Computer Graphics Assignment 2

Tu Nguyen

June 12, 2024

1 Overview

Github repository: `https://github.com/X24hedgehog/Project-2-CSE306`

In this project, I implement 3 compulsory labs and 2 ungraded lab of the course CSE306 in C++, each lab has its own objective:

- Lab 5: Sliced optimal transport approach for color matching (Ungraded)
- Lab 6: Voronoï Parallel Linear Enumeration algorithm in 2D
- Lab 7: Semi-discrete optimal transport in 2d
- Lab 8: Semi-discrete optimal transport fluid simulator with free surfaces
- Lab 9: Tutte-embedding (Ungraded)

2 Achievement

- I have implemented lab 5 and it works. Results can be seen in my repository
- For lab 6 and 7, the code run and produce results as expected (results can be seen below in the report). Running time is high since I did not implement the k-d tree.
- For lab 8, I come across error when running the lbfgs for fluid simulation (error code: -1001, rounding error). Still, the code works and produces results as I expected.
- I have implemented the algorithm in lab 9 and I will test it in the future.

3 Code Structure (Graded labs)

All the code are in the file `main.cpp`.

Main classes used in the file:

- Vector: reuse of the Vector class in ray-tracing project. The only difference is that I remove the third dimension as the second project works on 2-D space. All the vector operation is kept, except for a minor change in the cross function.
- Polygon: an object of this class consists of a list of vertices. Main functions in this class are:
  - check inside, check inside voronoi: check if a point P is in the counter-clockwise half lane with respect to an edge, with the direction of the edge from the first point of the edge to the second point of the edge
  - check intersect, check intersect voronoi: check if a segment of 2 points intersect with an edge
  - clip edge voronoi, clip polygon: convert the subject polygon to the polygon after being clipped using the Sutherland algorithm (for clip edge voronoi, the subject polygon is clipped by an edge, for clip polygon, the subject polygon is clipped by another polygon)
  - compute centroid: return the center of a polygon, used in lab 8 in the Gallouet Merigot algorithm
  - compute area: return the area of the subject polygon, used to optimize the function f in lab 7 and lab 8

There are other functions served as helping function:

- compute voronoi: use the clip edge voronoi to return the list of polygons as the output voronoi (include weighted version)
- generate disk: return a disk with known center, radius, and number of edges. This function is used to approximate the polygons as disks in lab 8
- intersect disks: return a list of polygons by computing the voronoi polygons and then approximate them with the disks generated by the previous function

There are 2 lbfgs functions for the optimization of the 2 labs 7 and 8: evaluate for lab 7 and fluid evaluate for lab 8.

The function gallouet merigot scheme to update the polygons after each frame in order to simulate the fluid.

Finally, there are 3 main functions in the main function of the file main.cpp. You can uncomment the function you want to run to get the result:

- Lab 6 and first half of lab 7: generate voronoi (Generate voronoi diagram with weights, where unweighted voronoi are created with all weights being equal). Results are stored in "test balance image.svg" and "test weighted image.svg"
- Second half of lab 7: generate optimized voronoi (Generate voronoi diagram with optimized function using lbfgs), where the cells area get bigger in the center and smaller further to the edges. Results are stored in "test lbfgs weighted image.svg"
- Lab 8: fluid simulation (Generate animated images, corresponding to frames of the fluid). Results are stored in "test fluid simulation.svg"

4 Result images

For the fluid simulation, please check my repository, which is put at the beginning of my report, to get the svg files and you can view the animated simulations with preview mode.

