Milestone III

Adnan Farid, Edward Guthrie, Armando Albornoz

---

1. Introduction

Delaunay triangulations are a fundamental geometric structure widely used in computational geometry, computer graphics, mesh generation, and in many other applications. Much of their popularity stems from their optimal angular properties: the Delaunay triangulation of a point set maximizes the minimum angle among all possible triangulations. This property, first proven by Sibson [1], helps in the process of creating triangulations with triangles that are “well-shaped” which is a highly desirable property in many applications. Notice, however, that even in Delaunay triangulations, poorly shaped triangles can still occur based on the point set utilized to build the triangulation. To address this problem, refinement techniques such as the insertion of additional points have been developed to improve the quality of the poorly shaped triangles.

Refinement through point insertion is a technique utilized for improving the quality of Delaunay triangulations. By adding points in certain locations, it is possible to improve the angle distribution, and achieve a more uniform triangulation. This process, known as Delaunay Refinement, has been well-studied for order-1 Delaunay triangulations. However, the role of refinement in higher-order Delaunay triangulations remains largely unexplored. We aim to explore the angle optimization properties of order-2 Delaunay Delaunay triangulations and the role of refinement in achieving these properties.

---

2. Problem Definition

- **2.1. Main Problem**: The main problem we want to address in this work is the following: How can refinement optimize angles in Delaunay triangulations of order 2. In particular we want to investigate the following:
    - The foundations of angle optimization in order-2 Delaunay triangulations.
    - The role of refinement in improving the quality of order-2 Delaunay triangulations by improving the angle distribution of the triangulation.
- **2.2. Sub-Problems**:
    - How can refinement optimize angles in Delaunay triangulations of order k. This is a more difficult problem since we are not sure if order k Delaunay triangulations, for k greater than 2, optimize their angle vector.
    - Implement an incremental algorithm that constructs order-2 Delaunay triangulations from a set of points.
    - Implement an incremental algorithm that constructs order-k Delaunay triangulations from a set of points.
    - Implement a randomized incremental algorithm for order-2 Voronoi Diagrams utilizing the algorithm given in [2].

---

3. Literature Review

1. **"A Simple Algorithm for Higher-Order Delaunay Mosaics and Alpha Shapes [3]"**
    a. **Introduction**
    This paper by Herbert Edelsbrunner and Georg Osang presents a novel and simple algorithm for computing higher-order Delaunay triangulations of any order and dimension. The algorithm provided differentiates itself from its counterparts due to its simplicity and ease of implementation. It utilizes a weighted first-order

Delaunay mosaic algorithm as a black box, and otherwise utilizes only combinatorial operations. We make use of the following implementation of the algorithm [4].

The authors explain the complexity of the number of vertices of the order-k Delaunay triangulation. They mention that such a complexity is well known in the plane; see [5, 6], and it is $\Theta(k(n-k))$. For dimensions higher than 2, this number varies significantly and varies a great deal depending on the distribution of the provided points. There exists, however, an upper bound of on the total $O(k^{\lceil \frac{d+1}{2} \rceil} n^{\lfloor \frac{d+1}{2} \rfloor})$ size of the first k higher-order Delaunay mosaics is known to be tight [7], while the lower bound $\Omega(k^d n)$ is
only conjectured; see [8]. If we consider the complexity for individual Delaunay Triangulations, we have that there are not many known results. The authors use an implementation of the algorithm and the results given above to perform an experimental study on the size of order-k Delaunay triangulations.

b. **Methods, Definitions and Techniques Used**::

The algorithm heavily makes use of the *rhomboid tiling [9]*. Below is an in depth explanation of it.

Let X be a locally finite set in R^d. If we assume that such a set is in general position, there exists a rhomboid tiling in R^{d+1} such that the Delaunay mosaics are horizontal slices of the tiling

Definition (Ordered Three Partition). Every (d-1)-dimensional sphere, S, in R^d partitions X into the points, inside, on, and outside S. We call this the ordered three partition of X, and we denote it as
$X = InS \cup OnS \cup OutS$

Now, we will map each order-3 partition defined by a (d-1)-sphere S, to a parallelepiped $Q \subseteq X$ which we call the rhomboid of S. In order to

achieve this we define $y_x = (x, -1) \in R^{d+1}$ for every x in X and

$$y_Q = \sum_{x \in Q} y_x$$

for every $Q \subseteq A$. The (d+1)-st coordinate of y_Q is named the depth.

Definition (Rhomboid of S).

The rhomboid of S is the following set:

$$rho(S) = conv\{y_Q : InS \subseteq Q \subseteq InS \cup OnS\}$$

Now, given these definitions the rhomboid tiling is just the collection of all rhomboids defined by spheres.

Knowing this we have that each cell of the order-k Delaunay mosaic is a slice of some rhomboid. The authors argue about this property in the following manner: “Combinatorially, each rhomboid is a cube and, again combinatorially each cell of $Del_k(X)$ is a slice orthogonal to the cube diagonal that passes through a nonempty set of vertices.” From this we have that for the (d+1)-cube, there d+1 such slices, which are indexed by the generation $0 \leq g \leq d + 1$.

Besides the rhomboid tiling the algorithm also makes use of weighted Delaunay triangulations. We know by [10] that the order-k Delaunay triangulation is the projection of the boundary of a convex polyhedron in R^{d+1}. This construction works as follows: first, lift the points $x \in X$ to $(x, ||x||^2)$. For each k-tuple $Q \subseteq X$, we take the barycenter of their lifts, and obtain the order-k Delaunay mosaic as the projection of the lower faces of the convex hull of these barycenters. Now,

Now, after this, the authors go on and prove a result of how, given a cell of an order-k Delaunay mosaic, one can find the rhomboid that cell is a slice of. Now, we can equivalently think of each of these barycenters as a weighted point in R^d and get the order-k Delaunay mosaic as the weighted order-1 Delaunay mosaic of these weighted points.

The paper now introduces a problem which we do not face: In $d \geq 3$, not all cells of the Delaunay triangulation are simplicial. The authors mention that being able to obtain these cells is necessary and this problem poses a challenge to the methods mentioned above since most algorithms that compute convex hulls or weighted first-order Delaunay mosaics return a triangulated version of the correct mosaic. The authors solve this problem by predicting the cells from the corresponding rhomboids.

The authors now prove that if we have a triangulation of the order-j Delaunay mosaic, for all all $j \leq k$, we can assemble the complete vertex set of the order-k Delaunay mosaic by taking slices at depth k obtained from first-generation cells at lower depths.

Finally, the authors present the algorithm in section 4 of the paper and, perform an experimental study in section 5 where they try to shed light on the size of the mosaic for dimension $d \geq 3$.

c. **Applicability**: This paper is highly useful to us since it provides us with an implementation of an algorithm that is capable of computing order-2 Delaunay mosaics. This algorithm is not perfect for our applications however since it does not appear to be incremental and it deals with complicated cases, such as cells not being simplices, that arise in $d \geq 3$. Since we are working in the plane these problems are of no concern to us

and there is still a possibility for us to simplify this algorithm, make it incremental and only work in the plane.

2. "ORDER-2 DELAUNAY TRIANGULATIONS OPTIMIZE ANGLES [11]"

a. **Introduction:**

The main motivation of this paper is to generalize optimal properties from order-1 to higher-order Delaunay triangulations. Order-1 Delaunay triangulations have the property that for any edge shared any two triangles, the sum of the two angles opposite to the edge is less than π. This property is known as the *local angle property.* Due to the potential usefulness of this result in applications, Lawson in 1977 turned the empty circle criterion into an iterative algorithm that turns a triangulation of a set of points into the Delaunay triangulation using $O(n^2)$ edge flips [12]. The correctness of this algorithm implies that the Delaunay triangulation is the unique triangulation that satisfies the local angle property. Later, Sibson proved in 1978 that among all the triangulations of a point set, the Delaunay triangulation lexicographically maximizes the sorted angle vector [13]. Now, the authors state that there is a scarcity of optimality properties known for higher-order Delaunay triangulations. They provide the following contributions:

1. They extend the local angle property from order-1 to order-k, for $1 \leq k \leq n-1$.
2. They prove that among all complete level-2 hypertriangulations of a finite generic set in $\Re^2$, the order-2 Delaunay triangulation lexicographically maximizes he sorted angle vector;

3. They show that among all maximal level-2 hypertriangulations of a finite generic set in $\Re^2$, the order-2 Delaunay triangulation is the only one that has the local angle property.

We know that for ordinary triangulations, the proof of properties 2 and 3 follow from the existence of a sequence of edge-flips that connects any initial triangulation to the order-1 Delaunay triangulation, in such a way that each edge-flip lexicographically increases the sorted angle vector. Now, level-2 hypertriangulations are connected by flips, see [14], but there exists cases in which the flips lexicographically decrease the sorted angle vector. This is a problem since the relation between the local angle property and the angle vector is not clear.

b. **Methods, Definitions and Techniques Used**: There are several definitions that require some attention that the authors present. Let $A \subset \Re^2$, be in general position.

Definition (Complete, Maximal, and Partial Triangulation). A triangulation P, of A is an edge-to-edge subdivision of the convex hull of A into triangles whose vertices are points in A. We identify it with its triangles, so we write $P = \{t_1, t_2, \ldots, t_m\}$. The triangulation is called complete if every point of A is a vertex of at least one triangle, partial if it is not complete, and maximal if there is no other triangulation of the same points that subdivide it.

For a set of k of points, I, we write [I] for the average of the points and, assuming $a \notin I$ and $I \cap J = \emptyset$, we write [Ia] and [IJ] for the averages of $I \cup \{a\}$ and $I \cup J$, respectively. We sometimes call [I] a label.

Definition (Hypertriangulations). Let n = |A|, and k an integer between 1 and n-1, and $A^{(k)} = \{[I] \mid I \subseteq A, |I| = k\}$ be the set of k-fold averages of the points in A. We have that a level-k hypertriangulation of A is a possibly partial partial triangulation of $A^{(k)}$ such that every edge with endpoints [I] and [J] satisfies $|(I \cap J))| = k - 1$.

We now define the aging function. First we need some preliminary definitions. A triangle in a level-k hypertriangulation is classified in two types. Let [I], [J], [K] be its vertices. We say the triangle is black if $|(I \cap J \cap K) = k - 2|$ and white if $|(I \cap J \cap K)| = k - 1$. The next function allows the transformation of a white to a black triangle.

Definition (The Aging Function). Letting t be a white triangle with vertices [Ya], [Yb], [Yc], the aging function F maps t to the black triangle [Yab], [Yac], [Ybc]. The aging function increases the level of the triangle by one.

Now, we say that a level-k hypertriangulation, P_k, ages to a level-(k+1) hypertriangulation P_{k+1}, if the aging function defines a bijection between the white triangles in P_k and the black triangles in P_{k+1}.

Now, the paper compiles several results relating level-1 and level-2 hypertriangulations. It also mentions that Delaunay triangulations are optimal among all complete triangulations, but not necessarily among the larger family of partial triangulations. The authors then go on and introduce a family of level-2 hypertriangulations to which we compare the order-2 Delaunay triangulation.

Definition (The Aging Function) (Complete and Maximal Level-2 Hypertriangulations). A level-2 hypertriangulation of A is complete if its

black triangles are images under the aging function of the triangles in a complete triangulation of A, and it is maximal if no other level-2 hypertriangulation subdivides it.

We know already that for $k = 1$ a triangulation of A is complete if and only if its maximal. Now, for $k > 1$ things are more complicated. We have that complete implies maximal.

After these definitions the authors focus on extending the local angle property from order-1 Delaunay triangulations to higher-order Delaunay triangulations, specifically order-2 Delaunay triangulations. They also prove that the order-2 Delaunay triangulation satisfies the local angle property and discuss its implications. Next, the authors focus on proving the angle vector optimality of the order-2 Delaunay triangulation. Specifically, they show that among all complete level-2 hypertriangulations, the order-2 Delaunay triangulation lexicographically maximizes the sorted angle vector. They utilize the local angle property just proved in order to achieve these results. Finally, the authors prove that the order-2 Delaunay triangulation is the only maximal level-2 hypertriangulation that satisfies the local angle property. This result is based on the maximality and completeness of the triangulation, as well as a detailed analysis of the geometric configurations of the triangles and edges.The uniqueness of the local angle property ensures that the order-2 Delaunay triangulation is uniquely characterized by its optimal angle distribution, making it the optimal triangulation in terms of angle quality.

c. Applicability

This paper is relevant to our work on angle optimization since it extends the local angle property from order-1 to order-2 Delaunay triangulations. This property ensures that the sum of the angles opposite any shared edge in the triangulation

is less than π, which is crucial for avoiding "skinny" triangles and ensuring well-shaped triangles. This directly aligns with our goal of optimizing angles in Delaunay triangulations through refinement. Moreover, it provides us with theoretical results that may be of help when we decide on concentrating on refinement techniques for order-2 Delaunay triangulations.

3. **"A new duality result concerning Voronoi diagrams [10]"**

   a. **Introduction**:

   This paper by Franz Aurenhammer provides a new duality between order-k Voronoi diagrams in E^d and convex hulls in E^{d+1}. This new duality is a generalization of an already known duality between order-1 Voronoi diagrams and it provides a simple algorithm to compute the order-k Voronoi diagram for n points in the plane in $O(k^2 nlogn)$ time and $O(k(n-1))$ space. The primal contribution of the paper is an iterative algorithm for constructing low-order Voronoi diagrams in E^2.

   b. **Methods, Definitions and Techniques Used**

   First we need to understand what duality is: let C and C' be two convex polyhedra in E^{d+1}. Consider the faces of C and C'. We say that C and C' are dual if there exists a bijective mapping Φ between the j-faces of C and the (d-j)-faces of C' such that for any two faces f and g of C, if f is contained in g, the the dual face $\Phi(f)$ is contained in $\Phi(g)$. Notice moreover, that since polyhedral partitions of d dimensional Euclidean space can be viewed as the affine degeneracies of the boundaries of convex polyhedra in E^{d+1}; see [15]. Therefore the notion of duality is still meaningful for Voronoi diagrams. Now, let P be a point set in E^d, the authors then go on and prove there exists a point set Q_k in [redacted] such that the order-k Voronoi diagrams is dual to the lower part of the convex hull of Q_k

   .

