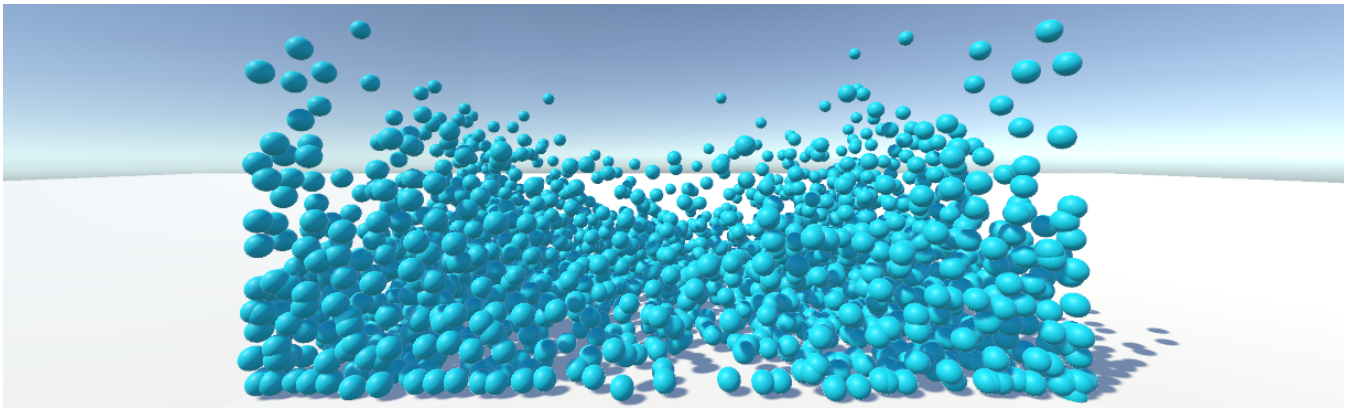
Smoothed Particle Hydrodynamics: A Real Time Implementation in the Unity Game Engine

Wong, Chris



Abstract

Smoothed particle hydrodynamics (SPH) is a popular way of modeling fluids by representing it as a group of particles. Originally used to study astronomical interactions, it has branched out to cover a wide range of uses including fluid simulation. Here smoothed particle hydrodynamics is implemented within the Unity game engine in an attempt to produce a real-time liquid simulation. Using the Navier-Stokes equation, the fluid can be modeled by computing its components of pressure, density, and viscosity per particle. The current implementation is able to simulate 2500 particles at 30 frames per second, however much work can still be done to improve the speed, stability, accuracy, and visual representation of the liquid.

CCS Concepts

• ***Computing methodologies*** → ***Physical simulation;***

1. Introduction

Modeling fluids can cover a large range of situations and materials with the most straightforward being liquids and gasses, however many other examples may not be as obvious. Some geological hazards, such as landslide and avalanches; astronomical collisions, such as planets colliding; and even fire can all be modeled partly or entirely as a fluids.

There are two major methods that are used when designing a fluid simulation. The first approach is the Eulerian method. This method divides up the problem space into discrete cells where flow of a fluid can be calculated between cells. Flow can then be represented as a velocity field in the grid and can be used to easily enforce constraints such as incompressibility. The underlying mathematics behind Eulerian fluids assume that the fluid involved is an inviscid, as such results from Eulerian fluids are excellent for simulating water, gasses, and properties such as diffusion, but as are unable to properly replicate honey and other non-Newtonian fluids.

Lagrangian fluids is the other major method. Instead of discretizing space, Lagrangian fluids discretize the fluid itself into a group of particles. This can be thought of as splitting the fluid into groups, or packets, of fluid. Each particle has their own set of values for properties, example being each particle has its own velocity, and will the able to influence other particles within the system in order to simulate a fluid. Lagrangian fluids can easily solve problems as guaranteeing conservation of mass by cause of being a particle system, but can have issues that Eulerian fluids solve such as the problem of incompressibility.

More methods, some of which take aspects of both previously mentioned methods, can be found such and the affine particle-in-cell (APIC) method, particle-in-cell/fluid-implicit-particle

(PIC/FLIP) method, and the material point method (MPM), with MPM becoming a popular for its usage in the 2013 film Frozen where it was used to simulate snow effects. In this approach the Lagrangian method of smoothed particle hydrodynamics (SPH) was used to simulate a body of liquid.

2. Related Work

The initial methods for SPH was first worked on by Gingold and Monaghan[GM77] in 1977. They developed this method to help model oscillating or non spherical stars. Later in the same year Lucy[Luc77] would further go on to use this approach and apply it to the progression and evolution of protostars. In 1983 Reeves[Ree83] presented a technique for modeling a class of objects he called fuzzy objects, which would include objects such as fire, clouds, water, and even grass. Dubbed "fuzzy" because of their complex shapes, Reeves would create a system of particles to represent these objects instead of the traditional polygon methods. As the system evolved over time, new particles would be introduced and old particles would be destroyed.

