Implementation of Basic Image Processing Operations Using C++

Pen-Yi Lee
National Yang Ming Chiao Tung University (NYCU)
Email: joshua12390902@gnail.com

***Abstract*—This paper presents a C++ implementation of fundamental image processing tasks, including raw and compressed image reading, basic enhancement, and resampling. Negative, log, and gamma transforms were applied, and resizing was performed with nearest-neighbor and bilinear interpolation. Results show clear differences among methods, reinforcing pixel-level processing concepts.**

***Index Terms*—Image Processing, C++, RAW Image, Enhancement, Resampling, Interpolation**

I. Introduction

This assignment aims to practice low-level image processing by implementing core operations *from scratch* in C++, without convenience libraries (e.g., OpenCV). The tasks include: (a) reading three `.raw` (512×512, grayscale, row-major) and three compressed images (BMP/JPG),and show on the screen (b) an enhancement toolkit (negative, log, gamma), and (c) resampling with nearest-neighbor and bilinear interpolation under specified size changes. The results are shown on screen and also saved as clean files for checking.

II. Implementation Details

A. (a) Image Reading

We implemented four utility functions to correctly load, output, and inspect images:

- `read_raw(path, img)`: reads 512×512 grayscale RAW data (row-major order).
- `read_gray_any(path, img)`: loads BMP/JPG using `stb_image` and converts to grayscale.
- `write_pgm(path, img)`: exports results into PGM format.
- `print_center10(img, tag)`: prints and saves the centered 10×10 pixel block to CSV.

B. (b) Image Enhancement Toolkit

We implemented three common intensity transforms. Each is encapsulated in a function that takes a `GrayImage` and returns a transformed copy.

1) Image Negative:

$$s = 255 - r$$

where r is the input gray level and s is the output. Implemented in `negative(img)`.

2) Log Transform:

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

This enhances dark regions by compressing high intensities. Implemented in `log_transform(img)`.

3) Gamma Transform:

$$s = 255 \cdot \left(\frac{r}{255}\right)^{\gamma}$$

where γ controls brightness: $\gamma < 1$ brightens, $\gamma > 1$ darkens. Implemented in `gamma_transform(img, γ)`.

C. (c) Downsampling and Upsampling

Resampling is handled by mapping coordinates and interpolation.

1) Coordinate Mapping: We map each destination pixel (u, v) back to the source:

$$u_s = \frac{(u + 0.5)\,\text{src_w}}{\text{dst_w}} - 0.5, \qquad v_s = \frac{(v + 0.5)\,\text{src_h}}{\text{dst_h}} - 0.5$$

Implemented in `map_coord()`.

2) Nearest Neighbor:

$$s(u, v) = f(\text{round}(u_s), \text{round}(v_s))$$

Implemented in `resize_nearest()`.

3) Bilinear Interpolation:

$$\begin{aligned} s(u, v) = {} & (1 - dx)(1 - dy)\, f(x_0, y_0) \\ & + dx(1 - dy)\, f(x_1, y_0) \\ & + (1 - dx)dy\, f(x_0, y_1) \\ & + dxdy\, f(x_1, y_1) \end{aligned} \tag{1}$$

where $(x_0, y_0), (x_1, y_0), (x_0, y_1), (x_1, y_1)$ are the four surrounding source pixels. Implemented in `resize_bilinear()`.

III. Experiments

This section presents the experimental results for three tasks: (a) image reading, (b) image enhancement, and (c) image resampling. All outputs were both displayed on screen (via SDL3) and saved in PGM format under the `results/` folder for verification.

A. Image Reading

Three RAW images (512×512, grayscale) were loaded. The central 10×10 pixel values of each are shown below for verification.

TABLE I: Central 10×10 pixel values of RAW/BMP images.

Lena

195	195	195	192	169	135	133	137	138	148
196	196	189	157	124	128	135	144	145	145
196	187	149	123	127	131	135	142	142	137
180	135	116	117	124	130	131	132	137	133
124	102	115	116	120	126	124	120	114	111
100	102	114	114	114	118	120	117	112	92
101	100	113	106	98	89	90	104	101	84
105	96	99	94	92	78	75	79	83	76
92	90	90	84	79	75	71	73	72	71
85	79	76	77	76	71	73	73	69	77

Goldhill

56	60	69	64	54	74	80	78	78	121
47	59	62	69	54	62	48	50	52	70
70	74	73	69	51	58	44	46	46	57
85	85	87	72	45	59	45	45	55	68
65	81	80	78	49	64	52	54	65	69
60	63	78	80	53	83	59	63	78	62
59	66	69	66	57	72	66	71	74	51
53	64	63	75	54	67	68	59	57	53
57	64	63	69	66	74	75	68	89	101
77	68	74	76	94	96	99	88	106	103

Peppers

63	43	45	60	71	61	58	25	4	20
66	50	46	42	55	40	52	34	4	26
60	57	35	36	47	32	48	25	15	56
86	57	33	37	23	34	25	35	66	59
112	91	57	27	27	14	27	49	57	46
108	69	69	39	20	9	42	73	52	46
113	92	45	32	13	44	55	65	55	32
131	90	59	27	33	62	72	62	59	47
144	96	40	30	57	92	73	66	60	45
133	72	34	61	99	112	100	97	80	48

