# CSE306 - Fluid Solver Project

## 1 Introduction

### 1.1 Project Overview

This project implements a free-surface 2D fluid solver using incompressible Euler's equations. My project implements all of the mandatory sections of the assignment (Vonoroï Diagrams, Power Diagram, Optimization with LBFGS, de Gallouet-Mérigot incompressible Euler scheme, spring force from each fluid particle to their Laguerre's cell centroid), as well as the ungraded labs (diffusion lab and Tutte embedding).

All of the project's code can be found in the present GitHub repository under two branches. The `main` branch is dedicated to Lab 6 and Lab 7 while the `FluidDynamics` branch is specific to the fluid simulation. Files have been split in headers and sources for organizational purposes, and I added a Makefile to simplify compilation. This report shows my implementation for each part of the project. As Professor Bonneel said it was not necessary to quoting and explain every algorithm used, this report does not explicitly cover formulas referenced in the textbook. I nonetheless tried to show my understanding as best as I could in the code directly by making my project structures and functions as clear as possible.

To ensure clarity and correctness, the code is also optimized as much as possible (without adding too many level of abstraction which would be counter-productive). To avoid and identify bugs easily, `const` statements were added wherever possible. To achieve the highest precision, floating-point operations were prioritized. To optimize memory, the most efficient types for each variable were used. Nonetheless, some typical concerns such as privacy of attributes have been disregarded as they would have overcomplicated the code with a multitude of getters and setter functions.

### 1.2 Execution Parameters

This fluid solver runs on a MacBook Pro M1 with C++11. Execution parameters can be set in `main.cpp`. For the fluid simulation, I run the simulation with 500 fluid particles and use $\varepsilon = 0.004$, $dt = 0.002$ and a mass of 200 for each particle.

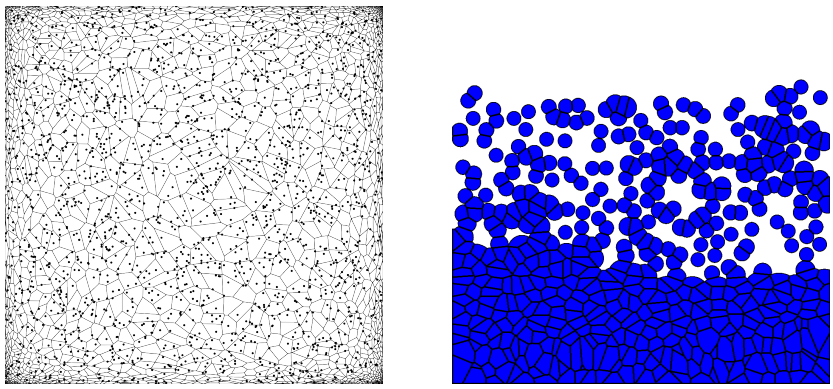

Figure 1: Presentation images for Lab 7 (left) and Lab 8 (right)

## 2 Clipping and Voronoï Diagrams

The first part of the project revolves around implementing the Voronoï Parallel Linear Enumeration algorithm from Section 4.3.3 of the lecture notes. For that, I implement the Sutherland-Hodgman polygon clipping algorithm from Section 4.2. After implementing clipping, a `Polygon` class, and a `VoronoiDiagram` class, I get the following result:

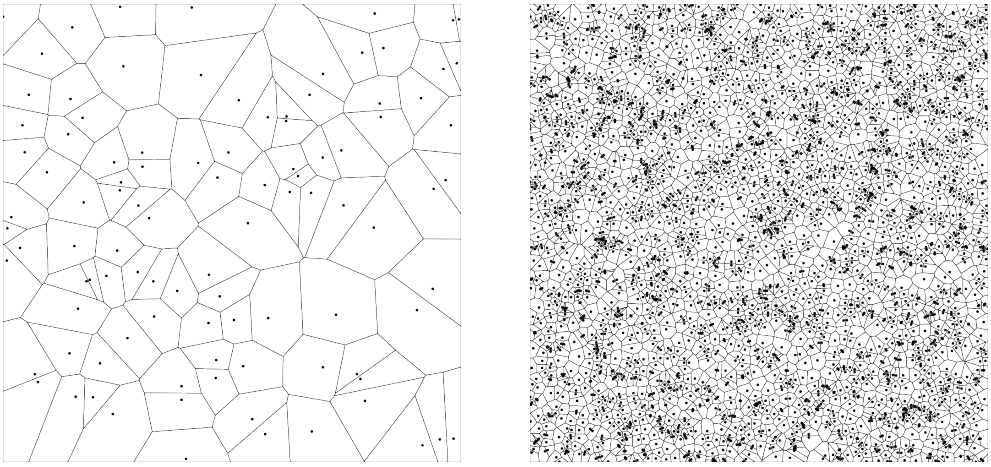

Figure 2: Voronoi Diagrams for $N = 100$ (left) $N = 3000$ (right) - *0.002s and 0.072s rendering.*

## 3 Power diagram and Weights Optimization with LBFGS

In this second part, we convert the Voronoi diagram to a Power diagram that generalizes this by assigning a weight to each point and implementing semi-discrete optimal transport with L-BFGS from Sections 4.4.3 and 4.4.4 of the textbook and using the `evaluate` function from the libLBFGS library. With uniform weights, we get the following result.

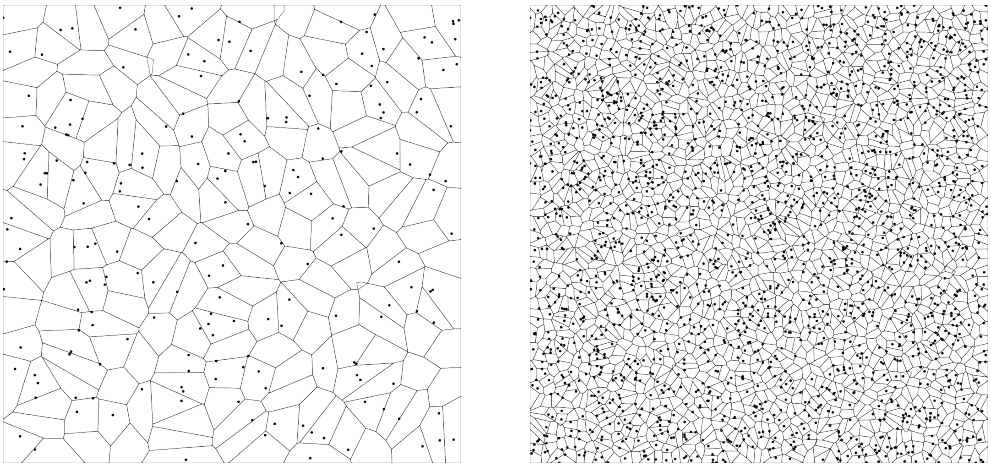

Figure 3: Uniform weights with $N = 200$ (left) and $N = 2000$ (right) - *0.041s and 0.730s rendering.*

We can also test the code with different set of weights. Here are two examples with a gaussian distribution (with a standard deviation $\sigma = 0.18$ tailored for best visual results) and an exponential distribution from the bottom left corner (with $\lambda = 4$ tailored for best visual results).

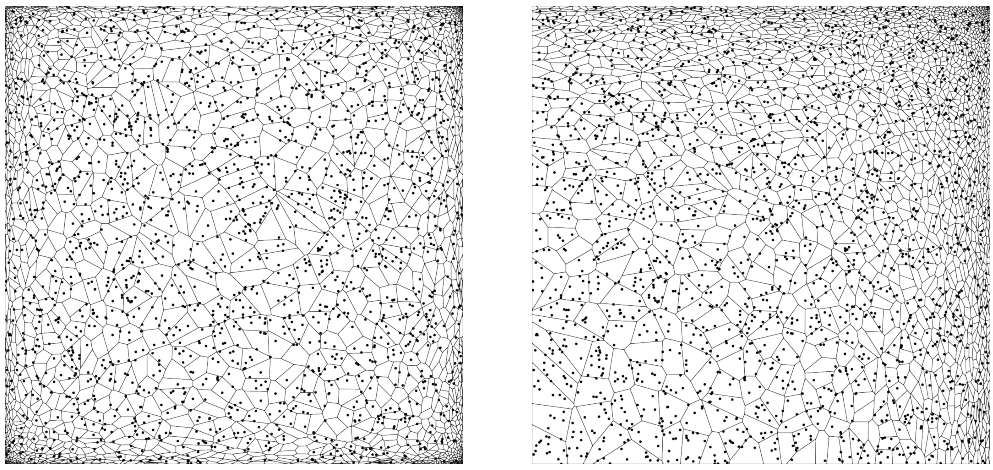

Figure 4: Uniform weights with $N = 200$ (left) and $N = 2000$ (right) - *0.041s and 0.730s rendering.*