In regards to fluid simulations, Müller et al.[MCG03] proposed a method of fluid simulation that derives its approach by modeling the Navier-Stokes fluid equations. Clavet et al.[CBP05] integrated springs into their SPH implementation in order to create viscoelsatic fluids to add a more realistic elasticity and plasticity to non-Newtonian fluids. Harada et al.[HKK07] and Yan et al[YWH*09] presented a SPH solutions that utilized the GPUs to speed up the intensive computations that are required, with the latter being able to achieve 66 frames per second with 16,000 particles

In more modern applications, SPH continues to be popular method of modeling problems in many fields. Faber et al.[FLR10] used SPH in order to simulate interactions between stellar objects. These interactions include simulating the merger of stars and collisions of astronomical bodies. Cao et al.[CPB*18] describes that using SPH can be used to obtain more accurate results when simulating volcanic ash plumes when compared to traditional mesh based modeling, which will allow for more accurate forecasts involving the ash plume. Afrasiabi et al.[AKRW21] created a method to more accurately model and simulate the effects of metal being cut. This model can produce realistic chips and deformations of metals as a result of physical and thermal forces. Mahallem et al.[MRKG22] design a weakly compressible SPH system to model forces between landslides and bodies of water. This involves modeling the interactions between a Newtonian and non-Newtonian fluids.

3. Overview

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

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

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

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.

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

with the gradient and Laplacian of A being

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

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

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.

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

$$\rho_i = \sum_j m_j \frac{\rho_j}{\rho_j} W(d,h) = \sum_j m_j W_{poly6}(d,h) \tag{9}$$

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.

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

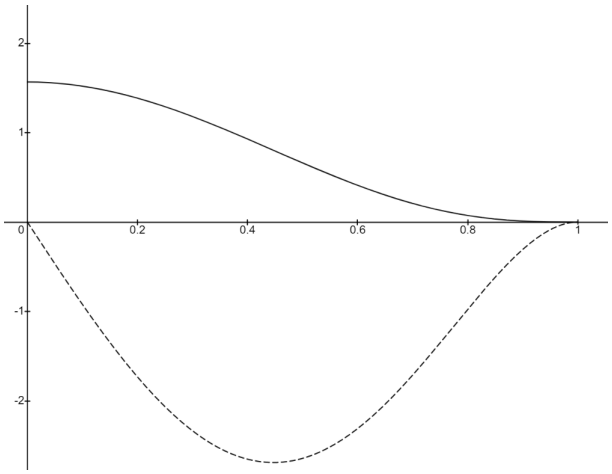


Figure 1: *The poly6 smoothing kernel with the solid line being the kernel and the dashed line being its derivative with the smoothing length of h = 1*

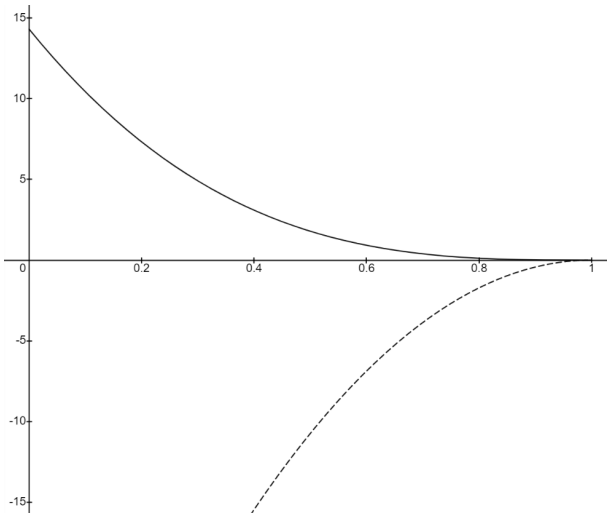


Figure 2: *The spiky smoothing kernel with the solid line being the kernel and the dashed line being its derivative with the smoothing length of h = 1*

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,

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

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,

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

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.

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

Which was then modified to produce the force vector.

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

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 non-zero gradient at 0. This is important for pressure forces; as Müller et al.[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

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

Again Müller et al.[MCG03] point out the this result leads to asymmetric forces being calculated. They suggest that because viscosity

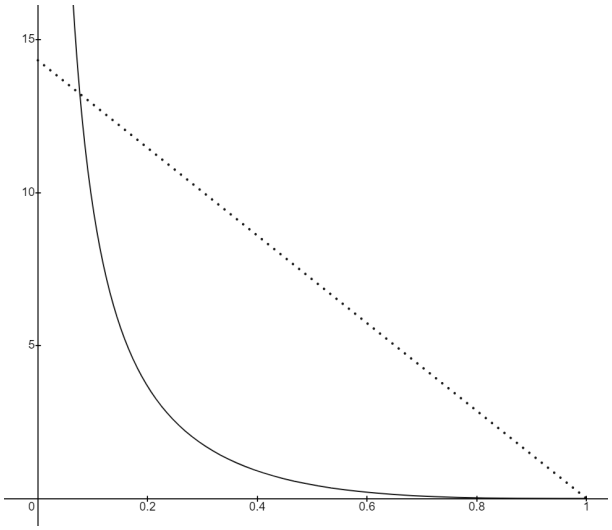


Figure 3: *The viscosity smoothing kernel with the solid line being the kernel dotted line being its Laplacian with the smoothing length h = 1*

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.

