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

Chapter 1

Introduction

This report aims to outline a complete implementation of a robust, fully GPU-parallel particle based solver for incompressible fluid flow problems with weakly coupled rigid bodies that can two-way interact with the fluid. The solver is efficient enough to make simulations with high spatial resolution and corresponding particle counts well into the millions feasible on a single machine with consumer hardware. Diffuse spray, foam and air bubbles are generated in a physically motivated way to enhance visual realism for simulations of sufficiently large spatial scale for such effects to occur.

The governing equations to be solved, namely the incompressible Navier-Stokes equations in Lagrangian form are introduced and discretized using the Smoothed Particle Hydrodynamics scheme, or SPH for short, in chapter 2.

This leads to a set of equations that can be worked into a fluid solver in chapter 3. Most challengingly, this solver must uphold incompressibility, leading to a pressure solver that must solve the pressure Poisson equation, in this case using a relaxed Jacobi solver with different choices of source term that lead to different behaviour of the solver.

Once the fluid solver is completed, including one-way coupling to boundaries as discussed in chapter 4, the remainder of the chapter discusses a second, rigid body solver, that uses an SPH-based surface representation to interact with the former and enable two-way interactions of rigid bodies with fluids. Challenges include a robust sampling of surfaces, the accurate calculation of the required volume and moments of mass of triangular meshes and

Chapter 2

Smoothed Particle Hydrodynamics

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 very concisely outlined in the following and the interested reader is referred to more extensive works such as by [1, Anderson] or a previous report authored by the present writer [2] for a more thorough derivation.

Lagrangian Continuum Mechanics

Since the flow problems in question occur not a microscopic scale, such as concerning the interaction of individual molecules, but 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 quantities [1]:

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

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 $\vec{\boldsymbol{v}}$. 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 [1]. We choose the Lagrangian representation in the following since the continuum itself is later discretized into particles that are advected with the 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 [3].

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 [2] into the continuity equation of the form:

Continuity Equation

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

for incompressible flows, which has to hold across the entire volume integral over the fluid domain [1]. 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 [4]:

1. $\frac{D\rho}{Dt} = 0$, the density of any fluid element cannot change

2. $\nabla \cdot \vec{\boldsymbol{v}} = 0$ 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 section 3.4.

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 $\vec{\boldsymbol{t}}$ acting in normal direction $\hat{\vec{\boldsymbol{n}}}$ on the surface of each fluid element on one side, and external body forces per unit mass $\vec{\boldsymbol{b}}^{ext}$ 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 $\mathbb{T}\hat{\vec{\boldsymbol{n}}} = \vec{\boldsymbol{t}}$ and applying the divergence theorem as well as the continuity equation to the resulting integral equation yields that for every fluid element it must hold [4]:

CAUCHY MOMENTUM EQUATION

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

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 [5]. 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 [4]. Assuming incompressible flow and a linearly viscous or Newtonian fluid, one can obtain [3, 5]:

CONSTITUTIVE RELATION

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

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 [3], 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 [5] one can finally obtain the Navier-Stokes momentum equation for incompressible, Newtonian fluids in Lagrangian form:

NAVIER STOKES MOMENTUM EQUATION

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

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 such flows [1], 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, will 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, 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 [6]:

DIRAC δ-DISTRIBUTION

$$\int_{-\infty}^{\infty} \delta(x)\,dx = 1 \qquad \textit{normalization} \tag{2.6}$$

$$x \neq 0 \implies \delta(x) = 0 \tag{2.7}$$

and analogously for higher dimensional arguments $\delta(\vec{\boldsymbol{x}})$. 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 [6]:

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

Which means that by the convolution theorem [3]:

$$f(\vec{\boldsymbol{x}}) = \int_{\Omega} f(\vec{\boldsymbol{x}}')\delta(\vec{\boldsymbol{x}} - \vec{\boldsymbol{x}}')\,dV' = (f \star \delta)(\vec{\boldsymbol{x}}) \tag{2.9}$$

where $\star$ 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 $\vec{\boldsymbol{x}}_i$ then reads:

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

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

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

where $\mathcal{N}_i = \{j : |\vec{\boldsymbol{x}}_{ij}| < \hbar\}$ is the set of neighbour samples of i within some radius $\hbar$, which is finite for compactly supported W. Denoting quantities $f(\vec{\boldsymbol{x}}_i)$ 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

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

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 0^{th}- and 1^{st}-order consistency respectively to be achieved, namely [7]:

CONSISTENCY CRITERIA

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

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

Note that the low asymptotic error order for arbitrary samplings appears to have questionable practical relevance [3] - 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 [7, 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 ensure Equation 2.15 [3, 7]. Similar consistency conditions exist for SPH approximations of differential operators [7].

Kernel Function

The kernel function is parametrized by $\vec{\boldsymbol{x}}_{ij} = \vec{\boldsymbol{x}}_i - \vec{\boldsymbol{x}}_j$ for samples i, j and a kernel support radius $\hbar$. We denote $W_{ij} := W(\vec{\boldsymbol{x}}_{ij}, \hbar)$ 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 [3]:

$$\int_\Omega W(\vec{\boldsymbol{x}}_{ij}, \hbar) = 1 \qquad \textit{normalization} \tag{2.16}$$

$$\lim_{\hbar \to 0} W(\vec{\boldsymbol{x}}_{ij}, \hbar) = \delta(\vec{\boldsymbol{x}}_{ij}) \qquad \textit{Dirac-}\delta\textit{ condition} \tag{2.17}$$

Further useful properties include [3]:

$$|\vec{\boldsymbol{x}}_{ij}| > \hbar \implies W(\vec{\boldsymbol{x}}_{ij}, \hbar) \qquad \textit{compact support} \tag{2.18}$$

$$W(\vec{\boldsymbol{x}}_{ij}, \hbar) = W(\vec{\boldsymbol{x}}_{ji}, \hbar) \qquad \textit{spherical symmetry} \tag{2.19}$$

$$W(\vec{\boldsymbol{x}}_{ij}, \hbar) > 0 \qquad \textit{positivity} \tag{2.20}$$

$$W(\vec{\boldsymbol{x}}_{ij}, \hbar) \in C^2 \qquad \textit{twice continuously differentiable} \tag{2.21}$$

since compact support can decrease the number of non-zero contributions for non-degenerate cases from $\mathcal{O}\left(N^2\right)$ to $\mathcal{O}\left(N\right)$ in the number N of samples and is vital for computational efficiency. Spherical symmetry and normalization together enable methods for ensuring first order consistency [3]. Positivity is vital for approximations of quantities such as absolute pressure, mass density etc. that must not be negative and can be neglected otherwise, while C^2-continuity is convenient for the discretization of second-order partial differential equations [3].

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 [8], 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 [9]:

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

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

$$\tag{2.24}$$

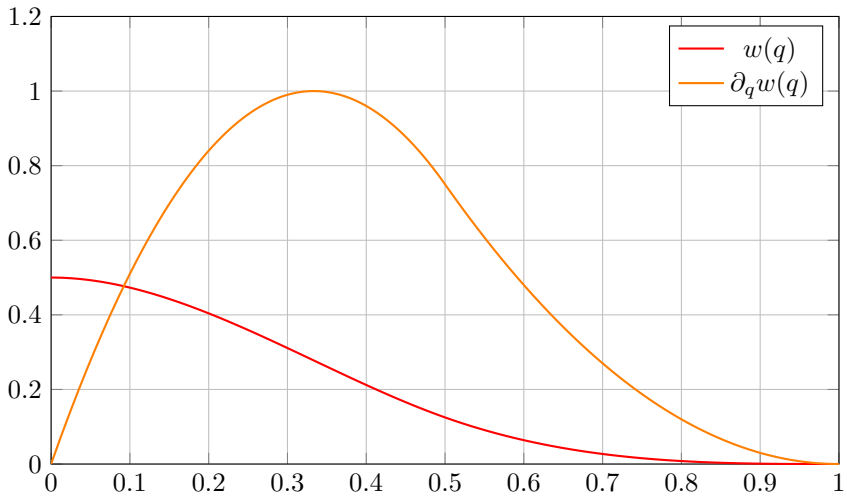


Figure 2.1: The shape function $w(q)$ that defines the Cubic Spline kernel function and its first derivative are plotted for all values of $q \in [0, 1]$ that yield non-zero $w(q)$. The function is compactly supported. Since the kernel function W is spherically symmetric, this one-dimensional function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

where $q = \frac{|\vec{x}|}{\hbar} \in [0, 1]$ and $\alpha(d)$ depends only on dimensionality. As in much of the literature, the Cubic Spline kernel is employed in this report [9]:

$$w(q) = (1-q)_+^3 - 4\,(1/2 - q)_+^3 \tag{2.25}$$

$$\partial_q w(q) = -3\,(1-q)_+^2 - 12\,(1/2 - q)_+^2 \tag{2.26}$$

$$\alpha(3) = 16/\pi \tag{2.27}$$

where $(x)_+ := \max(0, x)$. These functions are illustrated in Figure 2.1. As is often recommended [3], 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 to directly to approximate differential operators such as gradient, divergence and rotation on vector fields, the least straightforward example amongst them perhaps being:

$$\left\langle \nabla \vec{\boldsymbol{f}}_i \right\rangle_\otimes = \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} \vec{\boldsymbol{f}}_j \otimes \nabla W_{ij} \tag{2.28}$$

where $\otimes$ is the dyadic product [3]. 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 [3, 7]:

DIFFERENCE FORMULA

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

Another useful alternative discretization for the gradient of a scalar field is the **SYMMETRIC FORMULA**, which can be derived from the discrete Lagrangian particularly such that when applied to the pressure field for example and a standard approximation of density is used, symmetric forces that conserve linear and angular momentum can be obtained [3, 7]:

SYMMETRIC FORMULA

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

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 [3].

Lastly, second derivatives must be discretized, particularly the Laplace operator necessary to describe dissipative terms such as viscosity [7]. 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 suffer[1]. Instead, discretizations that rely on the kernel gradient approximation are typically preferred [3]. Amongst them, the following discretization of the Laplace operator is especially useful since for a divergence-free field $\vec{\boldsymbol{f}}(\vec{\boldsymbol{x}})$ 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 [3]:

LAPLACE OPERATOR DISCRETIZATION

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

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 to the average density across particles $1/N \sum_i^N \rho_i$, where ρ_i is SPH-discretized, depending 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.

[1] It 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 [10, 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.

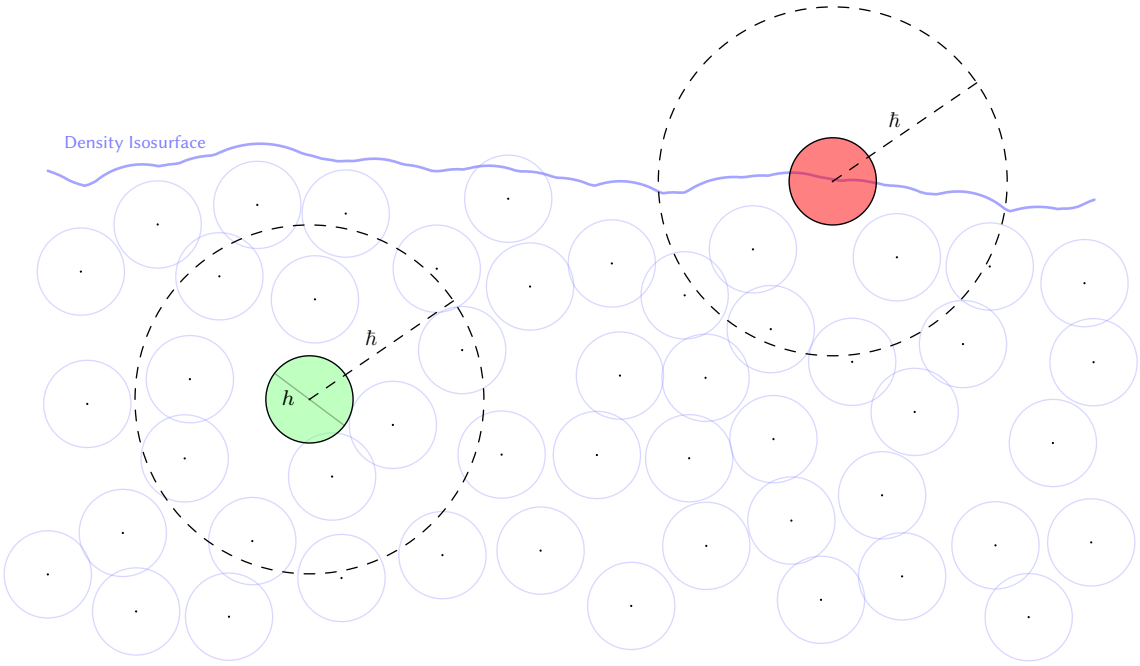


Figure 2.2: The particle deficiency phenomenon is illustrated in two dimensions, where particles positions are shown as dots with a surrounding circle of a diameter equal to the expected particle spacing h (Note that particles are *not* modelled as spheres but some volume of unspecified shape - only 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.

