MLS Pressure Boundaries for Divergence-Free and Viscous SPH Fluids

Stefan Band[a,*], Christoph Gissler[a,b], Andreas Peer[a], Matthias Teschner[a]

[a] *University of Freiburg, Germany*
[b] *FIFTY2 Technology GmbH*

Abstract

In this paper we present a novel method to predict pressure values at boundary particles in incompressible divergence-free SPH simulations (DFSPH). Our approach employs Moving Least Squares (MLS) to predict the pressure at boundary particles. Therefore, MLS computes hyperplanes that approximate the pressure field at the interface between fluid and boundary particles. We compare this approach with three previous techniques. One previous technique mirrors the pressure from fluid to boundary particles. Another one extrapolates the pressure from fluid to boundary particles, but uses a gradient that is computed with Smoothed Particle Hydrodynamics (SPH). The third one solves a pressure Poisson equation (PPE) for boundary particles. In our experiments, we indicate artifacts in the three previous approaches. We show that these artifacts are significantly reduced with our approach resulting in simulation steps that can be twice as large. We motivate that gradient-based extrapolation is more accurate than mirroring. We further motivate that, due to particle deficiency at the boundary, the SPH gradient is error prone. This is less the case for our proposed MLS gradient. Moreover, our approach is computationally less expensive as solving a PPE for the boundary particles. We present challenging and complex scenarios to illustrate the capabilities of our method. In addition, we demonstrate that the proposed boundary handling is applicable to highly viscous fluids.

Keywords: physically-based animation, fluid simulation, Smoothed Particle Hydrodynamics, Moving Least Squares, boundary handling

This is the authors version of a work that was accepted for publication in Computers & Graphics. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in Computers & Graphics (Volume 76, November 2018, Pages 37-46) `https://doi.org/10.1016/j.cag.2018.08.001`

1. Introduction

Iterative pressure solvers such as PCISPH [1], IISPH [2] or DFSPH [3] compute a pressure field p and apply pressure accelerations of the form $\boldsymbol{a}_i^{\mathrm{p}} = -\frac{1}{\rho_i}\nabla p_i = -\sum_j m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right)\nabla W_{ij}$ to particles i. The sum considers all neighboring particles j of particle i. The variables $m, \rho, \nabla W$ denote mass, density and the gradient of the SPH kernel function W, respectively. There exist minor variations, but the solvers follow the same concept (see Section 4).

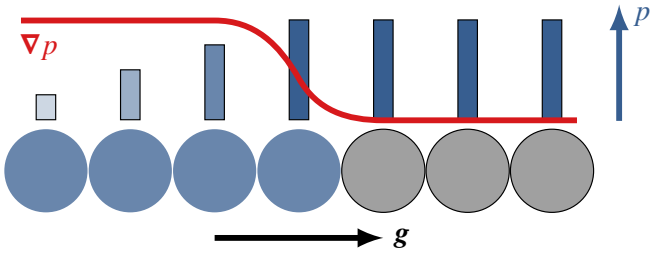
Iterative solvers typically compute pressure p_f only at fluid particles f. Pressure p_b at boundary particles b is not computed, but approximated if needed. This is, e.g., the case in the computation of the pressure acceleration $\boldsymbol{a}_f^{\mathrm{p}}$ at fluid particles f close to the boundary. Here, the respective SPH sum iterates over fluid neighbors with known pressure, but also over boundary neighbors with unknown pressure. I.e.,

$$\boldsymbol{a}_f^{\mathrm{p}} = -\sum_{f_f} m_{f_f}\left(\frac{p_f}{\rho_f^2} + \frac{p_{f_f}}{\rho_{f_f}^2}\right)\nabla W_{ff_f} - \sum_{f_b} m_{f_b}\left(\frac{p_f}{\rho_f^2} + \frac{p_{f_b}}{\rho_{f_b}^2}\right)\nabla W_{ff_b} \tag{1}$$

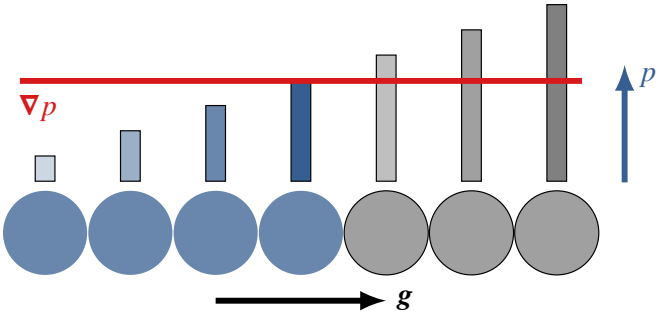
with f_f and f_b denoting fluid and boundary neighbors of fluid particle f, respectively. Equation (1) requires a notion of mass m_b, density ρ_b, and pressure p_b at boundary particles b. The density ρ_b is typically set to the rest density of the adjacent fluid and the mass m_b can be geometrically motivated from the boundary particle volume, see e.g. [4].

In terms of the pressure p_b, there exist different options. E.g., Akinci et al. [4] propose to mirror known pressure from fluid particles to adjacent rigid particles. Alternatively, Adami et al. [5] propose to extrapolate pressure from the fluid to boundary particles using an SPH approximation of the pressure gradient. Furthermore, Band et al. [6] propose to solve a pressure Poisson equation (PPE) for boundary particles.

*Corresponding author.
Email addresses: `bands@informatik.uni-freiburg.de` (Stefan Band), `gisslerc@informatik.uni-freiburg.de` (Christoph Gissler), `peera@informatik.uni-freiburg.de` (Andreas Peer), `teschner@informatik.uni-freiburg.de` (Matthias Teschner)



(a) *Mirrored pressure at boundary particles.*



(b) *Extrapolated pressure at boundary particles.*

Figure 1: *Mirroring pressure from fluid to boundary particles results in erroneous pressure gradients at fluid particles near the boundary. Extrapolating the pressure instead improves the gradient computation.*

Our contribution. This paper is an extended version of [7]. We propose a novel variant to predict unknown pressure p_b at boundary particles b. In contrast to Akinci et al. [4], we compute one unique pressure value per boundary particle. We further extrapolate the pressure using the pressure gradient instead of copying pressure from the fluid to the boundary. This improves the quality of the pressure gradient of fluid particles near the boundary as illustrated in Fig. 1. In contrast to Adami et al. [5], where pressure is extrapolated using SPH, we propose to use MLS [8, 9]. MLS is more accurate than SPH in case of particle deficiency which can particularly be the case near the boundary. We illustrate artifacts when using the boundary handling of Akinci et al. [4], Adami et al. [5] and Band et al. [6]. We further show that these artifacts can be reduced with the proposed MLS extrapolation of pressure values. We have implemented our boundary handling in a DFSPH framework. In contrast to our workshop paper [7], we propose to resolve the divergence error with preconditioned Conjugate Gradient (PCG) instead of relaxed Jacobi. To our best knowlege, PCG has never been used in an ISPH solver. All closely related solvers such as Local Poisson SPH (LPSPH) [10], IISPH [2], Position Based Fluids (PBF) [11] or DFSPH [3] employ Jacobi-style iterations. Capabilities of the approach are illustrated for scenarios with challenging boundary setups. We particularly show performance gain factors of up to two compared to [4]. Moreover, we show that our boundary handling approach is not only applicable to divergence-free fluids but can also be used in simulations with highly viscous fluids.

Organization. The remainder of this paper is organized as follows. The following Section 2 describes existing approaches related to SPH fluid simulation and the handling of solid boundaries. In Sections 3 and 5, we discuss the proposed pressure extrapolation concept whereas implementation details are described in Section 4. In Section 6, we show simulations employing our method and compare it to the boundary handling schemes of Akinci et al. [4], Adami et al. [5] and Band et al. [6]. Finally, we conclude in Section 7.

2. Related Work

SPH is a popular choice for Lagrangian simulations in computer graphics [12]. First used by Stam and Fiume [13] to simulate gaseous phenomena and by Desbrun and Gascuel [14] for deformable objects, Müller et al. [15] employed SPH to simulate compressible fluids. From that time on, research has focused on practical formulations of incompressible fluids with recent improvements in volume preservation [2, 3, 16], multiphase simulation [17, 18, 19, 20], highly viscous fluids [3, 21, 22, 23, 24, 25] and deformable objects [26, 27, 28]. Incompressibility can be enforced in various ways. Unlike non-iterative state equation solvers, e.g. [29, 15, 30, 31], iterative SPH pressure solvers, such as PCISPH [1], IISPH [2] and DFSPH [3], compute a pressure field p by solving a PPE of the form $\nabla^2 p = s$, c.f. [32]. Thereby, s is a source term that either encodes a predicted density deviation [33, 1, 2], the divergence of a velocity field [34, 35] or a combination of both [36, 3]. Computing the pressure field from a global formulation seems to improve the stability of the simulation. Rather large time steps can typically be used compared to the aforementioned non-iterative state equation solvers.

This paper focuses on the optimization of the boundary handling. Therefore, we briefly discuss related works regarding the modeling of solid boundaries in the next section.

