CSE306 Project - Fluid simulation

Philippe Guyard

Code structure

Running the project

In this project, I implemented all mandatory labs as well as one optional feature. I used Cmake as my main build tool, and VsCode as my editor. I do not know if Visual Studio parses CMakeLists files, but if not, the project can be compiled using the command

```
$ cmake . && make
```

Alternatively, I also made a convenience script. What it does is

1. Build the project
2. Run the executable, and forward its output to a file `sim_out.txt`
3. Make a video out of the resulting frames

You can run it on linux using

```
$ sh make_vid.sh
```

File structure

Here is a description for all the project files:

- `main.cpp`: This file is just some configurations for logging and which Voronoi algorithm to use. It creates a fluid instance and saves the frames.
- `fluids` directory:
  - `fluids/vector.*` Same as in raytracer, but for 2 coordinates
  - `fluids/poly.hpp` An implementation of the polygon class

  - `fluids/diagram.hpp` Two implementations of a power diagram: one uses the $O(n^2)$ Power diagram construction algorithm, whereas the other one uses `nanoflann` for a $O(nlogn)$ algorithm. They are interchangeable and are set using define directive. To use the slow version, add the line `#define VORONOI_ALGORITHM 0` to the top of the file. The default is the fast version
  - `fluids/ot.hpp` This file contains two classes. The first one is a `FluidPowerDiagram`: it inherits from the normal PowerDiagram, but has an extra weight (the weight of air particles), and clips all PowerDiagram polygons by a disk (in class we had a disk with 50 vertices. I noticed that clipping took a long time, so I decided to reduce it to 30). The second class is `OptimalTransport`. It uses L-BFGS to perform a semi-discrete optimal transport.
  - `fluids/fluid.hpp` This class does the fluid simulation, initializing water particles at random and then applying forces to them iteratively.
  - `fluids/vov_adaptor.h` This is just from the nanoflann library, it was necessary to make the kd-tree work
  - `fluids/kd.hpp` An unfinished attempt of my own kd-tree implementation that I plant to finish later

- `benchmarker.hpp` This is a tool that I use to measure the performance of different parts of my code After execution, my program prints a 'Benchmarking summary', breaking down the runtime into different parts.

The project also contains headers for the lbfgs and nanoflann libraries.

