Universität Stuttgart

Institut für Parallele und Verteilte Systeme

Universitätsstraße 32
70569 Stuttgart

Masterarbeit

Development of an Euler-Lagrangian framework for point-particle tracking to enable efficient multiscale simulations of complex flows

Helena Kschidock

Studiengang:	M.Sc. Simulation Technology
1. Prüfer:	Jun.-Prof. Dr. rer. nat. Benjamin Uekermann
2. Prüfer:	Prof. Dr. rer. nat. Miriam Schulte
Betreuer:	Dr. Andrea Pasquali
begonnen am:	02.05.2023
beendet am:	02.11.2023

Abstract

In this work, we implement, test, and validate an Euler-Lagrangian point-particle tracking framework for the commercial aerodynamics and aeroacoustics simulation tool ultraFluidX®, which is based on the Lattice Boltzmann Method and optimized for GPUs. Our framework successfully simulates one-way and two-way coupled particle-laden flows based on drag forces and gravitation. Trilinear interpolation is used for determining the fluid's macroscopic properties at the particle position. Object and domain boundary conditions are implemented using a planar surface approximation. The whole particle framework is run within three dedicated GPU kernels, and data is only copied back to the CPU upon output. We show validation for the velocity interpolation, gravitational acceleration, back-coupling forces and boundary conditions, and test runtimes and memory requirements. We also propose the next steps required to make the particle framework ready for use in engineering applications.

Abstrakt

In dieser Arbeit implementieren, testen und validieren wir Euler-Langrangsches Punkt-Teilchen Framework für das kommerzielle Aerodynamik- und Aeroakustik-Simulationsprogramm ultraFluidX®, das auf der Lattice-Boltzmann-Methode basiert und GPUs optimiert ist. Unser System simuliert erfolgreich ein- und zweiseitig gekoppelte partikelbeladene Strömungen auf der Grundlage von Widerstandkräften und Gravitation. Zur Bestimmung der makroskopischen Eigenschafter der Flüssigkeit an der Partikelposition wird eine trilineare Interpolation verwendet. Die Randbedingungen des Objekts und des Bereichs werden durch eine planare Oberflächenapproximation implementiert. Das gesamte Partikelframework wird in drei dedizierten GPU-Kernels asgefährt, und die Daten werden nur bei Ausgabe zurück auf die CPU kopiert. Wir zeigen die Validierung der Geschwindigkeitsinterpolation, der Gravitationsbeschleunigung und der Rückkopplungskräfte und testen Laufzeiten und Speicherbedarf. Wir schlagen auch die nächsten Schritte vor, die erforderlich sind, um das Partikelframework für den Einsatz in technischen Anwendungen bereit zu machen.

1 Introduction

Particle tracking is central to many applications in the world of Computational Fluid Dynamics (CFD), especially for the simulation of vehicles. Impact and deposition of snow, dust, fluid droplets or soil particles, as well as changes in the flow behavior can be of great relevance to automobile and agricultural engineers.

In CFD, particle tracking is performed using so-called particle-laden flow simulations. The term describes a two-phase flow combining a continuously connected fluid phase with a particle phase consisting of small immiscible entities [1]. There exists a large body of work on the modelling of particle-laden flows, with several different approaches. Eulerian methods treat particles as a second continuous phase described by transport equations, see e.g. [2], whereas Lagrangian methods model particles as discrete units with individual equations of motion, see e.g. [3]. For simulations based on the Lattice Boltzmann Method (LBM), Euler-Euler approaches are more efficient, especially for simulations with large numbers of light particles, while Euler-Lagrangian approaches are considered to provide the more accurate results [4]. Several established Euler-Lagrangian methods for particle-laden flows exist. The discrete element method is centered heavily around the accurate simulation of particle-particle interactions [5, 6], while particle-resolved direct numerical simulations use fine grids to capture the interactions between large particles and the fluid, see e.g. [7]. Both methods are computationally expensive, and are therefore used mostly for particle simulations with large volume fractions. For smaller or fewer particles, the point-particle approach, see e.g. [8], can be applied at far lower cost: Here, the actual shape and volume of the particle are disregarded and the particle itself is not resolved by the fluid. Instead, the interactions between fluid and particle phase are captured through additional force terms which attempt to model the relevant behaviors in a simplified way – from drag forces to temperature exchange.

Altair®ultraFluidX®[1] is a commercial simulation tool for external aerodynamics and aeroacoustics. With a GPU-based architecture and an implementation based on the Lattice Boltzmann method (LBM), it was developed to deliver over-night simulation results even for complex aerodynamic simulations. However, it does not have any capabilities for simulating particle-laden flows. **The goal of this thesis is therefore to develop and implement a framework for particle tracking into ultraFluidX®.** Based on the considerations above, we use an Euler-Lagrangian point-particle approach for our implementation. The work of this project is integrated directly into the ultraFluidX®code, and orients itself on the GPU-based architecture. The focus lies in developing a solid framework - from general data structures to algorithms. We also show extensive validation of the code, to ensure the fidelity of our framework and ease future use, maintenance and development of the software.

In order to define a sufficiently tight scope for this thesis, we limit our framework to one-

[1] https://altair.com/altair-cfd-capabilities#lbm

and two-way coupled simulations. This means that we only consider interactions between particles and fluid; particle-particle interactions (four-way coupling) are disregarded. In the future, accessibility to four-way coupled particle simulations will be achieved by coupling ultraFluidX®with Altair®'s direct element particle simulation software EDEM™ [2]. We further limit ourselves to single-GPU simulations on uniform grids for this thesis. In the future, this will be extended to enable multi-GPU simulations on refined grids.

This thesis is organized as follows: In Chapter 2, we relay the necessary theory behind our framework. We summarize the LBM as used in the code before setting out the equations used for one-way and two-way coupled particle simulations. Chapter 3 then goes into detail about our implementation of particle-laden flows in ultraFluidX®, describing all relevant algorithms and methods used for the general framework, fluid-particle coupling, particle motion, and boundary conditions. This is the core of the thesis. Additionally, we validate each part of the code that we have implemented. The results of such testing are shown in Chapter 4, alongside runtime and grid studies for performance evaluation. An outlook is provided in Chapter 5, which also includes a guideline for the next steps required to fully integrate particle-laden flows into the existing software.

[2] https://altair.com/edem

2 Theory

In this chapter, we introduce all the physics and theory that is either directly used in our implementation or is relevant to a general understanding of the code and approach. Section 2.1 introduces the LBM used for the fluid simulation. It draws heavily on Krüger et al.'s 2017 book [9] as a standard reference in the field. Section 2.2 introduces the theory behind one-way and two-way coupled particle-laden flows and contains most of the equations implemented in our particle framework. It orients itself on the 2015 paper by Banari et al. [10], who also used a point-particle method coupled with an LBM simulation and serves as a core reference for this thesis.

2.1 The Lattice Boltzmann Method

2.1.1 Introduction to the Lattice Boltzmann Method

The central equations of fluid dynamics are the Navier-Stokes equations, which describe the macroscopic behavior of a viscous fluid by means of a momentum balance [11]. The fluid is treated as a a continuum, with the base assumption that its macroscopic physical properties, such as density, velocity, and pressure, can be represented by continuous functions in time and space. The conventional CFD approach to simulating a fluid is then to discretize these macroscopic properties on a grid of choice before numerically solving the Navier-Stokes PDEs in whatever form is appropriate to the selected use case. A variety of methods exists, amongst which there are three main types that are currently dominating the field: finite difference methods, which approximate derivatives using values stored at the nodes of a regular grid; finite element methods, which approximate a global solution from values stored at the nodes of arbitrarily shaped elements via local interpolation functions; and finite volume methods, which are based on the conversion of divergence terms into surface integrals, with node values representing the average over a small surrounding control volume.

Over the last few decades, an alternative approach has emerged, and has by now established itself as a core CFD tool, specifically for solving the weakly compressible Navier-Stokes equation: the Lattice Boltzmann method (LBM) [12]. Instead of approaching the problem from the side of the macroscopic fluid properties, the LBM instead models the bulk behavior of the fluid particles (not to be confused with the particles in particle-laden flows) through discrete particle distribution functions $f_i(\vec{x}, t)$. These distribution functions, also referred to as particle *populations*, contain both density and velocity information for position $\vec{x}$ and time t [9, p. 63]. The domain is discretized using a square grid (in 2D) or cubic grid (in 3D). Each grid cell (which we will refer to as a *voxel* in the context of our framework) is assigned a fixed number of distribution functions based on a set of discrete velocities $\{\vec{c}_i\}$. Standard velocity

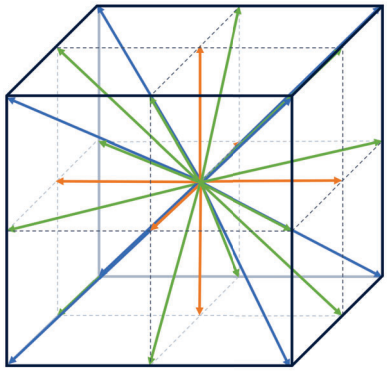

Figure 2.1: The D3Q27 pattern, with vectors representing the discrete velocity set used to define the density-velocity distributions. A $\vec{0}$-velocity rest is situated at the origin, the others are directed towards the corners (8, blue), the centers of the faces (6, orange) and the centers of the edges (12, green).

sets used for solving the Navier-Stokes equations include D2Q9, D3Q19 and D3Q27, where in the DdQq notation, d denotes the number of spatial dimensions and q refers to the number of discrete velocities. A visualization of D3Q27, which is used in ultraFluidX®, is shown in Figure 2.1. The choice of set depends on the application and computational capacities, as a higher number of distributions does increase accuracy, but also requires more memory and computations.

2.1.2 The Lattice Boltzmann equation

Before introducing the LB equation, we take a look at some of the physics that it is built on. Additional details can be found in [9]. In kinetic theory, particle distribution functions $f(\vec{x},\vec{\xi},t)$ can be understood as the density of particles with velocity $\vec{\xi}$ at position $\vec{x}$ and time t, which can be imagined as an extension of the 'classic' spatial particle density function with additional velocity information.

For large enough times, the distribution of identical non-interacting particles in thermal equilibrium will tend towards an equilibrium distribution

$$f^{\mathrm{eq}}(\vec{x},|\vec{v}|,t) = \rho\left(\frac{1}{2\pi RT}\right)^{3/2} e^{-|\vec{v}|^2/(2RT)}, \tag{2.1}$$

the so-called *Maxwell-Boltzmann distribution*, with density ρ, temperature T and the universal gas constant R.

The **Boltzmann equation** describes the evolution of the density function in time. With $f \equiv f(\vec{x},\vec{\xi},t)$ and using the Einstein convention for the summation index β, it can be written as

$$\left(\frac{\partial}{\partial t} + \xi_\beta \frac{\partial}{\partial x_\beta} + B_\beta \frac{\partial}{\partial \xi_\beta}\right) f = \frac{\mathrm{d}f}{\mathrm{d}t} \equiv \Omega(f), \tag{2.2}$$

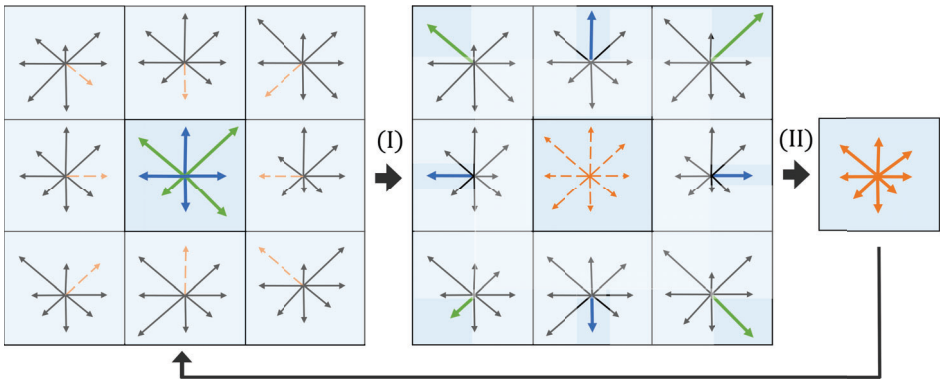

Figure 2.2: Visualization of the LB algorithm with streaming (I) and collision (II) steps.

where the second term represents the advection, $\vec{B}$ represents any external body forces acting on the fluid, and $\Omega(f)$ is the nonlinear source term caused by collisions amongst the fluid particles.

