Volume 0 (1981), Number 0

Smoothed Particle Hydrodynamics Fluid Simulation in Unity

Tommy Zhong

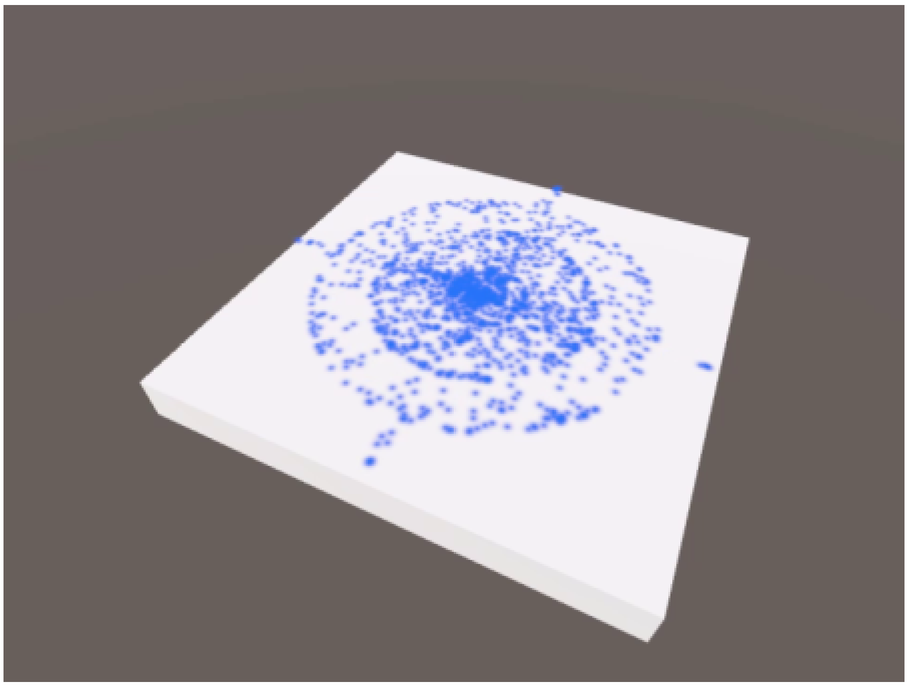

Figure 1: *One scenario of SPH fluid simulation 20s after the start. See Figure 5 for details.*

Abstract

Smoothed Particle Pydrodynamics (SPH) is a particle-based Lagrangian fluid simulation method designed to model real life liquid, gaseous, and plasma fluids. SPH approximates the pressure force and the viscosity force described in Navier-Stokes equation on each particle using a kernel smoother function. This article reports an SPH implementation in Unity game engine and a series of evaluations, from which the effectiveness, limitations, and potential improvements of the fluid simulation can be inferred and concluded.

CCS Concepts

• ***Computing methodologies*** $\rightarrow$ *Simulation by animation;* • ***Theory of computation*** $\rightarrow$ *Massively parallel algorithms;* • ***Software and its engineering*** $\rightarrow$ *Integrated and visual development environments;*

1. Introduction

From the blood that circulates in human bodies to the plasma that forms the sun, fluid exists everywhere. Out of the of the four states of matters, three of them are considered fluids. Realistic fluid animations are in high demand as they can simulate a variety of physics and engineering problems, thanks to the abundance of fluid in real life [DFF19]. These simulations are also some of the most difficult, as the fickleness of fluids makes them immensely complex to compute. This project explores the challenges of fluid simulations and their potential solutions through an implementation of Smoothed Particle Hydrodynamics (SPH) using a C# script in Unity game engine.

2. Related Work

SPH is one of the particle-based Lagrangian methods to simulate fluid, reportedly first proposed in astronomy field to model the stars [GM77]. SPH is not the only type of Lagrangian fluid. For example, Least-Squares Meshfree method [ZKY04] is another approach that uses particles. In Lagrangian methods, a large number of particles sample the conditions of a fluid at given positions. At each time step, particles affect each other to predict fluid movement.