# 4 Fluid Dynamics

After implementing these diagrams which we can parametrize with any kind of weights, we can move on to modeling fluid dynamics. However, when animated, this GIF presents a major optimization error. The cells compress to the bottom left as can be seen on that picture below. As implementing Fluid dynamics makes a big change in terms of code base, I created another branch `FluidDynamics` specific to Lab 8. This allowed me to save my work and is useful for archive purposes.

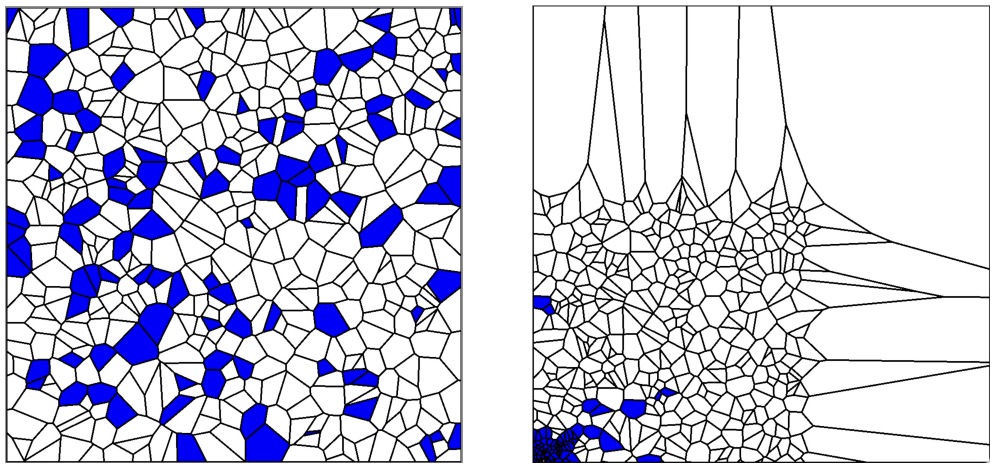

Figure 5: Fluid simulation (compression to lower left corner bug) over 100 frames - *101.6s rendering*

After debugging, this reveals to be a problem with LBFGS. Indeed, it returns the error code -999, indicating that the current search direction increases the objective function value. To solve that issue, I therefore used Gemini to change 13 lines of codes (cf. Acknowledgments Section). After this fix, I get the following image. Both animations can be found in my repository under `animation_compression_bug.gif` and `animation_.gif`.

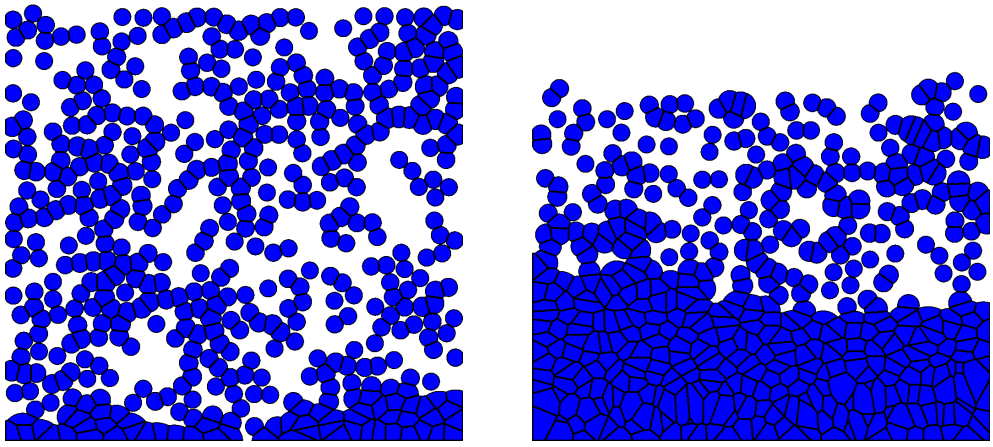

Figure 6: Working fluid simulation with over 250 frames - *205.8s rendering*

# 5 Acknowledgments

For this project, labs were typically started during the live coding sessions and finished at home using the course textbook and lecture slides. My implementation follows Prof. Bonneel's examples and each key function includes references to the textbook. Therefore, some functions and the overall structure of my code may nonetheless resemble what was written during tutorial sessions. I would also like to acknowledge the help of Andreea Patarlageanu for her explanations on Optimization with LBFGS. Note however that I did not take any of her code.