Discretizing the Boltzmann equation leads to the central equation of the LBM: the **Lattice Boltzmann equation**:

$$f_i(\vec{x} + \vec{c}_i \Delta t, t + \Delta t) = f_i(\vec{x}, t) + \Omega_i(\vec{x}, t)\,. \tag{2.3}$$

i here refers to the index of particle velocity within the set of discrete velocities $\{\vec{c}_i\}$ that we have already introduced in the context of the DdQq notation in Section 2.1.1. The equation expresses that particle distribution functions move from one spacetime position to the next via their corresponding discrete velocity $\vec{c}_i$ and timestep Δt, while being modified via the collision operator Ω_i. This form of the LB equation is reflected directly in the LB algorithm.

2.1.3 The Lattice Boltzmann algorithm

There are two steps to the general LB algorithm, which are repeated in every timestep - collision and streaming. Distributions are defined on grid cells. During **collision**, particles are redistributed amongst the distributions within a grid cell, modelling the energy exchange between particles due to particle-particle interactions. In the second step, the **streaming** step, distributions are exchanged between grid cells following the directions of their discrete velocity vectors. This simulates the advection process. A visualization of the two steps can be found in Figure 2.2.

The **collision operator** used for this is central to the implementation of any LB simulation, and various options exist. The standard one, the Bhatnagar-Gross-Krook (BGK) operator [13],

$$\Omega_i = -\frac{f_i - f_i^{\mathrm{eq}}}{\tau}\Delta t, \tag{2.4}$$

simply relaxes each distribution towards the equilibrium distribution f_i^{eq}. τ is the relaxation time, and represents the rate of the equilibration. However, this extremely simple approach is insufficient for many applications, lacking stability and coming with other deficiencies, such

as requiring a fixed ratio between kinematic and bulk viscosities. As an improvement, the generalized lattice Boltzmann equation (GLBE), also called multiple-relaxation-time (MRT) LBE was proposed [14]. As the name suggests, MRT methods allow the relaxation at different rates for distribution moments of different order. ultraFluidX®uses a cumulant collision operator with 4th order accuracy in diffusion [15].

2.1.4 Macroscopic fluid properties

The distributions themselves are generally of little interest to the user. By integrating them over the entire velocity space we can calculate the moments of the distribution, which correspond to macroscopic properties such as the mass density

$$\rho(\vec{x},t) = \int f(\vec{x},\vec{\xi},t)\,\mathrm{d}^3\xi, \tag{2.5}$$

or the local mean velocity

$$\vec{u}(\vec{x},t) = \frac{1}{\rho(\vec{x},t)} \int \vec{\xi} f(\vec{x},\vec{\xi},t)\,\mathrm{d}^3\xi\,. \tag{2.6}$$

The discrete equivalents of the above equations are then written as sums

$$\rho(\vec{x},t) = \sum_i f_i(\vec{x},t) \qquad \text{and} \qquad \vec{u}(\vec{x},t) = \frac{1}{\rho(\vec{x},t)} \sum_i \vec{c}_i f_i(\vec{x},t). \tag{2.7}$$

2.1.5 Lattice Boltzmann units

As we have already noted, an LB simulation is always based on square or cubic lattices. Additionally, the two steps above - streaming and collision - occur over a fixed timestep. This means that it would be beneficial to set our time step and mesh length to 1, and adapt all units accordingly. The new system of units is called *LB units* or *lattice units*. To convert between SI and LB units, we introduce the following conversion factors:

- The coarsest mesh size in SI units L_0 is used for converting between spatial units,

$$L_{LB} = \frac{L_{SI}}{L_{\mathrm{ref}}} \quad \text{with} \quad L_{\mathrm{ref}} = L_0\,. \tag{2.8}$$

- For the density, we simply normalize by the mean fluid density ρ_0,

$$\rho_{LB} = \frac{\rho_{SI}}{\rho_{\mathrm{ref}}} \quad \text{with} \quad \rho_{\mathrm{ref}} = \rho_0\,. \tag{2.9}$$

- We set the conversion factor for the velocity by keeping the relationship between Mach numbers in LB and SI units constant. The Mach number is defined as the ratio between the flow velocity u and the speed of sound c_s. Therefore

$$\frac{\mathrm{Ma_{LB}}}{\mathrm{Ma_{SI}}} = \mathrm{const} \quad \Rightarrow \quad u_{\mathrm{ref}} \equiv \frac{u_{\mathrm{SI}}}{u_{\mathrm{LB}}} = \frac{c_s^{\mathrm{SI}} \cdot \mathrm{Ma_{SI}}}{c_s^{\mathrm{LB}} \cdot \mathrm{Ma_{LB}}}\,. \tag{2.10}$$

The speed of sound in LB units is fixed to $c_s^{\mathrm{LB}} = \frac{1}{\sqrt{3}}$ [9, p.272]. Knowing that $c_s^{\mathrm{SI}} = \sqrt{\gamma R_* T}$, with specific heat ratio γ, specific gas constant R_*, and mean temperature T, we then arrive at our velocity conversion factor u_{ref} and

$$u_{LB} = \frac{u_{SI}}{u_{\mathrm{ref}}} \quad \text{with} \quad u_{\mathrm{ref}} = \sqrt{3\gamma R_* T} \cdot \frac{\mathrm{Ma}_{\mathrm{SI}}}{\mathrm{Ma}_{\mathrm{LB}}}. \tag{2.11}$$

- The conversion factor of the timestep can simply be derived from the three conversion factors above,

$$t_{LB} = \frac{t_{SI}}{t_{\mathrm{ref}}} \quad \text{with} \quad t_{\mathrm{ref}} = \frac{L_{\mathrm{ref}}}{u_{\mathrm{ref}}}. \tag{2.12}$$

  Due to $\Delta t_{\mathrm{LB}} = 1$, the physical time step is then $\Delta t_{\mathrm{SI}} = t_{\mathrm{ref}}$.

For other units, we proceed in the same fashion.

2.2 Particle-laden flows

2.2.1 Dispersed point-particle

We model the discrete particle phase using spherical Lagrangian point-particles. Each particle is given a density ρ_p and diameter d_p, and ergo a volume $V_p = \frac{\pi d_p^3}{6}$ and mass $m_p = \rho_p V_p$, as well as a position $\vec{x}_p$ and velocity $\vec{v}$. Particle volume is only used to approximate the drag and gravitational force, but does not displace the surrounding fluid. Particles are assumed to be rigid. Anisotropies and changes in the particle shape, as well as any form of particle rotation are neglected. In order to resolve turbulent phenomena and receive physically accurate results despite the point-particle approximation, particles should be smaller than the Kolmogorov length scale $d_p \ll \eta_K = \left(\frac{\nu^3}{\epsilon}\right)^{1/4}$ [16], where ν is the kinematic viscosity of the fluid and η is the dissipation rate of turbulence kinetic energy per unit mass. However, for certain validation cases, this assumption can be irrelevant.

2.2.2 The particle force balance equation

The movement of point-particles can be described in a simple Newtonian equation of motion. Depending on the simulated system's physical properties, scale, and the desired accuracy, there are various external force terms that can be included and added to the right-hand side of the equation, such as viscous drag, gravity, virtual mass (based on the fluid accelerating to fill the space left behind by the particle), and pressure gradient [17]. We limit ourselves to gravitational forces (weight and buoyancy) and the viscous drag force, which are the two most dominant terms for heavy particles with density ratios $\frac{\rho_p}{\rho_f} \gg 1$ [10]. Electromagnetic, thermal or acoustic forces are also not considered. This leads to the following equation, the *particle force balance equation* [10]:

$$m_p \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \underbrace{\vec{F}_p}_{\text{drag force}} + \underbrace{(\rho_p - \rho_f) V_p \vec{g}}_{\text{gravitational force}}, \tag{2.13}$$

where ρ_p an ρ_f are the particle and fluid density, m_p and V_p are the particle mass and volume, $\vec{v}$ is the particle velocity and $\vec{g}$ is the gravitational acceleration. In its general form, the drag force can be written as follows [10],

$$\vec{F}_p = \frac{1}{2} C_D A_p \rho_f (\vec{u} - \vec{v}) |\vec{u} - \vec{v}| \tag{2.14}$$

$$= \frac{\vec{u} - \vec{v}}{\tau_p} m_p, \tag{2.15}$$

where C_D is the drag coefficient, $A_p = \frac{\pi d_p^2}{4}$ is the particle cross section, and $\vec{u}$ is the fluid velocity. The particle response time

$$\tau_p = \underbrace{\frac{24}{C_D \mathrm{Re}_p}}_{f_D} \frac{\rho_p d_p^2}{18 \rho_f \nu}, \tag{2.16}$$

is a measure for the time it takes a particle to accelerate to the fluid velocity or adapt to changes in the fluid velocity, with f_D being the drag correlation and ν the kinematic viscosity. For very large τ_p, the drag force becomes negligible and particles retain their initial velocity. On the other hand, for small $\tau_p \approx 1$, the particle assumes the fluid velocity within one timestep Δt. For the drag coefficient C_D, different forms exist, depending on the shape and size of the particles and the particle Reynolds number

$$\mathrm{Re}_p = \frac{d_p |\vec{u} - \vec{v}|}{\nu}. \tag{2.17}$$

The latter describes how laminar or turbulent the flow immediately surrounding the particle is. Because of the assumption that our particles are spherical, and for $\mathrm{Re}_p < 1$, we can model the drag force using Stokes law, with the Stokes drag coefficient

$$C_D = \frac{24}{\mathrm{Re}_p}, \tag{2.18}$$

which simplifies the particle response time to

$$\tau_p = \frac{\rho_p d_p^2}{18 \rho_f \nu}. \tag{2.19}$$

Alternative drag coefficients for higher Re_p exist (see e.g. [18]) and will be considered in the future.

2.2.3 Particle-fluid coupling

Particle-laden flows are affected by the interactions between fluid and particles – fluids impacting particles and particles impacting the fluid –, as well as by particle-particle interactions. To reduce the computational cost, we want to model and simulate only those interactions that are relevant to the problem at hand, which depends on how much of the dispersed

Discrete velocities $\vec{c}_i$	Weight ω_i
$(0,0,0)$	$8/27$
$(\pm1,0,0),(0,\pm1,0),(0,0,\pm1)$	$2/27$
$(\pm1,\pm1,0),(0,\pm1,\pm1),(\pm1,0,\pm1)$	$1/54$
$(\pm1,\pm1,\pm1)$	$1/216$

Table 2.1: Lattice weight factors ω_i for the discrete lattice velocities of the D3Q27 velocity set, taken from [9, p.88].

phase is present in the flow. For a simuation with N_p particles being carried by a fluid of volume V_f, this can be determined through the particle volume fraction [10]

$$\Phi_{pv} = \frac{N_p \pi d_p^3/6}{V_f + N_p \pi d_p^3/6}, \tag{2.20}$$

which describes the volume occupied by the particles relative to the total volume available to the flow. For $\Phi_{pv} < 10^{-6}$, only the impact of the fluid on the particles is relevant (one-way coupling). For $10^{-6} \leq \Phi_{pv} \leq 10^{-3}$, the disperse phase is present enough in the flow to have a relevant impact on the flow, and reactive forces of the particles on the fluid need to be taken into account (two-way coupling). For $\Phi_{pv} < 10^{-3}$, particle-particle interactions need to be considered as well (four-way coupling) [10]. For our work, only one-way and two-way coupling is relevant, and we will stick to the corresponding particle volume fractions for our simulations. Four-way coupling can be achieved through coupling to a separate particle solver, and therefore is not included here.

One-way coupling is already achieved through the inclusion of the drag force in (2.13). Particles and fluid could also couple through other external force terms, as well as through mass or temperature exchange, but this is not included here. To model two-way coupling, we simply use Newton's third law, and apply the same but opposite drag force back to the fluid. In our LB equation, this is reflected in the body force $\vec{B}$, which is added as an additional term $\Delta t F_i$ to the right-hand side of (2.3), with

$$F_i = w_i \rho_f \xi_{ij} \frac{B_j}{c_s^2}, \tag{2.21}$$

[19, 10], where w_i are the lattice weight factors (see Table 2.1), ρ_f is the fluid density and B_j is the discretized body force. F_i is applied to the fluid via the exact difference method [20, 9].

3 Numerical methods and implementation

This chapter gives a detailed overview over our implementation of a framework for one-way and two-way coupled particles in ultraFluidX®.

3.1 General setup and data structure of the particle simulation

Particle simulations lend themselves very well to parallelization and in alignment with the ultraFluidX®code, most of our particle code is run on GPUs – a single GPU for the scope of this work –, and is implemented in Nvidia®CUDA® kernels, which are launched from a CPU. A CUDA kernel is simply a function of a specific syntax and structure that will be run on multiple threads by a CUDA-enabled GPU. There are three particle kernels: The *ParticleCollision* kernel, which handles the general fluid-particle coupling, the *ParticleMove* kernel responsible for particle motion and boundary conditions, and the *ParticleBackCoupling* kernel dedicated solely to the redistribution of the back coupling body force onto the fluid voxels. The former two are parallelized by particles, i.e. each thread calculates the forces on and motion of a single particle. Only the back coupling kernel is parallelized by fluid voxels to avoid race conditions when distributing the back-coupling force onto the fluid voxels. Particle data is handled in simple arrays. Except for initialization, these live almost exclusively on the GPU. Data is copied back to the CPU only for the purpose of writing output, which is done in *.csv* format. The number of particles within the simulation is fixed. This is handled via a particle 'status' array, which allows particles to be activated upon spawning and deactivated upon leaving the simulation domain. Particles stuck to a wall obtain a third distinct status, see Section 3.5.2.

