EUROGRAPHICS 2015 / O. Sorkine-Hornung and M. Wimmer
(Guest Editors)

Volume 34 (2015), Number 2

Implicit Formulation for SPH-based Viscous Fluids

Tetsuya Takahashi[1,2] Yoshinori Dobashi[3,2] Issei Fujishiro[4] Tomoyuki Nishita[2,5] Ming C. Lin[1]

[1]The University of North Carolina at Chapel Hill, USA
[2]UEI Research, Japan
[3]Hokkaido University, Japan
[4]Keio University, Japan
[5]Hiroshima Shudo University, Japan

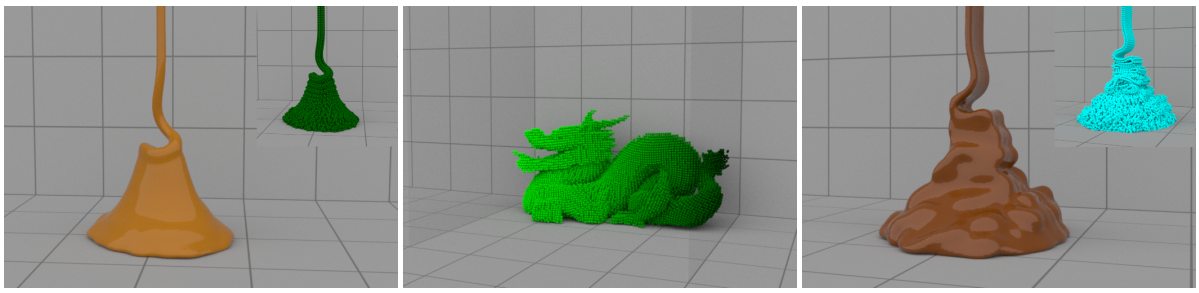

Figure 1: *Viscous fluids simulated with our implicit formulation. Left to right: caramel sauce coiling with a particle view in the inset; a dragon consisting of particles with different viscosities; melted chocolate buckling with a particle view in the inset.*

Abstract

We propose a stable and efficient particle-based method for simulating highly viscous fluids that can generate coiling and buckling phenomena and handle variable viscosity. In contrast to previous methods that use explicit integration, our method uses an implicit formulation to improve the robustness of viscosity integration, therefore enabling use of larger time steps and higher viscosities. We use Smoothed Particle Hydrodynamics to solve the full form of viscosity, constructing a sparse linear system with a symmetric positive definite matrix, while exploiting the variational principle that automatically enforces the boundary condition on free surfaces. We also propose a new method for extracting coefficients of the matrix contributed by second-ring neighbor particles to efficiently solve the linear system using a conjugate gradient solver. Several examples demonstrate the robustness and efficiency of our implicit formulation over previous methods and illustrate the versatility of our method.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Smoothed Particle Hydrodynamics (SPH) is becoming increasingly popular for simulating fluids because of its attractive features including automatic conservation of mass, implicit tracking of surfaces with frequent topology changes, and no need for grid structures or meshes. SPH has been developed in various directions, e.g., enforcing fluid incompressibility [BT07, SP09, ICS*14], handling fluid-fluid and fluid-solid interactions [MSKG05, SSP07, SP08, BTT09, AIA*12, HLW*12, RLY*14], and improving computational efficiency and saving memory usage [APKG07, IABT11, SG11, OK12], and is recognized as a state-of-the-art fluid solver in computer graphics [IOS*14].

Over the past decades, various SPH methods have been proposed and used for a variety of fluid effects in the literature. However, most of these SPH methods assume that fluid is inviscid or slightly viscous; thus an effective SPH method that can simulate highly viscous fluids has not yet been established although we see various viscous materials (e.g., honey, caramel sauce, melted chocolate, lava, machin-

ery oils, and bodily fluids) and their characteristic behaviors on a daily basis. There are two main reasons for this; First, previous SPH methods, e.g., [MCG03, MSKG05, SSP07], drop off-diagonal components of viscous stress tensor to simplify the viscosity term and consequently fail to generate rotational viscous fluid behaviors, such as coiling and buckling, due to neglect of the boundary condition on free surfaces, while leading to inaccurate handling of variable viscosity. This is also true for XSPH [Mon89] and artificial viscosity [Mon92] as they are essentially same as the Laplacian form in simplifying the viscosity term. Since the simplified term is described by Laplacian operator, in this paper we refer to this term as *Laplacian form of viscosity* and the original, unsimplified term as *full form of viscosity*. Second, previous SPH methods, e.g., [PPLT06, ASP*14], suffer from a restriction on available time steps because of their explicit viscosity integration schemes. Andrade et al. [ASP*14] proposed a condition on time steps $\Delta t \leq 0.1\frac{\rho h^2}{8\mu}$ (Δt: time step, ρ: fluid density, h: kernel radius, and μ: dynamic viscosity) to perform numerically stable simulations of viscous fluids with the full form, and this condition makes it difficult to simulate viscous fluids within a reasonable time when higher viscosity and resolutions (smaller kernel radii) are used.

To address the two problems above, we propose a new SPH-based method that uses implicit viscosity integration for the full form for robustly simulating highly viscous fluids. Our method offers the following advantages:

- It is efficient, allowing for use of larger time steps and finer spatial resolutions than explicit integration.
- It is robust and stable, even with large time steps and high viscosities.
- It can generate coiling and buckling phenomena and handle variable viscosity.

We exploit the *variational principle* that automatically enforces the boundary condition on free surfaces to derive our implicit formulation, constructing a sparse linear system with a symmetric positive definite matrix. To efficiently solve the linear system, we also propose a novel method for extracting coefficients of the matrix that includes contributions from first-ring neighbor particles and second-ring neighbor particles (neighbor particles' neighbor particles). Figure 1 demonstrates viscous fluids, simulated using our implicit formulation.

In Eulerian methods, we can easily discretize the full form (divergence of Jacobian of velocity) at a time using finite difference due to the staggered stress arrangement as in [BB08]. On the other hand, in SPH, we first need to compute Jacobian of velocity, and then compute divergence of Jacobian of velocity, separately applying SPH formulations to both steps. Although computing these two steps is easy for explicit integration [PPLT06, ASP*14], significant complexity is involved with implicit integration, and we need to take into account contributions from both of first-ring and second-ring neighbors to construct a linear system. Because of this complexity unique to SPH discretization and derivation of the implicit formulation, no work has been proposed regardless of the apparent simplicity of the idea in adopting implicit integration and demand for robust SPH-based simulators. To the best of our knowledge, our method is the first SPH method that uses implicit integration for the full form of viscosity, and is also the first method that extracts matrix coefficients contributed by second-ring neighbors.

2. Related Work

Eulerian viscous fluids. Carlson et al. [CMVHT02] first enabled stable simulations of highly viscous fluids with free surfaces by solving the Laplacian form of viscosity using implicit integration. Later, Rasmussen et al. [REN*04] proposed an implicit-explicit scheme for the full form of viscosity to correctly handle variable viscosity at the expense of numerical stability. Batty and Bridson [BB08] proposed a fully implicit viscosity integration scheme for the full form, making it possible to take larger time steps, handle variable viscosity, and generate coiling and buckling. This method was extended by Batty and Houston [BH11] for an adaptive tetrahedral fluid simulator. In the work of Stomakhin et al. [SSJ*14], a viscosity term was also solved using implicit integration in the framework of Material Point Method.