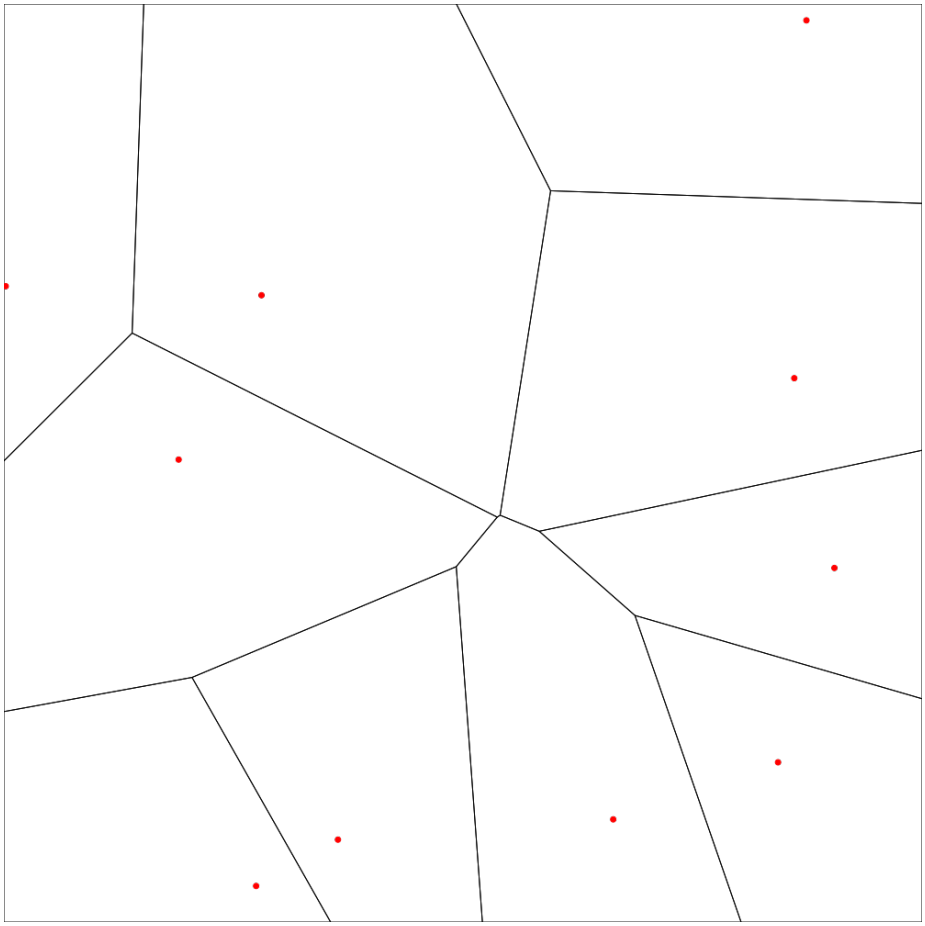

Figure 1: Unweighted voronoi - 10 points, running time: approximately 10^{-5} seconds

