# Mesh Simplification via Embedded Tree Collapsing Implemented in CUDA

Nicolas Mignucci, George Ralph

## URL:

https://github.com/gdr22/15-418-Final-Project

## Summary:

We will be implementing a mesh simplification algorithm in CUDA on the GPUs in the GHC servers. While the original implementation of the algorithm is sequential, we will be attempting to reproduce the results of a paper which describes a way to perform this mesh reduction algorithm in parallel.

## Background:

Quadric mesh simplification algorithms use a quadric error metric to determine which edges in a manifold mesh can be collapsed to reduce its vertex count while retaining the shape of the original mesh. The original implementation, devised by Garland and Heckbert computes an error metric for each edge on the mesh, and iteratively collapses edges with the least cost until the mesh is of desired size. As edges are collapsed, the cost of other edges must be updated based on the new construction of the mesh. Additionally, the halfedge data structure commonly used to permit these edge collapses involves some invariants which cannot be easily maintained by atomic operations. For these reasons, this implementation has sequential dependencies.

## The Challenge:

For our project, we will attempt to reproduce the results of the Lee-Kyung paper in CUDA using the GHC cluster machines.

The Lee-Kyung paper describes an alternative approach to quadric mesh simplification which observes the fact that many collapsable edges in a large mesh are not directly adjacent and can be collapsed in parallel without interfering with any other edges marked for removal. In the case where two or more adjacent edges must be collapsed however, the algorithm constructs a reduction tree for the vertices being removed, and performs all edge collapse updates after they are all selected in parallel. Because edges in different trees are not adjacent, they can be collapsed in parallel without the need of atomic operations. The paper also describes some conditions to filter out collapsable edges which would result in a malformed mesh.

Because we are operating on a halfedge data structure (which is essentially a more complex linked list) we are likely to encounter limitations due to memory bandwidth, though the Lee-Kyung paper does not address optimizing with respect to global memory accesses.

## Resources:

The paper describing the sequential quadric mesh reduction algorithm can be found here:

http://www.cs.cmu.edu/~garland/Papers/quadrics.pdf

The paper describing the parallel edge collapsing algorithm can be found here:

https://link-springer-com.cmu.idm.oclc.org/content/pdf/10.1007/s00371-016-1242-z.pdf

We will likely use the Open3D library to perform common mesh operations such as Halfedge mesh loading and conversion, as well as using the library implementation of Quadric mesh decimate as a sequential comparison to our parallel algorithm.

## Goals and Deliverables:

- 11/22/21 (Milestone)
    - Load meshes as halfedge meshes from a file and onto the GPU in the format specified by the paper
    - Compute and return Quadric error at every vertex using the GPU
    - Set up harness code to collect processing time results from Open3D's SimplifyQuadricDecimation()
- 11/26/21 (PLAN TO ACHIEVE)
    - Construct trees of collapsable edges (paper step 2)
- 11/29/21 (PLAN TO ACHIEVE)
    - Collapse trees to root vertices (paper step 5)
    - Update vertex positions (paper step 6)
    - At this point, we should have a functional mesh reduction algorithm, although the results it returns may have some errors
- 12/3/21 (HOPE TO ACHIEVE)
    - Add tree splitting to satisfy the decimation quality constraint (paper step 3)
    - Split trees further to remove illegal mesh transformations such as normal flipping or topological changes (paper step 4)
    - Both of these steps can be done optionally if we remain on schedule up to this point (stretch goals) as they are merely filter operations meant to improve the quality of the output mesh
- 12/6/21 (PLAN TO ACHIEVE)
    - Record/Finalize performance metrics (speedup, iteration count...) for various meshes
- 12/9/21 (Final Project Report)

- Include speedup compared to Open3D CPU implementation
- Include output meshes compared to Open3D implementation
- Ideally we can achieve at least a factor 10 speedup over the Open3D implementation, as this is under the speedup achieved by the algorithm described in the Lee-Kyung paper against a similar GPU reduction algorithm.

## Platform Choice:

The Lee-Kyung paper uses a "PC with a 3.5GHz Intel i7 CPU and an Nvidia GTX Titan with 6GB onboard memory" to collect their results. Due to their availability and similarity to the paper's target platform, we will use the GHC cluster machines to develop and test our algorithm. Ideally this should allow our results to be comparable to those of the paper.