Chapter 3

Incompressible Fluid Solver

With the SPH method from 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 such 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 iteratively solved by the Implicit Incompressible SPH solver of subsubsection 3.3. Multiple source terms and variations of the solver are available, which are 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 $\vec{\boldsymbol{a}}_i = \frac{D\vec{\boldsymbol{v}}_i}{Dt}$ necessary to compute particle trajectories using Newton's second law, while the continuity equation can be seen as a constraint on the former [3], ensuring incompressibility. Firstly, recall the momentum equation for some particle i:

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

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 accurate fashion using the discrete Laplace operator as defined in Equation 2.31:

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

Note the additional small term ϵ in the denominator that 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 [3]:

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

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

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 [3]:

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

Since the mass m_i is set at the start of the simulation, $\nu, \vec{\boldsymbol{g}}$ are given and initial $\vec{\boldsymbol{x}}_i, \vec{\boldsymbol{v}}_i$ 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 $p = k\rho$ [11], the Murnaghan equation of state [12]

as used in Weakly Compressible SPH [13] $p = k\left(\left(\rho/\rho_0\right)^\gamma - 1\right)$ 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 [3] 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, the most prevalent scheme [3, 13, 14, 15, 16] to this end is symplectic or semi-implicit Euler time integration:

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

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

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 later. A common upper bound on Δt is the Courant-Friedrichs-Lewy or CFL condition, which states that particles should not move further than some fraction $\lambda \in \left]0, 1\right]$ of their size or spacing h within one time step, which for a global adaptive time step size implies [3]:

$$\Delta t \leq \lambda \frac{h}{\max_i \left|\vec{\boldsymbol{v}}_i\right|} \tag{3.7}$$

with a common choice [16, 17] being $\lambda = 0.4$. Alternative formulations based on the maximum accelerating forces [17, 18] or more elaborately derived formulas that differ per kernel function and pressure solver [19] 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 [20] and a fluid particle is instantiated, otherwise the candidate is discarded[1]. 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 unnaturally high pressure values at t_0.
- The second problem may be alleviated by applying a jitter on initial positions in conjunction with non-uniform particle masses [2]. Sampling particles of uniform mass causes a density gradient due to particle deficiency towards the boundary of the fluid volume, which can in turn result in an erroneous pressure gradient that causes the fluid to expand towards the boundary, causing an

[1] The implementation of this report uses the `trimesh.ray.ray_pyembree.RayMeshIntersector` implementation of a ray cast operator [21]

'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:

- $$m_i^0 = \frac{1}{h^d} \tag{3.8}$$
$$\langle \rho_i \rangle^l \leftarrow \sum_{j \in \mathcal{N}_i} m_j^l W_{ij} \tag{3.9}$$
$$m_i^{l+1} \leftarrow m_i^l \cdot \frac{\rho_0}{\langle \rho_i \rangle^l} \tag{3.10}$$

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 $10^{-10}\,\frac{\mathrm{kg}}{\mathrm{m}^3}$ 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 exceptionally prone to aliasing artefacts in the initial stages of simulation. Even just a jitter of $\overrightarrow{\Delta x} \sim 0.01h \cdot \mathcal{N}\left(\vec{\boldsymbol{\mu}} = \vec{\mathbf{0}}, \vec{\boldsymbol{\sigma}} = \vec{\mathbf{1}}\right)$ 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 persistently improve the amorphousness of the particle sampling, reducing numerical viscosity [2].

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 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 [14].

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 [3], 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 this issue, the solution of the Navier-Stokes equation can be decomposed into sequential subproblems, namely into pressure accelerations and non-pressure accelerations such as caused by gravity and viscosity [3]. 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 [3]. This is also referred to as **OPERATOR SPLITTING** [3] and implemented here as computing the predicted velocity $\vec{\boldsymbol{v}}_i^*$ due to non-pressure accelerations:

$$\vec{\boldsymbol{v}}_i^*(t) = \vec{\boldsymbol{v}}(t) + \Delta t\left(\vec{\boldsymbol{g}} + \nu\,\nabla^2 \vec{\boldsymbol{v}}_i(t)\right) \tag{3.11}$$

before implicitly integrating the pressure acceleration $\vec{\boldsymbol{a}}_i^p$ 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 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 [3]. Pressure values that strongly enforce incompressibility, or in other words act as Lagrange multipliers to the continuity equation Equation 2.2 [3], 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 $\vec{\boldsymbol{v}}_i^*$ would result in positions $\vec{\boldsymbol{x}}_i^*$ that can be used to compute a predicted density ρ_i^* using, for example, Equation 4.7. The incompressibility constraint then means that:

$$\forall i : \rho_i \left(t + \Delta t\right) = \rho_0 \tag{3.12}$$

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

Updating particle positions to $\vec{\boldsymbol{x}}_i^*$ 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 $\left({}^{D\rho}/_{Dt}\right)^*$ due to $\vec{\boldsymbol{v}}_i^*$ to estimate the density at the next time step [3]:

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

$$\rho^* = \rho + \Delta t \left(-\rho \nabla \cdot \vec{\boldsymbol{v}}^*\right) \qquad \textit{insert continuity equation 2.2} \tag{3.14}$$

$$\rho^* = \rho - \Delta t \rho \nabla \cdot \vec{\boldsymbol{v}}^* \tag{3.15}$$

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:

$$\rho_p^* = \rho - \Delta t \rho \nabla \cdot \left(\Delta t \vec{\boldsymbol{a}}_p\right) \qquad \textit{Equation 3.15 using } \vec{\boldsymbol{v}}_p^* = \Delta t \vec{\boldsymbol{a}}_p \textit{ instead of } \vec{\boldsymbol{v}}^* \tag{3.16}$$

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

$$= \rho + \Delta t^2 \nabla \cdot \left(\nabla p\right) \qquad \textit{simplify} \tag{3.18}$$

$$\rho_p^* = \rho + \Delta t^2 \, \nabla^2 p \qquad \nabla \cdot \left(\nabla p\right) = \nabla^2 p \tag{3.19}$$

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:

$$\rho \left(t + \Delta t\right) \stackrel{!}{=} \rho_0 \qquad \textit{incompressibility constraint Item 1, Equation 3.12} \tag{3.20}$$

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

$$\rho^* \cancel{+\rho} + \Delta t^2 \, \nabla^2 p \cancel{-\rho} \stackrel{!}{=} \rho_0 \qquad \textit{insert Equation 3.19} \tag{3.22}$$

$$\Delta t \, \nabla^2 p \stackrel{!}{=} \frac{\rho_0 - \rho^*}{\Delta t} \qquad \textit{rearrange} \tag{3.23}$$

which is one form of the pressure Poisson equation [3]. 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:

$$\nabla\cdot\left(\vec{\boldsymbol{v}}\left(t+\Delta t\right)\right) \stackrel{!}{=} 0 \qquad \textit{divergence free velocity constraint 2} \tag{3.24}$$

$$\nabla\cdot\left(\vec{\boldsymbol{v}}^{*}-\Delta t\frac{1}{\rho}\nabla p\right) \stackrel{!}{=} 0 \qquad \textit{insert } \vec{\boldsymbol{v}}_{p}^{*}=\Delta t\left(-\frac{1}{\rho}\nabla p\right) \tag{3.25}$$

$$\nabla\cdot\vec{\boldsymbol{v}}^{*}-\Delta t\frac{1}{\rho}\nabla^{2}p \stackrel{!}{=} 0 \qquad \textit{linearity of divergence} \tag{3.26}$$

$$\Delta t\,\nabla^{2}p \stackrel{!}{=} \rho\nabla\cdot\vec{\boldsymbol{v}}^{*} \qquad \textit{rearrange} \tag{3.27}$$

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 [22]
2. whether to use density invariance (Equation 3.23)[14], velocity divergence (Equation 3.27) or some combination of both terms [16, 23] as source terms for the solver
3. which numerical solver to use

This report takes the **Implicit Incompressible SPH** or IISPH method by [14, 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:

$$\Delta t^{2}\,\nabla^{2}p=\rho_{0}-\rho^{*} \tag{3.28}$$

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 [17, 24]:

$$\nabla\cdot\vec{\boldsymbol{f}}=\frac{1}{\rho}\left(\nabla\cdot\left(\vec{\boldsymbol{f}}\rho\right)-\vec{\boldsymbol{f}}\cdot\nabla\rho\right) \qquad \textit{after [17, Monaghan 1992]} \tag{3.29}$$

$$\left\langle\nabla\cdot\vec{\boldsymbol{f}}\right\rangle=\underbrace{\frac{1}{\rho_{i}}\sum_{j\in\mathcal{N}_{i}}\frac{m_{j}}{\rho_{j}}\left(\rho_{j}\vec{\boldsymbol{f}}_{j}\right)\cdot\nabla W_{ij}}_{\left\langle\nabla\cdot\left(\vec{\boldsymbol{f}}\rho\right)\right\rangle}-\underbrace{\frac{1}{\rho_{i}}\vec{\boldsymbol{f}}_{i}\cdot\sum_{j\in\mathcal{N}_{i}}m_{j}\nabla W_{ij}}_{\vec{\boldsymbol{f}}_{i}\cdot\left\langle\nabla\rho\right\rangle} \qquad \textit{SPH sum (Equation 2.13)} \tag{3.30}$$

which simplifies to:

SPH divergence discretization

$$\left\langle\nabla\cdot\vec{\boldsymbol{f}}\right\rangle_{\nabla\cdot}=-\frac{1}{\rho_{i}}\sum_{j\in\mathcal{N}_{i}}m_{j}\vec{\boldsymbol{f}}_{ij}\cdot\nabla W_{ij} \tag{3.31}$$

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

$$\langle \rho_0 - \rho^* \rangle = \rho_0 - (\rho_i - \Delta t \rho_i \langle \nabla \cdot \vec{\boldsymbol{v}}_i^* \rangle_{\nabla \cdot}) \qquad \textit{Definition of } \rho^* \textit{ (Equation 3.15)} \tag{3.32}$$

$$= \rho_0 - \left(\rho_i + \Delta t \cancel{\rho_i} \left(\frac{\cancel{1}}{\cancel{\rho_i}} \sum_{j \in \mathcal{N}_i} m_j \vec{\boldsymbol{v}}_{ij}^* \cdot \nabla W_{ij} \right) \right) \qquad \textit{apply } \nabla \cdot \textit{ discretization (Equation 3.31)} \tag{3.33}$$

$$= \rho_0 - \rho_i - \Delta t \sum_{j \in \mathcal{N}_i} m_j \vec{\boldsymbol{v}}_{ij}^* \cdot \nabla W_{ij} \qquad \textit{simplify} \tag{3.34}$$

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

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

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

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 $\vec{\boldsymbol{a}}_i^p$ has been computed from each p_i [3]:

$$\left\langle \Delta t^2 \nabla \cdot (\nabla p) \right\rangle = \Delta t^2 \langle \nabla \cdot (-\rho_i \vec{\boldsymbol{a}}_i^p) \rangle_{\nabla \cdot} \qquad \textit{since } \vec{\boldsymbol{a}}_i^p = -\frac{1}{\rho_i} \nabla p \textit{ (Equation 3.1)} \tag{3.36}$$

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

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

which extends when considering boundary particles to a term:

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

with the simplifying assumption that pressure accelerations $\vec{\boldsymbol{a}}_k^p$ 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 [14, Implicit Incompressible SPH], which is presented here, to Equation 2.31 we refer to [22, Fürstenau 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 [14, as alternatively proposed by Ihmsen et al.], and more recently Nonsmooth Nonlinear Conjugate Gradient methods as suggested by [25, Probst and Teschner].

