Simulating Incompressible Fluids with Weakly 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]:

Definition 1. *The material derivative:*

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

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

$$\frac{D\rho}{Dt} + \rho\,(\nabla \cdot \vec{\boldsymbol{v}}) = 0 \tag{2.1}$$

which has to hold across the entire volume integral over the fluid domain [1]. For incompressible flows, 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 3.3.

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

$$\rho \frac{D\vec{\boldsymbol{v}}}{Dt} = \nabla \cdot \mathbb{T} + \rho \vec{\boldsymbol{b}}^{ext} \tag{2.2}$$

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

$$\mathbb{T} = -p\mathbb{1} + \mu \left(\nabla \vec{\boldsymbol{v}} + (\nabla \vec{\boldsymbol{v}})^T \right) \tag{2.3}$$

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

$$\frac{D\vec{\boldsymbol{v}}}{Dt} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \vec{\boldsymbol{v}} + \vec{\boldsymbol{b}}^{ext} \tag{2.4}$$

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

$$\int_{-\infty}^{\infty} \delta(x)\, dx = 1 \qquad \textit{normalization} \tag{2.5}$$

$$x \neq 0 \implies \delta(x) = 0 \tag{2.6}$$

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]:

$$f(x) = \int_{-\infty}^{\infty} f(x_s)\delta(x - x_s)\, dx \tag{2.7}$$

Which means that by the convolution theorem [3]:

$$f(\vec{\boldsymbol{x}}) = \int_{\Omega} f(\vec{\boldsymbol{x}}')\delta(\vec{\boldsymbol{x}} - \vec{\boldsymbol{x}}')\, dV' = (f \star \delta)(\vec{\boldsymbol{x}}) \tag{2.8}$$

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 2.7 is approximated by a sum over the finite fluid samples.

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

$$f(\vec{\boldsymbol{x}}_i) = \int_{\Omega} f(\vec{\boldsymbol{x}}_j)\delta(\vec{\boldsymbol{x}}_i - \vec{\boldsymbol{x}}_j)\, dV_j \tag{2.9}$$

$$\approx \int_{\Omega} f(\vec{\boldsymbol{x}}_j) W(\vec{\boldsymbol{x}} - \vec{\boldsymbol{x}}_j, \hbar)\, dV_j \qquad \delta(\vec{\boldsymbol{x}}_{ij}) \approx W(\vec{\boldsymbol{x}}_{ij}, \hbar) \tag{2.10}$$

$$\approx \sum_{j \in \mathcal{N}_i} f(\vec{\boldsymbol{x}}_j) W(\vec{\boldsymbol{x}}_{ij}, \hbar) V_j \qquad \textit{finite sample set} \tag{2.11}$$

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

$$\langle f_i \rangle = \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} f_j W_{ij} \tag{2.12}$$

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 fulfill [3]:

$$\int_\Omega W(\vec{\boldsymbol{x}}_{ij}, \hbar) = 1 \qquad \textit{normalization} \tag{2.13}$$

$$\lim_{\hbar \to 0} W(\vec{\boldsymbol{x}}_{ij}, \hbar) = \delta(\vec{\boldsymbol{x}}_{ij}) \qquad \textit{Dirac-}\delta\textit{ condition} \tag{2.14}$$

Further useful properties include [3]:

$$|\vec{\boldsymbol{x}}_{ij}| > \hbar \implies W(\vec{\boldsymbol{x}}_{ij}, \hbar) \qquad \textit{compact support} \tag{2.15}$$

$$W(\vec{\boldsymbol{x}}_{ij}, \hbar) = W(\vec{\boldsymbol{x}}_{ji}, \hbar) \qquad \textit{spherical symmetry} \tag{2.16}$$

$$W(\vec{\boldsymbol{x}}_{ij}, \hbar) > 0 \qquad \textit{positivity} \tag{2.17}$$

$$W(\vec{\boldsymbol{x}}_{ij}, \hbar) \in C^2 \qquad \textit{twice continuously differentiable} \tag{2.18}$$

