

Implicit velocity projection method for incompressible SPH fluids

MIN HYUNG KEE, Korea University, Republic of Korea
JINHYUNG AHN, Yonsei University, Republic of Korea
SEHYUN PARK, Korea University, Republic of Korea
KIWON UM, LTCI, Telecom Paris, IP Paris, France
JUNGHYUN HAN, Korea University, Republic of Korea



Fig. 1. **Hollow Tree.** Fluid fills in a complex boundary, involving 978K fluid and 1.75M boundary particles. Given a tight density error tolerance of 0.01%, our solver resolves each frame of this challenging scenario in fewer than 5 iterations, each of which involves an inner linear system solver for the subproblem.

Enforcing incompressibility has been extensively studied in smoothed particle hydrodynamics (SPH) fluid simulation. A variety of methods have been developed to this end, particularly based on implicit formulations to ensure stability and efficiency, which results in iterative pressure solvers. However, these iterative solvers, typically adopting the projected relaxed Jacobi (PRJ) method for the constant density constraint, suffer from slow convergence, mainly due to under-relaxation required for stability. In this paper, we interpret the PRJ iteration as an explicit fixed-point update that projects velocity onto the constraint manifold, and propose its implicit counterpart, the *implicit velocity projection* method. Our method enables significant over-relaxation, which leads to fast convergence while maintaining robustness, thereby achieving substantial performance gains. Our comprehensive experiments, including studies on different solvers with varying relaxation parameters, timestep sizes, density-error tolerances, and scene scales, demonstrate the superiority of our method over standard SPH methods.

CCS Concepts: • **Computing methodologies** → **Physical simulation.**

Additional Key Words and Phrases: Smooth Particle Hydrodynamics, Incompressible Fluid Simulation

ACM Reference Format:

Min Hyung Kee, Jinhyung Ahn, Sehyun Park, Kiwon Um, and JungHyun Han. 2026. Implicit velocity projection method for incompressible SPH fluids. In *SIGGRAPH Asia 2026 Conference Papers (SA Conference Papers '26)*, December 01–04, 2026, Kuala Lumpur, Malaysia. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3829340.3842220>

Authors' Contact Information: Min Hyung Kee, Korea University, Seoul, Republic of Korea, minh702@korea.ac.kr; Jinhyung Ahn, Yonsei University, Seoul, Republic of Korea, ajhh98@yonsei.ac.kr; Sehyun Park, Korea University, Seoul, Republic of Korea, rth52@korea.ac.kr; Kiwon Um, LTCI, Telecom Paris, IP Paris, Palaiseau, France, kiwon.um@telecom-paris.fr; JungHyun Han, Korea University, Seoul, Republic of Korea, jhan@korea.ac.kr.



This work is licensed under a Creative Commons Attribution 4.0 International License. *SA Conference Papers '26, Kuala Lumpur, Malaysia*
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2842-6/2026/12
<https://doi.org/10.1145/3829340.3842220>

1 Introduction

SPH is widely used in computer graphics to simulate incompressible fluids, thanks to its versatility in handling free-surface flows and complex boundaries as well as extensibility for multi-material scenarios. Thus, improving the stability and efficiency of the SPH simulation method offers significant benefits to many computer graphics applications.

A central challenge in SPH simulation for incompressible fluids lies in enforcing the incompressibility condition, which is crucial for preventing undesired visual artifacts. While various approaches have been proposed to this end, the pressure Poisson equation (PPE) based approach has become dominant; the representative methods include predictive-corrective incompressible SPH (PCISPH) [Solenthaler and Pajarola 2009], implicit incompressible SPH (IISPH) [Ihmsen et al. 2013], and divergence-free SPH (DFSPH) [Bender and Koschier 2015]. These methods, in essence, iteratively project the intermediate (also known as predicted) velocity onto the density constraint manifold, often using the first-order approximation, i.e., linearization, of the constraint function.

Solving the PPE is typically performed using the PRJ method. However, the underlying Jacobi method often suffers from inherent slow convergence. The under-relaxation is widely adopted to ensure numerical stability [Band et al. 2018; Gissler et al. 2019; Ihmsen et al. 2013], yet it restricts the fast projection of the solver.

This paper tackles the stability and convergence issues of the PPE solver, introducing an *implicit velocity projection* method. We first interpret the velocity update of prior PPE solvers as an explicit, fixed-point iteration and modify it into an implicit one. This enables significant over-relaxation, thus the velocity is stably projected toward the feasible state much faster. Unlike prior projection methods iterating PRJ steps until convergence, our formulation defines a sequence of subproblems, each being solved using an efficient iterative linear system solver. The sequence of velocity updates driven by the solution of each subproblem, then, incrementally projects

the velocity state onto the density constraint manifold linearized in velocity. Thanks to the implicit nature, our new formulation leads to significant improvement in convergence rate and performance, presenting superior stability over the prior solvers.

2 Related Work

SPH simulations for incompressible fluids typically involve a set of steps. Among others, enforcing incompressibility has received significant attention due to its crucial role in producing realistic fluid motion. Depending on how the incompressibility condition is formulated and which primary variables are updated during the solution procedure, we identify two major branches of incompressibility solvers: *velocity-based* and *position-based*. As our study targets the incompressibility solver in SPH, we will focus on the two below. For other parts, including viscosity [Solenthaler and Pajarola 2009; Takahashi et al. 2015; Weiler et al. 2018], surface tension [Akinci et al. 2013b; Jeske et al. 2023; Probst and Teschner 2024], and interactions with solid boundaries [Akinci et al. 2013a; Band et al. 2018; Bender et al. 2023], we refer to the comprehensive survey on SPH for computer graphics by Koschier et al. [2022].

Velocity-based methods. Since the seminal work of SPH for interactive applications [Müller et al. 2003], a wide range of incompressibility solvers has been developed for SPH. One of the earliest formulations is a penalty-based method, deriving pressure forces from an equation of state (EOS) and integrating them explicitly in time, as in weakly compressible SPH (WCSPH) [Becker and Teschner 2007]; however, the required high stiffness for desired incompressibility makes such methods highly sensitive to the timestep size. To overcome this limitation, projection-based methods, coupled with implicit time integration, solve the PPE that projects the velocity onto the constant density manifold linearized in velocity. Within this family, PCISPH [Solenthaler and Pajarola 2009] approximates the projection through pressure corrections scaled by a precomputed factor, while IISPH [Ihmsen et al. 2013] formulates a sparse linear system for pressure and solves it using PRJ. DFSPH [Bender and Koschier 2015] builds on the same PRJ-based PPE solver and exploits a secondary solve that enforces a divergence-free condition, improving convergence and stability. Recently, Probst and Teschner [2023] adopted a non-smooth nonlinear conjugate gradient scheme [Silcowitz-Hansen et al. 2010] that wraps PRJ to accelerate the pressure solve. Note that velocity-based methods exploit density constraints linearized in velocity; in other words, particle positions are fixed during the iterative projections per timestep. In this way, the density and its gradient are evaluated once at each simulation step, keeping each iteration cheap. However, as the timestep size grows, such a linearization potentially becomes a poor approximation of the end-of-step density, thus limiting the admissible timestep size. Our implicit velocity projection formulation is motivated by those limitations, aiming to improve stability in larger timestep sizes and efficiently enforce a strict incompressibility condition.

Position-based methods. Position-based fluid (PBF) derives the projection step for the constant density constraint using the particle positions [Macklin and Müller 2013]. This method directly updates particle positions during the iterative projections of the

solver. Here, the density and its gradient are recomputed every iteration, thus introducing additional per-iteration overhead, while maintaining superior stability even at large timestep sizes. In subsequent studies, as in projective fluids (PF) [Weiler et al. 2016], the incompressibility problem is reshaped as a penalty-based energy minimization problem on positions; consequently, the solver aims to search for a balance between advection and projection. PF uses local position projection as the per-particle step of its solver, formulating a stiff quadratic energy that penalizes position deviation from the locally projected positions. More recent methods construct the energy directly from the density deviation itself and solve the resulting nonlinear minimization either via semi-implicit successive substitution [He et al. 2025] or exploiting second-order information for improved stability [Diaz et al. 2025].

3 SPH for Incompressible Fluids

We focus on SPH simulations of incompressible fluids, whose governing equations, i.e., the incompressible Navier–Stokes equations, consist of the continuity equation and the momentum equation:

$$\frac{D\rho}{Dt} = 0 \iff \nabla \cdot \mathbf{v} = 0, \quad (1)$$

$$\rho \frac{D\mathbf{v}}{Dt} = -\nabla p + \mu \nabla^2 \mathbf{v} + \mathbf{f}, \quad (2)$$

where $\frac{D}{Dt}$ denotes the material derivative, ρ is the density, $\mathbf{v}$ is the velocity, p is the pressure, μ is the dynamic viscosity, and $\mathbf{f}$ is the external force.

