How to implement a Fluid Simulation on the CPU with Unity

Yue Zhang

April 17, 2024

Abstract

Smoothed Particle Hydrodynamics (SPH) is an effective method for simulating fluids, representing them through a system of interacting particles. Initially developed for astronomical applications, its utility has broadened to encompass real-time fluid simulations in the Unity game engine. This method utilizes the Navier-Stokes equations to calculate fluid properties such as pressure, density, and viscosity at the particle level. The current implementation successfully simulates between 5,000 and 10,000 particles at a consistent rate of 30 frames per second. While the system performs effectively, there are still opportunities for improvements in terms of simulation speed, stability, accuracy, and the visual quality of the fluid dynamics.

1 Introduction

The study of fluid dynamics spans various disciplines, encompassing not only the familiar liquids and gases but also extending to complex phenomena such as geological hazards, astronomical events, and fire behavior. Understanding and simulating these fluid movements require sophisticated computational models that can accurately capture the intricate behaviors of such diverse materials under different conditions.

Fluid simulation is a critical component in numerous applications, ranging from engineering and environmental science to entertainment and visual effects. As technology progresses, the need for more realistic, efficient, and versatile fluid simulations has become paramount. This has led to the development and refinement of various computational methods, each suited to specific types of fluid behavior and simulation requirements.

Two primary computational frameworks dominate the field of fluid dynamics: the Eulerian and Lagrangian methods. Each framework offers distinct advantages and limitations, making them suitable for different kinds of fluid simulation tasks.

Eulerian Method: This approach is characterized by a fixed spatial grid that divides the simulation space into discrete cells. Fluid properties such as velocity, pressure, and density are computed at fixed points in space, making this method particularly well-suited for simulating high-speed gases, water flow, and diffusion processes. However, the Eulerian method struggles with simulations that involve highly viscous fluids or those that require the tracking of fluid interfaces, such as molten lava or honey.

Lagrangian Method: In contrast, the Lagrangian method models the fluid by discretizing it into particles, each carrying individual properties and moving through space. This particle-based approach is adept at handling complex fluid interactions, such as splashing and merging, making it ideal for simulating non-Newtonian fluids, dense suspensions, and particulate flows. The method ensures mass conservation and can intuitively handle free surface flows but can be challenged by enforcing incompressibility.

To bridge the gaps between these methods, hybrid techniques such as the Material Point Method (MPM), Affine Particle-in-Cell (APIC), and Particle-in-Cell/Fluid-Implicit Particle (PIC/FLIP) have been developed. These methods combine the strengths of both Eulerian and Lagrangian approaches to offer more robust and flexible solutions for fluid simulation.

2 Background

The origins of Smoothed Particle Hydrodynamics (SPH) trace back to pioneering works by Gingold and Monaghan in 1977 [GM77] , who developed the method to model oscillating or non-spherical stars.

In the same year, Lucy [Luc77] expanded the application of this approach to study the progression and evolution of protostars, marking the beginning of SPH's adaptation in astrophysics.

In 1998 Reeves[Ree98] presented a technique for modeling a class of objects he called fuzzy objects, systems-a method for modeling fuzzy objects such as fire, clouds, and water. Particle systems model an object as a cloud of primitive particles that define its volume. Over a period of time, particles are generated into the system, move and change form within the system, and die from the system. The resulting model is able to represent motion, changes of form, and dynamics that are not possible with classical surface-based representations. The particles can easily be motion blurred, and therefore do not exhibit temporal aliasing or strobing. Stochastic processes are used to generate and control the many particles within a particle system. The application of particle systems to the wall of fire element from the Genesis Demo sequence of the film Star Trek II: The Wrath of Khan [10] is presented

Further advancements in fluid simulations were made by Müller et al. [MCG03], who derived their method from the Navier-Stokes fluid equations to enhance the realism of the simulations. Clavet et al. [CBP05] integrated springs into their SPH implementation to better model viscoelastic fluids, thus adding realistic elasticity and plasticity to non-Newtonian fluids. In subsequent developments, Harada et al. [HKK07] and Yan et al. [YWH+09] utilized GPUs to accelerate the computationally intensive tasks involved in SPH, with Yan achieving a significant milestone of 66 frames per second with 16,000 particles.

