ECE 484: Milestone 1 Report

Howard Edwards, Michael Micros, Jonathon Rigney, Megan Rowland

Abstract(Howard Edwards,Jonathon Rigne **– This milestone covers basic image processing which will be done in two variations: overlaying and histogram equalization. The image overlaying is do by adding an image selected by the user to the original input image. User-selected images maybe be words, such as subtitles, or pictures, such as a logo. Histogram equalization is used t ange the overall brightness of the picture by adjusting the contrast level. The purpose this milestone is to create two command line programs that loads 8 bit grayscale images for processing. Each program will be able to cess an image and produce a new image that should visibly differ from the original. The programs will create files displaying the end result of the processed image. The image processing will be done in two variations: overlaying and histogram equalization. Overlaying of a pict involves modification of the picture with another picture or caption. This will be implemented in such a way that the original and the overlaying picture pixel will differ in contrast. Histogram equalization involves comparing the changes done to the contrast or brightness of a picture.**

Index Terms **— Gray-scale, Image analysis, Image processing**

I. Introduction(Jonathon Rigney)

The goal of this report is to follow the process of the first milestone for an integrated system capable of real time video processing which is controlled by a user interface on PC and is based on an FPGA. The two main functions of video processing are divided into overlaying each frame of the video with user selected images or subtitles, and adding features that allow the user to adjust image quality with brightness and contrast.

The current module focuses on histogram equalization and image overlay of still, grayscale, bitmap images. This will build the foundation of the project by allowing us to perform these operations on one frame then replicating it for each subsequent frame in the video. The image overlay will be used to add features like company logos or subtitles for different languages. The histogram equalization is used to increase contrast to an image by ensuring that each level of pixel brightness occurs approximately as often as the next level while still maintaining the relative brightness(meaning that if one pixel was originally brighter than another this will remain true after equalization).

II. Design (Megan Rowland)

A. Summary of Design– The overall design consists of an Image class to input and output a BMP file, a function to compute histogram equalization of an image and a function to overlay two images. The image class can be seen in Fig.3 of the Appendix

B. Detail Description– Executing the command line program will first prompt for the filename of an 8-bit BMP grayscale image file.. This image will be used for the original image. Next, there will be a prompt to enter the filename of an overlay image that is the same size as the original image.

Example runs of the command line program for both test1.bmp and test2.bmp can be seen in Fig. 1 and Fig. 2.

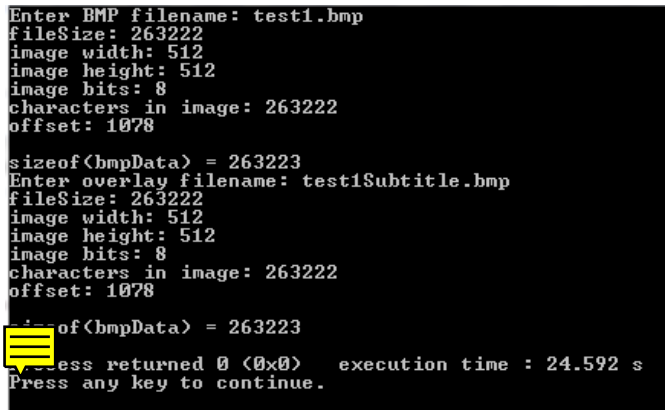

Figure 1: Sample run of "test1.bmp"

First, from main(), the first 8-bit grayscale image file is read into, with the function readBM(), an instance of the I class named original. The BM function initia the data types for that image. The data types include the width of the image (imageWidth), the height of the image (imageHeight), the amount of bits for the image (imageBits), the offset between the header and the pixels of the image (offset), a vector of all of the bytes extracted from the image (bmpData), a integer array of the bytes of the header (header[]), and a two-dimensional vector of all of the pixel bytes (pixels). If either image file

does not exist or cannot be read, the program will terminate at this point. The second 8-bit grayscale image file name is saved in the variable filename.