The authors then proceed to present an algorithm for constructing order-k Vornoi diagrams by iteratively building their duals. The innovative step of this algorithm is using information about the (k-1)-th step to efficiently compute the k-th step, avoiding the problem of having to consider all k-subsets of points.

The core algorithm is the following:

- **Input:** A set P if n points in E^d
- **Output:** The order-k Voronoi diagram of P
- **Steps:**
    - Lift k-subsets of P to E^{d+1}.
    - Compute the lower convex hull of Q_k.
    - Dualize this hull back to E^d.

The only problem with this algorithm is that Q_k has $\binom{n}{k}$ points, but only a small fraction of these points contribute to the lower hull. The algorithm they presented avoids computing all of these k-subsets by iteratively refining the solution form k-1 to k,

c. **Applicability:** This algorithm is useful for us since it provides us with a framework to work with when developing an incremental algorithm for order-2 Delaunay triangulations. Moreover, notice that the optimizations utilized here are not completely necessary for us since $\binom{n}{2}$ is not big enough that the problem becomes intractable.

4. "A Review on Delaunay Refinement Techniques[16]"

a. This paper provides an overview of a large number of refinement techniques for first order Delaunay Triangulations. I believe some of these methods may be useful for us and can generalize to higher order Delaunay Triangulations.

5. **"The Quickhull Algorithm for Convex Hulls[16]"**

a. Introduction: This paper proposes the Quickhull algorithm, which computes the convex hull of a pointset in $\mathbb{R}^d$, where $d \geq 2$ and d is the dimension. It is a combination of the Beneath-Beyond algorithm and the 2D quickhull version.

b. Methods, Definition, and Techniques Used: The d-dimensional convex hull is represented by d dimensional vertices and $d-1$ dimensional faces, which are called facets. In addition, the ridges (lines in 3d) are the $d-2$ dimensional faces, which are intersections of two facets. A point p is extreme if it is on the convex hull as a vertice. Each facet contains information about the vertices that define it, the neighboring facets, and a hyperplane equation; each plane is represented by its outward pointing normal and distance from the origin. The algorithm utilizes two primitive operations: the oriented hyperplane through d points and the signed distance to a hyperplane. The distance to a hyperplane from a point is calculated by computing the inner product of the point and the hyperplanes' normal vector added to the offset. The beneath-beyond algorithm comes into play when processing a point. If H is a convex hull in $\mathbb{R}^d$ and p is a point in $\mathbb{R}^d - H$, then F is a facet of $p \cup H$ if and only if:

- F is a facet of H and p is under F, which we calculate using the mentioned primitive operations
- OR F is not a facet of H, but its vertices are p and the vertices contain a ridge of H where one incident facet is below p and the other is above p

It is important to note that due to the linkage between faces, the algorithm only needs to find one visible facet to find the rest efficiently. To achieve this, every

unprocessed point gets assigned to an outside set. Each corresponding facet is visible from the point. The process, partitioning, finds the visible facet for each point after creating a cone of new facets. If there are multiple facets from each point, choose one of the new facets. If it is below all of the new facets, discard it, as it is in H . If it is above all of the new facets, select one visible facet. Note that partitioning also records the furthest point of each outside set.

We will now describe the algorithm:

- Input: A pointset in d dimensions, where $d \geq 2$
- Output: The set of facets that define the convex hull in R^d
- Steps:
    - Create an initial simplex of d + 1 points
    - Iterate through each facet F in the initial simplex:
        - if p is above F, assign p to F's outside set, O.
    - Iterate through each F where O is not empty:
        - Select the furthest point p (partitioning) from F's outside
        - Initialize V which is the set of visible facets from p
        - Iterate through all neighbors N of facets in V
            - If p is above N, add N to V
        - For each ridge R in the boundary of V (which is the horizon ridges) H:
            - Create a new facet from R and p (essentially connecting the hole in the convex hull to the point)
            - Update the neighbors of each facet.
        - For each new facet F':
            - For each unassigned point pt in an outside set of a facet in V:

        - If pt is above F', assign pt to F''s outside set.
    - Delete the facets in V

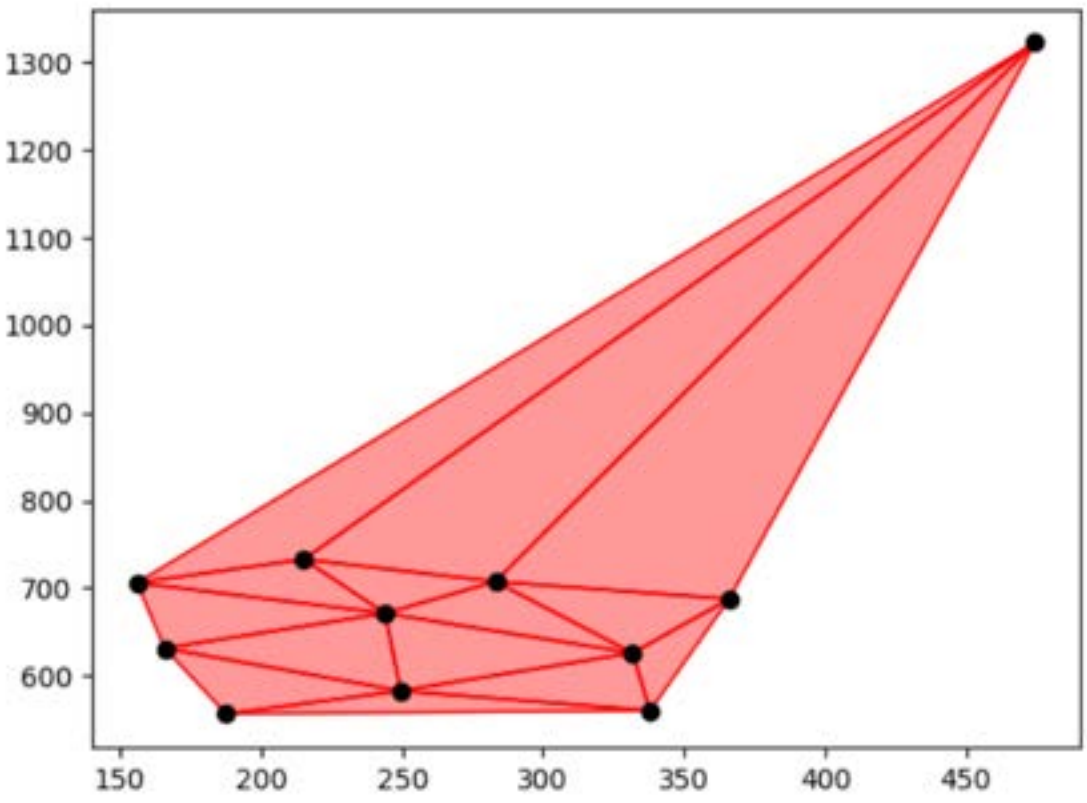
c. Applicability: Combining this algorithm with $d+1$ dimensional point projection can lead to an incremental algorithm to compute the order d Delaunay Triangulation by utilizing the barycenters of lifted points and projecting the lower hull down to d and constructing the triangulation.

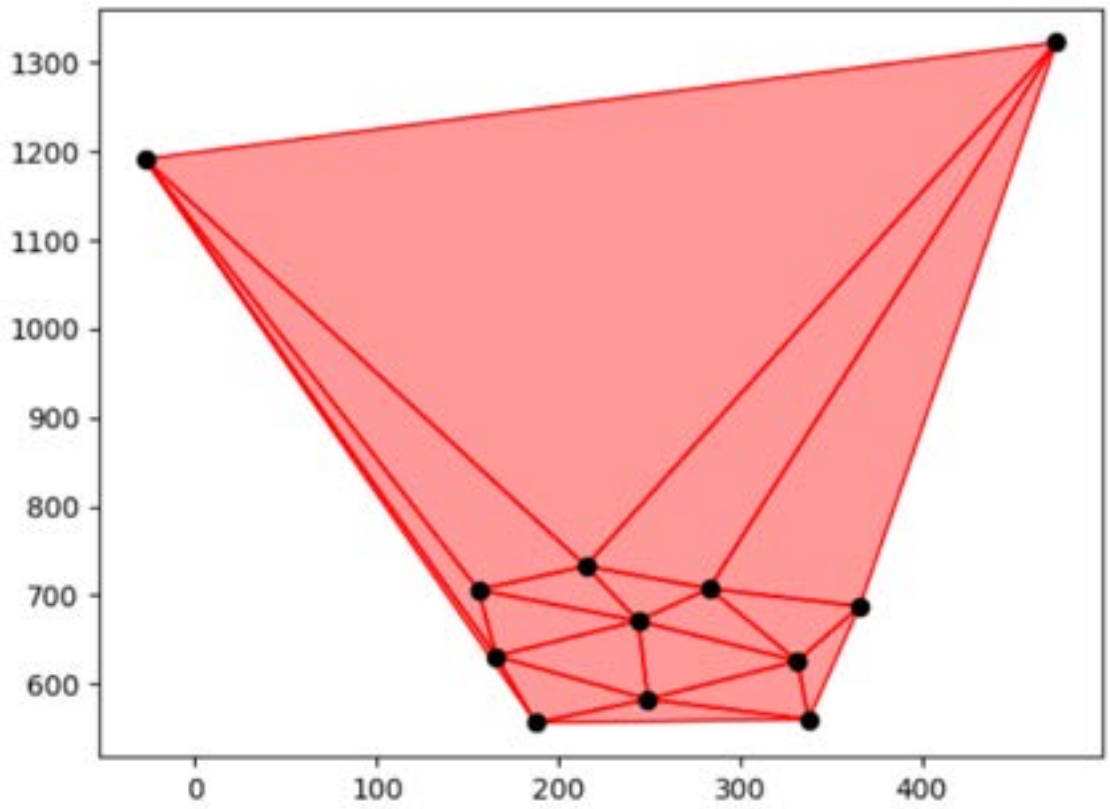
---

4. Solution Attempts

- **4.1. Delaunay Refinement Attempts:**
    - Our first attempt at refinement was to implement Ruppert's algorithm. At first the algorithm seemed to be working for small iterations, but for large iterations we noticed that the algorithm did not stop and was not producing the desired results. After discussion, we realized we had overlooked a critical detail in Ruppert's algorithm–all insertions assume a constrained Delaunay setting, whose boundary segments are carried out through the refinement. Since we treated the domain as an unbounded point set, any circumcenter that fell outside the Delaunay triangulation boundary was still inserted. This gave us two problems:
        - The algorithm did not terminate
        - Newly inserted points outside the convex hull generated triangles whose own circumcenters would lay far away from the convex hull of the input points.

Below are some figures of the erroneous algorithm:





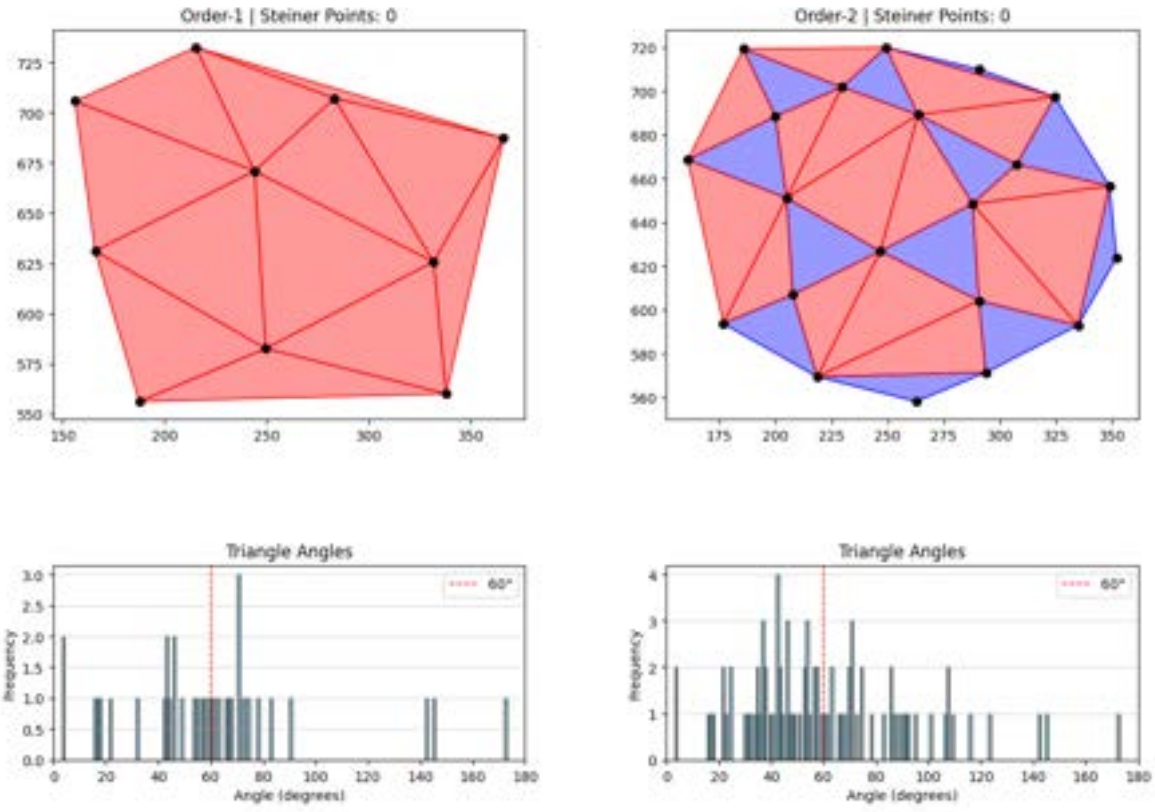
After noticing this error, we decided that instead of implementing the refinement logic ourselves it would be better to outsource the work to some library. After some thought we decided to use the Triangle library, which natively supports constrained Delaunay triangulation. Now, Triangle solved our problem of not having a constrained Delaunay triangulation in a simple manner: we just had to pass a flag to the function that performs the refinement. This flag computes the convex hull of the points, and inserts the hull edges as constrained segments.

