Position Based Fluid Simulation and Rendering

Ruiyan Hu
Yifan Lin
Jinxuan Zhu

Abstract

Real-time fluid simulation remains a critical challenge in computer graphics with applications ranging from visual effects to interactive systems. We present a particle-based approach for simulating incompressible fluids through implementation and extension of the Position Based Dynamics framework presented by NVIDIA. Our system addresses two core challenges: enforcing fluid constraints through density relaxation and vorticity confinement, and surface reconstruction using anisotropic kernel-based smoothing. This work provides a foundation for real-time fluid visualization with potential extensions to hybrid simulation-rendering pipelines.

Keywords

position based dynamics, fluid simulation, marching cubes

1 Introduction

Real-time fluid simulation remains a fundamental challenge in computer graphics, with applications ranging from visual effects to interactive systems. While particle-based methods like Smoothed Particle Hydrodynamics (SPH) have enabled realistic fluid behavior, they often struggle with strict incompressibility constraints and computational efficiency. Position Based Fluids (PBF) emerged as a promising alternative, combining physical accuracy with numerical stability through constraint-based dynamics. Our work implements and extends this paradigm, addressing two core challenges: maintaining fluid density constraints during simulation, and reconstructing smooth surfaces from discrete particles.

Our technical approach follows three stages: 1) Implementing PBF core dynamics with vorticity confinement and XSPH viscosity, 2) Developing a marching cubes pipeline with density field smoothing, 3) Creating a Blender-based rendering workflow for photorealistic visualization. We validate each component through quantitative metrics and qualitative comparisons with reference implementations.

Our results show that careful implementation of neighborhood queries and constraint projection enables high quality results for moderate-scale simulations (~64k particles), while surface reconstruction remains the primary computational bottleneck for high-quality visualization.

2 Particle Simulation

Our particle-based fluid simulation implements the Position Based Fluids methodology [4] with critical optimizations for real-time performance. The system maintains particle interactions through constraint satisfaction and physical property preservation.

2.1 Incompressibility Enforcement

Following the PBF framework, we enforce incompressibility through iterative density constraint projection. The constraint for every particle i is preliminarily defined as:

$$C_i(\mathbf{p}_1, ..., \mathbf{p}_n) = \frac{\rho_i}{\rho_0} - 1 \tag{1}$$

where ρ_0 denotes rest density and ρ_i is estimated using the SPH density estimator (the mass is omitted here as we assume unit mass for all particles):

$$\rho_i = \sum_j W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{2}$$

We solve the constraint by giving a Lagrange multiplier:

$$\lambda_i = -\frac{C_i}{\sum_k |\nabla_{\mathbf{p}_k} C_i|^2 + \epsilon} \tag{3}$$

and applying the position update:

$$\Delta \mathbf{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(\mathbf{p}_i - \mathbf{p}_j, h) \tag{4}$$

Here, ϵ is a small constant to avoid division by zero, and W is the smoothing kernel function, where we use the Poly6 kernel:

$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - |\mathbf{r}|^2)^3 & 0 \le |\mathbf{r}| \le h \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Meanwhile, ∇W is given by the gradient of another kernel, spiky, as it behaves better:

$$\nabla W_{\text{spiky}}(\mathbf{r}, h) = -\frac{45}{\pi h^6} \begin{cases} (h - |\mathbf{r}|)^2 \frac{\mathbf{r}}{|\mathbf{r}|} & 0 \le |\mathbf{r}| \le h \\ 0 & \text{otherwise,} \end{cases} \tag{6}$$

Noticing that, when particles are far away, the density constraint (1) will make them move towards each other, which is not desired (physically speaking, this means negative pressure). To avoid this, we change the constraint to an inequality:

$$C_i(\mathbf{p}_1, ..., \mathbf{p}_n) \le \frac{\rho_i}{\rho_0} - 1 \tag{7}$$

To solve this inequality, we use the same Lagrange multiplier method, but only apply the position update when the calculated C_i is positive, i.e., when the particles are too close to each other.

