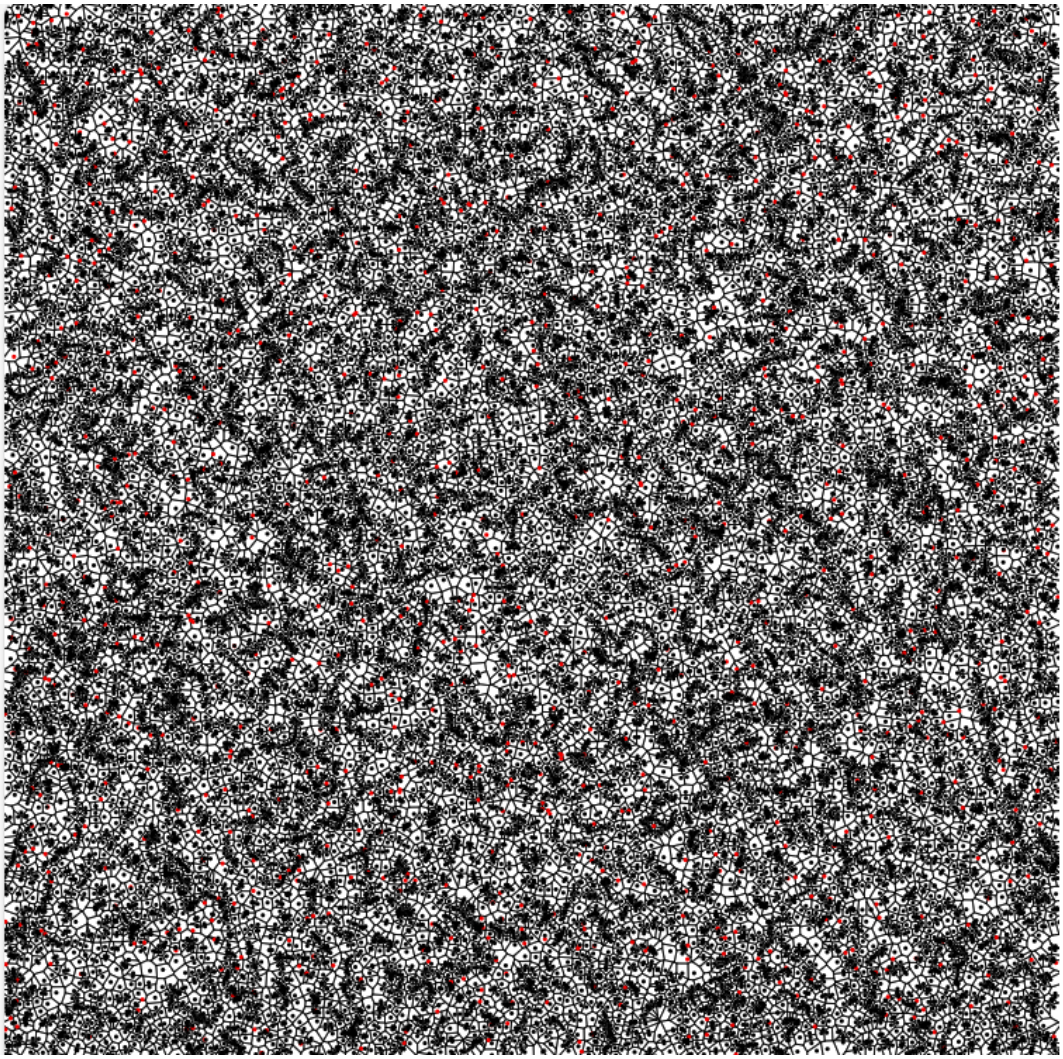
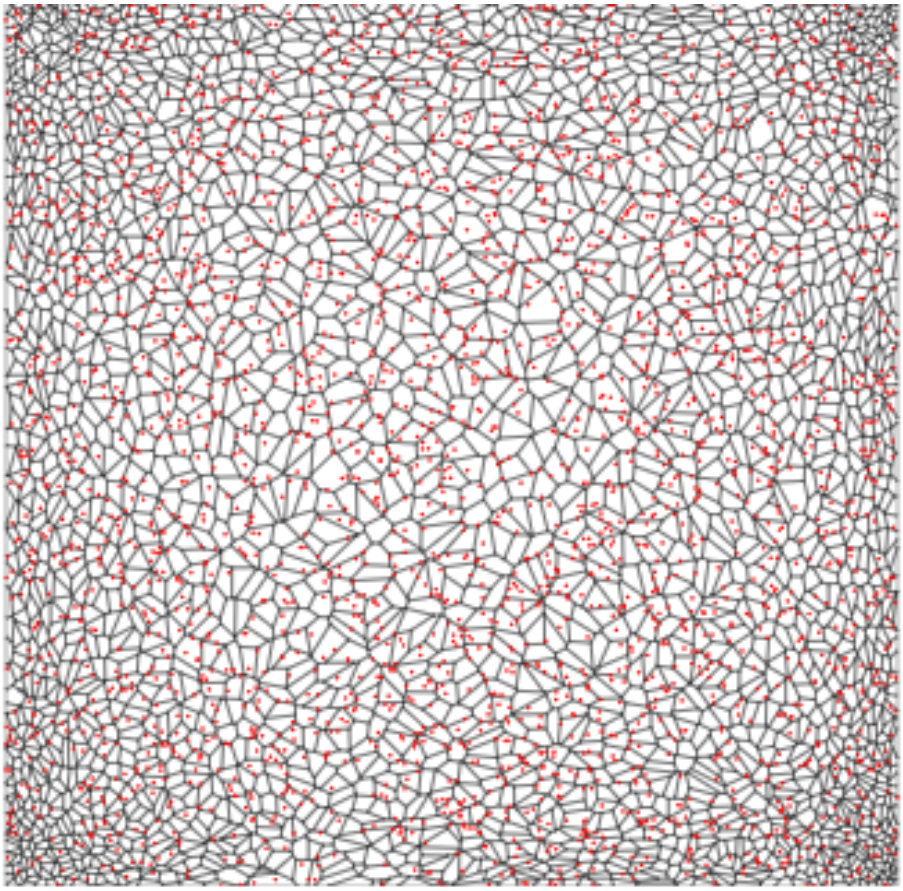
CSE306 Fluid Simulation