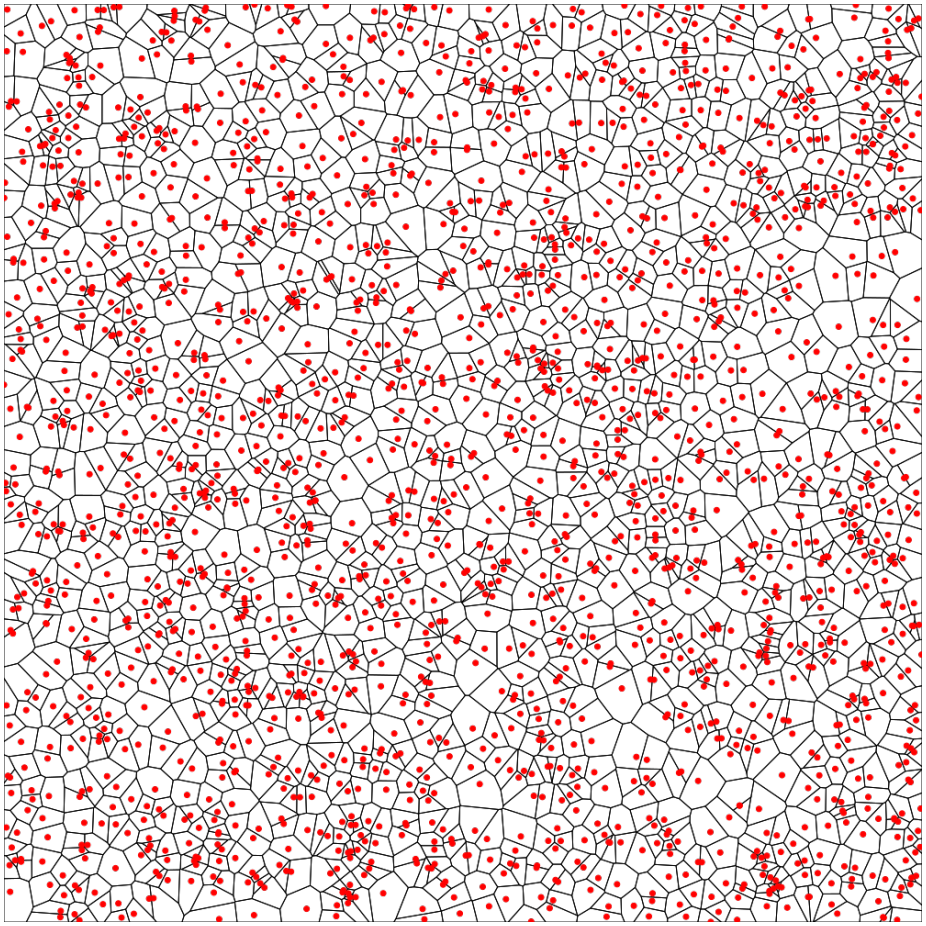

Figure 2: Unweighted voronoi - 2000 points, running time: approximately 8.5 seconds

