Position Based Fluid

MING-YEN KUO

Computer Animation and Visual Effects
Bournemouth University

August 2016
Abstract

Smoothed Particle Hydrodynamics (SPH) is a well known particle-based fluid simulation method, and Position Based Dynamic (PBD) is a strategy working on the vertices position data with unconditional stable feature. Position based fluid is a method combined with these two ideas, and using SPH neighbors concept working on PBD position calculation. This paper explains all the implementation details based on position based fluid.
# Table of Contents

Abstract i

List of Figures v

List of Tables 1

1 Introduction 2

2 Previous Work 3

3 Fluid with Position Based Dynamic 5
   3.1 Navier-Stoke Equation ........................................ 5
   3.2 Particle and Grid Based Methods ............................ 6
       3.2.1 Eulerian Method ........................................ 6
       3.2.2 Lagrangian Method .................................... 7
   3.3 Smoothed Particle Hydrodynamics .......................... 7
   3.4 Position Based Dynamic ..................................... 9
   3.5 Position Based Fluid ....................................... 11

4 Implementation 13
   4.1 Design ..................................................... 13
   4.2 Neighbors Search .......................................... 16
       4.2.1 Spatial Hashing ....................................... 16
       4.2.2 Neighbors Queries .................................... 17
# List of Figures

3.1 One cell from 3D MAC grid .......................... 6  
3.2 Water in glass with rotational force field .............. 7  
4.1 Class diagram - 1 .................................. 14  
4.2 Class diagram - 2 .................................. 15  
4.3 Kernel function effects in 3D space ..................... 18  
4.4 Poly6 kernel graph ................................ 19  
4.5 Spiky kernel graph ................................ 19  
4.6 Influence from artificial pressure term in position based fluid ................................. 22  
4.7 Influence from vorticity in position based fluid .............. 23  
5.1 Position based fluid simulator ....................... 26  
6.1 Result 1: Single Dam Break ............................ 28  
6.2 Result 2: Dam Break ................................ 28  
6.3 Result 3: Transparent Dam Break .................... 29  
6.4 Result: Bunny Drop .................................. 29  
A.1 PBF XML file layout ................................ 31  
B.1 PBF Initial HDA: Initial Value ......................... 33  
B.2 PBF Initial HDA: Particles Setup ................. 33  
B.3 PBF Rendering HDA ................................. 34
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Advantages and Disadvantages with particle-based and grid-based in fluid simulation</td>
<td>7</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The field of computational fluid dynamics has been developed in computer graphics for several years, from fire, smoke, gas, water, wave, high viscosity liquid, melting, multi-phases fluid, from simple to complex fluid simulation, and these techniques also have been designed and improved by numerous experts in physics, mathematical different background people. Looking for realistic and efficient simulation is always the purpose for further researching.

Smoothed Particle Hydrodynamics (SPH) is one well known particle-based fluid simulation method. It is useful in games because no need to fit the fluid simulation size within grid in advance. Moreover, the concept is based on particle system. Comparing to grid-based method, the idea and implementation knowledge are usually easily understood. However, SPH has one issue that once the particle has not enough neighbors data, it causes the density incorrect then negative pressure. This would lead to the simulation result not stable.

Position based dynamic (PBD) originally used for deformable object. Macklin and Müller [2013] propose a different fluid simulation framework based on SPH and PBD, and he proved that the incompressible flow could be simulated inside PBD. One characteristic in PBD is unconditional stable, and this would improve the problem in SPH.

The next following chapter would introduce the related work in the recent years, and the fluid and position based dynamic theory are in chapter 3. Main implementation work is explained in chapter 4, and the simulation pipeline is in chapter 5. The last ones are results and conclusion with future work.
Chapter 2

Previous Work

"Computational Fluid Dynamics has a long time history. In 1822 Claude Navier and in 1845 George Stokes formulated the famous Navier-Stokes Equation” from Miller et al. [2003]. The recent fluid simulation survey by Bridson [2015] and Bridson and Müller-Fischer [2007] give us a brief introduction in fluid dynamics.

Smoothed Particle Hydrodynamics (SPH) was first introduced by Lucy [1977]. Since Reeves [1983] designed the particle system, and it was highly been used in physics dynamic in computer graphics. SPH with particle system has become one of important simulation method in fluid dynamic. Later on, many researchers started to work on and improve the drawback of particle-based method. Miller et al. [2003] already improve it into real time fluid simulation interactively in games. The traditional SPH and weakly compressible SPH Becker and Teschner [2007] both require the stiff equation due to incompressibility issue. Predictive-corrective incompressible SPH Solenthaler and Pajarola [2009] used Jacobi method to accumulate pressure changes and apply forces until convergence for stability, and this also solved the limit of time step size.

