Divergence-Free SPH for Incompressible and Viscous Fluids

Jan Bender and Dan Koschier

Abstract—In this paper we present a novel Smoothed Particle Hydrodynamics (SPH) method for the efficient and stable simulation of incompressible fluids. The most efficient SPH-based approaches enforce incompressibility either on position or velocity level. However, the continuity equation for incompressible flow demands to maintain a constant density and a divergence-free velocity field. We propose a combination of two novel implicit pressure solvers enforcing both a low volume compression as well as a divergence-free velocity field. While a compression-free fluid is essential for realistic physical behavior, a divergence-free velocity field drastically reduces the number of required solver iterations and increases the stability of the simulation significantly. Thanks to the improved stability, our method can handle larger time steps than previous approaches. This results in a substantial performance gain since the computationally expensive neighborhood search has to be performed less frequently. Moreover, we introduce a third optional implicit solver to simulate highly viscous fluids which seamlessly integrates into our solver framework. Our implicit viscosity solver produces realistic results while introducing almost no numerical damping. We demonstrate the efficiency, robustness and scalability of our method in a variety of complex simulations including scenarios with millions of turbulent particles or highly viscous materials.

Index Terms—Fluid simulation, smoothed particle hydrodynamics, divergence-free fluids, incompressibility, viscous fluids, implicit integration

1 Introduction

In the area of computer graphics Smoothed Particle Hydrodynamics (SPH) is an important meshless Lagrangian approach to simulate complex fluid effects. The SPH formalism allows an efficient computation of a certain quantity of a fluid particle by considering only a finite set of neighboring particles. One of the most challenging research topics in the field of SPH methods is the simulation of incompressible fluids. The fact that most fluids we encounter in nature feature incompressible behavior, proves that enforcing incompressibility is essential to produce realistic animations for a wide range of materials. A strongly related research topic is the simulation of highly viscous fluids which represents a class of similarly important liquids, e.g., honey or mud. In this paper we introduce an efficient *divergence-free SPH* (DFSPH) method to simulate incompressible fluids. Moreover, we extend this method by an implicit viscosity solver which allows the simulation of highly viscous fluids.

In order to model incompressible fluids we base our simulation on the incompressible, isothermal Navier-Stokes equations in Lagrangian coordinates

$$\frac{D\rho}{Dt} = 0 \quad \Leftrightarrow \quad \nabla \cdot \mathbf{v} = 0 \tag{1}$$

$$\frac{D\mathbf{v}}{Dt} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{v} + \frac{\mathbf{f}}{\rho}, \tag{2}$$

where $D(\cdot)/Dt$ denotes the material derivative and ρ, p, ν, $\mathbf{v}$ and $\mathbf{f}$ denote density, pressure, kinematic viscosity, velocity and body force density field, respectively. According to Equation (1) an incompressible fluid satisfies the *divergence-free condition* $\nabla \cdot \mathbf{v} = 0$ and therefore has a divergence-free velocity field. Based on the observation that the density must not change over time, i.e., $\frac{D\rho}{Dt} = 0$, and based on the continuity equation $\frac{D\rho}{Dt} = -\rho \nabla \cdot \mathbf{v}$ the equivalence stated in Equation (1) easily follows. Therefore, in theory, a fluid with a divergence-free velocity field is incompressible, as it maintains constant density. However, in practice, incompressibility cannot be guaranteed in SPH simulations by only enforcing the divergence-free condition. The numerical time integration causes numerical errors which sum up over time. The resulting density deviations are not considered in the divergence-free condition and therefore a volume compression cannot be avoided. In order to enforce incompressibility a second condition is required:

$$\rho - \rho_0 = 0. \tag{3}$$

In the following we will refer to this as *constant density condition*. In summary, the incompressible, isothermal Navier-Stoke equations demand a divergence-free velocity field (see Equation (1)). Additionally, a stabilization is required to counteract numerical errors which can be realized by enforcing the constant density condition.

In recent years several implicit SPH solvers which compute pressure forces to counteract volume compression were investigated. Currently, the most efficient pressure solvers only consider the constant density condition. Since this condition solely depends on particle positions, the divergence-free condition is generally not fulfilled (see Fig. 2, left) as demanded by the continuity equation for incompressible flow. A correction of velocity divergence is considered only

• *The authors are with Graduate School CE, TU Darmstadt, Darmstadt 64289, Germany. E-mail: {bender, koschier}@gsc.tu-darmstadt.de.*

Manuscript received 29 Jan. 2016; revised 20 May 2016; accepted 29 May 2016. Date of publication 8 June 2016; date of current version 1 Feb. 2017. Recommended for acceptance by F. Bertails-Descoubes and S. Coros. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TVCG.2016.2578335

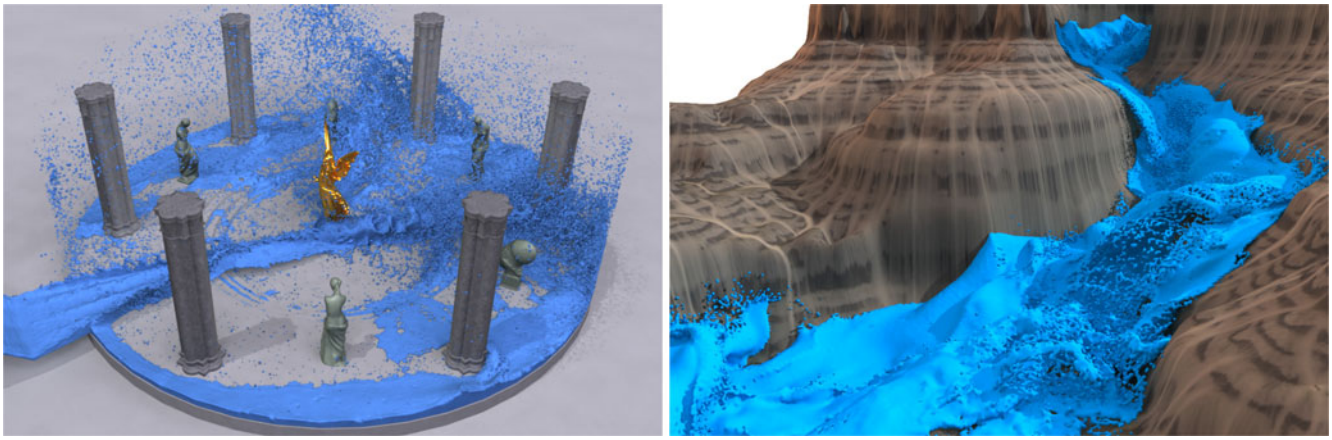


Fig. 1. Our SPH method enforces a divergence-free velocity field which allows a stable simulation of incompressible fluids with high velocities using large time steps. This is demonstrated in a simulation with 2.4 million fast moving fluid particles and a complex boundary consisting of 6 million boundary particles (left). Moreover, our approach only requires 5 seconds per time step in a complex simulation with 5 million fluid particles, 40 million boundary particles and a maximum volume compression of 0.01 percent (right). This is significantly faster than current state-of-the-art SPH methods.

by a few SPH approaches so far. However, they are either comparatively slow or cannot maintain a low density error.

In this paper we introduce a novel stable and efficient SPH method for incompressible flow. In contrast to most previous methods, it satisfies both the constant density condition and the divergence-free condition. Our method combines two pressure solvers. The first solver enforces a divergence-free velocity field (see Fig. 2, right) while the second solver satisfies the constant density condition. This combination has several advantages in SPH simulations. First, the stability of the simulation increases significantly, especially in simulations with fast moving particles (see Fig. 1). The improved stability allows to perform larger time steps. This yields a considerable performance gain as the neighborhood search which is the main bottleneck in SPH simulations has to be performed less frequently. Second, the number of iterations required by the constant density solver decreases significantly when a divergence-free velocity field is enforced. Finally, the divergence-free field allows a more accurate computation of the maximum time step size as the commonly used CFL condition depends on the maximum particle velocity.

In the last years different efficient constant density solvers have been introduced, e.g., Predictive-Corrective Incompressible SPH (PCISPH) [1] or Implicit Incompressible SPH (IISPH) [2]. Nevertheless, we propose a new approach in this paper that operates analogously to our divergence-free solver, and therefore benefits from reusing precomputed coefficients. The fluid simulation based on our combined solvers clearly outperforms state-of-the-art SPH methods and is more than 20 times faster in typical scenarios.

2 Related Work

In this section we will briefly discuss previous approaches related to this paper. For a more detailed discussion, we would like to refer to reader to the work of Bridson [3] for a general survey and to the state of the art report of Ihmsen et al. [4] for SPH-based simulation approaches.

2.1 Equation of State Based Solvers

The first pioneering approaches to simulate fluids using SPH discretizations were proposed by Monaghan [5], [6], [7]. Using an equation of state (EOS), deviations from a given rest density were penalized using a stiffness coefficient in order to weakly enforce incompressibility. Several years later, an EOS-based approach for fluid simulation was introduced to the computer graphics community by Müller et al. [8] and later extended by Adams et al. [9] for spatially adaptive SPH discretizations. In order to restrict the maximum compression using an EOS-based solver, Becker and Teschner [10] precomputed a scenario-dependent stiffness coefficient. However, this may lead to stiff differential equations which restrict the time step size for explicit integration schemes drastically.

2.2 Iterative Incompressibility Solvers Based on Splitting

An alternative strategy, often referred to as the concept of splitting, is to compute the pressure after advecting particles only based on non-pressure forces. Initially, the concept was applied to EOS-based solvers (cf. [3], [11]) only but soon combined with implicit pressure solvers as discussed in the following.

Solenthaler and Pajarola [1] developed an iterative pressure solver based on the splitting concept to approximately limit the maximum density error to a user-defined tolerance. Later, the method was extended by rigid-fluid coupling [12]

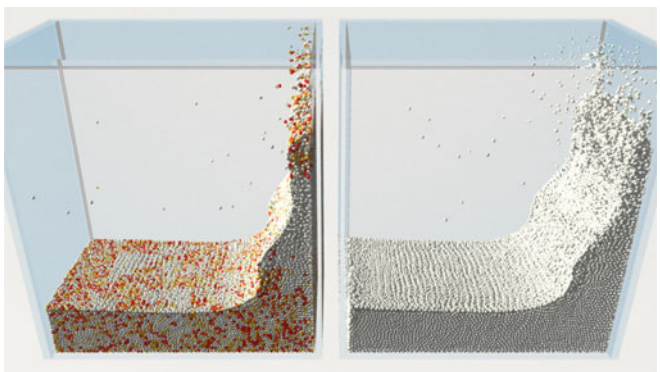


Fig. 2. Velocity divergence comparison of IISPH (left) and DFSPH (right) in a breaking dam simulation with 125k particles. The divergence errors are color coded, where white is the minimum and red is the maximum.

and a novel surface tension model [13]. Macklin and Müller [14] proposed Position Based Fluids (PBF)—a similar approach that iteratively corrects particle positions to enforce incompressibility. A velocity-level formulation using holonomic constraints based on rigid body mechanics was proposed by Bodin et al. [15]. However, due to numerical errors, their regularization approach and velocity-level correction, they reported density errors of up to 17 percent. In contrast to the discussed iterative approaches our method does not only enforce incompressibility on velocity or density level but considers both. Another method eliminating the compression on both levels was recently proposed by Kang and Sagong [16]. They extended the work of Macklin and Müller [14] by additionally projecting the velocity field onto a the divergence-free state. However, the velocity field is not guaranteed to contain zero divergence after a time step as particle positions are updated subsequently to the velocity projection. Moreover, their extension leads to a considerable overhead in comparison to the underlying method of Macklin and Müller while our method yields large speedup factors of up to one order of magnitude, especially in case of large time steps (see Table 2).