After having implemented the refinement logic utilizing Triangle, we turned it into an animation that shows step by step how the order-1 and order-2 Delaunay triangulations and their corresponding angle histograms change as we add Steiner points to the input point set. Below are some images and a video of the results, also demoed interactively on the website. While this technique works successfully

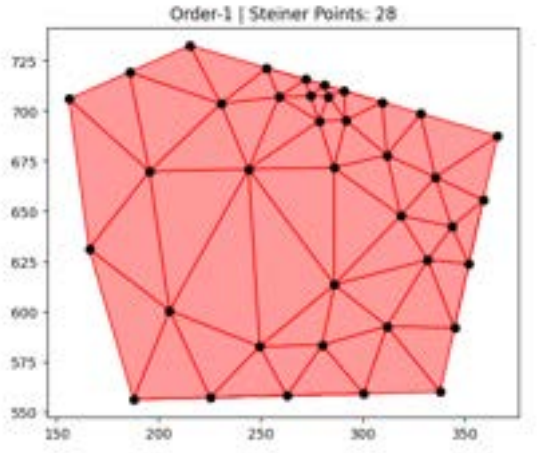
for order-1 DT, when computing the order-2 DT on the new points, the angle histogram is improved, although some bad angles still remain. Below you can see the order-1 and order-2 Delaunay triangulations of the point set

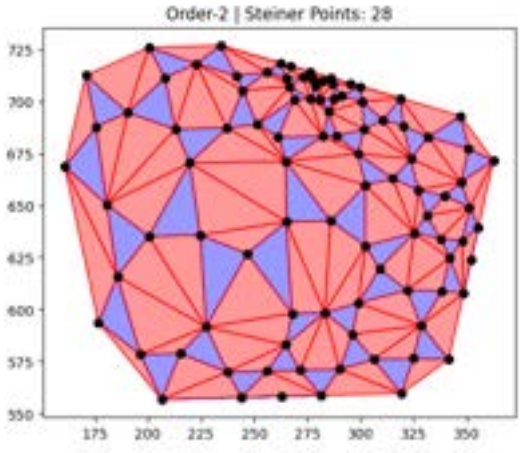
```
points = np.array([[156.006,705.854], [215.257,732.63],
[283.108,707.272], [244.042,670.948], [366.035,687.396],
[331.768,625.715], [337.936,559.92], [249.525,582.537], [187.638,556.13],
[165.912,631.197]])
```

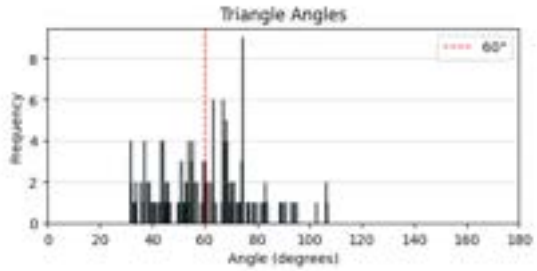
and their corresponding histograms. As it can be seen in the histograms, there are some small angles that we would like to remove

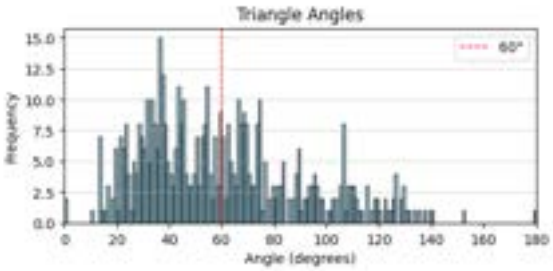


Below are the resulting triangulations after the refinement has been performed. It can be noticed that the refinement for the order-1 Delaunay triangulations has improved the angle histogram significantly, but for order-2 some small angles still remain.

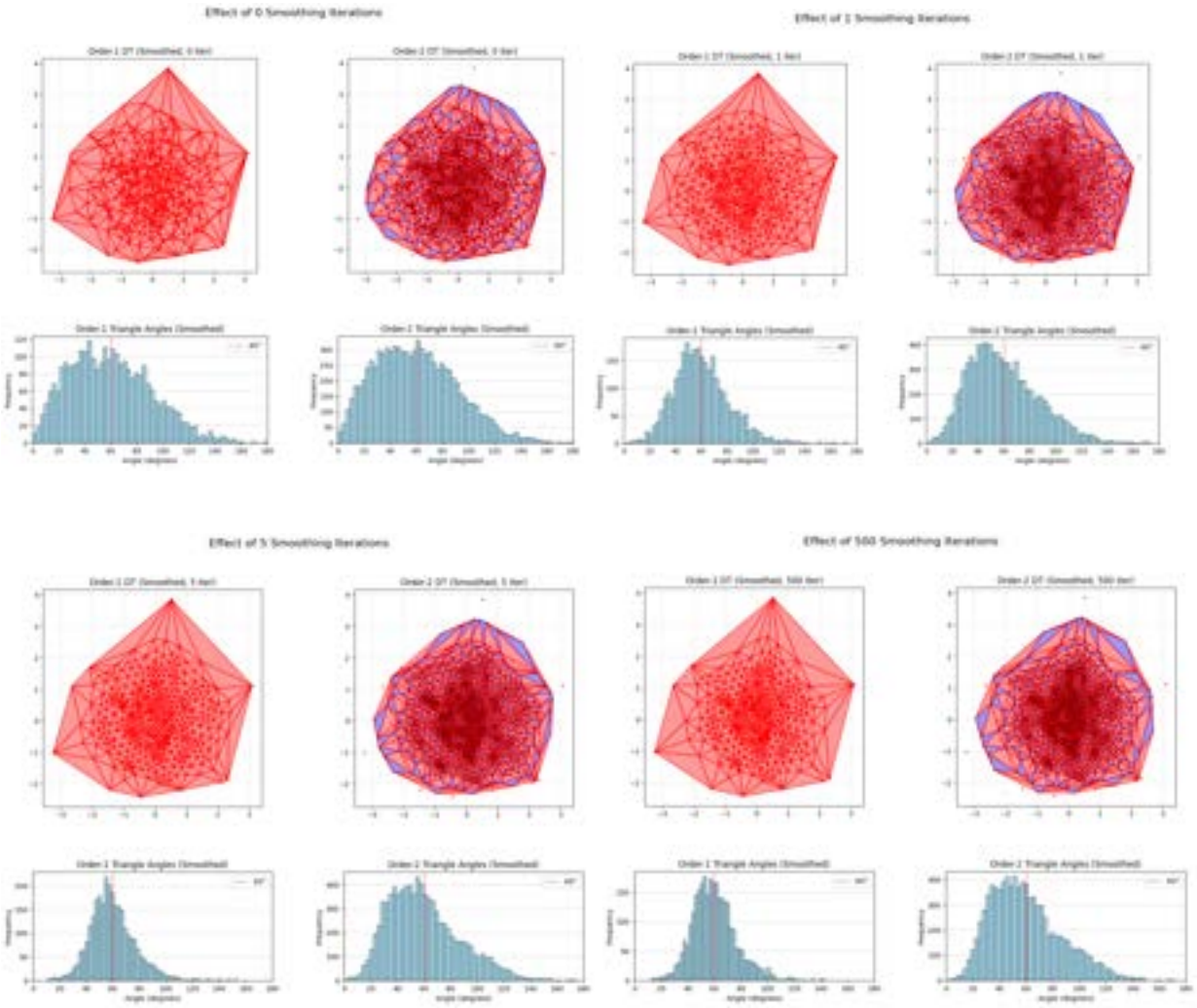


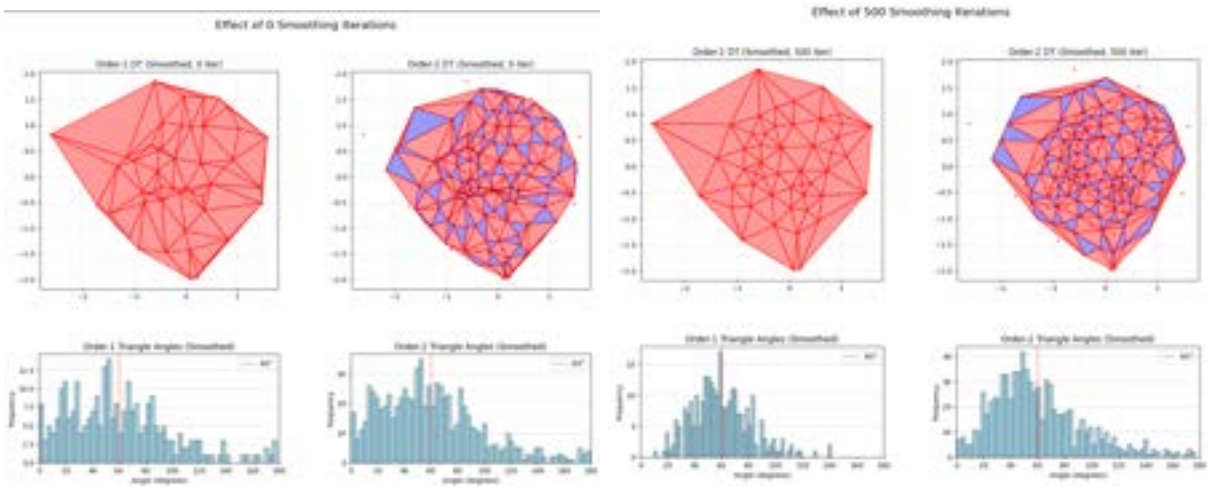






- **4.2 Mesh Smoothing:**
    - Another refinement strategy we explored was using mesh smoothing. We compute the order-1 DT on the given point set, and then move each point to the centroid of the points it is connected to in the order-1 DT, leaving the points on the convex hull boundary unchanged to prevent shrinkage. As the number of iterations increases, up to a point, the order-1 DT is much improved, and the order-2 DT is somewhat improved, although some bad angles still remain. Below is an example of this for a point set of size 500 with 0, 1, 5, and 500 smoothing iterations, as well as for a point set of size 50.





- **4.3 Incremental Algorithm Attempts:**
  - The first attempt was to utilize the rhomboid tiling algorithm. However, the rhomboid tiling algorithm is not incremental. The rhomboid tiling algorithm reuses old generation d-cells (for example order 1 triangulation) to create the order k triangulation. However, if a new point is inserted, the original d-cells are invalid, and therefore it will recalculate the k d-cells again.
  - The next attempt was to utilize the barycenter lifting technique described in [3] in section 3.1. Given a point set p in R^2, calculate the barycenters (averages: $(p1 + p2)/2$, where $p1$ is the first point and $p2$ is the second point) of $\binom{n}{2}$ where n is the number of points. After that, project those points to R^3 by mapping the z coordinate of each point to $x^2 + y^2$. However, after calculating the convex hull, all points ended up on the paraboloid in R^3 (Figure 1).

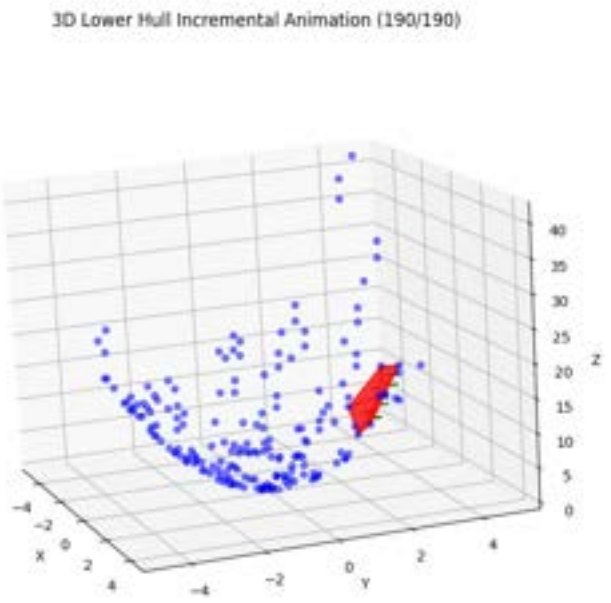


Figure 1: Pointset after barycenters are computed **before** lifting to R^3

- After realizing that the lifting wasn't working as expected, it was important to revisit [3] and reread the supporting section carefully ("1. For each k-tuple Q $\subseteq$ A, we take the barycenter of their **lift**"). The error in the previous attempt was that the barycenters should be calculated after the points are lifted, not before. So, for each 2-tuple in the $\binom{n}{2}$ points, project them to R^3 and then compute the barycenters between each pair (Figure 2).

