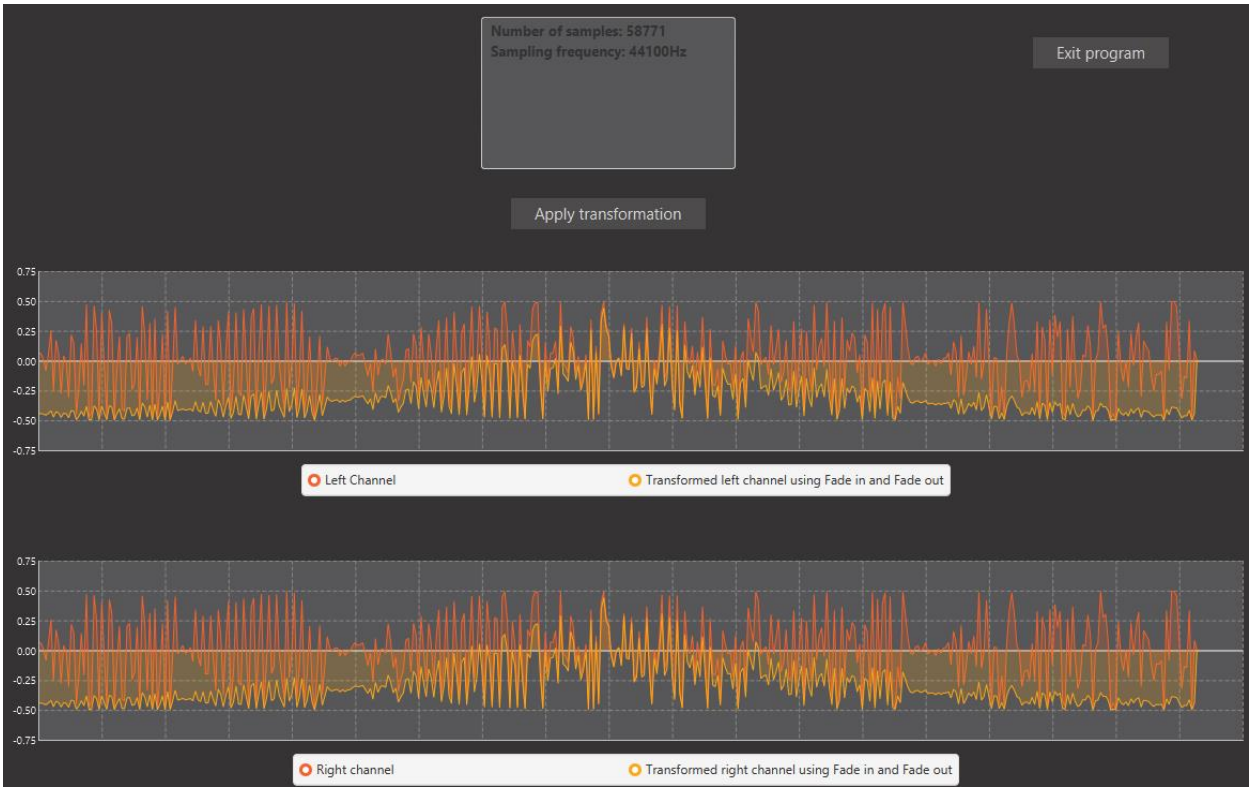


Dual channel .wave file of the choice without fade-in and fade-out:



Number of samples-58771. Sampling frequency- 44100Hz

Dual channel .wave file of the choice with fade-in and fade-out:



Explanation of fade-in and fade- out:

Individual samples were normalized to fit into the range of [-0.5,0.5].

-20 dB is the ratio of 0.1 (10%) and 0 is the ratio of 1 (100%) of the original audio sample. So, to gradually increase the first half of the sample from -20db to 0db, the algorithm creates an array of equally spaced values between [-20,0] and converts them to the ratios by the following formula:

$$\text{Ratio[i]} = 10^{\frac{the\ value\ of\ the\ array[i]}{20}}$$

Then the obtained ratios are multiplied by the original amplitudes.