2.3 Pressure Projection

Another popular choice to enforce incompressibility is to project the velocity field onto a divergence-free state by solving the pressure Poisson equation (PPE). This method is particularly popular for Eulerian discretizations (cf. [3]). A PPE derived from an approximate pressure projection was proposed by Cummins and Rudman [17]. They solved the resulting linear system using either a conjugate gradient method or a multigrid approach treating the particles as finest level grid. Foster and Fedkiw [18] developed a semi-Lagrangian method where incompressibility is enforced on the simulation grid. They counteracted mass dissipation using a level set and freely moving inertialess particles. Further hybrid approaches using particles and a background grid were developed in the following years [19], [20], [21]. An interesting and more elaborate approach was proposed by Losasso et al. [22]. The PPE was solved on a background grid not only to maintain zero divergence but to enforce a predefined target density. Sin et al. [23] constructed a Voronoi discretization of the PPE from point-samples in each step for solving. In contrast to the discussed approaches, our method does not rely on any background grid. For that reason we do not have to maintain memory consuming data-structures which is especially advantageous for large scenarios as presented in Fig. 1.

In contrast to solve the PPE on a background grid, it may be directly solved on a meshless discretization. Hu et al. [24] proposed an incompressible SPH-based approach for multiphase flows with multistep time integration. In a first halfstep the density error is eliminated using a position-altering iterative gradient descent solver. During the second halfstep, errors in velocity divergence are similarly corrected. However, they applied their method exclusively to non-complex two-dimensional scenarios of up to 14,400 particles while we demonstrate scenarios with over five million particles in three dimensions. A local Poisson solver extending the work of Solenthaler and Parajola [1] was proposed by He et al. [25]. The solver enforces both a divergence-free velocity and constant density. As the integration domain for the local solves is not necessarily equal or a subset of the local kernel function support, they have to recompute the particle neighbors. Moreover, a recomputation of particle neighborhoods is required in each solver iteration resulting in a considerable computational effort. Compared to the method of Solenthaler and Pajarola they are able to achieve a small speedup factor of approximately 1.5 while we experienced speedup factors of more than 20 with DFSPH. Ihmsen et al. [2] proposed a method based on a PPE that iteratively eliminates the density error to 0.1 percent or even less. However, they do not consider resolving velocity divergence which leads to a large number of solver iterations compared to our method and can even cause artifacts in special cases (cf. Section 5). Very recently, an incompressible particle-based approach using power-diagrams was proposed by de Goes et al. [26]. In each simulation step they construct a Voronoi diagram induced by the particle positions in order to enforce incompressibility. However, our approach clearly outperforms the method as the diagram construction is very time consuming.

2.4 Highly Viscous Fluids

Motivated by the Navier-Stokes equation, the standard approach to model viscosity is to discretize the Laplacian of the velocity term. As an alternative, viscous behavior can also be modeled using a strain rate measure or even by artificial viscosity models as discussed in the following.

Based on a Eulerian simulation methodology the Laplacian term was discretized using finite differences with explicit time integration in an early work of Foster and Metaxas [27]. Later, Stam [28] and Carlson et al. [29] employed implicit time updates to maintain stability for large viscosity parameters. In order to extend the method to variable viscosities Rasmussen et al. [30] presented a model based on the strain rate tensor. As pointed out by Batty and Bridson [31], both Laplacian as well as strain rate formulations work only on divergence-free velocity fields. For that reason, they performed a pressure projection precedent and subsequent to the viscosity solve. The method was later extended by a spatially adaptive [32] and a dimension-reduced [33] variant. An early Eulerian approach to simulate viscoelastic materials was proposed by Goktekin et al. [34] using additive decomposition and subsequent evolution of the strain tensor by explicit integration over time.

Following Lagrangian simulation methodologies Monaghan [5] and Morris and Monaghan [35] discretized the Laplacian operator using finite differences of the first derivative within their SPH framework. Later, Müller et al. [8] designed a specific kernel function for the discretization of the viscosity term while Premože et al. [36] proposed a method based on the Moving-Particle Semi-Implicit method. An alternative approach to approximate the Laplacian referred to as XSPH was introduced by Monoghan [5], [6] and later adopted in several works, e.g., [14], [37]. In the case of non-Newtonian fluids, viscous forces can also be modeled based on the deformation tensor [38], [39], [40]. Recently, highly viscous fluids were simulated using a splitting approach combined with an implicit solve of pressure and viscous forces. Using this strategy, Peer et al. [41]

compute the strain rate field throughout the fluid and enforce a target strain rate. Their solver is based on a symmetrized linearization of the velocity field which allows them to decouple the resulting equation system into three distinct linear equation systems. However, the linearization leads to a considerable numerical dissipation of kinetic energy which makes the approach less applicable for moderately viscous fluids compared to our method (cf. Fig. 9). Takahashi et al. [42] also use a model based on a strain rate measure. Adopting the concept of splitting they implicitly integrate viscous forces in an individual solver step in order to maintain a stable simulation.

Highly viscous materials were also considered for hybrid solid-fluid and viscoelastic models. An early spring-based method was presented by Clavet et al. [43] where viscous behavior is enforced using an impulse-based velocity filter. The spring-based concept was later paired with a position-based approach by Takahashi et al. [44] and modified using a velocity filter [45]. Point-based approaches were proposed by Gerszewski et al. [46] and Jones et al. [47] while a mesh-based approach with special treatment for thin features was presented by Wojtan et al. [48]. Lagrangian methods for the simulation of lower-dimensional viscous materials were proposed by Bergou et al. [49] and Batty et al. [33] while Zhu et al. [50] recently developed a codimensional approach.

In this paper we propose an efficient implicit solver to handle high viscosities. Based on a continuous strain rate measure, we capture complex phenomena as demonstrated in Section 5. In contrast to very recent existing SPH-based implicit viscosity solvers, our method produces significantly less numerical damping than the method proposed by Peer et al. [41] while being substantially faster compared to the solver of Takahashi et al. [42]. Moreover, the solver operates analogously to the proposed divergence-free and constant density solvers and is for that reason very easy to integrate.

3 Simulation of Incompressible Fluids

Our fluid simulation is based on the incompressible, isothermal Navier-Stokes equations introduced in Section 1. However, we omit the viscous term on the right hand side of Equation (2). Instead we use the XSPH variant of Schechter and Bridson [37] in order to simulate low viscous fluids like water. Moreover, we introduce an implicit strain rate based viscosity formulation to simulate highly viscous fluids in Section 4.1.

We use the SPH method to spatially discretize Equation (2) as described in the following. Equation (1) represents the incompressibility condition. Our method satisfies this condition by enforcing the divergence-free condition and the constant density condition as described in Sections 3.2 and 3.3, respectively.

Using the SPH concept a quantity A_i at position $\mathbf{x}_i$ is approximated by the values A_j at a set of neighboring points $\mathbf{x}_j$ [6]:

$$A_i \approx \sum_j \frac{m_j}{\rho_j} A_j W_{ij}, \tag{4}$$

where m_j is the mass of particle j, ρ_j is its density and $W_{ij} = W(\mathbf{x}_i - \mathbf{x}_j, h)$ is a smoothing kernel function with support radius h. The spatial derivative of A_i is determined by

$$\nabla A_i \approx \sum_j \frac{m_j}{\rho_j} A_j \nabla W_{ij}. \tag{5}$$

An important quantity in fluid simulation is the density which can be approximated using this concept as:

$$\rho_i = \sum_j \frac{m_j}{\rho_j} \rho_j W_{ij} = \sum_j m_j W_{ij}.$$

The pressure field of a fluid is often computed by the following equation of state:

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

where ρ_0 is the rest density and κ and γ are stiffness parameters. In our work we set the parameter γ to 1 which is a common choice in computer graphics [4], [8], [51]:

$$p_i = \kappa(\rho_i - \rho_0). \tag{6}$$

The pressure field can be directly used to determine pressure forces which counteract a density deviation [6]. However, a high stiffness coefficient κ is required to simulate weakly compressible fluids which leads to stiff differential equations. Therefore, implicit pressure solvers like PCISPH [1] or IISPH [2] were investigated for the simulation of incompressible fluids. Such solvers typically solve a linear system to determine the pressure field. This allows a stable simulation with larger time steps and as a consequence a significant performance gain.

In this paper we introduce a novel implicit SPH approach to simulate incompressible fluids. In contrast to previous methods we use two different solvers: the *divergence-free solver* (see Section 3.2) and the *constant density solver* (see Section 3.3). The first solver enforces a divergence-free velocity field. Since density deviations resulting from numerical errors cannot be corrected in this way, as already discussed in Section 1, our second solver eliminates these density errors by satisfying the constant density condition. In summary, we enforce incompressibility and a divergence-free velocity field as demanded by Equation (1). In both solvers we fulfill the respective condition for each neighborhood independently by computing a stiffness coefficient κ_i (see Equation (6)) and the corresponding pressure forces. To obtain a global solution the solvers process the neighborhoods in a parallel Jacobi fashion which is a common choice in SPH pressure solvers [1], [2], [14].

3.1 Simulation Step

This section is intended to abstractly outline the simulation step. A detailed description of the substeps is given in the following sections. The Navier-Stokes equations for incompressible fluids demand that the constant density condition and the divergence-free condition are satisfied at the end of a simulation step. In order to fulfill the first condition we determine pressure forces which must be integrated twice to get the required position changes. Afterwards we compute pressure forces and integrate them once to make the resulting velocity field divergence-free. Note that performing the density stabilization before computing a divergence-free velocity field does not impose any restrictions since both steps are executed in a loop.

The simulation with our novel method is outlined in Algorithm 1. Before the simulation loop begins we determine all particle neighborhoods N_i using compact hashing [52]. Furthermore, the densities ρ_i and the factors α_i (see Section 3.2) are computed. The value α_i is a common factor which is required by both solvers. This factor only depends on the current positions and therefore it does not change during the iterative process of the solvers. For that reason, it must be computed only once in each simulation step and can be used by both solvers. This reduces the computational effort of the solver iterations considerably.

