Real-time Interactive Simulation of Diverse Particle-Based Fluids

Niall Tessier-Lavigne

Year 4 Project
School of Informatics
University of Edinburgh
2017
I wish to thank the following friends:

My supervisor, **Taku Komura**, for encouragement throughout

**Sean Tann**, for his invaluable help in GPU-wrangling

**Jake Browning**, for his stoic and unwavering guidance.
Abstract

In this project, we create a real-time physics simulation modelling a diverse range of fluid behaviour. Our simulator is capable of being interacted with in real-time with a high level of simulation detail. We robustly handle complex fluid behaviour including varying viscosity, visco-elastic and plastic deformation, and multi-phase interactions. We provide an overview of several techniques to model differing fluid material properties, and arrive at a stable simulation which can be easily authored to produce visually compelling results. We give both a performant CPU-based and fully parallelised GPU implementation.

Moreover, we implement several techniques to expand the range of material properties capable of being simulated. We extend existing models for surface tension and multi-phase fluids, adding support for varying miscibility, and reformulate a visco-elasticity model to be unconditionally stable and physically motivated. We additionally present a novel optimization strategy to adaptively adjust constraint iterations per-particle, improving computation time without sacrificing visual fidelity or stability.
Contents

Abstract

1 Introduction

1.1 Background
  1.1.1 Motivation and Project Goals
  1.1.2 Overview of Simulation Methods
  1.1.3 Considerations and Contributions

1.2 Previous Work

2 Smoothed Particle Hydrodynamics

2.1 Method
  2.1.1 Governing Equations of Fluid Motion
  2.1.2 Introduction to the SPH model
3.7 Viscoelastic Fluids ........................................ 35
  3.7.1 Background ........................................... 35
  3.7.2 Previous Work ........................................ 36
  3.7.3 Adaptation to PBF .................................... 37
3.8 Multiple Phase Fluid Simulation ............................. 38
  3.8.1 Implementation ....................................... 38
  3.8.2 Surface Tension and Miscibility in Multiphase Simulations 39
3.9 Discussion .................................................. 40

4 Implementation and Optimisation ........................... 43
  4.1 Overview ............................................... 43
  4.2 CPU Implementation ..................................... 44
  4.3 GPU Implementation ..................................... 45
    4.3.1 Position Based Dynamics on the GPU ............... 46
  4.4 GPU Neighbourhood Search ............................... 47
    4.4.1 Overview .......................................... 47
    4.4.2 Implementation ..................................... 47
    4.4.3 Discussion ......................................... 49
  4.5 Slackening: A Novel Adaptive Constraint Method .......... 50
    4.5.1 Overview .......................................... 50
    4.5.2 Implementation ..................................... 50
    4.5.3 Discussion ......................................... 52
  4.6 Rendering ............................................... 52
5 Results

5.1 Guide to Supplemental Material

5.1.1 Testing Environment

5.2 Smoothed Particle Hydrodynamics

5.2.1 Overview

5.2.2 Analysis of Compressibility

5.3 Position Based Fluids

5.3.1 Compressibility

5.3.2 Surface Tension

5.3.3 Viscosity

5.3.4 Viscoelasticity

5.3.5 Multiple Phases

5.3.6 Slackened Position Based Fluids

6 Conclusion

6.1 Summary of Work Completed

6.2 Motivation for Future Research

Bibliography
Chapter 1

Introduction

1.1 Background

1.1.1 Motivation and Project Goals

Fluid simulation is a widely studied topic in computer graphics, with a large array of applications ranging from visual effects to medical simulations and training environments. It is often a computationally intensive problem domain, and compromises must typically be made between computation time, visual fidelity and simulation stability.

Interactive contexts are a challenge due to high demands on simulation speed and limitations on time-step. The focus of this dissertation is on a real-time application of fluid simulation, and we are thus concerned with simulation methods robust and performant enough to handle user interaction at interactive frame rates.

We prioritise computational efficiency and simulation stability over physical accuracy, and choose to focus on techniques that plausibly and demonstrably resemble real-world fluid behaviour. We arrive at a simulator capable of simulating fluids with the following properties:

- Position-Based Fluids implementation
- Wide range of supported viscosity
• Visco-elasto-plastic deformation with physically derived values
• High surface tension fluids
• Multi-phase fluid-fluid interaction
• Authorable miscibility between fluid phases
• Vorticity confinement for encouraging turbulent motion
• Fully parallelised solver
• Simulation-level adaptive constraint optimisation

1.1.2 Overview of Simulation Methods

The computational simulation of fluid dynamics can be broadly divided into three approaches: Eulerian, Lagrangian and Semi-Lagrangian. In the Eulerian approach, fluid quantities such as density and pressure are discretised on a grid, with each cell containing some amount of ‘fluid material’. In contrast, the Lagrangian approach models fluids as a system of particles and solves inter-particle forces to achieve fluid behaviour. Semi-Lagrangian schemes make use of both representations, inferring fluid properties from both an underlying grid and particles, propagating data between the two.

All three approaches have achieved success in real-time simulation. In this project we have chosen to focus on the Lagrangian approach.

Treating the simulation domain as a system of particles provides us with several advantages. Eulerian approaches are ill-equipped to model free fluid flow, as the simulation domain is restricted to the grid. Large-scale scenes can become quickly inefficient as simulation detail is a direct consequence of the grid resolution. Lagrangian simulations do not suffer from this, as visual fidelity is a consequence of the particle resolution, and the simulation domain can be unbounded without compromising in detail. This allows us to simulate fluids at greatly varied scale.

Furthermore, Lagrangian approaches are easily integrable with existing physics simulation software, which typically model environments using meshes, rather than voxelized space more readily suited to grid-based approaches.
The choice of a particle-based method also allows us to build on the flexibility of the Position Based Dynamics method (Müller 2006), which has seen success in modeling a large range of physical phenomena. This means our implementation can be readily integrated into existing and future Position Based Dynamics research.

1.1.3 Considerations and Contributions

As our focus is on real-time simulation, computational efficiency plays a large role in our evaluation of the efficacy of certain approaches. Particle-based approaches to fluid simulation are often hindered by the computationally expensive neighbour-search process required in each time step. There are several ways to improve upon this through the use of broadphase acceleration structures, which we detail in chapter 4.

Our GPU implementation of the particle neighbour search leverages its parallel nature, which is a key to our high resolution 3D real-time results. We implement a GPU-based Counting Sort algorithm to gather particle neighbourhoods, and achieve a large speedup over the standard uniform grid based spatial subdivision of our CPU implementation.

In addition to our work implementing various Lagrangian fluid models, we present an entirely novel way to perform adaptive time-stepping based on particles’ relative density fluctuation, achieving further speed-up while maintaining simulation stability.

In total, our work and contributions are the following:

• Simulation 2- and 3-dimensional scenes with a large range of fluid materials in real-time

• Implementation of vorticity confinement, reducing system damping and encouraging turbulent motion

• Robust handling of multi-phase fluids

• Support for elasto-plastic deformation using mass-spring systems

• Implementation of several variations on both Smoothed Particle Hydrodynam-
ics and Position-Based Fluids

- Adaption of XPBD constraints to the model of (Clavet 2005) for physically motivated, unconditionally stable viscoelastic behaviour
- Extension of the surface tension model of (Akinci 2013) to allow fluids of varying miscibility to interact
- Parallelised fluid solver using Compute shaders for greatly enhanced performance
- Implementation of a fully parallel Counting Sort based neighbourhood search algorithm on the GPU
- Introduction of a novel adaptive Position-Based Fluids solver, improving computational efficiency while maintaining simulation stability and visual quality

1.2 Previous Work

1.2.1 Introduction and Context

We will begin our outline of previous work by looking at the development and subsequent refinement of the Smoothed Particle Hydrodynamics (SPH) method, before turning to the work of Position Based Dynamics and its support for fluid materials.

1.2.2 Smoothed Particle Hydrodynamics

SPH was first introduced by (Gingold 1977) to simulate problems in astrophysics. It was later adapted to simulating interactive fluids by (Müller 2003) and others. This early work showed promising results, but was limited in its ability to model small fluid volumes due to exhibiting undesirably compressible behaviour with interactively performant time-steps. It also introduced standard smoothing kernels, still widely used in particle-based models today.
1.2.3 Extensions and Improvements to SPH

There exist numerous extensions to the standard SPH formulation of fluids, allowing for a range of materials. Multiple fluid phases are handled by the model of (Müller 2003). (Solenthaler 2008) addressed the erroneous interface 'gap' that results from this model by providing adapted equations for determining density and pressure.

Surface tension is an important feature observed in fluids, not captured by the standard Navier-Stokes equations. Numerous augmentations to standard SPH exist to tackle this — (Morris 2000) and (Müller 2003) compute a surface tension force derived from the surface curvature, by evaluating gradients of a smoothed colour field. (Tartakovsky & Meakin 2005), (Becker 2007) instead look from a molecular perspective, modelling surface tension as cohesion forces between particles. This approach has the advantage of avoiding computation of surface normals, and avoids noise in the colour field in under-sampled areas (particles with few neighbours). It is further trivially momentum-conserving as forces are applied symmetrically. However, it does not explicitly attempt to minimise surface area as expected. (Akinci 2013) address this issue by providing both a cohesion and minimization term to their surface tension forces, though still relies on the computation of the smoothed colour field.

(Clavet 2005) present an alternative integration scheme for SPH, addressing surface tension, incompressibility and elasto-plastic behaviour. Their 'Double Density Relaxation' (DDR) scheme is not physically motivated, but provides visually pleasing results. DDR also solves the common tensile instability of SPH, in which particles tend to clump together undesirably. However, their predictive-corrective integrator is still only conditionally stable, with an asymmetric viscosity formulation and no explicitly modifiable surface tension parameter.

Other formulations of elasto-plastic deformation exist within SPH literature, motivated by a continuum-mechanics perspective. (Müller 2004) coupled SPH with an evolving strain tensor at particle locations. It does not however handle large-scale deformation such as fracturing, and degenerates in cases with few sample particles.

We adopt the elasto-plastic model of Clavet et al in our simulation due to its computational efficiency, and its simple formulation of viscoelastic behaviour. We later adapt this to our PBF simulation by reformulating the inter-particle springs into a position-based scheme.
1.2.4 Incompressibility in SPH

It is often desirable to model fluids as incompressible, to ensure there is no perceptible loss of volume. However, this restraint typically yields computationally stiff systems, requiring restrictively low time-steps. There are numerous approaches to maintaining visual coherency while achieving speedup, such as varying the resolution of the particle model across the simulation domain, and using a spatially adaptive resolution based on areas of visual interest.

Enforcing incompressibility is the focus of much SPH research, and numerous approaches to this problem exist. Several Incompressible SPH (ISPH) approaches formulate incompressibility as a Poisson equation, projecting variances into a divergence-free space. This however is typically too computationally expensive for real-time applications.

(Becker 2007) introduced Weakly Compressible SPH (WCSPH), addressing the incompressibility by a modified equation of state. This resulted in far less compressible results than standard SPH, at the cost of a tight restriction on time-steps for stable simulation. It is thus unsuitable for our aims.

Predictive-Corrective Incompressible SPH (Solenthaler 2008) instead achieves incompressibility by using a predictive-corrective iterative scheme, with a termination criterion based on a measure of density error, rather than providing an authorable stiffness variable characteristic as in Equation-of-State solvers. However, like WCSPH, it is too computationally expensive and restrictive on time-step value to be suitable for our purposes.

1.2.5 Position Based Fluids

Position Based Fluids (PBF) is a Lagrangian technique for simulating fluid flow, introduced by (Macklin 2013). It builds on the existing methodology of Position Based Dynamics (PBD) systems. PBD is an approach to the simulation of dynamic bodies built on a predictive-corrective numerical integrator. In PBD, collisions and constraints are built as a linear system and solved iteratively by geometrically projecting positions into a valid system configuration. Position Based Fluids reformulates the approach of SPH to enforce a constant density constraint across the fluid volume. It makes use of the same discretised
interpolation method of SPH, and as such many insights in the existing SPH literature also apply to PBF.

Despite its recent development, numerous improvements and additions to PBF have already been proposed. (Macklin 2014) integrated PBF into a wider framework of position based dynamics methods, adding support for coupling between fluids, rigid bodies and deformable solids. (Takahashi 2014) extended PBF to efficiently support high viscosity, elasticity and thermal conductivity. (Köster 2016) introduced a technique to improve computational efficiency by adaptively changing constraint iterations based on particles’ distance to the camera. (Barreiro 2017) offered a more sophisticated approach to viscoelastic PBF based on a notion of conformation constraints, further extending the range of materials that could be robustly simulated in real time.

We draw from several of the ideas presented in the abovementioned work. Though much of the recent work in PBF is beyond the scope of this project, we aim to develop a flexible and robust simulation that augments several different SPH- and PBF-based approaches. We begin our discussion by providing insight into Smoothed Particle Hydrodynamics, upon principles of which Position Based Fluids was later developed.
Chapter 2

Smoothed Particle Hydrodynamics

2.1 Method

2.1.1 Governing Equations of Fluid Motion

The governing equations of our approach to modeling flow is described by the Navier-Stokes equations for viscous fluids:

\[
\rho \frac{D\mathbf{u}}{Dt} = -\nabla \hat{p} + \mu \nabla^2 \mathbf{u} + \frac{1}{3} \mu \nabla (\nabla \cdot \mathbf{u}) + \rho \mathbf{g}. \tag{2.1}
\]

With pressure \( p \), density \( \rho \), velocity \( \mathbf{u} \), viscosity \( \mu \), and external forces such as gravity given by \( \mathbf{g} \). Physical properties such as fluid velocity and mass-density are considered as continuous fields that are advected and diffused through time.

These equations form the basis for many fluid solvers. Though the Navier-Stokes equations have no closed form analytic solution, they can be approached with numerical methods to discrete space and time.

Eulerian approaches to simulation aim to discretise the equations on a grid, typically evaluating terms like gradients using finite differences. The grid conceptually maps to a space of fluid-filled cells. (Bridson 2015)
In contrast, Lagrangian approaches discretise the fluid body as a set of points, or particles. Field quantities are typically evaluated at particle locations, interpolated based on particle neighbourhoods.

As given, the Navier-Stokes equations do not describe several aspects of observable fluid behaviour such as surface tension and elasto-plastic deformation. Such properties are typically augmented with Navier-Stokes solvers, tracking additional continuum properties such as stress, strain and surface curvature, injecting additional internal forces to approximate these effects.

### 2.1.2 Introduction to the SPH model

Smoothed Particle Hydrodynamics (SPH) is a Lagrangian simulation method. Originally introduced by (Gingold 1977) and (Lucy 1977) as a way to simulate problems in astrophysics, it has since been widely developed as a method for simulating fluid flow.

SPH represents the fluid material as a set of discrete particles. Quantities in the fluid body are interpolated at each particle position by considering local neighbourhoods, gathering nearby information and weighting values by radially symmetric kernels within some influence radius $h$ (Müller 2003).

In the SPH formulation, any fluid quantity $A$ at any spatial position $r$ can be evaluated as follows:

$$A(r) = \sum_j m_j \frac{A_j}{\rho_j} W(|r - r_j|, h),$$  

(2.2)

This equation sums over all particles within the support radius $h$. Here $m_j$ is the mass of particle $j$, and $A_j$ and $\rho_j$ are the quantity $A$ and density at particle $j$ respectively. $W$ is the weighted kernel function. The kernel function is radially symmetric with compact support given by $h$. It is chosen such that its falloff is proportional to the distance between point $r$ and $r_j$.

If the spatial derivative at location $r$ is needed, this affects only the kernel function used in evaluation:
\[ \nabla A(r) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(|r - r_j|, h). \] (2.3)

Typically, kernel functions vary depending on the particular quantity being evaluated. It is desirable for \( W \) to be symmetric and normalised, resulting in a second-order accurate interpolation (Müller 2003).

### 2.1.3 Choice of Interaction Radius

The choice of particle interaction radius \( h \) depends on several factors in an application. These include desired simulation detail (resolution), environment scale, and other computational constraints, such as real-time simulation with a strict time budget.

Additionally, a smaller support radius places further restriction on the maximum timestep of a simulation. A smaller support radius means a lower timestep must be used, to ensure stable simulation and correct behaviour.

As a result of these factors, a trade-off must be made between simulation resolution and computational efficiency. We understand a higher resolution simulation to be of higher visual quality, and aim to simulate scenes of detail comparable to existing work in the field.

Often the most computationally demanding aspect of a Lagrangian fluid simulation is the neighbourhood search. In each timestep of an SPH simulation, every single particle must evaluate the particles in its surrounding neighbourhood. As we use kernels with compact support radius, this can be greatly sped up through the use of acceleration structures such as uniform grids (Müller 2003). We detail our approach in section 4.

There exist several techniques to adaptively modify the simulation support radius, so as to reduce the amount of particles necessary for detailed simulation. Such approaches appear promising, but we regard them as outside the scope of this dissertation. We refer the reader to (Owen 1998) and (Winchenbach 2017) for examples.
2.1.4 Time Integration

There exist several techniques to integrate the equations of motion through time. Common choices are the second-order accurate *Leapfrog* integration technique as employed by (Müller 2003), and Semi-implicit Euler integration. Leapfrog integration takes the following form:

\[
x_{i+1} = x_i + v_i \Delta t + \frac{1}{2} a_i \Delta t^2 \\
v_{i+1} = v_i + \frac{1}{2} (a_i + a_{i+1}) \Delta t
\] (2.4)

and Semi-implicit Euler integration given as follows:

\[
v_{i+1} = v_i + a_i \Delta t \\
x_{i+1} = x_i + v_{i+1} \Delta t
\] (2.5)

Where \( a_i \) is the acceleration acting on the body at timestep \( i \), and \( v \) and \( x \) are velocity and position, respectively. In our SPH simulations we opt for the Semi-implicit Euler integration scheme, as it does not require storing an additional acceleration variable unlike Leapfrog integration. We note however that Leapfrog integration may result in slightly more stable behaviour as it is a second-order method, at the expense of increased memory usage.

2.2 SPH for Fluid Simulation

2.2.1 Kernel Functions

It is typical to use separate kernel functions for different stages of simulation in order to improve simulation quality. Though many kernels are well-established in the literature, there is no definitive consensus on optimal kernels for SPH simulation (Ihmsen 2014).

We use the following kernels given by (Clavet 2005), taking the following form:

\[
W(r_{ij}, h) = \begin{cases} 
(1 - \frac{r_{ij}}{h})^2, & \text{if } |r_{ij}| \leq h \\
0, & \text{otherwise}
\end{cases}
\] (2.6)
\[ \nabla W(r_{ij}, h) = \begin{cases} 
(1 - \frac{r_{ij}}{h}), & \text{if } |r_{ij}| \leq h \\
0, & \text{otherwise} 
\end{cases} \] (2.7)

These kernels can be computed very efficiently, and exhibit the desired properties of being normalised and symmetric. They also have the advantage of being dimension-agnostic, unlike those of (Müller 2003), which require additional normalisation constants that depend on the simulation dimension.

We refer the reader to (Ihmsen 2014) for further elaboration on potential choices of kernel functions.

With our chosen kernel functions, we are now ready to compute the required SPH forces for fluid simulation.

### 2.2.2 Pressure

Following the standard SPH quantity evaluation equations of section 2.1.2 in equation 2.2 yields assymetric pressure forces. We instead opt to use the established symmetrised equation for both pressure given by (Solenthaler 2008), motivated by its momentum conserving properties and improved multi-phase stability. The pressure equation is thus given as:

\[
f_i^{\text{pressure}} = - \sum m_i m_j \left( \frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W(r_{ij}, h) \] (2.8)

Where \( r_{ij} \) is the distance vector between particles \( i \) and \( j \), \( h \) is the support radius, and \( W \) is the SPH kernel function. Pressure values \( p_i \) and \( p_j \) determined via an equation of state for the fluid, detailed in section 2.2.3.

### 2.2.3 Equation of State

The equation of state is used in our implementation of SPH fluids to determine the pressure at an individual particle’s location in equation 2.9. A common choice for fluids and deformable bodies is given by (Desbrun, Cani 1996) as follows:
\[ p_i = k(\rho_i - \rho_0) \] (2.9)

Where \( k \) is a pressure constant governing relative density fluctuation, \( \rho_i \) is the density at particle \( i \) and \( \rho_0 \) is the desired rest density of the fluid.

An alternate equation of state was proposed by (Becker, Teschner 2007) that enforces low density variations, resulting in less compressible fluids. The Tait equation of state is as follows:

\[ p_i = \frac{k \rho_0}{\gamma} \left( \left( \frac{\rho_i}{\rho_0} \right)^\gamma - 1 \right) \] (2.10)

Where \( \gamma \) is some constant power term.

Though the Tait equation encourages desirably low density fluctuations, it requires prohibitively small timesteps in order to remain stable. We investigate this further in our discussion and results sections.

2.2.4 Viscosity

Viscosity is an important property of fluids we wish to model, allowing substances of varying thickness to be handled as a result of internal friction. As in our pressure equation, we employ the symmetrised viscosity force of (Solenthaler 2008):

\[ \mathbf{f}_{i}^{\text{viscosity}} = \frac{m_i}{\rho_i} \sum_j \left( \frac{\mu_i + \mu_j}{2} \right) m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(r_{ij}, h) \] (2.11)

Where \( \mu_i \) and \( \mu_j \) are viscosity values of particles \( p_i \) and \( p_j \) respectively, and \( \mathbf{v}_i \) and \( \mathbf{v}_j \) are particle velocities.

2.2.5 XSPH

In addition to the viscosity forces of section 2.2.4, we implement XSPH viscosity to improve simulation stability and improve coherence of motion (Monaghan 1992).
We apply the XSPH equation of (Schechter 2012) directly to the velocity of a particle $i$ as:

$$v_{i}^{\text{new}} = v_i + c \sum_j (v_j - v_i) W(r_{ij}, h)$$  \hspace{1cm} (2.12)

Where $c$ is a viscous smoothing constant, typically in the range of 0.01. This adjustment moves an individual particle’s velocity closer to the average of its neighbours’ velocities, weighted by the standard SPH kernel $W$. This has the effect of improving simulation stability by smoothing the discretised velocity field, while still conserving linear and angular momentum.

Though it appears to require an additional neighbourhood search to gather surrounding velocities and weights, this can be done per-particle in any of the previous stages such as density summation or pressure force calculation. We use an XSPH accumulation vector calculated per-particle in the pressure phase, and only apply this in the time integration step of our simulation.

### 2.3 Discussion

It is relatively straightforward to implement an SPH simulation capable of producing convincing fluid behaviour. However, certain factors are difficult to mitigate — namely density fluctuation errors. It is difficult to ensure a low level of compressibility, without significantly increasing the computational complexity of the SPH model.

Weakly Compressible SPH (Becker 2007) significantly aids in reducing density fluctuation simply through changing the equation of state. However, this comes at the cost of a significantly lower timestep being required to avoid unstable simulations. We include our WCSPH simulations in the supplemental material, but deem it unsuitable for our real-time purposes due to its restrictions on timestep. Both Predictive-Corrective Incompressible SPH (Solenthaler, Pajarola 2009) and Implicit Incompressible SPH (Ilmsen 2014) appear to be less restrictive than the results of WCSPH, and are an interesting area for further research.

The given viscosity force is well-suited for low viscosity fluids, but it is unstable
for larger viscosity values. Several approaches have attempted to address this. For instance, evolving a per-particle viscous stress tensor through time (Müller, 2004), or using alternate implicit integration schemes (Peer 2015). These approaches can greatly increase the complexity of the fluid model, and in the case of (Müller 2004), are significantly less performant.

Motivated by the established shortcomings of SPH and our desire for stable real-time interactivity, we now turn to the work of (Macklin 2013) and others on Position Based Fluids. We favour this approach due to its unconditionally stable time integration and demonstrable ability to handle real-time scenarios at scale. PBF improves upon standard SPH in several respects, which we expand upon in Chapter 3.
Chapter 3

Position-Based Fluids

3.1 Introduction

Building on our findings in Chapter 2, we will now introduce our Position Based Fluids simulator. It employs an SPH-style approach to gathering particle data, and reformulates fluid incompressibility as a constraint satisfaction problem. This is then solved using the Position Based Dynamics iterative method.

We will give an introductory overview of Position Based Dynamics, and go on to introduce the methodology of the adapted PBF fluid simulator. We will then introduce our simulation method, adaptations and improvements to existing PBF research. Finally, we will discuss the success of our approach, and give an overview of areas for future research.

3.2 Position Based Dynamics

Position Based Dynamics (Müller 2006) is a simulation approach for dynamic bodies. Rather than taking a force- or velocity impulse-based approach to the resolution of collision contacts and constraints, PBD operates directly on a position level. Position based simulators are particularly favourable for their stability, as their projective integration scheme avoids ”overshooting” problems typical of force-based explicit integrators (Bender 2017). Moreover, collisions and con-
constraints can be easily resolved by projecting point positions to valid configurations directly.

### 3.2.1 XPBD

Standard PBD follows a geometrically motivated approach to solving position-level constraints. As such it is not physically motivated, and regardless of the choice of stiffness constants, constraints will converge to infinite stiffness as PBD iterations are increased (Macklin 2016). Additionally, there is a non-linear connection between iteration count and constraint stiffness, which can make models hard to parametrise.

A solution to this was presented in (Macklin 2016), extended position based dynamics (XPBD). A reformulation of PBD-style constraints is given that solves the timestep-dependent stiffness problem. Moreover, the addition of a compliance constant per-constraint is provided that is physically motivated — constraint compliance has a direct correspondence to the Young’s modulus of the simulated material (Macklin 2016).

This is an attractive adaptation for our purposes, as our viscoelasticity model is dependent on softened distance constraints to connect particles. We continue to use standard PBD for our fluid density constraints, and use XPBD compliant constraints for our visco-elasticity model. This is detailed in section 3.7.3.

### 3.2.2 Second-Order Integration

Standard PBD as proposed by (Müller 2007) employs a first-order backward Euler (BDF1) update of both positions and velocities:

\[
x^{n+1} = x^n + \Delta t v^n \Delta t^2 M^{-1} f
\]

\[
v^{n+1} = \frac{1}{\Delta t} \left( x^{n+1} - x^n \right)
\]  

(3.1)

Where \( n \) is the simulation time value, \( x \) and \( v \) are position and velocity respectively, \( f \) is external forces, and \( M \) is the mass of the simulated body.

A second order accurate backward Euler update is detailed in its application
to PBD in (Bender 2017) as follows:

\[
x^{n+1} = \frac{4}{3}x^n - \frac{1}{3}x^{n-1} + \frac{8}{9}\Delta tv^n - \frac{2}{9}\Delta tv^{n-1} + \frac{4}{9}\Delta t^2M^{-1}f
\]

\[
v^{n+1} = \frac{1}{\Delta t}\left[\frac{3}{2}x^{n+1} - 2x^n + \frac{1}{2}x^{n-1}\right]
\]

We use this second-order update scheme in all of our simulations as it exhibits lower numerical damping than standard BDF1 updates, and requires fewer constraint iterations for convergence (Macklin 2017).

3.3 Position Based Fluids

3.3.1 Introduction

Position Based Fluids (Macklin 2013) is an extension to Position Based Dynamics, providing support for the simulation of fluid materials. PBF takes a similar particle-based simulation approach to SPH, reformulating the standard pressure and viscosity forces in terms of inter-particle positional constraints that maintain constant fluid density.

Due to the stability of the geometric constraint formulations of the PBD integrator, PBF is capable of being simulated efficiently with large timesteps. This makes it ideal for real-time simulation.

As the particle-based model of PBF is a mere reformulation of SPH into a position-based framework, many of our previous insights in Chapter 2 still hold. We see PBF as the logical next step to real-time particle based fluid simulation, as it relies on the same interpolated field discretisation of SPH, while further solving several of our previous issues. Namely, the timestep restrictions of SPH resulted in a trade-off between fluid compressibility and real-time stability.

In PBF, this is not an issue, due to its unconditionally stable integration scheme and constraint-based method of enforcing constant density. Furthermore, PBF typically requires relatively few constraint iterations to achieve acceptable density error in line with the results of more computationally intensive density correction schemes such as PCISPH (Macklin 2013).
3.3.2 Simulation Method

As in (Macklin 2013), we use a constant density constraint $C_i$ on a particle $p_i$ and its neighbours $p_1, \ldots, p_n$ based on an initial fluid rest density $\rho_0$.

$$C_i(p_1, \ldots, p_n) = \frac{\rho_i}{\rho_0} - 1$$ (3.2)

The density of a particle $\rho_i$ is calculated in an identical manner to our SPH implementation.

$$\rho_i = \sum_j m_j W(r_{ij}, h)$$ (3.3)

Following (Macklin 2013), we use ”constraint force mixing” (Smith 2006) regularized constraint projection, to enforce constant fluid density. First, we calculate the gradient of the constraint function with respect to particle $i$ as follows:

$$\nabla p_k C_i = \frac{1}{\rho_0} \begin{cases} \sum_j \nabla p_k W(r_{ij}, h) & \text{if } k = i \\ -\nabla p_k W(r_{ij}, h) & \text{if } k = j \end{cases}$$ (3.4)

This allows us then to calculate the constraint’s Lagrange multiplier $\lambda_i$ for particle $i$:

$$\lambda_i = \frac{-C_i(p_1, \ldots, p_n)}{\sum_k |\nabla p_k C_i|^2}$$ (3.5)

Where $\epsilon$ is a constant user-defined constraint relaxation parameter. In our simulations we use a value of 0.1 to avoid the instability noted by (Macklin 2013), when particles are nearing separation. Note that the larger the value of $\epsilon$, the more the constraint force is softened through the solver iterations, resulting in more compressible fluids.

Finally, we arrive at the positional update for particle $i$ in a single timestep of PBF:
$$\Delta \mathbf{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(r_{ij}, h)$$

(3.6)

We refer the reader to (Macklin 2013) for a detailed derivation of the given constraints. We employ the same kernel functions as in our SPH simulation detailed in Chapter 2.

### 3.4 Viscosity

The original implementation of Position Based Fluids by (Macklin, 2013) relies on XSPH (section 2.2.5) to improve simulation stability and visual coherence, adding support for a range of viscous fluids. XSPH is however not stable when modeling highly viscous fluids such as honey.

(Alduán 2016) note that a choice of XSPH damping coefficient above 0.5 typically results in erroneous gains in momentum and drastically reduced stability. Our findings are consistent with this, and we elaborate on this in chapter 5.

The viscosity approach detailed in section 2.2.4 is also susceptible to stability issues as it relies on explicit forces. When viscosity values are sufficiently high this instability is increased.

In order to support high viscosity fluids, (Alduán 2016) propose a simple iterative modification to XSPH. They divide each simulation step’s XSPH viscosity calculation into $N$ stable sub-steps. They first establish the maximum stable damping value. An XSPH pass over a particle $p_i$ is then performed a total of $N$ times in a single timestep.

The iterative XSPH velocity update then takes the form

$$\mathbf{v}_{i\text{new}} = \mathbf{v}_i + \frac{1}{N} \sum_j \min(c_i, c_j) \frac{m_j}{\rho_j} (\mathbf{v}_j - \mathbf{v}_i) W(r_{ij}, h)$$

(3.7)

Where $c_i$, $c_j$ are per-particle viscosity constants. Inter-particle velocity differences are thus updated and recalculated in each sub-step, and no single XSPH viscosity update compromises the stability of the simulation. Multiple phases are also supported.
This iterative approach has the advantage of being very straightforward to implement, and allows a diverse range of viscosity values to be handled by our simulator without the need to decrease the simulation time step (Alduán 2016).

3.5 Vorticity

Vorticity confinement (Fedkiw 2001) is a method originally introduced to remedy numerical dissipation effects in smoke simulations. It is now widely used in fluid simulations as a method of re-injecting motion into damped systems.

As highly dynamic and turbulent motion is typically expected in fluids, particularly so in low viscosity and inviscid materials, this is a desirable effect for us to support.

Vorticity confinement applies an additional force to the velocity field of a fluid based on its existing rotational energy at a given point. This is discretised into a particle-based formulation by first calculating the vorticity at individual particle locations:

$$\omega_i = \nabla \times v = \sum_j (v_i - v_j) \times \nabla p_j W(r_{ij}, h)$$  \hspace{1cm} (3.8)

A corrective force is then applied along particle the location vector $N$ of particle $p_i$:

$$N = \frac{\eta}{||\eta||}$$  \hspace{1cm} (3.9)

Where $\eta = \nabla |\omega_i|$. The final vorticity confinement force is as follows:

$$f_i^{vorticity} = \alpha (N \times \omega_i)$$  \hspace{1cm} (3.10)

Where $\alpha$ is a per-fluid constant scaling the amount of vorticity confinement to apply. We leave $\omega_i$ unnormalised, so as not to add spurious vorticity where it does not already exist. (Fedkiw 2001, Macklin 2013).

Vorticity confinement is entirely optional to our simulator, and may be enabled
depending on the desired fluid motion. We note that there is some computational overhead associated with the calculation of both particle vorticity $\omega$ and location vector $\mathbf{N}$. As such, we prefer to disable vorticity confinement in our simulations for computational efficiency whenever possible.

### 3.6 Surface tension

#### 3.6.1 Approach

Surface tension is a property of fluids that manifests as a tendency towards minimising fluid surface area. We regard this as an important element for coherent and visually pleasing simulations.

Several approaches exist to model surface tension in particle based fluid representations. We consider the categories given by (Akinci 2013) — curvature based methods, pairwise forces, and modified fluid model formulations.

Curvature based approaches e.g. (Müller 2003) attempt to calculate curvature values at the fluid surface or interface, applying forces accordingly.

Pairwise forces e.g. (Becker 2007) take a microscopic-level approach to modelling surface tension, treating particles as individual molecules subject to cohesive forces. This pairwise force approach has the advantage of being very computationally efficient and with guaranteed symmetry. However, such an approach does not actually attempt to explicitly minimise surface area, and is liable to settling in erroneously connected configurations. This can produce unexpected results, particularly with high surface tension values (Akinci 2013).

A modified fluid model approach is taken by both (Clavet 2005) and (Macklin 2013). Both approaches augment the standard fluid solver with an additional artificial pressure term. This pressure term reduces undesirable particle “clustering” in cases where particles have few neighbours, and the desired rest density cannot be reached (Macklin 2013).

As a consequence of the balance between both standard and artificial pressure forces, surface tension effects can be observed. However, this artificial pressure term is intrinsically linked to both spatial resolution and timestep. Additionally, there is no separability between the anti-clustering effects of the artificial pressure
term, and the surface tension of the fluid itself.

We adopt the approach of (Acinki 2013) in our implementation of surface tension. This is a curvature based approach. In spite of the associated computational overhead, the model’s support for a wide range of effects makes it attractive for our purposes. These effects include extremely high surface tension fluids, and wetting and adhesion effects.

### 3.6.2 Implementation

We begin by following the equations and values given by (Akinci 2013). We begin by calculating an approximation of the normal at the position of a particle \( p_i \) as follows:

\[
\mathbf{n}_i = h \sum_j \frac{m_j}{\rho_j} \nabla W(\mathbf{r}_{ij}, h)
\]  

(3.11)

This uses the standard SPH interpolated approximation as in equation 2.2. The normal \( \mathbf{n}_i \) is then used to calculate a curvature force:

\[
\mathbf{F}_{i \text{curvature}} = -\gamma m_i (\mathbf{n}_i - \mathbf{n}_j)
\]  

(3.12)

Where \( \gamma \) is a constant value governing fluid surface tension. We then calculate the cohesion force:

\[
\mathbf{F}_{i \text{cohesion}} = -\gamma m_i m_j C(|\mathbf{r}_{ij}|) \frac{\mathbf{r}_{ij}}{|\mathbf{r}_{ij}|}
\]  

(3.13)

We use the same spline function as (Acinki 2013), which takes the following form:

\[
C(r) = \begin{cases} 
(h - r)^3 r^3, & \text{if } 2r > h \text{ and } r \leq h \\
2(h - r)^3 r^3, & \text{if } r > 0 \text{ and } 2r \leq h \\
0, & \text{otherwise}
\end{cases}
\]  

(3.14)
Finally we calculate the combined surface tension force $F_{st}$

$$F_{st} = K_{ij} \left( F^{\text{cohesion}}_i + F^{\text{curvature}}_i \right)$$

(3.15)

Where $K_{ij} = \frac{2\rho_0}{\rho_i + \rho_j}$ is a normalisation term, preventing particles with insufficiently small neighbour lists from being isolated.

The combined cohesion and surface minimisation terms of the model of (Akinci 2013) allow us to simulate a vast range of different surface tension effects and interactions. We note that while we find this approach to be very stable, it still relies on explicit forces and as such can require a lower timestep to ensure robust behaviour. A modification to this surface tension model that avoids explicit forces would be an interesting problem for future research.

We found that higher surface tension values did require a lower timestep to avoid ”explosive” behaviour. Results of our tests with varying surface tension can be seen in chapter 5 and our supplemental material. Additionally, we will extend this surface tension model for improved handling of multiple phases in section 3.8.2.

### 3.7 Viscoelastic Fluids

#### 3.7.1 Background

Viscoelastic and plastic flow can be seen in a wide range of deformable substances, such as mucus and synthetic polymers. Modeling these behaviours has been approached many ways in simulation, from mass-spring systems to more complex constitutive models.

(Goktekin 2004) tackled viscoelastic materials in an Eulerian formulation by including elastic strain terms in the Navier-Stokes equations and solving these on a grid. Other physically motivated techniques exist in Lagrangian approaches. (Solenthaler 2007) introduced an SPH-based model of viscoelastic behaviour, including support for material phase transitions. (Becker 2009) extended this to correctly handle rotating bodies. Both of these approaches require evolving a
per-particle strain tensor.

We choose to focus on a simple implementation of viscoelasticity based on an adaptive mass-spring system. Our approach is similar to that of (Clavet 2005) and (Takahashi 2014). Though it is less directly physically motivated than constitutive model approaches previously detailed, it is simple to implement and easily integrable into our existing solver.

3.7.2 Previous Work

We will briefly outline the viscoelasticity model of (Clavet 2005) here.

Elasticity is modeled by the creation of springs at runtime between neighbouring particles. Viscoelasticity is parametrised by spring stiffness $k$, plasticity $\alpha$, deformation yield factor $\gamma$ (influencing plastic deformation), and spring rest length $L$. The displacement force exerted upon particles is as follows:

$$\Delta p_i = \frac{1}{2} \Delta t^2 \left( 1 - \frac{L_{ij}}{h} \right) \left( L_{ij} - |r_{ij}| \right) \hat{r}_{ij} \quad \text{(3.16)}$$

Note that this displacement is applied on a positional level as in PBD, however the integration scheme used by Clavet et al is only conditionally stable, and depends on the simulation timestep.

Plasticity is supported by dynamically modifying spring rest lengths over time, in response to deformative forces. Clavet et al detail a simple linear plasticity model. The rate of change of a spring $L$’s rest length is given as:

$$\Delta L_{ij} = \Delta t \alpha (|r_{ij}| - L_{ij}) \quad \text{(3.17)}$$

Where $\alpha$ represents the plasticity of the material, 0 being entirely elastic and 1 being fully plastic.

Plastic flow is further governed by the von Mises plasticity condition, which states that plastic flow only occurs if the deformation from a rest state is sufficiently large (Fung 1965), (Clavet 2005).
The final rest length modification term is then:

$$\Delta L_{ij} = \Delta t \alpha \text{sign}(\|\mathbf{r}_{ij} - L_{ij}\| - L_{ij}) \max(0, \|\mathbf{r}_{ij} - L_{ij}\| - \gamma L_{ij})$$

(3.18)

Note that this is identical to equation 3.17 when $\gamma$ is zero. We now turn to the adaptation and implementation of this visco-elasticity model in our position-based fluid solver.

### 3.7.3 Adaptation to PBF

The above model allows for a large range of viscous, elastic and plastic behaviour. However, the approach detailed in (Clavet 2005) is built on a predictive-corrective numerical integration scheme that is only conditionally stable. We include the results of a naive implementation of (Clavet 2005) in our supplemental material. Some adjustment is then required to adapt this model to PBF, in order to gain the benefits of a PBD-style unconditionally stable integrator. A mass-spring approach to viscoelasticity with PBD has been detailed by (Takahashi 2014). However, their model operates on a velocity level, rather than a positional one, and as such lacks the advantage of our XPBD-based solver in using physically derived stiffness values.

We note that the spring forces proposed by Clavet et al are simple softened distance constraints of length $L_{ij}$ for particles $p_i$ and $p_j$. Motivated by this, we modify this spring update based on the XPBD formulation of compliant constraints, as mentioned in section 3.2.1.

We store a per-spring Lagrange multiplier variable $\lambda_i$, which is updated in each XPBD iteration. Our springs are then simply XPBD-style distance constraints, with a stiffness coefficient $k_{\text{elastic}}$ corresponding to constraint compliance.

A compliance value of 0 thus represents an infinitely stiff spring, which our system will converge to as our iteration count is increased. Our stiffness coefficient maps to the inverse of the Young’s modulus of the material (Macklin 2016), and is thus physically motivated.

We now have an implementation of visco-elasticity that is timestep-independent, stable, physically motivated and easily authored. The model is parametrised by
an elastic stiffness value, plasticity constant, plastic flow yield ("creep" threshold), and rest spring length. All parameters are modifiable at run-time, allowing us to model a much wider range of fluid behaviour.

With this in mind, we now turn to the interaction between differing fluid materials.

### 3.8 Multiple Phase Fluid Simulation

In our aim to simulate a diverse range of materials, we wish to handle the interaction of different fluids with one another. So far, we have been considering only a single fluid phase.

As the properties of our fluids can be parametrised in many ways, it is important to us to stably simulate interactions between fluids with significantly different properties such as mass-density, viscosity, surface tension and visco-elasticity.

#### 3.8.1 Implementation

We follow the technique of (Alduán 2015) in our approach to multi-phase simulation with PBD. As our simulations use constant resolution, our particle density computation now becomes:

\[
\rho_i = m_i \sum_j W(r_{ij}, h) \tag{3.19}
\]

The positional correct for multi-phase PBF fluids is given by (Alduán 2015) as the following:

\[
\Delta p_i = \frac{1}{m_i} \sum_j \left( \frac{m_i V_j}{\rho_{0,i} V_i} \lambda_i + \frac{m_j V_i}{\rho_{0,j} V_j} \lambda_j \right) \nabla W(r_{ij}, h) \tag{3.20}
\]

Where \( V_i, V_j \) are particle volumes, which we calculate as \( \frac{m_i}{\rho_i} \). They calculate the Lagrange multiplier of the incompressibility constraint \( \lambda_i \) based on a pre-calculated regularisation constant for each fluid phase.

In contrast to this, we skip this regularisation computation entirely and instead...
regularise by the minimum rest density of the two particles being considered:

\[ \lambda_i = \frac{-C_i \min(\rho_{0,i}, \rho_{0,j})}{\sum_k |\nabla_{p_k} C_i|^2 + \epsilon} \] (3.21)

Though our simplified regularisation may be overly conservative in its handling of density constraints at the interface, we find this to be visually acceptable and stable in an interactive setting. We are able to simulate density ratios up to a ratio of 1 : 100 with time steps large enough for real-time simulation.

### 3.8.2 Surface Tension and Miscibility in Multiphase Simulations

In the real world, we can observe a range of complex phenomena arising as a result of surface tension at the interface between fluids — puddles and pools forming when oil and vinegar are mixed, for example. As it stands, our simulator is capable of supporting a wide range of surface tension values in single fluid phases, but we have yet to address the manner in which fluids with different surface tension values interact with each other.

In the work of (Akinci 2013), surface tension terms take particle mass-density into consideration. However, as there is no distinction given between the surface tension values of individual phases, they assume fluids of different phases to be fully miscible, and with identical surface tension values.

Following from the approach proposed by (Akinci 2013), we propose an alternative formulation of surface tension forces that allows for multiple fluid phases to interact with differing surface tension values. We further add support for the concept of miscibility \( \psi \), defined for each phase in the simulation.

We begin by modifying the symmetrisation value \( K_{ij} \) per particle given in equation 3.15. Accounting for differences between phase rest densities \( \rho_{0,i} \) and \( \rho_{0,j} \), this becomes:

\[ K_{ij} = \frac{\rho_{0,i} + \rho_{0,j}}{\rho_i + \rho_j} \] (3.22)
We then determine our surface tension value $\gamma_{ij}$ between 2 particles as

$$\gamma_{ij} = \frac{1}{2} \psi (\gamma_i m_i + \gamma_j m_j)$$  \hspace{1cm} (3.23)

This calculation considers the mass contributions of each particle on the surface tension force. Here $\psi_{ij}$ is the miscibility constant between 2 fluid phases, in the range 0 – 1. A value of 0 results in no surface tension forces being applied at the interface between particles of different phases (fully immiscible), and a value of 1 results in tension forces acting pair-wise between phases (fully miscible).

Finally, we modify the cohesion and curvature forces as follows:

$$F_i^{\text{cohesion}} = -\gamma_{ij} C(|r_{ij}|) \hat{r}_{ij}$$  \hspace{1cm} (3.24)

$$F_i^{\text{curvature}} = -\gamma_{ij} (n_i - n_j)$$  \hspace{1cm} (3.25)

These modified forces do not include the additional mass influence terms of (Akinci 2013), which have been accounted for in our calculation of the surface tension value in equation 3.23.

Our modified surface tension model results in fully symmetric pairwise forces. It retains the desirable area minimisation properties of (Akinci 2013), while adding support for varying miscibility between fluid phases.

### 3.9 Discussion

We have arrived at a full implementation of a position-based fluids simulator capable of robustly simulating a large range of fluid behaviour.

We began with a second-order accurate position-based dynamics system capable of solving constant density constraints, implementing the Position-Based Fluids model of (Macklin 2013).

We then implemented the iterative multi-phase XSPH extension of (Alduán 2015), allowing a much larger range of viscosity values to be supported.
We implemented vorticity confinement as described in (Macklin 2013), adding support for highly turbulent fluid motion.

The versatile surface tension model of (Akinci 2013) was added in place of standard PBF surface tension forces, extending our simulator’s range of supported fluid behaviour to fluids with high surface tension, exhibiting physically motivated surface area minimisation. Additionally, we extended the model of (Akinci 2013) to fully support fluid interactions between multiple phases with varying surface tension values. As a result, we are able to simulate a richer range of fluid mixing behaviours as demonstrated in the supplemental material.

We implemented a mass-spring based model of visco-elasto-plastic behaviour, allowing us to simulate fluids capable of elasto-plastic deformation and flow following the von Mises yield criterion (Fung 1965). We implemented and modified the mass-spring model of (Clavet 2005) to the strengths of our position based simulator. Namely, we reformulated the spring constraint implementation of (Clavet 2005) to instead be an XPBD-style compliant distance constraint. This allows our simulator to use physically motivated values for its spring constants, derived from the material Young’s modulus. Additionally, this reformulation enhanced both the stability and ease of authoring of our simulation. This is due to timestep-independent compliant constraint model offered by XPBD (Macklin 2016).

In total, our simulator is capable of supporting an extremely diverse range of materials, parametrised by rest density, mass, viscosity, surface tension, and visco-elasto-plastic properties. However, there remain several areas in which our simulator could be extended.

Currently, we employ only a simple linear plasticity model. In order to exhibit non-linear plasticity, a different plastic yield criterion must be employed.

We do not consider any form of visco-elastic forces between separate fluid phases. This would be a simple extension to our existing model — an example modification would be to expose an additional parameter controlling elastic spring tension between different phases.

Our iterative viscosity model incurs a performance penalty as fluid viscosity is increased. As our viscosity model is quite simple, we found this to be satisfactory for many real-time applications, however for highly viscous substances another viscosity model may be preferred. We refer the reader to the work of (Peer 2015)
and (Barreiro 2017) as promising approaches to this problem. As our visco-elasticity model relies on the continuous creation and deletion of many spring constraints that must be temporally consistent between frames, it can be quite computationally intensive. As such, we were only able to successfully implement this in our CPU implementation, as the GPU implementation suffered from very scattered memory accesses and impacted performance when managing per-particle spring lists. The work of (Barreiro 2017) appears to be an attractive alternate approach to visco-elastic simulation, and is an interesting direction for future research.

Given our fluid model as described in this chapter, we now turn to its implementation in a real-time interactive context.
Chapter 4

Implementation and Optimisation

4.1 Overview

Fluid simulation is typically a very computationally intensive task. Depending on the desired simulation detail, Lagrangian approaches to fluid simulation can require the concurrent simulation of thousands to hundreds of thousands of individual particles. As such, considerable work was put into optimising the simulator through the use of techniques detailed in this chapter.

As the computation requirements of Lagrangian simulation are highly regular and symmetric across particles, it is an ideal candidate for a GPU acceleration. Much success has been reported in particle-based simulations on the GPU, e.g. (Krog 2010) (Macklin 2013) (Bender 2017). We detail our approach to both our CPU- and GPU-based implementations in this chapter. We then introduce a novel simulation-level extension to position based fluids, reducing computation time on the CPU while maintaining stability and visual quality comparable to that of the baseline implementation.
4.2 CPU Implementation

We implemented our 2D position-based fluid solver on the CPU in a simple environment to allow fast prototyping. Our implementation is built in the Processing environment, in a simplified syntax based on Java. Our implementation is written entirely from scratch. Other than using Processing’s native graphics operations to draw fluid particles as rectangle primitives, all code is our own.

In order to accelerate the neighbourhood search of our particles, we implemented a basic spatial hashing scheme based on a uniform grid. In each time-step, particles calculate a 1-dimensional flattened grid cell index, and add themselves to the corresponding cell’s list of particles. We then search the cells immediately surrounding a particle’s location, adding all particles within the interaction radius \( h \) to the particle’s list of neighbours. (Figure 4.1).

Figure 4.1: A 2D uniform grid. A particle considers all particles in a radius \( h \) to be its neighbours (blue), by inspecting its neighbouring cells (green).
the interpolation and determination of various fluid properties.

In order to maintain a stable simulation, we typically choose a timestep value $\Delta t$ below 1 in all of our simulations. We perform $\left\lceil \frac{1}{\Delta t} \right\rceil$ timesteps every frame to ensure consistent simulation playback across different timestep values. Our performance goal is 60 frames per second, standard for interactive applications. This is equivalent to a time budget of $\frac{1000}{60}$, or 16.6 milliseconds per simulation frame, including all visualisation and event handling.

Our CPU implementation is light-weight and straightforward to implement, but it is not sufficient for modeling complex scenes with more than a few thousand particles. We now turn to our GPU-accelerated solver, capable of much higher simulation detail.

### 4.3 GPU Implementation

Our GPU based implementation was written using C# and DX11 HLSL-style *Compute shaders* within the *Unity* game engine. *Unity* was chosen to allow us an abstraction from low-level implementation details, such as input handling and camera setup.

All simulation-relevant code was written from scratch. We added support for basic first person camera controls in order to navigate scenes in 3D. Additionally, we implemented a fast mesh-based visualisation method using GPU instancing. This is detailed in section 4.6.

We initialise the simulation state on the CPU at the start of our program, and launch shader kernels every frame on the GPU to perform simulation.

In order to best leverage the parallel architecture provided by the GPU, our parallelised simulation is significantly different to that of our CPU implementation.

In particular, we do not create and manage per-particle neighbourhood lists. Instead, we perform a broadphase evaluation at every stage of the simulation requiring neighbourhood information. This is significantly faster than managing the per-particle memory costs of tracking individual lists.

For our broadphase neighbourhood search, we employ the *Counting Sort* approach of (Hoetzlein 2014). We chose this for its demonstrated success in large
scale real-time simulations. Our implementation of this in Compute shaders is detailed in section 4.4.

Our approach to simulation time-step is identical to that of our CPU implementation (section 4.2).

Before tackling the implementation of our neighbour search, we will begin by examining the more general problem of PBD simulation in parallel.

### 4.3.1 Position Based Dynamics on the GPU

As in standard Position Based Dynamics, we handle the density constraints of our CPU simulation in a Gauss-Seidel manner. That is, we iterate over particles and modify their positions in-place from within the per-particle constraint satisfaction loop. This has the advantage of ensuring fast convergence. Pseudo-code for this is shown below in Algorithm 4.3.1.

**Algorithm 1** Gauss-Seidel Constraint Update

1: procedure SolveConstraints
2: for each particle \( p_i \) in sequence do
3: for each constraint on \( p_i \) in sequence do
4: calculate \( \Delta p_i \) from Lagrange multiplier \( \lambda \)
5: update position in-place as \( p_i = p_i + \Delta p_i \)

However, this is an inherently serial algorithm, ill-suited to the GPU. If we modify the position of a particle \( p_i \) in our constraint resolution kernel, this may be simultaneously accessed in parallel by some other particle \( p_j \). This is a race condition and leads to non-deterministic unexpected behaviour.

On the GPU, we instead solve our density constraints in a Jacobi manner. We store a positional update accumulation value \( t_i \) at each particle, and update this in a subsequent Compute kernel. A sketch of this simple modification is shown in algorithm 4.3.1.

Though Jacobi methods typically exhibit slower convergence to that of Gauss-Seidel (Bender 2017), we find that in practice this is not a problem for our simulator. As position based fluids simulations typically only require a few iterations to reach desirable levels of incompressibility, the computational impact is low.

More sophisticated approaches to parallelised PBD exist, based on principles of graph colouring. We refer the reader to (Fratarcangeli 2016), (Bender 2017)
Algorithm 2 Jacobi Constraint Update

1: procedure SolveConstraintsJacobi
2:     for each particle $p_i$ in parallel do
3:         $t_i = 0$
4:         for each constraint on $p_i$ in sequence do
5:             calculate $\Delta p_i$ from Lagrange multiplier $\lambda$
6:             set $t_i = t_i + \Delta t_i$
7:     end for
8:     for each particle $p_i$ in parallel do
9:         $p_i = p_i + t_i$

for a description of such approaches.
We deem these as beyond the scope of this dissertation, as our simple Jacobi
modification exhibits satisfactory convergence with few iterations.

4.4 GPU Neighbourhood Search

4.4.1 Overview

As both our SPH and PBF implementations consider the surrounding neighbour-
hoods of each particle within a constant interaction radius, it is an ideal candidate
for broad-phase fixed-radius nearest neighbour search algorithms. Rather than
the standard uniform grid approach detailed in section 4.2, we employ a vari-
tion on the Counting Sort algorithm (Cormen 2001) adapted specifically to GPU
based particle simulations. We choose this for its demonstrated capacity to sim-
ulate large bodies of particles in real-time in the Fluids v.3 system (Hoetzlein,
2012-13).
An overview of the algorithm is detailed in (Hoetzlein 2014), and we include a
brief sketch of our implementation as follows.

4.4.2 Implementation

We divide our simulation space into a regular grid of bin structures, implemented
as a StructuredBuffer within a Compute shader. Our Bin struct takes the follow-
ing form in implementation:

```
struct Bin {
```
We also include a simplified rendition of our Particle struct, in order to show
the data as it is stored on a per-particle level.

```c
struct Particle {
   float3 pos;
   float3 vel;
   float3 force;
   float density;
   uint index_in_bin_array;
};
```

A step-by-step overview of our broad phase implementation is as follows:

- **Reset state at beginning of time-step**
  - Set all bins’ `value` to 0, indicating no particles stored

- **Store particles in bins**
  - Put particle positions through a simple hash function, mapping 3D position to 1D flattened bin buffer index
  - Store the current bin’s value into our particle $p_i$ for later use, as $p_i.index.in.bin.array$
  - Atomic-add 1 to `value` of our particle’s associated bin, indicating the particle has been stored

- **Re-index particles into a new, spatially sorted buffer**
  - Set each bin’s `num_held` value to be equal to its `value` for later use
  - Perform a prefix sum over bin values, storing the result of our prefix sum in each bin’s `value` variable
  - For each particle, determine its corresponding bin by re-hashing its position
  - Determine each particle’s sorted index as the sum of its bin’s prefix-summed original `value` plus the particle’s stored $idx.in.bin$
The result of this procedure is an array of particles sorted by the value of their corresponding hashed grid location. With this, we perform a standard uniform grid-style neighbourhood search of each particle’s surrounding cells. The advantage is that particles nearby each other are closely aligned in memory, resulting in coherent GPU memory accesses and improved performance relative to naive uniform grids (Hoetzlein 2014).

Note that we do not explicitly implement any form of dictionary or array structure for our grid of bins / cells. Instead, we merely store one auxiliary buffer for particle sorting, two integers per-bin and an additional integer value at each particle. As such, the memory requirements of this approach are low.

One area of significance for our GPU Counting Sort is its prefix sum step. Prefix sums are trivially implemented on the CPU, but their sequential nature does not seem immediately parallel-friendly. As our interaction radius is typically small relative to the size of our simulation domain, we end up with several thousand bins to sum over.

For example, with an interaction radius $h = 8$ and a simulation domain of size $1024^3$, we end up with $\left(\frac{1024}{8}\right)^3 = 2097152$ bins. This sequential sum quickly becomes the bottleneck of our broadphase algorithm.

To address this, we implement the parallel inclusive scan algorithm of (Nguyen 2007). This allows us to perform a parallel multi-level block scan over all bins, reducing the computation time of our prefix sum significantly.

We are left with a fully parallelised Counting Sort-based neighbourhood search algorithm that greatly enhances the performance of our simulations. Results of our GPU implementation are further detailed in chapter 5.

Equipped with an efficient method to determine particle interactions, we now turn to simulation model-level optimisation. We will introduce a new method which operates on a per-constraint basis in order to better our system’s performance.
4.5 Slackening: A Novel Adaptive Constraint Method

4.5.1 Overview

Both in standard PBD and PBF, constraints are resolved iteratively, correcting positions over the course of some iteration count \( k \). This is typically chosen to be a fixed value, high enough such as to exhibit acceptable convergence in the given timestep. We observe that in systems with large amounts of independent constraints, this can lead to redundant iterations over constraints that have already converged to an acceptable level.

In this section, we introduce our novel adaptive constraint approach for position based fluids, motivated by this observation.

4.5.2 Implementation

Position based fluids aims to enforce incompressibility through the use of per-particle constant density constraints. This constraint operates pair-wise between a given particle and its neighbouring particles within a radius \( h \). In a standard PBD solver, all constraints are iterated over a fixed \( k \) times per timestep.

We note that particles requiring the most attention are those in areas of high pressure. Particles e.g. moving at a high velocity counter to the flow of the surrounding fluid, interacting with boundaries, or responding to strong external forces, are most likely to violate the density constraint criterion and require several constraint iterations to be satisfied.

However, particles that exist already in a stable configuration typically require fewer iterations in order to satisfy constant density. From this, we propose an early-out criterion that skips per-particle constraint updates based on a particle’s relative density error. This is detailed in algorithm 3.

Where \( p_i.\text{iterations} \) is the number of constraint iterations performed this timestep for particle \( p_i \), and \( \zeta \) is the permitted density error for our simulation. We note that this criterion can be evaluated very efficiently, and is easily re-parametrised simply by choosing different values of \( \zeta \).

In our simulation loop, both when calculating the Lagrange multiplier \( \lambda_i \) and
Algorithm 3 Slackening Early-Out Criterion

1: procedure RequiresFurtherIteration(p_i)
2:   if p_i.iterations == 0 then
3:     return true  \(\triangleright\) Particle has not yet received constraint iterations
4:
5:   tolerance = \(\zeta \rho_{0,i}\)
6:   if \(|\rho_i - \rho_{0,i}|/\rho_{0,i}\) > tolerance then
7:     return true  \(\triangleright\) Density error exceeds permissible value
8:
9: \(\lambda_i = 0\)  \(\triangleright\) Reset constraint lambda, density criterion is satisfied
10: return false

In the constraint update for \(p_i\), we first test against our density error criterion. If this returns false, we deem further constraint iterations on this particle unnecessary.

We include an alternate early-out criterion that considers both a particle’s density error and that of its neighbours in algorithm 4.

Algorithm 4 Slackening Early-Out Criterion Extended

1: procedure RequiresFurtherIteration_EXTENDED(p_i)
2:   if p_i.iterations == 0 then
3:     return true  \(\triangleright\) Particle has not yet received constraint iterations
4:
5:   avg_error = \(|\rho_i - \rho_{0,i}|\)
6:   for particle \(p_j\) in \(p_i.neighbours\) do
7:     avg_error = avg_error + \(|\rho_j - \rho_{0,j}|\)
8:   avg_error = \(\text{avg\_error}/(1 + |p_i.neighbours|)\)  \(\triangleright\) Find mean density error
9:   tolerance = \(\zeta \rho_{0,i}\)
10: if \(|\rho_i - \rho_{0,i}| > \text{tolerance}\) then
11:    return true  \(\triangleright\) Density error exceeds permissible value
12:
13: return false

This alternate criterion results in more conservative constraint-skipping, continuing to perform constraint iterations on a particle as long as its neighbours are in violation of the constant density constraint. We found that this did not significantly impact the stability of our simulation, but higher computational cost was incurred by considering particle neighbours. We thus prefer the criterion of algorithm 3, but note that this is an area open for future analysis.
4.5.3 Discussion

The use of our density criterion allows us to skip all computation of particle neighbourhoods, weighting kernels and constraint updates in situations where particles already satisfy near-constant density. In practice, many particles already satisfy this in as little as 1 or 2 iterations. This can lead to significant improvements in computational efficiency while still supporting fluids of low compressibility. We discuss the impact our method has on performance in detail in chapter 5.

We are aware of two separate approaches of constraint-level adaptivity in the literature, but we believe our technique is new. (Köster 2016) describe a method for adapting the iteration count of PBF constraints based on a concept of level of detail, decreasing the iteration count dedicated to constraints based on a given particle’s distance to the camera or view frustum. As their approach assigns fewer iterations to far-away particles, we suspect this leads to higher compressibility as distance is increased, and a greater loss in fluid volume. This is an interesting area for future research. Additionally, their level of detail criterion depends on the presence of a 3D environment and camera setup, while our adaptive criterion is dimensionally agnostic and equally applicable in 2D simulations without modification.

(Bartels 2015) suggest an adapted iteration count per-constraint-type. They assign constraints a priority value based on their type (e.g. rope joints, rigid body constraints, deformable bodies), dedicating more solver iterations to high-priority constraints. However, this constraint type distinction does not apply to our PBF simulations, as we typically deal with only density constraints. In the case of visco-elastic simulation, we could optionally assign a separate priority to our connective spring model, but this is left as future work.

4.6 Rendering

Though rendering is not the focus of our dissertation, we will briefly outline our approach to the visualisation of our 3D simulations in this section.

We created a custom shader for the rendering of our particles, supporting GPU instanced meshes. This allows us to render several millions of particles in real-time on a GTX 560 Ti graphics card. We use the Unity engine’s support
for indirect draw calls, and supply the \textit{ComputeBuffer} of particle data directly to our shader every frame.

In all supplemental material, we render each individual particle as a single cube mesh, and determine colour in a fragment shader based on individual particle properties such as Z-axis depth, fluid density and velocity.

\section*{4.7 Discussion}

In this chapter, we have:

\begin{itemize}
  \item Detailed the approach and implementation of our PBF simulation on both the CPU and GPU.
  \item Introduced the problem of PBD constraints on the GPU and offered an implementation sketch of our Jacobi based solution
  \item Given a brief overview of a method to improve the efficiency of particle-based simulations using nearest neighbour search algorithm
  \item Outlined our uniform grid approach on the CPU, and detailed our fully parallelised GPU implementation of a fast fixed-radius nearest neighbour search using a Counting Sort implementation including a parallel scan
  \item Introduced a novel optimisation strategy for PBF simulations, based on an easily authored constraint satisfaction criterion
  \item Provided a brief description of our rendering method
\end{itemize}

We have arrived at a simulator capable of modeling several hundreds of thousands of particles in real-time that can be stably interacted with and modified at runtime. We will now present the supplemental material to this dissertation, and discuss the results of our simulation.
Chapter 5

Results

5.1 Guide to Supplemental Material

 Included with our report are a number of supplemental materials. As our project’s aim is to create a real-time interactive system, we felt it important to include video demonstrating our interaction with its capabilities. This video also acts as a reference point to alternate implementations and shortcomings of past work, and we provide comparative results between different techniques. We go into further detail in the sections below.

 In addition to our video demonstration of test scenes, we have included code archives of both CPU and GPU implementations of our simulator. We have split the given material into a set of folders, each demonstrating the results of different fluid models as described in chapters 2 and 3. Our CPU implementation is given in the form of Processing source files, and our GPU implementation comes as Unity engine project files (for use in Unity version 5.6.4).

5.1.1 Testing Environment

 All of our experiments were run on a PC running a 64-bit edition of Windows 7 with an Intel Core i3 4160 3.60GHz processor, GTX 560 Ti graphics card and 8GB RAM. We recorded our supplemental material using the video capture software Open Broadcaster Software.
In both our 2D (CPU) and 3D (GPU) simulations we implemented basic user interaction through the use of the mouse. The user is able to apply force to the fluid materials relative to the mouse’s position at runtime. In some simulations we implemented a moving boundary similar to a wave tank, to encourage dynamic fluid motion when not under direct user interaction.

We will now discuss the behaviour exhibited by each implementation of our particle-based fluid dynamics simulator. We refer the reader to the results shown in our supplemental video as a reference point for the results discussed.

5.2 Smoothed Particle Hydrodynamics

5.2.1 Overview

We begin our analysis by considering standard Smoothed Particle Hydrodynamics, as detailed in chapter 2.

In each experiment, we calculate and indicate the average density fluctuation from the desired fluid rest density $\rho_0$ as a percentage — e.g. with a rest density value of 1, an error value of 5% indicates the average density error across the entire fluid volume is 0.05. This is measured by taking the mean value across particles’ individual normalised density errors $\frac{|\rho_i - \rho_0|}{\rho_0}$.

High density errors result in a perceptible loss in fluid volume — thus, maintaining consistent and low density fluctuation is desirable. In our SPH simulations, we choose the maximum stiffness constant $k$ for our equation of state (section 2.2.3) to be the maximum exhibiting stable behaviour with our chosen timestep value $\Delta t = 0.15$.

5.2.2 Analysis of Compressibility

Our basic implementation of SPH with low viscosity, no surface tension and the standard equation of state given by equation 2.9 is shown first (Scene 1). We note the high density error value in the range of 20% on average through the simulation. The fluid exhibits visibly high compression when under pressure either through interaction with the boundary or when interacted with by the user.
When an excess of pressure is applied, the simulation becomes visibly unstable, "exploding" erroneously before re-settling.

To address this high compressibility, we implemented the Tait equation of state (equation 2.10) with gamma value $\gamma = 3$. This resulted in a much lower density error value of 0.5% on average throughout the simulation. However, we note that the simulation became instantly unstable with a timestep value identical to that of our previous results. The results shown are thus with a timestep value $\Delta t = 0.05$ in Scene 2. This requires more than double the simulation iterations per-frame of our standard equation. As such the computational cost of using the Tait equation is a factor of 2 larger than our basic SPH implementation, which we deem unsuitable for larger scale real-time simulation.

We now return to our standard equation of state to investigate the effects of the standard SPH viscosity model (section 2.2.4). We increase our fluid viscosity value by a factor of 10 ($\mu = 10$) and observe the expected fluid behaviour (Scene 3). Flow is significantly damped, with the effect of a much thicker fluid substance. The result of increasing the viscosity to double that of the previous simulation ($\mu = 20$) are then shown in Scene 4. It can be seen that the simulation becomes unstable, exhibiting erroneous jittering due to the large explicit viscosity forces attempting to over-smooth the velocity field.

This concludes our experiments with the Smoothed Particle Hydrodynamics model of fluid simulation. We now turn to our implementation of Position Based Fluids, which addresses several of the shortcomings of SPH.

### 5.3 Position Based Fluids

#### 5.3.1 Compressibility

Our experiments with Position Based Fluids yielded desirably low density fluctuations. The density error after 3 constraint iterations of PBD yielded similar compressibility to that of our SPH model with the Tait equation, yet remained stable at a significantly larger timestep value $\Delta t = 0.15$. This can be seen in Scene 5 of the supplemental material video. We note that decreasing the number of constraint iterations does not result in unstable behaviour, but merely higher density fluctuation (greater compressibility).
5.3.2 Surface Tension

We then implemented the standard surface tension term of (Macklin 2013). With low surface tension values this results in the desirable formation of filament-like structures and "blobs" of fluid material. This is demonstrated in Scene 6.

Increasing the surface tension coefficient with this model of surface tension yields visibly unstable ("explosive" behaviour under high pressure, as seen in Scene 7. We believe this is due to the presence of position-level surface tension corrections being applied in conflict with the constant density constraints.

We then implemented the surface tension model of (Akinci 2013), removing the surface tension terms of (Macklin 2013). With a surface tension value of $\gamma = 3$ we see comparable behaviour to that of (Macklin 2013), with filament and blob-like formation. (Scene 8)

In Scene 9, we raised this surface tension coefficient significantly ($\gamma = 20$), and note that strong surface tension behaviour can be observed that remains stable under user interaction forces. We increased this value to $\gamma = 100$ and lower the timestep to ensure stable behaviour. Results of this are shown in test scene 10.

5.3.3 Viscosity

We now consider the modeling of high-viscosity fluids in our PBF simulator. Our previous PBF experiments feature a fixed XSPH viscosity value $c = 0.3$, using standard non-iterative XSPH (equation 2.12). In Scene 11 we raise this to a value of $\gamma = 1$, and note that it becomes unstable under high pressure. This is exacerbated as the viscosity value is raised.

The results of our iterative XSPH viscosity model (section 3.4) are shown in Scene 12. With no reduction in timestep, the simulation is fully stable with significantly higher viscosity values (shown with $\gamma = 10$).
5.3.4 Viscoelasticity

We begin our investigation of viscoelastic behaviour by showing a full implementation of the viscoelastic fluid simulation approach given in (Clavet 2005). The time integration scheme used results in unstable behaviour under high pressure, indicated in Scene 13.

We then present our modified, position based implementation of viscoelasticity using compliant XPBD-style constraints. We demonstrate user interaction with several materials of varying visco-elasto-plastic parameters. Our model is able to stably handle higher elasticity values to that of (Clavet 2005), and as such is capable of exhibiting a wider range of phenomena such as high tensile strength, and ductile and brittle fracturing. The results of varying material parameters can be seen in scenes 14, 15 and 16.

5.3.5 Multiple Phases

We demonstrate the flexibility of our simulator in its support of multi-phase fluids. In test scene 17, two fluids with a density ratio of 1 : 6 interact, the heavier fluid sinking below the lighter one. We also demonstrate the capabilities of our modified surface tension model (section 3.8.2) in supporting different miscibility values.

On the left hand side, we simulate fluids with zero miscibility. On the right, a fully miscible simulation is shown. We note the separation between phases in each condition. We note the relative ease with which the fluid phases separate in the fully immiscible condition, indicating the success of our adapted model.

We conclude our section on multiple phases with a demonstration of 3-way coupled fluid interaction in Scene 18. Three fluids of varying viscosity, mass, surface tension and elasto-plasticity mix with one another and remain stable under the influence of user interaction in real time.

5.3.6 Slackened Position Based Fluids

Our final point of analysis is in the effects of implementing our novel adaptive optimisation criterion, slackening.
We first demonstrate the adaptive iteration count of the slackening approach and its ability to maintain low density fluctuation (Scene 19). We then compare this with the results of a non-slackened implementation of PBF (Scene 20). In the version with slackening, we colour-code particles according to the number of PBF iterations they have received. We tabulate these results below and include a sample frame detailing our visualisation (Figure 5.1).

<table>
<thead>
<tr>
<th>Simulation</th>
<th>Mean density error</th>
<th>Mean FPS</th>
<th>Max iterations</th>
<th>Avg. iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>1%</td>
<td>9.3</td>
<td>30</td>
<td>30</td>
</tr>
<tr>
<td>Slackening</td>
<td>1%</td>
<td>10.5</td>
<td>30</td>
<td>22</td>
</tr>
</tbody>
</table>

Figure 5.1: Sample frame from slackened 2D PBF simulation. Hue indicates iterations — particles receiving maximum iterations are coloured orange. Note the lighter areas near the free surface where pressure is low, fewer iterations are required to resolve density constraint.

We see a slight improvement in performance with slackened PBF. Low density error is maintained by the chosen threshold $\zeta = 0.01$ (1% permissible fluctuation), and the timestep was identical in each case. In comparison to standard PBF with 30 constraint iterations, we are able to maintain an acceptable density variation with an average of 22 iterations per particle.
Chapter 6

Conclusion

6.1 Summary of Work Completed

In this project, we implemented a series of Lagrangian fluid simulation methods, with the aim of stably simulating a diverse range of fluid behaviours at interactive frame rates. We provided a brief introduction to the theory behind Lagrangian fluid discretisation, and described the methodology of both Smoothed Particle Hydrodynamics and Position Based Fluids.

A simple SPH fluid solver was implemented, and different approaches to solving high density fluctuations were presented, implemented and discussed.

Motivated by the issue of SPH fluid compressibility at interactive frame-rates, we implemented the Position Based Fluids method of (Macklin 2013). We included support for vorticity confinement to address artifacts of numerical damping, and added support for variable viscosity using XSPH. We also implemented a second-order accurate variation on the standard numerical integration of position based dynamics. This led to increased simulation stability and an improved constraint iteration requirement.

We extended our PBF simulator significantly to support a greater range of fluid behaviour. We implemented the iterative XSPH technique of (Alduán 2015) to add support for high viscosity fluids. We then implemented the versatile surface tension model of (Akinci 2013), and further extended the model to allow varying miscibility between different fluid phases.
Viscoelastic behaviour was introduced and implemented in accordance with the work of (Clavet 2005). We demonstrated the conditional instability problem of the resulting model, and proposed a reformulated variation of visco-elasticity for position based fluids. We implemented a new visco-elastic simulation model with support for plastic deformation, based on unconditionally stable XPBD-style compliant constraints. We then demonstrated its efficacy in supporting a wider range of visco-elastic behaviour than that of Clavet et al.

We introduced slackening, a novel simulation-level optimisation technique for PBF that yields greater computational efficiency. Slackening reduces the number of required constraint iterations in a PBF simulation while ensuring density fluctuations are kept within an acceptable range. We provided a comparison between standard and slackened PBF in our analysis.

Finally, we implemented a fully parallelised 3D PBF simulator on the GPU using DX11 Compute shaders. We implemented the Counting Sort neighbourhood search algorithm to improve memory access latency. Our implementation is capable of simulating up to 100,000 PBF particles in real-time on a GTX 560 Ti at a rate of 60 frames per second. Our simulator can be interacted with by applying forces to the fluid body at run-time with no loss in simulation stability.

6.2 Motivation for Future Research

Though we have implemented a number of different Lagrangian fluid simulation techniques, there remain many areas of interest that remain unexplored. We consider the following areas of interest for future work:

- **Viscoelasticity on the GPU**
  Our viscoelasticity model depends on the continual creation and deletion of references between connected particles, using a secondary *Spring* structure. As such, each particle must retain some reference to a dynamically resizable list of *Springs*. This requires significant work to maintain, and will result in potentially very scattered memory accesses, irrespective of the data structure used at each particle. A consequence of this is that our viscoelasticity model is ill-suited to the GPU. We see the work of (Barreiro 2017) as a promising direction for future research, as they demonstrate the ability to simulate a range of viscoelastic behaviour in realtime on the GPU.
• **Physically motivated and performant viscous flow**
Our current model of PBF based viscosity is very easy to implement and stable with large timesteps. However, its performance impact scales with the number of required iterations. Thus, highly viscous fluids can be less efficiently simulated. Additionally, as it is a mere extension of classic XSPH, it aims only to progressively smooth the velocity field. As a result, it cannot exhibit certain viscous flow phenomena such as buckling and coiling. We would like to investigate further alternative models of viscous flow such as those of (Barreiro 2017, Takahashi 2014) capable of demonstrating these effects.

• **Heat propagation and melting effects**
Currently our simulations contain no notion of temperature in a fluid volume. We imagine support for basic heat diffusion and melting effects to be a simple extension of our existing system. It would simply require the addition of a heat transfer function at a particle level, that then maps to particle viscosity. More complex material behaviours such as freezing to a rigid state or boiling to form a gas are less trivial, but are in theory possible. We note that all three material types are present in the NVIDIA Flex (Macklin 2014) system, and a fully unified phase transition system is an interesting future problem.

• **Improvements to the slackening early-out criterion** The early-out criterion we have proposed is very simple to evaluate, but we believe there are numerous ways in which it could be improved. For instance, we could choose to consider neighbours’ relative velocities in order to rule out further iterations on particles near a resting state. This could potentially allow us to increase the density error tolerance value and yield greater performance.
Bibliography