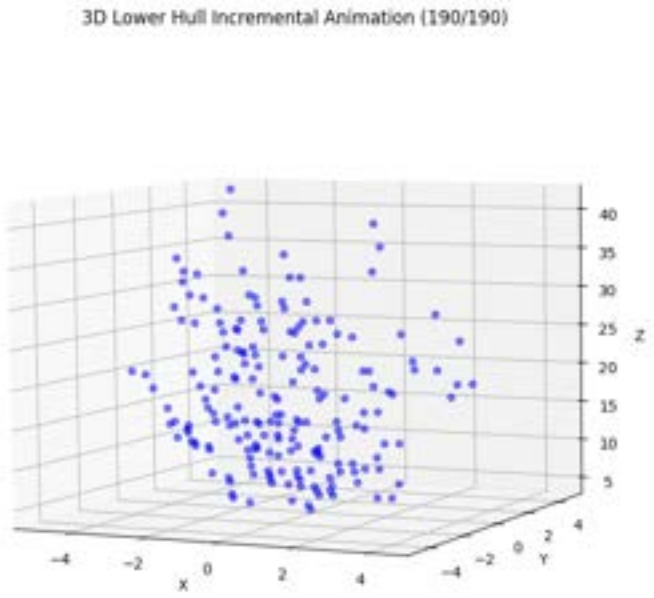


Figure 2: Pointset after barycenters are computed **after** lifting to R^3

- From Figure 2, it is apparent that all points no longer lie on the paraboloid. Therefore, many points can get eliminated as they lie inside the convex hull.
- After the pointset issue was rectified, the next issue was that there is no method in the qhull implementation to return the lower hull without iterating through the whole convex hull to identify the facets with downward facing normals. In order to rectify this, the next attempt was to create an implementation of the quickhull algorithm that keeps track of the lower hull after every insert, so that the lower hull could be returned in $O(1)$ amortized time. When testing with random points in R^3, this algorithm correctly returned the convex hull (tested against qhull), however, when testing with lifted points and barycenters (as described

before), the initial simplex was causing errors. It seems that with the barycenters, it is very often that the initial simplex has degeneracy with the first 4 points, specifically that they are coplanar. To remedy this issue, the degeneracy handling in qhull is the best solution. However, the degeneracy handling in qhull is quite sophisticated and proved to be too difficult to implement in a short timeframe. Refer to 6.2 & 5.3 for more information on runtime analysis.

---

5. Theoretical Results

- **5.1. Observations on Delaunay Refinement:** We conjecture based on observations that in the order-2 Delaunay triangulation, the blue triangles replicate the structure of the triangles of the first-order Delaunay triangulation, and the red triangles are responsible for the introduction of low-degree angles. Notice that the color of these triangles is related to the generation in the rhomboid tiling algorithm, so the methods there could be used to prove the aforementioned conjecture.
- **5.2 Incremental Algorithm Pseudocode:**
    - Input: A pointset in 2 dimensions
    - Output: The set of facets that define the convex hull in R^2
    - Steps:
        - Lift all points using $z = x^2 + y^2$
        - Take the barycenters of $\binom{n}{2}$ points after lift.
        - Create an initial simplex of d + 1 points
        - Initialize L, the set of all facets on the lower hull

- Iterate through each facet F in the initial simplex:
    - if p is above F, assign p to F's outside set, O.
- Iterate through each F where O is not empty:
    - Select the furthest point p (partitioning) from F's outside
    - Initialize V which is the set of visible facets from p
    - Iterate through all neighbors N of facets in V
        - If p is above N, add N to V
    - Iterate through each face, F'' in V
        - If F'' is in L, remove it.
    - For each ridge R in the boundary of V (which is the horizon ridges) H:
        - Create a new facet from R and p (essentially connecting the hole in the convex hull to the point)
        - If the normal vector of the facet is negative, add it to L
        - Update the neighbors of each facet.
    - For each new facet F':
        - For each unassigned point pt in an outside set of a facet in V:
            - If pt is above F', assign pt to F''s outside set.
    - Delete the facets in V
    - Compute the order 2 Delaunay Triangulation by projecting the lower hull down to R^2

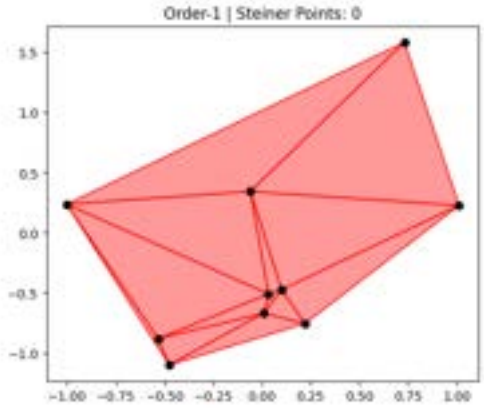
- **5.3 Incremental Algorithm Bounds for Order 2:**

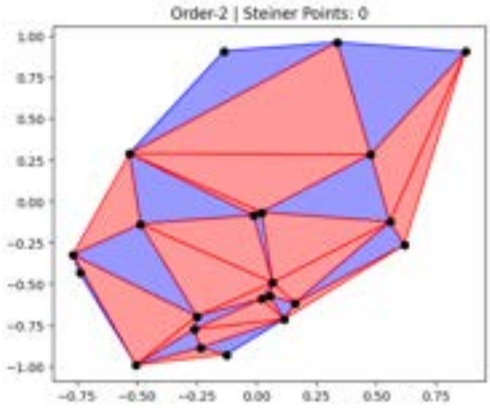
- From [18], observe that the tight upper bound is $O(n^2)$ in R^2 for the vanilla quickhull algorithm (if all points lie on the convex hull). In the algorithm, after V is calculated, we iterate through each face in V to check if they were in L, and if they are we remove them. This step costs $O(|V|)$ time, which in the worst case can equal to $O(h)$, where h is the number of facets on the convex hull. The next step is to check whether a facet constructed from the connection of R and p has a downward facing normal, and if it does add it to L. This step is done in $O(1)$ time, as the plane equation is calculated on creation of the facet object. Lastly, the last step added to the original quick hull algorithm is to project the lower hull down to R^2. This step costs $O(n)$ time, where n is the number of facets on the convex hull, which can approach $O(n)$ in the worst case. Therefore, the overall tight upper bound is $O(n^2)$. Note that the generation of the $\binom{n}{2}$ points, where n is the number of points in R^2 before lifting, is also $O(n^2)$. To optimize this algorithm, we should find a way to eliminate points inside the convex hull.

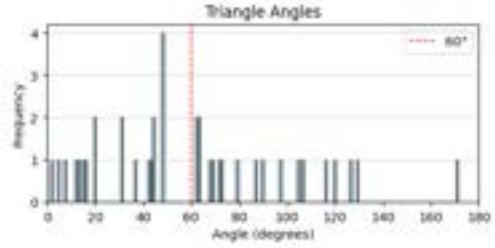
---

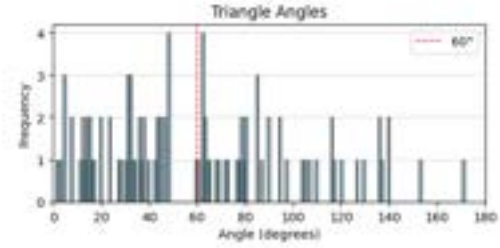
6. Experimental Results

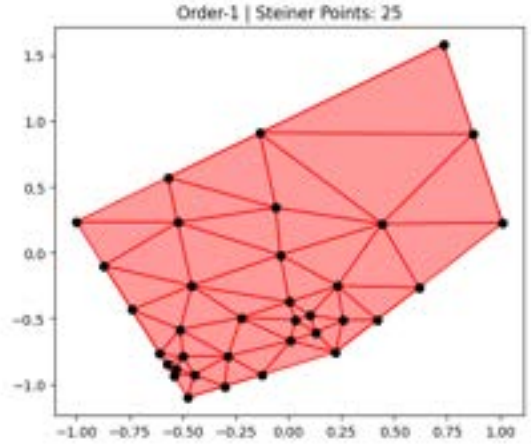
- **6.1 Delaunay Refinement:** min angle 30 degrees and maximum number of iterations 100.
    - **10 points sampled from Gaussian distribution**

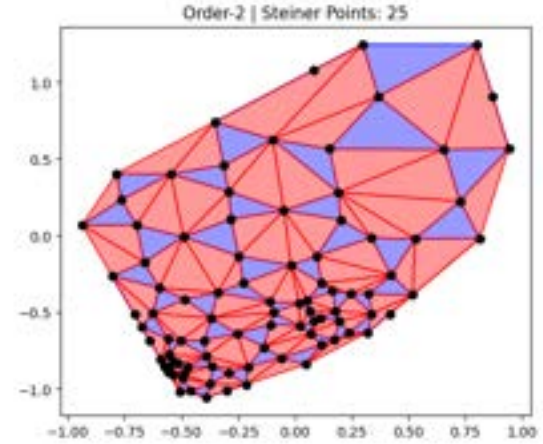

Order-1 | Steiner Points: 0

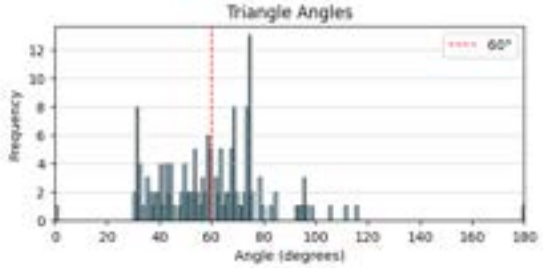

Order-2 | Steiner Points: 0

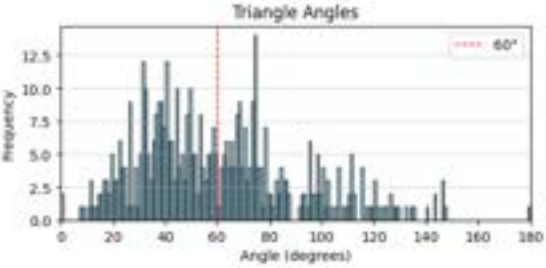

Triangle Angles
60°
Frequency
Angle (degrees)


Triangle Angles
60°
Frequency
Angle (degrees)

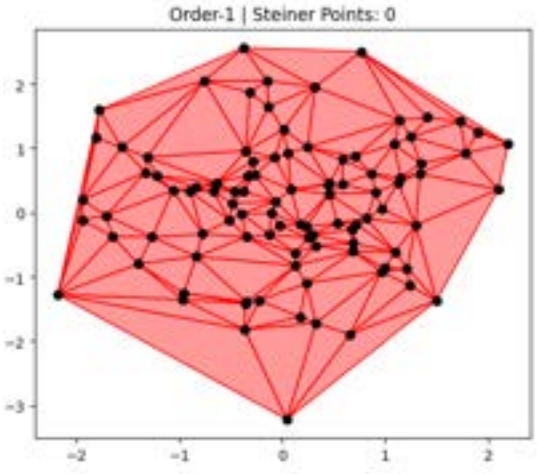

Order-1 | Steiner Points: 25

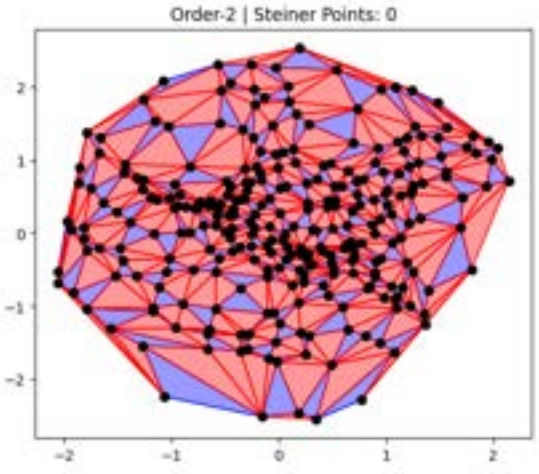

Order-2 | Steiner Points: 25

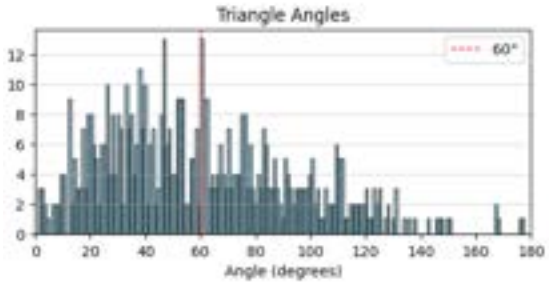

Triangle Angles
60°
Frequency
Angle (degrees)

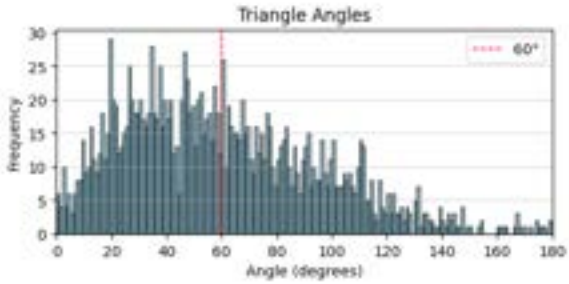

Triangle Angles
60°
Frequency
Angle (degrees)