To solve Eqs. (1) and (2), SPH typically takes the Lagrangian specification and discretizes the fluid domain with particles. A physical quantity at the position of particle i , i.e., $\mathbf{x}_i$, is, then, evaluated as:

$$A_i \approx \sum_j \frac{m_j}{\rho_j} A_j W(\mathbf{x}_i - \mathbf{x}_j, h), \quad (3)$$

where m_j and ρ_j are the mass and density of particle j , respectively, and $W(\cdot, h)$ is a smoothing kernel with support radius h . We will, henceforth, denote $W(\mathbf{x}_i - \mathbf{x}_j, h)$ as W_{ij} for notational simplicity.

Spatial derivative operators such as gradient, divergence, and Laplacian can be derived based on Eq. (3), and many variants of these operators have been proposed. For a detailed discussion of practices for various spatial derivatives in SPH, we refer to the survey by Koschier et al. [2022].

4 Constant Density Solver

This section reviews the iterative constant density solver, which is a governing concept of many SPH variants. This solver aims to find the global pressure force vector and eventually the velocity state vector update addressing the density error, i.e., particularly compression. We revisit a common particle-wise formulation practice and discuss the solver’s procedure in global state vector representation, which motivates our extension in Sec. 6.

4.1 Pressure Force

One of the fundamental quantities in SPH is the density. Following Eq. (3), the density at $\mathbf{x}_i$ is evaluated as

$$\rho_i = \sum_j m_j W_{ij}, \quad (4)$$

which gauges local compression and drives introducing pressure forces that resolve the deviation from the rest density ρ_0 .

Given the density value in Eq. (4), the pressure can be computed using an EOS, e.g.,

$$p_i = \kappa_i \frac{\rho_0}{\gamma} \left(\left(\frac{\rho_i}{\rho_0} \right)^\gamma - 1 \right), \quad (5)$$

where κ_i controls stiffness and γ is an exponent constant. We focus on a special case with $\gamma = 1$, which is commonly used in computer graphics [Bender and Koschier 2015; Desbrun and Gascuel 1996]:

$$p_i = \kappa_i (\rho_i - \rho_0). \quad (6)$$

It is worth noting that κ_i can be handled as unknown particle-wise pressure coefficients and hence solved to enforce constant density [Band et al. 2018; Ihmsen et al. 2013].

Adopting Eq. (6), the symmetric pressure force exerted on particle i $\mathbf{f}_i^p \in \mathbb{R}^3$ can be written as [Bender and Koschier 2015]

$$\mathbf{f}_i^p = -\frac{m_i}{\rho_i} \nabla p_i = -m_i \sum_j m_j \left(\frac{\kappa_i}{\rho_i} + \frac{\kappa_j}{\rho_j} \right) \nabla W_{ij}. \quad (7)$$

Global-vector representation. Let us denote the density evaluation of Eq (4) as a function of the global position state vector, which stacks all of the N particle positions (i.e., $\mathbf{x} \in \mathbb{R}^{3N}$). This leads to a density function $\rho_i(\mathbf{x}) : \mathbb{R}^{3N} \rightarrow \mathbb{R}$. Then, the pressure force in Eq. (7) is denoted in the global-vector representation, $\mathbf{f}^p \in \mathbb{R}^{3N}$ as

$$\mathbf{f}^p = -\sum_{i=1}^N \kappa_i \frac{m_i}{\rho_i(\mathbf{x})} \left(\frac{\partial \rho_i(\mathbf{x})}{\partial \mathbf{x}} \right)^\top, \quad (8)$$

where $\frac{\partial \rho_i(\mathbf{x})}{\partial \mathbf{x}} \in \mathbb{R}^{1 \times 3N}$ is the density gradient with respect to the global position state vector; see Appendix ?? for the derivation. It is worth noting that the k -th 3×1 block of $\left(\frac{\partial \rho_i(\mathbf{x})}{\partial \mathbf{x}} \right)^\top$ can be naturally given in the particle-wise form as

$$\left(\frac{\partial \rho_i}{\partial \mathbf{x}_k} \right)^\top = \begin{cases} \sum_j m_j \nabla W_{ij}, & k = i, \\ -m_j \nabla W_{ij}, & k = j, \\ \mathbf{0} & \text{otherwise.} \end{cases} \quad (9)$$

Now, the pressure force in Eq. (8) can be written compactly in the global-vector representation as

$$\mathbf{f}^p = -\mathbf{J}^\top \mathbf{V} \mathbf{\kappa}, \quad (10)$$

where $\mathbf{J} \in \mathbb{R}^{N \times 3N}$ stacks the row-wise density gradients (i.e., $\mathbf{J}_i = \frac{\partial \rho_i(\mathbf{x})}{\partial \mathbf{x}}$), $\mathbf{\kappa} \in \mathbb{R}^N$ stacks the particle-wise coefficients κ_i , and $\mathbf{V} \in \mathbb{R}^{N \times N}$ is a diagonal matrix with each entry being volume (i.e., $V_{ii} = \frac{m_i}{\rho_i}$).

4.2 Density Prediction

The semi-implicit Euler integration gives the updates of global state vectors for both position $\mathbf{x} \in \mathbb{R}^{3N}$ and velocity $\mathbf{v} \in \mathbb{R}^{3N}$ as

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \Delta t \mathbf{M}^{-1} (\mathbf{f}^{np} + \mathbf{f}^p(t)), \quad (11)$$

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t \mathbf{v}(t + \Delta t), \quad (12)$$

where $\Delta t \in \mathbb{R}$ is the timestep size, $\mathbf{M} \in \mathbb{R}^{3N \times 3N}$ is the block-diagonal mass matrix, $\mathbf{f}^{np} \in \mathbb{R}^{3N}$ aggregates non-pressure forces (such as

viscosity, surface tension, and external forces), and $\mathbf{f}^p(t)$ is the pressure force evaluated at time t .

Replacing $\mathbf{f}^p(t)$ in Eq. (11) with Eq. (10) yields

$$\mathbf{v}(t + \Delta t) = \mathbf{v}^* - \Delta t \mathbf{M}^{-1} \mathbf{J}^\top \mathbf{V} \mathbf{\kappa}, \quad (13)$$

where $\mathbf{v}^* = \mathbf{v}(t) + \Delta t \mathbf{M}^{-1} \mathbf{f}^{np}$ denoting the intermediate velocity. Here, note that $\mathbf{J}$ and $\mathbf{V}$ are evaluated at time t using particle densities and corresponding gradients.

With the position update in Eq. (12), on the other hand, the density of particle i at time $t + \Delta t$ can be predicted using a first-order expansion as

$$\begin{aligned} \rho_i(\mathbf{x}(t + \Delta t)) &= \rho_i(\mathbf{x}(t) + \Delta t \mathbf{v}(t + \Delta t)) \\ &\approx \rho_i(\mathbf{x}(t)) + \Delta t \mathbf{J}_i \mathbf{v}(t + \Delta t). \end{aligned} \quad (14)$$

Let us stack each particle density $\rho_i(\mathbf{x}) \in \mathbb{R}$ and denote the global density state vector as $\boldsymbol{\rho} \in \mathbb{R}^N$. Then, the density prediction is

$$\boldsymbol{\rho}(\mathbf{x}(t + \Delta t)) \approx \boldsymbol{\rho}(\mathbf{x}(t)) + \Delta t \mathbf{J} \mathbf{v}(t + \Delta t). \quad (15)$$

Substituting Eq. (13) into Eq. (15) finally yields

$$\boldsymbol{\rho}(\mathbf{x}(t + \Delta t)) \approx \boldsymbol{\rho}^* - \Delta t^2 \mathbf{J} \mathbf{M}^{-1} \mathbf{J}^\top \mathbf{V} \mathbf{\kappa}, \quad (16)$$

where $\boldsymbol{\rho}^* = \boldsymbol{\rho}(\mathbf{x}(t)) + \Delta t \mathbf{J} \mathbf{v}^*$ denoting the intermediate density.