Aside from particle-based Lagrangian methods, another popular approach is grid-based Eulerian method [HK17] [Par12]. An Eulerian fluid uses a mesh formed by a set of grids fixed in space. Information from each grid is used to simulate the fluid's movement in the next time step. A recent example of an Eulerian method is pressure force midpoint method [JSG*22].

Compared to a Lagrangian approach, an Eulerian fluid's main advantage is the relative ease to conserve volume and incompressibility. Since two grids hold a fixed distance between each other, an incompressible fluid must maintain a certain amount of volume, whereas a Lagrangian fluid with variable proximity between particles cannot access or change its volume directly. Another advantage of Eulerian approaches is the compatibility with a continuous fluid. A Lagrangian implementation of a liquid or a smoke cloud can easily be visually discrete unless a large number of particles are used, but an Eulerian fluid can achieve a natural look using less number of grids and therefore allows better performance.

Eulerian approaches have two main disadvantages. First, a grid-based fluid can only exist in a set space while a particle-base fluid can move freely (although a spatial constraint benefits Lagrangian approaches as it enables more optimizations e.g., grid based neighbourhood search [IOS*14]). Second, a Lagrangian fluid employs widely-used particle physics to interact with other objects, whereas a Eulerian fluid must define its own mesh physics. Implementation of Eulerian fluids on an existing platform often requires an extension to the physics engine, and the quality of such implementation is generally weaker and less consistent than in Lagrangian fluids.

Some approaches combine Lagrangian and Eulerian techniques and seek to overcome the difficulties of both sides, such as the particle-in-cell (PIC) method and its extension, the material point method [SCS94], but these approaches have their own shortcomings. A part of the goal of this SPH project is to seek methods to mitigate the shortcomings of Lagrangian methods.

SPH can be extended to fit specific needs for simulations. To accurately simulate realistic fluids, an SPH implementation must take extra steps to limit compression. One method to numerically achieve low compressibility is an iterative solver that adjusts SPH pressure force calculation (see sessions 3.2.1 and 3.3.3) to reduce compression in every loop until an acceptable level is reached [IOS*14].

Another extension is two-way interaction between fluid particles and rigid bodies. Examples include a generic solution that treats a rigid body as a type of fluid [CMT04] and an SPH-specific interlinked method that incorporates pressure force calculation and collision resolution to produce a desired effect [GPB*19].

Though generally not applicable to scientific or engineering researches, simulation of a liquid matter can be implemented as a surface or a height field [Par12]. These methods maintain only an appearance of a liquid and are useful to animate watery environments in video games and movies. To add a limited amount of physical interactions, ripples and waves simulations can be implemented for the surface method, and external forces can be added to the height field method.

3. Overview

As a Lagrangian method, SPH simulates a fluid with collection of sample points following the particle form of Navier-Stokes equation. SPH can be implemented as an addition to an existing particle system. To better explain SPH implementation, this overview begins with particle system introduction in section 3.1, followed by a breakdown of Navier-Stokes equations in section 3.2, and then finally an example of SPH algorithm in section 3.3.

3.1. Particle System

A particle system governs a collection of particles and their changes over time. Each particle i in the collection stores attributes at time t including the particle's position $x_i(t)$ and velocity $v_i(t) = \frac{dx_i}{dt}$. Depending on the purpose of a particle system, particles can have more attributes as required, such as a life span and a noise in velocity for a spark effect. In a Newtonian motion simulation, each particle has a mass attribute m_i, so that the particle can be accelerated by forces using Newton's second law of motion

$$a_i(t) = \frac{dv_i}{dt} = \frac{d^2x_i}{dt^2} = \frac{1}{m_i}\sum F_i \tag{1}$$

where F_i represents a force applied to particle i. At each time step of size Δt, a particle system updates each particle's acceleration $a_i(t)$ with each force F_i, and then velocity $v_i(t+\Delta t)$ and position $x_i(t+\Delta t)$ with any solution for Ordinary Differential Equations, such as the symplectic Euler method

$$v_i(t+\Delta t) = v_i(t) + \Delta t a_i(t) \tag{2}$$

$$x_i(t+\Delta t) = x_i(t) + \Delta t v_i(t+\Delta t) \tag{3}$$

which has low complexity at the cost of low accuracy. Unity's physics engine updates particle positions and velocities internally in every fixed step. This SPH implementation incorporates Unity physics using only Equation 2 of the Euler method to add custom-defined acceleration to particle velocities before every internal physics update.

