

Incompressible SPH Fluids with Two-Way Coupled Rigid Bodies

Julian Karrer

Faculty of Engineering
Department of Computer Science
Supervised by Prof. Dr.-Ing. Matthias Teschner

universität freiburg

Contents

1	Introduction	2
2	Smoothed Particle Hydrodynamics	6
2.1	Navier-Stokes Equations	6
2.2	SPH Discretization	8
3	Incompressible Fluid Solver	13
3.1	Discretization of the Navier-Stokes Equations	13
3.2	The Pressure Poisson equation	15
3.3	Implicit Incompressible SPH	17
3.4	Solver Variations and Degrees of Freedom	24
3.4.1	Divergence-Free Source Term	24
3.4.2	Performance Optimizing Metropolis Time Step Selection	26
4	Two-Way Coupled Rigid Bodies	33
4.1	Particle-Based Robust Boundary Representation	33
4.1.1	Sampling and Initialization	33
4.1.2	Dynamic and Procedural Boundaries	36
4.2	Two-Way Coupled Rigid Body Solver	37
4.2.1	Exact Mass Moments of Watertight Meshes	40
5	Visualization	44
5.1	Surface Reconstruction	44
5.2	Spray, Foam and Bubble Generation	47
6	Analysis	53
6.1	Methodology	53
6.2	Performance Scaling in Scene Complexity	56
6.3	Performance Comparison of Solvers	58
6.4	Stability Comparison of Solvers	59
7	Conclusion	62
Appendices		63
A	Fixed Radius Neighbour Search	63
B	Exact Inertia Tensor Terms for Tetrahedra	66
Bibliography		68

INTRODUCTION

The physically based simulation of fluids is an undeniably relevant field of study with diverse applications from engineering and Computational Fluid Dynamics to Computer Graphics and the photorealistic synthesis of images and video for films, commercials and games. At their intersection, solvers based on the Smoothed Particle Hydrodynamics or SPH discretization are being developed that use a Lagrangian framework especially convenient for otherwise challenging scenes involving free-surface fluid flows, deformable solids, highly viscous materials, highly complex boundary geometries and multi-material interactions [1]. These advances have spawned commercial software used in both computer-aided engineering and in animation [1].

This report outlines a complete implementation of a robust, fully GPU-parallel particle based solver for incompressible fluid flow problems with two-way coupled rigid bodies based on a Smoothed Particle Hydrodynamics discretization, or SPH for short, implemented in the Taichi programming language [2]. An Implicit Incompressible SPH pressure solver is used to solve the pressure Poisson equation and enforce incompressibility within an arbitrarily small, user defined error threshold, guaranteeing a high simulation quality without perceivable oscillations, drift or other artefacts. The solver is efficient enough to make simulations with high spatial resolution and correspondingly high particle counts well into the millions feasible on a single machine with consumer hardware. It can handle close to arbitrary, complex boundary geometry, procedurally moving and intersecting boundaries and can accurately calculate the dynamics of rigid bodies specified from arbitrary triangular meshes. Diffuse particles representing spray, foam and air bubbles can be generated in a physically motivated way to enhance visual realism and animate large-scale dynamic scenes such as the one shown in Figure 1.1, while also being able to handle high-resolution, smaller-scale scenes like the one shown in Figure 1.2.

The report gives a conceptual, high-level overview of each topic before diving deeper into implementation-relevant details. It is structured along the three main components required to realize simulations such as the one shown in Figure 1.1 or the one in Figure 1.2: an incompressible SPH fluid solver, a corresponding boundary handling scheme and rigid body solver and finally a pipeline for visualization of the simulation results.

Chapter 2 is concerned with the theoretical underpinning of the fluid simulation, discussing the Navier-Stokes equations that describe the dynamics of the fluid flow problem and the Smoothed Particle Hydrodynamics discretization scheme employed to numerically solve it.

These concepts are then used in chapter 3 to derive an incompressible SPH fluid solver that solves the pressure Poisson equation or PPE for short, using a matrix-free relaxed Jacobi solver to guarantee incompressibility and uphold the continuity equation. A particular focus in this chapter is a comprehensive derivation of the diagonal element of the system matrix posed by the PPE, which is crucial for a correct pressure solver implementation but rarely discussed in the literature to the presented level of detail. Chapter 3 also presents pressure solver variations with differing source terms that are considered state-of-the-art [4, 5] and are assessed for their relative performance in chapter 6. Said analysis makes use of a proposed controller that uses a metaheuristic to dynamically adjust the time step size used in the simulation to maximize performance, thereby closing a critical free parameter and making performance measurements more comparable while also increasing overall simulation efficiency, which is described in subsection 3.4.2.

The following chapter 4 discusses a particle-based boundary representation that uses the previously derived pressure solver to enforce impenetrability, can handle nearly arbitrary and highly complex boundary geometry robustly and can deal with moving and intersecting boundaries such as the wave-generator used in the making of Figure 1.1. An iterative scheme is proposed to accurately initialize the effective masses of boundary particles and Poisson disk sampling is employed to generate a low-discrepancy set of particles on the surface of the boundary geometry. The same boundary handling scheme is used to sample the surfaces of dynamic rigid bodies described in section 4.2, which are simulated and two-way



Figure 1.1: A maritime scene with over 1.5 million fluid particles, highly complex boundary geometry represented by over 3 million particles, a wave-generating, sinusoidally moving procedural boundary and a large, two-way coupled battleship floating dynamically on an ocean of incompressible fluid with an average density error of less than 0.1% is simulated by the solver described in this report, highlighting its versatility, robustness and ability to accurately produce highly realistic fluid simulations. Up to 1.6 million diffuse particles representing spray, foam and bubbles are visible at the shoreline, enhancing the visual realism and dynamicity of the scene. The fluid surface is accurately reconstructed and finally rendered using [3, Blender] to produce the above image. Full videos of the scenes used in this report are available online (<https://www.youtube.com/playlist?list=PL9sgPh0msS271Sp0r7aQ8-PJpn9vhxVJ>)



Figure 1.2: In contrast to the physically large-scale scene in Figure 1.1, this high resolution simulation with 1.3 million fluid particles and 3 million boundary particles is concerned with a small-scale setting of a red liquid being poured into a glass. Even on these scales, the incompressible fluid simulation yields realistic and visually plausible results, despite some effects that are more relevant on smaller scales, such as surface tension, not being explicitly modelled in this report. Diffuse particles are not generated here, since they are more prevalent in scenes of larger scales. Full videos of the scenes used in this report are available online (<https://www.youtube.com/playlist?list=PL9sgPh0msS271Sp0r7aQ8-PJpn9vhvxVJ>)

coupled to the fluid solver from chapter 3. The rigid body solver is enhanced by using quaternions as the representation of orientation in a way that minimizes numerical drift in the orthonormalization of the derived rotation matrices. The mass moments and properties required for this solver, such as the centre of mass and inertia tensor, are calculated using a method that is analytically exact for watertight triangular meshes of uniform density using a rather uncommon signed-volumes approach described in subsection 4.2.1, with further details and a more convenient formula than that commonly found in the literature discussed in Appendix B.

Having described all necessary components for creating a realistic fluid simulation with two-way coupled rigid bodies, the visualization of the results is discussed in chapter 5. Surfaces are reconstructed from volumetric particle data and smoothed using a specialized procedure in section 5.1. Physically motivated generation of diffuse particles representing spray, foam and bubbles is then discussed, enhancing the visual realism of the simulated scenes in a post-processing step.

Finally, chapter 6 analyses the behaviour of the presented pressure solvers using the time step controller from subsection 3.4.2, with a particular focus on how performance and the optimal time step size vary with scene complexity and how varying source terms and the warm-starting of pressure solvers affects relative performance and stability.

SMOOTHED PARTICLE HYDRODYNAMICS

Before moving on to the more concrete construction of an incompressible fluid solver in chapter 3, the underlying governing equations and the Smoothed Particle Hydrodynamics or SPH scheme used to discretize them are discussed in the following. This chapter gives a high-level overview of the equations that describe incompressible, Newtonian fluid flow and the general approach taken in this report to simulate it numerically.

2.1 Navier-Stokes Equations

In order to develop a numerical solver for fluid flow problems, the respective governing equations must first be discussed. In the case of linearly viscous, incompressible fluid flow, the framework of continuum mechanics yields a set of equations called the **NAVIER-STOKES EQUATIONS**. Since the focus on this report is on the implementation of the solver, these equations are only briefly outlined in the following and the interested reader is referred to more extensive works such as by [6, Anderson] or a previous report authored by the present writer [7] for a more thorough derivation.

Lagrangian Continuum Mechanics

Since the fluid flow problems in question occur not at a microscopic scale, such as concerning the interaction of individual molecules, but at a macroscopic scale instead, continuum mechanics is a suitable framework for the formulation of the governing equations. It assumes that the fluid is a continuum that is under deformation, wherein field quantities such as density, velocity etc. are convected with continuum elements as they deform, and makes use of the **MATERIAL DERIVATIVE** to apply the rules of calculus to these field quantities [6]:

$$\frac{D}{Dt} := \underbrace{\frac{\partial}{\partial t}}_{\text{local derivative}} + \underbrace{(\vec{v} \cdot \nabla)}_{\text{convective derivative}} \quad (2.1)$$

The first term represents the local derivative, or instantaneous rate of change of the quantity with respect to the continuum element, while the second describes the rate of change of the quantity at a fixed position due to the deformation of the continuum and consequently the convection of the continuum element with the velocity field \vec{v} . The frame of reference for each continuum element that is convected along with the velocity field is precisely the one in which the second term becomes zero, yielding the Lagrangian or so-called non-conservation form of the equations instead of the Eulerian form of the problem in which a static frame of reference is chosen [6]. We choose the Lagrangian representation in the following since the continuum itself is later discretized into particles that are convected with the fluid flow and only quantities at particle positions will be of interest. An Eulerian discretization of the space within which the continuum exists could be chosen instead, but may be less suitable in application to free-surface flows with complex and dynamic boundary geometry [1].

Governing Equations

The governing equations can be derived from laws of conservation applied to the continuum. The first such conservation law is the **CONSERVATION OF MASS** $\frac{Dm}{Dt} = 0$, which can be transformed using either the Reynolds Transport theorem or the divergence theorem [7] into the continuity equation of the form:

CONTINUITY EQUATION

$$\frac{D\rho}{Dt} + \rho (\nabla \cdot \vec{v}) = 0 \quad (2.2)$$

for incompressible flows, which has to hold across the entire volume integral over the fluid domain [6]. Since the flow is incompressible, the density of any fluid element must remain constant across a wide range of pressures, as will be assumed is the case of water. This assumption leads to two equivalent constraints posed by the continuity equation [8]:

1. $\frac{D\rho}{Dt} = 0$, the density of any fluid element cannot change
2. $\nabla \cdot \vec{v} = 0$ the velocity field of the fluid must be divergence free

These expressions can be used to derive alternative source terms for the pressure solver in subsection 3.4.1.

The second conservation law to be considered is the **CONSERVATION OF MOMENTUM**. Since momentum is conserved, the material derivative of momentum across the fluid must be balanced by conservative forces acting on the fluid. These forces can be decomposed into stresses \vec{t} acting in normal direction $\hat{\vec{n}}$ on the surface of each fluid element on one side, and external body forces per unit mass \vec{b}^{ext} such as gravity acting on the entire volume. Inserting the definition of the **CAUCHY-STRESS TENSOR** \mathbb{T} (sometimes referred to as $\bar{\sigma}$) where $\mathbb{T}\hat{\vec{n}} = \vec{t}$ and applying the divergence theorem as well as the continuity equation to the resulting integral equation yields that for every fluid element the following must hold [8]:

CAUCHY MOMENTUM EQUATION

$$\rho \frac{D\vec{v}}{Dt} = \nabla \cdot \mathbb{T} + \rho \vec{b}^{ext} \quad (2.3)$$

The stress tensor \mathbb{T} can further be decomposed into the isotropic component $\mathbb{T} = -p\mathbb{1} + \mathbb{V}$ where p is the hydrostatic pressure and \mathbb{V} is the deviatoric stress or viscous stress tensor [9]. This component can be modelled in terms of the symmetric rate of deformation tensor \mathbb{D} , which in sum with its antisymmetric counterpart, the spin tensor \mathbb{W} , makes up the velocity gradient [8]. Assuming incompressible flow and a linearly viscous or Newtonian fluid, one can obtain [1, 9]:

CONSTITUTIVE RELATION

$$\mathbb{T} = -p\mathbb{1} + \mu \left(\nabla \vec{v} + (\nabla \vec{v})^T \right) \quad (2.4)$$

where μ is the dynamic viscosity, which is related to the kinematic viscosity ν by $\nu = \mu/\rho$. For strongly enforced incompressibility, the pressure value p can be interpreted as a Lagrange multiplier chosen such that the momentum equation satisfies the continuity equation [1], which will be the motivation for the iterative pressure solver discussed in chapter 3. Inserting the constitutive relation into the Cauchy momentum equation, rearranging and applying the Theorem of Schwarz [9] one can finally obtain the Navier-Stokes momentum equation for incompressible, Newtonian fluids in Lagrangian form:

NAVIER STOKES MOMENTUM EQUATION

$$\frac{D\vec{v}}{Dt} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \vec{v} + \vec{b}^{ext} \quad (2.5)$$

Apart from the momentum and continuity equations, the conservation of energy can be used to derive the energy equation which can be used to describe the thermal processes within fluid flows [6], however this is not usually as relevant in the application to Computer Graphics and is neglected in the following. As such, the continuity and momentum equations are the main point of focus of this report, which will simply be referred to as the Navier-Stokes equations and solved numerically.

2.2 SPH Discretization

In order to numerically solve the Navier-Stokes equations, it is necessary to discretize the equations in space and time to make them tractable for simulation. As previously discussed, a Lagrangian simulation will be conducted, meaning the continuum itself is discretized into fluid elements of some mass or volume, at which field quantities such as density, velocity and pressure must be determined and which are advected with the velocity field according to Newton's laws of motion.

Smoothed Particle Hydrodynamics, or SPH for short, is an interpolation scheme which can be used to reconstruct a continuous and differentiable field from quantities sampled at individual points that each represent some fluid volume, which we refer to as particles. This representation and the SPH scheme allow field quantities to be computed at any point of interest and at particle locations in particular, making it suited for realizing a Lagrangian simulation.

Derivation

The SPH method can be derived from a relaxation of the unit impulse or Dirac δ distribution, which is defined as [10]:

DIRAC δ -DISTRIBUTION

$$\int_{-\infty}^{\infty} \delta(x) dx = 1 \quad \text{normalization} \quad (2.6)$$

$$x \neq 0 \implies \delta(x) = 0 \quad (2.7)$$

and analogously for higher dimensional arguments $\delta(\vec{x})$. The δ -distribution describes point-like samples in space, as indicated by the sampling property $f(x)\delta(x - x_s) = f(x_s)\delta(x - x_s)$ for sample positions x_s from which the sifting property can be obtained [10]:

$$f(x) = \int_{-\infty}^{\infty} f(x_s)\delta(x - x_s) dx \quad (2.8)$$

Which means that by the convolution theorem [1]:

$$f(\vec{x}) = \int_{\Omega} f(\vec{x}')\delta(\vec{x} - \vec{x}') dV' = (f * \delta)(\vec{x}) \quad (2.9)$$

where $*$ is the convolution operator on functions. While this serves as a precise description of a perfect sampling of a field, it is unsuited for numerical simulation since important properties such as differentiability are not given for a finite number of samples. Instead, two approximations are made to arrive at the SPH scheme:

1. The δ -distribution is approximated by a kernel function W with desirable properties
2. The integral in Equation 2.8 is approximated by a sum over the finite fluid samples.

The SPH sum at a sample position \vec{x}_i then reads:

$$f(\vec{x}_i) = \int_{\Omega} f(\vec{x}_j)\delta(\vec{x}_i - \vec{x}_j) dV_j \quad (2.10)$$

$$\approx \int_{\Omega} f(\vec{x}_j)W(\vec{x}_i - \vec{x}_j, \hbar) dV_j \quad \delta(\vec{x}_{ij}) \approx W(\vec{x}_{ij}, \hbar) \quad (2.11)$$

$$\approx \sum_{j \in \mathcal{N}_i} f(\vec{x}_j)W(\vec{x}_{ij}, \hbar)V_j \quad \text{finite sample set} \quad (2.12)$$

where $\mathcal{N}_i = \{j : |\vec{x}_{ij}| < \hbar\}$ for $\vec{x}_{ij} := \vec{x}_i - \vec{x}_j$ is the set of neighbour samples of i within some radius \hbar , which is finite for compactly supported W . The computation of fixed-radius neighbour sets is described in detail Appendix A. Denoting quantities $f(\vec{x}_i)$ as f_i etc., the volume V_j associated with sample j can be expressed as $\frac{m_j}{\rho_j}$ for short, resulting in the standard SPH sum:

STANDARD SPH SUM

$$\langle f_i \rangle = \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} f_j W_{ij} \quad (2.13)$$

Consistency and Normalization

A Taylor series error analysis of the standard SPH discretization $\langle f_i \rangle$ reveals that there are two conditions that must be fulfilled for 0th- and 1st-order consistency respectively to be achieved, namely [11]:

CONSISTENCY CRITERIA

$$\sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} W_{ij} \stackrel{!}{=} 1 \quad 0^{\text{th}} \text{ order consistency} \quad (2.14)$$

$$\sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} (\vec{x}_j - \vec{x}_i) W_{ij} \stackrel{!}{=} \vec{0} \quad 1^{\text{st}} \text{ order consistency} \quad (2.15)$$

Note that the low asymptotic error order for arbitrary samplings appears to have questionable practical relevance [1] - perhaps because there appears to be some trade-off in Lagrangian methods between the linear error order of a discretization and the quality of the physical conservation properties and sampling isotropy caused by discrete operators, as discussed for example by [11, Price]. If desired however, the conditions can be fulfilled for normalized, symmetric kernels by multiplying a scalar to ensure Equation 2.14 holds and solving a 3×3 matrix inversion per particle to enforce Equation 2.15 [1, 11]. Similar consistency conditions exist for SPH approximations of differential operators [11].

Kernel Function

The kernel function is parametrized by $\vec{x}_{ij} = \vec{x}_i - \vec{x}_j$ for samples i, j and a kernel support radius \hbar . We denote $W_{ij} := W(\vec{x}_{ij}, \hbar)$ for short. For the approximation to be consistent, W must converge to δ as the support radius goes to zero, meaning the kernel must fulfil the criteria [1]:

$$\int_{\Omega} W(\vec{x}_{ij}, \hbar) = 1 \quad \text{normalization} \quad (2.16)$$

$$\lim_{\hbar \rightarrow 0} W(\vec{x}_{ij}, \hbar) = \delta(\vec{x}_{ij}) \quad \text{Dirac-}\delta \text{ condition} \quad (2.17)$$

Further useful properties include [1]:

$$|\vec{x}_{ij}| > \hbar \implies W(\vec{x}_{ij}, \hbar) \quad \text{compact support} \quad (2.18)$$

$$W(\vec{x}_{ij}, \hbar) = W(\vec{x}_{ji}, \hbar) \quad \text{spherical symmetry} \quad (2.19)$$

$$W(\vec{x}_{ij}, \hbar) > 0 \quad \text{positivity} \quad (2.20)$$

$$W(\vec{x}_{ij}, \hbar) \in C^2 \quad \text{twice continuously differentiable} \quad (2.21)$$

since compact support can decrease the number of non-zero contributions for non-degenerate cases from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ in the number N of samples and is vital for computational efficiency. Spherical symmetry and normalization together enable methods for ensuring first order consistency [1]. Positivity is vital for approximations of quantities such as absolute pressure, mass density etc. that must not be negative, while C^2 -continuity is convenient for the discretization of second-order partial differential equations [1].

Due to many highly convenient signal-theoretical properties of the Gaussian kernel, such as not overshooting approximated step-functions, not creating new zero-crossings of second derivatives and having optimal spatial and frequency locality in the sense of the uncertainty inequality [12], many commonly employed kernel functions mimic a truncated Gaussian distribution using polynomial splines that are efficient to evaluate. In d dimensions, due to symmetry and compact support to \hbar , all kernel functions can be represented with respect to a one-dimensional shape function w and its derivative $\partial_q w$ as [13]:

$$W(\vec{x}_{ij}, \hbar) = \frac{\alpha(d)}{\hbar^d} w(q) \quad (2.22)$$

$$\nabla W(\vec{x}_{ij}, \hbar) = \frac{\alpha(d)}{\hbar^{d+1}} \frac{\vec{x}_{ij}}{|\vec{x}_{ij}|} \partial_q w(q) \quad (2.23)$$

$$(2.24)$$

where $q = \frac{|\vec{x}|}{\hbar} \in [0, 1]$ and $\alpha(d)$ depends only on dimensionality. As in much of the literature, the Cubic

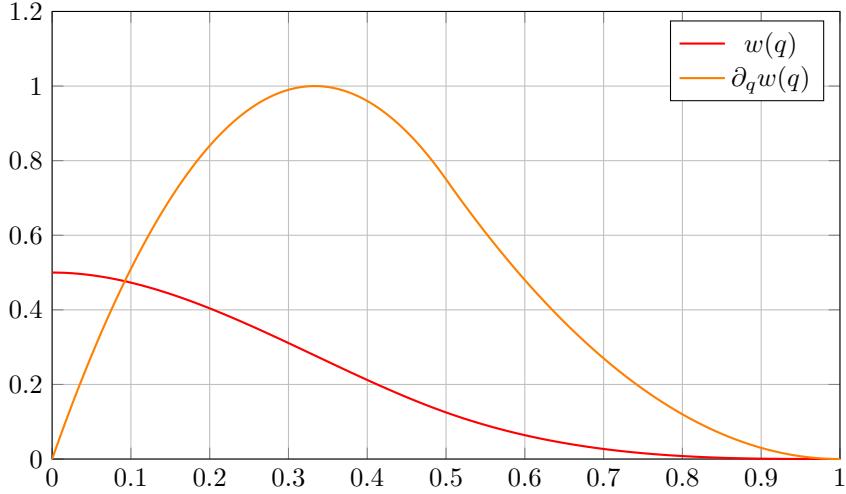


Figure 2.1: The shape function $w(q)$ that defines the Cubic Spline kernel function and its first derivative are plotted for all values of $q \in [0, 1]$ that yield non-zero $w(q)$. The function is compactly supported. Since the kernel function W is spherically symmetric, this one-dimensional function of the scaled distance is sufficient to describe W in any dimensionality, only the scaling factors α have to be adjusted.

Spline kernel is employed in this report [13]:

$$w(q) = (1 - q)_+^3 - 4(1/2 - q)_+^3 \quad (2.25)$$

$$\partial_q w(q) = -3(1 - q)_+^2 - 12(1/2 - q)_+^2 \quad (2.26)$$

$$\alpha(3) = 16/\pi \quad (2.27)$$

where $(x)_+ := \max(0, x)$. These functions are illustrated in Figure 2.1. As is often recommended [1], a kernel support $\hbar = 2h$ of two times the initial spacing between particles h is used for all calculations.

Differential Operators

The gradient of the kernel function can be used directly to approximate differential operators such as gradient, divergence and rotation on vector fields, the least straightforward example amongst them perhaps being:

$$\langle \nabla \vec{f}_i \rangle_{\otimes} = \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} \vec{f}_j \otimes \nabla W_{ij} \quad (2.28)$$

where \otimes is the dyadic product [1]. Such direct discretization can lead to poor approximation quality in practice, especially for low support radii \hbar , which has lead to the development of alternative discretizations with useful properties.

One such discretization is the **DIFFERENCE FORMULA**, which subtracts the first Taylor series term of the discretization error to restore 0th order consistency (see Equation 2.14) even for suboptimal samplings, yielding a more accurate approximation for the gradient of a scalar field [1, 11]:

DIFFERENCE FORMULA

$$\langle \nabla f_i \rangle_- = \langle \nabla f_i \rangle - f_i \langle \nabla 1 \rangle = \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} (f_j - f_i) \nabla W_{ij} \quad (2.29)$$

Another useful alternative discretization for the gradient of a scalar field is the **SYMMETRIC FORMULA**, which can be derived from the discrete Lagrangian specifically such that symmetric forces that conserve linear and angular momentum can be obtained from its application, for example to the pressure field [1, 11]:

SYMMETRIC FORMULA

$$\langle \nabla f_i \rangle_{\parallel} = \rho_i \sum_{j \in \mathcal{N}_i} m_j \left(\frac{f_i}{\rho_i^2} + \frac{f_j}{\rho_j^2} \right) \nabla W_{ij} \quad (2.30)$$

While this formula is generally less accurate than the difference formula, the conservation of momenta can be a benefit that in practice outweighs lacking guarantees of first or even 0th order consistency for arbitrary samplings [1].

Lastly, second derivatives must be discretized, particularly the Laplace operator necessary to describe dissipative terms such as viscosity [11]. However, since the second derivative of $w(q)$ varies more intensely across the kernel support, a direct discretization would have to be exceptionally well sampled to yield accurate results - meaning the approximation quality for low support radii, sparse or high-discrepancy samplings would suffer¹. Instead, discretizations that rely on the kernel gradient approximation are typically preferred [1]. Amongst them, the following discretization of the Laplace operator is especially useful since for a divergence-free field $\vec{f}(\vec{x})$ it can be used to derive forces that obey momentum conservation: each summand is antisymmetric in i, j and all vectorial terms are projected onto the line spanned by the positions of samples i and j [1]:

LAPLACE OPERATOR DISCRETIZATION

$$\langle \nabla^2 \vec{f}_i \rangle_{\Delta} = 2(d+2) \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} \frac{\vec{f}_{ij} \cdot \vec{x}_{ij}}{|\vec{x}_{ij}|^2} \nabla W_{ij} \quad (2.31)$$

where again d is the dimensionality of the problem. With this, all necessary SPH discretizations for the following chapters are available.

Particle Deficiency and Free Surfaces

One peculiarity of the SPH discretization scheme is the behaviour at free surfaces, where the fluid phase discretized into particles meets the non-discretized phase representing for example air in a liquid simulation or vacuum in simulations of gases. Since sums over fluid particles are used to interpolate the density field ρ , a lack of samples in the unrepresented phase can be thought of as a perfectly dense sampling with particles that have zero density. Similarly, other field quantities such as pressure, velocity etc. are also implicitly assumed to be zero in the absence of fluid particles, which is relevant for the clamping of pressure values to positive values in subsubsection 3.3.

This leads to a phenomenon called **PARTICLE DEFICIENCY** at the free surface, where only a section of each kernel support radius is filled with neighbouring fluid particles, leading to an underestimation of the density ρ at the surface. While the implicit assumption of $\rho = 0$ in the non-represented phase is not a bad assumption where $\rho_{air} \ll \rho_{H_2O}$ etc., it leads the average density across particles $1/N \sum_i^N \langle \rho_i \rangle$, to depend on the geometry of the fluid, as well as always being an underestimation of the 'true' average density in the fluid. This should be accounted for when average field quantities are used, such as in the convergence criteria of pressure solvers in subsubsection 3.3. An illustration of this phenomenon is shown in Figure 2.2.

¹It might help this intuition to consider the sample positions to be quasi-randomly distributed values X . There exist numerous examples of upper bounds on the variance of a function of a random variable in terms of some measure of how intensely that function varies. As an example, consider random values $Z_i := f(X_i)$, $\mathbb{V}[Z] < \infty$ in terms of some function $f : \chi \mapsto \mathbb{R}$ for a measurable χ and independently distributed sample positions X_i from any distribution. If f has the *bounded difference property* $\sup_{x'} |w(x) - w(x')| \leq c$ for a non-negative bound c , the Efron-Stein inequality implies $\mathbb{V}[Z] \leq \frac{c}{4}$ (see [14, Boucheron et al.] for more such bounds). Since the first derivative of our shape function $\partial_q w(q)$ is more tightly bound than the second $\partial_q^2 w(q)$, achieving a lower c , the variance of its sampling has a tighter bound, where lower variance results in better estimator performance.

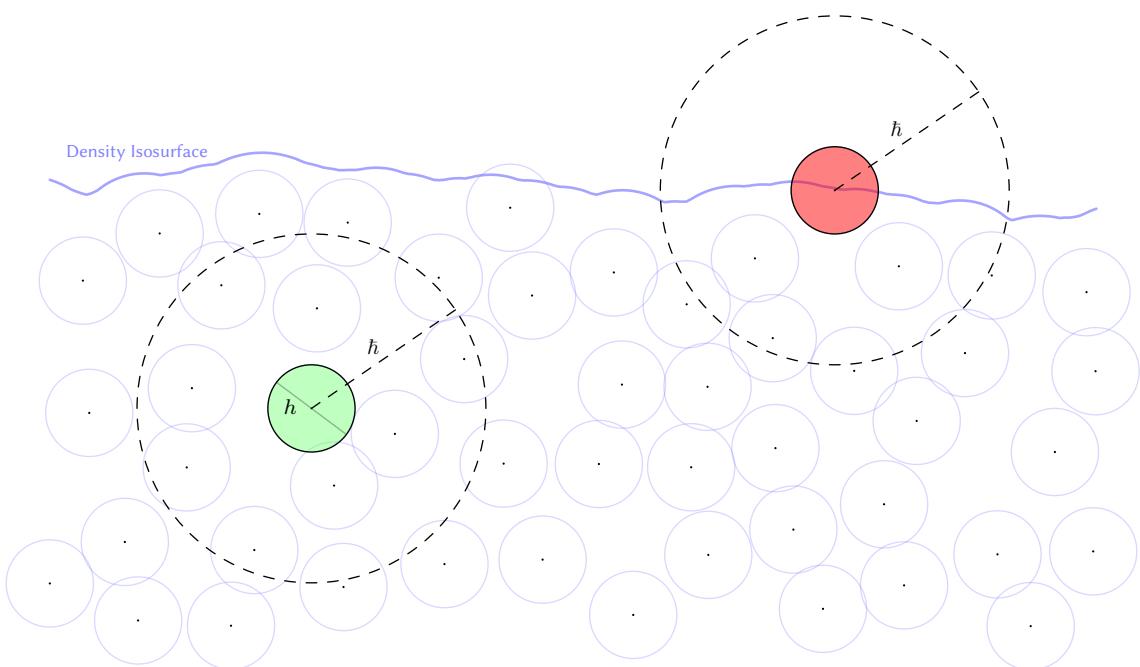


Figure 2.2: The particle deficiency phenomenon is illustrated in two dimensions, where particles positions are shown as dots with a surrounding circle of a diameter equal to the expected particle spacing h (Note that SPH particles are *not* modelled as hard spheres but some volume of unspecified shape - only the isosurfaces of the spherically symmetric kernel function justifies this artistic choice). The particle highlighted in green lies within the fluid and has a well-sampled neighbourhood, allowing for relatively accurate estimation of the local density field through the SPH interpolation scheme. This very scheme however results in an underestimation at the free surface as highlighted for the particle shown in red which is barely inside the fluid volume, the SPH-estimated density of which however is below rest density since the lack of samples in the upper region of the kernel support is implicitly assumed to represent a density of zero, which is smoothed out across the free surface. While this effect diminishes with increasing spatial resolution and accordingly decreasing kernel support radius, leading to better resolution of the sharp discontinuity in the density field at the ideal fluid surface, the phenomenon can still lead to undesired clustering at the free surface and incorrect average densities across particles if not handled carefully, especially at lower resolutions.

INCOMPRESSIBLE FLUID SOLVER

With the SPH method described in chapter 2, the tools to discretize and numerically solve the Navier-Stokes equations outlined in section 2.1 are available. This chapter focuses more concretely on implementing a fluid solver and facing the challenge of ensuring the continuity equation is upheld by an incompressible fluid. This leads to the pressure Poisson equation or PPE for short, which is derived in section 3.2 and discretized in subsubsection 3.3. The discretization is chosen in accordance with the implicit Incompressible SPH algorithm or IISPH for short, which is discussed in subsubsection 3.3 and uses a relaxed Jacobi solver outlined in subsubsection 3.3 to solve the PPE. The diagonal element of the system matrix representing the pressure Poisson equation is derived in particular detail, since it is of central importance to the implementation of the fluid solver but often only briefly discussed in the literature.

