Low Poly Effect Parallel Renderer Milestone Report

Author: Felicity Xu (hanyux), Otto Wang (aow)

Web Page: Low-Poly-Effect-Parallel-Renderer

1 Summary of work done

Over the past three weeks, we have made steady progress in the development of our low-poly image renderer using C++ and CUDA. In the first week, we thoroughly researched techniques and steps to implement the rednerer and setup the basic project structure. We intend to develop and test our renderer on GHC machines. Since OpenCV is not supported on these machines, we explored various methods to load, process, and display images. Ultimately, we decided to use CImg (CImg Github Repository).

During Week 2, we first implemented and tested the Gaussian Blur using CUDA and obtained initial performance results compared to the CPU-based implementation from CImg. Following this, we implemented the Canny edge drawing algorithm, which involves calculating gradients, applying non-maximum suppression, and extracting edges by hysteresis. An example of the results from these steps is shown below:

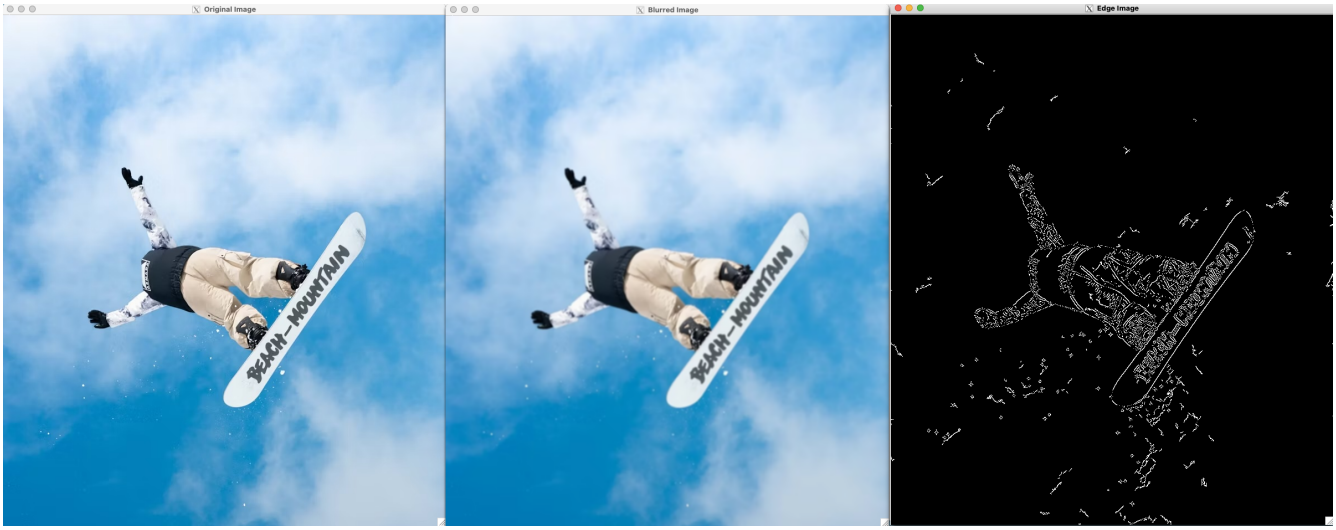

Figure 1: Blur and Edge Detection Result

When testing the Gaussian Blur algorithm with a single image, the benefits are not readily apparent, as most of the time is consumed by loading and initialization processes. We are planning to develop a small python program to generate test images from videos and test simultaneously with our renderer to more effectively demonstrate the algorithm's true capabilities.

Currently, we are working on a basic version of the Delaunay algorithm, which is the third step in our rendering process. The algorithm offers multiple implementation methods and involves various parameters. Our goal is to improve the visual output of our renderer by selecting the most suitable implementation and parameters. We are looking forward to seeing the results of this implementation.

2 Progress and Goals

Although we have not yet obtained preliminary results concerning performance enhancement, we are confident that we can deliver a fully optimized low-poly parallel renderer by the time of the poster session. Our progress is closely aligned with the goals and deliverables outlined in our initial proposal. We are on track to achieve significant parallelization and efficiency in the renderer.

However, due to upcoming final assignments and examinations, we are cautious about committing to the completion of the ”nice-to-have” video renderer. While this feature would enhance our project, we must prioritize our core deliverables given the limited time available.

For the upcoming poster session, our revised list of goals includes the following:

1. Complete CPU-version low-poly renderer that produces good effect for a broad range of images.
2. Complete and optimize the CUDA-version low-poly renderer such that it has performance at least 10x better than CPU version.
3. Implement the automated test program and conduct comprehensive testing to validate the functionality and performance of the renderer across various scenarios.

These goals are designed to ensure that we meet our primary project objectives, while also preparing a robust demonstration for the poster session.

3 Poster Session Plan

For the poster session, we plan to present a poster that illustrates the four key steps of our rendering process. Accompanying the illustrations, we will display a series of before-and-after images to visually demonstrate the effects and capabilities of our renderer. Additionally, we will have a live demo, where attendees can observe our renderer in action. This demo will involve inputting images into our system, which will then showcase the computation results and relevant statistics in real time.

4 Issues and Concerns

Our largest concern is about Delaunay triangulation, in the following two aspects:

1. Extracting Anchors from Discontinuous Edge Detection Results

The process of extracting anchors for Delaunay triangulation relies heavily on the output from the edge detection phase. However, edge detection results, like those from Canny, often yield discontinuous edges. This discontinuity can lead to incomplete or sparse triangulations, which adversely affect the visual quality and integrity of the low-poly rendering.

Possible Solution: Edge Linking and Contour Tracing

Implement an additional processing step that involves edge linking or contour tracing algorithms. This step would connect discontinuous edge fragments to form a more continuous outline, serving as a better

foundation for triangulation. Techniques such as the Hough Transform for line detection or contour tracing algorithms could be utilized to bridge gaps in detected edges, providing a more robust set of vertices for triangulation.

2. Balancing Load on GPU for Parallel Triangulation Tasks

Balancing computational load across GPU cores is crucial for optimizing the performance of parallel algorithms like Delaunay triangulation. Uneven distribution of tasks can lead to some GPU cores being underutilized while others are overloaded, resulting in inefficient computation and increased processing time.

Possible Solution: Spatial Decomposition Using Uneven Grids

Divide the image or computational domain into smaller grids that can be processed independently in parallel. Assign these segments to GPU cores in such a way that each core handles a roughly equal number of anchors. This spatial decomposition approach helps in achieving a more uniform distribution of computational load, which is particularly effective for image-based applications like triangulation.