

**CS 478**

**Computational Geometry**

Progress Report

*Three Delaunay Triangulation*

**Group Members**

Ahmet Faruk Ulutaş - 21803717

Mustafa Azyoksul - 21501426

15.03.2022

# Progress Report of Project

In the current part of the project, we have done the necessary research about the Delaunay triangle, the Randomized Incremental, and Divide and Conquer algorithms we will use. In the implementation part, we determined which data types and libraries we will use. We have prepared the general template of our code that we will present in the graphical user interface. Also, we studied algorithms and prepared our pseudo-codes. After that, we will implement the algorithms and complete our project with minor corrections.

# Survey of the Subject Area

Triangulation of a series of P points and ensuring that no vertices lie within the circumcircle of any triangle is Delaunay Triangulation (DT) [1]. All the angles of these triangulations are balanced minima. To keep the circumcircle area small, it prevents the formation of skinny triangles whose height is too great from the base. DT cannot be done when there are points on the same line. It is possible to generalize for non-Euclidean metrics. DT can be done on a quadrilateral, but triangulation is not unique. This is an important feature because using the flip technique, other options can be checked by changing the common side of the triangle that does not meet the DT. In these cases, it cannot be known that a DT exists or is unique [2].

A dual graph of Voronoi tessellation can be constructed using DT. To create this graph, DT is done first. Then the center points of all circumscribed circles in this triangulation are taken. These center points are combined with other points.

With the number of points n and the number of dimensions d;

* The union of all the simple triangulations is the convex hull of the points.
* Delaunay triangulation has at most O(n^(d/2)) simplicity.
* With the Euler property, if there are b vertices on the convex hull, then any triangulation of points has at most 2n − 2 − b triangles plus an outer face.
* In the plane, each vertex has on average six surrounding triangles.
* If a circle passing through two of the input points contains no other than these in its interior, the line segment joining the two points is an edge of the Delaunay triangulation of the given points.

Delaunay triangulations are often used to construct meshes for the finite element method because of the angle guarantee and when fast triangulation algorithms are developed [3].