4.3 Enforcing Constant Density

The incompressibility condition typically leads to solving for the pressure force that resolves density deviation. To this end, we can define the constant density condition, i.e., $\boldsymbol{\rho}(\mathbf{x}(t + \Delta t)) = \boldsymbol{\rho}_0$, where $\boldsymbol{\rho}_0 \in \mathbb{R}^N$ stacks the rest density, i.e., $\boldsymbol{\rho}_0 = \rho_0 \mathbf{1}$ with $\mathbf{1} \in \mathbb{R}^N$ being the all-ones vector. Enforcing the condition on the approximation of the density prediction in Eq. (16) yields the PPE:

$$\mathbf{J} \mathbf{M}^{-1} \mathbf{J}^\top \mathbf{V} \mathbf{\kappa} = \frac{\boldsymbol{\rho}^* - \boldsymbol{\rho}_0}{\Delta t^2}. \quad (17)$$

Since $\mathbf{V}$ has positive diagonal entries (i.e., $V_{ii} = \frac{m_i}{\rho_i} > 0$) and the pressure force depends on $\mathbf{V} \mathbf{\kappa}$ (as in Eq. (10)), we can introduce a scaled unknown $\boldsymbol{\lambda} = \mathbf{V} \mathbf{\kappa}$ and solve the equivalent system, instead:

$$(\mathbf{J} \mathbf{M}^{-1} \mathbf{J}^\top) \boldsymbol{\lambda} = \frac{\boldsymbol{\rho}^* - \boldsymbol{\rho}_0}{\Delta t^2}. \quad (18)$$

However, the system in Eq. (18) allows $\lambda_i < 0$ for particles that are deficient in neighbors. This causes the well-known *tensile instability* problem, which produces undesirable attractive pressure forces that pull particles toward each other, e.g., near the free surface. To remedy this, many SPH variants adopt PRJ, which iteratively updates $\boldsymbol{\lambda}$ via relaxed Jacobi iterations and clamps negative values to zero (i.e., enforcing $\boldsymbol{\lambda} \geq \mathbf{0}$) [Band et al. 2018; Ihmsen et al. 2013; Probst and Teschner 2023].

Constrained optimization. Solving Eq. (18) using PRJ and enforcing $\boldsymbol{\lambda} \geq \mathbf{0}$ can be described as solving the following constrained optimization problem for velocity:

$$\begin{aligned} \min_{\mathbf{v}(t + \Delta t)} \quad & \frac{1}{2} \|\mathbf{v}(t + \Delta t) - \mathbf{v}^*\|_{\mathbf{M}}^2 \\ \text{s.t.} \quad & \mathbf{c}(\mathbf{x}(t)) + \Delta t \mathbf{J} \mathbf{v}(t + \Delta t) \leq \mathbf{0}. \end{aligned} \quad (19)$$

Here, $\mathbf{c}(\mathbf{x}) = \boldsymbol{\rho}(\mathbf{x}) - \boldsymbol{\rho}_0$ is the constraint function and $\mathbf{J} = \nabla_{\mathbf{x}} \mathbf{c}(\mathbf{x}(t))$ is the Jacobian of the constraint function, which is equivalent to the density gradient in Eq. (10). Solving the optimization problem

of Eq. (19) means projecting the intermediate velocity $\mathbf{v}^*$ onto a velocity state feasible for the density constraint linearized in velocity, as $\mathbf{c}(\mathbf{x}(t + \Delta t)) \approx \mathbf{c}(\mathbf{x}(t)) + \Delta t \mathbf{J} \mathbf{v}(t + \Delta t)$, in the $\mathbf{M}$ -metric. For notational simplicity, we will henceforth let $\mathbf{c}$ and $\mathbf{v}$ denote $\mathbf{c}(\mathbf{x}(t))$ and $\mathbf{v}(t + \Delta t)$, respectively.

5 Explicit Velocity Projection

The constrained optimization problem of Eq. (19) can be solved using PRJ; this method is adopted for IISPH [Ihmsen et al. 2013]. Alternatively, we can decompose the constrained optimization problem of Eq. (19) into a sequence of subproblems as follows:

$$\arg \min_{\mathbf{v}^{l+1}} \frac{1}{2} \|\mathbf{v}^{l+1} - \mathbf{v}^l\|_{\mathbf{M}}^2 \quad \text{s.t.} \quad \mathbf{c} + \Delta t \mathbf{J} \mathbf{v}^{l+1} \leq \mathbf{0}, \quad (20)$$

where the superscript l denotes the subproblem index and $\mathbf{v}^0 = \mathbf{v}^*$. Then, in each subproblem, a single PRJ iteration is performed:

$$\mathbf{r}^l = \mathbf{c} + \Delta t \mathbf{J} \mathbf{v}^l, \quad (21)$$

$$\boldsymbol{\lambda}^l = \frac{\omega}{\Delta t^2} \mathbf{D}^{-1} \boldsymbol{\Phi}(\mathbf{r}^l), \quad (22)$$

$$\mathbf{v}^{l+1} = \mathbf{v}^l - \Delta t \mathbf{M}^{-1} \mathbf{J}^\top \boldsymbol{\lambda}^l. \quad (23)$$

Here, $\mathbf{r}^l$ is the global constraint residual vector, $\boldsymbol{\lambda}^l$ is the corresponding Lagrange multiplier, and $\mathbf{D} = \text{diag}(\mathbf{J} \mathbf{M}^{-1} \mathbf{J}^\top) + \delta \mathbf{I}$, where $\delta = 10^{-6}$ to prevent division by zero; note that $\mathbf{D} \in \mathbb{R}^{N \times N}$ is evaluated once at every frame. $\boldsymbol{\Phi}(\cdot) : \mathbb{R}^N \rightarrow \mathbb{R}^N$ denotes the clamping operator that can be defined component-wise as $\phi(r_i) = \max(r_i, 0)$, which is evaluated at every iteration, and ω is the relaxation parameter. It is worth noting that Eqs. (21)-(23) perform a single *projection* step in *velocity*, via an *explicit* fixed-point update (see Appendix ??). We therefore refer to this scheme as *explicit velocity projection*.

6 Implicit Velocity Projection

The relaxation parameter ω in the explicit velocity projection has been widely set to a conservative value of 0.5 since it presents a good trade-off between convergence speed and numerical stability [Band et al. 2018; Gissler et al. 2019; Ihmsen et al. 2013]. However, this limits the convergence speed. In this section, we propose our extension to an implicit formulation, i.e., *implicit velocity projection*, which permits a much larger value of $\omega \gg 1$, significantly improving stability and performance.

6.1 Linearization

Let us denote the Lagrange multiplier $\boldsymbol{\lambda}^l$ in Eqs. (21)-(22) as a function of the velocity state vector, i.e., $\boldsymbol{\lambda}(\mathbf{v}^l) : \mathbb{R}^{3N} \rightarrow \mathbb{R}^N$:

$$\boldsymbol{\lambda}(\mathbf{v}^l) = \frac{\omega}{\Delta t^2} \mathbf{D}^{-1} \boldsymbol{\Phi}(\mathbf{c} + \Delta t \mathbf{J} \mathbf{v}^l). \quad (24)$$

The explicit velocity update formula of Eq. (23) is then rearranged as

$$\mathbf{M}(\mathbf{v}^{l+1} - \mathbf{v}^l) + \Delta t \mathbf{J}^\top \boldsymbol{\lambda}(\mathbf{v}^l) = \mathbf{0}. \quad (25)$$

Naturally, its implicit formulation is derived by replacing $\mathbf{v}^l$ in $\boldsymbol{\lambda}(\mathbf{v}^l)$ of Eq. (25) with the unknown *next* velocity state, $\mathbf{v}^{l+1}$:

$$\mathbf{M}(\mathbf{v}^{l+1} - \mathbf{v}^l) + \Delta t \mathbf{J}^\top \boldsymbol{\lambda}(\mathbf{v}^{l+1}) = \mathbf{0}. \quad (26)$$

Algorithm 1 Implicit Velocity Projection

- 1: Find neighborhoods.
- 2: Compute non-pressure forces $\mathbf{f}^{np}$.
- 3: Compute $\mathbf{v}^* \leftarrow \mathbf{v}(t) + \Delta t \mathbf{M}^{-1} \mathbf{f}^{np}$.
- 4: Precompute $\mathbf{c}$, $\mathbf{J}$, and $\mathbf{D}^{-1}$.
- 5: Set $\mathbf{v}^0 \leftarrow \mathbf{v}^*$ and $l \leftarrow 0$.
- 6: Repeat the following while $c^{avg} \geq \eta \rho_0$ (Eq. (30)):
- 7: Compute $\Delta \mathbf{v}^l$ using the matrix-free CG solver (Eq. (29)).
- 8: Set $\mathbf{v}^{l+1} \leftarrow \mathbf{v}^l + \Delta \mathbf{v}^l$.
- 9: Set $l \leftarrow l + 1$.
- 10: End repetition.
- 11: Update $\mathbf{v}(t + \Delta t)$ and $\mathbf{x}(t + \Delta t)$.

