

Comparison of Flow Path Mapping Between Unreal Engine and ArcGIS: The Potential Role for Game Engines in GIScience

by

Amerald Fang

A thesis

presented to the University of Waterloo

in fulfillment of the

thesis requirement for the degree of

Master of Science

in

Geography

Waterloo, Ontario, Canada, 2025

© Amerald Fang 2025

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Advances in the videogame industry, particularly game engines, offer promising, unconventional tools for processing spatial data and representing complex geographical processes through integrated physics. This thesis explores the potential of using Unreal Engine (UE) as a multi-disciplinary platform for combining simulation models from the field of Computational Fluid Dynamics (CFD) with GIS. We present a case study implementing a fluid simulation workflow using Smoothed Particle Hydrodynamics (SPH) and quantitatively compare its results to conventional flowpath mapping methods (D8). A multi-spatial resolution raster comparison revealed that the UE model produced flow paths with a similar length to traditional methods, but with fine-scale disagreements on where flow occurs. The vector path analysis found that the UE model produced more but shorter paths than the D8. The comparison highlights the viability of game engines for dynamic simulation and suggests extensions to broader geocomputation applications such as erosion modelling. Moreover, this research demonstrates how leveraging game engine capabilities can contribute to a more integrative evolution of GIScience.

Acknowledgements

I would like to express my deepest gratitude to Professor Derek T. Robinson, who found my passion for game engines at the intersection of GIS and building models, and offered guidance, insight, and patience that have been invaluable throughout this project. I have been truly privileged to work on a one-in-a-million opportunity, which I will remain grateful for and remember for life.

I am also grateful to my thesis committee: Bruce MacVicar, Michelle K. Ruty, for their willingness to learn about my research, examine, and provide thoughtful feedback and encouragement.

Special thanks to the staff in the Games Institute and the Game Development Club, for their events and facility that assisted me with finding my way in the dark and connecting with experts in the community.

Additionally, my respect to Elisabeth from the Writing and Communication Centre, for their patience and advice in reviewing my thesis.

This research was supported by NSERC under the ID: 562706-2021.

I acknowledge the use of GenAI, specifically Grok, for assistance in editing, searching for literature, and organizing a portion of the content by condensing paragraphs.

I am indebted to my colleagues in the Geospatial Lab: Hongliang, Brandon, Chenxi, Hayden, and Lan, for therapy sessions as I frustrate over seemingly unsolvable problems, providing technical guidance, and for fostering a supportive environment by listening to my ramblings.

Finally, to my family and to my partner, thank you for your unwavering love, encouragement, and the much-needed distractions that kept me grounded throughout countless nights of questioning my sanity.

Dedication

Dedicated to games, for teaching me the meaning of journey, the means to escape, the joy of discovery, for providing growth from challenge, catharsis from triumph, and healing from trauma.

Table of Contents

List of Figures	xv
List of Tables	xvi
List of Abbreviations	xvii
Chapter 1. What Is a Game Engine?	1
1.1 Introduction	1
1.1.1 Research Gap	2
1.1.2 Objective	2
1.2 Engine Structure	3
1.2.1 Unity	6
1.2.2 Unreal Engine	8
1.3 Contemporary Workflows	11
1.3.1 Static Environment Models	13
1.3.2 Computer Graphics Simulations	14
1.3.3 Integrating Multi-disciplinary Views	15
1.4 Opportunities for Environment Science	17
1.4.1 Time-Dependent Simulations	17
1.4.2 Integrated Physics	19
1.4.3 Real-time Rendering	20

1.4.4 Data Compatibility	21
1.4.5 Multi-disciplinary Collaboration	23
1.4.6 Represent Dimensions	24
1.4.7 Modular Development Environments	26
1.4.8 Existing Standards & Pipelines	27
1.5 Limitations of Game Engines.....	29
1.5.1 Rigid Structure.....	29
1.5.2 Unused Overhead	31
1.5.3 Rapid Iterations.....	32
1.6 Discussion	32
1.6.1 Specialized Engines	33
1.6.2 Community Resources.....	33
1.6.3 Low Maintenance High Performance	34
1.6.4 Experimentation.....	34
1.7 Conclusion.....	35
1.7.1 Next Steps.....	35
Chapter 2. Literature Review	36
2.1 Introduction	36
2.2 Theme I: Game Engines in Environmental Science.....	36

2.3 Theme II: Challenges in GIScience	39
2.3.1 Traditional Philosophies	40
2.3.2 Modern Perspectives.....	41
2.3.3 Challenges	42
2.3.3.1 Challenges in Capturing Temporal Dynamics	42
2.3.3.2 Challenges in Geo-Visualization	44
2.3.3.3 Challenges in Geo-Computation.....	45
2.3.3.4 Challenges in Validation and Interoperability	46
2.3.4 Theme III: Flow Path Modelling in GIS & Computer Graphics	47
2.3.4.1 GIS Methods	48
2.3.4.2 Eulerian to Lagrangian.....	49
2.3.4.3 Computer Graphics	49
2.3.4.4 Validation of Computer Graphics Models	52
2.3.5 Gap Synthesis	53
Chapter 3. Bridging Disciplines.....	54
3.1 Introduction	54
3.1.1 Research Gap	56
3.2 Conceptual Framework	58
3.2.1 Dynamic Processes	58

3.2.1.1 Integrating Time.....	59
3.2.1.2 Integrating Models	60
3.2.1.3 Structure and Modularity	61
3.2.1.4 Entity Component System	62
3.2.1.5 Principles of Dynamics	64
3.2.2 Representation, Realism, and Interpretation.....	65
3.2.2.1 Data Types	66
3.2.2.2 Graphics Simulation.....	66
3.2.2.3 Engine Constructs	67
3.2.2.4 Principles of Representation	68
3.2.3 Interaction.....	69
3.2.3.1 Exploration.....	69
3.2.3.2 Experimentation	70
3.2.3.3 Participation	70
3.2.3.4 Principles of Interaction.....	72
3.2.4 Validation	73
3.2.4.1 Reference Model.....	73
3.2.4.2 Visual Assessment	74
3.2.4.3 Reproducibility	75

3.2.4.4 Principles of Validation	77
3.3 Discussion	78
3.4 Conclusion.....	79
Chapter 4. Case Study	81
4.1 Introduction	81
4.1.1 Research Gap.....	82
4.1.2 Objectives	83
4.2 Research Materials	85
4.2.1 Study Site.....	85
4.2.2 ArcGIS.....	86
4.2.3 Engine Tools.....	87
4.3 Methodology	90
4.3.1 Pre-processing	90
4.3.2 Reference Flow Path.....	93
4.3.3 Fluid Simulation	94
4.3.4 Data Export.....	104
4.3.5 Validation	110
4.3.5.1 Raster Comparison.....	110
4.3.5.2 Vector Comparison	113

4.4 Results	115
4.4.1 Viability Overview	115
4.4.2 Key Findings.....	119
4.4.2.1 Raster Analysis	119
4.4.2.2 Vector Analysis.....	122
4.5 Discussion	127
4.5.1 Observations	127
4.5.2 Computation Expense.....	129
4.5.3 Model Design	130
4.5.4 Assumptions	134
4.5.5 Limitations.....	137
4.5.5.1 Export Limitation.....	138
4.5.5.2 Congested Pipeline.....	139
4.5.5.3 Additional Limitation Testing.....	141
4.6 Conclusion.....	142
4.6.1 Future Opportunities.....	144
Chapter 5. Synthesis.....	145
5.1 Introduction	145
5.1.1 Objective	145

5.2 Discussions.....	146
5.2.1 Dynamics	146
5.2.2 Representation.....	147
5.2.3 Interaction	147
5.2.4 Validation.....	148
5.2.5 Literature Context	149
5.2.5.1 Relational Space.....	149
5.2.5.2 Design Tensions.....	151
5.3 Limitations	153
5.3.1 Scale Balance	153
5.3.2 Conceptual Compatibility	154
5.3.3 Resource Availability	155
5.3.4 Simplifications	156
5.3.5 Optimizations.....	157
5.4 Broader Implications.....	158
5.4.1 Theoretical	158
5.4.2 Practical	159
5.5 Future Directions.....	161
5.5.1 Realism	161

5.5.2 Data Enrichment	162
5.5.3 Model Enhancement.....	163
5.5.4 Interdisciplinary Models.....	166
5.5.5 Plugins	167
5.6 Conclusions.....	167
References.....	169
Appendix A Additional Literature Review.....	219
A.1 Research on Modifiable Temporal Unit Problem	219
Appendix B Additional Tables and Figures.....	221
B.1 Comparison of Point Cloud Processing Tools.....	221
B.2 Unreal Engine Materials.....	221
Appendix C Hardware/Software Specifications	231
Appendix D High-Level Shading Language Code	232
D.1 Populate Neighbor Grids.....	232
D.2 Intra-particle Collision	233
D.3 Accumulate Forces	238
D.4 Accumulate External Forces	240
D.5 Update Position	241
D.6 Mark Grid Location.....	242

Appendix E Jupyter Notebook Code Blocks in R	243
Appendix F Python Scripts for Vector Analysis.....	245
F.1 Absolute Length Deviation	245
F.2 Calculate Direction Deviation.....	245
Glossary	246

List of Figures

Figure 1. Dataset of farm field and study area of basin	86
Figure 2. Workflow diagram.....	93
Figure 3. Visualizing the exported raster from UE	109
Figure 4. The different paths of the two models overlaid on top of each other.....	119
Figure 5. Magnitude Aggregations across scales.....	124
Figure 6. Directional Aggregations across scales	125
Figure 7. Mapping challenges to conceptual pillars	131
Figure 8. The DEM of the cropped study area.....	224
Figure 9. Stair-case artifacts are exposed in lit shading conditions to show the loss of precise elevation data in lower bit depths	225
Figure 10. D8 flow paths visualized with stream orders overlaid on DEM	226
Figure 11. A simplified color scheme and perspective to show slow moving particles as darker blue and faster as brighter shades of blue.....	227
Figure 12. A top-down orthographic perspective of the model during simulation in an unlit environment.	228
Figure 13. The evolution of particles tracing their travel paths onto a render target.....	229
Figure 14. Collapsed flow paths produced from UE’s simulated output. The lines with average width of above 10 meters are depicted as brighter paths.....	230

List of Tables

Table 1. Universal Transverse Mercator (UTM) extent coordinates of crop area by easting and northing.....	91
Table 2. Parameters used for coordinates in UE.....	92
Table 3. Primary emitter properties	95
Table 4. NSF system properties	96
Table 5. Friction coefficients are small because particles are expected to collide frequently. More flow movement is encouraged	99
Table 6. Parameters that affect intra-particle collisions.....	102
Table 7. Simulation stability parameters	103
Table 8. Extent for secondary emitter.....	104
Table 9. Alignment parameters within NSF and the Render Target file	105
Table 10. Secondary emitter custom modules and properties	105
Table 11. Aligning scale of grid to world.....	106
Table 12. Total Operating Characteristics of the multi-spatial resolution analysis.....	120
Table 13. Vector statistics between the UE model and D8 model.....	122
Table 14. Vector comparison metrics at multiple aggregation scales	125
Table 15. Modified parameters in built-in modules.....	221

List of Abbreviations

A

Agent-based models (ABMs)	5
---------------------------------	---

C

Central Processing Unit (CPU).....	17
Community Surface Dynamics Modeling System (CSDMS)	78
Computational Fluid Dynamics (CFD)	19
Compute Unified Device Architecture (CUDA)	13

D

Digital Elevation Models (DEMs).....	45
--------------------------------------	----

E

Eight Direction (D8).....	73
Extended Range (EXR)	104

F

Filmbox (FBX).....	22
Findability, Accessibility, Interoperability, Reusability (FAIR)	83
Frames Per Second (FPS).....	18

G

Geographic Information Science (GIScience).....	2
Geographic Information System (GIS).....	3
Graphics Processing Unit (GPU).....	17

H

High Dynamic Range (HDR)	104
High-Level Shading Language (HLSL)	96

I

Integrated Development Environment (IDE)	29
--	----

L

Laser (LAS).....	22
LiDAR point clouds (LAZ)	22

M

Mean Absolute Deviation (MAD).....	113
Modifiable Spatio-Temporal Unit Problem (MSTUP).....	226

Modifiable Temporal Unit Problem (MTUP).....	43
<i>N</i>	
Niagara Simulation Framework (NSF).....	10
<i>O</i>	
Object-Oriented Programming (OOP).....	61
<i>P</i>	
Partial Differential Equations (PDE)	60
Particle Attribute Reader (PAR).....	107
Portable Network Graphics (PNG).....	91
Position Based Dynamics (PBD).....	89
<i>R</i>	
Random Access Memory (RAM).....	87
<i>S</i>	
Smoothed Particle Hydrodynamics (SPH)	iv, 51
Software Development Kit (SDK)	166
Standard Tessellation Language (STL)	22
<i>T</i>	
Tagged Image File Format (TIFF).....	109
Tagged Image Format or World File (TFW)	91
Triangulated Irregular Network (TIN).....	90
<i>U</i>	
United States Department of Agriculture (USDA).....	21
Universal Soil Loss Equation (USLE).....	13
Universal Transverse Mercator (UTM)	91
Unmanned Aerial Vehicles (UAV)	39
Unreal Engine (UE).....	20
<i>V</i>	
Video Random Access Memory (VRAM)	92
Virtual Reality (VR)	25
<i>W</i>	
Water Erosion Prediction Project (WEPP)	14
Wavefront Object (OBJ).....	22

Chapter 1. What Is a Game Engine?

1.1 Introduction

Game engines are software composed of several interrelated toolsets and libraries that facilitate the construction of virtual experiences, which are typically manifested as a game. A game engine is conceptually akin to a development framework that is used to avoid the duplication of code and reduce the time and cost (i.e., increase the efficiency) of building a virtual experience or game. There are various components in a typical game engine, but the most relevant ones are the rendering component for 2D or 3D graphics, a physics handler for collision detection, and a scene graph to process spatial data from objects, as well as scripting. These components can be linked together to manipulate data in real-time, simulate interactions, and among other uses, produce high-quality visualizations of the phenomena being simulated. The role of game engines in game development has been described as analogous to the engine of a car, whereby it provides the power and is the heart of the game (J. Romero quoted in Lowood, 2014). In other words, game engines are essential, and similar to how a person does not need to understand the innerworkings of a car engine to grasp its function and essential role, their development and inclusion of visual scripting has enabled game engines to be used as general-purpose software for not only the development of games, but also to advance scientific modelling (Rehn, n.d.).

Over the past 10 years, there has been increasing interest in the use of games and game engines in academia and professional settings, especially in education, aviation, urban planning, and architectural disciplines. For example, Flight simulators developed with game engines, such as X-Plane 11, have been certified by the Federal Aviation Administration in the United States for pilot training (Plane Professional: X-Plane, 2018). And game engines are also being explored as a platform in urban planning to visualize and navigate large urban datasets in an immersive environment (Perhac et al., 2017). The combination of improvements of game engine software as a framework for developing games or simulations; hardware that enables

the computation of big data; increasing realism of graphics (Wolf, 2012; Predescu, Mocanu, 2020); and continued advancements of physics within the engine are enabling the generation of simulations that more closely approximating many aspects of reality (Richter et al., 2021) including many key processes in the natural sciences.

1.1.1 Research Gap

Despite the potential use of game engines to represent natural processes, there is a gap in research on the possibilities for game engines to overcome challenges in Geographic Information Science (GIScience) and build dynamic process-models for environment science. To address the gap in research, this thesis outlines five objectives:

1. Identify opportunities for game engines to contribute towards creating dynamic models from the lens of GIScience.
2. Review the existing literature on how game engines have been used to address problems in environment science, challenges that remain in GIScience, and techniques from the computer graphics field that can address these challenges.
3. Suggest a conceptual framework for game engines to align its opportunities to the challenges faced by modelling in traditional Geographic Information System (GIS) tools.
4. Demonstrate the potential of game engines by creating a flow path model and contrast its results to the D8 model.
5. Synthesize the experience gained from creating the model with the conceptual framework to assess how challenges have been addressed.

The goal of each chapter corresponds to achieving the above objectives respectively.

1.1.2 Objective

To identify opportunities offered by a game engine, a primer is required to inform environment scientists about the components of game engines, existing applications, and their

potential contributions to advancing environment science. The objective of this chapter is to explore and examine potential applications of game engines as a tool for scientific visualization, analysis, and simulation in environment science. In addition, the chapter identifies the advantages of game engines relative to conventional toolsets, bridge connections between the two fields, ideate unique and innovative solutions, as well as translate existing techniques into workflows that can be performed on a game engine. To achieve this, two of the most popular game engines, *Unity* and *Unreal Engine*, will be reviewed based on their defining characteristics and compatibility with existing environment science workflows from case studies. Additionally, the chapter will discuss current limitations that game engines face when used outside of their original context and offer strategies for overcoming these to better adapt to and exploit the advantages of game engines over conventional methods.

1.2 Engine Structure

The evolution of game engine structures has followed a trajectory from highly specialized, proprietary systems to increasingly versatile platforms. Early engines in the 1990s were tightly coupled with specific games and hardware configurations (e.g., Quake, Doom), offering limited extensibility (Gregory, 2018). As development costs increased and technical requirements expanded, the industry shifted toward reusable frameworks capable of powering multiple projects. Modern engines have embraced open development ecosystems, extensive documentation, and visual programming interfaces that have democratized access to sophisticated real-time rendering and simulation capabilities previously available only to large development studios (Christopoulou & Xinogalos, 2017).

Most modern game engines (e.g., post-2010s) have been built using a modular structure (Gregory, 2018), which facilitates their use as a general-purpose software for the creation of virtual environments that reflect or depart from reality. This modularity revolves around four primary components: physics, graphics, scene, and scripting tools. The physics component simulates real-world physical behaviors including collision, gravity, fluid dynamics, and material properties, determining how objects interact within the virtual space. The graphics

component handles the visual representation of the environment through rendering pipelines, shader systems, lighting models, and texture management to translate the underlying data into what users ultimately see. Scene management tools provide an organizational framework for virtual worlds that coordinate spatial relationships, object hierarchies, and environment states. Finally, scripting tools enable developers to define logic, behaviors, and interactions without modifying the engine's core code, allowing for extensibility and customization of the virtual environment.

The specific implementation and intended usage of components differentiates game engines, making some more appropriate for accomplishing tasks in their niche (e.g., simulation, prototyping, film production). As the structure of a game engine evolves, whole components or sub-components are swapped out and replaced with other approaches, which enables the engine to emphasize performance, accuracy, customization, realism, or development speed for a variety of projects (e.g, 2D or 3D games). Over time, components of a game engine have become more reusable and generalizable as well as capable of creating diverse virtual experiences. The combinatorial assemblages that can be created by game engines for different audiences have been fueling their adoption in various industries.

The modular components of game engines can potentially contribute to the development of modular agent-based models (ABMs), the concept of which remains an ongoing and somewhat elusive goal for many academic modellers (e.g., Bell et al., 2015). Of these components, the most relevant to an environment science application are physics and graphics, which are often intertwined and co-operating at any time during a real-time environment in games. Physical behaviors of the real world, such as collision, sound, and light can function similarly in the virtual environment developed using a game engine, given the correct mathematical model and appropriate visual representations. The implementation of these components necessarily involves inherent trade-offs that engine developers and users must navigate.

There is no single engine that excels universally across all applications, as optimizing for one aspect often requires compromising on others (Christopoulou & Xinogalos, 2017). These are some of the primary considerations for choosing a game engine:

Performance versus accuracy: achieving physically accurate simulations typically demands greater computational resources, potentially giving rise to optimization schemes that sacrifice collision details to combat unstable frame rates or limiting scene complexity.

Visual fidelity versus hardware compatibility: higher fidelity rendering increases hardware demand, requiring careful consideration of techniques based on available equipment.

Development speed versus accessibility: easier development often correlates inversely with customization depth, whereas engines offering streamlined workflows may limit low-level access to core systems.

These trade-offs typically explain why different engines have evolved to serve different market segments and use cases.

Evaluations of a suite of physics engines demonstrate that there is no single physics engine that outperforms others (Boeing, Bräunl, 2007). Instead, physics engine choice should be based on the anticipated frequency of certain problem scenarios and how well an engine is tuned to handle it (Boeing, Bräunl, 2007). However, given the number of simultaneous and consecutive calculations to maintain smooth framerates, rounding errors in simulations (e.g., Karssenberg & De Jong, 2005) can compound to create unexpected physical behavior. Including more precision to reduce rounding errors comes at the cost of computational overhead and does not resolve the fundamental issue of compounding errors. In response, game engines typically use workarounds such as actively checking the physics conditions in every frame and correcting behavior based on a generalized understanding of what is expected to be realistic (Gregory,

2018). As a result, this usually creates a perceptually correct approximation rather than an empirically accurate simulation.

The approximation prevalent in physics engines can significantly influence simulation outcomes. In the field of Computational Fluid Dynamics (CFD), the use of a higher quality mesh where the simulation environment occurs may potentially yield less accuracy (Messler et al., 2021). Given these types of approximations, modellers must understand the inherent trade-offs in choosing different approaches and be aware of the opportunity to improve game engine components to yield empirically accurate results or to create high-quality visualizations of their work.

Unity and *Unreal Engine* are two of the industry's leading providers of game engines. The differences in their detailed implementation necessitate a trade-off in how the engine functions, is used by developers, and the advantages it offers. In addition, different terminology may be used (with consideration of branding) by their userbase to refer to architectures and technologies of each respective engine, despite their utility being similar in nature. To illustrate this example and to reduce ambiguity, a preview of similarities and differences from a small section of broader game engine architecture is provided below.

1.2.1 Unity

The game engine with the most documentation and least barriers to entry is *Unity* (Christopoulou & Xinogalos, 2017; Barczak & Woźniak, 2020; Vohera et al., 2021). *Unity* encourages a component-based architecture called Data-Oriented Technology Stack (DOTS) to structure projects, where components represent self-contained objects that include both data and behavior. The expected approach to development is composition-oriented, focusing on the assembly of generalizable behaviors of a particular object. In the context of simulations, this structure is well-suited to modelling a group of objects, each with advanced behavior and a large amount of data (e.g. small ecological populations). It is flexible and intuitive due to the

modular nature of components, and enables rapid prototyping with short iteration cycles, which is advantageous in the exploratory stages of projects (Christopoulou & Xinogalos, 2017).

Physics in *Unity* is realized by attaching physics-related components, such as mesh colliders that operate based on aspects of classical mechanics, onto a game object. Physics of 3D virtual worlds created using *Unity* are handled by NVIDIA's *PhysX*, a middleware that provides a solver for 3D simulations (Hongpan et al., 2010; González et al., 2017). Components typically have customizable parameters that inform the physics engine of how the object resolves physics interactions with the environment and other objects by default, such as mass, friction, gravity and collision. Four collision detection modes are offered, *Discrete*, *Continuous Speculative*, *Continuous* and *Continuous Dynamic*, ordered from least computationally expensive to most expensive respectively.

The user is expected to make an informed decision on appropriate modes to use based on the velocity of the object or the frequency of collisions when selecting the most appropriate modes (Unity, 2024, "Choose a collision detection mode"). For example, the *Discrete* mode checks for collisions only during fixed time steps and is best suited for objects that do not move faster than a certain velocity threshold. The *Continuous Speculative* mode checks for collisions with other fast-moving objects, but with a relatively low accuracy. The *Continuous* mode checks for collisions with other static colliders in its path even between frames but may misbehave with other rigid bodies, and *Continuous Dynamic* mode uses the most amount of computation to check for collision with all static and moving objects.

For an environment science application, *Discrete* would ideally work for features that are not expected to actively collide (e.g., ground, large vegetation, permanent geological features), *Continuous Speculative* for fast-moving collisions (e.g., bullets, vehicles, explosive debris), *Continuous* for one dynamic object in a static environment (e.g., breaking dams, slow moving glaciers), and *Continuous Dynamic* for multiple moving objects (e.g., loose debris in a landslide, sediment in water, flock of animals). Built-in physics offers some control over

surface material to enable the simulation of factors such as friction and elasticity. Compared to the other engines, *Unity* offers an easier understanding and application of the features.

In addition to its physics engine, *Unity* features highly customizable rendering, mainly the *Universal Rendering Pipeline* that prioritizes compatibility when porting to various platforms, and the *High-Definition Rendering Pipeline* designed to deliver advanced graphics to specific devices with high-end hardware (Unity, 2023, “Render pipelines introduction”). Shaders are the primary method of communicating how objects are rendered in *Unity*. These components can be customized to modify how materials are interpreted by the graphics engine by considering its UV coordinate, vertex, lighting information, and more calculated through mathematical models. The UV coordinate is a set of two numbers in an axis similar to X and Y on a 2D graph commonly used in 3D modelling and shader computation to describe how 2D textures should be rendered when projecting them onto 3D surfaces. The reason UV coordinates are used is to distinguish them from the X, Y, Z coordinates already used for describing the location of objects in 3D space. Real-time global illumination is achieved through the use of another middleware called *Enlighten*, which consumes discernible amounts of system resources and should only be used for slow, simple dynamic lighting (Unity Technologies, 2018, “Real-time lighting”). An alternative lighting method involves the use of baked lightmaps, whereby the scene lighting is pre-calculated and static, but an illusion of dynamic lighting is achieved by modifying materials to simulate phenomenon such as color bleeding. Both methods limit the level of realism that can be achieved in the *Unity* engine or have boundary limitations that are easy to reach (Soni & Kaur, 2024).

1.2.2 Unreal Engine

Unreal Engine is well-regarded for its particle simulation, realistic graphic processing and real-time rendering capabilities that have been increasingly used in professions such as filmmaking, news broadcasts, education, and safety training (FLIPSIDE, 2021; Rickards et al., 2021). The engine offers a diverse set of features that enable the generation of more complex and realistic environments than *Unity* (Soni & Kaur, 2024). The increased complexity and

realism over *Unity* require more computational power, an expensive hardware configuration, and *Unreal Engine* is more dependent on internal-company software developments than *Unity*. In *Unreal Engine*, the physics engine, called *Chaos Physics*, is integrated with the *Unreal* system and workflows (Unreal Engine, n.d., “Physics”). The benefits associated with greater internal-software development are optimization and ease of use compared to multiple layers of modules and middleware that must adhere to the constraints of an overarching application programming interface. For example, *Chaos Physics* features procedural geometric fracturing, which can be used to simulate complex debris formation, but because of its integration in the system it can easily be used with other modules (e.g., physics fields) that governs where the debris should go.

The collision features in *Unreal Engine* achieve similar results to *Unity*, but through a different paradigm. Objects can block, overlap, or ignore each other. These interactions can be defined by the user through a table of checkboxes. Blocking means they collide, overlapping means they touch or intersect each other without blocking, and ignoring means no interaction (Unreal Engine, n.d., “Physics”). The interface to define how object types detect collision are termed collision channels. Each channel represents a type of object or interaction, such as “*WorldStatic*” for static objects or “*Pawn*” for characters. By assigning objects to specific channels, developers can control how distinct types of objects interact, ensuring only collisions of interest are processed, which optimizes performance and allows for precise control over physical interactions.

There are various object types that the engine provides as preset templates, the physics behavior of an object is built into their class type by default. For example, *Static* objects remain immobile and are designed to act as collision obstacles (e.g., ground, walls, sky), *Dynamic* objects move programmatically or through animation but are not simulated (e.g., doors, grass, clouds), while *Rigid Bodies* are fully simulated with physical properties. A solver is used for collision detection among all object types, applying any constraint resolution to update positions and orientations on every physics frame. Specifically, Worthen (2022) provides an

in-depth exploration of *Unreal Engine*'s in-house physics framework, focusing on its internal classes and locating the implementations of calculations. It explains how the engine tracks and updates the positions of actors in the scene through simulation and API interactions. Among the collection of data structures, the most significant ones are *UWorld*, which represents the map and manages actor registration, scene ticking, and querying. *FPhysScene_Chaos*, which is the primary interface to the physics simulation, and *FPBDRigidsSolver*, which is the solver for rigid body behavior.

Unreal Engine adopts an *Actor-Component* system which is similar to the encouraged architecture found in *Unity* (DOTS), but prioritizes a tightly integrated, event-driven workflow with an emphasis on the use of a visual scripting tool called *Blueprints*. *Unreal Engine* heavily relies on an inheritance-based hierarchy of objects, which is similar to traditional object-oriented programming. The particle system in *Unreal*, called Niagara Simulation Framework (NSF), is particularly optimized and suited for complex visual effects and simulations on a large scale.

To efficiently visualize physical behavior in a scene, *Unreal Engine*'s newest rendering techniques represent solid, liquid and gaseous states of matter through the introduction of the NSF. It achieves this through a process called *Depth Field Analysis* where particles react to their environment by analyzing the camera's perspective of the scene, projecting an optimized overlay of the expected simulation result to the screen. The method is described in a Eulerian fluid simulation paper (Rabbani, Khiat, 2020), where 3D forces such as wind, motion, and collision can be re-interpreted to 2D UV surfaces to optimize calculations without hindering the quality of visual output. Particles in NSF are the centerpiece to creating a physically believable state of matter, so forces such as collision, rigid or loose constraint, and gravity can also be rendered in Unreal. The collision module used in NSF achieves Newtonian motion of rigid bodies using *Continuous Collision Detection* techniques (Redon, 2003; Unreal Engine, 2020, "Advanced Niagara Effects | Inside Unreal"; ibbles, 2020; C., 2022), which can correct particle trajectories when misbehaviors occur, and Position Based Dynamics (PBD) to handle

motion of deformable objects via repeated collisions between particles using their difference in mass (Müller, 2007). This approach is also dependent on the framerate as the implementation performs predictions for each frame based on the velocity that a certain particle on-screen is traveling. However, NSF also handles simulation complexity in user-defined stages, so it allows for finer control of the performance versus accuracy trade-off. In higher frame rates, this method produces a visually convincing render of how particles would react to physical interaction with the world and each other.

The 5th addition to the *Unreal Engine* software release (i.e., Unreal Engine 5) includes major upgrades to its existing rendering pipeline, allowing for the direct import of source photogrammetric scans with fewer demands for optimized geometry (Unreal Engine, 2022, “Unreal Engine 5.0 Release Notes”). The feature called *Nanite* automatically controls the Level of Detail (LOD) that objects should be rendered at, further simplifying the work required to assemble performant photorealistic scenes and greatly increasing the number of objects that can be placed without affecting graphic performance. *Unreal Engine 5* also includes *Lumen* which features dynamic indirect global illumination for real-time lighting simulations, automatically handles color bleeding to nearby objects, advanced indirect shadowing, and geometrically precise reflection. These provide significant improvement over traditional methods which rely on direct and static lighting techniques, acting as a scalable and flexible tool for creating realistic lighting in a scene (Xin et al., 2024).

1.3 Contemporary Workflows

Procedural algorithms are used in both the domain of Geographic Information Science (GIScience) and computer graphics for representing dynamic natural processes, but with different goals. For example, GIScience applies hydrological modeling to understand and predict real environmental systems, whereas computer graphics generate visually convincing or stylized surface effects for visual media and interactive virtual environments. Despite their divergent purposes, both fields share a core workflow of using procedural methods to simulate

surface dynamics, creating a unique multi-disciplinary bridge between these otherwise distinct disciplines.

Three views emerge to differentiate digital representations of geographic phenomena: particles, object, and fields. These terms may create ambiguity when used to mean different approaches within GIScience and computer graphics disciplines. To differentiate their meaning and clarify their use within the context of using a game engine to model agents, the following definition is used:

The particle view models numerous, simplified discrete entities, such as individual raindrops in a runoff simulation (Shen et al., 2004). The emphasis is on their collective behavior, movement, and distribution, often governed by simplified rules suitable for large-scale simulation (e.g., simple collective agents).

The object view focuses on distinct, identifiable geographic features with well-defined boundaries, such as specific land parcels, lakes, or buildings (Anand et al., 2016). Each object serves as a container for a rich set of attributes and can be involved in complex, entity-specific spatial analyses and processes (e.g., complex individual agents).

The field view conceptualizes a grid-based continuum where a variable (e.g., elevation, temperature) possesses a value at every point across the landscape (Couclelis, 1992). These represent agents that diffuse over space and time. This perspective emphasizes gradual spatial variation rather than discrete, bounded entities.

Both particle and object views represent the world using discrete entities, differing primarily in the level of individual complexity and the focus (collective behavior vs. individual characteristics and interactions).

Numerous environment science models have been developed with the field view in GIScience, made accessible through use of tools such as ArcGIS (Bregt et al., 2002; Bekkema, 2016). In parallel, the field of computer graphics has also built platforms (e.g., Compute Unified Device

Architecture [CUDA] and OpenGL) and developed techniques to simulate realistic renderings of natural phenomena. However, these techniques are less accessible due to the lack of a sharing platform. For example, both disciplines have taken an interest in producing models that attempt to capture the flow of water and their influences on landscape evolution, but there are no common repositories or standardized data exchange between the two domains. As a result, hydrologists often rely on static and simplified models in desktop GIS without graphic simulations, while graphics developers build visually rich but physically ad hoc demos.

1.3.1 Static Environment Models

The quantifying, mapping, and modelling of sediment movement and its mitigation remains a scientific and societal challenge given the effects of erosion and deposition on crop yields (e.g., Li et al., 2011), harmful algae blooms (e.g., Heisler et al., 2008), and water quality (e.g., Fu et al., 2019). While environment scientists have been using a variety of different approaches to quantify erosion and deposition, most models rely on raster data and GIS software to provide a field representation of surfaces and their change. The calibration and validation of these models use empirical data collected through in-situ measurements (e.g., Shahabinejad et al., 2019) or remote sensing (e.g., drones, Meinen & Robinson, 2020; satellite, Gao & Zhang, 2021).

Many types of erosion models have been developed and categorized to distinguish their fundamentally different approaches (Refsgaard, 1996; Andualem et al., 2023). Simple empirical models, such as the Universal Soil Loss Equation (USLE), have been extended to cellular applications in a GIS environment to utilize spatially complex datasets and visualize input and output data on maps (Csáfordi et al., 2012). These types of models are limited in their ability to accurately represent sediment movement (e.g., Meinen & Robinson, 2020) and surface change given that they exclude conservation of mass and do not maintain mass balance (Trimble & Crosson, 2000).

Complementing empirical models are process models, such as those from the Water Erosion Prediction Project (WEPP), which incorporate processes such as irrigation, infiltration, water balance and plant growth (Flanagan et al., 2012). The hydrology component of the WEPP models water erosion generally as a 3-stage process, before, during and after a rainfall event (Baiaumont & Agnese, 2010). First, it uses the Green-Ampt infiltration equation to calculate the rate of water absorption into the soil in the initial positions of constant along the surface. Then, kinematic wave equations are used to predict the movement of run-off flows on the slope. Once rainfall ceases, the model calculates the declining depth of overland flow as water is progressively infiltrated and stored in the soil. Due to its physically-based approach, the model is often applied to concentrated flow and cropland ephemeral gullies in small watersheds to evaluate sediment transport (e.g., Karimov et al., 2015). Notably, the WEPP model requires a very large number of input parameters, most of them determined from field exhaustive investigations.

Static models have a lack of consideration for dynamic transformations and their compounded effects on erosion (Tucker et al., 2001), and the details of topography are not always represented accurately in models across multiple studies (Lidberg et al., 2017). For example, a flow path may change over time due to degradation or various external factors in a realistic environment, which are not always reflected in even the most widely used empirical (e.g., USLE) and process-based models (e.g., WEPP). Models in one study region may require a significant amount of re-tuning to be applied in another. Therefore, despite the current level of sophistication, existing models in GIScience struggle to represent the processes of overland flow and erosion and deposition sufficiently to accurately model the evolution of a real physical landscape over a long period of time, with high spatial transferability, and at field scales (Raza et al., 2021).

1.3.2 Computer Graphics Simulations

The computer graphics industry has been generating computer-simulated representations of hydraulic erosion without the use of GIS tools (e.g., Beyer, 2015). Most

studies used a particle-based approach (e.g., Chiba et al., 1999). The particle approach allows particles to follow classical mechanics-based physics that can exert influence on a virtual landscape, creating a real-time visual representation of erosion and changes in the rate of its occurrence (e.g., Benes & Forsbach, 2002; Neidhold et al., 2005; Odhner & Fleming, 2009). In these studies, particles were not explicitly observed nor manipulated during flow for analytical purposes and their potential to carry information on sediment deposition is not explored in depth. (e.g., Benes, 2007).

Musgrave et al. (1989) is a representative model in this category of simulations, being one of the first to simulate erosion by storing water and soil values in a field view, and sediment is transported through the interaction of neighboring grid cells. Popular factors that are taken into account in these models are the effects of thermal, hydraulic, and wind forces (e.g., Balázs, Balázs, 2011; Mei et al., 2007) for the purpose of creating realistic visuals. The terrain used by these computer graphics studies (e.g., Shaker et al., 2016) were typically generated from randomly produced height maps by mixing noise textures, so their accuracy and effectiveness on real terrain data at the watershed scale remain unknown. In addition, the modelling of water flow was simplified through the use of level-of-detail (LOD) techniques on 2D approaches (e.g., Dong et al., 2010). Results from a literature review identified only three studies that focus on the feasibility of computer graphic techniques for the use of environment science ([Appendix A](#)), and little to no validation or accuracy assessment despite its similarity and relation to many existing erosion models used in soil science.

1.3.3 Integrating Multi-disciplinary Views

The distinction between current approaches in soil science and the computer graphics industry appears to be disappearing since they are both moving toward the same goal, both consider and combine the effects of multiple realistic factors to achieve a high-quality simulation of the change in soil over time. However, it is important to recognize their original differences, goals and overlaps between the models produced by these industries.

There are three distinct disappearing differentiations:

Purpose: Static environment models were designed for scientific analysis, decision-making, and planning, emphasizing accuracy and realism in representing real-world spatial processes. They were data-driven, while computer graphics models still are aesthetic and simulation-driven, serving visual and entertainment needs.

Data Source: Static environment models use real-world, spatially referenced data for precision, while computer graphics models use abstract or synthetic data for flexibility and visual appeal.

Scientific Validity: Static environment models prioritize accuracy through empirical and physically based methods, while computer graphics models use abstract and artistic techniques for rendering efficiency in real-time environments.

The growing overlap occurs because artistic approaches in computer graphics models have become physically-based to a degree in order to pursue realism, as demonstrated in Beyer, (2015). Moreover, theoretical models for fluid, smoke and fire are increasingly being used by particle simulations to implement advanced visual effects that closely mimic their real-world counterparts (Xu et al., 2014; Singh et al., 2016). This overlap aligns with the research goals of GIScience as research methodologies have shifted toward exploring ABMs (e.g., Bell, 2017), and particles in these simulations can be understood as agents that follow natural rules of interaction based on the laws of physics.

In terms of data views, most erosion studies (e.g., Andualem et al., 2023) carried out with ArcGIS rely on an Eulerian view performing calculations in a grid space focusing on the locational interactions between neighboring cells (e.g., López-Vicente & Navas, 2010), whereas the computer graphics industry has used a variety of tools pursuing a Lagrangian perspective with multiple moving objects acting as agents inducing change. This difference also brings to light the hardware dependency of each approach, as the Eulerian view will usually rely heavily on a capable Central Processing Unit (CPU), and the Lagrangian view

requires a powerful Graphics Processing Unit (GPU) in addition to a powerful CPU. Researchers have shown interest in the topic of expanding a 2D grid to 3D (Brooks, Whalley, 2008), potentially simplifying the visualization of complex raster data. There have also been studies layering Lagrangian approaches with Eulerian models to create a hybrid method for simulating hydraulic erosion (e.g., Krištof et al., 2009).

While the ArcGIS suite of software has been the main enabler for spatial data analysis in a Eulerian view, there still lacks a standard approach for Lagrangian methods with equal availability and power. The use of computer graphics has shown promise in simulating erosion, but its potential for scientific validity has yet to be critically evaluated. An attempt to build a specialized tool suited for Lagrangian studies from scratch would incur massive resource costs, as well as demand the geospatial expertise of a software development team. Furthermore, the current state of GIS tools is heavily limited by their licensing, pricing, and stagnating visualization features. Merging these issues together, a multi-faceted problem is formed relating to the monotonousness of available methods for observing data, the narrow accessibility of tools and compatibility with other platforms, a lack of competing alternatives, and an increasing need to visualize and present objective findings in a realistic, intuitive manner to the end-user.

1.4 Opportunities for Environment Science

The emergence of game engines invites an opportunity for cross-disciplinary collaboration, enabling environment scientists to take advantage of the substantial engineering done from computer graphics. Eight potential opportunities are suggested that position game engines as ideal candidates for integrating with existing GIS workflows.

1.4.1 Time-Dependent Simulations

As an aid to traditional GIS and spatial analysis tools, game engines offer complete control over the actors of a study area and their behaviors in the environment, as well as the

environment itself. For example, the unique control on simulating timesteps, tracking individual flow paths of water particles, and applying a real-time 3D transformation to the environment because of their movement. Given the flexibility of game engines, such a use case would be appropriate for experimenting with various sampling methods, performing accuracy tests, as well as comparing multiple approaches to measure natural processes. Results could then be presented through a visually immersive and interactive experience that is easy to publish and distribute.

Game engines typically employ a fixed-timestep architecture to decouple simulation stability from rendering framerates, ensuring deterministic outcomes across various hardware configurations (Unity Technologies, n.d.; Nystrom, n.d.). This approach prioritizes accurate depictions of frame-independent processes over unstable framerates that often occur in real-time performance scenarios, enabling physical behavior such as collision resolution or rigid body dynamics to stay consistent irrespective of rendering rates (e.g., 30 Frames Per Second [FPS] and 60 FPS will both display the same results in respect to real-world time). This type of architecture is well-suited for simulating natural processes for scientific inquiry since the simulated process can be interrogated at any point in time and at any given scale.

Fixed-timestep methodologies are also commonly employed in CFD tools, particularly to compute the state of a system in a later time step using known current states, to simulate fluid flow and related phenomena. However, their application can be constrained by computational demands, such as when simulating large spatial scales at high resolution (Schmid & Burrell, 2016). While CFD simulations are often associated with microscale phenomena, such as molecular interactions, or the scale of mechanical components like airfoils and turbines, they are not limited to these domains. CFD has been used in studies that encompasses mesoscale and macroscale systems, including atmospheric modeling (Labovský & Jelemenský, 2013), oceanography (Tang & Wu, 2010), and urban airflow studies (McAlpine & Ruby, 2004), at scales that align with the land parcels and urban districts frequently examined in GIScience. Nevertheless, simulating these larger scales with fine detail remains computationally intensive,

and their implementation often requires advanced numerical techniques and significant resources to balance accuracy and feasibility. Game engines can simplify this development process as it provides a platform for integrating real-time CFD simulations with spatial data visualization, enabling researchers and engineers to interactively explore fluid dynamics within virtual environments.

1.4.2 Integrated Physics

The physics simulations powered by NVIDIA hardware in the computer graphics industry, such as those integrated into *Unreal Engine*, provide a framework that can be used to substantially advance our ability to model natural processes. The adoption of methodologies in the computer graphics industry introduces a level of detail and interactivity unattainable by conventional 2D models that reduce physics to coefficients in equations. While environment scientists typically rely on simplified representations (e.g., Qin et al., 2007), these approaches struggle to capture the complexity of phenomena like fluid dynamics or erosion. The GPU-driven techniques developed for games are capable of rendering real-time, three-dimensional (3D) simulations, and are ideal for visualizing and manipulating natural systems.

The advancement of graphics technology in recent years offers transformative opportunities for environment sciences to leverage cutting-edge tools to create visually realistic and interactive models for research and analysis. NVIDIA hardware is typically chosen as they are competitively used for their capability and efficiency in handling simulation scenarios compared to other graphics hardware (e.g., Chentanez, Müller, 2010; Vanderbauwhede, Takemi, 2013), which can influence the methodologies for visualization or scope of analysis. NVIDIA offers various frameworks and tools, such as NVIDIA Gameworks and FleX, a suite of tools to enhance video game development. It includes various components designed to improve graphics, physics, and overall game performance, particularly on NVIDIA GPUs. NVIDIA Gameworks has made significant advances in recent years on the software front of simulating fluid physics through heavy reliance on GPU computations. These advancements open the path to multi-disciplinary simulation-driven research. For example, a sparse volume

representation of solid objects through the use of the particle concept can appropriately leverage the application of GPU power through a game engine's rendering systems (Harada, n.d.; Mower, 2016). Physics models such as these can be compiled and imported into game engines such as Unreal Engine (UE) to create real-time interactive demonstrations (Connor Hill, 2020).

The physics simulations in NVIDIA *FleX* rely more on a deterministic rather than a stochastic model, as the same parameters will produce the same results each time it is run as long as the spawning behavior is defined to be non-randomized and chaotic actors are removed from the simulation (Unreal Xinda, 2020). Therefore, it can act as a laboratory for experiments that require tight constraints for the properties of the water body, including discharging an appropriate amount, adjusting the rate of the flow and defining a clear path. The resolution of water observed in NVIDIA *FleX* demonstrations reveals the implications that water particles may appear quite large using this method, leading to a low-resolution view of potential flow pathways in a hydrological flow study (Gartia Jenkins, 2016).

1.4.3 Real-time Rendering

There is a separate path forward beyond the typical visualization methods currently used by environment scientists, involving the use of 3D modeling software capable of animation and is focused on high-quality visual presentation. These models which are implemented in custom software or are integrated with graphics tools like Blender and Maya, excel in pre-rendered visualizations, prioritizing display of intricate details (e.g. foam and bubbles), as demonstrated by Ihmsen et al. (2012). However, these lack the same capacity to represent physical processes and response to changing physical conditions in real-time, which is why game engines would be more appealing to the scientific audience for the purpose of simulating physical phenomena.

The difference between choosing interactive and pre-computed 3D environments can be a missed opportunity for projects that attempt to translate their workflow into 3D environments

through visualization-oriented software. For example, the United States Department of Agriculture (USDA) published a project attempting to characterize and translate existing 2D workflows to 3D models for visual simulation of coastal change, attempting to develop techniques and introduce new methodologies (Ackerman, Cave, 2018). The goal of the study was to better communicate the challenges of climate change and its impact on coastal conditions by using 3D modelling software and animation to accurately portray coastal ecological systems, with the intention to inform decision-makers of potential sedimentation and erosion patterns. However, the methodologies introduced must function under multiple assumptions, such as constant erosion rates, and fixed material composition, as well as ignoring the effects of how wind forces would impact the dispersion of water through sites during storm events, all of which would be difficult to account for in a tool that was primarily visualization-oriented, but achievable in a game engine.

3D modelling software focuses on precomputed, batch-processed results rendered as static visualizations or video outputs, which lack runtime dynamics and interactivity. In contrast, game engines integrate dynamic temporal control, permitting real-time modification of timestep intervals (e.g., accelerated or decelerated simulation speeds) and iterative parameter adjustments (e.g., fluid viscosity, gravitational forces) during execution. This interactive layer, coupled with the capacity to modify environmental geometries or boundary conditions mid-simulation, creates bidirectional feedback that distinguishes game engines as platforms for exploratory, adaptive physics modeling rather than deterministic offline computation. This operational flexibility positions game engines uniquely for applications requiring iterative experimentation, such as interactive engineering prototyping or real-time environmental scenario testing.

1.4.4 Data Compatibility

There is an opportunity at the intersection of data compatibility and game engine adoption, particularly for leveraging remote sensing and photogrammetry. While the integration of new technologies often necessitates data transformation, game engines readily

support a wide array of formats commonly generated by drones and LiDAR techniques. Specifically, workflows involving heightmaps and 3D scene construction are streamlined through support for formats like Laser (LAS), LiDAR point clouds (LAZ), Filmbox (FBX), Wavefront Object (OBJ), Standard Tessellation Language (STL) (e.g., 3D models), and Geospatial Data Abstraction Library (GDAL)-compatible raster formats. This compatibility minimizes the need for extensive data manipulation, facilitating direct integration. A major difference between the data handling of traditional spatial data and game engine assets is the clear separation of shape and materials, as models are typically treated as a combination of vertex, faces, and pixels. This means there is a need to utilize the projection of 2D textures, matching and mapping them to 3D planes precisely, often via shader techniques and UV manipulation. This inherent compatibility presents a significant opportunity for environment science research to seamlessly integrate their data into game engines, establishing a prescribed workflow for visualizing and interacting with their data.

Most existing photogrammetry tools do not have the same level of flexibility and modelling capability as a game engine. For example, CloudCompare offers specialized analysis capabilities for technical users working directly with point cloud data, but infrequent updates and a rigid user interface impact its usability (CloudCompare, n.d.). Potree excels in visualization, smoothly loading vast point clouds and supporting procedural data streaming on the web, but it lacks features for editing or transforming point clouds (Schütz, 2015). Though not originally intended for point clouds, Blender’s ability to process them comes from its versatile plugin platform, which includes add-on functionalities for converting point clouds into objects and applying simple edits, yet it faces challenges with performance on larger datasets and integration into a broader pipeline (Blender, 2025). Lastly, *Unreal Engine 5* boasts visual quality and rendering capabilities that exceed specialized point cloud tools (Unreal Engine, 2025, “LiDAR Point Cloud Plugin Overview in Unreal Engine. Epic Developer Community”). The engine provides advanced lighting, shadowing, and material systems that can transform point cloud data into photorealistic visualizations. This rendering quality creates

opportunities for immersive applications beyond technical analysis, including virtual tours, simulations, and interactive experiences.

The selection of an appropriate platform for point cloud data handling ultimately depends on specific project requirements, technical resources, and intended outcomes. Complex projects may benefit from a hybrid approach, using CloudCompare for initial analysis, Potree for web-based sharing and collaboration, and UE for creating high-end immersive experiences from the same underlying point cloud data. This combined workflow leverages the strengths of each platform while mitigating their individual limitations.

1.4.5 Multi-disciplinary Collaboration

Traditional models are reliant proprietary software with restricted accessibility and collaboration. Game engines are available to the public and designed to facilitate multi-disciplinary collaboration within development teams (Gregory, 2018). Harrap et al., (2019) has shown that Unity proves to be a robust decision support tool for simulating the collision of rocks for up to hundreds and possibly more objects interacting in real-time, proving to be a valuable tool for communication with stakeholders. The paper also suggests that GIScience can gain considerable advantages from the functionality available in modern game development tools, overcoming several limitations in traditional workflows. The limitations discussed in the paper include the difficulty of modifying internal methods to modify simulation rules, lack of support for diverse data formats, long development periods, and an overly specific target usage that limits its ability to benefit a wider range of similar 3D physical simulations. Game engines overcome these limitations as they provide a flexible, 3D platform for users of all skill levels and allow them to learn through interactions. Since game engines are created and maintained by teams of professional software developers, they reduce repetitive coding, contain robust physics engines which can be generalizable to any physical simulation, and support diverse data imports similar to a GIS. The increasingly large and active ecosystem enable rapid development, growth of a community, comprehensive and continuously updated documentation, and customizable models, facilitating software integration and data transfer.

Game engines can also overcome challenges with knowledge mobilization by acting as a common platform for sharing research. Frieze & Herwig (2008) discuss how game engines can visualize scientific data in 3D, making it easier for researchers and the public to understand and interact with the findings. For example, Bryceson et al. (2022) demonstrates the integration of system dynamics modeling into the Unity game engine to develop an Augmented Reality (AR) application, visualizing eutrophication for educational purposes. WebGL technology enables game engines to export some of the fluid simulations so that they can be experienced through a modern browser (Wallace, n.d.; Blaskovic, n.d.; PavelDoGreat, 2019).

Most game engines include networking capabilities, such as network replication, which allow multiple users to interact with the same simulation in real-time. The *Unreal Engine for Research* website (Rehn, n.d.) mentions this feature, noting its utility for experiments with multiple participants on separate network-connected devices. This facilitates collaborative research in educational settings, enabling scientists and students to explore shared virtual environments together. Game engines accelerate dissemination of knowledge through their compatibility with various accessible platforms. The WebGL format export allows simulations and complex data to be accessible via web browsers, without the need for the client to possess expert knowledge or own expensive software, which broadens the scientific reach to a wider audience. Resch et al. (2014) demonstrated the effectiveness of WebGL for web-based 4D visualization of marine geo-data and discusses the likely improved accessibility and support for hardware and browsers for WebGL in the future, which is true in the present. Some preliminary hydrological erosion simulations can already be experienced through WebGL (e.g., LanLou123, 2018).

1.4.6 Represent Dimensions

Climate modellers often face challenges with precipitation data, which can be difficult to visualize in traditional 2D models, hindering effective analysis and communication. For example, traditional methods often fail to convey the multidimensionality and uncertainty inherent in such data, leading to misinterpretations (Gerst et al., 2020), and overly complex

visualizations can cause the result to become difficult to interpret for a non-expert audience. Virtual Reality (VR) presents an opportunity for enhancing the visualization and communication of complex scientific data, with game engines serving as the foundation for developing immersive VR experiences. These challenges can be addressed through features such as interactivity (Laksono & Aditya, 2019), advanced shading techniques (Kolb et al., 2018), real-time pipelines and efficient data handling (Yin et al., 2024), which work in unison to create realistic environments that make complex data more accessible and understandable. Horton et al. (2019) showcases this potential through a visualization framework that allows for intuitive exploration and manipulation of complex datasets. Due to the growing demand for better and more advanced visualizations, shading techniques and rendering pipelines are becoming an increasingly important aspect in the development of game engines. Most modern game engines enable the user to write custom shader code for parallel processing on the GPU. This assists users with creating complex visualizations that extend beyond basic rendering capabilities, which can integrate processes that are used by various parts of the engine to create real-time simulations.

