Constantin Verine

BACHELOR OF SCIENCE
GAMES PROGRAMMING

What are the challenges and limitations of realistic real-time water physics simulation ? Analysis from an SPH implementation in a homemade 3D physics engine.

Bachelor thesis – Module 6GST0XF10x 0325
Référents : Elias Farhan, Nicolas Siorak.
Student n° : #12-135616

SAE Institute - Geneva
18.07.2025

8358 words

I, Constantin Verine, certify having personally written this Bachelor thesis. I also certify that I have not resorted to plagiarism and have conscientiously and clearly mentioned all borrowings from others.

Geneva, 18 July 2025

Foreword

Since my childhood, I have always been fascinated by how things work, from the mechanics of toys to the logic behind video games. This curiosity naturally evolved into a passion for programming, especially in the realm of real-time systems and game development. Over the years, I've spent countless hours experimenting, breaking, and rebuilding virtual environments, driven by the desire not just to play games, but to understand and create the systems that power them.

This bachelor project represents the culmination of that passion. By diving deep into the inner workings of physics simulation and exploring the challenges of performance and realism in a 3D environment, I have not only expanded my technical skills but also pushed myself to think critically and creatively. It reflects months of research, problem-solving, and dedication.

I hope this project serves as both a meaningful conclusion to my academic journey and a solid foundation for my future as a game programmer and software engineer.

Acknowledgement

First and foremost, I would like to express my deepest gratitude to my supervisor, Nicolas Siorak, for his invaluable guidance, support, and encouragement throughout this project. His insights and feedback were instrumental in helping me structuring and writing this thesis.

I would also like to thank Elias Farhan for providing a solid foundation in programming and game development, for fostering an environment that encourages curiosity, experimentation, and growth and without whom my physics engine wouldn't be what it is.

A special thanks to Olivier Pachoud, who has been an incredible source of motivation, collaboration, and shared coding sessions. Your feedback and camaraderie made this journey both productive and enjoyable.

Lastly, I want to thank my family for their unwavering support and belief in me. Their patience and encouragement were essential in helping me stay focused and determined from start to finish.

Abstract

This bachelor project focuses on the transition of a custom-built 2D physics engine to a functional 3D simulation framework, with a particular emphasis on fluid dynamics using Smoothed Particle Hydrodynamics (SPH). The objective was to design and implement a performant, scalable system capable of simulating particles in real time mostly on a mono threaded CPU architecture.

The project began with the extension of the existing architecture to support 3D math and collision detection. A naive SPH simulation was first implemented to establish the fundamentals of fluid behavior, followed by significant optimisation using a spatial hashing grid to efficiently retrieve neighboring particles. Finally, the fluid simulation was ported to a compute shader, using the GPU to drastically improve performance.

The result is a modular and extensible 3D physics engine that serves as both a learning platform and a foundation for future experimentation in physically-based simulation and game development.

Subject keywords: physics engine, fluid simulation, smoothed-particle hydrodynamics (SPH), optimisation, performance, compute shader

Table of Contents

1. Introduction

1.1 Starter

Simulating water in real time is known as one of the most complex challenges in physics-based simulations and computer graphics. Achieving a high level of realism while maintaining performance requires efficient numerical methods and optimised computations. One promising approach is Smoothed Particle Hydrodynamics (SPH), a particle-based method widely used for fluid simulations.

Over the past few decades, numerous communities have shown growing interest in fluid simulation, ranging from computational physics and applied mathematics to the visual effects industry and real-time graphics in video games. In academic research, fluid dynamics has long been a field of intense study, with scientists developing complex models to simulate natural phenomena like ocean currents, weather systems, and blood flow. These simulations often prioritize physical accuracy over computational performance.

In contrast, the visual effects (VFX) community, particularly in cinema and high-end animation, has adopted fluid simulation techniques to enhance realism in scenes involving water, smoke, and explosions. Here, the focus is on visual plausibility and artistic control, with less concern for real-time performance. More recently, the video game industry and interactive media developers have sought to incorporate fluid simulations into real-time environments, sparking innovations that balance realism with strict performance constraints. Among the most relevant and contemporary communities are those building real-time engines and game physics middleware, where Smoothed Particle Hydrodynamics (SPH) is increasingly explored for its scalability and GPU-friendly nature.

However, a persistent challenge lies in bridging the gap between physical realism and real-time interactivity. While SPH has proven capable of simulating convincing fluid behavior in precomputed environments, its integration into real-time systems often suffers from performance bottlenecks and numerical instability. Attempts to optimise the method such as spatial partitioning, adaptive time-stepping, and parallel computation have shown promise, but achieving both high visual fidelity and responsiveness remains an elusive goal (Müller et al., 2003).

It is within this context that the current study finds its relevance, given these ongoing challenges and partial solutions, it becomes pertinent to rigorously investigate the capabilities and limits of SPH in real-time simulations. Can it truly deliver the realism demanded by modern interactive applications without compromising performance?

1.2 Problematic

This project investigates the feasibility of achieving realistic water behaviour using a real-time, SPH-based simulation within a custom 3D physics engine, and the associated limitations. The focus is on evaluating the physical accuracy, computational constraints and scalability of this method when applied in real time. The study aims to identify key technical challenges, such as stability, performance bottlenecks and precision trade-offs, in replicating

fluid dynamics under strict real-time constraints. The project will highlight the potential and limitations of SPH for real-time water simulation.

1.3 Project Plan

This study is divided into four main stages. First, we describe the transition from a 2D to a 3D physics engine to lay the foundation for three-dimensional fluid simulation. Next, we implement the Smoothed Particle Hydrodynamics (SPH) method within this engine. Then, we focus on optimising the simulation, conducting performance measurements and visual assessments. Finally, we present our conclusions based on the results, discussing the limitations and potential improvements of the approach. Additionally, we will collaborate with another student working on fluid rendering to evaluate the visual fidelity and integration of the simulation.

2. Background and Related Work

Real-time water simulation is a key research topic in the fields of computer graphics, computational physics and game development. Various methods have been suggested for balancing realism and performance, each offering different advantages and limitations.

2.1 Fluid Simulation Techniques

Fluid dynamics simulations can be categorised broadly into two types: Eulerian grid-based and Lagrangian particle-based methods.

- **Eulerian Approaches** (e.g., Navier-Stokes equations solved on a grid) are widely used in offline simulations and precomputed effects. While they provide highly detailed results, their computational cost makes them impractical for real-time applications.
- **Lagrangian Methods**, model fluids as discrete particles interacting under physical forces. This method is particularly suited for real-time applications due to its adaptability and efficient neighbor search algorithms.

2.2 Smoothed Particle Hydrodynamics (SPH)

SPH (Smoothed Particle Hydrodynamics), introduced by (Gingold & Monaghan, 1977) (initially for astrophysical problems), has become a popular technique for simulating fluids. It approximates continuous fluid properties using a weighted sum of neighbouring particles, making it well-suited to real-time simulations. Modern SPH implementations include Weakly Compressible SPH (WCSPH) and Predictive-Corrective Incompressible SPH (PCISPH), which are designed to enhance stability and mitigate numerical errors.

2.2.1 Main concepts regarding the topic

Here are some important words and their definitions to bear in mind.

2.2.1.1 Density

In fluid dynamics, density is defined as the mass of a fluid per unit volume, and is a fundamental property used to describe its state.
In SPH, a particle's density is computed by summing the contributions of its surrounding particles using a smoothing kernel.

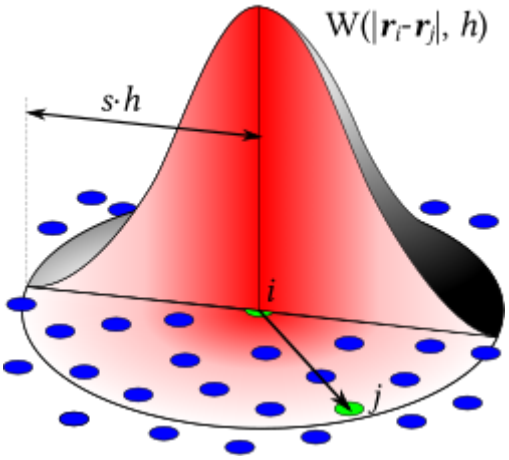

Figure 1: Schematic view of an SPH convolution

2.2.1.2 Pressure

In fluid dynamics, pressure refers to the force exerted per unit area by a fluid. In SPH, pressure is usually calculated using an equation of state based on the fluid's density.

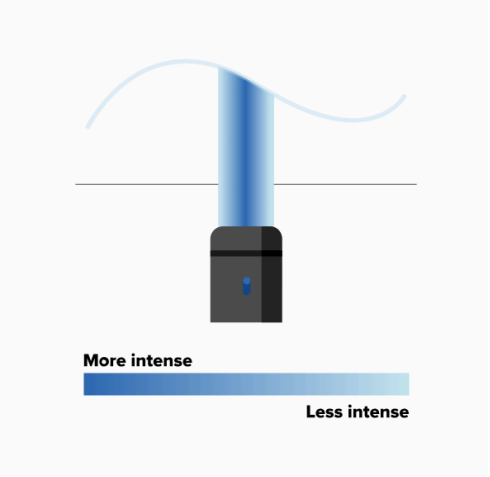

Figure 2: Laser representation of pressure curve

2.2.1.3 Viscosity

Viscosity is the internal friction of a fluid, which resists the relative motion of adjacent layers. It is also responsible for the diffusion of momentum within the fluid.

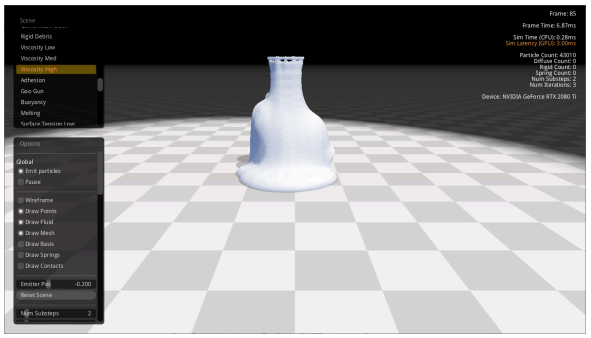

Figure 3: Nvidia Flex, high viscosity sample

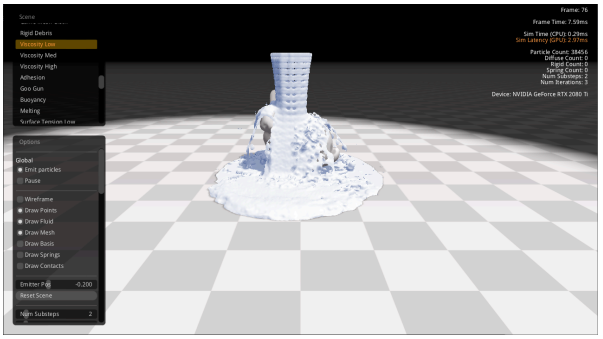

Figure 4: Nvidia Flex, low viscosity sample

2.2.1.4 Vorticity

Vorticity is a vector quantity that measures the local rotation, or 'spin', of fluid elements. It is defined as the curl of the velocity field.