Alternative source terms and variations of the solver are available, which are then discussed in section 3.4.

3.1 Discretization of the Navier-Stokes Equations

The Navier-Stokes equations for incompressible Newtonian fluids in Lagrangian form that we consider are the continuity equation Equation 2.2 and momentum equation Equation 2.5. These form a system of equations where the momentum equation provides means of calculating the acceleration $\vec{a}_i = \frac{D\vec{v}_i}{Dt}$ necessary to compute particle trajectories using Newton's second law, while the continuity equation can be seen as a constraint on the former [1], ensuring incompressibility. Firstly, recall the momentum equation for some particle i :

$$\underbrace{\vec{a}_i}_{\text{total acceleration}} = -\underbrace{\frac{1}{\rho_i} \nabla p_i}_{\text{pressure acceleration } \vec{a}_i^p} + \underbrace{\nu \nabla^2 \vec{v}_i}_{\text{viscosity acceleration } \vec{a}_i^\nu} + \underbrace{\vec{b}^{ext}}_{\text{external accelerations eg. } \vec{g}} \quad (3.1)$$

Each of these terms can now be discretized using the SPH formulas from chapter 2. For previously discussed reasons, the viscous term may be approximated in a symmetric and momentum-conserving fashion using the discrete Laplace operator as defined in Equation 2.31:

$$\langle \nu \nabla^2 \vec{v}_i \rangle_\Delta = \nu \langle \nabla^2 \vec{v}_i \rangle_\Delta = 2\nu(d+2) \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} \frac{\vec{v}_{ij} \cdot \vec{x}_{ij}}{|\vec{x}_{ij}|^2 + \epsilon} \nabla W_{ij} \quad (3.2)$$

Note that the additional small term ϵ in the denominator has been added to prevent numerical instabilities and divisions by zero if the distance between two particles was close to zero - in this implementation $\epsilon = 0.01h^2$ in the expected particle spacing h was chosen. Since viscous forces tend to be dominated by pressure forces for large-scale, low viscosity or high-Reynolds simulations as this report is subject to, the pressure acceleration is the major contributor to the particles' momenta, making the symmetric formula in Equation 2.30 a robust choice to obtain physically accurate results that retain conservation properties [1]:

$$\left\langle -\frac{1}{\rho_i} \nabla p_i \right\rangle_\parallel = -\frac{1}{\rho_i} \langle \nabla p_i \rangle_\parallel = -\sum_{j \in \mathcal{N}_i} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij} \quad (3.3)$$

The external accelerations \vec{b}^{ext} in this case are simply the gravitational acceleration \vec{g} with $|\vec{g}| = 9.81 \text{ m/s}^2$.

The density ρ_i in the above approximations is a scalar quantity and can be discretized using the standard SPH sum from Equation 2.13 [1]:

$$\langle \rho_i \rangle = \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} \rho_j W_{ij} = \sum_{j \in \mathcal{N}_i} m_j W_{ij} \quad (3.4)$$

Since the mass m_i is set at the start of the simulation, ν, \vec{g} are given and initial \vec{x}_i, \vec{v}_i are known, the only quantity in the momentum equation yet to be accounted for is the pressure field p . The system could be closed by employing a state equation relating the pressure directly to the density field using some stiffness parameter k through the ideal gas equation $p = k\rho$ [15], the Murnaghan equation of state [16] as used in Weakly Compressible SPH [17] $p = k((\rho/\rho_0)^\gamma - 1)$ or similar equations of state. Since for the incompressible fluids simulated in this report the constraints imposed by the continuity equation must be strongly enforced, pressure is instead computed iteratively by solving a pressure Poisson equation or PPE as outlined in subsubsection 3.3.

In the following, quantities f_i will generally refer to the SPH-discretized fields for brevity.

Time Discretization

After spatial discretization using the SPH scheme, the second order partial differential equation Equation 2.5 is turned into an ordinary differential equation [1] that must be discretized in time.

For this purpose, time is discretized into time steps Δt and a numerical integration scheme is applied to obtain particle trajectories by updating particle positions and velocities. In Computer Graphics literature, a highly prevalent scheme [1, 17, 18, 19, 4] to this end is symplectic or semi-implicit Euler time integration:

$$\vec{v}(t + \Delta t) = \vec{v}(t) + \Delta t \vec{a}(t) \quad (3.5)$$

$$\vec{x}(t + \Delta t) = \vec{x}(t) + \Delta t \vec{v}(t + \Delta t) \quad (3.6)$$

There is a trade-off in choosing a time step size: large time steps mean more progress per solver step and can lead to greater efficiency, while time steps that are too large might lead to instability and can actually cause lower overall performance, especially for iterative solvers, as will be discussed in subsection 3.4.2. A common upper bound on Δt is the Courant-Friedrichs-Lowy or CFL condition, which states that particles should not move further than some fraction $\lambda \in]0, 1]$ of their size or spacing h within one time step, which for a global adaptive time step size implies [1]:

$$\Delta t \leq \lambda \frac{h}{\max_i |\vec{v}_i|} \quad (3.7)$$

with a common choice [4, 20] being $\lambda = 0.4$. Alternative formulations based on the maximum accelerating forces [20, 21] or more elaborately derived formulas that differ per kernel function and pressure solver [22] exist.

Initial conditions

The numerical time integration scheme propagates a solution to the governing equations forwards in time, calculating a new valid state of the fluid at $t + \Delta t$ from a current one at time t . To seed this recursion, an initial state at t_0 must be provided. Perhaps the simplest and most common method of creating such an initial state is to define a volume filled with liquid in the simulation domain and sample it with fluid particles on a regular, square grid with spacing h , initial zero velocities, and a uniform rest volume $V_0 = 1/h^d$, or equivalently mass $m_0 = \rho_0/h^d$ in d dimensions. This method poses two challenges in particular: one is sampling an arbitrary fluid volume, the other is preventing artefacts due to the anisotropic, regular sampling and ensuring that the initial stages of the simulation run smoothly.

- The first problem of sampling some closed volume with fluid particles is trivial when that volume is a box. More generally, a watertight mesh that bounds the fluid volume can be provided, where candidate fluid particle positions are sampled on a regular grid of spacing h within the axis-aligned bounding box of said mesh. For each such candidate position, since the mesh is watertight, a single ray cast operation is sufficient to determine whether the position is within the fluid volume or not: according to the Jordan Curve theorem, if any ray originating at the candidate position intersects the bounding mesh an odd number of times, the point lies within the volume [23] and a fluid particle is instantiated, otherwise the candidate is discarded¹. Additionally, all candidate positions closer than h to a boundary particle as described in section 4.1 can be discarded to prevent boundary penetration and inconveniently high pressure values at t_0 .

¹The implementation of this report uses the `trimesh.ray.ray_pyembree.RayMeshIntersector` implementation of a ray cast operator [24]

- The second problem may be alleviated by applying a jitter on initial positions in conjunction with non-uniform particle masses [7]. Sampling particles of uniform mass causes a density gradient due to the particle deficiency phenomenon illustrated in Figure 2.2 towards the boundary of the fluid volume, which can in turn result in an erroneous pressure gradient that causes the fluid to initially expand towards the boundary, causing an ‘explosion’. The mass of each particle can instead be chosen such that uniform rest density ρ_0 is achieved everywhere at t_0 by assigning a higher rest volume to particles at boundaries, ensuring $\forall i : \rho_i(t_0) = \rho_0 \implies p_i(t_0) = 0$. It is easy to see that this can be implemented in a manner consistent with the employed SPH density approximation by iterating:

$$m_i^0 = \frac{1}{h^d} \quad (3.8)$$

$$\langle \rho_i \rangle^l \leftarrow \sum_{j \in \mathcal{N}_i} m_j^l W_{ij} \quad (3.9)$$

$$m_i^{l+1} \leftarrow m_i^l \cdot \frac{\rho_0}{\langle \rho_i \rangle^l} \quad (3.10)$$

until convergence, which in this implementation was defined as when the average density error $\frac{1}{N} \sum_i \rho_i - \rho_0$ changes by less than $10^{-10} \frac{\text{kg}}{\text{m}^3}$ compared to the previous iteration. Stricter bounds are unproblematic since the cost is a one-off preprocessing computation. Slow convergence can be seen as an indicator for an ill-defined scene specification.

- This initialization to rest density enables another possible improvement in the form of jittered initial positions. A regular grid in conjunction with a kernel with support radius \hbar that is an integer multiple of the grid spacing h was found to be prone to aliasing artefacts in the initial stages of simulation. Even just a jitter of $\Delta \vec{x} \sim 0.01h \cdot \mathcal{N}(\vec{\mu} = \vec{0}, \vec{\sigma} = \vec{1})$ was found in this implementation to relieve numerical issues and anisotropic behaviour, while the uniform density initialization prevents the jitter from resulting in erroneous initial pressure forces. Slightly non-uniform particle masses are also suspected to improve the amorphousness of the particle sampling, possibly reducing numerical viscosity [7].

3.2 The Pressure Poisson equation

With the discretization of the Navier-Stokes equations from section 3.1, a simple equation-of-state based solver such as Weakly Compressible SPH or WCSPH for short, could be implemented directly. However, such a solver that computes pressure accelerations at particles only locally, from information about only immediately neighbouring particles at each time step, can exhibit noticeable oscillations as the complexity of the scene, namely the maximum height of a column of fluid supported by the pressure, increases.

As such a column of particles is uniformly accelerated by gravity but faces an impenetrable boundary at its bottom, a counteracting pressure gradient has to build up. Since pressure is only computed locally and the numerical speed of sound is low, particles will continue to fall, increasing measured density and thereby pressure beyond the linear gradient necessary to counteract gravity, until this superlinear gradient pushes the top of the column up beyond its hydrostatic resting position and repeating, resulting in unrealistic oscillations. In other words, it is hard to enforce incompressibility with such a solver: even if a large stiffness coefficient k is chosen to counteract this effect and increase the numerical speed of sound, a correspondingly small time step size Δt would be required to keep the system numerically stable, decreasing efficiency [18].

Alternatively, the differential density deviation $\frac{D}{Dt} \rho$ could be penalized, however while no oscillations would occur, numerical errors would instead accumulate into volume drift [1], leading to arguably worse results where the fluid volume would irrecoverably decrease over time and accurate boundary conditions are challenging to implement. The misconception that volume oscillations are inherent to Lagrangian methods such as SPH fluid simulation can instead be interpreted as a deliberate choice enabling accurate simulation of free-surface flows where volume drift would lead to noticeable errors.

To tackle the issue of strongly enforcing incompressibility, the solution of the Navier-Stokes equation can be decomposed into sequential sub-problems, namely into the computations of pressure accelerations and non-pressure accelerations such as caused by gravity and viscosity [1]. The idea is to integrate the less problematic sub-problem explicitly, while the dominating pressure accelerations can subsequently be implicitly determined to leave the fluid in a divergence-free or uncompressed predicted state at the next

time step [1]. This is also referred to as **OPERATOR SPLITTING** [1] and implemented here as computing the predicted velocity \vec{v}_i^* due to non-pressure accelerations:

$$\vec{v}_i^*(t) = \vec{v}(t) + \Delta t (\vec{g} + \nu \nabla^2 \vec{v}_i(t)) \quad (3.11)$$

before implicitly integrating the pressure acceleration \vec{a}_i^p as described in subsubsection 3.3. This equates to solving the **PRESSURE POISSON EQUATION**, or PPE for short, which will be derived and discretized in the following and is used to project the predicted particle velocities onto an uncompressed or divergence-free state.

Derivation of the Pressure Poisson Equation

Pressure computation is, in the case of inviscid flows, the stiff sub-problem in the partial differential equations to solve [1]. Pressure values that strongly enforce incompressibility, or in other words act as Lagrange multipliers to the continuity equation Equation 2.2 [1], should be computed efficiently. This can be done using iterative, global solvers that solve the Pressure Poisson equation, which asserts that the pressure forces do in fact enforce the continuity equation.

One way to derive the PPE is to focus on the density invariance constraint imposed by the continuity equation (see Constraint 1). A predicted velocity \vec{v}_i^* would result in positions \vec{x}_i^* that can be used to compute a predicted density ρ_i^* using, for example, Equation 4.7. The incompressibility constraint then means that:

$$\forall i : \rho_i(t + \Delta t) = \rho_0 \quad (3.12)$$

must hold, or that the pressure acceleration must cause a change to the predicted density that keeps the measured density $\rho_i(t + \Delta t)$ at the next time step at the rest density everywhere.

Updating particle positions to \vec{x}_i^* in a sub-step, recomputing the sets of neighbours that depend on those positions and using Equation 4.7 to compute ρ_i^* is not done in practice, since the recalculation of neighbours is computationally expensive. Instead, the predicted density can be obtained by numerically integrating the change in density $(D\rho/Dt)^*$ due to \vec{v}_i^* to estimate the density at the next time step [1]:

$$\rho^* = \rho + \Delta t \left(\frac{D\rho}{Dt} \right)^* \quad \text{approximate density at next time step} \quad (3.13)$$

$$\rho^* = \rho + \Delta t (-\rho \nabla \cdot \vec{v}^*) \quad \text{insert continuity equation 2.2} \quad (3.14)$$

$$\rho^* = \rho - \Delta t \rho \nabla \cdot \vec{v}^* \quad (3.15)$$

Equation 3.15 encodes the fact that a velocity field with negative divergence at a point will see an increase in density, while positive divergence of the velocity field means a decrease in mass density.

This same formula can be used to predict a density change ρ_p^* due to the pressure acceleration:

$$\rho_p^* = \rho - \Delta t \rho \nabla \cdot (\Delta t \vec{a}_p) \quad \text{Equation 3.15 using } \vec{v}_p^* = \Delta t \vec{a}_p \text{ instead of } \vec{v}^* \quad (3.16)$$

$$= \rho - \Delta t \rho \nabla \cdot \left(\Delta t \left(-\frac{1}{\rho} \nabla p \right) \right) \quad \text{Definition of } \vec{a}_p \text{ from Equation 3.1} \quad (3.17)$$

$$= \rho + \Delta t^2 \nabla \cdot (\nabla p) \quad \text{simplify} \quad (3.18)$$

$$\rho_p^* = \rho + \Delta t^2 \nabla^2 p \quad \nabla \cdot (\nabla p) = \nabla^2 p \quad (3.19)$$

Finally, one can assert that adding the predicted change in density due to the pressure acceleration $\rho_p^* - \rho$ to the predicted density due to non-pressure accelerations ρ^* should result in rest density ρ_0 at the next time step:

$$\rho(t + \Delta t) \stackrel{!}{=} \rho_0 \quad \text{Incompressibility Constraint 1, Equation 3.12} \quad (3.20)$$

$$\rho^* + (\rho_p^* - \rho) \stackrel{!}{=} \rho_0 \quad (3.21)$$

$$\rho^* \cancel{+} \rho_p^* + \Delta t^2 \nabla^2 p \cancel{+} \rho \stackrel{!}{=} \rho_0 \quad \text{insert Equation 3.19} \quad (3.22)$$

$$\Delta t \nabla^2 p \stackrel{!}{=} \frac{\rho_0 - \rho^*}{\Delta t} \quad \text{rearrange} \quad (3.23)$$

which is one form of the pressure Poisson equation [1]. Basically the exact same derivation can be used with constraint 2 derived from the continuity equation, namely a divergence free velocity field at the next

time step. Note the similarity to the derivation above:

$$\nabla \cdot (\vec{v}(t + \Delta t)) \stackrel{!}{=} 0 \quad \text{divergence free velocity constraint 2} \quad (3.24)$$

$$\nabla \cdot \left(\vec{v}^* - \Delta t \frac{1}{\rho} \nabla p \right) \stackrel{!}{=} 0 \quad \text{insert } \vec{v}_p^* = \Delta t \left(-\frac{1}{\rho} \nabla p \right) \quad (3.25)$$

$$\nabla \cdot \vec{v}^* - \Delta t \frac{1}{\rho} \nabla^2 p \stackrel{!}{=} 0 \quad \text{linearity of divergence} \quad (3.26)$$

$$\Delta t \nabla^2 p \stackrel{!}{=} \rho \nabla \cdot \vec{v}^* \quad \text{rearrange} \quad (3.27)$$

3.3 Implicit Incompressible SPH

The iterative solution of the Pressure Poisson Equation in the context of SPH-based fluid simulations leads to a family of methods commonly named **INCOMPRESSIBLE SPH** or ISPH for short. A few choices can be made regarding the implementation of such a fluid solver, such as:

1. how to discretize the Laplace operator in the PPE [25]
2. whether to use density invariance (Equation 3.23)[18], velocity divergence (Equation 3.27) or some combination of both terms [4, 26] as source terms for the solver
3. which numerical solver to use

This report takes the **IMPLICIT INCOMPRESSIBLE SPH** or IISPH method by [18, Ihmsen et al.] as its basis and answers these questions by discretizing the Laplacian using the discretized divergence of the pressure gradient, using the density invariant source term and solving the resulting system using a Relaxed Jacobi solver. Each of these components of the final pressure solver used in Algorithm 1 are outlined in the following.

Discretizing the Pressure Poisson Equation

The focus in the following lies on the density invariance source term, or in other words discretizing the PPE:

PRESSURE POISSON EQUATION

$$\Delta t^2 \nabla^2 p = \rho_0 - \rho^* \quad (3.28)$$

using the SPH scheme so that it can be solved numerically at each discrete particle location.

For this purpose, the following discretization of the divergence of a vector field, derived from the standard SPH sum (Equation 2.13) by pulling the density into the differential operator, is commonly employed [20, 27]:

$$\nabla \cdot \vec{f} = \frac{1}{\rho} \left(\nabla \cdot (\vec{f} \rho) - \vec{f} \cdot \nabla \rho \right) \quad \text{after [20, Monaghan 1992]} \quad (3.29)$$

$$\langle \nabla \cdot \vec{f} \rangle = \underbrace{\frac{1}{\rho_i} \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} (\rho_j \vec{f}_j) \cdot \nabla W_{ij}}_{\langle \nabla \cdot (\vec{f} \rho) \rangle} - \underbrace{\frac{1}{\rho_i} \vec{f}_i \cdot \sum_{j \in \mathcal{N}_i} m_j \nabla W_{ij}}_{\vec{f}_i \cdot \langle \nabla \rho \rangle} \quad \text{SPH sum (Equation 2.13)} \quad (3.30)$$

which simplifies to:

SPH DIVERGENCE DISCRETIZATION

$$\langle \nabla \cdot \vec{f} \rangle_{\nabla \cdot} = -\frac{1}{\rho_i} \sum_{j \in \mathcal{N}_i} m_j \vec{f}_{ij} \cdot \nabla W_{ij} \quad (3.31)$$

Using this, the source term $s_i = \rho_0 - \rho_i^*$ of the pressure Poisson equation can be discretized as:

$$\langle \rho_0 - \rho^* \rangle = \rho_0 - (\rho_i - \Delta t \rho_i \langle \nabla \cdot \vec{v}_i^* \rangle_{\nabla \cdot}) \quad \text{Definition of } \rho^* \text{ (Equation 3.15)}$$

(3.32)

$$= \rho_0 - \left(\rho_i + \Delta t \rho_i \left(\frac{1}{\rho_i} \sum_{j \in \mathcal{N}_i} m_j \vec{v}_{ij}^* \cdot \nabla W_{ij} \right) \right) \quad \text{apply } \nabla \cdot \text{ discretization (Equation 3.31)}$$

(3.33)

$$= \rho_0 - \rho_i - \Delta t \sum_{j \in \mathcal{N}_i} m_j \vec{v}_{ij}^* \cdot \nabla W_{ij} \quad \text{simplify}$$

(3.34)

which can be extended using the boundary handling of chapter 4 like in subsubsection 4.1.2 to:

DENSITY INVARIANCE SOURCE TERM

$$s_i = \langle \rho_0 - \rho^* \rangle = \rho_0 - \rho_i - \Delta t \sum_{j \in \mathcal{N}_i} m_j \vec{v}_{ij}^* \cdot \nabla W_{ij} - \Delta t \sum_{k \in \mathcal{B}_i} m_k \vec{v}_{ik}^* \cdot \nabla W_{ik} \quad (3.35)$$

where for static boundaries one can assume $\vec{v}_k^* = \vec{0}$ while for weakly coupled dynamic rigid bodies as described in section 4.2 $\vec{v}_k^* \approx \vec{v}_k(t)$ can serve as the best guess.

Secondly, the term $\Delta t^2 \nabla^2 p$ needs to be discretized. To this end, a discretization consistent with the computation of the pressure acceleration in Equation 4.8 should be chosen, such that the solver computes pressures that actually translate to the predicted, uncompressed state after \vec{a}_i^p has been computed from each p_i [1]:

$$\langle \Delta t^2 \nabla \cdot (\nabla p) \rangle = \Delta t^2 \langle \nabla \cdot (-\rho_i \vec{a}_i^p) \rangle_{\nabla \cdot} \quad \text{since } \vec{a}_i^p = -\frac{1}{\rho_i} \nabla p \text{ (Equation 3.1)} \quad (3.36)$$

$$= -\rho_i \Delta t^2 \langle \nabla \cdot (\vec{a}_i^p) \rangle_{\nabla \cdot} \quad \text{use linearity} \quad (3.37)$$

$$= \Delta t^2 \sum_{j \in \mathcal{N}_i} m_j \vec{a}_{ij}^p \cdot \nabla W_{ij} \quad \text{apply } \nabla \cdot \text{ discretization (Equation 3.31)} \quad (3.38)$$

which extends when considering boundary particles to a term:

$$\langle \Delta t^2 \nabla \cdot (\nabla p) \rangle = \Delta t^2 \sum_{j \in \mathcal{N}_i} m_j \vec{a}_{ij}^p \cdot \nabla W_{ij} + \Delta t^2 \sum_{k \in \mathcal{B}_i} m_k \vec{a}_i^p \cdot \nabla W_{ik} \quad (3.39)$$

with the simplifying assumption that pressure accelerations \vec{a}_k^p at boundary particles are zero.

Note that the discretization of the Laplace operator $\nabla^2 p$ using the discretization of the divergence in $\langle \nabla \cdot (\nabla p) \rangle$ is not the only possibility, and other discretizations such as the discrete Laplace operator in Equation 2.31 commonly used to compute viscosities, is also thinkable. For one comparison of the discretization used in [18, Implicit Incompressible SPH], which is presented here, to Equation 2.31 we refer to [25, Fürstenu et al.].

Solving the Discretized PPE

With the discretized pressure Poisson equation obtained, one can go about solving the resulting system of N equations at N particles for the unknown pressure values p_i . Many possible solvers for this problem exist, such as Conjugate-Gradient [18, as alternatively proposed by Ihmsen et al.], and more recently Nonsmooth Nonlinear Conjugate Gradient methods as suggested by [28, Probst and Teschner].

A relatively straightforward implementation that is robust, matrix-free, i.e. does not rely on explicitly constructing system matrices of prohibitive $\mathcal{O}(N^2)$ proportions, and which can be fully parallelized is achieved by a **RELAXED JACOBI SOLVER** as suggested in Implicit Incompressible SPH [18, by Ihmsen et al.].

Remember that the pressure Poisson equation for each of the N particles using the density invariance constraint as in Equation 3.23 is:

$$\langle \Delta t^2 \nabla^2 p_i^2 \rangle = \rho_i - \rho_i^* \quad (3.40)$$

where $\langle \Delta t^2 \nabla^2 p_i^2 \rangle$ can be discretized as seen in Equation 3.39. This can be interpreted as a set of N equations that can be written in matrix form as:

$$\mathbb{A}\vec{p} = \vec{s} \quad (3.41)$$

where \vec{s} is the vector of all source terms $s_i = \rho_i - \rho_i^*$ which are known from Equation 3.35 and \vec{p} is the vector of all unknown pressure values p_i . The system matrix \mathbb{A} then represents the discrete Laplace operator (with a factor of Δt^2) and is defined by Equation 3.39.

In order to solve this system of equations the Jacobi Solver in subsubsection 3.3 requires:

1. each of the products $(\mathbb{A}\vec{p})_i$. This is precisely the same as $\langle \Delta t^2 \nabla^2 p_i \rangle$ and is computed as stated in Equation 3.39 by choosing an SPH discretization of the Laplace operator.
2. the source terms s_i . This is also known, see Equation 3.35 for density invariance. Alternative source terms, such as the velocity divergence, could be used instead, resulting in Equation 3.70 discussed in subsection 3.4.1.
3. the diagonal element \mathbb{A}_{ii} . This is not yet known.

The next step is therefore finding the diagonal element of the discrete Laplacian so that one can solve for the unknown pressures.

Diagonal Element \mathbb{A}_{ii}

The diagonal element \mathbb{A}_{ii} of the discretized PPE is less obvious to compute, but thankfully there is a conceptually easy way to derive it from now known quantities: simply take the left-hand side of the i -th Pressure Poisson equation, i.e. the value $(\mathbb{A}\vec{p})_i$ that is known from Equation 3.39, and apply the partial derivative with respect to the i -th pressure value:

$$\mathbb{A}_{ii} = \frac{\partial}{\partial p_i} (\mathbb{A}\vec{p})_i \quad (3.42)$$

Intuitively, the result can be thought of as asking *how much does changing p_i affect the predicted error at particle i ?* Unfortunately, since the SPH-discretized expression $(\mathbb{A}\vec{p})_i$ is quite large, consisting of multiple SPH sums over particle neighbours, this partial derivative can be quite cumbersome to derive, so the pressure acceleration term \vec{a}_i^p within it is first rearranged into a more convenient form by factoring out all occurrences of p_i :

$$\vec{a}_i^p = - \sum_{j \in \mathcal{N}_i} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij} - \sum_{k \in \mathcal{B}_i} m_k \frac{p_i}{\rho_i^2} \nabla W_{ik} \quad \text{Equation 4.8}$$

$$(3.43)$$

$$= -p_i \frac{1}{\rho_i^2} \left(+ \sum_{j \in \mathcal{N}_i} m_j \nabla W_{ij} + \sum_{k \in \mathcal{B}_i} m_k \nabla W_{ik} \right) - \sum_{j \in \mathcal{N}_i} m_j \frac{p_j}{\rho_j^2} \nabla W_{ij} \quad \text{factor out } p_i, 1/\rho_i^2$$

$$(3.44)$$

$$= -p_i \frac{1}{\rho_i^2} (\vec{g}_{ij} + \vec{g}_{ik}) - \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \frac{p_j}{\rho_j^2} \nabla W_{ij} \quad \text{use definitions Equation 3.46}$$

$$(3.45)$$

where for brevity, the following definitions were used:

$$\vec{g}_{ij} = \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \nabla W_{ij} \quad (3.46)$$

$$\vec{g}_{ik} = \sum_{k \in \mathcal{B}_i} m_k \nabla W_{ik} \quad (3.47)$$

Note that since the kernel function W is differentiable and spherically symmetric (Equation 2.19, Equation 2.21), it follows that $\nabla W_{ii} = \vec{0}$, so the i -th term of the sum can be excluded and a sum over $\mathcal{N}_i \setminus \{i\}$ be used instead, which is convenient in the derivation since it implies $\frac{\partial}{\partial p_i} \vec{g}_{ij} = \frac{\partial}{\partial p_i} \vec{g}_{ik} = 0$.