The multi-dimensional capabilities of game engines open new avenues for visualizing and modelling interactive phenomena that are inherently volumetric and difficult to represent in traditional models. For example, air pollution dispersion models involve complex interactions between building geometry, wind patterns, and emission sources, which determine the spatial distribution of pollutants. The user might interactively change the direction and strength of the wind in order to influence simulation outcomes. Game engines can simulate these interactions in real-time, showing how pollutants flow around buildings, accumulate in street canyons, or disperse at different elevations. This volumetric approach to air quality modeling enables more nuanced analysis than 2D concentration maps, potentially informing more effective mitigation strategies and urban design practices. Similar examples can apply to modeling subsurface water movement, forest canopy interactions, or indoor air circulation.

Aside from visual-related simulation, game engines provide the opportunity to explore other dimensions of an environment that are less capable of being visualized in space, such as audio. The virtualized 3D environment allows complex placement of audio samples, as well as making adjustments to various properties that may affect the perceived audio in an environment, such as attenuation, occlusion, reverb and more. These simulations can then be experienced, visualized, and analyzed in real-time through a variety of means. The user can choose to navigate through an immersive environment to listen from multiple points, recreating surround audio either through headphones or VR headsets. Similarly, many more invisible factors can also be modeled and accounted for, such as wind and light, which are currently being explored (Kim, 2016; Kohek et al., 2019) by leveraging the flexibility of game engines or a related technology.

1.4.7 Modular Development Environments

Game engine scripting is fundamental to the creation and manipulation of dynamic virtual environments. Commercial game engines typically offer two approaches: visual scripting and code-based scripting, which can be mixed to obtain various degrees of success depending on the situation. Scripting enables precise control over object spatial properties such as position, rotation, and scale, as well as the coordination of interactions among other entities. It is the building-blocks that define the behavior and interactions of objects within the game world. These scripts link and define the logical relationships between different elements, manage data storage and transfer, and control the flow of the virtual environment. The rockfall simulation by Harrap et al. (2019) demonstrates the use of scripting by utilizing collision events of a wall to set up a filter in which certain objects are allowed to pass through based on preset custom conditions. It is also possible to mix multiple behaviors together in the same scene this way and consider a wide range of factors that may contribute to simulating complex environment interaction.

In addition to traditional code-based scripting, visual scripting provides an alternative approach that leverages a graphical interface and node-based systems, trading ease of use and speed of

development for performance. The act of scripting visually usually entails the combination and connection of pre-packaged nodes, each containing some generic logic for achieving a function. Nodes expose their inputs and outputs, and their internal processes cannot be altered. This method accelerates prototyping and iteration, as well as making it more accessible to users without extensive programming backgrounds. Through visually mapping out the flow of operations, users are able to intuitively see and adjust the relationships and interactions among nodes, streamlining the process of crafting complex behaviors and interactions within the virtual environment. For example, visual scripting tools like the Blueprint system in *Unreal Engine* enable domain experts without coding expertise to create logic-driven simulations, such as flood dynamics or land-use changes (Newell et al., 2024; Yin et al., 2024). Although traditional coding experience is not strictly necessary with visual scripting, a basic understanding of programming practices and conventions remains needed to effectively construct scripts that are both efficient and robust. Depending on the design philosophy of each game engine, vastly different operations are presented for manipulation of nodes though active or passive actors. For example, landforms could be morphed to change in elevation as hydrological processes are occurring in a Lagrangian view, sediment load and deposit of each water particle can be tracked and visualized, or if the audio level surpasses a certain threshold an avalanche simulation can be triggered.

1.4.8 Existing Standards & Pipelines

The modular structure of components encourages a clearly divided teamwork structure, which conveniently simplifies collaboration between multi-disciplinary members. Each member only needs to be an expert in one part of the assignment, expect proper input and produce an output for another process. For example, a programmer might specialize in scripting logic and events between entities, another team member in placement and reconstruction of a virtual environment, while an interaction designer concentrates on integration of interfaces. The game engine ensures that each component interacts smoothly with the others, and if a particular module needs an update or fix, it can be addressed

independently and re-connected to the relevant functional parts. These pipelines not only simplify collaboration among multi-disciplinary experts but also streamline the debugging and optimization process, ultimately accelerating the development cycle.

Management, development practices and pipelines have already been established for popular game engines such as *Unity* and *Unreal*. Game engines encourage and facilitate the combination of programming patterns, which can be used as a reference for planning and constructing project teams with the goal of putting existing theories, models, and workflows into practice (Schmalstieg & Stork, 2019). For example, the Singleton pattern ensures that important programmatic processes or managers of an object group exist as a single instance throughout the project. This prevents redundant resource allocation and centralizes control, making it easier for team members to access shared components reliably. On the other hand, the Observer pattern enables different modules to communicate effectively without being tightly coupled, allowing one part of the system to broadcast changes or events, which multiple listening components can then react to independently. These patterns can be used in combination and interchangeably to coordinate complex information transfer between team members, disparate game engine modules or environment science models.

In addition, it has become standard for third-party version-control software such as GitHub to be used in many of these development pipelines. Compared to ArcGIS cloud storage options, a free git repository can offer storage space and is not technically demanding to manage or access from multiple platforms, giving the team full control over their datasets. Tutorials and guides are also widely available online providing free training resources for independent functionalities in a game engine. These self-contained aspects of components and inter-component dependency can lower the learning curve of the tool, improve intra-communication in the team, and define goals or issues clearly.

1.5 Limitations of Game Engines

Game engines provide a highly structured environment for building virtual experiences, but this structure comes with performance and usability costs. For example, to preserve real-time performance, engines lock down and hide core physics implementations and embrace expert-only workarounds. This creates rigidity in how an engine must be used in comparison to a traditional Integrated Development Environment (IDE) where packages and libraries can be freely imported.

1.5.1 Rigid Structure

The structure of a game engine can offer a flexible and powerful interface for non-expert users, but this modularity also carries inherent constraints. Specifically, it creates a weakly defined link between components. Early implementations of game engines gave users the authority to perform simple changes such as importing assets and modifying interactions. The reusing of a codebase simplified development and enabled a focus on adding in new content but caused the components of previous games to become outdated and unsupported in early game engine architectures. This linear connection of assets (affecting interactions, reliance on component compatibility) also carries over to modern-day game engine architecture, partly because it has worked well so far in production environments. For example, the separation of processing for physics, rendering methods, methods of interaction and also animations remain largely unchanged in the modern game engine architecture (Gregory, 2018).

However, this separation creates challenges when simulating phenomena that span multiple traditional component boundaries. Branching of features in the engine provided room for the independent growth of components, but they do not always provide the ideal solution to simulating particular situations. For example, data exchange between systems that do not explicitly belong in one component category (e.g., fluid dynamics requiring simultaneous physics calculations, particle rendering, and environmental collision systems) is difficult, as it may involve tight cooperation between multiple components and entities in an environment

that is not as commonly paid attention to in games. Thus, the framework is rigid for developing cohesively linked forms of simulations that are efficient. Tools in game engines cannot compete with models developed using lower-level languages (e.g., real-time ray tracing algorithms optimized in CUDA for photorealistic lighting).

Modern engine architecture has not changed its software framework since it has worked well for the purpose of games, so efforts have instead been put toward developing new techniques to raise the efficiency of each component. As a result, a high-quality simulation demands expert education in the subject, a theoretical as well as practical knowledge of the implementation details, which requires detailed planning of the systems architecture as well as familiarity with the engine-native data types and constructs, as evidenced by Gregory (2018).

The performance of physics simulations in game engines are dependent on an implementation often hidden and non-customizable in most commercialized engines. For this reason, game developers use physics as a supporting tool to create illusions that speed up computation without affecting the end visual result.

Since the simulation of physics is costly, workarounds are always preferred in game engines to replace real physical calculations wherever possible (Lewis & Jacobson, 2002). This is done so that the user may assume a complex process has been carried out, when in reality, it was achieved through alternative means. Many features implemented in game engines are designed around this process and encourage the use and discovery of optimal workarounds. One example of this is the aforementioned lightmap baking technique for global illumination in Unity, with the developer working under the assumption that there will be no dynamic lighting changes happening in the scene so the perceived light reflections do not need to be calculated constantly every frame, it is instead layered once over the 2D textures on the surface of static objects which visually renders a similar result (Unity Technologies, 2018, “Baked lighting”). Therefore, no actual lighting or ray tracing calculation is done, or it is only performed once because the assumption is that this aspect of a simulation will remain static. Shortcuts such as

these raise scalability and create significant performance improvements, but it sacrifices the scientific calculation of accurate, dynamic real lighting conditions (Jones et al., 2003).

These shortcuts may also exist in physics simulations involving movement of objects, as entirely physical scenes can suffer from compute overload due to the emphasis on calculating collisions with multiple objects and surfaces in every frame, which often lead to unwanted or unreliable physics interactions. Therefore, to ensure the illusion is not broken and physical processes are accurately presented where expected, physics engines are often tightly coupled with their engine, allowing little to no modifications from the engine user (Ullmann et al., 2023).

There are a few exceptions to physics engines being tightly coupled with their engine, such as Godot. As an open-source game engine, Godot is able to circumvent the issue by allowing the user to replace the physics engine with a custom implementation, but it also demands the capability of that physics model to be compatible with other parts of the engine, or enough time and resources to develop such a component (TomBursch, 2019; Bradfield, 2018).

1.5.2 Unused Overhead

Game engine components are expected to provide a variety of uses for the user to simplify the learning and understanding of development processes. To achieve this, game engines sometimes include more purposes into one functionality, which sacrifices computation efficiency for features that may not be useful to the average user but will nonetheless consume resources (Messaoudi et al., 2015). For example, a standard physics engine component might offer support for complex simulations like fluid dynamics, cloth physics, and soft-body interactions, even though a specific application might only require basic rigid-body collision detection. Most simple operations in a visual-scripting environment add auto-generated boilerplate code and generic pipelines that may be convenient for the average game developer, but may cause optimization issues for a model builder (Strodick & Schattkowsky, 2024). These

unused capabilities still consume memory and may produce computational overhead during initialization or runtime checks.

This problem may be reduced through a process called “Engine Stripping”, which involves identifying and removing unnecessary engine modules, libraries, or features that are not used by the specific project. Then, the engine is often recompiled or packaged without these components, resulting in a smaller, more optimized build tailored to the actual needs, thereby reducing memory footprint and potentially improving load times and performance. However, successfully implementing this approach requires a deep understanding of the engine internal architecture and a high level of technical expertise (Hoon, 2003).

1.5.3 Rapid Iterations

Unlike traditional GIS tools such as the ArcGIS suite, game engines can change drastically from one iteration to the next. For example, Unity frequently releases new features, keeps multiple different versions, and modifies or removes older features by replacing them (Unity, 2020; Garry’s Blog, 2020). This dynamically changing aspect of game engines may be unpredictable and interfere destructively with the needs of long-term development projects. In relation to packages obtained from the community or asset store, they may also be incompatible in different versions, leaving the user to manually debug or rely on the original creator to also update their assets.

1.6 Discussion

There is a trend of game engines being widely adopted as their potential extends beyond scientific demands. This has led to the development of more engines for specialized industries and communities (e.g., Evergine, Unigine). Communities of game engines also develop integrations that facilitate further multi-disciplinary research (Rehn, n.d.). Thus, an ecosystem is emerging around game engines as a well-maintained platform for general software development and experimentation.

1.6.1 Specialized Engines

Engines are becoming more specialized. Recent iterations of game engines such as Evergine have begun adding features geared toward applications in aerospace, architecture, energy, and healthcare (Evergine, 2021). Unigine 2 has become specialized in physics simulation for water surfaces, clouds and weather effects (Unigine, n.d.). Epic, the company behind Unreal Engine, has shown interest in creating a photo-realistic digital model of the Earth by collaborating with Cesium to apply GIS techniques on gathering data on the planet's terrain and buildings (Takahashi, 2021). Thus, there is an interest from both scientific communities and engine developers in forming a research-oriented interactive platform that can be customized and specialized to take on problems in geography.

1.6.2 Community Resources

Game engine communities provide helpful resources. Most popular engines on the market offer a variation of an asset store where other creators upload plug-in packages for the purpose of selling assets, functionalities, or custom API libraries they have developed. Users may download or purchase this community-made content to reduce production time or use them as a reference or tool for learning complex features available in the engine.

There is common ground for game developers and environment scientists. Both are interested in producing realistic simulations, and game engines provide a path forward for both industries by translating natural environment models into dynamic virtual scenes. The game engine's complete governance over a virtual space, as well as its potential for setting up scenes with advanced physics and rendering, creates an opportunity for environment science to take an object-oriented approach in analyzing, visualizing data and interpreting results. It carries the foundational potential for simulating at higher levels of realism and can present advantages over traditional GIS tools through customizability, pricing, and accessibility.

1.6.3 Low Maintenance High Performance

Users can conveniently access state-of-the-art technology by using updated versions of game engines. This allows researchers to produce higher quality visuals and immersive environments without needing to understand the underlying technological advancements. Furthermore, past releases are publicly available for download. This provides multiple versions as options for teams to choose the best toolset for their current project phase, and the community along with the asset store provides a rich ecosystem of resources and creative solutions for backward-compatibility. These benefits accelerate development and encourage experimentation, ultimately driving progress and enhancing the overall quality of long-term projects.

1.6.4 Experimentation

Game engines deserve further exploration due to their potential to spawn new workflows. The object and particle features of game engines have currently seen some limited use as an artistic tool for carrying out Lagrangian approaches to eroding natural terrain for CGI production. Models used for the basis of these simulations are similar concepts or modifications to ones used in soil science studies, but their scientific implications in terms of accuracy and precision have not yet been evaluated. There is much to be experimented with in terms of applying the game engine's method of organizing object characteristics, separating shape, material, and mass, and using it to represent realistic environments. While there are some limitations to the physical accuracy of simulations, the flexibility to access and manipulate these factors, along with accounting for unobservable forces through freely assigning visual symbology, and the power to work with a large number of agent-based interactions in a scene are all valuable contributions that game engines can bring to the GIS community. They offer new ways to think about environment modelling, uniting cross-disciplinary talent between software engineers and environment science researchers and can advance GIScience by providing a controlled space for experimentation.

1.7 Conclusion

Game engines lower the technical expertise required to build advanced simulations. They allow users to take advantage of the state-of-the-art continuous time and physics techniques (e.g., fluid simulations), provide a flexible workspace for merging diverse data formats, enable collaboration online and across various development environments, visualize results, customize workflows, and work within a standardized platform, without needing to learn and manually implement any new technology or expend efforts to maintain the engine. However, this conceptual realignment should not be merely a technical adjustment but a fundamental shift in how scientific knowledge can be implemented, visualized, and shared in interactive digital environments. To achieve this, further research is needed.

1.7.1 Next Steps

There remains a need for a coherent conceptual framework that bridges the gap between environment science models and game engine implementations. Traditional static modeling approaches need to evolve to take advantage of the interactive, real-time nature of game engines, overcoming inefficiencies and limitations in scientific visualization and simulation. A more effective paradigm would embrace agent-based modeling methodologies, identify structured approaches and principles, and align naturally with the component-based architecture and event-driven systems that form the foundation of modern game engines. Such a framework would provide scientists with a deeper theoretical understanding and relevant tools for translating conceptual models into interactive simulations, defining appropriate trade-offs between accuracy and performance, and establishing verification protocols specific to game engine implementations. By shifting toward this approach, researchers can more effectively represent complex systems with emergent behaviors, spatial heterogeneity, and dynamic interactions, which are characteristics that are inherently supported by game engine architectures.

Chapter 2. Literature Review

2.1 Introduction

Models are abstractions of a real-world system, and simulations are models executed over space and over time to mimic behavior of a dynamic real-world system (Daneshjo, 2012). The field of environment science and Geographic Information Science (GIScience) rely on various types of models to address challenging tasks, such as spatial process models (Banerjee et al., 2003), landscape evolution models (Pazzaglia, 2003), and hydrological models (Devia et al., 2015). However, as these disciplines evolve and their problems become increasingly complex, there is a need to move beyond traditional static models to deliver real time interactive simulations and visualizations. Game engines are software frameworks that present an alternative to traditional Geographic Information System (GIS) development environments, which bring opportunities for building real-time simulations and dynamic models in GIScience.

The application of game engines to GIScience requires a literature review exploring several intersecting themes: First, the existing efforts of how game engines have been applied in environmental science. Second, identify the challenges within GIScience that game engines can help overcome. Finally, a more specific thematic example of flow path modeling was chosen for review, drawing overlaps from both GIS and computer graphics fields to highlight relevant techniques within a game engine implementation.

2.2 Theme I: Game Engines in Environmental Science

A systemized literature review to explore the use of game engines within environment science was conducted using Scopus, Google Scholar, and Web of Science. The search queries were formulated as:

Scopus: (TITLE (game PRE/1 engine) OR
ABS (game PRE/1 engine) OR
KEY (game PRE/1 engine)) AND
(TITLE-ABS-KEY ("natural science" OR
GIS OR hydrology OR "environment science"))

Google Scholar: (allintitle: "game engine" OR
allintext: "game engine" OR
allinkeywords: "game engine") AND
"environmental science"

Web of Science: (TS=("game engine") OR
AB=("game engine") OR
KP=("game engine")) AND
SU=("Environmental Sciences")

86 results were documented on Scopus, with a growing trend beginning in 2017, reaching a steady publication amount of more than 5 documents annually. Of these, 39 documents were articles and 41 were conference papers, and the rest were reviews and book chapters. Google Scholar yielded 3 results. None of the results were relevant to the field of environment sciences or the goal of the review. Web of Science yielded 27 results and closely followed the same trend as Scopus.

From these databases only 106 articles were retrieved in total, among which, only 28 were deemed relevant. The criterion for relevance is defined by the application of game engines for an environment science context, with the goal of finding papers related to building interactive simulations ideally driven by process-based modelling containing some degree of numerical calculations to mimic realistic behaviors of natural features. Articles where the end use-case

was to utilize game engines purely as a tool for visualization with no numerical modelling were excluded.

From the relevant 28 articles most uses of game engines included a strong visualization component or were educational and aimed at stakeholder communication (Gross et al., 2023) on topics such as flooding (Yin et al., 2024), heat risks (Shibata et al., 2025) and climate change mitigation scenarios. Two developed workflows featuring game engines as a platform specifically for data transfer between GIS tools (Evans et al., 2023), and model accuracy assessment (Subramaniyam et al., 2024). Three papers applied Geo-design in the context of urban planning, incorporating models for wind turbine site planning (Rafiee et al., 2017; Rafiee et al., 2018), and procedural modelling of city buildings (Yang & Delparte, 2022). One other article reviewed voxel capabilities in Unity (Aleksandrov et al., 2022). The remaining articles highlighted a range of applications such as immersive 3D geovisualization for teaching landscape design (Carbonell-Carrera et al., 2021), online 3D terrain visualization using Unmanned Aerial Vehicles (UAV) images (Mahayudin & Mat, 2016), visualizing and analyzing environmental spatiotemporal mobile sensor data in urban contexts (Helbig et al., 2022), creating virtual forests for decision support and stakeholder communication (Holm & Schweier, 2024), and gamified urban planning methodologies enhanced with co-creation and participatory approaches (Kavouras et al., 2023).

The increasing trend of game engine usage appears to be greatly diversified for applications such as immersive 3D geo-visualization for landscape design teaching (Carbonell-Carrera et al., 2021), wind turbine planning platforms integrating sound propagation models and GIS (Rafiee et al., 2017), online 3D terrain visualization using UAV images (Mahayudin & Mat, 2016), visualizing and analyzing environmental spatiotemporal mobile sensor data in urban contexts (Helbig et al., 2022), creating virtual forests for decision support and stakeholder communication (Holm & Schweier, 2024), and gamified urban planning methodologies enhanced with co-creation and participatory approaches (Kavouras et al., 2023).

The majority of the existing literature incorporating game engines has utilized game engines for visualization purposes. However, game engines can be used beyond visualization purposes (e.g., Burch et al., 2022) because they represent aspects of the natural world such as gravity, mass, acceleration, velocity, collision, light, fluid, and particle diffusion among others via core engine components (e.g., physics engine, rendering pipeline, scripting modules). For example, much attention has been paid to the creation of digital twins, and game engines have played a prominent role in the creation and exploration of 3D digital twins (e.g., Rundel & De Amicis, 2023; Jiang et al., 2025). In Zhao et al. (2025), a digital twin eco-system is constructed using game engines as a tool to support decision-making in urban environments for water reserves management, integrating dynamic simulation models, explainable artificial intelligence, and data-retrieval via real-time sensors. These recent literatures seek to further leverage the potential of game engines to create multi-layered complex systems, with dynamic simulations serving as the underlying data source for a responsive visualization layer.

Therefore, despite the benefits of game engines for simulation and their potential scientific applications, there remains scarcity of research using game engines to advance modelling in environment science through 3D spatial process-based models. Building on this context, an investigation to situate the role of game engines for addressing persistent challenges in GIScience is merited.

2.3 Theme II: Challenges in GIScience

In order to position the integration of game engines with GIS workflows as a potential pathway for future geospatial research, there is a need to understand the general history and recent advancements of GIScience in the context of simulation and modelling, departing from a philosophical lens to a practical lens. The review will be structured in a general chronological order, starting from foundational understandings of space and progressing through the development of modern perspectives, reflecting contemporary understandings and methodologies. Such a structure effectively highlights the trajectory of ideas, motivations, and

culminates in identifying trends of current challenges in GIScience. This will provide an entry for a conceptual framework that addresses the challenges directly, presenting opportunities from which game engines can contribute.

2.3.1 Traditional Philosophies

In Meentemeyer's (1989) discussion of space, two philosophical views emerge: absolute and relative. Absolute space is described as an “*empty container*”, stating it is a geographer's responsibility to compile it with information. In contrast, relative space involves creating hierarchical theories from “*inter-extrapolation of high-level and low-level information to form hierarchical theories*”. This interpretation highlights the relationships between objects based on their spatial variation. During the time of Meentemeyer (1989), GIS was the primary tool for geographical analysis, but its limitations were recognized, particularly in handling temporal data and multi-scale complexity. The paper's reference to “*enhanced GIS*” suggests a vision for a tool that could overcome these shortcomings, integrating spatio-temporal data seamlessly.

Couclelis (1992) states it is important to represent space in a way natural to human cognition, so it would be a shame to apply our inherently 3D spatial understanding to 2D abstractions. The decision to pick appropriate data representations to solve certain types of problems remains relevant today, and papers often refer back to Couclelis (1992) as justification for developing new data models (e.g., Kumar et al. 2010; Peek et al. 2021). These emphasize simplifying complex geographic phenomena into accessible formats while retaining accuracy. Modelers must critically assess how audiences perceive data, the appropriateness of representations, and the unique insights non-traditional methods (beyond vector/raster) provide.

The ArcGIS suite of tools offered by ESRI demonstrate the ideas presented in Couclelis (1992), as vector polygons, points, and lines relate to the object view, while raster cells exemplify the field view. This also influences the method of query for certain spatial objects, as it is often

necessary to understand relations between objects in the scene to find and manipulate corresponding fields. The reliance on fields as the primary method of representing data extends to many recent and modern studies (e.g., Mollalo et al. 2020; Liu et al. 2020). Fields are useful and efficient for processing information but are abstract models that do not reflect the cognitive, multi-dimensional perspectives humans perceive and experience space.

2.3.2 Modern Perspectives

The GIScience community continually seeks to develop a general theory and universal framework for representing spatial and temporal data in discrete, continuous, and ideally both forms at the same time (Goodchild et al. 2007). Because of the difficulty in producing generalized theories of inherently random and unpredictable real-world occurrences, GISystems are designed for specific problems with clearly defined variables, patterns, spatial relations and representations. However, this also implies that unifying discrete and continuous representations remains a challenge, as systems prioritize tailored solutions over theoretical universality.

The framework which needs to be produced may be one that needs to address a more general point, such as recognizing the role of GIS and its advantages in specific issues and whether a different tool is needed or needs to be developed. As Gahegan (2018, pp. 22-23) suggests, GIScientists should be encouraged to build an innovation platform that integrates the complex models and simulations which currently live outside of GIScience.

Building on Gahegan (2018), the extension of the software developer role for GIScience means that it must begin to consider perspectives of other disciplines, evolve to integrate human-computer interaction principles, cognitive science, and participatory design approaches that align tool capabilities with how users perceive and reason about space. Despite advancements in geo-visualization technologies, several challenges remain in bridging these principles. For example, there exists a persistent gap between understanding how users mentally understand spatial information (Çöltekin, 2017). This understanding translates into insights that assist with

designing interaction to effectively guide user attention and support complex reasoning tasks. Developers struggle to create interaction models that are compatible with the nuanced ways in which people perceive, interpret, and engage with multidimensional spatial data (Roth, 2017). Therefore, the developers must carefully solve challenges by envisioning features that remain intuitive to users of all backgrounds and expertise levels. The ability of creating interfaces that are simultaneously engaging, powerful, and immediately accessible remains elusive for many GIS practitioners and existing geo-visualization implementations.

2.3.3 Challenges

The literature review categorizes challenges in GIScience into four themes:

- Temporal dynamics cannot be captured by traditional GIS frameworks, which favors static snapshots over evolving and changing processes.
- Visualization of data through immersive, multidimensional mediums is a frontier that requires solutions beyond existing models for conventional 2D mapping.
- Geocomputation as a means for intuitive interactions and interdisciplinary experimentation remains largely unrealized in current GIS implementations.
- Validation of models using creative methodologies must begin building trust by embracing open science and creating replicable research to achieve widespread adoption.

2.3.3.1 Challenges in Capturing Temporal Dynamics

GIS struggles with temporal data, particularly the Modifiable Temporal Unit Problem (MTUP) (Cöltekin et al., 2011) which affects data interpretation as temporal scale and resolution vary. Research on MTUP is scarce, with few mitigation strategies proposed (e.g., [Appendix A](#)). Additionally, understanding changes in geoinformation is limited by insufficient resolution of temporal data, inadequate processing techniques (Bregt et al., 2002), and underuse of time geography frameworks proposed by Hägerstrand (1981). For example,

traditional spatial models fail to capture movement data effectively, offering only static snapshots (Laube, 2015). Hägerstrand's (1981) time-space paths and prisms provide frameworks to depict movement and its constraints, stressing the importance of temporal dynamics (Meier, n.d.). Omitting these dynamics can lead to misinterpretation of environmental processes.

Modeling features that evolve in scale over time, such as melting glaciers (Kirkbride, 1993; Schild et al., 2018) or expanding forests (Reyer et al., 2015), challenges traditional approaches. In soil and erosion studies, neglecting dynamic changes hinders simulation of phenomena like oxbow river formations (Sun et al., 1996), underscoring the need for finer spatial and temporal resolutions. Agent-Based Models (ABMs) address these issues (Zellner, 2008) by simulating individual agents to capture complex dynamics. However, challenges persist in producing realistic outputs from emergent patterns (An et al., 2021) and in calibrating and validating models with non-linear responses (Ngo & See, 2012).

Designing models for complex systems like rainfall (Cox & Isham, 1988), carbon cycling (Pearman et al., 1983), soil variation (Nakane & Shinozaki, 1978), and wildlife behavior (Skidmore, 2002) is hindered by static GIS tools. Inflexible parameters in current tools limit scaling flexibility, ranging from micro to macro scales (Elsawah et al., 2020). Traditional decision-making strategies, such as Multi-Criteria Analysis (Cinelli et al., 2020), rely on subjective assumptions that oversimplify or ignore temporal dynamics (e.g., changes over time). These tools often omit local dynamics (e.g., spatially heterogeneous effects), store disconnected data (e.g., criteria and weights), and fail to capture cause-and-effect dynamics (e.g., lack of feedback loops). Instead, decision models should reflect fundamental understanding, including internal processes and emergent patterns, despite computational challenges.

Modelling dynamic environmental changes is difficult with traditional methods that offer only static snapshots. Continuous data collection (e.g., sensor networks) is resource-intensive and struggles with representing complex interactions like feedback loops (Shrivastava, 2020). In

addition, traditional landscape evolution models (e.g., CEASAR) assume steady state flows in hydrological processes, which is a simplification that can lead to biased findings (Coulthard et al., 2013).

To create dynamic human-environmental interaction models, Seppelt (2009) suggests an interdisciplinary approach that perspectives from specialized disciplines like landscape ecology should be considered in order to bring a deeper, specialized understanding to models. This sentiment can also be extended to disciplines and tools outside the field of environment and into computer science or engineering.

2.3.3.2 Challenges in Geo-Visualization

Longley (2004) emphasizes that GIS struggles to fully represent the physical world due to challenges in capturing individual decision-making and underlying dynamic processes, such as selection of habitat restoration sites (Jankowski & Nyerges, 2001) and soil erosion (Lane et al., 1988). This limitation calls for tools that can represent the complexity of underlying mechanisms from the perspective of both modellers and stakeholders, finding a balance between accurate and convincing visualizations.

The multi-dimensional nature of geography is often oversimplified into 2D models, leading to representation inaccuracies, such as in hydrological modeling where elevation impacts water flow (Hosseini pour et al., 2012). This 2D bias (Entrikin, 1977; Li, 2007; Ohori et al., 2013) restricts both analysis and problem-solving approaches.

LiDAR point clouds (Mehendale & Neoge, 2020) offer detailed 3D data, efficiently collected via UAVs (Stumpf et al., 2015). However, processing tools (e.g., Cloud Compare, PoTree) are limited, expensive, and may lack a graphic interface for accessibility ([Appendix B](#)), struggle with compatibility, often separating GIS from analysis workflows, complicating data handling (Bekkema, 2016). Despite their 3D richness, point clouds are frequently reduced to 2D representations (e.g., Digital Elevation Models [DEMs]), losing critical surface complexity (Fitzpatrick, 2024). This simplification aids mapping but sacrifices the data's original nuances.

Herman et al., (2018) show that 3D maps significantly enhance user performance and spatial understanding compared to static 2D views. Recognizing that geographic phenomena encompass more than quantitative aspects, geo-visualization faces similar representational challenges when trying to capture nuanced, human-centric aspects of place (Davies, 2020), attention has turned towards integrating other forms of rich, contextual information into geospatial analysis (Šaška et al., 2019).

Qualitative GIS (Mocnik, 2022) has recently emerged as a way to enhance geospatial analysis by integrating both quantitative and qualitative data. Qualitative data can enrich GIS analysis by incorporating place-based information, which reflects human perceptions of the environment (Purves et al. 2019). Traditional GIS tools, built for numerical data, lacks standard metrics and visualization tools to effectively handle qualitative information, necessitating new frameworks (Jung, 2009). Millington and Wainwright (2017) proposes that ABMs can embody both quantitative and qualitative aspects of the world. However, the difficulty associated with technically operating and implementing such frameworks is a barrier to many researchers (Rahman et al., 2001; Çöltekin, 2017).

2.3.3.3 Challenges in Geo-Computation

Geocomputation increasingly uses 3D data for simulating complex landscapes and urban environments, but struggles with rendering, performance, and usability (MacEachren & Kraak, 2001). Many platforms lack the ability to handle 3D data efficiently, requiring powerful hardware and specialized software. However, high-performance computing and advanced graphics are costly and not widely accessible, hindering adoption. Furthermore, platform-agnostic interactive models require significant computational power for large datasets (Liu & Sanyal, 2020). Therefore, intuitive interfaces for 3D interaction are complex and resource-intensive to develop.

Harris et. al (2017) defines geocomputation as “... *process over form, dynamics over statics, and interaction over passive response.*”. In contrast, most modern GIS platforms offer static

views, limiting visualization of time-dependent processes (Ragia & Fiume, 2007). Interactivity has been introduced to some web GIS platforms that allows exploration of time-series data and continuous changes, revealing hidden trends (Degbelo, 2022), but web-based tools face technical challenges with web development, such as browser compatibility and cybersecurity issues.

Effective use of interactive tools demands technical proficiency, which domain experts like geographers may not possess (Ziegler & Chasins, 2023). Integrating interactive geospatial tools into educational environments is challenging due to steep learning curves and the need to constantly adapt new skills (Bowlick et al., 2022). Thus, bridging this user skill gap through training and interdisciplinary collaboration remains an ongoing challenge.

2.3.3.4 Challenges in Validation and Interoperability

Advances in critical GIS have expanded spatial and temporal representations through geovisualization and multimedia integration, enabling contextualized and dynamic visualizations of geographic phenomena (Elwood, 2010). However, this necessitates a balance between creative methods and replicable frameworks. Oreskes et al. (1994) describes natural systems as impossible to validate, as they are open and not fully observable. Despite this, academia continues to create models that can be useful by validating them against observations and other models.

Validating environmental models, ensuring correspondence to real-world processes (Ngo & See, 2012) remains difficult, demanding creative interdisciplinary strategies to build trust in model performance (de Vos et al., 2011; Charney et al., 2021). Thus, GIScientists often need to learn or apply skills outside of their own expertise (Ricker et al., 2020).

Many GIS tools are developed in highly context-specific settings and underlying assumptions that are not easily transferable to different geographic or temporal contexts (e.g., Abdel-Aty et al., 2013). The lack of standardization procedures and automated pipelines forces researchers to engage in extensive pre-processing and data cleaning for virtually any project involving

multiple data sources, which incurs costs in time and effort, taking away from actual analysis to ensure compatibility (Noardo, 2022). This challenge is exacerbated when trying to combine data from various domains, where differences in data structure and semantics can lead to mixed interpretations or inaccurate visualizations (e.g., loss of data integrity, metadata).

Agent-based environmental models present unique validation challenges due to their inherent complexity and equifinality, incorporating numerous interrelated parameters that govern agent behaviors, interaction rules, and environmental responses across multiple scales (Collins et al., 2024). Shannon & Walker (2018) advocate for greater GIScience reproducibility through a framework that balances design tensions, fosters transparent community engagement, and ensures model interoperability. Effective validation also requires knowledge transfer, rigorous assumption testing, and collaborative refinement (Wilson et al., 2021). To enhance trust and reproducibility in modeling, particularly agent-based models, the GIScience community must cultivate a culture emphasizing standardized structures, transparent parameter sharing, and comprehensive workflow documentation (Wilson et al., 2021).

Bekkema (2016) notes that there is a disconnect between complex geographic processes and our modeling tools (e.g., ArcGIS, QGIS), advocating for dynamic modeling integration in GIS and industry-standard models. However, existing solutions are often either independent and specialized solutions with loose integration into GIS tools (Powers et al., 2017), or are commercial, and often not open-source (e.g., ArcHydro).

2.3.4 Theme III: Flow Path Modelling in GIS & Computer Graphics

To support the proposal and implementation of a model to demonstrate the potential for game engines to act as dynamic flow path modelling tools, the below literature review will assess traditional GIS methodologies for flow path modelling, identifying similarities and differences with methods used in the field of computer graphics. In addition, appropriate techniques for modelling realistic and dynamic fluids will be investigated for adoption into Unreal Engine (UE).

2.3.4.1 GIS Methods

Traditional approaches to flow path mapping primarily rely on static elevation data and simplified hydrological models. For example, Quinn et al. (1991) calculates flow directions by the greatest slope. These models employ algorithms that focus on the in and out flows for every cell and are limited in their ability to accurately describe the complex hydrodynamics of water movement, including the effects of inertia, variable flow velocities, and complex interactions with surface roughness.

A significant area of flow path research involves using similar numerical models to couple water flow and sediment transport dynamics (e.g., Simpson & Castelltort, 2006; Li & Duffy, 2011; Liu & Beljadid, 2017), where finite volume methods are used to discretize shallow water equations. These equations are particularly useful in modeling of water movement across surfaces near the boundaries of lakes, rivers, and coastal areas where vertical accelerations are negligible compared to horizontal ones. However, these approaches share a lack of consideration for factors that influence realistic physical characteristics of fluids, such as volume, velocity, pressure, and viscosity. Landscape evolution models (e.g., CEASAR LISFLOOD-FP) are hybrid 2D hydrodynamic flow models generally sensitive to grid resolution of input data (e.g., rainfall), struggle with complex flow shapes (e.g., meandering rivers), and may be difficult for a model user to use and tune parameters (Coulthard & Skinner, 2016; Skinner & Coulthard, 2023)

As a result, traditional GIS methods oversimplify the flow dynamics, leading to inaccuracies in predicting flow paths (Goodchild & Glennon, 2008), especially in heterogeneous landscapes with a wide range of elevation differences or during extreme weather events. As digital terrain data becomes more accessible and our discipline more diverse, research has shifted towards using other approaches with the goal of achieving models that better describe complex behaviors of water (Clark et al., 2017).

2.3.4.2 Eulerian to Lagrangian

Eulerian-based models are views where space is discretized into a uniform grid with every point in space possessing some value, even if it is empty (Dimov et al., 1996). This is the view used by most traditional 2D models. Lagrangian-based models are views where the focus is on movement of objects through space as discrete entities (Lee & Sirjani, 2018). Lagrangian-based models represent a different paradigm in modelling flow path, in contrast to Eulerian views which have been favored historically due to their compatibility with 2D numerical models (Price, 2006). Lagrangian-based flow path models are less common than Eulerian approaches largely due to their higher computational and implementation complexity. Despite this, Frawley et al. (2010) proved Lagrangian methods can be more accurate for complex water behavior such as slurry flows, using measured data for validation. More recently, Carretier et al. (2023) created a landscape evolution model by tracking the movement and accumulation of radioactive isotopes within sediment using a Lagrangian approach. The conclusions show that while computationally expensive, Lagrangian models are capable of handling more complex scenarios than their Eulerian counterpart. Therefore, the choice of Eulerian views to store and analyze data historically are mostly due to convenience and may not be a realistic reflection of human perception, comparison, and decision (Pontius et al., 2007). A key advantage of Lagrangian models is that their realism scales with the number of simulated particles, allowing for adaptability and relevance across a wide range of domains (e.g., fluid, gas, fire simulations). With the increasing availability and computation power of hardware, a transition to development of more Lagrangian models to represent nature is becoming feasible and realistic.

2.3.4.3 Computer Graphics

Hybrid approaches emerge in the field of computer graphics to offer a more nuanced representation of hydrological processes through mixing Eulerian and Lagrangian views (e.g., Pazouki et al., 2014; Zhou et al., 2024). By accounting for gravity, pressure, and energy dissipation, these models provide a more realistic depiction of water movement. For instance,

hydrodynamic models can capture the formation of flow vortices, changes in flow velocity due to varying channel slopes, and the impact of obstacles and vegetation on flow patterns.

Many methods are based in physical principles and concerned with accurate rendering in 3D. Müller et al. (2007) describe an approach to solving particle positions by prioritizing constraint satisfaction over force-based calculations (Position Based Dynamics). Within each frame, constraints such as collision avoidance, positional correction, and volume preservation are iteratively applied to individual objects. This iterative process prioritizes stability and robustness, producing visually plausible results even under conditions that typically lead to numerical instability in force-driven models. Due to the robustness of this approach, it is often applied as a base logic in many collision simulations, including default implementations in game engines.

A systemized literature review was conducted on Scopus to explore the intersection between graphics computing and water-based erosion. The focus is on finding literature which apply computer graphics techniques to hydrological and landscape models within a GIScience context, preferably on realistic terrain and datasets. The search query was formulated as:

TITLE-ABS-KEY (
("erosion model" OR "erosion simulation" OR "hydrological erosion" OR
"sediment transport" OR "water erosion" OR "geomorphological modeling") AND
("computer graphics" OR "procedural terrain" OR
"terrain simulation" OR "virtual landscape") AND
("validation" OR "accuracy assessment" OR
"remote sensing" OR "LiDAR" OR "satellite"))

The search yielded 21 results. Of these, only 3 were highly relevant to the application of computer graphic techniques for water-based erosion. Shen et al. (2003) introduced a voxel-based erosion process on real terrain data using particles, but the results were not validated against real data or compared to other similar workflows. Additionally, the model

implementation details were not disclosed, and its development using Visual C++ and OpenGL demands advanced expertise in that technical environment. Dazzi et al. (2019) implements an erosion model using GPU-accelerated water simulation in 2D but is focused on levee breaches instead of soil erosion. Amicarelli et al. (2020) presents an open-source simulation framework capable of handling complex flow by using modern CFD techniques. However, this model has a steep learning curve requiring knowledge in Fortran to use and relies on extended periods of offline batch simulation, making validation difficult.

The remainder of the results were not relevant because they primarily relied on static numerical models (e.g., Lugina et al., 2020), were not at the watershed scale (e.g., Muin et al., 2016), 2D (e.g., Her et al., 2021) or focused on visualization of data than computation of results (e.g., Jiang et al., 2007).

Recent advancements make further strides in simulating water realistically in 3D space, with an interest in simulating a large quantity of particles that exhibit fluid mechanics. Smoothed Particle Hydrodynamics (SPH) is a technique for solving intra-particle interactions by representing a fluid or continuous medium as a collection of particles, each carrying properties such as mass, density, and velocity (Pazouki et al., 2014). These particles interact with their neighbors using a smoothing kernel function, which averages their contributions to approximate physical quantities such as pressure and forces at any point based on their relative distance. These calculations are performed over every particle on every frame in a loop to update their velocity and position, which enables the simulation of complex fluid behaviors such as splashing, waves, or surface tension (Goswami et al., 2010). The computation cost of this technique increases with more complex behaviors on each particle as well as particle count and is usually applied on the Graphics Processing Unit (GPU). Thus, though computationally intensive, lays a robust foundation for developing sophisticated flow path models.

2.3.4.4 Validation of Computer Graphics Models

A systemized literature review was done on Scopus and the query was formulated as:

TITLE-ABS-KEY (
("hydrological erosion" OR "sediment transport" OR "fluvial erosion" OR "flow path") AND
("3D" OR "Three-dimensional") AND
("procedural modelling" OR "procedural generation" OR
"simulation" OR "computer graphics") AND
("validation" OR "accuracy assessment" OR "GIS" OR "model verification")) AND
(LIMIT-TO (SUBJAREA,"ENVI"))

The search yielded 69 results. Of these, only 3 were highly relevant to the application of computer graphic techniques for water-based erosion. Shen et al. (2003) introduced a voxel-based erosion process on real terrain data using particles, but the results were not validated against real data or compared to other similar workflows. Additionally, the model implementation details were not disclosed, and its development using Visual C++ and OpenGL demands advanced expertise in that technical environment. Dazzi et al. (2019) implements an erosion model using GPU-accelerated water simulation in 2D but is focused on levee breaches instead of soil erosion. Amicarelli et al. (2020) presents an open-source simulation framework capable of handling complex flow by using modern computational fluid dynamics techniques. However, this model has a steep learning curve requiring knowledge in Fortran to use and relies on extended periods of offline batch simulation, making validation difficult.

The remainder of the results were not relevant because they primarily relied on static numerical models (e.g., Lugina et al., 2020), were not at the watershed scale (e.g., Muin et al., 2016), or focused on visualization of data than computation of results (e.g., Jiang et al., 2007). This shows that there is a limited amount of research on the intersecting domains of environment modelling, computer graphics, and assessment or validation of models.

2.3.5 Gap Synthesis

The literature review of game engine usage in environment science shows most research applied to the tool in a visualization context, yet there are many challenges in a modelling context which game engines are capable of addressing in GIScience. The review shows that game engines can be applied to support modelling efforts, but there is not a single peer-reviewed paper that focuses on the intersection between dynamic modelling capabilities of game engines and its integration with GIScience frameworks. This points to a literature gap between these disciplines which impedes collaborative and multi-disciplinary research efforts, of which can benefit from the advanced simulations one can build in a game engine. For example, there was no literature that combines hydrodynamically rigorous flow path solvers with real world elevation data, and export of simulation data back into GIS for spatial analysis. Therefore, a gap in research exists on the methodologies to leverage modern game engines for their rendering, physics computation, and scripting architectures to build dynamic flow path models for realistic 3D environments, as well as exporting results of the model into a GIS environment for quantitative analysis and decision support.

Chapter 3. Bridging Disciplines

3.1 Introduction

The development of complex simulation models presents ongoing challenges in the field of Geographic Information System (GIS). Specifically, academia, and to a lesser extent industry, have struggled with the representation of time in spatial data (e.g., Erwig et al., 1999), spatial analyses (e.g., Peuquet, 1994), and spatial process models (e.g., Yuan, 2001, Section IV). For example, the predominant spatial data models (i.e., raster and vector) typically comprise a single point in time of information, such as land cover derived from a satellite image (e.g., Longley et al., 2015), or from multiple acquisition efforts assembled as a compilation to provide broader spatial coverage (e.g., Alberta Environment and Parks, 2020). Static spatial datasets offer rich data representation but lack built-in mechanisms for tracking temporal change or collaborative edits. By contrast, versioning systems are widely available and used with ease for word processor files (e.g., via Microsoft Teams) and computer code (e.g., via GitHub) among other examples, yet their use with spatial data is limited to spatial databases (e.g., Postgres; Obe & Hsu, 2015; Oracle Corporation, n.d.) that are less accessible and have not been embraced by the majority of spatial data users (Vonk et al., 2005). While these limitations have constrained the creation and use of spatio-temporal data, the collection of most spatial data are not event driven and their temporal resolution does not coincide with time intervals associated with human decisions and their outcomes or the occurrence of natural processes and their impacts (Peuquet, 2002).

Despite the aforementioned challenges, we find ourselves in a time where there is an unprecedented growth in the rate of data collection. Specifically, novel internet-of-things technologies (e.g., Rahman & Asyhari, 2019), harvesting of social media posts (e.g., Stefanidis et al., 2013), mobile data tracking (e.g., Bell, 2017), and the on-demand and frequent use of remotely piloted aircraft (e.g., Noor et al., 2018) have significantly increased the temporal, and

in some cases spatial, resolution of spatial data collection, which more closely coincide with the time-interval at which human decisions and natural processes operate. However, these data still do not overcome the decades-old problem of linking patterns to process (e.g., Kemp, 1996; National Research Council, 1998).

In response to the division between static snapshots and changing phenomena, the computational representation of human decisions and natural processes in the GIS industry are increasingly achieved using spatial process models (e.g., Erwig et al., 1999; An, 2012). When calibrated and validated, spatial process models can provide a form of generative interpolation, whereby they simulate the process and its effects on spatial phenomena over time to generate synthetic data to fill data gaps between real-world observations (Shibasaki & Huang, 2001). In addition, they can be used as computational laboratories for experimentation that can push not only our understanding of complex systems comprising feedbacks, non-linearity, heterogeneity, and aggregation (e.g., Fatichi et al., 2016; Zellner, 2008); but, also for testing the ability of sampling to represent population dynamics (e.g., Buckland et al., 2016), capturing human decision-making (e.g., Evans et al., 2006; Zellner et al., 2014), and evaluate the influence of different perturbations to natural systems (e.g., Lippe et al., 2019). Simulation models synthesize our current understanding and provide a medium for thought experiments that forward scientific theory and their applications (Rounsevell et al. 2012).

Spatial simulation has been gradually adopted for scientific inquiry over the past three decades (e.g., Priya & Shibasaki, 2001). The majority of these efforts have been 2D in nature (e.g., Ahmad & Simonovic, 2004), which coincides with the Geographers and Planners affinity for maps and spatial thinking (Uhlenwinkel, 2013), but lacks the visual realism (e.g., Roberts, 2012) and real-world physics (e.g., gravity, acceleration, and force; Murdock, 1894; Beven, 1989) required to validate spatial process models at the scales of individual decision-making. Models often involve trade-offs in their design, the trade-offs between simplicity and complexity (Manson, 2001) shift from simple, deterministic models to more intricate, emergent systems with spatial extent, accuracy requirements, and human participation.

Harris et al. (2017) warns that there are dangers of over-reliance on simplistic representations, and that geocomputation possesses potential to overcome these dangers and empower research communities by making use of the advanced computing power as well as Big Data. Hence, some of the limitations we face may be due to the lack of innovative revolutions in GISystems as an effective response to the recent developments of Big Data (Harris et al., 2017). This calls for a move towards 3D GIS research where attention is shifted towards better conceptualizing spatial, platial, and temporal information via realism and immersion in a way that is more familiar to our cognitive perception (Stoter & Zlatanova 2003). However, the concept of a 3D GIS must overcome multiple challenges pertaining to generating and querying 3D geo-objects, structuring, manipulation, and analysis techniques (Couclelis, 1992; Zlatanova et al., 2011). Thus, an appropriate tool should also be compatible for a spatial modelling platform working with a wide range of problems in Geographic Information Science (GIScience).

3.1.1 Research Gap

Many current GIS tools available in the market still lack the necessary flexibility required to address such complex data processing needs. Although Gahegan (2018) and subsequent studies have identified core challenges in GIS, their solutions often involve expanding GIS into hybrid platforms. However, the nature of the discussion seems to escape the essence of the problem - which is that while our GIS is small, it is enough for the problems it has set out to solve. In attempting to expand GIS, we risk creating systems that encompass domains beyond its conceptual foundations, reinventing functionalities that are commonplace in other tools, and are non-practical for research reproducibility (Yu et al., 2012; Zaragozí et al., 2020; Wang et al., 2021). Even more so, we should not limit our science to the narrowness of our tools and its analytical features. Instead, we should consider developing complementary frameworks that address challenges while maintaining interoperability with existing GIS infrastructure (Gahegan, 2018). This poses a significant challenge to GIS scientists who must develop their own tools to address specific research questions, which in turn can lead to reproducibility issues.

In recent years, one promising avenue is the exploration of unconventional tools such as game engines and alternative related tools for handling geospatial data, which have shown potential to overcome current challenges in GIScience. Game engines, originally designed for interactive and immersive experiences in video games, possess inherent capabilities that align with the requirements of spatio-temporal analysis (Helbig et al., 2022). This emerging trend is driven by a need for more engaging mediums that can also serve as a tool for performing alternative methods of analysis to support scientific discovery and experimentation. Notably, game engines exhibit the potential to generate synthetic or procedural data that closely resemble real-world scenarios, reducing reliance on statistical inference and data-driven analysis (Agarwal et al., 2023). Additionally, their capacity to handle large datasets and efficiently render realistic virtual environments using standardized toolsets can be effectively harnessed to create informative visualizations and simulate complex processes. This potential integration preserves the integrity and efficacy of purpose-built tools while allowing for innovation in areas where current systems fall short.

Despite their potential, there is little to no knowledge and standardization, whether in methodology or terminology in this emerging area of research using game engines for modelling natural processes. The goal of this chapter is to effectively structure a broader perspective on how game engines can respond to existing limitations.

First, it is essential to review the traditional theories and frameworks which serve as foundations for the current state of GIScience. Second, identify the specific areas where current GIS face challenges and uncover recurrent themes within these difficulties. To achieve this, a focused literature review was conducted, specifically to understand the challenges for GIS in dealing with temporal dynamics, process-based models, visual representations, user interactions, and validation of these models. Third, a conceptual framework was proposed which articulates the potential for game engines to contribute in response, aligning to each theme of challenges. By bridging these gaps, future research can advance methodologies by

integrating standardized toolsets in game engines with the robust, specialized foundations of GIS.

3.2 Conceptual Framework

To effectively harness game engines, four interrelated conceptual pillars are discussed in correspondence to the challenges they address in Chapter 2. Each pillar represents an aspect of game engines that can offer new paradigms, pathways and act as a platform to integrate solutions into GIScience and overcome the challenges:

- “*Dynamics*” as a structure for implementing temporal and processes-based approaches.
- “*Representation*” as the medium for multi-dimensional visualization.
- “*Interaction*” as the gateway for real-time multi-disciplinary experimentation.
- “*Validation*” as a platform for open geoscience.

3.2.1 Dynamic Processes

The interconnected challenges of the Modifiable Temporal Unit Problem (MTUP) demand dynamic tools that inherently embrace temporality, spatial dynamism, and scalability. The adoption of a temporal GIS data model using a game engine offers opportunities to incorporate a methodology that applies a contextual theory of understanding, as described by de Regt (2017) on the underlying processes, and develop intelligible models that effectively manage uncertainties inherent in incomplete representations (Longley, 2004).

To overcome these limitations in a game engine, integrating multi-disciplinary perspectives, particularly from computer science, computational physics and graphics discipline is crucial. As advocated by Harris et al. (2017) and Gahegan (2018), methods such as parallel optimization represent significant opportunities for advancing GIScience and enhancing GIS capacity to analyze and interpret spatial data in novel ways.

In order to effectively embed temporal dynamics into game-engine models, we must first explore how time itself is measured and represented, examine how agents and systems can incorporate temporal dynamics, and finally consider architectural strategies that preserve modularity across time-driven components.

3.2.1.1 Integrating Time

Within a game engine, time can be measured in the smallest unit of a rendered frame, and the progression between frames, referred to as delta time, drives the update of dynamic processes. This mechanism ensures that physics calculations (such as those governing fluid dynamics, object movements, or environmental changes) remain consistent, regardless of hardware differences, by scaling the progression of time-dependent simulations. Processes that need to be calculated to derive their state in the next frame are carried out within a simulation loop, which describes this constant advancement of frames as a measure of simulation time. This approach is especially relevant if the simulation relies on a Graphics Processing Unit (GPU)-based method to render the result of a calculation and allows game engines to offer a powerful environment for dynamically visualizing and modeling processes based on their moment-to-moment logic.

