2D Incompressible Euler Fluid Solver

Erik Dziekonski Bautista

1 Introduction

This project implements a free-surface 2D fluid solver using the incompressible Euler equations. The key components include:

- Voronoi diagram generation using Voronoi Parallel Linear Enumeration with Sutherland-Hodgman polygon clipping
- Extension to Power diagrams
- Optimization of Power diagram weights using LBFGS
- Implementation of the de Gallouet-Mérigot incompressible Euler scheme with prescribed fluid cell areas and intersected with a disk

2 Project Structure

fluid.cpp • Implements fluid-related computations, including the construction of fluid diagrams and the evaluation function for LBFGS optimization.

fluid.h • Declares functions for constructing fluid diagrams and evaluating the optimization function.

• Includes necessary headers for vector operations, classes, and the LBFGS library.

gallouet.cpp • Implements the time-stepping function for fluid dynamics using the Gallouet method, which includes optimization and vertex updates.

gallouet.h • Declares the function for performing a time step in the Gallouet method.

main.cpp • Contains the main function to initialize the simulation, perform the fluid dynamics steps, and handle input/output operations.

rendering.cpp • Implements functions for rendering and saving frames of the fluid simulation as images.

rendering.h • Declares the functions used for rendering and saving simulation frames.

voronoi.cpp • Implements Voronoi diagram-related functions, including edge clipping and diagram generation.

voronoi.h • Declares the functions for generating Voronoi diagrams and clipping edges.

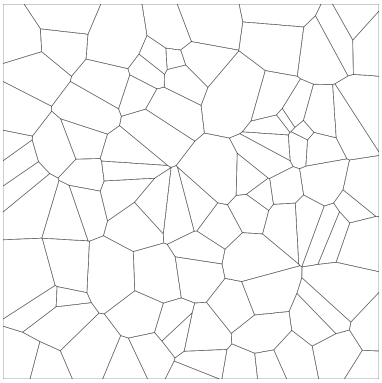
./utils/classes.cpp • Defines various utility functions and operations for vectors and polygons.

./utils/classes.h • Declares the Vector and Polygon classes, along with associated operations and utility functions.

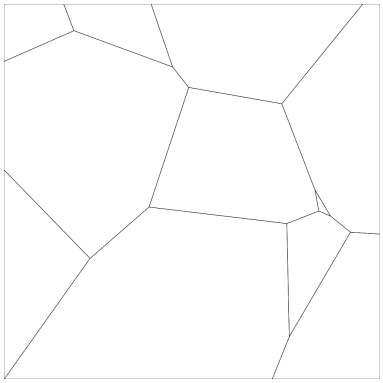
./utils/svg.h • Contains functions for saving static and animated SVG files representing the simulation's state.

3 Voronoi and Power Diagrams

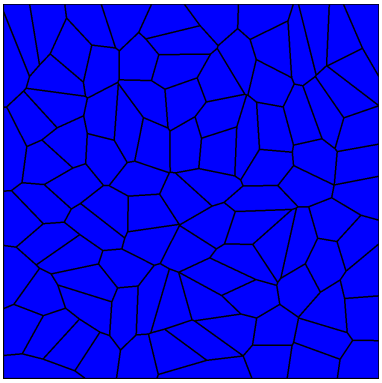
The Voronoi diagram is generated using the `generate_diagram` function, which implements the Voronoi Parallel Linear Enumeration algorithm and Sutherland-Hodgman polygon clipping. This is extended to Power diagrams in the same function by incorporating weights.



(a) Voronoi diagram



(b) Power diagram with random weights



(c) Optimized Power diagram

Figure 1: Comparison of Voronoi diagrams, $N = 100$

The weights of the Power diagram are optimized using the LBFGS algorithm to achieve cells of equal areas. The objective function and gradient computation from the lecture are implemented in the `eval_f` function.

4 Incompressible Euler Fluid Solver

The de Gallouet-Mérigot incompressible Euler scheme is implemented to simulate the fluid. The key steps are:

1. Update particle velocities and positions using spring forces and gravity in the `gal_step` function
2. Construct the fluid diagram using the `construct_fluid` function by clipping the Power diagram cells with disks
3. Optimize the weights to achieve the target cell areas using LBFGS in the `eval_f` function
4. Render the fluid surface by saving PNG frames using the `save_frame` function

The fluid simulation can be customized by adjusting the following parameters in the `gal_step` function:

- `mass`: Mass of each fluid particle
- `epsilon`: Epsilon value for the spring force
- `delta_time`: Time step for the simulation

By varying these parameters, different fluid behaviors can be achieved. Here are some example simulations with different parameters:

4.1 Example 1

For all the examples from here on, the number of frames has been set to 100.