Position based dynamic was been found that has been used in the last decades. For example, Provot [1995] and Desbrun et al. both use the constraints idea in mass spring system to prevent over-stretching problem. It is a strategy to limit the stretch force too large, not like the pure position based solution as a function in calculation. Jakobsen [2001] designed his physics engine on position based called "Fysix”, however, it only focused on distance constraint by manipulating position directly. Müller et al. [2005] proposed a solution working on deformable objects by moving vertices to a specific position satisfied the state condition. Clavet et al. [2005] used position based dynamic for simulating viscoelastic fluid, however, their approach is
conditional stable because of the time-step integration. Until Müller et al. [2007], he organized and built position based dynamic into a structural framework.

In the recent year, Macklin et al. [2014] modified the fluid constraint function from equality to inequality, and the purpose is to clamp the density constraint to be non-negative. So this would influence the separate particles. Alduán et al. [2015] also design a fluid simulator for VFX production pipelines based on position based fluid, and it is efficient, robust and scalable.

In this project, we mainly focus on Macklin and Müller [2013] for the fluid dynamic calculation and Miller et al. [2003] for kernel function design.
Chapter 3

Fluid with Position Based Dynamic

This chapter shows the mathematical fluid theory knowledge combined with position based dynamic.

3.1 Navier-Stoke Equation

When speaking about fluid movement, it always reminds us about the physics background knowledge. In [Bridson 2015], it introduced the derivation from Newtons equation: \( \vec{F} = m\vec{a} \) to Navier-Stoke equation in (3.1). It mentions that the forces cause the fluid flowing mainly because of internal forces: like pressure, viscosity; external force: gravity. Pressure\((-V\nabla p)\), viscosity\((V\mu\nabla \cdot \nabla \vec{u})\) and gravity\((mg)\) in (3.3) are original version of Navier-Stoke equation. These three terms are the values that we have to compute when doing fluid simulation in computer graphics.

\[
\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} - \frac{1}{\rho} \nabla p = \vec{F} + \nu \nabla \cdot \nabla \vec{u} \tag{3.1}
\]

\[
\nabla \cdot \vec{u} = 0 \tag{3.2}
\]

\[
m \frac{D\vec{u}}{Dt} = m\vec{g} - V\nabla p + V\mu \nabla \cdot \nabla \vec{u} \tag{3.3}
\]

After knowing the basic physics calculation in fluid, how is it work in the real simulation? According to [Bridson 2015], Algorithm I gives us the idea of simulation structure. The flowing
fluid is a continuous sequence of movement by time. In this algorithm, every iteration accumulate attribute values in fluid. For instance, forces cause velocity updating, and velocity causes position updating. This is also called *advection*.

**Algorithm 1** Navier-Stokes algorithm

1: Initial velocity field \( \vec{v}^{(0)} \)
2: for time step \( n = 0, 1, 2, \ldots \) do
3:   Determine \( \Delta t \) from \( t_n \) to \( t_{n+1} \)
4:   Set \( \vec{v}^A = \text{advect}(\vec{v}^n, \Delta t, \vec{v}^n) \)
5:   Add \( \vec{v}^B = \vec{v}^A + \Delta t \vec{g} \)
6:   Set \( \vec{v}^{n+1} = \text{project}(\Delta t, \vec{v}^B) \)
7: end for

### 3.2 Particle and Grid Based Methods

To compute those values in fluid, there are two different methods based on different mathematical model. One is called *Eulerian, Grid-Based*, while the other is named *Lagrangian, Particle-Based*.

#### 3.2.1 Eulerian Method

In Eulerian method, fluid is fix in a limited volume in space. In Figure 3.1, it is an individual cell in 3D grid. The \( u \) means velocity in six direction, \( p \) is the pressure, and \( \delta h \) is the cell size. This is called *MAC grid, (Marker-and-Cell), staggered grid*. Each grid has the velocity and pressure attribute values. All the calculation is based on those cells. In other words, fluid simulation in grid based method is limited by the grid size.

![Figure 3.1: One cell from 3D MAC grid.](image)
### 3.3. SMOOTHED PARTICLE HYDRODYNAMICS

<table>
<thead>
<tr>
<th></th>
<th>Particle-Based</th>
<th>Grid-Based</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantages</td>
<td>Could be used in multiphases simulation.</td>
<td>Higher numerical accuracy.</td>
</tr>
<tr>
<td>Disadvantages</td>
<td>Need more particles for realistic result. Hard to track fluid surface.</td>
<td>Simulation is fixed by grid size.</td>
</tr>
</tbody>
</table>

**Table 3.1**: Advantages and Disadvantages with particle-based and grid-based in fluid simulation

#### 3.2.2 Lagrangian Method

On the other hand, in Lagrangian method, the fluid is formed by tiny individual element, usually called *particle*. As a result, it could be said that Lagrangian fluid simulation is a special case in a particle system. Each particle exists with its own values, such as pressure, velocity, and position.

Figure 3.2 shows the simulation result in [Müller et al. 2003], full of particles in container and the rendering image.

