

CS 384P Final Project Report

Fluid Simulation with Smoothed Particle Hydrodynamics

May 3, 2024

Katie Clark
kac5676

Marco Ravelo
mar7655

1 Background

1.1 Introduction to SPH

Smoothed Particle Hydrodynamics (SPH) is a method of modelling the behavior of incompressible fluids using the Navier-Stokes equations. These equations describe the movement of incompressible fluids and are defined as:

$$\rho \dot{\mathbf{u}} = -\rho(\nabla \cdot \mathbf{u})\mathbf{u} - \nabla \mathbf{p} + \nu \nabla^2 \mathbf{u} + \mathbf{F}$$

$$\nabla \cdot \mathbf{u} = 0$$

where $\mathbf{u}$ represents the velocity of the fluid, ρ is the density, $\mathbf{p}$ is the pressure, ν is the coefficient of viscosity, and $\mathbf{F}$ represents external forces acting on the fluid (for our simulation, this would be gravity) [1]. From these equations, there are three main components to simulating a fluid with SPH:

1. Density: which describes the concentration of particles in a certain area.
2. Pressure: which describes the force exerted by particles in different concentrations of particle density.
3. Viscosity: which describes the fluid's resistance to deformation and is determined by the difference in velocities of neighboring particles.

1.2 Goals

The goals of this project are as follows to accurately simulate incompressible fluid motion and behavior using the SPH method in both 2D and 3D and to allow a user to interact with the fluid with both mouse input as well as deforming the fluid's container.

2 Implementation

Our implementation relies on iteratively solving for the forces acting on each particle (like in the cloth project) rather than taking a linear algebra based approach (as in the other projects). Both the 2D and 3D simulations utilize the same techniques described below.

Algorithm 1 Simulation Update Step

```
function SIMULATESTEP
    CalculateDensity()
    CalculatePressure()
    CalculateViscosity()
    for Particle  $i$  in particles do
        Update position and velocity of  $i$ 
        Apply velocity constraint
        Apply boundary constraint
    end for
    for Particle  $i$  in particles do
        Update adjacency/neighbor list of particle  $i$ 
    end for
end function
```

2.1 Particles and Neighbors

We began our implementation by creating a class to store information about the particles in the script `Particle.cs` and a separate struct to represent information about the other particles within a given particle’s “radius of support,” or the particles within a certain distance, [2] in the script `Neighbor.cs`.

Each particle holds information about itself including, current position $\mathbf{q}$, previous position $\mathbf{q}_{\text{prev}}$, current velocity $\mathbf{q}_{\text{dot}}$, total force acting on it $\mathbf{F}$, radius, mass, density, pressure, and a list of neighboring particles.

The `Neighbor` struct contains a reference to the particle represented by the struct as well as its weighted distance and squared weighted distance to the particle the struct is in relation to.

2.2 Density

The density ρ of a particle i can be approximated using a kernel function W over the neighboring particles j within the predefined radius of support h :

$$\rho_i = \sum_j m_j \cdot W(|\mathbf{r}_{ij}|, h)$$

where m_j is the mass of particle j , $|\mathbf{r}_{ij}|$ is the distance between particles i and j , and function W defines the weight/influence of a particle as a function of their distance from i [2].

In the script `SimController.cs`, we iterate over each particle i and each of its neighbors j . Within the loop, we calculate the weight or contribution of each neighboring particle to the total density of particle i using weighted distance from i to j .

Particles that are within i ’s radius of support are added as neighbors of i in its adjacency list and their weighted distances are stored. The contributions from all of i ’s neighboring particles are summed to compute the density of particle i .

Our implementation, following the method used in tutorial [1], our implementation computes two separate, weighted densities. The first, `density`, is weighted with the squared distance from i to j while the other, `adj_density`, is weighted with the cubed distance. This allows us to create greater pressure force on the particles nearest to i in

later calculations while still allowing particle i to exert some lessened force on particles that are further away.