Algorithm 1. Simulation

```
1: function performSimulation
2:   for all particles i do                          // init neighborhoods
3:     find neighborhoods N_i(0)
4:   for all particles i do                          // init ρ_i and α_i
5:     compute densities ρ_i(0)
6:     compute factors α_i(0)
7:   while (t < t_max) do                            // simulation loop
8:     for all particles i do
9:       compute non-pressure forces F_i^adv(t)
10:    adapt time step size Δt according to CFL condition
11:    for all particles i do                        // predict velocities v_i^*
12:      v_i^* = v_i + Δt F_i^adv / m_i
13:    correctDensityError(α, v^*)                   // ρ^* − ρ_0 = 0
14:    for all particles i do                        // update positions
15:      x_i(t + Δt) = x_i(t) + Δt v_i^*
16:    for all particles i do                        // update neighbors
17:      find neighborhoods N_i(t + Δt)
18:    for all particles i do                        // update ρ_i and α_i
19:      compute densities ρ_i(t + Δt)
20:      compute factors α_i(t + Δt)
21:    correctDivergenceError(α, v^*)                // Dρ/Dt = 0
22:    for all particles i do                        // update velocities
23:      v_i(t + Δt) = v_i^*
```

In the first step of the simulation loop all non-pressure forces $\mathbf{F}^{\text{adv}}$ like gravity, surface tension and viscosity are computed. Then we adapt the time step size according to the Courant-Friedrich-Levy (CFL) condition [6]

$$\Delta t \leq 0.4 \frac{d}{\|\mathbf{v}_{\max}\|},$$

where $\mathbf{v}_{\max}$ is the maximum particle velocity and d is the particle diameter. Note that in contrast to PCISPH [1] no special treatment is required to perform adaptive time-stepping. In line 12 of the algorithm predicted velocities $\mathbf{v}_i^*$ are determined for the particles by considering the non-pressure forces. This prediction and the precomputed factors α_i are then used by our constant density solver to compute the pressure forces which correct the density error $\rho_i^* - \rho_0$ (see Section 3.3) for each neighborhood. The time integration in line 15 determines the new positions of the particles. Hence, all neighborhoods N_i, densities ρ_i and factors α_i have to be updated. Finally, the condition $\frac{D\rho_i}{Dt} = 0$ must be satisfied. This is done in line 21 where our divergence-free solver determines pressure forces to make the velocity field divergence-free (see Section 3.2). In the last step the particle velocities are updated.

Note that the neighborhoods N_i, the densities ρ_i and the factors α_i must be computed only once per simulation step. These values are initialized before the simulation loop in lines 2-6 and updated once per time step in lines 16-20. The values are not determined in the beginning of the simulation loop as in previous methods since our solvers are executed at different points in time.

3.2 Divergence-Free Solver

The goal of our divergence-free solver is to satisfy the condition $\frac{D\rho}{Dt} = 0$ which means that the density must not change over time. From Equation (1) it follows that in this case the divergence of the velocity field $\nabla \cdot \mathbf{v}$ must be zero.

In order to enforce the condition $\frac{D\rho_i}{Dt} = 0$ for a particle i our solver determines a set of pressure forces which correct the divergence error in the neighborhood of the particle. The pressure force density of particle i is defined by

$$\mathbf{f}_i^{\text{p}} = -\nabla p_i. \tag{7}$$

The pressure gradient ∇p_i is determined by computing the spatial derivative of Equation (6) using the SPH approach (see Equation (5)):

$$\nabla p_i = \kappa_i^{\text{v}} \nabla \rho_i = \kappa_i^{\text{v}} \sum_j m_j \nabla W_{ij},$$

where κ_i^{v} is the stiffness parameter that we want to determine. In order to get symmetric pressure forces for each neighborhood we also consider the force densities $\mathbf{f}_{j \leftarrow i}^{\text{p}}$ which act from the particle i on its neighboring particles j. A set of symmetric pressure forces must satisfy the condition $\mathbf{f}_i^{\text{p}} + \sum_j \mathbf{f}_{j \leftarrow i}^{\text{p}} = \mathbf{0}$ which means that the forces sum up to zero which is required to conserve momentum. The pressure force densities $\mathbf{f}_{j \leftarrow i}^{\text{p}}$ for the neighboring particles are determined analogously to Equation (7) except that the pressure must be differentiated with respect to position $\mathbf{x}_j$:

$$\mathbf{f}_{j \leftarrow i}^{\text{p}} = -\frac{\partial p_i}{\partial \mathbf{x}_j} = \kappa_i^{\text{v}} m_j \nabla W_{ij}. \tag{8}$$

The current divergence error in particle i is determined using the SPH formulation of the divergence [4]:

$$\frac{D\rho_i}{Dt} = \sum_j m_j (\mathbf{v}_i - \mathbf{v}_j) \nabla W_{ij}. \tag{9}$$

The goal of our divergence-free solver is to compute a set of symmetric pressure forces to enforce $\frac{D\rho_i}{Dt} = 0$. These pressure forces cause the following velocity changes $\Delta \mathbf{v}_i = \Delta t \mathbf{f}_i^{\text{p}} / \rho_i$ and $\Delta \mathbf{v}_j = \Delta t \mathbf{f}_{j \leftarrow i}^{\text{p}} / \rho_i$. Since we search for the velocity changes that eliminate the divergence, we get the following equation:

$$-\frac{D\rho_i}{Dt} = \Delta t \sum_j m_j \left(\frac{\mathbf{f}_i^{\text{p}}}{\rho_i} - \frac{\mathbf{f}_{j \leftarrow i}^{\text{p}}}{\rho_i} \right) \nabla W_{ij}. \tag{10}$$

Moreover, we get an equation for the stiffness parameter κ_i^{v} by using Equations (7) and (8) in Equation (10):

$$\frac{D\rho_i}{Dt} = -\Delta t \sum_j m_j \left(\frac{\mathbf{f}_i^{\mathrm{p}}}{\rho_i} - \frac{\mathbf{f}_{j\leftarrow i}^{\mathrm{p}}}{\rho_i} \right) \nabla W_{ij}$$

$$\frac{D\rho_i}{Dt} = \frac{\Delta t}{\rho_i} \sum_j m_j \left(\kappa_i^{\mathrm{v}} \sum_j m_j \nabla W_{ij} + \kappa_i^{\mathrm{v}} m_j \nabla W_{ij} \right) \nabla W_{ij}$$

$$\frac{D\rho_i}{Dt} = \kappa_i^{\mathrm{v}} \frac{\Delta t}{\rho_i} \left(\left| \sum_j m_j \nabla W_{ij} \right|^2 + \sum_j \left| m_j \nabla W_{ij} \right|^2 \right).$$

Solving for κ_i^{v} yields:

$$\kappa_i^{\mathrm{v}} = \frac{1}{\Delta t} \frac{D\rho_i}{Dt} \cdot \underbrace{\frac{\rho_i}{\left| \sum_j m_j \nabla W_{ij} \right|^2 + \sum_j \left| m_j \nabla W_{ij} \right|^2}}_{\alpha_i}, \tag{11}$$

where α_i is a factor that solely depends on the current particle positions. If we compute pressure forces with the resulting stiffness parameter κ_i^{v}, the condition $\frac{D\rho_i}{Dt} = 0$ will be exactly fulfilled. This means that we obtain a divergence-free velocity field in the neighborhood of particle i. In order to obtain a globally divergence-free velocity field we determine the pressure forces for all particles iteratively with a parallel Jacobi solver.

If particle i has a small number of neighbors, the denominator of α_i can lead to instabilities. In our simulations we clamp the denominator if it gets smaller than 10^{-6} in order to solve this problem. We did not notice any visual artifacts in our experiments. Alternatively, the problem could be solved by adding a small constant to the denominator [14] or using a reference configuration with a filled neighborhood [1].

The total force $\mathbf{f}_{i,\mathrm{total}}^{\mathrm{p}}$ for a particle i is the sum of all pressure forces:

$$\begin{aligned}
\mathbf{f}_{i,\mathrm{total}}^{\mathrm{p}} &= \frac{m_i}{\rho_i} \mathbf{f}_i^{\mathrm{p}} + \sum_j \frac{m_j}{\rho_j} \mathbf{f}_{i\leftarrow j}^{\mathrm{p}} \\
&= -\frac{m_i}{\rho_i} \kappa_i^{\mathrm{v}} \sum_j m_j \nabla W_{ij} + \sum_j \frac{m_j}{\rho_j} \kappa_j^{\mathrm{v}} m_i \nabla W_{ji} \\
&= -m_i \sum_j m_j \left(\frac{\kappa_i^{\mathrm{v}}}{\rho_i} + \frac{\kappa_j^{\mathrm{v}}}{\rho_j} \right) \nabla W_{ij}.
\end{aligned}$$

Note that our symmetric pressure force is equivalent to the one introduced by Monaghan [6].

In order to get a velocity field that is globally divergence-free we compute pressure forces for each particle using parallel Jacobi iterations. In each iteration we have to compute the stiffness parameters κ_i^{v} that depend on the complex factors α_i. Fortunately, the factors can be precomputed before the iterative process since they only depend on the current positions which yields computationally cheap iteration steps. Note that Equation (11) must be adapted for static boundary particles since in this case $\mathbf{f}_{j\leftarrow i}^{\mathrm{p}} = 0$.

Our divergence-free solver is outlined in Algorithm 2. It finishes when the average density change rate is smaller than a given threshold η^{div}. To improve the convergence of our solver we use a "warm start" operating as explained in the following. The stiffness values κ_i^{v} are summed up for each particle during the iterative process. Before the divergence-free solver in the next simulation step starts we evaluate line 7 once for each particle using the resulting values.

Algorithm 2. Divergence-Free Solver

1: **function** CORRECTDIVERGENCEERROR(α, $\mathbf{v}^*$)
2: **while** $\left(\left(\frac{D\rho}{Dt} \right)_{\mathrm{avg}} > \eta^{\mathrm{div}} \right) \vee (\mathrm{iter} < 1)$ **do**
3: **for all** particles i **do** // compute $\frac{D\rho}{Dt}$
4: $\frac{D\rho_i}{Dt} = -\rho_i \nabla \cdot \mathbf{v}_i^*$
5: **for all** particles i **do** // adapt velocities
6: $\kappa_i^{\mathrm{v}} = \frac{1}{\Delta t} \frac{D\rho_i}{Dt} \alpha_i$, $\kappa_j^{\mathrm{v}} = \frac{1}{\Delta t} \frac{D\rho_j}{Dt} \alpha_j$
7: $\mathbf{v}_i^* := \mathbf{v}_i^* - \Delta t \sum_j m_j \left(\frac{\kappa_i^{\mathrm{v}}}{\rho_i} + \frac{\kappa_j^{\mathrm{v}}}{\rho_j} \right) \nabla W_{ij}$

3.3 Constant Density Solver

Our constant density solver minimizes the deviation $\rho - \rho_0$ of the actual density to the rest density. Previous pressure solvers like PCISPH [1] or IISPH [2] also minimize the density deviation and could be combined with our divergence-free solver. However, we decided to develop a new solver which works analogously to our divergence-free solver. This has the advantage that the precomputed factors α_i can be reused which reduces the computational effort significantly. Our solver uses a predictor-corrector scheme similar to PCISPH in order to determine particle positions for the next time step that correct the density error. However, in contrast to PCISPH we do not precompute a reference configuration with a filled neighborhood to solve the system since this reduces the convergence.

