Real-time Multi-phase Fluid Simulation

Cameron Pearson - 40530119

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BSc (Hons) Computing

School of Computing, Engineering & The Built Environment
April 2025

Authorship Declaration

I, **Cameron Pearson**, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed: _____ Date: _____

Matriculation no: _____

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

This dissertation investigates real-time multi-phase fluid simulation using Smoothed Particle Hydrodynamics (SPH), emphasising computational efficiency and visual realism. Multi-phase interactions, such as oil and water, utilising GPU acceleration via compute shaders in Unity. This project implements an efficient real-time SPH simulation capable of simulating multi-phase fluid interactions in real time.

The study looks into fluid simulation methodologies, highlighting the advantages of SPH for handling free-surface and dynamic fluid interactions. Optimisation strategies, including spatial partitioning and adaptive time-stepping, which enable simulations involving up to 100,000 particles to maintain interactive frame rates.

Implementation details cover collision detection, boundary handling, and realistic rendering methods to ensure visually accurate fluid s. Performance and visual quality tests evaluate the stability and effectiveness of multi-phase interactions.

Results show that optimised SPH simulations can accurately represent multi-phase fluids in real-time, suitable for applications in gaming, virtual reality, and scientific visualisation. This dissertation contributes to the development of efficient and realistic fluid simulation techniques for multiphase fluids in real time environments.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my initial supervisor, Dr Babis Koniaris, for his support in shaping the early stages of this project. His guidance was invaluable in helping me develop the core ideas and begin my literature review.

I would like to thank my supervisor, Dr Ian Donald, for his continuous support, encouragement, and constructive feedback throughout the duration of this project. His insight and guidance helped in keeping me progressing with my work.

Introduction

Motivation

The simulation of fluids has been a major challenge in both scientific research and entertainment industries, particularly in fields like game development, animation, and virtual reality. The demand for realistic and interactive fluid simulations that can be simulated in real time has grown significantly with advancements in computing hardware, allowing real-time simulations to become feasible. This project focuses on the real-time simulation of multi-phase fluids, such as oil and water and their interactions, using Smoothed Particle Hydrodynamics (SPH), with the ultimate goal of balancing computational efficiency with visual accuracy.

Research Context and Background

Fluid dynamics is a well-established field within computational physics, with modern solutions being highly optimised and efficient. Fluid simulation has been a topic of research for several decades, with early implementations dating back to the 1950s. The mathematical foundations of modern fluid simulation are rooted in the Navier-Stokes equations, first formulated in 1822(Darrigo 2002). Over the past few decades, techniques such as Computational Fluid Dynamics (CFD) and SPH have been developed to simulate fluid motion Monaghan (2005) . While these methods have proven effective in various domains, simulating multi-phase fluids in real time presents unique challenges, particularly in balancing computational performance with realistic interaction between different fluid types.

Aims and Objectives

The primary aim of this project is to implement a real-time multi-phase fluid simulation using Smoothed Particle Hydrodynamics (SPH) and GPU acceleration. The project investigates existing methods in fluid simulation, focusing on techniques that enable ef-

ficient real-time simulation and support multi-phase fluid interactions. By using GPU parallelisation, the project aims to significantly enhance computational performance while maintaining high-quality visualisation of complex fluids. The project evaluates the feasibility and effectiveness of these methods in real-world scenarios, where both accuracy and performance are critical.

The project can be split into two major parts, an exploration of the theoretical foundations behind fluid simulations and the practical implementation of those theories in a GPU-accelerated environment. Through an investigation into the mathematical models, physics, and computational techniques governing fluid dynamics, the project provides a detailed understanding of how multi-phase fluids interact and how these interactions can be simulated in real time. It also examines how modern GPU architectures can be used to optimise the performance of these simulations, ensuring that fluids are rendered interactively without compromising accuracy.

The objectives of this project are:

- Understanding and implementing the mathematical foundations of SPH.
- Investigating both foundational and modern fluid simulation methods.
- Implementing GPU-accelerated optimisations, including grid-based neighbour searches, adaptive time-stepping, and efficient memory management.
- Researching methods for simulating multi-phase fluid interactions.
- Implementing a particle-based SPH model that can simulate multi-phase fluids.
- Visualising the fluid interactions using a real-time rendering system with compute shaders
- Evaluating the performance of the simulation in terms of computational speed and visual quality.

By achieving these objectives, this project not only provides a detailed understanding of multi-phase fluid simulation but also demonstrates the practicality of real-time implementation using modern GPU hardware. The findings from this research have potential applications in gaming, virtual environments, scientific visualisation, and engineering simulations where high-quality fluid dynamics are crucial.

Scope of the Project

This project focuses on the simulation of multi-phase fluids using Smoothed Particle Hydrodynamics (SPH), with a focus on real-time performance for applications in gaming and interactive simulations Dharma & Kistijantoro (2017). The primary objective is to create a fluid simulation that is both computationally efficient and visually convincing, allowing for interactions between different fluid phases, such as water and oil, in real-time environments. Real-time performance is essential in these contexts, as the simulation

must respond in real-time to user inputs or changing conditions without significant delays or loss of visual fidelity.

While scientific accuracy plays a role, the goal is to balance precision and performance, recognising that the demands of interactive environments, such as video games or virtual reality, differ from those of highly detailed scientific simulations that require large amounts of time to generate seconds of simulation time. This project does not aim to replicate the precision required in industrial or scientific simulations, such as those used in computational fluid dynamics (CFD) for engineering or climate modelling Blazek (2015). Instead, the focus is on achieving fluid that is visually realistic, offering smooth, natural interactions between multiple fluid types, but at a computational cost low enough to maintain high frame rates in real time.

The project is designed to run on modern GPUs using Unity's compute shader pipeline, leveraging parallelism for real-time fluid dynamics. The multi-phase fluid interactions demonstrated in this project will be applicable to game physics engines, interactive visual effects, or virtual environments, where performance is critical. By carefully balancing computational efficiency with realistic , the project aims to show that visually convincing fluid simulations can be achieved in a GPU-accelerated real-time environment.

Structure of the Dissertation

This dissertation is structured as follows:

- **Chapter 2: Fluid Simulation** – An in-depth exploration of the foundations of fluid simulation, including key physics concepts, the historical development of the field, and a comparison of Eulerian and Lagrangian methods, with emphasis on Smoothed Particle Hydrodynamics (SPH) and its variants.
- **Chapter 3: Real-time Methods** – A discussion of the techniques and challenges associated with achieving real-time performance, including GPU acceleration, optimisation strategies for SPH, and practical applications of fluid simulation in interactive environments such as games.
- **Chapter 4: Multi-phase Fluids** – A technical overview of multi-phase fluid simulation using SPH, addressing challenges such as interfacial dynamics, pressure stability, surface tension modelling, and boundary interactions, with reference to relevant literature and current best practices.
- **Chapter 5: Methodology** – Details the functional and non-functional requirements, the design principles behind the system architecture, the data structures and algorithms employed, and the rationale for using GPU compute shaders within Unity for the simulation.
- **Chapter 6: Implementation, Testing, and Evaluation** – Provides a thorough breakdown of the implementation process, including particle generation, compute shader logic, and real-time rendering techniques. It also includes testing outcomes

that validate the simulation's physical accuracy and real-time performance across a range of scenarios.

- **Chapter 7: Conclusion and Future Work** – Evaluates the success of the project against its original objectives, reflects on its achievements and limitations, and outlines recommendations for future improvements and extensions, including practical application areas.
- **Appendices** – Contains supporting materials including code snippets, validation test cases, personal development reflections, and project management documentation.

Fluid Simulation

Introduction

This chapter discusses the principles and computational techniques used in fluid simulation, looking into both the theoretical and practical aspects. Starting with an overview of fluid simulation, looking into the roles of fluid simulation in applications such as gaming, science, and engineering. Then, the key physics concepts will be discussed, looking into the mechanics and equations used and the differences between Eulerian and Lagrangian approaches, each with advantages and disadvantages for simulation quality and efficiency.

This chapter then looks into Computational Fluid Dynamics (CFD) looking at the development of fluid simulation and the Navier-Stokes equation, which is key to fluid simulation but complex to solve in real-time. Then looking into Smoothed Particle Hydrodynamics (SPH) as an important technique when using the Lagrangian method for particle-based simulations. Then finally, I will look at grid-based techniques for optimising fluid simulation, discussing fixed grid and adaptive grid approaches by looking at pros and cons for both.

Overview of Fluid Simulation

Fluid simulation is the process of replicating the motion and behaviour of fluids in a computational environment, this can include liquids, gases, or any other materials that have fluid-like behaviour. It is a critical tool in areas such as scientific research, visual effects and gaming, where simulating water, smoke and other fluids can add realism to virtual environments.

Fluid simulation is governed by the Navier-Stokes equations, which describe the motion of fluid Batchelor (2000). These equations are used for calculating fluid properties like pressure, velocity, and viscosity, offering a way to accurately simulate a variety of fluid behaviours. A challenge this creates is solving the full Navier-Stokes equations in real-time, this is very computationally expensive, especially for high quality larger simulations, which has led to the creation of more practical methods.

One of the most prominent methods in real-time simulations is Smoothed Particle Hydrodynamics (SPH), a particle-based approach that models fluids using individual particles rather than a continuous field Monaghan (2005). This technique is well-suited for real-time applications due to its flexibility and ability to simulate free-surface flows efficiently. By tracking each particle's position, velocity, and density, SPH simplifies the computation, making it easier to simulate complex behaviours like fluid mixing or interactions with solid boundaries.

Importance and Applications of Fluid Simulation

Fluid simulation plays a major role in a wide range of industries, such as scientific research, engineering and entertainment(Monaghan 2005). In scientific fields, it helps give insight into things such as naturally occurring events such as ocean currents and weather patterns, which can help in predicting these events and potentially saving lives in extreme situations. It can also help with medical research by supporting research on blood flow and respiratory systems in the human body. Entertainment applications are a major use for fluid simulation. Creating visually appealing environments by accurately replicating water, smoke, fire or any other fluid like elements can immerse viewers in the virtual world that has been created. Recent projects that take advantage of modern fluid simulation, such as the major box office success Avatar: The Way of Water, are perfect examples of how accurate and high fidelity fluid simulation can help create beautiful looking immersive environments Tessendorf (2004).

Engineering applications of fluid simulation allows for the in depth testing of systems that have interactions with anything that has fluid-like properties. Greatly reducing the need to create real world prototypes for testing purposes, which can result in reduced development costs for projects. This could include a range of applications from simulating the aerodynamics of a windmill to simulating fuel injection in engines.

Historical Development of Fluid Simulation

The history of fluid simulation dates back to when classic physics was first being discovered. Frst being developed in the 19th century with the establishment of fluid dynamics principles and the Navier-Stokes equations Darrigo (2002). These described the movement of fluids and still remain as the foundation to modern fluid simulations.

In the late 20th century the growth of computer graphics began to skyrocket, and the demand for realistic computational fluid simulation grew with it. As a result of this demand, researchers developed algorithms that balanced accuracy with efficiency. From this began the evolution of Computational Fluid Dynamics (CFD). With the first practical development of CFD software being developed in the 1970's by companies such as McDonnell Douglas and IBM and in the 1980's the first development of visualisation tools allowing engineers to see the results of simulations in 3D(Resolved Analytics n.d.). This allowed for fluid to be simulated under a variety of different conditions, but only after

taking time to render as real-time simulations remained too computationally complex to be simulated at this time.

The 21st century has seen greater developments in optimisations for fluid simulation, beginning to allow for real-time and interactive fluid simulation. Methods that take advantage of GPU processing power began to be developed, allowing simulations to be run faster and on a much larger scale than ever before(Huang et al. 2023). Developments in hardware accelerated methods such as SPH and grid based approaches began to be frequently implemented in a wider variety of applications such as game engines and scientific visualisation.

More recently, research has become focused on hybrid techniques with researchers combining multiple approaches to fluid simulation, such as combing grid based methods with the Lagrangian method and even implementing deep learning to create a balance between precision and computational speed. These newly developed techniques help push the boundary of what is possible in the realm of fluid simulation, continuing to make larger real-time fluid simulations possible even in the most complex scenarios.

Computational Fluid Dynamics (CFD)

Computational Fluid Dynamics (CFD) focuses on the simulation of fluid by solving complex mathematical equations that describe fluid flow, such as the Navier-Stokes equation. CFD is used in the field of engineering to give detailed and accurate fluid simulations when interacting with solid boundaries. It works by splitting up the fluid space into a mesh and then applying numerical methods to calculate the flow at every point on the mesh.

CFD is a key branch of fluid simulation and is fundamental for accurately predicting the flow of fluids for engineering and scientific purposes Ferziger & Perić (2002). It is crucial for any tasks that require accurate simulation of interactions with solids and complex flow patterns. CFD allows engineers to optimise their designs with a level of accuracy that is important for meeting safety and regulation standards.

The difference between CFD and general fluid simulation is that CFD focuses on accuracy providing detailed and accurate models, whereas the general use of fluid simulation focuses on a balance of visual quality and accuracy. Whilst they both share the fundamentals of fluid simulation by using the Navier-Stokes equations. General fluid simulations use methods such as Smoothed Particle Hydrodynamics (SPH) and/or grid-based methods to allow for simulations to be run faster and on larger scales at the cost of the simulation accuracy.

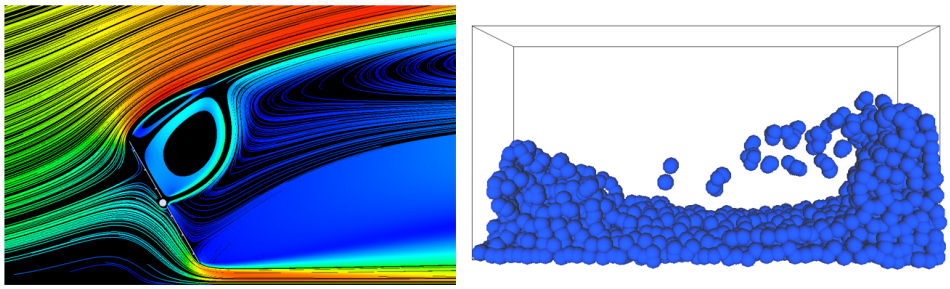

Figure 2.1: CFD (left) vs SPH (right)

Physics

Newtonian Mechanics

Newtonian Mechanics are a foundational part of any physics based simulation Batchelor (2000). It describes the relationship between an object's mass and the forces acting on it. The primary equation is Newton's second law of motion which is as follows:

$$F = ma \tag{2.1}$$

Where: F is the force, m is mass, a is acceleration

This equation shows the objects acceleration is dependant on the force applied and its mass. In physics simulations, the equation is usually broken down into time steps, allowing for changes in the object's speed and position to be calculated over time.

Newton's third law also plays a key role in physics simulation, it states that every action has an equal and opposite reaction:

$$\mathbf{F}_{A \text{ on } B} = -\mathbf{F}_{B \text{ on } A} \tag{2.2}$$

This shows that if two objects collide, the force object A exerts on object B is equal in magnitude but opposite in direction to the force object B exerts on object A. This ensures that the total momentum of both the objects remains the same, but their individual velocities change according to their masses and forces. This allows for accurate energy transfer between objects, which is essential for realistic physics simulations.

Navier-Stokes Equations

The Navier–Stokes equations are a set of equations that describe fluid motion Ferziger & Perić (2002). They include the continuity equation for mass conservation and the momentum equation, which applies Newton's second law to fluids. These work together

to simulate how fluids behave.

Continuity Equation:

The continuity Equation makes sure that mass is conserved, it is expressed as follows:

$$\nabla \cdot \vec{u} = 0 \tag{2.3}$$

Where $\vec{u}$ is the velocity vector. This ensures that if a mass of fluid moves around, changes shape or does anything, that if no mass is added or removed then its mass is conserved.

Momentum Equation

The momentum equation describes the velocity changes caused by both internal and external forces, it is expressed as follows:

$$\rho \left(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} \right) = -\nabla p + \mu \nabla^2 \vec{u} + \vec{f} \tag{2.4}$$

Where: ρ is the fluid density, $\vec{u}$ is the velocity vector, p is the pressure, μ is the dynamic viscosity, $\vec{f}$ represents external forces (such as gravity).

This equation describes the motion of fluid. The left side represents the fluids acceleration showing that the fluids velocity changes over space and time. The right side represents the forces acting on the fluid, which influence its movement.

The Navier-Stokes equations are a foundational part of fluid simulation Batchelor (2000). Using these equations allows for realistic modelling of any fluid-like entity by describing how velocity, pressure and external forces influence fluids. However, the equations are still not fully understood, their complexity means that when simulating fluids numerical approximations are sometimes required. For unsolved reasons when using the Navier-Stokes equations in three dimension some solutions are not able to be solved. This is one of the Clay Millennium Problems due to the equations' inconsistencies Tianren (2015).

Lagrangian and Eulerian Methods

In fluid simulation, there are two primary computational approaches: the *Lagrangian method* and the *Eulerian method*. These approaches differ fundamentally in how they conceptualise and track the movement of fluids over time.

Lagrangian Method

The Lagrangian approach tracks individual fluid particles as they move through space and time. This method focuses on following the trajectory of each particle, allowing the simulation to account for interactions between particles directly. The Smoothed Particle Hydrodynamics (SPH) technique, a widely used particle-based simulation method, is an example of the Lagrangian approach.

Key Characteristics

- Particle-centric: Each particle represents a small volume of fluid, carrying properties like mass, velocity, and density.
- High adaptability: Well-suited for problems involving free surfaces, interfaces (e.g., multi-phase fluids), and large deformations.
- Efficiency in collision handling: By using kernel functions, interactions between particles within a certain radius are computed, reducing the computational cost compared to global calculations.

Advantages

- Handles complex fluid interfaces and free surfaces effectively.
- Easily incorporates physical interactions like surface tension and viscosity.
- Suitable for simulations where detailed particle-level interaction is critical, such as multiphase flows.

Disadvantages

- Computationally intensive for large-scale simulations with millions of particles.
- Requires efficient neighbour-search algorithms to maintain performance.

Eulerian Method

In contrast, the Eulerian approach observes fluid properties at fixed points in a spatial grid rather than tracking individual particles. This grid-based method divides the fluid domain into a structured or unstructured mesh and solves the governing equations (e.g., Navier-Stokes equations) at each grid point.

Key Characteristics

- Grid-based: The fluid's velocity, pressure, and density are computed at discrete locations in space.
- Strong in numerical stability: Often employs advanced numerical schemes to ensure the stability and accuracy of simulations, even under high computational loads.

Advantages

- Highly efficient for simulating large-scale flows, especially when fine details at the particle level are unnecessary.
- Avoids the complexity of tracking individual particles.
- Integrates seamlessly with Computational Fluid Dynamics (CFD) methods for industrial applications.

Disadvantages

- Poor handling of free surfaces and interface dynamics without additional techniques like Volume of Fluid (VoF) or Level Set methods.
- Computational cost increases significantly with grid refinement, especially in three-dimensional simulations.

Comparison

Feature	Lagrangian Method	Eulerian Method
Approach	Particle-based	Grid-based
Data Representation	Particles	Fixed grid
Interface Handling	Natural (good for free surfaces)	Requires special techniques
Computational Scalability	Limited for large systems	Scalable with grid-based optimisations
Application	Multiphase flows, granular materials	Large-scale industrial fluid flows

Table 2.1: Comparison of Lagrangian and Eulerian Methods

Figure 2.2 shows the conceptual difference between Eulerian and Lagrangian methods. In the Eulerian approach, the simulation observes how fluid properties change at fixed positions in space, while in the Lagrangian approach, the simulation follows individual fluid elements as they move, effectively capturing their trajectory and deformation over time.

Both methods have their strengths and limitations. In practice, *hybrid methods* that combine aspects of Lagrangian and Eulerian techniques are increasingly used to leverage the advantages of both approaches Ihmsen et al. (2014). For example, coupling SPH with grid-based CFD methods can provide a balance between computational efficiency and detailed interaction modelling.

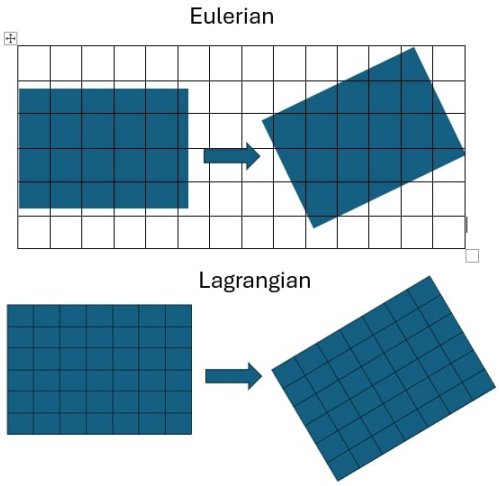

Figure 2.2: Comparison of Lagrangian and Eulerian Methods

Smoothed Particle Hydrodynamics (SPH)

Fundamentals of Particle Systems

Particle systems represent a versatile simulation methodology wherein objects, such as fluids, gases, and granular materials, are modelled as collections of discrete particles Monaghan (2005). These particles interact with one another and their environment, collectively simulating the behaviour of the material they represent. Each particle carries intrinsic properties such as position, velocity, mass, temperature, and density, which change over time according to physical laws.

Particle systems are particularly useful in simulating dynamic and complex situations where continuous models might struggle to represent small-scale or irregular behaviours effectively. By approximating a fluid or other material into particles, simulations can more easily accommodate irregular geometries, free surfaces, and interactions with solid boundaries.

Advantages of Particle Systems:

- Adaptability: Particle systems are well-suited for phenomena with large deformations, such as splashes or smoke clouds, where traditional grid-based approaches may fail.
- Detailed Interactions: Inter-particle interactions allow for accurate modelling of cohesion, repulsion, and other forces that govern fluid and gas s.
- Scalability: Particle systems can range from a few thousand particles for visual effects to millions of particles for high-resolution simulations.

Challenges in Particle Systems: Despite their versatility, particle systems pose unique computational challenges. Maintaining stable interactions requires careful tuning of particle properties and simulation parameters. Furthermore, as the number of particles increases, computational costs can grow significantly, requiring the use of efficient algorithms and hardware acceleration.

Particle systems form the foundation for Smoothed Particle Hydrodynamics (SPH), a method extensively employed in both research and industry to simulate fluids with free surfaces, complex interfaces, and dynamic interactions.

SPH Methodology and Applications

Smoothed Particle Hydrodynamics (SPH) is a computational method used to simulate fluid flows by treating fluids as collections of discrete particles(Monaghan 2005). Originally developed for astrophysical simulations, SPH has since become a key technique in fluid dynamics due to its adaptability and efficiency.

In SPH, the physical properties of fluids—such as density, pressure, and velocity—are computed at each particle's location using smoothing kernels. These kernels aggregate the influence of neighbouring particles within a defined radius, ensuring that the simulation captures localised fluid behaviours accurately.

Core Steps in SPH:

1. Initialisation: Define the initial positions, velocities, and properties of particles, along with boundary conditions.
2. Neighbour Search: Identify particles within a smoothing radius using spatial data structures like k-d trees or hash grids for computational efficiency.
3. Kernel Function Application: Compute properties such as density and pressure by applying smoothing kernels to nearby particles.
4. Force Calculations: Evaluate forces acting on particles, including pressure gradients, viscosity, surface tension, and external forces like gravity.
5. Time Integration: Update particle properties using numerical methods, such as explicit Euler or Verlet integration, to simulate motion over time.

Advantages of SPH:

- Adaptability: Handles complex geometries, such as free surfaces and interfaces, naturally.
- Efficiency: Requires fewer computational resources compared to grid-based methods for free-surface flows.

- Versatility: Extensible to simulate multiphase fluids, granular materials, and elastic bodies.

Applications: SPH is widely applied across various domains: Liu & Liu (2010)

- Entertainment Industry: Simulating water, smoke, and explosions in movies and video games.
- Engineering: Modelling fluid-structure interactions in machinery and pipelines.
- Environmental Science: Predicting the of tsunamis, dam breaks, and sediment transport.
- Astrophysics: Simulating galaxy formation, star collisions, and planetary accretion.

Fixed Grid vs Adaptive Grid

Grid-based methods provide an alternative approach to particle systems in fluid simulation by approximating the fluid domain into a structured or unstructured grid. Fluid properties are computed at discrete grid points, enabling efficient solution of the governing equations of fluid motion, such as the Navier-Stokes equations.

Fixed Grid: A fixed grid maintains a uniform resolution throughout the simulation domain, regardless of the complexity or variability of the flow features.

- Advantages:
  - Simple to implement and computationally predictable.
  - Ideal for flows with uniform features and steady-state .
- Disadvantages:
  - Inefficient for simulations requiring high resolution in small regions, such as turbulent vortices or fluid interfaces.
  - Significant memory overhead for large domains due to uniform resolution.

Adaptive Grid: Adaptive grids dynamically refine or coarsen grid resolution based on simulation requirements, allocating computational resources to areas of interest.

- Advantages:
  - Allows high-resolution simulations in critical regions while reducing computational costs elsewhere.

  - Well-suited for phenomena with sharp gradients, such as shock waves or phase boundaries.

- Disadvantages:
  - Increased algorithmic complexity due to dynamic grid management.
  - Potential for numerical artefacts at grid refinement boundaries.

Applications: Grid-based methods are commonly used in large-scale industrial simulations, such as weather prediction, aerodynamics, and oceanography, where precision and scalability are paramount.

Hybrid Methods

Hybrid methods combine the strengths of particle-based and grid-based approaches to overcome the limitations of each. By integrating SPH with grid-based techniques, hybrid methods provide a balance between the adaptability of particle systems and the stability and precision of grid-based methods Prosperetti & Tryggvason (2009).

Features of Hybrid Methods:

- Coupling Techniques: Interpolation functions or interface algorithms are used to transfer data between particle and grid representations.
- Optimisation: Employs grid-based methods for bulk flow and particle-based techniques for detailed interactions, such as fluid-solid coupling.
- Scalability: Allows simulations to handle both large-scale domains and fine-scale details efficiently.

Advantages:

- Provides high accuracy near interfaces, such as fluid boundaries, without excessive computational costs.
- Improves numerical stability in regions with high turbulence or sharp gradients.

Applications: Hybrid methods are particularly effective in:

- Multiphase Fluid Dynamics: Modelling interactions between immiscible fluids, such as oil and water.
- Environmental Modelling: Simulating complex interactions in ecosystems, such as sediment transport in rivers.

- Engineering Design: Analysing fluid flows in complex geometries, such as aircraft wings or turbine blades.

Real-time Methods

Introduction

This chapter explores the methods and strategies necessary for achieving real-time performance in fluid simulations. It examines key considerations such as computational efficiency, GPU acceleration, and balancing visual fidelity with performance. Additionally, the chapter delves into optimisation techniques for Smoothed Particle Hydrodynamics (SPH) and large-scale simulations, as well as parallelisation approaches to leverage modern hardware effectively. These discussions aim to provide a comprehensive understanding of the techniques required to simulate fluid dynamics in real-time environments.

Performance and Optimisation for Real-time Applications

Real-time fluid simulation demands efficient utilisation of computational resources to ensure interactive frame rates Müller et al. (2003). Key performance optimisation strategies includeMüller et al. (2003):

- GPU Acceleration: Leveraging the parallel processing capabilities of GPUs to perform high-volume computations, such as particle interaction calculations in Smoothed Particle Hydrodynamics (SPH) Hendra (2015).
- Spatial Partitioning: Using techniques like uniform grids or octrees to optimise neighbour searches, reducing computational overhead. Ihmsen et al. (2010)
- Adaptive Time-stepping: Dynamically adjusting the simulation time step based on fluid activity to maintain stability while minimising unnecessary computations Ihmsen et al. (2010).
- Data Compression: Applying methods such as culling particles outside the view range or reducing the detail of distant fluid regions to balance memory usage and computational demand Huang et al. (2023).

By employing these techniques, simulations can achieve the responsiveness required for applications like video games and virtual reality environments.

GPU Acceleration for SPH-based Simulations

GPU acceleration is a cornerstone of modern real-time fluid simulations, especially for particle-based methods like SPH Harada et al. (2007). GPUs excel in handling the parallel nature of SPH computations due to their many-core architecture Harris (2004). Specific optimisations include:

- Parallel Processing: Assigning threads to compute properties for individual particles, such as density, pressure, and forces, concurrently.
- Efficient Memory Access: Minimising latency by leveraging shared memory for neighbour lists and kernel computations, reducing the need for frequent global memory access.
- Load Balancing: Distributing workload evenly across GPU cores to maximise throughput and prevent bottlenecks.
- Frameworks: Utilising GPU programming frameworks like CUDA and OpenCL for high-performance kernel implementations tailored to fluid simulation requirements Harris (2004).

With these optimisations, GPU-accelerated SPH can simulate tens of thousands of particles at interactive frame rates.

Balancing Visual Fidelity and Computational Performance

One of the primary challenges in real-time fluid simulation is maintaining a balance between visual realism and computational efficiency Macklin & Müller (2013). Strategies to address this include:

- Level of Detail (LoD): Adjusting the simulation detail based on the camera's distance from the fluid, ensuring high fidelity where it matters most while conserving computational resources.
- Surface Reconstruction: Employing efficient algorithms to generate visually appealing surfaces from particle data, such as screen-space surface reconstruction or smoothing techniques.
- Simplified Dynamics: Using approximations for less critical aspects of the simulation, such as turbulence or minor fluid interactions, to reduce computational complexity.

- Real-time Rendering Enhancements: Incorporating post-processing effects like reflections, refractions, and shading to enhance visual quality without significantly impacting simulation performance.

By prioritising computational resources for high-impact visual features, simulations can deliver engaging and immersive experiences in real-time applications.

Hybrid Approaches

Hybrid simulation techniques that combine particle-based methods like SPH with grid-based methods provide additional flexibility in balancing performance and fidelity Solenthaler & Pajarola (2009). For instance:

- Coupling Techniques: Interpolating data between particles and grids to benefit from the strengths of both methods.
- Adaptive Refinement: Refining the resolution dynamically in regions with complex fluid interactions, such as splashes or phase boundaries Macklin & Müller (2013).
- Scalability: Employing grid methods for large-scale flow calculations while reserving particle-based techniques for detailed, localised interactions.

Hybrid methods ensure robust performance across diverse scenarios, enabling simulations to scale efficiently for both large and small fluid domains Prosperetti & Tryggvason (2009).

Optimisation Techniques in Fluid Simulation

Techniques to Improve SPH Performance

Smoothed Particle Hydrodynamics (SPH) is computationally intensive, particularly in real-time applications. To achieve optimal performance, several strategies can be used:

- Neighbour Search Optimisation: The most computationally expensive operation in SPH is determining neighbouring particles for interaction. Efficient spatial partitioning techniques, such as uniform grids or hash grids, significantly reduce the cost of these searches. Each particle only interacts with others within a defined radius, and these structures ensure rapid querying of potential neighbours.
  - *Uniform Grids:* Particles are grouped into fixed grid cells. Only particles in the same or adjacent cells are considered neighbours.
  - *k-d Trees:* Hierarchical structures like k-d trees partition space into regions, enabling efficient neighbour searches in large-scale simulations.

- Kernel Function Efficiency: SPH relies on smoothing kernel functions to interpolate particle properties. Using optimised kernel functions, such as the cubic spline kernel, balances computational cost with interpolation accuracy. Precomputing kernel values for discrete distances and storing them in lookup tables further accelerates computations.
- Adaptive Time-Stepping: Fixed time steps often result in unnecessary computations during periods of low activity. Adaptive time-stepping adjusts the simulation's time resolution based on particle velocities and interactions, ensuring stability without excessive computations.
- Parallel Processing on GPUs: The inherently parallel nature of SPH lends itself well to GPU acceleration. By assigning threads to individual particles or groups of particles, GPUs can compute densities, forces, and integration steps simultaneously. Frameworks such as CUDA and OpenCL are widely used to implement these optimisations.
- Density Correction Techniques: Real-time SPH simulations often face challenges with density fluctuations, leading to visual artefacts. Techniques like Density Invariant SPH (DISPH) and pressure projection help maintain consistent density distributions while reducing computational overhead.
- Dynamic Domain Decomposition: In large-scale simulations, dynamically splitting the domain into smaller regions and assigning each to different processing units optimises load balancing. Ensuring that each region has an equal computational load improves performance.

Optimisation for Large-scale Fluid Simulations

Simulating large-scale fluid interactions, such as in environmental or industrial applications, demands strategies that scale effectively with the number of particles Liu & Liu (2010):

- Multi-scale Modelling: Employing multi-scale approaches, allows high-resolution simulations in critical regions (e.g., near interfaces or obstacles) while using coarse resolutions elsewhere. Hybrid methods combining particle-based and grid-based approaches often achieve this balance.
- Hierarchical Data Structures: Techniques like octrees or hierarchical grids enable efficient management of particle data. These structures dynamically adapt to particle density, refining in areas of high complexity and coarsening in simpler regions.
- Optimised Memory Management: Memory bandwidth often becomes a bottleneck in large simulations. Techniques such as memory coalescing, caching frequently accessed data, and reducing memory duplication between CPU and GPU significantly enhance performance.

- Distributed Computing: Using distributed systems allows simulations to scale across multiple machines. Particles are assigned to different nodes, and communication protocols ensure consistency at domain boundaries. Distributed SPH frameworks, like parallel SPH implementations, leverage this approach.
- Advanced Integration Schemes: Large-scale simulations benefit from integration schemes that maintain stability without excessive computational cost. Semi-implicit methods, such as the Implicit Incompressible SPH (IISPH), allow larger time steps while preserving simulation fidelity.
- Load Balancing and Task Scheduling: Dynamic load balancing ensures that computational resources are used efficiently. Assigning tasks based on the complexity of the fluid interactions and rebalancing workloads during runtime prevents bottlenecks.

By leveraging these optimisation techniques, fluid simulations can achieve both scale and efficiency, enabling their application in diverse fields such as environmental modelling, virtual reality, and game design.

Fluid Simulation in Games

Fluid simulation has become a key component in modern game development, adding realism to water, smoke, fire, and other liquid interactions. However, the vast computational cost of physically accurate fluid simulations means that most games do not use true fluid simulation, instead relying on precomputed effects, shaders, and simplified approximations. Multi-phase fluids, in particular, are even rarer in real-time applications due to their increased complexity.

Games That Use Real Fluid Simulation A few advanced games and engines have incorporated real-time fluid simulation, often leveraging Smoothed Particle Hydrodynamics (SPH) or hybrid methods. Some notable examples include:

- Hunt: Showdown (CryEngine)
  - Uses SPH-based water simulation with real-time multi-phase interactions, where water responds dynamically to players and environmental factors.
  - Objects and characters interact with water realistically, producing ripples, splashes, and phase-dependent buoyancy effects.
  - Fluid movement is simulated at a particle level, allowing waves, foam, and bubbles to form based on real physics Engel (2012).
- Battlefield V (Frostbite Engine)
  - Implements real-time water interactions, allowing objects to float and waves to react to explosions dynamically.

  - Uses a hybrid particle-grid approach where large bodies of water are simulated using grids, but splashes and small-scale turbulence are handled with particles.
  - Includes dynamic wetting effects, where objects absorb water based on material properties.

- The Legend of Zelda: Breath of the Wild (Custom Engine)
  - Simulates basic buoyancy and current-driven water motion using SPH-like principles.
  - Features fire-water interactions, where water can extinguish fire dynamically based on particle-based effects.
  - Implements simplified air-water dynamics, where wind can push water and create ripples.

- Assassin's Creed IV: Black Flag (AnvilNext Engine)
  - One of the first mainstream games to integrate ocean simulation with dynamic waves using Fast Fourier Transform (FFT) wave modellingTessendorf (2004).
  - Uses a mix of precomputed water textures for distant waves while allowing ships to interact with simulated water near their hulls.

Figure 3.1: Fluid Simulation in Assassins Creed IV (Left) and Battlefield V (Right)

While these games make some use of real-time fluid physics, the complexity of full-scale multi-phase interactions (e.g., oil floating on water, sediment transport, or blood mixing with water) is still limited due to computational constraints Engel (2012).

Why Most Games Avoid True Fluid Simulation Despite advancements in hardware and simulation techniques, most games opt for approximations rather than full fluid physics due to:

1. Computational Cost
   - True Navier-Stokes-based fluid solvers are too expensive for real-time use, requiring significant CPU/GPU resources.

- Multi-phase simulations are even more demanding due to the added complexity of surface tension, viscosity, and density differences.

2. Performance Trade-offs
   - Games prioritise frame rate and responsiveness over strict physics accuracy.
   - A fully simulated fluid system might require millions of particles, which would be impractical for real-time rendering.
3. Simplified Alternatives
   - Instead of true 3D fluid physics, most games use:
     - Heightmaps for Water – A simple 2D representation of waves, preventing full 3D movement but offering fast, GPU-friendly rendering.
     - Shallow Water Equations – Used for approximating wave propagation without simulating the actual liquid.
     - Precomputed Fluid Effects – High-quality fluid simulations are pre-rendered and played as textures instead of being dynamically computed.
     - Particle-based Sprites – Fire, smoke, and splashes are often particle effects, rather than physics-based simulations.
4. Hybrid Approaches
   - Some games combine precomputed effects with limited real-time physics, blending performance and realism.
   - Example: *Uncharted 4* uses precomputed water splashes for distant waves but particle-based foam effects for up-close interactions.

Feature	True Fluid Simulation	Game Approximations
Realism	High	Medium
Computational Cost	Very High	Low
Surface Interaction	Full 3D movement	Mostly 2D
Performance Impact	Heavy on CPU/GPU	Minimal
Game Examples	Hunt: Showdown, Battlefield V	Assassin's Creed, Zelda BOTW

Table 3.1: Comparison of true fluid simulation vs. game approximations.

Comparison: True Fluid Simulation vs. Game Optimisations

Future of Fluid Simulation in Games With advancements in GPU computing, AI-assisted physics, and hybrid simulation models, real-time multi-phase fluid interactions may become more common. Technologies like NVIDIA FleX, Unreal Engine Niagara Fluids, and CryEngine SPH are pushing real-time fluids closer to scientific accuracy while keeping them computationally efficient Engel (2012).

However, for now, most games will continue using optimisations rather than full-fledged fluid physics to balance performance with visual fidelity.

Multi-phase Fluids

The simulation of multi-phase fluids presents additional complexities compared to single-phase simulations, requiring numerical techniques to accurately capture interactions between different fluid types. These interactions are crucial in various applications, including environmental modelling, engineering, and real-time visual effects. *Smoothed Particle Hydrodynamics* (SPH) offers a flexible approach to handling multi-phase dynamics, allowing for free-surface interactions and phase separations Monaghan (2005).

Multi-phase Fluid Dynamics Using SPH

Multi-phase fluid dynamics refers to the interaction of two or more immiscible fluids, such as water and air or oil and water, within a shared simulation domain. Unlike single-phase fluids, multi-phase interactions introduce challenges related to density contrasts, surface tension, viscosity differences, and phase transition effects. These complexities must be handled efficiently to ensure stable and physically accurate results Solenthaler & Pajarola (2009).

Density and Pressure Stability

A key challenge in multi-phase SPH simulations is maintaining stability in the presence of fluids with significantly different densities. Traditional SPH formulations struggle with high-density ratios, leading to numerical artefacts and pressure instabilities Liu & Liu (2010). Several techniques have been developed to address this:

- Density Renormalisation: Adjusting density formulations to mitigate inconsistencies in multi-phase simulations.
- Pressure Poisson Equation (PPE): Used in implicit SPH methods to ensure pressure continuity across fluid phases, improving numerical stability.
- Density Invariant SPH (DISPH): A modified approach that reduces pressure fluctuations and prevents numerical instabilities.

These approaches help improve the numerical stability and accuracy of SPH-based multi-phase fluid simulations.

Surface Tension Modelling

Surface tension is a crucial factor in multi-phase simulations, particularly in cases involving small-scale fluid interactions such as droplet formation and bubble dynamic Prosperetti & Tryggvason (2009). Common SPH techniques for handling surface tension include:

- Continuum Surface Force (CSF) Model: Applies a force proportional to the curvature of the interface to maintain distinct phase boundaries.
- Color Field Gradient Method: Uses a scalar field to track fluid interfaces, ensuring proper phase separation.
- Laplacian-based Surface Tension: Smooths particle positions based on curvature to prevent artificial diffusion Macklin & Müller (2013).

Interfacial Dynamics

The interface between different fluid phases requires careful handling to ensure that properties such as viscosity and density transition smoothly Suresh et al. (2019). Techniques include:

- Multi-phase Kernel Blending: Adjusts kernel weight functions dynamically based on the proximity of different phases.
- Artificial Repulsion Forces: Introduces additional forces at interfaces to prevent unwanted blending between phases.
- Adaptive Resolution Methods: Uses locally refined particle distributions to capture fine details at phase boundaries.

Proper interfacial treatment is necessary to ensure stability, especially in high-density-ratio simulations. Advanced approaches such as kernel blending and adaptive resolution methods help address these challenges Yan et al. (2009).

Challenges of Simulating Multi-phase Fluids

Real-time multi-phase fluid simulation faces several challenges due to computational constraints and the need for stable interactions:

- Computational Cost: Multi-phase SPH requires increased computation due to interfacial force calculations and pressure stabilisation.
- Instabilities at High Density Ratios: Air and water simulations, for example, exhibit extreme density differences, requiring specialised pressure solvers Solenthaler & Pajarola (2009).
- Loss of Detail in Visual Effects: In interactive settings, fluid fidelity is often reduced to maintain performance.

To address these challenges, several optimisations are implemented in game engines Harris (2004):

- Level of Detail (LoD) Techniques: Reducing simulation detail for distant fluids while maintaining high fidelity near the camera.
- Hybrid Approaches: Combining SPH with grid-based solvers to improve efficiency.
- Particle Reduction Methods: Using adaptive particle counts to prioritise high-detail areas.

Application of SPH to Multi-phase Fluids

SPH has proven to be highly adaptable for real-time multi-phase fluid effects in gaming and interactive simulations. Some notable applications include:

- Water-Air Interactions: Used to simulate ocean waves, splashes, and breaking surf effects in video games and films.
- Fire and Smoke: Multi-phase SPH methods simulate interactions between fuel, gas, and combustion byproducts in physics-based rendering.
- Lava and Molten Metal Flows: Used in both industrial simulations and visual effects to model fluid-like materials with variable viscosity.

SPH is widely used in real-time applications, particularly in game engines such as *Unity* and *Unreal Engine*, where it enhances realism while maintaining computational efficiency Harris (2004).

Collision Detection and Boundary Handling

Proper handling of collisions and boundaries is critical in multi-phase fluid simulations to ensure realistic interactions between fluids and solid objects.

Techniques for Handling Fluid-boundary Interactions

When multi-phase fluids interact with solid boundaries, specialised techniques are required to model reflection, adhesion, and absorption effects.

- Ghost Particles: Virtual particles introduced near solid boundaries enforce fluid-solid interaction constraints, ensuring smooth interactions Liu & Liu (2010).
- Dynamic Contact Angles: Determines the interaction between a fluid and a solid surface based on surface properties.
- Friction and Adhesion Models: Implements forces that dictate how fluids stick to or slide along solid surfaces, simulating effects such as droplet spreading.

Boundary handling techniques, such as ghost particles and friction models, play a critical role in stabilising interactions between multi-phase fluids and solid objects Liu & Nistor (2018).

SPH-specific Approaches for Collision Detection

Collision handling in SPH differs from traditional grid-based solvers due to its Lagrangian nature Ihmsen et al. (2010). Key SPH collision detection techniques include:

- Neighbour Search Optimisation: Efficiently determines particle interactions within a fluid domain using spatial partitioning structures like k-d trees and hash grids.
- Kernel-based Repulsion Forces: Introduces artificial pressure to prevent particle overlap and penetration into solid boundaries Taylor & Miller (2012).
- Rigid Body Coupling: Integrates solid objects into SPH simulations to model fluid-structure interactions, allowing for two-way coupling.

Fixed Grid Boundary Interactions

While SPH operates in a particle-based framework, hybrid approaches often use fixed grids to enforce boundary conditions. These methods offer improved stability for complex interactions:

- Eulerian-Lagrangian Hybrid Solvers: Combines SPH with grid-based pressure solvers to improve accuracy.
- Adaptive Grid Refinement: Uses higher resolution near fluid-solid interfaces for increased precision.
- Signed Distance Fields (SDFs): Represent boundaries as implicit surfaces for smoother interactions.

Conclusion

Multi-phase SPH simulations present unique challenges, including pressure stability, surface tension modelling, and efficient boundary handling. By employing advanced numerical methods and GPU acceleration, SPH-based multi-phase fluid simulations can achieve both accuracy and real-time performance, making them invaluable in applications ranging from gaming to scientific research.

Methodology

Requirements Analysis

The requirements analysis for this project identifies the essential functionalities and constraints necessary to achieve a robust and visually convincing real-time multi-phase fluid simulation using Smoothed Particle Hydrodynamics (SPH). The system must balance computational efficiency with visual realism while simulating complex fluid interactions within specified boundaries.

Functional Requirements

The functional requirements define the essential features and s that the simulation must exhibit to meet the project objectives effectively:

1. Multi-phase Fluid Interaction
   The simulation must accurately represent the interaction between different fluid phases, such as oil and water. The fluids should exhibit realistic s such as mixing, separation due to density differences, and the preservation of distinct boundaries between different fluid types.

2. Particle-based Simulation Using SPH
   The simulation will utilise the Smoothed Particle Hydrodynamics (SPH) method for fluid modelling. This particle-based approach will allow for the efficient simulation of free-surface flows and dynamic fluid interactions.

3. Real-time User Interaction
   Users must be able to observe fluid dynamically in real-time. The simulation should update fluid positions, velocities, and interactions continuously to reflect real-time changes within the system.

4. Boundary Enforcement
   The simulation should constrain fluid particles within a defined three-dimensional

boundary. Particles must not escape the boundary and should respond appropriately upon collision with the limits of the simulation space.

5. Visual Rendering
   The visual representation of the fluids should reflect realistic physical , including visually accurate rendering of density variations, surface tension effects, and colour differentiation for multiple fluid phases.

6. Initial Conditions Configuration
   The system must allow the definition of initial particle properties such as density, viscosity, mass, and velocity for different fluid phases. These initial conditions will be specified through predefined spawn areas.

Non-functional Requirements

The non-functional requirements define the system's performance and operational constraints to ensure that the simulation meets both user expectations and technical benchmarks:

1. Performance and Scalability

   - The simulation must maintain a minimum frame rate of 30 frames per second (FPS) to ensure smooth, real-time execution.
   - The system should efficiently handle a particle count of up to 100,000 particles without dropping below the target frame rate.

2. Visual Realism

   - The visual rendering must accurately represent fluid , including realistic surface tension, reflection, and refraction effects.
   - The simulation should incorporate advanced shading techniques to enhance the realism of multi-phase fluid interactions.

3. Computational Efficiency

   - The simulation will leverage GPU acceleration to ensure high-performance execution, using compute shaders to manage parallel processing effectively Hendra (2015).
   - Optimisations such as grid-based neighbour searches and adaptive time-stepping will be employed to reduce computational load.

4. Boundary Confinement
   All fluid particles must remain within a defined boundary throughout the simulation, ensuring accurate representation of container-based fluid dynamics.

5. Usability

   - The system must feature a user-friendly interface for adjusting simulation parameters, including particle count, fluid properties (density, viscosity), and visual effects.
   - Clear documentation will be provided to guide users through system functionality and configuration.

6. Portability
   The simulation will be developed for execution on standard desktop platforms with GPU support, primarily targeting Windows operating systems.

7. Reliability
   The system should remain stable under high computational loads and maintain consistent across multiple runs, even with high particle counts.

8. Maintainability
   The codebase will be modular and well-documented, allowing for future enhancements and the addition of new features, such as the integration of external forces or additional boundary conditions.

Software Design and Architecture

The design and architecture of the simulation system are focused on achieving high-performance, real-time multi-phase fluid simulation while leveraging Unity's built-in features and GPU acceleration capabilities. The software is structured to be modular, scalable, and optimised for real-time performance, allowing for efficient development and maintenance. The core components of the system focus on the integration of Smoothed Particle Hydrodynamics (SPH) for fluid simulation, real-time rendering, and GPU-based computation.

System Architecture Overview

The system architecture is designed around Unity's component-based framework, taking advantage of its physics engine, rendering pipeline, and GPU compute capabilities. The architecture is composed of several interconnected modules, each responsible for a specific aspect of the simulation:

1. Particle System Module
   This module handles the creation, initialisation, and lifecycle management of fluid particles using Unity's scripting environment. Particles are instantiated through custom scripts that allow for defining properties such as density, mass, viscosity, and initial velocity, based on the multi-phase nature of the simulation.

2. SPH Simulation Module
   This module is responsible for implementing the Smoothed Particle Hydrodynamics (SPH) algorithm. The SPH computation is offloaded to compute shaders, leveraging Unity's GPU acceleration capabilities for parallel processing of particle interactions Harada et al. (2007). This allows for the efficient handling of large-scale particle systems (up to 100,000 particles) while maintaining real-time performance.

3. Boundary and Collision Handling Module
   This module manages the confinement of particles within a predefined boundary using Unity's collision detection systems. It ensures that particles respect the simulation space's constraints and implements basic collision responses using Unity's physics engine.

4. Rendering and Visualisation Module
   Unity's rendering pipeline will be utilised for visually representing fluid particles with a high degree of realism. Custom shaders will handle the visualisation of fluid properties, such as reflection, refraction, and colour differentiation for different fluid phases. Particle colouration will help distinguish between fluid types and highlight multi-phase interactions.

5. User Interface and Parameter Configuration
   A simple UI will be developed to allow users to adjust simulation parameters dynamically, such as particle count, viscosity, density, and other physical properties. Unity's built-in UI system will handle this functionality.

6. Performance Monitoring
   Unity includes tools for monitoring frame rates and memory usage to ensure that the simulation remains within performance targets, particularly the 30 FPS threshold for real-time interaction.

Data Structures and Algorithms

The efficiency of the simulation relies on optimised data structures and algorithms that can manage the computational complexity of real-time fluid simulation. The following data structures and algorithms form the foundation of the simulation:

1. Particle Data Structure
   Each particle in the simulation is represented by a structured data type that includes the following attributes:

   - Position ($\vec{r}$)
   - Velocity ($\vec{v}$)
   - Acceleration ($\vec{a}$)
   - Density (ρ)
   - Pressure (p)
   - Mass (m)

- Viscosity (μ)
- Colour (for multi-phase visualisation)

2. Spatial Grid for Neighbour Search
   To optimise neighbour searching, a uniform grid-based spatial partitioning system is employed. The simulation space is divided into grid cells, and each particle is assigned to a cell based on its position. This reduces the computational complexity of neighbour searches from $O(n^2)$ to approximately $O(n)$, as only neighbouring cells need to be checked during interaction calculations.

3. **SPH Core Algorithms**
   The SPH simulation operates through a set of core algorithms that model fluid behaviour at a particle level, capturing both the physical dynamics and inter-particle interactions fundamental to realistic fluid simulation. These core components include:

   - **Density and Pressure Computation:**
     A crucial part of SPH is estimation of local density for each particle, which is achieved through the use of smoothing kernel functions. These kernels, such as the Poly6 kernel, define a weighted influence based on the distance between particles, enabling each particle to accumulate density contributions from its neighbours within a support radius. The Poly6 kernel is particularly well-suited for this purpose due to its smooth, continuous nature and zero-gradient at the boundary, which ensures numerical stability in the simulation.
   - **Force Calculations:**
     The interaction between particles is governed by several force models, taken from the physical principles of fluid. Pressure forces are computed using a pressure kernel, which ensures repulsive behaviour as particles come closer together. Viscosity forces account for the internal friction of the fluid, smoothing out velocity differences across particles and promoting smooth flow. Additionally, surface tension is handled by analysing the gradient of the colour field taken from particle densities to identify and stabilise fluids. These force components are computed per timestep to control particle motion and maintain structure.
   - **Velocity Verlet Integration:**
     To update particle positions and velocities over time, the simulation will use Velocity Verlet integration. This method is well-suited for SPH due to its balance of computational efficiency and stability, particularly when dealing with stiff pressure gradients. The algorithm updates positions based on the current velocity and half-step acceleration, followed by a velocity update using the new acceleration values. This two-phase update scheme reduces energy drift over long simulations and helps maintain consistent motion, which is key to helping maintain a stable simulation over time.
   - **Collision Detection and Response:**
     Boundary interactions are managed through an axis-aligned bounding box referred to as the boundary cube, which defines the limits of the simulation domain. Each particle is checked against the extents of this cube to determine

if it has exited the valid region. If a particle crosses the boundary, a response is applied that adjusts its velocity and position to reflect a realistic collision. This response involves applying a repulsion force to redirect the particle inward and a damping factor to simulate the loss of momentum during impact. These adjustments ensure that particles remain within the domain while maintaining realistic fluid motion. The boundary cube can be repositioned or resized, and the simulation continuously updates its parameters to reflect any changes, supporting dynamic control over the simulation space while maintaining performance and physical consistency.

GPU Acceleration Strategy

Given the computational demands of simulating up to 100,000 particles at 30 FPS, GPU acceleration is critical for real-time performance. Unity's support for compute shaders allows the offloading of intensive particle calculations to the GPU, significantly increasing computational efficiency.

(a) Compute Shader Usage
The core SPH calculations, including density-pressure computations, force calculations, and particle position updates, are executed on the GPU using Unity's compute shader functionality. This parallelises the processing of particle interactions, leveraging the massive parallelism of modern GPUs.

(b) Memory Management
Particle data is stored in GPU buffers that can be read and written during shader execution. This minimises data transfer overhead between the CPU and GPU, allowing for efficient real-time updates.

(c) Optimisation Techniques

- Thread Group Sizing: The simulation uses optimised thread group sizes to maximise GPU utilisation.
- Grid-based Neighbour Search: By organising particles into spatial grids on the GPU, neighbour searches become significantly faster, reducing computational bottlenecks.
- Adaptive Time-stepping: Dynamically adjusting the simulation time step based on the simulation's computational load helps maintain real-time performance while preserving accuracy.

(d) Visual Rendering on GPU
The rendering of particles will also utilise GPU acceleration through Unity's shader system. This allows for real-time visual effects such as reflection, refraction, and surface tension visualisation, enhancing the visual realism of the simulation.

This architectural design ensures that the system is capable of handling complex multi-phase fluid interactions while maintaining real-time performance and visual realism. By leveraging Unity's built-in systems and GPU capabilities, the simulation can efficiently manage high particle counts without compromising computational performance.

Development Methodology

Overview of Development Approach

The development methodology adopted for this project is an iterative and incremental approach. Given the computational demands of real-time *multi-phase fluid simulation*, an iterative cycle was chosen to allow for progressive optimisation of simulation accuracy and computational efficiency.

Key reasons for this choice:

- Optimisation in Phases: The complexity of real-time *Smoothed Particle Hydrodynamics (SPH)* requires fine-tuning across multiple iterations to achieve a balance between computational speed and visual realism.
- Flexibility for Enhancements: The methodology allows for incremental improvements, particularly in *GPU acceleration*, *boundary handling*, and *multi-phase interaction refinements*.
- Performance Testing Throughout Development: Rather than evaluating performance at the end of the project, iterative testing ensures that each modification maintains or improves real-time performance.

The iterative nature of the methodology ensures that new techniques, such as *grid-based neighbour searches and adaptive time-stepping*, can be incorporated progressively to refine fluid dynamics and computational performance.

Agile Software Development

An Agile methodology was adopted due to its flexibility and support for iterative development—essential for real-time, GPU-accelerated fluid simulation. The project required frequent testing and refinement of SPH dynamics, rendering methods, and performance bottlenecks. Agile's iterative approach enabled focused development cycles, where features such as particle interaction, surface tension, and collision handling were incrementally implemented and evaluated.

Unlike rigid models such as Waterfall, Agile supports evolving requirements, crucial for performance-sensitive systems. Agile methodologies are particularly beneficial in projects facing high uncertainty and needing continuous adaptation Sommerville (2011). It is especially suitable for experimental software projects where feedback and iteration are essential Pressman & Maxim (2014).

A Kanban board was utilised within the Agile framework to visually track progress, manage workflow, and effectively identify bottlenecks. Compared to Scrum, which uses fixed-length sprints and defined roles, Kanban provided greater flexibility, allowing dynamic re prioritisation of tasks and continuous adjustments without rigid constraints.

Agile also enabled efficient integration between numerical and visual components. Changes in simulation stability often directly influenced rendering design, reflecting the tight coupling between computation and graphics.

In summary, Agile facilitated rapid iteration, real-time performance testing, and adaptive planning—making it highly suited to the technical and experimental nature of this project.

Primary Tools and Frameworks

Unity Engine: Unity is selected for its robust support for compute shaders, built-in real-time physics engine, and extensive rendering pipeline. It excels at efficient GPU resource management, making it ideal for real-time simulation development. Additionally, Unity's user-friendly interface simplifies the creation of interactive and visually rich simulations. The reason for choosing Unity is its well-documented ecosystem and industry-standard toolset, which ensure both development efficiency and scalability for the project.

Compute Shaders (HLSL): Compute shaders, written in HLSL, allow direct parallel processing on the GPU. This capability is critical for accelerating the Smoothed Particle Hydrodynamics (SPH) calculations necessary for simulating multi-phase fluids in real time. The primary motivation for using compute shaders is their ability to handle high particle counts with performance far exceeding that of CPU-based approaches, enabling real-time responsiveness in the simulation.

C#: C# is used to script the simulation logic, manage user interactions, control simulation states, and facilitate communication between Unity's core systems and the GPU computations. It is chosen due to its seamless integration with Unity, high readability, and comprehensive standard libraries, all of which contribute to rapid development and code maintainability.

These tools, when combined within an Agile-driven iterative workflow, provided the flexibility, performance, and scalability needed to implement a real-time multi-phase fluid simulation system.

Implementation, Testing, and Evaluation

Implementation of SPH Simulation

This chapter details the implementation of the Smoothed Particle Hydrodynamics (SPH) simulation for real-time multi-phase fluid interactions. The primary goal of this implementation is to achieve efficient and stable fluid simulation while ensuring real-time performance through GPU acceleration.

Smoothed Particle Hydrodynamics (SPH) models fluids using a Lagrangian framework, where the fluid is represented as a set of discrete particles that move with the flow. Each particle carries physical properties such as position, velocity, density, and pressure, and these properties change as the particles interact with their neighbours. Instead of computing changes across a fixed spatial grid, interactions are determined based on local neighbourhoods using smoothing kernel functions. These functions enable the interpolation of continuous fluid quantities, such as pressure gradients and viscosity, directly from the particle data. This approach provides strong adaptability and is particularly effective for simulating free-surface flows, multi-phase dynamics, and interactions with complex or moving boundaries.

The key challenges in implementing real-time SPH simulation include:

- Efficient Particle Management – Ensuring an optimal balance between accuracy and computational cost.
- Neighbourhood Search Optimisation – Reducing the $O(N^2)$ complexity of neighbour searches using spatial partitioning techniques.
- Stable Multi-phase Interactions – Handling density discontinuities and surface tension for accurate fluid .
- Collision Handling – Ensuring correct interactions with boundaries and obstacles.
- GPU Acceleration – Implementing compute shaders to parallelise computations and achieve real-time performance.
- Pressure Calculation – Ensuring stable and realistic particle interactions through accurate pressure computation based on local density estimates.

The following sections describe the implementation of these components, including particle generation, multi-phase handling, collision detection, GPU acceleration, and visualisation techniques.

Particle Generation and Initialization

In the SPH simulation, fluid is represented as a collection of discrete particles, each carrying physical properties that define its motion and interaction with other particles. These particles move according to the governing equations of fluid dynamics, interacting with neighbouring particles via smoothing kernel functions. The initialisation of these particles is a critical step in ensuring a stable and efficient simulation.

Particle Data Structure Each SPH particle is stored as a structured data type, optimised for compute shader execution. The structure is defined in C# as follows:

```
public struct ParticleData
{
    public Vector3 position;        // Position in world space
    public Vector3 velocity;        // Velocity vector
    public Vector3 acceleration;    // Net force-induced
    acceleration
    public float density;           // Local density
    public float pressure;          // Computed pressure from
    neighbours
    public float restDensity;       // Resting density of the fluid
    public float viscosity;         // Governs fluid resistance
    public float mass;              // Mass of the particle
    public Color color;             // Used to visualise multi-phase
    particles
}
```

Listing 6.1: Particle Data Structure in SPH Simulation

Initialisation of Particles Particles are generated within predefined bounding volumes using the SpawnBox class. Each particle is assigned initial properties based on the fluid type it belongs to.

The particle generation process follows these steps:

(a) **Uniform Grid Placement:** Particles are initially placed in a structured grid within the spawning region to ensure even distribution.

(b) **Random Perturbation:** A small random offset is applied to prevent large amounts particle overlap, which can cause numerical instability.

(c) **Property Assignment:** Each particle is given appropriate mass, density, and viscosity values depending on the fluid type.

(d) **Neighbourhood Structure Initialisation:** The particles are indexed within a spatial hashing system to optimise neighbour searches.

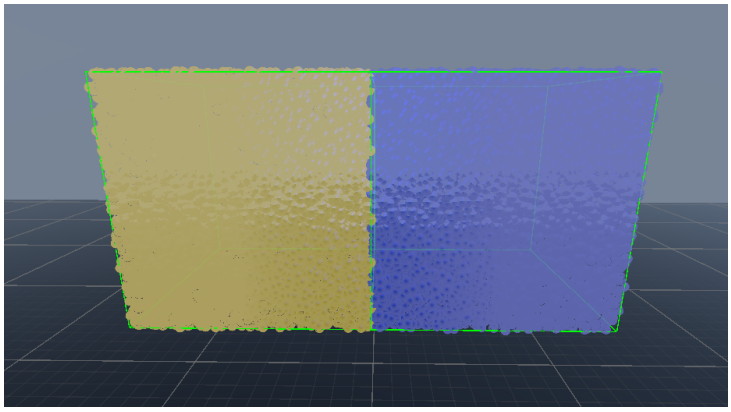

Figure 6.1: Initial distribution of SPH particles before simulation begins.

Multi-Phase Initialisation For multi-phase simulations, each fluid type is assigned properties at the time of creation within the SpawnBox class. These properties include density, viscosity, and colour, ensuring consistency across the simulation. The CreateParticles() function in SPH.cs simply transfers these values to each particle, without requiring additional reassignment.

The following snippet from SpawnBox.cs demonstrates how each fluid type is initialised:

```
public class SpawnBox : Mono
{
    public FluidType fluidType; // Defines the fluid type (e.g., Oil, Water)

    public float restDensity = 20f;
    public float viscosity = 0.1f;
    public float particleMass = 0.03f;

    public Color particleColor = Color.blue;
}
```

Listing 6.2: Multi-Phase Fluid Initialisation

Each particle then inherits its properties during creation:

```
void CreateParticles()
{
    int index = 0;
    foreach (var sb in spawnBoxes)
    {
        for (int i = 0; i < sb.particleCount; i++)
        {
            Particle p;
            p.position = GetRandomSpawnPosition(sb);
            p.velocity = sb.initialVelocity;
            p.restDensity = sb.restDensity;
            p.viscosity = sb.viscosity;
            p.mass = sb.particleMass;
            p.color = sb.particleColor;

            initArray[index] = p;
            index++;
```

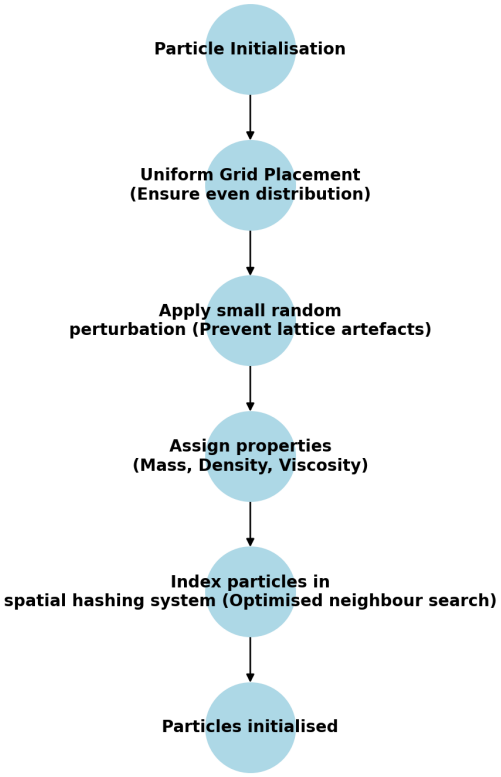

Figure 6.2: Flowchart illustrating the particle initialisation process.

```
18            }
19        }
20    }
```

Listing 6.3: Assigning Properties in CreateParticles()

GPU Acceleration Implementation

The computational cost of simulating real-time fluid dynamics using Smoothed Particle Hydrodynamics (SPH) is significantly high, particularly when dealing with large-scale particle systems. To achieve real-time performance, the SPH simulation is offloaded to the GPU using compute shaders, leveraging the parallel processing capabilities of modern graphics hardware Unity Technologies (2024*b*) Hendra (2015).

Compute shaders are used to accelerate the most computationally expensive operations in SPH, including:

- Neighbourhood Search: Efficiently identifying nearby particles using a grid-based spatial partitioning method.
- Density and Pressure Computation: Estimating fluid density and pressure at each particle using SPH kernel functions.
- Force Computation: Computing pressure, viscosity, and surface tension forces based on particle interactions.

- Boundary Handling: Enforcing boundary constraints and obstacle interactions via penalty forces.
- Time Integration: Updating particle positions and velocities using an integration method such as Velocity Verlet.

The following sections describe the GPU acceleration implementation in detail.

Compute Shaders Setup

The compute shader setup follows a structured pipeline, where different kernels are responsible for specific SPH operations. The core compute shaders used in this implementation are defined in SPHCompute.compute and are called from SPH.cs.

GPU Buffer Management The particle data is stored in GPU buffers to minimise CPU-GPU memory transfer overhead Unity Technologies (2024*a*). These buffers include:

- `particleBuffer` – Stores all particle properties, including position, velocity, density, and pressure.
- `gridCountsBuffer` – Stores the number of particles in each spatial grid cell.
- `gridIndicesBuffer` – Stores the indices of particles within each grid cell to optimise neighbour searches.
- `collisionCounterBuffer` – Tracks particle collisions with boundaries for constraint enforcement.

The buffers are initialised in SPH.cs:

```
particleBuffer = new ComputeBuffer(particleCount, sizeof(float) *
    8);
gridCountsBuffer = new ComputeBuffer(totalCells, sizeof(int));
gridIndicesBuffer = new ComputeBuffer(totalCells *
    maxParticlesPerCell, sizeof(int));
collisionCounterBuffer = new ComputeBuffer(1, sizeof(int)); //
    Boundary collision tracking
```

Listing 6.4: GPU Buffer Initialization in SPH.cs

Kernel Functions The compute shader consists of multiple kernels, each responsible for a specific stage of the SPH computation:

- `kernel_ClearGrid` – Resets the grid cell counters.
- `kernel_BuildGrid` – Assigns particles to grid cells to facilitate efficient neighbour searching.
- `kernel_DensityPressure` – Computes fluid density and pressure using the SPH summation method.
- `kernel_ForceXSPH` – Computes viscosity forces using the XSPH method.
- `kernel_VvHalfStep` – Performs the first half of Velocity Verlet integration.

- `kernel_VvFullStep` – Completes the integration step and updates particle positions.
- `kernel_BoundObs` – Handles boundary constraints and obstacle interactions using collision forces.
- `kernel_VelocityDamping` – Applies artificial damping to maintain numerical stability.
- `kernel_Clear` – Resets temporary computation buffers before force updates.

These kernels are invoked in sequence to execute a complete SPH update per frame. The execution pipeline in SPH.cs is structured as follows:

```
DispatchCompute(kernel_ClearGrid, totalCells);
DispatchCompute(kernel_BuildGrid, particleCount);

for (int i = 0; i < subSteps; i++)
{
    DispatchCompute(kernel_VvHalfStep, particleCount);
    DispatchCompute(kernel_Clear, particleCount); // Reset temporary buffers
    DispatchCompute(kernel_DensityPressure, particleCount);
    DispatchCompute(kernel_ForceXSPH, particleCount);
    DispatchCompute(kernel_VvFullStep, particleCount);
    DispatchCompute(kernel_BoundObs, particleCount); // Boundary enforcement
}

DispatchCompute(kernel_VelocityDamping, particleCount);
```

Listing 6.5: Compute Shader Dispatch Sequence

Each kernel executes in parallel across multiple GPU threads, allowing thousands of particles to be updated simultaneously.

Collision Detection and Boundary Handling

Boundary Representation The simulation domain is encapsulated within a bounding cube, ensuring particles do not escape. This boundary is dynamically updated using the boundaryCube transform, converting local-space coordinates into world-space for accurate collision handling. If no explicit boundary cube is assigned, an axis-aligned bounding box (boundsCenter, boundsSize) is used as a fallback.

Boundary parameters are updated via:

```
if (boundaryCube != null)
{
    if (Vector3.Distance(boundaryCube.position, lastBoundaryCubePosition) > boundaryUpdateThreshold ||
        Quaternion.Angle(boundaryCube.rotation, lastBoundaryCubeRotation) > boundaryUpdateThreshold)
    {
        UpdateGridParametersFromBoundary();
        lastBoundaryCubePosition = boundaryCube.position;
        lastBoundaryCubeRotation = boundaryCube.rotation;
    }
```

```
}
```

Listing 6.6: Boundary Grid Update

Penalty Force Method To prevent boundary violations, a repulsive force is applied when particles exceed the boundary. This force correction is computed in the shader (kernelBoundObs).

The penalty force applied to a violating particle is given by:

$$\mathbf{F}_b = -k(\mathbf{r}_p - \mathbf{r}_b) \tag{6.1}$$

where:

- $\mathbf{r}_p$ is the particle's position.
- $\mathbf{r}_b$ is the closest boundary point.
- k is the stiffness coefficient controlling force magnitude.

Boundary force application is executed in the compute shader via:

```
DispatchCompute(kernel_BoundObs, particleCount);
```

Listing 6.7: Boundary Force Application in Compute Shader

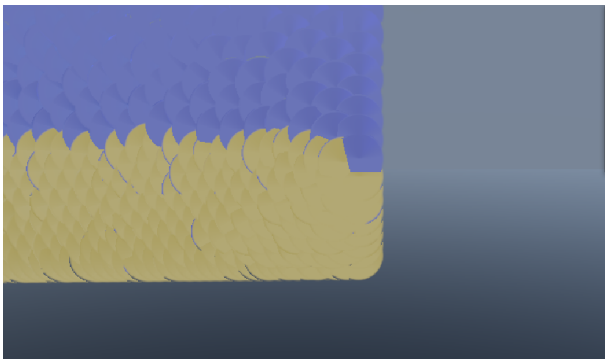

Figure 6.3: Particles interacting with boundaries using penalty forces

This GPU-accelerated method ensures real-time boundary enforcement, preventing particles from escaping while maintaining stable interactions.

Parallel Processing Strategies

Thread Group Configuration The compute shaders process particles in parallel using a thread group configuration that maximises GPU efficiency. Each kernel is dispatched with a thread group size defined as:

```
private const int THREAD_GROUP_SIZE = 256;
int groups = Mathf.CeilToInt(count / (float)THREAD_GROUP_SIZE);
```

Listing 6.8: Defining Thread Group Size

Each thread processes one particle, and multiple groups execute concurrently to ensure high throughput Huang et al. (2023).

Spatial Partitioning for Neighbour Search A key computational challenge in SPH simulations is the $O(N^2)$ complexity of brute-force neighbour searches. To optimise performance, this implementation uses grid-based spatial partitioning, reducing complexity to approximately $O(N)$.

Each particle is assigned to a grid cell, and neighbour searches are restricted to adjacent cells, significantly reducing computational overhead. The kernel_BuildGrid function executes this in parallel:

```
uint cellID = GetGridCellID(particlePos);
uint index = atomicAdd(gridCounts[cellID], 1);
gridIndices[cellID * maxParticlesPerCell + index] = particleID;
```

Listing 6.9: Neighbour Search Grid in Compute Shader

This approach ensures that only relevant particle interactions are computed, improving efficiency without compromising accuracy.

Memory Optimisation To maximise GPU memory efficiency and minimise latency, the following optimisations are employed:

- Shared Memory Usage: Frequently accessed neighbour data is stored in shared memory, avoiding slow global memory access.
- Memory Coalescing: Particle properties are stored in contiguous memory to improve cache efficiency and reduce memory fetch time.
- Atomic Operations: atomicAdd is used to update grid counters safely, avoiding race conditions during parallel execution.

Memory buffers are correctly allocated and assigned to compute shaders in SPH.cs:

```
sphCompute.SetBuffer(kernel_BuildGrid, "GridCounts",
    gridCountsBuffer);
sphCompute.SetBuffer(kernel_BuildGrid, "GridIndices",
    gridIndicesBuffer);
sphCompute.SetBuffer(kernel_DensityPressure, "Particles",
    particleBuffer);
sphCompute.SetBuffer(kernel_BoundObs, "CollisionCounter",
    collisionCounterBuffer);
```

Listing 6.10: Memory Buffer Setup in Compute Shaders

Summary

This section described the GPU acceleration strategies used in the SPH simulation, focusing on:

- Compute shader thread group configuration for efficient parallel execution.
- Grid-based spatial partitioning to accelerate neighbour searches.
- Memory optimisations including shared memory, coalescing, and atomic operations.
- Adaptive time-stepping to improve numerical stability and performance.

By leveraging these parallel processing techniques, the simulation achieves real-time performance, handling tens of thousands of particles efficiently while maintaining accurate fluid dynamics.

XSPH Velocity Correction

To improve stability and reduce visual artefacts such as clumping or jittering, the simulation applies an XSPH velocity correction. This method smooths out the motion between neighbouring particles by slightly adjusting each particle's velocity based on its neighbours Monaghan (2005).

Instead of using only its own velocity, each particle takes into account how its neighbours are moving, and gently shifts its velocity in that direction. This blending creates a more natural and stable flow, especially in regions where particles are densely packed or moving rapidly.

The correction is implemented in the `kernel_ForceXSPH` compute shader, and works by averaging the velocity differences between a particle and its neighbours, then applying a small adjustment. A scaling factor controls how strong this correction is—typically a small value to avoid interfering with the main dynamics.

This technique helps keep the fluid looking smooth and prevents unnatural motion, contributing to the overall stability and realism of the simulation.

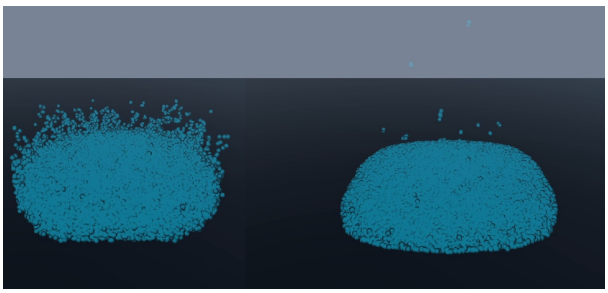

Figure 6.4: XSPH (0.1 Left) (0.9 Right)

Neighbourhood Grid System

Efficient neighbour search is essential for maintaining real-time performance in SPH simulations. To reduce the computational cost of particle interaction checks, the simulation uses a uniform grid-based spatial partitioning system. This system divides the simulation domain into discrete grid cells, allowing neighbour searches to be limited to nearby cells rather than performing brute-force comparisons between all particles.

Grid Creation and Resolution At initialisation, the simulation domain is divided into a 3D uniform grid, with the size of each cell set to match the smoothing radius h of the SPH kernel. This ensures that all particles potentially influencing

one another reside within the same or adjacent cells. The grid resolution is calculated based on the boundary size and smoothing radius, ensuring coverage of the entire simulation volume.

Each frame, particles are reassigned to grid cells based on their updated positions using:

$$\text{cellID} = \left\lfloor \frac{\mathbf{r}_p}{h} \right\rfloor \tag{6.2}$$

where $\mathbf{r}_p$ is the particle's position and h is the smoothing radius. This ID is used to place particles into corresponding cells in the GPU grid buffers.

Neighbour List Construction Once particles are sorted into grid cells, a neighbour list is constructed by scanning each particle's own cell and the 26 adjacent cells (6 faces, 12 edges, 8 corners). This process identifies neighbouring particles within the support radius, significantly limiting the number of distance checks. The neighbour list allows each particle to access the indices of nearby particles during density, pressure, and force calculations.

This is implemented in the `kernel_BuildGrid` shader using atomic operations to populate cell lists in parallel (as shown in Listing 6.9)

Dynamic Grid Updates To maintain accuracy, the grid structure is rebuilt every frame based on the current particle positions. This ensures that all spatial queries reflect the latest state of the fluid. If the simulation domain changes (e.g., due to movement or resizing of the boundary cube), the grid dimensions and resolution are recalculated accordingly to continue optimising neighbour searches.

By combining a fixed-resolution grid with dynamic updates and efficient parallel indexing, the neighbourhood system enables the simulation to scale to large particle counts while preserving real-time performance.

Implementation This pressure calculation is performed in the `kernel_DensityPressure` compute shader after the local density has been summed via the Poly6 kernel. The pressure is then used in the subsequent force computation step to apply inter-particle repulsion, which drives the fluid apart in denser regions and allows it to expand where underdense.

By grounding pressure in the difference between current and rest density, the simulation maintains realistic particle spacing, prevents unnatural clustering, and enforces volume conservation over time.

Rest Density and Pressure Calculation

Accurate pressure estimation is fundamental in SPH to ensure particles respond realistically to local density changes. Pressure acts as the primary force driving fluid expansion and compression, influencing particle motion and maintaining fluid structure.

Rest Density Each fluid phase is assigned a rest density value at initialisation, which defines the density the fluid seeks to maintain under equilibrium. This rest density is a critical reference point for determining how compressed or expanded a fluid region is. Particles deviating from this value will experience pressure forces directing them back towards equilibrium.

In multi-phase simulations, rest density varies between fluid types (e.g., water vs oil), which contributes to phase separation and buoyancy effects Tryggvason et al. (2011).

Pressure Equation Pressure is computed for each particle using an equation of state derived from the Tait equation, which ensures a nonlinear response to density deviations Liu & Liu (2010). The standard SPH formulation is:

$$p_i = k \left(\left(\frac{\rho_i}{\rho_0} \right)^{\gamma} - 1 \right) \tag{6.3}$$

where:

- p_i is the pressure of particle i.
- ρ_i is the current local density of the particle.
- ρ_0 is the rest density of the fluid.
- k is a stiffness constant controlling compressibility.
- γ is typically set to 7 for weakly compressible fluids.

This nonlinear formulation ensures stability and helps preserve incompressibility, while still allowing small compressions necessary for practical simulation.

Implementation This pressure calculation is performed in the `kernel_DensityPressure` compute shader after the local density has been summed via the Poly6 kernel. The pressure is then used in the subsequent force computation step to apply inter-particle repulsion, which drives the fluid apart in denser regions and allows it to expand where underdense.

By grounding pressure in the difference between current and rest density, the simulation maintains realistic particle spacing, prevents unnatural clustering, and enforces volume conservation over time.

Visualisation Techniques

The visualisation of fluid particles is essential for understanding and analysing the simulation. In this implementation, real-time rendering techniques are used to efficiently display thousands of particles while maintaining visual realism. The rendering pipeline is implemented using Unity's Graphics API and custom shaders to achieve accurate surface reconstruction and realistic fluid effects Unity Technologies (2024*d*).

Particle Rendering System

Rendering Approach The rendering system uses a point-based representation where each fluid particle is drawn as a small sphere. The rendering pipeline consists of:

(a) GPU Instancing for Performance: Each particle is rendered using billboarding techniques, allowing efficient rendering of thousands of particles.

(b) Screen-Space Splatting for Surface Reconstruction: Depth-based rendering reconstructs the fluid surface.

(c) Post-Processing for Visual Enhancement: Effects such as reflections, refractions, and transparency are applied through shader-based rendering.

GPU Instanced Rendering To efficiently render large numbers of particles, GPU instancing is utilised Harada et al. (2007) . Each particle's data is passed to the GPU, avoiding expensive per-frame CPU draw calls Unity Technologies (2024*c*).

The following code snippet from SPH.cs shows how the particles are drawn using Unity's Graphics API:

```
void OnRenderObject()
{
    if (fluidMaterial != null)
    {
        fluidMaterial.SetPass(1);
        Graphics.DrawProceduralNow(MeshTopology.Points,
   particleCount);
    }
}
```

Listing 6.11: GPU Instanced Particle Rendering

This method ensures that all particles are drawn in a single GPU draw call, significantly improving performance.

Surface Reconstruction Using Screen-Space Splatting Since the simulation uses particle-based rendering, a surface reconstruction method is required to create the illusion of a smooth fluid surface. This is achieved using screen-space splatting, where each particle is rendered as a small sphere and blended in a post-processing pass Ihmsen et al. (2010).

The rendering shader applies a depth-based splatting technique Unity Technologies (2020):

```
ZWrite On               // Do not write to the depth buffer
ZTest LEqual            // Use depth testing but rely on sorting
```

Listing 6.12: Depth-Based Splatting in Shader

These settings ensure that particles are correctly blended to form a continuous fluid surface.

Visual Effects for Realistic Simulation

To enhance realism, several post-processing techniques are applied, including refraction, reflection, and surface shading.

Refraction Effects Real-world fluids distort light passing through them, creating a refractive effect. This effect is implemented in the fluid rendering shader using screen-space UV perturbations:

```
float3 refraction = tex2Dproj(_WaterGrabTex, UNITY_PROJ_COORD(
    grabUV)).rgb;
```

Listing 6.13: Screen-Space Refraction Effect

The refraction is calculated based on the grab texture, capturing the background before blending the refraction effect.

Reflection Approximation Fluid surfaces often exhibit reflective properties, especially when simulating materials like oil or mercury. Reflections are implemented using environment cubemaps:

```
float3 reflection = texCUBE(_Cube, reflect(-viewDir, normal)).rgb;
```

Listing 6.14: Cubemap Reflection Calculation

This allows the fluid to reflect the surrounding environment, improving realism.

Fresnel Effect for Realistic Transparency The Fresnel effect changes the transparency of the fluid based on the viewing angle. This ensures that edges of the fluid appear more transparent, Just like real-world fluid :

$$F = F_0 + (1 - F_0)(1 - \cos\theta)^5 \tag{6.4}$$

where:

- F_0 is the base reflectance.
- θ is the angle between the view direction and the surface normal.

The effect is implemented in the shader:

```
float fresnelFactor = pow(1.0 - saturate(dot(viewDir, normal)),
    _FresnelPower);
```

Listing 6.15: Fresnel Transparency Effect

This ensures that at shallow angles, the surface appears more reflective.

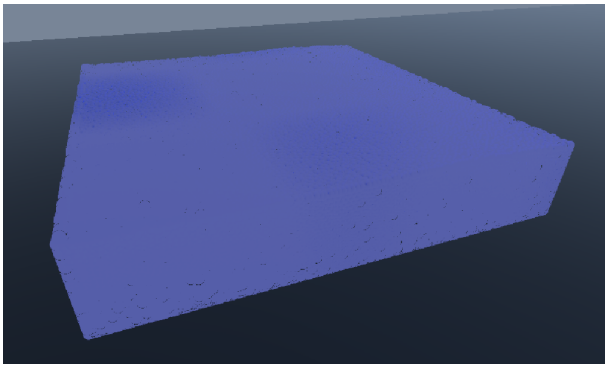

Figure 6.5: Final fluid surface render

Testing and Evaluation

Functional Testing

This section presents a series of functional tests designed to evaluate the core s of the real-time SPH simulation. The objective is to determine whether the system operates as intended, both in terms of physical accuracy and implementation reliability, within the constraints of real-time performance.

The simulation is assessed across three key areas: correct particle creation and initialisation, approximation of physical laws such as mass and energy conservation, and the accurate interaction of multiple fluid phases. Each subsection outlines the intended , describes the testing process and results, and discusses the reasoning behind the observed outcomes with consideration for the trade-offs inherent to real-time simulation.

Particle Creation and Initialisation

Particles should be spawned with accurate initial values from their associated `SpawnBox` components, including position within the defined volume, initial velocity, rest density, viscosity, mass, and colour. The objective of this test was to ensure that all particles are created as intended and correctly configured before entering the simulation loop.

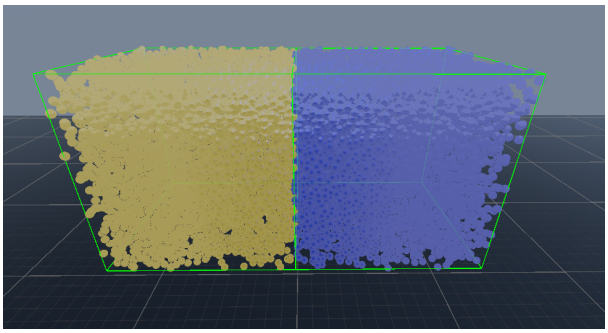

Figure 6.6: Particles created and positioned correctly within spawn volume

To validate this, a test was conducted where all particle attributes were logged upon initialisation. The output was analysed to confirm the following:

- All particles were spawned within the boundaries of their given spawn boxes.
- Each particle was assigned the correct mass, rest density, and viscosity.
- The total number of particles matched the expected count calculated from all spawn boxes combined.
- The overall mass was consistent with the total particle count and per-particle mass value.

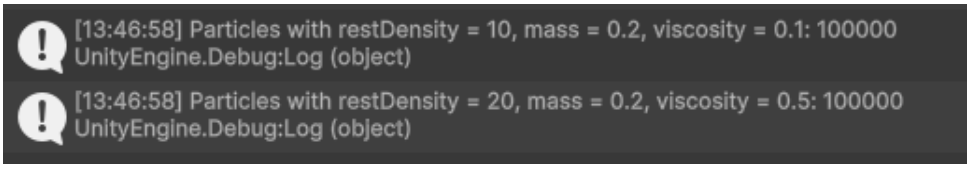

Figure 6.7: Console output showing total particle count and accurate assignment of mass, rest density, and viscosity

This test confirms that the spawning system correctly applies configuration data from each `SpawnBox`. Both visual inspection (Figure 6.6) and value output (Figure 6.7) indicate that there were no abnormalities in particle placement or value assignment. Verifying that particles are correctly initialised is a critical for accurate fluid in the simulation pipeline.

Basic Physics Approximation

An essential part of validating the fluid simulation is ensuring that it adheres to the basic principles of physics — particularly the conservation of mass, and the approximation of momentum and energy in a way that supports visually realistic . This includes particles responding correctly to gravity, collisions, and fluid motion consistent with something seen in real life.

Figure 6.8: Total mass and kinetic energy of particles over time

The graph in Figure 6.8 shows total mass and kinetic energy recorded over the duration of the simulation. Mass remains conserved throughout, with no loss. This validates that the simulation's core data structures correctly retain the particles state, and confirms that properties like particle count and per-particle mass remain constant.

Kinetic energy shows a gradual decay over time — that is both expected. In physical fluid systems, energy is continually lost to internal friction, viscosity, and turbulence until the fluid comes to rest. This is displayed in the simulation through the inclusion of viscosity and damping forces. However, as seen in the figure, total energy does not reach zero. This is a result of the limitations of real-time SPH, where the use of simplified integration methods and approximated force models can cause residual motion or minor energy changes that can add up.

Importantly, these results do not indicate that the simulation is badly designed, but rather a compromise with the project's aims. In real-time applications, computational performance and visual realism take precedence over strict physical accuracy.

This test confirms that the simulation balances the expected trade-offs effectively: mass is strictly conserved, and energy behaves in a believable way despite known computational approximations.

Multi-phase Fluid Interactions

The simulation aims to support realistic multi-phase fluid — for example, oil and water as immiscible fluids that naturally separate and interact based on differences in their physical properties. The goal was to verify that fluids with distinct densities and viscosities would not blend unnaturally and would instead demonstrate expected layering and phase boundaries.

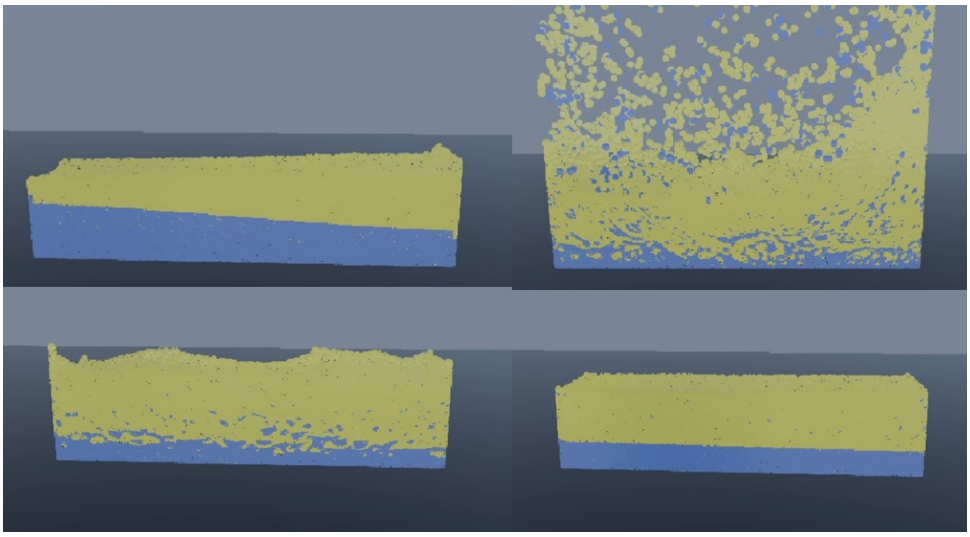

Figure 6.9: Simulation showing oil and water resting separately, then mixed and returning to a separate rest state

The test involved spawning two separate fluid phases using different `SpawnBox` components — one with a higher rest density and viscosity (representing water), and

one with lower values (representing oil). The result, as shown in Figure 6.9, was that the heavier phase consistently settled beneath the lighter one, and distinct boundaries were maintained throughout interaction. Colour coding of each fluid allowed for easy visual tracking during the mixing and settling process.

As the SPH model handles phase separation intrinsically through localised pressure and density calculations. Since each particle retains its own properties (e.g., rest density, mass, viscosity), the simulation naturally respects the physics of multi-phase fluids, relying on the local interactions defined by the smoothing kernel and force equations.

The results indicate that the implementation successfully supports visually accurate fluid dynamics. Although some minor numerical inaccuracies may occur due to the approximations, the overall separation and are consistent with real-world expectations of multi-phase fluids.

Performance Testing

Overview

The performance evaluation of the real-time multi-phase fluid simulation was conducted to assess the computational efficiency of the implementation. Two primary tests were carried out: the first comparing single-phase and multi-phase simulations at different particle counts, and the second measuring the impact of grid resolution on performance. The hardware used for testing included an **RTX 4080 Laptop GPU**, an **Intel i9-13900H CPU**, and **32GB of RAM**, ensuring that performance was primarily limited by the simulation rather than hardware constraints.

Single-phase vs Multi-phase Performance

The first test focused on how the simulation scaled with an increasing number of particles. FPS was recorded for both **single-phase (water only)** and **multi-phase (water and oil)** simulations.

Particles	FPS (Single-phase)	FPS (Multi-phase)
1,000	646	615
10,000	620	600
100,000	344	320
200,000	169	149
400,000	59	56
600,000	25.4	28.1
1,000,000	9.5	8.4

Table 6.1: Comparison of FPS between single-phase and multi-phase simulations

At lower particle counts (1,000–10,000), the simulation maintained high FPS, as the computational load remained low. As the number of particles increased, FPS declined due to the rising number of particle interactions, which significantly increased computational complexity. The single-phase simulation performed slightly

better across all cases, as multi-phase simulations require additional calculations for surface tension and density variations at fluid interfaces. However, the difference in performance remained relatively small, indicating that the GPU compute shader efficiently handled the additional multi-phase computations.

Effect of Grid Resolution on Performance

The second test assessed how increasing the **grid resolution** affected FPS when simulating **200,000 water particles**.

Grid Resolution	FPS
10	3.18
20	8.8
40	29.2
60	69
80	121.8
100	176

Table 6.2: Effect of grid resolution on FPS with 200,000 water particles

Performance improved as the grid resolution increased, with FPS rising from **3.18 FPS at a resolution of 10** to **176 FPS at a resolution of 100**. This improvement was due to more efficient spatial partitioning, which reduced the number of unnecessary particle interactions. The neighbour search algorithm, which determines which particles interact with each other, benefits significantly from a finer grid, as it limits the number of particles that need to be checked per cell. However, grid resolution scales cubically in memory usage, meaning that increasing it beyond a certain point results in diminishing returns due to memory constraints, although the simulation would fail to load the particles into a grid of this size due to the exponential scaling not allowing for testing on grid resolutions above 100.

Performance on Other GPU's

The performance testing was done on an RTX 4080 Laptop GPU, a high-end card designed for more demanding computational tasks. However, additional testing on more budget-friendly GPUs demonstrated that the simulation remains highly efficient across a range of hardware. Running the simulation on an RTX 4060, the system maintained stable performance and only began to drop below 30 FPS at around 450,000 particles. Further testing on an RTX 3050 Laptop GPU showed a drop in performance at around the 200,000 particle count, where maintaining 30 FPS became difficult. These results show that the simulation is well-optimised and does not rely solely on high-end GPUs or raw computational power. Instead, it offers stable performance even at high particle counts on a range of mid-tier and entry-level graphics cards.

Performance Tests Conclusion

The results clearly demonstrate that the SPH simulation scales poorly at high particle counts due to quadratic growth in particle interactions, a phenomenon well-documented in SPH literature Ihmsen et al. (2014). Multi-phase simulations introduced additional computational overhead from surface tension and interfacial calculations, aligning with findings in prior research Liu & Liu (2010). However, consistent with earlier GPU-accelerated SPH studies Harada et al. (2007), the GPU effectively handled the increased computations, leading to minor performance impacts.

The tests on grid resolution underline the critical importance of spatial partitioning for efficient SPH computations, which also highlighted substantial performance gains from optimised neighbour searches. Nonetheless, memory limitations imposed practical constraints, a challenge similarly reported in existing GPU-based SPH implementations Hendra (2015).

Overall, these findings reinforce that performance in real-time fluid simulations is primarily constrained by particle interaction complexity and spatial partitioning efficiency—key factors repeatedly emphasised in SPH research Ihmsen et al. (2014), Macklin & Müller (2013).

Visual Fidelity Assessment

This section evaluates the visual realism of the simulation through a series of targeted tests, focusing on how fluid responds to dynamic boundaries, object interactions, and external forces. As visual fidelity is a critical measure in real-time fluid simulations, each test was chosen to assess how convincingly the fluid mimics real-world physical responses under different visual conditions.

Boundary Adaptation Test

To evaluate the simulation's ability to handle dynamic boundary transformations, a boundary adaptation test was conducted using a cube-shaped container that underwent continuous rotation during runtime. This test assessed whether the simulation system could maintain fluid confinement and stability when the spatial constraints were in motion.

The simulation domain was explicitly configured to use cubic bounds, matching the system's grid structure and spatial partitioning logic. Throughout the test, the container was rotated along multiple axes using incremental updates to its transformation matrix. These rotations were applied gradually to ensure that both the particle system and the spatial grid had sufficient time to adjust.

As demonstrated in Figure 6.10, the fluid particles maintained consistent alignment with the rotating container, adapting their motion and distribution to the transformed boundary in real time. Notably, no particles escaped the bounds or displayed erratic , which indicates that the updated grid transformation logic successfully recalculates the simulation space with respect to the boundary cube's rotation.

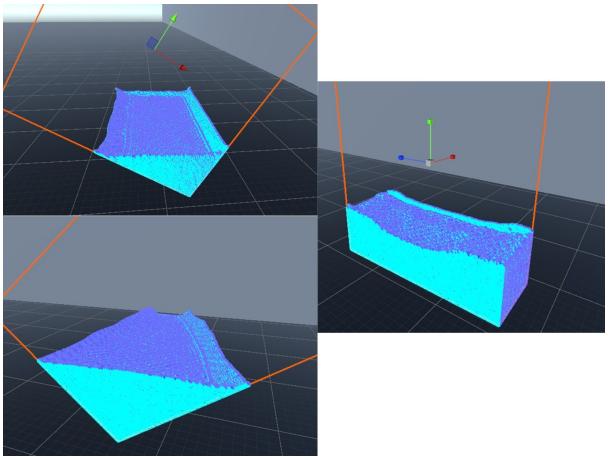

Figure 6.10: Fluid conforming to rotated boundary cube

Furthermore, visual stability was preserved across all frames, with no noticeable artefacts or instability. This confirms the robustness of the boundary handling mechanism implemented in the simulation, particularly the integration between particle-boundary collision detection and the dynamic grid alignment logic..

The test provides strong evidence that the SPH implementation, combined with real-time boundary detection and transformation-aware spatial partitioning, is capable of maintaining physically plausible even under dynamic spatial conditions.

Object Interaction Test

To further evaluate the fluid system's responsiveness to dynamic environmental conditions, a test was conducted in which a rigid sphere was moved through a settled body of fluid. The aim of this test was to observe the resulting fluid deformation, displacement , and the system's ability to simulate realistic wake formation and pressure gradients in response to solid-body motion.

In the simulation, solid objects such as spheres interact with fluid particles through a repulsion-based collision response implemented via Unity's compute shader pipeline. The object's transform is passed into the shader each frame, and its world-space position and radius are used to detect overlapping particles. Any particle found within the object's collision radius is pushed outward using a penalty force, calculated based on the direction and depth of overlap. This approach ensures that the object realistically displaces fluid in real time, while maintaining consistent particle behaviour and simulation stability on the GPU.

The sphere was moved laterally at a constant, moderate velocity, intersecting the particle-based fluid domain. During this interaction, both the front-facing and trailing s of the fluid were observed, with a focus on assessing the smoothness of deformation, adherence to conservation principles.

As illustrated in Figure 6.11, the fluid exhibits physically plausible deformation in response to the sphere's passage. The particles ahead of the sphere compact and curve smoothly, forming a bow-shaped depression, while the trailing region forms coherent wake structures. The reaction of the fluid is continuous, with no visible

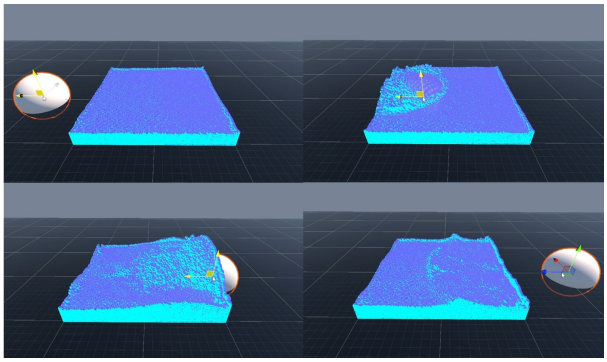

Figure 6.11: Realistic fluid deformation as sphere moves through it

tearing, jittering, or unphysical motion, highlighting the stability of the underlying SPH solver.

This confirms that the implemented pressure and viscosity solvers are functioning as intended. In particular, the smooth displacement and gradual re convergence of particles behind the sphere suggest that the simulation handles local pressure gradients effectively while preserving momentum. Additionally, the stability of the interaction—despite being computed in real time—demonstrates the effectiveness of the time integration and force calculation strategies adopted in the implementation.

Overall, this test validates the capability of the system to simulate complex, real-time interactions between fluids and dynamic solid objects with both visual and physical credibility.

Splash and Wave Response Test

To assess the system's response to high-energy interactions and its capacity to simulate complex surface dynamics, a splash test was conducted by releasing a compact volume of fluid from a height above the boundary container. This test primarily evaluated the simulation's handling of rapid momentum transfer, splash formation, and wave propagation across the fluid surface.

The falling fluid mass was configured to have a higher initial potential energy, ensuring a significant impact on the settled fluid body below. The resulting was observed for qualities such as symmetry, dispersion patterns, and the generation and attenuation of secondary waves.

As shown in Figure 6.12, the system effectively simulated a realistic splash event. Upon impact, fluid particles dispersed radially from the collision site, forming an outwardly expanding wavefront. The splash exhibited coherent structure, with upward jets and a central cavity consistent with observations of real-world fluid impacts. Following the initial burst, surface waves propagated across the fluid volume, eventually diminishing due to viscosity-driven damping.

The result demonstrates the accuracy of the simulation's momentum transfer calculations and confirms that the pressure, surface tension, and damping models contribute meaningfully to maintaining stability during energetic interactions. Additionally, the lack of numerical artefacts or instability further validates the ro-

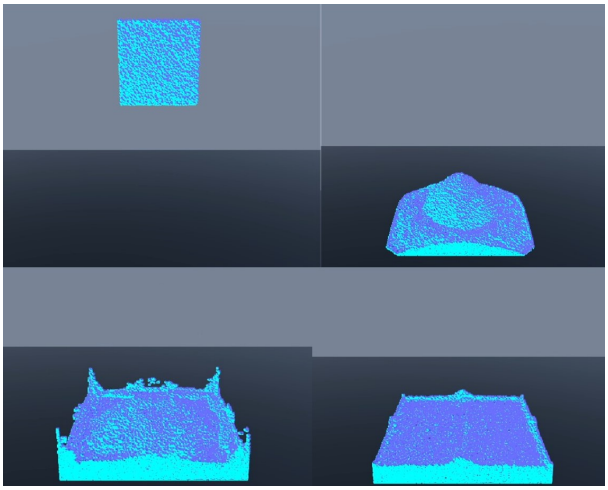

Figure 6.12: Splash and wave propagation from a falling water volume

bustness of the time-stepping scheme and the SPH kernel smoothing used in the simulation.

This test reinforces the system's ability to handle sudden changes in energy and momentum, reproducing complex surface s such as splashes and wave interactions in real time with high fidelity and stability.

Evaluation of Objectives

This section evaluates the extent to which the project's original objectives were achieved. These objectives were defined in terms of technical implementation, simulation fidelity, and performance, with an emphasis on real-time simulation of multi-phase fluids using Smoothed Particle Hydrodynamics (SPH) and GPU acceleration.

SPH Fundamentals and Implementation

Smoothed Particle Hydrodynamics (SPH) treats fluids as discrete particles whose physical properties are approximated through kernel-based interpolation. This formulation enables flexible, mesh-free simulation of continuous media, making it particularly effective for complex and dynamic fluid flows Monaghan (2005).

The simulation incorporated core SPH elements, including pressure and viscosity forces, density estimation, and external body forces. These were implemented in GPU compute shaders, allowing highly parallelised processing. Stability was maintained through the use of smoothing kernels, velocity dampening, and semi-implicit time integration Liu & Liu (2010). Tests under static and dynamic conditions confirmed correct physical responses and consistent particle behaviour.

While predictive-corrective schemes such as PCISPH Solenthaler & Pajarola (2009) offer increased incompressibility and stability, this project prioritised simpler force models and kernel smoothing to maintain real-time performance. This approach reflects common trade-offs in GPU-based SPH, where advanced methods can incur significant computational overhead Ihmsen et al. (2014).

Additionally, the particle interaction performance aligns with observations by Harada et al. Harada et al. (2007), confirming that compute shader implementations can effectively handle large-scale SPH systems on consumer-grade hardware. Though surface fidelity does not fully match hybrid models Prosperetti & Tryggvason (2009), the results remain plausible under interactive constraints.

Review and Application of Fluid Simulation Methods

A literature review guided key design decisions, focusing on Lagrangian methods for their adaptability to free-surface and multi-phase flows. Methods such as Position-Based Fluids and Predictive-Corrective SPH were evaluated for their strengths in stability and incompressibility Macklin & Müller (2013), Solenthaler & Pajarola (2009). Though not implemented due to time constraints, these approaches informed choices in neighbour search, force resolution, and boundary treatment.

The review also highlighted the trade-offs between grid-based and particle-based models. The adopted SPH method, supported by the reviewed material, provided the best compromise between flexibility, accuracy, and real-time feasibility.

Real-Time Optimisation Using GPU Shaders

To ensure real-time performance, the simulation offloaded all particle interaction calculations to the GPU. Operations such as neighbour searching and force integration were executed in parallel via compute shaders. A uniform spatial grid was employed to optimise locality and reduce neighbour search time, while adaptive time-stepping was introduced to maintain stability during high-energy interactions Harada et al. (2007).

Performance strategies followed practices from real-time SPH literature, ensuring cache-efficient memory access and avoiding thread divergence Ihmsen et al. (2010). The final system maintained stable frame rates across a range of fluid configurations and particle counts.

Multi-phase Interaction Techniques

Fluids with different rest densities and viscosities were simulated to represent distinct material phases. Interfacial behaviour was maintained through careful force blending and consistent smoothing radius usage. Visual identifiers, including colour mapping and phase-specific properties, were employed to distinguish phases clearly.

Simulation results demonstrated stable separation between phases and accurate buoyancy effects, aligning with behaviours observed in multi-phase fluid dynamics literature Liu & Liu (2010). Although more advanced effects such as phase change were beyond the project scope, the system effectively captured the core behaviours of immiscible fluids.

SPH System Architecture and Behaviour

The simulation architecture was designed to provide full control over particle generation, state management, and physics computation. Core simulation logic was written in C# to interface with Unity's rendering pipeline and control simulation states, while the physics engine resided in HLSL compute shaders.

This modular structure supported both single-phase and multi-phase scenarios with minimal code duplication. The system was validated through a series of test cases involving fluid-body interaction, container dynamics, and collision response. All tests confirmed numerical stability and consistent fluid motion under a variety of conditions.

Real-Time Fluid Visualisation

Visualisation was achieved using a custom particle rendering solution integrated into Unity's rendering pipeline. Particles were shaded using screen-space lighting models and coloured based on their fluid phase. This provided immediate visual feedback of simulation dynamics and improved interpretability.

While the system rendered fluids as discrete particles, coherence and motion were sufficient to give the impression of a continuous medium. Visual quality remained high under all tested conditions, including stress tests involving fast-moving boundaries and high-energy impacts.

Performance and Visual Fidelity Evaluation

The simulation consistently achieved frame rates above 30 FPS for particle counts up to 100,000 on modern consumer hardware. Benchmarking across different configurations showed that compute-bound tasks scaled predictably with particle count, while rendering remained performant due to GPU-side visual processing.

Visual results were physically plausible and free from artefacts such as clumping or instability. Although performance declined beyond 100,000 particles, the simulation remained visually stable and responsive. These findings confirm that the core goals of real-time interactivity and simulation fidelity were met within the defined system constraints.

Conclusion

Conclusion Overview

This project set out to explore and implement a real-time simulation of multi-phase fluids using Smoothed Particle Hydrodynamics (SPH), accelerated by GPU compute shaders. The motivation, as outlined in the introduction, stemmed from the need for physically plausible and computationally efficient fluid models in interactive environments such as games, scientific visualisation, and simulation-based training.

The core aim was to build a technically sound framework that could capture the complexity of fluid-fluid interactions while maintaining real-time responsiveness. Drawing from foundational SPH theory and recent developments in GPU-accelerated simulation, the project combined numerical accuracy with modern rendering techniques to produce a system that is both visually convincing and computationally performant

Throughout the development process, the system was guided by a commitment to balance accuracy, stability, and speed—three qualities frequently in tension in fluid simulation literature Liu & Liu (2010). The final implementation not only validates these principles but also contributes a solid base for further exploration into hybrid rendering, thermodynamic modelling, and complex boundary interaction.

This chapter reflects on the project's success in meeting its objectives, evaluates the development process, and outlines directions for future research and practical application.

Summary of Achievements

The project successfully met its core objectives, delivering a fully functional, GPU-accelerated, real-time simulation of multi-phase fluids based on Smoothed Particle Hydrodynamics (SPH). Each goal outlined at the outset of the project was addressed as follows:

- **Mathematical Foundations of SPH:** The core physical and numerical models—density estimation, pressure and viscosity forces, and kernel interpolation—were implemented following established SPH theory Monaghan (2005),

Liu & Liu (2010). These formed the foundation of the simulation and were validated through dynamic and static test cases.

- **Application of Contemporary Techniques:** The system architecture was shaped by a review of current simulation methods, particularly those leveraging GPU parallelism and Lagrangian frameworks. While advanced methods such as Predictive-Corrective SPH were reviewed, implementation remained within a stable, real-time capable SPH model.
- **GPU Optimisation:** Key computations, including neighbour searches and force accumulation, were offloaded to compute shaders. Optimisations such as uniform spatial grids and adaptive time-stepping were incorporated to sustain high performance on consumer-grade hardware.
- **Multi-phase Interaction:** Fluids with differing densities and viscosities were simulated concurrently, maintaining interface separation and demonstrating effects such as buoyancy and immiscibility. These results confirmed the plausibility of the SPH-based multi-phase model.
- **Real-Time Visualisation:** The system included a custom rendering pipeline using particle-based colour mapping and screen-space lighting. Although a continuous fluid surface was not implemented, the visual output provided a cohesive and responsive experience suitable for interactive use.

Collectively, these outcomes demonstrate the successful fulfilment of the project's aims and provide a technically robust and extensible framework for real-time fluid simulation.

The real-time architecture reflects ongoing research into hybrid approaches for balancing visual fidelity and computational cost Yan et al. (2009), Harris (2004). While the simulation left out surface reconstruction or grid coupling, its responsiveness and stability under multi-phase conditions show good potential for integration into interactive applications.

The chosen techniques reflect those explored in recent SPH-based multiphase systems where visual clarity and physical realism are optimised Macklin & Müller (2013), Suresh et al. (2019). This demonstrates the value of using lightweight but stable fluid models in scenarios that are constrained by hardware limitations or runtime interactivity.

Reflection on Project Process

The project followed an iterative, Agile-inspired methodology, allowing for continuous development, testing, and refinement. This approach aligned well with the experimental nature of the work, where system behaviour often had to be observed before finalising design decisions. Development sprints were centred around discrete functionality—such as implementing SPH forces, visual rendering, or performance optimisation—each culminating in a functional test build.

Early prototyping and modular design were instrumental in managing technical risk. By separating compute logic from rendering and simulation control, individual components could be debugged and iterated in isolation. This modularity also

facilitated the addition of features like multi-phase interactions and adaptive time-stepping without requiring significant architectural changes.

Time management was generally effective, though certain aspects, such as fine-tuning the visual output and ensuring multi-phase stability, required more time than initially planned. This reflects the broader challenge of balancing research-driven experimentation with milestone-driven delivery. Nonetheless, consistent progress tracking and flexible goal adjustment helped maintain momentum throughout the development lifecycle.

The process demonstrated the importance of responsiveness in technical projects. The ability to adjust implementation details based on performance observations and visual output was a key factor in the project's overall success.

Recommendations for Future Work

While the current implementation successfully meets its objectives, there remain several areas for further technical development and research.

Boundary Geometry Enhancements

The simulation domain is currently constrained to a cubic boundary, chosen for its simplicity in grid handling and collision detection. However, this limits the variety and realism of simulated scenarios. Future work could introduce support for arbitrary geometries, such as cylindrical, spherical, or mesh-defined boundaries. Implementing Signed Distance Fields (SDFs) or surface-conforming constraints would enable more complex interactions with curved or dynamic environments. Techniques like ghost particle boundaries could also be refined to improve the accuracy of fluid–boundary interaction without introducing significant computational overhead.

Improved Visual Representation of Fluids

Fluids are presently visualised as discrete particles, which, while effective for debugging and validation, lack the visual continuity associated with real liquid surfaces. Enhancing the visual output with surface reconstruction techniques—such as marching cubes, point splatting, or screen-space meshing—would produce more cohesive fluid bodies. Incorporating post-processing effects like refraction, reflection, and foam would further elevate visual fidelity, particularly for entertainment or cinematic applications.

Advanced Multi-phase Simulation

The current system supports basic immiscible multi-phase interactions. Future extensions could explore complex phenomena such as phase transitions (e.g., evaporation or condensation), temperature-dependent properties, or fluid–solid coupling. Integrating thermodynamic models or supporting non-Newtonian fluids would broaden

the scientific scope of the simulation and expand its application to engineering, biomedical, or geophysical domains Prosperetti & Tryggvason (2009).

Applications and Extensions

The successful development of a real-time, GPU-accelerated multi-phase fluid simulation system opens up a broad range of potential applications and future extensions across both industry and research domains.

Interactive Entertainment and Game Development

Real-time fluid simulation plays an increasingly significant role in enhancing immersion within video games and interactive media. The framework developed in this project could be integrated into game engines to simulate dynamic water bodies, destructible environments, or interactive fluid puzzles Engel (2012). The multi-phase capabilities support effects such as oil–water separation, blood splashes, or lava flows, which are particularly relevant in narrative-heavy or survival-themed game genres.

Virtual and Augmented Reality

In virtual and augmented reality environments, physical responsiveness of fluids to user interaction can greatly enhance realism. This system's performance-oriented design is particularly suited for VR constraints, offering opportunities for applications such as virtual lab training, chemical simulation Macklin & Müller (2013), or emergency response scenarios involving fluid-based hazards.

Scientific and Educational Visualisation

The system can serve as a tool for teaching or visualising fundamental fluid dynamics concepts, including surface tension, buoyancy, and immiscibility. With further calibration, the framework could be employed in academic or industrial research to support real-time experimentation, hypothesis testing, or demonstration of complex fluid behaviours.

Closing Remarks

This project has been both a technical and personal milestone. It brought together advanced concepts in fluid dynamics, GPU programming, and interactive systems into a cohesive, functioning application. The challenges encountered—ranging from simulation instability to visual fidelity—offered valuable opportunities for problem-solving and technical growth.

The completed system not only validates the feasibility of real-time, multi-phase SPH simulation on consumer hardware, but also demonstrates the potential of combining rigorous computational models with efficient visual rendering. The outcomes reflect both the depth of research undertaken and the adaptability of the development process.

Looking forward, this work lays a solid foundation for further exploration into more sophisticated simulations and cross-disciplinary applications. It is a meaningful contribution to the field of real-time graphics and a significant step in my journey as a developer and researcher.

Bibliography

Batchelor, G. K. (2000), *An Introduction to Fluid Dynamics*, first cambridge mathematical library edition edn, Cambridge University Press, Cambridge, UK.

Blazek, J. (2015), *Computational Fluid Dynamics: Principles and Applications*, Elsevier.

Darrigo, O. (2002), 'Between hydrodynamics and elasticity theory: The first five births of the navier-stokes equation', `https://www.jstor.org/stable/41134138`.

Dharma, D. & Kistijantoro, A. I. (2017), Material point method based fluid simulation on gpu using compute shader, *in* '2017 International Conference on Advanced Informatics: Concepts, Theory and Applications (ICAICTA)'.
URL: *https://www.researchgate.net/publication/319525082*

Engel, W., ed. (2012), *GPU PRO 3: Advanced Rendering Techniques*, 1 edn, A K Peters/CRC Press. Accessed: 08-04-2025.
URL: *https://doi.org/10.1201/b11642*

Ferziger, J. H. & Perić, M. (2002), *Computational Methods for Fluid Dynamics*, third, revised edition edn, Springer. Accessed 09-03-2025.

Harada, T., Koshizuka, S. & Kawaguchi, Y. (2007), 'Smoothed particle hydrodynamics on gpus', *The Visual Computer* .

Harris, M. J. (2004), Fast fluid dynamics simulation on the gpu, *in* 'GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics', Addison-Wesley, chapter 38. Accessed 09-03-2025.
URL: *https://developer.nvidia.com/gpugems*

Hendra, A. (2015), Accelerating fluids simulation using sph and implementation on gpu, Master's thesis, Uppsala University. Accessed 09-03-2025.
URL: *https://www.diva-portal.org/smash/get/diva2:885188/FULLTEXT01.pdf*

Huang, C., Wang, X., Liu, Q. & Wang, H. (2023), 'Three-dimensional numerical simulation of large-scale landslide-generated surging waves with a gpu–accelerated soil–water coupled sph model', *ICCES* **25**(1).

Ihmsen, M., Akinci, N., Gissler, M. & Teschner, M. (2010), Boundary handling and adaptive time-stepping for pcisph, *in* 'Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS)', The Eurographics Association, pp. 79–88.
URL: *https://www.researchgate.net/publication/221622694*

Ihmsen, M., Orthmann, J., Solenthaler, B., Kolb, A. & Teschner, M. (2014), Sph fluids in computer graphics, *in* S. Lefebvre & M. Spagnuolo, eds, 'Eurographics 2014 - State of the Art Reports', The Eurographics Association. Accessed: 08-04-2025.
URL: *https://doi.org/10.2312/egst.20141034*

Liu, M. B. & Liu, G. R. (2010), 'Smoothed particle hydrodynamics (sph): an overview and recent developments', *Archives of Computational Methods in Engineering* **17**, 25–76. Accessed 09-03-2025.

Liu, S. & Nistor, I. (2018), 'Evaluation of the solid boundary treatment methods in sph', *International Journal of Ocean and Coastal Engineering* .
URL: *https://www.researchgate.net/publication/329939541*

Macklin, M. & Müller, M. (2013), 'Position based fluids', *ACM Transactions on Graphics* **32**(4), Article 104. Accessed 09-03-2025.

Monaghan, J. J. (2005), 'Smoothed particle hydrodynamics', *Reports on Progress in Physics* **68**(8), 1703–1759.

Müller, M., Charypar, D. & Gross, M. (2003), Particle-based fluid simulation for interactive applications, *in* D. Breen & M. Lin, eds, 'Eurographics/SIGGRAPH Symposium on Computer Animation', The Eurographics Association. Accessed 09-03-2025.
URL: *https://www.eg.org/*

Pressman, R. S. & Maxim, B. R. (2014), *Software Engineering: A Practitioner's Approach*, 8th edn, McGraw-Hill Education, New York.

Prosperetti, A. & Tryggvason, G. (2009), *Computational Methods for Multiphase Flow*, Cambridge University Press.
URL: *https://books.google.co.uk/books?id=KBuKZkEUWMIC*

Resolved Analytics (n.d.), 'The history of computational fluid dynamics', `https://www.resolvedanalytics.com/cfd/history-of-cfd`. Accessed 09-03-2025.

Solenthaler, B. & Pajarola, R. (2009), 'Predictive-corrective incompressible sph', *ACM Transactions on Graphics* **28**(3), Article 40. Accessed 09-03-2025.

Sommerville, I. (2011), *Software Engineering*, 9th edn, Addison-Wesley, Boston.

Suresh, P., Kumar, S. P. & Pantaik, B. (2019), 'A comparative study of two different density estimation techniques for multi-phase flow simulations using sph', *International Journal for Computational Methods in Engineering Science and Mechanics* **20**(1), 29–47.

Taylor, P. A. & Miller, J. C. (2012), 'Measuring the effects of artificial viscosity in sph simulations of rotating fluid flows', *Monthly Notices of the Royal Astronomical Society* **426**, 1687–1700.

Tessendorf, J. (2004), Simulating ocean water, Technical report, Fine Light Visual Technology. Accessed 09-03-2025.
URL: *http://www.finelightvisualtechnology.com*

Tianren, C. (2015), 'Reference (tfsi 2012-2013)'. Dataset, accessed 09-03-2025. **URL:** *https://www.researchgate.net/publication/281612631*$_{R}eference_{T}FSI_{2}012-$2013

Tryggvason, G., Scardovelli, R. & Zaleski, S. (2011), *Direct Numerical Simulations of Gas-Liquid Multiphase Flows*, Cambridge University Press. Accessed 09-03-2025.

Unity Technologies (2020), 'ShaderLab Reference', `https://docs.unity3d.com/2020.1/Documentation/Manual/SL-Reference.html`. Unity Manual.

Unity Technologies (2024*a*), 'ComputeBuffer', `https://docs.unity3d.com/6000.0/Documentation/ScriptReference/ComputeBuffer.html`. Unity Scripting API.

Unity Technologies (2024*b*), 'ComputeShader', `https://docs.unity3d.com/Manual/class-ComputeShader.html`. Unity Manual.

Unity Technologies (2024*c*), 'GPU Instancing', `https://docs.unity3d.com/Manual/GPUInstancing.html`. Unity Manual.

Unity Technologies (2024*d*), 'Universal Render Pipeline (URP) Introduction', `https://docs.unity3d.com/6000.2/Documentation/Manual/urp/urp-introduction.html`. Unity Manual.

Yan, H., Wang, Z., He, J., Chen, X., Wang, C. & Peng, Q. (2009), 'Real-time fluid simulation with adaptive sph', *Computer Animation and Virtual Worlds* **20**, 417–426.

Appendices

Appendix 1: Personal Appraisal

This project provided a valuable opportunity to apply and expand my technical skills while tackling a problem I was genuinely interested in. Although I already had experience with GPU programming, I had very limited knowledge of Unity at the start. Learning the Unity environment and integrating my compute-based simulation into it was something I had to pick up quickly. Fortunately, I didn't find this too challenging, but it was still a necessary learning curve.

Coming from a C++ background, adapting to C presented a few small hurdles, especially around syntax and Unity's component-based workflow. However, since both languages are C based, the transition was fairly smooth.

The most challenging part of the project by far was developing the shader file for particle rendering. I had very little prior knowledge of shader programming, and combining that with a new engine like Unity made it even more difficult. I spent a considerable amount of time trying to understand how Unity's rendering pipeline interacts with custom compute buffers and materials.

Although I'm pleased with the outcome, I would have liked to incorporate more visual realism into the fluid rendering. My limited shader knowledge restricted the fidelity I could achieve, and this is an area I'd like to explore further in future work.

Overall, I've strengthened my understanding of SPH, improved my ability to integrate low-level GPU code into a real-time engine, and learned how to overcome unfamiliar technical challenges through structured experimentation and problem-solving.

Appendix 2: IPO Form and Feedback

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Cameron Pearson - 40530119

Title of Project:

Real-Time Multi-Phase Fluid Simulation

1. Overview of Project Content and Milestones

My project will focus on developing a real-time multi-phase fluid simulation in which I plan to use Smoothed Particle Hydrodynamics (SPH) and GPU acceleration. The project will be developed using C++ in Visual Studio, using specific libraries and GPU acceleration to ensure performance. It will consist of two parts: an extensive literature review and the development of the simulation.

Key stages include:

- **Literature Review:** An analysis of research papers and technical reports on SPH, multi-phase fluid simulations, and GPU-accelerated techniques. The review will focus on identifying both well-established and newer methods, guiding the development process.
- **Design and Development:** Using C++, design and implement the simulation framework with GPU support for real-time performance.
- **Optimization:** Implement methods to ensure that the simulation runs at a high frame rate whilst also correctly simulating the particles interactions, particularly when simulating the large particle systems and handling multi-phase interactions.
- **Testing and Evaluation:** Ensure the simulation's real-time performance is run at a high frame rate, especially in handling interactions between different fluid phases, such as water and oil, and ensuring these interactions are realistic

2. The Main Deliverable(s)

- A real-time fluid simulation program developed in C++ using GPU acceleration, focused on multi-phase interactions and SPH.
- A literature review document summarising previous, current and relevant research in SPH, GPU acceleration techniques and multi-phase fluid interactions, and how this project adds to this field.

3. The Target Audience for the Deliverable(s)

- Researchers in fluid dynamics and computational graphics.
- Game developers and visual effects professionals interested in real-time fluid simulations.
- Educational institutions for teaching advanced real-time simulation techniques.

4. The Work to be Undertaken

This project is divided into two main phases:

Literature Review:

A major part of the project will be reviewing existing research papers on Smoothed Particle Hydrodynamics (SPH), multi-phase fluid simulations, and GPU optimization. Specific attention will be paid to papers that discuss:

- **Numerical Methods:** Understanding how SPH handles fluid dynamics, pressure, and density, especially in multi-phase scenarios.
- **Real-Time Simulation Techniques:** Papers detailing real-time GPU implementations and optimisations for fluid simulations.
- **Challenges in Multi-phase Fluids:** papers discussing the challenges of simulating multi-phase fluids like water and oil, and existing solutions for these challenges.

Development and Implementation:

The core of the project will involve implementing the simulation using C++ in Visual Studio. Key libraries and technologies that will be used include:

- CUDA or OpenCL for GPU acceleration.
- GLM (OpenGL Mathematics) for vector and matrix operations.
- Thrust (part of CUDA) for parallel algorithms and memory management.
- OpenGL for real-time rendering of the simulation.
- SFML or GLFW for window management and user input, depending on visual requirements.

The development will focus on:

- Implementing SPH.
- Ensuring stability and realistic fluid interactions through correct integration and boundary handling.

5. Additional Information / Knowledge Required

- **C++ Development and Libraries:** I will need to become proficient in CUDA or OpenCL for GPU programming, as well as math libraries such as GLM for vector/matrix operations. Understanding how to interface with GPU libraries from C++ will be important for optimising performance.
- **Literature on Multi-phase Fluids and SPH:** I will need to read papers that discuss SPH and its implementation in detail, especially how it has been adapted for multi-phase fluids.
- **Numerical Stability and Fluid Dynamics:** I will need a strong understanding of numerical methods to ensure the simulation remains stable, particularly with complex multi-phase fluid interactions.

6. Information Sources that Provide a Context for the Project

- **Academic Papers:** These will include foundational SPH research as well as GPU-accelerated fluid simulations from recent papers that I can find from sources such as google scholar.
- **Books:** "Real-Time Rendering" and "Fluid Simulation for Computer Graphics" can provide in-depth technical knowledge on graphics and fluid dynamics.
- **Online Tutorials and Open-Source Libraries:** GitHub repositories such as NVIDIA's Flex SDK can offer practical insights into real-world implementations of SPH and GPU-accelerated techniques.

7. The Importance of the Project

This project will contribute to the field of real-time multi-phase fluid simulation by addressing the challenge of simulating complex fluid interactions in real time using C++ and GPU acceleration. While SPH is a well-established technique, this project becomes more complex with its application of GPU-accelerated techniques to efficiently manage large-scale particle interactions, specifically in multi-phase fluid systems like I am planning to develop. This has direct applications in industries like gaming and visual effects, where fluid dynamics are computationally intensive and crucial for realism.

Figure 8.1: Initial Project Overview pages 1–4

8. The Key Challenge(s) to be Overcome

- **Extensive Research:** Turning existing research into a functional and innovative solution will be challenging, especially with the range of literature on SPH, multi-phase fluids, and GPU optimisations. The literature review must assess current methods and identify gaps that this project can address.
- **C++ and GPU Programming:** Implementing a fluid simulation in C++ while ensuring effective GPU utilisation will be complex. Efficient memory management, algorithm parallelisation, and numerical stability will require careful attention.
- **Achieving Real-Time Performance:** Balancing accuracy and performance will be challenging, particularly when simulating interactions between different fluids. As a realistic simulation will be very computationally expensive.

Figure 8.2: Initial Project Overview page 5

Is the project of the required standard? (if not, what changes are required?)	Yes the project is of a suitable standard for the Honours module
Is the project viable? (if not, what changes are required?)	The project is viable, the goal is clear cut and there are suitable resources out there
Does the project provide a realistic challenge for the student? (if not, what changes are required?)	Yes the project will be a suitable challenge
Is the project appropriate for the student's programme? (consult project guidance - if not, what changes are required?)	Yes, the proposed work is very much inline with the programme
Is the student adequately supported? (if not, what changes are required?)	Yes the student is adequately supported
Any other feedback?	

Figure 8.3: IPO Feedback

Appendix 3: Interim Report Feedback

SOC10101 Honours Project (40 Credits)

Week 9 Report

Student Name: Cameron Pearson
Supervisor: Iain Donald
Second Marker: Dimitrios Darzentas
Date of Meeting: 20/1/25

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **yes no***

If not, please comment on any reasons presented

Diary sheets were not required by original supervisor, this will be rectified going forwards

Please comment on the progress made so far

The artefact implementation is progressing very well
The written work has a complete literature review, but needs dedicated writing time

Is the progress satisfactory? **yes no***
Can the student articulate their aims and objectives? **yes no***
If yes then please comment on them, otherwise write down your suggestions.

The student is very familiar with their work and can very clearly articulate the aims, objectives and work so far. The literature review covers what would be expected and the student has coded a rudimentary framework for their artefact.

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Does the student have a plan of work? **yes no***
If yes then please comment on that plan otherwise write down your suggestions.

Due to unavoidable circumstances outside of the students' control, and supervision changes, the IRO meeting has been delayed by approx. two months which has impacted the student's planned work. There is a plan to achieve the work within the timeframe of the module, however progress would very likely have been much further along if full support that been in place for the student.

Does the student know how they are going to evaluate their work? **yes no***
If yes then please comment otherwise write down your suggestions.

There is a clear plan to evaluate the system using synthetic benchmarks

Any other recommendations as to the future direction of the project

Additional discussion on stretch goals were made in regards to making the code more employer and user accessible – these are stretch goals.

Signatures:
Supervisor: Iain Donald
Second Marker: D.Darzentas
Student: Cameron Pearson
The student should submit a copy of this form to Moodle immediately after the review meeting; A copy should also appear as an appendix in the final dissertation.

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Figure 8.4: Enter Caption

Appendix 4: Project Diary

Project Diary Timeline

Cameron Pearson (40530119)

Weekly Timeline

Date	Supervisor	Objectives for Next Week	Progress & Meeting Discussions
30/09/2024	Babis Koniaris	Establish dissertation structure, define research topics, and gather relevant papers.	Introductory meeting with Babis. Discussed overall project scope, importance of SPH in real-time fluid simulation, and initial research areas. Started outlining dissertation structure and sourcing academic papers.
07/10/2024	Babis Koniaris	Research SPH fundamentals and software tools (Unity, Unreal, or a custom engine).	Meeting with Babis: Reviewed key SPH papers, compared software tools. Decided Unity is the best option due to its compute shader support. Explored Unity's physics system and how SPH could be implemented.
14/10/2024	Babis Koniaris	Study SPH algorithms, understand mathematical foundations and computational challenges.	Meeting with Babis: Covered Navier-Stokes equations, Lagrangian vs Eulerian methods, and SPH simplifications. Spent the week breaking equations into implementable steps.
21/10/2024	Babis Koniaris	Begin writing the literature review, structuring it around SPH's historical development and real-time applications.	Meeting with Babis: Structured literature review to include historical context, SPH methodology, and GPU acceleration techniques. Started writing early SPH development and game physics applications.
28/10/2024	Babis Koniaris	Expand literature review with real-time performance considerations and SPH comparisons.	Meeting with Babis: Reviewed SPH performance issues, especially neighbour search optimisations. Read papers on efficient computation for real-time SPH.
04/11/2024	Babis Koniaris	Finalise core literature review sections, explore GPU-based SPH implementations.	Meeting with Babis: Discussed GPU acceleration's impact, compared CUDA and OpenCL but decided to focus on Unity's compute shaders.

1

Date	Supervisor	Objectives for Next Week	Progress & Meeting Discussions
11/11/2024	Babis Koniaris	Summarise literature review findings, prepare a first draft, and investigate GPU-based SPH simulations.	Meeting with Babis: Shared a rough draft of literature review, identified missing areas on real-time SPH advancements. Explored GPU-based methods.
18/11/2024	Babis Koniaris	Plan project implementation steps and refine research focus.	Meeting with Babis: Reviewed refined research questions and how to transition from theory to implementation. Planned a development timeline.
25/11/2024	Ian Donald	Meet with Ian Donald to discuss project direction, finalise literature review.	Supervisor changed to Ian Donald. Scheduled meeting to discuss project feasibility and methodology.
02/12/2024 - 06/01/2025	Ian Donald	Work on implementation over the Christmas break, finalise literature review.	December was unproductive due to coursework and taking time off to relax. Over the break, finished the literature review and started basic SPH implementation in Unity.
13/01/2025	Ian Donald	Develop a basic SPH solver, ensuring particle interactions work correctly.	Built basic SPH framework in Unity, tested initial particle interactions, and worked on rendering.
20/01/2025	Ian Donald	Investigate performance bottlenecks, begin optimising SPH for real-time simulation.	**Meeting with Ian:** Discussed CPU performance issues. Ian suggested spatial partitioning. Explored spatial hashing and acceleration structures.
27/01/2025	Ian Donald	Implement GPU acceleration using Unity compute shaders, compare against CPU performance.	Implemented basic GPU acceleration using compute shaders. Performance improved but needed further refinement.
03/02/2025	Ian Donald	Refine GPU acceleration, implement optimised neighbour search algorithms.	Neighbour search improved using spatial partitioning. Added grid-based particle sorting for better performance.
10/02/2025	Ian Donald	Implement better boundary conditions and improve simulation stability.	**Meeting with Ian:** Reviewed progress, discussed boundary handling issues. Ian suggested refining collision detection. Adjusted simulation parameters for stability.
17/02/2025	Ian Donald	Implement multi-phase fluid interactions, ensuring stable simulation behaviour.	Started multi-phase SPH by assigning different densities and viscosities to particles. Faced issues with phase separation.

2

Date	Supervisor	Objectives for Next Week	Progress & Meeting Discussions
24/02/2025	Ian Donald	Research and implement techniques for improving phase separation and surface tension effects.	**Meeting with Ian:** Discussed multi-phase instability issues. Ian suggested modifying kernel functions for better interfacial force handling. Improved density contrast calculations.
03/03/2025	Ian Donald	Continue refining phase separation, begin methodology section of dissertation.	Improved phase interactions, started methodology section, documenting design choices and implementation steps.
10/03/2025	Ian Donald	Expand methodology section, begin implementation write-up.	**Meeting with Ian:** Discussed methodology structure, justification of design decisions. Implemented additional visual improvements for fluid rendering.
17/03/2025	Ian Donald	Work on evaluation framework, prepare test cases for performance validation.	Set up benchmarking tools in Unity to analyse performance under different particle counts. Documented early findings.
24/03/2025	Ian Donald	Conduct GPU performance testing and write performance evaluation section.	**Meeting with Ian:** Ian lent his laptop with an RTX 3080 for performance testing. Ran benchmarks with various particle counts and documented the GPU vs CPU results. Completed the performance evaluation section.
31/03/2025	Ian Donald	Finalise dissertation structure and ensure all sections are coherent.	Completed conclusion, reviewed the complete dissertation. Discussed structure, clarity, and flow. Agreed on minor edits to improve transitions and consistency.
07/04/2025	Ian Donald	Polish dissertation formatting, add final figures and refine references.	Focused on final formatting tweaks, adding missing figures, cross-referencing, and ensuring reference consistency. Proofread entire document for clarity.

3

Figure 8.5: Project diary showing weekly supervisor meetings