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)

# Introduction

This portfolio project will investigate the methodology of creating rivers in videogames and create a tool to assist with the creation of digital landscapes. Digital representations of geographical features are often created by artists, with little or no reference to real-world geographical data. While artistically impressive landscapes are prevalent in modern videogames, especially those in an open-world environments, the geographical basis behind these features is often forgotten, producing inaccurate or unrealistic rivers and pools. The rise of procedurally generated games also often turns a blind eye to real-world data, instead opting for a simple representation of “a line of water” (https://ieeexplore.ieee.org/abstract/document/7295776). A tool that could generate a landscape, simulate years of fluid movement and erosion, and then provide accurate soil and landscape height data to artists could assist in ensuring artistic landscapes use a more realistic basis.

Node-based simulations of sediment pick up and deposit could allow a digital representation of hydrological landscape features. Such a system would allow for the creation of complex geographical features found in rivers (such as ox-bow lakes) that are scarcely seen in artistic landscapes and assist in the creation of far more naturally-inspired rivers. Features like bank erosion and sediment transfer(https://www.therrc.co.uk/MOT/References/EA\_DEFRA\_Sediment\_transport\_and\_alluvial\_resistance\_in\_rivers.pdf) are rarely present in these representations due to their complexity(http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.449.5576&rep=rep1&type=pdf) and computational intensity to simulate(<https://huw-man.github.io/Interactive-Erosion-Simulator-on-GPU/>).

Particle-based systems can also be used to streamline fluid simulation due to their more simplistic mechanical nature- it may be impractical for an assisting program to run on a landscape for several hours if a map is constantly being iterated on by artists. Various fluid simulation methods exist using node-based, particle-based, and mesh-based systems, and analysing various options and finding the correct simulation method will be vital in the creation of such a tool.

Aims

The aim of this project is to create a tool that generates a randomized 3D landscape and manipulates it to form rivers and lakes through simulation of fluid dynamics, exploring the effects of sediment acquisition and deposition. The tool will create landscape with natural-looking height differences, a simple representation of foliage, and soil maps determining the exact properties of the land at any given point. It will be able to run a rainfall and spring water simulation, showing the effect that fluid would have on the landscape through means of erosion and sediment deposit over multiple years. Fluids should be able to flow through the landscape and form pools, causing erosion and behaving as they would in traditional fluid dynamics (http://www.jlakes.org/config/hpkx/news\_category/2015-06-01/PhysicsofLakesVolume3MethodsofUnderstandingLakesasComponents.pdf) , transferring solids from banks and cliffsides. Simulation will be particle-based, representing the movement of fluid down the landscape, and forming pools as it comes to rest. The precision of the model should be variable, allowing for both large and small-scale simulation, from a single riverbank to a kilometre squared of land.

The simulation will not account for the effect of humans or animals on the landscape, the freezing and thawing of water, or the effects of altitude on the behaviour of fluids and gasses. The inherent complexity of these features would require a full study (https://www.sciencedirect.com/science/article/pii/S2211379717302437.) and would prove difficult to simulate in the scope of this project.

The key objectives of this project are:

* Determine the optimal method of fluid simulation for use in creating a map creation tool
* To create an algorithm to generate a random landscape with varying terrain height, existing bodies of water, and a large range of soil and rock types.
* To simulate the formation of rivers and flow of water over the generated landscape, including the effect of rain and spring water flowing and pooling without use of a pre-defined river spline.
* To simulate the erosion of terrain and deposit of sediment through fluid dynamics and force calculation to manipulate the generated landscape over time.
* To allow specification of map properties, so the user can change both terrain generation and water behaviour to affect the simulation outcome. Such examples are hill height and rarity, chance of natural spring generation, scale and size of landscape generation, and the amount of rainfall to simulate within a given year.
* To compare the results of this updated model with real-life geographical data and iterate for maximum realism.

# Literature Review

Digital artistic representations of fluid behaviour are most commonly seen in videogame landscapes. It can be argued that even the most basic representation of fluid movement, such as the logs in Frogger (1981) (citation), are an artistic take on the way water behaviour can affect a game. However, in the modern era, complex open-world games such as Red Dead Redemption 2 (date) (citation) provide artistic representations of real-world locations such as the Hudson River. These representations often stem from artistic works referencing the same area, such as those of Albert Bierstadt (<https://www.polygon.com/red-dead-redemption/2018/10/26/18024982/red-dead-redemption-2-art-inspiration-landscape-paintings>)

Geographical experts are often involved in the development of these digital landscapes, due to the complex nature of fluid behaviour and variety of landscape features that may form as a result (https://core.ac.uk/download/pdf/301635894.pdf). Procedurally-generated games often neglect reference to such behaviour, in favour of performance instead. Games such as Rimworld(date)(citation) use simple spline-based river generation, which can often cause unintended behaviour and unrealistic-looking landscapes.

Landscape splines are often used in many non-procedural games too(https://docs.unrealengine.com/4.26/en-US/BuildingWorlds/Landscape/Editing/Splines/) , designating the points at which a river flows to, and automatically filling the area with a simple fluid model or plane. While this can provide an accurate representation of man-made channels, it often fails to look create natural-looking areas of water. Ideally, fluid simulation could be used alongside or instead of splines, to create realistic landscapes with the correct soils and sands surrounding bodies of water.

Studies into fluid simulation for games have been used to generate pools, and simple moving bodies of water, on large-scale environments in the past (<https://ep.liu.se/ecp/034/010/ecp083410.pdf>), but it is currently not a commonly-applied practice. In theory, this could be simplified using an authored or generated landscape, then a fluid simulation representation could be run to ensure that it is geographically accurate. Fluid simulation representations have existed for centuries, since the time of Archimedes(citation), but computational methods allow for such representations to be run on a far larger scale. In game world creation, it could be used to simulate landscapes, and provide geographical representations of real mineralogical features.

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 liquid’s surface, in a similar node-and-vector system. These can also account for temperature and viscosity, 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 (<https://ep.liu.se/ecp/034/010/ecp083410.pdf>). 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 several minutes 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, which should allow landscape deformation in a reasonable amount of time (under an hour.)

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 use 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, as well as deposits developing on the edge of rivers. This would allow for a far more accurate representation of a river bank, which could be used as artistic reference during map creation for a game. 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.

The Hydraulic Erosion algorithm also considered the behaviour of foliage on the landscape, simulating tree spread and growth. However, McDonald’s study was on a smaller scale than what I wished to simulate, and I was concerned at the computational cost of simulating individual trees. A simpler method was used in my final product.

# 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.

Potential advantages/disadvantages of this approach compared to traditional methods

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

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