Point Cloud Simplification

Matthew Cirimele and Ryan Benson

Computational Geometry

Rensselaer Polytechnic Institute

matthew.cirimele@gmail.com

bensonryan023@gmail.com

ABSTRACT

This paper will discuss the implementation of two point cloud simplification algorithms, consisting of Kd-Tree and Octree data structures, comparing and contrasting the results of both point-mesh simplification and mesh reconstruction across both functions for 3D point sets. Each simplification method is calculated with a varying bucket size, exploring the effects of an increasing reduction of points over the duration of each mesh construction. The results of the experiment showcase that simplification of points with Octrees is faster than simplification with Kd-Trees at higher bucket sizes, but is slower at lower bucket sizes. Additionally, simplification using Kd-Trees results in a lower average distance between the original point cloud and the simplified point cloud, while also resulting in a lower “nearest point distance”, signifying a more accurate (closer match) simplification process than the Octree method. Lastly, simplification using the Kd-Tree method results in point clouds with less overall points compared to Octrees, resulting in more efficient space savings.

1. INTRODUCTION

The goal of this project is to explore the usage of two tree-based data structures, Kd-Trees and Octrees, for the purposes of implementing efficient point cloud simplification algorithms, and by extension mesh

simplification. Point cloud simplification is the process in which a large point cloud, consisting of many points, is “simplified” producing a point cloud consisting of less points while maintaining the core structure and arrangement of the points. Point cloud data, often collected through techniques like LIDAR scanning or other environmental scanning methods, can consist of millions of unique points. The sheer size of these point clouds presents challenges in terms of mobility and sharing, necessitating the use of compression or simplification strategies for efficient data management. For instance, large point clouds often contain clusters of redundant points that contribute to unnecessary increases in data size without adding value to the final analysis. In scenarios requiring real-time analysis, such as robotic navigation, it's advantageous to discard or simplify fine details to enhance both accuracy (by reducing noise) and processing speed. Simplification also benefits other applications like artificial intelligence training, where analyzing compressed point cloud datasets is common. Here, reducing the size of point clouds is essential, as AI model training involves processing numerous samples, which is impractical if each sample is excessively large. The resolution of the simplified point cloud must be carefully adjusted to suit different applications, as the required precision or the degree of similarity between the original and simplified cloud varies. This paper introduces two methods for point cloud simplification: one based on Octrees and the other on Kd-Trees. In both methods, an advancing front mesh reconstruction algorithm is applied to the simplified point clouds, which also includes the removal of 5% of outliers.

2. LITERATURE REVIEW

Through our research, we have investigated and used a number of research articles to gain information and viable concepts against which to compare our work on this project. Though these are shown in the "Bibliography" section at the end of the paper, they will be described here in more detail. In the order presented in the bibliography, the first of these, "Point cloud simplification with preserved edge based on normal vector", by Han, et all, begins outlining a method by which point clouds are simplified with the express goal of preserving the "edge points", as these points are reported to store more data about the

shape of the cloud than non-edge points, and as a result they built their algorithm with the express goal of ensuring that these are kept as intact as possible. It goes on to elaborate that it sorts non edge points by importance based on distance from other nearby points and then are weaned down based on this metric. Our next paper, "A Novel Simplification Method for 3D Geometric Point Cloud Based on the Importance of Point", by Ji, et al, also outlines an algorithm for point cloud simplification, known as Detail Feature Points Simplified Algorithm, or DFPSA, wherein they use k neighborhood searching to find the nearest points to a sample point, then sorts these points by importance and removes less important ones before inserting them into an octree to simplify the remainder. Though they do not seem to have specifically been intending to optimize their algorithm for this purpose initially, they note that it preforms exceedingly well at preserving and "simplifying a subjects narrow contours" (Ji, 2019). The next paper presents a linear programming approach to the topic of point cloud simplification, designed to accurately preserve data even in the presence of noise or outlier data, and again based on the concept of identifying edges and corners to preserve them specifically. Next was a paper titled "A new point cloud simplification algorithm", by Carsten Moenning and Neil Dodgson, outlining their algorithm which is based on the Fast Marching algorithm and designed to work in a "coarse to fine grain" method, allowing for very customizable outputs in terms of resolution and file size. The final two papers we used were written by the same two authors, Hao Song and Hsi-Yung Feng, titled "A global clustering approach to point cloud simplification with a specified data reduction ratio" and "A progressive point cloud simplification algorithm with preserved sharp edge data", outline two separate algorithms for point cloud simplification, the first of which involves clustering the input set into a group of point clusters and representing them as a single point, very similar to our design. The second again seeks more to preserve sharp edge data most, and as a result also relies on a method of categorizing data points by their importance to the overall shape of the model and weaning out the ones that are least important.