The prediction of the density error $\rho_i^* - \rho_0$ is determined by integrating Equation (9) using an explicit Euler scheme:

$$\rho_i^* = \rho_i + \Delta t \frac{D\rho_i}{Dt} = \rho_i + \Delta t \sum_j m_j (\mathbf{v}_i^* - \mathbf{v}_j^*) \nabla W_{ij}.$$

Analogous to Equation (10), we compute forces to correct this error by solving:

$$\rho_i^* - \rho_0 = \Delta t^2 \sum_j m_j \left(\frac{\mathbf{f}_i^{\mathrm{p}}}{\rho_i} - \frac{\mathbf{f}_{j\leftarrow i}^{\mathrm{p}}}{\rho_i} \right) \nabla W_{ij}. \tag{12}$$

The resulting stiffness parameter is determined by:

$$\kappa_i = \frac{1}{\Delta t^2} (\rho_i^* - \rho_0) \alpha_i.$$

Algorithm 3 outlines our implicit constant density solver. Note that we perform a warm start analogous to the one of our divergence-free solver to improve the convergence.

Algorithm 3. Constant Density Solver

1: **function** CORRECTDENSITYERROR(α, $\mathbf{v}^*$)
2: **while** $(\rho_{\mathrm{avg}} - \rho_0 > \eta^{\rho}) \vee (\mathrm{iter} < 2)$ **do**
3: **for all** particles i **do** // predict density
4: compute ρ_i^*
5: **for all** particles i **do** // adapt velocities
6: $\kappa_i = \frac{\rho_i^* - \rho_0}{\Delta t^2} \alpha_i$, $\kappa_j = \frac{\rho_j^* - \rho_0}{\Delta t^2} \alpha_j$
7: $\mathbf{v}_i^* := \mathbf{v}_i^* - \Delta t \sum_j m_j \left(\frac{\kappa_i}{\rho_i} + \frac{\kappa_j}{\rho_j} \right) \nabla W_{ij}$

4 Extensions

4.1 Viscosity Solver

In order to allow the simulation of highly viscous fluids we introduce an implicit viscosity solver. Viscosity is defined as the resistance of a fluid to flow. That means that viscosity forces counteract viscous stress. In a Newtonian fluid the viscous stress tensor $\boldsymbol{\tau}$ is defined by the strain rate tensor $\dot{\boldsymbol{\epsilon}}$ and the viscosity coefficient $\boldsymbol{\mu}$:

$$\boldsymbol{\tau} = \boldsymbol{\mu}\dot{\boldsymbol{\epsilon}}.$$

The strain rate tensor is defined as the symmetric part of the velocity gradient $\nabla\mathbf{v}$ [3]:

$$\dot{\boldsymbol{\epsilon}} = \frac{1}{2}\left(\nabla\mathbf{v} + \nabla\mathbf{v}^T\right). \tag{13}$$

In general the viscosity coefficient $\boldsymbol{\mu}$ is a rank four tensor containing 81 coefficients. However, since $\boldsymbol{\tau}$ and $\dot{\boldsymbol{\epsilon}}$ are symmetric, $\boldsymbol{\mu}$ can be reduced to 36 independent values. For a specific material the number of independent coefficients can even be significantly smaller, e.g., isotropic material requires only two coefficients. Please note that viscosity models based on the strain rate commonly assume a divergence-free velocity field as noted in several previous works, e.g., [31], [41], [53]. Thanks to our divergence-free solver presented in Section 3.2, the velocity field is approximately divergence-free, even after the constant density solve.

The key idea of our implicit viscosity solver is analogous to the idea of our divergence-free solver (see Section 3.2) and our constant density solver (see Section 3.3): For each neighborhood i we determine an individual viscosity coefficient $\boldsymbol{\mu}_i'$ and corresponding impulses in order to enforce a target strain rate locally. We use a parallel Jacobi method in order to fulfill the strain rate condition globally. In this work we introduce a viscosity parameter $0 \leq \gamma \leq 1$ to define the target strain rate:

$$\dot{\boldsymbol{\epsilon}}^{\mathrm{t}} = \gamma\dot{\boldsymbol{\epsilon}}^{*},$$

where $\dot{\boldsymbol{\epsilon}}^{*}$ denotes the strain rate tensor determined by the velocities $\mathbf{v}^{*}$ (see Algorithm 5). For $\gamma = 0$ the target strain rate tensor vanishes which corresponds to a fluid with maximum viscosity while a minimum viscosity is obtained for $\gamma = 1$.

Using the SPH formulation for a symmetric gradient [6] the velocity gradient is determined by

$$\nabla\mathbf{v}_i = \frac{1}{\rho_i}\sum_j m_j\mathbf{v}_{ji}\nabla W_{ij}^T, \tag{14}$$

where $\mathbf{v}_{ji} = \mathbf{v}_j - \mathbf{v}_i$. Substituting Equation (14) in Equation (13) and writing the symmetric tensor $\dot{\boldsymbol{\epsilon}}_i$ for particle i as six-dimensional vector yields:

$$\begin{pmatrix} \dot{\boldsymbol{\epsilon}}_i^{xx} \\ \dot{\boldsymbol{\epsilon}}_i^{yy} \\ \dot{\boldsymbol{\epsilon}}_i^{zz} \\ \dot{\boldsymbol{\epsilon}}_i^{xy} \\ \dot{\boldsymbol{\epsilon}}_i^{xz} \\ \dot{\boldsymbol{\epsilon}}_i^{yz} \end{pmatrix} = \frac{1}{2\rho_i}\sum_j m_j \begin{pmatrix} 2\mathbf{v}_{ji}^x\nabla W_{ij}^x \\ 2\mathbf{v}_{ji}^y\nabla W_{ij}^y \\ 2\mathbf{v}_{ji}^z\nabla W_{ij}^z \\ \mathbf{v}_{ji}^x\nabla W_{ij}^y + \mathbf{v}_{ji}^y\nabla W_{ij}^x \\ \mathbf{v}_{ji}^x\nabla W_{ij}^z + \mathbf{v}_{ji}^z\nabla W_{ij}^x \\ \mathbf{v}_{ji}^y\nabla W_{ij}^z + \mathbf{v}_{ji}^z\nabla W_{ij}^y \end{pmatrix}.$$

In our work we use the formulation of the strain rate tensor as six-dimensional vector in order to remove redundancy and to facilitate the computation of its derivative (see below).

For each neighborhood we want to compute a viscosity coefficient $\boldsymbol{\mu}_i'$ so that the current strain rate error $\dot{\boldsymbol{\epsilon}}_i - \dot{\boldsymbol{\epsilon}}_i^{\mathrm{t}}$ is corrected. Since the strain rate tensor has six independent values, we determine a six-dimensional viscosity coefficient in the solver. Analogous to Equations (7) and (8) we compute the following impulse densities in order to correct the strain rate error:

$$\mathbf{p}_i = -\frac{\partial\boldsymbol{\tau}_i}{\partial\mathbf{v}_i} = -\left(\frac{\partial\dot{\boldsymbol{\epsilon}}_i}{\partial\mathbf{v}_i}\right)^T\boldsymbol{\mu}_i',$$
$$\mathbf{p}_{j\leftarrow i} = -\frac{\partial\boldsymbol{\tau}_i}{\partial\mathbf{v}_j} = -\left(\frac{\partial\dot{\boldsymbol{\epsilon}}_i}{\partial\mathbf{v}_j}\right)^T\boldsymbol{\mu}_i'.$$

Due to our six-dimensional formulation the required derivatives of the strain rate $\dot{\boldsymbol{\epsilon}}_i$ with respect to the velocities $\mathbf{v}_i$ and $\mathbf{v}_j$ are determined by the following 6×3 matrices:

$$\frac{\partial\dot{\boldsymbol{\epsilon}}_i}{\partial\mathbf{v}_j} = \frac{1}{2\rho_i}m_j\begin{pmatrix} 2\nabla W_{ij}^x & 0 & 0 \\ 0 & 2\nabla W_{ij}^y & 0 \\ 0 & 0 & 2\nabla W_{ij}^z \\ \nabla W_{ij}^y & \nabla W_{ij}^x & 0 \\ \nabla W_{ij}^z & 0 & \nabla W_{ij}^x \\ 0 & \nabla W_{ij}^z & \nabla W_{ij}^y \end{pmatrix},$$
$$\frac{\partial\dot{\boldsymbol{\epsilon}}_i}{\partial\mathbf{v}_i} = -\sum_j\frac{\partial\dot{\boldsymbol{\epsilon}}_i}{\partial\mathbf{v}_j}.$$

The impulses cause the velocity changes $\Delta\mathbf{v}_i = \frac{1}{\rho_i}\mathbf{p}_i$ and $\Delta\mathbf{v}_j = \frac{1}{\rho_i}\mathbf{p}_{j\leftarrow i}$. The resulting change of the strain rate error is determined by $\Delta\dot{\boldsymbol{\epsilon}} - \dot{\boldsymbol{\epsilon}}^{\mathrm{t}}$, where $\Delta\dot{\boldsymbol{\epsilon}}$ is computed by Equation (13) using the velocity changes $\Delta\mathbf{v}$. Our goal is to compute $\Delta\dot{\boldsymbol{\epsilon}}$ so that the strain rate error is eliminated. This yields the following equation for $\boldsymbol{\mu}_i'$:

$$\mathbf{A} = \frac{1}{\rho_i}\frac{\partial\dot{\boldsymbol{\epsilon}}_i}{\partial\mathbf{v}_i}\left(\frac{\partial\dot{\boldsymbol{\epsilon}}_i}{\partial\mathbf{v}_i}\right)^T + \sum_j\frac{1}{\rho_i}\frac{\partial\dot{\boldsymbol{\epsilon}}_i}{\partial\mathbf{v}_j}\left(\frac{\partial\dot{\boldsymbol{\epsilon}}_i}{\partial\mathbf{v}_j}\right)^T,$$
$$\boldsymbol{\mu}_i' = \underbrace{\mathbf{A}^{-1}}_{\boldsymbol{\beta}_i}\left(\dot{\boldsymbol{\epsilon}}_i - \dot{\boldsymbol{\epsilon}}_i^{\mathrm{t}}\right),$$

which is determined analogously to Equation (11). Note that the matrix $\boldsymbol{\beta}_i$ only depends on the current positions. Hence, it can be precomputed and must not be updated in the Jacobi solver.