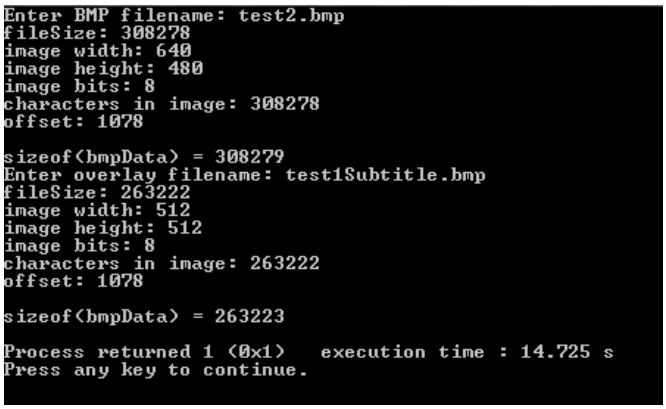

Figure 2: Sample run of "test2.bmp"

Next, from main(), the overlay() function is called on the original image with the overlay filename as the parameter. Within the overlay() function, an instance of the Image class, foreground, is created for the overlay BMP file. The readBM() function is called on the foreground Image using the filename as the parameter. This initializes all of the data types for the image to overlay (foreground) over the original. The twoImageSameDimension() function is called to return turn if the image width and image height are the same. If these two values are not the same, the function will cease to move on and exit. The overlay() function then iterates through the two-dimensional vector of pixels of the foreground and original Image. Anywhere there is a value of 0 (black) in the foreground Image pixels, the original Image pixel is replaced with 255 (white). The overlaid image is then printed to a new file, out1.bmp, with the original image overlaid by the foreground image, changing the foreground image from black to white.

Next, from main(), the histogramEqualization() function is called on the original image. In the histogramEqualization() function, an array, hist[], is created to count the total number of pixels associated with each pixel intensity (0 to 255) of the original Image. An array, cumulative[], is created to calculate the cumulative histogram. The pixel values are then adjusted by the histogram equalization algorithm. Histogram equalization is calculated by the following equation: round (255 * (cumulative histogram / total number of pixels)). The outputBM() function is called and creates a new BMP file, out2.bmp, with the pixels adjusted by histogram equalization. The results of all functions will be displayed in the Evaluation section.

III. Evaluation (Megan Rowland / Michael Micros)

The program functions clearly perform as intended, yielding the expected results. For the overlay function, the user is prompted to enter the filename of the original image and then is asked to enter the filename of the image to be overlaid on the original. The results of the overlay() function called on the test images overlaid with the overlay images are displayed in Fig. 4 and Fig.5 of the Appendix.

As can be seen, the subtitle images are overlaid on top of the originals without altering any of the pixels except those that are needed. The results from the histogram equalization for both test1.bmp and test2.bmp are displayed in Fig of the Appendix. Finally the histograms for the original and equalized images, as well as the cumulative histogram can be seen in Fig. 6 and Fig.7 of the Appendix.

IV. Discussion (Michael Micros)

Based on the results presented in the Evaluation the overlay function works exactly as intended, without much room for improvement. The only scenario in which a problem may arise is when it is required to overlay an image that is very bright or completely white in the area that the overlaid image will affect. A situation like this could make difficult to make out the subtitles or overlaid image. Such a problem can be easily solved by adding an outline to the overlay image (around the subtitles for our particular implementation) of a different color. For example, if in the overlay image a pixel value of 255 is detected, the pixel from the original image will appear in the output image. If a pixel value of 0 is detected, a white pixel will appear in the output image. Finally, if a pixel value of 128 (an arbitrary value we select for the outline of the text) is detected, a black pixel will appear in the output image. This will provide a black outline to the white overlay that we have chosen.

As for the histogram equalization, even though the resulting images seem to provide a more equal distribution of color it is obvious that better results could be achieved by implementing a more complex equalization technique. By comparing the histograms of the original and equalized image it is important to note that when the number of pixels is very small for colors that are very similar, those pixels may end up having the same color in the equalized image. This does not exactly violate any of the restrictions that were imposed, and sometimes may be desirable. Another observation that can be made is that when performing histogram equalization, in order to improve the contrast in a large area we sacrifice the contrast in a smaller area of the image. An example of this can be seen in the bottom right corner of test2.bmp and its equalized counterpart.

V. Appendix

```
class Image{
    private:
        int imageWidth ;
        int imageHeight ;
        int imageBits ;
        int offset ;
         vector< unsigned int> bmpData;          // Contains all the bytes extracted from the bitmap
         vector< vector <unsigned int>> pixels; // pixel bytes
         int header[54];                        // Header bytes
         vector< vector <unsigned int> > pixelsOverlaid;

    public:
        bool twoImageSameDimension(Image img);
        void outputBM(char  filename[], vector< vector <unsigned int> > pixelsMulitpleImages);

         int getImageWidth();
         int getImageHeight();
         int getImageBits();
         int getOffset();

         vector< unsigned int> getBmpData();
         vector<vector <unsigned int> > getPixels();

         void readBM (char  filename[]);
         void outputBM(char  filename[]);
         void histogramEqualization();
         void overlay(char  filename[]);
};
```