2.3 Pressure

The pressure P at each particle i is determined based on its density ρ compared to a predefined rest density ρ_{rest} :

$$P_i = k \sum_j (W(|\mathbf{r}_{ij}|, h) \cdot ((\rho_i - \rho_{\text{rest}}) + (\rho_j - \rho_{\text{rest}})))$$

where k is a fluid constant [2].

In the script `SimController.cs`, we again iterate over each particle i and each of its neighbors j . The pressure between particles i and j is computed based on the pressure difference between rest density and current density as well as the distance between the two particles. This effectively represents the gradient of pressure in the fluid.

The pressure force exerted by a particle i is iteratively calculated by incrementing the total force at i by the combined influences of each of its neighbors. Because this is a somewhat simplified approach using a state equation with local values, the fluid may behave as a weakly compressible fluid rather than being fully incompressible [2].

2.4 Viscosity

The viscosity, or resistance to the fluid's flow and deformation, of a fluid in our SPH simulation is created by calculating a viscosity impulse. This impulse acts as an adjustment to the velocity of each particle based on the velocities of its neighbors. This is modelled by:

$$I = \sum_j \frac{m_j}{\rho_j} \cdot \sigma (\mathbf{v}_j - \mathbf{v}_i) \cdot \beta W(|\mathbf{r}_{ij}|, h)$$

where I is the viscosity impulse, $\mathbf{v}_j - \mathbf{v}_i$ is the relative velocity between particles i and j , and σ and β are constants of viscosity [2].

In the same script, `SimController.cs`, we again iterate over each particle i and each of its neighbors j . We calculate the contribution to the viscosity impulse, I , using the difference between the velocities of i and j , the distance between the two particles, and the direction of the vector from i to j .

Following tutorial [1], we also scale the components of the viscosity impulse using two tunable constants, `SIGMA` for the linear term and `BETA` for the quadratic term. These values allow us to create more detailed variations in viscosity to simulate different fluid materials.

2.5 Boundary Constraints

In order for the particles to behave properly as a fluid, we constrain it within a boundary. This boundary consists of a an axis-aligned *rect* in 2D and an axis-aligned *cuboid* in 3D. We allow for the modification of the boundary during run-time in which users can both move and scale the dimensions of the container.

The boundary consists of two parts: (1) an arrangement of *ghost* particles along the edges of the rect (and along the faces of the cuboid) which are dynamically spawned in,

and (2) a fall-back constraint which simply restricts particles from passing through the designated boundary.

The *ghost* particle technique, which was implemented from [2], is quite simple. Particles are spawned along the boundary at regular intervals in order to prevent the normal particles from passing through. These *ghost* particles remain fixed and are not updated in the simulator. However, they are considered when calculating density and pressure, repelling the normal particles inward into the boundary. Multiple layers of *ghost* particles can be used to better constrain the normal particles (as is done in 2D with 4 layers, but not in 3D).

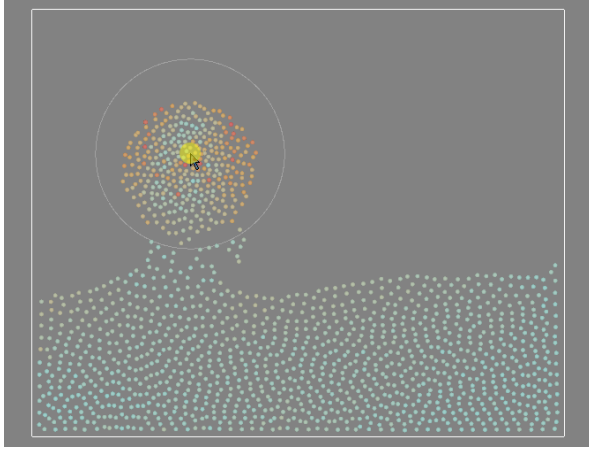
There is still the possibility for normal particles to squeeze through the *ghost* particle layers if their velocity is too large or if the boundary is modified too rapidly. As such, a fall-back constraint was implemented which detects any particles that make it past the boundary, setting their position to be within the boundary and negating their velocity so as to “reflect” the particle off the boundary wall.