Finally, we compute the sum of all impulses for each particle i including the impulses acting from neighboring particles j as

$$\begin{aligned}\mathbf{p}_{i,\mathrm{total}} &= \frac{m_i}{\rho_i}\mathbf{p}_i + \sum_j\frac{m_j}{\rho_j}\mathbf{p}_{i\leftarrow j} \\ &= \frac{m_i}{2}\sum_j m_j\begin{pmatrix} 2\nabla W_{ij}^x & 0 & 0 \\ 0 & 2\nabla W_{ij}^y & 0 \\ 0 & 0 & 2\nabla W_{ij}^z \\ \nabla W_{ij}^y & \nabla W_{ij}^x & 0 \\ \nabla W_{ij}^z & 0 & \nabla W_{ij}^x \\ 0 & \nabla W_{ij}^z & \nabla W_{ij}^y \end{pmatrix}^T\left(\frac{1}{\rho_i^2}\boldsymbol{\mu}_i' + \frac{1}{\rho_j^2}\boldsymbol{\mu}_j'\right)\end{aligned} \tag{15}$$

and determine the resulting velocity changes.

Our implicit viscosity solver is outlined in Algorithm 4. Analogous to the divergence-free solver and the constant density solver we perform a warm start to improve its convergence. To integrate the viscosity solver in the simulation step we have to modify Algorithm 1. By adding the viscosity solve before the position update we get the modified Algorithm 5.

Algorithm 4. Implicit Viscosity Solver

function CORRECTSTRAINRATEERROR($\mathbf{v}^*$)
  for all particles i **do**
    compute factors $\boldsymbol{\beta}_i$
    compute target strain rates $\dot{\boldsymbol{\epsilon}}_i^{\mathrm{t}}$
  while $((\dot{\boldsymbol{\epsilon}}^* - \dot{\boldsymbol{\epsilon}}^{\mathrm{t}})_{\mathrm{avg}} > \eta^{\mathrm{visco}}) \vee (\mathrm{iter} < 1)$ **do**
    for all particles i **do** // compute strain rate
      compute $\dot{\boldsymbol{\epsilon}}_i^*$
    for all particles i **do** // adapt velocities
      $\boldsymbol{\mu}_i' = \boldsymbol{\beta}_i(\dot{\boldsymbol{\epsilon}}_i^* - \dot{\boldsymbol{\epsilon}}_i^{\mathrm{t}})$, $\boldsymbol{\mu}_j' = \boldsymbol{\beta}_j\left(\dot{\boldsymbol{\epsilon}}_j^* - \dot{\boldsymbol{\epsilon}}_j^{\mathrm{t}}\right)$
      $\mathbf{v}_i^* := \mathbf{v}_i^* - \frac{1}{m_i}\mathbf{p}_{i,\mathrm{total}}$ // see Equation (15)

Algorithm 5. Simulation with Implicit Viscosity

function PERFORMSIMULATION
  init neighborhoods $N(0)$ // initialization
  compute densities $\rho(0)$
  compute factors $\alpha(0)$
  while $(t < t_{\max})$ **do** // simulation loop
    compute non-pressure forces $\mathbf{F}^{\mathrm{adv}}(t)$
    predict velocities $\mathbf{v}^* = \mathbf{v} + \Delta t\mathbf{M}^{-1}\mathbf{F}^{\mathrm{adv}}$
    correctDensityError(α, $\mathbf{v}^*$) // $\rho^* - \rho_0 = 0$
    correctStrainRateError($\mathbf{v}^*$) // $\dot{\boldsymbol{\epsilon}}^* - \dot{\boldsymbol{\epsilon}}^{\mathrm{t}} = 0$
    update positions $\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \Delta t\mathbf{v}^*$
    update neighborhoods $N(t + \Delta t)$
    update densities $\rho(t + \Delta t)$
    update factors $\alpha(t + \Delta t)$
    correctDivergenceError(α, $\mathbf{v}^*$) // $\frac{D\rho}{Dt} = 0$
    update velocities $\mathbf{v}(t + \Delta t) = \mathbf{v}^*$

Note that Peer et al. [41] perform a decomposition of the strain rate tensor into the expansion rate tensor $\mathbf{V} = \frac{1}{3}(\nabla \cdot \mathbf{v})\mathbf{I}$ and the shear rate tensor $\mathbf{S} = \dot{\boldsymbol{\epsilon}} - \mathbf{V}$ in order to preserve the divergence of the fluid particles. This decomposition can also be used in our viscosity solver. However, due to our divergence-free solver the divergence $\nabla \cdot \mathbf{v}$ is in practice small, even after the pressure solve. Therefore, we did not use the decomposition in this work in order to increase the performance.

We also want to note that the viscosity parameter γ depends on the time step size. Our implicit viscosity solver has this limitation in common with recent approaches, e.g., the one of Peer et al. [41]. We do not further address this issue in the paper but it is one of our research topics for future work.

4.2 Efficient Kernel Computation

SPH methods typically use kernel functions which approximate a Gaussian. Sometimes different kernels are used to determine W_{ij} and its gradient ∇W_{ij} (e.g., [8]). However, in a predictor-corrector method the same kernel must be used for both. Otherwise, the prediction and correction steps do not fit together. In our solvers we use the cubic spline kernel [6].

The SPH approach uses a kernel function with compact support. This means the function vanishes at a finite distance which is also known as the support radius h. Such a kernel function can be written as $W_h(q(\mathbf{x}))$ with $q = \frac{\|\mathbf{x}\|}{h}$, where the function is non-zero for $0 \leq q < 1$. Therefore, the kernel gradient $\nabla W_h(q(\mathbf{x}))$ has the same compact support.

Since the kernel gradients must be determined for the whole neighborhood of each particle in the non-pressure force computation, in each iteration step of all solvers and when computing the factors α_i and β_i, it is one of the most time consuming tasks in our simulation. However, due to the large memory consumption it is not recommended to store all gradients for large scenes. In the following we introduce an efficient computation of the kernel and its gradient based on precomputed lookup tables to increase the performance of our simulation. Lookup tables have already been employed in other areas to speed up function evaluations but to the best of our knowledge they have not been used yet in SPH simulations.

It is easy to generate a lookup table for the kernel W_h by a regular sampling since it is a smooth scalar function with compact support. However, the gradient ∇W_h is a vector function and sampling this function in all three dimensions yields a larger memory consumption and computational overhead. Therefore, we introduce the scalar function $g(q)$:

$$\nabla W_h(q(\mathbf{x})) = \mathbf{x} \cdot g(q) \quad \text{with} \quad g(q) = \frac{\partial W_h}{\partial q} \cdot \frac{1}{h\|x\|}.$$

This function is also a smooth scalar function with compact support which can be sampled regularly to get a corresponding lookup table. Finally, only a single lookup and a multiplication with $\mathbf{x}$ is required to compute a kernel gradient.

Our kernel lookup tables are easy to implement and can also be used in other SPH methods. Details about the sampling distance, the performance gain and the approximation error are discussed in Section 5.

5 Results

In this section we show results for our divergence-free SPH method and for the introduced extensions. All timings were measured on two Intel Xeon E5-2697 processors with 2.7 GHz, 12 cores per processor and 64 GB RAM. We used OpenMP to parallelize our fluid simulation. In our experiments we performed the boundary handling using the rigid-fluid coupling of Akinci et al. [12] and the neighborhood search using the parallel method of Ihmsen et al. [52]. We successfully combined our method with the surface tension models of Becker and Teschner [10], Akinci et al. [13] and He et al. [54]. However, in the experiments in this paper we solely used the surface tension model of Akinci et al. [13]. We employed the XSPH variant of Schechter and Bridson [37] to simulate the viscosity of all low viscous fluids. In Section 4.1 we introduced an extension of our divergence-free SPH approach for the simulation of highly viscous fluids. The results of our implicit viscosity solver are discussed in the last paragraph of this section.

TABLE 1
Performance Comparison of DFSPH, IISPH, PBF and PCISPH for a Breaking Dam Simulation with 125k Particles Using Different Fixed Time Step Sizes (See Fig. 2).

	DFSPH			IISPH			PBF			PCISPH		
Δt [ms]	iter. (cd/df)	solver [s]	total [s]	iter.	solver [s]	total [s]	iter.	solver [s]	total [s]	iter.	solver [s]	total [s]
4.0	4.5/2.8	45.2	**51.3**	50.5	312.1	318.1	105.7	607.1	613.5	160.0	1,079.1	1,085.7
2.0	2.1/1.3	47.8	59.4	21.4	256.4	267.9	42.7	508.4	520.7	73.9	1,021.2	1,033.5
1.0	2.0/1.0	85.0	107.5	7.3	197.9	**220.7**	13.2	314.8	338.0	23.9	656.9	680.0
0.5	2.0/1.0	164.1	210.8	2.3	182.3	225.6	3.4	194.1	**240.5**	6.7	394.9	**440.9**
0.25	2.0/1.0	288.3	372.0	2.0	322.5	402.2	2.0	263.3	354.0	3.0	409.3	498.0

The table contains the average iteration count, the total computation time of the solvers and the total simulation time including the neighborhood search in a simulation over one second. The best total computation times are marked bold. The column with the average iteration count of DFSPH contains the values for the constant density solver (cd) and the divergence-free solver (df). Note that the solver time of DFSPH is the sum of the times needed by both solvers since they need almost the same amount of time for one iteration step.

A well-known issue of SPH fluid simulations is particle deficiency at free surfaces which leads to an underestimation of the density and therefore to particle clustering. We clamp negative pressures to zero to solve this problem which is a common solution in computer graphics, see e.g., [2]. In our simulations we enforced an average density error of less than 0.01 percent and a density error due to the density change rate of less than 0.1 percent. Moreover, we used adaptive time-stepping according to the CFL condition (see Section 3.1) unless stated otherwise.

5.1 Performance

We compared the performance of our novel simulation method for incompressible fluids with IISPH, PBF and PCISPH using a breaking dam scenario with 125k particles with different fixed time step sizes. The particle radius in the simulation was 0.02 m. The results for a simulation over one second are summarized in Tables 1 and 2 shows the speedup factors. Note that Ihmsen et al. [2] compared the performance between PCISPH and IISPH with a similar scenario and measured comparable results.