To gradually decrease the second half of the samples from 0 dB to -20 dB, the inverse procedure was taken, where an array of equally spaced values between [0, -20] was created and the ratios were obtained. Then the obtained ratios were multiplied by the sample amplitudes of the second half of the samples.

Question 2

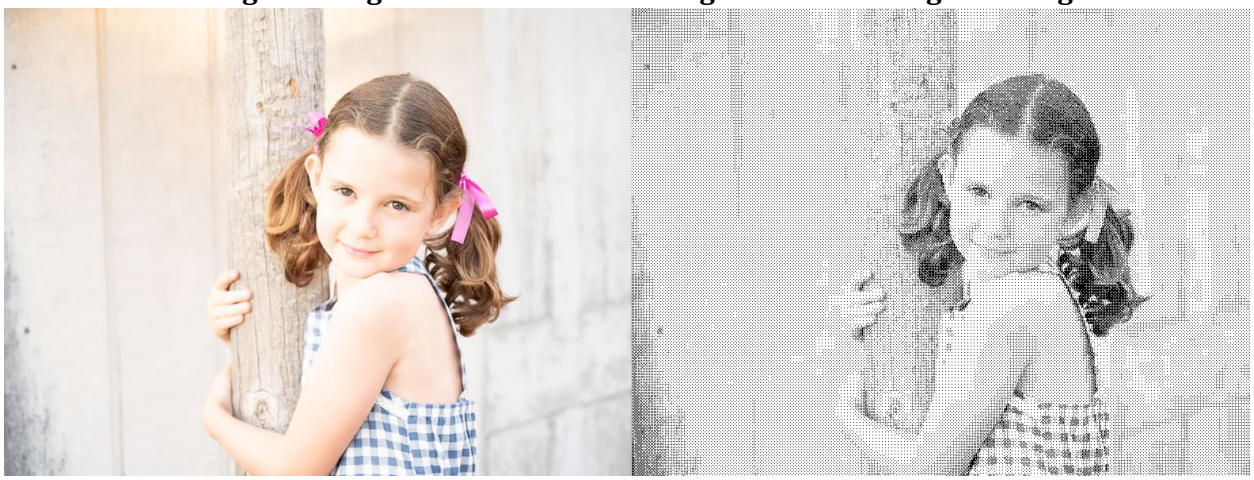
The original images and the grayscale images for the three given images

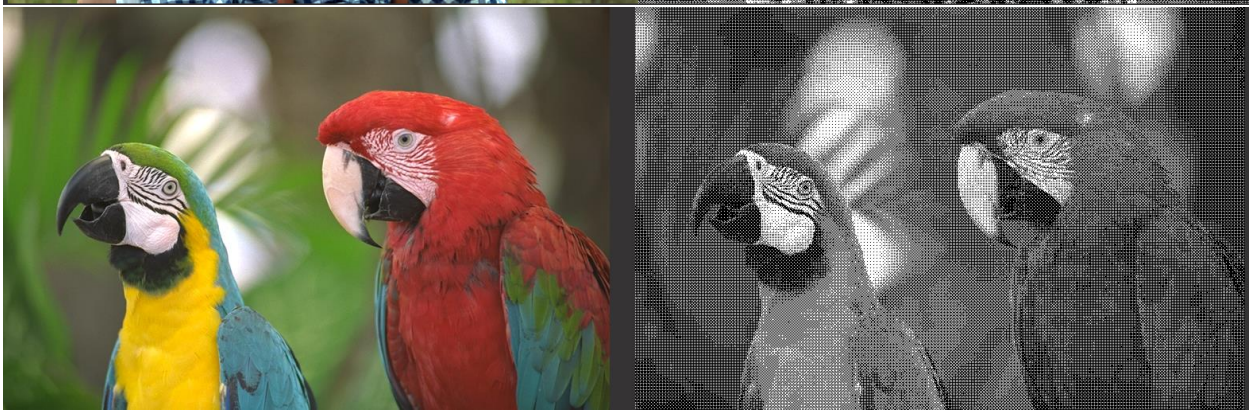
Explanation of the algorithm:

Each RGB value is extracted from the original image. Then to obtain the brightness levels from RGB, the following formula was applied:

$$Brightness = \frac{R + G + B}{3}$$

The original images and the dithered images for the three given images:





Dithering matrix used for the given images: {0,8,2,10}, {12,4,14,6}, {3,11,1,9}, {15,7,13,5}

Experiments with different dithering matrices

The first set of dithered images use a dithering matrix with values stored in increasing order from 0 to n-1 (where n is the dimension of the matrix). The experiment uses 2x2, 4x4, and 8x8 dithering matrices.