Lagrangian viscous fluids. We categorize Lagrangian methods into five groups: Lagrangian Finite Element Methods (Lagrangian FEM), dimensionally reduced discrete methods, spring-based methods, deformation-based methods, and SPH methods. Our method belongs to the SPH methods.

Lagrangian FEM has been used to accurately simulate viscous fluids, and various developments have been done in the literature. Bargteil et al. [BWHT07] proposed an efficient remeshing method to reduce the cost of time-consuming remeshing process, and Wojtan and Turk [WT08] improved the remeshing method of Bargteil et al. [BWHT07]. Wicke et al. [WRK*10] proposed a local remeshing method to keep the number of tetrahedra small. Clausen et al. [CWSO13] proposed a Lagrangian FEM that can handle elastic, plastic, and fluid materials in a unified manner.

To accurately simulate viscous threads and sheets, Bergou et al. [BAV*10] and Batty et al. [BUAG12] proposed dimensionally reduced discrete methods and generated coiling and buckling, respectively, limiting the dimension of materials that they can simulate.

Spring-based methods have been used because of its conceptual simplicity. Miller and Pearce [MP89] and Terzopoulos et al. [TPF91] proposed a spring-based model that computes repulsion and attraction forces between particles. Clavet et al. [CBP05] extended this model to simulate materials that exhibit elasticity, viscosity, and plasticity. Takahashi et al. [TNF14] also simulated such materials in a unified framework of Position-based dynamics.

Gerszewski et al. [GBB09] proposed a deformation-based method that approximates motions of neighbor particles based on deformations of particle configurations for reproducing elastoplastic materials. Their method was extended by Zhou et al. [ZLKW13] to improve its numerical stability using implicit integration and by Jones et al. [JWJ*14] to handle varying mass materials.

Desbrun and Gascuel [DG96] used SPH to simulate viscous materials. Müller et al. [MCG03] proposed the Laplacian form of viscosity and simulated slightly viscous fluid. The Laplacian form was also used in [MSKG05, SSP07]. Paiva et al. [PPLT06] proposed the full form of viscosity and accurately simulated viscous fluids. The full form was also used in the work of Andrade et al. [ASP*14]. Rafiee et al. [Raf07] presented a method based on a Maxwell model to simulate coiling of viscoelastic fluids. Dagenais et al. [DGP12] simulated viscous fluid motions by adding extra forces that move particles to their original positions. In Astrophysics, Monaghan [Mon97] and Laibe and Price [LP12] used pair-wise implicit formulations for artificial viscosity. Their methods consider only pair-wise particles for implicit integration, and thus can be numerically unstable. In Computational Mechanics, Fan et al. [FTZ10] proposed an implicit scheme for simulating viscous fluids using SPH. However, they computed viscous stress with gradient of velocity, not Jacobian of velocity, and hence their method is essentially equivalent to a method that uses the Laplacian form.

Paiva et al. [PPLT06] and Andrade et al. [ASP*14] solved the full form of viscosity. However, our method differs from theirs in that theirs used explicit integration whereas ours uses implicit integration to improve the robustness of the simulation and enable use of larger time steps with higher viscosities and finer spatial resolutions than these methods.

3. Fundamentals for Simulating Viscous Fluids

Formulations. We aim to simulate incompressible, highly viscous fluids using SPH. In the Lagrangian setting, the Navier-Stokes equations for particle i can be described as

$$\rho_i \frac{d\mathbf{u}_i}{dt} = -\nabla p_i + \nabla \cdot \mathbf{s}_i + \frac{\rho_i}{m}\mathbf{F}_i^{\text{ext}}, \tag{1}$$

$$\mathbf{s}_i = \mu_i \left(\nabla \mathbf{u}_i + (\nabla \mathbf{u}_i)^T \right), \tag{2}$$

where ρ_i denotes density of particle i, t time, $\mathbf{u}_i = [u_i, v_i, w_i]^T$ velocity, p_i pressure, $\mathbf{s}_i$ viscous stress tensor, m mass (we use a constant mass for all particles), $\mathbf{F}_i^{\text{ext}}$ external force, and μ_i dynamic viscosity. The combination of the second term on the right side in Eq. (1), and Eq. (2) is the full form of viscosity and is required to handle variable viscosity [REN*04] and rotational behaviors [BB08]. We separate the terms in Eq. (1) to independently solve them, taking the standard approach of operator splitting as with Eulerian methods [BB08, BH11], and enforce fluid incompressibility using a particle-based incompressible fluid solver, e.g., [SP09, ICS*14].

To generate rotational behaviors by solving Eq. (1), we need to consider the boundary condition that there is no traction on free surfaces [BB08]. In other words, as a boundary condition for Eq. (1), we must satisfy $(-p_i\mathbf{I} + \mathbf{s}_i)\mathbf{n}_i = 0$ ($\mathbf{n}_i$: normal to the free surface). As with [BB08, SSJ*14], we decouple these terms and independently enforce the boundary condition for pressure $-p_i\mathbf{I}\mathbf{n}_i = 0$ (we enforce this by setting pressures of surface particles to 0), and for viscosity $\mathbf{s}_i\mathbf{n}_i = \mu_i \left(\nabla \mathbf{u}_i + (\nabla \mathbf{u}_i)^T \right) \mathbf{n}_i = 0$. Since our method is based on the variational principle, the boundary condition for viscosity is automatically enforced by solving Eq. (1) (see e.g., [BB08, SSJ*14]).

Algorithm. First, we apply only external force $\mathbf{F}_i^{\text{ext}}$ and obtain first intermediate velocity $\mathbf{u}_i^*$. Then, we solve viscosity using our implicit formulation (see § 4) and obtain second intermediate velocity $\mathbf{u}_i^{**}$ with intermediate viscous stress tensor $\mathbf{s}_i^{**}$:

$$\mathbf{u}_i^{**} = \mathbf{u}_i^* + \frac{\Delta t}{\rho_i}\nabla \cdot \mathbf{s}_i^{**}, \tag{3}$$

$$\mathbf{s}_i^{**} = \mu_i \left(\nabla \mathbf{u}_i^{**} + \left(\nabla \mathbf{u}_i^{**} \right)^T \right). \tag{4}$$

Next, to enforce fluid incompressibility, we compute pressure p_i using a particle-based fluid solver with the boundary condition for pressure, and compute pressure force $\mathbf{F}_i^p$ [Mon92]. Finally, particle velocity $\mathbf{u}_i$ and position $\mathbf{x}_i = [x_i, y_i, z_i]^T$ are integrated using Euler-Cromer scheme. We summarize a full procedure of our method in Algorithm 1.

Algorithm 1 Procedure of our method

```
// j: neighbor particle of i
// W_ij: kernel with a kernel radius h
for all particle i do
    find neighbor particles
for all particle i do
    apply external force u_i^* = u_i^t + Δt F_i^ext / m
for all particle i do
    solve viscosity using Eqs. (3) and (4) // § 4
for all particle i do
    compute p_i using a particle-based fluid solver
for all particle i do
    compute F_i^p = -m^2 Σ_j ( p_i/ρ_i^2 + p_j/ρ_j^2 ) ∇W_ij
for all particle i do
    integrate particle velocity u_i^{t+1} = u_i^** + Δt F_i^p / m
    integrate particle position x_i^{t+1} = x_i^t + Δt u_i^{t+1}
```

Unlike [BB08] that enforces fluid incompressibility twice, we enforce fluid incompressibility once in one simulation step as in [BH11] because our method can sufficiently maintain incompressibility over all simulation steps (when fluid incompressibility was enforced with a tolerable density error of 0.1%, the maximum density deviation was lower than 0.5%), and we observed indiscernible differences between enforcing incompressibility once and twice.