Boundary Handling in SPH. As particle-based representations are very flexible and can handle arbitrarily shaped geometries, representing solid boundaries with particles is a popular choice for SPH fluid simulations, e.g. [37, 38, 2, 3, 16]. One popular technique for handling fluid-boundary contact is to apply penalty forces between two particles as soon as they are within a certain distance, e.g. [37, 39, 40]. Penalty forces should prevent fluid particles from penetrating the boundary. Therefore, the magnitude of the penalty force is determined based on a penetration measure between the particles. As the results are sensitive to the stiffness parameter of the penalty force, small time steps are typically required to produce a smooth pressure field.

In order to achieve larger time steps, the direct forcing method of Becker et al. [41] uses a predictor-corrector scheme to compute control forces and velocities. This method guarantees non-penetration. However, due to an incomplete support domain at the boundary, stacking of fluid particles can occur.

Another technique to treat boundaries are ghost particles [42, 43, 44]. For fluid particles that are located at a certain distance to the boundary, a narrow layer of ghost particle is generated. Those ghost particles mirror the hydrodynamic quantities of their associated fluid particle, i.e. they have the same viscosity, mass, density and pressure. However, for complex geometries,

generating such ghost particles is challenging. Furthermore, ghost particles have to be re-generated per simulation step.

Using only one layer of pre-generated boundary particles, Akinci et al. [4] treat irregular samplings by computing volume contributions, c.f. [45, 18], and by mirroring the quantities of a fluid particle onto its neighboring boundary particles. While adhering to the SPH concept, this approach is efficient to compute and allows a versatile coupling of fluids and solid objects [46]. Band et al. [47] proposed an extension of the boundary handling scheme of Akinci et al. [4] by employing MLS. To improve the accuracy of the density estimate and normal computation in planar regions, they locally reconstruct the surface of the true boundary by fitting boundary particles to a plane. This approach results in a smooth representation of the boundary. However, it is only applicable to planar boundaries. Furthermore, Band et al. [47] do neither compute unique pressure values nor perform any pressure extrapolation at boundary particles. Instead, they use the mirroring scheme of Akinci et al. [4]. Yet, MLS techniques have been successfully applied in many research areas, e.g. [48, 49, 50, 51].

Instead of mirroring fluid particle quantities onto the boundary, Adami et al. [5] propose to use pre-generated dummy boundary particles. Thereby, fluid particles at the boundary interact with dummy particles according to the overlap of the kernel function. This has the advantage that, even for complex geometries, the boundary is well-described through-out the simulation. Furthermore, by extrapolating the pressure of boundary particles from the surrounding fluid particles, this method allows an accurate approximation of a fluid particle's pressure gradient near the boundary.

To realize physically meaningful pressure and pressure gradients at the boundary, Band et al. [6] derived an extended PPE discretization at fluid and boundary particles. In contrast to previous approaches, their PPE does not only consider unknown pressure at fluid particles, but also at boundary particles. This led to reduced pressure oscillations, improved solver convergence and larger time steps.

As an alternative to particles, boundaries are also representable with triangle meshes [52, 53]. Yet, as stated in [4], handling discontinuous surface normals and non-manifold structures that cause spatial and temporal discontinuities of the fluid properties is challenging for triangulated boundaries. Another alternative is an implicit representation of the boundary as proposed by Koschier and Bender [54]. Based on a pre-computed density map, this approach allows to efficiently evaluate the density and pressure gradient of fluid particles at the boundary.

3. Method

We first discuss the previous concepts of Akinci et al. [4] in Section 3.1, Adami et al. [5] in Section 3.2 and Band et al. [6] in Section 3.3. Our approach is introduced in Section 3.4. This section focuses on the concepts. The combination of the boundary handling with DFSPH is described in Section 4.

3.1. Pressure mirroring

Pressure mirroring is motivated by its simple and efficient implementation. When a pressure acceleration is computed at a fluid particle f that has a boundary particle f_b with unknown pressure in its neighborhood, the pressure at the boundary particle is simply set to the pressure of the fluid particle, i.e. $p_{f_b} = p_f$. The computation in Eq. (1) slightly changes to

$$\boldsymbol{a}_f^{\mathrm{p}} = -\sum_{f_f} m_{f_f}\left(\frac{p_f}{\rho_f^2}+\frac{p_{f_f}}{\rho_{f_f}^2}\right)\nabla W_{ff_f} - \sum_{f_b}\rho_f^0 V_{f_b}\left(\frac{p_f}{\rho_f^2}+\frac{p_f}{(\rho_f^0)^2}\right)\nabla W_{ff_b}\,. \tag{2}$$

Compared to Eq. (1), the mass of a boundary neighbor m_{f_b}, i.e. its contribution in the SPH sum, is computed as $m_{f_b} = \rho_f^0 V_{f_b}$ and the density of a boundary neighbor is set to $\rho_{f_b} = \rho_f^0$. Please refer to [4] for a motivation of these choices and for a discussion how to compute the volume V_{f_b}.

While density and mass in Eq. (2) play an important role for the accurate weighting of a boundary particle in the SPH sum, the employed pressure approximation $p_{f_b} = p_f$ negatively affects the accuracy of the pressure gradient computation, i.e. the computation of the pressure acceleration. As illustrated in Fig. 1, it would be more appropriate to extrapolate the pressure from the fluid to the boundary.

Pressure mirroring does not require to iterate over boundary particles or to store pressure values at boundary particles. If a boundary pressure is required, it is simply set to the pressure of the currently processed fluid particle. While this efficiency is positive, it results in inconsistent pressure values at boundary particles. If two fluid particles f^1 and f^2 with different pressure values p_{f^1} and p_{f^2} share the same boundary particle b, the gradient computations at both fluid particles work with different pressure values. Fluid particle f^1 works with $p_b = p_{f^1}$ in Eq. (2), while the other fluid particle f^2 uses $p_b = p_{f^2}$ at the same boundary particle b.

3.2. Pressure extrapolation with SPH

Pressure extrapolation can be motivated by Pascal's law for hydrostatic pressure which states that the pressure difference at two fluid points is proportional to their height difference. For a boundary particle b and an adjacent fluid particle b_f, this can be written as $p_b = p_{b_f} + \rho_{b_f}\boldsymbol{g}\cdot\boldsymbol{x}_{bb_f}$ with gravity $\boldsymbol{g}$ and distance $\boldsymbol{x}_{bb_f} = \boldsymbol{x}_b - \boldsymbol{x}_{b_f}$. In order to handle the interaction of one boundary particle with several neighboring fluid particles, the respective contributions are weighted with the kernel function W_{bb_f}, summed up and normalized as proposed by Adami et al. [5]:

$$p_b = \frac{\sum_{b_f} p_{b_f} W_{bb_f} + \boldsymbol{g}\cdot\sum_{b_f}\rho_{b_f}\boldsymbol{x}_{bb_f}W_{bb_f}}{\sum_{b_f} W_{bb_f}}\,. \tag{3}$$

In contrast to the pressure mirroring in [4], this approach requires an additional loop over boundary particles to compute the pressure which is also stored at boundary particles. On the

other hand, boundary pressures are not inconsistent as in [4]. Instead, each boundary particle is attributed a unique pressure value. If the pressure p_b is computed for all boundary particles b, Eq. (1) can be used to compute the pressure acceleration at fluid particles.

Although the computation in Eq. (3) is normalized, it nevertheless suffers from the typical SPH particle deficiency issue. The neighborhood of a boundary particle is only partially filled with fluid neighbors which falsifies the pressure computation.

3.3. Pressure Boundaries

Here, we first describe the general idea of ISPH [34], followed by the specific PPE discretization of Pressure Boundaries [6].

ISPH computes the pressure field p by solving a PPE of the form $\nabla^2 p = s$ with s being a source term, e.g. [34, 33, 35, 36, 1, 2, 3]. Compared to state-equation based solvers, e.g. [29, 17, 30], this can positively affect the stability. Which in turn allows iterative solvers to use a larger time step size and therefore results in faster simulations. To solve the PPE, different discretizations of $\nabla^2 p$ can be used. Moreover, the source term s can also be computed in different ways [55].

Pressure Boundaries [6] employs the IISPH [2] discretization of $\nabla^2 p$, i.e. $\langle \nabla \cdot \langle \nabla p \rangle \rangle$ is computed rather than $\langle \nabla^2 p \rangle$. However, in contrast to IISPH, not only unknown pressure values at fluid particles are considered, but also at boundary particles. This is motivated by the various assumptions that are made in the boundary handling of IISPH. Since IISPH uses the boundary handling of Akinci et al. [4], it assumes inconsistent pressure values at the boundary, as explained in Section 3.1. However, if the PPE is extended by unique pressure values for boundary particles (which might be similar to pressure extrapolation), the robustness of the boundary handling is improved. This allows the usage of larger time steps.

