CSE306 Project 2 Report - Geometry Processing

Arjun Bommadevara

June 2024

1 Introduction

This Geometry Processing project was built in C++, and implements the following: Suntherland-Hodgman Polygon clipping, Voronoi Parallel Line Enumeration, Power Diagram using L-BFGS, Fluid Dynamics simulator, and Tutte Mapping. I have divided the different sections into different files.

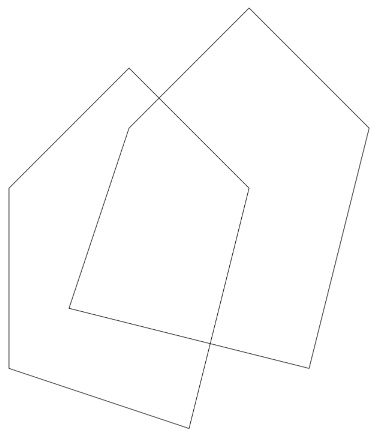
All the code was run on a MacBook Air, with the apple silicon M1 chip.

2 Lab 6

2.1 Sutherland Hodgman Clipping

The implementation of the Sutherland-Hodgman algorithm, seen in clipping.cpp, clips a subject polygon to fit within the boundaries defined by a clipping polygon. This process involves iterating through each edge of the clipping polygon and determining how the vertices of the subject polygon relate to these edges.

The implementation of the Sutherland-Hodgman algorithm uses the "inside" function to determine whether a vertex of the subject polygon lies inside the clipping edge. This is for deciding which vertices to retain or discard during the clipping process. The "intersect" function is used to calculate the exact point where a polygon edge intersects the clipping edge, allowing us to add these intersection points as new vertices in the resulting clipped polygon. Below we can see two polygons before clipping, and after.


(a) Before Clipping

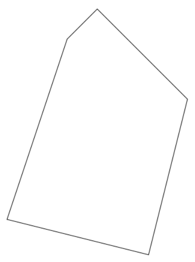
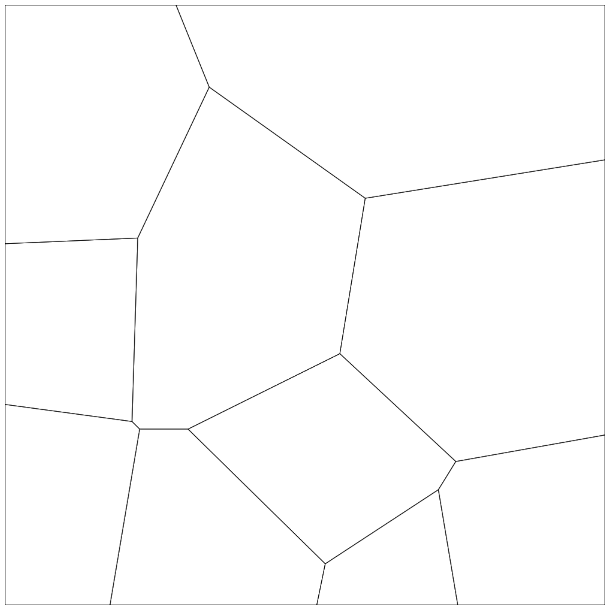

(b) After Clipping

Figure 1: Sutherland-Hodgman Clipping

2.2 Voronoi Parallel Linear Enumeration

We then implemented the Voronoi Parallel Linear Enumeration, found in voronoi.cpp. The images below show the results, which took 0.4ms and 10ms respectively:


(a) n=10

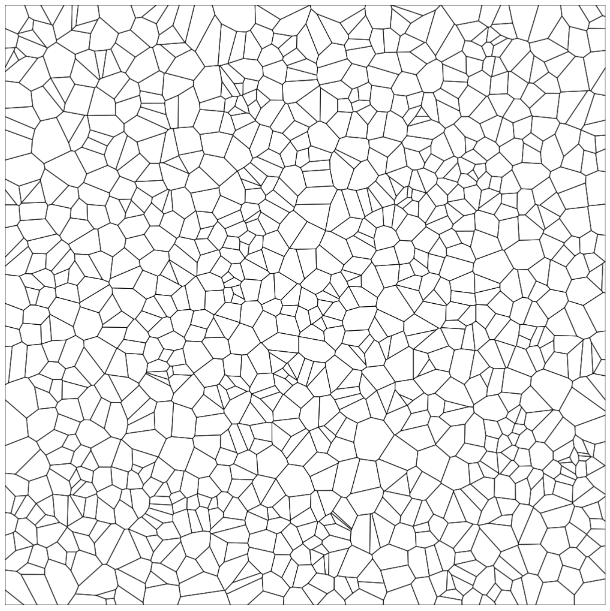

(b) n=1000

Figure 2: Voronoi Diagrams

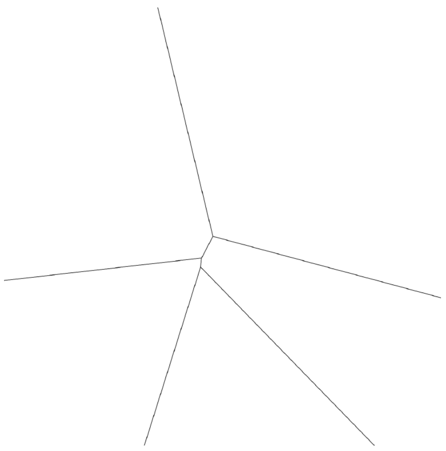
3 Lab 7

3.1 Power Diagram and LFBGS

Power Diagrams and LBFGS have been implemented in power_diagram.cpp. In the implementation, the clipPowerDiagram function clips a subject polygon against the power bisector between two sites, siteA and siteB, each with respective weights weightA and weightB. The algorithm computes power distances for each vertex and determines intersections with the power bisector to construct the resulting polygon.