4. Implicit Formulation for Full Form of Viscosity

In this section, we first describe how to solve the viscosity term using implicit integration in the SPH framework while constructing a linear system (§ 4.1). Next, we explain sparsity of the coefficient matrix (§ 4.2), and our solver and coefficient extraction method for the system (§ 4.3). Then, we give implementation details and show our algorithm for solving our implicit formulation (§ 4.4).

4.1. Implicit Integration for Full Form of Viscosity

We derive an SPH-based implicit viscosity formulation from Eqs. (3) and (4). Hereafter, we drop the symbol $**$ for readability.

Since the boundary condition for viscosity is automatically enforced because of the variational principle [BB08, SSJ*14], we directly discretize Eqs. (3) and (4) using implicit integration in the SPH framework:

$$\mathbf{u}_i = \mathbf{u}_i^* + m\Delta t \sum_j \left(\frac{\mathbf{s}_i}{\rho_i^2} + \frac{\mathbf{s}_j}{\rho_j^2} \right) \nabla W_{ij}, \quad (5)$$

$$\mathbf{s}_i = \mu_i \sum_j V_j \left((\mathbf{u}_j - \mathbf{u}_i) \nabla W_{ij}^T + \nabla W_{ij} (\mathbf{u}_j - \mathbf{u}_i)^T \right). \quad (6)$$

By substituting $\mathbf{s}_i$ in Eq. (6) into Eq. (5) and arranging the terms in these equations, we obtain an implicit formulation:

$$\mathbf{u}_i + \hat{m} \sum_j \left(\hat{\mu}_i \mathbf{Q}_{ij} + \hat{\mu}_j \mathbf{Q}_{jk} \right) \nabla W_{ij} = \mathbf{u}_i^*, \quad (7)$$

$$\mathbf{Q}_{ij} = \begin{bmatrix} 2\sum_j a_{ij,x} u_{ij} & q_{ij,xy} & q_{ij,xz} \\ q_{ij,xy} & 2\sum_j a_{ij,y} v_{ij} & q_{ij,yz} \\ q_{ij,xz} & q_{ij,yz} & 2\sum_j a_{ij,z} w_{ij} \end{bmatrix}, \quad (8)$$

$$q_{ij,xy} = \sum_j \left(a_{ij,y} u_{ij} + a_{ij,x} v_{ij} \right), q_{ij,xz} = \sum_j \left(a_{ij,z} u_{ij} + a_{ij,x} w_{ij} \right),$$

$$q_{ij,yz} = \sum_j \left(a_{ij,z} v_{ij} + a_{ij,y} w_{ij} \right),$$

where $\hat{m} = m\Delta t$, $\hat{\mu}_i = \mu_i / \rho_i^2$, k is a neighbor particle of j, $\mathbf{a}_{ij} = [a_{ij,x}, a_{ij,y}, a_{ij,z}]^T = V_j \nabla W_{ij} = V_j [\nabla W_{ij,x}, \nabla W_{ij,y}, \nabla W_{ij,z}]^T$, $u_{ij} = u_i - u_j$, $v_{ij} = v_i - v_j$, and $w_{ij} = w_i - w_j$. $\mathbf{Q}_{ij}$ is symmetrical due to the symmetrical property of $\mathbf{s}_i$. This implicit formulation Eq. (7) is a linear system and can be rewritten in a matrix form as $\mathbf{CU} = \mathbf{U}^*$ ($\mathbf{C}$ is a coefficient matrix and $\mathbf{U} = [\ldots, u_i, v_i, w_i, \ldots]^T$). Let N denote the number of fluid particles, the size of $\mathbf{C}$ is $3N \times 3N$, and that of $\mathbf{U}$ is $3N \times 1$. We assign serial numbers (id) to particles to specify locations in a coefficient matrix.

4.2. Sparsity of Coefficient Matrix

In SPH, particle i interacts with its neighbor particles within its kernel radius, namely inside of its kernel sphere. Since our formulation requires viscous stress of particles i and j to compute velocity update for particle i using Eq. (5), not only i's neighbors J_i (a set of particle j) but j's neighbors K_{ij} (a set of particle k) must be taken into account. In short, we need to include contributions from first-ring neighbors J_i and second-ring neighbors K_{ij} to compute i's velocity update and construct a linear system. This setup is illustrated in Figure 2. Assuming that particles i and j are spherically surrounded by others, i and j generally have 30-40 first-ring neighbors within their kernel radius h [SP09]. Since particle j can exist anywhere within i's kernel sphere, the total number of second-ring neighbors of particle i without overlaps can be 240-320. This is because the maximum (total) number of second-ring neighbors of i without overlaps is smaller than the number of particles within the largest Minkowski sum of i's kernel sphere and js' kernel spheres, and the largest Minkowski sum M_i is a sphere whose center is $\mathbf{x}_i$ and radius is $2h$. Since all particles in J_i and K_{ij} are included in M_i, each particle can interact with up to 320 particles. Hence, non-zero values for each velocity component can be 960, as there are 320 particles in M_i and each has three velocity components. Although this number is much larger than the number of non-zero values in grid-based methods, and particle-based methods that involve only first-ring neighbors, our coefficient matrix is still a sparse matrix.

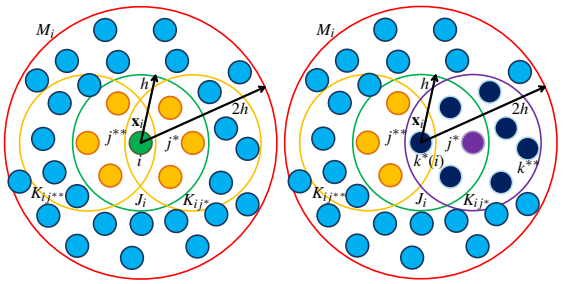

Figure 2: *Illustration for first- and second-ring neighbors. Left: particle i (green) directly interacts with i's first-ring neighbors J_i (particle j colored in orange) inside of i's kernel sphere with its radius h shown as a green circle. Right: particle j^* (purple) has neighbor particles (within j^*'s kernel sphere shown as a purple circle), which are second-ring neighbors K_{ij^*} for particle i. Possible second-ring neighbors for particle i exist within the largest Minkowski sum (M_i) of J_i and K_{ij}, which is a sphere shown as a red circle, and M_i's center and radius are $\mathbf{x}_i$ and $2h$, respectively.*

4.3. Solver and Coefficient Extraction

Solver. Since the linear system constructed by our formulation is sparse and also symmetric positive definite, we use a conjugate gradient (CG) solver (though we also tested Jacobi method and Modified Incomplete Cholesky Conjugate Gradient (MICCG), they did not work well. See § 6). Unlike the CG method described in [Ihm13], which repeatedly computes matrix coefficients in the CG algorithm by performing extra particle scans without storing the coefficients, we explicitly construct and preserve a coefficient matrix, namely extract all coefficients in the matrix. This approach enables us to efficiently perform the CG method without extra loops,