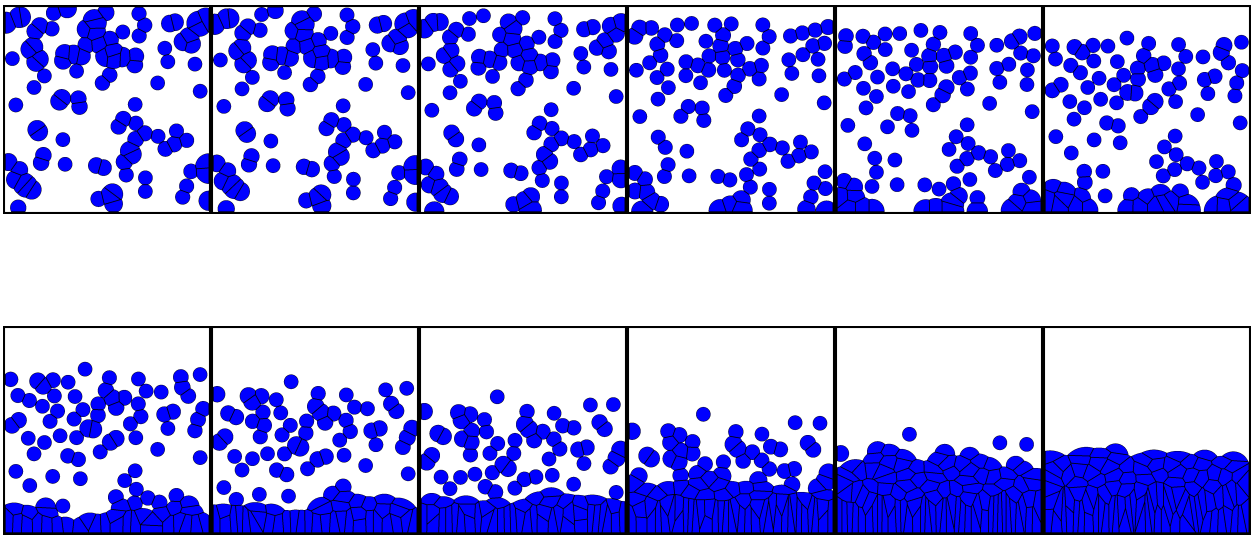


Figure 2: N=100, mass=200, $\epsilon = 0.004, \delta_t = 0.005$

With these parameters we had an average of 9.89825 seconds per frame.

4.2 Example 2

Now, let's start observing what happens when we start changing the parameters. Before that, let's set a base with $N = 20$ and the aforementioned parameters.