A relatively straightforward implementation that is robust, matrix-free, i.e. does not rely on explicitly constructing system matrices of prohibitive $\mathcal{O}\left(N^2\right)$ proportions, and which can be fully parallelized is achieved by a **Relaxed Jacobi Solver** as suggested in Implicit Incompressible SPH [14, 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:

$$\left\langle \Delta t^2 \nabla^2 p_i^2 \right\rangle = \rho_i - \rho_i^* \tag{3.40}$$

where $\left\langle \Delta t^2 \nabla^2 p_i^2 \right\rangle$ 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:

$$\mathbb{A}\vec{\boldsymbol{p}} = \vec{\boldsymbol{s}} \tag{3.41}$$

where $\vec{\boldsymbol{s}}$ is the vector of all source terms $s_i = \rho_i - \rho_i^*$ which are known from Equation 3.35 and $\vec{\boldsymbol{p}}$ 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 $(\mathbb{A}\vec{\boldsymbol{p}})_i$. This is precisely the same as $\left\langle \Delta t^2 \nabla^2 p_i \right\rangle$ 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.
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 $(\mathbb{A}\vec{\boldsymbol{p}})_i$ that is known from Equation 3.39, and apply the partial derivative with respect to the i-th pressure value:

$$\mathbb{A}_{ii} = \frac{\partial}{\partial p_i} \left(\mathbb{A}\vec{\boldsymbol{p}}\right)_i \tag{3.42}$$

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 $(\mathbb{A}\vec{\boldsymbol{p}})_i$ 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 $\vec{\boldsymbol{a}}_i^p$ within it is first prepared by factoring out all occurrences of p_i:

$$\vec{\boldsymbol{a}}_i^p = -\sum_{j\in\mathcal{N}_i} m_j \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2}\right)\nabla W_{ij} - \sum_{k\in\mathcal{B}_i} m_k \frac{p_i}{\rho_i^2}\nabla W_{ik} \qquad \textit{Equation 4.8} \tag{3.43}$$

$$= -p_i \frac{1}{\rho_i^2}\left(+\sum_{j\in\mathcal{N}_i} m_j \nabla W_{ij} + \sum_{k\in\mathcal{B}_i} m_k \nabla W_{ik}\right) - \sum_{j\in\mathcal{N}_i} m_j \frac{p_j}{\rho_j^2}\nabla W_{ij} \qquad \textit{factor out } p_i, {}^1\!/_{\rho_i^2} \tag{3.44}$$

$$= -p_i \frac{1}{\rho_i^2}\left(\vec{\boldsymbol{g}}_{ij} + \vec{\boldsymbol{g}}_{ik}\right) - \sum_{j\in\mathcal{N}_i\setminus\{i\}} m_j \frac{p_j}{\rho_j^2}\nabla W_{ij} \qquad \textit{use definitions Equation 3.46} \tag{3.45}$$

where for brevity, the following definitions were used:

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

$$\vec{\boldsymbol{g}}_{ik} = \sum_{k\in\mathcal{B}_i} m_k \nabla W_{ik} \tag{3.47}$$

Note that since the kernel function W is differentiable and spherically symmetric (Equation 2.19, Equation 2.21), it follows that $\nabla W_{ii} = \vec{\boldsymbol{0}}$, 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 $\frac{\partial}{\partial p_i}\vec{\boldsymbol{g}}_{ij} = \frac{\partial}{\partial p_i}\vec{\boldsymbol{g}}_{ik} = 0$.

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

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

(3.48)

$$= \Delta t^2 \sum_{j\in\mathcal{N}_i} m_j \vec{\boldsymbol{a}}^p_i \cdot \nabla W_{ij} - \Delta t^2 \sum_{j\in\mathcal{N}_i} m_j \vec{\boldsymbol{a}}^p_j \cdot \nabla W_{ij} + \Delta t^2 \sum_{k\in\mathcal{B}_i} m_k \vec{\boldsymbol{a}}^p_i \cdot \nabla W_{ik} \qquad \textit{split } \vec{\boldsymbol{a}}^p_{ij}$$

(3.49)

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

(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:

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

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

$$\frac{\partial}{\partial p_i} a_i = \frac{\partial}{\partial p_i} \sum_{j\in\mathcal{N}_i\setminus\{i\}} m_j \vec{\boldsymbol{a}}^p_i \cdot \nabla W_{ij} \qquad \textit{Equation 3.50}$$

$$= \frac{\partial}{\partial p_i} \sum_{j\in\mathcal{N}_i\setminus\{i\}} m_j \left(-p_i \frac{1}{\rho_i^2} (\vec{\boldsymbol{g}}_{ij} + \vec{\boldsymbol{g}}_{ik}) - \underbrace{\cancel{\sum_{j\in\mathcal{N}_i\setminus\{i\}} m_j \frac{p_j}{\rho_j^2} \nabla W_{ij}}}_{\textit{no } p_i\textit{, no contribution}} \right) \cdot \nabla W_{ij} \qquad \textit{insert Equation 3.45}$$

$$= -\frac{1}{\rho_i^2} \sum_{j\in\mathcal{N}_i\setminus\{i\}} m_j (\vec{\boldsymbol{g}}_{ij} + \vec{\boldsymbol{g}}_{ik}) \cdot \nabla W_{ij} \qquad \textit{apply partial derivative}$$

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

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

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:

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

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

$$\frac{\partial}{\partial p_i} b_i = \frac{\partial}{\partial p_i} - \sum_{j\in\mathcal{N}_i\setminus\{i\}} m_j \vec{\boldsymbol{a}}^p_j \cdot \nabla W_{ij} \qquad \textit{Equation 3.50}$$

$$= -\frac{\partial}{\partial p_i} \sum_{j\in\mathcal{N}_i\setminus\{i\}} m_j \underbrace{\left(\underbrace{\cancel{-p_j \frac{1}{\rho_j^2} (\vec{\boldsymbol{g}}_{jl} + \vec{\boldsymbol{g}}_{jk})}}_{\textit{no } p_i\textit{, no contribution}} - \sum_{l\in\mathcal{N}_j\setminus\{j\}} m_l \frac{p_l}{\rho_l^2} \nabla W_{jl} \right)}_{=\vec{\boldsymbol{a}}^p_j} \cdot \nabla W_{ij} \qquad \textit{insert Equation 3.45}$$

Since the acceleration $\vec{\boldsymbol{a}}_j^p$ 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:

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

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, 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:

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

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

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

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

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:

$$\mathbb{A}_{ii} = \Delta t^2 \left(\frac{\partial}{\partial p_i} a_i + \frac{\partial}{\partial p_i} b_i + \frac{\partial}{\partial p_i} c_i \right) \qquad \text{Equation 3.51}$$

$$= \Delta t^2 \left(\underbrace{-\frac{1}{\rho_i^2} (\vec{\boldsymbol{g}}_{ij} + \vec{\boldsymbol{g}}_{ik}) \cdot \vec{\boldsymbol{g}}_{ij}}_{\frac{\partial}{\partial p_i} a_i} \underbrace{-\frac{1}{\rho_i^2} (\vec{\boldsymbol{g}}_{ij} + \vec{\boldsymbol{g}}_{ik}) \cdot \vec{\boldsymbol{g}}_{ik}}_{\frac{\partial}{\partial p_i} c_i} \underbrace{-\frac{m_i}{\rho_i^2} \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_j \left| \nabla W_{ij} \right|^2}_{\frac{\partial}{\partial p_i} b_i} \right) \qquad \text{reinsert}$$

$$= -\frac{\Delta t^2}{\rho_i^2} \left(\underbrace{(\vec{\boldsymbol{g}}_{ij} + \vec{\boldsymbol{g}}_{ik}) \cdot \vec{\boldsymbol{g}}_{ij} + (\vec{\boldsymbol{g}}_{ij} + \vec{\boldsymbol{g}}_{ik}) \cdot \vec{\boldsymbol{g}}_{ik}}_{\text{binomial}} + \sum_{j \in \mathcal{N}_i \setminus \{i\}} m_i m_j \left| \nabla W_{ij} \right|^2 \right) \qquad \text{factor out } -\frac{1}{\rho_i^2}$$

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

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

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

Diagonal Element $\mathbb{A}_{ii}$

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

The diagonal element depends only on the current positions, neighbour sets, masses and 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 density has been computed and then reused in each iteration of the iterative solver described in the following.

The diagonal element derived here is notably similar to the optimal stiffness coefficient κ derived in [16, 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 [16]:

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

and another [26, 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 [27, 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 $\vec{\boldsymbol{a}}_i^p$ in Equation 4.8 and the right-hand side of the PPE $(\mathbb{A}\vec{\boldsymbol{p}})_i$ 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:

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

until the predicted density error per particle $\left(\rho_i^{err}\right)^l = (\mathbb{A}\vec{\boldsymbol{p}})_i - s_i$ 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 [25] and ensures that pressure values are always positive. This prevents undesirable clumping artefacts that could otherwise occur when using an SPH discretization [14] 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) [3]. 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 [3].

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 [14].

For the initial iteration $l = 0$, the authors of the [14, original paper] recommend warm-starting the solver by initializing all pressure values $\vec{\boldsymbol{p}}$ with half the pressure computed in the previous time step, i.e.:

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

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 $p_i = 0$, $(\mathbb{A}\vec{\boldsymbol{p}}_i) = 0$ into Equation 3.62 and obtaining:

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

effectively skipping the first solver iteration by setting the pressure to the result thereof.

The relaxed Jacobi solver as presented in Equation 3.62 can suffer in practice from divergence and in consequence 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 $\left(\rho_i^{err}\right)^l = (\mathbb{A}\vec{\boldsymbol{p}})_i - s_i$ 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 [3, 14].

Two predicted error metrics are commonly employed [14]:

$$\rho^l_{max} = \frac{1}{\rho_0} \max\left(\{\forall i \in \mathcal{N}_1^N : (\rho_i^{err})^l\}\right) \qquad \textit{Maximum predicted density error} \tag{3.65}$$

$$\rho^l_{avg} = \frac{1}{\rho_0} \frac{1}{N} \sum_{i=1}^{N} (\rho_i^{err})^l \qquad \textit{Average predicted density error} \tag{3.66}$$

The [14, 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^l_{max} and ρ^l_{avg} 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 [14].

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^l_{max} 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:

$$\left(\rho^l_{avg} \leq \rho^{thresh}_{avg}\right) \wedge \left(\rho^l_{max} \leq \rho^{thresh}_{max}\right) \wedge \left(l_{min} \leq l \leq l_{max}\right) \tag{3.67}$$

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 $\rho^{thresh}_{avg} = 0.1\%$ [14] and $\rho^{thresh}_{max} = 0.5\%$.

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 $\left|\rho^l_{avg}\right| \leq \rho^{thresh}_{avg}$ 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

Step 1 – Preliminary Computations

Find neighbour sets $\mathcal{N}_i, \mathcal{B}_i$ ▷ Fixed Radius Neighbour Search (Appendix A)
Compute densities ρ_i ▷ Equation 4.7
Compute non-pressure accelerations $\vec{\boldsymbol{a}}_i^{\nu}, \vec{\boldsymbol{g}}$ ▷ Equation 3.2
Determine time step size Δt ▷ Equation 3.7 or subsection 3.4.1
Compute predicted velocities $\vec{\boldsymbol{v}}_i^*$ ▷ Operator splitting, Equation 3.11

Step 2 – IISPH - Precomputations

Compute diagonal element $\mathbb{A}_{ii}$ ▷ Equation 3.60
Compute source term s_i ▷ Equation 3.35
Reset pressure values or warm-start p_i ▷ Equation 3.63 or Equation 3.64

Step 3 – IISPH Solver Iteration

for iteration $l = 0$ until convergence **do** ▷ convergence critertion Equation 3.67
    Compute pressure acceleration $\vec{\boldsymbol{a}}_i^{p,l}$ ▷ Equation 4.8
    Compute predicted density error due to pressure accelerations $(\mathbb{A}\vec{\boldsymbol{p}})_i^l$ ▷ Equation 3.39
    Update pressure using relaxed Jacobi iteration ▷ Equation 3.62
end for

Step 4 – Numerical Time Integration

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

Step 5 – Update Rigid Bodies and Dynamic Boundary Particles

Use final $\vec{\boldsymbol{a}}_i^p$ to compute $\vec{\boldsymbol{f}}_i, \vec{\boldsymbol{\tau}}_i$ on rigid bodies
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 3.4.1
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

A particularly important and oft-discussed degree of freedom 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). 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 ($\nabla \cdot \vec{\boldsymbol{v}} = 0$). 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 [16]. 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 [16]) and the quality of the particle sampling can degrade over time, rendering the SPH approximations increasingly inaccurate [23]. 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 preserves the quality and regularity of the particle sampling and does not suffer from volume loss [23].