![Water in glass with rotational force field](image)

**Figure 3.2**: Water in glass with rotational force field [Müller et al. 2003]

Grid-based and particle-based are both have pros and cons in different perspectives, and this would leave to user to choose depends on the situation in simulation. In table [3.1](#), listed a basic idea of comparison between these two methods.

#### 3.3 Smoothed Particle Hydrodynamics

Smoothed Particle Hydrodynamics (SPH) is an interpolation method used for quantifying the value relation between the particles with their neighbors strategy. In equation (3.4), it is the core meaning in SPH. \( A(r) \) is a scalar quantity at location \( r \) by a weighted sum of contributions from other particles filter by \( W(r, h) \) smoothing kernel with radius \( h \). The \( h \) is usually called
smoothing length or kernel size, and this value is mainly used for adjusting the particles neighbors number. \( m_j \) is the particle mass, \( \rho_j \) is the density, and \( A_j \) is field quantity at \( \mathbf{r}_j \). Equation (3.5) and (3.6) is the gradient and Laplacian representation respectively. They would usually be used base on different type of kernel function. More detail will be explained in 4.3.

\[
A_S(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h) \tag{3.4}
\]

\[
\nabla A_S(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(\mathbf{r} - \mathbf{r}_j, h) \tag{3.5}
\]

\[
\nabla^2 A_S(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(\mathbf{r} - \mathbf{r}_j, h) \tag{3.6}
\]

Algorithm 2 shows the basic iteration of SPH fluid simulation. Firstly, find the nearby neighbors by kernel size \( h \), secondly, use kernel function \( W(\mathbf{r}, h) \) to calculate the local density. Then compute the pressure by density. High density area causes high pressure, and this become the force on particles. Lastly, accumulating pressure, viscosity, and external force together. Updating particles position and velocity. [Ihmsen et al.][2014]

**Algorithm 2 SPH Algorithm**

1. for all particle \( i \) do
2. find neighbors \( j \)
3. end for
4. for all particle \( i \) do
5. \( \rho_i = \sum_j m_j W_{ij} \)
6. compute \( p_i \) using \( \rho_i \)
7. end for
8. for all particle \( i \) do
9. \( F_{i \text{ pressure}} = -\frac{m_i}{\rho_i} \nabla p_i \)
10. \( F_{i \text{ viscosity}} = m_i \nu \nabla^2 \mathbf{v}_i \)
11. \( F_{i \text{ other}} = m_i \mathbf{g} \)
12. \( F_i(t) = F_{i \text{ pressure}} + F_{i \text{ viscosity}} + F_{i \text{ other}} \)
13. end for
14. for all particle \( i \) do
15. \( \mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \Delta t F_i(t)/m_i \)
16. \( \mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \Delta t \mathbf{v}_i(t + \Delta t) \)
17. end for
3.4 Position Based Dynamic

Position based dynamic (PBD) is officially introduced by [Müller et al., 2007], and main idea is not using the traditional physics dynamic. External or internal forces cause the acceleration, then comes to updating the velocity by time, following the position. The acceleration and velocity is accumulated by time. On the other hand, PBD only apply the force at the first time for each step of simulation with the initial force value. Next, using the predicted position and the old position to calculate the relative velocity fit into the constraint function. From [Müller et al., 2007], he mentions that the advantages of PBD is the stability and controllability by user. Compared with the traditional physics dynamic, sometimes it would be difficult to remain the expected result. This characteristic becomes important in some area, like computer game, as it is ideally to control the expected result even plausible instead of looking for accuracy result.

Furthermore, from Bender et al. [2015] There are many different types of constraint in position based dynamic used in different conditions, and fluid is one of them.

- Stretching
- Bending
- Isometric Bending
- Collisions
- Volume Conservation
- Long Range Attachments
- Strands
- Continuous Materials
- Rigid Body Dynamics
- Fluids
- Shape Matching

There are two types of constraint, one is called equality constraint in (3.7), while the other is called inequality in (3.8). All type of constraints will belong to either equality or inequality.

\[
C_j(x_{i_1}, ..., x_{i_n}) = 0 \tag{3.7}
\]

\[
C_j(x_{i_1}, ..., x_{i_n}) \geq 0 \tag{3.8}
\]
Main issue in PBD is moving a set of points to satisfy the constraint function, and the most important condition is the linear and angular momentum should be conserved. In equation 3.9, it is the condition should satisfied:

$$\sum_i m_i \Delta p_i = 0$$  \hspace{1cm} (3.9)

Because the momenta should be conserved, according to Müller et al. [2007], we need to find $\Delta p$ such that $C(p + \Delta p) = 0$, in equation 3.10.

$$C(p + \Delta p) \approx C(p) + \nabla_p C(p) \cdot \Delta p = 0$$  \hspace{1cm} (3.10)