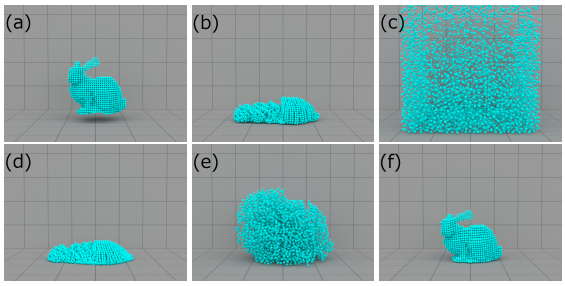

Figure 3: *Numerical stability test with different combinations of time steps and viscosities. (a) initial state, (b) explicit integration [ASP*14] with $\Delta t = 5.0 \times 10^{-6}$ s and $\mu = 1{,}000.0$ kg/(s·m), (c) explicit integration [ASP*14] with $\Delta t = 1.3 \times 10^{-3}$ s and $\mu = 1{,}000.0$ kg/(s·m), (d) implicit integration (our method) with $\Delta t = 1.3 \times 10^{-3}$ s and $\mu = 1{,}000.0$ kg/(s·m), (e) explicit integration [ASP*14] with $\Delta t = 5.0 \times 10^{-6}$ s and $\mu = 50{,}000.0$ kg/(s·m), and (f) implicit integration (our method) with $\Delta t = 1.0 \times 10^{-4}$ s and $\mu = 50{,}000.0$ kg/(s·m).*

thereby improving the performance of the solver. In addition to this advantage, extracting all coefficients allows us to use preconditioning techniques, e.g., algebraic multigrid (AMG), and external libraries, separate solver code from others to improve programming maintainability, and take full advantage of GPGPU, parallelizing matrix-vector multiplications (in both rows and columns).

After a coefficient matrix is constructed, we solve the linear system with the CG method. We terminate iterations in the CG algorithm when a relative residual becomes smaller than a convergence criterion η.

Coefficient extraction. By substituting $\mathbf{Q}_{ij}$ in Eq. (8), we can rewrite Eq. (7) for x component of $\mathbf{u}_i$, u_i as

$$u_i + \hat{m}\sum_j \left(\hat{\mu}_i \left(2\nabla W_{ij,x} \sum_j a_{ij,x} u_{ij} + \nabla W_{ij,y} \sum_j (a_{ij,y} u_{ij} + a_{ij,x} v_{ij}) + \nabla W_{ij,z} \sum_j (a_{ij,z} u_{ij} + a_{ij,x} w_{ij}) \right) + \hat{\mu}_j \left(2\nabla W_{ij,x} \sum_k a_{jk,x} u_{jk} + \nabla W_{ij,y} \sum_k (a_{jk,y} u_{jk} + a_{jk,x} v_{jk}) + \nabla W_{ij,z} \sum_k (a_{jk,z} u_{jk} + a_{jk,x} w_{jk}) \right) \right) = u_i^*. \quad (9)$$

Then, we further convert Eq. (9) into the following equation to straightforwardly extract coefficients $c_{u_i u_i}, c_{v_i u_i}, c_{w_i u_i}, c_{u_j u_i}, c_{v_j u_i}, c_{w_j u_i}, c_{u_k u_i}, c_{v_k u_i}$, and $c_{w_k u_i}$:

$$\begin{bmatrix} c_{u_i u_i} \\ c_{v_i u_i} \\ c_{w_i u_i} \end{bmatrix}^T \begin{bmatrix} u_i \\ v_i \\ w_i \end{bmatrix} + \sum_j \begin{bmatrix} c_{u_j u_i} \\ c_{v_j u_i} \\ c_{w_j u_i} \end{bmatrix}^T \begin{bmatrix} u_j \\ v_j \\ w_j \end{bmatrix} + \sum_k \begin{bmatrix} c_{u_k u_i} \\ c_{v_k u_i} \\ c_{w_k u_i} \end{bmatrix}^T \begin{bmatrix} u_k \\ v_k \\ w_k \end{bmatrix} = u_i^*,$$

$$\begin{aligned}
c_{u_i u_i} &= 1 + \hat{m}\hat{\mu}_i \left(2\omega_{ij,x}\alpha_{ij,x} + \omega_{ij,y}\alpha_{ij,y} + \omega_{ij,z}\alpha_{ij,z} \right), \\
c_{v_i u_i} &= \hat{m}\hat{\mu}_i \omega_{ij,y}\alpha_{ij,x}, \\
c_{w_i u_i} &= \hat{m}\hat{\mu}_i \omega_{ij,z}\alpha_{ij,x}, \\
c_{u_j u_i} &= \hat{m}\Big(-\hat{\mu}_i (2a_{ij,x}\omega_{ij,x} + a_{ij,y}\omega_{ij,y} + a_{ij,z}\omega_{ij,z}) + \\
&\quad \hat{\mu}_j (2\nabla W_{ij,x}\alpha_{jk,x} + \nabla W_{ij,y}\alpha_{jk,y} + \nabla W_{ij,z}\alpha_{jk,z}) \Big), \\
c_{v_j u_i} &= \hat{m}\left(-\hat{\mu}_i a_{ij,x}\omega_{ij,y} + \hat{\mu}_j \nabla W_{ij,y}\alpha_{jk,x} \right), \\
c_{w_j u_i} &= \hat{m}\left(-\hat{\mu}_i a_{ij,x}\omega_{ij,z} + \hat{\mu}_j \nabla W_{ij,z}\alpha_{jk,x} \right), \\
c_{u_k u_i} &= -\hat{m}\sum_j \hat{\mu}_j (2\nabla W_{ij,x} a_{jk,x} + \nabla W_{ij,y} a_{jk,y} + \nabla W_{ij,z} a_{jk,z}), \quad (10) \\
c_{v_k u_i} &= -\hat{m}\sum_j \hat{\mu}_j \nabla W_{ij,y} a_{jk,x}, \quad (11) \\
c_{w_k u_i} &= -\hat{m}\sum_j \hat{\mu}_j \nabla W_{ij,z} a_{jk,x}, \quad (12)
\end{aligned}$$

where $\alpha_{ij} = [\alpha_{ij,x}, \alpha_{ij,y}, \alpha_{ij,z}]^T = \sum_j \mathbf{a}_{ij}$ and $\omega_{ij} = [\omega_{ij,x}, \omega_{ij,y}, \omega_{ij,z}]^T = \sum_j \nabla W_{ij}$. We use $c_{u_i u_i}$ to denote a coefficient of u_i to u_i, and $c_{v_i u_i}$ a coefficient of v_i to u_i, and similarly define other coefficients. The other components (y and z) and 2D version can be straightforwardly derived. By scanning particles i, j, and k, we extract all coefficients for the linear system (see Appendix A for details).

4.4. Implementation Details and Algorithm

We summarize procedures of our implicit formulation in Algorithm 2. To handle solid boundaries, we use solid particles arranged on object surfaces. When fluid particles collide with solid particles, we use explicit viscosity integration for fluid particles with low viscosity while using Dirichlet boundary condition similar to [BB08, SSJ*14, ASP*14], namely setting averaged solid particle velocities $\mathbf{u}^{\text{solid}}$ to fluid particles if viscosity of the fluid particles is higher than a criterion $\mu^{\text{Dirichlet}}$. As a structure of a sparse matrix, we use compressed sparse row (CSR) and reserve sufficient memory for each particle so that the matrix construction can be parallelized over particle i.