Figure 3: Image class

Figure 4: Test1.bmp with overlaid subtitles

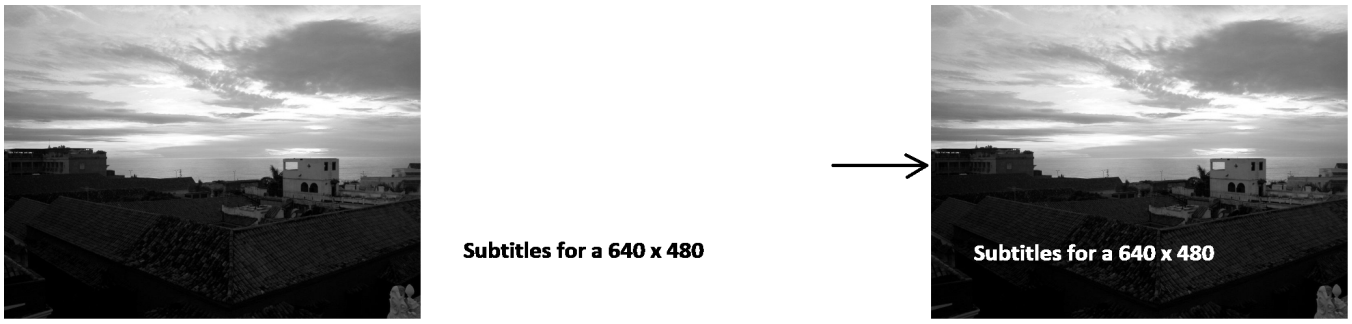

Figure 5: Test2.bmp with overlaid subtitles