3.2. Navier-Stokes Equations

Navier-Stokes equations describe a fluid's movement with both external and internal forces. The equations for Lagrangian and Eulerian implementations differ slightly [IOS*14]. The Lagrangian version of Navier-Stokes equation shows acceleration of a particle i with an alternative version of Equation 1:

$$a_i(t) = -\frac{\nabla p_i}{\rho_i} + \nu\nabla^2 v_i + \frac{F_i^{external}}{m_i} \tag{4}$$

In addition to total external force $F_i^{external} = \sum F_i$, each particle is accelerated by pressure force and viscosity force. Pressure force acceleration $-\frac{\nabla p_i}{\rho_i}$ and viscosity force acceleration $\nu\nabla^2 v_i$ would be discussed in the following subsections. Strictly speaking, these two terms need to be multiplied by the particle mass m_i to obtain the amount of each force. For the purpose of calculating accelerations, this step is ignored in Equation 4. With respect to conservation of momentum, the total internal force of a particle system must be zero.

3.2.1. Pressure Force

Acceleration from fluid pressure force is $-\frac{\nabla p_i}{\rho_i}$ at the location x_i [HK17]. Local density ρ_i is a scalar value defined by mass per volume unit (i.e., particle), and local pressure gradient ∇p_i is a vector value defined by force per volume unit. A gradient in 3D space $\nabla = [\frac{\delta}{\delta x} \frac{\delta}{\delta y} \frac{\delta}{\delta z}]^T$ obtains a vector value from a scalar value, such as pressure p_i. In a particle-based fluid, density and pressure of a particle are not constant, and the former is sometimes referred to as dynamic density to differentiate from a fluid's constant rest density ρ_0 (strictly speaking, rest density is only constant under constant temperature and atmospheric pressure. Environmental factors are ignored in this SPH fluid simulation). If a particle's dynamic density differs from rest density, the fluid at this position should have nonzero amount of pressure. The gradient of pressure ∇p_i returns a vector that points from a position with high pressure to a position with low pressure. A force along this vector propels a fluid to balance the density difference.

3.2.2. Viscosity Force

Acceleration from viscosity force $\nu\nabla^2 v_i$ is also known as shear, diffusion [HK17], and friction [IOS*14]. Kinematic viscosity ν directly controls the amount of viscosity force on every fluid particle. The term $\nabla^2 v_i$ uses Laplacian operator $\nabla^2 = \nabla \cdot \nabla = \frac{\delta^2}{\delta x^2} + \frac{\delta^2}{\delta y^2} + \frac{\delta^2}{\delta z^2}$. When applied to a vector field, Laplacian produces a smoothing effect by reducing the difference between two vectors nearby. Fluid viscosity is created from the application of this effect to velocity field of the particle system.

3.3. SPH

The main concept of SPH is to solve a particle's arbitrary value A_i with a sum of every nearby particle's value A_j scaled by a kernel smoother w_{ij} [IOS*14]. This technique can approximate both scalar values such as density ρ_i and vector values including the pressure and viscosity forces in Navier-Stokes equation listed in section 3.2. The following subsections describe the process to implement SPH in three general steps. Each step depends on the completion of previous step. Algorithm within a step is thread-safe, and therefore can, and should, be parallelized. This SPH implementation currently has parallelized forces calculations described in section 3.3.3.