2.6 Mouse Input

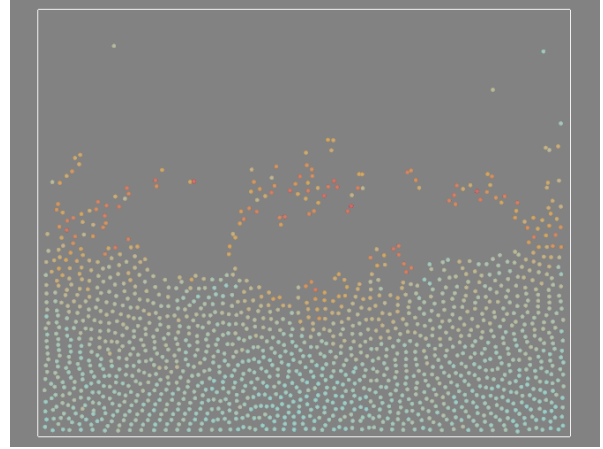
We implemented a simple means for the user to interact with the fluid simulation in real-time. Clicking in the application window creates a *force* in world-space which attracts particles within a certain radius, allowing the user to “pick-up” sections of the fluid. This was implemented for both the 2D and 3D fluid simulations.

References

- [1] Pelfrey, B. (2009). Real-Time Physics 103: Fluid Simulation in Games.
- [2] Koschier, D., et al. (2019). Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids. Eurographics.

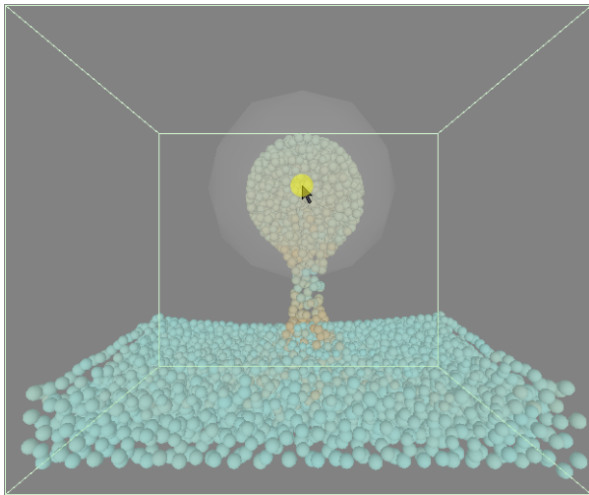


(a) Using mouse controls

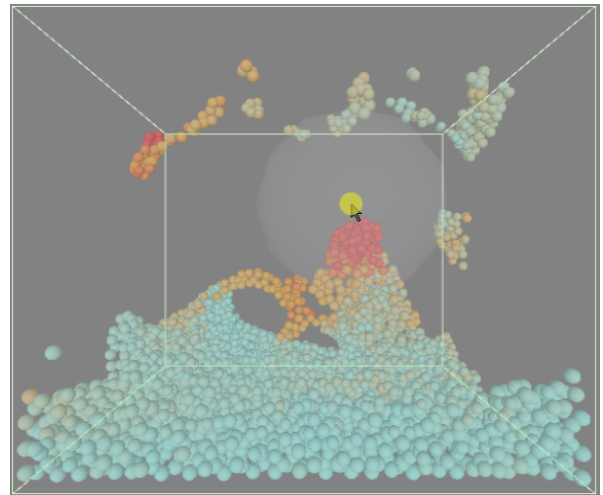


(b) Shortly after dropping the fluid

Figure 1: Figures showcasing our 2D fluid simulator using 2048 particles.



(a) Picking up some fluid



(b) Perturbing the fluid

Figure 2: Figures showcasing our 3D fluid simulator using 4096 particles.