The general workflow of the particle-laden simulation is shown in Figure 3.1.

3.2 Initialization and user interface

We initialize our particles in particle zones. For all particle zones, there are two options for initial position, velocity and release period each, which all can be combined with each other arbitrarily. A simulation can contain several particle zones with different initialization settings. Particles within a particle zone all share the same density ρ_p and diameter d_p.

For the **position**, particles are initialized uniformly either within a box or at the inlet (see 3.5.2). Initial **velocities** can either be chosen by the user or interpolated from the fluid velocity. Particles within a zone can either be released simultaneously or over a chosen **release period**. In order to enable this last option, we build a list containing a random float $r_p \in [0,1]$ per particle. A particle spawns once its spawn probability p reaches this random number.

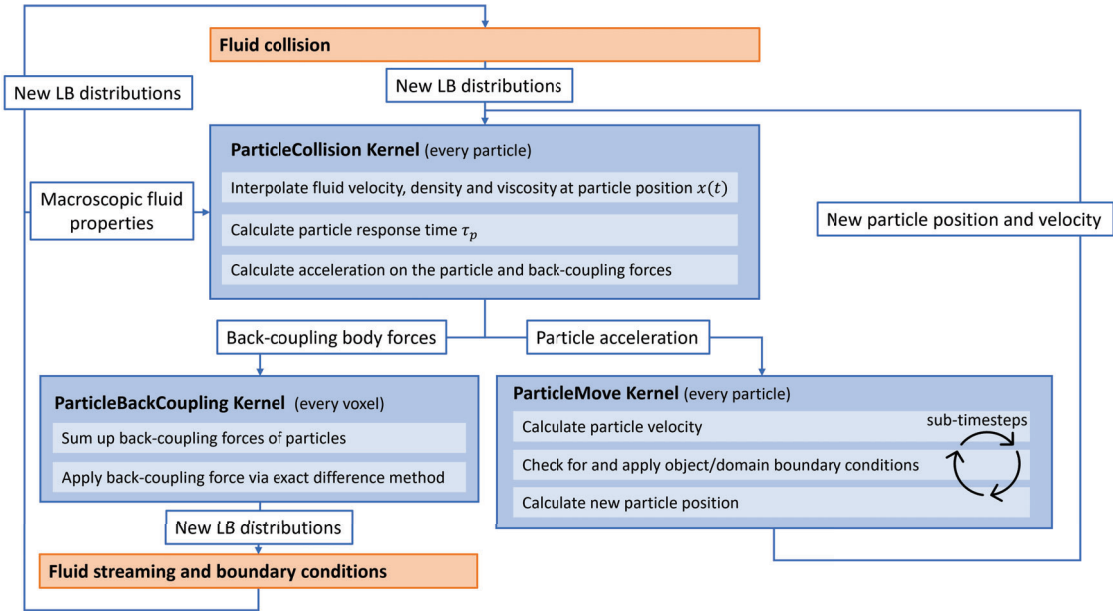

Figure 3.1: The general workflow of the coupled fluid-particle simulation. The three particle kernels are executed after the LB collision step but before streaming.

The spawn probability is calculated based on the start and end iterations $\mathrm{it}_{\mathrm{start}}$ and $\mathrm{it}_{\mathrm{end}}$ of the release period of its designated zone,

$$p(\mathrm{it}_{\mathrm{current}}) = \frac{(\mathrm{it}_{\mathrm{current}} - \mathrm{it}_{\mathrm{start}} + 1)}{(\mathrm{it}_{\mathrm{end}} - \mathrm{it}_{\mathrm{start}} + 1)}. \tag{3.1}$$

3.3 Time stepping

The LBM uses a constant timestep, which is set to $\Delta t = 1$ in LB units (see Section 2.1.5). We keep this same fixed timestep for the particle simulation and use an explicit Euler scheme for discrete integration of the acceleration. Sub-timestepping is used in the ParticleMove kernel to check for objects and boundaries, which will be explained in more detail in Section 3.5.2, but it is purely spatial in nature, and particle-fluid coupling is only carried out once, in the ParticleCollision kernel. This could lead to stability issues, as high-velocity particles could move across many voxels within one timestep, applying a high back-coupling force on only a few voxels along its trajectory. Apart from velocity limits, alternative timestepping and time integration schemes could therefore be considered in the future.

3.4 Fluid-particle coupling

Looking back at the drag force in Equation 2.15 as well as Section 2.2.3, the two-way coupling between fluid and particles requires three distinct steps: Locating the particles within

the fluid grid, which is done by a simple snapping algorithm, interpolating the fluid velocity to calculate the drag force, and redistributing the reactive drag force back onto the fluid voxels. Snapping to fluid nodes based on the particle position is very simple given that we are working on a uniform cubic grid. We will describe the other two steps in the following sections.

3.4.1 Interpolation of the fluid velocity

For interpolating the fluid velocity, many different schemes exist [21]. We use the standard **trilinear interpolation** method [22], visualized in Figure 3.2. A particle is situated within a cube made up of the 8 surrounding nodes (voxel centers), with lower corner (x_0, y_0, z_0) and upper corner (x_1, y_1, z_1). The particle's relative position within the cube is then defined as

$$x_d = \frac{x - x_0}{x_1 - x_0}, \qquad y_d,\ z_d \text{ analogously.} \tag{3.2}$$

From the node values $c_{ijk} = v(x_i, y_j, z_k)$ with $i, j, k \in \{0, 1\}$, shown in blue, we do simple linear interpolations with inverse distance weighting, one dimension after the other, first in x-direction to calculate the edge values (in green)

$$c_{jk} = c_{0jk}(1 - x_d) + c_{1jk}x_d \quad \forall\, j, k \in 0, 1, \tag{3.3}$$

then in y-direction to calculate the top and bottom face values (orange)

$$c_k = c_{0k}(1 - y_d) + c_{1k}y_d \quad \forall\, k \in 0, 1\,, \tag{3.4}$$

and finally in z-direction to calculate the final interpolated value at the particle position (black)

$$c = c_0(1 - z_d) + c_1 z_d\,. \tag{3.5}$$

This corresponds a sum over all the corner node values c_{ijk}, each weighted by the volume of the diagonally opposing corner of the cube (shown in gray in Figure 3.2), i.e.

$$c = \sum_{ijk} c_{ijk} \prod_{d=x,y,z} (d_1 - d_0) - |d - d_i|\,. \tag{3.6}$$

If no eight fluid nodes can be found, we use the nearest neighbor interpolation and the velocity of the fluid voxel in which the particle is currently situated as a fallback.

3.4.2 Back-coupling

For the back-coupling, we need to calculate the body force acting on a voxel due to nearby particles and apply it to the populations, which is visualized in Figure 3.3. We do this in several steps.

In the first step, each particle calculates the total drag force it exerts upon the surrounding fluid, which is simply the inverse of the drag force it itself experiences,

$$\vec{F}_D^{\,p} = -m_p \frac{\vec{u} - \vec{v}}{\tau_p}\,. \tag{3.7}$$

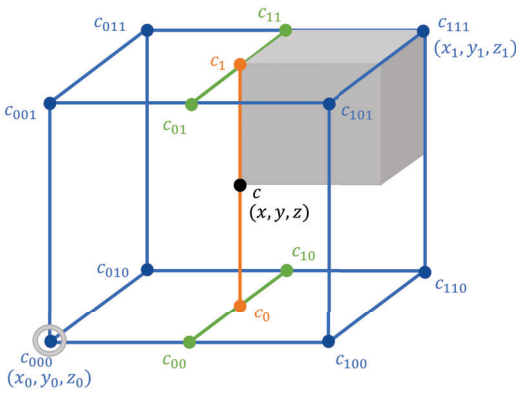

Figure 3.2: Visualization of the trilinear interpolation. In gray, the total weight of c_{000} is shown, which is equivalent to the volume of the cuboid at the diagonally opposite corner.

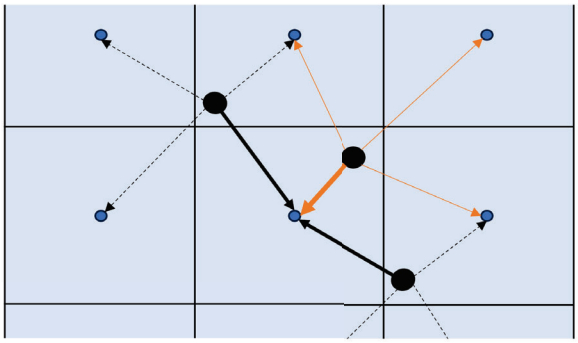

Figure 3.3: 2D visualization of the distribution of the back coupling force from particles onto the fluid nodes. One voxel can be affected by several particles. In our 3D simulation, 8 nodes are affected instead of the 4 shown here.

In the second step, the particle redistributes this total drag force onto the surrounding fluid nodes. Many different possible methods exist for this. In our implementation, we always distribute onto the nodes that contributed to the interpolated fluid velocity (i.e. one or eight), either uniformly as done in [10] with weights $w_n^p = \frac{1}{8}$ for the eight-node and $w_n^p = 1$ for the one-node case, or using inverse-distance scaling, with weights

$$w_n^p = \frac{1}{\sum_n w_n^p} \frac{1}{\left\| \vec{x}_p - \vec{x}_f \right\|}, \tag{3.8}$$

where $\vec{x}_p$ is the particle position position and $\vec{x}_f$ is the position of the fluid node. Weights are normalized over the total sum of weights $\sum_n w_n^p$ for the affected nodes n for each particle. To avoid zero-division, $\left\| \vec{x}_p - \vec{x}_f \right\|$ is given a minimum cutoff value.

Consequently, the particles then write the weighted force components

$$\vec{F}_n^p = \vec{F}_D^p \cdot w_n^p \tag{3.9}$$

to dedicated lists of length $8 \cdot N_p$, where N_p is the total number of particles. There are five lists containing the three force components, as well as two values to identify the voxels. We use this to avoid race conditions, as one fluid node could be affected by several particles.

In the third step, in a dedicated back-coupling CUDA kernel, each fluid node iterates over these lists of force components and sums up all the force components that affect it,

$$\vec{F}_n = \sum_p \vec{F}_n^p . \tag{3.10}$$

It then converts the result into a velocity difference

$$\Delta \vec{u} = \frac{\vec{F}_n}{m_f} \Delta t , \quad \text{with} \quad m_f = \rho_f L_0^3 . \tag{3.11}$$

In the fourth step, the $\Delta \vec{u}$ are applied to the fluid distributions via the exact difference method, by modifying the equilibrium distribution: First, the current equilibrium distribution distributions are calculated from the fluid velocity $\vec{u}$ and fluid density ρ_f,

$$f_i^{\text{eq}} = \omega_i \rho_f \left[1 + \frac{\vec{c}_i \cdot \vec{u}}{2c_s^2} + \frac{\vec{c}_i \cdot \vec{u}^2}{2c_s^4} \right], \tag{3.12}$$

which is the discretization of (2.1) derived in [23]. The equilibrium distributions are then subtracted from the current LB distributions $\tilde{f}_i = f_i - f_i^{\text{eq}}$, leaving only the non-equilibrium part which we want to conserve. We then recalculate the equilibrium distributions $\tilde{f}_i^{\text{eq}}$ while adding our velocity differences that have been calculated based on the body force to the current fluid velocity $\tilde{\vec{u}} = \vec{u} + \Delta \vec{u}$. These new equilibrium distributions are then added back to the non-equilibrium part, $f_i = \tilde{f}_i + \tilde{f}_i^{\text{eq}}$.

3.5 Particle motion

3.5.1 Discrete equations of motion

The implementation of the particle motion follows the equations from Section 2.2.2. The acceleration for each individual particle can be directly adopted as

$$\vec{a} = \frac{\vec{u} - \vec{v}_{\text{old}}}{\tau_p} + \frac{(\rho_p - \rho_f)}{\rho_p}\vec{g}, \tag{3.13}$$

with τ according to (2.19). Fluid density ρ_f, velocity $\vec{u}$, and kinematic viscosity ν are all interpolated via trilinear interpolation as described in Section 3.4.1.