In physic theory, to satisfy equation 3.10, and it means that the $\Delta p$ should be kept in the same direction as $\nabla_p C(p)$. Hence, it exists a scalar value $\lambda$ such that:

$$\Delta p = \lambda \nabla_p C(p)$$  \hspace{1cm} (3.11)

Substituting equation 3.11 into equation 3.10 solving $\lambda$ and substituting it back to equation 3.11 $\Delta p$ becomes as in equation 3.12

$$\Delta p = \left( \frac{C(p)}{\nabla_p C(p)} \right)^2 \nabla_p C(p)$$  \hspace{1cm} (3.12)

Equation 3.12 could be simplified by scalar value $s$ into equation 3.13. Now the value $s$ is the scaling factor for position correction, shown in equation 3.14. This is also the most important part in PBD.

$$\Delta p_i = -s \nabla_p C(p_1, \ldots, p_n)$$ \hspace{1cm} (3.13)

$$s = \frac{C(p_1, \ldots, p_n)}{\sum_j \left( \nabla_{p_j} C(p_1, \ldots, p_n) \right)^2}$$  \hspace{1cm} (3.14)
3.5. POSITION BASED FLUID

Above are the general PBD should be computed equations, and it assumes that all the points are with same mass. For position based fluid, each particle we assume that they are with same mass. If it becomes to more complex geometry, there are some weighted scalar should be considered. For more detail about different mass PBD please reference [Müller et al., 2007].

Algorithm 3 is the general position based dynamic algorithm. The core computation in PBD algorithm is shown in line (10), line (15)-(17), and line (19)-(20). The predicted new position in line (10) is calculated by Euler integration step. From line (15) to (17), it is the iteration for estimating new position satisfied the constraint function. Then points position could be optimized and the velocity could be updated in line (19) to (20).

Algorithm 3 PBD Algorithm

1: for all vertices i do
2: initialize \( x_i = x_i^0, v_i = v_i^0, w_i = \frac{1}{m_i} \)
3: end for
4: for Simulation do
5: for all vertices i do
6: \( v_i \leftarrow v_i + \Delta t w_i f_{ext}(x_i) \)
7: end for
8: dampVelocities(\( v_1, ..., v_N \))
9: for all vertices i do
10: \( p_i \leftarrow x_i + \Delta t v_i \)
11: end for
12: for all vertices i do
13: generateCollisionConstraints (\( x_i \rightarrow p_i \))
14: end for
15: for SolverIteration do
16: projectConstraints(\( C_1, ..., C_{M+M_{coll}}, p_1, ..., p_N \))
17: end for
18: for all vertices i do
19: \( v_i \leftarrow (p_i - x_i)/\Delta t \)
20: \( x_i \leftarrow p_i \)
21: end for
22: velocityUpdate(\( v_1, ..., v_N \))
23: end for

3.5 Position Based Fluid

From the last section, we know that the essence of PBD is to find the vertices new location and fit into the constraint conditions. Fluid constraint is one of the constraint in PBD. Luckily, in
position based fluid, there is only one constraint that we have to deal with: density constraint. In equation 3.15, it is the fluid density constraint function. It is not necessary to calculate pressure, viscosity, etc in SPH. Moreover, the collision constraint is ignored in position based fluid. In the other words, collision detection could be covered by the original SPH boundary condition method.

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

(3.15)

In algorithm 4 PBF algorithm, it would look familiar now because it is a combination of SPH and PBD algorithm. From line (1)-(4), to initial velocity and position for each particle by default external force. Same as SPH in line(5)-(7), neighbors searching should be computed before the real simulation calculation. The most important part in PBF is from line(8) to line(19) based on the neighbor particles, as each iteration would calculate the \( \Delta p \). Using \( \Delta p \) to get new position, then update the correct velocity. This will be explained step by step in chapter 4.

**Algorithm 4 PBF Simulation Loop**

1: for all particle \( i \) do
2: apply forces \( v_i \leftarrow v_i + \Delta t f_{ext}(x_i) \)
3: predict position \( x_i \leftarrow x_i + \Delta t x_i \)
4: end for
5: for all particle \( i \) do
6: find neighboring particles \( N_i(x_i^*) \)
7: end for
8: while \( \text{iter} < \text{solverIteration} \) do
9: for all particle \( i \) do
10: calculate \( \lambda_i \)
11: end for
12: for all particle \( i \) do
13: calculate \( \Delta p_i \)
14: perform collision detection and response
15: end for
16: for all particle \( i \) do
17: update position \( x_i^* \leftarrow x_i^* + \Delta p_i \)
18: end for
19: end while
20: for all particle \( i \) do
21: update velocity \( v_i \leftarrow \frac{1}{\Delta t} (x_i^* - x_i) \)
22: apply vorticity confinement and XSPH viscosity
23: update position \( x_i \leftarrow x_i^* \)
24: end for
Chapter 4

Implementation

In chapter 4, we will demonstrate how to implement position based fluid based on [Macklin and Müller, 2013].