The resulting linear system of Pressure Boundaries consists of one equation per unknown pressure value for fluid particles and one equation per unknown pressure value for boundary particles. At fluid particles f, the equation is given by

$$\sum_{f_f} \frac{m_{f_f}}{\rho_{f_f}} \left(\boldsymbol{a}^{\mathrm{p}}_f - \boldsymbol{a}^{\mathrm{p}}_{f_f} \right) \cdot \nabla W_{f f_f} + \sum_{f_b} \frac{m_{f_b}}{\rho_{f_b}} \boldsymbol{a}^{\mathrm{p}}_f \cdot \nabla W_{f f_b} = \frac{1}{\Delta t^2} \left(1 - \frac{\rho^0_f}{\rho_f} \right) + \frac{1}{\Delta t} \nabla \cdot \boldsymbol{v}^\star_f \,. \quad (4)$$

As computing the pressure acceleration at boundary particles is more involved, Pressure Boundaries assumes that $\boldsymbol{a}^{\mathrm{p}}_b = \boldsymbol{0}$. With this simplification, the equation at boundary particles b can be derived as

$$- \sum_{b_f} \frac{m_{b_f}}{\rho_{b_f}} \boldsymbol{a}^{\mathrm{p}}_{b_f} \cdot \nabla W_{b b_f} = \frac{1}{\Delta t^2} \left(1 - \frac{\rho^0_b}{\rho_b} \right) + \frac{1}{\Delta t} \nabla \cdot \boldsymbol{v}^\star_b \,. \quad (5)$$

To compute the source term of boundary particles (the right-hand-side of Eq. (5)), the ratio of rest density ρ^0_b and density ρ_b has to be determined. Since $\rho = m/V$, Band et al. [6] follow the concept of volume elements, c.f. [18, 56], to rewrite $\rho^0_b/\rho_b = V_b/V^0_b$. I.e. a special treatment of the density computation at the interface can be avoided by using volumes instead. Thereby, the rest volume V^0_b is computed as

$$V^0_b = \frac{\gamma}{\sum_{b_b} W_{b b_b}} \,. \quad (6)$$

The coefficient γ accounts for an incomplete neighborhood (as only one layer of boundary particles is used to represent the surface of the boundary [4]). The actual volume V_b of a boundary particle b is computed as

$$V_b = \frac{V^0_b}{\sum_{b_f} h^3 W_{b b_f} + \gamma + \beta} \,, \quad (7)$$

with h being the initial particle spacing and β another coefficient to account for incomplete neighborhoods.

The left-hand-side of Eqs. (5) and (4) corresponds to the discretization of $\nabla^2 p$. Accordingly, the right-hand-side of Eqs. (5) and (4) corresponds to the discretized source term s. To solve this linear system of equations, Band et al. [6] employ relaxed Jacobi. Thereby, the pressure values of fluid and boundary particles i are iteratively updated with

$$p^{l+1}_i = \max \left(0, p^l_i + \omega_i \frac{s_i - \nabla^2 p_i}{\lambda_i} \right) , \quad (8)$$

where l denotes the iteration number, λ_i the diagonal element and ω_i the relaxation factor. As Band et al. [6] reported convergence issues in case of large volume ratios between fluid and boundary particles, they used different relaxation factors for fluid ($\omega_f = 0.5$) and boundary particles ($\omega_b = 0.5\, V^0_b/h^3$).

Although Pressure Boundaries provides consistent pressure values and pressure gradients at boundary particles and tries to account for the particle deficiency issue, it has the drawback of solving a larger PPE compared to, e.g., IISPH [2] or DFSPH [3]. Therefore, it uses more memory and might require more computation time.

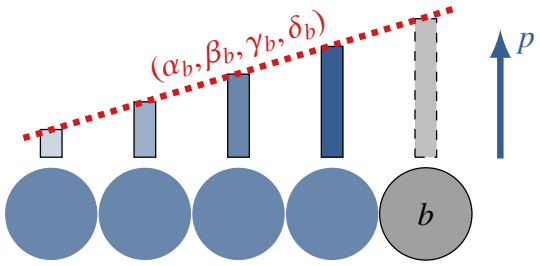


Figure 2: *MLS hyperplane fitting to compute pressure at a boundary particle b (grey). The plane parameters $\alpha_b, \beta_b, \gamma_b$ and δ_b are estimated from pressure values at adjacent fluid particles (blue).*

3.4. Pressure Extrapolation with MLS

We propose to resolve the particle deficiency issue by using MLS instead of SPH for the pressure computation at boundary particles. Therefore, we fit hyperplanes that approximate the pressure field as illustrated in Fig. 2. This is conceptually different to Band et al. [47] where MLS is used to fit planes through

boundary sample positions. In our case, MLS is performed from the perspective of a boundary particle b. It employs all known pressure values p_{b_f} of fluid neighbors b_f adjacent to this boundary particle b to fit a hyperplane through the pressure field.

Problem Formulation. We wish to obtain a function $\tilde{p}_b(\boldsymbol{x})$ that approximates the given pressure values p_{b_f} of b's fluid neighbors b_f in a least-squares sense. Thus, we have to

$$\text{minimize} \sum_{b_f} \left(\tilde{p}_b(\boldsymbol{x}_{b_f}) - p_{b_f} \right)^2 V_{b_f} W_{bb_f} \,. \tag{9}$$

The function $\tilde{p}_b$ can be written as $\tilde{p}_b(\boldsymbol{x}) = \boldsymbol{b}(\boldsymbol{x}) \cdot \boldsymbol{c}_b$, c.f. [9], where $\boldsymbol{b}(\boldsymbol{x})$ is the polynomial basis vector and $\boldsymbol{c}_b = (\alpha_b, \beta_b, \gamma_b, \delta_b)^{\mathrm{T}}$ is the vector of unknown coefficients that describe b's hyperplane. To obtain a first order accurate approximation scheme, we use the linear basis $\boldsymbol{b}(\boldsymbol{x}) = (1, x, y, z)^{\mathrm{T}}$. Considering the fact that the partial derivatives of Eq. (9) with respect to the hyperplane parameters should be zero at the minimum, we get the following system that has to be solved:

$$\sum_{b_f} \boldsymbol{b}(\boldsymbol{x}_{b_f}) \left(\boldsymbol{b}(\boldsymbol{x}_{b_f}) \cdot \boldsymbol{c}_b - p_{b_f} \right) V_{b_f} W_{bb_f} = \boldsymbol{0} \,. \tag{10}$$

In order to solve for the unknown hyperplane parameters $\boldsymbol{c}_b$, we rewrite Eq. (10) as

$$\sum_{b_f} \left(\boldsymbol{b}(\boldsymbol{x}_{b_f}) \otimes \boldsymbol{b}(\boldsymbol{x}_{b_f}) \right) \boldsymbol{c}_b V_{b_f} W_{bb_f} = \sum_{b_f} \boldsymbol{b}(\boldsymbol{x}_{b_f}) p_{b_f} V_{b_f} W_{bb_f} \tag{11}$$

which corresponds to

$$\left(\sum_{b_f} \begin{bmatrix} 1 & x_{b_f} & y_{b_f} & z_{b_f} \\ x_{b_f} & x_{b_f}^2 & x_{b_f} y_{b_f} & x_{b_f} z_{b_f} \\ y_{b_f} & x_{b_f} y_{b_f} & y_{b_f}^2 & y_{b_f} z_{b_f} \\ z_{b_f} & x_{b_f} z_{b_f} & y_{b_f} z_{b_f} & z_{b_f}^2 \end{bmatrix} V_{b_f} W_{bb_f} \right) \begin{bmatrix} \alpha_b \\ \beta_b \\ \gamma_b \\ \delta_b \end{bmatrix} = \sum_{b_f} \begin{bmatrix} 1 \\ x_{b_f} \\ y_{b_f} \\ z_{b_f} \end{bmatrix} p_{b_f} V_{b_f} W_{bb_f} \,. \tag{12}$$

This 4×4 system can be rewritten such that α_b can be directly computed and the parameters β_b, γ_b, δ_b can be computed by solving a 3×3 system. We therefore propose to perform a basis transform: all considered positions $\boldsymbol{x}_{b_f}$ and also $\boldsymbol{x}_b$ are translated to positions $\bar{\boldsymbol{x}}_{b_f}$ and $\bar{\boldsymbol{x}}_b$ by

$$\boldsymbol{d}_b = \frac{\sum_{b_f} \boldsymbol{x}_{b_f} V_{b_f} W_{bb_f}}{\sum_{b_f} V_{b_f} W_{bb_f}} \tag{13}$$

i.e.

$$\bar{\boldsymbol{x}}_{b_f} = (\bar{x}_{b_f}, \bar{y}_{b_f}, \bar{z}_{b_f})^{\mathrm{T}} = \boldsymbol{x}_{b_f} - \boldsymbol{d}_b \tag{14}$$

$$\bar{\boldsymbol{x}}_b = (\bar{x}_b, \bar{y}_b, \bar{z}_b)^{\mathrm{T}} = \boldsymbol{x}_b - \boldsymbol{d}_b. \tag{15}$$

This basis transform, i.e. the translation of all incorporated particle positions by the same vector $\boldsymbol{d}_b$, does not affect the parameters of the hyperplane. Thus, instead of solving Eq. (12), we now consider the following system with the same solution:

$$\left(\sum_{b_f} \begin{bmatrix} 1 & \bar{x}_{b_f} & \bar{y}_{b_f} & \bar{z}_{b_f} \\ \bar{x}_{b_f} & \bar{x}_{b_f}^2 & \bar{x}_{b_f} \bar{y}_{b_f} & \bar{x}_{b_f} \bar{z}_{b_f} \\ \bar{y}_{b_f} & \bar{x}_{b_f} \bar{y}_{b_f} & \bar{y}_{b_f}^2 & \bar{y}_{b_f} \bar{z}_{b_f} \\ \bar{z}_{b_f} & \bar{x}_{b_f} \bar{z}_{b_f} & \bar{y}_{b_f} \bar{z}_{b_f} & \bar{z}_{b_f}^2 \end{bmatrix} V_{b_f} W_{bb_f} \right) \begin{bmatrix} \alpha_b \\ \beta_b \\ \gamma_b \\ \delta_b \end{bmatrix} = \sum_{b_f} \begin{bmatrix} 1 \\ \bar{x}_{b_f} \\ \bar{y}_{b_f} \\ \bar{z}_{b_f} \end{bmatrix} p_{b_f} V_{b_f} W_{bb_f} \,. \tag{16}$$