In more recent applications, SPH has continued to find relevance across various fields. Faber et al. [FLR10] utilized SPH to simulate interactions between stellar objects, including star mergers and astronomical collisions. Cao et al. [CPB+18] highlighted the advantages of SPH over traditional mesh-based modeling for simulating volcanic ash plumes, which could lead to more accurate predictions of ash dispersal. Afrasiabi et al. [AKRW21] developed a method using SPH for more realistic simulation of metal cutting, producing lifelike chips and deformations under physical and thermal stresses. Lastly, Mahallem et al. [MRKG22] designed a weakly compressible SPH system to model interactions between landslides and bodies of water, incorporating both Newtonian and non-Newtonian fluid dynamics.

3 Overview

This implementation of a SPH water simulation is based off of the Navier-Stokes fluid equation[MCG03]

$$\rho\left(\frac{\partial v}{\partial t} + v \cdot \nabla v\right) = -\nabla p + \rho g + \varepsilon \nabla^2 v \tag{1}$$

$$F = -\nabla p + \rho g + \varepsilon \nabla^2 v \tag{2}$$

This formula can be split up into three distinct components to create the force acted upon a particle. $-\nabla p$ is the total pressure force applied to a particle, ρg are external forces, and $\varepsilon \nabla^2 v$ is the viscosity force. All of these are summed to create the total force applied to a particle. As shown by Müller et al.[MCG03], all three components of equation can be calculated by substituting their them into one of the following formulas.

$$A\left(x_i\right) = \sum_j m_j \frac{A_j}{\rho_j} W(d, h) \tag{3}$$

with the gradient and Laplacian of A being

$$\nabla A\left(x_i\right) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(d, h) \tag{4}$$

$$\nabla^2 A\left(x_i\right) = \sum_j m_j \frac{A_j}{\rho_j} \nabla^2 W(d, h) \tag{5}$$

Where A is some scalar value representation for some field around the particle position x_i, j represents all other particles, m is the mass of a particle, ρ is a particles density W is some smoothing kernel, d is the distance between two particles, and h a smoothing radius. These three equations are used to represent their respective fields at x_i.

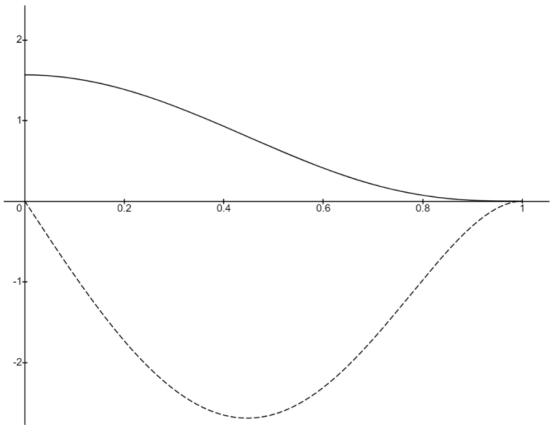

Figure 1: The poly6 smoothing kernel with the solid line being the kernel and the dashed line being its derivative with the smoothing length of h = 1

To obtain the acceleration we can use the following formula[MCG03].

$$a_i = \frac{Dv_i}{Dt} = \frac{F_i}{\rho_i} \tag{6}$$

A symplectic Euler integration is done with the acceleration to obtain the final velocity and position updates.

$$v_i(t + \Delta t) = v_i(t) + a_i \Delta t \tag{7}$$

$$x_i(t + \Delta t) = x_i(t) + v_i(t + \Delta t)\Delta t \tag{8}$$

From here the density of a particle must be found before any components of the Navier-Stokes fluid equation can be calculated.