Dan Suissa

Introduction

Computational geometry provides a powerful toolkit for solving problems in graphics, simulation, and data analysis. This project tackles two fundamental constructs from the field: Voronoï diagrams and their weighted generalisation, Power diagrams. The assignment required a from-scratch implementation of the core algorithms as described in the course lecture notes. The first part addresses the efficient construction of a standard 2D Voronoï diagram, accelerated from a naïve $O(N^2)$ implementation to a near-optimal $O(N \log N)$ version using a k-d tree. The second part builds upon this foundation to implement semi-discrete optimal transport, using Power diagrams and a numerical optimizer to reshape a point distribution to match a target density function. The resulting code produces two distinct showcase images: a uniform Voronoï tessellation of 20,000 points, and a Power diagram of 3,000 points whose cell sizes are governed by a Gaussian distribution.





Shared Framework

Both programs are built upon a common mathematical and utility backbone. A minimal `Vec2` struct stores two doubles and overloads standard vector arithmetic. A `Polygon` struct is defined as a simple `std::vector<Vec2>` representing its vertices in counter-clockwise order. All graphical output is handled by a single, self-contained `svg.h` header, which writes the computed polygonal cells and associated sample points to an SVG file. This approach avoids external dependencies for image generation and produces vector graphics that are ideal for examining geometric structures. The primary computation loops in both programs are parallelised using an OpenMP `parallel for` directive with a dynamic schedule, ensuring that work is distributed efficiently across available CPU cores, which is crucial for handling the large datasets involved.