Please note that W_{bb_f} in Eq. (16) depends on the distance between $\boldsymbol{x}_b$ and $\boldsymbol{x}_{b_f}$. Now, some elements of the 4×4 matrix can be replaced by zero based on the following observation:

$$\begin{aligned} \sum_{b_f} \bar{\boldsymbol{x}}_{b_f} V_{b_f} W_{bb_f} &= \sum_{b_f} \left(\boldsymbol{x}_{b_f} - \boldsymbol{d}_b \right) V_{b_f} W_{bb_f} \\ &= \sum_{b_f} \boldsymbol{x}_{b_f} V_{b_f} W_{bb_f} - \boldsymbol{d}_b \sum_{b_f} V_{b_f} W_{bb_f} \\ &= \boldsymbol{d}_b \sum_{b_f} V_{b_f} W_{bb_f} - \boldsymbol{d}_b \sum_{b_f} V_{b_f} W_{bb_f} \\ &= \boldsymbol{0} \,. \end{aligned} \tag{17}$$

Thereby, we get the following form:

$$\left(\sum_{b_f} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \bar{x}_{b_f}^2 & \bar{x}_{b_f} \bar{y}_{b_f} & \bar{x}_{b_f} \bar{z}_{b_f} \\ 0 & \bar{x}_{b_f} \bar{y}_{b_f} & \bar{y}_{b_f}^2 & \bar{y}_{b_f} \bar{z}_{b_f} \\ 0 & \bar{x}_{b_f} \bar{z}_{b_f} & \bar{y}_{b_f} \bar{z}_{b_f} & \bar{z}_{b_f}^2 \end{bmatrix} V_{b_f} W_{bb_f} \right) \begin{bmatrix} \alpha_b \\ \beta_b \\ \gamma_b \\ \delta_b \end{bmatrix} = \sum_{b_f} \begin{bmatrix} 1 \\ \bar{x}_{b_f} \\ \bar{y}_{b_f} \\ \bar{z}_{b_f} \end{bmatrix} p_{b_f} V_{b_f} W_{bb_f} \,. \tag{18}$$

Solution. Now, we can directly solve for α_b:

$$\alpha_b = \frac{\sum_{b_f} p_{b_f} V_{b_f} W_{bb_f}}{\sum_{b_f} V_{b_f} W_{bb_f}} \,, \tag{19}$$

which corresponds to the weighted average of the pressure values of fluid particles b_f adjacent to boundary particle b. The other hyperplane parameters β_b, γ_b and δ_b are obtained by solving

$$\begin{bmatrix} \beta_b \\ \gamma_b \\ \delta_b \end{bmatrix} = \left(\sum_{b_f} \begin{bmatrix} \bar{x}_{b_f}^2 & \bar{x}_{b_f} \bar{y}_{b_f} & \bar{x}_{b_f} \bar{z}_{b_f} \\ \bar{x}_{b_f} \bar{y}_{b_f} & \bar{y}_{b_f}^2 & \bar{y}_{b_f} \bar{z}_{b_f} \\ \bar{x}_{b_f} \bar{z}_{b_f} & \bar{y}_{b_f} \bar{z}_{b_f} & \bar{z}_{b_f}^2 \end{bmatrix} V_{b_f} W_{bb_f} \right)^{-1} \sum_{b_f} \begin{bmatrix} \bar{x}_{b_f} \\ \bar{y}_{b_f} \\ \bar{z}_{b_f} \end{bmatrix} p_{b_f} V_{b_f} W_{bb_f} \,. \tag{20}$$

Finally, the pressure p_b at boundary particle b is computed as

$$p_b = (1, \bar{x}_b, \bar{y}_b, \bar{z}_b)^{\mathrm{T}} \cdot \boldsymbol{c}_b \,. \tag{21}$$

In our experiments, however, we experienced issues with a singular matrix in Eq. (20) for boundary particles whose fluid neighbors are co-linear or co-planar. In these cases, we follow Müller et al. [50] and use safe inversion via Singular Value Decomposition (SVD) [57] to avoid the problems with singular matrices.

4. Implementation

We have combined our proposed boundary handling with a slightly modified DFSPH solver [3] that is outlined in Algorithm 1. We follow the idea of combining two solvers, one for the velocity divergence and one for the density invariance. As a minor notation change to DFSPH, our two solvers compute pressure p instead of stiffness parameter κ. DFSPH implicitly introduces κ with the relation $\nabla p_f = \sum_{f_j} m_{f_j} \kappa_{f_j} \nabla W_{ff_j}$ with f_j denoting a fluid or a boundary neighbor of f. From the SPH formulation $\nabla p_f = \sum_{f_j} \frac{m_{f_j}}{\rho_{f_j}} p_{f_j} \nabla W_{ff_j}$, it follows that $p_{f_j} = \kappa_{f_j} \rho_{f_j}$. So, we use the DFSPH solver, but multiply all κ values with the density ρ to get pressure values.

The functions CorrectDivergenceError and CorrectDensityError in Algorithm 1 compute pressure at fluid particles, but they are also responsible for the pressure computation at boundary particles. In case of pressure mirroring, the pressure is not explicitly computed, but just considered in the computation of $\boldsymbol{a}^*$ and $\boldsymbol{a}^{**}$ by using Eq. (2) instead of Eq. (1). The SPH extrapolation and our proposed MLS extrapolation loop over the boundary particles to compute and store pressure. Then, Eq. (1) is used to compute the pressure accelerations $\boldsymbol{a}^*$ and $\boldsymbol{a}^{**}$. Finally, we update the pressure values p_f of fluid particles f in a Jacobi step. In our experiments, we set the relaxation coefficient $\omega = 0.5$. In case of Pressure Boundaries, pressure at the boundary is not re-computed in each iteration, but updated with a Jacobi step (Eq. (8)).

If all particles have the same rest density and mass, we propose to solve the linear system that is used to resolve the velocity-divergence error with PCG instead of relaxed Jacobi. For the pre-conditioner, we use the diagonal element λ_f. Since we require the computation of the diagonal element for resolving the density-invariant error anyway, this pre-conditioner choice does not add any considerable computational overhead. PCG can be applied because we do not have to clamp pressure values in-between the iterations (unlike for the density-invariant error as stated by Ihmsen et al. [2] and Takahashi et al. [16]).

5. Discussion

Concept. Using a Taylor approximation, the continuous pressure field $\tilde{p}$ in the neighborhood of a boundary particle b can be approximated as

$$\tilde{p}(\boldsymbol{x}_b + \Delta\boldsymbol{x}) = p(\boldsymbol{x}_b) + \langle \nabla p \rangle(\boldsymbol{x}_b) \cdot \Delta\boldsymbol{x} + O\left(\|\Delta\boldsymbol{x}\|^2\right). \qquad (22)$$

Since we do not know the pressure value p at position $\boldsymbol{x}_b$, our MLS fitting approach is conceptually different to, e.g., [50, 58, 30]. They use MLS to estimate the gradient, i.e. three unknowns,

Algorithm 1 DFSPH with MLS pressure extrapolation and PCG

procedure PerformSimulation
  for each fluid particle f **do**
    find neighborhoods $N_f(t)$
  for each fluid particle f **do**
    compute density $\rho_f(t)$
    compute factor $\lambda_f \leftarrow \frac{\left\|\sum_j m_j \nabla W_{fj}\right\|^2 + \sum_{f_f} m_{f_f} m_{f_f} \left\|\nabla W_{f f_f}\right\|^2}{-\rho_f(t)^2}$
  $\boldsymbol{a}^* \leftarrow$ CorrectDivergenceError ▷ divergence-free
  for each fluid particle f **do**
    predict velocity $\boldsymbol{v}_f^* \leftarrow \boldsymbol{v}_f(t) + \Delta t\, \boldsymbol{a}_f^*$
  for each fluid particle f **do**
    predict velocity $\boldsymbol{v}_f^{**} \leftarrow \boldsymbol{v}_f^* + \Delta t\, \boldsymbol{a}_f^{\text{non-pressure}}$
  $\boldsymbol{a}^{**} \leftarrow$ CorrectDensityError ▷ density invariant
  for each fluid particle f **do**
    update velocity $\boldsymbol{v}_f(t + \Delta t) \leftarrow \boldsymbol{v}_f^{**} + \Delta t\, \boldsymbol{a}_f^{**}$
    update position $\boldsymbol{x}_f(t + \Delta t) \leftarrow \boldsymbol{x}_f(t) + \Delta t\, \boldsymbol{v}_f(t + \Delta t)$