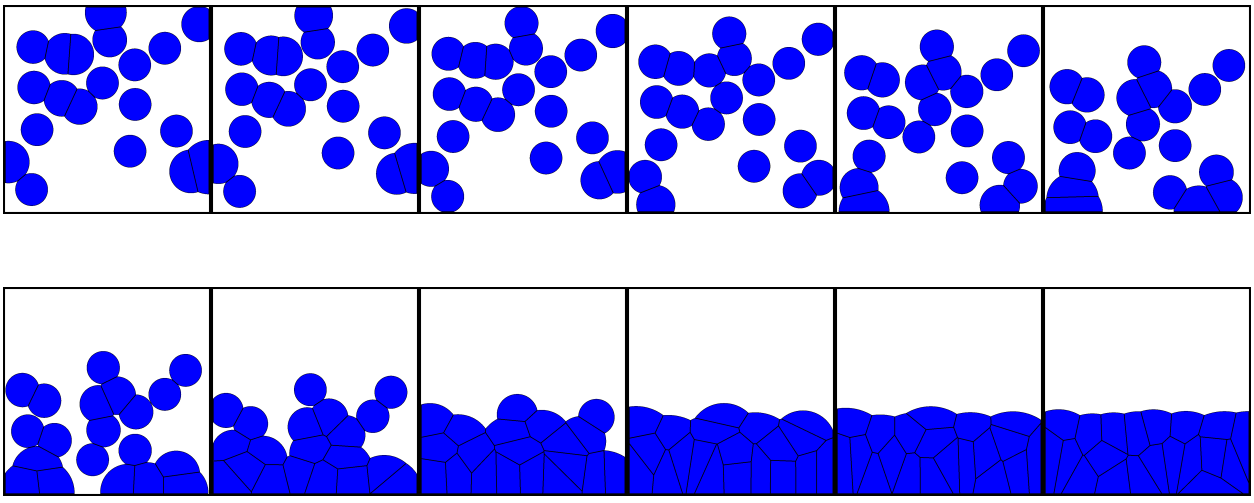


Figure 3: N=20, mass=200, $\epsilon = 0.004, \delta_t = 0.005$

Average computation time per frame: 1.203564 seconds

4.3 Example 3

Let's see what happens when we greatly reduce the mass to 50.

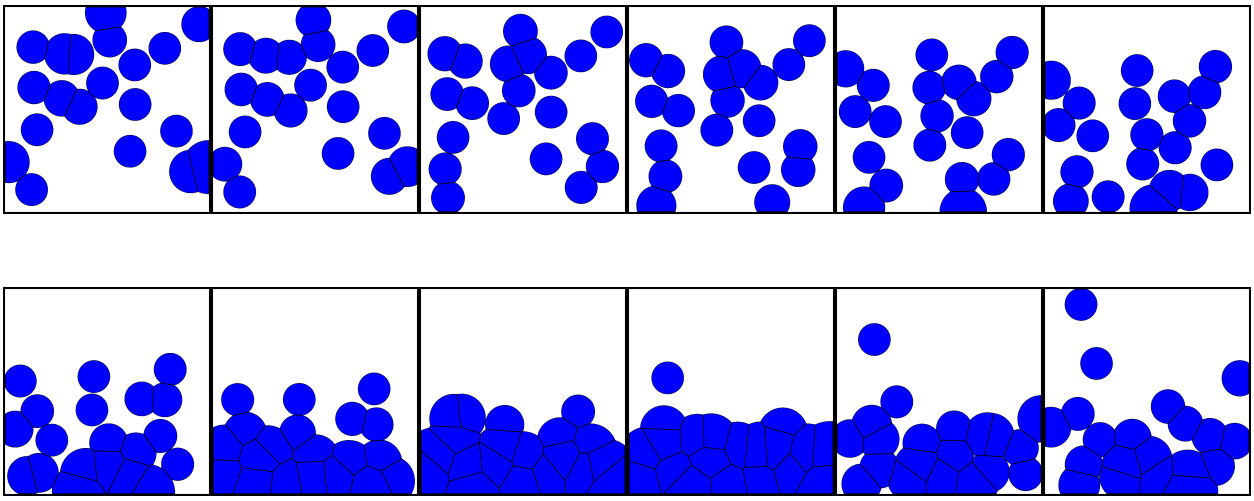


Figure 4: N=20, mass=50, $\epsilon = 0.004, \delta_t = 0.005$

Average computation time per frame: 2.32561 seconds.

4.4 Example 4

Finally, we observe what happens with a big spring force constant.