Our SPH method enforces the constant density condition and the divergence-free condition at the same time. Therefore, the density error is kept small during the whole simulation which leads to significantly lower number of iterations required by the solvers. The iteration count is further reduced by our warm start which initializes both solvers with the sum of the stiffness values of the previous time step. In the dam break scenario we measured a speedup factor of approximately 3 due to the warm start. DFSPH uses a full warm start. In contrast to that IISPH performs best when multiplying the solution of the previous time step with a factor of 0.5 [2]. As summarized in Table 2 the performed warm start and the divergence-free velocity field in our simulation leads to speedup factors of 6.9 in comparison to IISPH up to 23.9 in comparison to PCISPH for a time step size of 4 ms. There are two reasons for this large speedup. First, in contrast to previous works we enforce both conditions which increases the stability of the simulation and therefore allows us to perform larger time steps and to apply a full warm start. Second, in our method the reuse of precomputed coefficients leads to fast iterations. Our constant density solver required only an average of 4.5 iterations while our divergence-free solver needed 2.8 iterations. The second best approach in our experiment was IISPH which already required 50.5 iterations. For smaller time step sizes DFSPH often used the minimum number of iterations which reduced the speedup for these step sizes. However, in simulations with more particles, where more iterations are required, the speedup is also larger for small step sizes. In our simulations we noticed that DFSPH performs best when a time step size is chosen so that the number of iterations ranges between 2 and 20.

TABLE 2
Speedup Factors of DFSPH in Comparison to IISPH, PBF and PCISPH Based on the Measurements in Table 1

	IISPH		PBF		PCISPH	
Δt [ms]	solver	total	solver	total	solver	total
4.0	6.9	6.2	13.4	12.0	23.9	21.2
2.0	5.3	4.5	10.6	8.8	21.4	17.4
1.0	2.3	2.1	3.7	3.1	7.7	6.3
0.5	1.1	1.1	1.2	1.1	2.4	2.1
0.25	1.1	1.1	0.9	1.0	1.4	1.3

The best performance of our method was reached for larger time steps than IISPH, PBF and PCISPH (see Table 1). The usage of larger time steps has the advantage that the computationally expensive neighborhood search is needed less frequently.

In Section 4.2 we introduced a kernel optimization based on lookup tables. This optimization yields a performance gain of approximately 30 percent in the breaking dam scenario. In this performance test the kernel function and its gradient were sampled at 1,000 points. For this sampling we measured a maximum local error of less than $10^{-11}\,\mathrm{m}^{-3}$. The implementation of the kernel optimization is very simple and the error is negligible. Hence, this is a useful extension for SPH simulations.

In order to demonstrate the performance of our method in more complex scenarios, we performed another breaking dam simulation with a large number of fluid particles and a boundary consisting of two dragon models and a moving wall (see Fig. 3). We used 2.3 million fluid particles and 0.7 million boundary particles in this breaking dam scenario. As a second example we simulated 5 million fluid particles flowing through a canyon consisting of 40 million boundary particles (see Fig. 1, right). Table 3 shows the average computation times of the most important steps in Algorithm 1 for both scenarios.

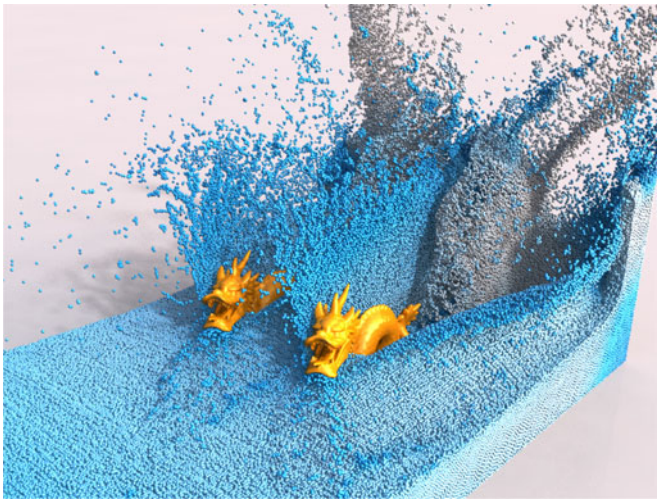


Fig. 3. Breaking dam model with 2.3 million fluid particles and a complex boundary including two dragon models and a moving wall. We color coded the velocity field: white is the maximum and blue is the minimum.

5.2 Memory Requirements

The memory requirements of our method are low. DFSPH only needs to store the scalar value α_i per particle which is then used by both solvers. We need an additional scalar value per particle for each solver if a warm start is performed. Therefore, DFSPH requires considerably less memory than IISPH which needs seven scalar values for the solver and one for the warm start. This is especially an advantage when simulating large scale scenarios with millions of particles.

5.3 Stability

In this paragraph we demonstrate that current state-of-the-art SPH pressure solvers which do not enforce a divergence-free velocity field explicitly cannot satisfy the condition $\frac{D\rho}{Dt} = 0$ as demanded by the continuity equation. However, this can cause instabilities, especially when fluid particles have high velocities.

We compared the velocity divergence error of DFSPH and IISPH using a breaking dam scenario with 125k particles (see Fig. 2). The results demonstrate that DFSPH is able to maintain a divergence-free velocity field in contrast to approaches which do not enforce a divergence-free condition explicitly. In the comparison the maximum local divergence error of IISPH was $108.3\,\mathrm{s}^{-1}$ while the one of our method was only $1.9\,\mathrm{s}^{-1}$.

Large divergence errors can lead to instabilities as we demonstrate in the following comparisons with PCISPH and IISPH. First, we simulated a cube of 27k fast moving particles falling on the ground in order to show how this influences the stability (see Fig. 5). We used the adaptive time-stepping according to the CFL condition. In this experiment we compared PCISPH, DFSPH with deactivated divergence-free solver and DFSPH with activated divergence-free solver. A stable simulation was only possible with DFSPH when using both solvers. Without the divergence-free solver artifacts occurred in the DFSPH simulation. PCISPH even became totally instable due to the impact and several fluid particles passed through the boundary. A stable simulation with PCISPH was only possible when using a time step size which was clearly below the one suggested by the CFL condition which reduced the overall performance significantly.

TABLE 3
Performance of the Canyon Simulation (See Fig. 1, Right) and the Dam Break Simulation (See Fig. 3)

	neigh. search	α	$\mathbf{F}^{\mathrm{adv}}$	const. density	div.-free	total
canyon	1.2	0.2	0.7	1.9	1.3	5.3
dragons	0.4	0.1	0.3	0.7	0.6	2.1

The table shows the average times per simulation step (in seconds) required by the neighborhood search, the computation of α, the computation of all non-pressure forces $\mathbf{F}^{\mathrm{adv}}$, the constant density solver and the divergence-free solver.

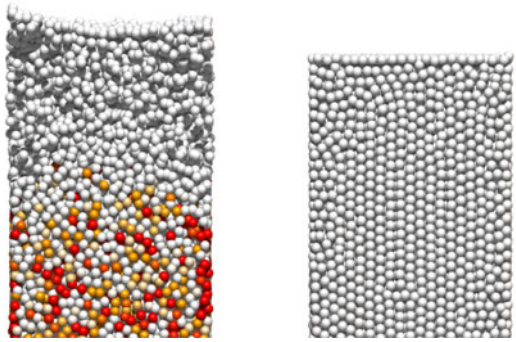


Fig. 4. Top of a fluid pillar with 80k particles. The divergence errors are color coded: red is the maximum and white is the minimum. Large divergence errors in the IISPH simulation (left) lead to jumping artifacts. The DFSPH approach (right) maintains a divergence-free velocity field which allows a stable simulation without artifacts.

To compare our method with IISPH we simulated a resting fluid pillar model with 80k fluid particles (see Fig. 4). We restricted the time step size to a maximum of 5 ms as the CFL condition allows arbitrary large values for the resting particles. In the DFSPH simulation the maximum local divergence error was $2.5\,\mathrm{s}^{-1}$ which allowed a stable simulation without artifacts. In the IISPH simulation large divergence errors of up to $72.9\,\mathrm{s}^{-1}$ occurred. In combination with the large time steps this led to visual artifacts: fluid particles jumped up due to the errors (see Fig. 4, left). By decreasing

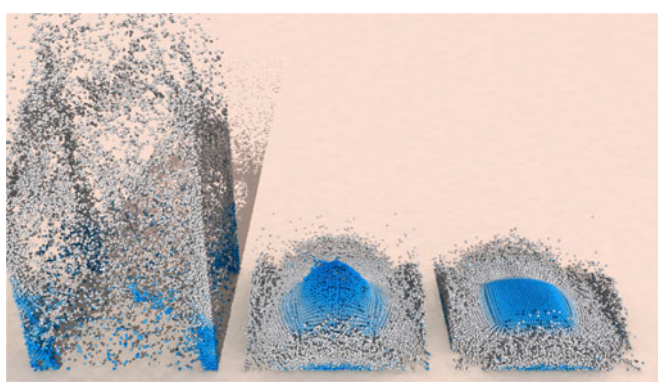


Fig. 5. Stability comparison of DFSPH with PCISPH. The same color coding as in Fig. 3 is used. A cube with 27k fluid particles is falling on the ground with a high velocity. Instabilities occur in the PCISPH simulation (left) and several fluid particles pass through the boundary. DFSPH without activating the divergence-free solver shows artifacts (middle) while DFSPH with activated divergence-free solver stays stable.

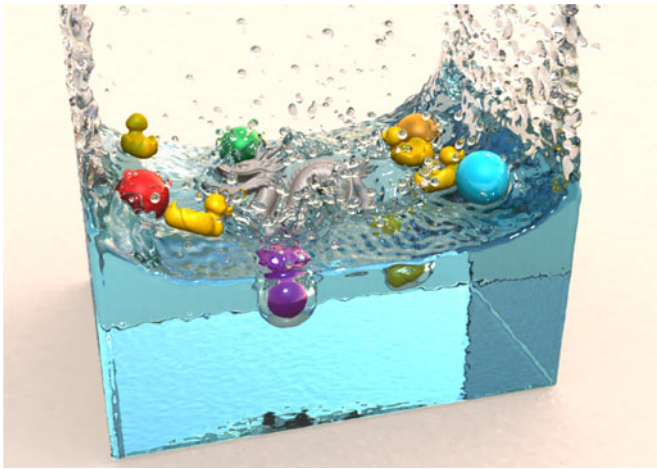


Fig. 6. Two-way coupling of several dynamic rigid bodies with 330k fluid particles.

the time step size significantly this problem can be solved. However, this reduces the overall performance considerably.

In summary our comparisons showed that enforcing a divergence-free velocity field improves the stability of the simulation and allows to perform larger time steps.

The next two experiments demonstrate the stability of our method in simulations with dynamic boundaries and in large scale scenarios. In the first simulation several rigid bodies fall in a breaking dam scenario with 330k particles (see Fig. 6). We used the Bullet physics library [55] to simulate the rigid bodies. In the second experiment 2.4 million fast moving particles hit a complex boundary (see Fig. 1, left). In both scenarios DFSPH allowed a stable simulation with the maximum possible time step size defined by the CFL condition.