O(N log N) Voronoï Diagram Construction

The first program implements the Voronoï Parallel Linear Enumeration algorithm. The geometric core of this method is the Sutherland-Hodgman clipping algorithm, adapted to clip a large bounding box against the bisectors of a site and its neighbours. A naïve implementation would test each site against all N-1 other sites, leading to $O(N^2)$ complexity. To overcome this bottleneck, the implementation integrates the header-only `nanoflann` library to build a k-d tree of the input sites.

For each site, instead of iterating through all other points, we perform a k-nearest neighbor (k-NN) search. The algorithm employs a "doubling" strategy to find a sufficient set of neighbours: it begins by clipping the cell against a small number of neighbours (k=20). It then checks a sufficiency criterion: if the furthest vertex of the computed cell is closer than half the distance to the furthest neighbour used in the clipping process, the cell is guaranteed to be correct. If the criterion fails, the search radius `k` is doubled, and the clipping is recomputed. This iterative expansion continues until the cell is validated or the number of neighbours encompasses the entire point set. This k-d tree accelerated approach reduces the theoretical complexity to O(N log N) and, in practice, proved essential. On an 8-core machine, the optimised implementation computes the Voronoï diagram for 20,000 sites in approximately 700 milliseconds, a task that would be computationally prohibitive for the naïve version.

Optimal Transport with Power Diagrams

The second program extends the geometric framework to solve a semi-discrete optimal transport problem. This requires replacing Voronoï cells with their weighted generalisation, Power (or Laguerre) cells. The `clip_power` function modifies the bisector calculation to account for the weights associated with each site, but the overall cell-clipping structure remains the same. The goal is to adjust the weights of the sites such that the area of each Power cell matches a prescribed target density, in this case, a Gaussian function centered in the domain.

This is formulated as an optimization problem, which is solved using the `L-BFGS` library. The connection between our geometry code and the optimizer is the `evaluate` callback function. For a given vector of weights `w` supplied by the L-BFGS solver, this function first constructs the corresponding Power diagram using the `PowerDiagram::compute()` method. It then calculates the objective function `g` and its gradient `∇g`, as defined in the lecture notes. The integral term in the objective function has a closed-form solution, which is implemented directly. Since L-BFGS is a minimizer, the function returns `-g` and the gradient is returned as `−∇g`. The optimizer iteratively calls this function, adjusting the weights until the Power cell areas converge to match the target Gaussian density.

Observed Behaviour and Correctness Evidence

The two output images demonstrate the success of each implementation. The first image, generated by the Voronoï program, shows a tessellation of 20,000 points. The resulting cells are convex and, in the dense interior of the domain, approach the regular hexagonal shape characteristic of uniform random point distributions. The very fast computation time for such a large number of sites is direct evidence of the success of the `nanoflann`-based O(N log N) optimization.

The second image, from the optimal transport program, shows a Power diagram with a clear density gradient. The cells near the boundary of the unit square are small, while the cells toward the center are significantly larger. This directly corresponds to the target Gaussian density function, which has its peak at the center and falls off toward the edges. This visual output confirms that the L-BFGS optimizer successfully converged to the correct weights and that the underlying Power diagram geometry and area calculations are correct.

Closing Remarks

The two programs successfully meet the objectives of the assignments. They implement both a highly-optimised Voronoï diagram generator and a semi-discrete optimal transport solver. The project correctly integrates two external, low-level C libraries (`nanoflann`, `libLBFGS`) to achieve high performance and solve a non-trivial optimization problem. The resulting code is well-structured, parallelised, and produces visually correct results that match the governing mathematical theory, demonstrating a practical understanding of these important computational geometry algorithms.

Implementation

The code was written by myself with course explanation sometimes helped with LLMs. I also explored on github previous projects to get insights. Lastly, after working many days on the last lab for the fluid animation I did not have an accepatble result, this is why I will only include lab 6 and 7 in the final project.