3.1 Density

The density of a certain particle can be found by taking a sum of contributions from all neighbouring particles. From Müller et al.[MCG03] we substitute ρ for A in Equation 3

$$\rho_i = \sum_j m_j \frac{\rho_j}{\rho_j} W(d, h) = \sum_j m_j W_{\text{poly }6}(d, h) \tag{9}$$

With all particles in this implementation having a mass of 1, the density calculation ends up being a sum of the distances of all neighbouring particles adjusted by the smoothing kernel.

In order to help with stability, speed, and accuracy of the simulation, a smoothing kernel must be introduced. As Müller et al.[MCG03] describe, kernels along with their derivative that converge to 0 at the smoothing length are useful for increasing stability. The kernel that they designed exhibits this behaviour and can be seen in Figure 1. It is because of this that their density smoothing kernel was used, with values of the exponents tuned. The following function appears to give the best results in terms of particle stability.

$$W_{\text{poly6}}\,(d, h) = \frac{315}{64\pi h^9} \left(h^{1.6} - d^{1.6}\right)^4 \tag{10}$$

3.2 Pressure

With the density solved the next step is to solve for the pressure forces affecting each particle. Before any acting forces can be calculated, first the pressure for every particle must be found. Desbrun and Gascuel[DG96] describe that, unlike the original astronomical uses of smooth particle hydrodynamics, particles as fluids should also be affected by their density while at rest and suggest to use the following pressure calculation to replace the ideal gas law,

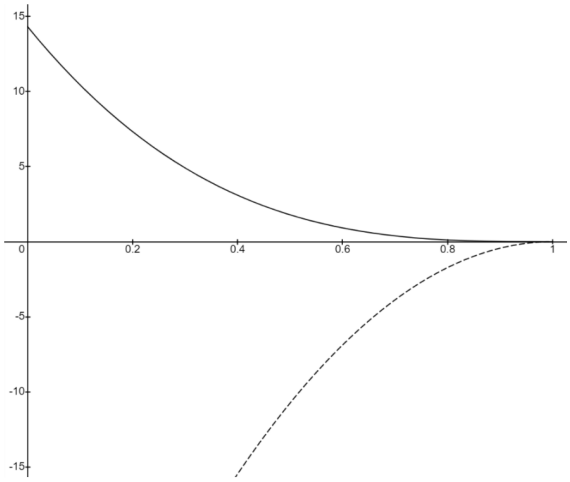

Figure 2: The poly6 smoothing kernel with the solid line being the kernel and the dashed line being its derivative with the smoothing length of h = 1

$$p_i = k\left(\rho_i - \rho_0\right) \tag{11}$$

Where k is some pressure constant and p_0 is the rest density. As pressure is a gradient, we can substitute $-\nabla p$ into Equation 4 to get the force applied to a particle,

$$F_i^{\text{pressure}} = -\nabla p_i = -\sum_j m_j \frac{p_j}{\rho_j} \nabla W_{spiky}(d,h) \tag{12}$$

However this approach does not work. As Müller et al.[MCG03] describe, the forces forces calculated in this way does not produce symmetric forces, thus violating conservation of momentum. Instead they propose the following formula to ensure symmetric forces are generated.

$$F_i^{\text{pressure}} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W_{\text{spiky}}\ (d,h) \tag{13}$$

Which was then modified to produce the force vector.

$$\vec{F}_i^{\text{pressure}} = F_i^{\text{pressure}}\ \hat{x_{ij}} \tag{14}$$

Many different functions were found and tested to find an appropriate kernel. The current implementation ended up using a design by Harada et al. [HKK07], which can be seen in Figure 2.

$$\nabla W_{spiky}(d,h) = \frac{45}{\pi h^6}(h-d)^3 \tag{15}$$

Compared to the kernel used for the density calculations, this kernel decays at and exponential rate and has the property of having a nonzero gradient at 0. This is important for pressure forces; as Müller etal.[MCG03] points out, a smoothing function without these properties will cause the repulsive forces to disappear and force particles to clump into groups as the pressure increases.