With this, the larger expression $(\mathbb{A}\vec{p})_i$ can be stated as:

$$(\mathbb{A}\vec{p})_i = \Delta t^2 \sum_{j \in \mathcal{N}_i} m_j \vec{a}_{ij}^p \cdot \nabla W_{ij} + \Delta t^2 \sum_{k \in \mathcal{B}_i} m_k \vec{a}_i^p \cdot \nabla W_{ik} \quad \text{Equation 3.39}$$

$$\begin{aligned} &= \Delta t^2 \sum_{j \in \mathcal{N}_i} m_j \vec{a}_i^p \cdot \nabla W_{ij} - \Delta t^2 \sum_{j \in \mathcal{N}_i} m_j \vec{a}_j^p \cdot \nabla W_{ij} + \Delta t^2 \sum_{k \in \mathcal{B}_i} m_k \vec{a}_i^p \cdot \nabla W_{ik} \\ &\quad \text{split } \vec{a}_{ij}^p \end{aligned} \quad (3.49)$$

$$= \Delta t^2 \left(\underbrace{\sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \vec{a}_i^p \cdot \nabla W_{ij}}_{a_i :=} - \underbrace{\sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \vec{a}_j^p \cdot \nabla W_{ij}}_{b_i :=} + \underbrace{\sum_{k \in \mathcal{B}_i} m_k \vec{a}_i^p \cdot \nabla W_{ik}}_{c_i :=} \right) \quad \text{exclude } \nabla W_{ii} = \vec{0} \quad (3.50)$$

where each of the terms a_i, b_i, c_i need to be partially differentiated with respect to p_i to find the diagonal element:

$$\mathbb{A}_{ii} = \frac{\partial}{\partial p_i} (\mathbb{A}\vec{p})_i = \Delta t^2 \left(\frac{\partial}{\partial p_i} a_i + \frac{\partial}{\partial p_i} b_i + \frac{\partial}{\partial p_i} c_i \right) \quad (3.51)$$

First, the partial derivative of the term a_i can be computed as:

$$\begin{aligned} \frac{\partial}{\partial p_i} a_i &= \frac{\partial}{\partial p_i} \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \vec{a}_i^p \cdot \nabla W_{ij} \quad \text{Equation 3.50} \\ &= \frac{\partial}{\partial p_i} \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \left(-p_i \frac{1}{\rho_i^2} (\vec{g}_{ij} + \vec{g}_{ik}) - \underbrace{\sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \frac{p_j}{\rho_j^2} \nabla W_{ij}}_{\text{no } p_i, \text{ no contribution}} \right) \cdot \nabla W_{ij} \quad \text{insert Equation 3.45} \\ &= -\frac{1}{\rho_i^2} \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j (\vec{g}_{ij} + \vec{g}_{ik}) \cdot \nabla W_{ij} \quad \text{apply partial derivative} \end{aligned}$$

Since the dot product is linear and both \vec{g}_{ij} and \vec{g}_{ik} are the same in every summand, they can be factored out, simplifying the expression to:

$$\frac{\partial}{\partial p_i} a_i = -\frac{1}{\rho_i^2} (\vec{g}_{ij} + \vec{g}_{ik}) \cdot \vec{g}_{ij} \quad \text{apply } \vec{g}_{ij} \text{ definition} \quad (3.52)$$

Secondly, the derivation for $\frac{\partial}{\partial p_i} c_i$ is exactly analogous to $\frac{\partial}{\partial p_i} a_i$, only that indices $j \in \mathcal{N}_i \setminus \{i\}$ of fluid neighbours are replaced by indices $k \in \mathcal{B}_i$ for boundary neighbours, yielding:

$$\frac{\partial}{\partial p_i} c_i = -\frac{1}{\rho_i^2} (\vec{g}_{ij} + \vec{g}_{ik}) \cdot \vec{g}_{ik} \quad \text{analogous to as Equation 3.52} \quad (3.53)$$

Lastly, only the partial derivative $\frac{\partial}{\partial p_i} b_i$ remains to be computed, where \vec{a}_j^p is used:

$$\begin{aligned} \frac{\partial}{\partial p_i} b_i &= \frac{\partial}{\partial p_i} - \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \vec{a}_j^p \cdot \nabla W_{ij} \quad \text{Equation 3.50} \\ &= -\frac{\partial}{\partial p_i} \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \underbrace{\left(-p_j \frac{1}{\rho_j^2} (\vec{g}_{ji} + \vec{g}_{jk}) - \sum_{l \in \mathcal{N}_j \setminus \{j\}} m_l \frac{p_l}{\rho_l^2} \nabla W_{jl} \right)}_{\text{no } p_i, \text{ no contribution}} \cdot \nabla W_{ij} \quad \text{insert Equation 3.45} \\ &\quad = \vec{a}_j^p \end{aligned}$$

Since the acceleration \vec{a}_j^p at particle j is used here, the indices l are introduced to refer to neighbours l of each neighbour j of particle i . Note that these second-degree neighbours of particle i include the particle i itself:

$$i \in \bigcup_{j \in \mathcal{N}_i \setminus \{i\}} (\mathcal{N}_j \setminus \{j\}) \quad (3.54)$$

In other words, there is only one term in the nested sum that includes the pressure p_i and therefore contributes to the partial derivative with respect to p_i , namely the term where $l = i$. This means that applying the partial derivative leaves only a single term in the inner sum and the expression simplifies to:

$$\frac{\partial}{\partial p_i} b_i = -\frac{\partial}{\partial p_i} \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \underbrace{\left(-\sum_{l \in \mathcal{N}_j \setminus \{j\}} m_l \frac{p_l}{\rho_l^2} \nabla W_{jl} \right)}_{\text{only contributes for } l = i} \cdot \nabla W_{ij} \quad \text{see above} \quad (3.55)$$

$$= -\frac{\partial}{\partial p_i} \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \left(-m_i \frac{p_i}{\rho_i^2} \nabla W_{ji} \right) \cdot \nabla W_{ij} \quad \text{summand } l = i \text{ remains} \quad (3.56)$$

$$= -\frac{\partial}{\partial p_i} \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j m_i \frac{p_i}{\rho_i^2} \underbrace{\nabla W_{ij} \cdot \nabla W_{ij}}_{=|\nabla W_{ij}|^2} \quad -\nabla W_{ji} = \nabla W_{ij} \quad (3.57)$$

$$= -\frac{m_i}{\rho_i^2} \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j |\nabla W_{ij}|^2 \quad \text{apply partial derivative} \quad (3.58)$$

Finally, the partial derivatives $\frac{\partial}{\partial p_i} a_i$, $\frac{\partial}{\partial p_i} b_i$, $\frac{\partial}{\partial p_i} c_i$ can be reinserted into Equation 3.51 to obtain the diagonal element:

$$\begin{aligned} \mathbb{A}_{ii} &= \Delta t^2 \left(\frac{\partial}{\partial p_i} a_i + \frac{\partial}{\partial p_i} b_i + \frac{\partial}{\partial p_i} c_i \right) && \text{Equation 3.51} \\ &= \Delta t^2 \left(\underbrace{-\frac{1}{\rho_i^2} (\vec{g}_{ij} + \vec{g}_{ik}) \cdot \vec{g}_{ij}}_{\frac{\partial}{\partial p_i} a_i} - \underbrace{\frac{1}{\rho_i^2} (\vec{g}_{ij} + \vec{g}_{ik}) \cdot \vec{g}_{ik}}_{\frac{\partial}{\partial p_i} c_i} - \underbrace{\frac{m_i}{\rho_i^2} \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j |\nabla W_{ij}|^2}_{\frac{\partial}{\partial p_i} b_i} \right) && \text{reinsert} \\ &= -\frac{\Delta t^2}{\rho_i^2} \left(\underbrace{(\vec{g}_{ij} + \vec{g}_{ik}) \cdot \vec{g}_{ij} + (\vec{g}_{ij} + \vec{g}_{ik}) \cdot \vec{g}_{ik}}_{\text{binomial}} + \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_i m_j |\nabla W_{ij}|^2 \right) && \text{factor out } -\frac{1}{\rho_i^2} \end{aligned}$$

Since the dot product is a linear operator, the binomial formula and the identity $\vec{x} \cdot \vec{x} = |\vec{x}|^2$ can be applied to simplify:

$$(\vec{g}_{ij} + \vec{g}_{ik}) \cdot \vec{g}_{ij} + (\vec{g}_{ij} + \vec{g}_{ik}) \cdot \vec{g}_{ik} = |\vec{g}_{ij} + \vec{g}_{ik}|^2 \quad (3.59)$$

Reinserting the definitions of \vec{g}_{ij} , \vec{g}_{ik} from Equation 3.46 results in:

DIAGONAL ELEMENT \mathbb{A}_{ii}

$$\mathbb{A}_{ii} = -\frac{\Delta t^2}{\rho_i^2} \left(\left| \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \nabla W_{ij} + \sum_{k \in \mathcal{B}_i} m_k \nabla W_{ik} \right|^2 + \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_i m_j |\nabla W_{ij}|^2 \right) \quad (3.60)$$

The diagonal element depends only on the current positions, neighbour sets, masses and current densities of the particles, all of which do not change within the simulation step, meaning the diagonal element can be computed once per time step (after the current density ρ_i has been updated) and then reused in each iteration of the iterative solver described in subsubsection 3.3.

The diagonal element derived here is notably similar to the optimal stiffness coefficient κ derived in [4, Divergence-Free SPH] for use in an iterative solver that is otherwise similar in structure to a regular state equation solver. The optimal κ can in fact be phrased in terms of the diagonal element above as [4]:

$$\kappa_i = \frac{\rho_0 - \rho_i^*}{\mathbb{A}_{ii} \rho_i} \quad (3.61)$$

and another [29, survey paper] shows that the density solve of DFSPH is essentially equivalent to the IISPH solver presented in this report.

It is also worth noting that both the specific boundary handling method described in chapter 4, namely [5, consistent boundaries] with density mirroring, as well as the semi-implicit time integration scheme used in Equation 3.5 and in the operator splitting in Equation 3.11 are used in the derivation of the expression for \mathbb{A}_{ii} and the diagonal element needs to be adjusted if any of these components differ.

Lastly, a slight variation of the algorithm was implemented based on the assumption that for strongly enforced incompressibility $\rho_i = \rho_j = \rho_0$ holds, namely all densities in the pressure solver and the pressure acceleration calculation were replaced with ρ_0 , which was hoped to result in a more robust solver.

Relaxed Jacobi Solver

With the source term s_i shown in Equation 3.35, the diagonal element \mathbb{A}_{ii} from Equation 3.60, both of which can be pre-computed, as well as the pressure acceleration \vec{a}_i^p in Equation 4.8 and the right-hand side of the PPE $(\mathbb{A}\vec{p})_i$ in Equation 3.39, all the values required to implement a relaxed Jacobi solver and solve the system of equations for the unknown pressures p_i are available. The core idea here is to iterate:

RELAXED JACOBI STEP

$$p_i^{l+1} = \left(p_i^l + \omega \frac{s_i - (\mathbb{A}\vec{p})_i}{\mathbb{A}_{ii}} \cdot \mathbb{1}(\mathbb{A}_{ii} > \epsilon) \right)_+ \quad (3.62)$$

until the predicted density error per particle $(\rho_i^{err})^l = (\mathbb{A}\vec{p})_i - s_i$ fulfils some convergence criterion. The indicator function $\mathbb{1}(P)$ for a predicate P is used to only update the pressure when the diagonal element, which the predicted density error is divided by, is greater than some positive, small threshold like $\epsilon = 10^{-6}$ in order to avoid numerical instabilities and divisions by zero. Note also the use of the clamping operator $(x)_+ = \max(0, x)$ that can be viewed as a proximal operator [28] and ensures that pressure values are always positive. This prevents undesirable clumping artefacts that could otherwise occur when using an SPH discretization [18] and are speculated to result from the assumption that regions with incompletely sampled regions at free surfaces are implicitly assumed to have a pressure value of zero where there are no fluid particles (as illustrated in Figure 2.2) [1]. Since the solver is run until convergence, this clamping is unproblematic because it effectively shifts all pressure values up to be positive, while not changing the pressure gradient that is actually used to compute pressure forces and accelerations [1].

The choice of a relaxation parameter $\omega \in]0, 1[$ is not just the namesake of the relaxed Jacobi solver but determines its convergence properties: like with any relaxation scheme, a higher value of ω leads to faster convergence, while a lower value of ω might prevent numerical instabilities. In practice, the critical value $\omega = 0.5$ is used, which is the highest value that has been shown to lead to robust convergence [18].

For the initial iteration $l = 0$, the authors of the [18, original paper] recommend warm-starting the solver by initializing all pressure values \vec{p} with half the pressure computed in the previous time step, i.e.:

$$p_i^0(t) = \omega_{ws} p_i^0(t - \Delta t_{prev}) \quad (3.63)$$

with $\omega_{ws} = 0.5$. Since the initial time step $t = 0$ ought to be an uncompressed state anyways, the base case of this backwards recursion is unproblematic. However, a more conservative initialization which realizes a cold start of the solver can be used instead to trade off some performance for more robust behaviour. In this case, the pressure can be more efficiently reset by inserting $p_i = 0$, $(\mathbb{A}\vec{p})_i = 0$ into Equation 3.62 and obtaining:

$$p_i^0 = \frac{\omega s_i}{\mathbb{A}_{ii}} \quad (3.64)$$

effectively skipping the first solver iteration by setting the pressure to the result thereof. The relative performance and stability of these solver variations is analysed in chapter 6.

The relaxed Jacobi solver as presented in Equation 3.62 can suffer in practice from poor convergence or even exploding pressure values when the initial guess for the pressures p_i^0 is too high, so for especially complex scenes with high fluid columns and where pressures might change abruptly, cold-starting the solver might be preferable when a slightly higher iteration count per time step can be tolerated.

Convergence Criteria

The Jacobi iteration in Equation 3.62 is repeated until the pressure values p_i have converged to a solution that ensures incompressibility at the next time step, but the exact stopping criterion or convergence criterion has yet to be discussed.

In subsubsection 3.3 the predicted density error $(\rho_i^{err})^l = (\mathbb{A}\vec{p})_i - s_i$ that appears in the Jacobi update has already been mentioned. This predicted error is not exactly equal to the density error at the next time step since some approximations were made, for example when a repeated neighbourhood search was foregone in favour of calculating predicted density ρ^* using the velocity divergence in Equation 3.15. However, the predicted density error is, by design, as close as possible to the actual density error at the next time step as can be, given the information available, and can be reused from Equation 3.62 at no additional computational cost, so it is often used [1, 18].

Two predicted error metrics are commonly employed [18]:

$$\rho_{max}^l = \frac{1}{\rho_0} \max_i \left((\rho_i^{err})^l \right) \quad \text{Maximum predicted density error} \quad (3.65)$$

$$\rho_{avg}^l = \frac{1}{\rho_0} \frac{1}{N} \sum_{i=1}^N (\rho_i^{err})^l \quad \text{Average predicted density error} \quad (3.66)$$

The [18, authors of IISPH] recommend iterating until the average predicted density error is below a threshold of 0.1%, such that no oscillations in the fluid are perceptible. The ratio of ρ_{max}^l and ρ_{avg}^l varies, so the authors argue that the keeping the maximum error fixed does not prevent visual artefacts that stem from varying average density, especially for tall fluid columns in complex scenes [18].

On the other hand, recalling the particle deficiency phenomenon discussed in subsubsection 2.2, the average density can inadvertently be influenced by the geometry of the fluid, where a fluid with a large surface area and therefore many particles at the free surface can have its 'true' average density underestimated by the SPH discretization. This was found in this implementation to negatively impact simulations of, for example, shallow waters, where the solver might stop iterating prematurely because of this underestimation of average density. The maximum density metric does not suffer from this issue, so a conjunction of both convergence criteria was used in the implementation presented here to achieve more robust behaviour for arbitrary scenes.

Note that the maximum predicted density error is a strict upper bound of the average predicted error, so the threshold for ρ_{max}^l poses a stricter constraint and can be relaxed to 0.5% for example.

The final convergence criterion is a boolean predicate on the predicted density errors $(\rho_i^{err})^l$ in the l -th iteration and can be stated as:

CONVERGENCE CRITERION

$$(\rho_{avg}^l \leq \rho_{avg}^{thresh}) \wedge (\rho_{max}^l \leq \rho_{max}^{thresh}) \wedge (l_{min} \leq l \leq l_{max}) \quad (3.67)$$

where minimum and maximum solver iteration counts such as $l_{min} = 3, l_{max} = \infty$ can be added to increase robustness and ensure reasonably converged pressure values to use when warm-starting the next iteration. Recommended choices for the threshold values are $\rho_{avg}^{thresh} = 0.1\%$ [18] and $\rho_{max}^{thresh} = 0.5\%$.

Using only upper bounds as thresholds, i.e. \leq in Equation 3.67, ensures that only compressions are penalized by the solver whereas a predicted density below rest density is allowed, since the particle deficiency phenomenon (see subsubsection 2.2) could otherwise cause the solver to never converge when the ratio of surface area to volume of the fluid in its current configuration is large and for example $|\rho_{avg}^l| \leq \rho_{avg}^{thresh}$ could never be true.

IISPH Solver

Putting all the equations in subsubsection 3.3 that form the implicit pressure solver together with the discretized Navier-Stokes equations in section 3.1 and the boundary handling from section 4.1 results in Algorithm 1. The rigid body solver from section 4.2, which enables the two-way interaction of the fluid with dynamic rigid bodies, has also been included in the overview.

Before the algorithm is invoked, the initial conditions of the fluid can be prepared as described in subsubsection 3.1 while the initialization of the boundary is discussed in subsection 4.1.1.

Algorithm 1 IISPH Fluid Solver and Two-way Coupled Rigid Solver Step

<i>Step 1 – Preliminary Computations</i>	
1: Find neighbour sets $\mathcal{N}_i, \mathcal{B}_i$	▷ Fixed Radius Neighbour Search (Appendix A)
2: Compute densities ρ_i	▷ Equation 4.7
3: Compute non-pressure accelerations \vec{a}_i^v, \vec{g}	▷ Equation 3.2
4: Determine time step size Δt	▷ Equation 3.7 or subsection 3.4.2
5: Compute predicted velocities \vec{v}_i^*	▷ Operator splitting, Equation 3.11
<i>Step 2 – IISPH - Precomputations</i>	
6: Compute diagonal element A_{ii}	▷ Equation 3.60
7: Compute source term s_i	▷ Equation 3.35
8: Reset pressure values or warm-start p_i	▷ Equation 3.63 or Equation 3.64
<i>Step 3 – IISPH Solver Iteration</i>	
9: for iteration $l = 0$ until convergence do	▷ convergence criteriton Equation 3.67
10: Compute pressure acceleration $\vec{a}_i^{p,l}$	▷ Equation 4.8
11: Compute predicted density error due to pressure accelerations $(\mathbb{A}\vec{p})_i^l$	▷ Equation 3.39
12: Update pressure using relaxed Jacobi iteration	▷ Equation 3.62
13: end for	
<i>Step 4 – Numerical Time Integration</i>	
14: $\vec{v}(t + \Delta t) \leftarrow \vec{v}_i^* + \Delta t \vec{a}_i^p$	
15: $\vec{x}(t + \Delta t) \leftarrow \vec{x}_i + \Delta t \vec{v}(t + \Delta t)$	▷ like Equation 3.5, respecting operator splitting
<i>Step 5 – Update Rigid Bodies and Dynamic Boundary Particles</i>	
16: Use final \vec{a}_i^p to compute $\vec{f}_i, \vec{\tau}_i$ on rigid bodies	
17: Update rigid body state and corresponding boundary particles	▷ Algorithm 5

3.4 Solver Variations and Degrees of Freedom

There exist numerous variations of the presented IISPH solver that aim to improve solver performance or some measure of simulation accuracy or quality. Some degrees of freedom in the solver implementation that have been mentioned in this report include:

1. the time step size to use, as limited by the CFL condition in Equation 3.7 and further discussed in subsection 3.4.2
2. whether to warm-start the solver using pressure values of the previous iteration, see ω_{ws} in Equation 3.63. Generally, cold starts tend to be more stable, while warm starts tend to increase performance
3. using SPH-estimated densities ρ_i, ρ_j or the rest density ρ_0 in pressure acceleration computations and the derivation of the diagonal element as mentioned in subsubsection 3.3
4. minimum and maximum iteration counts as well as error thresholds in the convergence criterion as seen in Equation 3.67
5. The choice of discretization of the Laplacian in the pressure Poisson equation as discussed in subsubsection 3.3

The choice of time step size and source term in particular are deemed especially relevant in this report and discussed in the following.

3.4.1 Divergence-Free Source Term

A particularly important and oft-discussed degree of freedom in IISPH implementations is the choice of **SOURCE TERM** for the pressure Poisson equation. Recall that the defining constraint of the pressure solver is that pressure accelerations computed by the solver should result in a fluid state that satisfies the continuity equation (Equation 2.2) at the next time step. For incompressible fluids, this equation results in two equivalent constraints: the density in the fluid should remain at rest density at all times ($\frac{D\rho}{Dt} = 0$)

and the velocity field should remain divergence-free ($\nabla \cdot \vec{v} = 0$). The IISPH solver presented in Algorithm 1 solves a PPE that uses a discretized density invariance source term as stated in Equation 3.35.

Both source term formulations have benefits and drawbacks.

Minimizing the velocity divergence can result in a more accurate and smoother velocity field with fewer oscillations and improved stability, and is often the method of choice in Eulerian approaches [4]. On the other hand, errors accumulate as drift, since the minimized quantity is a differential in time, meaning that only minimizing velocity divergence can lead to volume loss (as is often the case in Eulerian approaches [4]) and the quality of the particle sampling can degrade over time, rendering the SPH approximations increasingly inaccurate [26]. Particle-based boundary representations that are often desired for their flexibility and robustness (as outlined in section 4.1) can be challenging to realize in this setting due to drift causing leakage.

Only enforcing density invariance on the other hand can result in a less accurate and temporally consistent velocity field where errors manifest as unwanted oscillations and can cause significant artificial viscosity, but may better preserve the quality and regularity of the particle sampling and does not suffer from volume loss [26].

After the IISPH method had been established, authors have sought to find pressure solvers that can leverage the strengths of each of these source term formulations without suffering from its drawbacks, resulting in [26, more optimized source term formulations] and, very similarly, the [4, Divergence-Free SPH] or DFSPH method. The idea is to solve the pressure Poisson equation twice, using each source term, and combining the results. The former paper achieves this by computing a velocity field that minimizes divergence, estimating an updated velocity gradient $\nabla \vec{v}$ from it by employing Equation 2.28 and using this velocity gradient to interpolate the divergence-free velocity field computed by the first solver to the updated particle positions calculated by a second solver, which uses the density invariance source term.

DFSPH on the other hand was originally proposed to appear similar to an iterative Equation-of-State or EOS solver, but use a state equation $\nabla p_i = \kappa_i \nabla \rho_i$ [4] with stiffness coefficients κ_i alluded to in Equation 3.61, which are specifically chosen per-particle to realize a solution to the pressure Poisson equation in a way that is [29, essentially equivalent] to the relaxed Jacobi iteration used by IISPH. Crucially, two such coefficients are calculated:

$$\kappa_i = \frac{1}{\Delta t^2} (\rho_i^* - \rho_0) \alpha_i \quad (3.68)$$

$$\kappa_i^v = \frac{1}{\Delta t} \frac{D\rho_i}{Dt} \alpha_i \quad (3.69)$$

where the computationally expensive but common factor $\alpha_i = -\frac{\Delta t^2}{\rho_i \mathbb{A}_{ii}}$ can be reused. Each coefficient is used in a subsequent, iterative solve of the PPE using the respective source term formulation, each updating predicted velocities that are finally integrated to update particle positions. Note that the terms given in the [4, DFSPH paper] do not include boundary handling, proper clamping of values and other details that can be critical for stability and are present in the [30, authors' own implementation]. Since the density invariance solve of DFSPH is not just equivalent to IISPH but the main author of the DFSPH paper [4] [5, in a later paper] defined DFSPH to be a relaxed Jacobi solver as outlined in this report, but using an additional velocity divergence source term, followed by the exact density invariant solver seen in Algorithm 1, this is the form of the DFSPH solver stated in the following, rather than an equivalent form using κ_i, κ_i^v . The discretized velocity divergence source term was already mentioned in Equation 3.35 and is simply:

VELOCITY DIVERGENCE SOURCE TERM

$$s_i^v = -\Delta t \left(\frac{D\rho}{Dt} \right)^* = -\Delta t \rho_0 \langle \nabla \cdot \vec{v}_i^* \rangle_\nabla. \quad (3.70)$$

$$= -\Delta t \rho_0 \left(\sum_{j \in \mathcal{N}_i} m_j \vec{v}_{ij}^* \cdot \nabla W_{ij} + \sum_{k \in \mathcal{B}_i} m_j \vec{v}_{ik}^* \cdot \nabla W_{ik} \right) \quad (3.71)$$

This source term results in the DFSPH fluid solver outlined in Algorithm 2. This algorithm necessitates a second convergence criterion for the velocity divergence - in the most simple case, a fixed and low iteration count such as $l_{max} = 2$ is set and the divergence solve is simply seen as another application of operator splitting, yielding what is hoped to be a much easier problem to solve to the later, density invariant solver. Care needs to be taken to now consider both resulting pressure accelerations from the two solvers in the computation of total forces \vec{f}_i exerted by the fluid on dynamic rigid bodies.

One drawback of implementing the DFSPH solver as outlined in Algorithm 2 is that the meaning of warm-starting each of the solvers becomes questionable. While the [4, original paper] proposes using the respective values of κ_i, κ_i^v of the previous simulation step to warm-start each of the two solver separately, the same principle does not carry over to Algorithm 2 in an obvious way - *should a single pressure value be shared by the solvers and halved before one takes over from the other, should the solvers compute completely separate pressure values that are separately initialized or should some other scheme be used?* Generally, initializing pressures to a value that is too high leads to instability, while resetting pressures to a value too far from a previous solution might lead to suboptimal performance. In this report, warm-starting DFSPH is realized by using a single set of pressure values p_i , applying Equation 3.63 once at the start of the time step to initialize these pressures and not resetting them at all between the two solvers. This scheme was not found to result in instability for the tested scenarios in chapter 6.

Algorithm 2 DFSPH Fluid Solver and Two-way Coupled Rigid Solver Step

Step 1 – Per-Step Precomputations

- 1: Find neighbour sets $\mathcal{N}_i, \mathcal{B}_i$ ▷ Fixed Radius Neighbour Search (Appendix A)
 - 2: Compute densities ρ_i ▷ Equation 4.7
 - 3: Compute non-pressure accelerations \vec{a}_i^v, \vec{g} ▷ Equation 3.2
 - 4: Determine time step size Δt ▷ Equation 3.7 or subsection 3.4.2
 - 5: Compute predicted velocities \vec{v}_i^* ▷ Operator splitting, Equation 3.11
 - 6: Compute diagonal element \mathbb{A}_{ii} ▷ Equation 3.60
 - 7: Compute velocity divergence source term s_i^v ▷ Equation 3.70
 - 8: Reset pressure values or warm-start p_i ▷ Equation 3.63 or Equation 3.64
-

Step 2 – Velocity Divergence Solver

- 9: **procedure** RELAXEDJACOBISOLVER(s_i)
 - 10: **for** iteration $l = 0$ until convergence **do**
 - 11: Compute pressure acceleration $\vec{a}_i^{p,l}$ ▷ Equation 4.8
 - 12: Compute predicted density error due to pressure accelerations $(\mathbb{A}\vec{p})_i^l$ ▷ Equation 3.39
 - 13: Update pressure using relaxed Jacobi iteration ▷ Equation 3.62
 - 14: **end for**
 - 15: **end procedure**
 - 16: RELAXEDJACOBISOLVER(s_i^v) ▷ s_i^v from Equation 3.70, use a fixed iteration count
 - 17: $\vec{v}_i^* \leftarrow \vec{v}_i^* + \Delta t \vec{a}_i^p$ ▷ Update predicted velocities
 - 18: Accumulate the used \vec{a}_i^p to compute \vec{f}_i on rigid bodies
-

Step 3 – Density Invariance Solver

- 19: Compute density invariance source term s_i ▷ Equation 3.35
 - 20: RELAXEDJACOBISOLVER(s_i) ▷ s_i from Equation 3.35, convergence criteriton Equation 3.67
-

Step 4 – Numerical Time Integration

- 21: $\vec{v}(t + \Delta t) \leftarrow \vec{v}_i^* + \Delta t \vec{a}_i^p$
 - 22: $\vec{x}(t + \Delta t) \leftarrow \vec{x}_i + \Delta t \vec{v}(t + \Delta t)$ ▷ like Equation 3.5, respecting operator splitting
-

Step 5 – Update Rigid Bodies and Dynamic Boundary Particles

- 23: Use further accumulated \vec{a}_i^p to compute $\vec{f}_i, \vec{\tau}_i$ on rigid bodies
 - 24: Update rigid body state and corresponding boundary particles ▷ Algorithm 5
-

3.4.2 Performance Optimizing Metropolis Time Step Selection

Given the amount of degrees of freedom in the implementation of an incompressible SPH solver, the question of how to robustly measure the relative performances of solver variations naturally arises. In the literature there appears to be no clear consensus on how to measure such differences in performance and stability, or the comparisons that are conducted can be called into question due to the amount of free parameters: *how can one be sure that suboptimal relative performance is due to an algorithmic disadvantage and not suboptimal selection of free parameters?*

One such parameter in particular is critical for the algorithm's performance, namely the time step size Δt . While the Courant-Friedrichs-Lowy condition in Equation 3.7 does put an upper limit on the time step size, a couple of issues arise:

1. For scenes such as a resting column of water under hydrostatic pressure, the maximum velocity $\max_i |\vec{v}_i|$ is close to zero, enabling close to arbitrary time step sizes. Commonly, a maximum time step size Δt_{max} is introduced to circumvent this.
2. The CFL condition only provides a safe upper bound on the time step size but says nothing about the optimality of any given Δt

It is clear that too small a time step size results in suboptimal performance, since an error within the bounds of the convergence criterion could also have been achieved using a larger time step at a given iteration count. Similarly, a time step size that is too large must be suboptimal, since even assuming it is still in the bounds enforced by the CFL condition that guarantee stability, the required number of solver iterations for larger time steps sizes will grow disproportionately after some point. Assuming the solver converges not linearly, but makes most progress in its first few iterations and then eases towards convergence, a Δt requiring many solver iterations must at some point become less efficient than using fewer solver iterations at the cost of making smaller steps. If the constant computational cost per time step is large and the pressure solve is cheap, a larger time step size tends to be more efficient, but if the cost of the pressure solve is significant compared to constant costs per step, as can be the case for incompressible SPH in particular, a lower iteration count might be desirable.

Defining Performance from Noisy Measurements

To further formalize the problem of performance-optimal time step selection, the notion of solver performance can be defined as:

$$P = \frac{T_{sim}}{T_{comp}} \quad (3.72)$$

where T_{sim} is the simulated time and can be exactly computed by summing the Δt of subsequent time steps under consideration, while T_{comp} is some measurement of the wall-clock-time or process time taken by the solver to compute the same steps. This means that for a given hardware setup, scene complexity and particle count, $P \geq 1$ means 'real-time' performance, while for challenging scenes a value of $P \ll 1$ can be expected.

Unfortunately, simply measuring the execution time of a single time step has limitations: due to inherent imprecisions of the timing method itself, differing resource allocations over time and other factors outside the control of the fluid solver, performance measurement of single steps can be noisy and yield high variance. Instead, a model for the estimated performance of a time step is suggested here, which takes into account the structure of the fluid solver in Algorithm 1 to filter some of this noise:

$$\hat{T}_{comp} \approx \bar{T}_{const} + l \cdot \bar{T}_{iter} + w \quad (3.73)$$

Here, the estimated computation time \hat{T}_{comp} is decomposed into a constant component per time step T_{const} , which comprises calculations such as neighbourhood search, computation of densities ρ_i , non-pressure accelerations, pre-computed values such as the source terms s_i and diagonal elements \mathbb{A}_{ii} and so on, which can be expected to take very similar computational effort at each step. Similarly, the computation time per iteration of the pressure solver T_{iter} can be assumed to be near-constant, approximated by its long-term average \bar{T}_{iter} and multiplied by the iteration count l in the current step, leaving only a noise term w , which can reasonably be modelled as a Gaussian, additive noise accounting for any variation in the measurement that does appear.

The assumption that T_{const} and T_{iter} are near-constant across time steps is supported by a couple of observations. Note that the sorting-based neighbourhood search outlined in Appendix A comprises a very predictable and near-constant number of operations in $\mathcal{O}(N)$ in the number N of particles. The SPH sums can also be assumed to take near-identical computational effort in each step since the maximum number of neighbours of each particle M is near-constant for many scenes when incompressibility is strongly enforced, and neighbourhoods of particles tend to be filled as resolution increases, making the SPH sums which are in $\mathcal{O}(MN)$ very predictable in runtime. This is further supported by the frequent resorting of particle attribute buffers enabled by the counting sort in Appendix A, which enhances memory coherence and makes performance degradation due to scattered read and write operations much less significant, thus keeping the computational cost lower and more constant.

This means that assuming noisy observations in the k -th time step $T_{const,k}$ and $T_{iter,k}$ that are spread about some constant, expected value, an unbiased Monte-Carlo estimate of the expected values

of $\bar{T}_{const}, \bar{T}_{iter}$ that converges to the true constant can be obtained by the long-term average values:

$$\bar{T}_{const} = \frac{1}{K} \sum_{k=0}^K T_{const,k} \quad (3.74)$$

$$\bar{T}_{iter} = \frac{1}{\sum_{k=0}^K l_k} \sum_{k=0}^K l_k \cdot T_{iter,k} \quad (3.75)$$

where K is the total number of time steps observed so far and l_k is the solver iteration count of the k -th time step. This leads to an estimated performance metric of the k -th time step:

ESTIMATED PERFORMANCE SCORE

$$\hat{P}_k = \frac{\Delta t_k}{\bar{T}_{const} + l_k \cdot \bar{T}_{iter}} \quad (3.76)$$

Problem Statement and Outlook

If there was an algorithm that chose time step sizes Δt_k automatically as to optimize \hat{P}_k , the relative performance of algorithms could be compared robustly and empirically without the choice of the upwards-bound but otherwise free parameter Δt , which directly affects the simulation performance as defined in Equation 3.76. Furthermore, this optimization task has practical relevance as it would increase performance within the bounds on stability and simulation quality imposed by the CFL-condition and convergence criterion respectively and at the same time eliminate hand-tuning of parameters of the fluid solver related to time step size. When appropriate, the solver would simply choose to use a lower-than-possible time step size to achieve maximal performance at no cost to quality.

Unfortunately, this optimization problem appears to have some challenging features. It can generally be understood as a constrained, non-linear, possibly non-convex, black-box optimization of a time-dependent and noisy function in a delayed feedback loop, placing the problem at a particularly challenging intersection of Artificial Intelligence (and reinforcement learning in particular), control theory, the study of time-inhomogeneous Markov decision problems, numerical optimization and related fields. Despite the arguably great difficulty of this general formulation of the problem, many simplifying assumptions can reasonably be taken that result in a feasible algorithm for automatic time step selection as described in the following, which already shows promising results. Avenues of future improvement to this first approximation might include:

1. Considering the problem as an instance of [31, online learning for black box optimization] or controller tuning and applying a form of [32, Time-Varying Bayesian Optimization] or similar [33, time-varying optimization]
2. Considering the time step controller an agent in a changing environment and applying Reinforcement Learning approaches to it
3. Lifting the veil of black-box optimization by modelling the performance function (as already partly done in Equation 3.73), taking a closer analysis of the convergence behaviour of the solver into account or calculating a target iteration count (which can be controlled using a simple feedback loop) based on the ratio of constant to per-iteration costs.

Design goals in creating a time step controller include:

1. Correctness, as in finding an optimal time step size given the problem specification
2. Fast convergence towards that optimal time step size
3. Swift adaption to changes in problem difficulty for dynamic scenes
4. Smooth convergence towards the optimum, since sudden jumps in time step size can cause solver instabilities

Especially the last goal represents a challenge, since it rules out some common optimization techniques and can be seen as at odds with some of the other design goals, making compromise necessary.