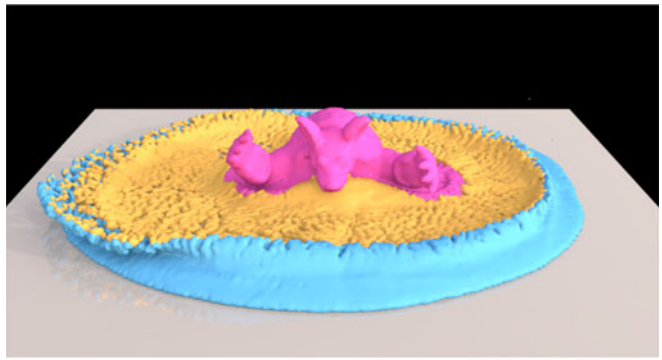
5.4 Viscous Fluids

In order to simulate highly viscous fluids we introduced an implicit viscosity solver in Section 4.1. In the following we present results with our method and a comparison with the approach of Peer et al. [41]. In all experiments with viscous fluids we used a fixed time step size of $\Delta t = 1$ ms, a particle radius of 0.025 m and no surface tension forces. In all simulations we enforced an average strain rate error of less than $0.01\,\mathrm{s}^{-1}$.

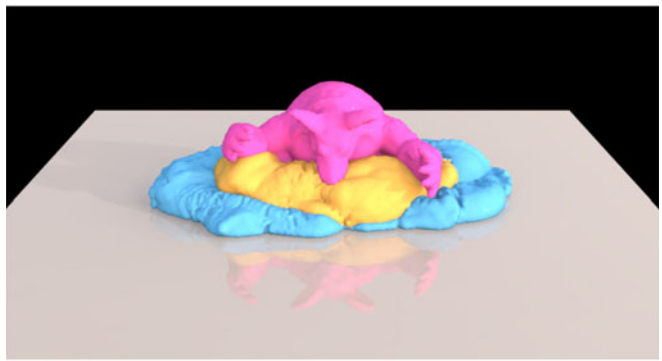
In our first experiment with the implicit viscosity solver we dropped a dragon, a bunny and an armadillo model on a plane while varying the viscosity parameter γ. Fig. 7 shows the results for the values $\gamma = 0.95$, $\gamma = 0.7$ and $\gamma = 0.4$ after 2 seconds of simulation. This experiment demonstrates that our implicit solver can handle low viscous fluids as well as highly viscous fluids. The simulation model contains 390k fluid particles and 91k boundary particles. We measured the average computation times of the viscosity solver for all three viscosity parameters (see Table 4). As we can see the solver converges faster for low viscous fluids. The reason for this is that in case of low viscosity, the difference between the target strain rate and the current strain rate is always small.

In the next experiment we demonstrate that our viscosity solver is able to simulate the phenomena of viscous fluid buckling and coiling (see Fig. 8). In the first simulation we emitted a sheet of 150k fluid particles using a viscosity parameter of $\gamma = 0.25$ (see Fig. 8a) and in the second one we emitted 37k fluid particles using a viscosity parameter of $\gamma = 0.75$ (see Fig. 8b). The performance values of these simulations are given in Table 4.

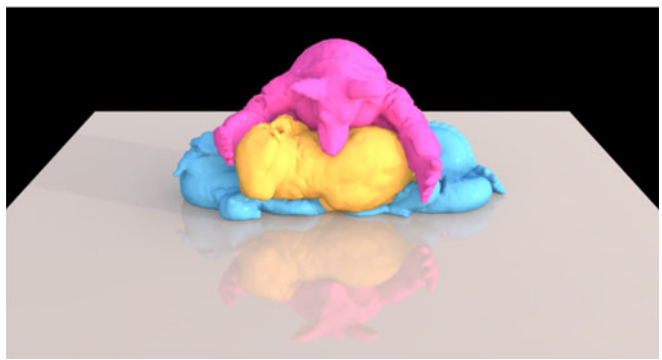
Finally, we performed a comparison with the implicit viscosity approach of Peer et al. [41]. In this comparison we dropped a bunny model on the ground using different viscosity parameters γ (see Fig. 9). For the method of Peer et al. we used the value of $\xi = \gamma$ to determine the target velocity gradient. Note that an exact comparison of both methods is hard since viscosity parameters with a different meaning are used and the conditions to terminate the iterative solvers are different. However, in a visual comparison we can see that our approach covers a broader range of



(a) $\gamma = 0.95$



(b) $\gamma = 0.7$



(c) $\gamma = 0.4$

Fig. 7. Stanford model. A Stanford dragon, bunny and armadillo are dropped on a plane. The simulation was performed with different viscosity parameters γ. The images show the results after two seconds of simulation.

TABLE 4
Number of Fluid Particles, Viscosity Parameters and Average Computation Times (in Seconds) per Simulation Step of the Pressure Solver and the Viscosity Solver for the Viscous Fluid Examples Shown in Figs. 7, 8 and 9

model	fluid particles	γ	pressure	viscosity
Stanford	390k	0.95	0.16	2.4
		0.7	0.22	4.7
		0.4	0.26	5.3
coiling	37k	0.75	0.01	0.32
buckling	150k	0.25	0.01	0.79
		0.9	0.02	0.13
		0.75	0.03	0.20
bunny	50k	0.5	0.03	0.30
		0.25	0.03	0.37

The time for the pressure solve contains the time for the computation of α and the times for the constant density and the divergence-free solver.

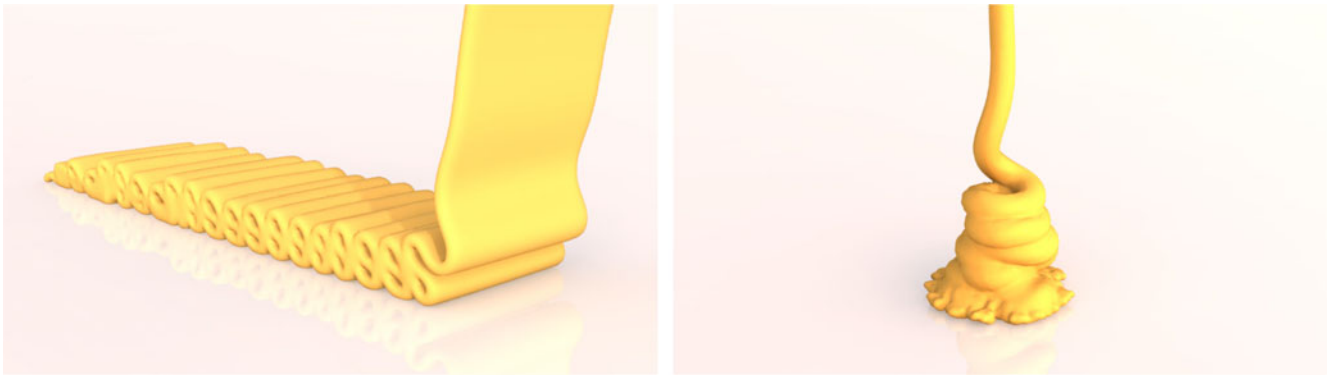
viscous materials. While Peer et al. state that their approach should not be used for low viscous fluids our method is able to handle low viscosity as well.

Peer et al. reconstruct the final velocities from the target velocity gradient. A first-order Taylor approximation is used in the reconstruction. This approximation has the advantage that a smaller linear system has to be solved in comparison to our method. However, the drawback of the approximation of Peer et al. is that it introduces a large numerical damping. For that reason the method of Peer et al. cannot simulate low viscous fluids. Our approach does not use any approximation and therefore does not suffer from numerical damping (see Fig. 9).

6 Conclusion and Future Work

In this paper we introduced a novel implicit SPH approach to simulate incompressible and viscous fluids. In contrast to previous state-of-the-art SPH methods for incompressible fuids two solvers are employed in order to enforce a constant density condition and a divergence-free condition. The combination of both solvers improves the stability and the performance of the simulation. Our method handles scenarios with millions of fast moving particles robustly and is considerably faster than current state-of-the-art methods. Moreover, we presented an implicit viscosity solver to simulate highly viscous fluids which can be integrated easily in our SPH method.

Since DFSPH is based on the same principles as other SPH pressure solvers, it has similar limitations. The density near free surfaces is underestimated due to particle deficiency. This leads to unnatural particle clustering artifacts. We solve this problem by clamping negative pressures to zero which is a common solution in computer graphics. However, Schechter and Bridson [37] introduced a better solution for this problem. They prevent particle deficiencies



(a) buckling (b) coiling

Fig. 8. Realistic buckling and coiling effects simulated with our implicit viscosity solver. Left: a sheet of 150k fluid particles is emitted with a viscosity parameter of $\gamma = 0.25$ to simulate buckling. Right: coiling is simulated by emitting 37k fluid particles with $\gamma = 0.75$.

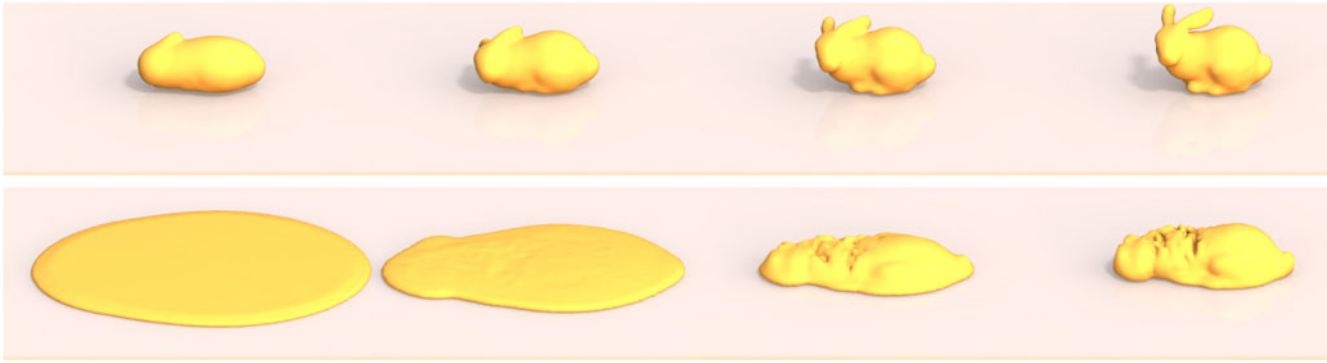


Fig. 9. Comparison of our method with the implicit viscosity solver of Peer et al. [41]. For the comparison we dropped a viscous bunny on the ground using different viscosity parameters (from left to right: $\gamma = 0.9$, $\gamma = 0.75$, $\gamma = 0.5$ and $\gamma = 0.25$). The top row shows the results of the solver of Peer et al. while the bottom row shows the results of our method. The comparison demonstrates that our approach covers a broader range of viscous materials and is able to handle low viscosity in contrast to the solver of Peer et al.

by generating ghost particles near the free surface to improve the physical behavior of the fluid. Avoiding pressure clamping by ghost particles would also allow to use solving algorithms with a better convergence rate like the conjugate gradient method in our pressure solvers. Therefore, the integration of ghost particles is one of our future goals. Another goal is to analyze if our method is able to improve the stability of multi-phase simulations with high density contrasts.