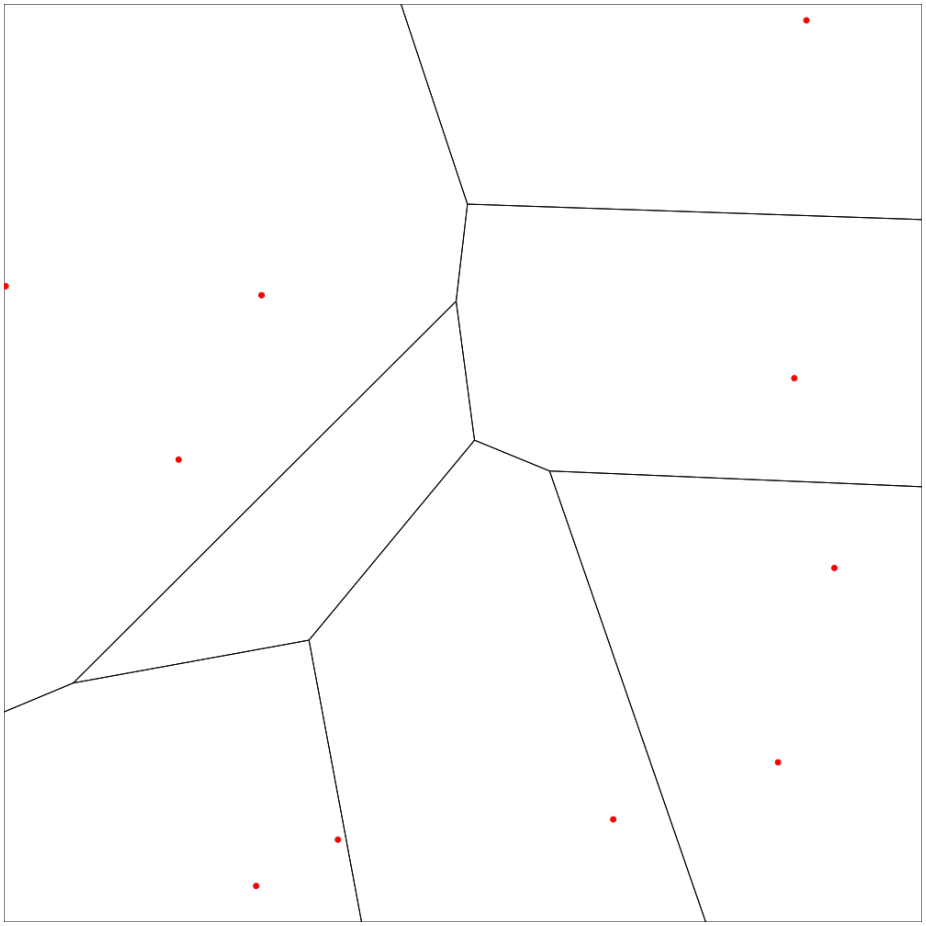

Figure 3: Weighted voronoi - 10 points, running time: approximately 10^{-5} seconds

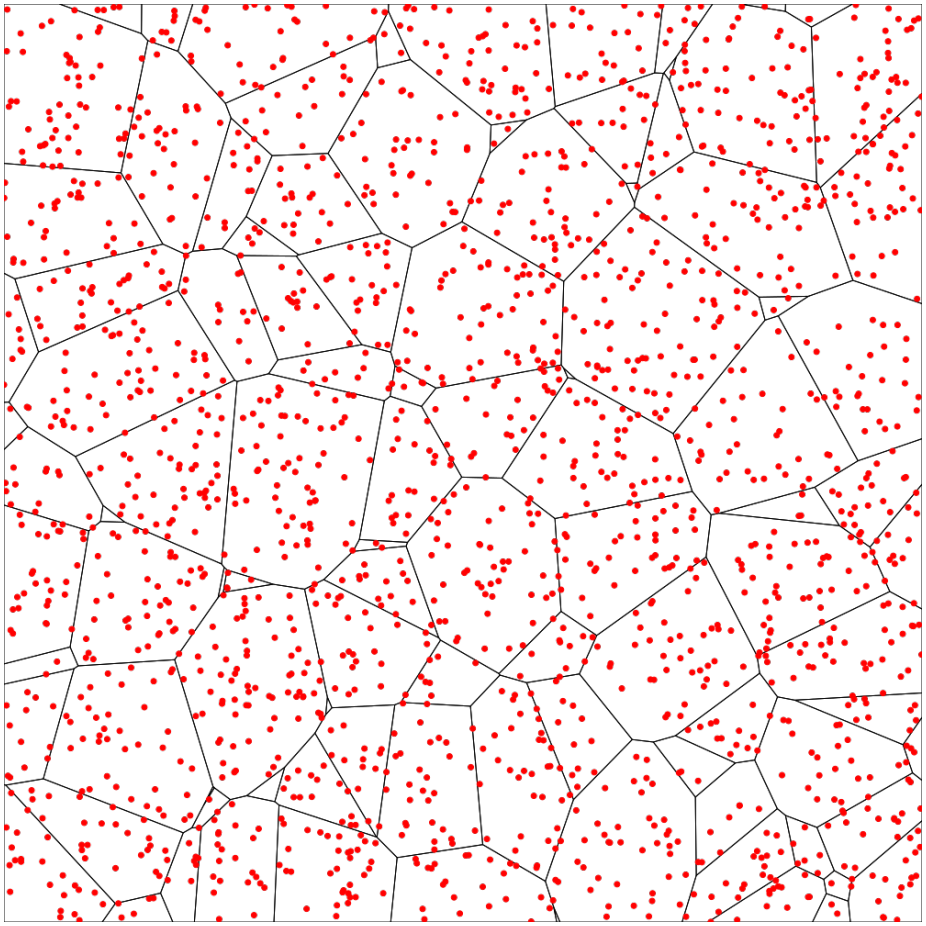

Figure 4: Weighted voronoi - 2000 points, running time: approximately 8.5 seconds