Observation:

The results show that when the dimensions increase in the dithering matrices, the dithered blocks of pixels produce a square-like pattern with sharp edges. All the matrices showed a poor distinction between similar colors and out-of-focus background elements.

The second set of dithered images use dithering matrices with close entry values separated far apart from one another. The Wikipedia's article on dithered images provides a very good instance of such matrices.

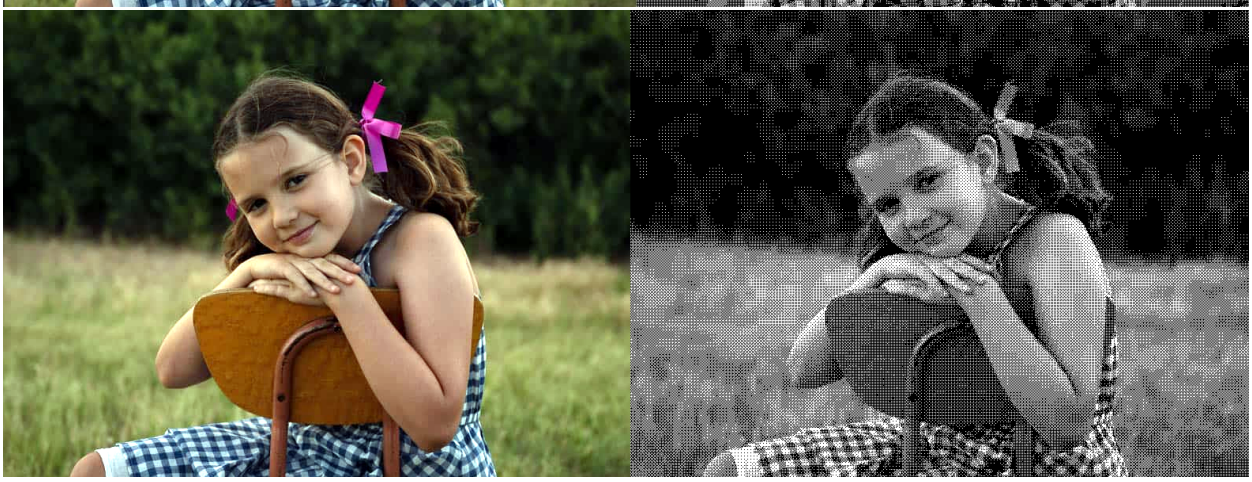
The matrices used in this experiment:

$$\begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 32 & 8 & 40 & 2 & 34 & 10 & 42 \\ 48 & 16 & 56 & 24 & 50 & 18 & 58 & 26 \\ 12 & 44 & 4 & 36 & 14 & 46 & 6 & 38 \\ 60 & 28 & 52 & 20 & 62 & 30 & 54 & 22 \\ 3 & 35 & 11 & 43 & 1 & 33 & 9 & 41 \\ 51 & 19 & 59 & 27 & 49 & 17 & 57 & 25 \\ 15 & 47 & 7 & 39 & 13 & 45 & 5 & 37 \\ 63 & 31 & 55 & 23 & 61 & 29 & 53 & 21 \end{bmatrix}$$

Processed dithered images



Observation and conclusion:

The matrices produce significantly better results than the first experiment. The dithering matrices with the dimensions of 4x4 and 8x8 produced very a similar outcome. The 4x4 matrix is chosen for the project since it uses much fewer resources and produces comparatively similar results to the 8x8 matrix.

