# Fluid Simulation using Smoothed Particle Hydrodynamics

Burak Ertekin
i7637445

MSc Computer Animation and Visual Effects


August, 2015

# Contents

# List of Figures

# Abstract

Smoothed Particle Hydrodynamics (SPH) is one of the most popular approaches to fluid simulations. This paper explains the implementation of this method and the theory behind this method. The implementation is designed to simulate both Newtonian and Non-Newtonian fluids using Navier-Stokes equations. Different tests were made on various scenarios and results were analysed. Possible future works were suggested to improve these results.

**Keywords:** Fluid, Non-Newtonian, SPH, Smoothed Particle Hydrodynamics, Navier-Stokes

# Chapter 1

# Introduction

Realistic fluid animations are one of the most important phenomena in computer graphics. They are frequently used in computer gaming and visual effects industries. This high demand makes it a popular topic between physically-based simulations. Fluids, liquids in particular, are everywhere in everyday life. They may look simple and ordinary however it is as complex and challenging to create a simulation that looks realistic. As Premoze *et al.* (2003) stated:"Due to our familiarity with fluid movement, plausible simulation of fluids remains a challenging problem despite enormous improvements". While a well implemented simulations can create an epic and glorious effects in a scene, a poorly done simulation can cause a make-believe and cheap effect.

Even though fluid animation is a popular and old research area, the progression still continues. Besides simulating the motion of the fluid still being a challenge, the visualisation of the fluid is remains another important topic to be covered. There are several different approaches developed for different needs and different behaviors; such as, Eulerian grids and Lagrangian particles. This project focuses on using one of the most popular Lagrangian approach on fluid simulations: Smoothed Particle Hydrodynamics (SPH). The aim of this project is to simulate both Newtonian, water-like fluids, and Non-Newtonian, fluids with viscoplastic behavior, using SPH. The simulation was implemented in C++ with a simple OpenGL visualisation and data exporting functionality in

order to create more realistic visualisations using a secondary software package, Houdini.

In the following sections, previous work done on fluid simulations have been discussed. Third section is giving the essential technical background of computational fluid dynamics, Navier-Stokes equations and SPH. After this, the implementation of the project is being covered explaining the design, the main algorithm and the physical equations aswell as the neighbor searching procedure. At the fifth section, the pipeline of the project is explained. Following this section, the results obtained are discussed. And lastly, after analysing the results, future work is discussed in the conclusion.

# Chapter 2

# Related Work

In 1983, Reeves (1983) introducted particle systems - "a technique for modeling a class of fuzzy objects". With this, computer graphics started using particles to simulate physically based elements. As Foster and Metaxas (1996) were the first ones to solve 3D Navier-Stokes equations to simulate liquids on a regular grid. Using semi-Lagrangian method, Stam (1999) allowed using larger time-steps while being stable which was an important step towards real-time simulation of fluids. Foster and Fedkiw (2001) improved his work on liquids using a both level-set method for tracking the fluid interface.

Smoothed Particle Hydrodynamics(SPH) introduced by Lucy (1977) and Gingold and Monaghan (1977) as alternative gridless particle method to simulate astrophysical problems such as galaxial collisions, solving the dynamics of star formations. Desbrun and Gascuel (1995) is the first one to apply SPH to simulate implicit surfaces. In 1996, Desbrun and Gascuel (1996) used SPH to derive interaction for particle systems and from this point SPH became a popular approach in fluid simulations. Müller *et al.* (2003) used interaction force between particles from SPH and Navier-Stokes equations to simulate water with free surfaces. Same year, Premoze *et al.* (2003) introduced Moving-Particle Semi Implicit(MPS) method which allowed incompressible fluid simulations. In order to achieve a better incompressibility, Raveendran *et al.* (2011) proposed a hybrid method that uses Poisson solving and local density cor-

rection of the particles. He *et al.* (2012) presented a local Poisson SPH method to achieve a better fluid incompressibility. And recently, Ihmsen *et al.* (2014) gathered all recent work and created an extensive SPH fluid simulation.

For Non-Newtonian fluids, Goktekin *et al.* (2004) used grid-based method to simulate the stress tensor in these fluids. They used a linear Maxwell model with von Misses plastic yield condition. On the other hand, Mao and Yang (2005a) used a co-rotational Maxwell model and SPH to simulate Non-Newtonian fluids. Another approach was made by Clavet *et al.* (2005) using SPH and elastic springs to simulate the plastic behavior of viscoelastic fluids. Ellero *et al.* (2002) presented viscoelastic flows using SPH. Following their previous work, Mao and Yang (2005b) added heat transfer method to control the elasticity of fluids. Paiva *et al.* (2006) used SPH and heat equation to derive the viscosity which helps defining the stress tensor. And in Paiva *et al.* (2009) they extend their previous work and simulate the viscoplastic effect of solid materials using the same method. Recently, Andrade *et al.* (2014) used Cross model to simulate Non-Newtonian fluid flow under shear stress with jet buckling effects. The book of Chhabra (2006) is a great reference to understand the behavior of Non-Newtonian fluids.

Neighbor searching is the most critical step in SPH implementations. Teschner *et al.* (2003) proposed a spatial hashing algorithm for collision detection of deformable objects. This approach was used extensively in this project. Hierarchical tree method of Hernquist and Katz was implemented aswell.

# Chapter 3

# Computational Fluid Dynamics

Computational Fluid Dynamics (CFD) is a branch of fluid mechanics that is being used in Computer Graphics to create realistic animations of fluids such as liquids, smoke and fire. The fundamental basis of the fluid motion in these simulations mostly being solved by Navier-Stokes equations. There are several different methods that are using these equations which can be used to create a fluid simulation: Smoothed Particle Hydrodynamics (SPH) method, Eulerian methods, Moving Particle Semi-Implicit method, Lattice Boltzmann methods etc. Different methods are preferred for different needs.

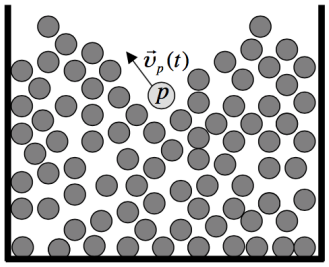
We use either the "Lagrangian" or the "Eulerian" description when defining the fluid flow. Both of these descriptions are explained in the following section.

## 3.1 Particle-Based vs Grid-Based Methods

### 3.1.1 Particle-Based Method

In Lagrangian description, we define the fluid flow with particles where each particle caries its own properties. This description can be gener-

alised as particle-based method. Each particle has various properties, such as mass, velocity, density etc. The logic behind this method is fairly straight-forward. Conservation of mass and Newton's laws apply directly to each fluid particle. Refer to figure 3.1 for visualisation.




Lagrangian description; snapshot

**Figure 3.1:** *Retrieved from MIT (2011)*

There are numerous different methods as meshfree solutions however this paper is focusing on using one of the most popular approaches in fluid simulations: Smoothed Particle Hydrodynamics (SPH). Although SPH was developed by Lucy (1977) and Gingold and Monaghan (1977) for the simulation of astrophysical problems, it is extensively being used in fluid simulations today. This approach is explained in section 3.3.

### 3.1.2 Grid-Based Method

In Eulerian description, we define the properties of the fluid at every point in space/field. Instead of keeping record of every single particle, we keep record of the cells of a grid. This description is often called a grid-based method. Refer to figure 3.2 for visualisation.

There are two main frames for describing grid-based methods: Eulerian grid and Lagrangian grid. The Eulerian description is a spatial description while The Lagrangian description is a material description. They correspond to two disparate kinds of grid of domain discretization, Liu *et al.* (2004). Grid-based techniques are more accurate than particle-based methods since it is easier to work with spatial derivatives

on a fixed grid.

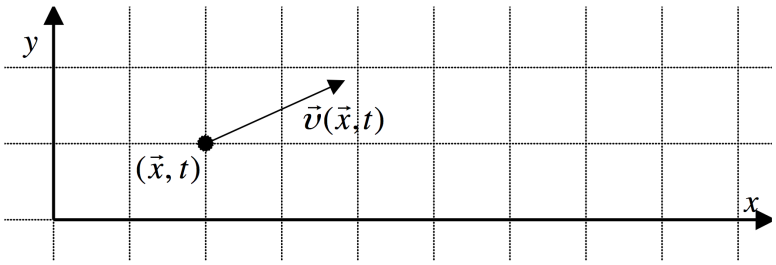



**Figure 3.2:** *Retrieved from MIT (2011)*

#### 3.1.2.1 Lagrangian Grid

In this description, the grid is fixed on the material and it moves together with the material. This method is very popular and succesful in solving computational solid mechanics however very difficult to apply for cases with exteremely distorted mesh, Liu *et al.* (2004).

#### 3.1.2.2 Eulerian Grid

The Eulerian Grid is fixed on the space, in which the simulated object is located and moves across the fixed cells in the grid. The volume of the cells remain unchanged for the entire simulation. Large deformations in the object don't cause any deformations in the mesh itself, therefore it is very popular in fluid simulations, Liu *et al.* (2004).

## 3.2 Navier-Stokes Equations

Navier-Stokes equations are describing the motion of a fluid at any point within a flow by a set of non-linear equations, Foster and Metaxas (1996).

These equations can be written in three dimensions:

$$\begin{aligned}
\frac{\partial u}{\partial t} + \frac{\partial u^2}{\partial x} + \frac{\partial uv}{\partial y} + \frac{\partial uw}{\partial z} &= -\frac{\partial p}{\partial x} + g_x + \nu(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}) \\
\frac{\partial v}{\partial t} + \frac{\partial vu}{\partial x} + \frac{\partial v^2}{\partial y} + \frac{\partial vw}{\partial z} &= -\frac{\partial p}{\partial y} + g_y + \nu(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}) \\
\frac{\partial w}{\partial t} + \frac{\partial wu}{\partial x} + \frac{\partial wv}{\partial y} + \frac{\partial w^2}{\partial z} &= -\frac{\partial p}{\partial z} + g_z + \nu(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2})
\end{aligned} \tag{3.1}$$

where $u, v, w$ are velocities in the $x, y, z$ directions while $p$ is the local pressure, $g$ is the gravity and $\nu$ is the viscosity of the fluid. They are derived from Newton's Second Law of Motion which means the momentum is always conserved, Foster and Metaxas (1996).

In much simpler terms, Müller *et al.* (2003) formulates the equation for Newtonian fluids as:

$$\rho\Big(\frac{D\mathbf{v}}{Dt}\Big) = -\nabla p + \rho\mathbf{g} + \mu\nabla^2\mathbf{v} \tag{3.2}$$

where $\rho$ is density, $\mu$ is viscosity constant; describes the same principal for Lagrangian fluids.

While Mao and Yang (2005a) formulates the equation for Non-Newtonian fluids as:

$$\frac{dv}{dt} = -\frac{1}{\rho}\nabla p + \frac{1}{\rho}\nabla \cdot T + \frac{\mu}{\rho}\nabla^2 v + \frac{1}{\rho}f \tag{3.3}$$

where $T$ being the stress tensor.

## 3.3 Smoothed Particle Hydrodynamics

SPH is an interpolation method which allows any function to be expressed in terms of its values at a set of disordered particles, Monaghan (1992). To do that, SPH distributed local quantities to the neighboring particles of each particle using radial symmetrical smoothing kernels, Müller *et al.* (2003).