Vorticity is particularly useful for visualising and analysing rotational structures, such as vortices and turbulence, within a fluid.

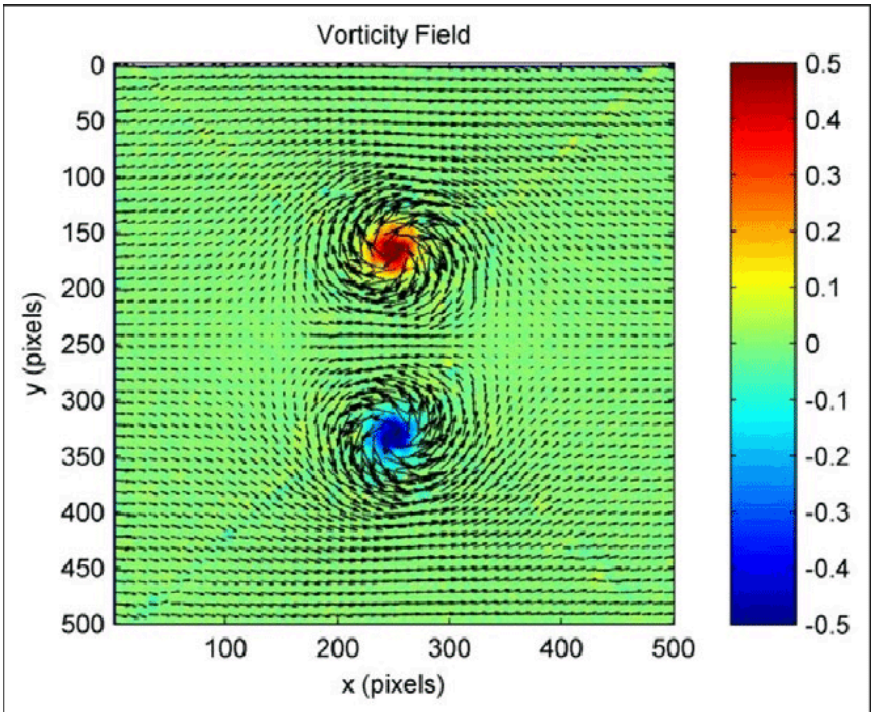

Figure 5: Schematic representation of vorticity field

2.3 Real-Time Applications and Performance Challenges

Real-time fluid simulation plays a crucial role in interactive applications such as video games, virtual reality, and simulation training environments. These use cases demand both a high degree of visual realism and computational efficiency, posing several technical challenges that must be carefully balanced.

- **Computational Complexity**: Simulating realistic fluid behavior requires tracking the dynamics of thousands or even millions of individual particles. Each particle must interact with its neighbors through pressure, viscosity, and external forces, leading to an $O(n \cdot k)$ computational complexity, where k is the average number of neighbors per particle. To make this tractable in real-time, efficient spatial partitioning techniques such as uniform grids or spatial hashing are used to accelerate neighborhood queries (Ihmsen et al., 2014). Without such structures, brute-force neighbor search becomes prohibitively expensive as the number of particles grows.

- **Memory and GPU Optimisation**: The rise of GPU programming through platforms such as CUDA has enabled significant performance improvements in fluid simulations. By parallelizing force computation, integration, and neighbor search, GPUs can simulate tens of thousands of particles in real time (Macklin et al., 2014). However, memory layout and cache coherence become critical at this scale. Simulation frameworks often rely on Structure of Arrays (SoA) data layouts and shared memory usage to minimize memory access latency and improve throughput.

- **Stability and Realism Trade-offs**: Ensuring numerical stability in real-time simulations often requires compromises between physical accuracy and performance. Time step size, in particular, is constrained to avoid instability. To compensate, techniques such as XSPH viscosity (Monaghan, 1992) are used to reduce numerical noise and promote smooth motion. Surface tension models and artificial pressure terms (Müller et al., 2003) help simulate cohesive behavior in liquids without requiring prohibitively small time steps. Furthermore, adaptive time-stepping strategies can be employed to dynamically adjust simulation resolution based on particle interactions, improving performance without significant loss of realism (Bender and Koschier, 2017).

In summary, achieving visually plausible fluid behavior in real time requires careful engineering and algorithmic trade-offs. Performance constraints must be addressed at both the algorithmic and hardware levels, while stability-enhancing techniques help bridge the gap between accuracy and interactivity.

2.4 Existing Implementations and Frameworks

Several frameworks and engines offer SPH-based fluid simulations:

2.4.1 LiquiGen and the Stable Fluids Method

LiquiGen by Jangafx is a real-time fluid simulation system designed primarily for use in game engines and interactive applications. It distinguishes itself from traditional particle-based methods like Smoothed Particle Hydrodynamics (SPH) by relying on grid-based (Eulerian) techniques to simulate the behavior of fluids. LiquiGen's design is influenced heavily by the foundational work of Jos Stam, particularly his 1999 paper *"Stable*

Fluids", which introduced a breakthrough in unconditionally stable fluid simulation methods suitable for real-time applications.

2.4.1.1 Stable Fluids: Background and Core Concepts

The *Stable Fluids* paper presented a novel approach to numerically solving the Navier-Stokes equations, the fundamental equations governing fluid motion on a fixed grid. Jos Stam's method was groundbreaking because it allowed for unconditional stability, meaning simulations could run with large time steps without becoming unstable or producing non-physical artifacts. This was in contrast to earlier fluid solvers, which often required very small time steps to remain stable, making them impractical for interactive applications.

The Stable Fluids algorithm is based on three main components:

1. **Advection using Semi-Lagrangian Methods**: Instead of moving fluid quantities forward in time (Eulerian advection), the algorithm traces backward along the velocity field to determine where fluid properties came from. This method, known as semi-Lagrangian advection, is stable even with large time steps and avoids the numerical diffusion problems of simpler advection schemes.

2. **Diffusion Solved with Implicit Methods**: Viscosity (diffusion of velocity) is handled using an implicit solver, which is unconditionally stable and does not require restrictive time step sizes. This is essential for simulating thicker or more viscous fluids in real time.

3. **Pressure Projection Step**: To enforce incompressibility, Jos Stam introduced a pressure projection step that adjusts the velocity field to be divergence-free. This involves solving Poisson's equation on the grid, typically using an iterative method like Gauss-Seidel or conjugate gradient.

These innovations allowed fluids to be simulated in a visually plausible and stable manner, even on limited hardware, making *Stable Fluids* a cornerstone of real-time fluid dynamics research.

2.4.1.2 LiquiGen: Real-Time Grid-Based Fluid Simulation

Building on the principles introduced by Jos Stam, LiquiGen implements a grid-based (Eulerian) fluid simulation system optimised for modern GPU architectures. Unlike SPH or particle-based systems, which represent fluids as discrete particles, LiquiGen discretizes the simulation space into a uniform 3D grid and stores fluid properties (velocity, pressure, density) at grid cells or cell faces.

LiquiGen retains the key advantages of the Stable Fluids approach (Vassvik, 2022):

- **Unconditional stability**, allowing large time steps and real-time interaction.

- **Efficient pressure projection**, using optimised solvers adapted for GPU execution.

- **Visual plausibility**, including swirling flows, vortices, and incompressible motion.

To enhance visual richness and interactivity, LiquiGen also integrates features such as:

- **Vorticity confinement**, to compensate for the damping of small-scale detail caused by numerical diffusion.
- **Obstacle interaction**, enabling two-way coupling between fluid and solid objects using signed distance fields or velocity boundary conditions.
- **Multiresolution grids**, to simulate large-scale effects efficiently while preserving detail where necessary.

LiquiGen is particularly well suited for applications where stability, performance, and plausible visual fidelity are more important than strict physical accuracy. This includes video games, interactive simulations, VR experiences, and special effects. Its reliance on GPU compute shaders and CUDA enables simulations involving thousands of grid cells at interactive frame rates.

Figure 6: LiquiGen & Blender - 4K bubbles and foam liquid simulation test

2.4.2 NVIDIA Flex and the Particle-Based Method

NVIDIA Flex is a unified, particle-based physics simulation framework developed by NVIDIA, designed to fully leverage the massively parallel architecture of modern GPUs. Unlike traditional rigid body physics engines, which often handle specific interaction types separately, Flex adopts a unified approach where all simulated entities (rigid bodies, soft bodies, fluids, cloth, and gases) are represented as particles. This generalization allows Flex to simulate a broad range of physical phenomena within a consistent computational framework (Macklin et al., 2014).

At the core of Flex lies the Position-Based Dynamics (PBD) method, a constraint-based simulation technique that prioritizes stability and controllability. Unlike force-based methods, which compute forces and integrate them over time, PBD updates particle positions directly by satisfying a set of positional constraints. This method is inherently well-suited to GPU architectures due to its parallelism and superior numerical stability, especially over long simulation times or under complex interactions.

For fluid simulation, Flex uses a particle-based representation similar to Smoothed Particle Hydrodynamics (SPH), but adapted to the PBD paradigm. Fluid particles maintain a target density through iterative positional corrections, effectively simulating pressure forces and preserving incompressibility. Cohesion and surface tension effects are modeled via particle-particle attraction forces and anisotropic kernels, enabling realistic phenomena such as droplet formation and splashing.

To achieve high performance, Flex takes advantage of CUDA-enabled GPUs, executing all major simulation steps (neighbor searches, constraint solving, and position integration) in parallel. These neighbor searches are accelerated using spatial partitioning structures, such as uniform grids or hierarchical grids, optimised for GPU execution.