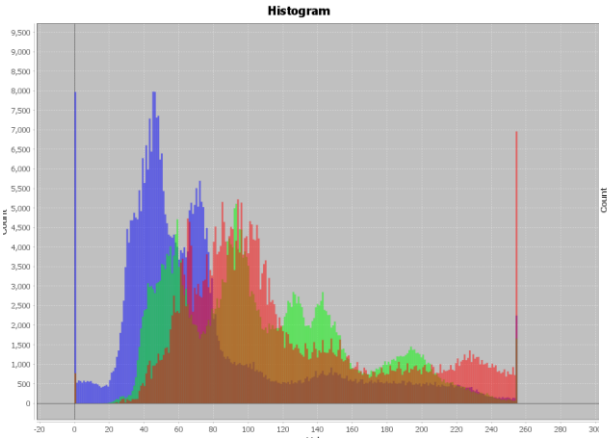
Explanation of the algorithm:

The algorithm implements the pseudo-code given in the “Representation of Images” lecture

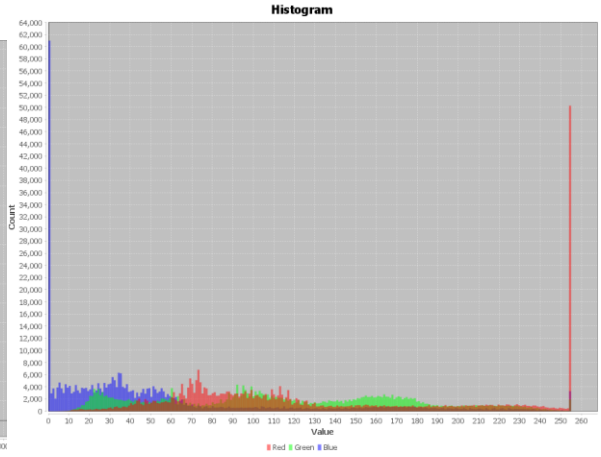
(slide 47). The algorithm is applied to the brightness values of the grayscale images. The dithering matrix was chosen from the lecture as well. Brightness levels were rescaled to fit the range of [0,n-1] of the dithering matrix, where n is the dimension of the matrix. Each time a block of pixels was chosen and compared to dithering matrix entry. If the value of the brightness was greater than the entry in the dithering matrix, then a white dot was printing for that pixel. If it was not, a black dot was printed for that pixel.

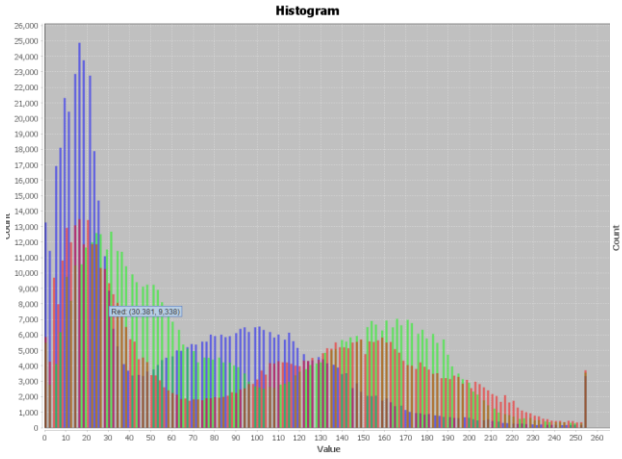
The original images and the auto leveled images