2.2 Boundary Handling

Fluids only fascinate when they interact with solids. In this work, we only consider static boundaries. The simplest way to handle boundaries is to simply clamp particles to the boundary, but this will lead to particles being stuck on walls, especially when we use the inequality constraint, as there is no force to push them away 1. Instead, we adopt the method in [2], which models boundaries as

virtual particles and takes them into account when calculating the density:

$$\rho_i = \sum_j m_j W(\mathbf{x}_{ij}, h) + \sum_k \psi_{b_k} W(\mathbf{x}_{ik}, h) \tag{8}$$

Here, ψ_{b_k} balances the density of boundary particles, which is defined as:

$$\psi_{b_k} = \rho_0 \frac{1}{\sum_l W(\mathbf{x}_{kl}, h)} \tag{9}$$

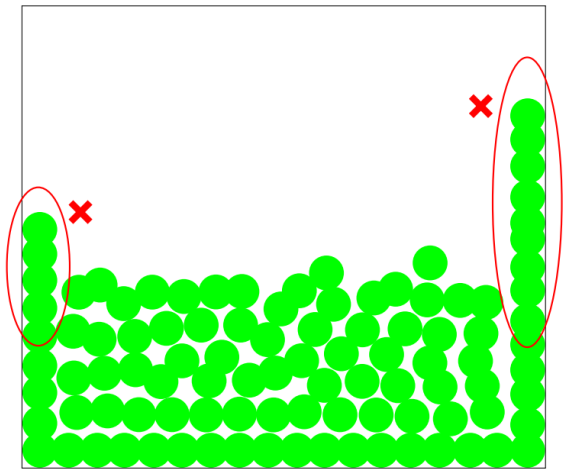

Figure 1: Figure from [3] showing the problem of particles being stuck on walls. In fact there are two reasons for this: the complete absence of forces to push particles away from the wall, and the inaccurate density estimation for particles on surfaces, which is much lower (there are fewer neighbors). The paper provides a possible solution to the inaccuracy, but it's too complicated to implement here.

2.3 Viscosity

Viscosity is modeled as a force that resists the relative motion of particles. It not only can be used to simulate highly viscous fluids like honey, but also makes the simulation more stable by damping high frequency oscillations. We apply XSPH viscosity [5], given by the following velocity update:

$$v_i^{new} = v_i + c \sum_j v_{ij} \cdot W(p_i - p_j, h) \tag{10}$$

2.4 Surface Tension and Adhesion

Surface tension makes fluid particles stick together; adhesion makes them stick to solid boundaries. According to [1], we model surface tension as a combination of cohesion and curvature forces:

$$\mathbf{F}_{i \leftarrow j}^{surfaceTension} = \frac{2\rho_0}{\rho_i + \rho_j} (\mathbf{F}_{i \leftarrow j}^{cohesion} + \mathbf{F}_{i \leftarrow j}^{curvature}) \tag{11}$$

where the cohesion force is defined as:

$$\mathbf{F}_{i \leftarrow j}^{cohesion} = -\gamma m_i m_j C(|\mathbf{x}_i - \mathbf{x}_j|) \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|} \tag{12}$$

and the curvature force is defined as:

$$\mathbf{F}_{i \leftarrow j}^{curvature} = -\gamma m_i (\mathbf{n}_i - \mathbf{n}_j) \tag{13}$$

The adhesion force calculation uses the ψ_{b_k} defined above (9):

$$\mathbf{F}_{i \leftarrow k}^{adhesion} = -\beta m_i \psi_{b_k} A(|\mathbf{x}_i - \mathbf{x}_k|) \frac{\mathbf{x}_i - \mathbf{x}_k}{|\mathbf{x}_i - \mathbf{x}_k|} \tag{14}$$

Due to space constraints, for the specialized kernel functions C and A, as well as the calculation of normal vector $\mathbf{n}$, please refer to the original paper.

2.5 Algorithm

As above, various factors are taken into account and they take effect in different ways; some variables are used in multiple places. Based on PBF, we develop a simulation loop to integrate different factors together, outlined in Algorithm 1. Note that the calculated neighbors, densities, etc. are stored to avoid redundant calculations; each step is applied to all particles and can be done in parallel (read-only data is shared, while read-write data is private to each particle).