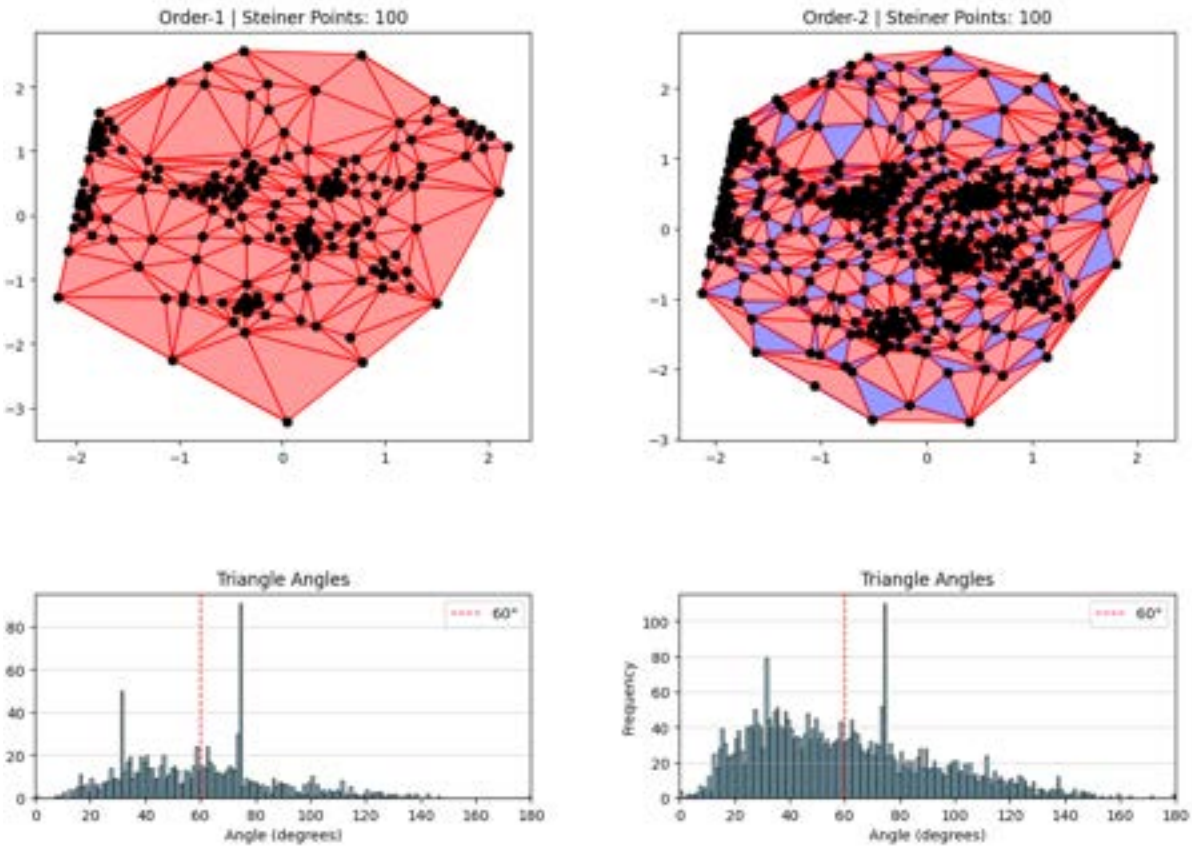
- **100 points sampled from Gaussian distribution**



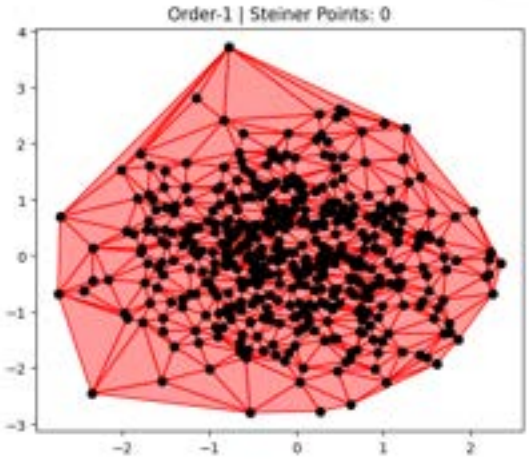


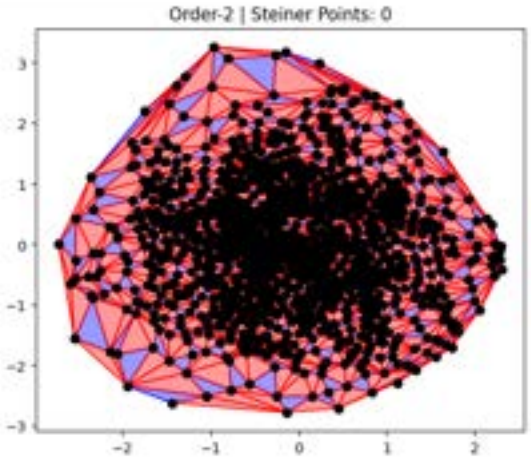


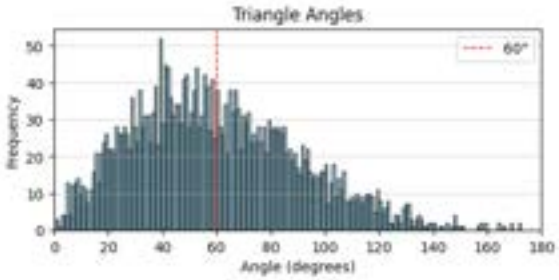


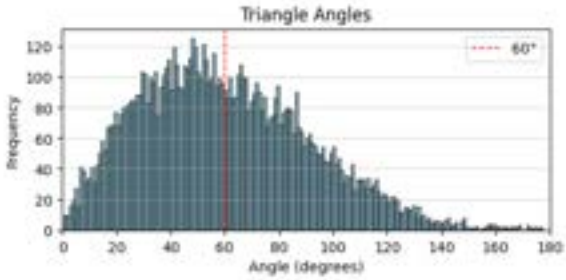


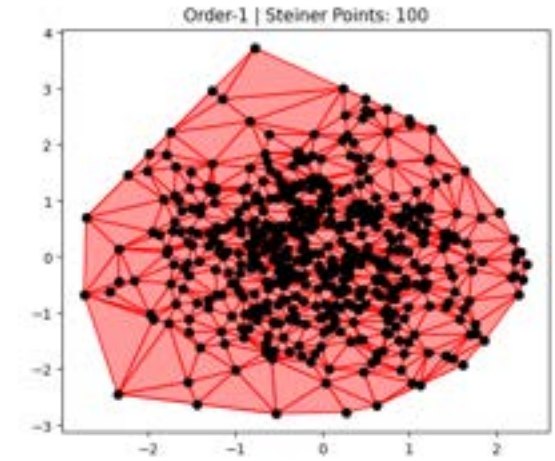
- **500 points sampled from Gaussian distribution**

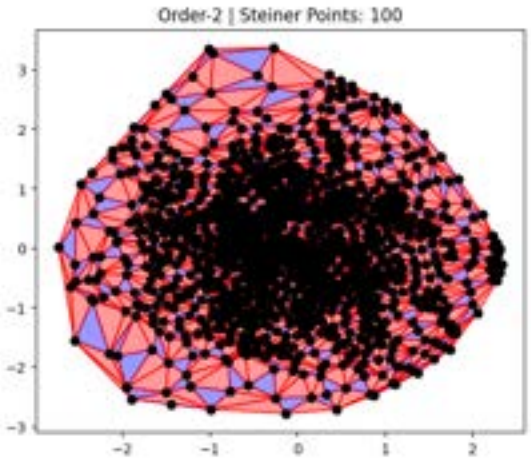

Order-1 | Steiner Points: 0

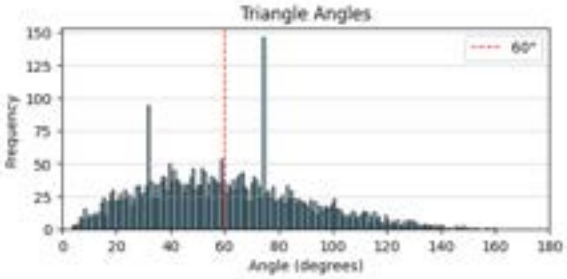

Order-2 | Steiner Points: 0

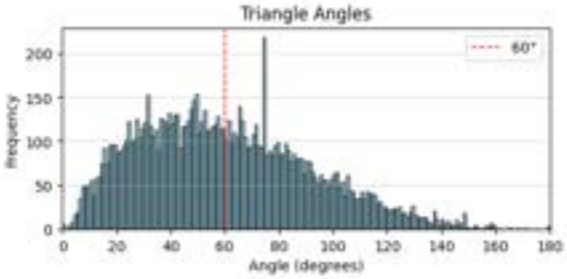

Triangle Angles
60°
Frequency
Angle (degrees)


Triangle Angles
60°
Frequency
Angle (degrees)

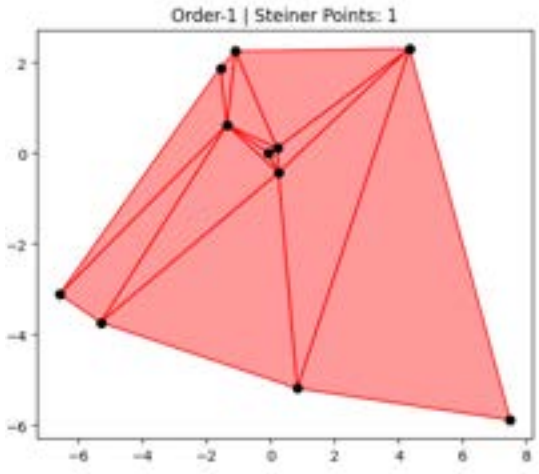


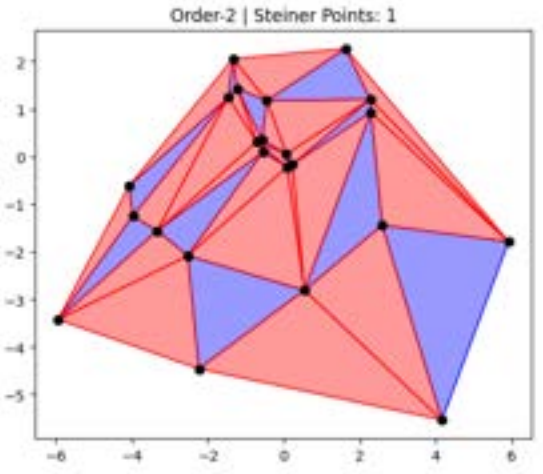


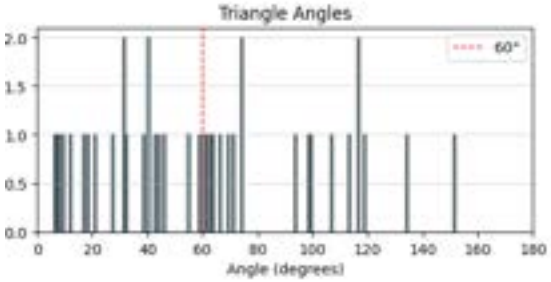


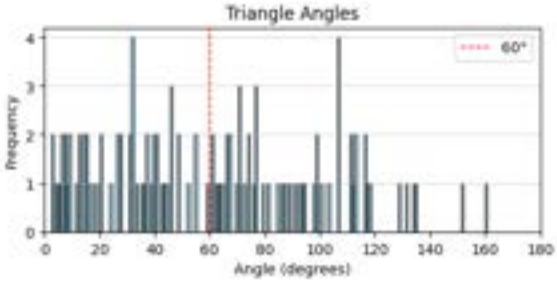


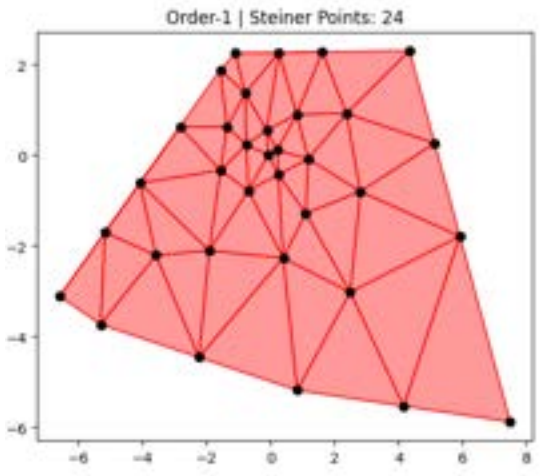
- **10 points sampled from Explosion distribution**

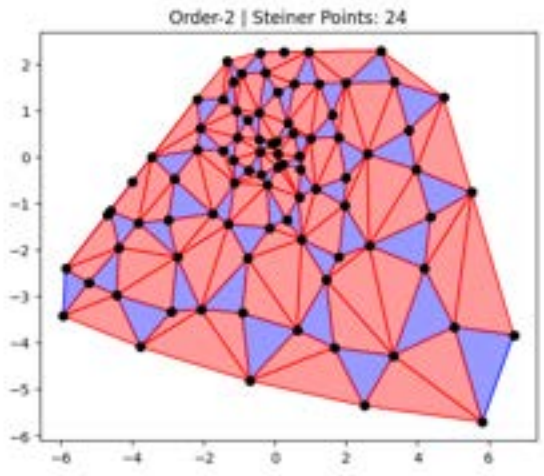

Order-1 | Steiner Points: 1

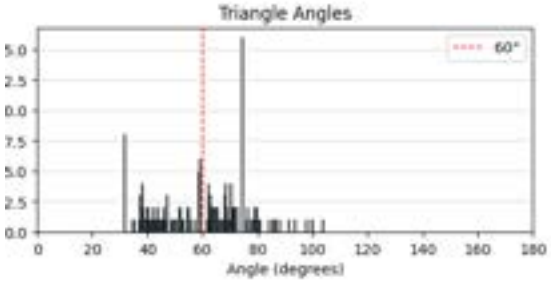

Order-2 | Steiner Points: 1

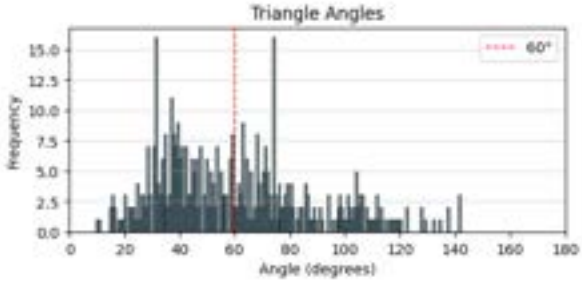

Triangle Angles
60°
Angle (degrees)


Triangle Angles
60°
Frequency
Angle (degrees)

