Digital Image Processing
Assignment 1
Thananon “Arm” Patinyasakdikul

1. **Read the image from file, divide intensity by 2 and write image to file.**

   This assignment has been implemented with C code. Image reading and writing is from libpng and will work only for 8 bit, grayscale images. The code will read the image and store pixel data into 2D array with the intensity value for each pixel and then get manipulated accordingly. The source code is provided in the appendix.

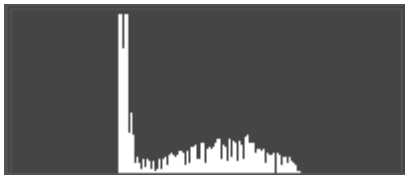
Original Image

Intensity Halved

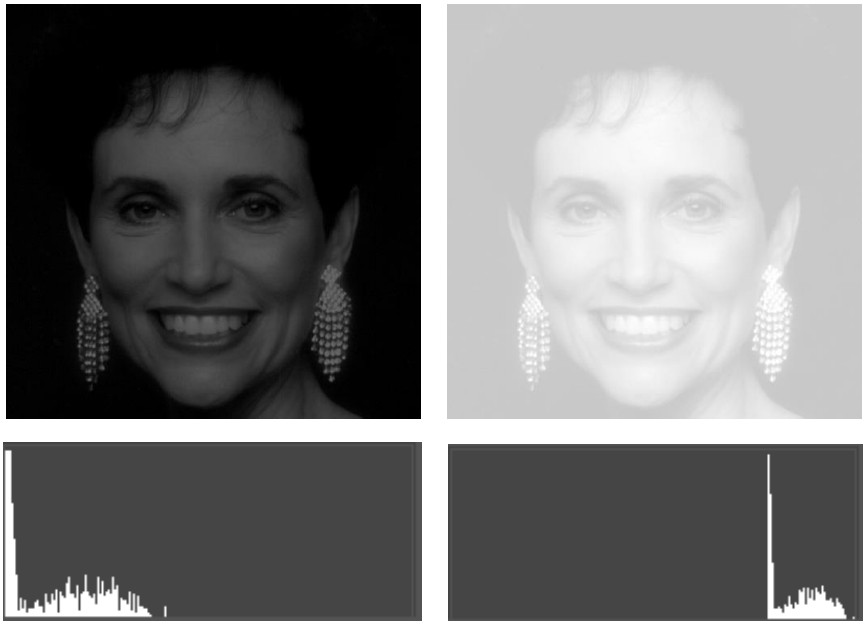
2. **Image Scaling**

   Original image (woman.png) has the intensity range of 0 – 255. For this problem, we will rescale image into multiple ranges (A,B).

From 0 – 255 (Original)



From 70 – 200



From 0 – 100 *From 200 – 255*

The intensity value of each pixel is calculated by the formula,

$$x' = \left\{ \frac{x - \min(x)}{\max(x) - \min(x)} \, (B - A) \right\} + \mathrm{A}$$