Algorithm 2 Algorithm for solving viscosity

```
assemble the matrix // see Appendix A
solve the linear system with CG
for all fluid particle i do
    if μ^Dirichlet < μ_i ∧ neighbor solid particle exists then
        enforce solid boundary condition u_i = u^solid
```

5. Results

Implementation. We implemented our method in C++ and parallelized it using Open MP 2.0. We adopted SPH kernels proposed in [MCG03] and used surface tension force

Table 1: *Simulation parameters and performance. N: the number of particles, μ $(kg/(s \cdot m))$: dynamic viscosity of particles, Δt (s): time step, and t^{visc} (s) and t^{total} (s): simulation time for viscosity and total simulation time per frame, respectively.*

Figure	Viscosity form	Integration	N	μ	Δt	t^{visc}	t^{total}
3 (b)	Full	Explicit	5.5k	1,000.0	5.0×10^{-6}	6.3	51.7
3 (c)	Full	Explicit	5.5k	1,000.0	1.3×10^{-3}	0.0	0.5
3 (d)	Full	Implicit	5.5k	1,000.0	1.3×10^{-3}	15.0	15.4
3 (e)	Full	Explicit	5.5k	50,000.0	5.0×10^{-6}	2.9	40.1
3 (f)	Full	Implicit	5.5k	50,000.0	1.0×10^{-4}	361.4	366.1
5	Full	Implicit	47.4k	up to 800.0	1.0×10^{-4}	815.2	836.1
6 (a), (c)	Laplacian	Implicit	up to 19.6k	600.0	2.0×10^{-4}	20.4	59.3
6 (b), (d)	Full	Implicit	up to 46.5k	600.0	2.0×10^{-4}	287.1	314.1
7 (a), (c)	Full	Implicit	up to 34.6k	100.0	2.0×10^{-4}	135.4	142.0
7 (b), (d)	Full	Implicit	up to 23.4k	600.0	2.0×10^{-4}	116.3	120.7

presented in [BT07]. We adopted Implicit Incompressible SPH (IISPH) [ICS*14] as an incompressible fluid solver and used the fluid solid coupling method proposed in [AIA*12]. We used a variant of the z-index neighbor search method presented in [IABT11]. In addition to our viscosity formulation, we also used artificial viscosity to stabilize simulations [Mon92].

Setting. We executed all the scenes on a PC with a 4-core Intel Core i7 3.40 GHz CPU and RAM 16.0 GB, and rendered all the figures using a physically-based renderer, *Mitsuba*. We used fixed time steps and set a convergence criterion as $\eta = 1.0 \times 10^{-4}$. Simulation parameters and performance are listed in Table 1, where the surface reconstruction and rendering are not included in performance measurement.

5.1. Numerical Stability

To verify the numerical stability of our implicit formulation over the previous method [ASP*14] that uses explicit viscosity integration, we performed a simple test, as shown in Figure 3, where a viscous bunny was dropped onto the ground, and the initial state is shown in Figure 3 (a). In this scene, we chose Δt satisfying the viscosity condition ($\Delta t \leq 0.1 \frac{\rho h^2}{8\mu}$) given in [ASP*14], and used IISPH as a fluid solver of [ASP*14] for fair comparison. The method of [ASP*14] can generate a plausible behavior of the bunny with a small time step and low viscosity, as shown in Figure 3 (b). However, when a large time step or high viscosity is used, their method easily fails to simulate the bunny, as shown in Figures 3 (c) and (e). By contrast, our implicit method successfully simulates the bunny with a much larger time step (Figure 3 (d)), and with a large time step and high viscosity (Figure 3 (f)).

5.2. Performance

We compared performance of our method and the previous method [ASP*14] in Figure 4 using the bunny scene shown in Figures 3 (b) and (d). Because of our robust implicit formulation, we can take a 260.0 times larger time step than the method of [ASP*14], and our computation is 3.4 times faster than theirs although per step cost of our implicit method is more expensive than that of explicit methods. Since time steps for [ASP*14] are restricted by the viscosity condition above, our method can be more advantageous for scenes with higher viscosities and finer resolutions.

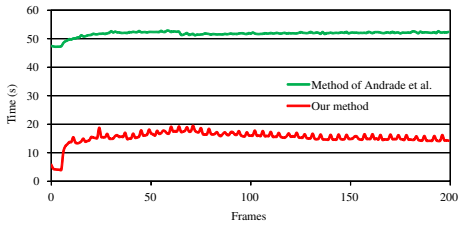

Figure 4: *Performance profile for Figures 3 (b) and (d).*

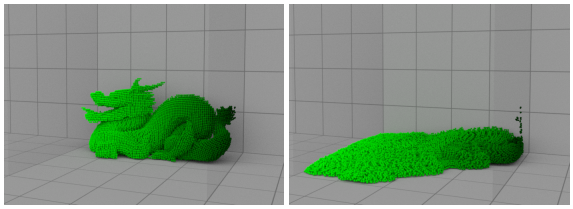

Figure 5: *A dragon consisting of particles with different viscosities from 0.0 to 800.0 kg/(s·m). Light (dark) green particles represent low (high) viscosity.*

5.3. Variable Viscosity

Figure 5 illustrates an example of a dragon consisting of particles with different viscosities from 0.0 (light green) to 800.0 kg/(m·s) (dark green). Particles flow at different rates depending on particle velocities.

5.4. Buckling and Coiling

We performed a buckling test to demonstrate that our implicit formulation can generate buckling of a viscous material while the Laplacian form with implicit integration fails

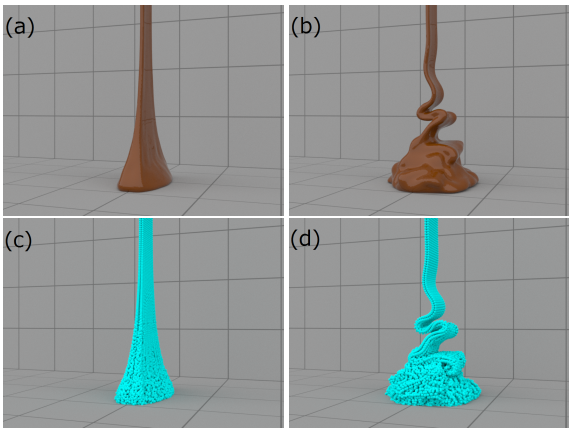

Figure 6: *Buckling test for comparison of our method with the Laplacian form using implicit integration. (a) Laplacian form with meshes. (b) our method with meshes. (c) Laplacian form with particles. (d) our method with particles.*

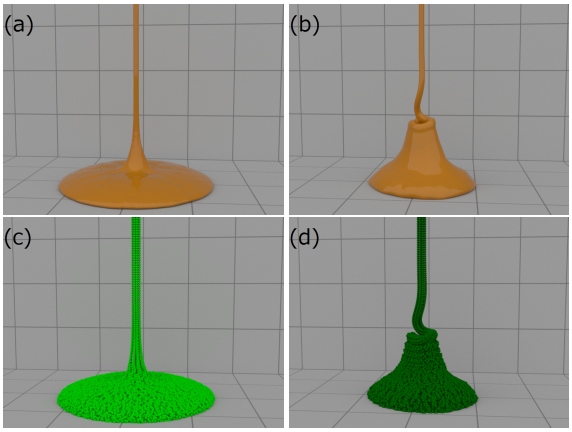

Figure 7: *Coiling test for different viscosities. (a) Low viscosity with meshes. (b) High viscosity with meshes. (c) Low viscosity with particles. (d) High viscosity with particles. Light (dark) green particles represent low (high) viscosity.*

to generate a buckling phenomenon. Figure 6 illustrates a dropped chocolate onto the ground, where (a) and (c) are simulated with the Laplacian form, and (b) and (d) are with our formulation. The chocolate simulated with the Laplacian form does not bend, and exhibits unnatural fluid flows regardless of its high viscosity. By contrast, our implicit formulation successfully generates natural chocolate buckling.