3. PROJECT DESCRIPTION

The core features of the algorithm include two functions to simplify the point cloud, as well as a function to construct a mesh representation of the final point cloud. Each simplification algorithm is run on a set of bucket sizes: 1, 2, 3, 5, 10. Each bucket size corresponds to the number of leaves the bottom-most inner leaves have. For example, the final cell of an Octree with a bucket size of 3 contains 3 nodes. From there, the simplified point sets discard 5% of outlier points, then proceeds to mesh reconstruction. The resulting meshes are stored for later viewing, as the average point distance and nearest point distance are calculated.

4. ALGORITHMS

The first simplification algorithm utilizes Octrees as the primary data structure. In this algorithm, points are individually added to an Octree, with a bucket size specified by the user. From there, a leaf traversal algorithm is run over the leaves in the Octree, and the bounding box of that node is found. The center of that bounding box, calculated by averaging the four corners, is then added to a new "simplified" set. As there may be many leaf nodes that share the same bounding box if the bucket size is greater than 1, the usage of a set prevents duplicate "middle" points from entering the simplified set.

```
Function octree_middles(octree: Octree) -> Point_set:
    Initialize set as an empty Point_set

    For each node in octree.traverse(Leaves_traversal):
        If node is not empty:
            Insert the center of the bounding box of node into set
    Return set
```

Once this algorithm is complete for the desired bucket size, the simplified point cloud is constructed into a mesh. In this step, 5% of the outlier points are discarded from this new point set, and an Advancing Front mesh reconstruction algorithm creates and returns the new mesh. Additionally, a second iteration over the leaves constructs a new mesh that contains these bounding boxes. This visualizes the structure of the Octree as it contains the point set.

This process is repeated for the Kd-Tree simplification process. Because there is no leaf iterator for the CGAL Kd-Tree implementation, the tree construction, mesh construction, and point simplification processes take place simultaneously in a recursive function.

```
Function construct_kd_mesh(node: Node, bbox: Kd_tree_rectangle, mesh: Mesh, averagedPoints:
Point_set):
    If node is null, return
    If node is a leaf node:
        Cast node to Leaf_node and assign to leaf_node
        Insert the center of bbox into averagedPoints
    Else:
        Cast node to Internal_node and assign to internal_node
        // Optionally, print the bounds of bbox
        Add a cube to mesh using the min and max coordinates of bbox
        // Split the bounding box into two
        Initialize lower_bbox as a copy of bbox
        Initialize upper_bbox as a copy of bbox
        Split bbox into lower_bbox and upper_bbox using internal_node
        // Recursively process the child nodes
        Call construct_kd_mesh with internal_node's lower child and lower_bbox
        Call construct_kd_mesh with internal_node's upper child and upper_bbox
```

This function creates both the point simplification and the tree visualization. The same mesh

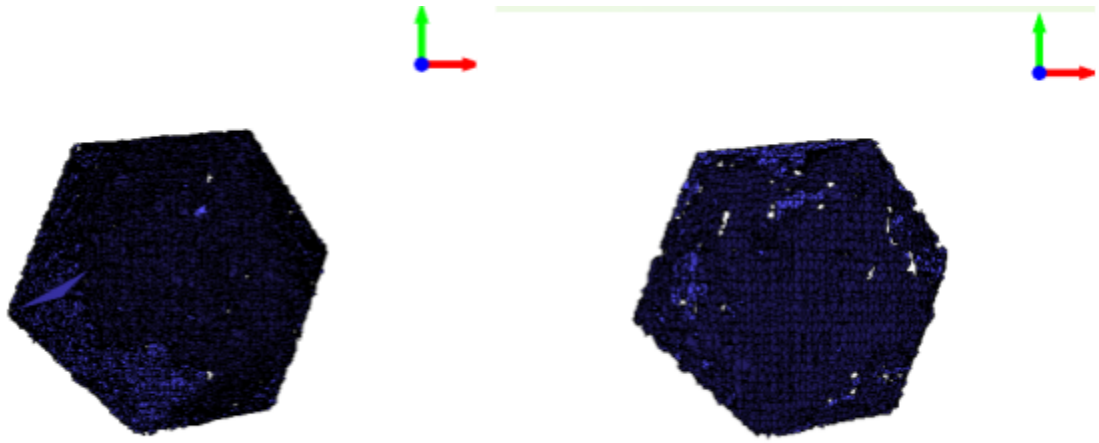
reconstruction method is then performed across this new point set.