Temporal challenges can be addressed by providing flexible and dynamic control over temporal scales within simulation loops, where temporal scale is described by the number of frames in a second, calibrated by the length of delta time to match real time. Higher frame rates enable the display and processing of finer temporal resolution in datasets, which produce shorter delta time.

This parameter provides users with the option to interactively adjust time steps and experiment with various temporal resolutions, thereby allowing an empirical evaluation of how temporal aggregation can influence simulation outcomes. This method of exploration can facilitate the determination of the appropriate temporal scales for a specific analysis, partially mitigating the ambiguity introduced by the MTUP. Finer resolution synthetic data can also be generated

through higher frame rates, as it implies shorter delta time and all forms of physical interpolation must be normalized by this value to achieve temporally consistent, device-agnostic simulations. Time, in reality, is not discrete, but through delta time game engines can provide a basis for real-time manipulation of temporal data and the exploration of how different time scales impact the aggregation of micro and macro phenomena.

3.2.1.2 Integrating Models

Game engines excel at integrating with mathematical models that can be discretized into time steps, especially if complex calculations are required at every frame with respect to delta time (e.g., Jong et al., 2014). There is a plethora of advanced simulation techniques from specialized fields that research ways to simulate and render natural elements. For example, water is a highly researched area of simulation to produce realistic behaving fluids in computer graphics, namely the field of Computational Fluid Dynamics (CFD). Traditional methods of fluid simulation are modeled with the Navier-Stokes Equation (NSE), which is a partial differential system of equations used to calculate the movement of objects in continuous fields such as water, making them computationally feasible in a simulation (e.g., Chen & Lobo, 1995). These continuous domains must be discretized into grids or particles, allowing the mechanics to be captured by programming the procedural interaction between particles and their movement in a game engine at every frame. In this approach, a hydrological model can be represented by a set of particles, each with attributes of position, velocity, and mass.

These mathematical equations are categorized as Partial Differential Equations (PDEs) that commonly describe how the velocity field of a fluid evolves over time and space, capturing a relatively complete spectrum of motion, including viscosity, pressure, acceleration and conserve momentum at each timestep (Fefferman et al., 2018). These equations can be generalized to model many natural features that behave like a continuum (e.g., airflow, heat conduction, and movement of crowds in a confined space). Moreover, they are highly compatible for integration into game engines, due to incorporating time as a core component of describing Dynamics. When these equations are implemented within a game engine, they

allow the simulation of fluid behavior in a way that is both realistic and interactive. Researchers and practitioners can manipulate parameters in real-time (defined as above 24 frames per second), observe the resulting dynamic transitions, and evaluate how closely the simulated dynamics match the expected behavior. This focus on timesteps can describe natural processes in terms of emergent interactions that occur at every step of simulation, where time drives the formation of patterns.

Game engines excel at defining and managing the rules of interaction between different actors in a simulation. For example, the engine could explicitly model how water particles interact with each other and the terrain, based on programmable physical parameters. This capacity to simulate interactions based on foundational principles positions game engines as a structured environment for creating process-based models with a high degree of flexibility. Unlike empirical models derived from statistical correlations, a game-engine based model would simulate the sequence of physical events and their causal relationships, offering a more mechanistic and potentially more accurate representation of how phenomena like erosion unfold dynamically over space and time. This makes them suitable for the rule-based simulation environments, thereby offering a dynamic, interactive, and potentially more intuitive way to explore and understand complex environmental systems.

3.2.1.3 Structure and Modularity

Game engines are flexible and modular, capable of adapting to changing requirements and future expansions. They embody many of the best practices in computer sciences and can apply the Object-Oriented Programming (OOP) paradigm through its rich ecosystem of Application Programming Interfaces (APIs) and built-in systems. As noted by Gregory (2009), most contemporary games utilize an OOP language due to their agent-based nature (e.g., C++). This programming paradigm stores data in individual agents as objects, each with its own distinct attributes and behavioral rules, facilitating the simulation of complex interactions. Thus, game engines allow a high degree of flexibility in what can be contained in objects,

including multiple other agents, which underscores the software's adaptability to allow complex compositions of objects dynamically simulating in environments.

The flexibility of game engines allows it to construct a wide range of complex simulations without compromising modularity. For example, a simulation project consists of merging multiple independent systems (rendering, physics, input), representing agents, environmental factors, data processors, or visualization elements that interface and communicate effectively with one another through a standardized set of APIs. By incorporating these well-defined interfaces, models are modular, and new behaviors can be integrated without affecting existing systems.

To facilitate communication between systems in game engines, common data structures are offered, which can be creatively interpreted to fit the context as needed. These data structures act as fundamental building blocks and come with built-in standards for input and output data types, often defining how information is transferred with other objects in a sequence (e.g., triggering events, accessing data). The use of appropriate data types in combination with OOP practices enables game engines to be an accessible platform for developing extensible simulations by cleanly separating data, behavior, and control logic into modular units. This modularity lets researchers plug in sub models (e.g., different precipitation patterns) without affecting other parts of the system.

3.2.1.4 Entity Component System

Researchers should take advantage of established design patterns that exist in modern engines, such as the Entity-Component-System (ECS) pattern (Wiebusch & Latoschik, 2015; Redmond et al., n.d.). Particle systems in game engines are often implemented using ECS principles. Game engines can apply these principles by emphasizing on composition of actors in a model over defining their inheritance relationships, encouraging the separation of data, behaviors, and governing logic of an object. In this framework, *Entities* represent individual objects (e.g., a tree, wildlife, or a soil patch), *Components* store their specific properties (e.g.,

location, moisture level, or growth rate), and *Systems* define the rules and processes that dictate how these entities relate, interact and evolve over time (e.g., erosion, terra-formation, gravity). This separation creates a modular and flexible structure, allowing developers to add, remove, or adjust components and systems without overhauling the entire model. For example, ECS can be used as a conceptual model to design the architecture of simulations for essential and well-established dynamics between natural features using existing mathematical or physical models in specialized domains.

Entity is the container of data, holding agents and their position in space, such as a parcel of water. *Components* drive their physical interactions with the environment and each other, such as gravity and collisions. *Systems* govern their spawn and lifecycle, ensuring that entities interact correctly with other *Systems* in the environment. In implementing this pattern for representing natural processes, *Components* can be modular, allowing *Entities* to represent various agents. In this context, game engines represent a path forward to better understanding environment dynamics through separation of concerns, and evolving compatibility of the modelling between the inherent physical qualities of environmental features.

This paradigm enables creative re-interpretations of *Entities* in agent-based models. For example, one could allow entire study areas to be treated conceptually as agents themselves. These environmental features are typically passive or stable (like patches of soil) but enables storing their inherent properties as complex models such as erosion susceptibility (e.g., Shen et al., 2004; Shi et al., 2013). Simulations can be enriched through the inclusion of appropriate *Components* integrated to respond to changes in the environment alongside the discrete actors and agents that are usually the focus.

The *Component* should encapsulate geo-processes through precise digitization of physical behavior and strive to be modular so it can fit within the context of many other models. Through representing processes as *Components*, game engines are suited to produce simulations that abstract reality to a degree. The specific scope and fidelity of this abstraction depends on the spatial scale adopted to study the phenomena (e.g., land parcel scale, global

scale), and the degree of realism to capture, with the goal of constructing a virtual environment that reflects human cognitive perception more closely. For example, gravity is a *Component* that applies a downward velocity to *Entities* on every frame. This velocity accumulates with each passing frame, simulating the effect of acceleration.

Particle systems in a game engine fulfill the *System* role of the ECS design pattern, as it iterates over all *Entities*, applying the rules of their *Components* on every frame. Researchers may use this pattern to build Agent-Based Models (ABMs) in game engines, define cause-and-effect relationships between *Systems*, and develop databases of modular processes. In addition, particle systems are directly integrated with GPU-based approaches and allow users to access parallel computing techniques in this embedded platform.

The ECS pattern can produce simulation models with greater explanatory power and flexibility, particularly when faced with novel scenarios or changing conditions.

3.2.1.5 Principles of Dynamics

Models in game engines should implement formal process representations, often utilizing mathematical frameworks such as PDEs, agent-based rules, or other relevant numerical methods, to compute the transition of states with respect to time. Process evolution (e.g., physical movement of objects) must adhere to simulation time integrity, managed via mechanisms like delta time, ensuring computational consistency and reproducibility across diverse execution environments. The equations move static snapshots towards a high-fidelity process-based representation of system evolution, enabling analysis of short- and long-term behaviors as well as complex temporal dependencies.

The computational power of game engines allows for the explicit modeling of the processes that generate spatial patterns. Traditional empirical models rely on assuming correlation in the data collected from remote sensing processes, which merely reflect the appearance of phenomena, but do not offer the scientific explanation nor does it expose its underlying dynamics. However, in Meentemeyer's (1989) perspective, the role of the geographer is to take

notice of these phenomena and dynamic processes and explore the geographical forces which brought about its appearance. As Dr. Derek T. Robinson (2025), Geography professor at the University of Waterloo, aptly stated, "*Humans choose, but physics does not*". As such, game engines should build models that borrow the laws of physics to act as the interpolator between static snapshots over time, define the rules of interaction and transitions, and present a process-based model for the representation of complex dynamic processes.

The creation of standardized terms, interfaces, design practices, and documentation for dynamic models would grow existing communities that already use game engines for research (Rehn, n.d.), and establish a common language with the broader scientific community. The framework would then be able to contribute to an engine-agnostic model, such that assets and workflows can be shared regardless of the engine used.

3.2.2 Representation, Realism, and Interpretation

Representation and manipulation techniques of 3D data has advanced in the engineering, computer graphics and computer vision fields for point clouds (e.g., Huang et al., 2022; Tesema et al., 2023; Lv et al., 2024), meshes (e.g., Dai & Nießner, 2018), and voxels (e.g., Laine & Karras, 2010; Li et al., 2022). Notably, when state-of-the-art data manipulation and visualization techniques are developed, game engines are typically first adopters, as it aligns with their goals of creating immersive, realistic, and interactive virtual environments in commercial games.

The elements of the natural world do not conform to traditional representations of data (e.g., vector and raster) (Pontius, 2007). Therefore, applying advanced computer graphics techniques can offer alternative representations of data for GIS workflows. The modular nature of modern game engines allows researchers to leverage these tools without necessarily adopting the entire game development pipeline. A cross-disciplinary approach can accelerate the development of geospatial applications by building upon the substantial engineering investments already made

in the gaming industry, reducing development time and providing access to optimization techniques that might otherwise be beyond the scope of specialized GIS research projects.

Game engines are positioned at the crossroads of quantitative and qualitative methods, enabling visualization that can be customized to provide accurate representations of the dimensionality in data, as well as recreating realistic environments that align with human perception. This integration is supported by various 3D data types, state-of-the-art rendering pipelines, a flexible scene compositor, and native or external engine constructs.

3.2.2.1 Data Types

To achieve representations of the natural world in a way that is closer to human cognition, the movement to 3D representations enables us to better capture and instill these geographical elements to the data being recorded. In game engines, there are three popular representations of 3D data: Point Clouds, Mesh, and Voxels (Prytuła, 2024). Each representation possesses their own specialization and domain-specific appropriateness, varying in terms of how they are obtained (e.g., collected, modelled, or generated) and handled (e.g., processing, analysis, and visualization). In addition, they are conceptually equal 3D counterparts to existing 2D methods of representing data.

3.2.2.2 Graphics Simulation

Game engines adopt state-of-the-art computer graphics algorithms by iteratively integrating new frameworks to work with existing pipelines, packaging complex functionality into accessible API calls for users in each new engine version (Tatarchuk, 2009). This pioneering role makes game engines valuable technological platforms that often implement cutting-edge rendering techniques, physics simulations, and data processing algorithms before they become mainstream in other domains. As such, researchers who benefit from the use of these representation techniques may also benefit from using relevant components of game engines (e.g., particle systems, rendering, lighting), repurposing them to support a similar

workflow for visualization tasks. For example, particle systems have been used to build Boid models (Reynolds, 1987; Xueying et al., 2017), simulate liquid and gas (Kriushichev et al., 2020), spread of fire (Gedlaman, 2024), physics systems to simulate earthquakes (Sun et al., 2024), and more.

3.2.2.3 Engine Constructs

There are conceptually equal counterparts in 3D data representations to 2D. Mesh representations are built from vertices, edges, and faces, which can be interpreted as a 3D representation of vector and polygon features, since both are sets of data defined from the connection of points in space. The advantage of meshes are in their capacity to encapsulate complex geometries with precision, making them the most commonly used representation in game engines and related pipelines due to their compatibility with animation, rendering techniques and physics simulations (Mount, 2018). Rather than being collected directly from sensors, meshes are typically built by hand in modeling software such as Blender or Autodesk Maya. In many applications, the goal is to use point cloud data to reconstruct a 3D environment, essentially creating a mesh that transforms the color and position information of points into closed geometries with well-defined boundaries (Jianning Wang et al., 2005).

Voxels (volumetric pixels) offer a 3D representation by dividing space into a grid of discrete cubic units. Every unit of a voxel can store properties such as density, color, or material type, with an added spatial dimension. The advantage of this representation shares benefits with its 2D counterpart. As a result of the uniform distribution of space, each voxel can be easily indexed, operations in a voxel grid can be parallelized, leading to efficient memory usage and caching (Kaufman & Bakalash, 1988; Aleksandrov et al., 2021). It is immediately compatible with many numerical models as they typically assume a uniform grid. It can be an intuitive method to represent natural phenomena, such as atmospheric conditions, soil layers, or complex cave structures, whereas point clouds and meshes may struggle to (e.g., Kaufman & Mueller, 2005). Thus, voxels are powerful in run-time environments such as a game engine for simulating dynamic systems, such as destructible environments, erosion processes, or fluid

flows, due to their ability to modify 3D shapes and represent internal compositions rather than surfaces.

Using these engine constructs, a 3D geo-digitization process can be conducted to reconstruct real environments, given high quality data. For example, a dynamic terrain can be represented with voxels for real-time modifications like erosion, while meshes add static elements such as rocks and trees for precise geometric detail. Water flow can be simulated using particle systems to depict realistic movement and terrain interactions, using the mesh as a collision surface. Materials and lighting enhance visual fidelity, creating an immersive and realistic representation of the natural world aligning with human cognition. Interactivity allows users to tweak variables and modify data in real-time, making the visualization a dynamic tool for exploration and analysis.

3.2.2.4 Principles of Representation

Producing real-time visualizations that are realistic enough to be convincing requires finding the middle ground between accuracy and speed. While physics engines in game engines are not meant to be scientifically perfect, their results when visualized are close enough for human cognition and believable enough in the scope we hope to model (e.g., Lee et al., 2020). We should use game engines to address these gaps in modelling because it allows researchers to move past the implementation of specialized topics, such as the Euler-Lagrange equation or Newtonian mechanics just to model where an object would move to over time (Craighead et al., 2007).

Models within game engines should aim to design integrated geo-computational workflows. Specifically, workflows should leverage the tight integration of game engines with their visualization systems, treating the visual presentation as an intrinsic component of the spatial processing workflow. Bekkema (2016) indicates that GIS is often separated from processing, but game engines offer a potential pathway to re-establish the loose integration between modelling and visualization workflows in traditional GIS. Game engines often possess

extensible visualization architectures that are closely tied to the existing objects in the environment and could provide a framework for a similar integration scheme. Effective implementation requires the use, combination, and transformation of appropriate 3D geospatial formats, as well as the development of new techniques to visualize and creatively present the complex data that is generated from simulations.

3.2.3 Interaction

The integration of game engines into geocomputation and geo-visualization holds transformative potential for model development and spatial data analysis by addressing multiple critical challenges (Fitzpatrick & Hedley, 2024). Their capacity to create intuitive, interactive 3D interfaces lowers the programming barriers for domain experts, enabling users to build emergent visualizations dynamically without deep coding expertise. Furthermore, game engines facilitate a controlled space for continuous feedback loops between simulation and experimentation, multi-platform deployment, intuitive and standardized controls, allowing for the real-time refinement of models and promoting a more exploratory, participatory approach to understanding spatial and temporal processes.

3.2.3.1 Exploration

Interactive approaches allow properties of emergence in simulation, driving exploration of fine geographic mechanics that should govern physical behaviors consistent across all scales. This allows representation of a dynamic responsive system with an accurate physically based model (e.g., Hajigholizadeh et al., 2018). For example, once the mechanics of water and soil interactions are captured, this model could demonstrate different types of erosion based on a user selected type of soil and fluid, exposing the parameters of the model.

Game engines allow users to adjust parameters (e.g., rainfall intensity, soil type, slope angle) to see how they affect simulation results in real-time. This flexibility may enable hydrologists

(studying water flow) and agricultural experts (tracking soil loss) to obtain insights from the same simulation, eliminating the need for separate models.

3.2.3.2 Experimentation

The interactive nature of game engines facilitates experimentation with parameters and variables in ways that static models cannot. As Droll & Söbke (2021) suggests, additional modifications (mods) are evidence of the possibility to bring realism into entertainment games. Mods often use similar tools as game engines to alter simulations and imply the possibilities for users to experiment and surpass limitations of models built in a game engine. Developers and end users can modify input values and environment objects in real-time to immediately observe the consequences, creating a dynamic feedback loop that enhances understanding of complex systems, as well as assist with debugging during development. The most basic modifications involve changing position, rotation, and scale of objects in the environment. More advanced modifications may involve implementing emergent behavior or systems. This capability is particularly valuable when investigating hypothetical scenarios or exploring the implications of different management strategies. For instance, land managers could interact with a terrain model to test various erosion control measures, observing their effectiveness under different rainfall conditions before implementing them in the real world. In voxel representations, this translates to sculpting and levelling entire landscapes, acting as a controlled space for digital geomorphology. Further, game engines could be used to set up a laboratory for determining where the measurement and sampling points should be, as well as how many sampling points should be taken before taking manual measurements. Virtual laboratories present an alternative approach of studying physical geography, taking a digital and interdisciplinary approach as described by O'Sullivan (2015) and Massey (2004).

3.2.3.3 Participation

Game engines can bridge the gap between scientific modeling and public engagement. Traditional GIS outputs often remain inaccessible to non-experts due to their technical nature.

In contrast, interactive visualizations created using game engines can transform complex data into intuitive, explorable experiences that communicate scientific findings to broader audiences (Cristie & Berger, 2017; Harrap et al., 2019). This democratization of information aligns with the growing emphasis on science communication and public participation in environmental decision-making. For example, multiple users can simultaneously interact with the same simulation, connecting to a hosted network server, visually see and interact with each other and experience an authentic and immersive exploration of the virtual environment or dataset. This collaborative capability is particularly relevant in addressing complex challenges that span various domains of expertise, where interactions facilitate a large portion of the communication of knowledge.

Another form of participation is opening the model to the wider scientific community to create changes. Model adjustments and development may follow open-source approaches, such as hosting it on GitHub for others to fork, providing feedback, and suggest continuous development directions (e.g., Crystal-Ornelas et al., 2021). There are also a variety of other methods to share models built with game engines, such as transferring the project files or making a build, which offer flexible deployment options. Most engines provide built-in export tools and standardized APIs that support platform-agnostic development (e.g., web, mobile). There are an abundance of tutorials and guidelines for building an application targeted to Windows systems, Mac, a web browser, a mobile device, or emerging Virtual Reality (VR) platforms. This multi-platform compatibility enables interactive geospatial models to be shared widely and used in numerous contexts, ensuring that data and dynamic simulations are accessible to all users.

Game engines produce accessible virtual environments, with the ability to tailor the computational load and graphics quality to fit the client hardware capabilities. For example, the modeller can define several levels of rendering detail and various display options to provide a customized run-time environment, allowing users to select different graphical standards, such as low-, mid-, or high-quality rendering modes. This dynamic adjustment means that even if

models require intensive real-time simulation, the engine can adapt the level of detail and computational intensity to provide smooth performance on a range of devices, from high-end desktops to more modest machines (Luebke et al., 2012).

Game engines enable standardized input control schemes that are universal to many users, such as the widely adopted WASD keys for movement and mouse input for camera panning. In addition to these control schemes, a comprehensive graphic user interface library is offered that includes a wide array of standardized interactive elements such as buttons, sliders, text fields, and menus. This accelerates development by providing ready-to-use components for building custom interfaces and ensures consistent and intuitive user experience. Established input patterns and interface components help reduce the learning curve, which may aid with faster adoption in an academic context for building geospatial applications.

3.2.3.4 Principles of Interaction

This framework advocates incorporating user interaction into game engine models beyond a presentation layer, but as an integral part of the modeling and analysis workflow. This necessitates intuitive interfaces allowing both model developers and users to actively engage with the simulation through real-time parameter manipulation, direct modification of environmental conditions, and immersive spatiotemporal navigation within the dataset. Game engines possess exploratory modeling capabilities that facilitate hypothesis testing, sensitivity analysis, and scenario planning. Furthermore, interaction should support participatory validation, enabling users (including domain experts or stakeholders) to directly compare simulation outputs against empirical data, reference scenarios, or expert knowledge within the interactive environment (e.g., Anand et al., 2016). Through interaction, game engines should provide an environment for active data inquiry as well as qualitative and quantitative assessment of model validity.

3.2.4 Validation

Validating models built within game engines presents unique challenges that require methodological innovation. The novelty of employing game engines for scientific modeling implies validation frameworks currently remain largely undeveloped. The validation strategies required for game engines should share similarities to ones employed for ABMs, due to their similar functions for generating dynamic and emergent outcomes (e.g., Darvishi & Ahmadi, 2014). Effective validation strategies must account for the diverse output types that game engines can produce, ranging from visual representations and spatial analytics to temporal sequences and interactive simulations. Furthermore, traditional validation approaches often assume static model outputs, whereas game engine models are inherently dynamic, featuring emergent behaviors, real-time adjustments, and interactive elements that evolve throughout execution. This fundamental difference necessitates rethinking validation criteria to accommodate the temporal and interactive dimensions that distinguish game engine-based modeling from conventional GIS approaches.

Preliminary approaches to validation may involve using a reference model of traditional methodologies as a baseline of comparison, in combination with visual assessments by domain experts. Additionally, an alternate form of validation is ensuring the model can be replicated by other researchers, on different devices, and its ability to produce consistent results.

3.2.4.1 Reference Model

Validating models built within game engines demands methodologies that bridge the gap between static 2D outputs produced from traditional frameworks and the dynamic multi-dimensional outputs from game engines. One potential approach to compare and validate the differences between models is establishing a traditional model as reference to benchmark the game engine model. For example, a hydrology path model in traditional approaches represent flow paths as vector networks (lines) or raster layers (aggregated flow accumulation), derived from static algorithms (e.g., Eight Direction [D8]). The game engine model represents the flow

using a particle system with real-time dynamics and dynamics, and the path is represented by aggregating the changes in position over time at a specified temporal resolution (frame rate) during the dynamic simulation. To compare and analyze the output from these methods equally, a common data format should be used to ensure that the results of both models are consistently represented, spatially and temporally (e.g., Heath et al., 2009).

The reference model may be spatially static and possess no temporal dimensions or discrete-time outputs (e.g., annual flood risk maps), whereas game engines generate continuous time-dependent simulations (e.g., per frame positions). One approach to overcome this, using a flow path example, may be to project the 3D particle trajectories onto a 2D plane, rasterized into probability surfaces over a specified interval of simulation time, and then spatially aligned with traditional outcomes. However, the outputs from game engine models typically do not possess geographic metadata and thus are often not georeferenced, making it difficult to spatially align datasets. The lack of native geospatial metadata support in game engines requires the development of customized export pipelines to translate coordinate systems between the game engine and geographic data, or alternative methods of georeferencing. For example, if the projection, extent and coordinate data from the reference model is available (e.g., in a GIS-based environment) and the study area perfectly overlaps the simulation space in the game engine, then these metadata can be applied to anchor the outputs from the game engine for comparison in the same environment as the reference model.

3.2.4.2 Visual Assessment

Visual and interactive validation methods (e.g., Anand et al., 2016) leverages the inherent strengths of game engines in rendering realistic environments. This approach allows researchers to assess model outputs by observing simulated processes unfold as one would expect in the real world, it is particularly valuable for processes that exhibit distinctive spatio-temporal patterns such as river meandering, vegetation growth, and wildlife activity. Game engines render simulations in real-time 3D, allowing researchers to visually track agent interactions, spatial patterns, and system-level feedback. This facilitates a causal and

interactive method of discovery through observing parameter changes (e.g., resource depletion rates) cascades through the model. This aspect is further enhanced by Unity ECS or Data Layers in Unreal Engine (UE), which enable granular inspection of agent-state histories, environmental triggers, and event logs, simplifying documentation and reporting of mechanisms within the model. By enabling users to actively see the model behavior in real-time, it becomes easier to detect anomalies, validate theoretical predictions, and refine model assumptions. The intuitive nature of visual validation makes it accessible even to stakeholders without technical expertise, enabling broader participation in the validation process (Cristie & Berger, 2017; Kavouras et al., 2023).

Visual validation should assess how differences in methodology influence cause-and-effect relationships that do not exist in static models. For example, the inclusion of physics for a flow path algorithm may reveal differences in path between the reference model and the game engine output, caused by various topographical variations in 3D or microtopography that traditional methods simplify. The reference model acts as the basis for a control space to isolate the impact of fundamental methodological differences. The discrepancies can then be interpreted to gain insight into the flaws and areas of improvement for existing methodologies. Additionally, a participatory form of validation leverages the interactivity of game engines, encouraging domain experts to manipulate and experiment with different parameter sets (e.g., adjusting rainfall intensity) to observe feedback in real-time and compare the predictions with expectations derived from traditional models or to empirical data.

3.2.4.3 Reproducibility

Game engines inherently foster open science by prioritizing accessible, visual workflows over proprietary code. For example, components (e.g., terrain generation, particle systems) are designed into reusable assets or plugins with additional APIs that are added to existing visual scripting libraries (e.g., Blueprint in UE). These plugins contain modules which can act as self-contained units or logic within a simulation, carrying out independent but powerful operations without the user needing to understand their underlying implementation.

Modules serve as building blocks for creating reproducible and extensible workflows that assist with validating existing research. In an ecology context, one module might dictate that animals avoid human settlements, while another has them prioritize water sources. Researchers can swap these modules to compare how each rule set affects migration patterns and better understand the processes contributing to equifinality. Additionally, cloud-based sharing of entire project repositories (e.g., GitHub) ensures peers can access the model in its entirety and replicate experiments on a common platform.

The modules and features within a game engine are highly interoperable internally and externally, designed in a way that can promote collaborative research. The structure of projects in a game engine is composed of many data structures similar to a file system, and as such can be organized, read and accessed in multiple simulation scenarios (e.g., loading scenes). Additionally, creations within a game engine can also be interoperable externally through sharing model components, migrating data structures to other conventional formats. For example, plugins and frameworks exist (e.g., UE Datasmith) to ensure that innovations in one field can be readily accessed into other projects for knowledge mobilization and an adaptive modeling ecosystem.

Game engines can enhance the scrutability of research, since it has already built a large community of experts who use the tool for a variety of purposes, meaning the abundance of open and free education materials. This includes abstract data frameworks for structuring virtual environments and managing dynamic spatial objects that are compatible with existing game development techniques (e.g. design patterns, version control). Game engines offer plugins to standardize workflow processes, import pipelines and file formats (e.g., Perforce), but also retain enough flexibility to manipulate and create original representations of data, leaving space for research or experimentation. The flexibility of virtual environments provided by game engines reduces the necessity for researchers to develop specialized tools for each new project, thereby saving valuable time and resources. These existing materials may aid in

developing a reproducible research environment for GIScience, with game engines taking on the role of an assembly platform for various models.

3.2.4.4 Principles of Validation

Validation strategies must ensure that any spatial data are correctly georeferenced, and transformations, analyses, or visualizations should preserve data integrity when exported or imported. The output from game engines typically do not have geographical metadata associated with them. Therefore, spatial data from game engines must be cross-checked with real-world coordinates to confirm alignment, ideally using the original reference extents, if data was imported in reference to an existing GIS framework, to preserve spatial relationships.

The stability of models should be considered during validation, as the model should provide consistent results across multiple runs and under varying conditions. Game engines automatically incorporate many settings that may not be obvious in their influence on the outcomes of the model, which models should be aware of to not be influenced by specific configurations, whether hardware or software (e.g., framerate, GPU limits). Additionally, the performance of game engines with large geospatial datasets of all scales should be tested. Evaluating processing time, memory usage, and accuracy degradation as data size increases helps identify computational limits, ensuring the framework remains practical for scientific applications.

Transparent documentation of parameters, workflows, and engine-specific configurations (e.g., version, default values) are necessary to bridge the gap between game engine abstractions and GIScience rigor. This enables reproducibility and provides context for other researchers to validate and interpret model outputs against their own. Finally, direct comparisons against established GIS tools validate equivalence. Results from game engine models should be compared to those generated by tools such as ArcGIS or GDAL using a similar but representative traditional approach for alignment with domain standards.

3.3 Discussion

Complementing the challenges of realism and spatio-temporal dimensionality in historical and contemporary spatial process modelling, is the lack of building blocks and integrative software for quickly generating spatial process models. While the need to share model building blocks and components in addition to complete models has been raised in the past (e.g., Watson et al., 1998; Bell et al. 2015) and there are increasingly useful model repositories (e.g., Community Surface Dynamics Modeling System [CSDMS], Computational Modeling in the Social and Ecological Sciences [COMSES Net]), efforts for workflow replication developed (e.g., whole tale.org), and programming libraries and packages (e.g., via Python and R), these resources seem disparate, can be difficult to integrate (i.e., plug and play), and are typically not affiliated with a central repository that can coordinate their download, update, and connectivity. This disparity exemplifies the lack of an innovation platform as discussed by Gahegan (2018). Modellers and scientists should be able to download model components the same way we download phone applications (e.g., via Apple or Google Play) or image and 3D assets (e.g., via Fab) to avoid “reinventing the wheel” (e.g., Singleton & Arribas-Bel, 2019; Evans et al. 2013) and speed up scientific discovery.

Game engines can act as a platform to quickly assemble spatial process models. They enable this by using the ECS pattern to integrate existing models such as ones from CSDMS (Tucker et al., 2022), translating them into engine-native constructs (e.g., scene objects, physics simulations). Modellers can develop modular *Components* to fit a variety of scenarios and share them through the asset store, allowing other researchers to download and quickly assemble models by combining and linking processes through multiple *Systems*.

There is great emphasis in the field of environmental modelling on the need to connect predictive models to statistical data and make decisions about what aspects of the data should be treated as deterministic and what can be ignored (Clark 2005). This trade-off directly impacts the realism and complexity of the model produced, and there is a need for tools that can strike a balance between simplicity and complexity to ensure that the models capture the

intricacies of the real-world phenomena they aim to represent (e.g., Lobell & Asseng, 2017; Roberts et al., 2017). Furthermore, processing large amounts of temporal data requires high computational power and optimization of memory, and calls for a highly flexible and customizable approach to each dataset. This requires the tool to be capable of developing and implementing specialized algorithms that are optimized to focus on a particular aspect of the modelling process, as well as contextually making use of techniques such as parallel processing and memory management to minimize processing times and maximize efficiency.

Game engines can provide environments that address the needs of the decision-maker by acting as the filter for the modeller to distill meaningful processes and visualizations from a scientifically sound ABM driving dynamic simulation underneath. They provide a structured interface to negotiate the trade-off between simplicity and complexity, offering tools to selectively emphasize or omit data features based on contextual needs (e.g., via visibility toggles, parameter adjustments, and turning modules on or off). By abstracting low-level computational challenges, game engines empower modellers and decision-makers to concentrate on higher-order decisions, positioning it as a computational workhorse for interactive experimentation in a scientifically grounded sandbox.

3.4 Conclusion

This chapter outlines the conceptual perspectives of a calling in the GIS community for a tool that resembles the capabilities presented by a game engine. In response, game engines can address these challenges through *Dynamics, Representation, Interaction, and Validation*.

We should design game engine models with architectural modularity and semantic standards. Game engine models should be structured as adaptable, ECS systems, use PDEs with well-defined interfaces to facilitate tight coupling with rendering systems, compatibility with external data sources, analytical tools, and other simulation models. This approach would promote model reusability, supports collaborative development, enables robust validation through model coupling, interaction or comparison, and fosters an extensible ecosystem where

Components can be updated or replaced as research advances. Finally, comprehensive documentation should clearly store, organize, and articulate the structure of models, so that it may act as a modular block for research in other or similar domains.

This framework provided aims to equip model builders and researchers with a high-level understanding of the potential and paradigms that underscore the development and integration of environment science models in game engines. The alignment of game engines for environment science positions them not merely as visualization tools, but as laboratories for interrogating time as an active, manipulable dimension of geographic inquiry, a vision echoed by Gahegan's (2018) call for dynamic process representation and O'Sullivan's (2015) advocacy for digital experimentation in physical geography. By unifying the strengths of process-based modeling, interoperability, immersive visualization, and computational adaptability, game engines can integrate with models that challenge the static, snapshot-driven paradigms that have long constrained temporal GIScience.

Several research directions emerge from this framework. There is a need for standardized methodologies to integrate real-world geospatial data into game engine environments, ensuring accuracy and reliability in simulations. Consequently, deeper investigation into the computational feasibility of incorporating complex simulations, such as realistic fluid models, within these digital laboratories are required. There would need to be an examination on not only the technical challenges associated with simulating natural phenomena but also assessing the extent to which data can be seamlessly exported from game engine environments to compare them with traditional GIS methods. Finally, more work is needed to evaluate the stability of these simulations over iterative engine updates and explore the practical difficulties involved in developing, validating, and maintaining such high-fidelity models. Addressing these aspects are essential to ensure that the benefits of an interactive, dynamically evolving framework are fully realized while mitigating the risks of obsolescence and technical debt in model development.

Chapter 4. Case Study

4.1 Introduction

Modelling hydrological processes is crucial for the understanding and management of water resources. Predictive models, particularly for flow path mapping, serve as a foundation for subsequent research, such as studies on water quality and erosion patterns (dos Santos et al., 2023). The evolution of flow path mapping techniques has been driven by technological innovation and methodological refinement in the field of Geographic Information Science (GIScience) and geomorphology (Boonya-aroonnet et al., 2007). Initially, these flow mapping efforts relied heavily on manual interpretation of topographic maps and field surveys, which were foundational but labor-intensive methods prone to inaccuracies due to subjective interpretation and limitations in topographic data quality and availability (Usery, 2018). The introduction of Digital Elevation Models (DEMs), noted as early as the 1950s (Lakshmi & Yarrakula, 2019), marked a significant shift, providing digital terrain representations that enabled more precise and automated delineation of flow paths using algorithms such as the Eight Direction (D8) method and its variants, which calculates flow direction based on steepest descent from each cell in raster data. Further progress in the development of hardware for data collection came with the proliferation of remote sensing technologies. Unmanned Aerial Vehicle (UAV) surveys now provide high-resolution elevation data capable of capturing fine-scale topographic features (Meinen & Robinson, 2020), significantly enhancing the precision of derived flow paths and the hydrological models they inform.

While watershed modeling (which encompasses flow path mapping) is often closely associated with erosion modeling, the two fields traditionally focus on different primary processes and research questions. Watershed models typically emphasize simulating components of the hydrological cycle and water distribution across a catchment, often geared towards water resource management objectives such as assessing water quality and availability. In contrast, erosion models concentrate specifically on the detachment, transport, and deposition of soil

resulting from erosive forces like rainfall impact and runoff, often without simulating the full suite of underlying hydrological processes driving the water-movement itself, such as flow acceleration and influences from topology of the flow surface. This distinction isolates water erosion as a secondary effect from the primary movement of water across the landscape. Therefore, accurately modeling this dynamic water movement itself becomes crucial for fundamentally understanding and predicting subsequent processes.

However, despite technological advancements for enhanced collection of topographic data, there is a gap in researching methods of increasing realism and temporal reliability of flow path models (Kirchner, 2006; Sidle, 2021). Conventional Geographic Information System (GIS)-based methods, even when utilizing high-resolution DEMs derived from modern sensors, fundamentally operate on static representations of the landscape. This limits their ability to effectively capture the dynamic, time-varying nature of hydrological processes like surface runoff generation, flow path evolution during storm events, or gradual channel adjustments.

The research question is: How can game engines be used to bring graphics computing technologies to hydrological modelling? To bridge the gap between GIS approaches to hydrological modelling and the approaches in the computer graphics industry, a literature review will be conducted to situate the appropriateness of using a game engine to model hydrological processes.

4.1.1 Research Gap

Despite the enhanced simulation technologies, there is a lack of physics-based dynamic spatial models for GIS that focus on processes at the field scale and are in 3D (e.g., De Roo et al., 2000). There are significant challenges in implementing such models for the purpose of flow path mapping in a GIS context, as it requires substantial computational resources and technical expertise in numerical modeling and Computational Fluid Dynamics (CFD).

Additionally, most existing tools for CFD are often behind a commercial fee and are specialized software frameworks tailored to solve engineering challenges on a small scale (e.g., Raman & Dewang, 2018). Thus, one of the primary gaps in research lies in purposing these sophisticated simulations for GIS applications, which traditionally focus on static data analysis and visualization rather than fluid dynamics.

Game engines may address this gap by applying their advanced particle simulation frameworks, which are compatible with techniques in the field of CFD to represent realistic fluid behaviors. However, more documentation and studies on the application of game engines to simulate fluid dynamics is needed, as there are particularly few documentations for Unreal Engine (UE) despite its steep learning curve (Yudin et al., 2019), and workflows connecting game engine data to GIS environments are poorly researched. Thus, a deeper investigation is needed to explore the extent of the simulation capabilities in modern game engines to develop a novel methodology for modelling hydrological processes, demonstrating its potential to work with geospatial data, and how it may be integrated with existing GIS workflows.

4.1.2 Objectives

This study aims to evaluate the viability of incorporating game engines as a tool for flow path mapping workflows. The assessment of viability will focus on a combination of the four criteria by the FAIR principles (Wilkinson et al., 2016) in addition to accuracy, interpreted as:

Findability: The ability for game engines to catalogue, organize, and retrieve data based on metadata.

Accessibility: How simple are the features in UE for operational use by a universal audience when compared to established tools, such as ArcGIS. This includes affordability, such as licensing costs and hardware requirements associated with using a game engine compared to ArcGIS.

Interoperability: The extent of transparency that allows one to review, verify, or modify the processes of the tool in UE compared to ArcGIS.

Reusability: The generalizability of tools and models produced by a game engine that can be used, shared or extended by other users in the future.

Accuracy: How closely do the results in UE match realistic flow behavior, benchmarked against established models such as D8 in ArcGIS.

This chapter seeks to document the use of tools in UE for processing spatial data and generate outputs in formats that are applicable within conventional GIS frameworks. The viability will be assessed through creating a proof-of-concept flow path simulation model inside UE, then using its output to conduct a comparative analysis with traditional GIS methodologies. The specific goals of the model are to:

- Establish a dynamic environment where the terrain can be interactively modified using voxel-based representation of elevations derived from DEMs, preserving data integrity and detailed accuracy upon import.
- Implement Smoothed Particle Hydrodynamics (SPH) to enable realistic fluid mechanics and collision with the voxel terrain.
- Define parameters that influence output of the simulation, such as allowing for flexible adjustment of different flow map resolutions and controlling a wide range of particle behaviors. Export results produced by UE in formats that are compatible for further analysis in ArcGIS.

Finally, the model aims to lay the groundwork for further discussion on the potential to simulate water-solid interactions and erosion mechanics in the future. The chapter seeks to identify practical workflows for enhancing geospatial methodologies, advocating for accessible, intuitive, and efficient solutions in GIScience research and environment modelling.

4.2 Research Materials

4.2.1 Study Site

The study site is an agricultural field located in Milverton, Southern Ontario, within the Grand River Watershed, adjacent to the Nith River. The high-resolution drone data (10cm x 10cm per cell) utilized in this study was derived from optical imagery from a remotely piloted aircraft using structure-from-motion techniques, originally collected by Meinen & Robinson (2020) on May 17th, 2018. This dataset was selected because May is typically a transitional period where the soil is sufficiently moist from spring precipitation yet not overly saturated by late-season rains, providing a reasonable baseline for representing typical surface runoff and drainage pattern on soil. Moreover, vegetation in May is typically in an early growth stage, implying existence of some influence on microtopography but not enough to alter the terrain morphology captured in a DEM. This means the results can be generally interpreted as an accurate depiction of the ground surface without concern for seasonal anomalies such as snow. The site exhibits a range of elevation variations spanning 322.525 to 345.404 meters, indicating a gently rolling terrain. To the northeast corner of the dataset, there is a consistent downward slope culminating in a prominent convergent flow path basin. This high-resolution data allows for the clear identification of the main tributaries, facilitating detailed analysis of water flow patterns. The field's topography features a mix of gentle and moderate slopes, transitioning to steeper gradients near the flow path basin, where water concentrates, and erosion potential is greater. This basin will be referred to as the main basin, shown in Figure 1.



Figure 1. Dataset of farm field and study area of basin

The study area is a well-defined flow accumulation zone, encompasses a substantial portion of the field's upslope catchment, and can be distinguished by its noticeable elevation changes. This basin was chosen specifically for its potential to demonstrate flow accumulation, erosion, and sediment transport processes. Two primary tributaries converge at the edge of the field, feeding into the main basin, indicating preferential flow channels that concentrate water movement. Additionally, the high-resolution data reveals variations in surface roughness, including furrows from tilling and natural microtopography. Notably, the basin also features vegetation in its lower left corner, which acts as low-relief ridges, effectively barricading depressions and separating a potentially connected path.

4.2.2 ArcGIS

ArcGIS Pro was used to create a reference flow path using the D8 algorithm. ArcGIS Pro belongs to a suite of a commercial software application developed by Esri for spatial data analysis, mapping, and management of geospatial databases (Esri, n.d., "Introduction to

ArcGIS”). The annual licenses start at \$1300 per year for the basic version, and up to \$7930 per year for the professional plus version, in Canadian Dollars.

The software offers a graphical user interface for data processing tasks and provides additional processing features in the form of extensions available to professional plus versions (Esri, n.d., “ArcGIS Pro Pricing”) or at an additional cost of \$1430 per toolbox (e.g., Network Analyst, Spatial Analyst). These toolboxes provide an easy and accessible interface and is compatible with automation workflows (e.g., ModelBuilder) available as built-in features of ArcGIS, which can be used to conveniently customize and establish workflows for batch processing (Esri, n.d., “Automation with ArcGIS Pro”). The version of ArcGIS pro used was 3.5, with the minimum hardware requirement of 2 Central Processing Unit (CPU) cores, 8GB of Random Access Memory (RAM), and can emulate with the integrated Graphics Processing Unit (GPU) from a supported CPU configuration, if there is none.

4.2.3 Engine Tools

To develop the simulation, several tools were chosen to reduce complexity and facilitate a modular environment. The game engine used is UE version 5.1, released in 2022, and is free for public use. It is designed to leverage the capabilities of modern multi-core CPUs and GPUs to offer advanced real-time rendering features, a robust physics engine, and flexibility with diverse data formats (Krüger et al., 2024). The official UE documentation lists recommended hardware specifications of a Quad-core Intel or AMD, 2.5 GHz or faster CPU, 32 GB of RAM, and a dedicated GPU with 8GB or more video RAM (Epic Games, n.d., “Hardware and Software Specifications for Unreal Engine”). While the minimum requirement to use UE can be lower, more advanced tasks will require more powerful hardware.

The goal of this chapter requires selecting a set of appropriate equipment that can handle the intensive computational and graphical demands of experiences built in UE while within a \$2000 spending range (in Canadian Dollars), not accounting for external peripherals. The hardware specifications are listed in [Appendix C](#). The chosen hardware is sufficiently powerful

to handle foreseeable updates and advancements in game engine technology. This future-proofing ensures that the system will remain relevant and capable of handling new features and more demanding simulations for several years to come (estimated to be at least 5 years).

Voxel Plugin is a third-party asset for UE designed to create and manipulate 3D voxel-based terrains, it offers convenient tools for creating infinite, volumetric terrains with options for smooth or cubic rendering modes (Voxel Plugin, 2024, “Getting Started with Smooth Terrains”). To render a smooth 3D surface, the plugin enables the use of the Marching Cubes algorithm presented by Lorensen & Cline (1987) to parse voxel datasets and draws mesh surfaces by connecting the corner points of each voxel unit. The technique classifies each point as either inside or outside the surface based on their voxel density values. Because each of the eight corners of the cube voxel can be inside or outside, there are 2^8 (256) possible cases for describing how a surface intersects the cube (Lorensen & Cline, 1987). Using a predefined lookup table, the algorithm selects the correct triangle configuration for each case. The resulting triangles collectively form a smooth surface representation, while the underlying voxel data remains hidden, providing the volumetric structure for the rendered surface (Lorensen & Cline, 1987).

The convenience of Voxel Plugin allows the creation and manipulation of voxel terrain without an understanding of the underlying algorithm for visualization. The plugin allows generation of voxel terrain based on user-imported heightmap data, changing the scale of voxels, and includes a library of functions that simplify voxel manipulation operations. Therefore, it is particularly useful for interactive features requiring dynamic terrain changes. The plugin was purchased in the asset store for a cost of \$350 to access the tool transforming height-map into voxel terrain. The version used was Legacy 1.2, as version 2.0 was in active development during the time of the project and presented drastic changes to workflow that were incompatible with import and voxel manipulation operations planned in the methodology.

The Niagara Simulation Framework (NSF) is part of the advanced visual effects system in UE5, designed to create realistic, real-time simulations for visual effects like particle systems,

fluid dynamics, smoke, and fire (Unreal Engine, 2021, “Niagara Overview”). The framework is designed to leverage GPU-based computations, ensuring efficiency and high performance for real-time applications to produce frame-dependent results. It is useful for simulations that require parallel processing on a significant number of particles, such as fluid flow. NSF enables extension of functionality by applying Simulation Stages to achieve a higher degree of physical behavior between particles. Simulation Stages allow application of sophisticated techniques such as Position Based Dynamics (PBD) and SPH, which allow particles to interact with each other and the environment in a realistic way.

Simulation Stages is the core feature of the NSF, which is a simulation pipeline with the below primary stages of processing: Emitter Spawn, Emitter Update, Particle Spawn, Particle Update, and Render (Unreal Engine, 2024, "Niagara Overview"). These stages define the order in which calculations and behaviors are performed in the system by the use of modules, which are blocks of rendering logic used to program particles. In the Spawn stages, new particles or emitters are initialized once with their properties, such as position, velocity, spawn rates and other initial attributes. The Update stages process is run every frame by applying behaviors, forces, or environmental influences on particles or emitters based on the stack of modules. During the life cycle of a particle, any constraints or complex interactions between particles, such as collisions or bindings, are expected to be resolved with modules in the update stage to ensure consistency and physical accuracy. Finally, the Render stage prepares the visual output, ensuring that the particles or simulated entities are properly rendered for the desired effect. This structured pipeline ensures efficient and dynamic particle simulation within the NSF. In addition to these default stages, users have the flexibility to add custom simulation stages for specialized calculations and processing, such as stages to achieve fluid simulations.

In NSF, Neighbor Grid 3D is a spatial partitioning and hashing data structure used to optimize particle interactions by organizing particles into a 3D grid (Unreal Engine, 2024, “UNiagaraDataInterfaceNeighborGrid3D”). Each particle is assigned to neighboring cells based on its position, allowing the particle system to efficiently query and search for particles

within cells that are considered spatially relevant, ignoring cells where no particles are nearby. This structure is especially critical for the application of SPH as it avoids a brute-force check of every particle against all other particles, reducing computation cost to only handling particle collisions of a localized collision zone, and is commonly used for SPH simulations (Joselli et al., 2015).

A combined approach that includes a mix of both PBD and SPH is used as there is a need for each particle to consider both collisions with the voxel terrain as well as each other continuously. The simulation stages have built-in collision modules for implementing PBD on particles, which handles how each particle interacts with the mesh and colliders of the voxel terrain, and a custom module for SPH was implemented to describe how particles should move together to behave like a fluid.

The methodology below describes how these tools were used to create the model and export a resulting flowpath map for analysis in ArcGIS.

4.3 Methodology

4.3.1 Pre-processing

The point cloud data was provided in Laser (LAS) file format. This was imported into ArcGIS to filter their classification codes, where a LAS dataset was first created by combining ground data. Then, the LAS Dataset To Raster Geoprocessing tool was run using interpolation type of linear triangulation, floating point data type, and a cell size of 0.1 and Z-factor of 1. The value field used was elevation to produce a DEM. Triangulation interpolation was used because the aim is to use the DEM as the basis for a 3D environment and a Triangulated Irregular Network (TIN) would produce a higher quality representation of the continuous surface for the intended usage. All pre-processing done in ArcGIS used the coordinate system WGS_1984_UTM_Zone_17N.

Following the creation of the DEM, the next phase focuses on spatially constraining the DEM to the study area, which is the boundaries of the main basin. Since it was difficult to precisely control the size of the crop area within ArcGIS, the DEM was first exported as a 16-bit .Portable Network Graphics (PNG) and then cropped to 2048 x 2048 grids using GDAL:

```
gdal_translate -projwin
```

The extent used was provided in Table 1:

Table 1. Universal Transverse Mercator (UTM) extent coordinates of crop area by easting and northing

Extent Corner	Coordinates (X, Y)
Upper Left	(512127.720, 4815646.194)
Lower Left	(512127.720, 4815441.394)
Upper Right	(512332.520, 4815646.194)
Lower Right	(512332.520, 4815441.394)
Center	(512230.120, 4815543.794)

The newly clipped DEM was loaded into the ArcGIS map view for inspection and validation to ensure it accurately reflects the topography of the farm field, as well as to generate a geo-reference file in relation to the original full dataset in the Tagged Image Format or World File (TFW) format. This cropped dataset featured 10 cm cell resolution in a 2048×2048 grid, covering approximately $204.8 \text{ m} \times 204.8 \text{ meters}$. The cropped DEM of the main basin is shown in Figure 7.

The 2048×2048 grid dimension was determined to ensure alignment with the simulation output from UE to achieve a reasonable raster comparison. Due to Video Random Access Memory (VRAM) limitations a larger area could not be loaded within the engine without significantly affecting performance and further operations. In addition, the generation of a

higher resolution flow path map in NSF also faces similar limitations that require additional computation power.

Then, the minimum and maximum range of the elevation was obtained by using:

```
gdalinfo -mm "filename"
```

The elevation of the DEM was remapped to the full 16-bit range of 0 to 65535 using the command, using the minimum and maximum values obtained previously:

```
gdal_translate -of PNG -ot UInt16 -scale MIN MAX 0 65535  
"source" "destination.png"
```

The resulting .PNG is the heightmap input to UE for Voxel Plugin to create a voxel representation.

Typically, ArcGIS exports DEMs with an 8-bit color depth, which confines the raster to only 256 distinct elevation values. This limited resolution can result in terrace artifacts in the resulting 3D representation due to a lack of smooth gradient values, in addition to loss of data accuracy, often appearing as staircase-like visuals on vertical slopes (e.g., Figure 8). Higher depths such as 32-bit were incompatible with the import capabilities of Voxel Plugin. Thus, a 16-bit .PNG was needed to preserve the precision of the elevation data.

Lastly, the location and temporal metadata of the dataset is used to set up lighting conditions. The tools to complete the setup are provided within the environment as part of the template and necessary objects have already been placed in the scene. In the scene tree, there is a *SunSky* object provided that simulates the position of the sun based on the given latitude and longitude of the dataset. Table 2 below documents set of values that were used.

Table 2. Parameters used for coordinates in UE

Property Name	Property Value
Latitude	43.49301

Longitude	-80.84838
Time Zone	-5.0
North Offset	-90.0
Month	5
Day	17
Solar Time	16.24

A diagram showing the flow and transformation of data is shown in Figure 2.

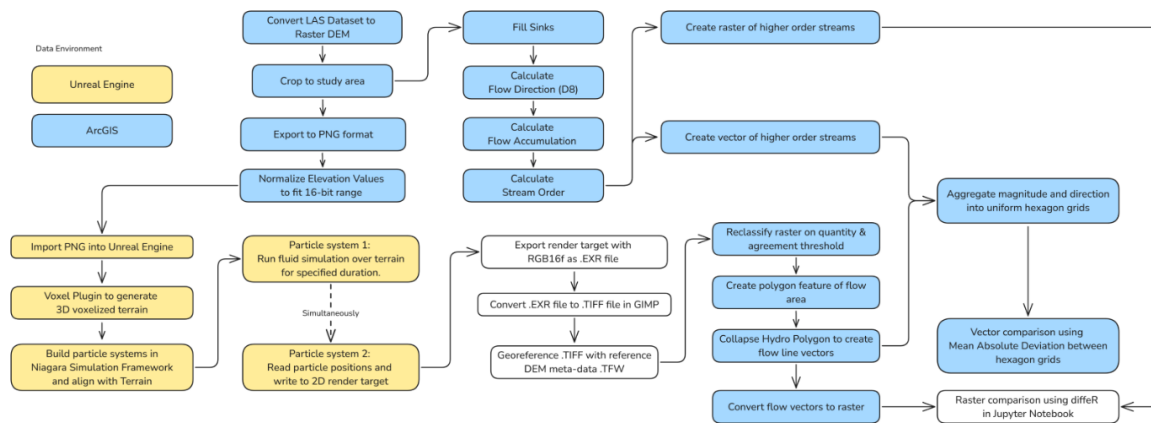


Figure 2. Workflow diagram

4.3.2 Reference Flow Path

The DEM was first processed by running the Fill tool to eliminate depressions and sinks that could distort water flow, ensuring a continuous surface for flow paths. Using the D8 algorithm, which assigns each cell one of eight possible flow directions based on the steepest descent, the Flow Direction tool was then executed to determine the path water is likely to follow from each cell. Next, the Flow Accumulation tool calculated the cumulative number of upstream cells contributing to each cell. Finally, the Stream Order tool was applied to assign Strahler stream orders to the flow network, classifying stream segments according to their

hierarchy and significance. The workflow can be seen in the upper sections of the workflow diagram in blue, shown in Figure 2.