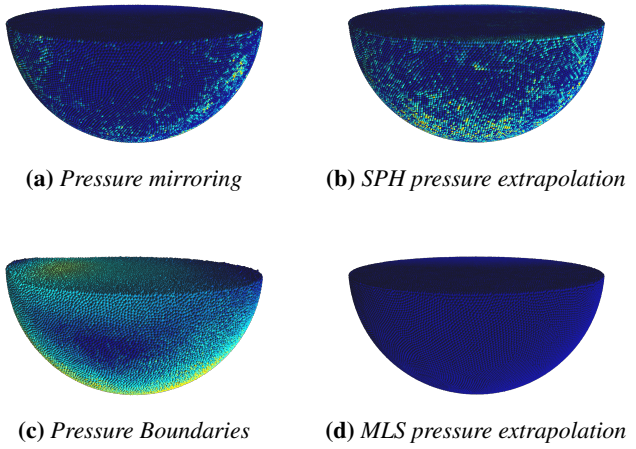
procedure CorrectDivergenceError
  for each fluid particle f **do**
    compute source term $s_f \leftarrow -\frac{1}{\Delta t} \nabla \cdot \boldsymbol{v}_f(t)$
    initialize pressure $p_f \leftarrow 0$
  while not converged **do**
    for each boundary particle b **do**
      compute pressure p_b using MLS ▷ Eq. (21)
    for each fluid particle f **do**
      compute pressure acceleration $\boldsymbol{a}_f^*$ ▷ Eq. (1)
    for each fluid particle f **do**
      set pressure $p_f \leftarrow p_f + \frac{\omega}{\lambda_f}\left(s_f - \nabla \cdot \boldsymbol{a}_f^*\right)$
  return $\boldsymbol{a}^*$

procedure CorrectDensityError
  for each fluid particle f **do**
    compute source term $s_f \leftarrow \frac{1}{\Delta t^2}\left(1 - \frac{\rho_f^0}{\rho_f(t)}\right) - \frac{1}{\Delta t} \nabla \cdot \boldsymbol{v}_f^{**}$
    initialize pressure $p_f \leftarrow 0$
  while not converged **do**
    for each boundary particle b **do**
      compute pressure p_b using MLS ▷ Eq. (21)
    for each fluid particle f **do**
      compute pressure acceleration $\boldsymbol{a}_f^{**}$ ▷ Eq. (1)
    for each fluid particle f **do**
      set pressure $p_f \leftarrow \max\left(0, p_f + \frac{\omega}{\lambda_f}\left(s_f - \nabla \cdot \boldsymbol{a}_f^{**}\right)\right)$
  return $\boldsymbol{a}^{**}$

at a position with a known value. In contrast to this, we estimate the gradient and an offset, i.e. four unknowns.

Computation. Similar to the SPH pressure extrapolation and Pressure Boundaries, our approach performs a loop over boundary particles to compute and store pressure at boundary particles. Each boundary particle has a unique pressure value. In contrast to the SPH extrapolation, our MLS approach does not suffer from the particle deficiency issue.

Pressure Boundaries. Our approach is more computationally and memory efficient than Pressure Boundaries as it does not have the overhead associated with solving a PPE. It does not require the computation and storage of actual volumes, diagonal elements, source terms and divergence of the pressure accelerations for boundary particles. Another benefit of our approach is the reduced parameter count compared to Pressure Boundaries. First, it does not depend on a relaxation factor ω_b for boundary particles. And second, it does not require the volume correction coefficient β in case of particle deficiency as no actual volumes must be computed for boundary particles.



(a) *Pressure mirroring* **(b)** *SPH pressure extrapolation*

(c) *Pressure Boundaries* **(d)** *MLS pressure extrapolation*

Figure 3: *Comparison of the different boundary handling schemes. Velocities are color-coded with blue corresponding to minimal and red corresponding to maximal velocity. In contrast to pressure mirroring (3a), pressure extrapolation with SPH (3b) and Pressure Boundaries (3c), our approach (3d) does not show an incorrect movement of the fluid particles.*

6. Results

In this section, we compare our proposed MLS pressure extrapolation to the pressure mirroring of Akinci et al. [4], the pressure extrapolation with SPH of Adami et al. [5] and the extended PPE of Band et al. [6]. We show that our approach can handle challenging scenarios, such as complex and fast-moving boundary geometries and high water depths. We use different particle spacings and time steps for the simulations. Table 2 gives an overview of the scenarios. For the SPH interpolation, we use the cubic spline kernel [37] with a support of two times the particle spacing. The rest density of the fluids is $1000\,\mathrm{kg\,m^{-3}}$ while the largest permissible compression error is 0.1 %. Furthermore, as we want to focus on comparing concepts instead of solver implementations, we never performed a warm-start for the pressure solver, i.e. we always initialize pressure values with zero. Besides, we employed the PCG solver only for the experiments that are described in Sections 6.3, 6.7 and 6.8.

In our implementation, we employ compact hashing [59] for the neighbor search. We apply a drag force to the fluid as described in [60] and model surface tension as proposed in [61]. Viscosity is modeled as proposed by [62]. To reduce the loss in turbulent details, we use a micropolar material model [63]. All computations are fully parallelized with Intel Threading Building Blocks [64]. We use [65] to reconstruct the fluid surface. The ray-traced images are rendered with [66]. All presented scenarios have been computed on a 12-core 2.6 GHz Intel Xeon E5-2690 with 32 GB of RAM.

6.1. Rotating Sphere

First, we compare our new approach to [4, 5, 6] in a setting where fluid is placed inside a rigid sphere with free-slip boundary conditions as illustrated in Fig. 3. The boundary sphere has a radius of 3 m and is rotating slowly at 7 revolutions per minute. We use a particle spacing of 5 cm. The scenario consists of 417 k fluid and 44.5 k boundary particles and is simulated for ten seconds with a fixed time step of 1 ms. The number of density invariant iterations of our DFSPH solver was fixed to ten while the divergence-free iterations count was set to zero. This basically corresponds to an IISPH solver and resulted in a density error of approximately 0.037 %.

In this experiment, the boundary particles should not influence the fluid velocities and the fluid should rest inside the sphere. However, as shown in Fig. 3, this is not the case for [4, 5, 6]. All three boundary handling schemes introduce an artificial viscosity, which causes an incorrect movement of the fluid particles. In contrast to this, our MLS pressure extrapolation does not suffer from artificial viscosity at the boundary. The computation time for the pressure field is very similar for all approaches (our: 64.29 ms, [4]: 63.12 ms, [5]: 63.44 ms, [6]: 65.89 ms).

The reason why Pressure Boundaries [6] has the largest error is that it has more potential sources for errors: First, the discretization of the source term of boundary particles might be erroneous. Second, the correction factor β is used to account for missing particle neighbor contributions. As discussed in Band et al. [6], the boundary is assumed to be planar to calculate this factor, which is obviously not the case in this scenario.

6.2. Breaking Dam

In order to compare the solver iteration counts of our approach with [4, 5, 6], we simulate a breaking dam scenario inside a cylindric-shaped domain of size $3\,\mathrm{m}\times 3\,\mathrm{m}\times 0.5\,\mathrm{m}$ with a particle spacing of 8 mm. Thus, making a total of 2.95 million fluid and 182.2 k boundary particles. The scenario is illustrated in Fig. 4. Furthermore, we use different fixed time step sizes. Table 1 summarizes the iteration measurements and computation times to solve the PPE for a simulation over ten seconds.

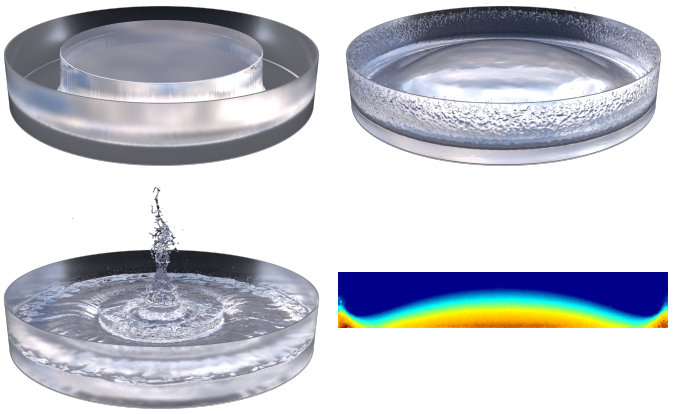


Figure 4: *Cylindrical breaking dam with 2.95 million fluid particles simulated with our MLS pressure extrapolation approach. The smooth color-coded pressure field on the bottom-right corresponds to the top-right frame.*

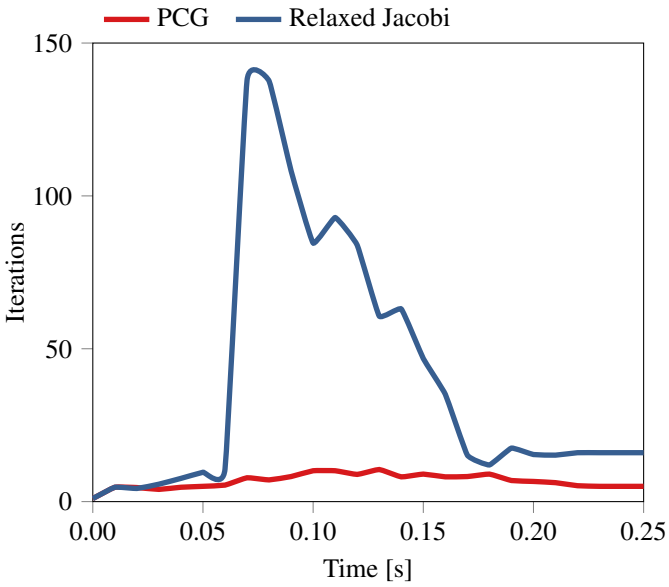


Figure 5: *Iteration counts for the divergence-free solve with PCG and relaxed Jacobi for the Breaking Dam scenario.*

In our experiment, our MLS pressure extrapolation approach always requires the minimum number of iterations per simulation step. For larger time step sizes, our approach outperforms [5] due to more accurate pressure gradients at the boundary. Computing unique pressure values for boundary particles is not expensive. For [5], we measured an average computation time for the boundary pressures of 1.12 ms per iteration. For our approach, as we have to iterate twice over fluid neighbors of the boundary particles, the computation time slightly increased to 1.87 ms. Furthermore, even solving a system for the boundary pressure (1.95 ms) is not much more expensive.