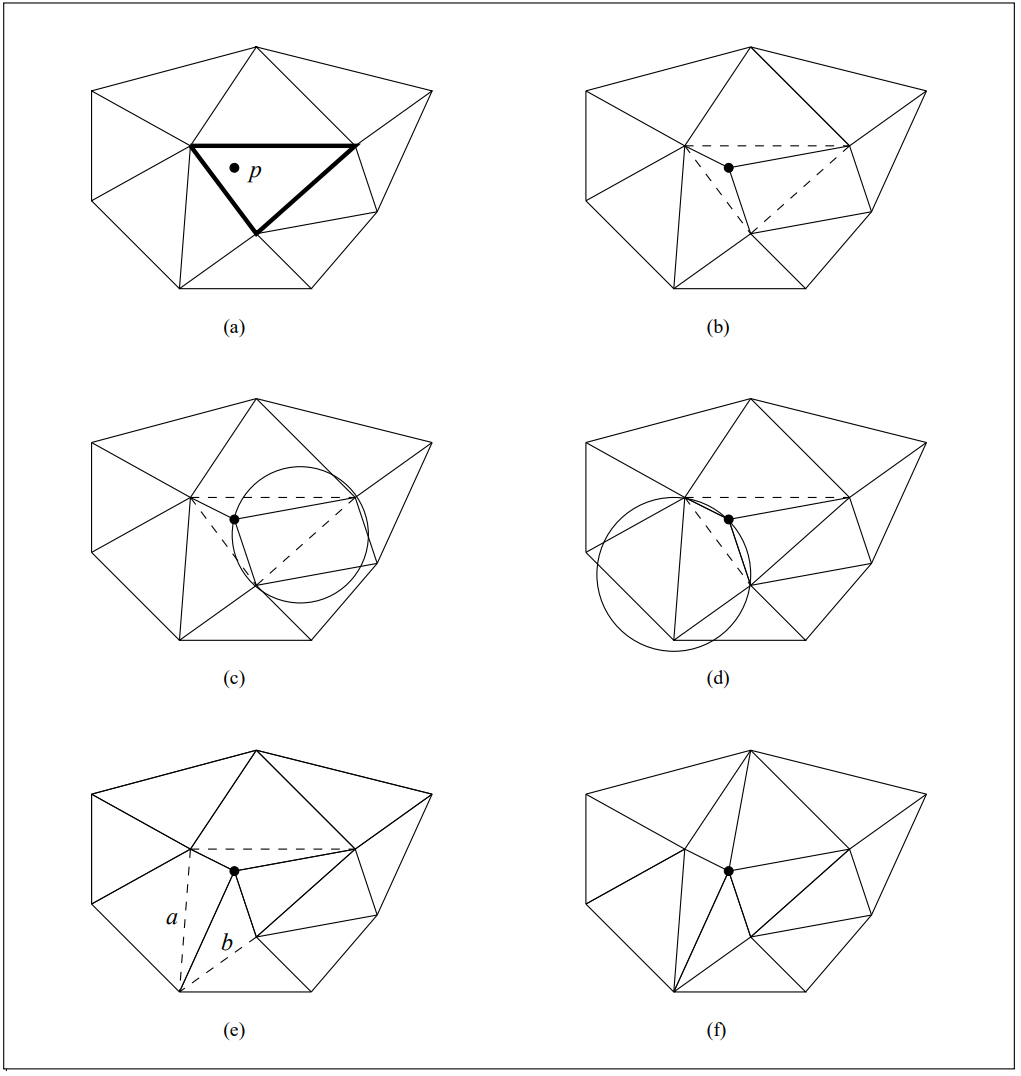
# Algorithms

## 1. Randomized Incremental Algorithm

The incremental DT algorithm starts with a triangle large enough to contain all of the points in the input. Points are added into the triangulation one by one, maintaining the invariant that the triangulation is Delaunay[4].

### Algorithm

1. Select a new point P (1a)
2. Find the triangle in the current triangulation that contains P (1a)
3. Connect P to the vertices of this triangle (1b)
4. Add old edges of the triangle into a candidate list (1b)
5. Repeat until the candidate list is empty
   1. Draw the circumcircle of the triangle formed by the candidate edge and P (1c)
   2. If no point is in the circumcircle (1c)
      1. Continue loop
   3. If there is a point Q inside the circumcircle (1d)
      1. Swap the candidate edge with the edge [PQ] (1e)
      2. Add edges from the ends of candidate edge to Q to candidate list (1e)
6. End (1f)



*Figure 1 [4]. Inserting a point into the triangulation. Dashed lines show the candidate edges.*

### Pseudo-Code

procedure RandomizedIncrementalAlgorithm( PointList)

begin

CandidateList = []

edgeList = []

P = Select a new point from PointList

Find the triangle in the current triangulation that contains P

Connect P to the vertices of this triangle

CandidateList.add( old edges of the triangle)

while ( CandidateList is not empty)

circleArea = draw the circumcircle of the triangle formed by the candidate edge and P  
 if ( points not in the circleArea)

continue

endif

Swap the candidate edge with the edge [PQ]

Add edges from the ends of candidate edge to Q to candidate list

endwhile

end

### Analysis

In the worst case the insertion of a point can require O(n) edges to be flipped. However, in practice the average number of edges tested per insertion is small (< 9). Guibas, Knuth, and Sharir have shown that if the insertion order is randomized, the expected time is O(1) per insertion [5].

Locating the containing triangle can be done in an optimal O(log n) time, but this requires maintaining complicated data structures. Alternatively, the triangle can be located by starting from an arbitrary place in the triangulation and moving in the direction of p until the containing triangle is reached. This requires O(n) time, but if the inserted points are uniformly distributed, the expected number of operations to locate a point is only O(n1=2 ). A simple improvement is to always resume the search from the triangle that was found last: in this way, when the points to be located are near each other, the containing triangles are determined quickly[4].

## 2. Divide and Conquer Algorithm

In the divide and conquer algorithm, a recursive line is drawn to divide the vertices into two sets. The Delaunay triangulation is then calculated for each dividing cluster and the clusters are joined along the dividing line.

### Algorithm

1. Put all of the points into order of increasing x-coordinates. If two points have the same x-coordinate, one with lower y-coordinate comes first.
2. Divide the set into subsets by halving the set of points recursively until each group has at most 3 points.
3. Trivially triangulate the subsets of at most 3 points.
4. Run the recursive stack back and merge the subsets in each step.

#### Merge procedure

Let’s call edges formed on the left side LL, edges formed on the right side RR and edges formed by connecting one edge from each side LR.

