Ray Tracer - Computer Graphics

Ángela Garcinuño Feliciano

June 13, 2025

The objective of this project was to build a free-surface 2D fluid solver using incompressible Euler's equations. We first implemented Voronoï diagram and the Sutherland-Hodgman polygon clipping algorithm. We then extended it to Power diagrams, and built an optimal transport fluid simulator with free surfaces.

1 Overview of Implementation

Throughout this project, we implement several classes:

- **Polygon**: a vector of vertices (type: $vector < Vector >$), where the vertices delimit the polygon. Throughout the project, we equipped this class with *area*, *integral* and *centroid* methods, which compute the area, the integral of the squared distance, and the centroid (the geometric center of the polygon).
- **Voronoï Diagram**: This class initially consisted of a vector of points (type: $vector < Vector >$) and a vector of polygons (type: $vector < Polygon >$). We later added a vector of weights to build power diagrams, and increased the size of the weights vector by one to account for the weight of air in the fluid simulator. We also add a vector of unit disks (type: $vector < Vector >$) to simulate the particles in the fluid simulator. We equipped this class with two methods: *clip_by_edge* and *clip_by_bisector*, used to build the disks and power diagram respectively. Finally, *compute* uses the previous two clipping methods to build the Voronoï (or power) diagram.
- **Optimal Transport**: contains a Voronoï Diagram and an *optimize* method, which aims to optimize the weights to center the diagram around the central points and ensuring we obtain the target area.
- **Fluid**: contains an Optimal Transport, a vector of particles, one of velocities as well as the number of particles *N*, and the volume of the fluid. When initializing, the Fluid object, we initialize the optimal transport, with all particles generated around the center point $(0.5, 0.5)$. We equip it with two methods. *time_step* simulates a time step, updating the velocities and positions of each particle and optimizing the optimal transport. *Run_simulation* runs calls iteratively *time_step* to run a fluid simulation.

2 Clipping and Voronoï Diagrams

We begin with a Voronoi Diagram class, which has a vector of points and one of cells as attributes. We equipped it with two methods: *clip_by_bisector* and *compute*. Clip_by_bisector implements the Sutherland-Hodgman polygon clipping algorithm which aims to restrict a polygon to the inside of another convex polygon. This algorithm takes a polygon and a half line (represented as two points). It returns a new point by adding edges, creating a new edge by taking intersection between an edge and the half line, and ignoring edges, depending on the case. Compute builds a Vornoï diagram, starting from a polygon (a square) and iteratively use *clip_by_bisector* to build the diagram.

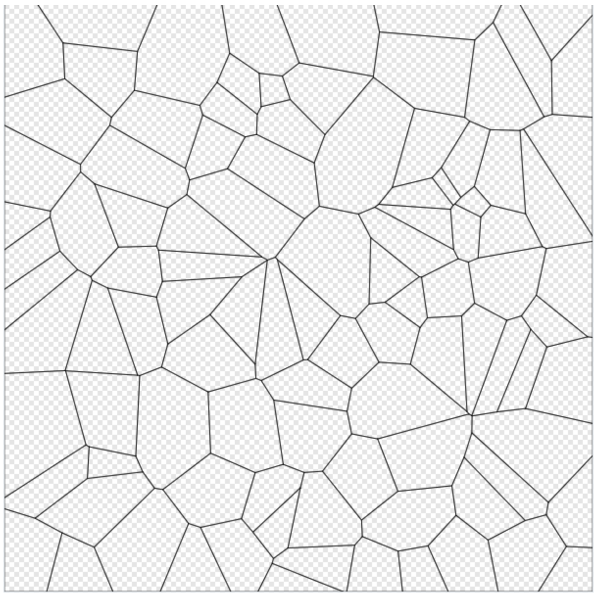

Figure 1: Voronoï diagram generated with 100 cells using the Sutherland-Hodgman clipping algorithm. Computed in 10,421 microseconds.

3 Power Diagrams

We first extended our Voronoï diagram implementation to Power diagrams by adding weights to the points. We then used a semi-discrete optimal transport to optimize the weights using L-BFGS.