Greedy Time Step Size Control

The performance function changes with time. Consider for example a scene with a flat, water-filled basin flowing into a tall container: at first, the maximum height of any water column in the scene is low, but it later increases, changing the performance-optimal time step size. However, it can be reasonably assumed that such changes to $\hat{P}_{k,t}$ occur on a timescale much larger than that of individual time steps Δt , for example on the scale of seconds in scenes where Δt is on the scale of milliseconds. For now, it can be assumed that \hat{P}_k is approximately constant on the scale of more time steps than necessary to find an optimum of \hat{P} . Perhaps the simplest method for finding such an optimum would then be a greedy **MONTE CARLO LOCAL SEARCH**. This method explores values in the neighbourhood of the current time step size Δt_k using random sampling from some proposal distribution \mathcal{P} , which is in this case simply taken to be the normal distribution with some variance such as $\sigma_{\mathcal{P}}^2 = 10^{-6}$ sec that is orders of magnitude smaller than the anticipated time step size:

PROPOSAL DISTRIBUTION

$$\mathcal{P} = \mathcal{N}(\mu = \Delta t_k, \sigma_{\mathcal{P}}^2) \quad (3.77)$$

Values that yield higher performance are then greedily accepted, all other proposals are rejected. The algorithm is outlined in Algorithm 3. Note that the variance of the proposal distribution is an important hyperparameter: too high a value will mean sudden jumps in time step size, which can greatly impact the stability of the solver in this application. Variance that is too small will lead to slow convergence to optimal time step sizes, which is not ideal but more tolerable than instability, so a conservatively low choice of $\sigma_{\mathcal{P}}^2$ is recommended.

Algorithm 3 Monte Carlo Local Search

```

1: for every time step  $k$  do
2:   Draw a proposed time step size  $\Delta t'_{k+1} \sim \mathcal{P}$  from proposal distribution  $\mathcal{P}$ 
3:   Ensure  $\Delta t'_{k+1} \leq \Delta t_{CFL}$  by clamping
4:   Run simulation step and measure performance  $\hat{P}(\Delta t'_{k+1})$  of the proposal
5:   if  $\hat{P}(\Delta t'_{k+1}) > \hat{P}(\Delta t_k)$  then
6:      $\Delta t_{k+1} \leftarrow \Delta t'_{k+1}$ 
7:   else
8:      $\Delta t_{k+1} \leftarrow \Delta t_k$ 
9:   end if
10:  end for

```

This greedy scheme unfortunately becomes inadequate in the context of performance that varies with time: an optimum with a performance score could be found which can no longer be matched by any proposal because the time-dependent problem has gotten more difficult in the meantime, leading to every proposal being rejected and the controller being stuck in a past optimum. To circumvent this, an **EXPLORATION-EXPLOITATION TRADE-OFF** must famously be struck [31], balancing the greedy and exploitative behaviour of the local search with exploration of the search space which increases entropy and enables the escape from convergence to local extrema [34].

Random-Walk-Metropolis with Automatic Temperature Selection

The **RANDOM WALK METROPOLIS ALGORITHM** or RWM solves this problem by accepting even non-improving proposals by using an acceptance probability of [34]:

$$\Pr\{\text{accept } \Delta t'_{k+1}\} = \begin{cases} \exp\left(\frac{\hat{P}'_{k+1} - \hat{P}_k}{T_k}\right) & \hat{P}_k > \hat{P}'_{k+1} \\ 1 & \hat{P}_k \leq \hat{P}'_{k+1} \end{cases} \quad (3.78)$$

where T_k is a control parameter, also called ‘temperature’ in the context of simulated annealing [34] due to the physical analogy of annealing metal to find a crystalline minimum in the energy landscape of atom configurations when temperature is slowly brought down [34]. Note that the intended behaviour is $\lim_{T_k \searrow 0} \Pr\{\text{accept } \Delta t'_{k+1}\} = 0$ to avoid division by zero. Usually, a temperature schedule for T_k such as geometric cooling [34], which eases the temperature towards zero, is used. The same can not be

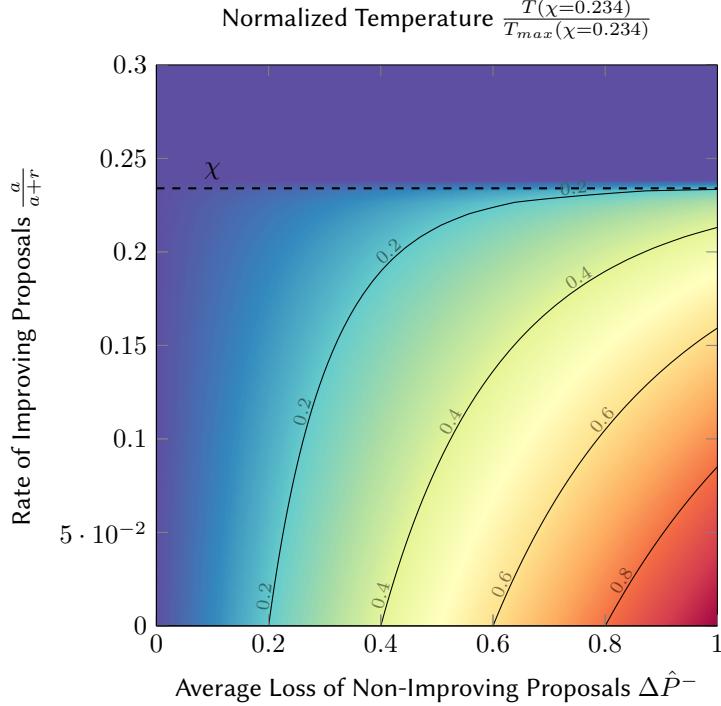


Figure 3.1: The temperature T ($\chi = 0.234$), normalized from $[0; T_{max}]$ to $[0; 1]$ is plotted for differing rates of improving proposals z and average loss of non-improving proposals $\Delta\hat{P}^-$. Note that the function is scale-invariant in $\Delta\hat{P}^-$, i.e. if the temperature is normalized between its minimum and maximum values to values in $[0; 1]$ then the shape of the graph does not change if $\Delta\hat{P}^-$ is plotted from zero to one or from zero to any other positive number. This scale-invariance shows that the unit of measurement of performance has no impact on the behaviour of the algorithm, which is intended. The plot can be interpreted by assuming lower temperature values mean more greedy behaviour, while higher temperatures result in more exploration. When the rate of improving proposals z is higher than the target acceptance rate χ , the algorithm is strictly greedy. As the rate of improving proposals drops below the threshold, and particularly as $\Delta\hat{P}^-$ increases and the proposals get worse, the temperature increases so that the acceptance rate can be pushed up to the target level of χ . This means that when improvement becomes stale or the current optimum is no longer valid and proposals get worse, the temperature increases to enable more exploration in hopes of discovering a better time step size Δt .

applied in this context, since \hat{P} is actually time-dependent and potential changes to the function and its maximum should be explored by increasing the temperature even at a later time in the simulation.

Temperature is instead controlled using the estimation for the rate of acceptance of proposals χ [34]:

$$\chi(T) \approx \frac{a + r \exp\left(-\frac{\Delta\hat{P}^-}{T}\right)}{a + r} \quad (3.79)$$

where a and r are the number of improving (i.e. $\hat{P}_k < \hat{P}'_{k+1}$) and non-improving proposals (i.e. $\hat{P}_k \geq \hat{P}'_{k+1}$) respectively and $\Delta\hat{P}^- = \frac{1}{r} \sum_{k=0}^K (\hat{P}_k - \hat{P}'_{k+1})_+$ is the average decrease in performance score of non-improving transitions. This can be rearranged to yield a formula for the temperature T in terms of a target acceptance rate χ [34]:

TEMPERATURE FUNCTION

$$T(\chi, z) = \frac{\Delta\hat{P}^-}{\ln\left(\frac{1-z}{\chi-z}\right)} \cdot \Theta(\chi - z) \quad (3.80)$$

where $z = \frac{a}{a+r}$ is the observed rate of improving proposals. When $z \geq \chi$ the fraction is undefined due to a negative argument to the logarithm and the temperature is set to zero instead by the Heaviside step function Θ . This results in a temperature $T(\chi)$ as shown in Figure 3.1, which differs from zero only if the rate of improvement z drops below the target acceptance rate χ .

While this might seem like exchanging one arbitrary parameter choice for another, there is actually a reasonable default for the acceptance rate χ that can be used: surprisingly, for a wide variety of proposal and target distributions it can be shown that as the dimensionality of the problem increases, the most

efficient choice of the target acceptance rate χ converges to $\chi = 0.234$ [35, 36]. Although the applicability of this *optimal scaling* to the specific problem at hand has not been rigorously investigated, as this would exceed the scope of this report, it serves here as a reasonable default to eliminate hyperparameter-tuning.

While Equation 3.80 is usually employed to set the initial temperature T_0 and then apply a cooling schedule [34], it is here applied throughout the simulation for the reasons mentioned: temperature should increase again when the acceptance rate drops below the threshold of $\chi = 0.234$, either because an optimum was found or because the underlying function \hat{P} has changed and what currently appears to be an ‘optimal’ performance score is in fact no longer attainable. To provide this flexibility at the cost of possibly oscillating about an optimal time step size, the changes of \hat{P} can be estimated to occur on an order of 1 sec $\gg \Delta t$ and a time horizon of e.g. $t_{hrz} = 1$ sec is introduced, beyond which observations are forgotten. Instead of calculating $T(\chi)$ from Equation 3.80 using all observed proposals, only proposals made in the last second are considered and proposals beyond this time horizon do not contribute to the values a , r and $\Delta\hat{P}^-$. This results in scheme that flexibly increases the temperature as to keep the rate of improving proposals in the last second of simulated time above the threshold $\chi = 0.234$.

Lastly, the fact that the time controller is in a *delayed* feedback loop and changes to the time step size might only see effects to performance after some iterations is taken into account. This delay can be accounted for and the performance function \hat{P} can be made yet less noisy by dividing the simulation into episodes $E = \{k, k+1, \dots, k+K_E - 1\}$, using $K_E = 5$ time steps each in this instance, which constitute a single, averaged performance measurement. This makes rewards for a good proposal of $\Delta t'$ more likely to be captured by the performance score of the corresponding episode, instead of rewarding some later, possibly unrelated proposal.

The total performance score in Equation 3.76 applied to one episode then becomes:

$$\hat{P}(E) = \frac{\sum_{k \in E} \Delta t_k}{K_E \cdot \bar{T}_{const} + (\sum_{k \in E} l_k) \cdot \bar{T}_{iter}} \quad (3.81)$$

while variance $\sigma_{\mathcal{P}}^2$ of the proposal distribution is simply linearly scaled by the steps per episode K_E because the proposed changes to Δt are uncorrelated:

$$\mathcal{P} = \mathcal{N}(\mu = \Delta t_k, K_E \cdot \sigma_{\mathcal{P}}^2) \quad (3.82)$$

This results in the time step controller shown in Algorithm 4. This controller was tested on both dam-break and static water column scenarios and found to converge to time step sizes close to the optimum obtained by hand-tuning parameters for performance.

Algorithm 4 Random Walk Metropolis Controller for Δt

- 1: **for** every episode E **do**
 - 2: Draw a proposed time step size $\Delta t' \sim \mathcal{P}$ from proposal distribution \mathcal{P} ▷ Equation 3.82
 - 3: Run simulation for K_E steps using $\Delta t_k = \min(\Delta t', \Delta t_{CFL})$ ▷ Δt_{CFL} from Equation 3.7
 - 4: Estimate performance $\hat{P}(E)$ ▷ Equation 3.81
 - 5: Update a, r, z and calculate temperature $T(\chi, z)$ ▷ Equation 3.80
 - 6: Calculate acceptance probability $\Pr\{\text{accept } \Delta t'\}$ ▷ Equation 3.78
 - 7: Draw uniform random variable between 0 and 1: $u \sim \mathcal{U}([0; 1])$
 - 8: **if** $u < \Pr\{\text{accept } \Delta t'\}$ **then**
 - 9: $\Delta t \leftarrow \Delta t'$ ▷ accept proposal
 - 10: **end if**
 - 11: **end for**
-

The result of this automatic time step selection for an exemplary static water column is shown in Figure 3.2, highlighting how performance is approximately maximized through a convergence towards (despite an oscillation about) an optimal time step size. Note that for both the recommended value of $\chi = 0.234$ and the greedy local search with $\chi = 0$ the pressure solver iteration count for the performance-optimal time step size tends towards low values, between the minimum of 3 iterations and 10 iterations, which indicates that the neighbourhood search as presented in Appendix A is fast enough to the point where pressure solver iterations are a bottleneck of the fluid solver implementation and are minimized by the algorithm.

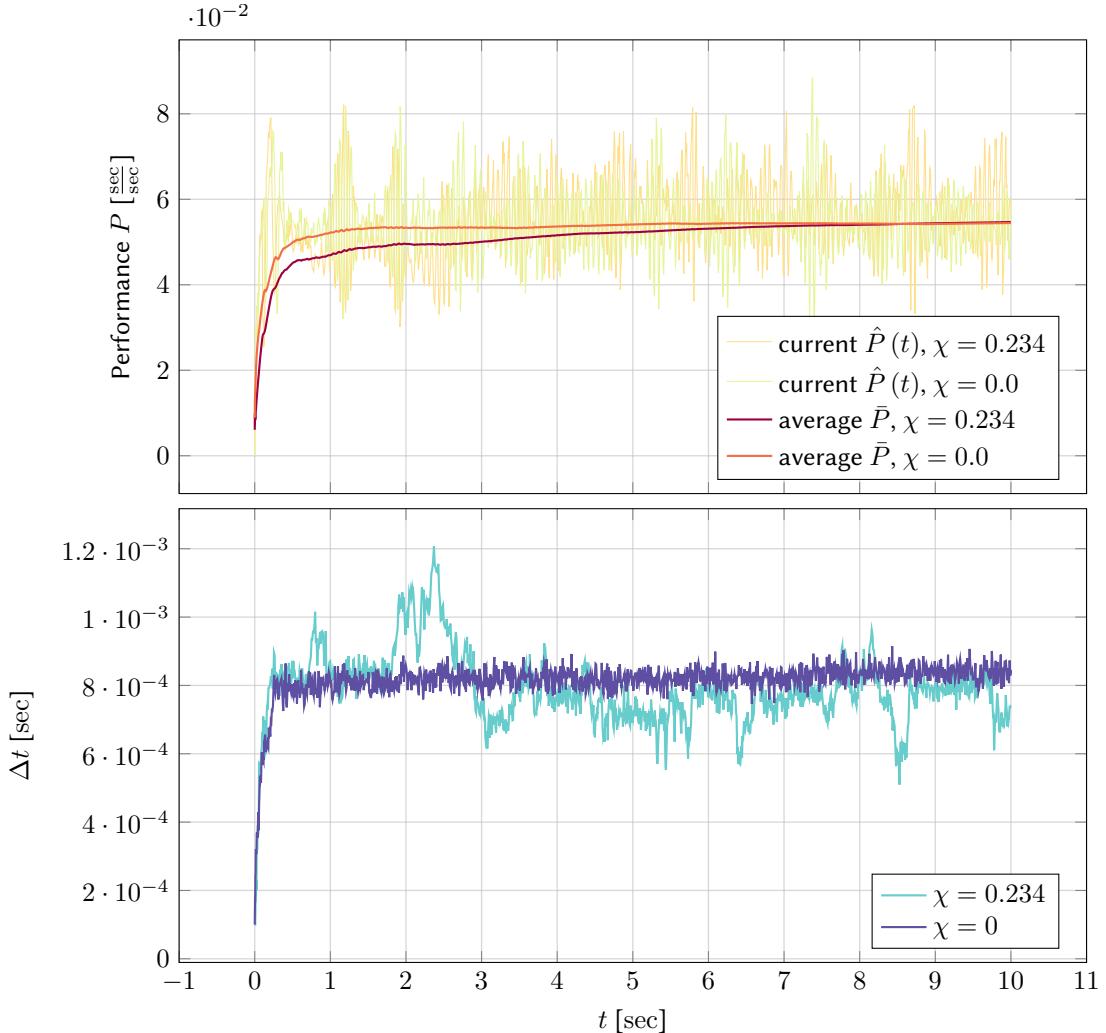


Figure 3.2: A $2m \times 2m \times 5m$ static water column of $302K$ fluid and $217K$ boundary particles is simulated for 10 sec of simulated time using automatic time step selection with $\sigma^2 = 5 \cdot 10^{-6}$ sec, $K_E = 5$ as described in Algorithm 4 using the IISPH fluid solver (subsubsection 3.3). The time horizon describes the expected timescale of changes to the problem difficulty and is therefore disabled using $t_{hrz} = \infty$ for this static scene. It can be observed that the initial, conservatively small time step size is quickly and automatically increased, increasing performance up to about 5% of real-time performance and subsequently oscillating about that optimum value, as the time step size oscillates about its respective optimum. The long-term average performance \bar{P} plateaus, its most recent value yielding a robust performance metric to compare other algorithms against. A relatively high target acceptance rate of $\chi = 0.234$ causes the time step size to vary throughout the entire simulation, exploring metastable local maxima, which would be desirable for dynamic scenes. A value of $\chi = 0$ instead equates to a purely greedy Monte-Carlo local search as described in Algorithm 3. This can be observed to lead to slightly faster convergence towards the optimum and less oscillation thereabout in the case of this static scene, but could result in the algorithm becoming stuck in a past optimum due to lack of exploration for a dynamic scene.

TWO-WAY COUPLED RIGID BODIES

The fluid solver from chapter 3 is not quite complete without discussing the boundary conditions within which the mixed initial-boundary value problem that the Navier-Stokes equations pose is solved [1]. A versatile method for discretizing boundary objects in a particle-based manner consistent with the SPH discretization of the fluid is discussed in section 4.1 [37]. While this already realizes one-way coupling of the fluid to a static boundary, a rigid body solver is subsequently developed and two-way coupled to the fluid solver to achieve interaction between dynamic rigs and fluids. The prerequisite mass moments are discussed in subsection 4.2.1, which are then used to implement a solver for rigid body kinematics in section 4.2. More challenging interactions such as rigid-rigid contacts with accurate friction that are still an active area of research [28] are not handled in this report.

4.1 Particle-Based Robust Boundary Representation

A common idea in modelling boundary interactions in SPH fluid solvers is to discretize the boundary into particles and extend the SPH sums from chapter 2 to include such boundary particles, which are treated in much the same way as fluid particles. This keeps the discretization of the governing equations consistent between fluid and boundary particles, using pressure forces of the incompressible fluid to guarantee non-penetration of the boundary while viscous forces or the absence thereof can realize slip or no-slip boundary conditions.

In order to extend the SPH sums over boundary particles, masses m_k , positions \vec{x}_k and velocities \vec{v}_k must be known for all boundary particles k . While \vec{v}_k is zero for static boundaries and can be computed as outlined in section 4.2 otherwise, the remaining two quantities highlight two challenges in implementing particle based boundary conditions: computing the size of the boundary particles, which equates to their mass m_k , and how best to sample the boundary domain, which yields the \vec{x}_k .

4.1.1 Sampling and Initialization

As will be outlined in the following, the boundary handling method presented in this report requires only a single layer of non-uniformly distributed boundary particles on the surface of the boundary to ensure accurate fluid-boundary interactions [37]. This is helpful, since thin shells and complex, non-manifold meshes can be represented [37], which is generally not trivial or even possible when multiple layers of particles or particles of uniform size must be used. Instead, a non-uniform but low-discrepancy sampling of the boundary's surface is sought in this instance.

To generate such points on a boundary's surface, an input triangular mesh representing an arbitrary boundary geometry is first approximated by a **WATERTIGHT MANIFOLD** with vertices that are roughly uniformly distributed across the surface using the method outlined by [38, Huang et al.]. This method represents the geometry using an octree and reconstructs it by isosurface extraction before projecting the newly created vertices onto the original mesh to improve accuracy [38].

The resulting mesh, while retaining the shape of the geometry as well as possible, is a watertight manifold, meaning that the notion of distance of any two points across the mesh becomes well-defined and sensible: the distance between two points that are not on the same triangle can be calculated by taking the mesh topology into account, since only at most two triangles in the mesh ever share a single edge across which the shortest path between two points can be unambiguously defined. This enables sampling techniques on the manifold that guarantee minimum or maximum distances between points, such that low-discrepancy or blue noise samplings can be found that ensure that boundary particles are evenly spread across the mesh but do not leave holes in the mesh for the fluid to penetrate.

In this instance, **POISSON DISK SAMPLING** was chosen as such a sampling technique. It guarantees that sampled points are at least some user-specified radius r apart. This can be done, for example [39], by repeatedly sampling the spherical annulus between r and $2r$ about some point in the sample set and

adding points to the set that are no less than r away from any other point in the set, until the entire surface is sampled. This can be done efficiently in $\mathcal{O}(N)$ in the number N of points in the final sampling using auxiliary data structures such as a uniform grid [39]. An example of the blue noise distribution of boundary particles generated on a mesh by this method is shown in Figure 4.1.

Both the Poisson disk sampling and resampling of the mesh as a watertight manifold are implemented in the *Point Cloud Utils* library that was employed in the implementation discussed in this report [40].

Boundary Particle Effective Mass

With the boundary particles' positions sampled on the boundary surface according to a blue noise distribution, a method for determining the effective mass of each boundary particle as it exerts forces upon a fluid particle is required to integrate the boundary formulation into the SPH sums in section 3.1. As seen in chapter 2, the SPH sum approximates an integral over field quantities, which can be partitioned into an integral over the fluid domain Ω_{fl} and the boundary Ω_{bdy} . The latter can be assumed to be of the rest density ρ_0 of the fluid under consideration, since contact forces are modelled as pressure forces and the pressure solver is reused to provide impenetrable boundaries. Note that in this report ρ_0 is a single, fixed value, while for multiphase flows it can have differing values. The rest density used in the pressure solver to handle boundaries is *not* the actual mass density of the material, such as wood, stone or steel, being modelled by the boundary Ω_{bdy} , which instead might influence the rigid body kinematics described in section 4.2.

For the density of a fluid particle near the boundary, one can write [41]:

$$\rho_i = \int_{\Omega_{fl}} W_{ij} dm_j + \int_{\Omega_{bdy}} W_{ik} dm_j \quad (4.1)$$

$$\langle \rho_i \rangle = \sum_{j \in \mathcal{N}_i} m_j W_{ij} + \sum_{k \in \mathcal{B}_i} m_k W_{ik} \quad (4.2)$$

where \mathcal{B} is defined analogously to \mathcal{N} (see Appendix A) but for boundary samples within a radius \hbar of the fluid particle i , while m_k is the effective mass of a boundary particle assuming a boundary of density ρ_0 in Ω_{bdy} .

Since the entire boundary volume is represented by only a single layer of particles at the interface to the fluid, missing contributions and the variation in effective mass due to the non-uniform sampling must be accounted for. It is apparent that accurate values of m_k would have to be at least a function of the distance to the boundary in order to capture the fact that there are more missing boundary contributions in the kernel support of some fluid particle i if it is close to the boundary. This could be approximated by linearly extending the boundary density field via $m_k = \rho_0 - \frac{\rho_0 \cdot |\vec{x}_{ik}|}{\hbar}$, which could be computed using signed distance fields, storing the results on a grid and interpolating them to particle locations as proposed by [41, Koshier and Bender]. Instead, for the relatively small kernel support radius $\hbar = 2h$ in the expected fluid particle spacing h used in this report, a constant approximation that is not distance-dependent can be deemed sufficiently accurate and more computationally efficient, yielding the method of [37, Akinci et al.].

With this piece-wise constant model, the effective mass that achieves a boundary domain with density ρ_0 is $m_k = \rho_0 V_k$ where V_k is the effective volume of the boundary domain represented by particle k . This in turn can be computed using the SPH normalization criterion in Equation 2.14, which implies that $\sum_{l \in \mathcal{B}_k} V_l W_{kl} \stackrel{!}{=} 1$ ought to hold at any boundary particle k [37]. This means that the kernel sum effectively measures the volume error of a particle given a neighbour sampling: if the kernel sum is greater than one, all V_l ought to be correspondingly lower until a value of one is achieved and vice versa, which means [37]:

$$V_k = \frac{m_k}{\rho_0} \stackrel{!}{=} \frac{m_k}{\sum_{l \in \mathcal{B}_k} m_l W_{kl}} \approx \gamma_1 \frac{m_k}{m_k \sum_{l \in \mathcal{B}_k} W_{kl}} \quad (4.3)$$

$$\Rightarrow m_k = \gamma_1 \frac{\rho_0}{\sum_{l \in \mathcal{B}_k} W_{kl}} \quad (4.4)$$

where the parameter γ_1 in the approximation can be used to adjust the density contribution per boundary particle for differing kernel support sizes, choosing for example a value that ensures rest density for a uniformly sampled fluid resting on a uniformly sampled boundary plane. In this report, the simplest choice of $\gamma_1 = 1$ is made. Since the effective mass is assumed to be independent of distance to the boundary, m_k can be precomputed once and stored for each boundary particle.

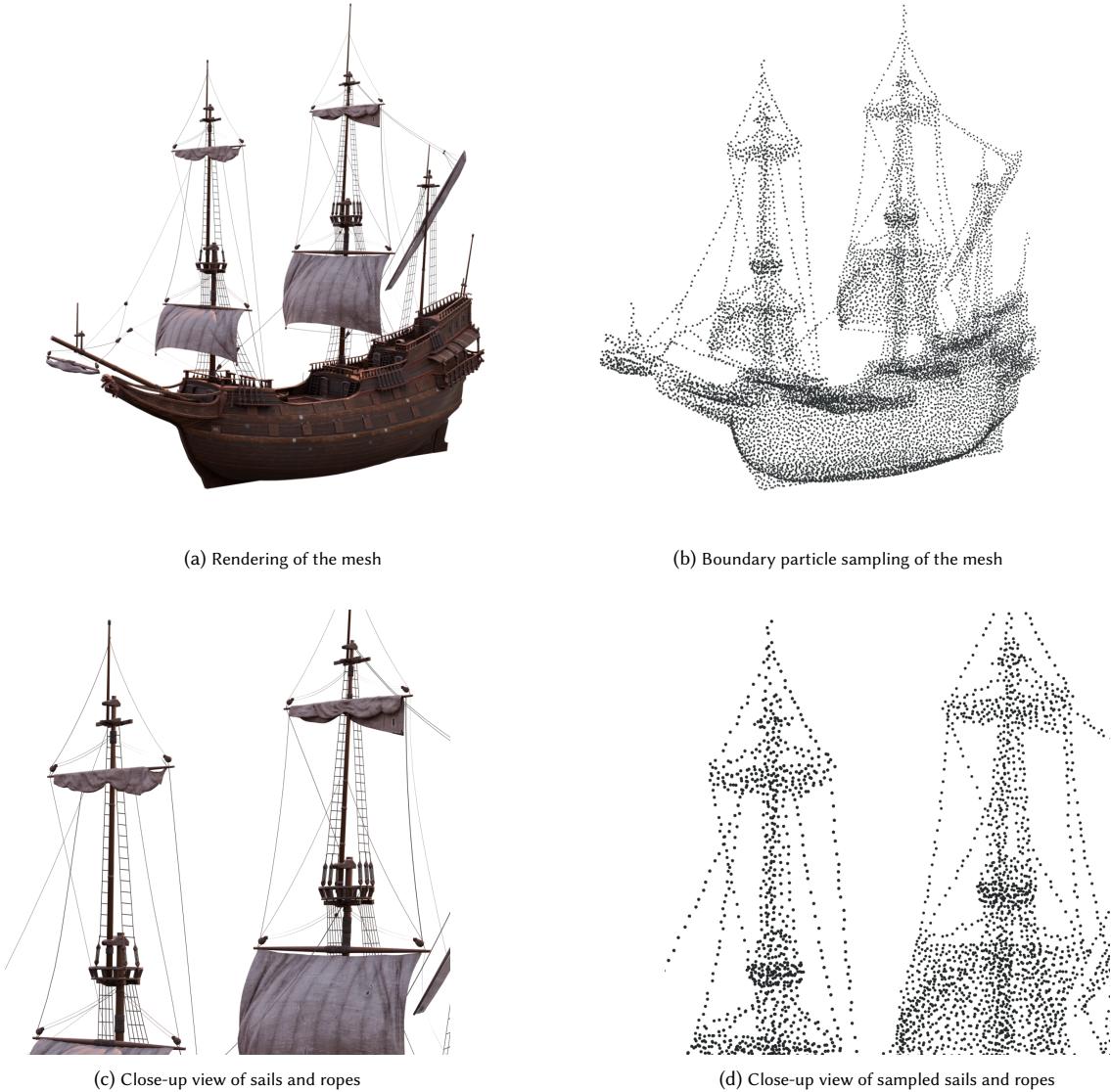


Figure 4.1: A rendering of a triangular mesh of a ship is shown in Figure 4.1a, with a close-up view of ropes and sails that are particularly challenging to sample in Figure 4.1c. The corresponding, low-discrepancy or blue noise samplings of boundary particles obtained by Poisson Disk sampling [40] are shown in Figure 4.1b and Figure 4.1d respectively, where particle positions are represented by small spheres. This highlights that the algorithm samples the mesh robustly even for complex and problematic geometries such as long, acute triangles in the ropes or non-manifold input geometry, at a resolution independent of that of the input mesh.

Instead of using the approximation of assuming roughly equal boundary masses in Equation 4.3, the exact equation can be seen as a system of equations and solved for m_k in the same fashion as discussed for initializing fluid particles with rest density in Equation 3.10:

$$m_k^0 = \frac{\rho_0}{\sum_{l \in \mathcal{B}_k} W_{kl}} \quad (4.5)$$

$$m_k^{l+1} \leftarrow m_k^l \cdot \frac{\rho_0}{\sum_{l \in \mathcal{B}_k} W_{kl}} \quad (4.6)$$

In the implementation of this report, the above system is solved once initially, using Equation 4.6, since the one-off cost of the iterative solver is acceptable, but subsequent recalculations of masses due to dynamic boundaries are handled with a single application of the approximate formula Equation 4.4 suggested by [37, Akinci et al.] to avoid additional runtime cost.

4.1.2 Dynamic and Procedural Boundaries

Not all boundaries are static: fluids might also interact with boundary particles representing dynamic rigid bodies simulated as described in section 4.2 or procedural boundaries, which behave according to some analytically specified equation of motion. In this implementation, procedural boundaries that implement rigid body translations were included, while rotations were neglected, since this limitation greatly simplifies the setting: given two equations $\vec{x}_{cm}(t), \vec{v}_{cm}(t)$, all boundary particles belonging to the same scripted object (or procedural boundary) are shifted from their initial position by exactly $\vec{x}_{cm}(t)$ and all have the same velocity $\vec{v}_{cm}(t)$ at any point t in time.

The previously discussed boundary handling can thankfully naturally handle intersecting boundaries [37]. For example, a procedural boundary plane moving sinusoidally to produce waves may intersect a container of water. However, the masses of boundary particles at such an intersection might need to be recalculated so that the pressure forces exerted on the fluid by the boundary do not change at the intersection and the boundary domain remains at rest density from the perspective of the pressure solver. This recalculation of effective masses is done according to two rules, which in conjunction conservatively estimate the set of boundary particles that should have their masses updated according to Equation 4.4 in each time step:

1. All masses of boundary particles belonging to scripted, procedural boundary objects are recalculated at every step. While performing the SPH kernel sum in Equation 4.4 necessary to do so, all neighbouring particles l have a counter set to 2 if they are static boundary samples.
2. All static boundary samples with such a counter value greater than zero have their masses recalculated and their counter decremented.

This means that not only are the masses of static boundary samples re-evaluated when procedural boundaries *enter* their kernel support radius \hbar , ensuring that all boundary particles remain at an effective rest density as observed by a neighbouring fluid particle, but the masses are also re-evaluated once more when a scripted boundary particle has just *left* the kernel support radius, ensuring it returns to the same effective mass m_k as before the intersection.

Note that in the Taichi language used for this implementation, a quantized data type can be used to represent the counter since only 3 bits per boundary particle are required, which can be densely packed to reduce memory usage and possibly improve performance, see [42, Hu et al.]