For the particle velocity, we use an explicit Euler scheme

$$\vec{v}_{\text{new}} = \vec{v}_{\text{old}} + \vec{a} \cdot \Delta t, \tag{3.14}$$

and for the particle position, a second order Taylor approximation

$$\vec{x}_{\text{new}} = \vec{x}_{\text{old}} + \vec{v} \cdot \Delta t + \frac{\vec{a}}{2}\Delta t^2, \tag{3.15}$$

corresponding to the methods used in [10].

3.5.2 Boundary conditions

Particle motion algorithm

Our implementation includes periodic, inflow, outflow, and collision boundary conditions. The particle boundary condition algorithm as implemented in the ParticleMove CUDA kernel is shown in Figure 3.4. Because there is no general limit to particle velocity, sub-timestepping is used to ensure wall detection. The new particle velocity $\vec{v}$ is calculated once, based on Equation (3.14), and is only ever modified in the case of collision, no additional coupling to the fluid occurs during the sub-timesteps.

Based on the individual maximum possible velocity $\vec{v}_{\max}$ for each particle within a certain timestep as calculated via its velocity, acceleration, and the grid size L_0, we determine a safe sub-timestep size

$$\Delta t_{\text{sub}} = \min\left(\Delta t, \frac{L_0}{v_{\max}}, t + \Delta t - t_{\text{sub}}\right), \tag{3.16}$$

which will be used as a sub-timestep unless it is reset due to a particle arriving at an object or wall. The sum of sub-timesteps always eventually reaches the fluid timestep Δt, which ends the current particle motion cycle.

In each sub-timestep, we calculate the new theoretical particle position $x(t^{\text{new}})$ via (3.15) and identify the nearest node around the old particle position $x(t^{\text{old}})$ by snapping to it. Nodes can be fluid, boundary or solid nodes, as visualized in Figure 3.5. In the two latter cases, we check for domain and object boundary conditions according to the Algorithm shown in Figure 3.4.

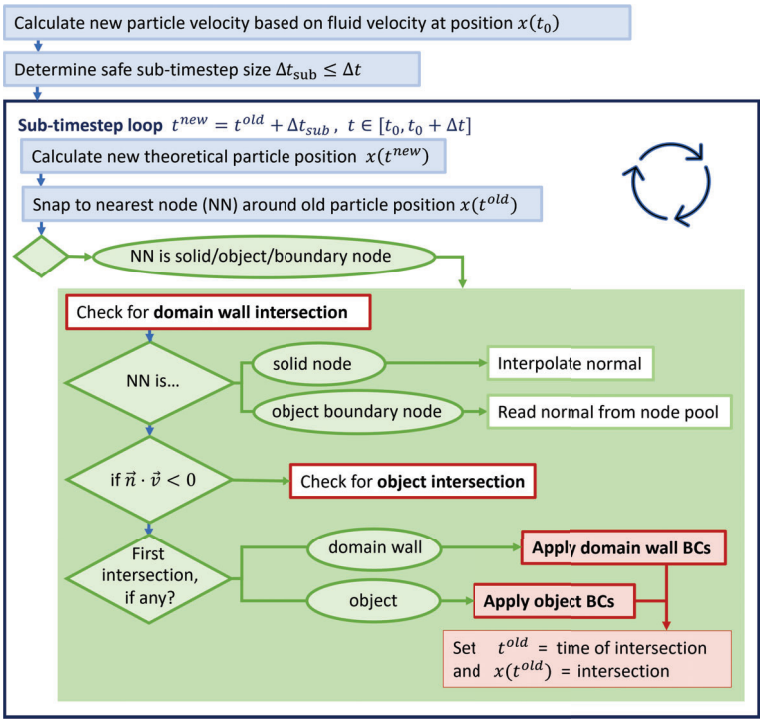

Figure 3.4: General algorithm for the particle boundary conditions.

For domain boundary conditions, we simply check whether $x(t^{\text{new}})$ lies outside of one of the domain boundaries, which are known. If this is the case, we calculate the intersection between the parabolic particle trajectory and the corresponding domain wall using basic geometry: A plane with normal $\vec{n}$ containing a random point $\vec{m}$ can be defined via

$$\langle (\vec{x} - \vec{m}), \vec{n} \rangle = 0 \tag{3.17}$$

for all points $\vec{x}$ on the plane. By substituting $\vec{x}$ via (3.15), we receive a quadratic equation for the time of intersection t_{wall},

$$t_{\text{wall}}^2 + t_{\text{wall}} \cdot 2\frac{\langle \vec{v}, \vec{n} \rangle}{\langle \vec{a}, \vec{n} \rangle} + 2\frac{\langle \vec{x}_{\text{old}} - \vec{m}, \vec{n} \rangle}{\langle \vec{a}, \vec{n} \rangle} = 0\,, \quad t_{\text{wall}} \in [0, \Delta t] \tag{3.18}$$

of which we use the lowest positive solution as our result. In case the acceleration $\vec{a}$ falls below a certain threshold, we simply do a linear approximation instead, with

$$t_{\text{wall}} = \frac{\langle \vec{m} - \vec{x}_{\text{old}}, \vec{n} \rangle}{\langle \vec{v}, \vec{n} \rangle}\,. \tag{3.19}$$

We currently neglect cases in which, due to normal acceleration and velocity having opposite signs, the patricle leaves and reenters the simulation domain within one sub-timestep. Inserting the solution back into (3.15) returns the point of intersection, i.e. the point where the

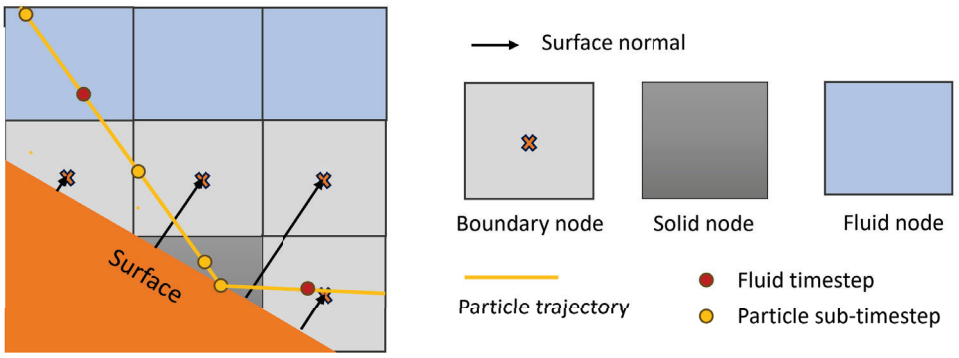

Figure 3.5: Schematic visualization of object boundary conditions. Surface normals for each boundary node are shown by black arrows, particle trajectories are represented by the yellow line. Particles couple to the fluid only once per fluid timestep (at the red dots), but move in subtimesteps (yellow dots) along the trajectory to check for wall intersections. If an intersection is found, the subtimestep is shortened and the next subtimestep starts from the surface.

particle impacts the wall. If there are several such intersections with different walls (e.g. in a domain corner), the earliest interaction is the relevant one.

For object boundaries, the process is similar, but with some significant differences. Upon calculating the new particle position in a sub-timestep, we do not inherently know whether or not we have entered or even traversed an object in the process. However, because of our sub-timestepping, we can be sure to always enter a boundary node before hitting any object. Additionally, each boundary node carries its distance vector from the surface, providing us with both a local surface normal $\vec{n}$ and an achor point on the surface $\vec{m} = \vec{x}_{\text{node}} - \vec{n}$. By locally approximating the surface as a plane, we can thus follow the same steps as above to calculate the intersection time and position between the particle trajectory and the object.

To reduce the computational effort, we only check for object intersection in case

$$\vec{n} \cdot \vec{v} < 0\,, \tag{3.20}$$

i.e. we only check surfaces that we are not moving away from. This increases efficiency, but neglects the possibility of particles 'turning around' during one sub-timestep due to strong acceleration.

Particles may also find themselves within solid nodes, which do not have normal information available. In this case, we choose the nearest normal (if any) amongst the adjacent nodes which fulfills (3.20). Note that particles within a solid node do not couple to the fluid. Their acceleration is zero and they do not apply any back-coupling force to the fluid. Of course, particles within a solid node may also have simply glitched into the object due to the imperfect surface approximation. They are therefore deactivated once they are no longer adjacent to any boundary nodes.

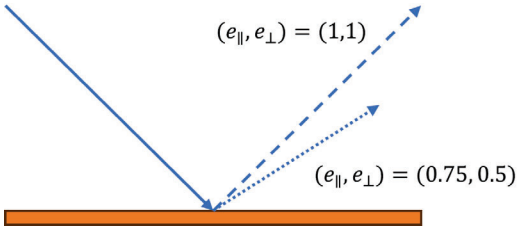

Figure 3.6: Visualization of the collision boundary condition for different tangential and normal coefficients of restitution $(e_\parallel, e_\perp)$.

After domain and object intersections have been calculated as necessary, we verify whether both occur in the same subtimestep, in which only the earlier impact actually occurs. We then apply the actual corresponding boundary condition, place our particle onto the point of intersection and restart the sub-timestep loop from the moment of intersection.

The only exception to this are periodic boundary conditions: Here, x_{old} is set to the opposing side of the domain, from where the particle will begin its next sub-timestep.

Collision boundary conditions

Collision boundary conditions, which are available for objects and domain walls, follow a simple principle: If a particle collides with a surface, the velocity component normal to the surface changes sign. Beyond that, we control our collision behavior via two parameters, the tangential and normal coefficients of restitution $e_\parallel$ and $e_\perp$, see e.g. [24, 25]. With values ranging between 0 and 1, the coefficients of resitution represent the fractions of the tangential or normal velocity that are retained by the particle upon collision with the surface. The missing fraction models dissipation losses upon impact.

This simple model captures two important edge cases: For $e_\perp = 0$, particles stick to the surface upon impact. In our implementation, this corresponds to a special particle status in which particles remain stationary and no longer undergo the particle motion algorithm, but still affect the fluid through back-coupling. For $(e_\perp, e_\parallel) = (1,1)$, on the other hand, the particle undergoes elastic collision. There exists a wide range of literature on the measurement and estimation of COR values for a different materials, particles and surfaces, see e.g. [26, 27, 25]. Amongst other factors, they are dependent on the dynamic yield strength, elastic modulus and density of the materials involved, as well as the impact velocity [26]. Generally, $e_\parallel \geq e_\perp$.

This surface model remains, of course, quite simplistic. The actual volume and center position of the particles is not considered. No rotational effects are considered, and no sliding on the surface is allowed. Surface roughness, as well as deformation and thermal effects are also disregarded. Some of these approximations, such as surface roughness or wall distance upon impact could be improved by model extensions, whereas others might require larger changes to the framework.

Inflow boundary conditions

In addition to the particles spawning at a random iteration during their zone-specific release interval (see Section 3.2), we also randomize the starting position by already advancing t^{old} of the sub-timestep loop to a random point $t_0 + \Delta t_{\text{fluid}} \cdot r$, where $r \in [0, 1]$ is a uniformly sampled float. To reduce the memory requirements, we reuse the same list of random floats per particle used in Section 3.2, and simply shift the particle index to avoid correlations. In case of velocity interpolation initialization, particles perform a nearest neighbour interpolation on the first layer of boundary nodes.

4 Results and validation

In this chapter, we test the different components of our implementation with a variety of test cases. We validate the velocity interpolation, gravitational acceleration, back-coupling and boundary conditions, and test the computation time and performance.

4.1 Poiseuille flow case - Interpolation error

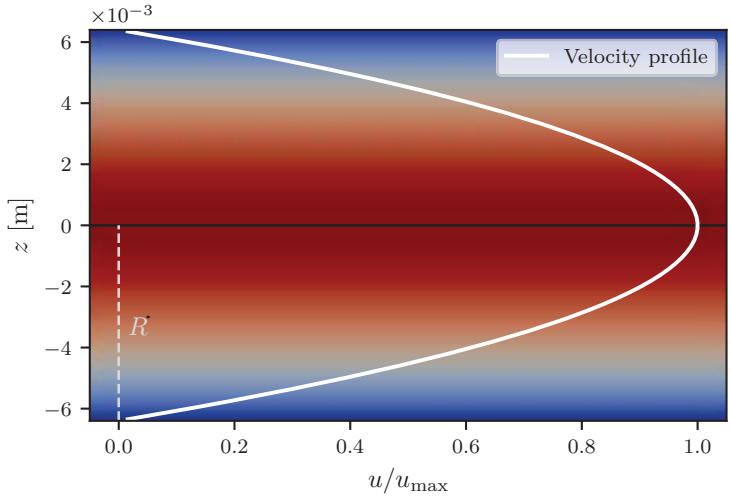

Figure 4.1: Velocity profile of a planar Poiseuille flow as a line plot and x-z - colormap.