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 [23, more optimized source term formulations] and, very similarly, the [16, 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 $\nabla \vec{\boldsymbol{v}}$ 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 $\nabla p_i = \kappa_i \nabla \rho_i$ [16] 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 [26, essentially equivalent] to the relaxed Jacobi iteration used by IISPH. Crucially, two such coefficients are calculated:

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

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

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 [16, DFSPH paper] do not include boundary handling, proper clamping of values and other variations that can be critical for stability and are present in the [28, authors' own implementation]. Since the density invariance solve of DFSPH is not just equivalent to IISPH but the main author of the DFSPH paper[16] [27, in a later paper] *defined* DFSPH to be a relaxed Jacobi solver as outlined in this report using the 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:

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

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

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 $\vec{\boldsymbol{f}}_i$ 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 [16, 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
Find neighbour sets $\mathcal{N}_i, \mathcal{B}_i$ ▷ Fixed Radius Neighbour Search (Appendix A)
Compute densities ρ_i ▷ Equation 4.7
Compute non-pressure accelerations $\vec{\boldsymbol{a}}_i^{\nu}, \vec{\boldsymbol{g}}$ ▷ Equation 3.2
Determine time step size Δt ▷ Equation 3.7 or subsection 3.4.1
Compute predicted velocities $\vec{\boldsymbol{v}}_i^*$ ▷ Operator splitting, Equation 3.11
Compute diagonal element $\mathbb{A}_{ii}$ ▷ Equation 3.60
Compute velocity divergence source term s_i^v ▷ Equation 3.70
Reset pressure values or warm-start p_i ▷ Equation 3.63 or Equation 3.64

Step 2 – Velocity Divergence Solver
procedure RelaxedJacobiSolver(s_i)
  for iteration $l = 0$ until convergence **do**
    Compute pressure acceleration $\vec{\boldsymbol{a}}_i^{p,l}$ ▷ Equation 4.8
    Compute predicted density error due to pressure accelerations $(\mathbb{A}\vec{\boldsymbol{p}})_i^l$ ▷ Equation 3.39
    Update pressure using relaxed Jacobi iteration ▷ Equation 3.62
  end for
end procedure
RelaxedJacobiSolver(s_i^v) ▷ s_i^v from Equation 3.70, use a fixed iteration count
$\vec{\boldsymbol{v}}_i^* \leftarrow \vec{\boldsymbol{v}}_i^* + \Delta t \vec{\boldsymbol{a}}_i^p$ ▷ Update predicted velocities
Accumulate the used $\vec{\boldsymbol{a}}_i^p$ to compute $\vec{\boldsymbol{f}}_i$ on rigid bodies

Step 3 – Density Invariance Solver
Compute density invariance source term s_i ▷ Equation 3.35
RelaxedJacobiSolver(s_i) ▷ s_i from Equation 3.35, convergence critertion Equation 3.67

Step 4 – Numerical Time Integration
$\vec{\boldsymbol{v}}(t + \Delta t) \leftarrow \vec{\boldsymbol{v}}_i^* + \Delta t \vec{\boldsymbol{a}}_i^p$
$\vec{\boldsymbol{x}}(t + \Delta t) \leftarrow \vec{\boldsymbol{x}}_i + \Delta t \vec{\boldsymbol{v}}(t + \Delta t)$ ▷ like Equation 3.5, respecting operator splitting

Step 5 – Update Rigid Bodies and Dynamic Boundary Particles
Use further accumulated $\vec{\boldsymbol{a}}_i^p$ to compute $\vec{\boldsymbol{f}}_i, \vec{\boldsymbol{\tau}}_i$ on rigid bodies
Update rigid body state and corresponding boundary particles ▷ Algorithm 5

3.4.1 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-Lewy 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 $\max_i |\vec{\boldsymbol{v}}_i|$ 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 disproportionally 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:

$$P = \frac{T_{sim}}{T_{comp}} \tag{3.72}$$

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 P much less than one 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:

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

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

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

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

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

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

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 [29, online learning for black box optimization] or controller tuning and applying a form of [30, Time-Varying Bayesian Optimization] or similar [31, 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 $\sigma_{\mathcal{P}}^2 = 10^{-6}\,\mathrm{sec}$ that is orders of magnitude smaller than the the expected time step size:

Proposal Distribution

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

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

for every time step k **do**
  Draw a proposed time step size $\Delta t'_{k+1} \sim \mathcal{P}$ from proposal distribution $\mathcal{P}$
  Ensure $\Delta t'_{k+1} \leq \Delta t_{CFL}$ by clamping ▷ Equation 3.7
  Run simulation step and measure performance $\hat{P}\left(\Delta t'_{k+1}\right)$ of the proposal
  if $\hat{P}\left(\Delta t'_{k+1}\right) > \hat{P}\left(\Delta t_k\right)$ **then**
    $\Delta t_{k+1} \leftarrow \Delta t'_{k+1}$ ▷ accept proposal
  else
    $\Delta t_{k+1} \leftarrow \Delta t_k$ ▷ reject proposal
  end if
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 [29], 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 [32].

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 [32]:

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

where T_k is a control parameter, also called 'temperature' in the context of simulated annealing [32] 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 [32]. 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 [32], which eases the temperature towards zero, is used. The same can not be 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 [32]:

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

where a and r are the number of improving (i.e. $\hat{P}_k \leq \hat{P}'_{k+1}$) and non-improving proposals (i.e. $\hat{P}_k > \hat{P}'_{k+1}$) respectively and $\Delta \hat{P}^- = \frac{1}{r} \sum_{k=0}^{K} \left(\hat{P}_k - \hat{P}'_{k+1}\right)_+$ is the average decrease in performance score of

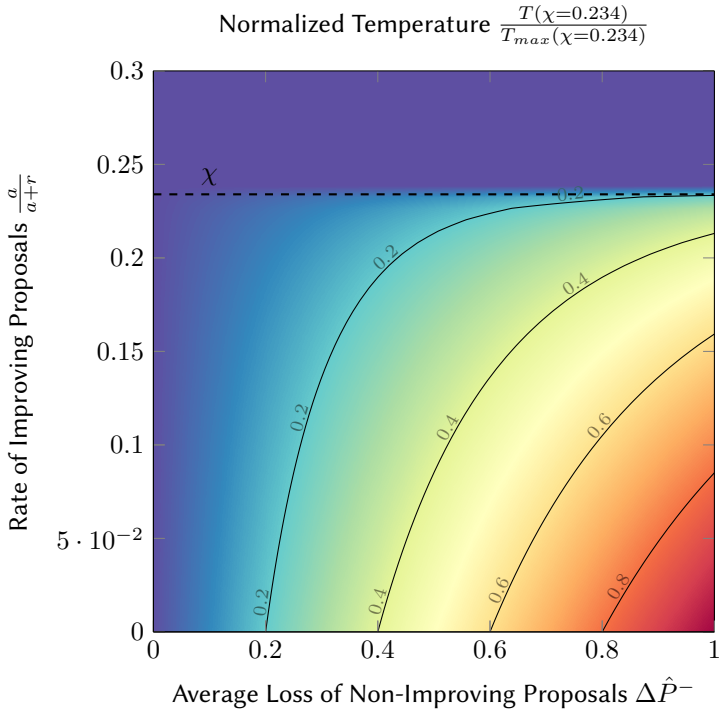


Figure 3.1: The temperature $T(\chi = 0.234)$, normalized from $[0; T_{max}]$ to $[0; 1]$ is plotted for differing rates of improving proposals z and average loss of non-improving proposals $\Delta\hat{P}^-$. Note that the function is scale-invariant in $\Delta\hat{P}^-$, i.e. if the temperature is normalized 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.

non-improving transitions. This can be rearranged to yield a formula for the temperature c in terms of a target acceptance rate χ [32]:

Temperature Function

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

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 $\chi = 0.234$ [33, 34]. Although the applicability of this *optimal scaling* to the specific problem at hand has admittedly not been thoroughly theoretically 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 [32], 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 $1\,\mathrm{sec} \gg \Delta t$ and a time horizon of e.g. $t_{hrz} = 1\,\mathrm{sec}$ 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 so-called episodes $E = \{k, k+1, \ldots, k+K_E-1\}$, using $K_E = 5$ time steps each in this instance. 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:

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

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:

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

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 on time step sizes close to the optimum obtained by hand-tuning parameters for performance.

Algorithm 4 Random Walk Metropolis Controller for Δt

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

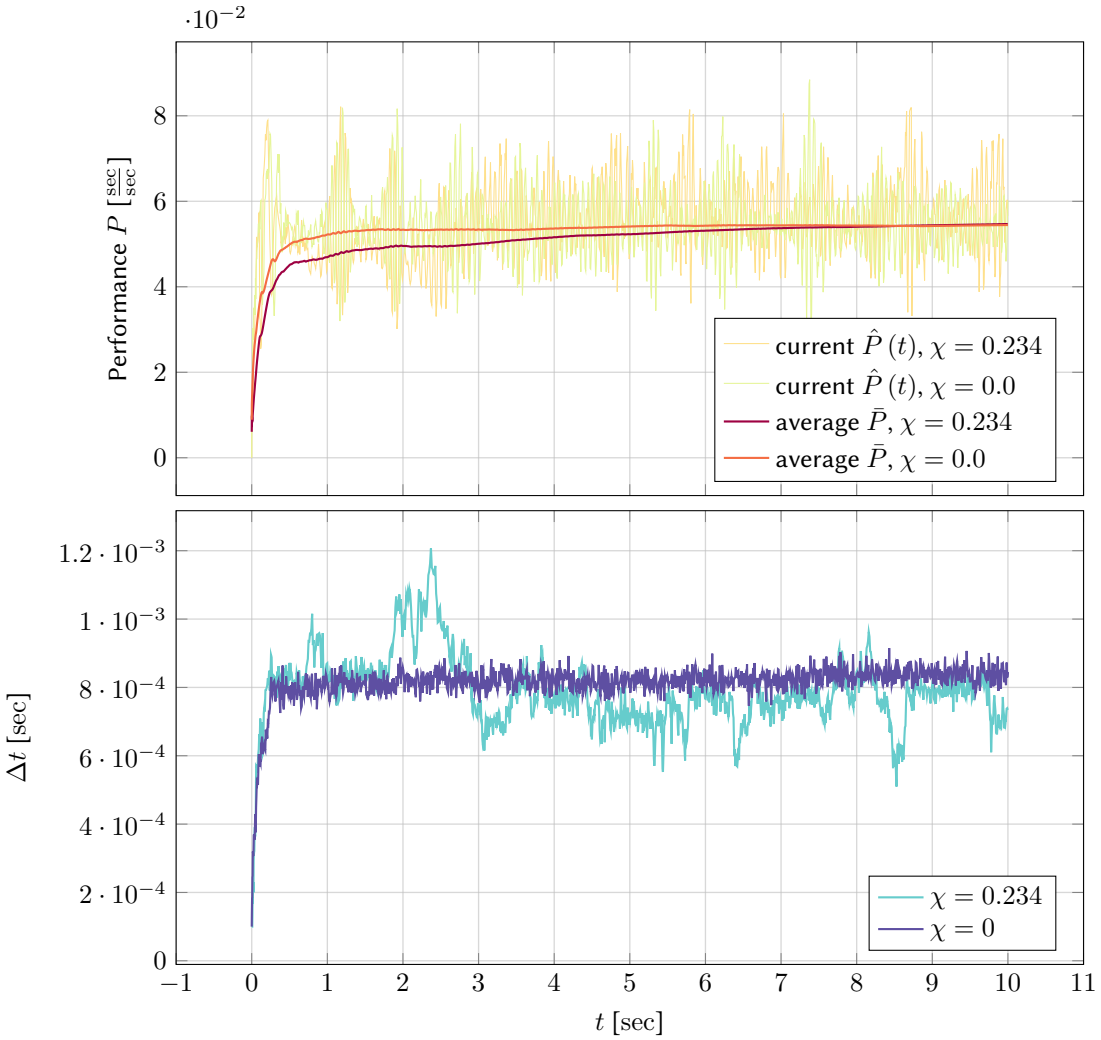


Figure 3.2: A $2m \times 2m \times 5m$ static water column of $302K$ fluid and $217K$ boundary particles is simulated for 10 sec of simulated time using automatic time step selection with $\sigma^2 = 5 \cdot 10^{-6}$ sec, $K_E = 5$ as described in Algorithm 4 using the IISPH fluid solver (subsubsection 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 plateaus, its maximum 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.