Introducing the velocity increment $\Delta \mathbf{v}^l = \mathbf{v}^{l+1} - \mathbf{v}^l$, we linearize Eq. (26) using the first-order Taylor expansion as

$$\mathbf{M} \Delta \mathbf{v}^l + \Delta t \mathbf{J}^\top \left(\boldsymbol{\lambda}(\mathbf{v}^l) + \frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{v}} \bigg|_{\mathbf{v}^l} \Delta \mathbf{v}^l \right) = \mathbf{0}. \quad (27)$$

Applying the chain rule, the term $\frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{v}} \big|_{\mathbf{v}^l}$ is derived as

$$\begin{aligned} \frac{\partial \boldsymbol{\lambda}}{\partial \mathbf{v}} \bigg|_{\mathbf{v}^l} &= \frac{\omega}{\Delta t^2} \mathbf{D}^{-1} \frac{\partial \boldsymbol{\Phi}}{\partial \mathbf{r}} \bigg|_{\mathbf{r}^l} \frac{\partial \mathbf{r}}{\partial \mathbf{v}} \bigg|_{\mathbf{v}^l} \\ &= \frac{\omega}{\Delta t^2} \mathbf{D}^{-1} \frac{\partial \boldsymbol{\Phi}}{\partial \mathbf{r}} \bigg|_{\mathbf{r}^l} (\Delta t \mathbf{J}). \end{aligned} \quad (28)$$

Substituting Eq. (28) into Eq. (27) yields a linear system for $\Delta \mathbf{v}^l$:

$$\left(\mathbf{M} + \omega \mathbf{J}^\top \mathbf{D}^{-1} \frac{\partial \boldsymbol{\Phi}}{\partial \mathbf{r}} \bigg|_{\mathbf{r}^l} \mathbf{J} \right) \Delta \mathbf{v}^l = -\Delta t \mathbf{J}^\top \boldsymbol{\lambda}(\mathbf{v}^l). \quad (29)$$

Here, $\frac{\partial \boldsymbol{\Phi}}{\partial \mathbf{r}} \big|_{\mathbf{r}^l}$ is a diagonal matrix whose entries are defined using the Heaviside step function with a small numerical threshold. Specifically, the i -th entry is 1 for active constraints where $r_i > 10^{-12}$ and 0 otherwise. Although the Heaviside step function is non-differentiable at $r_i = 0$, this poses no issue, as $\frac{\partial \boldsymbol{\Phi}}{\partial \mathbf{r}} \big|_{\mathbf{r}^l}$ remains fixed while solving for $\Delta \mathbf{v}^l$.

Projection stiffness. Our implicit formulation embeds the relaxation parameter ω in the system matrix of Eq. (29), allowing $\omega \gg 1$. Here, ω weights the constraint term relative to the mass matrix, thereby acting as projection stiffness. A larger ω enables a stiffer projection of each velocity update with the solution $\Delta \mathbf{v}^l$, potentially accelerating convergence.

Matrix-free conjugate gradient (CG) solver. Since $\mathbf{M} \succ 0$ and $\mathbf{D}^{-1} \frac{\partial \boldsymbol{\Phi}}{\partial \mathbf{r}} \big|_{\mathbf{r}^l}$ is diagonal with non-negative entries, the system matrix of Eq. (29) is symmetric positive definite, which allows an efficient iterative solver such as CG. We adopt a matrix-free implementation for the CG solver (see Appendix ??). For each frame, $\mathbf{J}$, $\mathbf{D}^{-1}$, and the term $\mathbf{c}$ used in $\boldsymbol{\lambda}(\mathbf{v}^l)$ can be precomputed once and reused across the subproblems of our implicit velocity projection.

6.2 Termination Condition

Algorithm 1 outlines the complete procedure of our solver. Our solver iteratively performs the velocity update using the velocity increment as $\mathbf{v}^{l+1} = \mathbf{v}^l + \Delta \mathbf{v}^l$, which we call *outer* iterations. The

outer iterations update the velocity state until the average density constraint error c^{avg} falls below a threshold:

$$c^{avg} = \frac{1}{N} \sum_{i=1}^N \max(c_i + \Delta t \mathbf{J}_i \mathbf{v}^l, 0) < \eta \rho_0, \quad (30)$$

where η is a user-specified tolerance that defines the acceptable average error relative to ρ_0 .

Unlike the explicit velocity projection, our solver includes iterations of the CG solver for the linear system of each subproblem, which we call *inner* iterations. The linear system solver also requires a termination condition. Denoting the linear system of Eq. (29) at the l -th subproblem as $\mathbf{A}^l \Delta \mathbf{v}^l = \mathbf{b}^l$, the termination condition is defined as $\|\mathbf{b}^l - \mathbf{A}^l \Delta \mathbf{v}^l\|^2 \leq \eta_{cg}$, where η_{cg} is the threshold of the CG solver. Note that the accuracy of the linear system solver naturally affects the performance of our solver. Although a tighter termination condition (i.e., smaller η_{cg}) for the CG solver increases the number of inner iterations, it significantly reduces the outer iterations, thus leading to better performance overall.

7 Experimental Results

This section reports our experimental results. All experiments are performed on an Intel i7 Gen 13 processor and an NVIDIA RTX 4090 GPU, with our method and the SPH variants implemented in NVIDIA Warp [Macklin 2022]. Henceforth, we denote our implicit velocity projection as IP and the explicit velocity projection as EP. Unless otherwise noted, we adopt the following implementation setups. Table. 1 summarizes the statistics of the experiments.

Implementation Details. The solvers adopt adaptive time-stepping using the Courant-Friedrichs-Lewy (CFL) condition [Monaghan 1992] for which the CFL number is set to 0.4. The termination criterion η in Eq. (30) is set to 10^{-3} , corresponding to an average density error below 0.1%, which is a widely accepted tolerance in many incompressible SPH implementations [Bender et al. 2026]. We use the cubic spline kernel for SPH evaluations and the derivative of the spiky kernel for the gradients; the support radius h is set to twice the particle diameter. To accelerate neighborhood queries, we adopt a uniform grid for the spatial hashing. We apply the artificial viscosity of Schechter and Bridson [2012] and the surface tension model of Akinci et al. [2013b]; additionally, more sophisticated models [Jeske et al. 2023; Weiler et al. 2018] are employed in selected experiments and noted. Boundary conditions are enforced using the approach of Bender et al. [2023].

7.1 Projection Stiffness ω

We first investigate the effect of the projection stiffness ω on our IP solver using a *Fluid Pillar* example, in which a fluid block of 100k fluid particles is falling into a narrow rectangular container. This scenario stacks the fluid particles vertically upon impact, producing sustained compression under gravity and thus placing continuous numerical stress on the pressure solver. Fig. 2-(a) shows representative snapshots of the example. We note that the converged results with varying ω values are visually indistinguishable.

Since ω affects the condition number of the linear system for each subproblem in Eq. (29), it influences the performance of the CG solver. For varying ω values ranging from under-relaxation (e.g., 0.5)

Table 1. **Statistics.** “iter.” and “time” denote the density solver’s iteration count and elapsed time [ms], respectively, averaged per unit timestep. For IP, “iter.” shows outer (per unit timestep) and inner (per outer) iterations. “-” indicates diverged simulation. **Bold** indicates the best performance in each row.

(a) **Scene configurations.** “#f” and “#b” denote fluid and boundary particle counts, respectively; r is particle radius [mm]; Δt is timestep size [ms], which is used as the unit timestep size for performance measurement; and η is density error threshold.

Scene		#f	#b	r	Δt	η
Hollow Tree	(Fig. 1)	978k	1.75M	3	2	10^{-4}
Fluid Pillar	(Fig. 2)	100k	742k	3	2	10^{-3}
Dam Break	(Fig. 5)	97k	248k	2	8	10^{-3}
Stack	(Fig. 6)	315k	179k	3	4	10^{-5}
Small Fluid Pillar	(Fig. 12)	40k	37k	6	8	10^{-3}

(b) **Dam Break:** Varying timestep size Δt .

Δt	IP ($\omega = 10$)			IISPH		DFSPH	
	iter. outer	iter. inner	time	iter.	time	iter.	time
2	2.01	5.52	10.03	9.06	6.62	6.67	4.90
4	2.31	8.11	15.19	29.91	22.27	20.94	15.22
6	3.92	7.88	25.26	-	-	-	-
8	5.46	7.66	34.93	-	-	-	-

(c) **Stack:** Varying density error tolerance η .

η	IP ($\omega = 10^2$)			IISPH		DFSPH	
	iter. outer	iter. inner	time	iter.	time	iter.	time
10^{-3}	4.48	21.55	157.24	37.42	79.65	27.87	59.53
10^{-4}	6.80	22.40	283.39	-	-	133.82	282.11
10^{-5}	63.00	5.87	924.68	-	-	-	-

(d) **Fluid Pillar:** Scalability.