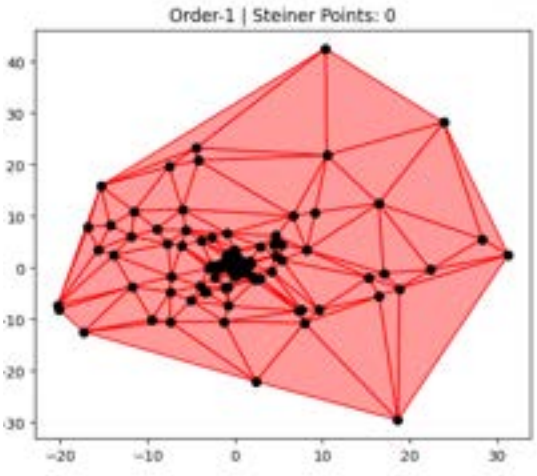


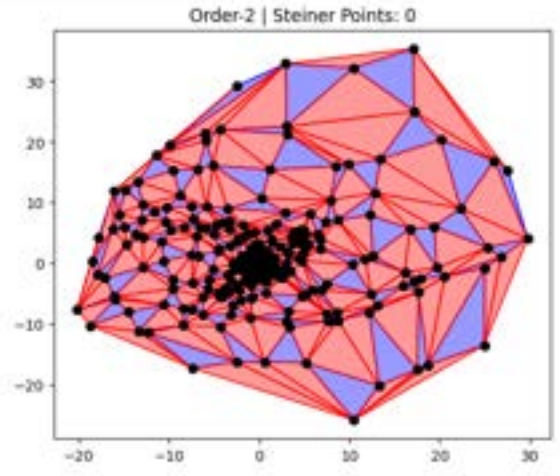


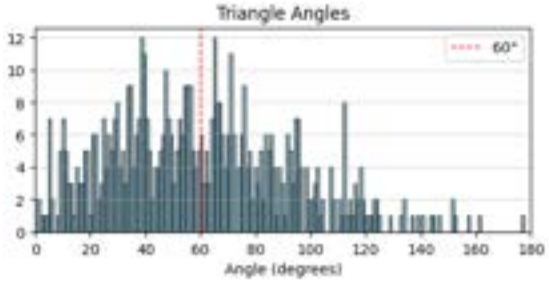


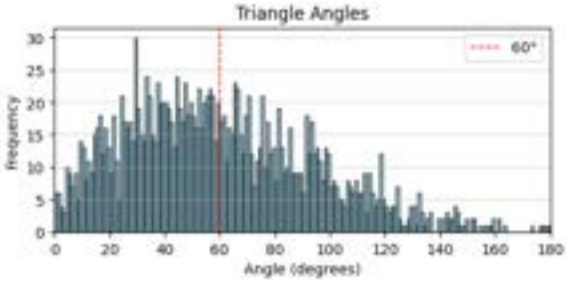


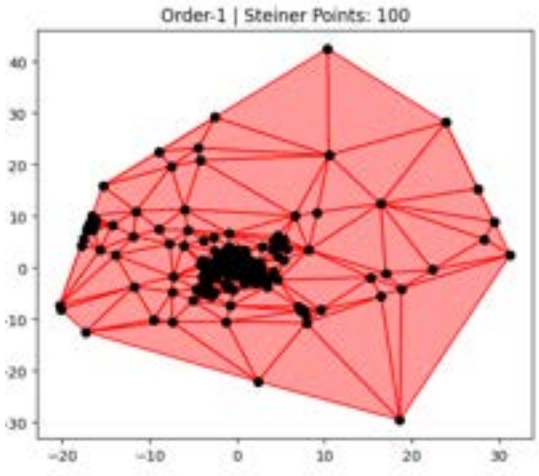
- **100 points sampled from Explosion distribution**

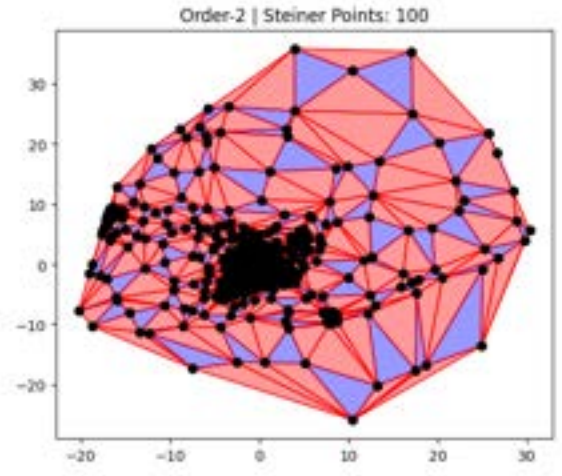

Order-1 | Steiner Points: 0

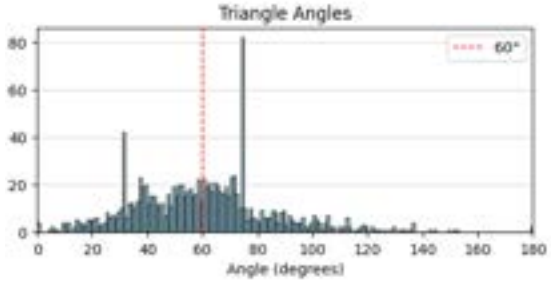

Order-2 | Steiner Points: 0

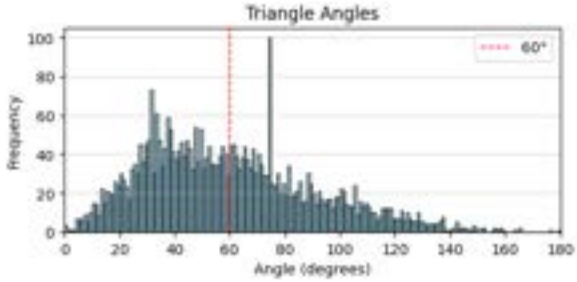

Triangle Angles
60°
Angle (degrees)


Triangle Angles
60°
Frequency
Angle (degrees)

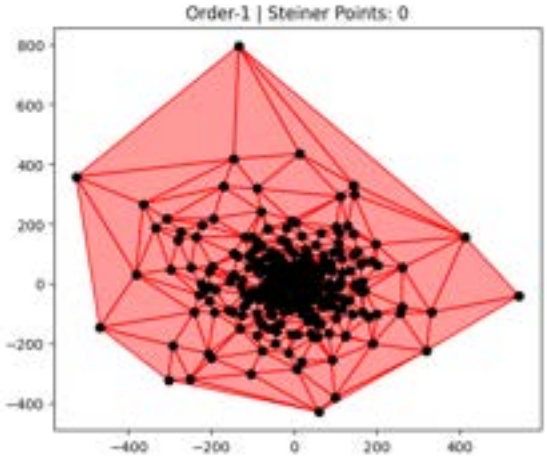


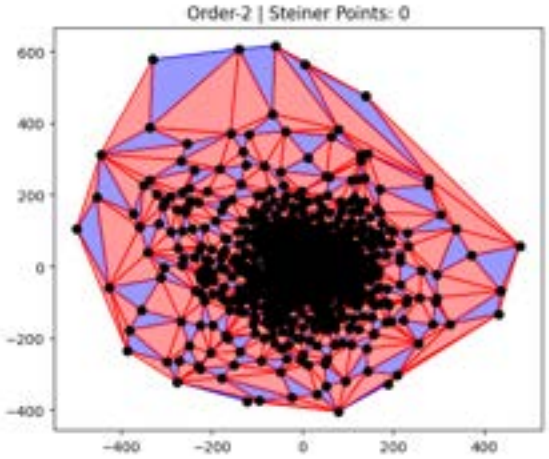


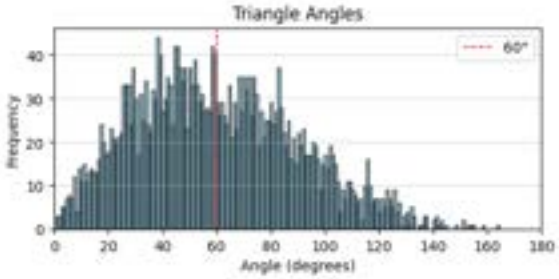


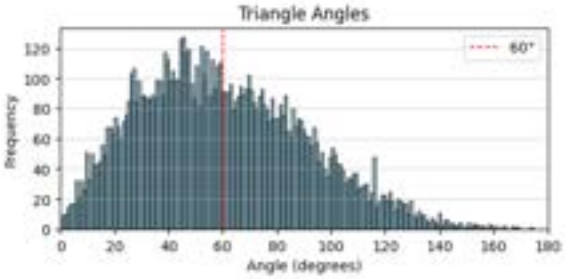


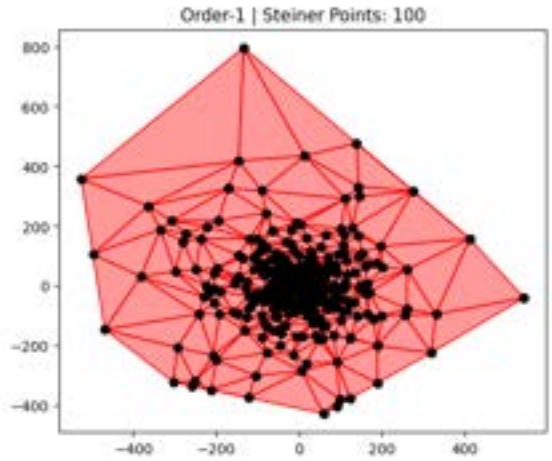
- **500 points sampled from Explosion distribution**

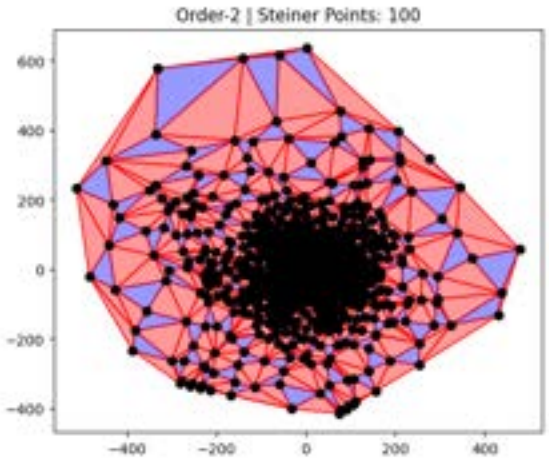

Order-1 | Steiner Points: 0
800
600
400
200
0
−200
−400
−400
−200
0
200
400

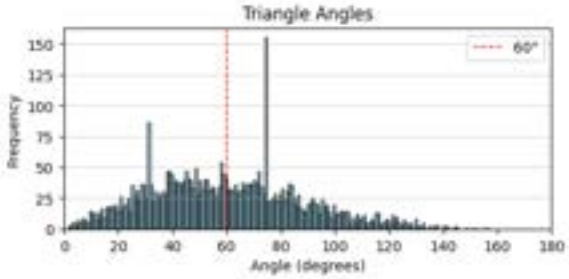

Order-2 | Steiner Points: 0
600
400
200
0
−200
−400
−400
−200
0
200
400

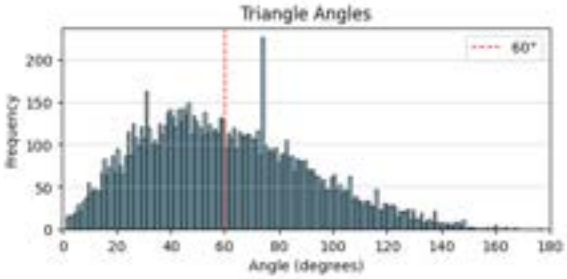

Triangle Angles
60°
Frequency
40
30
20
10
0
0
20
40
60
80
100
120
140
160
180
Angle (degrees)


Triangle Angles
60°
Frequency
120
100
80
60
40
20
0
0
20
40
60
80
100
120
140
160
180
Angle (degrees)

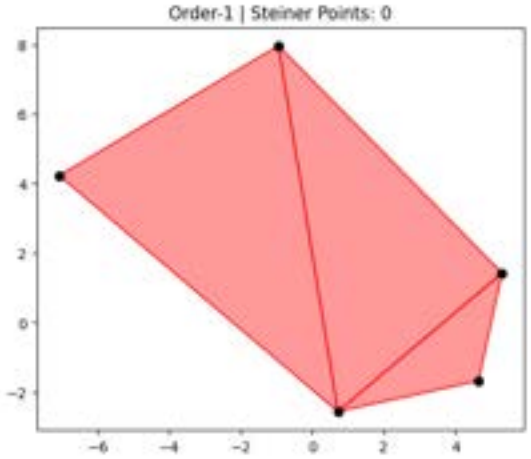


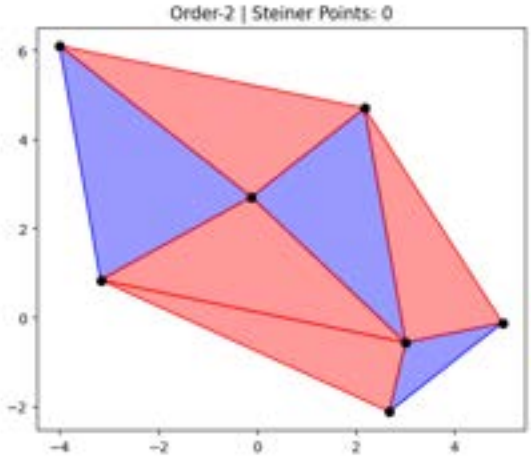


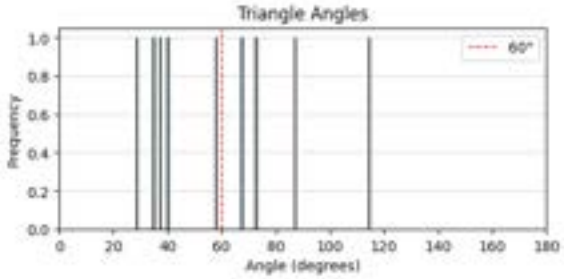


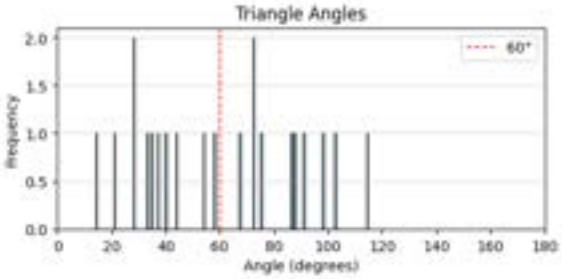


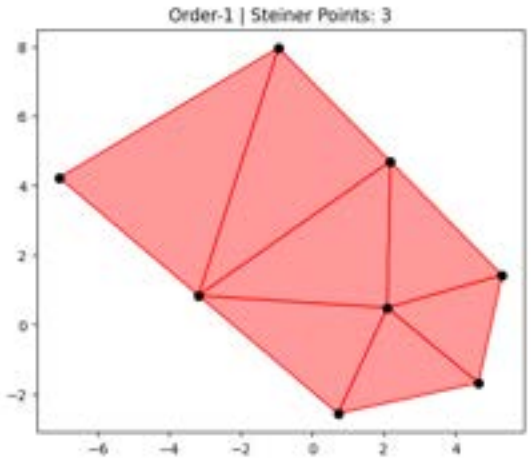
- **10 points sampled from Convex distribution with radius 10**