Flex also supports continuous collision detection and two-way coupling with rigid bodies, allowing fluid and solid objects to interact dynamically. Thanks to the unified particle representation, both fluids and rigid bodies can share the same solver, eliminating the need for complex interfacing or data conversion.

NVIDIA Flex has been adopted in a variety of real-time applications, including games and interactive simulations, where both visual realism and performance are essential. While it favors stability and speed over strict physical accuracy, Flex demonstrates how modern GPU architectures can be harnessed to perform complex physics simulations in real time. Its architecture marks a shift in simulation design from specialized, CPU-bound solvers to highly parallel, GPU-based systems.

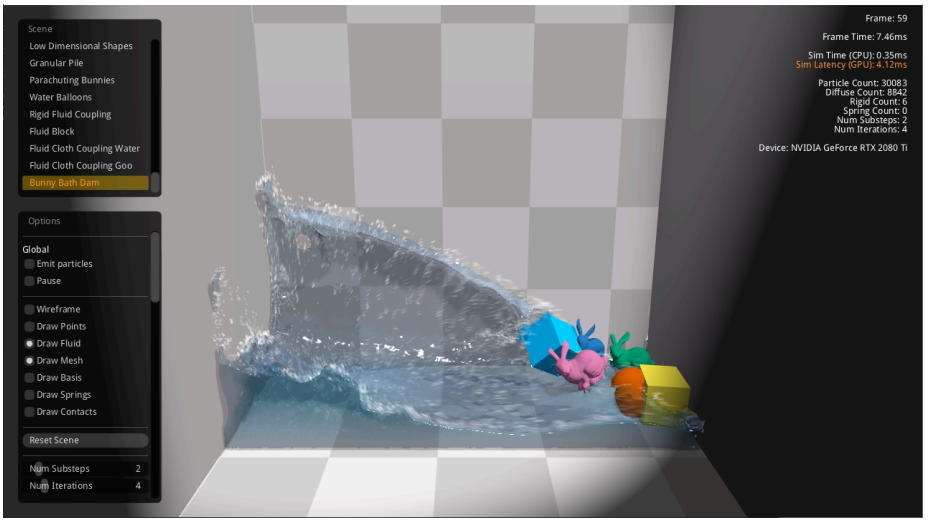

Figure 7: Nvidia Flex, Bunny Bath Dam Sample

2.4.3 NVIDIA PhysX

NVIDIA PhysX is a real-time physics simulation engine widely used in the game development industry. While it is best known for its rigid body dynamics and collision detection, PhysX also supports fluid simulation, which is based on particle methods and GPU acceleration. Unlike Flex, which uses position-based dynamics for flexibility and stability, PhysX implements a more traditional Smoothed Particle Hydrodynamics (SPH) approach, focusing on realistic incompressible or weakly compressible fluid behavior.

At its core, PhysX models liquids as discrete particles, each representing a small volume of fluid. These particles interact using SPH formulations, which approximate continuum fluid dynamics by computing local properties such as density, pressure, and viscosity from contributions of nearby particles. In each simulation step, particles are advanced using Newtonian mechanics, with forces derived from pressure gradients, viscosity, gravity, and user-defined interactions.

Density is estimated by summing the mass contributions of neighboring particles, weighted by a smoothing kernel such as the poly6 kernel. Pressure is calculated using an equation of state, penalizing deviations from a rest density. The resulting pressure forces are computed from pressure gradients, typically using the spiky kernel for precise directionality. Viscosity effects are simulated by analyzing velocity differences between neighbors and applying damping forces, often using the Laplacian viscosity kernel.

Unlike CPU-based implementations, PhysX exploits CUDA-enabled GPUs to parallelize computations. Particle data (positions, velocities, densities, and more) are stored in GPU-optimised buffers, and all major operations (neighbor searches, force calculations, and time integration) are performed on the GPU. A spatial grid (or uniform hash grid) divides the simulation space into fixed-size cells, allowing efficient neighbor queries by limiting checks to adjacent cells. This reduces the computational complexity from $O(n^2)$ to approximately $O(n)$, enabling real-time simulation of thousands of particles.

PhysX also includes collision handling with both static and dynamic objects. Particles detect contact with colliders and respond with repulsive forces, enabling effects like surface tension, adhesion, and splashing. Additionally, two-way coupling between fluids and rigid bodies allows fluid momentum to influence and move lightweight objects.

A key distinction between PhysX and Flex lies in their target use cases and physical fidelity. While Flex emphasizes stability and versatility via position-based methods, PhysX targets physically accurate simulation of incompressible fluids, though it demands higher computational resources. This made it particularly suited for visual effects requiring plausible fluid motion, though adoption in production-level games was often limited by performance and hardware constraints.

PhysX was prominently featured in NVIDIA GameWorks, powering demos of real-time water surfaces, pouring liquids, and splash effects. However, due to its complexity and performance costs, it was eventually phased out in favor of newer, more unified GPU-accelerated systems.

Recent versions of PhysX (like PhysX 5) have introduced improved fluid simulation using Finite Element Methods (FEM) and advanced particle-based models, incorporating lessons from both PhysX and Flex (Nvidia-omniverse, 2022).

In summary, NVIDIA has an implementation of SPH-based fluid simulation within the PhysX framework, using GPU acceleration to simulate dynamic and particle-based fluid behavior in real time. Its integration with other physics systems and support for physically plausible interactions made it a powerful tool for high-fidelity visual effects and interactive simulations.

2.4.4 Open-source libraries

Beyond commercial solutions like NVIDIA Flex and PhysX, the research and development community has produced several open-source fluid simulation libraries that offer robust, customizable tools for both academic study and production use. Among the most notable are DualSPHysics, Position-Based Fluids (PBF), and SPlisHSPlasH, each implementing different methods of particle-based fluid simulation, optimised for performance, scalability, and physical accuracy.

2.4.4.1 DualSPHysics

DualSPHysics is a Smoothed Particle Hydrodynamics (SPH) based solver developed collaboratively by several universities and research centers. Designed for high-performance computing applications, it is especially well-suited for engineering simulations involving free-surface flows, wave-structure interactions, and dam break scenarios.

Built in C++ and CUDA, DualSPHysics is capable of simulating millions of particles in real time by leveraging GPU acceleration. Its design emphasizes physical accuracy over visual appeal, making it ideal for scientific research and hydrodynamic modeling. Features include adaptive time stepping, open boundary conditions, floating body interactions, and wave generation tools. The software also supports MPI-based parallelization, allowing it to scale across multi-GPU or multi-node systems.

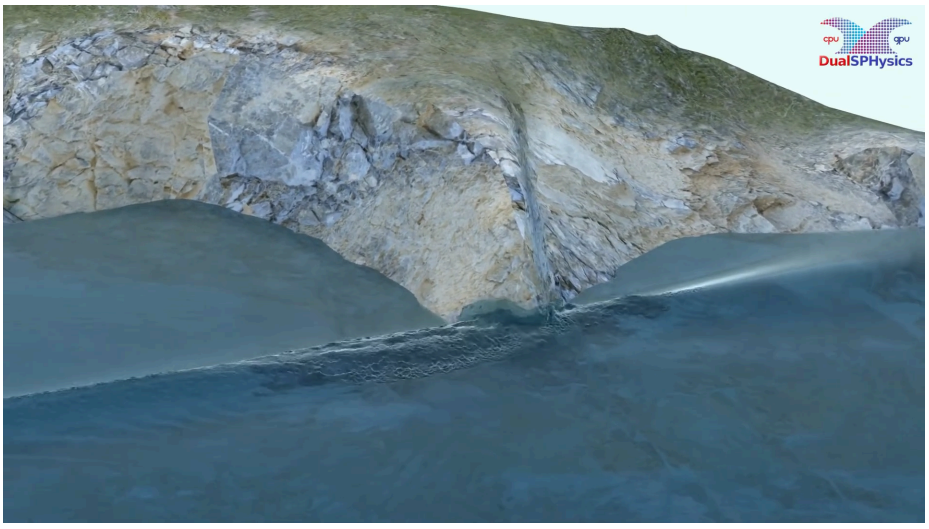

Figure 8: Interaction of large waves with a real coast using Blender & DualSPHysics (SPH on GPU)

2.4.4.2 Position-Based Fluids (PBF)

Position-Based Fluids (PBF) is a constraint-based simulation method derived from Position-Based Dynamics (PBD), initially introduced by (Macklin and Müller, 2013). This method prioritizes stability, real-time performance, and visual plausibility, making it popular in games and interactive applications.

Unlike traditional SPH, which integrates forces over time, PBF enforces density constraints directly by iteratively correcting particle positions, maintaining incompressibility and preventing fluid collapse. This method supports features such as surface tension, vorticity confinement, and viscosity, all while remaining highly parallelizable, ideal for GPU computation.

Several open-source implementations of PBF are available, often integrated into larger real-time physics frameworks or graphics engines, making it accessible for developers aiming to include fluid effects without sacrificing simulation speed or stability.

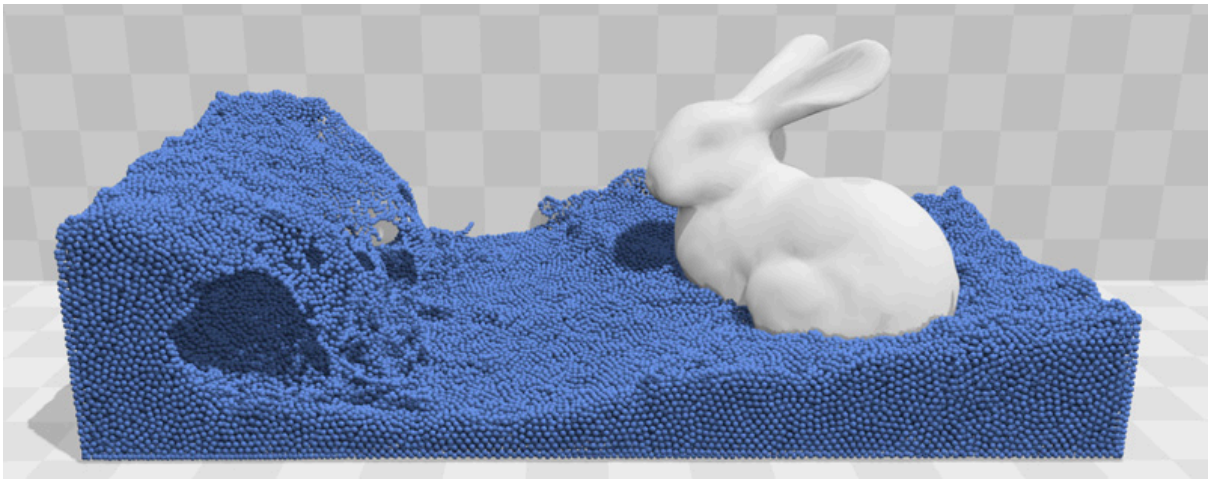

Figure 9: Bunny taking a bath

2.4.4.3 SPlisHSPlasH

SPlisHSPlasH is a high-quality, open-source fluid simulation library focused on physically accurate SPH simulations for real-time graphics. Developed by Jan Bender, (since 2016 and the Interactive Graphics and Simulation Group at the University of Freiburg, it features modular, highly optimised SPH solvers designed for GPU and multi-threaded CPU execution.