Pictures

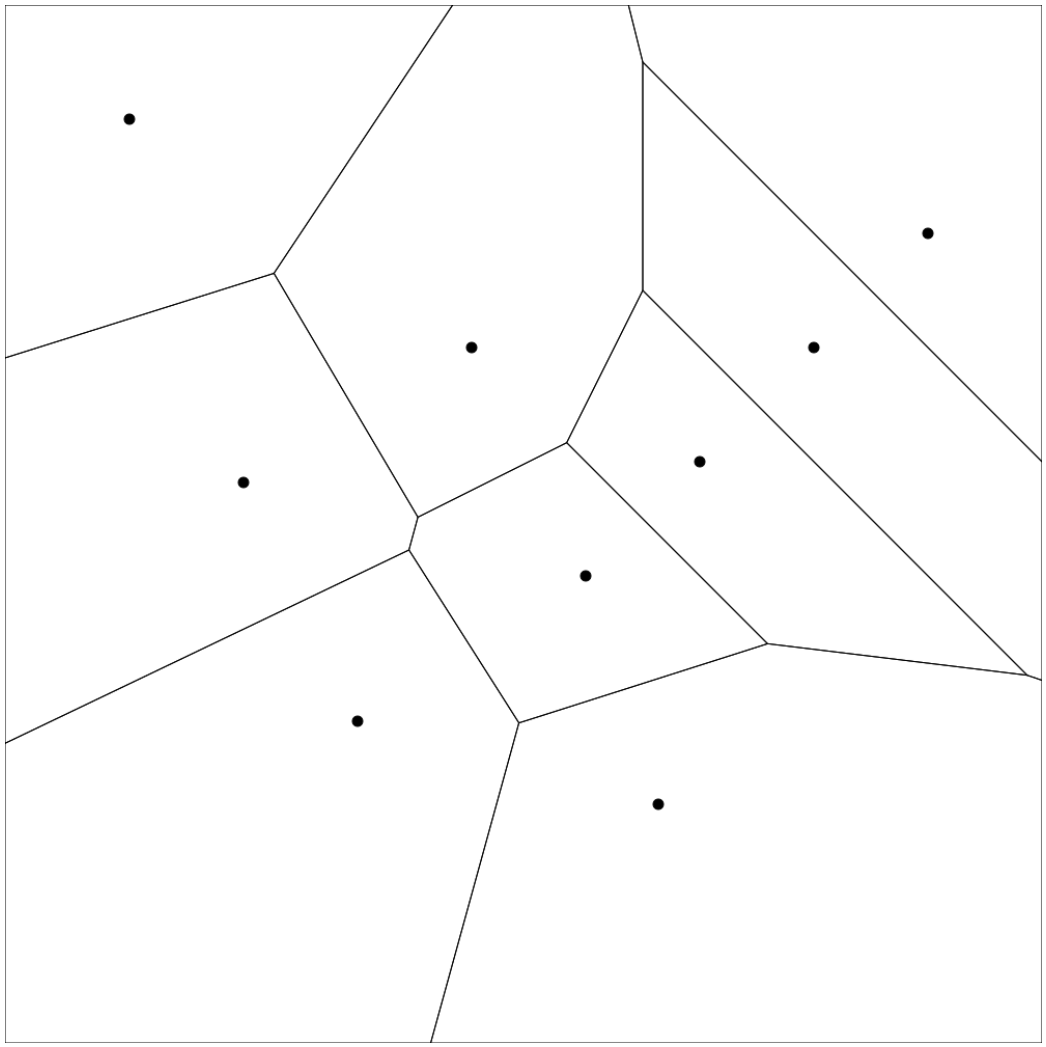


Figure 1: Example of Voronoi diagram for 9 points

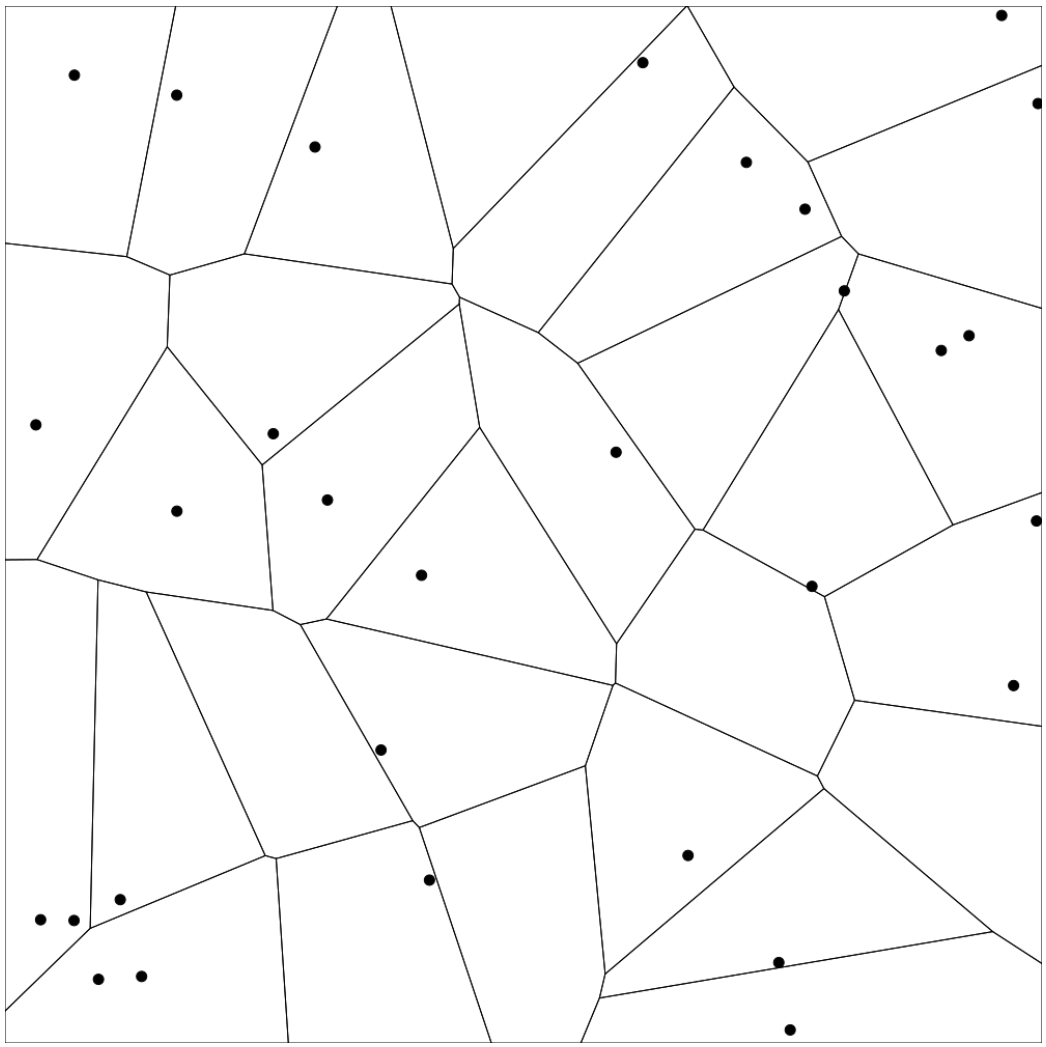


Figure 2: Example of Optimal Transport result for 30 points generated at random with uniform densities