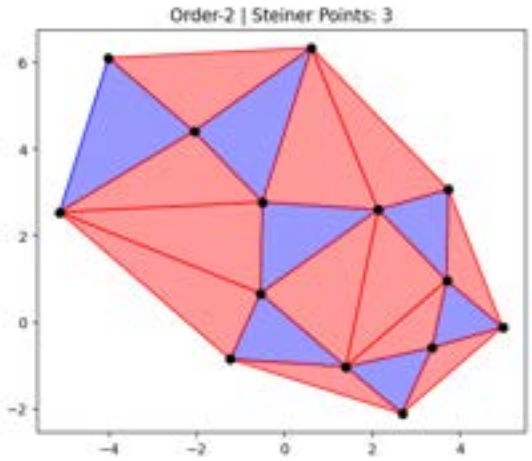

Order-1 | Steiner Points: 0

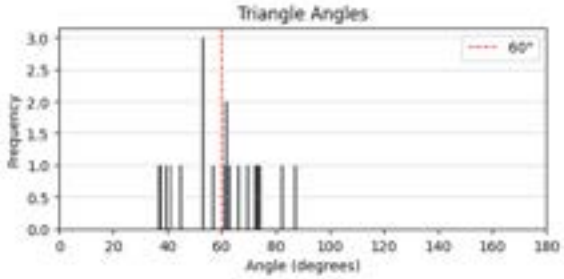

Order-2 | Steiner Points: 0

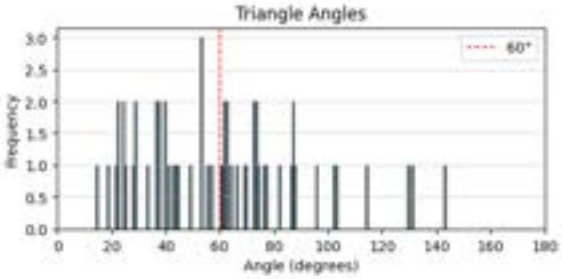

Triangle Angles
60°
Frequency
Angle (degrees)


Triangle Angles
60°
Frequency
Angle (degrees)

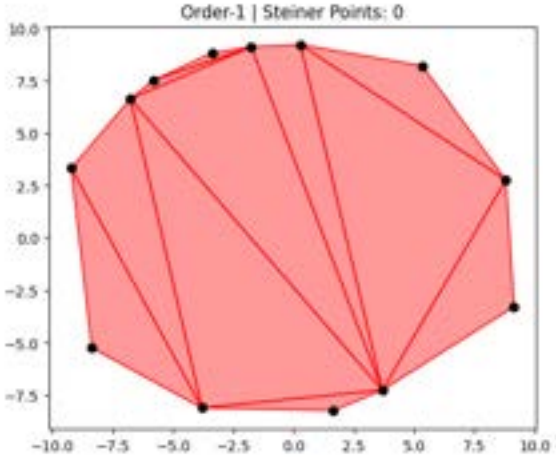


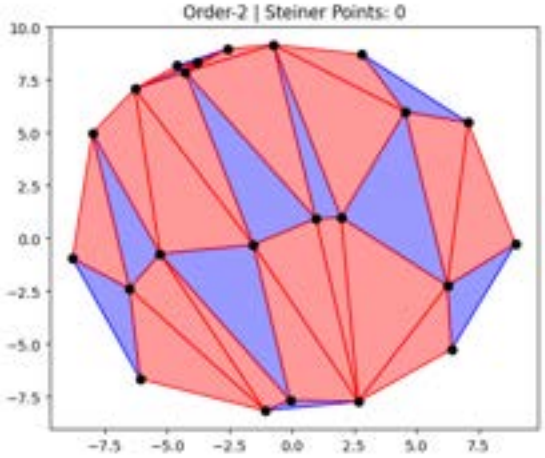


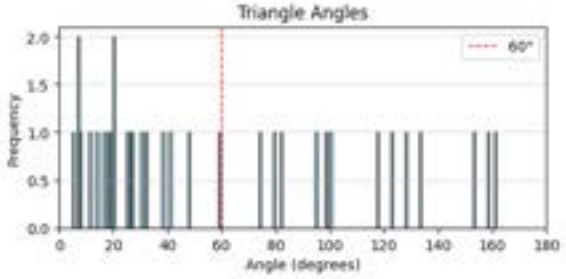


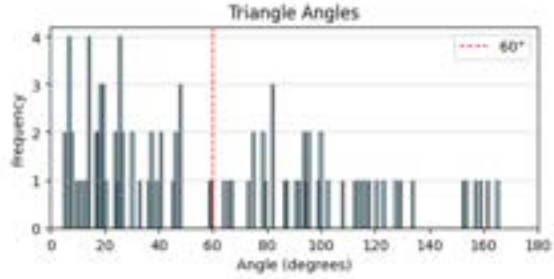


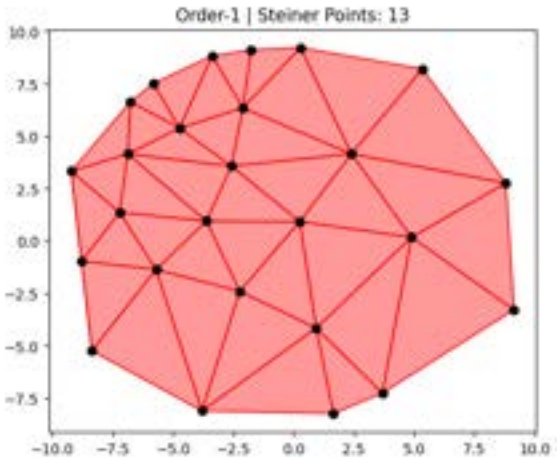
- **100 points sampled from Convex distribution with radius 10**

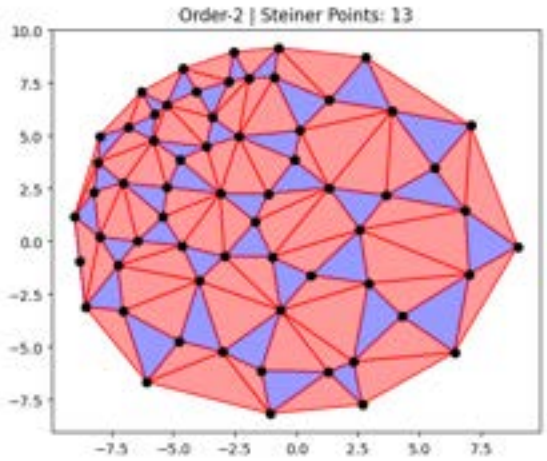

Order-1 | Steiner Points: 0

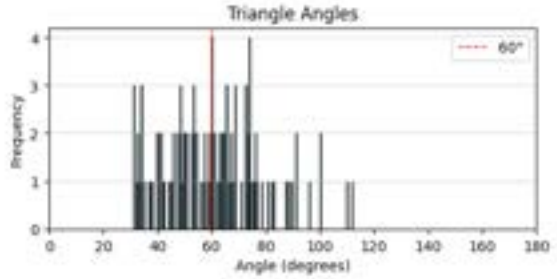

Order-2 | Steiner Points: 0

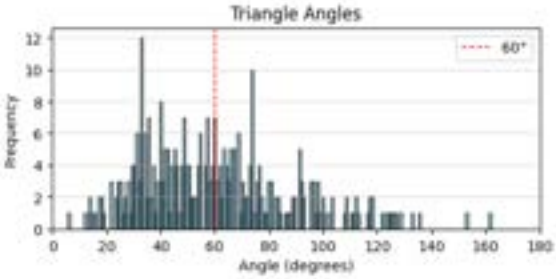

Triangle Angles
60°
Frequency
Angle (degrees)


Triangle Angles
60°
Frequency
Angle (degrees)

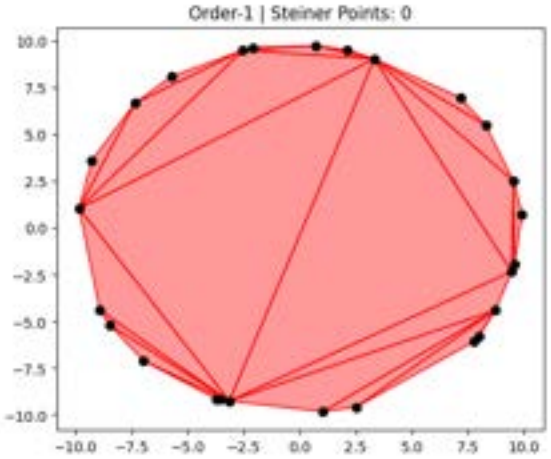


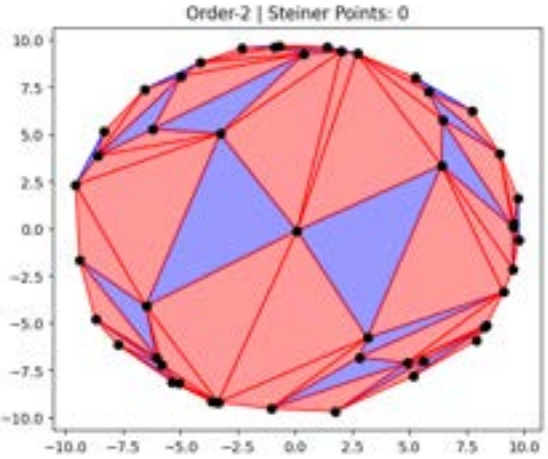


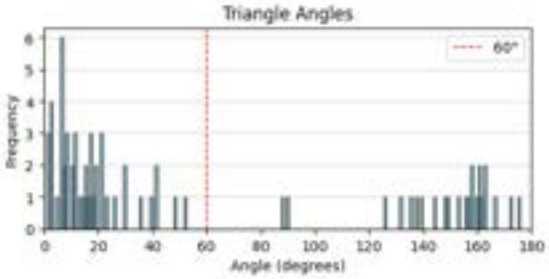


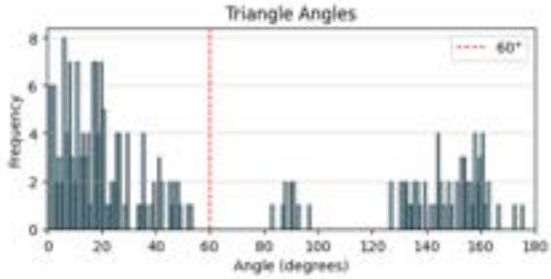


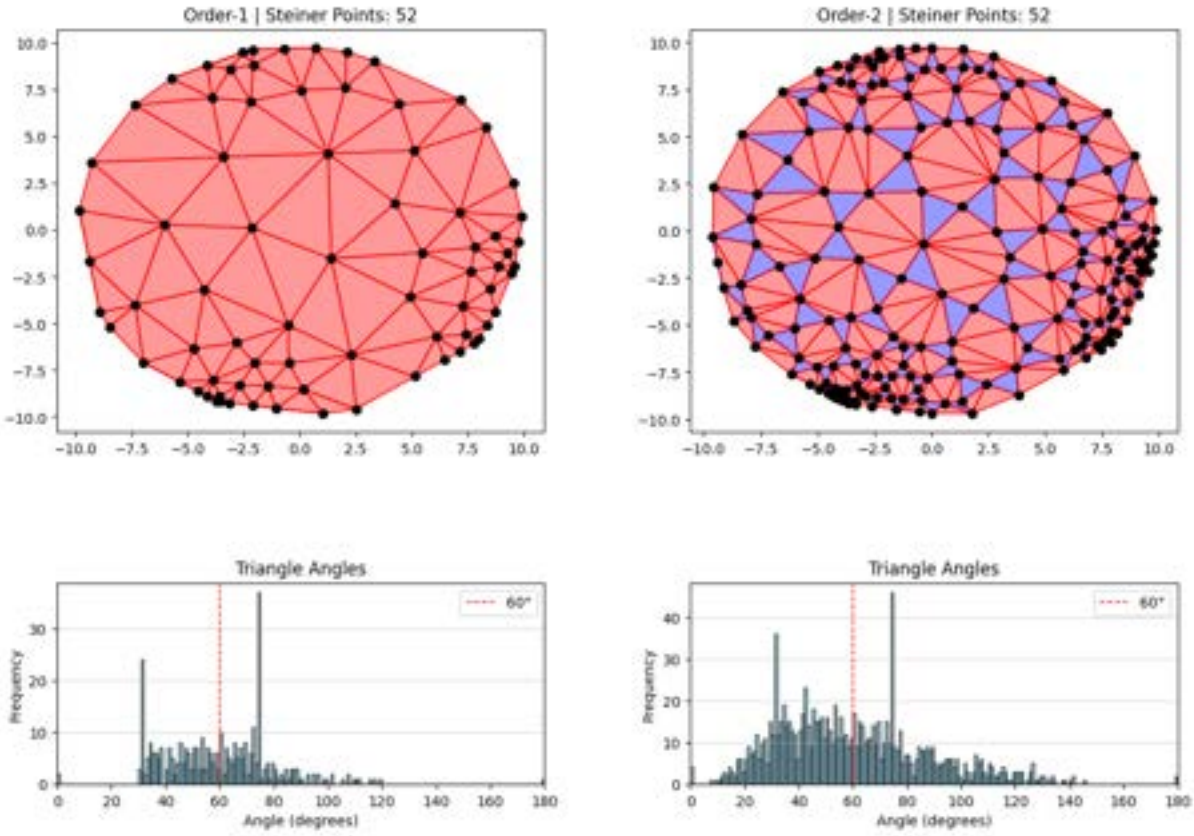
- **500 points sampled from Convex distribution with radius 10**