This workflow produced a map depicting the major flow paths overlaying the original DEM, shown in Figure 9. The lower order streams were removed to enforce an even comparison with the UE model by focusing on the primary flow paths, which are the longest continuous lines to the basin. The first and second orders, which represent smaller tributaries, were selected and filtered by their Strahler Order field values then subsequently removed to isolate and emphasize only the higher order flows that are the primary contributors to the overall drainage system for comparison. It was expected that both models would produce different but roughly similar paths of flow and focusing on primary paths will allow a fairer analysis of the main differences between approaches.

4.3.3 Fluid Simulation

The flow path model centers around the integration of three core areas within UE: Voxel Plugin, NSF, and Blueprints. The main concepts employed are PBD, SPH, and their implementation via the use of Neighbor Grid 3D in Simulation Stages within the NSF. Voxel Plugin was used to create a dynamic and detailed 3D representation of DEM data, NSF for building emitters that represent fluids, using the built-in collision module to collide and interact with voxels, SPH was used to model movement and behavior of fluid particles with each other, and Blueprint was used to manage interactions between the voxel system and NSF particles.

First, a new Unreal Project was created using the Simulation Template, due to its simple setup and initial provided set of tools for geography. A *Voxel Heightmap Asset* was created by importing the cropped DEM. In the *Voxel World* object, an object generator was used to read the *Voxel Heightmap Asset* to render the terrain. Voxel size is set to 1.0 with a world size of 2048 to match the resolution of the original .PNG file. The render type used is Marching Cubes, and the default material was used. This is then saved to a *Voxel World Save Object* for convenient loading. The *Voxel World* object is centered at zero co-ordinates.

Then, a Niagara Particle System was created starting with the Radial Burst Template. The trail emitters are deleted, and only the main burst emitter is kept. The system and template provide various particle parameters such as ID, position, and velocity by default which can be accessed and modified in any stage. The modified main emitter properties are listed with their values in Table 3.

Table 3. Primary emitter properties

Property Name	Property Type	Property Value
Local Space	Boolean	True
Determinism	Boolean	True
Interpolated Spawning	Boolean	True
Sim Target	Enum	GPUCompute Sim
Calculate Bounds Mode	Enum	Fixed
Fixed Bounds	Vector2	Min(-1024,-1024,-512) Max(1024,1024,512)
Require Persistent IDs	Boolean	True

Next, the System Properties of the Niagara Overview Node is modified. A fixed bound of 1024 units in all directions is set to match the simulation boundaries of both the voxel terrain data and set the expected simulation space, with determinism checked to true. By constraining the simulation within fixed limits, each particle's position can be deterministically mapped to a corresponding cell in a neighbor grid, reducing the potential for computational errors that might arise from particles moving outside the expected domain, as well as allowing the system to pre-allocate buffers and optimize parallel processing routines (Joselli et al., 2015). Without fixed boundaries, the simulation might require dynamic resizing of data structures or more complex boundary-checking logic, which could significantly reduce computational efficiency and increase the risk of performance bottlenecks.

Under System Update, the modules and properties are modified as seen in Table 4. All other properties which are not listed are left unchanged, so they take their default value.

Table 4. NSF system properties

Module	Property Name	Property Type	Property Value
Initialize Neighbor Grid	Grid Extents	Vector3	(1024,1024,1024)
	Local Pivot	Vector3	(0.5, 0.5, 0.5)
	Coordinate Space For Rotation and offset	Enum	Simulation
	Define Neighbor Grid	Boolean	True
	Max Neighbors per Cell	Int32	6
System State	Loop Behavior	Enum	Multiple
	Loop Count	Int32	150
	Loop Duration	Float	5.0

The Emitter Spawn stage is kept empty as only one emitter will be used for the fluid particles. For the subsequent stages and their respective properties in each module, see Table 15. The default modules that are not listed were deleted or set to none. Then, a custom simulation stage was added to populate the initialized Neighbor Grid 3D. The code snippet provided in [Appendix D](#) was written in a custom High-Level Shading Language (HLSL) node. The stage iterates once over all particles, transforms the particle's position into the coordinate system of the relevant grid, identifies the appropriate grid cell, and then conditionally adds the particle's ID into that cell's list of neighbors. Following the registration of particle ID, this custom node was connected to receiving input from the previous stages in the system, modifying the neighbor grid as a custom module in the simulation stage. The input to this module was linked to the reference of the initialized Neighbor Grid 3D, the particle position parameter, and a transformation matrix used to spatially map the world unit to grid units. The transformation

matrix maps world space to simulation space to align the position of the emitter and the neighbor grid to make use of the earlier defined fixed boundaries. This process effectively populates the grid with particle information, which was used in later stages to calculate intra-particle collisions and forces.

To add intra-particle collisions, another custom simulation stage was added. This stage iterated over every particle four times to minimize errors that may persist after a single iteration. The code snippet provided in [Appendix D](#) implements collision detection and response using a PBD approach, also as a custom HLSL node. The implementation references Müller et al. (2007). The core of the process was encapsulated within a GPU simulation block, which first defines an array of 27 index offsets corresponding to the current cell and its immediate neighbors in 3D space using the previously registered IDs in the Neighbour Grid 3D. These offsets were used to iterate over every neighboring cell in a $3 \times 3 \times 3$ grid around the particle's current position. The particle's current position was converted from simulation space to unit space using the same transformation matrix as before, and then this unit position was converted to a discrete cell index in the neighbor grid. For each neighboring cell, the code loops over all potential neighbor slots to retrieve neighbor particle IDs. For each neighbor particle, it fetches the other particle's position and collision radius via direct reads and calculates the distance between the two particles (Müller et al., 2007). The difference and direction were calculated as below:

$$v_{ij} = p_i - p_j$$

$$d_{ij} = |v_{ij}|$$

$$n_{ij} = \frac{d_{ij}}{v_{ij}}$$

where p_i, p_j are the positions of the neighboring particles, v_{ij} is the difference vector between the particles, d_{ij} is the magnitude of the Euclidean distance, and n_{ij} is the unit normal vector

pointing from particle j to particle i , indicating the direction of force that will push the particles apart.

If the neighboring particles overlap, a collision gets reported. When a collision gets detected, the response is applied by accumulating an offset vector proportional to the overlap magnitude and the mass ratio, and a constraint count was incremented to record the number of collision constraints affecting the particle (Müller et al., 2007). The overlap magnitude, or penetration depth, is defined as:

$$\Delta_{ij} = (r_i + r_j) - d_{ij}$$

where Δ_{ij} represents the amount of inter-penetration between particles, r_i and r_j are the radius of the particles respectively. A very small threshold constant ε of 10^{-5} is also used to check that the penetration distance is significant enough to count a collision.

The mass-weight ratio determines how the collision resolution is distributed between the two particles, defined as:

$$\alpha_{ij} = \frac{m_i}{m_i + m_j}$$

where α_{ij} is the proportion of the total mass between particles, and m_i , m_j are the mass of the particles respectively.

After processing all neighboring cells, if any collision constraints were accumulated, the particle's position was adjusted by a scaled offset of the vectors, defined as:

$$O_i \leftarrow O_i + (1 - \alpha_{ij})\Delta_{ij}n_{ij}$$

where O_i is the accumulated offset of a particle, representing positional corrections that need to be applied to resolve interpenetrations. The term $(1 - \alpha_{ij})$ weights the contribution based on the relative mass of the particles.

The new velocity of a collided particle was then updated based on the change in position over the time step. The input to this custom node is also connected and added to the stage as a module, then linked to user defined parameters, considering static and kinetic friction, a relaxation amount, referencing the same Neighbour Grid 3D, the particle attribute data reader FluidPAR and the same transformation matrix. This process determines overlapping particles through their neighboring distance and collision radius calculations, then resolves these collisions by applying appropriate positional corrections and velocity adjustments in a mass-weighted manner. The input constants used are listed in Table 5.

Table 5. Friction coefficients are small because particles are expected to collide frequently. More flow movement is encouraged

Property Name	Property Type	Property Value
Kinetic Friction	Float	0.001
Static Friction	Float	0.01
Relaxation Amount	Float	0.005

To add fluid-like movement to particles, an SPH simulation stage was added. This stage iterates over each particle four times, and includes three custom modules for calculating fluid forces, external forces, and updating particle position respectively. The implementation below follows the methodologies described in Müller et al. (2003), Pazouki et al. (2014) and references approaches available in (Unreal Engine, 2024, “Content Examples”).

The goal of the first custom module in the SPH stage was to accumulate fluid forces on each particle by computing density, pressure, and the resulting force using a set of central kernel functions. Code for the module is provided in [Appendix D](#). The custom module implements a Poly6 kernel function to compute the density contributions from neighboring particles (Müller et al., 2003). For distances r between 0 and the smoothing length h , the kernel is defined as:

$$W_{poly6}(r, h) = \frac{315}{64\pi h^9} (h^2 - r^2)^3$$

The coefficient term $\frac{315}{64\pi h^9}$ is used as a normalization constant to ensure mass conservation across the smoothing kernel (Müller et al., 2003). For distances greater than h , a value of 0 is used to indicate no density contributed. Then, the density is computed for each particle by iterating the kernel over nearby particles to sum their contributions to the particle's density, which is defined as:

$$\rho_i = \sum_j m_j W_{poly6}(r_{ij}, h)$$

where ρ_i is the density at particle i , m_j is the mass of neighboring particle j , and $W_{poly6}(r_{ij}, h)$ is the kernel function applied to obtain the absolute distance between the two particles r_{ij} .

Then, a pressure value was calculated based on deviation from a target density (Pazouki et al., 2014). Pressure controls how much force each particle should exert on their neighboring particle. The pressure for a particle i is defined as:

$$p_i = \max(P_s(\rho_i - \rho_0), 0)$$

where p_i is the pressure at a particle's position, P_s is the pressure scale that controls how strongly pressure responds to deviations from the target density, ρ_0 represents the desired density of the fluid when it is in equilibrium, and max is used to ensure that the pressure is not negative.

The pressure force acting on a particle i is defined as:

$$F_{p,i} = -S_p \sum_{j \neq i} \frac{p_i + p_j}{2\rho_j} \nabla W_{spiky}(r_{ij}, h)$$

where $F_{p,i}$ is the pressure force on particle i , and p_i, p_j are the pressures for the current particle i and its neighboring particle j respectively (Müller et al., 2003). The coefficient term S_p is the

scale of the pressure force, this term was exposed as a module parameter and used a default value of $\frac{15}{\pi h^6}$. The term $\frac{p_i + p_j}{2}$ averages the pressure of both particles to ensure a symmetrical interaction from both, ρ_j is the density of the neighboring particle and the term $\nabla W_{spiky}(r_{ij}, h)$ is the gradient of the spiky kernel, which provides the directional derivative of the function with respect to the distance vector r . (Müller et al., 2003).

The spiky kernel gradient determines the pressure force direction and magnitude, defined as:

$$\nabla W_{spiky}(r, h) = (h - r)^2 \frac{r}{|r|}$$

where the term $(h - r)^2$ provides the scaling based on distance, and the term $\frac{r}{|r|}$ is the unit vector that provides direction.

The viscosity force acting on a particle i is defined as:

$$F_{v,i} = S_v \sum_{j \neq i} \frac{v_j - v_i}{\rho_j} \nabla^2 W_{viscosity}(r_{ij}, h)$$

where $F_{v,i}$ is the viscosity force acting on particle i , and v_i, v_j are the velocities for the current particle i and its neighboring particle j respectively. The coefficient term S_v is the scale of the viscosity force for controlling how much resistance there is to relative motion and deformation of the fluid where a higher value leads to a more pronounced smoothing effect in the velocity field (Müller et al., 2003), this term was exposed as a module parameter and used a default value of $\frac{45}{\pi h^6}$. ρ_j is the density of the neighboring particle and the term $\nabla^2 W_{viscosity}(r_{ij}, h)$ is the Laplacian of the viscosity kernel, quantifying how much a particle's velocity is adjusted based on the difference in velocities with its neighbors.

The Laplacian function of the viscosity kernel which quantifies how the smoothing effect decays with distance is defined as:

$$\nabla^2 W_{viscosity}(r, h) = h - r$$

The constants usually present in the kernel functions were absorbed into the scale coefficients to reduce the number of operations and simplify calculations. The pressure and viscosity calculations also check that particle densities exceed the threshold $\bar{\rho}$.

Finally, these forces are combined from all neighbors into a net force, defined as:

$$F_i = F_{p,i} + F_{v,i}$$

where F_i is the net force acting on a particle i . The constant values used are listed in Table 6.

Table 6. Parameters that affect intra-particle collisions

Property Name	Property Type	Property Value
Kernel H (h)	Float	0.5 cm
Mass (m)	Float	0.02 kg
Pressure Force Scale (S_p)	Float	5.8
Pressure Scale (P_s)	Float	3.6
Target Density (ρ_0)	Float	1.0 g/cm ³
Viscosity Force Scale (S_v)	Float	2.5

The second custom module in the SPH stage accumulates external fluid forces such as gravity, grounds, and walls. The HLSL is provided in Appendix D. The particle's position is evaluated relative to a defined field boundary on all axes, nudging the particle away from walls by adding or subtracting a unit value. This avoidance vector is then scaled by a factor and added to the force. In addition, a gravity of -9.81 in the Z-direction is also added to the force to keep the fluid grounded. These adjustments modify the particle's force to account for interactions with external boundaries and gravitational acceleration.

The third custom module in the SPH stage updated the particle positions and velocities by using the accumulated forces of previous modules as inputs, then converted them into velocity

defined as force multiplied by time between each frame. The HLSL is provided in [Appendix D](#). Initially, the force was clamped to a maximum threshold to prevent excessive acceleration. The velocity of a particle was then updated using the clamped force scaled by the time delta, and if the resulting velocity exceeds a specified maximum, it was normalized and scaled down. Finally, the position was updated by adding the velocity to the current position. This ensures that particle motion remains physically plausible and within predefined limits, which was necessary to maintain stability in the simulation. The constants used are listed in Table 7.

Table 7. Simulation stability parameters

Property Name	Property Type	Property Value
Delta Time	Float	0.001 seconds
Max Force	Float	0.05 cN
Max Velocity	Float	500.0 cm/s

Finally, a sprite renderer was used in the Render stage because the goal is to emit a large quantity of particles. Sprite renderers are inherently less computationally expensive than mesh or ribbon renderers, each sprite is rendered as a simple white circle texture. This simplicity translates to significantly lower vertex processing and rendering overhead.

To visualize the speed of particles, their velocity parameter was linked to the X parameter where a color spectrum normalized to the minimum and maximum range of speed was used.

After the emitter was constructed, the particle system is placed in the 3D scene and aligned with the *Voxel World* at the world center coordinates. The parameters were obtained by manual calibration (trial and error) and known default values (e.g., gravity, water density), until stable and realistic behavior was observed. Visual validation of the fluid-terrain interaction was then conducted through iterative sessions, where particle emission patterns, movement, collision, fluid properties, and potential penetration issues were qualitatively assessed from various camera perspectives in an unlit shading environment. Then, incremental adjustments to the Niagara system parameters were performed to refine the simulated fluid dynamics based on

observed discrepancies so that physical behaviors are not unrealistic. Screenshots of the particle system during simulation in lit and unlit environments can be seen in Figure 10 and 11.

4.3.4 Data Export

To achieve a comparable analysis with a two-dimensional map created from ArcGIS, the three-dimensional flow of the simulated fluid particles was projected onto a 2D plane. This was accomplished through the creation of a secondary Niagara emitter designed to track and represent the spatial progression of the primary fluid simulation emitter. This emitter samples the particle positions from the source fluid emitter and transforms these 3D positions into a 2D representation, drawing their paths onto an image texture (Zukowski, 2025). The image is then exported for further processing in ArcGIS.

First, a Render Target 2D was created in the project directory with the format RTF RGBA16f. Next, a new Material was created, linking its emissive color parameter to a texture sample node reading from the newly created Render Target 2D.

Render Target 2D is a rendering data structure for dynamically manipulating visual data within UE, whereas Materials can be used as an output channel to export visual data outside of the engine. The use of the RTF RGBA16f format allows the default export format of the image to be Extended Range (EXR), a specialized variant of High Dynamic Range (HDR) imaging that provides 16-bit floating point precision to store detailed color information.

Then, a secondary emitter was created in the Niagara system with the same properties as the main emitter. In the Emitter Spawn stage of the secondary emitter, a Grid2D Collection data structure was defined. Next, the built-in module to set resolution of the Grid 2D was added. The properties used in the module are listed in Table 8.

Table 8. Extent for secondary emitter

Property Name	Property Type	Property Value
Grid	Grid2D Collection	Linked to <Emitter> Grid2D
NumCellsX (N_x)	Int32	4096
NumCellsY (N_y)	Int32	4096
Resolution Mult	Float	1.0
World Grid Extents	Float	(1.0, 1.0)
Resolution Method	Enum	Independent

A matching Render Target user parameter to reference the previously created data structure was also defined in the system. The Render Target properties are listed in Table 9.

Table 9. Alignment parameters within NSF and the Render Target file

Property Name	Property Type	Property Value
Size	Vector2	(4096, 4096)
Mip Map Generation	Enum	Disabled
Mip Map Generation Type	Enum	Unfiltered
Inherit User Parameter	Boolean	False
Override Format	Boolean	False
Render Target User Parameter	User Parameter	Linked to <User> UserRenderTarget

This emitter does not have any modules in the Emitter Update, Particle Spawn, Particle Update, and Render stages. Instead, three custom simulation stages are added: Initialize, Write to Grid, and Write to Texture. The simulation stage settings are listed in Table 10. No modules were added to the Initialize stage, which ensures that on every simulation reset, the grid is also reset to an empty state.

Table 10. Secondary emitter custom modules and properties

Property Name	Property Type	Property Value
Initialize		
Iteration Source	Enum	Data Interface
Data Interface	User Parameter	Linked to <Emitter> Grid2D
Execute Behavior	Enum	On Simulation Reset
Write to Grid		
Iteration Source	Enum	Data Interface
Data Interface	User Parameter	Linked to <Emitter> Grid2D
Execute Behavior	Enum	Always
Write to Texture		
Iteration Source	Enum	Data Interface
Data Interface	User Parameter	Linked to <Emitter> RenderTarget2D
Execute Behavior	Enum	Always

Setting the iteration source to the Grid2D data interface enables logic within the simulation stage to iterate over each cell in the 2D grid, instead of over particles.

In the Write to Grid stage, a custom module is added that takes a list of properties as input, seen in Table 11.

Table 11. Aligning scale of grid to world

Property Name	Property Type	Property Value
Grid2D	Grid2D Collection	Linked to <Emitter> Grid2D

PAR	Particle Attribute Reader	Read from Emitter Name (Main_Particle_Emitter)
World Scale (S_w)	Float	0.5
Distance Threshold (ϵ_d)	Float	2.0

The Grid2D property uses Particle Attribute Reader (PAR) which reads particle information (position) from the main emitter, looping over every particle. This operation is done on every cell in the grid and exposes a scale parameter for mapping the particle position to the 2D grid. A value of 0.5 was used as a default to match the resolution of the terrain, which is chosen to obtain a higher resolution flow map.

First, to ensure the center of a cell is the origin of comparison with particle positions, the grid positions are offset by half in the X and Y direction based on the number of cells. The adjusted indices are defined as:

$$i' = i + (-0.5) \cdot N_x$$

$$j' = j + (-0.5) \cdot N_y$$

where N_x, N_y are the number of cells in the X and Y directions previously declared in the creation of the Grid2D, i and j are the original X and Y indices. This shifts the indices by half the grid size in the negative direction, centering the grid around the origin (0,0).

i', j' are adjusted indices after centering, representing the cell's position relative to the grid's center rather than its corner. The grid indices are obtained by converting an execution index into grid coordinates, which was available as a built-in node.

These adjusted indices are then utilized to construct a 2D vector. This vector is subsequently scaled by a world scale factor defined as:

$$G = S_w \cdot (i', j')$$

where G is the 2D grid, S_w is the world scale factor to align the local cell size and grid space with world space. G is then assigned to the local namespace as the grid's 2D vector, effectively defining the spatial layout of the grid in the simulation environment.

Following this setup, an HLSL node is added to perform an iteration over the positions of all particles in the main emitter to evaluate their proximity to the cell's center at every (i', j') in the secondary emitter. If a particle has not collided, then it was skipped. This evaluation checks whether a colliding particle's position falls within a predefined distance threshold ϵ_d from the cell's center point using their Euclidean distance:

$$|p_n - c_{i',j'}| < \epsilon_d$$

where p_n is the position of the particles being iterated over, $c_{i',j'}$ is the center point of the cell in the current iteration. If the above condition is satisfied, then the color value is incremented by a solid white color. Thus, the operation can be described as:

$$G_{i',j'} \leftarrow G_{i',j'} + \sum_{n=1}^{N_p} \chi(|p_n - c_{i',j'}| < \epsilon_d) \cdot (1,1,1,1)$$

where $G_{i',j'}$ are the grid cells containing an RGBA value at indices i', j' . N_p is the total number of particles to iterate, starting at $n = 1$. χ represents the proximity condition, equating to 1 if true and 0 if false.

Next, in the Write to Texture stage, the Render Target 2D will render the grid to the matching Material created. Another custom module is added, taking only the reference for the Grid2D and the Render Target reference as input. The execution index of the Grid2D was converted to an X, Y index format, then linked to corresponding indices on the render target. Finally, the render target value sets the RGBA value at all indices of the corresponding locations in the Grid2D, displaying white pixels where the particles have traveled through. A completed simulation of 150 cycles was done in approximately 30 minutes while the rendering speed was

at an average of 12 frames per second, with an approximate total of 120,000 particles. Figure 12 shows the evolution of particles drawing flow paths mid-simulation.

After the particle has completed its lifecycle and all particles have been spawned. The Material containing the flow paths of particles was exported to a file in the EXR format. The EXR file was opened in GIMP, an image editor, and exported in the .Tagged Image File Format (TIFF) format. The TIFF format was chosen because it was an ArcGIS compatible image format also capable of maintaining 16-bit depth information. Lastly, the TFW world file from the original cropped DEM was copied and renamed to match the filename of the converted TIFF file with the flow path. This enables the geo-reference of the exported image to match the original DEM, when loaded into ArcGIS. When rendered based on a threshold of flow agreement in ArcGIS, the low and high flow channels in the UE model can be visualized, as seen in Figure 3.

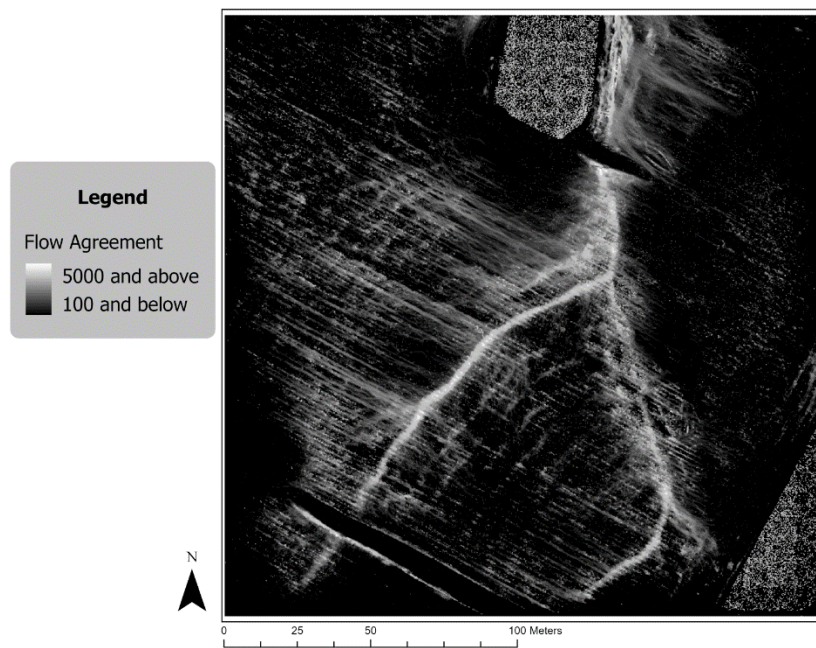


Figure 3. Visualizing the exported raster from UE

The visualization adjusted the minimum and maximum visualization threshold to show cells where at least 100 particles agreed, and up to 5000 particles with flow agreement. The brighter pixels indicate a higher agreement, the darker pixels indicate less to no agreement.

4.3.5 Validation

The approach to validation primarily followed the methodologies described in Pontius (2022), using both a raster-oriented and a vector-oriented comparison. The goal is to assess how well the UE simulation model aligns with an established traditional model that is used to validate against real-world observations. The comparison between models emphasizes benchmarking the UE model against the D8 (the assumed truth), so that similarities and differences can be interpreted. The metrics used revealed similarities and discrepancies between the results obtained from both models, as well as provided insights into how the underlying assumptions and algorithms of the UE model led to results that differ from the D8 model.

4.3.5.1 Raster Comparison

The raster comparison methodology references Chapter 6 in Pontius (2022). Multiple spatial resolution analysis is a method used to gain insight into the spatial relationship between categories by analyzing data at multiple, increasingly coarser spatial resolutions. The multiple-resolution analysis was chosen because it showed how the agreement between these two methods changes as the spatial resolution becomes coarser, revealing allocation differences. There were only two categories analyzed at multiple resolutions: presence and absence of water at any cell. The raster comparison measures the diagnostic ability of the approach using UE (X) against a traditional D8 algorithm (Y), where presence of water flow in a particular cell is the phenomenon being diagnosed. A Hit is when both models predict flow at a cell, a Miss is when X does not predict flow at a cell that has flow in Y, a Correct Rejection is when both models do not predict flow at a cell, and a False Alarm is when X predicts flow at a cell that does not have flow in Y. A square contingency table is constructed at each resolution to provide

insights into the spatial allocation of a category's False Alarms and Misses. The contingency table calculates components of difference for Quantity, Exchange, and Shift. Quantity describes the difference in total number of cells with flow, Exchange describes the difference in allocation of flow zones among cells, and Shift describes more complex proximity differences in spatial patterns and distribution.

The UE model requires time to simulate the formation of flow channels and produces a raster describing the amount of agreement between particles at each cell, which higher cell count indicating higher confidence that flow exists in that cell. This contrasts with the static D8 model which determines flow direction based solely on topographic gradients without simulating temporal dynamics or providing confidence measures for flow presence. To achieve a meaningful raster comparison between the two maps, the flow path map produced by the UE simulation must first be simplified into a delineated center line through a filter to represent the precise trajectory of flows. Then, the lower-order streams were removed as they introduced significant discrepancies and noise between the results of models, which may obscure the comparison between the primary (most significant) flow paths of interest with the D8 model.

First, an initial identification and reclassification of cells within the UE simulation raster is performed to remove cells with a low confidence of flow. The confidence threshold was chosen as 100, obtained from iterative manual calibration to reduce noise and preserve the shape of continuous flow in the main tributaries. Cells where the flow value exceeded an agreement of more than 100 particles were reclassified to a class representing sufficient confidence of flow presence, while all remaining pixels indicative of low or absent flow were assigned to a separate class representing low confidence of flow presence.

The resulting binary raster was converted into individual polygon features using the Raster to Polygon tool, without simplifying the polygon. This resulting polygon effectively produces areas that represent flow agreement above the threshold, and the spatial extent of the flow. Then, the Collapse Hydro Polygon tool in ArcGIS was used with a collapse width of 0 and

without merging adjacent input polygons to show all potentially existing paths. This conversion outputs a vector polyline feature, representing the central line of all flow paths. Finally, to facilitate a direct pixel-based comparison with the D8 stream map, the vector centerline line was converted back into a raster format using the Polyline to Raster tool with the cell size set to match the original resolution of the UE simulation raster.

To complete the pre-processing, some simplification of the resulting flow is performed based on two assumptions:

The first assumption is that the initial reclassification threshold can be interpreted as a measure of certainty: a higher threshold implies that only cells with strong flow are retained, thus increasing confidence in the existence of a real flow path, whereas a lower threshold includes regions with lower flow quantity and, consequently, lower certainty.

The second assumption comes from the process of collapsing the polygon, which generated a field containing the average width of the polygon at each center line. This field was used as an estimation that the width of the polygon (distance from both sides of the polygon to the central line) can generally infer stream order, where greater flow is assumed to correspond with wider channels. This assumption was needed because the collapsed polygon could not be used in the Stream Order tool in ArcGIS due to a lack of connectivity and node relationships (e.g., to- and end-, flow accumulation) between polylines.

Combining the above assumptions, only the flows above a significant average flow width threshold of 10cm were extracted to create a new representative layer for final comparison and analysis. Figure 13 shows the vectorized flow lines from the simulated output.

Lastly, the contingency table for the multiple spatial resolution analysis was created with the provided code snippet in [Appendix D](#), which makes use of the `diffeR` package in R.

4.3.5.2 Vector Comparison

The vector comparison methodology references Chapter 11 in Pontius (2022). The goal of the metrics for vector comparisons is to achieve a quantitative assessment of how X and Y differ in terms of their mean magnitude, mean direction, and the overall spread and nature of the deviations in both magnitude and direction. In the book, the Mean Absolute Deviation (MAD) metric is defined as the sum of the Quantity and Allocation components. However, due to the different number of paths in the vector datasets X and Y, a pairing method could not be fairly made to ensure every vector in X had a corresponding vector in Y. This did not meet the prerequisite condition to use the method of the recommended tools provided by Pontius (2022). Thus, to obtain the MAD for direction, an alternative method will be used to calculate Quantity and Allocation differences separately, specifically using spatial aggregation at various scales. According to Pontius (2022), the Quantity and Allocation Deviations are defined as:

$$\text{Magnitude Quantity Deviation} = |\bar{Y} - \bar{X}|$$

$$\text{Magnitude Allocation Deviation} = \max\left(0, \left(\frac{1}{N} \sum_{i=1}^N |Y_i - X_i|\right) - |\bar{Y} - \bar{X}|\right)$$

$$\text{Magnitude MeanAbsolute Deviation} = \max\left(|\bar{Y} - \bar{X}|, \frac{1}{N} \sum_{i=1}^N |Y_i - X_i|\right)$$

$$\text{Direction Quantity Deviation} = |\bar{\delta}|$$

$$\text{Direction Allocation Deviation} = \max\left(0, \left(\frac{1}{N} \sum_{i=1}^N |\delta_i|\right) - |\bar{\delta}|\right)$$

$$\text{Direction Mean Absolute Deviation} = \max\left(|\bar{\delta}|, \frac{1}{N} \sum_{i=1}^N |\delta_i|\right)$$

where $\bar{Y}$ is the magnitude $\bar{X}$ is the magnitude i represents the individual vectors for magnitude N in the dataset. $\bar{\delta}$ is the direction deviation between the mean vectors of X and Y.

To analyze allocation difference, an aggregated method was used to prepare the vector datasets. First, the vector dataset of Y was obtained by converting using the Raster to Polyline tool. Both the direction and magnitude data of the flow paths were joined into a grid of hexagons to calculate the MAD for magnitude and direction between X and Y, where each hexagon represents a consistent spatial extent for both datasets, allowing pair-wise comparisons. The Generate Tesselation tool was used to create three hexagon grid sizes: 10, 20, and 30 square meters, to ensure the robustness of the calculated results across aggregation scales.

For each dataset, the attribute table was opened, and two fields for direction in the x and y component were added to store vector components. Using the field calculator with Python 3, `Direction_X` was calculated as `math.sin(math.radians(!Direction!))` and `Direction_Y` was calculated as `math.cos(math.radians(!Direction!))`, converting direction angles (degrees from North) into components on a unit circle.

Then, the hexagons at each size were spatially joined with the vector datasets for their magnitude and direction. The Spatial Join tool is used with a Join One to One operation, targeting the tessellated grid layer and joining the polyline layers containing `Direction_X` and `Direction_Y`. In the Field Map, the merge rule for `Direction_X` and `Direction_Y` is set to Sum, producing the output fields `Sum_X` and `Sum_Y` for each hexagon.

Next, the circular mean was calculated to aggregate vector directions in each hexagon. The mean angle was formulated as:

$$\bar{\theta} = \arctan 2 \left(\frac{1}{N} \sum_{i=1}^N \cos \theta, \frac{1}{N} \sum_{i=1}^N \sin \theta \right)$$

where $\bar{\theta}$ is the mean direction and θ represents each individual vector direction.

After the join, the circular mean direction was calculated for each hexagon. A new field, `Circular_Mean_Dir` (type Double), is added to the output grid's attribute table. This represents the average direction of flow paths within each hexagon.

Differences in magnitude and direction between X and Y are then derived by comparing their respective values in corresponding hexagons. The Python code written for calculation in the field calculator is provided in [Appendix D](#).

To maintain the accuracy of the angular calculations, the data is projected into WGS84 / UTM zone 17N (EGM 96 Geoid), which preserves true directionality. The differences in flow path directions between X and Y were calculated such that the maximum angular deviation is constrained to 180 degrees, ensuring that the directionality comparison remains both meaningful and consistent.

4.4 Results

4.4.1 Viability Overview

The objectives were to assess the viability of game engines as a tool for producing flow path predictions based on how well it aligns with the four criteria of the FAIR principle: Findability, Accessibility, Interoperability and Reusability. In addition, the accuracy of the UE model will also be considered in assessing viability, comparing its results to the D8 model.

The *Findability* of UE tools pertaining to particle systems was moderate. The core particle system components and modules were reasonably findable through the interface, and a simple description of key features and nodes were conveniently accessed by hovering over elements. Due to the visual-scripting centric interfaces, its search functionality was moderately intelligent and often suggested relevant nodes to assist in development. However, it was

difficult to locate a detailed documentation portal or official and comprehensive knowledge base cataloging up-to-date features and nodes. There were significant challenges in locating specific SPH implementation examples and in-depth guidance on methodologies for building the simulation and exporting results. As a result, a large amount of trial-and-error was needed. The engine's search functionality often returned fragmented results that required substantial cross-referencing between different sources of knowledge, which were often from different (older) versions of the engine that may no longer be relevant.

Comparatively, the documentation for ArcGIS Pro was more findable as there are less changes between versions of the software. The accumulation of older guides and learning resources remain mostly relevant for the current available features and functionalities offered. There were built-in descriptions and in-depth metadata descriptions for most data and tools inside the ArcGIS Pro environment, as well as on the official website where a library of features is well-documented. This suggests that while UE meets basic *Findability* requirements, it falls short of the principle ideal of seamless resource discovery, and researchers may frequently face roadblocks during development related to navigating organizational barriers (e.g., insufficient information on data structures, modules, and node usage).

In terms of *Accessibility*, the UE model required significant development time and a steep learning curve. It had low accessibility due to the workflow requiring a deep understanding and implementation of custom modules to achieve desired results, in addition to familiarity with integrations from the rest of the development environment (e.g., terminology and development strategies presented a high technical barrier). In contrast, Spatial Analyst extensions provided by ArcGIS Pro was able to produce a similar result in less time and technical difficulty. The output uses conventional GIS-native data structures and can be directly used for further analysis. The input of data and output of results can be easily achieved by a non-technical person using ArcGIS Pro, allowing the workflow to be accessible for a larger group of users.

In contrast, the use of UE was more financially affordable than ArcGIS Pro in the context of producing and analyzing flow path maps. Assuming that only the basic annual license of ArcGIS Pro (\$1300) is purchased alongside the Spatial Analyst extension (\$1430), and no hardware upgrades or purchases are required (\$0), the annual total fee is \$2730. UE requires no license for this purpose (\$0), the Voxel Plugin is a one-time purchase (\$350), and assuming the hardware upgrade or purchase cost for a capable system is \$2000, this produces a total of \$2350 in the first year. The affordability advantage of using UE grows more significant over time due to the lack of recurring fees for a license and offers accessibility for users who value development costs.

The UE model and its workflow offered low *Interoperability* compared to the ArcGIS environment. Results produced by the UE model were not intended to be used in a GIS environment, so additional data transformations and conversions were needed to achieve compatibility. This intensive data transformation and conversion may lead to loss of data quality and integrity and may lead to limitations in available strategies for analysis. Additional post-processing work (e.g., assigning stream order to paths) is often necessary for an effective comparison within GIScience frameworks. In ArcGIS, the toolbox interface is opaque, and the internal computations are concealed, which allows it to conform to pre-established data standards. This is difficult to achieve in UE, as the export functionalities are aimed at creating software executables instead of diverse standard data formats. To overcome these barriers, various workarounds were needed (e.g., with the help of plugins and additional tools such as GIMP).

The *Reusability* of the UE model was moderate compared to geoprocessing toolboxes in ArcGIS Pro. Despite the game engine offering modular component-based architecture, the approach taken to simulate realistic particle interactions was memory intensive and called for a focus on technical optimization. Its open-source framework provided transparency into its underlying processes and calculations. For example, the built-in modules in NSF can be opened to view or modify its implementation. This openness facilitates detailed scrutiny and

verification of the model's accuracy in representing processes and allows modifications to achieve any implementation within the framework's constraints.

Modelers may reuse the model to refine or simplify operations to build models with a tailored level of precision and detail (e.g., flow path predictions for urban or agricultural use cases), and define model outputs to include only data of interest. This allows the model to be adopted for reuse by a variety of specialized technical and scientific domains (e.g., computer graphics, hydrology), as well as allowing for GIScience to mix approaches which is exemplified by this chapter. Comparatively, the tools in ArcGIS Pro have a closed architecture which limits algorithmic transparency but delivers superior practical reuse through standardized, hardware-agnostic toolboxes. The results from these toolboxes may be simplified in comparison to the level of depth that can be achieved in UE, but are easier to share, interpret, and reuse universally given their rigid interface for input and output data formats.

Lastly, the UE model had moderate *Accuracy* based on results obtained from the raster and vector analysis with the D8 model as a benchmark. The results showed that the UE model

was able to produce an accurate quantity of cells with flow, but did not produce flow in the same areas or direction as the D8 model. This reveals that the two models generally agree on size of flow but disagree on where the flow occurs. The D8 model produced vector paths that were on average longer than the UE model, but the direction of flow was drastically different. These comparisons suggest that there are fundamental and systematic differences in how flows are predicted in the two models. Figure 4 shows an overview of the produced flow paths. Further analysis of the results is provided in proceeding sections.

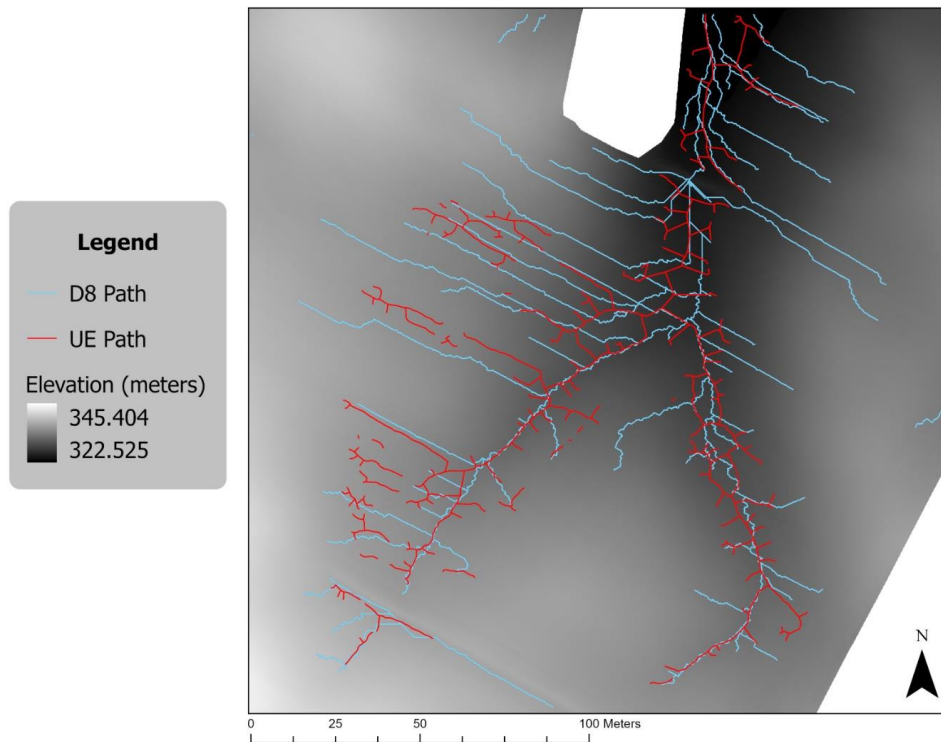


Figure 4. The different paths of the two models overlaid on top of each other

4.4.2 Key Findings

4.4.2.1 Raster Analysis

The multiple spatial resolution analysis produced Table 12, which presents the differences between two maps at aggregation multiples ranging from 1 (finest) to 2047 (coarsest), representing the size of cell aggregation windows. The statistics provided in the table represent proportions of the total map area that contribute to the types of differences: Quantity, Exchange, Shift, and Overall. However, the results are aggregate measures based on absolute differences and expressed as a percentage. Therefore, it does not indicate the direction of the difference, and it does not produce insights into whether the UE model or the D8 model is greater or lesser in the Quantity component.

Table 12. Total Operating Characteristics of the multi-spatial resolution analysis

Multiples	Resolution	Quantity	Exchange	Shift	Overall
1	10 cm ²	0.070	1.091	-6.328e-15	1.162
2	20 cm ²	0.070	1.086	-1.177e-14	1.156
4	40 cm ²	0.070	1.072	1.998e-15	1.142
8	80 cm ²	0.070	1.043	-7.883e-15	1.112
16	160 cm ²	0.063	0.994	9.048e-15	1.057
32	320 cm ²	0.062	0.917	3.331e-16	0.979
64	640cm ²	0.062	0.799	-6.606e-15	0.861
128	1280cm ²	0.017	0.633	-5.662e-15	0.650
256	2560cm ²	0.074	0.326	-2.442e-15	0.399
512	5120cm ²	0.008	0.133	-6.994e-15	0.141
1024	10240cm ²	0.019	0.000	0.000	0.019
2047	20470cm ²	0.000	0.000	0.000	0.000

The Quantity column represents the overall difference in category amounts between the two maps at that specific spatial resolution. This component exposes how one model might predict a larger area of water flow than the other, regardless of where the predicted flow occurs. Quantity is expressed as a percentage where 1 indicates a 100% difference in the number of predicted cell categories between the UE model and the D8 model flow path raster. The relative stability of the Quantity component across multiple resolutions indicates that overall, the two models have a consistent difference in the total area of predicted flow, and this is not strongly influenced by changing the spatial scale until at multiples of 128. For example, at finer resolutions (multiples 1 to 8), Quantity remains relatively stable around 0.07 (e.g., 0.071 at multiple 1, 0.070 at multiple 8). However, it fluctuates at coarser resolutions: dropping to 0.017 at 128, peaking at 0.074 at 256, falling to 0.009 at 512, rising to 0.020 at 1024, and reaching 0

at 2047 meaning both are identical because they have been aggregated to one pixel. This is to be expected, as Quantity should theoretically stay the same when responding to resolution changes because it depends on the total amounts of each category, which should persist under aggregation if proportions are preserved (Chapter 6 in Pontius (2022)). In addition, coarser resolutions result in larger scale aggregations which cause under or over-representations in heterogeneous distributions due to spatial clustering.

The Exchange component reflects locational differences in where water flow is predicted. A high exchange component indicates that the two models are predicting water flow in significantly different areas, even if their total predicted flow area is similar. The component is calculated as two times the minimum number of locational swaps between pairs of cells required to convert incorrect predictions, such as Misses and False Alarms, into Hits. This calculation method means the upper limit of the Exchange component is 2 (or 200%), rather than 1. For example, 1.091 reveals that at the finest resolution there is a large amount of exchange. At finer resolutions, the UE model and D8 model differ significantly in spatial allocation (Exchange dominates at 1.091 vs. Quantity 0.071 at multiple 1). The observed value of 1.091 for the UE model indicates that approximately at least 50% of its cells would need to be exchanged with a pair to match the D8 reference model. As resolution coarsens, these allocation differences diminish, disappearing by resolution 1024, where Exchange and Shift are 0. The shrinking of the Exchange component as resolution becomes coarser shows that some of these locational differences in cells with flow disappear when the analysis is done at coarser scales. This means the near-allocation differences in the locations of predicted water flow are lost when observations are aggregated into blocks, which is to be expected as an effect of aggregation.

The Shift component analyzes any systematic movement of cells. The produced Table 12 does not show a consistent pattern for Shift. The column's values fluctuate between negative and positive with no observable trend, showing a slight decrease at finer resolutions and an increase as resolution coarsens. However, the values are significantly smaller compared to other

components, which suggests that they are contributing to relatively minor spatial differences between the models that disappear at coarser resolutions. The lack of trend may also indicate that the two models do not have many systematic agreements in terms of diagnostic process, which is to be expected due to the difference in methodology.

The Overall component represents the total difference between the two maps at each resolution, and it is the sum of the Quantity, Exchange and Shift components. The Overall differences decrease as the resolution becomes coarser. The consistent, non-zero Quantity value across resolutions, despite the shrinking Exchange and Shift components, shows that there is a consistent bias in the location of predicted flow between the two models. For example, while one of the models consistently predicts a similar amount of more flow than the other, the rapid shrinking of the Quantity and Exchange components at middle-scale (multiples 4 to 8) indicates that a substantial part of the differences between the two models is due to fine-scale disagreements. At the finer resolutions (multiples 1 to 4), the models strongly disagree on the precise location of the flow paths, which can be interpreted as a type of near allocation disagreement. In other words, the result suggests the two models may not spatially match fine resolutions, but do not have systematic, large-scale differences in where they predict flow.

4.4.2.2 Vector Analysis

The vector analysis without aggregation produced Table 13. Path length is described as the magnitude of the vector, displayed in meters and direction as degrees from north.

Table 13. Vector statistics between the UE model and D8 model

Metric	UE Model	D8 Model
Vector Count	653	434
Mean Length (m)	2.358	6.045
Median Length (m)	1.802	3.201

Minimum Length (m)	0.192	0.050
Maximum Length (m)	14.846	71.771
Mean Direction (degrees)	57.694	123.625
Magnitude Quantity Deviation (m)	3.687	
Direction Quantity Deviation (degrees)	65.931	

In terms of magnitude, the UE model contains more vectors overall, but these vectors are noticeably shorter on average. This implies that the D8 model has stronger, longer straight flows. Specifically, both the mean and median lengths in model X (2.358 m and 1.802 m) are less than or half of those in the D8 model (6.045 m and 3.201 m). However, the UE model does have a slightly higher minimum length (0.192 m vs. 0.050 m) compared to D8 model. The maximum length in the D8 model is an extreme value that is significantly larger (71.771 m vs. 14.846 m), which suggests that its vectors span a much wider range of lengths and could be exaggerating flow in some regions. The Magnitude Quantity Deviation represents the difference in the average size of flow, which is relatively small (3.687 m) and suggests that both models may have a slight disagreement on flow lengths.

In terms of direction, the mean direction of the UE model indicates a more northeast orientation (57.694 degrees), whereas the mean direction of the D8 model points more toward the southeast (123.625 degrees). The Direction Quantity Deviation reveals that the difference in the mean directions of flow is large (65.931 degrees), meaning the two models strongly disagree on the predicted overall flow direction.

The spatial differences in magnitude and direction at three aggregation sizes are shown in Figures 5 and 6. Vector length differences did not change in response to larger aggregation areas, indicating that the patterns are robust and consistent across aggregation scales. The Natural Jenks method was used to categorize the intervals for visualization, as it would

highlight inherent trends in the data. Vector directional differences were less consistent in different aggregation scales. These small fluctuations across scales were due to the spatial disagreement between the models on overall trend of flow directions, which is reflected as local variability in the hexagons.

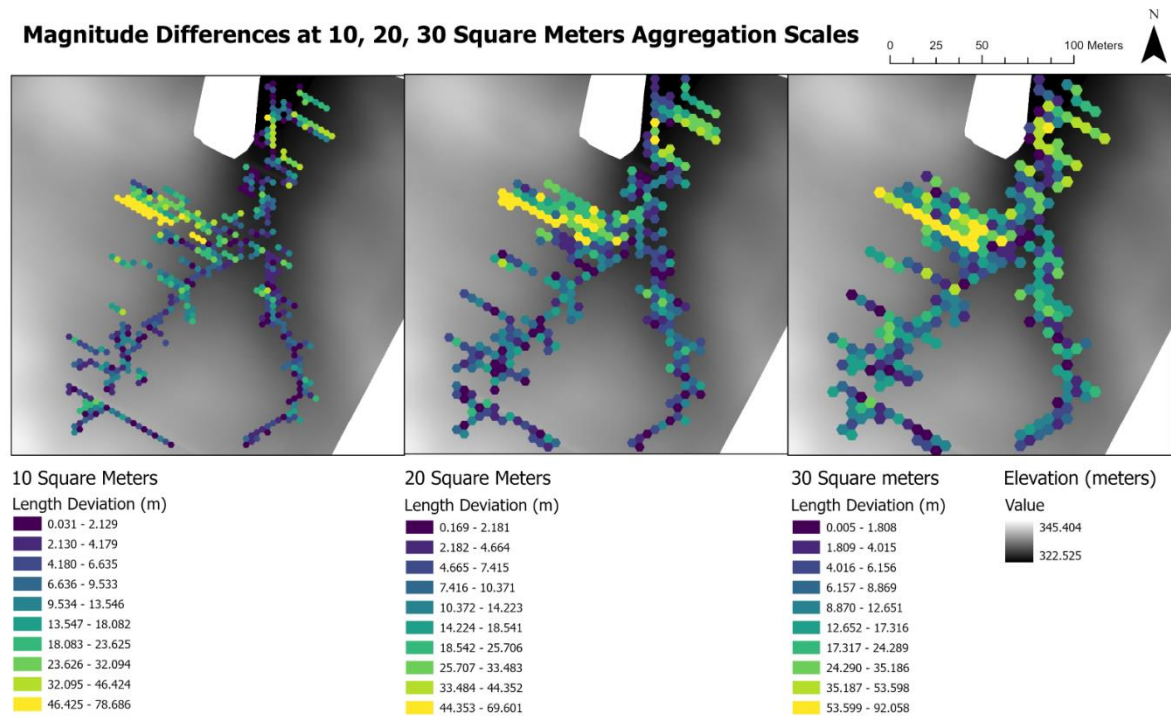


Figure 5. Magnitude Aggregations across scales

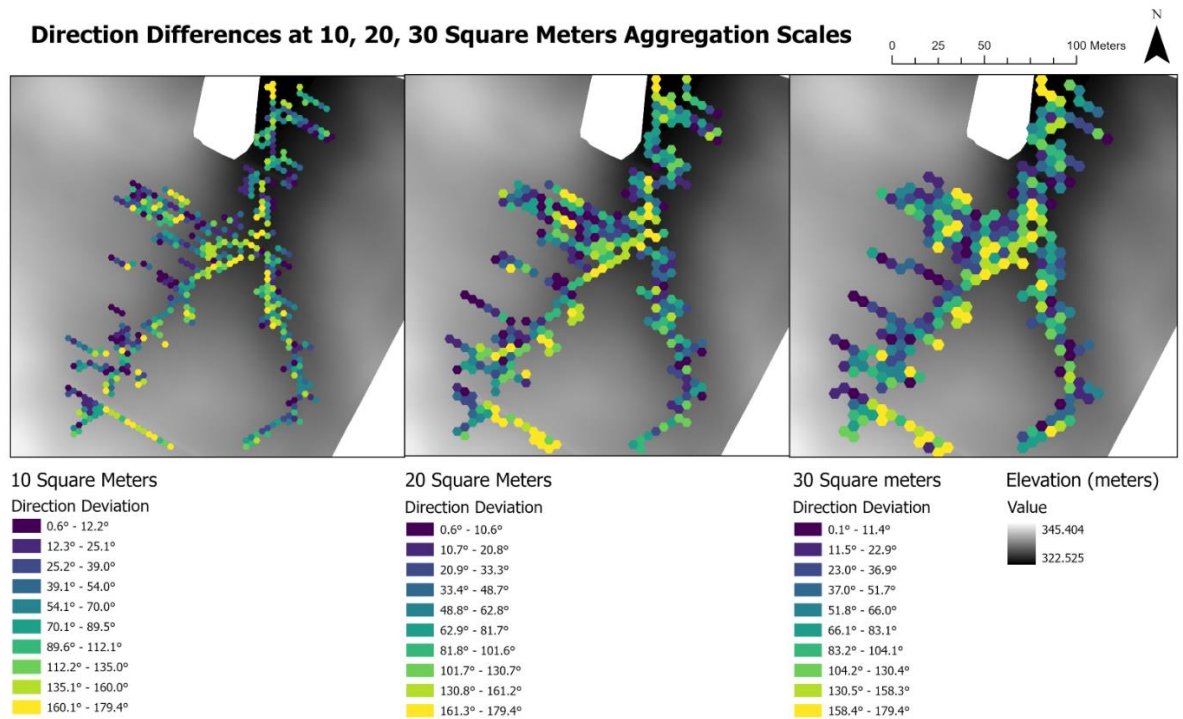


Figure 6. Directional Aggregations across scales

The Allocation Deviation metrics required aggregation, so they have been calculated separately for each aggregation scale. For Allocation Deviation and MAD metrics of magnitude and direction at each scale, refer to Table 14.