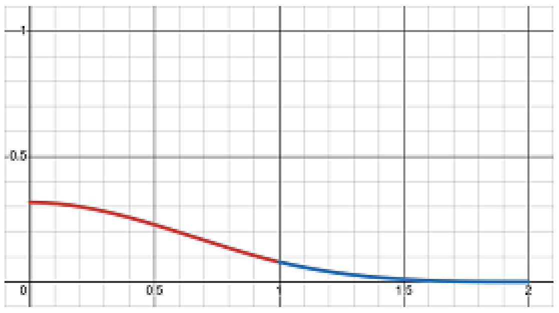

Figure 2: *A graph of $w_{ij} = w(||x_{ij}||)$ implemented with function f from Equation 6 using $h = 1$. Red and blue lines indicate two pieces of a piecewise function.*

3.3.1. Kernel Smoother

A kernel smoother w_{ij} scales the amount of influence between two distinct particles i and j. Let $x_{ij} = x_i - x_j$ be the position difference of two particles. With smoothing radius h in dimension d, a kernel smoother is defined as:

$$w_{ij} = w(||x_{ij}||) = \frac{1}{h^d} f(\frac{||x_{ij}||}{h}) \tag{5}$$

The kernel function $f : \mathbb{R} \to \mathbb{R}$ can be any function that respects a given set of constraints including non-negative, symmetric, and decaying, etc. (see [DFF19] for a full list). These properties ensure that the amount of influence exchanged between a pair of particles have equal strength, and the closer two particles are, the more influence they have on each other. In this SPH implementation, f is a cubic spline function [IOS*14]:

$$f(q) = \frac{3}{2\pi}\begin{cases} \frac{2}{3} - q^2 + \frac{1}{2}q^3 & 0 \le q < 1 \\ \frac{1}{6}(2-q)^3 & 1 \le q < 2 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Smoothing radius constant h determines both the effect and performance of kernel smoother. A low value leads to fewer calculations per time step and more animated fluid. See Figure 2 and Figure 3 for a comparison between two radius values.

In acceleration calculations, SPH requires not only the kernel smoother but also its gradient ∇w_{ij}. The gradient of kernel smoother is defined using the chain rule, as seen in Equation 7.

$$\nabla w_{ij} = \frac{1}{h^{d+1}} f'(\frac{||x_{ij}||}{h}) \frac{x_{ij}}{||x_{ij}||} \tag{7}$$

In SPH, values of w_{ij} and ∇w_{ij} for all particle pairs must be computed in every time step. Note the facts that $w_{ij} = w_{ji}$ and

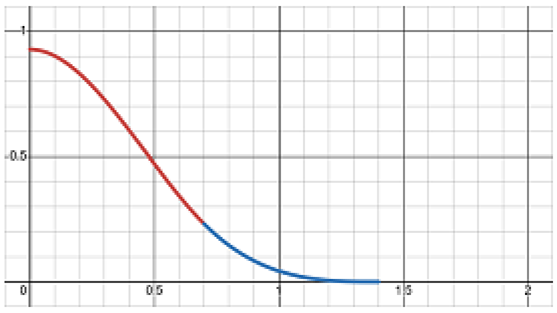

Figure 3: *A graph of the same implementation of w_{ij} using $h = 0.7$. Both graphs are generated using Desmos graphing calculator.*

$\nabla w_{ij} = -\nabla w_{ji}$ reduce the amount of computations needed by a half. Further optimizations of this step is discussed in session 5.

3.3.2. Attributes

Attributes required by Equation 4 include each particle's mass m_i, rest density ρ_0, dynamic density ρ_i, pressure p_i, and kinematic viscosity μ. Particle mass, rest density, and kinematic viscosity use constant values. In an SPH particle system that simulates only one type of fluid, these values should be the same for every particle. Some SPH implementations obtain particle mass from values of rest density and kernel smoothing radius e.g., $m_i = h^3\rho_0$ [IOS*14]. Relations between particle mass, smoothing radius, and rest density are evaluated in section 4.2.

For the dynamic values, a particle's dynamic density is initially the rest density ρ_0, and then updated by information from other particles j using kernel smoother:

$$\rho_i = \sum_j m_j w_{ij} \tag{8}$$

Pressure of a particle is obtained from the difference between dynamic and rest densities using the Tait equation [DFF19][HK17]:

$$p_i = \frac{k\rho_0}{\gamma}\left(\frac{\rho_i}{\rho_0} - 1\right)^{\gamma} \tag{9}$$

The stiffness constant k, defined as the square of speed of sound in a fluid, controls the amount of dynamic pressure in a moving fluid. The reference value γ is generally set to 7 for high quality [IOS*14] or 1 for high performance [HK17]. Since Unity particle system uses single-precision floating point values, this SPH implementation chooses $\gamma = 1$ to avoid potential overflows.

