Real-time Position Based Fluid Simulation and Interaction Using Taichi Framework

ZIYI CAI, Zhiyuan College, SJTU, China
YECHEN XU, Zhiyuan College, SJTU, China
LEJUN MIN, Zhiyuan College, SJTU, China

Fluid simulation has been a challenge in Computer Graphics domain. In this project we implemented Macklin and Müller's algorithm specified in their paper [Macklin and Müller 2013]. To achieve real-timeness, we utilized Taichi framework to render the fluid in real time, which is proven to be robust and handy comparing to CUDA programming. We also raised several interactive ways to enhance its playability. In order to have a better realistic view, foam simulation is also added into our scene. Generally, our work can be regarded as a re-implementation of the algorithm with augmentations in rendering, gaming and interaction. In the end, we also prove with statistics that the algorithm is efficient and robust.

Additional Key Words and Phrases: Fluid Simulation, Particle Simulation, Real-time Rendering, Interactive Scene

ACM Reference Format:
Ziyi Cai, Yechen Xu, and Lejun Min. 2022. Real-time Position Based Fluid Simulation and Interaction Using Taichi Framework. *ACM Trans. Graph.* 1, 1 (November 2022), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Overview

Fluid simulation is an important topic in modern computer graphics field as it can produce many visually rich phenomena. There are lots of works aim to simulate fluids without the expense of efficiency or realistic look.

There are a variety of techniques available for fluid simulation. Here we focus on particle-based methods, where the fluids are represented by a cluster of single particles, and for each round of computing, we update the positions and velocities of these particles in order to form a overall realistic movement of the fluids. Particle-based methods are popular due to their simplicity and flexibility.

In physics, fluids can be divided into two categories: compressible fluids and incompressible fluids. Liquids, for example water, fall into the latter category. Achieving robust incompressibility is crucial for rendering realistic liquids. In the original paper [Macklin and Müller 2013] we refer to, the authors raised a method based on Position Based Dynamics (PBD) framework [Müller et al. 2007].

Apart from the simulation part, interaction is pivotal for both evaluating and further developments like games. It remains an open topic for game engineers and researchers to design possible ways of interfering with the fluids.

1.2 Related Works

Smoothed Particle Hydrodynamics (SPH) [Monaghan 1992] is a well known particle based method for fluid simulation. However SPH is sensitive to density fluctuations from neighborhood deficiencies, and enforcing incompressibility is costly due to the unstructured nature of the model. Müller [Müller et al. 2003] showed that SPH can be used for interactive fluid simulation with viscosity and surface tension by using a low stiffness equation of state (EOS). However to maintain incompressibility, standard SPH require stiff equations, resulting in large forces that limit the time-step size.

There are hybrid methods like Fluid Implicit Particle (FLIP) [Brackbill et al. 1988] which use a grid for the pressure solve and transfer velocity changes back to particles. This work is extended later to incompressible flow with free surfaces ([Zhu and Bridson 2005]). Raveendran et al. [Raveendran et al. 2011] use a coarse grid to solve for an approximately divergence free velocity field before an adaptive SPH update.

Clavet et al. [Clavet et al. 2005] also use a position based approach to simulate viscoelastic fluids. However, because the time step appears in various places of their position projections, their approach is only conditionally stable as in regular explicit integration.

Position Based Dynamics [Müller et al. 2007] provides a method for simulating dynamics in game based on Verlet integration. By eschewing forces, and deriving momentum changes implicitly from the position updates, the typical instabilities associated with explicit methods are avoided.

In the main paper we refer to [Macklin and Müller 2013], Macklin and Müller raised a method to achieve incompressibility inside PBD framework. By addressing the issue of particle deficiency at free surfaces, and handling large density errors, their method allows users to trade incompressibility for performance, while remaining stable.

1.3 Rationale

The work of Macklin and Müller has provided us with a realistic and robust fluid simulation method. Yet during the development of Computer Graphics in the recent years, new languages and frameworks have been created to achieve simplicity in implementation as well as real-timeness thanks to GPU acceleration. Taichi [Hu et al. 2019] for example, serves as an excellent framework for developing GPU accelerated applications without sacrificing easiness. Moreover, novel ways of interactions should be taken into consideration if an algorithm is going to be used in the game industry. Therefore, we implemented Macklin's algorithm on Taichi framework, and suggested several interesting ways to interact with the scene.

Authors' addresses: Ziyi Cai, Zhiyuan College, SJTU, Shanghai, China; Yechen Xu, Zhiyuan College, SJTU, Shanghai, China; Lejun Min, Zhiyuan College, SJTU, Shanghai, China.

Apart from that, in order to have a more realistic rendering of fluids, we refer to a common approach of foam [Scherzer et al. 2018] and implement it in our project scene. This is also a redemption of realisticism in our particle-restricted rendering.

1.4 Methodology

To simulate a physically realistic fluid, we should preserve the incompressibility property and prevent loss of details such as vortices, viscositiy and foams.

We mainly follow the approach of [Macklin and Müller 2013] in the simulation process. For foam rendering we learn from [Scherzer et al. 2018] and build a simplified version for our scene.

To measure the results of our simulations, we examine the convergence rate and the maximum divergence with respect to the rest density. Meanwhile, since we focus on the real-time rendering effect, we also require the performance to be acceptable at the standard of animation.

1.5 Main Contribution

In this project, we first confirmed the robustness of Macklin's algorithm by achieving real-time rendering using Taichi framework, and we raised several novel ways of interaction. We also add foam simulation to it. The final scene should be informative and fun to play around.

2 IMPLEMENTATION

2.1 Position Based Simulation

2.1.1 Position Based Dynamics. Enforcing incompressibility is crucial for realistic fluid simulation. To achieve this, we should first develop an effective way to estimate the density given the position based fluid.

Borrowing the ideas from the standard SPH, the density estimator is defined to be

$$\rho_i = \sum_j m_j W(\mathbf{p}_i - \mathbf{p}_j, h). \tag{1}$$

To achieve constant density, we follow [Bodin et al. 2011] and solve a system of non-linear constraints with one constraint per-particle.

$$C_i(\mathbf{p}_1, \dots, \mathbf{p}_n) = \frac{\rho_i}{\rho_0} - 1, \tag{2}$$

where ρ_0 is the rest density.

In each round of iteration, the position based dynamics method aims to find a correction $\Delta\mathbf{p}$ satisfying $C(\mathbf{p} + \Delta\mathbf{p}) = 0$. To use a series of Newton steps to approximate the continuous process, it's sufficient to solve

$$C(\mathbf{p}) + \nabla C^T \nabla C \lambda = 0. \tag{3}$$

[Monaghan 1992] gives the SPH recipe for the gradient of a function defined on position based particles. Applying this and plugging the result into (3) we have

$$\lambda_i = -\frac{C_i\left(\mathbf{p}_1, \dots, \mathbf{p}_n\right)}{\sum_k |\nabla_{\mathbf{p}_k} C_i|^2}. \tag{4}$$

2.1.2 Constraint Force Mixing. When particles are close to separating, the denominator in equation (4) will approach zero and cause instability. Constraint force mixing (CFM) [Smith 2004] is a method to regularize the constraint. By mixiong in some of the constraint force back into the constraint function, we can obtain a more robust value

$$\lambda_i = -\frac{C_i(\mathbf{p}_1, \dots, \mathbf{p}_n)}{\sum_k |\nabla_{\mathbf{p}_k} C_i|^2 + \varepsilon}, \tag{5}$$

and a soft constraint correction

$$\Delta\mathbf{p}_i = \frac{1}{\rho_0} \sum_j (\lambda_i + \lambda_j) \nabla W(\mathbf{p}_i - \mathbf{p}_j, h). \tag{6}$$

2.1.3 Tensile Instability. On the free surface of the fluid, a particle has only a few neighbors and is unable to satisfy the density constraint. This will cause negative pressures, which leads to clustering and clumping.

Following the approach of [Monaghan 2000], we include an artificial pressure

$$s_{corr} = -k \left(\frac{W(\mathbf{p}_i - \mathbf{p}_j, h))}{W(\Delta\mathbf{q}, h)} \right)^n \tag{7}$$

in the particle position update as

$$\Delta\mathbf{p}_i = \frac{1}{\rho_0} \sum_j \left(\lambda_i + \lambda_j + s_{corr}\right) \nabla W \left(\mathbf{p}_i - \mathbf{p}_j, h\right). \tag{8}$$

2.1.4 Vorticity Confinement and Viscosity. Fluids have details such as vortex and viscosity. To preserve these detailed motions, it's necessary to apply some extra adjustment.

Lack of vorticity is due to energy lost in undesirable damping. We use vorticity confinement to replace the lost. [Monaghan 1992] gives an estimator for vorticity of a particle:

$$\omega_i = \sum_j \mathbf{v}_{ij} \times \nabla_{\mathbf{p}_j} W(\mathbf{p}_i - \mathbf{p}_j, h). \tag{9}$$

With the vorticity, we can calculate a corrective force for each particle:

$$\mathbf{f}_i^{vorticity} = \varepsilon^{vorticity} \left(\mathbf{N} \times \omega_i\right). \tag{10}$$

To reduce fluid oscillations which are not physically realistic, we apply XSPH viscosity [Schechter and Bridson 2012] to correct the velocity:

$$\mathbf{v}_i^{new} = \mathbf{v}_i + c \sum_j \mathbf{v}_{ij} \cdot W(\mathbf{p}_i - \mathbf{p}_j, h). \tag{11}$$

2.2 Visual Effects

We implemented our work with Taichi framework [Hu et al. 2019], with which we can have easy access to GPU acceleration and cross-platform support abilities. We build our code from scratches in a Python-style code, end up being compiled by Taichi compiler into CUDA or Vulkan backend machine code.

2.3 Problems and Solutions

2.3.1 Detailed Setting. We use the Poly6 kernel for density estimation and the Spiky kernel for gradient calculation, as in [Müller et al. 2003].

The parameters are listed in Table 1.

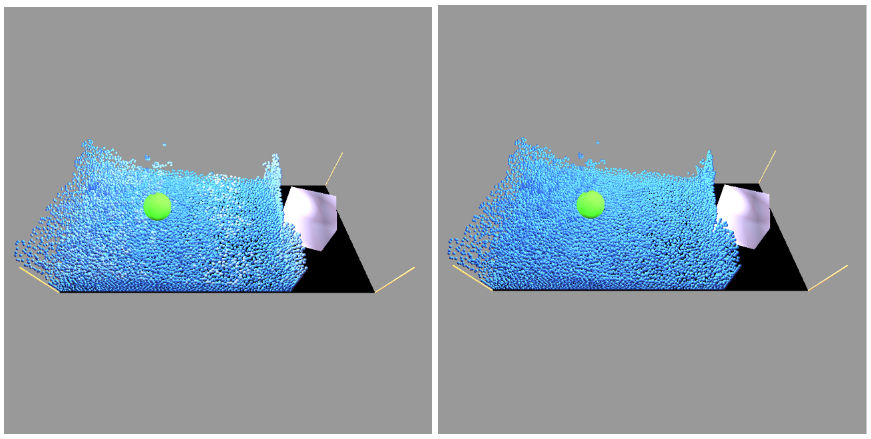

Fig. 1. Comparison between effects with or without foams

ε	10^2
k	10^{-4}
n	4
$\|\Delta q\|$	0.3h
$\varepsilon^{vorticity}$	10^{-2}
c	10^{-2}

Table 1. Parameters in the project.

2.3.2 Neighbour Finding. To calculate the estimators, we need to find neighbouring particles for each particle. To be precise, we need to find all particles in a sphere of radius h, the radius of the kernel function.

To accelerate this process, we use the grid hashing method, which divides the space into grids of equal volume. When searching for neighbours, we only enumerate the particles in the neighbouring grids. This optimization works well for our scene.

2.3.3 Performance. At the very beginning of our project, it was natural for us to write code on the basis of Scotty3D[15-462 2022]. However, out of education purpose, Scotty3D could not meet our standard of real-time simulation and rendering. After investigation, we decided to use Taichi framework, which has integrated native GPU feature support and has been widely used for efficient simulation.

2.3.4 Rendering. Although Taichi is a good framework for simulation, it contains only the basic support for rendering. Current version(1.0.2) of Taichi only support rendering particles and meshes in RGB, without alpha, channels. What's worse, because Taichi achieves fascinating performance by compiling Python source code into intermediate representation then native backend, we cannot modify Taichi as we modify third-party libraries.

There are two options left for us: export our simulated data and render them in external program like Blender; or we focus on the real-time feature and try to make simulation prettier within the range of Taichi's capability. After discussion, given that the real-time feature is more important to us, we decided to adopt the latter option.

In our early program, the overall visual effects were not satisfactory. In order to make our simulation in terms of rendering more realistic, we decided to find a way. In real world, foam appears on the surface of fluids. With foams added into our rendering, we can achieve a more desirable look.

To simulate this, we need to know the property of foams. According to [Scherzer et al. 2018], we must compute Weber number, by which we can determine whether a water is needed to be turned into foam by mixing air. The Weber number is computed by

$$We = \frac{\rho v^2 l}{\sigma}$$

where *rho* is the density of the liquid, v is the speed of the particle related to air (in our context air is stationary), l is the diameter of the particle and *sigma* is the surface tension of the liquid.

By comparing Weber number with a heuristic threshold, we decide whether a particle can be a foam particle, then give it proper color and internal property. The comparison is shown in figure 1.

2.4 Lessons Learned

By implementing our project, we gained a lot of experience. Thorough research in the early stage is key to efficient implementation. We wrote a lot of bugs, most of which were because we were not familiar with Taichi framework and missed vital part in Taichi's documentations. Also we can use techniques such as unit test in traditional software development to ensure we do all things, especially those formula-critic, right.

3 RESULTS

We tested our algorithm by half-filling a tank, which has an obstacle inside, with a special kind of fluid ($\rho_0 = 3$).

The convergence of our method over multiple iterations is in Figure 2 and Figure 3. The max density is not too far from the desired rest density and the average density converges at a rapid rate.

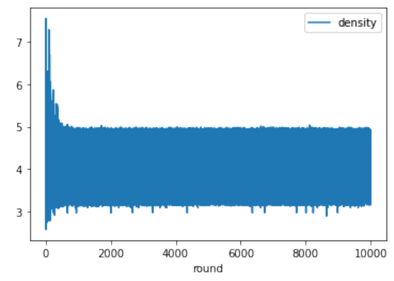

Fig. 2. Max density by round.

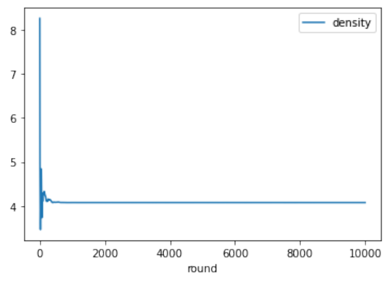

Fig. 3. Average density by round.

Our performance is also sufficient for real-time rendering scenario. With the paralleling support provided by Taichi framework, even when running our simulations on an NVIDIA MX250, the program can keep reaching 30fps.

Our final result shows in Figure 4 and external video. Just check them out.

4 FUTURE WORK

As we mentioned, although Taichi is a good framework for simulation, it has little space for advanced rendering. So we need to seek another platform for high quality real-time simulation and rendering. CUDA and Vulkan are both good choices, with which we can implement real-time rendering like screen space rendering.

Beyond that, currently we have only preset hand built scene and objects. We can also add support for importing external model for better scene and customization.

Note that for more complicated scenario, the number of particles we need will become extremely large. In that case, the naive grid hashing method will fail to meet our requirement. Much more time-and-space efficient data structures such as k-d tree should be introduced to accelerate the neighbour finding process.

Finally, the visual effects are not impressive enough yet. To achieve a both physically and visually realistic fluid rendering, we should apply some fluid surface constructing algorithm such as ellipsoid splatting [van der Laan et al. 2009].

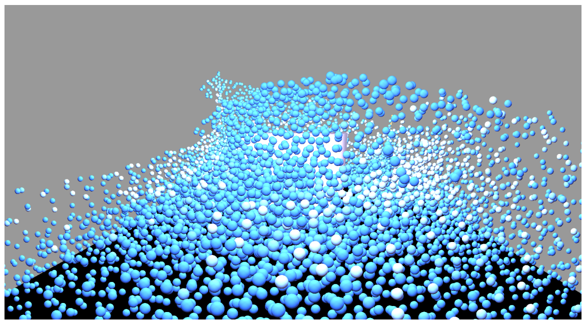

Fig. 4. Final image

5 CONCLUSION

Fluid simulation still attracts lots of attention both from researchers and industries. A fast, responsive and high-quality method is essential to so many applications. We confirmed the robustness of Macklin's algorithm, implemented real-time simulator and renderer using Taichi framework, and we raised novel ways for beautier rendering, improved running efficiency and finally provided a joyful interactive program for people.

REFERENCES

CMU 15-462. 2022. Scotty3D. [EB/OL]. https://github.com/CMU-Graphics/Scotty3D.

Kenneth Bodin, Claude Lacoursiere, and Martin Servin. 2011. Constraint fluids. *IEEE Transactions on Visualization and Computer Graphics* 18, 3 (2011), 516–526.

Jeremiah U Brackbill, Douglas B Kothe, and Hans M Ruppel. 1988. FLIP: a low-dissipation, particle-in-cell method for fluid flow. *Computer Physics Communications* 48, 1 (1988), 25–38.

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

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. 2019. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 201.

Miles Macklin and Matthias Müller. 2013. Position based fluids. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.

Joe J Monaghan. 1992. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics* 30 (1992), 543–574.

Joseph J Monaghan. 2000. SPH without a tensile instability. *Journal of computational physics* 159, 2 (2000), 290–311.

Matthias Müller, David Charypar, and Markus H Gross. 2003. Particle-based fluid simulation for interactive applications.. In *Symposium on Computer animation*, Vol. 2.

Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. 2007. Position based dynamics. *Journal of Visual Communication and Image Representation* 18, 2 (2007), 109–118.

Karthik Raveendran, Chris Wojtan, and Greg Turk. 2011. Hybrid smoothed particle hydrodynamics. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics symposium on computer animation.* 33–42.

Hagit Schechter and Robert Bridson. 2012. Ghost SPH for animating water. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–8.

Daniel Scherzer, Florian Bagar, and Oliver Mattausch. 2018. Volumetric real-time water and foam rendering. *GPU Pro 360 Guide to Rendering* (2018), 189–201.

Russell Smith. 2004. Open dynamics engine v0. 5 user guide. *http://ode. org/* (2004).

Wladimir J. van der Laan, Simon Green, and Miguel Sainz. 2009. Screen Space Fluid Rendering with Curvature Flow. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games* (Boston, Massachusetts) *(I3D '09)*. Association for Computing Machinery, New York, NY, USA, 91–98. https://doi.org/10.1145/1507149.1507164

Yongning Zhu and Robert Bridson. 2005. Animating sand as a fluid. *ACM Transactions on Graphics (TOG)* 24, 3 (2005), 965–972.

A CONTRIBUTIONS FROM TEAM MEMBERS

- Ziyi Cai (35%): most of the simulation algorithm.
- Yechen Xu (35%): rendering, part of simulation, miscellaneous, debugging and testing.

- Lejun Min (30%): interaction with scene object, part of simulation, video presentation.