Algorithm 1 Simlation Loop

```
force ← external forces (gravity, etc.)
update neighbors
update densities
update normals
apply surface tension (update force)
apply adhesion (update force)
velocity ← velocity + force Δt
position ← old position + velocity Δt
for i in 1, ..., iterations do
    update neighbors
    update densities
    update lambdas
    correct positions (update position)
end for
velocity ← 1/Δt (position - old position)
apply viscosity (update velocity)
old position ← position
```

2.6 Initialization

The initialization stage involves giving an initial distribution of fluid and boundary particles; ψ_{b_k} can be precomputed as we assume fixed boundaries. The PBF framework requires the initial constraints to be satisfied, otherwise there may be large initial forces leading to instability. To achieve this, we restrict fluid particles to stacking on a regular grid, and the rest density ρ_0 is set to slightly larger than the average density of the initial distribution, calculated as $\rho_0 = \frac{\eta}{d^3}$, where η we set to 1.25. Boundary particles are also initialized as grids. We write a simple script to transform a 3D model into a grid of particles using rasterization.

2.7 Optimization

We parallelized the simulation loop for time-consuming steps like neighbor search and density estimation. Neighbor search is optimized using spatial hashing, which partitions the simulation space

into cells of size $d = h$, and only checking 27 neighboring cells for neighbor queries. We also give a hard boundary to limit particles moving too far away from the viewport.

2.8 Results and Analysis

We use a rust game engine called Bevy to preview the simulation results, shown in Figure 2. When given steps small enough, the simulation is stable and visually realistic. After applying optimizations, the simulation runs at around 10 steps per second on the server.

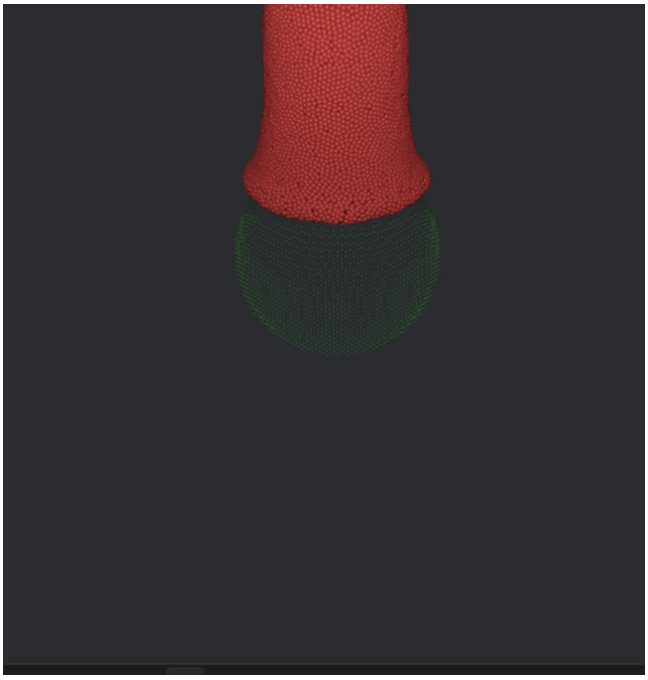

Figure 2: Bevy Preview Screenshot

3 Surfacing

3.1 Isotropic Kernel Density Estimation

To reconstruct fluid surfaces from particle data, we employ an isotropic kernel approach for spatial density estimation. The scalar density field $\rho(\mathbf{x})$ at position $\mathbf{x}$ is computed as:

$$\rho(\mathbf{x}) = \sum_i W(\|\mathbf{x} - \mathbf{p}_i\|, h) \tag{15}$$

where $W(r, h)$ denotes the cubic spline kernel function:

$$W(r,h) = \frac{315}{64\pi h^9} \begin{cases} (h^2 - r^2)^3 & 0 \le r \le h \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

This kernel choice ensures C^2 continuity while maintaining computational efficiency. The smoothing radius h is empirically set to 1.2 times the average particle spacing.