3.3.3. Forces

The final step of SPH is to numerically solve pressure and viscosity forces defined in Navier-Stokes equation 4. With kernel smoother and attributes obtained in previous steps, an arbitrary field's value at a particle's position A_i can be described as following [IOS*14]:

$$A_i = \sum_j \frac{m_j A_j}{\rho_j} w_{ij} \tag{10}$$

Since the terms other than w_{ij} are considered constant, it follows that $\nabla A_i = \sum_j \frac{m_j A_j}{\rho_j} \nabla w_{ij}$ and $\nabla^2 A_i = \frac{m_j A_j}{\rho_j} \nabla^2 w_{ij}$. However, these results are not guaranteed to be symmetric. Although this version of ∇A_i and $\nabla^2 A_i$ can directly substitute pressure force term $\frac{\nabla p_i}{\rho_i}$ and viscosity force term $\nu\nabla^2 v_i$ to solve Navier-Stokes equation 4, the sum of these internal forces is not always zero, and the particle system's momentum would not be conserved.

To avoid the asymmetry problem in pressure force acceleration, an alternative gradient equation derived from the quotient rule is used [HK17]:

$$\frac{\nabla p_i}{\rho_i} = \sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right) \nabla w_{ij} \tag{11}$$

For a symmetric solution to the viscosity force acceleration, some methods require the Laplacian term $\nabla^2 w_{ij}$ [HK17] and some others require only the gradient term ∇w_{ij} [DFF19] [IOS*14]. This implementation uses one of the latter solutions

$$\nu\nabla^2 v_i = 2\nu \sum_j \frac{m_j v_{ij}(x_{ij} \cdot \nabla w_{ij})}{\rho_j(||x_{ij}||^2 + 0.01h^2)} \tag{12}$$

where $x_{ij} = x_i - x_j$ and $v_{ij} = v_i - v_j$. By adding Equations 11 and 12 to a particle's acceleration term in Equation 2, this SPH implementation is complete.

4. Evaluation

The following evaluations use Unity 2021.3 on a Windows 10 64-bit home computer. Specifications include Intel Core i7-8700K CPU, Nvidia GeForce GTX 1070 Ti, and 16 GB RAM. Simulations are tested in 20 frames per second (FPS), equivalent to 0.05s time step, and 960*720 resolution. Time performance of simulations is measured with a C# Stopwatch object.

4.1. Quantitative Results

Since each SPH particle requires information of potentially all other particles, the time and memory complexities are both $O(n^2)$ where n is the number of particles. Under Unity's default settings, observations show that a real time fluid simulation maintains 20 FPS when the scene has around 500 particles. A simulation of 1000 particles has 5 FPS, and a simulation of 1600 particles has 2 FPS. These results correlate with the asymptotic analysis.

4.2. Qualitative Results

The main focus of this section is to observe the effects of constant values on the SPH fluid. All simulations use 1500 particles, initially in the shape of a 1^3 cube, on the surface of a cube container