3.3 External Forces

The only external forces within the system are the force of gravity and any collisions particles have with environmental objects such as walls. The gravitational force is simply added to the sum of forces affecting the particle. The implementation within the Unity game engine was used to simplify the collision detection. Unity keeps track of all collisions that occur to rigid body objects. This makes it so that all particles can be assigned a rigid body property and check for any contact with objects that have a collider property. As particles should be able to avoid contact with each other through the fluid forces, the Unity project settings were changed so that particles should only collide with the

environment. When a collision does occur a simple reflection and dampening of the particles velocity is calculated and applied..

3.4 Viscosity

Viscosity can be considered a form of dampening. It's analogous to frictional forces can be thought of as how resistant a fluid, or particles within said fluid, are to flowing. To find the force of the viscosity force affecting the particles $\varepsilon\nabla^2 v$ can be substituted into Equation 5

$$F_i^{\text{viscosity}} = \varepsilon \sum_j m_j \frac{v_j}{\rho_j} \nabla^2 W_{\text{viscosity}}\ (d, h) \tag{16}$$

Again Müller et al.[MCG03] point out the this result leads to asymmetric forces being calculated. They suggest that because viscosity depends purely on velocity of the two particles, a simple way of creating a symmetric force is to take the relative velocity instead. We then get the following equation for the force vector

$$\vec{F}_i^{viscosity} = \varepsilon \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla^2 W_{viscosity}(d, h) \hat{x_{ij}} \tag{17}$$

$$F_i^{viscosity} = \varepsilon \sum_j m_j \frac{v_j - v_i}{\rho_j} \nabla^2 W_{viscosity}(d, h) \tag{18}$$

For the smoothing kernel, a design used by Müller et al.[MCG03] and Harada et al. [HKK07] was applied. This kernel itself is as follows.

$$W_{\text{viscosity}}\ (d, h) = \frac{15}{\pi h^3} \left(-\frac{d^3}{2h^3} + \frac{d^2}{h^2} + \frac{h}{2d} - 1 \right) \tag{19}$$

In order to utilize this kernel for the viscosity calculations we must the the Laplacian of it, giving us

$$\nabla^2 W_{\text{viscosity}}\ (d, h) = \frac{45}{\pi h^6} (h - d) \tag{20}$$

This smoothing kernel is used as the Laplacian remains positive everywhere within h, which can be seen plotted in Figure 3. As Müller et al.[MCG03] state, this kernel should only be able to have an damping effect on the velocity field, which is not guaranteed when using the other smoothing kernels. When using previous kernels there may be times where the instead of slowing the velocity of particles as they approach each other, they instead increase their velocities.

3.5 Neighbour Search

As the previous calculations involve taking sums from all neighbouring particles within a certain radius a method had to be designed to do this neighbour check. A brute force method would beto check all pairs of particles to ensure that they are close enough to each other to be considered a neighbour and then to calculate the densities, pressures, and forces. This approach, while simple to implement, comes with the computational complexity of $O\left(n^2\right)$, and becomes extremely inefficient as the number of particles grows.

In order to speed up the simulation a spatial grid was implemented to separate the space into a 3-dimensional grid, similar to the one described by Clavet et al[CBP05]. The grid cells each store a list of of all particles within their respective cell which updates every loop of the simulation. This reduces the amount of particle pairs need to be checked in the neighbour check from calculating all pairs of particles to just calculating a particle's current grid cell and the particles in range within the surrounding 26 cells and reduces the complexity to $O(\text{mn})$.

As the density calculation is required to be done before any other calculations can be computed, we can further improve the the efficiency of the simulation by computing the density in tandem with the neighbour check. References to neighbours for each particle can then be saved for later when the pressure and force calculations need to be computed. The final loop for the simulation can be seen in Algorithm 1.

Execution time of Functions			
Particles	Neighbour Search	Pressure Calculation	Pressure Force
500	17960	50	3322
1000	58287	100	8913
1500	116800	159	16141
2000	190119	210	24005
2500	282017	260	34745

Figure 3: Average execution times for function calls measured in ticks $\left(10^{-5}\right.$ milliseconds).

