Computer Graphics, CSE306, Assignment 2

Natali Gogishvili

June 17, 2024

1 Introduction

As part of this project I implemented all the required parts of TDs 6, 7, and 8. Because of having some difficulties with the evaluate function of lbfgs, I did not have time to work on optional parts for the second project. For implementation I used lecture notes, slides, and code from live sessions during the TDs.

2 Notes

I had problem with save_frame function provided by the professor to display the png file. I asked for help to Dimitrije Zdrale, who helped me fix it. I used his solution (adding sgn function) in save_frame.h file. Also, I used ChatGPT's help for producing the python file for generating the gifs from png files.

3 Voronoi and Power diagrams

For TD 6 I implemented calculation of Voronoi diagram given random points using Sutherland-Hodgman algorithm. You can find results on figures 1 and 2. For TD 7 I added weights. You can find results in figures 3 and 4. Everything is located inside `Voronoi` class in project_2.cpp file. `clip_by_bisector` function clips the cell given 2 points and weights. This function is used by `compute` function which uses points and weights to construct a point diagram. There are several additional functions inside the class but they are only used for fluid simulation. `clip_by_circle` function clips a cell with a circle centered at it's point with a provided radius. `compute_fluid` is very similar to `compute` with the only difference that shapes are clipped by circles after the calculation of usual power diagram.

4 Semi-Discrete Optimal Transport

To do semi-discrete optimal transport, I used LBFGS library. For this to work I had to write evaluate function. I used formulas on page 103 of lecture notes for the formula and the gradient and formula 4.12 to calculate the integral of

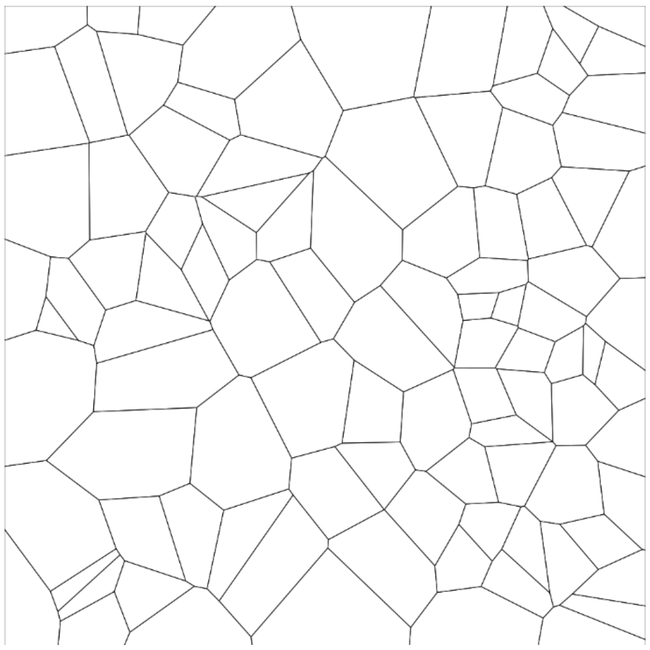

Figure 1: Voronoi diagram, no weights, 100 random points

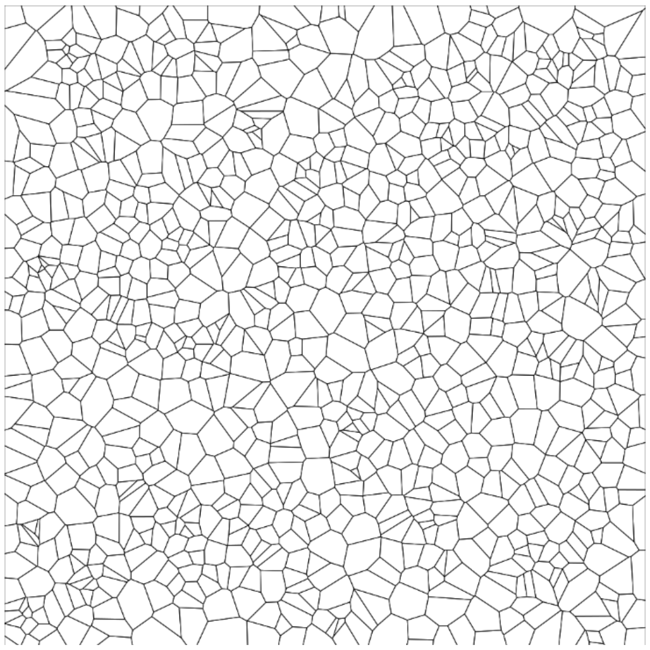

Figure 2: Voronoi diagram, no weights, 1000 random points

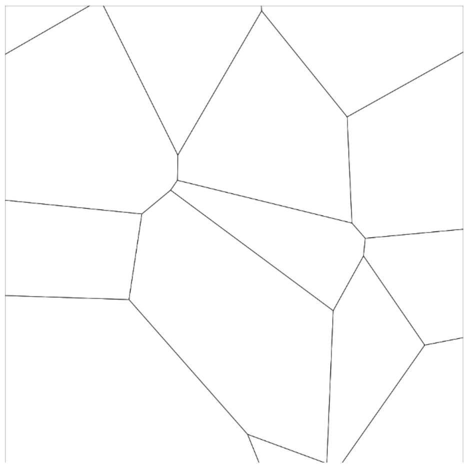
Figure 3: Power diagram, 1000 random points and random weights