#f	IP ($\omega = 10$)			IISPH		DFSPH	
	iter. outer	iter. inner	time	iter.	time	iter.	time
50K	2.0	5.84	8.0	15.5	6.8	10.2	4.5
100K	2.1	6.82	15.1	28.5	23.5	18.6	15.1
200K	3.3	6.93	39.6	54.9	82.6	38.2	57.2
300K	4.7	6.61	78.7	79.2	182.8	60.9	140.4
400K	6.0	6.46	131.1	101.8	292.0	84.0	240.4

(e) **Fluid Pillar (Small):** Comparison between IP and PBF.

Δt	Time [s]	IP ($\omega = 10$)			PBF	
		iter. outer	iter. inner	time	iter.	time
8	0–10	5.32	9.33	17.38	104.12	34.60
16	10–15	-	-	-	404.35	137.21
24	15–20	-	-	-	760.28	255.49

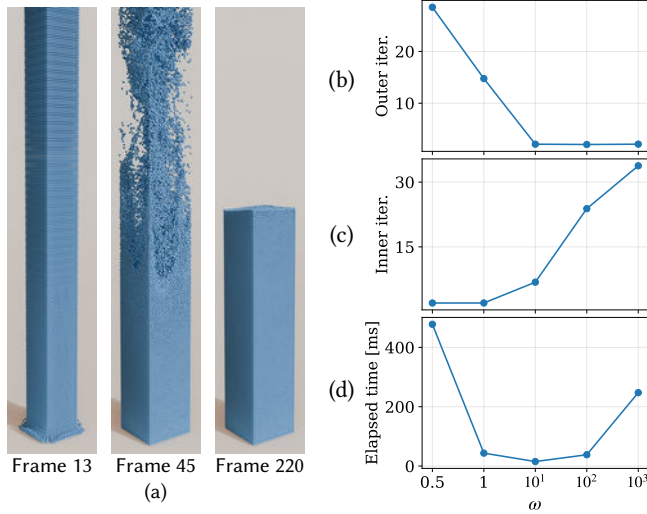


Fig. 2. **Fluid Pillar.** (a) Snapshots of the IP solver results ($\omega = 10$). Average (b) outer iterations per unit timestep, (c) inner iterations per outer iteration, and (d) elapsed time per unit timestep for varying ω .

to significant over-relaxation (e.g., 10^3), the graphs in Fig. 2 show the average numbers of both outer and inner iterations and the average total elapsed time for our solver. As Fig. 2-(b) indicates, the larger relaxation values, specifically $\omega \geq 10$, lead to faster convergence in general, requiring only two outer iterations on average. Conversely, the systems with larger ω require more inner iterations as shown in Fig. 2-(c). With the termination threshold for the CG solver η_{cg} set to 10^{-6} , Fig. 2-(d) shows the average total elapsed time for varying ω , indicating $\omega = 10$ is optimal for this example. Analyzing the average total elapsed time further in our experiments, we find $\omega \in \{10, 10^2\}$ empirically optimal and adopt them for all subsequent experiments.

Furthermore, we examine the effect of ω on other solvers, comparing IP, EP, and IISPH. Fig. 3-(a) shows close-up snapshots, captured once the dynamics of the pillar example have stabilized. The density error is visualized using a color mapping; red indicates higher error. The typical relaxation of $\omega = 0.5$ performs well for both EP and IISPH. However, when a larger relaxation of $\omega = 1.0$ is applied, EP exhibits a noisy error distribution and suffers from severe temporal oscillations. We note that IISPH explodes when $\omega = 1.0$, whereas IP remains robust even with a much higher relaxation of $\omega = 10$.

Fig. 3-(b) and -(c) show the average outer iterations and elapsed times, respectively, per unit timestep over simulation time for the different solvers. Compared to the other solvers, our IP solver with $\omega = 10$ significantly reduces the outer iteration count and maintains a consistent iteration count without drastic fluctuations. Our IP solver presents an average elapsed time of 15.1 ms per unit timestep; the inner CG solver dominates the cost. Although this is higher than the 13.3 ms recorded for EP with $\omega = 1.0$, EP suffers from the instability drawbacks mentioned above. We note that IISPH (26.5 ms) and EP with $\omega = 0.5$ (26.9 ms) exhibit nearly identical performance; this can be attributed to the procedural similarity in their numerical algorithms as discussed in Sec. 5. We will detail

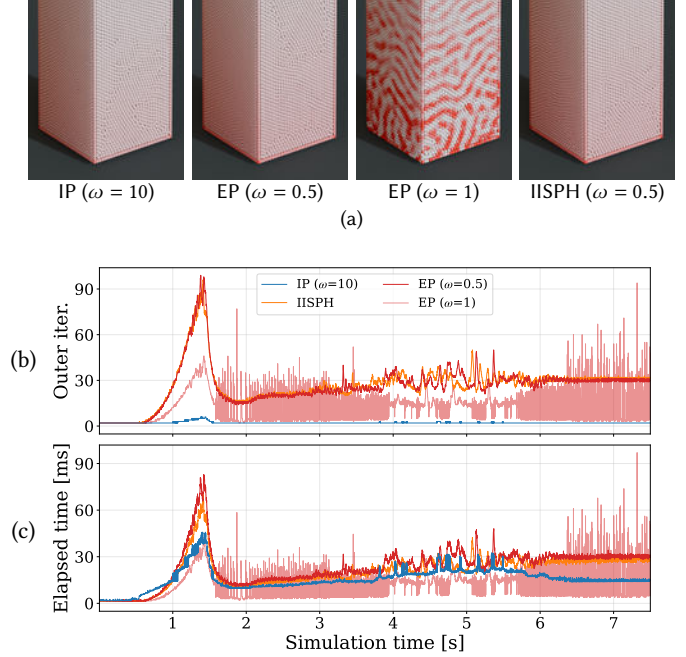


Fig. 3. **Fluid Pillar.** (a) Snapshots of the pillar base, where red indicates higher density error. (b) Outer iterations and (c) elapsed time per unit timestep over simulation time.

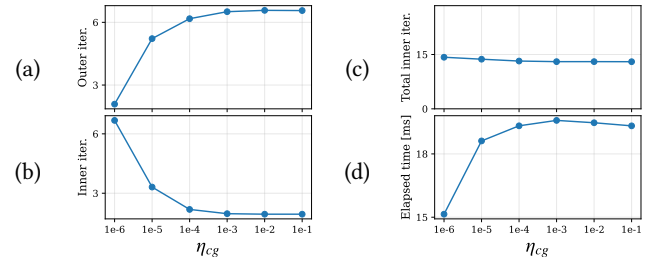


Fig. 4. **Fluid Pillar.** Effect of η_{cg} on performance. Tighter η_{cg} reduces (a) the outer iterations per unit timestep and increases (b) the inner CG iterations per outer iteration. With fewer outer iterations, (c) the total inner iterations remain nearly constant, (d) yielding better overall performance.

our further evaluation on the performance of different solvers in Sec. 7.3.

7.2 Inner CG threshold η_{cg}

Our IP solver requires an iterative linear system solver for each subproblem, for which we employ a matrix-free CG implementation. As discussed in Sec. 6.2, η_{cg} determines the accuracy of the solution for each subproblem, thus influencing the performance of our solver. Intuitively, the smaller η_{cg} value results in more accurate solutions for each subproblem at the cost of more inner iterations, while the more accurate solutions generally lead to faster convergence on the density constraint error.

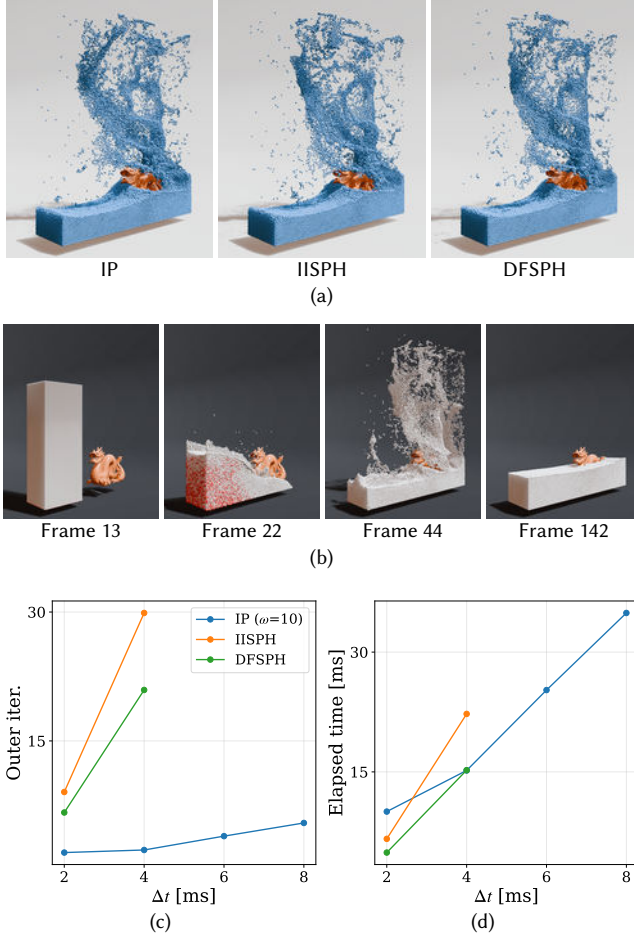


Fig. 5. **Dam Break.** (a) Snapshots of the IP, IISPH, and DFSPH solvers' results ($\Delta t = 4$ ms) at Frame 44. (b) Snapshots of the IP's result ($\Delta t = 8$ ms) (red: higher density error). (c) Average outer iteration and (d) average elapsed time, for varying Δt .