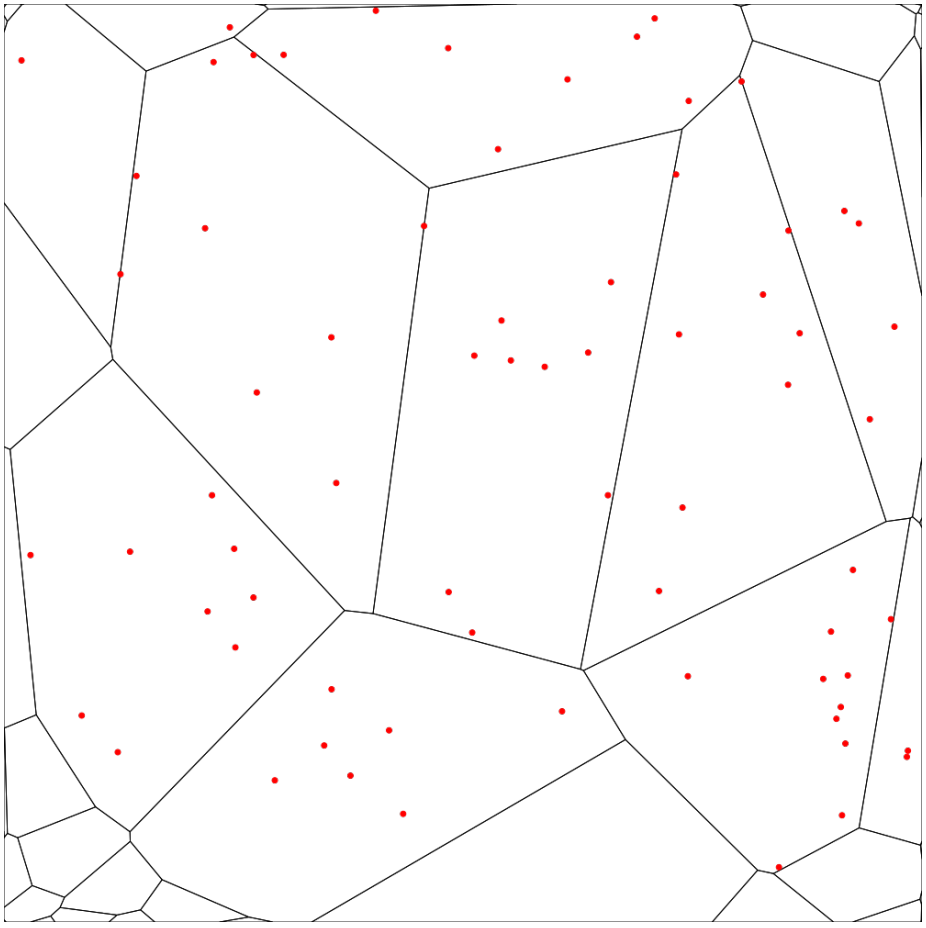

Figure 5: Optimized weighted voronoi - 70 points, running time: approximately 19.75 seconds