Extending the Governing Equations

With the surfaces of boundary domains sampled using a blue-noise distribution and the effective boundary particle masses m_k calculated such that they can be treated by the pressure solver like a fluid at rest with density ρ_0 , the SPH discretized governing equations from section 3.1 can be extended to include the boundary domain.

The density computation in Equation 3.4 is straightforward:

$$\langle \rho_i \rangle = \sum_{j \in \mathcal{N}_i} m_j W_{ij} + \sum_{k \in \mathcal{B}_i} m_k W_{ik} \quad (4.7)$$

The viscosity computation in Equation 3.2 remains the same as before. While viscous forces at the boundary can be implemented and the symmetric forces acting on boundary particles can be integrated into

the rigid body solver (see [37, Akinci et al.]), slip-conditions were chosen instead in this implementation, since rather inviscid flows at larger scales are investigated.

Lastly, the pressure computation must be adjusted. Equation 3.3 requires a density and pressure value per boundary particle k , which are unknown. Since the effective mass m_k was chosen to represent ρ_0 in the boundary domain, the rest density $\rho_k \approx \rho_0$ is a reasonable approximation. For choosing a pressure value of a boundary particle, there exist many methods, such as simply mirroring the pressure of the fluid particle under consideration $p_i = p_k$ [37], solving for pressures at boundary particles in the pressure solver explicitly to avoid inconsistent values of p_k for different fluid particles i [43], extrapolating the pressure field into the boundary domain using some scheme like *Moving Least Squares* [44], or simply setting the pressure at boundary particles to zero [5]. This choice can have a profound impact on pressure solver convergence and the quality of the resulting pressure field [5]. In accordance with recent literature by [5, Bender et al.], a so-called consistent boundary handling which imposes $p_k = 0$ is chosen in this instance, which is exceptionally easy to implement, was found to improve solver stability and yield a reasonably smooth pressure field. It is equivalent to the approach of mirroring pressure values as suggested by [37, Akinci et al.], but with a weighting of boundary pressure forces by a coefficient $\gamma_2 = 0.5$ [1]. Using $p_k = 0$, some terms in Equation 3.3 drop out, leaving the expression:

$$\vec{a}_i^p = \left\langle -\frac{1}{\rho_i} \nabla p_i \right\rangle_{\parallel} = - \sum_{j \in \mathcal{N}_i} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \nabla W_{ij} - \sum_{k \in \mathcal{B}_i} m_k \frac{p_i}{\rho_i^2} \nabla W_{ik} \quad (4.8)$$

This expression is used to derive the diagonal element in the IISPH solver of subsubsection 3.3 using the additional assertion that $\rho_i \stackrel{!}{=} \rho_j \stackrel{!}{=} \rho_0$. Since this discretization of \vec{a}_i^p makes use of the symmetric SPH approximation $\langle \cdot \rangle_{\parallel}$, one can easily derive the symmetric forces that the fluid particle i exerts on each boundary particle k using Newton's second law as [37]:

$$f_{k \leftarrow i} = -m_i \vec{a}_{i \leftarrow k}^p = m_i m_k \frac{p_i}{\rho_i^2} \nabla W_{ik} \quad (4.9)$$

which is used to exert forces and torques upon dynamic rigid bodies in section 4.2.

4.2 Two-Way Coupled Rigid Body Solver

Using the mass properties of rigid bodies, the computation of which is later outlined in subsection 4.2.1, the dynamics of unconstrained rigid bodies are discussed in the following, leading to a rigid solver that can be weakly coupled to the fluid solver developed in chapter 3. Two-way coupling is achieved through two interfaces between the fluid and rigid solvers:

1. By sampling the boundary of the rigid body as explained in section 4.1 and moving the boundary particles as the rigid body moves, contact with the fluid is resolved in the same way as for static boundaries which exhibit one-way coupling: pressure forces computed by the fluid's pressure solver in subsubsection 3.3 resolve contacts and keep the fluid from leaking into the rigid body.
2. On the other hand, the fluid must also exert forces and torques on the rigid body, influencing its movement and creating two-way coupling. This was alluded to in Equation 4.9, where the pressure force exerted on a boundary particle by a fluid particle was formulated.

These interfaces define the inputs and outputs of the rigid body solver. It takes a state of a rigid body and all the forces and torques that act on it, then computes an updated state of the body and finally moves all particles that sample the rigid body boundary to reflect the changes in the rigid body's state. These three steps are outlined in the following.

Forces and Torques

As per Chasles' theorem, rigid body motion can be described purely by the position and rotation of the body [45], since ideal rigidity means that the relative positions of each two points in the body do not change and no other deformations such as observed in elastic bodies or fluids can occur. This means that the dynamics can be split into the two separate problems of computing the linear and angular movement of the body [45]. This is conveniently done by using the centre of mass \vec{x}_{cm} as the origin which defines

body space. In this coordinate system, any point $\vec{x}'(t) := \vec{x}(t) - \vec{x}_{cm}$ belonging to the rigid body can be described using the equation of motion [46]:

$$\vec{x}(t) = \mathbb{O}(t)\vec{x}'(t=0) + \vec{x}_{cm}(t) \quad (4.10)$$

where \mathbb{O} encodes the current rotation about the centre of mass in a rotation matrix and $\vec{x}_{cm}(t)$ is the position of the centre of mass in world space. This means that all rotational aspects of the motion are encoded in the orientation \mathbb{O} , while the translational movement only affects the position and velocity of the centre of mass.

Accordingly, for the change in linear momentum $\vec{f}_{rigid} = \frac{d}{dt}M\vec{v}_{cm}$ and constant mass M it holds that all forces \vec{f}_i affecting the rigid body (in this instance the symmetric pressure forces in Equation 4.9) can simply be summed, including the external gravitational force acting on the body [46]:

$$\vec{f}_{rigid} = \vec{f}_g + \sum_i \vec{f}_i \quad (4.11)$$

irrespective of the position at which each force acts on the rigid body. This results in a single net translational force, which can be interpreted as an acceleration $\vec{a}_{cm} = \frac{1}{M}\vec{f}_{rigid}$ of the centre of mass \vec{x}_{cm} and integrated with respect to time, for example by using the semi-implicit scheme in Equation 3.5.

Meanwhile, the rate of change of the angular momentum $\frac{d}{dt}\vec{L} = \vec{\tau}_{rigid}$ defines the net torque $\vec{\tau}_{rigid}$, which can be computed as [46]:

$$\vec{\tau}_{rigid} = \sum_i \vec{\tau}_i = \sum_i (\vec{x}_i - \vec{x}_{cm}) \times \vec{f}_i \quad (4.12)$$

where the position \vec{x}_i in world space at which the force \vec{f}_i acts upon the rigid body is relevant. Intuitively, the vector $\vec{\tau}_i$ can be understood as pointing in the axis of the rotation that a force \vec{f}_i would cause if the rigid body was fixed at its centre of mass and left to spin [46].

Updating Rigid Body State

Using the total input force \vec{f}_{rigid} and torque $\vec{\tau}_{rigid}$, the state of the rigid body at the next time step $t + \Delta t$ must be computed from its current state. To obtain the updated position of the centre of mass $\vec{x}_{cm}(t + \Delta t)$ and the orientation $\mathbb{O}(t + \Delta t)$ required to solve the equation of motion (Equation 4.10), the corresponding differential quantities, namely the linear velocity \vec{v}_{cm} and angular velocity $\vec{\omega}$ are required and are integrated with respect to time.

The linear velocity required for updating the position of the centre of mass is straightforward and can again be obtained using semi-implicit time integration (Equation 3.5) from the linear acceleration described above as:

$$\vec{v}_{cm}(t + \Delta t) = \vec{v}_{cm}(t) + \Delta t \vec{a}_{cm}(t) = \vec{v}_{cm}(t) + \frac{\Delta t}{M} \vec{f}_{rigid}(t) \quad (4.13)$$

$$\vec{x}_{cm}(t + \Delta t) = \vec{x}_{cm}(t) + \Delta t \vec{v}_{cm}(t + \Delta t) \quad (4.14)$$

The angular velocity $\vec{\omega}$ points in the axis of rotation with a magnitude that encodes the speed of the revolution in radians per second [46]. Analogous to how the linear velocity is related through mass to the linear momentum, the angular velocity is related through the inertia tensor to the angular momentum [45]:

$$\vec{L}(t) = \mathbb{I}(t)\vec{\omega}(t) \quad (4.15)$$

$$\mathbb{I}(t)^{-1} \vec{L}(t) = \vec{\omega}(t) \quad (4.16)$$

where the angular momentum can be found by numerically integrating the torque with respect to time, since $\vec{\tau} = \frac{d}{dt}\vec{L}$ [46]:

$$\vec{L}(t + \Delta t) = \vec{L}(t) + \Delta t \vec{\tau}_{rigid}(t) \quad (4.17)$$

The inertia tensor (or its inverse) at some point in time t can be determined as [46]:

$$\mathbb{I}(t) = \mathbb{O}(t)\mathbb{I}(t=0)\mathbb{O}(t)^T \quad (4.18)$$

$$\mathbb{I}^{-1}(t) = \mathbb{O}(t)\mathbb{I}^{-1}(t=0)\mathbb{O}(t)^T \quad (4.19)$$

in terms of the initial inertia tensor $\mathbb{I}(t = 0)$ that is computed in subsubsection 4.2.1. Note that since \mathbb{O} is a rotation matrix, the transpose of the matrix \mathbb{O}^T is its inverse.

The orientation \mathbb{O} can be updated using:

$$\mathbb{O}(t + \Delta t) = \mathbb{O}(t) + \Delta t \dot{\mathbb{O}}(t) \quad (4.20)$$

where $\dot{\mathbb{O}}(t)$ or the rate of change of the orientation matrix is the skew-symmetrix matrix that encodes the angular velocity $\vec{\omega}(t) = (\omega_x, \omega_y, \omega_z)^T$, namely [46]:

$$\dot{\mathbb{O}}(t) = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \quad (4.21)$$

With this, the orientation can be updated and the angular velocity at the next time step can be estimated using Equation 4.16 as:

$$\vec{\omega}(t + \Delta t) = \mathbb{I}^{-1}(t + \Delta t) \vec{L}(t + \Delta t) \quad (4.22)$$

$$= \mathbb{O}(t + \Delta t) \mathbb{I}^{-1}(t = 0) \mathbb{O}(t + \Delta t)^T \vec{L}(t + \Delta t) \quad (4.23)$$

Orthonormalization of \mathbb{O} and Quaternions

In Equation 4.20, the orientation is updated by explicitly numerically integrating a differential change in orientation. Unfortunately, after this process, the orientation \mathbb{O} is no longer guaranteed to be a valid rotation matrix. Numerical drift can lead to accumulating errors, potentially leading to an unphysical scaling or skew of the rigid body [46], indicated by the determinant $\det(\mathbb{O})$ becoming different from one. To prevent this, \mathbb{O} could be re-orthonormalized every so often, using for example the Gram-Schmidt process. However, this would cause the accumulated error to be projected onto arbitrarily chosen axes that depend on the order of the basis vectors considered in the Gram-Schmidt process. A more elegant approach is choosing a different representation of orientations that can be normalized in a more physically meaningful way.

One such representation are quaternions, which extend the concept of complex numbers to four instead of two dimensions. A quaternion $s + xi + yj + zk$ has, in addition to the imaginary unit i , the basis vectors j and k , and can be represented by a single vector $\vec{q} = (s, x, y, z)^T$ in four dimensions or a pair $[s, \vec{v}]$ of a scalar s and a vector $\vec{v} = (x, y, z)^T$ holding its coefficients [46]. Using the second definition, quaternion multiplication can be written as [46]:

$$\vec{q}_1 \vec{q}_2 = [s_1, \vec{v}_1] [s_2, \vec{v}_2] \quad (4.24)$$

$$= [s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2, s_1 \vec{v}_2 + s_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2] \quad (4.25)$$

which is not commutative. If a quaternion has unit length, it represents a rotation and the product of two such quaternions represents the composite of each rotation in sequence [46], making them especially convenient in this context. It turns out that the rate of change of the orientation $\dot{\vec{q}}$ can be expressed in terms of the angular velocity $\vec{\omega}$ using quaternion multiplication as:

$$\dot{\vec{q}} = \frac{1}{2} [0, \vec{\omega}] [s, \vec{v}] \quad (4.26)$$

This can be explicitly numerically integrated:

$$\vec{q}(t + \Delta t) = \vec{q}(t) + \Delta t \dot{\vec{q}}(t) \quad (4.27)$$

and then normalized like any four-dimensional vector, so it remains a representation of a rotation. Finally, the resulting unit quaternion $\hat{\vec{q}}(t + \Delta t) = (s, x, y, z)^T$ can be converted to a rotation matrix $\mathbb{O}(t + \Delta t)$ using the relation [46]:

$$\mathbb{O} = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2sz & 2xz + 2sy \\ 2xy + 2sz & 1 - 2x^2 - 2z^2 & 2yz - 2sx \\ 2xz - 2sy & 2yz + 2sx & 1 - 2x^2 - 2y^2 \end{bmatrix} \quad (4.28)$$

which can be used as an auxiliary variable to compute $\mathbb{I}^{-1}(t + \Delta t)$ and $\vec{\omega}(t + \Delta t)$ as outlined above, but does not need to be stored, since \vec{q} is now used to represent the current orientation of the rigid body. Note that the initial orientation can be encoded by $\hat{\vec{q}}(t = 0) = (1, 0, 0, 0)^T$.

Updating the Dynamic Boundary Particles

Once a new orientation $\mathbb{O}(t + \Delta t)$ and position of the centre of mass $\vec{x}_{cm}(t + \Delta t)$ has been obtained, all boundary particles sampled on the surface of the rigid body as described in subsection 4.1.1 can be moved according to [46]:

$$\vec{r}_i(t + \Delta t) = \mathbb{O}(t + \Delta t)(\vec{x}_i(t = 0) - \vec{x}_{cm}(t = 0)) \quad (4.29)$$

$$\vec{x}_i(t + \Delta t) = \vec{r}_i(t + \Delta t) + \vec{x}_{cm}(t + \Delta t) \quad (4.30)$$

$$\vec{v}_i(t + \Delta t) = \vec{v}_{cm} + \vec{\omega}(t + \Delta t) \times \vec{r}_i(t + \Delta t) \quad (4.31)$$

using the initial positions at $t = 0$. In these equations, \vec{r}_i was introduced for readability and need not be stored.

All the steps necessary to perform a single update of a rigid body are summarized in Algorithm 5.

Algorithm 5 Rigid Body Solver Step

Step 1 – Compute Linear Movement

- 1: $\vec{v}_{cm} \leftarrow \vec{v}_{cm} + \Delta t \left(\vec{g} + \frac{1}{M} \sum_i \vec{f}_i \right)$ ▷ Equation 4.13
 - 2: $\vec{x}_{cm} \leftarrow \vec{x}_{cm} + \Delta t \vec{v}_{cm}$ ▷ Equation 4.14
-

Step 2 – Update Orientation

- 3: $\dot{\vec{q}} \leftarrow \frac{1}{2} [0, \vec{\omega}] \vec{q}$ ▷ Equation 4.26 using quaternion multiplication
 - 4: $\vec{q} \leftarrow \frac{\vec{q} + \Delta t \dot{\vec{q}}}{|\vec{q} + \Delta t \dot{\vec{q}}|}$ ▷ Equation 4.27 and normalize quaternion
 - 5: Calculate \mathbb{O} from \vec{q} ▷ Equation 4.28
-

Step 3 – Compute Rotational Movement

- 6: $\vec{L} \leftarrow \vec{L} + \Delta t \vec{\tau}_{rigid}$ ▷ using $\vec{\tau}_{rigid}$ from Equation 4.12
 - 7: $\vec{\omega} \leftarrow \mathbb{O} \mathbb{I}^{-1}(t = 0) \mathbb{O}^T \vec{L}$ ▷ Equation 4.22
-

Step 4 – Update Particles

- 8: **for** $i \in \mathbb{N}_0^{N_{rigid}-1}$ **in parallel do**
 - 9: $\vec{r}_{i,0} = (\vec{x}_i(t = 0) - \vec{x}_{cm}(t = 0))$ ▷ Initial body space position, auxiliary variable
 - 10: $\vec{x}_i \leftarrow \mathbb{O} \vec{r}_{i,0} + \vec{x}_{cm}$ ▷ Equation 4.30
 - 11: $\vec{v}_i \leftarrow \vec{v}_{cm} + \vec{\omega} \times (\mathbb{O} \vec{r}_{i,0})$ ▷ Equation 4.31
 - 12: **end for**
-

4.2.1 Exact Mass Moments of Watertight Meshes

In order to implement the rigid body solver in section 4.2, one requires the total mass M , centre of mass \vec{x}_{cm} , volume V and the inertia tensor \mathbb{I} of the rigid body being simulated. The algorithm used to compute these values is described in detail in the following section, with additional notes found in Appendix B.

As an input to this computation, some arbitrary triangular mesh is given, which can be approximated by a watertight manifold using the algorithm discussed in subsection 4.1.1, which is in turn interpreted as the surface $\partial\Omega_{rigid}$ of some volume Ω_{rigid} that represents a rigid body of homogeneous density ρ . In other words, the mass moments necessary for the computation of $M, \mathbb{I}, \vec{x}_{cm}$ etc. of the closed triangular mesh of $\partial\Omega_{rigid}$ must be computed, in this case using a method by [47, Zhang and Chen], which is outlined in the following. The mass density moments of Ω_{rigid} are defined as an integral [47]:

$$\mathcal{M}_{pqr} := \iiint_{\Omega_{rigid}} \rho x^p y^q z^r dx dy dz \quad (4.32)$$

over the volume of the rigid body for some p, q, r . The 'zero-th' moment $p = q = r = 0$ is the total mass of the object, the first moment $p = q = r = 1$ is used to calculate the centre of mass, and the entries of the inertia tensor can be computed from some other $p, q, r \in \mathbb{N}_0^2$ that are discussed in subsubsection 4.2.1.

Volume Integrals as Sums over Triangles

Using the fact that the input triangular mesh is now a closed manifold, there exists a method for computing the analytic mass moment of the mesh.

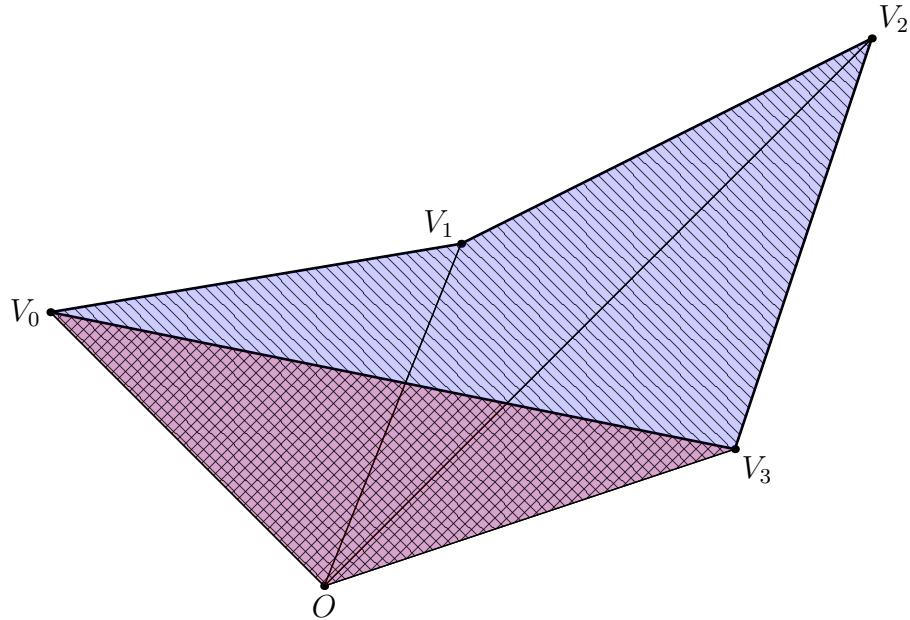


Figure 4.2: In a figure similar to the one by [47, Zhang and Chen], a volume integral using the described method of signed volumes is illustrated for a polygon with vertices V_0, V_1, V_2, V_3 and an additional vertex O at the origin. Whereas in three dimensions, volumes of tetrahedra are added, and we sum over the triangles of a mesh, in this two-dimensional setting the areas of triangles are added in a sum over edges of the mesh. Positive signed areas are coloured blue and hatched along one diagonal, negative areas are red and hatched along the other diagonal. The purple, cross-hatched area outside the mesh is therefore added once and subtracted once, resulting in a net contribution to the total area of zero, while the area inside the polygon does not cancel out and remains as the result of the computation.

The method makes use of the notion of signed volumes V' of tetrahedra, where given some vertex O of a tetrahedron and the triangle \triangle_{ABC} formed by its remaining, ordered vertices, it is assigned either a positive or a negative volume depending on whether the normal of the triangle \triangle_{ABC} faces towards or away from O . Note that since the input mesh is now a watertight manifold, each connected component of the mesh defines an interior and an exterior, consists of triangles and can be stored in a consistent winding order such that the normals $\frac{\overrightarrow{AB} \times \overrightarrow{AC}}{|\overrightarrow{AB} \times \overrightarrow{AC}|}$ all point towards the exterior, as is convention, or all point towards the interior. In this instance, counter-clockwise winding was used, but differing input meshes can be accounted for by flipping the sign of the signed volume accordingly.

More traditionally, the integration domain Ω_{rigid} could be partitioned into a set of disjoint tetrahedra $\{\Delta_i\}$ such that $\Omega_{rigid} = \bigcup_i \Delta_i$ and $\forall i, j : \Delta_i \cap \Delta_j = \emptyset$. This means that the integral could be written as:

$$\int_{\Omega_{rigid}} dV = \sum_{\Delta_i} V(\Delta_i) \quad (4.33)$$

where $V(\Delta_i)$ is the volume of the i -th tetrahedron that can be computed analytically from its vertices. This would necessitate a decomposition of the interior of the boundary mesh into tetrahedra that is neither trivial nor particularly efficient.

Instead, one fixed point O anywhere in space can be chosen to construct tetrahedra Δ_{ABCO} by simply summing over all the triangles Δ_{ABC} in the mesh and appending the vertex O to form a tetrahedron. These tetrahedra no longer partition space, so a simple sum over their volumes would not yield the correct total volume of the domain but some possibly larger value. However, the crucial insight lies in the fact that summing up the signed volumes $V'(\Delta_{ABCO})$ with a consistent winding order as described above results in multiply accounted for volumes outside Ω_{rigid} exactly cancelling out, leaving only the volume inside Ω_{rigid} , which is accounted for an odd number of times with alternating signs. This is conceptually similar to how parity of the number of intersections along a ray is used to determine if a point lies inside a polygon as per the Jordan Curve theorem in subsection 3.1, only the parity of faces with normals oriented towards or away from some vertex O is now used to determine subvolumes inside or outside the mesh. Figure 4.2 might make this concept clearer by illustrating a concrete example in two dimensions.

Since the fixed point O is arbitrary, the most computationally efficient choice is $O = (0, 0, 0)^T$. With

this choice, the signed volume of any tetrahedron \triangle_{ABCO} can be defined as [47, 48]:

$$V'(\triangle_{ABCO}) = \frac{1}{6} \det(J(\triangle_{ABCO})) \quad (4.34)$$

$$= \frac{1}{6} \det \left(\begin{bmatrix} A_x & B_x & C_x \\ A_y & B_y & C_y \\ A_z & B_z & C_z \end{bmatrix} \right) \quad (4.35)$$

$$= \frac{1}{6} (\vec{A} \times \vec{B}) \cdot \vec{C} \quad (4.36)$$

where J denotes the Jacobian of the tetrahedron and the position vector of each vertex $\overrightarrow{OA} = \vec{A}$ etc. can be used in the Jacobian since O was chosen to be the origin. The triple scalar product used to compute the determinant is anticommutative, so the order of any two of the vectors $\vec{A}, \vec{B}, \vec{C}$ in the expression above can be swapped to alternate the sign of the result, for example to account for a different winding order convention. With this, the volume integral can be written as a sum [47]:

$$V(\Omega_{rigid}) = \int_{\Omega_{rigid}} dV = \left| \sum_{\triangle_{ABC}} V'(\triangle_{ABCO}) \right| \quad (4.37)$$

From now on, it will be assumed that the winding order and the order of the vectors in Equation 4.36 were chosen such that the sum of signed volumes are always positive and the absolute value $|\cdot|$ in Equation 4.37 could be dropped. In other words, $\sum_{\triangle_{ABC}} V'(\triangle_{ABCO})$ being a positive quantity defines some function $\text{sgn}(\triangle_{ABCO}) \in \{-1, 1\}$, which assigns a positive or negative weight to any tetrahedron \triangle_{ABCO} , yielding the sign of its signed volume V' .

One of the greatest properties of this method, apart from its exact accuracy for closed meshes, is that it generalizes to any mass moments as defined in Equation 4.32 through the relation [47]:

$$\mathcal{M}_{pqr} = \sum_{\triangle_{ABC}} \left(\text{sgn}(\triangle_{ABCO}) \cdot \iiint_{\Omega(ABCO)} \rho x^p y^q z^r dV \right) \quad (4.38)$$

where $\Omega(ABCO)$ is the domain of the tetrahedron \triangle_{ABCO} , for which exact analytic solutions to the triple integral exist when homogeneous density is assumed, as outlined in Appendix B.

Zeroth Mass Density Moment: Total Mass

It follows that the total mass of a rigid body of homogeneous density ρ is:

$$M(\Omega_{rigid}) = \int_{\Omega_{rigid}} \rho dV = \mathcal{M}_{000} = \sum_{\triangle_{ABC}} V'(\triangle_{ABCO}) \rho =: \sum_{\triangle_{ABC}} M'(\triangle_{ABCO}) \quad (4.39)$$

which can be efficiently and conveniently computed for each face of the mesh in parallel and summed up using a parallel reduction.

First Mass Density Moments: Centre of Mass

Describing changes of linear momentum of a rigid body is conveniently done by applying forces to the centre of mass, which is defined as:

$$\vec{x}_{cm}(\Omega_{rigid}) = \frac{1}{M(\Omega_{rigid})} \int_{\Omega_{rigid}} \vec{x}' \cdot \rho(\vec{x}') dV(\vec{x}') = \frac{1}{\mathcal{M}_{000}} \begin{bmatrix} \mathcal{M}_{100} \\ \mathcal{M}_{010} \\ \mathcal{M}_{001} \end{bmatrix} \quad (4.40)$$

Since the centre of mass of any tetrahedron is its centroid, the signed centre of mass of each tetrahedron in the sum can be given in terms of the signed mass $M'(\triangle_{ABCO})$ from Equation 4.39:

$$\vec{x}_{cm}(\triangle_{ABCO}) = \frac{M'(\triangle_{ABCO})}{4} (\vec{A} + \vec{B} + \vec{C}) \quad (4.41)$$

which means the centre of mass of the entire rigid body can be computed as:

$$\vec{x}_{cm} = \frac{1}{M} \sum_{\triangle_{ABC}} \vec{x}_{cm}(\triangle_{ABCO}) \quad (4.42)$$

Second Mass Density Moments: Inertia Tensor

To describe how torques create rotations of rigid bodies, the inertia of the body about any axis has to be known. The inertia can be thought of as the quotient of the angular momentum \vec{L} and the angular velocity $\vec{\omega}$, yielding a second rank tensor \mathbb{I} which is a 3×3 matrix in three dimensions with diagonal elements called *moments of inertia* and non-diagonals referred to as *products of inertia* [45]. The eigenvectors of the inertia tensor are the principal axes of rotation; it is a symmetric tensor with six degrees of freedom and is defined as [49, 45]:

$$\mathbb{I} = \begin{bmatrix} \int_{\Omega} \rho(x, y, z) (y^2 + z^2) dV & -\int_{\Omega} \rho(x, y, z) xy dV & -\int_{\Omega} \rho(x, y, z) xz dV \\ -\int_{\Omega} \rho(x, y, z) yx dV & \int_{\Omega} \rho(x, y, z) (x^2 + z^2) dV & -\int_{\Omega} \rho(x, y, z) yz dV \\ -\int_{\Omega} \rho(x, y, z) zx dV & -\int_{\Omega} \rho(x, y, z) zy dV & \int_{\Omega} \rho(x, y, z) (x^2 + y^2) dV \end{bmatrix} \quad (4.43)$$

Note that the symmetry $I_{ab} = I_{ba}$ results in the six degrees of freedom and that for homogeneous density, ρ can be moved out of the integral. Further, since $\int_{\Omega} x^2 + y^2 dV = \int_{\Omega} x^2 dV + \int_{\Omega} y^2 dV$, the inertia tensor can be written in terms of the mass density moments \mathcal{M}_{pqr} as:

$$\mathbb{I} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \quad (4.44)$$

$$I_{xx} = \mathcal{M}_{200} \quad (4.45)$$

$$I_{yy} = \mathcal{M}_{020} \quad (4.46)$$

$$I_{zz} = \mathcal{M}_{002} \quad (4.47)$$

$$I_{xy} = \mathcal{M}_{100} + \mathcal{M}_{010} \quad (4.48)$$

$$I_{xz} = \mathcal{M}_{100} + \mathcal{M}_{001} \quad (4.49)$$

$$I_{yz} = \mathcal{M}_{010} + \mathcal{M}_{001} \quad (4.50)$$

where Equation 4.38 can be applied to calculate each invocation of \mathcal{M} efficiently. In Equation 4.38, the analytic solution to volume integrals over tetrahedra Δ_{ABCO} are required, which in this instance have the form:

$$\rho \int_{\Omega(ABCO)} x^2 + y^2 dV \quad (4.51)$$

$$\rho \int_{\Omega(ABCO)} xy dV \quad (4.52)$$

and analogously for the other two axes. These exact analytic expressions are cited from [48, Tonon] and moved to Appendix B since they are rather verbose. Using Equation B.19, the final inertia tensor \mathbb{I} can be computed. With this, all computations required for the implementation of the simple rigid body solver discussed in section 4.2 can be implemented.

VISUALIZATION

A crucial challenge in the application of fluid solvers based on SPH to Computer Graphics is the visualization of the simulation results. Even if a simulation is conducted only to obtain the resultant raw data, as sometimes desired in engineering, some form of visual inspection of the results is invariably part of the process of fluid simulation.

While subsection 4.1.1 is concerned with sampling triangular meshes in order to obtain a particle-based representation, the visualization of the simulated scene often requires the opposite process, a reconstruction of triangular meshes for rendering from particle data. To this end, simulated rigid bodies must be correctly displayed, the surface of the fluid in the case of hydrodynamic simulations must be reconstructed (unlike the simulation of e.g. air, which would require some other form of visualization) and diffuse particles representing spray, foam and air bubbles that are particularly prominent in large-scale liquid simulations can be generated to greatly improve the visual realism of the scene [50], all of which is described in the following chapter. The focus of this report lies on the synthesis of photorealistic videos of virtual scenes, while scientific visualizations such as sensor planes (as used in e.g. [51, this paper]) or streamline plots are not discussed.

Note that the time step size Δt of the simulation is typically on scales of temporal resolution much finer than what is required to display a smooth video ($\Delta t \ll 16\mu\text{s} = \Delta t_{frame}$ for 60 frames per second), which means that only relatively few points in time must be visualized, which reduces computational cost. For sufficiently small step sizes Δt , one can simply visualize the simulation data at each of the smallest values of t larger than some integer multiple of the frame time Δt_{frame} . Other methods, such as interpolating the simulation results temporally to the exact point in time shown in the rendered frame are also conceivable.

Visualizing Boundaries

The visualization of boundaries is trivial in the case of static boundaries: exactly the triangular mesh that served as input to the particle sampling, e.g. as described in subsection 4.1.1, can be used for visualization.

Procedural boundaries as described in subsection 4.1.2 are similarly unproblematic as they simply require a translational offset defined by $\vec{x}_{cm}(t)$ or an additional rotation about their centre of mass (which was not discussed in this report).