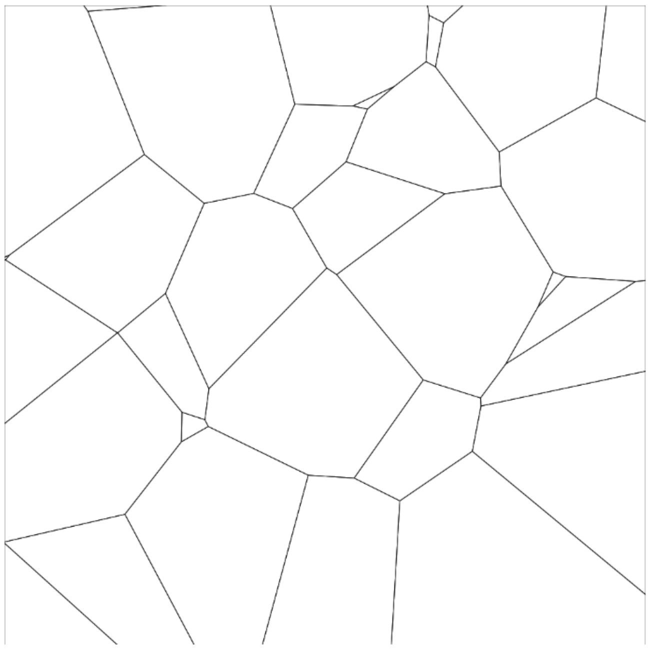
Figure 4: Power diagram, 1000 random points and random weights

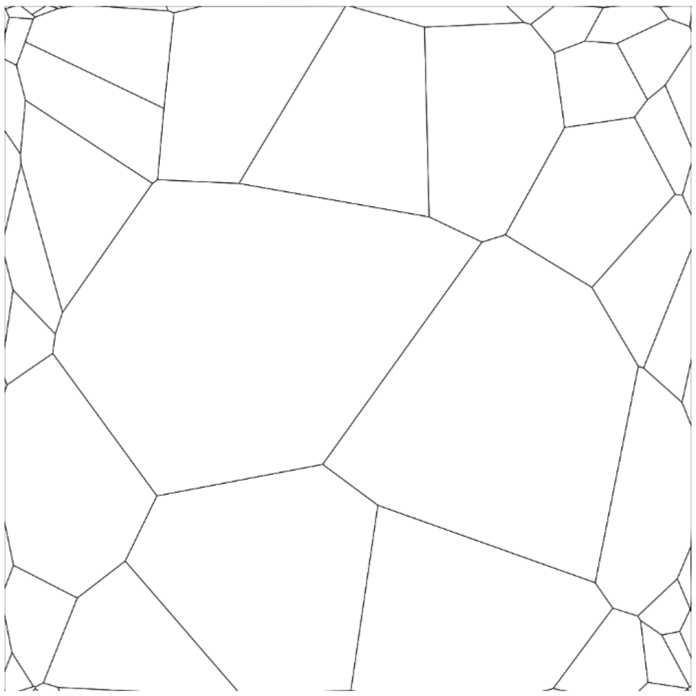

Figure 5: Optimized diagram. Lambdas are $exp(-\frac{1}{0.02}||points_j - C||^2)$ as in lecture notes. 100 points. Took 871 iterations with status code 0.

distances. I have to note that debugging this took a lot of time with optimization ending with failure code -1001 most often. I managed to solve this issue by making sure that all numbers were being interpreted as doubles. I also had to correct indexing issues inside integrals. In the end it worked and now the code runs with success code 0 on any lambdas and any weight initialization I tried (I tried several values from 0 to 1). We have to make sure that lamdas sum to 1 since it is a distribution. Results can be seen on figures 5 and 8.

5 Fluid simulations

For simulating fluids I used two different methods. First, I used the previously written evaluate function and only adjusted lambdas appropriately. You can find the result in one_air_cell.gif of output_gifs folder. For water particles I put lambdas be the ratio of water on the image divided by number of water particles. For air I used lambda value one minus the water ratio. Code for this can be found in project_2_2.cpp. It converged at every point with status code 0. Note that almost everything in this file is the same as in project_2.cpp. I created this file since I wanted to try the air bubble implementation but did not want to contaminate the main project_2.cpp file with extra not very useful code.

