3D Object Rasterization and Rendering with CUDA

Minyu Cai (minyuc), Jinyan Zhu (jinyanz)

webpage: `https://jerrycmy2001.github.io/3d-object-rendering/`

1 Summary

We are going to implement 3D object rasterization and render a 360-degree camera view of the object. The entire project will be built on CUDA. We will first model the 3D object and camera position, and then use rasterization and z-buffering to render the view.

2 Background

In 3D graphics, rasterization and rendering are essential processes that transform complex 3D scenes into 2D images. These tasks are computationally demanding, as they involve transforming, shading, and displaying millions of pixels per frame, often at high frame rates for real-time applications. CUDA framework provides a parallel programming platform that enables the execution of these tasks across thousands of GPU cores, allowing for substantial acceleration in rendering tasks. Specifically, a CUDA core can be mapped to render an area of pixels, so that the entire rendering can be done in parallel and achieve huge speedup.

3 Challenge

In rasterization and rendering, the main challenge lies in the 3D objects to be rendered, as shown the following ways:

- Each thread need to access the global data of 3D objects, like vertexes and edges, causing potential bottlenecks in memory transfer.
- As we cannot predict the distribution of 3D objects in the scene, there might be workload imbalance between various threads working on different areas of the scene.
- The real-time rendering task imposes huge latency constraints in order to display the video smoothly.

4 Resources

4.1 Hardware

During development, we'll use local NVIDIA 3060 desktop GPU for testing purposes. Afterwards, we'll further parallelize the task among a batch of GPUs on PSC machine to evaluate the final results.

4.2 Code

We'll reuse the code structure from project 2, including window display and basic CUDA operations. For the rest of the implementation, we'll search online for the algorithm details of rasterization and relevant tasks, implement the serialized version and attempt to parallelize them.

5 Goals and Deliverables

5.1 Basic Goals - plan to achieve

Some basic goals of our project are:

- Model 3D objects, camera position and direction. This is the prerequisite to performing rendering and creating the camera view. The 3D objects may be polygons and we need to design how to model them in a correct and efficient way.
- Render camera view with 3D object projection with rasterization and z-buffering. This is the second step after modeling. When rendering the camera view, we need to focus on each view with specific positions and directions. Therefore, rasterization and z-buffering are a must for our project.
- Rotate the camera to get a 360-degree view of the objects. After we achieve the first two goals, the final step is to use the techniques we have implemented to get the entire view. Till this, we believe the whole object is finished.

5.2 Stretch Goals - hope to achieve

If we can finish the above basic goals beforehand, we would like to try the anti-aliasing technique to enhance rendering. We believe this can optimize the final output, but this is just a "nice-to-have" part and is not included in our basic goals.

5.3 Deliverables

5.3.1 System Overviews

Our final system will be capable of creating a 360-degree camera view of a 3D object already set in the system. We won't accept any pictures or real-time photos as our input. The system will just give the camera view of the modeled object and we hope there will be satisfying speedups with parallelism.

5.3.2 Demo

In the demo at the poster session, we will show the final 360-degree view of a 3D object and also some tables and graphs of the speedups.

6 Platform Choice

We'll use CUDA as the main framework because graphics rendering is almost always performed on GPU in real life. Meanwhile, we'll use `OpenMP` to further distribute the task among multiple GPUs on PSC.

7 Schedule

- Week 1 (11/11)
  - model 3D object data structures and create example scenes
  - migrate window display and timing from project 2 codebase
- Week 2 (11/18)
  - implement naive version of rasterization on CUDA, including projection and z-buffering
- Week 3 (11/25)
  - bug fix the rasterization algorithm and perform optimizations
  - implement camera rotation
- Week 4 (12/02)
  - use `OpenMP` to parallelize the rendering across multiple GPUs
  - experiment on PSC machine, collect data and analyze speedup
- Week 5 (12/09)
  - bug fix or additional optimization if needed
  - stretch task if time permits (anti-aliasing algorithm)