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

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

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

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 be

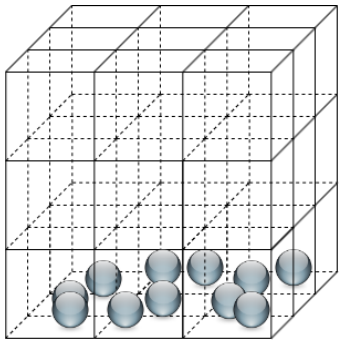


Figure 4: *Representation of the spatial grid*

to 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 $O(n^2)$, 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 $O(mn)$.

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.

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^viscosity
    end for
    Update position of x_i
end for
```

4. Evaluation

All results and simulations were run on a PC with an Intel i7-13700K 3.4GHZ CPU, 32GB RAM, and a NVIDIA GeForce RTX 2080 TI GPU. The simulation was created in the Unity Game Engine using version 2021.3.15f1. 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 Table 1 and Table 2.

Table 1: *Average execution times for function calls measured in ticks (10^{-5} milliseconds).*

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

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.

Table 2: *More average execution times for function calls.*

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

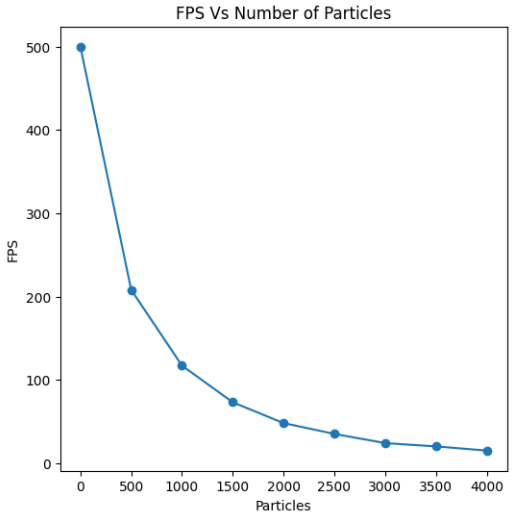


Figure 5: *Average frame rate measured with increasing amounts of particles.*

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 $\varepsilon = 0.02$. Seen in Figure 5, as the amount of particles increase, the frame rate decays at an exponential rate. Past 4000 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 2500 particles arranged in a $10 \times 25 \times 10$ tower, seen in Figure 6. 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.

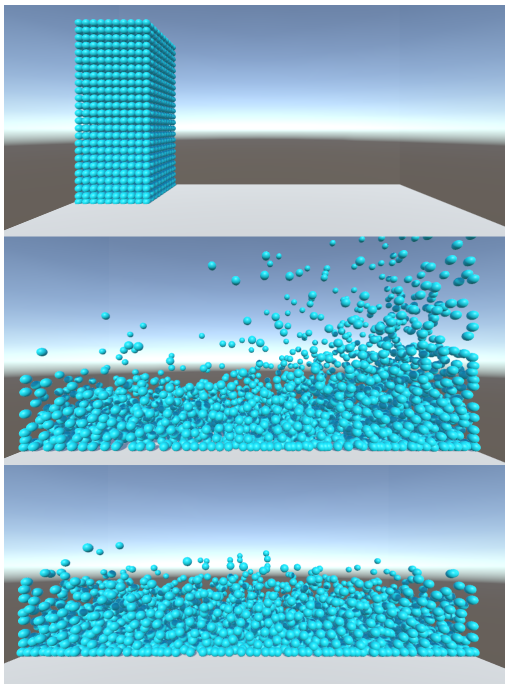


Figure 6: *A dam break simulation with 2500 particles. The top image shows the initial state. The middle image shows the fluid gathering to the other side of the simulation. The bottom image shows the system settling.*

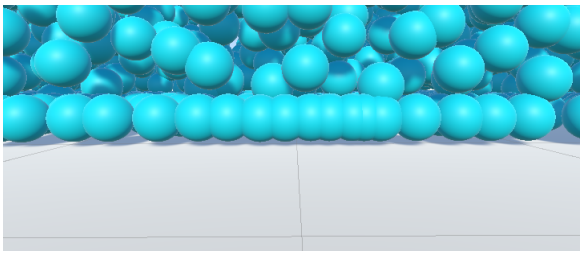


Figure 7: *Particles intersecting with each other.*

The fluid generally acts as expected, however issues remain. As seen in the bottom image of Figure 6 individual particles continue to bounce out of the main body of fluid, seemingly at random. This may be the result of viscosity forces being unable to counteract the build up of pressure forces from many particles acting upon a single particle. Other issues can be seen when taking a closer look at particles along the edge of the simulation space, such as in Figure 7. When many particles are in the in the simulation some of them along the edge get compressed to the point where they begin to intersect one another. This is caused by the pressure force not being able to overcome the compressive forces being applied, however increasing the pressure constant k tends to make the simulation even more unstable.

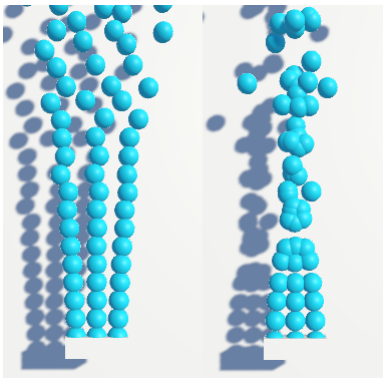


Figure 8: *Jets creating water particles. On the left* $\rho_0 = 8$ *and on the right* $\rho_0 = 200$

4.5. Changes in simulation variables

The rest density was adjusted to see how it affects the simulation. The first test was done in a setup where multiple particle jets are set up to spawn particles with an initial velocity at a fixed rate. Using Unity's inspector panel, the rest density was adjusted from $\rho_0 = 8$ to $\rho_0 = 200$. As Figure 8 shows, the rest density increase causes the particles to converge into each other.