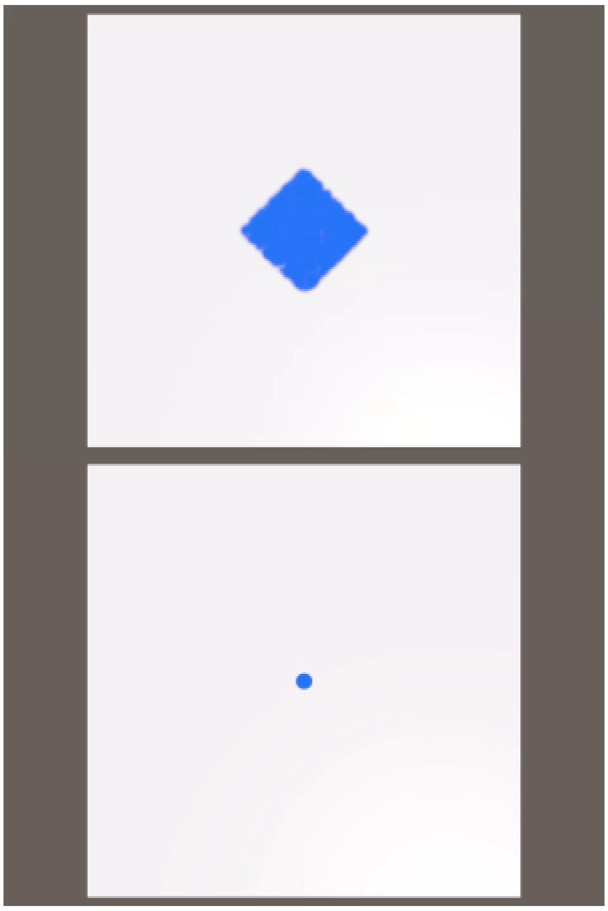

Figure 4: *Snapshots of a simulation with $m_i = 2$, $\rho_0 = 10$, $h = 5$, $k = 5$, and $\nu = 0$. The fluid switches between the initial square shape and a point.*

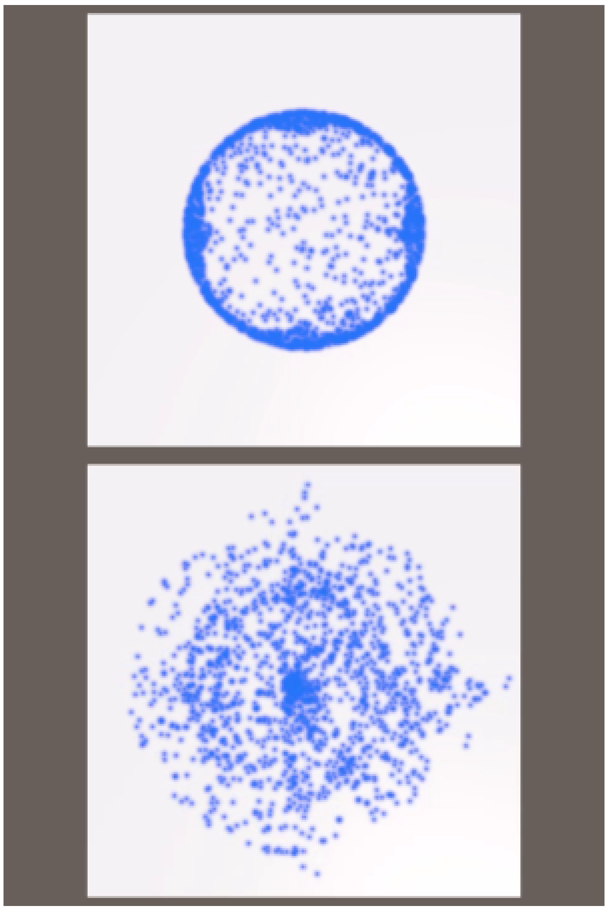

Figure 5: *Snapshots of a simulation similar to Figure 4 but with $m_i = 3$ instead, taken 10s and 50s after the simulation begins.*

with 5^3 volume and 2 thickness. Unity particle system gravity and collision physics are enabled to add external forces. Particle collision on the container produce no dampening or bouncing forces. In all simulation scenarios, the fluid is observed to possess high compressibility and form a single layer on a flat surface under the effect of gravity. Due to this and the relatively low number of particles, the simulations are observed through a top-down orthographic camera in a two-dimensional manner. In the following figures, fluid is represented by blue particles bounded within a box represented by a white square. Note that Unity particle material has an adaptive size and displays multiple particles in close proximity with one large particle.

The ratio between mass m_i, rest density ρ_0, and smoothing radius h control the pressure force that expands the fluid. In the following scenarios, kinematic viscosity is set to 0 (unless explicitly stated) to observe the effect of pressure force alone. In one scenario shown in Figure 4, the fluid does not expand, but instead cycles between the initial formation and a collapse to a single point. With a slightly higher particle mass, the fluid particles are evenly distributed roughly in a circular shape, as shown in Figure 5. Further increase in particle mass causes the fluid particles to gather at the edges and corners of the bounding box, and may escape it if time step is set above 0.05s.