Dynamic rigid bodies on the other hand, if implemented as described in section 4.2 should be noted to be described in body space with the centre of mass as the origin, not in the global coordinate system. Since the transformations in question do not generally commute, this means that before the rotation encoded by the orientation matrix O , or equivalently the quaternion \vec{q} , can be applied to the body, the centre of mass of the object that the rotation refers to must be set as the origin. For a body with an initial centre of mass $\vec{x}_{cm}(t=0)$ in global coordinates, one must:

1. translate the centre of mass to the global origin by shifting the object by $-\vec{x}_{cm}(t=0)$
2. apply the rotation encoded by $O(t)$ or $\vec{q}(t)$
3. translate the rotated body to its current position by adding $\vec{x}_{cm}(t)$

for every vertex of the input mesh at $t=0$. Since only rigid body dynamics are described in this report and deformations of the mesh therefore do not occur, this procedure suffices to render the effects of the solver in Algorithm 1.

5.1 Surface Reconstruction

The fluid is less trivial to visualize, since an explicit representation of the fluid surface is not usually part of the simulation and must instead be reconstructed from the explicitly computed quantities in a

post-processing step. Although methods that [52, explicitly operate on a surface representation] exist, most simulation methods in which field quantities are computed throughout the continuum rely on an isosurface reconstruction of a scalar field to construct a surface mesh from a set of particle positions. In this report, the `splashsurf` utility implementing [53, Weighted Laplacian Smoothing for Surface Reconstruction] was used, the main concepts of which are outlined in the following.

Note that the presented method is a more recent addition to a long line of research into reconstruction of surfaces from particle data and many alternative solutions exist. This includes early [54, implicit, Gaussian point-splatting] approaches, the later use of [55, Marching Cubes] directly applied to whether some grid cell contains a particle or not [15], then its application to a smoothed field with added post-processing [56], more elaborate approaches like [57, anisotropic kernels] that better capture detail and smooth out planar regions, or methods that replace Marching Cubes with a different isosurface reconstruction such as [58, dual contouring] entirely, trading possibly non-manifold results for the flexibility of not relying on uniform grids, amongst other considerations.

The results of [53, Weighted Laplacian Smoothing] compared to a regular [55, Marching Cubes] reconstruction are illustrated in Figure 5.1, highlighting the advantages of the following method.

It is intuitively clear that the density field, for example, is suited to indicating whether any point in space lies within or outside the fluid: a density near the rest density of the fluid certainly implies that a point is inside the fluid volume, while a density close to zero certainly belongs to a point outside the fluid. The fluid surface can then be approximated by an isosurface of some threshold value t , which can be set ad-hoc to yield visually plausible results. Assuming that particles have (approximately, see subsubsection 3.1) uniform sizes and masses, a commonly used alternative to the density field that is suited for post-processing scenarios where only particle positions are known is the so-called **COLOUR FIELD**, which can be defined as the standard SPH discretization (Equation 2.13) of the indicator function of the fluid volume and which is therefore closely related to the density field [53]:

$$\langle 1 \rangle(\vec{x}) = \sum_{j \in \mathcal{N}_{\vec{x}}} \frac{m_j}{\rho_j} W(\vec{x} - \vec{x}_j, h) \quad (5.1)$$

Recall from section 2.2 that the SPH interpolation scheme is valid not only at particle positions \vec{x}_i but at any position \vec{x} in space, which is used in the definition of the colour field in Equation 5.1 for any point in space \vec{x} . This means that a common building block in surface reconstruction can be applied here, namely the **MARCHING CUBES ALGORITHM** [55]. This algorithm subdivides space into a uniform grid of cubes and classifies each of the resulting vertices as *inside* or *outside* of a volume depending on whether a scalar field at the vertex position is above or below the threshold t . It follows that the surface of the volume must lie between each adjacent pair of vertices that have differing in-out classifications, resulting in $2^3 = 256$ possible triangle topologies per 3-dimensional cube under consideration [55]. The corresponding triangle topology (which can be reduced to fewer cases due to symmetry) can simply be stored in a table and looked up. The approximate position of the surface vertex along each edge of the uniform grid can be found through linear interpolation. More concretely this means that the sign of a potential [53]:

$$\Phi(\vec{x}) = \langle 1 \rangle(\vec{x}) - t \quad \text{see Equation 5.1} \quad (5.2)$$

the zero-set of which is the surface, can be used to classify vertices as inside or outside $(\vec{x}_{in}, \vec{x}_{out})$. The surface vertex \vec{x}_s between two such vertices can be approximated via linear interpolation as [53]:

$$\vec{x}_s = \vec{x}_{out} + \frac{t - \Phi(\vec{x}_{out})}{\Phi(\vec{x}_{in}) - \Phi(\vec{x}_{out})} (\vec{x}_{in} - \vec{x}_{out}) \quad (5.3)$$

Even if the Marching Cubes algorithm is executed at a resolution much finer than the expected particle spacing h so that individual particles and splashes are resolved, the result can unfortunately include many *bumps* resulting from the limited resolution of both the particle sampling and the reconstruction. To combat this, especially in regions that are supposed to represent flat, planar regions, explicit **LAPLACIAN SMOOTHING** can be applied to the resulting surface mesh [53]. In this procedure, vertices of the surface mesh obtained from marching cubes are interpolated using some weight $\lambda \in [0; 1]$ towards the average position of all vertices that are immediate edge-neighbours of the vertex under consideration. While a diffusion process with uniform weights does result in less curvature, smoothing out planar regions of the

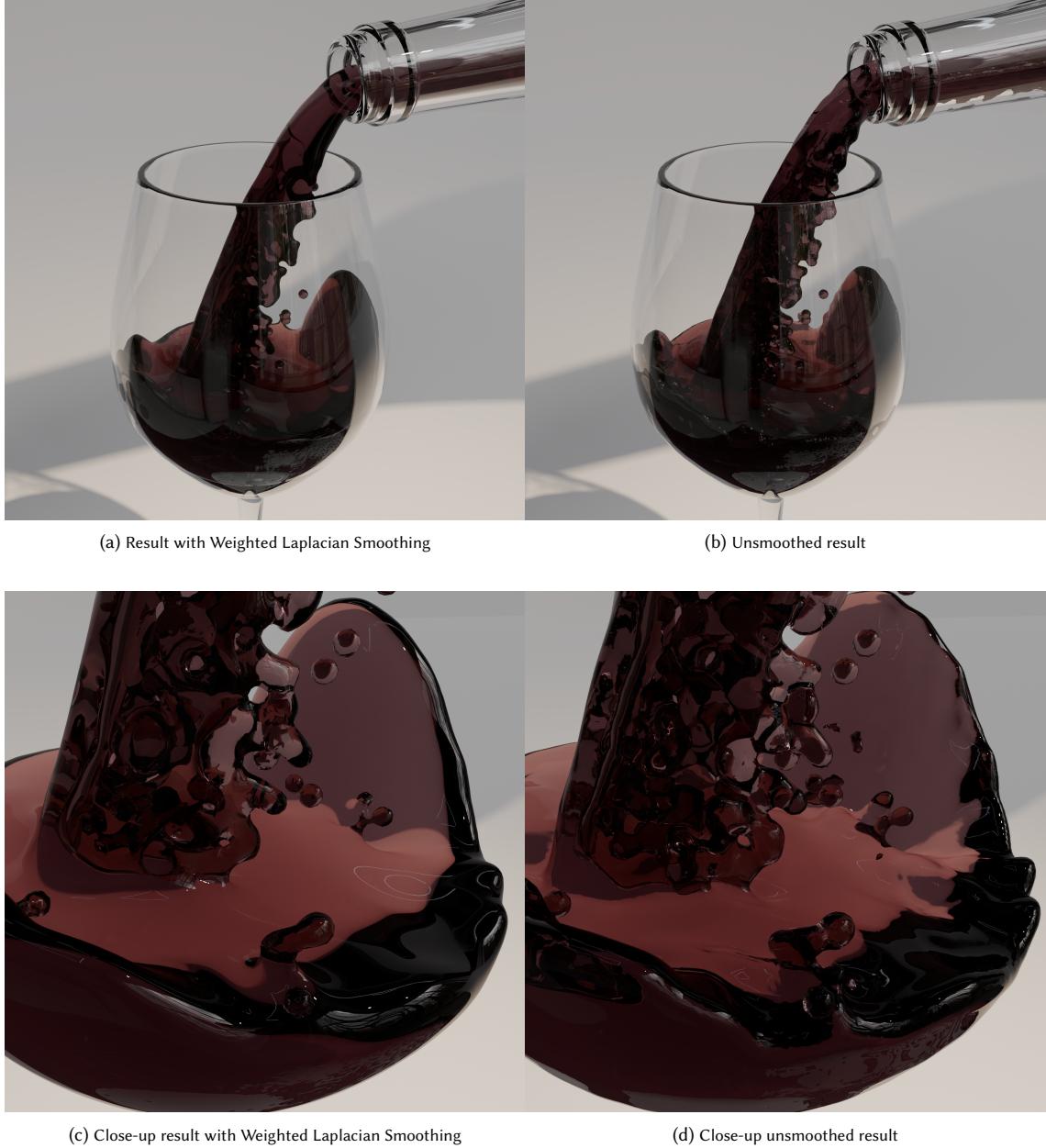


Figure 5.1: Renderings of a fluid simulation using the IISPH fluid solver in Algorithm 1 of a glass of red liquid being poured are shown, reconstructed with Weighted Laplacian Smoothing in Figure 5.1a and Figure 5.1b on the left side and a basic marching cubes algorithm in Figure 5.1b and Figure 5.1d on the right side. The top row shows that results might appear similar at first glance, but the fluid flow in Figure 5.1a in the vicinity of the bottle's opening appears more laminar, whereas artefacts of the Marching Cubes algorithm, resulting in bumps, make the flow in Figure 5.1b appear more irregular. Due to the refraction applied to the material of the liquid in rendering, the smoothed result appears more translucent, whereas small surface bumps cause more unpredictable light bounces and therefore more volumetric absorption in Figure 5.1b, making the stream appear darker. The close-up view in Figure 5.1c shows a smooth surface, while Figure 5.1d makes these small bumps clearly visible. Note that the smoothing does *not* cause individual droplets in Figure 5.1c to lose volume or even collapse, retaining detail up to the actual particle resolution, while volume in more planar regions is indeed reduced in the smoothing process as compared to Figure 5.1d.

surface, it can also cause volume shrinkage, especially for droplets and splashes that may collapse in on themselves [53]. Instead, as the name of the [53, Weighted Laplacian Smoothing] implies, varying weights λ can be employed that produce more smoothing in planar regions, where higher neighbour counts of particles can be measured, and less smoothing for isolated particles and splashes, where particles i have a smaller number of neighbours $|\mathcal{N}_i|$ [53]. The neighbour count can be interpolated to vertex positions via standard SPH interpolation $\langle \cdot \rangle$ (Equation 2.13), clamped to a maximum value N_{HS} that is achieved in a planar, half-space particle configuration and finally a smooth step function such as $S(x) = 6x^5 - 15x^4 + 10x^3$ can be used to distribute the resulting values in the $[0; 1]$ interval required of the weights λ , resulting in [53]:

$$\langle |\mathcal{N}| \rangle = \sum_{j \in \mathcal{N}_{\vec{x}}} \frac{m_j}{p_j} |\mathcal{N}_j| W(\vec{x} - \vec{x}_j, \hbar) \quad (5.4)$$

$$\lambda(\vec{x}) = S\left(\max\left(1, \frac{\langle |\mathcal{N}| \rangle}{N_{HS} \langle 1 \rangle}\right)\right) \quad (5.5)$$

With these weights, each vertex \vec{x}^k in the k -th smoothing iteration can be moved towards the average position $\bar{\vec{x}}$ of neighbouring vertices using [53]:

$$\vec{x}^{k+1} = \vec{x}^k + \lambda \bar{\vec{x}} \quad (5.6)$$

yielding the final smoothing procedure applied to the Marching Cubes reconstruction.

The [53, original paper] goes more into depth about implementation details, as well as improvements to temporal coherence of resulting meshes in animations and an improved subsequent mesh decimation procedure. This report discusses only the main principles required for a basic implementation and uses the splashsurf utility by the same authors as an optimized implementation.

5.2 Spray, Foam and Bubble Generation

In reality, large masses of water clashing against solid obstacles result in a misty spray, waves carry patches of foam that might slowly dissolve and moving water may trap air, creating air bubbles. Since water-air mixture effects originating from free surfaces are prevalent, especially for masses of water on the scale of meters and beyond, emulating such effects in simulations can have a significant impact on the perceived realism of an animated scene [59]. An example of such an improvement in perceived realism is shown in Figure 5.2.

To this end, for particle-based simulations and SPH fluid simulations in particular, a popular method was introduced by Ihmsen et al. in their paper [59, Unified Spray, Foam and Bubbles for Particle-Based Fluids], with suggestions for improvement addressed by [60, Bender et al.] in the context of micropolar SPH fluids, which were implemented for this report and are outlined in the following. No explicit interactions between diffuse particles are simulated, which makes the process trivially parallelizable and scale into millions of diffuse particles without prohibitive computational cost. The diffuse particles are instead generated, advected and destroyed as a post-processing step using the particle positions and velocity field obtained from the fluid simulation, according to physically and visually plausible but relatively ad-hoc criteria that have proven effective in practice [60, 50].

Three main processes are considered in the generation of diffuse particles where water and air mix, resulting in so-called potentials I that are used to determine the number of particles spawned [59]:

1. Whitewater is created at wave crests, characterized by a convex surface with strong curvature, captured by the potential I_{wc}
2. Air is trapped into bubbles and dragged underwater at sites of impact, where water clashes with particles of sufficiently opposing velocity, measured by a potential I_{ta}
3. More spray, foam and bubbles are created when the formerly mentioned effects occur with higher kinetic energy of the water, while slow-moving water has a smaller potential I_{kin} for water-air mixture. Higher kinetic energy equates for a simplified, constant surface tension to a higher Weber number, causing more interaction of the two phases [59]



Figure 5.2: Rendering comparison of a scene with a large dynamic battleship floating in water by a coast, moved by waves that crash upon the shore - while the right-hand Figure 5.2b and Figure 5.2d are rendered with the addition of diffuse particles representing spray, foam and air bubbles, Figure 5.2a and Figure 5.2c neglect such effects. The rendering with added diffuse particles can appear much more visually plausible, faithfully rendering the dynamicity of the scene, where patches of foam act as a visual clue to the movement present in the ocean. The plain rendering without spray, foam and bubbles on the other hand can seem still, since there is less visual indication of the dynamics of the waves and water surface, making the liquid appear too calm or even viscous. The effect is especially prominent at the shoreline and around the stern of the ship.

Generating Potentials

To identify strongly convex areas, the surface normal can be approximated using the negative SPH kernel gradient sum as:

$$\vec{n}_i = - \sum_{j \in \mathcal{N}_i} \nabla W_{ij} \quad (5.7)$$

which can be normalized to obtain $\hat{\vec{n}}_i$ or set to $\vec{0}$ if the magnitude $|\vec{n}|$ is insufficiently large, such as inside the fluid volume. This basically leverages the particle deficiency phenomenon discussed in subsection 2.2 to detect surface normals from the incomplete particle samplings in the kernel support of particles at the free surface.

Since such irregular sampling at the free surface is not only expected but relied upon, an alternative Kernel function K that better handles such scenarios is used instead of the cubic spline kernel W (Equation 2.22, Equation 2.25) otherwise employed in this report, which is designed to be accurate within the fluid volume [59]. The kernel is additionally normalized as per [60, later suggestions], which makes the process more scale-invariant:

$$K(\vec{x}_{ij}, \hbar) = \underbrace{\frac{3}{\pi \hbar^3}}_{\text{normalization}} \cdot \underbrace{1 - \frac{|\vec{x}_{ij}|}{\hbar}}_{\text{linear kernel}} \cdot \underbrace{\Theta(\hbar - |\vec{x}_{ij}|)}_{=1 \text{ if } |\vec{x}_{ij}| < \hbar \text{ else } 0} \quad (5.8)$$

Using this kernel, the curvature at some fluid particle i can be measured and weighted using a step function Θ such that only convex curvature contributes to a value κ [59, 60]:

$$\kappa_i = \sum_{j \in \mathcal{N}_i} \underbrace{(1 - \hat{\vec{n}}_i \cdot \hat{\vec{n}}_j)}_{\text{measure curvature}} \cdot K_{ij} \cdot \underbrace{\Theta(\vec{x}_{ij} \cdot \hat{\vec{n}}_i)}_{\text{only when convex}} \quad (5.9)$$

To identify wave crests, an additional criterion is used that excludes other strongly convex areas such as edges of the liquid volume: the particle under consideration should significantly move in the direction of the surface normal, i.e. [59]:

$$\delta_i = \kappa_i \cdot \Theta(\hat{\vec{v}}_i \cdot \hat{\vec{n}}_i - 0.6) \quad (5.10)$$

where $\hat{\vec{v}}_i$ is the normalized velocity of particle i and the threshold of 0.6 is set ad-hoc.

The magnitude of δ_i is rather arbitrary and scene-dependent, so it is clamped between minimum and maximum values and normalized to a percentage in $[0; 1]$ for subsequent computations, using a clamping and normalization function Φ like:

$$\Phi(x, \tau_{min}, \tau_{max}) = \frac{\min(x, \tau_{max}) - \min(x, \tau_{min})}{\tau_{max} - \tau_{min}} \quad (5.11)$$

where $\tau_{max} \geq \tau_{min}$ are user-defined.

This yields a final generating potential for wave-crest diffuse particles of:

$$I_{wc} = \Phi(\delta_i, \tau_{min}^{wc}, \tau_{max}^{wc}) \quad (5.12)$$

Similarly, a potential in $[0; 1]$ can be derived that captures the effect of air bubbles becoming entrapped by a liquid where free surfaces of the liquid phase collide with sufficiently opposing velocities. The basis of this potential is simply the scaled velocity difference [59]:

$$v_i^{diff} = \sum_{j \in \mathcal{N}_i} |\vec{v}_{ij}| \left(1 - \hat{\vec{v}}_{ij} \cdot \hat{\vec{x}}_{ij}\right) K_{ij} \quad (5.13)$$

$$I_{ta} = \Phi(v_i^{diff}, \tau_{min}^{ta}, \tau_{max}^{ta}) \quad (5.14)$$

Lastly, the kinetic energy of a particle is used as basis of a potential. To make the process more resolution-invariant, in this report the mass is normalized using the expected rest mass given the particle spacing h and a rest density ρ_0 , which yields:

$$E_{kin,i} = \frac{1}{2} \underbrace{\frac{m_i}{\rho_0 h^3}}_{=\frac{m_i}{m_0}} |\vec{v}_i|^2 \quad (5.15)$$

$$I_{kin} = \Phi(E_{kin,i}, \tau_{min}^{kin}, \tau_{max}^{kin}) \quad (5.16)$$

Spawning Diffuse Particles

These potentials can then be used to determine the number of diffuse particles n_d spawned by some fluid particle at some time step [60]:

$$n_d = k \cdot I_k \frac{I_{ta} + I_{wc}}{2} \Delta t \quad (5.17)$$

where Δt is not necessarily related to the simulation's time step size but rather the simulated time between frames in which foam is generated (e.g. 60 such frames per simulated second might be required for video) and the factor k determines the maximum number of diffuse particles spawned per second of simulated time by each fluid particle [60]. Note that the kinetic energy influences both the amount of foam on wave crests and the amount of trapped air where liquids collide, which is reflected in its multiplicative contribution to the sum of other potentials in Equation 5.17. Individual parameters k_{ta}, k_{wc} as suggested in the [59, original paper] are set to $\frac{1}{2}$ and therefore effectively [60, foregone, as later suggested].

The resulting $n_d \in \mathbb{R}$ must be converted to an integer to be a valid number of diffuse particles to spawn. While this is not discussed in any of the mentioned papers [59, 60], it is suggested in this report to use stochastic rounding:

$$N_d = \begin{cases} \lfloor n_d \rfloor + 1 & \text{if } \xi_{n_d} < n_d - \lfloor n_d \rfloor \\ \lfloor n_d \rfloor & \text{otherwise} \end{cases} \quad (5.18)$$

where $\xi_{n_d} \sim \mathcal{U}([0; 1])$ is drawn from a uniform distribution and $\mathbb{E}[N_d] = n_d$ is correct in expectation. To illustrate why this might be much more accurate than the usual, deterministic rounding per-particle, imagine millions of fluid particles with a value of $n_d = 0.49$ compared to the same particles with a slightly increased $n_d = 0.51$ each - the diffuse particle count would spontaneously explode as an artefact of the rounding scheme, while it behaves as expected when using stochastic rounding.

Now that a physically motivated number $N_d \in \mathbb{N}$ of diffuse particles to generate for each fluid particle and time step has been determined, the particles can be spawned about their respective fluid particle. The fact that diffuse particles interact only with the fluid but not the other way around, meaning they are one-way coupled to the simulated system - can now be used: the computations relating to diffuse particles for some time step are independent of the fluid simulation and can simply be delayed until the fluid solver has already calculated the 'future' positions $\vec{x}_f(t + \Delta t)$ of all fluid particles at the next time step. This allows diffuse particles associated with a fluid particle to be spawned precisely in the cylinder traced by the movement of the particle that spawned them. The cylinder has height corresponding to the distance $|\vec{x}_\Delta| = |\vec{x}_f(t + \Delta t) - \vec{x}_f(t)|$ travelled by the fluid particle and the same radius r_v the fluid particle would have [59] if it was assumed to be spherical, which in terms of particle mass at rest density can be expressed as:

$$r_v = \sqrt[3]{\frac{3}{4}\pi \cdot \frac{m_i}{\rho_0}} \quad (5.19)$$

Using the fluid particles' normalized movement direction $\hat{\vec{x}}_\Delta$, an orthonormal basis $\hat{\vec{e}}_1, \hat{\vec{e}}_2, \hat{\vec{x}}_\Delta$ with $\vec{x}_f(t)$ as its origin can be found, using the Gram-Schmidt procedure for example, and used to sample a random point in the cylinder [59]:

$$r = r_v \sqrt{\xi_r} \quad (5.20)$$

$$\theta = 2\pi\xi_\theta \quad (5.21)$$

$$h_c = |\vec{x}_\Delta| \xi_h \quad (5.22)$$

$$\vec{x}_d = \vec{x}_f(t) + r \cos \theta \hat{\vec{e}}_1 + r \sin \theta \hat{\vec{e}}_2 + h_c \hat{\vec{x}}_\Delta \quad (5.23)$$

with uniformly randomly drawn $\xi_r, \xi_\theta, \xi_h \sim \mathcal{U}([0; 1])$. The velocities of the diffuse particles are initialized to point in the direction of fluid velocity but away from the cylinder's centre axis, such that splashes can be more accurately modelled [59]:

$$\vec{v}_d = r \cos \theta \hat{\vec{e}}_1 + r \sin \theta \hat{\vec{e}}_2 + \underbrace{\frac{1}{\Delta t} \vec{x}_\Delta}_{\vec{v}_f} \quad (5.24)$$

Classification and Advection

Just as three different phenomena leading to diffuse particle generation are captured by the generating potentials, the diffuse particles themselves are also divided into three categories at each time step: spray,

foam and bubbles. Each diffuse particle is classified into one of these categories at each time step and advected according to a corresponding set of rules. The classification is done in a rather ad-hoc manner based on the number of fluid particle neighbours in the kernel support radius $\hbar = 2h$, where less than 6 neighbours indicate spray, more than 20 neighbours indicate bubbles and any other value classifies as foam [60].

- Spray moves ballistically through the air, only affected by gravity. It is updated according to [59]:

$$\vec{v}_{spay}(t + \Delta t) = \vec{v}_{spay}(t) + \Delta t \vec{g} \quad (5.25)$$

$$\vec{x}_{spay}(t + \Delta t) = \vec{x}_{spay}(t) + \Delta t \vec{v}_{spay}(t + \Delta t) \quad (5.26)$$

- The trajectories of foam and bubbles on the other hand are strongly influenced by the fluid's velocity field, the average of which is interpolated to diffuse particle positions using the kernel-weighted average [59]:

$$\tilde{\vec{v}}_i = \frac{\sum_{j \in \mathcal{N}_i} \vec{v}_j W_{ij}}{\sum_{j \in \mathcal{N}_i} W_{ij}} \quad (5.27)$$

where i refers to a diffuse particle and j to the fluid particles in its neighbourhood.

- Since foam floats on the fluid's surface, it is modelled to move with precisely the local average velocity $\tilde{\vec{v}}_i$ and no momentum of its own [59]:

$$\vec{v}_{foam}(t + \Delta t) = \tilde{\vec{v}}_i(t + \Delta t) \quad (5.28)$$

$$\vec{x}_{foam}(t + \Delta t) = \vec{x}_{foam}(t) + \Delta t \vec{v}_{foam}(t + \Delta t) \quad (5.29)$$

- Meanwhile, bubbles are modelled to be affected by both hydrostatic buoyancy, accelerating them upwards in the negative direction of gravity $-\vec{g}$, and by drag forces, pulling them along with the fluid's average velocity $\tilde{\vec{v}}_i$, resulting in [59]:

$$\vec{v}_{bubble}(t + \Delta t) = \vec{v}_{bubble}(t) + \Delta t \left(-k_b \vec{g} + k_d \frac{\tilde{\vec{v}}_i(t + \Delta t) - \vec{v}_{bubble}(t)}{\Delta t} \right) \quad (5.30)$$

$$\vec{x}_{bubble}(t + \Delta t) = \vec{x}_{bubble}(t) + \Delta t \vec{v}_{bubble}(t + \Delta t) \quad (5.31)$$

with coefficients $k_b = 0.8$, $k_d = 0.2$ that can be set with artistic freedom.

Dissolution

All diffuse particles are spawned with some predefined lifetime and are destroyed when this lifetime is exceeded. While a fixed, constant time could be used, it is recommended to set the lifetime to be proportional, within some minimum and maximum bounds, to the generating potentials that gave rise to the diffuse particle [59], such as the kinetic energy potential [60], or additionally vary the lifetime randomly [60] to obtain more realistic results.

In this report, particles that leave the simulation domain or are found within some small radius of a boundary particle are also destroyed, since diffuse particles might otherwise pass through boundaries unrealistically or fall through the bottom of the fluid's container without contributing to the enhancement of the visuals.

Free parameters

The full algorithm as outlined in this section has a large number of free parameters, the tuning of which by hand can be tedious. Instead, it is [60, suggested] to employ heuristics to reduce the amount of hand-tuning. The minimum values τ_{min} used as an argument to the normalization function Φ in Equation 5.11 for example are set to 10% of the respective maximum values τ_{max} [60], eliminating 3 free parameters. The maximum values τ_{max} can be estimated by taking the maximum value of the quantity to be normalized across all fluid particles and time-averaging these values across the simulation, i.e.:

$$E_{kin}^{max}(t) = \max_i E_{kin,i} \quad \text{using Equation 5.15} \quad (5.32)$$

$$\tau_{max}^{kin} = \frac{1}{T} \sum_{t=0}^T E_{kin}^{max}(t) \Delta t \quad (5.33)$$

and analogously for the wave-crest and trapped-air potentials. This method can be applied directly to the algorithm if it is used as an offline post-processing step, where all the data necessary to compute each τ_{max} is known - this does however require storing all necessary particle positions and velocities of the simulation frames under consideration and rebuilding neighbourhood search data structures to compute SPH sums when generating foams, spray and particles. In this report, the method is used online instead, such that only two sets of particle data must be buffered at most and data structures can be reused, resulting overall faster computation. The estimation of τ_{max} -parameters is performed on a separate, low-resolution run of the simulation beforehand, in which Equation 5.32 and similar are used to estimate parameters that are later used in the full-scale simulation. This is unproblematic, since both the Kernels W, K used in SPH computations and the kinetic energy in Equation 5.15 are normalized to be as insensitive as possible to the particle resolution h .

ANALYSIS

In chapter 3, an incompressible SPH fluid solver was derived and solver variations were presented in subsection 3.4.1, resulting in the Implicit Incompressible SPH or IISPH solver (Algorithm 1), which may vary by whether pressure values are reset in every step (Equation 3.64) or a warm start based on pressures of the previous time step is performed (Equation 3.63), as well as a Divergence-Free or DFSPH solver (Algorithm 2) using an additional source term formulation.

The time step size Δt was investigated as a performance-critical parameter in subsection 3.4.2 and a controller designed to converge to a performance-optimal time step size for static scenes was developed, which eliminates arbitrary hand-tuning of the Δt parameter and therefore yields a more robust performance metric that algorithms and variations can be compared by. This metric is employed in this chapter to investigate questions about the behaviour of the presented incompressible SPH solvers and compare them. Some questions of interest that were alluded to in the report include:

- How does the fluid solver's performance scale in the height of the tallest water column in the scene, which is commonly used to categorize a scene's complexity?
- How does the optimal time step size, as determined by the Metropolis controller in subsection 3.4.2, relate to the simulated scene's complexity?
- How does warm-starting a given solver affect its performance? How does it affect stability, as measured by undesired oscillations of the water columns?
- How do IISPH and DFSPH compare in performance and stability?
- How do IISPH and DFSPH solvers compare in performance and stability to simpler, Equation-of-State based iterative pressure solvers?

It is the aim of this chapter to present data that might help answer all of these questions and more.

6.1 Methodology

In order to answer these questions, the time step size controller from subsection 3.4.2 is employed. To summarize, a Metropolis Random Walk with a heuristic that balances exploration and exploitation is used to search the space of possible time step sizes Δt to maximize a performance metric. In SPH literature, the performance of solvers is often compared by running simulations at a given Δt that is decided ad-hoc and comparing the resulting total runtimes [18, 4, 61], or the fact that a larger time step size is tolerable by a more stable algorithm is taken as indication of its superior performance. This might be a suboptimal methodology, since smaller than necessary time step sizes can be shown to be superior in terms of performance for concrete implementations in which the pressure solver iteration itself is the computational bottleneck and the computational cost of per-step preliminary computations such as neighbourhood searches are relatively low. In such settings, a lower iteration count as caused by a smaller time step size can be more beneficial. The time step controller eliminates Δt as a rather dubious free parameter by optimizing it to maximize performance. The time-averaged performance \bar{P} described in subsection 3.4.2 that the controller converges towards can then be used as a measurement of the solver's performance on a given scene. This metric approximates the ratio of simulated time T_{sim} to the real time, or wall-clock time it took to run the simulation, meaning that $\bar{P} \geq 1$ represents a simulation running at least in real-time, while much lower values are expected of more demanding scenes.

Since no guarantees about the speed of convergence of the time step controller towards an optimum can be made at this time, it is more robust to rely in the following on a static test scene in which a water column rests under hydrostatic pressure, instead of some dynamic scene in which the quality of time step controller and the heuristics and hyperparameters it uses are relevant - for a static scene it only

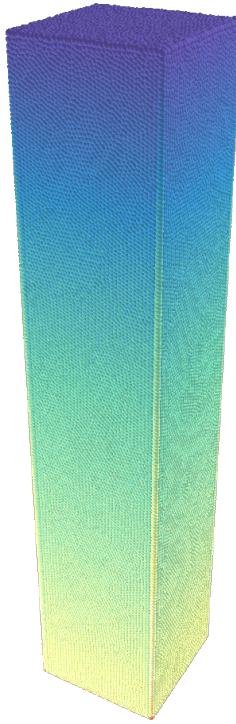


Figure 6.1: An exemplary rendering of the static water column used as a scene for performance analysis, showing particles as opaque spheres of radius $1/2 \cdot h$ in the particle spacing $h = 5\text{cm}$. Pressure is colour coded from low pressures in blue through yellow and orange towards red. This particular square column measures $\sim 303K$ fluid particles and a size of $2\text{m} \times 2\text{m} \times 10\text{m}$, where the height of 10m is referred to as the scene's complexity. The same $2\text{m} \times 2\text{m} \times 20\text{m}$ container consisting of $\sim 218K$ boundary particles is used in every measurement. The rendering is taken from a simulation using the standard, cold-started IISPH solver in Algorithm 1 at $t = 1\text{s}$.

matters that it *does* converge on some performance score reliably and irrespective of its hyperparameters, as demonstrated in Figure 3.2, not *how fast* it does.