Lowering the pressure constant has directly causes the particles to stick closer together, similar to how the particles become compact in Figure 7. This also has the effect that more particles become neighbours of each other causing the performance to drop. Raising the pressure constant increases the rest distance between particles, but also increases the pressure forces applied. After a certain threshold the simulation becomes unstable and particles begin to escape the simulation space. This threshold differs depending on various factors including the number of particles and the size of bounding box.

The viscosity force counteracts some of the repulsive forces generated from the pressure calculations. If the viscosity constant is set to a very large value the force becomes large enough to cause particles to launch each other at high speeds.

5. Further Work

Problems regarding stability and incompressibility remain in the current implementation. 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 space

can 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[LC87]. 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 2500 particles at 30 FPS with the upper limit of to 4000 particles. Much can still be improved in regards to speed, stability, and visuals.

References

[AKRW21] AFRASIABI M., KLIPPEL H., ROETHLIN M., WEGENER K.: An improved thermal model for sph metal cutting simulations on gpu. *Applied Mathematical Modelling 100* (2021), 728–750.

[CBP05] CLAVET S., BEAUDOIN P., POULIN P.: Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2005), pp. 219–228.

[CPB*18] CAO Z., PATRA A., BURSIK M., PITMAN E. B., JONES M.: Plume-SPH 1.0: a three-dimensional, dusty-gas volcanic plume model based on smoothed particle hydrodynamics. *Geoscientific Model Development 11*, 7 (July 2018), 2691–2715. doi:10.5194/gmd-11-2691-2018.

[DG96] DESBRUN M., GASCUEL M.-P.: Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation '96* (Vienna, 1996), Boulic R., Hégron G., (Eds.), Springer Vienna, pp. 61–76.

[FLR10] FABER J., LOMBARDI J., RASIO F.: Starcrash: 3-d evolution of self-gravitating fluid systems. *Astrophysics Source Code Library* (2010), ascl–1010.

[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.

[HKK07] HARADA T., KOSHIZUKA S., KAWAGUCHI Y.: Smoothed particle hydrodynamics on gpus. In *Computer Graphics International* (2007), vol. 40, SBC Petropolis, pp. 63–70.

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics 21*, 4 (1987), 163–169.

[Luc77] LUCY L. B.: Numerical approach to the testing of the fission hypothesis. *Astron. J.; (United States) 82:12* (12 1977). doi:10.1086/112164.

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on computer animation* (2003), pp. 154–159.

[MRKG22] MAHALLEM A., ROUDANE M., KRIMI A., GOURI S. A.: Smoothed particle hydrodynamics for modelling landslide–water interaction problems. *Landslides 19*, 5 (2022), 1249–1263.

[Ree83] REEVES W. T.: Particle systems—a technique for modeling a class of fuzzy objects. *ACM Trans. Graph. 2*, 2 (apr 1983), 91–108. doi:10.1145/357318.357320.

[YWH*09] YAN H., WANG Z., HE J., CHEN X., WANG C., PENG Q.: Real-time fluid simulation with adaptive sph. *Computer animation and virtual worlds 20*, 2-3 (2009), 417–426.