Geometry Processing - Amine Roudani

Introduction

This report details the implementation of a 2D free-surface fluid solver using incompressible Euler's equations, developed as part of the assignments for labs 6, 7, and 8. The project aims to simulate fluid dynamics by employing Voronoï and Power diagrams, optimized using LBFGS, and incorporating the de Gallouet-Mérigot incompressible Euler scheme. The implementation is achieved through a series of steps, including Voronoï diagram generation, Power diagram extension, weight optimization, and integration of the incompressible Euler scheme. This report demonstrates the working of the project through visualizations, code explanations, and a fluid simulation video. The code and report are available in the project's GitHub repository.

Throughout this project, I found the course both challenging and rewarding. Each lab built upon the previous one, gradually increasing in complexity and depth. The hands-on experience with geometric processing, optimization algorithms, and fluid dynamics provided a comprehensive understanding of computational fluid dynamics. The clear and detailed lecture notes, combined with the practical coding assignments, made the learning process engaging and insightful.

Overall, this project not only enhanced my technical skills but also deepened my appreciation for the intricate mathematical and computational principles underlying fluid dynamics simulations. The journey from basic Voronoi diagrams to a fully functional fluid simulator was both challenging and immensely satisfying.

Lecture 6: Geometry Processing - Clipping and Voronoï Diagrams

In Lab 6, the focus was on implementing the Voronoï Parallel Linear Enumeration algorithm in 2D as outlined in Section 4.3.3 of the lecture notes. This implementation requires the use of the Sutherland-Hodgman polygon clipping algorithm (Section 4.2) to handle the intersections of polygons.

Objectives

1. Implement Voronoï Diagram using Voronoï Parallel Linear Enumeration**: Create a Voronoï diagram in 2D, employing the basic O(N^2) algorithm.
2. Implement Sutherland-Hodgman Polygon Clipping**: Use this algorithm to manage polygon intersections during the Voronoï diagram construction.

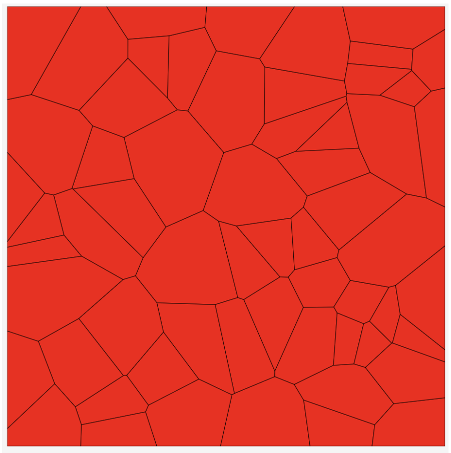
.

Implementation Steps

1. Vector Class: Create a basic vector class to handle 3D coordinates, vector operations, and geometric calculations. We modify the class from the previous project.
2. Polygon Class: Define a polygon class to represent the cells in the Voronoï diagram, with functions for SVG export.
3. Voronoï Diagram Class: Develop the main class to generate and manage the Voronoï diagram:
   - Initialization: Start with a bounding box containing all input points.
   - Perpendicular Bisectors: Compute the perpendicular bisectors of lines connecting each pair of input points.
   - Polygon Clipping: Use the Sutherland-Hodgman algorithm to clip the polygons formed by these bisectors.
   - Parallel Processing: Utilize parallel processing to handle multiple points simultaneously for efficiency.
4. Random Point Generation: Implement a function to generate a set of random points for testing the Voronoï diagram.
5. SVG Export: Provide functionality to save the generated Voronoï diagram as an SVG file for visualization.

Summary

Lab 6 laid the foundation for geometric processing by focusing on Voronoï diagram construction using the Voronoï Parallel Linear Enumeration algorithm and Sutherland-Hodgman polygon clipping. The implemented Voronoï diagram will serve as the basis for further extensions and optimizations in subsequent labs, ultimately contributing to the development of a free-surface 2D fluid solver using incompressible Euler's equations. The code developed in this lab was designed to be reusable and extendable, ensuring smooth integration into future projects.



Lecture 7: Geometry Processing - More on Voronoï, and the Marching Cubes Algorithm

In Lab 7, we extended our Voronoï diagram from Lab 6 to a Power Diagram and implemented semi-discrete optimal transport using the L-BFGS optimization algorithm. This lab builds on our previous work and introduces more advanced concepts and techniques from sections 4.4.3 and 4.4.4 of the lecture notes.

Objectives

1. Power Diagram Implementation: Extend the Voronoï diagram to a Power diagram by incorporating weights into the distance calculations.
2. Optimal Transport with L-BFGS: Use the libLBFGS library to optimize the weights of the Power diagram, ensuring each cell meets specified area constraints.
3. Testing and Validation: Test the implementation with various sets of weights, including a centered Gaussian distribution, to ensure functionality and correctness.