Chapter 4

Weakly 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 is solved [3]. A versatile method for discretizing boundary objects in a manner consistent with the SPH discretization of the fluid is discussed in section 4.1 [35]. While this already realizes one-way coupling of the fluid to a static boundary, a rigid body solver has to be developed and two-way coupled to the fluid solver to achieve interacting rigids 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. Note that while fluids and rigids then interact, more challenging interactions such as rigid-rigid contacts with accurate friction that are an active area of research at time of writing [25] are not handled in this report.

4.1 Uniform and Robust Boundary Representation

A common idea in modelling boundary interactions in SPH 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, the quantities m_k, $\vec{\boldsymbol{x}}_k$ and $\vec{\boldsymbol{v}}_k$ must be known for all boundary particles k. While $\vec{\boldsymbol{v}}_k$ 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 such a boundary: the size of the boundary particles, which equates to their mass m_k, and the sampling of the boundary domain, which yields the $\vec{\boldsymbol{x}}_k$.

4.1.1 Sampling and Initialization

As will be outlined in the following, a single layer of non-uniformly distributed boundary particles on the surface of the boundary are sufficient to ensure accurate fluid-boundary interactions. This is helpful, since thin shells and complex triangular meshes should be able to be represented, 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.

Firstly, an input triangular mesh representing an arbitrary boundary geometry is approximated by a **watertight manifold** with vertices that are roughly uniformly distributed across the surface using the method outlined by [36, 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 [36].

The resulting watertight manifold, while retaining the shape of the geometry as well as possible, amongst other desirable properties enables the notion of distance of points across the mesh to be sensible. Since the mesh is now a manifold, the distance between two points that are not on the same triangle can be calculated by taking the mesh topology into account, since only exactly two triangles in the mesh 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 points are at least some radius r according to some distance metric apart by, for example [37], 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 [37]. An example of the blue noise distribution of boundary particles generated on a mesh 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 [38].

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}, which for the purpose of handling contact forces as pressure forces using the fluid solver can be assumed to be of the rest density ρ_0 of the fluid under consideration. Note that in this instance ρ_0 is fixed, for multiphase flows it can have differing values and in any case it is distinct from 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 [39]:

$$\rho_i = \int_{\Omega_{fl}} W_{ij}\, dm_j + \int_{\Omega_{bdy}} W_{ik}\, dm_j \tag{4.1}$$

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

where $\mathcal{B}$ is defined analogously to $\mathcal{N}$ but for boundary samples that neighbour particle i and 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 interface 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 $m_k = \rho_0 - \frac{\rho_0 \cdot |\vec{\boldsymbol{x}}_{ik}|}{\hbar}$ using for example signed distance fields, storing the results on a grid and interpolating them as proposed by [39, Koshier and Bender]. Instead, for the small kernel support radius $\hbar = 2h$ in the expected fluid particle spacing h, a constant approximation that is not distance-dependent can be deemed sufficiently accurate and more efficient, yielding the method of [35, 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 Equation 2.14 implying that $\sum_{l \in \mathcal{B}_k} V_l W_{kl} \stackrel{!}{=} 1$ ought to hold at any boundary particle k [35]. This means that the kernel sum effectively measures the inverse of the volume of a particle given a sampling: if not 1 but a higher value is obtained, all V_l ought to be correspondingly lower and vice versa, which means [35]:

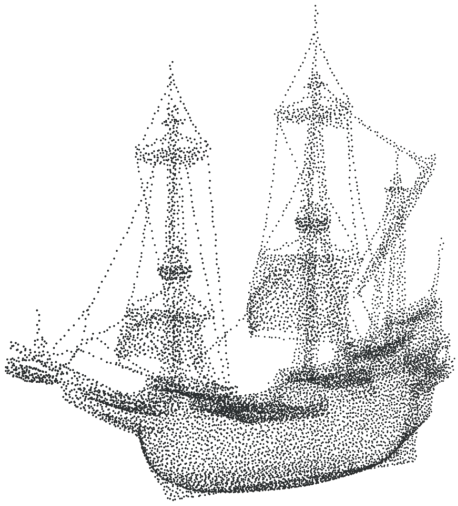
$$V_k = \frac{m_k}{\rho_0} \stackrel{!}{=} \frac{m_k}{\sum_{l \in \mathcal{B}_k} m_l W_{kl}} \approx \gamma_1 \frac{m_k}{m_k \sum_{l \in \mathcal{B}_k} W_{kl}} \tag{4.3}$$

$$\implies m_k = \gamma_1 \frac{\rho_0}{\sum_{l \in \mathcal{B}_k} W_{kl}} \tag{4.4}$$

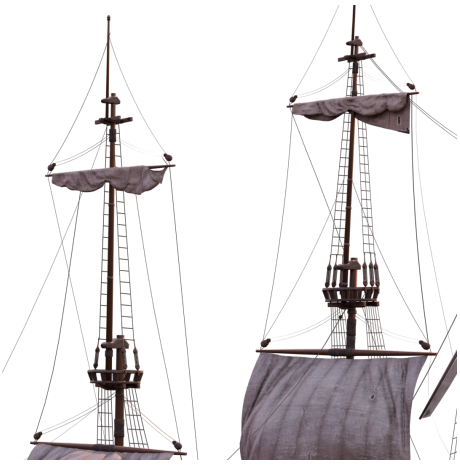
where the parameter γ_1 in the approximation can be used to adjust the density contribution per boundary particle for differing kernel support sizes, choosing for example a value that ensures rest density for a regular sampling of a plane, however $\gamma_1 = 1$ is chosen in this instance. Since the effective mass is assumed to be independent of distance to the boundary, m_k can be precomputed once and stored for each boundary particle.

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

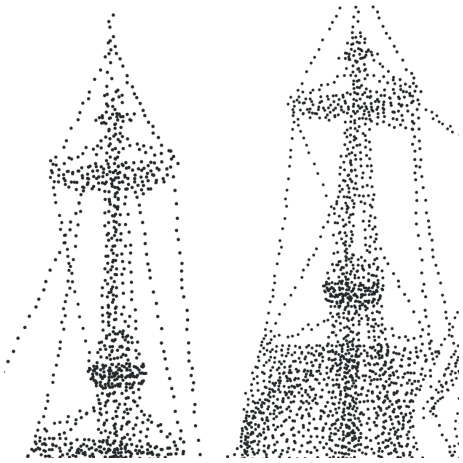
(a) Rendering of the mesh



(b) Sampling of the mesh



(c) Close-up view of sails and ropes



(d) Close-up view of sampled sails and ropes

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 [38] 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.

for initializing fluid particles with rest density in Equation 3.10:

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

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

In the implementation, the exact 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 by [35, 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 where neglected, since this limitation greatly simplifies the setting: given two equations $\vec{\boldsymbol{x}}_{cm}(t), \vec{\boldsymbol{v}}_{cm}(t)$, all boundary particles belonging to the same scripted object are shifted from their initial position by exactly $\vec{\boldsymbol{x}}_{cm}(t)$ and all have the same velocity $\vec{\boldsymbol{v}}_{cm}(t)$ at any point t in time.

The previously discussed boundary handling can thankfully naturally handle intersecting boundaries. For example, a scripted boundary plane moving sinusoidally to produce waves may intersect a container of water. However, the masses of involved boundary particles might need to be recalculated so that the pressure forces exerted on the fluid by the boundary does not change where boundaries intersect. This is done according to two rules, which in conjunction conservatively estimate the set of boundary particles that need their masses recalculated according to Equation 4.4 in each time step:

1. All masses of boundary particles belonging to scripted 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 the counter decremented.

This means that not only are the masses of static boundary samples re-evaluated when scripted boundaries intersect them to ensure all boundary particles correctly exert forces on fluids as if they were the same fluid at rest density at all times, but the masses are also re-evaluated once more when a scripted boundary particle has just left the kernel support radius, ensuring correct masses as before.

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 [40, 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:

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

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 [35, 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 $p_i = p_k$ [35], solving for pressures at boundary particles in the pressure solver explicitly to avoid inconsistent values of p_k for different fluid particles i [41], extrapolating the pressure field into the boundary domain using some scheme like *Moving Least Squares* [42], or simply setting the pressure at boundary particles to zero [27]. This choice can have a profound impact on pressure solver convergence and the quality of the resulting pressure field [27]. In accordance with recent literature by [27, 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 with a weighting of boundary pressure forces by a coefficient $\gamma_2 = 0.5$ [3]. Using $p_k = 0$, some terms in Equation 3.3 drop out, leaving the expression:

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

This expression is used to derive the diagonal element in the IISPH solver of subsubsection 3.3 using the additional assertion that $\rho_i \overset{!}{=} \rho_j \overset{!}{=} \rho_0$. Since this discretization of $\vec{\boldsymbol{a}}_i^p$ 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 [35]:

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

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

4.2 Two-Way Coupled Rigid Body Solver

Assuming the mass properties of rigid bodies outlined in subsection 4.2.1 are known, the dynamics of unconstrained rigid bodies can be discussed in order to arrive at a solver which can be weakly coupled to the fluid solver from 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 solver in subsubsection 3.3 resolve penetrations 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 [43], 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 [43]. This is conveniently done by using the centre of mass $\vec{\boldsymbol{x}}_{cm}$ as the origin which defines *body space*. In this coordinate system, any point $\vec{\boldsymbol{x}}'(t) := \vec{\boldsymbol{x}}(t) - \vec{\boldsymbol{x}}_{cm}$ belonging to the rigid body can be described using the equation of motion [44]:

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

where $\mathbb{O}$ encodes the current rotation about the centre of mass in a rotation matrix and $\vec{\boldsymbol{x}}_{cm}(t)$ 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 $\vec{\boldsymbol{f}}_{rigid} = \frac{d}{dt} M \vec{\boldsymbol{v}}_{cm}$ and constant mass M it holds that all forces $\vec{\boldsymbol{f}}_i$ 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 [44]:

$$\vec{\boldsymbol{f}}_{rigid} = \vec{\boldsymbol{f}}_g + \sum_i \vec{\boldsymbol{f}}_i \tag{4.11}$$

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 $\vec{\boldsymbol{a}}_{cm} = \frac{1}{M} \vec{\boldsymbol{f}}_{rigid}$ of the centre of mass $\vec{\boldsymbol{x}}_{cm}$ and integrated with respect to time as in Equation 3.5.

Meanwhile, the rate of change of the angular momentum $\frac{d}{dt} \vec{\boldsymbol{L}} = \vec{\boldsymbol{\tau}}_{rigid}$ defines the net torque $\vec{\boldsymbol{\tau}}_{rigid}$, which can be computed as [44]:

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

where the position $\vec{\boldsymbol{x}}_i$ in world space at which the force $\vec{\boldsymbol{f}}_i$ acts upon the rigid body is relevant. Intuitively, the vector $\vec{\boldsymbol{\tau}}_i$ can be understood as pointing in the axis of the rotation that a force $\vec{\boldsymbol{f}}_i$ would cause if the rigid body was fixed at its centre of mass and left to spin [44].

Updating Rigid Body State

Using the total input force $\vec{\boldsymbol{f}}_{rigid}$ and torque $\vec{\boldsymbol{\tau}}_{rigid}$, 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 $\vec{\boldsymbol{x}}_{cm}(t + \Delta t)$ 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 $\vec{\boldsymbol{v}}_{cm}$ and angular velocity $\vec{\boldsymbol{\omega}}$ are required and are integrated with respect to time.

The linear velocity and updated centre of mass are straightforward and can be obtained using for example semi-implicit Euler time integration (Equation 3.5) from the linear acceleration described above as:

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

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

The angular velocity $\vec{\boldsymbol{\omega}}$ points in the axis of rotation with a magnitude that encodes the speed of the revolution in radians per second [44]. 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 [43]:

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

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

where the angular momentum can be found by numerically integrating the torque with respect to time, since $\vec{\boldsymbol{\tau}} = \frac{d}{dt} \vec{\boldsymbol{L}}$ [44]:

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

The inertia tensor (or its inverse) at some point in time t can be determined as [44]:

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

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

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 is its inverse.