Algorithm 1 Main simulation loop
for all particles x_i do
for all particles x_j in surrounding cells do
if x_j in neighbour radius then
Compute ρ_i
Add x_j to list of neighbours
end if
end for
Compute p_i
for all neighbours x_n do
Compute F_i^{pressure}
Compute $F_i^{\text{viscosity}}$
end for
Update position of x_i
end for

(21)

4 Evaluation

The simulation was created in the Unity Game Engine using version 2019.2.15f1. An initial 2D prototype was implemented before creating the 3D version. Withing the final 3D simulation, particles were rendered as spheres with radius of size 0.2 within the simulations space. A number of scenarios were run, which include filling a space with fluid using a stream of particles and the dam break scenario.

4.1 Grid Size and Neighbour Radius

Many different sizes for both the size of the grid cells and the size of the neighbour check radius were tested. Grid cells values range from size 0.1 to 1 and neighbour radius ranged from 0.3 to 1. When both values approach 1 performance quickly degrades as more particles must be checked to see if they are considered neighbours and more calculations must be done. When setting the grid size to 0.1, the small amount of neighbours causes the particles to hang more in the air. The best settings found for the grid was to set it to the size of the particle, in this case 0.2, and setting the neighbour radius to 0.4. This will allow all particles within the surrounding cells to be considered neighbours while still having good performance.

4.2 Runtime of functions

To find the average time each takes for the computations in each step of the loop, the Unity's internal stopwatch function was started when a function was called and then stopped when exited. This process was run over 30 seconds of the simulation to get an average time in ticks. The usage of ticks, which are 10^{-5} milliseconds, were used as some function calls are extremely quick and return a time of 0 when using milliseconds. The results can be seen in Figure 3 and Table 4.

Execution time of Functions			
Particles	Viscosity Force	External Forces	Particle Update
500	3651	67	1191
1000	9915	133	2349
1500	17948	196	4066
2000	26793	261	5421
2500	38872	325	7754

Figure 4: More average execution times for function calls.

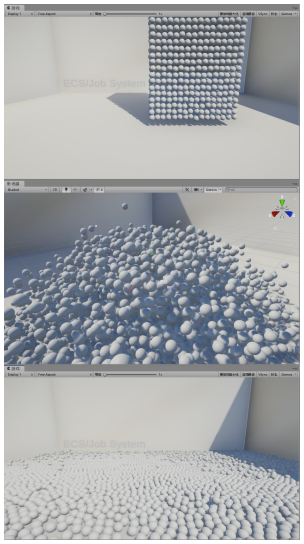

Figure 5:

As expected the shortest function calls are the pressure calculations and external force calculations, which only involve a single pass over all the particles. Pressure and viscosity force calculations, while still taking a moderately long time in comparison, are shortened as during the neighbour search step each particle saves a list of references to all particles considered neighbours. The neighbour search is the longest, as each particle must check every other particle in it's own grid cell as well as every particle in all the surrounding grid cells.

4.3 Performance as particles increase

The average frame rate was taken with particles counts increasing at intervals of 500. The set up has the spatial grid cell set at 0.2 , pressure constant $k = 3$, rest density $\rho_0 = 8$, and viscosity constant $\varepsilon = 0.02$. As the amount of particles increase, the frame rate decays at an exponential rate. Past 10000 particles the simulation begins to exhibit erratic behaviour. The system becomes unstable, as particles collide their velocities increase dramatically and shoot outside of the simulation space.

4.4 Dam break simulation

A common setup done with fluid simulations is the dam break. This involves creating an initial state of a tightly packed tower of particles, all of which have no velocity. This initial condition emulates a body of water at rest confined within a dam. When the system is run the particles fall freely within the simulation space, as if the dam has broken away. A dam break scenario was tested using 10000 particles arranged in a tower, seen in Figure 5 . As the system evolves over time the tower of particles collapses, and acting as a body of fluid, begins to rush horizontally to the other side of the simulation space. The fluid reaches the end of the simulation space and collides with the wall of the bounding box, which causes the velocity to redirect upwards.