6.3. Pre-conditioned Conjugate Gradient vs. Relaxed Jacobi

To compare the divergence-free solve with PCG and relaxed Jacobi, we simulated the breaking dam scene described in Section 6.2 with a fixed time step of 1 ms for 0.25 s, i.e. 250 simulation steps. We measured the iteration counts required to halve the initial divergence error.

Fig. 5 shows the results. Although PCG (76.1 ms) is more expensive per iteration than relaxed Jacobi (56.5 ms), it performs better overall since it requires less iterations to reach the allowed error. Therefore, instead of resolving the divergence error with relaxed Jacobi, as it was done in our previous work [7], we propose to use PCG.

6.4. Vase

As shown in [6], computing unique pressure values for boundary particles can be beneficial for scenarios with a high water depth. In order to show that our approach can also handle such challenging scenarios, we simulate a vase of height 10 m that is filled with water over a duration of eighteen seconds. The scene is illustrated in Fig. 6. The particle spacing is 2 cm and the adaptive time step [38] is 0.47 ms on average. The scene consists of up to 13.33 million fluid and 826 k boundary particles. The total computation time per simulation step is 3.70 s on average with MLS pressure extrapolation, 3.72 s for pressure mirroring and 3.84 s for SPH pressure extrapolation. The reduced computation time of our approach is the result of a reduced solver iteration count.

6.5. Teacup

In order to demonstrate the applicability of our approach to two-way coupled dynamic objects, we integrated the Bullet physics library [67] in our simulation framework. Figure 7 shows a teacup that contains a two-way coupled rigid rubber duck that has a rest density of 500 kg m^{-3}. As fluid is filled into the cup, the rubber duck begins to rise. We have simulated the scene for ten seconds. It consists of up to 3.57 million fluid particles and 1.25 million boundary particles. The particle spacing is 3 mm. Our DFSPH solver requires a total of 13.22 iterations on average per simulation step with the adaptive time step being 0.28 ms on average. The total average computation time per simulation step is 1.112 s whereof computing the boundary pressures takes 34.33 ms.

6.6. Washing Machine

Our proposed method is particularly appropriate for fast-moving and complex boundaries. This is indicated in Fig. 8 where we simulate a washing machine that contains seven two-way coupled rigid spheres with different radii. The washing drum is animated and contains holes, i.e. the fluid drains. The particle spacing is 2 cm and the scene consists of up to 2.04 million fluid particles and 1.18 million boundary particles. We use an adaptive time step with an average of 0.5 ms. Overall, with our approach the average computation time is 409 ms per simulation step whereof computing the boundary pressures takes 17.6 ms. The average iteration count is 4.42. Due to instabilities at the fast-moving boundary, pressure mirroring requires a time step that is half as large compared to our MLS extrapolation.

	average per time step Δt							
	iterations				computation time			
Δt	Mirroring	SPH extrapol.	MLS extrapol.	ext. PPE	Mirroring	SPH extrapol.	MLS extrapol.	ext. PPE
0.25 ms	4.0	4.0	4.0	4.0	116 ms	120 ms	123 ms	128 ms
0.50 ms	5.7	5.7	5.6	5.7	186 ms	192 ms	192 ms	205 ms
1.00 ms	12.0	12.2	11.8	12.0	448 ms	467 ms	462 ms	494 ms
1.50 ms	14.9	16.0	14.9	16.6	559 ms	618 ms	587 ms	684 ms

Table 1: *Comparison of the different boundary handling schemes using different time steps for the Breaking Dam scenario.*

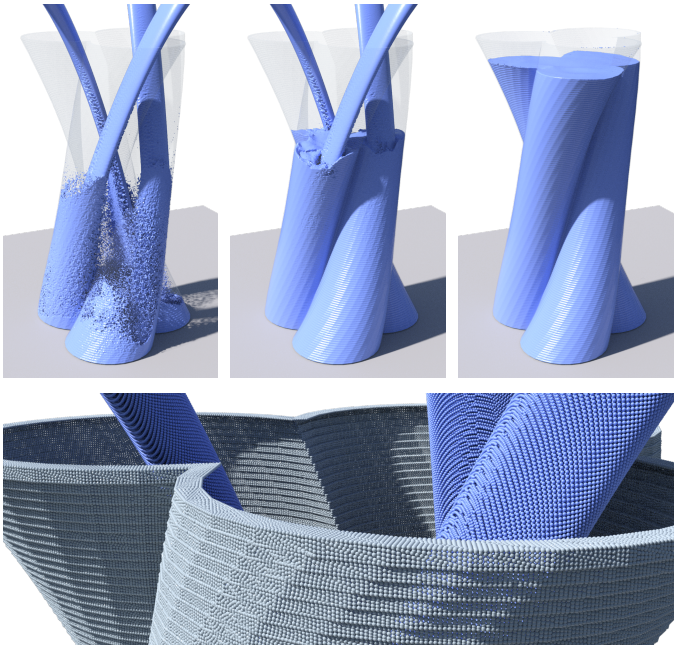


Figure 6: *Vase scenario with up to* 13.33 *million fluid particles. The bottom image shows a closeup, visualizing the complex boundary geometry and the particles.*

Figure 7: *Our boundary handling processes complex geometries with reduced artifacts and more efficiently compared to previous methods.*

This results in a speed-up of factor 1.8 compared to Akinci et al. [4]. This speed-up factor is particularly remarkable considering the fact that [4] typically works for time steps that correspond to rather large CFL numbers.

6.7. Glasses

In another experiment we compare our MLS-based approach to Pressure Boundaries [6]. For this, we use the scene that is shown in Fig. 9. The particle spacing is 1.5 mm and the scene consists of up to 26 million fluid particles and 8.2 million boundary particles. Please note that due to the small particle size, the water depth gets very high and challenging as fluid is filled into the glasses. As a result, the time step size is 0.16 ms. The average number of divergence-free iterations is 2 while the average number of density-invariant iterations is 5.45. Due to the overhead of solving a linear system for the pressure at boundary particles, the total average computation time per simulation step of Pressure Boundaries is higher than for our approach (our: 8.20 s vs. Pressure Boundaries: 8.46 s).

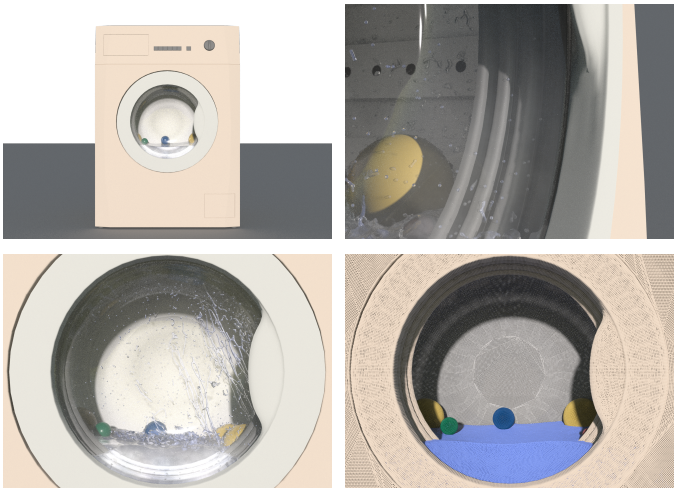


Figure 8: *Animated washing machine with two-way coupled rigid spheres.*

Figure 9: *Glasses scenario with various complex boundary shapes and up to* 26 *million fluid particles simulated with our approach.*

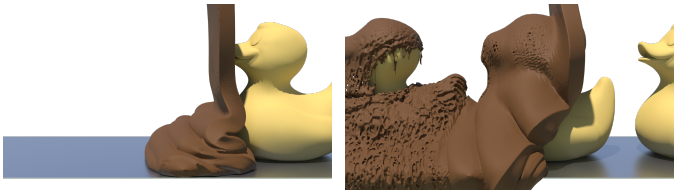


Figure 10: *Highly viscous fluid scene with up to* 10.1 *million fluid particles.*