We also performed a coiling test with different viscosities in Figure 7. In this scene, we poured caramel sauce with low and high viscosity onto the ground. While caramel sauce with low viscosity ((a) and (c)) behaves similar to the results generated with the Laplacian form in Figures 6 (a) and (c), caramel sauce with high viscosity exhibits a coiling phenomenon ((b) and (d)).

6. Discussions and Limitations

Robustness. Our implicit formulation significantly improves the robustness of viscosity integration and allows us to avoid the time step restriction for explicit integration [ASP*14]. Our method introduces a much more relaxed restriction on time steps, which depends only slightly on viscosity and spatial resolutions. Consequently, our method may not generate plausible fluid behaviors, when very large time steps and very high viscosity and resolutions are used, even though an implicit formulation for the Laplacian form (which is unconditionally stable) can perform stable simulations with the same condition.

From our experiments, we deduce that one decisive factor for this weaker robustness is due to second-ring neighbors. To examine the factor, we consider solving differential equations with Laplacian operator by using SPH formulations for Laplacian, and also solving the equations by separating Laplacian into divergence and gradient, applying SPH formulations for both of divergence and gradient operators, and including second-ring neighbors, because $\beta\nabla^2\phi \approx \nabla\cdot(\beta\nabla\phi)$ (β and ϕ are arbitrary quantities), and this decomposition is numerically similar to the relation of the Laplacian form $\mu\nabla^2\mathbf{u}$ and full form $\nabla\cdot(\mu\nabla\mathbf{u}+\mu(\nabla\mathbf{u})^T)$. We actually tested these with the Laplacian form of viscosity and heat equations, and as expected, we observed numerical instability with larger time steps and higher coefficients and resolutions when Laplacian operator was decomposed while we were able to perform stable simulations with undecomposed Laplacian operator under the same condition.

Solver. In addition to CG, we tested Jacobi method and MICCG to solve our linear system. Although we were able to solve the system using the two solvers, there are a few problems to note. First, Jacobi method worked for slightly viscous fluids with smaller time steps under low spatial resolutions. However, since the convergence rate of Jacobi method is slow, Jacobi method failed to solve the system with large time steps and high viscosity and spatial resolutions that can make a coefficient matrix ill-conditioned. In contrast to Jacobi method, MICCG worked for such an ill-conditioned matrix, and the rate of convergence was actually faster than our CG solver. However, MICCG requires Cholesky factorization that is costly (and difficult to parallelize), especially with our matrix that includes a great number of non-zero values (see § 4) although serial forward and back substitution steps are not so slow compared to other steps in the CG algorithm. In our experiments, Cholesky factorization occupied more than 90% of computational time for our viscosity solver including coefficient extractions, and therefore our non-preconditioned CG solver was more than 6 times faster than MICCG. A similar behavior that the incomplete Cholesky preconditioner performed worse due to the cost of constructing a preconditioning matrix for particle-based methods (many non-zeros) was also reported in [HLW*12].

Performance. Solving our viscosity formulation generally occupies more than 90% of the whole computational time, and this weakens one of our advantages of efficient computation with larger time steps. Our viscosity formulation consists of extracting coefficients through triple loops and solving a linear system, and they occupy around 30% and 70% of computational time, respectively, for Figure 3 (f). To accelerate the speed of coefficient extractions, an efficient algorithm for extracting coefficients involving second-ring neighbors would be useful. One possible fast algorithm is to avoid triple loops when extracting coefficients, and we can avoid them by using precomputations for particle k. However, these computations require additional storage and scans over the storage, making coefficient extractions more complex. Consequently, benefits of the precomputations over our method can be lost, or extracting coefficients with the precomputations can be more costly than ours. As for solving a linear system, using a faster solver or a low cost and effective preconditioner would be helpful.

Memory. Preserving a coefficient matrix requires a large memory (e.g., 12 GB memory for 500k particles, due to 1k of 8 byte double values for 3 velocity components of 500k particles). However, this is not a big issue with current memory capacity, given advantages explained in § 4.3.

Scalability. Another issue to note is that the size of a matrix grows proportionally to the number of particles. It has a scaling factor larger than grid-based methods and particle-based methods that involve only first-ring neighbors. On larger scenes with up to 200k particles, the memory usage and computational cost for coefficient extraction increased at nearly 1.0x linear-scale with respect to the number of particles – similar to grid-based methods and particle-based methods involving only first-ring neighbors – whereas computational cost for solving linear system increased at around 1.1x scale. This is because increased number of particles requires more CG iterations and thus more computational time. Although computational cost might increase superlinearly when more particles are used, using AMG preconditioners (which cannot be used without explicit matrix preservation) can potentially address this problem.

7. Conclusion and Future Work

We proposed a new SPH-based implicit formulation for the full form of viscosity. Our method enables efficient and stable viscous fluid simulations with larger time steps and higher viscosities and resolutions than previous methods that use explicit integration while handling variable viscosity and generating coiling and buckling. We additionally presented a novel coefficient extraction method for a sparse matrix that involves second-ring neighbors to efficiently solve a linear system with a CG solver. By taking advantage of our implicit formulation and coefficient extraction method, we achieved an accelerated performance by a factor of 3.4.

For future work, we plan to implement our method using GPGPU techniques to accelerate constructing and solving the linear system. In particular, our explicit matrix preservation allows us to take full advantage of dynamic parallelism, fully parallelizing matrix-vector multiplications. Additionally, finding better solvers, efficient and effective preconditioners as well as fast coefficient extraction methods would be promising. Similar to improving memory and computational efficiency for enforcing incompressibility in SPH fluids, using multi-sized particles, domain decomposition, and background grids could also further optimize performance.

Our coefficient extraction method can be applied to not only SPH, but also other point-based methods that involve second-ring neighbors (IISPH [ICS*14] and mesh smoothing [DMSB99]). Extending our method to accelerate such problems could also lead to interesting future research directions.

Acknowledgements

This work is supported in part by JASSO for Study Abroad, JST CREST, JSPS KAKENHI (Grant-in-Aid for Scientific Research(A)26240015), U.S. National Science Foundation, and UNC Arts and Sciences Foundation. We would like to thank Ryoichi Ando and anonymous reviewers for their valuable suggestions and comments.

References

[AIA*12] AKINCI N., IHMSEN M., AKINCI G., SOLENTHALER B., TESCHNER M.: Versatile rigid-fluid coupling for incompressible SPH. *ACM Transactions on Graphics 31*, 4 (2012), 62:1–62:8. 1, 6

[APKG07] ADAMS B., PAULY M., KEISER R., GUIBAS L. J.: Adaptively sampled particle fluids. *ACM Transactions on Graphics 26*, 3 (2007). 1

[ASP*14] ANDRADE LUIZ F. D. S., SANDIM M., PETRONETTO F., PAGLIOSA P., PAIVA A.: SPH fluids for viscous jet buckling. In *Proceedings of SIBGRAPI 2014* (2014), pp. 65–72. 2, 3, 5, 6, 7

[BAV*10] BERGOU M., AUDOLY B., VOUGA E., WARDETZKY M., GRINSPUN E.: Discrete viscous threads. *ACM Transactions on Graphics 29*, 4 (2010), 116:1–116:10. 2

[BB08] BATTY C., BRIDSON R.: Accurate viscous free surfaces for buckling, coiling, and rotating liquids. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008), pp. 219–228. 2, 3, 4, 5