When η_{cg} varies in the *Fluid Pillar* example, Fig. 4 shows its effect on the performance of our solver in four aspects: (a) outer iteration counts, (b) inner iteration counts per outer iteration, (c) total inner iteration counts, and (d) elapsed times. The smaller threshold generally leads to faster convergence as shown in Fig. 4-(a), while increasing the inner iterations per outer iteration as shown in Fig. 4-(b). Nonetheless, it does not significantly increase the total inner iterations thanks to faster convergence as shown in Fig. 4-(c), resulting in better performance as shown in Fig. 4-(d). Consequently, all subsequent experiments adopt $\eta_{cg} = 10^{-6}$.

7.3 Stability and Performance

We examine the robustness of our solver first across varying timestep sizes, Δt , with a *Dam Break* example involving 97K particles. In this example, a static dragon-shaped obstacle is placed on the floor, and a fluid column at a corner of the domain collapses under gravity. To evaluate the stability limit beyond the CFL-constrained

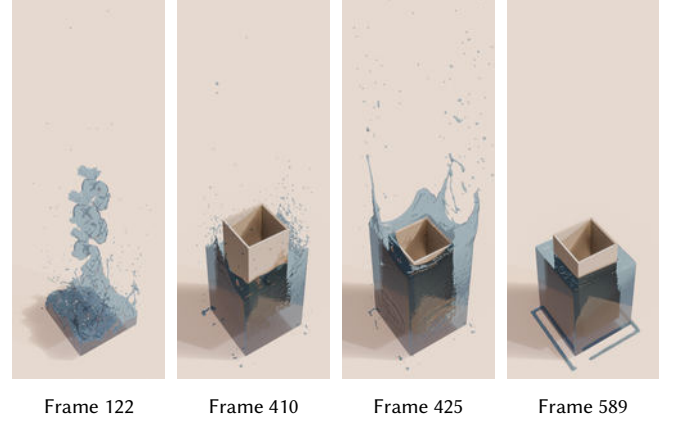


Fig. 6. **Stack.** Snapshots of the IP solver results when $\eta = 10^{-5}$.

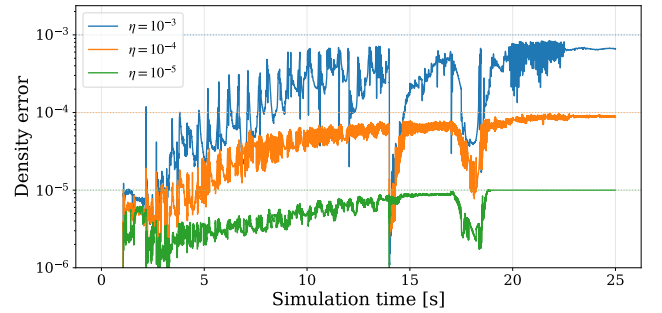


Fig. 7. **Stack.** Average density error of the IP solver results over simulation time on varying η .

adaptive time-stepping, we employ a fixed timestep integration. Fig. 5-(a) first shows that IP, IISPH, and DFSPH with $\Delta t = 4$ ms produce visually similar results. The one-way interaction with complex solid boundaries produces a significant amount of splashes, and then they eventually stabilize as shown in Fig. 5-(b).

Fig. 5-(c) and -(d) show the average outer iteration counts and total elapsed times, respectively, for the different solvers. Our IP solver maintains stability even with a significantly large timestep of 8 ms at the cost of additional iterations for convergence. In contrast, we observe that both IISPH and DFSPH fail to converge within 10^4 iterations when Δt exceeds 4 ms, specifically at the first impact of the fluid volume on the floor; consequently, their data points are absent from the graphs.

We now vary the density error tolerance using η and examine the solvers further with a *Stack* example. In this example, a pair of dragon-shaped fluid volumes is emitted 25 times in a rectangular container at every 0.5 seconds, resulting in 315K fluid particles in total. This sequence of emissions places sustained stress on the pressure solver as successive volumes of fluid accumulate. After the fluid is stabilized, a narrower rigid box rapidly descends into the domain, introducing additional severe stress on the incompressibility solver. For this experiment, we adopt an implicit viscosity model [Weiler et al. 2018] and an implicit surface tension model [Jeske et al. 2023],

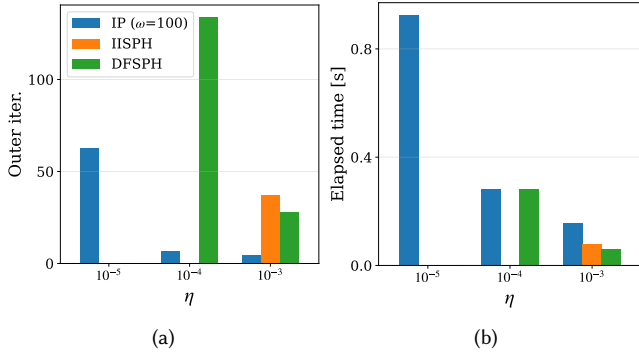


Fig. 8. **Stack.** (a) Average outer iterations and (b) average elapsed times, per unit timestep on varying η .

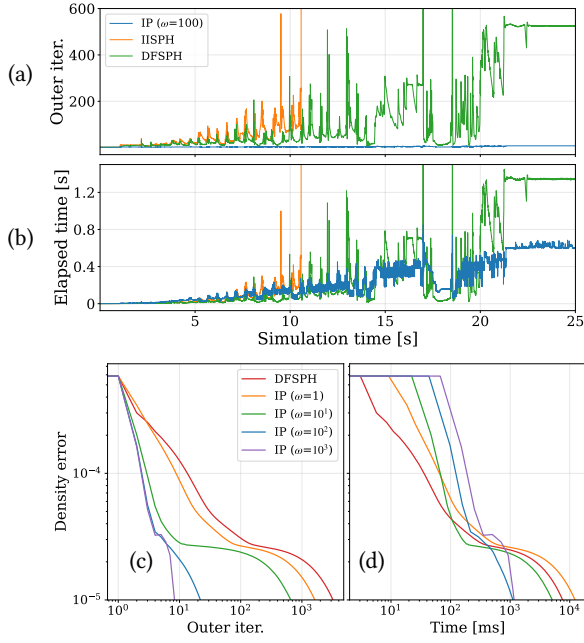


Fig. 9. **Stack.** (a) Outer iterations and (b) elapsed time per unit timestep over simulation time when $\eta = 10^{-4}$. Average density-error convergence over (c) outer iterations and (d) cumulative solve time at the last frame.

setting each of those coefficients to 0.05 and 500, respectively. Fig. 6 shows representative snapshots. Our solver effectively maintains the average density error below the given tolerance as shown in Fig. 7. We note that the tolerance values are tight enough, i.e., $\eta \leq 10^{-3}$, and thus the results of them are visually similar.

Intuitively, a smaller η requires more iterations to converge. Accordingly, all the solvers take more iterations and longer computation times as η decreases, as shown in Fig. 8-(a) and -(b). The graphs notably show that our IP solver robustly converges even when the tolerance is significantly tight, e.g., $c^{avg} \leq 10^{-5}\rho_0$, whereas both IISPH and DFSPH fail to converge within 10^4 iterations for the same

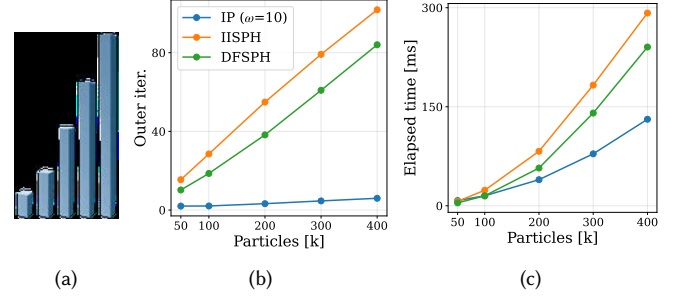


Fig. 10. **Scalability.** (a) Snapshots of *Fluid Pillar* example at different scales. (b) Average outer iterations and (c) average elapsed time, per unit timestep size for different particle counts.

tolerance. When the tolerance is loosened to $c^{avg} \leq 10^{-4}\rho_0$, on the other hand, DFSPH converges too, with comparable performance.