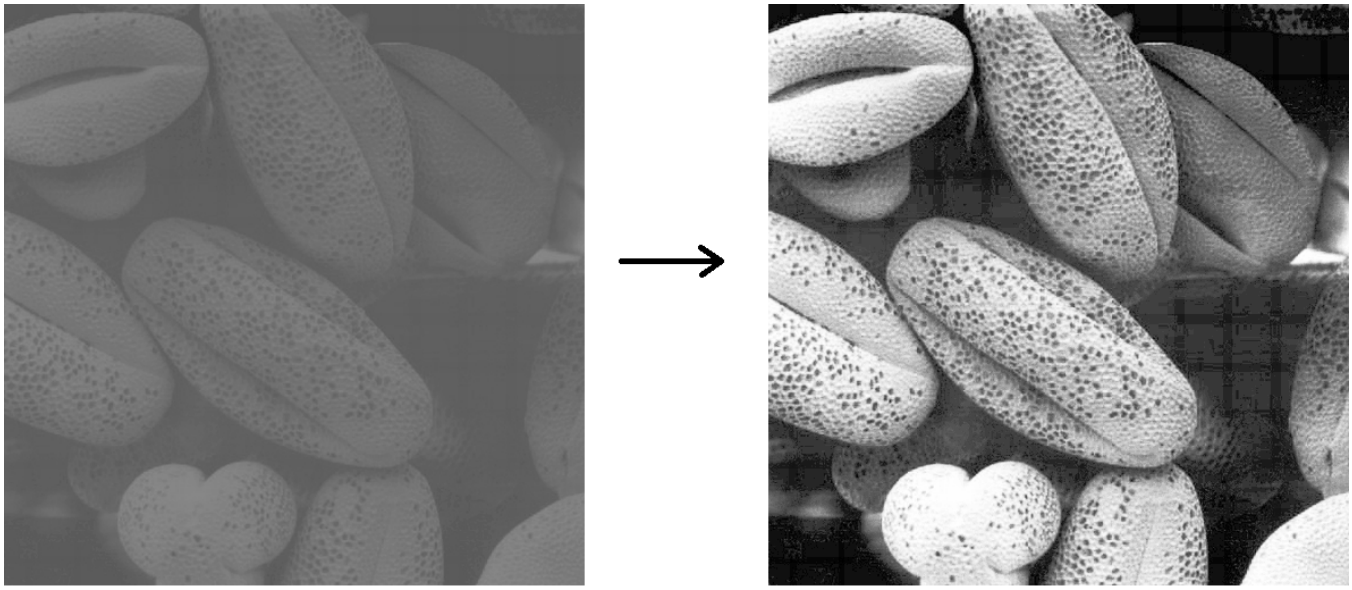

Figure 6: Test1.bmp before and after equalization

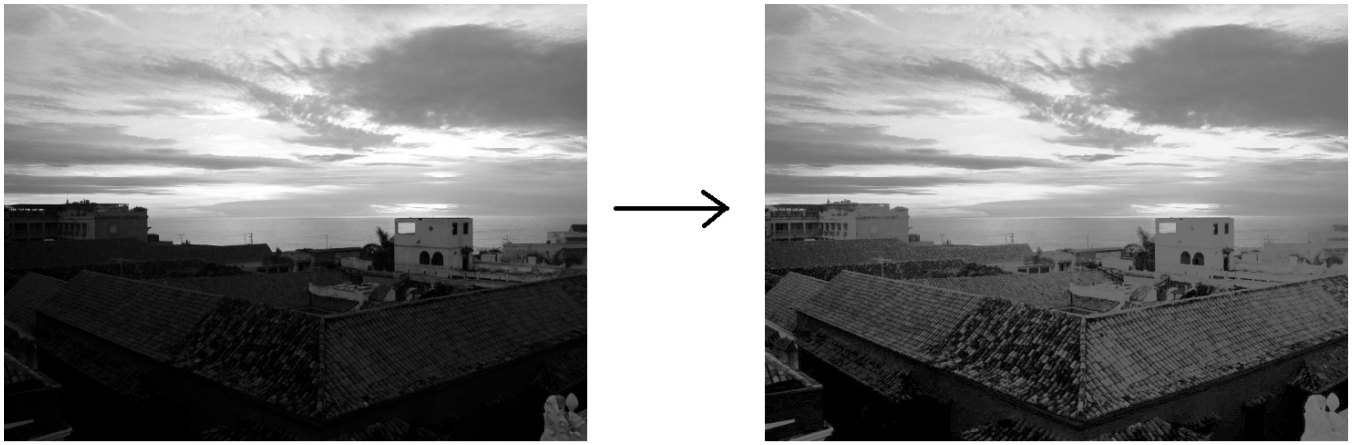