[BH11] BATTY C., HOUSTON B.: A simple finite volume method for adaptive viscous liquids. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2011), pp. 111–118. 2, 3

[BT07] BECKER M., TESCHNER M.: Weakly compressible SPH for free surface flows. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2007), pp. 209–217. 1, 6

[BTT09] BECKER M., TESSENDORF H., TESCHNER M.: Direct forcing for lagrangian rigid-fluid coupling. *IEEE Transactions on Visualization and Computer Graphics 15*, 3 (2009), 493–503. 1

[BUAG12] BATTY C., URIBE A., AUDOLY B., GRINSPUN E.: Discrete viscous sheets. *ACM Transactions on Graphics 31*, 4 (2012), 113:1–113:7. 2

[BWHT07] BARGTEIL A. W., WOJTAN C., HODGINS J. K., TURK G.: A finite element method for animating large viscoplastic flow. *ACM Transactions on Graphics 26*, 3 (2007). 2

[CBP05] CLAVET S., BEAUDOIN P., POULIN P.: Particle-based viscoelastic fluid simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2005), pp. 219–228. 2

[CMVHT02] CARLSON M., MUCHA P. J., VAN HORN III R. B., TURK G.: Melting and flowing. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2002), pp. 167–174. 2

[CWSO13] CLAUSEN P., WICKE M., SHEWCHUK J. R., O'BRIEN J. F.: Simulating liquids and solid-liquid interactions with lagrangian meshes. *ACM Transactions on Graphics 32*, 2 (2013), 17:1–17:15. 2

[DG96] DESBRUN M., GASCUEL M.-P.: Smoothed particles: A new paradigm for animating highly deformable bodies. In *Proceedings of the Eurographics Workshop on Computer Animation and Simulation* (1996), pp. 61–76. 3

[DGP12] DAGENAIS F., GAGNON J., PAQUETTE E.: A prediction-correction approach for stable SPH fluid simulation from liquid to rigid. In *Proceedings of the Computer Graphics International 2012* (2012). 3

[DMSB99] DESBRUN M., MEYER M., SCHRÖDER P., BARR A. H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (1999), SIGGRAPH '99, pp. 317–324. 8

[FTZ10] FAN X.-J., TANNER R., ZHENG R.: Smoothed particle hydrodynamics simulation of non-Newtonian moulding flow. *Journal of Non-Newtonian Fluid Mechanics 165*, 5-6 (2010), 219–226. 3

[GBB09] GERSZEWSKI D., BHATTACHARYA H., BARGTEIL A. W.: A point-based method for animating elastoplastic solids. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2009), pp. 133–138. 3

[HLW*12] HE X., LIU N., WANG G., ZHANG F., LI S., SHAO S., WANG H.: Staggered meshless solid-fluid coupling. *ACM Transactions on Graphics 31*, 6 (2012), 149:1–149:12. 1, 7

[IABT11] IHMSEN M., AKINCI N., BECKER M., TESCHNER M.: A parallel SPH implementation on multi-core CPUs. *Computer Graphics Forum 30*, 1 (2011), 99–112. 1, 6

[ICS*14] IHMSEN M., CORNELIS J., SOLENTHALER B., HORVATH C., TESCHNER M.: Implicit incompressible SPH. *IEEE Transactions on Visualization and Computer Graphics 20*, 3 (2014), 426–435. 6, 8

[Ihm13] IHMSEN M.: *Particle-based Simulation of Large Bodies of Water with Bubbles, Spray and Foam*. PhD thesis, University of Freiburg, 2013. 4

[IOS*14] IHMSEN M., ORTHMANN J., SOLENTHALER B., KOLB A., TESCHNER M.: SPH fluids in computer graphics. In *EUROGRAPHICS 2014 State of the Art Reports* (2014), pp. 21–42. 1

[JWJ*14] JONES B., WARD S., JALLEPALLI A., PERENIA J., BARGTEIL A. W.: Deformation embedding for point-based elastoplastic simulation. *ACM Transactions on Graphics 33*, 2 (2014), 21:1–21:9. 3

[LP12] LAIBE G., PRICE D. J.: Dusty gas with smoothed particle hydrodynamics - II. implicit timestepping and astrophysical drag regimes. *Monthly Notices of the Royal Astronomical Society 420*, 3 (2012). 3

[MCG03] MÜLLER M., CHARYPAR D., GROSS M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2003), pp. 154–159. 2, 3, 5

[Mon89] MONAGHAN J. J.: On the problem of penetration in particle methods. *Journal of Computational Physics 82*, 1 (1989), 1–15. 2

[Mon92] MONAGHAN J. J.: Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics 30* (1992), 543–574. 2, 3, 6

[Mon97] MONAGHAN J. J.: Implicit SPH drag and dusty gas dynamics. *Journal of Computational Physics 138*, 2 (Dec. 1997), 801–820. 3

[MP89] MILLER G., PEARCE A.: Globular dynamics: A connected particle system for animating viscous fluids. *Computers & Graphics 13*, 3 (1989), 305–309. 2

[MSKG05] MÜLLER M., SOLENTHALER B., KEISER R., GROSS M.: Particle-based fluid-fluid interaction. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2005), pp. 237–244. 1, 2, 3

[OK12] ORTHMANN J., KOLB A.: Temporal blending for adaptive SPH. *Computer Graphics Forum 31*, 8 (2012), 2436–2449.

[PPLT06] PAIVA A., PETRONETTO F., LEWINER T., TAVARES G.: Particle-based non-newtonian fluid animation for melting objects. In *Proceedings of SIBGRAPI 2006.* (2006), pp. 78–85. 2, 3

[Raf07] An incompressible SPH method for simulation of unsteady viscoelastic free-surface flows. *International Journal of Non-Linear Mechanics 42*, 10 (2007), 1210–1223. 3

[REN*04] RASMUSSEN N., ENRIGHT D., NGUYEN D., MARINO S., SUMNER N., GEIGER W., HOON S., FEDKIW R.: Directable photorealistic liquids. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2004), pp. 193–202. 2, 3

[RLY*14] REN B., LI C., YAN X., LIN M. C., BONET J., HU S.-M.: Multiple-fluid SPH simulation using a mixture model. *ACM Transactions on Graphics 33*, 5 (2014), 171:1–171:11.

[SG11] SOLENTHALER B., GROSS M.: Two-scale particle simulation. *ACM Transactions on Graphics 30*, 4 (2011), 81:1–81:8. 1

[SP08] SOLENTHALER B., PAJAROLA R.: Density contrast SPH interfaces. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008), pp. 211–218. 1

[SP09] SOLENTHALER B., PAJAROLA R.: Predictive-corrective incompressible SPH. *ACM Transactions on Graphics 28*, 3 (2009), 40:1–40:6. 1, 3, 4

[SSJ*14] STOMAKHIN A., SCHROEDER C., JIANG C., CHAI L., TERAN J., SELLE A.: Augmented MPM for phase-change and varied materials. *ACM Transactions on Graphics 33*, 4 (2014), 138:1–138:11. 2, 3, 4, 5

[SSP07] SOLENTHALER B., SCHLÄFLI J., PAJAROLA R.: A unified particle model for fluid-solid interactions. *Computer Animation and Virtual Worlds 18*, 1 (2007), 69–82. 1, 3

[TNF14] TAKAHASHI T., NISHITA T., FUJISHIRO I.: Fast simulation of viscous fluids with elasticity and thermal conductivity using position-based dynamics. *Computers & Graphics 43* (2014), 21–30. 2