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

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

where $\dot{\mathbb{O}}(t)$ or the rate of change of the orientation matrix is the skew-symmetrix matrix that encodes the angular velocity $\vec{\boldsymbol{\omega}}(t) = (\omega_x, \omega_y, \omega_z)^T$, namely [44]:

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

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

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

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

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 not guaranteed to be a rotation matrix any more. Numerical drift can lead to accumulating errors, potentially leading to an unphysical scaling or skew of the rigid body [44], indicated by the determinant $\det(\mathbb{O})$ becoming different from one. To prevent this, $\mathbb{O}$ could be re-orthonormalized after every so often, using for example the Gram-Schmidt process. However, this would cause the error to be projected onto arbitrarily chosen axes that depend on the order of the basis vectors considered in the process. A more elegant approach is choosing a different representation of orientations that can alleviate this issue.

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 $\vec{\boldsymbol{q}} = (s, x, y, z)^T$ in four dimensions or a pair $[s, \vec{\boldsymbol{v}}]$ of a scalar s and a vector $\vec{\boldsymbol{v}} = (x, y, z)^T$ holding its coefficients [44]. Using the second definition, quaternion multiplication can be written as [44]:

$$\vec{\boldsymbol{q}}_1 \vec{\boldsymbol{q}}_2 = [s_1, \vec{\boldsymbol{v}}_1]\,[s_2, \vec{\boldsymbol{v}}_2] \tag{4.24}$$

$$= [s_1 s_2 - \vec{\boldsymbol{v}}_1 \cdot \vec{\boldsymbol{v}}_2, s_1 \vec{\boldsymbol{v}}_2 + s_2 \vec{\boldsymbol{v}}_1 + \vec{\boldsymbol{v}}_1 \times \vec{\boldsymbol{v}}_2] \tag{4.25}$$

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 [44], making them especially convenient in this context. It turns out that the rate of change of the orientation $\dot{\vec{\boldsymbol{q}}}$ can be expressed in terms of the angular velocity $\vec{\boldsymbol{\omega}}$ using quaternion multiplication as:

$$\dot{\vec{\boldsymbol{q}}} = \frac{1}{2}\,[0, \vec{\boldsymbol{\omega}}]\,[s, \vec{\boldsymbol{v}}] \tag{4.26}$$

This can be explicitly numerically integrated:

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

and then normalized like any four-dimensional vector, so it remains a representation of a rotation. Finally, the resulting unit quaternion $\hat{\vec{\boldsymbol{q}}}(t + \Delta t) = (s, x, y, z)^T$ can be converted to a rotation matrix $\mathbb{O}(t + \Delta t)$ using the identity [44]:

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

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

Updating the Dynamic Boundary Particles

Once a new orientation $\mathbb{O}(t + \Delta t)$ and position of the centre of mass $\vec{\boldsymbol{x}}_{cm}(t + \Delta t)$ has been obtained, all boundary particles sampled on the surface of the rigid body can be moved according to [44]:

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

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

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

using the initial positions at $t = 0$, where $\vec{r}_i$ 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

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

$\vec{\boldsymbol{x}}_{cm} \leftarrow \vec{\boldsymbol{x}}_{cm} + \Delta t \vec{\boldsymbol{v}}_{cm}$ ▷ Equation 4.14

Step 2 – Update Orientation

$\dot{\vec{\boldsymbol{q}}} \leftarrow \frac{1}{2} [0, \vec{\boldsymbol{\omega}}] \, \vec{\boldsymbol{q}}$ ▷ Equation 4.26 using quaternion multiplication

$\vec{\boldsymbol{q}} \leftarrow \frac{\vec{\boldsymbol{q}} + \Delta t \dot{\vec{\boldsymbol{q}}}}{|\vec{\boldsymbol{q}} + \Delta t \dot{\vec{\boldsymbol{q}}}|}$ ▷ Equation 4.27 and normalize quaternion

Calculate $\mathbb{O}$ from $\vec{\boldsymbol{q}}$ ▷ Equation 4.28

Step 3 – Compute Rotational Movement

$\vec{\boldsymbol{L}} \leftarrow \vec{\boldsymbol{L}} + \Delta t \vec{\boldsymbol{\tau}}_{rigid}$ ▷ using $\vec{\boldsymbol{\tau}}_{rigid}$ from Equation 4.12

$\vec{\boldsymbol{\omega}} \leftarrow \mathbb{O} \mathbb{I}^{-1} \left(t = 0 \right) \mathbb{O}^T \vec{\boldsymbol{L}}$ ▷ Equation 4.22

Step 4 – Update Particles

for $i \in \mathbb{N}_0^{N_{rigid}-1}$ in parallel **do**

  $\vec{\boldsymbol{r}}_{i,0} = \left(\vec{\boldsymbol{x}}_i \left(t = 0 \right) - \vec{\boldsymbol{x}}_{cm} \left(t = 0 \right) \right)$ ▷ Initial body space position, auxiliary variable

  $\vec{\boldsymbol{x}}_i \leftarrow \mathbb{O} \vec{\boldsymbol{r}}_{i,0} + \vec{\boldsymbol{x}}_{cm}$ ▷ Equation 4.30

  $\vec{\boldsymbol{v}}_i \leftarrow \vec{\boldsymbol{v}}_{cm} + \vec{\boldsymbol{\omega}} \times \left(\mathbb{O} \vec{\boldsymbol{r}}_{i,0} \right)$ ▷ Equation 4.31

end for

4.2.1 Exact Mass Moments of Watertight Meshes

In order to implement the rigid body solver in section 4.2, prerequisite include knowing the total mass M, centre of mass $\vec{\boldsymbol{x}}_{cm}$, volume V and the inertia tensor $\mathbb{I}$ of the rigid body being simulated. 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 $M, \mathbb{I}, \vec{\boldsymbol{x}}_{cm}$ etc. of the closed triangular mesh of $\partial\Omega_{rigid}$ must be computed, in this case using a method by [45, Zhang and Chen], which is outlined in the following. The mass density moments of the Ω_{rigid} are defined as an integral [45]:

$$\mathcal{M}_{pqr} := \iiint_{\Omega_{rigid}} \rho x^p y^q z^r \, dxdydz \tag{4.32}$$

over the volume of the rigid body for some p, q, r, which means that 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.

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.

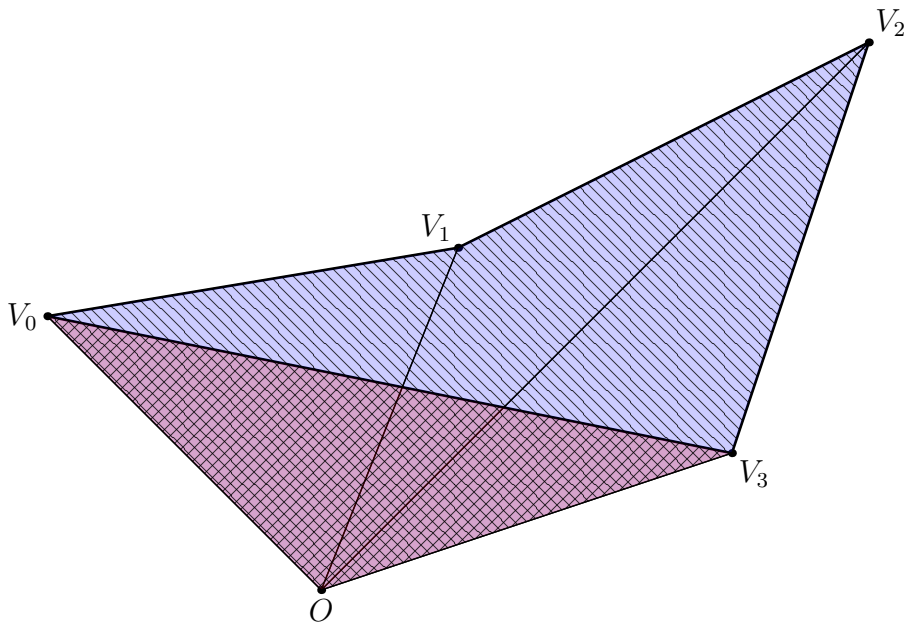


Figure 4.2: In a figure similar to the one by [45, Zhang and Chen], a volume integral using the described method of signed volumes is illustrated for a polygon with vertices V_0, V_1, V_2, V_3 and an additional vertex O at the origin. Whereas in three dimensions, volumes of tetrahedra are added, and 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.

More traditionally, the integration domain Ω_{rigid} could be partitioned into a set of tetrahedra $\{\triangleq_i\}$ such that $\Omega_{rigid} = \bigcup_i \triangleq_i$ and $\forall i, j : \triangleq_i \cap \triangleq_j = \emptyset$ so that the integral can be written as:

$$\int_{\Omega_{rigid}} dV = \sum_{\triangleq_i} V\left(\triangleq_i\right) \tag{4.33}$$

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

Instead, one fixed point O anywhere in space can be chosen to construct tetrahedra $\triangleq_{ABCO}$ by simply summing over all the triangles $\triangle_{ABC}$ 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 $V'\left(\triangleq_{ABCO}\right)$ 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 simlar 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 subsubsection 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 $\triangleq_{ABCO}$ can be defined as [45, 46]:

$$V'\left(\triangleq_{ABCO}\right) = \frac{1}{6}\det\left(J\left(\triangleq_{ABCO}\right)\right) \tag{4.34}$$

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

$$= \frac{1}{6}\left(\vec{\boldsymbol{A}} \times \vec{\boldsymbol{B}}\right) \cdot \vec{\boldsymbol{C}} \tag{4.36}$$

where J denotes the Jacobian of the tetrahedron and the position vector of each vertex $\overrightarrow{OA} = \vec{\boldsymbol{A}}$ 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. With this, the volume integral can be written as a sum [45]:

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

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, $\sum_{\triangle_{ABC}} V'\left(\triangleq_{ABCO}\right)$ being a positive quantity defines some function $\operatorname{sgn}\left(\triangleq_{ABCO}\right) \in \{-1, 1\}$, which assigns a positive or negative weight to any tetrahedron $\triangleq_{ABCO}$, 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 [45]:

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

where $\Omega\left(ABCO\right)$ is the domain of the tetrahedron $\triangleq_{ABCO}$, 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:

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

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:

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

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 $M'\left(\triangleq_{ABCO}\right)$ from Equation 4.39:

$$\vec{x}_{cm}\left(\triangleq_{ABCO}\right) = \frac{M'\left(\triangleq_{ABCO}\right)}{4} \left(\vec{A} + \vec{B} + \vec{C}\right) \tag{4.41}$$

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

$$\vec{x}_{cm} = \frac{1}{M} \sum_{\triangle_{ABC}} \vec{x}_{cm}\left(\triangleq_{ABCO}\right) \tag{4.42}$$

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* [43]. 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 [47, 43]:

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

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:

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

$$I_{xx} = \mathcal{M}_{200} \tag{4.45}$$

$$I_{yy} = \mathcal{M}_{020} \tag{4.46}$$

$$I_{zz} = \mathcal{M}_{002} \tag{4.47}$$

$$I_{xy} = \mathcal{M}_{100} + \mathcal{M}_{010} \tag{4.48}$$

$$I_{xz} = \mathcal{M}_{100} + \mathcal{M}_{001} \tag{4.49}$$

$$I_{yz} = \mathcal{M}_{010} + \mathcal{M}_{001} \tag{4.50}$$

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 $\triangle_{ABCO}$ are required, which in this instance have the form:

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

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

and analogously for the other two axes. These exact analytic expressions are cited from [46, 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 quantities are known and a simple rigid body solver as discussed in section 4.2 can be implemented.

Chapter 5

Visualization

A crucial factor in the application of fluid solvers based on SPH to Computer Graphics is the visualization of the simulation results. Even if the application of the simulation is focused on engineering and the resultant data, some form of visual inspection is invariably part of the process of fluid simulation.

While subsection 4.1.1 is concerned with the sampling of particles on a triangular mesh, visualization of the simulated scene requires a reconstruction of triangular meshes for rendering. 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 [48], all of which is outlined 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 observed in e.g. [49, 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 ($\Delta t \ll 16\mu\mathrm{s} = \Delta t_{frame}$ 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 t_{frame}. Other methods, such as interpolating the simulation results temporally to the exact point in time $i \in \mathbb{N} : i \cdot t_{frame}$ shown in the i-th 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 $\vec{\boldsymbol{x}}_{cm}(t)$ or an additional rotation about their centre of mass (which was not described 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 at the origin, not in the global coordinate system. Since transformation do not generally commute, this means that before the rotation encoded by the orientation matrix $\mathbb{O}$, or equivalently the quaternion $\vec{\boldsymbol{q}}$, 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 $\vec{\boldsymbol{x}}_{cm}(t=0)$ in global coordinates, one must:

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

to 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.