Figure 7: Test2.bmp before and after equalization

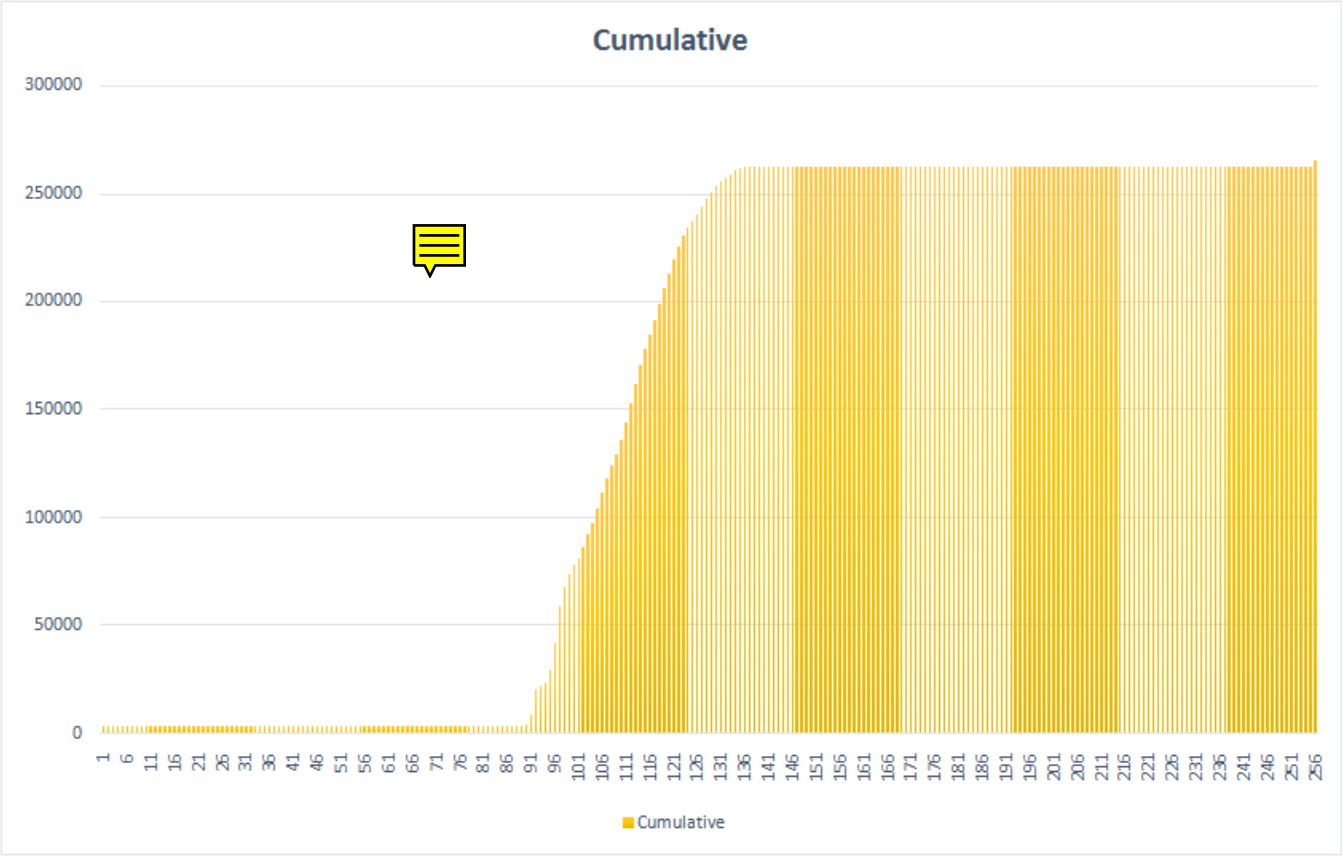

Figure 8: Cumulative histogram of test1.bmp