SPlisHSPlasH supports various fluid solvers, including:

- **DFSPH** (Divergence-Free SPH): for improved volume preservation
- **IISPH** (Implicit Incompressible SPH): for accurate pressure solving
- **PCISPH** (Predictive-Corrective Incompressible SPH): for stable time integration

The library is known for its extensibility, clean C++ architecture, and integration with visualization tools like OpenGL. It also includes boundary handling, rigid body coupling, and realistic surface rendering, making it suitable for both academic experimentation and interactive content creation.

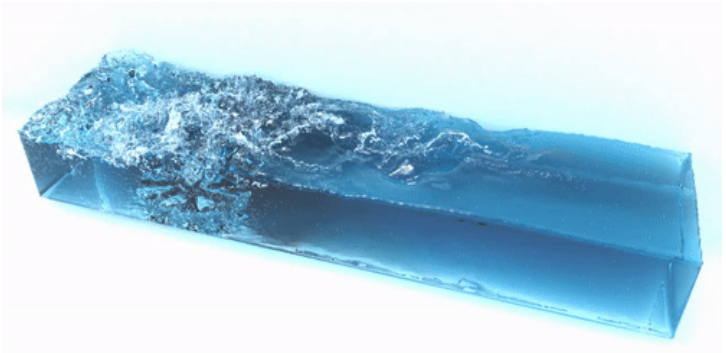

Figure 10: SPlisHSPlasH showcase

Open-source fluid simulation libraries like DualSPHysics, Position-Based Fluids, and SPlisHSPlasH provide powerful alternatives to commercial tools, each tailored to specific use cases, from scientific accuracy and scalability to real-time interactivity and visual fidelity. These frameworks not only support cutting-edge research but also enable broader access to advanced fluid simulation techniques across academia, industry, and independent development.

3. Methodology

This bachelor project employs a methodology that combines theoretical understanding with the practical analysis of existing fluid simulation implementations. The aim is to study prior works and extract valuable insights in order to guide the development and optimisation of a custom 3D Smoothed Particle Hydrodynamics (SPH) fluid simulation system. This chapter outlines the chosen case studies and the anticipated development methodology.

3.1 Study Cases

To ground the development process in concrete examples and to better understand the nuances of fluid simulation techniques, two public projects were selected as study cases: Elijah Nicol's SPH-Fluid-Simulator and Sebastian Lague's Fluid Simulation. These projects each approach fluid simulation from different perspectives and offer valuable insights into both algorithmic structure and performance considerations. The comparative study of these two works serves multiple purposes: to validate physical models, assess computational efficiency and identify key implementation strategies. Lessons learned from these implementations are instrumental in shaping the structure and optimisation of this simulation.

3.1.1 Elijah Nicol's SPH-Fluid-Simulator

A good example of an independently developed SPH solver is the work of the user Elijah Nicol *on* GitHub, who progressively explored different computational strategies for

implementing Smoothed Particle Hydrodynamics. His project stands out for the solver's evolution from a single-threaded CPU implementation to a multithreaded version, and finally to a planned migration toward GPU-based computation.

Initially, Elijah Nicol developed a monothreaded CPU-based SPH solver, which served as a baseline for understanding particle-based fluid dynamics. This first version included fundamental SPH components such as density estimation, pressure and viscosity forces, all computed sequentially. Although functionally correct, the performance was inherently limited by the sequential nature of the code, particularly when simulating a large number of particles.

To overcome these limitations, he transitioned to a multithreaded implementation using CPU parallelism. By distributing the workload across multiple threads, he achieved significant performance improvements. This version leveraged concurrent computation of per-particle properties such as force accumulation and density calculation, effectively reducing the computational complexity associated with neighbor interactions. However, scalability remained constrained by the limited number of CPU cores and the overhead introduced by synchronization between threads.

Recognizing the inherently parallel nature of SPH simulations, Elijah Nicol proposed the GPU as a more suitable platform for the solver. He outlined the advantages of compute shaders and the high degree of data parallelism offered by modern graphics hardware. Although his GPU implementation remained at the planning stage, his rationale was well-founded: the large number of available GPU cores and their optimised memory bandwidth are well-matched to the parallel structure of SPH algorithms, especially for tasks like neighbor search, density computation, and pressure force evaluation.

Overall, Elijah Nicol work provides a useful reference for understanding the trade-offs and performance implications of different computational backends for SPH solvers. His project illustrates the natural progression from sequential to parallel computation and highlights the GPU as a compelling target architecture for real-time particle-based simulations.

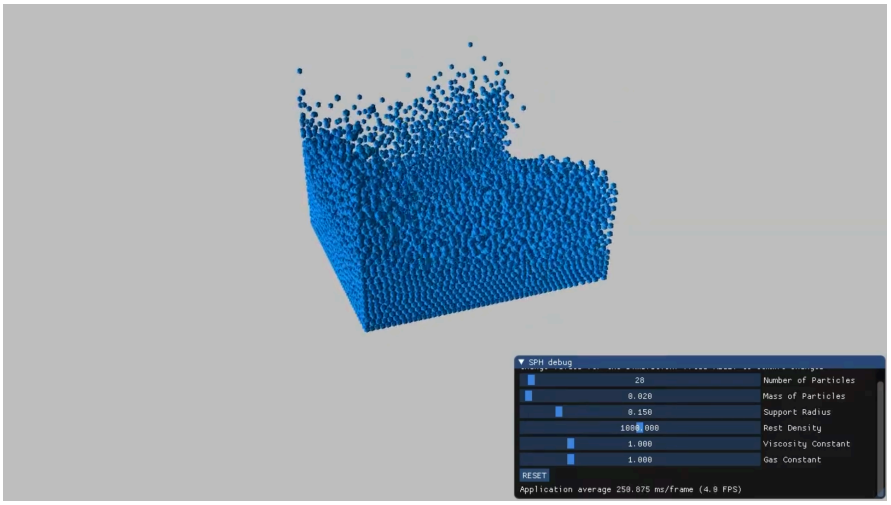

Figure 11: Simulation showcase

3.1.2 Sebastian Lague’s Fluid-Sim

Sebastian Lague’s fluid simulation project within the Unity engine provides a notable example of an incremental and educational approach to Smoothed Particle Hydrodynamics (SPH) implementation. His work is particularly valuable for its clear breakdown of the solver's computational stages and his gradual transition from CPU-based methods to GPU-accelerated techniques using compute shaders.

The initial version of Sebastian Lague's SPH solver was executed entirely on the CPU, where each particle's physical properties (including density, pressure, and the resulting forces) were computed in a sequential or minimally parallelized fashion. While pedagogically clear and structurally simple, this approach was inherently limited in terms of scalability and real-time performance, especially as the number of simulated particles increased. Sebastian Lague's implementation followed standard SPH formulations, applying the Navier-Stokes equations through kernel-based approximations for inter-particle interactions.

To address the performance bottlenecks inherent to CPU-bound execution, Sebastian Lague transitioned the solver to run on the GPU via Unity's compute shader pipeline. This transformation involved parallelizing the core stages of the SPH algorithm: neighbor search, density calculation, pressure and viscosity force evaluation. By leveraging the parallel processing capabilities of modern GPUs, Sebastian Lague was able to dramatically increase simulation performance, enabling real-time interaction even with tens of thousands of particles.

One of the critical steps in this transition was restructuring data access patterns and computation logic to suit the GPU’s massively parallel architecture. The shift required careful management of memory buffers, thread groups, and synchronization across compute threads. Despite Unity’s abstractions, Sebastian Lague’s work demonstrates a sophisticated understanding of how to map traditional CPU logic into the compute shader paradigm while maintaining numerical stability and coherence in the fluid behavior.

Sebastian Lague's compute-shader-based solver stands out not only for its educational clarity but also for its efficient and practical design. It exemplifies a successful application of GPU acceleration for particle-based fluid simulation and serves as a valuable reference for real-time SPH implementation in game engines or interactive environments.

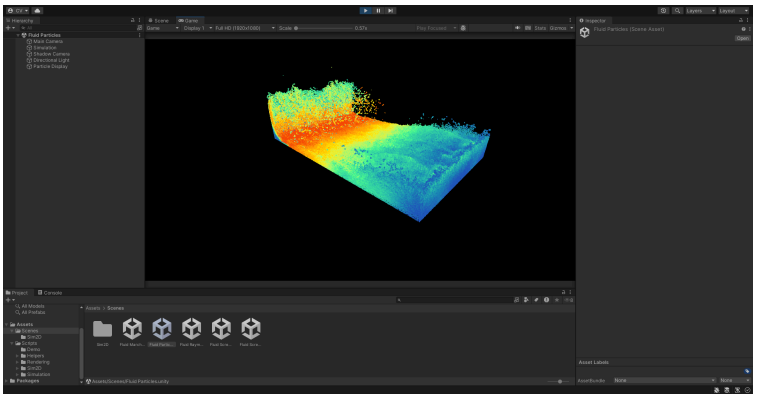

Figure 12: Sebastian Lague’s simulation in Unity

3.2 Influence of preliminary research on practical project

The planning of the fluid simulation development in this project is significantly influenced by two prior study cases: the monothreaded CPU-based SPH fluid simulation by Elijah Nicol, and the GPU-accelerated implementation in Unity using compute shaders by Sebastian Lague. These two sources of preliminary research provide both theoretical grounding and practical insights, each contributing to different aspects of the chosen methodology.

Elijah Nicol's implementation presents a clear and didactic version of SPH running on a single-threaded CPU. Its main strength lies in the simplicity of the architecture, which facilitates a thorough understanding of the fundamental mechanisms involved in SPH simulation. Following a detailed analysis of this implementation, it becomes possible to clearly identify both effective strategies to adopt and common pitfalls to avoid when initiating the development of a new SPH system.

The second study case, based on Sebastian Lague's GPU-based fluid simulation, introduces a performant approach to real-time SPH using compute shaders. This implementation demonstrates the advantages of massively parallel architectures, particularly when processing large numbers of interacting particles.

This research illustrates the transition from a mono threaded CPU model to a compute shader-based system capable of real-time performance. The architectural shift requires a redesign of the particle interaction system, including the introduction of a spatial partitioning structure.

Together, the two study cases provide a complementary foundation for the development of the fluid simulation. The CPU-based version offers clarity, correctness, and a means to verify simulation behavior, while the compute shader approach introduces techniques necessary to achieve performance suitable for real-time applications.

The progression from Elijah Nicol's serial model to Lague's parallel architecture mirrors the planned evolution of the project itself, from a focus on correctness and clarity to an emphasis on scalability and efficiency. Lessons learned from the CPU model inform the decision making for the early stages of development, while insights from the GPU implementation will guide the optimisation and final architecture.

In summary, the preliminary research directly shapes the design decisions, implementation strategies, and performance goals of the practical project. It provides both the theoretical framework and the technical roadmap necessary to implement a reliable and performant SPH fluid simulation.

3.3 Anticipated methodology

The project involves the design and implementation of a fluid simulation using Smoothed Particle Hydrodynamics (SPH) within a homemade 3D physics engine. The

methodology will follow a progressive development pipeline, emphasizing modular design, performance analysis, and iterative optimisation.

1. **Porting from 2D to 3D**
   The starting point of the project is an existing 2D physics engine named Bark and developed by the author. This codebase will be adapted to handle three-dimensional simulations, which will require a significant refactoring of core components, including collision detection, physics integration, and spatial data structures.

2. **Math Library Transition**
   To improve compatibility and performance for 3D calculations, the engine's custom math library will be replaced with a more optimised library suited for 3D vector and matrix operations (e.g., DirectXMath or similar). This change will facilitate SIMD-friendly operations and help unify spatial transformations.

3. **Naive SPH Implementation**
   A basic version of SPH will be implemented using mono threaded CPU-based calculations. This version will focus on validating the fluid simulation model and include key components such as density, pressure and viscosity computations as well as all the force calculations.
   Each particle's behavior will be computed independently, using brute-force neighbor searches, resulting in a $3*(O(n^2))$ complexity that will surely highlight the need for optimisation.

4. **Hash Grid Optimisation**
   To improve performance, a uniform spatial hash grid will be needed to accelerate neighborhood lookups. This grid will divide space into cells and will hash particle positions to cell indices, enabling reduced neighbor search complexity. This optimisation will significantly reduce computational overhead while maintaining simulation quality.

5. **Profiling and Iterative Optimisation**
   Throughout development, performance profiling will be conducted using tools such as Tracy Profiler to identify bottlenecks and guide further optimisations. Specific attention will be given to the framerate and the number of particles

6. **Pooling with another student**
   Collaboration with the student Olivier Pachoud who will be responsible for developing a realistic water rendering technique using ray tracing will take place. The goal of this cooperation is to apply visually advanced rendering (refraction, reflection, light absorption, etc) on top of the fluid data produced by the SPH simulation. The integration will probably involve sharing particle positions, densities, and possibly velocity data to drive the visual appearance of the fluid in a way that aligns with the physical simulation. This pooling of efforts will allow the project to showcase both accurate fluid behavior and high-quality visual output.

4. Practical Project

This project was developed under the constraint of single-threaded CPU execution, meaning all simulation and processing tasks (except the compute shader stage) are performed sequentially on a single core. This limitation emphasizes algorithmic efficiency and careful resource management, as no parallelism or multithreading is leveraged to improve performance.

4.1 Resources

This section outlines the hardware and software used throughout the development and testing of the project. These resources were essential for implementing, debugging, profiling, and evaluating the real-time SPH-based water simulation in a custom 3D physics engine.

4.1.1 Hardware

The project was developed and tested on two different machines: a primary development setup and a secondary one. This allowed for additional performance evaluation and comparison across varying hardware configurations.

4.1.1.1 Primary setup

- **CPU**: Intel® Core™ i9-9900K @ 3.60GHz
- **RAM**: 64 GB
- **GPU**: NVIDIA GeForce RTX 2080 Ti (11GB)

4.1.1.2 Secondary setup

- **CPU**: AMD Ryzen 7 5800U with Radeon Graphics 1.90 Ghz
- **RAM**: 16 GB
- **GPU**: NVIDIA GeForce RTX 3050 Ti Laptop GPU

4.1.2 Software

The following software tools and libraries were used during the development process. These supported compilation, debugging, rendering, and performance profiling of the simulation:

- **Operating System**: Windows 10 for primary setup and Windows 11 for secondary setup

- **IDE**: Visual Studio 2022
- **Build System**: CMake 3.29.0
- **Dependency Management**: vcpkg
- **Graphics API**: OpenGL, using FreeGLUT and GLEW
- **Profiling Tool**: Tracy
- **UI and Debugging Utilities**: Dear ImGui
- **Math Library**: DirectXMath

4.1.3 Specific resources for pooling

The Falcor framework version 8.0 (NVIDIA, August 2024) was selected by Olivier Pachoud due to its ease of use and efficiency in creating 3D ray-traced scenes. This advantage stems from its encapsulation of the DirectX 12 API (Microsoft, 2014) along with its Raytracing extension, known as DXR (DirectX Raytracing) (Microsoft, 2018), which significantly simplifies the implementation of raytracing pipelines.

4.2 Porting from 2D to 3D

In the initial phase, the custom math library was replaced with DirectXMath to leverage SIMD optimisations and enhance performances. Then all two-dimensional vectors were converted to three-dimensional ones, and polygons were removed as their 3D versions were not required for the SPH simulation. The rendering was implemented using GLUT due to its simplicity, alongside the development of a dynamic camera system. All relative positions were updated to align with the new world coordinate system. The spatial data structure was modified from a quadtree to an octree, during which a significant bug related to trigger collisions was encountered and resolved after considerable effort. At this stage, a sample named *WaterBathSample* was created, where particles were instantiated to observe their behavior prior to the implementation of the SPH algorithm.

4.3 Naive SPH Implementation

The implementation of Smoothed Particle Hydrodynamics (SPH) began with an evaluation of various data structures for storing particles, with the primary objective of preserving physical interactions between fluid particles and other physics objects. After a series of tests, the chosen approach involved reusing the existing **Body** class (originally designed for rigid body objects) by adapting it to represent fluid particles. This class includes essential attributes such as position, velocity, and a mechanism for applying forces. To

distinguish fluid particles from other types of bodies within the simulation, an enumeration value **BodyType::FLUID** was introduced.

```
enum class BodyType { DYNAMIC, STATIC, FLUID, NONE };

class Body {
public:
    XMVECTOR Position = XMVectorZero();
    XMVECTOR Velocity = XMVectorZero();

    XMVECTOR PredictedPosition = XMVectorZero();

    float Mass = -1.f;  // Body is disabled if mass is negative
    BodyType Type = BodyType::DYNAMIC;

private:
    XMVECTOR _force = XMVectorZero(); // Total force acting on the body
```

Figure 13: Body class

To simulate realistic fluid behavior, three primary physical quantities were computed for each fluid particle: density, pressure, and viscosity. These computations were carried out using a straightforward, unoptimised approach. The algorithm involved iterating over every particle in a brute-force manner, resulting in a time complexity of $3*(O(n^2))$, where n is the total number of fluid particles.

Density was computed using the standard SPH formulation, which sums the contributions of nearby particles weighted by a smoothing kernel function. The poly6 kernel was used for its smoothness and suitability for density estimation. For each particle, its density was calculated by iterating through all other particles and accumulating their mass contributions, multiplied by the kernel evaluated at the distance between the two particles.

Once density was determined, pressure was computed using an equation of state derived from the ideal gas law. This formulation relates pressure to density through a stiffness constant and a rest density, ensuring that the fluid behaves compressibly in accordance with SPH conventions.

Viscosity forces were also calculated using a naive implementation of the standard SPH viscosity formulation. This involves computing the velocity difference between pairs of particles and applying a viscosity kernel (commonly the spiky or Laplacian viscosity kernel). The resulting force is then accumulated to influence each particle's velocity over time.

Due to the lack of spatial partitioning or neighbor search optimisations, all particle interactions were evaluated globally, making the simulation computationally expensive for a large number of particles. Nonetheless, this naive implementation served as a reference baseline for later optimisations using acceleration structures such as spatial hash grids.

4.4 Optimisation

In an effort to reduce the computational complexity inherent in the naive SPH implementation, an initial attempt was made to reuse the existing spatial partitioning structure already employed for physical object interactions: an octree. The idea was to leverage the octree to quickly identify neighboring fluid particles, thereby avoiding the need for an exhaustive $O(n^2)$ comparison.

The octree, however, was originally designed to manage rigid body interactions and was optimised for sparse and unevenly distributed object sets. When applied to the fluid simulation, which involves a dense and relatively uniform distribution of particles, the octree structure proved to be suboptimal. Several limitations emerged: the cost of inserting and updating fluid particles into the tree each frame was non-trivial, and querying for neighbors within a fixed radius often required descending multiple branches or traversing several neighboring nodes, leading to performance bottlenecks.

Given these drawbacks, a spatial hash grid was implemented as an alternative neighbor search strategy. This structure partitions space into uniform cells, each storing a list of particles within its bounds. Fluid particles are hashed into grid cells based on their positions, and neighbor queries are limited to a particle's surrounding cells within the smoothing radius. This approach drastically reduces the number of comparisons per particle, achieving average-case performance closer to linear complexity with respect to the number of particles.

```
struct XMINT3Equal { ... };

// Converts a vector to grid indices
inline XMINT3 getGridIndex(const XMVECTOR& pos) { ... }

// Hash function for the grid
struct GridHash { ... };

constexpr XMINT3 offsets[] = { ... }

struct SpatialHashGrid {
    std::unordered_map<XMINT3, std::vector<BodyRef>, GridHash, XMINT3Equal> grid;

    void clear() { ... }

    // Add particle to the grid
    void insertParticle(const BodyRef& ref, const XMVECTOR& position) { ... }

    // Find neighbors of a particle based on its position and radius
    std::vector<BodyRef> findNeighbors(const XMVECTOR& position) { ... }
};
```

Figure 14: Spatial Hash Grid structure

The spatial hash grid proved to be significantly more efficient for dense particle systems, especially when combined with a fixed smoothing length and uniform particle sizes. It offered constant-time insertion and fast lookup for neighbor searches, making it a better-suited solution for SPH simulations than the previously used octree.

```cpp
void World::Update(const float deltaTime) noexcept
{
#ifdef TRACY_ENABLE
    ZoneScoped;
#endif
    UpdateForces(deltaTime); // for all bodies
    updateGrid(); // for fluid particles

    computeNeighborsDensity();
    computeNeighborsPressure();
    computeNeighborsViscosity();
    computeNeighborsVorticity();

    SetUpOctTree(); // for all physical colliders
    UpdateOctTreeCollisions(OctTree.Nodes[0]);
}
```

Figure 15: World Update function with all SPH computations

4.5 Pooling

After completing the simulation optimisations, the next step involved integrating it into the graphics rendering framework developed by the student Olivier Pachoud. This integration was carried out using his Falcor-based application, with the build and configuration managed through CMake.

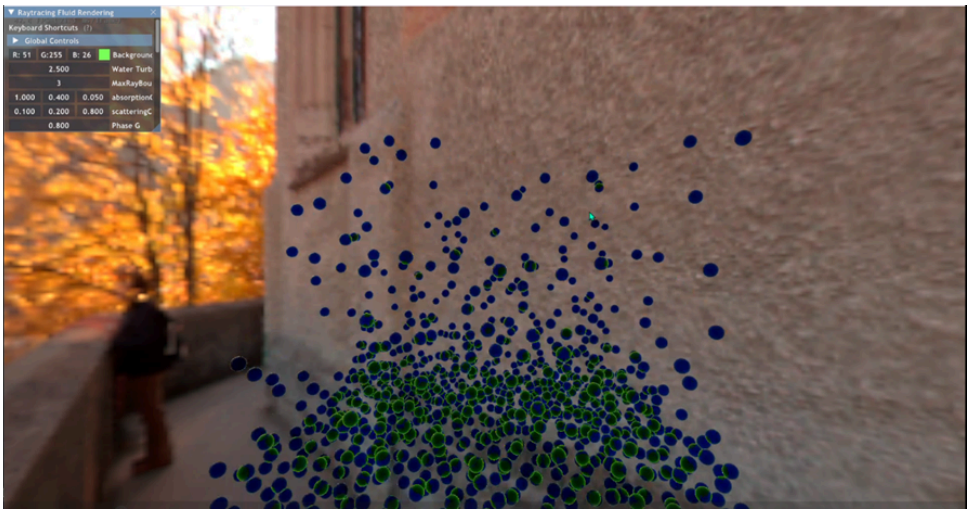

Figure 16: First step of the pooling (1000 particles)

4.6 Compute Shader Acceleration

To extract a surface from the fluid simulation, a density map must be generated by sampling fluid density at regular spatial intervals, determined by the resolution of a 3D texture. However, since the original simulation was CPU-based, evaluating density at numerous positions was computationally expensive. To overcome this limitation, the simulation was ported to a compute shader, enabling parallel computation of the density field directly on the GPU. This approach significantly accelerated the generation of the density map and enhanced the overall performance of the SPH simulation.

After transitioning the CPU-based simulation to a compute shader, a significant challenge emerged: the spatial hash grid previously used was incompatible with GPU architecture. Specifically, the grid cells stored collections of particles, a structure that is not well-suited for parallel execution on the GPU due to memory access conflicts.

To optimise memory alignment and access patterns, a Bitonic Sort algorithm was implemented on the GPU to sort particles according to their spatial cell indices. This sorting ensures better memory access, thereby significantly improving the efficiency of range queries. As a result, neighboring particles can be retrieved in constant time during the density sampling process, enhancing both performance and scalability of the simulation.

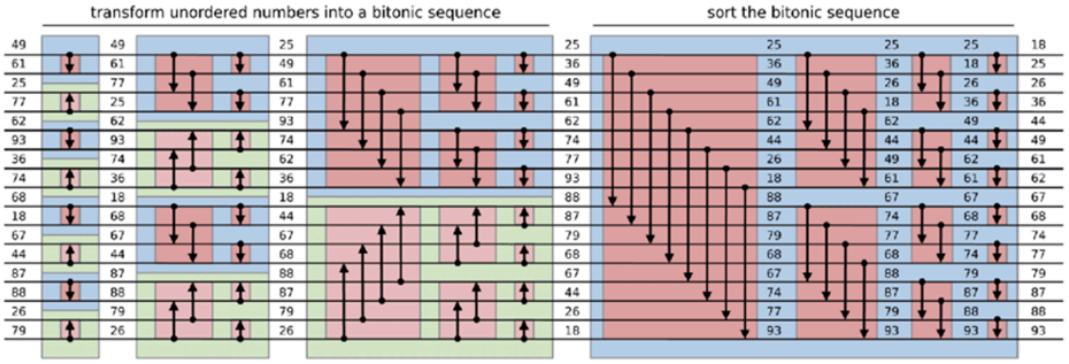

Figure 17: Visualization of the Bitonic Sort algorithm

Following the implementation of the Bitonic Sort algorithm on the GPU, a limitation was encountered related to thread availability. Specifically, the parallel nature of the original algorithm imposed a hard limit, allowing only 1,024 particles to be processed simultaneously due to the maximum number of threads per thread group. To overcome this constraint, the sorting procedure was restructured into an iterative variant commonly referred to as Bitonic Merge Sort with iterations. This version decomposes the sorting process into multiple sequential passes, each executed with a fixed number of threads. By iterating over these passes, the algorithm is no longer constrained by thread group size, enabling the efficient sorting of arbitrarily large particle arrays entirely on the GPU.

With the simulation now executed on the GPU using compute shaders, and the neighbor search process optimised through a spatial hash grid with the bitonic merge sort with iterations algorithm, the system achieves stable real-time performance with particle counts reaching up to 40,000.

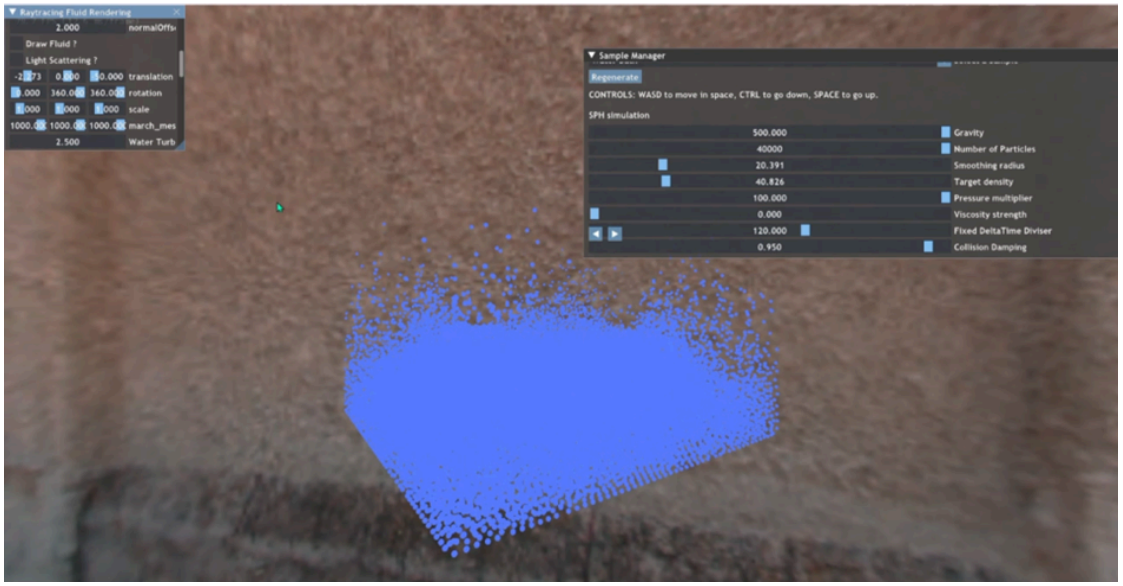

Figure 18: Final step of the pooling (40’000 particles)

Furthermore, a matrix transformation step has been integrated into the boundary collision response, enabling arbitrary rotation and scaling of the simulation domain.

5. Quantitative analysis

Performance evaluation is conducted using a structured testing protocol aimed at measuring the efficiency of the simulation under increasing computational loads. The main metric examined is the maximum number of particles that can be simulated within the time constraints of a single frame, which is fixed at 16.67 milliseconds (equivalent to 60 frames per second).

Detailed timing data is collected for the entire simulation process, as well as for each individual computational stage. These stages include grid construction, density estimation, pressure force computation and viscosity force calculation.

To capture performance trends and mitigate the impact of outliers, both the average and median execution times are recorded for each of these steps. Profiling is carried out using Tracy, a high-performance, real-time profiling tool that allows for precise, fine-grained analysis of time-critical sections of the simulation pipeline.

For reference, the performance figures for other fluid simulations vary widely. For example, Sebastian Lague’s multithreaded, CPU-based fluid simulation supports around 3,000 particles in real time without using a compute shader. Elijah Nicol’s SPH demo runs smoothly with around 1,000 particles using his single-threaded CPU implementation. By contrast, most NVIDIA Flex samples can simulate around 70,000 particles in real time using GPU acceleration. These benchmarks provide useful context for evaluating the scalability and performance of the current simulation.

Using the single-threaded CPU implementation, the current simulation is expected to handle at least 1,000 particles, and at least 20,000 particles in real time using the compute shader implementation.

5.1 Data charts

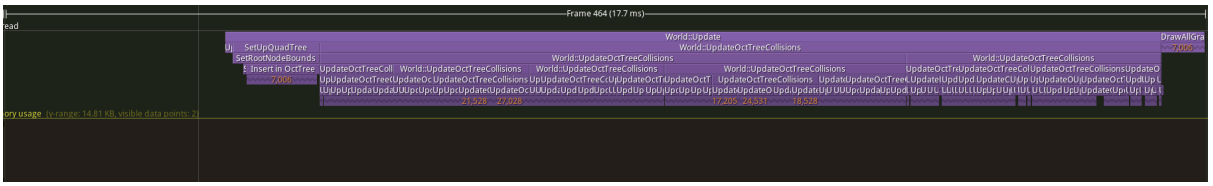

Figure 19: 3D engine without SPH (7'000 particles)

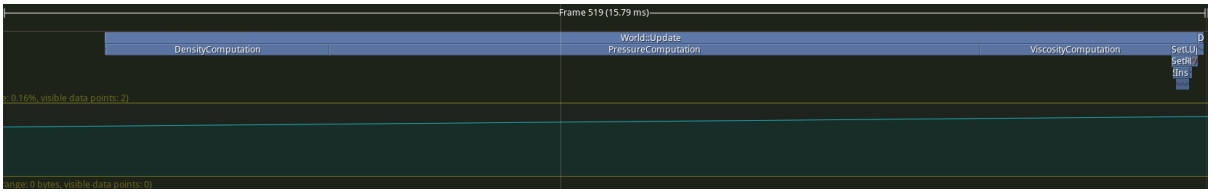

Figure 20: Naive SPH implementation (900 particles)

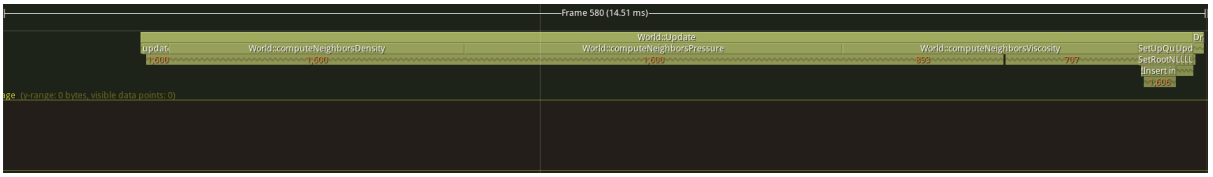

Figure 21: Spatial Hash Grid Optimisation (1600 particles)

	3D engine without SPH 7'000 particles	Naive SPH Implementation 900 particles	Spatial Hash Grid Optimisation 1'600 particles	Compute Shader Simulation 50'000 particles (CPU + GPU)
World Update	Mean: 13.96 ms Median: 14.26 ms α: 4.13 ms	Mean: 14.02 ms Median: 13.89 ms α: 462.85 µs	Mean: 14.13 ms Median: 13.69 ms α: 1.77 ms	Mean: 8.29 ms
Density	Not Implemented	Mean: 2.57 ms Median: 2.52 ms α: 135.94 µs	Mean: 4.05 ms Median: 3.9 ms α: 555.21 µs	Mean: 2.05 ms
Pressure	Not Implemented	Mean: 8.59 ms Median: 8.52 ms α: 346.95 µs	Mean: 5.04 ms Median:4.9 ms α: 642.38 µs	Mean: 2.29 ms
Viscosity	Not Implemented	Mean: 2.5 ms Median: 2.48 ms α: 118.82 µs	Mean: 3.97 ms Median: 3.83 ms α: 564.02 µs	Mean: 2.31
Grid	Not Implemented	Not Implemented	Mean: 351.18 µs Median: 341.13 µs α: 36.06 µs	Mean: 2.46 ms

5.2 Result Analysis

The performance of the SPH simulation was evaluated through successive implementation stages, each applying progressively more advanced optimisation techniques. Four major configurations were benchmarked: a baseline 3D engine without SPH, a naïve CPU-based SPH implementation, a spatially optimised version using a spatial hash grid, and a GPU-based compute shader implementation.

The initial configuration (Figure 17) can handle 7,000 physics particles without any SPH simulation. This version served as a performance baseline to isolate the cost of fluid dynamics computations. The mean world update time was 13.96 ms, with a median of 14.26 ms and a standard deviation (α) of 4.13 ms. This relatively high standard deviation indicates minor instability in frame processing time, likely caused by uneven CPU workload or external system processes.

The second configuration (Figure 18) introduced a naïve SPH solver using brute-force neighbor searches with a huge complexity $3*(O(n^2))$. Due to this inefficiency, the simulation could only support 900 particles within a similar frame time budget. The standard deviation (α) of 462.85 μs, showed more stable but constrained performance. Among the SPH computation phases, pressure computation was the most expensive, averaging 8.59 ms, followed by density at 2.57 ms, and viscosity at 2.5 ms. This distribution highlights the computational cost of pressure force calculations in unoptimised implementations.

The third configuration (Figure 19), a spatial hash grid was integrated to accelerate neighbor searches. This optimisation reduced the computational complexity of the solver, resulting in almost doubling the particle count (1,600 particles). The average time for density, pressure, and viscosity computations was 4.05 ms, 5.04 ms, and 3.97 ms respectively. In addition, the grid construction itself only required 351.18 μs, confirming its efficiency and negligible impact on the total update time. The performance gain demonstrates the effectiveness of grid-based spatial partitioning for fluid simulation.

The final implementation of the simulation leverages a compute shader to offload the Smoothed Particle Hydrodynamics (SPH) computations to the GPU, resulting in a substantial increase in the number of particles that can be simulated in real time. Performance profiling was conducted with a configuration of 50,000 particles, representing a 31 times improvement over the previous CPU-based implementation.

While this performance gain demonstrates the potential of GPU parallelism for large-scale particle simulations, the actual number of particles that can be simulated is currently constrained by several factors specific to the project's configuration. These limitations include resource allocation caps, memory management overhead, thread group size constraints, and possible bottlenecks related to buffer synchronization and data transfer between CPU and GPU. Despite these constraints, since the world update takes only 8.29 ms to be processed the current setup clearly shows that the system could support significantly larger particle counts with further optimization or architectural adjustments.

5.3 Further development

Although the current implementation successfully ports the physics engine to 3D and demonstrates a basic fluid simulation using Smoothed Particle Hydrodynamics (SPH), there are several areas that could be developed further. These could enhance performance, improve physical accuracy or extend the engine’s capabilities to encompass real-world and game-oriented use cases.

One key area for improvement is the physical realism of the simulation. While the current SPH formulation's basic pressure and viscosity model is suitable for demonstration purposes, it limits the range and stability of fluid behaviours. Implementing vorticity confinement would help to preserve the swirling motions that naturally arise in turbulent flows, but which tend to dissipate in standard SPH simulations. Similarly, adding surface tension models would enable the simulation to better represent small-scale fluid phenomena, such as droplets, cohesion and fragmentation, which are essential for achieving more visually convincing results.

Another critical improvement would be to enforce incompressibility more rigorously. Although the current method permits some density variation, more advanced techniques such as Predictive-Corrective Incompressible SPH (PCISPH) and Divergence-Free SPH (DFSPH) could be adopted to minimise pressure fluctuations and enable larger, more stable time steps. These methods have been shown to greatly enhance the accuracy of fluid simulations, particularly in scenarios involving tight constraints or confined volumes.

Significant refinement of fluid-solid interactions could also be beneficial. Currently, solid boundaries are handled using simplified constraints that do not fully capture the complexity of fluid behaviour near solid surfaces. Handling solid-fluid interactions more accurately, including dynamic boundaries and pressure projection methods, would enable fluids to respond more realistically to obstacles and moving objects.

6. Conclusion

The aim of this Bachelor's project was to explore the challenges and limitations of achieving realistic, real-time water physics through Smoothed Particle Hydrodynamics (SPH) in a custom-made 3D physics engine. The goal was to analyse the computational demands, algorithmic complexities and architectural design decisions involved in simulating fluid behaviour in real time.

This exploration involved the development and progressive optimisation of a naive SPH implementation. Early development efforts were concentrated on simulating fluid behavior through a particle-based approach, applying standard SPH equations to model density, pressure, and viscosity forces. This approach provided a foundational understanding of how local interactions between particles can approximate the behaviour of a continuous fluid. However, the naive approach quickly revealed performance bottlenecks, primarily due to the brute-force neighbour search and CPU-based computations.

To address this issue, a spatial partitioning system that uses a hash grid was implemented, which significantly improved neighbour retrieval efficiency. Subsequently, the core SPH computations were transferred to a compute shader for GPU acceleration, using parallelism to enhance simulation speed while preserving visual and physical accuracy. While this shift improved real-time responsiveness, it also exposed additional complexities such as memory management between the CPU and GPU, precision issues and debugging limitations inherent to shader development.

Ultimately, the project demonstrated that, although SPH is a powerful and intuitive method for simulating fluid dynamics, it presents significant computational and architectural challenges, particularly when aiming for real-time performance. The trade-off between physical accuracy and simulation speed remains a key limitation. Careful balancing is required between factors such as neighbour search complexity, time-step stability, solver accuracy and resource constraints.

Realistic water physics in real-time environments, particularly in games and interactive simulations, often necessitate compromises. While SPH strikes a good balance between realism and controllability, further optimisations, hybrid models or level-of-detail techniques are necessary to scale the simulation to more complex or large-scale scenarios.

In conclusion, this project has provided a technical and conceptual understanding of the complexities involved in real-time fluid simulation using SPH. It has highlighted the mathematical and computational demands, as well as the architectural challenges, of integrating such a system into a homemade 3D physics engine. A key takeaway is that incorporating fluid simulation into an existing physics system poses significant challenges. Many of the necessary data structures, memory layouts and update loops are not designed to handle the high levels of parallelism and precision required by a fluid simulation system. To achieve optimal performance and maintainability, it is often wiser to design the engine from the ground up with fluid simulation in mind, preparing the architecture, data flow and modular systems accordingly.