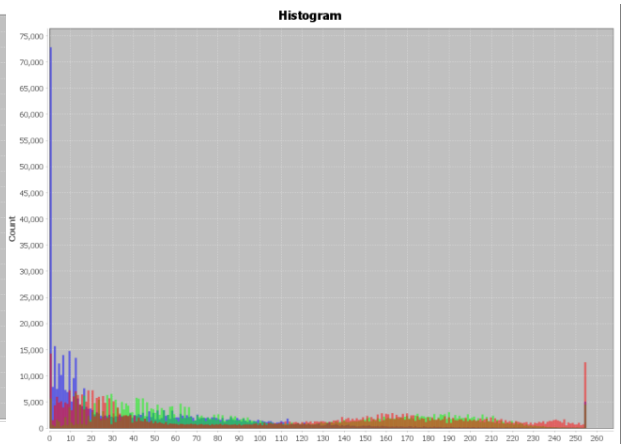
Original Images

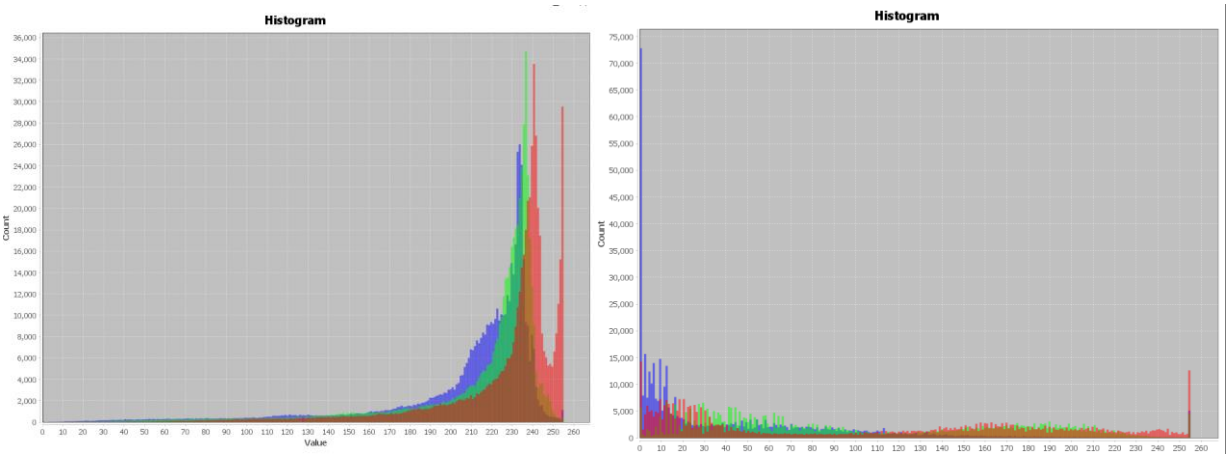


Auto-leveled Images




Histogram
Count
Value
Red (30.381, 9,338)


Histogram
Count



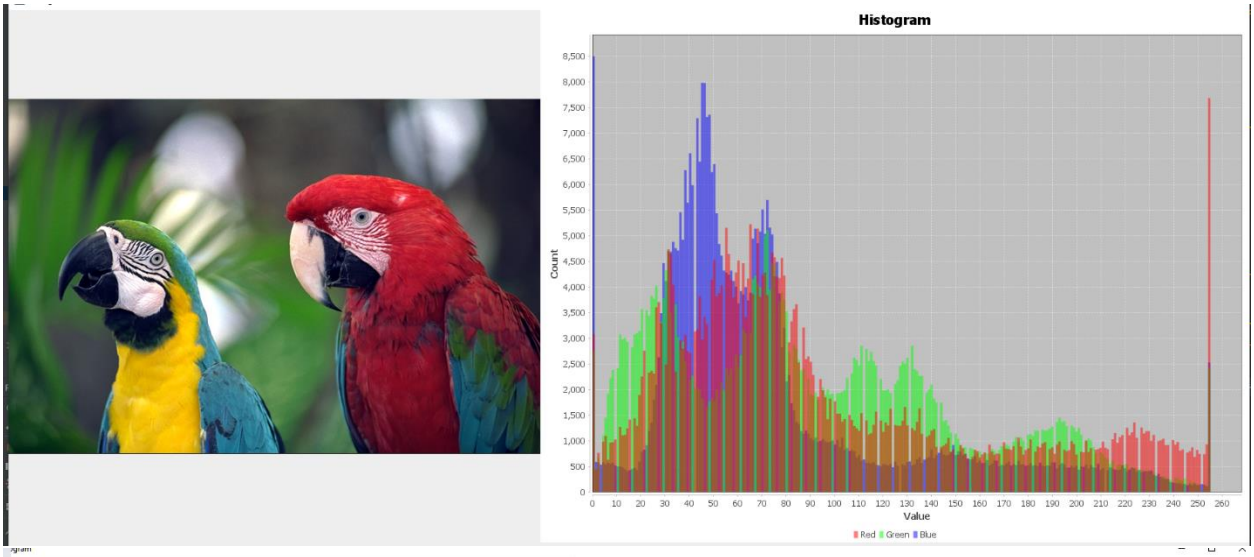
Explanation of the algorithm:

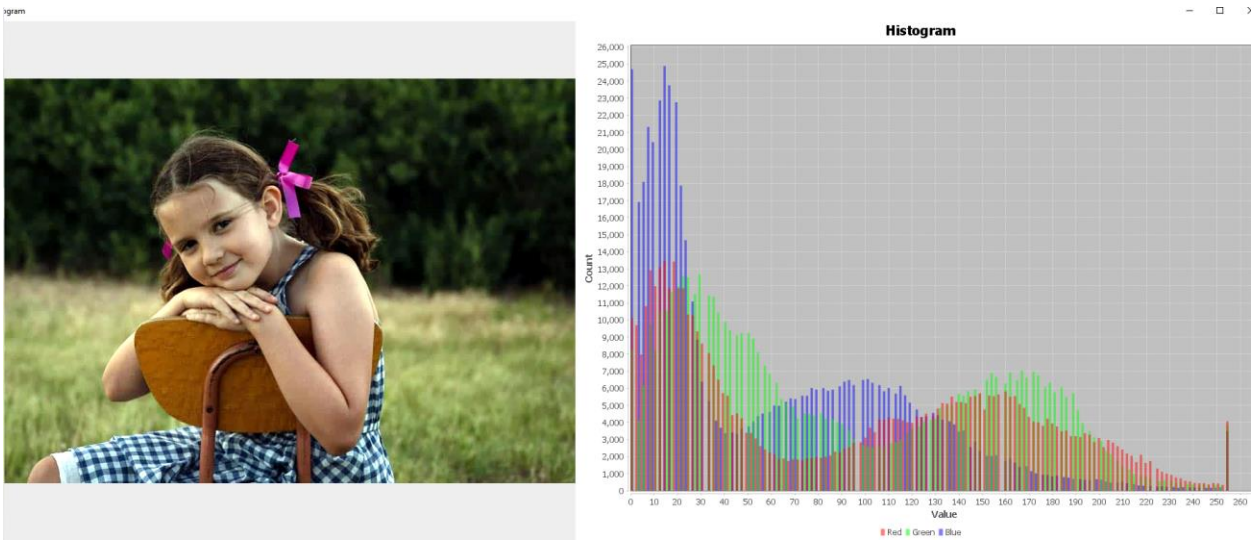
- To stretch the histogram, the original RGB channels were converted to YCoCg channels.
- The contrast of the image was calculated by subtracting the minimum color brightness by the maximum color brightness.
- To increase the contrast, the algorithm maximizes the distance between minimum color brightness and the maximum color brightness by applying the following formula:

$$Y' = \frac{Y - min\ brightness}{contrast} * 255$$

- After obtaining modified Y'CoCg values, the channels were converted back to RGB by applying an inverse function.

Gimp auto-leveled images generated by going to *layers* and applying *auto level.*





Conclusion

The images produced by GIMP auto are visually more appealing than the manually written algorithm. GIMP's auto-level function seems to be better at stretching the histograms and preserving the relative initial shape of the histograms. The auto-level function of the program seems to work best for images with a darker tone. Also, the written program seems to have a bug where it does not stretch the endpoints of the image, which would be a possible modification in the future.

Acknowledgments and sources that helped in the development of the project:

For sound API:

https://docs.oracle.com/javase/8/docs/technotes/guides/sound/programmer_guide/contents.html

https://github.com/goxr3plus/Java-Audio-Wave-Spectrum-API

For JavaFX:

Stylesheet - https://lankydan.dev/2017/01/29/javafx-graphs-look-pretty-good

http://www.cse.uaa.alaska.edu/~afkjm/csce201/handouts/LoopsImages.pdf

Theory:

https://en.wikipedia.org/wiki/Ordered_dithering

Histogram stretching - Tutorialspoint