4.1 Design

In this project, considering about whole pipeline for user, it is necessary to design a quick importing and exporting framework. Flexibility and efficiency has become the main direction in this project, and it is also inspired by [Ertekin, 2015], [Rajiv, 2011] and [Priscott, 2010].

There are some requirements we need in this project, as listed in the following:

- User friendly interface to modify simulation parameters.
- A wide range of flexibility of test model for simulation.
- Simulation could be the interaction of multiple fluids.
- Easy to estimate the final simulation result.

Class diagram are shown in Figure 4.1 and Figure 4.2. MainWindow class in charge of all GUI work, including parse XML file, load .obj file. After loading and setting done all the initial data then the Draw class could start visualization on screen. Class Simulation take the responsibility of the whole simulation iteration, and in Figure 4.2 it shows the relation between other classes. Position based fluid calculations are in class PBD. As for class Definition, it stores all global variables from the XML file parsed by MainWindow.
Figure 4.1: Class diagram 1.
Figure 4.2: Class diagram 2.
4.2 Neighbors Search

In particle-based fluid simulation, each iteration is based on the particle and its neighbors, as a result, this becomes the first bottleneck in simulation. In the trivial way is comparing the distance for each particle, so the time complexity will be $O(n^2)$. Obviously, this is not efficient enough for particle-based fluid simulation. How to find neighbors efficiently has become another topic for many experts. For instance, Onderik and Durikovic [2008] compared different methods between spatial hashing and cell indexing while Domnguez et al. [2011] made an experiment between cell-linked list (CLL) and verlet list (VL) methods.

According to Ihmsen et al. [2014], there are five different methods: Uniform Grid, Index Sort, Z-index Sort, Hashing, and Compact Hashing for neighbors searching in SPH. In this project, we use spatial hashing. From equation (4.1) to equation (4.4) all are the calculation process for spatial hashing. At the first look, it would look complicated. Conkerjo provided a very good explanation for better understanding spatial hashing. Here, the neighbors search is implemented and referencing from Kelager [2006] explanation.

4.2.1 Spatial Hashing

Spatial Hash is a mapping from a point in 3D space into a 1D hash index key, and it is defined in equation (4.1):

$$\text{hash}(\hat{x}) = (\hat{x}_x p_1 \text{ xor } \hat{x}_y p_2 \text{ xor } \hat{x}_z p_3) \mod n_H$$ (4.1)

Where $n_H$ is the hash table size, and $p_1 = 73, 856, 093$, $p_2 = 19, 349, 663$, $p_3 = 83, 492, 791$ in equation (4.1) and

$$\hat{x}(x) = \left(\lfloor x_x/l \rfloor, \lfloor x_y/l \rfloor, \lfloor x_z/l \rfloor\right)^T$$ (4.2)

is the point in 3D space after discretization, divided by $l$. As for the value for $l$, because of $W(r - r_j, h) = 0$ if $\|r - r_j\| > h$, thus

$$l = h$$ (4.3)
4.3 Smoothing Kernel Function

In equation 4.4, \( \text{prime}(x) \) is the function return the next prime \( \geq x \), and \( n \) is the number of particle total amount.

\[
n_H = \text{prime}(2n)
\]  (4.4)

4.2.2 Neighbors Queries

After building the hash table, each particle is mapping with a key value. Then need to use kernel function for filtering those particles distance larger than the kernel size, and it means that the distance less than the kernel size particles belongs to its neighbors. Here, we use a simple algorithm run through the nearby cell unit, and each cell unit with its own position. Cell position \( \hat{x}(x) \) is divided by \( l \) of point position, as shown in equation 4.2. Running through a \( 3 \times 3 \times 3 \) grid in 3D (\( 3 \times 3 \) in 2D case). This part is implemented in class NeighborsSearch.

Algorithm 5 Finding Neighbors Algorithm

1: for all particle \( i \) do
2:     Calculation cellPos
3:     for intx = -1; x <= 1; x ++ do
4:         for inty = -1; y <= 1; y ++ do
5:             for intz = -1; z <= 1; z ++ do
6:                 nearPos.x = cellPos.x + x
7:                 nearPos.y = cellPos.y + y
8:                 nearPos.z = cellPos.z + z
9:                 get particle list - L from table
10:                for all particle \( j \) in L do
11:                    Compare distance
12:                end for
13:            end for
14:        end for
15:    end for
16: end for

4.3 Smoothing Kernel Function

Smoothing kernel functions are used for gathering the particle neighbors based on kernel size \( h \) comparing with the distance between two particles. It could be one of the most important part in particle based fluid. The idea of kernel function could be visualized in Figure 4.3.
According to PukiWiki, this gives us a brief concept of all kind of kernel functions, listed in the following:

- Poly6
- Spiky
- Viscosity
- Spline
- Super Gaussian
- Fourth-order weighting

Different kernel functions were designed for different purposes. According to Macklin and Müller [2013], poly6 used for calculating density estimation, while spiky is used for the gradient calculation. As a result, in this project, we only implement these two kernel functions. It is implemented in class Kernel. For both poly6 and spiky, the input data is a vector difference by two particles position and the kernel size $h$. The difference of poly6 and spiky effects is shown in Figure 4.4 and Figure 4.5.