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 [7], 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 [8]:

$$W(\vec{\boldsymbol{x}}_{ij}, \hbar) = \frac{\alpha(d)}{\hbar^d} w\left(q\right) \tag{2.19}$$

$$\nabla W(\vec{\boldsymbol{x}}_{ij}, \hbar) = \frac{\alpha(d)}{\hbar^{d+1}} \frac{\vec{\boldsymbol{x}}_{ij}}{|\vec{\boldsymbol{x}}_{ij}|} \partial_q w\left(q\right) \tag{2.20}$$

$$\tag{2.21}$$

where $q = \frac{|\vec{\boldsymbol{x}}|}{\hbar} \in [0, 1]$ and $\alpha(d)$ depends only on dimensionality. As in much of the literature, the Cubic Spline kernel is employed in this report [8]:

$$w(q) = (1-q)_+^3 - 4\left(1/2 - q\right)_+^3 \tag{2.22}$$

$$\partial_q w(q) = -3\left(1-q\right)_+^2 - 12\left(1/2 - q\right)_+^2 \tag{2.23}$$

$$\alpha(3) = 16/\pi \tag{2.24}$$

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:

$$\left\langle \nabla \vec{\boldsymbol{f}}_i \right\rangle_\otimes = \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} \vec{\boldsymbol{f}}_j \otimes \nabla W_{ij} \tag{2.25}$$

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 even for suboptimal samplings, yielding a more accurate approximation for the gradient of a scalar field [3, 9]:

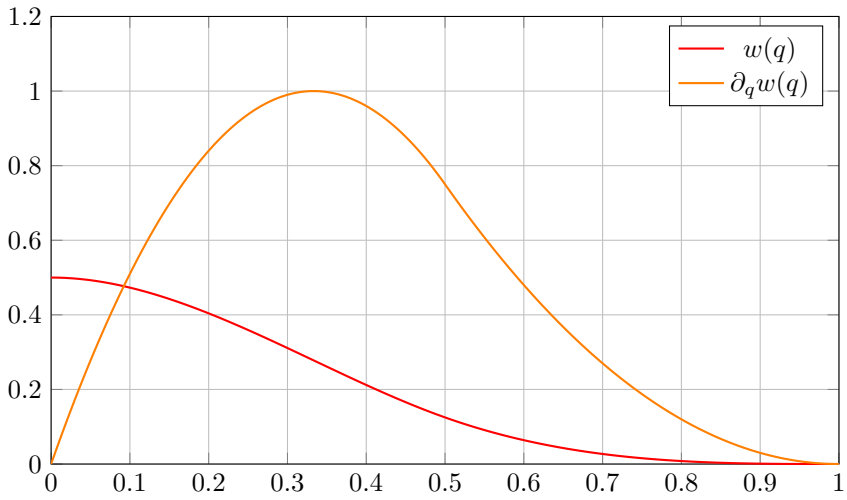

Figure 2.1: The shape function $w(q)$ that defines the Cubic Spline kernel function and its first derivative are plotted for all values of $q \in [0, 1]$ that yield non-zero $w(q)$. The function is compactly supported. Since the kernel function W is spherically symmetric, this one-dimensional function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

Difference Formula

$$\langle \nabla f_i \rangle_- = \langle \nabla f_i \rangle - f_i \langle \nabla 1 \rangle = \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} (f_j - f_i) \nabla_i W_{ij} \tag{2.26}$$

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, 9]:

Symmetric Formula

$$\langle \nabla f_i \rangle_\parallel = \rho_i \sum_{j \in \mathcal{N}_i} m_j \left(\frac{f_i}{\rho_i^2} + \frac{f_j}{\rho_j^2} \right) \nabla_i W_{ij} \tag{2.27}$$

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 [9]. 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

$$\left\langle \nabla^2 \vec{\boldsymbol{f}}_i \right\rangle_\Delta = 2(d+2) \sum_{j \in \mathcal{N}_i} \frac{m_j}{\rho_j} \frac{\vec{\boldsymbol{f}}_{ij} \cdot \vec{\boldsymbol{x}}_{ij}}{|\vec{\boldsymbol{x}}_{ij}|^2} \nabla_i W_{ij} \tag{2.28}$$

[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.

