Computer Graphics - Fluid Simulation

Anahi Tinel

June 2025

1 Introduction

This report presents an implementation of a 2D fluid simulator based on optimal transport theory, specifically following the Gallouët-Mérigot scheme. The approach uses Lagrangian particles and enforces incompressibility through semi-discrete optimal transport, avoiding the need to solve Poisson equations typical in traditional fluid solvers. The simulation represents fluid as a collection of particles whose motion is governed by spring forces derived from optimal transport maps.

2 Implementation Details

2.1 Core Classes

Polygon Class: Implements geometric operations for Voronoi cells including area calculation using the shoelace formula, centroid computation, and integral calculations for optimal transport energy evaluation.

VoronoiDiagram Class: Constructs power diagrams through iterative polygon clipping. The `clip_by_bisector` method implements the Sutherland-Hodgman clipping algorithm to construct each cell by intersecting half-spaces defined by perpendicular bisectors between particle pairs.

OptimalTransport Class: Currently implements the geometric construction of Voronoi cells. The optimization of weights (which would minimize the transport energy) is simplified in this implementation.

Fluid Class: Main simulation engine that manages particle positions, velocities, and temporal integration.

2.2 Key Algorithms

Cell Construction: Each Voronoi cell is constructed by starting with the unit square domain and successively clipping against bisector planes:

Listing 1: Voronoi Cell Construction

```
for (int j = 0; j < points.size(); j++) {
    if (i == j) continue;
    V = clip_by_bisector(V, points[i], points[j],
                         weights[i], weights[j]);
}
```

Time Integration: The simulation uses explicit Euler integration with spring forces and gravity:

Listing 2: Particle Update

```
Vector spring_force = (center_cell - particles[i]) / epsilon2;
Vector all_forces = m_i * g + spring_force;
velocities[i] = velocities[i] + dt / m_i * all_forces;
particles[i] = particles[i] + (dt * velocities[i]);
```

Boundary Handling: Simple reflection boundary conditions are implemented for domain boundaries with energy dissipation (coefficient 0.5).

2.3 Numerical Parameters

The simulation uses several key parameters:

- $\epsilon^2 = 0.004^2$: Spring constant controlling incompressibility enforcement
- $dt = 0.005$: Time step size
- $m_i = 200$: Particle mass
- $g = -9.81$: Gravitational acceleration
- Initial fluid volume fraction: 60% of domain

3 Simulation Workflow

The main simulation loop follows these steps:

Algorithm 1 Fluid Simulation Time Step

```
1: Update particle positions in Voronoi diagram
2: Compute Voronoi cells for all particles
3: For each particle $i$:
4:     Compute cell centroid $c_i$
5:     Calculate spring force: $F_i = (c_i - x_i)/\epsilon^2$
6:     Apply gravity: $F_i += m_i\mathbf{g}$
7:     Update velocity: $v_i += dt \cdot F_i/m_i$
8:     Update position: $x_i += dt \cdot v_i$
9:     Apply boundary conditions
10: Save visualization frame
```

4 Visualization and Results

The simulation generates PNG images at each time step using a custom rasterization routine. Fluid particles are colored blue, with cell boundaries shown in black. The `save_frame` function implements point-in-polygon testing to determine pixel colors within each Voronoi cell.

Initial conditions place 1000 particles in a circular distribution centered at $(0.5, 0.75)$ with radius 0.15, simulating a fluid drop. The simulation runs for 100 time steps, generating frames that can be assembled into an animation showing fluid motion under gravity.

5 Implementation Challenges and Attempted Optimization

While I was able to implement most of the core framework, I found myself unable to solve the issue with importing the lbfgs library. The Gallouët-Mérigot scheme requires solving a constrained optimization problem to find the optimal weights that minimize the transport energy while maintaining the incompressibility constraint, but this is impossible without said library.

5.1 Attempted L-BFGS Implementation

I attempted to (blindly) implement the complete optimization using the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm through the `liblbfgs` library. The implementation approach included:

- **Objective Function:** Formulated the dual problem with Lagrange multipliers to enforce volume constraints
- **Gradient Computation:** Implemented analytical gradients based on the area differences between target and actual cell volumes
- **Air Particle Handling:** Added an additional weight variable to represent the air phase in free surface simulations
- **Integration Framework:** Structured the code to call the optimization routine at each time step

The attempted implementation included proper setup of the optimization parameters, constraint handling for both fluid and air volumes, and integration with the existing Voronoi diagram computation. The mathematical formulation correctly identified that the gradient with respect to each weight should be the difference between the target volume fraction and the actual cell area.

5.2 Sources

Some of my code is from the td sessions as I wanted to have a good starting point, and then when it came to learning how to use the lbfgs library which I could not properly set up on my macOS, I used the two following githubs:

- https://github.com/chokkan/liblbfgs/tree/master (attempt at installing the lbfgs library)
- https://github.com/joshuapjacob/computer-graphics/tree/main?tab=readme-ov-file (code from a student from a previous year that managed to implement the optimization. The code that I am submitting is mine, but I did use his code to assist me in understanding how to use the library tools, since I could not test them myself, notably understand the lbfgs types)

5.3 Library Integration Issues

Despite giving extensive effort to integrate the `liblbfgs` library, compilation and linking issues prevented me from testing the optimization routine. I attempted several approaches like building the library from source using the official repository and using different compilation configurations and linking strategies, as well as looking for other optimization libraries as potential substitutes.

These technical difficulties ultimately necessitated falling back to the simplified implementation without weight optimization and a rather interesting (?) output. However, I left in the comments the attempted code structure which I believe demonstrates understanding of the underlying mathematical principles and provides a foundation for the implementation once I manage to resolve the library integration issues.

5.4 Impact on Results

Without the optimization component, the current implementation cannot enforce the incompressibility constraint as strictly as intended in the original Gallouët-Mérigot scheme. This limitation affects the physical accuracy of the simulation, particularly for longer time periods or more complex flow scenarios.

6 Conclusion

This implementation demonstrates the core geometric concepts of optimal transport-based fluid simulation using Voronoi diagrams. While the full optimization could not be completed due to technical challenges, the simulation successfully shows how geometric constraints can enforce incompressibility without solving large linear systems. The spring force model produces stable and visually plausible fluid motion, suggesting that even simplified optimal transport approaches can be valuable for computer graphics applications. Future work should focus on resolving the optimization implementation to achieve the full potential of the Gallouët-Mérigot scheme.