Table 14. Vector comparison metrics at multiple aggregation scales

Metric	10 sqm	20 sqm	30 sqm
Magnitude Allocation Deviation (m)	10.212	10.771	12.797
Direction Allocation Deviation (degrees)	9.249	5.873	7.278
Mean Absolute Magnitude Deviation (m)	13.896	14.458	16.483
Mean Absolute Direction Deviation (degrees)	75.180	71.804	73.209

Allocation Deviation reveals how the flows are distributed among different locations. For example, the 10 sqm scale has a Magnitude Allocation Deviation of 10.212 m vs. 12.797 m at

30 sqm, implying that the allocation disagreement grows with coarser scales, which is expected because of aggregation effects. Both models respond to aggregation differently and are sensitive to size, shape and placement of the aggregation zones. The hexagon shape was chosen to minimize the Modifiable Areal Unit Problem experienced by the analysis (Strimas-Mackey, 2020). Smaller aggregation zones do not always capture both paths in the same hexagon area, which leads to missing differences in deviation. When using larger hexagons, paths from both models are captured more frequently, which leads to calculation of more magnitude differences. This shows that while the models might sometimes locally match, once aggregated, the net difference in length becomes more pronounced. Conversely, it may also imply that with finer scales, the deviation decreases, and the two models are in more agreement about the length of the flow paths.

The Direction Allocation Deviation remains relatively low (5.873 to 9.249 degrees) compared to the overall MAD for direction (above 70 degrees). This shows there is a substantial global (quantity) difference in flow direction between the two models, but the allocation component remains relatively small, implying that once a global offset is accounted for, the two models may show a consistent pattern in vector alignment.

The MAD for magnitude shows that the vectors in one model (likely the D8) are consistently longer than the other by more than 10 meters, across all scales. The MAD for direction shows that directional disagreement is consistently high, even when aggregating. This indicates that across the domain the two sets of flow directions are, on average, still off at more than 70 degrees.

While there are some slight agreements between the models on the magnitude of vectors, they are in strong disagreement over what the direction of flow should be. The discrepancy is large enough that, from an application perspective, they imply drastically different flow corridors.

4.5 Discussion

In this discussion, a visual interpretation will be provided focusing on emergent behaviors and patterns that were observed in the model. Then, the model's design choices and trade-offs will be discussed, as well as assumptions made when producing results and conducting analysis. Finally, this section highlights the limitations, emphasizing the need for robust validation strategies and the integration of erosion and rainfall to enhance the model.

4.5.1 Observations

In Figure 8, the resulting flow map clearly delineates primary flow channels across the basin, with the most dominant channel from the south-west moving toward the center of the basin. From this, it can be inferred that erosion is most likely to occur in these high-flow zones. The intensity and brightness at the central convergence point suggest that this region is a likely erosion hotspot due to high energy and flow volume. The UE model produces similar visual characteristics to the D8 method, as seen in the series of parallel linear flow lines running diagonally across the basin. These are consistent with the direction of tillage, which are designed to impose a structure, acting as micro-channels that guide the initial overland flow.

On the east side of the secondary channel, there is a nearly rectangular, leveled plot which appears artificially flattened. This feature is difficult to see in the original point cloud and DEM datasets but exert an influence that disrupts the natural continuity of the flow on the east side of the field, creating a break in the flow path where water may either pool or redirect unnaturally. This plot likely acts as a localized sink or barrier, forcing the water to widen or reroute its flow around the structure. However, this feature does not prominently influence the flow in the D8 approach, as it uses grid-based abstractions and may have reduced its impact because of the transformation and pre-processing in the dataset that averaged out the influence of micro-topography. As a result, the game engine model has a less concentrated flow path on the west side of the field both due to this feature and its wider channel.

The differences observed in the flow pathways between the UE model and the D8 model stem largely from how water movement is conceptualized and computed. As seen in Figure 4, traditional grid-based models assign flow direction based solely on the steepest descent to a neighboring cell, producing deterministic and grid-constrained pathways. This approach is computationally efficient but often oversimplifies real-world variability by enforcing uniformity and limiting lateral dispersion. In contrast, particle-based models in game engines simulate flow as emergent behavior from individual particle dynamics. The UE model inherently incorporates randomness (by seed), chronology, and interaction with terrain microfeatures, leading to more organic, irregular flow paths that often resemble natural water movement more closely. For example, the path of the particles moved in a sinusoidal motion downstream, travelling up slopes and overcoming local microtopography in tills to form bonds with other particles and create local pools.

The D8 model was prone to creating unnaturally uniform and straight pathways where the variability of the gradually decreasing elevation is not considered. This may be an explanation of the disagreement in flow direction between the UE model and the D8 produced in the results and analysis. As a result, flow paths generated in game engine simulations may provide a closer approximation to reality, especially in heterogeneous landscapes where microtopography could play a key role in channel development.

The UE model introduced emergent behaviors that were not accounted for in the D8 model. For example, one phenomenon emerged due to adjusting the size parameter of the particles, as they interacted differently with the terrain microtopography. Smaller particles are more prone to becoming trapped in tilled soils, as their reduced size increases the frequency of collisions with the uneven ground. This accumulation illustrates an emergent form of friction, because the particles require more force to overcome these irregularities due to their higher collision frequency. As the simulation progresses, these initial sinks become gradually filled, allowing newer particles to slide over the deposited matter. This process creates an evolving flow path which periodically appears and disappears, contributing to channel formation. In contrast,

setting particle sizes to be larger provides them with the tendency to bypass micro-scale irregularities altogether. They collide less frequently with the terrain, thus experiencing a lesser degree of friction and leads to smoother flow patterns that remain consistent over time.

Another emergent behavior arises from the fluid properties incorporated into the particles. Pressure within the simulated fluid can occasionally propel particles in unexpected ways, such as causing them to travel uphill or in directions that traditional models cannot anticipate. There is path dependency due to the dataset already containing the shape of flow channels formed from past erosion events, influencing formation of potential future pathways. This causes the UE model to react to the channel geometry differently at various locations, especially in long continuous paths where water is expected to gain momentum downhill and overcome obstacles.

Increasing the max force and velocity of particles or increasing delta time in the UE model created more chaotic movement in particles, causing them to be propelled off the sharper edges of local geometric features, creating splashes in other flows. In addition, the viscosity of the particle fluid plays a significant role in group behavior. Higher viscosity causes particles to lose velocity together, inhibiting their movement down a clearly defined stream channel. Over time, this collective bonding force can result in the formation of small pools where particles accumulate, altering the local collision conditions for other particles. Therefore, unlike the D8 model, the UE model does not assume all water movement are steady-state flows always travelling to lower elevations. This can be considered an advantage of the UE model approach, as it accounts for realistic phenomenon that are not observed in the D8 model.

4.5.2 Computation Expense

Traditional 2D methods like D8 are computationally lightweight, requiring minimal resources as they process a static grid without iterative physics calculations. The UE model, however, faces scalability challenges in terms of computation cost due to its voxel-based terrain representation and particle-based fluid simulation. For example, when initializing the

Neighbor Grid 3D, limiting the "Max Neighbors per Cell" parameter to 6 (instead of an ideal 26) reduces computational load but sacrifices accuracy in detecting edge cases, which is representative of the many trade-offs needed to ensure the simulation can successfully run.

Despite expensive computation costs, the UE model offers trade-offs that traditional methods cannot provide. For example, the model enables real-time fluid simulation and dynamic terrain modification, allowing particles to alter the terrain during runtime. This offers immediate feedback and parameter adjustment (e.g., particle number, voxel resolution), which provides a base environment for iterative testing. A comprehensive set of adjustable parameters empowers users to tailor the simulation to a diverse array of scenarios, ranging from gentle water flows to intense erosive events. Key parameters influencing fluid behavior and terrain interaction include particle mass, pressure scales, density, and viscosity which collectively dictate the fluid's movement and resistance properties. Additionally, collision-related parameters such as particle radius, restitution, and friction refine how particles interact with the environment with precision.

4.5.3 Model Design

The design of the game engine model maps to the challenges discussed in Chapter 2: static models, detached visualizations, passive interfaces, and immobile validation strategies. Figure 7 shows the relationship between the pillars and the challenges and the technologies that support each pillar. Specific capabilities and technologies of UE that address these challenges are demonstrated through the workflow: NSF, Voxel Plugin, Blueprint, and exporting the raster to ArcGIS.

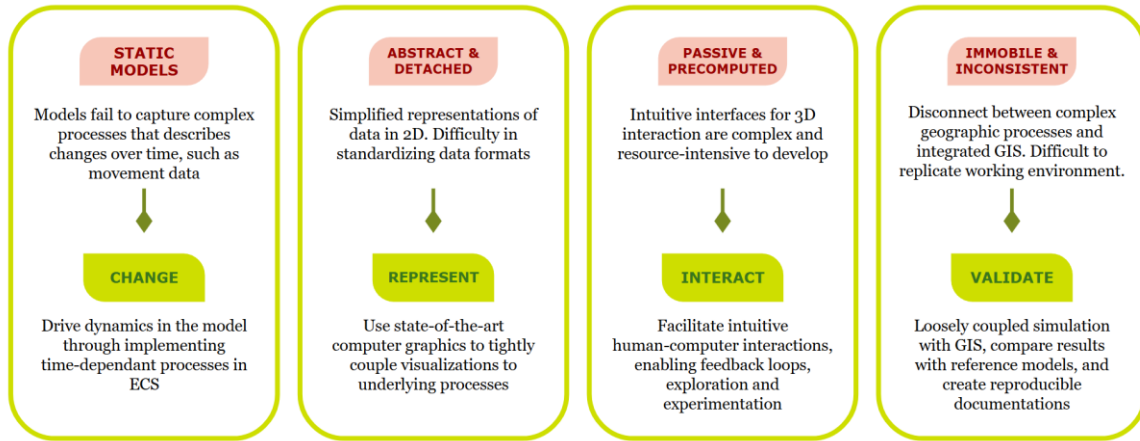


Figure 7. Mapping challenges to conceptual pillars

The interpretation and application of the simulation stages and modules in NSF as an ECS paradigm drives dynamic behavior for particles that can be controlled on multiple levels, driving dynamic changes to the terrain surface by exporting data to Blueprint. This decision allows the main subject (water flow) to be scalable in the model, due to its reliance on GPU processes. The advantage is shown by the minimal compute load required to simulate a significant number of particles, with a simple application and interface that does not require lower-level knowledge of how ECS or particle systems such as NSF are implemented. The responsibility of the modeler can be focused on recreating realistic dynamics between Entities.

The behaviors for particles are captured by the modules (as Components) of the flow process, which describes how they move through the terrain. Unlike most traditional flow models, the particles representation does not assume a steady state flow due to the acceleration and friction experienced along paths of varying slopes, as well as their interaction with other particles. Thus, the model creates more realistic scenarios that may influence the resulting direction and length of paths.

The decision to utilize a voxel-based terrain representation shows how state-of-the-art computer graphics technology can become accessible through plugins. It has an advantage over simplified representations of elevation models by being in 3D and allowing dynamic

modifications during run-time. A voxel-based representation was chosen over the UE default vertex-based system by the need for an accurate recreation of the DEM to preserve data integrity, as well as to lay the groundwork for a dynamically modifiable environment. Voxels provides a foundation for studying dynamic landscape evolution and environmental interactions, with optimization strategies such as Marching Cubes for mitigating rendering and performance bottlenecks. This was essential for simulating processes like erosion that frequently alter the terrain in real-time.

Voxel based representations allow for efficient modification operations, which facilitate the creation of new flow pathways through particle interactions. In contrast, vertex-based representations are slower to modify in real-time, as they require re-calculation of geometry and surface details (e.g., triangulation, normals, UVs). This could quickly lead to issues in scenarios such as eroding flow paths, where the spatial locations of modifications are not evenly distributed on the surface, thus causing dense vertex clustering and bottleneck the calculation. The voxel-based approach relies on localized voxel edits and would only need to process data of cells that should react to change, which better enables emergent flow pathways that evolve during the simulation, replicating real-world hydrological processes. However, if the goal ends at achieving simpler tasks such as generating basic flow paths over a static surface, the voxel method proves excessive, as it introduces significant computational overhead compared to alternatives such as the vertex-based terrain representations.

The degree of interaction available in a game engine presents a significant advantage over static models. The model provided a higher degree of interactivity through the terrain modification tools by Voxel Plugin, customizability of particle properties, and visualization properties (e.g., environment lighting, surface rendering, particle size). In addition, the available tools can be used to define custom interaction rules, which might influence the flow during a simulation. For example, a key can be assigned to pause or progress the flow of simulation time. These interactions can change how a modeller designs the model to be

presented to an audience and offer a richer set of tools for communicating knowledge by building experiences.

The simulation from the UE model serves as a demonstration of equifinality. Users can adjust variables such as number of particles, voxel resolution, and fluid parameters to explore how different initial conditions and settings produce similar flow path patterns. For example, lowering the number of particles allows the shape of the flow to form the same networks but takes more time, or changing the voxel grid's resolution which still produces similar outcomes but with a coarser path. The model is sensitive to any factor that may affect the physical interactions between the particles and ground, or fluid properties which can lead to a chain cause-and-effect on various scales but ultimately lead to the same flow patterns forming if given realistic parameters.

The inclusion of physics provided by SPH in a voxel model potentially creates a powerful and dynamic way for conducting tributary analysis of how paths form and evolve over time when erosion is included in the simulation. Unlike traditional 2D methods, which rely on static terrain elevation data to predict water flow directions, the 3D model simulates real-time interaction between water and terrain, allowing researchers to observe tributary formation as it happens. This approach allows physical mechanics to drive the emergence of new channels or the shifting of existing ones, which static 2D models often oversimplify or miss entirely. An advantage of the 3D model is also its inclusion of depth to influence erosion, where flowing water vertically displaces soil particles and reshapes the terrain, directly influencing the development of new paths. This capability makes it possible to explore natural processes like river system evolution or the creation of features like oxbow lakes. In addition, through comparing simulations with and without erosion, researchers can assess relationships between flow intensity and terrain changes.

No extra visualization was applied to render the particles as a cohesive fluid. Although established techniques exist to create realistic water effects, often using blurring to interpolate between particles and render the illusion of a continuous water body, such methods were

deliberately excluded from this implementation. The rationale behind this decision was twofold. First, applying these techniques could compromise the accuracy of tracking individual particle positions, which is critical for precisely evaluating the flow path. Instead, sprites were used to ensure that the exact positions of particles were maintained for robust analysis of the 2D flow path export. Second, the additional computational cost imposed on the GPU by such advanced rendering methods was considered unnecessary, as it would not contribute meaningfully to the accuracy or quality of the simulation output.

The resulting flow raster can be exported to validate against other models in ArcGIS. This is an example of one approach that enables validation of game engine models by integrating them with existing GIScience frameworks. However, as the analysis compares results from a dynamic model to a static one, it is difficult to establish criteria for an equal comparison. Static models do not consider the temporal aspects of processes and how they converge, which may be lacking in depicting the realism of the result. Therefore, validation for game engine models can also be dynamic in that even the same parameters create different results depending on the time of simulation. The expectation is that when given longer simulation time, the model's results will stabilize and reach a state of saturation, which produces more consistent results.

4.5.4 Assumptions

The methods employed in this model rely on the core assumption that all processes are visible to the camera, since rendering is a key influencer of the results. This is because the performance-driven design relies on GPU-based collision tracing for particle systems. This view-centric method requires particles to be visible in the viewport and the terrain to be within a certain proximity to the camera, as non-visible processes are ignored for optimization in the algorithm. For example, one cannot always assume that the terrain's surface details are consistently represented due to Level of Detail (LOD) adjustments. When the terrain is sufficiently distant from the camera, the engine simplifies the voxel terrain to a lower resolution version, consequently reducing the fidelity of surface details and potentially compromising the accuracy of collision detection for particles. While this automatic

adjustment is beneficial to maintaining efficiency of computation, it is problematic for purposes that prioritize the accuracy of results.

The methodology used for simplification during validation, based on the assumption that greater stream order corresponds to wider channels, may not be widely applicable to many scenarios. For instance, the certainty of a flow path existing in the model can greatly vary depending on the total time of simulation. If enough particles were allowed to flow, then even areas with relatively less overlap can cross significant thresholds to imply that flow exists in small quantity there, because the threshold was defined as a static value with no relationship to the time of simulation.

It is important to distinguish strong and weak flows, and the threshold approach may discriminate against the existence of weaker flows due to lesser contrast between weak and no flows. This threshold value was also decided based on visual data exploration and thus is not a robust method of determining the validity of the existence and strength of flow. Additionally, the width of the hydro polygon does not necessarily correlate with the stream order. This assumption is based on the study area being a relatively flat farm field and assuming the flow being surface run-off. In other sites, the depth of channels can play a significant role in depicting the stream order. However, depth information is often not available in a 2D polygon, and the quantity of flow does not necessarily match the shape of the channel, thus the polygon width is not reliable for assuming stream order.

The degree of filtering and simplification to allow a comparison between the two models were also assumed to be based on visual assessment (e.g., threshold to remove lower-stream orders). The D8 model's sinks were filled, impacting the integrity of the dataset to better allow for its algorithm to produce continuous paths, but the data imported to UE was not preprocessed in any way. This may have impacted the final number of vectors that were used in the analysis, which can skew the magnitude and direction of the vector analysis as well as the allocation comparison in the raster analysis. However, the outcome may still be the same. For example, most of the filtered paths in the D8 model exhibit similar properties to the full dataset (long,

straight flows), and this may have only caused more bias in directional data. The filtered paths in the UE model would also not have been critical in causing a difference, as the paths were mostly in and around the same area as the main flow but significantly shorter. Thus, their inclusion may only introduce noise that was not contributing to the comparison between the main pathways.

Further, due to various technological and conceptual limitations, the representation of water is simplified as spheres and referred to as particles. The assumption is that they would be the closest physically realistic portrayal of water movement when collisions are applied. This simplified representation can be a factor that influences future processes such as erosion, since the points of collision between a sphere and ground can be sparse and non-uniform. For approaches where the contact point serves as an event trigger, it should only serve as a reference point for where erosion should generally happen.

The use of the D8 method for validation of the model is an assumption that the results from D8 represent a credible result which mirrors empirical data. However, this is likely not accurate nor representative of ground truth. As such, the insights from the results of the comparison should only be applied in the context of understanding how the two models compare to each other, their agreements and disagreements, and not whether one model is closer to reality.

The current particle distribution patterns employed in the simulation were generated randomly through offsetting particle spawn locations from spawn points aligned in a grid, which is not a realistic assumption of rainfall patterns. While this approach is computationally straightforward, it fails to capture the spatial and temporal complexity inherent in natural meteorological conditions. A more realistic approach would involve the integration of a meteorological model, thereby enabling the joint modeling of atmospheric and hydrological processes. Such an integration would require modifications to the Grid Location module and adjustments to the spawning behaviors in both the Emitter and Particle Update stages within the NSF framework.

Finally, it is also assumed that the flow during simulation does not alter the landscape, and that all voxel cells carry the same data (e.g, uniform soil quality). This means processes such as erosion are omitted to allow a stable and consistent environment for the simulation to occur, saving computation cost but potentially decreasing realism. However, the use of a static landscape implies the effects of path dependency as the ground has already been eroded to form preferential paths from past flows. The D8 model by design is deterministic due to its static flow direction-based algorithm. The UE model also uses a static landscape, but due to more stochastic behaviors in the particle spawn and interactions, different paths may be formed on each batch of simulation. The particle system offers the choice for particle spawn seeds to stay the same for each run. In this methodology, a deterministic approach was chosen. Thus, the D8 and UE model may have only been able to capture the flow state at a single point in time.

4.5.5 Limitations

Despite their advantages, the implementation of physics-based models presented significant challenges in efficiency. An initial ambition during the model's development was the integration of dynamic erosion mechanics, aiming to simulate the complex interactions between fluid dynamics and terrain morphology more comprehensively. The envisioned system involved a larger number of interactive particles (targeting over 1000) influencing a sufficiently high-resolution voxel terrain (2048x2048), driven by realistic fluid behavior. However, incorporating the necessary erosion logic into this already demanding simulation framework presented a series of significant, and ultimately insurmountable, technical and conceptual challenges. This section reflects on these challenges, detailing the specific hurdles encountered and the iterative attempts made to overcome them within current technological constraints and personal ability.

A critical bottleneck was identified in the data transfer pipeline between the NSF, responsible for the parallelized particle simulation, and Blueprint, tasked with processing the erosion logic

and updating the voxel terrain data structure. The intended workflow demanded a continuous, high-bandwidth exchange of information, summarized in three steps:

1. The Niagara particle emitter was configured to export data for particles interacting with the terrain (e.g., position, velocity, potential sediment load) at the end of the Particle Update stage. This data was intended to trigger events processed by a Blueprint script on the CPU.
2. The Blueprint script receives the particle data, calculates the corresponding terrain modification (erosion or deposition) based on predefined rules, updates the relevant voxels in the terrain grid, and potentially calculate updated properties for the particle (e.g., increased sediment load).
3. The updated particle information needed to be relayed back to the specific particle within the Niagara system on the GPU, influencing its subsequent behavior.

This cyclical data flow, repeated every frame, created significant performance issues, primarily related to the efficiency and capacity of the GPU-CPU data bridge within the Unreal Engine architecture.

4.5.5.1 Export Limitation

The Unreal Engine's Niagara system, at the time of development, imposed strict limitations on the structure and volume of data that could be exported per particle via its event-handling mechanism. Specifically, the system permitted the transfer of only two Vector3 variables and one float variable per particle event. This limitation proved severely restrictive for the intended erosion model. Accurately simulating erosion required passing a richer dataset for each interacting particle. To overcome this limitation several parameters were chosen to be passed at minimum:

- Position: (Vector3) Location of the particle-terrain interaction.
- Velocity: (Vector3) Magnitude and direction of the particle for calculating erosive force.

- Sediment Load: (float) Amount of sediment currently carried by the particle.
- Particle ID: (int32) Tracking individual particles and ensuring updates were applied correctly back on the GPU side.

Attempts were made to circumvent this limitation through data encoding techniques. For instance, efforts were made to decompose Vector3 data into individual float components or to utilize the decimal places of float variables to store encoded information. However, these workarounds proved unviable. Critical data like position and velocity require high precision, which was lost during the encoding/decoding process, rendering the data unreliable for accurate physical calculations. Overcoming this fundamental engine limitation would have necessitated modifications to the engine's source code, requiring deep engine-specific expertise beyond the scope of this project and introducing significant complexities regarding engine updates and maintenance.

4.5.5.2 Congested Pipeline

The second stage of the workflow, processing particle interaction events and modifying the voxel grid on the CPU, encountered further obstacles. As the number of simulated particles increased, the sheer volume of interaction events generated per frame overwhelmed the CPU's capacity to process them in real-time. The Blueprint system, while powerful for scripting, struggled to handle the high frequency and volume of incoming data triggers. This resulted in processing delays, where the particle simulation on the GPU continued to advance while the CPU lagged in updating the terrain. The desynchronization led to simulation instability, drastically reduced frame rates, and a failure to achieve the intended real-time feedback loop between particles and terrain. Attempts to mitigate this by slowing the particle simulation rate or reducing particle counts provided only a temporary relief; as soon as sufficient particles generated simultaneous events, the CPU bottleneck re-emerged.

A conceptual and implementation issue also arose in how erosion was calculated. Initial logic tied erosive force directly to particle velocity. However, particles spawning and falling onto

the terrain often possessed high initial velocities due to the fall height and gravity forces, this splash erosion caused disproportionately large forces upon first impact. This led to an unrealistic burrowing effect, where particles rapidly carved deep pits at their initial contact points, rather than simulating more gradual surface erosion. Several corrective measures were explored in response to this phenomenon:

1. Introducing a maximum sediment load per particle, allowing its mass/erosive potential to increase only up to a threshold.
2. Implementing a concurrent deposition process, where particles would also deposit sediment based on factors like velocity and current load, aiming to balance the erosive action.
3. Reducing the initial drop height to lower impact velocities.
4. Applying smoothing to distribute the terrain modification over a wider area, reducing the sharpness of the depressions.

Despite these attempts, achieving a stable and visually plausible erosion behavior that avoided the burrowing artifact proved difficult. Fine-tuning the parameters often led to new issues appearing in other aspects of the simulation, performance issues, or other unrealistic outcomes. For example, once the simulation was under heavy computation load, particle collisions began to behave abnormally, such as ignoring collisions with each other and the ground.

The final step of the workflow needed to relay updated particle information (e.g., sediment load) from Blueprint back to the correct particle instance within the Niagara simulation on the GPU, which presented its own set of complex challenges. The proposed method involved constructing data structures (e.g., arrays) on the CPU mapping Particle IDs to their updated states. This data then needed to be efficiently transferred back to the GPU and used to update the corresponding particles. However, ensuring that the data arrived and was applied to the correct particle at the correct point in its simulation lifecycle, without interfering with other ongoing particle calculations, proved extremely difficult. The parallel execution on the GPU meant that particles were being simulated concurrently within a frame, leading to potential race

conditions where data might be read or written at inconsistent times. Implementing logic within the Niagara system to search for the specific particle ID, check its status and state, apply the update from the CPU, and storing the new data added significant computational overhead to the GPU simulation itself. This further strained GPU resources already taxed by the fluid dynamics calculations. Effectively managing these synchronization issues and efficiently implementing the update logic within the GPU simulation required advanced knowledge of parallel programming paradigms and Niagara's architecture.

4.5.5.3 Additional Limitation Testing

A frequent limitation encountered during development stems from hardware constraints, specifically the available VRAM in the GPU. Exceeding VRAM capacity led to system instability and crashes, impacting both the environmental representation and the particle simulation components. The most significant performance bottleneck identified was the voxelization process used to represent the study area geometry. Attempts to increase the environmental detail beyond an octree depth of 6 (resulting in a 2048 cubic squared voxel resolution) consistently exhausted available memory resources, preventing the simulation from running with higher-fidelity terrain. This voxel count and the overall spatial extent of the study area, therefore, represent a hard limit on the scale and resolution achievable with the current hardware setup.

While the utilized voxel plugin supports importing large datasets, rendering these extensive environments in their entirety poses a significant challenge due to the aforementioned VRAM limitations. To manage this, the plugin allows for the definition of specific view boundaries, enabling only portions of the dataset to be rendered at any given time. This approach permits the use of large-scale environmental data, albeit without simultaneous visualization of the complete area. For the specific datasets used, which were considerably wider (X/Y dimensions) than they were tall (Z dimension), a notable performance improvement was achieved by aggressively limiting the rendering boundary along the Z-axis. This optimization reduced the rendering load without sacrificing critical vertical details near the terrain surface,

effectively tailoring the rendering strategy to the dataset's aspect ratio. However, this step also implies that there is a limit to the amount of voxel modification in the Z dimension.

The particle system itself faced constraints, with performance considerations necessitating a limit on the maximum number of concurrently active particles and the size of particle emitters (set to spawn in a grid of 20x20x1 units). While increasing the particle count had a less severe computational scaling cost compared to increasing voxel resolution, stability issues arose specifically concerning collision detection. The reliability of particles correctly interacting with the voxel terrain was found to be highly sensitive to the visual scale of particles and the simulation frame rate. At higher particle counts, the GPU-based tracing method for collision struggled with the sheer number of potential simultaneous collision events, sometimes resulting in particles erroneously passing through surfaces if they weren't visibly intersecting within a frame. This behavior is exacerbated by lower frame rates. To counteract this instability, particularly under fluctuating performance conditions, the particle simulation's delta time could be adjusted, effectively slowing the particles' movement speed to allow more intermediate frames for accurate collision processing between position updates. However, this also extends the total time of simulation, while also influencing the calculation of fluid forces with respect to changes over time.

4.6 Conclusion

A model was developed within UE demonstrating the potential for game engines to act as a tool for simulating hydrological processes using CFD. Real-world elevation data was imported into the game engine and represented as a dynamic voxel terrain. A combination of PBD and SPH was employed to model surface flow, which produced a flow path map. The data from the simulation was effectively converted into a 2D image format, exported from the engine, and subsequently imported into ArcGIS. Within the GIS environment, a quantitative comparison was performed against flow paths generated using the traditional D8 algorithm.

The results indicate that the UE model was capable of more advanced simulation of flow paths, as it produced an accurate output for the number of cells that contain flow in the raster analysis, but the allocation of flow was different from the D8 model. The probable cause of this difference arises from the UE model being able to consider more realistic physical factors than D8. This finding was supported in the vector analysis producing similar total length of flow, but different overall directions of flow.

The UE model was not viable overall due to its low level of alignment with the FAIR principles, mainly due to poor accessibility. Specifically, the use of UE to model flow path is not yet viable for non-technical user groups, as it demands significant development time and knowledge to effectively build and modify models in the absence of high-quality documentation. Leveraging the advantages of UE also demanded a level of expertise in multiple engineering domains (e.g., software, graphics, and relevant scientific disciplines), whereas ArcGIS Pro is less transparent, but its simpler workflow may be more practical to users who value ease of use over the ability to deeply customize model behaviors.

This methodology was also not viable for achieving high frequency voxel modifications on the CPU based on particle events from the GPU, as there was insufficient expertise, documentation, as well as financial resources to overcome bottlenecks and achieve the desired performance and effect. More powerful simulation (e.g., more particles, higher voxel resolution, larger study area) may require hardware configurations beyond an average consumer budget (\$2000). The availability of expertise may also contribute to lowering accessibility issues for other user groups, through designing a potential application interface for models outside of the engine environment.

Finally, the affordability of using UE is significant compared to the licensing model of ArcGIS Pro, which is an advantage for users that value lowering long-term financial cost and can afford longer development periods. The proposed methodology demonstrates how game engines

might be used in combination with conventional tools (e.g., ArcGIS) for GIScience at a low cost, despite the significant technical challenges that limit more complex, coupled simulations.

4.6.1 Future Opportunities

Future research may explore the different methods of analysis that can be conducted in game engines. For example, capturing snapshots of the flow at various timeframes, thereby enabling a temporal dimension in the comparison that considers change. This methodology can also be extended by changing the underlying terrain dataset, which was collected at different real-world times, to simulate changes due to erosion events albeit with large skips in time. This approach would incorporate a temporal resolution component, such as time sliders, to advance or backtrack simulation results and create distinct layers for filter comparisons (e.g., before and after specific events). By analyzing these temporal snapshots, researchers can evaluate how changes in the terrain affect flow dynamics over time. This analysis method could facilitate the identification of trends and critical transition periods, enhancing the robustness and applicability of the model in both predictive and diagnostic contexts.

Erosion was omitted from the final model presented in this chapter, but the exploration process may have provided insights into the practical limits of current simulation capabilities of game engines and highlighted promising, albeit challenging, avenues for future research in computational geomorphology and environmental simulation.

Chapter 5. Synthesis

5.1 Introduction

The preceding chapters have explored the intersection of game engine technology and environment science, particularly within the domain of Geographic Information Science (GIScience). The investigation began in Chapter 1 and 2 by identifying the significant yet underutilized opportunity of game engines for scientific visualization, simulation, and analysis, which established a need for greater understanding of the potential of game engines and development of methodologies different from conventional Geographic Information System (GIS) toolsets. In Chapter 3, a conceptual framework was developed to contrast traditional GIScience modeling paradigms with the dynamic capabilities inherent in game engines, advocating for novel approaches to representing time and process effectively. This framework's principles were grounded in addressing limitations of static models and aligning with calls for a more dynamic geographic inquiry. In Chapter 4, a case study was presented using a particle system within Unreal Engine (UE) to develop a dynamic flowpath mapping approach as an example of the concepts advocated for in Chapter 3, illustrating the tangible application of game engines for modeling hydrological processes. Having laid the groundwork, developed conceptual guides, and demonstrated practical feasibility, there is now a clear need to synthesize the collective insights from the previous chapters.

5.1.1 Objective

This chapter aims to interpret and contextualize the findings presented in previous chapters by discussing how the utility of the model in Chapter 4 addresses the body of literature describing challenges of GIS, as well as how the model aligns to the concepts of each pillar in the framework from Chapter 3. This discussion extends to reveal the possible broader implications from theoretical to practical perspectives and finally outline promising future research directions that emerge from the complete body of work presented in this thesis. Thus,

this chapter argues that game engines are indeed a feasible tool for the purpose of aiding GIS in building dynamic models, realizing a portion of the opportunities discussed in Chapter 1 and 2, but the implementation in Chapter 4 has not overcome all the challenges yet revolving around representations of temporal and dynamic processes in GIScience.

5.2 Discussions

The core functionalities selected for use in the model from Chapter 4 align with the four conceptual pillars. The frame-by-frame advancement provides a discrete temporal basis for simulating *Dynamics*. *Representation* is achieved through the tight coupling between the rendering systems, employing voxels, materials, and shaders that can dynamically respond to changes. *Interaction* is facilitated through integrated input interfaces and the capacity to build custom controls, although the initial implementation may not fully exploit the interactive potential. Finally, the engine supports *Validation* through immediate visual feedback, the capability to export simulation outputs, and a contained environment of established workflows that enable replication of research pipelines.

5.2.1 Dynamics

The model directly aligns with the need for tools embracing temporality and spatial dynamism by simulating fluid flow over spatially accurate, drone-captured terrain. It is embedded equation-based dependencies (Computational Fluid Dynamics [CFD] solver operating on principles such as NSE) within a game engine, offering a unified platform for the integration of multi-disciplinary approaches (Ricker et al. 2020). This setup moves beyond static representations, actively modeling and visualizing the flow of water particles, an inherently dynamic process involving continuous change in movement. Parameters are provided to enable temporally relevant investigations of the model, such as allowing change of delta time, the spawn patterns of particles, and the formation of the flow paths on a 2D plane.

The dynamic nature of water is encapsulated in Niagara Simulation Framework (NSF) by structuring fluid simulations into discrete, sequential simulation stages, each with a specific responsibility and controlled data access via namespaces. This separation of concerns, mirroring the ECS pattern, treats particles as entities, their properties (e.g., position, velocity) as components, and emitters as systems, enhancing maintainability and readability. The modularity enables researchers to extend or modify simulations with great flexibility, such as adding a turbulence stage, or spawn additional particles representing detached sediment, demonstrating the pillar focus on model evolution. Namespaces enforce disciplined data access, ensuring stages only interact with designated properties in the right temporal context, which supports internal interoperability by organizing data flows and facilitating integration of complex simulation components.

5.2.2 Representation

This model enhances representation by visualizing quantitative CFD outputs such as velocity vectors or pressure fields on the particles. The visualization itself offers an intuitive representation of flow patterns, and the evolution of flow paths can be observed over multiple frames and iterations. Parameter changes are immediately reflected in changes to the output, presenting a richer representation of the phenomenon compared to traditional, static algorithms in GIS. Any property that is calculated and stored on the particles can be visualized. Lastly, the primary emitter provides a specialized rendering stage where additional complexity in visualization can be applied, leaving space for future extensions in the model.

5.2.3 Interaction

While the model output is a 2D map, the engine provides the foundation for intuitive, interactive 3D visualization, potentially lowering barriers for domain experts to explore the flow dynamics directly within the virtual terrain. This includes the ability to terraform the voxel landscape so that new flow pathways can be created during simulation, and the voxel dataset can be freely sculpted to create hypothetical situations. The engine acts as a controlled space

for the simulation, and the visualization provides immediate feedback for model interactions (O’Sullivan, 2015). This setup facilitates a more exploratory approach to navigating datasets, where users can visualize the results of complex computations within an accessible graphical environment. Thus, the future extension and refinement of models can happen within this development feedback loop.

5.2.4 Validation

The model supports validation through the adjustment of parameters, Digital Elevation Models (DEMs), and allowing the export of flow path at any point in time during the simulation, for comparison with traditional tools. The model's capability to import DEMs and export the generated 2D flow map exemplifies the framework's emphasis on interoperability and data transfer. By outputting the simulation results into a transferable format, the model provides a well-defined interface for external systems. This allows the quantitative flow data to be readily exchanged and integrated with other analytical tools for further analysis, calibration, or use in subsequent models. The logic that governs the flow of water is independent of the study area and thus can be mobilized to any terrain at various scales.

The ability to reduce the movement data to 2D paths, drawn over time, enables direct comparison with observed data or outputs from established GIS tools for quantitative validation. The generalizable logic of flow can be applied across various scales and terrains and can be used in a modular method to validate results when given a DEM input. Additionally, the reproduction of research is simple as all components used in the model are modular. The modules, emitters, plugins, and the project file itself can be packaged and transferred, either by physical devices or uploaded to a cloud storage for download. The project can be hosted on services such as GitHub and be accessed by any machine with a compatible version of UE. A build can also be made to ship the project for any platform, but consideration must be given for the availability of rendering and compilation methods on target devices, as well as implementation of appropriate controls and distribution pipelines.

5.2.5 Literature Context

The workflow using game engines to produce analytical data for ArcGIS bridged the ambitious promises of geocomputation by enabling the marriage of theoretical insight and operational usability in the tool, discussed by Harris et al. (2017), and evolves to accommodate interdisciplinary needs as an answer to Gahegan (2018). It provides a platform to couple complex processes with agents (An et al., 2021), geo-visualize data dynamically (Ragia & Fiume, 2007), provide a structure for model reproducibility required to advance the science creatively (Elwood, 2010; Wilson et al., 2021), and is structured in a way that balances the key tensions described by Shannon & Walker (2018).

By applying the framework in Chapter 3 and implementing the game engine model in Chapter 4, the lessons learned can be re-positioned in the context of established geographic theories. This interdisciplinary synthesis bridges the philosophical to the practical, in an attempt to further answer the challenges identified by interpreting the results from the game engine model.

5.2.5.1 Relational Space

Through physics simulations, object interactions, and event-driven systems, the game engine model represents geographical space as described by Meentemeyer (1989), where relationships and processes define the spatial context. The contrasting views of absolute and relative space, as well as their hierarchical relationships can coexist within modern game engines. The relationship of actors within the scene of the game engine primarily operates on an object-oriented paradigm but are also compatible with this hierarchical perspective. Each object in a game engine is stored in Euclidean space, devoid of inherent meaning. It is the geographer's role to establish and extrapolate the relationships between objects and their peers, parents, or children. In game engines, objects can be flexibly defined and imported, emerging from measured data or as a result of processes or phenomena. Rendering techniques facilitate

their visualization, while animation or simulation enables the interpolation of attribute changes over time.

Integrating Meentemeyer's (1989) process-oriented relative space with Hägerstrand's (1981) focus on individual trajectories over time provides a geographic lens closely paralleling a physicist description of motion relative to reference systems. This approach emphasizes understanding how change occurs through interactions and relative movement, rather than mapping static patterns. This theoretical alignment strongly validates the methodological choice of using particle systems for modeling dynamic geographic processes such as fluid flow, as it provides a computationally tractable and cognitively intuitive way to represent complex, relational dynamics that are central to both modern geographic thought and physics.

Particle systems, which consist of emitters that generate and control numerous individual particles, offer an alternative to the field-based conceptualization of data and aligns with the goals described in Couclelis (1992). Modelling of flow as a collection of particles provides an intuitive analogy to the molecular structure of water that simplifies the understanding of complex fluid dynamics and is closer to human cognition, though at a much larger scale due to current hardware limitations. This particle-based approach bridges the gap between abstract data and human perception, enabling users to visualize and interact with water flow in a way that feels natural and cognitively accessible.

The hierarchical structure inherent in data types and design patterns in game engines is structured in a highly object-oriented manner, which underscores the presented modeling approach. Particle systems are often implemented as components attached to game objects, allowing them to inherit properties and behaviors from parent objects while retaining their own unique attributes. This object-oriented view supports the separation of concerns in implementation, enabling creation of complex simulations where particles can dynamically interact with other scene elements, such as terrain or other particles, in a computationally efficient and manageable way. The traditional field structure can coexist as parameters assigned to objects by their ID but does not limit the dynamic nature of the various methods of

assigning and accessing data that can be associated with particles. An example of this is apparent in the implementation of the neighbor grid structure, as well as tools that can influence the simulation based on locations of particles (e.g., event-driven logic). This flexibility enables the simulation of intricate behaviors and interactions, resulting in representations that are both more accurate and perceptually meaningful, supporting the relational spatial perspectives in traditional geographic theories.

5.2.5.2 Design Tensions

The model finds a balance in the design tensions introduced by Shannon & Walker (2018). A game engine is inherently flexible as it allows customization through scripting, and integration with a variety of data types. However, there can be challenges with standardization due to the various potential methodology to recreate a feature, each presenting unique trade-offs. Implementing a CFD simulation within a game engine to map flow paths demands expertise in numerical modeling and game engine programming. Experts are essential for designing the simulation's core functionality, but it also offers the tools to create interactions with the resulting model that allow non-experts to customize visualizations and experiment with parameters. Game engine-based tools excel at serving multiple audiences due to their interactive and visual strengths. It allows academic researchers to build and manipulate simulations, policy makers to visualize and assess decisions, and communities to apply personal scenarios. Lastly, results from game engines can be compared with or create established metrics against observed data using built-in analytics, and its interactivity can invite emerging metrics to track meta-data about its users as well as the tool's utility for adoption.

There is another trade-off in the idea of connecting the model to real-time visualization. In cases where computing power is limited, calculations that drive the scientific representation compete over the resources allocated for continuous display and visualization of the process. Visualization itself consumes computation resources, and since it relies on the same underlying calculations as the simulation, it should not take precedence over the model's accuracy and

science for processing power. This tightly coupled approach could potentially become counterproductive if too many resources are diverted towards rendering in real-time.

Similarly, interaction components also compete for these limited resources since additional memory and processing capacity must be allocated to handle potential inputs. In traditional methods, data calculation and visualization are relatively separate processes that depend on optimized, simple representations such as fields. However, one viable approach to overcoming resource competitions is to use interactivity as a tool for optimization. Through turning simulation components on and off at strategic times or advancing simulation time in short, user-controlled steps, the system can better balance the computational load between science, interactivity, and visualization, ensuring that neither aspect overwhelms the other.

In response to Bekkema (2016) identifying the insufficient integration of dynamic models within traditional GIS workflows, game engines present a viable alternative framework. The model developed proposes a direct integration scheme via community-built plugins to achieve dynamic voxel representations of the terrain, allowing discrete models for specific visualization or analytical tasks to be encapsulated and run directly within the engine environment. The plugin also has an active community maintaining its continued updates. In contrast to the often-proprietary nature of existing integrated GIS tools, many robust game engine tools are freely available, and several prominent examples are open-source, potentially mitigating the barriers to access and collaborative development previously hindering geo-computational advancement.

The game engine presents an opportunity to use a creative mindset for thinking outside the box in GIScience, as called for in Elwood (2010), by providing flexible frameworks such as the custom stages in the emitters, where researchers can experiment with unconventional methods of representing data. The integration of particle systems with voxel terrain exemplifies the fusion of data representation paradigms that traditionally remain separate in conventional GIS platforms. This hybrid approach combines the Lagrangian-based nature of particles with the volumetric, Eulerian-based structure of voxels, creating an environment where multiple

conceptual models coexist and interact meaningfully. Combinations of visualization methods, data structures, and interaction paradigms may reveal patterns and relationships otherwise invisible in traditional GIS.

5.3 Limitations

5.3.1 Scale Balance

A key challenge in Euler-Lagrangian models is managing computational resources. Performance hinges on a single factor in purely grid or particle-based methods, but hybrid models as presented in Chapter 4 require careful optimization for both to achieve a practical balance between accuracy and speed. This balance between voxel resolution and particle size directly impacts the model's performance and accuracy. If the voxel resolution is extremely high, the terrain surface becomes highly detailed, with numerous small irregularities and bumps that. In this scenario, smaller particles, which are intended to capture fine-scale flow details, may become trapped within these physically rough surface features, hindering their movement. While this technically represents realistic behavior to a certain extent and can potentially be overcome with a much larger number of particles to sustain surface flow, the computation cost implied is significantly higher. Conversely, if the particles are too large relative to the voxel size, they will effectively ignore the terrain's fine details. The particles will glide over smaller bumps and depressions, failing to accurately represent the surface's influence on flow. This results in a loss of detailed terrain information and renders the high voxel resolution redundant, essentially wasting computational resources that were dedicated to capturing those details. Therefore, finding the optimal ratio between voxel resolution and particle size is critical to ensure that the model accurately represents the terrain's influence on fluid flow while maintaining computational efficiency.

The scale and scope-based challenges in traditional 2D GIS approaches take on a new form in this approach, as voxel resolution and particle scales need to maintain a relative granularity

relationship. In summary, finer resolution terrain also requires a larger number of smaller-sized particles to interact with microtopography, for the generation of an accurate and meaningful flow.

Beyond the technical implementation, conceptually developing a robust and computationally tractable mathematical representation of the soil exchange process (erosion and deposition) was unsuccessful. This required formulating equations that accurately model the gain and loss of sediment mass by a particle based on physical principles (e.g., velocity, slope angle, soil properties, current sediment load). The equation must also be implemented using the limited event triggers and data manipulation capabilities within the Niagara system to represent realistic erosion mechanics, as well as execute efficiently within the real-time constraints.

5.3.2 Conceptual Compatibility

The Smoothed Particle Hydrodynamics (SPH) technique is normally deployed at much higher particle counts and on much smaller scales to capture detailed volumetric flow properties (e.g., Long et al., 2023). In the model built within the game engine, however, a relatively lower number of particles were used and the fluid exhibited less vertical interaction. This marks a difference in application of full-fledged SPH fluid simulations. While this approach may be seen as a missed opportunity to fully exploit the power of SPH in rendering detailed volumetric dynamics, it also represents a significant step forward for exploring effects of subtle fluid dynamics on a larger spatial scale. It demonstrates the conceptual compatibility between game engines, GIS, and CFD in a geospatial context for flow path analysis, an application area that SPH fluid models have not traditionally been used for. This fusion broadens the scope of SPH application to hydrological studies in environment modelling as well and potentially highlights a promising multi-disciplinary method that leverages the interactive and visualization strengths of game engines alongside the detailed process modeling found in CFD and GIS. With the development of hardware capability, the scale at which this workflow can be applied may increase exponentially.

5.3.3 Resource Availability

This specific problem domain of achieving real-time erosion simulation using particle systems within a commercial game engine, proved to be operating at the edge of readily available knowledge and community support. Despite the extensive documentation and active community surrounding UE, resources addressing the topic of combining large-scale Niagara particle systems and dynamic voxel terrain modification were non-existent. Consequently, much of the development process involved extensive trial-and-error experimentation, consuming significant time and effort. While this iterative process yielded insights into the engine's capabilities and limitations, the lack of established methodologies or expert guidance for this particular application made progress exceptionally challenging.

Despite the model's attempt to pursue realism, there is still a lack of consideration for many realistic factors outside of modelling water. Research efforts for watershed models often produce predictions on 2D grids (e.g., Beguería et al., 2009), and when extended to higher-dimensional applications like 3D grids within game engines or geospatial analyses, the computational complexity of numerical data simulation increases exponentially. This cost of computation forces models to focus on one aspect of the simulation, or to balance the available resources to achieve a comprehensive model. However, the predictive value of environmental simulations, particularly for phenomena like water flow, depends critically on capturing the nuances of the real-world landscape. For instance, suppose a region is characterized by a combination of urban areas with impervious surfaces, rivers with varying topography, and vegetated regions. The accurate representation of these features is vital for predicting how water will flow, accumulate, and potentially cause flooding during heavy rainfall or storm events. These factors were not included in the model due to the lack of technical resources of their implementation, which meant the simulation lacks precision and fails to account for realistic external characteristics, such as the impact of any building material, the flow resistance introduced by vegetation, or the distribution of soil types. The lack of realism can cause the model's predictions to deviate significantly from reality.

The use of game engines reduced the programming ability required to build simple simulations, but further progress required deeper technical understanding to implement features and build experiences to engage audiences, in addition to more development time and higher demands for the quality of computation hardware. The level of demand for technical skills as well as hardware capacity to optimally represent these erosion mechanics on the expected scale was significant, and limited the progress and scope of the project. As engines mature, the continued development and refinement of these models, increased access to knowledge surrounding the engine, along with improvements in hardware and data acquisition technologies, are likely factors to overcome the current challenges. This direction of research can potentially allow physics-based models and mapping in game engines to become an increasingly viable and valuable tool in geospatial science.

5.3.4 Simplifications

The Position Based Dynamics (PBD) implementation is fundamentally constrained by its reliance on a simplified collision model that assumes all interacting shapes are spherical. This approach, which uses only a collision radius to determine interactions, fails to account for the complex geometries encountered in real-world fluid dynamics. In realistic scenarios, fluids are not perfect spheres but closer to continuous surfaces in contact with the ground, and as the simulated spheres scale larger, the effective contact area with the terrain diminishes. This results in an idealized and less accurate representation of fluid behavior, limiting the model's capacity to simulate accurate and nuanced interactions between fluid and solid boundaries.

Similarly, the SPH approach assumes that the fluid is Newtonian and incompressible. In reality, fluids can experience a much wider range of compressibility under varying conditions than is typically represented in SPH simulations. Moreover, to maintain simulation stability, SPH implementation imposes thresholds on forces and velocities. These constraints limit the model's ability to simulate extreme fluctuations, shear stress, potentially omitting behaviors that influence the realism of overall fluid motion and interaction.

Furthermore, fluid simulations using SPH only approximate the NSE equations. The NSEs also remain an unresolved challenge in mathematics as one of the Millennium Prize Problems. This implies that the accuracy of any predictions struggles with each incremental frame, because fluid flow is naturally chaotic and cannot (at the time of writing this) be fully captured in a simulation. Validation of whether a dynamic process such as this has been accurately captured is complicated as the benchmark is against the chaotic nature of reality, and the model cannot fully reflect real fluid complexity as of yet.

5.3.5 Optimizations

Despite the optimizations of modern voxel algorithms, there are still an array of challenges this representation faces. Since the primary objective of voxels is to represent volume, surface detail may be lost when constrained to a grid of insufficient resolution. This implies that a volumetric representation of curved surfaces is prone to over or under-representing the actual volume of an object (e.g., Hess et al., 2018). The effectiveness of voxel representation is also limited by the shape of the dataset, such as overly sparse distributions or dense clusters, both of which require high-resolution only at particular regions of the grid, relying on complex optimization techniques for resolution (e.g., Laine & Karras, 2010). In addition, every cell of the grid is assigned memory regardless of it containing useful data, which can cause significant computational delay in large scale scenarios (e.g., Čerešník et al., 2019). Different from meshes, which only store information about the surface and the inside is hollow, voxel grids can hide a large amount of data that is behind visible voxels, which require complex volumetric rendering techniques to appropriately represent (e.g., Fong et al., 2017). As such, voxel data are not typically built by hand or collected via sensors, but transformed from other datasets for further processing purposes, or generated procedurally. It is likely that traditional GIS research has often avoided voxels due to these constraints and the technical difficulty of representing such data in dynamic 3D frameworks.

Meshes have significant limitations in their ability to represent dynamically shape-shifting entities, such as liquids or soft bodies (Prytuła, 2024), due to their reliance on predefined

surfaces. For this reason, they are commonly chosen to represent objects which do not change shape in a game engine.

These trade-offs generally exist when designing any efficient software solutions. In modern computing platforms, hardware parallelism is ubiquitous, but it requires specialized knowledge and a substantial investment in technical skill development to fully harness its potential benefits. Game engines are no exception in this balance, they are designed to take the middle road by offering some limited benefits of software efficiency and advanced optimization for the non-expert, whereas the full capabilities still require a level of expertise.

5.4 Broader Implications

5.4.1 Theoretical

As the rate and quantity of data collection increases, workflows may evolve to ultimately become real-time data streams that need to be processed, analyzed, and visualized in real-time. If so, game engines exist in this workflow as a generalizable yet optimized tool, taking on the role of an assembler that imports various data formats, an automation workbench that processes and merges them, a host to the logic that analyzes these data, as well as a renderer of those results, all within the same environment. With this increase, there is also a need to develop more theories with respect to geographical processes in relation to time as a central aspect. For example, game engines enable investigations in the area of temporal emergence in complex natural processes to understand the formation of patterns in a procedural, iterative feedback system. A river path in a game engine model might evolve through iterative feedback between water particles and erodible terrain, mimicking a physical stream table, a process that traditional hydrological models approximate with simplified equations.

The incorporation of simulations facilitates a controlled space for experimenting with existing theories. The implied understanding of geographical space in this manner suggests a high degree of compatibility with the geographic theories discussed in Goodchild (2007), as most

objects in a game engine can be understood as a geo-atom. Similarly, the emitter and the voxel dataset form geo-fields. Therefore, it is possible to build a large theoretical database of all the geo-objects in the real world, assigning their dynamic properties based on the types defined in Goodchild (2007), from which the game engine automatically draws from to defines their interaction. This potentially creates a powerful digital reconstruction tool that can then interface with the drone collected data via object classification techniques to quickly create functional environment models of a geographical area.