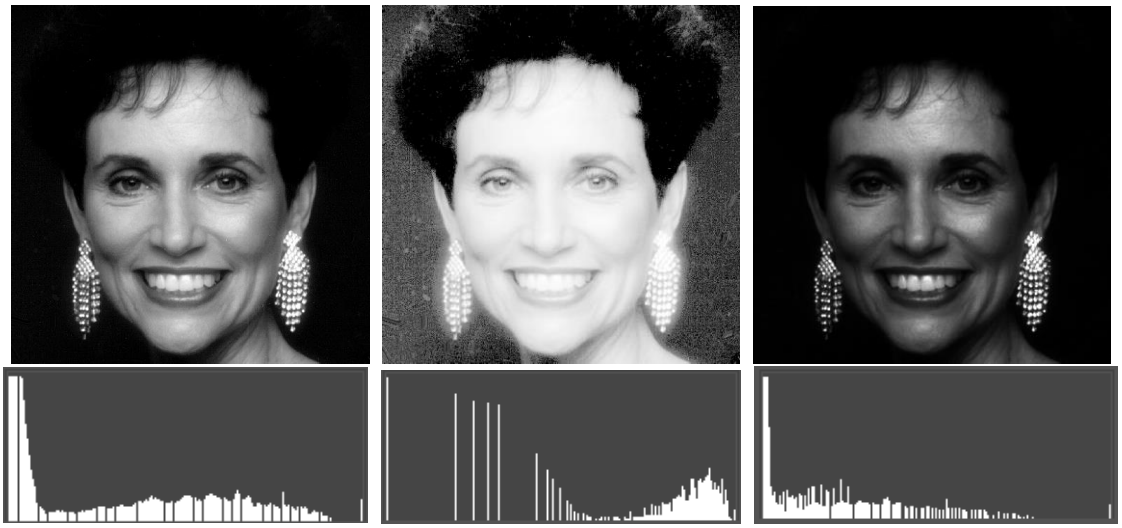
$B = new\ maximum$
$A = new\ minimum$

We can see that if the range (A,B) is short, the result image will have low contrast and will be very difficult to get the detail out of it.

3. Negative image, log and exp.

Original Image *Negative Image*

Negative image is obtained by the formula $x' = 255 - x$. Negative image sometime show the detail that we are not able to see in the original image. In this case, we can see the outline of the cheek, the detail around eyes area and wrinkles better.



Original Image *Log Transformation* *Exp Transformation*

For log transformation, the formula $x' = c\ln(1 + x)$ is applied. Constant c for this assignment has been set to 39. The result then gets rescaled to have the range of 0 – 255. With the transformation, the result image is brighter and we can clearly see the background isolated from the woman's hair but at the same time, the face's detail is vanished.

With exponential transformation, the formula $x' = c \times (b^x - 1)$ is applied. The -1 term is to prevent the case of x = 0, then x' = c. For this assignment b = 1.01 and c = 20 and the result then gets rescaled into the range of 0 – 255. The result image appears to be darker as expected from the transformation.

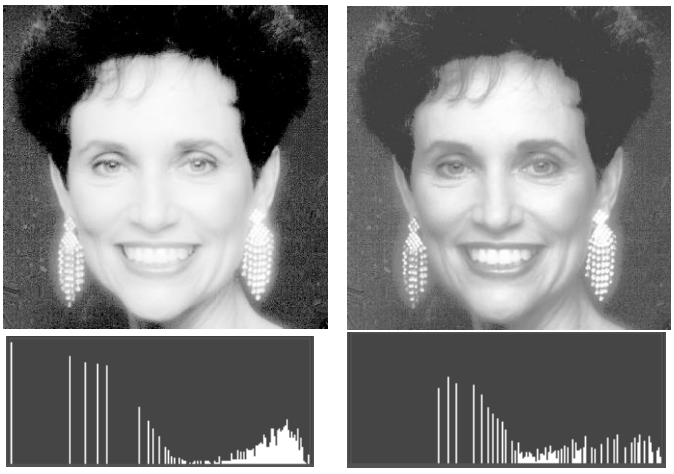
4. Power law and effect of gamma

$\gamma = 0.5$ $\gamma = 0.8$ $\gamma = 1.1$

$\gamma = 1.3$ $\gamma = 2.0$ $\gamma = 2.5$

For power law, the intensity is obtained by the formula $x' = c \times x^{\gamma}$. For this assignment, c = 1.0 is used and the result gets rescaled into the range of 0 – 255. The image appears to be brighter for the lower gamma and gets darker as gamma is higher.

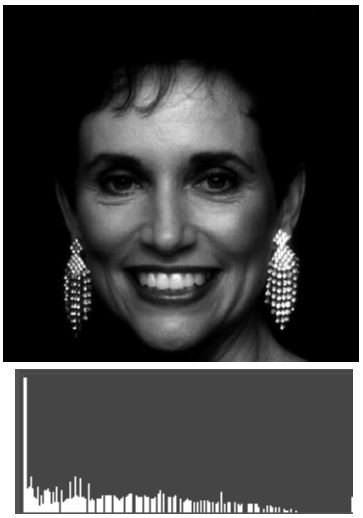
5. Histogram equalization



Log Transformation *Histogram Equalized*

Exponential Transformation

Histogram Equalized



Power Transformation $\gamma = 2$

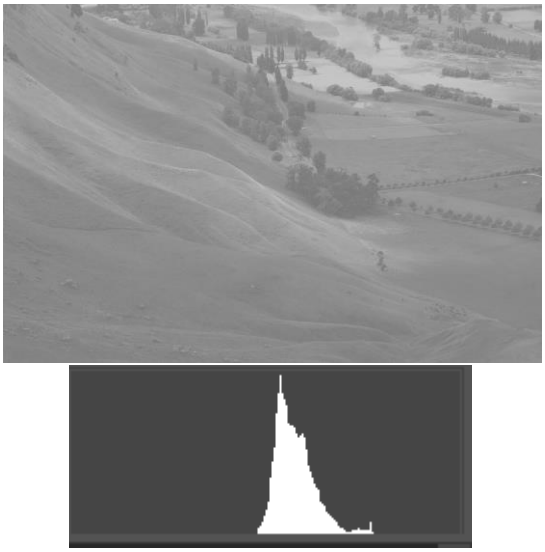
Histogram Equalized

For this problem, the intensity has to be calculated to equalize the histogram by the formula,

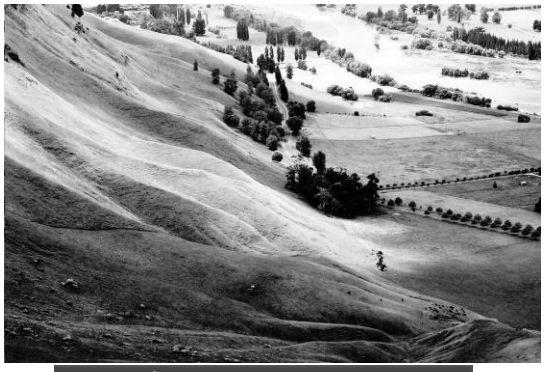
$$T(x) = 255 \times \sum_{i=0}^{x} pdf(x)$$

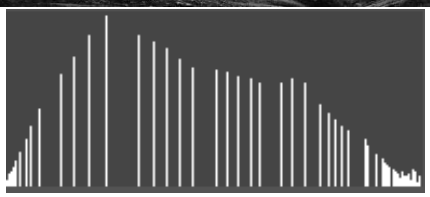
$$X' = T(x) \ , pdf = probability\ density\ function$$

The result of histogram equalization is very impressive as we can see the improvement for every scenario. The detail is much more visible to the eyes after applied. However, the image "woman.png" is not the best showcase to do histogram equalization because the image is mostly dark which means the majority of the intensity is low. To illustrated the better case of image equalizer, please see the figure below.



Original Image





Histogram Equalized

For this image, we can easily see the improvement from histogram equalization. More information is presented as we have more spread out histogram and able to represent more data.

Conclusion

For this assignment, we have hands on experience on manipulating the image intensity to achieve a better image overall. The basic functions like log will make the image brighter, exponential function will make the image darker. The power law is used to adjust the brightness of the image depends on the gamma. All of the transformation needed to be rescaled into the range of 0 – 255 to be able to write the transformed intensity into an 8 bit image.

Histogram equalization is a very good tool to push out the detail in an image. By wider the range and spread the intensity range out, some detail that we might miss on the original image might be able to clearly see after the equalization.

Source Code :

Process.h

```
#include<stdio.h>
#include <stdlib.h>
#include "png.h"
#include <math.h>

png_structp png_ptr;
png_infop info_ptr;
png_byte color_type, bit_depth;

int max = 0;
int min = 255;
int hist[255];
int **tmp_free;
float csum[255];
int readmode;
png_bytep *row_ptrs;

#define MALLOC_OUTPUT_INT(out, width, height)                                   \
                do {                                                            \
                        (out) = (int**)malloc(sizeof(int*) * (height));         \
                        for(int i=0;i<(height);i++)                             \
                                (out)[i] = (int*) malloc(sizeof(int) * (width));\
                }                                                               \
                while(0)


#define MALLOC_OUTPUT_FLOAT(out, width, height)                                 \
                do {                                                            \
                        (out) = (float**)malloc(sizeof(float*) * (height));     \
                        for(int i=0;i<(height);i++)                             \
                                (out)[i] = (float*) malloc(sizeof(float) * (width));\
                }                                                               \
                while(0)

#define FREE_PIXEL(out)                                                         \
                do {                                                            \
                        tmp_free = (out);                                       \
                        (out) = NULL;                                           \
                        free(tmp_free);                                         \
                }                                                               \
                while(0)
int **readPNG(char *filename,int* width, int* height){

                FILE *fp = fopen(filename,"rb");
                char header[8];
                fread(header, 1, 8, fp);
                if(png_sig_cmp((png_const_bytep)header,0 ,8)) {
                        printf("not png file\n");
                        exit(-1);
                }

                png_ptr = png_create_read_struct(PNG_LIBPNG_VER_STRING,NULL, NULL,NULL);
                if (!png_ptr) printf("fail create");
                info_ptr = png_create_info_struct(png_ptr);
                setjmp(png_jmpbuf(png_ptr));
                png_init_io(png_ptr, fp);
                png_set_sig_bytes(png_ptr, 8);
                png_read_info(png_ptr, info_ptr);

                *width = png_get_image_width(png_ptr, info_ptr);
                *height = png_get_image_height(png_ptr, info_ptr);
                color_type = png_get_color_type(png_ptr, info_ptr);
                bit_depth = png_get_bit_depth(png_ptr, info_ptr);


                setjmp(png_jmpbuf(png_ptr));
                row_ptrs = (png_bytep*) malloc(sizeof(png_bytep) * (*height));
                for(int y=0;y<(*height); y++){
                        row_ptrs[y] = (png_bytep) malloc (png_get_rowbytes(png_ptr, info_ptr));
                }
                png_read_image(png_ptr,row_ptrs);
                png_read_end(png_ptr,info_ptr);
                printf("\n\n\n=== Original Photo Information ================================\n");
                printf("File Name : %s\n",filename);
                printf("%d x %d Pixels\nBit Depth : %d\nColor Type : ",*width,*height,bit_depth);
                fclose(fp);
                int colot_type = png_get_color_type(png_ptr, info_ptr);
                switch(color_type){
                        case PNG_COLOR_TYPE_RGB:
                             printf("RGB\n");
                             readmode = 3;
                             break;
                        case PNG_COLOR_TYPE_RGBA:
                                printf("RGB ALPHA\n");
                                readmode = 4;
                                break;
                        case PNG_COLOR_TYPE_GRAY:
                                printf("GRAYSCALE\n");
                                readmode = 1;
                                break;
                        case PNG_COLOR_TYPE_GRAY_ALPHA:
                                printf("GRAYSCALE ALPHA\n");
                                readmode = 2;
                                break;
                        case PNG_COLOR_TYPE_PALETTE:
                                printf("PALETTE\n");
                                readmode = 1;
                                break;
                        default :
```

```
                                        printf("OTHERS\n");
                                        break;
                }
                printf("Row bytes : %d\n",(int)png_get_rowbytes(png_ptr,info_ptr));
                int **pixel;
                pixel = (int**) malloc (sizeof(int*) * (*height));
                if(NULL == pixel) printf("NULL\n");
                for(int i=0;i< (*height);i++){
                        pixel[i] = (int*) malloc (sizeof(int) * (*width));
                        if(NULL == pixel[i]) printf("NULL\n");
                }

                for(int y = 0; y < (*height); y++){
                        png_byte *row = row_ptrs[y];
                        for( int x = 0; x< (*width);x++){
                                png_byte *ptr = &(row[x*readmode]);
                                pixel[y][x] = (int)ptr[0];
                                if( (int)ptr[0] > max ) max = (int) ptr[0];
                                if( (int)ptr[0] < min ) min = (int) ptr[0];
                                hist[pixel[y][x]]++;
                        }
                        //printf("\n");
                }
                for(int i=0;i<256;i++){
                        for(int j=0;j<=i;j++)
                                csum[i] += hist[j];
                        csum[i] /= ((*width)*(*height));
                }
                printf("Max Intensity : %d\n",max);
                printf("Min Intensity : %d\n",min);
                printf("================================================================\n\n");
                free(row_ptrs);
                return pixel;

}

void printHist(int **pixel,int width,int height){

                int lHist[256];
                float csum[256];
                for(int i=0;i<256;i++) lHist[i] = csum[i] = 0;
                for(int i=0;i<height;i++){
                        for(int j=0;j<width;j++){
                                lHist[pixel[i][j]]++;
                        }
                }
                for(int i=0;i<256;i++) {
                        for(int j=0;j<=i;j++) csum[i]+=lHist[j];
                }

                for(int i=0;i<256;i++) printf("%d\t\t%.2f\n",i,csum[i]/(width*height));
}

void printPixel(int **pixel,int width,int height){
                for(int i=0;i<height;i++){
                        for(int j=0;j<width;j++){
                                printf("%d ",pixel[i][j]);
                        }
                        printf("\n");
                }
}

void writePNG(char *filename,int **pixel,int width,int height){

                png_bytep *row_ptrs;
                row_ptrs = (png_bytep*) malloc(sizeof(png_bytep) * height);
                for(int y=0;y<height; y++){
                        //row_ptrs[y] = (png_bytep*) malloc (sizeof(png_bytep) * width);
                        row_ptrs[y] = (png_bytep) malloc (width * sizeof(png_byte));
                }

                // write pixel array to row pointers
                for(int y = 0;y<height;y++){
                        for(int x = 0;x<width; x++){
                                int z = x;
                                row_ptrs[y][x]   = pixel[y][x];
                                //row_ptrs[y][z+1]   = 255;
                        }
                }

                FILE *fp = fopen(filename,"wb");
                if(!fp)
                        exit(-1);

                png_structp wpng_ptr = png_create_write_struct(PNG_LIBPNG_VER_STRING,NULL,NULL,NULL);
                if(!wpng_ptr)
                        exit(-1);



                png_infop winfo_ptr = png_create_info_struct(wpng_ptr);
                setjmp(png_jmpbuf(wpng_ptr));
                png_init_io(wpng_ptr, fp);

                setjmp(png_jmpbuf(wpng_ptr));
                png_set_IHDR(wpng_ptr, winfo_ptr, width, height,
                                        bit_depth, PNG_COLOR_TYPE_GRAY, PNG_INTERLACE_NONE,
                                        PNG_COMPRESSION_TYPE_BASE, PNG_FILTER_TYPE_BASE);
                png_write_info(wpng_ptr, winfo_ptr);


                setjmp(png_jmpbuf(wpng_ptr));
                png_write_image(wpng_ptr, row_ptrs);

                setjmp(png_jmpbuf(wpng_ptr));
                png_write_end(wpng_ptr, NULL);

                fclose(fp);
```

```
		free(row_ptrs);

}

int **IRescale(int **pixel, int width, int height,int min,int max,int tomin,int tomax){

		int **out;
		MALLOC_OUTPUT_INT(out, width, height);
		for(int y=0;y<height;y++){
			for(int x =0;x<width;x++){
				out[y][x] = (int)((float)((pixel[y][x] - min)) / (max-min) * (tomax-tomin))+tomin;
			}
		}
		return out;

}


int **Normalize(float **pixel, int width, int height){

		int **out;
		float min = pixel[0][0];
		float max = pixel[0][0];
		MALLOC_OUTPUT_INT(out, width, height);
		for(int y=0;y<height;y++){
			for(int x =0;x<width;x++){
				if(pixel[y][x] < min) min = pixel[y][x];
				if(pixel[y][x] > max) max = pixel[y][x];
			}
		}
		printf("mix = %f\nmax=%f\n",min,max);
		// Scale them up
		for(int y=0;y<height;y++){
			for(int x =0;x<width;x++){
				out[y][x] = (int)((float)((pixel[y][x] - min)) / (max-min) * 255);
			}
		}

		return out;

}

int **ParseInt(float **pixel, int width, int height){

		int **out;
		float min = pixel[0][0];
		float max = pixel[0][0];
		MALLOC_OUTPUT_INT(out, width, height);
		for(int y=0;y<height;y++){
			for(int x =0;x<width;x++){
				out[y][x] = (int)pixel[y][x];
			}
		}

		return out;

}


int **histEQ(int **pixel, int width, int height){

		int T[256];
		int **out;
		MALLOC_OUTPUT_INT(out, width, height);

		for(int i=0;i<256;i++){
			T[i] = 255 * (csum[i]);
		}

		for(int y=0;y<height;y++){
			for(int x=0;x<width;x++){
				out[y][x] = T[pixel[y][x]];
			}
		}
		return out;
}

int **halfIntensity(int **pixel, int width, int height){

		int **out;
		MALLOC_OUTPUT_INT(out, width, height);

		for(int y=0;y<height;y++){
			for(int x=0;x<width;x++){
				out[y][x] = pixel[y][x]/2;
			}
		}
		return out;
}


int **getNegative(int **pixel, int width, int height){

		int **out;
		MALLOC_OUTPUT_INT(out, width, height);
		for(int y=0;y<height;y++){
			for(int x=0;x<width;x++){
				out[y][x] = 255-pixel[y][x];
			}
		}
		return out;
}

float **getLog(int **pixel, int width, int height,float c){

		float **out;
		MALLOC_OUTPUT_FLOAT(out, width, height);
```

```
	for(int y=0;y<height;y++){
		for(int x=0;x<width;x++){
			out[y][x] = c * log(1+pixel[y][x]);
		}
	}
	return out;
}

float **getExp(int **pixel, int width, int height,float c,float b){

	float **out;
	MALLOC_OUTPUT_FLOAT(out, width, height);
	for(int y=0;y<height;y++){
		for(int x=0;x<width;x++){
			out[y][x] = c * (pow(b,pixel[y][x])-1);
		}
	}
	return out;
}
float **getPower(int **pixel, int width, int height,float c,float gamma){

	float **out;
	MALLOC_OUTPUT_FLOAT(out, width, height);
	for(int y=0;y<height;y++){
		for(int x=0;x<width;x++){
			out[y][x] = c * pow(pixel[y][x],gamma);
		}
	}
	return out;
}
```

pngread.c

```
#include<stdio.h>
#include <stdlib.h>
#include "png.h"
#include "process.h"

int main(int argc,char **argv){
	int width,height;
	int **pixel;
	int **out;
	float **tmp;
	pixel = readPNG(argv[1],&width,&height);

	writePNG("res_original.png",pixel,width,height);

	out = histEQ(pixel,width,height);
	writePNG("res_Equalized.png",out,width,height);
	FREE_PIXEL(out);

	out = getNegative(pixel,width,height);
	writePNG("res_Negative.png",out,width,height);
	FREE_PIXEL(out);

	out = halfIntensity(pixel,width,height);
	writePNG("res_HalfI.png",out,width,height);
	FREE_PIXEL(out);

	out = IRescale(pixel,width,height,min,max,0,100);
	writePNG("res_scaleto0100.png",out,width,height);
	FREE_PIXEL(out);

	out = IRescale(pixel,width,height,min,max,200,255);
	writePNG("res_scaleto200255.png",out,width,height);
	FREE_PIXEL(out);

	tmp = getExp(pixel,width,height,20,1.01);
	out = Normalize(tmp,width,height);
	writePNG("res_exp.png",out,width,height);
	FREE_PIXEL(out);

	tmp = getLog(pixel,width,height,20.2);
	out = Normalize(tmp,width,height);
	writePNG("res_log.png",out,width,height);
	FREE_PIXEL(out);

	tmp = getPower(pixel,width,height,1,0.5);
	out = Normalize(tmp,width,height);
	writePNG("res_powerg05.png",out,width,height);
	FREE_PIXEL(out);

	tmp = getPower(pixel,width,height,1,2);
	out = Normalize(tmp,width,height);
	writePNG("res_powerg2.png",out,width,height);
	FREE_PIXEL(out);

	tmp = getPower(pixel,width,height,1,2.5);
	out = Normalize(tmp,width,height);
	writePNG("res_powerg25.png",out,width,height);
	FREE_PIXEL(out);
	//printHist(out,width,height);

	free(pixel);
return 0;
}
```