Changes in smoothing radius not only produce an opposite effect compared to changes in mass, but also affect how uniform the particles move. For example, a decrease in smoothing radius causes the fluid to expand similar to the scenario in Figure 5, but in this case the particles move more randomly. A smoothing radius below 2 makes the fluid unstable, with some particles occasionally having extremely strong acceleration and escaping the bounding box. Curiously, a fluid with low smoothing radius, high particle mass, and a small amount of viscosity can form condensed chunks that repel each other, and the distance between chunks correlates with smoothing radius. Figure 6 shows one such scenario.

Stiffness k and kinematic viscosity ν scale the pressure force and viscosity force. An increase in stiffness makes a fluid move faster, but has no significant effect on the fluid's shape. An increase in viscosity slows down the fluid during expansion or collapse and reduces the randomness of particle movements. Figure 7 showcases two scenarios with different viscosity level.

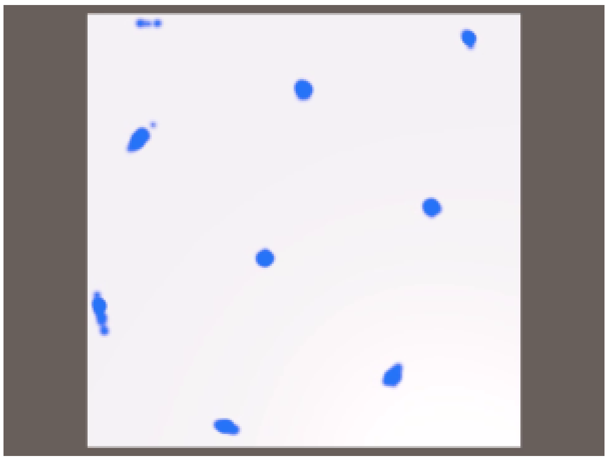

Figure 6: *A simulation with $m_i = 10$, $\rho_0 = 10$, $h = 2$, $k = 5$, and $\nu = 2$. The snapshot is taken 20s after the simulation begins.*

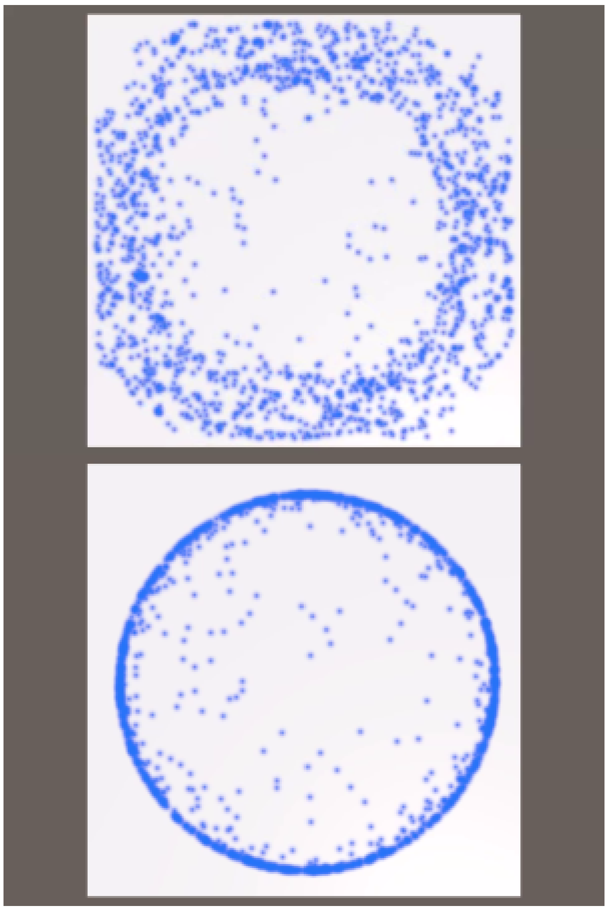

Figure 7: *Comparison of two simulations similar to Figures 4 and 5 with $m_i = 4$. The fluid above has $\nu = 0$, and the fluid below has $\nu = 10$. Both snapshots are taken 50s after simulations begin.*