The first step for merging the two halves is to insert the base LR-edge. The base LR-edge is the bottom-most LR-edge which does not intersect any LL or RR-edges.

Working upward, we need to determine the next LR-edge to be added just above the base LR-edge. Clearly, such an edge will have, as one endpoint, either the left or right endpoint of the base LR-edge. The other endpoint, then, will come either from the left or the right point subset. Naturally, we narrow our decision by selecting two candidate points: one from the left subset and one from the right subset.

On the right side, the first potential candidate is the point connected to the right point of the base LR-edge by the RR-edge which defines the smallest clockwise angle from the base LR-edge. Similarly, the next potential candidate defines the next smallest clockwise angle from the base LR-edge.

The potential candidate is then checked for the following two criteria:

1. The clockwise angle from the base LR-edge to the potential candidate must be less than 180 degrees.
2. The circumcircle defined by the two endpoints of the base LR-edge and the potential candidate must not contain the next potential candidate in its interior.

If both criteria are satisfied, the potential candidate becomes our final candidate for the right side. If the first criterion does not hold, then no candidate for the right side is chosen. If the first criterion holds but the second does not, then the RR-edge from the potential candidate to the right endpoint of the base LR-edge is deleted. The process is then repeated with the next potential candidate as the potential candidate until a final right candidate is chosen or it is determined that no candidate will be chosen.

As for the left candidate selection, the process is just the mirror image of that for the right.

When neither a right nor a left candidate is submitted, the merge is complete. If only one candidate is submitted, it automatically defines the LR-edge to be added. In the case where both candidates are submitted, the appropriate LR-edge is decided by a simple test: if the right candidate is not contained in the interior of the circle defined by the two endpoints of the base LR-edge and the left candidate, then the left candidate defines the LR-edge and vice-versa. By the guaranteed existence of the Delaunay triangulation (here applied to only four points), at least one of the candidates will satisfy this; by the uniqueness of the Delaunay triangulation, only one candidate will satisfy this (except in the case when the four points are co-curricular).

Once the new LR-edge is added, the entire process is repeated with the new LR-edge as the base LR-edge.

Finally, once the last two halves (those that resulted from the first "halving" of the point set) are merged, the Delaunay triangulation is complete [6].

### Pseudo-Code

### procedure DivideAndConquer( P: point\_set, AFL: (d-1)face\_list, d-simplex\_list)

var f: (d-1)face; AFL0, AFL1, AFL2: (d-1)face\_list;

t: d-simplex; sum: d-simplex\_list; a: splitting\_plane;

begin

AFL0, AFL1, AFL2 = emptylist

Pointset\_Partition( P, a, P1, P2)

/\* Simplex Wall Construction \*/

if AFL is empty then

t = MakeFirstSimplex( P, a)

AFL = (d-1)faces(t); Insert(t, sum);

for each element of AFL do

if IsIntersected(f,a) then Insert(f, AFL0)

if Vertices(f) is subset of P1 then Insert(f, AFL1)

if Vertices(f) is subset of P2 then Insert(f, AFL2)

while AFL0 is not empty do

f = Extract(AFL0)

t = MakeSimplex(f, P)

if t is not null then

sum = sum union t;

for each f’ is element of (d-1)faces(t) AND f’ is not equal f do

if IsIntersected(f’,a) then

if Member(f’, AFL0) then Delete(f’, AFL0)

else Insert(f’, AFL0)

if Vertices(f’) is subset of P1 then

if Member(f’, AFL1) then Delete(f’, AFL1)

else Insert(f’, AFL1)

if Vertices(f’) is subset of P2 then

if Member(f’, AFL2) then Delete(f’, AFL2)

else Insert(f’, AFL2)

