**Delaunay Triangulation from Point Cloud**

**ME6104**

**Patrick Gardocki**

**Spring 2023**

**Introduction**

Point cloud models can represent a complex object with finite elements. This is a useful advantage of point cloud modeling if the specific geometry of a model is not known but there is access to 3D scanning technology. Using 3D scanning, models can be quickly generated. This is widely used in various industries such as construction or design to help visualize various components. While point cloud modeling is useful in representing solid models, they can cause problems when attempting to use the data for manufacturing. Point cloud models do not give information about the models surfaces or curves. However, converting the point cloud model to a 3D mesh can help the model be used in other capacities.

For my project I will implement Delaunay triangulation to create a 3D mesh given a point cloud. Delaunay’s triangulation relies on ‘the Delaunay criterion’, which sets for a set of geometric rules to determine if a given pair of neighboring triangles, is the optimal choice. I will want to compare my implementation to other public packages that also use the same method to observe the robustness and accuracy of my implementation.

Point Clouds

Point clouds are a set of coordinates that represent a physical object or model. These datasets can be gathered using various methods such as LiDAR or laser scanning. From these point clouds, 3D models can be visualized and are used in various fields such as engineering, architecture, and construction. Another use of point clouds is terrain modeling. Utilizing LiDAR, accurate representations of land can be accomplished with minimal time and effort. These point clouds are also used in augmented/virtual reality as the method to represent objects. Point cloud datasets do not contain any information about the represented objects surfaces or curves but converting it into a mesh can be beneficial. A mesh allows for the model to be accurately measures and as well as allowing for features to be edited and manipulated. A method that can convert a point cloud into a mesh is Delaunay Triangulation.

Delaunay Triangulation

This method converts the set of points into a triangulated network in a controlled manner. Since a set of points can be joined by triangles in numerous ways, Delaunay triangulation determines the optimal representation of a triangular mesh. The process relies on a set of “Delaunay criterion”, that specify geometric rules that determine if a pair of triangles are the best choice. The process is able to choose "good" connections between vertices. The figure below shows a random triangulation on the left where triangles were created based on the order of the points and there was no optimization of the configuration during the assembly process. The right side is an example of Delaunay triangulation. Also, the Delaunay set consists of triangles that are much more equilateral rather than long and narrow like the random set of triangles. This has two benefits. For one, the Delaunay triangles are much more aesthetic to the viewer but more importantly, the triangles produce more gradual changes in slope and have better continuity as the edge transitions are smoother.

Chart, radar chart

Description automatically generated

*Figure # - Random vs Delaunay Triangulation*

Delaunay Criterion

The foundation of Delaunay Triangulation relies on the use of circumcircles generated from each triangle. Three non-collinear points will have an associated circumcircle. The first criterion is that all triangles are non-degenerate, i.e. no three collinear points can be combined to make a triangle. The next criterion states that for a triangle to be considered, the corresponding circumcircle must not contain any other points inside it. Because of this, another property emerges. This property states that the Delaunay triangulation maximizes the minimum angle. This created a ‘flipping’ technique. Looking at the figure below with △ABD and △BCD, if the sum of ⍺ and 𝛾 is less than 180°, the triangles satisfy the Delaunay criterion. If not, the common edge can be ‘flipped’ and the two new triangles will meet the criterion.

Diagram, shape

Description automatically generatedShape, polygon

Description automatically generatedA picture containing green, accessory, colorful

Description automatically generated

*Figure # - Process of ‘Flipping’*

Implementation Algorithms

There are various algorithms that have been developed to compute the Delaunay triangulation. The most basic algorithm is the incremental algorithm. This method starts by creating a ‘super triangle’ that encompasses all of the points. Then one by one a point is inserted, which creates a set of three new triangles. Then the circumcircle test is done on each triangle to determine if any other points contained in it. If so, the edge not containing the new point is ‘flipped’. This process is repeated for each of the triangles and then a new point is added. This algorithm can take O(N^2), where N is number of points, time but it is possible for it to approach O(NlogN) if the points are chosen at random.

Another algorithm that is utilized in Delaunay triangulation is called ‘Divide and Conquer’. This algorithm first splits the data points into equal subsets that can be triangulated easily with a simple method. Then, each subset is merged, creating new triangles by connecting each vertex to its nearest neighbor on the other boundary. This is done for each subset until all the points are triangulated. This method takes O(NlogN) time and is better suited for large datasets but can run into issues if the dataset is highly irregular with high density variation as there is a higher probability for the subsets to become unbalanced and lead to slow convergence.

Implementation

There are numerous libraries that have built-in functions to compute the Delaunay Triangulation. I will create my own functions that will compute the triangulation in 2D and 3D space.

The first function is for 2D Delaunay Triangulation using the Incremental Algorithm. It takes in a list of tuples and returns another list of tuples that represent the edges of the triangulation. Each tuple contains two elements for the x and y coordinates. The 3D implementation is only adjusted by using tetrahedron rather than triangles.

Chart

Description automatically generated

*Figure # - 2D Triangulation Using Incremental Algorithm*

The SciPy package contains its own Delaunay Triangulation implementation. It relies on using the Qhull library. The Qhull implementation computes the Delaunay triangulation by finding a convex hull. It first computes the convex hull of the input dataset and then triangulates the convex hull using an ear clipping algorithm. Then any triangles that are not inside the convex hull are removed. Finally, all triangles’ edges are checked if they satisfy the Delaunay criterion, and if not, they are flipped.

Chart

Description automatically generated

*Figure # - 2D Triangulation Using Qhull Algorithm*

Comparing the two methods, they result in the same triangulation. This validates that my function generates the correct Delaunay triangulation. However, there is a large difference in the efficiency. My implementation took 18.92 seconds while the SciPy method only took 0.052 seconds. This is a drastic difference between the two methods. The Qhull library is much more robust and utilizes optimes algorithms and is much more efficient.

Comparing the 3D functions, similar findings are observed as seen in Figure #. The results are identical but the time to complete are very different. Once again, the Qhull method was faster, completing in 0.067 seconds, while my method tool over 91.7 seconds.

Chart, surface chart

Description automatically generatedChart, surface chart

Description automatically generated

*Figure # - identical Outputs for Qhull and my Delaunay functions.*

The above output consisted of 64 data points. Raising the number of input points to 125 results in the following graph. It is a much smoother graph since the point density is much higher. Trying to run the larger dataset in my function took over 306.9 seconds, compared to 0.135 seconds in Qhull.

Chart, surface chart

Description automatically generated

*Figure # - Denser point density*

Future Work

The main limitation of my implementation of Delaunay triangulation was the inefficiency and time limitation with larger sets of points. At 125 points, the function was taking over 5 minutes to run. IN real world applications, datasets will contain hundreds or thousands of sets of points. Using my current functions would not be advised as the time requirements would be very large. With further work, my functions could be optimized by possibly using other algorithms as discussed above. Also, the algorithms used could be parallelized to reduce the barrier that large dataset create. [parallel]

**References**

[parallel] <https://john.cs.olemiss.edu/~rhodes/papers/Nguyen18.pdf>

[1] <https://gwlucastrig.github.io/TinfourDocs/DelaunayIntro/index.html>

[2] <https://resources.system-analysis.cadence.com/blog/msa2022-conquering-3d-delaunay-triangulation>

[3] <https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Delaunay.html>

[4] <http://www.qhull.org/html/qdelaun.htm>