Method of GPU acceleration for a 3D Convex Hull

Project Title: 3D Geometry HULL Calculations

SEP Group: HULL2

By: Simon Lieu (a1740750)

Research Scope:

Part A (Hull Calculation – 3D projection) requires the 3D geometry to be 'flattened'. The methodology used may utilise the graphics processing unit (GPU) to compute the convex hull of points in a set in $\mathbb{R}^3$. This short summary paper looks into the gHull Algorithm.

Motivation:

CPU-based 3D convex hull algorithms have been developed in recent years with QuickHull being the most efficient and popular. However, such algorithms do not 'map' well to the current architectures of GPU's. Computational capabilities of GPU have surpassed that of CPU and are being used to solve large-scaled problems.

Basic Concept:

The main idea of the proposed 3D GPU convex hull algorithm is to exploit the relationship between 3D Voronoi diagram and 3D convex hull to maximise parallelism. Computing slices of the 3D Voronoi Diagram in a digital space and forming a box enclosure, a derived approximation of the convex hull can be formed.

Algorithm Overview and Explanation:

Steps:

1. **Voronoi Construction**: Compute six 2D slices of the 3D digital Voronoi Diagram (VD) for the point set S on the boundary of B. S' is the set of points for which the Voronoi regions appear on it.
   Explanation: Approximate a Voronoi Diagram restricted to siz faces, then scale the input points such that the bounding box fits inside a 3D grid. The digital VD is produced for the set on the GPU. For each side the pints are projected and recorded with one nearest point among those that fall into the same grid cell of the box.

2. **Star Creation**: Dualise the restricted Voronoi Diagram to obtain for each point s in S' s et of neighbours, called the working set of s. This is then used to construct a convex cone (represented as a star for the point).
   Explanation: Dualise the restricted VD obtained in the previous step to get a set of triangles. Dualising the restricted VD in a closed box will produce a 3D polyhedron (not always convex), however, this may be disconnected which results in holes or duplicated triangles. Instead of constructing a polyhedron, we only record the information on the adjacencies of the Voronoi cells. Each GPU thread handling a points first creates an initial star from 3 points in the working set, and then incrementally inserts the rest using the beneath-beyond algorithm.

3. **Hull Approximation**: Apply star splaying algorithm to obtain the convex hull.
   Explanation: Star splaying is an iterative approach. Stars are repeatedly checked for inconsistency, and insertions are performed using the beneath-beyond algorithm to splay the stars when needed. To perform the star splaying algorithm in parallel while achieving regularized work for different GPU threads, we carry out the consistency checking and the insertions of points in two separate steps, alternately performed until all the stars are consistent.

4. **Point Addition**: Collect points of S that lie outside of the convex hull set and for each of them construct its star using nearby vertices.
   Explanation: Due to points being an approximation, it may not contain all extreme vertices. We check the points in the set and remove those that are inside the hull.

5. **Hull Completion**: Perform star splaying again to approximate to transform into a convex hull. **Explanation**: Approximation now contains all possible extreme vertices of S and a few more. By performing star splaying again, as detailed in Step 3, we transform our approximation into the convex hull of S.

Source:

Gao, M., Cao T.-T., Nanjappa, A., Tan, T.-S., and Huang, Z. 2013, ‘gHull: A GPU algorithm for 3D convex hull’, *ACM Transactions on Mathematical Software,* vol. 40, no. 1