The final two functions are average_point_distance and nearest_point_distance. Nearest_point_distance computes the nearest point distance between two point sets. It initializes the final_distance with the Euclidean distance between the first points of each set. It then iterates through every point in set1 and finds the closest point in set2, updating final_distance if a closer point is found. The function also tracks the progress of these calculations, displaying the progress in 10% increments to the standard output. The returned value is the smallest distance found between any two points in the two sets, which is used to determine the minimum distance between the original point set and the simplified point set. The average_point_distance function calculates the average distance between all pairs of points from the two sets. It initializes final_distance to 0, then iterates through each point in set1 and calculates the Euclidean distance to each point in set2, accumulating these distances. Like the first function, it also tracks and displays the progress in 10% increments. The final result, the average distance, is obtained by dividing the total accumulated distance by the total number of iterations (which is the product of the sizes of both point sets).

5. RESULTS

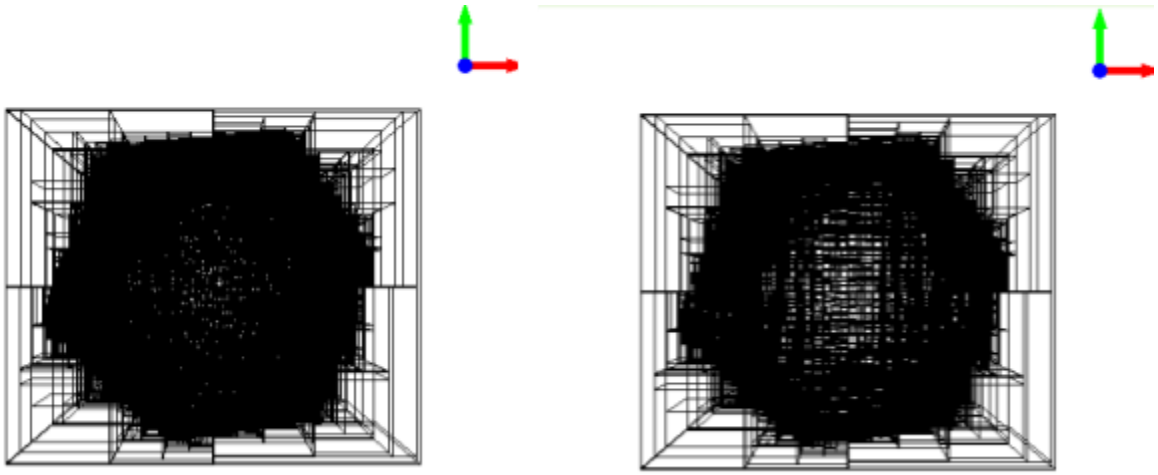
The mesh construction and simplification algorithm was applied to point clouds using varying bucket sizes of 1, 2, 3, 5, and 10. Taking the example of a point cloud named "the ball", which is a pentagonal prism consisting of 31,374 points, the application of Kd-Tree simplification with a bucket size of 5 reduced the point count to 5,870, marking a 68.02% decrease. In this process, the simplified point cloud exhibited an average point-to-point distance of 49.9552 and the nearest point distance was 0.00283438. By contrast, when the same algorithm was applied with a bucket size of 1, the point count was higher at 29,805, with an average distance of 49.9636 and a nearest point distance of 0.00108075.

This indicates that even though the point count significantly decreased when increasing the bucket size from 1 to 5, the two resulting point clouds maintained a high degree of similarity.



Left: The ball mesh reconstruction with a bucket size of 1
Right: The ball mesh reconstruction with a bucket size of 5

In the mesh reconstructions of "the ball", the left image depicts a reconstruction with a bucket size of 1, while the right image shows one with a bucket size of 5. A noticeable difference between point clouds of these bucket sizes is the increased noise in mesh construction as the bucket size grows. This is attributed to the points being more spaced out, posing challenges for the Advancing Front reconstruction method. While the point clouds are effectively simplified, as evidenced by the distance calculations, these methods are less effective for mesh simplification.



Left: The ball kd-tree visualization with a bucket size of 1
Right: The ball kd-tree visualization with a bucket size of 5

Additionally shown is the Kd-Tree visualization of the ball object. The results for Octree simplification were similar, with variations in construction times and accuracy. In point cloud simplification, the Octree method was slower at a bucket size of 1 but sped up substantially at larger sizes. Conversely, the Kd-Tree method was quicker at smaller sizes but did not accelerate as much at larger sizes. Generally, the Octree method outperformed the Kd-Tree method at bucket sizes between 5 and 10. Across all tested point clouds, the average point distance and smallest point distance for both Octree and Kd-Tree simplifications were remarkably consistent, differing by no more than one hundredth (0.01), demonstrating the consistent effectiveness of these simplification methods despite the use of different data structures.