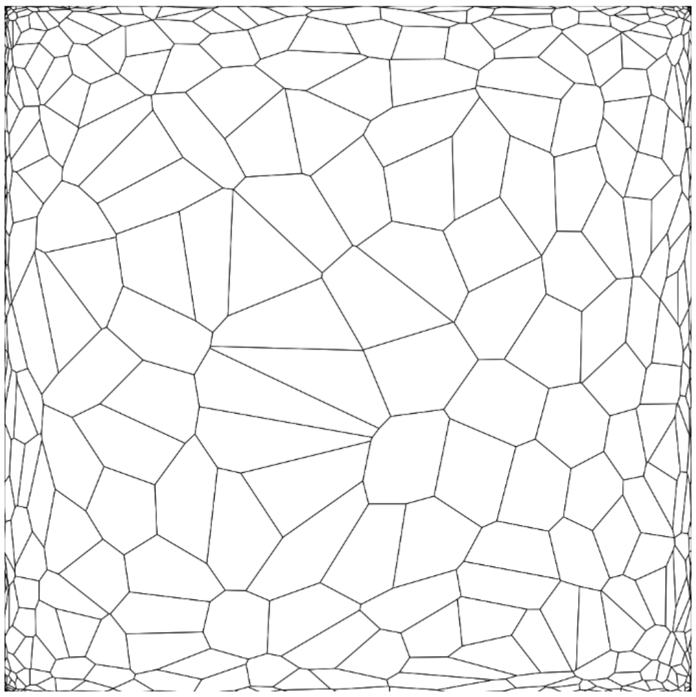

Figure 6: Optimized diagram. Lambdas are $exp(-\frac{1}{0.02}||points_j - C||^2)$ as in lecture notes. 1000 points. Took 1318 iterations with status code 0.

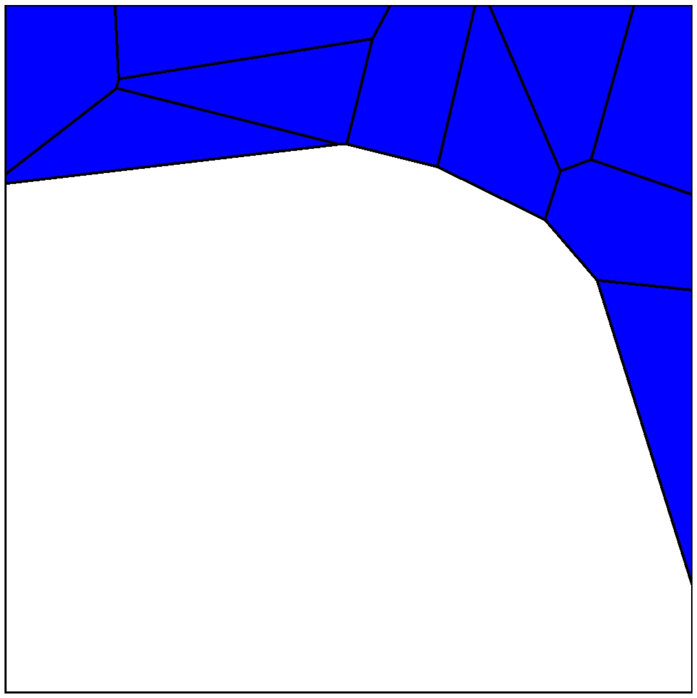

Figure 7: Fluid simulation. One air bubble and 10 water particles. One frame. eps=0.004, mass=200, dt=0.02.

Second, more advanced implementation is inside project_2.cpp file using a different code for fluid optimization. This implementation corresponds to the limit to infinity of the number of air particles. I use `compute_fluid` function instead of `compute` as mentioned before for calculating the power diagram in this case. We have to note that there was a problem with lbfgs in this case. It gave status code -1001, which can be due to numerical errors. I could not manage to solve this problem before the submission of the project. However, it still runs several iterations per timestamp and results seem reasonable. The problem is that sometimes there is a big difference between shapes of the cells, even though all water cells have same lambdas.

We provide results with different number of water particles and epsilon sizes in output_gifs folder.

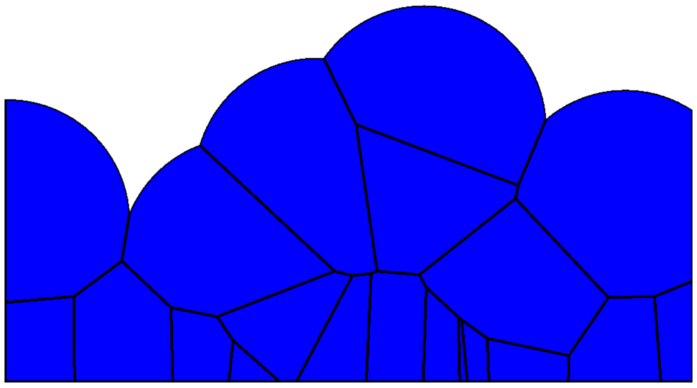

Figure 8: Fluid simulation. 20 water particles. Limit to infinity. One frame. eps=0.004, mass=200, dt=0.02.