Baboon

152	151	154	167	155	142	139	136	139	143
150	175	177	178	180	177	174	173	153	130
149	165	155	155	170	174	173	171	163	159
149	158	169	171	172	166	151	165	156	138
164	170	177	177	180	177	172	178	170	156
172	186	185	175	177	181	184	180	176	165
184	183	166	168	173	170	167	176	171	171
181	170	175	190	190	188	178	176	175	171
183	187	186	190	189	188	176	162	158	153
190	192	184	176	188	186	184	174	164	159

Boat

88	74	105	105	48	57	62	75	140	142
83	71	85	101	58	50	63	65	129	143
84	70	65	67	53	49	54	66	123	141
82	66	56	53	45	53	55	69	127	136
86	65	60	49	47	52	55	67	118	138
79	71	59	50	51	50	55	72	122	133
86	76	61	53	54	54	58	68	117	132
89	77	61	59	52	56	55	65	116	132
84	78	61	57	55	57	58	65	117	137
78	77	55	58	52	56	57	68	127	136

F16

105	110	108	111	118	130	126	123	115	107
106	109	109	114	123	131	124	117	116	107
111	107	111	120	129	127	124	115	115	101
110	107	116	118	123	126	119	111	113	108
106	109	109	114	112	113	108	109	114	108
99	101	102	103	104	108	109	108	109	109
101	102	95	100	104	106	106	106	107	108
94	96	102	105	108	101	102	104	108	107
94	99	104	104	103	101	103	98	106	108
101	101	101	99	104	99	104	101	104	104

B. Image Enhancement Toolkit

Each image was processed using three methods:

- **Negative:** $s = 255 - r$, inverting brightness.
- **Log Transform:** $s = c \cdot \log(1 + r)$, $c = 255/\log(256)$, enhancing dark regions.
- **Gamma Transform:** $s = 255 \cdot (r/255)^{\gamma}$, $\gamma = 2.2$, adjusting brightness nonlinearly.

C. Image Downsampling and Upsampling

In this part, we implemented two resampling methods: *nearest-neighbor interpolation* and *bilinear interpolation*. The helper function `map_coord()` was used to map the pixel center in the new image back to a real-valued coordinate in the original image. Since six images were processed in the experiments, the results are very similar across them. For clarity, we only present the **Lena** image as a representative example in this report. Five scaling cases were performed:

1) $512 \times 512 \rightarrow 128 \times 128$
2) $512 \times 512 \rightarrow 32 \times 32$
3) $32 \times 32 \rightarrow 512 \times 512$
4) $512 \times 512 \rightarrow 1024 \times 512$
5) $128 \times 128 \rightarrow 256 \times 512$

Functions used: `map_coord()`, `resize_nearest()`, `resize_bilinear()`. **Observations:** Nearest Neighbor is sharper but produces blocky artifacts, while Bilinear is smoother but slightly blurred. Extreme downsampling (32×32) loses significant detail in both methods.

IV. Difficulties and Lessons Learned

Numeric care: log/gamma require clamping before casting to 8-bit. **Interpolation edges:** bilinear needs careful boundary handling. **Display setup:** SDL3 on Windows requires include/lib configuration and placing `SDL3.dll` beside the executable; headless setups fall back to PGM/CSV.

V. Reproducibility (Build & Run)

Deps: `stb_image.h`, SDL3 (VC development package). **Include/Lib:** point include to `SDL3/include` and link `SDL3.lib`; copy `SDL3.dll` beside the executable. **Example (MinGW):**

```
g++ Assignment1.cpp -IC:/SDL3/include -LC:/SDL3/li
```

Inputs: three `.raw` (512×512) and three BMP/JPG under `./data`. **Outputs:** PGM in `./results` and centered 10×10 CSV per input.

VI. Reproducibility and Environment Setup

Our project is structured as a VSCode workspace with integrated build and run tasks. Both `tasks.json` and `launch.json` are configured so that, on the original development machine, pressing `F5` automatically compiles and executes the program. To reproduce the results on another system, the following setup steps are required.

A. Dependencies

- **stb_image.h**: Included in the repository; no installation needed.
- **SDL3**: Must be installed separately.

TABLE II: Comparison of enhancement methods (Lena, Goldhill, Peppers,Baboon, Boat, F16).