As a consequence, the scene shown in Figure 6.1 is employed in the following, consisting of a simple water column of variable height in a square prism container.

The analysis in this report does not claim to make absolute, quantitative observations about the performance of these solvers in general, but rather to provide a framework in which statements about the *relative* performance of solver variations on a given machine, using a given implementation, can more robustly be made. So while analysing the impact of warm-starting an otherwise identical algorithm or comparing the same algorithm in varying scenes can result in more reliable conclusions, the comparison of two more starkly different solvers could be affected by inefficiencies in the respective implementation rather than inherent properties of the algorithm. Note once again that all the presented solvers, including the solver introduced in the following subsubsection 6.1 for comparison, have been shown recently to be [29, equivalent] in a strong sense, meaning they differ at most by factors such as the ease of reusing values for more efficient computation or by their convergence behaviour. In fact, later results in this chapter (Figure 6.2, Figure 6.3) are in line with this expectation, showing similar behaviour up to a multiplicative factor.

An Iterative Equation-of-State Solver for Comparison

One of the questions of interest is how the more elaborate solvers presented in this report compare against simpler fluid solvers. The most basic and therefore obvious comparison would be between the discussed IISPH and DFSPH solvers and a basic, weakly compressible SPH solver, or WCSPH for short, which computes pressures directly as a function of the density error using an equation of state, without explicitly, globally solving the pressure Poisson equation [17]. This solver, although easy to implement potentially efficient for low-complexity scenes, is hard to compare against the presented solvers using the outlined methodology: since the solver is not iterative, it cannot be forced to stay within the error threshold specified by the convergence criterion in Equation 3.67. While the time step size can be controlled such that

the criterion is upheld most of the time or in expectation, it can momentarily be broken, leading to an unfair advantage in a comparison of the performance metric, which is extremely sensitive to the error threshold that is enforced.

Instead, the next simplest, iterative solver that realizes an *incompressible* fluid solver that can strictly enforce the convergence criterion by employing operator splitting and iteratively applying a specialized equation of state was implemented here as a baseline for comparison: **PREDICTIVE-CORRECTIVE SPH** or PCISPH for short. The solver relates errors in the predicted density at a particle directly to pressures by simply using the ideal gas law as an equation of state:

$$p_i = k \underbrace{(\rho_0 - \rho_i^*)}_{s_i} \quad (6.1)$$

Note the density invariance source term $s_i = \rho_0 - \rho_i^*$ from Equation 3.35. However, it uses a special, geometrically motivated stiffness coefficient k of [61]:

$$\beta = -\frac{\rho_0^2}{2\Delta t^2 m^2} \quad (6.2)$$

$$k = \beta \cdot \underbrace{\left(\left| \sum_{\vec{x}_j \in \mathbb{Z}^3} \nabla W(h \cdot \vec{x}_j, \hbar) \right|^2 + \sum_{\vec{x}_j \in \mathbb{Z}^3} |\nabla W(h \cdot \vec{x}_j, \hbar)|^2 \right)^{-1}}_{=:k_{tplt}} \quad (6.3)$$

where k_{tplt} is computed for an idealized template particle at the origin, surrounded by perfectly sampled neighbours on a regular cubic lattice $h \cdot \mathbb{Z}^3$ with particle spacing h [61].

This means that k_{tplt} only depends on the geometry of the problem, such as the choice of kernel function W , particle spacing h , smoothing length and support radius \hbar , and must only ever be evaluated once per such choice. Note the similarity of k in Equation 6.3 to the diagonal element of IISPH (Equation 3.60), which is in a sense [29, equivalent] to PCISPH. However, in contrast to IISPH and DFSPH where values equivalent to stiffness coefficients are computed per-particle and time step, k_{tplt} is a global constant and the only value that must be regularly recomputed in Equation 6.3 is β . Assuming equal masses of particles, even this has to only be done once per time step, not once per-particle.

The resulting, computationally inexpensive pressure computation is factored out using the concept of operator splitting (Equation 3.11) and iterated upon to accumulate a pressure $p_i = \sum_l \Delta p_i^l$ that results in an uncompressed state, fulfilling some convergence criterion (e.g. Equation 3.67). This results in the solver outlined in Algorithm 6 [61].

Algorithm 6 PCISPH Fluid Solver Step

Step 1 – Preliminary Computations

- 1: Find neighbour sets $\mathcal{N}_i, \mathcal{B}_i$ ▷ Fixed Radius Neighbour Search (Appendix A)
 - 2: Compute non-pressure accelerations \vec{a}_i^ν, \vec{g} ▷ Equation 3.2
 - 3: Determine time step size Δt ▷ Equation 3.7 or subsection 3.4.2
-

Step 2 – PCISPH Solver Iteration

- 4: **for** iteration $l = 0$ until convergence **do** ▷ convergence criteriton Equation 3.67
 - 5: Compute predicted velocities \vec{v}_i^* from accelerations ▷ Operator splitting, Equation 3.11
 - 6: Compute density invariance source term $\rho_0 - \rho_i^*$ ▷ Equation 3.35
 - 7: Update β in stiffness coefficient k ▷ Equation 6.3
 - 8: Update pressures $p_i \leftarrow (k \cdot s_i)_+$ ▷ Equation 6.1
 - 9: Update pressure accelerations $\vec{a}_i^{p,l}$ ▷ Equation 4.8
 - 10: **end for**
-

Step 3 – Numerical Time Integration

- 11: $\vec{v}(t + \Delta t) \leftarrow \vec{v}_i^* + \Delta t \vec{a}_i^p$ ▷ like Equation 3.5, respecting operator splitting
 - 12: $\vec{x}(t + \Delta t) \leftarrow \vec{x}_i + \Delta t \vec{v}(t + \Delta t)$
-

Parameter Choices and Specifics

In an effort to give a complete and reproducible description of the performed analyses, some parameter choices are now discussed.

With regard to the time step controller, since the test scene is static, the time horizon discussed in subsection 3.4.2 is disabled by setting $t_{hrz} = \infty$, since the problem difficulty does not change. Episodes of $K = 5$ steps are averaged to obtain a more accurate performance estimate and conservatively small values for the exploration rate $\chi = 0.01$, the proposal distribution variance $K \cdot \sigma^2 = 5 \cdot 10^{-6} \text{ sec}^2$ and the initial time step size $\Delta t_0 = 10^{-5} \text{ sec}$ are used to minimize sudden jumps in time step size that might negatively affect solver stability or average performance.

A low minimum iteration count of $l_{min} = 3$ is used, which is in line with the observation in subsection 3.4.2 that optimal time steps for this specific implementation seem to hover towards the lowest end of possible iteration counts since neighbourhood search is relatively fast. For this reason, the DFSPH solver in Algorithm 2 must have the same total minimum iteration count, resulting in a fixed number of $l = 2$ iterations of the divergence-free solver and an additional minimum of $l_{min} = 1$ iterations of the density-invariant solver. Maximum iteration counts are disabled using $l_{max} = \infty$. Small error thresholds of $\rho_{avg}^{thresh} = 0.1\%$ [18] and $\rho_{max}^{thresh} = 0.5\%$ are used for the convergence criterion in Equation 3.67 as recommended in the literature and provide a realistic setting in which a high quality of the simulation is enforced.

The relaxed Jacobi solver uses the recommended [18] relaxation factor of $\omega = 0.5$ and a minimum value of $\epsilon = 10^{-6}$ for diagonal elements \mathbb{A}_{ii} to prevent numerical issues due to divisions by minuscule values or zero in Equation 3.62.

In order to obtain a performance metric that is as accurate as possible, a large span of simulated time, $T_{sim} = 60 \text{ sec}$ in this case, is simulated and the time-averaged performance score $\bar{P}(T_{sim}) = \frac{1}{T_{sim}} \hat{P}(t_k) \Delta t_k$ using individual scores $\hat{P}(t_k)$ as outlined in Equation 3.76 is taken across this entire time span. This prevents anomalies at the start of the simulation or the speed of convergence of the time step controller in subsection 3.4.2 from having a significant impact on the overall result of the performance measurement.

It was observed in the experiments that cold-starting the DFSPH solver results in unnecessarily bad performance, since the low minimum iteration counts and resultingly small time step sizes in this setting do not justify resetting pressure values between time steps, let alone in between the two stages of the solver, which is why cold and warm starts are only compared for the IISPH solver.

6.2 Performance Scaling in Scene Complexity

It was previously claimed in this report that a scene's complexity is under most circumstances predominantly defined by the height of the highest column of fluid under hydrostatic pressure. Having set up a methodology for comparing the relative performances of solver variations against each other and under varying scenes, this claim can now be substantiated and the relationship of how the maximum height influences the computational cost can be more closely investigated.

The test scene is simulated for varying fluid column heights by repeatedly doubling them from 0.5m and $\sim 15K$ fluid particles all the way to a column of 16m height with $\sim 485K$ particles for $T_{sim} = 60 \text{ sec}$. The resulting average performance score is plotted for each solver variation in Figure 6.2.

The graph reveals that the performance decreases at what appears to be at least an exponential rate for the scenes under consideration, substantiating that the fluid column height is in fact the dominating factor in the computational cost of simulating each scene in this static setting. The performance curves seem follow more or less the same trend, which is unsurprising considering that they can [29, all be shown to be functionally equivalent to some degree], albeit with a difference in relative performance that appears to manifest in a vertical offset of the curves in the logarithmic plot. This points to the conclusion that while these incompressible SPH fluid solvers behave similar in terms of the scaling of their performance in scene complexity, they are separated by some multiplicative factor for the scenes measured. This factor is more closely investigated in the analysis surrounding Figure 6.4.

It can also be observed that the PCISPH solver that uses a single, global stiffness coefficient is, in this implementation, outperformed across all scenes by the solvers that are more explicitly designed to solve the full system posed by the pressure Poisson equation efficiently. This becomes substantial at greater scene complexities where performance is generally low and a relative increase in performance results in greater absolute savings of compute time. Warm-starting the IISPH solver can be observed in this case to lead to the consistently highest performance measurements.

Since using a time step controller to search the space of possible values of Δt and optimizing the

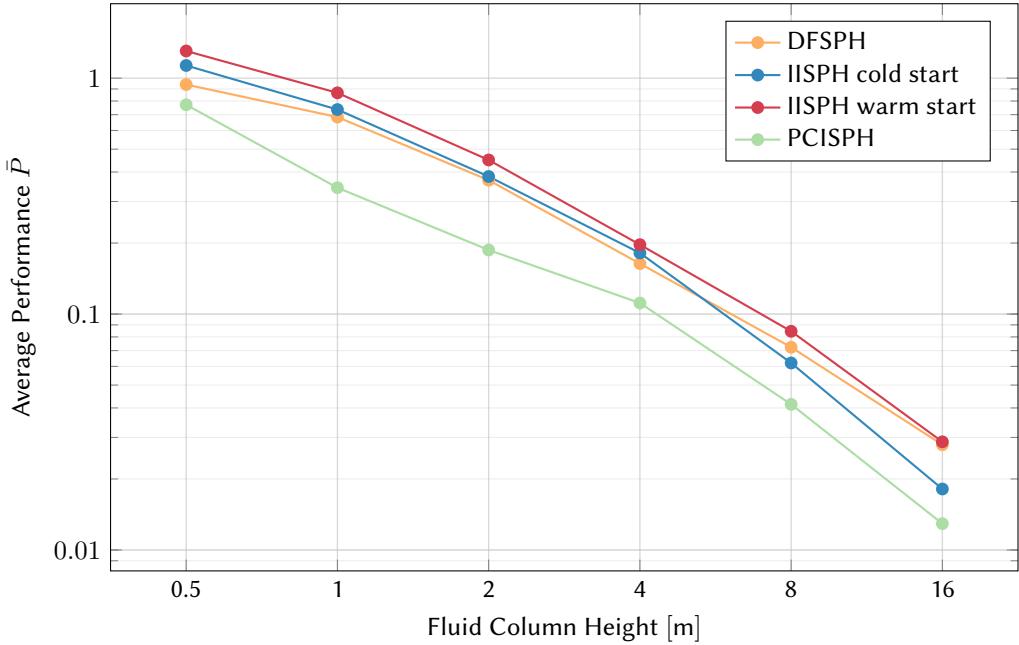


Figure 6.2: Each solver is tested on a static fluid column as observed in Figure 6.1 of varying height. It is observed that performance drastically decreases as the height of the fluid column increases, substantiating that the fluid column height is a valid measure of scene complexity in this setting. Note the logarithmic scaling on both axes of the plot. It can be seen that all curves follow more or less the same trajectory but are spaced out vertically, suggesting that while the general scaling of performance with scene complexity might be similar amongst solvers, the performance of each solver differs by a factor, which is in line with expectations. For all test scenes, it can be observed that IISPH and DFSPH consistently outperform PCISPH and that warm-starting the IISPH solver results in higher performance than resetting pressure values does. Figure 6.4 provides a closer look at the relative performances of solvers by dividing each curve by the result for the cold-started IISPH solver.

choice to maximize a performance metric is rather uncommon, it might be interesting to take a closer look at what this optimization algorithm deems an optimal time step size for the varying scenes and solvers measured in Figure 6.2. Since the simulated timespan is orders of magnitude longer than the time it takes for the controller to converge to a time step size and little variation was observed once such an optimum has been found, the mean time step size $\bar{\Delta}t$ seems like a reasonable metric to estimate this optimal time step size. This mean time step size is plotted against the fluid column height for all solver variations in Figure 6.3.

The graph shows that just as the performance decreases exponentially as the scene's complexity increases, so does the optimal time step size. A couple of conclusions can be drawn from this.

Firstly, since the optimal time step size seems to scale similarly to the performance metric, the optimal average iteration count per solver step appears to be near constant across the tested scenes, since that is precisely the factor relating the graphs in Figure 6.2 and Figure 6.3.

Secondly, while it is intuitive for a more demanding scene to *require* a smaller time step size to remain stable, it should be noted that this behaviour was found by maximizing *performance* while staying within the specified error bounds (Equation 3.67), *not* maximizing the stability or quality of the simulation per se. This seems to hint at the conclusion that increased simulation quality and stability and increased performance are not so much a trade-off in this scenario, with one excluding the other, as they are correlated and simultaneously increased by a well-chosen fluid solver and time step size.

Thirdly, it appears as though all relaxed-Jacobi-based solvers converge towards roughly the same optimal time step size as the scene complexity increases, while PCISPH eventually requires smaller time steps. While this observation could be inaccurate due to the low number of data points and limited complexity of the most complex scene in this data series, it does point towards an underlying mechanism relating optimal time step size and scene complexity for the entire respective class of fluid solvers. It could be speculated that PCISPH less efficiently propagates pressure information for complex scenes, leading to a smaller numerical speed of sound and therefore a smaller required time step size to propagate information throughout the entire fluid column, while using solvers that handle stiffness on a per-particle basis seem to eventually more efficiently establish a consistent pressure field, which is in line with expectations. An interesting avenue for future analysis could be considering the impact of conjugate gradient solvers on the optimal time step size, which could be expected to be even higher if that hypothesis were

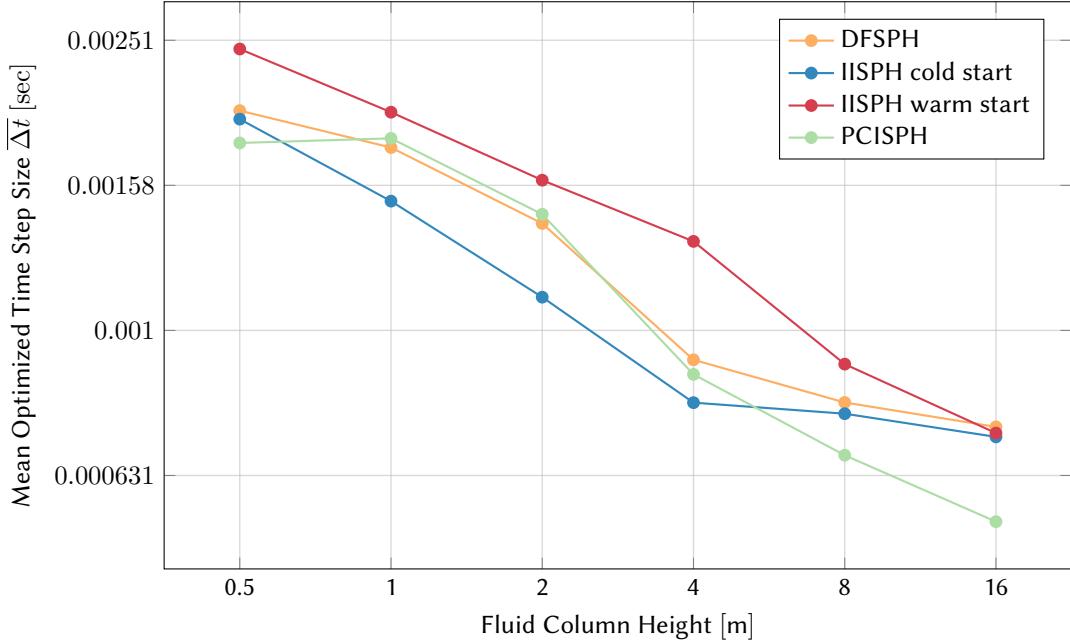


Figure 6.3: The average of the time step size Δt across the simulation, as optimized by the time step controller in subsection 3.4.2, is plotted against the fluid column height. It can be observed that, perhaps unsurprisingly, the optimal time step size decreases as the complexity of the scene increases. While the IISPH and DFSPH solvers run optimally at more similar time step sizes as complexity increases, PCISPH requires increasingly small time step sizes in comparison. Warm-starting the IISPH solver appears to consistently increase the optimal time step size.

true.

6.3 Performance Comparison of Solvers

The obvious question arising from the different solver variations presented in this report and the presented methodology for performance analysis is how much relative performance each solver achieves in direct comparison. While performance is certainly not the only criterion for the quality of a fluid solver, and perhaps not even the most important one when compared to ease of use, stability, robustness, the versatility of boundary handling schemes, quality and temporal consistency of the velocity and pressure fields, ability to handle viscous effects, surface tension effects, interplay with other solvers with regard to multi-physics simulation and many more, it is still an important and oft-discussed factor when considering which pressure solver to use. Note that comparisons between differing solvers are less conclusive, since performance differences due to implementation details rather than algorithmic advantages cannot be ruled out.

To make the performance comparison easier, the performance measurement from Figure 6.2 is normalized by dividing each curve by the results for the cold-started IISPH solver, which serves as a baseline \bar{P}_0 for relative comparisons, defining the value 1. This results in a graph showing the relative performance of each solver as a factor of the cold-started IISPH performance for varying scene complexity in Figure 6.4.

The graph confirms a couple of assumptions. Firstly, it appears that warm-starting the IISPH solver unsurprisingly results in increased performance, with up to a $\sim 60\%$ increase found for the most complex scenes. The warm-started DFSPH solver, which initially performs worse than the baseline implementation with seemingly more overhead due to the additional source term than it is worth, achieves increasingly similar performances to the best-performing warm-started IISPH solver. Finally, a third group of performance scores is traced by the PCISPH solver, which performs consistently worse than its counterparts across all scenes, hovering between 50% and 70% of the baseline IISPH implementation's performance, which could indicate the possible performance advantages to be gained in complex scenes by solving the pressure Poisson equation using per-particle coefficients.

This method of comparing solver performances in a robust way for benchmark scenes might serve as a framework for evaluating not just the established methods presented in this report, but could potentially serve as useful feedback during the development of new solvers or while optimizing implementations,

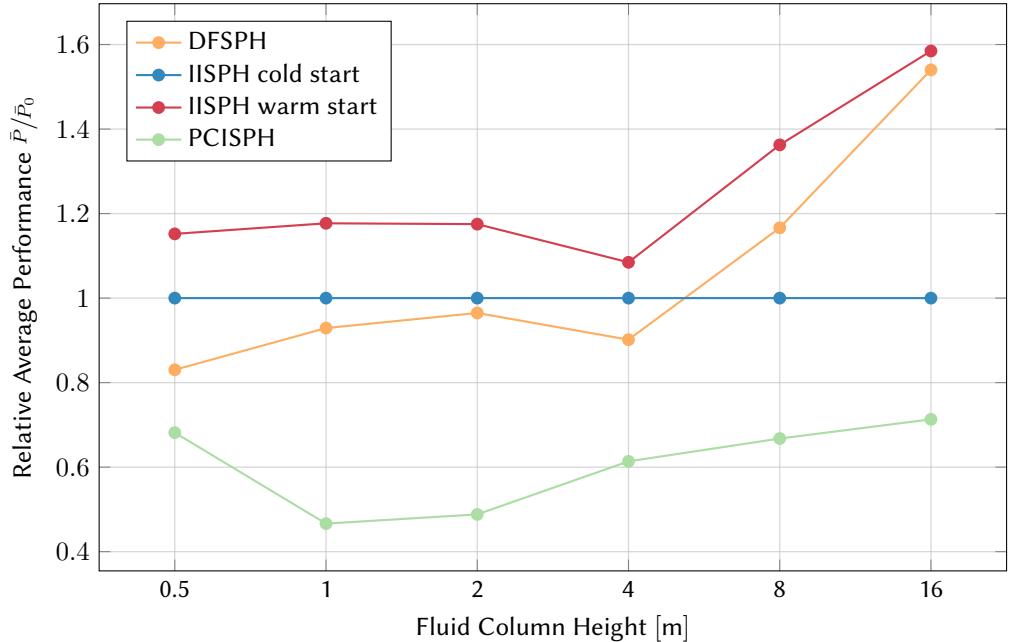


Figure 6.4: The average performance score \bar{P} of each solver variation as a function of the simulated fluid column’s height, as shown in Figure 6.2, is rescaled by dividing each data point by the result for the cold-started IISPH solver \bar{P}_0 , the behaviour of which defines the value 1 in this graph. As a result, the relative performances as percentages of this baseline IISPH implementation can be observed for varying scene complexity. It again becomes clear that for the tested scenes, the equation-of-state-based PCISPH consistently performs worse than its counterparts that explicitly solve the pressure Poisson equation on a per-particle basis. While the divergence-free source term of DFSPH seems to result in more overhead than it is worth for low complexities, it results in a performance increase for more demanding scenes, as is intended, eventually catching up to the warm-started IISPH solver. IISPH solver performance again consistently improves across all considered scenes when a warm start is employed, resulting in the highest performances measured each time.

making the process more rigorous. Instead of simply assuming an optimization would result in a performance gain, changes to an implementation can be compared against a previous iteration thereof to gain confidence in whether a change did indeed have a positive impact, without worrying about the effects of ad-hoc parameter choices regarding the time step size Δt .

6.4 Stability Comparison of Solvers

As previously mentioned, performance is by far not the only, or even the most relevant, aspect to consider when choosing a fluid solver. Another especially important criterion is the stability and quality of the simulation produced by the solver, which in application to Computer Graphics might be judged by whether any undesired visual artefacts occur. A usual artefact inherent to solvers that minimize absolute density errors and compressions (and, contrary to some beliefs, irrespective of whether a Lagrangian scheme such as SPH or an Eulerian scheme is chosen), is the manifestation of accumulated solver errors as oscillations, as opposed to volume shrinkage or drift caused by differential source terms. One of the declared goals of DFSPH [4] and the idea of combining source terms is indeed minimizing such oscillations and producing a more temporally consistent velocity and pressure field, amongst other things. While this supposed boon of DFSPH might not be as prominent in this analysis, where a very low minimum iteration count for each solver was chosen in order to better compare performances, giving the divergence-free solver less opportunity to reduce such artefacts, it might still be interesting to compare oscillations in the data obtained in the previous sections.

Again, the goal of solver stability requirements in practical application to Computer Graphics could be seen to lie in the reduction of visual artefacts and not directly, as might be the case in more engineering-focused applications, to reduce errors in field quantities, as measured for example in scenes for which analytic solutions to the Navier-Stokes equations exist. For that reason, this report opts not to measure the kinetic energy present in the ideally stationary, hydrostatic water column, the velocity divergence or some other indirect error metric, but to observe undesired oscillations in the column’s height, since this is a property of the trajectories of the particles that persists beyond surface reconstruction and is certainly visible, whereas velocity errors and kinetic energies might not, or only indirectly, be.

Stability is therefore measured by considering the maximum of the vertical projection of all particle positions, or the maximum height of any particle $\epsilon = |\max_i \pi_y(\vec{x}_i) - H|$ where π_y is the projection on the y -axis. This value is compared against the intended rest height of the fluid column H shown on the horizontal axis of all graphs in this chapter to yield an error ϵ . This absolute, L1 error in maximum height is then time-averaged across the entire $T_{sim} = 60$ sec of simulated time to obtain an average $\bar{\epsilon}$. In order to enable a more insightful analysis of this value, it is divided by the error $\bar{\epsilon}_0$ obtained by the baseline, cold-started IISPH implementation, yielding an error metric that shows the relative amount of undesired oscillations and instability when compared to the basic IISPH solver.

Apart from obvious, large-scale oscillations of the entire fluid column that are undesired, lacking stability or incompressibility can also manifest in particles at the surface bouncing violently, as if evaporating - this effect can be observed for example when a time step size too large is used in a weakly compressible SPH solver. Again, this effect is captured by the height error metric.

The height error is plotted for varying fluid column heights in Figure 6.5.

Firstly, it can be observed that the solver variations do not differ immensely in stability, since even the worst solver in the most complex scene only scores an error less than 0.7% greater than that of the best solver. This can be taken to mean that all tested solvers are in fact valid and useful incompressible SPH solvers that are capable of producing visually plausible results for scenes at least up to the measured maximum difficulty.

Secondly, it can be observed that the clearest trend amongst the curves is traced by the PCISPH solver, which once more consistently performs worse than its counterparts and the error of which compared to standard IISPH grows more and more as the scene's complexity increases.

Thirdly, it can be seen that the warm-started IISPH and DFSPH solvers might achieve better height errors than IISPH for lower complexity scenes (although the empirical significance of this result is questionable), while eventually being outperformed by cold-started IISPH. It was argued when introducing Equation 3.63 that cold-starting the solver can improve stability for especially difficult scenes at the cost of some performance, which is substantiated here. Note that all measurements are normalized by the error of the cold-started IISPH implementation, so the apparent rise in errors of warm-started solvers for complex scenes can just as well be interpreted as a relative *decrease* in errors by the cold-started IISPH. Contrary to the interpretation regarding the time step size in the previous section, there appears to be a *negative* correlation or trade-off between stability and performance regarding whether to cold- or warm-start an IISPH solver, with cold starts resulting in a more robust solver for complex scenes at the cost of some performance.

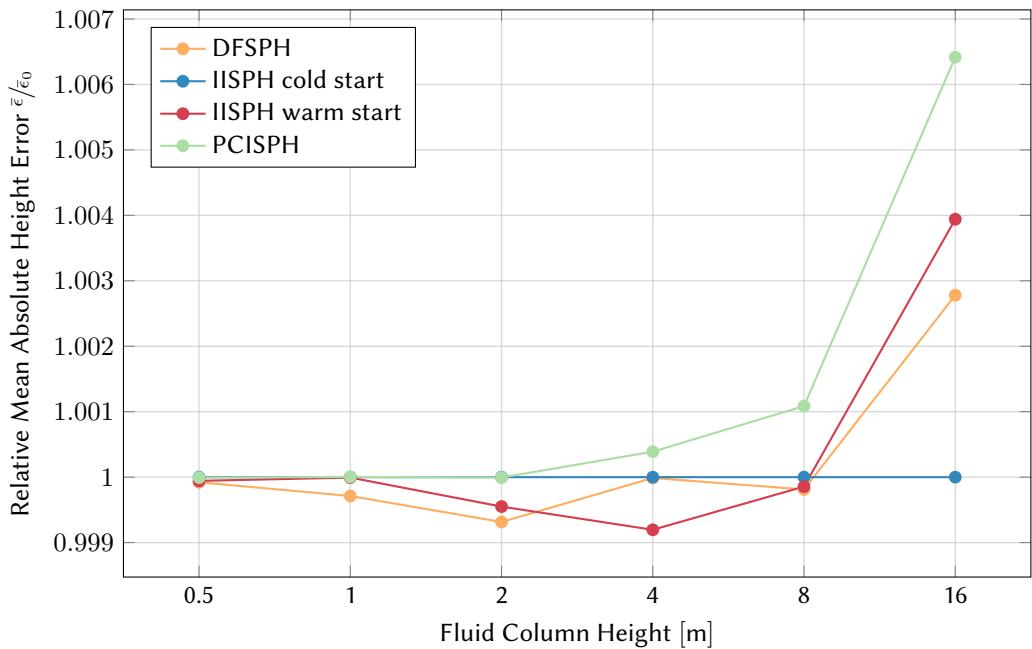


Figure 6.5: The stability of a simulation is measured by tracking the maximum y -position of any fluid particle in the scene and comparing it to the expected rest height H of the simulated fluid column at each simulated time step. The absolute error $\epsilon = |\max_i \pi_y(\vec{x}_i) - H|$ (where π_y is the projection on the vertical axis) is measured at each time step and averaged across the entire simulation. It is then normalized by the error incurred by the cold-started IISPH solver, meaning the vertical axis in the graph shows the absolute height error *relative* to that of the baseline, cold-started IISPH implementation from Algorithm 1. It is revealed that PCISPH in particular suffers increasingly from unwanted oscillations in the fluid column, resulting in increasing height errata and an unstable surface as the scene becomes more complex. While the warm-started IISPH and DFSPH solvers manage to keep the column even closer to its intended height than the cold-started IISPH for some scenes, the latter appears to eventually win out in terms stability as the scene complexity increases, which is within expectations.

CONCLUSION

In this report, an implementation of a fully GPU-parallel, performant and robust state-of-the-art SPH-based incompressible fluid solver with a two-way coupled rigid body solver was motivated, outlined and analysed. The solver is capable of handling complex, intersecting and procedurally moving boundary geometry, enforcing incompressibility up to an arbitrarily small threshold and processing the results for visualization, including the addition of physically motivated spray, foam and bubble generation.

The Navier-Stokes equations that describe fluid flow in general and incompressible flow in particular were presented along with the Smoothed Particle Hydrodynamics scheme employed to discretize the equations. The pressure Poisson equation was derived, discretized and a perhaps uncommonly detailed derivation of the relaxed Jacobi solver and the diagonal element used to solve the equations using Implicit Incompressible SPH pressure solvers was shown. Similarities and differences to other methods and solver variations such as Divergence-Free SPH, which uses a differing source term, were presented and discussed.

An uncommon time step size controller using the concept of Metropolis Random Walks and Simulated Annealing and a corresponding heuristic for control of the exploration-exploitation trade-off in dynamic scenes was developed and put to practical use. This enabled more efficient simulation and more robust performance analyses in chapter 6 which are free of ad-hoc parameter choices of the time step size.

A particle based, robust and flexible boundary handling scheme was presented that can handle close to arbitrary boundary geometry, including procedurally or dynamically moving and intersecting boundaries. The scheme monolithically integrates into the incompressible fluid pressure solver, using it to resolve contacts. An iterative mass initialization per-particle was proposed and a process for sampling low-discrepancy particle distributions on input surfaces to obtain a boundary sampling was described.

A rigid body solver that uses quaternions to more accurately represent orientations and rotation with less numerical drift was shown and the calculation of mass moments required in the solver was outlined in detail using a less common method that remarkably is exact for watertight triangular input meshes.

The visualization of the simulation results was discussed, including a stat-of-the-art surface reconstruction and smoothing approach and a physically motivated procedure for generating diffuse particles representing spray, foam and bubbles, enhanced by a proposed stochastic approach for spawning precisely the desired number of particles in expectation. This concluded the outline of a complete fluid solver and two-way coupled rigid body solver from the input scene specification to final, synthesized photorealistic images and videos of virtual scenes.