5 Further Work

Future improvements may include implementing what Clavet et al.[CBP05] call the "double density relaxation". This process differentiates neighbour particles into the categories of neighbour and close neighbour. Force contributions are then measured differently depending on whether particles are regular or close neighbours. Surface tension forces can be added to further increase the stability and realism of particles at the surface of the fluid.

Currently the simulation is restricted to a preset simulation space. This space is defined beforehand and cannot be changed as the simulation runs. By implementing the hashed version of the spatial grid that Clavet et al.[CBP05] used the simulation spacecan be unbounded, allowing for situations where the fluid can flow freely throughout spaces unconfined by the predetermined grid.

Visuals of the liquid can be improved from a collection of particles to a full body of fluid. Muller et al.[MCG03] and Clavet et al.[CBP05] utilized a marching cube algorithm designed by Lorensen and Cline[LC98]. This will march through the simulation space in a fixed grid in order to derive the surfaces of cell. Values for each cell can reference a lookup table to then render the appropriate triangle needed to form the surface.

6 Conclusion

This paper presented a SPH implemented within the Unity game engine. With Unity, the problems of rendering and collision detection were simplified through using its internal systems. Using the Navier-Stokes equation as a base and building upon it with smoothing kernels and a spatial grid, it is able to achieve running simulations of 5000 particles with the upper limit of to 10000 particles. Much can still be improved in regards to speed, stability, and visuals.

References

[AKRW21] Mamzi Afrasiabi, Hagen Klippel, Matthias Röthlin, and Konrad Wegener. An improved thermal model for sph metal cutting simulations on gpu. *Applied Mathematical Modelling*, 100:728–750, 2021.

[CBP05] Simon Clavet, Philippe Beaudoin, and Pierre Poulin. Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 219–228, 2005.

[CPB+18] Zhixuan Cao, Abani Patra, Marcus Bursik, E Bruce Pitman, and Matthew Jones. Plume-sph 1.0: a three-dimensional, dusty-gas volcanic plume model based on smoothed particle hydrodynamics. *Geoscientific Model Development*, 11(7):2691–2715, 2018.

[DG96] Mathieu Desbrun and Marie-Paule Gascuel. Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation'96: Proceedings of the Eurographics Workshop in Poitiers, France, August 31–September 1, 1996*, pages 61–76. Springer, 1996.

[FLR10] Joshua Faber, Jamie Lombardi, and Fred Rasio. Starcrash: 3-d evolution of self-gravitating fluid systems. *Astrophysics Source Code Library*, pages ascl–1010, 2010.

[GM77] Robert A Gingold and Joseph J Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly notices of the royal astronomical society*, 181(3):375–389, 1977.

[HKK07] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Smoothed particle hydrodynamics on gpus. In *Computer Graphics International*, volume 40, pages 63–70. SBC Petropolis, 2007.

[LC98] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Seminal graphics: pioneering efforts that shaped the field*, pages 347–353. 1998.

[Luc77] Leon B Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal, vol. 82, Dec. 1977, p. 1013-1024.*, 82:1013–1024, 1977.

[MCG03] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159. Citeseer, 2003.

[MRKG22] Abderrahmane Mahallem, Mohamed Roudane, Abdelkader Krimi, and Sid Ahmed Gouri. Smoothed particle hydrodynamics for modelling landslide–water interaction problems. *Landslides*, 19(5):1249–1263, 2022.

[Ree98] William T Reeves. Particle systems—a technique for modeling a class of fuzzy objects. In *Seminal graphics: pioneering efforts that shaped the field*, pages 203–220. 1998.

[YWH+09] He Yan, Zhangye Wang, Jian He, Xi Chen, Changbo Wang, and Qunsheng Peng. Real-time fluid simulation with adaptive sph. *Computer Animation and Virtual Worlds*, 20(2-3):417–426, 2009.