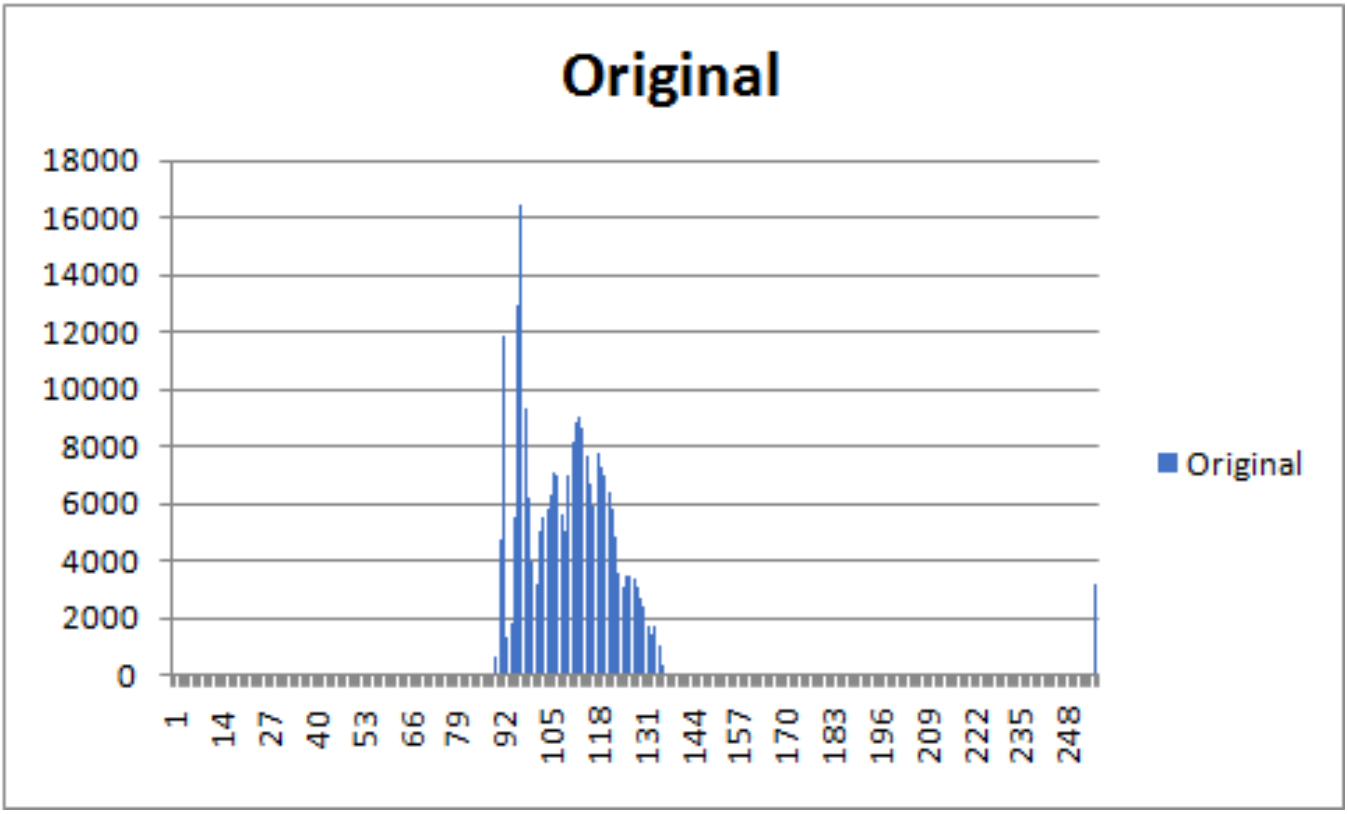

Figure 9: Histogram of original test1.bmp

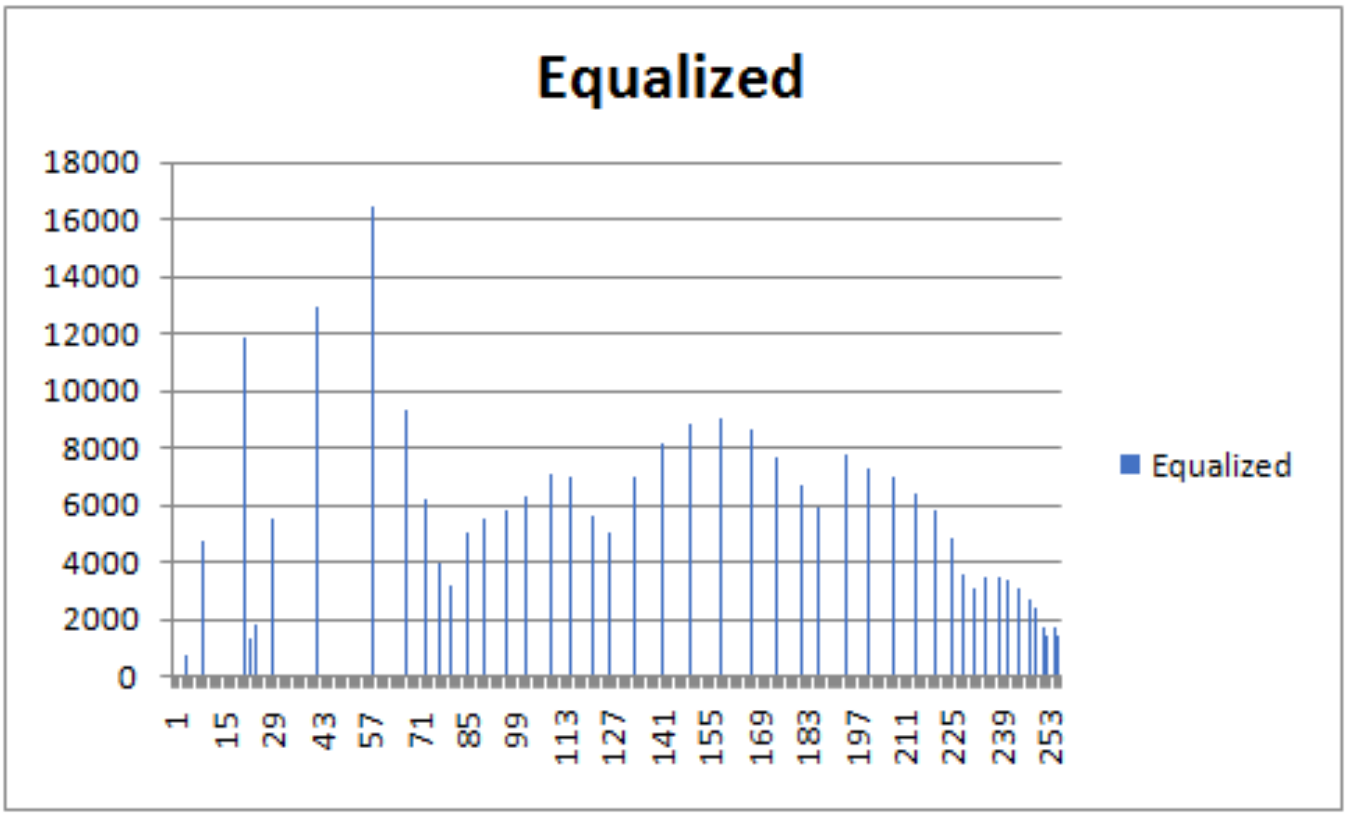

Figure 10: Histogram of equalized test1.bmp