# CSE306 Assignment2

Alexis Roger
June 2020

## 1. Introduction

This project is aimed at implementing Fluid Dynamics in C++ with minimal libraries. This implementation was done in multiple steps which we will detail one at the time, with them corresponding to the different lab sessions.

For reference, all runtimes given here, unless indicated otherwise, where achieved running the code in parallel on a laptop with an Intel i7-4700MQ CPU (4 cores at 2.4GHz).

The GitHub repository with the code of the project may be found below. It is in the 'master' branch under the folder 'assignment 2'.
https://github.com/Alexis-BX/CSE306-Raytracer

## 2. General code structure

The code is organized in a linear fashion. To begin, all external imports and constants are in 'master.cpp'. This file is then imported by 'vector.cpp' which defines the Vector class and all of its overloaded operators. These are the same files as those used previously for the raytracer. We then have 'voronoi.cpp' where the 'Polygon' class is defined as well as most of the functions seen later. Finally we have 'testVoronoi.cpp' which is used as a main file and contains the 'main' function (so this is the file to compile and run for testing).

## 3. Image processing

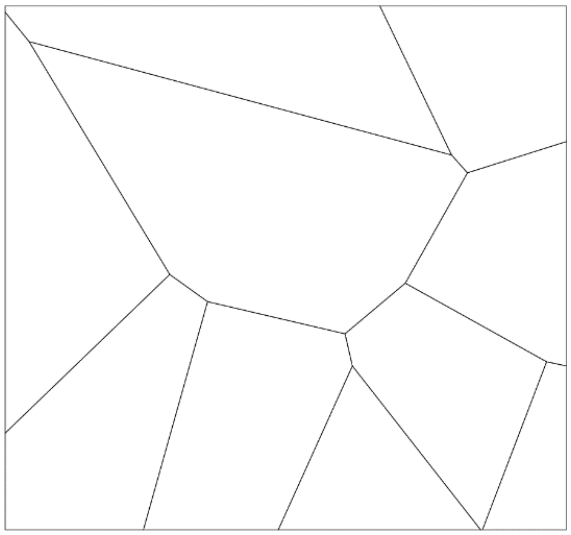
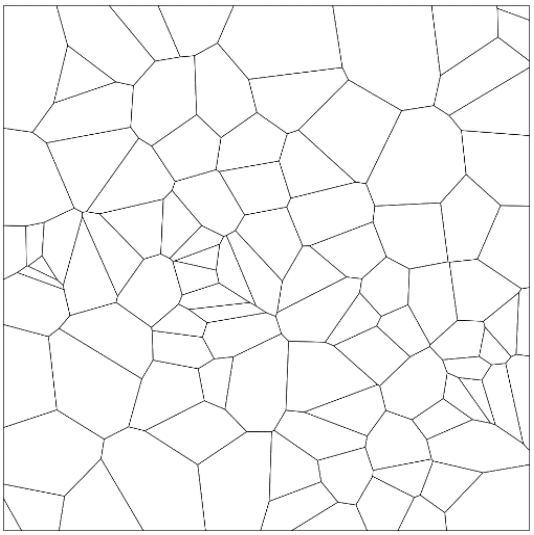
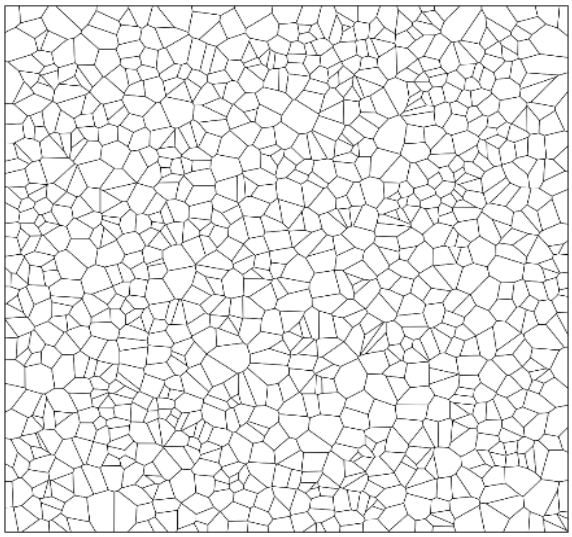
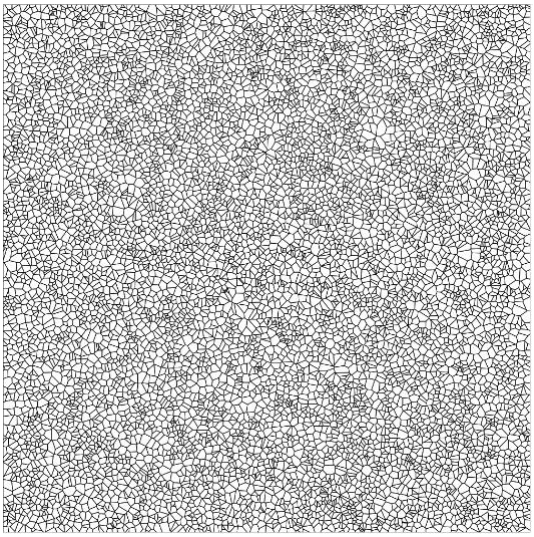
To begin we did some image processing. Both the sliced optimal transport approach for color matching and the image retargeting algorithm for shrinking images where implemented in the file 'image.cpp'. The optimal transport color matching is a technique which takes a base image and gets its color palette closer to a reference image. This allows us to change the style of an image very easily to match a specific theme.