Game engines provide a platform for GIS to take advantage of high-performance computing. It enables direct manipulation of model parameters and provides real-time feedback. This interactive platform could transform how geographers conceptualize and engage with spatial phenomena. Such systems would allow users to probe the relationships between different geographical agents across scales, revealing emergent behaviors that remain hidden in conventional static analyses. The ability to actively explore spatial dynamics through user-driven interaction could encourage their adoption in interdisciplinary fields to bridge the gap between theoretical models and experiential understanding, providing insights that more accurately reflect the complex, interconnected nature of environmental systems and their constituent agents.

5.4.2 Practical

Game engines are an accessible tool to a wide audience of researchers, as demonstrated by the consumer-level hardware specifications required by the model, as well as the simplicity in implementing a simulation using template components. The current transition of home users' access to relatively high-end graphics hardware is making previous demands on powerful, affordable hardware and software less relevant, making new strides in the direction of GIS development for complex and realistic representation an achievable goal. However, at the consumer-level of hardware, complex and large-scale simulations are still challenging. As such, the application of this approach must consider the context and scale of the intended usage.

The adoption and development of physics-based flowpath mapping methods may have significant implications in various domains that are equipped with hardware above the consumer-grade level. In flood risk management, these models may provide more accurate floodplain delineations and predictions of flood extents, enabling better preparedness and response strategies. In water resource management, they support the design of more effective drainage systems, irrigation schemes, and watershed management plans by providing a volumetric view and estimate of water quantity. Furthermore, in environmental conservation, physics-based models may help identify or mitigate soil erosion impacts on agricultural productivity, water quality, and ecosystem health.

Integrating CFD with drone-collected data creates a digital laboratory by using the drone's measurements to define and validate the simulation environment. The CFD model then numerically solves the governing fluid flow equations in the laboratory. Further drone data collected within the area of interest can be compared against the CFD results for validation, allowing researchers to calibrate simulation parameters (e.g., delta time, fluid forces) until the model accurately reproduces the expected or observed real-world fluid behavior, establishing a robust, data-grounded virtual environment for experimentation and analysis.

The physics-based foundations of CFD provide both a robust framework for analyzing fluid dynamics and can serve as a reference point for guiding future innovations in the field of hydrology. Techniques in the field of CFD are based on fundamental physics principles but can also be understood as pattern-oriented modelling, as they offer a rigorous and well-established framework for understanding fluid dynamics. Game engines are tools uniquely positioned to inherit and adapt CFD methodologies. By embedding simplified or hybridized forms of these models into interactive, immersive frameworks, game engines can extend the accessibility of fluid modeling while maintaining fidelity to the core physical principles. This synergy allows hydrological research to remain grounded in scientific accuracy while embracing new forms of dynamic visualization, experimentation, and user interaction.

Game engines allow a high degree of freedom in methodologies to validate simulations by comparing their results with real-world data or traditional models. This could be done by creating precise measurement tools in the engine, matching engine space to real-world scale. Data export capabilities such as the method used in Chapter 4 can be altered and modified for other analysis goals. However, this validation approach presents several implications when applied to game engine environments.

To create a meaningful comparison between results generated from traditional GIS methods and game engines, researchers may need to compress, aggregate and restructure data generated by the engine, converting rich 3D representations into static 2D formats for compatibility with conventional analytical tools. This transformation inevitably results in the loss of dynamic elements (e.g., time, interaction) and dimensional complexity, which are properties that constitute the game engine's primary advantages. The reduction process may obscure emergent patterns, temporal variations, and spatial relationships that exist in the original simulation. For example, it masks the game engine's capacity to capture transient phenomena such as flow turbulence, short-term obstructions, and rates of melting snow.

5.5 Future Directions

5.5.1 Realism

Future directions of research could explore alternative collision models that incorporate more complex and realistic shape representations beyond simple spheres, if hardware allows. Advancements might include the development of hybrid models that blend PBD with other techniques, or the incorporation of adaptive collision detection algorithms that dynamically adjust based on local geometry. These improvements could pave the way for a more accurate depiction of collision between objects in the model, moving beyond the scale-dependent limitations associated with particle size and quantity, and ultimately result in simulations that better capture the intricate dynamics of fluid-solid interactions.

Additionally, future research may experiment with incorporating different fluid models with the aim to develop a more flexible approach to better fit the needs of geospatial models or be able to switch models depending on the context. This could involve the formulation of adaptive thresholds that allow for temporary deviations during extreme events. Examples such as integrating additional physical parameters to capture rapid, high-magnitude fluctuations in force and velocity, potentially make them more robust for applications that demand a faithful representation of fluid behavior under a wide array of environmental conditions.

The current voxel terrain is a simplification of the ground surface, which can be modelled as flat and is appropriate for fine-scale study areas that assume a flat projected coordinate reference system. However this does not limit the spatial scale a hydrology study would be appropriate for. In UE and the Voxel Plugin, there is the alternative to model the Earth using a globe as well (e.g., Voxel Plugin, 2024, “Cubemap planets”), which could enhance and contribute to large-scale studies. This would be potentially at a higher computational cost, require alternative methodologies, or demand the data to be at a much lower resolution.

5.5.2 Data Enrichment

The current dataset was used primarily for its visible data, but drone-derived information could extend beyond imagery to include less visual soil properties such as type, texture, and structure, as well as vegetation indices that reflect greenness, health, productivity, moisture levels, and nutrient availability. Integrating these factors into a broader modeling framework could enhance the identification of areas at risk of soil degradation and inform strategies for soil conservation, enabling a complex network of models in the game engine. Additionally, this data could serve as initial conditions for interactive experimentation within a voxel-grid terrain representation, with each voxel carrying these soil properties, if hardware and software constraints allow for efficient processing and optimization.

Expanding upon data richness, the potential for integrating infiltration mechanics into this framework is considerable. For example, one promising approach involves utilizing the voxel-

grid representation of the terrain where each voxel cell is capable of storing a parameter indicative of its wetness. This methodology would enable the simulation of infiltration processes by allowing water to travel downwards under the influence of gravity and sideways through osmosis. By assigning wetness values to individual voxels, the model could dynamically track the movement and retention of water within the soil matrix. Such an approach would not only enhance the accuracy of flood simulations by accounting for the absorption and redistribution of water in the subsurface but also improve predictions of soil moisture levels. This information can be vital for assessing the risk of soil degradation, planning effective irrigation strategies, and understanding the dynamics of groundwater recharge. Integrating infiltration mechanics into physics-based flowpath models would thereby provide a richer, more nuanced representation of soil-water interactions, ultimately leading to more robust and generalizable models that capture the complexity of natural hydrological processes.

Data export is also an area for further investigation, as the richness of synthetic data generated within a game engine can be difficult to extract. In the demonstrated model, the movement paths of particles were drawn onto a 2D plane, which only contained the number of particles that pass through a spatial location. This method can be extended to capture the other properties of the particles as well, such as the velocity of flow aggregated over time in a cell, or the amount of sediment that has been transported away or to that cell.

5.5.3 Model Enhancement

The use of SPH in modelling run-off processes is not unique in the field of hydrology (e.g., Chang et al., 2016). However, its integration in UE offers a novel approach that enhances both the visualization and interaction with hydrological simulations in real-time. Unlike traditional hydrological modelling software, which often relies on batch processing and post simulation analysis, UE enables real-time manipulation of the terrain as well as particle spawn patterns. The ability to customize where and how the spawning happens is a built-in feature of

the particle system. This leaves several directions for enhancing the model to increase its utility and relevance, as well as integrating it with other models.

There is a need for more models that challenge the static paradigm by introducing dynamic, event-driven simulations that reflect more realistic, agent-like behaviors. The model could potentially interface and integrate with a well-defined set of equations for driving hydrological erosion, taking advantage of the physical forces that exist on each particle. For example, using the velocity of the particle as a variable for determining the strength and rate that erosion should occur, as well as tracking the balance of sediment within each particle to create branching conditions that signal whether erosion or deposition should happen. In addition to this, the balance of sediment might be modelled with consideration to conservation of energy and mass, to ensure that the precise amount of eroded soil is returned to the ground over the lifespan of a particle.

Dynamic models present advantages over traditional empirical models of erosion, which often rely on simple assumptions to make large-scale simulations computationally convenient. One such assumption is the uniformity of erosion processes across a study area, typically conceptualized as occurring simultaneously in a single step. However, erosion does not occur evenly or all at once in nature. Instead, the process may be more similar to the approach in the presented case study, where erosion spots are influenced by the spatial distribution of particle spawn, how they move, and especially the precise order in which they interact with the environment. Future directions of research should explore how particles may create emergent pathways: particles that land earlier modify the terrain in a way that alters the trajectory of subsequent particles, often deepening and reinforcing particular channels. This kind of interaction generates a cascading feedback loop that mimics natural sediment flow more closely than uniform models. In addition, The UE model was capable of representing more complex patterns of flow, such as braided streams and looping flows. These flow path shapes are beyond the scope of D8, which assumes a single, predetermined flow path and cannot account for terrain evolution or multidirectional flows.

The model can be enhanced by incorporating realistic meteorological data, such as wind direction, precipitation patterns, and temperature variations, which better position it to simulate the initial distribution and subsequent evolution of particles in a manner that reflects real-world dynamics. For example, the spawn algorithm can be integrated with weather data generators (e.g., CLIGEN). This combined approach may enhance the fidelity of the simulation by linking atmospheric inputs to hydrodynamic responses and provide a more holistic view of the interactions between phenomenon such as rainfall patterns to flow processes, further contributing to the overall completion of the model.

In the context of time-dependent equations, game engine implementations also potentially enable temporal data to be manipulated bidirectionally. This functionality could be implemented either by maintaining a comprehensive database of object states that can be recorded and replayed in reverse, or by incorporating reverse processes where existing data serves to "fix" the simulation when interpolations or predictions become misaligned after a certain amount of time. This can be achieved through procedural methods of detecting inconsistencies within the model. Rather than treating simulations as a linear, unidirectional processes, this direction of research further enhances dynamic exploration, refinement, and verification, potentially opening new avenues for understanding complex spatiotemporal phenomena through iterative engagement with the underlying data and processes.

The continuous improvement of modelling dynamic processes, especially in software optimization of calculations on a frame-by-frame basis, can lead to breakthroughs in modeling erosion dynamics, which further influence land evolution models. The developed model contributes the need for more studies on the continuous mechanisms of sediment transfer by providing a base layer to include complex erosion mechanics, to eventually achieving a more generalizable and efficient model. The model building on realistic flow may use the particle as a base to incorporate a deep implementation of the various soil water interactions, considering physics of the detached and transported sediments as well.

5.5.4 Interdisciplinary Models

A promising future direction for model development involves integrating multiple agents into a unified environment within a game engine or linking related models with clearly defined interfaces. This multi-agent simulation could represent a sustainable agricultural landscape, such as a regenerative farm combining agroforestry, crop production, livestock grazing, and more. Ecological agents might simulate vegetation, the growth of certain crops, and wildlife or farm animals, capturing a system that influences local biodiversity and nutrient cycling. Crop yield agents would model the growth of diverse crops, responding to soil health and weather. Carbon storage estimation agents might track sequestration in biomass and soil alongside emissions from farming practices.

This multi-modal system could capture the energy cycle, with plants converting solar energy into biomass via photosynthesis, soil microbes recycling nutrients to support growth, and trophic interactions (e.g., pollination enhancing yields) driving energy flow. This could be useful for experimenting with management strategies that balance productivity with sequestration. Insights from this approach could inform climate-smart agriculture by identifying sustainable practices, experiment with different crop rotations, polyculture, or no-till farming, which enhance yields, biodiversity, and carbon storage, leveraging the game engine's interactivity for scenario testing and education.

The emergence of game engines and similar tools not only democratizes access to advanced modelling and visualization, but also fosters a collaborative environment, facilitating knowledge exchange and the collective development of innovative solutions in the academic community. The technical convergence between game engines and geospatial applications necessarily implies an overlap in the skills required in both domains. Game developers possess expertise in real-time rendering, interactive design, user experience, and performance optimization, which coincidentally are skills increasingly valuable in geospatial visualization. Conversely, geospatial scientists bring deep knowledge of geodata structures, spatial analysis, and domain-specific modeling that can enhance development, particularly for simulations with

real-world applications. Gahegan (2018) and much literature have already mentioned the growth of the GIS community requires a platform or a dynamic tool that changes in response to the evolving needs of scientific research. A platform built on game engines acts as the leverage of GIScience to conduct multi-disciplinary work, which cannot exist without opportunities for professionals to traverse between these domains or to develop hybrid expertise.

5.5.5 Plugins

The development of community-driven libraries and tools specifically designed for GIScience applications within game engines would accelerate adoption and innovation in this field (e.g., Cesium for Unreal, ArcGIS Maps Software Development Kit (SDK) for Unity), as well as those not originally intended for scientific inquiry (e.g., Voxel Plugin). Collaborative initiatives between the gaming industry, academia, and geospatial professionals could foster the exchange of knowledge and best practices. Additionally, the integration of machine learning plugins with game engine-based simulations is a probable direction of development that could enhance the workflow and predictive capabilities of environmental models.

5.6 Conclusions

Game engines possess significant, yet largely untapped, potential for advancing environment science research through visualization, simulation, and analysis. Environment scientists are often unaware of these capabilities, and current usage is limited, failing to leverage the full potential, especially compared to the limitations of traditional 2D GIS tools. A conceptual framework was suggested to advocate the adoption of game engine technology within the domain of GIScience. The framework aimed to facilitate a clearer understanding of how game engines can address specific problems within the field, guide the design of subsequent research or practical implementations, and establish a common ground for discussion and development in this emerging area of study. Subsequently, a case study was conducted to explore the use of particle systems in UE fluid simulation capabilities to achieve

flowpath mapping, a process traditionally challenged by limitations in accounting for physical properties of moving water, such as acceleration and fluid dynamics. This demonstrated the application of the developed framework, presenting an extensible workflow for further research and delivers a comparison with traditional workflows.

While the modelling of known events could be animated through many assumptions about the movement of that object, it is significantly harder to predict phenomenon that are driven by complex laws of nature. To make a scientific prediction, the models we create act as a capsule of our knowledge about natural laws to convey potential future states. The sensors we have do not understand the underlying physics of the features it captures, and it is ultimately up to model builders to apply our understanding of environment science to better recreate the subject of our study. Game engines can provide the building blocks for a platform where simulating realistic environments extends beyond entertainment purposes, reaching to become an active toolset of scientific inquiry offering analytical and theoretical merit.

The approach proposed in using a game engine serves as a viable path to build better models for geospatial science, it enables us to introduce realistic forces and fill the model with physical interactions to predict natural processes. Incorporating hydrodynamics enhances realism in mapping flow paths, providing more realistic representations on sediment transport pathways and potential erosion hotspots. This groundwork can serve as a reference for the current capabilities of a game engine when used by a non-expert. The insights gained may be valuable for further developing and refining erosion models in game engines, and for furthering the communication and understanding of complex natural systems.

References

- Ackerman, Aidan C.S., Cave, Jonathan. (2018). Modeling Coastal Sedimentation and Erosion for Design Applications within the Fields of Landscape Architecture and Architecture. In: Gobster, Paul H.; Swardon, Richard C., eds. Visual resource stewardship conference proceedings: landscape and seascape management in a time of change. Gen. Tech. Rep. NRS-P-183. Newtown Square, PA: U.S. Department of Agriculture, Forest Service, Northern Research Station: 262-269.
- Abdel-Aty, M., Lee, J., Siddiqui, C., & Choi, K. (2013). Geographical unit based analysis in the context of transportation safety planning. *Transportation Research Part A: Policy and Practice*, 49, 62–75. <https://doi.org/10.1016/j.tra.2013.01.030>
- Agarwal, D., Kucukpinar, T., Fraser, J., Kerley, J., Buck, A. R., Anderson, D. T., & Palaniappan, K. (2023). Simulating City-Scale Aerial Data Collection Using Unreal Engine. 2023 IEEE Applied Imagery Pattern Recognition Workshop (AIPR), 1–9. 2023 IEEE Applied Imagery Pattern Recognition Workshop (AIPR). <https://doi.org/10.1109/AIPR60534.2023.10440697>
- Ahmad, S., & Simonovic, S. P. (2004). Spatial system dynamics: New approach for simulation of water resources systems. *Journal of Computing in Civil Engineering*, 18(4), 331-340. [https://doi.org/10.1061/\(ASCE\)0887-3801\(2004\)18:4\(331\)](https://doi.org/10.1061/(ASCE)0887-3801(2004)18:4(331))
- Alberta Environment and Parks. (2020). Alberta Merged Wetland Inventory. Government of Alberta. Retrieved from <https://geodiscover.alberta.ca>.
- Aleksandrov, M., Zlatanova, S., & Heslop, D. J. (2021). Voxelisation Algorithms and Data Structures: A Review. *Sensors*, 21(24), 8241. <https://doi.org/10.3390/s21248241>
- Aleksandrov, M., Zlatanova, S., & Heslop, D. J. (2022). Voxelisation and voxel management options in Unity3D. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, X-4/W2-2022, 13–20. <https://doi.org/10.5194/isprs-annals-X-4-W2-2022-13-2022>

- Anand, N., Meijer, D., van Duin, J.H.R., Tavasszy, L., & Meijer, S. (2016). Validation of an agent-based model using a participatory simulation gaming approach: The case of city logistics. *Transportation Research Part C: Emerging Technologies*, 71, 489-499. <https://doi.org/10.1016/j.trc.2016.08.002>
- Andualem, T.G., Hewa, G.A., Myers, B.R., Peters, S., & Boland, J. (2023). Erosion and sediment transport modeling: A systematic review. *Land*, 12(7), 1396. doi:10.3390/land12071396.
- Anifowose, H., Yan, W., & Dixit, M. (2022). BIM LOD + Virtual Reality—Using Game Engine for Visualization in Architectural & Construction Education (No. arXiv:2201.09954). arXiv. <https://doi.org/10.48550/arXiv.2201.09954>
- An, L. (2012). Modeling human decisions in coupled human and natural systems: Review of agent-based models. *Ecological Modelling*, 229, 25-36. <http://dx.doi.org/10.1016/j.ecolmodel.2011.07.010>
- An, L., Grimm, V., Sullivan, A., Turner, B.L., Malleson, N., Heppenstall, A., Vincenot, C., Robinson, D., Ye, X., Liu, J., Lindkvist, E., & Tang, W. (2021). Challenges, tasks, and opportunities in modeling agent-based complex systems. 457, 109685. <https://doi.org/10.1016/j.ecolmodel.2021.109685>
- Amicarelli, A., Manenti, S., Albano, R., Agate, G., Paggi, M., Longoni, L., Mirauda, D., Ziane, L., Viccione, G., Todeschini, S., Sole, A., Baldini, L.M., Brambilla, D., Papini, M., Khellaf, M.C., Tagliafierro, B., Sarno, L. & Pirovano, G. (2020). SPHERA v.9.0.0: A Computational Fluid Dynamics research code, based on the Smoothed Particle Hydrodynamics mesh-less method. *Computer Physics Communications*, 250, 107157. <https://doi.org/10.1016/j.cpc.2020.107157>
- Ascough II, J. C., et al. (1997). The WEPP watershed model: I. Hydrology and erosion. *Transactions of the ASAE* 40.4: 921-933.

- Automation with ArcGIS Pro. (n.d.). ArcGIS Architecture Center. Retrieved from <https://architecture.arcgis.com/en/framework/architecture-practices/automation/automation-with-arcgis-pro.html>
- Balázs, T., Balázs, J. (2011). Fast Hydraulic and Thermal Erosion on the GPU. Retrieved from <http://dx.doi.org/10.2312/EG2011/short/057-060>.
- Baiamonte, G. & Agnese, C. (2010). An analytical solution of kinematic wave equations for overland flow under Green-Ampt infiltration. *Journal of Agricultural Engineering*, 41(1), pp. 41–48. doi:10.4081/jae.2010.1.41.
- Banerjee, S., Carlin, B. P., Gelfand, A. E., & Banerjee, S. (2003). Hierarchical Modeling and Analysis for Spatial Data. Chapman and Hall/CRC. <https://doi.org/10.1201/9780203487808>
- Barczak, A., & Woźniak, H. (2020). Comparative Study on Game Engines. *Studia Informatica*, 2020. <https://doi.org/10.34739/si.2019.23.01>
- Beguiría, S., Van Asch, T. W. J., Malet, J.-P., & Gröndahl, S. (2009). A GIS-based numerical model for simulating the kinematics of mud and debris flows over complex terrain. *Natural Hazards and Earth System Sciences*, 9(6), 1897-1909. <https://doi.org/10.5194/nhess-9-1897-2009>
- Bekkema, M. (2016). The implementation of GIS for integrated environmental modelling: times are changing.
- Bell, A. R., Robinson, D. T., Malik, A., & Dewal, S. (2015). Modular ABM development for improved dissemination and training. *Environmental Modelling & Software*, 73, 189-200. <https://doi.org/10.1016/j.envsoft.2015.07.016>

- Bell, A. (2017). Informing decisions in agent-based models — A mobile update. *Environmental Modelling & Software*, 93, 310-321. <http://dx.doi.org/10.1016/j.envsoft.2017.03.028>
- Benes, B., & Forsbach, R. (2002). Visual Simulation of Hydraulic Erosion. WSCG. 10.79-94.
- Benes, B. (2007). Real-Time Erosion Using Shallow Water Simulation. In: Dingliana, J., Ganovelli, F., eds. *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS"* (2007). The Eurographics Association: 43-50.
- Bergmann, L., & Lally, N. (2021). For geographical imagination systems. *Annals of the American Association of Geographers*, 111(1), 26-35.
- Beven, K. (1989). Changing ideas in hydrology - The case of physically-based models. *Journal of Hydrology*, 105(1-2), 157-172. [https://doi.org/10.1016/0022-1694\(89\)90101-7](https://doi.org/10.1016/0022-1694(89)90101-7)
- Beyer, H. T. (2015). Implementation of a method for hydraulic erosion. Retrieved from <https://www.firespark.de/resources/downloads/implementation%20of%20a%20methode%20for%20hydraulic%20erosion.pdf>
- Blaschke, & Merschdorf, H. (2014). Geographic information science as a multidisciplinary and multiparadigmatic field. *Cartography and Geographic Information Science*, 41(3), 196–213. <https://doi.org/10.1080/15230406.2014.905755>
- Blaskovic, P. (n.d.). Fire Fluid Simulation 3. Retrieved from https://www.escapemotions.com/experiments/fluid_fire_3/index.php
- Blender. (2025). Point Cloud. Blender 4.3 Manual. Retrieved from https://docs.blender.org/manual/en/latest/modeling/point_cloud.html
- Boeing, A., Bräunl, T. (2007). Evaluation of real-time physics simulation systems. *Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia*, 281–288. <https://doi.org/10.1145/1321261.1321312>

- Boeing, G. (2020). The right tools for the job: The case for spatial science tool-building. *Transactions in GIS*, 24(5), 1299–1314. <https://doi.org/10.1111/tgis.12678>
- Boonya-aroonnet, S., Maksimovic, C., Prodanović, D., & Djordjević, S. (2007). Urban pluvial flooding: Development of GIS based pathway model for surface flooding and interface with surcharged sewer model. <https://www.semanticscholar.org/paper/Urban-pluvial-flooding%3A-Development-of-GIS-based-Boonya-aroonnet-Maksimovic/33bcff359678a2ee4dcb1900557510daf223d11>
- Bowlick, F., Thompson, C., Thompson, C., Cox, C., Clark, B., Drummy, N., Fox, J., Grady, A., Musshorn, L., Dony, C. (2022). Understanding geocomputation education: A survey and syllabi informed review. *Research in Geographic Education*, 23, 20-51.
- Bradfield, C. (2018). *Godot Engine Game Development projects: Build five cross-platform 2D and 3D games with Godot 3.0*. Packt Publishing Ltd, 2018.
- Brazier, R. E., Beven, K. J., Freer, J., & Rowan, J. S. (2000). Equifinality and uncertainty in physically based soil erosion models: Application of the GLUE methodology to WEPP-the Water Erosion Prediction Project-for sites in the UK and USA. *Earth Surface Processes and Landforms*, 25(8), 825-845. doi:10.1002/1096-9837(200008)25:83.0.co;2-3
- Bregt, A.K., Skidmore, A.K., & Nieuwenhuis, G. (2002). Environmental modelling: issues and discussion. In H. Skidmore & A.K. Skidmore (Eds.), *Environmental modelling with GIS and remote sensing* (pp. 237-254). Taylor & Francis.
- Broderick, J., Duggan, J., & Redfern, S. (2016). Using game engines for marine visualisation and collaboration. 2016 International Conference on Image, Vision and Computing (ICIVC), 96–101. 2016 International Conference on Image, Vision and Computing (ICIVC). <https://doi.org/10.1109/ICIVC.2016.7571280>

- Brooks, S., & Whalley, J. L. (2008). Multilayer hybrid visualizations to support 3D GIS. *Computers, Environment and Urban Systems*, 32(4), 278-292. doi:10.1016/j.compenvurbsys.2007.11.001
- Bryceson, K. P., Leigh, S., Sarwar, S., & Grøndahl, L. (2022). Affluent Effluent: Visualizing the invisible during the development of an algal bloom using systems dynamics modelling and augmented reality technology. *Environmental Modelling and Software*, 147, Article 105253. <https://doi.org/10.1016/j.envsoft.2021.105253>
- Buckland, S. T., Oedekoven, C. S., & Borchers, D. L. (2016). Model-based distance sampling. *Journal of Agricultural, Biological, and Environmental Statistics*, 21, 58–75.
- Burch et al. (2022). Investigation of 3D Game Engines to Support Modeling Efforts. https://cfpub.epa.gov/si/si_public_record_report.cfm?dirEntryId=356430&Lab=CESER
- C., A. (2022). Niagara Collision. Epic Games. Retrieved from <https://dev.epicgames.com/community/learning/knowledge-base/oDPB/unreal-engine-niagara-collisions>
- Carbonell-Carrera, & Hess-Medler, S. (2019). Interactive Visualization Software to Improve Relief Interpretation Skills: Spatial Data Infrastructure Geoportal versus Augmented Reality. *The Professional Geographer*, 71(4), 725–737. <https://doi.org/10.1080/00330124.2019.1595064>
- Carbonell-Carrera, C., Saorin, J. L., & Melián Díaz, D. (2021). User VR Experience and Motivation Study in an Immersive 3D Geovisualization Environment Using a Game Engine for Landscape Design Teaching. *Land*, 10(5), 492. <https://doi.org/10.3390/land10050492>
- Carretier, S., Regard, V., Abdelhafiz, Y., & Plazolles, B. (2023). Modelling detrital cosmogenic nuclide concentrations during landscape evolution in Cidre v2. 0. *Geoscientific Model Development*, 16(22), 6741-6755.

- Čerešník, P., Madoš, B., & Bilanová, Z. (2019). SSVDAG*: Efficient Volume Data Representation Using Enhanced Symmetry-Aware Sparse Voxel Directed Acyclic Graph. 2019 IEEE 15th International Scientific Conference on Informatics. <https://doi.org/10.1109/Informatics47936.2019.9119298>
- Chang, T.-J., Chang, Y.-S. & Chang, K.-H. (2016). Modeling rainfall–runoff processes using smoothed particle hydrodynamics with mass-varied particles. *Journal of Hydrology*, 543, 749–758. <https://doi.org/10.1016/j.jhydrol.2016.10.045>
- Charney, N. D., Record, S., Gerstner, B. E., Merow, C., Zarnetske, P. L., & Enquist, B. J. (2021). A Test of Species Distribution Model Transferability Across Environmental and Geographic Space for 108 Western North American Tree Species. *Frontiers in Ecology and Evolution*, 9, Article 689295. <https://doi.org/10.3389/fevo.2021.689295>
- Cheng, T., & Adepeju, M. (2014). Modifiable temporal unit problem (MTUP) and its effect on space-time cluster detection. *PloS one*, 9(6), e100465. Modifiable Temporal Unit Problem (MTUP) and Its Effect on Space-Time Cluster Detection | PLOS ONE
- Chen, J. X., & Lobo, N. D. V. (1995). Toward Interactive-Rate Simulation of Fluids with Moving Obstacles Using Navier-Stokes Equations. *Graphical Models and Image Processing*, 57(2), 107–116. <https://doi.org/10.1006/gmip.1995.1012>
- Chentanez, Nuttapong & Müller, Matthias. (2010). Real-time Simulation of Large Bodies of Water with Small Scale Details.. 197-206. 10.2312/SCA/SCA10/197-206.
- Chiba, N., Muraoka, K., Fujita, K. (1999). An erosion model based on velocity fields for the visual simulation of mountain scenery. *The Journal of Visualization and computer Animation*, vol. 9, 185-194.
- Christopoulou, E., & Xinogalos, S. (2017). Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. *International Journal of Serious Games*, 4(4). <https://doi.org/10.17083/IJSG.V4I4.194>

- Cinelli, M., Kadziński, M., Gonzalez, M., & Słowiński, R. (2020). How to support the application of multiple criteria decision analysis? Let us start with a comprehensive taxonomy. *Omega*, 96, 102261. <https://doi.org/10.1016/j.omega.2020.102261>
- Clark, J. S. (2005). Why environmental scientists are becoming Bayesians. *Ecology letters*, 8(1), 2-14.
- Clark, M. P., Bierkens, M. F. P., Samaniego, L., Woods, R. A., Uijlenhoet, R., Bennett, K. E., Pauwels, V. R. N., Cai, X., Wood, A. W., & Peters-Lidard, C. D. (2017). The evolution of process-based hydrologic models: Historical challenges and the collective quest for physical realism. *Hydrology and Earth System Sciences*, 21(7), 3427–3440. <https://doi.org/10.5194/hess-21-3427-2017>
- CloudCompare. (n.d.). Retrieved from <https://www.cloudcompare.org/>
- Çöltekin, A., Bleisch, S., Andrienko, G., & Dykes, J. (2017). Persistent challenges in geovisualization – a community perspective. *International Journal of Cartography*, 3(sup1), 115–139. <https://doi.org/10.1080/23729333.2017.1302910>
- Cöltekin, A., De Sabbata, S., Willi, C., Vontobel, I., Pfister, S., Kuhn, M., & Lacayo, M. (2011). Modifiable temporal unit problem. In *Persistent problems in geographic visualization; ICC2011 Workshop*, Paris, France, July 2, 2011. University of Zurich. <https://doi.org/10.5167/uzh-54263>
- Collins, A., Koehler, M., & Lynch, C. (2024). Methods that support the validation of agent-based models: An overview and discussion. *Journal of Artificial Societies and Social Simulation*, 27(1), 11. <https://doi.org/10.18564/jasss.5258>
- Connor Hill. (2020, July 21). INSANE! This is the MOST ADVANCED PHYSICS ENGINE I've EVER SEEN! I can't believe this even exists.... Retrieved from <https://youtu.be/DBcqiQLp7lY>

- Cosgrove, D. (2012). *Geography and vision: Seeing, imagining and representing the world*. Bloomsbury Publishing.
- Couclelis, H. (1992). People manipulate objects (but cultivate fields): beyond the raster-vector debate in GIS. In *Theories and methods of spatio-temporal reasoning in geographic space* (pp. 65-77). Springer, Berlin, Heidelberg.
- Coulthard, T. J., Neal, J. C., Bates, P. D., Ramirez, J., de Almeida, G. A. M., & Hancock, G. R. (2013). Integrating the LISFLOOD-FP 2D hydrodynamic model with the CAESAR model: Implications for modelling landscape evolution. *Earth Surface Processes and Landforms*, 38(15), 1897–1906. <https://doi.org/10.1002/esp.3478>
- Coulthard, T. J., & Skinner, C. J. (2016). The sensitivity of landscape evolution models to spatial and temporal rainfall resolution. *Earth Surface Dynamics*, 4(3), 757–771. <https://doi.org/10.5194/esurf-4-757-2016>
- Cox, D. R., & Isham, V. (1988). A simple spatial-temporal model of rainfall. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 415(1849), 251-260. <https://doi.org/10.1098/rspa.1988.0016>
- Craighead, J., Murphy, R., Burke, J., & Goldiez, B. (2007). A Survey of Commercial & Open Source Unmanned Vehicle Simulators. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 852–857. 2007 IEEE International Conference on Robotics and Automation. <https://doi.org/10.1109/ROBOT.2007.363092>
- Cristie, V., & Berger, M. (2017). Game Engines for Urban Exploration: Bridging Science Narrative for Broader Participants. In A. Nijholt (Ed.), *Playable Cities: The City as a Digital Playground* (pp. 87–107). Springer. https://doi.org/10.1007/978-981-10-1962-3_5
- Crystal-Ornelas, R., Varadharajan, C., Bond-Lamberty, B., Boye, K., Burrus, M., Cholia, S., Crow, M., Damerow, J., Devarakonda, R., Ely, K. S., Goldman, A., Heinz, S., Hendrix, V., Kakalia, Z., Pennington, S. C., Robles, E., Rogers, A., Simmonds, M., Velliquette, T., ...

- Agarwal, D. A. (2021). A Guide to Using GitHub for Developing and Versioning Data Standards and Reporting Formats. *Earth and Space Science*, 8(8), e2021EA001797. <https://doi.org/10.1029/2021EA001797>
- Csáfordi, P., Pödör, A., Bug, J., Gribovsky, Z. (2012). Soil Erosion Analysis in a Small Forested Catchment Supported by ArcGIS Model Builder. *Acta Silvatica Et Lignaria Hungarica*, 8(1), 39-56. doi:10.2478/v10303-012-0004-5
- Dai, A., & Nießner, M. (2018). Scan2Mesh: From Unstructured Range Scans to 3D Meshes. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://doi.org/10.1109/CVPR.2019.00572>
- Daneshjoo, N. (2012). Computers Modeling and Simulation. *Advanced Materials Research*, 463–464, 1102–1105. <https://doi.org/10.4028/www.scientific.net/AMR.463-464.1102>
- Darvishi, M., & Ahmadi, G. (2014). Validation techniques of agent based modelling for geospatial simulations. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-2-W3, 91–95. WG II/1, WG II/4, ICWG II/IV, WG IV/7 </br> The 1st ISPRS International Conference on Geospatial Information Research (Volume XL-2/W3) - 15–17 November 2014, Tehran, Iran. <https://doi.org/10.5194/isprsarchives-XL-2-W3-91-2014>
- Davies, C. (2020). Place-based knowledge systems: human and machine. Zenodo. <https://doi.org/10.5281/zenodo.3628865>
- Dazzi, S., Vacondio, R. and Mignosa, P. (2019). Integration of a levee breach erosion model in a GPU-accelerated 2D shallow water equations code. *Water Resources Research*, 55(1), pp. 682–702. <https://doi.org/10.1029/2018WR023826>
- Deckard, M., & Schnell, C. (2022). The Temporal (In)Stability of Violent Crime Hot Spots Between Months and The Modifiable Temporal Unit Problem. *Crime & Delinquency*, 69(6-7), 1312-1335. <https://doi-org.proxy.lib.uwaterloo.ca/10.1177/00111287221128483>

- Degbelo, A. (2022). FAIR geovisualizations: Definitions, challenges, and the road ahead. *International Journal of Geographical Information Science*, 36(6), 1059–1099. <https://doi.org/10.1080/13658816.2021.1983579>
- de Jong, R., & de Bruin, S. (2012). Linear trends in seasonal vegetation time series and the modifiable temporal unit problem. *Biogeosciences*, 9(1), 71. Retrieved from <https://proxy.lib.uwaterloo.ca/login?url=https://www.proquest.com/scholarly-journals/linear-trends-seasonal-vegetation-time-series/docview/919685489/se-2>
- de Regt, H. W. (2017). A Contextual Theory of Scientific Understanding. *Understanding Scientific Understanding* (New York, 2017; online edn, Oxford Academic, 24 Aug. 2017), <https://doi-org.proxy.lib.uwaterloo.ca/10.1093/oso/9780190652913.003.0004>
- De Roo, A. P. J., Wesseling, C. G., & Van Deursen, W. P. A. (2000). Physically based river basin modelling within a GIS: The LISFLOOD model. *Hydrological Processes*, 14(11–12), 1981–1992. [https://doi.org/10.1002/1099-1085\(20000815/30\)14:11/12<1981::AID-HYP49>3.0.CO;2-F](https://doi.org/10.1002/1099-1085(20000815/30)14:11/12<1981::AID-HYP49>3.0.CO;2-F)
- Devia, G. K., Ganasri, B. P., & Dwarakish, G. S. (2015). A Review on Hydrological Models. *Aquatic Procedia*, 4, 1001–1007. <https://doi.org/10.1016/j.aqpro.2015.02.126>
- de Vos, M. G., Janssen, S. J. C., van Bussel, L. G. J., Kromdijk, J., van Vliet, J., & Top, J. L. (2011). Are environmental models transparent and reproducible enough? In 19th International Congress on Modelling and Simulation (pp. 2954–2961). Modelling and Simulation Society of Australia and New Zealand.
- Dimov, I., Georgiev, K., Waśniewski, J., & Zlatev, Z. (1996). Three-dimensional version of the Danish Eulerian Model. In J. Dongarra, K. Madsen, & J. Waśniewski (Eds.), *Applied Parallel Computing Computations in Physics, Chemistry and Engineering Science* (pp. 151–157). Springer. https://doi.org/10.1007/3-540-60902-4_18

- Dong, L., You-Quan, L., Kai, B., & En-Hua, W. (2010). Real-time shallow water simulation on terrain. *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry*, 331–338. <https://doi.org/10.1145/1900179.1900249>
- dos Santos, F. M., Pelinson, N. de S., de Oliveira, R. P., & Di Lollo, J. A. (2023). Using the SWAT model to identify erosion prone areas and to estimate soil loss and sediment transport in Mogi Guaçu River basin in Sao Paulo State, Brazil. *CATENA*, 222, 106872. <https://doi.org/10.1016/j.catena.2022.106872>
- Droll, D., & Söbke, H. (2021). Realism of Simulation Models in Serious Gaming: Two case studies from Urban Water Management Higher Education (Vol. 13134, pp. 277–282). https://doi.org/10.1007/978-3-030-92182-8_29
- Ellul, C., & Haklay, M. (2006). Requirements for Topology in 3D GIS. *Transactions in GIS*, 10(2), 157-175.
- Elsawah, S., Filatova, T., Jakeman, A.J., Kettner, A.J., Zellner, M.L., Athanasiadis, I.N., Hamilton, S.H., Axtell, R.L., Brown, D.G., Gilligan, J.M., Janssen, M.A., Robinson, D.T., Rozenberg, J., Ullah, I.I.T., & Lade, S.J. (2020). Eight grand challenges in socio-environmental systems modeling. *Socio-Environmental Systems Modelling*, vol. 2, 16226, 2020, doi:10.18174/sesmo.2020a16226
- Elwood. (2010). Thinking Outside the Box: Engaging Critical Geographic Information Systems Theory, Practice and Politics in Human Geography: Thinking outside the box. *Geography Compass*, 4(1), 45–60. <https://doi.org/10.1111/j.1749-8198.2009.00289.x>
- Entrikin, J. (1977). GEOGRAPHY'S SPATIAL PERSPECTIVE AND THE PHILOSOPHY OF ERNST CASSIRER. *Canadian Geographer*. <https://doi.org/10.1111/J.1541-0064.1977.TB01594.X>

- Epic Games. (n.d.). Hardware and Software Specifications for Unreal Engine | Unreal Engine 5.6 Documentation | Epic Developer Community. Epic Games Developer. Retrieved June 5, 2025, from <https://dev.epicgames.com/documentation/en-us/unreal-engine/hardware-and-software-specifications-for-unreal-engine>
- Erwig, M., Güting, R. H., Schneider, M., & Vazirgiannis, M. (1999). Spatio-Temporal Data Types: An Approach to Modeling and Querying Moving Objects in Databases. *GeoInformatica*, 3(3), 269–296. <https://doi.org/10.1023/A:1009805532638>
- Esri. (n.d.-a). ArcGIS Pro Pricing | Buy Now. Retrieved June 5, 2025, from <https://www.esri.ca/en-ca/store/products/buy/arcgis-pro>
- Esri. (n.d.-b). Introduction to ArcGIS. ArcGIS Architecture Center. Retrieved June 5, 2025, from <https://architecture.arcgis.com/en/overview/introduction-to-arcgis/introduction.html>
- Evans, D., Johnson, S., & Chamberlain, B. (2023). Transferring Spatial Data from Unity to ArcGIS. *Journal of Digital Landscape Architecture*, 8, 515–522. <https://doi.org/10.14627/537740054>
- Evans, T. P., Sun, W., & Kelley, H. (2006). Spatially explicit experiments for the exploration of land-use decision-making dynamics. *International Journal of Geographical Information Science*, 20(9), 1013–1037. <https://doi.org/10.1080/13658810600830764>
- Evergine. (2021, November 16). The multiplatform graphics engine designed for industry. Retrieved from <https://evergine.com/>
- Fatichi, S., Vivoni, E. R., Ogden, F. L., Ivanov, V. Y., Mirus, B., Gochis, D., Downer, C. W., Camporese, M., Davison, J. H., Ebel, B., Jones, N., Kim, J., Mascaro, G., Niswonger, R., Restrepo, P., Rigon, R., Shen, C., Sulis, M., & Tarboton, D. (2016). An overview of current applications, challenges, and future trends in distributed process-based models in hydrology. *Journal of Hydrology*, 537, 45-60. <https://doi.org/10.1016/j.jhydrol.2016.03.026>

- Fefferman, C. L., Robinson, J. C., & Rodrigo, J. L. (Eds.). (2018). *Partial Differential Equations in Fluid Mechanics* (1st ed.). Cambridge University Press. <https://doi.org/10.1017/9781108610575>
- Fitzpatrick Mia , Hedley Nick. (2024). Review of the state of practice in geovisualization in the geosciences. *Frontiers in Earth Science*, Volume 11 – 2023. DOI10.3389/feart.2023.1230973. ISSN2296-6463. <https://www.frontiersin.org/journals/earth-science/articles/10.3389/feart.2023.1230973>
- Flanagan, D. C., Frankenberger, J. R., Ascough II, J. C. (2012). WEPP: Model use, calibration, and validation. *Transactions of the ASABE* 55.4: 1463-1477.
- FLIPSIDE. (2021, July 12). Why Unreal Engine is not just a game engine. Retrieved from <https://youtu.be/dLGOtGEb00U>
- Fong, J., Wrenninge, M., Kulla, C., & Habel, R. (2017). Production volume rendering. *ACM SIGGRAPH 2017 Courses*, Article No. 2, 1-79. <https://doi.org/10.1145/3084873.3084907>
- Frank, & Mark, D. M. (1991). *Cognitive and linguistic aspects of geographic space* (Frank & D. M. Mark, Eds.; 1st ed. 1991.). Kluwer Academic Publishers. <https://doi.org/10.1007/978-94-011-2606-9>
- Frawley, P., O'Mahony, A. P., & Geron, M. (2010). Comparison of Lagrangian and Eulerian Simulations of Slurry Flows in a Sudden Expansion. *Journal of Fluids Engineering*, 132(9), Article 091301. <https://doi.org/10.1115/1.4002357>
- Friese, KI., Herrlich, M., Wolter, FE. (2008). Using Game Engines for Visualization in Scientific Applications. In: Ciancarini, P., Nakatsu, R., Rauterberg, M., Roccetti, M. (eds) *New Frontiers for Entertainment Computing. ECS 2008*. IFIP International Federation for Information Processing, vol 279. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09701-5_2

- Fu, B., Merritt, W. S., Croke, B. F. W., Weber, T. R., & Jakeman, A. J. (2019). A review of catchment-scale water quality and erosion models and a synthesis of future prospects. *Environmental Modelling & Software*, 114, 75-97. <https://doi.org/10.1016/j.envsoft.2018.12.008>
- Gahegan, M. (2018). Our GIS is too small. *The Canadian Geographer/Le Géographe Canadien*, 62(1), 15-26. <https://doi.org/10.1111/cag.12434>
- Gao, F., & Zhang, X. (2021). Mapping crop phenology in near real-time using satellite remote sensing: Challenges and opportunities. *Journal of Remote Sensing*, 2021(1), Article 8379391. <https://doi.org/10.34133/2021/8379391>
- Garry's Blog. (2020, May 22). What Unity Is Getting Wrong. Retrieved from <https://garry.net/posts/unity-2020>
- Gartia Jenkins. (2016, December 2). UE4 Dynamic Water Nvidia GameWorks. Retrieved from <https://youtu.be/SMQ7-YuO9Xo>
- Gedlaman, H. E. (2024). Representing High-Fidelity Geospatial Data in Unreal Engine: Establishing a Foundation for Numerical Computation and Simulation in a Virtual Reality Environment. Master's Thesis. University of Montana. <https://scholarworks.umt.edu/etd/12348>
- Gerst, M. D., Kenney, M. A., Baer, A. E., Speciale, A., Wolfinger, J. F., Gottschalck, J., Handel, S., Rosencrans, M., & Dewitt, D. (2020). Using Visualization Science to Improve Expert and Public Understanding of Probabilistic Temperature and Precipitation Outlooks. *Weather, Climate, and Society*, 12(1), 117-133. <https://doi.org/10.1175/WCAS-D-18-0094.1>
- Gold, Chau, M., Dziesko, M., & Goralski, R. (2005). 3D Geographic Visualization: The Marine GIS. In *Developments in Spatial Data Handling* (pp. 17–28). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-26772-7_2

- González, J. D., Escobar, J. H., Sánchez, H., Hoz, J. D. la, & Beltrán, J. R. (2017). 2D and 3D virtual interactive laboratories of physics on Unity platform. *Journal of Physics: Conference Series*, 935(1), 012069. <https://doi.org/10.1088/1742-6596/935/1/012069>
- Goodchild. (1992). Geographical information science. *International Journal of Geographical Information Systems*, 6(1), 31–45. <https://doi.org/10.1080/02693799208901893>
- Goodchild, M. F., & Glennon, A. (2008). Representation and Computation of Geographic Dynamics. In *Understanding Dynamics of Geographic Domains*. CRC Press.
- Goodchild, Michael & Yuan, May & Cova, Thomas. (2007). Towards A General Theory of Geographic Representation in GIS. *International Journal of Geographical Information Science*. 21. 239-260. 10.1080/13658810600965271.
- Goswami, P., Schlegel, P., Solenthaler, B., & Pajarola, R. (2010). Interactive SPH Simulation and Rendering on the GPU. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation* (p. 10 pages). The Eurographics Association. <https://doi.org/10.2312/SCA/SCA10/055-064>
- Gregory, J. *Game engine architecture*. crc Press, 2018.
- Gross, F., Petersen, L., Wallmeier, C., Barrett, R., Kwasnitschka, T., & Karstens, S. (2023). From coastal geomorphometry to virtual environments. *Frontiers in Marine Science*, 10, Article 1229489. <https://doi.org/10.3389/fmars.2023.1229489>
- Hägerstrand, T., Pred, A., & Törnqvist, G. (1981). *Space and Time in Geography : Essays Dedicated to Torsten Hägerstrand*. Lund: CWK Gleerup.
- Hajigholizadeh, M., Melesse, A. M., & Fuentes, H. R. (2018). Erosion and Sediment Transport Modelling in Shallow Waters: A Review on Approaches, Models and Applications. *International Journal of Environmental Research and Public Health*, 15(3), 518. <https://doi.org/10.3390/ijerph15030518>

- Harada, T. (n.d.). Chapter 29. Real-Time Rigid Body Simulation on GPUs. Retrieved from <https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-29-real-time-rigid-body-simulation-gpus>
- Harrap, R., Hutchinson, J., Sala, Z., Ondercin, M., Difrancesco, P. (2019). Our GIS is a Game Engine: Bringing Unity to Spatial Simulation of Rockfalls. *GeoComputation 2019*. The University of Auckland. Conference contribution. <https://doi.org/10.17608/k6.auckland.9848612.v2>
- Harris, R., O’Sullivan, D., Gahegan, M., Charlton, M., Comber, L., Longley, P., Brunsdon, C., Malleson, N., Heppenstall, A., Singleton, A., Arribas-Bel, D., & Evans, A. (2017). More bark than bytes? Reflections on 21+ years of geocomputation. *Environment and Planning. B, Urban Analytics and City Science*, 44(4), 598–617. <https://doi.org/10.1177/2399808317710132>
- Heath, B. L., Hill, R., & Ciarallo, F. (2009). A Survey of Agent-Based Modeling Practices (January 1998 to July 2008). *J. Artif. Soc. Soc. Simul.* [https://www.semanticscholar.org/paper/A-Survey-of-Agent-Based-Modeling-Practices-\(January-Heath-Hill/1e661d0aae853d98b7fa8eda47785d7de947dd43](https://www.semanticscholar.org/paper/A-Survey-of-Agent-Based-Modeling-Practices-(January-Heath-Hill/1e661d0aae853d98b7fa8eda47785d7de947dd43)
- Heisler, J., Glibert, P. M., Burkholder, J. M., Anderson, D. M., Cochlan, W., Dennison, W. C., Dortch, Q., Gobler, C. J., Heil, C. A., Humphries, E., Lewitus, A., Magnien, R., Marshall, H. G., Sellner, K., Stockwell, D. A., Stoecker, D. K., & Suddleson, M. (2008). Eutrophication and harmful algal blooms: A scientific consensus. *Harmful Algae*, 8(1), 3-13. <https://doi.org/10.1016/j.hal.2008.08.006>
- Helbig, C., Becker, A. M., & Schlink, U. (2022). A game engine based application for visualising and analysing environmental spatiotemporal mobile sensor data in an urban context. *Frontiers in Environmental Science*, 10, 123456. <https://doi.org/10.3389/fenvs.2022.952725>

- Her, Y., Yang, K., & Song, J.-H. (2021). Parallelization of a two-dimensional time-area watershed routing. *Environmental Modelling & Software*, 146, 105222. <https://doi.org/10.1016/j.envsoft.2021.105222>
- Herman, L., Juřík, V., Stachoň, Z., Vrbík, D., Russnák, J., & Řezník, T. (2018). Evaluation of user performance in interactive and static 3D maps. *ISPRS International Journal of Geo-Information*, 7(11), 415. <https://doi.org/10.3390/ijgi7110415>
- Hess, C., Härdtle, W., Kunz, M., Fichtner, A., & von Oheimb, G. (2018). A high-resolution approach for the spatiotemporal analysis of forest canopy space using terrestrial laser scanning data. *Ecol Evol*. <https://doi.org/10.1002/ece3.4193>
- Holm, S., & Schweier, J. (2024). Virtual forests for decision support and stakeholder communication. *Environmental Modelling & Software*, 180, 105678. <https://doi.org/10.1016/j.envsoft.2024.106159>
- Hongpan, N., Yong, G., & Zhongming, H. (2010). Application research of PhysX engine in virtual environment. 2010 International Conference on Audio, Language and Image Processing, 587–591. <https://doi.org/10.1109/ICALIP.2010.5684497>
- Hoon, M. and K. (2003). Enhancing Architectural Communication with Gaming Engines. *Connecting >> Crossroads of Digital Discourse* [Proceedings of the 2003 Annual Conference of the Association for Computer Aided Design In Architecture / ISBN 1-880250-12-8] Indianapolis (Indiana) 24-27 October 2003, Pp. 349-355. https://papers.cumincad.org/cgi-bin/works/paper/acadia03_045
- Horton, B. K., Kalia, R. K., Moen, E., Nakano, A., Nomura, K., Qian, M., Vashishta, P., & Hafreager, A. (2019). Game-Engine-Assisted Research platform for Scientific computing (GEARS) in Virtual Reality. *SoftwareX*, 9, 112–116. <https://doi.org/10.1016/j.softx.2019.01.009>

- Hosseini pour, E. Z., Vargas, S., O'Brien, K., & Kasraie, H. (2012). 2-D fine grid hydrodynamic modeling for more accurate floodplain mapping in Southern California. In World Environmental and Water Resources Congress 2012: Crossing Boundaries. American Society of Civil Engineers. <https://doi.org/10.1061/9780784412312.112>
- Huang, B., & Lin, H. (1999). GeoVR: a web-based tool for virtual reality presentation from 2D GIS data. *Computers & Geosciences*, 25(10), 1167-1175.
- Huang, Z., Wen, Y., Wang, Z., Ren, J., & Jia, K. (2022). Surface Reconstruction from Point Clouds: A Survey and a Benchmark. *arXiv*. <https://doi.org/10.48550/arXiv.2205.02413>
- ibbles. (2022). Inside Unreal - Advanced Niagara Effects. GitHub. Retrieved from <https://github.com/ibbles/LearningUnrealEngine/blob/master/Inside%20Unreal%20-%20Advanced%20Niagara%20Effects.md>
- Ihmsen, M., Akinci, N., Akinci, G., Teschner, M. (2012). Unified Spray, Foam and Bubbles for Particle-Based Fluids. *Computer Graphics Forum*, 31(8): 2371-2380.
- Iones, A., Krupkin, A., Sbert, M., & Zhukov, S. (2003). Fast, realistic lighting for video games. *IEEE Computer Graphics and Applications*, 23(3), 54–64. <https://doi.org/10.1109/MCG.2003.1198263>
- Jankowski, P., & Nyerges, T. (2001). GIS-Supported Collaborative Decision Making: Results of an Experiment. *Annals of the Association of American Geographers*, 91(1), 48–70. <https://doi.org/10.1111/0004-5608.00233>
- Jenson, S. K., & Domingue, J. O. (1988). Extracting topographic structure from digital elevation data for geographic information-system analysis. *Photogrammetric Engineering and Remote Sensing*, 54(11), 1593-1600.