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 [50, 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 [51, Weighted Laplacian Smoothing for Surface Reconstruction] was used, the main concepts of which are outlined in the following.

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 field of the fluid volume and which is therefore closely related to the density field [51]:

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

Recall from section 2.2 that the SPH interpolation scheme is valid not only at particle positions $\vec{\boldsymbol{x}}_i$ but at any position $\vec{\boldsymbol{x}}$ in space, as observed in the definition of the colour field in Equation 5.1. This means that a common building block in surface reconstruction can be applied here, namely the **MARCHING CUBES ALGORITHM** [52]. 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 [52]. The corresponding triangle topology can therefore 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 conretely this means that the sign of a potential [51]:

$$\Phi(\vec{\boldsymbol{x}}) = \langle 1 \rangle (\vec{\boldsymbol{x}}) - t \qquad \textit{see Equation 5.1} \tag{5.2}$$

the zero-set of which is the surface, can be used to classify vertices as inside or outside ($\vec{\boldsymbol{x}}_{in}, \vec{\boldsymbol{x}}_{out}$). The surface vertex $\vec{\boldsymbol{x}}_s$ between two such vertices can be approximated via linear interpolation as [51]:

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

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 [51]. 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 immeadiate 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 surface, it can also cause volume shrinkage, especially for droplets and splashes that may collapse in on themselves [51]. Instead, as the name of the [51, Weighted Laplacian Smoothing] implies, 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 $|\mathcal{N}_i|$ [51]. 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 [51]:

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

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

With these weights, each vertex $\vec{\boldsymbol{x}}^k$ in the k-th smoothing iteration can be moved towards the average position $\bar{\vec{\boldsymbol{x}}}$ of neighbouring vertices using [51]:

$$\vec{\boldsymbol{x}}^{k+1} = \vec{\boldsymbol{x}}^k + \lambda \bar{\vec{\boldsymbol{x}}} \tag{5.6}$$

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

The [51, 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 cites 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 [53]. An example of such an improvement in perceived realism is shown in Figure 5.1.

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 [53, Unified Spray, Foam and Bubbles for Particle-Based Fluids], with suggestions for improvement addressed by [54, Bender et al.] in the context of micropolar SPH fluids, which were implemented 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 [54, 48].

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 [53]:

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 [53]

Generating Potentials

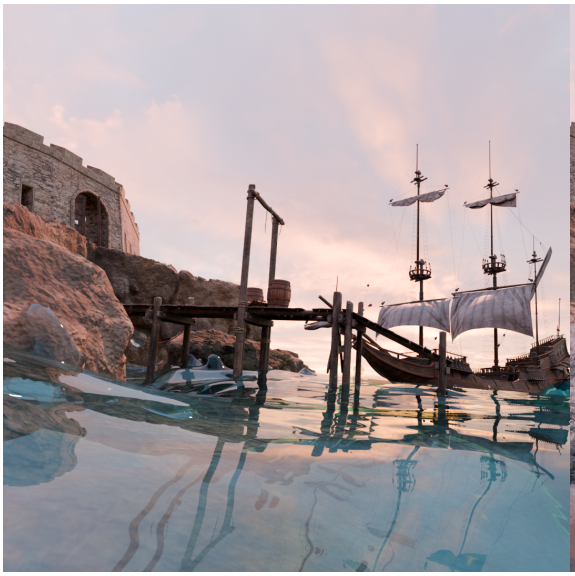
To identify strongly convex areas, the surface normal can be approximated using the normalized, negative SPH kernel gradient as:

$$\vec{\boldsymbol{n}}_i = -\sum_{j \in \mathcal{N}_i} \nabla W_{ij} \tag{5.7}$$

which can be normalized to obtain $\hat{\vec{\boldsymbol{n}}}_i$ or set to $\vec{\mathbf{0}}$ if the magnitude $|\vec{\boldsymbol{n}}|$ is insufficiently large, such as inside the fluid volume. This basically leverages the particle deficiency phenomenon from subsubsection 2.2 to detect surface normals from the incomplete particle samplings in the neighbourhood.

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 [53]. The kernel is additionally normalized as per [54, later suggestions], which makes the process more scale-invariant:

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



(a) Close-up rendering without diffuse particles

(b) Close-up rendering with diffuse particles

(c) Overview rendering without diffuse particles

(d) Overview rendering with diffuse particles

Figure 5.1: 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 5.1b and Figure 5.1d are rendered with the addition of diffuse particles representing spray, foam and air bubbles, Figure 5.1a and Figure 5.1c 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.

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 [53, 54]:

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

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. [53]:

$$\delta_i = \kappa_i \cdot \Theta\left(\hat{\vec{\boldsymbol{v}}}_i \cdot \hat{\vec{\boldsymbol{n}}}_i - 0.6\right) \tag{5.10}$$

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

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

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

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

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

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

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 [53]:

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

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

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

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

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

These potentials can then be used to determine the number of diffuse particles $N_d \in \mathbb{N}$ spawned at some time step [54]:

$$N_d = \left\lfloor k \cdot I_k \frac{I_{ta} + I_{wc}}{2} \Delta t \right\rfloor \tag{5.17}$$

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 is the maximum number of diffuse particles spawned per second of simulated time by each fluid particle [54]. 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.

Analysis

6.1 IISPH Performance Analysis

6.1.1 Impact of Solver Warm Start

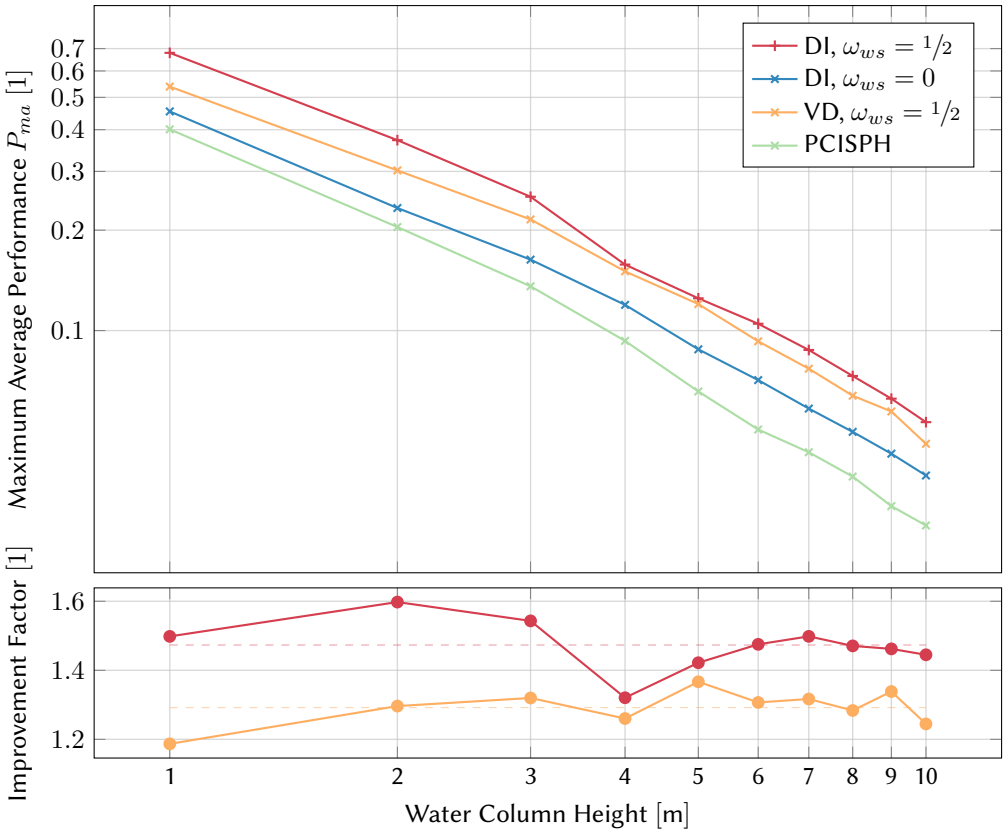


Figure 6.2: The relative performance of the standard IISPH fluid solver outlined in Alg. 1 is compared between a warm start $\omega_{ws} = 0.5$ as defined in Equation 3.63, which initializes pressure values to half the value obtained in the previous time step, and a cold start with $\omega_{ws} = 0$ that resets the pressure values to zero at the start of each time step. To measure the performance impact, the long-time average of the estimated performance metric defined in Equation 3.76 is taken and its maximum value across a simulation of 60 sec of simulated time $P_{ma}(\omega_{ws}) = \max_K \frac{1}{K} \sum_1^K \hat{P}_k = \max_K \bar{P}_K$ is plotted in the upper graph. An example of the estimated average performance graph $\bar{P}(t)$ converging towards its maximum due to the time step size controller outlined in subsection 3.4.1 can be observed in Figure 3.2 - in this instance, the maximum performance that is converged towards is measured. This maximum average performance is measured for scenes of varying complexity using water columns as shown in Figure 6.1 of increasing heights. Assuming the curves in the graph are approximately linear and parallel, both variations of the IISPH solver decrease in performance exponentially as the scene complexity increases, but the warm start results in a speed-up modelled by a constant factor. This factor is shown in the lower subplot for varying scene complexities and seems to oscillate about an improvement factor of about $\times 1.5$ when using a warm-started solver.

6.1.2 Impact of Alternative Source Terms

6.2 Comparison of IISPH with Iterated WCSPH

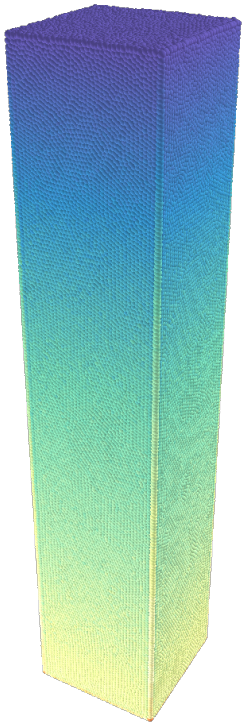


Figure 6.1: An exemplary rendering of the static water column used as a scene for performance analysis, showing particles as opaque spheres of radius $1/2 \cdot h$ in the particle spacing $h = 5$cm. Pressure is colour coded from low pressures in blue through yellow and orange towards red. This particular 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 Alg. 1 at $t = 1$s.

Appendix A

Fixed Radius Neighbour Search

There exist numerous methods for implementing the computation of the neighbour sets $\mathcal{N}_i = \{j : |\vec{\boldsymbol{x}}_{ij}| < \hbar\}$ 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 [55, Hoetzlein].

It is apparent that a naïve neighbour search incurs a cost on the order of $\mathcal{O}\left(N^2\right)$ 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 $|\vec{\boldsymbol{x}}_{ij}| < \hbar$. 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 [3].

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 [56, 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 surrounding the problem at hand, which relies on no ad-hoc parameters specific to the simulation domain or implementation.

Construction

Firstly, the grid cell a particle i resides in is computed as [49]:

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

which can be point-wise lifted to a vectorial function $(k, l, m)^T = \vec{\boldsymbol{c}}(\vec{\boldsymbol{x}}_i)$ using the point $\vec{\boldsymbol{x}}_{min}$ that defines the lowest extent of the axis-aligned bounding box of the simulation domain across each axis, with $(x_{max}, y_{max}, z_{max})^T = \vec{\boldsymbol{x}}_{max}$ 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 [49] for ensuring that spatial proximity is mirrored by proximity in memory, improving cache coherency and reducing scattered reads [55], 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 [57] 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:

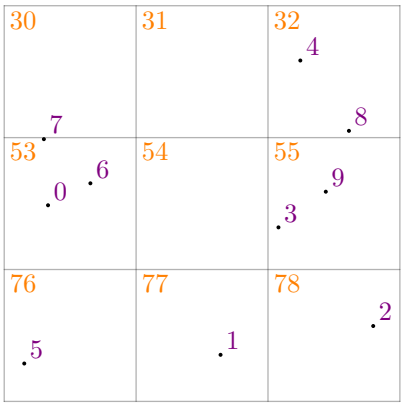
$$I\left((k, l, m)^T\right) = k + l \cdot K + m \cdot LM \tag{A.2}$$

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 $I\left(\vec{\boldsymbol{c}}(\vec{\boldsymbol{x}}_i)\right)$ of each particle can be computed and saved to the i-th entry of `indices`, while the $I\left(\vec{\boldsymbol{c}}(\vec{\boldsymbol{x}}_i)\right)$-th entry of `counts` is simultaneously incremented using an atomic add operation.