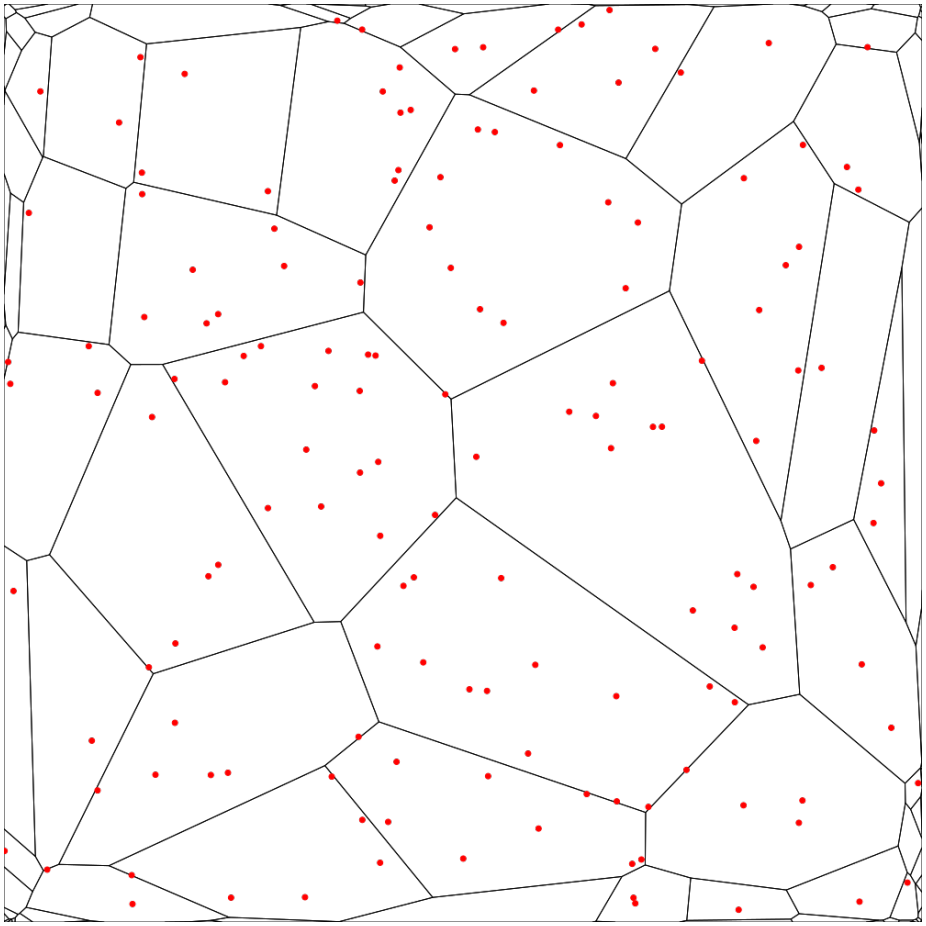

Figure 6: Optimized weighted voronoi - 150 points, running time: approximately 115.2 seconds