To measure the error of the fluid velocity interpolation, we use a one-way coupled simulation. By using a fixed $\tau_p = 1$ and turning off the gravity, particles always move with the interpolated fluid velocity. In order to validate the interpolation algorithm, we perform a Poiseuille flow simulation where the analytical profile is known. The velocity profile of a laminar flow between two infinite planes is parabolic can be described as

$$u(r) = u_{\max}\left(1 - \left(\frac{r}{R}\right)^2\right), \tag{4.1}$$

where in this case, the radial coordinate $r = |z|$, R is half the channel height and $u_{\max}$ is the maximum velocity, which is found at the center of the channel. The flow profile of such a planar **Poiseuille flow** is visualized in Figure 4.1. For the derivation from the Navier-Stokes equations, see e.g. [28, p.34].

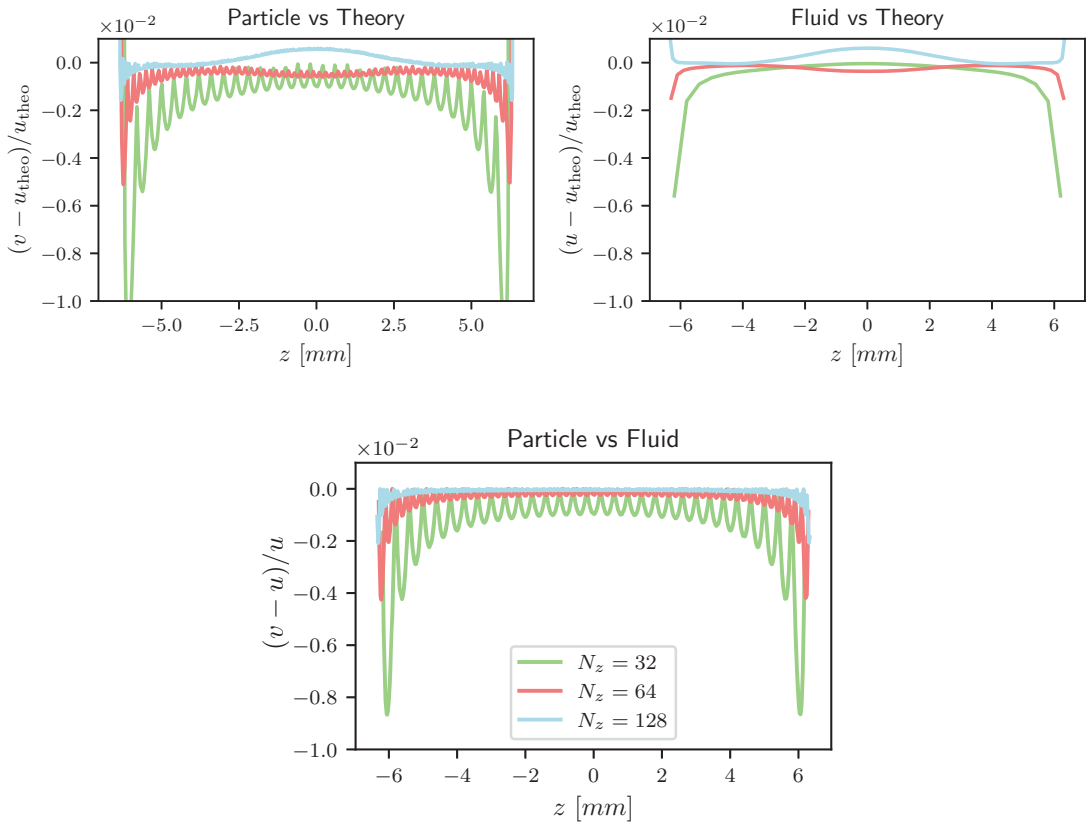

Figure 4.2: The relative interpolation error for the converged flow for different mesh sizes, comparing (upper left) particle velocities and theoretical Poiseuille profile, (upper right) fluid velocity and theoretical Poiseuille profile, and (lower center) particle and interpolated fluid velocities. N_z represents the number of voxels in z-direction.

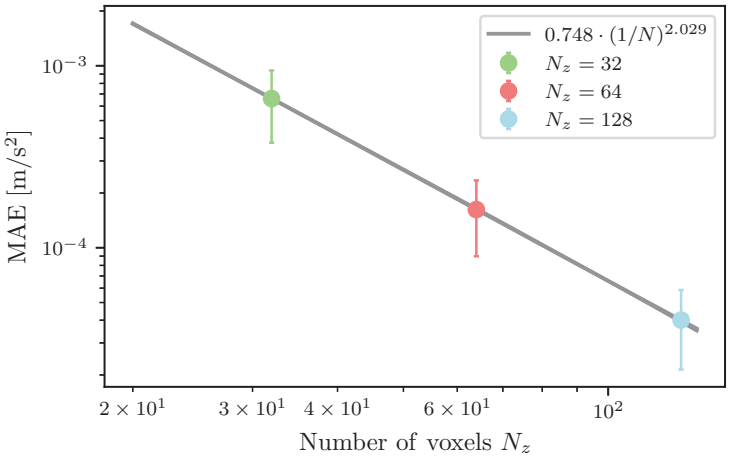

Figure 4.3: The mean absolute error of the particle velocity for different mesh sizes with the fit function $f(N_z)$ in gray. Error bars indicate the standard deviation of the absolute error from the MAE.

For the case setup, we use periodic boundary conditions in x and y direction with no-slip boundary conditions for the z walls. The flow is accelerated by a volume acceleration per area in x of 0.738m/s^2. The periodic boundary conditions allow for a setup with a relatively small domain size while the no-slip boundary conditions are necessary to develop the Poiseuille profile. 1000 particles are used, positioned randomly within a y-z plane. To reduce the necessary number of iterations, we seed our flow with a hard-coded Poiseuille profile before letting it converge. The flow profile produced differs very slightly from the theoretical Poiseuille profile, as can be seen in Figure 4.2. To be able to measure the actual interpolation error of the particles, we require a parabolic profile that accounts for the fluid error - the actual fluid profile is not sufficient as it is only available at the nodes, where particle velocities converge towards the fluid velocity. We achieve this, by using a quadratic interpolation between the fluid nodes, thus obtaining pseudo-fluid velocities at the particle positions. Alternatively, one could also compare the particle velocities to the theoretical Poiseuille profile immediately upon initialization (when the fluid flow has just been seeded with the exact Poiseuille profile). Both approaches lead to the same results.

The relative interpolation error for different mesh sizes is shown in Figure 4.2. Due to the concave nature of the Poiseuille profile, the trilinear interpolation, being linear, always underestimates the velocity, meaning that the signed error is always ≤ 0. At lower velocities, the relative error e_{rel} becomes larger, and tends towards infinity at the edges: The absolute error $e_{\text{abs}} = |v - u|$ is capped at the velocity value of the boundary voxel $u(|R - \frac{L_0}{2}|) > 0$, whereas $u_{\text{theo}}(R) = 0$. e_{rel} oscillates due between high error between nodes and $e_{\text{rel}} \to 0$ when particles get very close to fluid nodes.

The trilinear interpolation scheme has second-order accuracy due to being based on first-

order polynomials. We confirm this in Figure 4.3, by fitting the mean absolute error

$$\text{MAE} = \frac{1}{N_p} \sum_{i=0}^{N_p} |v - u|, \tag{4.2}$$

over all particles N_p within $[-R + L_0/2, R - L_0/2]$ with a function

$$f(N_z) = a + bN_z^{-c} \tag{4.3}$$

$$= 0.748 N_z^{-2.029}. \tag{4.4}$$

Particles within $L_0/2$ of the domain boundaries, where trilinear interpolation is replaced by a nearest-neighbor interpolation, are not included in this calculation.

4.2 Terminal velocity case - Gravity

To test our implementation of drag force and gravity, we let a single particle accelerate in a stationary fluid with periodic boundary conditions until its velocity converges and compare with the analytical solution for the terminal velocity. The balance between gravitational and drag forces

$$\vec{F}_G + \vec{F}_D = 0 \tag{4.5}$$

for spherical particles leads to a terminal velocity

$$v_t = \pm \sqrt{\frac{4 g d_p}{3 C_d} \frac{|\rho_p - \rho_f|}{\rho_f}}, \tag{4.6}$$

with $v_t > 0$ for $\rho_p < \rho_f$ and $v_t < 0$ for $\rho_p > \rho_f$, where we assume $\vec{g} \parallel \vec{e}_z$, and standard gravity $g \equiv g_z = -9.80665 \frac{\text{m}}{\text{s}^2}$. For Stokes drag, we can insert (2.18) and (2.17) with fluid velocity $u = 0$, and get

$$v_t = \frac{g d_p^2}{18 \mu} (\rho_p - \rho_f). \tag{4.7}$$

Using the parameters from Table 4.1, the resulting theoretical terminal velocity for Stokes drag is $v_t = -0.0090141$. In Figure 4.4, we can see that our simulated particle converges towards this value within about 25 milliseconds.

L_0 [m]	(N_x, N_y, N_z)	$\rho_f \left[\frac{\text{kg}}{\text{m}^3}\right]$	$\mu \left[\frac{\text{kg}}{\text{m·s}}\right]$	d_p [m]	$\rho_p \left[\frac{\text{kg}}{\text{m}^3}\right]$	C_D
0.005	(8,8,8)	1.2041	$1.8194e^{-5}$	2.4082	0.0005	Stokes

Table 4.1: Selected parameters for the terminal velocity validation case.

.

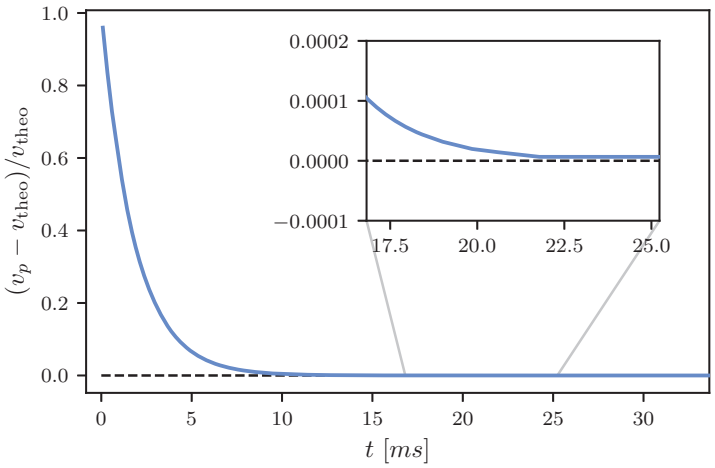

Figure 4.4: Particle velocity relative to the theoretical terminal velocity.

4.3 Sphere case - Back-coupling

In order to validate the back-coupling force, we compare the effects of the particle drag on the fluid to the drag force of an actual solid sphere modeled as a surface and simulated via boundary conditions within the flow. For a sphere and particle of fixed diameter $d_p = d_s = 0.016\,\text{m}$, we compare the flow behavior and drag force for different mesh sizes, as well as comparing to simulations where multiple smaller particles approximate the sphere. For the purpose of this validation case, we go beyond the usual physical modeling limit for point particles, i.e., particles are no longer smaller than the mesh size, as otherwise, the solid sphere could no longer be effectively modeled. The test cases have been chosen based on the current implementation of the back-coupling force, which always acts on either one or eight surrounding voxels.

We compare five particle test cases, all of which are visualized in Figure 4.5, with parameters listed in Table 4.2, and three sphere cases with fixed diameter and varying mesh sizes. Based on the mesh sizes, it makes sense to compare Sphere Case I to Particle Case A), Sphere Case II to Particle Case B), and Sphere Case III to Particle Cases C), D), and E). We use laminar flow with Re = 10, inlet and outlet boundary conditions in x direction, with an inlet velocity $u_{\text{inlet}} = 1.0\text{m/s}$, as well as periodic boundary conditions in y and z. All parameters beyond particle/sphere radius and mesh size are kept constant, the exception being that the particle/sphere position is shifted slightly for Case A)/I to place it at the voxel center. We consider two different types of results: flow behaviour and total drag force.

Flow behaviour is shown in Figure 4.6 by visualizing the fluid velocity. Comparing cases I and A), II and B), and III and C) with single particles, we see that generally, particles show a similar behavior to the spheres in that they decelerate the flow in the center, leading to slower fluid tail and an acceleration of the flow at the sides. However, we also see that single point-particles cannot adequately imitate the same wake effect that a solid sphere creates.