ACKNOWLEDGMENT

The work of the authors is supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School CE at TU Darmstadt. The dragon, bunny and armadillo models are courtesy of the Stanford Computer Graphics Laboratory. We would like to thank Manuel Scholz, Markus Ihmsen and Matthias Teschner for their help.

REFERENCES

[1] B. Solenthaler and R. Pajarola, "Predictive-corrective incompressible SPH," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 40:1–40:6, 2009.
[2] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner, "Implicit incompressible SPH," *IEEE Trans. Vis. Comput. Graph.*, vol. 20, no. 3, pp. 426–435, Mar. 2014.
[3] R. Bridson, *Fluid Simulation for Computer Graphics*. Boca Raton, FL, USA: A K Peters/CRC Press, 2008.
[4] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner, "SPH Fluids in Computer Graphics," in *Proc. Eurographics (State Art Rep.)*, 2014, pp. 21–42.
[5] J. Monaghan, "On the problem of penetration in particle methods," *J. Comput. Phys.*, vol. 82, no. 1, pp. 1–15, 1989.
[6] J. Monaghan, "Smoothed particle hydrodynamics," *Annu. Rev. Astronomy Astrophysics*, vol. 30, no. 1, pp. 543–574, 1992.
[7] J. Monaghan, "Simulating free surface flows with SPH," *J. Comput. Phys.*, vol. 110, pp. 399–406, 1994.
[8] M. Müller, D. Charypar, and M. Gross, "Particle-based fluid simulation for interactive applications," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2003, pp. 154–159.
[9] B. Adams, M. Pauly, R. Keiser, and L. J. Guibas, "Adaptively sampled particle fluids," *ACM Trans. Graph.*, vol. 26, no. 3, 2007, Art. no. 48.
[10] M. Becker and M. Teschner, "Weakly compressible SPH for free surface flows," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2007, pp. 1–8.
[11] A. J. Chorin, "Numerical solution of the Navier-Stokes equations," *Math. Comput.*, vol. 22, no. 104, pp. 745–762, 1968.
[12] N. Akinci, M. Ihmsen, G. Akinci, B. Solenthaler, and M. Teschner, "Versatile rigid-fluid coupling for incompressible SPH," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 62:1–62:8, 2012.
[13] N. Akinci, G. Akinci, and M. Teschner, "Versatile surface tension and adhesion for SPH fluids," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 1–8, 2013.
[14] M. Macklin and M. Müller, "Position based fluids," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 1–5, 2013.
[15] K. Bodin, C. Lacoursière, and M. Servin, "Constraint fluids," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 3, pp. 516–526, Mar. 2012.
[16] N. Kang and D. Sagong, "Incompressible SPH using the divergence-free condition," *Comput. Graph. Forum*, vol. 33, no. 7, pp. 219–228, 2014.
[17] S. J. Cummins and M. Rudman, "An SPH projection method," *J. Comput. Phys.*, vol. 152, pp. 584–607, 1999.
[18] N. Foster and R. Fedkiw, "Practical animation of liquids," *ACM Trans. Graph.*, vol. 28, pp. 12–17, 2001.
[19] Y. Zhu and R. Bridson, "Animating sand as a fluid," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 965–972, 2005.
[20] K. Raveendran, C. Wojtan, and G. Turk, "Hybrid smoothed particle hydrodynamics," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2011, pp. 33–42.
[21] R. Ando, N. Thürey, and C. Wojtan, "Highly adaptive liquid simulations on tetrahedral meshes," *ACM Trans. Graph.*, vol. 32, pp. 103:1–103:10, 2013.
[22] F. Losasso, J. O. Talton, N. Kwatra, and R. Fedkiw, "Two-way coupled SPH and particle level set fluid simulation," *IEEE Trans. Vis. Comput. Graph.*, vol. 14, no. 4, pp. 797–804, Jul./Aug. 2008.
[23] F. Sin, A. W. Bargteil, and J. K. Hodgins, "A point-based method for animating incompressible flow," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2009, pp. 247–255.
[24] X. Hu and N. Adams, "An incompressible multi-phase SPH method," *J. Comput. Phys.*, vol. 227, pp. 264–278, 2007.
[25] X. He, N. Liu, S. Li, H. Wang, and G. Wang, "Local poisson SPH for viscous incompressible fluids," *Comput. Graph. Forum*, vol. 31, pp. 1948–1958, 2012.
[26] F. de Goes, C. Wallez, J. Huang, D. Pavlov, and M. Desbrun, "Power particles: An incompressible fluid solver based on power diagrams," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 50:1–50:11, 2015.
[27] N. Foster and D. Metaxas, "Realistic animation of liquids," *Graph. Models Image Process.*, vol. 58, pp. 471–483, 1996.
[28] J. Stam, "Stable Fluids," in *Proc. ACM Comput. Graph. Interactive Tech.*, 1999, pp. 121–128.
[29] M. Carlson, P. J. Mucha, R. B. Van Horn III, and G. Turk, "Melting and flowing," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2002, pp. 167–174.
[30] N. Rasmussen, et al., "Directable photorealistic liquids," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2004, pp. 193–202.
[31] C. Batty and R. Bridson, "Accurate viscous free surfaces for buckling, coiling, and rotating liquids," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2008, pp. 219–228.
[32] C. Batty and B. Houston, "A simple finite volume method for adaptive viscous liquids," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2011, pp. 111–118.
[33] C. Batty, A. Uribe, B. Audoly, and E. Grinspun, "Discrete viscous sheets," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 1–7, 2012.
[34] T. G. Goktekin, A. W. Bargteil, and J. F. O'Brien, "A method for animating viscoelastic fluids," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 463–468, 2004.
[35] J. Morris and J. Monaghan, "A switch to reduce SPH viscosity," *J. Comput. Phys.*, vol. 136, no. 1, pp. 41–50, 1997.
[36] S. Premoe, T. Tasdizen, J. Bigler, A. Lefohn, and R. T. Whitaker, "Particle-based simulation of fluids," *Comput. Graph. Forum*, vol. 22, pp. 401–410, 2003.
[37] H. Schechter and R. Bridson, "Ghost SPH for animating water," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 61:1–61:8, 2012.
[38] A. Paiva, F. Petronetto, T. Lewiner, and G. Tavares, "Particle-based non-Newtonian fluid animation for melting objects," in *Proc. 19th IEEE Brazilian Symp. Comput. Graph. Image Process.*, 2006, pp. 78–85.
[39] A. Paiva, F. Petronetto, T. Lewiner, and G. Tavares, "Particle-based viscoplastic fluid/solid simulation," *Comput.-Aided Des.*, vol. 41, no. 4, pp. 306–314, 2009.
[40] L. F. de Souza Andrade, M. Sandim, F. Petronetto, P. Pagliosa, and A. Paiva, "SPH fluids for viscous jet buckling," in *Proc. 27th IEEE SIBGRAPI Conf. Graph. Patterns Images*, 2014, pp. 65–72.
[41] A. Peer, M. Ihmsen, J. Cornelis, and M. Teschner, "An implicit viscosity formulation for SPH fluids," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 1–10, 2015.
[42] T. Takahashi, Y. Dobashi, I. Fujishiro, T. Nishita, and M. Lin, "Implicit formulation for SPH-based viscous fluids," *Comput. Graph. Forum*, vol. 34, no. 2, pp. 493–502, 2015.
[43] S. Clavet, P. Beaudoin, and P. Poulin, "Particle-based viscoelastic fluid simulation," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, 2005, pp. 1–11.
[44] T. Takahashi, T. Nishita, and I. Fujishiro, "Fast simulation of viscous fluids with elasticity and thermal conductivity using position-based dynamics," *Comput. Graphics*, vol. 43, no. 1, pp. 21–30, 2014.
[45] T. Takahashi, Y. Dobashi, I. Fujishiro, and T. Nishita, "Volume preserving viscoelastic fluids with large deformations using position-based velocity corrections," *The Vis. Comput.*, vol. 32, pp. 57–66, 2014.
[46] D. Gerszewski, H. Bhattacharya, and A. W. Bargteil, "A point-based method for animating elastoplastic solids," in *Proc. ACM SIGGRAPH/Eurographics Symp. Comput. Animation*, vol. 1, 2009, pp. 133–138.

[47] B. Jones, S. Ward, A. Jallepalli, J. Perenia, and A. W. Bargteil, "Deformation embedding for point-based elastoplastic simulation," *ACM Trans. Graph.*, vol. 33, no. 2, pp. 1–9, 2014.
[48] C. Wojtan and G. Turk, "Fast viscoelastic behavior with thin features," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 47:1–47:8, 2008.
[49] M. Bergou, B. Audoly, E. Vouga, M. Wardetzky, and E. Grinspun, "Discrete viscous threads," *ACM Trans. Graph.*, vol. 29, no. 4, 2010, Art. no. 116.
[50] B. Zhu, M. Lee, E. Quigley, and R. Fedkiw, "Codimensional non-newtonian fluids," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 1–9, 2015.
[51] M. Desbrun and M.-P. Gascuel, "Smoothed particles: A new paradigm for animating highly deformable bodies," in *Proc. Eurographics Workshop Comput. Animation Simul.*, 1996, pp. 61–76.
[52] M. Ihmsen, N. Akinci, M. Becker, and M. Teschner, "A parallel SPH implementation on multi-core CPUs," *Comput. Graph. Forum*, vol. 30, no. 1, pp. 99–112, 2011.
[53] F. Losasso, T. Shinar, A. Selle, and R. Fedkiw, "Multiple interacting liquids," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 812–819, 2006.
[54] X. He, H. Wang, F. Zhang, H. Wang, G. Wang, and K. Zhou, "Robust simulation of sparsely sampled thin features in SPH-based free surface flows," *ACM Trans. Graph.*, vol. 34, no. 1, pp. 7:1–7:9, 2014.
[55] E. Coumans, The bullet physics library, 2015. [Online]. Available: http://www.bulletphysics.org

Jan Bender received the diploma, PhD and habilitation in computer science from the University of Karlsruhe. He is a assistant professor of computer science at the Graduate School CE, TU Darmstadt. His research interests include interactive simulation methods, multibody systems, deformable solids, fluid simulation, collision handling, fracture, GPGPU, and real-time visualization.

Dan Koschier received the MSc degree in computational engineering in 2014. He is working toward the PhD degree at TU Darmstadt. His research interests include physically-based simulation of deformable solids, cutting, fracture and fluids.