### 4.3.1 Poly6

Poly6 kernel function is used for calculating density, and it is also called 6th degree polynomial kernel. Defined in equation 4.5.
4.3. SMOOTHING KERNEL FUNCTION

\[ W_{\text{poly}6}(r, h) = \frac{315}{64\pi h^6} \begin{cases} 
(h^2 - r^2)^3 & 0 \leq r \leq h \\
0 & \text{otherwise}
\end{cases} \quad (4.5) \]

Figure 4.4: Poly6 kernel graph. From PukiWiki

4.3.2 Spiky

In SPH, spiky is designed for solving the particle clustering problem, calculating pressure force. However, in position based fluid, it is not necessary to compute it. Gradient kernel was used when calculating the gradient constraint function and vorticity confinement. Gradient spiky kernel function is defined in equation 4.6:

\[ \nabla W_{\text{spiky}}(r, h) = -\frac{45}{\pi h^6} \begin{cases} 
(h - r)^2 & 0 \leq r \leq h \\
0 & \text{otherwise}
\end{cases} \quad (4.6) \]

Figure 4.5: Spiky kernel graph. From PukiWiki
CHAPTER 4. IMPLEMENTATION

4.4 Mass

In particle-based fluid, each particle holds its own mass value. In this project, we assume that each particle with the same mass value, and it is a user defined constant. Although the mass is the same for each particle, the density is different according to the surrounding neighbors. Equation 4.7 is from Kelager [2006], it shows the relation between particle size with mass. $\rho_0$ is the rest density, $V$ is the volume of particle, $n$ is the particle count. Kelager [2006] mentions about the mass value should be calculated automatically in stead of user defined due to the unbalance initial value would cause unstable result. However, mass value still leave for the flexibility for user in this project.

\[ \rho_0 = \frac{m}{V^n} \]  

(4.7)

4.5 Density

Density would change during the simulation time depends on the neighbors data at each time step. From Macklin and Müller [2013], use equation 4.8 for calculating the density with poly6 kernel function is the first step in the iteration solver. Each particle owns its density value, and density should be used in the calculation of density constraint function, as mentioned in the last chapter. Because this function is really important, the equation repeated again in equation 4.9

\[ \rho_i = \sum_j m_j W(p_i - p_j, h) \]  

(4.8)

\[ C_i(p_1, ..., p_n) = \frac{\rho_i}{\rho_0} - 1 \]  

(4.9)

4.6 Constraints

The most important part in position based fluid is the constraints calculation. In PBD, calculating $\Delta p_i$ is the key value to correct the position. As introduced in Section 3.4, equation 3.11, to get the $\Delta p_i$ value, and it must solves the gradient constraint function. Equation 4.10 is the gradient of equation 4.9 with respect to a particle $k$.

\[ \frac{\partial C_i}{\partial p_k} = \frac{\partial}{\partial p_k} \left( \frac{\rho_i}{\rho_0} - 1 \right) \]  

(4.10)
### 4.6. CONSTRAINTS

\[ \nabla p_k C_i = \frac{1}{\rho_0} \sum_j \nabla p_k W(p_i - p_j, h) \]  \hspace{1cm} (4.10)

There is one thing should be noticed from Macklin and Müller [2013]. Equation 4.10 should be separate in two cases. Whether \( k \) is a neighboring or not:

\[ \nabla p_k C_i = \left\{ \begin{array}{ll}
\sum_j \nabla p_k W(p_i - p_j, h) & \text{if } k = i \\
-\nabla p_k W(p_i - p_j, h) & \text{if } k = i
\end{array} \right. \]  \hspace{1cm} (4.11)

#### 4.6.1 Calculate \( \lambda \)

Now reference equation 3.14 in Section 3.4, solving the scaling factor by equation 4.12

\[ \lambda_i = -\frac{C_i(p_1, \ldots, p_n)}{\sum_k \left| \nabla p_k C_i \right|^2} \]  \hspace{1cm} (4.12)

Macklin and Müller [2013] combined the idea with constraint force mixing (CFM) Smith to soften the constraint. Now the equation 4.12 becomes to equation 4.13 \( \varepsilon \) is a user defined constant, and it could be define from the user.

\[ \lambda_i = -\frac{C_i(p_1, \ldots, p_n)}{\sum_k \left| \nabla p_k C_i \right|^2 + \varepsilon} \]  \hspace{1cm} (4.13)

#### 4.6.2 Calculate \( \Delta p \)

After run through all the particle with above calculation. Each particle has its own scaling factor \( \lambda_i \), then \( \Delta p \) value of each particle unit could be evaluated by equation 4.14.