Fig. 9-(a) and -(b) show the average outer iteration counts and total elapsed times, respectively, per unit timestep over simulation time when $c^{avg} \leq 10^{-4}\rho_0$. As shown in Fig. 9-(a), our IP solver presents significantly consistent outer iteration counts, whereas DFSPH shows oscillation with spikes, which correspond to the frames where strong compressions occur. As shown in Fig. 9-(b), we observe that our IP solver outperforms DFSPH in computation time, particularly after approximately 21 seconds when the fluid particles are stacked in a column and exerted by a continuous gravitational force. We note that IISPH fails to converge within 10^4 iterations after approximately 10 seconds when a pair of emitted fluid volumes reaches the existing volume on the floor, leading to a challenging compression.

We further investigate the solvers' behaviors for the last frame (i.e., at $t = 25$ s). Fig. 9-(c) and -(d) show changes in density error over outer iterations and cumulative solve time, respectively. As shown in Fig. 9-(c), our IP solver converges fast (i.e., requires fewer outer iterations) until the density error reaches below $10^{-5}\rho_0$, presenting its advantage over DFSPH. While the stronger relaxation on IP requires fewer outer iterations in general, the IP solver presents the best performance when $\omega = 100$ instead of $\omega = 1000$ as shown in Fig. 9-(d); note that the relaxation affects the inner CG solver.

Scalability. Given the improved stability and performance of our solver, we examine its scalability on the *Fluid Pillar* example, varying the number of fluid particles from 50k to 400k. Fig. 10-(a) illustrates the relative scales of the different particle counts, and Fig. 10-(b) and -(c) show the average outer iteration counts and average elapsed times, respectively, per unit timestep of the different solvers. As indicated in the graphs, the computational advantage of our solver becomes more prominent as the simulation scale increases. For example, when the number of particles increases eight-fold, the elapsed time of our IP solver increases by a factor of 16, whereas that of DFSPH (the second best) increases by a factor of 53. We also underline that, unlike the other methods, our method presents a marginal increase in outer iterations, indicating a significant potential for further performance improvement with a more efficient inner solver for linear systems.

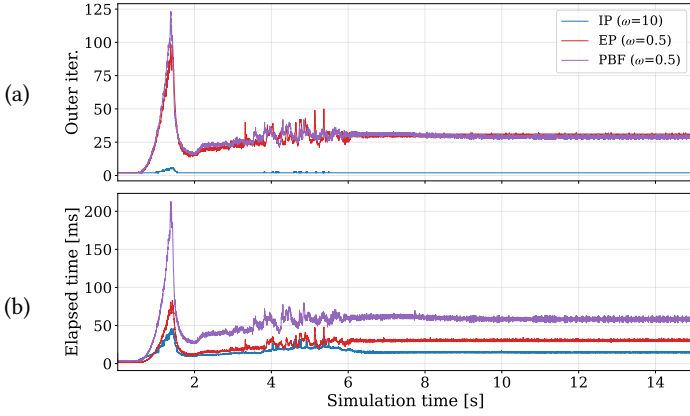


Fig. 11. **Comparison of IP, EP, and PBF at a small timestep size.** *Fluid Pillar* with 100k particles at $\Delta t = 2$ ms. (a) Outer iterations and (b) elapsed time per unit timestep over simulation time.

Showcase. Fig. 1 showcases our method with a *Hollow Tree* example, where fluid particles are continuously emitted up to 978k inside the tree-shaped boundary, represented using 1.75M boundary particles. The gradually increasing hydrostatic load makes incompressibility enforcement progressively more demanding, and the complex boundary geometry makes its resolution significantly challenging. Here, we adopt the viscosity [Weiler et al. 2018] and surface tension [Jeske et al. 2023], setting their coefficients to 0.1 and 100, respectively. On average, our IP solver requires 4.39 outer iterations per unit timestep and 35 inner iterations per outer iteration, resulting in 811 ms per unit timestep. Although not presented, we note that DFSPH requires 277 iterations per unit timestep, resulting in 1515 ms per unit timestep. This result highlights the superior efficiency of our method in handling challenging scenarios.

7.4 Comparison to PBF

In contrast to velocity-based methods, PBF formulates the constant density problem based on particle positions. The PBF solver performs iterative projections directly on positions rather than velocities. Each iteration updates the constraint values and their gradients; thus, PBF is well-known to be robust in terms of stability with large timestep sizes while requiring significant per-iteration overhead [Koschier et al. 2022].

We first examine the small timestep size regime in the *Fluid Pillar* example introduced in Sec. 7.1, comparing PBF with EP and IP. As shown in Fig. 11-(a), PBF and EP require comparable outer iteration counts, reporting 28.6 and 28.2 on average, respectively. However, the elapsed time for PBF, as shown in Fig. 11-(b), is 54.8 ms per frame, about twice the 26.8 ms of EP, and about 3.6 times the 15.1 ms of our IP solver. While stability is not an issue for all solvers at this timestep size, the per-iteration overhead of PBF accounts for this performance gap.

We next examine the large-timestep regime, where the robust stability of PBF becomes particularly advantageous. To evaluate our IP solver in comparison to PBF, we configure *Small Fluid Pillar* example and employ fixed time stepping with a prescribed schedule:

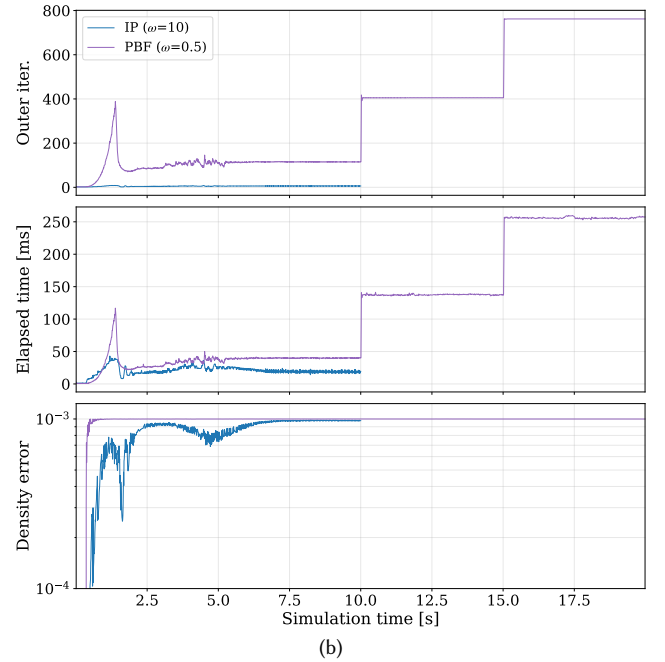
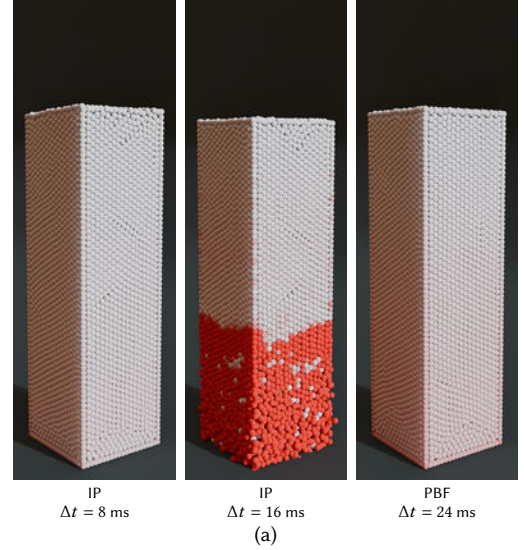


Fig. 12. **Comparison between IP and PBF at larger time step sizes.** (a) Snapshots of the two solvers' results for different Δt values. (b) Average density error, outer iterations, and elapsed time, from top to bottom, over simulation time as Δt increases during the simulation.

