Parallelization of the Möller–Trumbore intersection algorithm in a Human Shape Estimation context

Daniel Amadori
VR488052

LaTeX

A.A. 2023/24
Laurea Magistrale in Ingegneria e Scienze Informatiche
Dipartimento di Informatica
Università degli studi di Verona

Indice

1 Abstract

The Möller–Trumbore intersection algorithm is a method used to check if a ray (a line with a starting point and a direction) intersects a triangle in 3D space. It works by solving mathematical equations to determine if the ray and the triangle intersect. If they do, the algorithm provides the point of intersection. This is commonly used in computer graphics to determine which parts of a 3D model are visible from a certain viewpoint.

2 Introduction

This project focuses on the parallelization of the Möller–Trumbore intersection algorithm to accelerate human shape estimation. Given a dataset consisting of 6890 3D points and 13776 triangular meshes, the primary objective is to determine, for each 3D point, whether the segment connecting the origin (0,0,0) to that point intersects any of the provided meshes.

3 Implementation

The algorithm is implemented in CUDA using shared memories. The input of the kernel function is:

- **orig**: camera location, in our case is (0,0,0)
- **dir**: verts, values read from the 'rotated_verts.csv' file
- **V1, V2, V3**: values computed using verts and meshes, with the latter values read from the 'meshes.csv' file.
- **rows**: number of rows that is defined by BLOCK_ROWS_SIZE
- **border, lineType, planeType**: these parameters define the problem; in our case, we consider, respectively:
  - BORDER_EXCLUSIVE
  - LINE_TYPE_SEGMENT
  - PLANE_TYPE_TWO_SIDED

The output of the kernel function is just an array of boolean named **visible**.

The pragma **BLOCK_ROWS_SIZE** is used to define the size of the kernel that is (1, BLOCK_ROWS_SIZE). The value of this pragma is **352**, a multiple of 32 (WARP size), representing the maximum size allowed by the shared memory per block available on this GPU. With this kernel size we use 44 KB of the 48 KB available.
To obtain the value, an evaluation of the space cost was conducted. To optimize it, the function was divided into two parts, allowing the space used by the variables in the first part to be reused in the second part.
In the first part, **pvec** was used, with a size of 3 * BLOCK_ROWS_SIZE * sizeof(double).
In the second part, the following were used:

- **qvec**, size: 3 * BLOCK_ROWS_SIZE * sizeof(double)
- **t**, size: BLOCK_ROWS_SIZE * sizeof(double)
- **v**, size: BLOCK_ROWS_SIZE * sizeof(double)

Numerous variables are utilized in both parts. More information on the space cost analysis can be found in the file **include/fastRayTriangleIntersection_parallel.cuh**.

4 Performance Analysis

For the evaluation process, 100 executions were performed on the same dataset using a computer from the PARCO lab, equipped with an **Intel i5-7400 CPU** and an **NVIDIA GeForce RTX 2070 SUPER GPU** that has a memory of 7982 MB and a shared memory per block of 48 KB. The results obtained using the mean are:

- **Sequantial code**: 10110.92 ms, std: 1124.08 ms
- **Parallel code**: 1317.09 ms, std: 94.16 ms
- **Speedup**: 7.68

5 Conclusion

As previously explained, the bottleneck is the size of the shared memory which limits the maximum size of the kernel. Using other GPUs can easily run the code, showing memory usage and possibly modifying the kernel size to make the best use of the resources.