[TPF91] TERZOPOULOS D., PLATT J., FLEISCHER K.: Heating and melting deformable models. *Journal of Visualization and Computer Animation 2* (1991), 68–73. 2

[WRK*10] WICKE M., RITCHIE D., KLINGNER B. M., BURKE S., SHEWCHUK J. R., O'BRIEN J. F.: Dynamic local remeshing for elastoplastic simulation. *ACM Transactions on Graphics 29*, 4 (2010), 49:1–49:11. 2

[WT08] WOJTAN C., TURK G.: Fast viscoelastic behavior with thin features. *ACM Transactions on Graphics 27*, 3 (2008), 47:1–47:8. 2

[ZLKW13] ZHOU Y., LUN Z., KALOGERAKIS E., WANG R.: Implicit integration for particle-based simulation of elasto-plastic solids. *Computer Graphics Forum 32*, 7 (2013), 215–223. 3

Appendix A: Details of Coefficient Extraction

Due to complexity of extracting coefficients and similarity of computing coefficients, we divide them into three groups: G_i ($c_{u_iu_i}, c_{v_iu_i}$, and $c_{w_iu_i}$), G_j ($c_{u_ju_i}, c_{v_ju_i}$, and $c_{w_ju_i}$), and G_k ($c_{u_ku_i}, c_{v_ku_i}$, and $c_{w_ku_i}$).

By scanning particle j once from particle i, we can easily compute and extract coefficients in G_i.

Since computing coefficients in G_j requires $\mathbf{a}_{ij}$ and ω_{ij} (summation of ∇W_{ij} over particle j), and ∇W_{ij} and α_{jk} (summation of $V_k \nabla W_{jk}$ over particle k), we need to compute ω_{ij} and α_{ij} (we can access α_{jk} from particle j if α_{ij} is computed in a previous loop) in advance using another loop unlike the case of G_i. Therefore, we first compute ω_{ij} and α_{ij} in the first loop and then use them to compute and extract coefficients in G_j in the second loop. Computed ω_{ij} and α_{ij} can also be used to obtain coefficients in G_i.

To compute coefficients in G_k, we need to access particle k from particle i. However, since particle k is a neighbor of particle j, we cannot directly access particle k from particle i. Hence, we access particle k via particle j. In that case, however, we cannot take a sum of quantities computed between particles i and j at particle k (another scan for particle j at particle k leads to quadruple loops, which are costly and make coefficient extractions more complex). Therefore, we decompose coefficients $c_{u_ku_i}, c_{v_ku_i}$, and $c_{w_ku_i}$ without taking a sum over particle j and separately extract coefficients at particle k. Assuming that we are accessing particle k^* via particle j^*, we can write a part of coefficients $c_{u_ku_i}|_{j^*k^*}, c_{v_ku_i}|_{j^*k^*}$, and $c_{w_ku_i}|_{j^*k^*}$ for $c_{u_ku_i}, c_{v_ku_i}$, and $c_{w_ku_i}$ from Eqs. (10), (11), and (12) as

$$\begin{aligned} c_{u_ku_i}|_{j^*k^*} &= -\hat{m}\mu_{j^*} \\ (2\nabla W_{ij^*,x} a_{j^*k^*,x} &+ \nabla W_{ij^*,y} a_{j^*k^*,y} + \nabla W_{ij^*,z} a_{j^*k^*,z}), \\ c_{v_ku_i}|_{j^*k^*} &= -\hat{m}\mu_{j^*} \nabla W_{ij^*,y} a_{j^*k^*,x}, \\ c_{w_ku_i}|_{j^*k^*} &= -\hat{m}\mu_{j^*} \nabla W_{ij^*,z} a_{j^*k^*,x}. \end{aligned}$$

By adding the part of coefficients above to the matrix at particle k while also scanning particle j from particle i, we can extract all coefficients of the matrix.

Adding coefficients to the matrix separately does work. However, this approach is inefficient because we generally use structures specialized for a sparse matrix, e.g., CSR and Coordinate list, and adding values to such structures frequently is costly. To avoid this, we use auxiliary storage, which is associated with particle i, to preserve coefficients

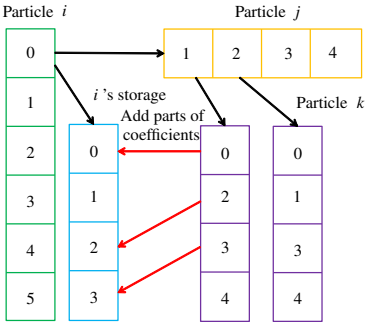

Figure 8: *Illustration of computational flow for extracting coefficients in G_k. Particle k is accessed from particle i via particle j. At particle k, parts of coefficients in G_k are added to a cell in particle i's storage, whose id matches k's id, as shown by red arrows. Black arrows represent accessible particles and storage.*

from particle k to i with k's id. We add parts of coefficients to i's storage at particle k, grouping them based on k's id to minimize the number of adding coefficients to the matrix. Then, after scans over particles j and k from particle i are finished, we add coefficients in G_k to the matrix using the storage. In our experiments, minimizing the number of access to the matrix using the auxiliary storage accelerated our coefficient extractions by a factor of 4.5 as compared to adding coefficients to the matrix directly at particle k. A flow for coefficient extractions for G_k is illustrated in Figure 8.

We also perform similar procedures explained above to extract coefficients $c_{u_iv_i}, c_{v_iv_i}, c_{w_iv_i}, c_{u_jv_i}, c_{v_jv_i}, c_{w_jv_i}, c_{u_kv_i}, c_{v_kv_i}$, and $c_{w_kv_i}$ for v_i and $c_{u_iw_i}, c_{v_iw_i}, c_{w_iw_i}, c_{u_jw_i}, c_{v_jw_i}, c_{w_jw_i}, c_{u_kw_i}, c_{v_kw_i}$, and $c_{w_kw_i}$ for w_i. We show our algorithm for coefficient extraction in Algorithm 3.

Algorithm 3 Algorithm for coefficient extraction

```
initialize a matrix
for all fluid particle i do
    compute μ̂_i, ω_ij and α_ij
compute m̂
for all fluid particle i do
    initialize storage for u_k, v_k, and w_k
    add c_{u_iu_i}, c_{v_iu_i}, c_{w_iu_i}, c_{u_iv_i}, c_{v_iv_i}, c_{w_iv_i}, c_{u_iw_i}, c_{v_iw_i}, and
    c_{w_iw_i} to the matrix
    for all fluid particle j do
        compute ∇W_ij and a_ij
        add c_{u_ju_i}, c_{v_ju_i}, c_{w_ju_i}, c_{u_jv_i}, c_{v_jv_i}, c_{w_jv_i}, c_{u_jw_i}, c_{v_jw_i},
        and c_{w_jw_i} to the matrix
        for all fluid particle k do
            compute a_jk
            add c_{u_ku_i}, c_{v_ku_i}, c_{w_ku_i}, c_{u_kv_i}, c_{v_kv_i}, c_{w_kv_i}, c_{u_kw_i}, c_{v_kw_i},
            and c_{w_kw_i} to the i's storage with k's id
    for all i's storage do
        add c_{u_ku_i}, c_{v_ku_i}, c_{w_ku_i}, c_{u_kv_i}, c_{v_kv_i}, c_{w_kv_i}, c_{u_kw_i}, c_{v_kw_i},
        and c_{w_kw_i} to the matrix using the storage
```