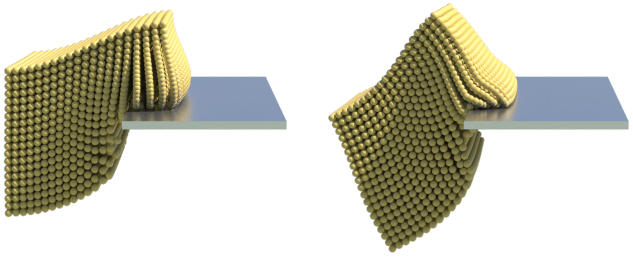


Figure 11: *Comparison of the different discretizations of the velocity Laplacian. On the left, the Laplacian is discretized with Eq.* (25) *[62] while on the right, it is discretized with Eq.* (24) *[25].*

6.8. Highly Viscous Fluids

In recent years the simulation of highly viscous fluids with SPH has become popular, e.g.[22, 3, 23, 25]. As indicated by Peer et al. [21], the handling of boundaries is not straightforward in case of iteratively computed implicit viscosity formulations. We therefore present scenarios to show that our MLS boundary handling scheme is also applicable to such highly viscous fluids. We implemented the approach of Weiler et al. [25], i.e. we use the implicit discretization

$$\boldsymbol{v}_f(t+\Delta t) = \boldsymbol{v}_f(t) + \Delta t \frac{\mu}{\rho_f(t)} \nabla^2 \boldsymbol{v}_f(t+\Delta t) \tag{23}$$

for the integration of the viscous forces and compute the Laplacian of the velocity as

$$\nabla^2 \boldsymbol{v}_f = 2(d+2) \sum_{f_j} V_{f_j} \frac{\boldsymbol{v}_{ff_j} \cdot \boldsymbol{x}_{ff_j}}{\|\boldsymbol{x}_{ff_j}\|^2 + 0.01h^2} \nabla W_{ff_j} \,. \tag{24}$$

We solve the resulting linear system with PCG with the 3×3 block pre-conditioner. However, we want to note that for radial symmetrical SPH kernels, like the cubic spline kernel [37], the pre-conditioner matrix is symmetric. Therefore, we only compute and store six coefficients of the matrix instead of nine to improve the computation time and to reduce the required memory amount of the viscosity solver.

6.8.1. Chocolate Ducks

In this scene, we simulated three ducks that are coated with chocolate. We used a particle spacing of 3 mm, $\mu = 2000\,\mathrm{N\,s\,m^{-2}}$, a rest density of $1325\,\mathrm{kg\,m^{-3}}$ and a fixed time step of 0.4 ms. The scene is illustrated in Fig. 10 and consists of up to 10.1 million fluid particles and 1 million boundary particles. The average number of iterations for the viscosity solver is 20 while the total computation time of one simulation step was 3.99 s on average.

6.8.2. Discretization of the Velocity Laplacian

There are many possibilities to discretize the Laplacian of the velocity with SPH, e.g. [62, 15, 37]. One option, that we used to simulate the non-highly viscous materials, was described by Morris et al. [62]:

$$\nabla^2 \boldsymbol{v}_f = 2(d+2) \sum_j V_j \frac{\nabla W_{fj} \cdot \boldsymbol{x}_{fj}}{\|\boldsymbol{x}_{fj}\|^2 + 0.01h^2} \boldsymbol{v}_{fj} \,. \tag{25}$$

Although Eq. (24) looks very similar to Eq. (25), the resulting pair-wise viscous force acts in a different direction: In Eq. (24) it acts in the direction of the SPH gradient whereas in Eq. (25) it acts in the direction of the relative velocity between the two particles. This difference motivated us to compare the two discretizations in the context of an implicit formulation for highly viscous fluids. Please note that for Eq. (25) the pre-conditioner is simply a scalar value instead of a 3×3 matrix.

Figure 11 shows that the two formulations behave differently. For this comparison, we used a cube-shaped viscous material in a free-fall setting. As the fluid hits the surface, it starts to rotate with the discretization Eq. (24) but not with Eq. (25). Please refer to the accompanying video to assess the differences in the dynamics.

7. Conclusion and Future Work

MLS pressure extrapolation at boundaries reduces artifacts at fluid-solid interfaces which can improve the performance of the pressure computation in iterative solvers. We have shown that pressure mirroring, SPH extrapolation and Pressure Boundaries suffer from artificial viscosity which is not the case for the proposed MLS extrapolation. We have also shown that the reduced velocity artifacts in our boundary handling can positively influence the pressure computation time for challenging scenarios. It is particularly remarkable that our local MLS approach can compete with the globally formulated Pressure Boundaries in terms of stability and time step size. Furthermore, we showed

			average	
scene	fluid particles	particle spacing h	time step size Δt	computation time per simulation step
Rotating Sphere	0.417 million	50 mm	1 ms	0.097 s
Breaking Dam	2.95 million	8 mm	1 ms	1.106 s
Vase	13.33 million	20 mm	0.47 ms	3.70 s
Teacup	3.57 million	3 mm	0.28 ms	1.112 s
Washing Machine	2.04 million	20 mm	0.5 ms	0.409 s
Glasses	26 million	1.5 mm	0.16 ms	8.20 s
Chocolate Ducks	10.1 million	3 mm	0.4 ms	3.99 s

Table 2: *Measurements for the scenarios simulated with our MLS boundary handling.*

that our boundary handling scheme can also be applied to highly viscous fluids.

As one of the next steps, we plan to investigate properties of the MLS gradient estimation for other purposes, e.g. the computation of the pressure acceleration at fluid particles.

Acknowledgments

The vase model is courtesy of Eckerput at https://www.cgtrader.com and is licensed under Royalty Free License. The cup model is courtesy of nerosoft at https://www.cgtrader.com and is licensed under Royalty Free License. The saucer model is courtesy of trapdormi at https://www.cgtrader.com and is licensed under Royalty Free License. The model of the washing machine is courtesy of vikinger at https://www.cgtrader.com and is licensed under Royalty Free License. The rubber duck is courtesy of willie at www.thingiverse.com and is licensed under Creative Commons - Public Domain Dedication license. The models of the glasses are courtesy of leighiria at https://www.cgtrader.com and are licensed under Royalty Free License.

References

[1] Solenthaler, B, Pajarola, R. Predictive-corrective Incompressible SPH. ACM Transactions on Graphics 2009;28(3):40:1–40:6.

[2] Ihmsen, M, Cornelis, J, Solenthaler, B, Horvath, C, Teschner, M. Implicit incompressible SPH. IEEE Transactions on Visualization and Computer Graphics 2014;20(3):426–435.

[3] Bender, J, Koschier, D. Divergence-Free SPH for Incompressible and Viscous Fluids. IEEE Transactions on Visualization and Computer Graphics 2017;23(3):1193–1206.

[4] Akinci, N, Ihmsen, M, Akinci, G, Solenthaler, B, Teschner, M. Versatile Rigid-fluid Coupling for Incompressible SPH. ACM Transactions on Graphics 2012;31(4):62:1–62:8.

[5] Adami, S, Hu, XY, Adams, NA. A generalized wall boundary condition for smoothed particle hydrodynamics. Journal of Computational Physics 2012;231(21):7057 – 7075.

[6] Band, S, Gissler, C, Ihmsen, M, Cornelis, J, Peer, A, Teschner, M. Pressure Boundaries for Implicit Incompressible SPH. ACM Transactions on Graphics 2018;37(2):14:1–14:11. Presented at SIGGRAPH 2018.

[7] Band, S, Gissler, C, Peer, A, Teschner, M. MLS Pressure Extrapolation for the Boundary Handling in Divergence-Free SPH. In: Virtual Reality Interactions and Physical Simulations. Eurographics Association. ISBN 978-3-03868-059-8; 2018,.

[8] Kenney, J, Keeping, E. Mathematics of Statistics Part I. Van Nostrand; 1956.

[9] Nealen, A. An as-short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. `http://www.nealen.de/projects/mls/asapmls.pdf`; 2004.

[10] He, X, Liu, N, Li, S, Wang, H, Wang, G. Local Poisson SPH For Viscous Incompressible Fluids. Computer Graphics Forum 2012;31(6):1948–1958.

[11] Macklin, M, Müller, M. Position Based Fluids. ACM Transactions on Graphics 2013;32(4):104:1–104:12.

[12] Ihmsen, M, Orthmann, J, Solenthaler, B, Kolb, A, Teschner, M. SPH Fluids in Computer Graphics. In: Eurographics (State of the Art Reports). 2014,.

[13] Stam, J, Fiume, E. Depicting Fire and Other Gaseous Phenomena Using Diffusion Processes. In: Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques. ISBN 0-89791-701-4; 1995, p. 129–136.

[14] Desbrun, M, Gascuel, MP. Smoothed particles: A new paradigm for animating highly deformable bodies. In: Computer Animation and Simulation. Springer; 1996, p. 61–76.

[15] Müller, M, Charypar, D, Gross, M. Particle-based Fluid Simulation for Interactive Applications. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ISBN 1-58113-659-5; 2003, p. 154–159.

[16] Takahashi, T, Dobashi, Y, Nishita, T, Lin, MC. An Efficient Hybrid Incompressible SPH Solver with Interface Handling for Boundary Conditions. Computer Graphics Forum 2018;37(1):313–324.

[17] Müller, M, Solenthaler, B, Keiser, R, Gross, M. Particle-based Fluid-fluid Interaction. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ISBN 1-59593-198-8; 2005, p. 237–244.

[18] Solenthaler, B, Pajarola, R. Density Contrast SPH Interfaces. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ISBN 978-3-905674-10-1; 2008, p. 211–218.