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

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 section 3.2. Multiple source terms and variations of the solver are available, which are discussed in section 3.3.

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 2.1 and momentum equation Equation 2.4. 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 2.28:

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

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 2.27 a robust choice to obtain physically accurate results:

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

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 2.12:

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

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 section 3.2.

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 2.4 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 $\lambda \in]0, 1]$ of their size or spacing h within one time step, which for a global adaptive time step size implies [3]:

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

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 '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:

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

- $$m_i^0 = \frac{1}{h^d} \tag{3.8}$$
$$\langle \rho_i \rangle^l \leftarrow \sum_{j \in \mathcal{N}_i} m_j^l W_{ij} \tag{3.9}$$
$$m_i^{l+1} \leftarrow \frac{1}{2} m_i^l + \frac{1}{2} \left(m_i^l + \frac{\rho_0}{\langle \rho_i \rangle^l} \right) \tag{3.10}$$

  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 Implicit Incompressible SPH

3.3 Alternative Source Terms and Variations

Chapter 4

Weakly Coupled Rigid Bodies

4.1 Versatile and Robust Boundary Particles

4.2 Mass Moments of Triangular Meshes

4.3 Rigid Body Kinematics

Chapter 5

Visualization

5.1 Spray, Foam and Bubble Generation

5.2 Rendering

Chapter 6

Analysis

6.1 Solver Convergence

6.2 Source Terms and Stability

6.3 Performance Scaling

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 [22, Hoetzlein].

It is apparent that a naïve neighbour search incurrs 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 [23, 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 [24]:

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

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 [24] for ensuring that spatial proximity is mirrored by proximity in memory, improving cache coherency and reducing scattered reads [22], 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 [25] that a Z-order curve would necessitate.

The index 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

indices than the cell under consideration. In Taichi, a work-efficient and optimized parallel inclusive prefix sum that avoids bank conflicts [26] and makes use of [27, 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 $\texttt{counts}_<$ 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 1

Algorithm 1 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 $I_i \leftarrow I\left(\vec{\boldsymbol{c}}(\vec{\boldsymbol{x}}_i)\right)$ ▷ XYZ curve of Equation A.2
  $\texttt{indices}\left[i\right] \leftarrow I_i$
  $\texttt{counts}\left[I_i\right] \leftarrow \texttt{counts}\left[I_i\right] + 1$ ▷ atomically increment
end for

Step 2 – Exclusive Prefix Sum

Compute $\texttt{counts}_<$ from parallel prescan of counts ▷ See [26, Harris et al.]

Step 3 – Counting Sort

for $i \in \mathbb{N}_0^{N-1}$ in parallel **do**
  $I_i \leftarrow \texttt{indices}\left[i\right]$
  $o_1 \leftarrow \texttt{counts}_<\left[I_i\right]$ ▷ offset due to previous cells
  $o_2, \texttt{counts}\left[I_i\right] \leftarrow \texttt{counts}\left[I_i\right] - 1$ ▷ offset within cell, atomically decrement!
  $\texttt{sorted}\left[o_1 + o_2 - 1\right] \leftarrow i$
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 then $\texttt{counts}[I_j]$ 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 9 coherent sections of memory to be read instead of 27 in the worst case.

Discussion

The algorithm shown [22] is simple, elegant, can achieve high performance and is very much suited for massively parallel hardware. One disadvantage however, is that the entire simulation domain is represented, 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, 24]. Since the explicit representation is small, with only a few 32bit 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, 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 each the positive and negative directions is then used, such

that bound checks are unnecessary and branches in the very hot code path of the query procedure are avoided.

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

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

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

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

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

[23] 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.

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

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

[26] 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.

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