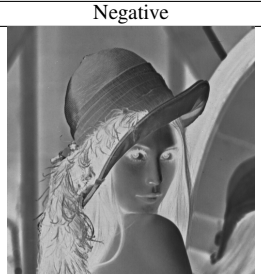
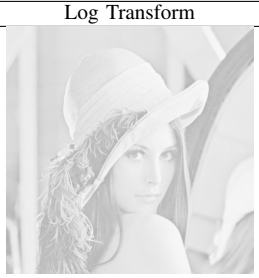
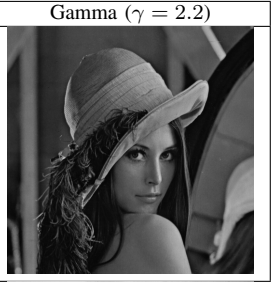
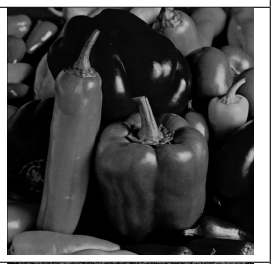
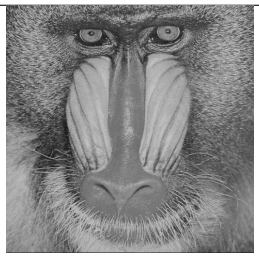
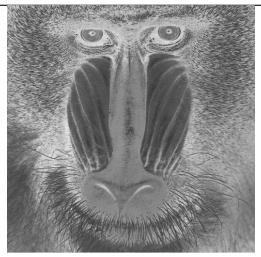
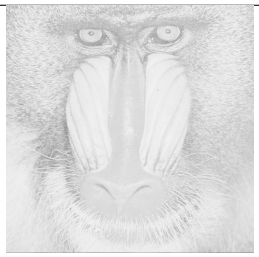
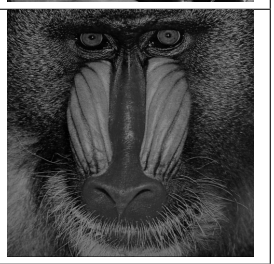
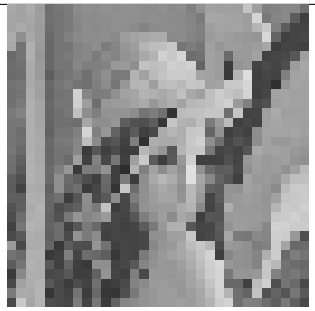
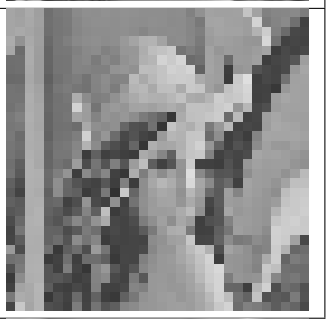
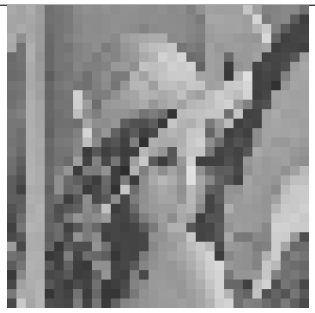
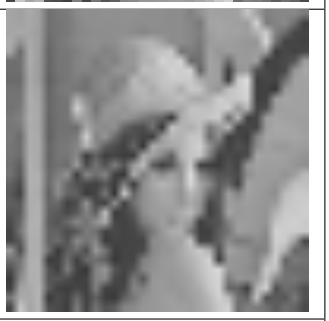
Image	Original	Negative	Log Transform	Gamma ($\gamma = 2.2$)
Lena				
Goldhill				
Peppers				
Baboon				
Boat				
F16				

TABLE III: RESAMPLING RESULTS ON LENA (NEAREST VS BILINEAR).

Case	Input	Nearest	Bilinear
(i) $512 \rightarrow 128$			
(ii) $512 \rightarrow 32$			
(iii) $32 \rightarrow 512$			
(iv) $512 \rightarrow 1024 \times 512$			
(v) $128 \rightarrow 256 \times 512$			

B. Windows Setup (MinGW + VSCode)

1) Download the official SDL3 development package from https://github.com/libsdl-org/SDL/releases.
2) Extract the archive. Ensure that the folders `include/` and `lib/x64/` are available.
3) Update the `.vscode/tasks.json` include and library paths:

```
-IC:/path/to/SDL3/include
-LC:/path/to/SDL3/lib/x64
```

4) Place `SDL3.dll` in the same folder as the compiled executable (`Assignment1.exe`).

C. Linux Setup

1) Install SDL3 development headers via:

```
sudo apt install libsdl3-dev
```

2) Compile with:

```
g++ Assignment1.cpp -lSDL3 -o Assignment1
```

D. Execution

After setup, simply press `F5` in VSCode to build and run. On systems without SDL3 or a graphics environment (e.g., headless Docker/WSL), the program gracefully falls back: all results are still written to the `results/` folder as `.pgm` images and 10×10 center CSV files, enabling verification without on-screen display.

VII. Conclusion

We implemented image reading, enhancement, and resampling in C++ without high-level libraries. Experiments illustrate the characteristics and trade-offs of negative/log/gamma and nearest vs. bilinear interpolation. The work consolidates pixel-level understanding and prepares for subsequent assignments.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Pearson, 2018.

[2] S. Barrett, `stb_image.h`. Available: https://github.com/nothings/stb

[3] Simple DirectMedia Layer (SDL3). Available: https://libsdl.org/