On the other hand the retargeting algorithm crops a line of the least significant pixels from the top of the image to the bottom. This allows us to crop images with pixel precision all the while not losing important features or elements. This will tend to remove pixels belonging to the background and can easily favor a unique side, leading to part of the image not looking to scale or for the central piece to be off centered.

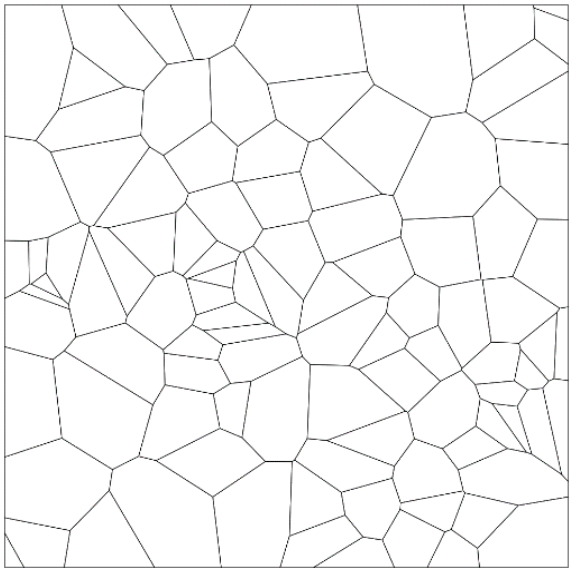
Running these functions is near instantaneous on small images but they scale very badly with the image size.

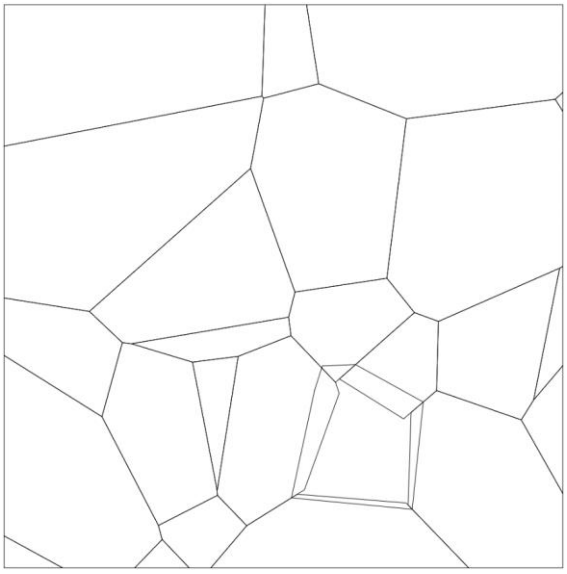
## 4. Clipping and Voronoï Diagrams

We then implemented the Voronoï Parallel Linear Enumeration algorithm in 2D with the naïve method, leading to a complexity of O(N^2), with N the number of points. Top demonstrate this effect we have the following images. From left to right we have 10, 100, 1000 and 10000 points. Computing 100 points took a second, while less was instantaneous and 10000 took 15 seconds.

On top of these we then added the power diagram functionality and with a 100 points we would get the following results. In order to compare these images with random points the same random seed was set with 'srand(0);' at the start of the program. The same amount of random calls were also done to avoid offsetting it. Both have 100 points and generated immediately. On the left is the image where all the points have identical weight and on the right is the power diagram.





## 5. Fluid dynamics

We then used the libLBFGS library to implement an ‘evaluate’ function which we were to use later in the fluid dynamics simulations.

However, I was unsuccessful in implementing the semi-discrete optimal transport fluid simulator.

## 6. Extra

The Tutte embedding was implemented in the ‘testVoronoi.cpp’ file as it could be done independently.