- Jiang, H., Lin, G., Zhang, M. and Huang, W. (2007). Simulating the soil erosion process in reservoir area based on the technology of VRGIS. In: Proceedings of the 4th International Conference on Image and Graphics, ICIG 2007, pp. 990–996. doi:10.1109/ICIG.2007.45
- Jiang, J., Xiong, C., Xu, Z., Mei, L., Li, L., & Long, W. (2025). Game engine-empowered digital twin for seismic damage simulation of buildings. *Engineering Structures*, 330, 119839. <https://doi.org/10.1016/j.engstruct.2025.119839>
- Jianning Wang, Oliveira, M. M., & Kaufman, A. E. (2005). Reconstructing Manifold and Non-Manifold Surfaces from Point Clouds. *IEEE Visualization 2005 - (VIS'05)*, 53–53. *IEEE Visualization 2005 - (VIS'05)*. <https://doi.org/10.1109/VIS.2005.85>
- Johnson, I. (2011). Real-time particle systems in the Blender Game Engine. <https://www.semanticscholar.org/paper/Real-time-particle-systems-in-the-Blender-Game-Johnson/72ce162d21baaff9e33f57616d1f5d442735788f>
- Jong, J. D., Wormnes, K., & Tiso, P. (2014). Simulating Rigid-Bodies, Strings and Nets for Engineering Applications Using Gaming Industry Physics Simulators. <https://www.semanticscholar.org/paper/Simulating-Rigid-Bodies%2C-Strings-and-Nets-for-Using-Jong-Wormnes/8500a7aa240d68f9806ded23d29206f8a04b6e33>
- Joselli, M., Junior, J. R. da S., Clua, E. W., Montenegro, A., Lage, M., & Pagliosa, P. (2015). Neighborhood grid: A novel data structure for fluids animation with GPU computing. *Journal of Parallel and Distributed Computing*, 75, 20–28. <https://doi.org/10.1016/j.jpdc.2014.10.009>
- Jung, J.-K. (2009). 7 Computer-Aided Qualitative GIS: A Software-Level Integration of Qualitative Research and GIS. SAGE Publications. <https://doi.org/10.4135/9780857024541.D12>
- Justvg. (2019). Justvg/HydraulicErosion: Particle-based erosion on a heightmap. Retrieved from <https://github.com/justvg/HydraulicErosion>

- Karimov, V., Sheshukov, A., & Barnes, P. (2015). Impact of precipitation and runoff on ephemeral gully development in cultivated croplands. *Proceedings of the International Association of Hydrological Sciences*, 367, pp. 87–94. doi:10.5194/PIAHS-367-87-2015.
- Karssenbergh, D., & De Jong, K. (2005). Dynamic environmental modelling in GIS: 2. Modelling error propagation. *International Journal of Geographical Information Science*, 19(6), 623-637. <https://doi.org/10.1080/13658810500104799>
- Kaufman, A., & Bakalash, R. (1988). Memory and processing architecture for 3D voxel-based imagery. *IEEE Computer Graphics and Applications*, 8(6), 10–23. <https://doi.org/10.1109/38.20314>
- Kaufman, A., & Mueller, K. (2005). Overview of Volume Rendering. In C. Johnson & C. Hansen (Eds.), *Visualization Handbook* (pp. 127-XI). Academic Press. <https://doi.org/10.1016/B978-012387582-2/50009-5>
- Kavouras, I., Sardis, E., & Doulamis, N. (2023). A Low-Cost Gamified Urban Planning Methodology Enhanced with Co-Creation and Participatory Approaches. *Sustainability*, 15(3), 789. <https://doi.org/10.3390/su15032297>
- Keil, J., Edler, D., Schmitt, T., & Dickmann, F. (2021). Creating immersive virtual environments based on open geospatial data and game engines. *KN-Journal of Cartography and Geographic Information*, 71(1), 53-65.
- Kemp, K. K. (1996). Fields as a framework for integrating GIS and environmental process models. Part 1: Representing spatial continuity. *Transactions in GIS*, 1(3), 219-234. <http://dx.doi.org/10.1111/j.1467-9671.1996.tb00046.x>
- Kim, J. (2016). Modeling and optimization of a tree based on virtual reality for immersive virtual landscape generation. *Symmetry*, 8(9), 93.

- Kirchner, J. W. (2006). Getting the right answers for the right reasons: Linking measurements, analyses, and models to advance the science of hydrology. *Water Resources Research*, 42(3), 2005WR004362. <https://doi.org/10.1029/2005WR004362>
- Kirkbride, M. P. (1993). The temporal significance of transitions from melting to calving termini at glaciers in the central Southern Alps of New Zealand. *The Holocene*, 3(3), 232-239. <https://doi.org/10.1177/095968369300300305>
- Klein, B. (2016). Managing the scalability of visual exploration using game engines to analyse UHI scenarios. *Procedia Engineering*, 169, 272-279.
- Kohek, Š., Strnad, D., Žalik, B., & Kolmanič, S. (2019). Interactive synthesis and visualization of self-organizing trees for large-scale forest succession simulation. *Multimedia Systems*, 25, 213-227.
- Kolb, D., Kurtz, W., Weismüller, J., von Ramm, A., Ludwig, R., & Kranzlmüller, D. (2018, September 5). Visualization of climate simulation data in virtual reality using commercial game engines. In H.-J. Bungartz, D. Kranzlmüller, V. Weinberg, J. Weismüller, & V. Wohlgemutz (Eds.), *Adjunct Proceedings of the 32nd EnviroInfo Conference* (pp. 39–45). Shaker Verlag GmbH.
- Krištof, P., Beneš, B., Krivánek, J., & Št'ava, O. (2009). Hydraulic Erosion Using Smoothed Particle Hydrodynamics. *Computer Graphics Forum*, 28(2), 219-228. doi:10.1111/j.1467-8659.2009.01361.x
- Kriushichev, A. G., Silkina, N. S., & Nenakhova, E. A. (2020). Interactive Visualization for Simulating the Movement of Liquids and Gases. 2020 Global Smart Industry Conference (GloSIC), 17-19 November, 2020, Chelyabinsk, Russia. <https://doi.org/10.1109/GloSIC50886.2020.9267832>

- Krüger, M., Gilbert, D., Kuhlen, T. W., & Gerrits, T. (2024). Game Engines for Immersive Visualization: Using Unreal Engine Beyond Entertainment. *PRESENCE: Virtual and Augmented Reality*, 33, 31–55. https://doi.org/10.1162/pres_a_00416
- Kumar, M., Bhatt, Gopal, & Duffy, C. J. (2010). An object-oriented shared data model for GIS and distributed hydrologic models. *International Journal of Geographical Information Science*, 24(7), 1061–1079. <https://doi.org/10.1080/13658810903289460>
- Labovský, J., & Jelemenský, L. (2013). CFD-based atmospheric dispersion modeling in real urban environments. *Chemické zvesti*, 67(4), 738-748. <https://doi.org/10.2478/s11696-013-0388-7>
- Laine, S., & Karras, T. (2010). Efficient Sparse Voxel Octrees— Analysis, Extensions, and Implementation. NVIDIA Technical Report NVR-2010-001. Retrieved from https://research.nvidia.com/sites/default/files/pubs/2010-02_Efficient-Sparse-Voxel/laine2010tr1_paper.pdf
- Lakshmi, S. E., & Yarrakula, K. (2019). Review and critical analysis on digital elevation models. *Geofizika*, 35(2), 129–157. <https://doi.org/10.15233/gfz.2018.35.7>
- Laksono, D., & Aditya, T. (2019). Utilizing a game engine for interactive 3D topographic data visualization. *ISPRS International Journal of Geo-Information*, 8(8), 361. <https://doi.org/10.3390/ijgi8080361>
- Lane, L. J., Shirley, E. D., & Singh, V. P. (1988). Modelling erosion on hillslopes. *Modelling geomorphological systems*, 287-308.
- LanLou123. (2018). LanLou123/Webgl-Erosion: Interactive Erosion simulation in Web Browser. Retrieved from <https://github.com/LanLou123/Webgl-Erosion>

- Laube, P. (2015). The low hanging fruit is gone: achievements and challenges of computational movement analysis. *SIGSPATIAL Special*, 7(1), 3-10. <https://doi.org/10.1145/2782759.2782762>
- Lee, A., Chang, Y.-S., & Jang, I. (2020). Planetary-Scale Geospatial Open Platform Based on the Unity3D Environment. *Sensors*, 20(20), Article 5967. <https://doi.org/10.3390/s20205967>
- Lee, E. A., & Sirjani, M. (2018). What Good are Models? In K. Bae & P. C. Ölveczky (Eds.), *Formal Aspects of Component Software* (pp. 3–31). Springer International Publishing. https://doi.org/10.1007/978-3-030-02146-7_1
- Lenntorp, B. (1999). Time-geography-at the end of its beginning. *GeoJournal*, 48(3), 155-158.
- Lewis, M., & Jacobson, J. (2002). Game engines in scientific research—Introduction. *Communications of the ACM*, 45(1), 27–31. <https://doi.org/10.1145/502269.502288>
- Lidberg, W., Nilsson, M., Lundmark, T., & Ågren, A. M. (2017). Evaluating preprocessing methods of digital elevation models for hydrological modelling. *Hydrological Processes*, 31(26), 4660–4668. <https://doi.org/10.1002/hyp.11385>
- Li, H., Yang, X., Zhang, G., et al. (2022). Vox-Surf: Voxel-Based Implicit Surface Representation. *IEEE Transactions on Visualization and Computer Graphics*. <https://doi.org/10.1109/TVCG.2022.3225844>
- Lin, X., Shi, M., & Li, C. (2017). Design and implement of swarm self-organization model based on Unity3D. 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 15-17 December, 2017, Chengdu, China. <https://doi.org/10.1109/ITNEC.2017.8284926>
- Lippe, M., Bithell, M., Gotts, N., Natalini, D., Barbrook-Johnson, P., Giupponi, C., Hallier, M., Hofstede, G. J., Le Page, C., Matthews, R. B., Schlüter, M., Smith, P., & Thellmann,

- K. (2019). Using agent-based modelling to simulate social-ecological systems across scales. *GeoInformatica*, 23(2), 269-298. <https://doi.org/10.1007/s10707-018-00337-8>
- Li, S., & Duffy, C. J. (2011). Fully coupled approach to modeling shallow water flow, sediment transport, and bed evolution in rivers. *Water Resources Research*, 47(3). <https://doi.org/10.1029/2010WR009751>
- Liu, Q., Sha, D., Liu, W., Houser, P., Zhang, L., Hou, R., Lan, H., Flynn, C., Lu, M., Hu, T., & Yang, C. (2020). Spatiotemporal Patterns of COVID-19 Impact on Human Activities and Environment in Mainland China Using Nighttime Light and Air Quality Data. *Remote Sensing*, 12(10), 1576. <https://doi.org/10.3390/rs12101576>
- Liu, X., & Beljadid, A. (2017). A coupled numerical model for water flow, sediment transport and bed erosion. *Computers & Fluids*, 154, 273–284. <https://doi.org/10.1016/j.compfluid.2017.06.013>
- Liu, Y. Y., & Sanyal, J. (2020). Scalable data-intensive geocomputation: A design for real-time continental flood inundation mapping. In *Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI (SMC 2020)* (Vol. 1315, pp. 130–144). Springer.
- Li, X. (2007). The First Law of Geography and Spatial-Temporal Proximity. *Chinese Journal of Nature*.
- Li, Z., Liu, W. Z., Zhang, X. C., & Zheng, F. L. (2011). Assessing the site-specific impacts of climate change on hydrology, soil erosion and crop yields in the Loess Plateau of China. *Climatic Change*, 105(2), 223-242. <https://doi.org/10.1007/s10584-010-9875-9>
- Lobell, D. B., & Asseng, S. (2017). Comparing estimates of climate change impacts from process-based and statistical crop models. *Environmental Research Letters*, 12(1), 015001. <https://doi.org/10.1088/1748-9326/aa518a>

- Longley, P. A. (2003). Geographical Information Systems: developments in socio-economic data infrastructures. *Progress in Human Geography*, 27(1), 114–121. <https://doi.org/10.1191/0309132503ph415pr>
- Longley, P. A. (2004). Geographical Information Systems: on modelling and representation. *Progress in Human Geography*, 28(1), 108-116.
- Longley, P. A., Goodchild, M. F., Maguire, D. J., & Rhind, D. W. (2015). *Geographic Information Science and Systems* (4th ed.). Wiley.
- Long, S., Wong, K. K. L., Fan, X., Guo, X., & Yang, C. (2023). Smoothed particle hydrodynamics method for free surface flow based on MPI parallel computing. *Frontiers in Physics*, 11, 1141972. <https://doi.org/10.3389/fphy.2023.1141972>
- López-Vicente, M., & Navas, A. (2010). Routing runoff and soil particles in a distributed model with GIS: Implications for soil protection in mountain agricultural landscapes. *Land Degradation & Development*, 21(2), 100–109. <https://doi.org/10.1002/ldr.901>
- Lorensen, W., & Cline, H. (1987, July). Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, vol. 21, no. 3, pp. 163-169.
- Lowood, H. (2014). *Game Engines and Game History*. Retrieved from <https://www.kinephanos.ca/2014/game-engines-and-game-history/#01>
- Lü, G., Batty, M., Strobl, J., Lin, H., Zhu, A. X., & Chen, M. (2019). Reflections and speculations on the progress in Geographic Information Systems (GIS): a geographic perspective. *International journal of geographical information science*, 33(2), 346-367.
- Luebke, D., Reddy, M., Cohen, J., Varshney, A., Watson, B., & Huebner, R. A. (2012, March 28). Level of Detail for 3D Graphics. <https://www.semanticscholar.org/paper/Level-of-Detail-for-3D-Graphics-Luebke-Reddy/39308ca1f1bd233da1ee716d8d69096d70202dd2>

- Lugina, F. P., Uchida, T., & Kawahara, Y. (2022). Study on numerical calculation of flow structures in a curved open channel with advanced depth-integrated models. IOP Conference Series. Earth and Environmental Science, 1065(1), 12004-.
<https://doi.org/10.1088/1755-1315/1065/1/012004>
- Lv, Q., Lu, T., Huang, F., et al. (2024). Research on Building Holes Repair and Modeling Based on Point Clouds from Oblique Photography Reconstruction. IEEE International Geoscience. <http://dx.doi.org/10.1109/IGARSS53475.2024.1064104>
- MacEachren, A. M., & Kraak, M.-J. (2001). Research challenges in geovisualization. Cartography and Geographic Information Science, 28(1), 3–23.
- Mahayudin, M. H., & Mat, R. C. (2016). Online 3D terrain visualisation using Unity 3D game engine: A comparison of different contour intervals terrain data draped with UAV images. IOP Conference Series: Earth and Environmental Science, 37, 012002.
<https://doi.org/10.1088/1755-1315/37/1/012002>
- Manson, S. M. (2001). Simplifying complexity: a review of complexity theory. Geoforum, 32(3), 405-414. [https://doi.org/10.1016/S0016-7185\(00\)00035-X](https://doi.org/10.1016/S0016-7185(00)00035-X)
- Manyoky, M., Wissen Hayek, U., Heutschi, K., Pieren, R., & Grêt-Regamey, A. (2014). Developing a GIS-based visual-acoustic 3D simulation for wind farm assessment. ISPRS International Journal of Geo-Information, 3(1), 29-48.
- Massey, D. (2004, March 31). Space-Time, ‘Science’ and the Relationship between Physical Geography and Human Geography. Transactions of the Institute of British Geographers, 24(3), 261–276.
- McAlpine, J., & Ruby, M. (2004). Using CFD to study air quality in urban microenvironments. Environmental Science, Engineering, Vol. II, EnviroComp Institute, Fremont, 1-31.

- Meentemeyer. (1989). Geographical perspectives of space, time, and scale. *Landscape Ecology*, 3(3-4), 163–173. <https://doi.org/10.1007/BF00131535>
- Mehendale, N., & Neoge, S. (2020). Review on Lidar Technology. SSRN. Retrieved from <http://dx.doi.org/10.2139/ssrn.3604309>
- Mei, X., Decaudin, P., & Hu, B.-G. (2007). Fast Hydraulic Erosion Simulation and Visualization on GPU. *Computer Graphics and Applications*, 2007. PG '07. 15th Pacific Conference on, 47-55. IEEE. <https://doi.org/10.1109/PG.2007.15>
- Meier, S. (n.d.). Space-Time Prisms Revisited Deriving Hägerstrand's space-time visualisations from mobility isochrones. Retrieved from <https://sebastian-meier.github.io/d3.isochrone2prism/>
- Meinen, B. U., & Robinson, D. T. (2020). Mapping erosion and deposition in an agricultural landscape: Optimization of UAV image acquisition schemes for SfM-MVS. *Remote Sensing of Environment*, 239, 111666. <https://doi.org/10.1016/j.rse.2020.111666>
- Meinen, B. U., & Robinson, D. T. (2021). From hillslopes to watersheds: Variability in model outcomes with the USLE. *Environmental Modelling & Software*, 146, 105229. <https://doi.org/10.1016/j.envsoft.2021.105229>
- Messaoudi, F., Simon, G., & Ksentini, A. (2015). Dissecting games engines: The case of Unity3D. 2015 International Workshop on Network and Systems Support for Games (NetGames), 1–6. 2015 International Workshop on Network and Systems Support for Games (NetGames). <https://doi.org/10.1109/NetGames.2015.7382990>
- Messler, J., Husser, N., & Brizzolara, S. (2021). The impacts of model uncertainty on RANS CFD simulations of a high-speed craft. *FAST Conference Proceedings*, 2021, 026. <https://doi.org/10.5957/fast-2021-026>

- Millington, J. D. A., & Wainwright, J. (2017). Mixed qualitative-simulation methods: Understanding geography through thick and thin. *Progress in Human Geography*, 41(1). <https://doi.org/10.1177/0309132515627021>
- Mocnik, F.-B. (2022). Putting Geographical Information Science in Place: Towards Theories of Platial Information and Platial Information Systems. *Progress in Human Geography*, 46(1), 31-45. <https://doi.org/10.1177/03091325221074023>
- Mollalo, A., Vahedi, B., Rivera, K. (2020). GIS-based spatial modeling of COVID-19 incidence rate in the continental United States. *Science of The Total Environment*, Volume 728, 138884, ISSN 0048-9697. <https://doi.org/10.1016/j.scitotenv.2020.138884>
- Mower, J. (2016). Real-Time Drainage Basin Modeling Using Concurrent Compute Shaders. <https://cartogis.org/docs/proceedings/2016/Mower.pdf>
- Mount, D. (2018). Solid Modeling and Mesh Representation. <https://www.semanticscholar.org/paper/Solid-Modeling-and-Mesh-Representation-Mount/5e92061229b1cc5f6cd3b6979c6d945eedc51339>
- Müller, M., Charypar, D., and Gross, M. (2003). Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation (SCA '03)*. Eurographics Association, Goslar, DEU, 154–159.
- Müller, M., Heidelberger, B., Hennix, M., & Ratcliff, J. (2007). Position based dynamics. *J. Vis. Comun. Image Represent.* 18, 2 (April, 2007), 109–118. <https://doi.org/10.1016/j.jvcir.2007.01.005>
- Muin, M., Idris, K. and Yuanita, N. (2016). Application of large-scale 3D non-orthogonal boundary fitted sediment transport model and small-scale approach for offshore structure in Cimanuk Delta North Java Sea. *Journal of Engineering and Technological Sciences*, 48(3), pp. 301–319. doi:10.5614/j.eng.technol.sci.2016.48.3.5

- Murdock, F. F. (1894). THE RELATION OF GEOGRAPHY AND PHYSICS.—(II). The Journal of Education, 39(15 (965)), 229–229. <http://www.jstor.org/stable/44043178>
- Musgrave, F., Kolb, C. E., Mace, R. (1989). The synthesis and rendering of eroded fractal terrain. SIGGRAPH '89: In proceedings of the annual conference of computer graphics and interactive techniques, 41-55, 1989. <https://doi.org/10.1145/74334.74337>
- Nakane, K., & Shinozaki, K. (1978). A mathematical model of the behavior and vertical distribution of organic carbon in forest soils. Japanese Journal of Ecology, 28(2), 111-122. https://doi.org/10.18960/seitai.28.2_111
- National Research Council. (1998). People and Pixels: Linking Remote Sensing and Social Science. Washington, DC: The National Academies Press. <http://dx.doi.org/10.17226/5963>
- Neidhold, B., Wacker, M., & Deussen, O. (2005). Interactive physically based Fluid and Erosion Simulation. NPH. Retrieved from <http://dx.doi.org/10.2312/NPH/NPH05/025-032>
- Newell, R., Dring, C., & Willyono, E. (2024). Imagining just and sustainable food futures: Using interactive visualizations to explore the possible land uses and food systems approaches in Revelstoke, Canada. Land, 13(9), 1345. <https://doi.org/10.3390/land13091345>
- Ngo, T.A., See, L. (2012). Calibration and Validation of Agent-Based Models of Land Cover Change. In: Heppenstall, A., Crooks, A., See, L., Batty, M. (eds) Agent-Based Models of Geographical Systems. Springer, Dordrecht. https://doi-org.proxy.lib.uwaterloo.ca/10.1007/978-90-481-8927-4_10
- Noardo, F. (2022). Multisource spatial data integration for use cases applications. Transactions in GIS, 26(5), e12987. <https://doi.org/10.1111/tgis.12987>

- Noor, N. M., Abdullah, A., & Hashim, M. (2018). Remote sensing UAV/drones and its applications for urban areas: a review. *IOP Conference Series: Earth and Environmental Science*, 169, 012003. <http://dx.doi.org/10.1088/1755-1315/169/1/012003>
- Nystrom, R. (n.d.). Game loop. *Game Programming Patterns*. Retrieved March 16, 2025, from <https://gameprogrammingpatterns.com/game-loop.html>
- Obe, R. O., & Hsu, L. S. (2015). *PostGIS in Action* (2nd ed.). Manning Publications.
- O'Callaghan, J. F., & Mark, D. M. (1984). The extraction of drainage networks from digital elevation data. *Computer Vision, Graphics, and Image Processing*, 28(3), 323-344. [https://doi.org/10.1016/S0734-189X\(84\)80011-0](https://doi.org/10.1016/S0734-189X(84)80011-0)
- Odhner, A., & Fleming, J. (2009). Hydraulic Erosion of Soil and Terrain. Retrieved from http://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S09/final_projects/odhner_fleming.pdf
- Ohashi, Y., Torii, T., Ishigami, Y., & Goto, E. (2020). Estimation of the light interception of a cultivated tomato crop canopy under different furrow distances in a greenhouse using the ray tracing. *Journal of Agricultural Meteorology*, 76(4), 188–193. <https://doi.org/10.2480/agrmet.D-20-00030>
- Ohuri, K., Biljecki, F., Ledoux, H., & Stoter, J. (2013). Manipulating higher dimensional spatial information. *Semantic Scholar*. <https://doi.org/10.4233/UUID:C4751AD5-E008-48A6-806B-AF38099E0B3A>
- Ondercin, M. (2016). *An Exploration of Rockfall Modelling Through Game Engines* (Doctoral dissertation). <http://hdl.handle.net/1974/14330>
- Openshaw, S., 1983. The modifiable areal unit problem. *Concepts and Techniques in Modern Geography* 48. Geo Books, Norwich, UK.

- Oracle Corporation. (n.d.). Oracle Spatial and Graph Developer's Guide. Available at <https://docs.oracle.com>
- Oreskes, N., Shrader-Frechette, K., & Belitz, K. (1994). Verification, Validation, and Confirmation of Numerical Models in the Earth Sciences. *Science*, 263(5147), 641-646. <https://doi.org/10.1126/science.263.5147.641>
- O'Sullivan, D., & Manson, S. M. (2015). Do Physicists Have Geography Envy? And What Can Geographers Learn from It? *Annals of the Association of American Geographers*, 105(4), 704–722. <http://www.jstor.org/stable/24537865>
- Paraline Games. (2016, November 01). Nvidia GameWorks Unreal Engine 4 - Physics Simulation. Retrieved from <https://youtu.be/-S8wq0dz1H4>
- PavelDoGreat. (2019). PavelDoGreat/WebGL-Fluid-Simulation: Play with fluids in your browser (works even on mobile). Retrieved from <https://github.com/PavelDoGreat/WebGL-Fluid-Simulation>
- Pazouki, A., Serban, R., & Negrut, D. (2014). A Lagrangian-Lagrangian framework for the simulation of rigid and deformable bodies in fluid. *Multibody Dynamics: Computational Methods and Applications*, 33–52. Springer International Publishing. https://doi.org/10.1007/978-3-319-07260-9_2
- Pazzaglia, F. J. (2003). Landscape evolution models. In *Developments in Quaternary Sciences* (Vol. 1, pp. 247–274). Elsevier. [https://doi.org/10.1016/S1571-0866\(03\)01012-1](https://doi.org/10.1016/S1571-0866(03)01012-1)
- Pearman, G. I., Hyson, P., & Fraser, P. J. (1983). The global distribution of atmospheric carbon dioxide: 1. Aspects of observations and modeling. *Journal of Geophysical Research: Oceans*, 88(C6), 3581-3590. <https://doi.org/10.1029/JC088iC06p03581>
- Peek, A., Martin, M., & Kolston, S. (2021). The Virtual Reality of GIScience. <https://doi.org/10.25436/E2J015>

- Perhac, J., Zeng, W., Asada, S., Arisona, S. M., Schubiger, S., Burkhard, R., & Klein, B. (2017). Urban Fusion: Visualizing Urban Data Fused with Social Feeds via a Game Engine. 2017 21st International Conference Information Visualisation (IV). doi:10.1109/iv.2017.33
- Peuquet, D. J. (1994). It's About Time: A Conceptual Framework for the Representation of Temporal Dynamics in Geographic Information Systems. *Annals of the Association of American Geographers*, 84(3), 441–461. <https://doi.org/10.1111/j.1467-8306.1994.tb01869.x>
- Peuquet, D. J. (2002). *Representations of Space and Time*. Guilford Press.
- Plane Professional: X-Plane. (2018, May 24). Retrieved from <https://www.x-plane.com/pro/>
- Pontius, R. G. (2022). *Metrics That Make a Difference: How to Analyze Change and Error*. Springer. <https://doi.org/10.1007/978-3-030-70765-1>
- Pontius, R. G., Walker, R., Yao-Kumah, R., Arima, E. Y., Aldrich, S., Caldas, M., & Vergara, D. (2007). Accuracy assessment for a simulation model of Amazonian deforestation. *Annals of the Association of American Geographers*, 97(4), 677–695. <https://doi.org/10.1111/j.1467-8306.2007.00577.x>
- Powers, J. G., Klemp, J. B., Skamarock, W. C., Davis, C. A., Dudhia, J., Gill, D. O., Coen, J. L., Gochis, D. J., Ahmadov, R., Peckham, S. E., Grell, G. A., Michalakes, J., Trahan, S., Benjamin, S. G., Alexander, C. R., Dimego, G. J., Wang, W., Schwartz, C. S., Romine, G. S., Liu, Z., Snyder, C., Chen, F., Barlage, M. J., Yu, W., & Duda, M. G. (2017). The Weather Research and Forecasting (WRF) Model: Overview, System Efforts, and Future Directions. *Bulletin of the American Meteorological Society*, 98(8), 1717-1737. <https://doi.org/10.1175/BAMS-D-15-00308.1>
- Predescu, A., Mocanu, M. (2020). A data driven survey of video games. 2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), 1–6. <https://doi.org/10.1109/ECAI50035.2020.9223203>

- Price, J. F. (2006). Lagrangian and Eulerian Representations of Fluid Flow: Kinematics and the Equations of Motion. Woods Hole Oceanographic Institution. Retrieved from <https://www.whoi.edu/science/po/people/jprice/class/elreps.pdf>
- Priya, S., & Shibasaki, R. (2001). National spatial crop yield simulation using GIS-based crop production model. *Ecological Modelling*, 136(2-3), 113-129. [https://doi.org/10.1016/S0304-3800\(00\)00364-1](https://doi.org/10.1016/S0304-3800(00)00364-1)
- Prytuła, T. (2024). Graph representations of 3D data for machine learning. arXiv. <https://doi.org/10.48550/arXiv.2408.08336>
- Purves, R. S., Winter, S., & Kuhn, W. (2019). Places in information science. *Journal of the Association for Information Science and Technology*, 70(11), 1173-1182.
- Qin, C., Zhu, A.-X., Pei, T., Li, B., Zhou, C., & Yang, L. (2007). An adaptive approach to selecting a flow-partition exponent for a multiple-flow-direction algorithm. *International Journal of Geographical Information Science*, 21(4), 443-458. <https://doi.org/10.1080/13658810601073240>
- Quinn, P., Beven, K., Chevallier, P., & Planchon, O. (1991). The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models. *Hydrological Processes*, 5(1), 59-79. <https://doi.org/10.1002/hyp.3360050106>
- Rabbani, A. H., Khiat, S. (2020). Fast Eulerian Fluid Simulation In Games Using Poisson Filters. Retrieved from <https://www.shahinrabbani.ca/torch2pd.html>
- Rafiee, A., Van der Male, P., Dias, E., & Scholten, H. (2017). Developing a wind turbine planning platform: Integration of “sound propagation model–GIS-game engine” triplet. *Environmental Modelling and Software*, 95, 326–343. <https://doi.org/10.1016/j.envsoft.2017.06.019>

- Rafiee, A., Van der Male, P., Dias, E., & Scholten, H. (2018). Interactive 3D geodesign tool for multidisciplinary wind turbine planning. *Journal of Environmental Management*, 205, 107–124. <https://doi.org/10.1016/j.jenvman.2017.09.042>
- Rahman, A.A., Zlatanova, S., & Pilouk, M. (2001). The 3D GIS software development : global efforts from researchers and vendors.
- Rahman, M. A., & Asyhari, A. T. (2019). The emergence of Internet of Things (IoT): Connecting anything, anywhere. *Computers*, 8(2), 40. <https://doi.org/10.3390/computers8020040>
- Ragia, L., & Fiume, E. (2007). Challenges in geovisualization. *IPSI Transactions*.
- Raman, R. K., Dewang, Y., & Raghuwanshi, J. (2018). A review on applications of computational fluid dynamics. *International Journal of LNCT*, 2(6), 137–143. Retrieved from <https://ijlnct.org/paper-directory/volume-2/issue-6/6.pdf>
- Raza, A., Ahrends, H., Habib-Ur-Rahman, M., & Gaiser, T. (2021). Modeling Approaches to Assess Soil Erosion by Water at the Field Scale with Special Emphasis on Heterogeneity of Soils and Crops. *Land*, 10(4), 422. <https://doi.org/10.3390/land10040422>
- Redmond, P., Castello, J., Calderón Trilla, J. M., & Kuper, L. (n.d.). Exploring the theory and practice of concurrency in the entity-component-system pattern. University of California, Santa Cruz & University of Maryland. Retrieved from <https://users.soe.ucsc.edu/~lkuper/papers/core-ecs-draft.pdf>
- Redon, S., Kheddar, A. & Coquillart, S. (2003, May 6). Fast Continuous Collision Detection between Rigid Bodies. *Computer Graphics Forum*, 21: 279-287. <https://doi.org/10.1111/1467-8659.t01-1-00587>

- Refsgaard, J. C. (1996). Terminology, Modelling Protocol and Classification of Hydrological Model Codes. In *Distributed Hydrological Modelling* (pp. 17–40). Kluwer. https://doi.org/10.1007/978-94-009-0257-2_2
- Rehn, A. (n.d.). Unreal Engine for Research. Retrieved from <https://ue4research.org/>
- Resch, B., Wohlfahrt, R., & Wosniok, C. (2014). Web-based 4D visualization of marine geo-data using WebGL. *Cartography and Geographic Information Science*, 41(3), 235–247. <https://doi.org/10.1080/15230406.2014.901901>
- Reyer, C. P. O., Brouwers, N., Rammig, A., Brook, B. W., Epila, J., Grant, R. F., Holmgren, M., Langerwisch, F., Leuzinger, S., Lucht, W., ... & Perring, M. P. (2015). Forest resilience and tipping points at different spatio-temporal scales: approaches and challenges. *Journal of Ecology*, 103(1), 5-15. <https://doi.org/10.1111/1365-2745.12337>
- Reynolds, C. W. (1987). Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics*, 21(4), 25-34. <https://doi.org/10.1145/37402.37406>
- Richter, S. R., AlHaija, H. A., Koltun, V. (2021). Enhancing Photorealism Enhancement. <https://doi.org/10.48550/arxiv.2105.04619>
- Rickards, M. A., Romanini, M., & De Gregori, G. (2021). The Effects and Innovations in Gaming and Digital Entertainment Forced by the Pandemic. In *Digital Transformation in a Post-Covid World* (pp. 171-180). CRC Press.
- Ricker, B. A., Rickles, P. R., Fagg, G. A., & Haklay, M. E. (2020). Tool, toolmaker, and scientist: case study experiences using GIS in interdisciplinary research. *Cartography and Geographic Information Science*, 47(4), 350–366. <https://doi.org/10.1080/15230406.2020.1748113>
- Rittel, H. W., & Webber, M. M. (1973). Dilemmas in a general theory of planning. *Policy sciences*, 4(2), 155-169.

- Roberts, E. (2012). Geography and the visual image: A hauntological approach. *Progress in Human Geography*, 37(3), 294-303. <https://doi.org/10.1177/0309132512460902>
- Roberts, M. J., Braun, N. O., Sinclair, T. R., Lobell, D. B., & Schlenker, W. (2017). Comparing and combining process-based crop models and statistical models with some implications for climate change. *Environmental Research Letters*, 12(9), 095010. <https://doi.org/10.1088/1748-9326/aa7f33>
- Robinson, D. T. (2025). Personal communication.
- Roth, R. E., Çöltekin, A., Delazari, L., Filho, H. F., Griffin, A., Hall, A., ... van Elzakker, C. P. J. M. (2017). User studies in cartography: opportunities for empirical research on interactive maps and visualizations. *International Journal of Cartography*, 3(sup1), 61–89. <https://doi.org/10.1080/23729333.2017.1288534>
- Rounsevell, M. D. A., Robinson, D. T., & Murray-Rust, D. (2012). From actors to agents in socio-ecological systems models. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1586), 259-269. <https://doi.org/10.1098/rstb.2011.0187>
- Rundel, S., & De Amicis, R. (2023). Leveraging digital twin and game-engine for traffic simulations and visualizations. *Frontiers in Virtual Reality*, 4, 1048753. <https://doi.org/10.3389/frvir.2023.1048753>
- Šaška, Č., Stachoň, Z., Sedlák, M., Chmelík, J., Herman, L., Kubíček, P., Šašinková, A., Doležal, M., Tejkl, H., Urbánek, T., Svatoňová, H., Ugwitz, P., & Juřík, V. (2019). Collaborative immersive virtual environments for education in geography. *ISPRS International Journal of Geo-Information*, 8(1), 3. <https://doi.org/10.3390/ijgi8010003>
- Schild, K. M., Renshaw, C. E., Benn, D. I., Luckman, A., Hawley, R. L., How, P., Trusel, L., Cottier, F. R., Pramanik, A., & Hulton, N. R. J. (2018). Glacier calving rates due to subglacial discharge, fjord circulation, and free convection. *Journal of Geophysical Research: Earth Surface*, 123(10). <https://doi.org/10.1029/2017JF004520>

- Schmalstieg, D., & Stork, A. (2019). Unified Patterns for Realtime Interactive Simulation in Games and Digital Storytelling. *IEEE Computer Graphics and Applications*. doi:10.1109/MCG.2018.2889218
- Schmid, F., & Burrell, G. (2016). CFD Analysis Challenges in Building Simulation for SimBuild2004 Conference. *Proceedings of the SimBuild2004 Conference*, 18 July 2016.
- Schütz, M. (2015). Potree: rendering large point clouds in web browsers [Diploma Thesis, Technische Universität Wien]. *repositUM*. <https://doi.org/10.34726/hss.2015.27710>
- Seppelt, R., Müller, F., Schröder, B., & Volk, M. (2009). Challenges of simulating complex environmental systems at the landscape scale: A controversial dialogue between two cups of espresso. *Ecological Modelling*, 220, issue 24, p. 3481-3489. <https://doi.org/10.1016/j.ecolmodel.2009.09.009>
- Shahabinejad, N., Mahmoodabadi, M., Jalalian, A., & Chavoshi, E. (2019). In situ field measurement of wind erosion and threshold velocity in relation to soil properties in arid and semiarid environments. *Environmental Earth Sciences*, 78(16), Article 501. <https://doi.org/10.1007/s12665-019-8508-5>
- Shaker, N., Togelius, J., & Nelson, M. J. (2016). Fractals, noise and agents with applications to landscapes. In N. Shaker, J. Togelius, & M. J. Nelson (Eds.), *Procedural Content Generation in Games* (pp. 57–72). Springer International Publishing. https://doi.org/10.1007/978-3-319-42716-4_4
- Shannon, J., & Walker, K. (2018). Opening GIScience: A process-based approach. *International Journal of Geographical Information Science*, 32(10), 1911–1926. <https://doi.org/10.1080/13658816.2018.1464167>
- Shen, D. Y., Ma, A. N., Lin, H., Nie, X. H., Mao, S. J., Zhang, B., & Shi, J. J. (2004). A new approach for simulating water erosion on hillslopes. *International Journal of Remote Sensing*, 25(9), 1791–1791. <https://doi.org/10.1080/01431160410001675514>

- Shibasaki, R., & Huang, S. (2001). Integration of Observational Data and Behavioral Models for Spatio-Temporal Interpolation—Application to Reconstructing Long-Term Land Use and Land Cover Changes. <https://www.semanticscholar.org/paper/Integration-of-Observational-Data-and-Behavioral-%E2%80%94Shibasaki-Huang/65a74de13c07e537204633307738761d92f5fcfb>
- Shibata, R., Ozaki, T., Kubota, S., & Yasumuro, Y. (2025). Visualization Method of Heat Risk through Light Environment Reproduction Considering Cloud Coverage. *E3S Web of Conferences*, 612, Article 02001. <https://doi.org/10.1051/e3sconf/202561202001>
- Shi, Z. H., Yue, B. J., Wang, L., Fang, N. F., Wang, D., & Wu, F. Z. (2013). Effects of mulch cover rate on interrill erosion processes and the size selectivity of eroded sediment on steep slopes. *Soil Science Society of America Journal*, 77(1), 257-267. <https://doi.org/10.2136/sssaj2012.0273>
- Shrivastava, S. (2020). A Review of Spatial Big Data Platforms, Opportunities, and Challenges. *IETE Journal of Education*, 61(2), 80–89. <https://doi.org/10.1080/09747338.2020.1835564>
- Sidle, R. C. (2021). Strategies for smarter catchment hydrology models: Incorporating scaling and better process representation. *Geoscience Letters*, 8(1), 24. <https://doi.org/10.1186/s40562-021-00193-9>
- Simpson, G., & Castelltort, S. (2006). Coupled model of surface water flow, sediment transport and morphological evolution. *Computers & Geosciences*, 32(10), 1600–1614. <https://doi.org/10.1016/j.cageo.2006.02.020>
- Singh, R. P., Rashmi Dubey, & Sugandha Agarwal. (2016). Reinforcing Particle System Effects Using Object-Oriented Approach and Real-Time Fluid Dynamics for Next-Generation 3D Gaming. 69–79. https://doi.org/10.1007/978-981-10-0448-3_6

- Singleton, A., & Arribas-Bel, D. (2019). Geographic Data Science. *Geographical Analysis*, Special Issue, first published online April 4, 2019. <https://doi.org/10.1111/gean.12194>
- Skidmore, A. (Ed.). (2002). *Application of remote sensing and geographic information systems in wildlife mapping and modelling*. CRC Press. <https://www.taylorfrancis.com/chapters/mono/10.4324/9780203302217-14>
- Skidmore, Prins, H., & Skidmore, A. (2002). *Environmental modelling with GIS and remote sensing*. Taylor & Francis.
- Skinner, C. J., & Coulthard, T. J. (2023). Testing the sensitivity of the CAESAR-Lisflood landscape evolution model to grid cell size. *Earth Surface Dynamics*, 11(4), 695–711. <https://doi.org/10.5194/esurf-11-695-2023>
- Soni, L., & Kaur, A. (2024). Merits and Demerits of Unreal and Unity: A Comprehensive Comparison. 2024 International Conference on Computational Intelligence for Green and Sustainable Technologies (ICCIGST), 1–5. 2024 International Conference on Computational Intelligence for Green and Sustainable Technologies (ICCIGST). <https://doi.org/10.1109/ICCIGST60741.2024.10717602>
- Spokas, K., & Forcella, F. (2006). Estimating hourly incoming solar radiation from limited meteorological data. *Weed Science*, 54(1), 182–189. <https://doi.org/10.1614/WS-05-098R.1>
- Stefanidis, A., Crooks, A., & Radzikowski, J. (2013). Harvesting Ambient Geospatial Information from Social Media Feeds. *GeoJournal*, 78, 319-338. <http://dx.doi.org/10.1007/s10708-011-9438-2>
- Stewart, R., Piburn, J., Sorokine, A., Myers, A., & White, D. A. (2015). *World Spatiotemporal Analytics and Mapping Project (WSTAMP): Discovering, Exploring, and Mapping Spatiotemporal Patterns across the World's Largest Open Source Geographic Data Sets (Vol. 2)*. Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States).

- Stoter, J. E., & Zlatanova, S. (2003). 3D GIS, where are we standing?. In ISPRS Joint Workshop on 'Spatial, Temporal and multi-dimensional data modelling and analysis', Québec, October, 2003.
- Strimas-Mackey, M. (2020, March 20). Fishnets and honeycomb: Square vs. hexagonal spatial grids. Matt Strimas-Mackey.
- Strodict, K., & Schattkowsky, T. (2024). Visual Scripting in Unity: A Comparative Analysis of Existing Frameworks. *Proceedings of the ACM/IEEE 8th International Workshop on Games and Software Engineering*, 44–49. <https://doi.org/10.1145/3643658.3643921>
- Stumpf, A., Malet, J.-P., Allemand, P., Pierrot-Deseilligny, M., & Skupinski, G. (2015). Ground-based multi-view photogrammetry for the monitoring of landslide deformation and erosion. *Geomorphology*, 231, 130-145. <https://doi.org/10.1016/j.geomorph.2014.10.039>
- Subramaniyam, B., Shylesh, D., Ramasamy, J., & Kumar, N. (2024). A Virtual Reality Tool for Accuracy Assessment of 3D Models in an Immersive Virtual Environment. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 10(4), 333–340. <https://doi.org/10.5194/isprs-annals-X-4-2024-333-2024>
- Sugam, S., Shashi, G., & Udoyara Sunday, T. (2013). Parametric database approach integration for handling temporal data in GIS. *Geo-spatial Information Science*, 16(2), 91-99.
- Sun, T., Meakin, P., Jøssang, T., & Schwarz, K. (1996). A Simulation Model for Meandering Rivers. *Water Resources Research*, 32(9), 2937-2954. <https://doi.org/10.1029/96WR00998>
- Sun, Y., Wang, H., Zhang, Z., Diels, C., & Asadipour, A. (2024). Executing realistic earthquake simulations in unreal engine with material calibration. *Computers & Graphics*, 124, Article 104091. <https://doi.org/10.1016/j.cag.2024.104091>

- Takahashi, D. (2021, March 30). Epic Games teams up with Cesium to bring 3D geospatial data to Unreal. Retrieved from <https://venturebeat.com/2021/03/30/epic-games-teams-up-with-cesium-to-bring-3d-geospatial-data-to-unreal/>
- Tang, H., & Wu, X. G. (2010). CFD and GFD hybrid approach for simulation of multi-scale coastal ocean flow. *Environmental Science, Engineering*.
- Tarboton, D. G. (1997). A new method for the determination of flow directions and upslope areas in grid digital elevation models. *Water Resources Research*, 33(2), 309-319. <https://doi.org/10.1029/96WR03137>
- Tatarchuk, N. (2009). Advances in real-time rendering in 3D graphics and games I. ACM SIGGRAPH 2009 Courses, 1–1. SIGGRAPH09: Special Interest Group on Computer Graphics and Interactive Techniques Conference. <https://doi.org/10.1145/1667239.1667243>
- Tesema, K., Hill, L., Jones, M. W., Ahmad, M. I., & Tam, G. K. L. (2023). Point Cloud Completion: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 99, 1-20. <https://doi.org/10.1109/TVCG.2023.3344935>
- TomBursch. (2019). TomBursch/Custom-Physics: Real-time physics engine using Bullet's collision detection to be used in the Godot game engine. Retrieved from <https://github.com/TomBursch/Custom-Physics>
- Trimble, S. W., & Crosson, P. (2000). U.S. soil erosion rates—myth and reality. *Science*, 289(5477), 248–250. <https://doi.org/10.1126/science.289.5477.248>
- Tucker, G. E., Hutton, E. W. H., Piper, M. D., Campforts, B., Gan, T., Barnhart, K. R., Kettner, A. J., Overeem, I., Peckham, S. D., McCreedy, L., & Syvitski, J. (2022). CSDMS: a community platform for numerical modeling of Earth surface processes. *Geoscientific Model Development*, 15(4), 1413-1439. <https://doi.org/10.5194/gmd-15-1413-2022>

Tucker, G., Lancaster, S., Gasparini, N., & Bras, R. (2001). The Channel-Hillslope Integrated Landscape Development Model (CHILD). In R. S. Harmon & W. W. Doe (Eds.), *Landscape Erosion and Evolution Modeling* (pp. 349–388). Springer US. https://doi.org/10.1007/978-1-4615-0575-4_12

Two Minute Papers. (2020, January 21). Is a Realistic Water Bubble Simulation Possible? Retrieved from <https://youtu.be/O-52enqUSNw>

Uhlenwinkel, A. (2013). Spatial thinking or thinking geographically? On the importance of avoiding maps without meaning. *GIScience*, 2013, 294-303. <https://doi.org/10.1553/GISCIENCE2013S294>

Ullmann, G. C., Guéhéneuc, Y.-G., Petrillo, F., Anquetil, N., & Politowski, C. (2023). Visualising Game Engine Subsystem Coupling. <https://doi.org/10.48550/ARXIV.2309.06329>

Unigine. (n.d.). Real-time 3D engine. Retrieved from <https://unigine.com/>

Unity. (2020, March 25). Unity Roadmap 2020: Core Engine & Creator Tools. Retrieved from <https://youtu.be/dDjsS4NPqFU>

Unity. (2023). Render pipelines introduction. Retrieved from <https://docs.unity3d.com/2023.1/Documentation/Manual/render-pipelines-overview.html>

Unity. (2024). Choose a collision detection mode. Retrieved from <https://docs.unity3d.com/Manual/choose-collision-detection-mode.html>

Unity Technologies. (2018). Baked lighting. Retrieved from <https://docs.unity3d.com/2018.4/Documentation/Manual/LightMode-Baked.html>

Unity Technologies. (2018). Real-time lighting. Retrieved from <https://docs.unity3d.com/2018.4/Documentation/Manual/LightMode-Realtime.html>

- Unity Technologies. (n.d.). Execution order of event functions. Unity Documentation. Retrieved from <https://docs.unity3d.com/Manual/execution-order.html>
- Unreal Engine. (2020). Advanced Niagara Effects | Inside Unreal. YouTube. Retrieved from <https://www.youtube.com/watch?v=31GXFW-MgQk&t=2086s>
- Unreal Engine. (2021). Niagara Overview. Epic Games. Retrieved from https://dev.epicgames.com/documentation/en-us/unreal-engine/niagara-overview?application_version=4.27
- Unreal Engine. (2022). Unreal Engine 5.0 release notes. Epic Games. Retrieved from https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5.0-release-notes?application_version=5.0
- Unreal Engine. (2024). Content Examples. Unreal Engine 5.5 Documentation. Epic Games. Retrieved from <https://dev.epicgames.com/documentation/en-us/unreal-engine/content-examples-sample-project-for-unreal-engine>
- Unreal Engine. (2024). Niagara Overview. Unreal Engine 5.5 Documentation. Epic Games. Retrieved from <https://www.epicgames.com/unrealengine/documentation/en-us/niagara-overview>.
- Unreal Engine. (2024). UNiagaraDataInterfaceNeighborGrid3D. Epic Games. Retrieved from <https://dev.epicgames.com/documentation/en-us/unreal-engine/API/Plugins/Niagara/UNiagaraDataInterfaceNeighborGrid3D>
- Unreal Engine. (2025). LiDAR Point Cloud Plugin Overview in Unreal Engine. Epic Developer Community. Retrieved from <https://dev.epicgames.com/documentation/en-us/unreal-engine/lidar-point-cloud-plugin-overview-in-unreal-engine>

- Unreal Engine. (n.d.). Collision in Unreal Engine – Overview. Epic Games. Retrieved from <https://dev.epicgames.com/documentation/en-us/unreal-engine/collision-in-unreal-engine--overview>
- Unreal Engine. (n.d.). Physics. Epic Games. Retrieved from <https://docs.unrealengine.com/5.5/en-US/physics/>
- Unreal Xinda. (2020, June 28). UE4 SPH Fluid Simulation with CUDA. Retrieved from <https://youtu.be/7ojILF1Dmro>
- Usery, E. L., Varanka, D. E., & Davis, L. R. (2018). Topographic Mapping Evolution: From Field and Photographically Collected Data to GIS Production and Linked Open Data. *The Cartographic Journal*, 55(4), 378–390. <https://doi.org/10.1080/00087041.2018.1539555>
- Valette, G., Prévost, S., Lucas, L., & Léonard, J. (2006). SoDA project: A simulation of soil surface degradation by rainfall. *Computers & Graphics*, 30(4), 494-506. doi:10.1016/j.cag.2006.03.016
- Vanderbauwhede, W. & Takemi, T. (2013). An investigation into the feasibility and benefits of GPU/multicore acceleration of the weather research and forecasting model. 2013 International Conference on High Performance Computing & Simulation (HPCS), Helsinki, Finland, pp. 482-489, doi: 10.1109/HPCSim.2013.6641457.
- Van Der Heijden, T., Hamerník, D., & Brozek, J. (2019). Use of Game Engines and VR in Industry and Modern Education. 574, 88–93. https://doi.org/10.1007/978-3-030-21507-1_13
- Van Kooten, K. (2019). Bridging Scientific Visualization and Unreal VR. *VR Developer Gems* (1st ed.). A K Peters/CRC Press. <https://www.taylorfrancis.com/chapters/edit/10.1201/b21598-9/bridging-scientific-visualization-unreal-vr-kees-van-kooten>

- Vincke, S. (2019). Immersive visualisation of construction site point cloud data, meshes and BIM Models in a VR environment using a gaming engine. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences-ISPRS Archives*, 42, 77-83.
- Vohera, C., Chheda, H., Jain, V., & Shah, A. (2021). Game Engine Architecture and Comparative Study of Different Game Engines. *International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, 2021, 9579618. <https://doi.org/10.1109/ICCCNT51525.2021.9579618>
- Vonk, G., Geertman, S., & Schot, P. (2005). Bottlenecks Blocking Widespread Usage of Planning Support Systems. *Environment and Planning A: Economy and Space*, 37(5), 909–924. <https://doi.org/10.1068/a3712>
- Voxel Plugin. (2024). Cubemap planets. Retrieved from <https://docs.voxelplugin.com/2.0p-320/basics/cubemap-planets>
- Voxel Plugin. (2024). Getting started with smooth terrains. Retrieved from <https://docs.voxelplugin.com/p2.0-304/getting-started/step-by-step-guides/getting-started-with-smooth-terrains>
- Wallace, E. (n.d.). WebGL Water. Retrieved from <http://madebyevan.com/webgl-water/>
- Wang, F., Zhou, C., Chen, Y., & Jing, X. (2021). Construction of 3D GIS based on VR and Edge Computing. *Journal of Physics: Conference Series*, 2138(1), 012027. <https://doi.org/10.1088/1742-6596/2138/1/012027>
- Water Erosion Prediction Project. (n.d.). Retrieved from <https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/water-erosion-prediction-project>

- Watson, R. A., Hornby, G. S., & Pollack, J. B. (1998). Modeling building-block interdependency. In A. E. Eiben, T. Bäck, M. Schoenauer, & H.-P. Schwefel (Eds.), *Parallel Problem Solving from Nature — PPSN V. Lecture Notes in Computer Science*, vol 1498. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/BFb0056853>
- Wiebusch, D., & Latoschik, M. E. (2015). Decoupling the Entity-Component-System Pattern using Semantic Traits for Reusable Realtime Interactive Systems. *IEEE VR Workshop on Software Engineering and Architectures for Realtime Interactive Systems*. <https://doi.org/10.1109/SEARIS.2015.7854098>
- Wilkinson, M., Dumontier, M., Aalbersberg, I. et al. The FAIR Guiding Principles for scientific data management and stewardship. *Sci Data* 3, 160018 (2016). <https://doi.org/10.1038/sdata.2016.18>
- Wilson, J. P., Butler, K., Gao, S., Hu, Y., Li, W., & Wright, D. J. (2021). A Five-Star Guide for Achieving Replicability and Reproducibility When Working with GIS Software and Algorithms. *Annals of the American Association of Geographers*, 111(5), 1311–1317. <https://doi.org/10.1080/24694452.2020.1806026>
- Wolf, Mark J. P. (2012, August 16). *Encyclopedia of Video Games: The Culture, Technology, and Art of Gaming*. Greenwood Publishing Group. pp. 3–7. ISBN 978-0-313-37936-9.
- Worthen, C. (2022). Exploring Unreal's Physics Framework. Retrieved from <https://itscai.us/blog/post/ue-physics-framework/>
- Xin, Y.-L., Ge, G.-P., Du, W., Wu, H., & Zhao, Y. (2024). Design of an Optical Physics Virtual Simulation System Based on Unreal Engine 5. *Applied Sciences*, 14(3), 955. <https://doi.org/10.3390/app14030955>
- Xu, Y., Kim, E., Lee, K., Ki, J., & Lee, B. (2014). Using PhysX Simulation Fire Model of Backdraft in Unity 3D Game Engine. *International Journal of Multimedia and Ubiquitous Engineering*, 9(6), 243–252. <https://doi.org/10.14257/ijmue.2014.9.6.24>

- Yang, X., & Delparte, D. (2022). A Procedural Modeling Approach for Ecosystem Services and Geodesign Visualization in Old Town Pocatello, Idaho. *Land*, 11(8), Article 1228. <https://doi.org/10.3390/land11081228>
- Yin, W., Hu, Q., Liu, W., Liu, J., He, P., Zhu, D., & Kornejady, A. (2024). Harnessing Game Engines and Digital Twins: Advancing Flood Education, Data Visualization, and Interactive Monitoring for Enhanced Hydrological Understanding. *Water (Switzerland)*, 16(17), Article 2528. <https://doi.org/10.3390/w16172528>
- Yuan, M. (2001). Temporal GIS and Spatio-Temporal Modeling. *Proceedings of the International Conference on Advances in Spatial Data Handling*, 334–348. https://doi.org/10.1007/978-3-642-56750-6_25
- Yudin, P. V., Fedina, M. I., Strizh, E. V., Yudina, O. V., & V, K. (2019). Computational Hydrodynamics in Air Flows Modeling: Using the Unreal Engine based on the numerical solution of the Navier-Stokes equations. *2019 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)*, 1–9. *2019 International Multi-Conference on Industrial Engineering and Modern Technologies (FarEastCon)*. <https://doi.org/10.1109/FarEastCon.2019.8934325>
- Yu, L.-J., Sun ,Dan-Feng, Peng ,Zhong-Ren, & and Zhang, J. (2012). A Hybrid System of Expanding 2D GIS into 3D Space. *Cartography and Geographic Information Science*, 39(3), 140–153. <https://doi.org/10.1559/15230406393140>
- Zaragozí, B. M., Trilles, S., & Navarro-Carrión, J. T. (2020). Leveraging Container Technologies in a GIScience Project: A Perspective from Open Reproducible Research. *ISPRS International Journal of Geo-Information*, 9(3), 138. <https://doi.org/10.3390/ijgi9030138>

- Zellner, M. L. (2008). Embracing complexity and uncertainty: The potential of agent-based modeling for environmental planning and policy. *Planning Theory & Practice*, 9(4), 437-457. <https://doi.org/10.1080/14649350802481470>
- Zellner, M., Watkins, C., Massey, D., Westphal, L., Brooks, J., & Ross, K. (2014). Advancing collective decision-making theory with integrated agent-based modeling and ethnographic data analysis: An example in ecological restoration. *Journal of Artificial Societies and Social Simulation*, 17(4), Article 11. <https://doi.org/10.18564/jasss.2605>
- Ziegler, P., & Chasins, S. E. (2023). A Need-Finding Study with Users of Geospatial Data. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*, April 23–28, 2023, Hamburg, Germany. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3544548.3581370>
- Zhang, Zhou, C., Zhang, J., & Chen, X. (2016). A ubiquitous knowledgeable data representation model (UKRM) for three-dimensional geographic information systems (3D GIS). *Science China. Earth Sciences*, 59(4), 780–794. <https://doi.org/10.1007/s11430-015-5255-7>
- Zhao, T., Song, C., Yu, J., Xing, L., Xu, F., Li, W., & Wang, Z. (2025). Leveraging Immersive Digital Twins and AI-Driven Decision Support Systems for Sustainable Water Reserves Management: A Conceptual Framework. *Sustainability*, 17(8), Article 8. <https://doi.org/10.3390/su17083754>
- Zhou, J., Chen, D., Deng, M., Deng, Y., Sun, Y., Wang, S., Xiong, S., & Zhu, B. (2024). Eulerian-Lagrangian Fluid Simulation on Particle Flow Maps. *ArXiv*. <https://doi.org/10.48550/arXiv.2405.09672>
- Zlatanova, S., Rahman, A. A., & Pilouk, M. (2011). 3D GIS: Current status and perspectives. *Geoinformatics*, October 2011. <https://www.researchgate.net/publication/228779275>

Zukowski, C. (2025, April 12). Niagara Grid 2D feels like a superpower: Drawing locations to a render target in Unreal 5.1. Zuko Media. Retrieved from <https://www.zukomedia.com/articles/niagara-grid-2d-feels-like-a-superpower-drawing-locations-to-a-render-target-in-unreal-5-1>

Appendix A

Additional Literature Review

A.1 Research on Modifiable Temporal Unit Problem

The search query was on Scopus and formulated as:

(TITLE("Modifiable Temporal Unit Problem") OR
ABS("Modifiable Temporal Unit Problem") OR
KEY("Modifiable Temporal Unit Problem"))

The search yielded 16 results, of which only 3 had MTUP in their title. de Jong & de Bruin (2012) aggregated time-series data of vegetation indices into different temporal units and found that analysis deviations experienced significant changes due to the effect of MTUP. Similarly, Cheng & Adepeju (2014) explored the MTUP by applying space-time cluster detection techniques to crime data, demonstrating that changes in temporal aggregation can lead to different statistical significance, and that there is a need for temporal calibration in working with temporal data. They put forth a compelling argument highlighting the importance of incorporating time information into geospatial data analysis. By delving into the temporal dimension, researchers can unlock a plethora of valuable insights and develop a deeper contextual understanding of the relationships between spatial events and time. The authors also highlight that temporal dimensions share similarities with spatial phenomena, such as segmentation, boundary, and scale, thereby influencing the statistical estimates obtained from temporal data aggregations.