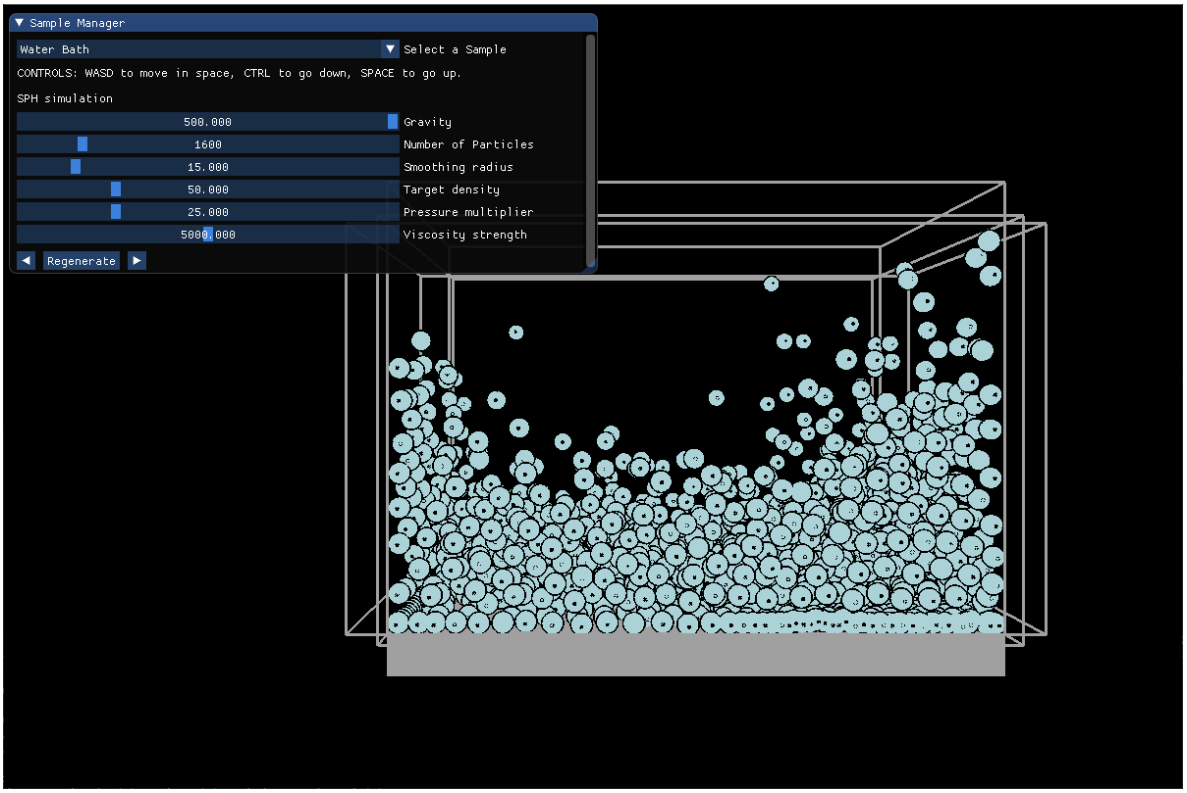

Figure 22: Final state of the CPU-based implementation (1600 particles)

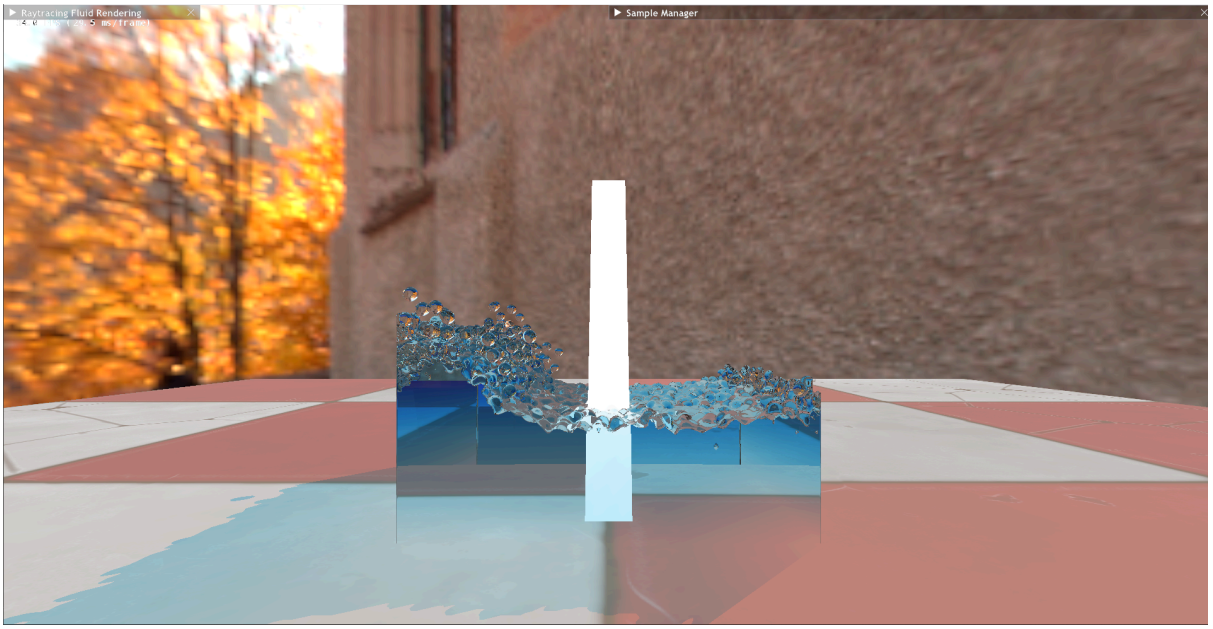

Figure 23: Final state of the GPU compute shader implementation (40'000 particles)

Sources

Bibliography

Bender, J., & Koschier, D. (2017). Divergence-free SPH for incompressible and viscous fluids.
https://animation.rwth-aachen.de/media/papers/2017-TVCG-ViscousDFSPH.pdf

Gingold, R. A., & Monaghan, J. J. (1977). Smoothed particle hydrodynamics: Theory and application to non-spherical stars.
https://articles.adsabs.harvard.edu/pdf/1977MNRAS.181..375G

Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A., & Teschner, M. (2014). SPH fluids in computer graphics.
https://cg.informatik.uni-freiburg.de/publications/2014_EG_SPH_STAR.pdf

Koschier, D., & Bender, J. (2017, July). Density maps for improved SPH boundary handling.
https://animation.rwth-aachen.de/media/papers/kb17.pdf

Macklin, M., Müller, M., Chentanez, N., & Kim, T. (2014). Unified particle physics for real-time applications. ACM Transactions On Graphics,33(4),1-12.
https://doi.org/10.1145/2601097.2601152

Macklin, M., & Müller, M. (2013). Position based fluids.
https://mmacklin.com/pbf_sig_preprint.pdf

Monaghan, J. J. (1992). Smoothed particle hydrodynamics. Annual Review Of Astronomy And Astrophysics, 30(1), 543-574.
https://doi.org/10.1146/annurev.aa.30.090192.002551

Müller, M., Charypar, D., & Gross, M. (2003). Particle-based fluid simulation for interactive applications.
https://matthias-research.github.io/pages/publications/sca03.pdf

Solenthaler, B., Bucher, P., Chentanez, N., Müller, M., & Gross, M. (2011). SPH based shallow water simulation.
https://matthias-research.github.io/pages/publications/SPHShallow.pdf

Solenthaler, B., & Pajarola, R. (2009). Predictive-Corrective Incompressible SPH
https://www.ifi.uzh.ch/dam/jcr:ffffffff-daa5-74d6-0000-00005a4f5c99/pcisph.pdf

Stam, J. (1999, August). Stable fluids.
https://pages.cs.wisc.edu/~chaol/data/cs777/stam-stable_fluids.pdf

Wikipedia contributors. (2025, 15 juillet). Finite element method. Wikipedia. https://en.wikipedia.org/wiki/Finite_element_method#:~:text=Finite%20element%20method%20(FEM)%20is,mass%20transport%2C%20and%20electromagnetic%20potential

Wikipedia contributors. (2025a, juin 26). Poisson's equation. Wikipedia. https://en.wikipedia.org/wiki/Poisson%27s_equation

Webography

Doyle, C. (2023) : Siggraph 2023 course - Real-time simulation and rendering of large-scale 3D liquid on GPU for games. https://www.youtube.com/watch?v=EF4KueUBDfM.

InteractiveComputerGraphics. (s. d.). GitHub - InteractiveComputerGraphics/SPlisHSPlasH : SPlisHSPlasH is an open-source library for the physically-based simulation of fluids. GitHub. https://github.com/InteractiveComputerGraphics/SPlisHSPlasH

James, D. (s. d.). Stanford CS348C : Prog. Assignment # 1 : Position based fluids. https://graphics.stanford.edu/courses/cs348c-20-winter/HW_PBF_Houdini/index.html

JangaFX - LiquiGen : Real-time liquid simulation tool. (s. d.). https://jangafx.com/software/liquigen

Lague, S. (2023). Coding Adventure: Simulating Fluids https://www.youtube.com/watch?v=rSKMYc1CQHE

Lijenicol. (s. d.). GitHub - lijenicol/SPH-Fluid-Simulator : An interactive fluid simulation software that uses Smoothed-Particle Hydrodynamics. GitHub. https://github.com/lijenicol/SPH-Fluid-Simulator

Müller, M. Publications. (s. d.). https://matthias-research.github.io/pages/publications/publications.html

NVIDIA. (n.d.). CUDA C Programming Guide. https://docs.nvidia.com/cuda/cuda-c-programming-guide/

NVIDIA. (2012). NVIDIA PhysX SDK Guide (Version 3.3.4). https://documentation.help/NVIDIA-PhysX-SDK-Guide/documentation.pdf

NVIDIA WaveWorks (s. d.) : NVIDIA Developer. https://developer.nvidia.com/waveworks

NVIDIA. Welcome to PhysX — physx 5.1.1 documentation. (s. d.). https://nvidia-omniverse.github.io/PhysX/physx/5.1.1/index.html

Publications | Jos Stam. (s. d.). Jos Stam. https://www.josstam.com/publications

Sommavilla, S. Basics of Smoothed Particle Hydrodynamics (SPH) Explained. (s. d.). https://www.divecae.com/resources/sph-basics

Vassvik, M. Realtime fluid simulation : projection. (s. d.). Gist. https://gist.github.com/vassvik/f06a453c18eae03a9ad4dc8cc011d2dc

Figures

Figure 1 : Smoothed-particle hydrodynamics, 2018. Schematic view of an SPH convolution, https://en.wikipedia.org/wiki/Smoothed-particle_hydrodynamics, page 9
Extracted 11/07/2025

Figure 2 : Sommavilla, S. 2020. Representation of pressure curve, https://www.divecae.com/resources/sph-basics, page 9
Extracted 11/07/2025

Figure 3 : Nvidia Flex, High viscosity sample,
Screenshot made by the author from the Nvidia Flex samples, page 10
Extracted 20/06/2025