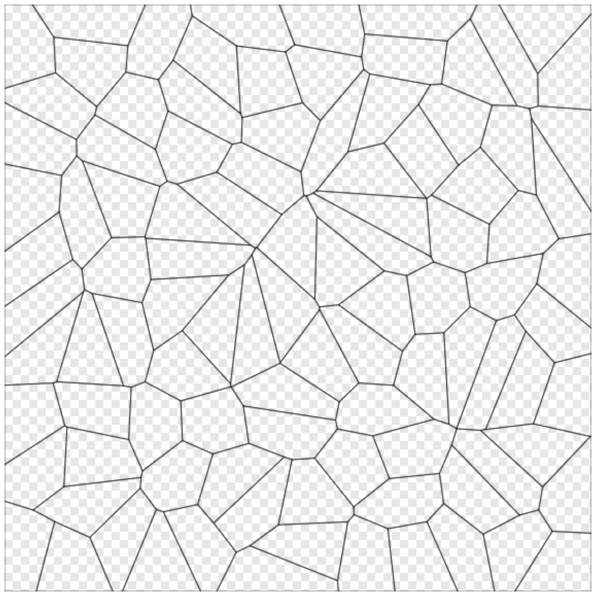

Figure 2: Power diagram with 100 cells, incorporating weighted points. Computation using the semi-discrete optimal transport method completed in 79,104 microseconds.

To test the implementation, we modified the function to aim for a target area, instead of having the same area for each cell (i.e. $\frac{1}{N}$). We define the target area to be the distance from the point to the center $(0.5, 0.5)$, obtaining Fig. 4 as a result. We thus conclude that the algorithm works properly.

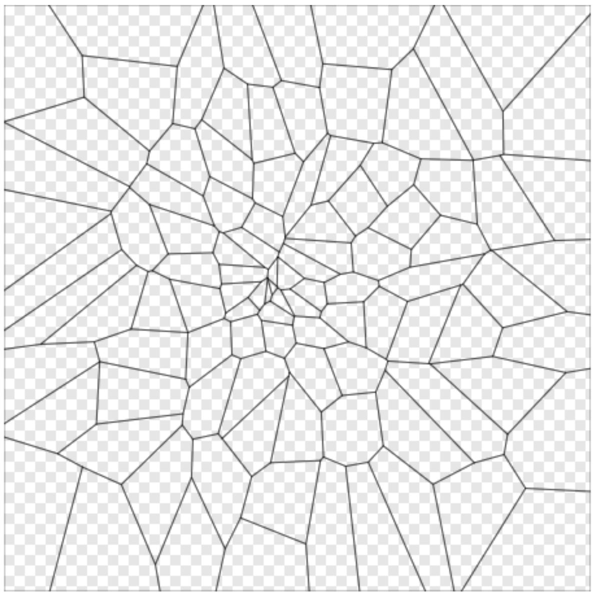

Figure 3: Power diagram with 100 cells, where each cell's target area is based on its distance to the center point (0.5, 0.5). The result reflects spatially varying cell sizes and was computed in 103,603 microseconds.

4 Fluid simulator

Finally, we used the implementation to make a fluid simulator. To do this, we added one more value to the list of weights, w_{air} which contains the weight of the air. We initialize it to 0.9 and the weights of the cells (water particles) to be 1. We first set the area of the Power Diagram cells to be $\frac{f}{N}$, where f is the percentage of the space that is filled with the fluid (the rest being air) and N is the number of particles. Then, the area of the fluid is distributed evenly between each particle. Then, we clip each polygon by a disk of appropriate size ($\sqrt{w_i - w_{air}}$, w_i the weight of the i-th particle). Finally, we make the simulation by taking several frames and modifying each particle depending on it's velocity, considering gravity and spring force. At each time step we optimize the weights, and update the particles using the velocity.

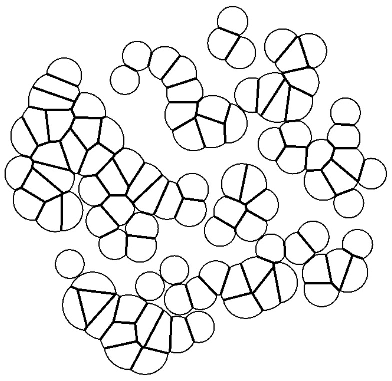

Figure 4: Initialized Fluid simulation: 200 particles