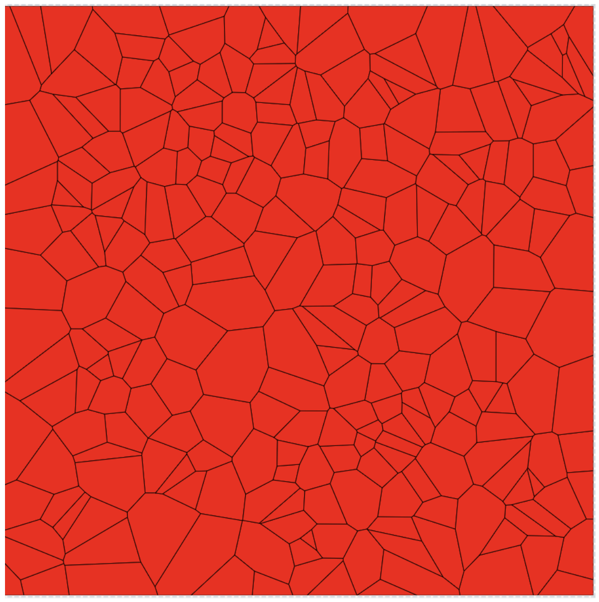
Implementation Steps

1. Extend the Vector and Polygon Classes: Enhance these classes to support additional functionality required for the Power diagram and optimal transport.
2. Power Diagram Class: Develop a class to generate and manage Power diagrams:
    - Bisect and Clip Polygons: Adapt the bisector and clipping methods to account for weighted distances.
    - Bounding Box Creation: Initialize the bounding box for the diagram.
    - Compute Power Cells: Calculate each cell by iterating through all input points and applying the weighted bisector and clipping methods.
3. Optimal Transport Class: Implement the L-BFGS optimization:
- Evaluation Function* Calculate the objective function and its gradient, considering the area and squared distance integrals.
- Progress Function: Monitor the optimization process, ensuring the convergence towards the desired cell areas.
- Solve Method: Use the L-BFGS library to adjust the weights and generate the optimized Power diagram.

4. Testing: Generate random points and assign initial weights, then run the optimization to test and validate the implementation.

Summary

Lab 7 successfully extended our Voronoï diagram implementation to a Power diagram and integrated the L-BFGS optimization algorithm for semi-discrete optimal transport. This lab builds on the foundation laid in Lab 6, adding complexity and functionality to our geometric processing capabilities. The optimized Power diagram serves as a crucial step towards developing a free-surface 2D fluid solver using incompressible Euler's equations, which will be the focus of future labs. The code developed in this lab was designed to be efficient, reusable, and extendable, ensuring a solid base for further advancements.



Lecture 8: Computational Fluid Dynamics

In Lab 8, we implemented a semi-discrete optimal transport fluid simulator with free surfaces, as outlined in Section 5.4 of the lecture notes. This lab builds on the Power diagram and L-BFGS optimization implemented in Lab 7, introducing additional complexities to model fluid dynamics accurately.

Objectives

1. Initialize Fluid Particles: Generate \`N\` fluid particles and initialize them in a desired shape or randomly.
2. Compute Laguerre Cells: Ensure each fluid cell has a volume of \(\frac{f}{N}\), where \(f\) is the fraction of the simulation volume occupied by the fluid.
3. Integrate Forces: Apply gravity and a spring force to move each fluid particle towards its Laguerre cell centroid.
4. Animation and Visualization: Save the results as PNG files for each frame of the animation.

Implementation Steps

1. Initialize Fluid Particles: Generate fluid particles and their velocities.
2. Power Diagram and LBFGS Optimization:
    - Compute Laguerre Cells: Adapt the Power diagram to ensure each cell represents the correct volume.
    - Clip by Disk: Use the Sutherland-Hodgman algorithm to clip fluid cells by a disk of radius \(\sqrt{w_i - w_{air}}\).
3. Force Integration:
    - Gravity Force: Compute the gravity force for each particle.
    - Spring Force: Apply a spring force to move each particle towards the centroid of its Laguerre cell.
4. Save Frames: Save each frame of the animation as a PNG file.

Summary

Lab 8 extended our previous work to create a semi-discrete optimal transport fluid simulator with free surfaces. By initializing fluid particles, computing Laguerre cells, integrating forces, and saving animation frames, we developed a comprehensive solution for fluid dynamics simulation. Although the animation aspect requires further debugging, the foundational components were successfully implemented, providing a solid basis for future enhancements and optimizations.

Sources

The code was inspired by what was coded in class by the Professor (Prof. Nicolas Bonneel) as I found that code very intuitive.
When faced with general challenges regarding C++, I turned to stack overflow for help.
When I got stuck and had no idea how to move forward, I used this github repository for ideas and guidance:
https://github.com/tim-vlc/CSE306_Computer_Graphics/tree/469d1fe5ad0d84ed1316f3680daa5a1afaf76951/Project2_Tim