[19] Ren, B, Li, C, Yan, X, Lin, MC, Bonet, J, Hu, SM. Multiple-fluid sph simulation using a mixture model. ACM Transactions on Graphics 2014;33(5):171:1–171:11.

[20] Alduán, I, Tena, A, Otaduy, MA. DYVERSO: A versatile multi-phase position-based fluids solution for VFX. In: Computer Graphics Forum; vol. 36. Wiley Online Library; 2017, p. 32–44.

[21] Peer, A, Ihmsen, M, Cornelis, J, Teschner, M. An implicit viscosity formulation for SPH fluids. ACM Transactions on Graphics 2015;34(4):114.

[22] Takahashi, T, Dobashi, Y, Fujishiro, I, Nishita, T, Lin, MC. Implicit Formulation for SPH-based Viscous Fluids. Computer Graphics Forum 2015;34(2):493–502.

[23] Peer, A, Teschner, M. Prescribed Velocity Gradients for Highly Viscous SPH Fluids with Vorticity Diffusion. IEEE Transactions on Visualization and Computer Graphics 2017;23(12):2656–2662.

[24] Barreiro, H, García-Fernández, I, Alduán, I, Otaduy, MA. Conformation constraints for efficient viscoelastic fluid simulation. ACM Transactions on Graphics 2017;36(6):221.

[25] Weiler, M, Koschier, D, Brand, M, Bender, J. A Physically Consistent Implicit Viscosity Solver for SPH Fluids. Computer Graphics Forum 2018;37(2).

[26] Keiser, R, Adams, B, Gasser, D, Bazzi, P, Dutre, P, Gross, M. A unified Lagrangian approach to solid-fluid animation. In: Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics. 2005, p.

125–148.
[27] Solenthaler, B, Schläfli, J, Pajarola, R. A Unified Particle Model for Fluid-Solid Interactions. Computer Animation and Virtual Worlds 2007;18(1):69–82.
[28] Peer, A, Gissler, C, Band, S, Teschner, M. An Implicit SPH Formulation for Incompressible Linearly Elastic Solids. Computer Graphics Forum 2018;37(6):135–148.
[29] Monaghan, JJ. Simulating Free Surface Flows with SPH. Journal of Computational Physics 1994;110(2):399–406.
[30] Adams, B, Pauly, M, Keiser, R, Guibas, LJ. Adaptively Sampled Particle Fluids. ACM Transactions on Graphics 2007;26(3).
[31] Becker, M, Teschner, M. Weakly Compressible SPH for Free Surface Flows. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation. Eurographics Association. ISBN 978-1-59593-624-0; 2007, p. 209–217.
[32] Chorin, AJ. Numerical Solution of the Navier-Stokes Equations. Mathematics of Computation 1968;22(104):745–762.
[33] Shao, S, Lo, EY. Incompressible SPH method for simulating Newtonian and non-Newtonian flows with a free surface. Advances in Water Resources 2003;26(7):787 – 800.
[34] Cummins, SJ, Rudman, M. An SPH Projection Method. Journal of Computational Physics 1999;152(2):584–607.
[35] Purcell, TJ, Donner, C, Cammarano, M, Jensen, HW, Hanrahan, P. Photon Mapping on Programmable Graphics Hardware. In: ACM SIGGRAPH/Eurographics Conference on Graphics Hardware. Eurographics Association. ISBN 1-58113-739-7; 2003, p. 41–50.
[36] Hu, XY, Adams, NA. An incompressible multi-phase sph method. Journal of Computational Physics 2007;227(1):264–278.
[37] Monaghan, JJ. Smoothed Particle Hydrodynamics. Reports on Progress in Physics 2005;68(8):1703.
[38] Ihmsen, M, Akinci, N, Gissler, M, Teschner, M. Boundary Handling and Adaptive Time-stepping for PCISPH. In: Virtual Reality Interactions and Physical Simulations. ISBN 978-3-905673-78-4; 2010,.
[39] Müller, M, Schirm, S, Teschner, M, Heidelberger, B, Gross, M. Interaction of Fluids with Deformable Solids: Research Articles. Computer Animation and Virtual Worlds 2004;15(3-4):159–171.
[40] Monaghan, J, Kajtar, J. SPH particle boundary forces for arbitrary boundaries. Computer Physics Communications 2009;180(10):1811 – 1820.
[41] Becker, M, Tessendorf, H, Teschner, M. Direct Forcing for Lagrangian Rigid-Fluid Coupling. IEEE Transactions on Visualization and Computer Graphics 2009;15(3):493–503.
[42] Colagrossi, A, Landrini, M. Numerical Simulation of Interfacial Flows by Smoothed Particle Hydrodynamics. Journal of Computational Physics 2003;191(2):448–475.
[43] Yildiz, M, Rook, R, Suleman, A. SPH with the multiple boundary tangent method. International Journal for Numerical Methods in Engineering 2009;77(10):1416–1438.
[44] Schechter, H, Bridson, R. Ghost SPH for Animating Water. ACM Transactions on Graphics 2012;31(4):61:1–61:8.
[45] Ott, F, Schnetter, E. A modified SPH approach for fluids with large density differences. In: ArXiv Physics e-prints. 2003, p. 3112.
[46] Akinci, N, Cornelis, J, Akinci, G, Teschner, M. Coupling Elastic Solids with SPH Fluids. Computer Animation and Virtual Worlds 2013;24(3-4):195–203.
[47] Band, S, Gissler, C, Teschner, M. Moving Least Squares Boundaries for SPH Fluids. In: Virtual Reality Interactions and Physical Simulations. Eurographics Association. ISBN 978-3-03868-032-1; 2017,.
[48] Dilts, GA. Moving-least-squares-particle hydrodynamics I: Consistency and stability. International Journal for Numerical Methods in Engineering 1999;44(8):1115–1155.
[49] Alexa, M, Behr, J, Cohen-Or, D, Fleishman, S, Levin, D, T. Silva, C. Computing and Rendering Point Set Surfaces. IEEE Transactions on Visualization and Computer Graphics 2003;9(1):3–15.
[50] Müller, M, Keiser, R, Nealen, A, Pauly, M, Gross, MH, Alexa, M. Point Based Animation of Elastic, Plastic and Melting Objects. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation. Eurographics Association. ISBN 3-905673-14-2; 2004, p. 141–151.
[51] Bilotta, G, Russo, G, Hérault, A, Negro, CD. Moving least-squares corrections for smoothed particle hydrodynamics. Annals of Geophysics 2011;54(5).
[52] Huber, M, Eberhardt, B, Weiskopf, D. Boundary Handling at Cloth-Fluid Contact. Computer Graphics Forum 2015;34(1):14–25.
[53] Fujisawa, M, Miura, KT. An Efficient Boundary Handling with a Modified Density Calculation for SPH. Computer Graphics Forum 2015;34(7):155–162.
[54] Koschier, D, Bender, J. Density Maps for Improved SPH Boundary Handling. In: ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ACM. ISBN 978-1-4503-5091-4; 2017, p. 1:1–1:10.
[55] Cornelis, J, Bender, J, Gissler, C, Ihmsen, M, Teschner, M. An optimized source term formulation for incompressible SPH. The Visual Computer 2018;34:1–11.
[56] Rosswog, S. SPH Methods in the Modelling of Compact Objects. Living Reviews in Computational Astrophysics 2015;1(1).
[57] Press, WH, Teukolsky, SA, Vetterling, WT, Flannery, BP. Numerical Recipes 3rd Edition: The Art of Scientific Computing. 3 ed.; New York, NY, USA: Cambridge University Press; 2007. ISBN 0521880688, 9780521880688.
[58] Pauly, M, Keiser, R, Adams, B, Dutré, P, Gross, M, Guibas, LJ. Meshless Animation of Fracturing Solids. ACM Transactions on Graphics 2005;24(3):957–964.
[59] Ihmsen, M, Akinci, N, Becker, M, Teschner, M. A Parallel SPH Implementation on Multi-Core CPUs. In: Computer Graphics Forum; vol. 30. Wiley Online Library; 2011, p. 99–112.
[60] Gissler, C, Band, S, Peer, A, Ihmsen, M, Teschner, M. Generalized drag force for particle-based simulations. Computers & Graphics 2017;69:1–11.
[61] Akinci, N, Akinci, G, Teschner, M. Versatile Surface Tension and Adhesion for SPH Fluids. ACM Transactions on Graphics 2013;32(6):182:1–182:8.
[62] Morris, JP, Fox, PJ, Zhu, Y. Modeling Low Reynolds Number Incompressible Flows Using SPH. Journal of Computational Physics 1997;136(1):214–226.
[63] Bender, J, Koschier, D, Kugelstadt, T, Weiler, M. Turbulent Micropolar SPH Fluids with Foam. IEEE Transactions on Visualization and Computer Graphics 2018;PP:1–1.
[64] Pheatt, C. Intel® Threading Building Blocks. Journal of Computing Sciences in Colleges 2008;23(4):298–298.
[65] FIFTY2 Technology, . PreonLab. `www.fifty2.eu`; 2018.
[66] Side Effects Software, . Houdini. `www.sidefx.com`; 2018.
[67] Coumans, Erwin, . The bullet physics library. `www.bulletphysics.org`; 2018.