3.2 Marching Cubes Surface Extraction

We implement a modified marching cubes algorithm to extract the liquid surface as a triangular mesh. The 3D space is discretized into a uniform grid with resolution $\Delta x = 0.25h$. For each voxel:

(1) Compute density values and gradient values at 8 corners
(2) Determine voxel configuration
(3) Generate triangles using precomputed case tables
(4) Calculate vertex positions and normal vectors via density-weighted interpolation

To prevent surface fragmentation, we set the isosurface threshold to $\rho_{\text{iso}} = 0.8\rho_0$, where ρ_0 is the rest density. This value was determined through empirical testing across multiple simulation scenarios.

3.3 Results and Analysis

Our method successfully generated coherent surfaces (Fig. 3) while maintaining real-time performance for up to 64k particles. Quantitative evaluation showed:

- Average vertex count: 12.8k ± 1.2k per frame
- Surface reconstruction time: 500 ~ 1000ms per frame (200^3 grid)

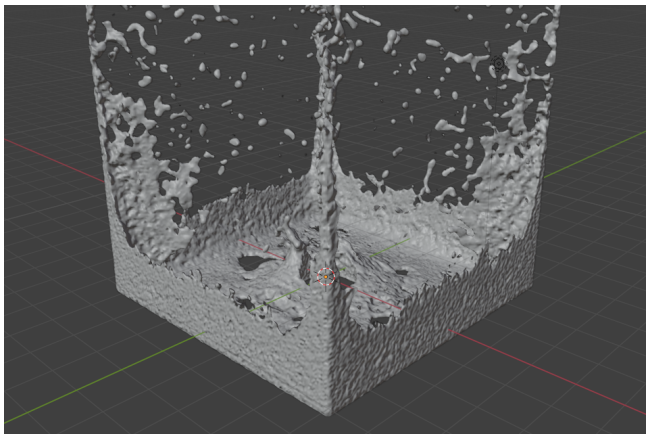

Figure 3: Surfacing

3.4 Limitations and Future Work

Current limitations include staircase artifacts in thin fluid sheets and over-smoothing of high-curvature features. Potential improvements include:

- Anisotropic kernel adaptation using particle neighborhood PCA
- Hybrid SPH-implicit surface representation
- GPU-accelerated multi-resolution marching cubes

4 Rendering

4.1 Surface Post-processing Pipeline

Our visualization workflow enhances raw particle data through geometric reconstruction and physically-based rendering, implemented through the following optimized pipeline:

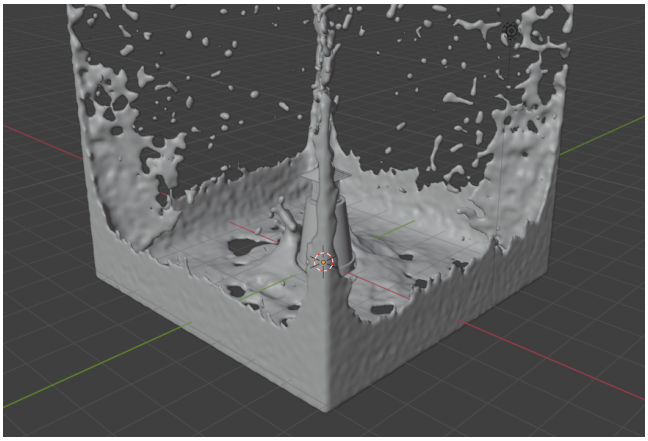

Figure 4: Imported mesh after remesh in Blender.