2. A parallel exclusive prefix sum or prescan of the particle counts per cell can then be performed, yielding an array $\texttt{counts}_<$ of size N_{grid} of cumulative particle counts of all cells with strictly lower



Compute cell indices

particle	0	1	2	3	4	5	6	7	8	9	…
indices	53	77	78	55	32	76	53	30	32	55	…

Count particles per cell, compute exclusive prefix sum

cell id	30	31	32	…	53	54	55	…	76	77	78
counts	1	0	2	…	2	0	2	…	1	1	1
$\text{counts}_<$	0	1	1	…	3	5	5	…	7	8	9

Sort particle indices with respect to cell id

i	0	1	2	3	4	5	6	7	8	9	…
particle	7	4	8	0	6	3	9	5	1	2	…
indices	30	32	32	53	53	55	55	76	77	78	…

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 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 [55, Hoetzlin] in the presentation the presented algorithm is based on.

indices than the cell under consideration. In Taichi, a work-efficient and optimized parallel inclusive prefix sum that avoids bank conflicts [58] and makes use of [59, 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 count

This algorithm is outlined in Algorithm 6 and illustrated in Figure A.1.

Algorithm 6 Counting Sort-based Uniform Grid Construction

Step 1 – Compute Indices and Counts

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

Step 2 – Exclusive Prefix Sum

Compute $\text{counts}_<$ from parallel prescan of counts ▷ See [58, Harris et al.]

Step 3 – Counting Sort

for $i \in \mathbb{N}_0^{N-1}$ in parallel **do**
  $I_i \leftarrow \text{indices}[i]$
  $o_1 \leftarrow \text{counts}_<[I_i]$ ▷ offset due to previous cells
  $o_2, \text{counts}[I_i] \leftarrow \text{counts}[I_i] - 1$ ▷ offset within cell, atomically decrement!
  $\text{sorted}[o_1 + o_2 - 1] \leftarrow i$
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 $\texttt{sorted}[\texttt{counts}_<]$ and the remaining particles in the same cell are the $\texttt{counts}[I_j]$ 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 $\sum_{i=-1}^{i=1} \texttt{counts}[I(k-i, l, m)]$ 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 [55] 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 [3, 49]. 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 $\vec{\boldsymbol{x}}_{min}, \vec{\boldsymbol{x}}_{max}$ 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 $\vec{\boldsymbol{x}}_i$ is guaranteed to lie in the axis aligned bounding box spanned by $\vec{\boldsymbol{x}}_{min}, \vec{\boldsymbol{x}}_{max}$. 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 $\vec{\boldsymbol{x}}_i, \vec{\boldsymbol{v}}_i, m_i$ 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.

Appendix B

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:

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

$$I_{xx} = \mathcal{M}_{200} \tag{B.2}$$

$$I_{yy} = \mathcal{M}_{020} \tag{B.3}$$

$$I_{zz} = \mathcal{M}_{002} \tag{B.4}$$

$$I_{xy} = \mathcal{M}_{100} + \mathcal{M}_{010} \tag{B.5}$$

$$I_{xz} = \mathcal{M}_{100} + \mathcal{M}_{001} \tag{B.6}$$

$$I_{yz} = \mathcal{M}_{010} + \mathcal{M}_{001} \tag{B.7}$$

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

Only the solution to the volume integrals in Equation B.8 are 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 [46, 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 [46]:

$$\frac{I_{xx}\left(\text{◬}_{ABCO}\right)}{\rho\left|\det(J\left(\text{◬}_{ABCO}\right))\right|} = \frac{\left(A_y^2 + A_yB_y + B_y^2 + A_yC_y + B_yC_y + C_y^2 + A_z^2 + A_zB_z + B_z^2 + A_zC_z + B_zC_z + C_z^2\right)}{60} \tag{B.9}$$

$$\frac{I_{yy}\left(\text{◬}_{ABCO}\right)}{\rho\left|\det(J\left(\text{◬}_{ABCO}\right))\right|} = \frac{\left(A_x^2 + A_xB_x + B_x^2 + A_xC_x + B_xC_x + C_x^2 + A_z^2 + A_zB_z + B_z^2 + A_zC_z + B_zC_z + C_z^2\right)}{60} \tag{B.10}$$

$$\frac{I_{zz}\left(\text{◬}_{ABCO}\right)}{\rho\left|\det(J\left(\text{◬}_{ABCO}\right))\right|} = \frac{\left(A_x^2 + A_xB_x + B_x^2 + A_xC_x + B_xC_x + C_x^2 + A_y^2 + A_yB_y + B_y^2 + A_yC_y + B_yC_y + C_y^2\right)}{60} \tag{B.11}$$

$$\frac{I_{xy}\left(\text{◬}_{ABCO}\right)}{\rho\left|\det(J\left(\text{◬}_{ABCO}\right))\right|} = \frac{(2A_xA_y + B_xA_y + C_xA_y + A_xB_y + 2B_xB_y + C_xB_y + A_xC_y + B_xC_y + 2C_xC_y)}{120} \tag{B.12}$$

$$\frac{I_{xz}\left(\text{◬}_{ABCO}\right)}{\rho\left|\det(J\left(\text{◬}_{ABCO}\right))\right|} = \frac{(2A_xA_z + B_xA_z + C_xA_z + A_xB_z + 2B_xB_z + C_xB_z + A_xC_z + B_xC_z + 2C_xC_z)}{120} \tag{B.13}$$

$$\frac{I_{yz}\left(\text{◬}_{ABCO}\right)}{\rho\left|\det(J\left(\text{◬}_{ABCO}\right))\right|} = \frac{(2A_yA_z + B_yA_z + C_yA_z + A_yB_z + 2B_yB_z + C_yB_z + A_yC_z + B_yC_z + 2C_yC_z)}{120} \tag{B.14}$$

where $\rho\left|\det(J\left(\text{◬}_{ABCO}\right))\right|$ 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 $\operatorname{sgn}\left(\triangleq_{ABCO}\right)$ and therefore yields values I'_{ij}, I'_{kk} that can be directly accumulated in a sum across all triangles $\triangle_{ABC}$. Here, i, j, k are used to index the components of the vertices. Let $\vec{\boldsymbol{V}} = (A_i, B_i, C_i)^T$ and $\vec{\boldsymbol{W}} = (A_j, B_j, C_j)^T$ be the vectors of the i-th and j-th components of $\vec{\boldsymbol{A}}, \vec{\boldsymbol{B}}, \vec{\boldsymbol{C}}$ for $i \neq j \neq k \neq i$ and one can obtain:

$$I'_{kk}\left(\triangleq_{ABCO}\right) = \frac{\rho \det(J\left(\triangleq_{ABCO}\right))}{120}\left(\vec{\boldsymbol{V}}^2 + \left(\left|\vec{\boldsymbol{V}}\right|_{L1}\right)^2 + \vec{\boldsymbol{W}}^2 + \left(\left|\vec{\boldsymbol{W}}\right|_{L1}\right)^2\right) \tag{B.15}$$

for the moments of inertia on the diagonal using $\vec{\boldsymbol{V}}^2 = \vec{\boldsymbol{V}} \cdot \vec{\boldsymbol{V}}$ and the L_1 or sum norm $\left|\vec{\boldsymbol{V}}\right|_{L1} = A_i + B_i + C_i$ as well as:

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

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 $\triangle_{ABC}$ and summed using a parallel reduction to yield the components $I^{sum}_{kk}, I^{sum}_{ij}$ of the inertia tensor:

$$I^{sum}_{kk} = \sum_{\triangle_{ABC}} I'_{kk}\left(\triangleq_{ABCO}\right) \tag{B.17}$$

$$I^{sum}_{ij} = \sum_{\triangle_{ABC}} I'_{ij}\left(\triangleq_{ABCO}\right) \tag{B.18}$$

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: [47]

$$I_{kk} = I^{sum}_{kk} - M\left(r_i^2 + r_j^2\right) \tag{B.19}$$

$$I_{ij} = I^{sum}_{ij} - M r_i r_j \tag{B.20}$$

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

Bibliography

[1] J. D. Anderson, *Computational Fluid Dynamics: The Basics with Applications.* (McGraw-Hill series in mechanical engineering). 1995.

[2] J. Karrer, "Equation of State Solvers for Smoothed Particle Hydrodynamics," (Lab Course Report), 2024.

[3] 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.

[4] W. M. Lai, D. H. Rubin, D. Rubin, and E. Krempl, *Introduction to continuum mechanics*, Third. Butterworth-Heinemann, 1999.

[5] 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.

[6] A. Falaschi, *Signal Processing and Information Theory*, 1st ed. Jul. 2022.

[7] D. J. Price, "Smoothed particle hydrodynamics and magnetohydrodynamics," *Journal of Computational Physics*, vol. 231, no. 3, pp. 759–794, Feb. 2012.

[8] P. Getreuer, "A Survey of Gaussian Convolution Algorithms," *Image Processing On Line*, vol. 3, pp. 286–310, 2013.

[9] S. Band, *Boundary Handling and Neighbor Search in Iterative Incompressible SPH*, (PhD Thesis), 2020.

[10] 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.

[11] 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.

[12] J. R. Macdonald, "Review of Some Experimental and Analytical Equations of State," *Rev. Mod. Phys.*, vol. 41, pp. 316–349, 2 Apr. 1969.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] D. Violeau and A. Leroy, "Optimal time step for incompressible SPH," *Journal of Computational Physics*, vol. 288, pp. 119–130, 2015, ISSN: 0021-9991.

[20] E. Haines, "Point in polygon strategies," in *Graphics Gems IV*. USA: Academic Press Professional, Inc., 1994, pp. 24–46, ISBN: 0123361559.

[21] Dawson-Haggerty et al., *Trimesh*, version 4.5.2, (software, `https://trimesh.org/`).

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] J. Bender *et al.*, *SPlisHSPlasH Library*, (software, `https://github.com/InteractiveComputerGraphics/SPlisHSPlasH`).

[29] A. Bardou, "Online Learning for the Black-Box Optimization of Wireless Networks," Theses, Ecole normale supérieure de lyon - ENS LYON, Sep. 2023.

[30] 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.

[31] N. Bastianello, "tvopt: A Python Framework for Time-Varying Optimization," in *2021 60th IEEE Conference on Decision and Control (CDC)*, 2021, pp. 227–232.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] J. Huang, H. Su, and L. J. Guibas, "Robust watertight manifold surface generation method for shapenet models," *CoRR*, vol. abs/1802.01698, 2018.

[37] 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.

[38] Francis Williams, *Point cloud utils*, version 4.5.2, (software, `https://www.github.com/fwilliams/point-cloud-utils`), 2022.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] H. Goldstein, J. L. Safko, and C. P. Poole, *Classical mechanics: Pearson new international edition*, en, 3rd ed. Pearson Higher Ed, Mar. 2014.

[44] D. Baraff, "Physically Based Modeling: Rigid Body Simulation," *SIGGRAPH Course Notes, ACM SIGGRAPH*, vol. 2, no. 1, pp. 2–1, 2001.

[45] 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.

[46] 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.

[47] B. Mirtich, "Fast and accurate computation of polyhedral mass properties," *J. Graphics, GPU, & Game Tools*, vol. 1, pp. 31–50, 1996.

[48] 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.

[49] S. Band, C. Gissler, and M. Teschner, "Compressed Neighbour Lists for SPH," *Computer Graphics Forum*, vol. 39, no. 1, pp. 531–542, 2020.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] R. Hoetzlein, "Fast Fixed-Radius Nearest Neighbors: Interactive Million-Particle Fluids," (talk slides), GPU Technology Conference (GTC) 2014, Santa Clara, CA, 2014.

[56] 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.

[57] H. Tropf and H. Herzog, "Multimensional Range Search in Dynamically Balanced Trees," *Angew. Inform.*, vol. 23, pp. 71–77, 1981.

[58] 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.

[59] G. E. Blelloch, "Prefix sums and their applications," Jun. 2018.