5. Conclusion

Quantity wise, this SPH implementation in Unity can be used to simulate the physics of low density fluids, such as gases. Liquid simulation may require hundreds or thousands times more particles, and is infeasible with the current time performance. As stated in section 3.3, the steps of SPH are independent and should be parallelized. For a massive number of fluid particles, the best solution is to utilize the concurrent computation power of a GPU. In Unity, GPU access cannot be done in C# scripts alone, and requires the use of High-Level Shader Language (HLSL). Aside from parallelization, one possible optimization is grid search [IOS*14]. When a domain space is represented by h^3 cubic grids (where h is smoothing radius as stated before), each particle's neighbours are either in the same grid or the surrounding 26 grids. With these two improvements, SPH can be expected to simulate liquid in real-time at an acceptable frame rate.

Quality wise, SPH particles show fluid-like dynamic movements under gravity and collision, but fails to simulate static fluid due to the overlap and compression of fluid particles. Fixing this problem requires additional implementation of compressibility constraint, such as the iterative method mentioned in section 2 [IOS*14]. This method requires control over external forces, which in turn may require low level configuration of Unity psychics execution order through PlayerLoop class. Another issue regarding the quality of SPH fluid is the visual representation. Unity particle system's default material should be replaced by one that depicts the desired type of fluid.

The main challenge of an SPH simulation is finding the appropriate input values for the desired type of fluid. Although subsection 4.2 explored the general effects of changes in mass, rest density, smoothing radius, stiffness, and kinematic viscosity, a strategy to find a combination for a specific type of fluid is not , and the differences between various SPH implementation choices, such as an alternative kernel function to Equation 6, are yet to be tested. Overall, this project has reached its original goal and explored the difficulties of fluid simulation.

References

[CMT04] CARLSON M., MUCHA P. J., TURK G.: Rigid fluid: Animating the interplay between rigid bodies and fluid. *ACM Trans. Graph. 23*, 3 (Aug 2004), 377–384. doi:10.1145/1015706.1015733.

[DFF19] DUTRA FRAGA FILHO C. A.: *Smoothed Particle Hydrodynamics Method*. Springer International Publishing, Cham, 2019. doi:10.1007/978-3-030-00773-7_3.

[GM77] GINGOLD R. A., MONAGHAN J. J.: Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society 181*, 3 (12 1977), 375–389. doi:10.1093/mnras/181.3.375.

[GPB*19] GISSLER C., PEER A., BAND S., BENDER J., TESCHNER M.: Interlinked sph pressure solvers for strong fluid-rigid coupling. *ACM Trans. Graph. 38*, 1 (Jan 2019). doi:10.1145/3284980.

[HK17] HOUSE D., KEYSER J. C.: *Foundations of physically based modeling and animation*. Boca Raton : Taylor & Francis, 2017, pp. 265–292.

[IOS*14] IHMSEN M., ORTHMANN J., SOLENTHALER B., KOLB A., TESCHNER M.: SPH Fluids in Computer Graphics. In *Eurographics 2014 - State of the Art Reports* (2014), Lefebvre S., Spagnuolo

M., (Eds.), The Eurographics Association. doi:10.2312/egst.20141034.

[JSG*22] JIANG J., SHEN X., GONG Y., FAN Z., LIU Y., XING G., REN X., ZHANG Y.: A second-order explicit pressure projection method for eulerian fluid simulation. *Computer Graphics Forum 41*, 8 (2022), 95–105. doi:https://doi.org/10.1111/cgf.14627.

[Par12] PARENT R.: *Computer animation algorithms and techniques*, 3rd ed. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, Waltham, Mass, 2012, pp. 251–281.

[SCS94] SULSKY D., CHEN Z., SCHREYER H.: A particle method for history-dependent materials. *Computer Methods in Applied Mechanics and Engineering 118*, 1 (1994), 179–196. doi:https://doi.org/10.1016/0045-7825(94)90112-0.

[ZKY04] ZHANG X. K., KWON K.-C., YOUN S.-K.: Least-squares meshfree method for incompressible navier–stokes problems. *International Journal for Numerical Methods in Fluids 46*, 3 (2004), 263–288. doi:https://doi.org/10.1002/fld.758.