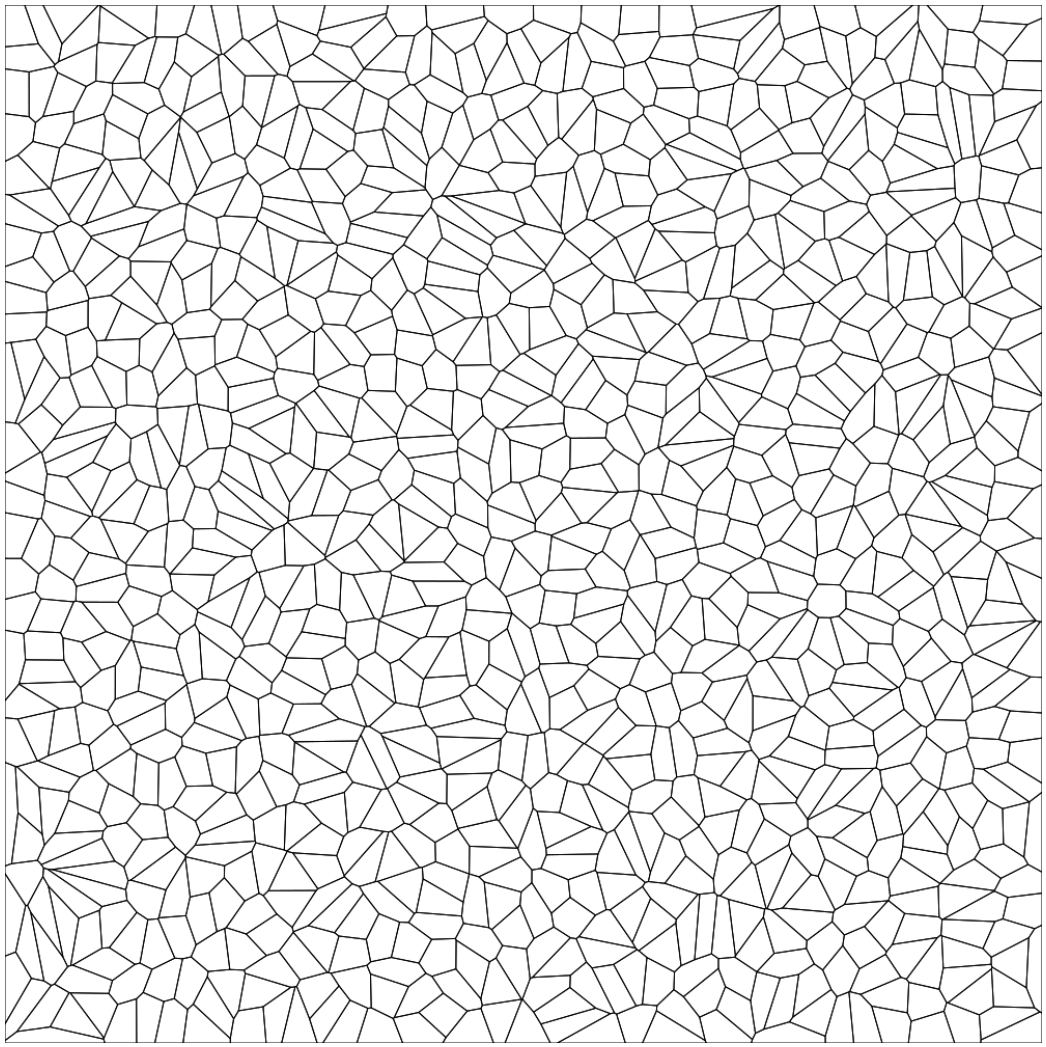
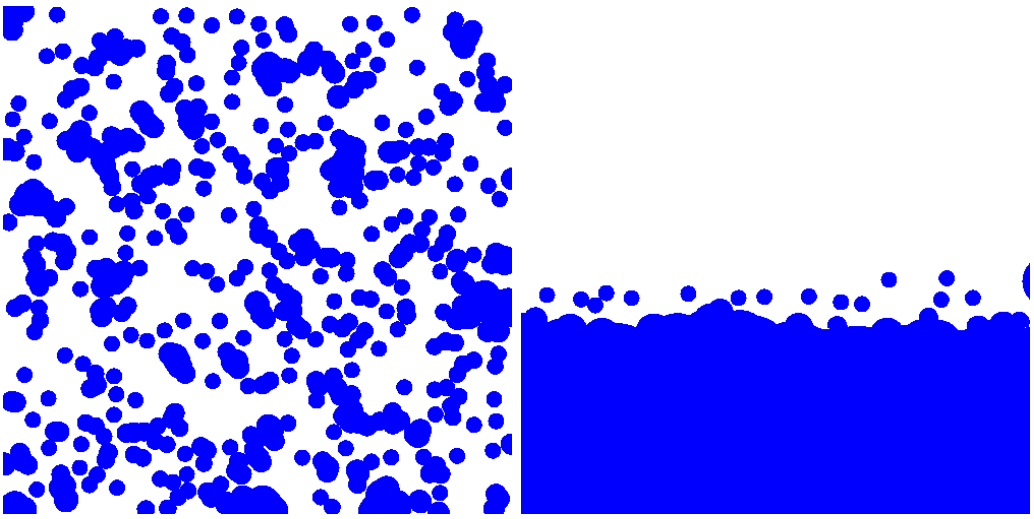
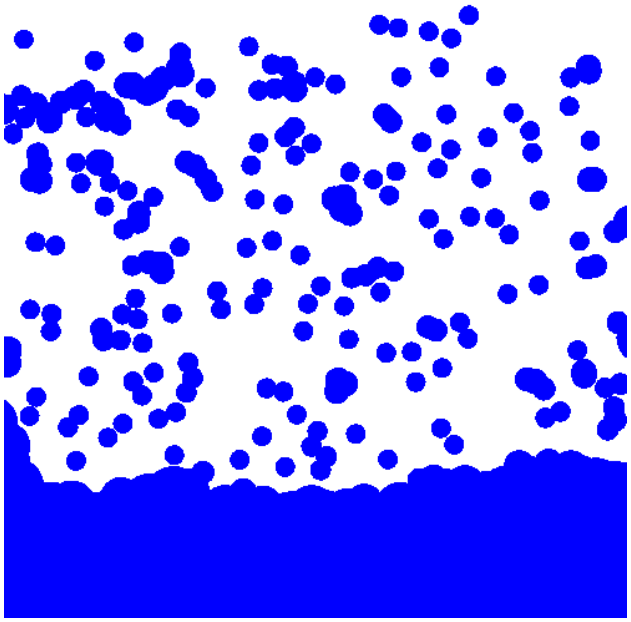


Figure 3: Example of Optimal Transport result for 1000 points generated at random with uniform densities. With the $O(n \log n)$ algorithm, this computation took about 30 seconds. With the $O(n^2)$ algorithm, it took about 120 seconds



(a) Frame 0

(b) Frame 39



(c) Frame 90

Figure 4: Examples from 100 frames of fluid simulation with iteration of $dt = 0.01$. Here we simulated 500 fluid particles with the following parameters: air is 60% of the scene, $\epsilon = 0.004$, $m_i = 200$. The $O(n \log n)$ algorithm was used for optimal transport. On average, one frame took 11 seconds to compute, with the computations slowing down to about 30-seconds per frame when there was a lot of fluid-boundary interaction.