6. BUGS AND LIMITATIONS

Our implementation currently has a few issues that could be improved on with a wider scope or more time. The first and most obvious is that of the mesh reconstruction, in which a point cloud is "remeshed", artificially adding faces between sets of points that it thinks should go together. Currently, our

implementation using the CGAL::advancing_front_surface_reconstruction() utility works well for the most part, however it currently has the tendency to leave large holes in the surface of the structure, as well as adding strange elongated triangles between different parts of the mesh where no surface should be. With more time and focus on the matter of mesh reconstruction from the simplified point set we would likely look deeper into other methods of surface reconstruction in CGAL, such as CGAL::poisson() reconstruction and CGAL::scale_space_surface_reconstruction(), to see if better results can be obtained that way, though at the moment the meshes generated are enough to visualize the changes made by our mesh simplification, and as a result are considered suitable for the time being. Another limitation we have at the moment is runtime. During our data collection, we ran our program several times on several different object files, and noticed that it takes a rather long time to even construct our octrees and kD trees, and then even more to complete the simplification and output a suitable mesh. One model we used, a 31,000 point model of an icosahedron (the "ball" model in our data), took nearly 139 seconds to even construct an octree for the data, and running the full program 5 times, to test each bucket size, took over an hour. It is worth noting that as the bucket size increased, the running time sharply decreased, however this is still a huge limitation if this is ever to be implemented in any useful capacity.

7. FUTURE WORK

Though this project covered two methods of simplifying point clouds, it was not exhaustive in terms of point simplification methods or parameters that could be adjusted. Further future works could be used to look deeper into the parameters we had available, such as the maximum depth of the kD trees and Octrees or the percentage of points that are trimmed off due to being far from other points, as well as optimization of running time, which we had problems with when running on large point clouds. Furthermore, there are several varying other options for point cloud simplification and compression that were laid out in the references and sources we read, and it may be worth looking deeper into the advantages and disadvantages of each one, both in terms of accuracy and running time. Finally, another work could be

made to investigate the viability of any or all of these changes in other dimensions, whether it be by reducing it to one or two dimensions, increasing it to higher dimensions, or generally for any dimension desired by the user.

8. WORK ALLOCATION

For this project, work was allocated as a measurable and reasonable objective for each person to achieve weekly. These objectives usually were designed to build on each other and to continue with one person working on extending the same task they had completed the prior week, in order to allow extra work to be done early and to ensure that both of us already knew the code we had been working on from previous weeks. During this time, Ryan largely focused on methods involving Octrees and averaging points within bounding boxes, while Matthew engaged mostly with working on implementations with kD trees and mesh reconstruction. For our first week, ending on November 13, our objectives were to submit our project proposal together at the beginning of the week, as well as individually to set up code to read in and visualize point clouds in CGAL for Matthew, and to construct and verify both octrees and kD trees from these input point clouds for Ryan. This also included figuring out how to adjust certain key settings of these, including maximum depth, which as a general rule we decided to leave as large as possible, and maximum bucket size, which was the main value we adjusted throughout the experiments we ran. The next week, ending November 20, Matthew worked on code designed to compare point clouds to each other by comparing distances between points and Ryan worked on simplifying point meshes further using octrees and code written to take the average location of points within octree bounding boxes. The remaining time was spent writing reports and testing the algorithms on different box sizes, to gather data in which box sizes worked better or worse than others in terms of number of points and ability to be reconstructed into comprehensible meshes after simplification

9. BIBLIOGRAPHY

Han, Huiyan, et al. "Point cloud simplification with preserved edge based on normal vector." *Optik-International Journal for Light and Electron Optics* 126.19 (2015): 2157-2162.

Ji, Chunyang, et al. "A novel simplification method for 3D geometric point cloud based on the importance of point." *IEEE Access* 7 (2019): 129029-129042.

Leal, Nallig, Esmeide Leal, and Sanchez-Torres German. "A linear programming approach for 3D point cloud simplification." *IAENG International Journal of Computer Science* 44.1 (2017): 60-67.

Moenning, Carsten, and Neil A. Dodgson. "A new point cloud simplification algorithm." *Proc. int. conf. on visualization, imaging and image processing*. 2003

Shi, Bao-Quan, Jin Liang, and Qing Liu. "Adaptive simplification of point cloud using k-means clustering." *Computer-Aided Design* 43.8 (2011): 910-922.

Song, Hao, and Hsi-Yung Feng. "A global clustering approach to point cloud simplification with a specified data reduction ratio." *Computer-Aided Design* 40.3 (2008): 281-292.

Song, Hao, and Hsi-Yung Feng. "A progressive point cloud simplification algorithm with preserved sharp edge data." *The International Journal of Advanced Manufacturing Technology* 45 (2009): 583-592.