CSE306 - Assigment II

Deivis Banys

1 Architecture

My whole project is distributed into the following parts:

- `Functions.cpp` - contains all my implemented functions.
- `main.cpp` - the main function used to test all those functions.
- `SVG_files` - a folder used to gather SVG files of the diagrams
- `Images` - a folder used to gather images created from those SVG files.
- `Vector.h` - file implemented in the previous assignment and slightly modified so that it can be used for 2D vectors.
- `svg_writer_for_polygons.cpp` - file with functions used to get the visual representation of the polygons.

2 Sutherland-Hodgman polygon clipping algorithm

Sutherland-Hodgman algorithm is an algorithm used to the clip one polygon (`subjectPolygon`) in a way that it can fit into the other Polygon(`clipPolygon`). To clip a `subjectPolygon` by a line, the algorithm iteratively goes through all the edges of it and changes them in a way that it fits in `clipPolygon` and thus creates a new polygon each time. After going through all the edges, the resulting `subjectPolygon` fits in `clipPolygon` (Figure 1.).

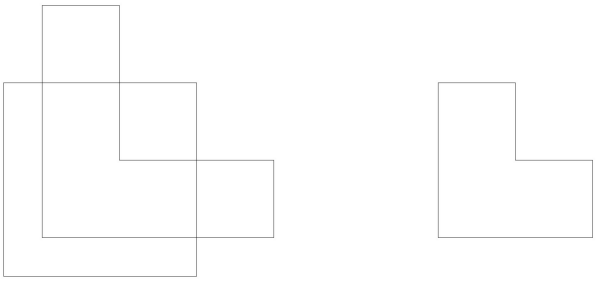


Figure 1: Sutherland-Hodgman polygon clipping algorithm (before and after)

To do so, I defined function `clip_Polygon(Polygon subjectPolygon, Polygon clipPolygon)`, which applies this procedure. Also, the function relies on two auxiliary functions:

- `is_inside(Vector P, vector<Vector> clipEdge)` - returns a boolean value indicating whether a point `P` is inside `clipEdge`.
- `intersect(Vector A, Vector B, vector<Vector> clipEdge)` - returns a point of intersection (if it exists) between the edge [A, B] and the infinite line, for which two points u and v are known (here, `clipEdge = {u,v}`)

3 Voronoï Parallel Linear Enumeration

To generate a Voronoï diagram, firstly I define a function `generate_points(int n)`, which generates n 2D vectors, such that each vector (x, y) is with the coordinates $x, y \sim \textbf{Uniform}(0, 1)$. Then, after generating a set of points P, I implement the algorithm Voronoï Parallel Linear Enumeration, which simply goes over all pairs of points $P_i, P_j \in P$ s.t. $i \neq j$ and applies previously discussed - Sutherland-Hodgman polygon clipping algorithm. The resulting function is called `generate_voronoi_diagram()` in my code. However, the functions - `is_inside()` and `intersect()` (that are also used by `generate_voronoi_diagram()`) had to be slightly changed, so I added boolean values `voronoi` to the input those function receive, so that they can be used in either way and changed the functions accordingly. A few examples of the results:

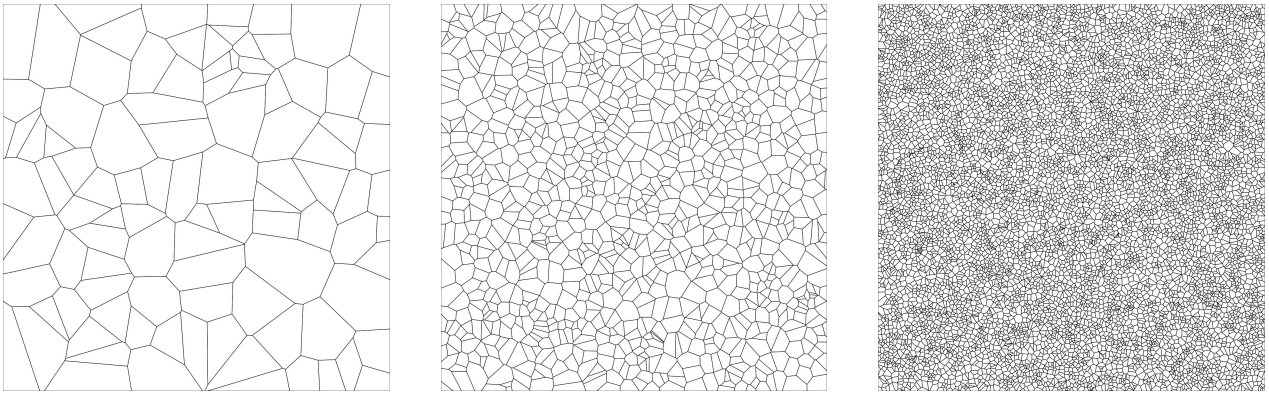


Figure 2: Voronoï diagrams from 100, 1,000 and 10,000 points respectively

4 Extending Voronoï diagram to a Power diagram

Now we are interested in the extension of Voronoï diagrams to so called Power diagrams. These diagrams allow us to control the size of cells in the diagram by choosing the weight for each of point . As it is just an extension of Voronoï diagrams, a small change of code was needed. I, once again, changed the functions `is_inside()` and `intersect()` by adding `vector<float> weights` to their inputs and changed them accordingly. Also, now I changed the function generating the Voronoï diagram, so that now inputting `vector<float> weights` to it, yields a power diagram. To generate weights I used several approaches (all the points are still chosen uniformly):

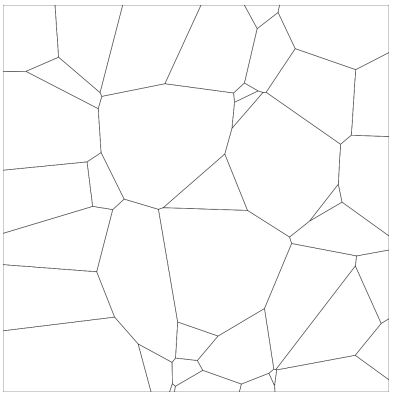
(a) I generated weights uniformly (weight $\sim$ **Uniform**$(0, 1)$ for all points)

(b) I used weight$(x, y) := cos(x + y)$

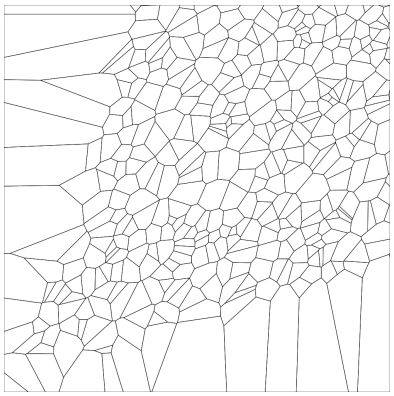
(c) I used weight$(x, y) := xy$

(d) I used weight$(x, y) := xy^2$

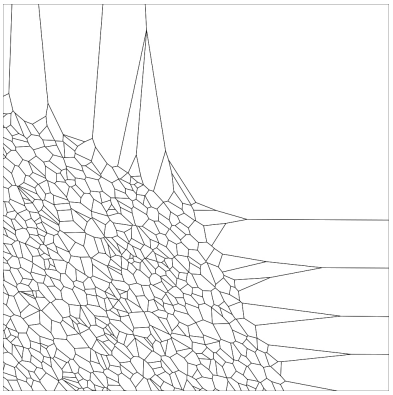
The corresponding graphs:



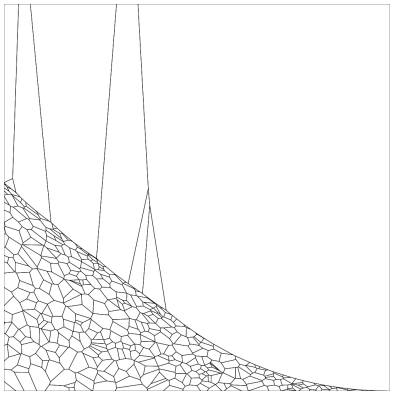
(a)



(b) caption



(c) caption



(d) caption