Furthermore, to solve an error code that I had with LBFGS (error code -999, indicating that the current search direction increases the objective function value) in the fluid simulation (only the last commit on the FluidDynamics branch), I also used Google Gemini. It indicated me that my centroid function was handling degenerate polygons badly, and told me about an issue with `memcpy`. I therefore integrated those changes in a separate commit, the last one on the `FluidDynamics` branch. Here, Gemini only contributed to 13 lines of code, a minor impact.

This use of Gemini was truly as a last resort after trying to debug for multiple hours. I understand that this can take points off Lab 8 but believe that using Gemini wouldn't discredit the rest of the work I have done. I have included both animations for the ultimate and penultimate commit, for you to see my result before and after those changes.

# 6 Miscellaneous

## 6.1 Lab 5 : Diffusion

For lab 5, I chose the generative image diffusion model lab, based on Section 3.2 from the textbook. This lab is included in my repository as `diffusion.ipynb`. Despite it not being graded, here is a brief explanation of how it is implemented.

In this lab, I implement a generative models for images that can produce images from noise using an auto-encoder. To do so, I first train a model on a set of images to which I add noise Gaussian noise iteratively. This allows the model to learn the conditional noise distributions needed to reverse

the diffusion process. Then, the goal of our trained model is to start with an image containing pure noise and apply the learned denoiser iteratively to clean the image and reveal a nice, clean, photograph. In practice, for this lab, I use equations 3.1, 3.2, and 3.3 of the textbook with the `unet_faces` dataset and its associated model to implement the denoiser.

### 6.2 Lab 9 : Tutte embedding

The other ungraded lab is an implementation of the Tutte Embedding as described in Section 4.6.1 of the lecture notes. In practice, I implement Algorithm 4 of this section, step by step. This algorithm takes a triangle mesh which can be reduced to a disk (meaning there are no holes inside it). Then, it maps boundary to a circle where the distance along the circle between successive vertices matches the length of the corresponding mesh edge. Finally, each non-boundary vertex is repeatedly updated to the barycenter of its neighbors to create a nice embedding. In this lab, I also implemented the optional harmonic mapping by replacing the graph Laplacian by the cotangent Laplacian. Intuitively, this means that instead of treating every neighbor equally, I weight each edge by the cotangent of its opposing angles. This improves the representation of the mesh's geometry by reducing distortion. I hope that this could slightly compensate for the imperfections in the main fluid solver project.

N.B: For this lab, Gemini helped me implement line 1 (out of 14) and the optional harmonic mapping of Algorithm 4. As this lab is ungraded, I hope that this does not harm my project grade.

### 6.3 Opinion of the course

I am very happy to discover Computer Graphics in such a fun way, especially since I love working on big projects. As a video-game player, I had always seen the terms ray tracing but never really thought of what it was. Also, when I talked to a student of the course that did an internship in Computer Graphics, she said that in her software, ray tracing was just a slider button that she would activate; no one ever saw what happened underneath the hood. Doing a dive into the inner working of ray tracing was therefore very interesting. I have similar feelings regarding the fluid solver project. We always say that modeling water and fluids on computers is complex and resource intensive, but we never get to see why. Discovering the numerous formulas and equations behind why that is the case is therefore very interesting.

In terms of course format, I also think this course could have also benefited from more guidance regarding the labs to focus more on the algorithms and less on the C++ implementation. Indeed, we often find ourselves worrying much more about the C++ implementation that the Computer Graphics algorithms. As this is not a C++ course, I think that it could be better to be given each week the classes and functions to implement with their signatures. This could allow us to focus more on the core of the formulas and algorithms.