CS 484 Project Proposal

Parallelizing 3D mesh distance computation

Group: dshin11

Problem statement: There are numerous uses cases for computing point-triangle distances in parallel, and depending on the use case, different parallelization strategies can be adopted. Given two 3D surfaces represented as triangle meshes, the surface-surface distance is a measurement of how much they differ, and it is used as a way to measure the similarities between two 3D object shapes. Suppose there is an AI system that proposes 3D shapes. The surface distance, in this case, can be used as a metric to measures how closely the predicted shape agrees with the true shape. It can be estimated by uniformly sampling a lot of points from one of the surfaces and computing the average of distances to the closest point on the other surface. Another, more common, use case is ray tracing in computer graphics. Parallelization strategies can differ in those two use cases because ray tracing involves building a large query data structure once and then reusing it many times, e.g. to generate multiple renderings for animation; whereas in the shape evaluation use case involves building a lot of small structures that are independent of each other.

Regardless of the specific use case or the data structure used, this problem in general is very parallelizable because it involves repeating the same procedure for every point-triangle pair. For example, suppose there are 5000 query points and 50000 triangles in a 3D space, and the goal is to find the closest part of the triangle from every query point. Or intersect oriented points with triangles.

Implementation:
An implementation used in my project is based on the method described in
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.479.8237&rep=rep1&type=pdf .
I initially wrote a non-parallel version in Python, but it was quite slow, so for this project, the minimum goal is to rewrite it in C++ and OpenMP/I and write a report comparing the improvements made in the process, likely ones that involve how the work is distributed and also optimizing for better cache behavior.

The above is a minimum goal, and another direction for the project is to use 3D data structures such as octrees for querying the triangles and parallelizing it. Note that, although the algorithm described in the paper above is optimized, the time complexity is still brute-force, but the advantage is that it can be vectorized very easily. An octree may be faster or slower depending on how many triangles are stored and how frequently the constructed tree is reused. Experimenting with this performance trade-offs a potential direction for this project.

Octree code for querying triangles was already written by me for CS 419 from scratch, so some of it will be reused for this project as a starter code. It is written in c++ and uses Eigen for vectorization, however it is single threaded. It can be extended to be multi-threaded. Octree parallelization on its own is a hard problem, so it may be a stretch goal, but a naive one can be implemented, and if I do, I will report speedups.

Dataset:

SHREC12 mesh dataset will be used for the experiments. It is a mesh retrieval dataset that consists of 1200 triangle meshes representing 3D objects in 40 categories. And I will report performance on two tasks 1. measuring the similarity between 3D object shapes 2. Rendering those 3D meshes to produce visualizations. Those two tasks are very different, but they both benefit from having an efficient point-triangle distance computation method, and parallelization strategies differ.