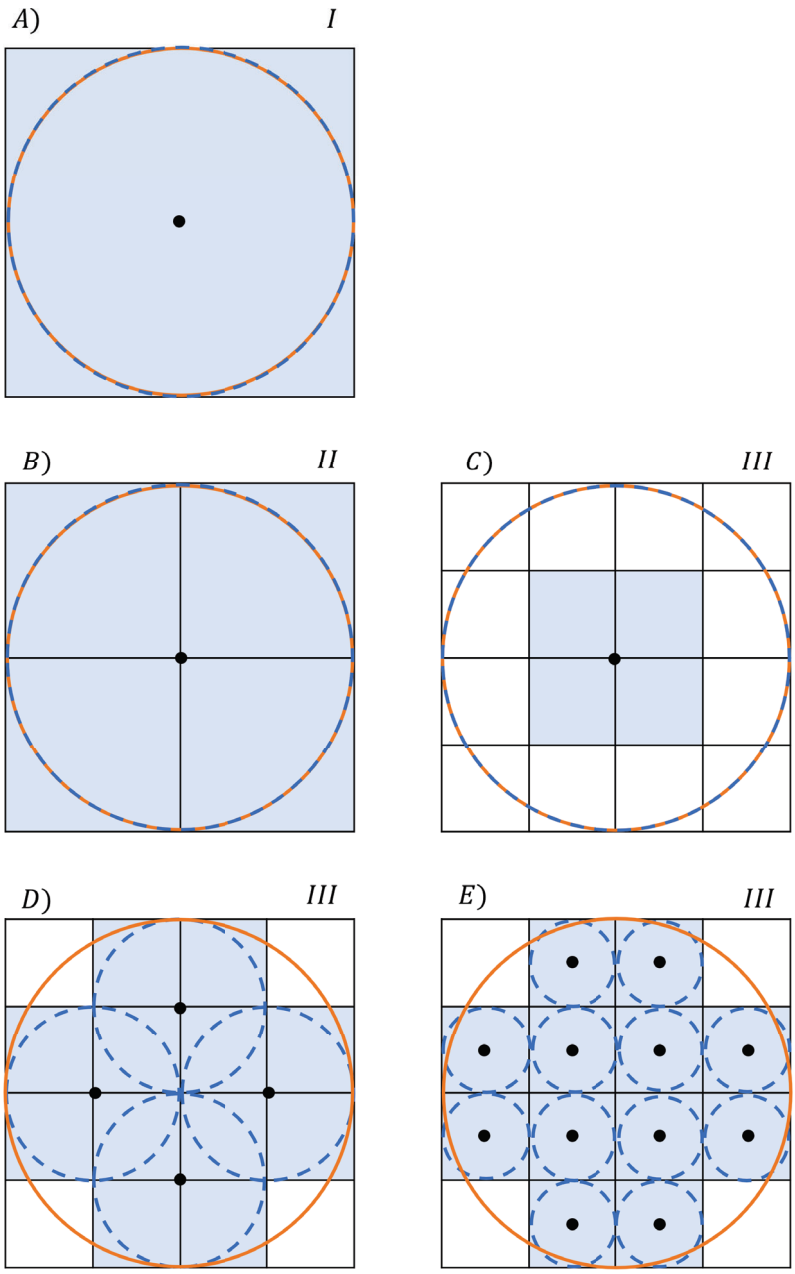

Figure 4.5: 2D-visualization of the five particle test cases A)-E) and three sphere test cases I-III. Black dots represent the center of the particle, voxels shaded in blue are subject to a back-coupling force. Blue circles represent the particle surface, orange circles the sphere's surface.

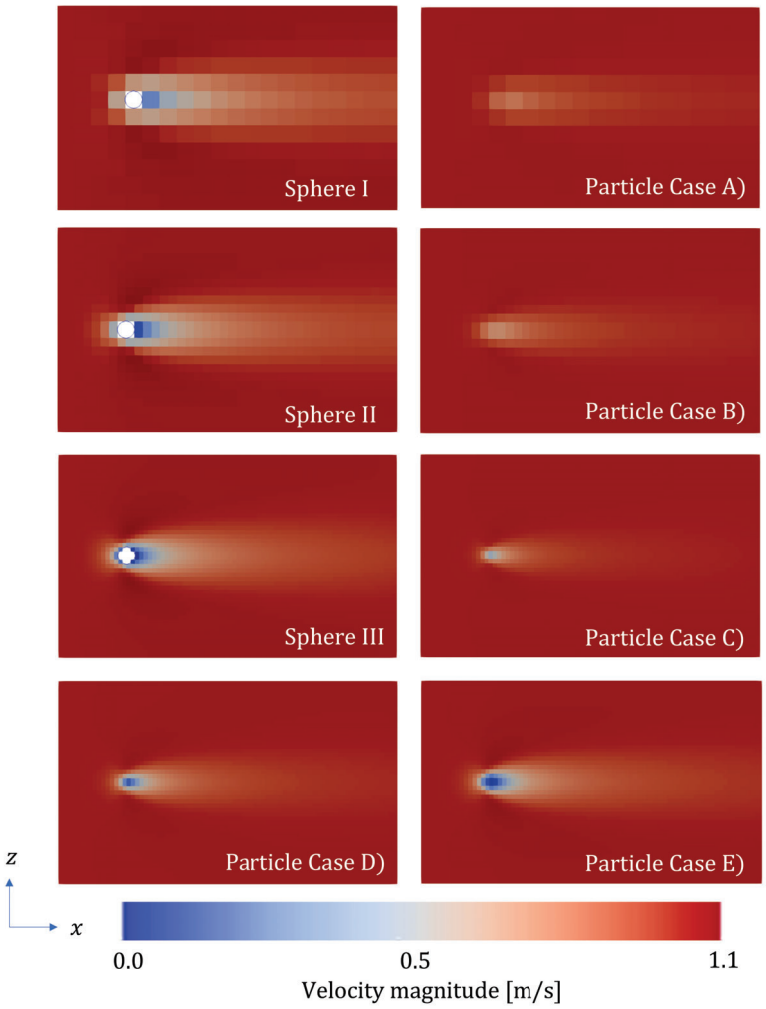

Figure 4.6: Velocity magnitude for particle and sphere cases.

	L_0 [m]	d_p [m]	N_p	$N_p \cdot d_p$	$\frac{0.016}{N_p \cdot d_p}$
A)	0.016	0.016	1	0.016	1
B)	0.008	0.016	1	0.016	1
C)	0.004	0.016	1	0.016	1
D)	0.004	0.008	6	0.048	$\frac{1}{3}$
E)	0.004	0.004	32	0.128	$\frac{1}{8}$
I)	0.016				
II)	0.008				
III)	0.004				

Table 4.2: Parameters for the different particle cases. The sphere diameter is fixed at d_s = 0.016 m. The velocity of the inlet is u_x = 1.0 m/s. The scaling factor $\frac{0.016}{N_p \cdot d_p}$ is used to ensure that $N_p \cdot d_p$ = const, in which case drag forces should theoretically also remain constant.

.

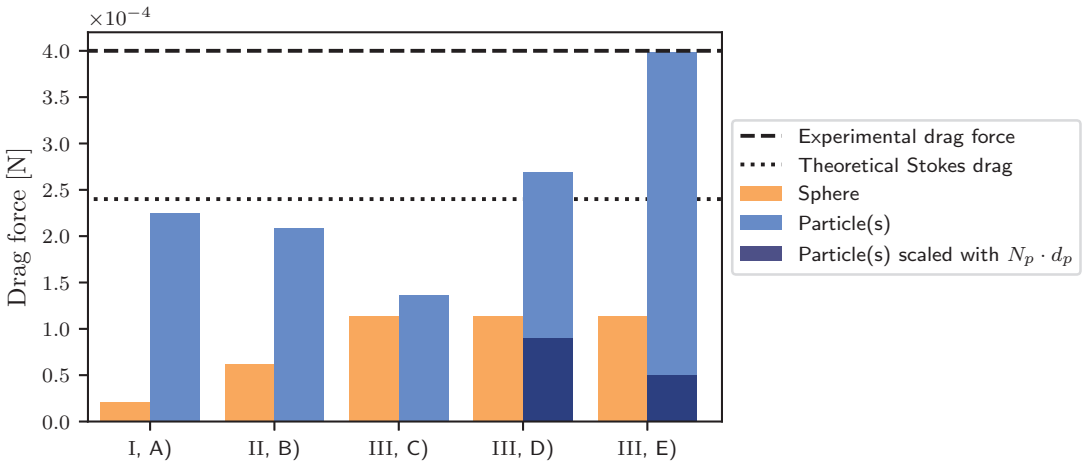

Figure 4.7: Drag forces for sphere and particle cases, as well as theoretical drag forces for the experimental drag coefficient and the Stokes drag coefficient at Re = 10.

This is expected: As particles do not block the flow, the drag force acts at completely different positions - at the center instead of the surface. By effectively discretizing the particle into 6 or 32 smaller point-particle spheres in cases D) and E) respectively, we show that a more spread-out application of the back-coupling force does indeed lead to very similar profile to the Sphere III case, and we can approximate the behavior very effectively.

In Figure 4.7, we compare the drag forces. For the single-particle cases A), B) and C), we see that the particle drag force approaches the sphere's drag force with better discretization. For the smallest mesh size in Case C)/III, the drag forces are very close. For cases D) and E) with 6 and 32 particles, the drag forces are far larger than the sphere's drag forces; however, this is to be expected, as $\vec{F}_p \propto N_p \cdot d_p$, which is not constant among cases C), D), and E). By scaling the drag forces in cases D) and E) with 1/3 and 1/8 to maintain $N_p \cdot d_p = \text{const.}$, we see that the effective drag force is getting closer to the sphere's drag force, and is actually smaller than for the single-particle case. This is due to particles being clustered up instead of individually interacting with the flow, meaning that the particles interact with flow that has already been decelerated by other particles, which reduces the velocity difference and thus also the drag force.

Comparing with experimental data for the drag coefficients of smooths spheres [29] for Re = 10, as well as the theoretical Stokes drag force, see Figure 4.7, we can also confirm that we are in the correct order of magnitude. Notably, Case A) corresponds very well to the Stokes drag, as is to be expected for the 1-voxel case. Better discretization and alternative drag coefficients could improve results even further.

4.4 Boundary conditions

For testing boundary conditions, we use particle simulations that have effectively been decoupled from the fluid simulation by setting a very large fixed τ_p and initializing with a fixed initial velocity, so that particles retain their initial velocity throughout the simulation unless collision occurs. This makes the flow field irrelevant, and we focus solely on the particle trajectories.

For object boundary conditions, we use two different test cases, with either a cube turned at a 45° angle with respect to the domain walls and initial particle trajectories, or a sphere. Both serve to test the general object collision boundaries. The results for both cases with different coefficients of restitution $e = (e_\perp, e_\parallel)$ are shown in Figure 4.8, confirming the desired behavior for elastic ($e = (1,1)$), stick ($e = (0,0)$), and mixed (here $e = (0.5, 0.7)$) collision boundary conditions: For elastic boundary conditions, incoming and outgoing angles are the same. This can be observed most distinctly in the cubic case, where particle trajectories show a right angle due to impacting the surface at 45°. For mixed boundary conditions, the velocity of the particle decreases (shown in the shortened trajectory in Figure 3.6), and the angle of reflection changes due to the changing ratio of the velocity components. In stick cases, particles stop at the object surface. Note that especially for non-planar surfaces, it can still occur that particles glitch through walls due to the inaccuracy of the surface approximation, in which case, they are deactivated.