\[ \Delta p_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(p_i - p_j, h) \]  \hspace{1cm} (4.14)
4.7 Tensile Instability

From Figure 4.6 Top: It could be seen that the particles clumping together because of the neighbor deficiencies. This problem happened when particle has few neighbors. Once particle has not enough neighbors, it would cause the negative pressure then could not satisfy the rest density. To solve this problem, [Macklin and Müller, 2013] claims that by adding the artificial pressure in terms of smoothing kernel itself. This value is in equation 4.15. Next, in the equation 4.14 for calculating $\Delta p$, $s_{corr}$ is added into it. The new equation become in equation 4.16

$$s_{corr} = -k\left(\frac{W(p_i - p_j, h)}{W(\Delta q, h)}\right)^n$$

(4.15)

Where $|\Delta q| = 0.1h, ..., 0.3h$, $k = 0.1$, and $n = 4$ works well in [Macklin and Müller, 2013].

$$\Delta p_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j + s_{corr}) \nabla W(p_i - p_j, h)$$

(4.16)

Figure 4.6: Top: PBF without artificial pressure term. Bottom: With artificial pressure term [Macklin and Müller, 2013].

In this project, tensile instability is implemented, however, it is not obvious to see the differences with or without $s_{corr}$ value, due to the whole simulation is not efficient enough.
4.8 Vorticity

From Macklin and Müller [2013], he proposed a method to replace the lost energy with water splash. It is called vorticity confinement. The implemented equation are listed as following:

\[ \omega_i = \nabla \times \mathbf{v} = \sum_j \mathbf{v}_{ij} \times \nabla p_j W(p_i - p_j, h) \]  

\[ f_{vorticity}^i = \varepsilon (\mathbf{N} \times \omega_i) \]  

In Figure 4.7, it shows the difference between with or without vorticity influence from Macklin and Müller [2013].

![Figure 4.7: Left: PBF without vorticity. Right: With vorticity. From Macklin and Müller [2013]](image)

Vorticity is not implemented in this project, due to the time limitation. This would leave to as future work.

4.9 XSPH

In Macklin and Müller [2013], it does not mention how to implementation XSPH. So Schechter and Bridson [2012] becomes the main resources for XSPH algorithm. According to Schechter and Bridson [2012], XSPH method is shown in equation 4.19:

\[ \mathbf{v}_i^{\text{new}} = \mathbf{v}_i + \varepsilon \sum_j \mathbf{v}_{ij} \cdot W(p_i - p_j, h) \]  

(4.19)
He redefined the XSPH functionality as "Noise in the raw particle velocities". The main purpose for adding XSPH is for adding some noise value into velocity, this would let each particle keep its velocity differ from its neighbors. From [Macklin and Müller (2013)], $c$ is set to 0.01. $c$ could set as a user-tuned constant, from [Schechter and Bridson (2012)].

4.10 Boundary

Boundary condition is also an important part for fluid simulation. As mentioned before, the collision constraints is ignored in position based fluid. The boundary condition we only need to focus on the original solution in particle-based fluid simulation method.

From [Macklin and Müller (2013)], he mentioned that one of the future work is the particle clustering problem near the high density boundary area. He suggest [Akinci et al. (2012)] method for solving this problem.

In this project, due to the time limitation, boundary condition is implemented by reversing the velocity with a scale factor and set the particle position at the boundary position. In this case, the boundary position is the water container size (tank size).

4.11 Initial Value

The relation between mass and volume is shown in equation (4.7). With another relation between particle radius and particle volume is shown in equation (4.20). With these two conditions, it could be roughly estimated fluid simulation initial values. All the values seem to with relatively scale.

$$\frac{4}{3}\pi r^3 = V_i$$  \hspace{1cm} (4.20)

As for the amount of particle neighbors, from [Macklin and Müller (2013)], the average particle neighbors count must be around 30-40 for preventing clustering problem.
Chapter 5

Pipeline

5.1 Stand Alone Application

This project is developed by C++ and used NGL library by Macey based on Macklin and Müller [2013]. In Figure 5.1, it shows the stand alone application in this project. The user interface is straight forward and easy to use. In this application, the following listed are the basic step for simulation:

- Step 1: Load XML file.
- Step 2: Load Particle.obj file.
- Step 3: Export Simulation.
- Step 4: Start Simulation.
- Step 5: End simulation or Reset.

Firstly, loading a XML file for initializing the parameters for simulation. Secondly, importing the initial particles data, and this is also could be done by Houdini digital asset provided in this project. For a quick check of simulation results, this application is not working efficiently once particles number increase to over 20k. As a result, exporting the simulation result for a quick review is necessary. The reason why exporting simulation result comes before the start simulation is because once start simulation, the application would become really slow. Export first to make sure with whole simulation process. Here, using the source code provided by Macey exporting alembic file. The exported alembic file is named particlesOut.abc. It would locate within the Qt project folder. During the simulation run time, user can pause simulation by
clicking the button “Start/Pause Simulation”, and vice versa. Lastly, after the whole simulation finish, user could reset all data. Then repeat from step 1, do another time new simulation.