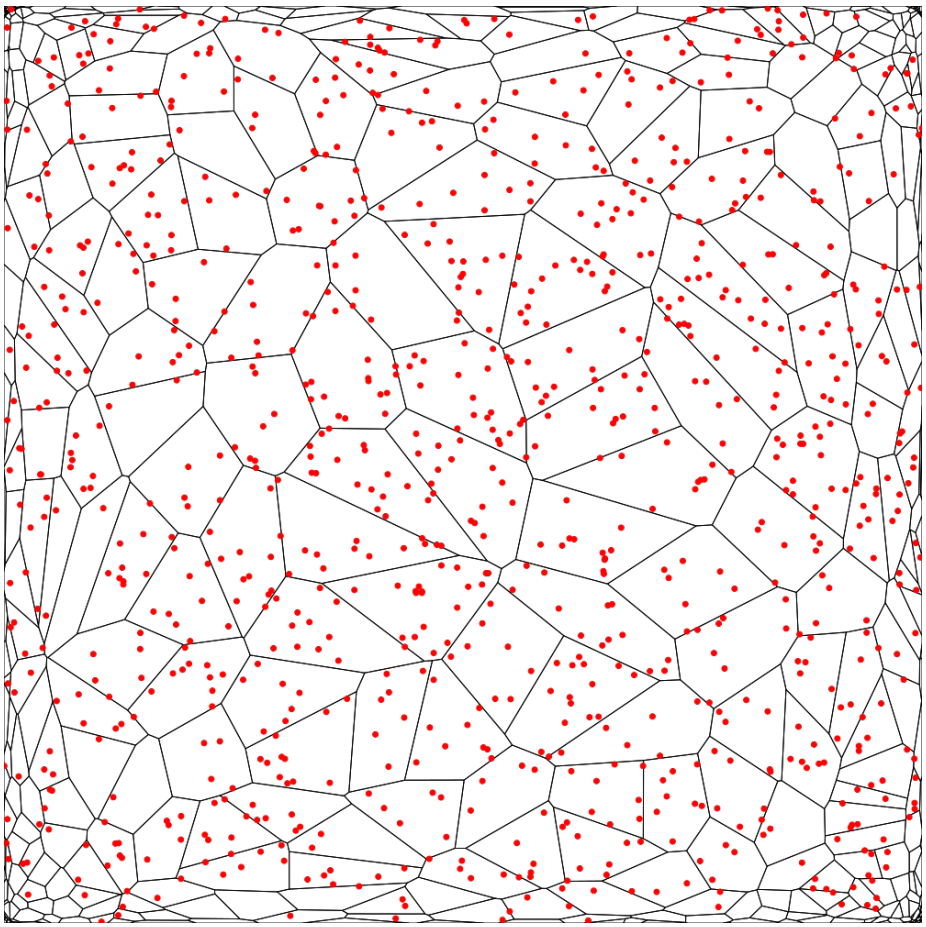

Figure 7: Optimized weighted voronoi - 1000 points, running time: approximately 1.5 hour

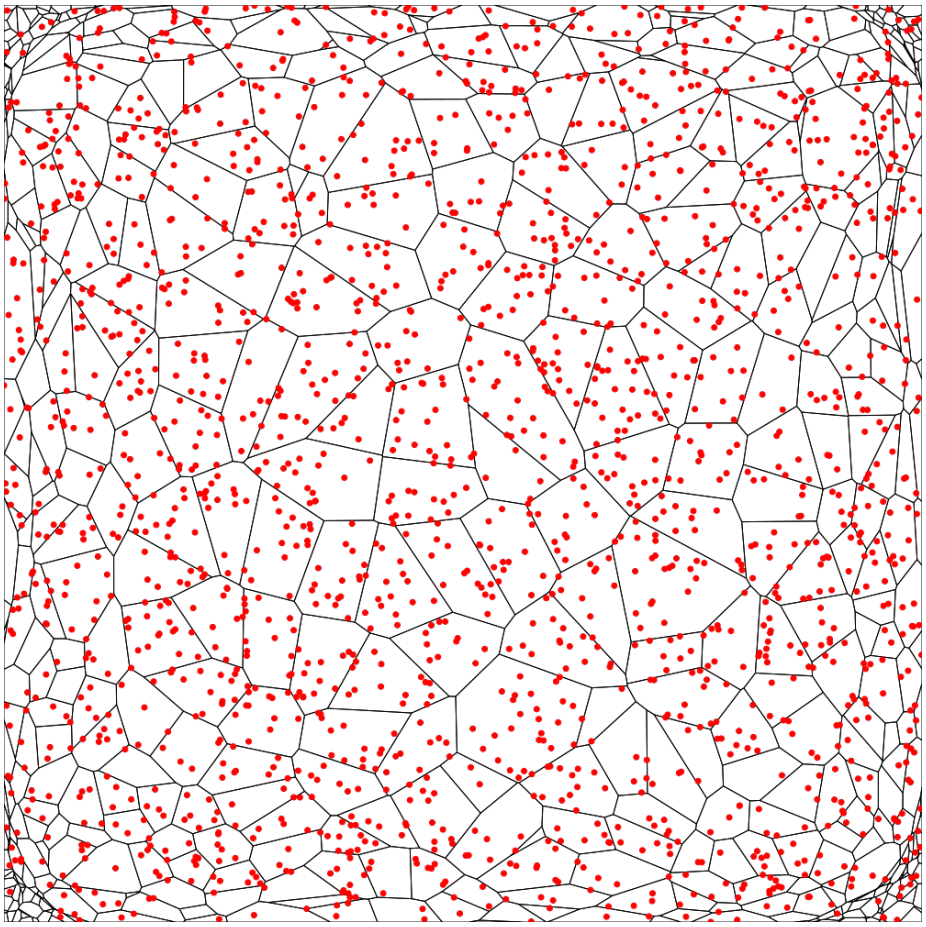

Figure 8: Optimized weighted voronoi - 2000 points