Implementing Three Voronoi Diagram Computation Algorithms and Comparing Their Performance

We will implement a program for calculating and visualizing the Voronoi Diagram of a set of points in 2D with three different algorithms. We will use a Randomized Incremental Algorithm, then a Fortune's Algorithm, and, finally, the Flipping Algorithm. Using Randomized Incremental Algorithm, the Voronoi Diagram is built incrementally, one point at a time. The algorithm locates the impacted Voronoi areas and updates them for each point. Fortune's Algorithm uses a sweepline across a plane while the Flipping Algorithm utilizes an arbitrary triangulation and then transforms said triangulation by flipping its diagonals. This random triangulation of the Flipping Algorithm is done by using Lexicographic Triangulation.

These algorithms are expected to produce a Voronoi diagram with the same topology and geometry when used but with varying runtimes and memory needs. When dealing with particular inputs, some algorithms may be more effective than others, and we will measure their effectiveness accordingly. Applications requiring the use of the Voronoi Diagram, such as computational geometry, image processing, and machine learning, are among those for which this information is crucial because the efficiency of the algorithm can significantly impact the overall performance of the system.

Additionally, the Delaunay triangulation is closely related to the Voronoi Diagram, and it is frequently computed as an intermediary stage in the Voronoi Diagram's creation. A simple formula can be used to convert between the two structures in order to compute the Voronoi Diagram using the Delaunay triangulation. The Voronoi Diagram can be produced specifically by calculating the circumcenter and radius of the circumcircle for each triangle in the Delaunay triangulation of a collection of points. The set of points that are evenly spaced from the circumcenters of nearby triangles is then known as the Voronoi Diagram. Therefore, for the algorithms we intend to implement in our project, we can use the Delaunay triangulation.

We will generate a set of random points in two dimensions via different distribution inputs, then calculate and visualize the 2D Voronoi Diagrams as graphical outputs. A clear user interface that specifies parameters such as the number of points, has zoom-in/out functions, and can translate while displaying/visualizing the 2D Voronoi Diagram will also be a part of our program. We will also make sure that each Voronoi Cell is visualized clearly and show the steps of Fortune's Algorithm. Finally, we will test our program using reasonable point sets in 2D, ranging from 100 to 1000000, and report our implementations’ performance by comparing the aforementioned approaches.