Position-Based Viscoelastic fluid simulation

by

Yuanqi Huang, BEng(Tech)

Dissertation

Presented to the

University of Dublin, Trinity College

in fulfillment

of the requirements

for the Degree of

Master of Science in Computer Science

(Interactive Entertainment Technology)

University of Dublin, Trinity College

September 2016

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Yuanqi Huang

August 24, 2016

Permission to Lend and/or Copy

Acknowledgments

Firstly I would like to thank you my supervisor Dr. Michael Manzke for all the advise, discuss and support through the dissertation.

I would like to thank my family for their support and encouragement through the whole year.

Yuanqi Huang

University of Dublin, Trinity College
September 2016

Position-Based Viscoelastic fluid simulation

Yuanqi Huang
University of Dublin, Trinity College, 2016

Supervisor: Dr. Michael Manzke

Deformable material has become an active area in animation. A huge number of techniques have been explored for achieving a large deformation in a more plausible result. Particle-based representation has its natural advantage for describing reasonable deformation. Specifically, deformable materials are supposed to be treated as continuous flow for more realistic behaviour. Thus research of non-Newtonian fluid has become a popular area as an alternative solution. Most of these approaches are built upon existing fluid simulation. Among all the fluid simulation, Position-Based-Dynamic fluid is a novel method which has a better control over particles but in less complex level.

This dissertation explored the viability of a new method for viscoelastic fluid building upon PBD fluid model. Two possible approximations (spring and strain tensor) were implemented for viscosity and elasticity behaviours. The full implementation was simulated on GPU platform with visualisation. The results showed that spring model does not fit the PBD, but that strain tensor model works partially. The limitations and reasons were deeply discussed in many aspects.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Fluid animation techniques has been developed 20 years since 1996. Commercial softwares like RealFlow, MAYA, and Houdini are widely used in todays movie effect and game development industries. Requiring more realistic effect but consuming less computational resources make this a highly popular area in research. Because ad-hoc particle system method introduced into fluid simulation, fluid simulation has been extended into a more general physical solution. Gas, cloud, explosion and hair, etc can be treated as a particle system simulation to some degrees. Even further, particles have the advantage for large scale deformation object. Considering its better representation of randomly deformation in general case, many researchers tend to find a unified particle system for various physical simulation. In fluid simulation area, integration technique like Semi-Lagrangian advection help to solve Navier-Stokes equation in relatively large time step, which increases the possibility of achieving more complex and accurate physical behaviour in simulation.

Not only in animation, particle based simulation has also been used in academic researches a lot, like mechanic or mathematic. Obviously, academic research and commercial use share many common techniques in many aspects. However, unlike most of the researches, fluid simulation in game industry or movie effect have to consider the limitation of memory and computation ability in platform all the time. Plausible result with rich enough features but using less resources requires approximation in physics description, novel integration scheme and specially designed data structure. Thus, in this area, any possible new technique will be tried by combining with previous good models.

Apart from the performance of fluid simulation, easy to use for Animation artists is another consideration for invented method. More intuitive parameters and rich features could help artist obtain better control of the fluid and its different formation.

To sum up, all the factors above involves finding an adequate physics formulation, which in most cases in differential form, and a numerical approximation scheme to discretise it.

1.2 Viscoelastic Fluid

Non-Newtonian fluid is an interesting material in our daily life, how to implement it in graphic animation is popular. Viscoelastic material is one kind of non-Newtonian fluid, which has both fluid and elastic responses under different conditions. i.e. this flow show viscosity property when weak force has been applied like honey, toothpaste, however such material will bounce like a rubber under huge force or tension. As a non-Newtonian fluid, viscosity acts as a measurement of a fluid's ability to resist gradual deformation by force or tension. Anther obvious property is that the particles in the fluid are large than that in the newtonian fluid in size, therefore, when sudden force happen, those particles have no time to react and become solid.

The main difference between viscoplastic and viscoelastic is the yield tress. From the aspect of rheological, viscoplastic material will only deform after the yield stress has been reached. In the contrast, Viscoelastic material will deform at any application of stress. Considering this main difference, an reasonable approximate model of yield stress threshold has to been chosen for simulation.

Unlike traditional newton fluid, whose rate of strain is constant, both of these non-newton fluids' rate of strain depend on stress applied. There are several models of viscoplastic liquids include Bingham plastic, Hershel-Bulkley and Casson [6] As for viscoelastic models, Maxwell and Kelvin-Voigt model, which provide different degrees of complexity for realistic behaviour.

1.3 Framework

For simulating the fluid behaviour in its liquid phase, framework of computational fluid dynamics(CFD) is required. Generally speaking, the most common model relies on Eulerian formulation where physical quantities are sampled on regular grid. This grid-based method suit well for the classical Newton fluid simulation like water. In this method, the whole space will be subdivided into cells, which shows the potential drawback that delicate fluid behaviour is limited by the resolution of the grid. The fundamental basis of all the CFD problem is Navier-Stokes equation. Thus, for single flow simulation, this equation is always simplified by removing the viscosity terms to yield the Euler equations. Further simplification, by removing terms describing vorticity yields the full potential equations [7]. However, the big problem of grid-based method is that fluid free surface have to be tracked in some way, which brings extra computation all the time.

Instead of using Eulerian formulation of grid-based framework, this thesis choose Lagrangian formulation on particle-based representation. The most popular one is Smoothed Particle Hydrodynamics (SPH), which is introduced by Monaghan [8] in 1977 for astrophysical problem. This method represent fluid into a small volume responded to forces as a particle. All the particles has certain spacial distance called smoothing length, over which their physical quantities are smoothed by smoothing kernel. In other words, individual particle's physical quantities can been obtained by summing all the other neighbouring particles' which lie within the smoothing length. Since the easy implementation of such framework, and good response for gravity, pressure and viscosity, this thesis will use such framework for fluid phase simulation.

1.4 Related Work

1.4.1 Discretization technique

In computer graphic field, physical simulation is a well established research topic that different methods have been developed.

[9] has mentioned Finite Element Method (FEM) , in which the domain of the material has been partitioned into distinct sub-domain. Each sub-domain share common

nodes on boundary with adjacent element. Certain function would be used to control the behaviour of entire domain but only by finite number of parameters.

Similarly, Finite Differences Method (FDM) [10] divide the entire domain into regular lattice which has challenge with irregular material.

In [11], Finite Volume Method (FVM)has been used to achieve arbitrary deformation in a geometrically intuitive way.

[12] shows another way of deformable material simulation called boundary element method (BEM). In which, the partition only be applied on the boundary of the object, and create n-dimensional non-overlapped elements. A function of displacement will be interpolated between each nodes. After each displacement, compute 3n linear equations through desired boundary condition. Such discretisation is widely used in many engineering areas, but involved many mathematic process.

However, compared to those partition techniques above, particle-based technique is more intuitive. [13] shows particle-based method for deformation materials simulation. In which they represent a piece of cloth as a set of mass particles with springs interpolated, integrated particles by Euler integration scheme. Meanwhile, for better coherent result, artificial viscosity has been added. However, such method is better for known shape simulation. To solve this problem, [14] point out a general solution for Geometrically Complex Deformable Solids.

Compared to traditional particle method, [15] has used a more novel way to simulation solid objects, called position-based dynamic. Since the model is highly controllable, PBD provides fast and plausible result. Most of force based dynamic follow the Newton's second law of motion, either internal and external forces are computed and accumulated in the time interval. Potential challenge like overshooting and penetration happen in those implementation all the time. In the contrast PBD could overcome these problems easily.

No matter for rigid body or deformable objects, the implementation focus on robust, efficient and plausible solution. Whereas, traditional particle approach calculates external forces like collision impulse and global force, and internal force including pressure and elastic forces at least twice, which mean 2 levels of integrations will be handled. Some simulations optimise scheme by using impulse instead of forces to avoid one level of integration.

Even worse, traditional particle approach control the positions of the vertex in an indirect way. In other words, fluid particles' positions on screen are result of two integrations, unrealistic conditions happen frequently and system could suffer numerical problem. i.e, in collision handling case, it's not hard to observe that some vertices could penetrate into others, which could damage the whole system. In the deformable cases, it's not hard to find that the whole system energy could exceed, which break the physics law and will damage the whole simulation as well.

In general, Position-Based-Dynamics has the advantage of controllability which overcomes collision and overshooting problems easily. Because PBD control the positions of vertices directly, which not only guarantee staying outside of the collision objects, but also make vertex-level manipulations more easily.

[16] introduce a new method that how to use position-based dynamic to achieve deformable material, the method called shape matching(SM). Briefly, it uses particles to represent certain shape. After each integration, goal translation and rotation has been calculated by using central point of the shape and it's configuration. The numerical positions of particles are resulted from external force like gravity and collision, this transformation will be modified by goal transformation to match the general shape in geometric aspect. This SM method help position based dynamic to be a more versatile solution for both rigid and soft materials.

1.4.2 Non-Newtonian Fluid

For the Non-Newtonian fluid simulation, many particle based simulations have been introduced.

Simon Clavet in [17] followed SPH framework for particle representation, but optimised the material with double density relaxation to adjust positions after artificial pressure. For the viscoelastic terms, they introduce changeable springs between particles. Spring force is for elastic phenomenon, adjustable spring rest length is to achieve plasticity. Viscosity is done by exchanging radial impulse determined by velocity difference. This method requires special data structure for storing the spring rest length and adding/removing the springs. This method share the same spring idea as that for cloth simulation.

Afonso Paiva provided a plausible result in [18] and [2] by adding an extra elastic force term into a particle based fluid simulation. The fluid simulation used classic

SPH framework, followed Lagrangian integration, modified final velocity with viscosity correction and XSPH velocity correction. As for elastic term, they calculated stress based on stress tensor. Instead of introducing a new elasticity term into the fluid equation, they change the viscosity coefficient to achieve different behaviour. For melting model, they use jump number which is linearly proportional to temperature equation to control the stress force.

[19] followed this method to achieve Jet Buckling simulation. The final result is pretty plausible.

Tolga G. Goktekin in [20] provide a simple implementation based on fluid simulation mentioned in [3]. Utilise the Navier-Stokes equation with an additional elastic term, update the acceleration of the particle every time step. Since computing deformation is impractical in Euler formulation, they compute strain by integrating strain-rate and get elastic strain by finding difference of total strain and plastic strain. Von Mises's criterion is chosen to determine when plastic flow should occur.

Similar approach as [20] was used in [21].

1.5 Motivation

To find a new way of simulation viscoelastic fluid by using PBD model, this thesis explored the possibility of embedding Elasticity force and Viscosity force into normal Position-Based Fluid. This approach has not been tried in the related area. Specifically, Elasticity force will be calculated in two ways, one is strain tensor elasticity inspired by [20], another is a response of spring with fixed rest length inspired by [17]. Since the highly orthogonal nature of elasticity, viscosity and other internal forces in the fluid mechanics, those two individual internal forces can be applied directly into normal fluid calculation loop. The reason for using PBD fluid is that PBD is simple model with direct control on the position of particles. It provides a plausible result with less resource consumption. If this approach achieve a plausible physical behaviour for viscoelastic fluid, it will reduce the complexity in computation comparing to using normal particle-based fluid [20], and will enhance the ability of describing deformable object for PBD.

Chapter 2

Fluid Simulation

2.1 Chapter Overview

This chapter discusses the theory and basic knowledge for position-based non-Newtonian fluid simulation, note that most of those theory works well for Newtonian fluid as well. In this chapter, only physical and mathematic theory will be included, next chapter will provide an easy visualisation method. Generally speaking, this implementation provide a good start point to develop a real-time interactive application for viscoelastic material. As an experimental implementation, the priority of this method would be put on repeatability and high effectiveness but on physical accuracy.

This chapter start with the discussion on some general concept of flow mechanics in 2.2, which clarify the approximation of Navier-Stokes equation from Newton's second law of motion. Meanwhile a brief discussion about the distinctive point of view in Eulerian and Lagrangian integrations was given, which will help reader to have a better understanding of governing equation.

After the introduction of the fluid equation, 2.3 would explain the famous Smoothed Particle Hydrodynamics framework. Section 2.4 tells the Position-based dynamics general algorithm and its advantages. In 2.5 section, mathematic model of each term in Navier-Stokes equation is explained in detail, including how to use SPH in calculation. 2.6 discusses the Smoothing kernels that are used in this simulation. 2.7 shows the implementation algorithm.

2.2 Governing Equation

2.2.1 Quantities

To describe properties of the fluid particle movement , fundamental quantities including mass, density, pressure and velocity needs to be updated every time step. The mass $\boldsymbol{m}$ for each particle specify how much weight it is, directly influence the inertial of the fluid. The density ρ measures the mass per volume, it's defined as: $\rho \equiv \lim_{\triangle V \to L^3} \frac{\triangle m}{\triangle V}$.

Pressure

Pressure p is a scalar quantity which is defined as a measurement of force per unit area, in simulation it is always described as: $p \equiv \lim_{\triangle A \to 0} \frac{\triangle F_n}{\triangle A}$. The force $F_n$could be any external force like gravity, and act in the normal direction on the fluid surface. Pressure act in all direction inside the water domain, and push all the particle from high pressure area to lower one until them reaching a balance condition.

Velocity

Velocity V is an another fundamental measure for how fast a particle passes at certain point in space on certain time. V both specify the scale and the direction, it's a vector quantity. The velocity field influence most of the other properties like dynamic pressure for newtonian fluid, or viscosity force and elastic force for non-newtonian fluid. More important, instead of acceleration impulse will be directly applied on velocity, which means final position will relay exclusively on velocity in this model.

Viscosity

Viscosity compensates the difference of particle velocities, which is always proportional to relative velocity over time. It can be compared to friction to some degrees. Viscosity η is responsible for shear stress, which is a kind of tangential force on the flow surface. Viscosity can be comprehended as a measure for how much momentum has been transferred in local area. Viscosity can be stated as dynamic term η (force) or kinematic term $v = \frac{\eta}{\rho}$.

Surface tension

Surface tension σ is the elastic tendency appear on fluid surface which minimise surface area possible. This phenomenon always happen at the border of different medium (liquid-air, or various density liquid). Brief explanation is that interface molecules attracted by greater cohesive force of the neighbouring molecules than the adhesive force of molecules in air. The inwards force cause the surface of fluid behave as if it was covered with a elastic membrane.

Shear stress

From two plates model 2.1 point of view, which treat material as two plates with imaginary fluid in-between, movement of upper plates causes a stress which is parallel to the fluid surface is called shear stress τ. τ is calculated as: $\tau = \frac{F}{A}\left[\frac{N}{m^2}\right]$. Another variable derived from this model is shear rate $\dot{\gamma}$ or D, which is defined as velocity of upper plate is divided by the distance h between the two plates: $\dot{\gamma} = (D =)\frac{V}{h}\left[s^{-1}\right]$.

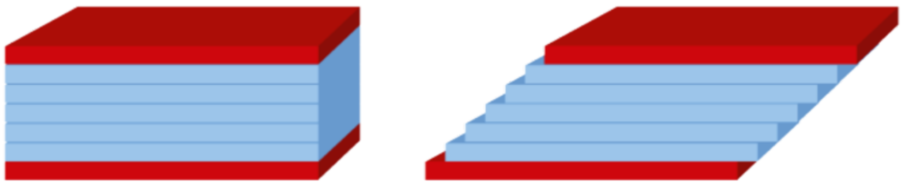


Figure 2.1: The Two-plates model - laminar flow taking the shape of infinitesimally thin layers [1]

From Newton law, the shear stress is shear rate times viscosity: $\tau = \eta \cdot \dot{\gamma} \Rightarrow \eta = \frac{\tau}{\dot{\gamma}}$.The vital question is whether the viscosity η does or does not change with the change of shear rate. This question draw a line between the Newtonian and non-Newtonian fluid behaviour.

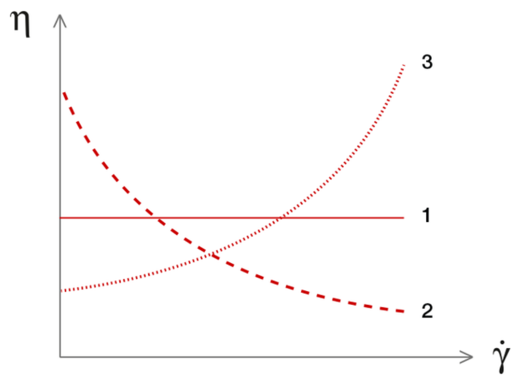


Figure 2.2: Viscosity function (dynamic viscosity over shear rate): 1 ... Newtonian liquid 2 ... shear-thinning substance 3 ... shear-thickening substance [1]

As one can find in diagram 2.2, if fluid inner resistance (viscosity) is independent of the external force, it's called Newtonian liquid with ideal viscosity (curve 1). In this fluid, however large the force applied, the deformation happens immediately.

In the contrast, non-Newtonian liquid's viscosity changes with shear rate. Some flow shows shear-thinning behaviour (curve 2). Their viscosity decreases when the shear rate increases, which mean huge force could make fluid deformed faster. For other flow the viscosity increases with increasing shear rate, that is called shear-thickening (curve 3).

2.2.2 Newton to fluid mechanics

We all know about the Newton's second law: $F = m \cdot a$. This states that an object's acceleration a depends on the mass m and the force F applied on. For dynamics description, Newton's second law is always interpreted from Lagrangian point of view. In the classic field theory, Lagrangian specification of field is a way of looking at motion objects, where observer focus on specific particle along with it over time and space. This can be visualised as sitting on a bout along with the flow. On the other hand, Euler specification of field is a way of looking at fluid motion where observer focus on specific location in space over time. This can be visualised as standing on the bank and stare at single location of the river.

In Euler filed, flow velocity could be described as:

$$u(x,t)$$

this is a function of position x and time t. Alternatively, it has Lagrangian description:

$$U(x_0,t)$$

in which x_0 is a time-independent vector, particle's initial position. The two specification are related as:

$$u(U(x_0,t),t) = \frac{\partial U}{\partial t}(x_0,t)$$

This relation in flow field provide a useful material derivation (or called particle derivation). If we have a velocity field v and some v related function in Euler specification $F(x,t)$, it's rate of change can be represented as:

$$\frac{DF}{Dt} = \frac{\partial F}{\partial t} + v \cdot (\nabla F) = \frac{\partial F}{\partial t} + u\frac{\partial F}{\partial x} + v\frac{\partial F}{\partial y} + w\frac{\partial F}{\partial z} \tag{2.1}$$

∇ donates the gradient with respect to x, $\frac{\partial F}{\partial t}$ expresses local rate of change of F and $v \cdot (\nabla F)$ the convective rate of change of F. This is result of chain rules.

Once get this bridge function, insert acceleration function into this. We got:

$$F = m\frac{Dv}{Dt} = m(\frac{\partial v}{\partial t} + v \cdot (\nabla v)) \tag{2.2}$$

If we replace the mass with density, we could write:

$$F = \rho(\frac{\partial v}{\partial t} + v \cdot (\nabla v)) \tag{2.3}$$

Until now, we could describe flow force by dividing force applied on fluid into internal forces and external force like gravity or electromagnetic force:

$$\rho(\frac{\partial v}{\partial t} + v \cdot (\nabla v)) = F_{Internal} + F_{External} \tag{2.4}$$

From here, The fluid is called incompressible when $\nabla \cdot v = 0$ (divergence of velocity field is zero), meaning that there are no source in the velocity field. Also, since the pressure force and viscosity force are orthogonal (one is normal direction of surface, another is parallel direction of the surface), we could simply split $F_{Internal}$ into $F_{Pressure}$ and $F_{Viscosity}$:

$$\rho(\frac{\partial v}{\partial t} + v \cdot (\nabla v)) = F_{Pressure} + F_{Viscosity} + \rho g \tag{2.5}$$

The pressure force as mentioned in previous section depends on the difference in pressure p, which push the flow particles from high pressure area to low pressure area. Therefore we model them with the negative gradient of the pressure field $-\nabla p$, whose direction is from high to low:

$$F_{pressure} = -\nabla p \tag{2.6}$$

As for viscosity force, it is simply threat as compensation for the velocity difference. The Laplacian $\nabla \cdot \nabla$ (written ∇^2) operator measures how far a quantity is from its neighbours' average, thus the viscosity is modelled as:

$$F_{Viscosity} = \eta \nabla^2 v \tag{2.7}$$

Since we model non-Newtonian fluid, we would modify the Navier-Stokes equation by adding additional Elastic force $F_{Elastic}$ to achieve elasticity behaviour as:

$$F_{Elasticity} = \mu_e \nabla \epsilon \tag{2.8}$$

In this, μ_e mean elastic coefficient, ϵ is elastic strain tensor.

Combine all of them, we get the Navier-Stokes equation:

$$\rho(\frac{\partial v}{\partial t} + v \cdot (\nabla v)) = -\nabla p + \eta \nabla^2 v + \mu_e \nabla \epsilon + \rho g \tag{2.9}$$

This equation is a widely popular basis for fluid models. In next section 2.3, SPH framework principle will be discussed and 2.4 will combine this equation and SPH to produce mathematic model that this thesis used.

2.3 SPH

Smoothed Particle Hydrodynamics is a technique introduced by Gingold and Monaghan [8] and Lucy [22] for astrophysical dynamic problems. In the SPH model of fluid simulation , the value of any physical quantity $A(r)$ at a position r have to be interpolated from set of neighbouring particles. Its summation interpolation is:

$$A(r) = \sum_j A_j \frac{m_j}{p_j} W(|r - r_j|, h) \tag{2.10}$$

As one can notice from the function, the contribution of other particles to the interested quantity is weighted by the distance to the central particle, which is governed by the kernel function W mathematically. To save computation, Kernel Function goes down to zero if the distance is further than the kernel length h (unlike *Gaussian kernel*, where a tiny contribution even at the infinite distance will be included). The 3D kernel looks like diagram 2.3.

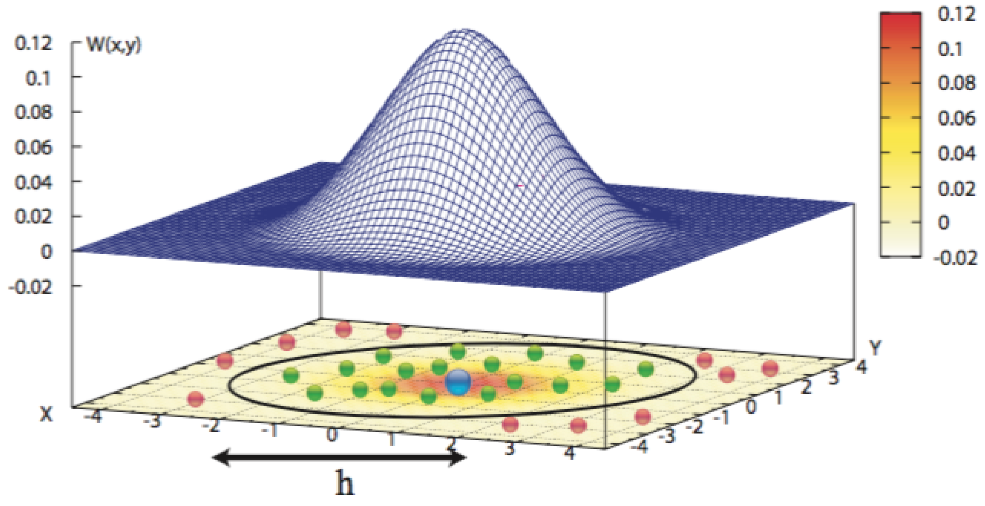


Figure 2.3: Quartic smoothing kernel: the particles farther than the smoothing length h are discarded. [2]

The biggest advantage of SPH is the spatial derivatives can be simply calculated by using the smoothing kernel. Thus the differentiation of any physical quantity could be computed as:

$$\frac{\partial A(r)}{\partial x} = \sum_j A_j \frac{m_j}{p_j} \frac{\partial W(|r - r_j|, h)}{\partial x} \tag{2.11}$$

The gradient becomes:

$$\nabla A(r) = \sum_j A_j \frac{m_j}{p_j} \nabla W(|r - r_j|, h) \tag{2.12}$$

Similarly, the Laplacian becomes:

$$\nabla^2 A(r) = \sum_j A_j \frac{m_j}{p_j} \nabla^2 W(|r - r_j|, h) \tag{2.13}$$

According to [23], this approximation is called classical gradient approximation formula (CGAF). This approximation is not perfect for all the quantities of the particle simulation. Since some basic physical law like symmetry of forces and conservation of momentum are not guaranteed, we would like to give some modifications to this approximation.

Derived from derivative rule of a product, we could write gradient as

$$\rho \nabla f = \nabla(\rho f) - f \nabla \rho$$

Thus another way of computing gradient with smoothing kernel is approximated as

$$\nabla A_i(r) = \sum_j (A_j - A_i) \frac{m_j}{p_j} \nabla W(|r - r_j|, h) \tag{2.14}$$

This is called the difference gradient approximation formula (DGAF). This approximation could be similarly apply on Laplacian function in the SPH model.

In many cases of physical description, we have to consider the symmetrical format of the model. To deal with the symmetric force law, derivative of a quotient function has been considered:

$$\frac{\nabla f}{\rho} = \nabla(\frac{f}{\rho}) + \frac{f \nabla \rho}{\rho^2}$$

which leads to a new form of gradient calculation:

$$\nabla A_i(r) = \rho_i \sum_j m_j (\frac{A_i}{\rho_i^2} + \frac{A_j}{\rho_j^2}) \nabla W(|r - r_j|, h) \tag{2.15}$$

This is called symmetric gradient approximation formula (SGAF) .

When is comes to choice of the proper formula in the simulation, the [23] has conclusion that DGAF is much more accurate than the other two formula. In their comparison, the SGAF and CGAF both show great error at the boundary area, and even increased further along with the test parameter increase. However, one should note that the conclusion in [23] is drew from a simple comparison, and the condition is limited.

In this thesis, most of the approximation formula will use DGAF, however only few of them we choose to use SGAF for the symmetrical consideration.

One may notice that by using SPH, we could simply solve the force equation mentioned in last section. The only problem left here is how to choose the correct kernel function, which will be discussed in the 2.6 section later.

2.4 Position-based Dynamic

As mentioned in the motivation in the first chapter, this thesis insist to use the PBD fluid as the foundation for the viscoelastic fluid. The reason to stick on PBD instead of the more common particle method will be obvious once we know the advantage of the PBD.

Traditional approach

Traditionally, the when we talk about using particle system for any simulation, we intuitively treat particle as a smaller solid ball which follow Newton motion law. This simple way of manipulating particle as solid object is studied widely and researchers try to a unified way of dealing with all object. However, it is not hard to notice the conflict here, solid object are almost represented only by shape meshes covered, but for water-like object, particles inside have totally different behaviour.

In traditional approaches, every particle is controlled by the newton's second law. In every time step, accelerate is calculated trough looking for the forces that possible appear. These forces can be divided into external and internal. Elasticity and viscosity force are typical internal force which is the main focus of this thesis, Gravity and collision force is the universal external force that every simulation would have. Through various force models, the accumulated accelerate will be get and applied to the velocity. Afterwards, the position can be computed by the velocity.

This typical way of integration actually never modify the position which will be the outcome for the scene, twice integrations could generate numerical problems. Also less consideration of treating material as a whole object will bring the overshooting and energy gain problems along simulation goes. To solve this and get more control, some model use impulse to direct change velocity.

Another problem is that its really hard to manipulate the object if some position based moves are used. In scene like water particles are poured onto cloth object , collision response at the interface would become a headache. Or the cloth would like to be attached on the surface of another object. Considering these, a approach which has direct control of the position and treats all the particles as whole object in microscopic point of view is needed.

PBD general approach

The main advantage of PBD is it provides a way which naturally maintain the energy conservation, and overcome the overshooting. In general way, it treat the material as a whole and try to use constraints to describe expected behaviours.

The core technique of PBD is the constraints which control the behaviour of the particles, they directly manipulate the positions. If we have a particle $i \in [1, \ldots, N]$ with position x_i and velocity v_i, the constraints functions are write as $C_j(x_i, \ldots, x_n)$. Through the constraints calculation, some parameters are get and it will be applied to the change of position $\triangle x_i$.

The general dynamic simulation loop could be model as algorithm 1 with step $\triangle t$

Algorithm 1: PBD general simulation loop

```
1  repeat
2      forall the particles $i$ do
3          $v_i \leftarrow v_i + \triangle t f_{external}$ ;
4      end
5      forall the particles $i$ do
6          $x_i^{predict} \leftarrow x_i + \triangle t v_i$ ;
7      end
8      forall the particles $i$ do
9          CollisionConstraints $x_i^{predict} \leftarrow x_i$ ;
10     end
11     for $i < IterationsTimes$ do
12         forall the particles $i$ do
13             Apply Constraints : $x_i^{predict} \leftarrow C_i(x_i^{predict}), \ldots, C_m(x_i^{predict})$ ;
14         end
15     end
16     forall the particles $i$ do
17         $v_i \leftarrow \frac{x_i^{predict} - x_i}{\triangle t}$ ;
18         $x_i \leftarrow x_i^{predict}$ ;
19     end
20     Some velocity Optimisation and Correction;
21 until Simulation time over;
```

The most important part of this algorithm is the iteration loop in line 11-14. The estimated positions $x_i^{predict}$ is the manipulated objects, $x_i^{predict}$ are updated in each iteration loop to satisfy all the constraints $C_j(x_i, \ldots, x_n)$. In this way, the $x_i^{predict}$ will be converged into a optimised value after iterations.

Once the satisfied $x_i^{predict}$ is got, 16-19 instruction will update velocities by the difference of position and estimated position accordingly.

The external force in the 1-4 lines manipulate the velocity directly as the normal way, since the it can not be converted into a constraint. Apart from that, in end line 20, some correction for the velocity may be needed. This is like a compensation for the velocity directly which guarantee a more reasonable behaviour.

In the whole loop scheme, the constraints are fixed, depending on expected behaviour. In other words, this converting all the force calculation or physical equation into certain proper constraints if possible. Another advantage of this model is the whole simulation will become more stable, and it does not rely on the time step size, but rely on shape of constraints.

Considering all these, PBD is supposed to be more flexible, more stable and easier to implement.

2.5 Fluid Mathematic model

In this section, mathematic model for Navier-Stokes with SPH will be presented. In most of the fluid simulation, properties like density, pressure, tension force and incompressibility are achieved in different ways. This thesis strictly followed models in [24], which are specially designed to fit in the position-based approach.

The algorithm detail could be seen in the 2.6 section, according to the algorithm, there is no acceleration in the solver, only impulse which modify the velocity directly. Therefore, we calculate the elasticity force as an internal force density, which has the same meaning as acceleration.

All the added internal forces will be in a force density format, which could be directly applied on velocity. The force density is represented as force vector divided by the density of the particle:

$$a_i = \frac{dv_i}{dt} = \frac{F_i}{\rho(r_i)} \tag{2.16}$$

In which the density can be calculated with smooth kernel function. Note that density is one of the core quantities in fluid simulation, it will be updated and store separately in each time loop as follow before other updates:

$$\rho(r_i) = \sum_j m_j W(r_i - r_j, h) \tag{2.17}$$

Density and Incompressibility constraints

According to [24], the density relaxation is achieved by the density constraint after Forces has been applied. This constraint is aim to achieve incompressible behaviour, which is very important for the fluid simulation. They used a non-linear constraints From[25], which can be calculated as 2.18 for each particle.

$$\lambda_i = -\frac{C_i(p_i, \dots p_n)}{\sum_k \|\nabla_{p_k} C_i\|^2 + \varepsilon} \tag{2.18}$$

ε is a user defined parameter to soften the constraint, the mathematic detail is in the [26]. Numerator could be computed as:

$$C_i(p_i, \dots p_n) = \frac{\rho_i}{\rho_0} - 1 \tag{2.19}$$

in which the ρ_i is the density at position r_i, ρ_0 is the rest density for the particles, in this thesis, it is defined as a constant parameter. The denominator can be computed as:

$$\nabla_k C_i = \frac{1}{\rho_0} \begin{cases} \sum_j \nabla_k W(p_i - p_j, h), & \text{if} k = i \\ -\nabla_k W(p_i - p_j, h) & \text{if} k = j \end{cases} \tag{2.20}$$

These terms can be simply computed by using the gradient approximation as mentioned in section 2.3. The effect of this density constraint is to maintain the incompressibility of the water. By guaranteeing the density of certain area to be a constant value, tiny displacement will be applied to the position of the particles. After several iterations each loop, relatively plausible result could be expected. The advantage is that this is a intuitive way of describing water behaviour, which will be close to the physical phenomenon naturally.

Tensile pressure

Surface tension is phenomenon at the surface of fluid materials, it can be simply comprehended as greater tension force between fluid particle than force between fluid-gas particles cause surface curves. One can easily note that particle-based simulation could suffer from this artifact, since no gas particles are simulated. The vacuum outside of the fluid body does not maintain any force to balance the interface particle, where the particle will be applied large inwards force from the density constraint in last subsection.

To solve the tensile problem at the board of fluid body, which caused by asymmetric density distribution of particles, some researcher[27] use ghost particles wrap the fluid domain to solve it. Such solution is quite intuitive, adding invisible air particles around the boundary, but also increase the complexity of the approach. In contrast, the model in [24] use an artificial pressure constraint which inspired from [28] to solve this artifact:

$$s_{corr} = -k(\frac{W(p_i - p_j, h)}{W(\nabla q, h)})^n \tag{2.21}$$

in which the $|\nabla q| = 0.1h$, it's a point at some distance inside the smoothing kernel function. This constraint could be simply comprehended as a compensation force will be applied on particles with no particles appear within the smoothing kernel radius which are interface particles. Till now density and tensile pressure constraints will be applied directly to the position update by SGAF as:

$$\triangle x_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j + S_{corr}) \nabla W(r_{ij}, h) \tag{2.22}$$

Viscosity and Elasticity

Viscosity behaviour is for the velocity difference compensation. By measuring the difference of the velocities of a pair of particles, we could get the information of the direction and magnitude of the relative displacement.Thus magnitude of compensate force is simply proportional to velocity difference. Those falling apart particles will be pulled together, and getting close particle will be pushed away. In this thesis, it calculated just as state of art of particle fluid simulation in [3] using DGAF :

$$F_i^{Viscosity} = \eta \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla^2 w(r_{ij}, h) \tag{2.23}$$

In which the η is the coefficient of viscosity, in this thesis, it will be modelled as constant. Although the behaviour of η determine which type of non-Newtonian fluid it belongs, this implementation decide to separate elasticity and viscosity as two orthogonal force terms.

For the Elasticity force in this thesis, two models will be implemented separately. This section will explain the strain tensor approach, which has many mathematic equations involved. The another fixed spring model will be explained directly in implementation section, since it's simple to understand. This strain tensor force followed method in [20],:

$$F_i^{Elasticity} = \sum_j \frac{m_j}{\rho_j} \frac{\mu_j - \mu_i}{2} S_i^{elastic} \cdot \nabla w(r_{ij}, h) \tag{2.24}$$

Except $S_i^{elastic}$ term, other terms are easy to calculated in SPH. μ_i is the elasticity coefficient of the material. The main problem is the elastic strain tensor term $S_i^{elastic}$.

It is a measurement of deformation of material, normally it can be derived from the spatial derivative of the deformation function of the material. However, there is no such function, since the large deformation and randomly motion of this fluid model. Thus the strain rate is used to compute strain of the object, and the strain tensor at certain time t will be integrated as:

$$S^{Elastic} = S_0^{Elastic} + \int_0^t \dot{S}^{Elastic}\, dt \tag{2.25}$$

the initial tensor S_0 will be set to 0, and rate $\dot{S}^{Elastic}$ will be calculated from:

$$\dot{S}^{Elastic} = \dot{S}^{Total} - \dot{S}^{Plastic} = (\nabla v + (\nabla v)^T)/2 - \alpha \frac{S'}{\|S'\|} max(0, \|S'\| - \gamma) \tag{2.26}$$

where S' is:

$$S' = S^{Elastic} - \frac{Trace(S^{Elastic})}{3} I \tag{2.27}$$

∇v, the 3×3 Jacobian matrix, is:

$$\nabla v = \sum_j \frac{m_j}{\rho_j}(v_j - v_i) \otimes \nabla W(r, h) \tag{2.28}$$

$\otimes$ is the outer product, which is important for Jacobian Matrix. $\|S'\|$ is Frobenius normal. α is elastic decay rate normally set to 2μ. γ is yield point, which is supposed to determine the different behaviour of plasticity and elasticity. As many matrix related calculations are involved here, matrix algebra library is needed. The math library used in this simulation is discussed in implementation chapter.

Velocity correction

The last term that need to add is the Velocity correction after every time step. XSPH viscosity was used:

$$v_i = v_i + c \sum_j (v_i - v_j) \cdot W(r, h) \tag{2.29}$$

2.6 Smoothing kernel

The smoothing kernels used is the key factor in the SPH method, it may influence the speed, stability or even physical outcome of the simulation. Therefore, carefully choice of the smoothing kernel is a must. In the begging works of SPH, a Gaussian like functions are used more. It defined as:

$$W(\mathbf{r}, h) = \sigma e^{-q^2} \tag{2.30}$$

where $q = \frac{\|\mathbf{r}\|}{h}$ and σ varies in different dimensions. $\frac{1}{\pi^{\frac{1}{2}} h}$ for the 1D, $\frac{1}{\pi h^2}$ for the 2D, and $\frac{1}{\pi^{\frac{3}{2}} h^3}$ for the 3D. The different order of σ is to simply guarantee the integral of the kernel is equal to 1($\int W(r,h)\,dr = 1$).

However, the Gaussian kernel has some drawbacks for SPH computation. First is the exponentiation, which makes it computationally expensive for simulation. The another problem is the small contribution in the part of outside smoothing length h, which would let the computation happen for far neighbours.

Following widely used kernel functions as mentioned in [3], the Poly6 kernels is :

$$W_{poly6}(\mathbf{r}, h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - r^3)^3, & 0 \le r \le h \\ 0, & \text{otherwise} \end{cases} \tag{2.31}$$

related gradient and Laplacian are:

$$\nabla W_{poly6}(\mathbf{r}, h) = -\mathbf{r}\frac{945}{32\pi h^9}(h^2 - r^2)^2 \tag{2.32}$$

$$\nabla^2 W_{poly6}(\mathbf{r}, h) = -\mathbf{r}\frac{945}{8\pi h^9}(h^2 - r^2)(r^2 - \frac{3}{4}(h^2 - r^2)) \tag{2.33}$$

As one can notice, those functions are much cheaper than Gaussian ones, and in this thesis, these kernels are used for everything except pressure force and Viscosity Force. As mentioned in [3], pressure force approximation will be vanished when two particles get close, since the gradient kernel approaching zero at 0 value. To solve this, they introduced Spiky kernel as follow:

$$W_{spiky}(\mathbf{r}, h) = \begin{cases} \frac{15}{\pi h^6}(h-r)^3, & 0 \le r \le h \\ 0, & \text{otherwise} \end{cases} \tag{2.34}$$

its gradient:

$$\nabla W_{spiky}(\mathbf{r}, h) = -\mathbf{r}\frac{45}{\pi h^6 r}(h-r)^2 \tag{2.35}$$

The problem for viscosity force is the negative value of the Laplacian function, which physically means negative compensation will be apply for the particle. The total conservation of energy will be broken because of the force from nowhere. Thus, special kernel for viscosity is:

$$W_{viscosity}(\mathbf{r}, h) = \begin{cases} \frac{15}{2\pi h^3}(-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1), & 0 \le r \le h \\ 0, & \text{otherwise} \end{cases} \tag{2.36}$$

its Laplacian:

$$\nabla^2 W_{viscosity}(\mathbf{r}, h) = -\mathbf{r}\frac{45}{\pi h^5}(1 - \frac{r}{h}) \tag{2.37}$$

For better understanding, these kernels are plotted as 2.4.

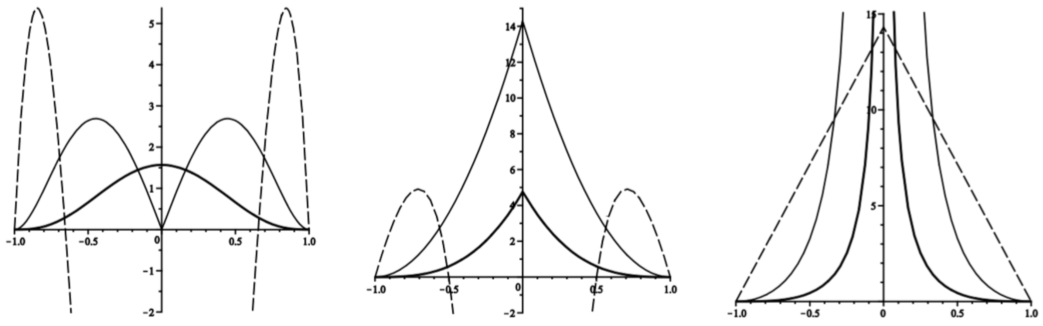


Figure 2.4: $W_{poly6}, W_{spiky}, W_{viscosity}$ kernels function (Thick) , gradient (Thin) and laplacian (Dash). [3]

2.7 Implementation algorithm

This section will present the basic algorithm in each integration loop. The whole algorithm is built upon the PBD fluid simulation described in [29].

Algorithm 2: Simulation loop

```
Result: Particle Position $x_i$
1  forall the Particle $i$ do
2      Apply External Force : $v_i \leftarrow v_i + \triangle t f_{external}$ ;
3      Predict Position: $x'_i \leftarrow x_i + \triangle t v_i$ ;
4      Confine particles in boundary ;
5  end
6  forall the Particle $i$ do
7      Clear Neighboring particles ;
8      Search neighbouring particles: $N_i(x'_i)$ ;
9  end
10 while $I <$ IterationNumber do
11     foreach particle $i$ do
12         Clear $\triangle x$ : $\triangle x \leftarrow 0$ ;
13         Particle-Particle Collision Handle : $\triangle x \leftarrow C(x'_i, N_i(x'_i))$ ;
14         Calculation Density: $\rho_i(x'_i, N_i(x'_i))$ ;
15         Apply Density Constraints: $\triangle x \leftarrow \triangle x + DensityC(x'_i, \rho_i, N_i(x'_i))$ ;
16         Apply Tensile artificial pressure: $\triangle x \leftarrow \triangle x + TensileC(x'_i, \rho_i, N_i(x'_i))$ ;
           // Apply fixed spring constraint:
               $\triangle x \leftarrow \triangle x + SpringDisplacementC(x'_i, \rho_i, N_i(x'_i))$ ;

17         Apply $\triangle x$: $x'_i \leftarrow x'_i + \triangle x$ ;
18     end
19 end
20 forall the particles $i$ do
21     Update Velocity: $v_i \leftarrow \frac{x'_i - x_i}{\triangle t}$ ;
22     Apply Internal Force: $v_i \leftarrow f_{Viscosity}(x'_i, v_i, N_i(x'_i)),\ f_{Elasticity}(x'_i, v_i, N_i(x'_i))$ ;
23     Calculated XSPH Velocity Correction ;
24     Update position $x_i \leftarrow x'_i$ ;
25 end
```

As one can notice, this simulation loop is similar to the general PBD model as described in 2.4. The core techniques are maintained, little modifications has been applied. Lines 6-9 is the process of neighbours searching, which is extremely important for the SPH Model. Every calculation that involved smoothing kernel will use the information of the neighbours. Thus before calculation constraints, we update the neighbours information for this loop and save it as a list of index. The detail about the neighbour searching is discussed in implementation chapter.

In the constraints iteration, mainly three constraints are computed here. Note the collision handle is treated as one kind of constraint. In this thesis, the collision is implemented very simple, since the limitation of the PBD approach itself, more accurate collision constraint is needed. The discussion of the collision can be found in 5.6 section. All the constraints are simply applied to predicted position x_i' as the general model. Note the comment spring constraint line, that is for fixed spring elasticity model. The detail of that is in algorithm 3 in implementation chapter.

After constraints calculation, the velocity is determined by the difference of the predicted and previous position. The important modification is the internal force computation in line 22. This is actually done by several CUDA kernel functions, detail of mathematic model is discussed in the previous chapter. These two terms will apply directly on the new velocity and update it. Theoretically, this velocity update is equal to applying the acceleration as Newton motion law. Apart from internal force, additional velocity correction is done for more accurate result.

Chapter 3

Visualisation

For all the fluid simulation, a plausible physical behaviour requires a realistic Rendering result. Visualisation probably become the crucial part to show a simulation work. It involves not only the the structure of data but also the better approximation of light calculation.

In the fluid visualisation, Marching Cube method is most popular one, which constructs polygon meshes from iso-surface. This method utilises density of the particles in 3D which is computed in GPU expensively. In Lagrangian Models, the surface normally determined by the smoothing kernel functions of particles. On the other hand, for Eulerian models, zero level set is used for the same purpose.

Mentioned by Mller in [30], a camera-independent way, called screen space Mesh, is introduced to achieve a water-like result. This method save the memory by only calculating the particles in the view space and constructing the surface in front of the viewer.

In this thesis, the rendering for simulation mainly followed [31], which is a practical rendering method inspired by [30]. In this GDC talk, Green use DirectX as the rendering API, but in this thesis, OpenGL will be used. Green rendered the particles directly as spherical sprites instead of generating meshes by Marching cube algorithm as [30], which is more easy to implement.

3.1 Rendering pass

In this implementation, the rendering pass is run as diagram 3.1

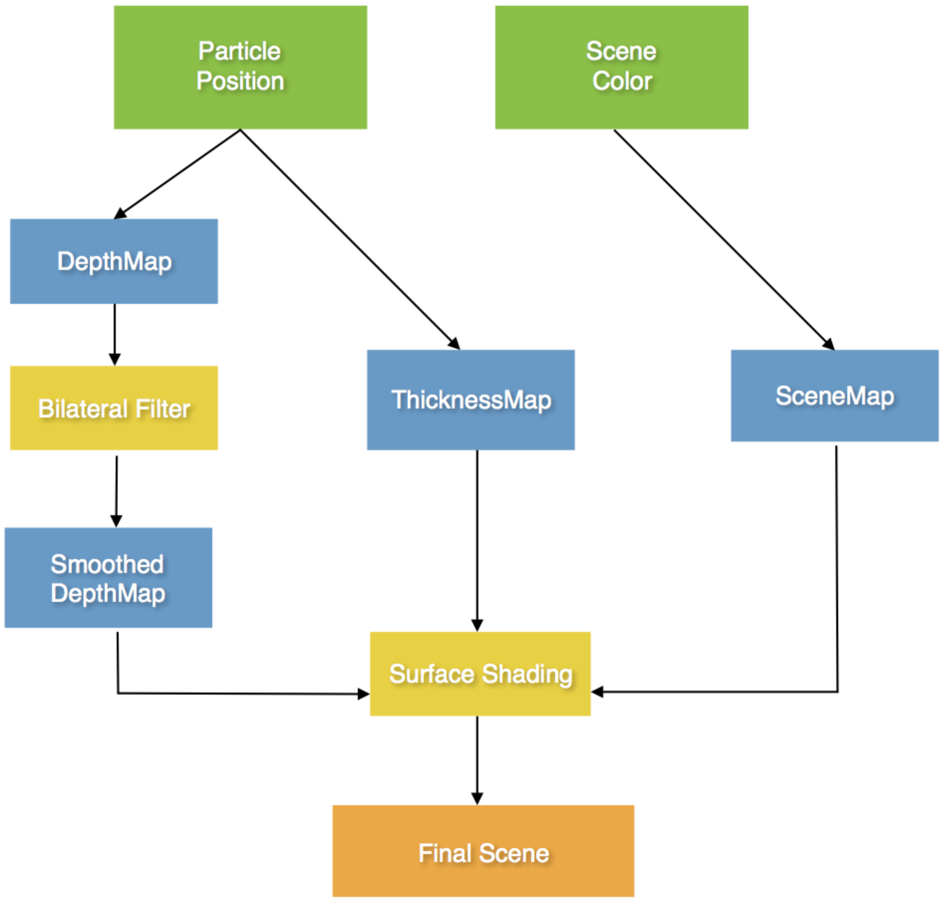


Figure 3.1: Rendering pass

All the calculation involved happen in the shader program, the only input from the physical simulation is the position value of the particles. The scene color pass is the environment color around the water domain. It could be the cube map in many applications, also it could be any reasonable color from the environment objects. These colors will be save into the scene map as 2D texture passing to the final water shader for blending. In this thesis it simply be treats as a floor with single color. In the following sections, every important process and method will be discussed in detail.

3.2 Basic knowledge

Render to texture

In this thesis, depth map and other color maps are important for the final rendering result. The technique help to get those maps is the render to texture. This a useful and simple technique.

To have a overview of this technique, one must understand the how the frame buffer work in the modern OpenGL, which is nothing but 2D arrays of pixels. In short, there are two kinds of framebuffer object(FBO) in the rendering pipeline. One is the default framebuffer that for displaying on the screen. Another is non-displayable framebuffer, which store the same information but without showing color on screen.

The second framebuffer object is used in this thesis, since it save the process of copying information from screen to texture and store all the needed information directly to texture.

Instead of using screen as the destination of informations, framebuffer-attachable images are used as the destination. There are two kinds that both are used in this thesis. One is the *Texture Image*, the outcome of the rendering will become a texture map in the end. Another is called *Renderbuffer Image*, in which OpenGL will perform *offscreen rendering*. In this thesis, we only use the *Texture Image* as the destination, since we will reuse the datas.

The famebuffer object contains color, depth and stencil information that could be used. The the way of using the application-created FBO is very similar to the default FBO, all the "create","bind" "draw" processes are the same, only difference is that result will be stored in the *Texture Image* which "generated" in the initialisation phase.

Coordinate transformation

Since the intensive use of the texture map (depth, thickness, scene maps), how the transform the coordinate of the texture into the view coordinate is the background knowledge.

First thing we have to note is that the texture map a normalised map range from [0-1] which all the value are stored on. In this thesis, the coordinate of each pixel in the 3D space can be accessed by transforming the coordinate from the texture map.

Another concept needed is the space transformation in the openGL API, the position in world space will be transformed into ***view coordinate***, then ***clip coordinate***, which is the 2D coordinate show on screen.

Thanks to the way we generate texture maps, which save all the information in the point of ***clip coordinate***. Thus, there is obvious connection between the texture coordinate and the ***clip coordinate*** (screen coordinate). The only thing we need to do is define the function that transform pixel in texture coordinate back to ***view coordinate***, where we could calculate the phong shading light model.

The core function is showed in following code:

```
        vec3 uvToEye(vec2 p, float z) {
        vec2 pos = p * 2.0f - 1.0f;
        vec4 clipPos = vec4(pos, z, 1.0f);
        vec4 viewPos = inverse(projection) * clipPos;
        return viewPos.xyz / viewPos.w;
}
```

This function has two inputs, texture coordinate as *vec2*, and it's depth value as *float*. By using this, function returns the 3D position *vec4* in ***view coordinate***. The first line is to transform range [0-1] into range [-1,1]. The normalised clip coordinate can be easily got in line 2. The reason we it can be done like this is that depth value is actually the z value looking from the screen point of view. Once get the 3D position in ***clip coordinate***, we multiply the position with inverse matrix of projection transforming matrix. Finally, the position in the ***view coordinate*** can be get by dividing the position with w value, which give the normalise position.

3.3 Depth Map

In screen space method, the core technique is to build the depth map from the particles position in shader.

To do this, we use particular *depth_shader* to render depth map and save it into texture by frame buffer. In the vertex shader, positions are treated as usual, but pass the position values in view coordinate to fragment shader. In fragment shader, we calculate the depth value and change it into clip space. Since we assume the particle as a perfect sphere, simple geometry calculation is used to get every fragment position on the sphere boundary. In short, the pixel position interpolated in fragment shader will be got automatically, which provide the $p(x, y)$ value, after that, Pythagorean theorem will give the z value if we assume radius of sphere is 1.

Once pixel Positions are computed, all the position will be transferred into clip coordinate from view space. Finally, all the Z values needs to be rescaled in range of $0-1$ and assigned into ***gl_FragDepth***.

3.4 Smooth the Depth map

Once the depth map got, next step is to apply some filter on the map for smoothing the surface shape of the water. Otherwise, the artifact of individual sphere shape would be too obvious in the final result.

Silhouette of the water domain is supposed to be saved, but noise is supposed to be reduced. In other words, strong edge feature saved, but small features averaged. To achieve this effect, we applied Non-linear bilateral filter to the depth map respectively in X and Y direction.

Bilateral filter is a edge-preserving noise-reducing technique in image processing, some image processing software call it ***Gaussian Blur***. In short, this filter replaces the intensity of each pixel by average intensity difference values of its neighbouring pixels. The normalised filtered intensity can be calculated as:

$$I_D(i,j) = \frac{\sum_{k,j} I(k,l) * w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)} \tag{3.1}$$

In which the $w(i, j, k, l)$ can be computed as:

$$w(i,j,k,l) = e^{-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}} + e^{-\frac{\|I(i,j)-I(k,l)\|^2}{2\sigma_r^2}} \tag{3.2}$$

In this thesis, I_D is the depth value, and it only considers 6 neighbours separately in either X or Y axises. Also terms in $w(i, j, k, l)$ will be simplified as difference of the depth value and neighbouring distance, both of them are manipulated in texture coordinate.

In the Shader, most of the work is done in fragment shader. A simple loop will be used for every pixel to look 6 nearest neighbours of it. Finally the smoothed depth value will be stored into depth map again. To save computation resources, we chose to do this in x and y axises separately, although it's not orthogonal perfectly.

3.5 Thickness map

To make this simple rendering more realistic,it's better not simply apply transparent color to the water domain evenly. Since the light will be absorbed by the water, namely the thickness of the water will influence the color (or how much we could look trough it), thickness map will be calculated in this implementation.

The thickness value could be easily get by rendering with ***glBlendFunc*** and ***glBlendEquation*** and render it into thickness texture. The source and destination color factors are both chosen ***GL_ONE***, blend equation we simply use ***GL_FUNC_ADD***
.

3.6 Water Shader

Once uploaded the depth map and thickness map as textures into the final shader, we are ready to combine them together to create water surface. At this stage, the rendering approach is like what we render a water or ocean surface on flat polygons. The only difference is to extract the correct position and normal value from 2D texture coordinates.

In the Final fragment shader, the normal of the water will be calculated. Just as the method one use in greyscale map, depth value at each pixel and its four neighbours' will be use. The change rate of the adjacent depth value $\triangle Z$ can be treated as hight gradient of the water. Once $\triangle Z_x$ and $\triangle Z_y$ are got, *cross* function will give the correct normal direction. Note that, all the calculation are happen in View coordinate, thus a *coordinateTransfer* function is supposed to be pre-defined for convenience .

After normal vector is calculated out, the pixels color will be determined as phong shading light model including Ambient, diffuse, and specular light contributions.

The following subsections are going to talk about the reflection and refraction model which makes the water more realistic.

Fresnel

The the fresnel effect require a model to keep balance between reflection colour and refraction color. Therefore we use the equation 3.3

$$C_{water} = (1 - R)C_{refraction} + RC_{reflection} \tag{3.3}$$

Where C_{Water} is the color of the water. R is the reflection coefficient . Follow Schlick's approximation, it can be approximated as:

$$R = R_0 + (1 - R_0)(1 - cos(\theta))^5 \tag{3.4}$$

R_0 is the reflectance of light coming with normal incidence, and $con(\theta)$ can be get as:

$$cos(\theta) = dot(N, v) \tag{3.5}$$

reflection

As any kind of water shading, reflection color could be divided into 3 major contributions, namely environment, light source, and self reflect. The model is:

$$C_{reflection} = C_{environment} + C_{lightSource} + C_{LocalReflect} \tag{3.6}$$

Environment colour could be get from the cube map, however for simplification in this thesis, we threat it simply as sky color (blue). As for the light source color, we model it simply as parallel sunlight. Thus, Sunlight reflection color could be calculated as phong shading model as:

$$C_{lightSource} = C_{SunLight}(\hat{R}_{reflection} \cdot \hat{V}_{ToView})^{shininess} \tag{3.7}$$

In which the $\hat{R}_{reflection}$ is the direction vector as a reflection of light on certain surface. $\hat{V}_{ToView}$ is the direction vector towards to viewer. As for the local reflection happening in the object itself, it normally requires Ray - Tracing method in fluid simulation and cost lot more computation. In our thesis, we did not include this part as contribution for simplification reason.

Refraction

To achieve refraction effect of the water, we converted it into a transparent task. For better visualisation, thickness of the particles is considered.

First, we computed the thickness value and save it as a 2D texture in screen space. Followed **Beer Lambert law**, we attenuate the color of the water depending on the thickness value. By doing this, those droplets or thin parts of the fluid domain will show much lighter color. Blending this water color with scene color will achieve a more realistic half opaque water surface. Since visualisation is not the core of this thesis, we used this way to approximate refraction effect.

Chapter 4

Implementation

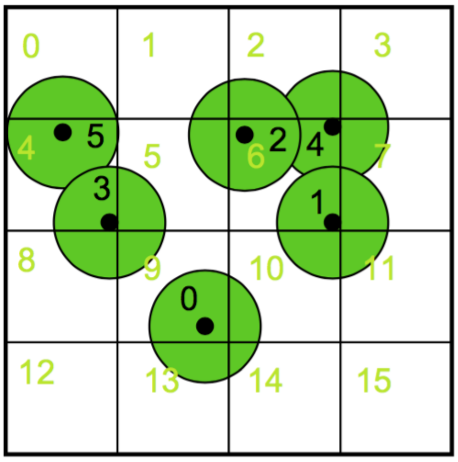
4.1 GPU and CUDA framework

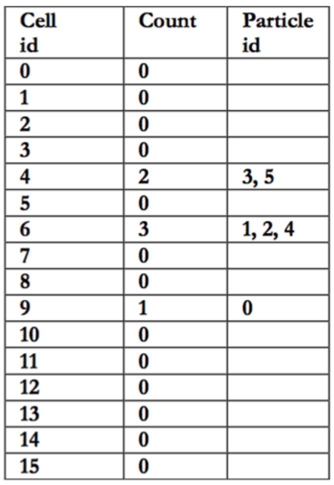
GPU is designed for large parallel computation since its special hardware structure. CUDA is Nvida's Application programming interface for general parallel computation directly on GPU. As the parallel nature of SPH method for all the particles, utilising GPU could help physical simulation running in a fast way. However CUDA requires a relatively careful design for the program flow, different way of access memory could result in divergent performances.

As mentioned in particle CUDA sample in [4], there are two ways for the grid problem. One is the atomic operation for building the grid that particle belongs to. Another is the Sorting method for the grid construction.

In this thesis, Atomic operation method is used for the grid. This method is relatively simple one that in every update time step, there is a kernel function called ***UpdateGrid***. In the ***UpdateGrid***, one thread per particle is run and it calculate which grid cell this particle belongs to, and increase the cell counter number by using the ***atomicAdd*** function simultaneously. The 2D diagram for Atomic grid is provided as 4.1.

There is a potential problem of this method, if too many particles are concentrated in the same grid, it gonna transfer this method into a serial method(particle will be handle one after another). To avoid this situation, the maximal number of particles per grid is limited.





Cell id	Count	Particle id
0	0	
1	0	
2	0	
3	0	
4	2	3, 5
5	0	
6	3	1, 2, 4
7	0	
8	0	
9	1	0
10	0	
11	0	
12	0	
13	0	
14	0	
15	0	

Figure 4.1: 2D Grid Construction Atomics [4]

4.2 Neighbour Search

As mentioned in the previous section, 3D grid with the side length, which is the same as smoothing length, will be reconstructed depending on references of that particles attached.

Such format of the particles will help to reduce neighbour search cost. For certain particle, its neighbours could only appear on the its adjacent 3 grids, which contains particles, in one of the three axises. Thus in 3D, the maximal number of the adjacent cells is 26($3 \times 3 \times 3 - 1$). In the complexity point of view, this format reduce the summation from $O(n^2)$ to $O(nm)$, m is the maximal number of particles in each cell. The diagram for 1D as follow 4.2

4.3 Data Structure

As mentioned in above section, program flow should be specially designed for using the GPU computation, so as the data structure. Since the data structure will influence the memory allocation directly, modified structure is better than simple arrays.

Generally speaking, there are two memory layouts. One is use an array of particle

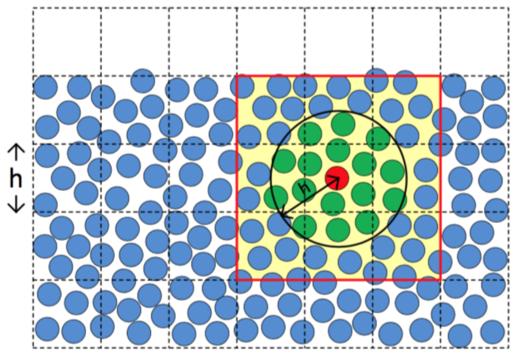


Figure 4.2: 1D Neighbour search in grid form [5]

structure, in which necessary properties are saved. Another one is separate but parallel array of properties for all the particles. Since every time step, properties of single particle are needed, it's better to pack the quantities particle by particle.

Concerning the properties of position based simulation, at least position, velocity and density are needed to be stored. Among which density is a scale, velocity is a first-order tensor. Thus we use float and vector(float3) to store them, and all of them are saved as a dynamic array *std::vector*.

Specifically, the positions are actually allocated in an OpenGL vertex array object (VBO) so that they can be rendered from directly.

Lastly , to calculate $f_{Elastic}$ terms, additional *3* $\times$ *3 matrix* strain tensor(second-order tensor) needs to be stored for each time step. Therefore, the structure of each particle is extended.

The whole fluid are treated as a continuous material instead of focus the behaviour of individual particles, thus we describe each particle with following quantities in structure table 4.1

4.4 Math Library

For all the particles calculation, vectors and even matrices will be appear nearly every step. How to use an efficient linear math library for computation will give a different result. Also for CUDA, which is a parallel structure processed, there are not too many options out there.

Symbols	Quantities(unit)	Data type
$x^{predict}(t)$	predicted position (m)	float3
$v(x,t)$	velocity (m/s)	float3
$\rho(x,t)$	density (kg/m^3)	float
$p(x,t)$	pressure (N/m^2)	float
$S(x,t)$	stress tensor (N/m^2)	mat3
$f_{internal}(x,t)$	internal force density (N/m^3)	float3

Table 4.1: Table of quantities per particle

In the beginning, there are two popular build-in linear math library which provide basic but enough linear functions. They are cuBLAS and cuSPARSE, which are GPU-acceleration version of BLAS and SPARSE Matrix library.

After testing several functions in these, it's easy to found that their functions are more suit for large mount of algebra calculation. However, the matrix related calculations in this thesis are only multiplication, transpose and normalisation, they are very basic functions. Also, the store and access method in those libraries are too complicated, which is not suitable for data structure of this simulation.

Considering these, we turned to another popular and powerful library GLM, the only problem is that we need to replace all the ***float3*** vectors that already used in this thesis, ***float3*** and ***glm::vec3*** are not compatible.

So in the end, I decided to extend the ***math helper.h*** CUDA math library, which only contain some vector (***float3***) functions. The comparison of performance on GPU between GLM and My extented library will be showed in the conclusion chapter later.

4.5 $F_{elastic}$ term

Strain tensor

Inspired by the [20] and [21], in this simulation, elastic force will be calculated as it mentioned in section 2.4. It will be applied as an internal force after computing external force.

Spring represent

Apart from that, inspired by another Position-based viscoelastic fluid simulation [17], in which they use spring model to represent elastic force, and by adjusting spring rest length to represent plastic behaviour. Thus, in this thesis, spring interpolated between particles with fixed rest length is used. This force will be applied directly on the $\triangle p$(change of position) after calculated external force.

Specifically speaking, one more spring constraint will be added as a way to control the particles. For every particle, we simply add a spring with fixed spring length L between them and calculated the displacement for it. The displacement magnitude of the spring force is proportional to the difference of the spring length L and particle distance r. Further more, we scale the magnitude with ratio of the current spring length to initial spring length. The detail is in the Algorithm 3

Algorithm 3: Spring Elasticity constraint

```
  Result: Spring Elasticity force
1 After Neighbour searching... ;
2 forall the particles i do
3     foreach particles j in the Neighbours do
4         $D \leftarrow \triangle t^2 K^{spring}(1 - L_{ij}/L_{initial})(L_{ij} - r_{ij})\hat{r}_{ij}$ ;
5         $\triangle x_i \leftarrow \triangle x_i - \frac{D}{2}$ ;
6         $\triangle x_j \leftarrow \triangle x_j + \frac{D}{2}$;
7     end
8 end
```

Chapter 5

Conclusion

5.1 Result

This viscoelastic Fluid simulation is run on Device Nvidia Quadro K2000. With 10000 particles and all the forces applied, the simulation run at around 7.4 FPS (with sprite rendering). At the same time without rendering, it runs around 8 FPS. The performance result could be see in graphic 5.1

As one can easily notice, the FPS go down quickly when the number of particle is go beyond 2k on this device but become stable around 8 FPS. If the simulation number goes upper than 18K, simulation will become unstable. Another thing need to point out here is that it turns out rendering doesn't take too much computation time down. The visualisation result is showing as screenshot 5.2 and 5.3

The scene is simulated as a cubic of viscoelastic material in a small invisible cubic container with a whole in the center. As the result of gravity, the material will be drag down to the lower level, which is little bigger invisible cubic container. The wall could be moved back and force to see the reaction of the materials.

As one may notice that this result is not behaving like an object with elasticity, but viscosity. It more or less shows the artifacts in the simulation, especially when the material interacts with the boundaries. Actually, no matter spring elastic force or strain tensor elastic force, both of them don't apply obvious elasticity to the material in the end. The possible reasons will be discussed in the next sections.

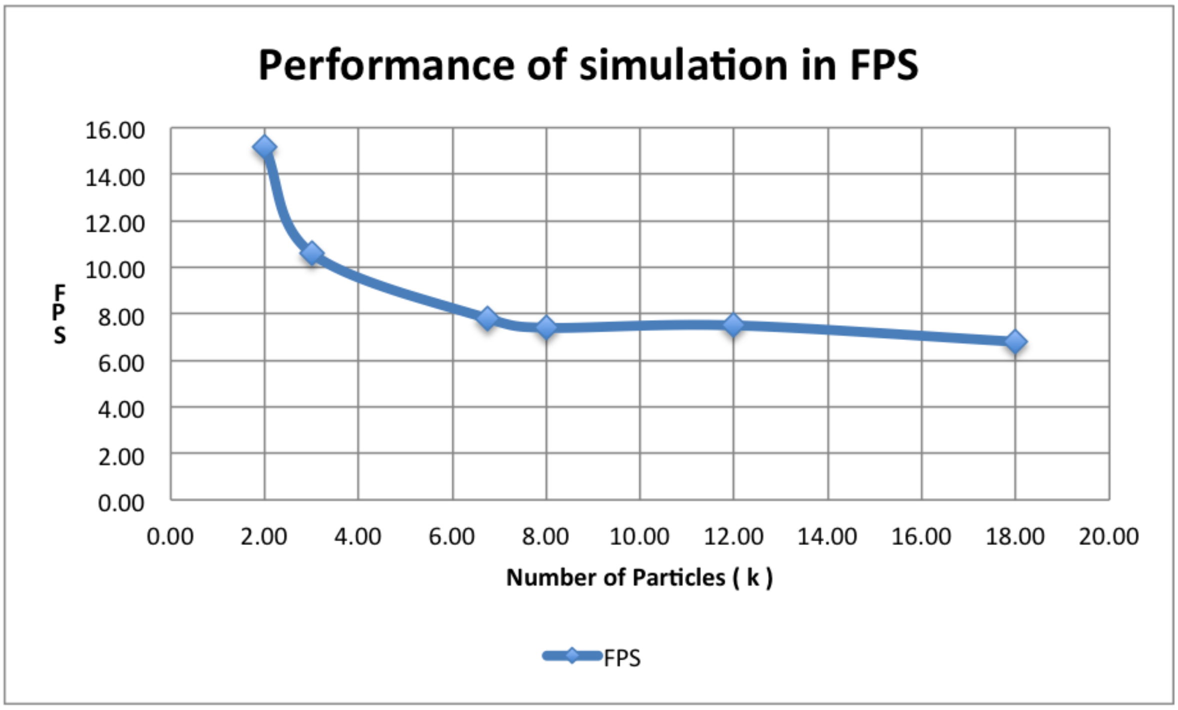


Figure 5.1: Performance of simulation on Nvidia Quadro K2000

5.2 Spring Elasticity Problem

As one can notice in the digram 5.4, the spring force did has influence on the material, however it looks like all the internal force give the same direction force which push all the material to the conner of the cube, which is not correct. This approach has been discarded in the early stage of my implementation.

In the thesis, the spring force is simply applied on each particle determined by all the neighbours it has, however, this may not consider the symmetrical problem in the beginning. The original method in the paper [17] adjust the spring length in every time loop, in some cases, the spring will be removed from the data array. The original method requires the data structure with good research and insert/remove performance which is not a good fit in this implementation. By adjusting spring length, the previous spring length memory is needed for every potential neighbour, which means every particle is supposed to have a memory space for saving all the other particle's spring. They whole advection loop will be added $O(n^2)$ complexity in the end.

Another thing is that, even without gravity, the whole material still get this accu-

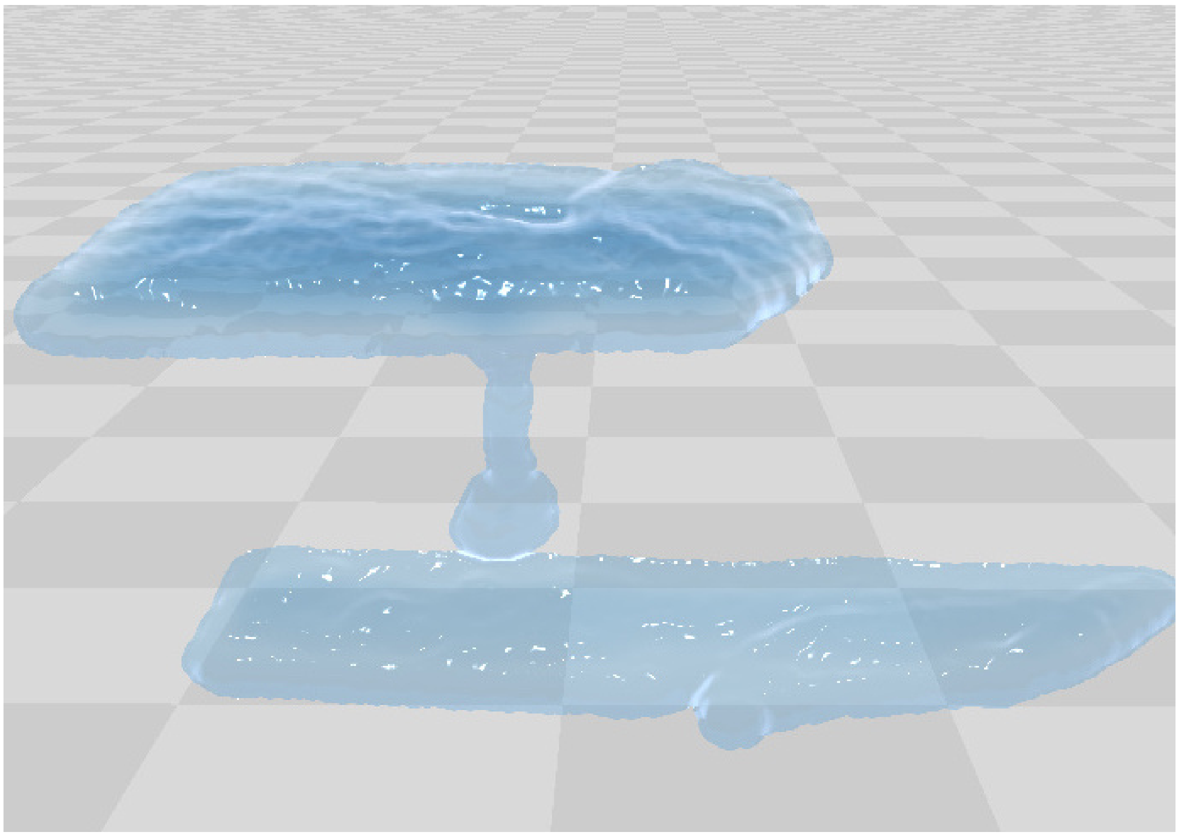
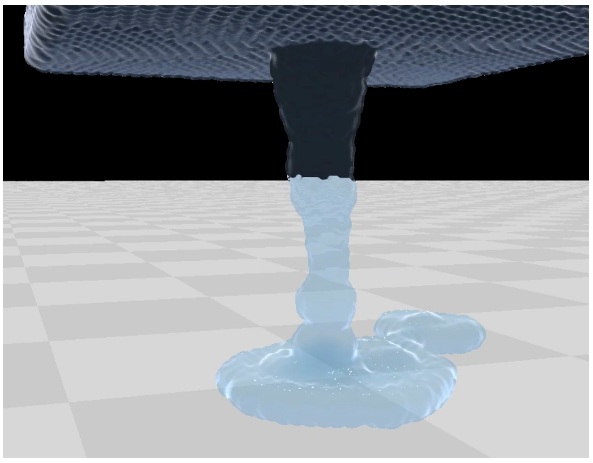
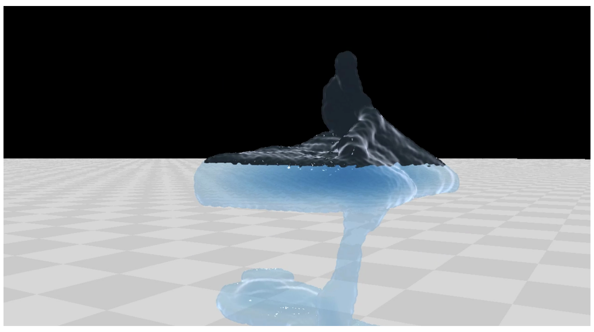


Figure 5.2: Simulation visualisation result



(a)



(b) Reaction of the moving wall

Figure 5.3: Simulation visualisation results

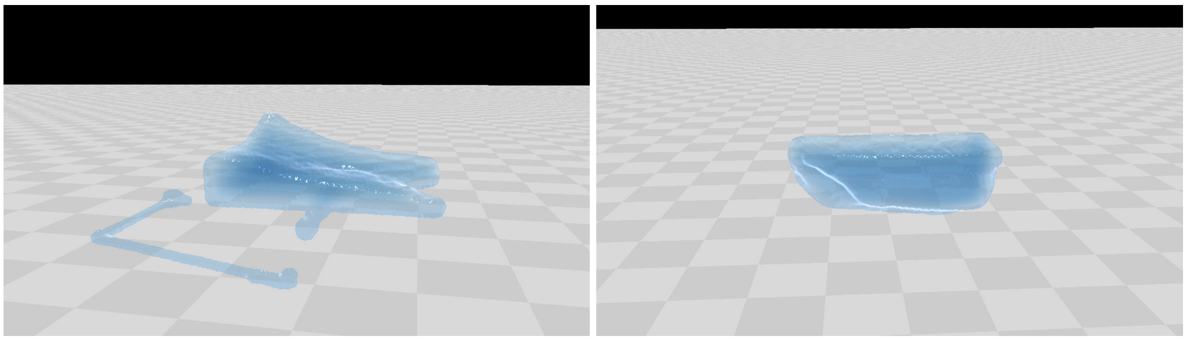

(a) With Gravity (b) Without Gravity

Figure 5.4: Simulation with fixed length spring results

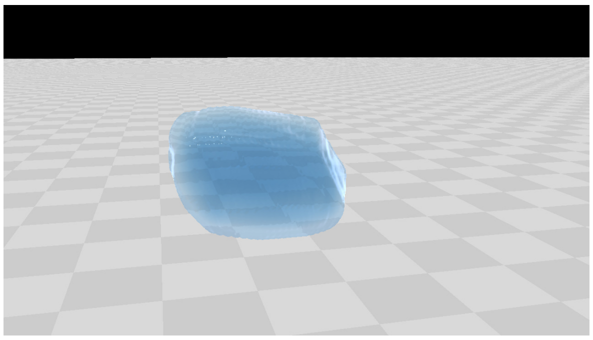
mulated force anyhow. However in theory the elasticity is reaction to the deformation (external force), it is not supposed to happen without force applied. It happened in this thesis, the reason could be the fixed rest spring length, which is half of the smoothing length(the neighbour searching length). Note that the major factor of the position based simulation is the density, and all the quantities is depend on the neighbours (SPH nature). Thus the particle's distribution could be all exceed the fixed spring rest length all the time. So such behaviour happened.

This spring representation idea is similar as that for cloth simulation, it seems practical. However the big difference is that particles' positions in cloth are relatively fixed, means the springs are actually part of the structure of the cloth, they could be easily found, they are attached with particles. The fluid model is never fixed, neighbours changes all the time, which means the spring has to be updated all the time, if there is no specially designed searching structure as [17], it could not be simply used for elasticity here.

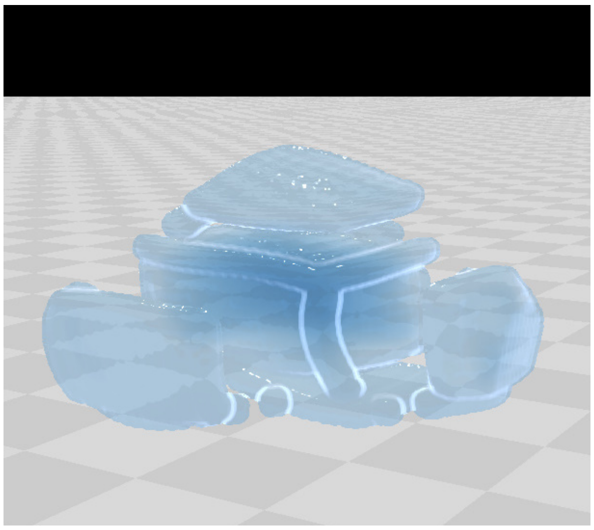
To sum up, the drawback of using spring elasticity representation is that with adjustable spring, the data structure does not fit, with fixed spring, the model is not accurate enough for the elasticity behaviour.

5.3 $F_{Elastic}$ Problem

Once when turn to use the strain tensor elastic force, the outcome is much stable although it's not perfectly plausible as expected. No matter what combination of



(a) Viscosity $\eta = 700$, Elasticity $\mu = 10^4$



(b) Viscosity $\eta = 700$, Elasticity $\mu = 10^6$

Figure 5.5: Elasticity effect without Gravity

the elasticity coefficient and other parameters, this simulation doesn't show too much elastic reaction. The outcome is not like that in the [20] we followed.

In expectation, the cubic material is supposed to keep the rough shape when it collide with the solid, when the yield stress is set to high(means the toleration of deformation is high). But it's hard to notice the obvious inner force that hold the shape. It just runs like honey. Thus, there is no bouncing behaviour in simulation.

No obvious bounce dose not mean the Elasticity force has not been correctly implement, it has the effect. Since the PBD is so sensitive to the external force, we zero out the gravity. The comparison shows the effect 5.5. When the elasticity is increased, the faces of the cube fall apart orderly outwards center, this because the tension force on the border particles make the deformation all the time. Theoretically, elasticity is a force to react deformation, higher elasticity tolerate larger deformation but push material back into initial shape. The orderly falling apart proof the correct effect of elasticity, but since no internal force to hold the shape in this model, it can not make a reasonable physical result as expected.

The reason could be complex. First is the collision response, for achieving better fluid behaviour, the original fluid simulation use extremely simple collision response, simply set position to the boundary and velocity to zero. However, this is fine with the position based water simulation, since density and artificial pressure will push particles against each, those stopped boundary particles could be pushed by the tiny force for compensating density. Thus this delicate balance between particles give a pretty plausible water feeling, it's incompressible, but so weak for new elastic force. In short, there is no practical collision response at boundary, what's worse is that it's not easy to achieve a proper response in PBD.

Unlike shape matching (one Position-Based approach for deformable object) , there is no such a strong inner force between particles which will help to keep the object's shape, which means no force will be transmitted from the deformed part to the opposite side of material. Applying Elasticity contrast the idea behind PBD fluid simulation, in which particle motion highly depends on the density of the whole material, random distribution of the particle is used to achieve better water-like behaviour. In other words, the nature of PBD fluid makes material deformed at every time step, which hugely damage the effect of the Elasticity force. This simulation shows that simply apply elasticity force to each particle can not achieve bouncing reaction.

5.4 Math Library Comparison

As mentioned in previous chapter, extended CUDA linear library is used in this simulation. The computation comparison with ***GLM*** is given in 5.6.

In this comparison, Matrix-Matrix and Matrix-Vector Multiplication has been tested. In general, ***GLM*** time consumption goes up linearly along with the elements handled increase. In the contrast, the ***Helper Math*** has a exponentially better performance in calculation.

5.5 Limitation

So far, We could draw the conclusion the applying of the spring and strain elasticity are not a suitable extension for this position based fluid simulation. As the original paper [24] said itself, the core controller of particles, namely artificial pressure depend on the simulation spatial resolution and time-step. Those parameters can not be easily adjust independently.

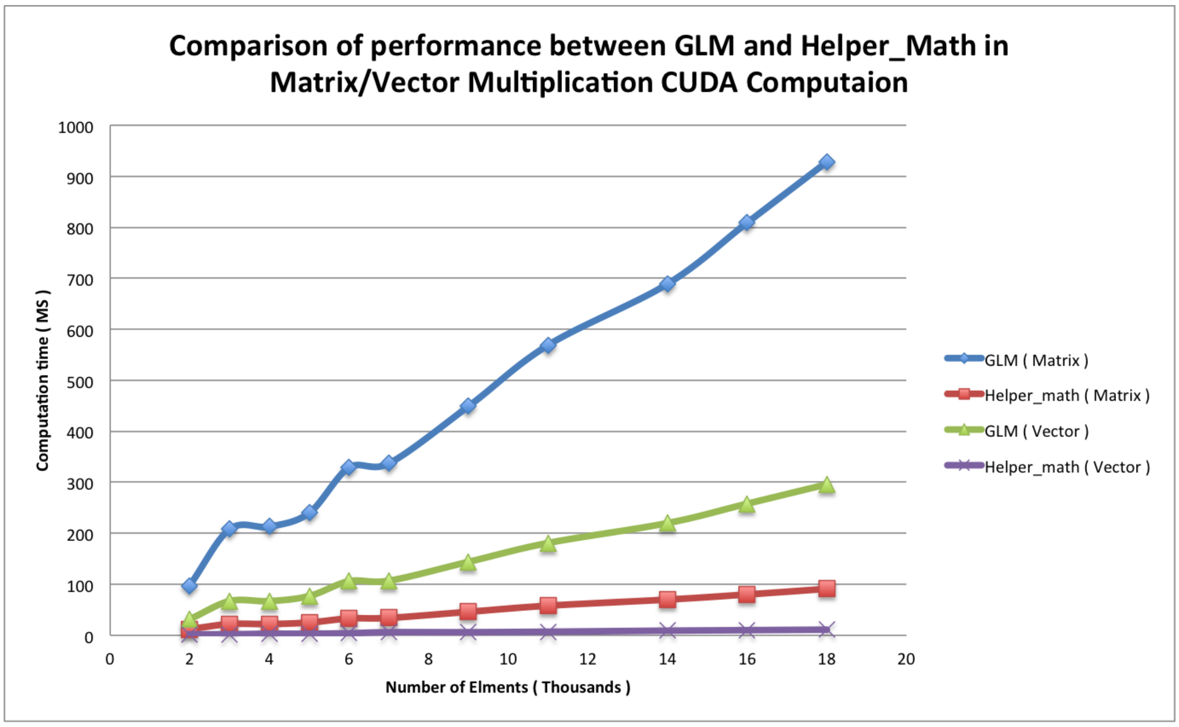


Figure 5.6: Comparison of Math library on CUDA

Apart from the elasticity force term, the position based model itself is designed as a direct way of control particles, and reduce the twice Euler integration numerical error. This model needs carefully and mathematically designed constraints to achieve certain behaviour, like the shape matching (SM). As mentioned by Miles Macklin and Matthias Mller(The major researchers for Position-based dynamics) in [32], they provide a unified particle representation for solid, fluid and deformable object based on position-based dynamic. These solutions share common idea of constraints, which treats material as a whole object. In their general solution, it's easy to transform from one phase into another by changing constraints, also the interaction among objects are easy to handle as well. Simply inserting an external term like this thesis is not a good way to go. In short, it's not easy to achieve a plausible behaviour without constraints for the position-based model.

5.6 Future Work

Collision

As the collision Response is the major cause of deformation, proper collision response will help the simulation to achieve better result, especially for the PBD model. As mentioned in[32], the collision response in PBD can not simply be implemented as other physical simulation. Since nature of PBD, the impulse of the collision will be zero out in terms of the effect. The possible way is to use a pre-stabilization which calculated both for predicted and previous position and accumulated the effects.

Constraints

As discussed in previous sections, Constraint is a much better way to control the Position-based particles. [32] use a multiple shape matching constraints to achieve large deformation. Thus, a proper constraint for handling the relative displace of particles could help the Elastic force term. Or in another prospect, a constraint for force transmitting instead of density is required for this simulation.

Grid Construction

Considering about the grid construction, it's possible to have a better computation performance as mentioned in [4]. Instead of using the atomic grid construction, sorting is even better for the grid calculation. Basically, by utilising the hash value for each particle cell and storing it as a list. Finally by sorting the list, it's ready to be used for certain purpose.

Rendering

Since the simplification of the rendering in this thesis, a lot more improvement could be made in the future. One thing is the shadow of the water, as one may notice there is no shadow but light. Any scene will become more realistic with the shadow applied. Since the sprites of the particles, it's easy to extend with shadow map. All it needs is to calculate the shadow map and render it in the final shader. The technique is render the whole scene again but from the point of light view, through which the water colour shadow will be calculated and applied in the end. Another effect that could improve the whole scene is the Caustic effect. Caustic effect happen when the light go through translucent material and provide special light patterns, which could be modelled as [33].

Appendix

The demo video is at https://youtu.be/X-htZ1UCS8k
The source code of simulation and math library test is at
https://github.com/YuanqiH/PositionBasedViscoealsticFluid

Bibliography

[1] "Defining viscosity." http://www.viscopedia.com/basics/defining-viscosity/.

[2] A. Paiva, F. Petronetto, T. Lewiner, and G. Tavares, "Particle-based viscoplastic fluid/solid simulation," *Computer-Aided Design*, vol. 41, no. 4, pp. 306–314, 2009.

[3] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, (Aire-la-Ville, Switzerland, Switzerland), pp. 154–159, Eurographics Association, 2003.

[4] S. Green, "Cuda particles," *nVidia Whitepaper*, vol. 2, no. 3.2, p. 1, 2008.

[5] J. Bender, K. Erleben, M. Teschner, *et al.*, "Boundary handling and adaptive time-stepping for pcisph," in *Workshop on virtual reality interaction and physical simulation VRIPHYS*, 2010.

[6] E. Mitsoulis, "Flows of viscoplastic materials: models and computations," *Rheology reviews*, vol. 2007, pp. 135–178, 2007.

[7] T. Lundgren, "Model equation for nonhomogeneous turbulence," *Physics of Fluids A*, vol. 12", no. 3, pp. 485–497, 1969.

[8] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars," *Monthly notices of the royal astronomical society*, vol. 181, no. 3, pp. 375–389, 1977.

[9] J. F. O'brien and J. K. Hodgins, "Graphical modeling and animation of brittle fracture," in *Proceedings of the 26th annual conference on Computer graphics and*

interactive techniques, pp. 137–146, ACM Press/Addison-Wesley Publishing Co., 1999.

[10] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, "Elastically deformable models," in *ACM Siggraph Computer Graphics*, vol. 21, pp. 205–214, ACM, 1987.

[11] J. Teran, S. Blemker, V. Hing, and R. Fedkiw, "Finite volume methods for the simulation of skeletal muscle," in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 68–74, Eurographics Association, 2003.

[12] D. L. James and D. K. Pai, "Artdefo: accurate real time deformable objects," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 65–72, ACM Press/Addison-Wesley Publishing Co., 1999.

[13] M. Desbrun, P. Schröder, and A. Barr, "Interactive animation of structured deformable objects," in *Graphics Interface*, vol. 99, p. 10, 1999.

[14] M. Teschner, B. Heidelberger, M. Muller, and M. Gross, "A versatile and robust model for geometrically complex deformable solids," in *Computer Graphics International, 2004. Proceedings*, pp. 312–319, IEEE, 2004.

[15] J. Bender, M. Müller, M. A. Otaduy, and M. Teschner, "Position-based methods for the simulation of solid objects in computer graphics," Eurographics, 2013.

[16] M. Müller, B. Heidelberger, M. Teschner, and M. Gross, "Meshless deformations based on shape matching," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 471–478, 2005.

[17] S. Clavet, P. Beaudoin, and P. Poulin, "Particle-based viscoelastic fluid simulation," in *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 219–228, ACM, 2005.

[18] A. Paiva, F. Petronetto, T. Lewiner, and G. Tavares, "Particle-based non-newtonian fluid animation for melting objects," in *2006 19th Brazilian Symposium on Computer Graphics and Image Processing*, pp. 78–85, IEEE, 2006.

[19] L. F. de Souza Andrade, M. Sandim, F. Petronetto, P. Pagliosa, and A. Paiva, "Particle-based fluids for viscous jet buckling," *Computers &amp; Graphics*, vol. 52, pp. 106–115, 2015.

[20] T. G. Goktekin, A. W. Bargteil, and J. F. O'Brien, "A method for animating viscoelastic fluids," *ACM Trans. Graph.*, vol. 23, pp. 463–468, Aug. 2004.

[21] Y. Chang, K. Bao, Y. Liu, J. Zhu, and E. Wu, "A particle-based method for viscoelastic fluids animation," in *Proceedings of the 16th ACM symposium on virtual reality software and technology*, pp. 111–117, ACM, 2009.

[22] L. Lucy, "A numerical approach to the testing of the fission hypothesis," *The Astronomical Journal*, vol. 82, pp. 1013–1024, 1977.

[23] F. Colin, R. Egli, and F. Y. Lin, "Computing a null divergence velocity field using smoothed particle hydrodynamics," *Journal of Computational Physics*, vol. 217, no. 2, pp. 680–692, 2006.

[24] M. Macklin and M. Müller, "Position based fluids," *ACM Trans. Graph.*, vol. 32, pp. 104:1–104:12, July 2013.

[25] R. Smith, "Open dynamics engine v0. 5 user guide, 2004," *URL http://www. ode. org/ode-docs. html*, 2008.

[26] K. Bodin, C. Lacoursiere, and M. Servin, "Constraint fluids," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, pp. 516–526, March 2012.

[27] H. Schechter and R. Bridson, "Ghost sph for animating water," *ACM Trans. Graph.*, vol. 31, pp. 61:1–61:8, July 2012.

[28] J. Monaghan, "Sph without a tensile instability," *Journal of Computational Physics*, vol. 159, no. 2, pp. 290–311, 2000.

[29] M. Macklin and M. Müller, "Position based fluids," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, p. 104, 2013.

[30] M. Müller, S. Schirm, and S. Duthaler, "Screen space meshes," in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 9–15, Eurographics Association, 2007.

[31] S. Green, “Screen space fluid rendering for games,” in *Proceedings for the Game Developers Conference*, 2010.

[32] M. Macklin, M. Müller, N. Chentanez, and T.-Y. Kim, “Unified particle physics for real-time applications,” *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 153, 2014.

[33] C. Wyman and S. Davis, “Interactive image-space techniques for approximating caustics,” in *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pp. 153–160, ACM, 2006.