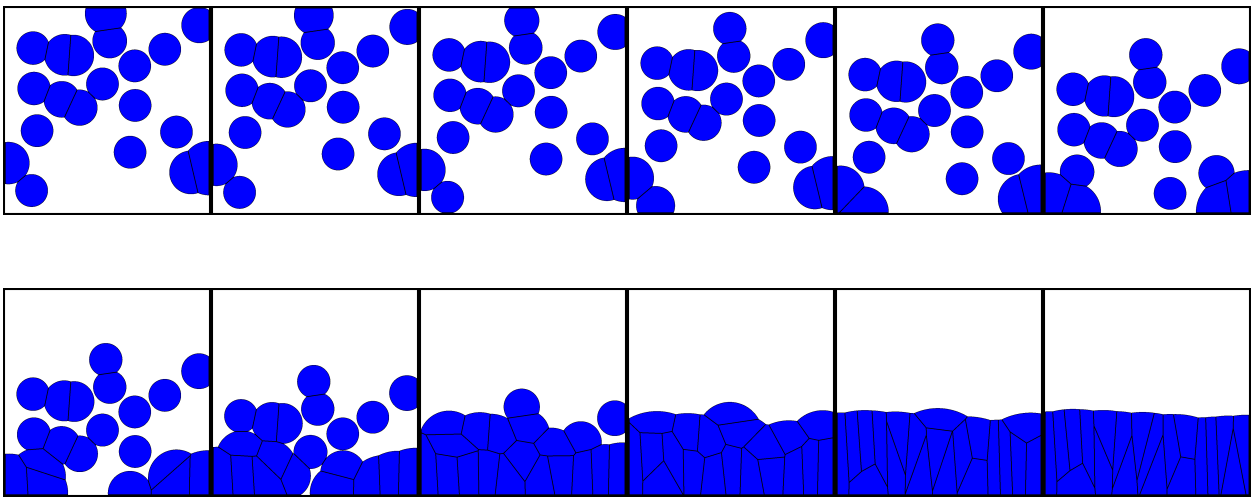
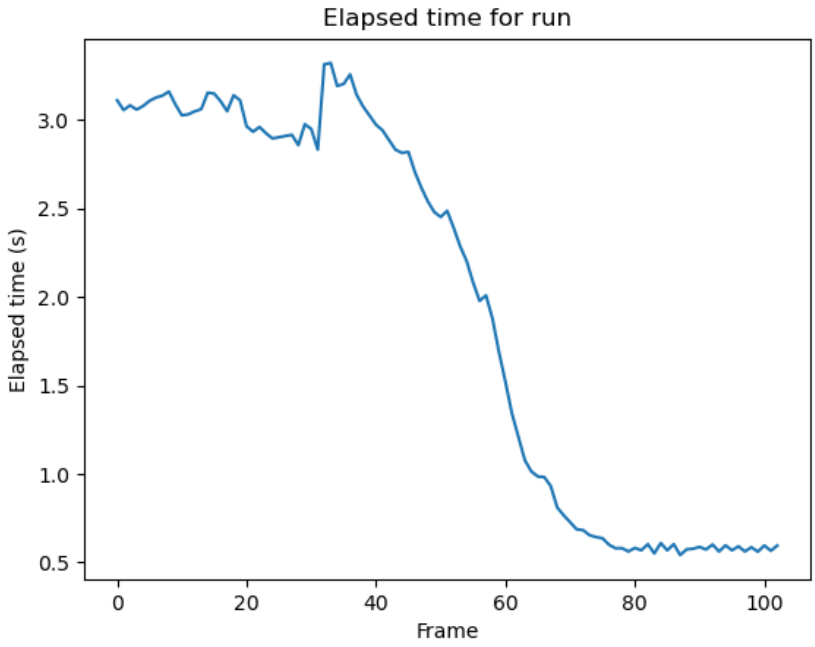


Figure 5: N=20, mass=200, $\epsilon = 0.04, \delta_t = 0.005$

Average computation time per frame: 1.95967 seconds
We can finally see the optimizer at work (run from example 4):



5 Conclusion

This project successfully implements a 2D incompressible Euler fluid solver using Voronoi and Power diagrams. The key features include:

- Efficient Voronoi and Power diagram generation with polygon clipping
- Optimization of diagram weights using LBFGS for equal cell areas
- Fluid simulation using the de Gallouet-Mérigot scheme with spring forces
- Parallel execution using OpenMP for improved performance

The fluid surface is rendered by saving PNG frames, which can be combined into an animated GIF to visualize the simulation. By adjusting parameters such as mass, epsilon, and time step, different fluid behaviors can be achieved, with varying computation times per frame.

Future work could explore further optimizations, additional fluid properties, and 3D extensions to enhance the fluid simulation capabilities.

6 References

Sources I looked at for inspiration while debugging:

- https://github.com/shayyn20/CSE306/blob/master/assignment%202/func/fluidSimulation.h
- https://github.com/jbhlevy/CSE306/blob/main/Project_2/main.cpp

- https://github.com/ekaterinaborisova/graphics_cse306
- https://www.geeksforgeeks.org/voronoi-diagram/
- https://github.com/JCash/voronoi
- https://stackoverflow.com/questions/20737987/extern-c-when-exactly-to-use/20738072#20738072
- https://www.geom.at/example12-voronoi-diagram/

Overall understanding:

- https://courses.cs.washington.edu/courses/cse326/00wi/projects/voronoi.html

Students that helped me:

- Milos Oundjian