Santanu Das (MA24M021)

Technical Approach & Kernel Design

This project implements a two-stage GPU pipeline using OpenCL: a 3x3 Gaussian blur followed by logarithmic tone mapping. Host-side Python (PyOpenCL) handles I/O, buffer creation, and kernel dispatch.

Gaussian Blur: Each work-item handles one pixel. A 3×3 Gaussian kernel is applied to R, G, and B channels. Boundary issues are handled using coordinate clamping, preserving edges and avoiding memory errors.

Logarithmic Tone Mapping: After blur, luminance is compressed using:

$$Y = 0.2126R + 0.7152G + 0.0722B, \quad Y_{\text{out}} = \frac{\log(1+Y)}{\log(1+\text{max\_luminance})}$$

Color is preserved using:

$$R_{\text{out}} = R \cdot \frac{Y_{\text{out}}}{Y}, \quad G_{\text{out}} = G \cdot \frac{Y_{\text{out}}}{Y}, \quad B_{\text{out}} = B \cdot \frac{Y_{\text{out}}}{Y}$$

Avoiding divide-by-zero for $Y = 0$. Alpha is unchanged.

Note: This formula $\log(1+Y) \cdot \log(1+\text{max\_luminance})$ was given in the assignment. It **increases** brightness and fails tone mapping. Correct formula ensures HDR→LDR compression. May be type error.

Challenges, Learning & Problem-Solving

Key Issues:

- Apple M1 Metal backend rejects static/2D arrays. OpenCL C disallows non-constant array initializers—used 1D arrays instead.
- Shape mismatch between PIL (HWC) and flat RGBA OpenCL buffer required careful conversion.
- Kernel sync: `queue.finish()` is correct. No `cl.finish()` exists in PyOpenCL.

Learned: Kernel vectorization, buffer flags, and OpenCL's memory and context setup.

Parallelism & Performance Considerations

Each pixel is handled by one GPU thread (2D global range). Both blur and tone mapping are embarrassingly parallel with no inter-thread dependency.

Performance Factors:

- **Memory Access:** Gaussian blur reads 9 pixels—could benefit from local memory.

Project Closure & Future Work

The project was approached as a daily task with progress committed regularly to Git and it took around 1 hr 15–30 mins daily.

Special thanks to Cursor and ChatGPT for intelligent suggestions and debugging.

Next Steps:

- Add 5x5 and separable filters for improved quality.
- Implement kernels that modify the alpha channel to control pixel transparency, enabling effects like fading, soft blending, and dynamic masking—useful for compositing and image enhancement pipelines.