/\* Recursive Triangulation \*/

if AFL1 is empty then DivideAndConquer(P1, AFL1)

if AFL2 is empty then DivideAndConquer(P2, AFL2)

end

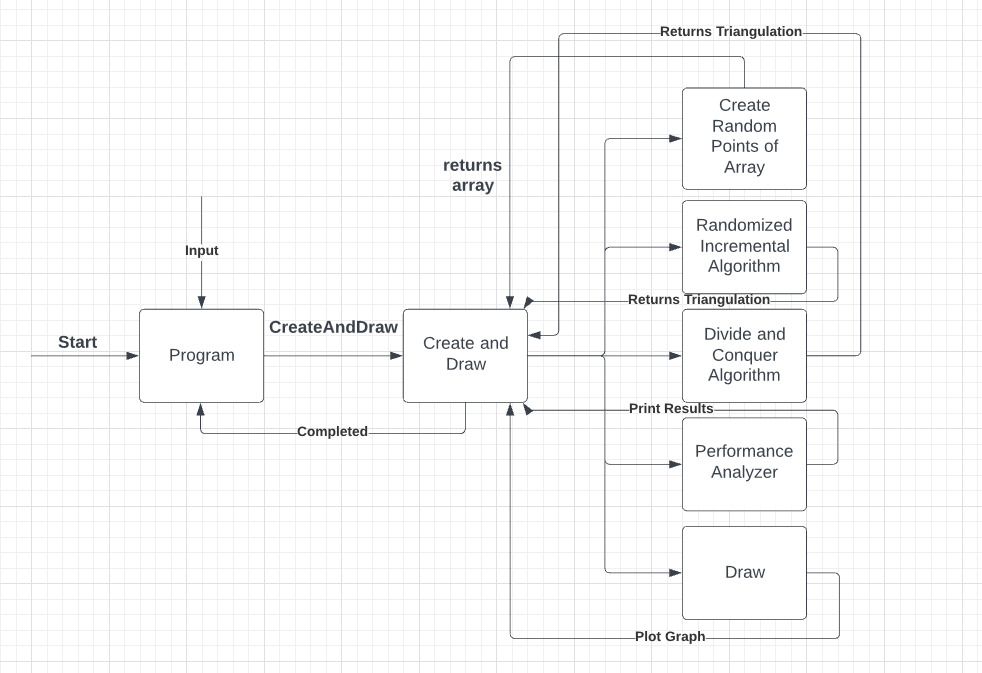
### Analysis

Merge operations can be developed and completed in O(n) time. Thus, the total running time is O(n log n). For certain clusters of points, such as a uniform random distribution, the expected time can be reduced to O(n log log n) by choosing the split lines wisely, i.e. continuously rotating the split axis by 90 degrees, while maintaining worst case performance [3,7].

# Implementation Details

Since we will use python as a programming language, we will make use of structures and libraries in python in our project. As data structure, we will use NumPy's arrays because they can perform fast processing. It also integrates with the matplotlib library that we will use for the illustrator. We chose it since it can work stably with Tkinter in the matplotlib library. We will also use the Tkinter library to provide the graphical user interface.

# Block Diagram



# References

[1] S.T.E.V.E.N. FORTUNE, “Voronoi diagrams and Delaunay triangulations,” Lecture Notes Series on Computing, pp. 225–265, 1995.

[2] B. Delaunay, “Sur la sphère vide, Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestven- nykh Nauk,” pp. 793–800, 1934.

[3] Leach, Geo. "Improving worst-case optimal Delaunay triangulation algorithms." *4th Canadian Conference on Computational Geometry*. Vol. 2. 1992.

[4] Cignoni, Paolo & Montani, Claudio & Scopigno, Roberto. (1998). DeWall: A fast divide and conquer Delaunay triangulation algorithm in E d. Computer-Aided Design. 30. 333-341. 10.1016/S0010-4485(97)00082-1.

[5] Lischinski, Dani. “Incremental Delaunay Triangulation” Graphics Gems IV. Paul S. Heckbert (Ed.). Academic Press Prof., Inc., San Diego, CA, USA. 1994. Page 47-59. http://www.karlchenofhell.org/cppswp/lischinski.pdf doi: https://dl.acm.org/citation.cfm?id=180900

[6] L. J. Guibas, D. E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. In Proc. 17th Int. Colloq. Automata, Languages, and Programming, volume 443 of Springer-Verlag LNCS, pages 414{431, Berlin, 1990. Springer-Verlag.

[7] S. Peterson, “Computing Constrained Delaunay Triangulations,” *COMPUTING CONSTRAINED DELAUNAY TRIANGULATIONS IN THE PLANE*. [Online]. Available: http://www.geom.uiuc.edu/~samuelp/del\_project.html#:~:text=The%20divide%20and%20conquer%20algorithm,by%20their%20y%2Dcoordinates). [Accessed: 22-Mar-2022].