The PowerDiagram class initializes with a set of points and weights. The compute method iteratively clips an initial square polygon using the clipPowerDiagram function for each pair of sites to generate the power cells.

Below we see the Power Voronoi diagram with 10 polygons and with 1000 polygons. In these diagrams, the weights of the polygons were set to zero if any of their vertices extended beyond the bounds of the square polygon defined by (0.2, 0.8) x (0.2, 0.8). They took 158ms and 2109ms respectively


(a) n=10

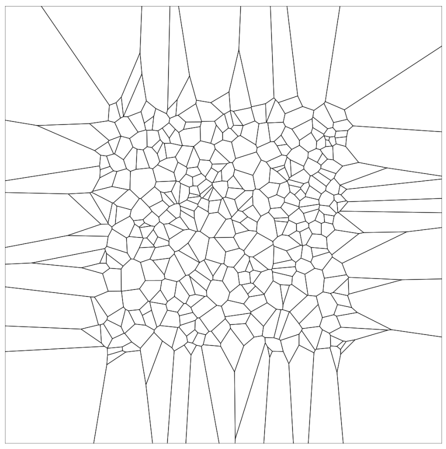
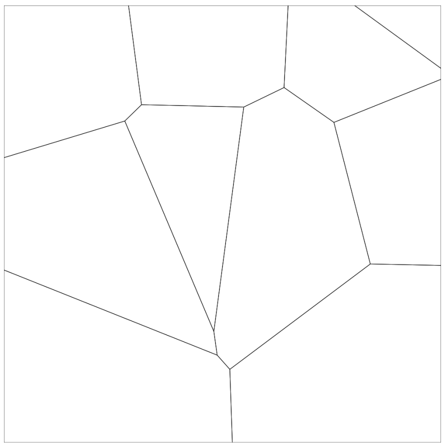

(b) n=1000

Figure 3: Power Voronoi Diagrams

3.1.1 LBFGS

The SemiOptimalTransport class uses LBFGS to optimize weights for semi-discrete optimal transport. The optimize method calls LBFGS to minimize an objective function by adjusting the weights. The evaluate method computes the objective function and its gradient. It calculates the areas of the power cells and the integral of the squared distances from the points. The computePolygons method regenerates the power cells with updated weights, ensuring the optimization progresses towards the optimal transport map. The main function sets up points and weights, initializes the SemiOptimalTransport class, and runs the optimization. Within the SemiOptimalTransport, I used some StackOverflow links to understand the progress and evaluate functions

Below we see the results for n=10 and n=1000


(a) n=10

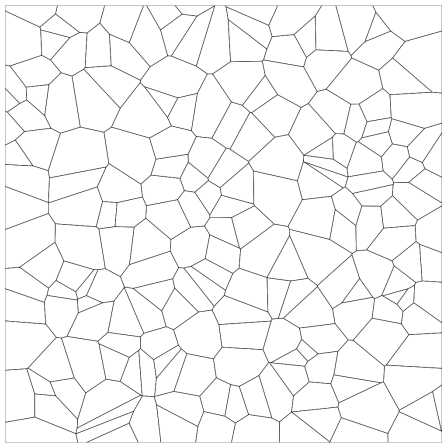
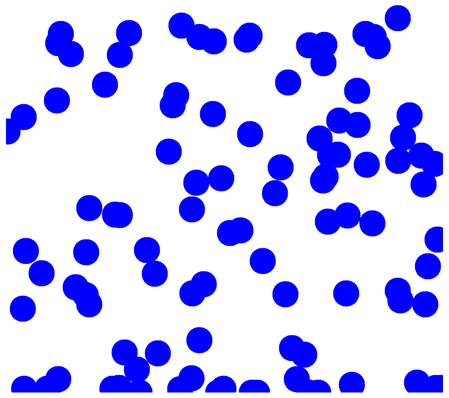

(b) n=1000

Figure 4: LBFGS optimization

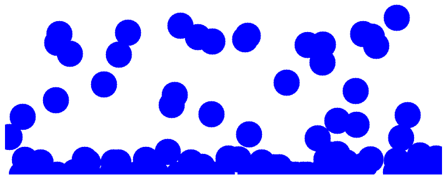
4 Lab 8

4.1 Fluid Dynamics Simulation

The updated implementation simulates particle movement influenced by gravity and spring forces toward Laguerre cell centroids. Particles experience a downward gravitational force and a spring force, with positions and velocities updated based on these forces. A damping factor simulates resistance, and boundary conditions ensure particles stay within bounds, reversing velocities upon collision. Each frame saves Laguerre cells and particle positions as an image, creating an animation. The main loop runs for a specified number of frames, using LBFGS optimization to adjust weights and minimize the objective function. After optimization, particle positions are updated, validated, and saved. The simulation primarily shows vertical movement due to gravity, lacking significant horizontal dynamics. Although not a perfect fluid simulation with Voronoi aspects, it provides basic fluid-like behavior using centroid calculations. The gif was made using python, as I was having trouble working with animating svg files. I also edited the save_frames function a bit to change sizes of the particles, and replaced "Facets" with "Polygons"



(a) gif frame 8



(b) gif frame 25

Figure 5: Fluid Dynamics Simulation