$\Delta t = 8$ ms for $t \in [0, 10]$ s, $\Delta t = 16$ ms for $t \in [10, 15]$ s, and $\Delta t = 24$ ms for $t \in [15, 20]$ s. The density error tolerance is set to $c^{avg} < 10^{-3} \rho_0$ for both solvers, and Fig. 12-(b)-top confirms that this tolerance is met throughout the simulation. We use an under-relaxation value of $\omega = 0.5$ for PBF as in the typical PBF implementation [Macklin et al. 2014]. As Fig. 12-(a) shows, PBF exhibits remarkable stability even in the regime of $\Delta t = 24$ ms, while

our IP solver becomes unstable when Δt is increased to 16 ms. On the other hand, Fig. 12-(b)-middle and -(b)-bottom report the outer iteration count and elapsed time, respectively, per unit timestep over simulation time when $\Delta t = 8$ ms. In this timestep size regime, our IP solver demonstrates clear computational benefit over PBF; on average, IP requires 17.38 ms per unit timestep, whereas PBF requires 34.60 ms. We note that, when $\Delta t = 24$ ms, PBF requires over 760 outer iterations per unit timestep. Accordingly, the two methods occupy distinct regimes with a trade-off between stability and efficiency: IP is faster within the stable timestep size range, while PBF retains robustness at much larger timestep sizes at the cost of additional overhead.

8 Discussion and Limitation

Optimal projection stiffness value. Our IP solver remains stable across a wide range of ω values, even with significantly large values. As discussed in Sec. 7.1, we find $\omega \in [10, 10^2]$ effective across the scenes examined. This recommendation, however, may not be optimal. As ω affects the linear system at each given frame, the optimal ω value is inherently dependent on the state of each frame and thus may vary across frames or even across outer iterations within a single frame. Finding the optimal value of ω for a given problem is underexplored in the current study. Nonetheless, our IP solver consistently demonstrated superior performance with the identified empirical range of ω as discussed in our experiments.

Preconditioning for CG. Our IP solver involves the inner CG solver. Thus, it may result in overhead, particularly when the given scenario can be resolved more efficiently with other solvers. A common practice to improve efficiency is preconditioning, which enhances the system's condition and thus accelerates convergence. However, we observed that a simple Jacobi preconditioning yielded no significant performance gains in our experiments. We tentatively attribute this to the structure of the system matrix in Eq. (29): D^{-1} normalizes the density gradient J , and the Heaviside entries in $\partial\phi/\partial r$ filter out inactive components. This suggests that the system is reasonably well-conditioned by construction. This view is consistent with Table 1 as the inner CG iteration counts per outer step stay small across all scenes. Nonetheless, a more comprehensive investigation into advanced preconditioning methods remains an interesting avenue for future work.

9 Conclusion and Future Work

In this work, we have demonstrated how to enhance the stability and performance of incompressible SPH fluid simulations through an implicit velocity projection method. We reinterpret the projection mechanism in conventional PRJ variants, which solve the constant density (i.e., incompressibility) problem, as an explicit fixed-point update for a constrained optimization problem and extend it to an implicit counterpart. This extension allows the solver to use substantially large relaxation values, which in turn yield fast convergence and improve computational efficiency, while maintaining stability. Our comprehensive experiments, investigating solver's robustness across varying timestep sizes, density-error tolerances, and particle counts, demonstrate the strength of our method over representative SPH fluid solvers.

The two limitations discussed in Sec. 8 naturally suggest future work. First, the relaxation parameter ω is fixed during the simulation in the current method, whereas the optimal value may vary with simulation state. Investigating adaptive mechanisms for determining the optimal ω is, therefore, a promising direction for future research. Second, since the inner CG solver is the main bottleneck of our method, adopting advanced linear system solvers and effective preconditioning techniques, such as multigrid methods, could further enhance performance. Furthermore, we plan to extend our implicit projection method to various simulation domains, such as collision handling for deformable solids.

Acknowledgments

This work was supported by grants from the Institute of Information & Communications Technology Planning & Evaluation (IITP), funded by the Korea government (Ministry of Science and ICT, MSIT), under the SW StarLab Program (No. RS-2020-II200861), the ICT Creative Consilience Program (No. IITP-2026-RS-2020-II201819), and the Information Technology Research Center (ITRC) Program (No. IITP-2026-RS-2020-II201460).

References

- Nadir Akinci, Gizem Akinci, and Matthias Teschner. 2013a. Versatile surface tension and adhesion for SPH fluids. *ACM Transactions on Graphics (TOG)* 32, 6 (2013), 1–8.
- Nadir Akinci, Jens Cornelis, Gizem Akinci, and Matthias Teschner. 2013b. Coupling elastic solids with smoothed particle hydrodynamics fluids. *Computer Animation and Virtual Worlds* 24, 3–4 (2013), 195–203.
- Stefan Band, Christoph Gissler, Markus Ihmsen, Jens Cornelis, A. Peer, and M. Teschner. 2018. Pressure Boundaries for Implicit Incompressible SPH. In *ACM Transactions on Graphics*. doi:10.1145/3180486
- Markus Becker and Matthias Teschner. 2007. Weakly compressible SPH for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 209–217.
- Jan Bender et al. 2026. *SPLisHSPlasH Library*. <https://github.com/InteractiveComputerGraphics/SPLisHSPlasH>
- Jan Bender and Dan Koschier. 2015. Divergence-free smoothed particle hydrodynamics. In *Proceedings of the 14th ACM SIGGRAPH/Eurographics symposium on computer animation*. 147–155.
- Jan Bender, Lukas Westhofen, and Stefan Rhys Jeske. 2023. Consistent SPH rigid-fluid coupling. In *VMV*. 209–217.
- Mathieu Desbrun and Marie-Paule Gascuel. 1996. 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*. Springer, 61–76.
- Elie Diaz, Jerry Hsu, Eisen Montalvo-Ruiz, Chris Giles, and Cem Yuksel. 2025. Implicit Position-Based Fluids. In *Proceedings of the SIGGRAPH Asia 2025 Conference Papers*. 1–9.
- Christoph Gissler, Andreas Peer, Stefan Band, Jan Bender, and Matthias Teschner. 2019. Interlinked SPH pressure solvers for strong fluid-rigid coupling. *ACM Transactions on Graphics (TOG)* 38, 1 (2019), 1–13.
- Xiaowei He, Shusen Liu, Yuzhong Guo, Jian Shi, and Ying Qiao. 2025. A Semi-Implicit SPH Method for Compressible and Incompressible Flows with Improved Convergence. *Computer Graphics Forum* (2025). doi:10.1111/cgf.70043
- Markus Ihmsen, Jens Cornelis, Barbara Solenthaler, Christopher Horvath, and Matthias Teschner. 2013. Implicit incompressible SPH. *IEEE transactions on visualization and computer graphics* 20, 3 (2013), 426–435.
- Stefan Rhys Jeske, Lukas Westhofen, Fabian Löschner, José Antonio Fernández-Fernández, and Jan Bender. 2023. Implicit surface tension for SPH fluid simulation. *ACM Transactions on Graphics* 43, 1 (2023), 1–14.
- Dan Koschier, Jan Bender, Barbara Solenthaler, and Matthias Teschner. 2022. A survey on SPH methods in computer graphics. In *Computer graphics forum*, Vol. 41. Wiley Online Library, 737–760.
- Miles Macklin. 2022. Warp: A High-performance Python Framework for GPU Simulation and Graphics. <https://github.com/NVIDIA/warp>. NVIDIA GPU Technology Conference (GTC).
- Miles Macklin and Matthias Müller. 2013. Position based fluids. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.

- Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. 2014. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–12.
- Joe J Monaghan. 1992. Smoothed particle hydrodynamics. In: *Annual review of astronomy and astrophysics*. Vol. 30 (A93-25826 09-90), p. 543–574. 30 (1992), 543–574.
- Matthias Müller, David Charypar, and Markus Gross. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 154–159.
- Timo Probst and Matthias Teschner. 2023. Monolithic friction and contact handling for rigid bodies and fluids using SPH. In *Computer Graphics Forum*, Vol. 42. Wiley Online Library, 155–179.
- Timo Probst and Matthias Teschner. 2024. Unified Pressure, Surface Tension and Friction for SPH Fluids. *ACM Transactions on Graphics* 44, 1 (2024), 1–28.
- Hagit Schechter and Robert Bridson. 2012. Ghost SPH for animating water. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–8.
- Morten Silcowitz-Hansen, Sarah Niebe, and Kenny Erleben. 2010. A nonsmooth non-linear conjugate gradient method for interactive contact force problems. *The Visual Computer* 26, 6 (2010), 893–901.
- B. Solenthaler and R. Pajarola. 2009. Predictive-corrective incompressible SPH. In *ACM Transactions on Graphics*. doi:10.1145/1576246.1531346
- Tetsuya Takahashi, Yoshinori Dobashi, Issei Fujishiro, Tomoyuki Nishita, and Ming C Lin. 2015. Implicit formulation for SPH-based viscous fluids. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 493–502.
- M. Weiler, Dan Koschier, and Jan Bender. 2016. Projective fluids. In *Motion in Games*. doi:10.1145/2994258.2994282
- Marcel Weiler, Dan Koschier, Magnus Brand, and Jan Bender. 2018. A physically consistent implicit viscosity solver for SPH fluids. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 145–155.