Dissertation (draft)

# Abstract

(General summary- explain purpose of application, potential real-world applications, limitations of existing simulation models)

# Contents

(list the sections that are below)

# Rationale

Discuss existing methods of geographical simulation with a focus on rivers

Difference between particle and node-based systems

The Lattice-Boltzman algorithm

Potential advantages/disadvantages of this approach compared to traditional methods

Landscape generation technology & real-time mesh deformation, 3D representation

# Literature Review

Many studies into fluid simulation exist, with varying scopes, scales, and approaches. Multiple algorithms exist to simulate fluid movement, with varying usefulness in tackling this project. One such example are the Euler Fluid Equations <https://levelup.gitconnected.com/create-your-own-finite-volume-fluid-simulation-with-python-8f9eab0b8305>, which emphasize having an incompressible fluid of constant density within a closed system. This is calculated as a flow velocity vector for points on a grid, considering all body acceleration and acting forces, such as gravity. These would prove perfect for my project, being both easy to simulate and considering the surrounding environment, as well as utilizing a similar node-based system. However, it is possible to hit a point of singularity, a possibility that is very likely to occur in a large-scale simulation due to the number of calculations being performed. <https://cmsa.fas.harvard.edu/euler-workshop/> .

Another example are Navier-Stokes equations <https://en.wikipedia.org/wiki/Navier%E2%80%93Stokes_equations> . These serve as an alternative to Euler Fluids, focussing on the conservation of mass and momentum at given points within a fluid, in a similar node-and-vector system to Euler fluids. These can also account for temperature and viscosity of a fluid, as well as both compressible and incompressible flow, allowing more complex simulation of fluids in multiple states. A common problem with Navier-Stokes equations is the fact they are infinitely differentiable (citation)- for any given point in the fluid’s domain, the vector velocity of the point can be infinitely refined- no answer will be 100% accurate. While this doesn’t cause a huge issue in terms of large-scale simulation (there will always need to be refinements, as true simulation on a particle-level would take an infinite or near-infinite amount of time [https://en.wikipedia.org/wiki/Quantum\_computing](https://en.wikipedia.org/wiki/Quantum_computing#:~:text=Quantum%20computing%20is%20a%20type,are%20known%20as%20quantum%20computers)) a cut-off point would have to be found in order to accept a solution with an acceptable level of accuracy. The Cauchy stress tenor <https://en.wikipedia.org/wiki/Cauchy_stress_tensor> of a unit space can be calculated to assist in the solving of these equations, but accuracy is still limited.

The Lattice-Boltzman algorithms <https://en.wikipedia.org/wiki/Lattice_Boltzmann_method> , avoid solving these equations by simulating a fluid as a lattice, with tension and relaxation points. The algorithm is very adjustable, mimicking both vapours and fluids on small scales(citation). However, complex boundaries significantly complicate the algorithm, and it operates better for small-scale fluid simulation, such as deformation of a single droplet (citation).

Initially, I aimed to implement an algorithm to solve the Navier-Stokes equations to an acceptable degree of accuracy, as other fluid simulations have used in the past (citation). However, when paired with the larger-scale of the environments I wished to simulate (estimating one km^2 area), preliminary testing revealed that the performance would be unacceptable. The solutions would be either far too inaccurate or take such a significant amount of time that the program would be impractical, potentially running for hours to simulate a single year of fluid movement. While acceptable for a flooding-avoidance program or smaller-scale animation render, this would be impractical for my program’s purposes.

A study by Nicholas McDonald into the movement of water through a simulated particle on a grid, called the “Hydraulic Erosion Algorithm”, proved a better basis for my study(citation). Using a previous study of sedimentation and mass transfer, a simple demonstration of water moving as a particle to form a river was developed, allowing streams to form in the terrain (<https://nickmcd.me/2020/04/10/simple-particle-based-hydraulic-erosion>.) Although more rooted in classical mechanics than traditional fluid simulation, I felt it could provide an accurate representation of water moving through a landscape, while keeping computational time reasonable. The pooling methods used by McDonald, in which a pool consisted of particles repeatedly striking their surroundings, seemed unrealistic, and I opted to develop my own, separate pooling method.

As well as this, I chose to develop a version of the Hydraulic Erosion Algorithm with an additional soil map for the landscape. Instead of treating it as a deformed plane, I would information about the terrain type and underground structure of the landscape, allowing for “true” erosion in which rocks and differing kinds of soil can be unearthed. Soil maps are often used in real life, when taking samples of farmland or geographical surveys (citation), allowing me to easily compare my simulation results to those in real scenarios.

Sediment, river splines, artistic approaches etc

Also foliage! And map generation! Read the proposal too for additional info

What can we learn from these/develop on from their methodology? Why have I chosen to use a soil map based on real life research (accurate sediment simulation)?

# Methodology

Planning, structure, and initial plans for the project.

How did I approach this? Class structure, models used, techniques for processing and rendering.

OpenGL, Perlin noise libraries, etc.

Equations used for fluid dynamics- this is a focus as I’ve been modifying these for the program’s behaviour. Show working and approach to these problems.

# Implementation & Results & Analysis

How I actually implemented the program. Explain development cycle, any potential issues (discuss mesh deformation tech and plan changes there)

Talk about any scope changes (cut back on the CPU vs GPU focus and more on simulation?)

And tech/mathematical changes to methodology?

Explain final implementation and how it came to be. Screenshots of program running, potential code snippets of mathematical implementation in C++. How to convert from series of equations into readable code?

Results of final implementation- look at the simulation in depth and probe for any particular strengths or weak points. Examples of landscapes with varying properties and the effects of water on them.

Lots of diagrams, examples, code snippets, etc.

What have I accomplished? Compare with existing models & simulations in terms of realism & representation. Looking back, would I use a node-based or particle-based simulation?

Did I hit my targets? Can I simulate an ox-bow lake?

# Conclusions

Improvements/time constraints. What could I add? (Temperature, more varying sediment types, complex foliage, etc.)

Analysis of results and **comparison with real life data.** This will require additional research into real events & geographical features.

# References & Appendices