CSE306 Assignment 2 - Geometry Processing

Maria Barbulescu

June 2020

1 Introduction

The purpose of this project is to introduce geometry processing. This report presents the different steps and their outputs as well as explains the code behind the results.

The C++ code is organized in two different files: `svg_polygon.cpp`, `voronoi.cpp`. The first file contains the `Polygon` class (a `Polygon` object is a vector of `Vectors` which represent the vertices - there is an edge between each consecutive vertex). It also contains two premade functions (https://pastebin.com/bEYVtqYy) `save_svg` and `save_svg_animated` that are used in creating `.svg` files. The `voronoi.cpp` file contains all of the auxiliary functions and the `main()`. It is worth noting that the `Vector` class is the one used in the Raytracer assignment.

2 Voronoi Parallel Linear Enumeration

The aim of this section is to compute the Voronoi diagram of a random point cloud. Initially, we implement the generalized Sutherland-Hodgman algorithm for clipping polygons. This is the `sutherland_hodgman(Polygon subject, Polygon clip)` function, which is iteratively clipping a polygon, and uses auxiliary functions `intersect(Vector A, Vector B, Vector u, Vector v)`, which returns the intersection point of [AB] and the bisector of the edge determined by u and v, and `inside(Vector P, Vector u, Vector v)` which returns `true` if P is inside the edge determined by u and v `false` otherwise.

To achieve the Voronoi diagram, we apply the Sutherland-Hodgman algorithm between each pair of points $P_i, P_j, i \neq j$, in `voronoi(Polygon subject, Polygon clip)`. This gives an algorithm that runs in $\mathcal{O}(n^2)$. To obtain uniformly random point clouds of size n we use the function `random_pointcloud(int n)`. For 100 points, the code runs in 14 milliseconds, and for 10000 points in 13 seconds (with parallelization, Figure 1).

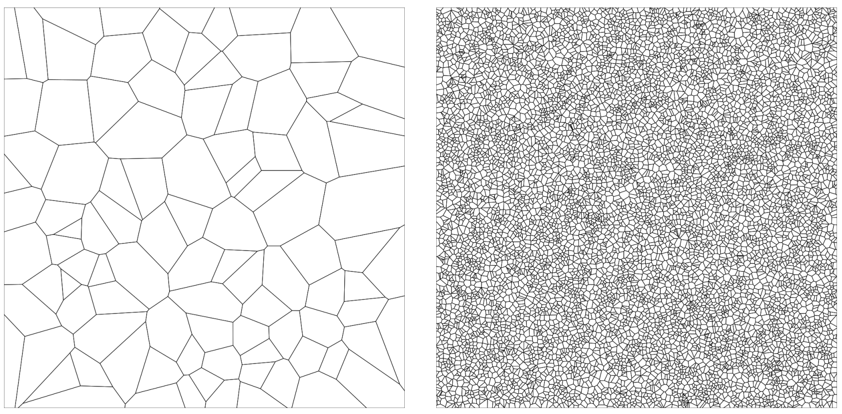

Figure 1: Voronoi diagram on a point cloud of size 100 and 10000 respectively

3 Power Diagram

We would now like to add weights to the initial point cloud that we compute the Voronoi diagram on. We do that by adding auxiliary functions which take into account the weight w of a point (thus, we slightly modify `intersect`, `inside`, and `voronoi`, and create new functions so that we can still compute everything without weights if it is the case). We take uniform random weights with values between 0 and $\frac{log(n)}{n}$. The result is displayed in Figure 2 for a point cloud of size 100.

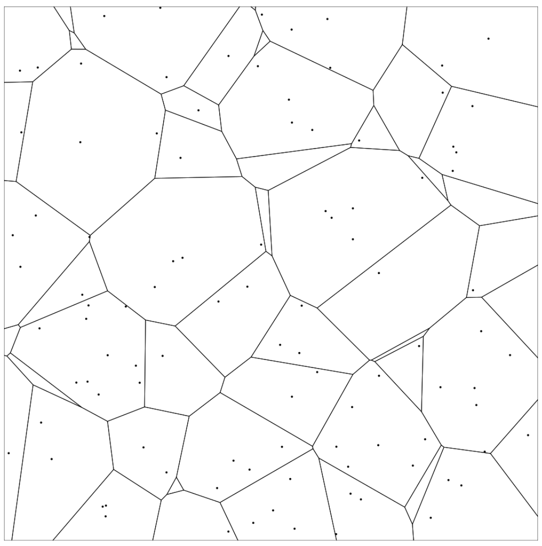

Figure 2: Power diagram for 100 points (with points included for reference)