Order-1 | Steiner Points: 0
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0

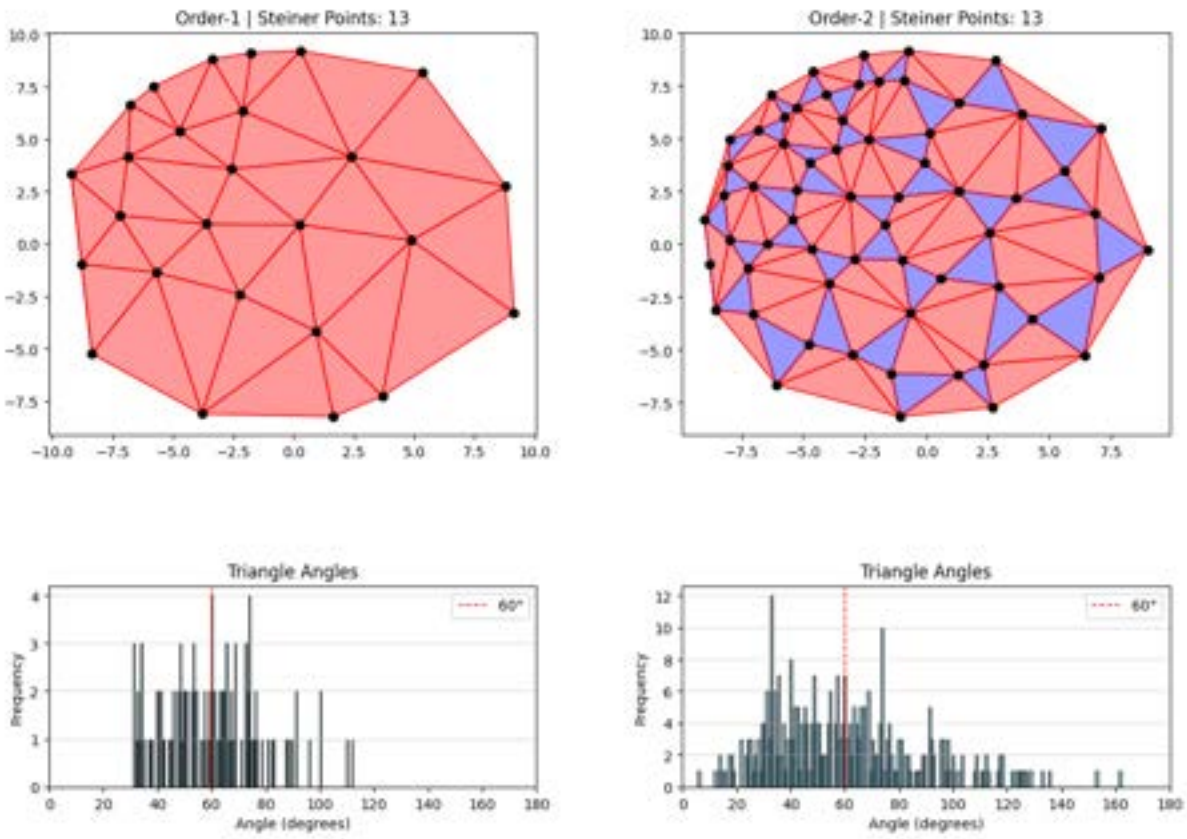

Order-2 | Steiner Points: 0
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0


Triangle Angles
60°
Frequency
Angle (degrees)
0 20 40 60 80 100 120 140 160 180


Triangle Angles
60°
Frequency
Angle (degrees)
0 20 40 60 80 100 120 140 160 180



From the previous figures there seems to be an improvement in the angle histogram of the order-2 Delaunay triangulations when refinement is performed. Notice, however, that the angle histogram does not converge at the same rate as the order-1 Delaunay triangulation angle histogram. This can be seen in the figure below for example:



where the angle histogram of the order-1 Delaunay triangulation does not contain any angles smaller than 30 degrees after 13 iterations of the refinement, whereas the angle histogram for the order-2 triangulation does contain several angles smaller than 30 degrees.

Another notable thing is the high number of angles of degrees about 33 and 74. We are not sure if this is a floating point problem or some characteristic of the algorithm Triangle is utilizing.

- **6.2 Incremental Algorithm:**

Input size (number of points)	**Rhomboid Tiling (s)**	**Paraboloid Projection Algorithm (s)**
10	**0.002**	**0.0010**

100	**0.004**	**0.0080**
1000	**0.0380**	**0.6348**
10000	**0.4533**	**99.9342**
100000	**6.8876**	**>60000**
1000000	**118.9461**	**>60000**

Table 1: Runtime Analysis on a Gaussian Point Distribution

-

Input size (number of points)	**Rhomboid Tiling (s)**	**Paraboloid Projection Algorithm (s)**
10	**0.002**	**0.00**
100	**0.0050**	**0.0080**
1000	**0.0478**	**0.6070**
10000	**0.4474**	**63.4843**
100000	**5.6137**	**>6000**
1000000	**61.9944**	**>6000**

Table 2: Runtime Analysis on a Convex Point Distribution

-

Input size (number of points)	**Rhomboid Tiling (s)**	**Paraboloid Projection Algorithm (s)**
10	**0.002**	**0.0**

100	**0.005**	**0.0075**
1000	**0.0391**	**0.6688**
10000	**0.4510**	**68.5789**
100000	**5.6866**	**>6000**
1000000	**73.6819**	**>6000**

Table 3: Runtime Analysis on an Explosion Distribution

As is evident in Tables 1-3, the incremental algorithm is running poorly in comparison to the Rhomboid Tiling algorithm. The reason for this is dominated by the convex hull computation and the $\binom{n}{2}$ barycenter calculation. This suggests that the algorithm can still be improved by not considering points that aren't on the convex hull.

---

7. Future Study Plan

- **7.1 Approaches:**
- **Incremental Algorithm:** Recall that for the incremental algorithm we lift each point of our point set Q, we find their barycenters, and then obtain the order-2 Delaunay triangulation as the vertical projection of the lower facets of the convex hull of these barycenters. Notice, however, that this approach is not very efficient for pointsets Q with large cardinality, since there are $\binom{|Q|}{2}$ such barycenters. Most barycenters, however, are irrelevant as they do not contribute to the lower faces of the convex hull. If we could, somehow, identify the relevant barycenters without wasting time on the irrelevant ones, this procedure would efficiently construct the cells of the order-k Delaunay mosaic. Notice that section 3.2 of [3]

provides a way to identify the relevant vertices of the order-k Delaunay mosaic without explicitly computing all barycenters. The key idea is that if we already know the order j Delaunay mosaics for all $j < k$, then the vertices of the order-k Delaunay mosaics can be assembled by taking certain slices from first generation cells at lower orders. The authors avoid computing irrelevant barycenters by using the combinatorial structure of the rhomboid tiling and hyperplane arrangement. To be precise, lemma 4 and theorem 5 in section 3.2 show that each needed barycenter corresponds to a bowl in the hyperplane arrangement, and their algorithm can find these systematically without checking all subsets. For our case the only "lower" triangulation that we have is the regular Delaunay triangulation. According to Lemma 3, we then have that a triangle is first-generation if the vertices {a,b,c} share exactly k-1 =0 common points. Notice this is true for every Delaunay triangle. Therefore all ordinary triangles in the ordinary triangulation are first-generation. Therefore we are just required to find the generation-2 cells of the rhomboid of the corresponding triangle. This g = 2 cell will consist of the triangle of midpoints of the vertices of our original triangle. Below is an algorithm that performs the previous task:

- Compute the ordinary Delaunay triangulation
- For every Delaunay edge compute its midpoint m
- Lift each of those midpoints
- Compute the convex hull of those lifted points
- Project each triangular face of the lower hull back to the plane
- The set of projected triangles is the order-2 Delaunay triangulation

We are still required to make this incremental. But since there are already incremental algorithms for order-1 Delaunay triangulations, it does not seem too difficult to Design an algorithm that will check for triangles that are removed or added after inserting a Steiner point.

- **Refinement:** We explored two main methods of refinement in our project. While both of them improve the angle histogram, and decrease the number of bad angles, neither of them seems to be able to fully remove all bad angles that are below/above a given threshold in the order-2 DT. Future steps to better solve this could involve modifying our current refinement strategies, or trying different ones.
- **Other problems:** Another problem that we considered studying, but didn't get a chance to address is the size of order-k Delaunay triangulations.

---

8. Learning Experience

- **8.1 Adnan Farid**: This was my first time participating in a research project. It was a very good experience to learn how to read research papers that aren't easy to consume in one sitting. This project, in addition to the class, although challenging, made me think about problems from different views to find unique solutions. This is definitely going to come in handy for problems where there is no clear solution. In addition, the experimentation component was valuable, as it taught me how results in computer science are supported.
- **8.2 Armando Albornoz**: I have learned a lot about computational geometry and discrete geometry while doing this. I particularly enjoyed reading [3] since it introduced the idea of a tiling, which is something that I have never seen in other math subjects. I also enjoyed performing the experimental study since it provided me with an experience that I have never had–all the other research projects I have worked on have been purely theoretical.
- **8.3 Edward Guthrie**: I really enjoyed this course and learning about many different algorithms in computational geometry. The project was quite interesting, as the subject material and problem is more theoretical than previous research that I've done, although most of my work on this problem was experimental.

---

9. References

- [1] R. Sibson. Locally equiangular triangulations. Comput. J. 21 (1978), 243–245.
- [2] F. Aurenhammer and O. Schwarzkopf. A simple on-line randomized incremental algorithm for computing higher order Voronoi diagrams. *Proc. 7th Annu. Sympos. Comput. Geom.* (1991), 142–151.
- [3] Edelsbrunner, H., Osang, G. A Simple Algorithm for Higher-Order Delaunay Mosaics and Alpha Shapes. *Algorithmica* **85**, 277–295 (2023). https://doi.org/10.1007/s00453-022-01027-6
- [4] Osang, G.: Higher order Delaunay mosaics in C++ and python. https://github.com/geoo89/orderkdelaunay (2019)
- [5] Lee, D.-T.: On k-nearest neighbor Voronoi diagrams in the plane. IEEE Trans. Comput. 31(6), 478–487 (1982)
- [6] Shamos, M.I., Hoey, D.: Closest-point problems. In: Proceedings of the 16th Annual IEEE Symposium on Foundations of Computer Science, pp. 151–162 (1975)
- [7] Clarkson, K.L., Shor, P.W.: Applications of random sampling in computational geometry. II. Discrete Comput. Geom. 4(1), 387–421 (1989)
- [8] Mulmuley, K.: Output sensitive construction of levels and Voronoi diagrams in Rd of order 1 to k. In: Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, pp. 322–330 (1990)
- [9] Edelsbrunner, H., Osang, G.: The multi-cover persistence of Euclidean balls. In: Proceedings of the 34th Annual Symposium on Computational Geometry, pp. 34:1–34:14 (2018)
- [10] Aurenhammer, F.: A new duality result concerning Voronoi diagrams. Discrete Comput. Geom. 5, 243–254 (1990)

- [11] H. Edelsbrunner, A. Garber, and M. Saghafian, "Order-2 Delaunay triangulations optimize angles," *Advances in Mathematics*, vol. 461, p. 110055, Feb. 2025. [Online]. Available: https://doi.org/10.1016/j.aim.2024.110055
- [12] C.L. Lawson. Software for C1 surface interpolation. In Mathematical Software, “Proc. Sympos. at the Mathematics Research Center, Univ. Wisconsin–Madison, 1977”, 161–194
- [13] R. Sibson. Locally equiangular triangulations. Comput. J. 21 (1978), 243–245.
- [14] H. Edelsbrunner, A. Garber, M. Ghafari, T. Heiss and M. Saghafian. Flips in two-dimensional hypertriangulations. Inst. Sci. Techn. Austria, Klosterneuburg, Austria, 2023, https://arxiv.org/abs/2212.11380
- [15]Aurenhammer, F. A criterion for the affine equivalence of cell complexes in R^d and convex polyhedra in R ^{d+1}. Discrete Comput. Geom. 2 (1987), 49-64
- [16] S. L. Gonzaga de Oliveira, "A Review on Delaunay Refinement Techniques," in *Computational Science and Its Applications -- ICCSA 2012*, B. Murgante, O. Gervasi, S. Misra, N. Nedjah, A. M. A. C. Rocha, D. Taniar, and B. O. Apduhan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 172–187. https://link.springer.com/chapter/10.1007/978-3-642-31125-3_14#citeas
- [17] Edelsbrunner, H., Garber, A., Ghafari, M. *et al.* On Angles in Higher Order Brillouin Tessellations and Related Tilings in the Plane. *Discrete Comput Geom* **72**, 29–48 (2024). https://doi.org/10.1007/s00454-023-00566-1
- [18] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, “The quickhull algorithm for convex hulls,” *ACM Transactions on Mathematical Software*, vol. 22, no. 4, pp. 469–483, Dec. 1996, doi: https://doi.org/10.1145/235815.235821