For domain boundary conditions, we do a simple check for all cases: A) inflow and outflow

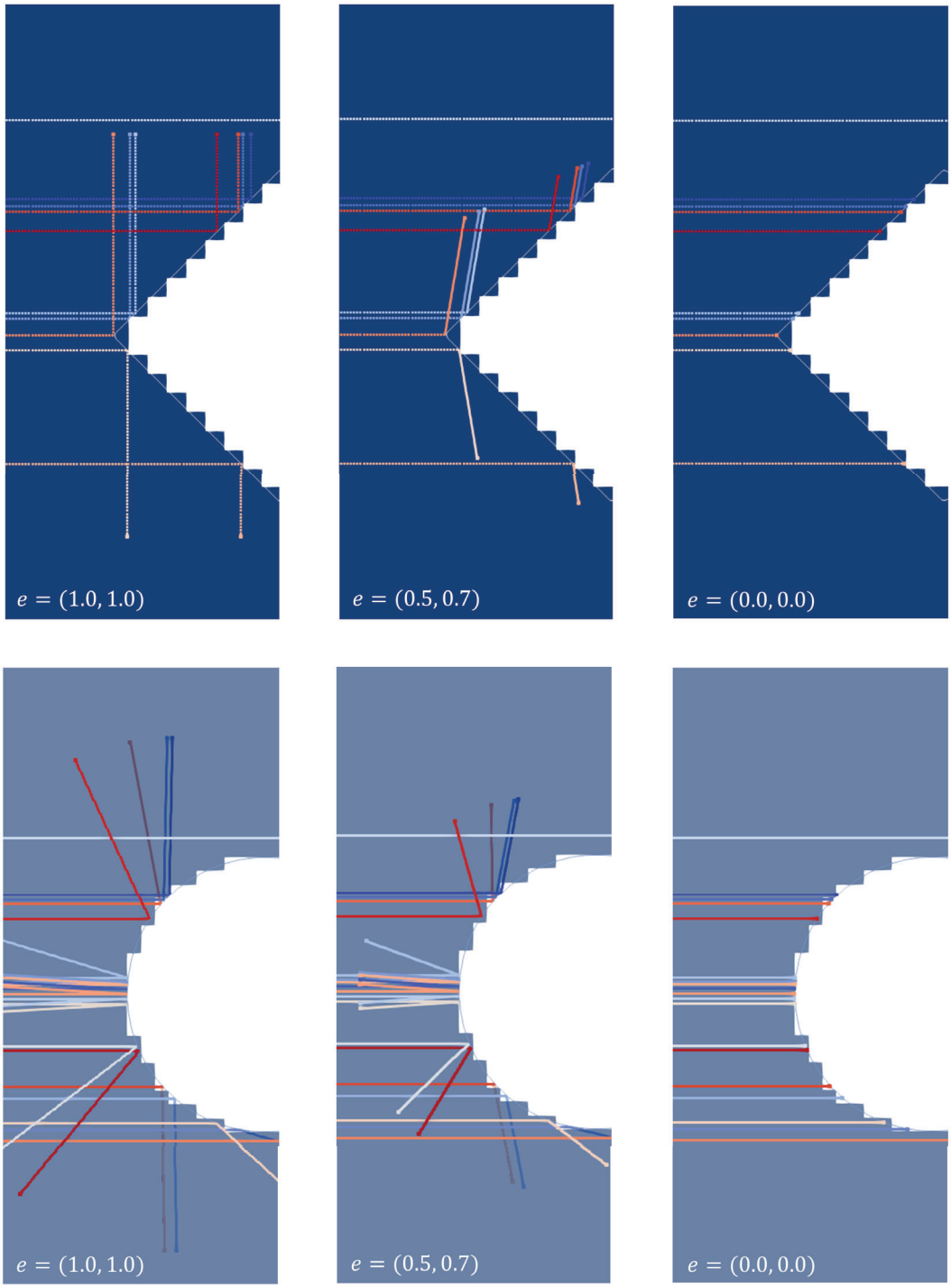

Figure 4.8: Pathlines of particles undergoing collision boundary conditions with various coefficients of restitution e for a cube and sphere. The surface of the object is shown by a thin white line, fully solid voxels are shown in white, fluid voxels in blue. Particles may be reflected at an angle outside of the shown x-y plane, which can lead to graying pathline colors due to the opacity of the plane.

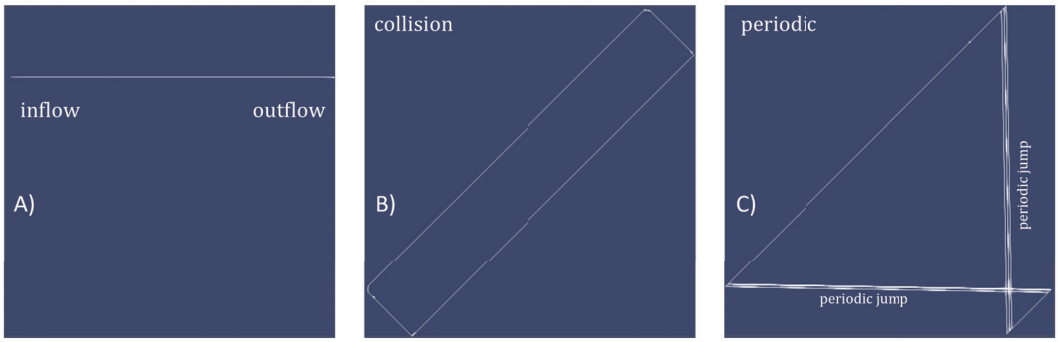

Figure 4.9: Testing of the domain wall boundary conditions visualized on a 2D view of the domain. Pathlines of the particles are shown in white, gray boxes show a 2D view of the domain.

boundary conditions, B) collision boundary conditions (here for $e = (1,1)$) and C) periodic boundary conditions. The results are shown in Figure 4.9. Inflow and outflow boundary conditions behave as expected, specifically also with particles being deactivated directly on the domain boundary.

For correct implementations of (elastic) collision and periodic boundary conditions, we can expect particles to continuously loop along the same pathlines. This is clearly the case. The only lines that are not perfectly overlapping are the corners of the trajectory in case B) (where collision occurs) and the "periodic jump lines" in case C) (where periodic boundary conditions occur), where several lines become visible. These lines are not a real part of the trajectory: The postprocessing tool draws straight lines between the timestep data, which are not necessarily in exactly the same position due to timestepping, even with the actual trajectory always following the same path.

4.5 Runtimes

As a part of our tests, we do a runtime analysis. We consider only the total computation time without pre-processing or initialization, and with only a single (final) output of data. We use a Poiseuille case similar to Section 4.1 with a fixed physical simulation time $t_{\text{sim}} \approx 1\,\text{s}$ and Stokes drag, and compare one-way and two-way coupled simulations for varying mesh sizes and numbers of particles.

The results relative to the same simulation without particles are shown in Figure 4.10. We consider between 10 and 10,000 particles. One-way coupled simulations show a small increase in runtime of around 10 %, which remains relatively stable and increases only slightly with the number of particles.

In comparison, two-way coupled simulations show large runtimes that are very dependent on the number of particles. At higher N_p, the complexity approaches $\mathcal{O}(N_p)$. Luckily, this is due to a very distinct bottleneck in the code: As described in Section 3.4.2, our current imple-

mentation of the back-coupling includes a loop where every voxels iterates over the particle force arrays to sum up the particle forces while avoiding race conditions. By avoiding this, the runtime complexity could be reduced far below $\mathcal{O}(N_p)$. Possible solutions are discussed in Section 5.4.

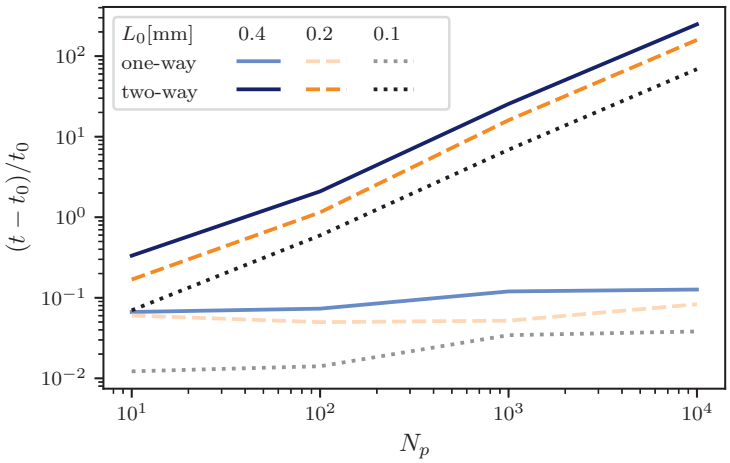

Figure 4.10: Computation times t for one-way and two-way coupled flow simulations relative to the computation time without particles t_0.

4.6 Memory requirements

For $N_p \leq 10,000$, the range in which we have worked so far, the memory requirements of our particles on both CPU and GPU are negligible. We therefore test memory requirements for simulations with one million particles by looking at the difference in peak allocated memory between simulations with or without particles for one-way and two-way coupled simulations with different mesh sizes.

The results are shown in Figure 4.11. As we can see, CPU memory requirements are independent of the coupling mode. For the GPU, particle memory requirements are shown to be independent of mesh size, as they depend mostly on the number of particles. In both one-way and two-way coupled simulations, there are 10 basic N_p-length particle arrays (positions, velocities, accelerations, status). With floats and integers taking up 4 bytes each, this accounts for 0.04 GB of additional memory, which corresponds very well to our test results.

Two-way coupled simulations have five additional particle force arrays of size $N_p \cdot 8$, implying an increase in required GPU memory by 400%, or, in our case, 0.16 GB. This also corresponds to the values displayed in Figure 4.11.

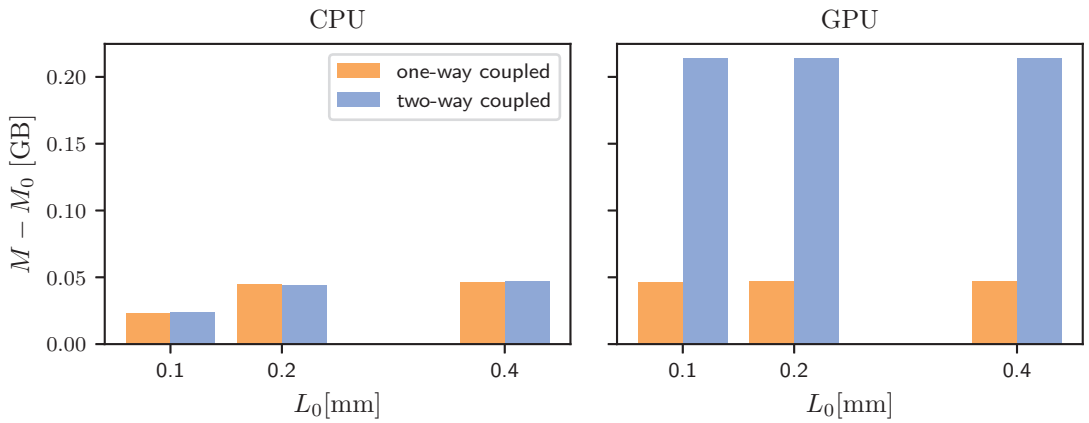

Figure 4.11: The difference in peak allocated CPU and GPU memory between simulations with (M) and without (M_0) particles for different grid sizes and one million particles.

5 Conclusion and Outlook

In this work, we have implemented, tested, and validated a framework for point-particle tracking in ultraFluidX® for one-way and two-way coupled simulations. Particle properties and release parameters can be chosen separately by user input for an arbitrary number of particle zones. The three main steps of the particle simulation - coupling from fluid to particle, back-coupling from particle to fluid, and particle movement - have been implemented in three separate GPU kernels. Several options for domain and object boundary conditions are included in the code, based on local planar approximations of the surface using the surface normals. We have tested all parts of our implementation and shown that they work as expected, which now provides us with most of the basic components necessary for using point-particle tracking. To make our work fully functional and utilize the full capabilities of ultraFluidX®, we consider four main extensions and improvements to be required:

- Currently, we only support uniform grids. A strategy for coupling to fluid simulations run on multi-resolution grids has to be chosen and implemented.
- Similarly, particle tracking simulations can currently only be run on a single GPU. Support for multi-GPU particle simulations needs to be implemented.
- The current code approximates surfaces locally as planes. A more sophisticated method for surface reconstruction is required to be able to handle complex geometries.
- Our testing revealed that the runtimes of two-way coupled flow simulations scale linearly with the number of particles due to voxels looping over particle arrays to sum up the particle forces pertaining to them. Here, a change in the implementation is required to allow fast simulation even for large numbers of particles.

We will go into detail on these four points, propose additional extensions and improvements to implementation and modeling, and give a general outlook into future applications of the framework.

5.1 Multi-resolution grids

Grid refinement for LBM with Cartesian grids is easily done by dividing mesh sizes in each dimension by a factor 2^n, where n is the refinement level. Time steps also follow this rule. At interfaces between areas of different refinement levels, the fluid information exists at both levels.

To couple this with the particle simulation, three options exist:

- Particles could couple to a uniform grid corresponding to a refinement level of choice. This might be very inaccurate or computationally expensive depending on the chosen refinement level.

- Particles could also couple directly to the fully refined grid using either the coarsest timestep or the local timestep. In the latter case, it would become essential to introduce a cutoff velocity based on the grid size instead of the current sub-timestepping method. Otherwise, particles would be free to jump between several refinement levels in one timestep.

- As the third option, a mix of the two might be possible, i.e. adapting to the fluid refinement levels up to a chosen 'cutoff' refinement level.

It should be kept in mind that there are three important points of contact between the particle and the fluid simulations: The velocity interpolation, the application of the back coupling force and, notably, the access to the normal information for boundary conditions. No matter which case is chosen, all the necessary information needs to be made available in the appropriate timesteps or even subtimesteps.

For the multi-grid particle data structure, it is likely best to keep only a single set of particle arrays, and (de-)activate particle threads based on the current refinement level they are in instead of moving particles between dedicated refinement level arrays. This would require an additional array containing the refinement level information for each particle.

5.2 Multi-GPU simulations

Multi-GPU simulations will likely require moving particles between GPUs. For such low numbers of particles as we are currently dealing with (up to 10,000), the easiest solution would likely be to reserve the necessary memory for all particles on each GPU, and activate/deactivate particles on the GPUs based on their location. The more difficult question is, again, the treatment of interfaces. Here too, a cutoff velocity will be required to ensure that particles do not move too far beyond a GPU boundary within one timestep.