![Position Based Fluid Simulator](image)

**Figure 5.1**: Position based fluid simulator - stand alone application.

### 5.2 Houdini Digital Asset

As mentioned in the last section, Houdini digital asset was designed for user could set up the data before simulation. There are two digital asset, *PBF Initial HDA* and *PBF Rendering HDA*. Obviously, one is used before simulation, and the other is for visualizing simulation result. In *PBF Initial HDA*, using python for creating a xml file located on the data path when user create, and this idea comes from [Priscott 2010]. In Houdini, node *point from volume* could create a numerous points data based on geometry, and it provides the parameters to change the point density. For example, input a *bunny.obj* model, then the particles would form a numerous points as a shape of bunny. For more Houdini digital asset user guide please reference the appendix document.
Chapter 6

Results

This chapter would show different type of dam break fluid simulation result based on one set of parameters:

- Tank size = 2
- Smoothing kernel size = 0.1
- Rest density = 1000
- Particle radius = 0.025
- Particle mass = 0.1
- Gravity = \(-9.81\)
- Max iteration = 5
- Time step = 0.03
- \(E_{po} = 1e^{-06}\)

Tank size is the size value of boundary. Max iteration is the value in PBF algorithm solver iteration. \(E_{po}\) is the relaxation parameter in equation 4.13.

In this project, mainly focusing on the simulation result correctness, so the visualization is not important. From Figure 6.1 to Figure 6.4 are the simulation results after rendering by Houdini. For further fluid rendering knowledge, van der Laan et al. [2009] provided an efficient method for only rendering the particles could be seen from the camera.
CHAPTER 6. RESULTS

Figure 6.1: Simulation result - Single Dam Break (particle number: 10k)

Figure 6.2: Simulation result - Dam Break (particle number: 10k)
Figure 6.3: Simulation result - Transparent Dam Break (particle number: 10k)

Figure 6.4: Simulation result - Bunny Drop (particle number: 14k)
Chapter 7

Conclusion

The goal in this project is to design and implement the paper from Macklin and Müller [2013]. Integrating the simulator with other 3D software is another aim in the initial design. Overall, the implementation in this project has basic fluid features in the limited of time. If looking for a better realistic simulation result, more suitable set of parameters, more efficient algorithm, speed up the simulation process, it would take some time to satisfy the expected result. Ideally, most of methods, equations from SPH and PBD are implemented, such as neighbors search, kernel functions, fluid density calculation, computing gradient constraint function, XSPH, fluid advection. At the end of this project, due to the code is without optimization, the running time should take some time to see the simulation result.

7.1 Future Work

As mentioned before about the boundary condition, particle-based fluid simulation disadvantage is the particle clustering problem. To solve this issue, Akinci et al. [2012] adjust the density calculation, and the results seems work well. In addition, the fluid with other rigid body interaction is another subject that we are interested. The initial values are really important in fluid simulation. For instance, once the smoothing kernel size changes, the simulation could become unstable cause like explosion effect. Different sets of parameter for tweaking the simulation result. Lastly, for most of the fluid simulation papers are working on real-time, paralleling with openMP or GPU calculation. It is essential to update the information in this area especially in computer graphics.
Appendix A

XML File

A.1 XML File Layout

Figure A.1: PBF XML file layout, written and exported from Houdini and parsed into C++ Qt project.
Appendix B

PBF HDA User Guide

From Chapter 5, Houdini digital assets are used to assist the stand alone application for initial setup and visualization. This appendix would go through a quick look into these assets.

B.1 PBF Initial HDA User Guide

In PBF Initial HDA, there are two tabs: Initial Value and Particles Setup, and are shown in Figure B.1 and B.2. The user interface in Houdini is straight forward and user friendly than QtGUI. Firstly, in tab Initial Value, the xml file should be assigned to a location and saved, and the rest parameters could be adjust from times to times for each simulation. Once creating the format of XML file, user could only changes the values in XML file for quick loading. Saving different set of XML files could compare the different simulation result.

As for tab Particles Setup, maximum two objects could be loaded into this framework. Result from Chapter 6, all are created by default.bgeo or bunny.obj. Basic translation, rotation, and scale could change by user as the expected particles position data.

B.2 PBF Rendering HDA User Guide

This one is really simple, purely for viewing the export simulation result. In Figure B.3, the viewing is at node alembic. For quick review of the simulation result, import the simulation result particlesOut.abc exported from stand alone application.
Figure B.1: Initial Value tab in PBF Initial HDA.

Figure B.2: Particles Setup tab in PBF Initial HDA.
Figure B.3: PBF Rendering HDA.


Mathieu Desbrun, Peter Schröder, and Alan Barr. Interactive animation of structured deformable objects.