The integral interpolant of any function A($\mathbf{r}$) is defined by

$$A(\mathbf{r}) = \int A(\mathbf{r}')W(\mathbf{r}-\mathbf{r}',h)\mathbf{dr}' \tag{3.4}$$

the integration is over the entire space where $W$ is an interpolating kernel with core radius $h$, Monaghan (1992). Smoothing kernel $W$ has two properties:

$$\int W(\mathbf{r}-\mathbf{r}',h)\mathbf{dr}' = 1 \tag{3.5}$$

and

$$\lim_{h\to 0} W(\mathbf{r}-\mathbf{r}',h) = \delta(\mathbf{r}-\mathbf{r}') \tag{3.6}$$

where the limit is to be interpreted as the limit of the corresponding integral interpolants, Monaghan (1992).

The default, gradient and Laplacian of A are:

$$\begin{aligned} A_S(\mathbf{r}) &= \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r}-\mathbf{r}_j,h), \\ \nabla A_S(\mathbf{r}) &= \sum_j m_j \frac{A_j}{\rho_j} \nabla W(\mathbf{r}-\mathbf{r}',h), \\ \nabla^2 A_S(\mathbf{r}) &= \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(\mathbf{r}-\mathbf{r}',h), \end{aligned} \tag{3.7}$$

where $m_j$ is the mass of particle $j$ and $\rho_j$ is the density of particle $j$ and $A_j$ is the field quantity at $\mathbf{r}_j$.

Field elements such as pressure, viscosity are explained in the implementation section together with example kernel functions(chapter 4) used in the implementation.

# Chapter 4

# Implementation

## 4.1 Design

Designing an extensive fluid simulation is a challenging process. This implementation contains two different type of fluids but the aim is to maintain these different approaches into a single implementation. In order to make a flexible structure, main elements were divided into their own classes and gathered in a main solver class, Fluid class. UML diagram can be seen in figure 4.1.

## 4.2 Algorithm

The main algorithm implemented for both fluid types are very similar. Newtonian fluid algorithm (Algorithm 1) is based on Müller *et al.* (2003) and Non-Newtonian algorithm(Algorithm 2) is based on Mao and Yang (2005a). And both of these algorithms can be found below as pseudo-code.

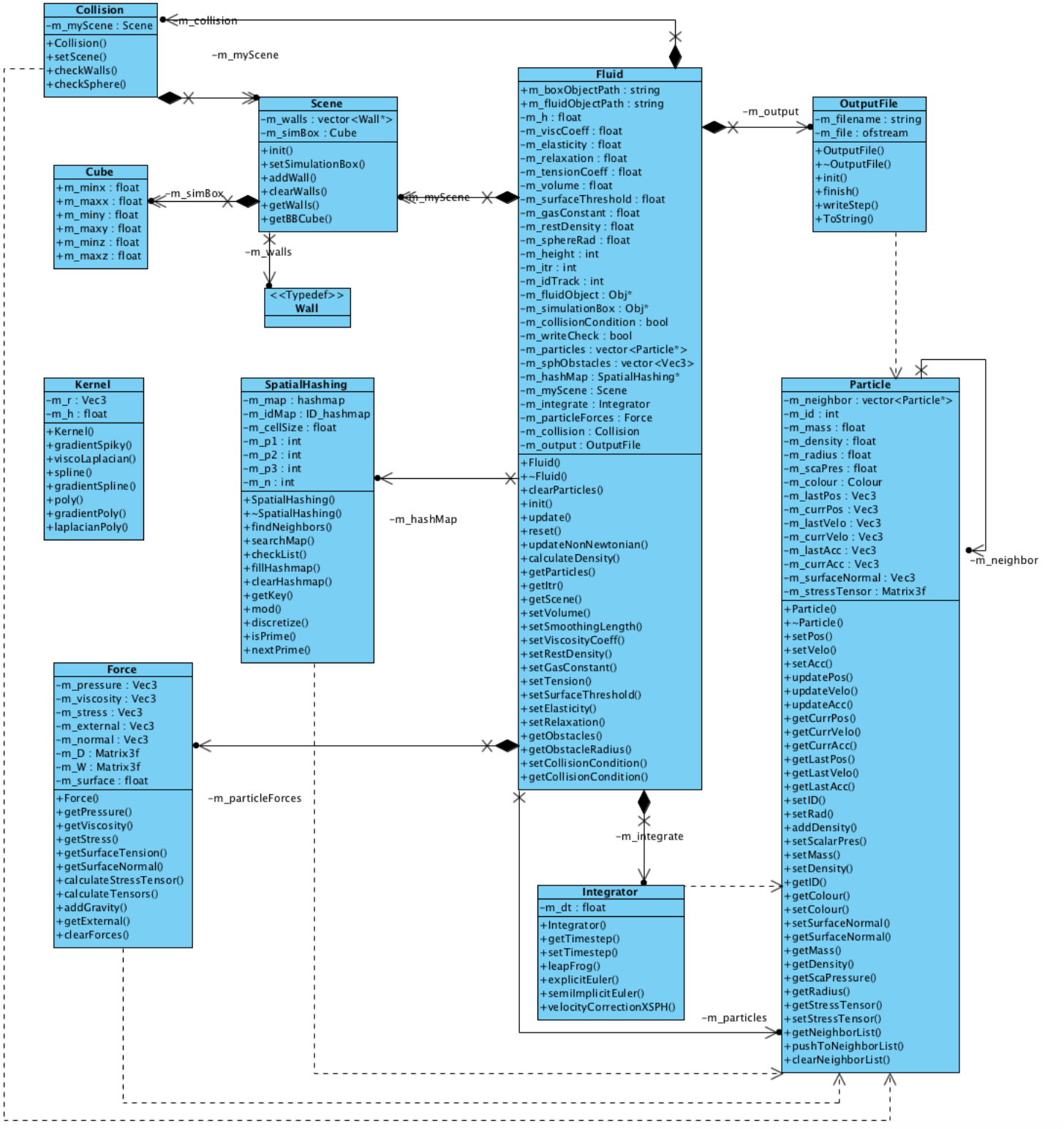



**Figure 4.1:** *UML Diagram*

### 4.2.1 Newtonian Fluid

**foreach** *particle* $i$ **do**
  find neighbors $j$;
  $\rho_j = \sum_j m_j W_{ij}$ (Eq. 4.2);
  compute $p_i$ using $\rho_i$ (Eq. 4.6);
**end**
**foreach** *particle* $i$ **do**
  **if** $i \neq j$ **then**
    $\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i}\nabla p_i$ (Eq. 4.4);
    $\mathbf{F}_i^{viscosity} = m_i v \nabla^2 \mathbf{v}_i$ (Eq. 4.8);
  **end**
  $\mathbf{F}_i^{surface} = m_i \nabla^2 c_S$ (Eq. 4.10);
  $\mathbf{F}_i^{other} = m_i \mathbf{g}$;
  $\mathbf{F}_i(t) = \mathbf{F}_i^{pressure} + \mathbf{F}_i^{viscosity} + \mathbf{F}_i^{surface} + \mathbf{F}_i^{other}$;
**end**
**foreach** *particle* $i$ **do**
  $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i + \Delta t \mathbf{F}_i(t)/m_i$
  $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$
  collision detection and response
  Correct $\mathbf{v}_i$ using XSPH (Eq: 4.37)
**end**

**Algorithm 1:** Newtonian Fluid SPH

### 4.2.2 Non-Newtonian Fluid

**foreach** *particle* $i$ **do**
  find neighbors $j$;
  $\rho_j = \sum_j m_j W_{ij}$ (Eq. 4.2);
  compute $p_i$ using $\rho_i$ (Eq. 4.6);
**end**
**foreach** *particle* $i$ **do**
  compute $\omega$ and $D$ (Eq: 4.16);
**end**
**foreach** *particle* $i$ **do**
  **if** $i \neq j$ **then**
    $\mathbf{F}_i^{pressure} = -\frac{m_i}{\rho_i}\nabla p_i$ (Eq. 4.4);
    $\mathbf{F}_i^{viscosity} = m_i v \nabla^2 \mathbf{v}_i$ (Eq. 4.8);
  **end**
  $\mathbf{F}_i^{stress} = \frac{m_i}{\rho_i} T \nabla$ (Eq. 4.19);
  $\mathbf{F}_i^{other} = m_i \mathbf{g}$;
  $\mathbf{F}_i(t) = \mathbf{F}_i^{pressure} + \mathbf{F}_i^{viscosity} + \mathbf{F}_i^{stress} + \mathbf{F}_i^{other}$;
**end**
**foreach** *particle* $i$ **do**
  $\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i + \Delta t \mathbf{F}_i(t)/m_i$
  $\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t)$
  collision detection and response
  Correct $\mathbf{v}_i$ using XSPH (Eq: 4.37)
**end**

**Algorithm 2:** Non-Newtonian Fluid SPH

## 4.3 Mass - Density

### 4.3.1 Mass

Mass and density elements are the most important parameters for SPH fluids. Since SPH is a Lagrangian approach, every single particle holds their own mass and density values. Even though the mass value is same for all particles, density parameter changes according to the neighboring

particles.

In order to calculate the mass of each particle, we use an adjusted version of volumetric mass density formula presented by Kelager (2006):

$$\rho_0 = \frac{m}{V} n \tag{4.1}$$

where $\rho$ is the default density of the fluid, $m$ is the mass of single particle, $V$ is the volume of the fluid object and $n$ being the particle count. This mass calculation is done at initialisation stage and stays the same for the entire simulation process. Calculating the mass is a more stable approach than allowing to the user to assign it because it's crucial to get physically correct value related to rest density. And since NGL OBJ class is being used in this implementation, it is easy to retrieve the particle count which is used in equation 4.1, Macey (a). However, the volume parameter is to be set by the user.

### 4.3.2 Density

As explained in section 3.3, each particle $i$ represents a certain volume inside the fluid. While mass of every particle, $m_i$, stays the same, density of every particle $\rho_i$ changes. In this implementation, the SPH approximation applied to the density as suggested by Monaghan (1992). This technique can be called as "density summation" and can be found in equation 4.2.

$$\begin{aligned} \rho_i(\mathbf{r}) &= \sum_j \rho_j \frac{m_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) \\ &= \sum_j m_j W(\mathbf{r} - \mathbf{r}_j, h) \end{aligned} \tag{4.2}$$

Density for each particle is calculated according to its neighboring particles. Finding the neighboring particles of every particle is explained in section 4.10. The density is updated every time-step for every particle.

## 4.4 Pressure

When the pressure term is applied on equation 3.7, we get the following equation:

$$\mathbf{f}_i^{pressure} = -\nabla p(\mathbf{r}_i) = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{4.3}$$

However, the pressure force is not symmetrical. When two particles interact, particle $i$ will only use the pressure of particle $j$ to compute its pressure force and vice versa. Müller *et al.* (2003) solved this problem by taking the arithmetic mean of pressures of interacting particles:

$$\mathbf{f}_i^{pressure} = \sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h). \tag{4.4}$$

The gradient of kernel function for the pressure term is explained explicitly in section 4.9. The individual pressure values for every particle, $p_i$ and $p_j$ are calculated right after the density is calculated. The ideal gas state equation is being used for this:

$$p = k\rho, \tag{4.5}$$

where k is the gas constant. But in order to keep density at $\rho_0$, Desbrun and Gascuel (1996) proposed to use:

$$p = k(\rho - \rho_0), \tag{4.6}$$

which comes from $(P + P_0)V = k$ where $V = 1/\rho$ is the volume per unit mass, and $P_0 = k\rho_0$, Desbrun and Gascuel (1996). This provides a constant density which results in a constant volume which allows the fluid to back to its initial volume after deformation, Desbrun and Gascuel (1996).

## 4.5 Viscosity

The viscosity is the fluid' resistance against deformation by shear stress or tensile stress. Chhabra (2006) defines viscosity for Newtonian fluids as, the value $\mu$ is independent of shear rate and it depends only on temperature and pressure. In a graph of shear stress ($\tau_{yx}$) vs shear rate ($\dot{\gamma}_{yx}$), which is called a rheogram or flow curve, a Newtonian fluid is a straight line of slope $\mu$ passing through the origin, Chhabra (2006). On the other hand, "when the viscosity depends upon flow conditions, such as flow geometry, shear rate (or stress) developed within the fluid, time of shearing, kinematic history of the sample etc." the fluid is called as Non-Newtonian, Chhabra (2006).

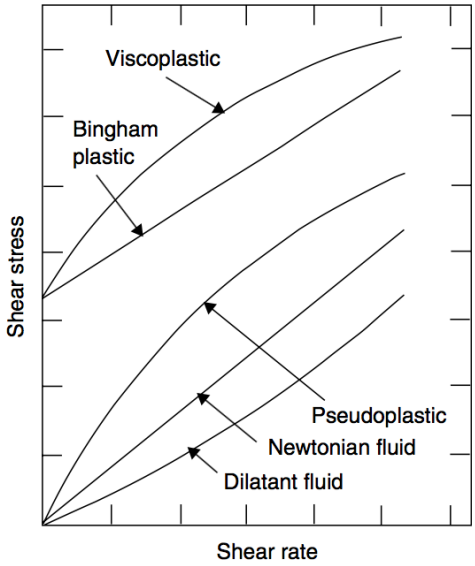



**Figure 4.2:** *Shear Stress vs Shear Rate Graph. Retrieved from Chhabra (2006)*

When the viscosity term is applied on equation 3.7, we get the following equation:

$$f_i^{viscosity} = \mu\nabla^2\mathbf{v}(\mathbf{r}_a) = \mu\sum_j m_j\frac{\mathbf{v}_j}{\rho_j}\nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h) \qquad (4.7)$$

but similar to the pressure term, this equation gives asymmetric forces because the velocity field varies from particle to particle. Müller *et al.*

(2003) proposed a solution since viscosity force depends on the difference of velocities, the force can be symmterized by using a velocity difference instead of absolute velocities:

$$f_i^{viscosity} = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h). \tag{4.8}$$

$\mu$ value is the viscosity coefficient in these equations. Increasing this value will make the fluid more viscous. The laplacian of kernel function of this equation is explained explicitly in section 4.9.

## 4.6 External Forces

### 4.6.1 Gravity

External forces such as gravity or collision forces are applied directly to each particle without using any SPH approximations. Gravity, $\mathbf{g}$, is applied to the net force without dividing by any density.

### 4.6.2 Collision Forces

Collision forces applied in this implementation are simple and straightforward. After detecting a collision between particle and the object, the particle is translated outside of the object with a reflected velocity that is perpendicular to the object's surface.

There are only two collision detection and response algorithms implemented for this project: collision with the boundaries (simulation box) and collision with spheres. They both use the similar approach as explained above. Collision with arbitrary objects can be implemented as a future work for this project.

## 4.7 Surface Tension

"The cohesive forces between molecules down into a liquid are shared with all neighboring atoms. Those on the surface have no neighboring atoms above, and exhibit stronger attractive forces upon their nearest neighbors on the surface. This enhancement of the intermolecular attractive forces at the surface is called surface tension" HyperPhysics. Which can be summarized as the elastic tendency which makes the liquid to acquire least surface area possible. Surface tension implementation in this paper is referenced from Müller *et al.* (2003) which referenced their work from Morris (2000). Surface tension implemented in this paper aimed for single fluid usage however it is possible to extend this using interface tension using the work presented by Morris (2000) in the future.

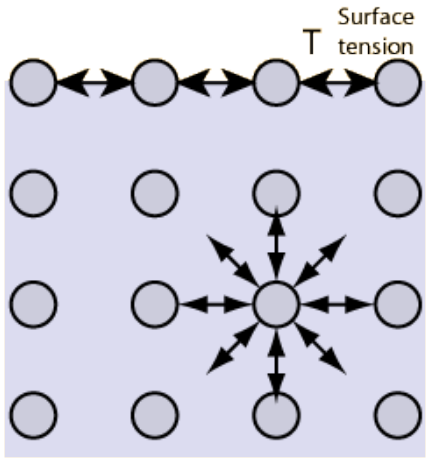



**Figure 4.3:** *Surface tension illustration. Retrieved from HyperPhysics*

Surface of the fluid is represented as color field in literature, which is 1 at particle locations and 0 elsewhere. In order to adapt this to SPH approximation the equation can be written as,

$$c_S(\mathbf{r}) = \sum_j m_j \frac{1}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h). \tag{4.9}$$

To get the surface tension force of each particle, the following equation

is implemented:

$$f^{surface} = \sigma\kappa\mathbf{n} = -\sigma\nabla^2 c_S \frac{\mathbf{n}}{|\mathbf{n}|}, \tag{4.10}$$

where $\mathbf{n}$ is the gradient field of the smoothed color field:

$$\mathbf{n} = \nabla c_s, \tag{4.11}$$

and $\sigma$ being the tension coefficient.

When calculating $\mathbf{n}/|\mathbf{n}|$ small $|\mathbf{n}|$ values causing problems. Therefore as suggested in Müller *et al.* (2003), a threshold value, $l$, is defined which identifies a particle as a surface particle if

$$|\mathbf{n}(\mathbf{r}_i)| > l. \tag{4.12}$$

Tension coefficient, $\sigma$, threshold value, $l$ are user defined parameters in this implementation.

## 4.8 Stress Tensor

The stress tensor is a nonlinear function of deformation tensor as described by Paiva *et al.* (2006). There are several different approaches to calculate the stress tensor. This implementation computes the stress tensor by integrating the tensor-rate as described in Mao and Yang (2005a) and Goktekin *et al.* (2004). The tensor-rate in this model is based on a nonlinear corotational Maxwell model defined by Ellero *et al.* (2002). This method is chosen for this implementation because the aim was to implement the stress tensor independent from the temperature.

This method is implemented according to the work of Mao and Yang (2005a) and can be found below. The tensor-rate model can be written as:

$$\frac{dT}{dt} = \Omega + \frac{\mu_e}{2} D' - \frac{1}{\lambda} T \tag{4.13}$$

where $\Omega$ is the rotational tensor, $\mu$ elasticity constant, $\lambda$ relaxation time

and $T$ being the stress tensor. $\Omega$ is described as "the coordinate transformation between the global inertial frame and a coordinate frame rotating with the instantaneous fluid angular velocity at the particle" by Mao and Yang (2005a):

$$\Omega = \frac{1}{2}(T \bullet \omega - \omega \bullet T). \tag{4.14}$$

$D'$ is the traceless strain tensor and can be calculated with:

$$D' = D - \frac{\text{Trace}(D)}{3} I. \tag{4.15}$$

Both $w$ and $D$ calculation is using the following similar procedure: the velocity gradient.

$$\begin{aligned} \omega &= \nabla v - (\nabla v)^T \\ D &= \nabla v + (\nabla v)^T \end{aligned} \tag{4.16}$$

The tensors can be expressed as their individual elements:

$$\begin{aligned} \omega^{\alpha\beta} &= \frac{\partial v^\beta}{\partial r^\alpha} - \frac{\partial v^\alpha}{\partial r^\beta} \\ D^{\alpha\beta} &= \frac{\partial v^\beta}{\partial r^\alpha} + \frac{\partial v^\alpha}{\partial r^\beta} \end{aligned} \tag{4.17}$$

In equation 4.17, $\alpha$ and $\beta$ values are representing the 3D spatial coordinates. Since the implementation is based on SPH method, the stress tensor elements needs to be formulated under SPH approximation aswell. The partial velocity derivative from equation 4.17 can be found with:

$$\frac{\partial v^\alpha}{\partial r^\beta} = \sum_{j=1}^{n} \frac{m_j}{\rho_j}(v_j^\alpha - v^\alpha)\frac{\partial W(r - r_j, h)}{\partial r^\beta} \tag{4.18}$$

Using equation 4.18, all the elements in $D$ and $\omega$ are calculated. Afterwards, with equations 4.15 and 4.14 we obtain equation 4.13. Using the current time-step, $T$ value of every particle is calculated.

In order to obtain the stress term of every single particle, the following

approximation from Paiva *et al.* (2006) have been used:

$$\frac{1}{\rho_i}\nabla \cdot \mathbf{T}_i = \sum_{j=1}^{n} \frac{m_j}{\rho_i \rho_j}(\mathbf{T}_i + \mathbf{T}_j) \cdot \nabla_i W(\mathbf{x}_{ij}, h). \tag{4.19}$$

As explained in section 4.2.2, instead of surface tension term, this stress term is used. Elasticity constant, $\mu_e$, and relaxation time, $\lambda$ are user-defined parameters.

Other approaches to stress tensor calculation can be made such as Generalized Newtonian Liquid model proposed by de Souza Mendes *et al.* (2007) that is used in Paiva *et al.* (2006). However, in this implementation, the aim was to implement the stress tensor independent from the temperature. This approach can be implemented as a future work.

## 4.9 Smoothing Kernels

Smoothing kernel functions are one of the most important points in SPH method. Stability, accuracy and speed of the whole method depends on these functions. Different kernels are being designed for different purposes. Since this paper is based on the work of Müller *et al.* (2003) and Mao and Yang (2005a), the kernel functions used in these papers were implemented for this project.

The basic logic of using kernel functions is to determine how much the neighboring particles are influenced from the function $A(\mathbf{r})$. The rules to follow when designing a kernel can be found in section 3.3. A kernel function takes two parameters as input: $\mathrm{x}_i - \mathrm{x}_j$, being the distance between the particle and its neighbor and $h$, being the smoothing length.

A separate kernel class is implemented in this project to have a flexibility to test different kernel functions with ease. After testing various different kernel functions, it's decided to use the ones presented in Müller *et al.* (2003) and Mao and Yang (2005a).

### 4.9.1 Poly6 Kernel

The default kernel function used in Müller *et al.* (2003) for density calculation (equation 4.2). This kernel is also known as the 6th degree polynomial kernel.

$$W_{poly6}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - ||\mathbf{r}||^2)^3, & 0 \leq ||\mathbf{r}|| \leq h \\ 0, & \text{otherwise} \end{cases} \tag{4.20}$$

The gradient of this kernel function used for surface normal (equation 4.11).

$$\nabla W_{poly6}(\mathbf{r}, h) = -\frac{945}{32\pi h^9} \begin{cases} \mathbf{r}(h^2 - ||\mathbf{r}||^2)^2, & 0 \leq ||\mathbf{r}|| \leq h \\ 0, & \text{otherwise} \end{cases} \tag{4.21}$$

The laplacian of this kernel function used for surface tension calculation (equation 4.10).

$$\nabla^2 W_{poly6}(\mathbf{r}, h) = -\frac{945}{32\pi h^9} \begin{cases} (h^2 - ||\mathbf{r}||^2)(3h^2 - 7||\mathbf{r}||^2), & 0 \leq ||\mathbf{r}|| \leq h \\ 0, & \text{otherwise} \end{cases} \tag{4.22}$$

As Müller *et al.* (2003) stated, if this kernel is used for the computation

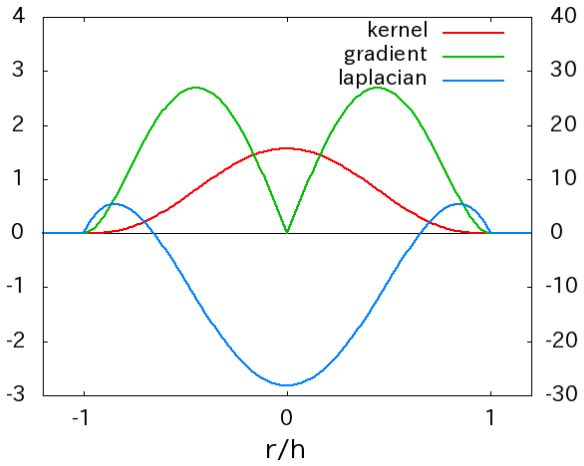



**Figure 4.4:** *Poly6 Kernel Graph. Retrieved from PukiWiki*

of pressure forces, particles tend to build clusters under high pressure

because "as particles get very close to each other, the repulsive force vanishes because the gradient of the kernel approaches zero at the center". Another kernel, spiky kernel, is proposed by Desbrun and Gascuel (1996) to solve this problem.

### 4.9.2 Spiky Kernel

The kernel proposed by Desbrun and Gascuel (1996) to solve the particle clustering problem which is used to calculate the pressure force (equation 4.4).

$$\nabla W_{pressure}(\mathbf{r}, h) = -\frac{45}{\pi h^6} \begin{cases} \frac{\mathbf{r}}{||\mathbf{r}||}(h - ||\mathbf{r}||)^2, & 0 \leq ||\mathbf{r}|| \leq h \\ 0, & \text{otherwise} \end{cases} \tag{4.23}$$

As can be seen from figure 4.5, the kernel function no longer tends

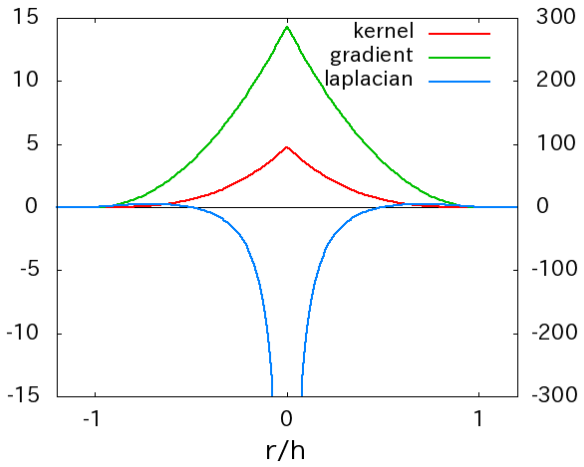



**Figure 4.5:** *Spiky Kernel Graph. Retrieved from PukiWiki*

towards zero.

### 4.9.3 Viscosity Kernel

"Viscosity is a property arising from collisions between neighboring particles in a fluid that are moving at different velocities", M. A. Boda (2015). However as mentioned in Müller *et al.* (2003); for two particles

that get close to each other, the Laplacian of the smoothed velocity field can get negative resulting in forces that increase their relative velocity. Because of this, artifacts may appear in coarsely sampled velocity fields. In order to overcome this problem, a kernel whose Laplacian is positive everywhere is being used.

$$\nabla^2 W_{viscosity}(\mathbf{r}, h) = \frac{45}{\pi h^6} \begin{cases} (h - ||\mathbf{r}||), & 0 \leq ||\mathbf{r}|| \leq h \\ 0, & \text{otherwise} \end{cases} \tag{4.24}$$

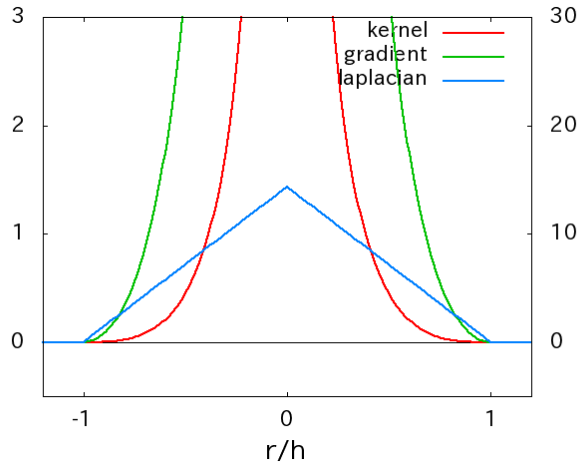



**Figure 4.6:** *Viscosity Kernel Graph. Retrieved from PukiWiki*

### 4.9.4 Spline Kernel

Mao and Yang (2005a) used the traditional spline kernel described in Monaghan (1992). The default spline kernel used for the density calculation and the gradient spline kernel is used for stress term calculations including the approximation of the stress tensors.

$$W_{spline}(\mathbf{r}, h) = \frac{1}{\pi h^3} \begin{cases} 1 - \frac{3}{2}\left(\frac{r}{h}\right)^2 + \frac{3}{4}\left(\frac{r}{h}\right)^3, & 0 \leq \frac{r}{h} \leq 1 \\ \frac{1}{4}\left(2 - \frac{r}{h}\right)^3, & 1 \leq \frac{r}{h} \leq 2 \\ 0, & \text{otherwise} \end{cases} \tag{4.25}$$

$$\nabla W_{spline}(\mathbf{r}, h) = \frac{9}{4\pi h^5} \begin{cases} \left(\frac{r}{h} - \frac{4}{3}\right)\mathbf{r}, & 0 \leq \frac{r}{h} \leq 1 \\ -\frac{1}{3}\left(2 - \frac{r}{h}\right)^2 \frac{h}{r}\mathbf{r}, & 1 \leq \frac{r}{h} \leq 2 \\ 0, & \text{otherwise} \end{cases} \tag{4.26}$$

Although this kernel is being proposed by Mao and Yang (2005a) in their paper, it wasn't effective in this implementation. The implementation remains in the final code, however not being used by the functions.

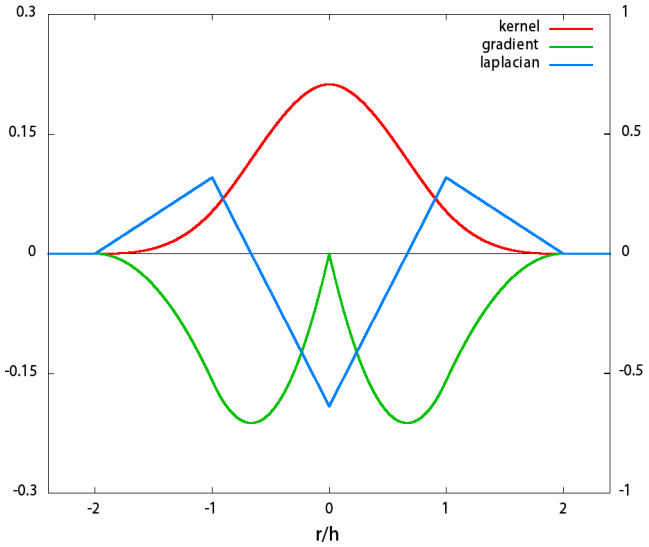



**Figure 4.7:** *Spline Kernel Graph. Retrieved from PukiWiki*

### 4.9.5 Smoothing Length

Smoothing length, $h$, is one of the most important parameters that affects the whole SPH method by changing the kernel value results and neighbor searching results. Too small or too big values would cause instabilities and inconsistencies to the simulation. Kelager (2006) developed a formulation on this issue however this hasn't been used in this implementation. The smoothing length is a user-specified variable for both of the simulations.

## 4.10 Neighbor Search

Neighbor search is one of the most crucial procedures in SPH method considering all interpolation equations, $A(\mathbf{r})$, needs the neighbor list for every particle (refer to equation 3.7). A naïve neighbor searching approach would end up with a complexity of $O(n^2)$. This complexity is not good enough since it is impossible to reach any interactive speed when the particle count increases. With an efficient nearest neighbor searching (NNS) algorithm, it is possible to have a significant performance increase since it is the most time consuming procedure in SPH computation.

Different algorithms are used for different approaches. In this project, two different NNS algorithms were implemented: Spatial Hashing and Hierarchical Tree. Spatial hashing algorithm is mainly referenced on Teschner *et al.* (2003) work and hierarchical tree algorithm is mainly referenced on Paiva *et al.* (2006) and Hernquist and Katz. The approach of these algorithms are very different. After testing both of these algorithms, Spatial Hashing was found to be more accurate and more efficient for this simulation. Therefore, it was used in this implementation.

### 4.10.1 Spatial Hashing

Spatial hashing is mapping the 3D world to a 1D hash table using a hash function. This hash function generates hash keys for each "cell" of the 3D world which allows to reach the cell in constant time. It doesn't always provide a unique mapping of grid cells to hash table entries. Grid cells with same index decreases the performance of algorithm. However, Teschner *et al.* (2003) narrowed down this problem using optimized parameters in their algorithm. This implementation uses multimap container of Standard C++ library for the primary hash map which allows multiple elements to have the same index and make use of this weakness.

#### 4.10.1.1 Discretization

In the first pass, the position of the neighbor searching particle, $i$, is being discretized with respect to a user-defined cell size, $l$. The discretization process is to divide the given coordinates, $(x, y, z)$, by the given cell size and round down to next integer:

$$\hat{\mathbf{r}}(\mathbf{r}_i) = \left(\lfloor \mathbf{r}_x/l \rfloor, \lfloor \mathbf{r}_y/l \rfloor, \lfloor \mathbf{r}_z/l \rfloor\right)^T \tag{4.27}$$

#### 4.10.1.2 Hash Function

Using the discretized coordinates,$(i, j, k)$, the hash function maps the coordinate to a 1D hash index:

$$h : h = hash(i, j, k). \tag{4.28}$$

The hash function used in this implementation is the same one defined in Teschner *et al.* (2003):

$$hash(i, j, k) = (i\ p1\ \mathbf{xor}\ j\ p2\ \mathbf{xor}\ k\ p3)\mathbf{mod}\ n. \tag{4.29}$$

However the grid cell with same index is being utilised in this implementation, it is crucial to keep this count as low as possible. $p1, p2, p3$ are large prime numbers: $73856093, 19349663, 83492791$ respectively and the value $n$ is the hash table size. Usage of prime numbers is important to obtain unique index values.

#### 4.10.1.3 Table Size and Cell Size

Teschner *et al.* (2003) states that larger table size reduces the risk of mapping different 3D positions to the same hash index. Kelager (2006) proposes to use

$$h = prime(2n), \tag{4.30}$$

$n$ being the number of particles. Ihmsen *et al.* (2011) also indicates the approach of choosing the table size two times the number of particles is

appropriate.

Grid cell size influences the number of particles that are mapped to same index value. Therefore, it has the most singificant impact on the performance than table size or the hash function as mentioned in Teschner *et al.* (2003). Both Ihmsen *et al.* (2011) and Kelager (2006) proposed to use the smoothing length (section 4.9.5) as the grid cell size. After having some tests, the grid cell size was set to smoothing length in this implementation aswell.

#### 4.10.1.4 Particle Queries

In the second pass, the neighbor searching algorithm takes place. Teschner *et al.* (2003), Ihmsen *et al.* (2011) and Kelager (2006) proposes a bounding box based particle query. The same approach is applied in this implementation. After iterating through the bounding box, the same hash index valued candidates were searched to add to the neighbor list.

A bounding box defined around the particle using the grid cell size. Two corner points, minimum and maximum were found by adding and subtracting the cell size from the position of the particle, $\mathbf{r}_i$.

$$BB_{min} = \mathbf{r}_i - (h, h, h)^T \ , \ BB_{max} = \mathbf{r}_i + (h, h, h)^T \tag{4.31}$$

After finding the bounding box, an iteration has been done over three dimensions. At every iteration, the candidate position is being searched if it's in the searching radius. The iteration over three dimensions were made using a float increment value which is used by Priscott (2010).

#### 4.10.1.5 Searching Algorithm

The searching algorithm was implemented as a separate function for particle query. As mentioned above, multimap of Standard C++ was used for hash map in this implementation. Given the input candidate position with a hash index key, hash map has been iterated. If the candidate already exists in the neighbor list, it passes on next candidate.

The checking of existing neighbor particles is done using hash mapping aswell with one difference. Instead of using a multimap, a normal hash map of Standard C++ library has been used since unique results were needed for this procedure. This approach increased the performance rather than using a dynamic list and iterating through the whole list.

If the candidate is not in the little hash map for existing neighbors, the distance between the candidate and the current particle is calculated. If this distance is smaller than the grid size, the candidate is finally pushed into the dynamic neighbor list of the particle. And the candidate is also added to the little hash map.

$$||\mathbf{r}_i - \mathbf{r}_j|| \leq h \tag{4.32}$$

The complexity of the first pass, creating the hash map, takes $O(n)$ time since all the vertices are to be evaluated. The particle query complexity is $O(n \cdot q)$ where $q$ is the average number of vertices per cell.

### 4.10.2 Hierarchical Tree

Paiva *et al.* (2006) proposed to use an adaptive hierarchy tree search to find particle neighbors. Since the simulation takes place in three dimensions, octree data structure was used in this approach. They used the work of Hernquist and Katz as their main reference. This approach was the first NNS algorithm implemented in this project.

An octree structure has been adapted from Macey (b) Octree Demo. The hierarchy tree is formed with a pre-defined height. The simulation box is divided recursively into eight pieces, nodes. The nodes at $height = 1$ are called leaves. Each node has a surrounding box for particle query which is used to check if the particle is inside the node. Each parent node, contains the elements that are divided through its children.

The particle is being checked at each level of the tree if it's intersecting the node. If it does, descend one level down in that node. After reaching to leaves, bottom level, particle has been checked if its within the search

distance, $h$. If the particle is in the range, add it to the neighbor list. Neighbor searching is done when the whole tree has been traversed. The search distance set to be smoothing length in this implementation.

The complexity of this tree search method is $O(nlog(n))$, $n$ being the number of particles. The performance of this algorithm is worse than Spatial Hashing method. In addition, the results obtained using this NNS algorithm wasn't accurate and stable for this implementation. Therefore as mentioned above, Spatial Hashing method was preferred.

## 4.11 Integration Methods

After calculating the net force/acceleration on the particle, an integration is needed to find the new velocity and position. Two basic integration methods have been implemented in this project: Leap-Frog Method and Explicit Euler Method.

### 4.11.1 Leap-Frog

Leap-Frog method is a popular integration method in SPH implementations. It is a second order method compared to first order Euler integration. It has a stable behavior for oscillatory motion as long as the time-step is constant as mentioned by Birdsall and Langdon (2005).

In Leap-Frog integration, the position and velocities are updated at interleaved time points which means they 'leapfrog' each other. In order to find the new velocity and position, the following equations are being used:

$$x_{i+1} = x_i + v_i \Delta t + \frac{1}{2} a_i \Delta t^2, \tag{4.33}$$

$$v_{i+1} = v_i + \frac{1}{2}(a_i + a_{i+1}) \Delta t. \tag{4.34}$$

In order to keep track of the unsynchronized velocity, position and acceleration in the implementation; besides the current values, the previous

values of velocity, position and acceleration have been kept in each particle.

### 4.11.2 Explicit Euler

Euler method is a simple integration method for solving ordinary differential equations with a given initial value. Given its simple structure, it is a very unstable integration method. Very small time-steps needed to be used.

$$\mathbf{v}_i^{t+h} = \mathbf{v}_i^t + \mathbf{F}_i^t \frac{h}{m_i}, \tag{4.35}$$

$$\mathbf{x}_i^{t+h} = \mathbf{x}_i^t + \mathbf{v}_i^{t+h} h, \tag{4.36}$$

where $\mathbf{v}_i^t$ is the velocity at time $t$, $\mathbf{F}_i^t$ is the net force, $\mathbf{x}_i^t$ is the position and $h$ being the time-step.

## 4.12 XSPH Velocity Correction

Monaghan (1989) introduced an improvement called XSPH velocity-correction, to prevent particle inter-penetration which causes stable clusters of particles. Using the following equation after the integration step, more stable results were obtained.

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \epsilon \sum_{j=1}^{n} \frac{m}{0.5(\rho_i + \rho_j)} (\mathbf{v}_j - \mathbf{v}_i) W(\mathbf{x}_{ij}, h) \tag{4.37}$$

By computing the average velocity from the velocities of the neighboring particles, the flow of the particles are kept in order. The variable $\epsilon$ is the correction constant. It's to be given between $[0, 1]$. High values create unrealistic effects on the simulation.

# Chapter 5

# Pipeline

The aim of this project as mentioned in the introduction before, is to simulate the motion of both Newtonian and Non-Newtonian fluids. Therefore, the visualisation wasn't the main focus point. In order to overcome this, an output file functionality provided to export .txt files. The stand alone application was assisted with input and output features using a Houdini Digital Asset. This chapter will explain the features of the both stand alone application and the Houdini Digital Asset.

## 5.1 Stand Alone Application

As described in chapter 4, the implementation is done following the works of Müller *et al.* (2003) and Mao and Yang (2005a). It was written in C++ and used following libraries: NGL of Macey (a), Boost C++ Libraries by boo and Eigen C++ Library by B. Jacob.

Used Vec3 class and few demo programs of NGL for OpenGL visualisation. Used Boost::ForEach macro in for loops. Used Matrix class of B. Jacob in the calculations of stress tensor.

The user interface of the application is simple and straight-forward: OpenGL visualisation is not responsive when the particle count is over eight thousand particles. Therefore, the functionality to export position data of each particle is implemented to visualise in a secondary software.

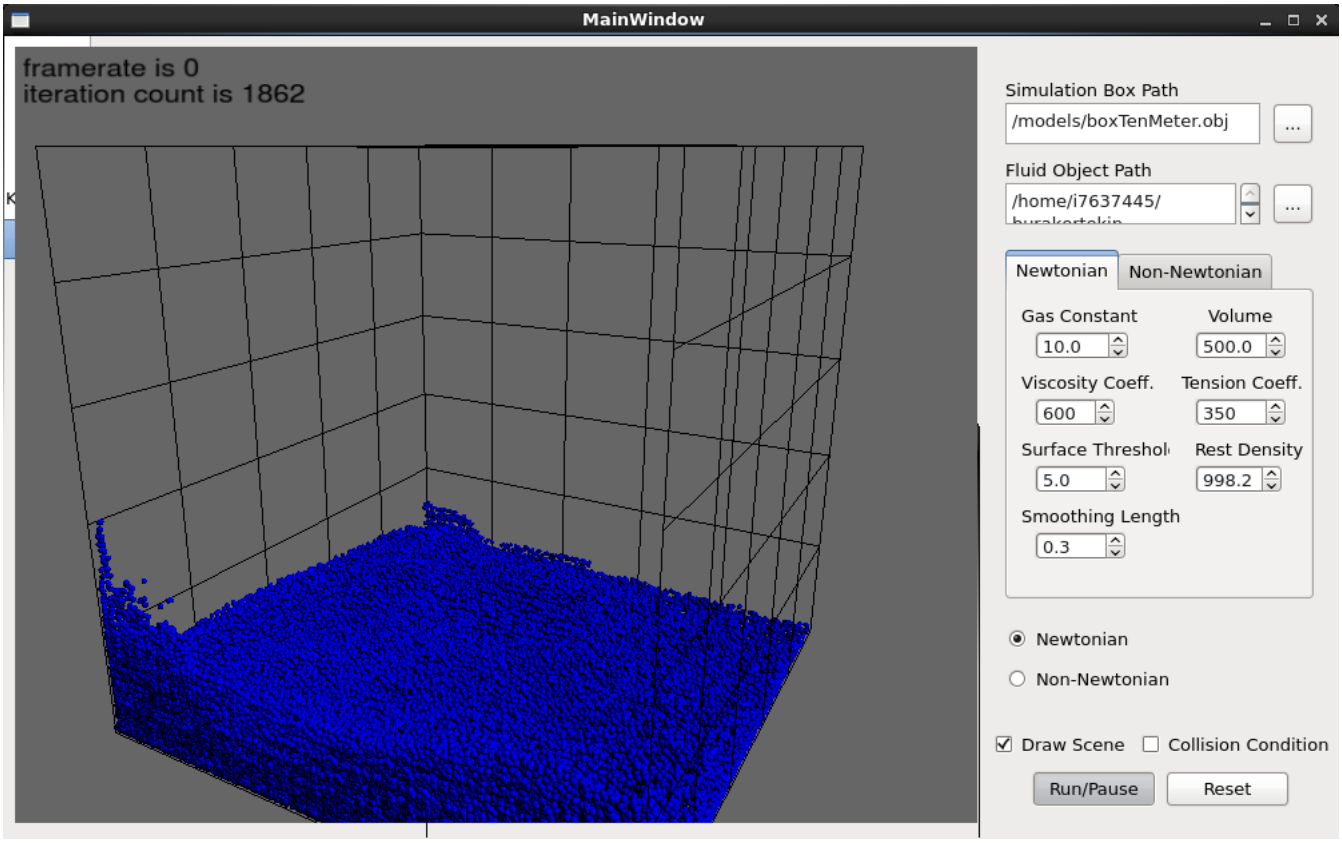



**Figure 5.1:** *User Interface of Stand Alone Application*

Output files (.txt files) generated to visualise can be found in the output folder of the project folder.

Houdini digital asset can be used to generate the input fluid .obj files. This process has been explained in appendix. For the simulation box, a regular box shaped object can be used as input. However, this box should be cubic shaped due to the limitation of the boundary box creation within the implementation.

## 5.2 Houdini Digital Asset

Houdini Digital Asset is created to assist the existing application. The application needs .obj files as inputs for both simulation box and the fluid object. Using the "Points from Volume" node of Houdini, a feature added in digital asset so the user can create input files easily.

In the digital asset, there are three sections: input generator, simulation visualisation and cache out. Input generator can be used to create proper .obj files with points. Simulation visualisation reads in the .txt files generated by the stand alone application and allows user to have simple options on visusalisation. Cache out section is to create .bgeo files of the scenes to have it ready for later use.

Using the digital asset is straight-forward but will be explained with images in Appendix.

# Chapter 6

# Results and Analysis

In this section, the results of the simulation will be discussed. Overall, Newtonian fluid simulation is working and is giving realistic results. Improvements can be done to make it more realistic and more efficient. However, Non-Newtonian fluid simulation is not working as expected. Analysis of the results will be given with each individual example. Used OpenVDB features in Houdini to create these simple surfacing of the particles in some of the examples.

In figure 6.1 and 6.2, the fluid dropped from one side of the simulation box. Causing it to rush to the other side of the wall and try to balance itself within the simulation box. At initial drop, the fluid particles that are located on top hit the bottom boundary as expected while the fluid particles at the lower area continue to flow to the otherside of the box. After the initial hit, the fluid flows inside the simulation box causing a whirlpool-like look which is caused because of the cube shaped box. Weakly compressible fluid behavior can be seen in this example when looked at the first impact towards the bottom and the otherside of the box. Incompressibility algorithms can be implemented to fix this problem in the future. Sphere obstacles are making the fluid behave accordingly. And the behavior of fluid to these obstacles are satisfying. More complex collision detection algorithms for arbitrary objects can be implemented in the future to test the fluid on different shaped obstacles.

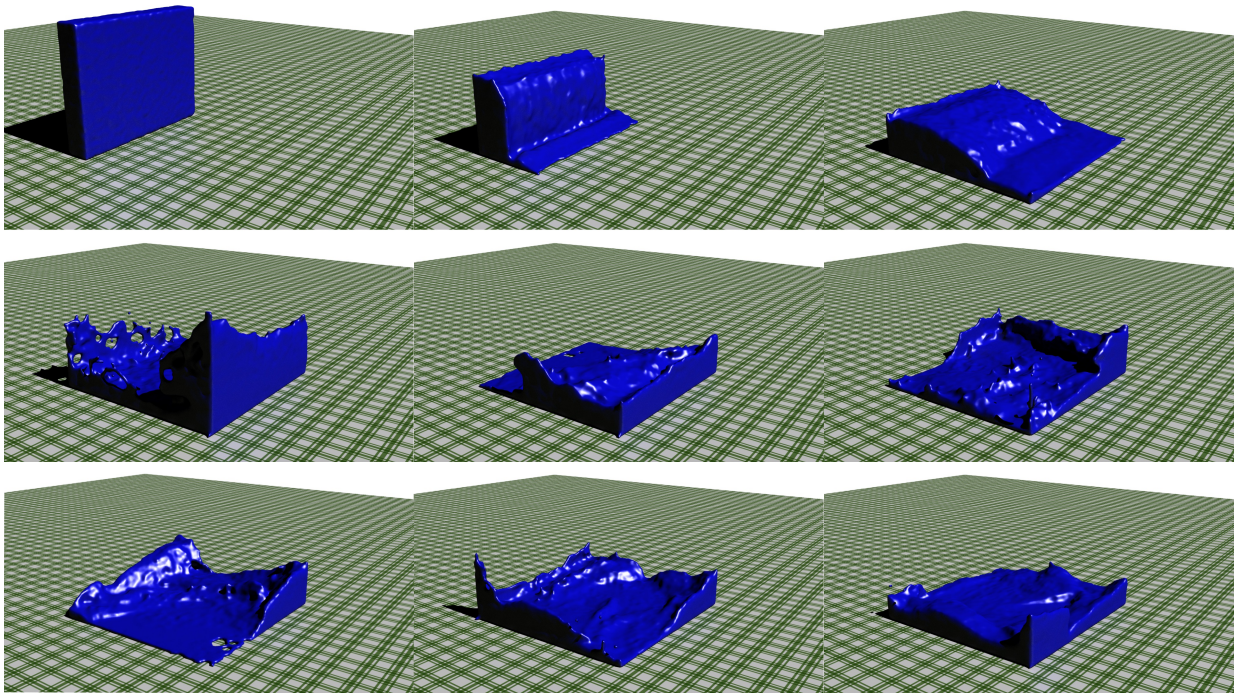


**Figure 6.1:** *Fluid being dropped from one side of the simulation box. Approx. 110k particles.*

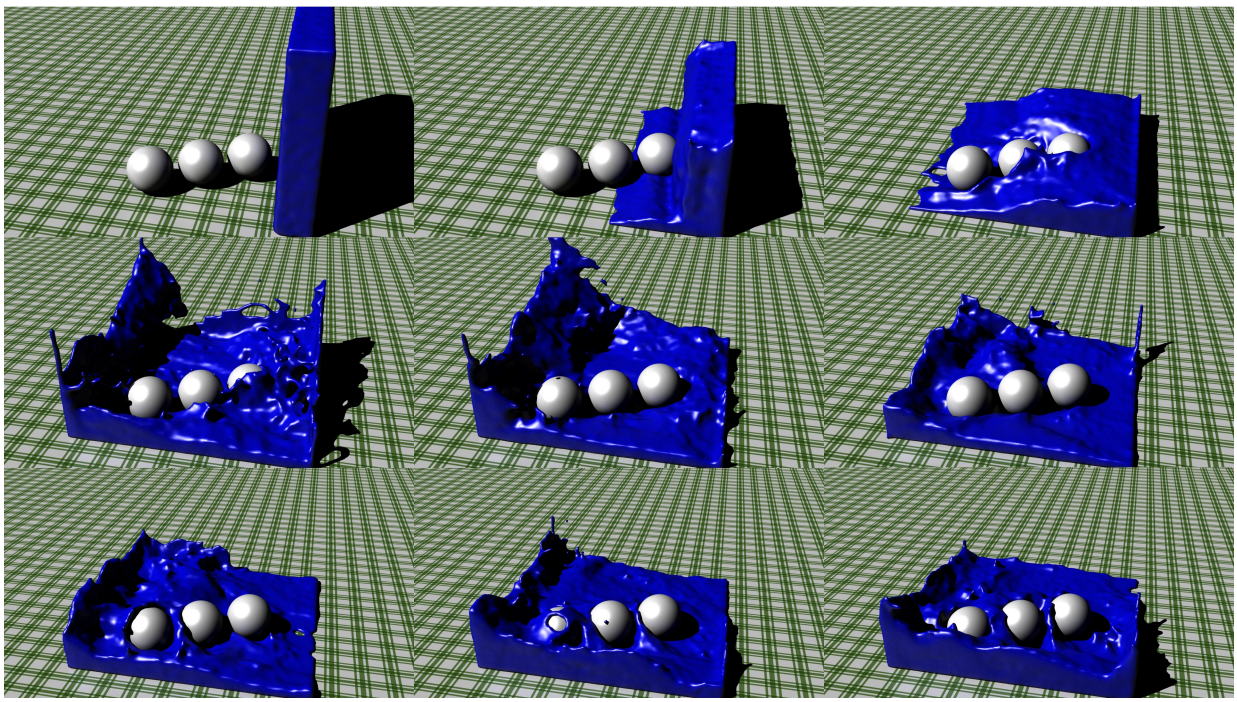


**Figure 6.2:** *Same test using 3 spheres as collision objects. Approx. 110k particles.*

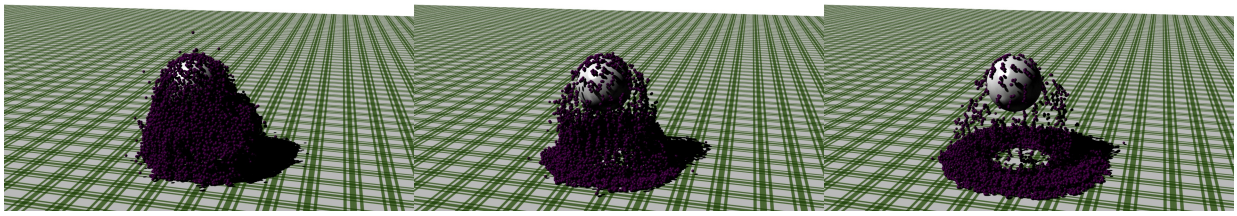


**Figure 6.3:** *Low viscosity fluid dropped on a sphere. Approx. 14k particles.*

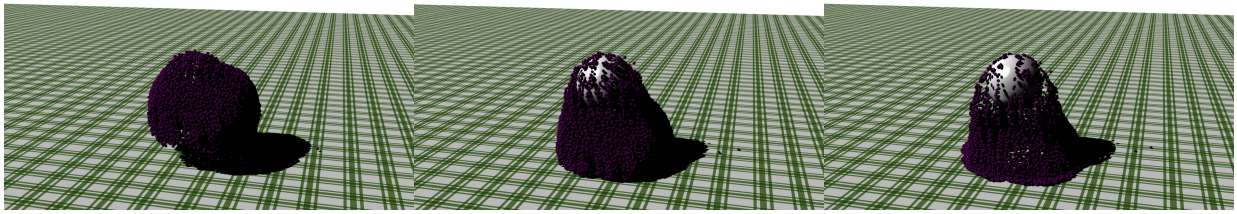


**Figure 6.4:** *Mid viscosity fluid dropped on a sphere. Approx. 110k particles.*

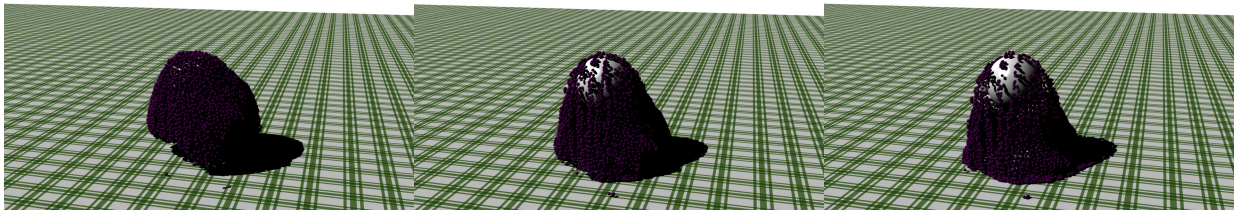


**Figure 6.5:** *High viscosity fluid dropped on a sphere. Approx. 110k particles.*

In figure 6.3, 6.4 and 6.5, sphere shaped particles representing the fluid with different viscosity values were dropped on a sphere to observe the behavior of different viscosity levels after an impact. Low viscosity fluid particles tend to scatter and separate from each other, while higher viscosity fluid particles want to stay together. This example has not surfaced because the particle count was low. High time-step values causes instabilities in this example because of inaccurate velocity values.

In figure 6.6, the fluid dropped as four identical spheres inside the box. Causing them to hit the bottom, splash because of the high pressure and mix. Usage of XSPH velocity correction in this example is the crucial point because it keeps spheres from scatter and helps them maintain their shape until they hit the ground. Incompressibility issues can be seen in

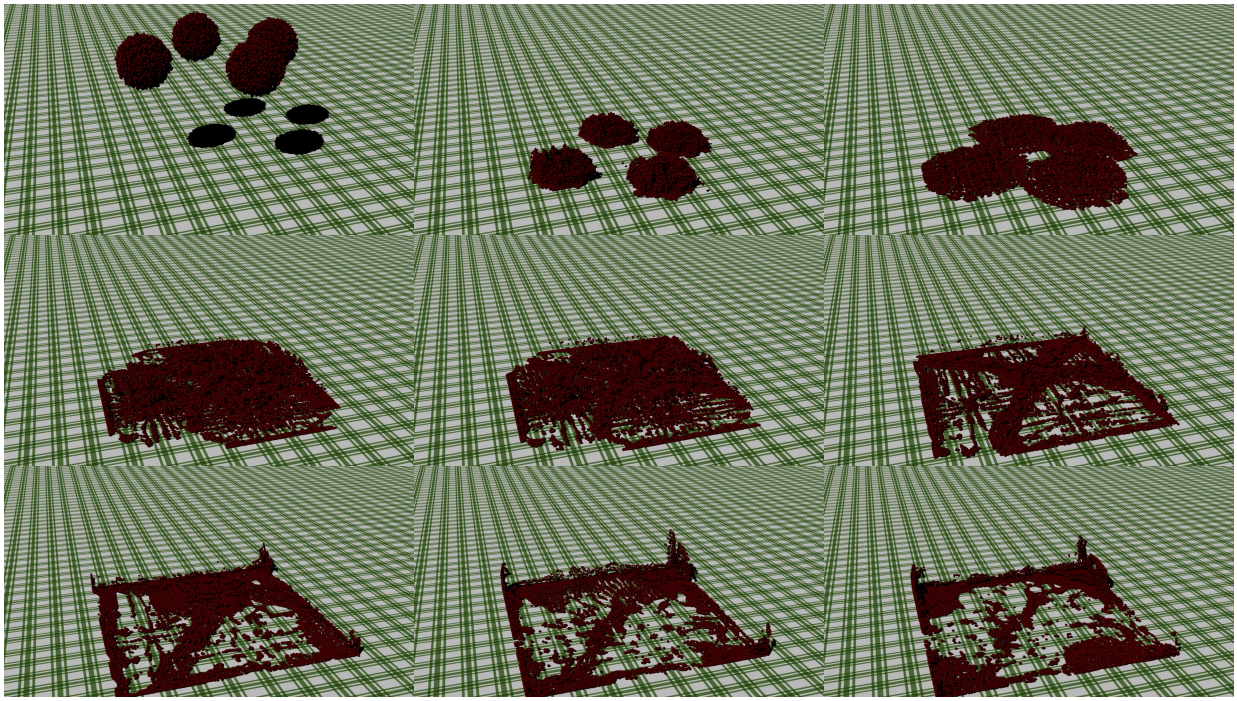


**Figure 6.6:** *Four identical sphere shaped fluids dropped at the same time. Approx. 20k particles.*

this example aswell. Since the particle count wasn't high, surfacing has not applied in this example. However, higher particle count example would expected to give a similar result.

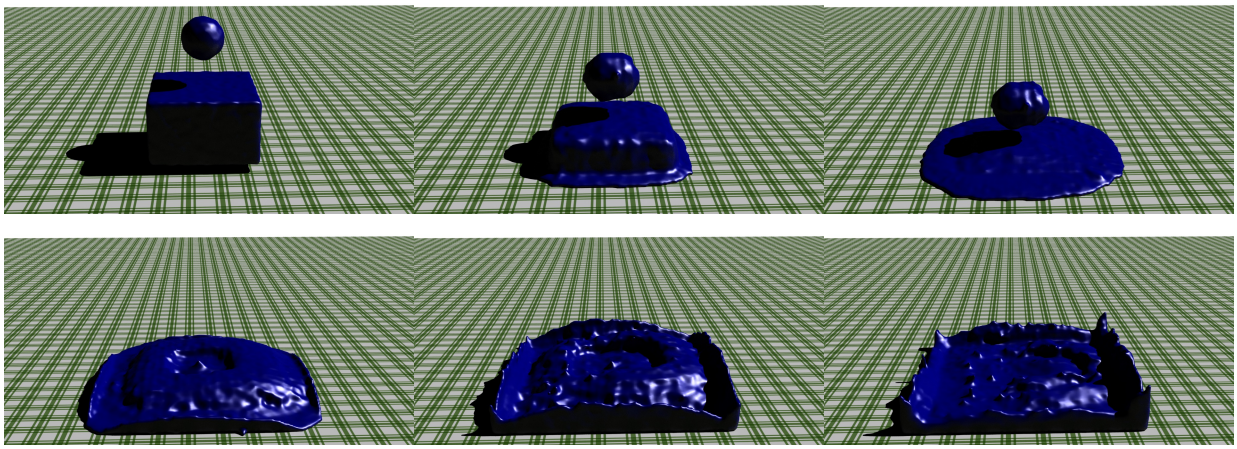


**Figure 6.7:** *A sphere shaped fluid dropped on top of a box shaped fluid. Approx. 100k particles.*

Another example on multiple fluid objects can be seen in figure 6.7. Two fluid objects were dropped on each other and caused the fluid to

splash. Even though the incompressibility issue still continues in this example, the results are pleasing.

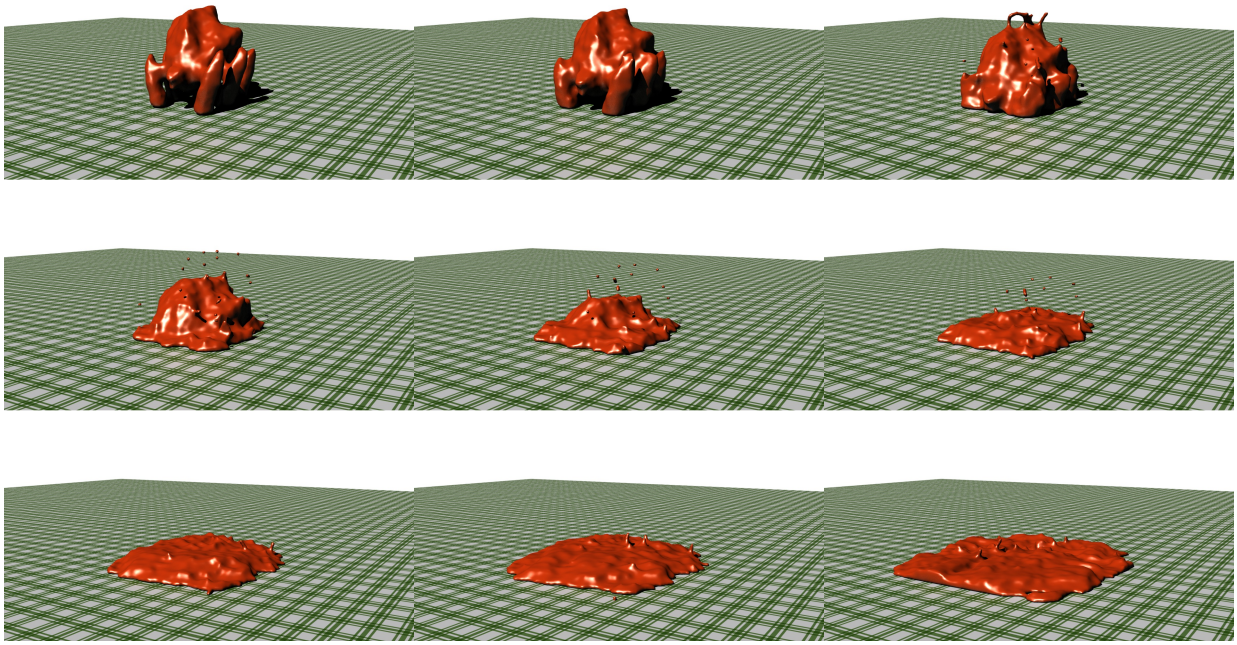


**Figure 6.8:** *Non-Newtonian fluid behavior. Stress tensor not working as expected. Model by Christoffer Stai. Approx. 35k particles.*

Non-Newtonian algorithms are used in this example. Even though high elasticity constant was used in this example, the behavior of the fluid is not how it's expected. The behavior of the fluid should've been more viscous and there shouldn't be particle scattering. Viscosity and stress tensor elements in this example are not working properly. However, it shows that any arbitrary shaped .obj can be used as an input for the fluid.

## 6.1 Known Issues

### 6.1.1 Non-Newtonian

Implementation of Non-Newtonian fluids are not working properly. After investigating and debugging for long hours, it's been found that the

velocity values of the particles are not very accurate. The fluid compressibility is another issue on this topic. Mao and Yang (2005a) used a modified version of Moving Semi-Implicit method to enforce incompressibility. Which would allow particles to have different response to each other and therefore, affecting the whole simulation. The papers that have been followed for the two different types of fluids were using two different default kernel functions. Changing the kernel functions causes completely different scaled density values which causes two different testing conditions for two fluid types.

### 6.1.2 Fluid Incompressibility

As mentioned above, the high pressured areas are running late to push back the compressing particles. This causes a huge issue on reality of the simulation. Several different fluid incompressibility techniques were presented. These techniques will be mentioned in section 7.1.

### 6.1.3 Efficiency

After running several program profiling on the implementation, it's been found that the bottle-neck is the neighbor searching process. Improving the nearest neighbor search (NNS) algorithms gives better and more efficient results. As mentioned in section 4.10, it is observed that Spatial Hashing is more efficient over Hierarchical Tree method.

The smoothing length, $h$ and the number of particles have big influence on the performance. As Teschner *et al.* (2003) stated, grid cell size has the most significant impact on performance. And in this implementation, the smoothing length is set as grid cell size. Which makes it the most crucial parameter regarding the efficiency. The particle count has a direct influence on NNS however the system should be running regardless of the dependency on the particle count. Therefore, multi-threaded implementations or other neighbor searching algorithms can be used to increase the performance.

# Chapter 7

# Conclusion

The aim of this project was to design and implement a fluid simulation using Smoothed Particle Hydrodynamics. It is designed in a way to simulate both Newtonian and Non-Newtonian fluids. Even though these fluids are using different algorithms, their structure is based on same equation defined by Navier-Stokes. SPH is a popular and powerful method to use when simulating fluids however since it was designed for compressible flow problems, the lack of incompressibility is the one major drawback, Kelager (2006).

As explained in chapter 6, results of the Newtonian fluids are looking as expected. Improvements can be done to increase the performance and to have more realistic results. However, unfortunately the simulation of Non-Newtonian fluids are not how they were expected. Particle scattering, low stress tensor values are causing the simulation to look like splashing but highly viscous fluid. XSPH velocity correction was used to maintain the shape of the fluid and prevent it from scattering but using high values in correction parameter would take away from the physical correctness of the the simulation. Other methods like "ghost particles" can be used to prevent the scattering, Schechter and Bridson (2012).

## 7.1 Future Work

This implementation covers the essential features of a fluid simulation due to the limited time. However, it is possible to create more realistic, better looking, faster and more adaptive simulation. Since the aim of this project was to focus on the motion not the visualisation, the future work presented in this section is again focusing on the same but more advanced aspects of this topic.

SPH is a method where each particle is evaluated alone. Therefore, they dont depend on each other. This allows the method to be implemented using parallel programming. Ihmsen *et al.* (2014), Harada *et al.* (2007), Ihmsen *et al.* (2011) and Krog and Elster (2012) created GPU-based fluid simulations. Multi-threaded approach would increase the performance of the simulations significantly.

Another issue on the performance is the time-steps used in the integration schemes. High time-step values creates inaccurate results on velocity and position values which leads to unrealistic simulations. Adaptive time-stepping is an important topic to increase the efficiency. Desbrun and Gascuel (1996) and Paiva *et al.* (2006) proposed a similar solution for this.

As mentioned in chapter 6, this implementation doesn't have incompressibility feature. Premoze *et al.* (2003) proposed a method called Moving Particle Semi-Implicit method which is adapted to SPH by Mao and Yang (2005a) that enforces the fluid incompressibility. He *et al.* (2012) propose a local Poisson SPH approach and Ihmsen *et al.* (2014) recently proposed a pressure projection method to maintain the incompressibility. Adding this feature to the implementation would change the response of each particle and create more realistic results.

There is only a simple collision detection and response algorithm implemented for this project (boundaries and spheres obstacles). Collisions with arbitrary objects, scenes would increase the possibilities to test and help creating better results. Using the same spatial hashing algorithm that's been used for neighbor searching, a collision detection algorithm

can be created, Teschner *et al.* (2003).

Heat equations can be implemented for Non-Newtonian fluid calculations which are proposed by both Paiva *et al.* (2006), Mao and Yang (2005b). Thus, the control of elasticity and viscosity would be more stable and realistic. Oscillations in the velocity field are causing particles to penetrate or repel each other which causes scattering. In order to prevent this, ghost particles which are proposed by Schechter and Bridson (2012) or artifical viscosity proposed by Paiva *et al.* (2006) can be used.

# Bibliography


Boost c++ libraries. 2015-14-08.

Andrade L. F. d. S., Sandim M., Petronetto F., Pagliosa P. and Paiva A., 2014. Sph fluids for viscous jet buckling. In *Proceedings of the 2014 27th SIBGRAPI Conference on Graphics, Patterns and Images*, SIBGRAPI '14, Washington, DC, USA. IEEE Computer Society, 65–72.

B. Jacob G. G. Eigen c++ library. 2015-14-08.

Birdsall C. K. and Langdon A. B., 2005. *Plasma physics via computer simulation*. Series in plasma physics. Taylor & Francis, New York. Originally published: New York ; London : McGraw-Hill, 1985.

Chhabra R. P., 2006. *Bubbles, Drops, and Particles In Non-Newtonian Fluids; 2nd ed.* Taylor and Francis, Abingdon.

Clavet S., Beaudoin P. and Poulin P., 2005. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '05, New York, NY, USA. ACM, 219–228.

de Souza Mendes P. R., Dutra E. S., Siffert J. R. and Naccache M. F., 2007. Gas displacement of viscoplastic liquids in capillary tubes. *Journal of Non-Newtonian Fluid Mechanics*, **145**(1), 30 – 40.

Desbrun M. and Gascuel M.-P., 1995. Animating soft substances with implicit surfaces. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, New York, NY, USA. ACM, 287–290.

Desbrun M. and Gascuel M.-P., 1996. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation '96*, New York, NY, USA. Springer-Verlag New York, Inc., 61–76.

Ellero M., Kröger M. and Hess S., 2002. Viscoelastic flows studied by smoothed particle dynamics. *J. Non-Newtonian Fluid Mech.*, **105**, 35–51.

Foster N. and Fedkiw R., 2001. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, New York, NY, USA. ACM, 23–30.

Foster N. and Metaxas D., 1996. Realistic animation of liquids. *Graphical Models and Image Processing*, **58**(5), 471 – 483.

Gingold R. A. and Monaghan J. J., nov 1977. Smoothed particle hydrodynamics - Theory and application to non-spherical stars. **181**, 375–389.

Goktekin T. G., Bargteil A. W. and O'Brien J. F., 2004. A method for animating viscoelastic fluids. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, New York, NY, USA. ACM, 463–468.

Harada T., Koshizuka S. and Kawaguchi Y., 2007. Smoothed particle hydrodynamics on gpus. *Computer Graphics International*, 63–70.

He X., Liu N., Li S., Wang H. and Wang G., 2012. Local poisson sph for viscous incompressible fluids. *Computer Graphics Forum*, **31**(6), 1948–1958.

Hernquist L. and Katz N. TREESPH - A unification of SPH with the hierarchical tree method.

HyperPhysics . Cohesion and surface tension. `http://hyperphysics.phy-astr.gsu.edu/hbase/surten.html`. Accessed: 2015-10-08.

Ihmsen M., Akinci N., Becker M. and Teschner M., 2011. A Parallel SPH Implementation on Multi-Core CPUs. *Computer Graphics Forum*.

Ihmsen M., Orthmann J., Solenthaler B., Kolb A. and Teschner M., 2014. SPH Fluids in Computer Graphics. In Lefebvre S. and Spagnuolo M., editors, *Eurographics 2014 - State of the Art Reports*. The Eurographics Association.

Kelager M., 2006. Lagrangian fluid dynamics using smoothed particle hydrodynamics.

Krog O. E. and Elster A. C., 2012. Fast gpu-based fluid simulations using sph. In *Proceedings of the 10th International Conference on Applied Parallel and Scientific Computing - Volume 2*, PARA'10, Berlin, Heidelberg. Springer-Verlag, 98–109.

Liu G. R., Liu M. B. and Li S., 2004. smoothed particle hydrodynamics a meshfree method. *Computational Mechanics*, **33**(6), 491–491.

Lucy L. B., dec 1977. A numerical approach to the testing of the fission hypothesis. **82**, 1013–1024.

M. A. Boda A. S. S. R. A. A., P. N. Bhasagi, 2015. Analysis of kinematic viscosity for liquids by varying temperature. **4**.

Macey J., a. Ngl. 2015-14-08.

Macey J., b. Octreeabstract. 2015-12-08.

Mao H. and Yang Y.-h., 2005a. Particle-Based Non-Newtonian Fluid Animation with Heating Effects. Technical report, Department of Computing Science, University of Alberta.

Mao H. and Yang Y.-h., 2005b. Particle-Based Non-Newtonian Fluid Animation with Heating Effects. Technical report, Department of Computing Science, University of Alberta.

MIT , 2011. Marine hydrodynamics.

Monaghan J. J., May 1989. On the problem of penetration in particle methods. *J. Comput. Phys.*, **82**(1), 1–15.

Monaghan J. J., 1992. Smoothed particle hydrodynamics. **30**, 543–574.

Morris J. P., 2000. Simulating surface tension with smoothed particle hy-

drodynamics. *International Journal for Numerical Methods in Fluids*, **33**(3), 333–353.

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

Paiva A., Petronetto F., Lewiner T. and Tavares G., october 2006. Particle-based non-newtonian fluid animation for melting objects. In *Sibgrapi 2006 (XIX Brazilian Symposium on Computer Graphics and Image Processing)*, Manaus, AM. IEEE, 78–85.

Paiva A., Petronetto F., Lewiner T. and Tavares G., April 2009. Particle-based viscoplastic fluid/solid simulation. *Comput. Aided Des.*, **41**(4), 306–314.

Premoze S., Tasdizen T., Bigler J., Lefohn A. and Whitaker R. T., 2003. Particle-based simulation of fluids. *Computer Graphics Forum*, **22**(3), 401–410.

Priscott C., 2010. 3d lagrangian fluid solver using sph approximations.

PukiWiki . `http://www.slis.tsukuba.ac.jp/~fujisawa.makoto.fu/cgi-bin/wiki/index.php?SPH%CB%A1%A4%CE%BD%C5%A4%DF%B4%D8%BF%F4`. Accessed: 2015-10-08.

Raveendran K., Wojtan C. and Turk G., 2011. Hybrid smoothed particle hydrodynamics. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, New York, NY, USA. ACM, 33–42.

Reeves W. T., April 1983. Particle systems—a technique for modeling a class of fuzzy objects. *ACM Trans. Graph.*, **2**(2), 91–108.

Schechter H. and Bridson R., 2012. Ghost sph for animating water. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2012)*, **31**(4).

Stam J., 1999. Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH

'99, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co., 121–128.

Teschner M., Heidelberger B., Müller M., Pomeranets D. and Gross M., 2003. Optimized spatial hashing for collision detection of deformable objects. *Proceedings of Vision, Modeling, Visualization VMV03*, 47–54.

# Appendix A

# Appendix

## A.1 HDA User Guide

As mentioned in chapter 5, this implementation has a Houdini Digital Asset extension. This digital asset is used to help create both input and output files. This guide will go over a quick look into this asset. The digital asset consists of three sections: input generator, simulation visualisation and cache out.

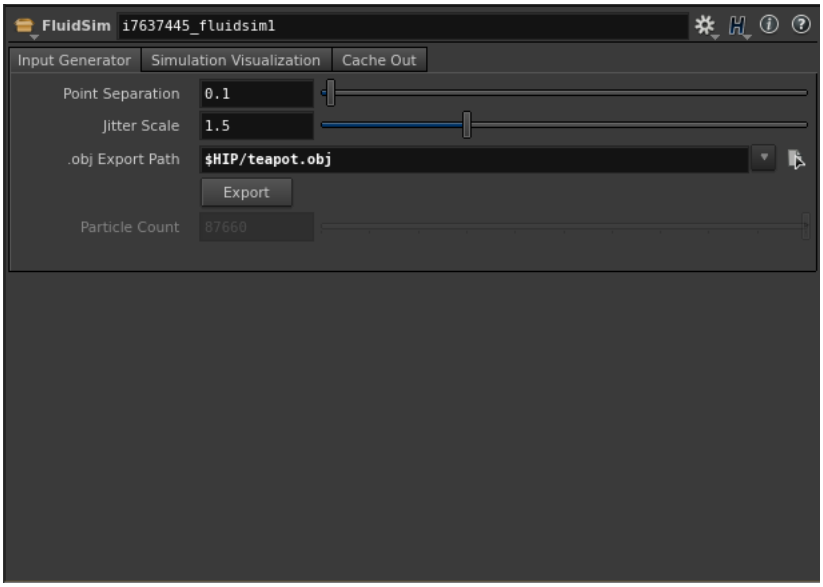



**Figure A.1:** *Input Generator Tab in HDA*

In this tab, user can upload an arbitrary .obj file as object to converted

into particles. The geometry loaded through connecting the related node to the asset' input node. Point separation will change the distance between neighboring particles. Jitter scale feature changes the placing of the particles. By looking at the particle count, user can track the amount of particles that are generated through the process. By pressing the Export button, particles will be exported as an .obj file relative to the given path.

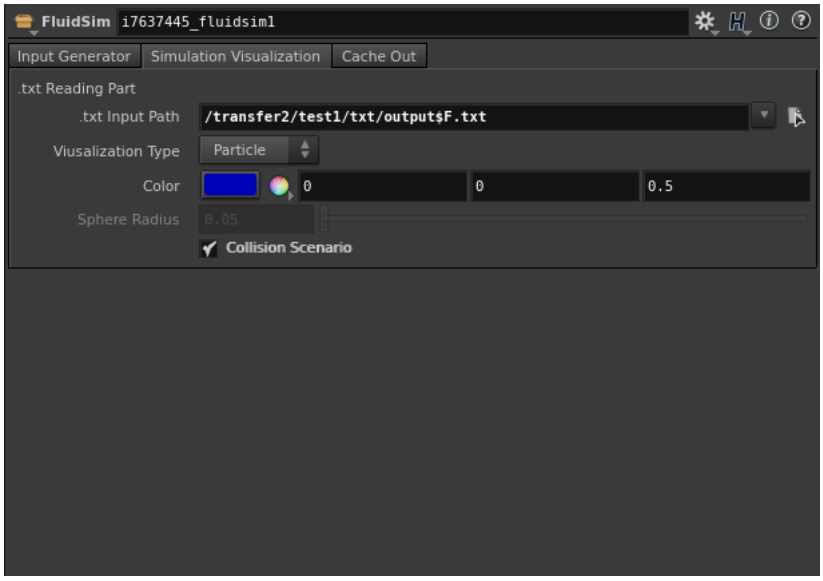



**Figure A.2:** *Simulation Visualisation Tab in HDA*

User can visualise the outputs generated by the stand alone application using this tab. First, the input .txt file sequence must be inserted to file path. Then the visualisation type can be selected between particles, spheres (replacing the particles) and surface (using OpenVDB feature in Houdini for this). Collision scenario is to visualise the spheres that created for collision in the implementation.

User can export the scene as a .bgeo sequence using this tab. It's using a ROP Solver to render out the data.

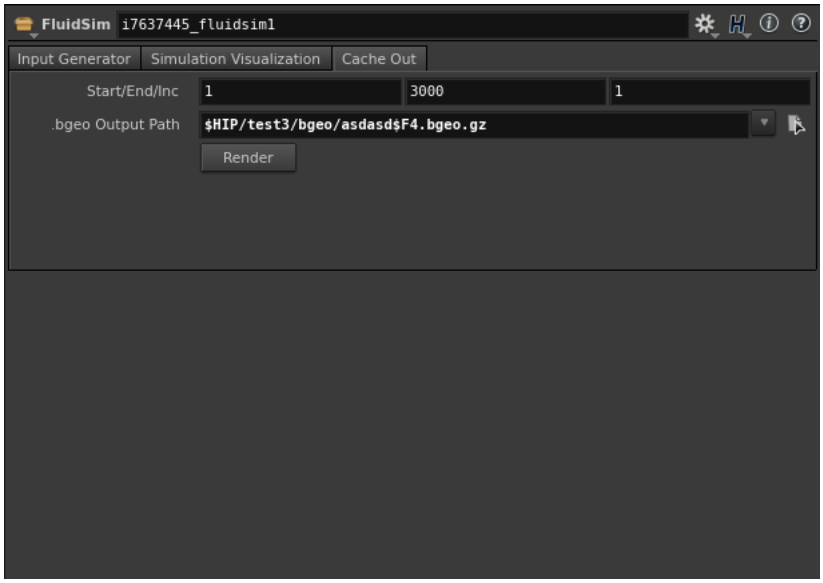



**Figure A.3:** *Cache Out Tab in HDA*