Deckard & Schnell (2022) further extended this inquiry by analyzing crime at a monthly resolution, discovering that annual aggregations may provide misleading results on when and where hotspots occur, such as displaying seasonal consistency when there may be none.

Thus, the Modifiable Spatio-Temporal Unit Problem (MSTUP) represents the logical next step in improving our understanding of the complex relationship between spatial and temporal dynamics.

Appendix B

Additional Tables and Figures

B.1 Comparison of Point Cloud Processing Tools

Software	Pricing	Native-Graphic User Interface
Cloud Compare	Free (open-source)	Yes
LAStools	Free (non-commercial), Contact vendor (commercial)	No
PDAL	Free (open-source)	No
Leica Cyclone	Contact vendor (commercial)	Yes
Pix4D	Contact vendor (commercial)	Yes

B.2 Unreal Engine Materials

Table 15. Modified parameters in built-in modules

Module	Property Name	Property Type	Property Value
Emitter Update Stage			
Emitter State	Life Cycle Mode	Enum	System
Spawn Particles in Grid	XYZ Count	Int32	(20, 20, 0)
Set Parameter	FluidPAR	Particle Attribute Reader	Read from Emitter Name (Main_Particle_Emitter)
Particle Spawn Stage			
Initialize Particle	Lifetime Mode	Enum	Direct Set
	Lifetime	Float	15.0 seconds
	Mass Mode	Enum	Unset

	Sprite Size Mode	Enum	Uniform
	Uniform Sprite Size	Float	25.0 cm
Grid Location	Grid Origin	Position	Linked to <Engine><Emitter> SimulationPosition
	Coordinate Space	Enum	Simulation
	Normalize Offsets	Boolean	False
	Randomize Placement within Cell	Vector3	(90.0, 90.0, 0.0)
	Dimensions Definition	Enum	Bounding Box Size
	XYZ Dimensions	Vector3	(1920.0, 1920.0, 0.0)
Particle Update Stage			
Particle State	Kill Particles When Lifetime Has Elapsed	Boolean	True
Gravity Force	Gravity	Float	(0.0, 0.0, -9.81)
Collision	GPU Collision Type	Enum	GPU Ray Traces
	Trace Provider	Enum	HW Ray Tracing
	Radius Calculation Type	Enum	Mesh
	Particle Radius Scale	Float	1.5
	Restitution	Float	0.0
	Simple Friction	Boolean	False
	Friction	Float	0.01
	Static Friction	Float	0.01
	Static Friction Engagement Speed	Float	0.01
	Maximum Penetration Correction Distance	Float	1.0

	Percentage of Penetration Before Rest	Float	0.7
	Rest State Time Range	Float	1.0
	Valid Collision Plane Normal for Rest	Vector3	(0.0, 0.0, 1.0)
	Minimum Collision Normal Alignment Percentage	Float	0.5
Kill Particles	Kill Particles	Boolean	Linked to <Transient> CollisionIsResting
Solve Forces and Velocity	N/A	N/A	N/A
Export Particle Data to Blueprint	Condition to Export Data	Bool	Custom Dynamic Input. On particle collision
	Vector to Send	Position	Linked to <Particles> LastPosition
	Vector to Send	Velocity	Linked to <Particles> Velocity
	Float to Send	Float	Linked to <Particles> ID (Converted Integer to Float)
	Export Particle Data Interface	Export Particle Data	Callback Handler <User> NiagaraCallbackHandler

**Digital
Elevation
Model of
Milverton
Farm Field**

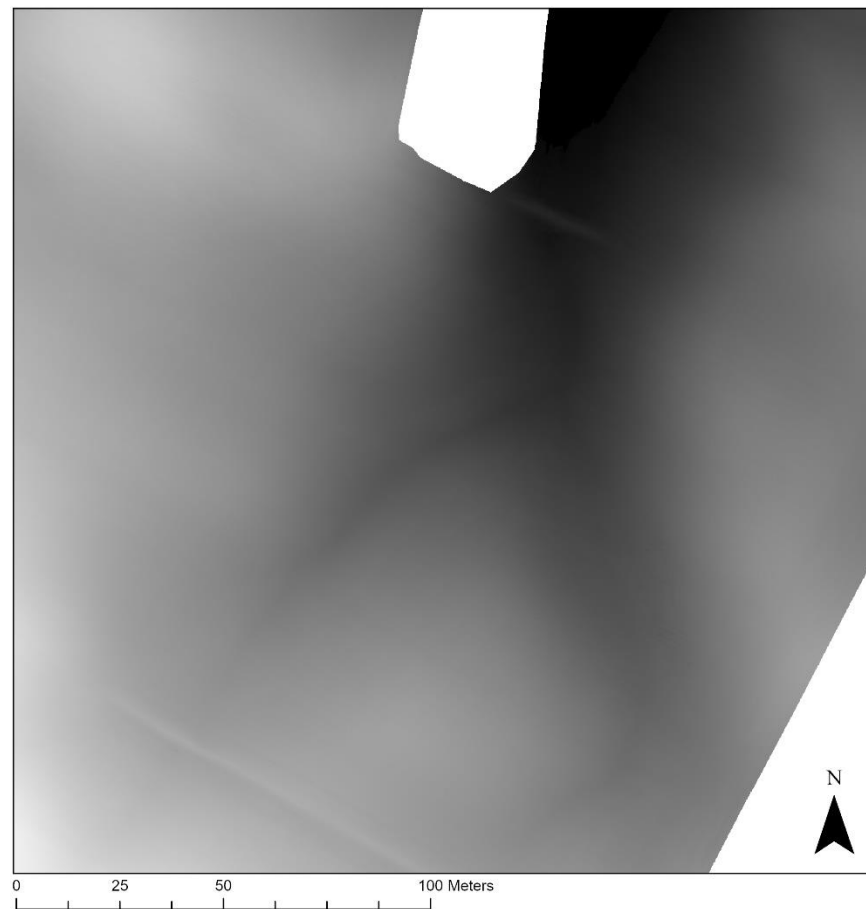
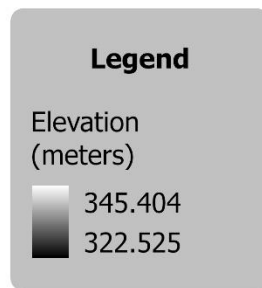


Figure 8. The DEM of the cropped study area

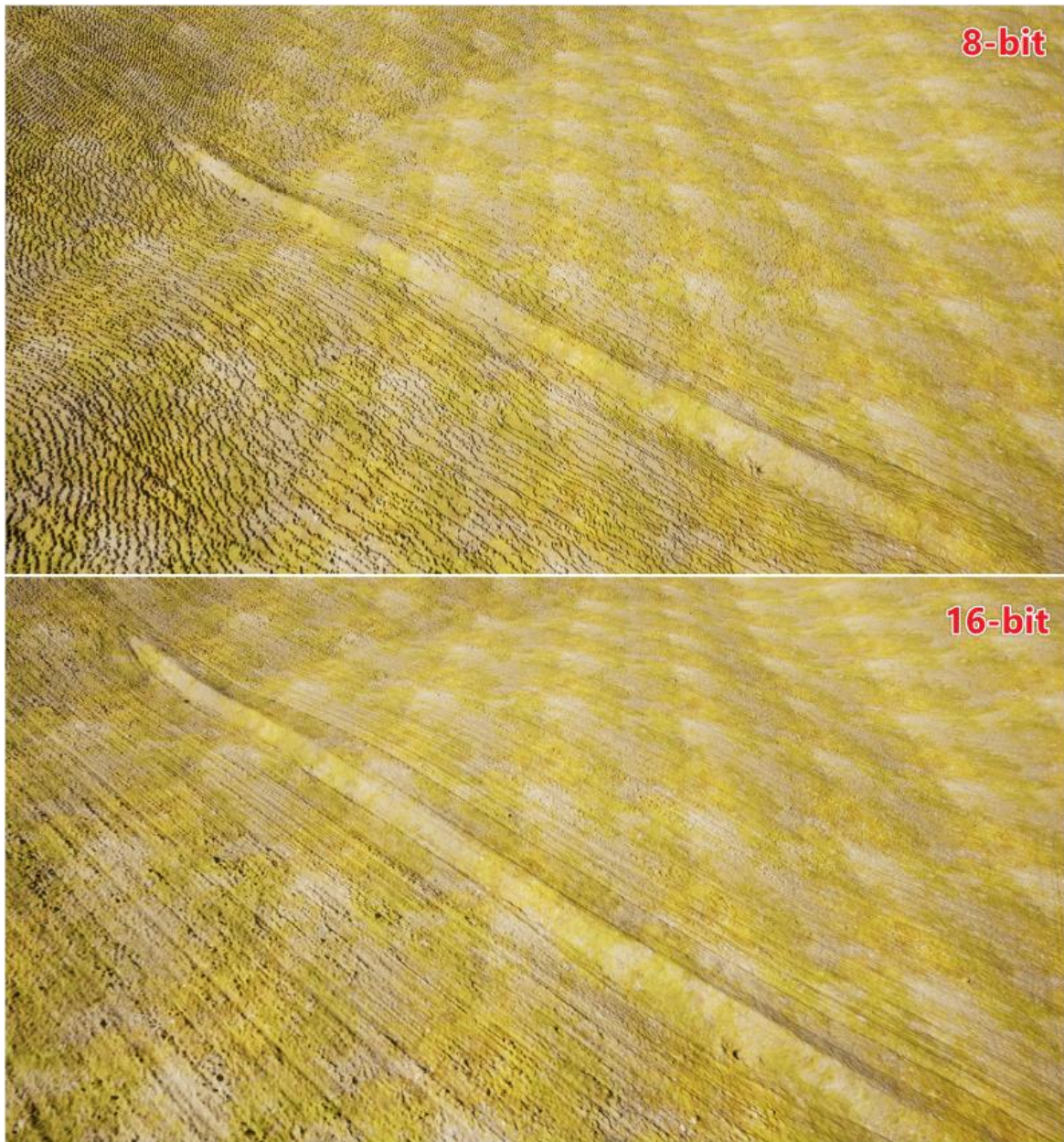


Figure 9. Stair-case artifacts are exposed in lit shading conditions to show the loss of precise elevation data in lower bit depths

**Strahler
Stream Order
Map From
D8 Flow
Direction**

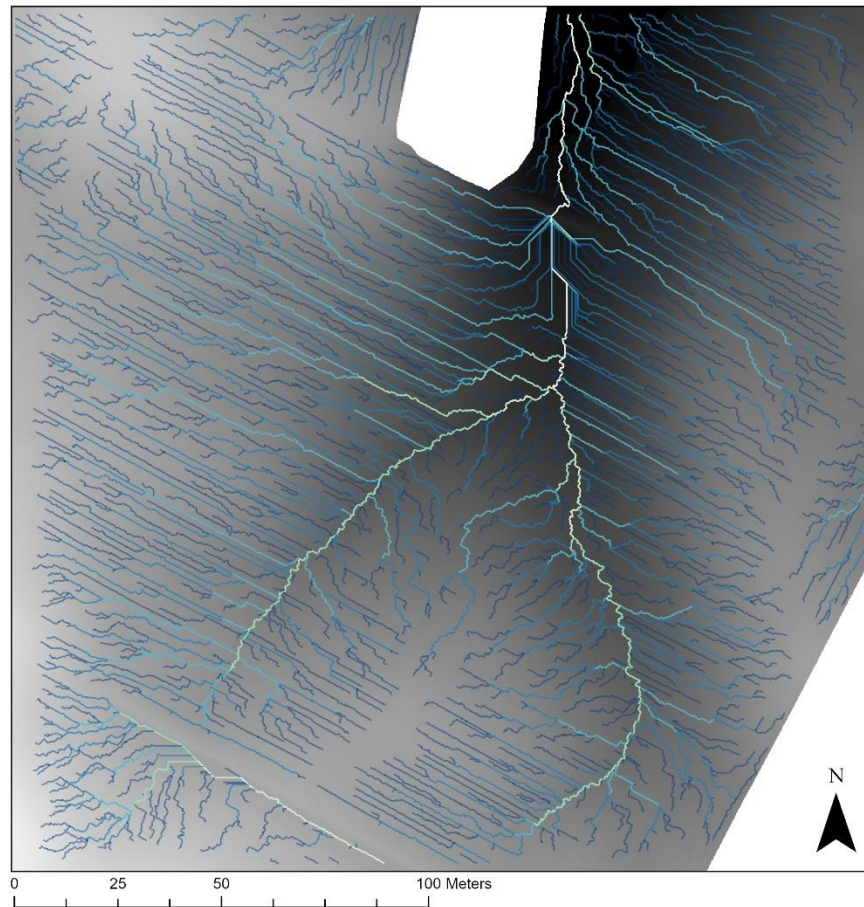


Figure 10. D8 flow paths visualized with stream orders overlaid on DEM

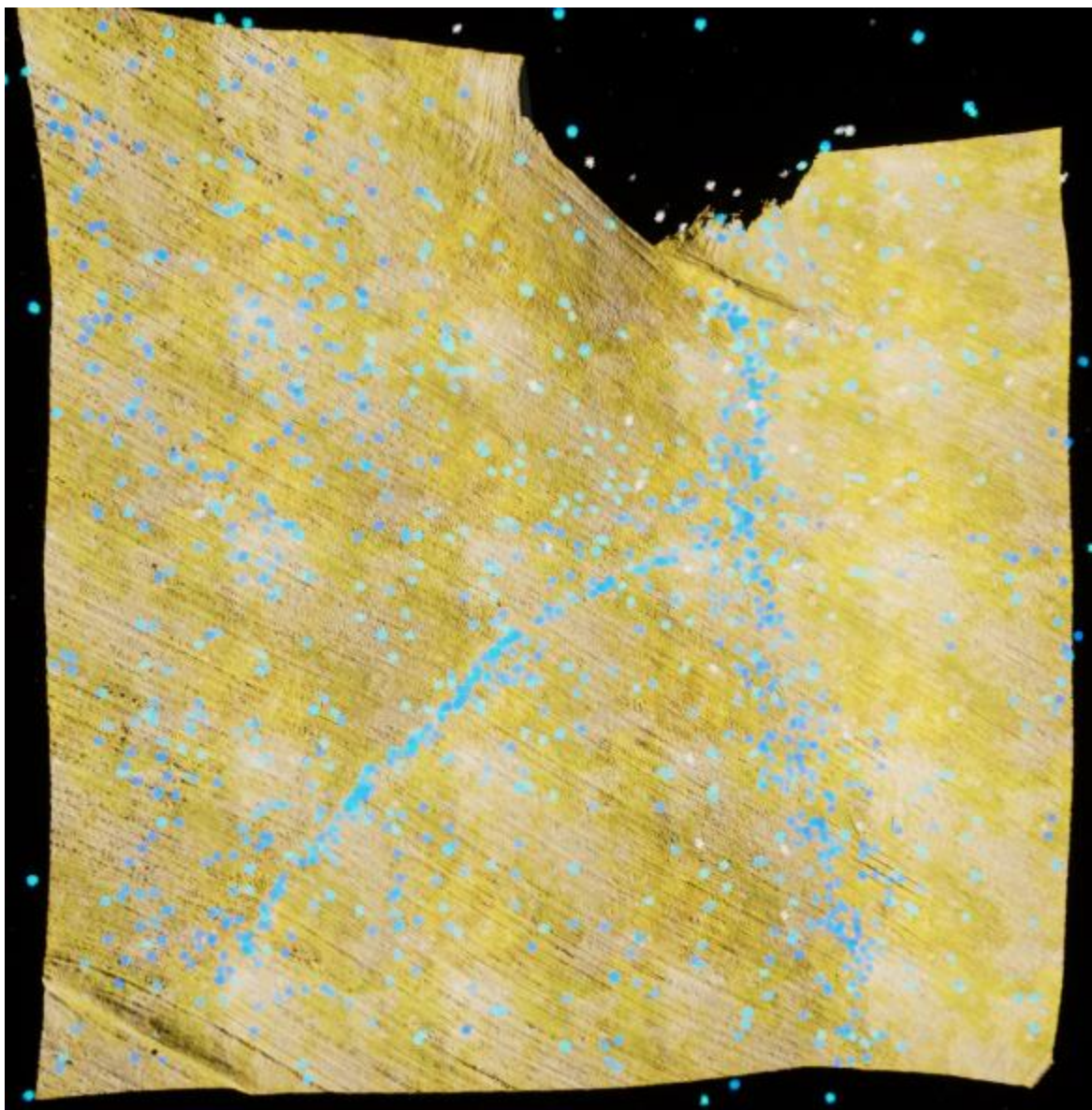


Figure 11. A simplified color scheme and perspective to show slow moving particles as darker blue and faster as brighter shades of blue

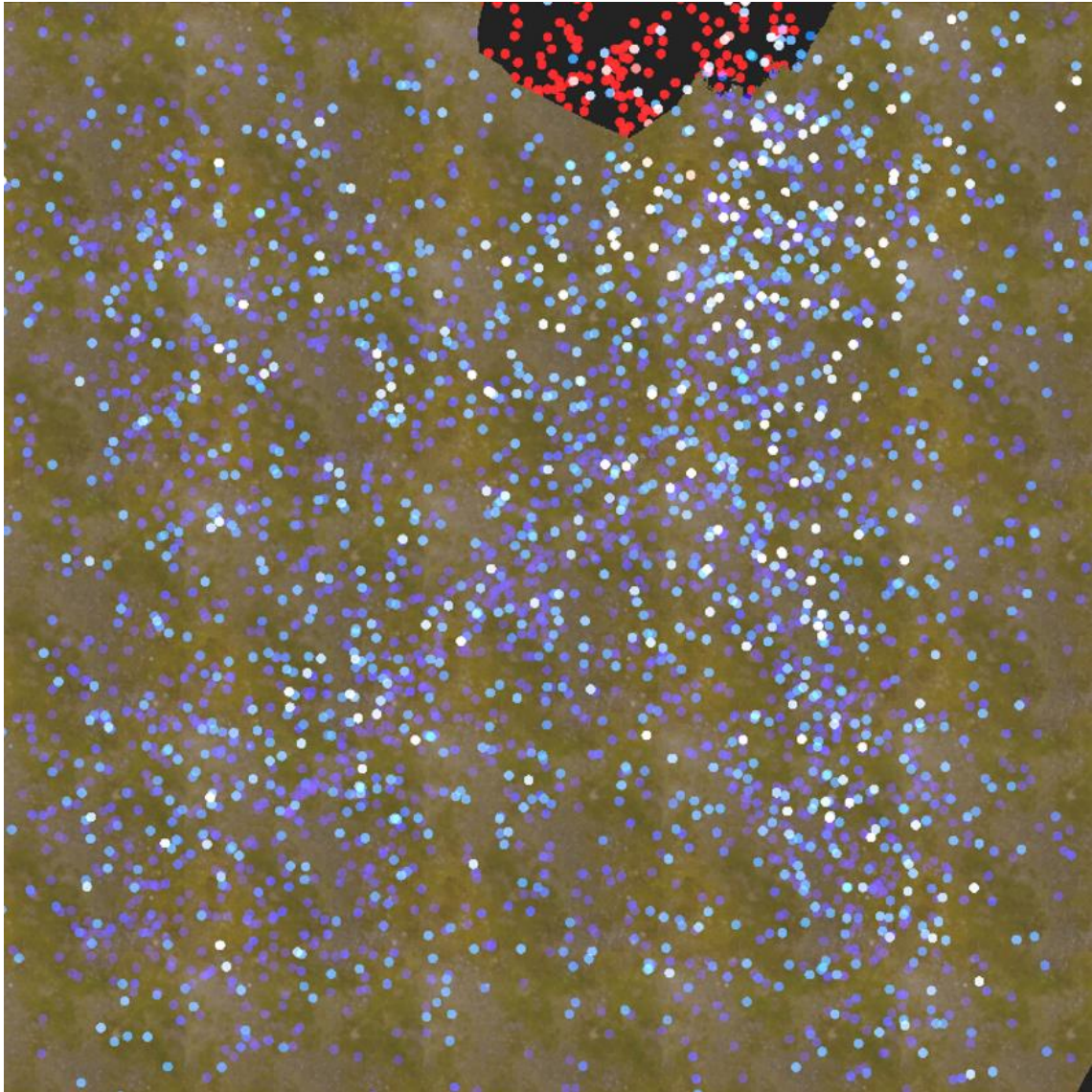


Figure 12. A top-down orthographic perspective of the model during simulation in an unlit environment.

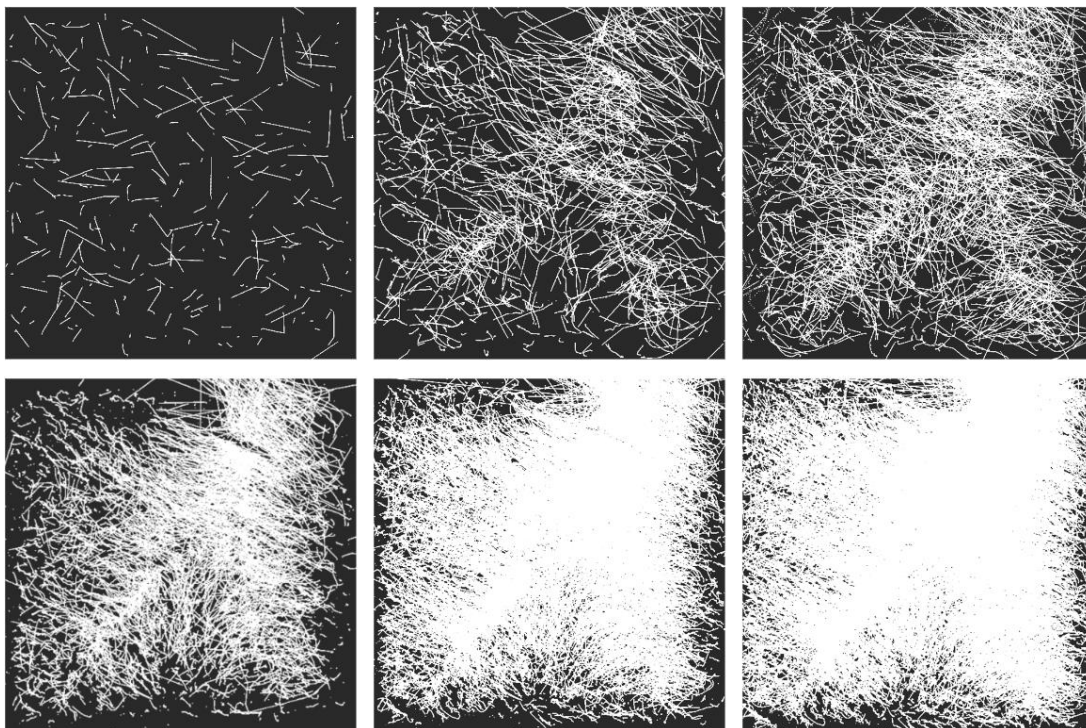


Figure 13. The evolution of particles tracing their travel paths onto a render target

**Average
Widths of
Simulated
Flow Paths**

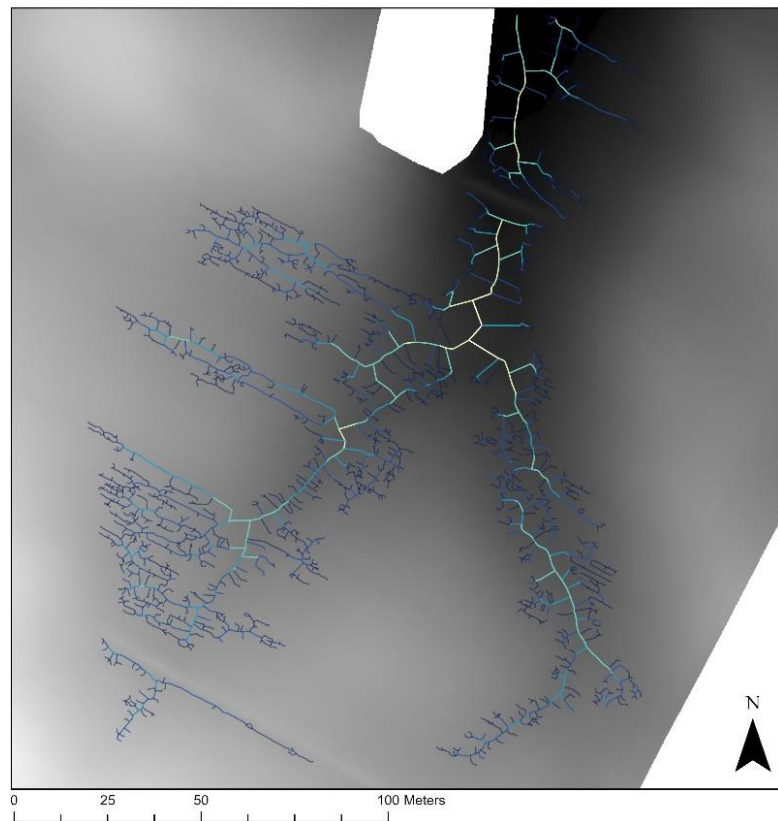


Figure 14. Collapsed flow paths produced from UE’s simulated output. The lines with average width of above 10 meters are depicted as brighter paths.

Appendix C

Hardware/Software Specifications

System	Operating System	Microsoft Windows 11 Home
	CPU Processor	11th Gen Intel(R) Core(TM) i9-11900K @ 3.50GHz, 3504 Mhz, 8 Core(s), 16 Logical Processor(s)
	Total Physical Memory	15.9 GB
	Available Physical Memory	8.36 GB
	Total Virtual Memory	29.9 GB
	Available Virtual Memory	16.8 GB
Graphics	Display Adapter (GPU)	NVIDIA GeForce RTX 3060 (12GB GDDR6) x2 Fans
	GPU Driver Version	32.0.15.5612
	Resolution	1920 x 1080 x 60 Hz
	Bits/Pixel	32

Storage	File System	NTFS
	Solid State Drive Size	465.00 GB
Unreal Engine	Version	5.1.1
	Plugins	Georeferencing (Template) Niagara Fluids (Built-in) Voxel Plugin 1.2 Legacy (Paid)

Appendix D

High-Level Shading Language Code

D.1 Populate Neighbor Grids

```

OutPosition = Position;

#if GPU_SIMULATION

float3 UnitPos;
NeighborGrid.SimulationToUnit(Position, SimulationToUnit, UnitPos);

int3 Index;
NeighborGrid.UnitToIndex(UnitPos, Index.x, Index.y, Index.z);

int3 NumCells;
NeighborGrid.GetNumCells(NumCells.x, NumCells.y, NumCells.z);

int MaxNeighborsPerCell;
NeighborGrid.MaxNeighborsPerCell(MaxNeighborsPerCell);

```

```

if (Index.x >= 0 && Index.x < NumCells.x &&
    Index.y >= 0 && Index.y < NumCells.y &&
    Index.z >= 0 && Index.z < NumCells.z)
{
    int LinearIndex;
    NeighborGrid.IndexToLinear(Index.x, Index.y, Index.z, LinearIndex);

    int PreviousNeighborCount;

    NeighborGrid.SetParticleNeighborCount(LinearIndex, 1,
        PreviousNeighborCount);
    if (PreviousNeighborCount < MaxNeighborsPerCell)
    {
        int NeighborGridLinear;

        NeighborGrid.NeighborGridIndexToLinear(Index.x, Index.y, Index.z,
            PreviousNeighborCount, NeighborGridLinear);

        int IGNORE;

        NeighborGrid.SetParticleNeighbor(NeighborGridLinear,
            InstanceIdx, IGNORE);
    }
}

#endif

```

D.2 Intra-particle Collision

```

OutPosition = Position;
OutVelocity = Velocity;
CollidesOut = CollidesIn;
NumberOfNeighbors = 0;
displayCount = 0.0;
//AveragedCenterLocation = float3 (0.0,0.0,0.0);

#if GPU_SIMULATION

const int3 IndexOffsets [ 27 ] = {

```

```

int3(-1,-1,-1),
int3(-1,-1, 0),
int3(-1,-1, 1),
int3(-1, 0,-1),
int3(-1, 0, 0),
int3(-1, 0, 1),
int3(-1, 1,-1),
int3(-1, 1, 0),
int3(-1, 1, 1),

int3(0,-1,-1),
int3(0,-1, 0),
int3(0,-1, 1),
int3(0, 0,-1),
int3(0, 0, 0),
int3(0, 0, 1),
int3(0, 1,-1),
int3(0, 1, 0),
int3(0, 1, 1),

int3(1,-1,-1),
int3(1,-1, 0),
int3(1,-1, 1),
int3(1, 0,-1),
int3(1, 0, 0),
int3(1, 0, 1),
int3(1, 1,-1),
int3(1, 1, 0),
int3(1, 1, 1),

};

float3 UnitPos;
myNeighborGrid.SimulationToUnit(Position, SimulationToUnit, UnitPos);

int3 Index;
myNeighborGrid.UnitToIndex(UnitPos, Index.x,Index.y,Index.z);

```

```

float3 FinalOffsetVector = {0,0,0};
uint ConstraintCount = 0;
float TotalMassPercentage = 1.0;
int3 NumCells;
myNeighborGrid.GetNumCells(NumCells.x, NumCells.y, NumCells.z);

int MaxNeighborsPerCell;
myNeighborGrid.MaxNeighborsPerCell(MaxNeighborsPerCell);

for (int xxx = 0; xxx < 27; ++xxx) {
    for (int i = 0; i < MaxNeighborsPerCell; ++i) {
        const int3 IndexToUse = Index + IndexOffsets[xxx];
        int NeighborLinearIndex;

        myNeighborGrid.NeighborGridIndexToLinear(IndexToUse.x, IndexToUse.y,
                                                    IndexToUse.z,
                                                    i,
NeighborLinearIndex);
        int CurrNeighborIdx;
        myNeighborGrid.GetParticleNeighbor(NeighborLinearIndex,
CurrNeighborIdx);
        bool myBool;
        float3 OtherPos;

        DirectReads.GetVectorByIndex<Attribute="Position">(CurrNeighborIdx,
myBool, OtherPos);
        const float3 vectorFromOtherToSelf = Position - OtherPos;
        const float dist = length(vectorFromOtherToSelf);
        const float3 CollisionNormal = vectorFromOtherToSelf / dist;

        float OtherRadius;
        DirectReads.GetFloatByIndex<Attribute="CollisionRadius">
            (CurrNeighborIdx, myBool, OtherRadius);

        float Overlap = (CollisionRadius + OtherRadius) - dist;
        if (IndexToUse.x >= 0 && IndexToUse.x < NumCells.x &&
            IndexToUse.y >= 0 && IndexToUse.y < NumCells.y &&
            IndexToUse.z >= 0 && IndexToUse.z < NumCells.z &&
            CurrNeighborIdx != InstanceIdx && CurrNeighborIdx != -1
                && dist > 1e-5)

```

```

{

    bool otherUnyielding = false;
    TotalMassPercentage = 1.0;

    if ( Overlap > 1e-5)
    {
        NumberOfNeighbors+=1;
        displayCount = NumberOfNeighbors;
        bool NeighborUnyieldResults;
        DirectReads.GetBoolByIndex<Attribute="Unyielding">
            (CurrNeighborIdx, myBool, NeighborUnyieldResults);

        CollidesOut = true;

        float OtherMass;
        DirectReads.GetFloatByIndex<Attribute="Mass">
            (CurrNeighborIdx, myBool, OtherMass);

        //1 moves this particle, 0 moves the other, .5 moves both

        TotalMassPercentage = Mass / (Mass + OtherMass);

        if ( NeighborUnyieldResults && Unyielding ){ // both this
                                                    particle and the other are unyielding
            TotalMassPercentage = TotalMassPercentage;
        }
        else if ( NeighborUnyieldResults ) { // this particle yeilds
                                                    but the other does
not
            TotalMassPercentage = lerp ( TotalMassPercentage, 0.0,
UnyieldingMassPercentage);
        //0
        }
        else if (Unyielding) { // this particle is unyielding but
The other is
            TotalMassPercentage = lerp ( TotalMassPercentage, 1.0,
UnyieldingMassPercentage);
        //1
        }
    }
}

```

```

        FinalOffsetVector += (1.0 - TotalMassPercentage) * Overlap
*
        CollisionNormal;
        float3 OtherPreviousPos;
DirectReads.GetVectorByIndex<Attribute="Previous.Position">
        (CurrNeighborIdx, myBool,
OtherPreviousPos);
        const float3 P0Velocity = Position - PreviousPosition;
        const float3 PNVelocity = OtherPos - OtherPreviousPos;
        const float3 P0RelativeMomentum = P0Velocity*Mass -
                                           PNVelocity*OtherMass;
        const float3 NormalBounceVelocity = (P0RelativeMomentum -
dot(P0RelativeMomentum, CollisionNormal)
*
        CollisionNormal)/Mass;
        const float NormalBounceVelocityMag =
        length(NormalBounceVelocity);
        ConstraintCount += 1;
    }
}
}
}

if (ConstraintCount > 0 && IsMobile) { OutPosition += 1.*FinalOffsetVector/
        (ConstraintCount * RelaxationAmount);
    OutVelocity = (OutPosition - PreviousPosition) * InvDt;
    if (Unyielding) {
        OutVelocity = float3 (0,0,0);
    }
}

#endif // GPU_SIMULATION

```

D.3 Accumulate Forces

```
#pragma once
OutDensity = 0.0; OutPressure = 0.0; OutForce = float3(0.0, 0.0, 0.0);

#if GPUSIMULATION
bool Valid;
float PI = 3.14159265358979323846264338327950288;
float EPSILON = 0.0000001;
float KernelH2 = KernelH*KernelH;

float KernelH9 = KernelHKernelHKernelHKernelHKernelHKernelHKernelH*KernelH;
float Poly6Constant = 315.0 / (64.0 * (PI * KernelH9));

struct Functions {

    float PolySix(float r) {
if (r > KernelH) {
return 0.0;
} else {
float x = KernelH * KernelH - r * r;
return Poly6Constant * x * x * x;
}
}

    float3 SpikyGradient(float3 direction) {
float r = length(direction); float h minus r = KernelH - r;
return h minus r * h minus r * direction / r;
}

float ViscosityLaplacian(float r, float h) { return h - r; }

float ComputeDensity() {
float density = 0.0;

for(int i=0; i<N; i++){
if(i==ID)continue;

float3 neighborPosition;
```

```

        AttributeReader.GetVectorByIndex<Attribute="Position">(i, Valid,
neighborPosition);
        float3 direction = Position - neighborPosition;

        float distance = length(direction);
        if(distance >= KernelH) continue;

        density += PolySix(distance);
    }
    return density * Mass;
}

float ComputePressure(float density) { return max(PressureScale * (density -
TargetDensity), 0.0); }

float3 ComputePressureForceAndViscosityForce(float density, float pressure) {
float3 force = float3(0.0, 0.0, 0.0);

for(int i=0; i<N; i++)
{
    if(i==ID) continue;

    float3 neighborPosition;
    AttributeReader.GetVectorByIndex<Attribute="Position">(i, Valid,
neighborPosition);

    float3 direction = Position - neighborPosition;
    float distance = length(direction);
    if(distance >= KernelH) continue;

    float neighborDensity;
    AttributeReader.GetFloatByIndex<Attribute="_Density">(i, Valid,
neighborDensity);
    float neighborPressure;
    AttributeReader.GetFloatByIndex<Attribute="_Pressure">(i, Valid,
neighborPressure);
    float3 neighborVelocity;

```

```

        AttributeReader.GetVectorByIndex<Attribute="Velocity">(i, Valid,
neighborVelocity);

        if(neighborDensity > EPSILON && density > EPSILON)
        {
            float3 pressureForce = PressureForceScale * ((pressure +
neighborPressure) / 2.0) * SpikyGradient(direction);
            force += pressureForce;
        }

        // Compute viscosity force by laplacian of viscosity kernel
        if (neighborDensity > EPSILON)
        {
            float3 viscosityForce = ViscosityForceScale * (neighborVelocity -
Velocity) / neighborDensity * ViscosityLaplacian(distance, KernelH);
            force += viscosityForce;
        }
    }
    return force;
}

}; Functions functions;

OutDensity = functions.ComputeDensity();
OutPressure = functions.ComputePressure(OutDensity);
OutForce = functions.ComputePressureForceAndViscosityForce(OutDensity,
OutPressure);

#endif

```

D.4 Accumulate External Forces

```

#pragma once
OutForce = Force;

#if GPUSIMULATION

```

```

if(EnableWall)
{
float3 avoidance = float3(0, 0, 0);
avoidance.x = (Position.x < -FieldScale) ? avoidance.x + 1.0 : avoidance.x;
avoidance.x = (Position.x > FieldScale) ? avoidance.x - 1.0 : avoidance.x;
avoidance.y = (Position.y < -FieldScale) ? avoidance.y + 1.0 : avoidance.y;
avoidance.y = (Position.y > FieldScale) ? avoidance.y - 1.0 : avoidance.y;
avoidance.z = (Position.z < -FieldScale) ? avoidance.z + 1.0 : avoidance.z;
avoidance.z = (Position.z > FieldScale) ? avoidance.z - 1.0 : avoidance.z;

OutForce += avoidance * WallBounce;
}

if(EnableGravity) OutForce += Gravity;

#endif // GPU_SIMULATION

```

D.5 Update Position

```

#pragma once

OutVelocity = float3(0.0, 0.0, 0.0);
OutPosition = float3(0.0, 0.0, 0.0);

#if GPU_SIMULATION

if(length(Force) > MaxForce)
{
    Force = normalize(Force) * MaxForce;
}

Velocity += Force * DeltaTime;

if(length(Velocity) > MaxVelocity)
{
    Velocity = normalize(Velocity) * MaxVelocity;
}

Position += Velocity * DeltaTime;

```

```
OutPosition = Position;  
OutVelocity = Velocity;  
  
#endif // GPU_SIMULATION
```

D.6 Mark Grid Location

```
#if GPU_SIMULATION  
  
float4 NewGridValue = 0;  
  
const float Distance = length(InGridLocation.xy - InValue.xy);  
if (Distance <= 2) {  
    NewGridValue = 1; // lerp(InRGBA, InRGBA + 1, InDeltaTime * 5);  
}  
  
OutRGBA = InRGBA + NewGridValue;  
#endif
```

Appendix E

Jupyter Notebook Code Blocks in R

```
ref <- rast("D8_DifferenceMR.png")[[1]] # Reference
comp <- rast("Unreal_DifferenceMR.png") # Comparison

# Ensure comparison is a single-band raster
comp <- comp[[1]]
dim(comp)

# Square extent for both rasters
max_size <- max(nrow(ref), ncol(ref))

# Square cropping extent
crop_extent <- ext(1, max_size, 1, max_size)

# Crop both rasters to ensure they match in size and extent
ref <- crop(ref, crop_extent)
comp <- crop(comp, crop_extent)

dim(ref)
dim(comp)

plot(ref, main = "Reference map")
plot(comp, main = "Comparison map")

# Convert pixel values to categorical (0 = absence, 255 = presence)
ref <- classify(ref, cbind(255, 1)) # Convert 255 to category 1
comp <- classify(comp, cbind(255, 1))

# Check both rasters have the same categorical levels
levels(ref) <- list(data.frame(ID = c(0, 1), Category = c("Absence",
"Presence")))
levels(comp) <- list(data.frame(ID = c(0, 1), Category = c("Absence",
"Presence")))

# SpatRaster to RasterLayer
ref_raster <- raster(ref)
```

```
comp_raster <- raster(comp)

# Align extents of rasters
ref_extent <- extent(ref_raster)
comp_extent <- extent(comp_raster)

# Adjust both to the same extent, make smaller one bigger if not identical
if (!identical(ref_extent, comp_extent)) {
  comp_raster <- extend(comp_raster, ref_extent)
}

# Run differenceMR function
diff_results <- differenceMR(comp_raster, ref_raster, eval = "multiple", fact
= 2)

print(diff_results)
```

Appendix F

Python Scripts for Vector Analysis

F.1 Absolute Length Deviation

```
def absolute_length_deviation(length1, length2):  
    """Calculate the absolute deviation between two double fields"""  
    if length1 is None or length2 is None:  
        return None  
  
    return abs(length1 - length2) # Direct subtraction, maintain double  
precision
```

F.2 Calculate Direction Deviation

```
def absolute_direction_deviation(direction1, direction2):  
    """Calculate the absolute deviation between two angles (0-360)"""  
    if direction1 is None or direction2 is None:  
        return None  
  
    diff = abs(direction1 - direction2)  
    return min(diff, 360 - diff) # Ensure circular distance is accounted for
```

Glossary

Actor

A term commonly used in Unreal Engine to represent a fundamental object in a game engine that can be placed in a scene. Actors can include components like meshes, lights, or particle systems, making them versatile building blocks for simulations.

Agent-Based Model (ABM)

Computational models simulating the actions and interactions of autonomous agents (e.g., particles or wildlife) to study complex systems, such as ecosystems or disease spread.

Asset Store Plugin

A pre-built component or extension from a game engine's marketplace, adding features like terrain generation or weather effects to streamline modeling tasks, and is developed by the community.

Computational Fluid Dynamics (CFD)

Numerical methods for analyzing fluid flows, used to simulate phenomena like airflow, water currents, or pollutant dispersion in simulation models.

Central Processing Unit (CPU)

The main processor in a computer, handling general computations in sequential order, often complemented by GPUs for specialized tasks.

D8 Algorithm

A method to determine flow direction on a Digital Elevation Model (DEM) by assigning flow to one of eight neighboring cells based on the steepest slope, widely used in hydrological modeling.

Digital Elevation Model (DEM)

A 3D grid of elevation values representing terrain surfaces, essential for simulating landscapes and geospatial phenomena.

Entity-Component-System (ECS)

A design pattern in game engines separating entities (objects), components (data), and systems (behavior) to manage complex simulations efficiently.

Emitter

A particle system component that generates and controls particles, defining their creation and behavior for simulating effects like fire or water.

Eulerian View

A grid approach focusing on fixed coordinates in space to study data stored and passing through it, often used in cell-based simulations.

Filmbox (FBX)

A common and robust 3D file format widely used to represent mesh and texture data in game engines and modelling tools, as well as supporting animation pipelines.

Graphics Library Transmission Format (glTF)

A lightweight, JSON-driven 3D asset format optimized for web-based real-time rendering and sharing online. Is recognized as a standardized 3D data format and used in various modern production pipelines.

Wavefront Object (OBJ)

A basic and open plain text data format that stores static 3D geometry data favored for simplicity and compatibility across platforms. Lacks in-depth metadata compared to FBX but is easier to interpret.

OpenEXR (EXR)

A high dynamic range (HDR) image format used in UE for storing visuals with a wide brightness range, useful for realistic rendering in simulations.

Game Engine

A software framework originally for video games, characterized for its graphics, physics, and interactivity in simulations and visualizations.

Geocomputation

The use of computational methods to solve geographic problems, analyzing spatial data with algorithms for complex modeling tasks.

Geovisualization

Visual representations (e.g., maps, 3D models) to explore and communicate geographic data, aiding in understanding simulation results.

GNU Image Manipulation Program (GIMP)

A free, open-source software for editing textures, maps, or visual assets.

GIS (Geographic Information System)

A general system for capturing, storing, analyzing, and displaying spatial data, typically referring to software such as ArcGIS or QGIS.

GIScience (Geographic Information Science)

The study of principles behind GIS technology, providing a theoretical basis for spatial data representation and analysis.

GISystems (Geographic Information Systems)

The comprehensive framework of softwares and hardware tools implementing GIS, enabling spatial data management and analysis in structured, automated pipelines and workflows.

Graphics Processing Unit (GPU)

A specialized processor for graphics and parallel computations, accelerating simulations with large datasets or complex calculations.

High Dynamic Range (HDR)

Imaging techniques capturing a wide luminosity range, enhancing realism in visualizations of natural environments.

High-Level Shading Language (HLSL)

A language for writing shaders in DirectX, often used in game engines for custom visual effects in simulations.

Integrated Development Environment

A comprehensive workspace where programmers can write, debug, and run code in one place.

Lagrangian View

A perspective for tracking individual data objects as they move, commonly used for particle-based simulations like SPH.

Laser (LAS)

A file format for LiDAR point cloud data, using a collection of 3D points in space to represent detailed representations of structures and surfaces in reality.

Level of Detail (LOD)

A graphics technique reducing 3D model complexity based on distance, optimizing computational resources in large-scale virtual environments.

Marching Cubes

An algorithm generating 3D meshes from voxel data, used to create smooth surfaces in terrain or medical imaging.

Mesh

A 3D object defined by vertices, edges, and faces, representing visual aspects of objects in virtual environments.

Navier-Stokes Equations (NSE)

A set of partial differential equations describing the conservation of momentum and mass in fluids, forming the theoretical basis for simulating realistic fluid behaviors.

Niagara Simulation Framework (NSF)

A visual effects system in Unreal Engine for complex particle simulations, such as fluid, fire or smoke.

NVIDIA

A company producing GPUs with parallel processing capabilities, widely used to speed up simulations and rendering.

Parallel Computation

Simultaneous processing by multiple units, enhancing the speed of large-scale or high-resolution computations.

Particle Attribute Reader (PAR)

A component in Niagara Simulation Framework for accessing particle data (e.g., velocity), aiding in data access and interoperability.

Particle System

A graphics technique simulating fuzzy phenomena (e.g., fire, water) with dynamic systems, typically using a large quantity of particles.

Particles

Individual elements in a particle system with properties like position or color, representing entities like water droplets.

Partial Differential Equation (PDE)

A mathematical equation modeling physical laws (e.g., heat transfer, fluid dynamics) in continuous systems.

Portable Network Graphics (PNG)

A raster image format with lossless compression and transparency, used for textures or visualization exports.

Rendering

Generating 2D images from 3D scenes, calculating lighting and effects for simulation visualizations so they can be projected onto a 2D monitor screen to show the scene.

RGBA

Color channels (Red, Green, Blue, Alpha) in digital images, with alpha indicating transparency for rendering.

Rigidbody

Physics simulation objects that maintain shape, used for modeling collisions or gravity in small quantities.

Shading

Determining surface color and brightness based on lighting, critical for realistic environmental visualizations.

Simulation Model

A computational representation of a real-world system to study its behavior, used for predictions and scenario testing.

Tagged Image File Format (TIFF)

A six-line text file with decimal numbers separated by dots on each line containing

geographic information on bitmaps, are ASCII text files describing the location, scale and orientation of an image, georeferencing them to real-world coordinates in GIS.

Tagged Image File Format (TIFF)

A raster graphics format supporting metadata and high bit depths, common in geospatial applications.

Triangulated Irregular Network (TIN)

A vector-based surface representation with irregular nodes and lines, used for detailed terrain modeling.

Unmanned Aerial Vehicle (UAV)

A drone capable of collecting aerial data, used to create 3D models or orthomosaics of study areas.

UV Texture

A 2D image mapped onto a 3D model using UV coordinates, adding detail like satellite imagery to terrain.

Video Random Access Memory (VRAM)

GPU memory for storing image data, vital for handling large textures and complex scenes.

Voxel

A 3D volumetric pixel, can be used in grids to represent volumetric terrain or build Eulerian fluid simulations.