The time step size controller was used to present an uncommon or novel methodology for performance analysis that was applied to investigate the relationships between performance of solver variations and scene complexity, whether the IISPH pressure solver is warm-started or not, and between different, equivalently incompressible pressure solvers that vary in their implementation details.

The following appendices give further insight into details of the neighbourhood search algorithm used in the implementation (Appendix A) and show expressions for the calculation of mass moments of tetrahedra that were corrected and re-written in a more convenient form from sources in the literature (Appendix B).

Future work based on this report might include investigating the relative performances and optimal time step sizes of solvers with differing convergence behaviour, such as Conjugate Gradient approaches [28, 18], further investigating the application of metaheuristics and learning algorithms to parameter choice in SPH fluid simulation, or extending the presented solver with new features such as accurate rigid-rigid interactions [28], surface tension [62], highly viscous [63] or elastic materials [64], accurate vorticity [60] and more.

FIXED RADIUS NEIGHBOUR SEARCH

There exist numerous methods for implementing the computation of the neighbour sets $\mathcal{N}_i = \{j : |\vec{x}_{ij}| < \hbar\}$ referred to in section 2.2. In this implementation, a GPU-friendly index sort based on counting sort was implemented as an implicit representation of a uniform grid to speed up neighbour computation, following the description of [65, Hoetzlein].

It is apparent that a naïve neighbour search incurs a cost on the order of $\mathcal{O}(N^2)$ in the number N of particles, where each of the (ordered) pairs i, j is considered and included in the set based on the predicate $|\vec{x}_{ij}| < \hbar$. Instead, space may be partitioned into a uniform, axis-aligned grid of cell size \hbar , such that for any particle i in d dimensions, only the particles within the 3^d grid cells immediately adjacent to the grid cell in which i resides have to be considered, reducing the computational complexity to $\mathcal{O}(MN)$ for compactly supported kernels $\hbar < \infty$ with at most M particles per grid cell [1].

It should be noted that an efficient and convenient, explicit uniform grid can be constructed in the Taichi language directly using pointer, bitmasked, hashed and dynamic nodes to create a tree structure that is compiled to behave like a simple 3-dimensional field of grid cells, which contain a list of particle indices, as outlined in [2, this paper]. While such an implementation yielded comparable results to the method described in the following and could have been further tuned, it was superseded in this instance by a method more common in the literature about GPU implementations of SPH solvers, and which does not rely on any ad-hoc parameters specific to the implementation, hardware or size of the simulation domain.

Construction

Firstly, the grid cell a particle i resides in is computed as [51]:

$$c(x_i) = \left\lfloor \frac{x - x_{min}}{\hbar} \right\rfloor \quad (\text{A.1})$$

which can be point-wise lifted to a vectorial function $(k, l, m)^T = \vec{c}(\vec{x}_i)$ using the point \vec{x}_{min} that defines the lowest extent of the axis-aligned bounding box of the simulation domain across each axis, with $(x_{max}, y_{max}, z_{max})^T = \vec{x}_{max}$ defined analogously.

The domain is discretized into a uniform grid that can be represented by a linear, one dimensional array using a space-filling curve. While Morton codes are a popular choice [51] for ensuring that spatial proximity is mirrored by proximity in memory, improving cache coherency and reducing scattered reads [65], the XYZ curve or natural order was chosen instead since it guarantees that any neighbour search results in exactly 3^{d-1} coherent sections of memory to be read, simplifying an efficient implementation without use of a BigMin-LitMax algorithm [66] or frequent binary or ternary searches that a Z-order curve would necessitate.

The indices k, l, m along each axis can be flattened into a one-dimensional index using the XYZ curve:

$$I((k, l, m)^T) = k + l \cdot K + m \cdot LM \quad (\text{A.2})$$

where $K = c(x_{max}) + 1$, $L = c(y_{max}) + 1$, $M = c(z_{max}) + 1$ are the number of grid cells along the x, y and z-axis respectively. I now acts as an index into a one-dimensional array of size $N_{grid} = K \cdot L \cdot M$. The remainder of the construction is performed as follows:

1. Let indices be an array of size N representing the one-dimensional cell index of each particle and counts be an array of size N_{grid} representing the number of particles in each cell.

With one parallel loop over particles i , the cell index $I(\vec{c}(\vec{x}_i))$ of each particle can be computed and saved to the i -th entry of indices, while the $I(\vec{c}(\vec{x}_i))$ -th entry of counts is simultaneously incremented using an atomic add operation.

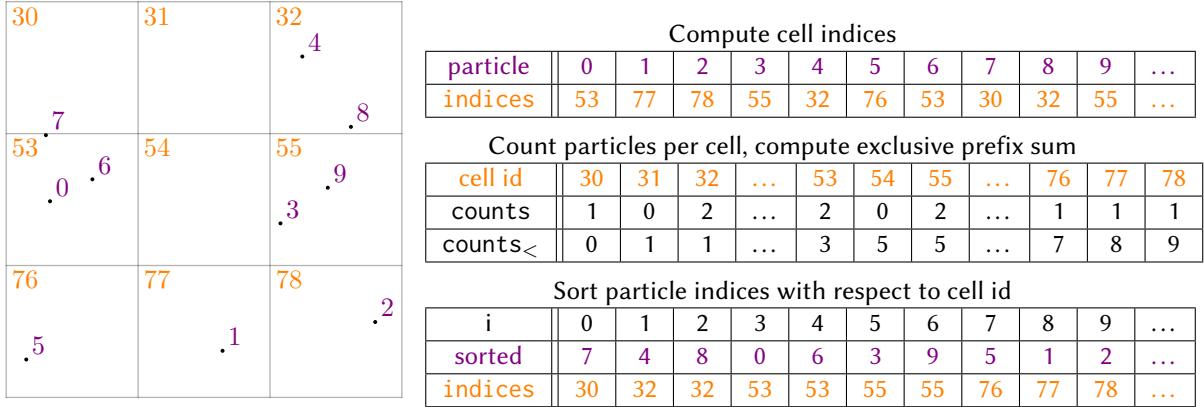


Figure A.1: The process of sorting particles with respect to the integer index of their uniform grid cell along an XYZ-space-filling curve is illustrated for an example set of particle positions in two dimensions. Cell identifiers are labelled in orange, particle identifiers in violet. A counting sort is used, resulting in a buffer of particle counts per cell and a prefix sum thereof $\text{counts}_{<}$ that can be used to look up the neighbours of each particle: the index into the sorted particle list of the first particle in each cell is known, as well as the number of subsequent particles in the list belonging to the same cell. As an example, $\text{counts}_{<}$ of cell 55 is the index of particle 3 in the sorted list and counts tells us that there are two particles in cell 55, which means that particle 9, which is immediately subsequent in the sorted list, is also contained in cell 55. This illustration is similar to one used by [65, Hoetzlin] in the presentation the presented algorithm is based on.

2. A parallel exclusive prefix sum or prescan of the particle counts per cell can then be performed, yielding an array $\text{counts}_{<}$ of size N_{grid} of cumulative particle counts of all cells with strictly lower indices than the cell under consideration. In Taichi, a work-efficient and optimized parallel inclusive prefix sum that avoids bank conflicts [67] and makes use of [68, Blelloch scans] is already implemented. An inclusive prefix sum or scan can be converted to a prescan by point-wise subtracting counts .
3. Finally, a counting sort can be performed, yielding an array sorted of size N representing the particle indices stored in the XYZ-order of the cells they appear in.

This is done by looping over all particles i in parallel again, this time storing the particle index i in sorted at the position calculated by looking up the cumulative particle count of cells with lesser indices than the cell of i in $\text{counts}_{<}$ and adding the result of atomically decrementing the particle count in the cell of i from a copy of counts

This algorithm is outlined in Algorithm 7 and illustrated in Figure A.1.

Algorithm 7 Counting Sort-based Uniform Grid Construction

Step 1 – Compute Indices and Counts

```

1: for  $i \in \mathbb{N}_0^{N-1}$  in parallel do
2:   Compute index  $I_i \leftarrow I(\vec{c}(\vec{x}_i))$  ▷ XYZ curve of Equation A.2
3:    $\text{indices}[i] \leftarrow I_i$ 
4:    $\text{counts}[I_i] \leftarrow \text{counts}[I_i] + 1$  ▷ atomically increment
5: end for

```

Step 2 – Exclusive Prefix Sum

```

6: Compute  $\text{counts}_{<}$  from parallel prescan of  $\text{counts}$  ▷ See [67, Harris et al.]

```

Step 3 – Counting Sort

```

7: for  $i \in \mathbb{N}_0^{N-1}$  in parallel do
8:    $I_i \leftarrow \text{indices}[i]$ 
9:    $o_1 \leftarrow \text{counts}_{<}[I_i]$  ▷ offset due to previous cells
10:   $o_2, \text{counts}[I_i] \leftarrow \text{counts}[I_i] - 1$  ▷ offset within cell, atomically decrement!
11:   $\text{sorted}[o_1 + o_2 - 1] \leftarrow i$ 
12: end for

```

Query

Using the array of sorted particle indices, the index function I and the counts of particles per cell as well as its prefix, all particles in a given cell can be queried in constant time. For a particle in cell I_i , all 3^d surrounding cells are queried. For each such surrounding cell with index I_j , the first particle in that cell has the index $\text{sorted}[\text{counts}_{<}]$ and the remaining particles in the same cell are the $\text{counts}[I_j]$ subsequent indices.

In fact, one optimization due to the use of the XYZ curve is that to find the neighbours of a particle in cell $I(k, l, m)$ in three dimensions, only the first particle in $I(k - 1, l, m)$ has to be found, after which the next $\sum_{i=-1}^{i=1} \text{counts}[I(k - i, l, m)]$ particles necessary to complete the query along the x-axis lie subsequent in memory, necessitating only nine coherent sections of memory to be read instead of 27 in the worst case for three dimensions.

Discussion

The algorithm shown [65] is simple, elegant, can achieve high performance and is very much suited for massively parallel hardware. One disadvantage however, is that the representation contains arrays that scale with the size of the simulation domain, which is a limitation compared to data structures that can more easily adapt to infinite domains and consume less memory for very sparsely filled domains, such as compact hashing [1, 51]. Since the explicit representation is small, with only a few 32-bit numbers stored per grid cell, this was not found to have a noticeable impact in this instance.

The simulation bounds $\vec{x}_{min}, \vec{x}_{max}$ must be known for the XYZ-curve to be applicable in the manner outlined here. In the implementation, the boundaries of the simulation domain are strictly enforced using clamping, such that \vec{x}_i is guaranteed to lie in the axis aligned bounding box spanned by $\vec{x}_{min}, \vec{x}_{max}$. A grid with two additional grid cells along each axis in both the positive and negative directions is then used, such that bound checks are unnecessary and branches in the hot code path of the query procedure are avoided.

In a simulation using structure-of-arrays data layouts, as is common in high-performance applications, particle attributes $\vec{x}_i, \vec{v}_i, m_i$ etc. can be re-sorted to restore memory coherency and make particles that are likely to be neighbours likely to be adjacent in memory, leading to less cache misses and better performance. One advantage of this sorting-based approach is that the sorted buffer is computed for neighbourhood computations and can be reused at no additional cost to perform such a resorting along the space-filling curve.

EXACT INERTIA TENSOR TERMS FOR TETRAHEDRA

We repeat the definition of the inertia tensor, the mass moments used in it and the equation used to compute these moments from subsection 4.2.1:

$$\mathbb{I} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{xy} & I_{yy} & -I_{yz} \\ -I_{xz} & -I_{yz} & I_{zz} \end{bmatrix} \quad (\text{B.1})$$

$$I_{xx} = \mathcal{M}_{200} \quad (\text{B.2})$$

$$I_{yy} = \mathcal{M}_{020} \quad (\text{B.3})$$

$$I_{zz} = \mathcal{M}_{002} \quad (\text{B.4})$$

$$I_{xy} = \mathcal{M}_{100} + \mathcal{M}_{010} \quad (\text{B.5})$$

$$I_{xz} = \mathcal{M}_{100} + \mathcal{M}_{001} \quad (\text{B.6})$$

$$I_{yz} = \mathcal{M}_{010} + \mathcal{M}_{001} \quad (\text{B.7})$$

$$\mathcal{M}_{pqr} = \sum_{\triangle ABC} \left(\operatorname{sgn}(\triangle_{ABCO}) \cdot \underbrace{\iiint_{\Omega(ABCO)} \rho x^p y^q z^r dV}_{\text{analytical solution for } \rho = \text{const}} \right) \quad (\text{B.8})$$

Only the solution to the volume integrals in Equation B.8 is yet to be shown. Exact analytic formulas for this integral in terms of the vertex coordinates $\vec{A}, \vec{B}, \vec{C}, \vec{D}$ of a general tetrahedron were taken from [48, Tonon], with the only difference being that a minor error in the paper (b' and c' are swapped) was fixed and that by using the origin $\vec{O} = \vec{D}$ as one of the four vertices, some terms drop out [48]:

$$\frac{I_{xx}(\triangle_{ABCO})}{\rho |\det(J(\triangle_{ABCO}))|} = \frac{(A_y^2 + A_y B_y + B_y^2 + A_y C_y + B_y C_y + C_y^2 + A_z^2 + A_z B_z + B_z^2 + A_z C_z + B_z C_z + C_z^2)}{60} \quad (\text{B.9})$$

$$\frac{I_{yy}(\triangle_{ABCO})}{\rho |\det(J(\triangle_{ABCO}))|} = \frac{(A_x^2 + A_x B_x + B_x^2 + A_x C_x + B_x C_x + C_x^2 + A_z^2 + A_z B_z + B_z^2 + A_z C_z + B_z C_z + C_z^2)}{60} \quad (\text{B.10})$$

$$\frac{I_{zz}(\triangle_{ABCO})}{\rho |\det(J(\triangle_{ABCO}))|} = \frac{(A_x^2 + A_x B_x + B_x^2 + A_x C_x + B_x C_x + C_x^2 + A_y^2 + A_y B_y + B_y^2 + A_y C_y + B_y C_y + C_y^2)}{60} \quad (\text{B.11})$$

$$\frac{I_{xy}(\triangle_{ABCO})}{\rho |\det(J(\triangle_{ABCO}))|} = \frac{(2A_x A_y + B_x A_y + C_x A_y + A_x B_y + 2B_x B_y + C_x B_y + A_x C_y + B_x C_y + 2C_x C_y)}{120} \quad (\text{B.12})$$

$$\frac{I_{xz}(\triangle_{ABCO})}{\rho |\det(J(\triangle_{ABCO}))|} = \frac{(2A_x A_z + B_x A_z + C_x A_z + A_x B_z + 2B_x B_z + C_x B_z + A_x C_z + B_x C_z + 2C_x C_z)}{120} \quad (\text{B.13})$$

$$\frac{I_{yz}(\triangle_{ABCO})}{\rho |\det(J(\triangle_{ABCO}))|} = \frac{(2A_y A_z + B_y A_z + C_y A_z + A_y B_z + 2B_y B_z + C_y B_z + A_y C_z + B_y C_z + 2C_y C_z)}{120} \quad (\text{B.14})$$

where $\rho |\det(J(\triangle_{ABCO}))|$ was moved to the left side of the equations to save space.

Since the origin was chosen as one of the vertices and the density is homogeneous, these rather lengthy expressions can be greatly simplified using vectorial notation. At the same time, the absolute value around the determinant can be dropped if the determinant is computed using the triple scalar product of Equation 4.36, which already contains $\text{sgn}(\Delta_{ABCO})$ and therefore yields values I'_{ij}, I'_{kk} that can be directly accumulated in a sum across all triangles Δ_{ABC} . Here, i, j, k are used to index the components of the vertices. Let $\vec{V} = (A_i, B_i, C_i)^T$ and $\vec{W} = (A_j, B_j, C_j)^T$ be the vectors of the i -th and j -th components of $\vec{A}, \vec{B}, \vec{C}$ for $i \neq j \neq k \neq i$ and one can obtain:

$$I'_{kk}(\Delta_{ABCO}) = \frac{\rho \det(J(\Delta_{ABCO}))}{120} \left(\vec{V}^2 + \left(|\vec{V}|_{L_1} \right)^2 + \vec{W}^2 + \left(|\vec{W}|_{L_1} \right)^2 \right) \quad (\text{B.15})$$

for the moments of inertia on the diagonal using $\vec{V}^2 = \vec{V} \cdot \vec{V}$ and the L_1 or sum norm $|\vec{V}|_{L_1} = A_i + B_i + C_i$ as well as:

$$I'_{ij}(\Delta_{ABCO}) = \frac{\rho \det(J(\Delta_{ABCO}))}{120} \left(\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \vec{V} \right) \cdot \vec{W} \quad (\text{B.16})$$

for the products of inertia using a matrix multiplication and a dot product, which is much more concise than the original formulation. These terms can trivially be computed on massively parallel hardware for each triangle Δ_{ABC} and summed using a parallel reduction to yield the components $I_{kk}^{sum}, I_{ij}^{sum}$ of the inertia tensor:

$$I_{kk}^{sum} = \sum_{\Delta_{ABC}} I'_{kk}(\Delta_{ABCO}) \quad (\text{B.17})$$

$$I_{ij}^{sum} = \sum_{\Delta_{ABC}} I'_{ij}(\Delta_{ABCO}) \quad (\text{B.18})$$

Lastly, note that the resulting elements of the tensor describe the inertia with respect to a rotation about the origin of the arbitrarily chosen coordinate system, whereas \mathbb{I} should in this case describe the inertia tensor for rotations about the centre of mass, which can be calculated as laid out in subsubsection 4.2.1. To transform the inertia tensor elements accordingly, the *transfer-of-axis relations* can be applied: [49]

$$I_{kk} = I_{kk}^{sum} - M(r_i^2 + r_j^2) \quad (\text{B.19})$$

$$I_{ij} = I_{ij}^{sum} - Mr_i r_j \quad (\text{B.20})$$

for $\vec{x}_{cm} = (r_x, r_y, r_z)^T$ and a body of mass M .

Bibliography

- [1] D. Koschier, J. Bender, B. Solenthaler, and M. Teschner, “Smoothed Particle Hydrodynamics Techniques for the Physics Based Simulation of Fluids and Solids,” in *Eurographics 2019 - Tutorials*, W. Jakob and E. Puppo, Eds., The Eurographics Association, 2019.
- [2] Y. Hu, T.-M. Li, L. Anderson, J. Ragan-Kelley, and F. Durand, “Taichi: A language for high-performance computation on spatially sparse data structures,” *ACM Trans. Graph.*, vol. 38, no. 6, Nov. 2019, ISSN: 0730-0301.
- [3] Blender Online Community, *Blender - a 3D modelling and rendering package*, (software, <https://www.blender.org/>), Stichting Blender Foundation, Amsterdam, 2025.
- [4] J. Bender and D. Koschier, “Divergence-free smoothed particle hydrodynamics,” in *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, ser. SCA ’15, Los Angeles, California: Association for Computing Machinery, 2015, pp. 147–155, ISBN: 9781450334969.
- [5] J. Bender, L. Westhofen, and S. R. Jeske, “Consistent SPH Rigid-Fluid Coupling,” in *Vision, Modeling, and Visualization*, The Eurographics Association, 2023, ISBN: 978-3-03868-232-5.
- [6] J. D. Anderson, *Computational Fluid Dynamics: The Basics with Applications*. (McGraw-Hill series in mechanical engineering). 1995.
- [7] J. Karrer, “Equation of State Solvers for Smoothed Particle Hydrodynamics,” (Lab Course Report), 2024.
- [8] W. M. Lai, D. H. Rubin, D. Rubin, and E. Krempl, *Introduction to continuum mechanics*, Third. Butterworth-Heinemann, 1999.
- [9] V. John, *Finite Element Methods for Incompressible Flow Problems* (Springer Series in Computational Mathematics 51). Springer, Jan. 2016, vol. 51, ISBN: 978-3-319-45749-9.
- [10] A. Falaschi, *Signal Processing and Information Theory*, 1st ed. Jul. 2022.
- [11] D. J. Price, “Smoothed particle hydrodynamics and magnetohydrodynamics,” *Journal of Computational Physics*, vol. 231, no. 3, pp. 759–794, Feb. 2012.
- [12] P. Getreuer, “A Survey of Gaussian Convolution Algorithms,” *Image Processing On Line*, vol. 3, pp. 286–310, 2013.
- [13] S. Band, *Boundary Handling and Neighbor Search in Iterative Incompressible SPH*, (PhD Thesis), 2020.
- [14] S. Boucheron, G. Lugosi, and P. Massart, *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press, Feb. 2013, ch. 3 - Bounding the variance, pp. 52–57, ISBN: 9780199535255.
- [15] M. Müller, D. Charypar, and M. Gross, “Particle-based fluid simulation for interactive applications,” in *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA ’03, San Diego, California: Eurographics Association, 2003, pp. 154–159, ISBN: 1581136595.
- [16] J. R. Macdonald, “Review of Some Experimental and Analytical Equations of State,” *Rev. Mod. Phys.*, vol. 41, pp. 316–349, 2 Apr. 1969.
- [17] M. Becker and M. Teschner, “Weakly compressible SPH for free surface flows,” in *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA ’07, San Diego, California: Eurographics Association, 2007, pp. 209–217, ISBN: 9781595936240.
- [18] M. Ihmsen, J. Cornelis, B. Solenthaler, C. Horvath, and M. Teschner, “Implicit Incompressible SPH,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 3, pp. 426–435, 2014.
- [19] J. Cornelis, M. Ihmsen, A. Peer, and M. Teschner, “IISPH-FLIP for incompressible fluids,” *Computer Graphics Forum*, vol. 33, no. 2, pp. 255–262, 2014.

- [20] J. J. Monaghan, "Smoothed Particle Hydrodynamics," *Annual Review of Astronomy and Astrophysics*, vol. 30, no. Volume 30, 1992, pp. 543–574, 1992, ISSN: 1545-4282.
- [21] M. Ihmsen, N. Akinci, M. Gissler, and M. Teschner, "Boundary Handling and Adaptive Time-stepping for PCISPH," in *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS"* (2010), K. Erleben, J. Bender, and M. Teschner, Eds., The Eurographics Association, 2010, ISBN: 978-3-905673-78-4.
- [22] D. Violeau and A. Leroy, "Optimal time step for incompressible SPH," *Journal of Computational Physics*, vol. 288, pp. 119–130, 2015, ISSN: 0021-9991.
- [23] E. Haines, "Point in polygon strategies," in *Graphics Gems IV*. USA: Academic Press Professional, Inc., 1994, pp. 24–46, ISBN: 0123361559.
- [24] Dawson-Haggerty et al., *Trimesh*, version 4.5.2, (software, <https://trimesh.org/>).
- [25] J.-P. Fürstenau, B. Avci, and P. Wriggers, "A comparative numerical study of pressure-Poisson-equation discretization strategies for SPH," in *(12th international SPHERIC workshop, Ourense, Spain)*, Jun. 2017.
- [26] J. Cornelis, J. Bender, C. Gissler, M. Ihmsen, and M. Teschner, "An optimized source term formulation for incompressible SPH," *The Visual Computer*, vol. 35, no. 4, pp. 579–590, Apr. 2019, ISSN: 1432-2315.
- [27] M. Ihmsen, J. Orthmann, B. Solenthaler, A. Kolb, and M. Teschner, "SPH Fluids in Computer Graphics," in *Eurographics 2014 - State of the Art Reports*, S. Lefebvre and M. Spagnuolo, Eds., The Eurographics Association, 2014.
- [28] T. Probst and M. Teschner, "Monolithic friction and contact handling for rigid bodies and fluids using sph," *Computer Graphics Forum*, vol. 42, no. 1, pp. 155–179, 2023.
- [29] D. Koschier, J. Bender, B. Solenthaler, and M. Teschner, "A survey on sph methods in computer graphics," *Computer Graphics Forum*, vol. 41, no. 2, pp. 737–760, 2022.
- [30] J. Bender et al., *SPlisHSPlasH Library*, (software, <https://github.com/InteractiveComputerGraphics/SPlisHSPlasH>).
- [31] A. Bardou, "Online Learning for the Black-Box Optimization of Wireless Networks," Theses, Ecole normale supérieure de Lyon - ENS LYON, Sep. 2023.
- [32] P. Brunzema, A. Von Rohr, and S. Trimpe, "On Controller Tuning with Time-Varying Bayesian Optimization," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, 2022, pp. 4046–4052.
- [33] N. Bastianello, "tvopt: A Python Framework for Time-Varying Optimization," in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 227–232.
- [34] D. Delahaye, S. Chaimatanan, and M. Mongeau, "Simulated Annealing: From Basics to Applications," in *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds. Cham: Springer International Publishing, 2019, pp. 1–35, ISBN: 978-3-319-91086-4.
- [35] M. Bédard, "Optimal acceptance rates for Metropolis algorithms: Moving beyond 0.234," *Stochastic Processes and their Applications*, vol. 118, no. 12, pp. 2198–2222, 2008, ISSN: 0304-4149.
- [36] A. Li, L. Wang, T. Dou, and J. S. Rosenthal, *Exploring the generalizability of the optimal 0.234 acceptance rate in random-walk Metropolis and parallel tempering algorithms*, 2024.
- [37] 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, Jul. 2012, ISSN: 0730-0301.
- [38] J. Huang, H. Su, and L. J. Guibas, "Robust watertight manifold surface generation method for shapenet models," *CoRR*, vol. abs/1802.01698, 2018.
- [39] R. Bridson, "Fast poisson disk sampling in arbitrary dimensions," in *ACM SIGGRAPH 2007 Sketches*, ser. SIGGRAPH '07, San Diego, California: Association for Computing Machinery, 2007, 22–es, ISBN: 9781450347266.
- [40] Francis Williams, *Point cloud utils*, version 4.5.2, (software, <https://www.github.com/fwilliams/point-cloud-utils>), 2022.
- [41] D. Koschier and J. Bender, "Density maps for improved sph boundary handling," in *Proceedings of the 2017 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ser. SCA '17, ACM, 2017, pp. 1–10.

- [42] Y. Hu, J. Liu, X. Yang, *et al.*, “QuanTaichi: a compiler for quantized simulations,” *ACM Trans. Graph.*, vol. 40, no. 4, Jul. 2021, ISSN: 0730-0301.
- [43] S. Band, C. Gissler, M. Ihmsen, J. Cornelis, A. Peer, and M. Teschner, “Pressure Boundaries for Implicit Incompressible SPH,” vol. 37, no. 2, Feb. 2018, ISSN: 0730-0301.
- [44] S. Band, C. Gissler, A. Peer, and M. Teschner, “MLS pressure extrapolation for the boundary handling in divergence-free SPH,” ser. VRIPHYS ’18, Delft, The Netherlands: Eurographics Association, 2018, pp. 55–63.
- [45] H. Goldstein, J. L. Safko, and C. P. Poole, *Classical mechanics: Pearson new international edition*, en, 3rd ed. Pearson Higher Ed, Mar. 2014.
- [46] D. Baraff, “Physically Based Modeling: Rigid Body Simulation,” *SIGGRAPH Course Notes*, ACM SIGGRAPH, vol. 2, no. 1, pp. 2–1, 2001.
- [47] C. Zhang and T. Chen, “Efficient feature extraction for 2D/3D objects in mesh representation,” in *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*, vol. 3, 2001, 935–938 vol.3.
- [48] F. Tonon, “Explicit Exact Formulas for the 3-D Tetrahedron Inertia Tensor in Terms of its Vertex Coordinates,” *Journal of Mathematics and Statistics*, vol. 1, pp. 8–11, Mar. 2005.
- [49] B. Mirtich, “Fast and accurate computation of polyhedral mass properties,” *J. Graphics, GPU, & Game Tools*, vol. 1, pp. 31–50, 1996.
- [50] M. Ihmsen, N. Akinci, G. Akinci, and M. Teschner, “Unified spray, foam and air bubbles for particle-based fluids,” *Vis. Comput.*, vol. 28, no. 6–8, pp. 669–677, Jun. 2012, ISSN: 0178-2789.
- [51] S. Band, C. Gissler, and M. Teschner, “Compressed Neighbour Lists for SPH,” *Computer Graphics Forum*, vol. 39, no. 1, pp. 531–542, 2020.
- [52] F. Da, D. Hahn, C. Batty, C. Wojtan, and E. Grinspun, “Surface-only liquids,” *ACM Trans. Graph.*, vol. 35, no. 4, Jul. 2016, ISSN: 0730-0301.
- [53] F. Löschner, T. Böttcher, S. R. Jeske, and J. Bender, “Weighted Laplacian Smoothing for Surface Reconstruction of Particle-based Fluids,” in *Vision, Modeling, and Visualization*, The Eurographics Association, 2023, ISBN: 978-3-03868-232-5.
- [54] J. F. Blinn, “A Generalization of Algebraic Surface Drawing,” *ACM Trans. Graph.*, vol. 1, no. 3, pp. 235–256, Jul. 1982, ISSN: 0730-0301.
- [55] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” ser. SIGGRAPH ’87, New York, NY, USA: Association for Computing Machinery, 1987, pp. 163–169, ISBN: 0897912276.
- [56] Y. Zhu and R. Bridson, “Animating sand as a fluid,” in *ACM SIGGRAPH 2005 Papers*, ser. SIGGRAPH ’05, Los Angeles, California: Association for Computing Machinery, 2005, pp. 965–972, ISBN: 9781450378253.
- [57] J. Yu and G. Turk, “Reconstructing surfaces of particle-based fluids using anisotropic kernels,” *ACM Trans. Graph.*, vol. 32, no. 1, Feb. 2013, ISSN: 0730-0301.
- [58] T. Ju, F. Losasso, S. Schaefer, and J. Warren, “Dual contouring of hermite data,” *ACM Trans. Graph.*, vol. 21, no. 3, pp. 339–346, Jul. 2002, ISSN: 0730-0301.
- [59] M. Ihmsen, N. Akinci, G. Akinci, and M. Teschner, “Unified spray, foam and air bubbles for particle-based fluids,” *Vis. Comput.*, vol. 28, no. 6–8, pp. 669–677, Jun. 2012, ISSN: 0178-2789.
- [60] J. Bender, D. Koschier, T. Kugelstadt, and M. Weiler, “Turbulent Micropolar SPH Fluids with Foam,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 6, pp. 2284–2295, 2019.
- [61] B. Solenthaler and R. Pajarola, “Predictive-corrective incompressible sph,” in *ACM SIGGRAPH 2009 Papers*, ser. SIGGRAPH ’09, New Orleans, Louisiana: Association for Computing Machinery, 2009, ISBN: 9781605587264.
- [62] T. Probst and M. Teschner, “Unified Pressure, Surface Tension and Friction for SPH Fluids,” *ACM Trans. Graph.*, vol. 44, no. 1, Dec. 2024, ISSN: 0730-0301.
- [63] M. Weiler, D. Koschier, M. Brand, and J. Bender, “A Physically Consistent Implicit Viscosity Solver for SPH Fluids,” *Computer Graphics Forum*, vol. 37, no. 2, pp. 145–155, 2018.
- [64] A. Peer, C. Gissler, S. Band, and M. Teschner, “An Implicit SPH Formulation for Incompressible Linearly Elastic Solids,” *Computer Graphics Forum*, vol. 37, no. 6, pp. 135–148, 2018.

- [65] R. Hoetzlein, “Fast Fixed-Radius Nearest Neighbors: Interactive Million-Particle Fluids,” (talk slides), GPU Technology Conference (GTC) 2014, Santa Clara, CA, 2014.
- [66] H. Tropf and H. Herzog, “Multimdimensional Range Search in Dynamically Balanced Trees,” *Angew. Inform.*, vol. 23, pp. 71–77, 1981.
- [67] M. Harris, S. Sengupta, and J. D. Owens, “Parallel prefix sum (scan) with CUDA,” in *GPU Gems 3*, H. Nguyen, Ed., Addison Wesley, Aug. 2007, ch. 39, pp. 851–876.
- [68] G. E. Blelloch, “Prefix sums and their applications,” Jun. 2018.