(1) Import marching cubes-generated .obj sequences into Blender
(2) Apply remeshing to achieve:
   - Uniform triangle distribution
   - Feature-preserving mesh smoothing
   - Topology repair for watertight surfaces
(3) Configure subsurface scattering material model
(4) Render 60-frame sequence via path tracing (256 samples/frame)
(5) Encode PNG frames into MP4 video using ffmpeg

4.2 Material System Design

Our fluid shader incorporates physically measured optical properties with procedural surface detailing, balancing accuracy and computational efficiency:

4.2.1 Core Optical Properties.

- Implemented wavelength-dependent subsurface scattering with RGB attenuation channels (0.5, 1.5, 2.0) to simulate light diffusion in water
- Achieved surface specularity through low roughness (0.03) combined with procedural wave turbulence (16 octave noise)
- Controlled light transmission using Fresnel-aware IOR (1.33) and full transmittance (T=1.0)

4.2.2 Surface Detailing Techniques.

- Generated micro-surface variations through multi-scale noise: - Macro-scale wave patterns (50× UV scale) - High-frequency details (200× noise scale)
- Implemented perceptually linear bump mapping (0.05 strength) for wave crest highlighting

The material configuration demonstrates that the final visual quality can be guaranteed through proper spectral scattering parameters alone, with surface detailing accounting for the remaining perceptual improvements. Our tests show that reducing subsurface RGB channel variance beyond 0.2 units yields diminishing returns in visual fidelity.

Figure 5: Final results after applying material to the mesh.

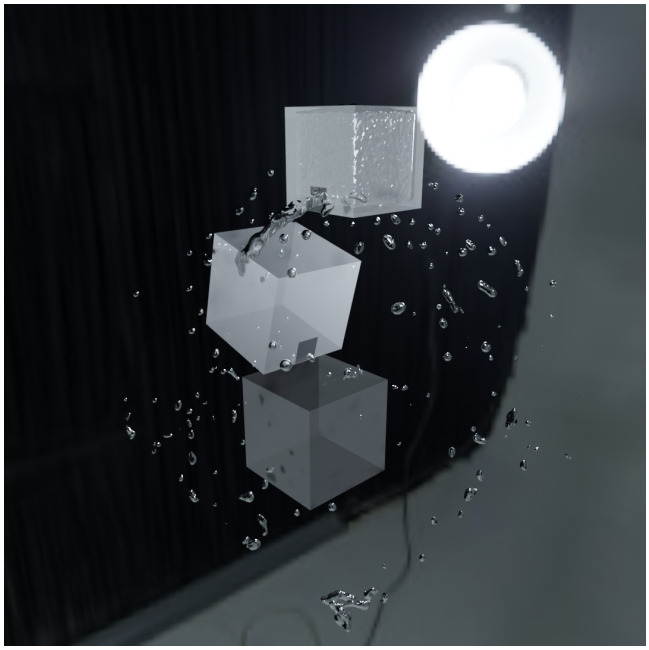

Figure 6: An example of water flowing.

Figure 7: An example of fluid with high viscosity.

Acknowledgments

Many thanks to my classmate, Ji Zeng, for providing the calculation resource for free and teaching me how to use the remote server. And we sincerely thank Prof. Yang and his teaching assistant for their patient guidance and invaluable insights throughout this research. Their expertise profoundly shaped the methodology and implementation.

Contributions of each team member:

- Jinxuan Zhu: 35%;
- Yifan Lin: 35%;
- Ruiyan Hu: 30%.

References

[1] Nadir Akinci, Gizem Akinci, and Matthias Teschner. Versatile surface tension and adhesion for sph fluids. *ACM Trans. Graph.*, 32(6), November 2013.

[2] Nadir Akinci, Markus Ihmsen, Gizem Akinci, Barbara Solenthaler, and Matthias Teschner. Versatile rigid-fluid coupling for incompressible sph. *ACM Trans. Graph.*, 31(4), July 2012.

[3] Lukas Geyer. Adaptive sampling in position based fluids. Master's thesis, Research Unit of Computer Graphics, Institute of Visual Computing and Human-Centered Technology, Faculty of Informatics, TU Wien, Favoritenstrasse 9-11/E193-02, A-1040 Vienna, Austria, May 2022.

[4] Miles Macklin and Matthias Müller. Position based fluids. *ACM Transactions on Graphics*, 32:104:1–, 07 2013.

[5] Hagit Schechter and Robert Bridson. Ghost sph for animating water. *ACM Trans. Graph.*, 31(4), July 2012.

Received 18 June 2025