5.3 Surface reconstruction

For surface reconstruction (or approximation), several methods exist, as this is a common problem across many domains, from mathematics to computer graphics. As input data, we essentially have a point cloud of surface points with normal information. Considerations need to be made as to whether surfaces are to be approximated locally in the proximity of each particle for each timestep or even subtimestep (which could lead to large runtimes), or whether a global surface approximation is performed once (which might be memory-intensive). Some first ideas on methods that could be considered include B-splines [30] and Moving Least Squares [31]. Special consideration needs to be given to very thin objects

(baffles) to avoid particles glitching through walls due to a lack in accuracy of the approximated surface. Likely, a trade-off will also have to be made between accurately approximating smooth surfaces and sharp edges.

5.4 Back-coupling computation time

In order to reduce the runtime, we need to replace the loop over which each voxel iterates to sum up the particle forces pertaining to itself with a different method. The priority here is to avoid race conditions, as several different particles might be found close enough to a voxel to contribute to its drag body force. A first idea on how to accomplish this would be to use atomics[1], which are operations that cannot be interrupted by another process, meaning that several particle threads would be able to write to the same voxel without it leading to undefined behavior.

5.5 Additional extensions and improvements

There are many additional improvements and extensions that could improve the performance, accuracy and scope of this work in the future. We are listing some of these below for future consideration:

- Implementation
  - Make use of shared memory to improve performance of particle kernels.
  - Consider alternative time-stepping schemes.
- Initialization and user interface
  - Allow velocity initialization from Boltzmann distribution.
  - Allow random particle sizes.
- Modeling improvements
  - Introduce alternative interpolation methods for the velocity.
  - Introduce alternative methods for distributing back-coupling force to fluid.
  - Validate alternative drag coefficients.
  - Consider alternative time integration methods.
  - Introduce a model for surface roughness.
  - Introduce coupling for different physics beyond drag, such as virtual mass or pressure gradient forces, thermal coupling, or turbulence modulation.
  - Consider the interaction of particles with moving walls.

[1] See e.g. `https://en.cppreference.com/w/cpp/atomic/atomic/fetch_add`.

5.6 Application outlook

As a more general outlook, this work provides a good basis for simulating many different particle-laden flow problems for real-world applications. At its current status, the framework could already be used for simulating changed flow behavior due to the presence of particles or impact modeling for abrasion studies. For more complex problems, such as snow deposition or clogging, four-way coupled simulations are required, which can be achieved through coupling the existing code with a separate particle simulation software. Although this would require many fundamental changes to the existing code, it might also simplify some aspects of the implementation, such as particle boundary conditions, which the dedicated particle solver would likely take care of.

Bibliography

[1] Sergio R. Idelsohn et al. “A multiscale approach for the study of particle-laden flows using a continuous model”. In: *Computer Methods in Applied Mechanics and Engineering* 401 (Nov. 2022), p. 115174. DOI: 10.1016/j.cma.2022.115174. URL: https://doi.org/10.1016/j.cma.2022.115174.

[2] Robin Trunk et al. “Inertial dilute particulate fluid flow simulations with an Euler–Euler lattice Boltzmann method”. In: *Journal of Computational Science* 17 (Nov. 2016), pp. 438–445. DOI: 10.1016/j.jocs.2016.03.013. URL: https://doi.org/10.1016/j.jocs.2016.03.013.

[3] Thomas Henn et al. “Parallel dilute particulate flow simulations in the human nasal cavity”. In: *Computers & Fluids* 124 (Jan. 2016), pp. 197–207. DOI: 10.1016/j.compfluid.2015.08.002. URL: https://doi.org/10.1016/j.compfluid.2015.08.002.

[4] Václav Heidler et al. “Eulerian–Lagrangian and Eulerian–Eulerian approaches for the simulation of particle-laden free surface flows using the lattice Boltzmann method”. In: *Journal of Computational and Applied Mathematics* 398 (Dec. 2021), p. 113672. DOI: 10.1016/j.cam.2021.113672. URL: https://doi.org/10.1016/j.cam.2021.113672.

[5] Chun Liu. “Principles and Implementation of DEM”. In: *Matrix Discrete Element Analysis of Geological and Geotechnical Engineering*. Singapore: Springer Singapore, 2021, pp. 1–26. ISBN: 978-981-33-4524-9. DOI: 10.1007/978-981-33-4524-9_1. URL: https://doi.org/10.1007/978-981-33-4524-9_1.

[6] Peter A. Cundall and Otto D. L. Strack. “A discrete numerical model for granular assemblies”. In: *Geotechnique* 29 (1979), pp. 47–65. URL: https://api.semanticscholar.org/CorpusID:128484753.

[7] B. Kravets et al. “Comparison of particle-resolved DNS (PR-DNS) and non-resolved DEM/CFD simulations of flow through homogenous ensembles of fixed spherical and non-spherical particles”. In: *Advanced Powder Technology* 32.4 (Apr. 2021), pp. 1170–1195. DOI: 10.1016/j.apt.2021.02.016. URL: https://doi.org/10.1016/j.apt.2021.02.016.

[8] M. Dietzel, M. Ernst, and M. Sommerfeld. “Application of the Lattice-Boltzmann Method for Particle-laden Flows: Point-particles and Fully Resolved Particles”. In: *Flow, Turbulence and Combustion* 97.2 (Jan. 2016), pp. 539–570. DOI: 10.1007/s10494-015-9698-x. URL: https://doi.org/10.1007/s10494-015-9698-x.

[9] Timm Krüger et al. *The Lattice Boltzmann Method*. Springer International Publishing, 2017. DOI: 10.1007/978-3-319-44649-3. URL: https://doi.org/10.1007/978-3-319-44649-3.

[10] Amir Banari et al. "The simulation of turbulent particle-laden channel flow by the Lattice Boltzmann method". In: *International Journal for Numerical Methods in Fluids* 79.10 (June 2015), pp. 491–513. DOI: 10.1002/fld.4058. URL: https://doi.org/10.1002/fld.4058.

[11] L. D. Landau and E. M. Lifshitz. *Fluid Mechanics, Second Edition: Volume 6 (Course of Theoretical Physics)*. 2nd ed. Course of theoretical physics / by L. D. Landau and E. M. Lifshitz, Vol. 6. Butterworth-Heinemann, Jan. 1987. ISBN: 0750627670. URL: http://www.worldcat.org/isbn/0750627670.

[12] D. Arumuga Perumal and Anoop K. Dass. "A Review on the development of lattice Boltzmann computation of macro fluid flows and heat transfer". In: *Alexandria Engineering Journal* 54.4 (2015), pp. 955–971. ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2015.07.015. URL: https://www.sciencedirect.com/science/article/pii/S1110016815001362.

[13] P. L. Bhatnagar, E. P. Gross, and M. Krook. "A Model for Collision Processes in Gases. I. Small Amplitude Processes in Charged and Neutral One-Component Systems". In: *Physical Review* 94.3 (May 1954), pp. 511–525. DOI: 10.1103/physrev.94.511. URL: https://doi.org/10.1103/physrev.94.511.

[14] Dominique d'Humières et al. "Multiple-Relaxation-Time Lattice Boltzmann Models in Three Dimensions". In: *Philosophical Transactions: Mathematical, Physical and Engineering Sciences* 360.1792 (2002), pp. 437–451. ISSN: 1364503X. URL: http://www.jstor.org/stable/3066323 (visited on 09/13/2023).

[15] Martin Geier, Andrea Pasquali, and Martin Schönherr. "Parametrization of the cumulant lattice Boltzmann method for fourth order accurate diffusion part I: Derivation and validation". In: *Journal of Computational Physics* 348 (Nov. 2017), pp. 862–888. DOI: 10.1016/j.jcp.2017.05.040. URL: https://doi.org/10.1016/j.jcp.2017.05.040.

[16] A. N. Kolmogorov. "A refinement of previous hypotheses concerning the local structure of turbulence in a viscous incompressible fluid at high Reynolds number". In: *Journal of Fluid Mechanics* 13.1 (May 1962), pp. 82–85. DOI: 10.1017/s0022112062000518. URL: https://doi.org/10.1017/s0022112062000518.

[17] Martin R. Maxey and James J. Riley. "Equation of motion for a small rigid sphere in a nonuniform flow". In: *The Physics of Fluids* 26.4 (Apr. 1983), pp. 883–889. DOI: 10.1063/1.864230. URL: https://doi.org/10.1063/1.864230.

[18] Sumudu S. Karunarathne and Lars-André Tokheim. "Comparison of the influence of drag models in CFD simulation of particle mixing and segregation in a rotating cylinder". In: *Linköping Electronic Conference Proceedings*. Linköping University Electronic Press, Sept. 2017. DOI: 10.3384/ecp17138151. URL: https://doi.org/10.3384/ecp17138151.

[19] J. M. Buick and C. A. Greated. “Gravity in a lattice Boltzmann model”. In: *Physical Review E* 61.5 (May 2000), pp. 5307–5320. DOI: 10.1103/physreve.61.5307. URL: https://doi.org/10.1103/physreve.61.5307.

[20] ALEXANDER L Kupershtokh. “New method of incorporating a body force term into the lattice Boltzmann equation”. In: *Proceeding of the 5th International EHD Workshop*. 2004, pp. 241–246.

[21] Nathan A. Keane et al. “Effect of interpolation kernels and grid refinement on two way-coupled point-particle simulations”. In: *International Journal of Multiphase Flow* 166 (2023), p. 104517. ISSN: 0301-9322. DOI: https://doi.org/10.1016/j.ijmultiphaseflow.2023.104517. URL: https://www.sciencedirect.com/science/article/pii/S0301932223001386.

[22] Alan Weiser and Sergio E. Zarantonello. “A note on piecewise linear and multilinear table interpolation in many dimensions”. In: *Mathematics of Computation* 50.181 (1988), pp. 189–196. DOI: 10.1090/s0025-5718-1988-0917826-0. URL: https://doi.org/10.1090/s0025-5718-1988-0917826-0.

[23] Xiaoyi He and Li-Shi Luo. “Theory of the lattice Boltzmann method: From the Boltzmann equation to the lattice Boltzmann equation”. In: *Physical Review E* 56.6 (Dec. 1997), pp. 6811–6817. DOI: 10.1103/physreve.56.6811. URL: https://doi.org/10.1103/physreve.56.6811.

[24] T. Schwager, V. Becker, and T. Pöschel. “Coefficient of tangential restitution for viscoelastic spheres”. In: *The European Physical Journal E* 27.1 (Aug. 2008), pp. 107–114. DOI: 10.1140/epje/i2007-10356-3. URL: https://doi.org/10.1140/epje/i2007-10356-3.

[25] Xinchen Zhang et al. “The influence of the coefficient of restitution on flow regimes within horizontal particle-laden pipe flows”. In: *Physics of Fluids* 33.12 (Dec. 2021). DOI: 10.1063/5.0075440. URL: https://doi.org/10.1063/5.0075440.

[26] Robert L. Jackson, Itzhak Green, and Dan B. Marghitu. “Predicting the coefficient of restitution of impacting elastic-perfectly plastic spheres”. In: *Nonlinear Dynamics* 60.3 (Sept. 2009), pp. 217–229. DOI: 10.1007/s11071-009-9591-z. URL: https://doi.org/10.1007/s11071-009-9591-z.

[27] Emanuel Willert. *Stoßprobleme in Physik, Technik und Medizin*. Springer Berlin Heidelberg, 2020. DOI: 10.1007/978-3-662-60296-6. URL: https://doi.org/10.1007/978-3-662-60296-6.

[28] John Happel and Howard Brenner. *Low Reynolds number hydrodynamics*. Springer Netherlands, 1983. DOI: 10.1007/978-94-009-8352-6. URL: https://doi.org/10.1007/978-94-009-8352-6.

[29] Jaber Almedeij. “Drag coefficient of flow around a sphere: Matching asymptotically the wide trend”. In: *Powder Technology* 186.3 (Sept. 2008), pp. 218–223. DOI: 10.1016/j.powtec.2007.12.006. URL: https://doi.org/10.1016/j.powtec.2007.12.006.

[30] H. Pottmann, S. Leopoldseder, and M. Hofer. “Approximation with active B-spline curves and surfaces”. In: *10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings.* IEEE Comput. Soc. DOI: 10.1109/pccga.2002.1167835. URL: https://doi.org/10.1109/pccga.2002.1167835.

[31] Z.-Q. Cheng et al. *A Survey of Methods for Moving Least Squares Surfaces.* en. 2008. DOI: 10.2312/VG/VG-PBG08/009-023. URL: http://diglib.eg.org/handle/10.2312/VG.VG-PBG08.009-023.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Druck-Exemplaren überein.

Datum und Unterschrift:

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted hard copies.

Date and Signature: