Assignment-1 Report

Name: Ting Yao Chen
ID: 314832008

***Abstract*—This report is for Assignment-1 of the MMIP course which contains three main tasks, 1) reading 512×512 grayscale RAW and common JPEG/PNG images and printing the centered 10×10 pixel matrix, 2) applying log, gamma, and negative point operations, and 3) performing down/up-sampling with nearest-neighbor and bilinear interpolation on given data and comparing the experimental results. Github: https://github.com/tingul4/MMIP-2025fall**

I. Introduction

The goal is to build end-to-end familiarity with image I/O, grayscale processing, point-wise intensity mappings, and geometric resampling, following an IEEE conference paper format for documentation.

II. Implementation

A. Image reading

Image reading functionality was implemented to handle both RAW grayscale images and standard formats like BMP. For RAW files (e.g., '.raw'), a custom function reads the byte stream directly into a memory buffer, assuming 8-bit grayscale pixels in row-major order. For common formats, the program leverages the 'stb_image.h' single-file public domain library [1], which simplifies the process of decoding various image types into a consistent pixel data array. All images are internally represented as a structure containing dimensions (width, height), channel count, and a pointer to the pixel data.

B. Image enhancement

The log mapping uses

$$s = c \cdot \log(1 + r), \quad c = \frac{255}{\log(256)}, \tag{1}$$

to enhance low intensities without saturating highlights, where $r \in [0, 255]$ and the output is rounded and clamped. Gamma correction uses

$$s = 255 \cdot (r/255)^{\gamma}, \tag{2}$$

with γ provided via the command line to explore brightening ($\gamma < 1$) and darkening ($\gamma > 1$). The negative is computed as $s = 255 - r$, which is also useful for debugging dynamic range assumptions.

C. Image downsampling and upsampling

Nearest-neighbor computes source coordinates and picks the closest integer pixel, which is fast but prone to aliasing and blockiness at large scale changes. Bilinear uses half-pixel alignment $g_x = (x + 0.5) \cdot s_x - 0.5$ and $g_y = (y + 0.5) \cdot s_y - 0.5$, interpolates from the four neighbors, and clamps at image borders for stability.

III. Experiments

Experiments were conducted to verify the implemented algorithms. The results are organized by task.

A. Part A: Image Reading and Centering

Six images ('baboon.bmp', 'boat.bmp', 'F16.bmp', 'goldhill.raw', 'lena.raw', 'peppers.raw') were read. For each, a 10×10 pixel block from the center was extracted. Figure 1 shows an example for 'lena.png', where the centered block is highlighted. You can find the shell script 'hw1/run_problem_a.sh' in my project and run it to get all results in the 'out/A'.

(a) Original Image (b) Center pixel

Fig. 1: Example of center 10x10 block extraction on 'lena'.

B. Part B: Image enhancement

Log transform, gamma correction ($\gamma = 2.2$), and negative operations were applied to all six images. Figure 2 compares these effects on the 'peppers' image. You can find the shell script 'hw1/run_problem_b.sh' in my project and run it to get all results in the 'out/B'.

C. Part C: Image downsampling and upsampling

The 'F16.bmp' image was resized to various dimensions using both nearest-neighbor and bilinear interpolation. Figure 3 compares down-sampling to 128×128, while Figure 4 compares up-sampling to 1024×512. You can find the shell script 'hw1/run_problem_c.sh' in my project and run it to get all results in the 'out/C'.

(a) Original

(b) Log Transform

(c) Gamma ($\gamma = 2.2$)

(d) Negative

Fig. 2: Point operations applied to ‘peppers‘ image.

(a) Nearest (128x128)

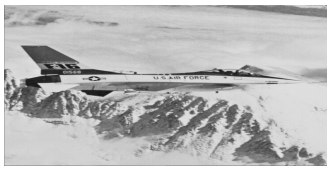
(b) Bilinear (128x128)

Fig. 3: Down-sampling comparison.

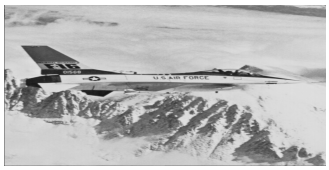
IV. Results and Discussion

The experimental results align with theoretical expectations. In point operations, the log transform successfully expanded the dynamic range of darker pixels, making details in shadows more visible. Gamma correction with $\gamma = 2.2$ produced a perceptually darker image, as expected for display on systems with a lower native gamma. The negative transform correctly inverted all pixel intensities.

In resizing, nearest-neighbor interpolation was significantly faster but produced noticeable blocky artifacts, especially in up-sampling and at edges. Bilinear interpolation produced much smoother results by averaging pixel values, but this came at the cost of slightly blurring the image and losing some high-frequency details. The difference is most apparent in the up-sampled images (Fig. 4), where the smooth gradients of the bilinear method are visually preferable to the sharp, jagged edges from the nearest-neighbor method.



(a) Nearest (1024x512)



(b) Bilinear (1024x512)

Fig. 4: Up-sampling comparison.

V. Difficulties and Solutions

A. Reading Compressed Image Files

a) Problem: Directly reading pixel data from compressed image formats like JPEG (.jpg) is complex. Unlike raw data or simple BMP files, JPEGs use sophisticated compression algorithms (like DCT and Huffman coding).

b) Solution: To efficiently handle various image formats, we integrated the `stb_image.h` library. This library abstracts the decoding process, allowing us to load images like JPEGs, PNGs, and BMPs with a simple function call.

B. Boundary Handling in Image Resizing

a) Problem: During image resizing operations (e.g., using bilinear interpolation), the algorithm calculates source coordinates that may fall outside the image's actual boundaries (e.g., less than 0 or greater than width-1). Attempting to read pixel data from these invalid coordinates can lead to memory access errors (segmentation faults) or visual artifacts in the output image.

b) Solution: Before accessing a source pixel, we check if the calculated coordinates are within the valid range. If a coordinate is out of bounds, it is ”clamped” to the nearest edge. For instance, a calculated x-coordinate of -5 would be treated as 0, and a y-coordinate exceeding the image height would be set to `height-1`.

VI. Conclusion

This assignment provided hands-on experience with fundamental image processing tasks. Key functionalities including image I/O for different formats, point-wise intensity transformations (log, gamma, negative), and geometric resizing (nearest-neighbor, bilinear interpolation) were successfully implemented and verified. The experiments visually confirmed the theoretical properties of each algorithm and highlighted the trade-offs between methods, such as speed versus quality in the case of resizing. The main difficulties I encountered were solving pixel value overflow and underflow after image processing, and understanding how computers process images. Overall, these were the key concepts I grasped through this assignment.

References

[1] S. Barrett, "stb single-file public domain libraries," https://github.com/nothings/stb, accessed 2025.